ICL 2900 order code

(System Function Language)

15.1 Introduction

This chapter contains full specifications of all SFL functions. The specifications are in alphabetical order of function mnemonic and, with slight variations, are in a standard format as described below.

Each specification is largely self-contained, but assumes a basic knowledge of SFL and the 2900 Series. For further information on any point, the reader should consult other chapters of this publication by use of the index.

15.2 Format of specifications

Each specification is headed by the function mnemonic and hexadecimal function code.

The specifications are divided into sections as follows. Standard abbreviations are used as described in section 15.3.

1 TYPE This defines the type of function (primary, secondary, tertiary or pseudo-tertiary) and the length of the function (see section 6.1)

2 TERMINAL OPERAND LENGTH This gives, where applicable, the length in bits of the terminal operand of the function (see section 6.2)

3 PERMISSIBLE ACS Where the accumulator is involved in the operation, the permissible value or values of ACS are given

4 EFFECT ON REGISTERS This section contains a table showing the state of the registers affected by the successful execution of the function. This is a useful quick reference section particularly for functions setting such registers as CC and OV

5 PROGRAM ERRORS A list of possible object program errors caused by incorrect execution of the function. Appropriate sections of Chapter 11 are referenced for further information

6 SUMMARY A brief outline description of the function

7 FORMATS A summary of the possible formats of the function, including permitted variants and operands

8 DESCRIPTION A full description of the function and its execution. In the case of simple functions the information in the summary and format sections is adequate and the description section is omitted

15.3 Standard abbreviations

A number of abbreviations are used in the specifications. These include the normal abbreviations for registers described in Chapter 1 (such as ACC for the accumulator). The following abbreviations are used for function variants:

Abbreviation	*Meaning*
s	A simple variant (see section 6.3.2)
c	A completing variant (see section 6.3.3)
ep	One of the explicit variants allowed for a primary function (see section 6.3.4) Note that the D variant is an exception to this general format, as it may be used with: 1 No operand 2 With a symbolic operand which may be an on-stack, local data item or in-code literal
et	One of the explicit variants allowed for a tertiary function (see section 6.5.2)

Type: Primary, length 32 bits

Terminal operand length: 128

Effect on registers

LSTB Overwritten by value in operand

Other registers are loaded with the values in the process dump.

Program errors

1 Operand addressing errors

2 Privilege

Note: Masking of program error interrupts is contolled by the program and interrupt mask bits in PSR and SSR at the start of execution of ACT. The occurrence of an unmasked program error during the early stages of execution will prevent loading of LSTB, SSN and the other processor registers (effectively it is the old process being interrupted).

Summary

ACT restores a process dumped following a stack-switching interrupt (see section 11.1) and resumes execution of it. If bit 31 of word 1 of the operand is set to 1 an event pending interrupt occurs on resumption of the process.

Formats

ACT	*oper*
ACT.*s*	*im*
ACT.*c*	
ACT.*ep*	*intlit*

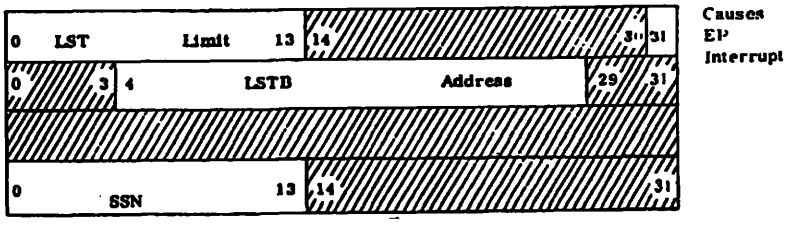
where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

The format of the terminal operand is summarised in the diagram below.



Generates Base Address of (SSN + 1) by concatenating SSN bits 0 - 12 with the bit pattern 100 00 (Note SSN is always even).

Description

The first two words of the terminal operand are loaded into LSTB. Bits 0 to 13 of the fourth word contain the new stack segment number; the base address of segment SSN+1 is generated by concatenating the bit pattern

10...0

with the operand bits. If the segment number is in the range 0 to 8191 the new LSTB is used to translate it and obtain the real address of the dump. Words 0 to 15 of the segment (the *process state*) are copied to the appropriate registers.

If there is disagreement between the values of SSN specified by the operand and words 0 and 4 of the process state the result is undefined. A system error interrupt may occur if an attempt is made to load an odd number into SSN.

The restored process is resumed at the instruction specified by PC, qualified if necessary by the setting of II.

If bit 31 of word 1 of the operand is 1 an event pending interrupt occurs on resumption of the process unless the EP interrupt mask is set in the new SSR. The interrupt may occur before the dumped registers have been completely restored or, if II is set, it may occur after execution of the incomplete instruction. The EP bit of SSR (bit 6) is ignored and not cleared.

Notes:

1 Virtual addressing mode is assumed

2 Use of ACT requires privilege (see section

3 PSTB is not altered

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

B Contains result

OV Cleared unless B overflow occurs

All other registers unchanged

Program errors

Operand addressing errors

B overflow

Summary

The contents of the operand are added to the contents of B, both being treated as signed 32 bit integers. The least significant 32 bits of the sum are left in B. If overflow occurs (sum $<-2^{31}$ or $>2^{31}-1$) OV is set, otherwise OV is cleared.

Formats

ADB	*oper*
ADB.*s*	*im*
ADB.*c*	
ADB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

ACC Contains result

OV Cleared

All other registers are unchanged

Program errors

Operand addressing errors

ACS = 128

Summary

A logical AND operation is performed between ACC and the operand. The result is placed in ACC.

The result of the AND operation is determined by the following table:

Original ACC bit	*Operand bit*	*Resultant ACC bit*
0	0	0
0	1	0
1	0	0
1	1	1

Formats

AND	*oper*
AND.*s*	*im*
AND.*c*	
AND.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Permissible ACS: 64

Effect on registers

ACC, DR } The descriptors in these registers have the length and address fields updated.

Program errors

Failure of standard checks for store-to-store operations (see section 6.4.4)

Summary

A descriptor in DR describes a string of bytes (the *DR string*). A logical AND operation is performed between each byte of this string and either

1 The filler byte (32 bit form), or

2 The corresponding byte of a string described by a descriptor in ACC (16 bit form)

Each byte of the DR string (up to the specified number) is replaced by the result of the comparison.

Formats

ANDS	*mask, filler*
ANDS.N	*ulit*
ANDS.N	*ulit, mask, filler*

where

mask is unused and must be a literal zero

filler is a one-byte literal (conventionally hexadecimal)

ulit is an integer literal specifying the number of bytes of the DR string to be operated upon

Description

A string descriptor must be loaded into DR (and, if necessary, ACC) before ANDS is executed.

In the case of the 16 bit form, the ACC string is used and must be of adequate length (that is, at least equal in length to the number of bytes to be compared).

In the 32 bit case, the specified number of bytes of the DR string is compared to the filler byte.

If the number of bytes to be compared is specified as zero, a null operation is performed and DR and ACC are unchanged.

For each byte compared, the DR (or ACC and DR) descriptors are updated by incrementing the address field and decrementing the length field.

6E ASF

(adjust stack front)

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

SF Points to new stack front

All other registers are unchanged

Program errors

1 Operand addressing errors

2 New SF ≤ LNB

3 Segment overflow

Summary

The operand, regarded as a signed number of words, is added to the word address in SF. Any new stack locations created are not cleared.

Formats

ASF	*oper*
ASF.*s*	*im*
ASF.*c*	
ASF.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The terminal operand, regarded as a signed integer, is added to the word address in SF. The stack front is thus conceptually raised or lowered depending on whether the operand is positive or negative. If it is raised, the new stack locations created are not cleared.

If the terminal operand is TOS, SF is decremented as the operand is accessed before being adjusted.

ASF

The following checks are made:

1 That the adjustment would not cause the stack to overflow into another segment

2 That the new value of SF would be greater than LNB

If either check fails, an error occurs and SF remains unaltered.

If the location specified by the new value in SF lies beyond the stack segment limit (unpaged segment), or lies in a page not in main store, a virtual store interrupt occurs as if an attempt had been made to access the specified location.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

ACC, ACS, B, CC	Undefined for system call, otherwise unchanged
DR	Contains copy of access descriptor to target procedure (descriptor-descriptor if target procedure has a PLT, otherwise a code descriptor)
LNB	Unchanged
PC	Contains address of target procedure's code
XNB	Undefined for system call, otherwise unchanged

Note: The state of the registers is that on entry to the target procedure.

Program errors

1 Operand addressing errors for jump functions (save that code and system call descriptors are permitted)

2 SF ≤ LNB+2

Summary

The CALL function is used within a procedure (the *calling procedure*) usually to enter another (the *target procedure*), which may be

1 Declared as a PROC, accessible from any module

2 Declared as an LPROC, accessible only from the module in which it is declared

3 A system or library procedure

However, the destination of a call may be a label, or given by a descriptor in DR or elsewhere.

Certain standard operations should be performed before the call. These are termed the *precall sequence* (see *Description*). Execution of CALL stores details of the calling procedure for return using EXIT.

Formats

The most usual form of CALL is

CALL	.*procname*

where *procname* is the name of the target procedure.

This form compiles into one of the following, which the programmer may also use explicitly:

CALL.IX	*PLTd*
CALL.IP	*reld*

where

PLTd is the displacement within a normal PLT of the descriptor to the target procedure

reld is the displacement within an in-line PLT relative to the CALL instruction of the descriptor to the target procedure

The other formats of a primary function may also be used, namely:

CALL	*sname*
CALL.*s*	*im*
CALL.*c*	
CALL.*ep*	*intlit*

where

sname is a symbolic name

im is a operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The precall sequence must include the following steps if no parameters are passed:

1 Store the current LNB at TOS (which will be the target procedure's LNB)

2 Adjust SF by 4 words to allow space for the descriptor stored by CALL, see below

3 Adjust LNB for the target procedure so that it cannot corrupt the calling procedure's stack

Note: There is a system library macro available to perform the precall sequence and call, see *Developing SFL programs*

Execution of CALL has the following effects:

1 The *link descriptor* pointing to the instruction following CALL is stored on stack in positions LNB+1 and LNB+2 for eventual return from the target procedure

2 A copy of the access descriptor to the target procedure is placed in DR

3 PC is set to the first instruction in the code of the target procedure

Execution of the target procedure then begins.

The link descriptor is an unbounded code descriptor (type 3), subtype 33). The second word (at LNB+1) contains the *return address* (the address of the instruction after CALL in the calling procedure). The first word (at LNB+2) contains the descriptor details in bits 0 to 7 and a copy of the calling procedure's PSR in the remaining bits in the following format.

Bits	*Contents*
8 to 11	ACR
12	D
13	PRIV
14	OV
15	E
16 to 23	program mask
24 to 26	zero
27	I
28 and 29	CC
30 and 31	ACS

If parameters are passed to the target procedure they must be set up on stack in the pre-call sequence. SF should be adjusted to allow space for them.

If the call is to a system procedure, or to a procedure at a different ACR level to the calling procedure a system call interrupt occurs and the system call mechanism is invoked (see section .). In this case, the state of ACC, B and XNB is undefined on entry to the target procedure.

If the terminal operand is an escape descriptor, the escape mechanism is invoked.

Type: Primary, length 16 bits, see section 6.3.5

Permissible ACS: 32, 64

Effect on registers

ACS Unchanged unless it was originally 128 when it will be set to 64

OV Cleared unless fixed point overflow occurs

Program errors

Fixed-point overflow

Summary

CBIN converts the contents of ACC from packed decimal to fixed-point binary format. The length of the result can be 32 or 64 bits depending on ACS.

Format

```
CBIN
```

Description

The contents of the accumulator (assumed to be in packed decimal format) are converted to fixed-point binary. The setting of ACS determines the length of the binary number. If ACS = 128 it is set to 64.

If overflow occurs OV is set and a fixed-point overflow interrupt occurs.

EE CDEC

(convert to decimal)

Type: Primary, length 16 bits, see section 6.3.5

Permissible ACS: 32, 64

Effect on registers

ACC	Converted to decimal
ACS	Doubled
OV	Cleared

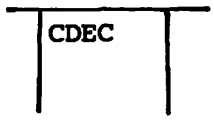
Program errors

ACS = 128

Summary

The contents of ACC, a signed integer, are converted to decimal format and left in ACC. ACS is doubled.

Formats



Type: Secondary, length 16 bits, see section 6.4.2

Effect on registers

ACC unchanged

CC
- 0 No overlap, or length of ACC string zero
- 1 Overlap on the left (ACC address > DR address) This is a permissible overlap
- 2 Overlap on the right (ACC address < DR address). This overlap may cause corruption during a string move

DR Unchanged

All other registers are unchanged

Program errors

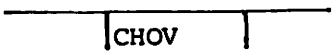
Failure of either of standard checks 1 and 3 for store-to-store functions

Summary

CHOV checks whether the strings described by descriptors in ACC and DR overlap and, if so, whether the overlap is to the left or the right (for example, before a MOVE function is used).

The result of the test is indicated by the setting of CC as described above. For a description of the various types of overlap and their significance, see section

Format



Type: Secondary, length 16 bits, see section 6.4.2

Effect on registers

ACC DR	Address and length fields updated

Program errors

1 Failure of any of the standard checks 1, 2 or 3 for store-to-store functions

2 Address in DR not a multiple of 4

3 Number of bytes to be packed specified as zero or not a multiple of 4

4 Destination vector too small

Summary

COM converts a string of bytes from unpacked (8-bit characters) form to packed (6-bit characters) form. The unpacked source string is described by a descriptor in DR, the vector to hold the packed string by a descriptor in ACC.

Formats

COM COM.N	*ulit*

where *ulit* (an unsigned integer literal) is the number of characters to be packed.

Description

When there is no variant, the number of bytes to be packed is determined by the descriptor in DR, otherwise by the operand *ulit*. The first two bits of each byte of the unpacked string are ignored. The following conditions must be observed (where *l* is the number of bytes to be packed):

1 Both descriptors must be loaded into the registers

2 $l > 0$

3 *l* must be a multiple of 4

4 The destination vector must be at least $^3/_4 l$ in length

5 The initial address in DR must be a multiple of 4

The address and length fields of the descriptors are updated during execution of the function.

If the strings overlap, the correct results will still be produced provided that the first or last byte of the unpacked string lies within the packed string (the string lengths are defined as *1* and $^{3}/_{4}$*1*) otherwise the results are undefined.

For a description of packed and unpacked character formats, see section

Type: Primary, length 16 bits, see section 6.3.5

Permissible ACS: 32, 64

Effect on registers

ACC Packed

OV Cleared

All other registers are unchanged

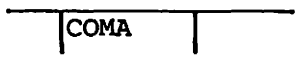
Program error

ACS = 128

Summary

The COMA function converts the contents of the accumulator from unpacked character form to packed character form. The accumulator size may be 32 or 64 bits.

Format

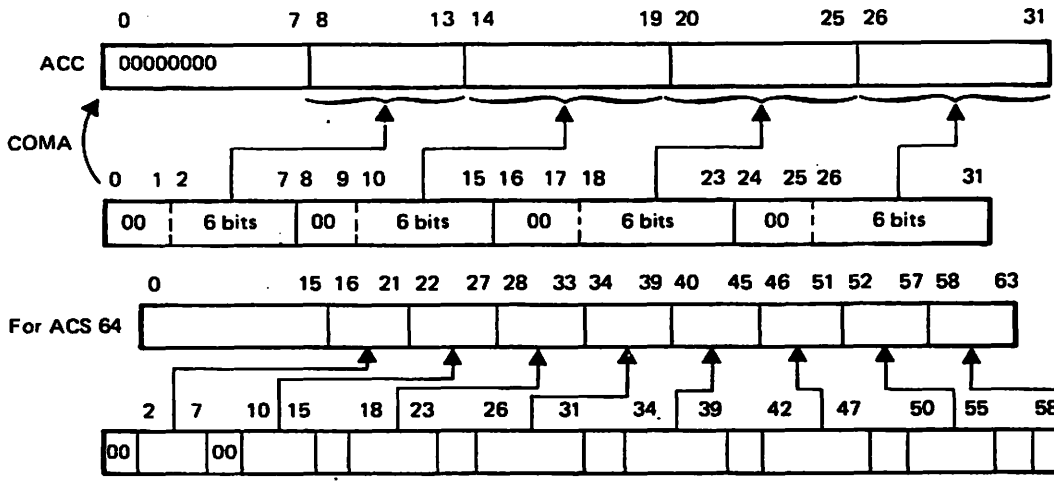


Description

The contents of ACC are converted to packed character form (see section). The original contents of the first two bits of each byte packed are ignored. Zeros are generated in bits 0 to 7 (ACS = 32) or 0 to 15 (ACS = 64).

The process is summarised in the diagrams below.

For ACS 32



Type: Primary, length 16 or 32 bits, see section 6.3.5.

Terminal operand length: 32

Effect on registers

CC 0 Operand = B

1 B < operand

2 B > operand

Program errors

Operand addressing errors

Summary

The contents of B are compared arithmetically with the operand, both being regarded as signed integers. The result of the comparison is given by the setting of CC.

Formats

CPB	*oper*
CPB.*s*	*im*
CPB.*c*	
CPB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

B Incremented by one

CC
- 0 B = operand
- 1 B < operand
- 2 B > operand

OV Normally cleared. Set if fixed-point overflow occurs as a result of incrementing B.

All other registers are unchanged

Program errors

1 Operand addressing errors

2 Fixed-point overflow

Summary

The contents of B are compared arithmetically with the operand and B is then incremented by 1. The result of the comparison is given by the setting of CC.

Formats

CPIB	*oper*
CPIB.*s*	*im*
CPIB.*c*	
CPIB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Permissible ACS: 64

Effect on registers

ACC The descriptor points to the byte found unequal (CC > 0), or the end of the string if reached (CC = 0)

CC
- 0 Strings equal or null operation performed
- 2 Inequality: DR string byte > ACC string byte (unmasked portions)
- 3 Inequality: DR string byte < ACC string byte (unmasked portions)

DR The descriptor points to the byte found unequal (CC > 0) or the byte reached when the ACC string was exhausted (CC = 0)

Program errors

Failure of standard checks for store-to-store operations, see section 6.4.4.

Summary

Descriptors in ACC and DR describe byte strings. CPS causes the strings to be compared, byte by byte, until an unequal pair of bytes is found or the specified number of bytes have been compared. CC is set to indicate the result of the comparison.

Formats

CPS	
CPS	*mask, filler*
CPS.N	*ulit*
CPS.N	*ulit, mask, filler*

where

mask and *filler* are one-byte literals (conventionally hexadecimal)

ulit is an unsigned integer literal specifying the number of bytes to be compared.

Description

The string descriptors must be loaded into ACC and DR before the CPS function is executed.

In the case of the 16 bit form of the function, the string described in ACC must be of adequate length, that is, at least equal to the number of bytes to be compared.

When the 32 bit form is used, the mask and filler bytes can be employed as follows:

1 If the mask is present, bits of each byte in the two strings are only compared when they correspond to zeros in the mask byte. Otherwise that bit position is ignored

2 If the string described in ACC is too short, it is extended to the required length with copies of the filler byte before comparison

The descriptors in ACC and DR are updated during execution of CPS and will be left pointing to the last bytes compared. These will be the bytes found unequal, or the point reached when the end of the ACC string was encountered if no inequality was found.

If the number of bytes to be compared is specified as zero, no error will result. No comparisons are performed, ACC and DR are not altered, and CC is set to 0.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

Program errors

1 Operand addressing errors

2 Non-zero bits of stored item truncated

Summary

The contents of the PM, CC and ACS fields of the program status register are copied to the location specified by the terminal operand (which must be 32 bits long).

Formats

CPSR	*oper*
CPSR.*s*	*im*
CPSR.*c*	
CPSR.*ep*	*intlit*

where

oper is the terminal operand (32 bit store location)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The contents of PSR are copied to the specified location in the following format:

Field	*Contents*
0 to 15	Zeros
16 to 23	PM
24 to 27	1110 (see note below)
28 and 29	CC
30 and 31	ACS

Note: Subsequent use of the location by MPSR causes the PM and CC fields to be overwritten, but not the ACS field unless bit 27 is set to 1.

Type: Primary, length 16 bits, see section 6.3.5

Effect on registers

ACC Contents of DR

ACS 64

OV Cleared

All other registers are unchanged

Program errors

General errors

Summary

ACS is set to 64 and the contents of the descriptor register are copied into the accumulator.

Format

(decimal add)

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

OV Cleared unless decimal overflow occurs

All other registers are unchanged

Program errors

1 Operand addressing errors

2 Decimal overflow

Summary

The operand (length determined by ACS setting) is added to the contents of ACC. If overflow occurs, the result will be correct (including the sign digit) apart from the digit that overflowed.

Formats

DAD	*oper*
DAD.*s*	*im*
DAD.*c*	
DAD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

CC 0 ACC = operand

1 ACC < operand

2 ACC > operand

All other registers are unchanged

Program errors

Operand addressing errors, see section 11.2.1.

Summary

Each digit of the terminal operand is compared with the corresponding digit in ACC arithmetically. There is no check that the digits lie in the range 0 to 9. The result of the comparision is given by the setting of CC.

Formats

DCP	*oper*
DCP.*s*	*im*
DCP.*c*	
DCP.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

ACC Contains result

OV Cleared, even if the zero divide interrupt occurs

All other registers are unchanged

Program errors

1 Operand addressing errors

2 Zero divide (maskable)

Summary

The contents of the accumulator are divided by the terminal operand and the unrounded quotient left in the accumulator. If the divisor is zero the result is undefined and the zero divide interrupt occurs (maskable).

Formats

DDV	*oper*
DDV.*s*	*im*
DDV.*c*	
DDV.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The resulting quotient value is such as to produce a remainder which is either zero or of the same sign as the dividend and numerically less than the divisor.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

B Decremented by one

OV Cleared unless overflow occurs

All other registers are unchanged

Program errors

1 Operand addressing errors for a jump function (see section 11.2.1

2 B overflow (maskable)

Summary

The B register is decremented by one. If the result is not zero, control is transferred to the location specified by the terminal operand (label or virtual address) otherwise execution continues with the next function in sequence.

Formats

DEBJ	*destination*
DEBJ.*s*	*oper*
DEBJ.*c*	
DEBJ.*ep*	*intlit*

where

destination is either a label or a one-word LOCAL data item which holds the virtual address of the jump destination

oper is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

If B contains -2^{31} before execution of the DEBJ function, OV is set, the value $+2^{31}-1$ left in B and a B overflow interrupt occurs (maskable).

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

ACC Contains quotient

CC 0 Remainder = 0 or (remainder >0 and divisor >0)

1 Remainder >0 and divisor <0

2 Remainder <0 and divisor >0

3 Remainder <0 and divisor <0

OV Cleared, even if outcome erroneous

Program errors

1 Operand addressing errors, see section 12.2.1

2 Zero divide (maskable)

Summary

The contents of the accumulator are divided by the operand. The quotient is left in the accumulator and the remainder is stacked causing SF to be incremented appropriately (the size of all the quantities involved is determined by ACS).

Formats

DMDV	*oper*
DMDV.*s*	*im*
DMDV.*c*	
DMDV.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The quotient value is such as to produce a remainder which is either zero or of the same sign as the dividend and numerically smaller than the divisor. CC is set to indicate the result of the division.

If the divisor is zero, the quotient, the remainder (which is still stacked) and the setting of CC are all undefined and the zero divide interrupt occurs (maskable).

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

OV Cleared unless decimal overflow occurs

All other registers are unchanged

Program errors

1 Operand addressing errors, see section 11.2.1

2 Decimal overflow (maskable)

Summary

The product of the terminal operand and the accumulator is left in the accumulator. If the product cannot be accommodated by ACC, the result is undefined, OV is set and a decimal overflow interrupt occurs (maskable).

Formats

DMY	*oper*
DMY.*s*	*im*
DMY.*c*	
DMY.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

OV Cleared

All other registers are unchanged

Program errors

1 Operand addressing errors, see section 11.2.1

2 ACS = 128

Summary

The contents of the accumulator and the terminal operand are multiplied, ACS is doubled and the product left in the accumulator. An original ACS of 128 is not permitted.

Formats

DMYD	*oper*
DMYD.*s*	*im*
DMYD.*c*	
DMYD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

ACC Contains result

OV Cleared, even if the zero divide interrupt occurs

All other registers are unchanged

Program errors

1 Operand addressing errors

2 Zero divide (maskable)

Summary

The contents of the terminal operand are divided by the accumulator and the unrounded quotient left in the accumulator. If the divisor is zero the result is undefined and the zero divide interrupt occurs (maskable).

Formats

DRDV	*oper*
DRDV.*s*	*im*
DRDV.*c*	
DRDV.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The resulting quotient value is such as to produce a remainder which is either zero or of the same sign as the dividend and numerically less than the divisor.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

ACC Contains result

OV Cleared unless overflow occurs

Program errors

1 Operand addressing errors

2 Decimal overflow

Summary

The contents of ACC are subtracted from the terminal operand and the result left in ACC. If overflow occurs, the result (including the sign digit) will be correct, apart from the digit that overflowed.

Formats

DRSB	*oper*
DRSB.s	*im*
DRSB.*c*	
DRSB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

ACC Contains result

OV Cleared unless overflow occurs

All other registers are unchanged

Program errors

1 Operand addressing errors

2 Decimal overflow

Summary

The operand is subtracted from the contents of ACC. If overflow occurs, the result will be correct (including the sign digit) apart from the digit that overflowed.

Formats

DSB	*oper*
DSB.*s*	*im*
DSB.*c*	
DSB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 7

Permissible ACS: 32, 64, 128

Effect on registers

ACC Contents shifted as specified by operand

OV Set if any 1 bits shifted off the left end of ACC, otherwise cleared

Program errors

1 Operand addressing errors, see section 11.2.1

2 Decimal overflow (maskable)

Summary

The decimal value in ACC is shifted as specified by the terminal operand of the function. The least significant 7 bits of the operand are used, being interpreted as a signed integer in the range - 64 to +63 (*i*).

The contents of ACC are shifted 4*i* places (to the left if *i* > 0, to the right if *i* < 0).

Formats

DSH	*val*
DSH.*s*	*im*
DSH.*c*	
DSH.*ep*	*intlit*

where

val is a literal or store location of any length controlling the shift

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The terminal operand may be of any length. The least significant 7 bits are interpreted as a signed integer *i* in the range -64 to +63, all other bits being ignored.

DSH

If *i* > 0 then all but the least significant 4 bits of ACC are shifted *i* digits (4*i* binary places) to the left. The sign digit is unaltered. A zero bit is inserted in the bit position next to the sign digit for each place shifted. If any of the bits shifted off the left end of ACC are ones, OV is set and an interrupt occurs.

If *i* < 0 then all but the least significant 4 bits of ACC are shifted *i* digits (4*i* binary places) to the right. The sign digit is unaltered. Any bits shifted out of the position to the left of the sign digit are lost. A zero bit is inserted at the more significant end of ACC for each place shifted. OV is cleared.

Type: Primary, length 16 bits

Effect on registers

PC Overwritten by value at TOS

PSR The D bit is set initially but cleared by the instruction executed as a result of obeying ESEX

SF Decremented by 1

All other registers are unchanged.

Program errors

Universal types

Summary

ESEX is used to exit from an escape routine
Normally it results in the instruction that triggered the escape mechanism being executed.

Format

Description

Bits 0 to 30 of the word at TOS overwrite PC (bit 31 is ignored) and SF is decremented by 1. The D bit in PSR is set and a jump made to the instruction now pointed at by PC (normally the one which triggered the escape mechanism). If direct access, or access via DR is specified by the instruction, this will be carried out. If the instruction accesses store indirectly, via a descriptor in store, the descriptor in DR will be used instead (it is assumed that the escape routine will have loaded DR).
If the instruction specifies modification, this will be performed, but on the descriptor in DR rather than the one specified. The D bit is cleared by execution of the instruction.

Type: Primary, length 16 bits (operand must be 7-bit literal)

Terminal operand length: 7

Effect on registers

ACC	Unchanged
B	Unchanged
DR	Not preserved on inward return
LNB	Value from calling procedure restored
PC	Altered to value in link descriptor
PSR	Fields may be changed to values stored in LNB+1 depending on terminal operand (includes ACR, ACS, CC, D, OV, PM, PRIV)
SF	Restored to state before CALL

Program errors

1	SF ≤ LNB+2
2	Invalid link descriptor.
3	New value of PRIV > old
4	New value of ACR < old
5	Bits 0 to 13 of LNB not the same as SSN
6	Bits 14 to 29 of LNB ≥ new SF
7	Attempt to set ACS to zero using stored PSR

Summary

EXIT is used within a procedure to terminate execution of that procedure and return control to the calling procedure using the link descriptor stored at LNB+1 and LNB+2. The terminal operand (optional) may be used to restore certain registers from the PSR values stored with the link descriptor.

Formats

EXIT	
EXIT	*oper*

where

oper is a literal operand (of which only bits 25 to 29 are significant) specifying registers to be restored from

the stored PSR

Note: If *oper* is absent, the compiler uses a default value of X'6C'

Description

The effects of EXIT are as follows:

1 SF and LNB have the values they had at the time of the pre-CALL sequence restored

2 The link descriptor is accessed and PC set to point to the statement following CALL in the calling procedure and control transferred to this instruction

The fields of PSR (stored at LNB+1) are altered as follows:

1 If the value in bits 8 to 11 is not less than the current ACR, it is used to overwrite ACR, otherwise an interrupt occurs

2 If the value in bit 13 is not greater than PRIV it overwrites PRIV otherwise an interrupt occurs

Other alterations depend on whether bits 25 to 29 of the terminal operand are set as follows.

Bit set	*Effect*
25	Bits 16 to 23 of LNB+1 overwrite PM
26	Bits 28 to 29 of LNB+1 overwrite CC
27	Bits 30 to 31 of LNB+1 overwrite ACS. If these bits are zero, an error occurs.
28	Bit 12 of LNB+1 overwrites D
29	Bit 14 of LNB+1 overwrites OV (this will not cause an overflow interrupt)

Note: An operand value of X'FF' will cause all the above registers to be overwritten. If no operand is supplied a value of X'6C' is assumed (registers PM, CC, D, OV).

Type: Primary, length 16 bits

Permissible ACS: 32, 64

Effect on registers

ACC Unpacked

OV Cleared

Other registers are unchanged

Program error

ACS = 128

Summary

EXPA converts the contents of the accumulator from packed form (6-bit characters) to unpacked form (8-bit characters).

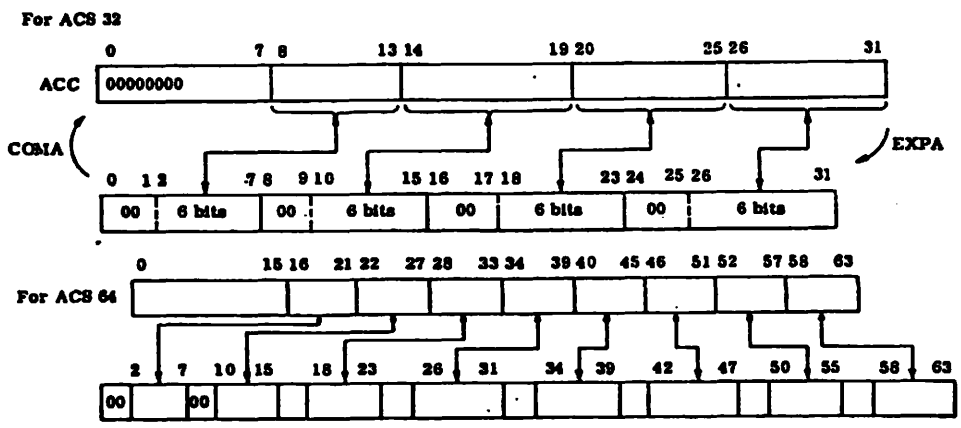
Format

```
|EXPA|
```

Description

The contents of ACC are converted to unpacked character form. The original contents of bits 0 to 7 (ACS = 32) or 0 to 15 (ACS = 64) are ignored. Zeros are generated in the first two bits of each byte (0 and 1, 8 and 9, etc.)

The process is summarised in the diagram below.



Type: Secondary, length 16 or 32 bits, see section 6.4.2

Permissible ACS: 64

Effect on registers

ACC, DR } Address and length fields updated during execution

Other registers are unchanged

Program errors

1 Failure of any of the standard checks 1, 2 or 3 for store-to-store functions (see section 6.4.4)

2 Address in DR not a multiple of 4

3 Number of bytes to be packed specified as zero or not a multiple of 4

4 Destination vector too small

Summary

EXP converts a string of bytes in packed form (6-bit characters) to unpacked form (8-bit characters).

The packed source string is described by a descriptor in ACC, the vector to hold the unpacked string by a descriptor in DR.

Formats

EXP EXP.N	*ulit*

where

ulit is an unsigned integer literal specifying the number of bytes to be unpacked

Description

When there is no variant, the number of bytes to be unpacked is determined by the descriptor in ACC, otherwise by the operand *ulit*. The first two bits of each byte of the unpacked string are set to zero. The following conditions must be observed (where *l* is the number of bytes to be unpacked):

1 Both descriptors must be loaded into the registers

2 $l > 0$

3 *l* must be a multiple of 4

4 The destination vector must be at least $^{3}/_{4}l$ in length

5 The initial address in DR must be a multiple of 4

The address and length fields of the descriptors are updated during execution of the function.

If the strings overlap, the correct results will still be produced provided that the first or last byte of the unpacked string lies within the packed string (the string lengths are defined as 1 and $^{3}/_{4}1$) otherwise the results are undefined.

For a description of packed and unpacked character formats, see section 1.4.5.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32, 64

Effect on registers

ACC As described below

ACS Set to 64 if originally 128, otherwise unchanged

CC 0 No non-zero bits lost

1 Only set if ACS originally 128. Lost portion < 0.5 (bit 71 = 0 after negation)

2 Only set if ACS originally 128. Lost portion ≥ 0.5 (bit 72 = 1 after negation)

Program errors

1 Operand addressing errors, see section 11.2.1

2 Non-zero bits of stored value truncated, see section 6.6.2

Summary

This function converts a floating point value in ACC (32 or 64 bits) to fixed point format. The exponent is adjusted and stored in the 32 bit location specified by the terminal operand. The fractional part is left in ACC as a signed integer.

Formats

FIX	*locs*
FIX.*s*	*im*
FIX.*c*	
FIX.*ep*	*intlit*

where

locs specifies a 32 bit store location to hold the adjusted exponent

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The floating point value in ACC is separated into two parts:

The exponent
The fraction and sign

The exponent is adjusted and stored as a single length signed integer in the location specified by the terminal operand of the function. The fraction and sign are left in ACC as a signed integer.

ACS may be 32, 64 or 128. If it is 128 it is set to 64 and the least significant 14 bits of the fraction lost, CC being set to indicate the nature of the lost part.

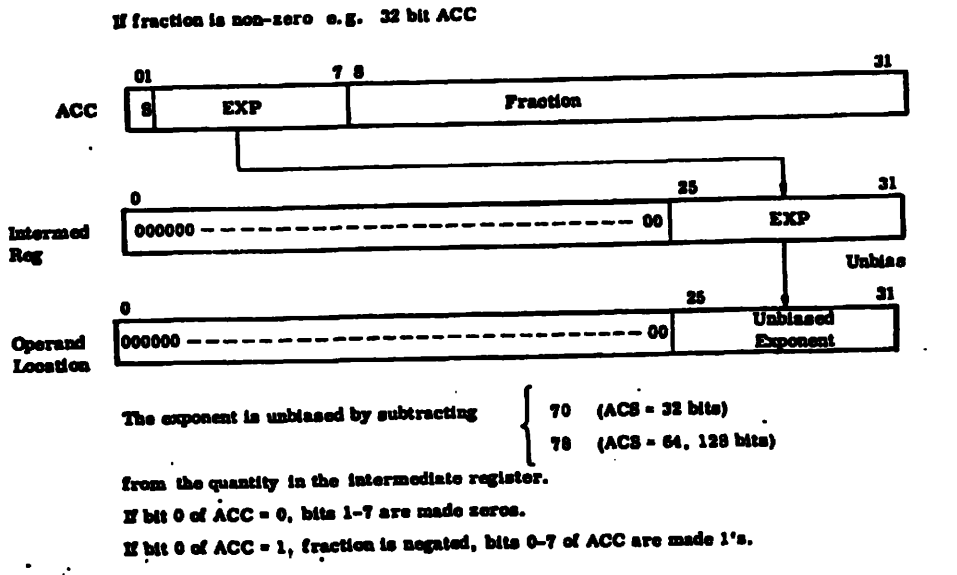
If all 6, 14 or 28 bits of the fraction are zero the following events occur:

1 If ACS is 128 it is set to 64

2 ACC and CC are set to zero

3 The location specified by the terminal operand is set to zero

If the fraction is not zero, bits 1 to 7 of ACC are stored in a 32 bit intermediate register. The exponent is unbiased by subtracting 70 (ACS = 32) or 78 (ACS > 32) from the contents of the intermediate register. The result is stored in the terminal operand location. This is summarised in the following diagram:



The contents of ACC are then tidied up as follows. If the sign bit (bit 0 of ACC) is zero then bits 1 to 7 are set to zero. If the sign bit is one, then the fraction is negated and bits 1 to 7 are set to one. If ACS > 32 then CC is set to zero.

Type: Primary, length 16 or 32 bits (see section 6.3.5)

Terminal operand length: 32 bits

Permissible ACS: 32, 64

Effect on registers

ACC See *Description*

ACS Doubled

OV Normally cleared; set if overflow occurs

All other registers are unchanged

Program errors

1 Operand addressing errors, see section 11.2.1

2 Floating- point overflow (maskable)

3 Floating-point underflow (maskable)

4 ACS = 128

Summary

The FLT function converts a fixed-point number (signed integer) in the accumulator into floating-point format. The least significant 8 bits of the terminal operand specify an exponent to be associated with the integer originally in the accumulator.

Formats

FLT	*oper*
FLT.*s*	*im*
FLT.*c*	
FLT.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

If the number in ACC is zero, the operand is ignored, ACS is doubled and the accumulator extended with zeros.

If the number in ACC is not zero, the following events take place:

1 ACS is doubled. The original contents of ACC are placed in the more significant half and the less significant half set to zero

2 The least significant 8 bits of the terminal operand are stored in an intermediate register. This value is then incremented by 72 (ACS = 64) or 80 (ACS = 128) to form the *intermediate characteristic*

3 The contents of ACC are shifted right arithmetically by 8 binary places. Bits 8 upwards then form the *intermediate fraction*. If bit 0 of ACC was originally 1 the current contents are negated to form the modulus of the fraction and bit 0 (the sign bit) is set to 1

4 The intermediate fraction is normalised by shifting it up 4 bits at a time until bits 8 to 11 are not all zeros. For each shift, 1 is subtracted from the intermediate characteristic

5 The least significant 7 bits of the intermediate characteristic overwrite bits 1 to 7 of ACC

6 If ACS = 128 the following steps take place

   (a) The least significant 64 bits of ACC are shifted to the right by 8 places

   (b) A copy of the characteristic in bits 1 to 7 is placed in bits 65 to 71, with 14 being subtracted from it (or 114 added to it if the value is less than 14)

   (c) Bit 64 is made the same as bit 0

If the most significant bit of the intermediate register is 1, underflow or overflow has occurred which will cause a maskable interrupt. If underflow occurs, ACC is set to 0.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

ACC Contains sum

OV Normally cleared; set if overflow occurs

All other registers are unchanged

Program errors

1 Operand addressing errors, see section 11.2.1

2 Fixed point overflow (maskable)

3 ACS = 128

Summary

The terminal operand is added to the contents of the accumulator and the result left in the accumulator.

Formats

IAD	*oper*
IAD.*s*	*im*
IAD.*c*	
IAD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

A fixed point overflow interrupt occurs (maskable) and OV is set if the result lies outside the range determined by ACS. In this case the result left in the accumulator is the least significant 32 or 64 bits of the true sum. Overflow can only occur when adding numbers of like signs.

ICP

(integer compare)

Type: Primary, length 16 or 32 bits 6.3.5

Operand length: ACS

Permissible ACS: 32, 64

Effect on registers

CC 0 ACC = operand

1 ACC < operand

2 ACC > operand

Program errors

1 Operand addressing errors (see section 11.2.1)

2 ACS = 128

Summary

The terminal operand is compared arithmetically with the contents of ACC and CC is set to indicate the result.

Formats

ICP	*oper*
ICP.*s*	*im*
ICP.*c*	
ICP.*ep*	*intlit*

where

int is the terminal operand (literal) or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Program errors

Universal types

Summary

IDLE causes execution of the program to cease until an interrupt of any kind occurs. A literal operand may be supplied. The value is displayed on hardware indicators and may be used to provide information to the operator.

Format

IDLE	*ulit*

where *ulit* is an unsigned literal.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

OV Cleared unless overflow occurs (see *Description*)

Program errors

1 Operand addressing errors, see section 11.2.1

2 Zero divide (maskable)

3 Fixed-point overflow (maskable)

4 ACS = 128

Summary

The contents of ACC are divided by the terminal operand and the unrounded quotient left in ACC.

Formats

IDV	*oper*
IDV.*s*	*im*
IDV.*c*	
IDV.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

If the divisor is zero OV is cleared, but a zero divide interrupt occurs and the contents of ACC are undefined. If an attempt is made to divide -1 into -2^{31} (ACS = 32) or -2^{63} (ACS = 64), OV is set, a fixed-point overflow interrupt occurs and the contents of ACC are unchanged.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

ACC	Contains result
CC	0 Remainder 0 or remainder > 0 and divisor > 0
	1 Remainder > 0 and divisor < 0
	2 Remainder < 0 and divisor > 0
	3 Remainder < 0 and divisor < 0
OV	Cleared unless fixed-point overflow occurs
SF	Incremented by 1 or 2

Program errors

1	Operand addressing errors, see section 11.2.1
2	Zero divide interrupt (maskable)
3	Fixed-point overflow (maskable)
4	ACS = 128

Summary

ACC is divided by the terminal operand, the result left in ACC and the remainder stacked.

Formats

IMDV	*oper*
IMDV.*s*	*im*
IMDV.*c*	
IMDV.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The remainder is numerically less than the divisor and, if not zero, it has the same sign as the dividend.

If the divisor is zero, OV is cleared, but a zero divide interrupt occurs and the contents of ACC, the remainder and the setting of CC are all undefined.

If the divisor is -1 and ACC has the values -2^{31} (ACS = 32) or -2^{63} (ACS = 64) OV is set, the contents of ACC are unchanged and the remainder and the setting of CC are both undefined.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

ACC Contains result

OV Cleared unless fixed-point overflow occurs

Program errors

1 Operand addressing errors, see section 11.2.1

2 Fixed-point overflow (maskable)

3 ACS = 128

Summary

The contents of ACC are multiplied by the terminal operand and the least significant 32 or 64 bits of the product (depending on ACS) are left in ACC. Overflow occurs if the product is too large.

Formats

IMY	*oper*
IMY.*s*	*im*
IMY.*c*	
IMY.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32

Effect on registers

ACC Contains product

CC 0 ACC and operand positive

1 ACC positive, operand negative

2 ACC negative, operand positive

3 ACC and operand negative

OV Cleared

Program errors

1 Operand addressing errors

2 ACS = 64 or 128

Summary

ACS should be 32 before execution of IMDV which sets it to 64. The 64 bit product of ACC and the terminal operand is left in ACC. CC is set to indicate the signs of the operand and the original contents of ACC.

Formats

IMYD	*oper*
IMYD.*s*	*im*
IMYD.*c*	
IMYD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

INCA

(increment address)

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

DR Address field updated by terminal operand

Program errors

Operand addressing errors, see section 11.2.1

Summary

The terminal operand value is added to bits 32 to 63 of DR (the address field); bits 0 to 31 are unaltered. Indirect addressing (I variants) is not permitted.

Formats

INCA	*oper*
INCA.*c*	
INCA.*ep*	*intlit*

where

oper is the terminal operand (literal or directly addressed data item name)

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Note: I and D variants are not permitted.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

CC 0 New value of operand = 0

1 New value of operand > 0

2 New value of operand < -1

3 New value of operand = -1

OV Unaltered (overflow is ignored)

Other registers are unchanged

Program errors

1 Operand addressing errors, see section 11.2.1

2 Incorrect descriptor type

3 Incorrect operand types

Summary

INCT is a semaphore instruction mainly used to provide interlocks on the peripheral communications areas in store. One is added to the terminal operand value and CC is set to indicate the new value. The original value of the operand is left in ACC. Between reading the original operand value and replacing it by the new value, access to the operand location is prevented by hardware.

Formats

INCT	*oper*
INCT.*s*	*im*
INCT.*c*	
INCT.*ep*	*intlit*

where

oper is the terminal operand (data item name)

im is an operand appropriate for indirection or moaification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Note: The terminal operand location must be in store, not a register. The following variants are not permitted: B, T, P.

Description

The prime use of this instruction is to implement semaphores, using semaphore descriptors.

The operand value is normally interpreted as the number of other processes waiting to use a shared resource. A value of -1 indicates that the resource is available.

The following restrictions apply to operand access:

1 Access is forced by hardware to bypass slave storage

2 If the operand is accessed indirectly, only a vector descriptor (type 0) with size 32 or a semaphore (type 3, subtype 40,41) descriptor may be used

3 The operand must be located in store rather than a register. Direct TOS and (PC+N) operand forms are not permitted

If slave storage is used in the processor, it is cleared as follows:

1 Stack slave store is cleared if the stack segment is marked as non-slaved in its segment table entry

2 Operand slave store is cleared of items marked non-slaved in either segment table

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Effect on registers:

DR The descriptor in DR is updated during execution of the function

Program errors

Failure of standard checks for store-to-store operations, see section 6.4.4

Summary

Each byte of the string described by a descriptor in DR is overwritten by the *source byte*. The location of this byte is determined by the form of the function and the setting of CC as follows:

CC	*32 bit form*	*16 bit form*
CC=0	Operand *literal*	Bits 24 to 31 of B
CC>0	Operand *mask*	Bits 16 to 23 of B

This function is intended for use with SUPK.

Formats

INS	
INS	*mask, literal*
INS.N	*ulit*
INS.N	*ulit, mask, literal*

where

mask and *literal* are one-byte literals (conventionally hexadecimal)

ulit is an unsigned literal specifying the number of bytes of the DR string to be overwritten

Description

When the operand *ulit* is present, it specifies the number of bytes of the DR string to be overwritten; when it is absent, all bytes are replaced.

Either the operand *mask* or *literal* may be used to overwrite the DR string depending on the setting of CC. *mask* is not used for

masking as it is with other string functions (such as MVL) whatever the setting of CC.

If the number of bytes to be overwritten is specified as zero, no error occurs. A null operation is performed leaving DR unaltered.

IRDV

(integer reverse divide)

Type: Primary, length 16 or 32, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

OV Cleared unless overflow occurs (see *Description*)

Program errors

1 Operand addressing errors, see section 11.2.1

2 Zero divide (maskable)

3 Fixed-point overflow (maskable)

4 ACS = 128

Summary

The terminal operand is divided by the contents of ACC and the unrounded quotient left in ACC.

Formats

IRDV	*oper*
IRDV.*s*	*im*
IRDV.*c*	
IRDV.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

(integer reverse subtract)

Type: Primary, length 16 or 32 bits, (see section 6.3.5)

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

ACC	Contains sum
OV	normally cleared; set if overflow occurs

Program errors

1	Operand addressing errors, see section 11.2.1
2	Fixed-point overflow (maskable)
3	ACS = 12f

Summary

The contents of the accumulator are subtracted from the terminal operand and the result left in the accumulator. The terminal operand is unchanged.

Formats

IRSB	*oper*
IRSB.*s*	*im*
IRSB.*c*	
IRSB.*ep*	*intlit*

where

oper	is the terminal operand (literal or data item name)
im	is an operand appropriate for indirection or modification
intlit	is an integer literal modifying the register specified by the explicit variant *ep*

Description

A fixed-point overflow interrupt occurs (maskable) and OV is set if the result lies outside the range determined by ACS. In this case the result left in the accumulator is the least significant 32 or 64 bits of the true result. Overflow can only occur when subtracting numbers of opposite signs.

Type: Primary, length 16 or 32 bits, (see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

ACC Contains result

OV Normally cleared; set if overflow occurs

Program errors

1 Operand addressing errors, see section 11.2.1

2 Fixed-point overflow (maskable)

3 ACS = 128

Summary

The terminal operand is subtracted from the contents of the accumulator and the result left in the accumulator.

Formats

ISB	*oper*
ISB.*s*	*im*
ISB.*c*	
ISB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

A fixed-point overflow interrupt occurs (maskable) and OV is set if the result lies outside the range determined by ACS. In this case the result left in the accumulator is the least significant 32 or 64 bits of the true result. Overflow can only occur when subtracting numbers of opposite signs.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32, 64

Effect on registers

ACC Shifted as specified by operand

CC 0: operand < 0, all bits shifted off right zeros

1: operand < 0, some ones shifted off right but last bit 0

2: operand < 0, last bit shifted off right a 1

3: operand ≤ 0, left shift or zero shift

OV Normally cleared. Set if overflow occurs

Program errors

1 Operand addressing errors

2 Fixed-point overflow (maskable)

3 ACS = 128

Summary

The contents of ACC are shifted as specified by the terminal operand. The least significant 7 bits of the operand are interpreted as a signed integer in the range -64 to +63 (*i*), the other bits are ignored.

The contents of ACC are shifted *i* places, to the left if *i*>0, to the right if *i*<0.

Formats

ISH	*oper*
ISH.*s*	*im*
ISH.*c*	
ISH.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

In a leftward shift ($i > 0$) zeros are inserted at the less significant end of ACC. If the contents of bit 0 of ACC change at any time during the shift, OV is set and a fixed-point overflow interrupt occurs (maskable). If bit 0 does not change OV is cleared.

In a rightward shift the sign bit is propagated by leaving bit 0 unchanged. Bits shifted off the right of ACC are lost, but CC is set to show what was lost. OV is always cleared.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

J causes unconditional transfer of control to a location specified by the terminal operand, which may be a program label or a virtual address.

Formats

J	*oper*
J.*s*	*im*
J.*c*	
J.*ep*	*intlit*

where

oper is the terminal operand

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Tertiary, length 16 or 32 bits, see section 6.5.4

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

This function tests the state of one of the registers ACC, B, DR or OV. Both register and state are specified by the value of the mask operand as shown in the table below. If the test fails (that is, the condition is false) control is transferred to the specified location.

Mask (hex)	*Condition*	
0	ACC = 0	Floating-point mode
1	ACC > 0	
2	ACC < 0	
3	Undefined	
4	ACC = 0	Fixed-point mode
5	ACC > 0	
6	ACC < 0	
7	Undefined	
8	ACC = 0	Decimal mode
9	ACC > 0	
A	ACC < 0	
B	DR length (bits 8 to 31) = 0	
C	B = 0	
D	B > 0	
E	B < 0	
F	OV set	

Formats

JAF	*location, mask*
JAF.et	*intlit, mask*
JAF.MD	*mask*

where

location is either a label or a LOCAL data item which holds the virtual address of the jump destination

mask is a one-byte literal (conventionally hexadecimal)
intlit is a literal modifying the register specified by *ep*
et is one of the variants D, L, X, P or S
Note: When the D variant is used, *intlit* may be omitted.

Type: Tertiary, length 16 or 32 bits, see section 6.5.4

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump instructions, see section 11.2.1

Summary

This function tests the state of one of the registers ACC, B, DR or OV. Both register and state are specified by the value of the mask operand as shown in the table below. If the test succeeds (that is, the condition is true) control is transferred to the specified location.

Mask (hex)	*Condition*	
0	ACC = 0	Floating-point mode
1	ACC > 0	
2	ACC < 0	
3	Undefined	
4	ACC = 0	Fixed-point mode
5	ACC > 0	
6	ACC < 0	
7	Undefined	
8	ACC = 0	Decimal mode
9	ACC > 0	
A	ACC < 0	
B	DR length (bits 8 to 31) = 0	
C	B = 0	
D	B > 0	
E	B < 0	
F	OV set	

Formats

JAT	*location, mask*
JAT.*et*	*intlit, mask*
JAT.MD	*mask*

where

location is either a label or a LOCAL data item which holds the virtual address of the jump destination

mask is a one-byte literal (conventionally hexadecimal)

intlit is a literal modifying the register specified by *et*

et is one of the variants D, L, X, P or S

Note: When the D variant is used, *intlit* may be omitted.

Type: Tertiary, length 16 or 32 bits, see section 6.5.4

Terminal operand length: 32

Effect on registers

PC Set to address of jump destination

All other registers are unchanged

Program errors

Operand addressing errors for jump functions

Summary

The CC register is tested against a mask supplied as an operand of the function. If the test is successful, control is transferred to a location specified either as a label or as a virtual address.

Formats

JCC	*location, mask*
JCC.*et*	*literal*
JCC.MD	*mask*

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see section ?) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

Description

The value or values of CC to be tested are selected by means of the mask, the four bits of which correspond to the four settings of CC. If the bits of the mask are m_0, m_1, m_2 and m_3 then transfer of control will occur if m_i is set to one and CC = i. Thus if bit 0 of the mask is set (X'8') the jump will occur if CC = 0.

Two or more mask bits may be set to jump on several values of CC. The jump will occur if CC has any of the selected values. Thus, a mask of X'C' (binary 1100) will cause a jump if CC is 0 or 1.

If the setting of CC is not one of those selected, execution continues with the next function insequence.

[02]

JE

(jump if equal)

Type

A pseudo-tertiary function which generates a JCC function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JE causes a transfer of control to a specified location (program label or virtual address) if CC = 0. It is used to test the result of algebraic comparisons such as ICP and UCP that set CC to 0 to indicate equality.

Formats

JE	*location*
JE.*et*	*literal*
JE.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[02]

JG

(jump if greater)

Type

A pseudo-tertiary function which generates a JCC function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JE causes a transfer of control to a specified location (program label or virtual address) if CC = 2. It is used to test the result of algebraic comparisons such as ICP and UCP that set CC.

Formats

JG	*location*
JG.*et*	*literal*
JG.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[02]

JGE

(jump if greater than or equal to)

Type: A pseudo-tertiary function which generates a JCC function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JGE causes a transfer of control to a specified location (program label or virtual address) if CC is zero or 2. It is used to test the result of algebraic comparisons such as ICP that set CC. For example, a jump will occur if JGE is used after ICP when ACC ≥ terminal operand.

Formats

JGE	*location*
JGE.*et*	*literal*
JGE.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[02]

JL

(jump if less)

Type: A pseudo-tertiary function which generates a JCC function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JL causes a transfer of control to a specified location (program label or virtual address) if CC = 1. It is used to test the result of algebraic comparisons such as ICP that set CC.

Formats

JL	*location*
JL.*et*	*literal*
JL.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[02]

JLE

(jump if less than or equal to)

Type: A pseudo-tertiary function which generates a JCC function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JLE causes a transfer of control to a specified location (program label or virtual address) if CC is 0 or 1. It is used to test the result of algebraic comparisons such as ICP set CC.

Formats

JLE	*location*
JLE.*et*	*literal*
JLE.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

JLK

(jump and link)

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

SF Incremented by 1

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

The address of the function following JLK is stacked as a 32-bit, byte address. Control is then transferred unconditionally to the location specified by the terminal operand which may be

1 A subroutine name or a label within a procedure

2 A virtual address. The top-of-stack item may be used (that is, T variants are permitted)

Formats

JLK	*oper*
JLK.*s*	*im*
JLK.*c*	
JLP.*ep*	*intlit*

where

oper is the terminal operand

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

[04]

JN

(jump if negative)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JN causes a transfer of control to a specified location (program label or virtual address) if the integer number in ACC is negative.

Formats

JN	*location*
JN.*et*	*literal*
JN.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[04]

JNB

(jump if negative, B)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNB causes a transfer of control to a specified location (program label or virtual address) if the value in the B register is negative.

Formats

JNB	*location*
JNB.*et*	*literal*
JNB.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[04]

JND

(jump if negative, decimal)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JND causes a transfer of control to a specified location (program label or virtual address) if the decimal number in ACC is negative.

Formats

JND	*location*
JND.*et*	*literal*
JND.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[02]

JNE

(jump if not equal)

Type

A pseudo-tertiary function which generates a JCC function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNE causes a transfer of control to a specified location (program label or virtual address) if CC is 1 or 2. It is used to test the result of algebraic comparisons such as ICP that set CC.

Formats

JNE	*location*
JNE.*et*	*literal*
JNE.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[06]

JNN

(jump if not negative)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNN causes a transfer of control to a specified location (program label or virtual address) if the integer number in ACC is positive or zero.

Formats

JNN	*location*
JNN.*et*	*literal*
JNN.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et in one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[06]

JNNB

(jump if not negative, B)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNNB causes a transfer of control to a specified location (program label or virtual address) if the value in B is positive or zero.

Formats

JNNB	*location*
JNNB.*et*	*literal*
JNNB.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[0E]

JNND

(jump if not negative, decimal)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNND causes a transfer of control to a specified location (program label or virtual address) if the decimal number in ACC is positive or zero.

Formats

JNND	*location*
JNND.*et*	*literal*
JNND.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[06]

JNNR

(jump if not negative, real)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNNR causes a transfer of control to a specified location (program label or virtual address) if the real number in ACC is positive or zero

Formats

JNNR	*location*
JNNR.*et*	*literal*
JNNR.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual add address of the jump destination.

Note: The D variant may be used with or without an operand.

[06]

JNOV

(jump if not overflow)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNOV causes a transfer of control to a specified location (program label or virtual address) if OV is not set.

Formats

JNOV	*location*
JNOV *et*	*literal*
JNOV.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[06]

JNP

(jump if not positive)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNP causes a transfer of control to a specified location (program label or virtual address) if the integer number in ACC is negative or zero.

Formats

JNP	*location*
JNP.*et*	*literal*
JNP.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination.

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[06]

JNPB

(jump if not positive, B)

Type:

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNPB causes a transfer of control to a specified location (program label or virtual address) if the value in B is negative or zero.

Formats

JNPB	*location*
JNPB.*et*	*literal*
JNPB.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[0E]

JNPD

(jump if not positive, decimal)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNPD causes a transfer of control to a specified location (program label or virtual address) if the decimal number in ACC is negative or zero.

Formats

JNPD	*location*
JNPD.*et*	*literal*
JNPD.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[06]

JNPR

(jump if not positive, real)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNPR causes a transfer of control to a specified location (program label or virtual address) if the real number in ACC is negative or zero.

Formats

JNPR	*location*
JNPR.*et*	*literal*
JNPR.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination.

Note: The D variant may be used with or without an operand.

[04]

JNR

(jump if negative, real)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNR causes a transfer of control to a specified location (program label or virtual address) if the real number in ACC is negative.

Formats

JNR	*location*
JNR.*et*	*literal*
JNR.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[06] JNZ

(jump if not zero)

Type

A pseuo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNZ causes a transfer of control to a specified location (program label or virtual address) if the integer number in ACC is not zero.

Formats

JNZ	*location*
JNZ.*et*	*literal*
JNZ.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[06]

JNZB

(jump if not zero, B)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNZB causes a transfer of control to a specified location (program label or virtual address) if the value in B is not zero.

Formats

JNZB	*location*
JNZB.*et*	*literal*
JNZB.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[06] JNZD

(jump if not zero, decimal)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNZD causes a transfer of control to a specified location (program label or virtual address) if the decimal number in ACC is not zero.

Formats

JNZD	*location*
JNZD.*et*	*literal*
JNZD.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[06]

JNZDL

(jump if not zero, descriptor length)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNZDL causes a transfer of control to a specified location (program label or virtual address) if the length field in DR is not zero.

Formats

JNZDL	*location*
JNZDL.*et*	*literal*
JNZDL.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[06]

JNZR

(jump if not zero, real)

Type

A pseudo-tertiary function which generates a JAF function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JNZR causes a transfer of control to a specified location (program label or virtual address) if the real number in ACC is not zero.

Formats

JNZR	*location*
JNZR.*et*	*literal*
JNZR.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[04]

JOV

(jump if overflow)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JOV causes a transfer of control to a specified location (program label or virtual address) if OV is set.

Formats

JOV	*location*
JOV.*et*	*literal*
JOV.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[04]

JP

(jump if positive)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JP causes a transfer of control to a specified location (program label or virtual address) if the integer number in ACC is greater than zero.

Formats

JP	*location*
JP.*et*	*literal*
JP.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[04]

JPB

(jump if positive, B)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JPB causes a transfer of control to a specified location (program label or virtual address) if B > 0.

Formats

JPB	*location*
JPB.et	*literal*
JPB.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JPR causes a transfer of control to a specified location (program label or virtual address) if the real number in ACC is greater than zero.

Formats

JPR	*location*
JPR.*et*	*literal*
JPR.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register seecified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[04]

JZ

(jump if zero)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JZ causes a transfer of control to a specified location (program label or virtual address) if the integer number in ACC is zero.

Formats

JZ	*location*
JZ.*et*	*literal*
JZ.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[04]

JZB

(jump if zero, B)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JZB causes a transfer of control to a specified location (program label or virtual address) if B is zero.

Formats

JZB	*location*
JZB.*et*	*literal*
JZB.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[04]

JZD

(jump if zero, decimal)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JZD causes a transfer of control to a specified location (program label or virtual address) if the decimal number in ACC is zero.

Formats

JZD	*location*
JZD.*et*	*literal*
JZD.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

[04]

JZDL

(jump if zero, descriptor length)

Type:

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JZDL causes a transfer of control to a specified location (program label or virtual address) if the length field of DR is zero.

Formats

JZDL	*location*
JZDL.*et*	*literal*
JZDL.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the address of the jump destination

Notes:

1 The D variant may be used with or without an operand

2 There is a possibility of a bound check error when jumping via DR. For example, use of the D or MD variants will cause an execution error if the jump occurs unless BCI is set (see section 1.4.6)

Type

A pseudo-tertiary function which generates a JAT function with the appropriate mask.

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Program errors

Operand addressing errors for jump functions, see section 11.2.1

Summary

JZR causes a transfer of control to a specified location (program label or virtual address) if the real number in ACC is zero.

Formats

JZR	*location*
JZR.*et*	*literal*
JZR.MD	

where

location is a program label or the name of a 32 bit local data item containing the virtual address of the jump destination

et is one of the variants D, L, X, P or S

literal is an arithmetic literal (see Chapter 5) which is used in conjunction with the register specified by the variant to access the location holding the virtual address of the jump destination

Note: The D variant may be used with or without an operand.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

ACC	Contains new value
OV	Cleared

Program errors

1 Operand addressing errors, see section 11.2.1

2 Non-zero bits of operand truncated (that is, ACS too small)

Summary

The value of the terminal operand is loaded into ACC. If operand length > ACS and non-zero bits are lost, a size interrupt occurs.

Formats

L	*oper*
L.*s*	*im*
L.*c*	
L.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand approrpiate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

B	Overwritten with new value
OV	Cleared

Program errors

Operand addressing errors, see section 11.2.1

Summary

The value of the terminal operand is loaded into B. The previous contents of B may be used as a modifier to access the terminal operand (that is, the M variant may be used).

Formats

LB	*oper*
LB.*s*	*im*
LB.*c*	
LB.*ep*	*intlit*

where

oper is the temrinal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effects on registers

None apart from CTB.

Program errors

Operand addressing errors (see section 11.2.1)

Summary

Bits 0 to 29 of the terminal operand are loaded into CTB.
Bits 30 and 31 are ignored.

Formats

LCT	*oper*
LCT.*s*	*im*
LCT.*c*	
LCT.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name).

im is an operand appropriate for indirection or modification.

initlit is an integer literal modifying the register specified by the explicit variant *ep*.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 64

Effect on registers

CC Indicates the type of descriptor loaded (for example, CC = 0 means a type 0 descriptor)

Program errors

Operand addressing errors, see section 11.2.1

Summary

The value of the terminal operand is loaded into DR, overwriting it. A descriptor in DR may be used to access the operand.

Formats

LD	*oper*
LD.*s*	*im*
LD.*c*	
LD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

DR See *Summary*

Program errors

Operand addressing errors, see section 11.2.1

Summary

The value of the terminal operand is loaded into the address field of DR (the less significant 32 bits). The remainder of DR is unchanged unless DR is used implicitly to access the terminal operand.

Formats

LDA	*oper*
LDA.*s*	*im*
LDA.*c*	
LDA.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Program errors

Operand addressing errors, see section 11.2.1

Summary

The least significant 24 bits of the terminal operand are copied to bits 8 to 31 of DR. The other bits of DR are unaltered unless indirect addressing is used (I variants), in which case they are overwritten by the corresponding bits of the descriptor used to access the terminal operand (address field unmodified). Bits 0 to 7 of the operand are ignored.

Formats

LDB	*oper*
LDB.*s*	*im*
LDB.*c*	
LDB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 64

Effect on registers

CC Set to indicate the type of descriptor loaded.
A value of *n* indicates a descriptor of type *n*; *n* = 0 to 3

Program errors

1 Operand addressing errors, see section 11.2.1

2 Invalid address forms

Summary

The terminal operand value is transferred to DR with the least significant 32 bits (the address field) augmented by the value of its own byte address. Carry out of the address field resulting from the addition is ignored.

A literal or register as terminal operand is not allowed.

Indirect access causes the operand rather than the descriptor used to access it to be left in DR.

Formats

LDRL	*oper*
LDRL.*s*	*im*
LDRL.*c*	
LDRL.*ep*	*intlit*

where

oper is the terminal operand (data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

(load type and bound)

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Program errors

Operand addressing errors, see section 11.2.1

Summary

The terminal operand value is copied to the more significant 32 bits of DR. The less significant 32 bits are unaltered unless indirect addressing is used to access the terminal operand (I variants) in which case they are replaced by the address field (unmodified) of the descriptor used.

Formats

LDTB	*oper*
LDTB.*s*	*im*
LDTB.*c*	
LDTB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

None other than LNB affected.

Program errors

1 Operand addressing errors, see section 11.2.1

2 Bits 0 to 13 of operand not equal to SSN

3 Bits 14 to 29 of operand ≥ SF

Summary

LLN loads bits 14 to 29 of the terminal operand into LNB as follows:

1 Bits 0 to 13 of the operand are checked to see if they are equal to SSN

2 The value of bits 14 to 29 of the operand is checked to see if it is < SF

3 If the above checks are successful, LNB is loaded with the new value

Bits 30 and 31 of the operand are ignored.

Formats

LLN	*oper*
LLN.*s*	*im*
LLN.*c*	
LLN.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 64

Effect on registers

ACC Overwritten with new value

ACS 64

OV Cleared

Program errors

Operand addressing errors, see section 11.2.1

Summary

ACS is set to 64 and the value of the terminal operand loaded into ACC.

If necessary, a literal operand is extended on the left to 64 bits, with sign regeneration, before loading.

Formats

LSD	*oper*
LSD.*s*	*im*
LSD.*c*	
LSD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 128

Effect on registers

ACC Overwritten with new value

ACS 128

OV Cleared

Program errors

Operand addressing errors

Summary

ACS is set to 128 and the value of the terminal operand loaded into ACC.

If necessary, a literal operand is extended on the left to 128 bits, with sign bit regeneration, before loading.

Formats

LSQ	*oper*
LSQ.*s*	*im*
LSQ.*c*	
LSQ.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

ACC	Overwritten with new value
ACS	32
OV	Cleared

Program errors

Operand addressing errors

Summary

ACS is set to 32 and the value of the terminal operand loaded into ACC.

Formats

LSS	*oper*
LSS.*s*	*im*
LSS.*c*	
LSS.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

LUH

(load upper half)

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Program errors

1 Operand addressing errors, see section 11.2.1

2 ACS = 128

Summary

ACS is doubled and the terminal operand value is loaded into the upper (newly created) half of ACC. The lower half of ACC is unchanged.

Formats

LUH	*oper*
LUH.*s*	*im*
LUH.*c*	
LUH.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

None apart from XNB

Program errors

Operand addressing errors

Summary

Bits 0 to 29 of the terminal operand are loaded into XNB.
Bits 30 and 31 are ignored.

Formats

LXN	*oper*
LXN.*s*	*im*
LXN.*c*	
LXN.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

DR Modified as described below

Program errors

1 Operand addressing errors, see section 11.2.1

2 Bound significant and ≤ terminal operand.

3 System call descriptor in DR

4 Invalid descriptor in DR

5 Indirect addressing

Summary

In general, the terminal operand is added to the address field of DR and subtracted from the bound/length field. Carry out of the address field is ignored and bits 0 to 7 of DR are unaltered. The precise effect of the function depends on the type of descriptor in DR, see *Description*.

Formats

MODD	*oper*
MODD.*s*	*im*
MODD.*c*	
MODD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Note: Indirect addressing is not permitted.

Description

The function executes as described above if DR contains a valid vector, string, descriptor or code descriptor, that is:

1 Type 0 with valid size code

2 Type 1 or 2

3 Type 3, subtype 32 or 33

If DR contains an escape descriptor, the escape mechanism is invoked and the required descriptor substituted in DR by an escape procedure before being modified.

A program error interrupt occurs if the descriptor in DR is one of the following:

1 Type 0 with an invalid size code

2 System call

3 Type 3 with an undefined subtype number

If the descriptor is of type 0 or 2 and the USC bit is not set, or of type 3, subtype 32 or 33, the terminal operand is scaled appropriately before addition to the address field. If the descriptor is type 0 with size code 0, the least significant 3 bits of the terminal operand are ignored as a result of the scaling operation.

If the terminal operand (regarded as unsigned, hence positive) is not less than the original contents of the bound/length field, only the least significant 24 bits of the difference are left in the bound/length field. In such cases, if the descriptor is type 0 or 2 with the BCI bit not set, or type 3, subtype 32, a maskable bound check error interrupt occurs. This does not apply to string descriptors.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

ACS, CC, PM — Modified as specified by terminal operand

Program errors

1 Operand addressing errors, see section 11.2.1

2 Attempting to set ACS to 0

Summary

The least significant 16 bits of the terminal operand are used to alter the program mask (PM), condition code (CC) and accumulator size (ACS) registers as follows:

Register affected	*Operand flag bit*	*Operand value bits*
ACS	27	30 and 31
CC	26	28 and 29
PM	24	16 to 23 (ones used)
PM	25	16 to 23 (zeros used)

Bits 0 to 15 of the terminal operand are ignored and may take any value.

Formats

MPSR	*oper*
MPSR.*s*	*im*
MPSR.*c*	
MPSR.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The bits described as *flag bits* in the table above specify the registers to be altered. The new value for the register is contained in the bits described as *operand value bits*.

The effects may be further described as follows:

1 ACS If bit 27 is 1, bits 30 and 31 are used to overwrite ACS. An error occurs if bits 30 and 31 are zero.

If bit 27 is 0, ACS is unaltered and bits 30 and 31 ignored

2 CC If bit 26 is 1, CC is set to the value in bits 28 and 29. Otherwise CC is unaltered and bits 28 and 29 may take any value

3 PM Two flag bits are used. If bit 24 is 1, bits of the program mask that correspond to ones in operand bits 16 to 23 are set to one, otherwise PM is unaltered.

If bit 25 is 1, bits of the program mask that correspond to zeros in operand bits 16 to 23 are set to zero, otherwise PM is unaltered

Note: ACS and CC may be set using a 7-bit unsigned literal operand, thus generating the 16-bit form of the function.

Type: Secondary, length 16 or 32 bits (see section 6.4.2

Permissible ACS: 64

Effect on registers

ACC, DR } The descriptors in these registers have their length and address fields updated as described below

Program errors

Failure of standard checks for store-to-store operations (see section 6.4.4

Summary

MV is a store-to-store function which moves a string of bytes from one location in store to another. Optional operands may be used to specify mask and filler bytes.

Formats

MV	
MV	*mask, filler*
MV.N	*ulit*
MV.N	*ulit, mask, filler*

where

mask and *filler* are one-byte constants (conventionally hexadecimal

ulit is an unsigned integer literal specifying the number of bytes to be moved

Description

The MV function causes a string of bytes to be copied from one store location to another. The string to be copied (the *source string*) is specified by a descriptor in ACC and the store location to receive the string (the *destination string*) is specified by a descriptor in DR. Both descriptors must be loaded into the registers before the MV function is executed. They may be either byte vector or string descriptors.

The source string is copied to the destination string, overwriting any data already present (unless the mask operand form is used, see below). The source string is unchanged. The number of bytes copied is determined by the operand *ulit* if present, otherwise by the length of the destination vector (DR).

The mask enables bits to be masked in the course of the move. Each bit of each byte of the destination string is compared to the corresponding bit of the mask byte. Only those bits of the destination byte which correspond to zeros in the mask byte are altered to the corresponding bits in the source byte (or the filler byte if the source string has been completely copied).

If the length of the source string is less than the number of bytes to be moved, the filler is used to fill the remaining bytes of the destination string until the count is satisfied.

For each byte moved the address field in each descriptor is increased by one and the length field is decreased by one.

If the number of bytes to be moved is specified as zero no error will result. No data will be moved and ACC and DR are not altered.

If the 16 bit form of the function is used, the number of bytes to be moved must be equal to or greater than the length of the destination string otherwise an error occurs.

The destination string must not overlap the source string on the right, see section 6.4.3. If it does the result of the move is undefined.

Otherwise, the fields may overlap in any way and the correct result is obtained.

The CHOV function may be used to check for overlap before using MV.

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Effect on registers

DR The descriptor in DR is updated during execution of the function

Program errors

Failure of standard checks for store-to-store operations, see section 6.4.4

Summary

Each byte of the string described by a descriptor in DR is overwritten by the *source byte* which is either

1 A literal operand (32 bit form) or

2 Bits 24 to 31 of the B register (16 bit form)

A mask byte may be provided either as an operand (32 bit form) or in bits 16 to 32 of the B register (16 bit form).

Formats

MVL	
MVL	*mask, literal*
MVL.N	*ulit*
MVL.N	*ulit, mask, literal*

where

mask and *literal* are one-byte literals (conventionally hexadecimal)

ulit is an unsigned literal specifying the number of bytes of the DR string to be overwritten

Description

When the operand *ulit* is present, it specifies the number of bytes of the DR string to be overwritten; when it is absent, all bytes are replaced.

The *mask* enables bits to be masked in the course of the move. Each bit of each byte of the DR string is compared to the corresponding bit of the mask byte. Only those bits of the DR byte which correspond to zeros in the mask byte are altered to the corresponding bits of the source byte.

If the number of bytes to be overwritten is specified as zero, no error occurs. A null operation is performed leaving DR unaltered.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

B Contains result

OV Cleared unless B overflow occurs

All other registers unchanged

Program errors

Operand addressing errors

B overflow

Summary

The contents of the operand are multiplied by the contents of B, both being treated as signed 32 bit integers. The least significant 32 bits of the product are left in B. If overflow occurs (product $<-2^{31}$ or $>2^{31}-1$) OV is set, otherwise OV is cleared.

Formats

MYB	*oper*
MYB.*s*	*im*
MYB.*c*	
MYB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

ACC Contains result

OV Cleared

All other registers are unchanged

Program errors

Operand addressing errors

ACS = 128

Summary

A logical not equivalent operation is performed between ACC and the operand. The result is placed in ACC.

The result of the operation is determined by the following table:

Original ACC bit	*Operand bit*	*Resultant ACC bit*
0	0	0
0	1	1
1	0	1
1	1	0

Formats

NEQ	*oper*
NEQ.*s*	*im*
NEQ.*c*	
NEQ.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Permissible ACS: 64

Effect on registers

ACC, DR } The descriptors in these registers have been the length and address fields updated.

Program errors

Failure of standard checks for store-to-store operations (see section 6.4.4

Summary

A descriptor in DR describes a string of bytes (the *DR string*). A logical NEQ operation is performed between each byte of this string and either

1 The filler byte (32 bit form), or

2 The corresponding byte of a string described by a descriptor in ACC (16 bit form)

Each byte of the DR string (up to the specified number) is replaced by the result of the comparison.

Formats

NEQS	*mask, filler*
NEQS.N	*ulit*
NEQS.N	*ulit, mask, filler*

where

mask is unused and must be a literal zero

filler is a one-byte literal (conventionally hexadecimal)

ulit is an integer literal specifying the number of bytes of the DR string to be operated upon

Description

A string descriptor must be loaded into DR (and, if necessary, ACC) before NEQS is executed.

In the case of the 16 bit form, the ACC string is used and must be of adequate length (that is, at least equal in length to the number of bytes to be compared).

In the 32 bit case, the specified number of bytes of the DR string is compared to the filler byte.

If the number of bytes to be compared is specified as zero, a null operation is performed and DR and ACC are unchanged.

For each byte compared, the DR (or ACC and DR) descriptors are updated by incrementing the address field and decrementing the length field.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64

Effect on registers

ACC Contains result

OV Cleared

All other registers are unchanged

Program errors

Operand addressing errors

ACS = 128

Summary

A logical OR operation is performed between ACC and the operand. The result is placed in ACC.

The result of the OR operation is determined by the following table:

Original ACC bit	*Operand bit*	*Resultant ACC bit*
0	0	0
0	1	1
1	0	1
1	1	1

Formats

OR	*oper*
OR.*s*	*im*
OR.*c*	
OR.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Permissible ACS: 64

Effect on registers

ACC, DR } The descriptors in these registers have the length and address fields updated.

Program errors

Failure of standard checks for store-to-store operations (see section 6.4.4

Summary

A descriptor in DR describes a string of bytes (the *DR string*). A logical OR operation is performed between each byte of this string and either

1 The filler byte (32 bit form), or

2 The corresponding byte of a string described by a descriptor in ACC (16 bit form)

Each byte of the DR string (up to the specified number) is replaced by the result of the comparison.

Formats

ORS	*mask, filler*
ORS.N	*ulit*
ORS.N	*ulit, mask, filler*

where

mask is unused and must be a literal zero

filler is a one-byte literal (conventionally hexadecimal)

ulit is an integer literal specifying the number of bytes of the DR string to be operated upon

Description

A string descriptor must be loaded into DR (and, if necessary, ACC) before ORS is executed.

In the case of the 16 bit form, the ACC string is used and must be of adequate length (that is, at least equal in length to the number of bytes to be compared).

In the 32 bit case, the specified number of bytes of the DR string is compared to the filler byte.

If the number of bytes to be compared is specified as zero, a null operation is performed and DR and ACC are unchanged.

For each byte compared, the DR (or ACC and DR) descriptors are updated by incrementing the address field and decrementing the length field.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Program errors

Operand addressing errors, see section 11.2.1

Summary

OUT causes a class 9 (out) interrupt to occur.
This is a stack switching interrupt. The value of the terminal operand is stored on the new stack as the interrupt parameter.

Formats

OUT	*oper*
OUT.*s*	*im*
OUT.*c*	
OUT.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Permissible ACS: 32, 64, 128

Effect on registers

ACC See below

OV Cleared unless overflow occurs out of ACC

Program errors

1 Failure of standard checks for store-to-store operations, see section 6.4.4

2 Decimal overflow (maskable)

Summary

This function converts zone/numeric format into packed decimal format leaving the result in ACC. DR should contain a descriptor to a string of bytes (the *DR string*) containing the unpacked numbers. For each byte of the DR string, the contents of ACC are shifted left by one decimal place and the least significant 4 bits of the DR string byte inserted in the space thereby created next to the sign digit in ACC.

Formats

PK	
PK.N	*ulit*

where

ulit is the number of DR string bytes to be packed (< 128)

Description

If *ulit* is absent, the number of bytes to be packed is specified by the length field of DR. The result is undefined if the number specified is ≥ 128.

If any non-zero bits are shifted out of ACC, OV is set and a maskable decimal overflow interrupt occurs. The sign digit in ACC is unaltered in all cases.

If the number of bytes to be packed is specified as zero, no error occurs, a null operation is performed and ACC and DR are not altered.

Type: Primary, length 16 (operand must be 7-bit literal)

Terminal operand length: 7

Effect on registers

SF points to new stack front.

Program errors

1 Incorrect operand type (must be 7-bit literal)

2 New SF ≤ LNB

3 Segment overflow

Summary

The stack is aligned to an odd-word boundary, the contents of LNB are stacked, and the operand is added to SF.

Formats

PRCL	*lit7*

where *lit7* is a 7-bit literal.

Description

1 If the address of stack front is an even number of words then SF is incremented by 1

2 The contents of LNB are expanded to form a 32-bit byte address (by concatenating the contents of SSN on the left with two zero bits on the right). Bit 31 is then set to 1 if SF was incremented (as in note 1) and the result stacked (note 3).

3 The action of adjust SF (ASF) is now followed as described earlier in this chapter (the operand is added to SF)

The effect of the PRCL function is:

1 To shorten the standard pre-call sequence:

```
PRCL   4   replaces   STLN.T
                      ASF  4
```

2 To align the programmer's local name space predicatbly, LNB+5 will always be on a long-word boundary if PRCL is used by the calling procedure. With this knowledge the

programmer can arrange his on-stack data so that double-word items are located on long-word boundaries and hence decrease the access time

F0 RAD

(real add)

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64,128

Effect on registers

OV Cleared unless floating-point overflow occurs.

Program errors

1 Operand addressing errors, see section 11.2.1

2 Floating-point overflow (maskable)

3 Floating-point underflow (maskable)

Summary

The value of the terminal operand is added to the contents of ACC and the normalised result left in ACC. The result is truncated (rounded towards zero).

Formats

RAD	*oper*
RAD.*s*	*im*
RAD.*c*	
RAD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

Floating-point underflow occurs if the result cannot be expressed in normalised form with a true exponent greater than -65. The result is made true zero and an interrupt occurs.

Floating-point overflow occurs when the normalised result requires a true exponent greater than 63. OV is set and an interrupt occurs. The result then has a normalised fraction and a characteristic 128 less than it should be.

The addition procedds as follows:

1 The fractional part of the number with the smaller characteristic is shifted down logically by the number of hexadecimal places which is the difference of the characteristics. The digit left in the position immediately to the right of that originally occupied by the least significant digit of the fraction is retained as *guard digit* (and the unshifted fraction is extended with a zero in the corresponding position). All the other digits shifted off are lost (effectively treated as zeros).

If the characteristics were equal, both fractions are extended with zero guard digits.

The above procedure is still carried out even if the number with the larger characteristic has a zero fraction (this will not occur with normalised operands)

2 The two signed fractions, including their guard digits, are added algebraically to form an intermediate sum in sign-and-modulus form. The intermediate sum has an associated sign bit, a possible carry bit and, including the guard digit, 7, 15 or 29 hexadecimal digits

3 The intermediate sum is normalised to generate the final result. The characteristic initially associated with the intermediate sum is the larger of the two original characteristics. Normalisation proceeds as follows:

(a) If all digits and the carry bit of the intermediate sum are zero, a true zero result is generated.

(b) If the carry bit is non-zero, the intermediate sum is shifted one hexadecimal place to the right (generating a 1 in the most significant hexadecimal digit position), and its carry bit and guard digit are removed to form the fractional part of the result. 1 is added to the characteristic (this may cause overflow to form the result characteristic. The sign bit of the result is that associated with the intermediate sum.

(c) If the carry bit is zero, but one or more digits of the intermediate sum are non-zero, the latter is shifted left until the most significant hexadecimal digit is non-zero. Following each hexadecimal shift, zero is inserted in the guard digit position and 1 is subtracted from the characteristic. Should this cause the characteristic to become negative, underflow occurs and a true zero result is generated. If the characteristic does not become negative, the result comprises the sign bit associated with the intermediate sum, the final characteristic and the normalised intermediate sum with the carry bit and guard digit removed.

(d) If ACS = 128, bits 64 to 71 of the result are generated as described in section 1.4.4.

RALN

(raise LNB)

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

None affected apart from LNB.

Program errors

1 Operand addressing errors, see section 11.2.1

2 Operand ≤ 0

3 Operand > (SF-LNB)

Summary

LNB is changed to the value of (SF-terminal operand). The operand is regarded as a number of words which must be less than the current value of SF (so operand bits 0 to 15 must be zero). The potential new value of LNB must not be less than the old. If either check fails, LNB is unchanged.

Formats

RALN	*oper*
RALN.*s*	*im*
RALN.*c*	
RALN.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

ACC	Unchanged	
CC	0	ACC = operand
	1	ACC < operand
	2	ACC > operand

Program errors

Operand addressing errors

Summary

The terminal operand is compared algebraically with ACC and CC set to indicate the outcome.

Formats

RCP	*oper*
RCP.*s*	*im*
RCP.*c*	
RCP.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

OV Cleared unless floating-point overflow occurs

Program errors

1 Operand addressing errors, see section 11.2.1

2 Zero divide

3 Floating-point underflow

4 Floating-point overflow

Summary

The contents of ACC are divided by the terminal operand and the normalised quotient left in ACC. If the divisor fraction is zero the result in ACC is undefined and a zero divide interrupt occurs (although OV is cleared).

Underflow or overflow may occur (see RAD) unless the dividend fraction is zero in which case no interrupt occurs and the result is a true zero.

Formats

RDV	*oper*
RDV.*s*	*im*
RDV.*c*	
RDV.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: Half ACS

Permissible ACS: 64, 128

Effect on registers

ACS	Halved
OV	Cleared unless floating-point overflow occurs

Program errors

1	Operand addressing errors, see section 11.2.1
2	Zero divide (maskable)
3	Floating-point underflow (maskable)
4	Floating-point overflow (maskable)
5	ACS = 32

Summary

The contents of ACC are divided by the terminal operand. ACS is halved and the normalised quotient, the size of which is the same as the new value of ACS, is left in ACC. The operation is otherwise the same as RDV.

Underflow or overflow may occur (see RAD) unless the dividend fraction is zero in which case no interrupt occurs and the result is a true zero.

Formats

RDVD	*oper*
RDVD.*s*	*im*
RDVD.*c*	
RDVD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64 or 128

Effect on registers

OV Cleared unless overflow occurs

Program errors

1 Operand addressing errors, see section 11.2.1

2 Floating-point underflow (maskable)

3 Floating-point overflow (maskable)

Summary

The normalised product of the terminal operand and the contents of ACC are left in ACC.

Overflow and underflow can occur as described for RAD but only if the final characteristic exceeds 127 or is negative. In the latter case a true zero result is generated.

Formats

RMY	*oper*
RMY.*s*	*im*
RMY.*c*	
RMY.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

If the fractional part of either the multiplier or the multiplicand is zero, a true zero result is obtained and neither overflow nor underflow occurs. If neither of the fractional parts is zero the fractional part of the result is that which would be produced as follows:

1 Forming the true, 64-bit product of the fractions

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32 or 64

Effect on registers

ACS	Doubled
OV	Cleared unless overflow occurs

Program errors

1	Operand addressing errors, see section 11.2.1
2	Floating-point underflow (maskable)
3	Floating-point overflow (maskable)
4	ACS = 128

Summary

The normalised, double-length product of the terminal operand and the contents of ACC are left in ACC. ACS is doubled.

Overflow and underflow can occur as described for RAD but only if the final characteristic exceeds 127 or is negative. In the latter case a true zero result is generated.

Except that the true product of the fractional part is not truncated and the result is thereby exact, this operation is otherwise identical to RMY.

Formats

RMYD	*oper*
RMYD.*s*	*im*
RMYD.*c*	
RMYD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32, 64

Effect on registers

ACC Shifted as described below

Program errors

1 Operand addressing errors, see section 11.2.1

2 ACS = 128

Summary

ROT causes a circular, left shift of the contents of ACC. The value of the terminal operand specifies the number of places to be shifted.

Formats

ROT	*oper*
ROT.*s*	*im*
ROT.*c*	
ROT.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The shift is circular and each bit shifted off bit 0 of ACC is re-inserted at the least significant end.

When ACS = 32, bits 0 to 26 of the terminal operand do not affect the result but may influence the time taken to execute the function. It is thus recommended that operand bits 25 and 26 should be the same as bit 27. Similarly when ACS = 64 bits 0 to 25 do not affect the result and bit 25 should be the same as bit 26.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

OV Cleared unless floating-point overflow occurs

Program errors

1 Operand addressing errors, see section 11.2.1

2 Zero divide (maskable)

3 Floating-point underflow (maskable)

4 Floating-point overflow (maskable)

Summary

The contents of the terminal operand are divided by ACC and the normalised quotient left in ACC. If the divisor fraction is zero the result in ACC is undefined and a zero divide interrupt occurs (although OV is cleared).

Underflow or overflow may occur (see RAD) unless the dividend fraction is zero in which case no interrupt occurs and the result is a true zero.

Formats

RRDV	*oper*
RRDV.*s*	*im*
RRDV.*c*	
RRDV.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

OV Cleared unless floating-point overflow occurs

Program errors

1 Operand addressing errors, see section 11.2.1

2 Floating-point overflow (maskable).

3 Floating-point underflow (maskable)

Summary

The value of ACC is subtracted from the terminal operand and the normalised result left in ACC. The result is truncated.

Formats

RRSB	*oper*
RRSB.*s*	*im*
RRSB.*c*	
RRSB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

Overflow and underflow can occur as described for RAD. The subtraction is performed by inverting the sign of the contents of ACC and adding (see RAD).

RRTC

(read real time clock)

Type: Primary, length 16 bits

Permissible ACS: 32, 64, 128

Effect on registers

ACS	64
ACC	Contains real time clock value
OV	Cleared

Program errors

Universal types

Summary

ACS is set to 64 and the real time clock registers (see section 1.1.2.10) are copied to ACC. RTCX is copied to the most significant half of ACC and RTCY into the least significant half.

Format

```
RRTC
```

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

OV Cleared unless floating-point overflow occurs

Program errors

1 Operand addressing errors, see section 11.2.1

2 Floating-point overflow (maskable)

3 Floating-point underflow (maskable)

Summary

The value of the terminal operand is subtracted from the contents of ACC and the normalised result left in ACC. The result is truncated.

Formats

RSB	*oper*
RSB.*s*	*im*
RSB.*c*	
RSB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

The subtraction is performed by inverting the sign bit of the terminal operand and following the rules for floating-point addition (see RAD).

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32, 64 or 128

Effect on registers

OV Cleared unless overflow occurs

Program errors

1 Operand addressing errors, see section 11.2.1

2 Floating-point overflow (maskable)

3 Floating-point underflow (maskable)

Summary

The least significant 8 bits of the terminal operand, treated as a signed integer (*i*), are added to the characteristic of the floating-point number in ACC, which is then normalised.

The contents of ACC are thus effectively multiplied by 16^i. RSC may be used with a zero terminal operand to normalise any floating-point number.

Formats

RSC	*oper*
RSC.*s*	*im*
RSC.*c*	
RSC.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Description

If the fractional part of the number in ACC is zero, a true zero result is generated. If the fraction is not zero and, after adding *i* and normalising, the characteristic exceeds 127, overflow occurs. Similarly, if the characteristic becomes negative, underflow occurs and the result is true zero.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

TS Contains result

OV Cleared unless B overflow occurs

All other registers unchanged

Program errors

Operand addressing errors

B overflow

Summary

The contents of the operand are subtracted from the contents of B, both being treated as signed 32 bit integers. The least significant 32 bits of the difference are left in B. If overflow occurs (difference $<-2^{31}$ or $>2^{31}-1$) OV is set, otherwise OV is cleared.

Formats

SBB	*oper*
SBB.*s*	*im*
SBB.*c*	
SBB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32, 64

Effect on registers

OV Cleared

Program errors

1 Operand addressing errors, see section 11.2.1

2 ACS = 128

Summary

The seven least significant bits of the terminal operand are interpreted as a signed integer, *i*. The least significant 32 bits of ACC are shifted logically *i* bits to the left ($i>0$) or right ($i<0$).

If ACS = 32, the function has the same effect as USH.

If ACS = 64, the more significant half of ACC is unaltered.

Zeros are inserted in the least or most significant bit position of the 32 bits shifted during the shift.

Formats

SHS	*oper*
SHS.*s*	*im*
SHS.*c*	
SHS.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

SHZ

(shift while zero)

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32, 64

Effect on registers

ACC Contents shifted as described below

OV Cleared

Program errors

1 Operand addressing errors, see section 11.2.1

2 Non-zero bits of stored item truncated

3 ACS = 128

Summary

The contents of ACC, if not zero, are shifted logically leftwards until bit 0 is one. The number of places shifted is stored in the terminal operand location (if ACC is zero then zero is stored).

Formats

SHZ	*oper*
SHZ.*s*	*im*
SHZ.*c*	
SHZ.*ep*	*intlit*

where

oper is the terminal operand (data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 64

Effect on registers

CC	1	CC originally 0 and descriptor stored or CC originally 1
	2	CC originally 2
	3	CC originally 3

Program errors

1 Operand addressing errors, see section 11.2.1

2 Non-zero bits of stored item truncated.

3 Indirect addressing

Summary

This function is for use with SUPK.

If CC = 0, a descriptor is created and stored in the terminal operand location and CC is set to 1. If CC is not zero, the function has no effect. The descriptor created has the following form:

Bits	*Value*
0, 1 (type)	1
2 to 7	011000
8 to 31 (length)	1
32 to 63 (address)	(contents of address field in DR-1)

Formats

SIG	*oper*
SIG..*c*	
SIG. *ep*	*inlit*
:	:

where

oper is the terminal operand (data item name).

inlit is an integer literal modifying the register specified by the explicit variant *ep*.

Note that indriect addressing is not permitted.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

ACC	Loaded with operand
OV	Cleared
SF	Incremented by ACS

Program errors

Operand addressing errors, see section 11.2.1

Summary

The contents of ACC are copied to an intermediate register. The terminal operand is loaded into ACC and the contents of the intermediate register stacked, causing SF to be incremented by ACS.

Formats

SL	*oper*
SL.*s*	*im*
SL.*c*	
SL.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Notes:

1 TOS may be used to access the terminal operand. Thus T, IT and MIT variants are permitted

2 SL.T results in the contents of ACC and TOS being interchanged

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effects on registers

B	Loaded with operand
OV	Cleared
SF	Incremented by 1

Program errors

Operand addressing errors, see section 11.2.1

Summary

The contents of B are copied to an intermediate register and the terminal operand is loaded into B. The contents of the intermediate register are stacked and SF incremented by 1.

Formats

SLB	*oper*
SLB.*s*	*im*
SLB.*c*	
SLB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Notes:

1 The original contents of B may be used as a modifier to access the terminal operand (M variants)

2 Variants T, IT, MIT may be used

Type: Primary, 16 or 32 bits, see section 6.3.5

Terminal operand length: 64

Effect on registers

CC Set to indicate type of descriptor loaded
(CC = *n* means descriptor type *n*; *n* = 0 to 3)

Program errors

Operand addressing errors, see section 11.2.1

Summary

The contents of DR are copied to an intermediate register. The operand is loaded into DR and the contents of the intermediate register are stacked, causing SF to be incremented by 2.

Formats

SLD	*oper*
SLD.*s*	*im*
SLD.*c*	
SLD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Notes:

1 DR may be used to access the terminal operand (the variants involving D)

2 The variants T, IT, MIT, DT may be used

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 64

Permissible ACS: 32, 64, 128

Effect on registers

ACC	Loaded with operand
ACS	Set to 64
OV	Cleared
SF	Incremented according to old value of ACS

Program errors

Operand addressing errors, see section 11.2.1

Summary

The contents of ACC are copied to an intermediate register. ACS is set to 64 and the terminal operand loaded into ACC. The contents of the intermediate register are then stacked causing SF to be incremented by the old value of ACS.

Formats

SLSD	*oper*
SLSD.*s*	*im*
SLSD.*c*	
SLSD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Notes:

1 TOS may be used to access the terminal operand. Thus the T, IT and MIT variants are permitted

2 SLSD.T results in the contents of ACC and TOS being interchanged

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 128

Permissible ACS: 32, 64, 128

Effect on registers

ACC	Loaded with operand
ACS	Set to 128
OV	Cleared
SF	Incremented according to old value of ACS

Program errors

Operand addressing errors, see section 11.2.1

Summary

The contents of ACC are copied to an intermediate register. ACS is set to 128 and the terminal operand loaded into ACC. The contents of the intermediate register are then stacked causing SF to be incremented by the old value of ACS.

Formats

SLSQ	*oper*
SLSQ.*s*	*im*
SLSQ.*c*	
SLSQ.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Notes:

1 TOS may be used to access the terminal operand. Thus the T, IT and MIT variants are permitted

2 SLSQ.T results in the contents of ACC and TOS being interchanged

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32, 64, 128

Effect on registers

ACC	Loaded with operand
ACS	Set to 32
OV	Cleared
SF	Incremented according to old value of ACS

Program errors

Operand addressing errors, see section 11.2.1

Summary

The contents of ACC are copied to an intermediate register. ACS is set to 32 and the terminal operand loaded into ACC. The contents of the intermediate register are then stacked causing SF to be incremented by the old value of ACS.

Formats

SLSS	*oper*
SLSS.*s*	*im*
SLSS.*c*	
SLSS.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification.

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Notes:

1 TOS may be used to access the terminal operand. Thus the T, IT and MIT variants are permitted

2 SLSS.T results in the contents of ACC and TOS being interchanged

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32, 64, 128

Effect on registers

None

Program errors

1 Operand addressing errors, see section 11.2.1

2 Non-zero bits of stored item truncated

Summary

The contents of ACC are stored in the terminal operand location. If the operand location is shorter than ACS and any more significant, non-zero bits are truncated then an interrupt occurs.

Formats

ST	*oper*
ST.*s*	*im*
ST.*c*	
ST.*ep*	*intlit*

where

oper is the terminal operand (data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

None

Program errors

1 Operand addressing errors, see section 11.2.1

2 Non-zero bits of stored item truncated

Summary

The contents of B are stored in the terminal operand location; they may be used as a modifier in accessing the operand.

Formats

STB	*oper*
STB.*s*	*im*
STB.*c*	
STB.*ep*	*intlit*

where

oper is the terminal operand (data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Note: The M variants may be used to access the terminal operand.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

None

Program errors

1 Operand addressing errors (see section 11.2.1)

2 Non-zero bits of stored item truncated

Summary

The contents of CTB are expanded to form a 32 bit byte address (by concatenating two zero bits on the right) and stored in the terminal operand location. A literal operand is not permitted.

Formats

STCT	*oper*
STCT.*s*	*im*
STCT.*c*	
STCT.*ep*	*intlit*

where

oper is the terminal operand (data item name).

im is the operand appropriate for indirection or modification.

intlit is an integer literal modifying the register specified by the explicit variant *ep*.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 64

Effect on registers

None

Program errors

1 Operand addressing errors, see section 11.2.1

2 Non-zero bits of stored item truncated

Summary

The contents of DR are stored in the terminal operand location. A literal operand and indirect addressing are not permitted.

Formats

STD	*oper*
STD.*s*	*im*
STD.*c*	
STD.*ep*	*intlit*

where

oper is the terminal operand (data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifyirg the register specified by the explicit variant *ep*

Note: I variants are not permitted.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

None

Program errors

1 Operand addressing errors, see section 11.2.1

2 Non-zero bits of stored item truncated

Summary

The contents of LNB are expanded to form a 32-bit byte address (by concatenating the contents of SSN on the left with two zero bits on the right) and stored in the terminal operand location. A literal operand is not permitted.

Formats

STLN	*oper*
STLN.*s*	*im*
STLN.*c*	
STLN.*ep*	*intlit*

where

oper is the terminal operand (data item name)

im is the operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

None

Program errors

1 Operand addressing errors, see section 11.2.1

2 Non-zero bits of stored item truncated

Summary

The contents of SF are expanded to form a 32-bit byte address (by concatenating the contents of SSN on the left and two zero bits on the right) and stored in the terminal operand location. A literal operand is not permitted. All variants (including T variants) may be used.

Formats

STSF	*oper*
STSF.*s*	*im*
STSF.*c*	
STSF.*ep*	*intlit*

where

oper is the terminal operand (data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: Half ACS

Permissible ACS: 64, 128

Effect on registers

ACS Halved

Program errors

1 Operand addressing errors

2 ACS = 32

2 Non-zero bits of stored items truncated

Summary

The contents of the more significant half of ACC are stored in the terminal operand location. ACS is halved. A literal operand is not permitted.

Formats

STUH	*oper*
STUH.*s*	*im*
STUH.*c*	
STUH.*ep*	*intlit*

where

oper is the terminal operand (data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

None

Program errors

1 Operand addressing errors (see section 11.2.1)

2 Non-zero bits of stored item truncated

Summary

The contents of XNB are expanded to form a 32 bit byte address (by concatenating two zero bits on the right) and stored in the terminal operand location. A literal operand is not permitted.

Formats

STXN	*oper*
STXN.*s*	*im*
STXN.*c*	
STXN.*ep*	*intlit*

where

oper is the terminal operand (data item name).

im is the operand appropriate for indirection or modification.

intlit is an integer literal modifying the register specified by explicit variant *ep*.

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Permissible ACS: 32, 64, 128

Effect on registers

ACC Contents unpacked and shifted (see below)

ACS Unchanged

CC
- 0 CC was 0 and all unpacked digits were zeros
- 1 CC was 1 and all unpacked digits were zeros
- 2 CC was 2 or some non-zero digits unpacked. Sign positive
- 3 CC was 3 or some non-zero digits unpacked. Sign negative

OV Cleared

SF May be incremented by 2 (see *Description*)

Other registers are unchanged

Summary

SUPK converts a packed decimal number in ACC to zone/numeric format, each digit generating one byte, and inserts the resulting bytes into a string described by a descriptor in DR.

If CC was set to zero before execution of SUPK, non-significant zeros are replaced by a specified literal byte (usually a space character). If this substitution is not required, CC should be set to 1. No sign digit is inserted in the unpacked string, but CC is set to indicate the sign of the packed value.

Formats

SUPK	
SUPK	*mask, literal*
SUPK.N	*ulit*
SUPK.N	*ulit, mask, literal*

where

literal is a one-byte literal used to repalce non-signficant zeros

mask is unused by the function and must be a zero literal

ulit is an unsigned integer literal specifying the number of digits to be unpacked (<128)

Description

Before using SUPK, a string descriptor should be loaded into DR and CC should be set to zero or one.

If the 16 bit form of the function is used, a substitute character should be loaded into bits 24 to 31 of the B register.

The operation of the function is as follows:

If CC was set to zero, non-significant zeros are replaced by the specified literal, otherwise all zeros are treated as significant. Significant digits are prefixed by the appropriate zone codes. The resulting byte is stored in the address pointed to by the descriptor in DR. The contents of ACC are shifted up one decimal place (except for the sign digit). DR is updated to point to the next byte of the string and the process is repeated until

1 The DR string is filled, or

2 *ulit* digits have been unpacked

The literal value used to replace non-significant xeros is either

1 Contained in bits 24 to 31 of the B register (16 bit form), or

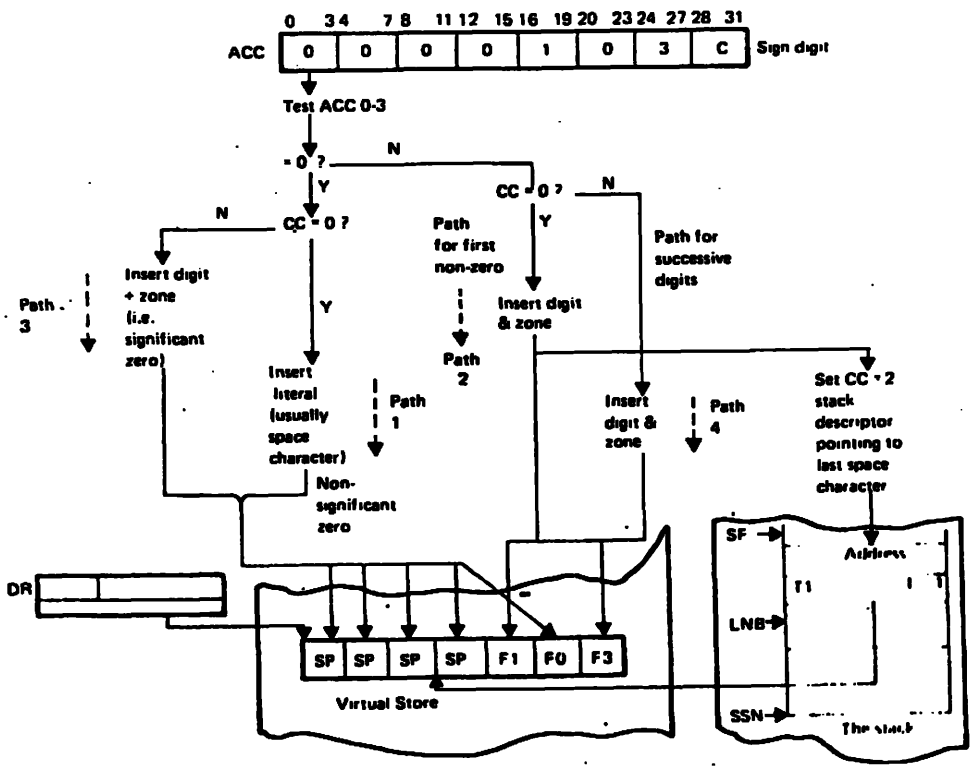
2 The operand *literal* (32 bit form)

The zone code used is determined by the mode bit of the system status register (SSR).

If the mode bit is one, the ISO code 3 is used; if it is zero, the EBCDIC code F is used.

If substitution of the literal value for zeros takes place and the number contains any significant digits, a descriptor is stacked pointing to the last literal value stored.

The operation of SUPK is summarised in the diagram below.



Type: Secondary, length 16 or 32 bits, see section 6.4.2

Effect on registers

CC 0 Specified number of bytes compared without finding inequality, or no scan performed

2 DR string byte > reference byte

3 DR string byte < reference byte

DR Left pointing to last byte checked

Program errors

Failure of standard checks for store-to-store operations, see section 6.4.4

Summary

DR should be loaded with a descriptor pointing to a string of bytes. The string is scanned left to right comparing each byte with a *reference byte* until either the end of the string is reached or a byte is found which is not identical to the reference byte. The reference byte can be

1 An operand (32 bit form)

2 Bits 24 to 31 of the B register (16 bit form)

A mask may be used.

Formats

SWEQ	
SWEQ	*mask, literal*
SWEQ.N	*ulit*
SWEQ.N	*ulit, mask, literal*

where

mask is a one-byte literal (conventionally hexadecimal)

literal is a one byte literal (the reference byte)

ulit is an unsigned integer literal specifying the number of bytes to be scanned

Description

When the operand *ulit* is present it specifies the number of DR string bytes to be checked; otherwise the length field in DR is used.

DR is updated after each byte is compared; when an inequality is found execution terminates leaving DR pointing to the first unequal byte.

If the mask is present, only those bits of each DR string byte that correspond to zeros in the mask byte are checked against the reference byte. When the 16 bit form is used, the mask should be loaded in bits 16 to 23 of B.

If the number of bytes to be scanned is specified as zero, no error occurs, a null operation is performed leaving DR unaltered and setting CC to 0.

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Effect on registers:

CC 0 Specified number of bytes compared without finding equality, or no scan performed

1 DR string byte equal to reference byte found

DR Left pointing to last byte checked

Program errors

Failure of standard checks for store-to-store operations, see section 6.4.4

Summary

DR should be loaded with a descriptor pointing to a string of bytes. The string is scanned left to right comparing each byte with a *reference byte* until either the end of the string is reached or a byte is found which is identical to the reference byte. The reference byte can be

1 An operand (32 bit form)

2 Bits 24 to 31 of the B register (16 bit form)

A mask may be used.

Formats

SWNE	
SWNE.	*mask, literal*
SWNE.N	*ulit*
SWNE.N	*ulit, mask, literal*

where

mask is a one-byte literal (conventionally hexadecimal)

literal is a one-byte literal (the reference byte)

ulit is an unsigned integer literal specifying the number of bytes to be scanned

Description

When the operand *ulit* is present it specifies the number of DR string bytes to be checked; otherwise the length field in DR is used.

DR is updated after each byte is compared; if a byte is found which is identical to the reference byte, execution terminates leaving DR pointing to that byte.

If the mask is present, only those bits of each DR string byte that correspond to zeros in the mask byte are checked against the reference byte. When the 16 bit form is used, the mask should be loaded in bits 16 to 32 of B.

If the number of bytes to be scanned is specified as zero, no error occurs, a null operation is performed leaving DR unaltered and setting CC to 0.

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Permissible ACS: 64

Effect on registers

CC	0	Specified number of characters in DR string checked and found valid, or no characters checked
	1	An invalid character found
DR		Updated during execution of function

Program errors

1	Failure of standard checks for store-to-store functions, see section 6.4.4
2	DR string bound ≥ bound field of ACC descriptor

Summary

ACC should contain a vector descriptor with size code 0 pointing to the start of a table of check bits. DR should contain a string or byte vector descriptor pointing to a string of characters (the *DR string*). The specified number of bytes of the DR string are checked against the bit table to ascertain if they contain valid characters.

Formats

TCH TCH.N	*ulit*

where

ulit is an unsigned integer literal specifying the number of bytes of the DR string to be checked.

Description

If the operand *ulit* is present it specifies the number of bytes to be checked, otherwise the length field of the descriptor in DR is used.

The most significant 5 bits of the DR string byte are used to modify the address in the ACC descriptor to access a byte in the check-bit table. The 3 least significant bits of the byte are used to locate one of the bits (numbered 0 to 7) in that byte.

TCH

A bound check interrupt occurs if the value used as modifier is not less than the bound field of the ACC descriptor.

Checking of the DR string continues until either the specified number of bytes have been checked (CC set to 0) or a byte is found whose check bit is 1 (CC set to 1). The DR descriptor is left pointing to the last byte checked.

The ACC descriptor and the DR string are unaltered.

If the number of bytes to be checked is specified as zero, no error occurs, a null operation is performed leaving ACC and DR unaltered, and CC is set to 0.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

CC 0 Original value of operand = 0

1 Original value of operand > 0

2 Original value of operand < -1

3 Original value of operand = -1

OV Unaltered (overflow is ignored)

Other registers are unchanged

Program errors

1 Operand addressing errors, see section 11.2.1

2 Incorrect descriptor type

3 Incorrect operand types

Summary

TDEC is a semaphore instruction used to provide interlocks on the peripheral communications areas in store. Between reading the original operand value and replacing it by the new value, access to the operand location is prevented by hardware.

CC is set to indicate the original value of the terminal operand and 1 is then subtracted from the operand. The original value of the operand is left in ACC.

Formats

TDEC	*oper*
TDEC.*s*	*im*
TDEC.*c*	
TDEC.*ep*	*intlit*

where

oper is the terminal operand (data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Note that the terminal operand location must be in store, not a register. The variants B, T, and P are not permitted.

Description

The prime use of this instruction is to implement semaphores, using semaphore descriptors.

The operand value is normally interpreted as the number of other processes waiting to use a shared resource. A value of -1 indicates that the resource is available.

The following restrictions apply to operand access:

1 Access is forced by hardware to bypass slave storage

2 If the operand is accessed indirectly, only a vector descriptor (type 0) with size 32 or a semaphore (type 3, subtype 40,41) descriptor

3 The operand must be located in store rather than a register. Direct TOS and (PC+N) operand forms are not permitted

If slave storage is used in the processor, it is cleared as follows:

1 Stack slave store is cleared if the stack segment is marked as non-slaved in its segment table entry

2 Operand slave store is cleared of items marked non-slaved in either segment table

Type: Secondary, length 16 or 32 bits, see section 6.4.2

Permissible ACS: 64

Effect on registers

DR — Contains a descriptor which is updated during execution of the function

Program errors

1 — Failure of standard checks for store-to-store functions, see section 6.4.4

2 — Value of DR string byte ≥ bound field of ACC descriptor

Summary

TTR is useful for code conversion. ACC should contain a vector descriptor (type 0, size code 8) pointing to a *translation table*. The string to be translated is pointed to by a descriptor in DR. Each byte in the DR string is replaced by a translation table byte using the value of the DR byte as a modifier to index the table.

Formats

TTR TTR.N	*ulit*

where

ulit is an unsigned integer literal specifying the number of bytes of the DR string to be translated.

Description

If the operand *ulit* is present, it specifies the number of bytes to be translated. If it is absent, all the bytes of the DR string are translated.

ACC is unaltered. If the number of bytes to be translated is zero, no error occurs, a null operation is performed and DR is unaltered.

To illustrate the execution of the function, if the byte address in ACC is *a* and the value of the DR string byte to be translated is *n*, the DR string byte is replaced by the contents of byte

location *a* + *n*.

A bound check interrupt occurs if the value of any DR string byte is not less than the bound field of the ACC descriptor.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32

Effect on registers

CC	0	No carry
	1	Carry
OV	Cleared	

Program errors

1	Operand addressing errors, see section 11.2.1
2	ACS = 64 or 128

Summary

The terminal operand and the contents of ACC are added and the sum left in ACC. CC is set to indicate whether or not carry occurred out of ACC bit 0.

Formats

UAD	*oper*
UAD.*s*	*im*
UAD.*c*	
UAD.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: ACS

Permissible ACS: 32 or 64

Effect on registers

CC	0	ACC = operand
	1	ACC < operand
	2	ACC > operand

Program errors

1	Operand addressing errors, see section 11.2.1
2	ACS = 128

Summary

The terminal operand is compared with the contents of ACC and CC set to indicate the result.

Formats

UCP	*oper*
UCP.*s*	*im*
UCP.*c*	
UCP.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32

Effect on registers

CC 0 No carry (indicates borrow into bit 0)

1 Carry (indicates no borrow, including the case where complementing causes carry, that is the value of ACC is zero.

Program errors

1 Operand addressing errors, see section 11.2.1

2 ACS = 64 or 128

Summary

The terminal operand and the contents of ACC are regarded as signed integers. The twos complement of the contents of ACC are added to the terminal operand and the result left in ACC.

Formats

URSB	*oper*
URSB.*s*	*im*
URSB.*c*	
URSB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modificatio

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32

Effect on registers

CC 0 No carry (indicates borrow into bit 0)

1 Carry (indicates no borrow, including the case where complementing causes carry, that is the operand value is zero)

Program errors

1 Operand addressing errors, see section 11.2.1

2 ACS = 64 or 128

Summary

The terminal operand and the contents of ACC are regarded as signed integers. The twos complement of the operand is added to ACC and the result left in ACC.

Formats

USB	*oper*
USB.*s*	*im*
USB.*c*	
USB.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Permissible ACS: 32, 64

Effect on registers

OV Cleared

Program errors

1 Operand addressing errors, see section 11.2.1

2 ACS = 128

Summary

The seven least significant bits of the terminal operand are interpreted as a signed integer, *i*. The contents of ACC are shifted *i* bits to the left (*i*>0) or right (*i*<0). Zeros are inserted in the least or most significant end of ACC (as appropriate) during the shift.

Formats

USH	*oper*
USH.*s*	*im*
USH.*c*	
USH.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers

CC
- 0 Read and write access permitted at specified level
- 1 Read access permitted, write access inhibited
- 2 Write access permitted, read access inhibited
- 3 Descriptor invalid, or
  Field crosses segment boundary, or
  Neither read not write access permitted (may mean segment number invalid)

Program errors

Operand addressing errors, see section 11.2.1

Summary

The VAL function is used to check whether a descriptor supplied to a procedure (invoked by CALL) is valid at the ACR level of the calling process. CC is set to indicate the result of the check. The descriptor should be loaded into DR for validation. Bits 8 to 11 of the operand gives the ACR of the caller.

Formats

VAL	*oper*
VAL.*s*	*im*
VAL.*c*	
VAL.*ep*	*intlit*

where

oper is the terminal operand (literal or data item name)

im is an operand appropriate for indirection or modification

intlit is an integer literal modifying the register specified by the explicit variant *ep*

Note: Indirect addressing using I variants is not permitted.

Description

The descriptor is assumed to be type 0, 1 or 2. A type 0 or 2 descriptor is assumed to be bounded.

If the descriptor is invalid, CC is set to 3 and execution terminates. An invalid descriptor is:

VAL

1 Type 0 or 2 with BCI set

2 Type 3

3 Type 0 with invalid sizecode

4 Type 0, 1 or 2 with zero bound/length

If the descriptor is valid, the address of the last word or byte in the field described is calculated, without altering DR, from the address of the first byte as follows:

1 TYPE 0:

(a) Add (bound-1), scaled if USC = 0

(b) If size = 64 add 4 bytes

(c) If size = 128 add 12 bytes (word alignment assumed)

2 TYPE 1 Add (length-1)

3 TYPE 2 As type 0 with size 64 bits

CC is set to 3 if the calculated address:

1 Has a different segment number to the initial address

2 Has the same segment number but lies beyond the upper limit of that segment

3 Has the same segment number as SSN but is not less than SSN+LNB

In case 3, if the address is less than SSN+LNB and the initial address is also in the stack segment, CC is set to 0.

Note: The second word of the segment table entry is ignored.

Type: Primary, length 16 or 32 bits, see section 6.3.5

Terminal operand length: 32

Effect on registers:

B	Contains calculated modifier
DR	Descriptor updated as described below
OV	Cleared

Program errors

1	Operand addressing errors
2	Incorrect type and size code of descriptor
3	Maskable bound check interrupt

Summary

This function calculates the modifier required to access an element of an array from the subscript supplied as the terminal operand and leaves it in the B register.

DR should contain a type 0 descriptor with a size code of 32 bits with the address field pointing to the first word of the dope vector for the array (or of the dimension currently being accessed if the array is multi-dimensional).

Formats

VMY	*oper*
VMY.*s*	*im*
VMY.*c*	
VMY.*ep*	*intlit*

where

oper	is the terminal operand (literal or data item name)
im	is an operand appropriate for indirection or modification
intlit	is an integer literal modifying the register specified by the explicit variant *ep*

Note: I variants are not permitted.

Description

The modifier is evaluated using the expression

$$(i-x)y$$

where

i is the subscript being accessed (supplied as the terminal operand of the function)

x and y are the first two words of the dope vector (x = lower bound of current dimension, y = number of elements in previous dimensions)

The modifier is checked to be $<z$, the third value of the dope vector (the number of elements up to the end of the dimension, including previous dimensions). The least significant 32 bits of the result are left in B.

As each of x, y and z are accessed the address in DR is incremented by 4 and the bound decreased by 1 (this may cause a bound check interrupt). Thus after execution of the function the address will be increased by 12 and the bound decreased by 3.

The modifier for a multi-dimensional array is calculated by evaluating the sum of the modifier for each dimension. Thus for an n dimensional array the modifier is

$$(i_1-x_1)y + (i_2-x_2)y + \ldots (i_n-x_n)y_n$$

A maskable bound check interrupt occurs:

1 If bound checks on x, y or z fail

2 If one of the following conditions is true:

(a) $i<x$

(b) $i-x \geq 2^{31}$

(c) $y<0$

(d) $z<0$

(e) $(i-x)y \geq z$

(f) $(i-x)y \geq 2^{31}$

Bound check interrupts are described in section 11.1.