INSTRUCTION SET

Revision 4
Nov-02-2022

CODING

	OPC	MNEMONIC	FLAGS			CODE											
						OPCODE				OPERAND X OPCODE				OPERAND Y			
▶	1	ADD RX,RY	V	Z	C	0	0	0	1	X	X	X	X	Y	Y	Y	Y
▶	2	ADC RX,RY	V	Z	C	0	0	1	0	X	X	X	X	Y	Y	Y	Y
▶	3	SUB RX,RY	V	Z	C	0	0	1	1	X	X	X	X	Y	Y	Y	Y
▶	4	SBB RX,RY	V	Z	C	0	1	0	0	X	X	X	X	Y	Y	Y	Y
▶	5	OR RX,RY		Z		0	1	0	1	X	X	X	X	Y	Y	Y	Y
▶	6	AND RX,RY		Z		0	1	1	0	X	X	X	X	Y	Y	Y	Y
▶	7	XOR RX,RY		Z		0	1	1	1	X	X	X	X	Y	Y	Y	Y
▶	8	MOV RX,RY				1	0	0	0	X	X	X	X	Y	Y	Y	Y
	9	MOV RX,#N				1	0	0	1	X	X	X	X	N	N	N	N
	A	MOV [XY],R0				1	0	1	0	X	X	X	X	Y	Y	Y	Y
	B	MOV R0,[XY]				1	0	1	1	X	X	X	X	Y	Y	Y	Y
	C	MOV [NN],R0				1	1	0	0	N	N	N	N	N	N	N	N
	D	MOV R0,[NN]				1	1	0	1	N	N	N	N	N	N	N	N
	E	MOV PC,NN				1	1	1	0	N	N	N	N	N	N	N	N
	F	JR NN				1	1	1	1	N	N	N	N	N	N	N	N
	00	CP R0,N	V	Z	C	0	0	0	0	0	0	0	0	N	N	N	N
	01	ADD R0,N	V	Z	C	0	0	0	0	0	0	0	1	N	N	N	N
▶	02	INC RY		Z	C	0	0	0	0	0	0	1	0	Y	Y	Y	Y
▶	03	DEC RY		Z	C	0	0	0	0	0	0	1	1	Y	Y	Y	Y
	04	DSZ RY				0	0	0	0	0	1	0	0	Y	Y	Y	Y
	05	OR R0,N		Z	C	0	0	0	0	0	1	0	1	N	N	N	N
	06	AND R0,N		Z	C	0	0	0	0	0	1	1	0	N	N	N	N
	07	XOR R0,N		Z	C	0	0	0	0	0	1	1	1	N	N	N	N
	08	EXR N				0	0	0	0	1	0	0	0	N	N	N	N
	09	BIT RG,M		Z	C	0	0	0	0	1	0	0	1	G	G	M	M
	0A	BSET RG,M				0	0	0	0	1	0	1	0	G	G	M	M
	0B	BCLR RG,M				0	0	0	0	1	0	1	1	G	G	M	M
	0C	BTG RG,M				0	0	0	0	1	1	0	0	G	G	M	M
▶	0D	RRC RY		Z	C	0	0	0	0	1	1	0	1	Y	Y	Y	Y
	0E	RET R0,N				0	0	0	0	1	1	1	0	N	N	N	N
	0F	SKIP F,M				0	0	0	0	1	1	1	1	F	F	M	M

Note: Modes SS and RUN support all instructions, and ALU mode supports only instructions with triangular sign "▶", but not in the same way as SS and RUN modes. Please refer to the section “INSTRUCTIONS IN ALU MODE”.

ADD RX,RY Add registers RX and RY

Syntax: {label} ADD RX, RY

Operands: RX ∈ [R0...R15]
RY ∈ [R0...R15]

Operation: (RX) ← (RX) + (RY)

Description: Add the contents of the register RY to the contents of the register RX and place the result in the register RX. Register direct addressing must be used for RX and RY.

Flags affected: If there is the overflow (if (RX)+(RY)>15), set C. Otherwise, reset C. If result=0000 after operation, set Z. Otherwise, reset Z. For signed 2's complement arithmetic: If there is overflow, set V. Otherwise, reset V.

Encoding:

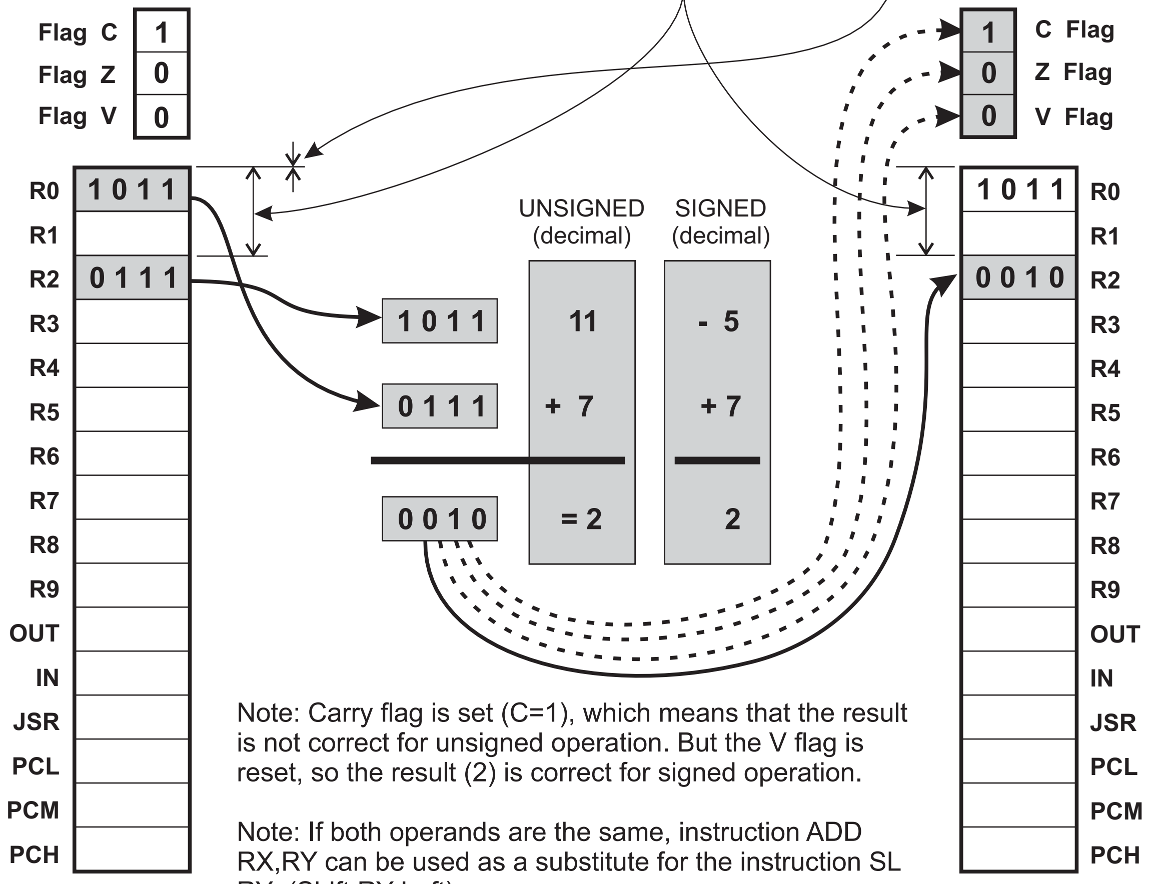
bit 11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	X	X	X	X	Y	Y	Y	Y

The "0001" bits are the ADD RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example:
ADD R2, R0

bit 11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	0	0	0	0

OPCODE 0001 = ADD OPERAND RX OPERAND RY



Note: Carry flag is set (C=1), which means that the result is not correct for unsigned operation. But the V flag is reset, so the result (2) is correct for signed operation.

Note: If both operands are the same, instruction ADD RX,RY can be used as a substitute for the instruction SL RY (Shift RY Left)

Before operation After operation

ADC RX,RY Add with carry registers RX and RY

Syntax:	{label} ADC RX, RY
Operands:	RX $\in$ [R0...R15] RY $\in$ [R0...R15]
Operation:	(RX) $\leftarrow$ (RX) + (RY) + Carry
Description:	Add the contents of the register RY plus the contents of Carry flag to the contents of the register RX and place the result in the register RX. Register direct addressing must be used for RX and RY.
Flags affected:	If there is the underflow (if (RY)<(RX)), reset C. Otherwise, set C. (note: Borrow is inverse C). If result=0000 after operation, set Z. Otherwise, reset Z. For signed 2's complement: If there is overflow, set V. Otherwise, reset V.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	0	X	X	X	X	Y	Y	Y	Y

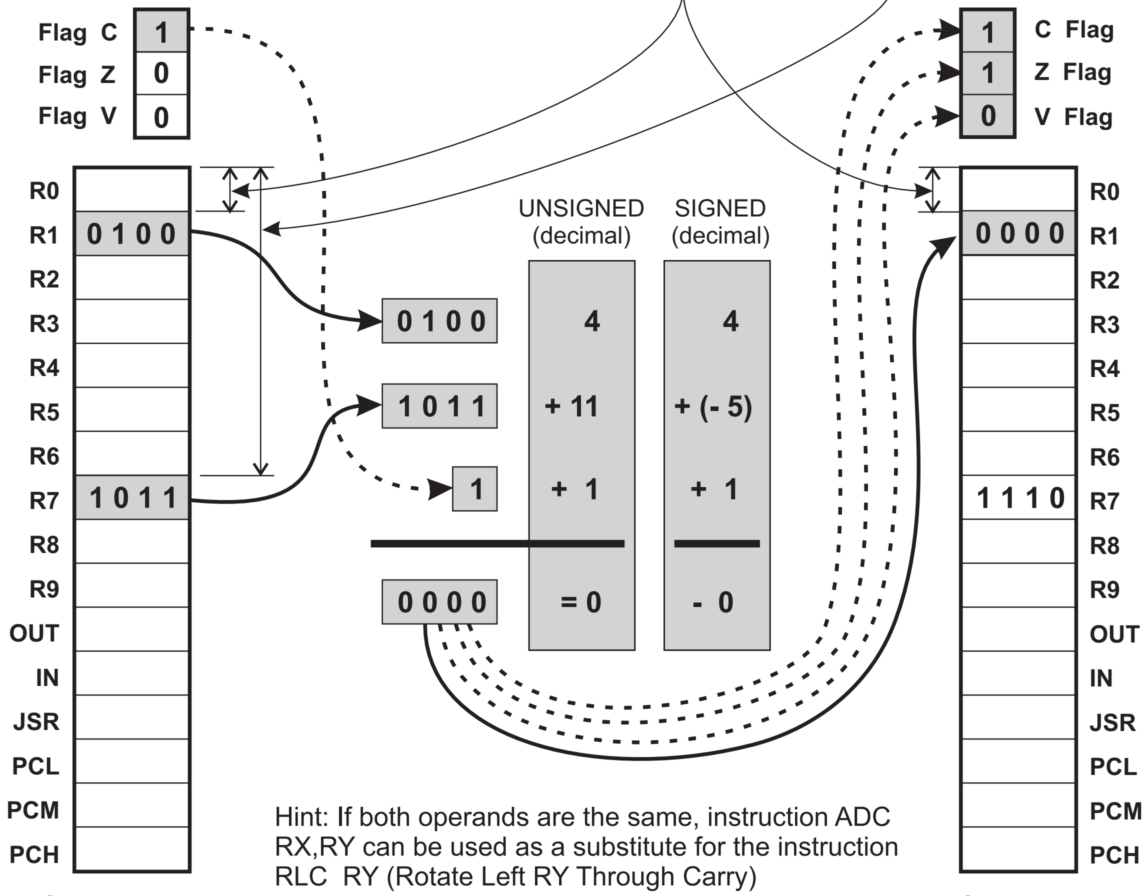
The "0010" bits are the ADC RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example:

ADC R1, R7

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	0	0	0	0	1	0	1	1	1

OPCODE 0010 = ADC — OPERAND RX — OPERAND RY



Hint: If both operands are the same, instruction ADC RX,RY can be used as a substitute for the instruction RLC RY (Rotate Left RY Through Carry)

Before operation — After operation

SUB RX,RY Subtract register RY from register RX

Syntax: {label} SUB RX, RY

Operands:
RX ∈ [R0...R15]
RY ∈ [R0...R15]

Operation: $(RX) \leftarrow (RX) - (RY)$

Description: Subtract the contents of the register RY from the contents of the register RX and place the result in the register RX. Register direct addressing must be used for RX and RY.

Flags affected: If there is the underflow (if (RY)<(RX)), reset C. Otherwise, set C. (note: Borrow is inverse C). If result=0000 after operation, set Z. Otherwise, reset Z. For signed 2's complement: If there is overflow, set V. Otherwise, reset V.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	X	X	X	X	Y	Y	Y	Y

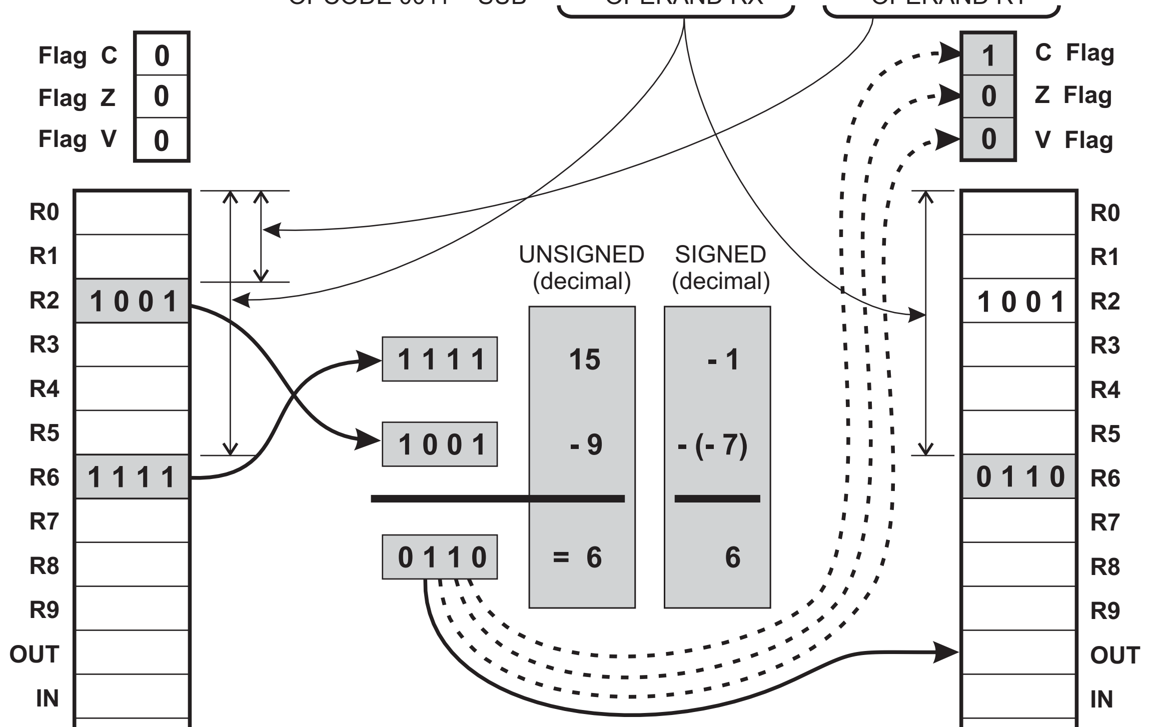
The "0011" bits are the SUB RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example 1:

SUB R6, R2

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	0	1	1	0	0	0	1	0

OPCODE 0011 = SUB; OPERAND RX; OPERAND RY



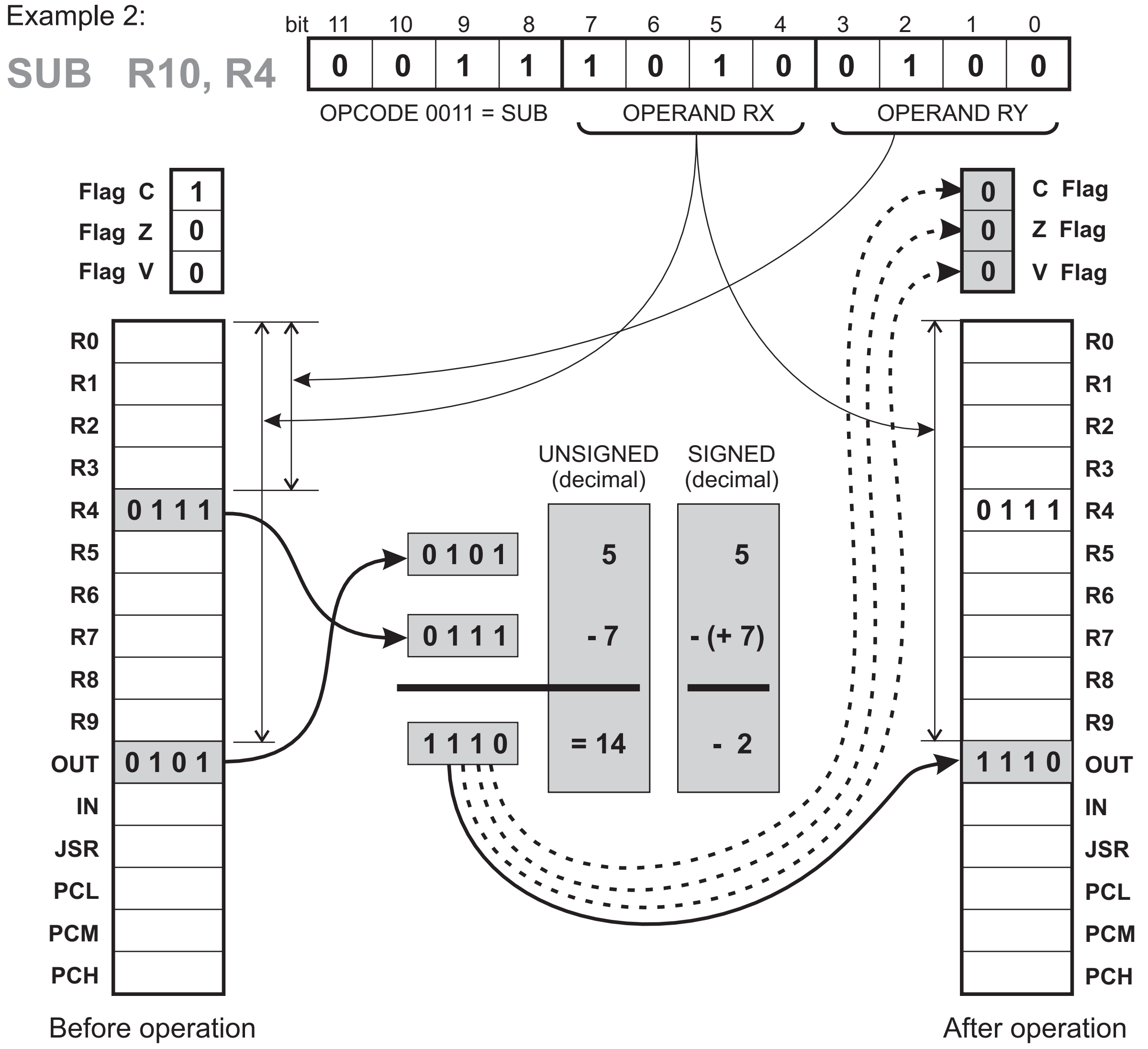
Before operation

Note: For subtraction, Carry flag is called Borrow, and it is actually inverse Carry. So, No-Carry (if C=0) means "Borrow", and Carry (C=1) means "No Borrow". In this example the resulting C=1, so the result is correct in unsigned representation. Flag V is reset, which means that the result is correct in signed representation also.

After operation

SUB RX,RY Subtract register RY from register RX (CONTINUED)

Example 2:



Note: For subtraction, Carry flag is called Borrow, and it is actually inverse Carry. So, No-Carry (if C=0) means "Borrow", and Carry (C=1) means "No Borrow". In this example C=0, so there is Underflow condition, which means that the result is not correct in unsigned representation. In signed representation, the result is 1110, which is -2. Flag V is not set, which means that -2 is the correct result.

SBB RX,RY Subtract register RY from register RX with borrow

Syntax:	{label} SBB RX, RY
Operands:	RX ∈ [R0...R15] RY ∈ [R0...R15]
Operation:	(RX) ← (RX) - (RY) - ($\bar{C}$)
Description:	Subtract the contents of the register Y from the contents of the register X and place the result in the register X. Register direct addressing must be used for X and Y.
Flags affected:	If there is the underflow (if (RY)<(RX)), reset C. Otherwise, set C. (note: Borrow is inverse C). If result=0000 after operation, set Z. Otherwise, reset Z. For signed 2's complement: If there is overflow, set V. Otherwise, reset V.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	X	X	X	X	Y	Y	Y	Y

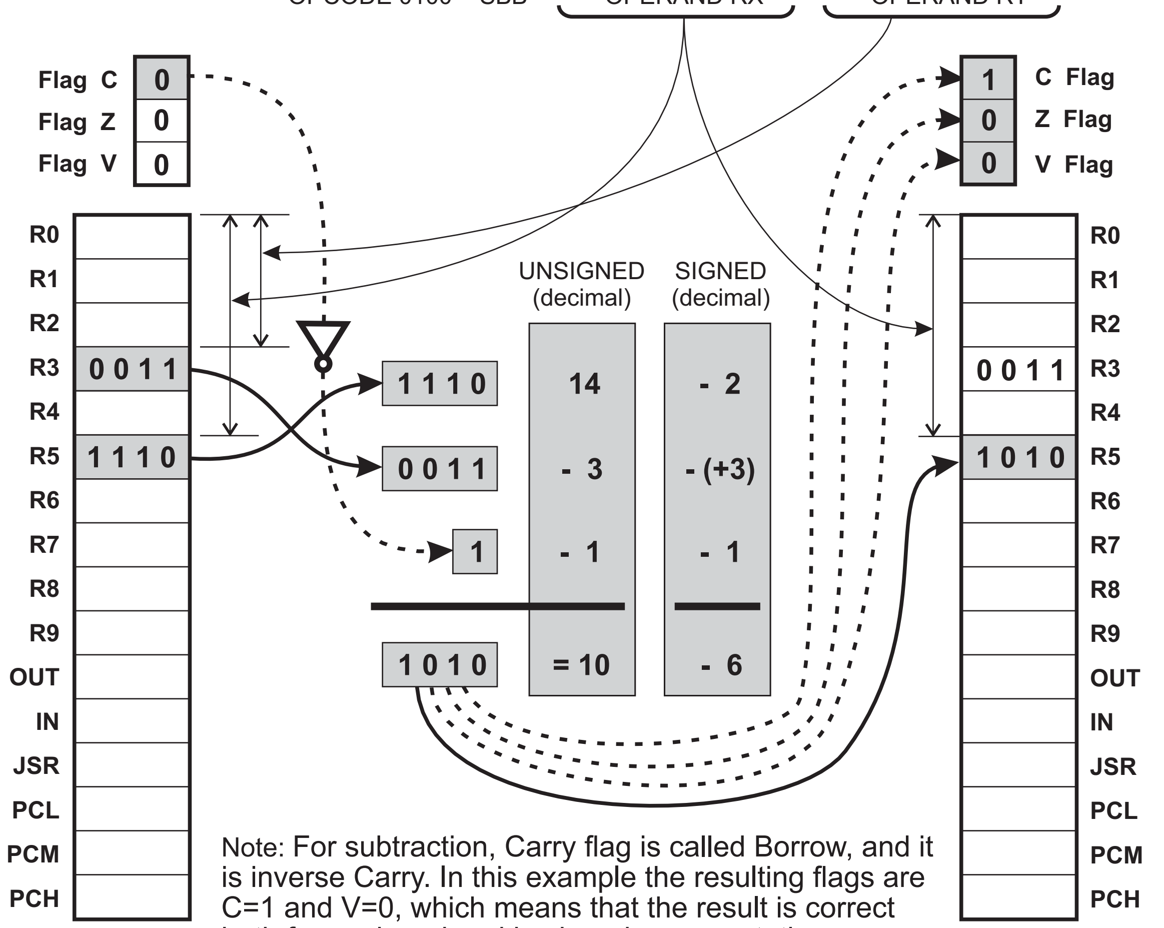
The "0100" bits are the SBB RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example 1:

SBB R5, R3

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	0	1	0	1	0	0	1	1

OPCODE 0100 = SBB OPERAND RX OPERAND RY



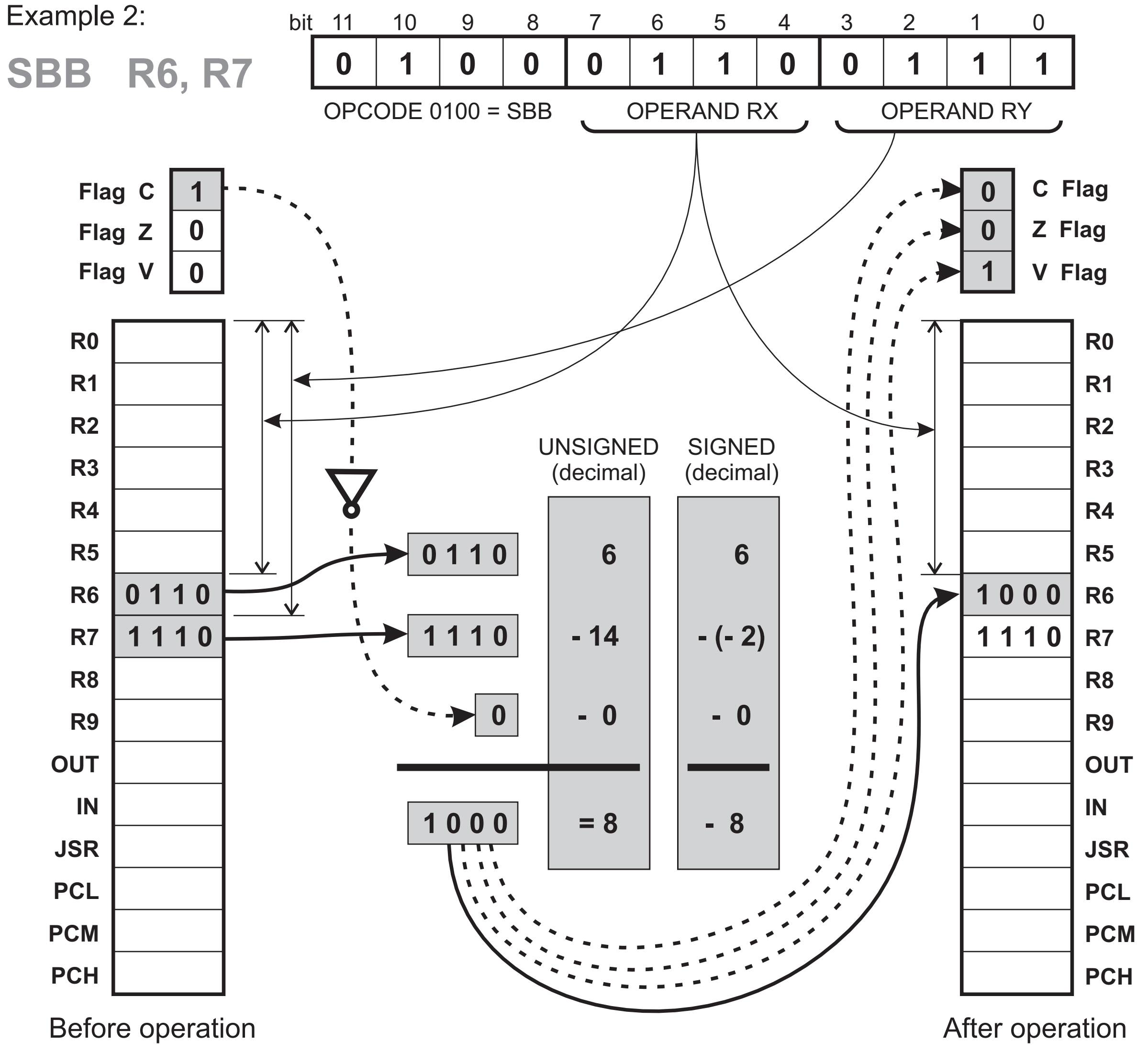
Note: For subtraction, Carry flag is called Borrow, and it is inverse Carry. In this example the resulting flags are C=1 and V=0, which means that the result is correct both for unsigned and in signed representations.

Before operation After operation

SBB RX,RY Subtract register RY from register RX with borrow (CONTINUED)

Example 2:

SBB R6, R7



Note: For subtraction, Carry flag is called Borrow, and it is actually inverse Carry. So, No-Carry (C=0) means "Borrow", and Carry (C=1) means "No Borrow". In this example the resulting flag C=0, so there is Underflow condition, which means that the result is not correct in unsigned representation. The Overflow flag is set (V=1), which means that the result is not correct even in signed representation.

OR RX,RY Inclusive OR registers RX and RY

Syntax:	{label} OR RX, RY
Operands:	RX ∈ [R0...R15] RY ∈ [R0...R15]
Operation:	(RX) ← (RX) .OR. (RY)
Description:	Compute the logical inclusive OR operation of the 4-bit register RX with register RY and place the result back into the register RX. Register direct addressing must be used for RX and RY.
Flags affected:	Flag C is not affected If result=0000 after operation, set Z. Otherwise, reset Z

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	1	X	X	X	X	Y	Y	Y	Y

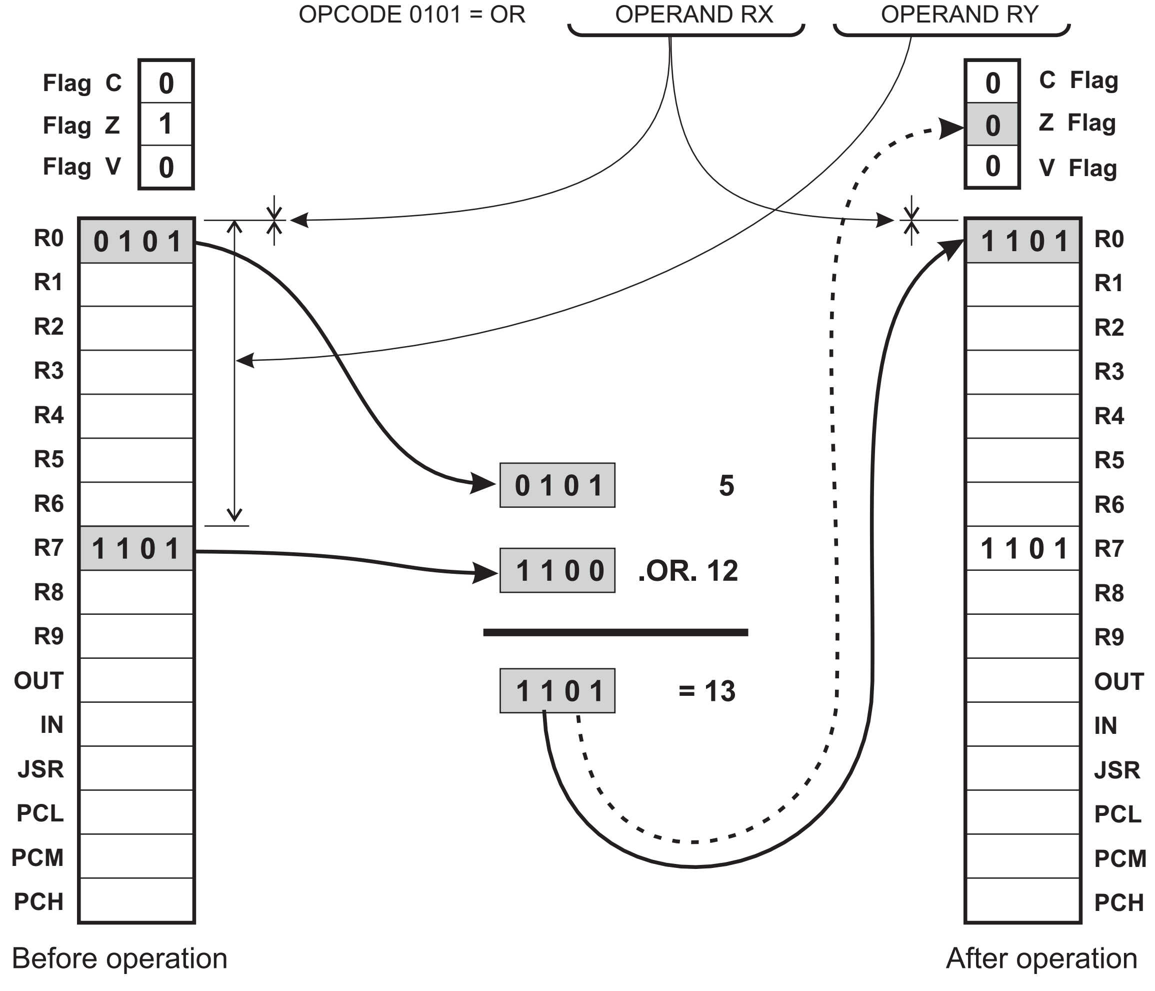
The "0101" bits are the OR RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example:

OR R0, R7

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	1	0	0	0	0	0	1	1	1

OPCODE 0101 = OR OPERAND RX OPERAND RY



AND RX,RY Logical AND registers RX and RY

Syntax: {label} AND RX, RY

Operands: RX ∈ [R0...R15]
RY ∈ [R0...R15]

Operation: (RX) ← (RX) .AND. (RY)

Description: Compute the logical AND operation of the 4-bit register RX with register RY and place result back into the register RX. Register direct addressing must be used for RX and RY.

Flags affected: Flag C is not affected
If result=0000 after operation, set Z. Otherwise, reset Z

Encoding:

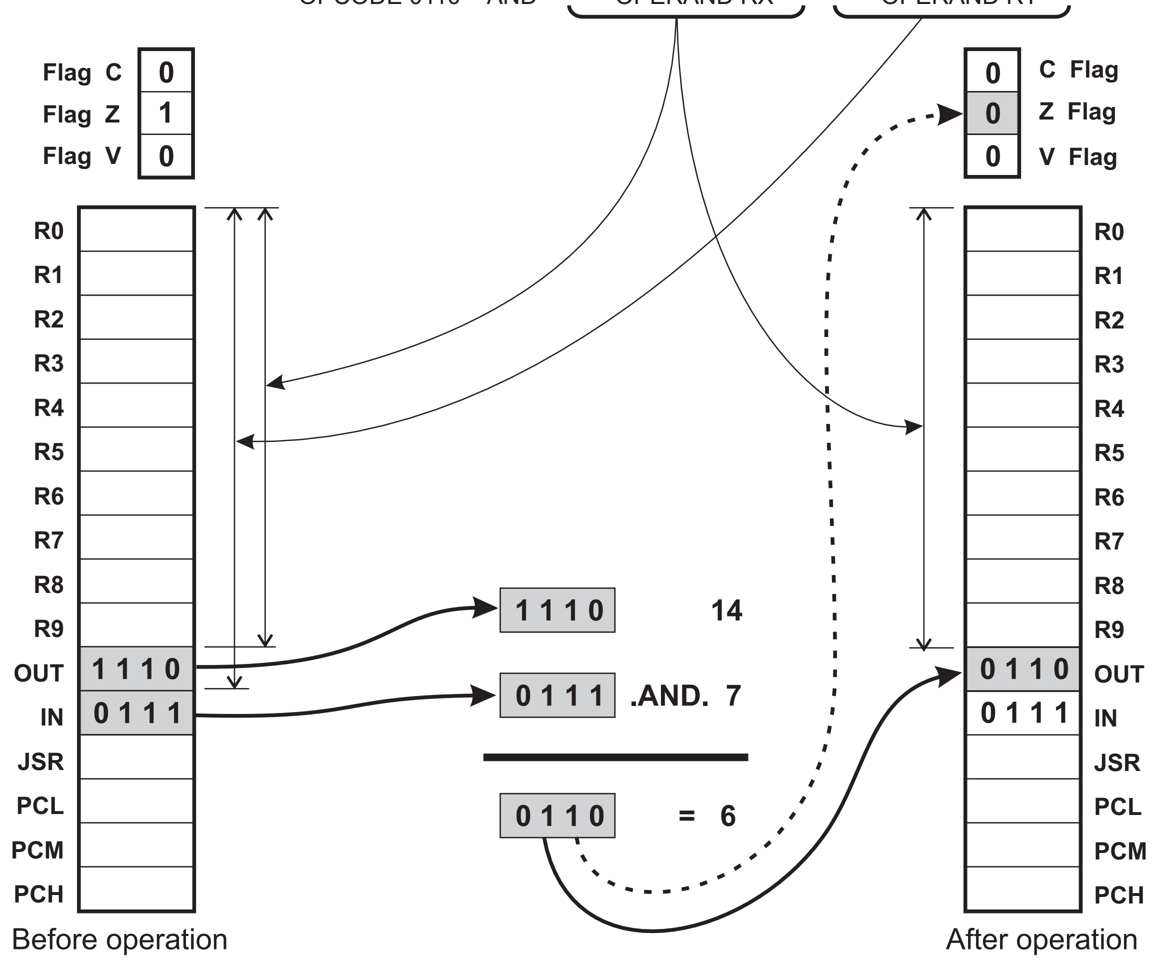
bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	0	X	X	X	X	Y	Y	Y	Y

The "0110" bits are the AND RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example:

AND R10, R11

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	0	1	0	1	0	1	0	1	1
	OPCODE 0110 = AND				OPERAND RX				OPERAND RY			



XOR RX,RY Exclusive OR registers RX and RY

Syntax:	{label} XOR RX, RY
Operands:	RX ∈ [R0...R15] RY ∈ [R0...R15]
Operation:	(RX) ← (RX) .XOR. (RY)
Description:	Compute the logical exclusive XOR operation of the 4-bit register RX with register RY and place the result back into the register RX. Register direct addressing must be used for RX and RY.
Flags affected:	Flag C is not affected If result=0000 after operation, set Z. Otherwise, reset Z

Encoding:

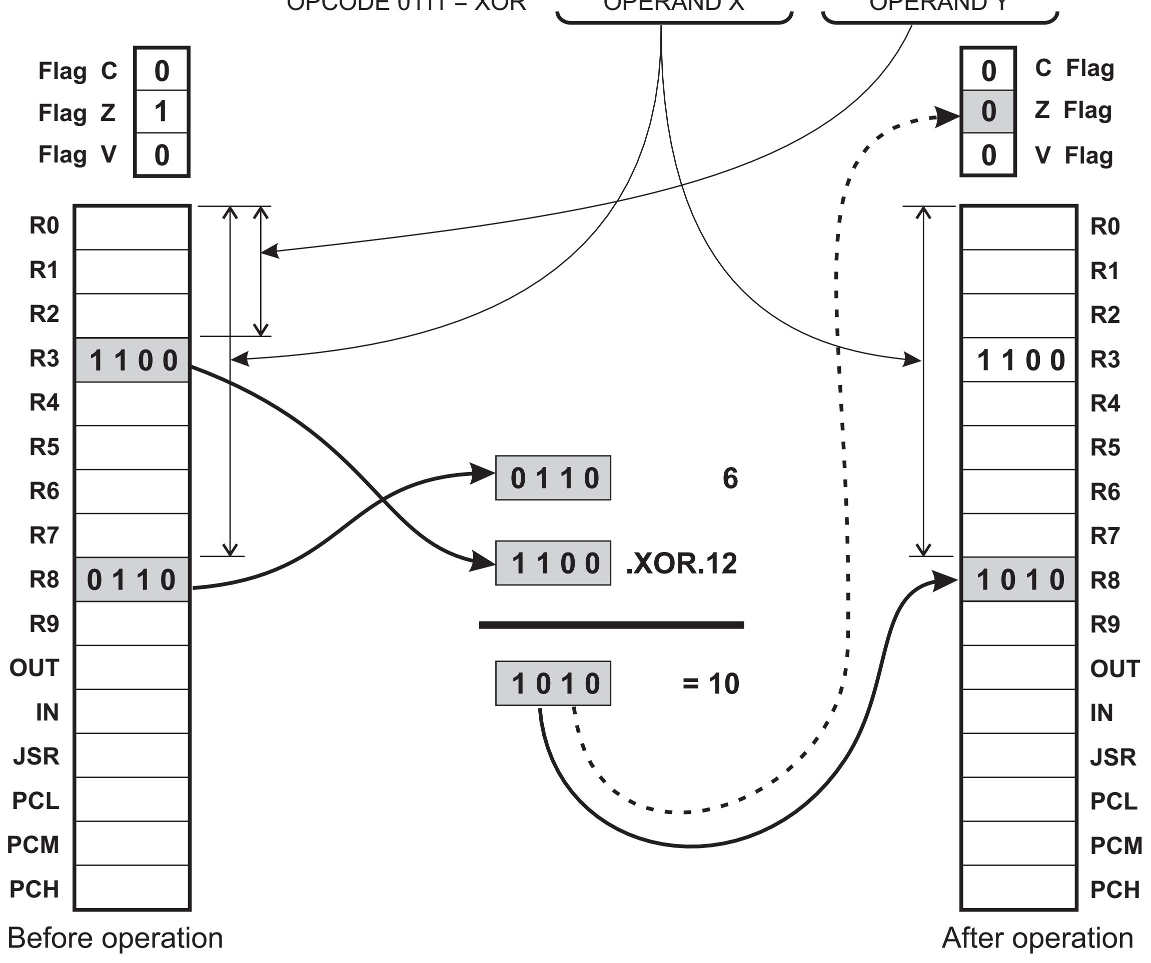
bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	1	X	X	X	X	Y	Y	Y	Y

The "0111" bits are the XOR RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example:

XOR R8, R3

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	1	1	0	0	0	0	0	1	1
	OPCODE 0111 = XOR				OPERAND X				OPERAND Y			



Before operation

After operation

MOV RX,RY Move contents from register RY to register RX

Syntax:	{label} MOV RX, RY
Operands:	RX ∈ [R0...R15] RY ∈ [R0...R15]
Operation:	(RX) ← (RY)
Description:	Move the 4-bit contents from the register RY to the register RX. Register direct addressing must be used for RX and RY.
Flags affected:	None.

Encoding:

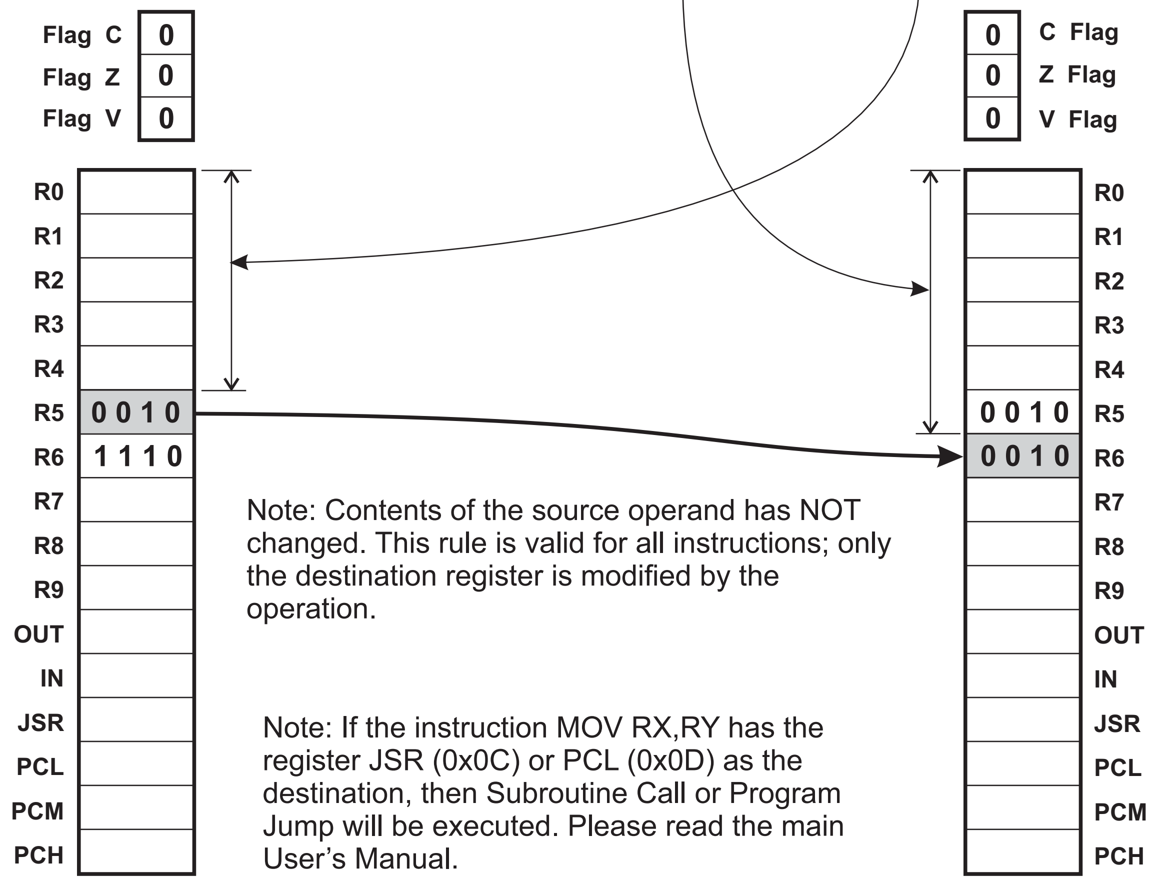
bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	X	X	X	X	Y	Y	Y	Y

The "1000" bits are the MOV RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example:

MOV R6, R5

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	0	1	1	0	0	1	0	1
	OPCODE 1000 = MOV				OPERAND RX				OPERAND RY			



Note: Contents of the source operand has NOT changed. This rule is valid for all instructions; only the destination register is modified by the operation.

Note: If the instruction MOV RX,RY has the register JSR (0x0C) or PCL (0x0D) as the destination, then Subroutine Call or Program Jump will be executed. Please read the main User's Manual.

Before operation

After operation

MOV RX,N Move 4-bit literal N to register RX

Syntax:	{label} MOV RX, N
Operands:	RX ∈ [R0...R15] N ∈ 0...15
Operation:	(RX) ← N
Description:	Move the 4-bit literal to the register RX. Register direct addressing must be used for RX
Flags affected:	None.

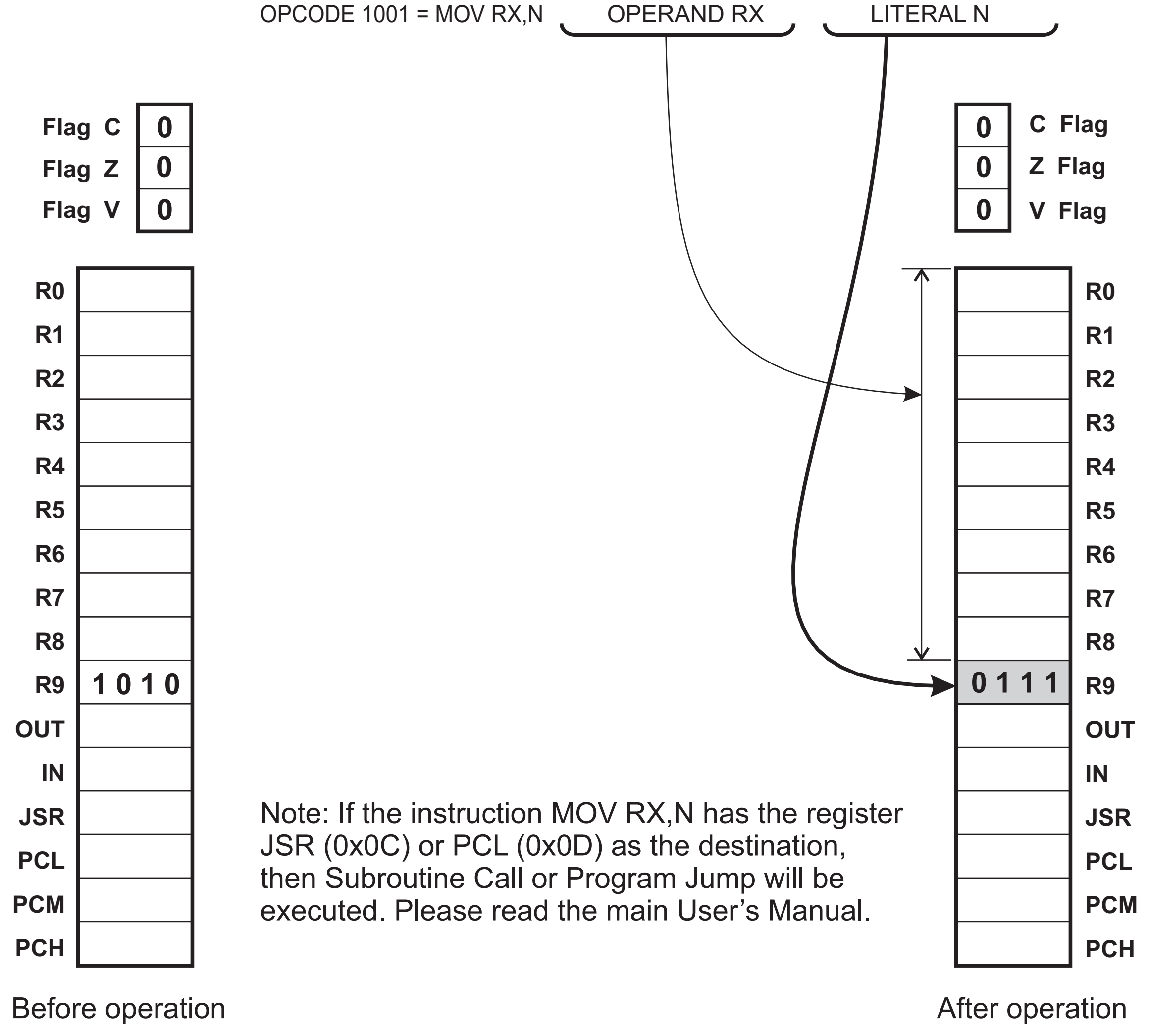
Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	**0**	**0**	**1**	**X**	**X**	**X**	**X**	**N**	**N**	**N**	**N**

The "1001" bits are the MOV RX,N opcode
The "XXXX" bits select the operand RX
The "NNNN" bits are the literal N

Example:
MOV R9, 7

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**
	OPCODE 1001 = MOV RX,N				OPERAND RX				LITERAL N			



Note: If the instruction MOV RX,N has the register JSR (0x0C) or PCL (0x0D) as the destination, then Subroutine Call or Program Jump will be executed. Please read the main User's Manual.

Before operation

After operation

MOV [XY],R0 Move contents of register R0 to data memory indirectly addressed by register RX (high nibble) and RY (low nibble)

Syntax:	{label} MOV [XY], R0
Operands:	RX ∈ [R0...R15] RY ∈ [R0...R15]
Operation:	((RX):(RY)) ← (R0)
Description:	Move the 4-bit contents of register R0 to data memory indirectly addressed by registers RX (high nibble) and RY (low nibble). Register direct addressing must be used for RX and RY.
Flags affected:	None.

Encoding:

bit 11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	X	X	X	X	Y	Y	Y	Y

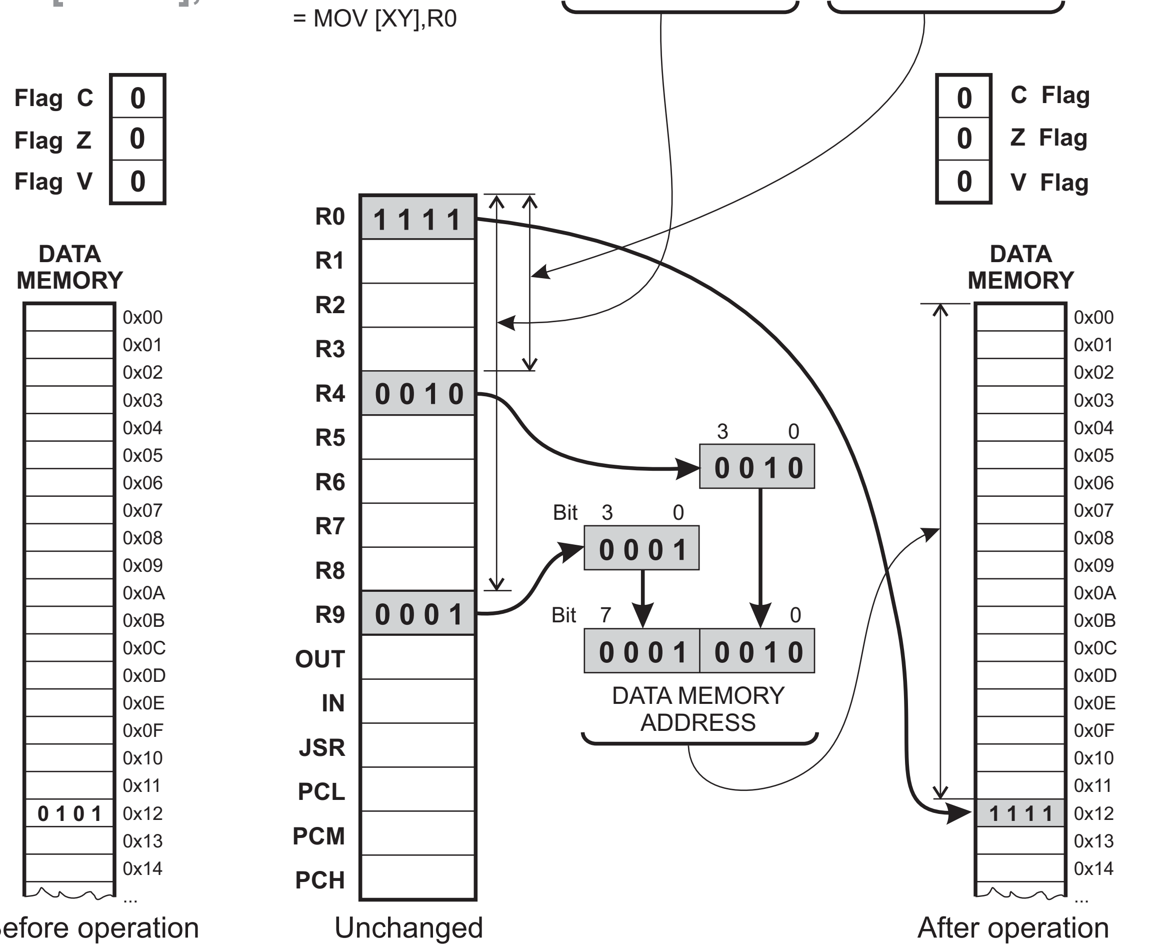
The "1010" bits are the MOV [XY],R0 opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example:

MOV [R6:R4],R0

bit 11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	0	1	0	0

OPCODE 1010 = MOV [XY],R0 | OPERAND RX | OPERAND RY



Before operation Unchanged After operation

MOV R0,[XY]
Move contents of data memory indirectly addressed by register RX (high nibble) and RY (low nibble) to register R0

Syntax:	{label} MOV R0, [XY]
Operands:	RX ∈ [R0...R15] RY ∈ [R0...R15]
Operation:	(R0) ← ((RX):(RY))
Description:	Move the 4-bit contents of data memory indirectly addressed by register RX (high nibble) and RY (low nibble) to register R0. Register direct addressing must be used for RX and RY.
Flags affected:	None.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	X	X	X	X	Y	Y	Y	Y

The "1011" bits are the MOV R0,[XY] opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example:

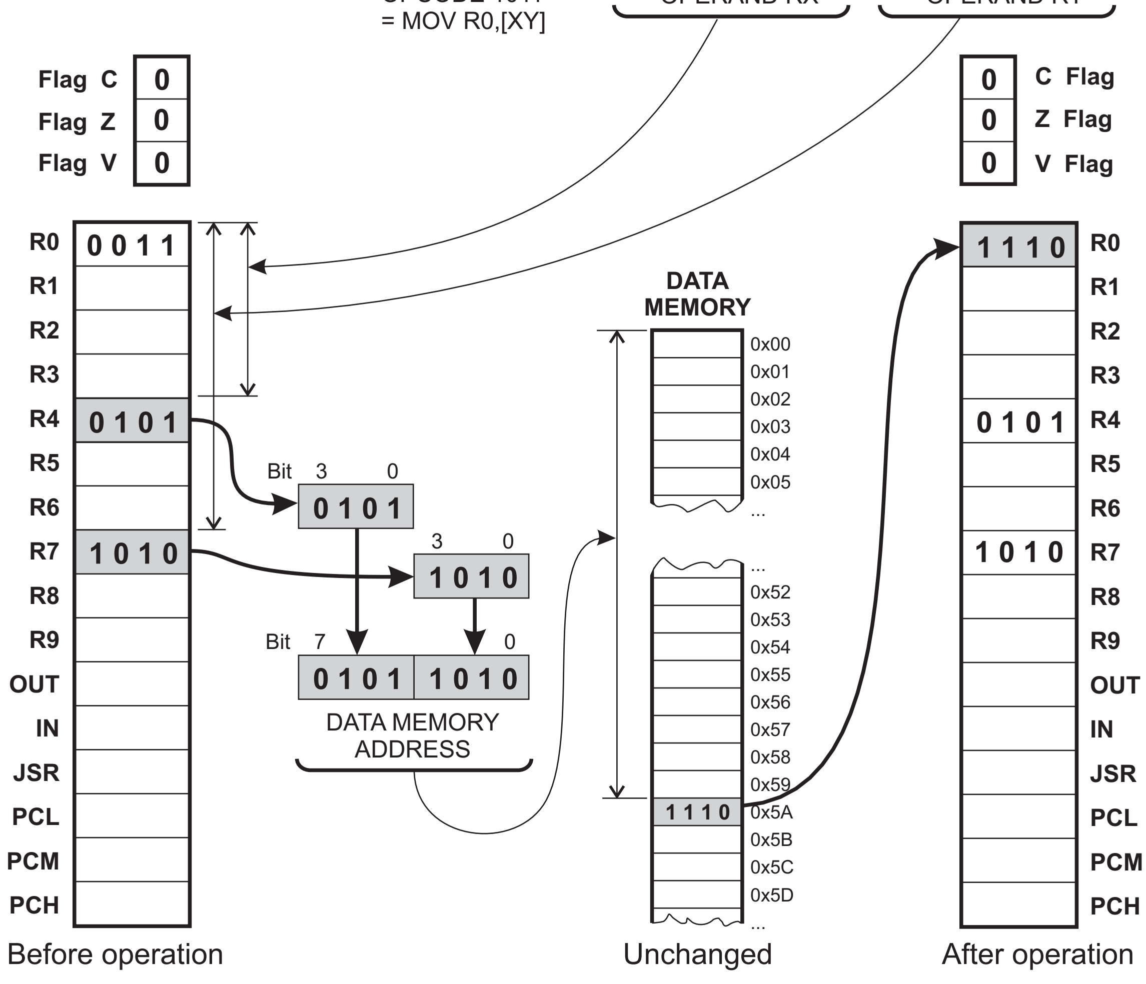
MOV R0, [R4:R7]

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	0	1	0	0	0	1	1	1

OPCODE 1011
= MOV R0,[XY]

OPERAND RX

OPERAND RY



MOV [NN],R0 Move contents of register R0 to data memory addressed by literal NN

Syntax:	{label} MOV [NN], R0
Operands:	NN ∈ [0...255]
Operation:	(NN) ← (R0)
Description:	Move the 4-bit contents of register R0 to data memory addressed by unsigned literal [NN].
Flags affected:	None.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	N	N	N	N	N	N	N	N

The "1100" bits are the MOV [NN],R0 opcode
The "NNNNNNNN" bits are unsigned literal NN

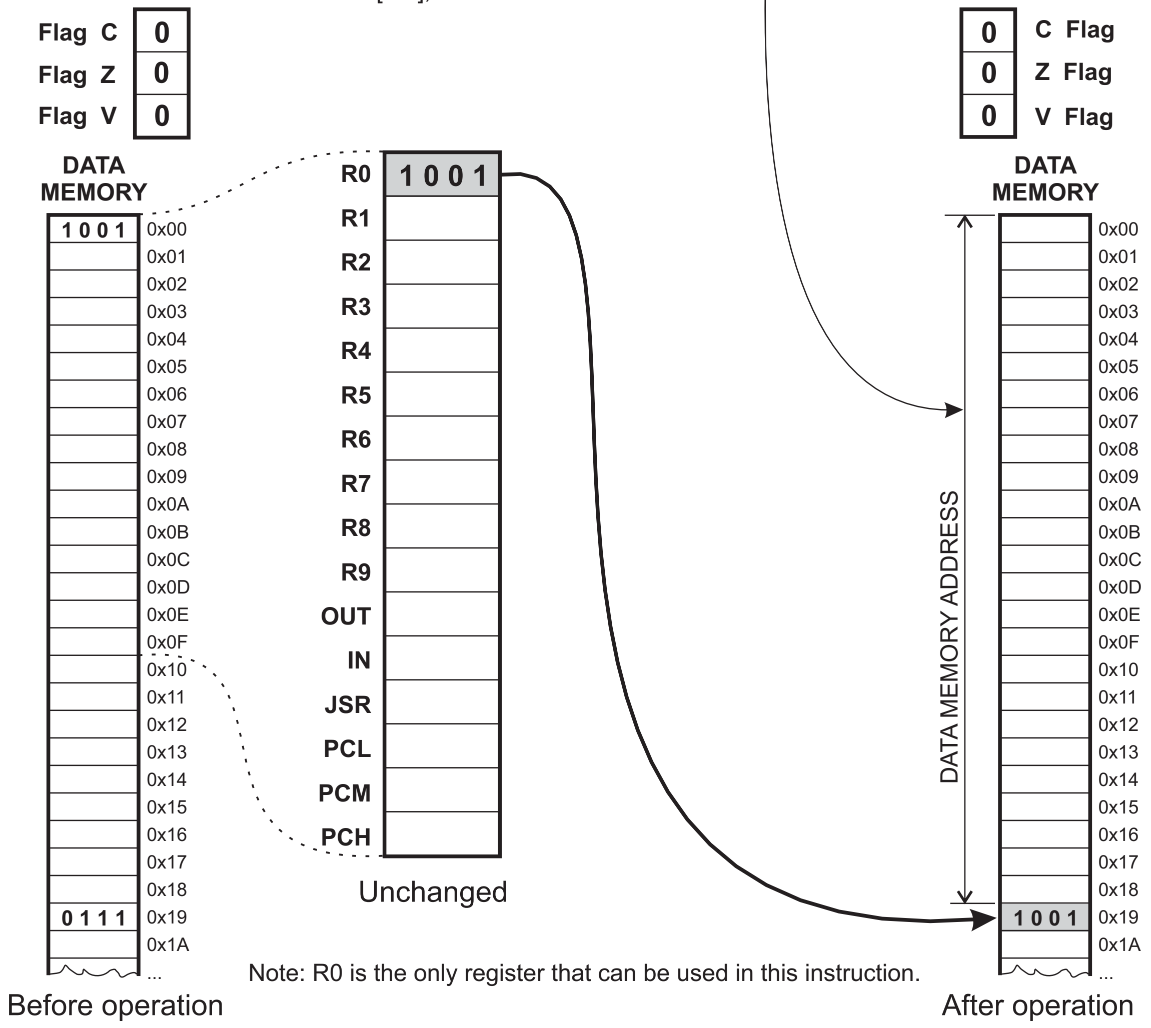
Example:

MOV [0x19], R0

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	0	0	0	1	1	0	0	1

OPCODE 1100
= MOV [NN],R0

LITERAL NN



Note: R0 is the only register that can be used in this instruction.

Before operation

After operation

MOV R0,[NN]

Move contents of data memory addressed by literal NN to register R0

Syntax:	{label} MOV R0, [NN]
Operands:	NN ∈ [0...255]
Operation:	(R0) ← (NN)
Description:	Move the 4-bit contents of data memory addressed by unsigned literal NN to register R0. Register direct addressing must be used for R0.
Flags affected:	None.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	1	N	N	N	N	N	N	N	N

The "1101" bits are the MOV R0,[NN] opcode
The "NNNNNNNN" bits are unsigned literal NN

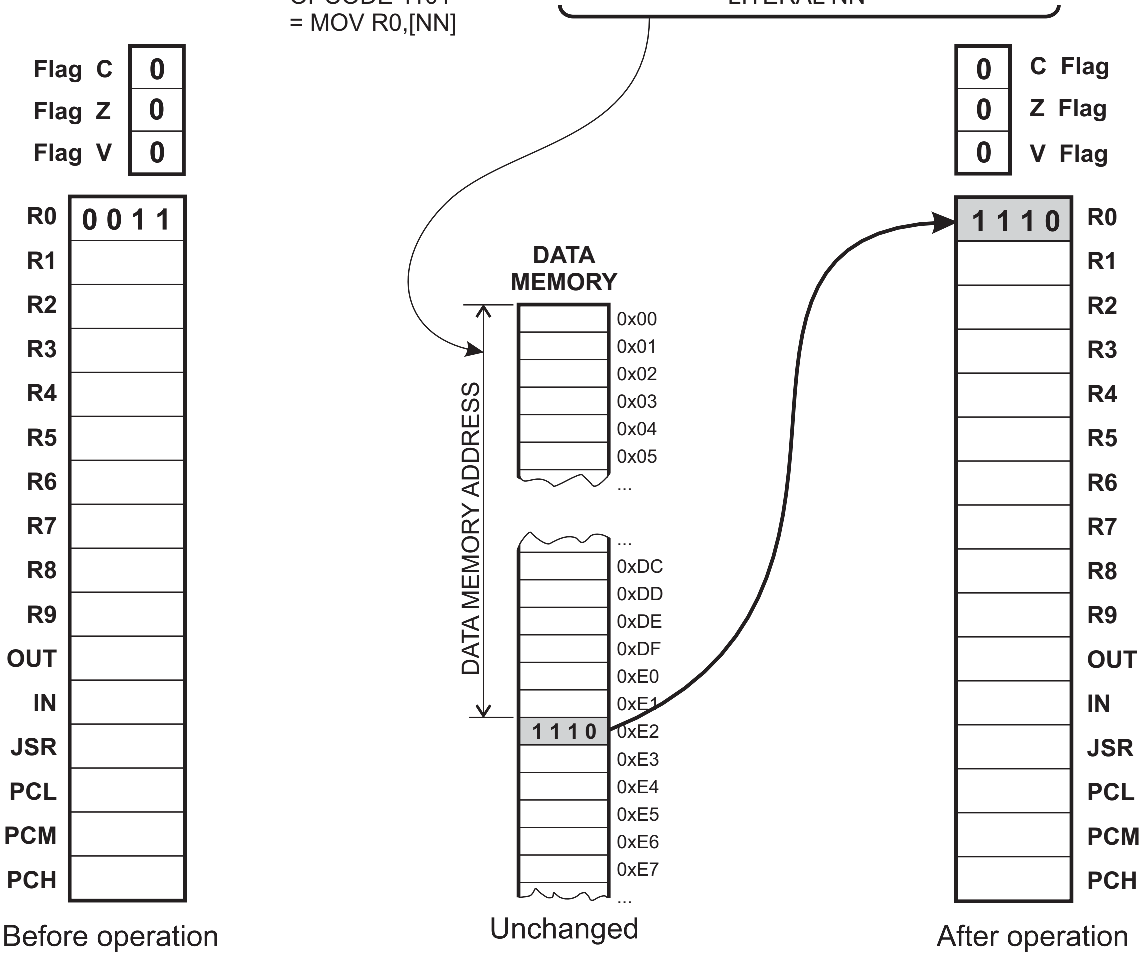
Example:

MOV R0, [0xE2]

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	1	1	1	1	0	0	0	1	0

OPCODE 1101
= MOV R0,[NN]

LITERAL NN



Before operation

Unchanged

After operation

Note: R0 is the only register that can be used in this instruction.

MOV PC,NN Load 8-bit literal to registers PCM and PCH

Syntax:	{label} MOV PC, NN

Operands:	NN ∈ [0...255]
	PCM = [R14]
	PCH = [R15]

Operation:	(R14) ← NN bit3...bit0, (R15) ← NN bit7...bit4
Description:	Move bits 3..0 of the 8-bit literal NN to R14, and bits 7...4 to R15.
Flags affected:	None.

Encoding:

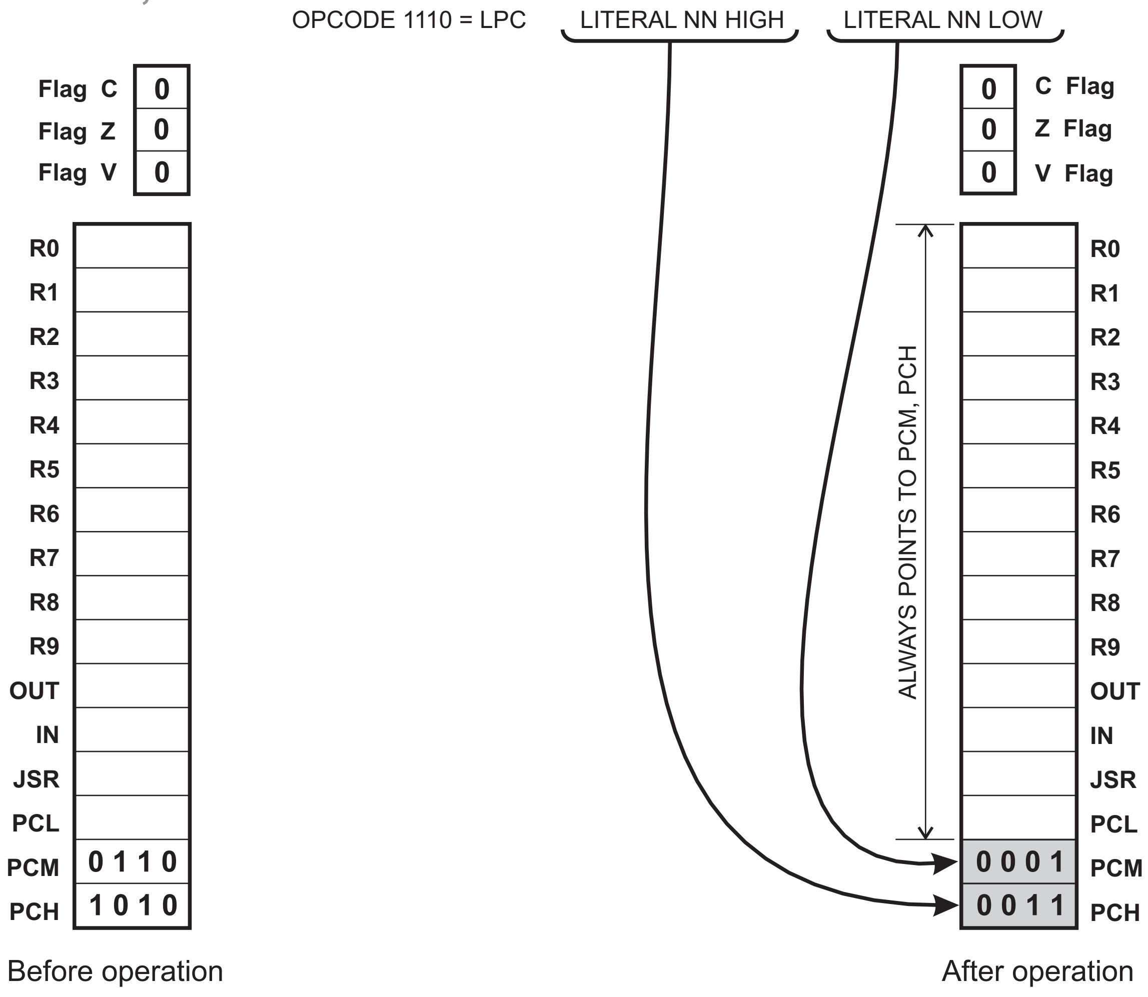
bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	N	N	N	N	N	N	N	N

The "1110" bits are the LPC #NN opcode
The "NNNNNNNN" bits are literal #NN

Example:

MOV PC,0x31

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	0	0	1	1	0	0	0	1



Before operation After operation

JR NN

Jump relative

Syntax:	{label} JR NN
Operands:	NN ∈ [-128...+127]
Operation:	Program Counter ← Program Counter + NN signed
Description:	Add signed integer value NN to the Program Counter
Flags affected:	None.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	**1**	**1**	**1**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**

The "1010" bits are the JR NN opcode
The "NNNNNNNN" bits are signed literal NN

Example:

JR -3

bit	11	10	9	8	7	6	5	4	3	2	1	0
	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**

OPCODE 1010 = JR $\underbrace{\qquad\text{LITERAL NN}\qquad}$

PROGRAM MEMORY ADDRESS WHERE THE INSTRUCTION JR IS

bit	11	10	9	8	7	6	5	4	3	2	1	0	
	0	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	64

Regularly Incremented After Instruction Fetch

NEXT PROGRAM MEMORY ADDRESS AFTER FETCHING INSTRUCTION JR

bit	11	10	9	8	7	6	5	4	3	2	1	0	
	0	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	65

Sign Bit

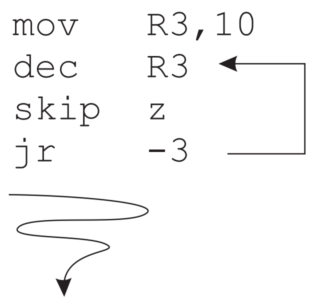
bit	7	6	5	4	3	2	1	0	
	1	**1**	**1**	**1**	**1**	**1**	**0**	**1**	- 3

bit	11	10	9	8	7	6	5	4	3	2	1	0	
	0	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	= 62

NEW PROGRAM MEMORY ADDRESS

Note 1: In this example, where NN = minus 3, program will loop not three, but two instructions back. This is because the Program Counter was already incremented after the JR instruction fetch, before the address calculation took place. Look at the program flow example at the right, program will loop 9× (plus one regular pass) before it skips instruction JR and continues further operation.

Note 2: Page crossing is allowed, even if it crosses boundary between address 1111 1111 1111 and 0000 0000 0000, so the address space can be treated as an infinite ring.



```
mov    R3,10
dec    R3
skip   z
jr     -3
```

CP R0,N

Compare register R0 with 4-bit literal

Syntax:	{label} CP R0, N
Operands:	R0 N ∈ 0...15
Operation:	(R0) - N, set flags only
Description:	Compute unsigned R0 – N and update the C and Z flags. The result of the subtraction is not stored.
Flags affected:	If (R0) > N or (R0) = N, set C. Otherwise, reset C. If (R0) = N, set Z. Otherwise, reset Z.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	N	N	N	N

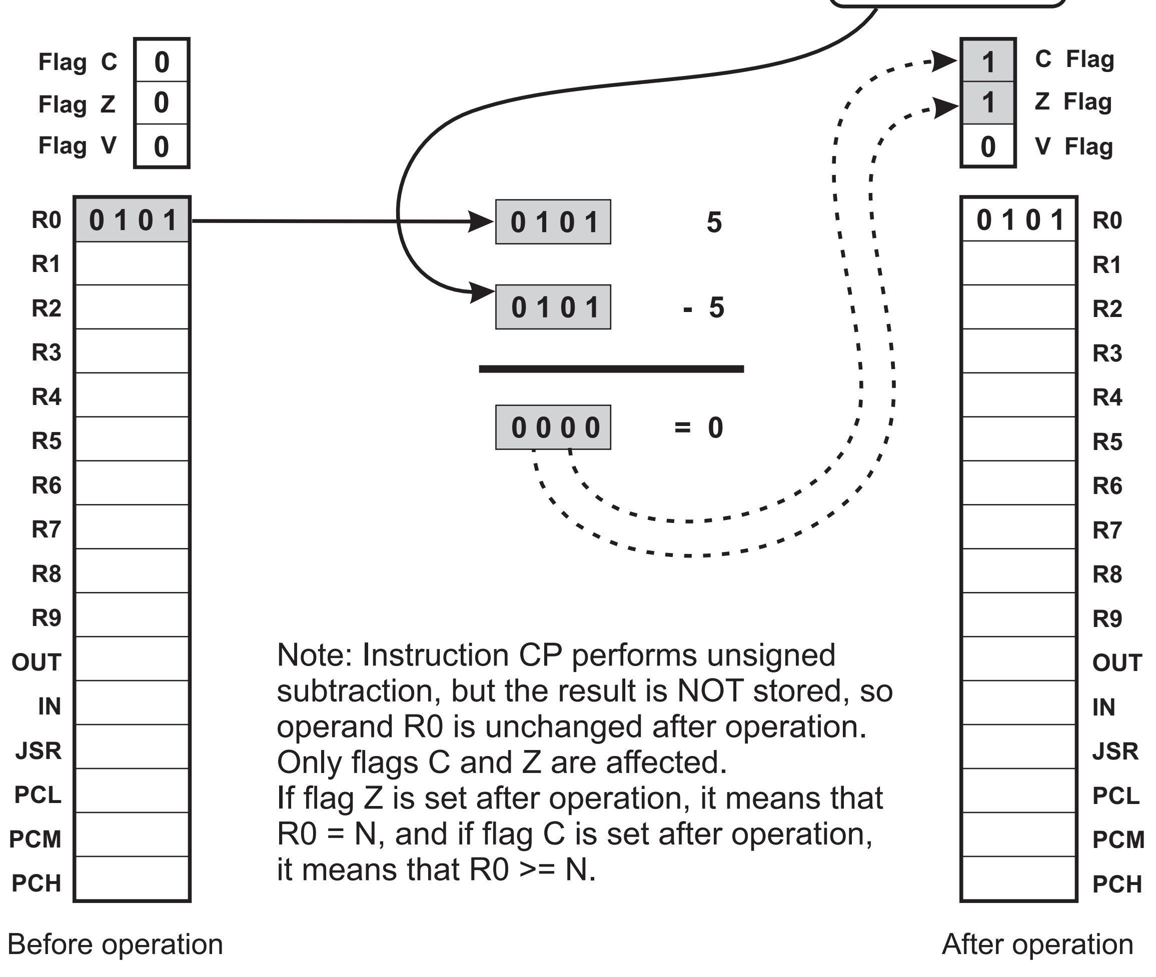
The "0000 0000" bits are the CP R0,N opcode
The "NNNN" bits are literal N

Example:

CP R0, 5

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	1	0	1

OPCODE 0000 0000 = CP R0,N   LITERAL N



Note: Instruction CP performs unsigned subtraction, but the result is NOT stored, so operand R0 is unchanged after operation. Only flags C and Z are affected.
If flag Z is set after operation, it means that R0 = N, and if flag C is set after operation, it means that R0 >= N.

ADD R0,N

Add 4-bit literal to register R0

Syntax:	{label} ADD R0, N
Operands:	R0 N ∈ 0...15
Operation:	(R0) ← (R0) + N
Description:	Add the contents of the literal N to the contents of the register R0 and place the result back in the register R0.
Flags affected:	If there is the overflow (if (R0)+N>15), set C. Otherwise, reset C. If result=0000 after operation, set Z. Otherwise, reset Z.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	N	N	N	N

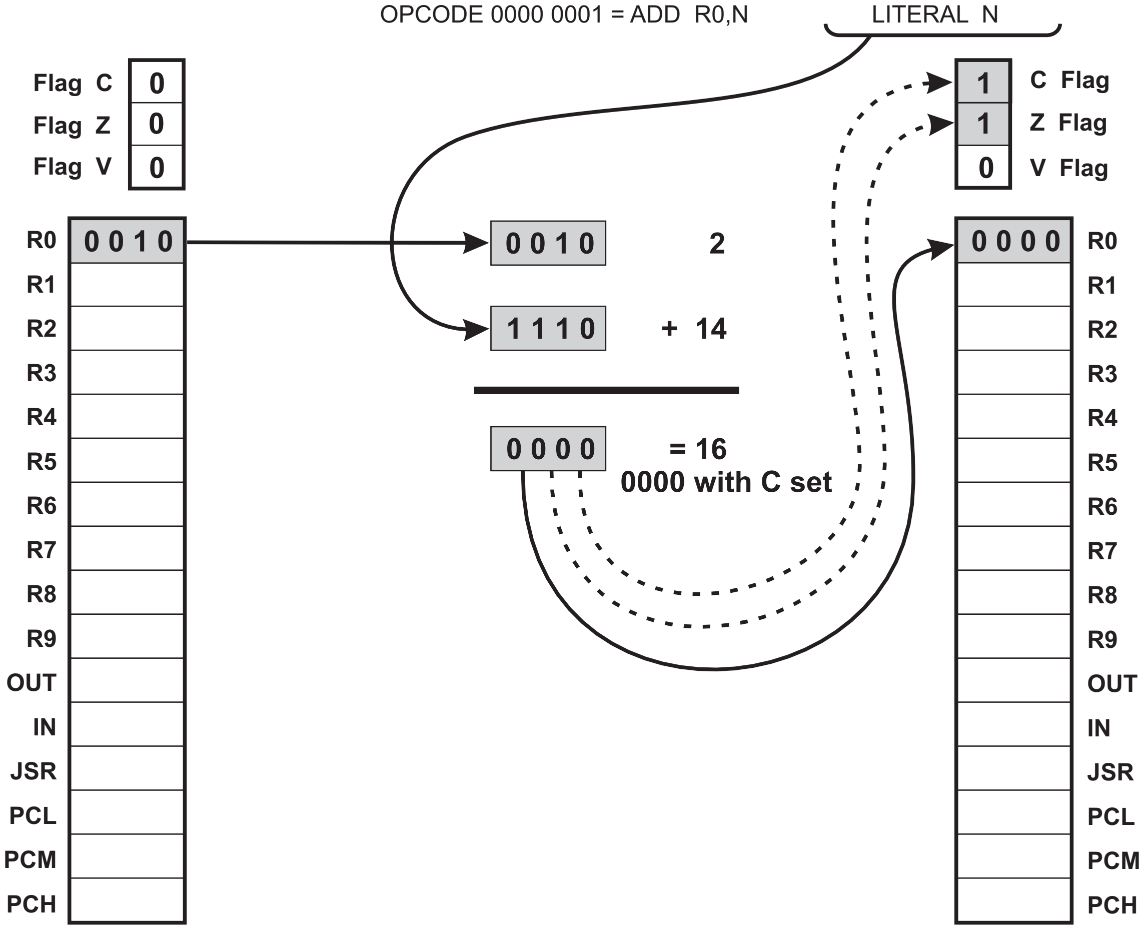
The "0000 0001" bits are the ADD R0,N opcode
The "NNNN" bits are literal N

Example:

ADD R0, 14

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	1	1	1	0

OPCODE 0000 0001 = ADD R0,N LITERAL N



Before operation After operation

INC RY

Increment register RY by 1

Syntax: {label} INC RY

Operand: RY ∈ [R0...R15]

Operation: (RY) ← (RY)+1

Description: Add 1 to the contents of the 4-bit register RY and place the result back into the register RY.

Flags affected: If result=0000 after operation, set flags Z and C. Otherwise, reset flags Z and C.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	1	0	Y	Y	Y	Y

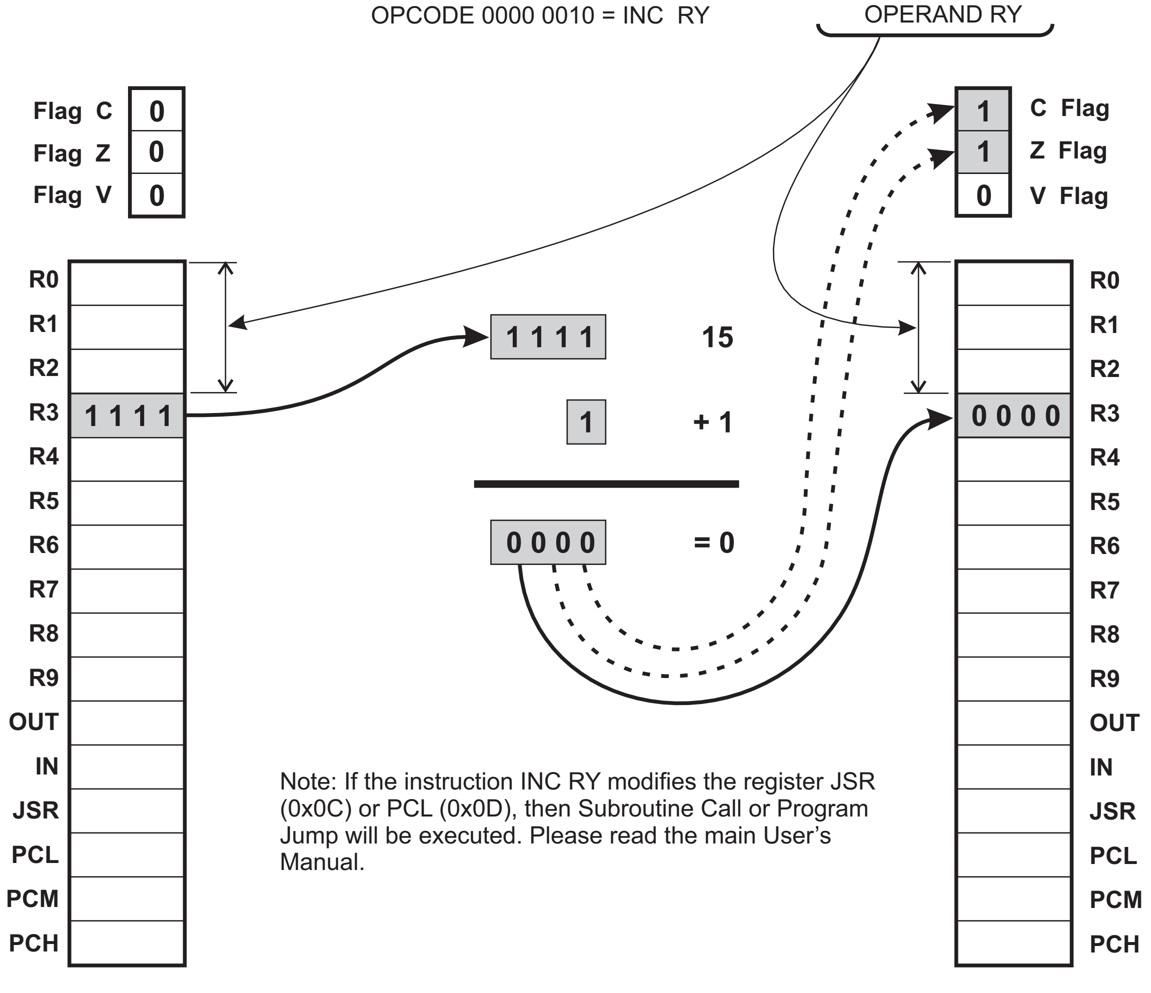
The "0000 0010" bits are the INC RY opcode
The "YYYY" bits select the operand RY

Example:

INC R3

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	1	0	0	0	1	1

OPCODE 0000 0010 = INC RY   OPERAND RY



Note: If the instruction INC RY modifies the register JSR (0x0C) or PCL (0x0D), then Subroutine Call or Program Jump will be executed. Please read the main User's Manual.

Before operation   After operation

DEC RY

Decrement register RY by 1

Syntax:	{label} DEC RY
Operand:	RY ∈ [R0...R15]
Operation:	(RY) ← (RY)-1
Description:	Subtract 1 from the contents of the 4-bit register RY and place the result back into the register RY.
Flags affected:	If result=0000 after operation, set flag Z. Otherwise, reset flag Z. If result=1111 after operation, reset flag C. Otherwise, set flag C.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	1	1	Y	Y	Y	Y

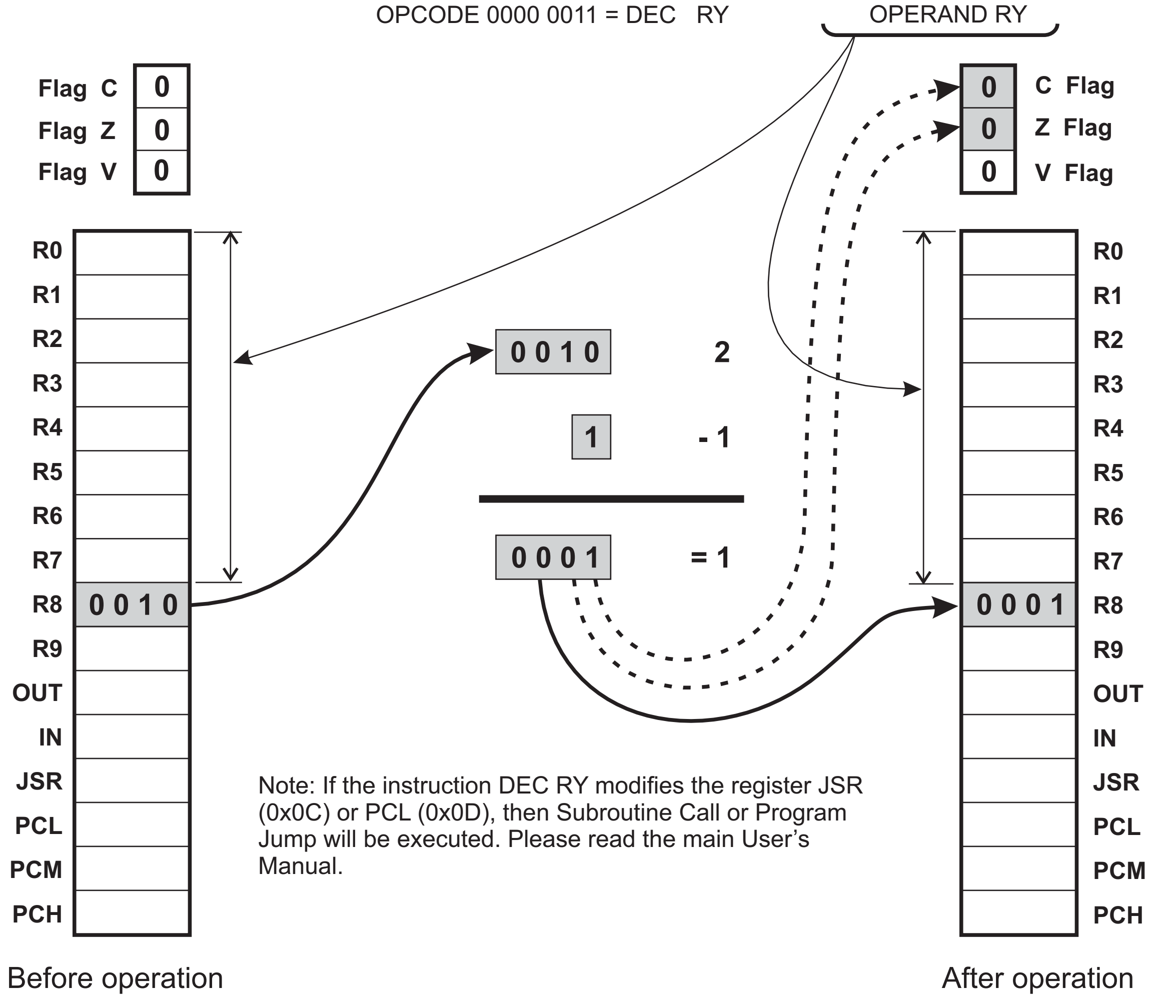
The "0000 0011" bits are the DEC RY opcode
The "YYYY" bits select the operand RY

Example:

DEC R8

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	1	1	1	0	0	0

OPCODE 0000 0011 = DEC RY   OPERAND RY



Note: If the instruction DEC RY modifies the register JSR (0x0C) or PCL (0x0D), then Subroutine Call or Program Jump will be executed. Please read the main User's Manual.

DSZ RY

Decrement register RY and, if the result is =0, skip the next instruction

Syntax:	{label} DSZ RY
Operands:	RY ∈ [R0...R15]
Operation:	(RY) ← (RY)-1, if result is =0, then PC ← PC+1
Description:	Subtract 1 from the contents of the 4-bit register RY and if result is =0, increment Program Counter by 1.
Flags affected:	None.

Encoding:

bit 11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	Y	Y	Y	Y

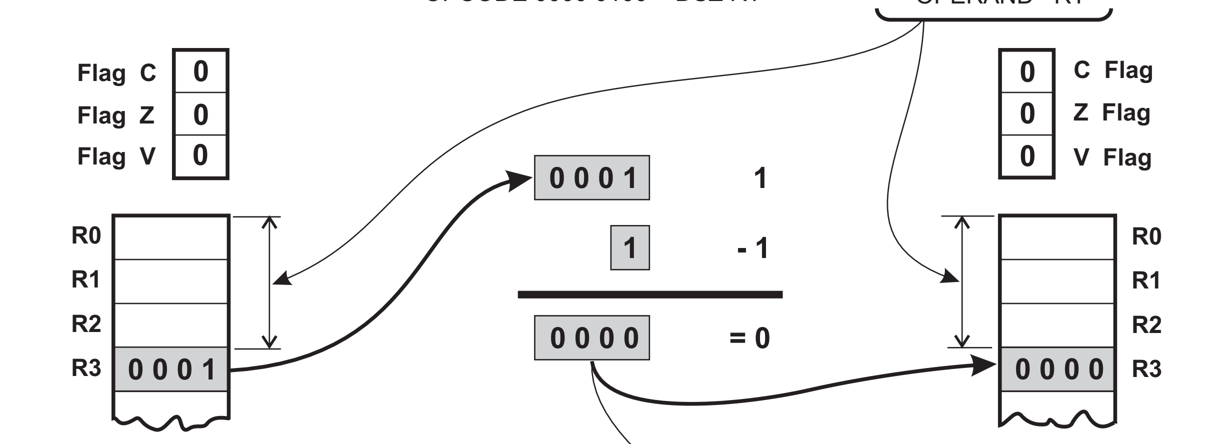
The "0000 0100" bits are the DSZ RY opcode
The "YYYY" bits are operand Y

Example:

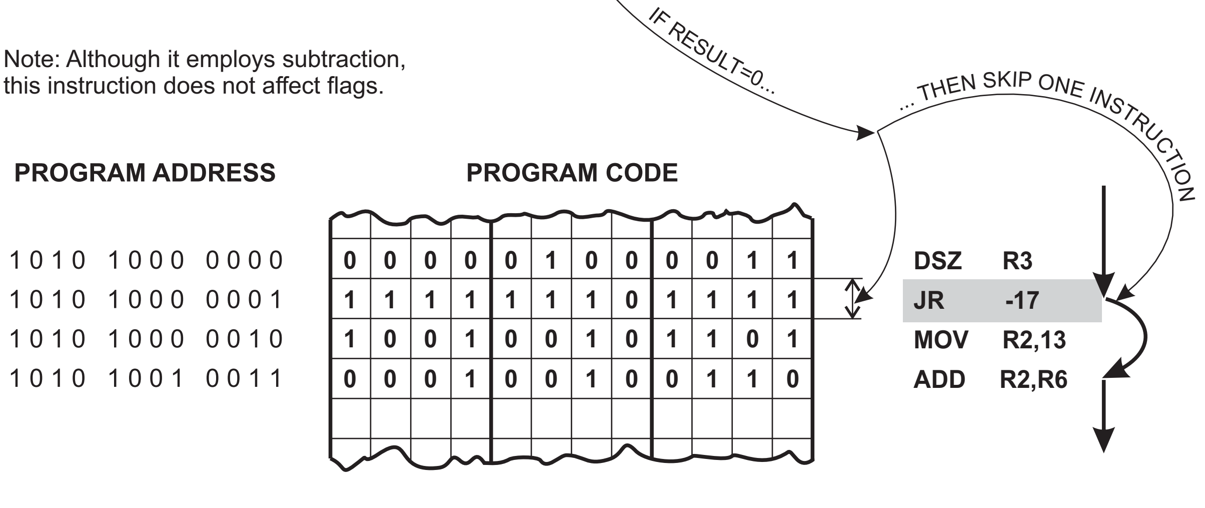
bit 11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0	0	1	1

OPCODE 0000 0100 = DSZ RY   OPERAND RY

DSZ R3



Note: Although it employs subtraction, this instruction does not affect flags.



PROGRAM ADDRESS	PROGRAM CODE												
1010 1000 0000	0	0	0	0	0	1	0	0	0	0	1	1	DSZ R3
1010 1000 0001	1	1	1	1	1	1	1	0	1	1	1	1	JR -17
1010 1000 0010	1	0	0	1	0	0	1	0	1	1	0	1	MOV R2,13
1010 1001 0011	0	0	0	1	0	0	1	0	0	1	1	0	ADD R2,R6

Note: In the example, register R3 is =0 after decrement, so the instruction DSZ R3 caused the program to skip one instruction on address 1010 1000 0001. Program execution continues at the address 1010 1000 0010.

OR R0,N

Inclusive OR register R0 with 4-bit literal N

Syntax:	{label} OR R0, N
Operands:	R0 N ∈ 0...15
Operation:	(R0) ← (R0) .OR. N, C ← 1
Description:	Compute the logical inclusive OR operation of the 4-bit register R0 with the 4-bit literal value N and place the result back into the register R0.
Flags affected:	Flag C is unconditionally set If result=0000 after operation, set Z. Otherwise, reset Z

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	0	1	N	N	N	N

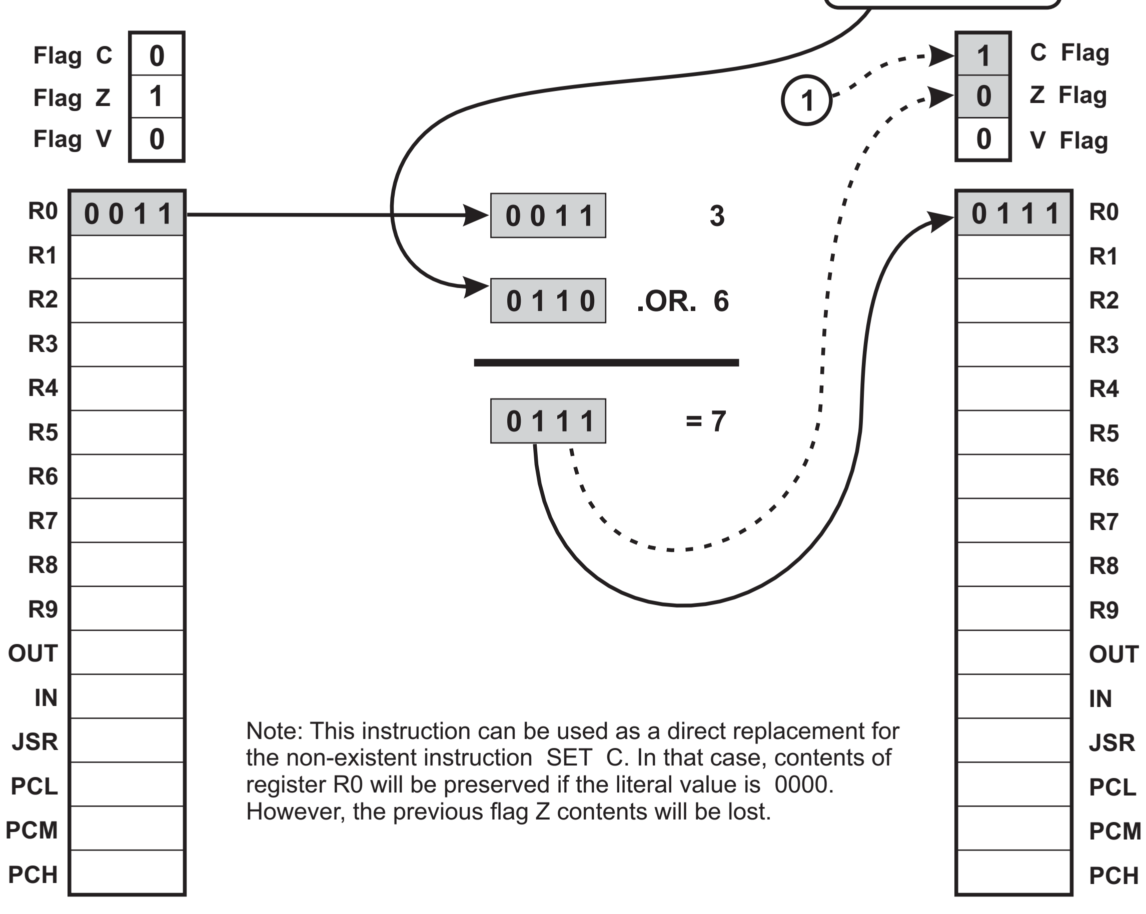
The "0000 0101" bits are the OR R0,N opcode
The "NNNN" bits are literal N

Example:

OR R0, 6

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	0	1	0	1	1	0

OPCODE 0000 0101 = OR R0,N   LITERAL N



Note: This instruction can be used as a direct replacement for the non-existent instruction SET C. In that case, contents of register R0 will be preserved if the literal value is 0000. However, the previous flag Z contents will be lost.

Before operation   After operation

AND R0,N — Logical AND register R0 with 4-bit literal N

Syntax:	{label} AND R0, N
Operands:	R0 N ∈ 0...15
Operation:	(R0) ← (R0) .AND. N, C ← 0
Description:	Compute the logical inclusive AND operation of the 4-bit register R0 with the 4-bit literal value N and place the result back into the register R0.
Flags affected:	Flag C is unconditionally reset If result=0000 after operation, set Z. Otherwise, reset Z

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	1	0	N	N	N	N

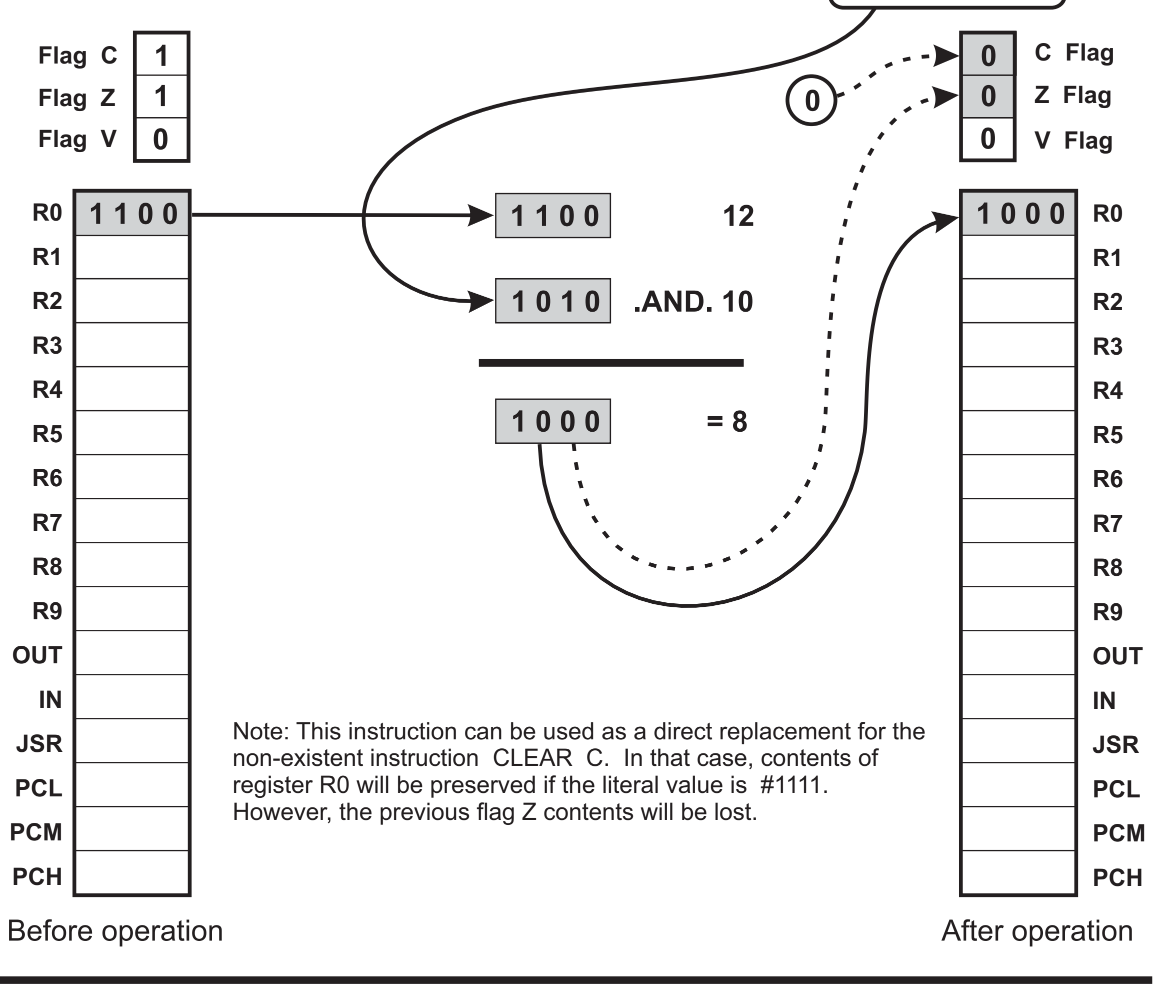
The "0000 0110" bits are the AND R0,N opcode
The "NNNN" bits are literal N

Example:

AND R0, 10

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	1	0	1	0	1	0

OPCODE 0000 0110 = AND R0,N   LITERAL N



Note: This instruction can be used as a direct replacement for the non-existent instruction CLEAR C. In that case, contents of register R0 will be preserved if the literal value is #1111. However, the previous flag Z contents will be lost.

XOR R0,N

Exclusive OR register R0 with 4-bit literal N

Syntax:	{label} XOR R0, N
Operands:	R0 N ∈ 0...15
Operation:	(R0) ← (R0) .XOR. N, C ← ¬C
Description:	Compute the logical exclusive OR operation of the 4-bit register (R0) with the 4-bit literal value N and place the result back into the register R0.
Flags affected:	Flag C is unconditionally toggled (complemented). If result=0000 after operation, set Z. Otherwise, reset Z

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	1	1	N	N	N	N

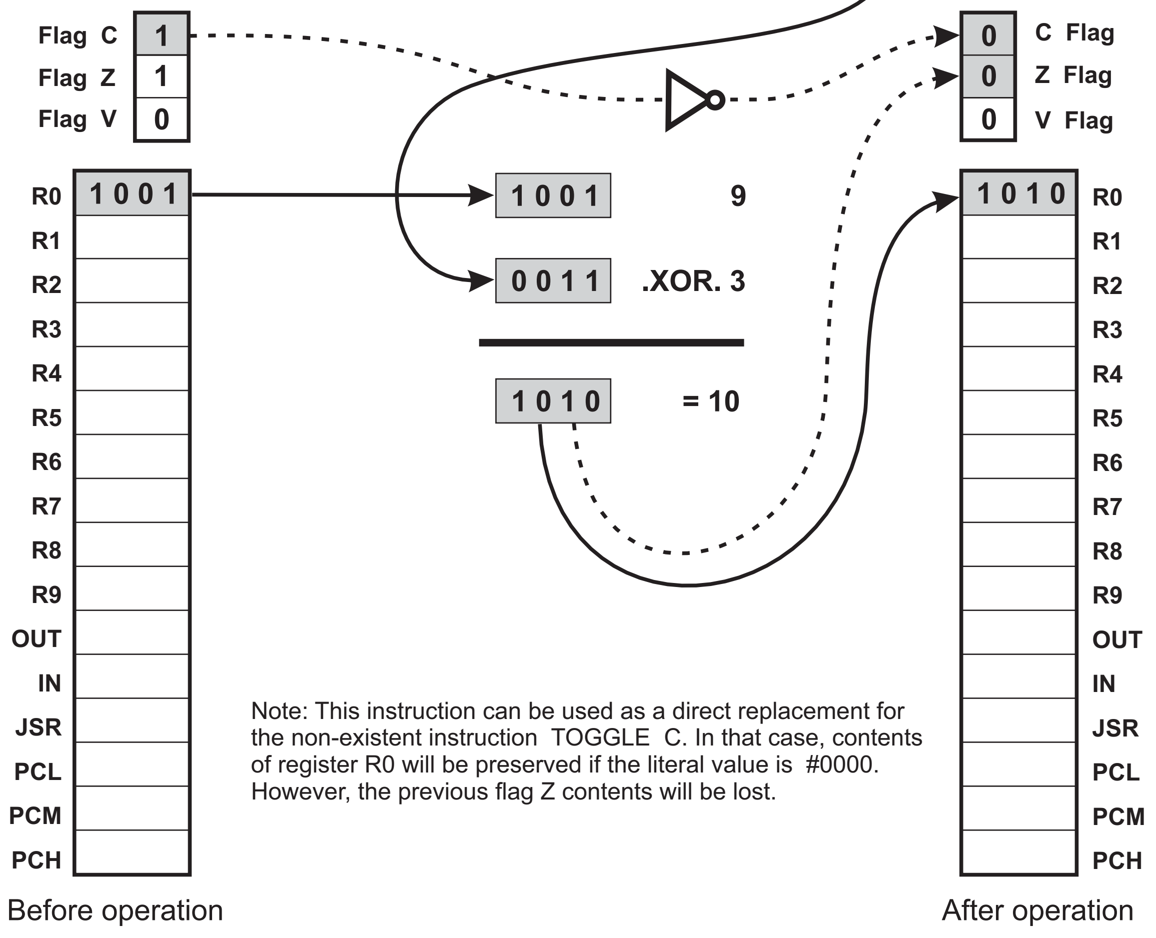
The "0000 0111" bits are the XOR R0,N opcode
The "NNNN" bits are literal N

Example:

XOR R0, 3

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	1	1	0	0	1	1

OPCODE 0000 0111 = XOR R0,N LITERAL N



Note: This instruction can be used as a direct replacement for the non-existent instruction TOGGLE C. In that case, contents of register R0 will be preserved if the literal value is #0000. However, the previous flag Z contents will be lost.

EXR N

Exchange N main registers from the page 0 with the same number of memory locations from the page 14 (page 0x0E)

Syntax:	{label} EXR N
Operands:	R0...R15 N ∈ 0...15 (Special case: N=0 means N=16)
Operation:	(R0)...(RN) ↔ (0xE0)...(0xEN)
Description:	Exchange N registers from the page 0 with registers from the alternate set in page 14. Special case: if N=0, then 16 registers will be exchanged
Flags affected:	None.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	0	N	N	N	N

The "0000 1000" bits are the EXR N opcode
The "NNNN" bits are literal N

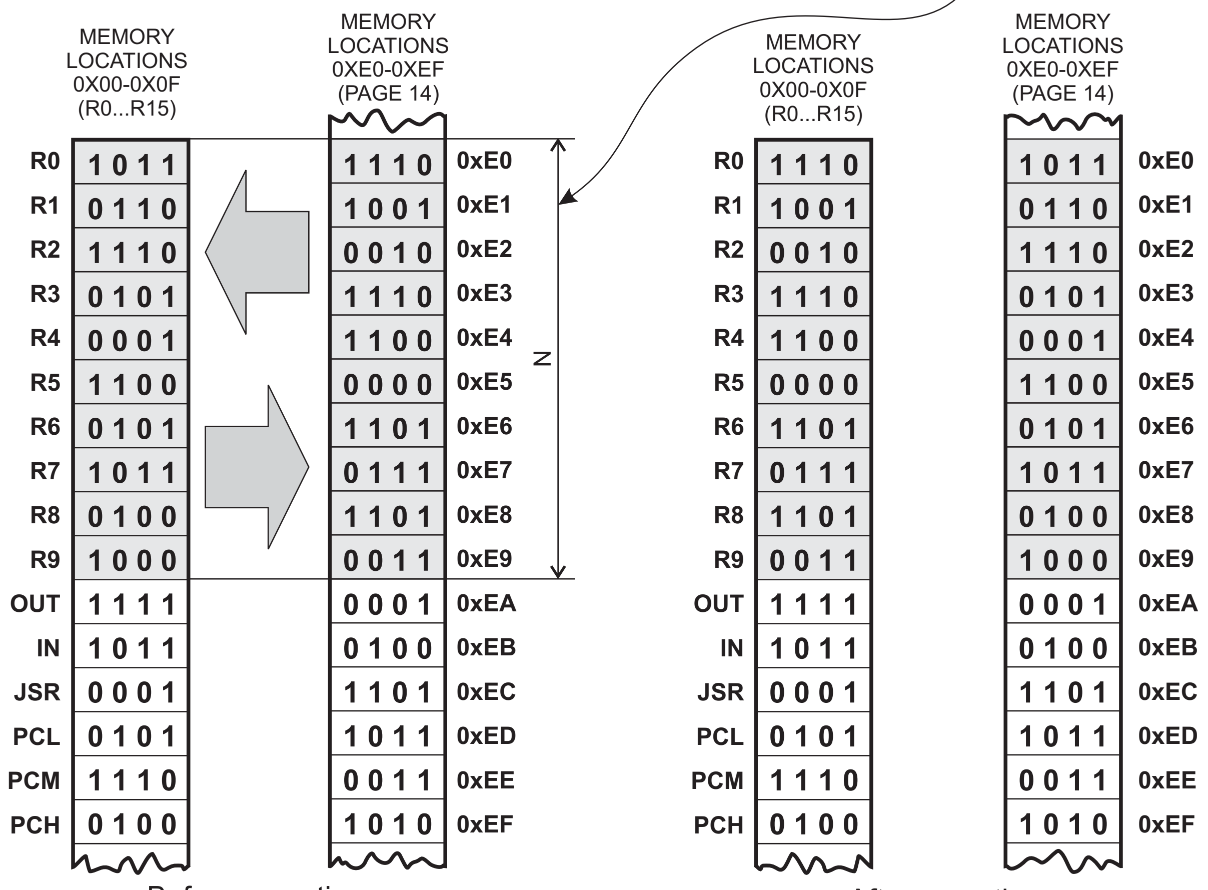
Example:

EXR 10

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	0	1	0	1	0

OPCODE 0000 10 = EXR N

LITERAL N



Note: Register R0 (data memory location 0x00) is always exchanged with memory location 0xE0, and then, if N≠1, other registers in consecutive order. (Special case: If N=0, then 16 registers will be exchanged.)

BIT RG, M Test bit M in register RG

Syntax:	{label} BIT RG, M
Operands:	RG ∈ [R0 | R1 | R2 | RS] N ∈ 0 | 1 | 2 | 3 or 0 | 1 | 2 | S
Operation:	Z ← -<bit>
Description:	Test bit N in register addressed by RG and update the Z flag.
Flags affected:	Flag C is not affected. If tested bit is =0, then set Z flag. Otherwise, reset Z.

Encoding:

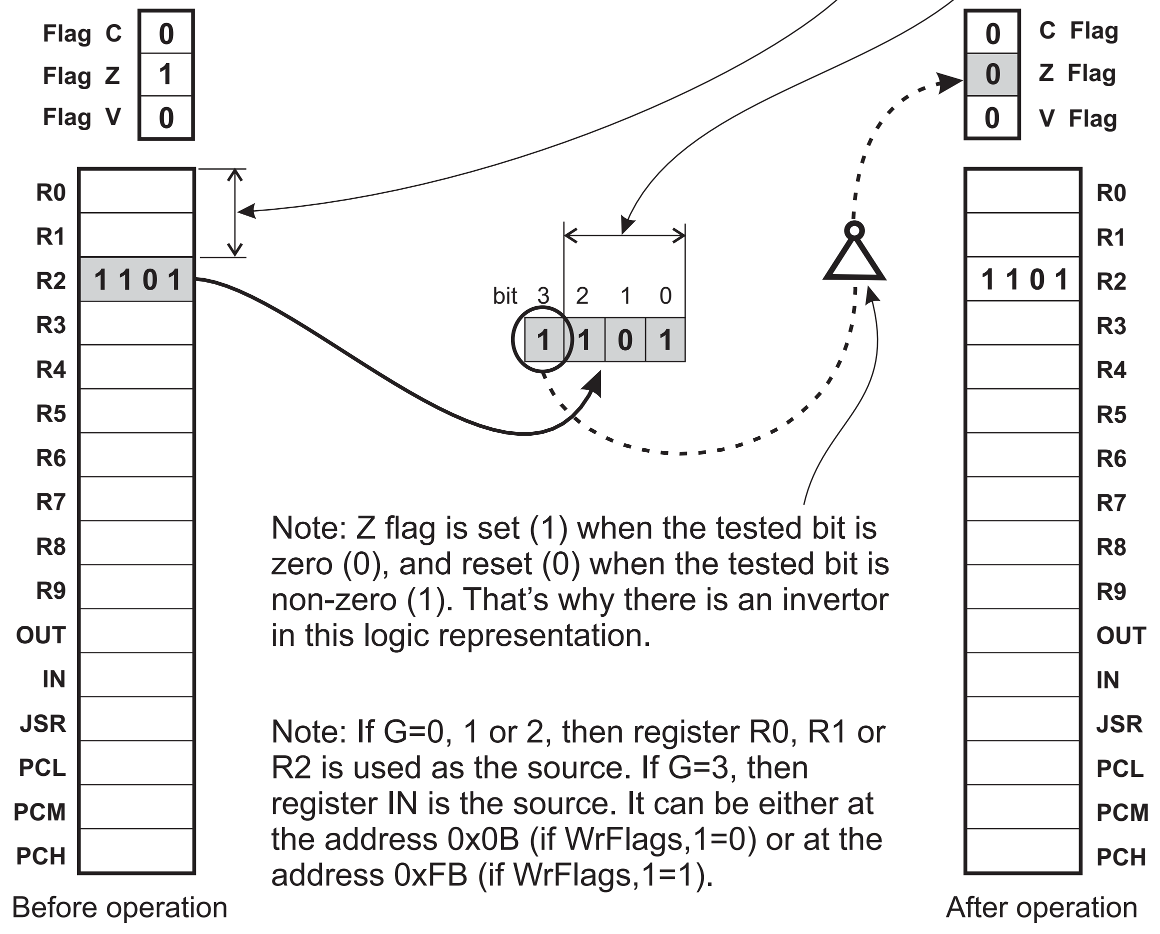
bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	1	G	G	M	M

The "0000 1001" bits are the BIT RG,M opcode
The "NNNN" bits are literal N

Example:

BIT R2, 3

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	1	1	0	1	1
	OPCODE 0000 1001 = BIT RG,M								G		M	



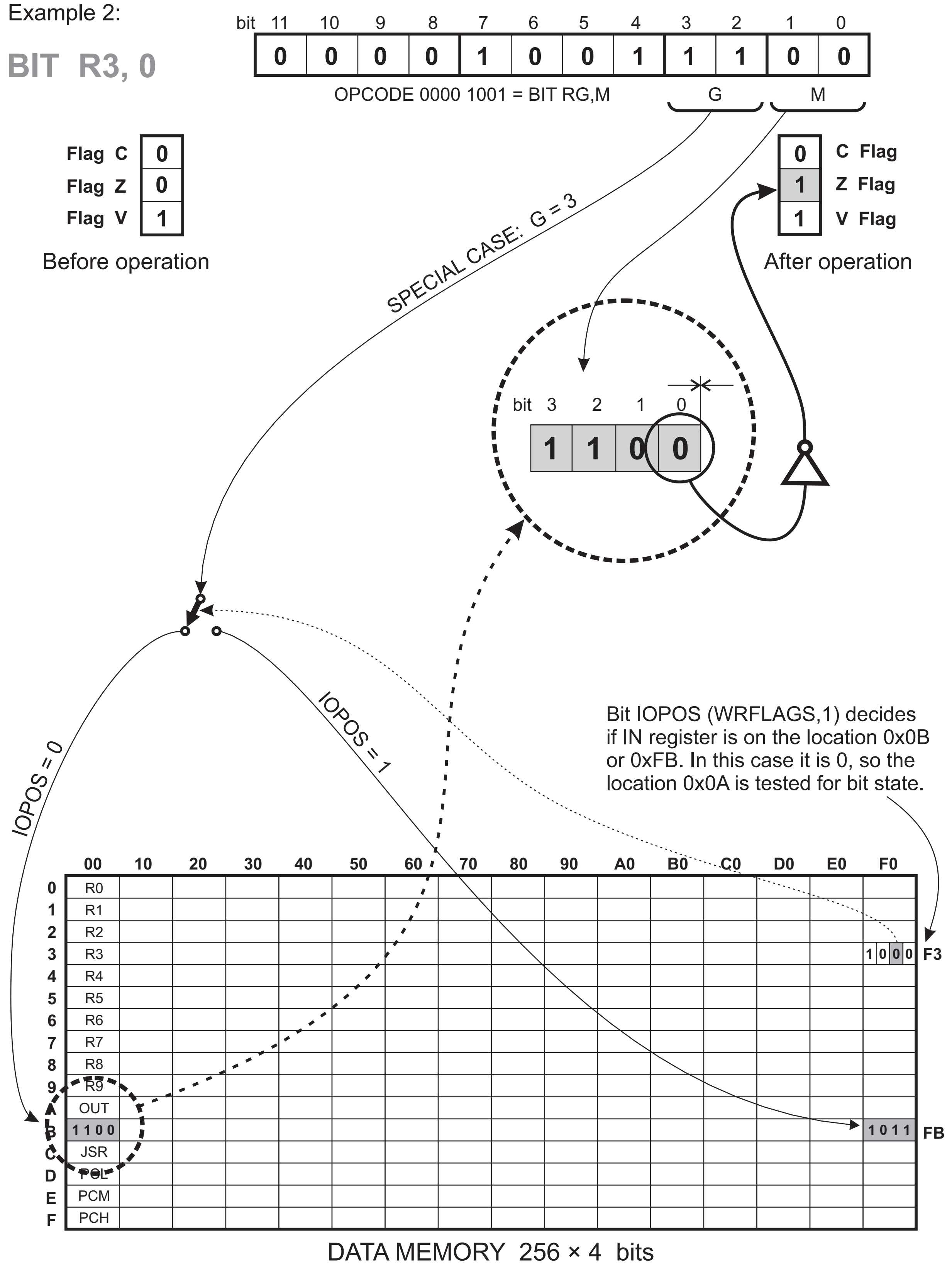
Note: Z flag is set (1) when the tested bit is zero (0), and reset (0) when the tested bit is non-zero (1). That's why there is an invertor in this logic representation.

Note: If G=0, 1 or 2, then register R0, R1 or R2 is used as the source. If G=3, then register IN is the source. It can be either at the address 0x0B (if WrFlags,1=0) or at the address 0xFB (if WrFlags,1=1).

BIT RG,M Test bit M in register RG (CONTINUED)

Example 2:

BIT R3, 0



bit 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 0 0 1 1 1 0 0
OPCODE 0000 1001 = BIT RG,M
G
M
Flag C 0
Flag Z 0
Flag V 1
Before operation
0 C Flag
1 Z Flag
1 V Flag
After operation
SPECIAL CASE: G = 3
bit 3 2 1 0
1 1 0 0
IOPOS = 0
IOPOS = 1
Bit IOPOS (WRFLAGS,1) decides if IN register is on the location 0x0B or 0xFB. In this case it is 0, so the location 0x0A is tested for bit state.
00 10 20 30 40 50 60 70 80 90 A0 B0 C0 D0 E0 F0
0 1 2 3 4 5 6 7 8 9 A B C D E F
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 OUT 1100 JSR PCL PCM PCH
1000 F3
1011 FB

DATA MEMORY 256 × 4 bits

Note: Registers WRFLAGS is described in the manual Special Function Registers

BSET RG,M Set bit M in register RG

Syntax:	{label} BSET RG, M
Operands:	RG ∈ [R0 | R1 | R2 | RS] N ∈ 0 | 1 | 2 | 3 or 0 | 1 | 2 | S
Operation:	<bit> ← 1
Description:	Set bit N in register addressed by RG.
Flags affected:	None.

Encoding:

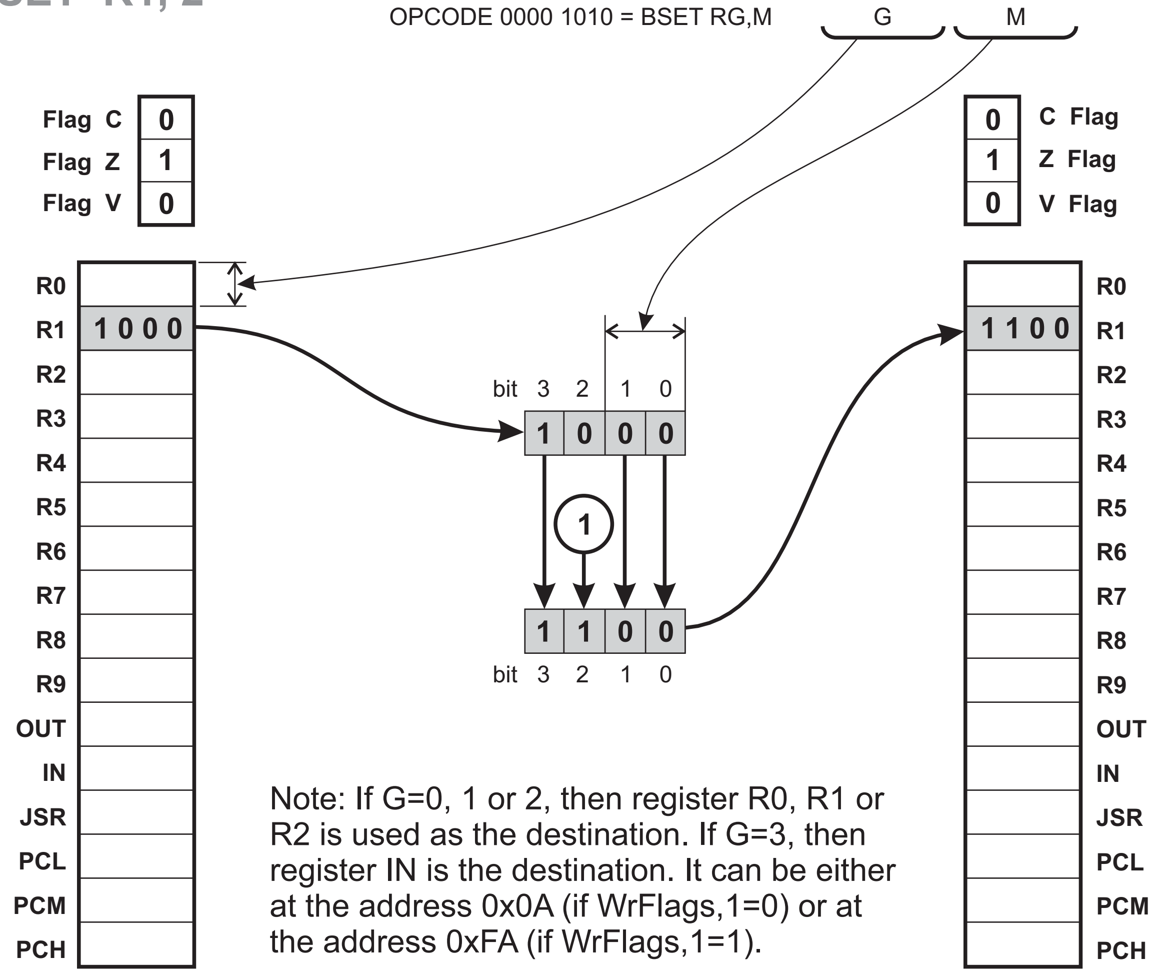
bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	1	0	G	G	M	M

The "0000 1010" bits are the BSET RG,M opcode
The "NNNN" bits are literal N

Example:

BSET R1, 2

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	1	0	0	1	1	0

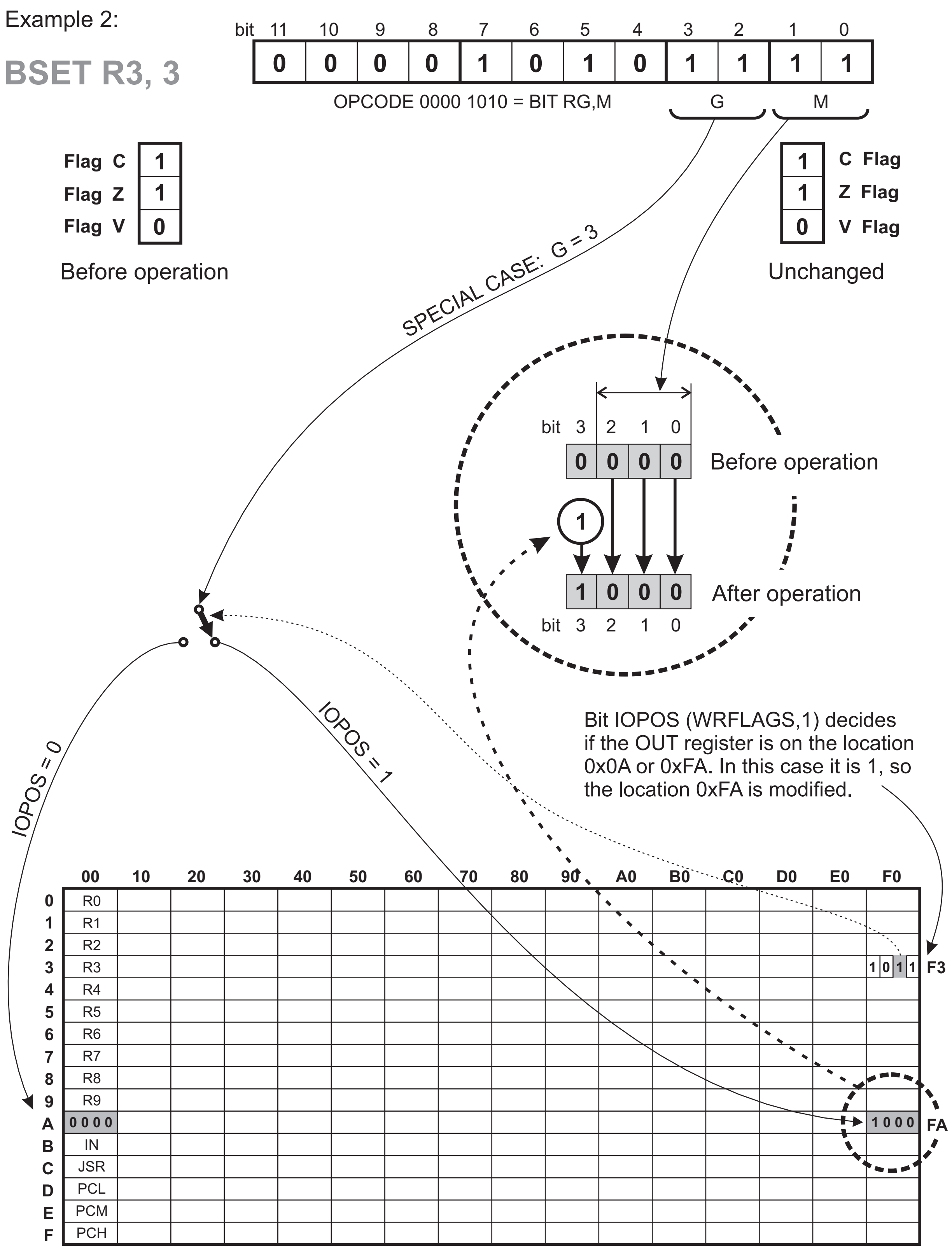


Note: If G=0, 1 or 2, then register R0, R1 or R2 is used as the destination. If G=3, then register IN is the destination. It can be either at the address 0x0A (if WrFlags,1=0) or at the address 0xFA (if WrFlags,1=1).

Before operation

After operation

BSET RG,M Set bit M in register RG (CONTINUED)



Bit IOPOS (WRFLAGS,1) decides if the OUT register is on the location 0x0A or 0xFA. In this case it is 1, so the location 0xFA is modified.

DATA MEMORY 256 × 4 bits

Note: Registers WRFLAGS is described in the manual Special Function Registers

BCLR RG,M Clear bit M in register RG

Syntax:	{label} BCLR RG, M
Operands:	RG ∈ [R0 | R1 | R2 | RS] N ∈ 0 | 1 | 2 | 3 or 0 | 1 | 2 | S
Operation:	<bit> ← 0
Description:	Clear bit #N in register addressed by RG.
Flags affected:	None.

Encoding:

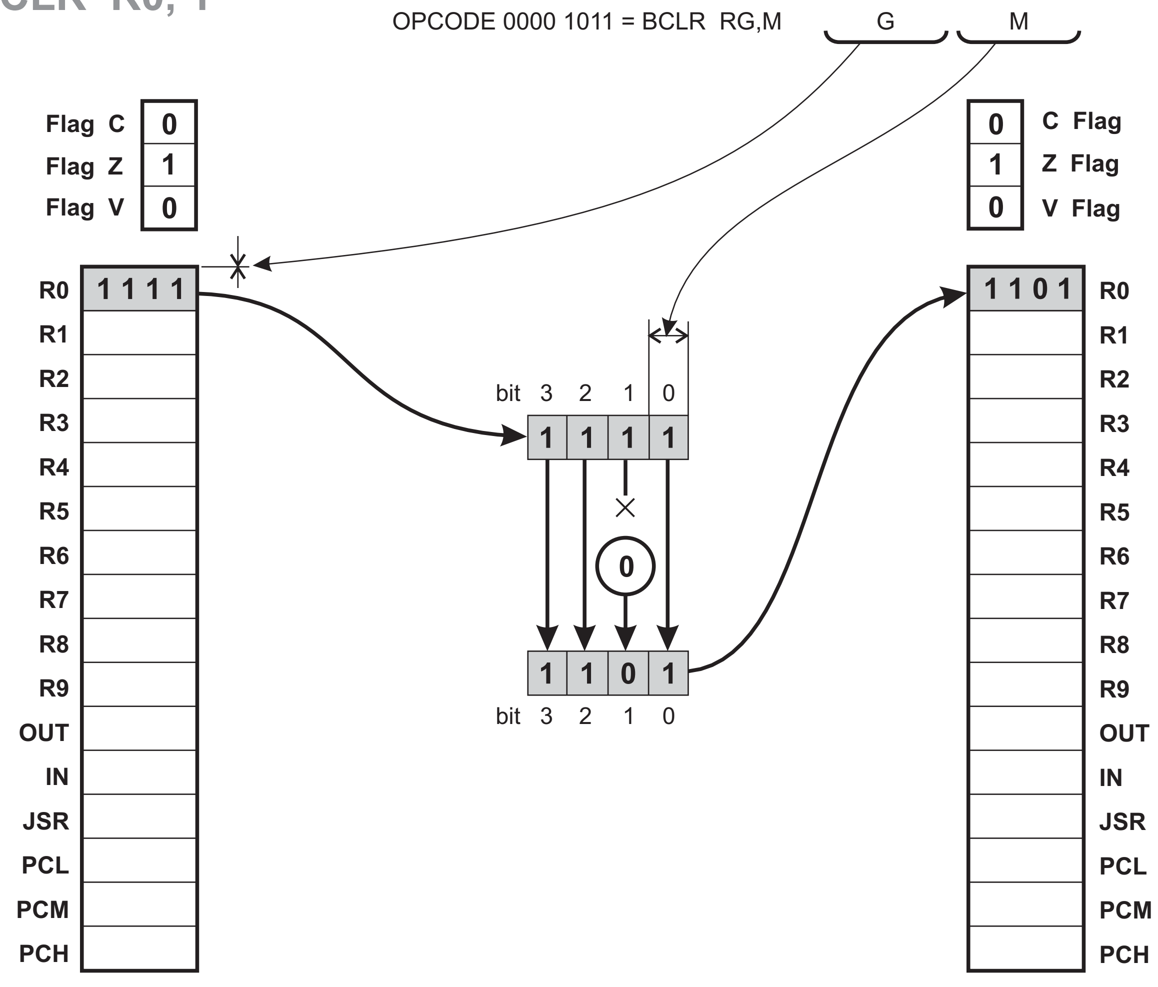
bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	1	1	G	G	M	M

The "0000 1011" bits are the BCLR RG,M opcode
The "NNNN" bits are literal N

Example:

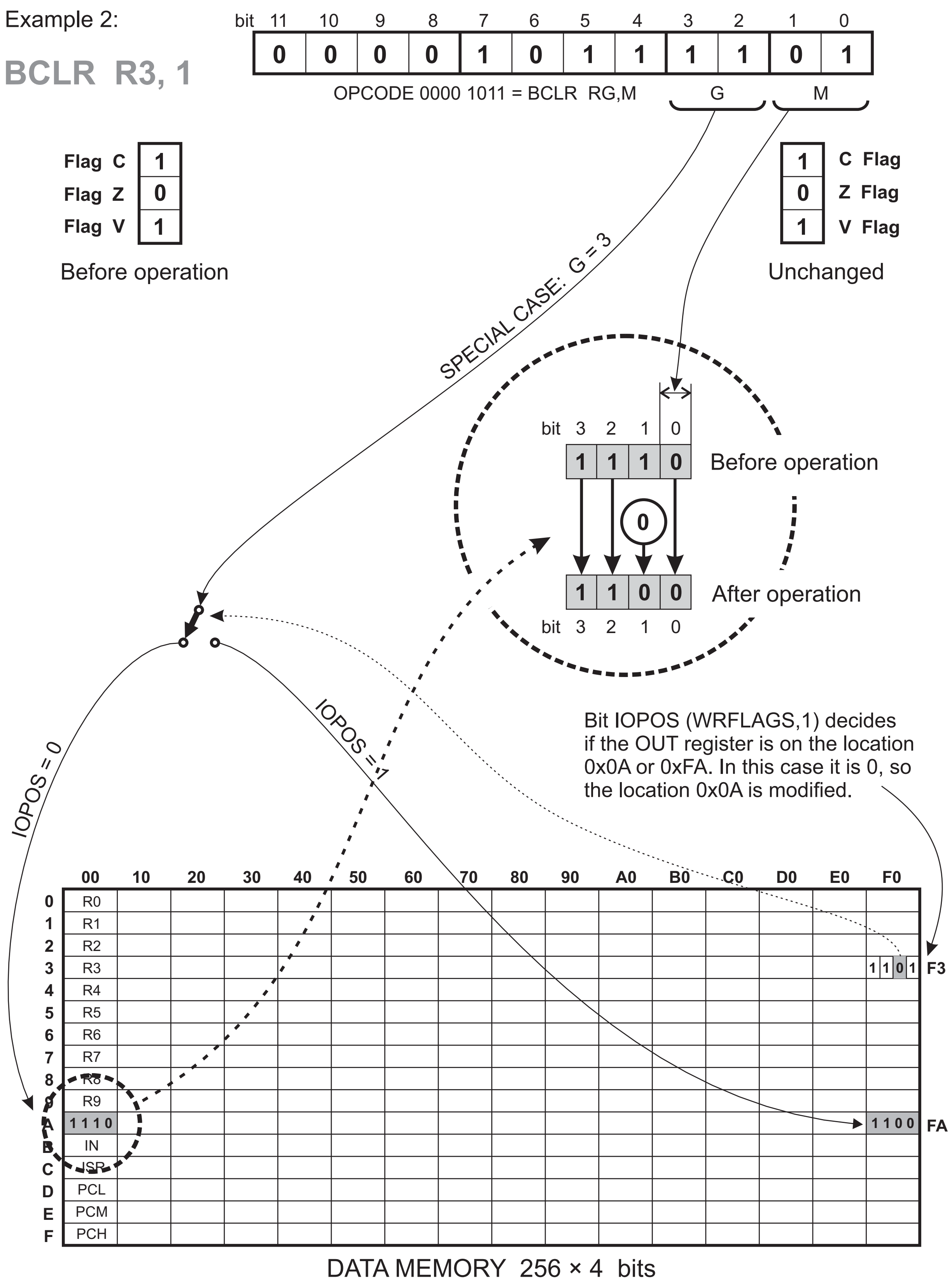
BCLR R0, 1

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	1	1	0	0	0	1



Before operation After operation

BCLR RG,M Clear bit M in register RG (CONTINUED)



Example 2:
BCLR R3, 1
bit 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 0 1 1 1 1 0 1
OPCODE 0000 1011 = BCLR RG,M
G
M
Flag C 1
Flag Z 0
Flag V 1
Before operation
1 C Flag
0 Z Flag
1 V Flag
Unchanged
SPECIAL CASE: G = 3
bit 3 2 1 0
1 1 1 0
Before operation
0
1 1 0 0
After operation
bit 3 2 1 0
IOPOS = 0
IOPOS = 1
Bit IOPOS (WRFLAGS,1) decides if the OUT register is on the location 0x0A or 0xFA. In this case it is 0, so the location 0x0A is modified.
00 10 20 30 40 50 60 70 80 90 A0 B0 C0 D0 E0 F0
0 1 2 3 4 5 6 7 8 9 A B C D E F
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 1 1 1 0 IN ISR PCL PCM PCH
1 1 0 1 F3
1 1 0 0 FA

DATA MEMORY 256 × 4 bits

Note: Registers WRFLAGS is described in the manual Special Function Registers

BTG RG,M Toggle bit M in register RG

Syntax:	{label} BTG RG, M
Operands:	RG ∈ [R0 | R1 | R2 | RS] N ∈ 0 | 1 | 2 | 3 or 0 | 1 | 2 | S
Operation:	<bit> ← -<bit>
Description:	Invert bit #N in register addressed by RG.
Flags affected:	None.

Encoding:

bit 11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	G	G	M	M

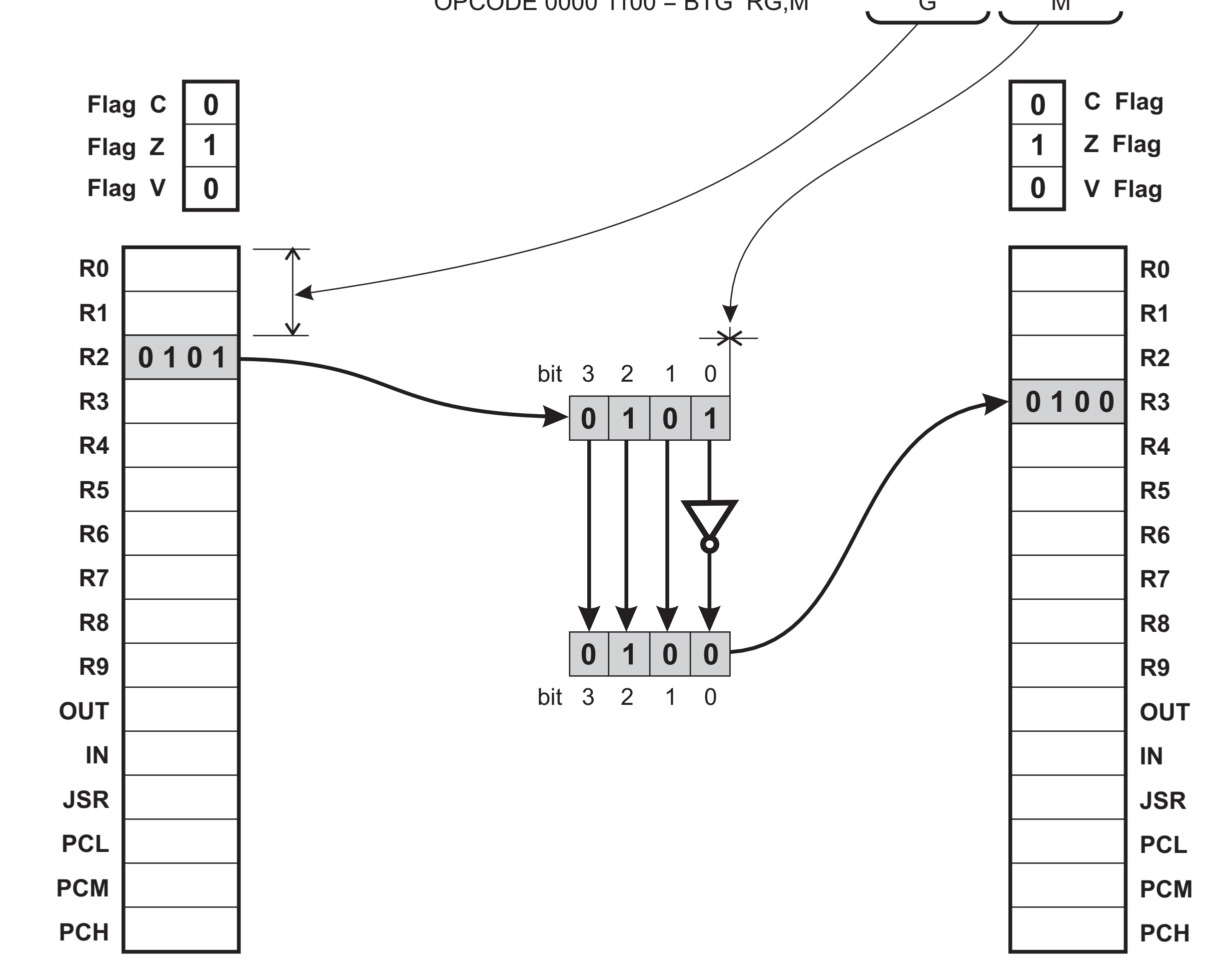
The "0000 1100" bits are the BTG RG,M opcode
The "NNNN" bits are literal N

Example:

BTG R2, 0

bit 11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	1	0	0	0

OPCODE 0000 1100 = BTG RG,M

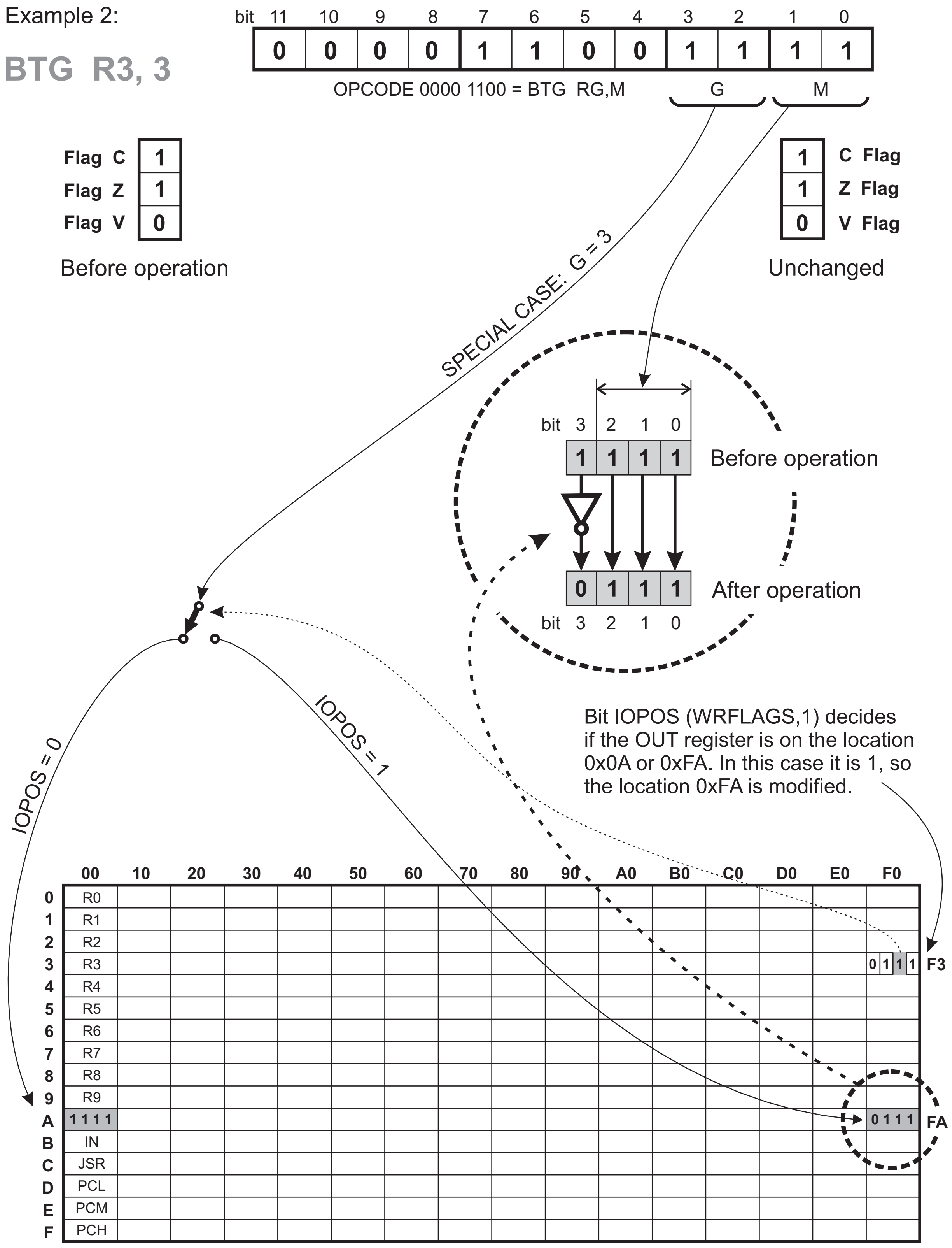


Before operation After operation

BTG RG,M Toggle bit M in register RG (CONTINUED)

Example 2:

BTG R3, 3



DATA MEMORY 256 × 4 bits

Note: Registers WRFLAGS is described in the manual Special Function Registers

RRC RY

Rotate right through Carry the register RY

Syntax:	{label} RRC RY
Operand:	RY ∈ [R0...R15]
Operation:	(C) ← (RY0), (RY3) ← (C), (RY2) ← (RY3), (RY1) ← (RY2), (RY0) ← (RY1)
Description:	Rotate the contents of the register Y one bit to the right through the Carry flag and place the result back in the register RY. The Carry flag is shifted into the Most Significant bit of the register RY, and it is then overwritten with the Least Significant bit of register RY.
Flags affected:	Bit 0 of the register RY is copied to Flag C. If result=0000 after operation, set Z. Otherwise, reset Z

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	1	0	1	Y	Y	Y	Y

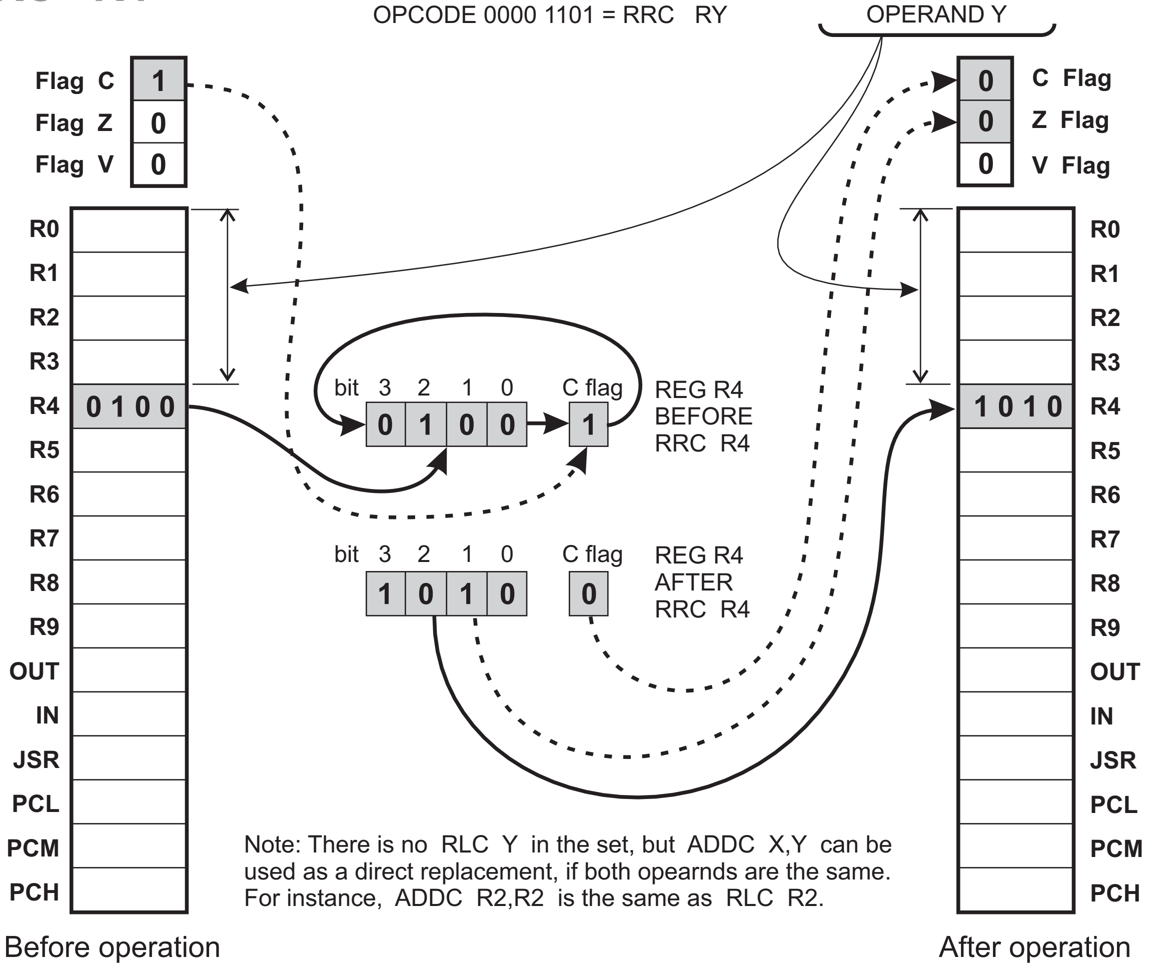
The "0000 1101" bits are the RRC RY opcode
The "YYYY" bits select the operand Y

Example:

RRC R4

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	1	0	1	0	1	0	0

OPCODE 0000 1101 = RRC RY OPERAND Y



Note: There is no RLC Y in the set, but ADDC X,Y can be used as a direct replacement, if both opearnds are the same. For instance, ADDC R2,R2 is the same as RLC R2.

RET R0,N

Return from subroutine with literal in register R0

Syntax:	{label} RET R0,N
Operands:	R0 N ∈ 0...15
Operation:	(R0) ← N SP ← SP-1 PC <11...0> ← ((SP×3+2), (SP×3+1), (SP×3))
Description:	Load R0 with literal value N and pop PC from stack
Flags affected:	None.

Encoding:

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	1	1	0	N	N	N	N

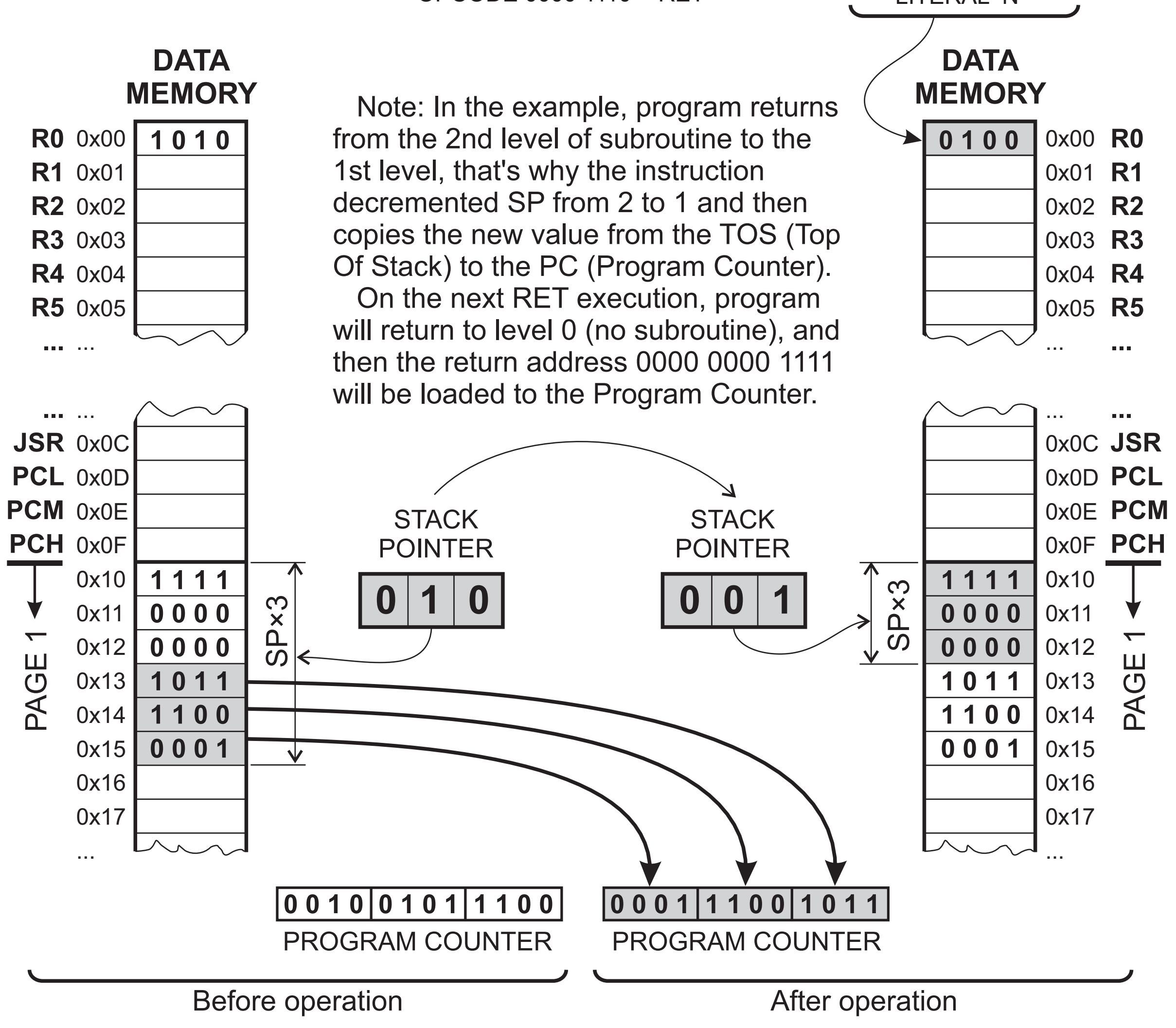
The "0000 1110" bits are the RET opcode
The "NNNN" bits are literal N

Example:

RET R0, #4

bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	1	1	0	0	1	0	0

OPCODE 0000 1110 = RET LITERAL N



Note: In the example, program returns from the 2nd level of subroutine to the 1st level, that's why the instruction decremented SP from 2 to 1 and then copies the new value from the TOS (Top Of Stack) to the PC (Program Counter).
On the next RET execution, program will return to level 0 (no subroutine), and then the return address 0000 0000 1111 will be loaded to the Program Counter.

SKIP F, M

Skip next M instructions conditionally

Syntax:	{label} SKIP z|nz|c|nc, M
Operands:	F ∈ z | nz | c | nc M ∈ 0...3 (Special case: M=0 means M=4)
Operation:	If condition=true, (PC) ← (PC+M)
Description:	If condition=true, skip the next M instructions. Special case: if M=0, then skip 4 instructions.
Flags affected:	None.

Encoding:

bit 11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**F**	**F**	**M**	**M**

The "0000 1111" bits are the SKIP opcode
The "FF" bits are condition code (see the Condition Table)
The "MM" bits are number of skip steps ("00" = "4")

Condition/Skip coding:

Condition table

CONDITION CODE F	CONDITION
0 0	C
0 1	NC
1 0	Z
1 1	NZ

Instructions skipped

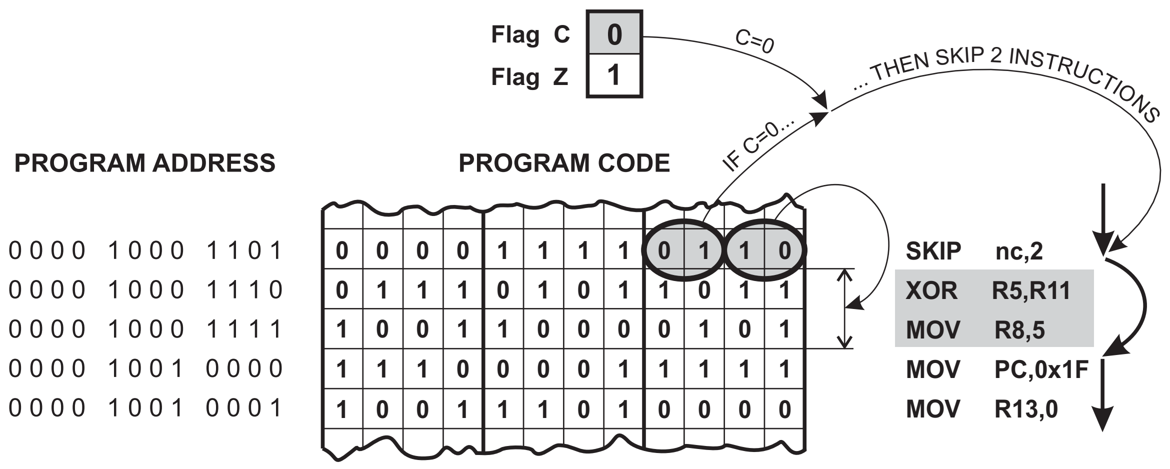
STEP COUNT M	SKIP INSTRUCTIONS
0 0	4
0 1	1
1 0	2
1 1	3

Example:

SKIP nc, #2

bit 11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**

OPCODE 0000 1101 = SKIP CONDITION COUNT



Note: In the example, flag C is =0, so the instruction SKIP nc,2 caused the program to skip two instructions on addresses 0000 1000 1110 and 0000 1000 1111. Program execution continues at the address 0000 1001 0000.