#### COPYRIGHTS AND TRADEMARKS

This documentation is copyright © 2022 by Parallax Inc. By downloading or obtaining a printed copy of this documentation or software you agree that it is to be used with, or with products containing, the Parallax Propeller 2 P2X8C4M64P microcontroller. Any other uses are not permitted and may represent a violation of Parallax copyrights, legally punishable according to Federal copyright or intellectual property laws. Any duplication of this documentation for commercial uses is expressly prohibited by Parallax Inc. Duplication for educational use is permitted, subject to the following Conditions of Duplication: Parallax Inc. grants the user a conditional right to download, duplicate, and distribute this text without Parallax's permission. This right is based on the following conditions: the text, or any portion thereof, may not be duplicated for commercial use; it may be duplicated only for educational purposes when used solely in conjunction with Parallax products, and the user may recover from the student only the cost of duplication.

Parallax, Propeller Spin, and the Parallax logos are trademarks of Parallax Inc. If you decide to use any trademarks of Parallax Inc. on your web page or in printed material, you must state that (trademark) is a trademark of Parallax Inc.” upon the first appearance of the trademark name in each printed document or web page. Other brand and product names herein are trademarks or registered trademarks of their respective holders.

#### DISCLAIMER OF LIABILITY

Parallax, Inc. makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Parallax, Inc. assume any liability arising out of the application or use of any product, and specifically disclaims any and all liability, including without limitation consequential or incidental damages even if Parallax, Inc. has been advised of the possibility of such damages.

#### INTERNET DISCUSSION LISTS

We maintain active web-based discussion forums for people interested in Parallax Propeller products, at [forums.parallax.com](https://forums.parallax.com/).

#### ERRATA

While great effort is made to assure the accuracy of our texts, errors may still exist. If you find an error, please let us know by commenting/suggesting on live documentation, or by sending an email to editor@parallax.com. We continually strive to improve all of our educational materials and documentation, and frequently revise our texts. Occasionally, an errata sheet with a list of known errors and corrections for a given text will be posted to our website, [www.parallax.com](https://www.parallax.com). Please check the individual product page’s free downloads for an errata file.

#### SUPPORTED HARDWARE AND FIRMWARE

This manual is valid with the following hardware and firmware versions:

| Hardware | Firmware |
| --- | --- |
| P2X8C4M64P | Rev B/C |

#### CREDITS

Authorship: Jeff Martin • Format & Editing: Stephanie Lindsay • Technical Graphics: Michael Mulholland

With many thanks to everyone in the Propeller Community and staff at Parallax Inc.

# 

# TABLE OF CONTENTS

[**PREFACE**](#_zyeu5q2a3jl) **10**

[**CONVENTIONS**](#_12fh0v8mc8fz) **10**

[**ASSEMBLY LANGUAGE REFERENCE**](#_9qqqmspnmzjp) **11**

[Multi-Long ADD/SUB/CMP Operations](#_uxtmxdreoq2s) 11

[Adding Two Multi-Long Values](#_qt1j03hxivpt) 11

[Subtracting Two Multi-Long Values](#_19e0pvyp0h0x) 12

[Comparing Two Multi-Long Values](#_kjzu1wz1qct1) 14

[Categorical Listing Of Propeller 2 Assembly Language](#_je0gsyt7f7wn) 15

[Directives](#_dq838bc22zwt) 15

[Conditions](#_ki1kkm7brm59) 15

[Effects](#_4y0ciq9th2g) 16

[Flag Modification](#_cdvnblowhyuw) 17

[Augmentation](#_6gqxl5an913i) 18

[Indirection](#_si46t1tj3ovj) 18

[Configuration](#_38hltgjd7fk) 18

[Cog Control](#_131dn518do28) 18

[Process Control](#_ctx5xge61csp) 18

[Flow Control](#_v82tlx47lbbn) 19

[Lookup Table (LUT) Memory Access](#_6rvupx6ksenl) 19

[Hub Memory Access](#_w13lxpiq71b6) 20

[Streamer](#_k4inekhhune) 20

[I/O Pins](#_21gxb0rq4onw) 20

[Math](#_xnk9pb8j5sof) 21

[Timing](#_rie544cg2hir) 23

[Event Handling](#_6ehq03n22voi) 23

[Interrupts](#_40bj2qk54rdh) 25

[Bit Operations](#_1ykf5b4dvpz7) 25

[Color Manipulation](#_oahsp8cfzdfl) 26

[Miscellaneous](#_1gzr437m04f) 27

[Registers](#_87ivzhp7ysag) 27

[Constants](#_owsd5q74b157) 27

[Operators](#_5hw3n9zgfb5c) 28

[Assembly Language Elements](#_uap4lnaifkwp) 29

[Term Definitions](#_gu75jpqwluhn) 29

[Opcode Tables](#_7jc5g15fikk4) 30

[Propeller Assembly Instruction Master Table](#_gs1ks0eq1j8v) 31

[ABS](#_lrvakeitwivz) 31

[ADD](#_qhneb9rqvxbb) 32

[ADDCT1/2/3](#_u2xzjautl27l) 32

[ADDPIX](#_n2at4numrp4) 33

[ADDS](#_566nu7hwsj0f) 33

[ADDSX](#_23s34xc9b4y1) 34

[ADDX](#_vhdbx9ift4rv) 34

[AKPIN](#_putbdsx21wr) 35

[ALIGNL](#_va7a6kt7oiii) 35

[ALIGNW](#_6y0y6yvl7r2z) 36

[ALLOWI](#_jv88q3ni4hqo) 38

[ALTB](#_r6knb6xhbyzp) 38

[ALTD](#_6iq3v0fpyc3a) 39

[ALTGB](#_q234901aifm8) 40

[ALTGN](#_asw5rsrcqtpn) 41

[ALTGW](#_hahk50n443um) 42

[ALTI](#_ttudk0n3xdym) 43

[ALTR](#_3cn9gj4kxitq) 45

[ALTS](#_f2hcvwgutfa0) 46

[ALTSB](#_2jkx93n2978v) 46

[ALTSN](#_o72f3jsaebvh) 47

[ALTSW](#_90jw6j8128ot) 48

[AND / ANDN](#_ry94d81q3emx) 49

[AUGD](#_qxo6681ulhaj) 50

[AUGS](#_b6omkiqrkged) 51

[BITC / BITNC](#_3gamq9k207in) 52

[BITH / BITL](#_pbf6oqcly2oa) 52

[BITNOT](#_hgixqtr6y6sz) 53

[BITRND](#_v3oe9q9od6uf) 54

[BITZ / BITNZ](#_3961stdwxxh2) 54

[BMASK](#_nui8bnpnjowd) 55

[BRK](#_d8ho6miavbop) 55

[CALL](#_lrwvkfn726tn) 56

[CALLA / CALLB](#_9bxg6hr5ntaa) 57

[CALLD](#_y34ndvfdx6a) 58

[CALLPA / CALLPB](#_11plynh8xrpt) 59

[CMP](#_jw7rhjdisro0) 59

[CMPM](#_xxivkrt1rtt) 60

[CMPR](#_6j1a3848ozh3) 60

[CMPS](#_nd6caax9v0zg) 61

[CMPSUB](#_tvdyh2esm2qw) 61

[CMPSX](#_ccc7sdk782ug) 62

[CMPX](#_3w81cwnriog1) 62

[COGATN](#_wx29grvrexs) 63

[COGBRK](#_okd9uhh52aur) 64

[COGID](#_uy4mwi8z5ww4) 64

[COGINIT](#_5knlosjgh2sz) 64

[COGSTOP](#_eg5nftcbzayn) 67

[Conditions ( IF\_x )](#_ofohv8f4qicc) 67

[DECMOD](#_jjzqxknhgv63) 69

[DECOD](#_vp3efvde1vn6) 69

[DIRC / DIRNC](#_q5818ady3ccx) 70

[DIRH / DIRL](#_adfu0crzr5tx) 71

[DIRNOT](#_8cdi9p9b7vgm) 72

[DIRRND](#_5p7siddmqpeb) 72

[DIRZ / DIRNZ](#_eaobslhzfr83) 73

[DJF / DJNF](#_jdehvyhk3dz3) 74

[DJZ / DJNZ](#_dm4flu2kbbs) 74

[DRVC / DRVNC](#_odv0wkzdsytw) 75

[DRVH / DRVL](#_cuc8iv9j9262) 76

[DRVNOT](#_ra3uexs4cuzi) 77

[DRVRND](#_cfks3rynv54x) 77

[DRVZ / DRVNZ](#_g20ikbjddn3e) 78

[ENCOD](#_brfhxn6t7t41) 79

[Effects ( WC, WZ, WCZ, ANDC, etc. )](#_2qhfue5jrv4x) 80

[FGE](#_4qzq1q9ohttb) 81

[FGES](#_6pb8tmuj8hrq) 81

[FLE](#_kjr0bke2otvv) 82

[FLES](#_oo1zbqj5xudg) 82

[FLTC / FLTNC](#_1lkdbkm563zm) 83

[FLTH / FLTL](#_rg1q051ww7h1) 84

[FLTNOT](#_3llxk0tkveub) 85

[FLTRND](#_8wfub36x4ss6) 85

[FLTZ / FLTNZ](#_enfuevkmlntw) 86

[GETBYTE](#_t84l3o6tozfm) 87

[GETNIB](#_8yjp9q7d9ncx) 88

[GETWORD](#_ppayepmbzjsl) 88

[IF\_x (Conditions)](#_9k1omnbvvbdl) 89

[IJZ / IJNZ](#_q67no8uizffw) 89

[INCMOD](#_ln0kx81xiaak) 89

[JATN / JNATN](#_yfon3snogp62) 90

[JCT1/2/3 / JNCT1/2/3](#_fryvonwfped0) 91

[JFBW / JNFBW](#_miks298accr1) 91

[JINT / JNINT](#_wzi7hl4t1ffr) 92

[JPAT / JNPAT](#_dpv2q3yh9cay) 93

[JQMT / JNQMT](#_fik5o0ebifr2) 93

[JSE1/2/3/4 / JNSE1/2/3/4](#_nv0zohzajupf) 94

[JXFI / JNXFI](#_k5iotkp9d869) 95

[JXMT / JNXMT](#_izeyc7za5235) 95

[JXRL / JNXRL](#_43ntogxc32ez) 96

[JXRO / JNXRO](#_gut56vubumxk) 96

[MODC / MODZ / MODCZ](#_m3xduhu6qlur) 97

[MOV](#_s858lrxz52tn) 99

[MUL](#_wv511llidyd6) 99

[MULS](#_9awkxilhgl30) 100

[MUXC / MUXNC](#_hujma23idlak) 100

[MUXNIBS](#_so9xp16tbilq) 101

[MUXNITS](#_xa6h00k6yh91) 101

[MUXQ](#_gg86hluj11kt) 101

[MUXZ / MUXNZ](#_fsy0qkuiqsd6) 102

[NEG](#_ied83z8c60zq) 102

[NEGC](#_jlhwuat7bnkq) 103

[NEGNC](#_323cveis97t7) 103

[NEGNZ](#_z1ulw37qnpt4) 104

[NEGZ](#_5vu7vu1xc9tm) 105

[NOP](#_gdk4kvxdog0t) 105

[NOT](#_7x0wsorjlaru) 105

[OR](#_49iljyigagii) 106

[ONES](#_1ezqugv82636) 106

[OUTC / OUTNC](#_73z1ljpedf7w) 107

[OUTH / OUTL](#_40vs5rlpgy9a) 108

[OUTNOT](#_mll0nu2hed2z) 108

[OUTRND](#_rxjj8bmeue2f) 109

[OUTZ / OUTNZ](#_16ycek7kzbz5) 110

[POLLATN](#_ref43okqtzzr) 111

[POLLCT1/2/3](#_dgitqmcw0hm) 111

[POLLFBW](#_86ta1z2g86ao) 112

[POLLINT](#_m2gclmhbm44b) 112

[POLLPAT](#_2sqdtda5rj0e) 113

[POLLQMT](#_ge1n23vwgw41) 113

[POLLSE1/2/3/4](#_n3mua6rjfhxs) 114

[POLLXFI](#_r7m2wcrdq00o) 114

[POLLXMT](#_mg19wcqd1pjt) 115

[POLLXRL](#_63dai97cdrvc) 115

[POLLXRO](#_n123uusqspjd) 116

[RCL](#_75xnwhklnjzs) 116

[RCR](#_fbag5uwbvmwv) 117

[RCZL](#_bl2w5tdjasam) 117

[RCZR](#_efvpakcsdenq) 118

[Registers](#_6ift2cu57w3u) 118

[REV](#_8x2v3buaskxv) 119

[ROL](#_e76w2o7qsgpp) 119

[ROLBYTE](#_9ruumu1gk11b) 119

[ROLNIB](#_5imtv7yml80t) 120

[ROLWORD](#_lhi32k9v1cp3) 121

[ROR](#_urnydzozfyv8) 121

[SAL](#_dflmaw9re4j) 122

[SAR](#_er4hl0kfn1wn) 122

[SCA](#_72dwpag2ycaq) 123

[SCAS](#_eh4usclguwow) 123

[SETBYTE](#_f8filcyhyg4l) 124

[SETD](#_jw7izhflytke) 124

[SETNIB](#_dtkktvehwfob) 125

[SETR](#_brza01jtowzy) 126

[SETS](#_94hz48griij7) 126

[SETWORD](#_9gwjt0ijlutf) 127

[SHL](#_cg4e6rxaezp) 128

[SHR](#_y0jm929edm64) 128

[SIGNX](#_923rdwqvf980) 129

[STALLI](#_4b448hiwgtyc) 129

[SUB](#_n28wi9jpddia) 130

[SUBR](#_ss3vapy9cq1h) 130

[SUBS](#_phb5pufpspb9) 131

[SUBSX](#_fwvdz7pvnpz3) 131

[SUBX](#_4wyzstsl5slq) 132

[SUMC / SUMNC](#_8370tvf23i61) 132

[SUMZ / SUMNZ](#_5u8few7uaee4) 133

[TEST](#_6m4jc27v9rub) 134

[TESTB / TESTBN](#_sjy72z2klc7g) 134

[TESTN](#_q7r5myfbnfk5) 135

[TESTP / TESTPN](#_d2afnf6ec911) 136

[TJF / TJNF](#_1rck8xngsheq) 136

[TJS / TJNS](#_rm5pk4tx7cv6) 137

[TJV](#_o9pr7ygsoyes) 138

[TJZ / TJNZ](#_nz9y81fecbaa) 138

[WAITATN](#_sb5186rnvudg) 139

[WAITCT1/2/3](#_u335ct3ffsan) 139

[WAITFBW](#_qoevcy1s6y48) 140

[WAITINT](#_rqj7wu7kf26o) 141

[WAITPAT](#_5i1v8ffdcpk2) 141

[WAITSE1/2/3/4](#_43d9knl3whqm) 142

[WAITXFI](#_zdbnefgvlfr5) 143

[WAITXMT](#_1ird9xvgzp82) 143

[WAITXRL](#_nesp1ncmao0z) 144

[WAITXRO](#_y7dswdrnedgw) 145

[WRC / WRNC](#_70my8h1n5l99) 145

[WRZ / WRNZ](#_4rwgnzl2tfbc) 146

[XOR](#_ci31vbchcwxx) 146

[ZEROX](#_9rn21a3cv769) 146

[**PROPELLER 2 ASSEMBLY LANGUAGE (PASM2) IN BRIEF**](#_em2rds3ngnl0) **148**

[**PROPELLER 2 RESERVED WORDS (SPIN2 + PASM2)**](#_76srt37q0r25) **160**

[**CHANGE LOG**](#_swmvo3n67qaa) **162**

[**PARALLAX INCORPORATED**](#_cenlv5sy0ptb) **162**

# 

# PREFACE

This guide is an in-depth description of the Propeller 2 PASM2 language (Propeller 2 Assembly Language). It serves as a full reference to the language and techniques for its use.

For additional documentation and resources, including programming tools, visit [www.parallax.com/P2](http://www.parallax.com/P2). The latest version of this guide, along with links to a commentable Google Doc version, are available from the Documentation section. In addition, there are links to other references for the Propeller 2 and its Spin2 language, which may include commentable Google Docs.

# CONVENTIONS

* % - indicates a binary number (containing the digits 0 and 1, and underscore "\_" characters)
  + ex: %0101 and %11000111
* $ - indicates either a hexadecimal number (containing the digits 0–9, A–F, and underscore "\_" characters) or the current address (the address of the current instruction)
  + ex: $2AF and $0D816
  + ex: $ (in an instruction's destination or source field)
* \_ - (underscore) is a visual separator in numbers and in certain symbols. Occasionally they are used as the leading or trailing character of a symbol to make it unique while keeping it similar to a same-named symbol that may already exist, or may be used in a syntax description to separate words of an all-lowercase phrase meant to be replaced by the reader when typing. In numbers (i.e. decimal, hexadecimal, and binary values) they are group indicators that may separate natural boundaries (like groups of 3 digits in decimal or 8 bits in binary) or may separate context-specific fields of a value (like smart mode bits and drive level bits in an I/O pin configuration value)
  + ex: %00111010\_01011101 and $4C1F\_0D816 and 2\_328\_476
  + ex: <low\_byte> and IF\_C\_AND\_Z and ZSTR\_
* x - indicates a group of symbols where the x-part can vary
  + ex: INx means both INA and INB
  + ex: RDxxxx / WRxxx means RDBYTE, RDWORD, etc. and WRBYTE, WRWORD, etc.

-- or --

x - indicates a *don't care* bit in a binary number (a bit that can be 0 or 1 without affecting the execution context nor the explanation)

* + ex: %x\_1\_xxx0 and %xxxx10
* ALL CAPS or ALL\_CAPS - indicates the item has special meaning, such as a reference to a label in a diagram, or a component in the Propeller 2, or is a predefined symbol (reserved word) such as a command, condition, register name, etc.
* <all\_lowercase> - indicates an item meant to be completely replaced by the user when typing code; the phrase in brackets "<>" describes the intent of the specific item.
* {item} or { item1 | item2 | item3 } - denotes an optional item or items. Either zero or one of the items can exist at the noted place; don't type the brackets "{}" or pipe symbol "|".
* item1 | item2 | item3 - denotes a single item, chosen from the list, that must be at the noted place; don't type the pipe symbol "|".

# ASSEMBLY LANGUAGE REFERENCE

## Multi-Long ADD/SUB/CMP Operations

Many useful integer quantities fit within 32 bits of space (0 to 4,294,967,295 or -2,147,483,648 to +2,147,483,647). This wide range means the majority of integer add/subtract/compare operations are achieved in just one step using a single 32-bit signed or unsigned math instruction; a *single-long* operation. However, if any such operation will use or create values larger than 32 bits, multiple math instructions must be chained together to form a *multi-long* "extended" operation; two instructions for 64-bits, three instructions for 96-bits, etc.

### Adding Two Multi-Long Values

To add two 64-bit values, use an unsigned [ADD](#_qhneb9rqvxbb) instruction (regardless of the values' unsigned or signed nature) followed by either an unsigned-extended [ADDX](#_vhdbx9ift4rv) or signed-extended [ADDSX](#_23s34xc9b4y1) instruction, for unsigned or signed numbers, respectively. *If the two values are larger than 64 bits in size (regardless of their signed/unsigned format) insert one ADDX instruction per extra 32-bits in-between the first and last instruction.*

Make sure to also use the WC or WCZ effect on the leading ADD and ADDX instructions so that the final ADDX or ADDSX instruction works properly.

#### 64-bit Unsigned Add

An unsigned double-long (64-bit) addition looks like this:

add XLow, YLow wcz 'Add low longs; save C (required) and Z (optional)

addx XHigh, YHigh wcz 'Add high longs; save C and Z (both optional)

Note: the optional flag effects are only required if the intent is to monitor for zero or an unsigned overflow.

After executing the above, the full double-long (64-bit) result is in the registers XHigh:XLow. If XHigh:XLow started out as $0000\_0000:FFFF\_FFFF (4,294,967,295) and YHigh:YLow was $0000\_0000:0000\_0001 (1), the result in XHigh:XLow would be $0000\_0001:0000\_0000 (4,294,967,296). This is demonstrated below.

Hexadecimal Decimal

(high) (low)

(XHigh:XLow) $0000\_0000:FFFF\_FFFF 4,294,967,295

+ (YHigh:YLow) + $0000\_0000:0000\_0001 + 1

─────────────────── ──────────────

= $0000\_0001:0000\_0000 = 4,294,967,296

An unsigned triple-long (96-bit) addition would look similar but with another ADDX instruction inserted between the ADD and ADDX instructions:

add XLow, YLow wcz 'Add low longs; save C (required) and Z (optional)

addx XMid, YMid wcz 'Add middle longs; save C (required) and Z (optional)

addx XHigh, YHigh wcz 'Add high longs; save C and Z (both optional)

During this multi-step operation, the Z flag always indicates if the result is turning out to be zero, but the C flag indicates unsigned carries until the final ADDX instruction, in which it indicates unsigned overflow (if any).

#### 64-bit Signed Add

In comparison to the above, a signed double-long (64-bit) addition uses an ADDSX instead of ADDX as the last instruction:

add XLow, YLow wcz 'Add low longs; save C (required) and Z (optional)

addsx XHigh, YHigh wcz 'Add high longs; save C and Z (both optional)

Note: the optional flag effects are only required if the intent is to monitor for zero or the final sign.

After executing the above, the full double-long (64-bit) result is in the registers XHigh:XLow. If XHigh:XLow started out as $0000\_0001:0000\_0000 (4,294,967,296) and YHigh:YLow was $FFFF\_FFFF:FFFF\_FFFF (-1), the result in XHigh:XLow would be $0000\_0000:FFFF\_FFFF (4,294,967,295). This is demonstrated below.

Hexadecimal Decimal

(high) (low)

(XHigh:XLow) $0000\_0001:0000\_0000 4,294,967,296

+ (YHigh:YLow) + $FFFF\_FFFF:FFFF\_FFFF + -1

─────────────────── ──────────────

= $0000\_0000:FFFF\_FFFF = 4,294,967,295

A signed triple-long (96-bit) addition is similar, but with an ADDX instruction inserted between the ADD and ADDSX instructions:

add XLow, YLow wcz 'Add low longs; save C (required) and Z (optional)

addx XMid, YMid wcz 'Add middle longs; save C (required) and Z (optional)

addsx XHigh, YHigh wcz 'Add high longs; save C and Z (both optional)

During this multi-step operation, the Z flag always indicates if the result is turning out to be zero, but the C flag indicates unsigned carries until the final instruction, ADDSX, in which it indicates the final result's sign.

### Subtracting Two Multi-Long Values

To subtract two 64-bit values, use an unsigned [SUB](#_q0s2vcp5e3i5) instruction (regardless of the values' unsigned or signed nature) followed by either an unsigned-extended [SUBX](#_4wyzstsl5slq) or signed-extended [SUBSX](#_fwvdz7pvnpz3) instruction, for unsigned or signed numbers, respectively. *If the two values are larger than 64 bits in size (regardless of their signed/unsigned format) insert one SUBX instruction per extra 32-bits in-between the first and last instruction.*

Make sure to also use the WC or WCZ effect on the leading SUB and SUBX instructions so that the final SUBX or SUBSX instruction works properly.

#### 64-bit Unsigned Subtraction

An unsigned double-long (64-bit) subtraction looks like this:

sub XLow, YLow wcz 'Subtract low longs; save C (required) and Z (optional)

subx XHigh, YHigh wcz 'Subtract high longs; save C and Z (both optional)

Note: the optional flag effects are only required if the intent is to monitor for zero or an unsigned underflow.

After executing the above, the full double-long (64-bit) result is in the registers XHigh:XLow. If XHigh:XLow started out as $0000\_0001:0000\_0000 (4,294,967,296) and YHigh:YLow was $0000\_0000:0000\_0001 (1), the result in XHigh:XLow would be $0000\_0000:FFFF\_FFFF (4,294,967,295). This is demonstrated below.

Hexadecimal Decimal

(high) (low)

(XHigh:XLow) $0000\_0001:0000\_0000 4,294,967,296

- (YHigh:YLow) - $0000\_0000:0000\_0001 - 1

─────────────────── ──────────────

= $0000\_0000:FFFF\_FFFF = 4,294,967,295

An unsigned triple-long (96-bit) subtraction is similar, but with another SUBX instruction inserted between the SUB and SUBX instructions:

sub XLow, YLow wcz 'Subtract low longs; save C (required) and Z (optional)

subx XMid, YMid wcz 'Subtract middle longs; save C (required) and Z (optional)

subx XHigh, YHigh wcz 'Subtract high longs; save C and Z (both optional)

During this multi-step operation, the Z flag always indicates if the result is turning out to be zero, but the C flag indicates unsigned borrows until the final SUBX instruction, in which it indicates unsigned underflow (if any).

#### 64-bit Signed Subtraction

In comparison to the above, a signed double-long (64-bit) subtraction uses a SUBSX instead of SUBX as the last instruction:

sub XLow, YLow wcz 'Subtract low longs; save C (required) and Z (optional)

subsx XHigh, YHigh wcz 'Subtract high longs; save C and Z (both optional)

Note: the optional flag effects are only required if the intent is to monitor for zero or the final sign.

After executing the above, the full double-long (64-bit) result is in the registers XHigh:XLow. If XHigh:XLow started out as $0000\_0000:0000\_0001 (1) and YHigh:YLow was $0000\_0000:0000\_0002 (2), the result in XHigh:XLow would be $FFFF\_FFFF:FFFF\_FFFF (-1). This is demonstrated below.

Hexadecimal Decimal

(high) (low)

(XHigh:XLow) $0000\_0000:0000\_0001 1

- (YHigh:YLow) - $0000\_0000:0000\_0002 - 2

─────────────────── ──────────────

= $FFFF\_FFFF:FFFF\_FFFF = -1

A signed triple-long (96-bit) subtraction would look similar but with a SUBX instruction inserted between the SUB and SUBSX instructions:

sub XLow, YLow wcz 'Subtract low longs; save C (required) and Z (optional)

subx XMid, YMid wcz 'Subtract middle longs; save C (required) and Z (optional)

subsx XHigh, YHigh wcz 'Subtract high longs; save C and Z (both optional)

During this multi-step operation, the Z flag always indicates if the result is turning out to be zero, but the C flag indicates unsigned borrows until the final instruction, SUBSX, in which it indicates the final result's sign.

### Comparing Two Multi-Long Values

The process of comparing multi-long values is similar to subtracting them. To compare two 64-bit values, use an unsigned [CMP](#_jw7rhjdisro0) instruction (regardless of their unsigned or signed form) followed by either an unsigned-extended [CMPX](#_3w81cwnriog1) or signed-extended [CMPSX](#_ccc7sdk782ug) instruction, for unsigned or signed numbers, respectively. *If the two values are larger than 64 bits in size (regardless of their signed/unsigned format) insert one CMPX instruction per extra 32-bits in-between the first and last instruction.*

Make sure to also use the WC or WCZ effect on the leading CMP and CMPX instructions so the final CMPX or CMPSX instruction works properly.

#### 64-bit Unsigned Compare

An unsigned double-long (64-bit) compare looks like this:

cmp XLow, YLow wcz 'Compare low longs; save C and Z (required)

cmpx XHigh, YHigh wcz 'Compare high longs; save C and Z (required)

Note: both C and Z flags are required to discern the possible outcomes of the comparison; X < Y, X = Y, or X > Y.

After executing the above, the C and Z flags will indicate the relationship between the two unsigned double-long (64-bit) values. If XHigh:XLow started as $0000\_0001:0000\_0000 (4,294,967,296) and YHigh:YLow was $0000\_0000:0000\_0001 (1) the resulting flags would be: Z = 0 and C = 0; (X > Y). This is demonstrated below. Note that the comparison is really just a subtraction with the result not written and the Z and C flags updated.

Hexadecimal Decimal Flags

(high) (low)

(XHigh:XLow) $0000\_0001:0000\_0000 4,294,967,296 n/a

- (YHigh:YLow) - $0000\_0000:0000\_0001 - 1 n/a

─────────────────── ────────────── ───────

= $0000\_0000:FFFF\_FFFF = 4,294,967,295 Z=0, C=0

#### 64-bit Signed Compare

In comparison to the above, a signed double-long (64-bit) comparison uses a CMPSX instead of CMPX as the last instruction:

cmp XLow, YLow wcz 'Compare low longs; save C and Z (required)

cmpsx XHigh, YHigh wcz 'Compare high longs; save C and Z (required)

Note: both C and Z flags are required to discern the possible outcomes of the comparison; X < Y, X = Y, or X > Y.

After executing the above, the C and Z flags indicate the relationship between the two signed double-long (64-bit) values. If XHigh:XLow started as $FFFF\_FFFF:FFFF\_FFFF (-1) and YHigh:YLow was $0000\_0000:0000\_0001 (1) the resulting flags would be: Z = 0 and C = 1; (Value1 < Value2). This is demonstrated below. Note that the comparison is really just a subtraction with the result not written and the Z and C flags updated.

Hexadecimal Decimal Flags

(high) (low)

(XHigh:XLow) $FFFF\_FFFF:FFFF\_FFFF -1 n/a

- (YHigh:YLow) - $0000\_0000:0000\_0001 - 1 n/a

─────────────────── ────────────── ───────

= $FFFF\_FFFF:FFFF\_FFFE = -2 Z=0, C=1

A signed triple-long (96-bit) comparison would look similar but with a CMPX instruction inserted between the CMP and CMPSX instructions:

cmp XLow, YLow wcz 'Compare low longs; save C and Z (required)

cmpx XMid, YMid wcz 'Compare middle longs; save C and Z (required)

cmpsx XHigh, YHigh wcz 'Compare high longs; save C and Z (required)

## Categorical Listing Of Propeller 2 Assembly Language

The following is a list of all elements of the PASM2 language grouped by category and ordered by type or frequency-of-use within each category.

### Directives

PASM2 directives instruct the compiler how to configure assembly code.

[ALIGNW](#_6y0y6yvl7r2z) Align to next word in Hub

[ALIGNL](#_va7a6kt7oiii) Align to next long in Hub

BYTE Insert byte data

WORD Insert word data

LONG Insert long data

ORG Set code for Cog RAM

ORGH Set code for Hub RAM

ORGF Fill Cog RAM with zeros

FIT Validate that code fits within Cog RAM or Hub RAM

RES Reserve long registers for symbol

### Conditions

Conditions are placed in front of instructions ( {Label} {Condition} Instruction Operands {Effect} ) to conditionally execute or exclude that instruction based on flag settings at runtime. Conditions are optional; omitting the condition means "always execute" the instruction (the default behavior).

[IF\_E](#_ofohv8f4qicc) If comparison/subtraction was equal (Z = 1)

[IF\_NE](#_ofohv8f4qicc) If comparison/subtraction was not equal (Z = 0)

[IF\_A](#_ofohv8f4qicc) If comparison/subtraction was above (C = 0 and Z = 0)

[IF\_AE](#_ofohv8f4qicc) If comparison/subtraction was above or equal (C = 0)

[IF\_B](#_ofohv8f4qicc) If comparison/subtraction was below (C = 1)

[IF\_BE](#_ofohv8f4qicc) If comparison/subtraction was below or equal (C = 1 or Z = 1)

[IF\_GT](#_ofohv8f4qicc) If comparison/subtraction was greater than (C = 0 and Z = 0)

[IF\_GE](#_ofohv8f4qicc) If comparison/subtraction was greater than or equal (C = 0)

[IF\_LT](#_ofohv8f4qicc) If comparison/subtraction was less than (C = 1)

[IF\_LE](#_ofohv8f4qicc) If comparison/subtraction was less than or equal (C = 1 or Z = 1)

[IF\_C](#_ofohv8f4qicc) If C set (C = 1)

[IF\_NC](#_ofohv8f4qicc) If C clear (C = 0)

[IF\_Z](#_ofohv8f4qicc) If Z set (Z = 1)

[IF\_NZ](#_ofohv8f4qicc) If Z clear (Z = 0)

[IF\_C\_EQ\_Z](#_ofohv8f4qicc) If C equal to Z (C = 0 and Z = 0 --or-- C = 1 and Z = 1)

[IF\_C\_NE\_Z](#_ofohv8f4qicc) If C not equal to Z (C = 0 and Z = 1 --or-- C = 1 and Z = 0)

[IF\_C\_AND\_Z](#_ofohv8f4qicc) If C set and Z set (C = 1 and Z = 1)

[IF\_C\_AND\_NZ](#_ofohv8f4qicc) If C set and Z clear (C = 1 and Z = 0)

[IF\_NC\_AND\_Z](#_ofohv8f4qicc) If C clear and Z set (C = 0 and Z = 1)

[IF\_NC\_AND\_NZ](#_ofohv8f4qicc) If C clear and Z clear (C = 0 and Z = 0)

[IF\_C\_OR\_Z](#_ofohv8f4qicc) If C set or Z set (C = 1 or Z = 1)

[IF\_C\_OR\_NZ](#_ofohv8f4qicc) If C set or Z clear (C = 1 or Z = 0)

[IF\_NC\_OR\_Z](#_ofohv8f4qicc) If C clear or Z set (C = 0 or Z = 1)

[IF\_NC\_OR\_NZ](#_ofohv8f4qicc) If C clear or Z clear (C = 1 or Z = 0)

[IF\_Z\_EQ\_C](#_ofohv8f4qicc) If Z equal to C (Z = 0 and C = 0 --or-- Z = 1 and C = 1)

[IF\_Z\_NE\_C](#_ofohv8f4qicc) If Z not equal to C (Z = 0 and C = 1 --or-- Z = 1 and C = 0)

[IF\_Z\_AND\_C](#_ofohv8f4qicc) If Z set and C set (Z = 1 and C = 1)

[IF\_Z\_AND\_NC](#_ofohv8f4qicc) If Z set and C clear (Z = 1 and C = 0)

[IF\_NZ\_AND\_C](#_ofohv8f4qicc) If Z clear and C set (Z = 0 and C = 1)

[IF\_NZ\_AND\_NC](#_ofohv8f4qicc) If Z clear and C clear (Z = 0 and C = 0)

[IF\_Z\_OR\_C](#_ofohv8f4qicc) If Z set or C set (Z = 1 or C = 1)

[IF\_Z\_OR\_NC](#_ofohv8f4qicc) If Z set or C clear (Z = 1 or C = 0)

[IF\_NZ\_OR\_C](#_ofohv8f4qicc) If Z clear or C set (Z = 0 or C = 1)

[IF\_NZ\_OR\_NC](#_ofohv8f4qicc) If Z clear or C clear (Z = 0 or C = 0)

[IF\_00](#_ofohv8f4qicc) If C clear and Z clear (C = 0 and Z = 0)

[IF\_01](#_ofohv8f4qicc) If C clear and Z set (C = 0 and Z = 1)

[IF\_10](#_ofohv8f4qicc) If C set and Z clear (C = 1 and Z = 0)

[IF\_11](#_ofohv8f4qicc) If C set and Z set (C = 1 and Z = 1)

[IF\_X0](#_ofohv8f4qicc) If Z clear (Z = 0)

[IF\_X1](#_ofohv8f4qicc) If Z set (Z = 1)

[IF\_0X](#_ofohv8f4qicc) If C clear (C = 0)

[IF\_1X](#_ofohv8f4qicc) If C set (C = 1)

[IF\_NOT\_00](#_ofohv8f4qicc) If C clear and Z clear (C = 0 and Z = 0)

[IF\_NOT\_01](#_ofohv8f4qicc) If C set or Z clear (C = 1 or Z = 0)

[IF\_NOT\_10](#_ofohv8f4qicc) If C clear or Z set (C = 0 or Z = 1)

[IF\_NOT\_11](#_ofohv8f4qicc) If C clear or Z clear (C = 0 or Z = 0)

[IF\_DIFF](#_ofohv8f4qicc) If C not equal to Z (C = 0 and Z = 1 --or-- C = 1 and Z = 0)

[IF\_SAME](#_ofohv8f4qicc) If C equal to Z (C = 0 and Z = 0 --or-- C = 1 and Z = 1)

[\_RET\_](#_ofohv8f4qicc) Always execute instruction then return if no branch; no context restore

### Effects

Nearly half of PASM2 instructions feature optional effects to modify the C and/or Z flags. Effects are placed at the end of such instructions ( {Label} {Condition} Instruction Operands {Effect} ) to affect flags, or omitted to leave flags as-is.

[ANDC](#_2qhfue5jrv4x) AND tested bit/pin into current C

[ANDZ](#_2qhfue5jrv4x) AND tested bit/pin into current Z

[ORC](#_2qhfue5jrv4x) OR tested bit/pin into current C

[ORZ](#_2qhfue5jrv4x) OR tested bit/pin into current Z

[XORC](#_2qhfue5jrv4x) XOR tested bit/pin into current C

[XORZ](#_2qhfue5jrv4x) XOR tested bit/pin into current Z

[WC](#_2qhfue5jrv4x) Write C flag

[WCZ](#_2qhfue5jrv4x) Write both C and Z flags

[WZ](#_2qhfue5jrv4x) Write Z flag

### 

### Flag Modification

Flag Modification commands (MODxx) alter the state of the C and/or Z flag according to the given modifier

( {Label} {Condition} MOD\_Instruction Modifier {Effect} ).

**MOD\_Instruction**

[MODC](#_m3xduhu6qlur) Modify C according to modifier

[MODZ](#_m3xduhu6qlur) Modify Z according to modifier

[MODCZ](#_m3xduhu6qlur) Modify C and Z according to modifier

**Modifier**

[\_CLR](#_hirqlphn43pc) Clear C/Z (C == 0 and/or Z == 0)

[\_E](#_hirqlphn43pc) Set C/Z if comparison/subtraction was equal (C == Z and/or Z == Z)

[\_NE](#_hirqlphn43pc) Set C/Z if comparison/subtraction was not equal (C == !Z and/or Z == !Z)

[\_GT](#_hirqlphn43pc) Set C/Z if comparison/subtraction was greater than (C == !C AND !Z and/or Z == !C AND !Z)

[\_GE](#_hirqlphn43pc) Set C/Z if comparison/subtraction was greater than or equal (C == !C and/or Z == !C)

[\_LT](#_hirqlphn43pc) Set C/Z if comparison/subtraction was less than (C == C and/or Z == C)

[\_LE](#_hirqlphn43pc) Set C/Z if comparison/subtraction was less than or equal (C == C OR Z and/or Z == C OR Z)

[\_C](#_hirqlphn43pc) Set C/Z to C (C == C and/or Z == C)

[\_NC](#_hirqlphn43pc) Set C/Z to inverse of C (C == !C and/or Z == !C)

[\_Z](#_hirqlphn43pc) Set C/Z to Z (C == Z and/or Z == Z)

[\_NZ](#_hirqlphn43pc) Set C/Z to inverse of Z (C == !Z and/or Z == !Z)

[\_C\_EQ\_Z](#_hirqlphn43pc) Set C/Z if C equal to Z (C == C = Z and/or Z == C = Z)

[\_C\_NE\_Z](#_hirqlphn43pc) Set C/Z if C not equal to Z (C == C <> Z and/or Z == C <> Z)

[\_C\_AND\_Z](#_hirqlphn43pc) Set C/Z to C *AND* Z (C == C AND Z and/or Z == C AND Z)

[\_C\_AND\_NZ](#_hirqlphn43pc) Set C/Z to C *AND* *NOT* Z (C == C AND !Z and/or Z == C AND !Z)

[\_NC\_AND\_Z](#_hirqlphn43pc) Set C/Z to *NOT* C *AND* Z (C == !C AND Z and/or Z == !C AND Z)

[\_NC\_AND\_NZ](#_hirqlphn43pc) Set C/Z to *NOT* C *AND* *NOT* Z (C == !C AND !Z and/or Z == !C AND !Z)

[\_C\_OR\_Z](#_hirqlphn43pc) Set C/Z to C *OR* Z (C == C OR Z and/or Z == C OR Z)

[\_C\_OR\_NZ](#_hirqlphn43pc) Set C/Z to C *OR* *NOT* Z (C == C OR !Z and/or Z == C OR !Z)

[\_NC\_OR\_Z](#_hirqlphn43pc) Set C/Z to *NOT* C *OR* Z (C == !C OR Z and/or Z == !C OR Z)

[\_NC\_OR\_NZ](#_hirqlphn43pc) Set C/Z to *NOT* C *OR* *NOT* Z (C == !C OR !Z and/or Z == !C OR !Z)

[\_Z\_EQ\_C](#_hirqlphn43pc) Set C/Z if Z equal to C (C == Z = C and/or Z == Z = C)

[\_Z\_NE\_C](#_hirqlphn43pc) Set C/Z if Z not equal to C (C == Z <> C and/or Z == Z <> C)

[\_Z\_AND\_C](#_hirqlphn43pc) Set C/Z to Z *AND* C (C == Z AND C and/or Z == Z AND C)

[\_Z\_AND\_NC](#_hirqlphn43pc) Set C/Z to Z *AND* *NOT* C (C == Z AND !C and/or Z == Z AND !C)

[\_NZ\_AND\_C](#_hirqlphn43pc) Set C/Z to *NOT* Z *AND* C (C == !Z AND C and/or Z == !Z AND C)

[\_NZ\_AND\_NC](#_hirqlphn43pc) Set C/Z to *NOT* Z *AND* *NOT* C (C == !Z AND !C and/or Z == !Z AND !C)

[\_Z\_OR\_C](#_hirqlphn43pc) Set C/Z to Z *OR* C (C == Z OR C and/or Z == Z OR C)

[\_Z\_OR\_NC](#_hirqlphn43pc) Set C/Z to Z *OR* *NOT* C (C == Z OR !C and/or Z == Z OR !C)

[\_NZ\_OR\_C](#_hirqlphn43pc) Set C/Z to *NOT* Z *OR* C (C == !Z OR C and/or Z == !Z OR C)

[\_NZ\_OR\_NC](#_hirqlphn43pc) Set C/Z to *NOT* Z *OR* *NOT* C (C == !Z OR !C and/or Z == !Z OR !C)

[\_SET](#_hirqlphn43pc) Set C/Z (C == 1 and/or Z == 1)

### 

### Augmentation

Augmentation instructions change or enhance the next instruction's behavior.

SETQ Set Q register for companion instruction

SETQ2 Set Q register for LUT RAM companion instruction

[AUGS](#_b6omkiqrkged) Augment next literal Src to 32-bits

[AUGD](#_qxo6681ulhaj) Augment next literal Dest to 32-bits

### Indirection

Indirection instructions modify the next instruction's target addresses.

[ALTB](#_r6knb6xhbyzp) Alter subsequent BITxxx instruction

[ALTSN](#_o72f3jsaebvh) Alter subsequent SETNIB instruction

[ALTGN](#_asw5rsrcqtpn) Alter subsequent GETNIB / ROLNIB instruction

[ALTSB](#_2jkx93n2978v) Alter subsequent SETBYTE instruction

[ALTGB](#_q234901aifm8) Alter subsequent GETBYTE / ROLBYTE instruction

[ALTSW](#_90jw6j8128ot) Alter subsequent SETWORD instruction

[ALTGW](#_hahk50n443um) Alter subsequent GETWORD / ROLWORD instruction

[ALTD](#_6iq3v0fpyc3a) Alter D field of next instruction

[ALTS](#_f2hcvwgutfa0) Alter S field of next instruction

[ALTR](#_3cn9gj4kxitq) Alter Result register address of next instruction

[SETD](#_jw7izhflytke) Set template D field for ALTI

[SETS](#_94hz48griij7) Set template S field for ALTI

[SETR](#_brza01jtowzy) Set template Result field for ALTI

[ALTI](#_ok3etaj2mvuf) Substitute next instruction's field values from template, per configuration

### Configuration

The HUBSET configuration instruction sets clock, write-protect, debug, filter, and PRNG configuration or initiates a software reset of the Propeller.

HUBSET Set global configuration, or software reset

### Cog Control

Cog control instructions monitor or alter current cog activity.

[COGID](#_uy4mwi8z5ww4) Get current cog’s ID or any cog's status by ID

[COGINIT](#_5knlosjgh2sz) Start an available cog, or restart a cog by ID

[COGSTOP](#_eg5nftcbzayn) Stop a cog by ID

### Process Control

Process control instructions coordinate exclusive actions between multiple cogs. See also the [Event Handling](#_gvm9il5cflgi) instructions like xxxATN and xxxSE1.

LOCKNEW Request a new lock

LOCKRET Return a lock

LOCKTRY Attempt ownership of a lock

LOCKREL Release ownership of a lock

### Flow Control

Flow control instructions direct execution to another part of the application. They are used for looping, subroutine, interrupt, or instruction skipping purposes. Also see the [Event Handling > Branch](#21avq9snug01) category for event-based flow control.

[CALL](#_lrwvkfn726tn) Call a subroutine; store return context on the stack

[CALLA](#_9bxg6hr5ntaa) Call a subroutine; store return context in the Hub long at PTRA++

[CALLB](#_9bxg6hr5ntaa) Call a subroutine; store return context in the Hub long at PTRB++

[CALLD](#_y34ndvfdx6a) Call a subroutine; store return context in PA/PB/PTRA/PTRB/D

[CALLPA](#_11plynh8xrpt) Call a subroutine; store return context on the stack and copy D into PA

[CALLPB](#_11plynh8xrpt) Call a subroutine; store return context on the stack and copy D into PB

RET Return from subroutine by popping stack; optional context restore

RETA Return from subroutine by reading Hub long at --PTRA; restore context

RETB Return from subroutine by reading Hub long at --PTRB; restore context

JMP Jump

JMPREL Jump relative (forward/back)

[DJZ](#_dm4flu2kbbs) Decrement value and jump if zero

[DJNZ](#_dm4flu2kbbs) Decrement value and jump if not zero

[DJF](#_jdehvyhk3dz3) Decrement value and jump if full (-1; $FFFF\_FFFF)

[DJNF](#_jdehvyhk3dz3) Decrement value and jump if not full (<> -1; <> $FFFF\_FFFF)

[IJZ](#_s858lrxz52tn) Increment value and jump if zero

[IJNZ](#_s858lrxz52tn) Increment value and jump if not zero

[TJZ](#_nz9y81fecbaa) Test value and jump if zero

[TJNZ](#_nz9y81fecbaa) Test value and jump if not zero

[TJF](#_1rck8xngsheq) Test value and jump if full (-1; $FFFF\_FFFF)

[TJNF](#_1rck8xngsheq) Test value and jump if not full (<> -1; <> $FFFF\_FFFF)

[TJS](#_rm5pk4tx7cv6) Test value and jump if signed

[TJNS](#_rm5pk4tx7cv6) Test value and jump if not signed

[TJV](#_o9pr7ygsoyes) Test value and jump if overflowed

REP Repeat next instructions

SKIP Skip next instructions by pattern

SKIPF Skip next instructions fast by pattern

EXECF Execute fast; jump to Cog RAM and set SKIPF pattern

RETI0 Return from interrupt 0

RETI1 Return from interrupt 1

RETI2 Return from interrupt 2

RETI3 Return from interrupt 3

RESI0 Resume from interrupt 0

RESI1 Resume from interrupt 1

RESI2 Resume from interrupt 2

RESI3 Resume from interrupt 3

### Lookup Table (LUT) Memory Access

Instructions that manipulate or share the cog's Lookup RAM.

RDLUT Read long from Lookup RAM

WRLUT Write long to Lookup RAM

SETLUTS Enable/disable Lookup RAM sharing

### Hub Memory Access

Instructions for manipulating Hub RAM.

**Random Access**

RDBYTE Read byte of Hub RAM

RDWORD Read word of Hub RAM

RDLONG Read long of Hub RAM

WRBYTE Write byte to Hub RAM

WRWORD Write word to Hub RAM

WRLONG Write long to Hub RAM

WMLONG Write non-zero bytes of long to Hub RAM

PUSHA Write long to Hub RAM at PTRA++

PUSHB Write long to Hub RAM at PTRB++

POPA Read long of Hub RAM at --PRTA

POPB Read long of Hub RAM at --PRTB

**FIFO Access**

GETPTR Get current FIFO Hub pointer

RDFAST Start new fast FIFO Hub read

WRFAST Start new fast FIFO Hub write

FBLOCK Set next block for FIFO

RFBYTE Read next byte from FIFO

RFWORD Read next word from FIFO

RFLONG Read next long from FIFO

WFBYTE Write next byte to FIFO

WFWORD Write next word to FIFO

WFLONG Write next long to FIFO

RFVAR Read variable-size (1 to 4 byte) value from FIFO

RFVARS Read variable-size (1 to 4 byte) sign-extended value from FIFO

### Streamer

Instructions to manipulate the cog's streamer.

SETXFRQ Set streamer NCO frequency

XINIT Issue streamer command immediately and zero the phase

XSTOP Stop streamer immediately

XZERO Buffer new streamer command; issued on final NCO rollover and zero the phase

XCONT Buffer new streamer command; issued on final NCO rollover and continue the phase

GETXACC Get the streamer's Goertzel accumulator results and clear the accumulators

### I/O Pins

Instructions for manipulating Core and Smart I/O pin features. In addition to these instructions, the DIRA / DIRB / OUTA / OUTB / INA / INB registers may be used as operands of instructions to affect or read I/O pins.

**Core Logic**

[DIRL](#_adfu0crzr5tx) Set pin(s) direction to input (0)

[DIRH](#_adfu0crzr5tx) Set pin(s) direction to output (1)

[DIRC](#_q5818ady3ccx) Set pin(s) direction to input/output according to C

[DIRNC](#_q5818ady3ccx) Set pin(s) direction to input/output according to !C

[DIRZ](#_eaobslhzfr83) Set pin(s) direction to input/output according to Z

[DIRNZ](#_eaobslhzfr83) Set pin(s) direction to input/output according to !Z

[DIRRND](#_5p7siddmqpeb) Set pin(s) direction to random input/output

[DIRNOT](#_8cdi9p9b7vgm) Toggle pin(s) to the opposite direction

[OUTL](#_40vs5rlpgy9a) Set pin(s) output level low (0)

[OUTH](#_40vs5rlpgy9a) Set pin(s) output level high (1)

[OUTC](#_73z1ljpedf7w) Set pin(s) output level to low/high according to C

[OUTNC](#_73z1ljpedf7w) Set pin(s) output level to low/high according to !C

[OUTZ](#_16ycek7kzbz5) Set pin(s) output level to low/high according to Z

[OUTNZ](#_16ycek7kzbz5) Set pin(s) output level to low/high according to !Z

[OUTRND](#_rxjj8bmeue2f) Set pin(s) output level to random low/high

[OUTNOT](#_mll0nu2hed2z) Toggle pin(s) to the opposite output level

[DRVL](#_cuc8iv9j9262) Set pin(s) direction to output and output level low (0)

[DRVH](#_cuc8iv9j9262) Set pin(s) direction to output and output level high (1)

[DRVC](#_odv0wkzdsytw) Set pin(s) direction to output and output level to low/high according to C

[DRVNC](#_odv0wkzdsytw) Set pin(s) direction to output and output level to low/high according to !C

[DRVZ](#_g20ikbjddn3e) Set pin(s) direction to output and output level to low/high according to Z

[DRVNZ](#_g20ikbjddn3e) Set pin(s) direction to output and output level to low/high according to !Z

[DRVRND](#_cfks3rynv54x) Set pin(s) direction to output and output level to random low/high

[DRVNOT](#_ra3uexs4cuzi) Set pin(s) direction to output and toggle to the opposite output level

[FLTL](#_rg1q051ww7h1) Set pin(s) direction to input and to an output level of low (0)

[FLTH](#_rg1q051ww7h1) Set pin(s) direction to input and to an output level of high (1)

[FLTC](#_1lkdbkm563zm) Set pin(s) direction to input and to an output level of low/high according to C

[FLTNC](#_1lkdbkm563zm) Set pin(s) direction to input and to an output level of low/high according to !C

[FLTZ](#_enfuevkmlntw) Set pin(s) direction to input and to an output level of low/high according to Z

[FLTNZ](#_enfuevkmlntw) Set pin(s) direction to input and to an output level of low/high according to !Z

[FLTRND](#_8wfub36x4ss6) Set pin(s) direction to input and to an output level of random low/high

[FLTNOT](#_9p7em82h5uqe) Set pin(s) direction to input and toggle to the opposite output level

[TESTP](#_d2afnf6ec911) Test pin and either store, AND, OR, or XOR the result into C/Z

[TESTPN](#_d2afnf6ec911) Test pin and either store, AND, OR, or XOR inverse result into C/Z

**Smart Logic**

[AKPIN](#_putbdsx21wr) Acknowledge smart pin(s)

RQPIN Read smart pin result and mode flag; no acknowledge (quiet)

RDPIN Read smart pin result and mode flag, then acknowledge

WRPIN Set smart pin(s) mode and acknowledge

WXPIN Set smart pin(s) "X" value and acknowledge

WYPIN Set smart pin(s) "Y" value and acknowledge

SETDACS Set smart pin digital to analog converters (DACs)

SETSCP Enable/disable four-channel oscilloscope pins

GETSCP Get four-channel oscilloscope samples

### Math

ALU circuit and CORDIC Solver math instructions. The ALU (Arithmetic Logic Unit) instructions perform common math operations in just 2 clock cycles each. The CORDIC (COordinate Rotation DIgital Computer) instructions perform more complicated math operations in 54 clock cycles each.

**ALU Circuit**

[ABS](#_lrvakeitwivz) Get the absolute value of a number

[NEG](#_ied83z8c60zq) Negate a value

[NEGC](#_jlhwuat7bnkq) Negate value according to C

[NEGNC](#_323cveis97t7) Negate value according to !C

[NEGZ](#_5vu7vu1xc9tm) Negate value according to Z

[NEGNZ](#_z1ulw37qnpt4) Negate value according to !Z

[ZEROX](#_9rn21a3cv769) Zero-extend value beyond designated bit

[SIGNX](#_923rdwqvf980) Sign-extend value beyond designated bit

[FGE](#_4qzq1q9ohttb) Force unsigned value to be greater than or equal to another

[FGES](#_6pb8tmuj8hrq) Force signed value to be greater than or equal to another

[FLE](#_kjr0bke2otvv) Force unsigned value to be less than or equal to another

[FLES](#_oo1zbqj5xudg) Force signed value to be less than or equal to another

[SUMC](#_8370tvf23i61) Adjust signed value by other C-negated value

[SUMNC](#_8370tvf23i61) Adjust signed value by other !C-negated value

[SUMZ](#_5u8few7uaee4) Adjust signed value by other Z-negated value

[SUMNZ](#_5u8few7uaee4) Adjust signed value by other !Z-negated value

[ADD](#_qhneb9rqvxbb) Add two unsigned values

[ADDX](#_vhdbx9ift4rv) Add two unsigned extended values

[ADDS](#_566nu7hwsj0f) Add two signed values

[ADDSX](#_23s34xc9b4y1) Add two signed extended values

[INCMOD](#_ln0kx81xiaak) Increment with modulus

[DECMOD](#_jjzqxknhgv63) Decrement with modulus

[SUB](#_q0s2vcp5e3i5) Subtract one unsigned value from another

[SUBX](#_4wyzstsl5slq) Subtract one unsigned extended value from another

[SUBS](#_phb5pufpspb9) Subtract one signed value from another

[SUBSX](#_fwvdz7pvnpz3) Subtract one signed extended value from another

[SUBR](#_ss3vapy9cq1h) Subtract one unsigned value from another (in reverse order to SUB)

[CMPSUB](#_tvdyh2esm2qw) Compare two unsigned values and subtract the second if it is lesser or equal

[MUL](#_wv511llidyd6) Multiply unsigned 16-bit x 16-bit values

[MULS](#_9awkxilhgl30) Multiply signed 16-bit x 16-bit values

[SHL](#_cg4e6rxaezp) Multiply 32-bit integer by power-of-two (shift left)

[SHR](#_y0jm929edm64) Divide unsigned 32-bit integer by power-of-two (shift right)

[SAR](#_er4hl0kfn1wn) Divide signed 32-bit integer by power-of-two (shift arithmetic right)

[SCA](#_72dwpag2ycaq) Create unsigned 16-bit scale value for next instruction's S value

[SCAS](#_eh4usclguwow) Create signed 18-bit scale value for next instruction's S value

[CMP](#_jw7rhjdisro0) Compare two unsigned values

[CMPX](#_3w81cwnriog1) Compare two unsigned values plus carry flag

[CMPS](#_nd6caax9v0zg) Compare two signed values

[CMPSX](#_ccc7sdk782ug) Compare two signed values plus carry flag

[CMPR](#_6j1a3848ozh3) Compare two unsigned values (in reverse order to CMP)

[CMPM](#_xxivkrt1rtt) Compare two unsigned values, get MSB of difference

**CORDIC Solver**

QLOG Start CORDIC number-to-logarithm conversion

QEXP Start CORDIC logarithm-to-number conversion

QMUL Start CORDIC unsigned multiplication

QDIV Start CORDIC unsigned division

QFRAC Start CORDIC unsigned division

QSQRT Start CORDIC square root

QROTATE Start CORDIC rotation or polar-to-cartesian conversion

QVECTOR Start CORDIC cartesian-to-polar conversion

GETQX Get lower long, quotient, root, X, length, logarithm, or integer CORDIC result

GETQY Get upper long, remainder, Y, or angle CORDIC result

### Timing

Instructions to retrieve the current clock value or wait for a future clock value. Also see xxxCT event instructions for handling timed events.

GETCT Get lower/upper 32-bits of System Counter

WAITX Wait for fixed or range-limited random number of clocks

### Event Handling

Event instructions create, configure, monitor, or branch on various events.

**Create/Configure**

[COGATN](#_wx29grvrexs) Strobe the attention signal of one or more cogs

[ADDCT1](#_u2xzjautl27l) Set counter event 1 trigger time

[ADDCT2](#_u2xzjautl27l) Set counter event 2 trigger time

[ADDCT3](#_u2xzjautl27l) Set counter event 3 trigger time

SETPAT Set pin pattern event

SETSE1 Set selectable event 1 configuration

SETSE2 Set selectable event 2 configuration

SETSE3 Set selectable event 3 configuration

SETSE4 Set selectable event 4 configuration

**Monitor**

[POLLATN](#_ref43okqtzzr) Retrieve and clear attention flag

[POLLCT1](#_dgitqmcw0hm) Retrieve and clear counter event 1 flag

[POLLCT2](#_dgitqmcw0hm) Retrieve and clear counter event 2 flag

[POLLCT3](#_dgitqmcw0hm) Retrieve and clear counter event 3 flag

[POLLFBW](#_86ta1z2g86ao) Retrieve and clear FIFO-interface-block-wrap event flag

[POLLINT](#_m2gclmhbm44b) Retrieve and clear interrupt-occurred event flag

[POLLPAT](#_2sqdtda5rj0e) Retrieve and clear pin-pattern-detected event flag

[POLLQMT](#_ge1n23vwgw41) Retrieve and clear CORDIC-read-but-empty event flag

[POLLSE1](#_n3mua6rjfhxs) Retrieve and clear selectable event 1 flag

[POLLSE2](#_n3mua6rjfhxs) Retrieve and clear selectable event 2 flag

[POLLSE3](#_n3mua6rjfhxs) Retrieve and clear selectable event 3 flag

[POLLSE4](#_n3mua6rjfhxs) Retrieve and clear selectable event 4 flag

[POLLXMT](#_mg19wcqd1pjt) Retrieve and clear streamer-empty event flag

[POLLXFI](#_r7m2wcrdq00o) Retrieve and clear streamer-finished event flag

[POLLXRO](#_n123uusqspjd) Retrieve and clear streamer-NCO-rollover event flag

[POLLXRL](#_63dai97cdrvc) Retrieve and clear streamer-LUT-RAM-rollover event flag

[WAITATN](#_sb5186rnvudg) Wait for and clear attention flag

[WAITCT1](#_u335ct3ffsan) Wait for and clear counter event 1 flag

[WAITCT2](#_u335ct3ffsan) Wait for and clear counter event 2 flag

[WAITCT3](#_u335ct3ffsan) Wait for and clear counter event 3 flag

[WAITFBW](#_qoevcy1s6y48) Wait for and clear FIFO-interface-block-wrap event flag

[WAITINT](#_rqj7wu7kf26o) Wait for and clear interrupt-occurred event flag

[WAITPAT](#_5i1v8ffdcpk2) Wait for and clear pin-pattern-detected event flag

[WAITSE1](#_43d9knl3whqm) Wait for and clear selectable event 1 flag

[WAITSE2](#_43d9knl3whqm) Wait for and clear selectable event 2 flag

[WAITSE3](#_43d9knl3whqm) Wait for and clear selectable event 3 flag

[WAITSE4](#_43d9knl3whqm) Wait for and clear selectable event 4 flag

[WAITXMT](#_1ird9xvgzp82) Wait for and clear streamer-empty event flag

[WAITXFI](#_zdbnefgvlfr5) Wait for and clear streamer-finished event flag

[WAITXRO](#_y7dswdrnedgw) Wait for and clear streamer-NCO-rollover event flag

[WAITXRL](#_nesp1ncmao0z) Wait for and clear streamer-LUT-RAM-rollover event flag

**Branch**

[JATN](#_yfon3snogp62) Jump if attention flag set

[JNATN](#_yfon3snogp62) Jump if attention flag clear

[JCT1](#_6f897k4gldkf) Jump if counter 1 event flag set

[JNCT1](#_6f897k4gldkf) Jump if counter 1 event flag clear

[JCT2](#_6f897k4gldkf) Jump if counter 2 event flag set

[JNCT2](#_6f897k4gldkf) Jump if counter 2 event flag clear

[JCT3](#_6f897k4gldkf) Jump if counter 3 event flag set

[JNCT3](#_6f897k4gldkf) Jump if counter 3 event flag clear

[JFBW](#_miks298accr1) Jump if FIFO interface block wrap event flag set

[JNFBW](#_miks298accr1) Jump if FIFO interface block wrap event flag clear

[JINT](#_wzi7hl4t1ffr) Jump if interrupt-occurred event flag set

[JNINT](#_wzi7hl4t1ffr) Jump if interrupt-occurred event flag clear

[JPAT](#_pnmnqy72ibwb) Jump if pin pattern event flag set

[JNPAT](#_pnmnqy72ibwb) Jump if pin pattern event flag clear

[JSE1](#_nv0zohzajupf) Jump if selectable event 1 flag set

[JNSE1](#_nv0zohzajupf) Jump if selectable event 1 flag clear

[JSE2](#_nv0zohzajupf) Jump if selectable event 2 flag set

[JNSE2](#_nv0zohzajupf) Jump if selectable event 2 flag clear

[JSE3](#_nv0zohzajupf) Jump if selectable event 3 flag set

[JNSE3](#_nv0zohzajupf) Jump if selectable event 3 flag clear

[JSE4](#_nv0zohzajupf) Jump if selectable event 4 flag set

[JNSE4](#_nv0zohzajupf) Jump if selectable event 4 flag clear

[JQMT](#_fik5o0ebifr2) Jump if CORDIC-read-but-empty event flag set

[JNQMT](#_fik5o0ebifr2) Jump if CORDIC-read-but-empty event flag clear

[JXMT](#_izeyc7za5235) Jump if streamer empty event flag set

[JNXMT](#_izeyc7za5235) Jump if streamer empty event flag clear

[JXFI](#_76euyh5v8o1z) Jump if streamer finished event flag set

[JNXFI](#_76euyh5v8o1z) Jump if streamer finished event flag clear

[JXRO](#_gut56vubumxk) Jump if streamer NCO rollover event flag set

[JNXRO](#_gut56vubumxk) Jump if streamer NCO rollover event flag clear

[JXRL](#_43ntogxc32ez) Jump if streamer LUT RAM rollover event flag set

[JNXRL](#_43ntogxc32ez) Jump if streamer LUT RAM rollover event flag clear

### Interrupts

Instructions to manage interrupts.

[ALLOWI](#_jv88q3ni4hqo) Allow interrupts (default)

[STALLI](#_4b448hiwgtyc) Stall Interrupts

TRGINT1 Trigger interrupt 1; regardless of STALLI mode

TRGINT2 Trigger interrupt 2; regardless of STALLI mode

TRGINT3 Trigger interrupt 3; regardless of STALLI mode

NIXINT1 Cancel interrupt 1

NIXINT2 Cancel interrupt 2

NIXINT3 Cancel interrupt 3

SETINT1 Set interrupt 1 source

SETINT2 Set interrupt 2 source

SETINT3 Set interrupt 3 source

[BRK](#_d8ho6miavbop) Trigger breakpoint in current cog

[COGBRK](#_okd9uhh52aur) Trigger breakpoint in specified cog

GETBRK Get breakpoint and cog status

### Bit Operations

Bit manipulation instructions.

[AND](#_ry94d81q3emx) Bitwise AND two values

[ANDN](#_ry94d81q3emx) Bitwise AND a value with the NOT of another

[OR](#_49iljyigagii) Bitwise OR two values

[XOR](#_ci31vbchcwxx) Bitwise XOR two values

[NOT](#_s858lrxz52tn) Bitwise NOT a value

[REV](#_8x2v3buaskxv) Reverse bits of value

[SHR](#_y0jm929edm64) Shift bits right

[SHL](#_cg4e6rxaezp) Shift bits left

[SAR](#_er4hl0kfn1wn) Shift bits right, extending the MSB

[SAL](#_dflmaw9re4j) Shift bits left, extending the LSB

[ROR](#_urnydzozfyv8) Rotate bits right

[ROL](#_e76w2o7qsgpp) Rotate bits left

[RCR](#_fbag5uwbvmwv) Rotate carry flag right into value

[RCL](#_75xnwhklnjzs) Rotate carry flag left into value

[RCZR](#_efvpakcsdenq) Rotate carry and zero flags right (2-bit rotate right)

[RCZL](#_bl2w5tdjasam) Rotate carry and zero flags left (2-bit rotate left)

[BITL](#_pbf6oqcly2oa) Set bit(s) low (0)

[BITH](#_pbf6oqcly2oa) Set bit(s) high (1)

[BITC](#_3gamq9k207in) Set bit(s) low/high according to C

[BITNC](#_3gamq9k207in) Set bit(s) low/high according to !C

[BITZ](#_3961stdwxxh2) Set bit(s) low/high according to Z

[BITNZ](#_3961stdwxxh2) Set bit(s) low/high according to !Z

[BITRND](#_v3oe9q9od6uf) Set bit(s) random low/high

[BITNOT](#_hgixqtr6y6sz) Toggle bit(s) to opposite state

[MUXC](#_hujma23idlak) Set discrete bits to C

[MUXNC](#_hujma23idlak) Set discrete bits to !C

[MUXZ](#_fsy0qkuiqsd6) Set discrete bits to Z

[MUXNZ](#_fsy0qkuiqsd6) Set discrete bits to !Z

[MUXNITS](#_xa6h00k6yh91) Set discrete bit pairs of a value to non-zero bit pair states of another

[MUXNIBS](#_so9xp16tbilq) Set discrete nibbles of a value to non-zero nibble states of another

[MUXQ](#_gg86hluj11kt) Set discrete bits of a value to that of another

[WRC](#_70my8h1n5l99) Write C to register

[WRNC](#_70my8h1n5l99) Write !C to register

[WRZ](#_4rwgnzl2tfbc) Write Z to register

[WRNZ](#_4rwgnzl2tfbc) Write !Z to register

[TESTB](#_sjy72z2klc7g) Test bit of D and either store, AND, OR, or XOR the result into flags

[TESTBN](#_sjy72z2klc7g) Test bit of !D and either store, AND, OR, or XOR the result into flags

[TEST](#_6m4jc27v9rub) Test D, or bitwise AND D with S, to affect flags only

[TESTN](#_q7r5myfbnfk5) Test D by bitwise ANDing with !S to affect flags only

[GETNIB](#_8yjp9q7d9ncx) Get a nibble from a value

[GETBYTE](#_t84l3o6tozfm) Get a byte from a value

[GETWORD](#_ppayepmbzjsl) Get a word from a value

[SETNIB](#_dtkktvehwfob) Set a nibble to new value

[SETBYTE](#_f8filcyhyg4l) Set a byte to new value

[SETWORD](#_9gwjt0ijlutf) Set a word to new value

[ROLNIB](#_5imtv7yml80t) Rotate a nibble left into a value

[ROLBYTE](#_8x2v3buaskxv) Rotate a byte left into a value

[ROLWORD](#_lhi32k9v1cp3) Rotate a word left into a value

[MOV](#_s858lrxz52tn) Move a value to a register

MOVBYTS Move bytes within D, described by S

[ENCOD](#_brfhxn6t7t41) Get bit position of top-most 1 of Src or Dest into Dest

[DECOD](#_vp3efvde1vn6) Decode value (0—31) into single-high-bit long

[BMASK](#_nui8bnpnjowd) Get 1..32-bit mask into Dest

[ONES](#_h7zpookt38cy) Get number of 1s from Dest or Src into Dest

RGBSQZ Squeeze 8:8:8 RGB value into word

RGBEXP Squeeze word RGB value into 8:8:8

SPLITB Split every 4th bit of D into bytes

MERGEB Merge bits of bytes in D

SPLITW Split odd/even bits of D into words

MERGEW Merge bits of words in D

SEUSSF Forward relocate and periodically invert bits within D

SEUSSR Reverse relocate and periodically invert bits within D

### Color Manipulation

Color value mixing and control instructions.

**Pixel Mixer**

[ADDPIX](#_n2at4numrp4) Add RGB colors with full saturation

MULPIX Multiply colors ($00—$FF = 0.0—1.0)

BLNPIX Alpha-blend colors using SETPIV value

MIXPIX Mix colors using SETPIX and SETPIV values

SETPIV Set blend factor for BLNPIX / MIXPIX

SETPIX Set mode for MIXPIX

**Color Space Converter**

SETCY Set colorspace luma (Y) value

SETCI Set colorspace chrominance in-phase (I) value

SETCQ Set colorspace chrominance quadrature (Q) value

SETCFRQ Set colorspace frequency

SETCMOD Set colorspace mode

### Miscellaneous

Miscellaneous instructions.

GETRND Get Xoroshiro128\*\* random value

XORO32 Create Xoroshiro32+ random value from D and store in next instruction's S value

CRCBIT Iterate CRC value with 1-bit injection against 32-bit polynomial

CRCNIB Iterate CRC value with 4-bit injection against 32-bit polynomial

PUSH Place value onto hardware stack

POP Remove value from hardware stack

LOC Create relative or absolutable address for relocatable code

[NOP](#_gdk4kvxdog0t) No operation, just elapse two cycles

### [Registers](#_6ift2cu57w3u)

Special register symbols for direct feature access.

[DIRA](#_6ift2cu57w3u) Output enables for P31..P0

[DIRB](#_6ift2cu57w3u) Output enables for P63..P32

[OUTA](#_6ift2cu57w3u) Output states for P31..P0

[OUTB](#_6ift2cu57w3u) Output states for P63..P32

[INA](#_6ift2cu57w3u) Input states for P31..P0

[INB](#_6ift2cu57w3u) Input states for P63..P32

[PA](#_6ift2cu57w3u) CALLD-imm return, CALLPA parameter, or LOC address

[PB](#_6ift2cu57w3u) CALLD-imm return, CALLPB parameter, or LOC address

[PTRA](#_6ift2cu57w3u) Pointer A to Hub RAM

[PTRB](#_6ift2cu57w3u) Pointer B to Hub RAM

[IJMP1](#_6ift2cu57w3u) Interrupt call address for INT1

[IJMP2](#_6ift2cu57w3u) Interrupt call address for INT2

[IJMP3](#_6ift2cu57w3u) Interrupt call address for INT3

[IRET1](#_6ift2cu57w3u) Interrupt return address for INT1

[IRET2](#_6ift2cu57w3u) Interrupt return address for INT2

[IRET3](#_6ift2cu57w3u) Interrupt return address for INT3

[PR0..PR7](#_6ift2cu57w3u) PASM2 to Spin2 shared registers

### Constants

Constants that represent special numeric values and modes.

**Numerics**

TRUE Logical true: -1 ($FFFFFFFF)

FALSE Logical false: 0 ($00000000)

POSX Maximum positive integer: 2,147,483,647 ($7FFFFFFF)

NEGX Maximum negative integer: -2,147,483,648 ($80000000)

PI Floating-point value for PI: ~3.141593 ($40490FDB)

**Cog Initialization**

COGEXEC Start a cog by ID in Cog Execution mode

COGEXEC\_NEW Start the next available cog in Cog Execution mode

COGEXEC\_NEW\_PAIR Start the next available cog pair in Cog Execution mode with LUT RAM sharing

HUBEXEC Start a cog by ID in Hub Execution mode

HUBEXEC\_NEW Start the next available cog in Hub Execution mode

HUBEXEC\_NEW\_PAIR Start the next available cog pair in Hub Execution mode with LUT RAM sharing

**Smart Pins**

P\_ADC..P\_XOR\_AB 116 smart pin mode constants for WRPIN instruction

**Streamer**

X\_16P\_2DAC8\_WFWORD..

X\_WRITE\_ON 78 streamer mode constants for XINIT / XZERO / XCONT instructions

**Event/Interrupt**

EVENT\_ATN..INT\_OFF 17 event/interrupt mode constants for SETINTx and GETBRK instructions

### Operators

Operators for constant-expressions in PASM2 operands.

**Unary**

! Bitwise: NOT

+ Positive (+X) unary form of Add

- Negate (−X); unary form of Subtract

**Binary**

>> Bitwise: Shift right

<< Bitwise: Shift left

& Bitwise: AND

| Bitwise: OR

^ Bitwise: XOR

+ Add

- Subtract

\* Multiply and return lower 32 bits (signed)

/ Divide and return quotient (signed)

+/ Divide and return quotient (unsigned)

// Divide and return remainder (signed)

+// Divide and return remainder (unsigned)

#> Limit minimum (signed)

<# Limit maximum (signed)

< Boolean: Is less than (signed)

+< Boolean: Is less than (unsigned)

> Boolean: Is greater than (signed)

+> Boolean: Is greater than (unsigned)

<= Boolean: Is less than or equal (signed)

+<= Boolean: Is less than or equal (unsigned)

>= Boolean: Is greater than or equal (signed)

+>= Boolean: Is greater than or equal (unsigned)

== Boolean: Is equal

<> Boolean: Is not equal

!! Boolean: NOT

&& Boolean: AND

|| Boolean: OR

^^ Boolean: XOR

<=> Signed comparison (<, =, > returns -1, 0, 1)

**Ternary**

? : Ternary: return 2nd or 3rd value based on 1st

## Assembly Language Elements

### Term Definitions

The Propeller 2 Assembly Language syntax, tables, and descriptions use the following terms extensively.

**A / Addr:** A 20-bit relative or absolute value used to change PC (the program counter).

**C** / **Carry Flag:** A 1-bit persistent "flag" value representing a special state prior to or after an instruction executes. The C flag is traditionally output by a processor's adder circuit (ALU) to indicate that the previous mathematical operation resulted in a "carry" or a "borrow." In addition to this use, the Propeller has many instruction-specific meanings for the C flag (both for instruction input and output). When **C** appears in an instruction opcode, it indicates that it optionally writes to the C flag at the end of its execution, governed by the WC or WCZ effect.

**D / Dest / Destination:** The target register that an instruction ultimately affects. Usually the Destination operand is a 9-bit register address, but may be a 32-bit augmented value (a 9-bit value extended to 32-bits by a previous AUGD instruction). Often the value in the Destination register is read, manipulated, and overwritten by the execution of the instruction. The final value is also referred to as Result (or R). Some instructions use Destination for both the source value and the result's landing place.

**H / Hub Long:** Hub RAM long (4 bytes) used to store subroutine calling context states (C and Z flags and return address).

**I:** Immediate "literal" flag. 1 = Src operand value is an immediate (literal) value; 0 = Src operand value is a register address.

**K / Stack:** The 8-level hardware stack used for subroutine calls (stores C, Z, and PC) and temporary 32-bit push/pop storage.

**L:** Literal flag. 1 = Dest operand value is a literal (immediate) value; 0 = Dest operand value is a register address.

**N:** Index number. 0–1, 0–3, or 0–7 = An index value for instructions with a third operand.

**PC:** Program counter that determines the next instruction to be read by the cog's pipeline. This is a hidden, dedicated register whose value automatically increments by 1 or 4 (Cog/LUT execution or Hub execution) unless altered by a branching instruction such as DJNZ, CALL, or JMP.

**R:** Relative flag. 1 = address is relative; 0 = address is absolute.

**Result:** Value written at the end of instruction execution; often the Destination (i.e. D or Dest) operand.

**S / Src / Source:** The origin value that many instructions operate with. Often, the Source operand is a 9-bit literal value, but can also be the address of a register containing the source value, or may be a 32-bit augmented value (a 9-bit literal that is extended to 32-bits by a previous AUGS instruction or the convenient ## prefix).

**W:** Register to write. 00–11 = index of special register to write (PA, PB, PTRA, or PTRB).

**Z** / **Zero** **Flag:** A 1-bit persistent "flag" value representing a special state prior to or after an instruction executes. The Z flag is traditionally output by a processor's adder circuit (ALU) to indicate that the previous mathematical operation resulted in a result of "zero." In addition to this use, the Propeller has other instruction-specific uses for the Z flag (for instruction input and output). When **Z** appears in an instruction opcode, it indicates that it optionally writes to the Z flag at the end of its execution, governed by the WZ or WCZ effect.

### Opcode Tables

Most syntax definitions include an opcode table similar to the one below. This table lists the instruction’s 32-bit opcode encoding, register and flag outputs, and execution time in clock cycles.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0001010 | CZI | DDDDDDDDD | SSSSSSSSS | D | sign of (D + S) | Result = 0 | 2 |

The above example is from the ADDS instruction which has one syntax form. For instructions with multiple syntax forms, the opcode table will include a separate row for each, like this from the CALL instruction:

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101101 | RAA | AAAAAAAAA | AAAAAAAAA | PC | — | — | 4 / 13–20 |
| EEEE | 1101011 | CZ0 | DDDDDDDDD | 000101101 | PC | D[31] | D[30] | 4 / 13–20 |

The table’s first column describes the PASM2 instruction's opcode, consisting of the following fields:

* **COND** (bits 31:28) - Indicates the condition in which to execute the instruction.
* **INSTR** (bits 27:21) - Indicates the instruction or class of instruction.
* **FX** (bits 20:18) - Indicates the instruction’s effect status and SRC field meaning.
* **DEST** (bits 17:9) - Contains the 9-bit destination register address.
* **SRC** (bits 8:0) - Contains the 9-bit source register address or literal value.

The bits of the **COND** field default to all ones (1111) if no condition is specified on the instruction line, as is common; meaning "execute always." The bit pattern will be different if a condition is specified, according to that shown for the IF\_x (Conditions).

The **INSTR** field uniquely identifies the specific instruction, or instruction class. Some instructions have aliases (different forms of the same class of instruction) which are further uniquely identified by the way the **FX**, **DEST**, and **SRC** fields are encoded.

Each bit of the **FX** (effects) field is high or low (1 or 0) usually to describe the ‘C’ flag, ‘Z’ flag, and ‘I’mmediate nature of the instruction; noted as the encoding "CZI" in that field. The C and Z bits are set (1) when the instruction line specifies a WC, WZ, or WCZ effect, meaning write (update) the C and/or Z flag at the end of the instruction execution. The "I" bit is set (1) when the instruction's Src operand is prefixed by a "#" sign, meaning the **SRC** field contains an immediate *literal* value, rather than a register address where the value is located. Certain instructions have fixed values (0 or 1) in some or all of these bits when the normal use (C, Z, or I) has no purpose or when the **DEST** or **SRC** fields have special meaning. A bit marked "L" is set (1) when the instruction's Dest operand is prefixed by a "#" sign, meaning the **DEST** field contains an immediate *literal* value, rather than a register address where the value is located. A bit marked "R" is set (1) when the instruction's Address operand is *not* prefixed by a "\" sign, meaning the address is relative rather than absolute.

The **DEST** and **SRC** fields show the encoding of the instruction's Destination and Source operands, often "D" and "S" to indicate 9-bits from the given destination and source symbol/value on the instruction line.

The **Write** column indicates what register (if any) is written by the instruction. This is most often D (the destination register indicated by the address in the D operand) but may be PC (the program counter) or I/O registers. A "—" means nothing is written.

The **Z Flag** and **C Flag** columns indicate what value (if any) is written to the flag by the instruction. This only applies when the instruction line includes a WC, WZ, or WCZ effect, otherwise the flag is left as-is. A "—" means the flag is never updated.

The **Clocks** column indicates the number of clock cycles the instruction takes to execute. A single value or range (# *--or--* #–#) is the number of clocks the instruction takes regardless of its location in memory (Reg/LUT/Hub). Two values or ranges separated by a slash "/" (# / #–# *--or--* #–# / #–#) is the number of clock cycles if the instruction is executing from Reg or LUT RAM (left of the slash) and the number of clock cycles if executing from Hub RAM (right of the slash). A value shown as "# or #" is the number of clock cycles if not branching (left of the "or") or if branching (right of the "or").

### Propeller Assembly Instruction Master Table

All elements (symbols) of the PASM2 language are listed below either in alphabetical order or grouped within a category that fits alphabetically. To find or learn about elements by category, use the main [categorical listing](#_je0gsyt7f7wn) and follow its links to reach the details below.

## ABS

Absolute

[**Math Instruction**](#_xnk9pb8j5sof) **-** Get the absolute value of a number.

ABS *Dest*, {#}*Src* {WC|WZ|WCZ}

ABS *Dest* {WC|WZ|WCZ}

**Result:** Absolute Src (or Dest) value is stored in Dest.

* Dest is the register in which to write the absolute value of Dest or Src.
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose absolute value is written to Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0110010 | CZI | DDDDDDDDD | SSSSSSSSS | D | S[31] | Result = 0 | 2 |
| EEEE | 0110010 | CZ0 | DDDDDDDDD | DDDDDDDDD | D | D[31] | Result = 0 | 2 |

**Related:** [NEG](#_ied83z8c60zq)

**Explanation:**

ABS determines the absolute value of Src or Dest and writes the result into Dest.

If the WC or WCZ effect is specified, the C flag is set (1) if the *original* Src or Dest value was negative, or is cleared (0) if it was positive.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

Literal Src values are zero-extended, so ABS is really best used with register Src (or augmented Src) values.

## ADD

Add

[**Math Instruction**](#_xnk9pb8j5sof) **-** Add two unsigned values.

ADD *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Sum of unsigned Src and unsigned Dest is stored in Dest.

* Dest is a register containing the value to add Src to, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is added into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0001000 | CZI | DDDDDDDDD | SSSSSSSSS | D | carry of (D + S) | Result = 0 | 2 |

**Related:** [ADDX](#_vhdbx9ift4rv), [ADDS](#_566nu7hwsj0f), [ADDSX](#_23s34xc9b4y1), and [SUB](#_q0s2vcp5e3i5)

**Explanation:**

ADD sums the two unsigned values of Dest and Src together and stores the result into the Dest register.

If the WC or WCZ effect is specified, the C flag is set (1) if the summation results in a 32-bit overflow (unsigned carry), or is cleared (0) if no overflow.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result of Dest + Src is zero, or is cleared (0) if it is non-zero.

To add unsigned, multi-long values, use ADD followed by ADDX as described in [Adding Two Multi-Long Values](#_qt1j03hxivpt). ADD and ADDX are also used in adding signed, multi-long values with [ADDSX](#_23s34xc9b4y1) ending the sequence.

## ADDCT1/2/3

Add counter 1/2/3

[**Event Handling Instruction**](#_6ehq03n22voi) **-** Set one of three counter events' trigger time.

ADDCT1 *Dest*, {#}*Src*

ADDCT2 *Dest*, {#}*Src*

ADDCT3 *Dest*, {#}*Src*

**Result:** The Src value is added into Dest and the result is also stored in the hidden CT1, CT2, or CT3 event trigger register. The event triggers when System Counter (CT) = original Dest + Src.

* Dest is a register containing the value to add Src to, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is added into Dest.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1010011 | 00I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1010011 | 01I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1010011 | 10I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |

**Related:** [POLLCTx](#_dgitqmcw0hm), [WAITCTx](#_u335ct3ffsan), [JCTx](#_6f897k4gldkf), and [JNCTx](#_6f897k4gldkf)

**Explanation:**

ADDCT1, ADDCT2, or ADDCT3 sets the hidden CT1, CT2, or CT3 event trigger register (respectively) to the value of Dest + Src. The result is also written to Dest.

Use the xxxxCTx [Event Handling](#_6ehq03n22voi) instructions to process time-based events.

## ADDPIX

Add pixels

[**Color Manipulation Instruction**](#_oahsp8cfzdfl) **-** Add RGB colors with full saturation.

ADDPIX *Dest*, {#}*Src*

**Result:** Src color value bytes are added into Dest color value bytes with full saturation.

* Dest is a register containing the RGB color value to add Src to, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose RGB color value bytes are added into Dest.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1010010 | 00I | DDDDDDDDD | SSSSSSSSS | D | — | — | 7 |

**Explanation:**

ADDPIX sums individual RGB (reg, green, blue) color values of Src into that of Dest and stores the result in the Dest register.

## ADDS

Add signed

[**Math Instruction**](#_xnk9pb8j5sof) **-** Add two signed values.

ADDS *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Sum of signed Src and signed Dest is stored in Dest.

* Dest is a register containing the value to add Src to, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is added into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0001010 | CZI | DDDDDDDDD | SSSSSSSSS | D | sign of (D + S) | Result = 0 | 2 |

**Related:** [ADD](#_qhneb9rqvxbb), [ADDX](#_vhdbx9ift4rv), [ADDSX](#_23s34xc9b4y1), and [SUBS](#_phb5pufpspb9)

**Explanation:**

ADDS sums the two signed values of Dest and Src together and stores the result into the Dest register. If Src is a 9-bit literal, its value is interpreted as positive (0-511; it is not sign-extended) — use ##Value (or insert a prior AUGS instruction) for a 32-bit signed value; negative or positive.

If the WC or WCZ effect is specified, the C flag is set (1) if the summation results in a signed overflow (signed carry), or is cleared (0) if no overflow.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result of Dest + Src is zero, or is cleared (0) if it is non-zero.

To add signed, multi-long values, use ADD (not ADDS) followed possibly by ADDX, and finally ADDSX as described in [Adding Two Multi-Long Values](#_qt1j03hxivpt).

## ADDSX

Add signed, extended

[**Math Instruction**](#_xnk9pb8j5sof) **-** Add two signed extended values.

ADDSX *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Sum of signed Src plus C and signed Dest is stored in Dest.

* Dest is a register containing the value to add Src pluc C to, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value plus C is added into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0001011 | CZI | DDDDDDDDD | SSSSSSSSS | D | sign of (D+S+C) | Z AND (Result = 0) | 2 |

**Related:** [ADD](#_qhneb9rqvxbb), [ADDX](#_vhdbx9ift4rv), [ADDSX](#_23s34xc9b4y1), and [SUBSX](#_fwvdz7pvnpz3)

**Explanation:**

ADDSX sums the signed values of Dest and Src plus C together and stores the result into the Dest register. The ADDSX instruction is used to perform signed multi-long (extended) addition, such as 64-bit addition.

If the WC or WCZ effect is specified, the C flag is set (1) if the result is negative (Result[31] = 1), or is cleared (0) if positive. Use WC or WCZ on preceding ADD and ADDX instructions for proper final C flag.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Z was previously set and the result of Dest + Src + C is zero, or it is cleared (0) if non-zero. Use WZ or WCZ on preceding ADD and ADDX instructions for proper final Z flag.

To add signed multi-long values, use ADD (not ADDS) followed possibly by ADDX, and finally ADDSX as described in [Adding Two Multi-Long Values](#_qt1j03hxivpt).

## ADDX

Add extended

[**Math Instruction**](#_xnk9pb8j5sof) **-** Add two unsigned extended values.

ADDX *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Sum of unsigned Src plus C and unsigned Dest is stored in Dest.

* Dest is a register containing the value to add Src pluc C to, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value plus C is added into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0001001 | CZI | DDDDDDDDD | SSSSSSSSS | D | carry of (D+S+C) | Z AND (Result = 0) | 2 |

**Related:** [ADD](#_qhneb9rqvxbb), [ADDSX](#_23s34xc9b4y1), and [SUBX](#_4wyzstsl5slq)

**Explanation:**

ADDX sums the unsigned values of Dest and Src plus C together and stores the result into the Dest register. The ADDX instruction is used to perform unsigned multi-long (extended) addition, such as 64-bit addition.

If the WC or WCZ effect is specified, the C flag is set (1) if the summation resulted in an unsigned carry, or is cleared (0) if no carry. Use WC or WCZ on preceding ADD and ADDX instructions for proper final C flag. If C is set after the last ADDX in a multi-long addition, it indicates unsigned overflow.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Z was previously set and the result of Dest + Src + C is zero, or it is cleared (0) if non-zero. Use WZ or WCZ on preceding ADD and ADDX instructions for proper final Z flag.

To add unsigned multi-long values, use ADD followed by one or more ADDX instructions as described in [Adding Two Multi-Long Values](#_qt1j03hxivpt).

## AKPIN

Acknowledge pin

[**I/O Pin (Smart Logic) Instruction**](#wetsnv5av285) **-** Acknowledge smart pin(s).

AKPIN {#}*Src*

**Result:** One or more Smart Pins is acknowledged; lowering their corresponding IN signal(s).

* Src is a register, 9-bit literal, or 11-bit augmented literal whose value identifies the Smart Pin(s) to acknowledge.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1100000 | 01I | 000000001 | SSSSSSSSS | Ack Bus | — | — | 2 |

**Explanation:**

AKPIN acknowledges the Smart Pin(s) designated by Src. This lowers the corresponding IN signal(s) so that future Smart Pin events may raise them again later.

Src[5:0] indicates the pin number (0–63). For a range of Smart Pins, Src[5:0] indicates the first pin number (0–63) and Src[10:6] indicates how many contiguous pins beyond the first should be affected (1–31).

A 9-bit literal Src is enough to express the starting pin (Src[5:0]) and a range of up to 8 contiguous pins (Src[8:6]). If needed, use the augmented literal feature (##Src) to augment Src to the required 11-bit literal value— this inserts an AUGS instruction prior.

When Src is a register, the register's value bits [10:0] are used as-is to form the 11-bit Smart Pin range, unless a SETQ instruction immediately precedes the AKPIN instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for AKPIN's use.

The range calculation (from Src[5:0] up to Src[5:0]+Src[10:6]) will wrap within the same 32-pin group (DIRA or DIRB); it will not cross the port boundary.

## ALIGNL

Align long

[**Directive**](#_dq838bc22zwt) **-** Align to next long in Hub RAM.

DAT

*code\_and\_data\_statements*

ALIGNL

*data\_statements*

**Result:** The next data element is long-aligned in Hub RAM by emitting up to three bytes (each $00) prior.

* Code\_and\_data\_statements are leading program code and/or data.
* Data\_statements begin long-aligned in Hub RAM.

**Explanation:**

ALIGNL aligns the next data element to the beginning of the next long of Hub RAM. ALIGNL is important to use when code requires certain data to begin on a long boundary (for access convenience and speed).

ALIGNL is only allowed in DAT blocks, not in in-line PASM.

**Example**

The following creates a data table of a byte ($11), a word ($2222), and a long ($33333333) meant for access from Hub RAM.

DAT

T1 BYTE $11

T2 WORD $2222

LONG $33333333

This data *may* be emitted into the Hub memory image like below; the actual data start and alignment will vary depending on the code and data that precede it. The L#, W#, and B# labels denote contiguous long, word, and byte boundaries.

| L0 | | | | L1 | | | | L2 | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| W0 | | W1 | | W2 | | W3 | | W4 | | W5 | |
| B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | B10 | B11 |
| $11 | $22 | $22 | $33 | $33 | $33 | $33 | -- | -- | -- | -- | -- |

Notice how each data element, regardless of size, is packed right next to the data before it. If the code that is meant to access Table T2 expects it to align with a long boundary (i.e. for convenience or speed), the ALIGNL directive achieves this, as follows.

DAT

T1 BYTE $11

ALIGNL

T2 WORD $2222

LONG $33333333

In comparison, this data will be emitted as follows:

| L0 | | | | L1 | | | | L2 | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| W0 | | W1 | | W2 | | W3 | | W4 | | W5 | |
| B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | B10 | B11 |
| $11 | $00 | $00 | $00 | $22 | $22 | $33 | $33 | $33 | $33 | -- | -- |

In this case, the ALIGNL instruction causes three zero ($00) bytes to emit after Table T1 to automatically pad and align the start of Table T2 to the boundary of L1. Note that the second element (a long) of Table T2 is still packed right after the first element (a word) which may require further attention depending on the needs of the code accessing it.

## ALIGNW

Align word

[**Directive**](#_dq838bc22zwt) **-** Align to next word in Hub RAM.

DAT

*code\_and\_data\_statements*

ALIGNW

*data\_statements*

**Result:** The next data element is word-aligned in Hub RAM by emitting zero or one byte ($00) prior.

* Code\_and\_data\_statements are leading program code and/or data.
* Data\_statements begin word-aligned in Hub RAM.

**Explanation:**

ALIGNW aligns the next data element to the beginning of the next word of Hub RAM. ALIGNW is important to use when code requires certain data to begin on a word boundary (for access convenience and speed).

ALIGNW is only allowed in DAT blocks, not in in-line PASM.

**Example**

The following creates a data table of a byte ($11), a word ($2222), and a long ($33333333) meant for access from Hub RAM.

DAT

T1 BYTE $11

T2 WORD $2222

LONG $33333333

This data *may* be emitted into the Hub memory image like below; the actual data start and alignment will vary depending on the code and data that precede it. The L#, W#, and B# labels denote contiguous long, word, and byte boundaries.

| L0 | | | | L1 | | | | L2 | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| W0 | | W1 | | W2 | | W3 | | W4 | | W5 | |
| B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | B10 | B11 |
| $11 | $22 | $22 | $33 | $33 | $33 | $33 | -- | -- | -- | -- | -- |

Notice how each data element, regardless of size, is packed right next to the data before it. If the code that is meant to access Table T2 expects it to align with a word boundary (i.e. for convenience or speed), the ALIGNW directive achieves this, as follows.

DAT

T1 BYTE $11

ALIGNW

T2 WORD $2222

LONG $33333333

In comparison, this data will be emitted as follows:

| L0 | | | | L1 | | | | L2 | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| W0 | | W1 | | W2 | | W3 | | W4 | | W5 | |
| B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | B10 | B11 |
| $11 | $00 | $22 | $22 | $33 | $33 | $33 | $33 | -- | -- | -- | -- |

In this case, the ALIGNW instruction causes one zero ($00) byte to emit after Table T1 to automatically pad and align the start of Table T2 to the boundary of W1.

## ALLOWI

Allow interrupts

[**Interrupt Instruction**](#_40bj2qk54rdh) **-** Allow interrupts.

ALLOWI

**Result:** Any stalled and future interrupts are allowed.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | 000 | 000100000 | 000100100 | — | — | — | 2 |

**Related:** [STALLI](#_4b448hiwgtyc)

**Explanation:**

ALLOWI re-enables interrupt branching; the default on cog start. ALLOWI is the complement of the STALLI instruction— both are used to protect short, vital sections of main code from timing jitter or state loss caused by asynchronous interrupt handling.

## ALTB

Alter bit

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Alter subsequent BITxxx instruction.

ALTB *Dest*, {#}*Src*

ALTB *Dest*

**Result:** The next instruction's pipelined Dest value is altered to be (Src + Dest[13:5]) & $1FF, or just Dest[13:5] for syntax 2.

* Dest is the register whose 14-bit value is the *index*, or the full bit address, for the BITxxx instruction to operate on.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value contains a *base* long address (Src[8:0]; added to *index* (Dest[13:5]) for BITxxx) and also an optional auto-indexer value (Src[17:9]; added to Dest at the end of execution).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001100 | 11I | DDDDDDDDD | SSSSSSSSS | D1 | — | — | 2 |
| EEEE | 1001100 | 111 | DDDDDDDDD | 000000000 | D1 | — | — | 2 |

1 Dest is post-adjusted by the auto-indexer value; the sign-extended Src[17:9]. In syntax 2, the auto-indexer value is 0.

**Explanation:**

ALTB should be followed by a BITxxx instruction— it modifies the BITxxx instruction's Dest value, enabling code to iterate through multiple bits of data across a range of Reg RAM. BITxxx's Dest value is changed to (Src + Dest[13:5]) & $1FF (for syntax 1), or to Dest[13:5] (for syntax 2).

Dest[13:5] corresponds to the target long register's 9-bit address and Dest[4:0] is the bit ID within it; values of 0–31 identify individual bits, by position, in least-significant bit order. Iteratively executing ALTB followed by a BITxxx instruction, and each time incrementing ALTB's 14-bit Dest value by one, effectively writes a stream of bit values to Reg RAM as if it were all made of bit-sized registers.

**Warning:** BITxxx instructions optionally operate on a range of bits, encoded in the Src value— they don't limit themselves to only reading Src[4:0] for the bit number. For this reason, care must be taken when using ALTB with BITxxx or the *index* value (often used for the Src of the altered instruction) will be misinterpreted as multiple bits to affect. One way to solve this is to use a SETQ #0 followed by the ALTB then BITxxx instructions to force BITxxx's Src[9:5] bits to 0; i.e. no extra bits beyond the single bit described by Src[4:0].

In syntax 1, Src consists of two 9-bit fields; a base address (Src[8:0]) and a signed auto-indexer (Src[17:9]).

* The base is the Reg RAM address where the series of bits begins. ALTB adds the *long* index (Dest[13:5]) to the base (Src[8:0]) to locate the register holding the target bit. The *bit* ID (Dest[4:0]) identifies the bit's position within that long register.
* At the end of ALTB execution, the optional auto-indexer value (usually 0, 1, or -1) is added to the 14-bit index (Dest) for a future ALTB+BITxxx iteration.

In syntax 2, Dest serves as the full bit address— it's the same format as in syntax 1, but represents the target long's absolute address and its bit index instead of the long's relative index (to add to a base) and bit index.

**Notes:**

* The instruction following ALTB is shielded from interrupt
* ALTB alters the next instruction regardless of its kind— the intention is for it to be a BITxxx
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## ALTD

Alter destination

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Alter D field of next instruction.

ALTD *Dest*, {#}*Src*

ALTD *Dest*

**Result:** The next instruction's pipelined Dest value is altered to be (Src + Dest) & $1FF, or just Dest[8:0] in syntax 2.

* Dest is the register whose 9-bit value is the *offset*, or the full value, for the next instruction to operate on.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value contains a *base* (Src[8:0]; added to *offset* (Dest) for the next instruction) and also an optional auto-indexer value (Src[17:9]; added to Dest at the end of execution).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001100 | 01I | DDDDDDDDD | SSSSSSSSS | D1 | — | — | 2 |
| EEEE | 1001100 | 011 | DDDDDDDDD | 000000000 | D1 | — | — | 2 |

1 Dest is post-adjusted by the auto-indexer value; the sign-extended Src[17:9]. In syntax 2, the auto-indexer value is 0.

**Explanation:**

ALTD modifies the next instruction's Dest value to be (Src + Dest) & $1FF (for syntax 1), or to Dest[8:0] (for syntax 2).

In syntax 1, Src consists of two 9-bit fields; a base value (Src[8:0]) and a signed auto-indexer (Src[17:9]).

* The base represents a starting point. ALTD adds the *offset* (Dest[8:0]) to the *base* (Src[8:0]) to determine the next instruction's Dest value.
* At the end of ALTD execution, the optional auto-indexer value (usually 0, 1, or -1) is added to the offset (Dest) for a future ALTD+instruction iteration.

In syntax 2, Dest serves as the full value— it is used as-is for the next instruction's substitute Dest value.

**Notes:**

* The instruction following ALTD is shielded from interrupt
* ALTD alters the next instruction regardless of its kind
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## ALTGB

Alter get byte

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Alter subsequent GETBYTE / ROLBYTE instruction.

ALTGB *Dest*, {#}*Src*

ALTGB *Dest*

**Result:** The next instruction's pipelined Src and Num fields are altered to be (Src + Dest[10:2]) & $1FF, or just Dest[10:2] for syntax 2, and Dest[1:0], respectively.

* Dest is the register whose 11-bit value is the *index*, or the full byte address, for the GETBYTE / ROLBYTE instruction to read.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value contains a *base* long address (Src[8:0]; added to *index* (Dest[10:2]) for GETBYTE / ROLBYTE) and also an optional auto-indexer value (Src[17:9]; added to Dest at end of execution).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001011 | 01I | DDDDDDDDD | SSSSSSSSS | D1 | — | — | 2 |
| EEEE | 1001011 | 011 | DDDDDDDDD | 000000000 | D1 | — | — | 2 |

1 Dest is post-adjusted by the auto-indexer value; the sign-extended Src[17:9]. In syntax 2, the auto-indexer value is 0.

**Explanation:**

ALTGB should be followed by GETBYTE or ROLBYTE— it modifies the GETBYTE / ROLBYTE instruction's Src and Num values, enabling code to iterate through multiple bytes of data across a range of Reg RAM. GETBYTE / ROLBYTE's Src value is changed to (Src + Dest[10:2]) & $1FF (for syntax 1), or to Dest[10:2] (for syntax 2), and its Num value is changed to Dest[1:0].

Dest[10:2] corresponds to the target long register's 9-bit address and Dest[1:0] is the byte ID within it; values of 0–3 identify individual bytes, by position, in least-significant byte order. Iteratively executing ALTGB followed by GETBYTE or ROLBYTE, and each time incrementing ALTGB's 11-bit Dest value by one, effectively reads a stream of byte values from Reg RAM as if it were all made of byte-sized registers.

In syntax 1, Src consists of two 9-bit fields; a base address (Src[8:0]) and a signed auto-indexer (Src[17:9]).

* The base is the Reg RAM address where the series of bytes begins. ALTGB adds the *long* index (Dest[10:2]) to the base (Src[8:0]) to locate the register holding the target byte. The *byte* ID (Dest[1:0]) identifies the byte's position within that long register.
* At the end of ALTGB execution, the optional auto-indexer value (usually 0, 1, or -1) is added to the 11-bit index (Dest) for a future ALTGB+GETBYTE or ROLBYTE iteration.

In syntax 2, Dest serves as the full byte address— it's the same format as in syntax 1, but represents the target long's absolute address and its byte index instead of the long's relative index (to add to a base) and byte index.

**Notes:**

* The instruction following ALTGB is shielded from interrupt
* ALTGB alters the next instruction regardless of its kind— the intention is for it to be a GETBYTE / ROLBYTE
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## ALTGN

Alter get nibble

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Alter subsequent GETNIB / ROLNIB instruction.

ALTGN *Dest*, {#}*Src*

ALTGN *Dest*

**Result:** The next instruction's pipelined Src and Num values are altered to be (Src + Dest[11:3]) & $1FF, or just Dest[11:3] for syntax 2, and Dest[2:0], respectively.

* Dest is the register whose 12-bit value is the *index*, or the full nibble address, for the next GETNIB / ROLNIB instruction to read.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value contains a *base* long address (Src[8:0]; added to *index* (Dest[11:3]) for GETNIB / ROLNIB) and also an optional auto-indexer value (Src[17:9]; added to Dest at end of execution).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001010 | 11I | DDDDDDDDD | SSSSSSSSS | D1 | — | — | 2 |
| EEEE | 1001010 | 111 | DDDDDDDDD | 000000000 | D1 | — | — | 2 |

1 Dest is post-adjusted by the auto-indexer value; the sign-extended Src[17:9]. In syntax 2, the auto-indexer value is 0.

**Explanation:**

ALTGN should be followed by GETNIB or ROLNIB— it modifies the GETNIB / ROLNIB instruction's Src and Num values, enabling code to iterate through multiple nibbles of data across a range of Reg RAM. GETNIB / ROLNIB's Src value is changed to (Src + Dest[11:3]) & $1FF (for syntax 1), or to Dest[11:3] (for syntax 2), and its Num value is changed to Dest[2:0].

Dest[11:3] corresponds to the target long register's 9-bit address and Dest[2:0] is the nibble ID within it; values of 0–7 identify individual nibbles, by position, in least-significant nibble order. Iteratively executing ALTGN followed by GETNIB or ROLNIB, and each time incrementing ALTGN's 12-bit Dest value by one, effectively reads a stream of nibble values from Reg RAM as if it were all made of nibble-sized registers.

In syntax 1, Src consists of two 9-bit fields; a base address (Src[8:0]) and a signed auto-indexer (Src[17:9]).

* The base is the Reg RAM address where the series of nibbles begins. ALTGN adds the *long* index (Dest[11:3]) to the base (Src[8:0]) to locate the register holding the target nibble. The *nibble* ID (Dest[2:0]) identifies the nibble's position within that long register.
* At the end of ALTGN execution, the optional auto-indexer value (usually 0, 1, or -1) is added to the 12-bit index (Dest) for a future ALTGN+GETNIB or ROLNIB iteration.

In syntax 2, Dest serves as the full nibble address— it's the same format as in syntax 1, but represents the target long's absolute address and its nibble index instead of the long's relative index (to add to a base) and nibble index.

**Notes:**

* The instruction following ALTGN is shielded from interrupt
* ALTGN alters the next instruction regardless of its kind— the intention is for it to be a GETNIB / ROLNIB
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## ALTGW

Alter get word

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Alter subsequent GETWORD / ROLWORD instruction.

ALTGW *Dest*, {#}*Src*

ALTGW *Dest*

**Result:** The next instruction's pipelined Src and Num fields are altered to be (Src + Dest[9:1]) & $1FF, or just Dest[9:1] for syntax 2, and Dest[0], respectively.

* Dest is the register whose 10-bit value is the *index*, or the full word address for the GETWORD / ROLWORD instruction to read.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value contains a *base* long address (Src[8:0]; added to *index* (Dest[9:1]) for GETWORD / ROLWORD) and also an optional auto-indexer value (Src[17:9]; added to Dest at end of execution).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001011 | 11I | DDDDDDDDD | SSSSSSSSS | D1 | — | — | 2 |
| EEEE | 1001011 | 111 | DDDDDDDDD | 000000000 | D1 | — | — | 2 |

1 Dest is post-adjusted by the auto-indexer value; the sign-extended Src[17:9]. In syntax 2, the auto-indexer value is 0.

**Explanation:**

ALTGW should be followed by GETWORD or ROLWORD— it modifies the GETWORD / ROLWORD instruction's Src and Num values, enabling code to iterate through multiple words of data across a range of Reg RAM. GETWORD / ROLWORD's Src value is changed to (Src + Dest[9:1]) & $1FF (for syntax 1), or to Dest[9:1] (for syntax 2), and its Num value is changed to Dest[0].

Dest[9:1] corresponds to the target long register's 9-bit address and Dest[0] is the word ID within it; values of 0–1 identify individual words, by position, in least-significant word order. Iteratively executing ALTGW followed by GETWORD or ROLWORD, and each time incrementing ALTGW's 10-bit Dest value by one, effectively reads a stream of word values from Reg RAM as if it were all made of word-sized registers.

In syntax 1, Src consists of two 9-bit fields; a base address (Src[8:0]) and a signed auto-indexer (Src[17:9]).

* The base is the Reg RAM address where the series of words begins. ALTGW adds the *long* index (Dest[9:1]) to the base (Src[8:0]) to locate the register holding the target word. The *word* ID (Dest[0]) identifies the word's position within that long register.
* At the end of ALTGW execution, the optional auto-indexer value (usually 0, 1, or -1) is added to the 10-bit index (Dest) for a future ALTGW+GETWORD or ROLWORD iteration.

In syntax 2, Dest serves as the full word address— it's the same format as in syntax 1, but represents the target long's absolute address and its word index instead of the long's relative index (to add to a base) and word index.

**Notes:**

* The instruction following ALTGW is shielded from interrupt
* ALTGW alters the next instruction regardless of its kind— the intention is for it to be a GETWORD / ROLWORD
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## 

## ALTI

Alter instruction

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Substitute next instruction's field values from template, per configuration.

ALTI *Dest*, {#}*Src*

ALTI *Dest*

**Result:** The next instruction's pipelined field value values are substituted from the Dest template, and Dest is modified per Src configuration (syntax 1), or the entire Dest opcode (instruction) is executed in place of the next instruction (syntax 2).

* Dest is the register whose value contains one or more of the next instruction's field substitutes or an entire 32-bit opcode for full substitution.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value describes the substitutions and Dest modifications to perform.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001101 | 00I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1001101 | 001 | DDDDDDDDD | 101100100 | — | — | — | 2 |

**Related:** [SETD](#_jw7izhflytke), [SETS](#_94hz48griij7), [SETR](#_brza01jtowzy), [ALTD](#_6iq3v0fpyc3a), [ALTS](#_f2hcvwgutfa0), and [ALTR](#_3cn9gj4kxitq)

**Explanation:**

ALTI substitutes fields from Dest for one or more of the next instruction's pipelined Dest, Src, Result, Instr, FX, and/or Cond values, and ALTI's Dest is then modified per Src configuration (syntax 1), or the entire Dest opcode (instruction) is executed in place of the next instruction (syntax 2).

The Dest register contains the ALTI template; a 32-bit value with the following format similar to an opcode:

| **Bits** | 31:28 (4 bits) | 27:19 (9 bits) | 18 | 17:9 (9 bits) | 8:0 (9 bits) |
| --- | --- | --- | --- | --- | --- |
| **Field Description** | Condition Field | Result Field | Indirect "I" Field | Dest "D" Field | Source "S" Field |

In syntax 1, Src consists of the following six 3-bit fields that describe field substitution and Dest modification.

%rrr\_ddd\_sss\_RRR\_DDD\_SSS

* %rrr = Result field increment/decrement mask size
* %ddd = Dest field increment/decrement mask size
* %sss = Src field increment/decrement mask size
* %RRR = Result/Instruction field control
* %DDD = Dest field control
* %SSS = Src field control

| Field Increment/Decrement Mask Size | |
| --- | --- |
| **%rrr**  **%ddd**  **%sss** | **Mask Selection** |
| 000 | Unlimited Increment/Decrement of 9 bits (default, full 512-register Reg RAM span) |
| 001 | Limit Increment/Decrement to 8 LSBs (256-register Reg RAM looped buffer) |
| 010 | Limit Increment/Decrement to 7 LSBs (128-register Reg RAM looped buffer) |
| 011 | Limit Increment/Decrement to 6 LSBs (64-register Reg RAM looped buffer) |
| 100 | Limit Increment/Decrement to 5 LSBs (32-register Reg RAM looped buffer) |
| 101 | Limit Increment/Decrement to 4 LSBs (16-register Reg RAM looped buffer) |
| 110 | Limit Increment/Decrement to 3 LSBs (8-register Reg RAM looped buffer) |
| 111 | Limit Increment/Decrement to 2 LSBs (4-register Reg RAM looped buffer) |

| Result/Instruction Field Control | | | |
| --- | --- | --- | --- |
| **%RRR** | **Substitution** | **Dest[27:19] Adjustment** | **Notes** |
| 000 | None | None |  |
| 001 | None | None | Cancel next instruction's Result write |
| 010 | None | Decrements per %rrr |  |
| 011 | None | Increments per %rrr |  |
| 100 | Dest[27:19] ⇨ next instruction's Result field | None |  |
| 101 | Dest[31:18] ⇨ next instruction's [31:18] | None | Cond+Instr+Fx substitution1, 2 |
| 110 | Dest[27:19] ⇨ next instruction's Result field | Decrements per %rrr |  |
| 111 | Dest[27:19] ⇨ next instruction's Result field | Increments per %rrr |  |

1 Substitutes for next instruction's conditional execution mode, instruction class, and/or effects (WC/WZ, I, etc.)

2 Can use with %DDD=1xx and %SSS=1xx for full instruction substitution; i.e. execute Dest opcode in place of next instruction

| Dest Field Control | | | |
| --- | --- | --- | --- |
| **%DDD** | **Substitution** | **Dest[17:9] Adjustment** | **Notes** |
| 000 | None | None |  |
| 001 | None | None | Same as %000 |
| 010 | None | Decrements per %ddd |  |
| 011 | None | Increments per %ddd |  |
| 100 | Dest[17:9] ⇨ next instruction's Dest field | None | Execute Dest w/ RRR = %101, SSS = %1xx |
| 101 | Dest[17:9] ⇨ next instruction's Dest field | None | Execute Dest w/ RRR = %101, SSS = %1xx |
| 110 | Dest[17:9] ⇨ next instruction's Dest field | Decrements per %ddd | Execute Dest w/ RRR = %101, SSS = %1xx |
| 111 | Dest[17:9] ⇨ next instruction's Dest field | Increments per %ddd | Execute Dest w/ RRR = %101, SSS = %1xx |

| Src Field Control | | | |
| --- | --- | --- | --- |
| **%SSS** | **Substitution** | **Src[8:0] Adjustment** | **Notes** |
| 000 | None | None |  |
| 001 | None | None | Same as %000 |
| 010 | None | Decrements per %sss |  |
| 011 | None | Increments per %sss |  |
| 100 | Dest[8:0] ⇨ next instruction's Src field | None | Execute Dest w/ RRR = %101, DDD = %1xx |
| 101 | Dest[8:0] ⇨ next instruction's Src field | None | Execute Dest w/ RRR = %101, DDD = %1xx |
| 110 | Dest[8:0] ⇨ next instruction's Src field | Decrements per %sss | Execute Dest w/ RRR = %101, DDD = %1xx |
| 111 | Dest[8:0] ⇨ next instruction's Src field | Increments per %sss | Execute Dest w/ RRR = %101, DDD = %1xx |

In syntax 2, Dest serves as the full opcode value— it is executed as-is in place of the next instruction and Dest remains unaltered afterward.

**Examples:**

ALTI ptrs,#%111\_111 'set next D and S fields, increment ptrs[17:9] and ptrs[8:0]

ADD 0,0 'add registers

ALTI inst 'execute inst; same as ALTI inst, #%101\_100\_100

NOP 'NOP becomes inst

**Notes:**

* The instruction following ALTI is shielded from interrupt
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## ALTR

Alter result

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Alter Result register address of next instruction.

ALTR *Dest*, {#}*Src*

ALTR *Dest*

**Result:** The next instruction's pipelined Result address (Dest address by default) is altered to be (Src + Dest) & $1FF, or just Dest[8:0] in syntax 2.

* Dest is the register whose 9-bit value is the *offset*, or the full value, for the next instruction to operate on.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value contains a *base* (Src[8:0]; added to *offset* (Dest) for the next instruction) and also an optional auto-indexer value (Src[17:9]; added to Dest at the end of execution).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001100 | 00I | DDDDDDDDD | SSSSSSSSS | D1 | — | — | 2 |
| EEEE | 1001100 | 001 | DDDDDDDDD | 000000000 | D1 | — | — | 2 |

1 Dest is post-adjusted by the auto-indexer value; the sign-extended Src[17:9]. In syntax 2, the auto-indexer value is 0.

**Explanation:**

ALTR modifies the next instruction's Result address to be (Src + Dest) & $1FF (for syntax 1), or to Dest[8:0] (for syntax 2).

The Result address is the Dest address by default— it identifies where the *result* value from the instruction's execution is written at the end of execution. During execution, the pipeline holds an instruction's Dest address and the Result address as two separate entities, normally set to the same location. ALTR causes the next instruction's Result to redirect to a different address; changing an instruction from a destructive (operand overwriting) operation to a non-destructive (operand preserving) operation.

In syntax 1, Src consists of two 9-bit fields; a base value (Src[8:0]) and a signed auto-indexer (Src[17:9]).

* The base represents a starting point. ALTR adds the *offset* (Dest[8:0]) to the *base* (Src[8:0]) to determine the next instruction's Result address.
* At the end of ALTR execution, the optional auto-indexer value (usually 0, 1, or -1) is added to the offset (Dest) for a future ALTR+instruction iteration.

In syntax 2, Dest serves as the full value— it is used as-is for the next instruction's substitute Result address.

**Notes:**

* The instruction following ALTR is shielded from interrupt
* ALTR alters the next instruction regardless of its kind
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## ALTS

Alter source

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Alter S field of next instruction.

ALTS *Dest*, {#}*Src*

ALTS *Dest*

**Result:** The next instruction's pipelined Src value is altered to be (Src + Dest) & $1FF, or just Dest[8:0] in syntax 2.

* Dest is the register whose 9-bit value is the *offset*, or the full value, for the next instruction to operate on.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value contains a *base* (Src[8:0]; added to *offset* (Dest) for the next instruction) and also an optional auto-indexer value (Src[17:9]; added to Dest at the end of execution).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001100 | 10I | DDDDDDDDD | SSSSSSSSS | D1 | — | — | 2 |
| EEEE | 1001100 | 101 | DDDDDDDDD | 000000000 | D1 | — | — | 2 |

1 Dest is post-adjusted by the auto-indexer value; the sign-extended Src[17:9]. In syntax 2, the auto-indexer value is 0.

**Explanation:**

ALTS modifies the next instruction's Src value to be (Src + Dest) & $1FF (for syntax 1), or to Dest[8:0] (for syntax 2).

In syntax 1, Src consists of two 9-bit fields; a base value (Src[8:0]) and a signed auto-indexer (Src[17:9]).

* The base represents a starting point. ALTS adds the *offset* (Dest[8:0]) to the *base* (Src[8:0]) to determine the next instruction's Src value.
* At the end of ALTS execution, the optional auto-indexer value (usually 0, 1, or -1) is added to the offset (Dest) for a future ALTS+instruction iteration.

In syntax 2, Dest serves as the full value— it is used as-is for the next instruction's substitute Src value.

**Notes:**

* The instruction following ALTS is shielded from interrupt
* ALTS alters the next instruction regardless of its kind
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## ALTSB

Alter set byte

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Alter subsequent SETBYTE instruction.

ALTSB *Dest*, {#}*Src*

ALTSB *Dest*

**Result:** The next instruction's pipelined Dest and Num values are altered to be (Src + Dest[10:2]) & $1FF, or just Dest[10:2] for syntax 2, and Dest[1:0], respectively.

* Dest is the register whose 11-bit value is the *index*, or the full byte address, for the SETBYTE instruction to operate on.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value contains a *base* long address (Src[8:0]; added to *index* (Dest[10:2]) for SETBYTE) and also an optional auto-indexer value (Src[17:9]; added to Dest at end of execution).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001011 | 00I | DDDDDDDDD | SSSSSSSSS | D1 | — | — | 2 |
| EEEE | 1001011 | 001 | DDDDDDDDD | 000000000 | D1 | — | — | 2 |

1 Dest is post-adjusted by the auto-indexer value; the sign-extended Src[17:9]. In syntax 2, the auto-indexer value is 0.

**Explanation:**

ALTSB should be followed by SETBYTE— it modifies the SETBYTE instruction's Dest and Num values, enabling code to iterate through multiple bytes of data across a range of Reg RAM. SETBYTE's Dest value is changed to (Src + Dest[10:2]) & $1FF (for syntax 1), or to Dest[10:2] (for syntax 2), and its Num value is changed to Dest[1:0].

Dest[10:2] corresponds to the target long register's 9-bit address and Dest[1:0] is the byte ID within it; values of 0–3 identify individual bytes, by position, in least-significant byte order. Iteratively executing ALTSB followed by SETBYTE, and each time incrementing ALTSB's 11-bit Dest value by one, effectively writes a stream of byte values to Reg RAM as if it were all made of byte-sized registers.

In syntax 1, Src consists of two 9-bit fields; a base address (Src[8:0]) and a signed auto-indexer (Src[17:9]).

* The base is the Reg RAM address where the series of bytes begins. ALTSB adds the *long* index (Dest[10:2]) to the base (Src[8:0]) to locate the register holding the target byte. The *byte* ID (Dest[1:0]) identifies the byte's position within that long register.
* At the end of ALTSB execution, the optional auto-indexer value (usually 0, 1, or -1) is added to the 11-bit index (Dest) for a future ALTSB+SETBYTE iteration.

In syntax 2, Dest serves as the full byte address— it's the same format as in syntax 1, but represents the target long's absolute address and its byte index instead of the long's relative index (to add to a base) and byte index.

**Notes:**

* The instruction following ALTSB is shielded from interrupt
* ALTSB alters the next instruction regardless of its kind— the intention is for it to be a SETBYTE
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## ALTSN

Alter set nibble

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Alter subsequent SETNIB instruction.

ALTSN *Dest*, {#}*Src*

ALTSN *Dest*

**Result:** The next instruction's pipelined Dest and Num values are altered to be (Src + Dest[11:3]) & $1FF, or just Dest[11:3] for syntax 2, and Dest[2:0], respectively.

* Dest is the register whose 12-bit value is the *index*, or the full nibble address, for the SETNIB instruction to operate on.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value contains a *base* long address (Src[8:0]; added to *index* (Dest[11:3]) for SETNIB) and also an optional auto-indexer value (Src[17:9]; added to Dest at the end of execution).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001010 | 10I | DDDDDDDDD | SSSSSSSSS | D1 | — | — | 2 |
| EEEE | 1001010 | 101 | DDDDDDDDD | 000000000 | D1 | — | — | 2 |

1 Dest is post-adjusted by the auto-indexer value; the sign-extended Src[17:9]. In syntax 2, the auto-indexer value is 0.

**Explanation:**

ALTSN should be followed by SETNIB— it modifies the SETNIB instruction's Dest and Num values, enabling code to iterate through multiple nibbles of data across a range of Reg RAM. SETNIB's Dest value is changed to (Src + Dest[11:3]) & $1FF (for syntax 1), or to Dest[11:3] (for syntax 2), and its Num value is changed to Dest[2:0].

Dest[11:3] corresponds to the target long register's 9-bit address and Dest[2:0] is the nibble ID within it; values of 0–7 identify individual nibbles, by position, in least-significant nibble order. Iteratively executing ALTSN followed by SETNIB, and each time incrementing ALTSN's 12-bit Dest value by one, effectively writes a stream of nibble values to Reg RAM as if it were all made of nibble-sized registers.

In syntax 1, Src consists of two 9-bit fields; a base address (Src[8:0]) and a signed auto-indexer (Src[17:9]).

* The base is the Reg RAM address where the series of nibbles begins. ALTSN adds the *long* index (Dest[11:3]) to the base (Src[8:0]) to locate the register holding the target nibble. The *nibble* ID (Dest[2:0]) identifies the nibble's position within that long register.
* At the end of ALTSN execution, the optional auto-indexer value (usually 0, 1, or -1) is added to the 12-bit index (Dest) for a future ALTSN+SETNIB iteration.

In syntax 2, Dest serves as the full nibble address— it's the same format as in syntax 1, but represents the target long's absolute address and its nibble index instead of the long's relative index (to add to a base) and nibble index.

**Notes:**

* The instruction following ALTSN is shielded from interrupt
* ALTSN alters the next instruction regardless of its kind— the intention is for it to be a SETNIB
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## ALTSW

Alter set word

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Alter subsequent SETWORD instruction.

ALTSW *Dest*, {#}*Src*

ALTSW *Dest*

**Result:** The next instruction's pipelined Dest and Num fields are altered to be (Src + Dest[9:1]) & $1FF, or just Dest[9:1] for syntax 2, and Dest[0], respectively.

* Dest is the register whose 10-bit value is the *index*, or the full word address, for the SETWORD instruction to operate on.
* Src is an optional register, 9-bit literal, or 18-bit augmented literal whose value contains a *base* long address (Src[8:0]; added to *index* (Dest[9:1]) for SETWORD) and also an optional auto-indexer value (Src[17:9]; added to Dest at end of execution).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001011 | 10I | DDDDDDDDD | SSSSSSSSS | D1 | — | — | 2 |
| EEEE | 1001011 | 101 | DDDDDDDDD | 000000000 | D1 | — | — | 2 |

1 Dest is post-adjusted by the auto-indexer value; the sign-extended Src[17:9]. In syntax 2, the auto-indexer value is 0.

**Explanation:**

ALTSW should be followed by SETWORD— it modifies the SETWORD instruction's Dest and Num values, enabling code to iterate through multiple words of data across a range of Reg RAM. SETWORD's Dest value is changed to (Src + Dest[9:1]) & $1FF (for syntax 1), or to Dest[9:1] (for syntax 2), and its Num value is changed to Dest[0].

Dest[9:1] corresponds to the target long register's 9-bit address and Dest[0] is the word ID within it; values of 0–1 identify individual words, by position, in least-significant word order. Iteratively executing ALTSW followed by SETWORD, and each time incrementing ALTSW's 10-bit Dest value by one, effectively writes a stream of word values to Reg RAM as if it were all made of word-sized registers.

In syntax 1, Src consists of two 9-bit fields; a base address (Src[8:0]) and a signed auto-indexer (Src[17:9]).

* The base is the Reg RAM address where the series of words begins. ALTSW adds the *long* index (Dest[9:1]) to the base (Src[8:0]) to locate the register holding the target word. The *word* ID (Dest[0]) identifies the word's position within that long register.
* At the end of ALTSW execution, the optional auto-indexer value (usually 0, 1, or -1) is added to the 10-bit index (Dest) for a future ALTSW+SETWORD iteration.

In syntax 2, Dest serves as the full word address— it's the same format as in syntax 1, but represents the target long's absolute address and its word index instead of the long's relative index (to add to a base) and word index.

**Notes:**

* The instruction following ALTSW is shielded from interrupt
* ALTSW alters the next instruction regardless of its kind— the intention is for it to be a SETWORD
* Field value modification occurs in the instruction pipeline only; code is not altered, values do not persist
* SETQ / SETQ2 does not affect ALTx instructions— the Q value passes through to the next instruction

## AND / ANDN

And not

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Bitwise AND a value with another, or with the NOT of another.

AND *Dest*, {#}*Src* {WC|WZ|WCZ}

ANDN *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Dest AND Src (or Dest AND !Src) is stored in Dest and flags are optionally updated with *parity* and *zero* status.

* Dest is the register containing the value to bitwise AND with Src (or with !Src) and is the destination in which to write the result.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value (or inverse value) will be bitwise ANDed into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0101000 | CZI | DDDDDDDDD | SSSSSSSSS | D | Parity of Result | Result = 0 | 2 |
| EEEE | 0101001 | CZI | DDDDDDDDD | SSSSSSSSS | D | Parity of Result | Result = 0 | 2 |

**Explanation:**

AND or ANDN performs a bitwise AND of the value in Src (or !Src) into that of Dest.

If the WC or WCZ effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits, or is cleared (0) if it contains an even number of high bits.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest AND Src (or Dest AND !Src) equals zero, or is cleared (0) if it is non-zero.

## AUGD

Augment destination

[**Augmentation Instruction**](#_f03he7b32t26) **-** Augment next literal Dest to 32-bits.

AUGD #*Dest*

**Result:** The 23-bit value formed from Dest is queued to prefix the next literal Dest occurrence (#Dest) to form a 32-bit literal for that instruction; interrupts are also temporarily disabled..

* Dest is a 32-bit literal whose upper 23 bits are prepended to the next literal Dest occurrence.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 11111DD | DDD | DDDDDDDDD | DDDDDDDDD | Hidden D Queue | — | — | 2 |

**Related:** [AUGS](#_b6omkiqrkged)

**Explanation:**

AUGD is an assistant instruction to aid with literal values that exceed 9 bits. Most PASM2 instructions have 9 bits available for literal Dest values; enough for many uses, but not all. AUGD augments the next occurrence of a literal Dest value to be a full 32-bits. When the instruction with the *soon-to-be-augmented* literal is later executed, the cog uses the lower 9 bits encoded in the instruction's Dest field and prepends AUGD's 23 bits to it.

**Notes:**

* All instructions following AUGD are shielded from interrupt until after the instruction with the newly-augmented literal Dest value is executed
* Dest value augmentation occurs in the instruction pipeline only; code is not altered, value does not persist
* SETQ / SETQ2 does not affect AUGD— the Q value passes through to the next instruction

Tip: Though AUGD may be manually entered wherever needed, the Parallax P2 compiler supports a convenient way to use this feature. In the target instruction's Dest field, use "##" followed by the desired 32-bit literal (instead of "#" followed by a 9-bit literal); the compiler will automatically invoke AUGD immediately before. When counting clock cycles, make sure to account for 2 extra clock cycles for instructions containing ## augmented literals.

**Example**

Both samples below use greater-than 9-bit literals (via AUGD functionality). The first is more convenient and readable than using the AUGD instruction directly.

PUSHA ##$ABCDEF '2+3..10 cycles (in Cog RAM)

QMUL ##8192, #4 '2+2..9 cycles

is the same as:

AUGD #$ABCDEF '2 cycles

PUSHA #$ABCDEF & $1FF '3..10 cycles (in Cog RAM)

AUGD #8192 '2 cycles

QMUL #8192 & $1FF, #4 '2..9 cycles

## AUGS

Augment source

[**Augmentation Instruction**](#_f03he7b32t26) **-** Augment next literal Src to 32-bits.

AUGS #*Src*

**Result:** The 23-bit value formed from Src is queued to prefix the next literal Src occurrence (#Src) to form a 32-bit literal for that instruction; interrupts are also temporarily disabled.

* Src is a 32-bit literal whose upper 23 bits are prepended to the next literal Src occurrence.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 11110SS | SSS | SSSSSSSSS | SSSSSSSSS | Hidden S Queue | — | — | 2 |

**Related:** [AUGD](#_qxo6681ulhaj)

**Explanation:**

AUGS is an assistant instruction to aid with literal values that exceed 9 bits. Most PASM2 instructions have 9 bits available for literal Src values; enough for many uses, but not all. AUGS augments the next occurrence of a literal Src value to be a full 32-bits. When the instruction with the *soon-to-be-augmented* literal is later executed, the cog uses the lower 9 bits encoded in the instruction's Src field and prepends AUGS's 23 bits to it.

**Notes:**

* All instructions following AUGS are shielded from interrupt until after the instruction with the newly-augmented literal Src value is executed
* Src value augmentation occurs in the instruction pipeline only; code is not altered, value does not persist
* SETQ / SETQ2 does not affect AUGS— the Q value passes through to the next instruction

Tip: Though AUGS may be manually entered wherever needed, the Parallax P2 compiler supports a convenient way to use this feature. In the target instruction's Src field, use "##" followed by the desired 32-bit literal (instead of "#" followed by a 9-bit literal); the compiler will automatically invoke AUGS immediately before. When counting clock cycles, make sure to account for 2 extra clock cycles for instructions containing ## augmented literals.

**Example**

Both samples below use greater-than 9-bit literals (via AUGS functionality). The first is more convenient and readable than using the AUGS instruction directly.

AND address, ##$FFFFF '2+2 cycles

DJNZ idx, ##far\_away '2+(2 or 4) cycles (in Cog RAM)

is the same as:

AUGS #$FFFFF '2 cycles

AND address, #$FFFFF & $1FF '2 cycles

AUGS #far\_away '2 cycles

DJNZ idx, #far\_away & $1FF '2 or 4 cycles (in Cog RAM)

## BITC / BITNC

Bit C or not C

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set bit(s) low/high according to C or !C.

BITC *Dest*, {#}*Src* {WCZ}

BITNC *Dest*, {#}*Src* {WCZ}

**Result:** Dest bit(s) described by Src are set to C or !C; the rest are left as-is.

* Dest is the register whose value will have one or more bits set to C or !C.
* Src is a register, 9-bit literal, or 10-bit augmented literal whose value identifies the bit(s) to modify.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0100010 | CZI | DDDDDDDDD | SSSSSSSSS | D | Original D base bit | Original D base bit | 2 |
| EEEE | 0100011 | CZI | DDDDDDDDD | SSSSSSSSS | D | Original D base bit | Original D base bit | 2 |

**Explanation:**

BITC or BITNC alters the Dest bit(s) designated by Src to equal the state, or inverse state, of the C flag. All other bits are left unchanged.

Src[4:0] indicates the bit number (0—31). For a range of bits, Src[4:0] indicates the base bit number (0—31) and Src[9:5] indicates how many contiguous bits beyond the base should be affected (1–31).

A 9-bit literal Src is enough to express the base bit (Src[4:0]) and a range of up to 16 contiguous bits (Src[8:5]). If needed, use the augmented literal feature (##Src) to augment Src to a 10-bit literal value— this inserts an AUGS instruction prior.

When Src is a register, the register's value bits [9:0] are used as-is to form the 10-bit ID range, unless a SETQ instruction immediately precedes the BITC / BITNC instruction; substituting SETQ's Dest[4:0] in place of value bits[9:5], for BITC / BITNC's use.

If the WCZ effect is specified, the C and Z flags are updated to the original state of Dest's base bit, identified by Src.

## BITH / BITL

Bit high or low

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set bit(s) high (1) or low (0).

BITH *Dest*, {#}*Src* {WCZ}

BITL *Dest*, {#}*Src* {WCZ}

**Result:** Dest bit(s) described by Src are set high (1) or low (0); the rest are left as-is.

* Dest is the register whose value will have one or more bits set high or low.
* Src is a register, 9-bit literal, or 10-bit augmented literal whose value identifies the bit(s) to modify.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0100001 | CZI | DDDDDDDDD | SSSSSSSSS | D | Original D base bit | Original D base bit | 2 |
| EEEE | 0100000 | CZI | DDDDDDDDD | SSSSSSSSS | D | Original D base bit | Original D base bit | 2 |

**Explanation:**

BITH or BITL alters the Dest bit(s) designated by Src to be high (1) or low (0). All other bits are left unchanged.

Src[4:0] indicates the bit number (0—31). For a range of bits, Src[4:0] indicates the base bit number (0—31) and Src[9:5] indicates how many contiguous bits beyond the base should be affected (1–31).

A 9-bit literal Src is enough to express the base bit (Src[4:0]) and a range of up to 16 contiguous bits (Src[8:5]). If needed, use the augmented literal feature (##Src) to augment Src to a 10-bit literal value— this inserts an AUGS instruction prior.

When Src is a register, the register's value bits [9:0] are used as-is to form the 10-bit ID range, unless a SETQ instruction immediately precedes the BITH / BITL instruction; substituting SETQ's Dest[4:0] in place of value bits[9:5], for BITH / BITL's use.

If the WCZ effect is specified, the C and Z flags are updated to the original state of Dest's base bit, identified by Src.

## BITNOT

Bit not

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Toggle bit(s) to the opposite state.

BITNOT *Dest*, {#}*Src* {WCZ}

**Result:** Dest bit(s) described by Src are toggled to their opposite state(s); the rest are left as-is.

* Dest is the register whose value will have one or more bits toggled.
* Src is a register, 9-bit literal, or 10-bit augmented literal whose value identifies the bit(s) to modify.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0100111 | CZI | DDDDDDDDD | SSSSSSSSS | D | Original D base bit | Original D base bit | 2 |

**Explanation:**

BITNOT alters the Dest bit(s) designated by Src to their inverse state. All other bits are left unchanged.

Src[4:0] indicates the bit number (0—31). For a range of bits, Src[4:0] indicates the base bit number (0—31) and Src[9:5] indicates how many contiguous bits beyond the base should be affected (1–31).

A 9-bit literal Src is enough to express the base bit (Src[4:0]) and a range of up to 16 contiguous bits (Src[8:5]). If needed, use the augmented literal feature (##Src) to augment Src to a 10-bit literal value— this inserts an AUGS instruction prior.

When Src is a register, the register's value bits [9:0] are used as-is to form the 10-bit ID range, unless a SETQ instruction immediately precedes the BITNOT instruction; substituting SETQ's Dest[4:0] in place of value bits[9:5], for BITNOT's use.

If the WCZ effect is specified, the C and Z flags are updated to the original state of Dest's base bit, identified by Src.

## BITRND

Bit random

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set bit(s) random low/high.

BITRND *Dest*, {#}*Src* {WCZ}

**Result:** Dest bit(s) described by Src are each set randomly low or high; the rest are left as-is.

* Dest is the register whose value will have one or more bits set randomly low or high.
* Src is a register, 9-bit literal, or 10-bit augmented literal whose value identifies the bit(s) to modify.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0100110 | CZI | DDDDDDDDD | SSSSSSSSS | D | Original D base bit | Original D base bit | 2 |

**Explanation:**

BITRND alters the Dest bit(s) designated by Src to each be an independent random low and high value, based on bit(s) from the Xoroshiro128\*\* PRNG. All other bits are left unchanged.

Src[4:0] indicates the bit number (0—31). For a range of bits, Src[4:0] indicates the base bit number (0—31) and Src[9:5] indicates how many contiguous bits beyond the base should be affected (1–31).

A 9-bit literal Src is enough to express the base bit (Src[4:0]) and a range of up to 16 contiguous bits (Src[8:5]). If needed, use the augmented literal feature (##Src) to augment Src to a 10-bit literal value— this inserts an AUGS instruction prior.

When Src is a register, the register's value bits [9:0] are used as-is to form the 10-bit ID range, unless a SETQ instruction immediately precedes the BITRND instruction; substituting SETQ's Dest[4:0] in place of value bits[9:5], for BITRND's use.

If the WCZ effect is specified, the C and Z flags are updated to the original state of Dest's base bit, identified by Src.

## BITZ / BITNZ

Bit Z or not Z

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set bit(s) low/high according to Z or !Z.

BITZ *Dest*, {#}*Src* {WCZ}

BITNZ *Dest*, {#}*Src* {WCZ}

**Result:** Dest bit(s) described by Src are set to Z or !Z; the rest are left as-is.

* Dest is the register whose value will have one or more bits set to Z or !Z.
* Src is a register, 9-bit literal, or 10-bit augmented literal whose value identifies the bit(s) to modify.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0100100 | CZI | DDDDDDDDD | SSSSSSSSS | D | Original D base bit | Original D base bit | 2 |
| EEEE | 0100101 | CZI | DDDDDDDDD | SSSSSSSSS | D | Original D base bit | Original D base bit | 2 |

**Explanation:**

BITZ or BITNZ alters the Dest bit(s) designated by Src to equal the state, or inverse state, of the Z flag. All other bits are left unchanged.

Src[4:0] indicates the bit number (0—31). For a range of bits, Src[4:0] indicates the base bit number (0—31) and Src[9:5] indicates how many contiguous bits beyond the base should be affected (1–31).

A 9-bit literal Src is enough to express the base bit (Src[4:0]) and a range of up to 16 contiguous bits (Src[8:5]). If needed, use the augmented literal feature (##Src) to augment Src to a 10-bit literal value— this inserts an AUGS instruction prior.

When Src is a register, the register's value bits [9:0] are natrally used as-is to form the 10-bit ID range, unless a SETQ instruction immediately precedes the BITZ / BITNZ instruction; substituting SETQ's Dest[4:0] in place of value bits[9:5], for BITZ / BITNZ's use.

If the WCZ effect is specified, the C and Z flags are updated to the original state of Dest's base bit, identified by Src.

## BMASK

Bit mask

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Get 1..32-bit mask into Dest.

BMASK *Dest*, {#}*Src*

BMASK *Dest*

**Result:** Bit mask of size Src+1, or Dest+1 (1—32 bits) is stored into Dest.

* Dest is the register in which to store the generated bit mask and optionally begins by containing the 5-bit mask size it is requesting (syntax 2).
* Src is an optional register or 5-bit literal whose value is the size of the bit mask to generate.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001110 | 01I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1001110 | 010 | DDDDDDDDD | DDDDDDDDD | D | — | — | 2 |

**Explanation:**

BMASK generates an LSB-justified bit mask (all ones) of Src+1 or Dest+1 length and stores it in Dest. The size value, whether specified by Src or Dest, is in the range 0—31 to generate 1 to 32 bits of bit mask.

In effect, Dest becomes (%10 << size) - 1 via the BMASK instruction.

* A size value of 0 generates a bit mask of %00000000\_00000000\_00000000\_00000001.
* A size value of 5 generates a bit mask of %00000000\_00000000\_00000000\_00111111.
* A size value of 15 generates a bit mask of %00000000\_00000000\_11111111\_11111111.

A bit mask is often useful in bitwise operations (AND, OR, XOR) to filter out or affect special groups of bits.

## BRK

Break

[**Interrupt Instruction**](#_40bj2qk54rdh) **-** Trigger breakpoint in current cog.

BRK {#}*Dest*

**Result:** If debug interrupts are enabled, a debug interrupt is triggered in the current cog and Dest's value becomes the debug code or the next debug condition.

* Dest is the register, 9-bit literal, or 32-bit augmented literal whose value becomes the debug code or condition depending on the state of execution (outside or inside of a Debug ISR).

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | 00L | DDDDDDDDD | 000110110 | Code/Cond.1 | — | — | 2 |

1 Cog's internal Debug Code or Condition register is written depending on the state of execution.

**Explanation:**

BRK triggers a breakpoint in the current cog and either defines a breakpoint *code* or the next breakpoint condition(s). The cog must have debug interrupts enabled, and if BRK is to be executed within the normal program (outside the Debug ISR), the "BRK instruction" interrupt must first be enabled from within a prior Debug ISR (interrupt service routine).

During normal program execution, the BRK instruction is used to generate a debug interrupt with an 8-bit code (from D[7:0]) which can be read within the Debug ISR using a GETBRK instruction.

During a Debug ISR, the BRK instruction is used instead to establish the next debug interrupt condition(s) and to select INA/INB, instead of the IJMP0/IRET0 registers exposed during the ISR, so that the pins' inputs states may be read.

The format of Dest for Debug ISR use is %AAAAAAAAAAAAAAAAAAAA\_BCDEFGHIJKLM

A: 20-bit breakpoint address or 4-bit event code

B: 1 = map INA/INB normally, 0 = map IJMP0/IRET0 at INA/INB (default during ISR). If Debug ISR sets B to 1, it must reset it to 0 before exiting the Debug ISR so the RETI0 instruction sees IJMP0 and IRET0.

C: 1 = enable interrupt on breakpoint address match

D: 1 = enable interrupt on event %eeee

E: 1 = enable interrupt on asynchronous breakpoint (via COGBRK from another cog)

F: 1 = enable interrupt on INT3 ISR entry

G: 1 = enable interrupt on INT2 ISR entry

H: 1 = enable interrupt on INT1 ISR entry

I: 1 = enable interrupt on BRK instruction

J: 1 = enable interrupt on INT3 ISR code (single step)

K: 1 = enable interrupt on INT2 ISR code (single step)

L: 1 = enable interrupt on INT1 ISR code (single step)

M: 1 = enable interrupt on non-ISR code (single step)

Upon Debug ISR entry, bits B through M are cleared (0). If a subsequent debug interrupt is desired, a BRK instruction must be executed before exiting the Debug ISR in order to establish the next breakpoint condition(s).

## CALL

Call

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Call a subroutine; store return context on the stack.

CALL #{\}*Addr*

CALL *Dest* {WC|WZ|WCZ}

**Result:** Push current C and Z flags and address of the next instruction onto the hardware stack, set PC to new relative or absolute address, and optionally update C and/or Z to new state.

* Addr is a symbolic reference to the target subroutine; the location to set PC to. Relative addressing is the default; use '\' to force absolute addressing.
* Dest is a register containing the 20-bit absolute address to set PC to and optional new C and Z states.
* WC, WZ, or WCZ are optional effects to update the flags from Dest's upper bit states.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101101 | RAA | AAAAAAAAA | AAAAAAAAA | K1 and PC | — | — | 4 / 13–20 |
| EEEE | 1101011 | CZ0 | DDDDDDDDD | 000101101 | K1 and PC | D[31] | D[30] | 4 / 13–20 |

1 The current C, Z, and effective next PC values are pushed onto the stack (K) prior to replacing them with that of Addr or Dest.

**Explanation:**

CALL records the current state of the C and Z flags and the address of the next instruction (PC + 1 if Cog/LUT execution; PC + 4 if Hub execution) by pushing to the stack (K), potentially updates the C and Z flags with new given states, and jumps to the given address or offset. The routine at the new address should eventually execute a RET instruction, or an instruction with a \_RET\_ condition, to return to the recorded address (the instruction following the CALL) and optionally restore the C and Z flag state as it was prior.

In syntax 1, #Addr and #\Addr encodes the instruction with relative and absolute addressing, respectively. The relative form (the default) is vital for creating relocatable code. In either case, use symbolic references for Addr and the assembler will encode it properly. Examples: CALL #SendBit or CALL #\DebugStatus

In syntax 2, the format of the value at Dest is CZxxxxxx\_xxxxAAAA\_AAAAAAAA\_AAAAAAAA. C is the new C flag state, Z is the new Z flag state, A is the new 20-bit address to jump to, and x are don't-care bits. Syntax 2 effectively swaps the flags and PC with the value in the Dest register (and RET swaps them back), making it convenient for switching between two threads.

If the WC or WCZ effect is specified, the C flag is updated to match D[31], after its original state is recorded.

If the WZ or WCZ effect is specified, the Z flag is updated to match D[30], after its original state is recorded.

## CALLA / CALLB

Call A or B

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Call a subroutine; store return context in the Hub long at PTRA++ or PTRB++.

CALLA #{\}*Addr*

CALLA *Dest* {WC|WZ|WCZ}

CALLB #{\}*Addr*

CALLB *Dest* {WC|WZ|WCZ}

**Result:** Write current C and Z flags and address of the next instruction into the 4-byte Hub RAM location at PTRA or PTRB, increment pointer, set PC to new relative or absolute address, and optionally update C and/or Z to new state.

* Addr is a symbolic reference to the target subroutine; the location to set PC to. Relative addressing is the default; use '\' to force absolute addressing.
* Dest is a register containing the 20-bit absolute address to set PC to and optional new C and Z states.
* WC, WZ, or WCZ are optional effects to update the flags from Dest's upper bit states.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101110 | RAA | AAAAAAAAA | AAAAAAAAA | Hub1 and PC | — | — | 5–122 / 14–322 |
| EEEE | 1101011 | CZ0 | DDDDDDDDD | 000101110 | Hub1 and PC | D[31] | D[30] | 5–122 / 14–322 |
| EEEE | 1101111 | RAA | AAAAAAAAA | AAAAAAAAA | Hub1 and PC | — | — | 5–122 / 14–322 |
| EEEE | 1101011 | CZ0 | DDDDDDDDD | 000101111 | Hub1 and PC | D[31] | D[30] | 5–122 / 14–322 |

1 The current C, Z, and effective next PC values are written to the Hub RAM long (H) referenced by PTRA or PTRB prior to replacing them with that of Addr or Dest.

2 +1 clock cycle if target address isn't long-aligned in Hub RAM.

**Explanation:**

CALLA and CALLB records the current state of the C and Z flags and the address of the next instruction (PC + 1 if Cog/LUT execution; PC + 4 if Hub execution) by writing them to the Hub RAM long (H) referenced by PTRA or PTRB, increments PTRA or PTRB, potentially updates the C and Z flags with new given states, and jumps to the given address or offset. The routine at the new address should eventually execute a RETA or RETB instruction to return to the recorded address (the instruction following the CALLA or CALLB) and optionally restore the C and Z flag state as it was prior.

In syntax 1 and 3, #Addr and #\Addr encodes the instruction with relative and absolute addressing, respectively. The relative form (the default) is vital for creating relocatable code. In either case, use symbolic references for Addr and the assembler will encode it properly. Examples: CALLA #SendBit or CALLB #\DebugStatus

In syntax 2 and 4, the format of the value at Dest is CZxxxxxx\_xxxxAAAA\_AAAAAAAA\_AAAAAAAA. C is the new C flag state, Z is the new Z flag state, A is the new 20-bit address to jump to, and x are don't-care bits. Syntax 2 effectively swaps the flags and PC with the value in the Dest register (and RETA or RETB swaps them back), making it convenient for switching between two threads.

If the WC or WCZ effect is specified, the C flag is updated to match D[31], after its original state is recorded.

If the WZ or WCZ effect is specified, the Z flag is updated to match D[30], after its original state is recorded.

## CALLD

Call D

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Call a subroutine; store return context in PA/PB/PTRA/PTRB/D.

CALLD PA|PB|PTRA|PTRB, #{\}*Addr*

CALLD *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Write current C and Z flags and address of the next instruction into either PA, PB, PTRA, PTRB, or Dest, set PC to new relative or absolute address (in Addr or Src), and optionally update C and/or Z to new state.

* PA|PB|PTRA|PTRB is the special register to store the current C and Z flags and next address into.
* Addr is a symbolic reference to the target subroutine; the location to set PC to. Relative addressing is the default; use '\' to force absolute addressing.
* Dest is a register to write the current C and Z flags and the address of the next instruction into.
* Src is a register, 9-bit literal, or 32-bit augmented literal that contains the relative or absolute address to set PC to and optional new C and Z states. Use # for relative addressing; omit # for absolute addressing.
* WC, WZ, or WCZ are optional effects to update the flags from Src's upper bit states.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 11100WW | RAA | AAAAAAAAA | AAAAAAAAA | Pxxx1 and PC | — | — | 4 / 13–20 |
| EEEE | 1011001 | CZI | DDDDDDDDD | SSSSSSSSS | D1 and PC | S[31] | S[30] | 4 / 13–20 |

1 The current C, Z, and effective next PC values are written to PA, PB, PTRA, PTRB, or Dest prior to replacing them with that of Addr or Src.

**Explanation:**

CALLD records the current state of the C and Z flags and the address of the next instruction (PC + 1 if Cog/LUT execution; PC + 4 if Hub execution) by writing them to the PA, PB, PTRA, PTRB, or Dest register, potentially updates the C and Z flags with new given states, and jumps to the given address or offset. The routine at the new address should eventually execute another CALLD instruction to return to the recorded address (the instruction following the original CALLD) optionally restore the C and Z flag state as it was prior, and optionally prep for another CALLD. This instruction is typically used for the P2 DEBUG function.

In syntax 1, #Addr and #\Addr encodes the instruction with relative and absolute addressing, respectively. The relative form (the default) is vital for creating relocatable code. In either case, use symbolic references for Addr and the assembler will encode it properly. Examples: CALLD PA, #SendBit or CALLD PB, #\DebugStatus

In syntax 2, the format of the value at Src is CZxxxxxx\_xxxxAAAA\_AAAAAAAA\_AAAAAAAA. C is the new C flag state, Z is the new Z flag state, A is the new 20-bit address to jump to, and x are don't-care bits. If Src is a 9-bit literal (immediate), it will be sign-extended to 20 bits and used as a relative offset; giving a range of -256 to +255 instructions, relative to the instruction following the CALLD. When relative, PC is adjusted by signed(Src) if Cog/LUT execution, or by signed(Src\*4) if Hub execution.

If the WC or WCZ effect is specified, the C flag is updated to match S[31], after its original state is recorded.

If the WZ or WCZ effect is specified, the Z flag is updated to match S[30], after its original state is recorded.

## CALLPA / CALLPB

Call parameter A or B

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Call a subroutine; store return context on the stack and copy D into PA or PB.

CALLPA {#}*Dest*, {#}*Src*

CALLPB {#}*Dest*, {#}*Src*

**Result:** Push current C and Z flags and address of the next instruction onto the hardware stack, copy D to PA or PB, and set PC to new relative or absolute address.

* Dest is a register, 9-bit literal, or 32-bit augmented literal whose value is copied to PA or PB.
* Src is a register, 9-bit literal, or 32-bit augmented literal that contains the relative or absolute address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011010 | 0LI | DDDDDDDDD | SSSSSSSSS | K1, PA and PC | — | — | 4 / 13–20 |
| EEEE | 1011010 | 1LI | DDDDDDDDD | SSSSSSSSS | K1, PB and PC | — | — | 4 / 13–20 |

1 The current C, Z, and effective next PC values are pushed onto the stack (K) prior to updating PC with Src.

**Explanation:**

CALLPA and CALLPB records the current state of the C and Z flags and the address of the next instruction (PC + 1 if Cog/LUT execution; PC + 4 if Hub execution) by pushing to the stack (K), copies Dest to PA or PB, and jumps to the given address or offset in Src. The routine at the new address should eventually execute a RET instruction, or an instruction with a \_RET\_ condition, to return to the recorded address (the instruction following the CALLPA or CALLPB) and optionally restore the C and Z flag state as it was prior.

The Src can be absolute or relative; use #Addr for calling a range of -256 to +255 instructions, relative to the instruction following the CALLPA or CALLPA. When relative, PC is adjusted by signed(Src) if Cog/LUT execution, or by signed(Src\*4) if Hub execution.

## CMP

Compare

[**Math Instruction**](#_xnk9pb8j5sof) **-** Compare two unsigned values.

CMP *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Greater/lesser and equality status is optionally written to the C and Z flags.

* Dest is the register containing the value to compare with that of Src.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is compared to Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0010000 | CZI | DDDDDDDDD | SSSSSSSSS | — | Unsigned (D < S) | D = S | 2 |

**Related:** [CMPR](#_6j1a3848ozh3), [CMPX](#_3w81cwnriog1), [CMPS](#_nd6caax9v0zg), and [CMPSX](#_ccc7sdk782ug)

**Explanation:**

CMP compares the unsigned values of Dest and Src (by subtracting Src from Dest) and optionally setting the C and Z flags accordingly.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest is less than Src.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Dest equals Src.

To compare unsigned, multi-long values, use CMP followed by CMPX as described in [Comparing Two Multi-Long Values](#_kjzu1wz1qct1).

## CMPM

Compare most significant bit

[**Math Instruction**](#_xnk9pb8j5sof) **-** Compare two unsigned values, get MSB of difference.

CMPM *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Greater/lesser and equality status is optionally written to the C and Z flags.

* Dest is the register containing the value to compare with that of Src.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is compared to Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0010101 | CZI | DDDDDDDDD | SSSSSSSSS | — | Result[31] | D = S | 2 |

**Explanation:**

CMPM compares the unsigned values of Dest and Src (by subtracting Src from Dest) and optionally setting the C and Z flags accordingly.

If the WC or WCZ effect is specified, the C flag is updated to the MSB of (Dest - Src).

If the WZ or WCZ effect is specified, the Z flag is set (1) if Dest equals Src.

## CMPR

Compare reverse

[**Math Instruction**](#_xnk9pb8j5sof) **-** Compare two unsigned values (in reverse order to CMP).

CMPR *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Greater/lesser and equality status is optionally written to the C and Z flags.

* Dest is the register containing the value to compare with that of Src.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is compared to Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0010100 | CZI | DDDDDDDDD | SSSSSSSSS | — | Unsigned (S < D) | D = S | 2 |

**Related:** [CMP](#_jw7rhjdisro0)

**Explanation:**

CMP compares the unsigned values of Dest and Src (by subtracting Dest from Src) and optionally setting the C and Z flags accordingly.

If the WC or WCZ effect is specified, the C flag is set (1) if Src is less than Dest.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Dest equals Src.

To compare unsigned, multi-long values, use CMP (instead) followed by CMPX as described in [Comparing Two Multi-Long Values](#_kjzu1wz1qct1).

## CMPS

Compare signed

[**Math Instruction**](#_xnk9pb8j5sof) **-** Compare two signed values.

CMPS *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Greater/lesser and equality status is optionally written to the C and Z flags.

* Dest is the register containing the value to compare with that of Src.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is compared to Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0010010 | CZI | DDDDDDDDD | SSSSSSSSS | — | Signed (D < S) | D = S | 2 |

**Related:** [CMP](#_jw7rhjdisro0), [CMPX](#_3w81cwnriog1), and [CMPSX](#_ccc7sdk782ug)

**Explanation:**

CMPS compares the signed values of Dest and Src (by subtracting Src from Dest) and optionally setting the C and Z flags to indicate the comparison and operation results.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest is less than Src.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Dest equals Src.

To compare signed, multi-long values, use CMP (not CMPS) followed possibly by CMPX, and finally CMPSX as described in [Comparing Two Multi-Long Values](#_kjzu1wz1qct1).

## CMPSUB

Compare and subtract

[**Math Instruction**](#_xnk9pb8j5sof) **-** Compare two unsigned values and subtract the second if it is lesser or equal.

CMPSUB *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Dest is decremented by Src unless it is less than Src, and the comparison results are optionally written to the C and Z flags.

* Dest is the register containing the value to compare with Src and is the destination written to if a subtraction is performed.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is compared with and possibly subtracted from Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0010111 | CZI | DDDDDDDDD | SSSSSSSSS | D1 | Unsigned(D => S) | Result = 0 | 2 |

1 Dest is only written if a subtraction was performed on it.

**Explanation:**

CMPSUB compares the unsigned values of Dest and Src, and if Src is less than or equal to Dest then it is subtracted from Dest. Optionally, the C and Z flags are set to indicate the comparison and operation results.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest was greater than or equal to Src.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result equals 0.

## CMPSX

Compare signed, extended

[**Math Instruction**](#_xnk9pb8j5sof) **-** Compare two signed values plus carry flag.

CMPSX *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Greater/lesser and equality status is optionally written to the C and Z flags.

* Dest is the register containing the value to compare with that of Src.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is compared to Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0010011 | CZI | DDDDDDDDD | SSSSSSSSS | — | Signed (D < S+C) | Z AND (D = (S+C)) | 2 |

**Related:** [CMP](#_jw7rhjdisro0), [CMPX](#_3w81cwnriog1), and [CMPSX](#_ccc7sdk782ug)

**Explanation:**

CMPSX compares the signed values of Dest and Src plus C (by subtracting Src + C from Dest) and optionally setting the C and Z flags accordingly. The CMPSX instruction is used to perform signed multi-long comparisons, such as 64-bit comparisons.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest is less than Src + C (as multi-long values). Use WC or WCZ on preceding CMP and CMPX instructions for proper final C flag.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Z was previously set and the result of Dest - (Src + C) is zero, or it is cleared (0) if non-zero. Use WZ or WCZ on preceding SUB and SUBX instructions for proper final Z flag.

To compare signed, multi-long values, use CMP (not CMPS) followed possibly by CMPX, and finally CMPSX as described in [Comparing Two Multi-Long Values](#_kjzu1wz1qct1).

## CMPX

Compare extended

[**Math Instruction**](#_xnk9pb8j5sof) **-** Compare two unsigned values plus carry flag.

CMPX *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Greater/lesser and equality status is optionally written to the C and Z flags.

* Dest is a register containing the value to compare with that of Src plus C.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value plus C is compared to Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0010001 | CZI | DDDDDDDDD | SSSSSSSSS | — | Unsigned (D < S+C) | Z AND (D = S+C) | 2 |

**Related:** [CMP](#_jw7rhjdisro0), [CMPX](#_3w81cwnriog1), and [CMPSX](#_ccc7sdk782ug)

**Explanation:**

CMPX compares the unsigned values of Dest and Src plus C (by subtracting Src + C from Dest) and optionally setting the C and Z flags accordingly. The CMPX instruction is used to perform unsigned multi-long comparisons, such as 64-bit comparisons.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest is less than Src plus C, or is cleared (0) otherwise. Use WC or WCZ on preceding CMP and CMPX instructions for proper final C flag.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Z was previously set and Dest equals Src + C, or it is cleared (0) otherwise. Use WZ or WCZ on preceding CMP and CMPX instructions for proper final Z flag.

To compare multi-long values, use CMP followed by one or more CMPX instructions as described in [Comparing Two Multi-Long Values](#_kjzu1wz1qct1).

## COGATN

Cog attention

[**Event Handling Instruction**](#_6ehq03n22voi) **-** Get the attention of one or more other cogs.

COGATN {#}*Dest*

**Result:** The attention signal of one or more cogs is strobed.

* Dest is the register or 9-bit literal whose value (lower 8-bit pattern) indicates which cogs to signal.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | 00L | DDDDDDDDD | 000111111 | — | — | — | 2 |

**Related:** [POLLATN](#_ref43okqtzzr), [WAITATN](#_sb5186rnvudg), [JATN](#_yfon3snogp62), and [JNATN](#_yfon3snogp62)

**Explanation:**

COGATN strobes the attention signal for one or more cogs. Dest bit positions 7:0 represent cogs 7 through 0; high (1) bits indicate the cog(s) to signal. The receiving cog(s) then latch the signal, setting an internal flag, and can use any of the attention monitor instructions (JATN, JNATN, POLLATN, WAITATN) or interrupts to respond and clear the flag.

COGATN #%00100010 'Get attention of cogs 1 and 5

In the intended use case, the cog receiving an attention request knows which other cog is strobing it and how to respond. In cases where multiple cogs may request the attention of a single cog, some messaging structure may need to be implemented in Hub RAM to differentiate requests.

## COGBRK

Cog break

[**Interrupt Instruction**](#_40bj2qk54rdh) **-** Trigger breakpoint in specified cog.

COGBRK {#}*Dest*

**Result:** If in the Debug ISR, trigger an asynchronous breakpoint in cog identified by Dest.

* Dest is the register or 9-bit literal whose value (lower 3-bits) indicates which cog to trigger.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | 00L | DDDDDDDDD | 000110101 | — | — | — | 2 |

**Explanation:**

COGBRK triggers an asynchronous breakpoint in a designated cog. The COGBRK instruction must be executed from within a Debug ISR (interrupt service routine) and the designated cog must already have its asynchronous breakpoint interrupt enabled. Dest[2:0] indicates the ID of the desired cog.

## COGID

Cog identification

[**Cog Control Instruction**](#_131dn518do28) **-** Get current cog’s ID or any cog's status by ID.

COGID {#}*Dest* {WC}

**Result:** Current cog's ID is written to Dest or C is set (1) or cleared (0) if the Dest cog is running or stopped.

* Dest is the register where the current cog's ID will be written, or is the register or 9-bit literal whose value (lower 3-bits) indicates which cog to get the status for.
* WC is an optional effect to update the C flag with the Dest cog's running status.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | C0L | DDDDDDDDD | 000000001 | D if reg and !WC | Cog Running | — | 2–9, +2 if result |

**Related:** [COGINIT](#_5knlosjgh2sz) and [COGSTOP](#_eg5nftcbzayn)

**Explanation:**

COGID writes the current cog's ID into Dest (if Dest is a register and WC is omitted) or sets/clears the C flag according to the running/stopped state of the cog indicated by Dest[2:0] (if WC is given).

If the WC effect is specified, the C flag is set (1) if the Dest[2:0] cog is running, or is cleared (0) if stopped.

## COGINIT

Cog initialize

[**Cog Control Instruction**](#_131dn518do28) **-** Start an available cog, or restart a cog by ID.

COGINIT {#}*Dest*, {#}*Src* {WC}

**Result:** Target cog is started, according to Dest, to execute code from Src. The code pointer (Src) is written to the target Cog's PTRB, and optionally a data pointer or user value is written to its PTRA (if previous SETQ is provided).

* Dest is the register or 9-bit literal describing the type of launch and possibly the ID of the desired cog to launch. If Dest is a register and WC is given, Dest is also where the ID of the launched cog will be written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value (lower 20 bits) is the target RAM address (for code) and the new cog's PTRB value.
* WC is an optional effect to update the C flag with the success (0) or fail (1) status and triggers Dest to be overwritten with new cog's ID.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1100111 | CLI | DDDDDDDDD | SSSSSSSSS | D if reg and WC | No cog available | — | 2–9, +2 if result |

**Related:** [COGID](#_uy4mwi8z5ww4) and [COGSTOP](#_eg5nftcbzayn)

**Explanation:**

COGINIT starts a new (unused) cog, a new pair of cogs (that may share LUT memory), or a specific cog by ID, to load code from Hub RAM to be executed within Reg/LUT RAM or to be executed right from Hub RAM.

If the WC effect is specified, the C flag is set (1) if the instruction failed to launch a cog, or is cleared (0) if it was successful. Also with the WC effect, if Dest is a register, it is overwritten with $F (on failure) or with the ID of the newly launched cog —or the lowest of the pair of cogs— (on success).

The format of Dest is %E\_N\_xVVV.

E: 0 = load from Hub RAM (Src) into target cog's Reg RAM ($000–$1F7) and begin execution at its Reg RAM ($000); 1 = no load, just execute from target cog's existing Reg/LUT RAM, or from Hub RAM.

N: 0 = target cog ID is V; 1 = start a free cog or cog pair according to V.

V: when N = 0, V is a specific cog ID (0–7; %000–%111); when N = 1, V is 0 (%000) to start a free cog or 1 (%001) to start a free odd/even pair of cogs for LUT RAM sharing. Use the WC effect to get the results in Dest and C.

The lower 20 bits of Src is the address of the start of code; the address in Hub RAM (or Reg/LUT RAM) where the executable code image begins. The entire 32-bit Src value will be written into the target cog's PTRB register.

If COGINIT is preceded by SETQ, the SETQ value will be written into the target cog's PTRA register. This is intended as a convenient means of pointing the target cog's program to some runtime data structure or passing it a 32-bit parameter. If no SETQ is used, the target cog's PTRA register will be cleared to zero.

Related instructions are [COGID](#_uy4mwi8z5ww4) and [COGSTOP](#_eg5nftcbzayn).

**Execution Modes**

Code can be loaded into Cog RAM (Reg RAM and optionally LUT RAM) and executed, or code can be executed from Hub RAM without pre-loading into Cog RAM. For PASM2 programs, the former achieves the fastest execution speed and natural isolation from other cogs while the latter achieves a larger effective code space beyond the ≈1,000 instruction limit inside Cog RAM. It is common to run PASM2 code from Reg RAM.

To execute code from Reg RAM, it must first be loaded from Hub RAM (i.e. a Dest value where E = 0 achieves both *load* and *execute* in a single COGINIT instruction).

To execute code from LUT RAM, it must first be manually copied or streamed into LUT RAM by code executing in Reg RAM (or by the neighboring cog in an even/odd pair that are sharing LUT RAM).

To execute code from Hub RAM, the Dest's E bit must be 1 and the Src must be an address beyond $3FF. Note that with a single COGINIT instruction, code could start out loaded into Reg RAM (E = 0), execute from there, and can branch to $400 or beyond (Hub execution), or it can even copy or stream code into LUT RAM, execute in Reg+LUT RAM, and then branch to $400 or beyond (Hub execution).

Additionally, it's possible to load and execute code in another cog's Reg RAM (one COGINIT instruction) and restart that cog thereafter (another COGINIT instruction) to execute starting at any place in its existing Reg RAM or LUT RAM (from the previous load) by using a Dest value with E = 1 and a Src value of less than $400. This works whether or not the cog is terminated first, since its Cog RAM is retained during its terminated (dormant) state. For example, one possibility is a Reg/LUT RAM-resident program consisting of a number of small independent routines which are each exclusively started at-will by another cog— apparent cog launch and parallel processing with no "loading" overhead latency.

**COGINIT Constants**

A number of constants are predefined for use in forming desired Dest values for COGINIT instructions.

| COGINIT Constants | | |
| --- | --- | --- |
| **Symbol (for Dest)** | **Dest Value** | **Notes** |
| COGEXEC | %0\_0\_0000 | The target cog loads its own Reg RAM registers $000–$1F7 from the Hub RAM  starting at Src address, then begins execution at register address $000. Target cog ID must be specified; i.e. COGEXEC+5 will launch Cog 5 in this fashion. |
| COGEXEC\_NEW | %0\_1\_0000 | Same as above except an available (non-running) cog is chosen automatically; do not specify an ID. Use WC effect and a register for Dest (register's value set to COGEXEC\_NEW) to get pass/fail (0/1) result in C flag and ID of employed cog in Dest register. |
| COGEXEC\_NEW\_PAIR | %0\_1\_0001 | Same as above except an available (non-running) even/odd pair of cogs is chosen automatically— useful for LUT RAM sharing. Dest receives the lowest ID of the pair if successful (when the WC effect is used). |
| HUBEXEC | %1\_0\_0000 | No loading; the target cog begins execution at Reg/LUT/Hub RAM address Src. Target cog ID must be specified; i.e. HUBEXEC+2 will launch Cog 2 in this fashion. NOTE: Though named *HUB*EXEC, this constant can also be used to execute pre-existing code in Reg RAM or LUT RAM as well as Hub RAM. |
| HUBEXEC\_NEW | %1\_1\_0000 | Same as above except an available (non-running) cog is chosen automatically; do not specify an ID. Use WC effect and a register for Dest (register's value set to HUBEXEC\_NEW) to get pass/fail (0/1) result in C flag and ID of employed cog in Dest register. |
| HUBEXEC\_NEW\_PAIR | %1\_1\_0001 | Same as above except an available (non-running) even/odd pair of cogs is chosen automatically— useful for LUT RAM sharing. Dest receives the lowest ID of the pair if successful (when the WC effect is used). |

**Examples**

The following are individual examples.

COGINIT #1,#$100 'load and start Cog 1 from Hub RAM $100

—or—

COGINIT #COGEXEC+1,#$100

COGINIT #%1\_0\_0101, PTRA 'skip load and start Cog 5 at PTRA

—or—

COGINIT #HUBEXEC+5, PTRA

SETQ ptra\_val 'ptra\_val will go into target cog's PTRA register

COGINIT #%0\_1\_0000, addr 'load and start a free cog at addr

—or—

SETQ ptra\_val

COGINIT #COGEXEC\_NEW, addr

COGINIT #%1\_1\_0001, addr 'skip load, start a free pair of cogs at addr (for LUT RAM sharing)

—or—

COGINIT #HUBEXEC\_NEW\_PAIR, addr

COGINIT id, addr WC '(id=HUBEXEC\_NEW) skip load, start a free cog at addr

'C=0 and id=cog if okay

COGID myID 'reload and restart me at Hub RAM PTRB

COGINIT myID, PTRB

## COGSTOP

Cog stop

[**Cog Control Instruction**](#_131dn518do28) **-** Stop a cog by ID.

COGSTOP {#}*Dest*

**Result:** Cog indicated by Dest is terminated (stopped).

* Dest is the register or 9-bit literal indicating (in lowest 3 bits) which cog to stop.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | 00L | DDDDDDDDD | 000000011 | — | — | — | 2–9 |

**Related:** [COGINIT](#_5knlosjgh2sz) and [COGSTOP](#_eg5nftcbzayn)

**Explanation:**

COGSTOP terminates the cog identified by Dest[2:0]. In this dormant state, the cog ceases to execute code and power consumption is greatly reduced.

## [Conditions](#_ki1kkm7brm59) ( IF\_x )

Every PASM2 instruction has an optional “condition” that dynamically executes or excludes the instruction based on flag settings at runtime. A condition, if provided, is placed in front of the instruction it controls.

{Label} {Condition} Instruction Operands {Effect}

Conditions are optional; omitting the condition means "always execute" the instruction (the default behavior). The *Condition* field on an instruction line can contain one of fifty condition symbols (including blank; the default). The full set of condition symbols resolves down to 16 unique condition patterns (4-bits) that are stored in the instruction's opcode (the COND field) during assembly.

This *Condition* feature, along with the instructions’ optional *Effect* feature, makes Propeller 2 Assembly very powerful. For example, the C and Z flags can be affected at will and later instructions can be conditionally executed based on those results, making for easy behavioral changes through a single non-branching code path.

When an instruction’s condition evaluates to FALSE, the instruction effectively does nothing— still elapsing 2 clock cycles, but not affecting any flags or registers. This makes the timing of multi-decision code deterministic.

| Conditions | | |
| --- | --- | --- |
| **Condition1** | **Encoding2** | **Instruction Executes...** |
| IF\_E | %1010 | if comparison/subtraction was equal (Z = 1) |
| IF\_NE | %0101 | if comparison/subtraction was not equal (Z = 0) |
| IF\_A | %0001 | if comparison/subtraction was above (C = 0 and Z = 0) |
| IF\_AE | %0011 | if comparison/subtraction was above or equal (C = 0) |
| IF\_B | %1100 | if comparison/subtraction was below (C = 1) |
| IF\_BE | %1110 | if comparison/subtraction was below or equal (C = 1 or Z = 1) |
| IF\_GT | %0001 | if comparison/subtraction was greater than (C = 0 and Z = 0) |
| IF\_GE | %0011 | if comparison/subtraction was greater than or equal (C = 0) |
| IF\_LT | %1100 | if comparison/subtraction was less than (C = 1) |
| IF\_LE | %1110 | if comparison/subtraction was less than or equal (C = 1 or Z = 1) |
| IF\_C | %1100 | if C set (C = 1) |
| IF\_NC | %0011 | if C clear (C = 0) |
| IF\_Z | %1010 | if Z set (Z = 1) |
| IF\_NZ | %0101 | if Z clear (Z = 0) |
| IF\_C\_EQ\_Z | %1001 | if C equal to Z (C = 0 and Z = 0 --or-- C = 1 and Z = 1) |
| IF\_C\_NE\_Z | %0110 | if C not equal to Z (C = 0 and Z = 1 --or-- C = 1 and Z = 0) |
| IF\_C\_AND\_Z | %1000 | if C set and Z set (C = 1 and Z = 1) |
| IF\_C\_AND\_NZ | %0100 | if C set and Z clear (C = 1 and Z = 0) |
| IF\_NC\_AND\_Z | %0010 | if C clear and Z set (C = 0 and Z = 1) |
| IF\_NC\_AND\_NZ | %0001 | if C clear and Z clear (C = 0 and Z = 0) |
| IF\_C\_OR\_Z | %1110 | if C set or Z set (C = 1 or Z = 1) |
| IF\_C\_OR\_NZ | %1101 | if C set or Z clear (C = 1 or Z = 0) |
| IF\_NC\_OR\_Z | %1011 | if C clear or Z set (C = 0 or Z = 1) |
| IF\_NC\_OR\_NZ | %0111 | if C clear or Z clear (C = 1 or Z = 0) |
| IF\_Z\_EQ\_C | %1001 | if Z equal to C (Z = 0 and C = 0 --or-- Z = 1 and C = 1) |
| IF\_Z\_NE\_C | %0110 | if Z not equal to C (Z = 0 and C = 1 --or-- Z = 1 and C = 0) |
| IF\_Z\_AND\_C | %1000 | if Z set and C set (Z = 1 and C = 1) |
| IF\_Z\_AND\_NC | %0010 | if Z set and C clear (Z = 1 and C = 0) |
| IF\_NZ\_AND\_C | %0100 | if Z clear and C set (Z = 0 and C = 1) |
| IF\_NZ\_AND\_NC | %0001 | if Z clear and C clear (Z = 0 and C = 0) |
| IF\_Z\_OR\_C | %1110 | if Z set or C set (Z = 1 or C = 1) |
| IF\_Z\_OR\_NC | %1011 | if Z set or C clear (Z = 1 or C = 0) |
| IF\_NZ\_OR\_C | %1101 | if Z clear or C set (Z = 0 or C = 1) |
| IF\_NZ\_OR\_NC | %0111 | if Z clear or C clear (Z = 0 or C = 0) |
| IF\_00 | %0001 | if C clear and Z clear (C = 0 and Z = 0) |
| IF\_01 | %0010 | if C clear and Z set (C = 0 and Z = 1) |
| IF\_10 | %0100 | if C set and Z clear (C = 1 and Z = 0) |
| IF\_11 | %1000 | if C set and Z set (C = 1 and Z = 1) |
| IF\_X0 | %0101 | if Z clear (Z = 0) |
| IF\_X1 | %1010 | if Z set (Z = 1) |
| IF\_0X | %0011 | if C clear (C = 0) |
| IF\_1X | %1100 | if C set (C = 1) |
| IF\_NOT\_00 | %1110 | if C clear and Z clear (C = 0 and Z = 0) |
| IF\_NOT\_01 | %1101 | if C set or Z clear (C = 1 or Z = 0) |
| IF\_NOT\_10 | %1011 | if C clear or Z set (C = 0 or Z = 1) |
| IF\_NOT\_11 | %0111 | if C clear or Z clear (C = 0 or Z = 0) |
| IF\_DIFF | %0110 | if C not equal to Z (C = 0 and Z = 1 --or-- C = 1 and Z = 0) |
| IF\_SAME | %1001 | if C equal to Z (C = 0 and Z = 0 --or-- C = 1 and Z = 1) |
| \_RET\_ | %0000 | always; execute instruction then return if no branch; no context restore |
|  | %1111 | always; this is the default, no condition expressed |

1 The Condition symbol must be entered in place of the *{Condition}* item (ahead of the *Instruction*) on an instruction line.

2 The Encoding value is emitted into the COND field (the EEEE bits) in the compiled instruction opcode.

Note that for every Condition that acts upon the state of C and/or Z flags, there's also a [Modifier](#_hirqlphn43pc) (for use with [MODxx](#_m3xduhu6qlur) instructions) capable of applying flag state(s) based on similar logic. Additionally, the Condition symbols and descriptions are similar to, and the encoding exactly matches, that of the related Modifiers.

## DECMOD

Decrement modulus

[**Math Instruction**](#_xnk9pb8j5sof) **-** Decrement with modulus.

DECMOD *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** If Dest was not equal to 0, it is decremented by 1; otherwise Dest is reset to Src. Optionally, C and Z are updated to indicate reset and zero result status.

* Dest is a register containing the value to decrement down to 0 with modulus, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is the modulus limit to apply to Dest's decrement operation.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0111001 | CZI | DDDDDDDDD | SSSSSSSSS | D | Modulus triggered | Result = 0 | 2 |

**Related:** [INCMOD](#_ln0kx81xiaak)

**Explanation:**

DECMOD compares Dest with 0— if not equal, it decrements Dest; otherwise it sets Dest equal to Src. If Dest begins in the range 0 to Src, iterations of DECMOD will decrement Dest repetitively from Src to 0.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest was equal to 0 and subsequently reset to Src; or is cleared (0) if not reset.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

DECMOD does not limit Dest within the specified range— if Dest begins as greater than Src, iterations of DECMOD will continue to decrement it down through Src before it will effectively cycle from Src to 0.

## DECOD

Decode

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Decode value (0—31) into single-high-bit long.

DECOD *Dest*, {#}*Src*

DECOD *Dest*

**Result:** A 32-bit value, with the bit position corresponding to Src, or Dest, value (0—31) set high, is stored in Dest.

* Dest is the register in which to store the decoded value and optionally begins by containing the 5-bit *bit position* value it is requesting (syntax 2).
* Src is an optional register or 5-bit literal whose value is the bit position to set high in the decoded value.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001110 | 00I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1001110 | 000 | DDDDDDDDD | DDDDDDDDD | D | — | — | 2 |

**Related:** [ENCOD](#_brfhxn6t7t41)

**Explanation:**

DECOD generates a 32-bit value with just one bit high, corresponding to the Src, or Dest, value (0—31) and stores that result in Dest.

In effect, Dest becomes %1 << value via the DECOD instruction; where value is Src[4:0] or Dest[4:0].

* A value of 0 generates %00000000\_00000000\_00000000\_00000001.
* A value of 5 generates %00000000\_00000000\_00000000\_00100000.
* A value of 15 generates %00000000\_00000000\_10000000\_00000000.

DECOD is the complement of ENCOD.

## DIRC / DIRNC

Direction C or not C

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to input/output according to C or !C.

DIRC {#}*Dest* {WCZ}

DIRNC {#}*Dest* {WCZ}

**Result:** The I/O pin direction bit(s), described by Dest, are set to output/input according to C or !C; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to output or input.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000010 | DIRx | Orig DIRx base bit | Orig DIRx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000011 | DIRx | Orig DIRx base bit | Orig DIRx base bit | 2 |

**Related:** [DIRZ](#_eaobslhzfr83), [DIRNZ](#_eaobslhzfr83), [DIRL](#_adfu0crzr5tx), [DIRH](#_adfu0crzr5tx), [DIRNOT](#_8cdi9p9b7vgm), and [DIRRND](#_5p7siddmqpeb)

**Explanation:**

DIRC or DIRNC alters the direction register's bit(s) designated by Dest to equal the state, or inverse state, of the C flag; i.e. set pin to the output (1) or input (0) direction. All other bits (pins) are left unchanged. Each of these instructions, DIRC and DIRNC, can affect one or more of the bits within the DIRA or DIRB registers.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the DIRC / DIRNC instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for DIRC / DIRNC's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of DIRA / DIRB's base bit, identified by Dest.

## DIRH / DIRL

Direction high or low

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to output (high; 1) or input (low; 0).

DIRH {#}*Dest* {WCZ}

DIRL {#}*Dest* {WCZ}

**Result:** The I/O pin direction bit(s), described by Dest, are set high (1; output) or low (0; input); the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to output or input.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000001 | DIRx | Orig DIRx base bit | Orig DIRx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000000 | DIRx | Orig DIRx base bit | Orig DIRx base bit | 2 |

**Related:** [DIRC](#_q5818ady3ccx), [DIRNC](#_q5818ady3ccx), [DIRZ](#_eaobslhzfr83), [DIRNZ](#_eaobslhzfr83), [DIRNOT](#_8cdi9p9b7vgm), and [DIRRND](#_5p7siddmqpeb)

**Explanation:**

DIRH or DIRL alters the direction register's bit(s) designated by Dest to be high (1) or low (0); i.e. set to the output or input direction. All other bits (pins) are left unchanged. Each of these instructions, DIRH and DIRL, can affect one or more of the bits within the DIRA or DIRB registers.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the DIRH / DIRL instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for DIRH / DIRL's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of DIRA / DIRB's base bit, identified by Dest.

## DIRNOT

Direction not

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Toggle pin(s) to the opposite direction.

DIRNOT {#}*Dest* {WCZ}

**Result:** The I/O pin direction bit(s), described by Dest, are toggled to their opposite state(s); the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to toggle to the opposite direction.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000111 | DIRx | Orig DIRx base bit | Orig DIRx base bit | 2 |

**Related:** [DIRRND](#_5p7siddmqpeb), [DIRL](#_adfu0crzr5tx), [DIRH](#_adfu0crzr5tx), [DIRC](#_q5818ady3ccx), [DIRNC](#_q5818ady3ccx), [DIRZ](#_eaobslhzfr83), and [DIRNZ](#_eaobslhzfr83)

**Explanation:**

DIRNOT alters the direction register's bit(s) designated by Dest to their inverse state. All other bits are left unchanged.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the DIRNOT instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for DIRNOT's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of DIRA / DIRB's base bit, identified by Dest.

## DIRRND

Direction random

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to random input/output.

DIRRND {#}*Dest* {WCZ}

**Result:** The I/O pin direction bit(s), described by Dest, are each set randomly low or high (input or output); the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the pins set randomly to inputs or outputs.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000110 | DIRx | Orig DIRx base bit | Orig DIRx base bit | 2 |

**Related:** [DIRNOT](#_8cdi9p9b7vgm), [DIRL](#_adfu0crzr5tx), [DIRH](#_adfu0crzr5tx), [DIRC](#_q5818ady3ccx), [DIRNC](#_q5818ady3ccx), [DIRZ](#_eaobslhzfr83), and [DIRNZ](#_eaobslhzfr83)

**Explanation:**

DIRRND alters the direction register's bit(s) designated by Dest to be random low and high (input and output), based on bit(s) from the Xoroshiro128\*\* PRNG. All other bits are left unchanged.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the DIRRND instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for DIRRND's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of DIRA / DIRB's base bit, identified by Dest.

## DIRZ / DIRNZ

Direction Z or not Z

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to input/output according to Z or !Z.

DIRZ {#}*Dest* {WCZ}

DIRNZ {#}*Dest* {WCZ}

**Result:** The I/O pin direction bit(s), described by Dest, are set to output/input according to Z or !Z; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to output or input.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZI | DDDDDDDDD | 001000100 | DIRx | Orig DIRx base bit | Orig DIRx base bit | 2 |
| EEEE | 1101011 | CZI | DDDDDDDDD | 001000101 | DIRx | Orig DIRx base bit | Orig DIRx base bit | 2 |

**Related:** [DIRC](#_q5818ady3ccx), [DIRNC](#_q5818ady3ccx), [DIRNOT](#_8cdi9p9b7vgm), [DIRRND](#_5p7siddmqpeb), [DIRL](#_adfu0crzr5tx), and [DIRH](#_adfu0crzr5tx)

**Explanation:**

DIRZ or DIRNZ alters the direction register's bit(s) designated by Dest to equal the state, or inverse state, of the Z flag; i.e. set pin to the output (1) or input (0) direction. All other bits (pins) are left unchanged. Each of these instructions, DIRZ and DIRNZ, can affect one or more of the bits within the DIRA or DIRB registers.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the DIRZ / DIRNZ instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for DIRZ / DIRNZ's use.

If the WCZ effect is specified, the C and Z flags are updated to the original state of DIRA / DIRB's base bit, identified by Dest.

## DJF / DJNF

Decrement, jump if full or not full

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Decrement value and jump if full (-1; $FFFF\_FFFF) or not full (<> -1; <> $FFFF\_FFFF).

DJF *Dest*, {#}*Src*

DJNF *Dest*, {#}*Src*

**Result:** Dest is decremented, and if the result is full (or not full in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Dest is a register whose value is decremented and tested for full or not full.
* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011011 | 10I | DDDDDDDDD | SSSSSSSSS | D and PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011011 | 11I | DDDDDDDDD | SSSSSSSSS | D and PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 Dest is always written; PC is written only when the result in Dest is full (or not full in syntax 2).

**Explanation:**

DJF or DJNF decrements the value in Dest, writes the result, and jumps to the address described by Src if the result is full (-1; $FFFF\_FFFF; in syntax 1) or not full (<> -1; <> $FFFF\_FFFF; in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the DJF / DJNF. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## DJZ / DJNZ

Decrement, jump if zero or not zero

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Decrement, jump if zero or not zero.

DJZ *Dest*, {#}*Src*

DJNZ *Dest*, {#}*Src*

**Result:** Dest is decremented, and if the result is zero (or not zero in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Dest is a register whose value is decremented and tested for zero or not zero.
* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011011 | 00I | DDDDDDDDD | SSSSSSSSS | D and PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011011 | 01I | DDDDDDDDD | SSSSSSSSS | D and PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 Dest is always written; PC is written only when the result in Dest is zero (or not zero in syntax 2).

**Explanation:**

DJZ or DJNZ decrements the value in Dest, writes the result, and jumps to the address described by Src if the result is zero (in syntax 1) or not zero (in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the DJZ / DJNZ. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## DRVC / DRVNC

Drive C or not C

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to output and output level to low/high according to C or !C.

DRVC {#}*Dest* {WCZ}

DRVNC {#}*Dest* {WCZ}

**Result:** The I/O pins described by Dest are set to the output direction and to an output level of low/high according to C or !C; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to output direction and output levels of low or high.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001011010 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001011011 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

1 New DIRx state is not data-forwarded; the next pipelined instruction sees the old state. Make sure any instruction that reads or modifies DIRx is at least two instructions after a DRVC or DRVNC.

**Explanation:**

DRVC or DRVNC sets the I/O pin(s) designated by Dest to the output direction and to a low/high output level according to the state, or inverse state, of the C flag; i.e. alters the pin's direction and output registers. All other pins are left unchanged. Each of these instructions, DRVC and DRVNC, can affect one or more of the bits within the DIRA or DIRB and OUTA or OUTB registers.

DRVC or DRVNC achieves the same effect as two instructions— OUTC, or OUTNC, followed by DIRH.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the DRVC / DRVNC instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for DRVC / DRVNC's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB and OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## DRVH / DRVL

Drive high or low

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to output and output level high (1) or low (0).

DRVH {#}*Dest* {WCZ}

DRVL {#}*Dest* {WCZ}

**Result:** The I/O pins described by Dest are set to the output direction and to an output level of high or low; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to output direction and output levels of high or low.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001011001 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001011000 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

1 New DIRx state is not data-forwarded; the next pipelined instruction sees the old state. Make sure any instruction that reads or modifies DIRx is at least two instructions after a DRVH or DRVL.

**Explanation:**

DRVH or DRVL sets the I/O pin(s) designated by Dest to the output direction and to a high or low output level; i.e. alters the pin's direction and output registers. All other pins are left unchanged. Each of these instructions, DRVH and DRVL, can affect one or more of the bits within the DIRA or DIRB and OUTA or OUTB registers.

DRVH or DRVL achieves the same effect as two instructions— OUTH, or OUTL, followed by DIRH.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the DRVH / DRVL instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for DRVH / DRVL's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB and OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## DRVNOT

Drive not

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to output and toggle to the opposite output level.

DRVNOT {#}*Dest* {WCZ}

**Result:** The I/O pins described by Dest are set to the output direction and to their opposite output level(s); the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to the output direction and toggle to opposite output levels.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001011111 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

1 New DIRx state is not data-forwarded; the next pipelined instruction sees the old state. Make sure any instruction that reads or modifies DIRx is at least two instructions after a DRVNOT.

**Explanation:**

DRVNOT sets the I/O pin(s) designated by Dest to the output direction and to their opposite output level(s); i.e. alters the pin's direction and output registers. All other pins are left unchanged. This instruction can affect one or more of the bits within the DIRA or DIRB and OUTA or OUTB registers.

DRVNOT achieves the same effect as two instructions— OUTNOT followed by DIRH.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the DRVNOT instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for DRVNOT's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB and OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## DRVRND

Drive random

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to output and output level to random low/high.

DRVRND {#}*Dest* {WCZ}

**Result:** The I/O pins described by Dest are set to the output direction and each output level is set randomly low or high; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to the output direction and with output level(s) set randomly to low or high.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001011110 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

1 New DIRx state is not data-forwarded; the next pipelined instruction sees the old state. Make sure any instruction that reads or modifies DIRx is at least two instructions after a DRVRND.

**Explanation:**

DRVRND sets the I/O pin(s) designated by Dest to the output direction and with output level(s) set randomly low and high, based on bit(s) from the Xoroshiro128\*\* PRNG. All other pins are left unchanged. This instruction can affect one or more of the bits within the DIRA or DIRB and OUTA or OUTB registers.

DRVRND achieves the same effect as two instructions— OUTRND followed by DIRH.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the DRVRND instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for DRVRND's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB and OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## DRVZ / DRVNZ

Drive Z or not Z

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to output and output level to low/high according to Z or !Z.

DRVZ {#}*Dest* {WCZ}

DRVNZ {#}*Dest* {WCZ}

**Result:** The I/O pins described by Dest are set to the output direction and to an output level of low/high according to Z or !Z; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to output direction and output levels of low or high.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001011100 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001011101 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

1 New DIRx state is not data-forwarded; the next pipelined instruction sees the old state. Make sure any instruction that reads or modifies DIRx is at least two instructions after a DRVZ or DRVNZ.

**Explanation:**

DRVZ or DRVNZ sets the I/O pin(s) designated by Dest to the output direction and to a low/high output level according to the state, or inverse state, of the Z flag; i.e. alters the pin's direction and output registers. All other pins are left unchanged. Each of these instructions, DRVZ and DRVNZ, can affect one or more of the bits within the DIRA or DIRB and OUTA or OUTB registers.

DRVZ or DRVNZ achieves the same effect as two instructions— OUTZ, or OUTNZ, followed by DIRH.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the DRVZ / DRVNZ instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for DRVZ / DRVNZ's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB and OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## ENCOD

Encode

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Get bit position of top-most 1 of Src or Dest into Dest.

ENCOD *Dest*, {#}*Src* {WC|WZ|WCZ}

ENCOD *Dest* {WC|WZ|WCZ}

**Result:** The bit position value of the top-most high bit (1) in Src, or Dest, is stored in Dest.

* Dest is the register in which to store the encoded bit position value and optionally begins by containing the 32-bit value it is encoding (syntax 2).
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose value is to be encoded into a bit position.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0111100 | CZI | DDDDDDDDD | SSSSSSSSS | D | S != 0 | Result = 0 | 2 |
| EEEE | 0111100 | CZ0 | DDDDDDDDD | DDDDDDDDD | D | Original D != 0 | Result = 0 | 2 |

**Related:** [DECOD](#_vp3efvde1vn6)

**Explanation:**

ENCOD stores the bit position value (0—31) of the top-most high bit (1) of Src, or Dest, into Dest. If the value to encode (Src, or original Dest (in syntax 2)) may be %0, the resulting Dest will be 0— use the WC or WCZ effect and check the resulting C flag to distinguish between the cases of input Src/Dest = %1 verses input Src/Dest = %0.

If the WC or WCZ effect is specified, the C flag is set (1) if Src (or original Dest in syntax 2) was not zero, or is cleared (0) if it was zero.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result equals zero, or is cleared (0) if not zero.

* A long of %00000000\_00000000\_00000000\_00000001 encodes to 0.
* A long of %00000000\_00000000\_00000000\_00100000 encodes to 5.
* A long of %00000000\_00000000\_10000001\_01000000 encodes to 15.
* A long of %00000000\_00000000\_00000000\_00000000 encodes to 0 with optional C cleared to 0.

ENCOD is the complement of DECOD.

## [Effects](#_4y0ciq9th2g) ( WC, WZ, WCZ, ANDC, etc. )

Nearly half of PASM2 instructions feature optional effects to modify the C and/or Z flags. An Effect is placed at the end of such instructions.

{Label} {Condition} Instruction Operands {Effect}

When included (where allowed) the flag or flags are updated by the instruction execution; when omitted, the flags remain as-is. Only zero or one Effect is allowed per instruction.

| Effects | |
| --- | --- |
| **Effect** | **Description** |
| ANDC | AND tested bit/pin into current C; used on TESTxx instructions |
| ANDZ | AND tested bit/pin into current Z; used on TESTxx instructions |
| ORC | OR tested bit/pin into current C; used on TESTxx instructions |
| ORZ | OR tested bit/pin into current Z; used on TESTxx instructions |
| XORC | XOR tested bit/pin into current C; used on TESTxx instructions |
| XORZ | XOR tested bit/pin into current Z; used on TESTxx instructions |
| WC | Write C flag; used on many instructions |
| WCZ | Write both C and Z flags; used on many instructions |
| WZ | Write Z flag; used on many instructions |

For example:

AND config, #%1000 WZ 'config[3] low? z=1

TESTP #4 ANDZ 'pin 4 high? z=1

if\_z JMP #MoreCode 'jump if config[3] low and pin 4 high

This code jumps to the label *MoreCode* only if *config* bit 3 is low (0) and I/O pin 4 is high. This first instruction bitwise AND's %1000 into *config* and sets the Z flag if that result is zero. The second instruction tests I/O pin 4 and bitwise AND's its high/low state into the Z flag. The third instruction, which specifies an if\_z condition, executes only if the z flag is high by then, jumping to *MoreCode*; otherwise it behaves like a nop instruction. Using an Effect on instructions, along with a Condition on later instructions, enables code to be much more powerful than what is possible with typical assembly languages. See [IF\_x (Conditions)](#_ofohv8f4qicc) for more information. NOTE: In this example, the first instruction modifies *config*— to do non-destructive bit tests, use a testb instead of and instruction.

## FGE

Force greater or equal

[**Math Instruction**](#_xnk9pb8j5sof) **-** Force unsigned value to be greater than or equal to another.

FGE *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Unsigned Dest is set to unsigned Src if Dest was less than Src. Optionally the C and Z flag indicates if the replacement happened and the zero status of the result.

* Dest is a register containing the unsigned value to limit to a minimum of unsigned Src, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose unsigned value is the lower limit to force upon Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0011000 | CZI | DDDDDDDDD | SSSSSSSSS | D | limit enforced | Result = 0 | 2 |

**Related:** [FLE](#_kjr0bke2otvv), [FGES](#_6pb8tmuj8hrq), and [FLES](#_oo1zbqj5xudg)

**Explanation:**

FGE sets unsigned Dest to unsigned Src if Dest is less than Src. This is also known as a *limit minimum* function; preventing Dest from sinking below Src.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest was limited (Dest was less than Src and now Dest is equal to Src), or is cleared (0) if not limited.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## FGES

Force greater or equal, signed

[**Math Instruction**](#_xnk9pb8j5sof) **-** Force signed value to be greater than or equal to another.

FGES *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Signed Dest is set to signed Src if Dest was less than Src. Optionally the C and Z flag indicates if the replacement happened and the zero status of the result.

* Dest is a register containing the signed value to limit to a minimum of signed Src, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose signed value is the lower limit to force upon Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0011010 | CZI | DDDDDDDDD | SSSSSSSSS | D | limit enforced | Result = 0 | 2 |

**Related:** [FLES](#_oo1zbqj5xudg), [FGE](#_4qzq1q9ohttb), and [FLE](#_kjr0bke2otvv)

**Explanation:**

FGES sets signed Dest to signed Src if Dest is less than Src. This is also known as a *limit minimum* function; preventing Dest from sinking below Src.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest was limited (Dest was less than Src and now Dest is equal to Src), or is cleared (0) if not limited.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## FLE

Force lesser or equal

[**Math Instruction**](#_xnk9pb8j5sof) **-** Force unsigned value to be less than or equal to another.

FLE *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Unsigned Dest is set to unsigned Src if Dest was greater than Src. Optionally the C and Z flag indicates if the replacement happened and the zero status of the result.

* Dest is a register containing the unsigned value to limit to a maximum of unsigned Src, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose unsigned value is the upper limit to force upon Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0011001 | CZI | DDDDDDDDD | SSSSSSSSS | D | limit enforced | Result = 0 | 2 |

**Related:** [FGE](#_4qzq1q9ohttb), [FLES](#_oo1zbqj5xudg), and [FGES](#_6pb8tmuj8hrq)

**Explanation:**

FLE sets unsigned Dest to unsigned Src if Dest is greater than Src. This is also known as a *limit maximum* function; preventing Dest from rising above Src.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest was limited (Dest was greater than Src and now Dest is equal to Src), or is cleared (0) if not limited.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## FLES

Force lesser or equal, signed

[**Math Instruction**](#_xnk9pb8j5sof) **-** Force signed value to be less than or equal to another.

FLES *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Signed Dest is set to signed Src if Dest was greater than Src. Optionally the C and Z flag indicates if the replacement happened and the zero status of the result.

* Dest is a register containing the signed value to limit to a maximum of signed Src, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose signed value is the upper limit to force upon Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0011011 | CZI | DDDDDDDDD | SSSSSSSSS | D | limit enforced | Result = 0 | 2 |

**Related:** [FGES](#_6pb8tmuj8hrq), [FLE](#_kjr0bke2otvv), and [FGE](#_4qzq1q9ohttb)

**Explanation:**

FLES sets signed Dest to signed Src if Dest is greater than Src. This is also known as a *limit maximum* function; preventing Dest from rising above Src.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest was limited (Dest was greater than Src and now Dest is equal to Src), or is cleared (0) if not limited.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## FLTC / FLTNC

Float C or not C

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to input and an output level of low/high according to C.

FLTC {#}*Dest* {WCZ}

FLTNC {#}*Dest* {WCZ}

**Result:** The I/O pins described by Dest are set to the input direction and to an output level of low/high according to C or !C; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to input direction and output levels of low or high.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001010010 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001010011 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

1 New DIRx state is not data-forwarded; the next pipelined instruction sees the old state. Make sure any instruction that reads or modifies DIRx is at least two instructions after a FLTC or FLTNC.

**Explanation:**

FLTC or FLTNC sets the I/O pin(s) designated by Dest to the input direction and to a low/high output level according to the state, or inverse state, of the C flag; i.e. alters the pin's direction and output registers. All other pins are left unchanged. Each of these instructions, FLTC and FLTNC, can affect one or more of the bits within the DIRA or DIRB and OUTA or OUTB registers.

FLTC or FLTNC achieves the same effect as two instructions— DIRL followed by OUTC or OUTNC.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the FLTC / FLTNC instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for FLTC / FLTNC's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB and OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## FLTH / FLTL

Float high or low

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to input and to an output level of high (1) or low (0).

FLTH {#}*Dest* {WCZ}

FLTL {#}*Dest* {WCZ}

**Result:** The I/O pins described by Dest are set to the input direction and to an output level of high or low; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to input direction and output levels of high or low.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001010001 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001010000 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

1 New DIRx state is not data-forwarded; the next pipelined instruction sees the old state. Make sure any instruction that reads or modifies DIRx is at least two instructions after a FLTH or FLTL.

**Explanation:**

FLTH or FLTL sets the I/O pin(s) designated by Dest to the input direction and to a high or low output level; i.e. alters the pin's direction and output registers. All other pins are left unchanged. Each of these instructions, FLTH and FLTL, can affect one or more of the bits within the DIRA or DIRB and OUTA or OUTB registers.

FLTH or FLTL achieves the same effect as two instructions— DIRL followed by OUTH or OUTL.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the FLTH / FLTL instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for FLTH / FLTL's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB and OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## 

## FLTNOT

Float not

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to input and toggle to the opposite output level.

FLTNOT {#}*Dest* {WCZ}

**Result:** The I/O pins described by Dest are set to the input direction and to their opposite output level(s); the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to the input direction and toggle to opposite output levels.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001010111 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

1 New DIRx state is not data-forwarded; the next pipelined instruction sees the old state. Make sure any instruction that reads or modifies DIRx is at least two instructions after a FLTNOT.

**Explanation:**

FLTNOT sets the I/O pin(s) designated by Dest to the input direction and to their opposite output level(s); i.e. alters the pin's direction and output registers. All other pins are left unchanged. This instruction can affect one or more of the bits within the DIRA or DIRB and OUTA or OUTB registers.

FLTNOT achieves the same effect as two instructions— DIRL followed by OUTNOT.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the FLTNOT instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for FLTNOT's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB and OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## FLTRND

Float random

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to input and to an output level of random low/high.

FLTRND {#}*Dest* {WCZ}

**Result:** The I/O pins described by Dest are set to the input direction and each output level is set randomly low or high; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to the input direction and with output level(s) set randomly to low or high.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001010110 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

1 New DIRx state is not data-forwarded; the next pipelined instruction sees the old state. Make sure any instruction that reads or modifies DIRx is at least two instructions after a FLTRND.

**Explanation:**

FLTRND sets the I/O pin(s) designated by Dest to the input direction and with output level(s) set randomly low and high, based on bit(s) from the Xoroshiro128\*\* PRNG. All other pins are left unchanged. This instruction can affect one or more of the bits within the DIRA or DIRB and OUTA or OUTB registers.

FLTRND achieves the same effect as two instructions— DIRL followed by OUTRND.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the FLTRND instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for FLTRND's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB and OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## FLTZ / FLTNZ

Float Z or not Z

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) direction to input and an output level of low/high according to Z.

FLTZ {#}*Dest* {WCZ}

FLTNZ {#}*Dest* {WCZ}

**Result:** The I/O pins described by Dest are set to the input direction and to an output level of low/high according to Z or !Z; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set to input direction and output levels of low or high.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001010100 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001010101 | DIRx1 + OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

1 New DIRx state is not data-forwarded; the next pipelined instruction sees the old state. Make sure any instruction that reads or modifies DIRx is at least two instructions after a FLTZ or FLTNZ.

**Explanation:**

FLTZ or FLTNZ sets the I/O pin(s) designated by Dest to the input direction and to a low/high output level according to the state, or inverse state, of the Z flag; i.e. alters the pin's direction and output registers. All other pins are left unchanged. Each of these instructions, FLTZ and FLTNZ, can affect one or more of the bits within the DIRA or DIRB and OUTA or OUTB registers.

FLTZ or FLTNZ achieves the same effect as two instructions— DIRL followed by OUTZ or OUTNZ.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the FLTZ / FLTNZ instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for FLTZ / FLTNZ's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (DIRA or DIRB and OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## GETBYTE

Get byte

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Get a byte from a value.

GETBYTE *Dest*, {#}*Src,* #*Num*

GETBYTE *Dest*

**Result:** Byte Num (0–3) of Src, or a byte from a source described by prior ALTGB instruction, is written to Dest.

* Dest is the register in which to store the byte.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value contains the target byte to read.
* Num is a 2-bit literal identifying the byte ID (0–3) of Src to read.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1000111 | NNI | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1000111 | 000 | DDDDDDDDD | 000000000 | D | — | — | 2 |

**Related:** [ALTGB](#_q234901aifm8), [GETNIB](#_8yjp9q7d9ncx), [GETWORD](#_ppayepmbzjsl), [SETNIB](#_dtkktvehwfob), [SETBYTE](#_f8filcyhyg4l), [SETWORD](#_9gwjt0ijlutf), [ROLNIB](#_5imtv7yml80t), [ROLBYTE](#_8x2v3buaskxv), and [ROLWORD](#_lhi32k9v1cp3)

**Explanation:**

GETBYTE reads the byte identified by Num (0–3) from Src, or a byte from the source described by a prior ALTGB instruction.

Num (0–3) identifies a value's individual bytes, by position, in least-significant byte order.

Syntax 2 is intended for use after an ALTGB instruction; i.e. in a loop to iteratively read a series of byte values within contiguous long registers.

## GETNIB

Get nibble

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Get a nibble from a value.

GETNIB *Dest*, {#}*Src,* #*Num*

GETNIB *Dest*

**Result:** Nibble Num (0–7) of Src, or a nibble from a source described by prior ALTGN instruction, is written to Dest.

* Dest is the register in which to store the nibble.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value contains the target nibble to read.
* Num is a 3-bit literal identifying the nibble ID (0–7) of Src to read.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 100001N | NNI | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1000010 | 000 | DDDDDDDDD | 000000000 | D | — | — | 2 |

**Related:** [ALTGN](#_asw5rsrcqtpn), [GETBYTE](#_t84l3o6tozfm), [GETWORD](#_ppayepmbzjsl), [SETNIB](#_dtkktvehwfob), [SETBYTE](#_f8filcyhyg4l), [SETWORD](#_9gwjt0ijlutf), [ROLNIB](#_5imtv7yml80t), [ROLBYTE](#_8x2v3buaskxv), and [ROLWORD](#_lhi32k9v1cp3)

**Explanation:**

GETNIB reads the nibble identified by Num (0–7) from Src, or a nibble from the source described by a prior ALTGN instruction.

Num (0–7) identifies a value's individual nibbles, by position, in least-significant nibble order.

Syntax 2 is intended for use after an ALTGN instruction; i.e. in a loop to iteratively read a series of nibble values within contiguous long registers.

## GETWORD

Get word

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Get a word from a value.

GETWORD *Dest*, {#}*Src,* #*Num*

GETWORD *Dest*

**Result:** Word Num (0–1) of Src, or a word from a source described by prior ALTGW instruction, is written to Dest.

* Dest is the register in which to store the word.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value contains the target word to read.
* Num is a 1-bit literal identifying the word ID (0–1) of Src to read.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001001 | 1NI | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1001001 | 100 | DDDDDDDDD | 000000000 | D | — | — | 2 |

**Related:** [ALTGW](#_hahk50n443um), [GETNIB](#_8yjp9q7d9ncx), [GETBYTE](#_t84l3o6tozfm), [SETNIB](#_dtkktvehwfob), [SETBYTE](#_f8filcyhyg4l), [SETWORD](#_9gwjt0ijlutf), [ROLNIB](#_5imtv7yml80t), [ROLBYTE](#_8x2v3buaskxv), and [ROLWORD](#_lhi32k9v1cp3)

**Explanation:**

GETWORD reads the word identified by Num (0–1) from Src, or a word from the source described by a prior ALTGW instruction.

Num (0–1) identifies a value's words, by position, in least-significant word order.

Syntax 2 is intended for use after an ALTGW instruction; i.e. in a loop to iteratively read a series of word values within contiguous long registers.

## IF\_x (Conditions)

See [Conditions ( IF\_x )](#_ofohv8f4qicc).

## IJZ / IJNZ

Increment, jump if zero or not zero

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Increment value and jump if zero or not zero.

IJZ *Dest*, {#}*Src*

IJNZ *Dest*, {#}*Src*

**Result:** Dest is incremented, and if the result is zero (or not zero in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Dest is a register whose value is incremented and tested for zero or not zero.
* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011100 | 00I | DDDDDDDDD | SSSSSSSSS | D and PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011100 | 01I | DDDDDDDDD | SSSSSSSSS | D and PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 Dest is always written; PC is written only when the result in Dest is zero (or not zero in syntax 2).

**Explanation:**

IJZ or IJNZ increments the value in Dest, writes the result, and jumps to the address described by Src if the result is zero (syntax 1) or not zero (in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the IJZ / IJNZ. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## INCMOD

Increment modulus

[**Math Instruction**](#_xnk9pb8j5sof) **-** Increment with modulus.

INCMOD *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** If Dest was not equal to Src, it is incremented by 1; otherwise Dest is reset to 0. Optionally, C and Z are updated to indicate reset and zero result status.

* Dest is a register containing the value to increment up to Src with modulus, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is the modulus limit to apply to Dest's increment operation.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0111000 | CZI | DDDDDDDDD | SSSSSSSSS | D | Modulus triggered | Result = 0 | 2 |

**Related:** [DECMOD](#_jjzqxknhgv63)

**Explanation:**

INCMOD compares Dest with Src— if not equal, it increments Dest; otherwise it sets Dest equal to 0. If Dest begins in the range 0 to Src, iterations of INCMOD will increment Dest repetitively from 0 to Src.

If the WC or WCZ effect is specified, the C flag is set (1) if Dest was equal to Src and subsequently reset to 0; or is cleared (0) if not reset.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

INCMOD does not limit Dest within the specified range— if Dest begins as greater than Src, iterations of INCMOD will continue to increment it through the 32-bit rollover point (back to 0) before it will effectively cycle from 0 to Src.

## JATN / JNATN

Jump if attention or not attention

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if attention flag is set or clear.

JATN {#}*Src*

JNATN {#}*Src*

**Result:** If ATN event flag is set (or is clear in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000001110 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000011110 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the ATN event flag is set (or is clear in syntax 2).

**Explanation:**

JATN or JNATN checks the cog's attention signal and jumps to the address described by Src if attention is set (in syntax 1) or is clear (in syntax 2). The cog's attention signal, when set, indicates that one or more other cogs are requesting this cog's attention. See the [Hardware Manual's](https://www.parallax.com/propeller-2/documentation/) Cog Attention section for more information.

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JATN / JNATN. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## 

## JCT1/2/3 / JNCT1/2/3

Jump if counter 1/2/3 or not counter 1/2/3

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if counter 1, 2, or 3 event flag is set or clear.

JCT1 {#}*Src*

JCT2 {#}*Src*

JCT3 {#}*Src*

JNCT1 {#}*Src*

JNCT2 {#}*Src*

JNCT3 {#}*Src*

**Result:** If counter 1, 2, or 3 event flag is set (or is clear in syntax 4–6), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000000001 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000000010 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000000011 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000010001 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000010010 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000010001 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the counter event flag is set (or is clear in syntax 4–6).

**Explanation:**

JCT1, JCT2, JCT3, or JNCT1, JNCT2, JNCT3 checks the cog's counter 1, 2, or 3 event flag and jumps to the address described by Src if the flag is set (in syntax 1–3) or is clear (in syntax 4–6). The cog's hidden registers, CT1, CT2, and CT3 are dedicated to System Counter timing and events— when a counter event flag is set, it means a specified time period has elapsed. See the [Hardware Manual's](https://www.parallax.com/propeller-2/documentation/) System Counter section for more information.

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JCTx / JNCTx. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

Related instructions are [ADDCTx](#_u2xzjautl27l), POLLCTx, and WAITCTx.

## JFBW / JNFBW

Jump if FIFO block wrap or not FIFO block wrap

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if FIFO interface block wrap event flag is set or clear.

JFBW {#}*Src*

JNFBW {#}*Src*

**Result:** If FIFO interface block wrap event flag is set (or is clear in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000001001 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000011001 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the FIFO interface block wrap event flag is set (or is clear in syntax 2).

**Explanation:**

JFBW or JNFBW checks the cog's FIFO interface block wrap flag and jumps to the address described by Src if the flag is set (in syntax 1) or is clear (in syntax 2). The cog's Fast Sequential FIFO Interface is used to swiftly transfer data between Hub and Cog. When the FIFO runs out of data (block count), it sets this flag before wrapping around to the start of the block again.

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JFBW / JNFBW. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## JINT / JNINT

Jump if interrupt or not interrupt

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if interrupt-occurred event flag is set or clear.

JINT {#}*Src*

JNINT {#}*Src*

**Result:** If interrupt-occurred event flag is set (or is clear in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000000000 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000010000 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the interrupt event flag is set (or is clear in syntax 2).

**Explanation:**

JINT or JNINT checks the cog's interrupt-occurred flag and jumps to the address described by Src if the flag is set (in syntax 1) or is clear (in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JINT / JNINT. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## 

## JPAT / JNPAT

Jump if pattern or not pattern

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if pin pattern event flag set or clear.

JPAT {#}*Src*

JNPAT {#}*Src*

**Result:** If pin pattern event flag is set (or is clear in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000001000 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000011000 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the pin pattern event flag is set (or is clear in syntax 2).

**Explanation:**

JPAT or JNPAT checks the cog's pin-pattern-detected event flag and jumps to the address described by Src if the flag is set (in syntax 1) or is clear (in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JPAT / JNPAT. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## JQMT / JNQMT

Jump if CORDIC empty or not CORDIC empty

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if CORDIC-read-but-empty event flag set or clear.

JQMT {#}*Src*

JNQMT {#}*Src*

**Result:** If CORDIC-read-but-empty event flag is set (or is clear in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000001111 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000011111 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the CORDIC-read-but-empty event flag is set (or is clear in syntax 2).

**Explanation:**

JQMT or JNQMT checks the cog's CORDIC results event flag and jumps to the address described by Src if the flag is set (in syntax 1) or is clear (in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JQMT / JNQMT. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## JSE1/2/3/4 / JNSE1/2/3/4

Jump if selectable event 1/2/3/4 or not selectable event 1/2/3/4

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if selectable event 1, 2, 3, or 4 flag is set or clear.

JSE1 {#}*Src*

JSE2 {#}*Src*

JSE3 {#}*Src*

JSE4 {#}*Src*

JNSE1 {#}*Src*

JNSE2 {#}*Src*

JNSE3 {#}*Src*

JNSE4 {#}*Src*

**Result:** If selectable event 1, 2, 3, or 4 flag is set (or is clear in syntax 5–8), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000000100 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000000101 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000000110 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000000111 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000010100 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000010101 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000010110 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000010111 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the selectable event 1, 2, 3, or 4 flag is set (or is clear in syntax 5–8).

**Explanation:**

JSE1, JSE2, JSE3, JSE4, or JNSE1, JNSE2, JNSE3, JNSE4 checks the cog's selectable event 1, 2, 3, or 4 event flag and jumps to the address described by Src if the flag is set (in syntax 1–4) or is clear (in syntax 5–8).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JSEx / JNSEx. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## 

## JXFI / JNXFI

Jump if streamer finished or not streamer finished

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if streamer finished event flag set or clear.

JXFI {#}*Src*

JNXFI {#}*Src*

**Result:** If streamer finished event flag is set (or is clear in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000001011 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000011011 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the streamer finished event flag is set (or is clear in syntax 2).

**Explanation:**

JXFI or JNXFI checks the cog's streamer finished event flag and jumps to the address described by Src if the flag is set (in syntax 1) or is clear (in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JXFI / JNXFI. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## JXMT / JNXMT

Jump if streamer empty or not streamer empty

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if streamer empty event flag set or clear.

JXMT {#}*Src*

JNXMT {#}*Src*

**Result:** If streamer empty event flag is set (or is clear in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000001010 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000011010 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the streamer empty event flag is set (or is clear in syntax 2).

**Explanation:**

JXMT or JNXMT checks the cog's streamer empty event flag and jumps to the address described by Src if the flag is set (in syntax 1) or is clear (in syntax 2). The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JXMT / JNXMT. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## JXRL / JNXRL

Jump if streamer rollover LUT or not streamer rollover LUT

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if streamer LUT RAM rollover event flag set or clear.

JXRL {#}*Src*

JNXRL {#}*Src*

**Result:** If streamer LUT RAM rollover event flag is set (or is clear in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000001101 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000011101 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the streamer LUT RAM rollover event flag is set (or is clear in syntax 2).

**Explanation:**

JXRL or JNXRL checks the cog's streamer LUT RAM rollover event flag and jumps to the address described by Src if the flag is set (in syntax 1) or is clear (in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JXRL / JNXRL. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## JXRO / JNXRO

Jump if streamer rollover NCO or not streamer rollover NCO

[**Event Branch Instruction**](#kix.py0qijsca5b1) **-** Jump if streamer NCO rollover event flag set or clear.

JXRO {#}*Src*

JNXRO {#}*Src*

**Result:** If streamer NCO rollover event flag is set (or is clear in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 01I | 000001100 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011110 | 01I | 000011100 | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when the streamer NCO rollover event flag is set (or is clear in syntax 2).

**Explanation:**

JXRO or JNXRO checks the cog's streamer NCO rollover event flag and jumps to the address described by Src if the flag is set (in syntax 1) or is clear (in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the JXRO / JNXRO. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## MODC / MODZ / MODCZ

Modify C, Z, or C and Z

[**Flag Modification Instruction**](#_hvsfxkjbocy8) **-** Modify C and/or Z flag(s) according to modifier mode and current state(s).

MODC *CModifier* {WC}

MODZ *ZModifier* {WZ}

MODCZ *CModifier*, *ZModifier* {WC|WZ|WCZ}

**Result:** The C and/or Z flag is set or cleared according to the given Modifier and the current state of the C and/or Z flags.

* CModifer is a Modifier symbol for the designated mode to apply to the C flag.
* ZModifer is a Modifier symbol for the designated mode to apply to the Z flag.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | C01 | 0cccc0000 | 001101111 | — | C=cccc[{C,Z}] | — | 2 |
| EEEE | 1101011 | 0Z1 | 00000zzzz | 001101111 | — | — | Z=zzzz[{C,Z}] | 2 |
| EEEE | 1101011 | CZ1 | 0cccczzzz | 001101111 | — | C=cccc[{C,Z}] | Z=zzzz[{C,Z}] | 2 |

**Explanation:**

MODC, MODZ, or MODCZ sets or clears the C and/or Z flag based on the mode described by the given Modifier symbol(s) and the current state of the C and/or Z flag. The WC, WZ, and WCZ effects are required to affect the designated flag.

These flag modifier instructions allow code to preset flags to a desired state which may be required for entry into certain code routines, or to set a special state based on multiple events that are otherwise not possible to realize with a single instruction.

If the WC, WZ, or WCZ effect is specified, the C, Z, or both C and Z flags are updated according to the given CModifier or ZModifier. See the Modifier Symbols table for details.

| Modifier Symbols (for MODC / MODZ / MODCZ) | | |
| --- | --- | --- |
| **Modifier1** | **Encoding2** | **Description** |
| \_CLR | %0000 | Clear C/Z (C == 0 and/or Z == 0) |
| \_E | %1010 | Set C/Z if comparison/subtraction was equal (C == Z and/or Z == Z) |
| \_NE | %0101 | Set C/Z if comparison/subtraction was not equal (C == !Z and/or Z == !Z) |
| \_GT | %0001 | Set C/Z if comparison/subtraction was greater than (C == !C AND !Z and/or Z == !C AND !Z) |
| \_GE | %0011 | Set C/Z if comparison/subtraction was greater than or equal (C == !C and/or Z == !C) |
| \_LT | %1100 | Set C/Z if comparison/subtraction was less than (C == C and/or Z == C) |
| \_LE | %1110 | Set C/Z if comparison/subtraction was less than or equal (C == C OR Z and/or Z == C OR Z) |
| \_C | %1100 | Set C/Z to C (C == C and/or Z == C) |
| \_NC | %0011 | Set C/Z to inverse of C (C == !C and/or Z == !C) |
| \_Z | %1010 | Set C/Z to Z (C == Z and/or Z == Z) |
| \_NZ | %0101 | Set C/Z to inverse of Z (C == !Z and/or Z == !Z) |
| \_C\_EQ\_Z | %1001 | Set C/Z if C equal to Z (C == C = Z and/or Z == C = Z) |
| \_C\_NE\_Z | %0110 | Set C/Z if C not equal to Z (C == C <> Z and/or Z == C <> Z) |
| \_C\_AND\_Z | %1000 | Set C/Z to C *AND* Z (C == C AND Z and/or Z == C AND Z) |
| \_C\_AND\_NZ | %0100 | Set C/Z to C *AND* *NOT* Z (C == C AND !Z and/or Z == C AND !Z) |
| \_NC\_AND\_Z | %0010 | Set C/Z to *NOT* C *AND* Z (C == !C AND Z and/or Z == !C AND Z) |
| \_NC\_AND\_NZ | %0001 | Set C/Z to *NOT* C *AND* *NOT* Z (C == !C AND !Z and/or Z == !C AND !Z) |
| \_C\_OR\_Z | %1110 | Set C/Z to C *OR* Z (C == C OR Z and/or Z == C OR Z) |
| \_C\_OR\_NZ | %1101 | Set C/Z to C *OR* *NOT* Z (C == C OR !Z and/or Z == C OR !Z) |
| \_NC\_OR\_Z | %1011 | Set C/Z to *NOT* C *OR* Z (C == !C OR Z and/or Z == !C OR Z) |
| \_NC\_OR\_NZ | %0111 | Set C/Z to *NOT* C *OR* *NOT* Z (C == !C OR !Z and/or Z == !C OR !Z) |
| \_Z\_EQ\_C | %1001 | Set C/Z if Z equal to C (C == Z = C and/or Z == Z = C) |
| \_Z\_NE\_C | %0110 | Set C/Z if Z not equal to C (C == Z <> C and/or Z == Z <> C) |
| \_Z\_AND\_C | %1000 | Set C/Z to Z *AND* C (C == Z AND C and/or Z == Z AND C) |
| \_Z\_AND\_NC | %0010 | Set C/Z to Z *AND* *NOT* C (C == Z AND !C and/or Z == Z AND !C) |
| \_NZ\_AND\_C | %0100 | Set C/Z to *NOT* Z *AND* C (C == !Z AND C and/or Z == !Z AND C) |
| \_NZ\_AND\_NC | %0001 | Set C/Z to *NOT* Z *AND* *NOT* C (C == !Z AND !C and/or Z == !Z AND !C) |
| \_Z\_OR\_C | %1110 | Set C/Z to Z *OR* C (C == Z OR C and/or Z == Z OR C) |
| \_Z\_OR\_NC | %1011 | Set C/Z to Z *OR* *NOT* C (C == Z OR !C and/or Z == Z OR !C) |
| \_NZ\_OR\_C | %1101 | Set C/Z to *NOT* Z *OR* C (C == !Z OR C and/or Z == !Z OR C) |
| \_NZ\_OR\_NC | %0111 | Set C/Z to *NOT* Z *OR* *NOT* C (C == !Z OR !C and/or Z == !Z OR !C) |
| \_SET | %1111 | Set C/Z (C == 1 and/or Z == 1) |

1 Use Modifier symbol(s) in MODC, MODZ, and MODCZ instructions. Note that the symbol and description is similar to, and the encoding exactly matches, that of the related [Conditions](#_ofohv8f4qicc) used to include/exclude instructions at run time.

2 The encoding is the 4-bit value placed into the MODC, MODZ, and MODCZ instruction's cccc or zzzz opcode field by the compiler.

**Examples:**

MODCZ \_NZ, 0 WC 'C = !Z, Z = Z (unaltered)

MODCZ 0, \_SET WZ 'C = C (unaltered), Z = 1

MODCZ \_CLR, \_Z\_OR\_C WCZ 'C = 0, Z |= C

MODC \_NZ\_AND\_C WC 'C = !Z & C

MODZ \_Z\_NE\_C WZ 'Z = Z ^ C; (Z <> C)

## MOV

Move

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set a value into a register.

MOV *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The Src value is stored in Dest and optionally flags are updated with the sign bit and zero status.

* Dest is the register to receive the Src value.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value will be stored into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0110000 | CZI | DDDDDDDDD | SSSSSSSSS | D | S[31] | Result = 0 | 2 |

**Explanation:**

Mov stores the Src value into Dest.

If the WC or WCZ effect is specified, the C flag is updated to be Src[31].

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest result equals zero, or is cleared (0) if it is non-zero.

## MUL

Multiply

[**Math Instruction**](#_xnk9pb8j5sof) **-** Multiply unsigned 16-bit x 16-bit values.

MUL *Dest*, {#}*Src* {WZ}

**Result:** The 32-bit unsigned product of the 16-bit Dest and Src multiplication is stored into Dest and optionally the Z flag is updated to the Dest or Src zero status.

* Dest is a register containing the 16-bit value to multiply with Src, and is where the result is written.
* Src is a register, 9-bit literal, or 16-bit augmented literal whose value is multiplied into Dest.
* WZ is an optional effect to update the Z flag.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1010000 | 0ZI | DDDDDDDDD | SSSSSSSSS | D | — | (D = 0) | (S = 0) | 2 |

**Related:** [MULS](#_9awkxilhgl30), SCA, and QMUL

**Explanation:**

MUL multiplies the lower 16-bits of each of Dest and Src together and stores the 32-bit product result into the Dest register. This is a fast (2-clock) 16 x 16 bit multiplication operation— to multiply larger factors, use the CORDIC Solver QMUL instruction.

If the WZ effect is specified, the Z flag is set (1) if either the Dest or Src values are zero, or is cleared (0) if both are non-zero.

## MULS

Multiply, signed

[**Math Instruction**](#_xnk9pb8j5sof) **-** Multiply signed 16-bit x 16-bit values.

MULS *Dest*, {#}*Src* {WZ}

**Result:** The 32-bit signed product of the signed 16-bit Dest and Src multiplication is stored into Dest and optionally the Z flag is updated to the Dest or Src zero status.

* Dest is a register containing the signed 16-bit value to multiply with Src, and is where the result is written.
* Src is a register, 9-bit literal, or signed 16-bit augmented literal whose value is multiplied into Dest.
* WZ is an optional effect to update the Z flag.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1010000 | 1ZI | DDDDDDDDD | SSSSSSSSS | D | — | (D = 0) | (S = 0) | 2 |

**Related:** [MUL](#_wv511llidyd6), SCA, and QMUL

**Explanation:**

MULS multiplies the signed lower 16-bits of each of Dest and Src together and stores the 32-bit signed product result into the Dest register. This is a fast (2-clock) signed 16 x 16 bit multiplication operation— to multiply larger factors, use the CORDIC Solver QMUL instruction.

If the WZ effect is specified, the Z flag is set (1) if either the Dest or Src values are zero, or is cleared (0) if both are non-zero.

## MUXC / MUXNC

Mux C or mux not C

[**Bit Operation Instructions**](#_1ykf5b4dvpz7) **-** Set discrete bits to C or !C

MUXC *Dest*, {#}*Src* {WC|WZ|WCZ}

MUXNC *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Dest bit(s) described by Src are set to C or !C; the rest are left as-is. Flags are optionally updated with *parity* and *zero* status of the result.

* Dest is the register whose value will have one or more bits set to C or !C.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value identifies the bit(s) to modify.
* WC, WZ, or WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0101100 | CZI | DDDDDDDDD | SSSSSSSSS | D | Parity of Result | Result = 0 | 2 |
| EEEE | 0101101 | CZI | DDDDDDDDD | SSSSSSSSS | D | Parity of Result | Result = 0 | 2 |

**Explanation:**

MUXC or MUXNC alters the Dest bit(s) designated by Src (high bits) to equal the state, or inverse state, of the C flag. All Dest bits corresponding to high (1) bits in Src are modified; all other Dest bits are left unchanged.

If the WC or WCZ effect is specified, the C flag is set (1) if the number of high (1) bits in the result is odd, or is cleared (0) if it is even.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is not zero.

## MUXNIBS

Mux nibbles

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set discrete nibbles of a value to non-zero nibble states of another.

MUXNIBS *Dest*, {#}*Src*

**Result:** Dest nibbles corresponding to non-zero Src nibbles are set to those nibble values; the rest are left as-is.

* Dest is a register whose value will be updated according to Src.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose non-zero nibbles will replace the corresponding nibbles in Dest.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001111 | 01I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |

**Explanation:**

MUXNIBS copies any non-zero nibbles from Src into the corresponding nibbles of Dest and leaves the rest of Dest's nibbles as-is.

## MUXNITS

Mux nits

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set discrete bit pairs of a value to non-zero bit pair states of another.

MUXNITS *Dest*, {#}*Src*

**Result:** Dest bit pairs corresponding to non-zero Src bit pairs are set to those bit values; the rest are left as-is.

* Dest is a register whose value will be updated according to Src.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose non-zero bit pairs will replace the corresponding bit pairs in Dest.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001111 | 00I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |

**Explanation:**

MUXNITS copies any non-zero bit pairs from Src into the corresponding bit pairs of Dest and leaves the rest of Dest's bit pairs as-is.

## MUXQ

Mux Q

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set discrete bits of a value to that of another.

MUXQ *Dest*, {#}*Src*

**Result:** Dest bits described by Q are set to corresponding Src bits; the rest are left as-is.

* Dest is a register whose bits will be updated according to Q and Src.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose select bits will replace the corresponding bits in Dest.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001111 | 10I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |

**Explanation:**

MUXQ copies all bits from Src corresponding to high (1) bits of Q into the corresponding bits of Dest. All other Dest bits are left as-is.

MUXQ must be preceded by SETQ to function properly since the Q value's high bits identify the bits to target in Src and Dest.

## MUXZ / MUXNZ

Mux Z or mux not Z

[**Bit Operation Instructions**](#_1ykf5b4dvpz7) **-** Set discrete bits to Z or !Z

MUXZ *Dest*, {#}*Src* {WC|WZ|WCZ}

MUXNZ *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Dest bit(s) described by Src are set to Z or !Z; the rest are left as-is. Flags are optionally updated with *parity* and *zero* status of the result.

* Dest is the register whose value will have one or more bits set to Z or !Z.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value identifies the bit(s) to modify.
* WC, WZ, or WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0101110 | CZI | DDDDDDDDD | SSSSSSSSS | D | Parity of Result | Result = 0 | 2 |
| EEEE | 0101111 | CZI | DDDDDDDDD | SSSSSSSSS | D | Parity of Result | Result = 0 | 2 |

**Explanation:**

MUXZ or MUXNZ alters the Dest bit(s) designated by Src (high bits) to equal the state, or inverse state, of the Z flag. All Dest bits corresponding to high (1) bits in Src are modified; all other Dest bits are left unchanged.

If the WC or WCZ effect is specified, the C flag is set (1) if the number of high (1) bits in the result is odd, or is cleared (0) if it is even.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is not zero.

## NEG

Negate

[**Math Instruction**](#_xnk9pb8j5sof) **-** Negate a value.

NEG *Dest*, {#}*Src* {WC|WZ|WCZ}

NEG *Dest* {WC|WZ|WCZ}

**Result:** The Src or Dest value is negated and stored into Dest and optionally the C and Z flags are updated to the resulting sign and zero status.

* Dest is a register to receive the -Src value (syntax 1), or contains the value to negate (syntax 2).
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose negated value is stored into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0110011 | CZI | DDDDDDDDD | SSSSSSSSS | D | Sign of result | Result = 0 | 2 |
| EEEE | 0110011 | CZ0 | DDDDDDDDD | DDDDDDDDD | D | Sign of result | Result = 0 | 2 |

**Related:** [ABS](#_lrvakeitwivz), [NEGC](#_jlhwuat7bnkq), [NEGNC](#_323cveis97t7), [NEGZ](#_5vu7vu1xc9tm) and [NEGNZ](#_z1ulw37qnpt4)

**Explanation:**

NEG negates Src (syntax 1) or Dest (syntax 2) and stores the result in the Dest register. The *negation* flips the value's sign; ex: 78 becomes -78, or -306 becomes 306.

If the WC or WCZ effect is specified, the C flag is set (1) if the result is negative, or is cleared (0) if positive.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## NEGC

Negate C

[**Math Instruction**](#_xnk9pb8j5sof) **-** Negate value according to C.

NEGC *Dest*, {#}*Src* {WC|WZ|WCZ}

NEGC *Dest* {WC|WZ|WCZ}

**Result:** The Src or Dest value, possibly negated according to C, is stored into Dest and optionally the C and Z flags are updated to the resulting sign and zero status.

* Dest is a register to receive the Src or -Src value (syntax 1), or contains the value to negate (syntax 2) according to C.
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose value (if C=0) or negated value (if C=1) is stored into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0110100 | CZI | DDDDDDDDD | SSSSSSSSS | D | Sign of result | Result = 0 | 2 |
| EEEE | 0110100 | CZ0 | DDDDDDDDD | DDDDDDDDD | D | Sign of result | Result = 0 | 2 |

**Related:** [NEGNC](#_323cveis97t7), [NEGZ](#_5vu7vu1xc9tm) and [NEGNZ](#_z1ulw37qnpt4)

**Explanation:**

NEGC negates Src (syntax 1) or Dest (syntax 2) if C = 1 and stores the result in the Dest register. If C = 0, the Src or Dest value is left as-is (not negated) and is stored into Dest. If the *negation* is performed, it flips the value's sign; ex: 5 becomes -5, or -200 becomes 200.

If the WC or WCZ effect is specified, the C flag is set (1) if the result is negative, or is cleared (0) if positive.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## NEGNC

Negate not C

[**Math Instruction**](#_xnk9pb8j5sof) **-** Negate value according to !C.

NEGNC *Dest*, {#}*Src* {WC|WZ|WCZ}

NEGNC *Dest* {WC|WZ|WCZ}

**Result:** The Src or Dest value, possibly negated according to !C, is stored into Dest and optionally the C and Z flags are updated to the resulting sign and zero status.

* Dest is a register to receive the Src or -Src value (syntax 1), or contains the value to negate (syntax 2) according to !C.
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose value (if !C=0) or negated value (if !C=1) is stored into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0110101 | CZI | DDDDDDDDD | SSSSSSSSS | D | Sign of result | Result = 0 | 2 |
| EEEE | 0110101 | CZ0 | DDDDDDDDD | DDDDDDDDD | D | Sign of result | Result = 0 | 2 |

**Related:** [NEGC](#_jlhwuat7bnkq), [NEGZ](#_5vu7vu1xc9tm) and [NEGNZ](#_z1ulw37qnpt4)

**Explanation:**

NEGNC negates Src (syntax 1) or Dest (syntax 2) if !C = 1 and stores the result in the Dest register. If !C = 0, the Src or Dest value is left as-is (not negated) and is stored into Dest. If the *negation* is performed, it flips the value's sign; ex: 21 becomes -21, or -1,374 becomes 1,374.

If the WC or WCZ effect is specified, the C flag is set (1) if the result is negative, or is cleared (0) if positive.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## NEGNZ

Negate not Z

[**Math Instruction**](#_xnk9pb8j5sof) **-** Negate value according to !Z.

NEGNZ *Dest*, {#}*Src* {WC|WZ|WCZ}

NEGNZ *Dest* {WC|WZ|WCZ}

**Result:** The Src or Dest value, possibly negated according to !Z, is stored into Dest and optionally the C and Z flags are updated to the resulting sign and zero status.

* Dest is a register to receive the Src or -Src value (syntax 1), or contains the value to negate (syntax 2) according to !C.
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose value (if !C=0) or negated value (if !C=1) is stored into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0110111 | CZI | DDDDDDDDD | SSSSSSSSS | D | Sign of result | Result = 0 | 2 |
| EEEE | 0110111 | CZ0 | DDDDDDDDD | DDDDDDDDD | D | Sign of result | Result = 0 | 2 |

**Related:** [NEGZ](#_5vu7vu1xc9tm), [NEGC](#_jlhwuat7bnkq) and [NEGNC](#_323cveis97t7)

**Explanation:**

NEGNZ negates Src (syntax 1) or Dest (syntax 2) if !Z = 1 and stores the result in the Dest register. If !Z = 0, the Src or Dest value is left as-is (not negated) and is stored into Dest. If the *negation* is performed, it flips the value's sign; ex: 193 becomes -193, or -3,062 becomes 3,062.

If the WC or WCZ effect is specified, the C flag is set (1) if the result is negative, or is cleared (0) if positive.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## NEGZ

Negate Z

[**Math Instruction**](#_xnk9pb8j5sof) **-** Negate value according to Z.

NEGZ *Dest*, {#}*Src* {WC|WZ|WCZ}

NEGZ *Dest* {WC|WZ|WCZ}

**Result:** The Src or Dest value, possibly negated according to Z, is stored into Dest and optionally the C and Z flags are updated to the resulting sign and zero status.

* Dest is a register to receive the Src or -Src value (syntax 1), or contains the value to negate (syntax 2) according to Z.
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose value (if Z=0) or negated value (if Z=1) is stored into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0110110 | CZI | DDDDDDDDD | SSSSSSSSS | D | Sign of result | Result = 0 | 2 |
| EEEE | 0110110 | CZ0 | DDDDDDDDD | DDDDDDDDD | D | Sign of result | Result = 0 | 2 |

**Related:** [NEGNZ](#_z1ulw37qnpt4), [NEGC](#_jlhwuat7bnkq) and [NEGNC](#_323cveis97t7)

**Explanation:**

NEGZ negates Src (syntax 1) or Dest (syntax 2) if Z = 1 and stores the result in the Dest register. If Z = 0, the Src or Dest value is left as-is (not negated) and is stored into Dest. If the *negation* is performed, it flips the value's sign; ex: 526 becomes -526, or -41 becomes 41.

If the WC or WCZ effect is specified, the C flag is set (1) if the result is negative, or is cleared (0) if positive.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## NOP

No operation

[**Miscellaneous Instruction**](#_1gzr437m04f) **-** No operation, just elapse two cycles.

NOP

**Result:** Two clock cycles are consumed.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0000 | 0000000 | 000 | 000000000 | 000000000 | — | — | — | 2 |

**Explanation:**

NOP simply consumes two clock cycles; no other operation is performed.

## NOT

Not

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Bitwise NOT a value.

NOT *Dest*, {#}*Src* {WC|WZ|WCZ}

NOT *Dest* {WC|WZ|WCZ}

**Result:** !Src (or !Dest) is stored in Dest and flags are optionally updated with the NOTed high bit and *zero* status.

* Dest is the register containing the value to bitwise NOT (syntax 2) or to be replaced by the bitwise NOT of Src (syntax 1).
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose value will be bitwise NOTed and stored into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0110001 | CZI | DDDDDDDDD | SSSSSSSSS | D | !S[31] | Result = 0 | 2 |
| EEEE | 0110001 | CZ0 | DDDDDDDDD | DDDDDDDDD | D | !D[31] | Result = 0 | 2 |

**Explanation:**

NOT performs a bitwise NOT (inverting all bits) of the value in Src (or in Dest) and stores the result into Dest.

If the WC or WCZ effect is specified, the C flag value is replaced by the inverse of either S[31] (syntax 1) or D[31] (syntax 2).

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result of !Src (syntax 1) or !Dest (syntax 2) equals zero, or is cleared (0) if it is non-zero.

## OR

Or

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Bitwise OR a value with another.

OR *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Dest OR Src is stored in Dest and flags are optionally updated with *parity* and *zero* status.

* Dest is the register containing the value to bitwise OR with Src and is the destination in which to write the result.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value will be bitwise ORed into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0101010 | CZI | DDDDDDDDD | SSSSSSSSS | D | Parity of Result | Result = 0 | 2 |

**Explanation:**

OR performs a bitwise OR of the value in Src into that of Dest.

If the WC or WCZ effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits, or is cleared (0) if it contains an even number of high bits.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest OR Src result equals zero, or is cleared (0) if it is non-zero.

## ONES

Ones

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Get number of 1s from Dest or Src into Dest.

ONES *Dest*, {#}*Src* {WC|WZ|WCZ}

ONES *Dest* {WC|WZ|WCZ}

**Result:** The number of high bits in Src, or Dest, is stored in Dest.

* Dest is the register in which to store the number of high bits found and optionally it begins by containing the value to check (syntax 2).
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose value is to be checked for ones.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0111101 | CZI | DDDDDDDDD | SSSSSSSSS | D | Result is odd | Result = 0 | 2 |
| EEEE | 0111101 | CZ0 | DDDDDDDDD | DDDDDDDDD | D | Result is odd | Result = 0 | 2 |

**Explanation:**

ONES tallies the number of high bits of Src, or Dest, and stores the count into Dest.

If the WC or WCZ effect is specified, the C flag is set (1) if the count is odd, or is cleared (0) if it is even.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result equals zero, or is cleared (0) if not zero.

## OUTC / OUTNC

Output C or not C

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) output level to low/high according to C or !C.

OUTC {#}*Dest* {WCZ}

OUTNC {#}*Dest* {WCZ}

**Result:** The I/O pin output level bit(s), described by Dest, are set to low/high according to C or !C; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) for which output levels are to be set low or high.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001001010 | OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001001011 | OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

**Explanation:**

OUTC or OUTNC alters the output level register's bit(s) designated by Dest to equal the state, or inverse state, of the C flag; i.e. set pin's output level low or high. All other bits (pins) are left unchanged. Each of these instructions, OUTC and OUTNC, can affect one or more of the bits within the OUTA or OUTB registers.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the OUTC / OUTNC instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for OUTC / OUTNC's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## OUTH / OUTL

Output high or low

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) output level to high (1) or low (0).

OUTH {#}*Dest* {WCZ}

OUTL {#}*Dest* {WCZ}

**Result:** The I/O pin output level bit(s), described by Dest, are set high (1) or low (0); the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to set high or low.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001001001 | OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001001000 | OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

**Explanation:**

OUTH or OUTL alters the output level register's bit(s) designated by Dest to be high (1) or low (0). All other bits (pins) are left unchanged. Each of these instructions, OUTH and OUTL, can affect one or more of the bits within the OUTA or OUTB registers.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the OUTH / OUTL instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for OUTH / OUTL's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## OUTNOT

Output not

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Toggle pin(s) to the opposite output level.

OUTNOT {#}*Dest* {WCZ}

**Result:** The I/O pin output level bit(s), described by Dest, are toggled to their opposite state(s); the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) to toggle to the opposite output level.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001001111 | OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

**Explanation:**

OUTNOT alters the output level register's bit(s) designated by Dest to their inverse state. All other bits are left unchanged.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the OUTNOT instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for OUTNOT's use.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## OUTRND

Output random

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) output level to random low/high.

OUTRND {#}*Dest* {WCZ}

**Result:** The I/O pin output level bit(s), described by Dest, are each set randomly low or high; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the pins set randomly to low or high.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001001110 | OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

**Explanation:**

OUTRND alters the output level register's bit(s) designated by Dest to be random low and high, based on bit(s) from the Xoroshiro128\*\* PRNG. All other bits are left unchanged.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the OUTRND instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for OUTRND's use.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## OUTZ / OUTNZ

Output Z or not Z

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Set pin(s) output level to low/high according to Z or !Z.

OUTZ {#}*Dest* {WCZ}

OUTNZ {#}*Dest* {WCZ}

**Result:** The I/O pin output level bit(s), described by Dest, are set to low/high according to Z or !Z; the rest are left as-is.

* Dest is the register, 9-bit literal, or 11-bit augmented literal whose value identifies the I/O pin(s) for which output levels are to be set low or high.
* WCZ is an optional effect to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001001100 | OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001001101 | OUTx | Orig OUTx base bit | Orig OUTx base bit | 2 |

**Explanation:**

OUTZ or OUTNZ alters the output level register's bit(s) designated by Dest to equal the state, or inverse state, of the Z flag; i.e. set pin's output level low or high. All other bits (pins) are left unchanged. Each of these instructions, OUTZ and OUTNZ, can affect one or more of the bits within the OUTA or OUTB registers.

Dest[5:0] indicates the pin number (0–63). For a range of pins, Dest[5:0] indicates the base pin number (0–63) and Dest[10:6] indicates how many contiguous pins beyond the base should be affected (1–31).

A 9-bit literal Dest is enough to express the base pin (Dest[5:0]) and a range of up to 8 contiguous pins (Dest[8:6]). If needed, use the augmented literal feature (##Dest) to augment Dest to an 11-bit literal value— this inserts an AUGD instruction prior.

When Dest is a register, the register's value bits [10:0] are used as-is to form the 11-bit ID range, unless a SETQ instruction immediately precedes the OUTZ / OUTNZ instruction; substituting SETQ's Dest[4:0] in place of value bits[10:6], for OUTZ / OUTNZ's use.

The range calculation (from Dest[5:0] up to Dest[5:0]+Dest[10:6]) will wrap within the same 32-pin group (OUTA or OUTB); it will not cross the port boundary.

If the WCZ effect is specified, the C and Z flags are updated to the original state of OUTA / OUTB's base bit, identified by Dest.

## POLLATN

Poll attention

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear attention flag.

POLLATN {WC|WZ|WCZ}

**Result:** Attention event flag is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000001110 | 000100100 | — | ATN Event | ATN Event | 2 |

**Related:** [COGATN](#_wx29grvrexs), [WAITATN](#_sb5186rnvudg), [JATN](#_yfon3snogp62), and [JNATN](#_yfon3snogp62)

**Explanation:**

POLLATN copies the state of the attention event flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the attention event flag prior to clearing it.

The attention event flag is set whenever another cog issues an attention request for this cog. The attention event flag is cleared upon cog start, or execution of POLLATN, WAITATN, JATN, or JNATN instructions.

## POLLCT1/2/3

Poll counter 1/2/3

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear counter event 1/2/3 flag.

POLLCT1 {WC|WZ|WCZ}

POLLCT2 {WC|WZ|WCZ}

POLLCT3 {WC|WZ|WCZ}

**Result:** Counter event flag 1, 2, or 3 state is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000000001 | 000100100 | — | CT1 Event | CT1 Event | 2 |
| EEEE | 1101011 | CZ0 | 000000010 | 000100100 | — | CT2 Event | CT2 Event | 2 |
| EEEE | 1101011 | CZ0 | 000000011 | 000100100 | — | CT3 Event | CT3 Event | 2 |

**Related:** [ADDCTx](#_u2xzjautl27l), [WAITCTx](#_u335ct3ffsan), [JCTx](#_6f897k4gldkf), and [JNCTx](#_6f897k4gldkf)

**Explanation:**

POLLCT1, POLLCT2, or POLLCT3 copies the state of the count 1, 2, or 3 event flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the corresponding counter event flag prior to clearing it.

The counter 1, 2, or 3 event flag is set whenever the System Counter (CT) passes the value in the CT1, CT2, or CT3 event trigger register, respectively; i.e. MSB of (CT - CTx is 0). The counter event flags are cleared upon execution of the corresponding ADDCTx, POLLCTx, WAITCTx, JCTx, or JNCTx instructions.

## POLLFBW

Poll FIFO block wrap

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear FIFO-interface-block-wrap event flag.

POLLFBW {WC|WZ|WCZ}

**Result:** FIFO-interface-block-wrap event flag is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000001001 | 000100100 | — | FBW Event | FBW Event | 2 |

**Related:** RDFAST, WRFAST, FBLOCK, [WAITFBW](#_qoevcy1s6y48), [JFBW](#_miks298accr1), and [JNFBW](#_miks298accr1)

**Explanation:**

POLLFBW copies the state of the FIFO-interface-block-wrap event flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the FIFO-interface‑block‑ wrap event flag prior to clearing it.

The FIFO-interface-block-wrap event flag is set whenever the Hub RAM FIFO interface exhausts its block count and reloads its *block count* and *start address*. The FIFO-interface-block-wrap event flag is cleared upon execution of RDFAST, WRFAST, FBLOCK, POLLFBW, WAITFBW, JFBW, or JNFBW instructions.

## POLLINT

Poll interrupt

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear interrupt-occurred event flag.

POLLINT {WC|WZ|WCZ}

**Result:** Interrupt-occurred event flag is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000000000 | 000100100 | — | INT Event | INT Event | 2 |

**Related:** [WAITINT](#_rqj7wu7kf26o), [JINT](#_wzi7hl4t1ffr), and [JNINT](#_wzi7hl4t1ffr)

**Explanation:**

POLLINT copies the state of the interrupt-occurred event flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the interrupt-occurred event flag prior to clearing it.

The interrupt-occurred event flag is set whenever interrupt 1, 2, or 3 occurs— debug interrupts are ignored. The interrupt-occurred event flag is cleared upon cog start, or execution of POLLINT, WAITINT, JINT, or JNINT instructions.

## POLLPAT

Poll pattern

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear pin-pattern-detected event flag.

POLLPAT {WC|WZ|WCZ}

**Result:** Pin-pattern-detected event flag is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000001000 | 000100100 | — | PAT Event | PAT Event | 2 |

**Related:** SETPAT, [WAITPAT](#_5i1v8ffdcpk2), [JPAT](#_pnmnqy72ibwb), and [JNPAT](#_pnmnqy72ibwb)

**Explanation:**

POLLPAT copies the state of the pin-pattern-detected event flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the pin-pattern-detected event flag prior to clearing it.

The pin-pattern-detected event flag is set whenever the masked input pins match or don't match the pattern described by a previous SETPAT instruction. The pin-pattern-detected event flag is cleared upon execution of SETPAT, POLLPAT, WAITPAT, JPAT, or JNPAT instructions.

## POLLQMT

Poll CORDIC empty

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear CORDIC-read-but-empty event flag.

POLLQMT {WC|WZ|WCZ}

**Result:** CORDIC-read-but-empty event flag is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000001111 | 000100100 | — | QMT Event | QMT Event | 2 |

**Related:** [JQMT](#_fik5o0ebifr2) and [JNQMT](#_fik5o0ebifr2)

**Explanation:**

POLLQMT copies the state of the CORDIC-read-but-empty event flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the CORDIC-read-but- empty event flag prior to clearing it.

The CORDIC-read-but-empty event flag is set whenever GETQX / GETQY executes without any CORDIC results available or in progress. The pin-pattern-detected event flag is cleared upon cog start or execution of POLLQMT, WAITQMT, JQMT, or JNQMT instructions.

## POLLSE1/2/3/4

Poll selectable event 1/2/3/4

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear selectable event 1, 2, 3, or 4 flag.

POLLSE1 {WC|WZ|WCZ}

POLLSE2 {WC|WZ|WCZ}

POLLSE3 {WC|WZ|WCZ}

POLLSE4 {WC|WZ|WCZ}

**Result:** Selectable event flag 1, 2, 3, or 4 state is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000000100 | 000100100 | — | SE1 Event | SE1 Event | 2 |
| EEEE | 1101011 | CZ0 | 000000101 | 000100100 | — | SE2 Event | SE2 Event | 2 |
| EEEE | 1101011 | CZ0 | 000000110 | 000100100 | — | SE3 Event | SE3 Event | 2 |
| EEEE | 1101011 | CZ0 | 000000111 | 000100100 | — | SE4 Event | SE4 Event | 2 |

**Related:** SETSEx, [WAITSEx](#_43d9knl3whqm), [JSEx](#_nv0zohzajupf), and [JNSEx](#_nv0zohzajupf)

**Explanation:**

POLLSE1, POLLSE2, POLLSE3, or POLLSE4 copies the state of the selectable event 1, 2, 3, or 4 flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the corresponding selectable event flag prior to clearing it.

The selectable event 1, 2, 3, or 4 flag is set whenever the corresponding configured event occurs. The selectable event flag is cleared upon execution of the corresponding SETSEx, POLLSEx, WAITSEx, JSEx, or JNSEx instructions.

## POLLXFI

Poll streamer finished

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear streamer-finished event flag.

POLLXFI {WC|WZ|WCZ}

**Result:** Streamer-finished event flag is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000001011 | 000100100 | — | XFI Event | XFI Event | 2 |

**Related:** XINIT, XZERO, XCONT, [WAITXFI](#_zdbnefgvlfr5), [JXFI](#_76euyh5v8o1z), and [JNXFI](#_76euyh5v8o1z)

**Explanation:**

POLLXFI copies the state of the streamer-finished event flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the streamer-finished event flag prior to clearing it.

The streamer-finished event flag is set whenever the streamer runs out of commands to process. The streamer-finished event flag is cleared upon execution of XINIT, XZERO, XCONT, POLLXFI, WAITXFI, JXFI, or JNXFI instructions.

## POLLXMT

Poll streamer empty

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear streamer-empty event flag.

POLLXMT {WC|WZ|WCZ}

**Result:** Streamer-empty event flag is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000001010 | 000100100 | — | XMT Event | XMT Event | 2 |

**Related:** XINIT, XZERO, XCONT, [WAITXMT](#_1ird9xvgzp82), [JXMT](#_izeyc7za5235), and [JNXMT](#_izeyc7za5235)

**Explanation:**

POLLXMT copies the state of the streamer-empty event flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the streamer-empty event flag prior to clearing it.

The streamer-empty event flag is set whenever the streamer is ready for a new command. The streamer-empty event flag is cleared upon execution of XINIT, XZERO, XCONT, POLLXMT, WAITXMT, JXMT, or JNXMT instructions.

## POLLXRL

Poll streamer rollover LUT

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear streamer-LUT-RAM-rollover event flag.

POLLXRL {WC|WZ|WCZ}

**Result:** Streamer-LUT-RAM-rollover event flag is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000001101 | 000100100 | — | XRL Event | XRLEvent | 2 |

**Related:** XINIT, XZERO, XCONT, [WAITXRL](#_nesp1ncmao0z), [JXRL](#_43ntogxc32ez), and [JNXRL](#_43ntogxc32ez)

**Explanation:**

POLLXRL copies the state of the streamer-LUT-RAM-rollover event flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the streamer-LUT-RAM- rollover event flag prior to clearing it.

The streamer-LUT-RAM-rollover event flag is set whenever location $1FF of the Lookup RAM is read by the streamer. The streamer-LUT-RAM-rollover event flag is cleared upon cog start or upon execution of POLLXRL, WAITXRL, JXRL, or JNXRL instructions.

## POLLXRO

Poll streamer rollover NCO

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Retrieve and clear streamer-NCO-rollover event flag.

POLLXRO {WC|WZ|WCZ}

**Result:** Streamer-NCO-rollover event flag is optionally copied into C and/or Z, then it is cleared.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000001100 | 000100100 | — | XRO Event | XRO Event | 2 |

**Related:** XINIT, XZERO, XCONT, [WAITXRO](#_y7dswdrnedgw), [JXRO](#_gut56vubumxk), and [JNXRO](#_gut56vubumxk)

**Explanation:**

POLLXRO copies the state of the streamer NCO rollover event flag into C and/or Z and then clears the flag (unless it's being set again by the event sensor).

If the WC, WZ, or WCZ effect is specified, the C flag and/or Z flag is updated to the state of the streamer-NCO- rollover event flag prior to clearing it.

The streamer-NCO-rollover event flag is set whenever the streamer's numerically-controlled oscillator (NCO) rolls over. The streamer-NCO-rollover event flag is cleared upon execution of XINIT, XZERO, XCONT, POLLXRO, WAITXRO, JXRO, or JNXRO instructions.

## RCL

Rotate carry left

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Rotate carry flag left into value.

RCL *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The bits of Dest are shifted left by Src bits, inserting C as new LSBs.

* Dest is the register containing the value to rotate carry left by Src bits.
* Src is a register or 5-bit literal whose value indicates the number of bit positions to rotate.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0000101 | CZI | DDDDDDDDD | SSSSSSSSS | D | Last bit out1 | Result = 0 | 2 |

1 C = last bit shifted out if Src[4:0] > 0, else C = Dest[31].

**Explanation:**

RCL shifts Dest's binary value left by Src places (0–31 bits) and sets the new LSBs to C.

If the WC or WCZ effect is specified, the C flag is updated to the value of the last bit shifted out if Src is 1–31, or to Dest[31] if Src is 0.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest result equals zero, or is cleared (0) if it is non-zero.

## RCR

Rotate carry right

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Rotate carry flag right into value.

RCR *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The bits of Dest are shifted right by Src bits, inserting C as new MSBs.

* Dest is the register containing the value to rotate carry right by Src bits.
* Src is a register or 5-bit literal whose value indicates the number of bit positions to rotate.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0000100 | CZI | DDDDDDDDD | SSSSSSSSS | D | Last bit out1 | Result = 0 | 2 |

1 C = last bit shifted out if Src[4:0] > 0, else C = Dest[0].

**Explanation:**

RCR shifts Dest's binary value right by Src places (0–31 bits) and sets the new MSBs to C.

If the WC or WCZ effect is specified, the C flag is updated to the value of the last bit shifted out if Src is 1–31, or to Dest[0] if Src is 0.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest result equals zero, or is cleared (0) if it is non-zero.

## RCZL

Rotate carry and zero left

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Rotate carry and zero flags left into value (2-bit rotate left).

RCZL *Dest* {WC|WZ|WCZ}

**Result:** The bits of Dest are shifted left by two places and C and Z are inserted as new LSBs.

* Dest is the register containing the value to rotate the carry and zero flags left into.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | DDDDDDDDD | 001101011 | D | D[31] | D[30] | 2 |

**Explanation:**

RCZL shifts Dest's binary value left by two places and sets Dest[1] to C and Dest[0] to Z.

If the WC or WCZ effect is specified, the C flag is updated to the original Dest[31] state.

If the WZ or WCZ effect is specified, the Z is flag is updated to the original Dest[30] state.

## RCZR

Rotate carry and zero right

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Rotate carry and zero flags right into value (2-bit rotate right).

RCZR *Dest* {WC|WZ|WCZ}

**Result:** The bits of Dest are shifted right by two places and C and Z are inserted as new MSBs.

* Dest is the register containing the value to rotate the carry and zero flags right into.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | DDDDDDDDD | 001101010 | D | D[1] | D[0] | 2 |

**Explanation:**

RCZR shifts Dest's binary value right by two places and sets Dest[31] to C and Dest[30] to Z.

If the WC or WCZ effect is specified, the C flag is updated to the original Dest[1] state.

If the WZ or WCZ effect is specified, the Z is flag is updated to the original Dest[0] state.

## Registers

[**Special Purpose Registers**](#_87ivzhp7ysag) **-** Rotate a byte left into a value.

Most Reg RAM registers are general-purpose (used for code and data) while a small set has a special purpose.

| **Name** | **Address** | **Access2** | **Special Purpose** |
| --- | --- | --- | --- |
| IJMP3 | $1F01 | Read/Write | Interrupt call address for INT3; set by code |
| IRET3 | $1F11 | Read/Write | Interrupt return address and C/Z flags for INT3; set by cog |
| IJMP2 | $1F21 | Read/Write | Interrupt call address for INT2; set by code |
| IRET2 | $1F31 | Read/Write | Interrupt return address and C/Z flags for INT2; set by cog |
| IJMP1 | $1F41 | Read/Write | Interrupt call address for INT1; set by code |
| IRET1 | $1F51 | Read/Write | Interrupt return address and C/Z flags for INT1; set by cog |
| PA | $1F61 | Read/Write | CALLD-imm return, CALLPA parameter, or LOC address |
| PB | $1F71 | Read/Write | CALLD-imm return, CALLPB parameter, or LOC address |
| PTRA | $1F8 | Read Special / Write Special | Pointer A to Hub RAM |
| PTRB | $1F9 | Read Special / Write Special | Pointer B to Hub RAM |
| DIRA | $1FA | Read / Write Special | Output enables (direction bits) for P31..P0 |
| DIRB | $1FB | Read / Write Special | Output enables (direction bits) for P63..P32 |
| OUTA | $1FC | Read / Write Special | Output states for P31..P0 |
| OUTB | $1FD | Read / Write Special | Output states for P63..P32 |
| INA | $1FE3 | Read Special | Input states for P31..P0 |
| INB | $1FF4 | Read Special | Input states for P63..P32 |

1 Locations $1F0–$1F7 are general-purpose code/data registers by default but become these named special-purpose registers if their associated functions are enabled.

2 Most reads and writes involve the underlying Reg RAM at these locations; however, those marked "special" may read from a special internal register (instead of Reg RAM) or write to both the Reg RAM location as well as to a special internal register.

3 Also debug interrupt call address.

4 Also debug interrupt return address.

Additionally, for PASM2 code that is either in-line (within a Spin2 method) or called (by a Spin2 method), the registers $1D8–$1DF are readable/writable by both languages using the symbols PR0–PR7. PASM2 code that is launched into another cog does not share this Reg RAM space with Spin2.

| **Name** | **Address** | **Access** | **Special Purpose** |
| --- | --- | --- | --- |
| PR0–PR7 | $1D8–$1DF | Read/Write | PASM2 ⇄ Spin2 communication mechanism |

## REV

Reverse

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Reverse bits of value.

REV *Dest*

**Result:** The 32-bit pattern in Dest is reversed.

* Dest is the register containing the bit value to reverse.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | 000 | DDDDDDDDD | 001101001 | D | — | — | 2 |

**Explanation:**

REV performs a bitwise reverse (bits 31:0 ⇒ bits 0:31) of the value in Dest and stores the result back into Dest. This is useful for processing binary data in a different MSB/LSB order than it is transmitted with.

## ROL

Rotate left

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Rotate bits left.

ROL *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The bits of Dest are rotated left by Src bits; any departing MSBs are moved into LSBs.

* Dest is the register containing the value to rotate left by Src bits.
* Src is a register or 5-bit literal whose value indicates the number of bit positions to rotate.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0000001 | CZI | DDDDDDDDD | SSSSSSSSS | D | Last bit out1 | Result = 0 | 2 |

1 C = last bit rotated out if Src[4:0] > 0, else C = Dest[31].

**Explanation:**

ROL rotates Dest's binary value left by Src places (0–31 bits). All MSBs rotated out are moved into the new LSBs.

If the WC or WCZ effect is specified, the C flag is updated to the value of the last bit rotated out (effectively C = result bit "0") if Src is 1–31, or to Dest[31] if Src is 0.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest result equals zero, or is cleared (0) if it is non-zero. Since no bits are lost by this operation, the result will only be zero if Dest started at zero.

## ROLBYTE

Rotate left byte

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Rotate a byte left into a value.

ROLBYTE *Dest*, {#}*Src,* #*Num*

ROLBYTE *Dest*

**Result:** Byte Num (0–3) of Src, or a byte from a source described by prior ALTGB instruction, is rotated left into Dest.

* Dest is the register in which to rotate the byte into.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value contains the target byte to read.
* Num is a 2-bit literal identifying the nibble ID (0–3) of Src to read.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001000 | NNI | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1001000 | 000 | DDDDDDDDD | 000000000 | D | — | — | 2 |

**Related:** [ALTGB](#_q234901aifm8), [ROLNIB](#_5imtv7yml80t), [ROLWORD](#_lhi32k9v1cp3), [GETNIB](#_8yjp9q7d9ncx), [GETBYTE](#_t84l3o6tozfm), [GETWORD](#_ppayepmbzjsl), [SETNIB](#_dtkktvehwfob), [SETBYTE](#_f8filcyhyg4l), and [SETWORD](#_9gwjt0ijlutf)

**Explanation:**

ROLBYTE reads the byte identified by Num (0–3) from Src, or a byte from the source described by a prior ALTGB instruction, and rotates it left into Dest.

ROLBYTE achieves the same effect as two instructions— an 8-bit SHL followed by [SETBYTE](#_f8filcyhyg4l) into byte 0.

Num (0–3) identifies a value's individual bytes, by position, in least-significant byte order.

Syntax 2 is intended for use after an ALTGB instruction; i.e. in a loop to iteratively read a series of byte values within contiguous long registers.

## ROLNIB

Rotate left nibble

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Rotate a nibble left into a value.

ROLNIB *Dest*, {#}*Src,* #*Num*

ROLNIB *Dest*

**Result:** Nibble Num (0–7) of Src, or a nibble from a source described by prior ALTGN instruction, is rotated left into Dest.

* Dest is the register in which to rotate the nibble into.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value contains the target nibble to read.
* Num is a 3-bit literal identifying the nibble ID (0–7) of Src to read.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 100010N | NNI | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1000100 | 000 | DDDDDDDDD | 000000000 | D | — | — | 2 |

**Related:** [ALTGN](#_asw5rsrcqtpn), [ROLBYTE](#_8x2v3buaskxv), [ROLWORD](#_lhi32k9v1cp3), [GETNIB](#_8yjp9q7d9ncx), [GETBYTE](#_t84l3o6tozfm), [GETWORD](#_ppayepmbzjsl), [SETNIB](#_dtkktvehwfob), [SETBYTE](#_f8filcyhyg4l), and [SETWORD](#_9gwjt0ijlutf)

**Explanation:**

ROLNIB reads the nibble identified by Num (0–7) from Src, or a nibble from the source described by a prior ALTGN instruction, and rotates it left into Dest.

ROLNIB achieves the same effect as two instructions— a 4-bit SHL followed by [SETNIB](#_dtkktvehwfob) into nibble 0.

Num (0–7) identifies a value's individual nibbles, by position, in least-significant nibble order.

Syntax 2 is intended for use after an ALTGN instruction; i.e. in a loop to iteratively read a series of nibble values within contiguous long registers.

## ROLWORD

Rotate left word

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Rotate a word left into a value.

ROLWORD *Dest*, {#}*Src,* #*Num*

ROLWORD *Dest*

**Result:** Word Num (0–1) of Src, or a word from a source described by prior ALTGW instruction, is rotated left into Dest.

* Dest is the register in which to rotate the word into.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value contains the target word to read.
* Num is a 1-bit literal identifying the nibble ID (0–1) of Src to read.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001010 | 0NI | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1001010 | 000 | DDDDDDDDD | 000000000 | D | — | — | 2 |

**Related:** [ALTGW](#_hahk50n443um), [ROLNIB](#_5imtv7yml80t), [ROLBYTE](#_8x2v3buaskxv), [GETNIB](#_8yjp9q7d9ncx), [GETBYTE](#_t84l3o6tozfm), [GETWORD](#_ppayepmbzjsl), [SETNIB](#_dtkktvehwfob), [SETBYTE](#_f8filcyhyg4l), and [SETWORD](#_9gwjt0ijlutf)

**Explanation:**

ROLWORD reads the word identified by Num (0–1) from Src, or a word from the source described by a prior ALTGW instruction, and rotates it left into Dest.

ROLWORD achieves the same effect as two instructions— a 16-bit SHL followed by [SETWORD](#_9gwjt0ijlutf) into word 0.

Num (0–1) identifies a value's individual words, by position, in least-significant word order.

Syntax 2 is intended for use after an ALTGW instruction; i.e. in a loop to iteratively read a series of word values within contiguous long registers.

## ROR

Rotate right

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Rotate bits right.

ROR *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The bits of Dest are rotated right by Src bits; any departing LSBs are moved into MSBs.

* Dest is the register containing the value to rotate right by Src bits.
* Src is a register or 5-bit literal whose value indicates the number of bit positions to rotate.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0000000 | CZI | DDDDDDDDD | SSSSSSSSS | D | Last bit out1 | Result = 0 | 2 |

1 C = last bit rotated out if Src[4:0] > 0, else C = Dest[0].

**Explanation:**

ROR rotates Dest's binary value right by Src places (0–31 bits). All LSBs rotated out are moved into the new MSBs.

If the WC or WCZ effect is specified, the C flag is updated to the value of the last bit rotated out (effectively C = result bit "31") if Src is 1–31, or to Dest[0] if Src is 0.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest result equals zero, or is cleared (0) if it is non-zero. Since no bits are lost by this operation, the result will only be zero if Dest started at zero.

## SAL

Shift arithmetic left

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Shift bits left, extending the LSB.

SAL *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The bits of Dest are shifted left by Src bits, extending Dest[0] into new rightmost bits.

* Dest is the register containing the value to arithmetically left shift by Src bits.
* Src is a register or 5-bit literal whose value indicates the number of bits to arithmetically shift left.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0000111 | CZI | DDDDDDDDD | SSSSSSSSS | D | Last bit out1 | Result = 0 | 2 |

1 C = last bit shifted out if Src[4:0] > 0, else C = Dest[31].

**Explanation:**

SAL shifts Dest's binary value left by Src places (0–31 bits) and sets the new LSBs to that of the original Dest[0]. SAL is the complement of SAR for bit streams but not for math operations— use SHL instead for swift 32-bit integer multiplication by a power-of-two.

If the WC or WCZ effect is specified, the C flag is updated to the value of the last bit shifted out (effectively C = result bit "32") if Src is 1–31, or to Dest[31] if Src is 0.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest result equals zero, or is cleared (0) if it is non-zero.

## SAR

Shift arithmetic right

[**Math Instruction**](#_xnk9pb8j5sof) **-** Divide signed 32-bit integer by power-of-two; a.k.a shift bits right, extending the MSB.

SAR *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The bits of Dest are shifted right by Src bits, extending Dest[31] (the sign bit) into new leftmost bits.

* Dest is the register containing the value to arithmetically right shift by Src bits.
* Src is a register or 5-bit literal whose value indicates the number of bits to arithmetically shift right.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0000110 | CZI | DDDDDDDDD | SSSSSSSSS | D | Last bit out1 | Result = 0 | 2 |

1 C = last bit shifted out if Src[4:0] > 0, else C = Dest[0].

**Related:** SHR, SH

**Explanation:**

SAR shifts Dest's binary value right by Src places (0–31 bits) and sets the new MSBs to that of the original Dest[31]; preserving the sign of a signed integer. This is useful for bit stream manipulation and for swift division— it is similar to SHR for swift division by a power-of-two, but is safe for both signed and unsigned integers.

If the WC or WCZ effect is specified, the C flag is updated to the value of the last bit shifted out (effectively C = result bit "-1") if Src is 1–31, or to Dest[0] if Src is 0.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest result equals zero, or is cleared (0) if it is non-zero.

## SCA

Scale

[**Math Instruction**](#_xnk9pb8j5sof) **-** Create unsigned 16-bit scale value for next instruction's S value.

SCA *Dest*, {#}*Src* {WZ}

**Result:** The upper 16 bits of the unsigned product from the 16-bit Dest and Src multiplication is substituted as the next instruction's Src value and optionally the Z flag is updated to the zero status.

* Dest is a register containing the 16-bit value to multiply with Src.
* Src is a register, 9-bit literal, or 16-bit augmented literal whose value is multiplied with Dest.
* WZ is an optional effect to update the Z flag.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1010001 | 0ZI | DDDDDDDDD | SSSSSSSSS | — | — | Product = 0 | 2 |

**Related:** SCAS

**Explanation:**

SCA multiplies the lower 16-bits of each of Dest and Src together, right shifts the 32-bit product by 16 (to scale down the result), and substitutes this value as the next instruction's Src value.

If the WZ effect is specified, the Z flag is set (1) if the product (before scaling down) is zero, or is cleared (0) if non-zero.

**Note:** The instruction following SCA is shielded from interrupt.

## SCAS

Scale, signed

[**Math Instruction**](#_xnk9pb8j5sof) **-** Create signed 18-bit scale value for next instruction's S value.

SCAS *Dest*, {#}*Src* {WZ}

**Result:** The upper 18 bits of the signed product from the 16-bit Dest and Src multiplication is substituted as the next instruction's Src value and optionally the Z flag is updated to the zero status.

* Dest is a register containing the signed 16-bit value to multiply with Src.
* Src is a register, 9-bit literal, or signed 16-bit augmented literal whose value is multiplied with Dest.
* WZ is an optional effect to update the Z flag.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1010001 | 1ZI | DDDDDDDDD | SSSSSSSSS | — | — | Product = 0 | 2 |

**Related:** SCA

**Explanation:**

SCAS multiplies the lower, signed 16-bits of each of Dest and Src together, right shifts the 32-bit product by 14 (to scale down the result), and substitutes this value as the next instruction's Src value.

If the WZ effect is specified, the Z flag is set (1) if the product (before scaling down) is zero, or is cleared (0) if non-zero.

**Note:** The instruction following SCAS is shielded from interrupt.

## SETBYTE

Set byte

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set a byte to new value.

SETBYTE *Dest*, {#}*Src,* #*Num*

SETBYTE {#}*Src*

**Result:** Src[7:0] is written to byte Num (0–3) of Dest, or to another register byte described by prior ALTSB instruction.

* Dest is the register in which to modify a byte.
* Src is a register or 8-bit literal whose bits [7:0] will be stored in the designated location.
* Num is a 2-bit literal identifying the nibble ID (0–3) of Dest to modify.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1000110 | NNI | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1000110 | 00I | 000000000 | SSSSSSSSS | D1 | — | — | 2 |

1 Dest, and the target byte ID, is specified by a prior ALTSB instruction.

**Related:** [ALTSB](#_2jkx93n2978v), [SETNIB](#_dtkktvehwfob), [SETWORD](#_9gwjt0ijlutf), [GETNIB](#_8yjp9q7d9ncx), [GETBYTE](#_t84l3o6tozfm), [GETWORD](#_ppayepmbzjsl), [ROLNIB](#_5imtv7yml80t), [ROLBYTE](#_8x2v3buaskxv), and [ROLWORD](#_lhi32k9v1cp3)

**Explanation:**

SETBYTE stores Src[7:0] into the byte identified by Num within Dest, or the byte and register described by a prior ALTSB instruction. No other bits are modified.

Num (0–3) identifies a value's individual bytes, by position, in least-significant byte order.

Syntax 2 is intended for use after an ALTSB instruction; i.e. in a loop to iteratively affect a series of byte values within contiguous long registers.

## SETD

Set destination

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Set template D field for ALTI.

SETD *Dest*, {#}*Src*

**Result:** The *D field* [17:9] of template Dest is set to Src[8:0].

* Dest is the register whose 32-bit value is a template for use with an ALTI instruction.
* Src is a register or 9-bit literal whose value (Src[8:0]) is copied to the D field of Dest.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001101 | 10I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |

**Related:** [SETS](#_94hz48griij7), [SETR](#_brza01jtowzy), and [ALTI](#_ok3etaj2mvuf)

**Explanation:**

SETD copies Src[8:0] to the *D field* of the template, Dest, to be used with an ALTI instruction. Bits outside the D field remain unaffected. The D field (or Dest field) holds the address of a register (or sometimes a literal value) for the instruction to use as it's *destination* value, and usually as its result destination, during its execution.

The ALTI template is a 32-bit value with the following format:

| **Bits** | 31:28 (4 bits) | 27:19 (9 bits) | 18 | 17:9 (9 bits) | 8:0 (9 bits) |
| --- | --- | --- | --- | --- | --- |
| **Field Description** | Condition Field | Result Field | Indirect "I" Field | Dest "D" Field | Source "S" Field |

SETD can also be used in self-modifying Reg RAM code. Unlike with ALTx instructions, when used this way, field value modification occurs in the program code itself (not the instruction pipeline); code is altered, values persist. Due to the instruction pipeline nature, after modifying a code register, it is necessary to elapse at least two instructions before executing the modified register.

SETD inst, op 'set "inst" register[17:9] to op[8:0]

NOP 'first spacer instruction, could be anything

NOP 'second spacer instruction, could be anything

inst MOV x, y 'operate on x using y; x may become any register per SETD

## SETNIB

Set nibble

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set a nibble to new value.

SETNIB *Dest*, {#}*Src,* #*Num*

SETNIB {#}*Src*

**Result:** Src[3:0] is written to nibble Num (0–7) of Dest, or to another register nibble described by prior ALTSN instruction.

* Dest is the register in which to modify a nibble.
* Src is a register or 4-bit literal whose bits [3:0] will be stored in the designated location.
* Num is a 3-bit literal identifying the nibble ID (0–7) of Dest to modify.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 100000N | NNI | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1000000 | 00I | 000000000 | SSSSSSSSS | D1 | — | — | 2 |

1 Dest, and the target nibble ID, is specified by a prior ALTSN instruction.

**Related:** [ALTSN](#_o72f3jsaebvh), [SETBYTE](#_f8filcyhyg4l), [SETWORD](#_9gwjt0ijlutf), [GETNIB](#_8yjp9q7d9ncx), [GETBYTE](#_t84l3o6tozfm), [GETWORD](#_ppayepmbzjsl), [ROLNIB](#_5imtv7yml80t), [ROLBYTE](#_8x2v3buaskxv), and [ROLWORD](#_lhi32k9v1cp3)

**Explanation:**

SETNIB stores Src[3:0] into the nibble identified by Num within Dest, or the nibble and register described by a prior ALTSN instruction. No other bits are modified.

Num (0–7) identifies a value's individual nibbles, by position, in least-significant nibble order.

Syntax 2 is intended for use after an ALTSN instruction; i.e. in a loop to iteratively affect a series of nibble values within contiguous long registers.

## SETR

Set result

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Set template Result field for ALTI.

SETR *Dest*, {#}*Src*

**Result:** The *Result field* [27:19] of template Dest is set to Src[8:0].

* Dest is the register whose 32-bit value is a template for use with an ALTI instruction.
* Src is a register or 9-bit literal whose value (Src[8:0]) is copied to the D field of Dest.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001101 | 01I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |

**Related:** [SETD](#_jw7izhflytke), [SETS](#_94hz48griij7), and [ALTI](#_ok3etaj2mvuf)

**Explanation:**

SETR copies Src[8:0] to the *Result field* of the template, Dest, to be used with an ALTI instruction. Bits outside the Result field remain unaffected. The Result field does not exist in instruction opcodes, but takes its value from the Dest field, holding the address of a register for the instruction to use as its result destination upon execution.

The ALTI template is a 32-bit value with the following format:

| **Bits** | 31:28 (4 bits) | 27:19 (9 bits) | 18 | 17:9 (9 bits) | 8:0 (9 bits) |
| --- | --- | --- | --- | --- | --- |
| **Field Description** | Condition Field | Result Field | Indirect "I" Field | Dest "D" Field | Source "S" Field |

SETR can also be used in self-modifying Reg RAM code, though it won't affect the *Register field* (which doesn't exist in instruction opcodes) but rather will affect the *Instr* *field* and the upper two bits of the *FX field*. Unlike with ALTx instructions, when used this way, field value modification occurs in the program code itself (not the instruction pipeline); code is altered, values persist. Due to the instruction pipeline nature, after modifying a code register, it is necessary to elapse at least two instructions before executing the modified register.

SETR inst, op 'set "inst" register[27:19] to op[8:0]

NOP 'first spacer instruction, could be anything

NOP 'second spacer instruction, could be anything

inst MOV x, y 'operate on x using y; MOV can become AND/OR/etc. per SETR

## SETS

Set source

[**Indirection Instruction**](#_si46t1tj3ovj) **-** Set template S field for ALTI.

SETS *Dest*, {#}*Src*

**Result:** The *S field* [8:0] of template Dest is set to Src[8:0].

* Dest is the register whose 32-bit value is a template for use with an ALTI instruction.
* Src is a register or 9-bit literal whose value (Src[8:0]) is copied to the S field of Dest.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001101 | 11I | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |

**Related:** [SETD](#_jw7izhflytke), [SETR](#_brza01jtowzy), and [ALTI](#_ok3etaj2mvuf)

**Explanation:**

SETS copies Src[8:0] to the *S field* of the template, Dest, to be used with an ALTI instruction. Bits outside the S field remain unaffected. The S field (or Src field) holds the address of a register or literal value for an instruction to use as it's *source* value during its execution.

The ALTI template is a 32-bit value with the following format:

| **Bits** | 31:28 (4 bits) | 27:19 (9 bits) | 18 | 17:9 (9 bits) | 8:0 (9 bits) |
| --- | --- | --- | --- | --- | --- |
| **Field Description** | Condition Field | Result Field | Indirect "I" Field | Dest "D" Field | Source "S" Field |

SETS can also be used in self-modifying Reg RAM code. Unlike with ALTx instructions, when used this way, field value modification occurs in the program code itself (not the instruction pipeline); code is altered, values persist. Due to the instruction pipeline nature, after modifying a code register, it is necessary to elapse at least two instructions before executing the modified register.

SETS inst, op 'set "inst" register[8:0] to op[8:0]

NOP 'first spacer instruction, could be anything

NOP 'second spacer instruction, could be anything

inst MOV x, y 'operate on x using y; y may become any register/value per SETS

## SETWORD

Set word

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Set a word to new value.

SETWORD *Dest*, {#}*Src,* #*Num*

SETWORD {#}*Src*

**Result:** Src[15:0] is written to word Num (0–1) of Dest, or to another register word described by prior ALTSW instruction.

* Dest is the register in which to modify a word.
* Src is a register, 9-bit literal, or 16-bit augmented literal whose bits [15:0] will be stored in the designated location.
* Num is a 1-bit literal identifying the word ID (0–1) of Dest to modify.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1001001 | 0NI | DDDDDDDDD | SSSSSSSSS | D | — | — | 2 |
| EEEE | 1001001 | 00I | 000000000 | SSSSSSSSS | D1 | — | — | 2 |

1 Dest, and the target word ID, is specified by a prior ALTSW instruction.

**Related:** [ALTSW](#_90jw6j8128ot), [SETNIB](#_dtkktvehwfob), [SETBYTE](#_f8filcyhyg4l), [GETNIB](#_8yjp9q7d9ncx), [GETBYTE](#_t84l3o6tozfm), [GETWORD](#_ppayepmbzjsl), [ROLNIB](#_5imtv7yml80t), [ROLBYTE](#_8x2v3buaskxv), and [ROLWORD](#_lhi32k9v1cp3)

**Explanation:**

SETWORD stores Src[15:0] into the word identified by Num within Dest, or the word and register described by a prior ALTSW instruction. No other bits are modified.

Num (0–1) identifies a value's individual words, by position, in least-significant word order.

Syntax 2 is intended for use after an ALTSW instruction; i.e. in a loop to iteratively affect a series of word values within contiguous long registers.

## SHL

Shift left

[**Bit Operation**](#_1ykf5b4dvpz7) **/** [**Math Instruction**](#_xnk9pb8j5sof) **-** Shift bits left; a.k.a. multiply 32-bit integer by power-of-two.

SHL *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The bits of Dest are shifted left by Src bits, inserting zeros (0) as new rightmost bits.

* Dest is the register containing the value to left shift by Src bits.
* Src is a register or 5-bit literal whose value indicates the number of bits to shift left.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0000011 | CZI | DDDDDDDDD | SSSSSSSSS | D | Last bit out1 | Result = 0 | 2 |

1 C = last bit shifted out if Src[4:0] > 0, else C = Dest[31].

**Explanation:**

SHL shifts Dest's binary value left by Src places (0–31 bits) and sets the new LSBs to 0. This is useful for bit-stream manipulation as well as for swift multiplication; signed or unsigned 32-bit integer multiplication by a power-of-two. Care must be taken for power-of-two multiplications since upper bits will shift through the MSB (sign bit); mangling large signed values.

If the WC or WCZ effect is specified, the C flag is updated to the value of the last bit shifted out (effectively C = result bit "32") if Src is 1–31, or to Dest[31] if Src is 0.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest result equals zero, or is cleared (0) if it is non-zero.

## SHR

Shift right

[**Bit Operation**](#_1ykf5b4dvpz7) **/** [**Math Instruction**](#_xnk9pb8j5sof) **-** Shift bits right; a.k.a. divide unsigned 32-bit integer by power-of-two.

SHR *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The bits of Dest are shifted right by Src bits, inserting zeros (0) as new leftmost bits.

* Dest is the register containing the value to right shift by Src bits.
* Src is a register or 5-bit literal whose value indicates the number of bits to shift right.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0000010 | CZI | DDDDDDDDD | SSSSSSSSS | D | Last bit out1 | Result = 0 | 2 |

1 C = last bit shifted out if Src[4:0] > 0, else C = Dest[0].

**Explanation:**

SHR shifts Dest's binary value right by Src places (0–31 bits) and sets the new MSBs to 0. This is useful for bit-stream manipulation as well as for swift division; unsigned 32-bit integer division by a power-of-two. For similar division of a signed value, use SAR instead.

If the WC or WCZ effect is specified, the C flag is updated to the value of the last bit shifted out (effectively C = result bit "-1") if Src is 1–31, or to Dest[0] if Src is 0.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest result equals zero, or is cleared (0) if it is non-zero.

## SIGNX

Sign extend

[**Math Instruction**](#_xnk9pb8j5sof) **-** Sign-extend value beyond designated bit.

SIGNX *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The Dest value is sign-extended above the bit indicated by Src and is stored in Dest. Optionally the C and Z flags are updated to the resulting MSB and zero status.

* Dest is a register containing the value to sign-extend above bit Src[4:0] and is where the result is written.
* Src is a register or 9-bit literal whose value (lower 5 bits) identifies the bit of Dest to zero-extend beyond.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0111011 | CZI | DDDDDDDDD | SSSSSSSSS | D | MSB of result | Result = 0 | 2 |

**Related:** ZEROX

**Explanation:**

SIGNX fills the bits of Dest, above the bit indicated by Src[4:0], with the value of that identified bit; i.e. sign-extending the value. This is handy when converting encoded or received signed values from a small bit width to a large bit with; i.e. 32 bits.

If the WC or WCZ effect is specified, the C flag is set to the result's MSB value.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## STALLI

Stall interrupts

[**Interrupt Instruction**](#_40bj2qk54rdh) **-** Prevent further interrupts.

STALLI

**Result:** All future interrupts are disallowed.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | 000 | 000100001 | 000100100 | — | — | — | 2 |

**Related:** [ALLOWI](#_jv88q3ni4hqo)

**Explanation:**

STALLI disables interrupt branching. STALLI is the complement of the ALLOWI instruction— both are used to protect short, vital sections of main code from timing jitter or state loss caused by asynchronous interrupt handling.

## 

## SUB

Subtract

[**Math Instruction**](#_xnk9pb8j5sof) **-** Subtract one unsigned value from another.

SUB *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Difference of unsigned Dest and unsigned Src is stored in Dest and optionally the C and Z flags are updated to the borrow and zero status.

* Dest is a register containing the value to subtract Src from, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is subtracted from Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0001100 | CZI | DDDDDDDDD | SSSSSSSSS | D | borrow of (D - S) | Result = 0 | 2 |

**Related:** [SUBX](#_4wyzstsl5slq), [SUBS](#_phb5pufpspb9), [SUBSX](#_fwvdz7pvnpz3), [ADD](#_qhneb9rqvxbb), and [SUBR](#_ss3vapy9cq1h)

**Explanation:**

SUB subtracts the unsigned Src from the unsigned Dest and stores the result into the Dest register.

If the WC or WCZ effect is specified, the C flag is set (1) if the subtraction results in a 32-bit underflow (unsigned borrow), or is cleared (0) if no borrow.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result of Dest - Src is zero, or is cleared (0) if it is non-zero.

To subtract unsigned, multi-long values, use SUB followed by SUBX as described in [Subtracting Two Multi-Long Values](#_19e0pvyp0h0x). SUB and SUBX are also used in subtracting signed, multi-long values with SUBSX ending the sequence.

## SUBR

Subtract reverse

[**Math Instruction**](#_xnk9pb8j5sof) **-** Subtract one unsigned value from another (in reverse order to SUB).

SUBR *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Difference of unsigned Src and unsigned Dest is stored in Dest and optionally the C and Z flags are updated to the borrow and zero status.

* Dest is a register containing the value to subtract from Src, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is subtracted by Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0010110 | CZI | DDDDDDDDD | SSSSSSSSS | D | borrow of (S - D) | Result = 0 | 2 |

**Related:** [SUB](#_q0s2vcp5e3i5)

**Explanation:**

SUBR subtracts the unsigned Dest from the unsigned Src and stores the result into the Dest register. This is the reverse of the subtraction order of SUB.

If the WC or WCZ effect is specified, the C flag is set (1) if the subtraction results in a 32-bit underflow (unsigned borrow), or is cleared (0) if no borrow.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result of Dest - Src is zero, or is cleared (0) if it is non-zero.

## SUBS

Subtract signed

[**Math Instruction**](#_xnk9pb8j5sof) **-** Subtract one signed value from another.

SUBS *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Difference of signed Dest and signed Src is stored in Dest and optionally the C and Z flags are updated to the sign and zero status.

* Dest is a register containing the value to subtract Src from, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value is subtracted from Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0001110 | CZI | DDDDDDDDD | SSSSSSSSS | D | sign of (D - S) | Result = 0 | 2 |

**Related:** [SUB](#_q0s2vcp5e3i5), [SUBX](#_4wyzstsl5slq), [SUBSX](#_fwvdz7pvnpz3), and [ADDS](#_566nu7hwsj0f)

**Explanation:**

SUBS subtracts the signed Src from the signed Dest and stores the result into the Dest register. If Src is a 9-bit literal, its value is interpreted as positive (0-511; it is not sign-extended) — use ##Value (or insert a prior AUGS instruction) for a 32-bit signed value; negative or positive.

If the WC or WCZ effect is specified, the C flag is set (1) if the subtraction results in a signed underflow (signed borrow), or is cleared (0) if no underflow.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result of Dest - Src is zero, or is cleared (0) if it is non-zero.

To subtract signed, multi-long values, use SUB (not SUBS) followed possibly by SUBX, and finally SUBSX as described in [Subtracting Two Multi-Long Values](#_19e0pvyp0h0x).

## SUBSX

Subtract signed, extended

[**Math Instruction**](#_xnk9pb8j5sof) **-** Subtract one signed extended value from another.

SUBSX *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Difference of signed Dest and signed Src (plus C) is stored in Dest and optionally the C and Z flags are updated to the extended sign and zero status.

* Dest is a register containing the value to subtract Src plus C from, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value plus C is subtracted from Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0001111 | CZI | DDDDDDDDD | SSSSSSSSS | D | sign of D-(S+C) | Z AND (Result = 0) | 2 |

**Related:** [SUB](#_q0s2vcp5e3i5), [SUBX](#_4wyzstsl5slq), [SUBSX](#_fwvdz7pvnpz3), and [ADDSX](#_23s34xc9b4y1)

**Explanation:**

SUBSX subtracts the signed value of Src plus C from the signed Dest and stores the result into the Dest register. The SUBSX instruction is used to perform signed multi-long (extended) subtraction, such as 64-bit subtraction.

If the WC or WCZ effect is specified, the C flag is set (1) if the result is negative (Result[31] = 1), or is cleared (0) if positive. Use WC or WCZ on preceding SUB and SUBX instructions for proper final C flag.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Z was previously set and the result of Dest - (Src + C) is zero, or it is cleared (0) if non-zero. Use WZ or WCZ on preceding SUB and SUBX instructions for proper final Z flag.

To subtract signed multi-long values, use SUB (not SUBS) followed possibly by SUBX, and finally SUBSX as described in [Subtracting Two Multi-Long Values](#_19e0pvyp0h0x).

## SUBX

Subtract extended

[**Math Instruction**](#_xnk9pb8j5sof) **-** Subtract one unsigned extended value from another.

SUBX *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Difference of unsigned Dest and unsigned Src (plus C) is stored in Dest and optionally the C and Z flags are updated to the extended borrow and zero status.

* Dest is a register containing the value to subtract Src plus C from, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value plus C is subtracted from Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0001101 | CZI | DDDDDDDDD | SSSSSSSSS | D | borrow of D-(S+C) | Z AND (Result = 0) | 2 |

**Related:** [SUB](#_q0s2vcp5e3i5), [SUBSX](#_fwvdz7pvnpz3), and [ADDX](#_vhdbx9ift4rv)

**Explanation:**

SUBX subtracts the unsigned value of Src plus C from the unsigned Dest and stores the result into the Dest register. The SUBX instruction is used to perform unsigned multi-long (extended) subtraction, such as 64-bit subtraction.

If the WC or WCZ effect is specified, the C flag is set (1) if the subtraction results in an unsigned borrow, or is cleared (0) if no borrow. Use WC or WCZ on preceding SUB and SUBX instructions for proper final C flag. If C is set after the last SUBX in a multi-long subtraction, it indicates unsigned underflow.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Z was previously set and the result of Dest - (Src + C) is zero, or it is cleared (0) if non-zero. Use WZ or WCZ on preceding SUB and SUBX instructions for proper final Z flag.

To subtract unsigned multi-long values, use SUB followed by one or more SUBX instructions as described in [Subtracting Two Multi-Long Values](#_19e0pvyp0h0x).

## SUMC / SUMNC

Sum C or Sum not C

[**Math Instruction**](#_xnk9pb8j5sof) **-** Adjust signed value by other C-negated or !C-negated value.

SUMC *Dest*, {#}*Src* {WC|WZ|WCZ}

SUMNC *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The sum of signed Dest and either Src or -Src (according to C) is stored in Dest and optionally the C and Z flags are updated to the sign and zero status.

* Dest is a register containing the signed value to adjust by Src or -Src, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value (if C=0 or !C=0) or negated value (if C=1 or !C=1) is added into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0011100 | CZI | DDDDDDDDD | SSSSSSSSS | D | sign of (D +/- S) | Result = 0 | 2 |
| EEEE | 0011101 | CZI | DDDDDDDDD | SSSSSSSSS | D | sign of (D +/- S) | Result = 0 | 2 |

**Related:** [SUMZ](#_5u8few7uaee4) and [SUMNZ](#_5u8few7uaee4)

**Explanation:**

SUMC or SUMNC adjusts the signed Dest value by Src or -Src (depending on C or !C) and stores the result into the Dest register. Prior to adding to Dest, the Src value is negated if C (SUMC) or !C (SUMNC) is 1.

If the WC or WCZ effect is specified, the C flag is set (1) if the result is negative, or is cleared (0) if positive.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result of Dest +/- Src is zero, or is cleared (0) if it is non-zero.

## SUMZ / SUMNZ

Sum Z or Sum not Z

[**Math Instruction**](#_xnk9pb8j5sof) **-** Adjust signed value by other Z-negated or !Z-negated value.

SUMZ *Dest*, {#}*Src* {WC|WZ|WCZ}

SUMNZ *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The sum of signed Dest and either Src or -Src (according to Z) is stored in Dest and optionally the C and Z flags are updated to the sign and zero status.

* Dest is a register containing the signed value to adjust by Src or -Src, and is where the result is written.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value (if Z=0 or !Z=0) or negated value (if Z=1 or !Z=1) is added into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0011110 | CZI | DDDDDDDDD | SSSSSSSSS | D | sign of (D +/- S) | Result = 0 | 2 |
| EEEE | 0011111 | CZI | DDDDDDDDD | SSSSSSSSS | D | sign of (D +/- S) | Result = 0 | 2 |

**Related:** [SUMC](#_8370tvf23i61) and [SUMNC](#_8370tvf23i61)

**Explanation:**

SUMZ or SUMNZ adjusts the signed Dest value by Src or -Src (depending on Z or !Z) and stores the result into the Dest register. Prior to adding to Dest, the Src value is negated if Z (SUMZ) or !Z (SUMNZ) is 1.

If the WC or WCZ effect is specified, the C flag is set (1) if the result is negative, or is cleared (0) if positive.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result of Dest +/- Src is zero, or is cleared (0) if it is non-zero.

## TEST

Test

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Test D, or bitwise AND D with S, to affect flags.

TEST *Dest* {WC|WZ|WCZ}

TEST *Dest,* {#}*Src* {WC|WZ|WCZ}

**Result:** The parity and zero-state of Dest, or of Dest bitwise ANDed with Src, is stored in the C and Z flags.

* Dest is the register whose value will be tested.
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose value is ANDed with Dest.
* WC, WZ, and WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0111110 | CZ0 | DDDDDDDDD | DDDDDDDDD | — | Parity of D | D = 0 | 2 |
| EEEE | 0111110 | CZI | DDDDDDDDD | SSSSSSSSS | — | Parity of (D & S) | (D & S) = 0 | 2 |

**Related:** [TESTN](#_q7r5myfbnfk5)

**Explanation:**

TEST determines the parity (number of high (1) bits) and the zero or non-zero state of Dest, or of Dest bitwise ANDed with Src, and stores the results in the C and/or Z flag.

If the WC or WCZ effect is specified, the C flag is set (1) if the number of high (1) bits in Dest (or Dest ANDed with Src) is odd, or is cleared (0) if it is even.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Dest (or Dest ANDed with Src) is zero, or is cleared (0) if it is not zero.

## TESTB / TESTBN

Test bit or bit not

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Test bit of D or !D and either store, AND, OR, or XOR the result into flags.

TESTB *Dest,* {#}*Src* WC|WZ

TESTB *Dest,* {#}*Src* ANDC|ANDZ

TESTB *Dest,* {#}*Src* ORC|ORZ

TESTB *Dest,* {#}*Src* XORC|XORZ

TESTBN *Dest,* {#}*Src* WC|WZ

TESTBN *Dest,* {#}*Src* ANDC|ANDZ

TESTBN *Dest,* {#}*Src* ORC|ORZ

TESTBN *Dest,* {#}*Src* XORC|XORZ

**Result:** The state of Dest's bit Src is read, possibly inverted, and either stored as-is, or bitwise ANDed, ORed, or XORed into C or Z.

* Dest is the register whose value will have a single bit tested.
* Src is a register or 5-bit literal whose value identifies the bit (0–31) of Dest to test.
* WC, WZ, ANDC, ANDZ, ORC, ORZ, XORC, and XORZ is a required effect to update or bitwise manipulate the C or Z flag.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0100000 | CZI | DDDDDDDDD | SSSSSSSSS | — | IN[D[4:0]] | IN[D[4:0]] | 2 |
| EEEE | 0100010 | CZI | DDDDDDDDD | SSSSSSSSS | — | C AND IN[D[4:0]] | Z AND IN[D[4:0]] | 2 |
| EEEE | 0100100 | CZI | DDDDDDDDD | SSSSSSSSS | — | C OR IN[D[4:0]] | Z OR IN[D[4:0]] | 2 |
| EEEE | 0100110 | CZI | DDDDDDDDD | SSSSSSSSS | — | C XOR IN[D[4:0]] | Z XOR IN[D[4:0]] | 2 |
| EEEE | 0100001 | CZI | DDDDDDDDD | SSSSSSSSS | — | !IN[D[4:0]] | !IN[D[4:0]] | 2 |
| EEEE | 0100011 | CZI | DDDDDDDDD | SSSSSSSSS | — | C AND !IN[D[4:0]] | Z AND !IN[D[4:0]] | 2 |
| EEEE | 0100101 | CZI | DDDDDDDDD | SSSSSSSSS | — | C OR !IN[D[4:0]] | Z OR !IN[D[4:0]] | 2 |
| EEEE | 0100111 | CZI | DDDDDDDDD | SSSSSSSSS | — | C XOR !IN[D[4:0]] | Z XOR !IN[D[4:0]] | 2 |

**Related:** [TESTP](#_d2afnf6ec911) and [TESTPN](#_d2afnf6ec911)

**Explanation:**

TESTB or TESTBN reads the state (0/1) of a bit in Dest designated by Src, possibly inverts that result, and either stores it as-is, or bitwise ANDs, ORs, or XORs it into the C or Z flag.

Src[4:0] indicates the bit number (0–31) to test.

If the WC or WZ effect is specified, the C or Z flag is overwritten with the state or inverse state of the bit.

If the ANDC or ANDZ effect is specified, the C or Z flag is bitwise ANDed with the state or inverse state of the bit.

If the ORC or ORZ effect is specified, the C or Z flag is bitwise ORed with the state or inverse state of the bit.

If the XORC or XORZ effect is specified, the C or Z flag is bitwise XORed with the state or inverse state of the bit.

## TESTN

Test not

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Test D by bitwise ANDing with !S to affect flags only.

TESTN *Dest,* {#}*Src* {WC|WZ|WCZ}

**Result:** The parity and zero-state of Dest bitwise ANDed with !Src is stored in the C and Z flags.

* Dest is the register whose value will be tested.
* Src is an optional register, 9-bit literal, or 32-bit augmented literal whose inverse value is ANDed with Dest.
* WC, WZ, and WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0111111 | CZI | DDDDDDDDD | SSSSSSSSS | — | Parity of (D & !S) | (D & !S) = 0 | 2 |

**Related:** [TEST](#_6m4jc27v9rub)

**Explanation:**

TESTN determines the parity (number of high (1) bits) and the zero or non-zero state of Dest bitwise ANDed with !Src and stores the results in the C and/or Z flag.

If the WC or WCZ effect is specified, the C flag is set (1) if the number of high (1) bits in Dest ANDed with !Src is odd, or is cleared (0) if it is even.

If the WZ or WCZ effect is specified, the Z flag is set (1) if Dest ANDed with !Src is zero, or is cleared (0) if it is not zero.

## TESTP / TESTPN

Test pin or pin not

[**I/O Pin Instruction**](#_21gxb0rq4onw) **-** Test pin and either store, AND, OR, or XOR the result or inverse result into C/Z.

TESTP {#}*Dest* WC|WZ

TESTP {#}*Dest* ANDC|ANDZ

TESTP {#}*Dest* ORC|ORZ

TESTP {#}*Dest* XORC|XORZ

TESTPN {#}*Dest* WC|WZ

TESTPN {#}*Dest* ANDC|ANDZ

TESTPN {#}*Dest* ORC|ORZ

TESTPN {#}*Dest* XORC|XORZ

**Result:** The state of the I/O pin described by Dest is read, possibly inverted, and either stored as-is, or bitwise ANDed, ORed, or XORed into C or Z.

* Dest is the register or 6-bit literal whose value identifies the I/O pin to test.
* WC, WZ, ANDC, ANDZ, ORC, ORZ, XORC, and XORZ is a required effect to update or bitwise manipulate the C or Z flag.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000000 | — | IN[D[5:0]] | IN[D[5:0]] | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000010 | — | C AND IN[D[5:0]] | Z AND IN[D[5:0]] | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000100 | — | C OR IN[D[5:0]] | Z OR IN[D[5:0]] | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000110 | — | C XOR IN[D[5:0]] | Z XOR IN[D[5:0]] | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000001 | — | !IN[D[5:0]] | !IN[D[5:0]] | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000011 | — | C AND !IN[D[5:0]] | Z AND !IN[D[5:0]] | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000101 | — | C OR !IN[D[5:0]] | Z OR !IN[D[5:0]] | 2 |
| EEEE | 1101011 | CZL | DDDDDDDDD | 001000111 | — | C XOR !IN[D[5:0]] | Z XOR !IN[D[5:0]] | 2 |

**Related:** [TESTB](#_sjy72z2klc7g) and [TESTBN](#_sjy72z2klc7g)

**Explanation:**

TESTP or TESTPN reads the state (0/1) of the I/O pin designated by Dest, possibly inverts that result, and either stores it as-is, or bitwise ANDs, ORs, or XORs it into the C or Z flag. Dest[5:0] indicates the pin number (0–63) to test.

If the WC or WZ effect is specified, the C or Z flag is overwritten with the state or inverse state of the pin.

If the ANDC or ANDZ effect is specified, the C or Z flag is bitwise ANDed with the state or inverse state of the pin.

If the ORC or ORZ effect is specified, the C or Z flag is bitwise ORed with the state or inverse state of the pin.

If the XORC or XORZ effect is specified, the C or Z flag is bitwise XORed with the state or inverse state of the pin.

## TJF / TJNF

Test, jump if full or not full

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Test value and jump if full (-1; $FFFF\_FFFF) or not full (<> -1; <> $FFFF\_FFFF).

TJF *Dest*, {#}*Src*

TJNF *Dest*, {#}*Src*

**Result:** Dest is tested and if it's full (or not full in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Dest is a register whose value is tested for full or not full.
* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011101 | 00I | DDDDDDDDD | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011101 | 01I | DDDDDDDDD | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when Dest is full (or not full in syntax 2).

**Explanation:**

TJF or TJNF tests the value in Dest and jumps to the address described by Src if the result is full (-1; $FFFF\_FFFF; in syntax 1) or not full (<> -1; <> $FFFF\_FFFF; in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the TJF / TJNF. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## TJS / TJNS

Test, jump if signed or not signed

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Test value and jump if signed or not signed.

TJS *Dest*, {#}*Src*

TJNS *Dest*, {#}*Src*

**Result:** Dest is tested and if it's signed (or not signed in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Dest is a register whose value is tested for sign or no sign.
* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011101 | 10I | DDDDDDDDD | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011101 | 11I | DDDDDDDDD | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when Dest is signed (or not signed in syntax 2).

**Explanation:**

TJS or TJNS tests the value in Dest and jumps to the address described by Src if the result is signed (Dest[31] = 1) or not signed (Dest[31] = 0).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the TJS / TJNS. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## TJV

Test, jump if overflow

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Test value and jump if overflowed.

TJV *Dest*, {#}*Src*

**Result:** Dest and C are tested and if there's been an overflow, PC is set to a new relative (#Src) or absolute (Src) address.

* Dest is a register whose value is tested for overflow (Dest[31] != C).
* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011110 | 00I | DDDDDDDDD | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when Dest has overflowed.

**Explanation:**

TJV tests the value in Dest against C and jumps to the address described by Src if Dest has overflowed (Dest[31] != C). This instruction requires that C be updated (to the correct sign) by the previous ADDS / ADDSX / SUBS / SUBSX / CMPS / CMPSX / SUMx instruction.

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the TJV. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## TJZ / TJNZ

Test, jump if zero or not zero

[**Flow Control Instruction**](#_v82tlx47lbbn) **-** Test value and jump if zero or not zero.

TJZ *Dest*, {#}*Src*

TJNZ *Dest*, {#}*Src*

**Result:** Dest is tested and if it's zero (or not zero in syntax 2), PC is set to a new relative (#Src) or absolute (Src) address.

* Dest is a register whose value is tested for zero or not zero.
* Src is a register, 9-bit literal, or 20-bit augmented literal whose value is the absolute or relative address to set PC to. Use # for relative addressing; omit # for absolute addressing.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1011100 | 10I | DDDDDDDDD | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |
| EEEE | 1011100 | 11I | DDDDDDDDD | SSSSSSSSS | PC1 | — | — | 2 or 4 / 2 or 13–20 |

1 PC is written only when Dest is zero (or not zero in syntax 2).

**Explanation:**

TJZ or TJNZ tests the value in Dest and jumps to the address described by Src if the result is zero (syntax 1) or not zero (in syntax 2).

The address (Src) can be absolute or relative. To specify an absolute address, Src must be a register containing a 20-bit address value. To specify a relative address, use #Label for a 9-bit signed offset (a range of -256 to +255 instructions) or use ##Label (or insert a prior AUGS instruction) for a 20-bit signed offset (a range of -524288 to +524287). Offsets are relative to the instruction following the TJZ / TJNZ. The signed offset value is in units of whole instructions— it is added to PC as-is when in Cog/LUT execution mode and is multiplied by 4 then added to PC when in Hub execution mode (long-aligned Hub code not required).

## WAITATN

Wait attention

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Wait for and clear attention flag.

WAITATN {WC|WZ|WCZ}

**Result:** Wait for attention event, then clear the flag; optionally aborting on timeout and setting C and/or Z on abort.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000011110 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |

**Related:** [COGATN](#_wx29grvrexs), [POLLATN](#_ref43okqtzzr), [JATN](#_yfon3snogp62), and [JNATN](#_yfon3snogp62)

**Explanation:**

WAITATN waits for an attention event to occur (unless the event flag is already set), then clears the event flag (unless it's being set again by the event sensor) and resumes execution at the next instruction. Optionally, WAITATN can time-out if the attention event doesn't occur soon enough; setting C and/or Z flags and then resuming execution at the next instruction.

To set the optional timeout, insert a SETQ (with a future System Counter target value) right before WAITATN.

The WC, WZ, or WCZ effect is recommended only if the optional timeout is specified, in which case the C flag and/or Z flag is set (1) if a timeout occurred before the event, or is cleared (0) if the event occurred before the timeout.

During a *wait*, the pipeline is stalled; no instructions execute and no interrupts are processed in the cog until the wait condition ends.

The attention event flag is set whenever another cog issues an attention request for this cog. The attention event flag is cleared upon cog start, or execution of POLLATN, WAITATN, JATN, or JNATN instructions.

## WAITCT1/2/3

Wait counter 1/2/3

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Wait for and clear counter 1, 2, or 3 event flag.

WAITCT1 {WC|WZ|WCZ}

WAITCT2 {WC|WZ|WCZ}

WAITCT3 {WC|WZ|WCZ}

**Result:** Wait for counter 1, 2, or 3 event, then clear the flag; optionally aborting on timeout and setting C and/or Z on abort.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000010001 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |
| EEEE | 1101011 | CZ0 | 000010010 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |
| EEEE | 1101011 | CZ0 | 000010011 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |

**Related:** [ADDCTx](#_u2xzjautl27l), [POLLCTx](#_dgitqmcw0hm), [JCTx](#_6f897k4gldkf), and [JNCTx](#_6f897k4gldkf)

**Explanation:**

WAITCT1, WAITCT2, or WAITCT3 waits for a counter 1, 2, or 3 event to occur (unless the event flag is already set), then clears the event flag (unless it's being set again by the event sensor) and resumes execution at the next instruction. Optionally, WAITCTx can time-out if the counter event doesn't occur soon enough; setting C and/or Z flags and then resuming execution at the next instruction.

To set the optional timeout, insert a SETQ (with a future System Counter target value) right before WAITCTx.

The WC, WZ, or WCZ effect is recommended only if the optional timeout is specified, in which case the C flag and/or Z flag is set (1) if a timeout occurred before the event, or is cleared (0) if the event occurred before the timeout.

During a *wait*, the pipeline is stalled; no instructions execute and no interrupts are processed in the cog until the wait condition ends.

The counter 1, 2, or 3 event flag is set whenever the System Counter (CT) passes the value in the CT1, CT2, or CT3 event trigger register, respectively; i.e. MSB of (CT - CTx is 0). The counter event flags are cleared upon execution of the corresponding ADDCTx, POLLCTx, WAITCTx, JCTx, or JNCTx instructions.

## WAITFBW

Wait FIFO block wrap

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Wait for and clear FIFO-interface-block-wrap event flag.

WAITFBW {WC|WZ|WCZ}

**Result:** Wait for FIFO-interface-block-wrap event, then clear the flag; optionally aborting on timeout and setting C and/or Z on abort.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000011001 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |

**Related:** RDFAST, WRFAST, FBLOCK, [POLLFBW](#_86ta1z2g86ao), [JFBW](#_miks298accr1), and [JNFBW](#_miks298accr1)

**Explanation:**

WAITFBW waits for a FIFO-interface-block-wrap event to occur (unless the event flag is already set), then clears the event flag (unless it's being set again by the event sensor) and resumes execution at the next instruction. Optionally, WAITFBW can time-out if the FIFO-interface-block-wrap event doesn't occur soon enough; setting C and/or Z flags and then resuming execution at the next instruction.

To set the optional timeout, insert a SETQ (with a future System Counter target value) right before WAITFBW.

The WC, WZ, or WCZ effect is recommended only if the optional timeout is specified, in which case the C flag and/or Z flag is set (1) if a timeout occurred before the event, or is cleared (0) if the event occurred before the timeout.

During a *wait*, the pipeline is stalled; no instructions execute and no interrupts are processed in the cog until the wait condition ends.

The FIFO-interface-block-wrap event flag is set whenever the Hub RAM FIFO interface exhausts its block count and reloads its block count and start address. The FIFO-interface-block-wrap event flag is cleared upon execution of RDFAST, WRFAST, FBLOCK, POLLFBW, WAITFBW, JFBW, or JNFBW instructions.

## WAITINT

Wait interrupt

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Wait for and clear interrupt-occurred event flag.

WAITINT {WC|WZ|WCZ}

**Result:** Wait for interrupt-occurred event, then clear the flag; optionally aborting on timeout and setting C and/or Z on abort.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000010000 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |

**Related:** [POLLINT](#_m2gclmhbm44b), [JINT](#_wzi7hl4t1ffr), and [JNINT](#_wzi7hl4t1ffr)

**Explanation:**

WAITINT waits for an interrupt-occurred event to occur (unless the event flag is already set), then clears the event flag (unless it's being set again by the event sensor) and resumes execution at the next instruction. Optionally, WAITINT can time-out if the interrupt-occurred event doesn't occur soon enough; setting C and/or Z flags and then resuming execution at the next instruction.

To set the optional timeout, insert a SETQ (with a future System Counter target value) right before WAITINT.

The WC, WZ, or WCZ effect is recommended only if the optional timeout is specified, in which case the C flag and/or Z flag is set (1) if a timeout occurred before the event, or is cleared (0) if the event occurred before the timeout.

During a *wait*, the pipeline is stalled; no instructions execute and no interrupts are processed in the cog until the wait condition ends.

The interrupt-occurred event flag is set whenever interrupt 1, 2, or 3 occurs— debug interrupts are ignored. The interrupt-occurred event flag is cleared upon cog start, or execution of POLLINT, WAITINT, JINT, or JNINT instructions.

## WAITPAT

Wait pattern

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Wait for and clear pin-pattern-detected event flag.

WAITPAT {WC|WZ|WCZ}

**Result:** Wait for pin-pattern-detected event, then clear the flag; optionally aborting on timeout and setting C and/or Z on abort.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000011000 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |

**Related:** SETPAT, [POLLPAT](#_2sqdtda5rj0e), [JPAT](#_pnmnqy72ibwb), and [JNPAT](#_pnmnqy72ibwb)

**Explanation:**

WAITPAT waits for a pin-pattern-detected event to occur (unless the event flag is already set), then clears the event flag (unless it's being set again by the event sensor) and resumes execution at the next instruction. Optionally, WAITPAT can time-out if the pin-pattern-detected event doesn't occur soon enough; setting C and/or Z flags and then resuming execution at the next instruction.

To set the optional timeout, insert a SETQ (with a future System Counter target value) right before WAITPAT.

The WC, WZ, or WCZ effect is recommended only if the optional timeout is specified, in which case the C flag and/or Z flag is set (1) if a timeout occurred before the event, or is cleared (0) if the event occurred before the timeout.

During a *wait*, the pipeline is stalled; no instructions execute and no interrupts are processed in the cog until the wait condition ends.

The pin-pattern-detected event flag is set whenever the masked input pins match or don't match the pattern described by a previous SETPAT instruction. The pin-pattern-detected event flag is cleared upon execution of SETPAT, POLLPAT, WAITPAT, JPAT, or JNPAT instructions.

## WAITSE1/2/3/4

Wait selectable event 1/2/3/4

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Wait for and clear selectable event 1, 2, 3, or 4 flag.

WAITSE1 {WC|WZ|WCZ}

WAITSE2 {WC|WZ|WCZ}

WAITSE3 {WC|WZ|WCZ}

WAITSE4 {WC|WZ|WCZ}

**Result:** Wait for selectable event, then clear the flag; optionally aborting on timeout and setting C and/or Z on abort.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000010100 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |
| EEEE | 1101011 | CZ0 | 000010101 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |
| EEEE | 1101011 | CZ0 | 000010110 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |
| EEEE | 1101011 | CZ0 | 000010111 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |

**Related:** SETSEx, [POLLSEx](#_n3mua6rjfhxs), [JSEx](#_nv0zohzajupf), and [JNSEx](#_nv0zohzajupf)

**Explanation:**

WAITSE1, WAITSE2, WAITSE3, or WAITSE4 waits for a selectable event to occur (unless the event flag is already set), then clears the event flag (unless it's being set again by the event sensor) and resumes execution at the next instruction. Optionally, WAITSEx can time-out if the selectable event doesn't occur soon enough; setting C and/or Z flags and then resuming execution at the next instruction.

To set the optional timeout, insert a SETQ (with a future System Counter target value) right before WAITSEx.

The WC, WZ, or WCZ effect is recommended only if the optional timeout is specified, in which case the C flag and/or Z flag is set (1) if a timeout occurred before the event, or is cleared (0) if the event occurred before the timeout.

During a *wait*, the pipeline is stalled; no instructions execute and no interrupts are processed in the cog until the wait condition ends.

The selectable event 1, 2, 3, or 4 flag is set whenever the corresponding configured event occurs. The selectable event flag is cleared upon execution of the corresponding SETSEx, POLLSEx, WAITSEx, JSEx, or JNSEx instructions.

## WAITXFI

Wait streamer finished

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Wait for and clear streamer-finished event flag.

WAITXFI {WC|WZ|WCZ}

**Result:** Wait for streamer-finished event, then clear the flag; optionally aborting on timeout and setting C and/or Z on abort.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000011011 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |

**Related:** XINIT, XZERO, XCONT, [POLLXFI](#_r7m2wcrdq00o), [JXFI](#_76euyh5v8o1z), and [JNXFI](#_76euyh5v8o1z)

**Explanation:**

WAITXFI waits for a streamer-finished event to occur (unless the event flag is already set), then clears the event flag (unless it's being set again by the event sensor) and resumes execution at the next instruction. Optionally, WAITXFI can time-out if the streamer-finished doesn't occur soon enough; setting C and/or Z flags and then resuming execution at the next instruction.

To set the optional timeout, insert a SETQ (with a future System Counter target value) right before WAITXFI.

The WC, WZ, or WCZ effect is recommended only if the optional timeout is specified, in which case the C flag and/or Z flag is set (1) if a timeout occurred before the event, or is cleared (0) if the event occurred before the timeout.

During a *wait*, the pipeline is stalled; no instructions execute and no interrupts are processed in the cog until the wait condition ends.

The streamer-finished event flag is set whenever the streamer runs out of commands to process. The streamer-finished event flag is cleared upon execution of XINIT, XZERO, XCONT, POLLXFI, WAITXFI, JXFI, or JNXFI instructions.

## WAITXMT

Wait streamer empty

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Wait for and clear streamer-empty event flag.

WAITXMT {WC|WZ|WCZ}

**Result:** Wait for streamer-empty event, then clear the flag; optionally aborting on timeout and setting C and/or Z on abort.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000011010 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |

**Related:** XINIT, XZERO, XCONT, [POLLXMT](#_mg19wcqd1pjt), [JXMT](#_izeyc7za5235), and [JNXMT](#_izeyc7za5235)

**Explanation:**

WAITXMT waits for a streamer-empty event to occur (unless the event flag is already set), then clears the event flag (unless it's being set again by the event sensor) and resumes execution at the next instruction. Optionally, WAITXMT can time-out if the streamer-empty event doesn't occur soon enough; setting C and/or Z flags and then resuming execution at the next instruction.

To set the optional timeout, insert a SETQ (with a future System Counter target value) right before WAITXMT.

The WC, WZ, or WCZ effect is recommended only if the optional timeout is specified, in which case the C flag and/or Z flag is set (1) if a timeout occurred before the event, or is cleared (0) if the event occurred before the timeout.

During a *wait*, the pipeline is stalled; no instructions execute and no interrupts are processed in the cog until the wait condition ends.

The streamer-empty event flag is set whenever the streamer is ready for a new command. The streamer-empty event flag is cleared upon execution of XINIT, XZERO, XCONT, POLLXMT, WAITXMT, JXMT, or JNXMT instructions.

## WAITXRL

Wait streamer rollover LUT

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Wait for and clear streamer-LUT-RAM-rollover event flag

WAITXRL {WC|WZ|WCZ}

**Result:** Wait for streamer-LUT-RAM-rollover event, then clear the flag; optionally aborting on timeout and setting C and/or Z on abort.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000011101 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |

**Related:** XINIT, XZERO, XCONT, [POLLXRL](#_63dai97cdrvc), [JXRL](#_43ntogxc32ez), and [JNXRL](#_43ntogxc32ez)

**Explanation:**

WAITXRL waits for a streamer-LUT-RAM-rollover event to occur (unless the event flag is already set), then clears the event flag (unless it's being set again by the event sensor) and resumes execution at the next instruction. Optionally, WAITXRL can time-out if the streamer-LUT-RAM-rollover event doesn't occur soon enough; setting C and/or Z flags and then resuming execution at the next instruction.

To set the optional timeout, insert a SETQ (with a future System Counter target value) right before WAITXRL.

The WC, WZ, or WCZ effect is recommended only if the optional timeout is specified, in which case the C flag and/or Z flag is set (1) if a timeout occurred before the event, or is cleared (0) if the event occurred before the timeout.

During a *wait*, the pipeline is stalled; no instructions execute and no interrupts are processed in the cog until the wait condition ends.

The streamer-LUT-RAM-rollover event flag is set whenever location $1FF of the Lookup RAM is read by the streamer. The streamer-LUT-RAM-rollover event flag is cleared upon cog start or upon execution of POLLXRL, WAITXRL, JXRL, or JNXRL instructions.

## WAITXRO

Wait streamer rollover NCO

[**Event Monitor Instruction**](#p4f24b4faur6) **-** Wait for and clear streamer-NCO-rollover event flag

WAITXRO {WC|WZ|WCZ}

**Result:** Wait for streamer-NCO-rollover event, then clear the flag; optionally aborting on timeout and setting C and/or Z on abort.

* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | CZ0 | 000011100 | 000100100 | — | Timeout Abort | Timeout Abort | 2+ |

**Related:** XINIT, XZERO, XCONT, [POLLXRO](#_n123uusqspjd), [JXRO](#_gut56vubumxk), and [JNXRO](#_gut56vubumxk).

**Explanation:**

WAITXRO waits for a streamer-NCO-rollover event to occur (unless the event flag is already set), then clears the event flag (unless it's being set again by the event sensor) and resumes execution at the next instruction. Optionally, WAITXRO can time-out if the streamer-NCO-rollover event doesn't occur soon enough; setting C and/or Z flags and then resuming execution at the next instruction.

To set the optional timeout, insert a SETQ (with a future System Counter target value) right before WAITXRO.

The WC, WZ, or WCZ effect is recommended only if the optional timeout is specified, in which case the C flag and/or Z flag is set (1) if a timeout occurred before the event, or is cleared (0) if the event occurred before the timeout.

During a *wait*, the pipeline is stalled; no instructions execute and no interrupts are processed in the cog until the wait condition ends.

The streamer-NCO-rollover event flag is set whenever the streamer's numerically-controlled oscillator (NCO) rolls over. The streamer-NCO-rollover event flag is cleared upon execution of XINIT, XZERO, XCONT, POLLXRO, WAITXRO, JXRO, or JNXRO instructions.

## WRC / WRNC

Write C or not C

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Write C or not C to register.

WRC *Dest*

WRNC *Dest*

**Result:** Dest value is replaced with state of C or !C.

* Dest is the register whose value will be replaced with the state of C or !C.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | 000 | DDDDDDDDD | 001101100 | D | — | — | 2 |
| EEEE | 1101011 | 000 | DDDDDDDDD | 001101101 | D | — | — | 2 |

**Explanation:**

WRC or WRNC writes the state or inverse state of C (0 or 1) to Dest.

## WRZ / WRNZ

Write Z or not Z

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Write Z or not Z to register.

WRZ *Dest*

WRNZ *Dest*

**Result:** Dest value is replaced with state of Z or !Z.

* Dest is the register whose value will be replaced with the state of Z or !Z.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 1101011 | 000 | DDDDDDDDD | 001101110 | D | — | — | 2 |
| EEEE | 1101011 | 000 | DDDDDDDDD | 001101111 | D | — | — | 2 |

**Explanation:**

WRZ or WRNZ writes the state or inverse state of Z (0 or 1) to Dest.

## XOR

Exclusive or

[**Bit Operation Instruction**](#_1ykf5b4dvpz7) **-** Bitwise XOR a value with another.

XOR *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** Dest XOR Src is stored in Dest and flags are optionally updated with *parity* and *zero* status.

* Dest is the register containing the value to bitwise XOR with Src and is the destination in which to write the result.
* Src is a register, 9-bit literal, or 32-bit augmented literal whose value will be bitwise XORed into Dest.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0101011 | CZI | DDDDDDDDD | SSSSSSSSS | D | Parity of Result | Result = 0 | 2 |

**Explanation:**

XOR performs a bitwise XOR of the value in Src into that of Dest.

If the WC or WCZ effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits, or is cleared (0) if it contains an even number of high bits.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the Dest XOR Src result equals zero, or is cleared (0) if it is non-zero.

## ZEROX

Zero extend

[**Math Instruction**](#_xnk9pb8j5sof) **-** Zero-extend value beyond designated bit.

ZEROX *Dest*, {#}*Src* {WC|WZ|WCZ}

**Result:** The Dest value is zero-extended above the bit indicated by Src and is stored in Dest. Optionally the C and Z flags are updated to the resulting MSB and zero status.

* Dest is a register containing the value to zero-extend above bit Src[4:0] and is where the result is written.
* Src is a register or 9-bit literal whose value (lower 5 bits) identifies the bit of Dest to zero-extend beyond.
* WC, WZ, or WCZ are optional effects to update flags.

| **COND** | **INSTR** | **FX** | **DEST** | **SRC** | **Write** | **C Flag** | **Z Flag** | **Clocks** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EEEE | 0111010 | CZI | DDDDDDDDD | SSSSSSSSS | D | MSB of result | Result = 0 | 2 |

**Related:** SIGNX

**Explanation:**

ZEROX fills the bits of Dest, above the bit indicated by Src[4:0], with zeros; i.e. zero-extending the value. This is handy when converting encoded or received unsigned values from a small bit width to a large bit with; i.e. 32 bits.

If the WC or WCZ effect is specified, the C flag is set to the result's MSB value.

If the WZ or WCZ effect is specified, the Z flag is set (1) if the result is zero, or is cleared (0) if it is non-zero.

## 

# PROPELLER 2 ASSEMBLY LANGUAGE (PASM2) IN BRIEF

| Math and Logic Instructions | | |
| --- | --- | --- |
| **Instruction** | **Description** | **Clocks  Reg, LUT, & Hub** |
| ABS D {WC/WZ/WCZ} | Get absolute value of D into D. D = ABS(D). C = D[31]. \* | 2 |
| ABS D,{#}S {WC/WZ/WCZ} | Get absolute value of S into D. D = ABS(S). C = S[31]. \* | 2 |
| ADD D,{#}S {WC/WZ/WCZ} | Add S into D. D = D + S. C = carry of (D + S). \* | 2 |
| ADDS D,{#}S {WC/WZ/WCZ} | Add S into D, signed. D = D + S. C = correct sign of (D + S). \* | 2 |
| 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). | 2 |
| 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). | 2 |
| AND D,{#}S {WC/WZ/WCZ} | AND S into D. D = D & S. C = parity of result. \* | 2 |
| ANDN D,{#}S {WC/WZ/WCZ} | AND !S into D. D = D & !S. C = parity of result. \* | 2 |
| BITC D,{#}S {WCZ} | Bits D[S[9:5]+S[4:0]:S[4:0]] = C. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]]. | 2 |
| BITH D,{#}S {WCZ} | Bits D[S[9:5]+S[4:0]:S[4:0]] = 1. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]]. | 2 |
| BITL D,{#}S {WCZ} | Bits D[S[9:5]+S[4:0]:S[4:0]] = 0. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]]. | 2 |
| BITNC D,{#}S {WCZ} | Bits D[S[9:5]+S[4:0]:S[4:0]] = !C. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]]. | 2 |
| BITNOT D,{#}S {WCZ} | Toggle bits D[S[9:5]+S[4:0]:S[4:0]]. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]]. | 2 |
| BITNZ D,{#}S {WCZ} | Bits D[S[9:5]+S[4:0]:S[4:0]] = !Z. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]]. | 2 |
| BITRND D,{#}S {WCZ} | Bits D[S[9:5]+S[4:0]:S[4:0]] = RNDs. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]]. | 2 |
| BITZ D,{#}S {WCZ} | Bits D[S[9:5]+S[4:0]:S[4:0]] = Z. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]]. | 2 |
| BMASK D | Get LSB-justified bit mask of size (D[4:0] + 1) into D. D = ($0000\_0002 << D[4:0]) - 1. | 2 |
| BMASK D,{#}S | Get LSB-justified bit mask of size (S[4:0] + 1) into D. D = ($0000\_0002 << S[4:0]) - 1. | 2 |
| CMP D,{#}S {WC/WZ/WCZ} | Compare D to S. C = borrow of (D - S). Z = (D == S). | 2 |
| CMPM D,{#}S {WC/WZ/WCZ} | Compare D to S, get MSB of difference into C. C = MSB of (D - S). Z = (D == S). | 2 |
| CMPR D,{#}S {WC/WZ/WCZ} | Compare S to D (reverse). C = borrow of (S - D). Z = (D == S). | 2 |
| CMPS D,{#}S {WC/WZ/WCZ} | Compare D to S, signed. C = correct sign of (D - S). Z = (D == S). | 2 |
| 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. \* | 2 |
| 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). | 2 |
| CMPX D,{#}S {WC/WZ/WCZ} | Compare D to (S + C), extended. C = borrow of (D - (S + C)). Z = Z AND (D == S + C). | 2 |
| 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). | 2 |
| CRCNIB D,{#}S | Iterate CRC value in D using Q[31:28] and polynomial in S. Like CRCBIT x 4. Q = Q << 4. Use 'REP #n,#1'+SETQ+CRCNIB+CRCNIB+CRCNIB... | 2 |
| 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. \* | 2 |
| DECOD D | Decode D[4:0] into D. D = 1 << D[4:0]. | 2 |
| DECOD D,{#}S | Decode S[4:0] into D. D = 1 << S[4:0]. | 2 |
| 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). \* | 2 |
| 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). \* | 2 |
| FGE D,{#}S {WC/WZ/WCZ} | Force D >= S. If D < S then D = S and C = 1, else D same and C = 0. \* | 2 |
| 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. \* | 2 |
| FLE D,{#}S {WC/WZ/WCZ} | Force D <= S. If D > S then D = S and C = 1, else D same and C = 0. \* | 2 |
| 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. \* | 2 |
| GETBYTE D | Get byte established by prior ALTGB instruction into D. | 2 |
| GETBYTE D,{#}S,#N | Get byte N of S into D. D = {24'b0, S.BYTE[N]). | 2 |
| GETNIB D | Get nibble established by prior ALTGN instruction into D. | 2 |
| GETNIB D,{#}S,#N | Get nibble N of S into D. D = {28'b0, S.NIBBLE[N]). | 2 |
| GETWORD D | Get word established by prior ALTGW instruction into D. | 2 |
| GETWORD D,{#}S,#N | Get word N of S into D. D = {16'b0, S.WORD[N]). | 2 |
| 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. \* | 2 |
| 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. | 2 |
| 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]}. | 2 |
| 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]}. | 2 |
| MODC c {WC} | Modify C according to cccc. C = cccc[{C,Z}]. | 2 |
| MODCZ c,z {WC/WZ/WCZ} | Modify C and Z according to cccc and zzzz. C = cccc[{C,Z}], Z = zzzz[{C,Z}]. | 2 |
| MODZ z {WZ} | Modify Z according to zzzz. Z = zzzz[{C,Z}]. | 2 |
| MOV D,{#}S {WC/WZ/WCZ} | Move S into D. D = S. C = S[31]. \* | 2 |
| 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]]}. | 2 |
| MUL D,{#}S {WZ} | D = unsigned (D[15:0] \* S[15:0]). Z = (S == 0) | (D == 0). | 2 |
| MULS D,{#}S {WZ} | D = signed (D[15:0] \* S[15:0]). Z = (S == 0) | (D == 0). | 2 |
| 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. \* | 2 |
| 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. \* | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. \* | 2 |
| 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). | 2 |
| 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. \* | 2 |
| NEG D {WC/WZ/WCZ} | Negate D. D = -D. C = MSB of result. \* | 2 |
| NEG D,{#}S {WC/WZ/WCZ} | Negate S into D. D = -S. C = MSB of result. \* | 2 |
| NEGC D {WC/WZ/WCZ} | Negate D by C. If C = 1 then D = -D, else D = D. C = MSB of result. \* | 2 |
| 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. \* | 2 |
| NEGNC D {WC/WZ/WCZ} | Negate D by !C. If C = 0 then D = -D, else D = D. C = MSB of result. \* | 2 |
| 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. \* | 2 |
| NEGNZ D {WC/WZ/WCZ} | Negate D by !Z. If Z = 0 then D = -D, else D = D. C = MSB of result. \* | 2 |
| 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. \* | 2 |
| NEGZ D {WC/WZ/WCZ} | Negate D by Z. If Z = 1 then D = -D, else D = D. C = MSB of result. \* | 2 |
| 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. \* | 2 |
| NOT D {WC/WZ/WCZ} | Get !D into D. D = !D. C = !D[31]. \* | 2 |
| NOT D,{#}S {WC/WZ/WCZ} | Get !S into D. D = !S. C = !S[31]. \* | 2 |
| 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. \* | 2 |
| 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. \* | 2 |
| OR D,{#}S {WC/WZ/WCZ} | OR S into D. D = D | S. C = parity of result. \* | 2 |
| 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]. \* | 2 |
| 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]. \* | 2 |
| RCZL D {WC/WZ/WCZ} | Rotate C,Z left through D. D = {D[29:0], C, Z}. C = D[31], Z = D[30]. | 2 |
| RCZR D {WC/WZ/WCZ} | Rotate C,Z right through D. D = {C, Z, D[31:2]}. C = D[1], Z = D[0]. | 2 |
| REV D | Reverse D bits. D = D[0:31]. | 2 |
| 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}. | 2 |
| 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]}. | 2 |
| 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]. \* | 2 |
| ROLBYTE D | Rotate-left byte established by prior ALTGB instruction into D. | 2 |
| ROLBYTE D,{#}S,#N | Rotate-left byte N of S into D. D = {D[23:0], S.BYTE[N]). | 2 |
| ROLNIB D | Rotate-left nibble established by prior ALTGN instruction into D. | 2 |
| ROLNIB D,{#}S,#N | Rotate-left nibble N of S into D. D = {D[27:0], S.NIBBLE[N]). | 2 |
| ROLWORD D | Rotate-left word established by prior ALTGW instruction into D. | 2 |
| ROLWORD D,{#}S,#N | Rotate-left word N of S into D. D = {D[15:0], S.WORD[N]). | 2 |
| 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]. \* | 2 |
| 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]. \* | 2 |
| 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]. \* | 2 |
| SCA D,{#}S {WZ} | Next instruction's S value = unsigned (D[15:0] \* S[15:0]) >> 16. \* | 2 |
| 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. \* | 2 |
| SETBYTE {#}S | Set S[7:0] into byte established by prior ALTSB instruction. | 2 |
| SETBYTE D,{#}S,#N | Set S[7:0] into byte N in D, keeping rest of D same. | 2 |
| SETD D,{#}S | Set D field of D to S[8:0]. D = {D[31:18], S[8:0], D[8:0]}. | 2 |
| SETNIB {#}S | Set S[3:0] into nibble established by prior ALTSN instruction. | 2 |
| SETNIB D,{#}S,#N | Set S[3:0] into nibble N in D, keeping rest of D same. | 2 |
| SETR D,{#}S | Set R field of D to S[8:0]. D = {D[31:28], S[8:0], D[18:0]}. | 2 |
| SETS D,{#}S | Set S field of D to S[8:0]. D = {D[31:9], S[8:0]}. | 2 |
| SETWORD {#}S | Set S[15:0] into word established by prior ALTSW instruction. | 2 |
| SETWORD D,{#}S,#N | Set S[15:0] into word N in D, keeping rest of D same. | 2 |
| SEUSSF D | Relocate and periodically invert bits within D. Returns to original value on 32nd iteration. Forward pattern. | 2 |
| SEUSSR D | Relocate and periodically invert bits within D. Returns to original value on 32nd iteration. Reverse pattern. | 2 |
| 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]. \* | 2 |
| 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]. \* | 2 |
| SIGNX D,{#}S {WC/WZ/WCZ} | Sign-extend D from bit S[4:0]. C = MSB of result. \* | 2 |
| 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]}. | 2 |
| 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]}. | 2 |
| SUB D,{#}S {WC/WZ/WCZ} | Subtract S from D. D = D - S. C = borrow of (D - S). \* | 2 |
| SUBR D,{#}S {WC/WZ/WCZ} | Subtract D from S (reverse). D = S - D. C = borrow of (S - D). \* | 2 |
| SUBS D,{#}S {WC/WZ/WCZ} | Subtract S from D, signed. D = D - S. C = correct sign of (D - S). \* | 2 |
| 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). | 2 |
| 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). | 2 |
| 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). \* | 2 |
| 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). \* | 2 |
| 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). \* | 2 |
| 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). \* | 2 |
| TEST D {WC/WZ/WCZ} | Test D. C = parity of D. Z = (D == 0). | 2 |
| TEST D,{#}S {WC/WZ/WCZ} | Test D with S. C = parity of (D & S). Z = ((D & S) == 0). | 2 |
| TESTB D,{#}S WC/WZ | Test bit S[4:0] of D, write to C/Z. C/Z = D[S[4:0]]. | 2 |
| 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]]. | 2 |
| 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]]. | 2 |
| 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]]. | 2 |
| TESTBN D,{#}S WC/WZ | Test bit S[4:0] of !D, write to C/Z. C/Z = !D[S[4:0]]. | 2 |
| 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]]. | 2 |
| 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]]. | 2 |
| 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]]. | 2 |
| TESTN D,{#}S {WC/WZ/WCZ} | Test D with !S. C = parity of (D & !S). Z = ((D & !S) == 0). | 2 |
| WRC D | Write 0 or 1 to D, according to C. D = {31'b0, C). | 2 |
| WRNC D | Write 0 or 1 to D, according to !C. D = {31'b0, !C). | 2 |
| WRNZ D | Write 0 or 1 to D, according to !Z. D = {31'b0, !Z). | 2 |
| WRZ D | Write 0 or 1 to D, according to Z. D = {31'b0, Z). | 2 |
| XOR D,{#}S {WC/WZ/WCZ} | XOR S into D. D = D ^ S. C = parity of result. \* | 2 |
| XORO32 D | Iterate D with xoroshiro32+ PRNG algorithm and put PRNG result into next instruction's S. | 2 |
| ZEROX D,{#}S {WC/WZ/WCZ} | Zero-extend D above bit S[4:0]. C = MSB of result. \* | 2 |

| Pin & Smart Pin Instructions | | |
| --- | --- | --- |
| **Instruction** | **Description** | **Clocks  Cog, LUT & Hub** |
| Pin | | |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| 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. | 2 |
| TESTP {#}D WC/WZ | Test IN bit of pin D[5:0], write to C/Z. C/Z = IN[D[5:0]]. | 2 |
| 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]]. | 2 |
| 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]]. | 2 |
| 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]]. | 2 |
| TESTPN {#}D WC/WZ | Test !IN bit of pin D[5:0], write to C/Z. C/Z = !IN[D[5:0]]. | 2 |
| 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]]. | 2 |
| 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]]. | 2 |
| 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]]. | 2 |
| Smart Pin | | |
| 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]. | 2 |
| GETSCP D | Get four-channel oscilloscope samples into D. D = {ch3[7:0],ch2[7:0],ch1[7:0],ch0[7:0]}. | 2 |
| RDPIN D,{#}S {WC} | Read smart pin S[5:0] result "Z" into D, acknowledge smart pin. C = modal result. | 2 |
| 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. | 2 |
| SETDACS {#}D | DAC3 = D[31:24], DAC2 = D[23:16], DAC1 = D[15:8], DAC0 = D[7:0]. | 2 |
| SETSCP {#}D | Set four-channel oscilloscope enable to D[6] and set input pin base to D[5:2]. | 2 |
| 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]. | 2 |
| 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]. | 2 |
| 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]. | 2 |

| Branch Instructions | | |
| --- | --- | --- |
| **Instruction** | **Description** | **Clocks  Cog & LUT / Hub** |
| 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. | 4 / 13...20 |
| 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]. | 4 / 13...20 |
| 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. | 5...12 1 / 14...32 1 |
| 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]. | 5...12 1 / 14...32 1 |
| 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. | 5...12 1 / 14...32 1 |
| 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]. | 5...12 1 / 14...32 1 |
| 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]. | 4 / 13...20 |
| 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. | 4 / 13...20 |
| CALLPA {#}D,{#}S | Call to S\*\* by pushing {C, Z, 10'b0, PC[19:0]} onto stack, copy D to PA. | 4 / 13...20 |
| CALLPB {#}D,{#}S | Call to S\*\* by pushing {C, Z, 10'b0, PC[19:0]} onto stack, copy D to PB. | 4 / 13...20 |
| DJF D,{#}S | Decrement D and jump to S\*\* if result is $FFFF\_FFFF. | 2 or 4 / 2 or 13...20 |
| DJNF D,{#}S | Decrement D and jump to S\*\* if result is not $FFFF\_FFFF. | 2 or 4 / 2 or 13...20 |
| DJNZ D,{#}S | Decrement D and jump to S\*\* if result is not zero. | 2 or 4 / 2 or 13...20 |
| DJZ D,{#}S | Decrement D and jump to S\*\* if result is zero. | 2 or 4 / 2 or 13...20 |
| EXECF {#}D | Jump to D[9:0] in cog/LUT and set SKIPF pattern to D[31:10]. PC = {10'b0, D[9:0]}. | 4 / 4 |
| IJNZ D,{#}S | Increment D and jump to S\*\* if result is not zero. | 2 or 4 / 2 or 13...20 |
| IJZ D,{#}S | Increment D and jump to S\*\* if result is zero. | 2 or 4 / 2 or 13...20 |
| JMP #{\}A | Jump to A. If R = 1 then PC += A, else PC = A. "\" forces R = 0. | 4 / 13...20 |
| JMP D {WC/WZ/WCZ} | Jump to D. C = D[31], Z = D[30], PC = D[19:0]. | 4 / 13...20 |
| JMPREL {#}D | Jump ahead/back by D instructions. For cogex, PC += D[19:0]. For hubex, PC += D[17:0] << 2. | 4 / 13...20 |
| REP {#}D,{#}S | Execute next D[8:0] instructions S times. If S = 0, repeat instructions infinitely. If D[8:0] = 0, nothing repeats. | 2 / 2 |
| RESI0 | Resume from INT0. (CALLD $1FE,$1FF WCZ) | 4 / 13...20 |
| RESI1 | Resume from INT1. (CALLD $1F4,$1F5 WCZ) | 4 / 13...20 |
| RESI2 | Resume from INT2. (CALLD $1F2,$1F3 WCZ) | 4 / 13...20 |
| RESI3 | Resume from INT3. (CALLD $1F0,$1F1 WCZ) | 4 / 13...20 |
| RET {WC/WZ/WCZ} | Return by popping stack (K). C = K[31], Z = K[30], PC = K[19:0]. | 4 / 13...20 |
| RETA {WC/WZ/WCZ} | Return by reading hub long (L) at --PTRA. C = L[31], Z = L[30], PC = L[19:0]. | 11...18 1 / 20...40 1 |
| RETB {WC/WZ/WCZ} | Return by reading hub long (L) at --PTRB. C = L[31], Z = L[30], PC = L[19:0]. | 11...18 1 / 20...40 1 |
| RETI0 | Return from INT0. (CALLD $1FF,$1FF WCZ) | 4 / 13...20 |
| RETI1 | Return from INT1. (CALLD $1FF,$1F5 WCZ) | 4 / 13...20 |
| RETI2 | Return from INT2. (CALLD $1FF,$1F3 WCZ) | 4 / 13...20 |
| RETI3 | Return from INT3. (CALLD $1FF,$1F1 WCZ) | 4 / 13...20 |
| SKIP {#}D | Skip instructions per D. Subsequent instructions 0..31 get cancelled for each '1' bit in D[0]..D[31]. | 2 / 2 |
| SKIPF {#}D | Skip cog/LUT instructions fast per D. Like SKIP, but instead of cancelling instructions, the PC leaps over them. | 2 / ILLEGAL |
| TJF D,{#}S | Test D and jump to S\*\* if D is full (D = $FFFF\_FFFF). | 2 or 4 / 2 or 13...20 |
| TJNF D,{#}S | Test D and jump to S\*\* if D is not full (D != $FFFF\_FFFF). | 2 or 4 / 2 or 13...20 |
| TJNS D,{#}S | Test D and jump to S\*\* if D is not signed (D[31] = 0). | 2 or 4 / 2 or 13...20 |
| TJNZ D,{#}S | Test D and jump to S\*\* if D is not zero. | 2 or 4 / 2 or 13...20 |
| TJS D,{#}S | Test D and jump to S\*\* if D is signed (D[31] = 1). | 2 or 4 / 2 or 13...20 |
| TJV D,{#}S | Test D and jump to S\*\* if D overflowed (D[31] != C, C = 'correct sign' from last addition/subtraction). | 2 or 4 / 2 or 13...20 |
| TJZ D,{#}S | Test D and jump to S\*\* if D is zero. | 2 or 4 / 2 or 13...20 |

1 +1 if crosses hub long

| Hub Control, FIFO, & RAM Instructions | | |
| --- | --- | --- |
| **Instruction** | **Description** | **Clocks  Cog & LUT / Hub** |
| Hub Control | | |
| 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. | 2...9, +2 if result / same |
| 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. | 2...9, +2 if result / same |
| COGSTOP {#}D | Stop cog D[3:0]. | 2...9 / same |
| LOCKNEW D {WC} | Request a LOCK. D will be written with the LOCK number (0 to 15). C = 1 if no LOCK available. | 4...11 / same |
| 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. | 2...9, +2 if result / same |
| LOCKRET {#}D | Return LOCK D[3:0] for reallocation. | 2...9 / same |
| 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. | 2...9, +2 if result / same |
| HUBSET {#}D | Set hub configuration to D. | 2...9 / same |
| Hub FIFO | | |
| GETPTR D | Get current FIFO hub pointer into D. | 2 / FIFO IN USE |
| 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. | 2 / FIFO IN USE |
| 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. | 2 or WRFAST finish + 10...17 / FIFO IN USE |
| 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. | 2 or WRFAST finish + 3 / FIFO IN USE |
| RFBYTE D {WC/WZ/WCZ} | Used after RDFAST. Read zero-extended byte from FIFO into D. C = MSB of byte. \* | 2 / FIFO IN USE |
| RFLONG D {WC/WZ/WCZ} | Used after RDFAST. Read long from FIFO into D. C = MSB of long. \* | 2 / FIFO IN USE |
| RFVAR D {WC/WZ/WCZ} | Used after RDFAST. Read zero-extended 1..4-byte value from FIFO into D. C = 0. \* | 2 / FIFO IN USE |
| RFVARS D {WC/WZ/WCZ} | Used after RDFAST. Read sign-extended 1..4-byte value from FIFO into D. C = MSB of value. \* | 2 / FIFO IN USE |
| RFWORD D {WC/WZ/WCZ} | Used after RDFAST. Read zero-extended word from FIFO into D. C = MSB of word. \* | 2 / FIFO IN USE |
| WFBYTE {#}D | Used after WRFAST. Write byte in D[7:0] into FIFO. | 2 / FIFO IN USE |
| WFLONG {#}D | Used after WRFAST. Write long in D[31:0] into FIFO. | 2 / FIFO IN USE |
| WFWORD {#}D | Used after WRFAST. Write word in D[15:0] into FIFO. | 2 / FIFO IN USE |
| Hub RAM | | |
| POPA D {WC/WZ/WCZ} | Read long from hub address --PTRA into D. C = MSB of long. \* | 9...16 1 / 9...26 1 |
| POPB D {WC/WZ/WCZ} | Read long from hub address --PTRB into D. C = MSB of long. \* | 9...16 1 / 9...26 1 |
| RDBYTE D,{#}S/P {WC/WZ/WCZ} | Read zero-extended byte from hub address {#}S/PTRx into D. C = MSB of byte. \* | 9...16 / 9...26 |
| 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. | 9...16 1 / 9...26 1 |
| RDWORD D,{#}S/P {WC/WZ/WCZ} | Read zero-extended word from hub address {#}S/PTRx into D. C = MSB of word. \* | 9...16 1 / 9...26 1 |
| PUSHA {#}D | Write long in D[31:0] to hub address PTRA++. | 3...10 1 / 3...20 1 |
| PUSHB {#}D | Write long in D[31:0] to hub address PTRB++. | 3...10 1 / 3...20 1 |
| 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. | 3...10 1 / 3...20 1 |
| WRBYTE {#}D,{#}S/P | Write byte in D[7:0] to hub address {#}S/PTRx. | 3...10 / 3...20 |
| WRLONG {#}D,{#}S/P | Write long in D[31:0] to hub address {#}S/PTRx. Prior SETQ/SETQ2 invokes cog/LUT block transfer. | 3...10 1 / 3...20 1 |
| WRWORD {#}D,{#}S/P | Write word in D[15:0] to hub address {#}S/PTRx. | 3...10 1 / 3...20 1 |

1 +1 if crosses hub long

| Event Instructions | | |
| --- | --- | --- |
| **Instruction** | **Description** | **Clocks  Cog & LUT / Hub** |
| ADDCT1 D,{#}S | Set CT1 event to trigger on CT = D + S. Adds S into D. | 2 |
| ADDCT2 D,{#}S | Set CT2 event to trigger on CT = D + S. Adds S into D. | 2 |
| ADDCT3 D,{#}S | Set CT3 event to trigger on CT = D + S. Adds S into D. | 2 |
| COGATN {#}D | Strobe "attention" of all cogs whose corresponding bits are high in D[15:0]. | 2 |
| JATN {#}S | Jump to S\*\* if ATN event flag is set. | 2 or 4 / 2 or 13...20 |
| JCT1 {#}S | Jump to S\*\* if CT1 event flag is set. | 2 or 4 / 2 or 13...20 |
| JCT2 {#}S | Jump to S\*\* if CT2 event flag is set. | 2 or 4 / 2 or 13...20 |
| JCT3 {#}S | Jump to S\*\* if CT3 event flag is set. | 2 or 4 / 2 or 13...20 |
| JFBW {#}S | Jump to S\*\* if FBW event flag is set. | 2 or 4 / 2 or 13...20 |
| JINT {#}S | Jump to S\*\* if INT event flag is set. | 2 or 4 / 2 or 13...20 |
| JNATN {#}S | Jump to S\*\* if ATN event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNCT1 {#}S | Jump to S\*\* if CT1 event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNCT2 {#}S | Jump to S\*\* if CT2 event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNCT3 {#}S | Jump to S\*\* if CT3 event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNFBW {#}S | Jump to S\*\* if FBW event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNINT {#}S | Jump to S\*\* if INT event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNPAT {#}S | Jump to S\*\* if PAT event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNQMT {#}S | Jump to S\*\* if QMT event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNSE1 {#}S | Jump to S\*\* if SE1 event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNSE2 {#}S | Jump to S\*\* if SE2 event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNSE3 {#}S | Jump to S\*\* if SE3 event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNSE4 {#}S | Jump to S\*\* if SE4 event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNXFI {#}S | Jump to S\*\* if XFI event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNXMT {#}S | Jump to S\*\* if XMT event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNXRL {#}S | Jump to S\*\* if XRL event flag is clear. | 2 or 4 / 2 or 13...20 |
| JNXRO {#}S | Jump to S\*\* if XRO event flag is clear. | 2 or 4 / 2 or 13...20 |
| JPAT {#}S | Jump to S\*\* if PAT event flag is set. | 2 or 4 / 2 or 13...20 |
| JQMT {#}S | Jump to S\*\* if QMT event flag is set. | 2 or 4 / 2 or 13...20 |
| JSE1 {#}S | Jump to S\*\* if SE1 event flag is set. | 2 or 4 / 2 or 13...20 |
| JSE2 {#}S | Jump to S\*\* if SE2 event flag is set. | 2 or 4 / 2 or 13...20 |
| JSE3 {#}S | Jump to S\*\* if SE3 event flag is set. | 2 or 4 / 2 or 13...20 |
| JSE4 {#}S | Jump to S\*\* if SE4 event flag is set. | 2 or 4 / 2 or 13...20 |
| JXFI {#}S | Jump to S\*\* if XFI event flag is set. | 2 or 4 / 2 or 13...20 |
| JXMT {#}S | Jump to S\*\* if XMT event flag is set. | 2 or 4 / 2 or 13...20 |
| JXRL {#}S | Jump to S\*\* if XRL event flag is set. | 2 or 4 / 2 or 13...20 |
| JXRO {#}S | Jump to S\*\* if XRO event flag is set. | 2 or 4 / 2 or 13...20 |
| POLLATN {WC/WZ/WCZ} | Get ATN event flag into C/Z, then clear it. | 2 |
| POLLCT1 {WC/WZ/WCZ} | Get CT1 event flag into C/Z, then clear it. | 2 |
| POLLCT2 {WC/WZ/WCZ} | Get CT2 event flag into C/Z, then clear it. | 2 |
| POLLCT3 {WC/WZ/WCZ} | Get CT3 event flag into C/Z, then clear it. | 2 |
| POLLFBW {WC/WZ/WCZ} | Get FBW event flag into C/Z, then clear it. | 2 |
| POLLINT {WC/WZ/WCZ} | Get INT event flag into C/Z, then clear it. | 2 |
| POLLPAT {WC/WZ/WCZ} | Get PAT event flag into C/Z, then clear it. | 2 |
| POLLQMT {WC/WZ/WCZ} | Get QMT event flag into C/Z, then clear it. | 2 |
| POLLSE1 {WC/WZ/WCZ} | Get SE1 event flag into C/Z, then clear it. | 2 |
| POLLSE2 {WC/WZ/WCZ} | Get SE2 event flag into C/Z, then clear it. | 2 |
| POLLSE3 {WC/WZ/WCZ} | Get SE3 event flag into C/Z, then clear it. | 2 |
| POLLSE4 {WC/WZ/WCZ} | Get SE4 event flag into C/Z, then clear it. | 2 |
| POLLXFI {WC/WZ/WCZ} | Get XFI event flag into C/Z, then clear it. | 2 |
| POLLXMT {WC/WZ/WCZ} | Get XMT event flag into C/Z, then clear it. | 2 |
| POLLXRL {WC/WZ/WCZ} | Get XRL event flag into C/Z, then clear it. | 2 |
| POLLXRO {WC/WZ/WCZ} | Get XRO event flag into C/Z, then clear it. | 2 |
| SETPAT {#}D,{#}S | Set pin pattern for PAT event. C selects INA/INB, Z selects =/!=, D provides mask value, S provides match value. | 2 |
| SETSE1 {#}D | Set SE1 event configuration to D[8:0]. | 2 |
| SETSE2 {#}D | Set SE2 event configuration to D[8:0]. | 2 |
| SETSE3 {#}D | Set SE3 event configuration to D[8:0]. | 2 |
| SETSE4 {#}D | Set SE4 event configuration to D[8:0]. | 2 |
| WAITATN {WC/WZ/WCZ} | Wait for ATN event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITCT1 {WC/WZ/WCZ} | Wait for CT1 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITCT2 {WC/WZ/WCZ} | Wait for CT2 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITCT3 {WC/WZ/WCZ} | Wait for CT3 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITFBW {WC/WZ/WCZ} | Wait for FBW event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITINT {WC/WZ/WCZ} | Wait for INT event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITPAT {WC/WZ/WCZ} | Wait for PAT event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITSE1 {WC/WZ/WCZ} | Wait for SE1 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITSE2 {WC/WZ/WCZ} | Wait for SE2 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITSE3 {WC/WZ/WCZ} | Wait for SE3 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITSE4 {WC/WZ/WCZ} | Wait for SE4 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITXFI {WC/WZ/WCZ} | Wait for XFI event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITXMT {WC/WZ/WCZ} | Wait for XMT event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITXRL {WC/WZ/WCZ} | Wait for XRL event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |
| WAITXRO {WC/WZ/WCZ} | Wait for XRO event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout. | 2+ |

| Interrupt Instructions | | |
| --- | --- | --- |
| **Instruction** | **Description** | **Clocks  Cog, LUT & Hub** |
| ALLOWI | Allow interrupts (default). | 2 |
| BRK {#}D | If in debug ISR, set next break condition to D. Else, set BRK code to D[7:0] and unconditionally trigger BRK interrupt, if enabled. | 2 |
| COGBRK {#}D | If in debug ISR, trigger asynchronous breakpoint in cog D[3:0]. Cog D[3:0] must have asynchronous breakpoint enabled. | 2 |
| GETBRK D WC/WZ/WCZ | Get breakpoint/cog status into D according to WC/WZ/WCZ. See documentation for details. | 2 |
| NIXINT1 | Cancel INT1. | 2 |
| NIXINT2 | Cancel INT2. | 2 |
| NIXINT3 | Cancel INT3. | 2 |
| SETINT1 {#}D | Set INT1 source to D[3:0]. | 2 |
| SETINT2 {#}D | Set INT2 source to D[3:0]. | 2 |
| SETINT3 {#}D | Set INT3 source to D[3:0]. | 2 |
| STALLI | Stall Interrupts. | 2 |
| TRGINT1 | Trigger INT1, regardless of STALLI mode. | 2 |
| TRGINT2 | Trigger INT2, regardless of STALLI mode. | 2 |
| TRGINT3 | Trigger INT3, regardless of STALLI mode. | 2 |

| Register Indirection Instructions | | |
| --- | --- | --- |
| **Instruction** | **Description** | **Clocks  Cog & LUT / Hub** |
| ALTB D,{#}S | Alter D field of next instruction to D[13:5]. | 2 |
| ALTB D,{#}S | Alter D field of next instruction to (D[13:5] + S) & $1FF. D += sign-extended S[17:9]. | 2 |
| ALTD D | Alter D field of next instruction to D[8:0]. | 2 |
| ALTD D,{#}S | Alter D field of next instruction to (D + S) & $1FF. D += sign-extended S[17:9]. | 2 |
| ALTGB D | Alter subsequent GETBYTE/ROLBYTE instruction. Next S field = D[10:2], N field = D[1:0]. | 2 |
| 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]. | 2 |
| ALTGN D | Alter subsequent GETNIB/ROLNIB instruction. Next S field = D[11:3], N field = D[2:0]. | 2 |
| 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]. | 2 |
| ALTGW D | Alter subsequent GETWORD/ROLWORD instruction. Next S field = D[9:1], N field = D[0]. | 2 |
| 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]. | 2 |
| ALTI D | Execute D in place of next instruction. D stays same. | 2 |
| ALTI D,{#}S | Substitute next instruction's I/R/D/S fields with fields from D, per S. Modify D per S. | 2 |
| ALTR D | Alter result register address (normally D field) of next instruction to D[8:0]. | 2 |
| ALTR D,{#}S | Alter result register address (normally D field) of next instruction to (D + S) & $1FF. D += sign-extended S[17:9]. | 2 |
| ALTS D | Alter S field of next instruction to D[8:0]. | 2 |
| ALTS D,{#}S | Alter S field of next instruction to (D + S) & $1FF. D += sign-extended S[17:9]. | 2 |
| ALTSB D | Alter subsequent SETBYTE instruction. Next D field = D[10:2], N field = D[1:0]. | 2 |
| 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]. | 2 |
| ALTSN D | Alter subsequent SETNIB instruction. Next D field = D[11:3], N field = D[2:0]. | 2 |
| 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]. | 2 |
| ALTSW D | Alter subsequent SETWORD instruction. Next D field = D[9:1], N field = D[0]. | 2 |
| 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]. | 2 |

| CORDIC Solver Instructions | | |
| --- | --- | --- |
| **Instruction** | **Description** | **Clocks  Cog, LUT & Hub** |
| GETQX D {WC/WZ/WCZ} | Retrieve CORDIC result X into D. Waits, in case result not ready. C = X[31]. 1 | 2...58 |
| GETQY D {WC/WZ/WCZ} | Retrieve CORDIC result Y into D. Waits, in case result not ready. C = Y[31]. 1 | 2...58 |
| QDIV {#}D,{#}S | Begin CORDIC unsigned division of {SETQ value or 32'b0, D} / S. GETQX/GETQY retrieves quotient/remainder. | 2...9 |
| QEXP {#}D | Begin CORDIC logarithm-to-number conversion of D. GETQX retrieves number. | 2...9 |
| QFRAC {#}D,{#}S | Begin CORDIC unsigned division of {D, SETQ value or 32'b0} / S. GETQX/GETQY retrieves quotient/remainder. | 2...9 |
| QLOG {#}D | Begin CORDIC number-to-logarithm conversion of D. GETQX retrieves log {5'whole\_exponent, 27'fractional\_exponent}. | 2...9 |
| QMUL {#}D,{#}S | Begin CORDIC unsigned multiplication of D \* S. GETQX/GETQY retrieves lower/upper product. | 2...9 |
| QROTATE {#}D,{#}S | Begin CORDIC rotation of point (D, SETQ value or 32'b0) by angle S. GETQX/GETQY retrieves X/Y. | 2...9 |
| QSQRT {#}D,{#}S | Begin CORDIC square root of {S, D}. GETQX retrieves root. | 2...9 |
| QVECTOR {#}D,{#}S | Begin CORDIC vectoring of point (D, S). GETQX/GETQY retrieves length/angle. | 2...9 |

1 Z = (result == 0)

| Color Space Converter and Pixel Mixer Instructions | | |
| --- | --- | --- |
| **Instruction** | **Description** | **Clocks  Cog, LUT & Hub** |
| Color Space Converter | | |
| SETCFRQ {#}D | Set the colorspace converter "CFRQ" parameter to D[31:0]. | 2 |
| SETCI {#}D | Set the colorspace converter "CI" parameter to D[31:0]. | 2 |
| SETCMOD {#}D | Set the colorspace converter "CMOD" parameter to D[8:0]. | 2 |
| SETCQ {#}D | Set the colorspace converter "CQ" parameter to D[31:0]. | 2 |
| SETCY {#}D | Set the colorspace converter "CY" parameter to D[31:0]. | 2 |
| Pixel Mixer | | |
| ADDPIX D,{#}S | Add bytes of S into bytes of D, with $FF saturation. | 7 |
| BLNPIX D,{#}S | Alpha-blend bytes of S into bytes of D, using SETPIV value. | 7 |
| MIXPIX D,{#}S | Mix bytes of S into bytes of D, using SETPIX and SETPIV values. | 7 |
| MULPIX D,{#}S | Multiply bytes of S into bytes of D, where $FF = 1.0 and $00 = 0.0. | 7 |
| SETPIV {#}D | Set BLNPIX/MIXPIX blend factor to D[7:0]. | 2 |
| SETPIX {#}D | Set MIXPIX mode to D[5:0]. | 2 |

| Lookup Table, Streamer, and Misc Instructions | | |
| --- | --- | --- |
| **Instruction** | **Description** | **Clocks  Cog & LUT / Hub** |
| Lookup Table | | |
| RDLUT D,{#}S/P {WC/WZ/WCZ} | Read data from LUT address {#}S/PTRx into D. C = MSB of data. \* | 3 |
| SETLUTS {#}D | If D[0] = 1 then enable LUT sharing, where LUT writes within the adjacent odd/even companion cog are copied to this cog's LUT. | 2 |
| WRLUT {#}D,{#}S/P | Write D to LUT address {#}S/PTRx. | 2 |
| Streamer | | |
| GETXACC D | Get the streamer's Goertzel X accumulator into D and the Y accumulator into the next instruction's S, clear accumulators. | 2 |
| SETXFRQ {#}D | Set streamer NCO frequency to D. | 2 |
| XCONT {#}D,{#}S | Buffer new streamer command to be issued on final NCO rollover of current command, continuing phase. | 2+ |
| XINIT {#}D,{#}S | Issue streamer command immediately, zeroing phase. | 2 |
| XSTOP | Stop streamer immediately. | 2 |
| XZERO {#}D,{#}S | Buffer new streamer command to be issued on final NCO rollover of current command, zeroing phase. | 2+ |
| Miscellaneous | | |
| AUGD #n | Queue #n to be used as upper 23 bits for next #D occurrence, so that the next 9-bit #D will be augmented to 32 bits. | 2 |
| AUGS #n | Queue #n to be used as upper 23 bits for next #S occurrence, so that the next 9-bit #S will be augmented to 32 bits. | 2 |
| 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. | 2 |
| GETRND WC/WZ/WCZ | Get RND into C/Z. C = RND[31], Z = RND[30], unique per cog. | 2 |
| 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. | 2 |
| NOP | No operation. | 2 |
| POP D {WC/WZ/WCZ} | Pop stack (K). D = K. C = K[31]. \* | 2 |
| PUSH {#}D | Push D onto stack. | 2 |
| SETQ {#}D | Set Q to D. Use before RDLONG/WRLONG/WMLONG to set block transfer. Also used before MUXQ/COGINIT/QDIV/QFRAC/QROTATE/WAITxxx. | 2 |
| SETQ2 {#}D | Set Q to D. Use before RDLONG/WRLONG/WMLONG to set LUT block transfer. | 2 |
| 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. | 2 + D |

# 

# PROPELLER 2 RESERVED WORDS (SPIN2 + PASM2)

Predefined symbols recognized by the compiler to have special meaning.

## \_ (leading underscore)

| \_C  \_C\_AND\_NZ  \_C\_AND\_Z  \_C\_EQ\_Z | \_C\_NE\_Z  \_C\_OR\_NZ  \_C\_OR\_Z  \_CLR | \_E  \_GE  \_GT  \_LE | \_LT  \_NC  \_NC\_AND\_NZ  \_NC\_AND\_Z | \_NC\_OR\_NZ  \_NC\_OR\_Z  \_NE  \_NZ | \_NZ\_AND\_C  \_NZ\_AND\_NC  \_NZ\_OR\_C  \_NZ\_OR\_NC | \_RET\_  \_SET  \_Z  \_Z\_AND\_C | \_Z\_AND\_NC  \_Z\_EQ\_C  \_Z\_NE\_C  \_Z\_OR\_C | \_Z\_OR\_NC |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |

## A - B

| ABORT  ABS  ADD  ADDBITS  ADDCT1 | ADDCT2  ADDCT3  ADDPINS  ADDPIX  ADDS | ADDSX  ADDX  AKPIN  ALIGNL  ALIGNW | ALLOWI  ALT  ALTB  ALTD  ALTGB | ALTGN  ALTGW  ALTI  ALTR  ALTS | ALTSB  ALTSN  ALTSW  AND  ANDC | ANDN  ANDZ  ARCHIVE  ASMCLK  AUGD | AUGS  BACKCOLOR  BITC  BITH  BITL | BITMAP  BITNC  BITNOT  BITNZ  BITRND | BITZ  BLACK  BLNPIX  BLUE  BMASK | BOX  BRK  BYTE  BYTEFILL  BYTEMOVE | BYTES\_1BIT  BYTES\_2BIT  BYTES\_4BIT |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

## C - D

| CALL  CALLA  CALLB  CALLD  CALLPA  CALLPB  CARTESIAN  CASE  CASE\_FAST  CHANNEL | CIRCLE  CLEAR  CLKFREQ  CLKMODE  CLKSET  CLOSE  CMP  CMPM  CMPR  CMPS | CMPSUB  CMPSX  CMPX  COGATN  COGBRK  COGCHK  COGEXEC  COGEXEC\_NEW  COGEXEC\_NEW\_PAIR  COGID | COGINIT  COGSPIN  COGSTOP  COLOR  CON  CRCBIT  CRCNIB  CYAN  DAT  DEBUG | DEBUG\_BAUD  DEBUG\_COGS  DEBUG\_DELAY  DEBUG\_DISPLAY\_LEFT  DEBUG\_DISPLAY\_TOP  DEBUG\_HEIGHT  DEBUG\_LEFT  DEBUG\_LOG\_SIZE  DEBUG\_PIN  DEBUG\_TIMESTAMP | DEBUG\_TOP  DEBUG\_WIDTH  DEBUG\_WINDOWS\_OFF  DECMOD  DECOD  DEPTH  DEV  DIRA  DIRB  DIRC | DIRH  DIRL  DIRNC  DIRNOT  DIRNZ  DIRRND  DIRZ  DJF  DJNF  DJNZ | DJZ  DLY  DOT  DOTSIZE  DRVC  DRVH  DRVL  DRVNC  DRVNOT | DRVRND  DRVZ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |

## E - F

| ELSE  ELSEIF  ELSEIFNOT  ENCOD  END | EVENT\_ATN  EVENT\_CT1  EVENT\_CT2  EVENT\_CT3  EVENT\_FBW | EVENT\_INT  EVENT\_PAT  EVENT\_QMT  EVENT\_SE1  EVENT\_SE2 | EVENT\_SE3  EVENT\_SE4  EVENT\_XFI  EVENT\_XMT  EVENT\_XRL | EVENT\_XRO  EXECF  FABS  FALSE  FBLOCK | FDEC  FDEC\_  FDEC\_ARRAY  FDEC\_ARRAY\_  FDEC\_REG\_ARRAY | FDEC\_REG\_ARRAY\_  FFT  FGE  FGES  FILE | FIT  FLE  FLES  FLOAT  FLTC | FLTH  FLTL  FLTNC  FLTNOT  FLTNZ | FLTRND  FLTZ  FRAC  FROM  FSQRT | FVAR  FVARS |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

## G - H

| GETBRK  GETBYTE  GETCT | GETMS  GETNIB  GETPTR | GETQX  GETQY  GETREGS | GETRND  GETSCP  GETSEC | GETWORD  GETXACC  GREEN | GREY  HIDEXY  HOLDOFF | HSV16  HSV16W  HSV16X | HSV8  HSV8W  HSV8X | HUBEXEC  HUBEXEC\_NEW  HUBEXEC\_NEW\_PAIR | HUBSET |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

## I - J

| IF  IF\_00  IF\_0000  IF\_0001  IF\_0010  IF\_0011  IF\_01  IF\_0100  IF\_0101  IF\_0110  IF\_0111  IF\_0X | IF\_10  IF\_1000  IF\_1001  IF\_1010  IF\_1011  IF\_11  IF\_1100  IF\_1101  IF\_1110  IF\_1111  IF\_1X  IF\_A | IF\_AE  IF\_ALWAYS  IF\_B  IF\_BE  IF\_C  IF\_C\_AND\_NZ  IF\_C\_AND\_Z  IF\_C\_EQ\_Z  IF\_C\_NE\_Z  IF\_C\_OR\_NZ  IF\_C\_OR\_Z  IF\_DIFF | IF\_E  IF\_GE  IF\_GT  IF\_LE  IF\_LT  IF\_NC  IF\_NC\_AND\_NZ  IF\_NC\_AND\_Z  IF\_NC\_OR\_NZ  IF\_NC\_OR\_Z  IF\_NE  IF\_NOT\_00 | IF\_NOT\_01  IF\_NOT\_10  IF\_NOT\_11  IF\_NZ  IF\_NZ\_AND\_C  IF\_NZ\_AND\_NC  IF\_NZ\_OR\_C  IF\_NZ\_OR\_NC  IF\_SAME  IF\_X0  IF\_X1  IF\_Z | IF\_Z\_AND\_C  IF\_Z\_AND\_NC  IF\_Z\_EQ\_C  IF\_Z\_NE\_C  IF\_Z\_OR\_C  IF\_Z\_OR\_NC  IFNOT  IJMP1  IJMP2  IJMP3  IJNZ  IJZ | INA  INB  INCMOD  INT\_OFF  IRET1  IRET2  IRET3  JATN  JCT1  JCT2  JCT3  JFBW | JINT  JMP  JMPREL  JNATN  JNCT1  JNCT2  JNCT3  JNFBW  JNINT  JNPAT  JNQMT  JNSE1 | JNSE2  JNSE3  JNSE4  JNXFI  JNXMT  JNXRL  JNXRO  JPAT  JQMT  JSE1  JSE2  JSE3 | JSE4  JXFI  JXMT  JXRL  JXRO |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

## L - M

| LINE  LINESIZE  LOC  LOCKCHK  LOCKNEW | LOCKREL  LOCKRET  LOCKTRY  LOGIC  LOGSCALE | LONG  LONGFILL  LONGMOVE  LONGS\_16BIT  LONGS\_1BIT | LONGS\_2BIT  LONGS\_4BIT  LONGS\_8BIT  LOOKDOWN  LOOKDOWNZ | LOOKUP  LOOKUPZ  LSTR  LSTR\_  LUMA8 | LUMA8W  LUMA8X  LUT1  LUT2  LUT4 | LUT8  LUTCOLORS  MAG  MAGENTA  MERGEB | MERGEW  MIDI  MIXPIX  MODC  MODCZ | MODZ  MOV  MOVBYTS  MUL  MULDIV64 | MULPIX  MULS  MUXC  MUXNC  MUXNIBS | MUXNITS  MUXNZ  MUXQ  MUXZ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

## N - O

| NAN  NEG  NEGC  NEGNC | NEGNZ  NEGX  NEGZ  NEWCOG | NEXT  NIXINT1  NIXINT2  NIXINT3 | NOP  NOT  OBJ  OBOX | ONES  OPACITY  OR  ORANGE | ORC  ORG  ORGF  ORGH | ORIGIN  ORZ  OTHER  OUTA | OUTB  OUTC  OUTH  OUTL | OUTNC  OUTNOT  OUTNZ  OUTRND | OUTZ  OVAL |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

## 

## P

| P\_ADC  P\_ADC\_100X  P\_ADC\_10X  P\_ADC\_1X  P\_ADC\_30X  P\_ADC\_3X  P\_ADC\_EXT  P\_ADC\_FLOAT  P\_ADC\_GIO  P\_ADC\_SCOPE  P\_ADC\_VIO  P\_AND\_AB  P\_ASYNC\_IO  P\_ASYNC\_RX  P\_ASYNC\_TX  P\_BITDAC  P\_CHANNEL  P\_COMPARE\_AB  P\_COMPARE\_AB\_FB  P\_COUNT\_HIGHS | P\_COUNT\_RISES  P\_COUNTER\_HIGHS  P\_COUNTER\_PERIODS  P\_COUNTER\_TICKS  P\_DAC\_124R\_3V  P\_DAC\_600R\_2V  P\_DAC\_75R\_2V  P\_DAC\_990R\_3V  P\_DAC\_DITHER\_PWM  P\_DAC\_DITHER\_RND  P\_DAC\_NOISE  P\_EVENTS\_TICKS  P\_FILT0\_AB  P\_FILT1\_AB  P\_FILT2\_AB  P\_FILT3\_AB  P\_HIGH\_100UA  P\_HIGH\_10UA  P\_HIGH\_150K  P\_HIGH\_15K | P\_HIGH\_1K5  P\_HIGH\_1MA  P\_HIGH\_FAST  P\_HIGH\_FLOAT  P\_HIGH\_TICKS  P\_INVERT\_A  P\_INVERT\_B  P\_INVERT\_IN  P\_INVERT\_OUT  P\_INVERT\_OUTPUT  P\_LEVEL\_A  P\_LEVEL\_A\_FBN  P\_LEVEL\_B\_FBN  P\_LEVEL\_B\_FBP  P\_LOCAL\_A  P\_LOCAL\_B  P\_LOGIC\_A  P\_LOGIC\_A\_FB  P\_LOGIC\_B\_FB  P\_LOW\_100UA | P\_LOW\_10UA  P\_LOW\_150K  P\_LOW\_15K  P\_LOW\_1K5  P\_LOW\_1MA  P\_LOW\_FAST  P\_LOW\_FLOAT  P\_MINUS1\_A  P\_MINUS1\_B  P\_MINUS2\_A  P\_MINUS2\_B  P\_MINUS3\_A  P\_MINUS3\_B  P\_NCO\_DUTY  P\_NCO\_FREQ  P\_NORMAL  P\_OE  P\_OR\_AB  P\_OUTBIT\_A  P\_OUTBIT\_B | P\_PASS\_AB  P\_PERIODS\_HIGHS  P\_PERIODS\_TICKS  P\_PLUS1\_A  P\_PLUS1\_B  P\_PLUS2\_A  P\_PLUS2\_B  P\_PLUS3\_A  P\_PLUS3\_B  P\_PULSE  P\_PWM\_SAWTOOTH  P\_PWM\_SMPS  P\_PWM\_TRIANGLE  P\_QUADRATURE  P\_REG\_UP  P\_REG\_UP\_DOWN  P\_REPOSITORY  P\_SCHMITT\_A  P\_SCHMITT\_A\_FB  P\_SCHMITT\_B\_FB | P\_STATE\_TICKS  P\_SYNC\_IO  P\_SYNC\_RX  P\_SYNC\_TX  P\_TRANSITION  P\_TRUE\_A  P\_TRUE\_B  P\_TRUE\_IN  P\_TRUE\_OUT  P\_TRUE\_OUTPUT  P\_TT\_00  P\_TT\_01  P\_TT\_10  P\_TT\_11  P\_USB\_PAIR  P\_XOR\_AB  PA  PB  PC\_KEY  PC\_MOUSE | PI  PINCLEAR  PINF  PINFLOAT  PINH  PINHIGH  PINL  PINLOW  PINR  PINREAD  PINSTART  PINT  PINTOGGLE  PINW  PINWRITE  PLOT  POLAR  POLLATN  POLLCT  POLLCT1 | POLLCT2  POLLCT3  POLLFBW  POLLINT  POLLPAT  POLLQMT  POLLSE1  POLLSE2  POLLSE3  POLLSE4  POLLXFI  POLLXMT  POLLXRL  POLLXRO  POLXY  POP  POPA  POPB  POS  POSX | PR0  PR1  PR2  PR3  PR4  PR5  PR6  PR7  PRECISE  PRECOMPILE  PRI  PTRA  PTRB  PUB  PUSH  PUSHA  PUSHB |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |

## Q - R

| QCOS  QDIV  QEXP  QFRAC  QLOG  QMUL | QROTATE  QSIN  QSQRT  QUIT  QVECTOR  RANGE | RATE  RCL  RCR  RCZL  RCZR  RDBYTE | RDFAST  RDLONG  RDLUT  RDPIN  RDWORD  RECV | RED  REG  REGEXEC  REGLOAD  REP  REPEAT | RES  RESI0  RESI1  RESI2  RESI3  RET | RETA  RETB  RETI0  RETI1  RETI2  RETI3 | RETURN  REV  RFBYTE  RFLONG  RFVAR  RFVARS | RFWORD  RGB16  RGB24  RGB8  RGBEXP  RGBI8 | RGBI8W  RGBI8X  RGBSQZ  ROL  ROLBYTE  ROLNIB | ROLWORD  ROR  ROTXY  ROUND  RQPIN |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

## S - T

| SAL  SAMPLES  SAR  SAVE  SBIN  SBIN\_  SBIN\_BYTE\_  SBIN\_BYTE\_ARRAY  SBIN\_BYTE\_ARRAY\_  SBIN\_LONG  SBIN\_LONG\_  SBIN\_LONG\_ARRAY  SBIN\_LONG\_ARRAY\_  SBIN\_REG\_ARRAY  SBIN\_REG\_ARRAY\_  SBIN\_WORD  SBIN\_WORD\_  SBIN\_WORD\_ARRAY | SBIN\_WORD\_ARRAY\_  SCA  SCAS  SCOPE  SCOPE\_XY  SCROLL SDEC  SDEC\_  SDEC\_BYTE  SDEC\_BYTE\_  SDEC\_BYTE\_ARRAY  SDEC\_BYTE\_ARRAY\_  SDEC\_LONG  SDEC\_LONG\_  SDEC\_LONG\_ARRAY  SDEC\_LONG\_ARRAY\_  SDEC\_REG\_ARRAY  SDEC\_REG\_ARRAY\_ | SDEC\_WORD  SDEC\_WORD\_  SDEC\_WORD\_ARRAY  SDEC\_WORD\_ARRAY\_  SEND  SET  SETBYTE  SETCFRQ  SETCI  SETCMOD  SETCQ  SETCY  SETD  SETDACS  SETINT1  SETINT2  SETINT3  SETLUTS | SETNIB  SETPAT  SETPIV  SETPIX  SETQ  SETQ2  SETR  SETREGS  SETS  SETSCP  SETSE1  SETSE2  SETSE3  SETSE4  SETWORD  SETXFRQ  SEUSSF  SEUSSR | SHEX  SHEX\_  SHEX\_BYTE  SHEX\_BYTE\_  SHEX\_BYTE\_ARRAY  SHEX\_BYTE\_ARRAY\_  SHEX\_LONG  SHEX\_LONG\_  SHEX\_LONG\_ARRAY  SHEX\_LONG\_ARRAY\_  SHEX\_REG\_ARRAY  SHEX\_REG\_ARRAY\_  SHEX\_WORD  SHEX\_WORD\_  SHEX\_WORD\_ARRAY  SHEX\_WORD\_ARRAY\_  SHL  SHR | SIGNED  SIGNX  SIZE  SKIP  SKIPF  SPACING  SPECTRO  SPLITB  SPLITW  SPRITE  SPRITEDEF  SQRT  STALLI  STEP  STRCOMP  STRING  STRSIZE  SUB | SUBR  SUBS  SUBSX  SUBX  SUMC  SUMNC  SUMNZ  SUMZ  TERM  TEST  TESTB  TESTBN  TESTN  TESTP  TESTPN  TEXT  TEXTANGLE  TEXTSIZE | TEXTSTYLE  TITLE  TJF  TJNF  TJNS  TJNZ  TJS  TJV  TJZ  TO  TRACE  TRGINT1  TRGINT2  TRGINT3  TRIGGER  TRUE  TRUNC |
| --- | --- | --- | --- | --- | --- | --- | --- |

## U, V, W

| UBIN  UBIN\_  UBIN\_BYTE  UBIN\_BYTE\_  UBIN\_BYTE\_ARRAY  UBIN\_BYTE\_ARRAY\_  UBIN\_LONG  UBIN\_LONG\_  UBIN\_LONG\_ARRAY  UBIN\_LONG\_ARRAY\_  UBIN\_REG\_ARRAY  UBIN\_REG\_ARRAY\_  UBIN\_WORD  UBIN\_WORD\_  UBIN\_WORD\_ARRAY | UBIN\_WORD\_ARRAY\_  UDEC  UDEC\_  UDEC\_BYTE  UDEC\_BYTE\_  UDEC\_BYTE\_ARRAY  UDEC\_BYTE\_ARRAY\_  UDEC\_LONG  UDEC\_LONG\_  UDEC\_LONG\_ARRAY  UDEC\_LONG\_ARRAY\_  UDEC\_REG\_ARRAY  UDEC\_REG\_ARRAY\_  UDEC\_WORD  UDEC\_WORD\_ | UDEC\_WORD\_ARRAY  UDEC\_WORD\_ARRAY\_  UHEX  UHEX\_  UHEX\_BYTE  UHEX\_BYTE\_  UHEX\_BYTE\_ARRAY  UHEX\_BYTE\_ARRAY\_  UHEX\_LONG  UHEX\_LONG\_  UHEX\_LONG\_ARRAY  UHEX\_LONG\_ARRAY\_  UHEX\_REG\_ARRAY  UHEX\_REG\_ARRAY\_  UHEX\_WORD | UHEX\_WORD\_  UHEX\_WORD\_ARRAY  UHEX\_WORD\_ARRAY\_  UNTIL  UPDATE  VAR  VARBASE  WAITATN  WAITCT  WAITCT1  WAITCT2  WAITCT3  WAITFBW  WAITINT  WAITMS | WAITPAT  WAITSE1  WAITSE2  WAITSE3  WAITSE4  WAITUS  WAITX  WAITXFI  WAITXMT  WAITXRL  WAITXRO  WC  WCZ  WFBYTE  WFLONG | WFWORD  WHILE  WHITE  WINDOW  WMLONG  WORD  WORDFILL  WORDMOVE  WORDS\_1BIT  WORDS\_2BIT  WORDS\_4BIT  WORDS\_8BIT  WRBYTE  WRC  WRFAST | WRLONG  WRLUT  WRNC  WRNZ  WRPIN  WRWORD  WRZ  WXPIN  WYPIN  WZ |
| --- | --- | --- | --- | --- | --- | --- |

## X, Y, Z

| X\_16P\_2DAC8\_WFWORD  X\_16P\_4DAC4\_WFWORD  X\_1ADC8\_0P\_1DAC8\_WFBYTE  X\_1ADC8\_8P\_2DAC8\_WFWORD  X\_1P\_1DAC1\_WFBYTE  X\_2ADC8\_0P\_2DAC8\_WFWORD  X\_2ADC8\_16P\_4DAC8\_WFLONG  X\_2P\_1DAC2\_WFBYTE  X\_2P\_2DAC1\_WFBYTE  X\_32P\_4DAC8\_WFLONG  X\_4ADC8\_0P\_4DAC8\_WFLONG  X\_4P\_1DAC4\_WFBYTE  X\_4P\_2DAC2\_WFBYTE  X\_4P\_4DAC1\_WFBYTE  X\_8P\_1DAC8\_WFBYTE  X\_8P\_2DAC4\_WFBYTE | X\_8P\_4DAC2\_WFBYTE  X\_ALT\_OFF  X\_ALT\_ON  X\_DACS\_0\_0\_0\_0  X\_DACS\_0\_0\_X\_X  X\_DACS\_0\_X\_X\_X  X\_DACS\_0N0\_0N0  X\_DACS\_0N0\_X\_X  X\_DACS\_1\_0\_1\_0  X\_DACS\_1\_0\_X\_X  X\_DACS\_1N1\_0N0  X\_DACS\_3\_2\_1\_0  X\_DACS\_OFF  X\_DACS\_X\_0\_X\_X  X\_DACS\_X\_X\_0\_0  X\_DACS\_X\_X\_0\_X | X\_DACS\_X\_X\_0N0  X\_DACS\_X\_X\_1\_0  X\_DACS\_X\_X\_X\_0  X\_DDS\_GOERTZEL\_SINC1  X\_DDS\_GOERTZEL\_SINC2  X\_IMM\_16X2\_1DAC2  X\_IMM\_16X2\_2DAC1  X\_IMM\_16X2\_LUT  X\_IMM\_1X32\_4DAC8  X\_IMM\_2X16\_2DAC8  X\_IMM\_2X16\_4DAC4  X\_IMM\_32X1\_1DAC1  X\_IMM\_32X1\_LUT  X\_IMM\_4X8\_1DAC8  X\_IMM\_4X8\_2DAC4  X\_IMM\_4X8\_4DAC2 | X\_IMM\_4X8\_LUT  X\_IMM\_8X4\_1DAC4  X\_IMM\_8X4\_2DAC2  X\_IMM\_8X4\_4DAC1  X\_IMM\_8X4\_LUT  X\_PINS\_OFF  X\_PINS\_ON  X\_RFBYTE\_1P\_1DAC1  X\_RFBYTE\_2P\_1DAC2  X\_RFBYTE\_2P\_2DAC1  X\_RFBYTE\_4P\_1DAC4  X\_RFBYTE\_4P\_2DAC2  X\_RFBYTE\_4P\_4DAC1  X\_RFBYTE\_8P\_1DAC8  X\_RFBYTE\_8P\_2DAC4  X\_RFBYTE\_8P\_4DAC2 | X\_RFBYTE\_LUMA8  X\_RFBYTE\_RGB8  X\_RFBYTE\_RGBI8  X\_RFLONG\_16X2\_LUT  X\_RFLONG\_32P\_4DAC8  X\_RFLONG\_32X1\_LUT  X\_RFLONG\_4X8\_LUT  X\_RFLONG\_8X4\_LUT  X\_RFLONG\_RGB24  X\_RFWORD\_16P\_2DAC8  X\_RFWORD\_16P\_4DAC4  X\_RFWORD\_RGB16  X\_WRITE\_OFF  X\_WRITE\_ON  XCONT  XINIT | XOR  XORC  XORO32  XORZ  XSTOP  XYPOL  XZERO  YELLOW  ZEROX  ZSTR  ZSTR\_ |
| --- | --- | --- | --- | --- | --- |

##### 

# CHANGE LOG

| **Date** | **Notes** |
| --- | --- |
| 11/11/2021 | Live draft workspace published. |
| 11/02/2022 | Official release. |

# PARALLAX INCORPORATED

| **Parallax Inc.**  **599 Menlo Drive, Suite 100**  **Rocklin, CA 95765**  **USA** | **Office: +1 916-624-8333**  **Toll Free US: 888-512-1024** | [**sales@parallax.com**](mailto:sales@parallax.com)  [**support@parallax.com**](mailto:support@parallax.com) | [**www.parallax.com/p2**](http://www.parallax.com/p2)  [**forums.parallax.com**](https://forums.parallax.com) |
| --- | --- | --- | --- |
| **Purchase of the P2X8C4M64P does not include any license to emulate any other device nor to communicate via any specific proprietary protocol; P2X8C4M64P connectivity objects and code examples provided or referenced by Parallax, Inc. are NOT licensed and are provided for research and development purposes only; end users must seek permission to use licensed protocols for their applications and products from the protocol license holders.**  **Parallax, Inc. makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Parallax, Inc. assume any liability arising out of the application or use of any product, and specifically disclaims any and all liability, including without limitation consequential or incidental damages even if Parallax, Inc. has been advised of the possibility of such damages.**  **Copyright © 2022 Parallax, Inc. All rights are reserved. Parallax, the Parallax logo, the P2 logo, and Propeller are trademarks of Parallax, Inc.** | | | |