370 Instructions

Problem-State

Table of Contents

L, Load
LA, Load Address
LCR, Load Complement Registers
LH, Load Halfword
LM, Load Multiples
LNR, Load Negative Registers
LPR, Load Positive Registers
LR, Load Register
LTR, Load and Test Register
M, Multiply
MH, Multiply Halfword
MP, Multiply Packed (Decimal)
MR, Multiply Registers
MVC, Move Characters
MVCIN, Move Characters Inverse
MVCL, Move Characters Long
MVI, Move Immediate
MVN, Move Numerics
MVO, Move with Offset
MVZ, Move Zones
N, And
NC, And Characters
NI, And Immediate
NR, And Registers
O, Or
OC, Or Characters
OI, Or Immediate
OR, Or Registers
PACK, Pack
S, Subtract
SH, Subtract Halfword
SL, Subtract Logical
SLA, Shift Left Single
SLDA, Shift Left Double
SLDL, Shift Left Double Logical
SLL, Shift Left Single Logical
SLR, Subtract Logical Registers
SP, Subtract Packed (Decimal)
SR, Subtract Registers
SRA, Shift Right Single
SRDA, Shift Right Double
SRDL, Shift Right Double Logical
SRL, Shift Right Single Logical
SRP, Shift and Round Decimal
ST, Store
STC, Store Character
STCM, Store Characters under Mask
STH, Store Halfword
STM, Store Multiples
SVC, Supervisor Call
TM, Test under Mask

Introduction

This document is intended to be used as a quick reference for the mainframe, problem-state, non-floating point instructions.

The source code for a sample program that executes each of the problem-state, non-floating point instructions provides additional detail. This information is available via an Internet Connection or Local Access.

List by Mnemonic Opcode

The following list is sequenced by the Mnemonic Opcode.

Instruction	Mnemonic	Hex	Format
Add	A	5A	R1,D2(X2,B2)
Add Halfword	AH	4A	R1,D2(X2,B2)
Add Logical	AL	5E	R1,D2(X2,B2)
Add Logical Registers	ALR	1E	R1,R2
Add Packed (Decimal)	AP	FA	D1(L1,B1),D2(L2,B2)
Add Registers	AR	1A	R1,R2
Branch and Link	BAL	45	R1,D2(X2,B2)
Branch and Link Register	BALR	05	R1,R2
Branch and Save	BAS	4D	R1,D2(X2,B2)
Branch and Save Register	BASR	0D	R1,R2
Branch, Save and Set Mode	BASSM	0C	R1,R2
Branch on Condition	BC	47	M1,D2(X2,B2)
Branch on Condition Register	BCR	07	M1,R2

Branch on Count	BCT	46	R1,D2((X2,B2)
Branch on Count Register	BCTR	06	R1,R2
Branch and Set Mode	BSM	0B	R1,R2
Branch on Index High	BXH	86	R1,R3,D2(B2)
Branch on Index Low/Equal	BXLE	87	R1,R3,D2(B2)
Compare	C	59	R1,D2(X2,B2)
Compare Double and Swap	CDS	BB	R1,R3,D2(B2)
Compare Halfword	CH	49	R1,D2(X2,B2)
Compare Logical	CL	55	R1,D2(X2,B2)
Compare Logical Characters	CLC	D5	D1(L,B1),D2(B2)
Compare Logical Characters Long	CLCL	0F	R1,R2
Compare Logical Immediate	CLI	95	D1(B1),I2
Compare Logical under Mask	CLM	BD	R1,M3,D2(B2)
Compare Logical Registers	CLR	15	R1,R2
Compare Packed (Decimal)	CP	F9	D1(L1,B1),D2(L2,B2)
Compare Registers	CR	19	R1,R2
Compare and Swap	CS	BA	R1,R3,D2(B2)
Convert to Binary	CVB	4F	R1,D2((X2,B2)
Convert to Decimal	CVD	4E	R1,D2((X2,B2)
Divide	D	5D	R1,D2((X2,B2)
Divide Packed (Decimal)	DP	FD	D1(L1,B1),D2(L2,B2)
Divide Registers	DR	1D	R1,R2
Edit	ED	DE	D1(L1,B1),D2(B2)
Edit and Mark	EDMK	DF	D1(L1,B1),D2(B2)
Execute	EX	44	R1,D2(X2,B2)
Insert Character	IC	43	R1,D2(X2,B2)
Insert Character under Mask	ICM	BF	R1,M3,D2(B2)
Load	L	58	R1,D2(X2,B2)
Load Address	LA	41	R1,D2(X2,B2)
Load Complement Registers	LCR	13	R1,R2
Load Halfword	LH	48	R1,D2(X2,B2)
Load Multiple	LM	98	R1,R3,D2(B2)
Load Negative	LNR	11	R1,R2
Load Postive	LPR	10	R1,R2
Load Register	LR	18	R1,R2
Load and Test Register	LTR	12	R1,R2
Multipy	M	5C	R1,D2(X2,B2)
Multipy Halfword	MH	4C	R1,D2(X2,B2)
Multipy Packed (Decimal)	MP	FC	D1(L1,B1),D2(L2,B2)
Multipy Registers	MR	1C	R1,R2
Move Characters	MVC	D2	D1(L,B1),D2(B2)
Move Inverse	MVCIN	E8	D1(L,B1),D2(B2)
Move Characters Long	MVCL	0E	R1,R2
Move Immediate	MVI	92	D1(B1),I2
Move Numerics	MVN	D1	D1(L,B1),D2(B2)
Move with Offset	MVO	F1	D1(L1,B1),D2(L2,B2)
Move Zones	MVZ	D3	D1(L,B1),D2(B2)
aNd	N	54	R1,D2(X2,B2)
aNd Characters	NC	D4	D1(L,B1),D2(B2)
aNd Immediate	NI	94	D1(B1),I2
aNd Registers	NR	14	R1,R2
Or	O	56	R1,D2(X2,B2)
Or Characters	OC	D6	D1(L,B1),D2(B2)
Or Immediate	OI	96	D1(B1),I2
Or Registers	OR	16	R1,R2
Pack	PACK	F2	D1(L1,B1),D2(L2,B2)
Subtract	S	5B	R1,D2(X2,B2)

Subtract Halfword	SH	4B	R1,D2(X2,B2)
Subtract Logical	SL	5F	R1,D2(X2,B2)
Shift Left Single	SLA	8B	R1,D2(B2)
Shift Left Double	SLDA	8F	R1,D2(B2)
Shift Left Double Logical	SLDL	8D	R1,D2(B2)
Shift Left Single Logical	SLL	89	R1,D2(B2)
Subtract Logical Registers	SLR	1F	R1,R2
Subtract Packed (Decimal)	SP	FB	D1(L1,B1),D2(L2,B2)
Subtract Registers	SR	1B	R1,R2
Shift Right Single	SRA	8A	R1,D2(B2)
Shift Right Double	SRDA	8E	R1,D2(B2)
Shift Right Double Logical	SRDL	8C	R1,D2(B2)
Shift Right Single Logical	SRL	88	R1,D2(B2)
Shift and Round Decimal	SRP	F0	D1(L1,B1),D2(B2),I3
Store	ST	50	R1,D2(X2,B2)
Store Character	STC	42	R1,D2(X2,B2)
Store Character under Mask	STCM	BE	R1,M3,D2(B2)
Store Halfword	STH	40	R1,D2(X2,B2)
Store Multiple	STM	90	R1,R3,D2(B2)
Supervisor Call	SVC	0A	I1
Test under Mask	TM	91	D1(B1),I2
Translate	TR	DC	D1(L1,B1),D2(B2)
Translate and Test	TRT	DD	D1(L1,B1),D2(B2)
Unpack	UNPK	F3	D1(L1,B1),D2(L2,B2)
eXclusive Or	X	57	R1,D2(X2,B2)
eXclusive Or Characters	XC	D7	D1(L,B1),D2(B2)
eXclusive Or Immediate	XI	97	D1(B1),I2
eXclusive Or Registers	XR	17	R1,R2
Zero Add Packed	ZAP	F8	D1(L1,B1),D2(L2,B2)

List by Hexadecimal Opcode

The following list is sequenced by the Hexadecimal Opcode.

Instruction	Mnemonic	Hex	Format
Branch and Link Register	BALR	**05**	R1,R2
Branch on Count Register	BCTR	**06**	R1,R2
Branch on Condition Register	BCR	**07**	M1,R2
Supervisor Call	SVC	**0A**	I1
Branch and Set Mode	BSM	**0B**	R1,R2
Branch, Save and Set Mode	BASSM	**0C**	R1,R2
Branch and Save Register	BASR	**0D**	R1,R2
Move Characters Long	MVCL	**0E**	R1,R2
Compare Logical Characters Long	CLCL	**0F**	R1,R2
Load Postive	LPR	**10**	R1,R2
Load Negative	LNR	**11**	R1,R2
Load and Test Register	LTR	**12**	R1,R2
Load Complement Registers	LCR	**13**	R1,R2
aNd Registers	NR	**14**	R1,R2
Compare Logical Registers	CLR	**15**	R1,R2
Or Registers	OR	**16**	R1,R2
eXclusive Or Registers	XR	**17**	R1,R2
Load Register	LR	**18**	R1,R2
Compare Registers	CR	**19**	R1,R2
Add Registers	AR	**1A**	R1,R2
Subtract Registers	SR	**1B**	R1,R2
Multipy Registers	MR	**1C**	R1,R2

Divide Registers	DR	**1D**	R1,R2
Add Logical Registers	ALR	**1E**	R1,R2
Subtract Logical Registers	SLR	**1F**	R1,R2
Store Halfword	STH	**40**	R1,D2(X2,B2)
Load Address	LA	**41**	R1,D2(X2,B2)
Store Character	STC	**42**	R1,D2(X2,B2)
Insert Character	IC	**43**	R1,D2(X2,B2)
Execute	EX	**44**	R1,D2(X2,B2)
Branch and Link	BAL	**45**	R1,D2(X2,B2)
Branch on Count	BCT	**46**	R1,D2((X2,B2)
Branch on Condition	BC	**47**	M1,D2(X2,B2)
Load Halfword	LH	**48**	R1,D2(X2,B2)
Compare Halfword	CH	**49**	R1,D2(X2,B2)
Add Halfword	AH	**4A**	R1,D2(X2,B2)
Subtract Halfword	SH	**4B**	R1,D2(X2,B2)
Multipy Halfword	MH	**4C**	R1,D2(X2,B2)
Branch and Save	BAS	**4D**	R1,D2(X2,B2)
Convert to Decimal	CVD	**4E**	R1,D2((X2,B2)
Convert to Binary	CVB	**4F**	R1,D2((X2,B2)
Store	STb	**50**	R1,D2(X2,B2)
aNd	N	**54**	R1,D2(X2,B2)
Compare Logical	CL	**55**	R1,D2(X2,B2)
Or	O	**56**	R1,D2(X2,B2)
eXclusive Or	X	**57**	R1,D2(X2,B2)
Load	L	**58**	R1,D2(X2,B2)
Compare	C	**59**	R1,D2(X2,B2)
Add	A	**5A**	R1,D2(X2,B2)
Subtract	S	**5B**	R1,D2(X2,B2)
Multipy	M	**5C**	R1,D2(X2,B2)
Divide	D	**5D**	R1,D2((X2,B2)
Add Logical	AL	**5E**	R1,D2(X2,B2)
Subtract Logical	SL	**5F**	R1,D2(X2,B2)
Branch on Index High	BXH	**86**	R1,R3,D2(B2)
Branch on Index Low/Equal	BXLE	**87**	R1,R3,D2(B2)
Shift Right Single Logical	SRL	**88**	R1,D2(B2)
Shift Left Single Logical	SLL	**89**	R1,D2(B2)
Shift Right Single	SRA	**8A**	R1,D2(B2)
Shift Left Single	SLA	**8B**	R1,D2(B2)
Shift Right Double Logical	SRDL	**8C**	R1,D2(B2)
Shift Left Double Logical	SLDL	**8D**	R1,D2(B2)
Shift Right Double	SRDA	**8E**	R1,D2(B2)
Shift Left Double	SLDA	**8F**	R1,D2(B2)
Store Multiple	STM	**90**	R1,R3,D2(B2)
Test under Mask	TM	**91**	D1(B1),I2
Move Immediate	MVI	**92**	D1(B1),I2
aNd Immediate	NI	**94**	D1(B1),I2
Compare Logical Immediate	CLI	**95**	D1(B1),I2
Or Immediate	OI	**96**	D1(B1),I2
eXclusive Or Immediate	XI	**97**	D1(B1),I2
Load Multiple	LM	**98**	R1,R3,D2(B2)
Compare and Swap	CS	**BA**	R1,R3,D2(B2)
Compare Double and Swap	CDS	**BB**	R1,R3,D2(B2)
Compare Logical under Mask	CLM	**BD**	R1,M3,D2(B2)
Store Character under Mask	STCM	**BE**	R1,M3,D2(B2)
Insert Character under Mask	ICM	**BF**	R1,M3,D2(B2)
Move Numerics	MVN	**D1**	D1(L,B1),D2(B2)
Move Characters	MVC	**D2**	D1(L,B1),D2(B2)

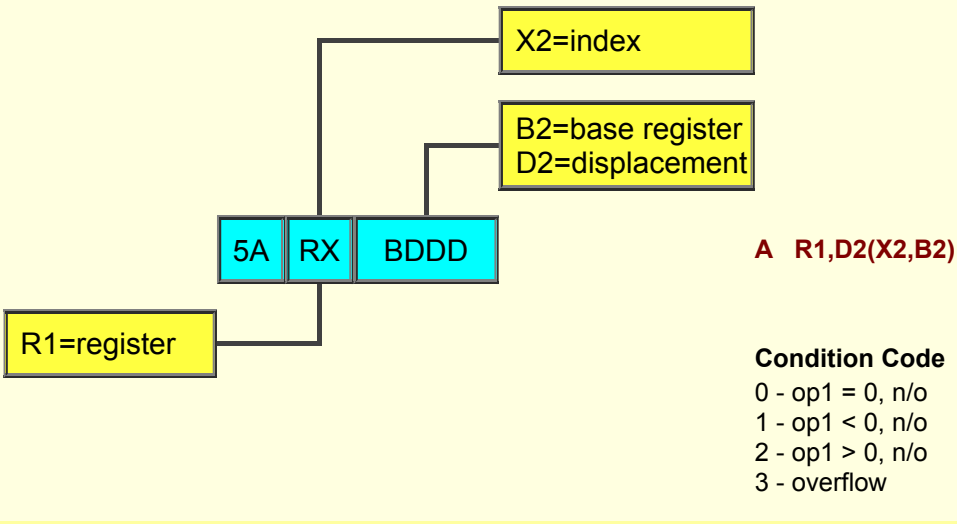
Move Zones	MVZ	**D3**	D1(L,B1),D2(B2)
aNd Characters	NC	**D4**	D1(L,B1),D2(B2)
Compare Logical Characters	CLC	**D5**	D1(L,B1),D2(B2)
Or Characters	OC	**D6**	D1(L,B1),D2(B2)
eXclusive Or Characters	XC	**D7**	D1(L,B1),D2(B2)
Translate	TR	**DC**	D1(L1,B1),D2(B2)
Translate and Test	TRT	**DD**	D1(L1,B1),D2(B2)
Edit	ED	**DE**	D1(L1,B1),D2(B2)
Edit and Mark	EDMK	**DF**	D1(L1,B1),D2(B2)
Move Inverse	MVCIN	**E8**	D1(L,B1),D2(B2)
Shift and Round Decimal	SRP	**F0**	D1(L1,B1),D2(B2),I3
Move with Offset	MVO	**F1**	D1(L1,B1),D2(L2,B2)
Pack	PACK	**F2**	D1(L1,B1),D2(L2,B2)
Unpack	UNPK	**F3**	D1(L1,B1),D2(L2,B2)
Zero Add Packed	ZAP	**F8**	D1(L1,B1),D2(L2,B2)
Compare Packed (Decimal)	CP	**F9**	D1(L1,B1),D2(L2,B2)
Add Packed (Decimal)	AP	**FA**	D1(L1,B1),D2(L2,B2)
Subtract Packed (Decimal)	SP	**FB**	D1(L1,B1),D2(L2,B2)
Multipy Packed (Decimal)	MP	**FC**	D1(L1,B1),D2(L2,B2)
Divide Packed (Decimal)	DP	**FD**	D1(L1,B1),D2(L2,B2)

Instruction Overview

This program may serve as a tutorial for programmers that are new to 370 assembler or as a reference for experienced programmers.

A, Add

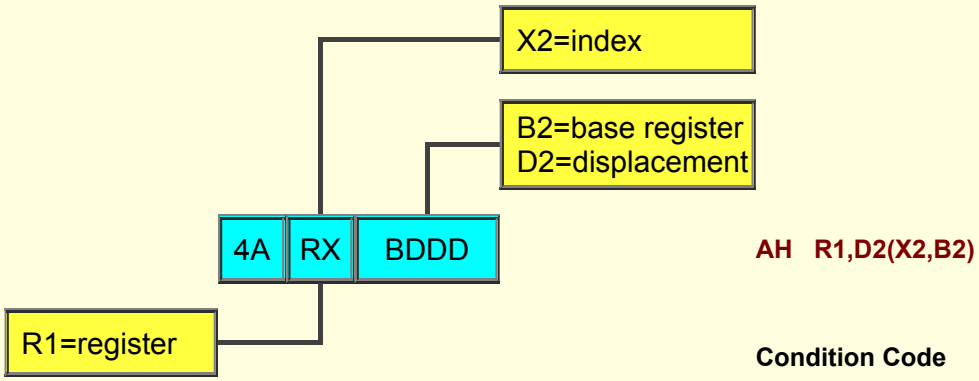
The full word (4 bytes) located at the storage address specified by operand-2 (x2+b2+d2} is added to the register specified by operand-1(r1). Operand-2 remains unchanged. The condition code is set as shown below.



A, Add Instruction, the operand format is R1,D2(X2,B2)

AH, Add Halfword

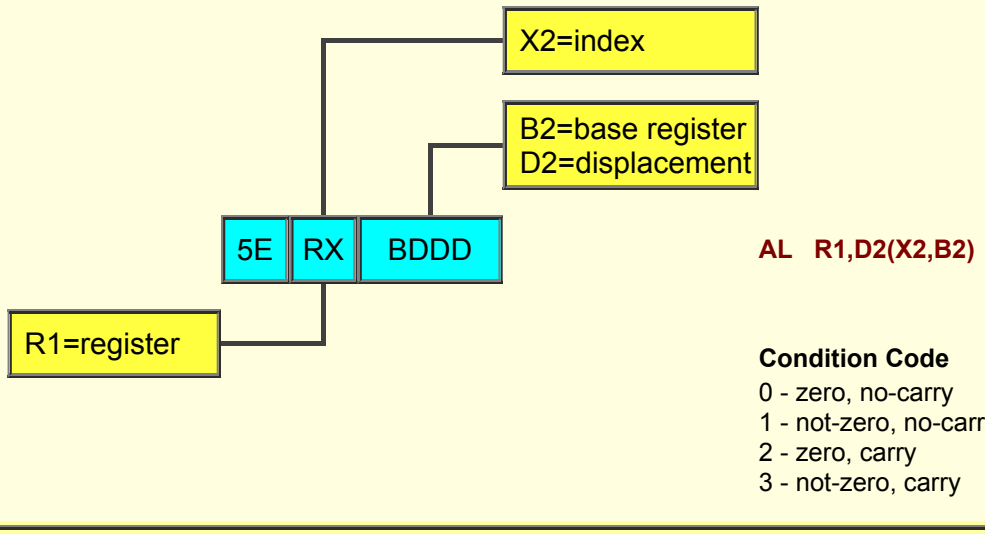
The half word (2 bytes) located at the storage address specified by operand-2 (x2+b2+d2) is added to the register specified by operand-1 (r1). Operand-2 remains unchanged. The condition code is set as shown below.



AH, Add Halfword Instruction, the operand format is R1,D2(X2,B2)

AL, Add Logical

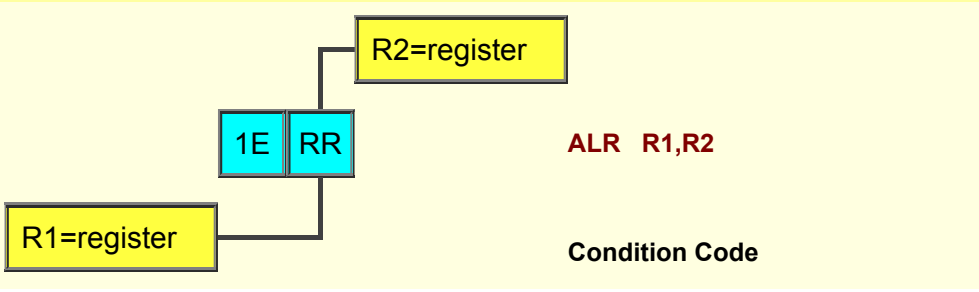
The full word (4 bytes) located at the storage address specified by operand-2 (x2+b2+d2) is added to the register specified by operand-1 (r1). Operand-2 remains unchanged. The condition code is set as shown below.



AL, Add Logical Instruction, the operand format is R1,D2(X2,B2)

ALR, Add Logical Registers

The register specified by operand-2 (r2) is added to the register specified by operand-1 (r1). Operand-2 remains unchanged. The condition code is set as shown below.



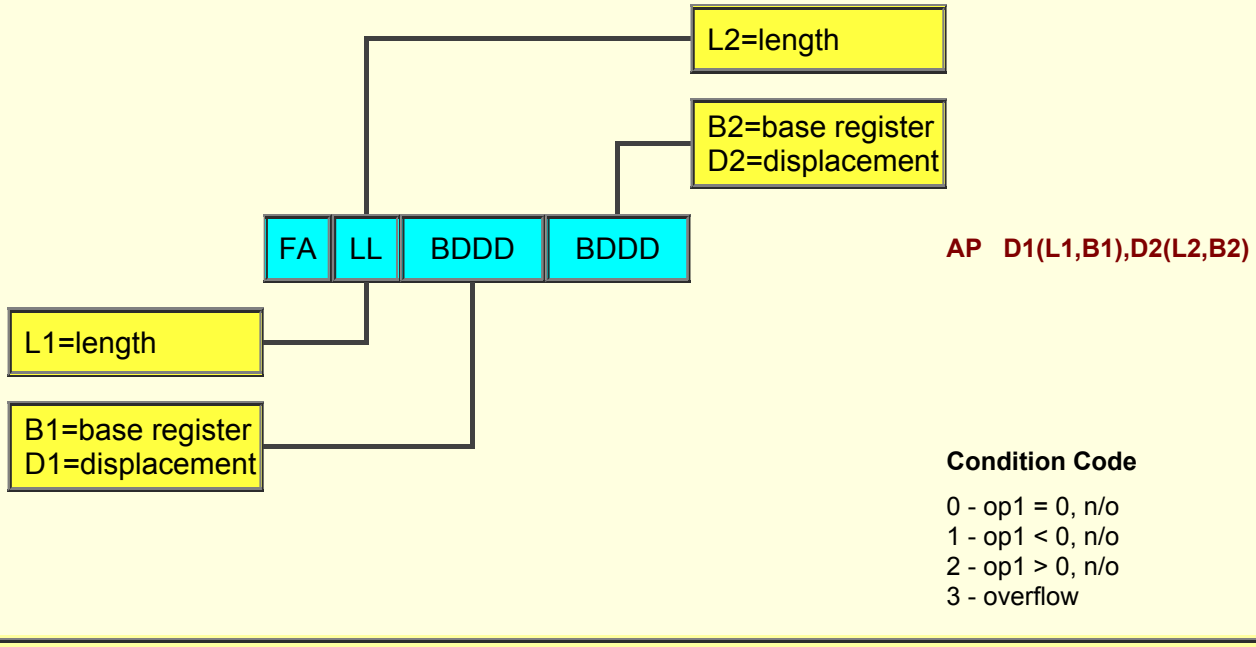
0 - zero, no-carry
1 - not-zero, no-carry
2 - zero, carry
3 - not-zero, carry

ALR, Add Logical Registers Instruction, the operand format is R1,R2

AP, Add Packed (Decimal)

The data string located at the storage address specified by operand-2 (b2+d2) is added to the data string located at the storage address specified by operand-1 (b1+d1). Operand-2 remains unchanged.

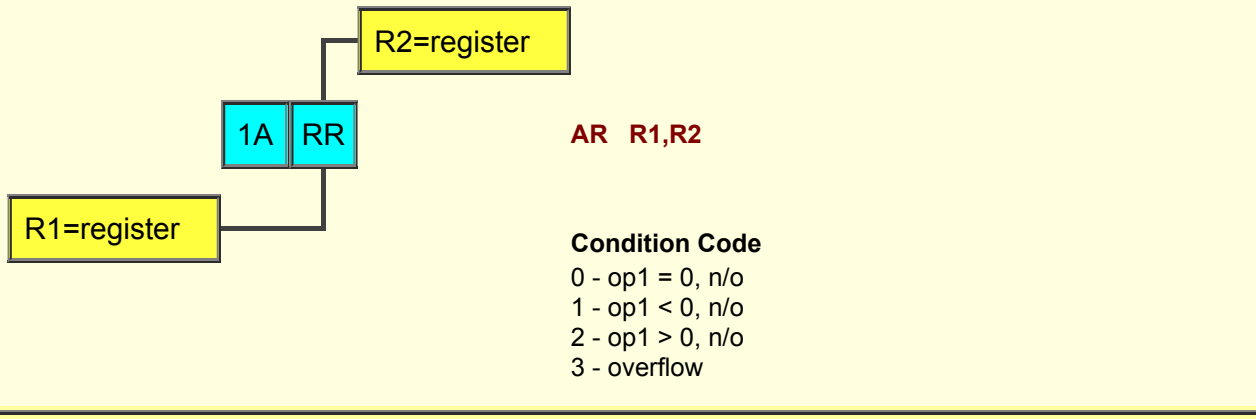
The operands may be different lengths with a maximum length of 16 bytes *(or 31 digits since this is packed)* for each operand. The condition code is set as shown below.



AP, Add Packed (Decimal) Instruction, the operand format is D1(L1,B1),D2(L2,B2)

AR, Add Registers

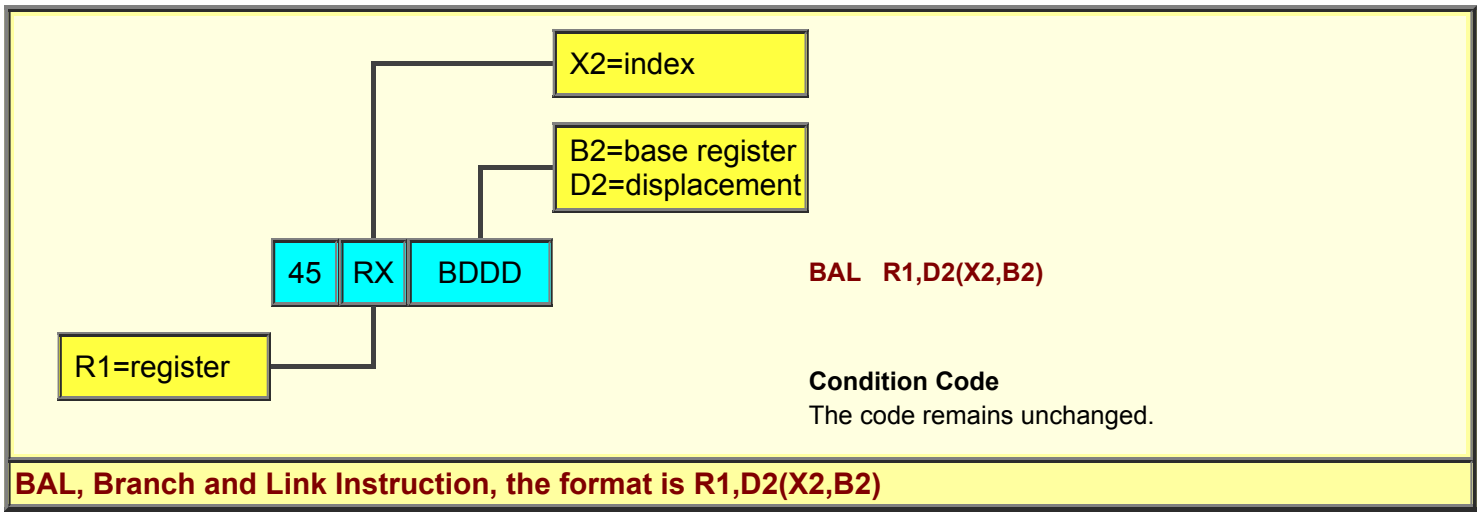
The register specified by operand-2 (r2) is added to the register specified by operand-1 (r1). Operand-2 remains unchanged. The condition code is set as shown below.



AR, Add Registers Instruction, the operand format is R1,R2

BAL, Branch and Link

The next sequential address is placed in operand-1 (r1) as linkage information, a branch to operand-2 (x2+b2+d2) is performed.



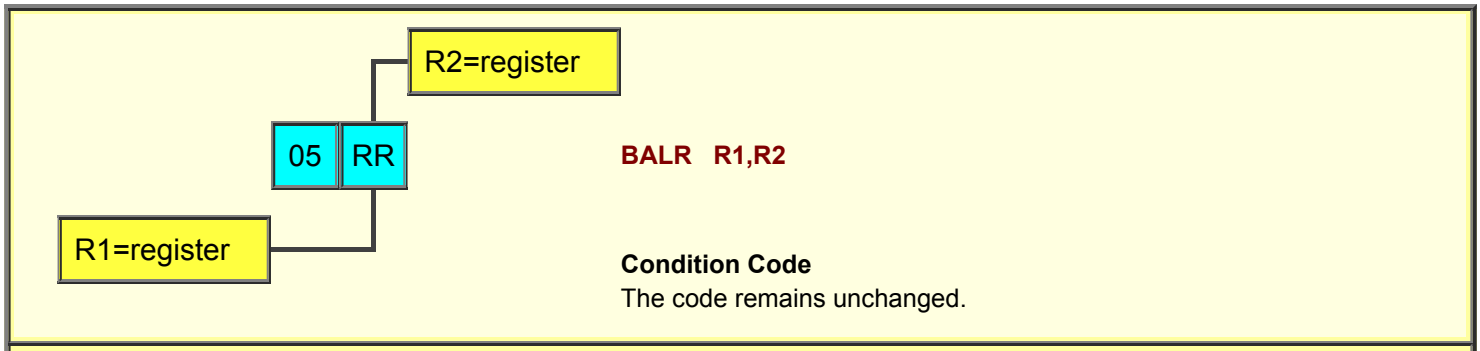
BAL, Branch and Link Instruction, the format is R1,D2(X2,B2)

This instruction was originally intended for use with 24-bit addressing and is still provided for back-level compatibility. Only the rightmost 24 bits (bits 8-31) of the full word are used when branching or linking. The first 8 bits (bits 0-7) are not used as part of the address.

Note: *This instruction will work with 31-bit addressing mode but it is recommended that the BAS instruction be used instead of the BAL instruction.*

BALR, Branch and Link Register

The next sequential address is placed in operand-1 (r1) as linkage information, a branch to operand-2 (r2) is performed.



Note: When the operand 2 value is zero, the link information is loaded without branching.

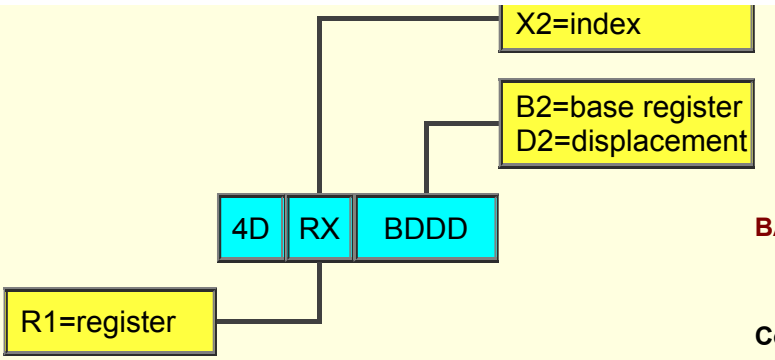
BALR, Branch and Link Register Instruction, the operand format is R1,R2

This instruction was originally intended for use with 24-bit addressing and is still provided for back-level compatibility. Only the rightmost 24 bits (bits 8-31) of the full word are used when branching or linking. The first 8 bits (bits 0-7) are not used as part of the address.

Note: *This instruction will work with 31-bit addressing mode but it is recommended that the BASR instruction be used instead of the BALR instruction.*

BAS, Branch and Save

The next sequential address is placed in operand-1 (r1) as linkage information, a branch to operand-2 (x2+b2+d2) is performed.



BAS R1,D2(X2,B2)

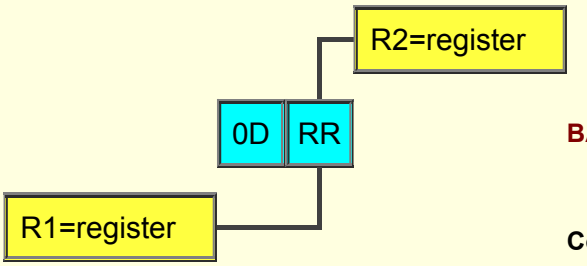
Condition Code
The code remains unchanged

BAS, Branch and Save Instruction, the format is R1,D2(X2,B2)

This instruction was introduced for use with 31-bit addressing and also works with 24-bit addressing mode. Only the rightmost 31 bits (bits 1-31) of the full word are used when branching or linking. The first bit (bit 0) is not used as part of the address.

BASR, Branch and Save Register

The next sequential address is placed in operand-1 (r1) as linkage information, a branch to operand-2 (r2) is performed.



BASR R1,R2

Condition Code
The code remains unchanged

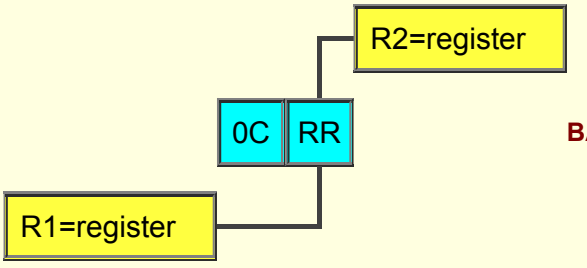
Note: When the operand 2 value is zero, the link information is loaded into R1 without branching.

BASR, Branch and Save Register Instruction, the operand format is R1,R2

This instruction was introduced for use with 31-bit addressing and also works with 24-bit addressing mode. Only the rightmost 31 bits (bits 1-31) of the full word are used when branching or linking. The first bit (bit 0) is not used as part of the address.

BASSM, Branch and Save and Set Mode

The next sequential address is placed in operand-1 (r1) as linkage information, a branch to operand-2 (r2) is performed. The addressing mode is also set by this instruction.

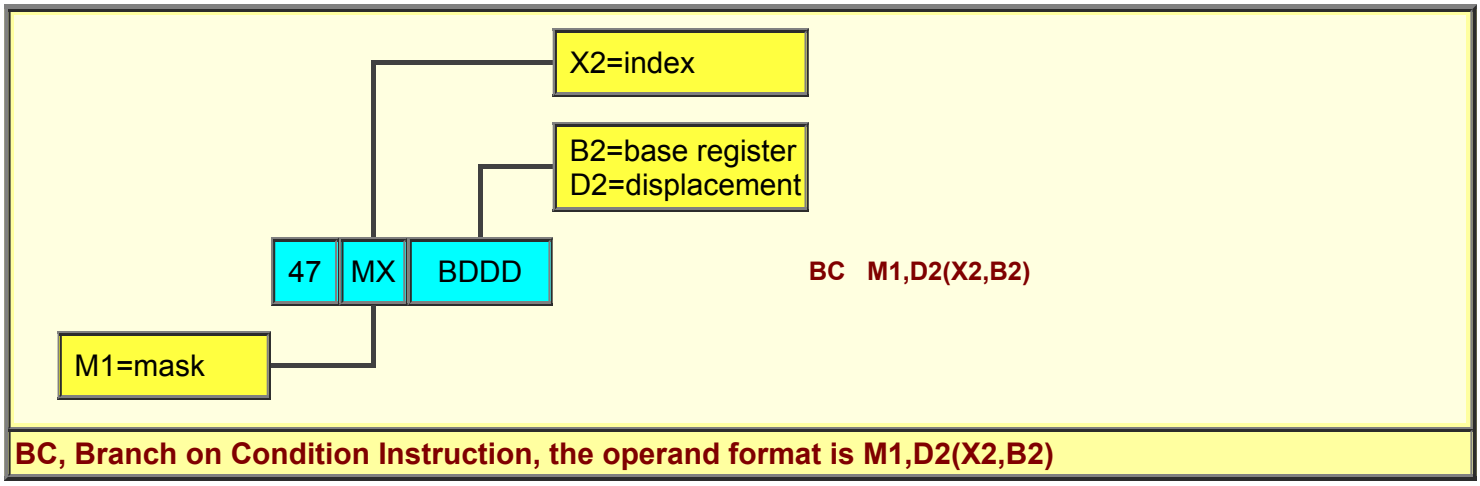


BASSM R1,R2

BASSM, Branch and Save and Set Mode Instruction, the operand format is R1,R2

BC, Branch on Condition

If the condition-code (cc) has a "bit-ON" match with the instruction-mask (m1) then branch to the address specified by operand-2 (x2+b2+d2) else do the next sequential instruction.



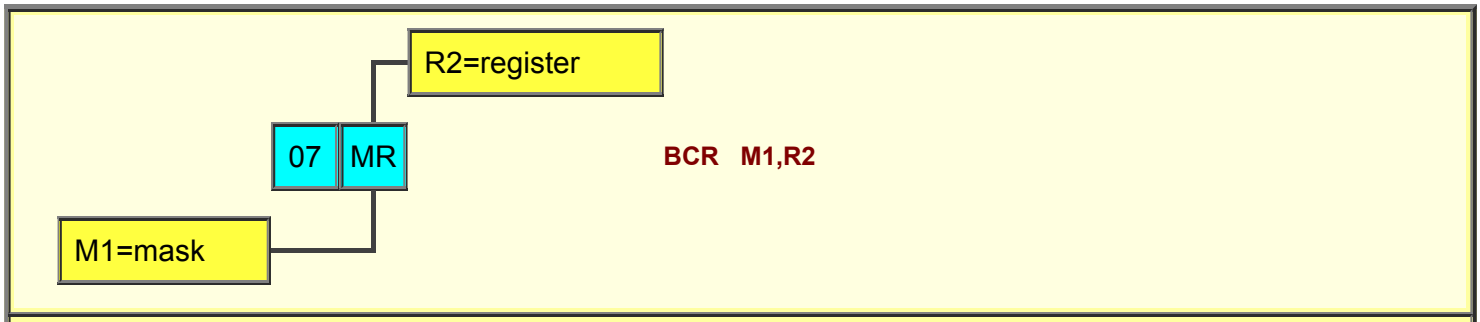
BC, Branch on Condition Instruction, the operand format is M1,D2(X2,B2)

Note: *The following table shows the Condition Code to Instruction-Mask relationship.*

cc	relationship	mask-bits	hex-code
0	equal-zero	1000	x'8x'
1	low-minus	0100	x'4x'
2	high-plus	0010	x'2x'
3	overflow	0001	x'1x'

BCR, Branch on Condition Register

If the condition-code (cc) has a "bit-ON" match with the instruction-mask (m1) then branch to the address specified by operand-2 (r2) else do the next sequential instruction.



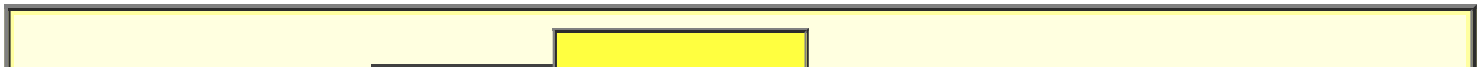
BCR, Branch on Condition Register Instruction, the operand format is M1,R2

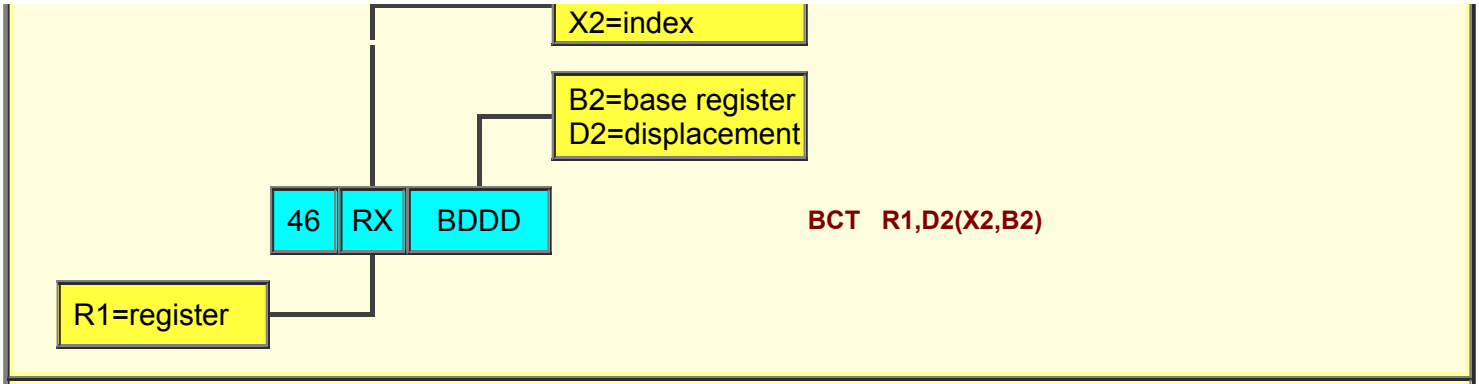
Note: *The following table shows the Condition Code to Instruction-Mask relationship.*

cc	relationship	mask-bits	hex-code
0	equal-zero	1000	x'8x'
1	low-minus	0100	x'4x'
2	high-plus	0010	x'2x'
3	overflow	0001	x'1x'

BCT, Branch on Count

A one is subtracted from operand 1 (r1). If r1 decrements to zero then normal instruction sequencing proceeds else branch to address specified by operand 2.

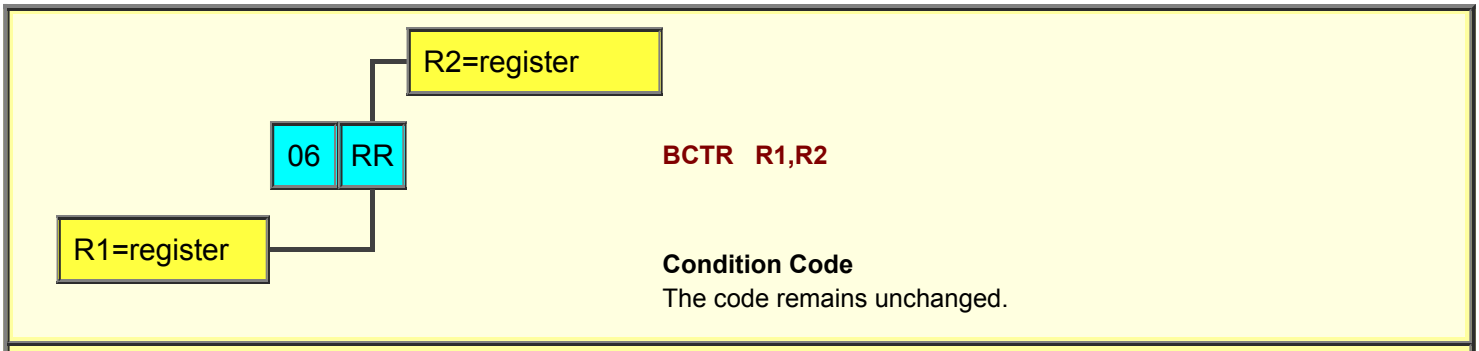




BCT, Branch on Count Instruction, the operand format is R1,D2(X2,B2)

BCTR, Branch on Count Register

A one is subtracted from operand 1 (r1). If r1 decrements to zero then normal instruction sequencing proceeds else branch to address specified by operand 2

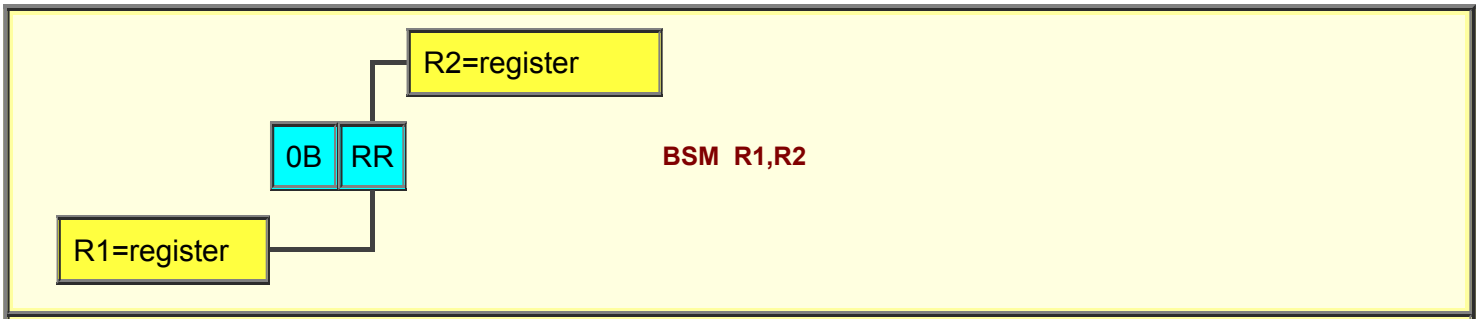


Note: If r2 is zero then r1 is decremented but a branch is never taken.

BCTR, Branch on Condition Register Instruction, the operand format is R1,R2

BSM, Branch and Set Mode

A branch is performed and the addressing mode is also set by this instruction.



BSM, Branch and Set Mode Instruction, the operand format is R1,R2

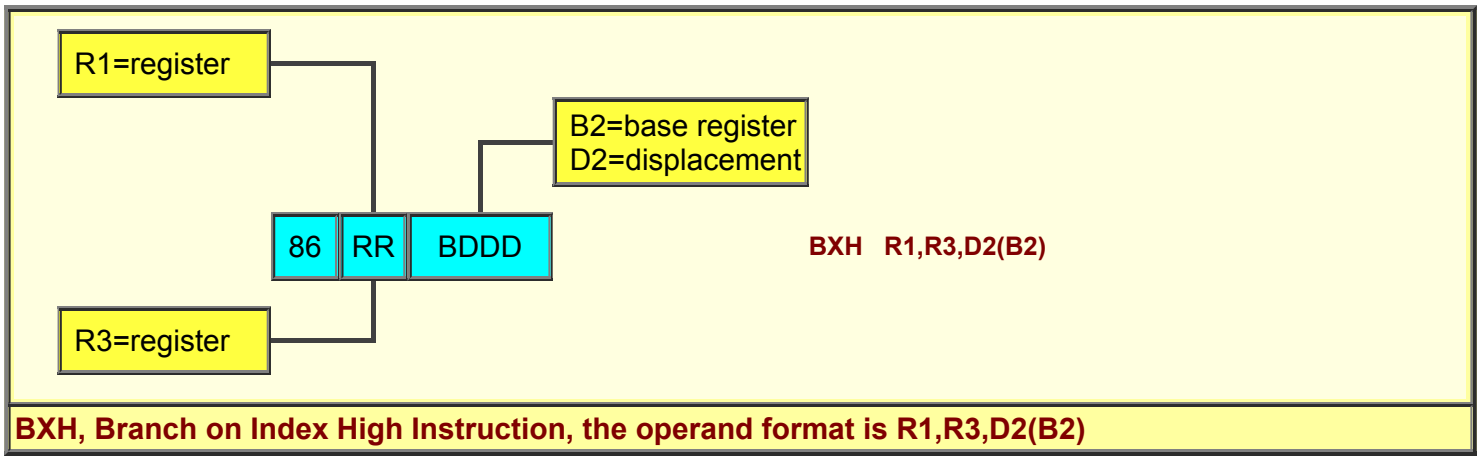
BXH, Branch on Index High

Operand-1 (r1) is the index value that will be incremented. Operand-2 (b2+d2) is the address to branch to when the compare conditions are met. Operand-3 (r3) may be a single register or a register pair.

If operand-3 register number is even then a register pair are used as the increment and the compare

value. If operand-3 is odd a single register is used as both the increment and the compare value. The increment is a signed binary number and may be used to increase or decrease the value in Operand-1

When the BXH instruction is executed the incrementing value specified by operand-3 is added to operand-1. Operand-1 is then compared with the compare value specified by operand-3 and if operand-1 is high then a branch is performed. Otherwise, the next sequential instruction is executed.



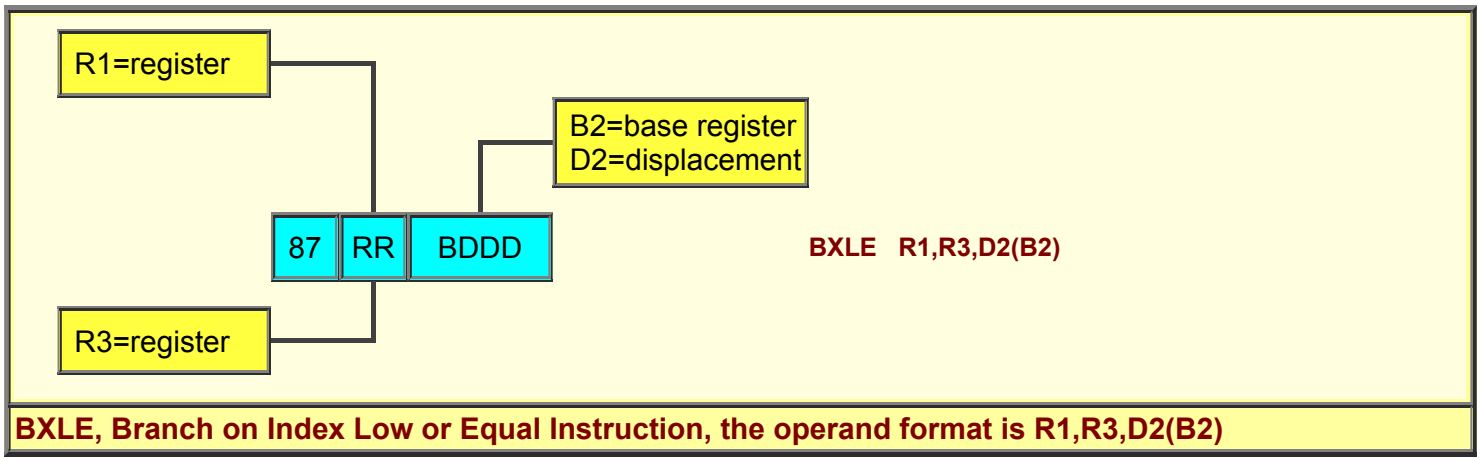
BXH, Branch on Index High Instruction, the operand format is R1,R3,D2(B2)

BXLE, Branch on Index Low or Equal

Operand-1 (r1) is the index value that will be incremented. Operand-2 (b2+d2) is the address to branch to when the compare conditions are met. Operand-3 (r3) may be a single register or a register pair.

If operand-3 register number is even then a register pair are used as the increment and the compare value. If operand-3 is odd a single register is used as both the increment and the compare value. The increment is a signed binary number and may be used to increase or decrease the value in Operand-1

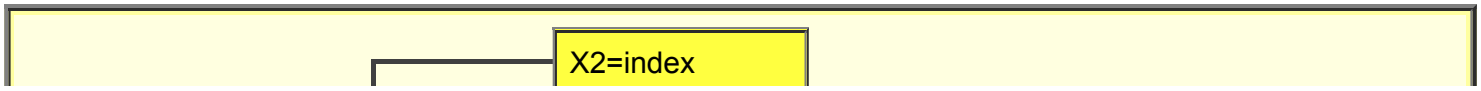
When the BXLE instruction is executed the incrementing value specified by operand-3 is added to operand-1. Operand-1 is then compared with the compare value specified by operand-3 and if operand-1 is low or equal then a branch is performed. Otherwise, the next sequential instruction is executed.

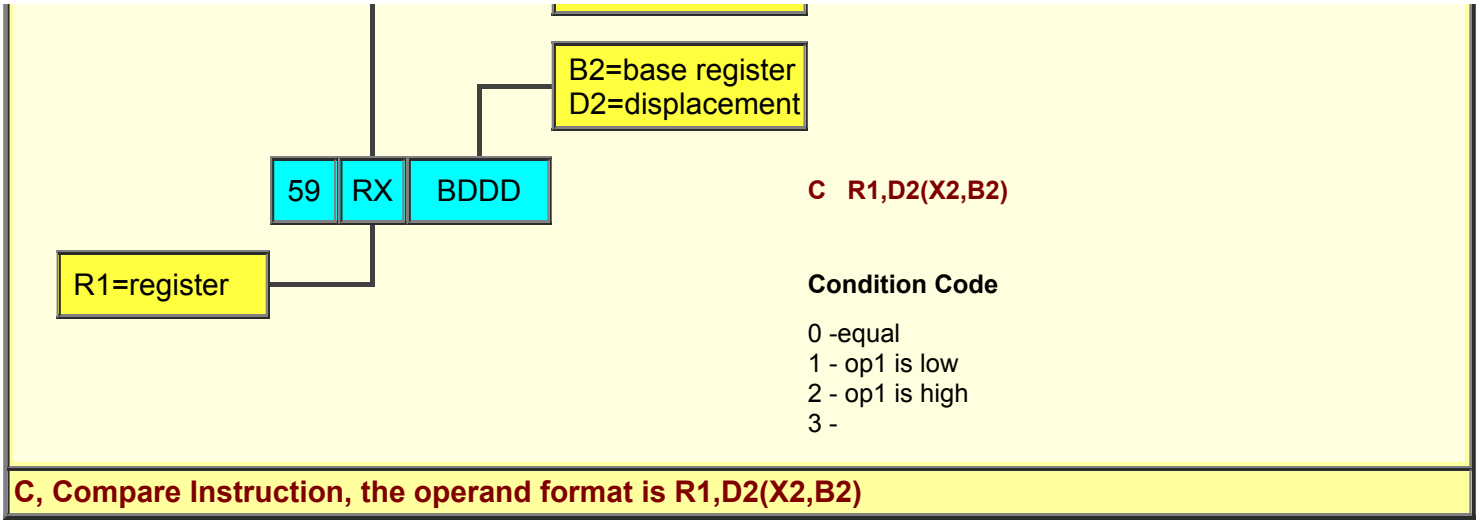


BXLE, Branch on Index Low or Equal Instruction, the operand format is R1,R3,D2(B2)

C, Compare

Operand-1 (r1) is compared with the data string located at the storage address specifed by operand-2 (x2+b2+d2) . The result is posted in the condition code.

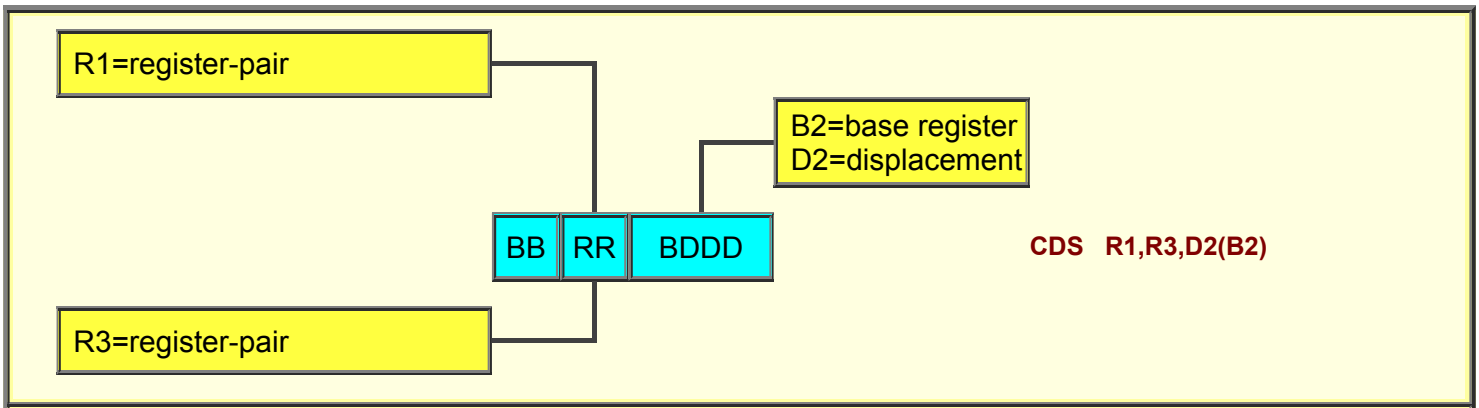




C, Compare Instruction, the operand format is R1,D2(X2,B2)

CDS, Compare Double and Swap

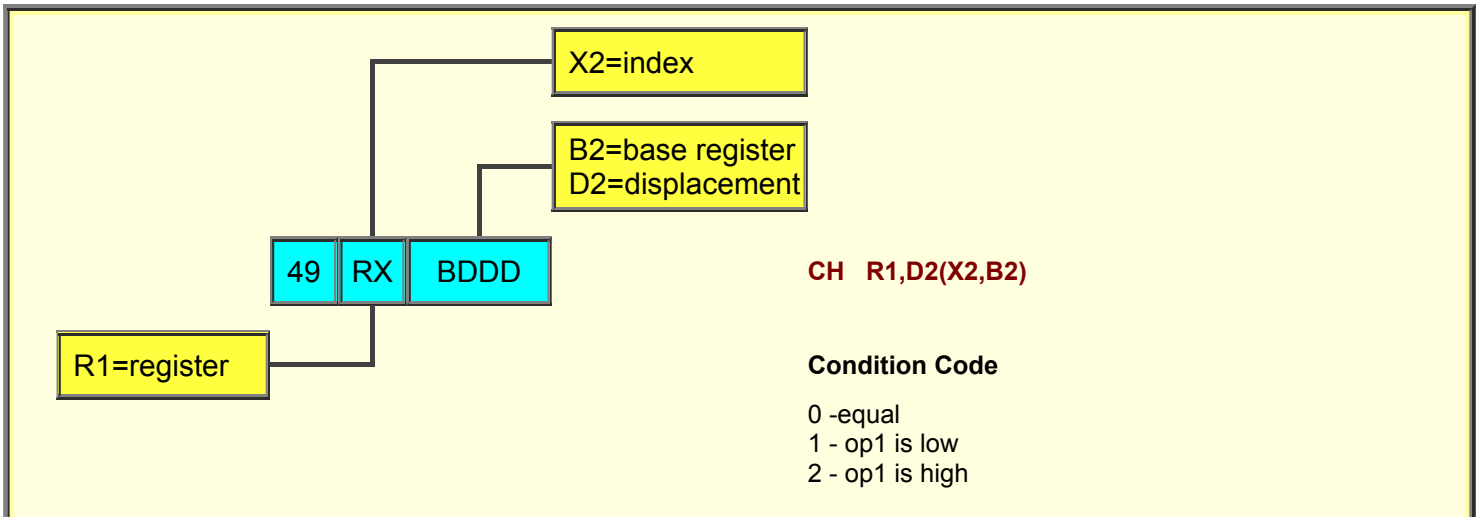
Operand-1 (r1) and operand-2 (b2+d2) are compared. If equal then operand-3 (r3) is stored at the storage address specified by operand-2 (b2+d2). Otherwise, the data string located at the address specified by operand-2 (b2+d2) is loaded into operand-1 (r1). The operands are 64 bits.



CDS, Compare Double and Swap Instruction, the operand format is R1,R3,D2(B2)

CH, Compare Halfword

Operand-1 (r1) is compared with the data string located at the storage address specifed by operand-2 (x2+b2+d2) . The result is posted in the condition code.

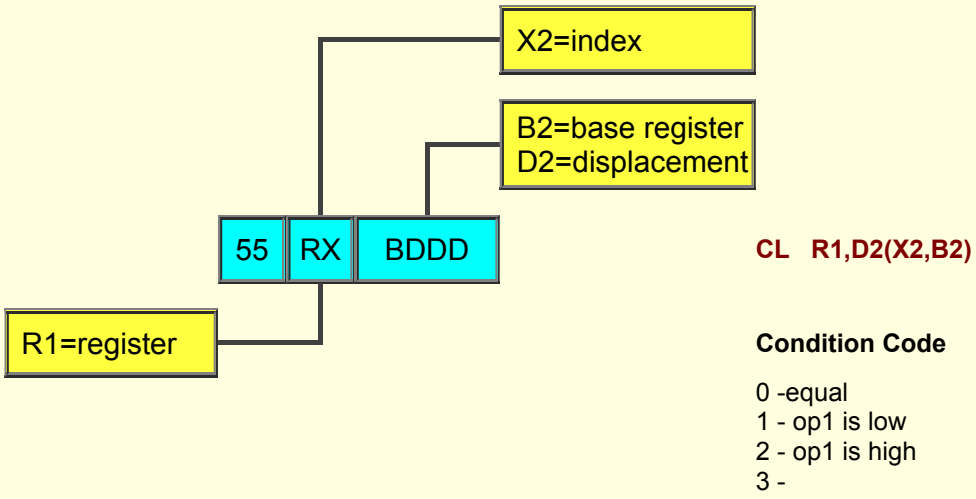


3 -

CH, Compare Halfword Instruction, the operand format is R1,D2(X2,B2)

CL, Compare Logical

Operand-1 (r1) is compared with the data string located at the storage address specified by operand-2 (x2+b2+d2). The result is posted in the condition code. This is a full word (32-bit) comparison.



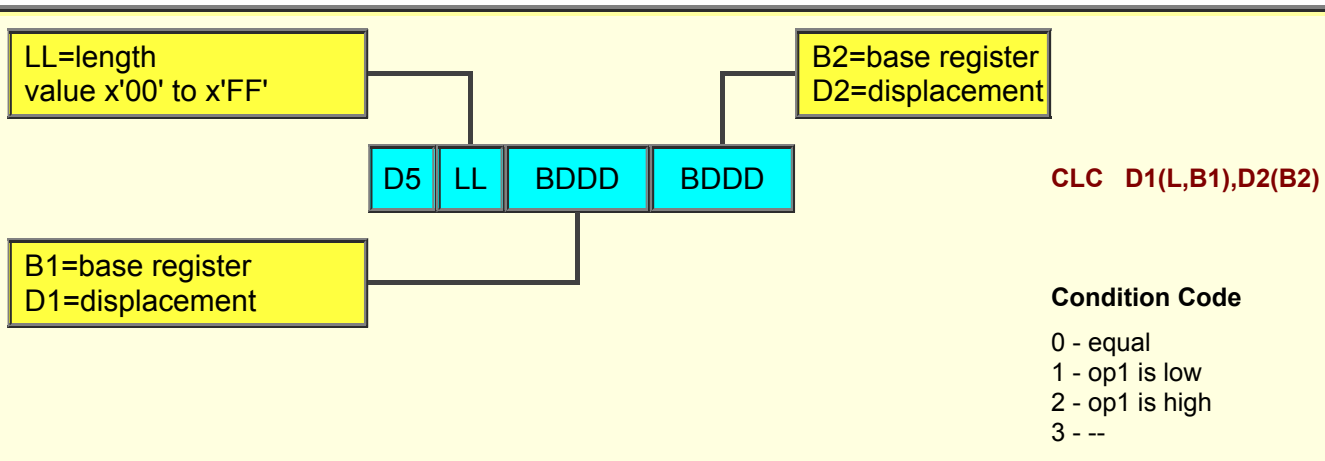
CL, Compare Logical Instruction, the operand format is R1,D2(X2,B2)

CLC, Compare Logical Characters

The data string located at the storage address specified by operand-2 (b2+d2) is compared to the data string located at the storage address specified by operand-1 (b1+d1). Both operands remains unchanged.

The number of bytes compared is determined by the length specified in the 2nd byte of the CLC instruction. The length specified is actually the length-1 or x'00' through x'FF'.

The length of each operand is the same. For example, if x'FF' is specified as the length then operand-1 would be 256 bytes and operand-2 would be 256 bytes. The condition code is set as shown below.



CLC, Compare Logical Characters Instruction, the operand format is D1(L,B1),D2(B2)

CLCL, Compare Logical Characters Long

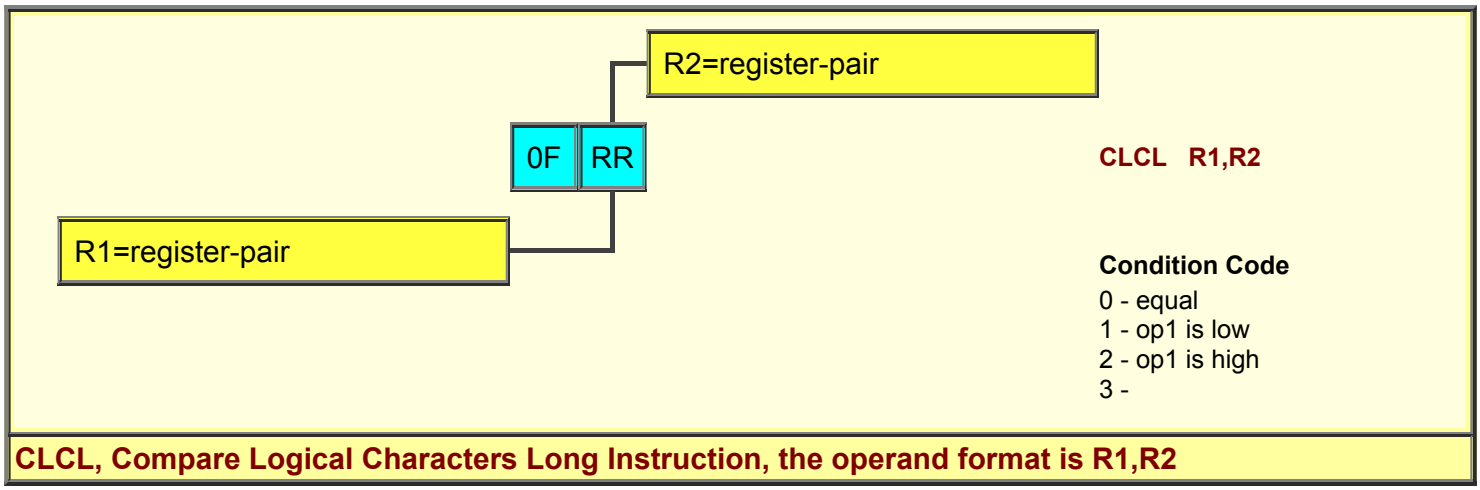
The data string identified by operand-1 is compared with the data string identified by operand-2. Both operands are register-pairs.

The first register of each operand needs to be loaded with the storage addresses of the data strings to be compared. The second register of each operand needs to be loaded with the length of each operand.

The data strings may be different length. The high-order byte of the second register of operand-2 is treated as the padding character if the operands are different lengths.

The compare proceeds from left to right, low storage to high storage and as each character is found to be equal the length registers are decremented. If the CLCL results in and equal condition then the length registers should be zero with one exception.

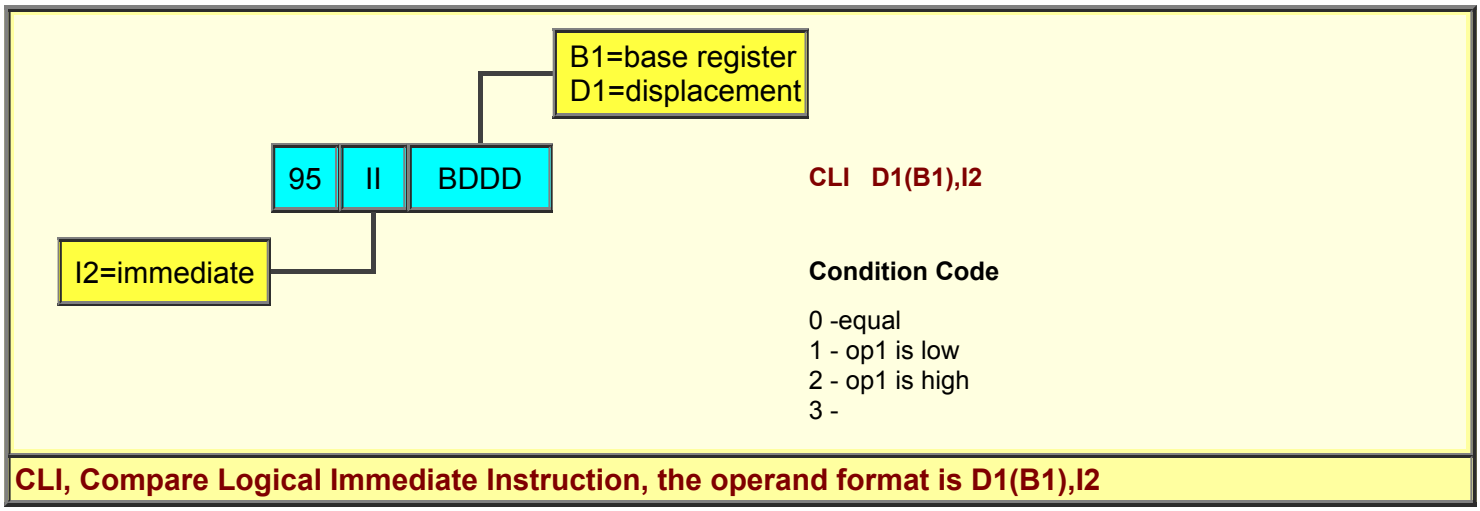
Remember, if a pad character was specified in the high-order (bits 0-7) byte of the second register of operand-2 then only the remaining three bytes (bits 8-31) will be zero. The results are posted in the condition code.



CLCL, Compare Logical Characters Long Instruction, the operand format is R1,R2

CLI, Compare Logical Immediate

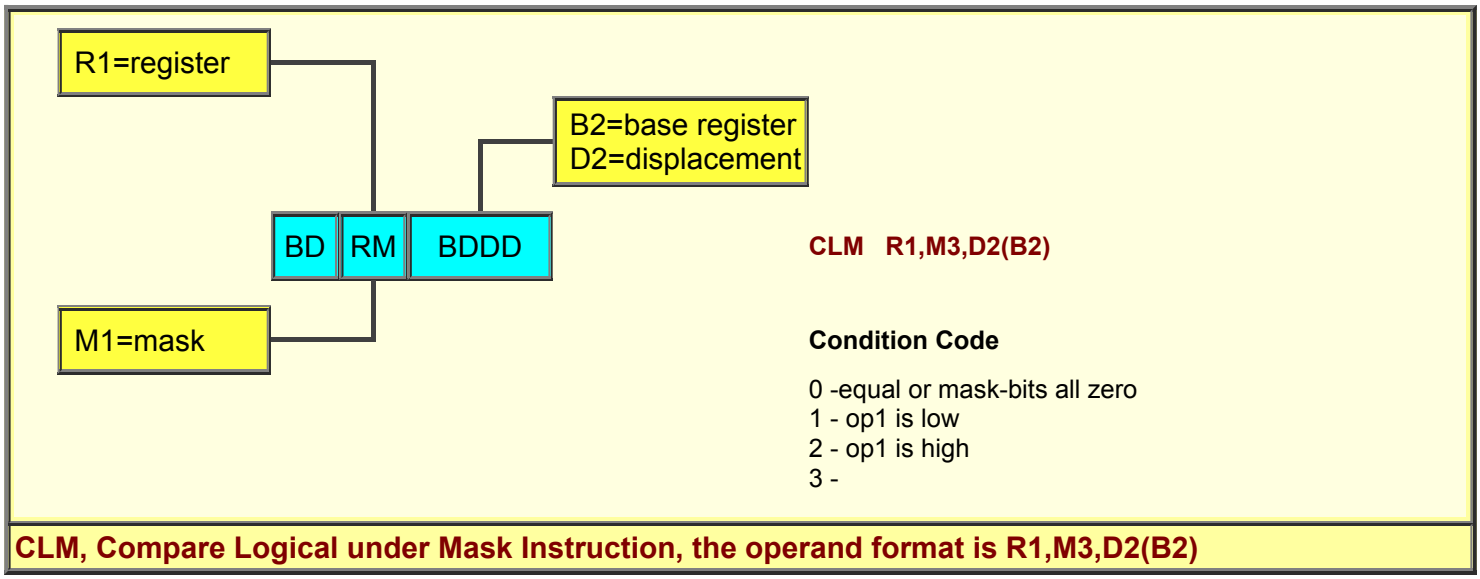
Operand-2 *(the immediate data that is the second byte of the instruction itself)* is compared to the one-byte at the storage address specifed by operand-1 (b1+d1). The results of the compare are posted in the condition code.



CLI, Compare Logical Immediate Instruction, the operand format is D1(B1),I2

CLM, Compare Logical under Mask

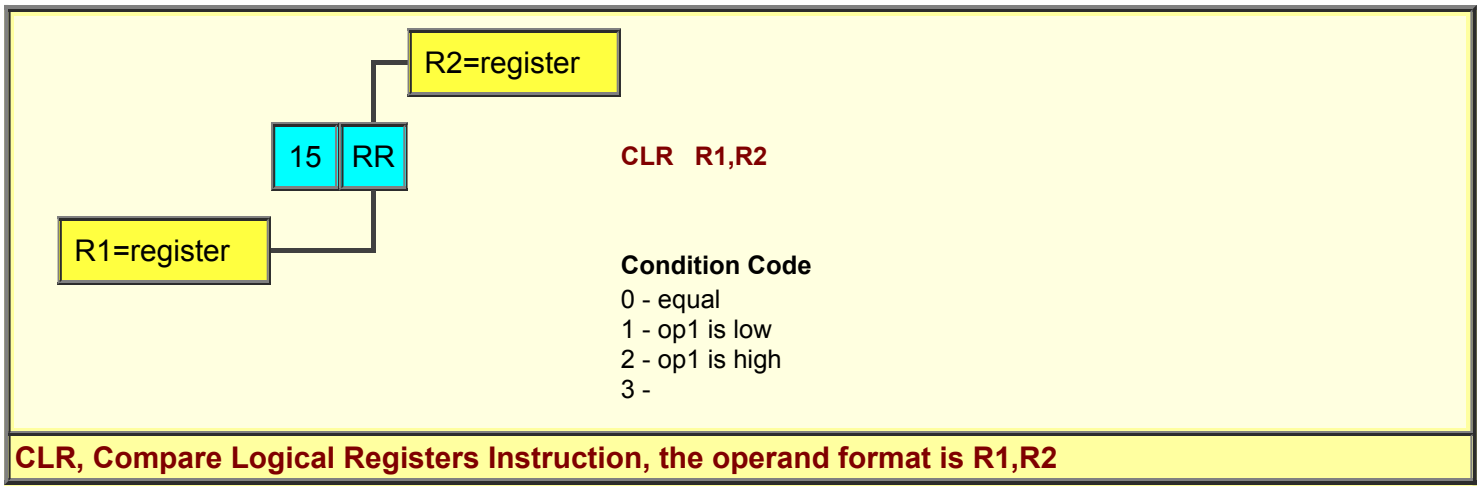
Operand-1 (r1) under control of the mask (m3) is compared to the data string located at the storage location specified by operand-2. The results are posted in the condition code.



CLM, Compare Logical under Mask Instruction, the operand format is R1,M3,D2(B2)

CLR, Compare Logical Registers

Operand-1 (r1) is compared with operand-2 (r2). The compare is a logical compare of all 32-bits. The results are posted in the condition code.

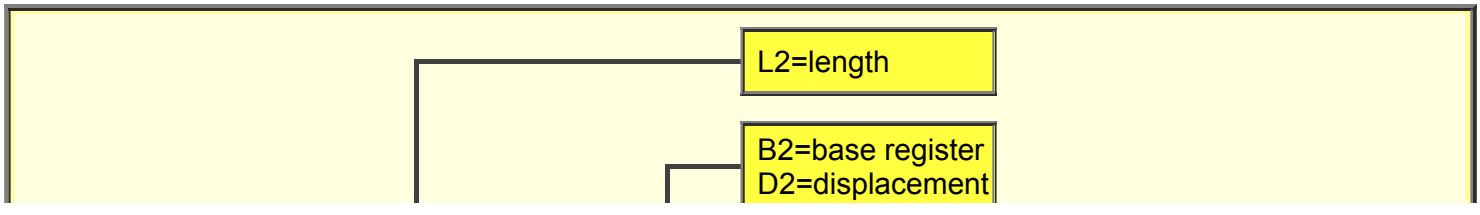


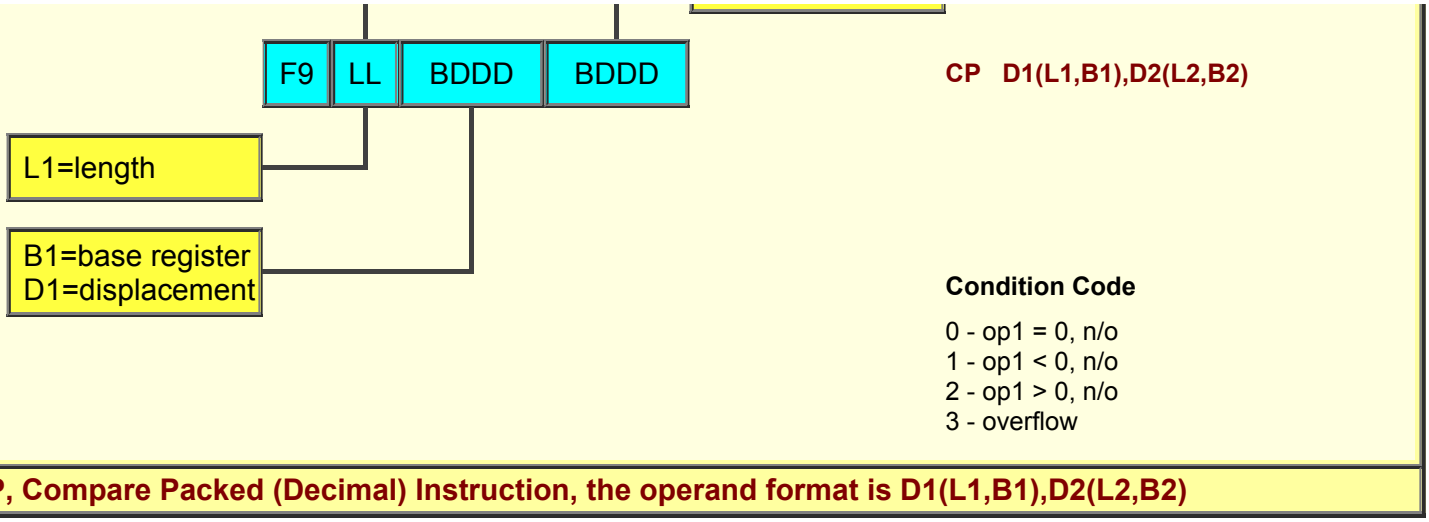
CLR, Compare Logical Registers Instruction, the operand format is R1,R2

CP, Compare Packed (Decimal)

The data string located at the storage address specified by operand-2 (b2+d2) is compared to the data string located at the storage address specified by operand-1 (b1+d1).

The operands may be different lengths with a maximum length of 16 bytes *(or 31 digits since this is packed)* for each operand. This is an arithmetic comparison. The condition code is set as shown below.

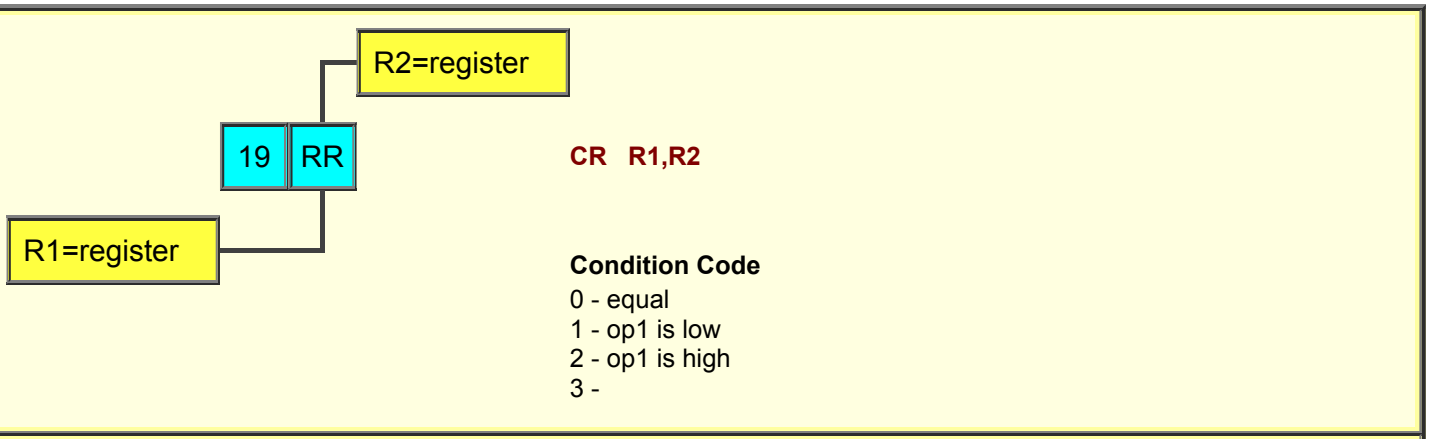




CP, Compare Packed (Decimal) Instruction, the operand format is D1(L1,B1),D2(L2,B2)

CR, Compare Registers

The register specified by operand-2 (r2) is compared to the the register specified by operand-1 (r1). Operand-2 remains unchanged. The condition code is set as shown below.

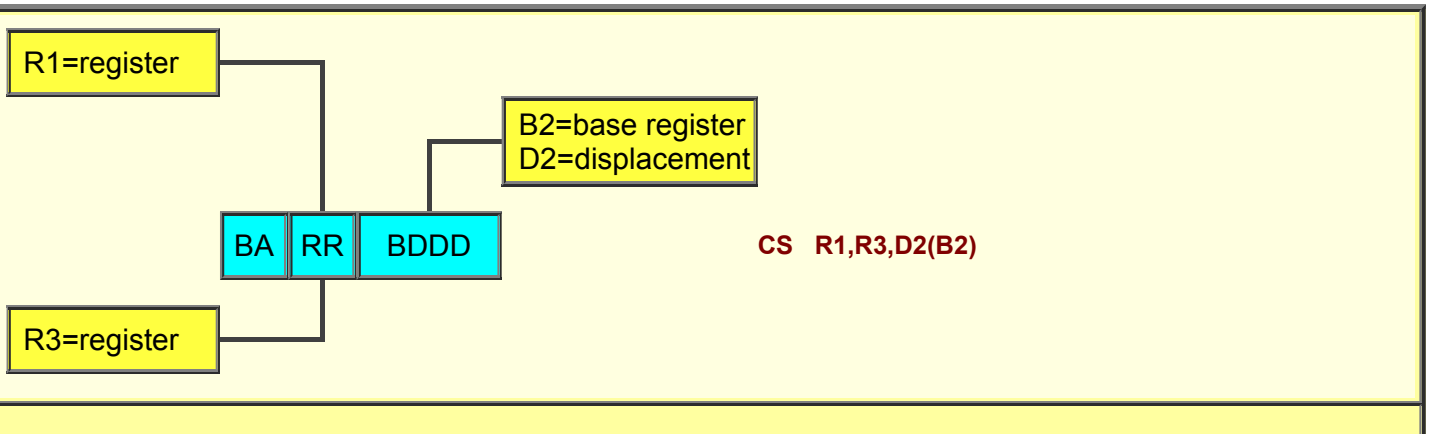


CR, Compare Registers Instruction, the operand format is R1,R2

CS, Compare and Swap

Operand-1 (r1) and operand-2 (b2+d2) are compared. If equal then operand-3 (r3) is stored at the storage address specified by operand-2 (b2+d2).

Otherwise, the data string located at the address specified by operand-2 (b2+d2) is loaded into operand-1 (r1). The operands are 32 bits.

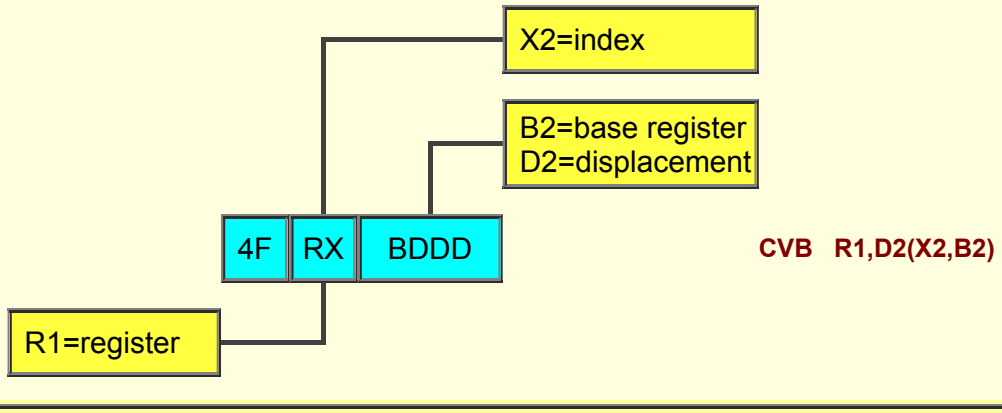


CS, Compare and Swap Instruction, the operand format is R1,R3,D2(B2)

CVB, Convert to Binary

The data string at the storage location specified by operand-2 (x2+b2+d2) is translated from decimal to binary and the result is stored in operand-1 (r1).

Operand-2 remains unchanged and should be an 8-byte packed (15 digit and sign), decimal value.

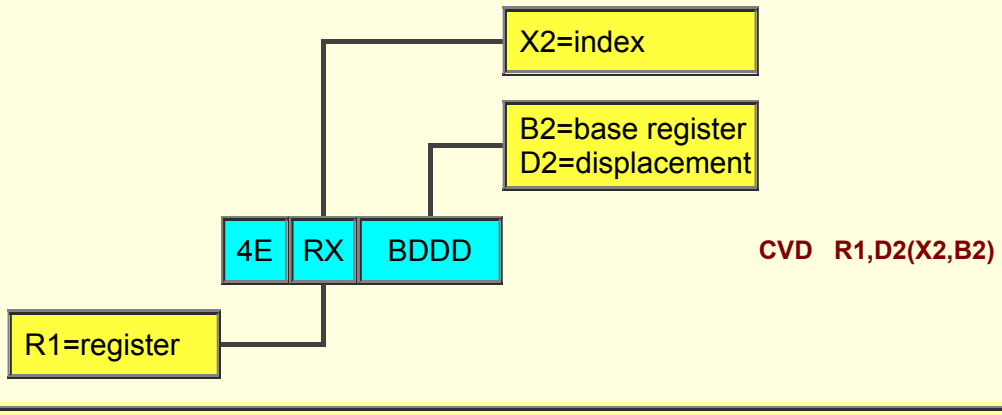


CVB, Convert to Binary Instruction, the operand format is R1,D2(X2,B2)

CVD, Convert to Decimal

Operand-1 (r1) is translated from binary to decimal and the result is stored at the storage location specified by operand-2 (x2+b2+d2). Operand-1 remains unchanged.

Operand-2 should be an 8-byte packed *(15 digit and sign)*, decimal value.

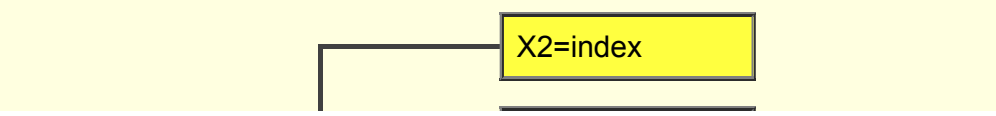


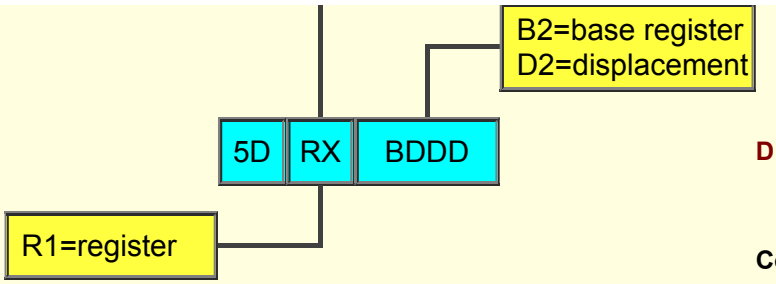
CVD, Convert to Decimal Instruction, the operand format is R1,D2(X2,B2)

D, Divide

Operand-1 (r1) is an even/odd pair of registers (dividend) that is divided by the value at the storage location specified by operand-2 (x2+b2+d2).

The remainder is put in r1-even and the quotient is put in r1-odd.





D R1,D2(X2,B2)

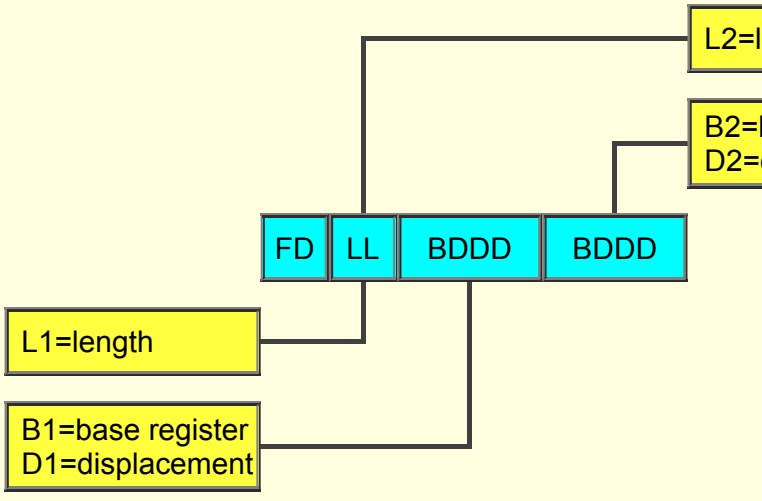
Condition Code
The code remains unchanged.

D, Divide Instruction, the operand format is R1,D2(X2,B2)

DP, Divide Packed (Decimal)

The data string located at the storage address specified by operand-1 (b1+d1) is divided by the data string located at the storage address specified by operand-2 (b2+d2).

Operand-2 remains unchanged.



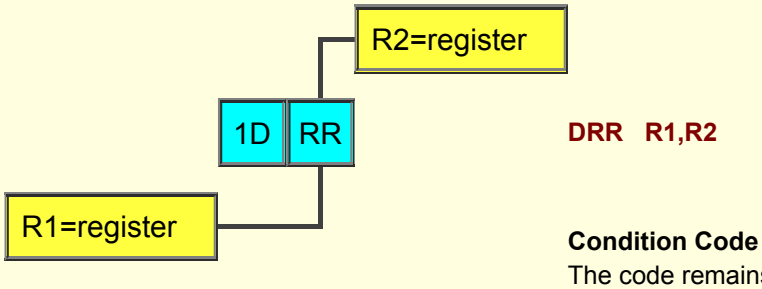
DP D1(L1,B1),D2(L2,B2)

Condition Code
The code remains unchanged.

DP, Divide Packed (Decimal) Instruction, the operand format is D1(L1,B1),D2(L2,B2)

DR, Divide Registers

Operand-1 (r1) is an even/odd pair of registers (dividend) that is divided by operand-2 (r2). The remainder is put in r1-even and the quotient is put in r1-odd.



DRR R1,R2

Condition Code
The code remains unchanged.

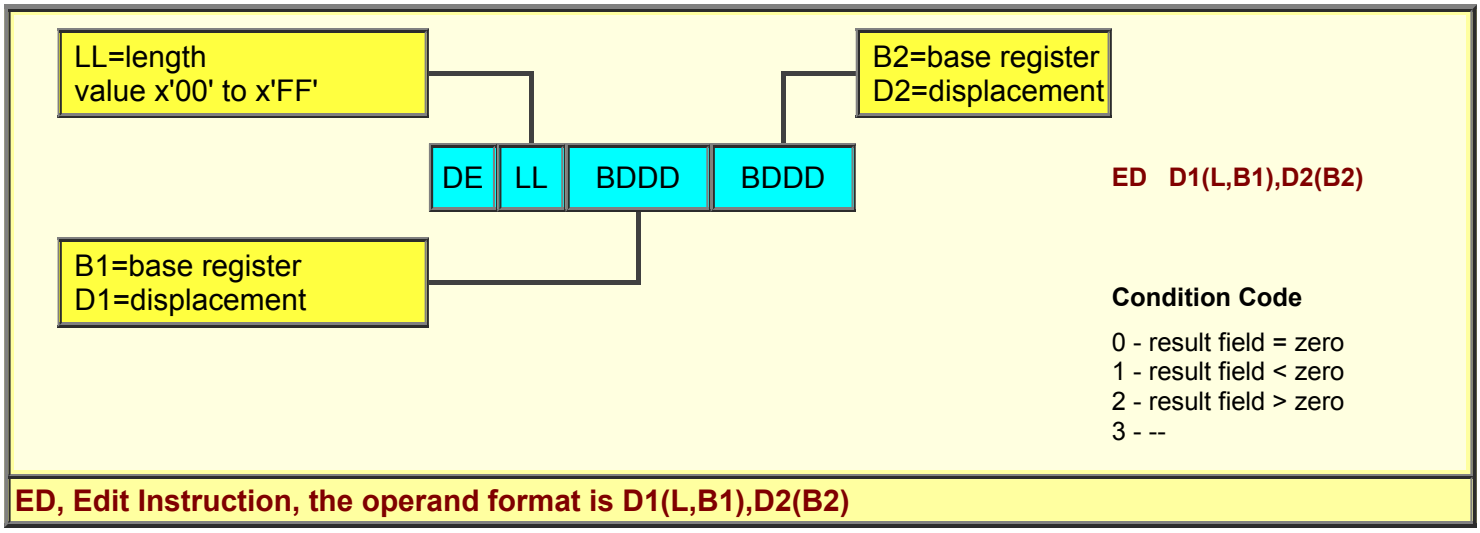
DR, Divide Registers Instruction, the operand format is R1,R2

ED, Edit

The data string located at the storage address specified by operand-2 (b2+d2) is integrated with the data string at the storage address specified by operand-1.

Operand-2 (b2+d2) remains unchanged. Operand-1 should be initialized with an edit word.

The length is determined by the length specified in the 2nd byte of the Edit instruction. The length field applies to the edit word or pattern *(the first operand)*.



ED, Edit Instruction, the operand format is D1(L,B1),D2(B2)

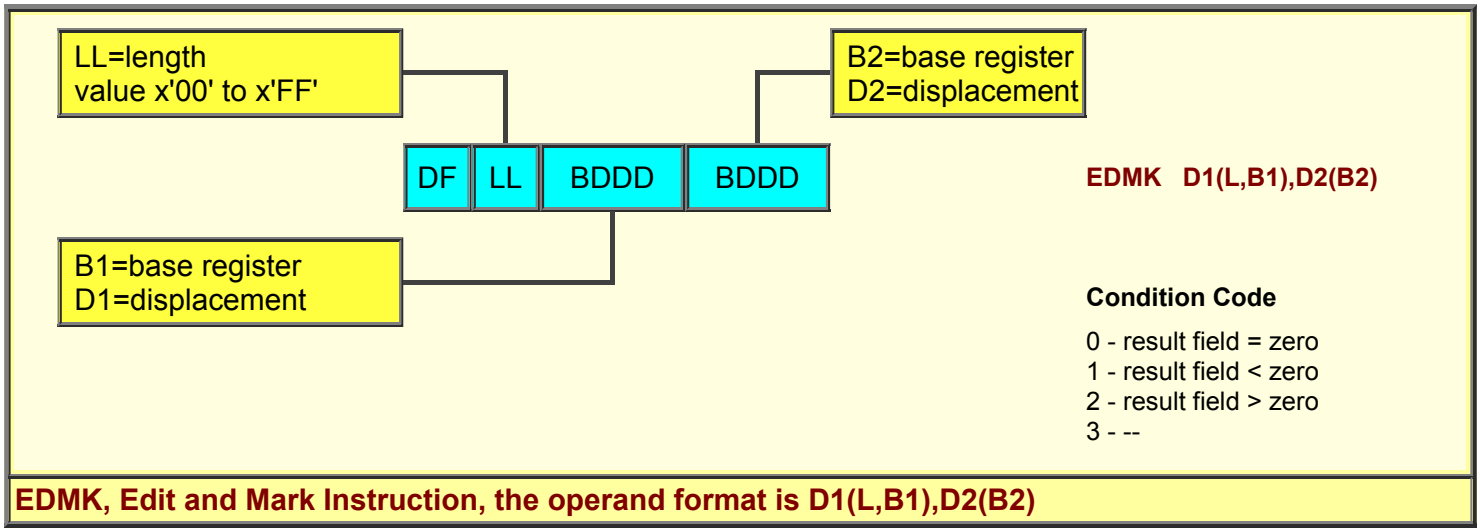
EDMK, Edit and Mark

The data string located at the storage address specified by operand-2 (b2+d2) is integrated with the data string at the storage address specified by operand-1.

The instruction is identical to ED (or Edit) instruction, except for the additional function of inserting a byte address in general register 1.

Operand-2 (b2+d2) remains unchanged. Operand-1 should be initialized with an edit word.

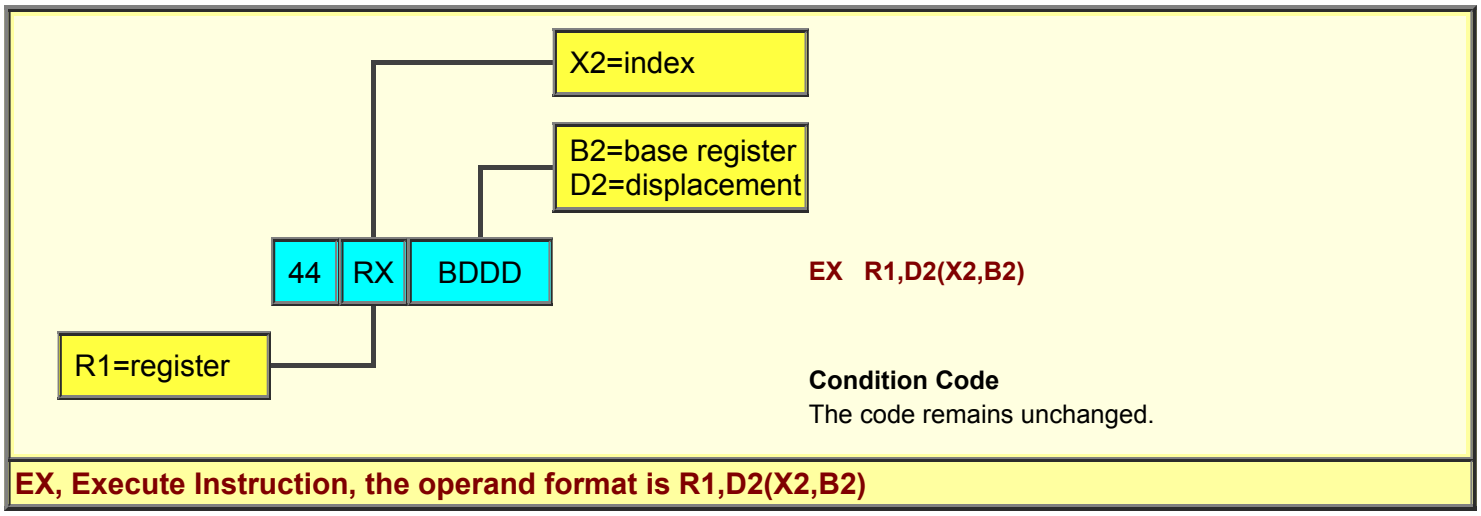
The length is determined by the length specified in the 2nd byte of the Edit instruction. The length field applies to the edit word or pattern *(the first operand)*.



EDMK, Edit and Mark Instruction, the operand format is D1(L,B1),D2(B2)

EX, Execute

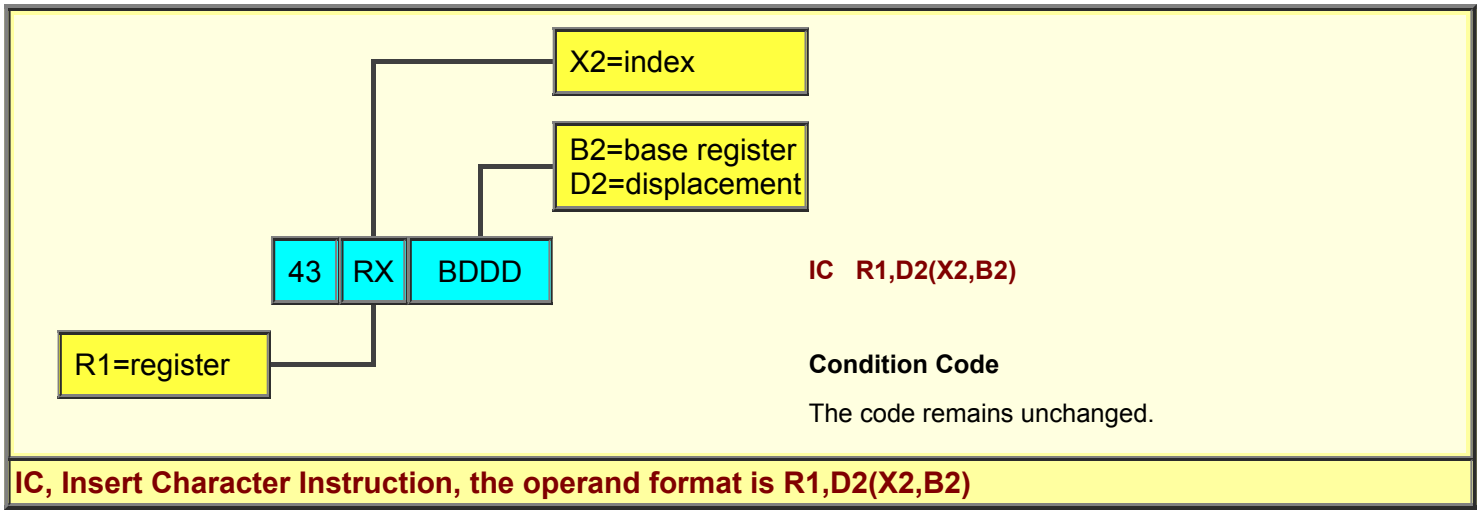
The instruction at the address specified by operand-2 (x2+b2+d2) is modified then executed using the contents of operand-1 (r1). Bits 8-15 of operand-1 and bits 24-31 of operand-1 are OR'ed together.



EX, Execute Instruction, the operand format is R1,D2(X2,B2)

IC, Insert Characters

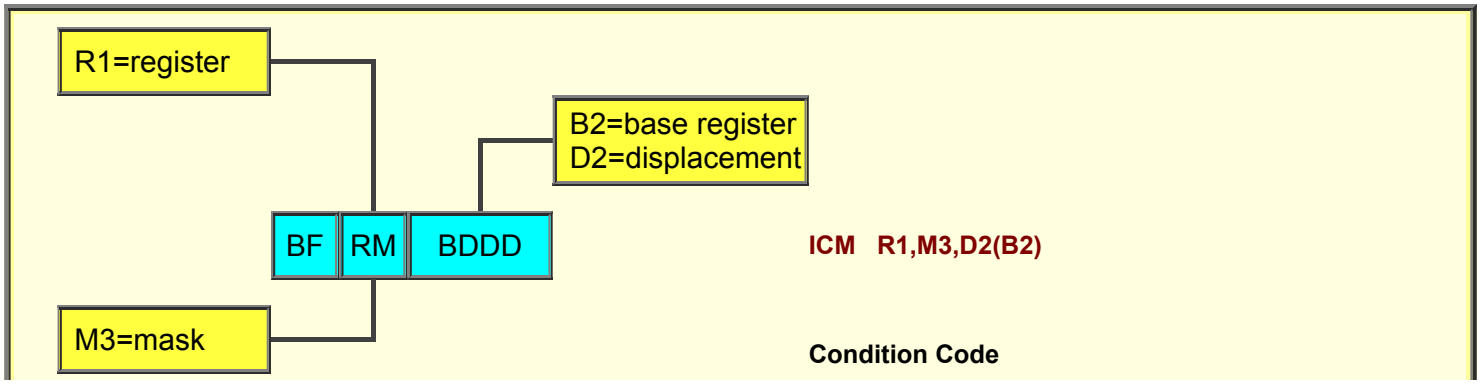
The byte at the address specified by operand-2 (x2+b2+d2) is inserted into bit positions 24-31 of operand-1 (r1).



IC, Insert Character Instruction, the operand format is R1,D2(X2,B2)

ICM, Insert Character under Mask

Bytes from storage location specified by operand-2 (b2+d2) are inserted into operand-1 (r1) under control of the mask (m3).

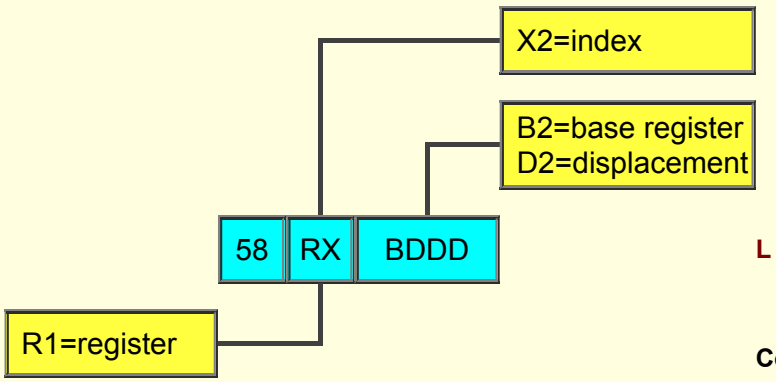


The code remains unchanged.

ICM, Insert Character under Mask Instruction, the operand format is R1,M3,D2(B2)

L, Load

The four bytes at the storage address specified by operand-2 (x2+b2+d2) are loaded into the register specified by operand-1 (r1).



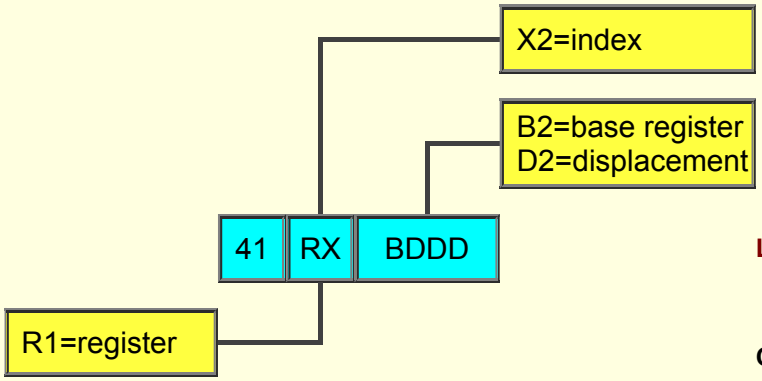
L R1,D2(X2,B2)

Condition Code
The code remains unchanged.

L, Load Instruction, the operand format is R1,D2(X2,B2)

LA, Load Address

The address of operand-2 (x2+b2+d2) is loaded into the register specified by operand-1 (r1).



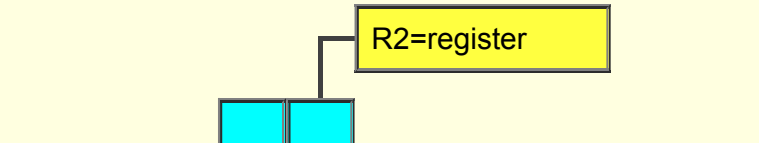
LA R1,D2(X2,B2)

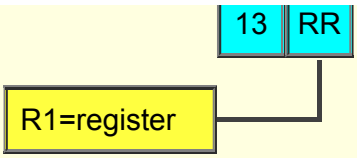
Condition Code
The code remains unchanged.

LA, Load Address Instruction, the operand format is R1,D2(X2,B2)

LCR, Load Complement Registers

The two's complement of operand-2 (r2) is put into operand-1 (r1). For example, if the register specified by operand-2 contained x'00000010' then after the LCR instruction was executed the register specified by operand-1 would contain x'FFFFFFF0'.





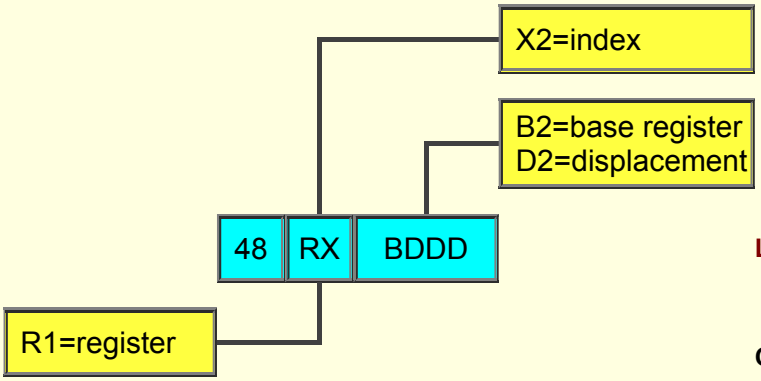
LCM R1,R2

Condition Code
The code remains unchanged.

LCR, Load Complement Registers Instruction, the operand format is R1,R2

LH, Load Halfword

The two bytes (halfword) located at the storage address specified by operand-2 (x2+b2+d2) is loaded into the rightmost two bytes (bits 16-31) of the register specified by operand-1 (r1).



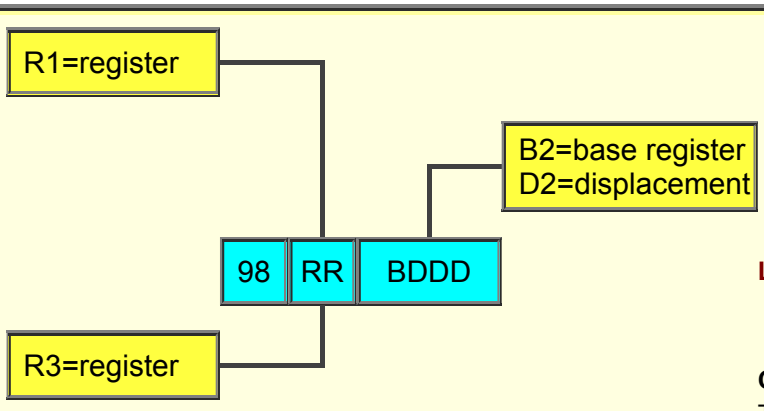
LH R1,D2(X2,B2)

Condition Code
The code remains unchanged.

LH, Load Halfword Instruction, the operand format is R1,D2(X2,B2)

LM, Load Multiples

The set of registers starting with operand-1 (r1) and ending with (r3) are loaded from the storage location specified by operand-2 (b2+d2).



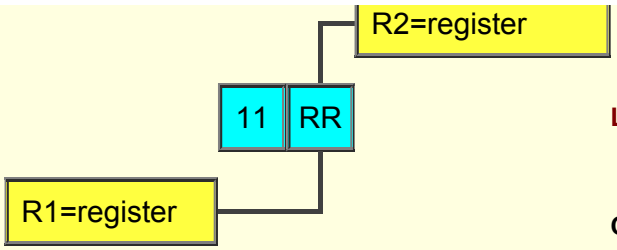
LM R1,D2(X2,B2)

Condition Code
The code remains unchanged.

LM, Load Multiples Instruction, the operand format is R1,D2(X2,B2)

LNR, Load Negative Registers

The two's complement of the absolute value of operand-2 (r2) is put into operand-1 (r1).



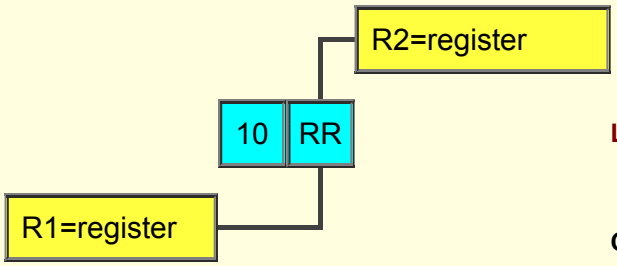
LNR R1,R2

Condition Code
The code remains unchanged.

LNR, Load Negative Registers Instruction, the operand format is R1,R2

LPR, Load Positive Registers

The absolute value of operand-2 (r2) is put into operand-1 (r1).



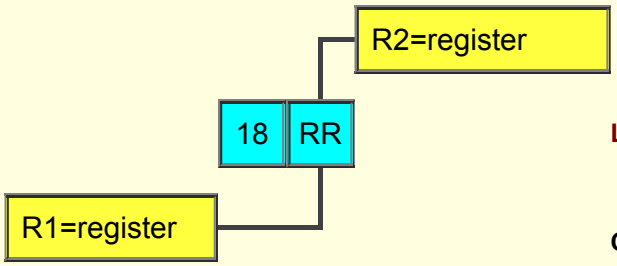
LPR R1,R2

Condition Code
The code remains unchanged.

LPR, Load Positive Registers Instruction, the operand format is R1,R2

LR, Load Register

The value of operand-2 (r2) is put into operand-1 (r1).



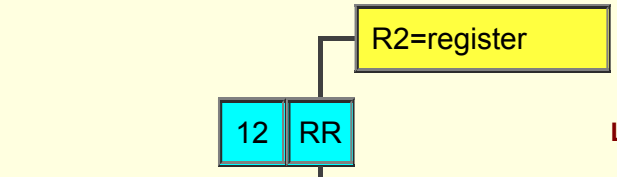
LR R1,R2

Condition Code
The code remains unchanged.

LR, Load Registers Instruction, the operand format is R1,R2

LTR, Load and Test Register

Operand-2 (r2) is put into operand-1 (r1). The sign and magnitude of operand-2 (r2), treated as a 32-bit signed binary integer are indicated in the condition code.



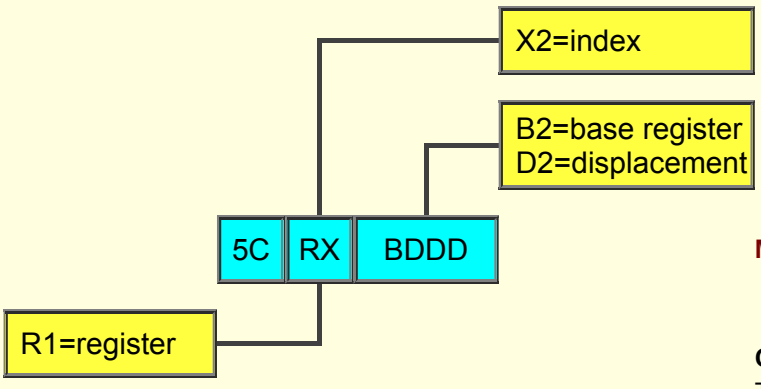
LTR R1,R2

Condition Code
0 - result = zero
1 - result < zero
2 - result > zero
3 - --

LTR, Load and Test Register Instruction, the operand format is R1,R2

M, Multiply

The value located at the storage address specified by operand-2 (x2+b2+d2) is multiplied with the 2nd-word (i.e. the second register of the pair) of operand-1 (r1). The doubleword product is put in operand-1.



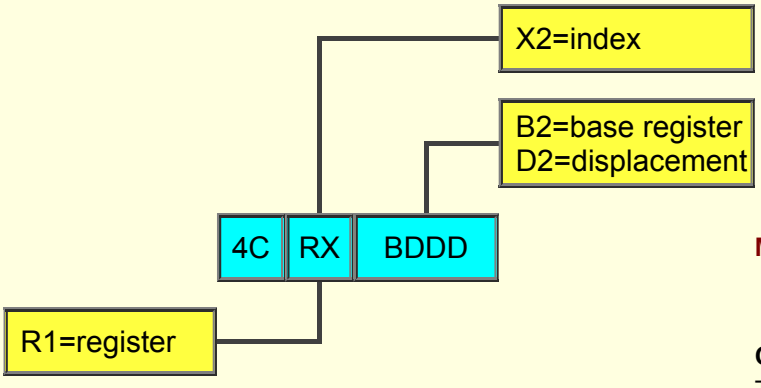
M R1,D2(X2,B2)

Condition Code
The code remains unchanged

M, Multiply Instruction, the operand format is R1,D2(X2,B2)

MH, Multiply Halfword

The value located at the storage address specified by operand-2 (x2+b2+d2) is multiplied with operand-1 (r1). The product is put in operand-1.



MH R1,D2(X2,B2)

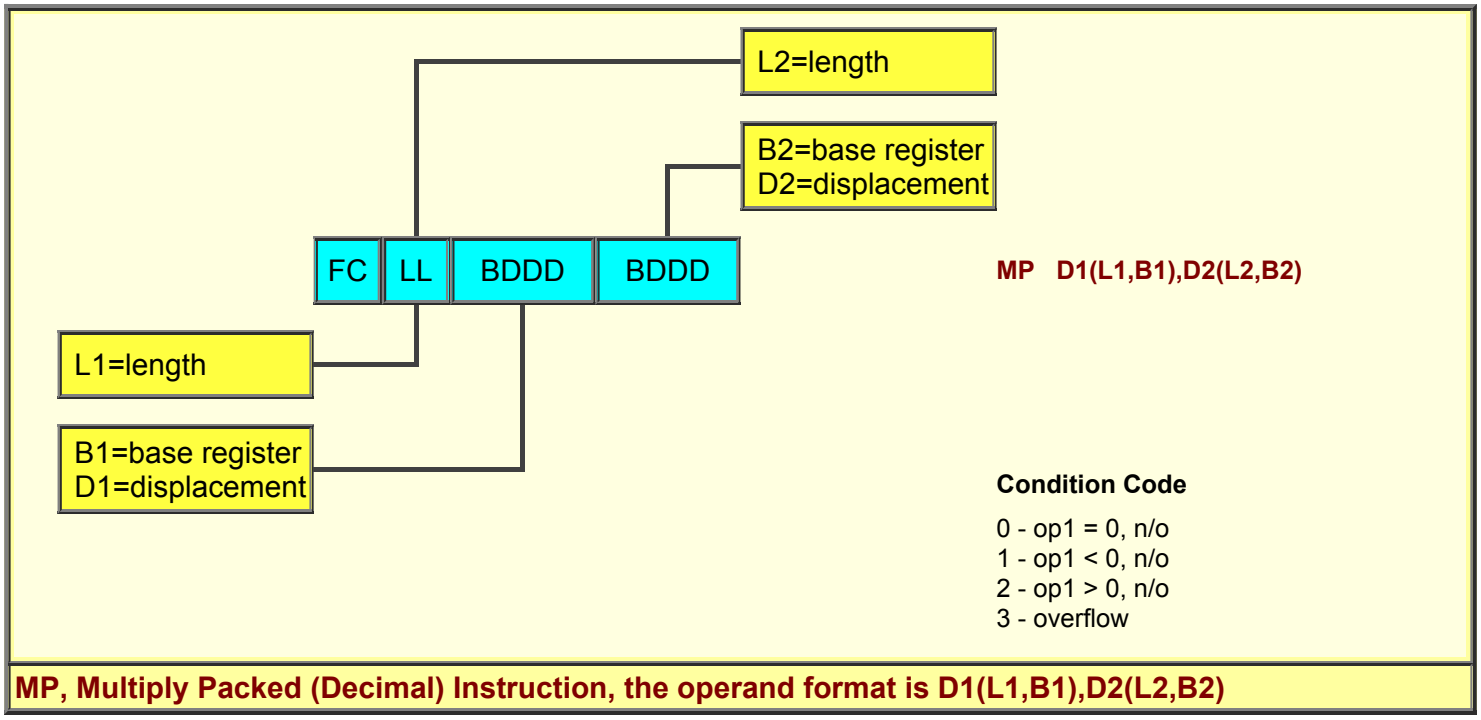
Condition Code
The code remains unchanged

MH, Multiply Halfword Instruction, the operand format is R1,D2(X2,B2)

MP, Multiply Packed (Decimal)

The product of the data string located at the storage address specified by operand-2 (b2+d2) and the data string located at the storage address specified by operand-1 (b1+d1) is placed in the operand-1 location. Operand-2 remains unchanged.

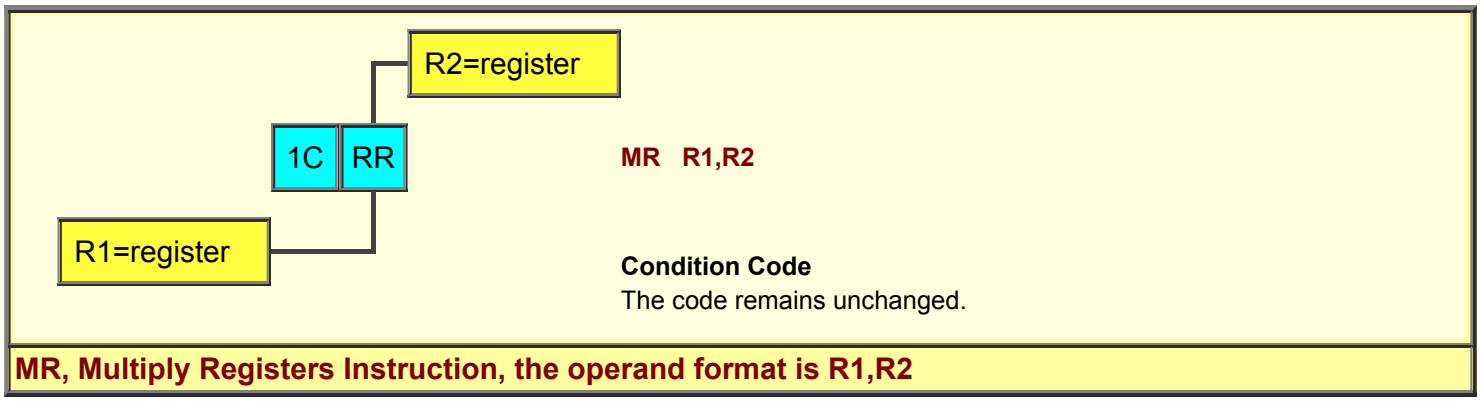
The operands may be different lengths with a maximum length of 16 bytes (or 31 digits since this is packed) for each operand. The condition code is set as shown below.



MP, Multiply Packed (Decimal) Instruction, the operand format is D1(L1,B1),D2(L2,B2)

MR, Multiply Registers

The second word (odd-register) of operand-1 (r1) is multiplied by operand-2 (r2), the doubleword product is put in operand-1 (r1, even/odd-pair).

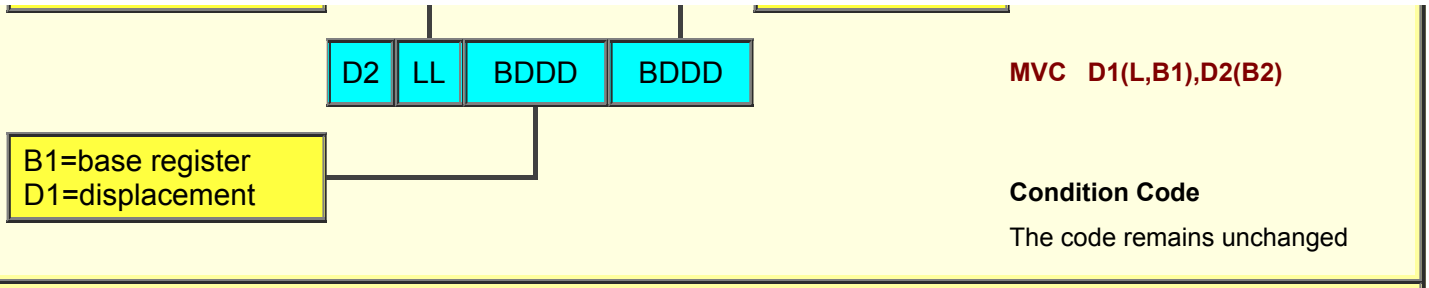


MR, Multiply Registers Instruction, the operand format is R1,R2

MVC, Move Characters

The data string located at the storage address specified by operand-2 (b2+d2) is moved to the storage address specified by operand-1 (b1+d1).

Operand-2 remains unchanged. The number of bytes moved is determined by the length specified in the 2nd byte of the MVC instruction.

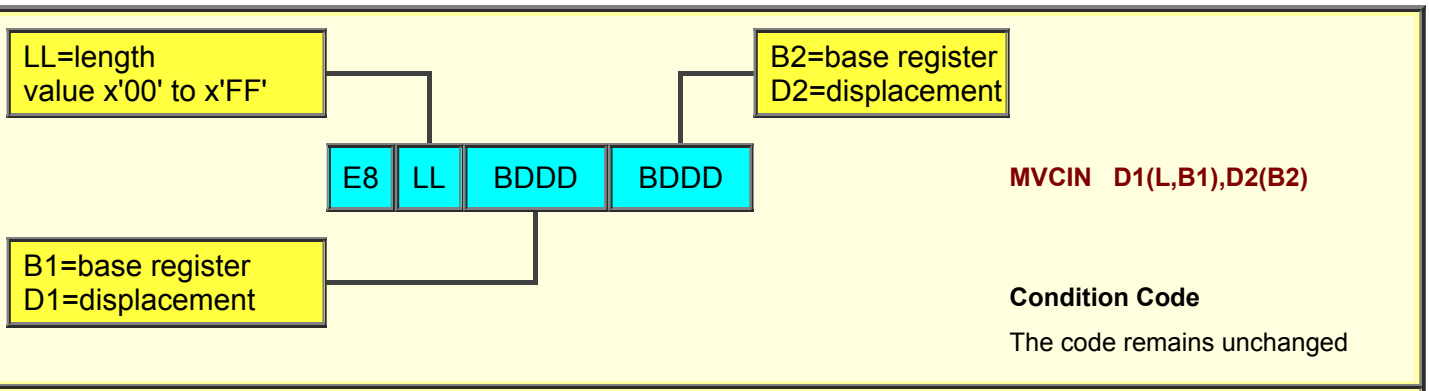


MVC, Move Characters Instruction, the operand format is D1(L,B1),D2(B2)

MVCIN, Move Characters Inverse

The data string located at the storage address specified by operand-2 (b2+d2) is moved to the storage address specified by operand-1 (b1+d1) with the left-to-right sequence of the bytes inverted.

Operand-2 remains unchanged. The number of bytes moved is determined by the length specified in the 2nd byte of the MVC instruction.



MVCIN, Move Characters Inverse Instruction, the operand format is D1(L,B1),D2(B2)

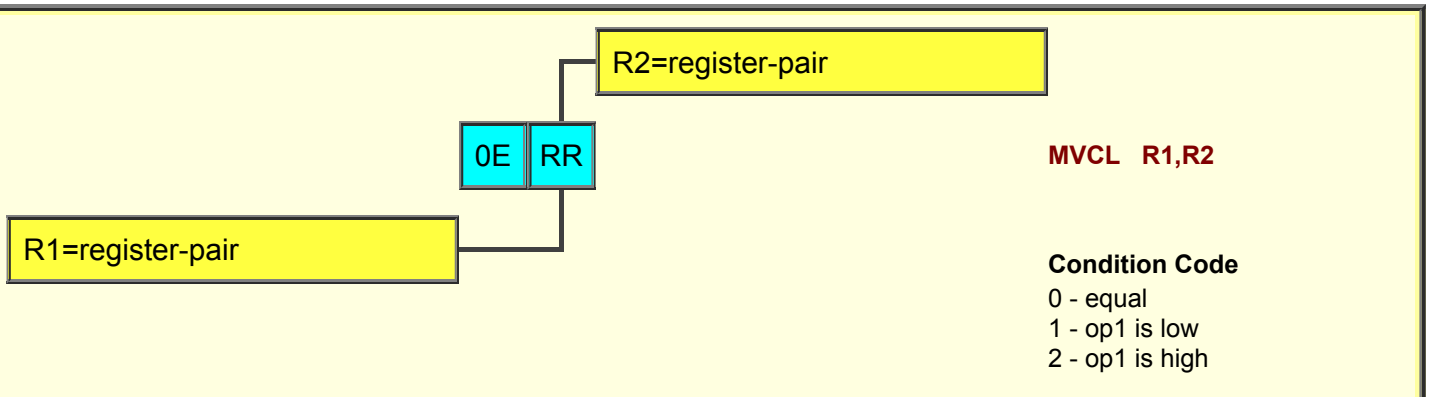
MVCL, Move Characters Long

The data string identified by operand-2 (r2) is moved to the storage location identified by operand-1 (r1). Both operands are register-pairs.

The first register of each operand needs to be loaded with the storage addresses of the data strings. The second register of each operands needs to be loaded with the length of each operand.

The data strings may be different length. The high-order byte of the second register of operand-2 is treated as the padding character if the operands or different lengths.

The move proceeds from left to right, low storage to high storage and as each character is moved the length registers are decremented.

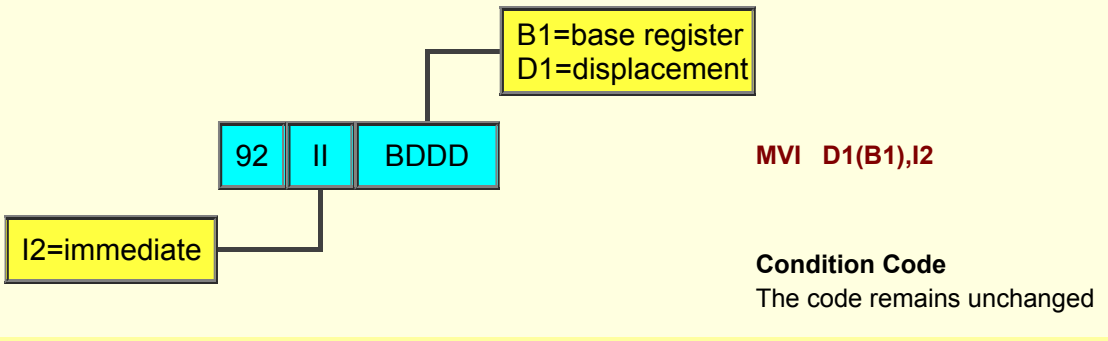


3 -

MVCL, Move Characters Long Instruction, the operand format is R1,R2

MVI, Move Immediate

Operand-2 *(the immediate data that is the second byte of the instruction itself)* is moved to the storage address specifed by operand-1 (b1+d1).

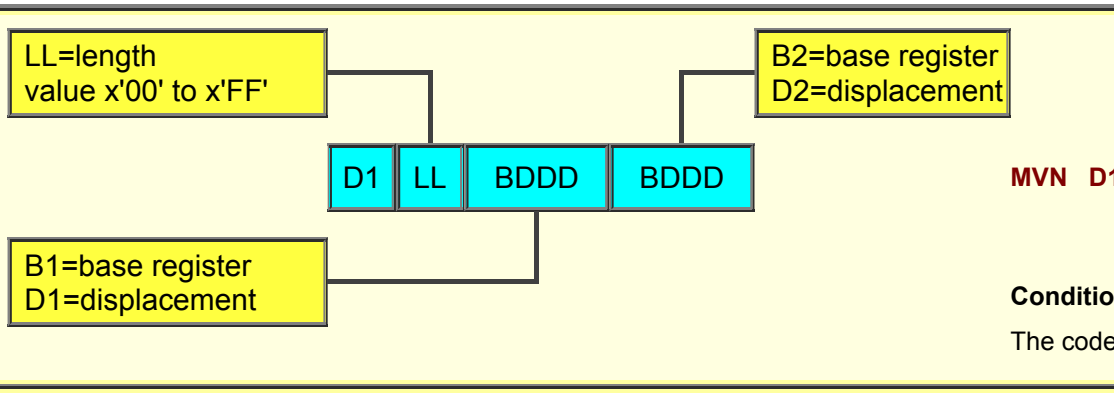


MVI, Move Immediate Instruction, the operand format is D1(B1),I2

MVN, Move Numerics

The rightmost 4-bits of each byte of the data string located at the storage address specified by operand-2 (b2+d2) are put into the storage address specified by operand-1 (b1+d1).

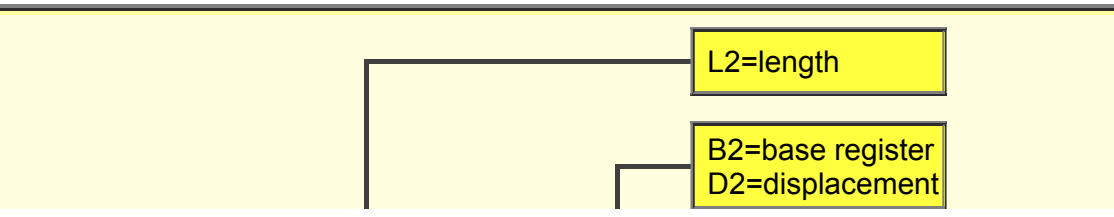
Operand-2 remains unchanged. The length is determined by the value in the 2nd byte of the MVN instruction.

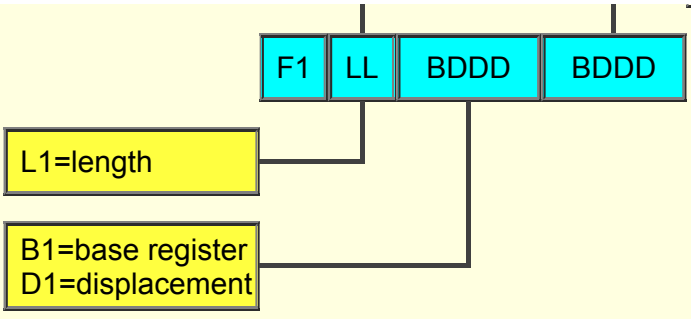


MVN, Move Numerics Instruction, the operand format is D1(L,B1),D2(B2)

MVO, Move with Offset

The data string located at the storage address specified by operand-2 (b2+d2) is shifted left four bits and put into the storage address specified by operand-1 (b1+d1). The rightmost 4-bits of operand-1 remain unchanged.





MVO D1(L1,B1),D2(L2,B2)

Condition Code

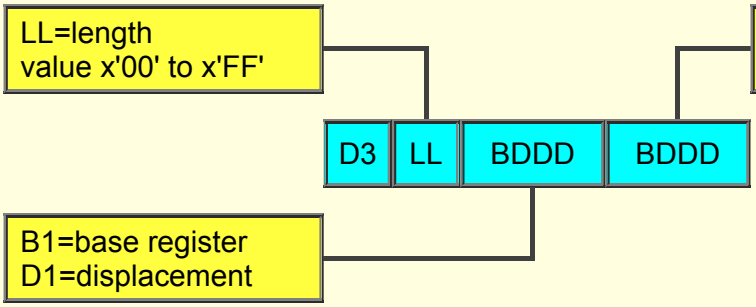
The code remains unchanged

MVO, Move with Offset Instruction, the operand format is D1(L1,B1),D2(L2,B2)

MVZ, Move Zones

The lefmost 4-bits of each byte of the data string located at the storage address specified by operand-2 (b2+d2) are put into the storage address specified by operand-1 (b1+d1).

Operand-2 remains unchanged. The length is determined by the value in the 2nd byte of the MVZ instruction.



MVZ D1(L,B1),D2(B2)

Condition Code

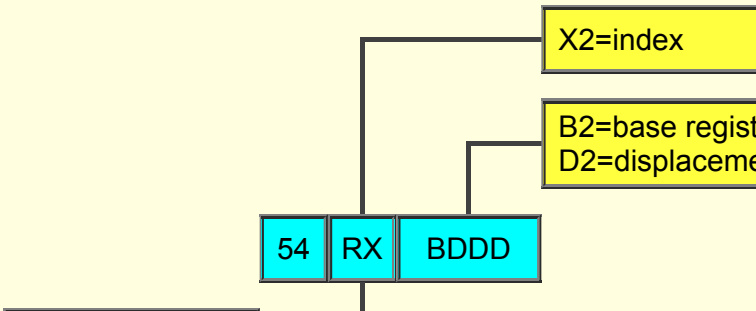
The code remains unchanged

MVZ, Move Zones Instruction, the operand format is D1(L,B1),D2(B2)

N, And

The content of operand-1 (r1) is AND'ed with the data string located at the storage address specified by operand-2 (x2+b2+d2).

The results of the AND'ing process is put into operand-1.



N R1,D2(X2,B2)

Condition Code

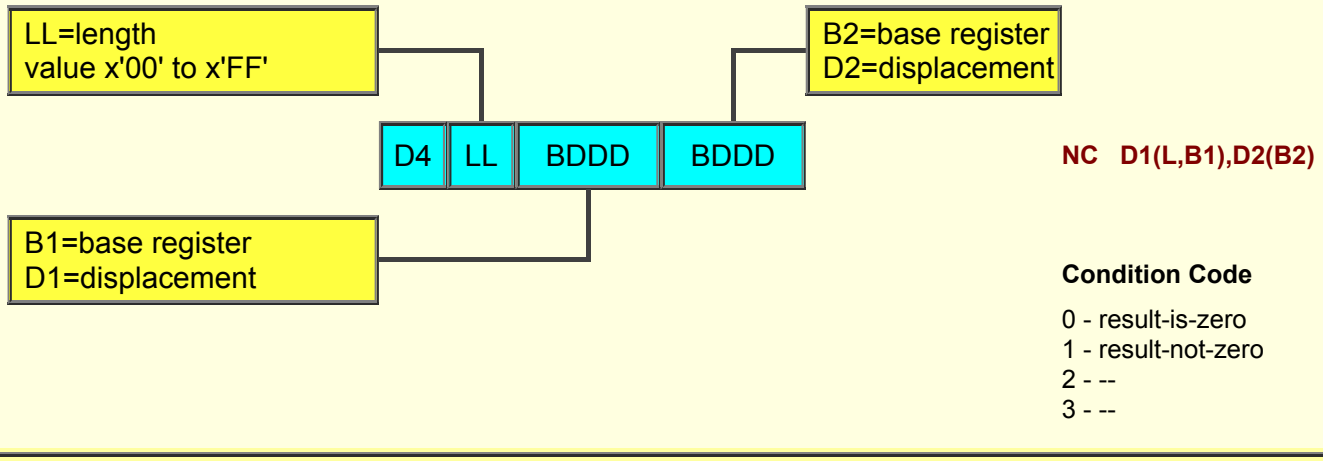
0 -result-is-zero
1 - result-not-zero
2 - --
3 - --

N, And Instruction, the operand format is R1,D2(X2,B2)

NC, And Characters

The data string located at the storage address specified by operand-1 (b1+d1) is AND'ed with the data string located at the storage address specified by operand-2 (b2+d2).

The results of the exclusive AND'ing process is put into operand-1.

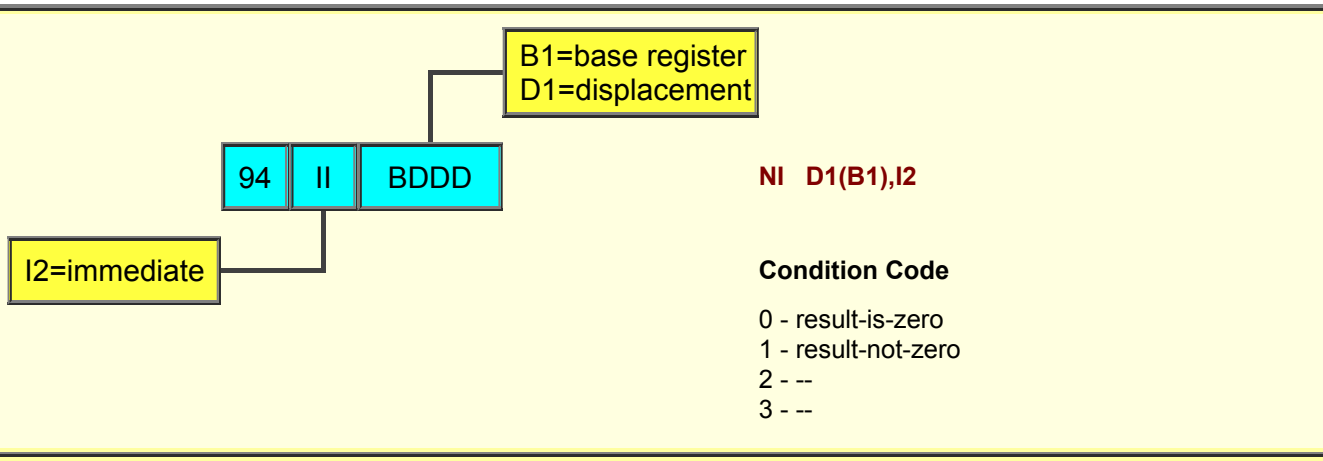


NC, And Characters Instruction, the operand format is D1(L,B1),D2(B2)

NI, And Immediate

The data string located at the storage address specified by operand-1 (b1+d1) is AND'ed with operand-2 *(i2 is immediate data included as the second byte of the instruction itself)*.

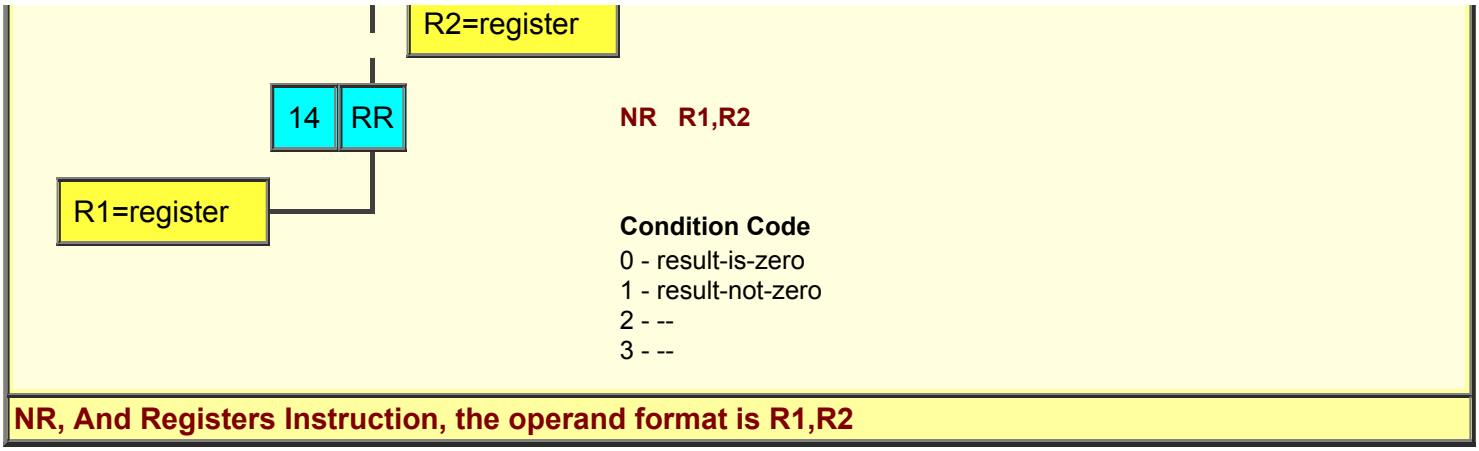
The results of the AND'ing process is put into operand-1.



NI, And Immediate Instruction, the operand format is D1(B1),I2

NR, And Registers

The content of operand-1 (r1) is AND'ed with the content of operand-2 (r2). The results of the AND'ing process is put into operand-1.

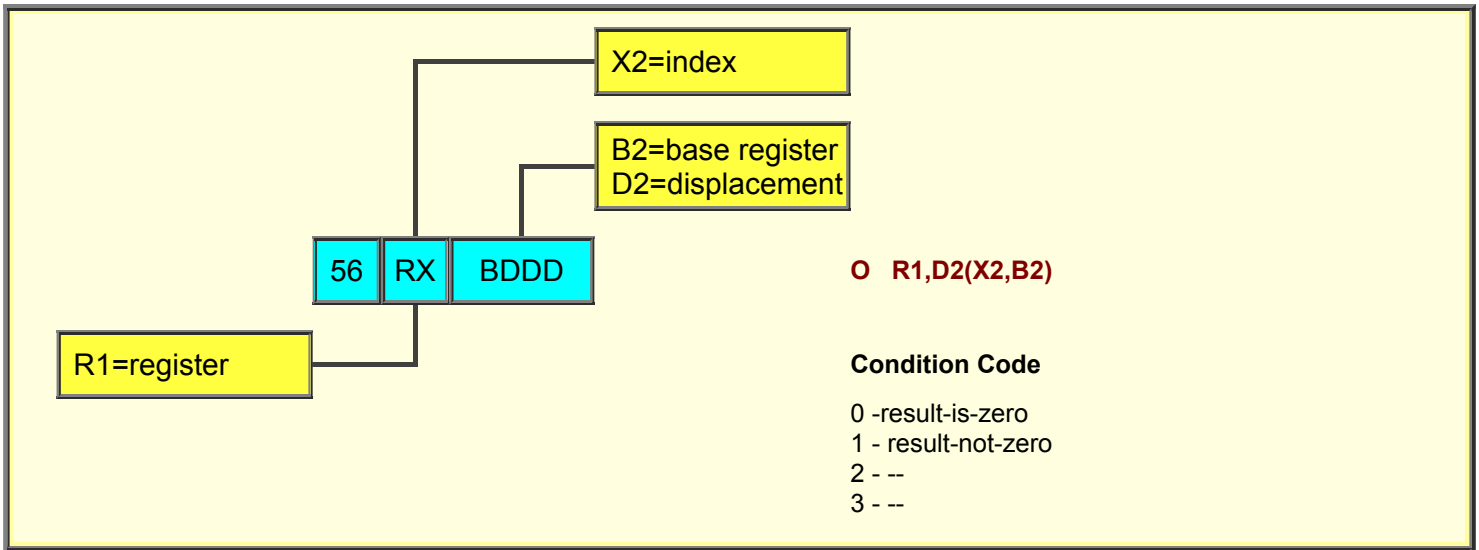


NR, And Registers Instruction, the operand format is R1,R2

O, Or

The content of operand-1 (r1) is OR'ed with the data string located at the storage address specified by operand-2 (x2+b2+d2).

The results of the OR'ing process is put into operand-1.

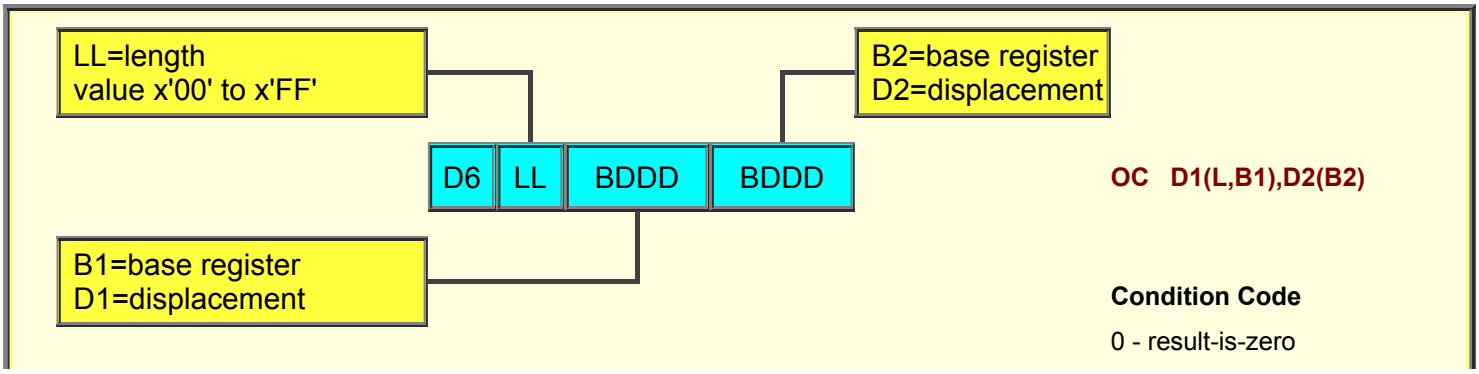


O, Or Instruction, the operand format is R1,D2(X2,B2)

OC, Or Characters

The data string located at the storage address specified by operand-1 (b1+d1) is OR'ed with the data string located at the storage address specified by operand-2 (b2+d2).

The results of the OR'ing process is put into operand-1.



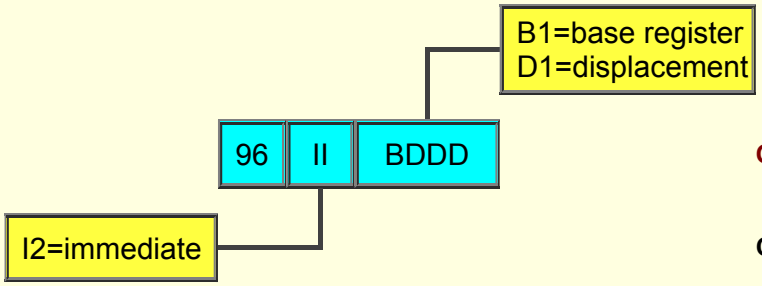
1 - result-not-zero
2 - --
3 - --

OC, Or Characters Instruction, the operand format is D1(L,B1),D2(B2)

OI, Or Immediate

The data string located at the storage address specified by operand-1 (b1+d1) is OR'ed with operand-2 *(i2 is immediate data included as the second byte of the instruction itself)*.

The results of the OR'ing process is put into operand-1.



OI D1(B1),I2

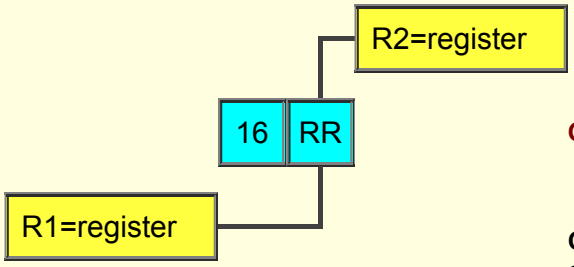
Condition Code

0 - result-is-zero
1 - result-not-zero
2 - --
3 - --

OI, Or Immediate Instruction, the operand format is D1(B1),I2

OR, Or Registers

The content of operand-1 (r1) is OR'ed with the content of operand-2 (r2). The results of the OR'ing process is put into operand-1.



OR R1,R2

Condition Code

0 - result-is-zero
1 - result-not-zero
2 - --
3 - --

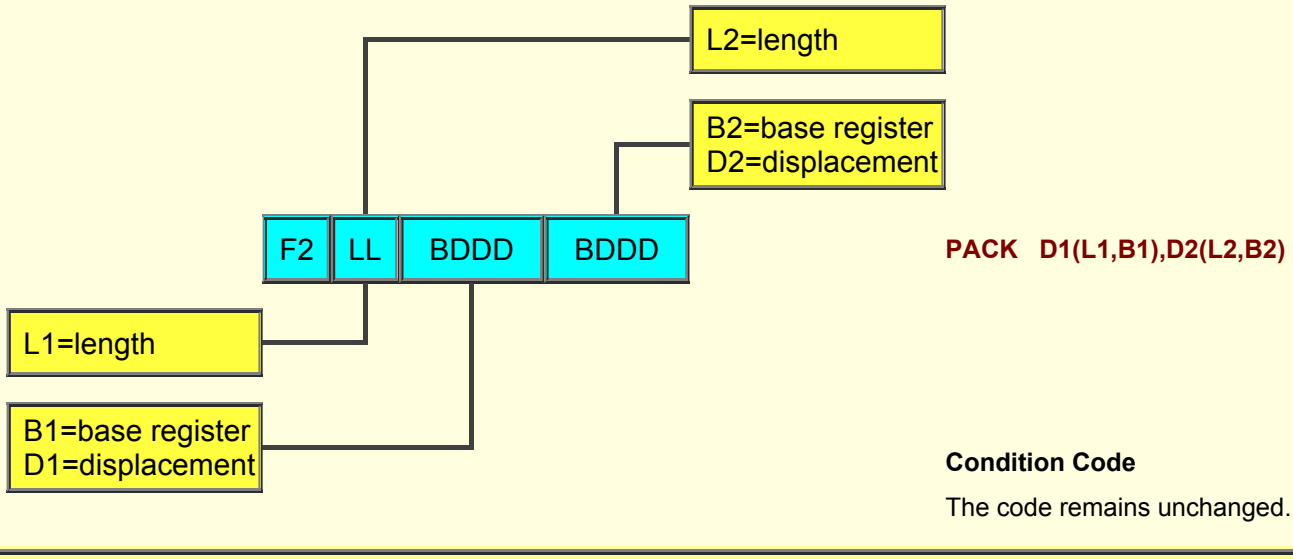
OR, Or Registers Instruction, the operand format is R1,R2

PACK, Pack

The data string located at the storage address specified by operand-2 (b2+d2) is changed from zoned-decimal to packed and the result is put into the storage address specified by operand-1 (b1+d1).

Operand-2 remains unchanged. The operands may be different lengths with a maximum length of 16

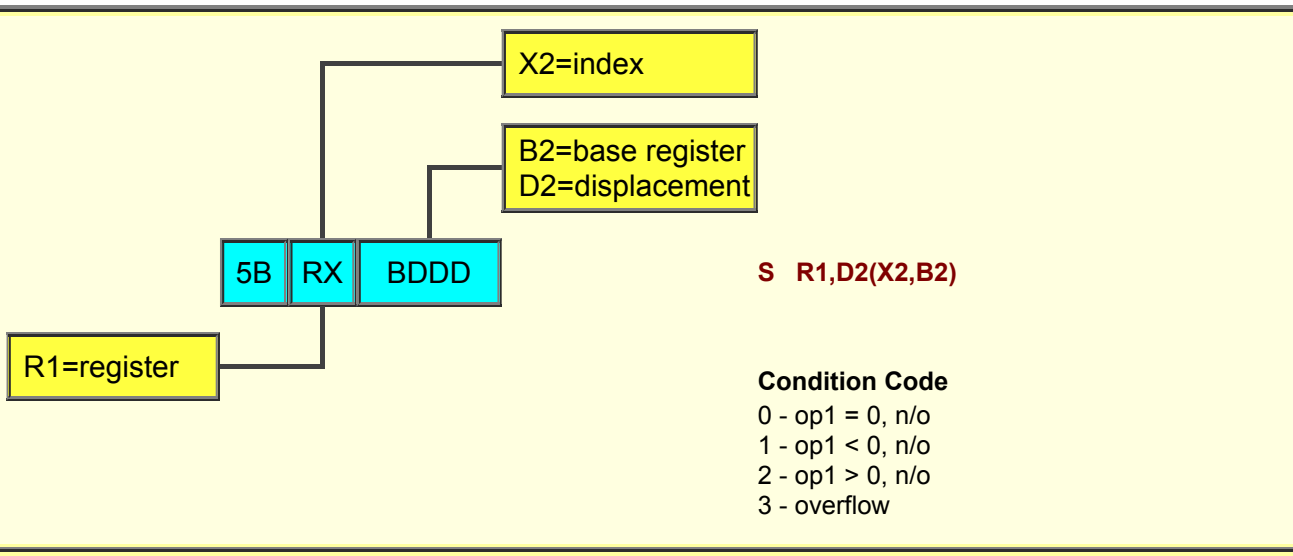
bytes (or 31 digits since this is packed) for each operand.



PACK, Pack (Decimal) Instruction, the operand format is D1(L1,B1),D2(L2,B2)

S, Subtract

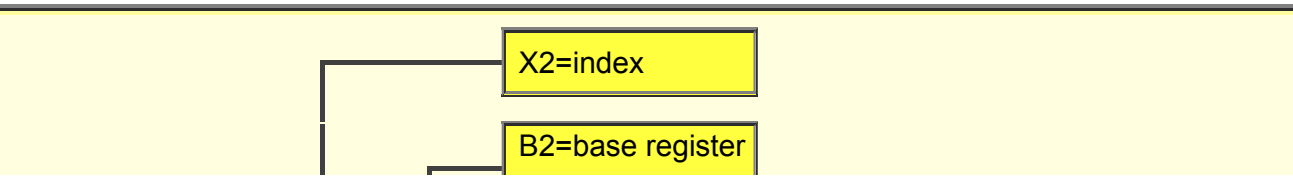
The full word (4 bytes) located at the storage address specified by operand-2 (x2+b2+d2} is subtracted from the register specified by operand-1(r1). Operand-2 remains unchanged. The condition code is set as shown below.

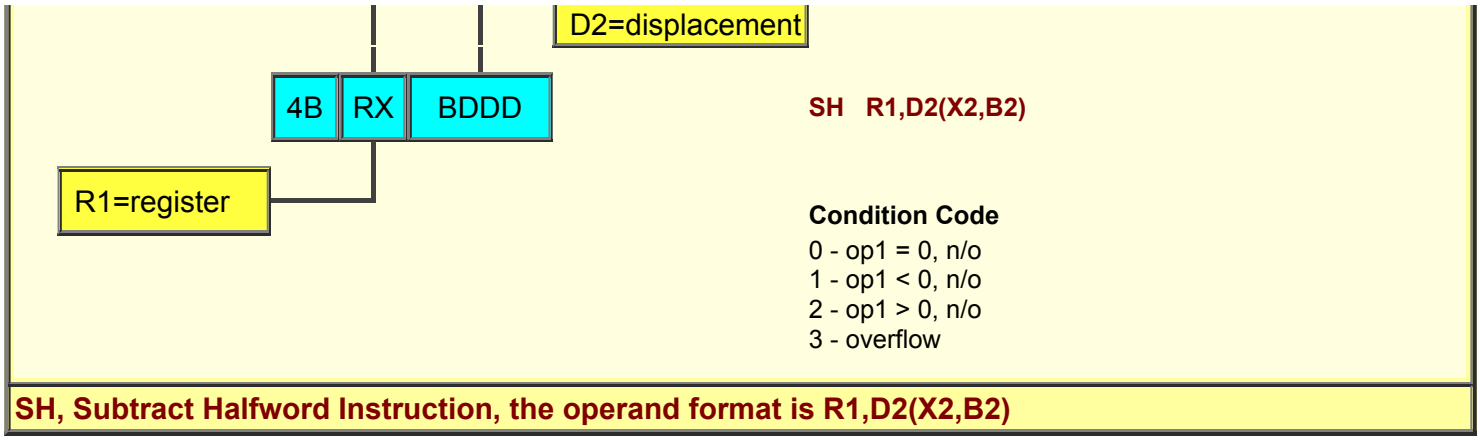


S, Subtract Instruction, the operand format is R1,D2(X2,B2)

SH, Subtract Halfword

The half word (2 bytes) located at the storage address specified by operand-2 (x2+b2+d2) is subtracted from the register specified by operand-1 (r1). Operand-2 remains unchanged. The condition code is set as shown below.

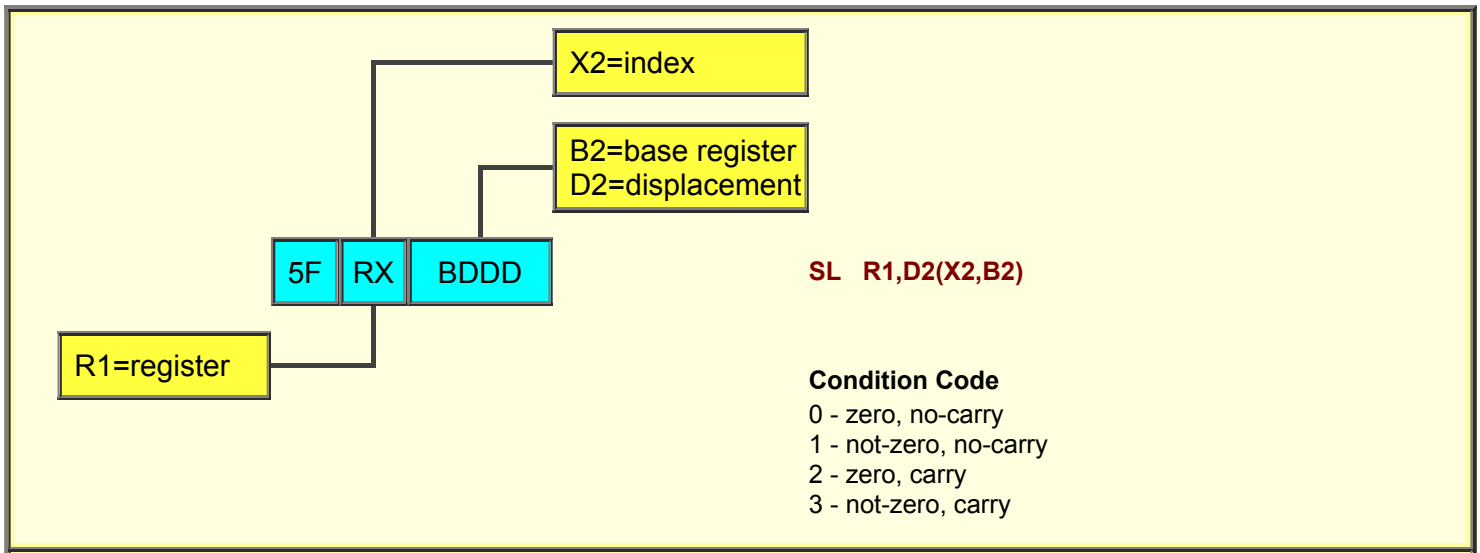




SH, Subtract Halfword Instruction, the operand format is R1,D2(X2,B2)

SL, Subtract Logical

The full word (4 bytes) located at the storage address specified by operand-2 (x2+b2+d2) is subtracted from the register specified by operand-1 (r1). Operand-2 remains unchanged. The condition code is set as shown below.

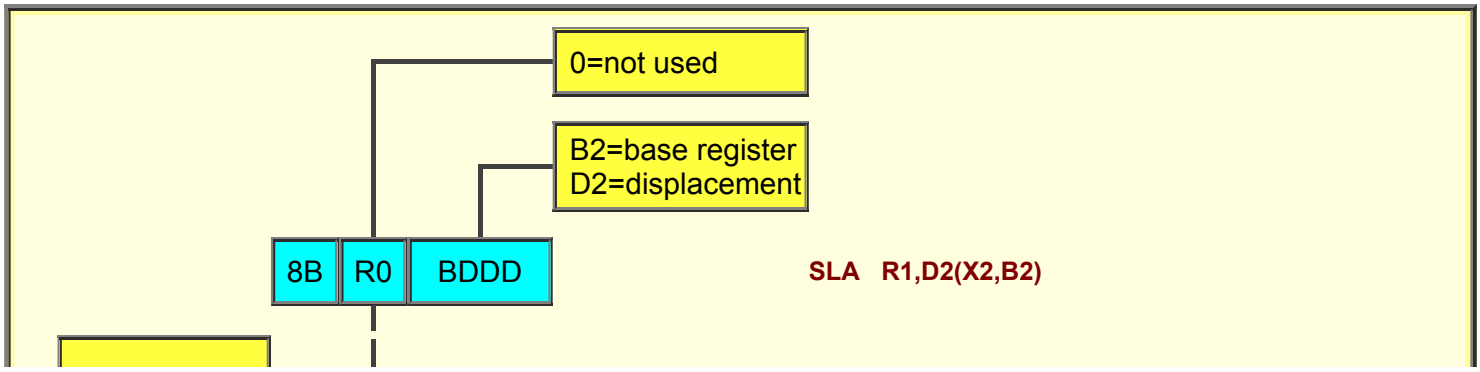


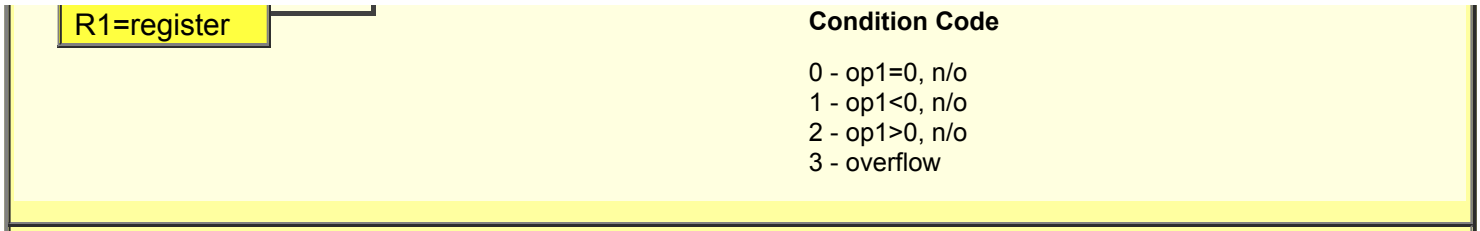
SL, Subtract Logical Instruction, the operand format is R1,D2(X2,B2)

SLA, Shift Left Single

The 31-bit numeric part (bit-0 is the sign and bits 1-31 are the numerics) of operand-1 (r1) is shifted left the number of bits specified by operand-2 (b2+d2).

With this instruction operand-2 (b2+d2) does not address storage. Bits 0-25 are ignored, bits 26-31 specify the number of bit positions to be shifted.





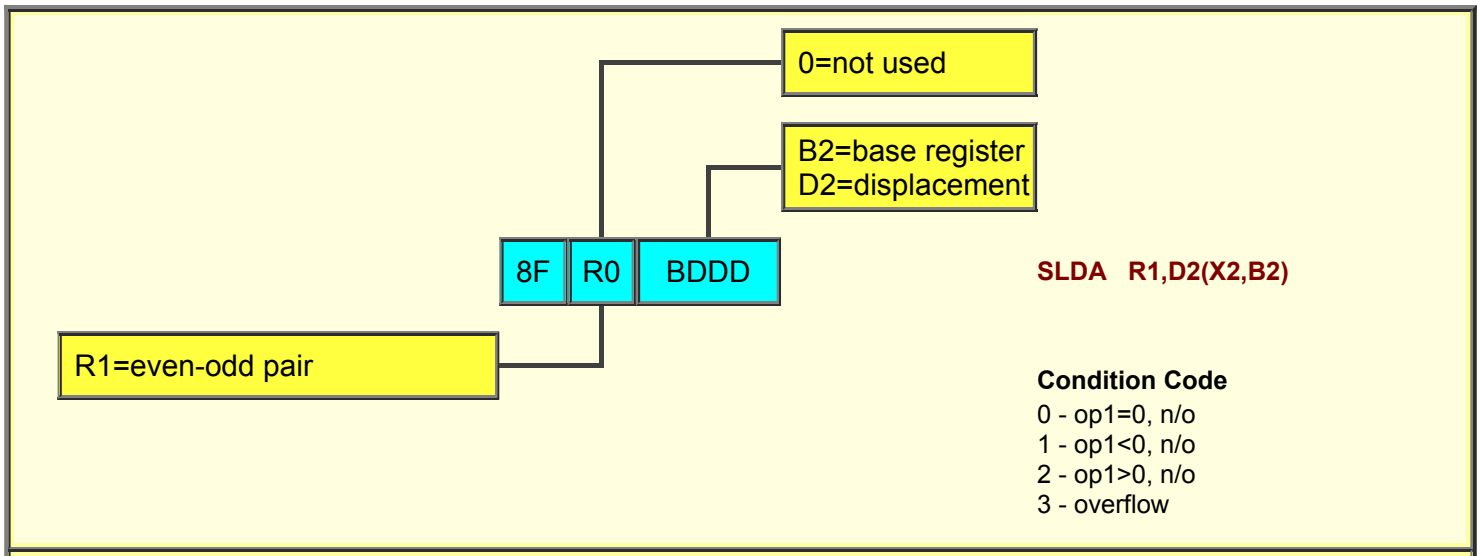
SLA, Shift Left Single Instruction, the operand format is R1,D2(X2,B2)

SLDA, Shift Left Double

The 31-bit numeric part (bit-0 is the sign and bits 1-31 are the numerics) of operand-1 (r1) is shifted left the number of bits specified by operand-2 (b2+d2).

With this instruction operand-2 (b2+d2) does not address storage. Bits 0-25 are ignored, bits 26-31 specify the number of bit positions to be shifted.

Note: *If an ODD/EVEN pair of registers is specified then an 0C6 error will occur*



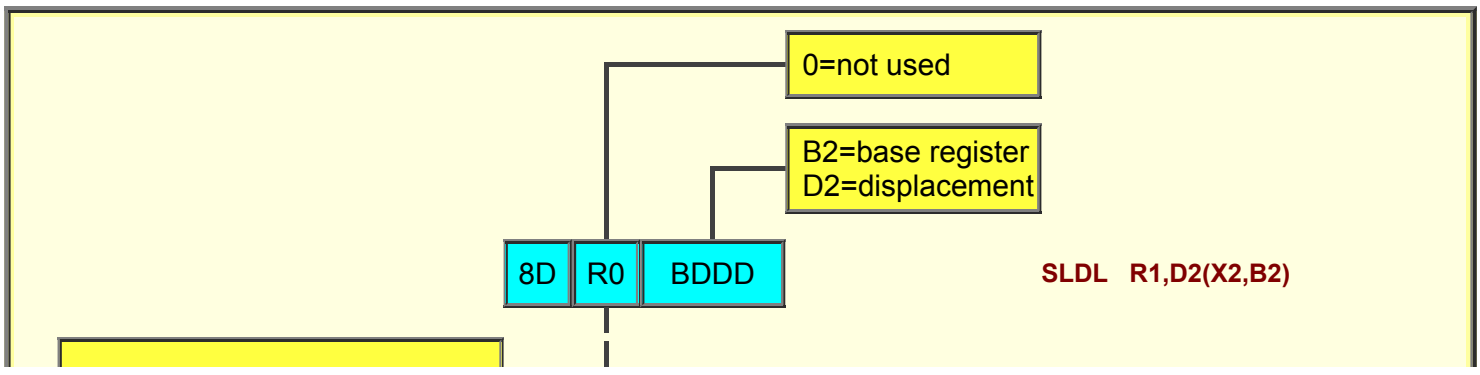
SLDA, Shift Left Double Instruction, the operand format is R1,D2(X2,B2)

SLDL, Shift Left Double Logical

The 64-bits of operand-1 (r1 is an EVEN/ODD pair of registers) is shifted left the number of bits specified by operand-2 (b2+d2).

With this instruction operand-2 (b2+d2) does not address storage. Bits 0-25 are ignored, bits 26-31 specify the number of bit positions to be shifted.

Note: *If an ODD/EVEN pair of registers is specified then an 0C6 error will occur*



Condition Code

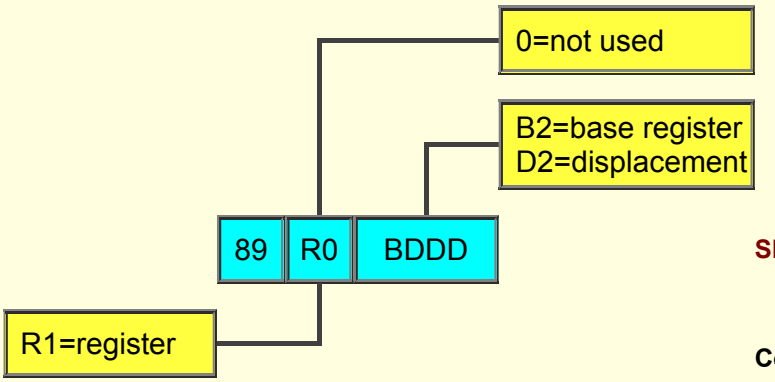
0 - op1=0, n/o
1 - op1<0, n/o
2 - op1>0, n/o
3 - overflow

SLDL, Shift Left Double Logical Instruction, the operand format is R1,D2(X2,B2)

SLL, Shift Left Single Logical

The 32-bits of operand-1 (r1) is shifted left the number of bits specified by operand-2 (b2+d2).

With this instruction operand-2 (b2+d2) does not address storage. Bits 0-25 are ignored, bits 26-31 specify the number of bit positions to be shifted.



SLL R1,D2(X2,B2)

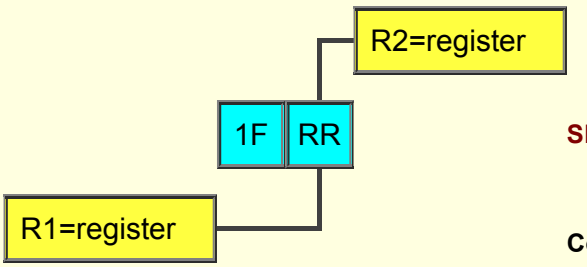
Condition Code

0 - op1=0, n/o
1 - op1<0, n/o
2 - op1>0, n/o
3 - overflow

SLL, Shift Left Single Logical Instruction, the operand format is R1,D2(X2,B2)

SLR, Subtract Logical Registers

The register specified by operand-2 (r2) is subtracted from the register specified by operand-1 (r1). Operand-2 remains unchanged. The condition code is set as shown below.



SLR R1,R2

Condition Code

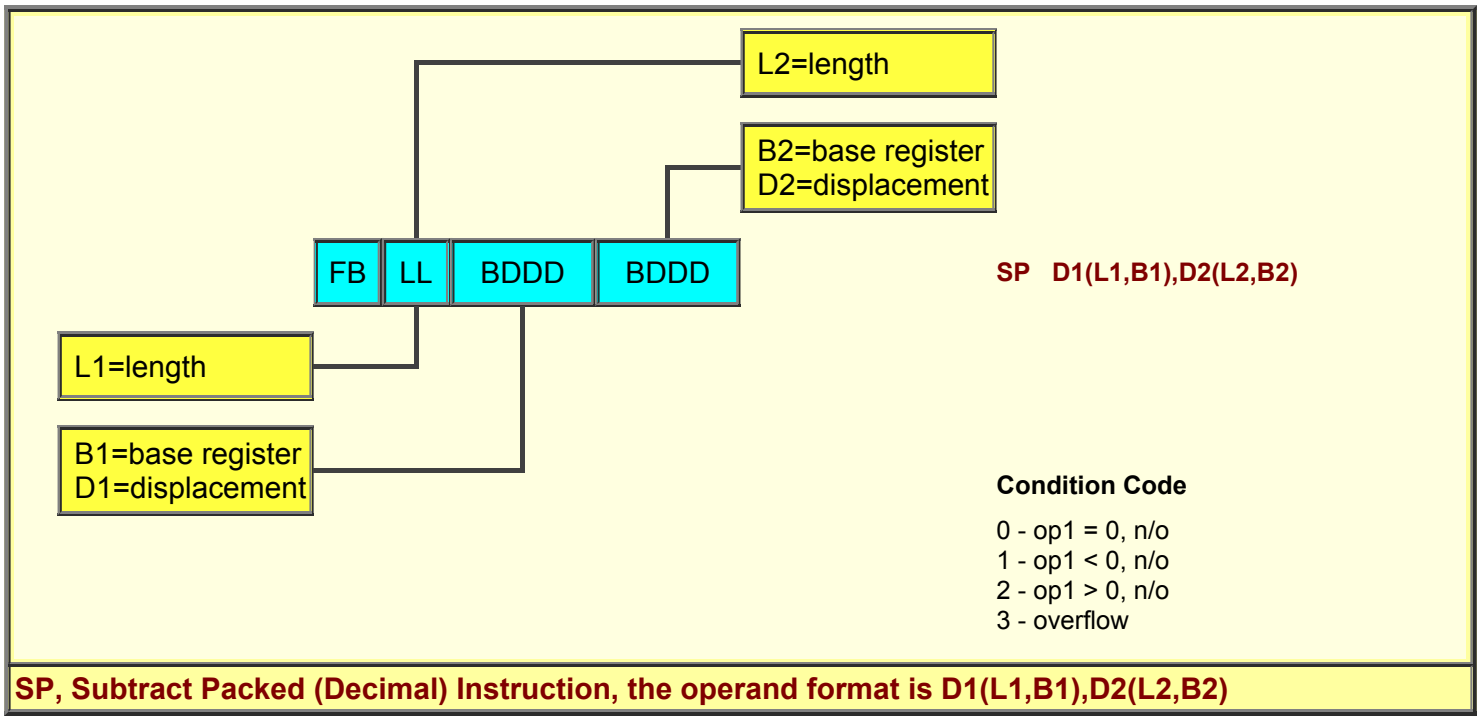
0 - zero, no-carry
1 - not-zero, no-carry
2 - zero, carry
3 - not-zero, carry

SLR, Subtract Logical Registers Instruction, the operand format is R1,R2

SP, Subtract Packed (Decimal)

The data string located at the storage address specified by operand-2 (b2+d2) is subtracted from the data string located at the storage address specified by operand-1 (b1+d1). Operand-2 remains unchanged.

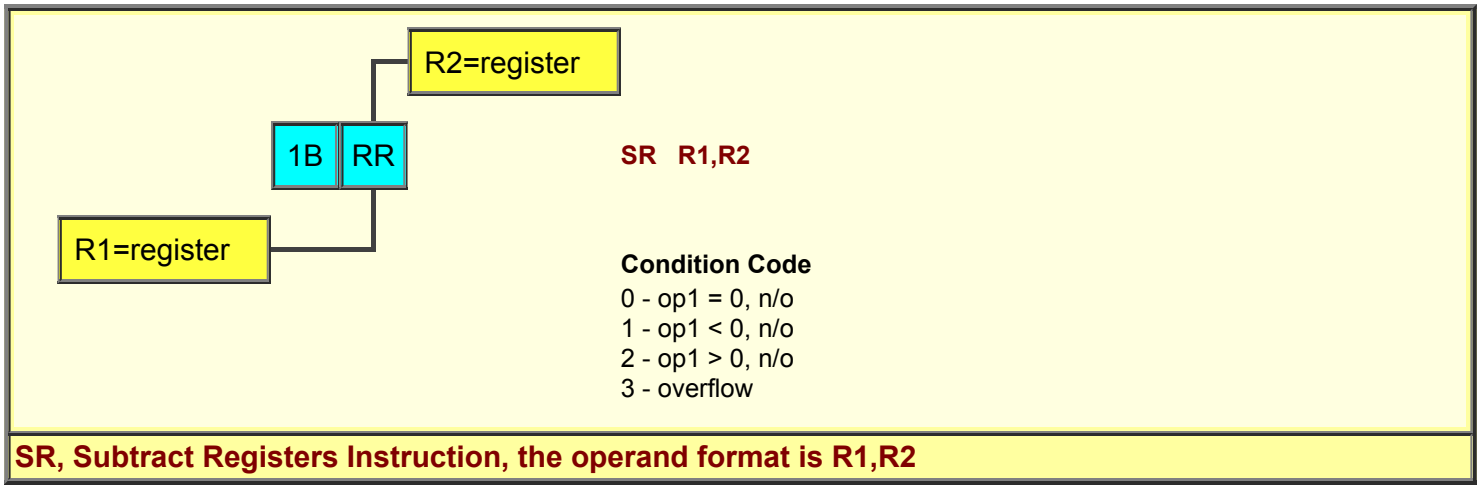
The operands may be different lengths with a maximum length of 16 bytes *(or 31 digits since this is packed)* for each operand. The condition code is set as shown below.



SP, Subtract Packed (Decimal) Instruction, the operand format is D1(L1,B1),D2(L2,B2)

SR, Subtract Registers

The register specified by operand-2 (r2) is subtracted from the register specified by operand-1 (r1). Operand-2 remains unchanged. The condition code is set as shown below.



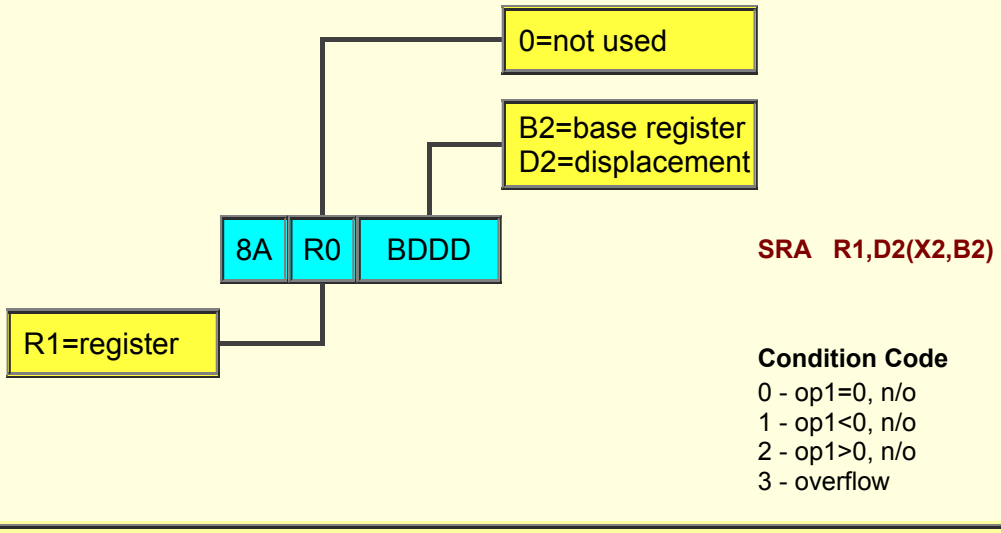
SR, Subtract Registers Instruction, the operand format is R1,R2

SRA, Shift Right Single

The 31-bit numeric part (bit-0 is the sign and bits 1-31 are the numerics) of operand-1 (r1) is shifted right the number of bits specified by operand-2 (b2+d2).

With this instruction operand-2 (b2+d2) does not address storage. Bits 0-25 are ignored, bits 26-31

specify the number of bit positions to be shifted.



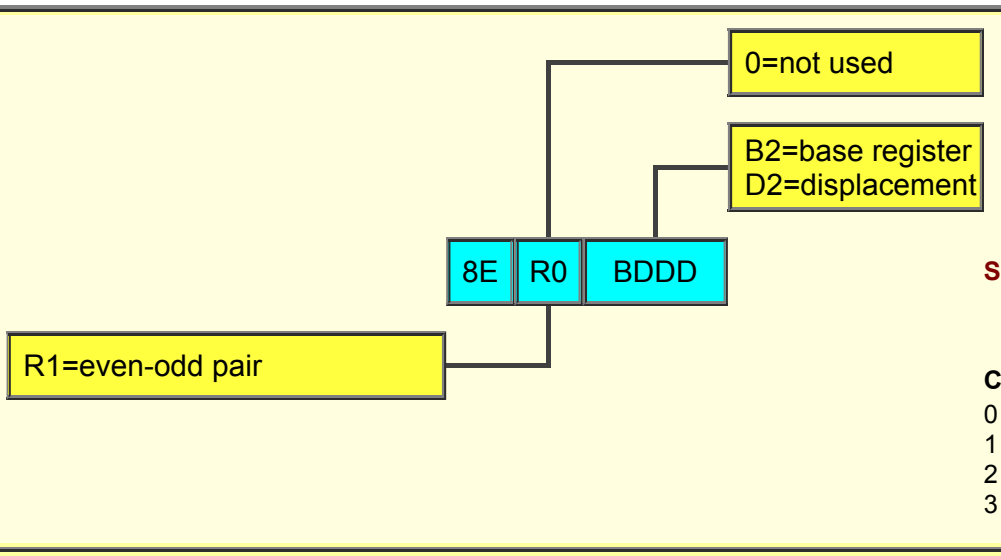
SRA, Shift Right Single Instruction, the operand format is R1,D2(X2,B2)

SRDA, Shift Right Double

The 63-bit numeric part (bit-0 is the sign and bits 1-63 are the numerics) of operand-1 (r1 is an EVEN/ODD pair of registers) is shifted right the number of bits specified by operand-2 (b2+d2).

With this instruction operand-2 (b2+d2) does not address storage. Bits 0-25 are ignored, bits 26-31 specify the number of bit positions to be shifted.

Note: *If an ODD/EVEN pair of registers is specified then an 0C6 error will occur*



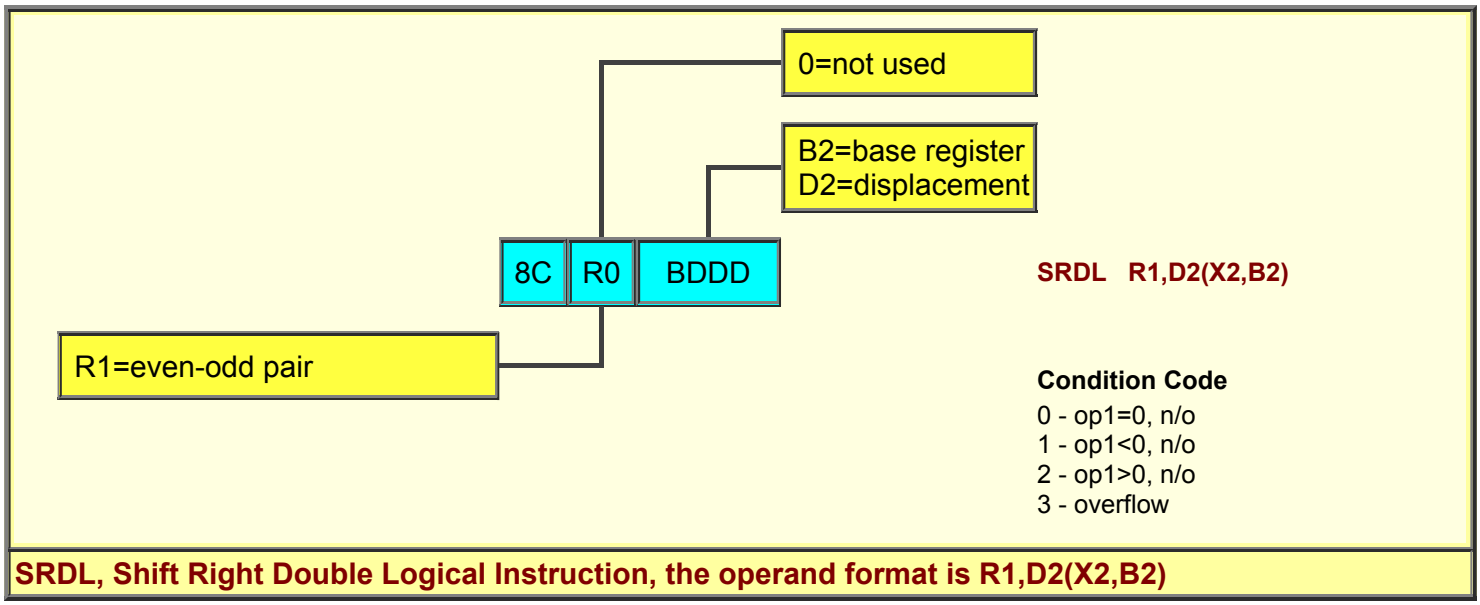
SRDA, Shift Right Double Instruction, the operand format is R1,D2(X2,B2)

SRDL, Shift Right Double Logical

The 64-bits of operand-1 (r1 is an EVEN/ODD pair of registers) is shifted right the number of bits specified by operand-2 (b2+d2).

With this instruction operand-2 (b2+d2) does not address storage. Bits 0-25 are ignored, bits 26-31 specify the number of bit positions to be shifted.

Note: *If an ODD/EVEN pair of registers is specified then an 0C6 error will occur*

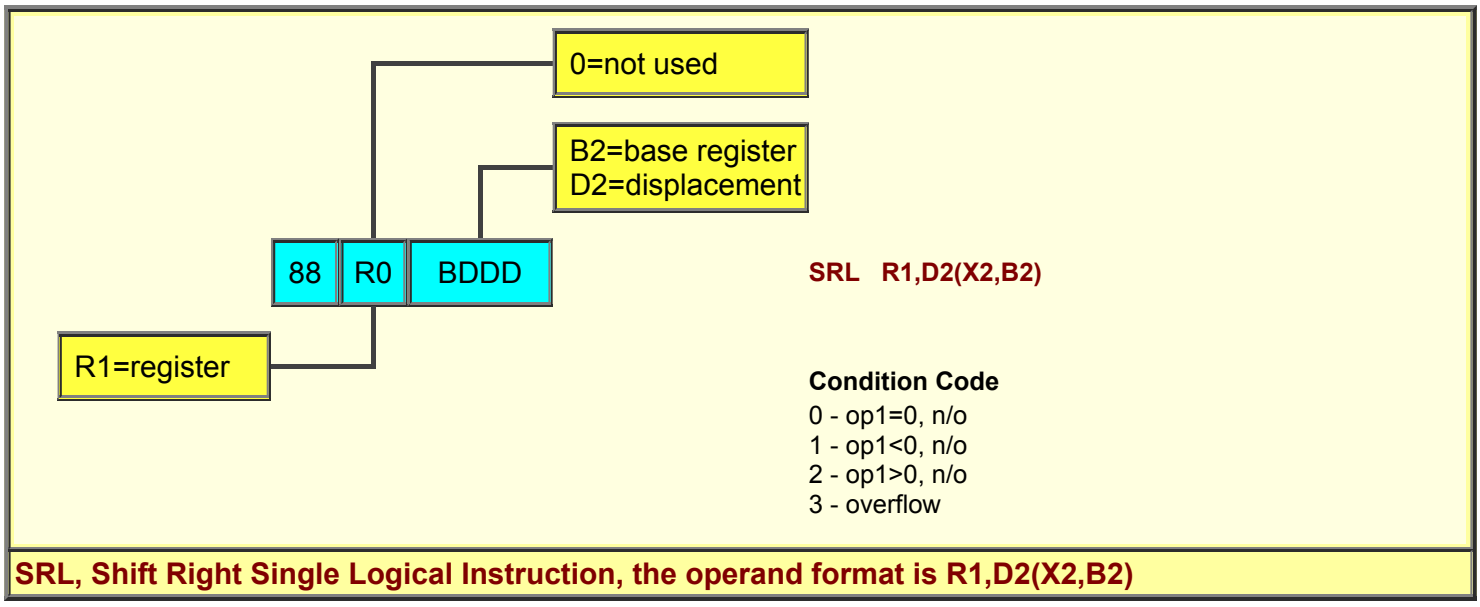


SRDL, Shift Right Double Logical Instruction, the operand format is R1,D2(X2,B2)

SRL, Shift Right Single Logical

The 32-bits of operand-1 (r1) is shifted right the number of bits specified by operand-2 (b2+d2).

With this instruction operand-2 (b2+d2) does not address storage. Bits 0-25 are ignored, bits 26-31 specify the number of bit positions to be shifted.

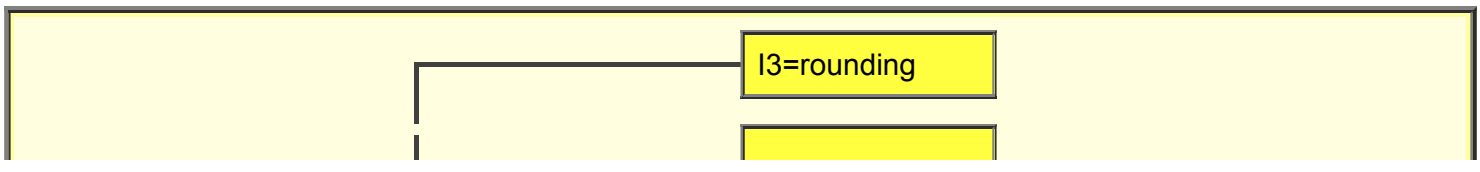


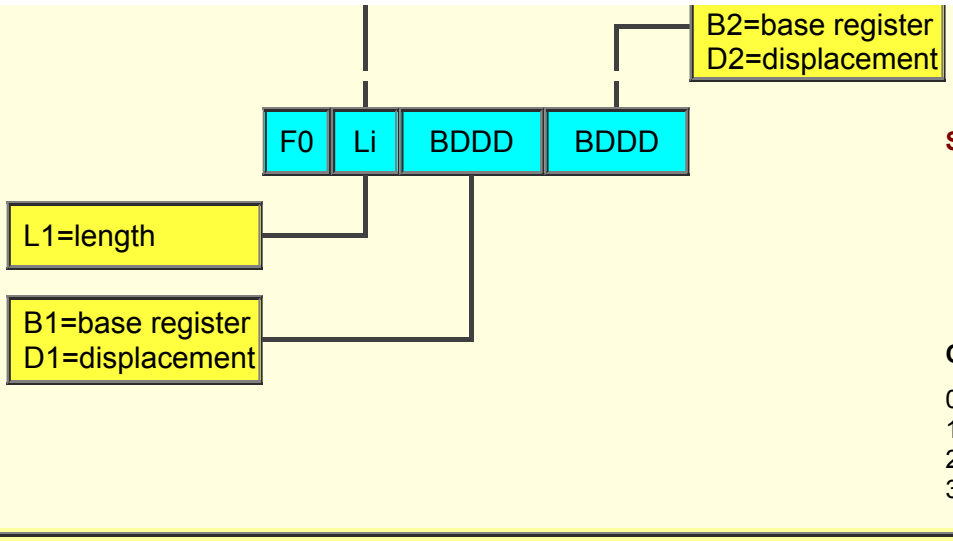
SRL, Shift Right Single Logical Instruction, the operand format is R1,D2(X2,B2)

SRP, Shift and Round Decimal

The data string located at the storage address specified by operand-1 (b1+d1) is shifted and rounded under control of operand-2 and i3. The i3 value is the rounding digit to be used.

With this instruction operand-2 (b2+d2) is not used to address storage. It is used to determine the shift value.

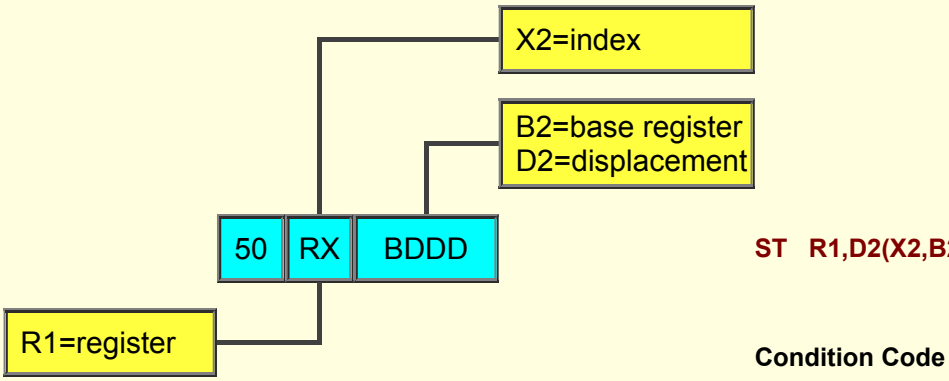




SRP, Subtract and Round Decimal Instruction, the operand format is D1(L1,B1),D2(L2,B2)

ST, Store

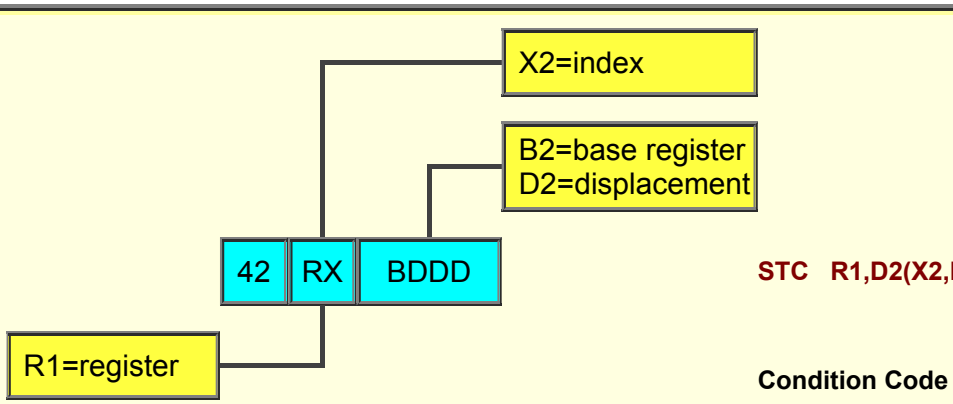
The register specified by operand-1 (r1) is stored at the storage address specified by operand-2 (x2+b2+d2).



ST, Store Instruction, the operand format is R1,D2(X2,B2)

STC, Store Character

Bits 24-31 (one-byte) of the register specified by operand-1 (r1) are stored at the storage address specified by operand-2 (x2+b2+d2).

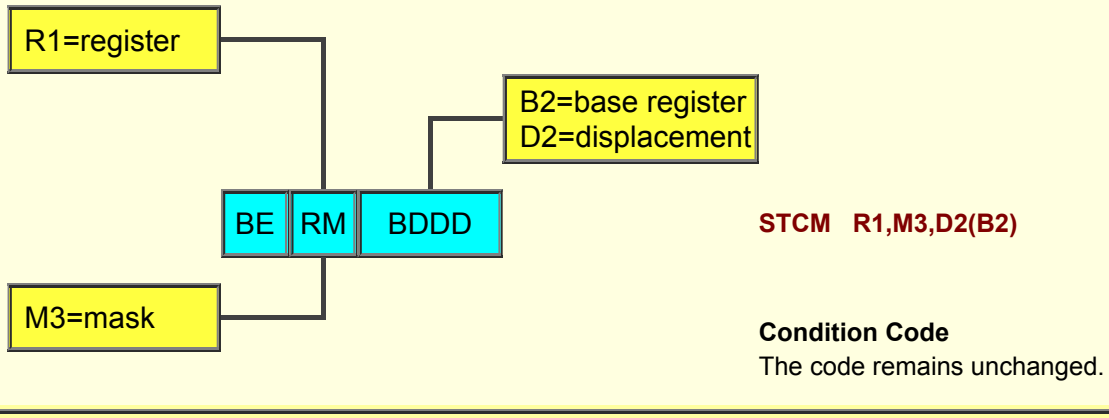


STC, Store Character Instruction, the operand format is R1,D2(X2,B2)

STCM, Store Characters under Mask

Bytes from storage location specified by operand-2 (b2+d2) are inserted into operand-1 (r1) under control of the mask (m3).

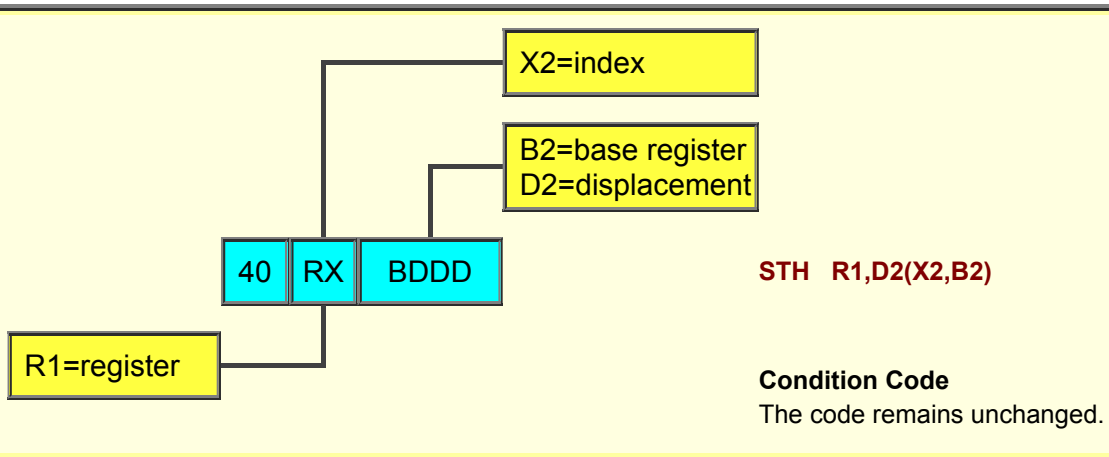
m3=0001, similar to STC
m3=0011, similar to STH
m3=1111, similar to ST
m3=0111, store 24-bit address



STCM, Store Character under Mask Instruction, the operand format is R1,M3,D2(B2)

STH, Store Halfword

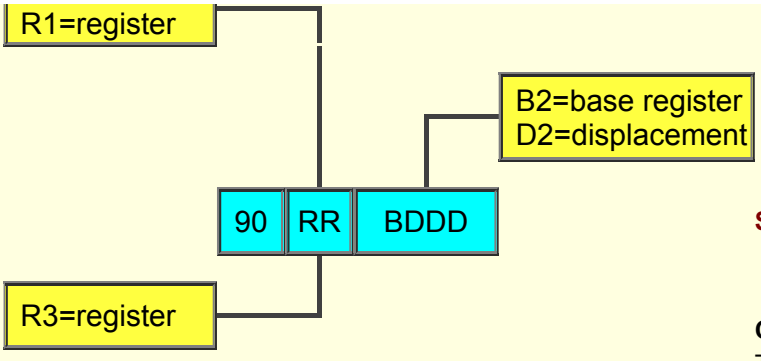
The rightmost two bytes or halfword (bits 16-31) of the register specified by operand-1 (r1) are stored at the storage address specified by operand-2 (x2+b2+d2).



STH, Store Halfword Instruction, the operand format is R1,D2(X2,B2)

STM, Store Multiples

The set of registers starting with operand-1 (r1) and ending with (r3) are stored at the storage location specified by operand-2 (b2+d2).



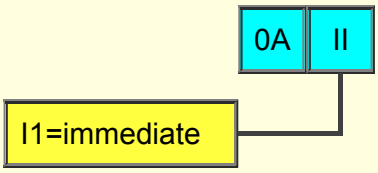
STM R1,D2(X2,B2)

Condition Code
The code remains unchanged.

STM, Store Multiples Instruction, the operand format is R1,D2(X2,B2)

SVC, Supervisor Call

This instruction causes a supervisor-call interrupt. The "ii" field *(or immediate data that is included as the 2nd byte of the instruction)* provides the interruption code (SVC number).



SVC I1

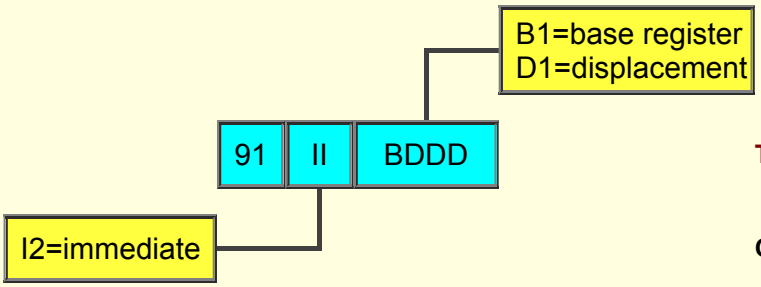
Condition Code
The code remains unchanged

Note: The 2nd byte of this instruction is used as immediate data that is the interuption code or SVC number.

SVC, Supervisor Call Instruction, the operand format is I

TM, Test under Mask

Operand-2 *(the immediate data that is the second byte of the instruction itself)* is used as a mask to test bits of the byte at the storage address specifed by operand-1 (b1+d1). The results are posted in the condition code.



TM D1(B1),I2

Condition Code

0 -Selected bits all-zero; mask is all-zero
1 - Selected bits mixed zero and one
2 - --
3 - Selected bits all-one

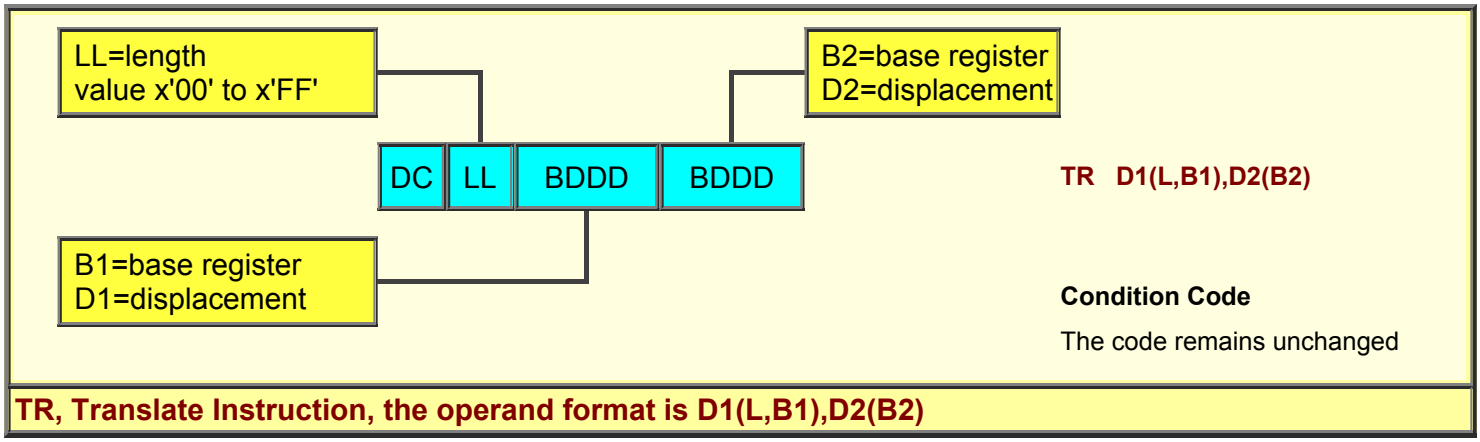
TM, Test under Mask Instruction, the operand format is D1(B1),I2

TR, Translate

The bytes at the storage address specified by operand-1 (b1+d1) are replaced by the byte located at

the calculated address within operand-2 (b2+d2).

The address is calculated from x2+d2 and the byte value from operand-1

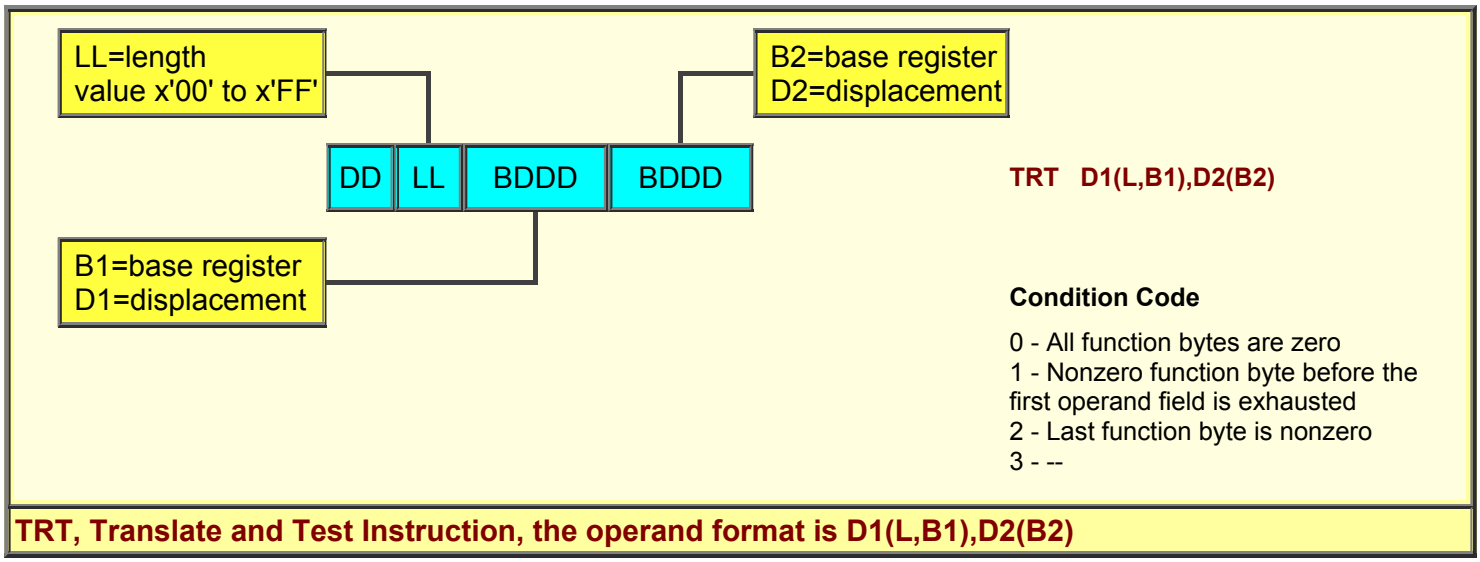


TR, Translate Instruction, the operand format is D1(L,B1),D2(B2)

TRT, Translate and Test

The bytes at the storage address specified by operand-1 (b1+d1) are used as arguments to select function-bytes from a list designated by operand-2 (b2+d2).

The first non-zero byte is put into register-2 with the address in register-1.

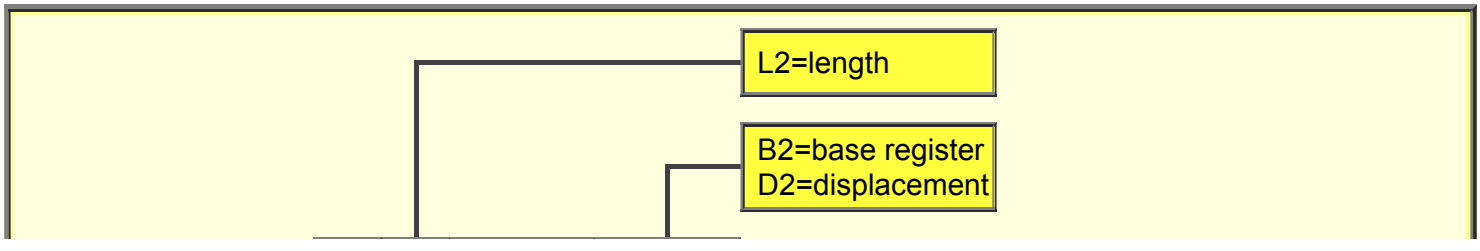


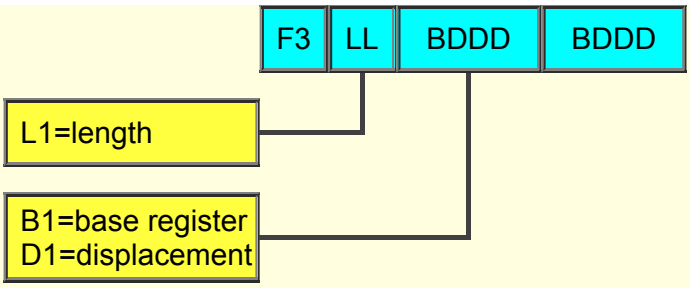
TRT, Translate and Test Instruction, the operand format is D1(L,B1),D2(B2)

UNPK, Unpack

The data string located at the storage address specified by operand-2 (b2+d2) is changed from packed to zoned-decimal and the result is put into the storage address specified by operand-1 (b1+d1).

Operand-2 remains unchanged. The operands may be different lengths with a maximum length of 16 bytes (or 31 digits since this is packed) for each operand.





UNPK D1(L1,B1),D2(L2,B2)

Condition Code

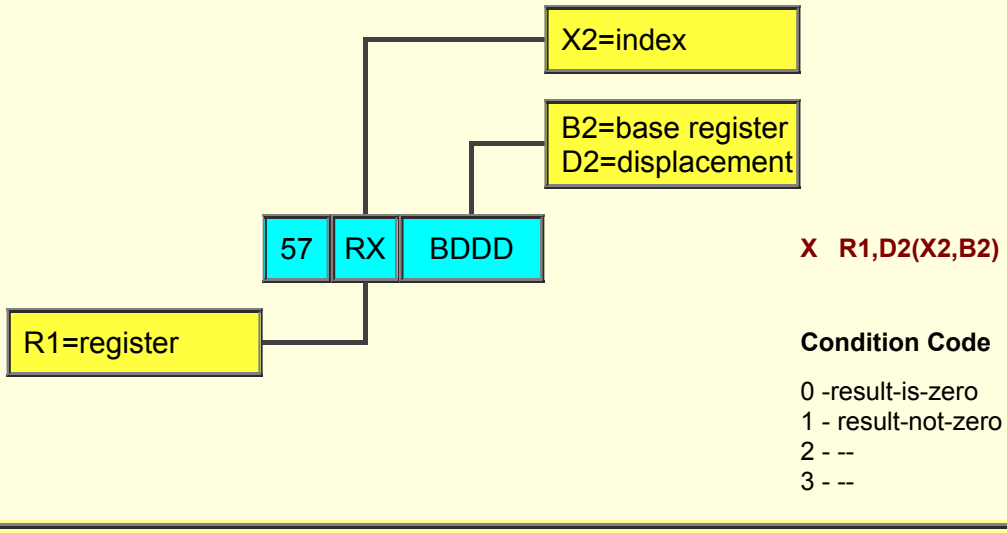
The code remains unchanged.

UNPK, Unpack (Decimal) Instruction, the operand format is D1(L1,B1),D2(L2,B2)

X, Exclusive Or

The content of operand-1 (r1) is exclusively OR'ed with the data string located at the storage address specified by operand-2 (x2+b2+d2).

The results of the exclusive OR'ing process is put into operand-1.



X R1,D2(X2,B2)

Condition Code

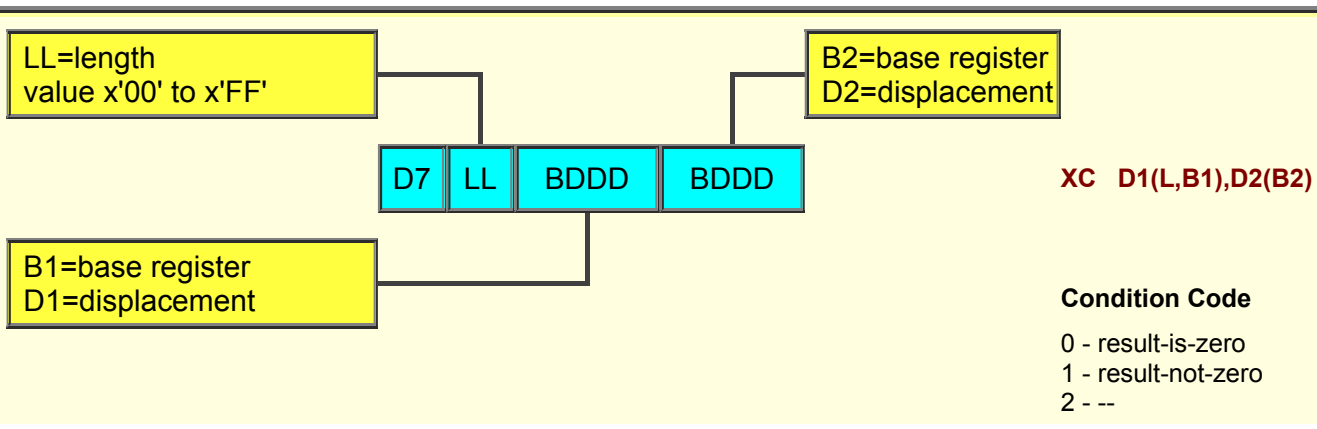
0 -result-is-zero
1 - result-not-zero
2 - --
3 - --

X, Exclusive Or Instruction, the operand format is R1,D2(X2,B2)

XC, Exclusive Or Characters

The data string located at the storage address specified by operand-1 (b1+d1) is exclusively OR'ed with the data string located at the storage address specified by operand-2 (b2+d2).

The results of the exclusive OR'ing process is put into operand-1.



XC D1(L,B1),D2(B2)

Condition Code

0 - result-is-zero
1 - result-not-zero
2 - --

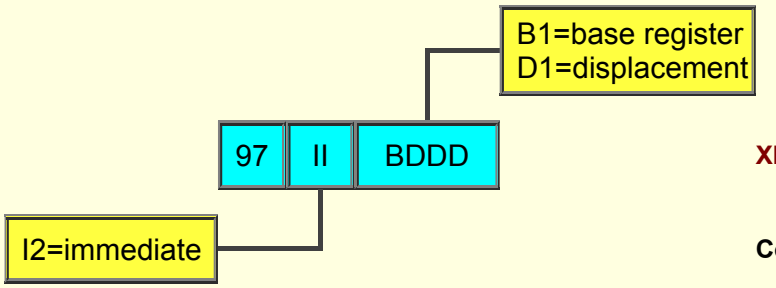
3 - --

XC, Excluse Or Characters Instruction, the operand format is D1(L,B1),D2(B2)

XI, Exclusive Or Immediate

The data string located at the storage address specified by operand-1 (b1+d1) is exclusively OR'ed with operand-2 *(i2 is immediate data included as the second byte of the instruction itself)*.

The results of the exclusive OR'ing process is put into operand-1.



XI D1(B1),I2

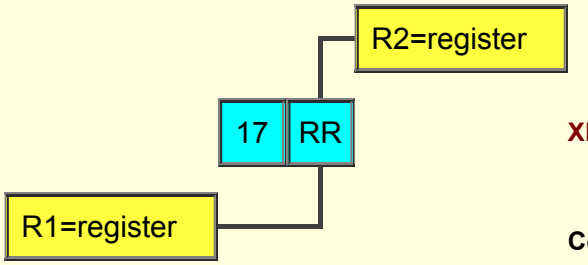
Condition Code

0 - result-is-zero
1 - result-not-zero
2 - --
3 - --

XI, Exclusive Or Immediate Instruction, the operand format is D1(B1),I2

XR, Exclusive Or Registers

The content of operand-1 (r1) is exclusively OR'ed with the content of operand-2 (r2). The results of the exclusive OR'ing process is put into operand-1.



XR R1,R2

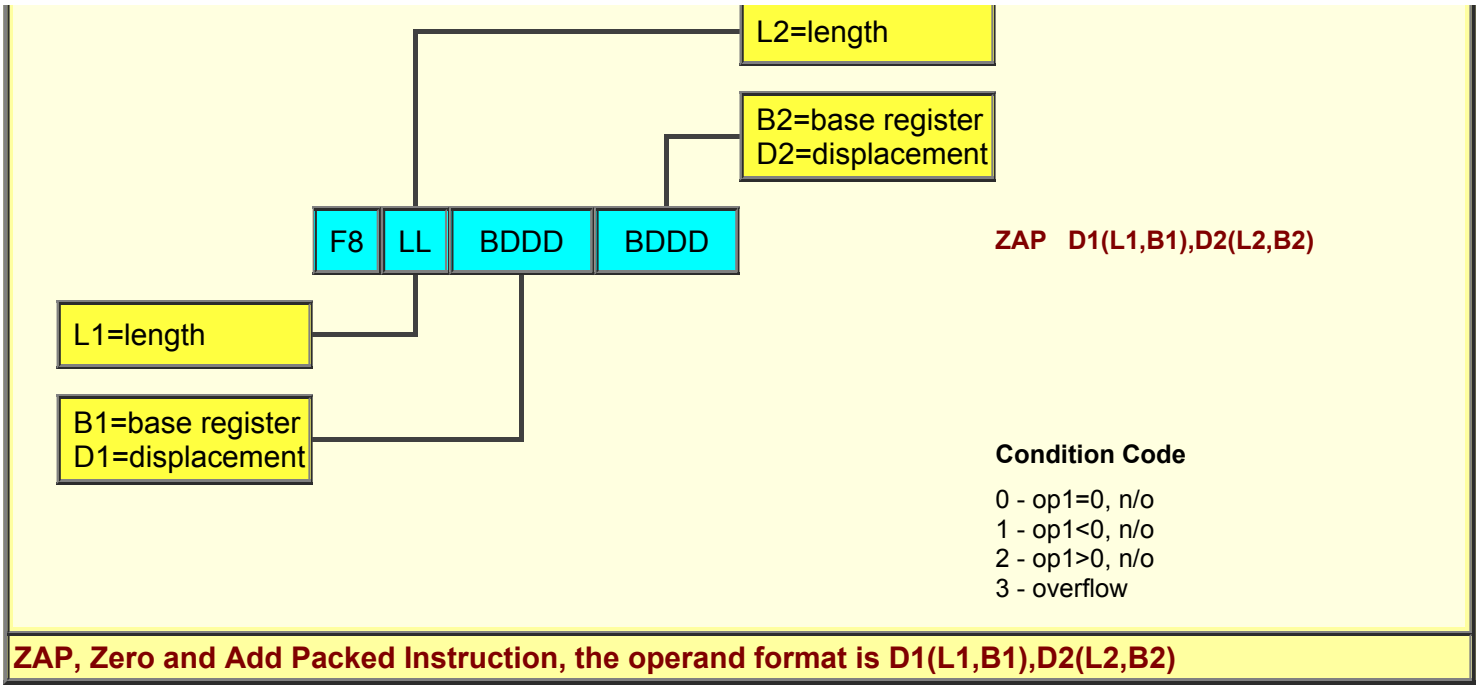
Condition Code

0 - result-is-zero
1 - result-not-zero
2 - --
3 - --

XR, Exclusive Or Registers Instruction, the operand format is R1,R2

ZAP, Zero and Add Packed

Operand-1 (b1+d1) is initialized to zero and the data string located at the storage address specified by operand-2 (b2+d2) is added to the data string located at the storage address specified by operand-1. Operand-2 remains unchanged. The operands may be different lengths with a maximum length of 16 bytes *(or 31 digits since this is packed)* for each operand. The condition code is set as shown below.



ZAP, Zero and Add Packed Instruction, the operand format is D1(L1,B1),D2(L2,B2)

Summary

The purpose of this document is to assist as a tutorial for new programmers or as a quick reference for experienced programmers. In the world of programming there are many ways to solve a problem. This document and the links to other documents are intended to provide a choice of alternatives.

Software Agreement and Disclaimer

Downloads and Links

This section includes links to documents with additional information that are beyond the scope and purpose of this document. The first group of documents may be available from a local system or via an internet connection, the second group of documents will require an internet connection.

Note: *A SimoTime License is required for the items to be made available on a local system or server.*

Current Server or Internet Access

The following links may be to the current server or to the Internet.

Note: *The latest versions of the SimoTime Documents and Program Suites are available on the*

Internet and may be accessed using the I icon. If a user has a SimoTime Enterprise License the Documents and Program Suites may be available on a local server and accessed using the S icon.

I S Explore An Enterprise System Model that describes and demonstrates how Applications that were running on a Mainframe System and non-relational data that was located on the Mainframe System were copied and deployed in a Microsoft Windows environment with Micro Focus Enterprise Server.

I S Explore the Assembler Connection for more examples of mainframe Assembler programming techniques and sample code.

I S Explore the Extended Mnemonic Opcodes included in the IBM Mainframe Assembler Language.

I S Explore an Assembler Program that Executes each of the problem-state, non-floating-point instructions in alphabetical sequence and will run as an MVS batch job on an IBM mainframe or on a Windows System with Micro Focus Technology.

I S Explore The ASCII and EBCDIC Translation Tables. These tables are provided for individuals that need to better understand the bit structures and differences of the encoding formats.

I S Explore The File Status Return Codes to interpret the results of accessing VSAM data sets and/or QSAM files.

Internet Access Required

The following links will require an internet connection.

A good place to start is The SimoTime Home Page via Internet Connect for access to white papers, program examples and product information.

Explore The Micro Focus Web Site via Internet Connect for more information about products and services available from Micro Focus.

Glossary of Terms

I S Explore the Glossary of Terms for a list of terms and definitions used in this suite of documents and white papers.

Comments or Feedback

This document was created and is copyrighted and maintained by SimoTime Technologies.

If you have any questions, suggestions, comments or feedback please call or send an e-mail to: helpdesk@simotime.com

We appreciate hearing from you.

Company Overview

Founded in 1987, SimoTime Technologies is a privately owned company. We specialize in the creation and deployment of business applications using new or existing technologies and services. We have a team of individuals that understand the broad range of technologies being used in today's environments. This includes the smallest thin client using the Internet and the very large mainframe systems. There is more to making the Internet work for your company's business than just having a nice looking WEB site. It is about combining the latest technologies and existing technologies with practical business experience. It's about the business of doing business and looking good in the process. Quite often, to reach larger markets or provide a higher level of service to existing customers it requires the newer Internet technologies to work in a complementary manner with existing corporate mainframe systems.

Whether you want to use the Internet to expand into new market segments or as a delivery vehicle for existing business functions simply give us a call or check the web site at http://www.simotime.com

---

Return-to-Top

370 Instructions, Reference Guide

When technology complements business

http://www.simotime.com