P16X64A
'P2'
ASSEMBLY MANUAL

PRELIMINARY - DOCUMENT UNDER CONSTRUCTION
web document link

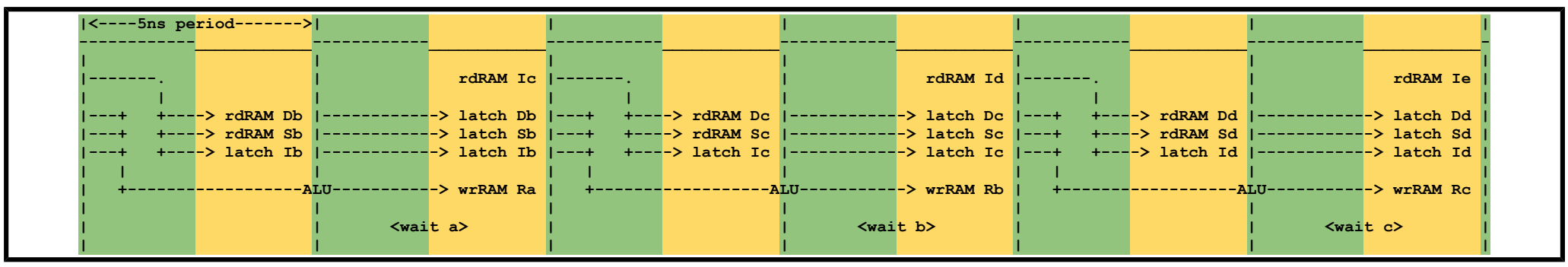
CONTENTS

COG REGISTERS

ADDR	NAME	READ	WRITE
000-1F7		RAM	RAM
1F8	**PTRA**	PTRA	RAM+PTRA
1F9	**PTRB**	PTRB	RAM+PTRB
1FA	**INA**	INA	RAM
1FB	**INB**	INB	RAM
1FC	**OUTA**	RAM	RAM+OUTA
1FD	**OUTB**	RAM	RAM+OUTB
1FE	**DIRA**	RAM	RAM+DIRA
1FF	**DIRB**	RAM	RAM+DIRB

Note: Instructions writing OUTx and DIRx also write to the shadow RAM. Therefore the shadow RAM retains the last values written to the OUTx and DIRx registers, which can then be read back (by reading the shadow RAM). This is useful in read-modify-write instructions such as: **`ANDN DIRx,#$003`** which turns off the DIRx bits [1:0].

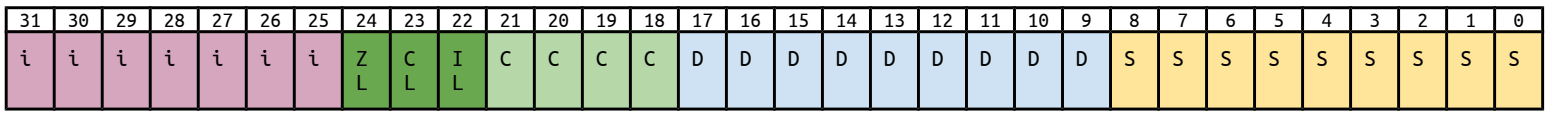
TIMING



P16X64A INSTRUCTION SET

Notes:
Unless stated otherwise if the WZ effect is specified, the Z flag is set (1) if the resulting Value equals zero.

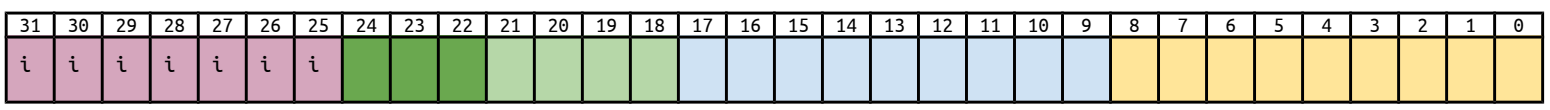
INSTRUCTION FORMAT (GENERAL)



All instructions utilise a 32-bit instruction:

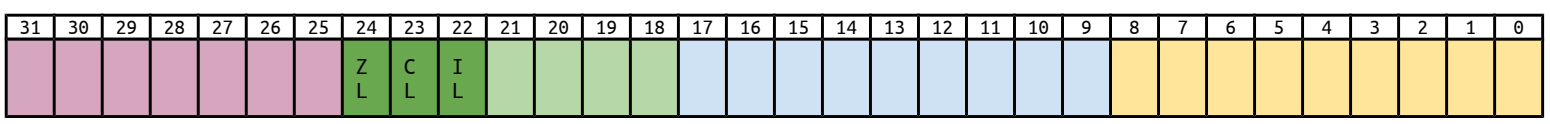
iiiiiii	Instruction OpCode
Z	If set (1) by WZ (with zero), then the Z (zero flag) will be updated by this instruction
C	If set (1) by WC (with carry), then the C (cary flag) will be updated by this instruction
I	If set (1) by #S, then the SSSSSSSSS will be used as an immediate value instead of register address
L	If set (1) by #D, then the DDDDDDDDD will be used as an immediate value instead of register address *Note: "L", when available, repurposes Z, C or I bit*
n/nn/nnn	Word/Byte/Nibble select (Repurposes lower opcode bit and Z C bits)
CCCC	Conditional execution code
DDDDDDDDD	Destination address, or immediate value when L=1
SSSSSSSSS	Source address, or immediate value when I=1

OpCode Field:



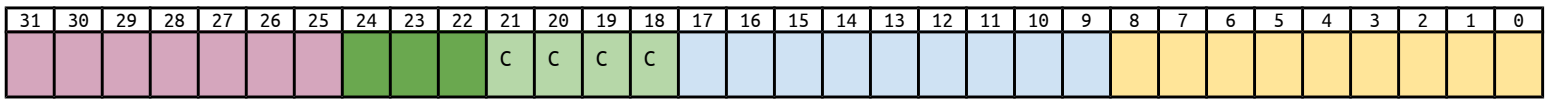
These field bits[31:25] represent the 7-bit instruction opcode.
Note: Some extended opcode instructions also utilise a mix of the Z, C, I and SSSSSSSSS bits.

ZCI (and L) Fields:



These field bits[24:22] represent the Z, C and I 1-bit fields for WZ (Zero flag), WC (Carry flag) and Immediate (#) S field modifier.
Some instructions my alternately utilise any of these bits as an Immediate (#) D field modifier, designated by "L".
Those instructions capable of changing the "Zero Flag" are designated with a "Z" in bit[24].
Those instructions capable of changing the "Carry Flag" are designated with a "C" in bit[23].
Those instructions capable of utilising a 9-bit "Immediate Source Field" in bits[8:0] are designated with an "I" in bit[22].
Those instructions capable of utilising a 9-bit "Immediate Destination Field" in bits[17:9] are designated with an "L" in one of bit[24], bit[23] or bit[22].
Note: Some extended opcode instructions repurpose a mix of the Z, C and I bits, making those bits unavailable for those instructions.

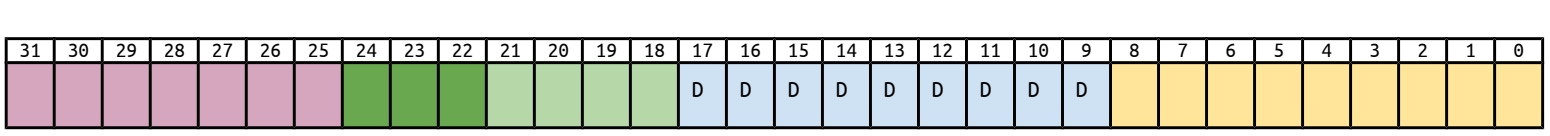
CCCC Field:



These field bits[21:18] represent the instructions' 4-bit conditional execution bits. This 4-bit code, together with the current value of the Z (zero flag) and C (carry flag), determines whether this instruction will execute or be transformed into an effective "NOP" (no-operation) instruction.

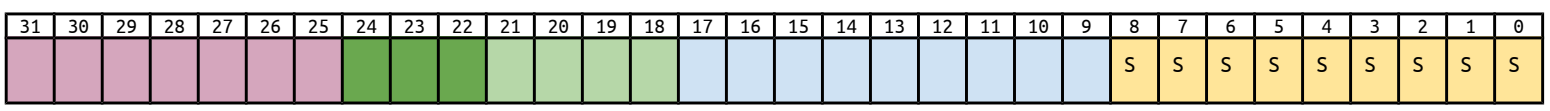
CONDITION	INSTRUCTION EXECUTES	CODE	SYNONYMS
IF_ALWAYS	always	1111	
IF_NEVER	never	0000	
IF_E	if equal (Z = 1)	1010	IF_Z
IF_NE	if not equal (Z = 0)	0101	IF_NZ
IF_A	if above (!C & !Z = 1)	0001	IF_NC_AND_NZ –and– IF_NZ_AND_NC
IF_B	if below (C = 1)	1100	IF_C
IF_AE	if above or equal (C = 0)	0011	IF_NC
IF_BE	if below or equal (C | Z = 1)	1110	IF_C_OR_Z –and– IF_Z_OR_C
IF_C	if C set	1100	IF_B
IF_NC	if C clear	0011	IF_AE
IF_Z	if Z set	1010	IF_E
IF_NZ	if Z clear	0101	IF_NE
IF_C_EQ_Z	if C equal to Z	1001	IF_Z_EQ_C
IF_C_NE_Z	if C not equal to Z	0110	IF_Z_NE_C
IF_C_AND_Z	if C set and Z set	1000	IF_Z_AND_C
IF_C_AND_NZ	if C set and Z clear	0100	IF_NZ_AND_C
IF_NC_AND_Z	if C clear and Z set	0010	IF_Z_AND_NC
IF_NC_AND_NZ	if C clear and Z clear	0001	IF_A –and– IF_NZ_AND_NC
IF_C_OR_Z	if C set or Z set	1110	IF_BE –and– IF_Z_OR_C
IF_C_OR_NZ	if C set or Z clear	1101	IF_NZ_OR_C
IF_NC_OR_Z	if C clear or Z set	1011	IF_Z_OR_NC
IF_NC_OR_NZ	if C clear or Z clear	0111	IF_NZ_OR_NC
IF_Z_EQ_C	if Z equal to C	1001	IF_C_EQ_Z
IF_Z_NE_C	if Z not equal to C	0110	IF_C_NE_Z
IF_Z_AND_C	if Z set and C set	1000	IF_C_AND_Z
IF_Z_AND_NC	if Z set and C clear	0010	IF_NC_AND_Z
IF_NZ_AND_C	if Z clear and C set	0100	IF_C_AND_NZ
IF_NZ_AND_NC	if Z clear and C clear	0001	IF_A –and– IF_NC_AND_NZ
IF_Z_OR_C	if Z set or C set	1110	IF_BE –and– IF_C_OR_Z
IF_Z_OR_NC	if Z set or C clear	1011	IF_NC_OR_Z
IF_NZ_OR_C	if Z clear or C set	1101	IF_C_OR_NZ
IF_NZ_OR_NC	if Z clear or C clear	0111	IF_NC_OR_NZ

DDDDDDDDD Field:



These field bits[17:9] represent the 9-bit register address to be utilised as the destination value.
Some instructions permit the use of the "D" bits[17:9] to be utilised as an immediate (#) 9-bit value. Those instructions will utilise an "L" bit modifier in one of the bit[24:22] positions (by repurposing the Z, C or I bit).

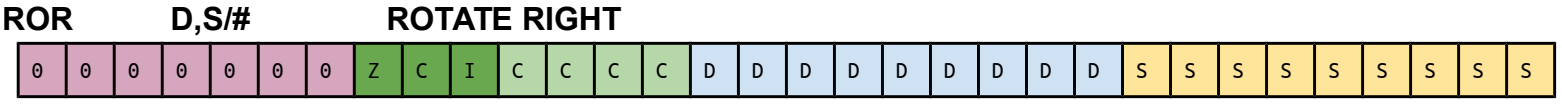
SSSSSSSSS Field:



These field bits[8:0] represent the 9-bit register address to be utilised as the source value.
Most instructions permit the use of the "S" bits[8:0] to be utilised as an immediate (#) 9-bit value. Those instructions will utilise the "I" bit modifier in bit[22].
Note: Some extended instructions repurpose the "S" bits[8:0] to extend the opcode space.

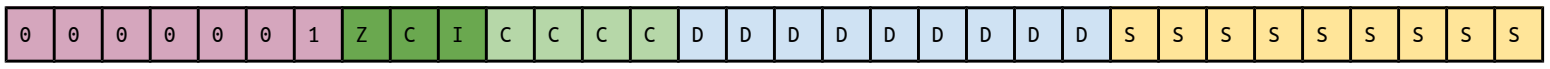
ROTATE INSTRUCTIONS

ROR D,S/# ROTATE RIGHT



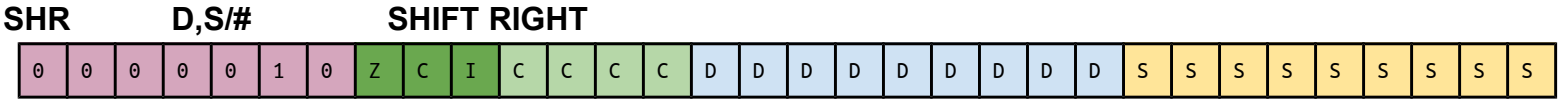
Rotate the contents of the destination D right by S bits with the carry out feeding back into bit 31.
If the WC effect is specified, the C flag is set equal to the last bit shifted out

ROL D,S/# ROTATE LEFT



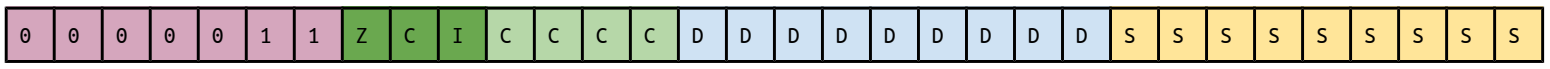
Rotate the contents of the destination D left by S bits with the carry out feeding back into bit 0.
If the WC effect is specified, the C flag is set equal to the last bit shifted out

SHR D,S/# SHIFT RIGHT



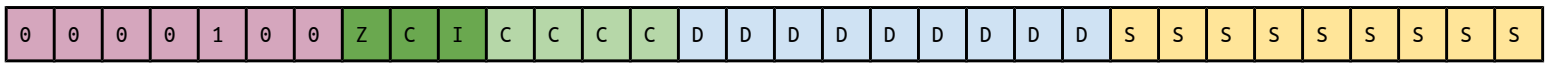
Shift the contents of the destination D right by S bits with the MSBs filling with zeros.
If the WC effect is specified, the C flag is set equal to the last bit shifted out

SHL D,S/# SHIFT RIGHT



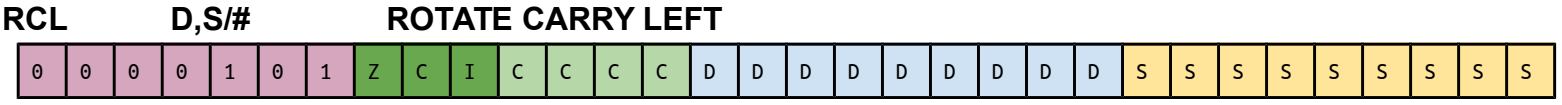
Shift the contents of the destination D left by S bits with the LSBs filling with zeros.
If the WC effect is specified, the C flag is set equal to the last bit shifted out

RCR D,S/# ROTATE CARRY RIGHT



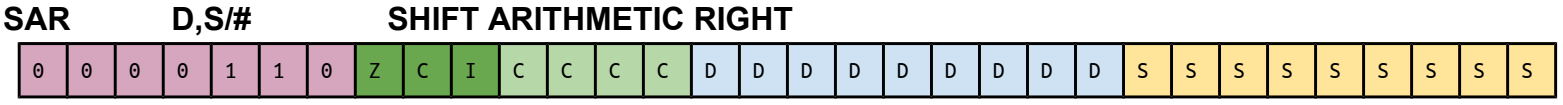
Rotate the contents of the destination D right by S bits using the C flag's original value for each of the MSBs affected
If the WC effect is specified, the C flag is set equal to the last bit shifted out

RCL D,S/# ROTATE CARRY LEFT



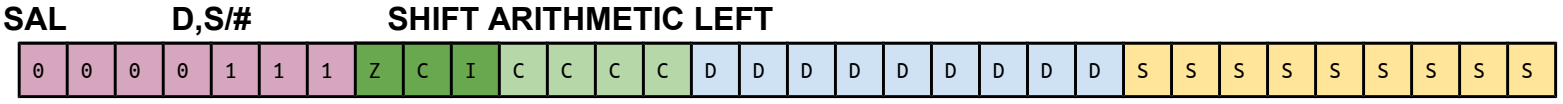
Rotate the contents of the destination D left by S bits using the C flag's original value for each of the LSBs affected
If the WC effect is specified, the C flag is set equal to the last bit shifted out

SAR D,S/# SHIFT ARITHMETIC RIGHT



Shift the contents of the destination D arithmetically right by S bits extending the MSB along the affected bits to preserve the sign.
If the WC effect is specified, the C flag is set equal to the last bit shifted out

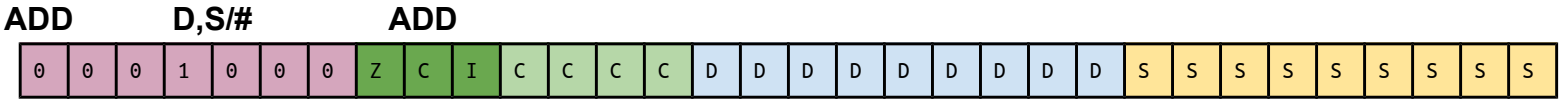
SAL D,S/# SHIFT ARITHMETIC LEFT



Shift the contents of the destination D arithmetically left by S bits while preserving the sign
If the WC effect is specified, the C flag is set equal to the last bit shifted out ??

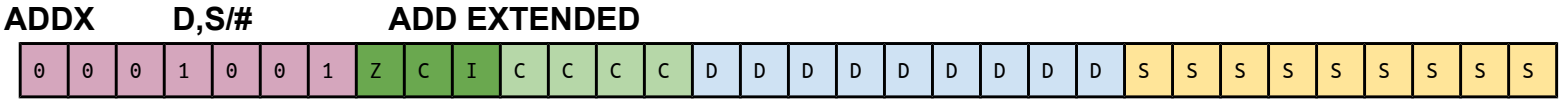
ADD INSTRUCTIONS

ADD D,S/# ADD



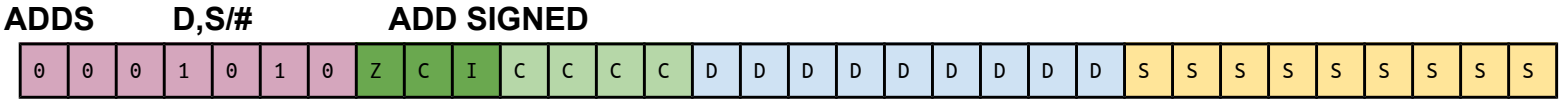
Add the unsigned contents of the source S to the destination D
If the WC effect is specified, the C flag is set if there was an overflow

ADDX D,S/# ADD EXTENDED



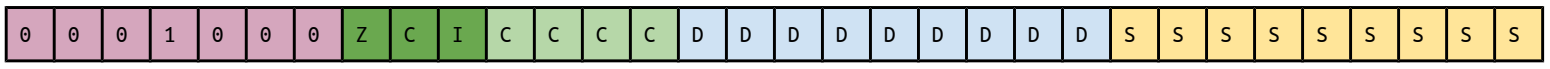
Add the unsigned contents of the source S along with the carry to the destination D
If the WC effect is specified, the C flag is set if there was an overflow

ADDS D,S/# ADD SIGNED



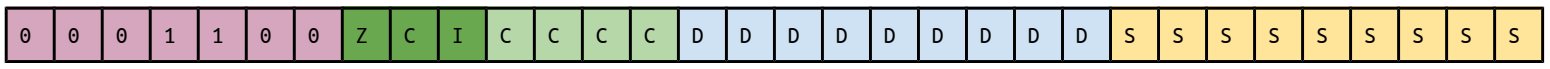
Add the signed contents of the source S to the destination D
If the WC effect is specified, the C flag is set if there was a signed overflow

ADDSX D,S/# ADD SIGNED EXTENDED



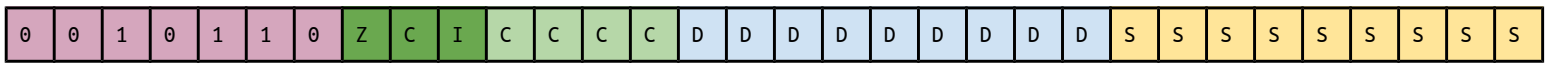
Add the signed contents of the source S along with the carry to the destination D
If the WC effect is specified, the C flag is set if there was a signed overflow

SUB D,S/# SUBTRACT



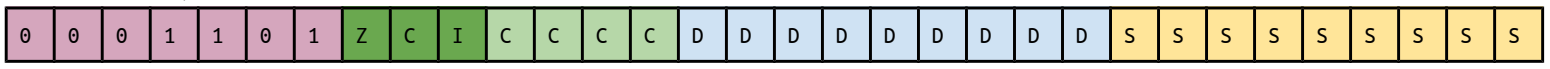
Subtract the unsigned contents of the source S from the destination D
If the WC effect is specified, the C flag is set if there was an overflow

SUBR D,S/# SUBTRACT REVERSE



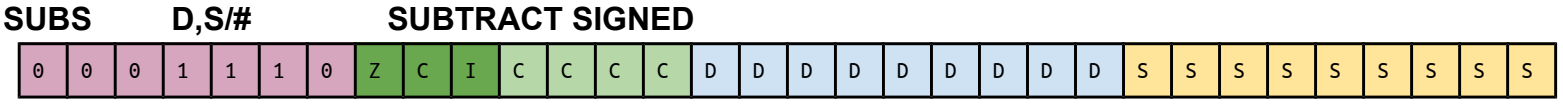
Subtract the destination from the source with the result into the destination (D = S - D)

SUBX D,S/# SUBTRACT EXTENDED



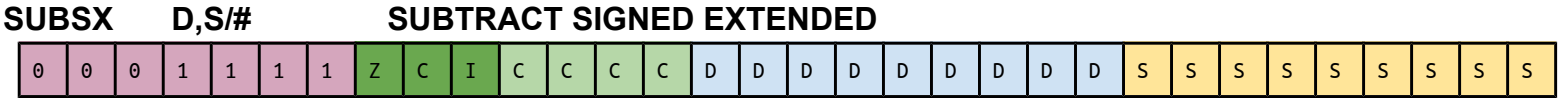
Subtract the unsigned contents of the source S along with the borrow (C flag) from the destination D
D = D - (S + C)
If the WC effect is specified, the C flag is set if there was an overflow

SUBS D,S/# SUBTRACT SIGNED



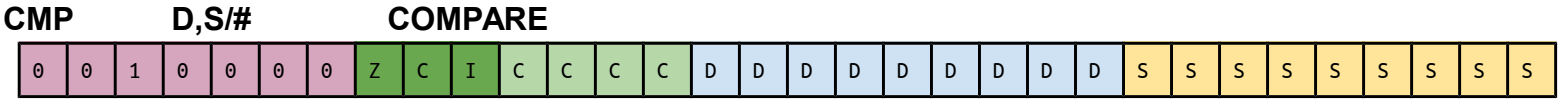
Subtract the signed contents of the source S from the destination D and update the sign
D = D - S
If the WC effect is specified, the C flag is set if there was a signed overflow

SUBSX D,S/# SUBTRACT SIGNED EXTENDED



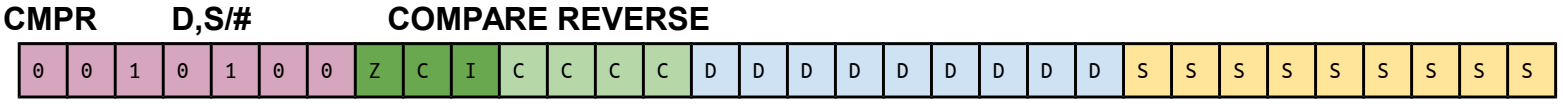
Subtract the signed contents of the source S along with the borrow (C flag) from the destination D
D = D - (S + C)
If the WC effect is specified, the C flag is set if there was a signed overflow

CMP D,S/# COMPARE



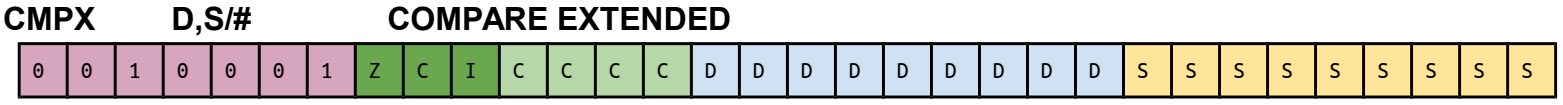
Compare…

CMPR D,S/# COMPARE REVERSE



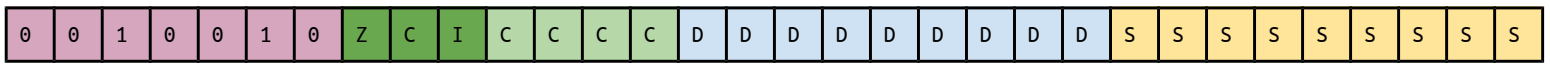
Compare the source with the destination in reverse (S - D)

CMPX D,S/# COMPARE EXTENDED



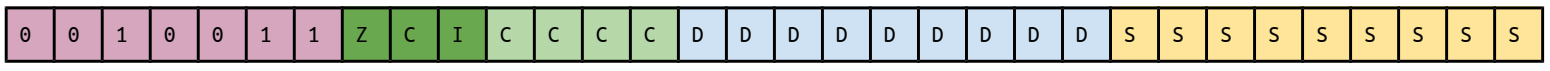
Compare…

CMPS D,S/# COMPARE SIGNED



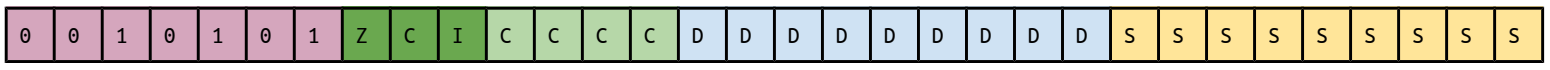
Compare…

CMPSX D,S/# COMPARE SIGNED EXTENDED



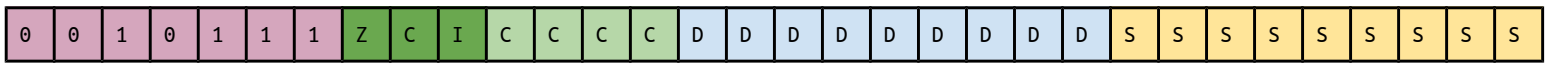
Compare…

CMPM D,S/# COMPARE M???



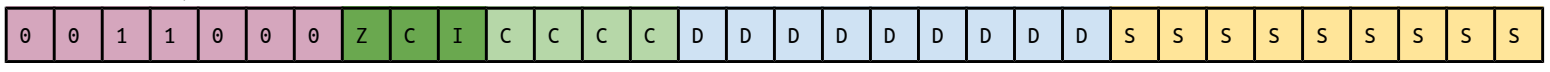
Compare…

CMPSUB D,S/# COMPARE SUBTRACT



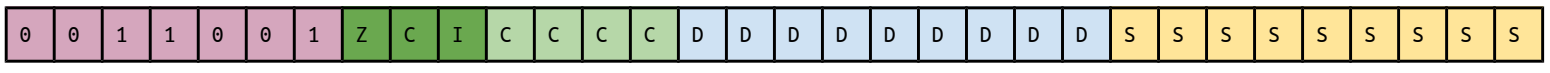
Compare two unsigned values and subtract the second if it is lesser or equal

MIN D,S/# MIN



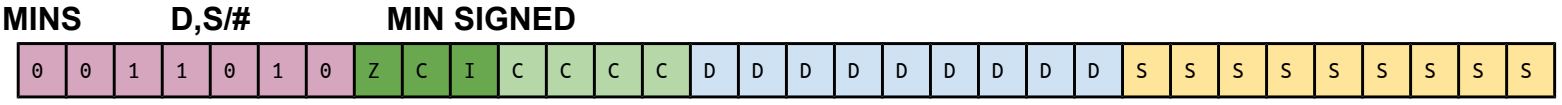
Limit minimum of unsigned destination D to source S by storing the greater in D.
If the WC effect is specified, the C flag is set (1) if the unsigned value D is less than the unsigned value S.

MAX D,S/# MAX



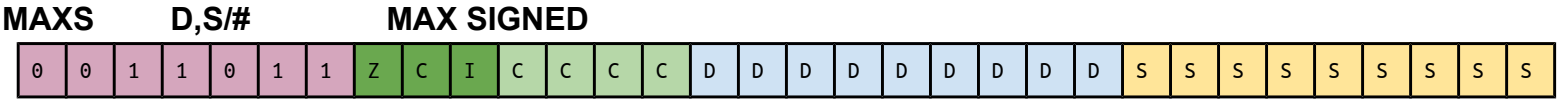
Store the less of the two values S and D in the destination D.
If the WC effect is specified, the C flag is set (1) if the unsigned value D is less than the unsigned value S

MINS D,S/# MIN SIGNED



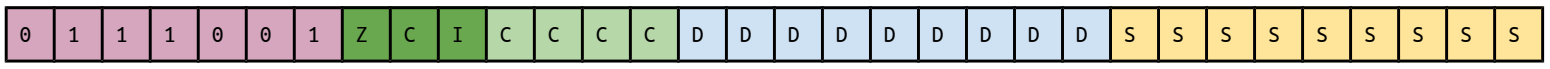
Limit minimum of signed destination D to source S by storing the greater in D.
If the WC effect is specified, the C flag is set (1) if the signed value D is less than the signed value S.

MAXS D,S/# MAX SIGNED



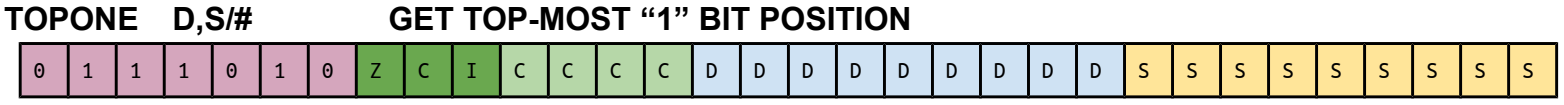
Store the lesser of the two signed values S and D in the Destination D.e
If the WC effect is specified, the C flag is set (1) if the signed value D is less than the signed value S.

DECOD D,S/# DECODE TO SETUP A BIT MASK



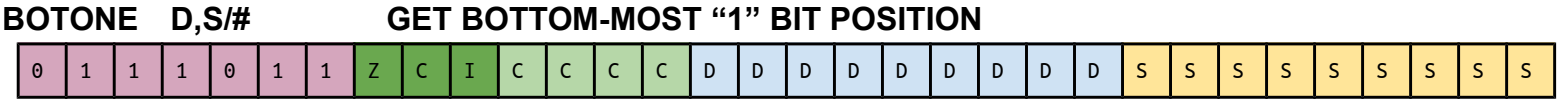
Decode lower 5 bits of S into a single bit mask.
S represents a bit number from 0 to 31. Decode sets all bits of the destination except the specified bit in S which is set to “1”. It is typically used to setup a bit mask.
If the WC effect is specified, the C flag is set (1) if
Note: BOTONE D,S can be used to perform the reverse of this instruction.

TOPONE D,S/# GET TOP-MOST “1” BIT POSITION



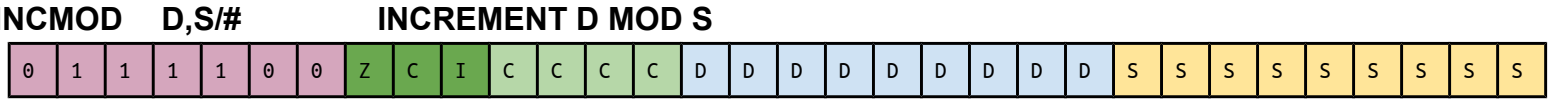
Get the top-most “1” bit position in D.
If the WC effect is specified, the C flag is set (1) if a “1” bit is located.

BOTONE D,S/# GET BOTTOM-MOST “1” BIT POSITION



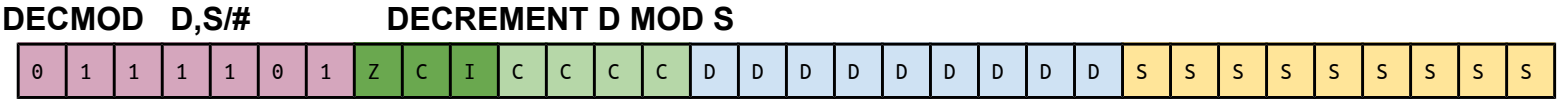
Get the bottom-most “1” bit position in D.
If the WC effect is specified, the C flag is set (1) if a “1” bit is located.

INCMOD D,S/# INCREMENT D MOD S



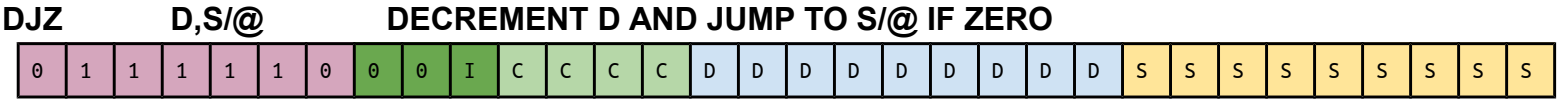
If D=S then D=0, else increment D (D++).
WARNING: If D>S it will not set D=0 because an equality is performed rather than a comparison with S.
If the WC effect is specified, the C flag is set (1) if

DECMOD D,S/# DECREMENT D MOD S



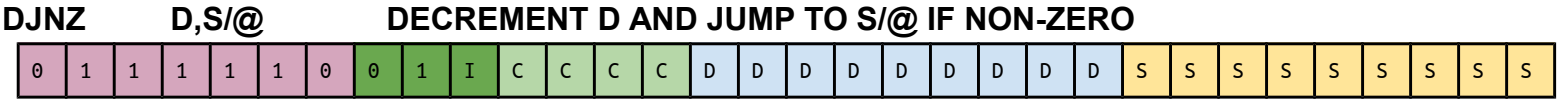
If D=0 then D=S, else decrement D (D--)
WARNING: If D>S it will not set D=S because an equality is performed rather than a comparison with S.
If the WC effect is specified, the C flag is set (1) if

DJZ D,S/@ DECREMENT D AND JUMP TO S/@ IF ZERO



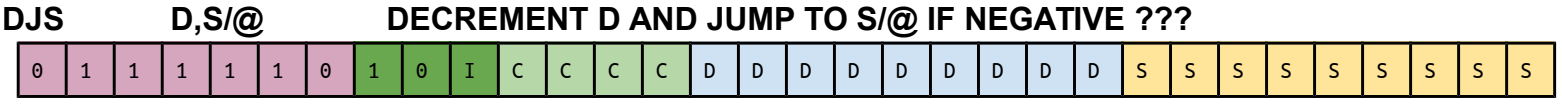
Decrement D and jump to S/@ if the result is zero.
Note: WZ and WC are not valid for this instruction.

DJNZ D,S/@ DECREMENT D AND JUMP TO S/@ IF NON-ZERO



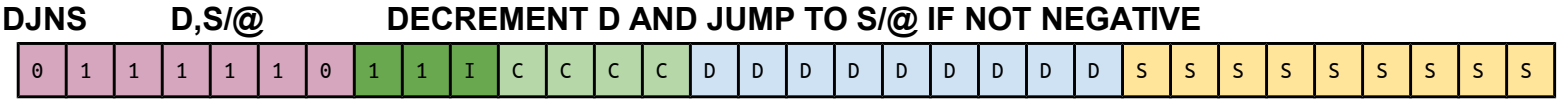
Decrement D and jump to S/@ if the result is non-zero.
Note: WZ and WC are not valid for this instruction.

DJS D,S/@ DECREMENT D AND JUMP TO S/@ IF NEGATIVE ???



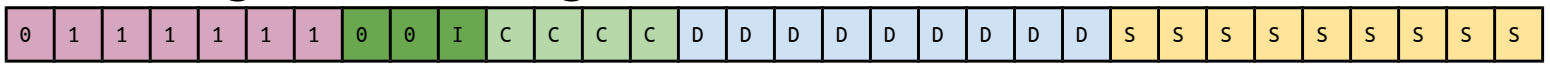
Decrement D and jump to S/@ if the result is negative.
Note: WZ and WC are not valid for this instruction.

DJNS D,S/@ DECREMENT D AND JUMP TO S/@ IF NOT NEGATIVE



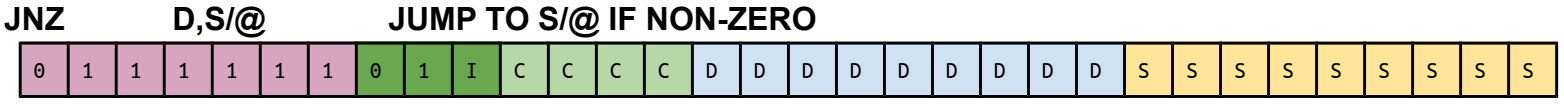
Decrement D and jump to S/@ if the result is not negative.
Note: WZ and WC are not valid for this instruction.

JZ D,S/@ JUMP TO S/@ IF ZERO



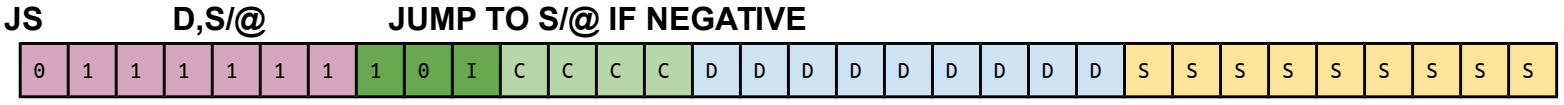
Jump to S/@ if D is zero.
Note: WZ and WC are not valid for this instruction.

JNZ D,S/@ JUMP TO S/@ IF NON-ZERO



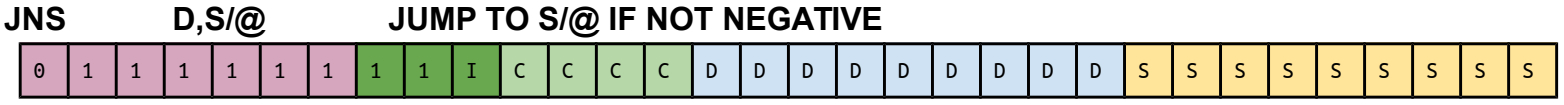
Jump to S/@ if D is zero.
Note: WZ and WC are not valid for this instruction.

JS D,S/@ JUMP TO S/@ IF NEGATIVE



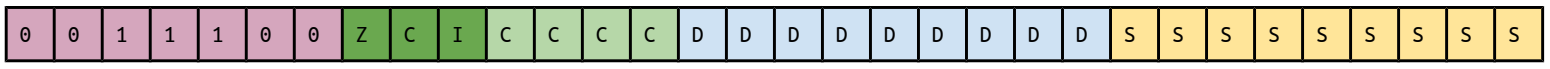
Jump to S/@ if D is zero.
Note: WZ and WC are not valid for this instruction.

JNS D,S/@ JUMP TO S/@ IF NOT NEGATIVE



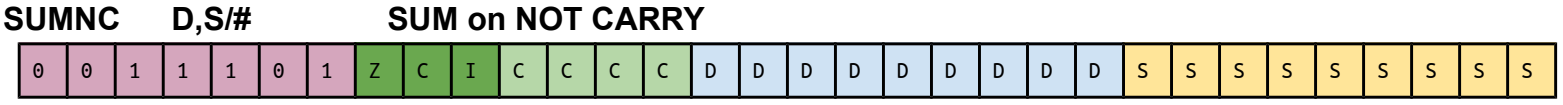
Jump to S/@ if D is not negative.
Note: WZ and WC are not valid for this instruction.

SUMC D,S/# SUM on CARRY



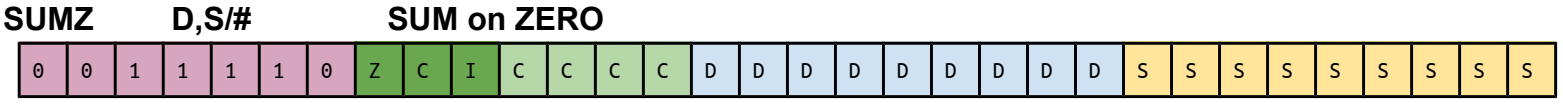
Sum a signed value with another whose sign is inverted depending on C

SUMNC D,S/# SUM on NOT CARRY



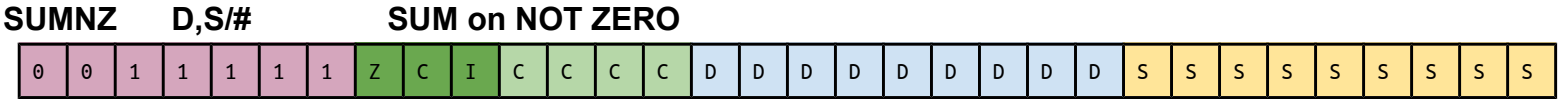
Sum a signed value with another whose sign is inverted depending on !C

SUMZ D,S/# SUM on ZERO



Sum a signed value with another whose sign is inverted depending on Z

SUMNZ D,S/# SUM on NOT ZERO

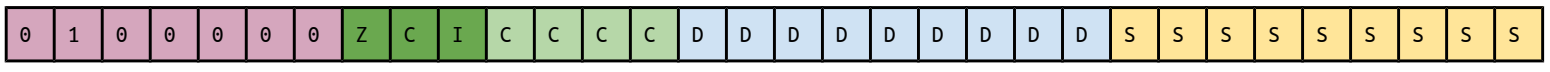


Sum a signed value with another whose sign is inverted depending on !Z

LOGICAL INSTRUCTIONS

ISOB D,S/# ISOLATE BIT

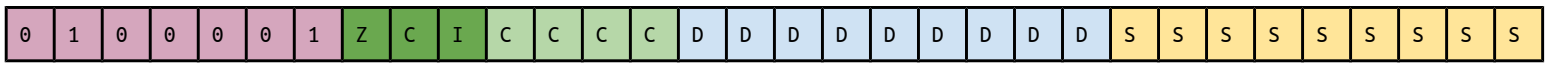
ZCMS



Lower 5 bits at S specify the bit while the upper 4 bits of S define the sub instruction??
If the WC effect is specified, the C flag is set (1) if selected bit is set()

NOTB D,S/# NOT BIT

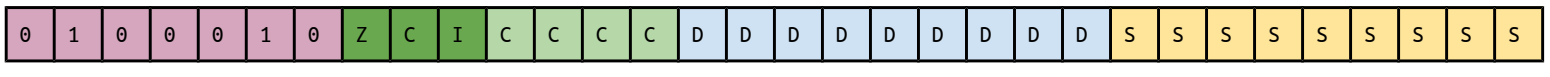
ZCMS



Invert (NOT) the bit in D specified at S ???
If the WC effect is specified, the C flag is set (1) if ???

CLRB D,S/# CLEAR BIT

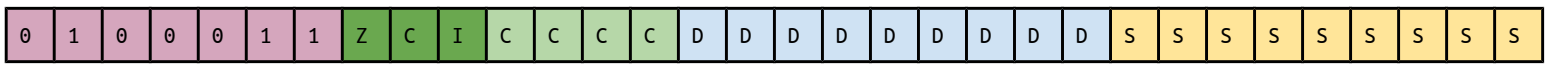
ZCMS



Clear the bit in D specified at S ???
If the WC effect is specified, the C flag is set (1) if ???

SETB D,S/# SET BIT

ZCMS

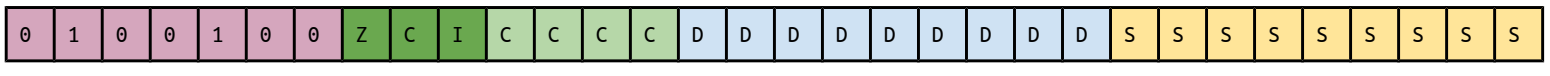


Set the bit in D specified at S ???
If the WC effect is specified, the C flag is set (1) if ???

```
ZCMS            0100011 ZCI CCCC DDDDDDDDD SSSSSSSSS            SETB    D,S/#           log
```

SETBC D,S/# SET BIT with C

ZCMS

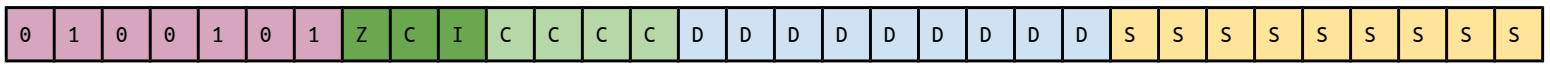


Set the bit in D specified at S if the C flag is set ???
If the WC effect is specified, the C flag is set (1) if ???

```
ZCMS            0100100 ZCI CCCC DDDDDDDDD SSSSSSSSS            SETBC   D,S/#           log
```

SETBNC D,S/# SET BIT with NOT C

ZCMS

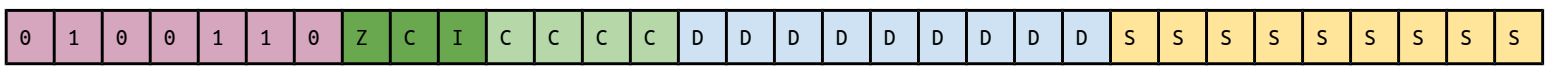


Set the bit in D specified at S if the C flag is clear ???
If the WC effect is specified, the C flag is set (1) if ???

```
ZCMS            0100101 ZCI CCCC DDDDDDDDD SSSSSSSSS            SETBNC  D,S/#           log
```

SETBZ D,S/# SET BIT with Z

ZCMS

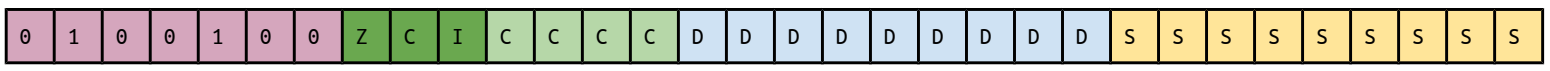


Set the bit in D specified at S if the Z flag is set ???
If the WC effect is specified, the C flag is set (1) if ???

```
ZCMS            0100110 ZCI CCCC DDDDDDDDD SSSSSSSSS            SETBZ   D,S/#           log
```

SETBNZ D,S/# SET BIT with NOT Z

ZCMS

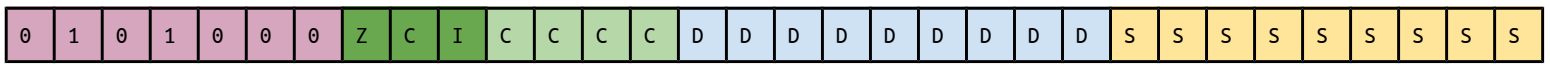


Set the bit in D specified at S if the Z flag is clear ???
If the WC effect is specified, the C flag is set (1) if ???

```
ZCMS            0100111 ZCI CCCC DDDDDDDDD SSSSSSSSS            SETBNZ  D,S/#           log
```

ANDN D,S/# BITWISE D AND NOT S

ZCMS

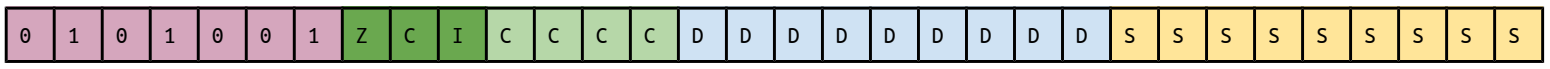


Perform a bitwise AND of the inverted value (bitwise NOT) in the source S with the value in the destination D.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.

If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.
See the TESTN instruction for a no-result (does not write result) version.

AND D,S/# BITWISE AND S WITH D

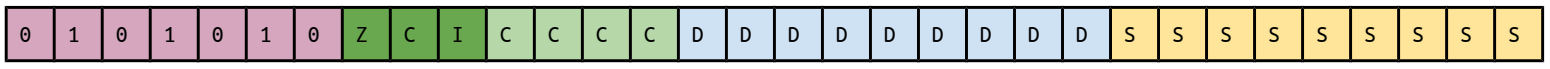
ZCMS



Perform a bitwise AND of the value in the source S with the value in the destination D.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.
See the TEST instruction for a no-result (does not write result) version.

OR D,S/# BITWISE OR S WITH D

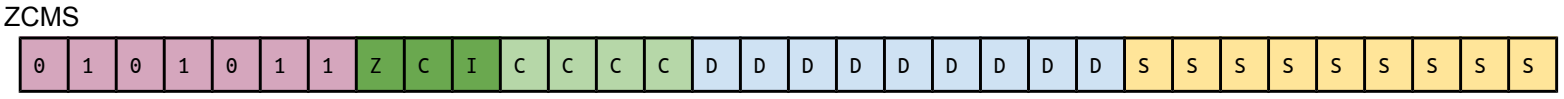
ZCMS



Bitwise OR the value in the source S into the value of the destination D.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.
See the ANYB instruction for a no-result (does not write result) version.

XOR D,S/# BITWISE EXCLUSIVE OR S WITH D

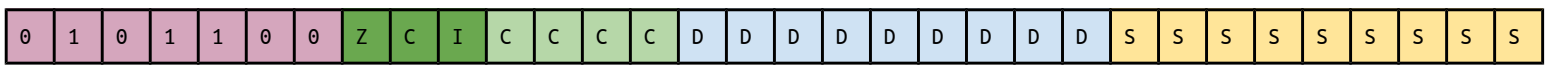
ZCMS



Bitwise Exclusive OR the value in the source S into the value of the destination D.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.

MUXC D,S/# D BITS, INDICATED BY MASK S, ARE SET TO THE STATE OF C

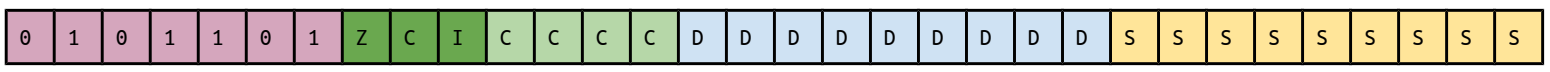
ZCMS



MUXC sets each bit of the value in the destination D, which corresponds to the Mask's high (1) bits in the source S, to the state of C.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.

MUXNC D,S/# D BITS, INDICATED BY MASK S, ARE SET TO THE STATE OF NOT C

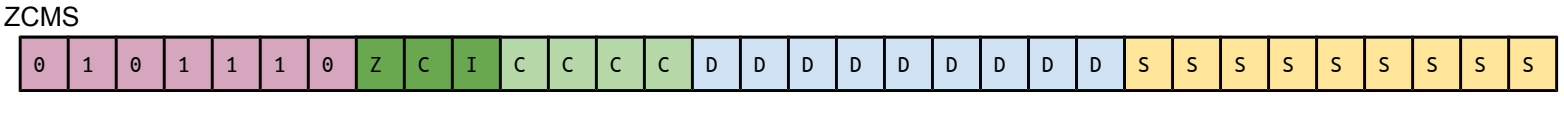
ZCMS



MUXNC sets each bit of the value in the destination D, which corresponds to the Mask's high (1) bits in the source S, to the inverted state of C.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.

MUXZ D,S/# D BITS, INDICATED BY MASK S, ARE SET TO THE STATE OF Z

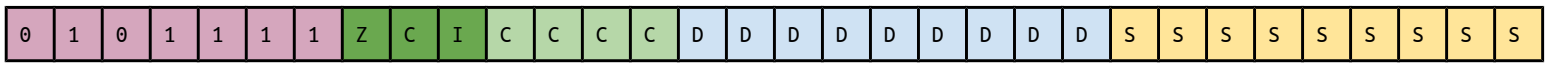
ZCMS



MUXZ sets each bit of the value in the destination D, which corresponds to the Mask's high (1) bits in the source S, to the state of Z.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.

MUXNZ D,S/# D BITS, INDICATED BY MASK S, ARE SET TO THE STATE OF NOT Z

ZCMS

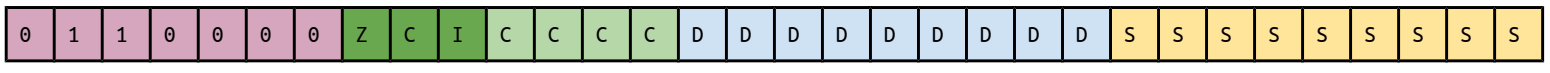


MUXNZ sets each bit of the value in the destination D, which corresponds to the Mask's high (1) bits in the source S, to the inverted state of Z.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.

INC INSTRUCTIONS

MOV D,S/# Move (copy) the value in the source S to the destination D

ZCMS

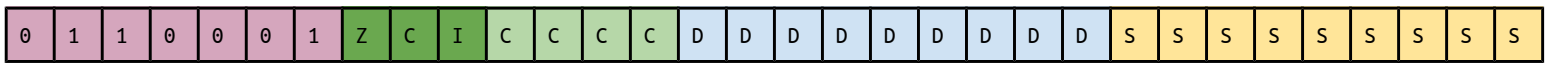


Move (copy) the value in the source S to the destination D.

If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set to the value of bit D[31] of the result.

NOT D,S/# Move (copy) the bitwise inversion of the value in the source S to the destination D

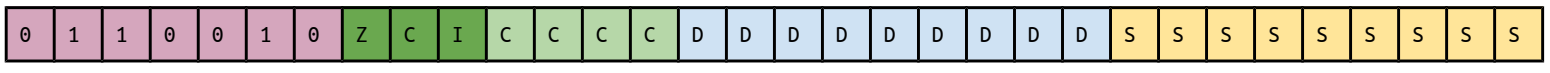
ZCMS



Move (copy) the bitwise inversion of the value in the source S to the destination D.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set to the value of bit D[31] of the result.

ABS D,S/# Move (copy) the Absolute value in the source S to the destination D

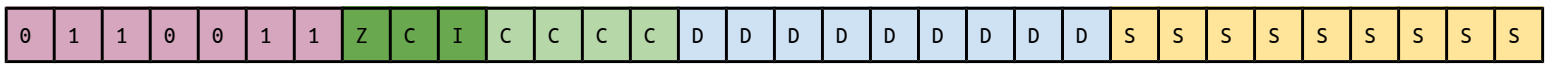
ZCMS



Move (copy) the absolute value in the source S to the destination D.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the value in the source S is negative, or cleared (0) if S is positive.

NEG D,S/# GET THE NEGATIVE OF A NUMBER

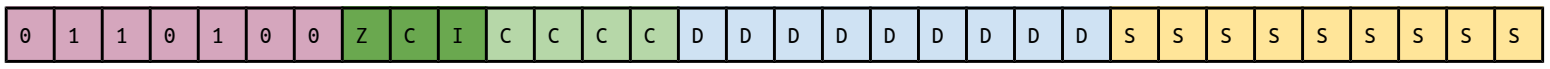
ZCMS



Set the destination D to the negative value in the source S.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the value in the source S is negative, or cleared (0) if it is positive.

NEGC D,S/# GET A VALUE, OR ITS ADDITIVE INVERSE, BASED ON C

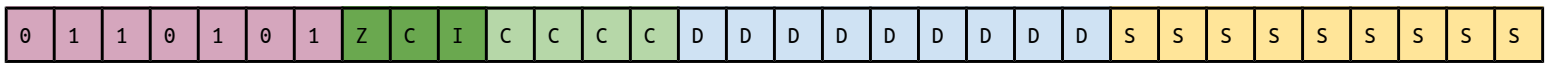
ZCMS



If C=0 copy the value in the source S, else copy the additive inverse of the value in the source S, and store in the destination D.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the value in the source S is negative, or cleared (0) if it is positive.

NEGNC D,S/# GET A VALUE, OR ITS ADDITIVE INVERSE, BASED ON !C

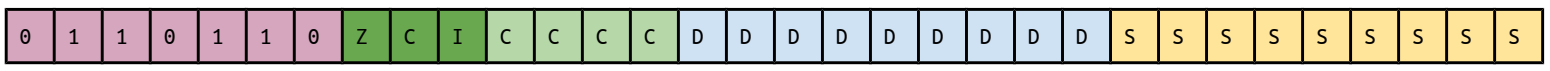
ZCMS



If C=1 copy the value in the source S, else copy the additive inverse of the value in the source S, and store in the destination D.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the value in the source S is negative, or cleared (0) if it is positive.

NEGZ D,S/# GET A VALUE, OR ITS ADDITIVE INVERSE, BASED ON Z

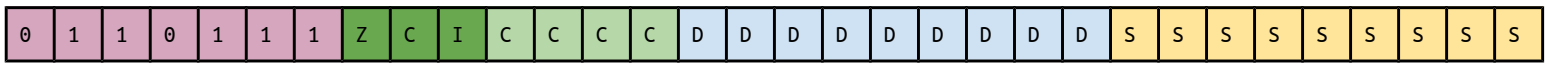
ZCMS



If Z=0 copy the value in the source S, else copy the additive inverse of the value in the source S, and store in the destination D.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the value in the source S is negative, or cleared (0) if it is positive.

NEGNZ D,S/# GET A VALUE, OR ITS ADDITIVE INVERSE, BASED ON !Z

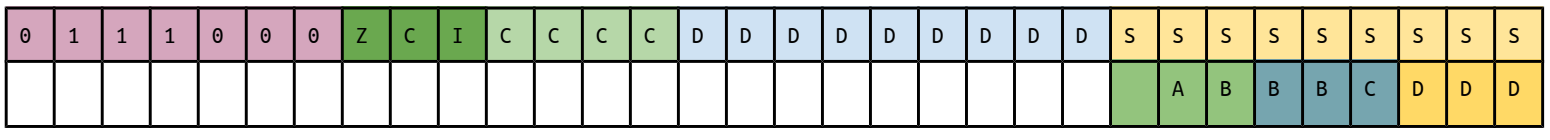
ZCMS



If Z=1 copy the value in the source S, else copy the additive inverse of the value in the source S, and store in the destination D.
If the WZ effect is specified, the Z flag is set (1) if the result equals zero.
If the WC effect is specified, the C flag is set (1) if the value in the source S is negative, or cleared (0) if it is positive.

ALTDS D,S/# SELECTIVELY ALTER D (read and write result addresses) & S address for the following instruction

ZCMS



ALTDS uses a D register for D/S field substitutions in the next instruction, while S/# modifies the D register's D and S fields and controls D/S substitution.

D - a register whose D/S fields may be substituted for the next instructions' D/S fields
S/# - an 8-bit code: %ABBBCDDD

%A: 0 = don't substitute next instructions' D field with current D register's D field.,

1= substitute next instructions' D field with current D register's D field

%BBB: 000 = leave the current D register's D field the same,
0xx = add 1/2/3 to D field,
1xx = subtract 1/2/3/4 from D field

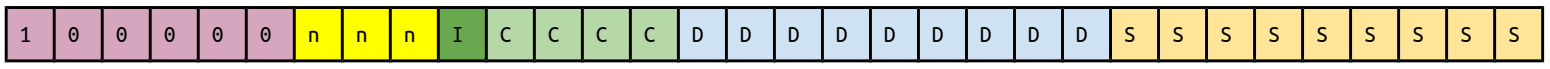
%C: 0 = don't substitute next instructions' S field with current D register's S field,
1 = substitute next instructions' S field with current D register's S field

%DDD: 000 = leave the current D register's S field the same,
0xx = add 1/2/3 to S field,
1xx = subtract 1/2/3/4 from S field

MUX INSTRUCTIONS

SETNIBn D,S/# SET NIBBLE

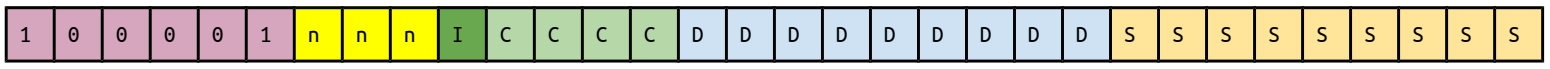
--MS



Set nibble n in D with nibble in S

GETNIBn D,S/# GET NIBBLE

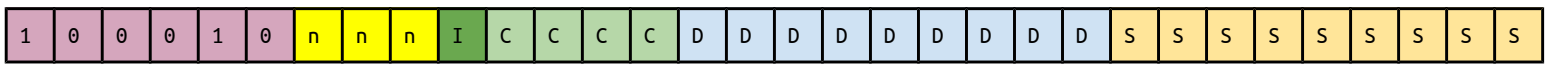
--WS



Get nibble n in S into D

ROLNIBn D,S/# ROTATE LEFT NIBBLE

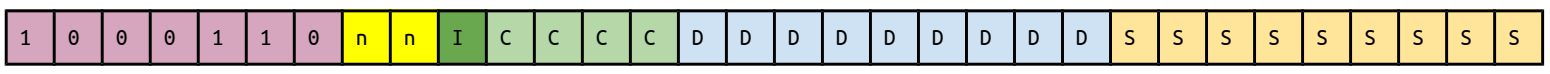
--MS



Rotate nibble in S left into D x n

SETBYTn D,S/# SET BYTE

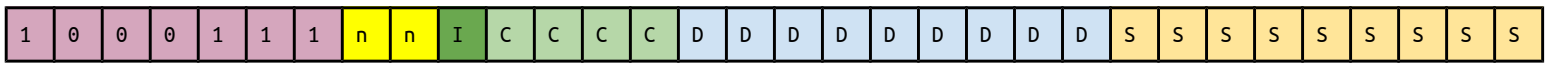
--MS



Set nth byte ???

GETBYTn D,S/# GET BYTE

--WS

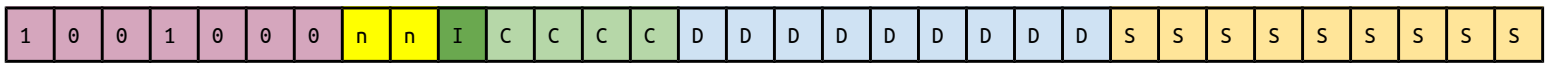


Get nth byte from source S into destination D ???

```
--WS            1000111 nnI CCCC DDDDDDDDD SSSSSSSSS            GETBYTn D,S/#          mux
```

ROLBYTn D,S/# ROTATE LEFT BYTE

--MS

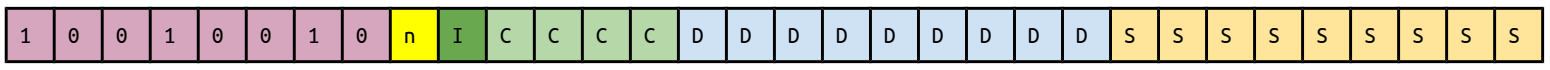


Rotate nth byte in D left by S bits ???

```
--MS            1001000 nnI CCCC DDDDDDDDD SSSSSSSSS            ROLBYTn D,S/#          mux
```

SETWRDn D,S/# SET WORD

--MS

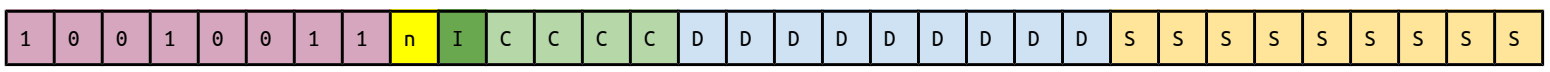


Set nth word in D with S ???

```
--MS            1001001 0nI CCCC DDDDDDDDD SSSSSSSSS            SETWRDn D,S/#          mux
```

GETWRDn D,S/# GET WORD

--WS

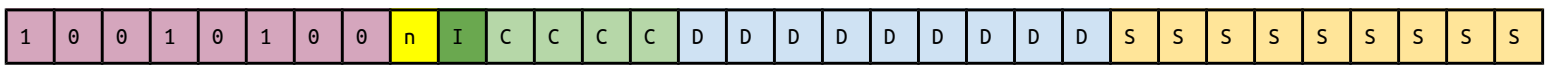


Get nth word in S to D ???

```
--WS            1001001 1nI CCCC DDDDDDDDD SSSSSSSSS            GETWRDn D,S/#          mux
```

ROLWRDn D,S/# ROTATE LEFT WORD

--MS

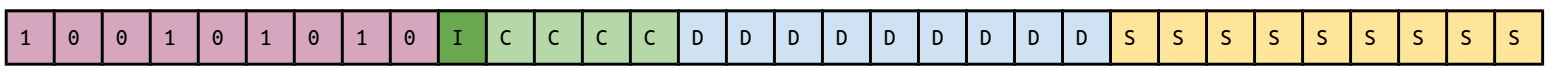


Rotate nth word in D by S ???

```
--MS            1001010 0nI CCCC DDDDDDDDD SSSSSSSSS            ROLWRDn D,S/#          mux
```

SETBYTS D,S/# SET BYTES

--WS

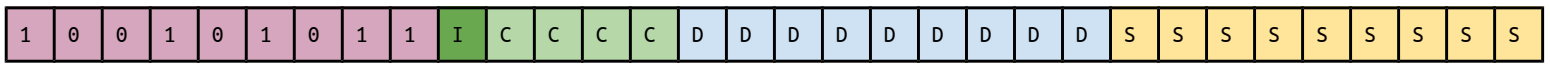


Set all bytes in D to S ???

```
--WS            1001010 10I CCCC DDDDDDDDD SSSSSSSSS            SETBYTS D,S/#          mux
```

MOVBYTS D,S/# MOVE BYTES

--MS

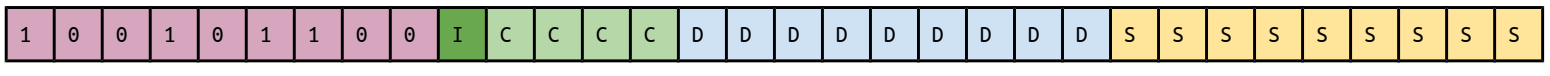


Move to all bytes in D with S ???

```
--MS            1001010 11I CCCC DDDDDDDDD SSSSSSSSS            MOVBYTS D,S/#          mux
```

SPLITB D,S/# SPLIT BYTE

--WS



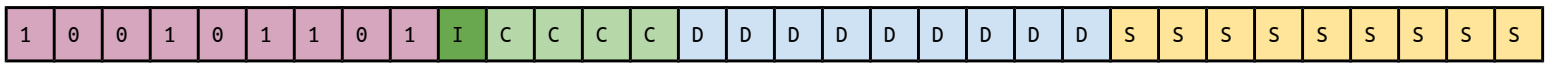
Split value in S where ???

the SPLITB/MERGEB pair can be used to do 3 and 4 dimensional z-order curve.

```
--WS            1001011 00I CCCC DDDDDDDDD SSSSSSSSS            SPLITB  D,S/#          mux
```

MERGEB D,S/# MERGE BYTE

--WS

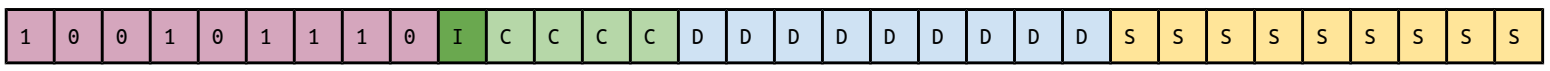


Merge bytes ???

```
--WS            1001011 01I CCCC DDDDDDDDD SSSSSSSSS            MERGEB  D,S/#          mux
```

SPLITW D,S/# SPLIT WORD

--WS



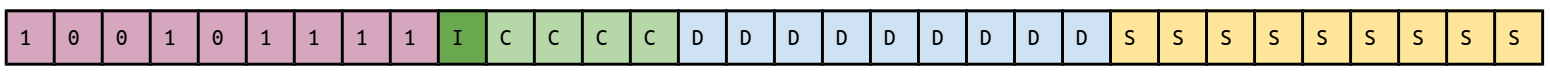
Split value in S where odd bits are copied to upper word and even bits to lower word of D

the SPLITW/MERGEW paiir is good for the normal 2 dimensional z-order curve

```
--WS            1001011 10I CCCC DDDDDDDDD SSSSSSSSS            SPLITW  D,S/#          mux
```

MERGEW D,S/# MERGE WORD

--WS

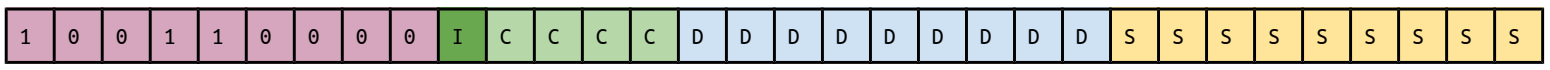


Merge upper word of S to odd bits of D and lower word of S to even bits of D ???

```
--WS            1001011 11I CCCC DDDDDDDDD SSSSSSSSS            MERGEW  D,S/#          mux
```

SETS D,S/# SET SOURCE

--MS

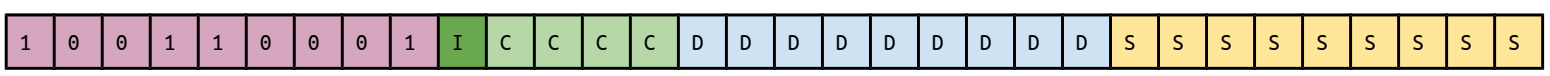


Set the source field of the instruction (b0..b8) specified at D to S

```
--MS            1001100 00I CCCC DDDDDDDDD SSSSSSSSS            SETS    D,S/#          mux
```

GETS D,S/# GET SOURCE

--WS

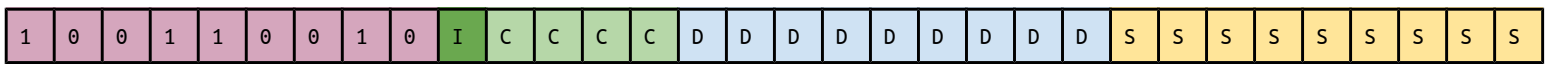


Get the source field of the instruction (b0..b8) specified at S to D

```
--WS            1001100 01I CCCC DDDDDDDDD SSSSSSSSS            GETS    D,S/#          mux
```

SETD D,S/# SET DESTINATION

--MS

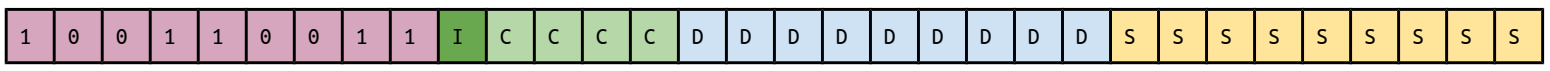


Set the destination field of the instruction (b9..b17) specified at D to S

```
--MS            1001100 10I CCCC DDDDDDDDD SSSSSSSSS            SETD    D,S/#          mux
```

GETD D,S/# GET DESTINATION

--WS

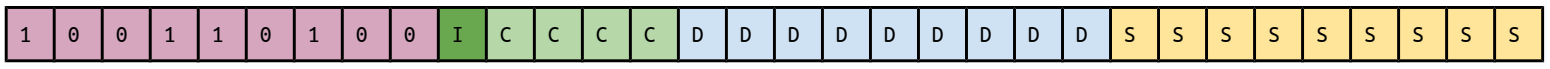


Get the destination field of the instruction (b9..b17) specified at S to D

```
--WS            1001100 11I CCCC DDDDDDDDD SSSSSSSSS            GETD    D,S/#          mux
```

SETDS D,S/# SET DESTINATION and SOURCE

--MS

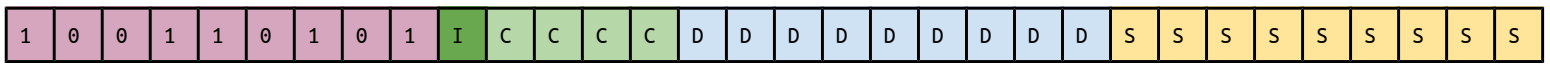


Set both the destination and source field of the instruction specified at D to S ???

```
--MS            1001101 00I CCCC DDDDDDDD SSSSSSSS            SETDS   D,S/#          mux
```

SETCOND D,S/# SET CONDITION

--MS

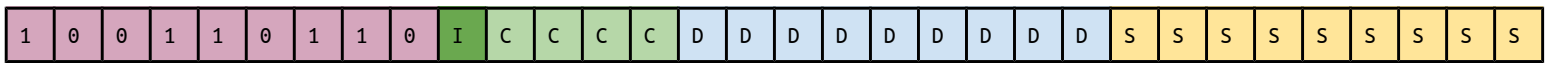


Sets the CCCC bits in D to the lower 4 bits in S

```
--MS            1001101 01I CCCC DDDDDDDD SSSSSSSS            SETCOND D,S/#          mux
```

SETI D,S/# SET I

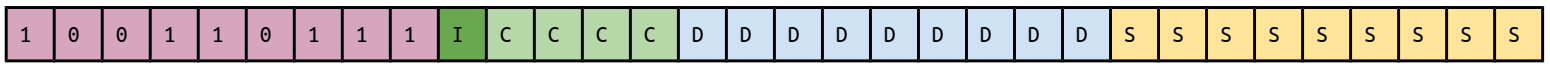
--MS



SETI D,S/# sets the bits in D[31:23] to the lower 9 bits in S (was MOVI)

REV D,S/# REVERSE BITS

--WS

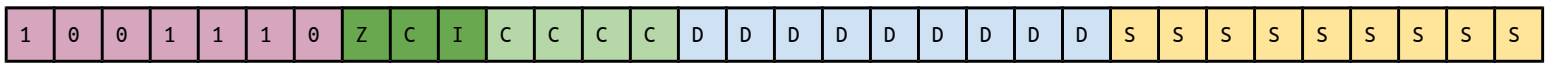


Reverses the lower (32 - Bits) of D’s LSB and clears the upper Bits of D’s MSBs

MULTIPLY INSTRUCTIONS

MUL D,S/# MULTIPLY

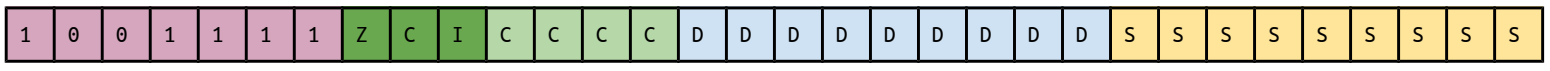
ZCMS



Multiply two unsigned 16-bit values
If the WC effect is specified, the C flag is set if ???

MULS D,S/# MULTIPLY SIGNED

ZCMS

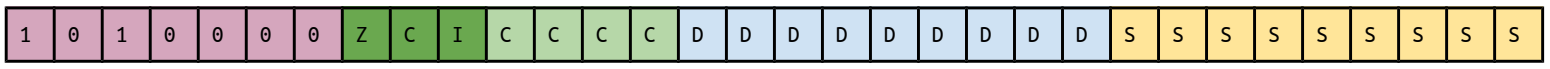


Multiply two signed 16-bit values
If the WC effect is specified, the C flag is set if ???

LOGICAL INSTRUCTIONS

TESTN D,S/# TEST NOT BITS

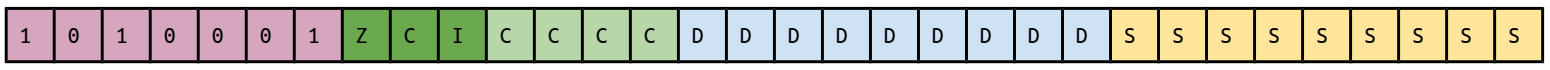
ZCRS



TEST the bits in D NOT specified in the bit mask specified in S, that is ANDN without write.
Set the Z flag if D AND !S = 0
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.

TEST D,S/# TEST BITS

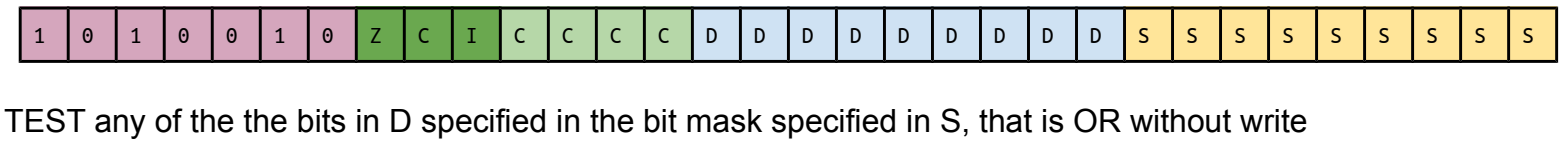
ZCRS



TEST the bits in D specified in the bit mask specified in S, that is AND without write
Set the Z flag if D AND S = 0
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.

ANYB D,S/# TEST ANY BITS

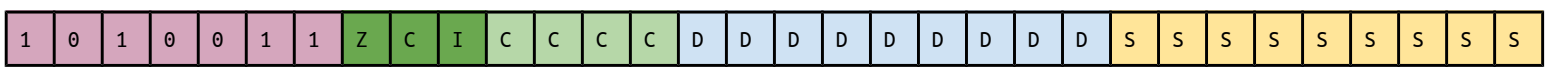
ZCRS



TEST any of the the bits in D specified in the bit mask specified in S, that is OR without write
Set the Z flag if
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.

TESTB D,S/# TEST A BIT

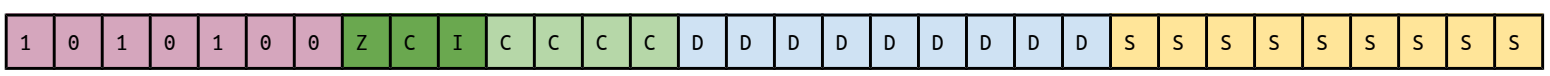
ZCRS



TEST a bit in D specified by the bit position (5 bits) at S (IOSB without write)
If the WZ effect is specifed set the Z flag to the state of the bit ?
If the WC effect is specified, the C flag is set (1) if the result contains an odd number of high (1) bits.

WAITCNT D,S/# WAIT CNT

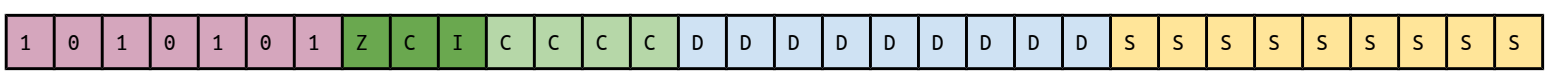
ZCMS



Wait for CNT

LINK D,S/@ LINK and JUMP

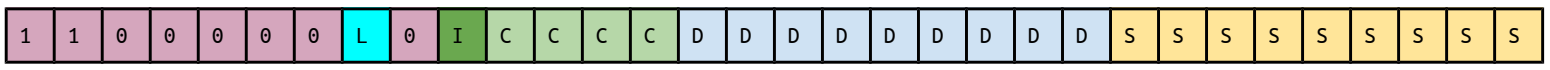
ZCMS



Write PC[18:0] and Z & C flags to D and jump to S

JP D/#,S/@ JUMP if PIN high

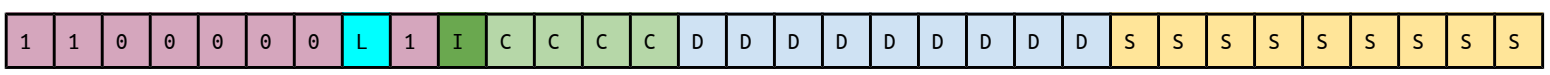
--LS



Jump if PIN is high. Uses INA registered at beginning of ALU cycle.

JNP D/#,S/@ JUMP if PIN low

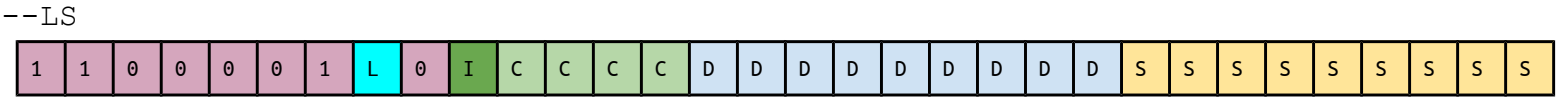
--LS



Jump if PIN is low. Uses INA registered at beginning of ALU cycle.

REP D/#,S/# REPEAT

--LS

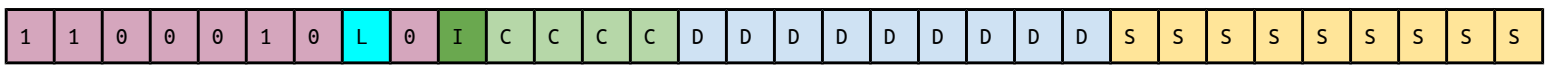


Begin repeat block of size D/# with S/# iterations

HUB INSTRUCTIONS

WRFAST D/#,S/PTRx WRITE FAST

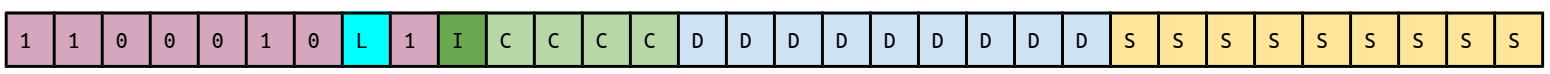
--LS



Write long to hub and bypass the FIFO when free (Blocking) ???

WRBYTE D/#,S/PTRx WRITE BYTE

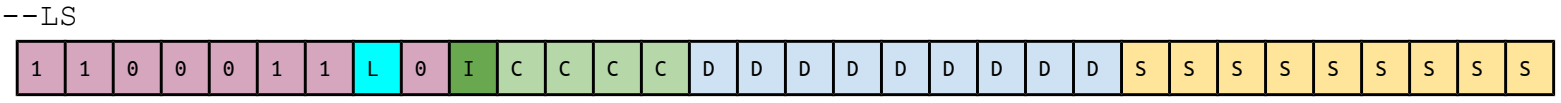
--LS



Write byte to hub via the FIFO (blocking) ???

WRWORD D/#,S/PTRx WRITE WORD

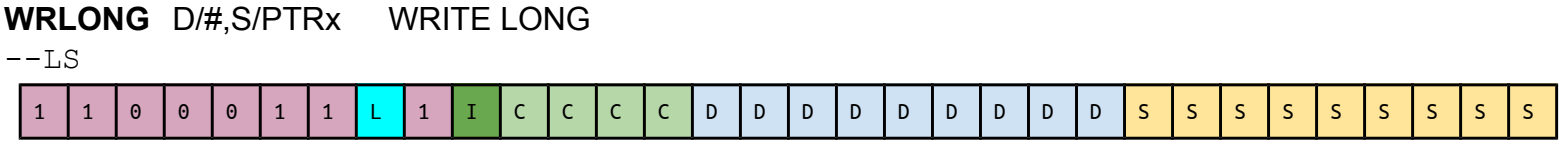
--LS



Write word to hub via the FIFO (blocking) ???

WRLONG D/#,S/PTRx WRITE LONG

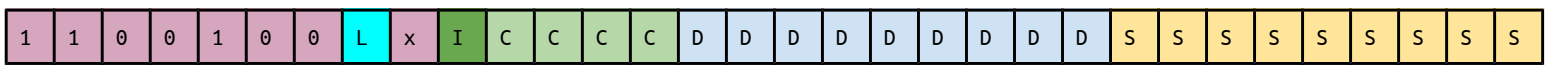
--LS



Write long to hub via the FIFO (Blocking) ???

RDFAST D/#,S/PTRx READ FAST

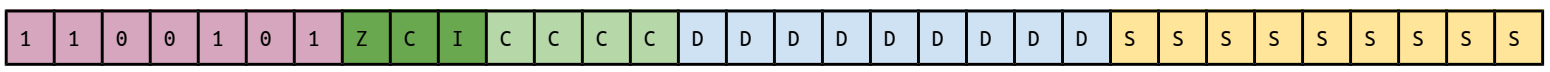
--LS



Read a long from hub memory bypassing the FIFO (blocking) ???

RDBYTE D,S/PTRx READ BYTE

ZCWS

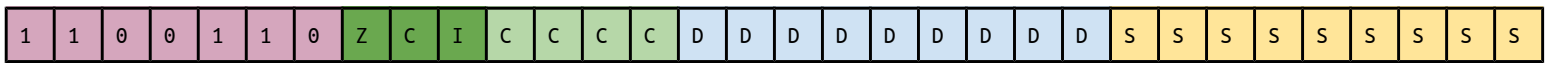


Read byte from hub memory

```
ZCWS            1100101 ZCI CCCC DDDDDDDDD SSSSSSSSS            RDBYTE  D,S/PTRx        mem             (waits for mem)
```

RDWORD D,S/PTRx READ WORD

ZCWS

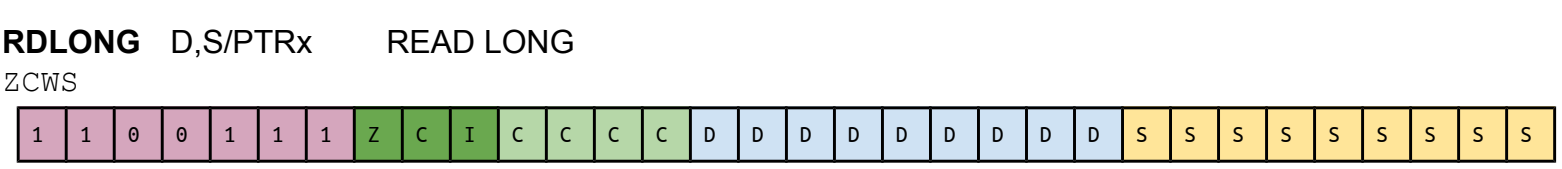


Read word from hub memory

```
ZCWS            1100110 ZCI CCCC DDDDDDDDD SSSSSSSSS            RDWORD  D,S/PTRx        mem             (waits for mem)
```

RDLONG D,S/PTRx READ LONG

ZCWS

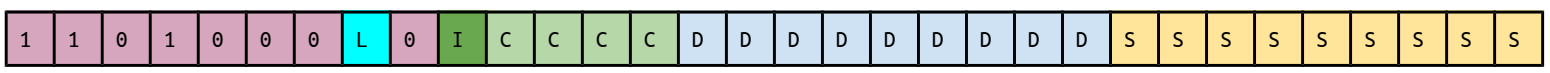


Read long from hub memory

```
ZCWS            1100111 ZCI CCCC DDDDDDDDD SSSSSSSSS            RDLONG  D,S/PTRx        mem             (waits for mem)
```

QSINCOS D/#,S# CORDIC SINCOS

--LS



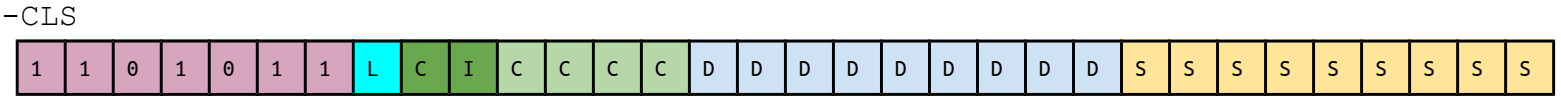
Cordic SIN COS

```
--LS            1101000 L0I CCCC DDDDDDDDD SSSSSSSSS            QSINCOS D/#,S/#         2       ..2     (1000)  wait sys

--LS            1101000 L1I CCCC DDDDDDDDD SSSSSSSSS            QARCTAN D/#,S/#         2       ..2     (1001)  wait sys
--LS            1101001 L0I CCCC DDDDDDDDD SSSSSSSSS            QROTATE D/#,S/#         3       ..2     (1010)  wait sys
--LS            1101001 L1I CCCC DDDDDDDDD SSSSSSSSS            QMUL    D/#,S/#         2       ..2     (1011)  wait sys
--LS            1101010 L0I CCCC DDDDDDDDD SSSSSSSSS            QDIV    D/#,S/#         3       ..2     (1100)  wait sys
--LS            1101010 L1I CCCC DDDDDDDDD SSSSSSSSS            QSQRT   D/#,S/#         2       ..1     (1101)  wait sys
```

COGINIT D/#,S# COG INIT

-CLS



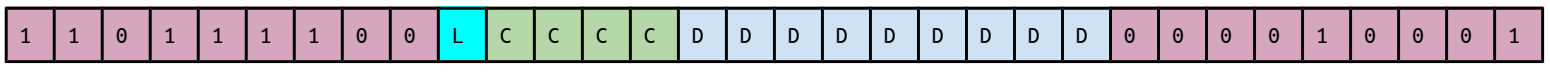
Cog init

```
-CLS wr if !L 1101011 LCI CCCC DDDDDDDDD SSSSSSSSS            COGINIT D/#,S/#        2       1?      (0010)  wait sys + 2 if WC and reg


ZCL- wr if C  1101111 0CL CCCC DDDDDDDDD 000000000            CLKSET  D/#            1       0       (0000)  wait sys + 2 if WC
ZCL-            1101111 Z1L CCCC DDDDDDDDD 000000001          COGID   D/#     WC     1       0/C     (0001)  wait sys + 2
ZCL- wr         1101111 Z00 CCCC DDDDDDDDD 000000001          COGID   D              0       1       (0001)  wait sys + 2
ZCL-            1101111 00L CCCC DDDDDDDDD 000000011          COGSTOP D/#            1       0       (0011)  wait sys
ZCL- wr         1101111 ZC0 CCCC DDDDDDDDD 000000100          LOCKNEW D              0       1/C     (0100)  wait sys + 2
ZCL-            1101111 00L CCCC DDDDDDDDD 000000101          LOCKRET D/#            1       0       (0101)  wait sys
ZCL-            1101111 0CL CCCC DDDDDDDDD 000000110          LOCKCLR D/#            1       0/C     (0110)  wait sys + 2 if WC
ZCL-            1101111 0CL CCCC DDDDDDDDD 000000111          LOCKSET D/#            1       0/C     (0111)  wait sys + 2 if WC
```

QLOG D/# LOG

ZCL-



QLOG ??

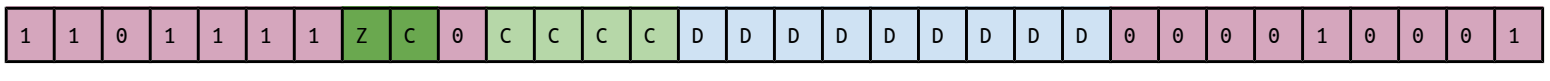
```
ZCL-            1101111 00L CCCC DDDDDDDDD 000001110          QLOG    D/#            1       ..1     (1110)  wait sys


ZCL-            1101111 00L CCCC DDDDDDDDD 000001111          QEXP    D/#            1       ..1     (1111)  wait sys

ZCL-            1101111 00L CCCC DDDDDDDDD 000000000          SETQ    D/#
```

RFBYTE D RFBYTE

ZCL- wr



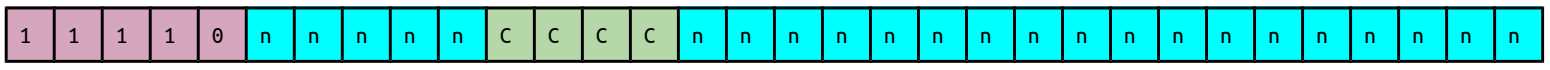
RFBYTE

```
ZCL- wr         1101111 ZC0 CCCC DDDDDDDDD 000010001          RFBYTE  D


ZCL- wr         1101111 ZC0 CCCC DDDDDDDDD 000010010          RFWORD  D
ZCL- wr         1101111 ZC0 CCCC DDDDDDDDD 000010011          RFLONG  D
ZCL-            1101111 00L CCCC DDDDDDDDD 000010100          WFBYTE  D/#
ZCL-            1101111 00L CCCC DDDDDDDDD 000011000          WFWORD  D/#
ZCL-            1101111 00L CCCC DDDDDDDDD 000011100          WFLONG  D/#
```

AUGS #23bits AUGMENT SOURCE

```
----
```



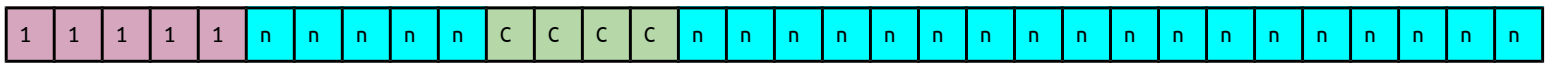
Appends n to upper bits of next immediate S

To help make hub execution practical, there are two instructions, AUGS and AUGD, which each provide 23 bits of data to extend 9-bit constants in subsequent instructions to 32 bits. AUGS is cancelled when a subsequent instruction expresses a constant S. AUGD is cancelled when a subsequent instruction expresses a constant D.

Note: Use of ## and @@ implements AUGx

AUGD #23bits AUGMENT DESTINATION

```
----
```



Appends n to upper bits of next immediate D

Instructions to be added:

```
ZCWS        * 1011001 ZCI CCCC DDDDDDDDD SSSSSSSSS          MSGIN   D,S/#          msg     (waits up to 32 clocks to receive %010_data32 message on pin S/#,
C=timeout)

--LS        * 1100000 1LI CCCC DDDDDDDDD SSSSSSSSS          WAITPAE D/#,S/#                (waits for INA)
--LS        * 1100000 0LI CCCC DDDDDDDDD SSSSSSSSS          WAITPAN D/#,S/#                (waits for INA)
--LS        * 1100001 1LI CCCC DDDDDDDDD SSSSSSSSS          WAITPBE D/#,S/#                (waits for INB)
--LS        * 1100001 0LI CCCC DDDDDDDDD SSSSSSSSS          WAITPBN D/#,S/#                (waits for INB)

--LS        * 1100100 0LI CCCC DDDDDDDDD SSSSSSSSS          MSGOUTA D/#,S/#                (send message to pin(s) on DIRA)
--LS        * 1100100 1LI CCCC DDDDDDDDD SSSSSSSSS          MSGOUTB D/#,S/#                (send message to pin(s) on DIRB)

--LS        * 1100110 0LI CCCC DDDDDDDDD SSSSSSSSS          PICKZC  D/#,S/#        adr     (always writes Z/C) - maybe change to {base5+offset4, base5) (1-
based offset)
```

JMP #abs JUMP

```
----
```

1	1	0	1	1	0	0	0	0	n	C	C	C	C	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

Jump to 19-bit absolute address

```
----        * 1101100 00n CCCC nnnnnnnnn nnnnnnnnn    *       JMP     #abs           adr     (jump to 19-bit absolute address)
```

JMP @rel JUMP

```
----
```

1	1	0	1	1	0	0	0	1	n	C	C	C	C	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

Jump to 19-bit relative address

```
----        * 1101100 01n CCCC nnnnnnnnn nnnnnnnnn    *       JMP     @rel           adr     (jump to 19-bit relative address)
```

CALL #abs CALL

```
----
```

1	1	0	1	1	0	0	1	0	n	C	C	C	C	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

Call to 19-bit absolute address using 4-level stack

```
----        * 1101100 10n CCCC nnnnnnnnn nnnnnnnnn    *       CALL    #abs           adr     (call to 19-bit absolute address, using 4-level stack)
```

CALL @rel CALL

```
----
```

1	1	0	1	1	0	0	1	1	n	C	C	C	C	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

Call to 19-bit relative address using 4-level stack

```
----        * 1101100 11n CCCC nnnnnnnnn nnnnnnnnn    *       CALL    @rel           adr     (call to 19-bit relative address, using 4-level stack)
```

CALLA #abs CALLA

```
----
```

1	1	0	1	1	0	1	0	0	n	C	C	C	C	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

Call to 19-bit absolute address using PTRA as stack pointer

```
----        * 1101101 00n CCCC nnnnnnnnn nnnnnnnnn    *       CALLA   #abs           adr     (call to 19-bit absolute address using PTRA)
```

CALLA @rel CALLA

```
----
```

1	1	0	1	1	0	1	0	1	n	C	C	C	C	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

Call to 19-bit relative address using PTRA as stack pointer

```
----        * 1101101 01n CCCC nnnnnnnnn nnnnnnnnn    *       CALLA   @rel           adr     (call to 19-bit relative address using PTRA)
```

CALLB #abs CALLB

```
----
```

1	1	0	1	1	0	1	1	0	n	C	C	C	C	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

Call to 19-bit absolute address using PTRB as stack pointer

```
----        * 1101101 00n CCCC nnnnnnnnn nnnnnnnnn    *       CALLB   #abs           adr     (call to 19-bit absolute address using PTRB)
```

CALLB @rel CALLB

```
----
```

1	1	0	1	1	0	1	1	1	n	C	C	C	C	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

Call to 19-bit relative address using PTRB as stack pointer

```
----        * 1101101 01n CCCC nnnnnnnnn nnnnnnnnn    *       CALLB   @rel           adr     (call to 19-bit relative address using PTRB)


ZCW-        * 1101111 ZCx CCCC DDDDDDDDD xxxxx0000          GETCNT  D              cog     (get CNT into D)
ZCW-        * 1101111 ZCx CCCC DDDDDDDDD xxxxx0001          GETRND  D              cog     (get RND into D)
```

```
ZCW-          * 1101111 ZCx CCCC DDDDDDDDD xxxxx0010           POP     D               cog     (pop 4-level stack into D) (D[20:19] into Z/C via WZ/WC for
POP..CALLB D)



ZCR-          * 1101111 ZCx CCCC DDDDDDDDD xxxxx0011           CALL    D               adr     (call to D[18:0] using 4-level stack)
ZCR-          * 1101111 ZCx CCCC DDDDDDDDD xxxxx0100    *      CALLA   D               adr     (call to D[18:0] using PTRA stack)
ZCR-          * 1101111 ZCx CCCC DDDDDDDDD xxxxx0101    *      CALLB   D               adr     (call to D[18:0] using PTRB stack)

--L-          * 1101111 00L CCCC DDDDDDDDD xxxxx0110           PUSH    D/#                     (push D/# into 4-level stack)
--L-          * 1101111 00L CCCC DDDDDDDDD xxxxx0111           SETVID  D/#                     (set video mode)
--L-          * 1101111 00L CCCC DDDDDDDDD xxxxx1000           WAIT    D/#                     (wait for some number of clocks, 0 same as 1)
--L-          * 1101111 00L CCCC DDDDDDDDD xxxxx1001           WAITPX  D/#                     (wait for any edge on pin D/#)
--L-          * 1101111 00L CCCC DDDDDDDDD xxxxx1010           WAITPR  D/#                     (wait for pos edge on pin D/#)
--L-          * 1101111 00L CCCC DDDDDDDDD xxxxx1011           WAITPF  D/#                     (wait for neg edge on pin D/#)

----          * 1101111 00x CCCC xxxxxxxxx xxxxx1100           SETQ    D/#

----          * 1101111 ZCx CCCC xxxxxxxxx xxxxx1101           RET                     cog     (return using 4-level stack)
----          * 1101111 ZCx CCCC xxxxxxxxx xxxxx1110    *      RETA                    cog     (return using PTRA stack)
----          * 1101111 ZCx CCCC xxxxxxxxx xxxxx1111    *      RETB                    cog     (return using PTRB stack)

----

---- wr       * 111000r rrn CCCC nnnnnnnnn nnnnnnnnn           LOCADDR reg,#abs        adr     (write 19-bit absolute address to $1F2..$1F9, includes PTRA/PTRB)
---- wr       * 111001r rrn CCCC nnnnnnnnn nnnnnnnnn           LOCADDR reg,@rel        adr     (write 19-bit relative address to $1F2..$1F9, includes PTRA/PTRB)
---- wr       * 111010r rrn CCCC nnnnnnnnn nnnnnnnnn    *      LINK    reg,#abs        adr     (jump to 19-bit absolute address, write {Z,C,P[18:0]} to
$1F2..$1F9)
---- wr       * 111011r rrn CCCC nnnnnnnnn nnnnnnnnn    *      LINK    reg,@rel        adr     (jump to 19-bit relative address, write {Z,C,P[18:0]} to
$1F2..$1F9)
```