**CSCI 360 MVCL, CLCL & EX Instructions**

**Move Character Long**

This instruction behaves like MVC but it can be used to move fields as long as 224-1 bytes long.

Format: label MVCL R1,R2

Each of the specified registers must be an even register that is used to designate an even-odd pair of registers. The contents of the field specified by R2 are moved to the field specified by R1.

**R1 EVEN register** - the last 3 bytes specify the address of the destination field

**R1 ODD register** - the last 3 bytes specify the length of the destination field

**R2 EVEN register** - the last 3 bytes specify the address of the source field

**R2 ODD register** - the last 3 bytes specify the length of the source field. The first byte is used as a pad character if the source field is shorter than the destination field.

If a byte is simultaneously in both the sending and receiving fields and if execution of the instruction causes the byte to receive a new value before the original contents are moved, then ***destructive overlap*** exists.

If destructive overlap occurs, no data is moved and the condition code is set.

Code Meaning

0 The source and destination field are the same length

1 The source field is larger

2 The destination field is larger

3 Destructive overlap

After execution of the instruction, the values in each of the 4 registers will be changed to:

**R1 EVEN register** is set to the address of the first byte past the end of the destination field

**R1 ODD register** is set to 0

**R2 EVEN register** is set to the address of the first byte beyond the last byte moved from the source field

**R2 ODD register** is decremented by the number of bytes moved from the source field.

Suppose that R2 contains the address of a field and that R3 contains

the length of that field. To set each byte of the field to X'00', the

following code can be used:

LR R4,R2 set the destination field address

SR R5,R5 set the pad character to X'00' and length to 0

MVCL R2,R4 puts the pad character of X'00' in the field

**Compare Character Long**

This instruction behaves like CLC but it can be used to compare fields as long as 224-1 bytes long.

Format: label CLCL R1,R2

Each of the specified registers must be an even register that is used to designate an even-odd pair of registers. The contents of the field specified by R2 are compared against the field specified by R1.

**R1 EVEN register** - the last 3 bytes specify the address of the destination field

**R1 ODD register** - the last 3 bytes specify the length of the destination field

**R2 EVEN register** - the last 3 bytes specify the address of the source field

**R2 ODD register** - the last 3 bytes specify the length of the source field. The first byte is used as a pad character. If the fields are of unequal length, the pad character is used to extend the shorter field for the comparison.

The condition code is set:

Code Meaning

0 Fields are equal

1 Field specified by R1 is low

2 Field specified by R2 is low

The comparison occurs from left to right, one byte at a time

When two bytes are EQUAL, the even registers are incremented by 1 and the odd registers are decremented by 1. The only exception to this rule occurs when the one of the fields has been exhausted and the comparison is occuring with the pad character. In this case, only the register pair associated with the non-exhausted field are altered.

**Execute**

This instruction is used to execute other operations when the length of a field cannot be determined until execution time.

Format: label EX R,D(X,B)

A copy of the instruction located at D(X,B) is made. The second byte of that copy is replaced by the rightmost byte of R. The copy of the instruction is then executed.

Neither the value in R nor the instruction at D(X,B) is changed.

If R is register 0, then the second byte is of the instruction is NOT changed, but it is still executed.

The byte to be replaced should initially contain X'00' if both hex digits are to be altered or X'0' in the place of the digit to be replaced.

Suppose that register 3 contains the address of a field that contains

a zoned decimal number and that register 7 contains the length of the

field. To pack the number into the field PACKNUM:

BCTR R7,0 decrement the length by 1

EX R7,PACKIT pack the number

In storage:

PACKNUM DS PL8

PACKIT PACK PACKNUM,0(0,R3)

The 0 within the parenthesis will be replaced at runtime by the

rightmost hex digit of register 7.

Suppose that we want to add two packed decimal fields. The address of

one field is in R3, while the length of that field is in R8. The

address of the other field is in R4, while its length is in R9.

We want to add the value pointed to by R4 to the value pointed to by

R3.

We need to get both lengths into the rightmost byte of one register so

that both lengths can be replaced at runtime.

R8 = 00 00 00 07 R9 = 00 00 00 04

BCTR R8,0 R8 now contains 00 00 00 06

SLL R8,4 R8 now contains 00 00 00 60

\*

BCTR R9,0 R9 now contains 00 00 00 03

\*

AR R8,R9 R8 now contains 00 00 00 63

\*

EX R8,ADD will produce FA63 3000 7000

\*

\* when decoded: AP 0(7,R3),0(4,R7)

In storage:

ADD AP 0(0,R3),0(0,R7) initially is FA00 3000 7000

Suppose that register 6 contains the address of a byte of storage that

contains an unknown 8 bit number and that we want to test to see which

bits are "on". A loop with an EXecuted TM instruction will be used.

If a bit is "on", print the word "ON". If the bit is "off", print the

word "OFF".

LA R3,8 there are 8 bits to test

LA R4,X'80' start with bit 0

EXLOOP EX R4,TEST test the bit

BZ BITOFF if bit is 0, branch to print "OFF"

XPRNT =C' ON',3 print "ON"

B NEXTBIT

BITOFF XPRNT =C' OFF',4 print "OFF"

NEXTBIT SRL R4,1 Switch to next bit to test

BCT R3,EXLOOP Repeat for all 8 bits

In storage:

TEST TM 0(R6),**0** Object code = 91**00** 6000

Loop Pass EX instruction produces

1 91**80** 6000

2 91**40** 6000

3 91**20** 6000

4 91**10** 6000

5 91**08** 6000

6 91**04** 6000

7 91**02** 6000

8 91**01** 6000