INSTRUCTION SET

Revision 4
Nov-02-2022

指令集

版本4
2022年11月02日

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”.

编码

OPC	助记	旗帜	CODE 操作码	操作数X 操作码	操作数Y
1	ADD RX、RY	V Z C	0 001	XXXYYYYY	
2	ADC RX、RY	V Z C	0 010	XXXYYYYY	
3	SUB RX、RY	V Z C	0 011	XXXYYYYY	
4	SBB RX、RY	V Z C	0 100	XXXYYYYY	
5	OR RX、RY	Z	0 101	XXXYYYYY	
6	AND RX、RY	Z	0 110	XXXYYYYY	
7	XOR RX、RY	Z	0 111	XXXYYYYY	
8	MOV RX、RY		1 000	XXXYYYYY	
9	MOV RX，#N		1 001	XX	
A	MOV [XY]，R0		1 010	XXXYYYYY	
B	MOV R0，[XY]		1 011	XXXYYYYY	
C	MOV [NN]，R0		1 100	NNNNNNNNNNNN	
D	MOV R0，[NN]		1 101	N	
E	MOV PC、NN		1 110		
F	JR NN		1 111		
00	CP R0,N	V Z C	0 0000000	NN	
01	ADD R0,N	V Z C	0 0000001	NN	
02	INC RY	Z C	0 0000010	依依	
03	DEC RY	Z C	0 0000011	依依	
04	DSZ RY		0 0000100	依依	
05	OR R0,N	Z C	0 0000101	NN	
06	AND R0,N	Z C	0 0000110	NN	
07	XOR R0,N	Z C	0 0000111	NN	
08	EXR N		0 0001000	NN	
09	BIT RG,M	Z C	0 0001001	GGM	
0A	BSET RG,M		0 0001010	GGM	
0B	BCLR RG,M		0 0001011	GGM	
0C	BTG RG,M		0 0001100	GGM	
0D	RRC RY	Z C	0 0001101	依依	
0E	RET R0,N		0 0001110	NN	
0F	SKIP F,M		0 000 11 11	FFMM	

注意事项：模式SS和RUN支持所有指令，ALU模式仅支持带有三角符号 “ “的指令，但与SS和RUN模式不同。
请参阅 “ALU模式下的指令” 一节。

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

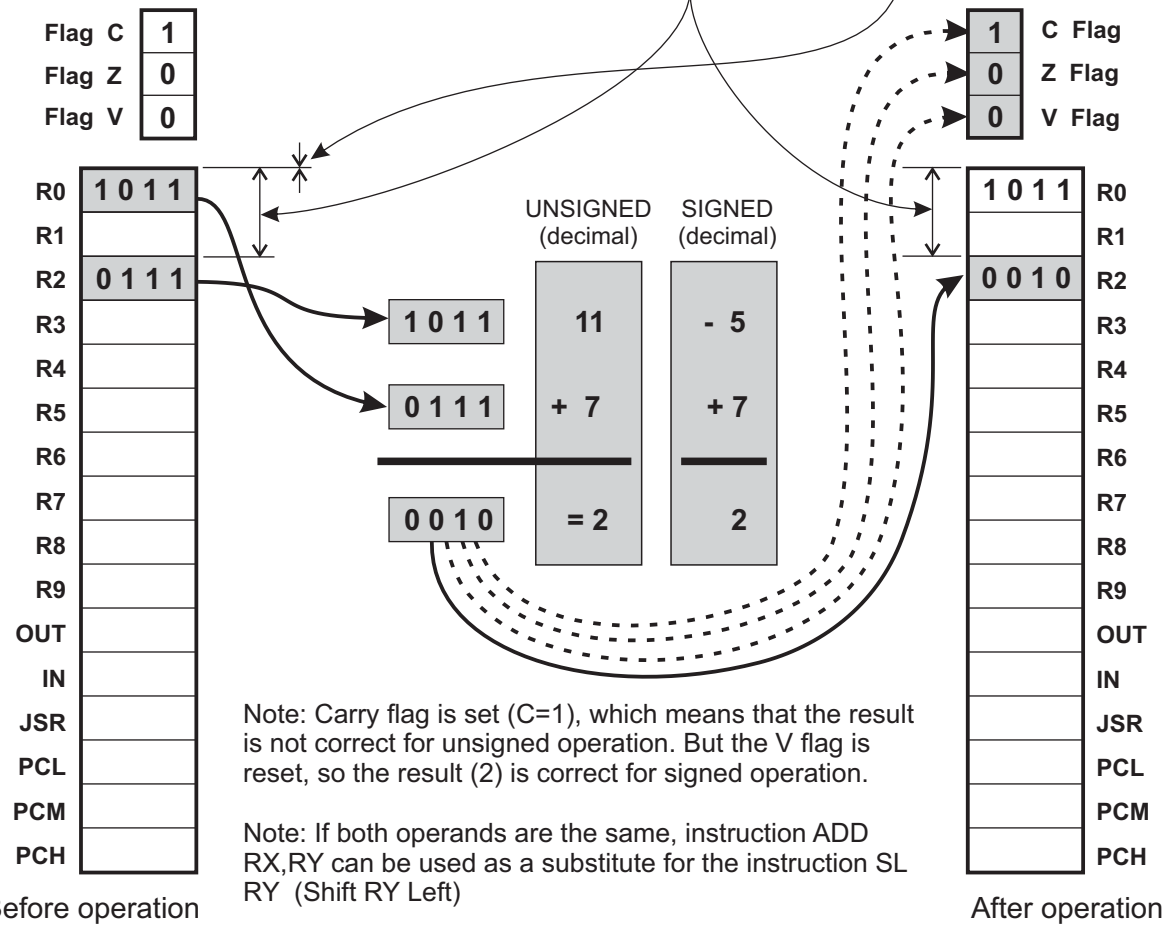
The "0001" bits are the ADD RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

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

ADD RX，RY —添加寄存器RX和RY

语法：　　{label}添加　　RX，　RY

操作数：　　RX ∈ [R0. R15]
RY ∈ [R0. R15]

操作方式：　　(RX)←（RX）+（RY）

产品描述：　　将寄存器RY的内容与寄存器RX的内容相加，并将结果放入寄存器RX。
RX和RY必须使用寄存器直接寻址。

受影响的旗帜：　　如果存在溢出（如果（RX）+（RY）> 15），则设置C。否则，重置C。如果操作后结果= 0000，则设置Z。否则，重置Z。对于有符号2的补码运算：如果有溢出，则设置V。否则，重置V。

编码方式：

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

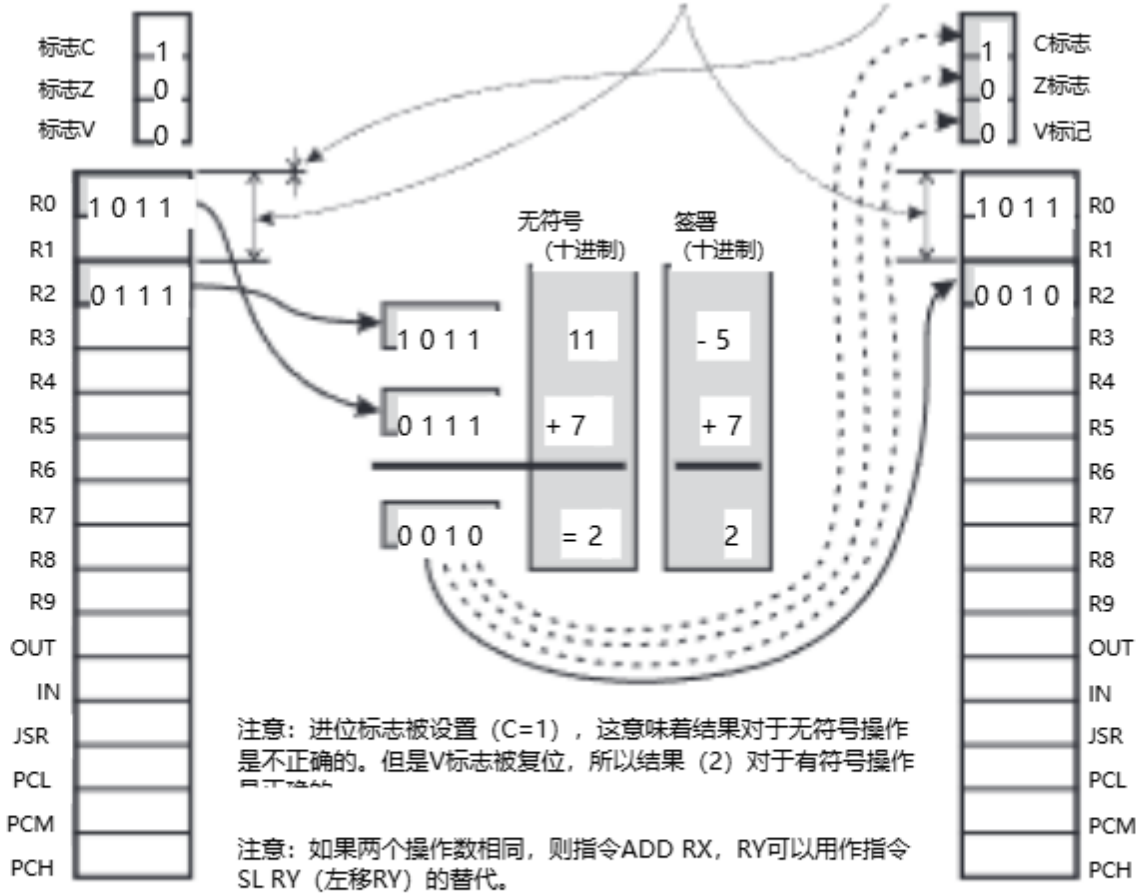
"0001"位是ADD RX、RY操作码。"0001"位选择操作数RX。"YYYY"位选择操作数RY

范例：

添加R2、R0

钻头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　　歌剧



注意：进位标志被设置（C=1），这意味着结果对于无符号操作是不正确的。但是V标志被复位，所以结果（2）对于有符号操作是正确的。

注意：如果两个操作数相同，则指令ADD RX，RY可以用作指令SL RY（左移RY）的替代。

术前　　术后

ADC RX,RY

Add with carry registers RX and RY

Syntax: {label} ADC RX, RY

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

Operation: (RX) ← (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:

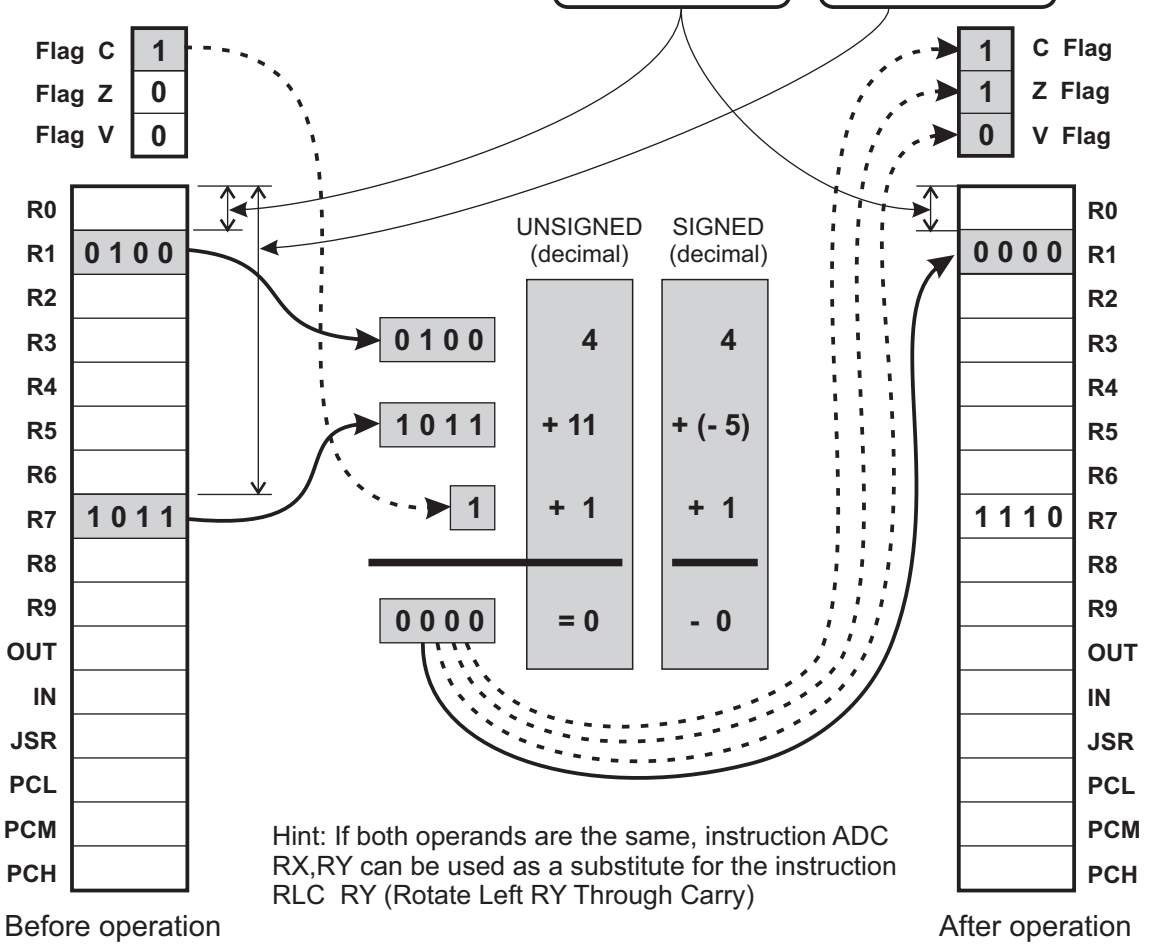
bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	0	X	X	X	X	Y	Y	Y	Y

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			



Before operation

After operation

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)

ADC RX、RY —与进位寄存器RX和RY相加

语法： {label} ADC RX、RY

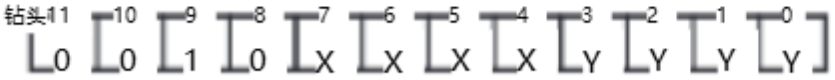
操作数： RX ∈ [R0. R15]
RY ∈ [R0. R15]

操作方式： (RX)← （RX）+（RY）+进位

产品描述： 将寄存器RY的内容加上进位标志的内容与寄存器RX的内容相加，并将结果放入寄存器RX。RX和RY必须使用寄存器直接寻址。

受影响的旗帜： 如果存在下溢（如果（RY）<（RX）），则重置C。否则，设置C。(note：Borrow is inverse C）.如果操作后结果=0000，则设置Z。否则，重置Z。对于有符号2的补码：如果有溢出，则设置

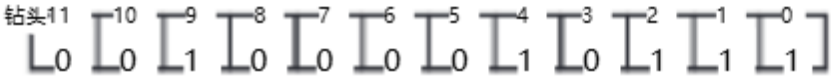
编码方式：

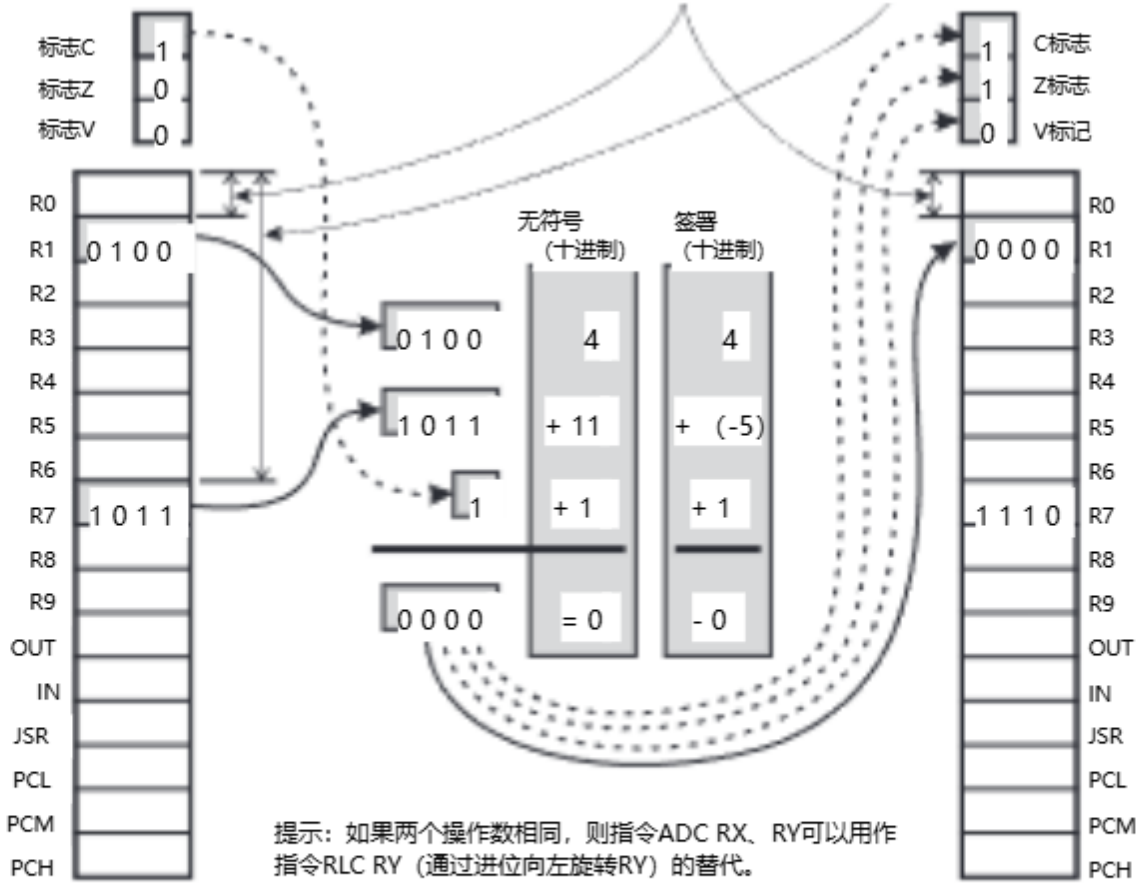


“0010”位是ADC RX、RY操作码。“0”位选择操作数RX。“YYYY”位选择操作数RY

范例：

ADC R1、R7





提示：如果两个操作数相同，则指令ADC RX、RY可以用作指令RLC RY（通过进位向左旋转RY）的替代。

SUB RX,RY

Subtract register RY from register RX

Syntax: {label} SUB RX, RY

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

Operation: (RX) ← (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

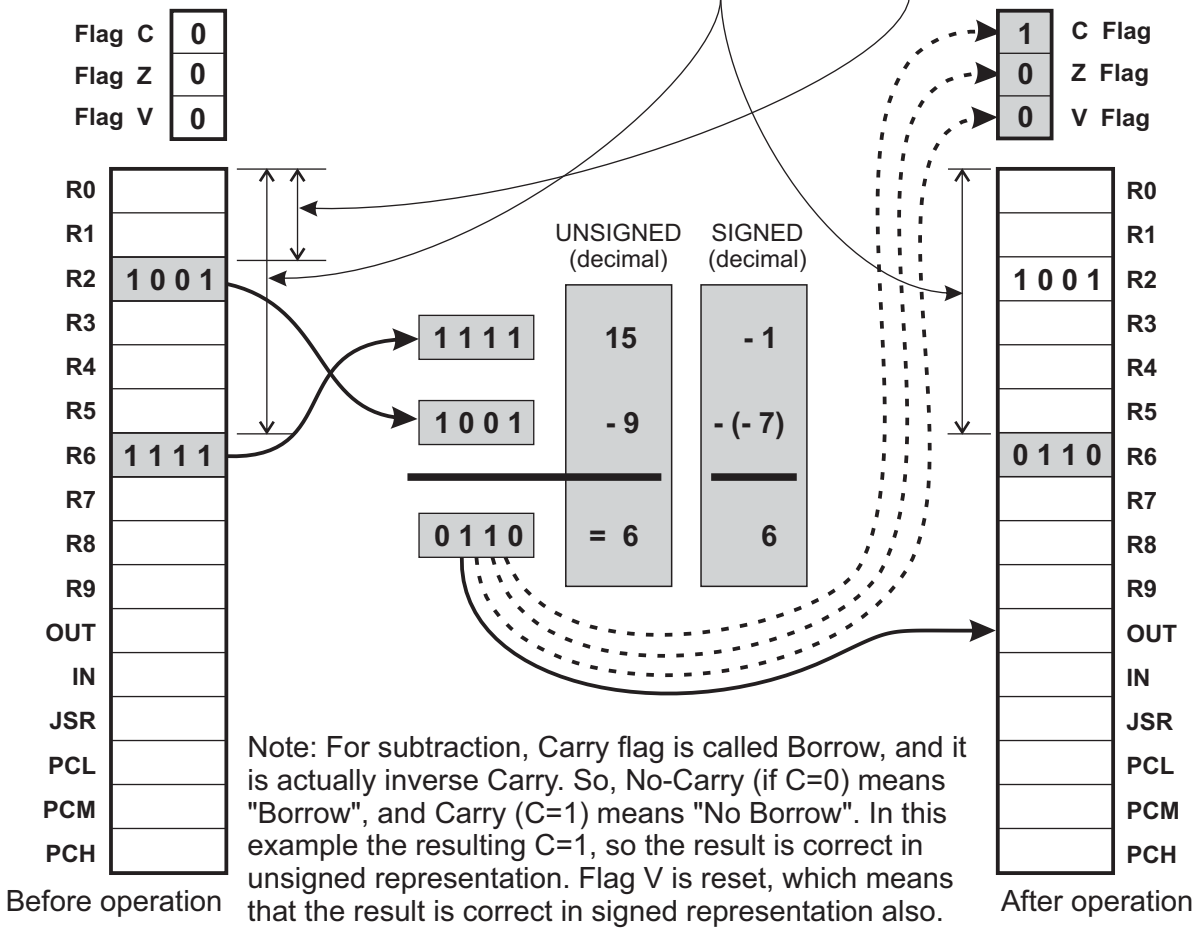
The "0011" bits are the SUB RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

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



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.

RYRX，RY

从寄存器RX中减去寄存器RY

语法：{label}　　RX，　RY

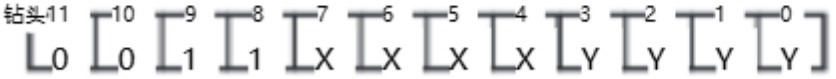
操作数：RX ∈ [R0. R15]
RY ∈ [R0. R15]

操作方式：(RX)← （RX）-（RY）

产品描述：从寄存器RX的内容中减去寄存器RY的内容，并将结果放入寄存器RX中。
RX和RY必须使用寄存器直接寻址。

受影响的旗帜：如果存在下溢（如果（RY）<（RX）），则重置C。否则，设置C。(note：Borrow is inverse C）.如果操作后结果=0000，则设置Z。否则，重置Z。对于有符号2的补码：如果有溢出，则设置

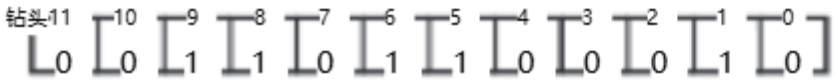
编码方式：

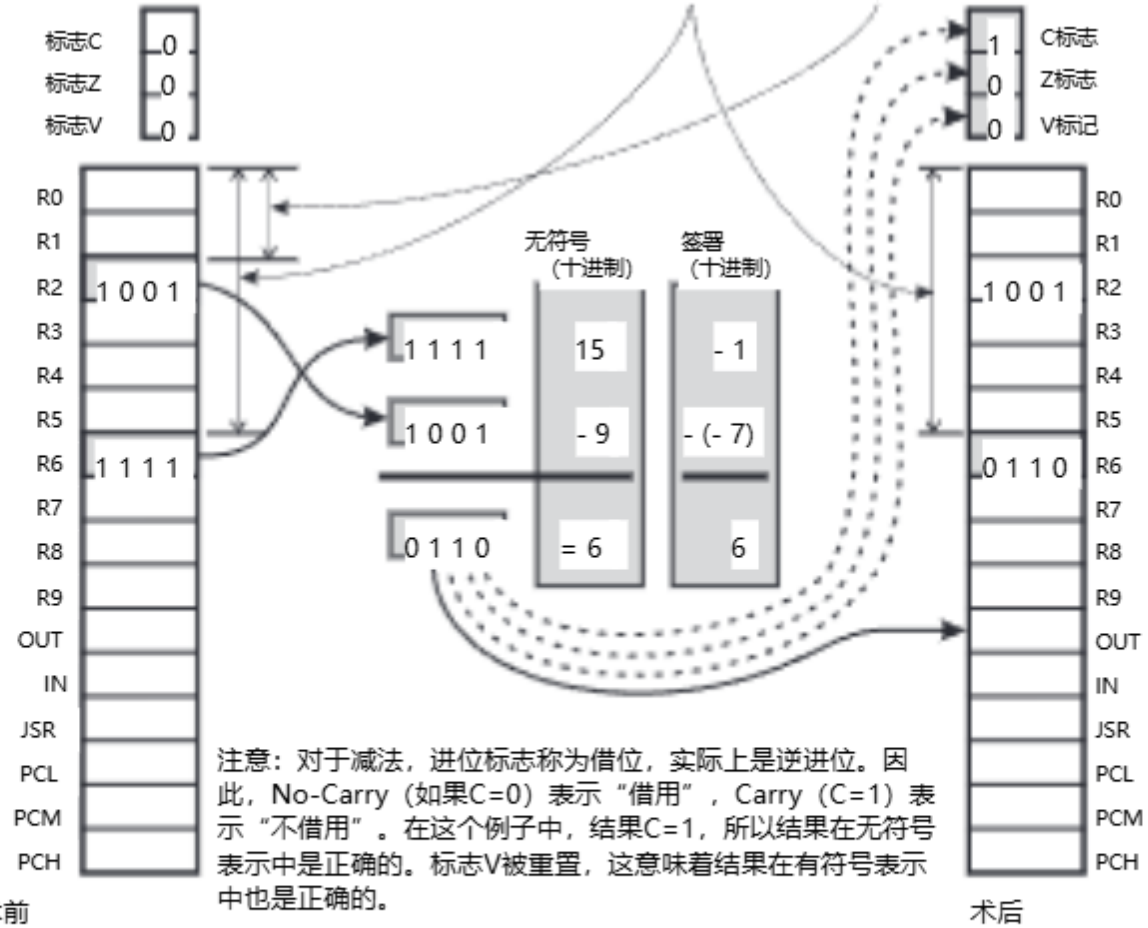


“0011”位是RX、RY操作码。“RX”位选择操作数RX。“YYYY”位选择操作数RY

例一：

R6，R2



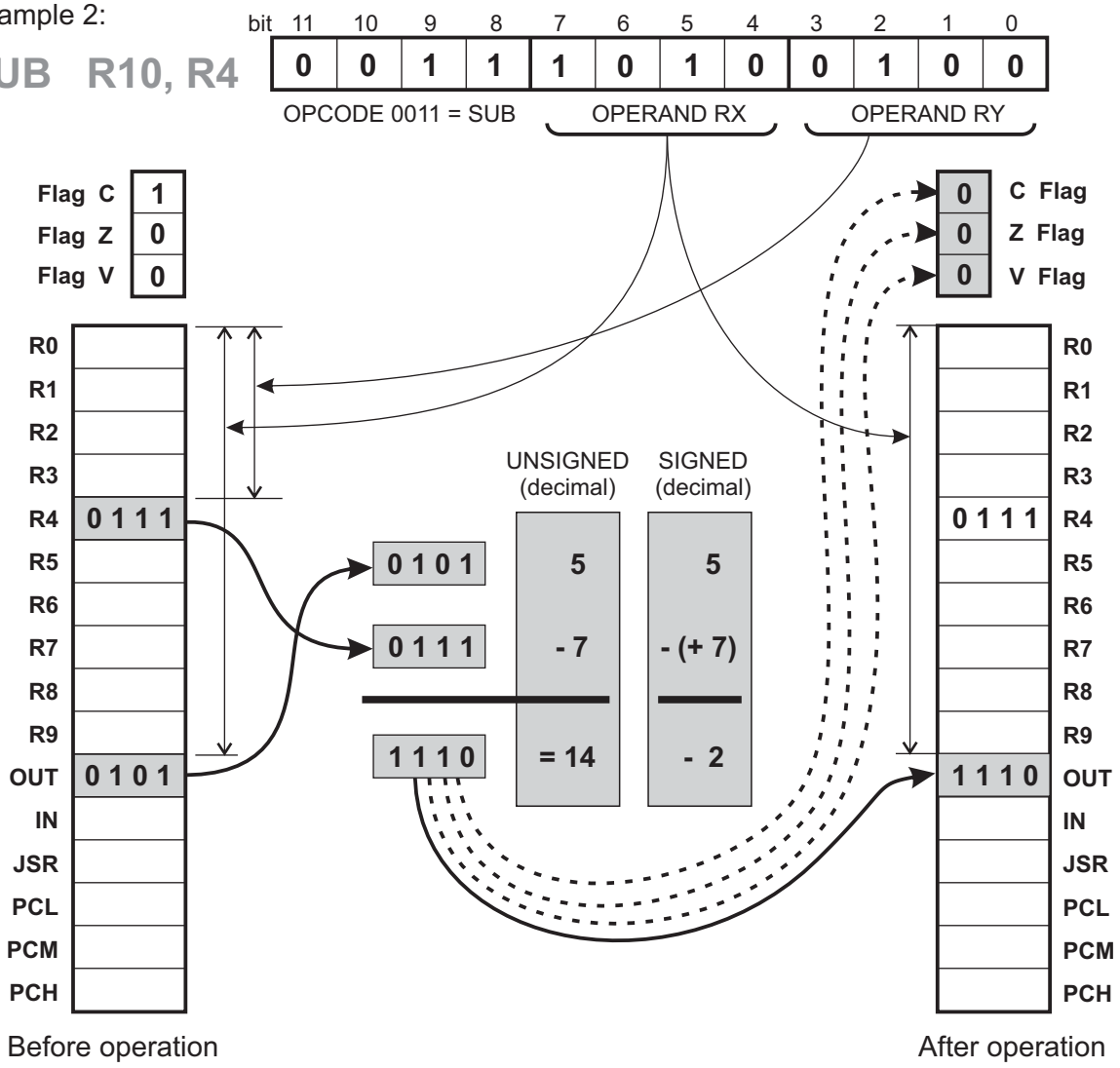


注意：对于减法，进位标志称为借位，实际上是逆进位。因此，No-Carry（如果C=0）表示“借用”，Carry（C=1）表示“不借用”。在这个例子中，结果C=1，所以结果在无符号表示中是正确的。标志V被重置，这意味着结果在有符号表示中也是正确的。

术前　　术后

Example 2:

SUB R10, R4

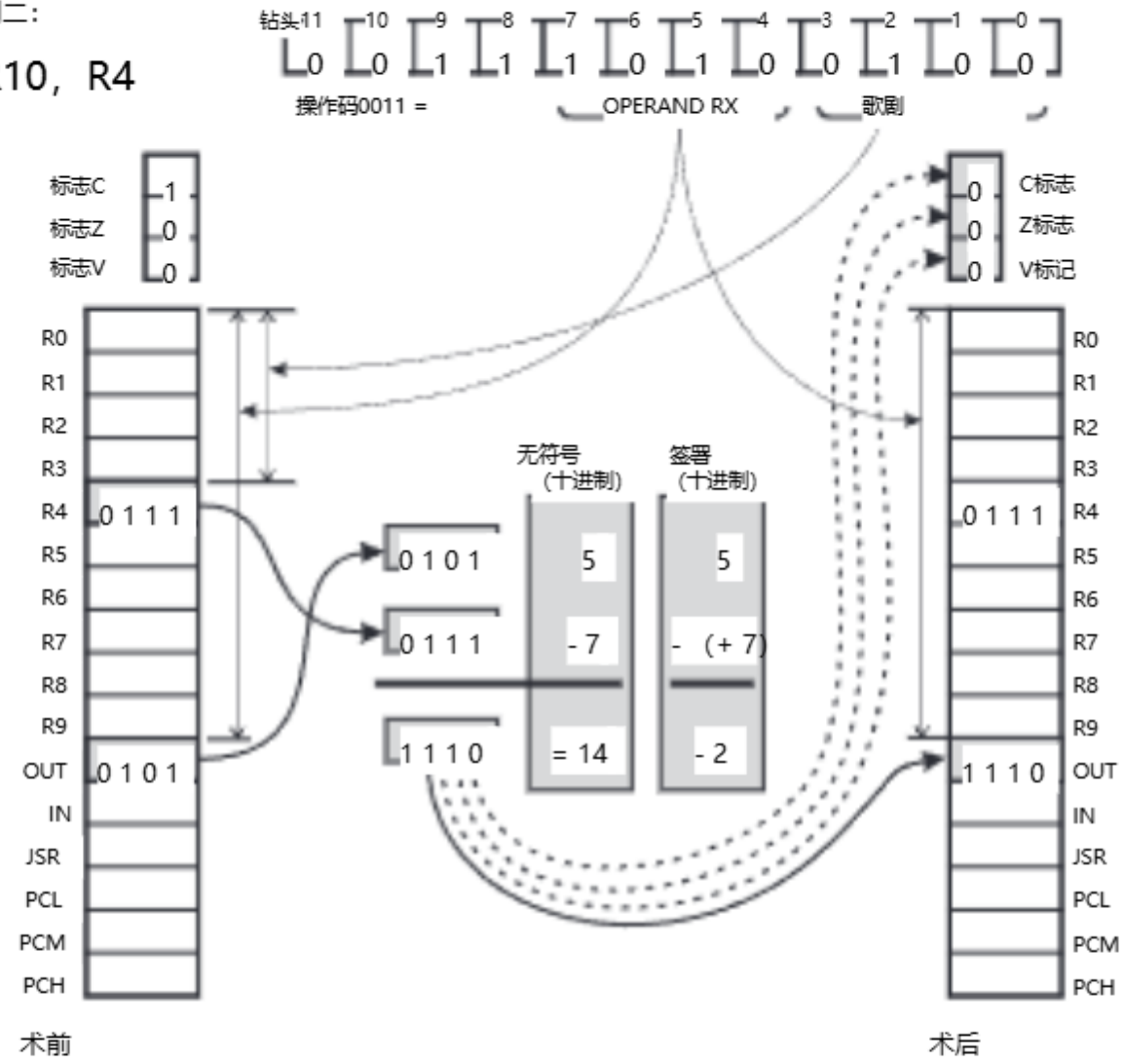


Note: For subtraction, Carry flag is called Borrow, and it is actually inverse Carry. So, No-Carry (if C=0) means "Borrow", and Carry (C=1) means "No Borrow". In this example C=0, so there is Underflow condition, which means that the result is not correct in unsigned representation. In signed representation,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

RYRX，RY　　从寄存器RX中减去寄存器RY（续）

例二：

R10，R4



注意：对于减法，进位标志称为借位，实际上是逆进位。因此，No-Carry（如果C=0）表示“借用”，Carry（C=1）表示“不借用”。在这个例子中，C=0，所以存在下溢条件，这意味着结果在无符号表示中不正确。在有符号表示中，结果是1110，即-2。标志V未被设置，这意味着-2是正确的结果。

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) - ($\overline{C}$)

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

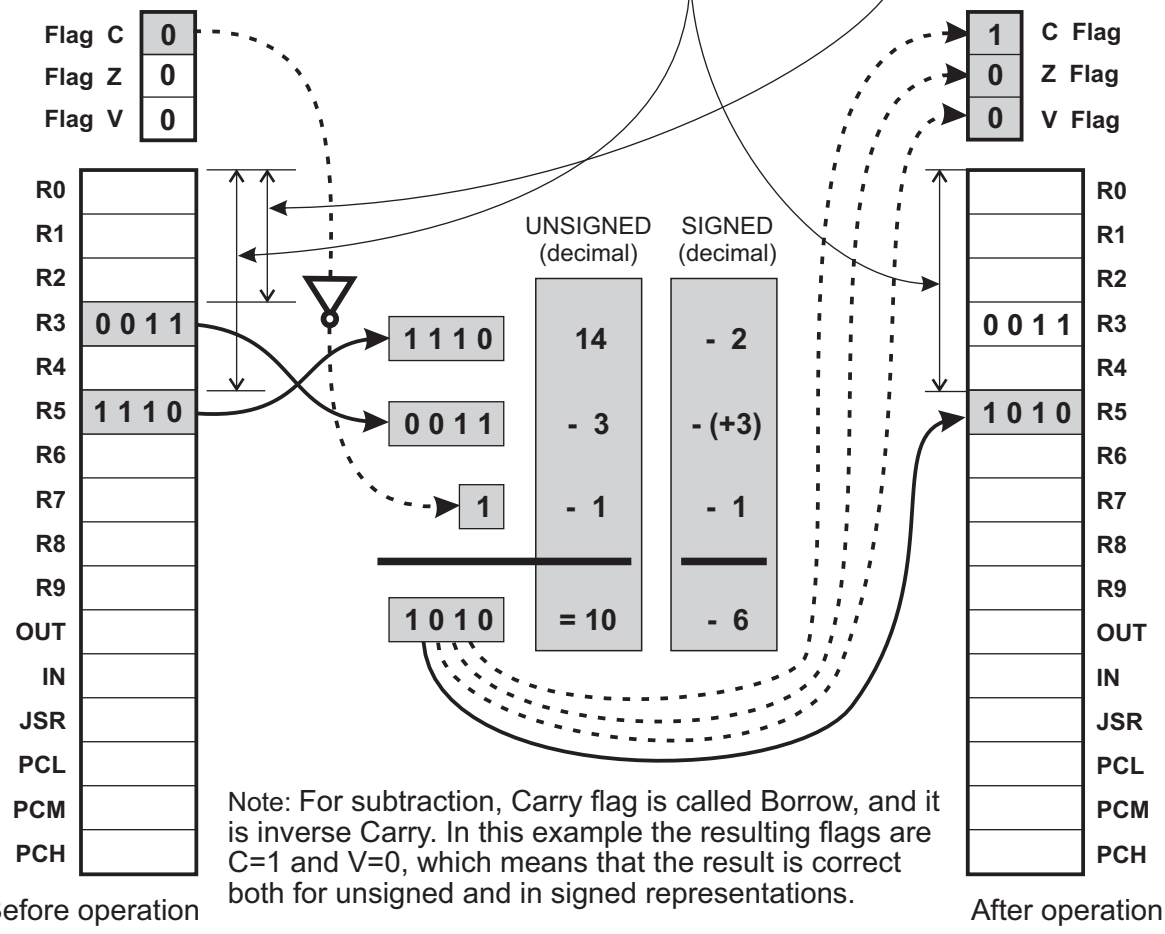
The "0100" bits are the SBB RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

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



Before operation

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.

After operation

SBB RX，RY —从寄存器RX中减去寄存器RY并借位

语法： {label} SBB RX, RY

操作数： RX ∈ [R0. R15]
RY ∈ [R0. R15]

操作方式： (RX)← （RX）-（RY）-（C）

产品描述： 从寄存器X的内容中减去寄存器Y的内容，并将结果放入寄存器X中。
X和Y必须使用寄存器直接寻址。

受影响的旗帜： 如果存在下溢（如果（RY）<（RX）），则重置C。否则，设置C。(note：Borrow is inverse C）.如果操作后结果=0000，则设置Z。否则，重置Z。对于有符号2的补码：如果有溢出，则设置

编码方式：

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

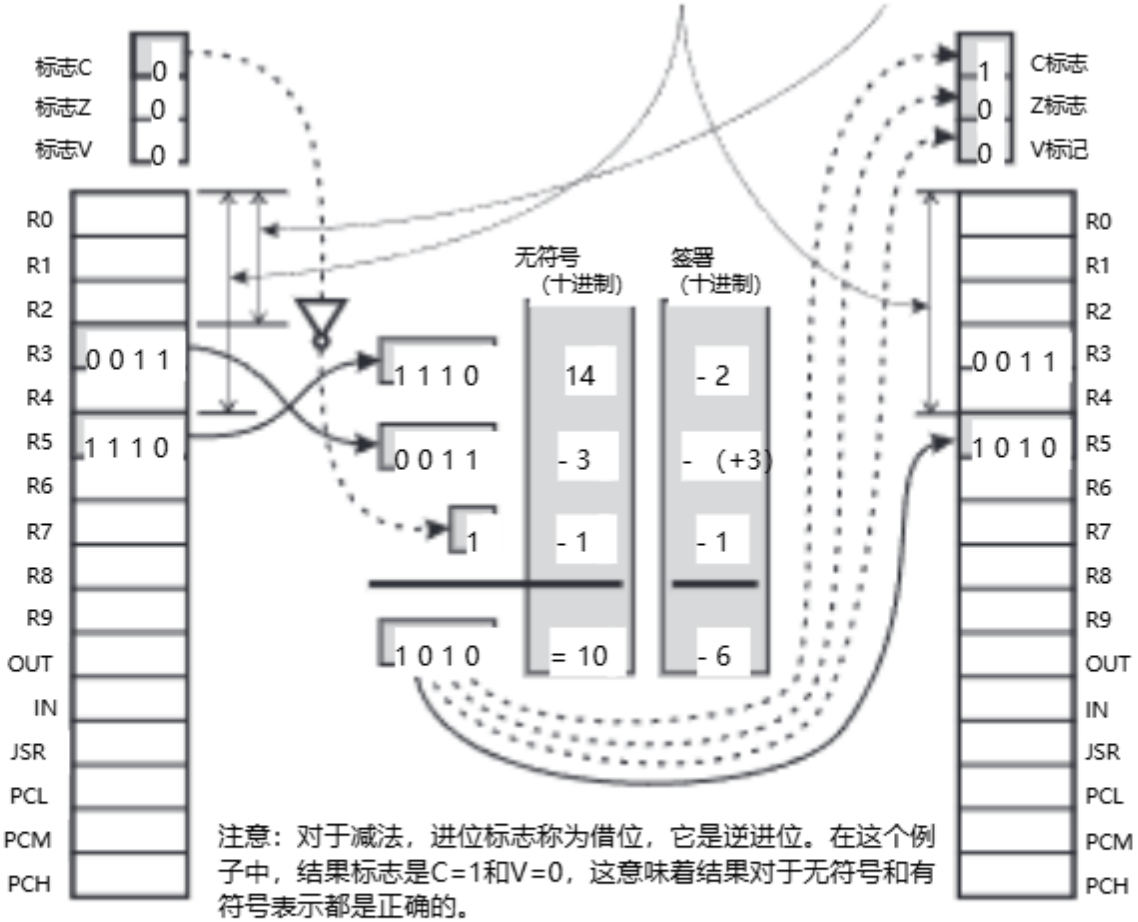
“0100”位是SBB RX、RY操作码。“0”位选择操作数RX。“YYYY”位选择操作数RY

例一：

SBB R5、R3

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

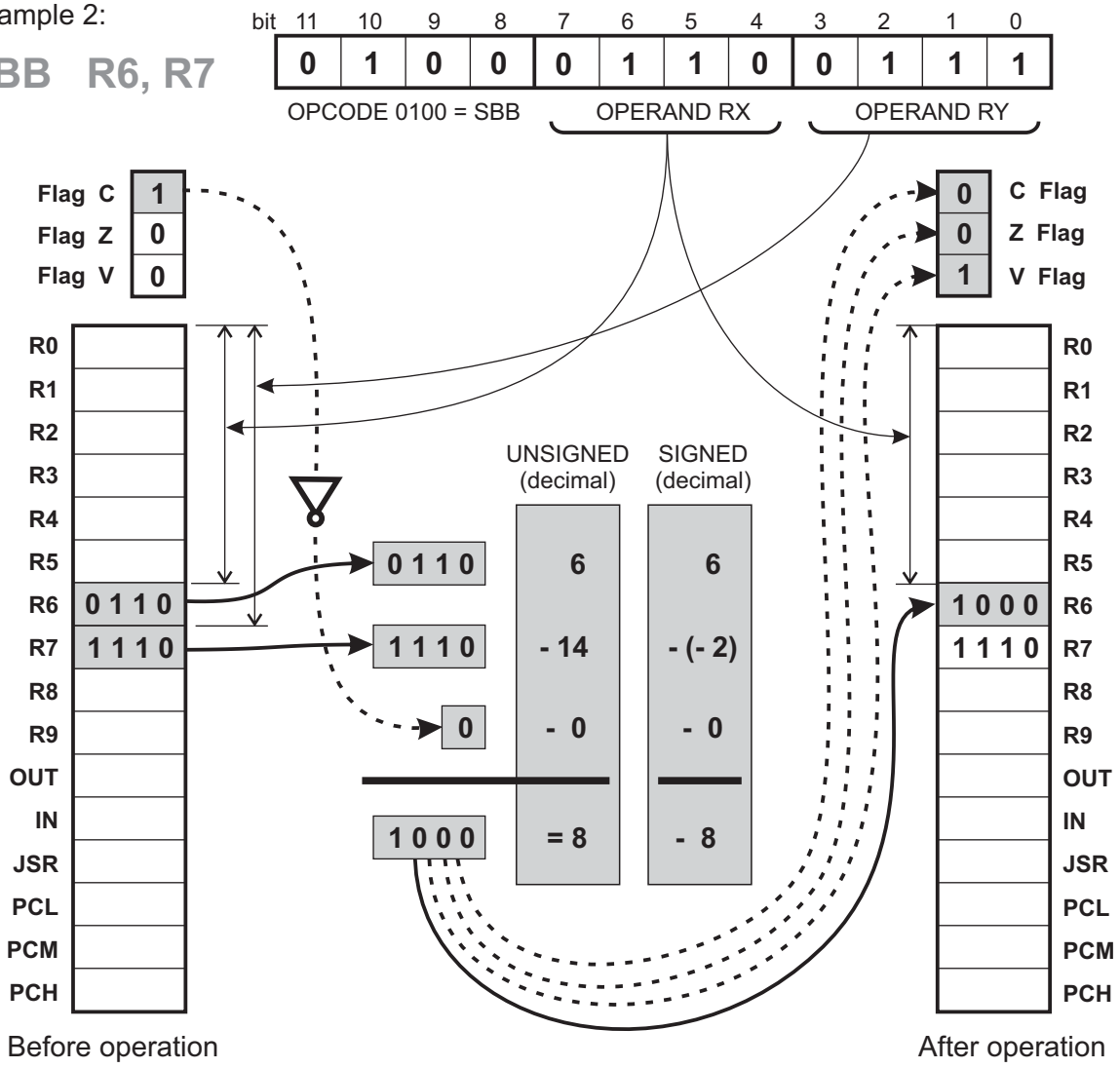
OPCODE 0100 = SBB　OPERAND RX　歌剧



注意：对于减法，进位标志称为借位，它是逆进位。在这个例子中，结果标志是C=1和V=0，这意味着结果对于无符号和有符号表示都是正确的。

Example 2:

SBB R6, R7

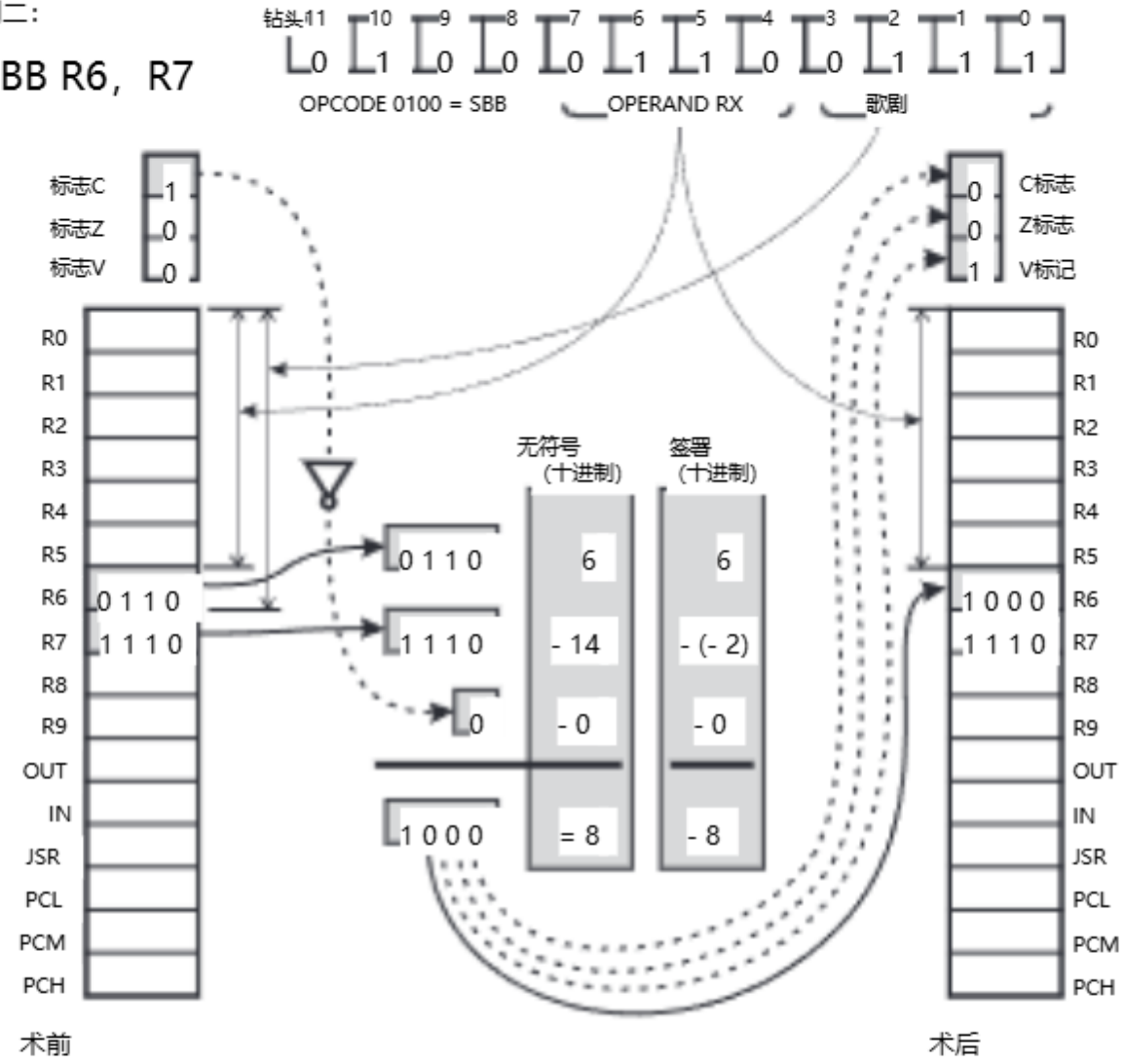


Note: For subtraction, Carry flag is called Borrow, and it is actually inverse Carry. So, No-Carry (C=0) means "Borrow", and Carry (C=1) means "No Borrow". In this example the resulting flag C=0, so there is Underflow condition, which means that the result is not correct in unsigned representation. The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

SBB RX，RY 从寄存器RX中减去寄存器RY并借位（续）

例二：

SBB R6，R7



注意：对于减法，进位标志称为借位，实际上是逆进位。因此，No-Carry（C=0）表示“借用”，Carry（C=1）表示“不借用”。在这个例子中，结果标志C=0，因此存在下溢条件，这意味着结果在无符号表示中不正确。溢出标志被设置（V=1），这意味着即使在有符号表示中结果也不正确。

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

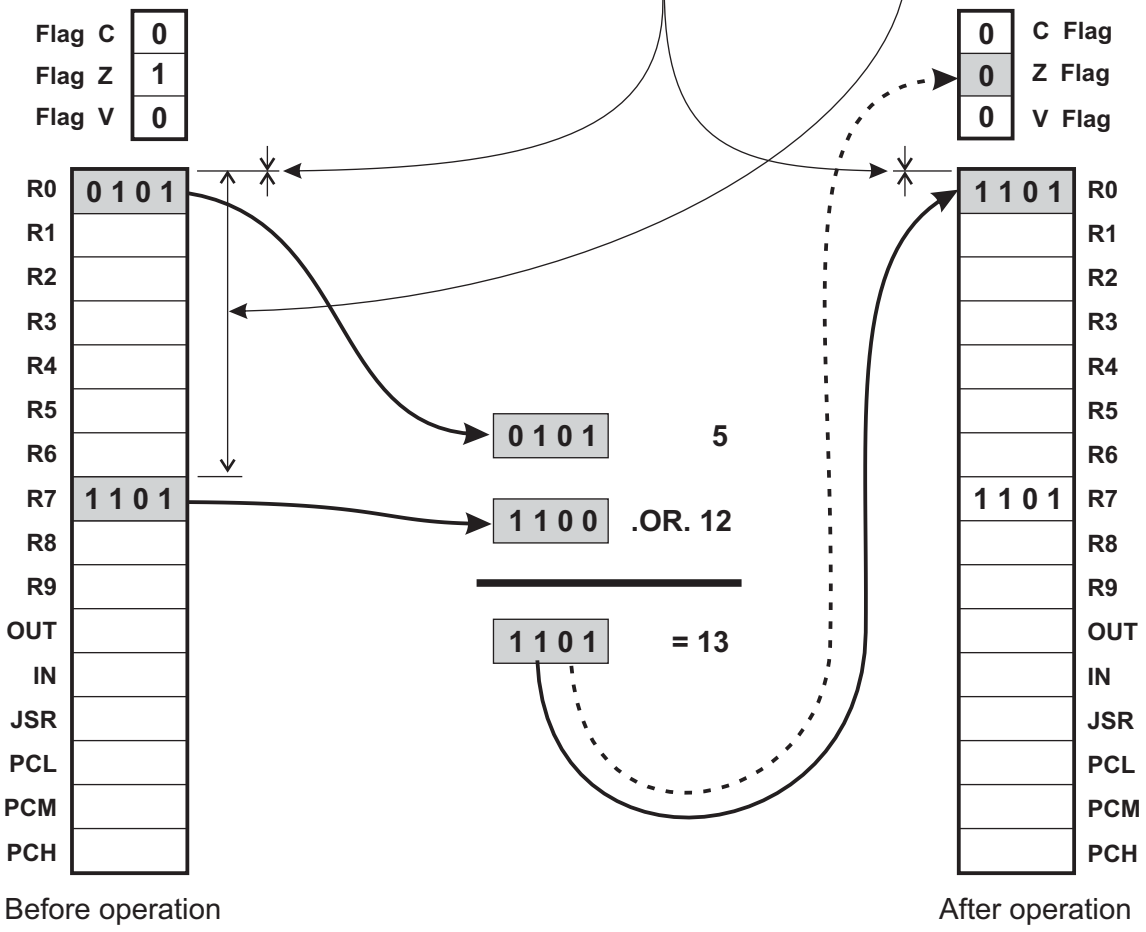
The "0101" bits are the OR RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

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



或RX、RY

包含或寄存器RX和RY

语法： {label}或 RX, RY

操作数： RX ∈ [R0. R15]
RY ∈ [R0. R15]

操作方式： (RX)← （RX）. OR.（RY）

产品描述： 计算4位寄存器RX与寄存器RY的逻辑“或”运算，并将结果放回寄存器RX。RX和RY必须使用寄存器直接寻址。

受影响的旗帜： 标志C不受影响如果操作后结果=0000，则设置Z。否则，重置Z

编码方式：

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

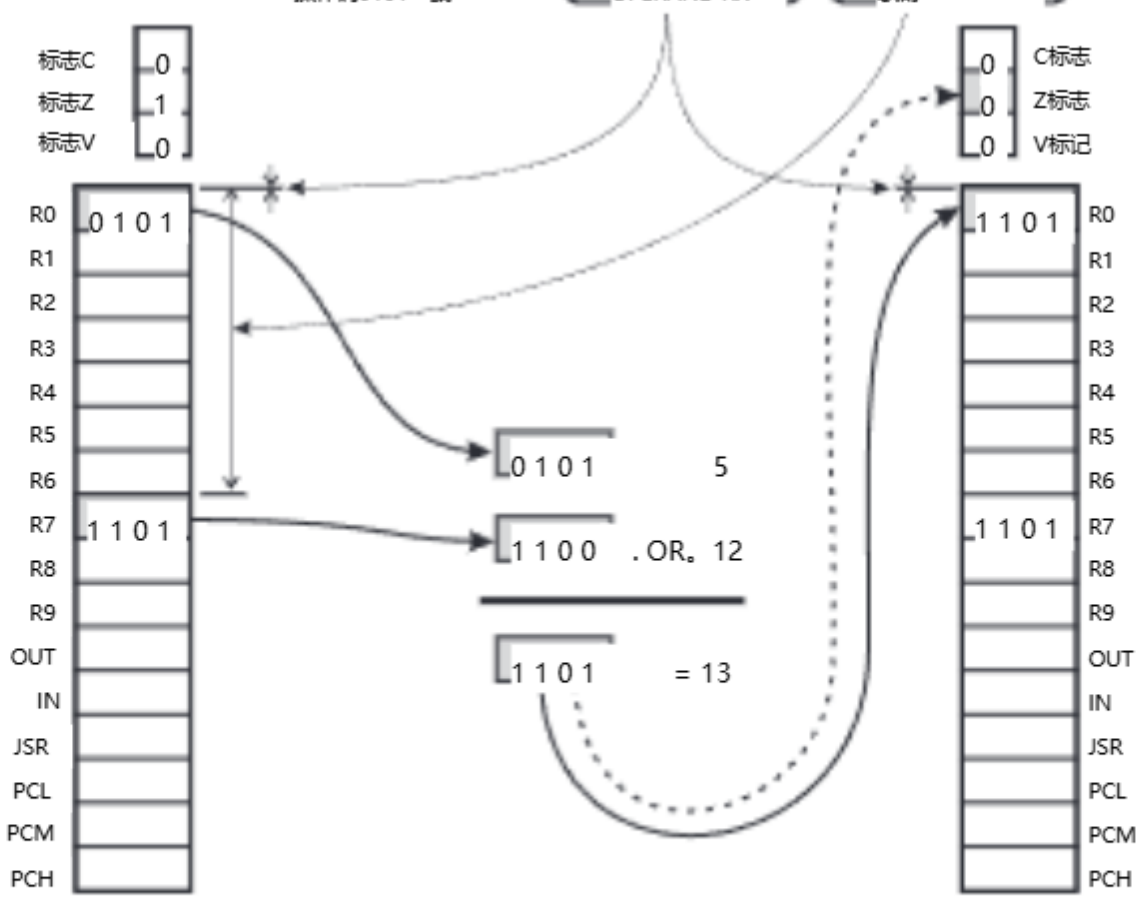
“0101”位是OR RX、RY操作码。“0101”位选择操作数RX。“YYYY”位选择操作数RY

范例：

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

操作码0101 =或 OPERAND RX 歌剧

或R0、R7



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:

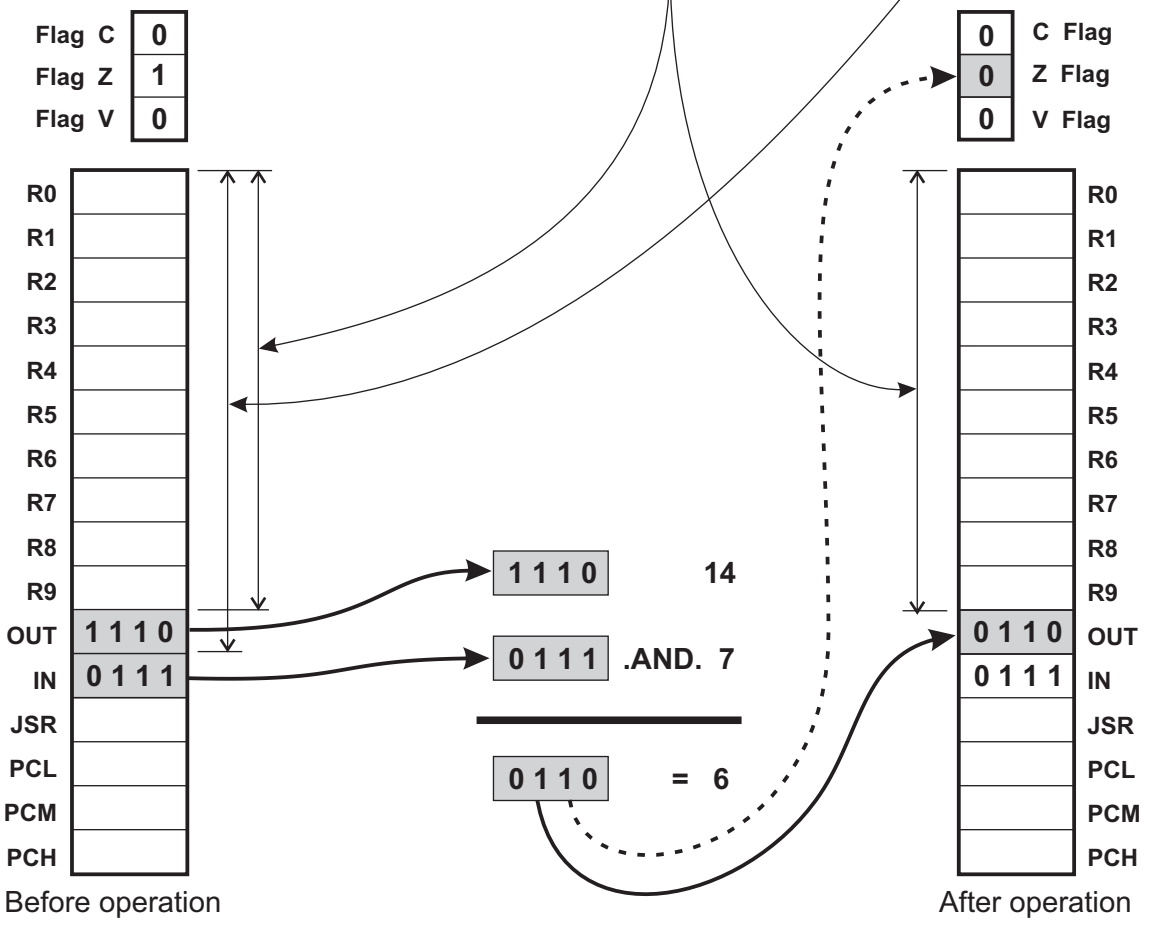
bit	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	0	X	X	X	X	Y	Y	Y	Y

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			



和RX，RY

逻辑与寄存器RX和RY

语法：{label} AND RX, RY

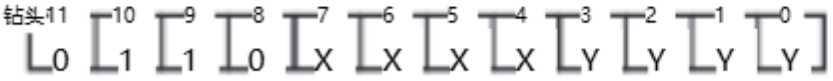
操作数：RX ∈ [R0. R15]
RY ∈ [R0. R15]

操作方式：(RX)← （RX）. AND。（RY）

产品描述：计算4位寄存器RX与寄存器RY的逻辑与运算，并将结果放回寄存器RX。RX和RY必须使用寄存器直接寻址。

受影响的旗帜：标志C不受影响如果操作后结果=0000，则设置Z。否则，重置Z

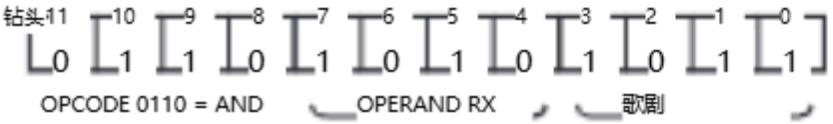
编码方式：

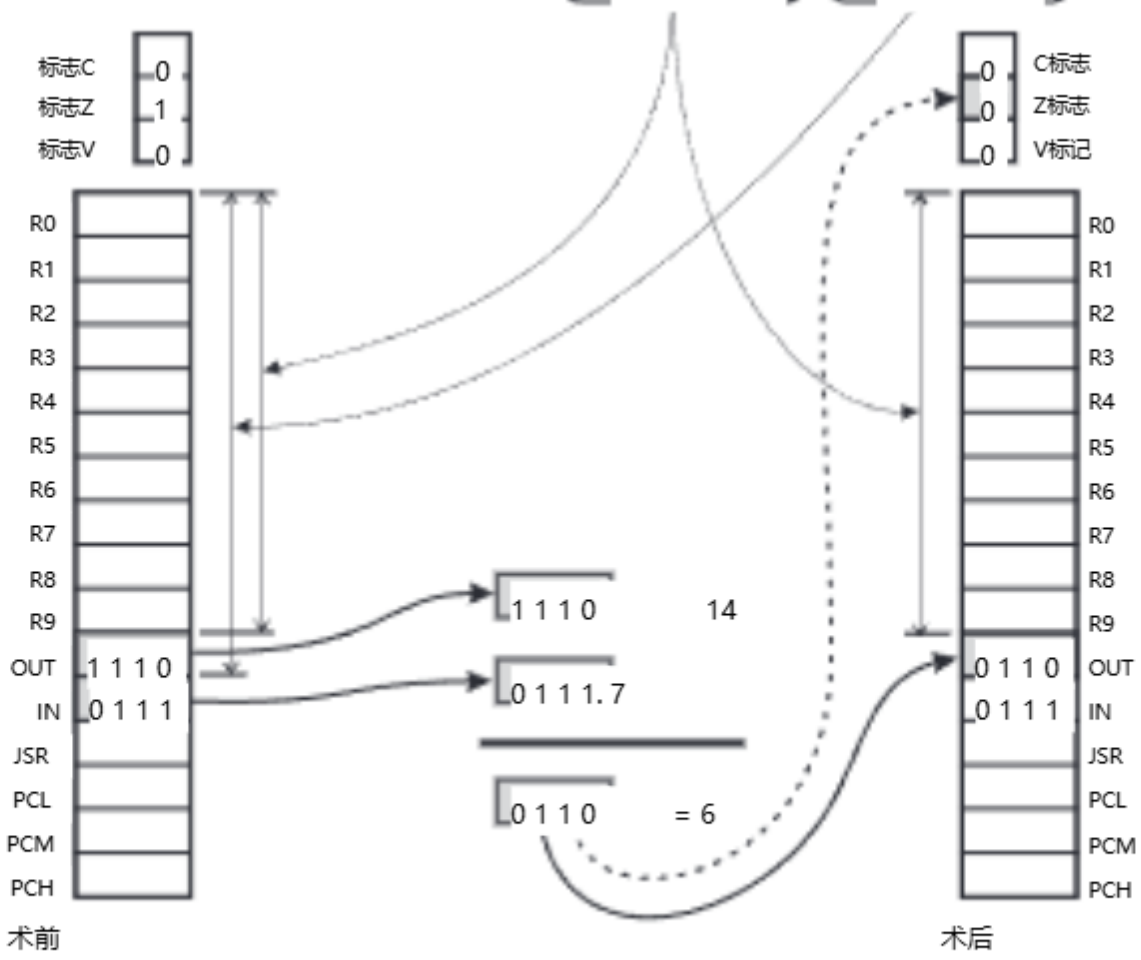


“0110”位是AND RX、RY操作码。“0”位选择操作数RX。“YYYY”位选择操作数RY

范例：

和R10、R11





XOR RX,RY

Exclusive OR registers RX and RY

Syntax: {label} XOR RX, RY

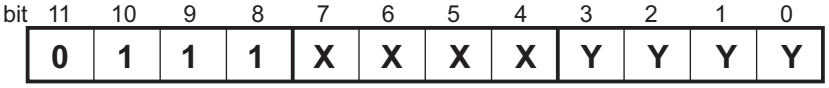
Operands: RX ∈ [R0...R15]
RY ∈ [R0...R15]

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

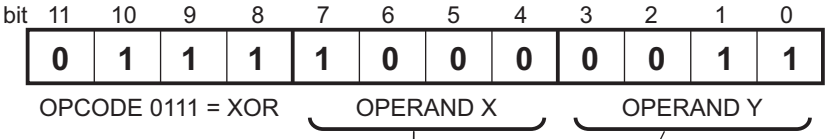
Encoding:

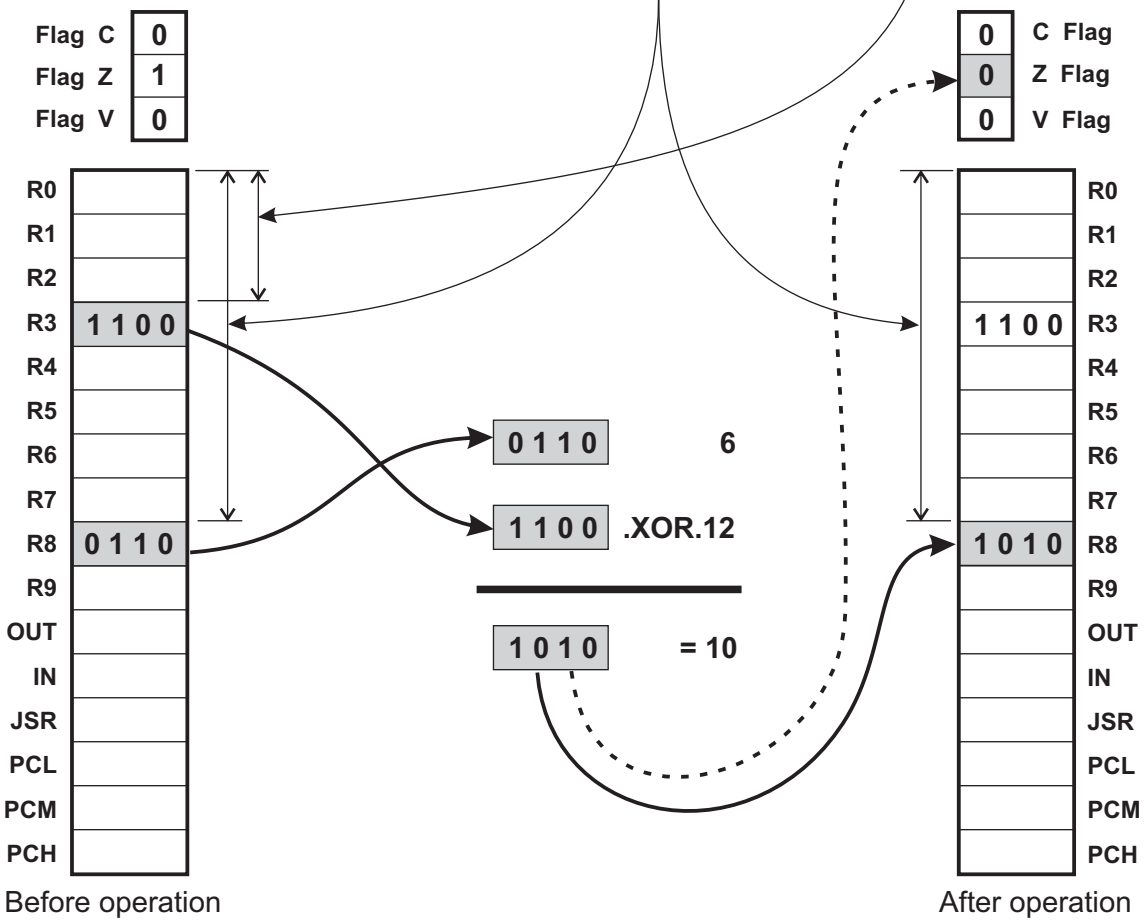


The "0111" bits are the XOR RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

Example:

XOR R8, R3





XOR RX，RY —异或寄存器RX和RY

语法：{label}异或 RX， RY

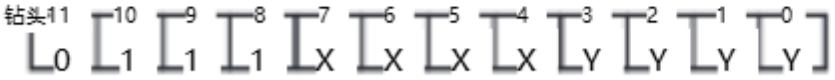
操作数：RX ∈ [R0. R15]
RY ∈ [R0. R15]

操作方式：(RX)← （RX）. XOR。（RY）

产品描述：计算4位寄存器RX与寄存器RY的逻辑异或运算，并将结果放回寄存器RX。RX和RY必须使用寄存器直接寻址。

受影响的旗帜：标志C不受影响如果操作后结果=0000，则设置Z。否则，重置Z

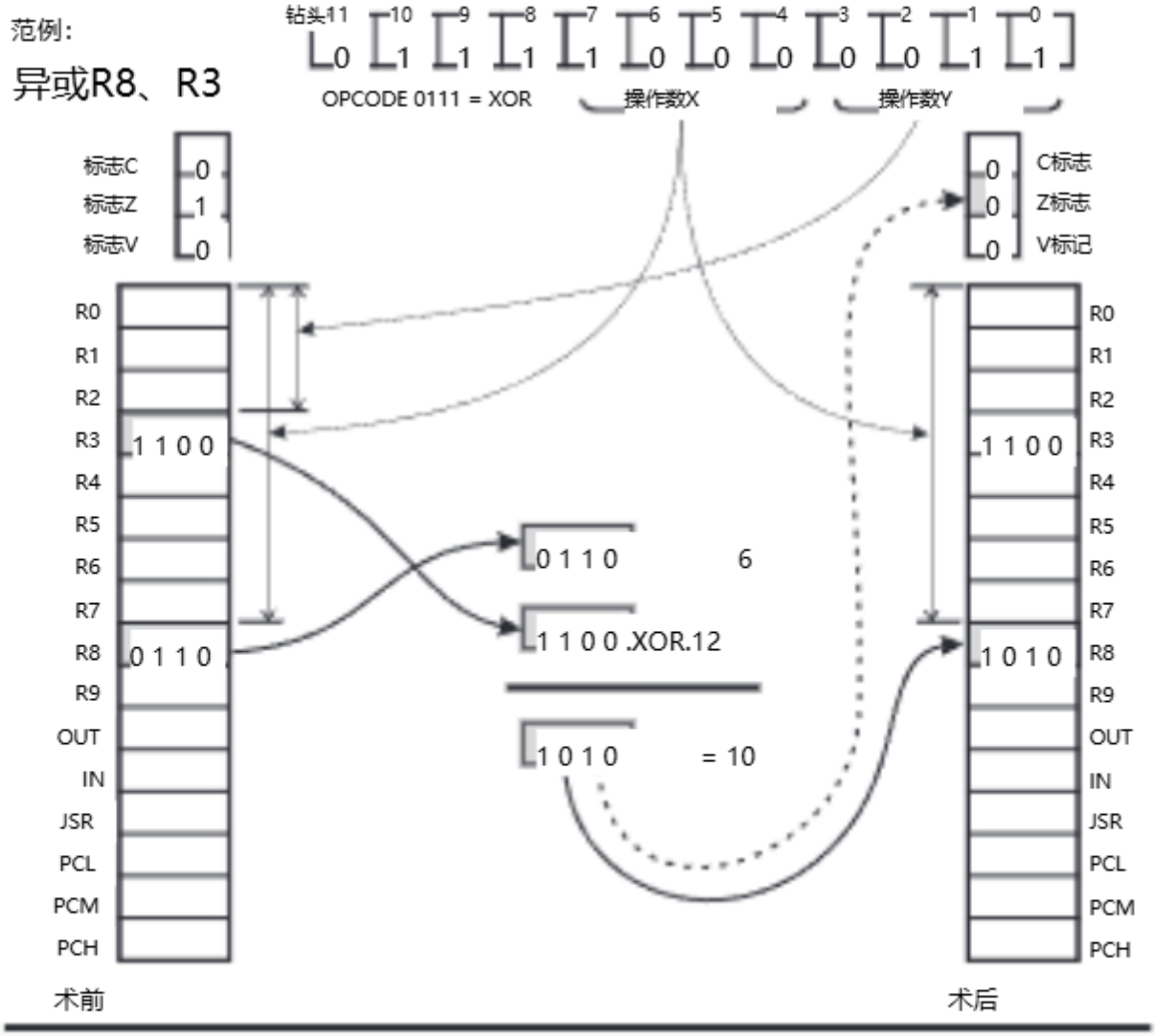
编码方式：



“0111”位是XOR RX、RY操作码。“0”位选择操作数RX。“YYYY”位选择操作数RY

范例：

异或R8、R3



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

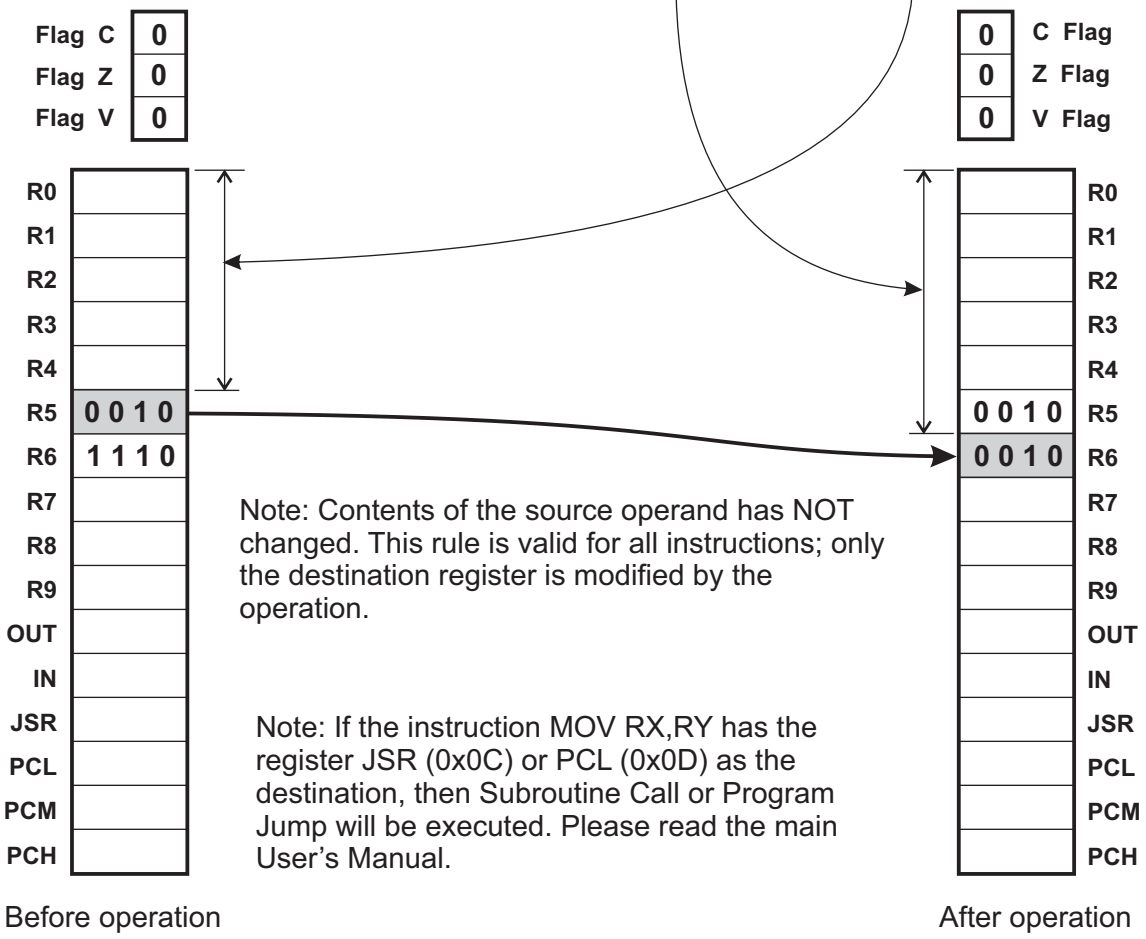
The "1000" bits are the MOV RX,RY opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

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，RY —将内容从寄存器RY移动到寄存器RX

语法： {label} MOV RX, RY

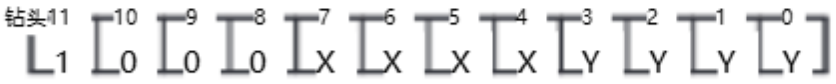
操作数： RX ∈ [R0. R15]
RY ∈ [R0. R15]

操作方式： (RX)← (RY)

产品描述： 将4位内容从寄存器RY移至寄存器RX。RX和RY必须使用寄存器直接寻址。

受影响的旗帜： 一个也没有。

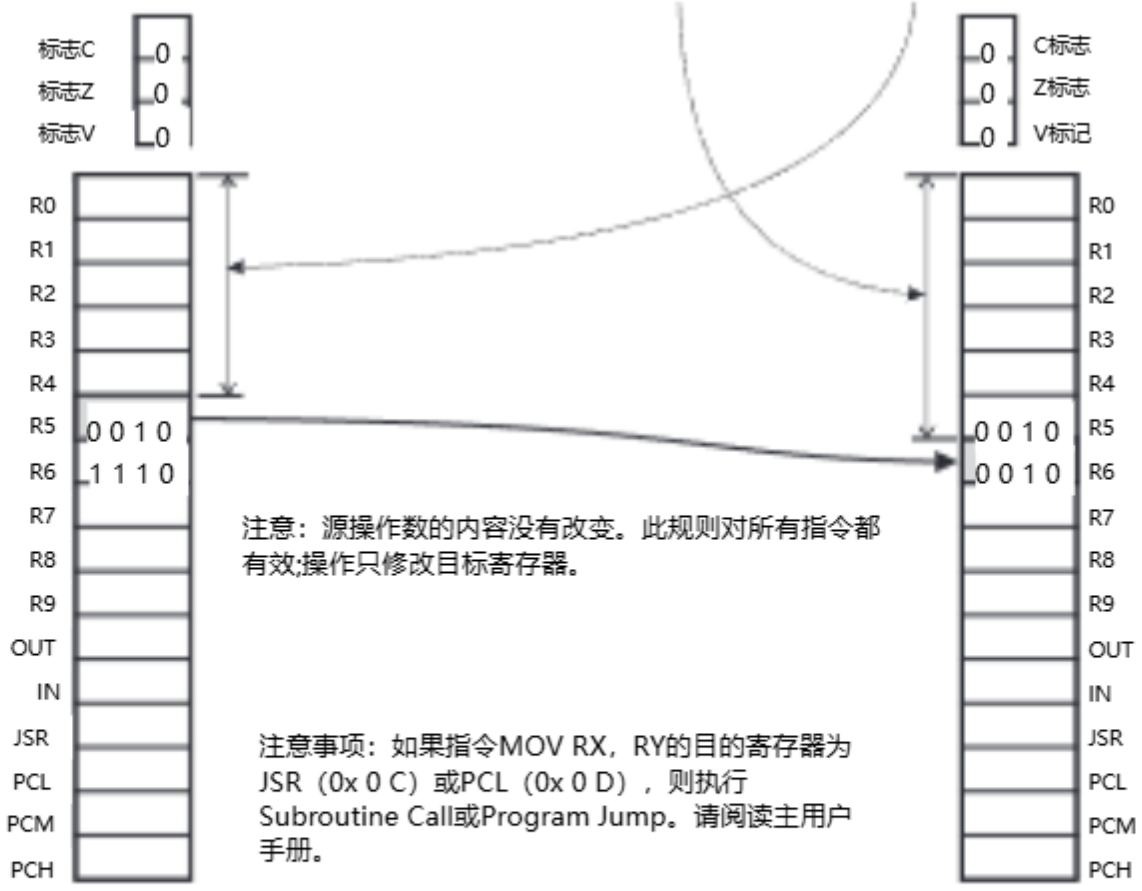
编码方式：



“1000”位是MOV RX、RY操作码。“0”位选择操作数RX。“YYYY”位选择操作数RY

范例：

MOV R6，R5



注意：源操作数的内容没有改变。此规则对所有指令都有效;操作只修改目标寄存器。

注意事项：如果指令MOV RX，RY的目的寄存器为JSR（0x 0 C）或PCL（0x 0 D），则执行Subroutine Call或Program Jump。请阅读主用户手册。

术前 术后

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

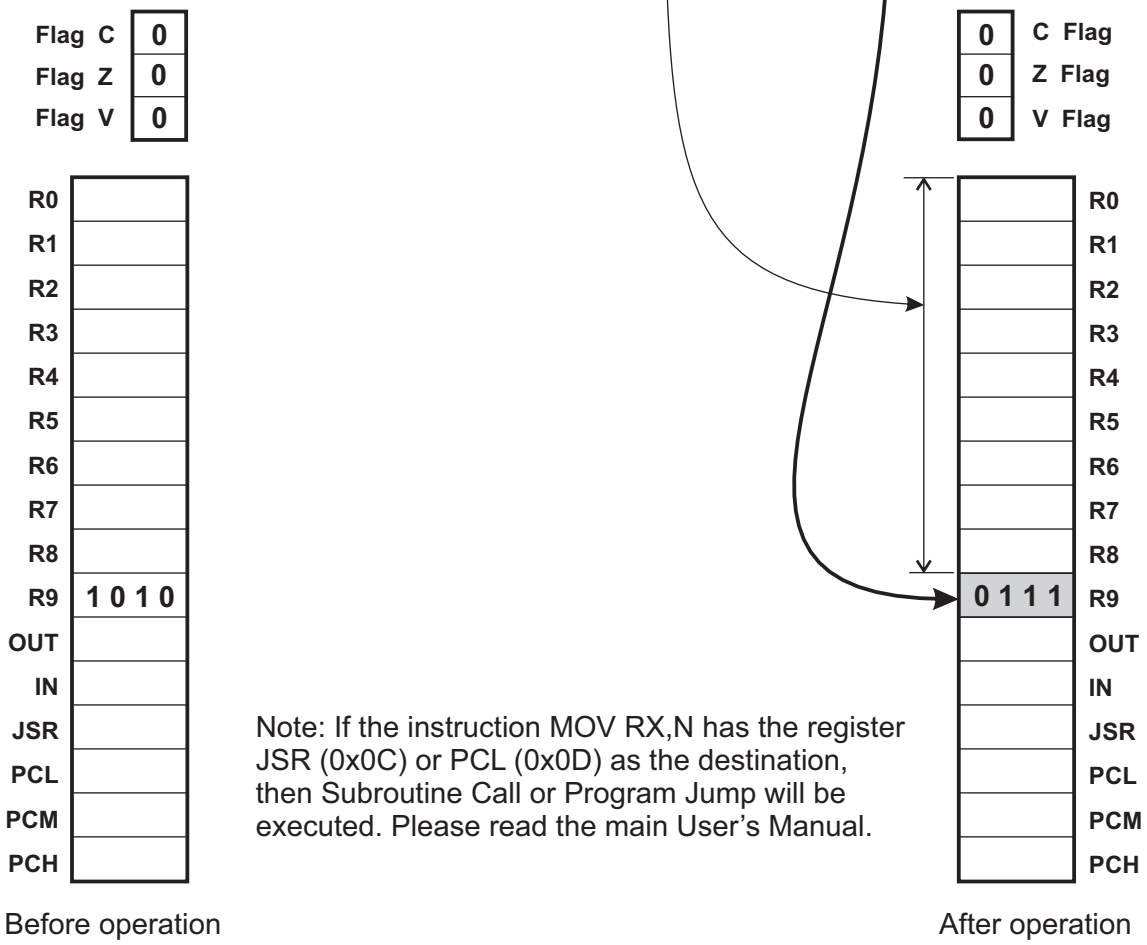
The "1001" bits are the MOV RX,N opcode
The "XXXX" bits select the operand RX
The "NNNN" bits are the literal N

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 RX，N ——将4位文字N移至寄存器RX

语法： {label} MOV RX，N

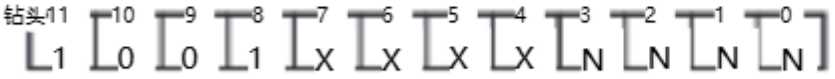
操作数： RX ∈ [R0. R15]
N ∈ 0. 15

操作方式： (RX)← N

产品描述： 将4位文字移至寄存器RX。RX必须使用寄存器直接寻址

受影响的旗帜： 一个也没有。

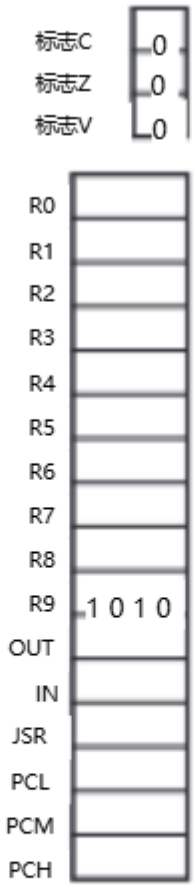
编码方式：



“1001”位是MOV RX，N操作码。“1001”位选择操作数RX。“NNNN”位是文字N。

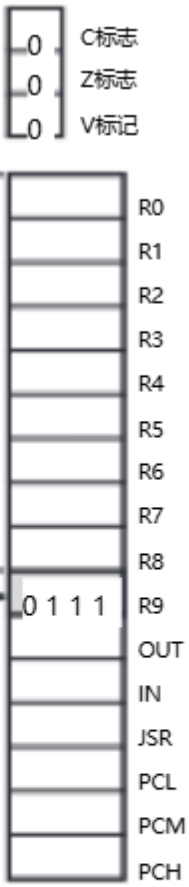
范例：

MOV R9，7



术前

注：如果指令MOV RX，N的目的寄存器为JSR（0x0C）或PCL（0x0D），则将执行Subroutine Call或Program Jump。请阅读主用户手册。



术后

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

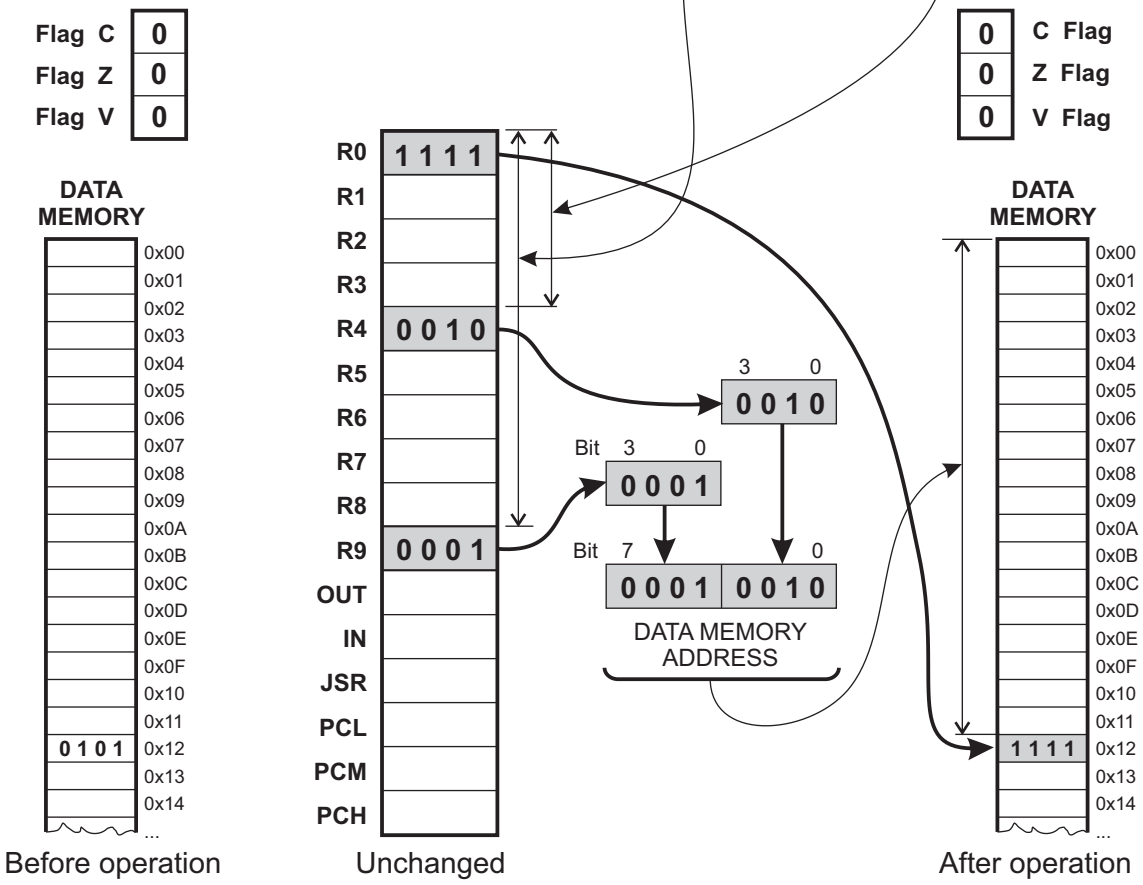
The "1010" bits are the MOV [XY],R0 opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

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



MOV [XY,] R0

将寄存器R0的内容移至由寄存器RX（高半字节）和RY（低半字节）间接寻址的数据存储器

语法：{label} MOV [XY], R0

操作数：RX ∈ [R0. R15]
RY ∈ [R0. R15]

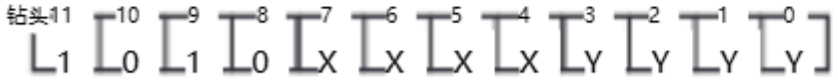
操作方式：((RX)：（RY）) ← （R0）

产品描述：将寄存器R0的4位内容移动到由寄存器RX（高半字节）和RY（低半字节）间接寻址的数据存储器。

RX和RY必须使用寄存器直接寻址。

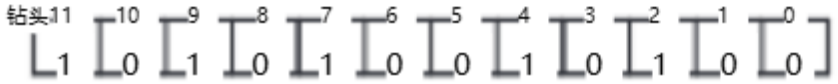
受影响的旗帜：一个也没有。

编码方式：

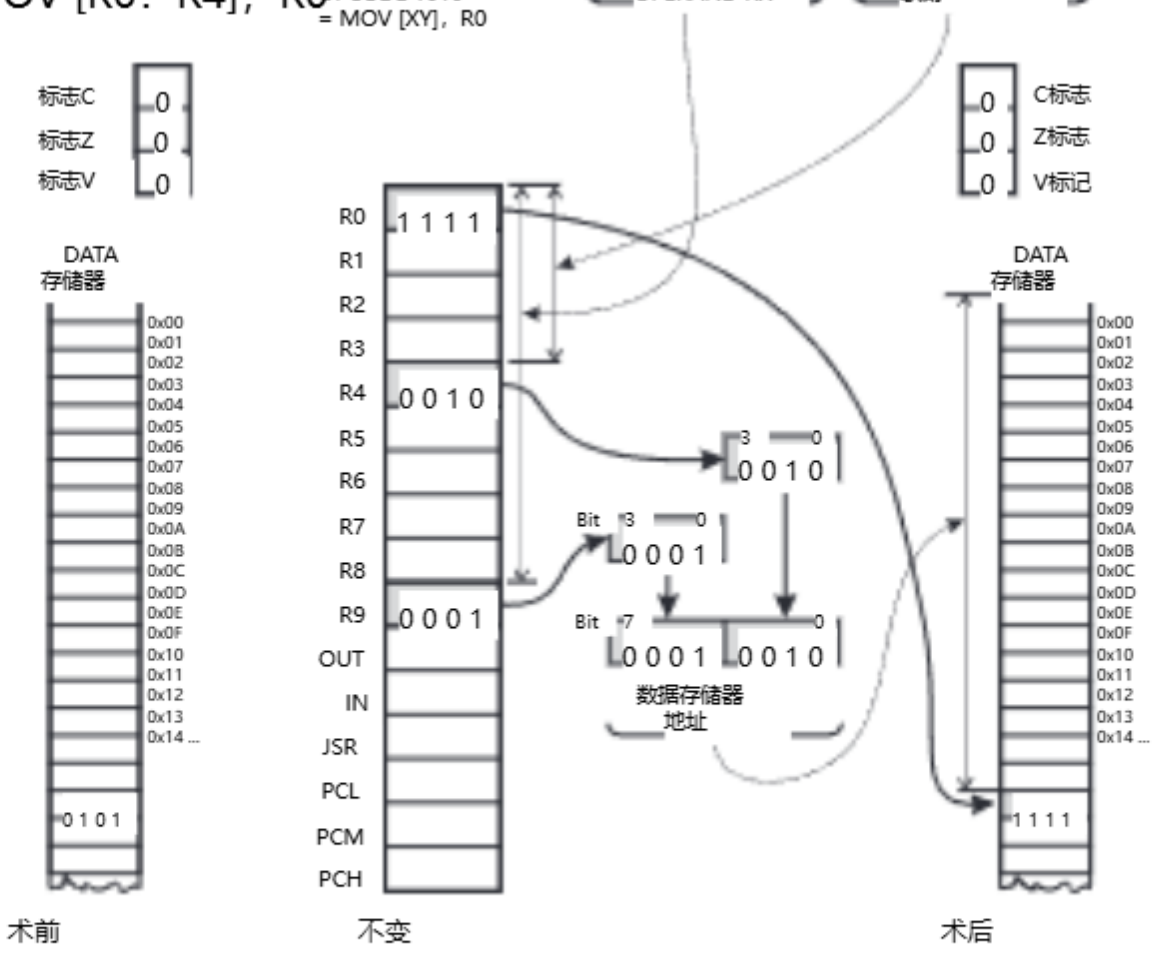


“1010”位是MOV [XY]、R0操作码。“1010”位选择操作数RX。“YYYY”位选择操作数RY

范例：



MOV [R6：R4]，R0



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

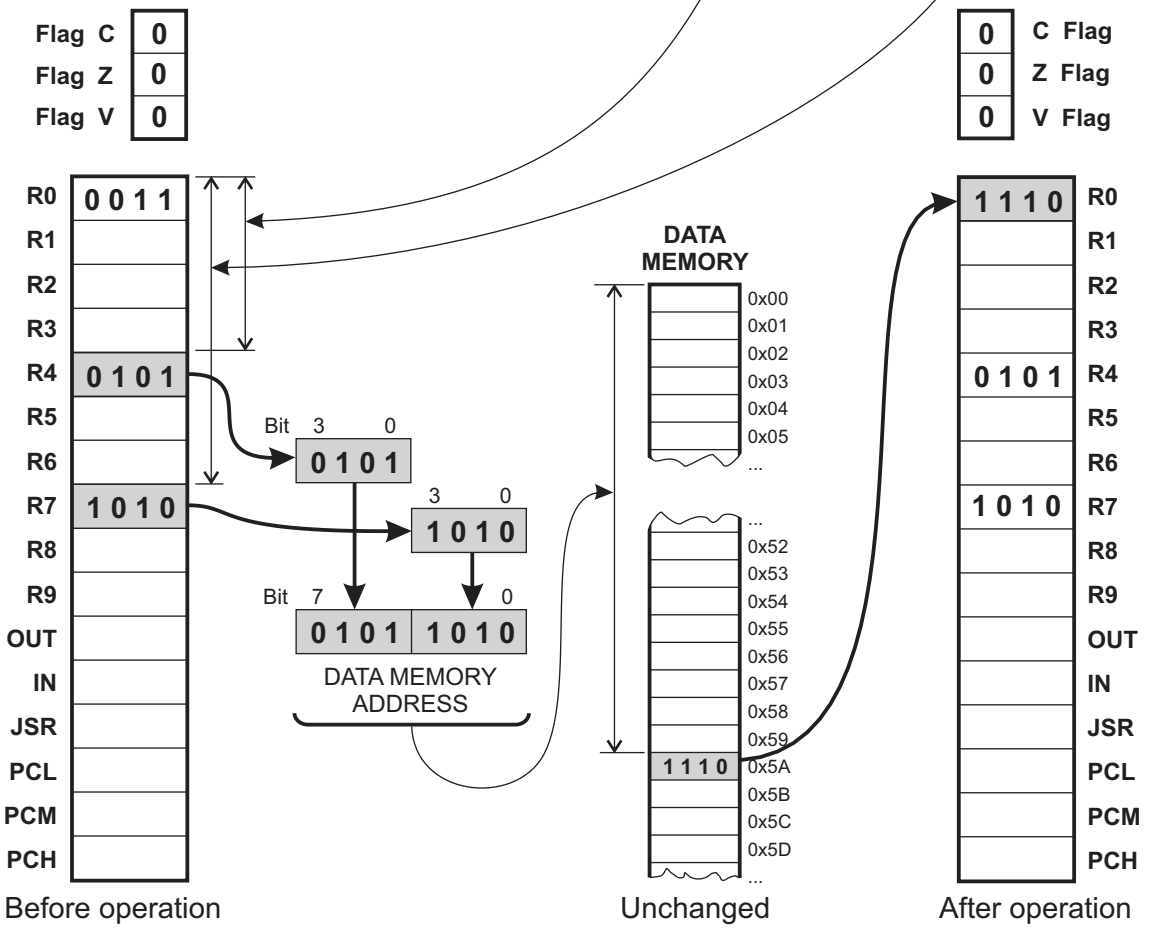
The "1011" bits are the MOV R0,[XY] opcode
The "XXXX" bits select the operand RX
The "YYYY" bits select the operand RY

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 R 0[XY]

将由寄存器RX（高半字节）和RY（低半字节）间接寻址的数据存储器的内容移动到寄存器R0

语法：{label} MOV R0，[XY]

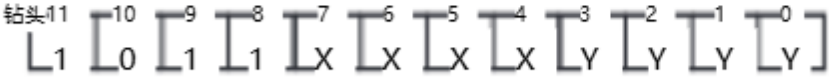
操作数：RX ∈ [R0. R15]
RY ∈ [R0. R15]

操作方式：(R0)←（（RX）：（RY））

产品描述：将由寄存器RX（高半字节）和RY（低半字节）间接寻址的数据存储器的4位内容移至寄存器R0。
RX和RY必须使用寄存器直接寻址。

受影响的旗帜：一个也没有。

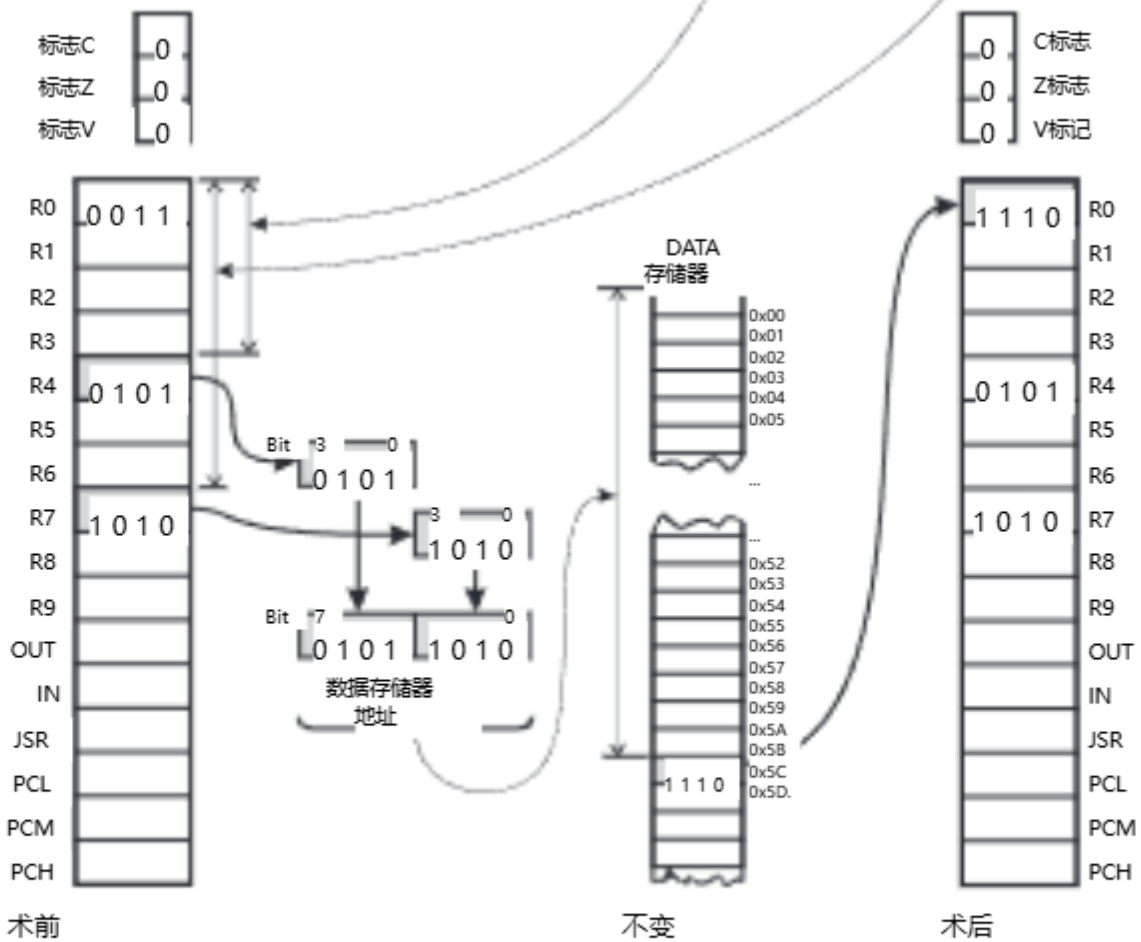
编码方式：



“1011”位是MOV R0，[XY]操作码。“0”位选择操作数RX。“YYYY”位选择操作数RY

范例：

MOV R0，[R4：R7]



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

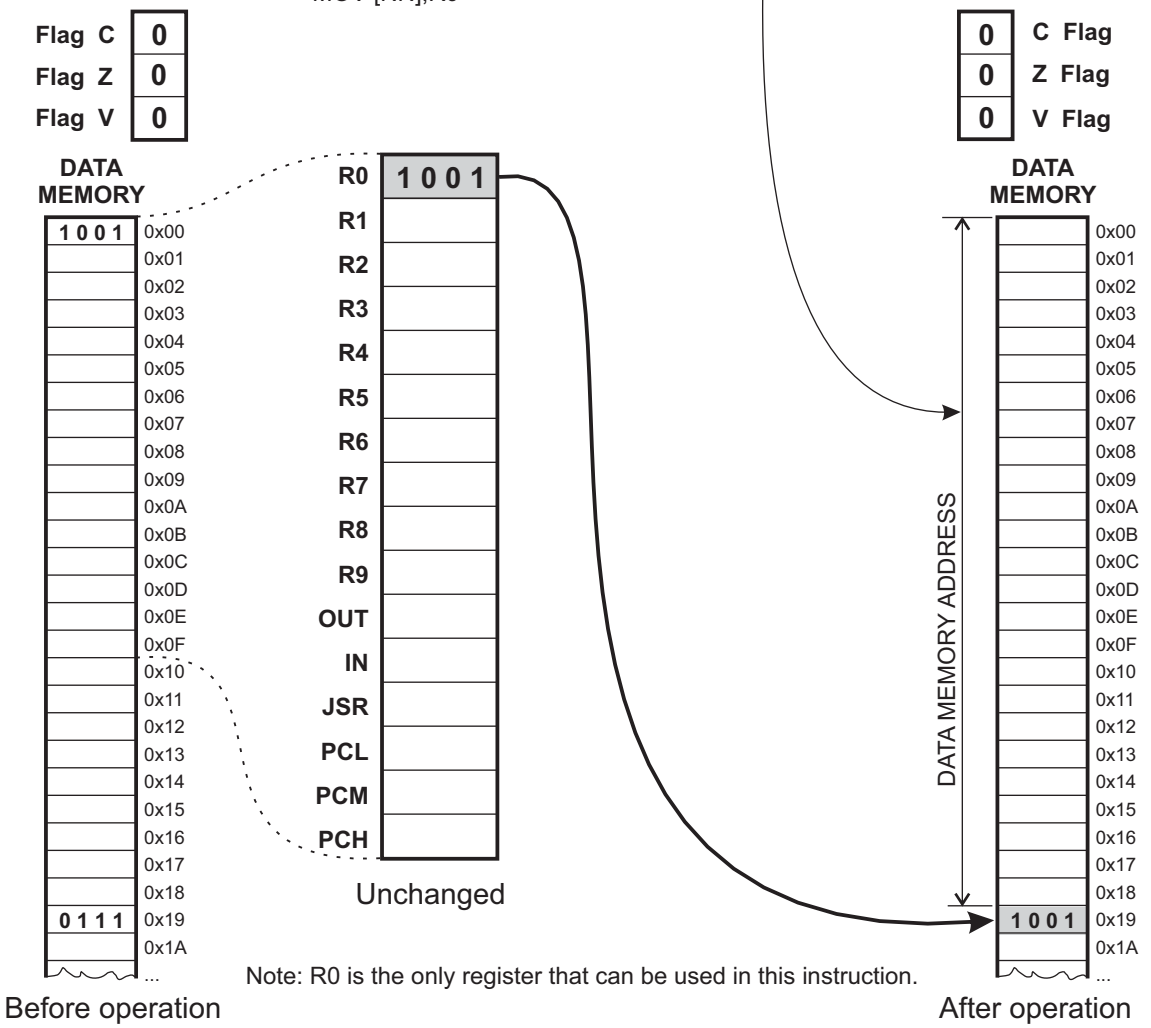
Example:

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.

MOV [NN,] R0

将寄存器R0的内容移动到由文字NN寻址的数据存储器

语法: {label} MOV [NN], R0

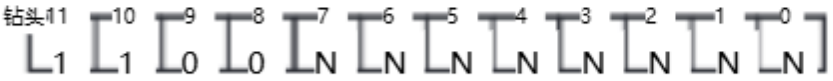
操作数: NN ∈ [0. 255]

操作方式: (NN)← (R0)

产品描述: 将寄存器R0的4位内容移至由无符号文字[NN]寻址的数据存储器。

受影响的旗帜: 一个也没有。

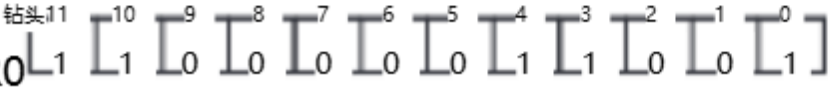
编码方式:

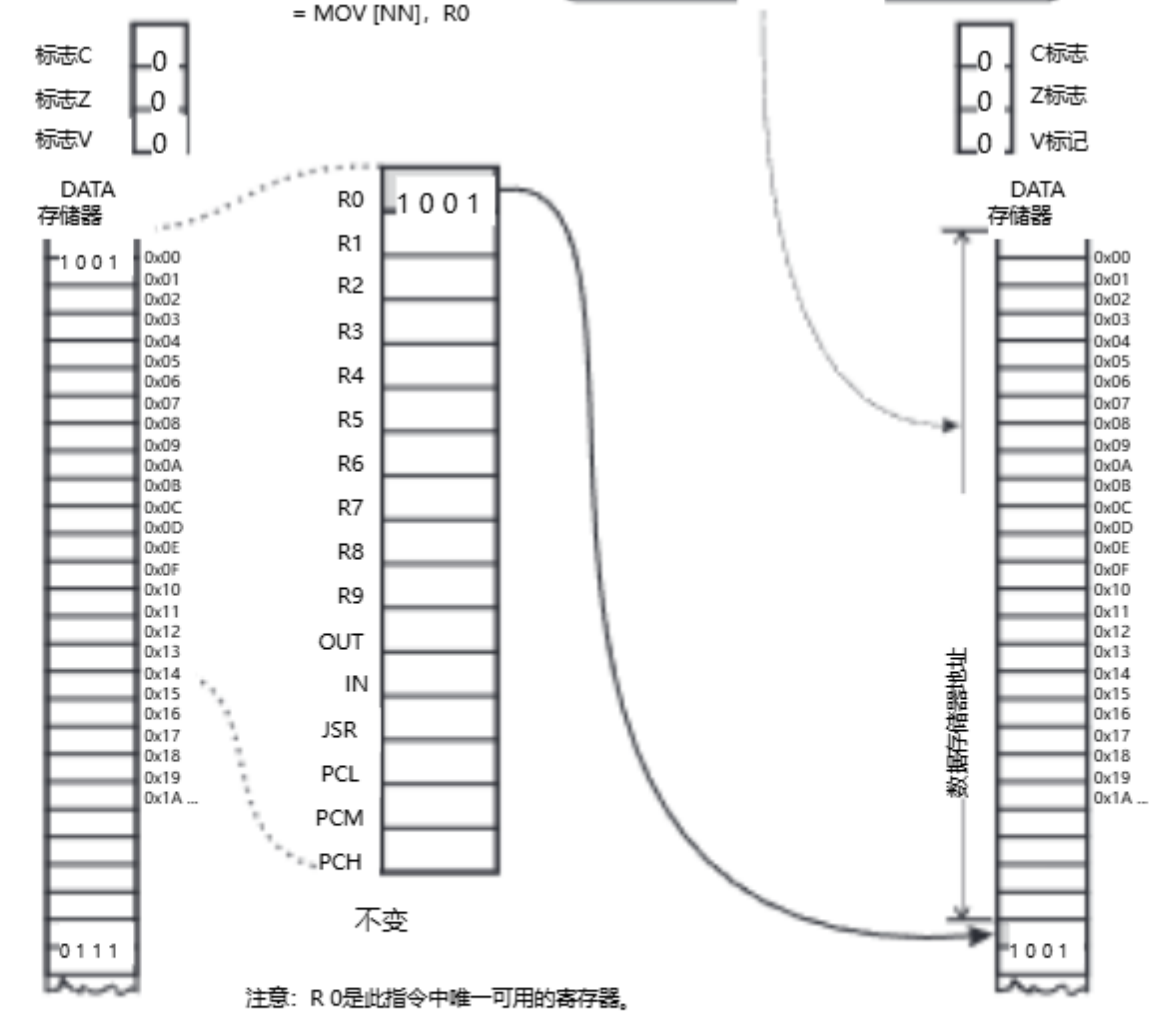


"1100" 位是MOV [NN]，R0操作码。
"NNNNNNNN" 位是无符号文字NN

范例:

MOV [0x19], R0





注意：R 0是此指令中唯一可用的寄存器。

术前 术后

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

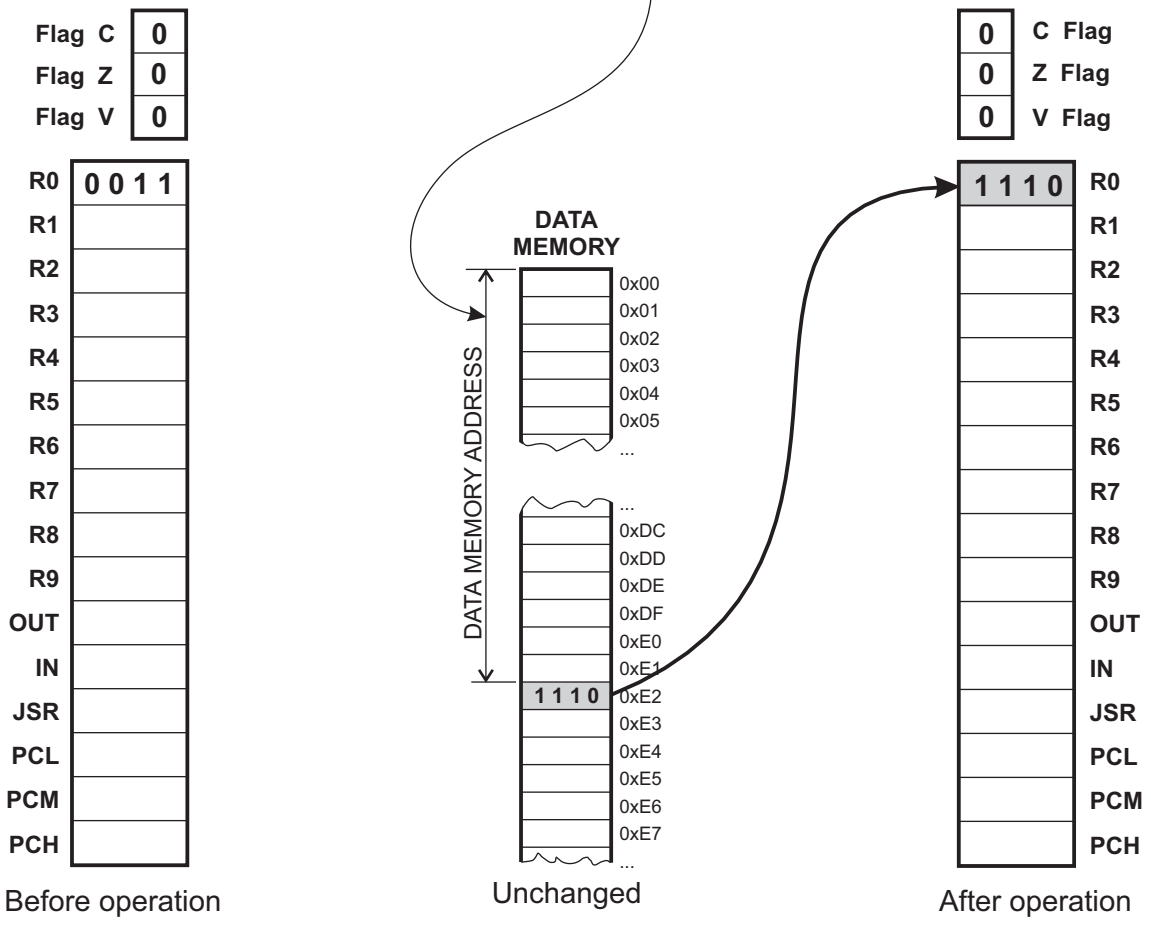
Example:

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 R0, [NN]

将由文字NN寻址的数据存储器的内容移动到寄存器R 0

语法: {label} MOV R0, [NN]

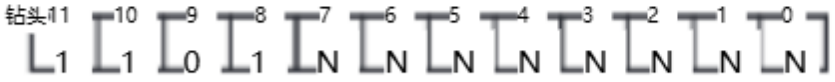
操作数: NN ∈ [0. 255]

操作方式: (R0)← (NN)

产品描述: 将由无符号字面值NN寻址的数据存储器的4位内容移动到寄存器R0。
R0必须使用寄存器直接寻址。

受影响的旗帜: 一个也没有。

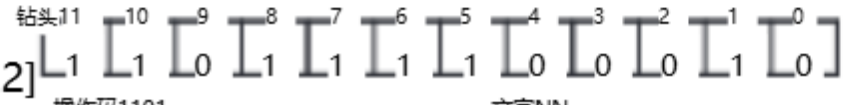
编码方式:

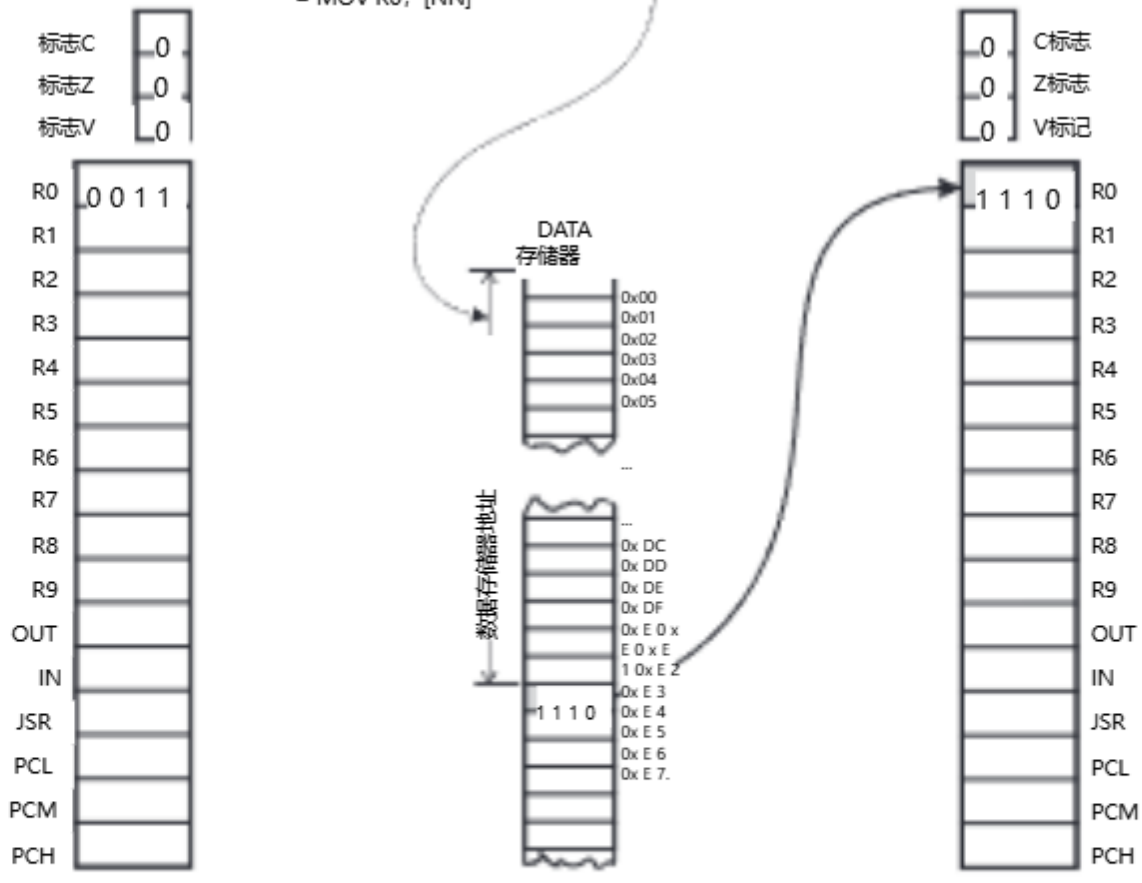


"1101"位是MOV R 0，[NN]操作码。
"NNNNNNNN"位是无符号文字NN

范例:

MOV R0，[0xE2]





术前 不变 术后

注意：R 0是此指令中唯一可用的寄存器。

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

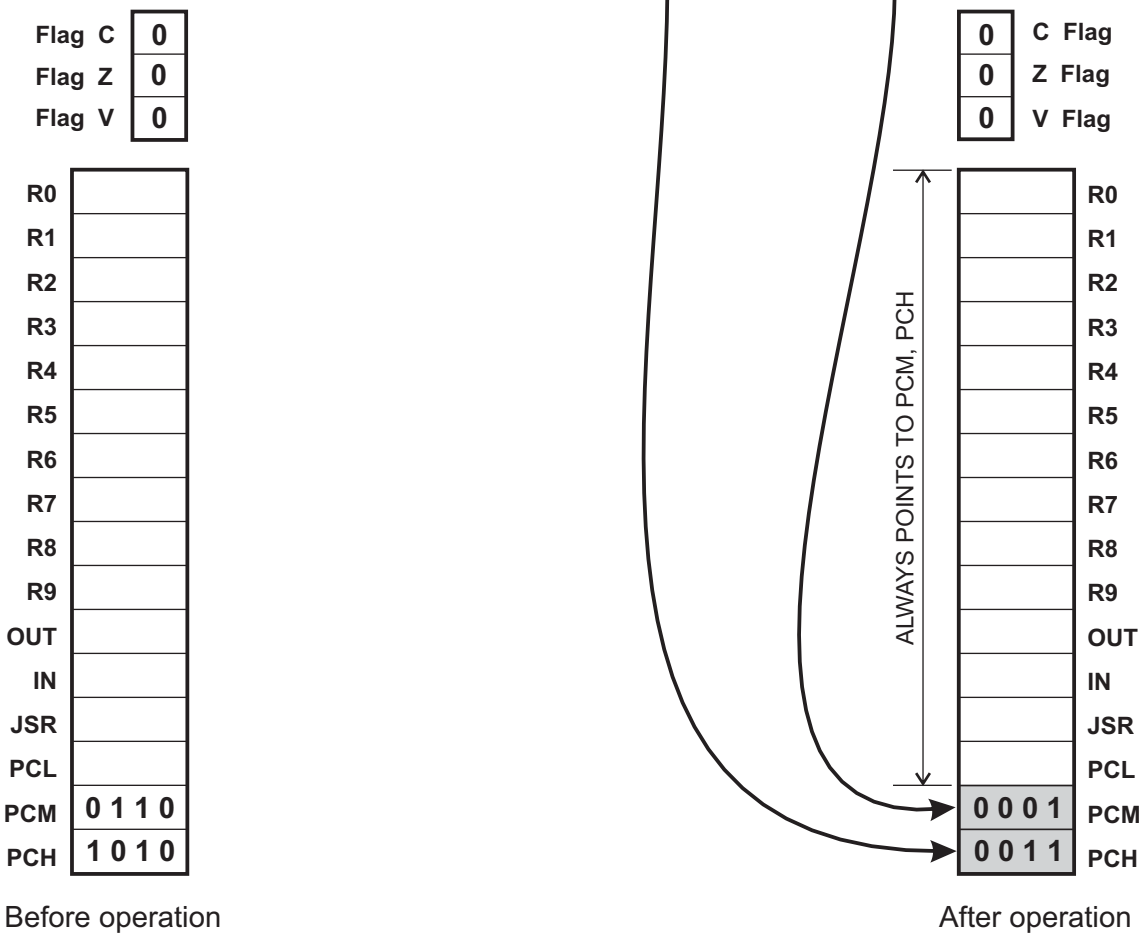
The "1110" bits are the LPC #NN opcode
The "NNNNNNNN" bits are literal #NN

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

OPCODE 1110 = LPC | LITERAL NN HIGH | LITERAL NN LOW



MOV PC，NN——将8位文字加载到寄存器PCM和PCH

语法：{label} MOV PC，NN

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

操作方式：(R14)← NN bit3. bit0，（R15）← NN bit7. bit4

产品描述：移动位3.. 0的8位字面值NN到R14，和位7. 4到R15。

受影响的旗帜：一个也没有。

编码方式：

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

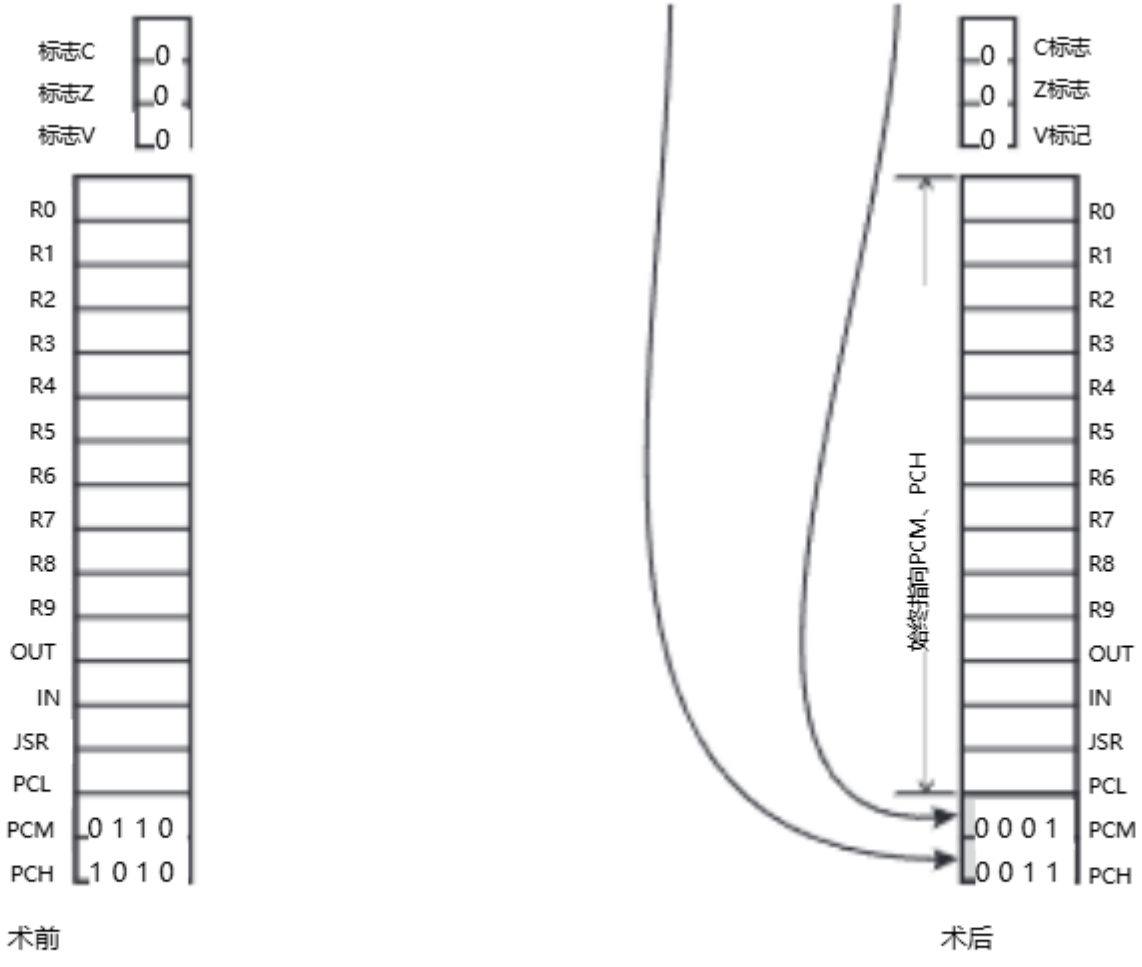
“1110”位是LPC #NN操作码。
“NNNNNNNN”位是文字#NN

范例：

MOV PC，0x 31

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

OPCODE 1110 = LPC　文字NN高　文献NN低



JR NN

Jump relative

Syntax: {label} JR NN

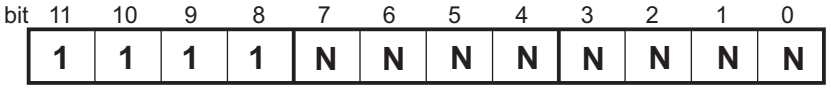
Operands: NN ∈ [-128...+127]

Operation: Program Counter ← Program Counter + NN signed

Description: Add signed integer value NN to the Program Counter

Flags affected: None.

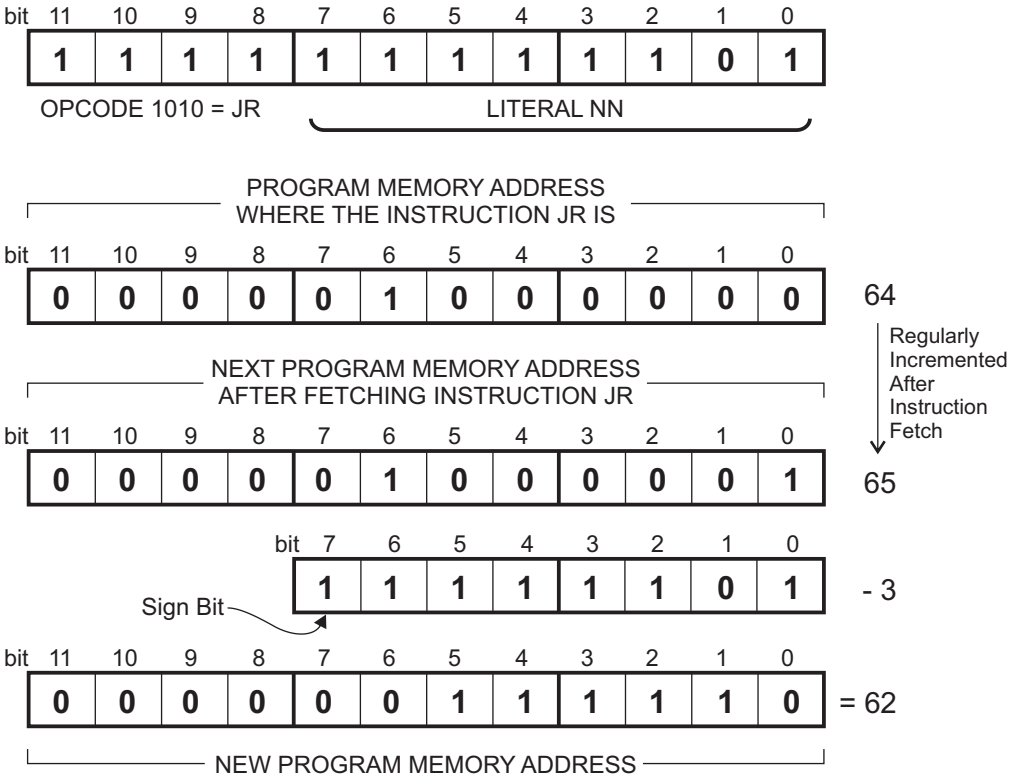
Encoding:



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

Example:

JR -3



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.

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

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.

JR NN 相对跳转

语法: {label} JR NN

操作数: ∈ N =—128. [127]

操作方式: 程序计数器←程序计数器+ NN签名

产品描述: 将有符号整数值NN添加到程序计数器

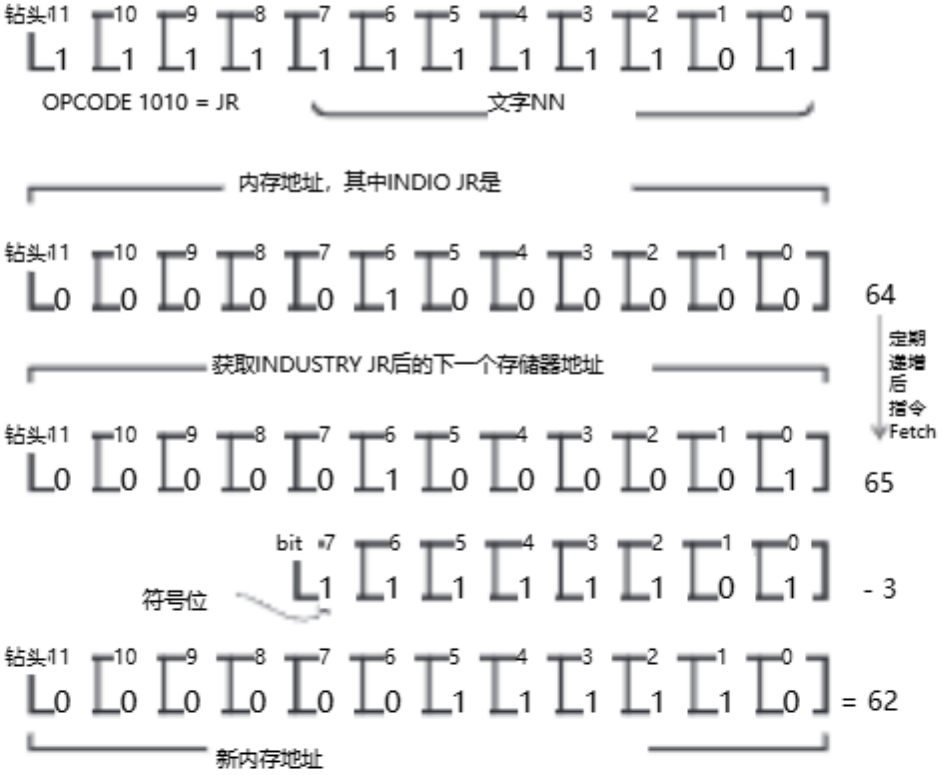
受影响的旗帜: 一个也没有。

编码方式:

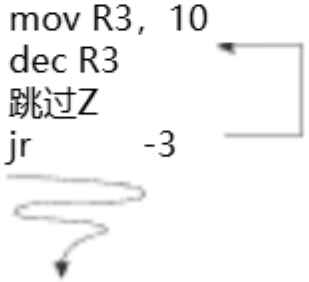
“1010”位是JR NN操作码。“NNNNNNNN”位是有符号的字面NN

范例:

JR-3



注1：在这个例子中，NN =-3，程序将循环不是三个，而是两个指令。这是因为程序计数器在JR指令提取之后、在地址计算发生之前已经递增。看看右边的程序流程示例，在跳过指令JR并继续进一步操作之前，程序将循环9次（加上一次常规通过）。



注二：允许跨页，即使它跨越地址1111 1111 1111和0000 0000 0000之间的边界，因此地址空间可以被视为无限环。

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

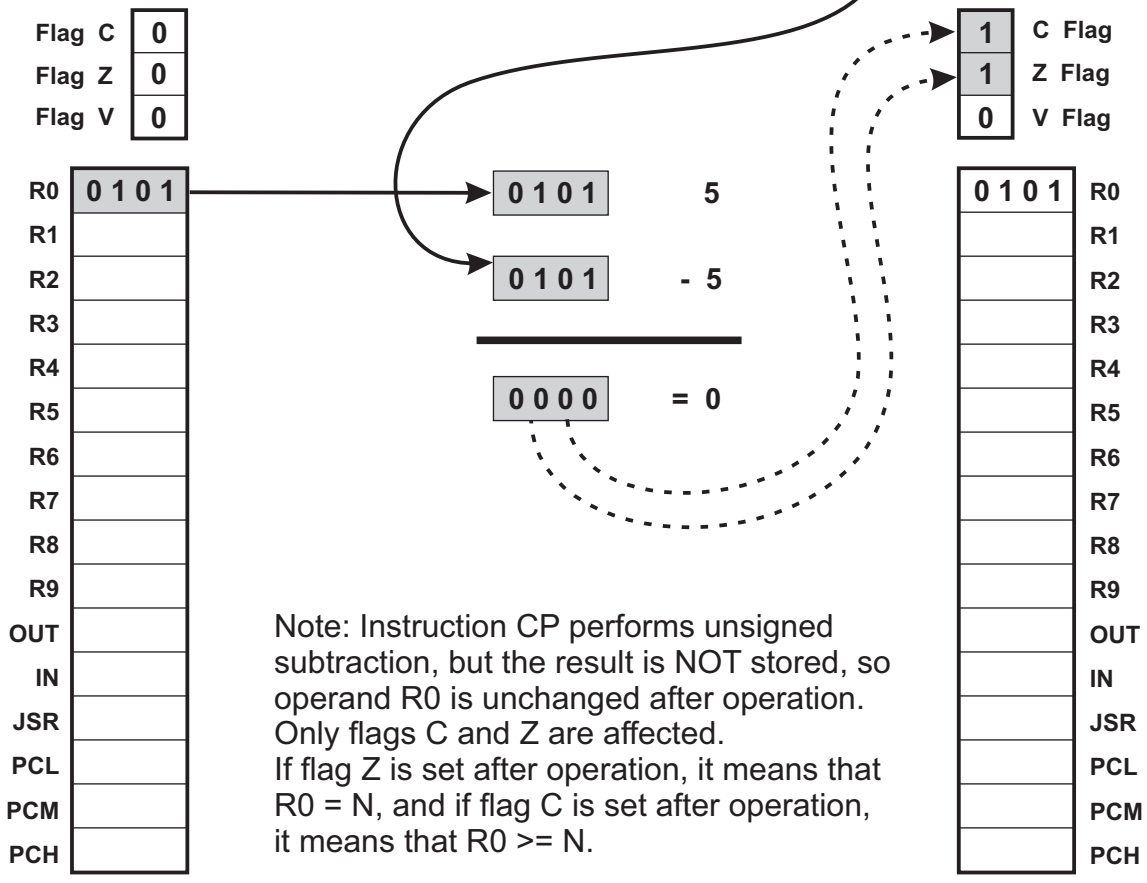
The "0000 0000" bits are the CP R0,N opcode
The "NNNN" bits are literal N

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



Before operation

After operation

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.

CP R0，N ——将寄存器R 0与4位文字进行比较

语法：{label} CP R0，N

操作数：R0
N ∈ 0. 15

操作方式：(R0)- N，仅设置标志

产品描述：计算无符号R 0- N并更新C和Z标志。
不存储减法的结果。

受影响的旗帜：如果（R 0）> N或（R 0）= N，则设置C。否则，重置C。
如果（R 0）= N，则设置Z。否则，重置Z。

编码方式：

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

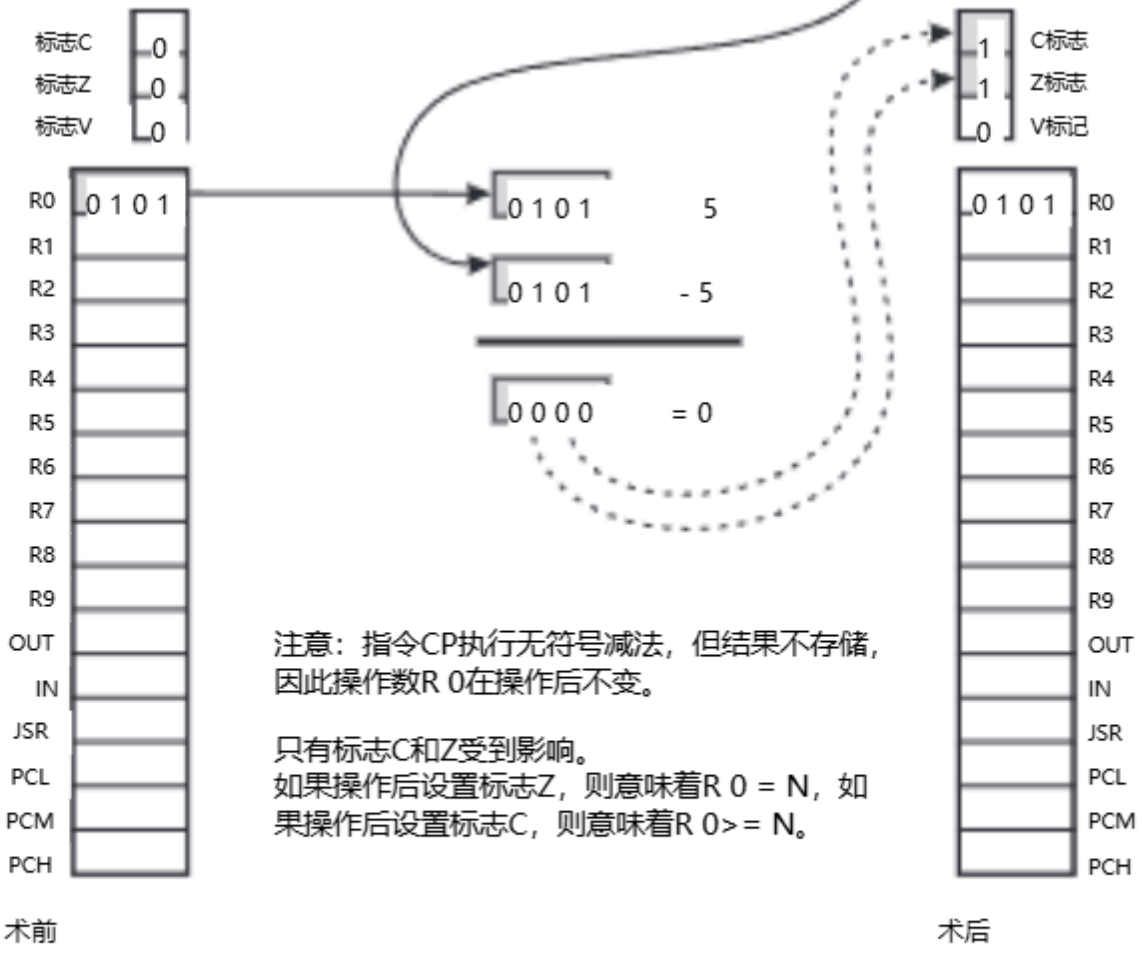
“0000 0000”位是CP R 0，N操作码。
“NNNN”位是文字N

范例：

CP R0，5

钻头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　　文字N



注意：指令CP执行无符号减法，但结果不存储，因此操作数R 0在操作后不变。

只有标志C和Z受到影响。
如果操作后设置标志Z，则意味着R 0 = N，如果操作后设置标志C，则意味着R 0>= 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

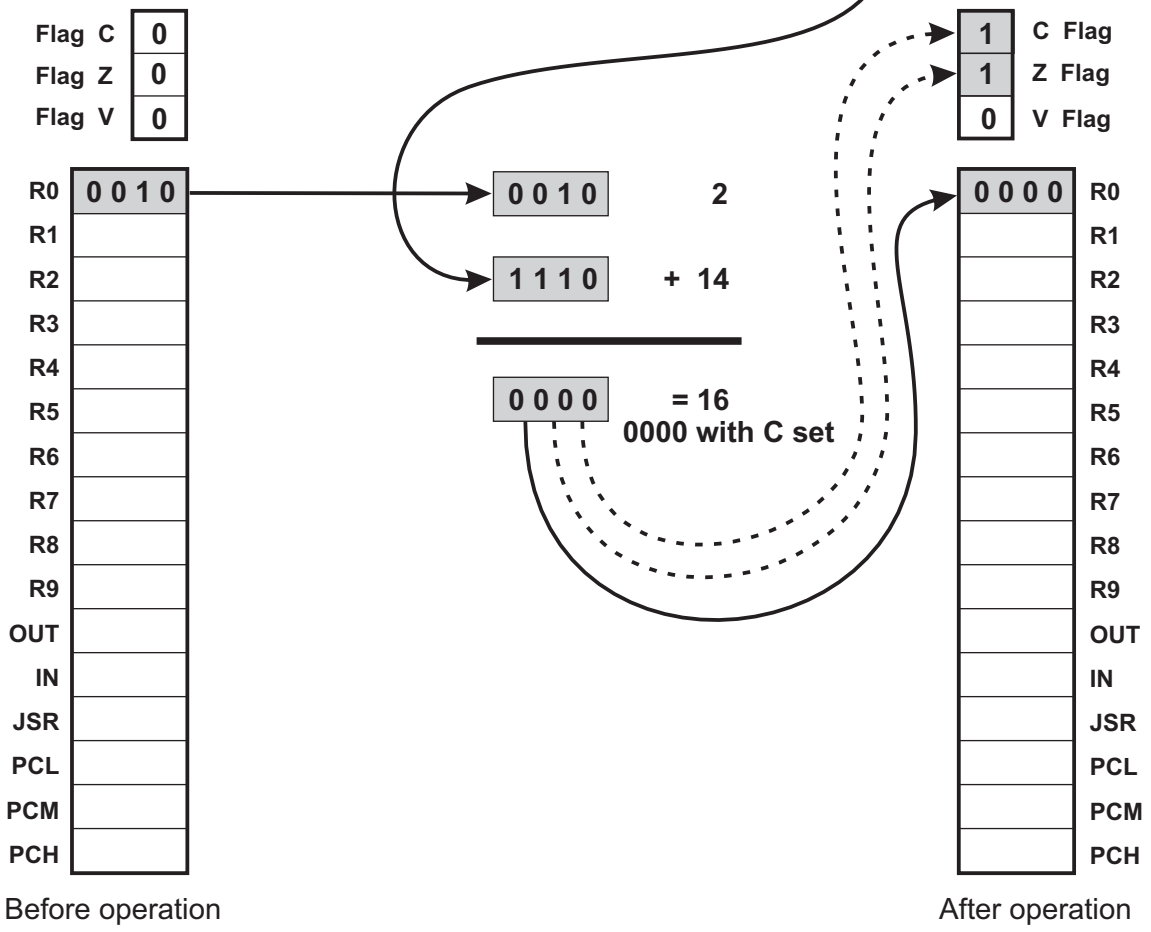
The "0000 0001" bits are the ADD R0,N opcode
The "NNNN" bits are literal N

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

ADD R0，N —向寄存器R0添加4位文字

语法： {label}添加 R0，N

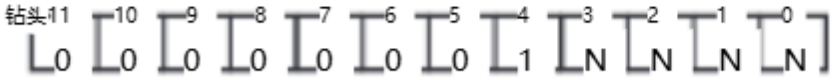
操作数： R0
N ∈ 0. 15

操作方式： (R0)←（R0）+ N

产品描述： 将文字N的内容与寄存器R0的内容相加，并将结果放回寄存器R0。

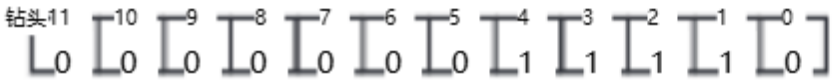
受影响的旗帜： 如果存在溢出（如果（R0）+N>15），则设置C。
否则，重置C。
如果操作后结果=0000，则设置Z。否则，重置Z。

编码方式：

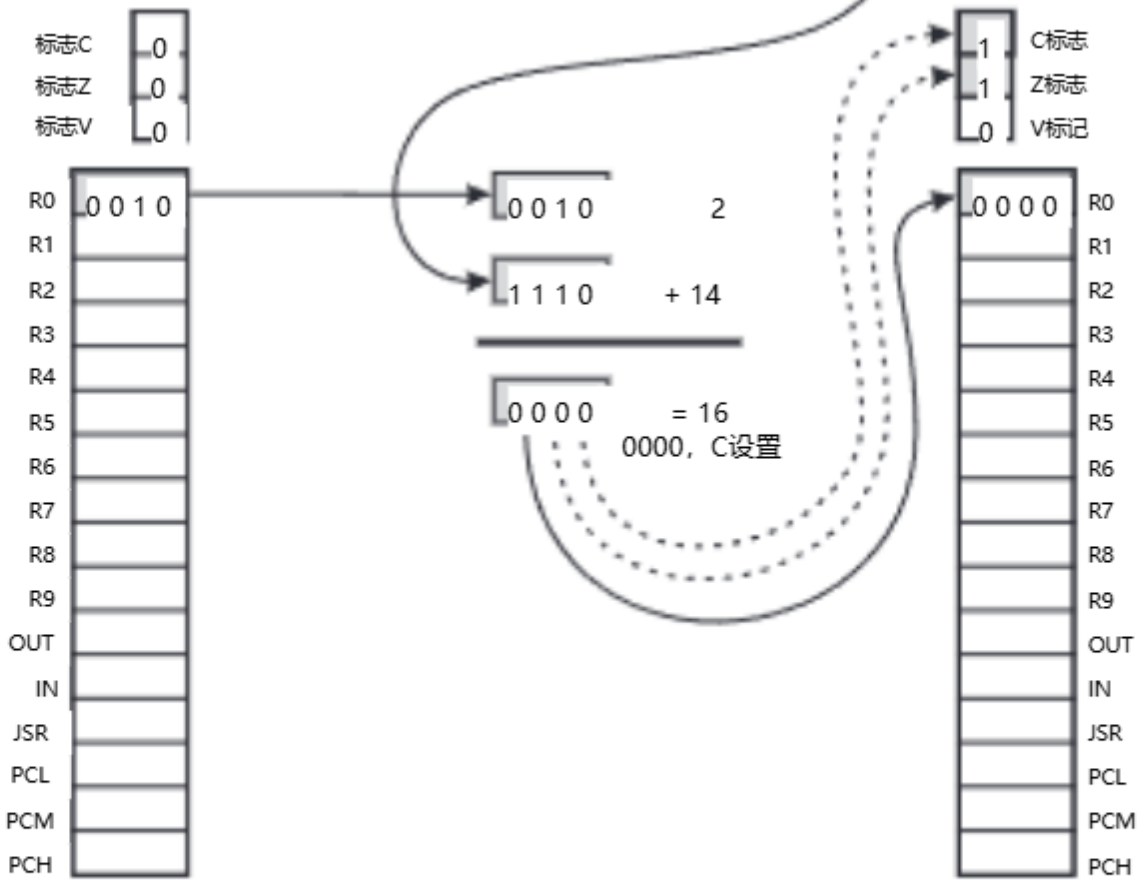


“0000 0001”位是ADD R0，N操作码。“NNNN”位是文字N

范例：



添加R0，14



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

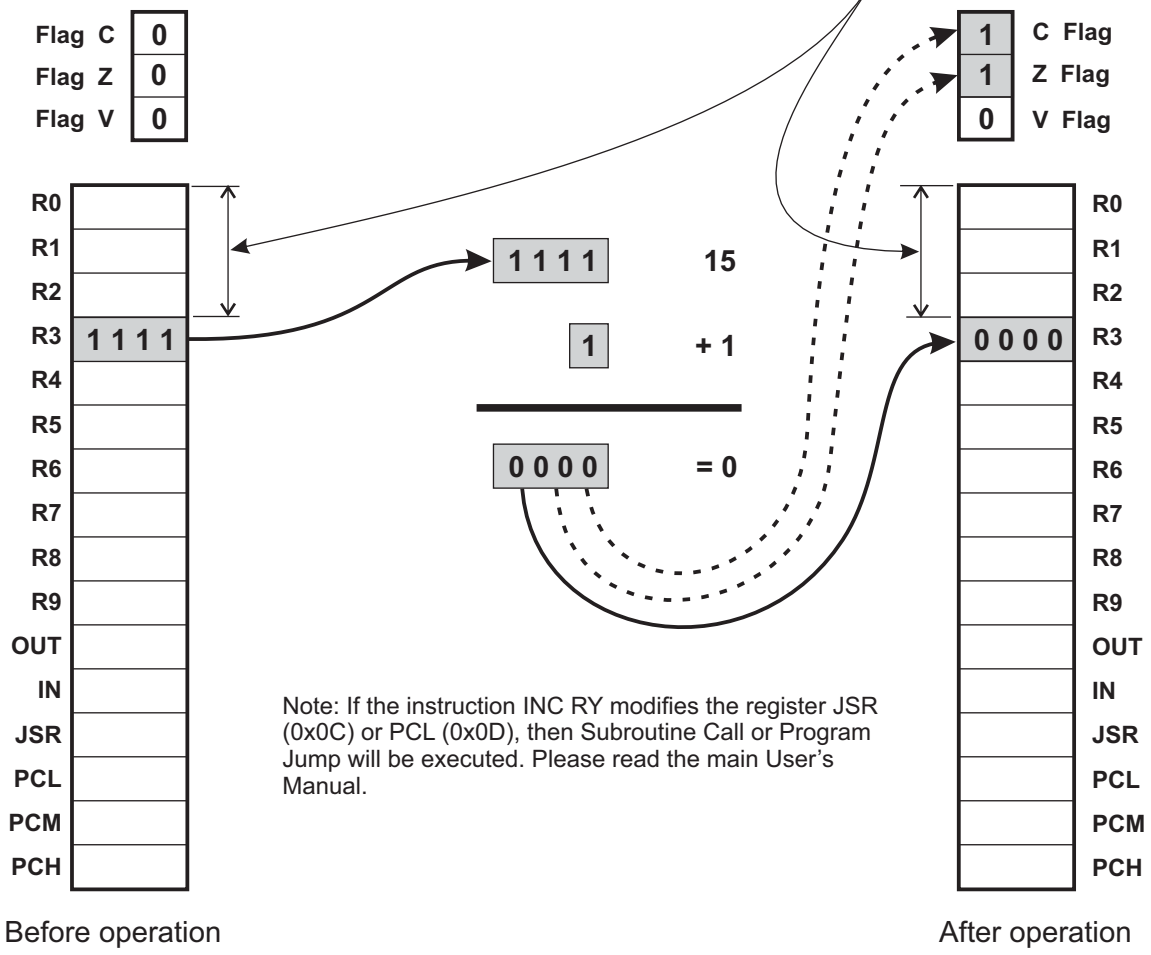
Example:

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

INC RY

将寄存器RY递增1

语法：{label} INC RY

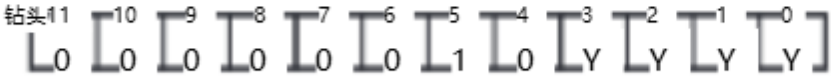
操作数：RY ∈ [R0. R15]

操作方式：(RY)← (RY) +1

产品描述：将4位寄存器RY的内容加1，并将结果放回寄存器RY。

受影响的旗帜：如果操作后结果=0000，则设置标志Z和C。否则，重置标志Z和C。

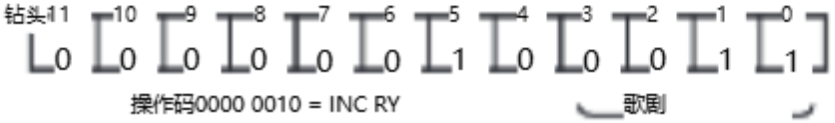
编码方式：

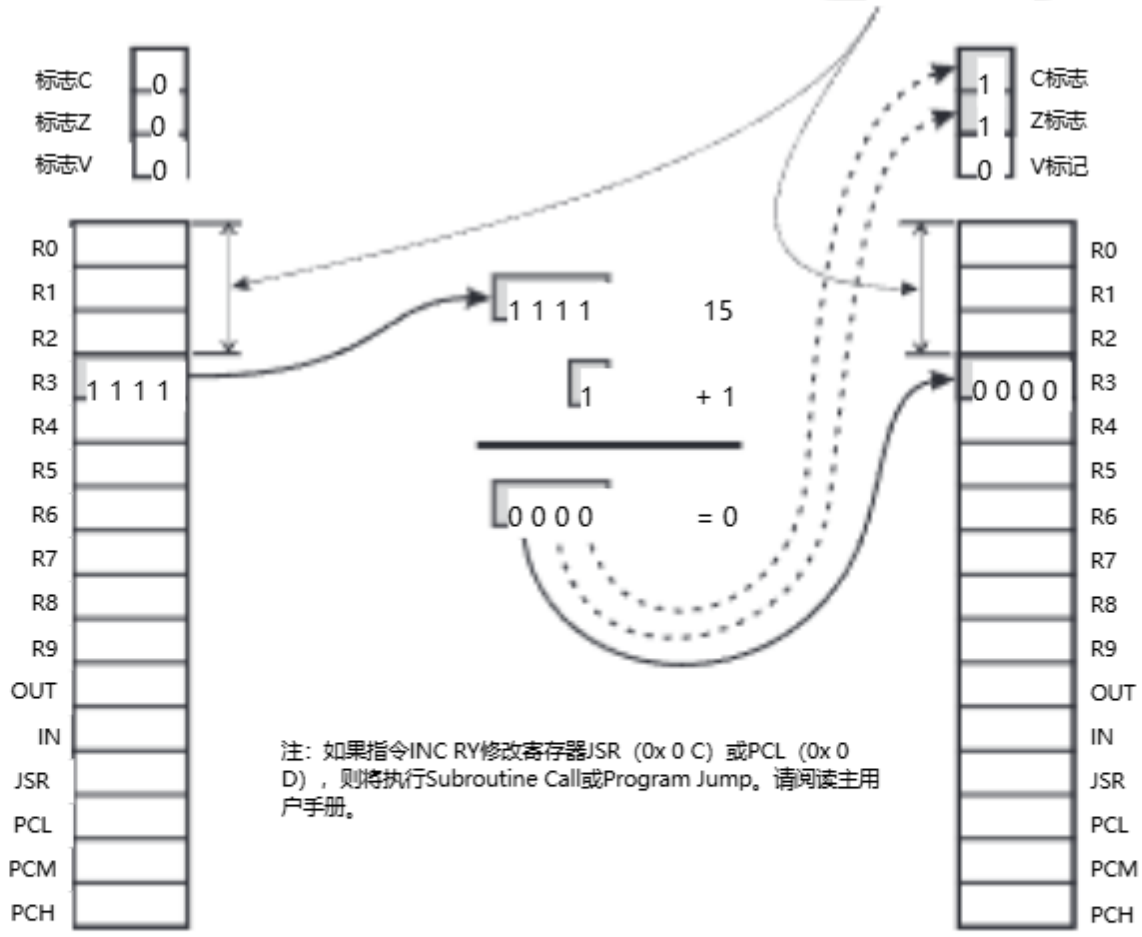


“0000 0010”位是INC RY操作码。“YYYY”位选择操作数RY

范例：

INC R3





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

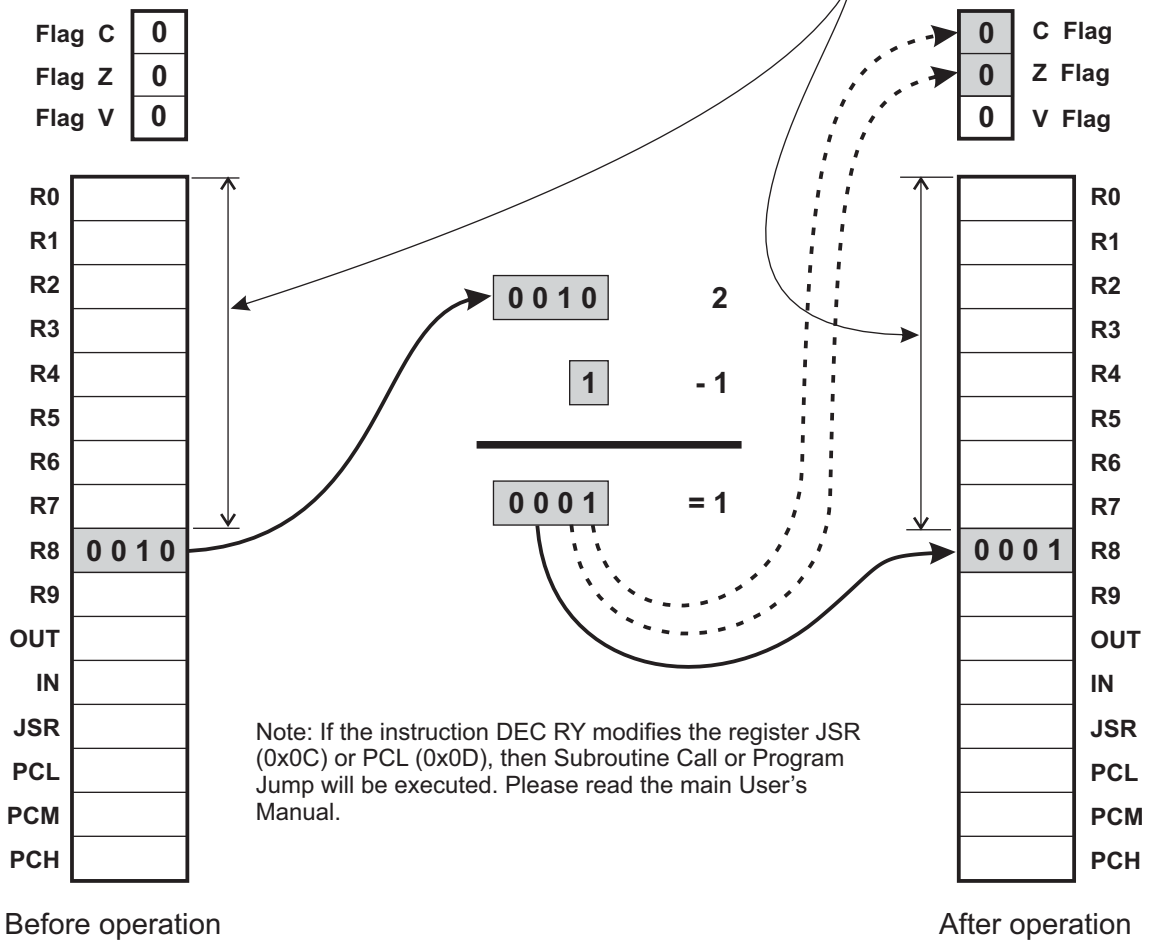
Example:

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.

DEC RY

寄存器RY减1

语法：{label} DEC RY

操作数：RY ∈ [R0. R15]

操作方式：(RY)← (RY) -1

产品描述：从4位寄存器RY的内容中减去1，并将结果放回寄存器RY。

受影响的旗帜：如果操作后结果=0000，则设置标志Z。否则，重置标志Z。
如果运算后结果=1111，则复位标志C。
否则，设置标志C。

编码方式：

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

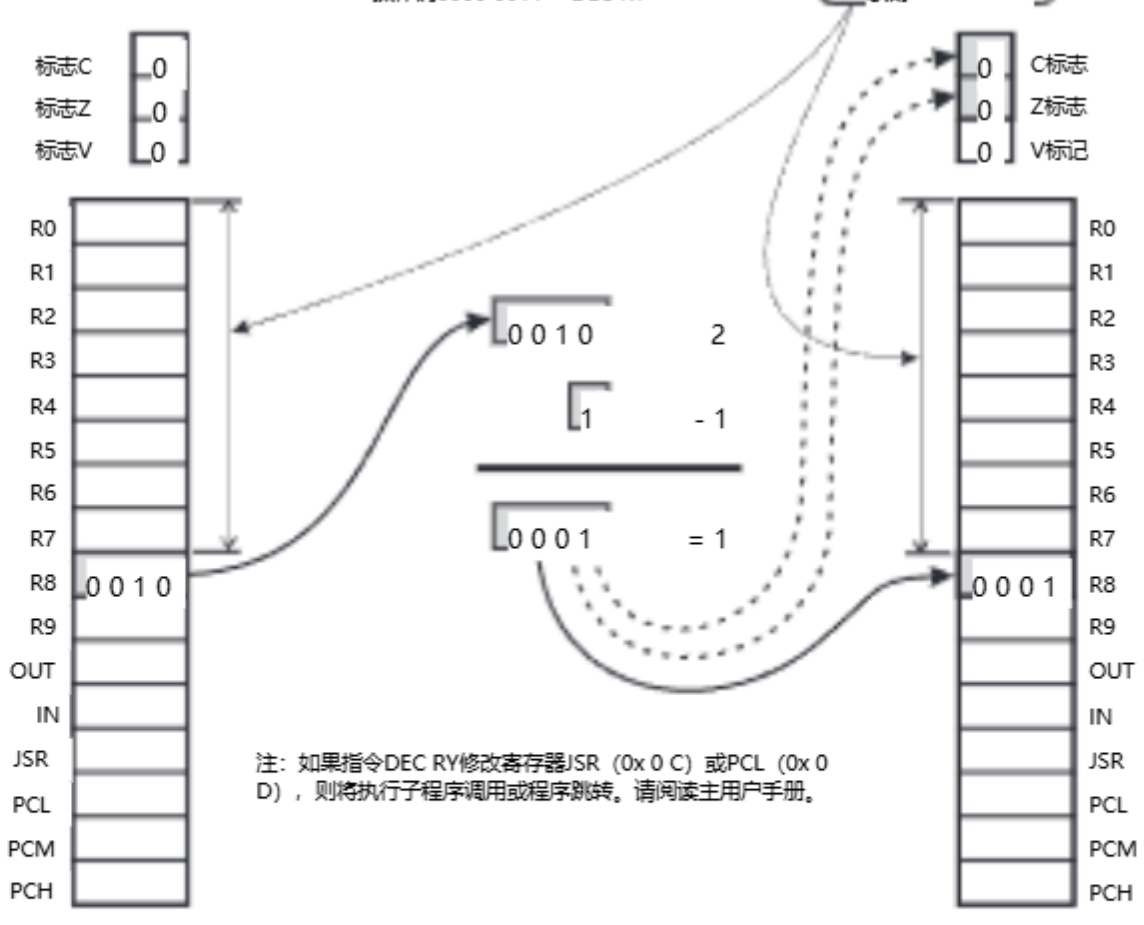
“0000 0011”位是DEC RY操作码。“YYYY”位选择操作数RY

范例：

DEC R8

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

操作码0000 0011 = DEC RY　　歌剧



注：如果指令DEC RY修改寄存器JSR（0x 0 C）或PCL（0x 0 D），则将执行子程序调用或程序跳转。请阅读主用户手册。

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

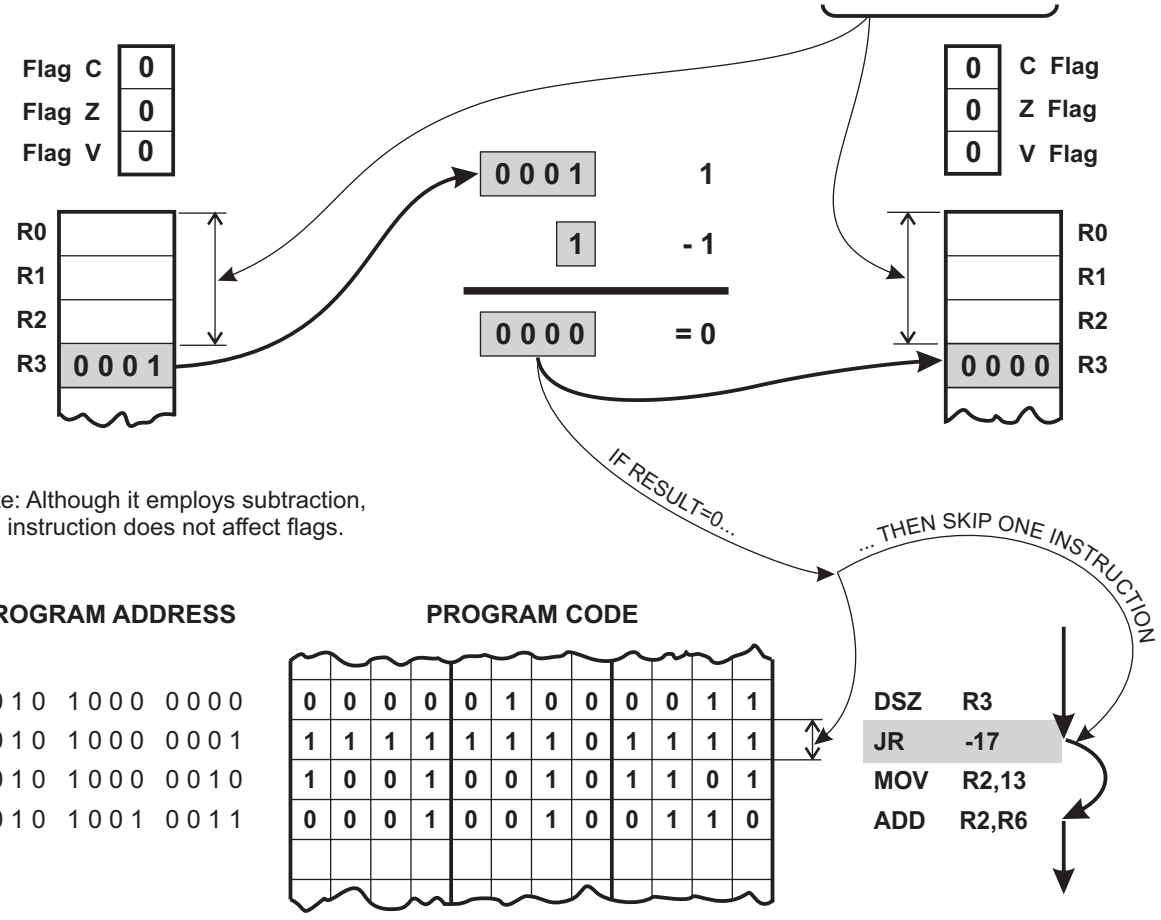
Example:

DSZ R3

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



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.

DSZ RY

递减寄存器RY，如果结果=0，则跳过下一条指令

语法：{label} DSZ RY

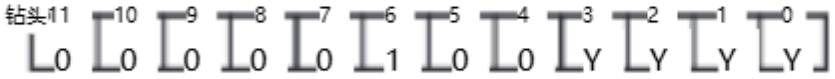
操作数：RY ∈ [R0. R15]

操作方式：(RY)← （RY）-1，如果结果=0，则PC ← PC+1

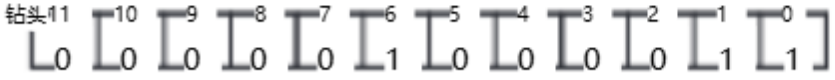
产品描述：从4位寄存器RY的内容中减去1，如果结果=0，则将程序计数器递增1。

受影响的旗帜：一个也没有。

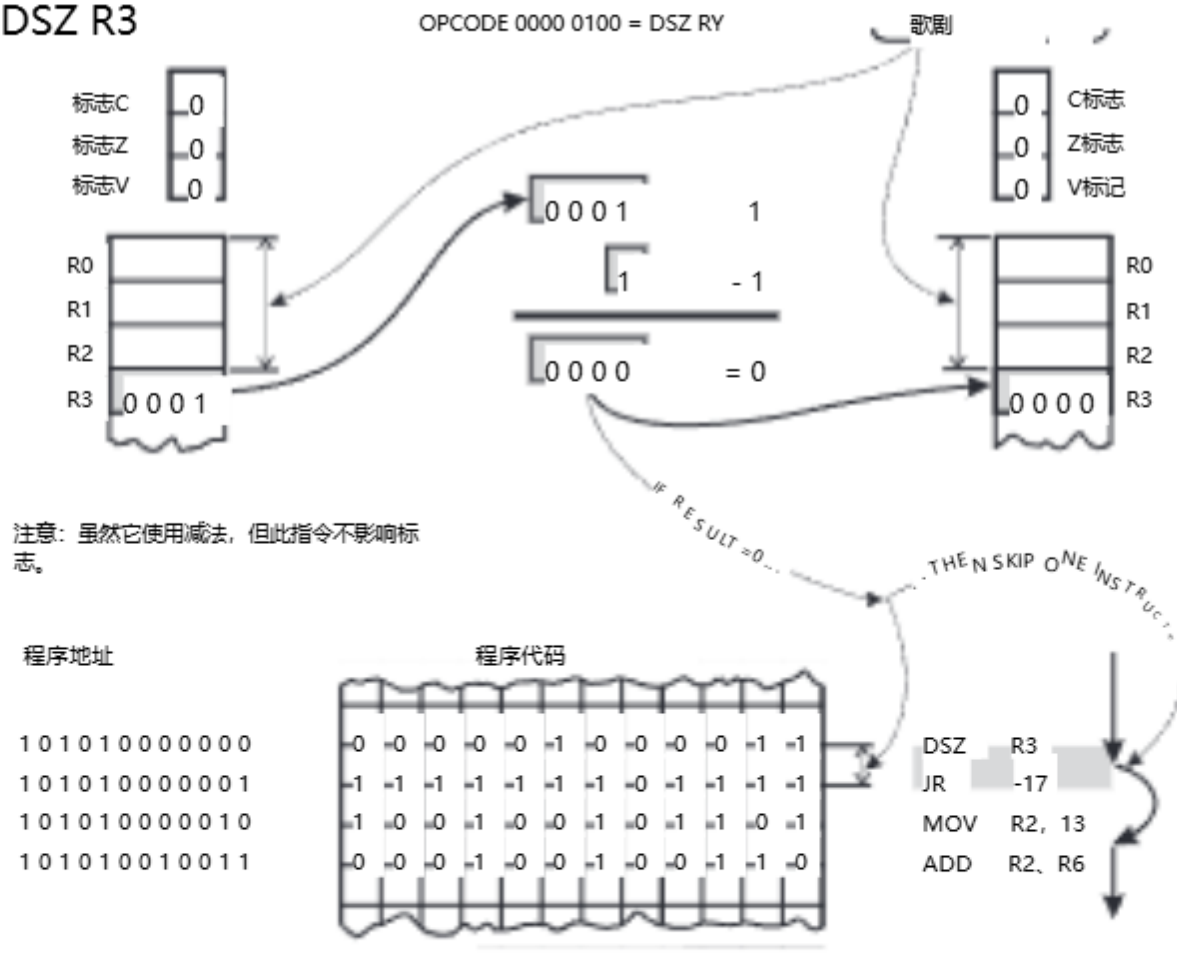
编码方式：



范例：“0000 0100”位是DSZ RY操作码。“YYYY”位是操作数Y



DSZ R3



注意事项：在本例中，寄存器R3在递减后为=0，因此指令DSZ R3导致程序跳过地址1010 1000 0001上的一条指令。程序继续在地址1010 1000 0010执行。

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

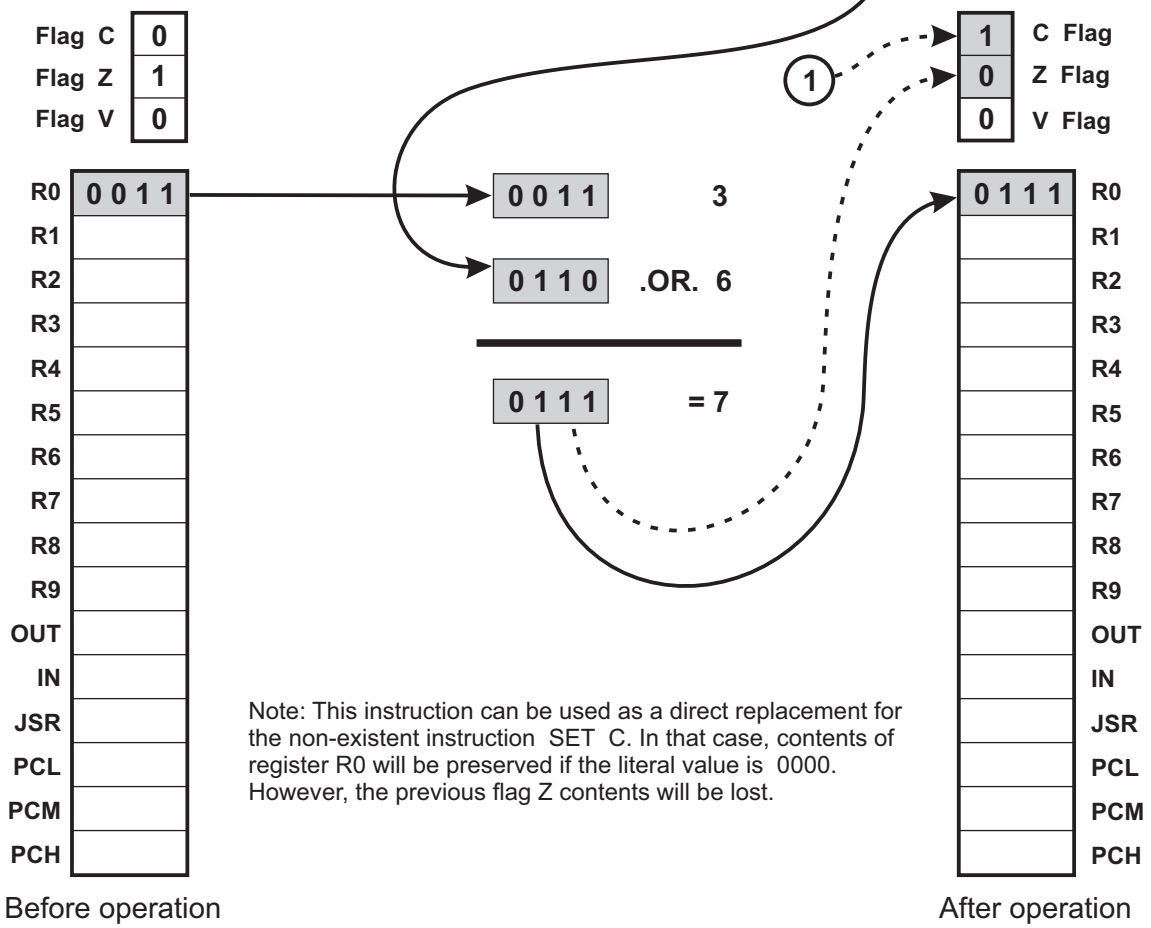
The "0000 0101" bits are the OR R0,N opcode
The "NNNN" bits are literal N

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

或R0，N

包含或寄存器R0与4位文字N

语法： {label}或 R0，N

操作数： R0
N ∈ 0. 15

操作方式： (R0)← （R0）. OR. N，C ← 1

产品描述： 计算4位寄存器R0与4位文字值N的逻辑“或”运算，并将结果放回寄存器R0。

受影响的旗帜： 如果运算后结果=0000，则设置Z。否则，重置Z

编码方式：

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

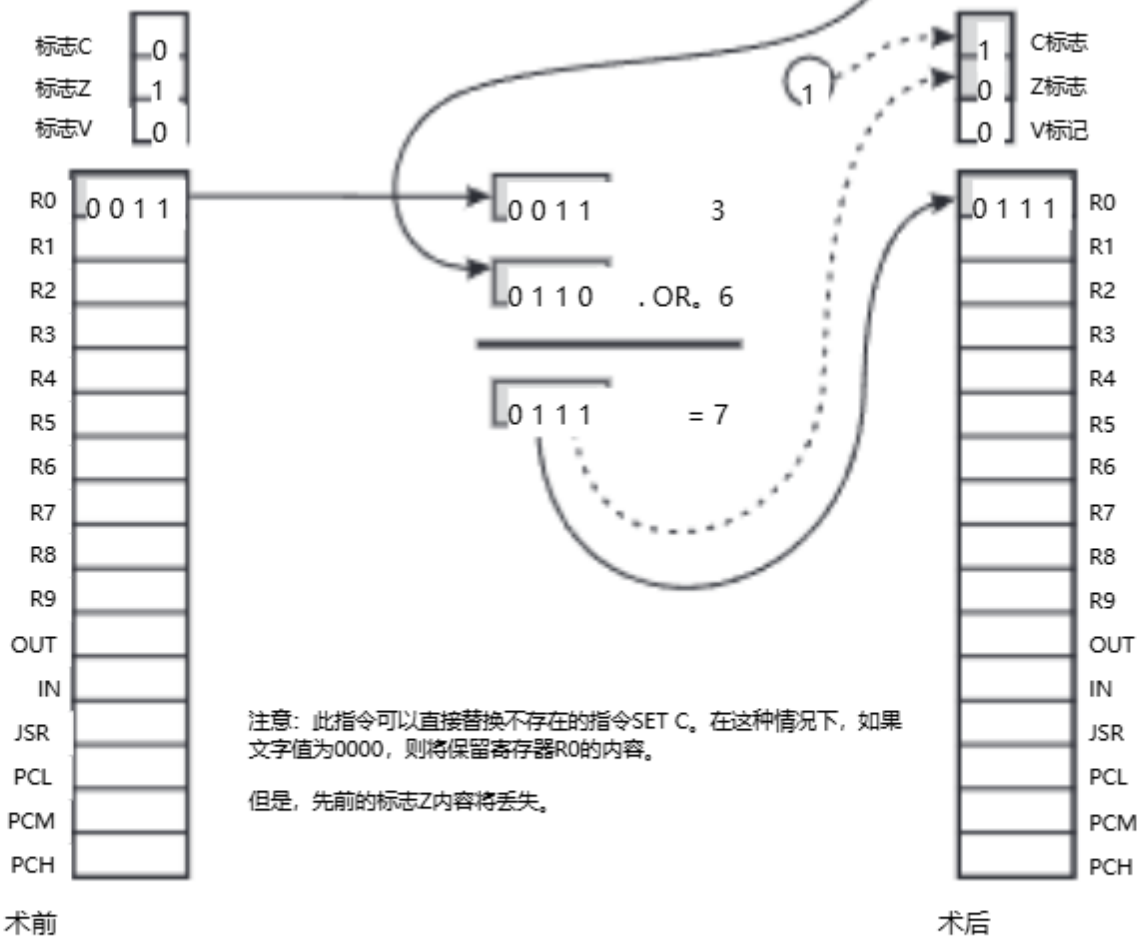
“0000 0101”位是OR R0，N操作码。“NNNN”位是文字N

范例：

OR R0，6

钻头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 文字N



注意：此指令可以直接替换不存在的指令SET C。在这种情况下，如果文字值为0000，则将保留寄存器R0的内容。

但是，先前的标志Z内容将丢失。

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

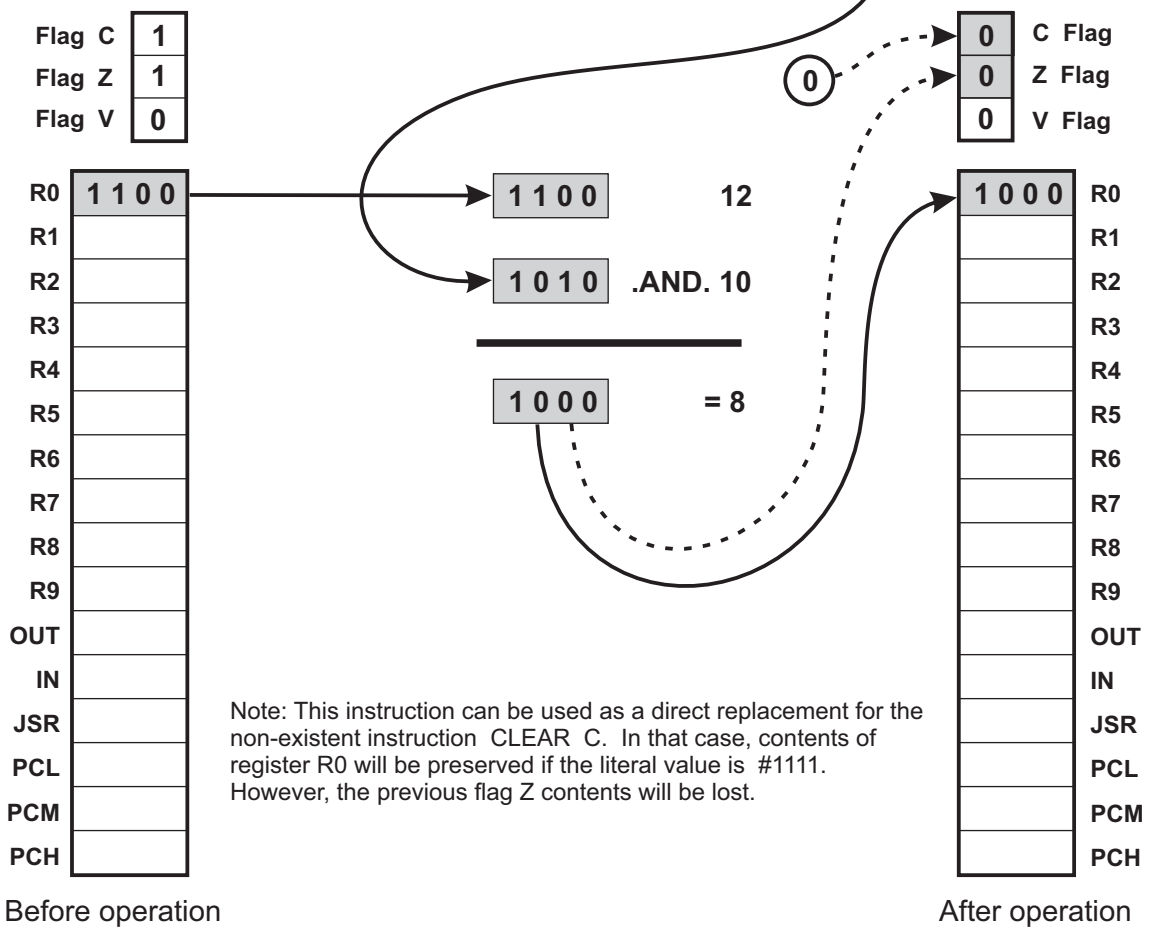
The "0000 0110" bits are the AND R0,N opcode
The "NNNN" bits are literal N

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



Before operation

After operation

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.

与R0，N

逻辑与寄存器R 0与4位文字N

语法：

{label} AND R0，N

操作数：

R0
N ∈ 0. 15

操作方式：

(R0)← （R0）. AND. N，C ← 0

产品描述：

用4位文字值N计算4位寄存器R0的逻辑“与”运算，并将结果放回寄存器R0。

受影响的旗帜：

标志C无条件复位如果运算后结果=0000，则置Z。否则，重置Z

编码方式：

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

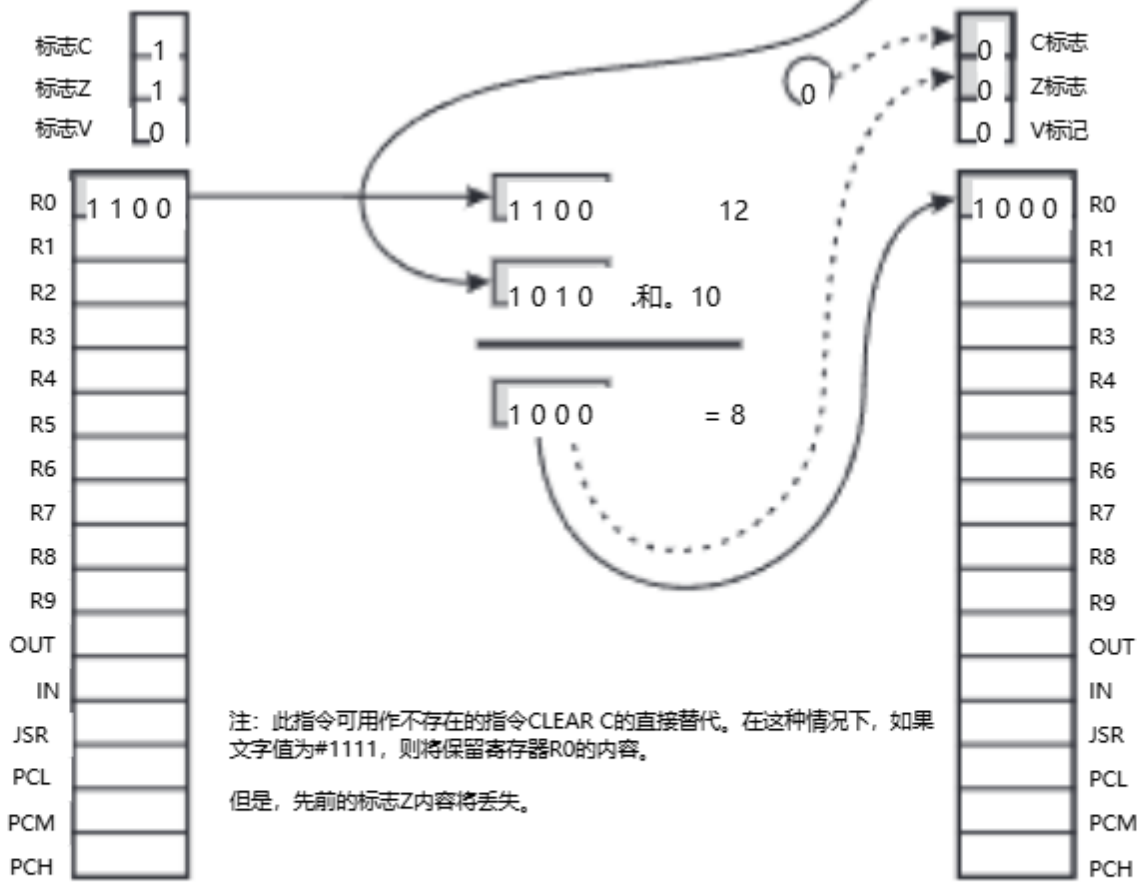
“0000 0110”位是AND R0，N操作码。
“NNNN”位是文字N

范例：

和R0，10

钻头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　　文字N



注：此指令可用作不存在的指令CLEAR C的直接替代。在这种情况下，如果文字值为#1111，则将保留寄存器R0的内容。

但是，先前的标志Z内容将丢失。

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

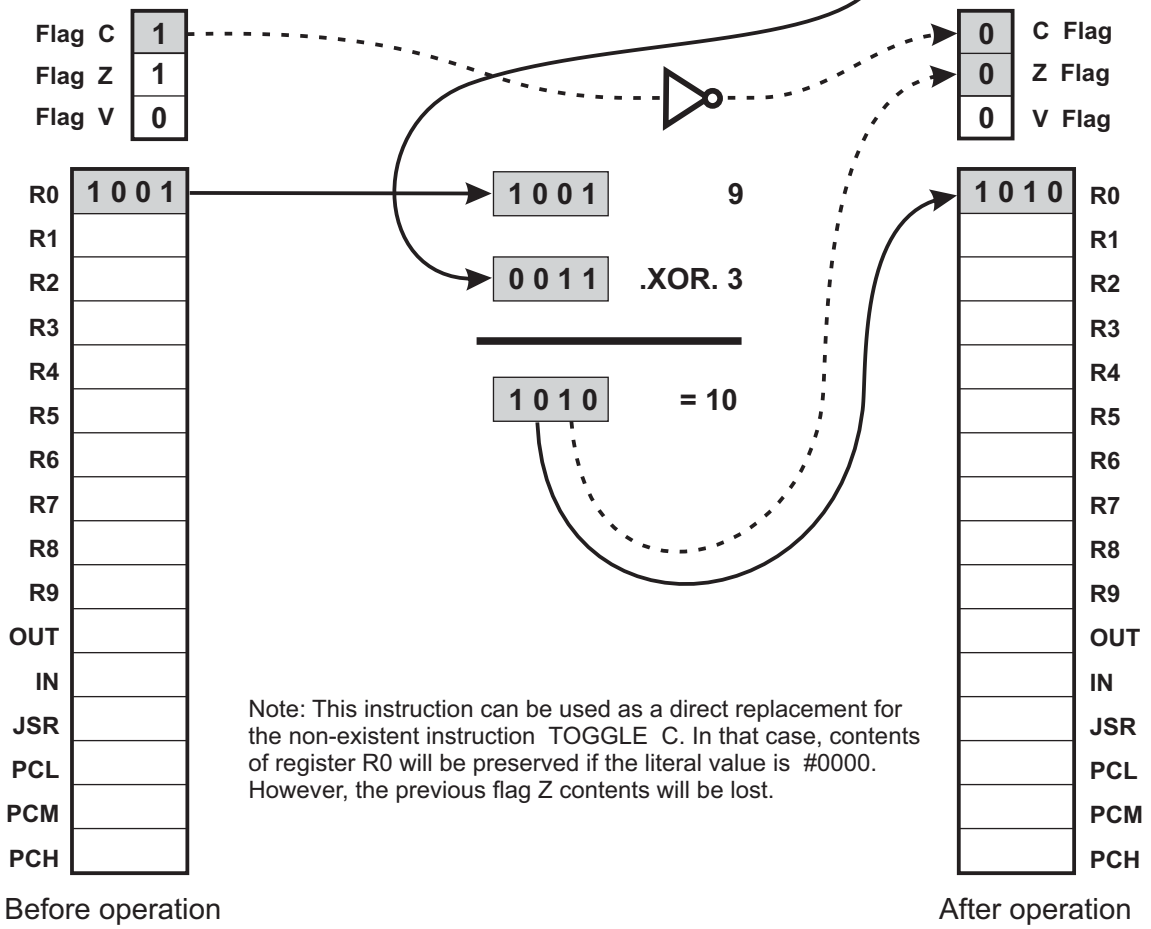
The "0000 0111" bits are the XOR R0,N opcode
The "NNNN" bits are literal N

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.

XOR R0，N —异或寄存器R 0与4位文字N

语法： {label}异或 R0，N

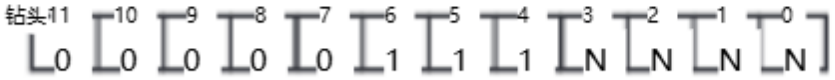
操作数： R0
N ∈ 0. 15

操作方式： (R0)← （R0）. XOR。N，C ← <$C

产品描述： 计算4位寄存器（R 0）与4位文字值N的逻辑异或运算，并将结果放回寄存器R 0。

受影响的旗帜： 标志C被无条件地切换（补）。如果操作后结果=0000，则设置Z。否则，重置Z

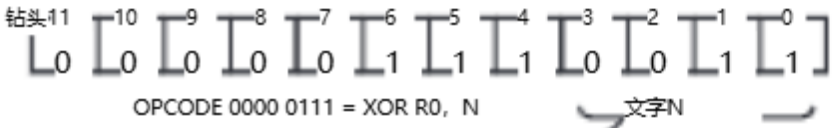
编码方式：

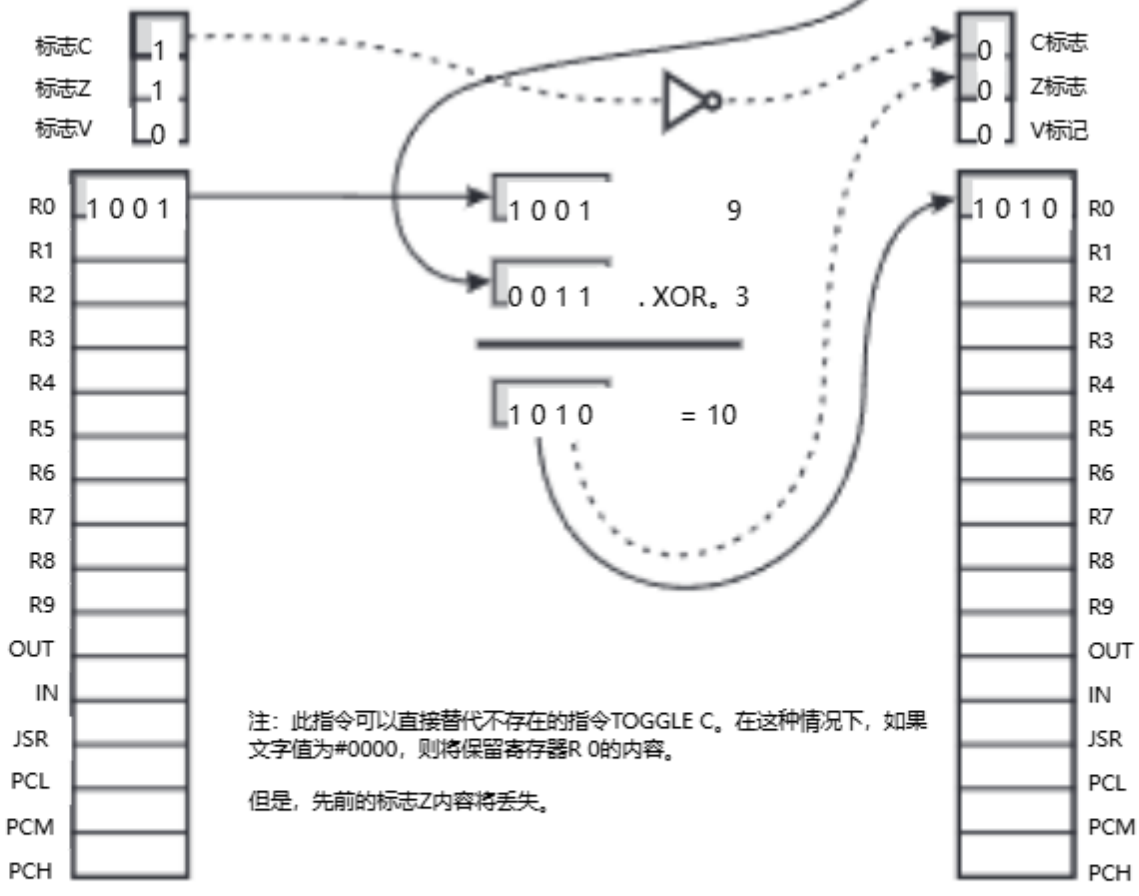


“0000 0111”位是XOR R 0，N操作码。
“NNNN”位是文字N

范例：

XOR R0，3





注：此指令可以直接替代不存在的指令TOGGLE C。在这种情况下，如果文字值为#0000，则将保留寄存器R 0的内容。

但是，先前的标志Z内容将丢失。

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

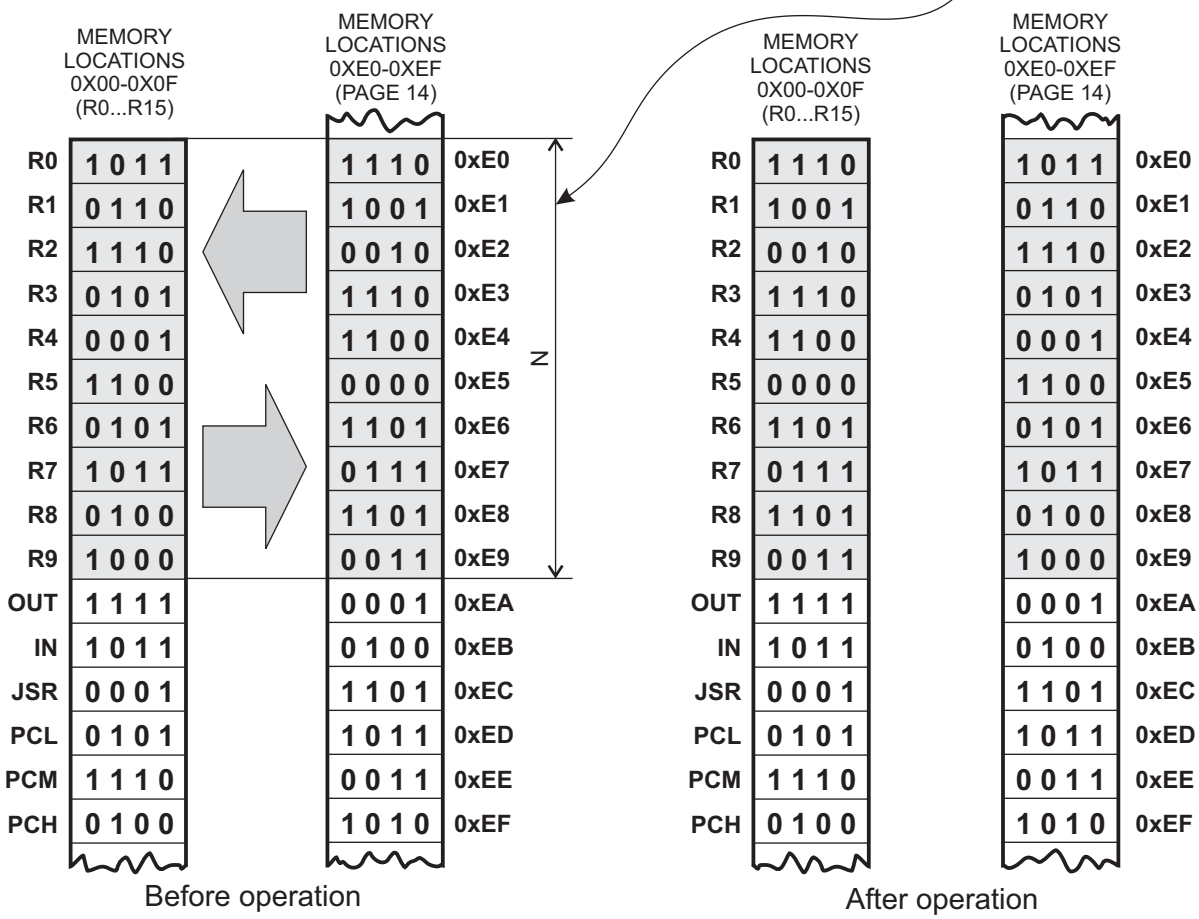
The "0000 1000" bits are the EXR N opcode
The "NNNN" bits are literal N

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.)

EXR N

将第0页的N个主寄存器与第14页（第0x0E页）的相同数量的存储器位置交换

语法： {label} EXR N

操作数： 罗... R15 N ∈ 0. 15（特殊情况：N=0表示N=16）

操作方式： （R0）.(RN)参与（0xE 0）.（0xEN）

产品描述： 将第0页的N个寄存器与第14页的备用组中的寄存器交换。特殊情况：如果N=0，则将交换16个寄存器

受影响的旗帜： 一个也没有。

编码方式：

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

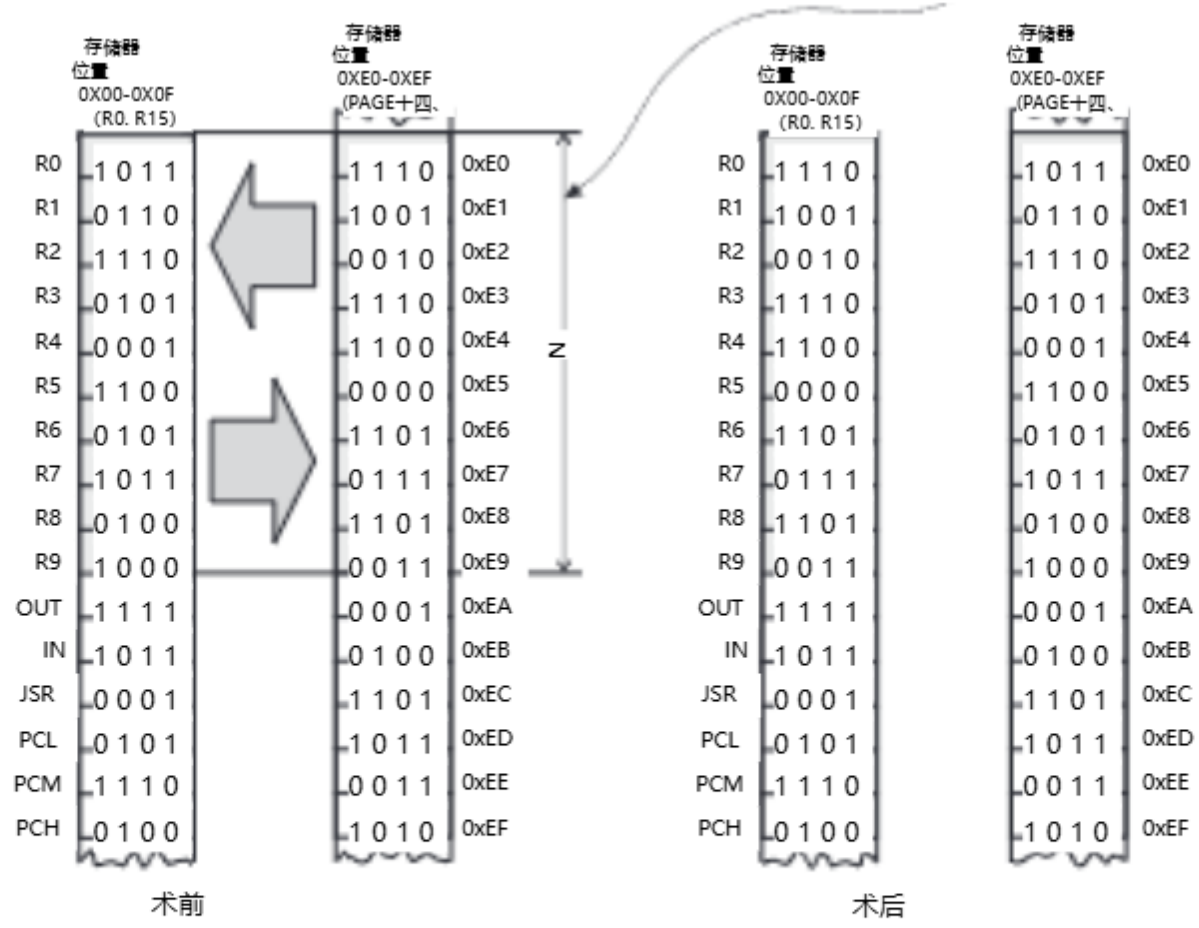
“0000 1000”位是EXR N操作码。“NNNN”位是文字N

范例：

EXR 10

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

操作码0000 10 = EXR N　　文字N



注：寄存器R 0（数据存储位置0x 00）始终与存储位置0xE 0交换，然后，如果N ≥ 1，则与其他寄存器交换。（特殊情况：如果N=0，则将交换16个寄存器。）

BIT RG, M

Test bit M in register RG

Syntax: {label} BIT RG, M

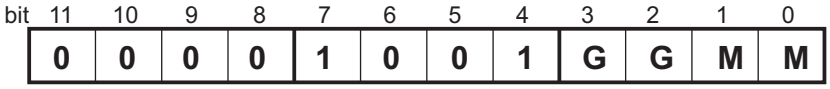
Operands: RG ∈ [R0 | R1 | R2 | RS]
N ∈ 0 | 1 | 2 | 3 or 0 | 1 | 2 | S

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.

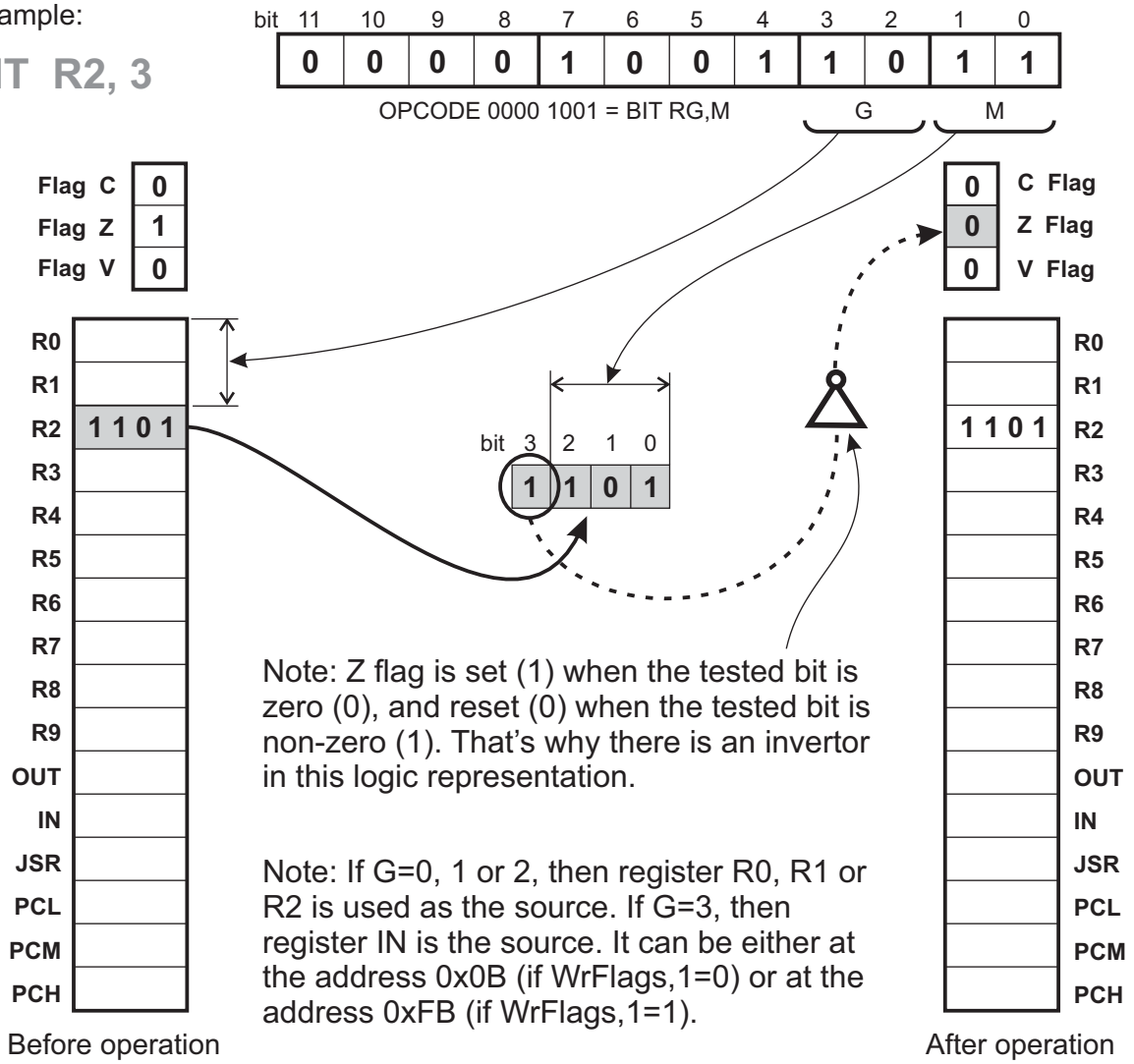
Encoding:



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

Example:

BIT R2, 3



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).

钻头RG，M —在RG register中测试bit M

语法：{label} BIT RG，M

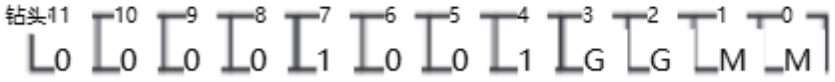
操作数：RG ∈ [R0| R1| R2| RS] N ∈ 0| 1 | 2 |3或0| 1 | 2 |S

操作方式：Z ← -<bit>

产品描述：测试RG寻址的寄存器中的位N，并更新Z标志。

受影响的旗帜：国旗C不受影响。
如果测试位=0，则设置Z标志。否则，重置Z。

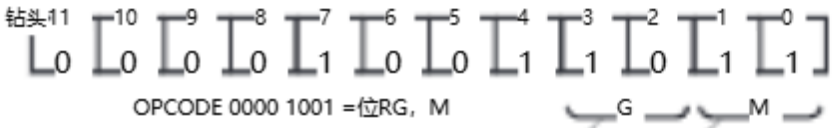
编码方式：

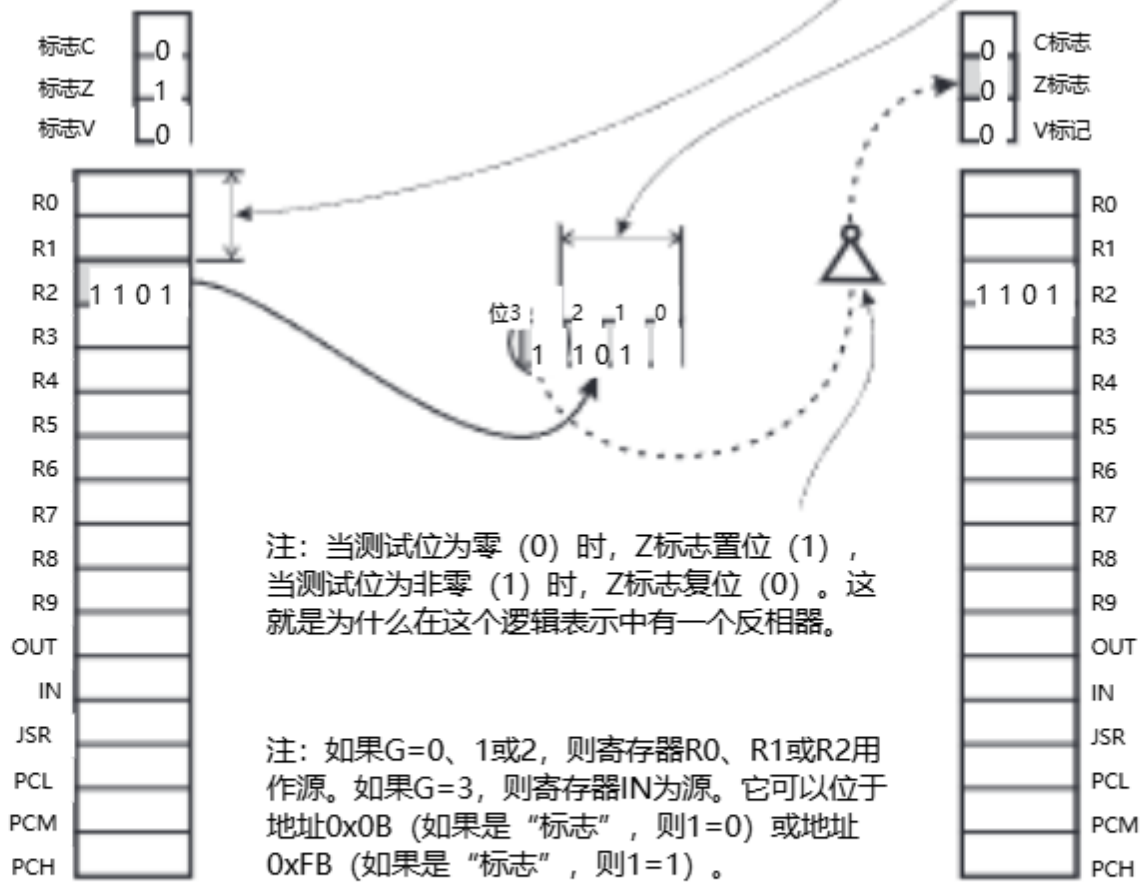


“0000 1001”位是BIT RG，M操作码。“NNNN”位是文字N

范例：

BIT R2，3





注：当测试位为零（0）时，Z标志置位（1），当测试位为非零（1）时，Z标志复位（0）。这就是为什么在这个逻辑表示中有一个反相器。

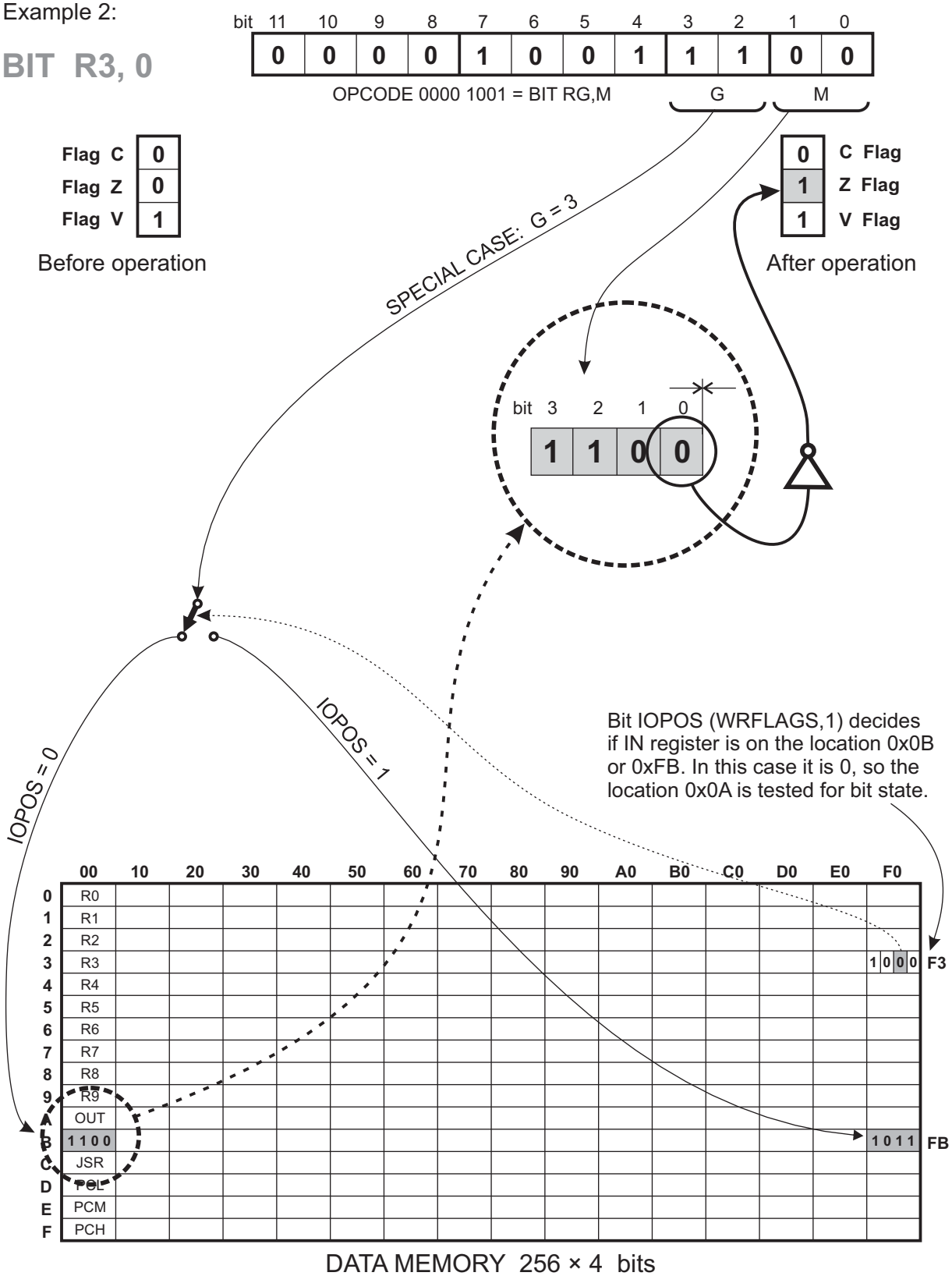
注：如果G=0、1或2，则寄存器R0、R1或R2用作源。如果G=3，则寄存器IN为源。它可以位于地址0x0B（如果是“标志”，则1=0）或地址0xFB（如果是“标志”，则1=1）。

BIT RG,M

Test bit M in register RG (CONTINUED)

Example 2:

BIT R3, 0



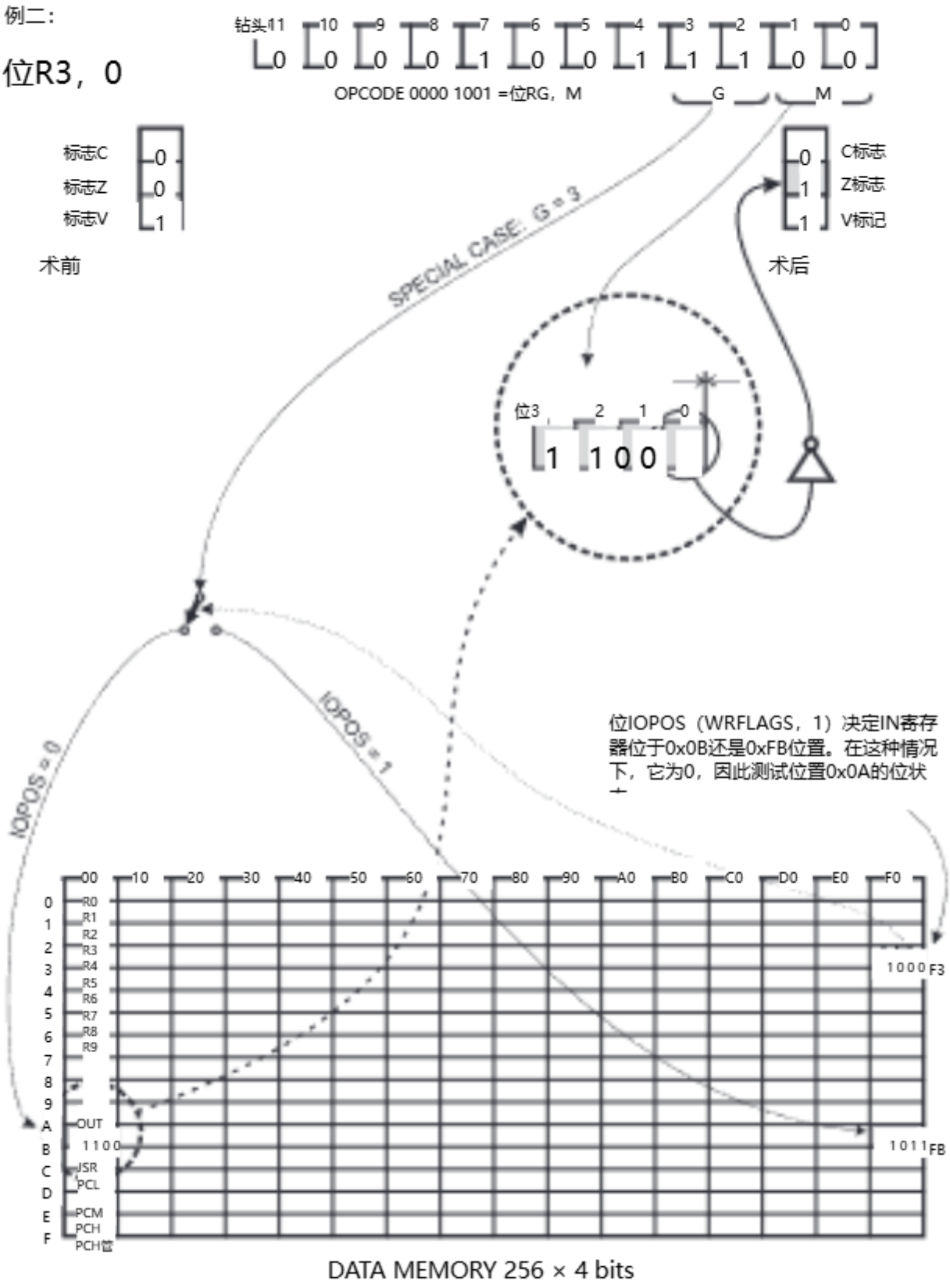
DATA MEMORY 256 × 4 bits

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

钻头RG，M —RG register中的测试位M（continued）

例二：

位R3，0



DATA MEMORY 256 × 4 bits

注意：寄存器WRFLAGS在手册《特殊功能寄存器》中有介绍

BSET RG,M

Set bit M in register RG

Syntax: {label} BSET RG, M

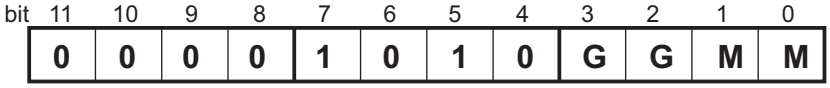
Operands: RG ∈ [R0 | R1 | R2 | RS]
N ∈ 0 | 1 | 2 | 3 or 0 | 1 | 2 | S

Operation: <bit> ← 1

Description: Set bit N in register addressed by RG.

Flags affected: None.

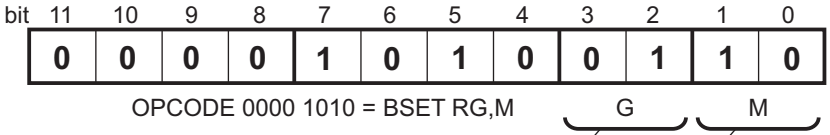
Encoding:

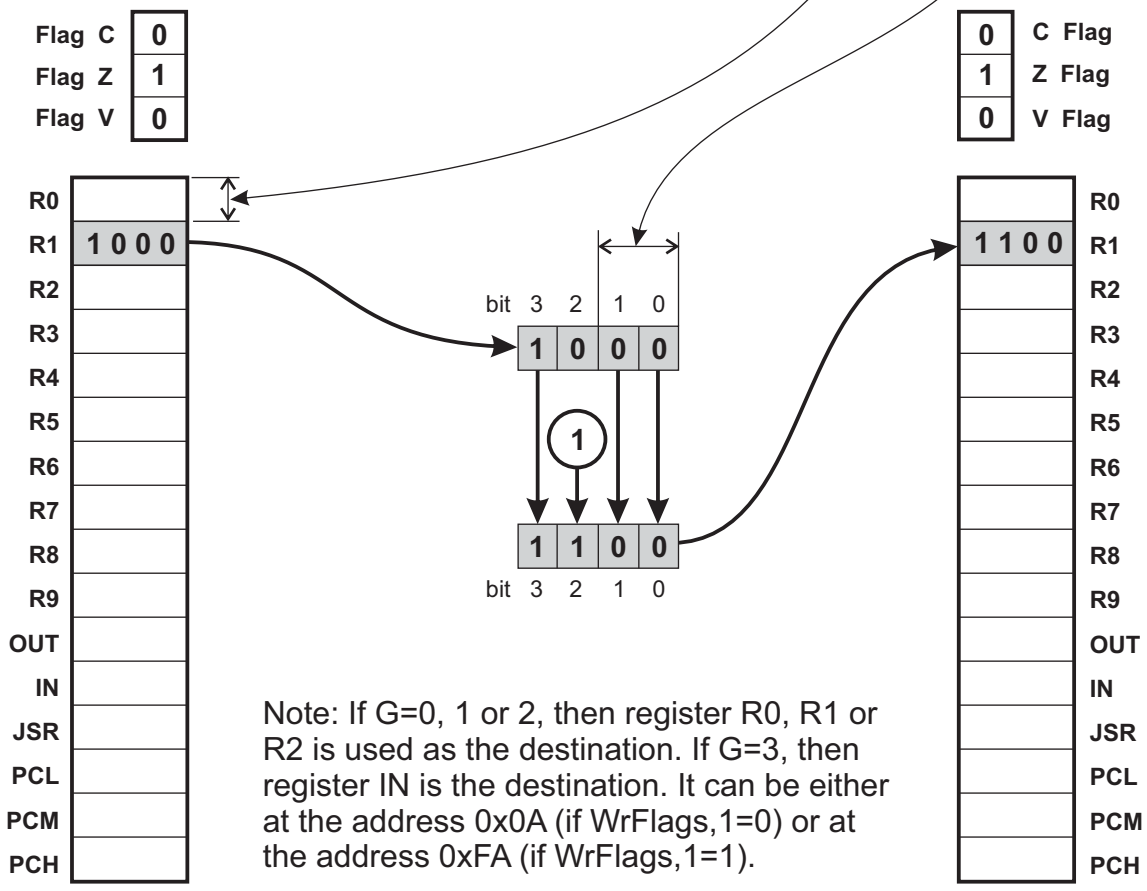


The "0000 1010" bits are the BSET RG,M opcode
The "NNNN" bits are literal N

Example:

BSET R1, 2





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注册RG

语法： {label} BSET RG，M

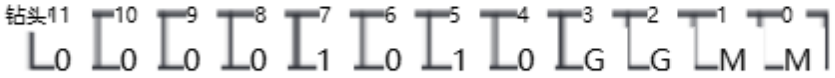
操作数： RG ∈ [R0| R1| R2| RS] N ∈ 0| 1 | 2 |3或0| 1 | 2 |S

操作方式： <bit> ← 1

产品描述： 设置RG寻址的寄存器中的位N。

受影响的旗帜： 一个也没有。

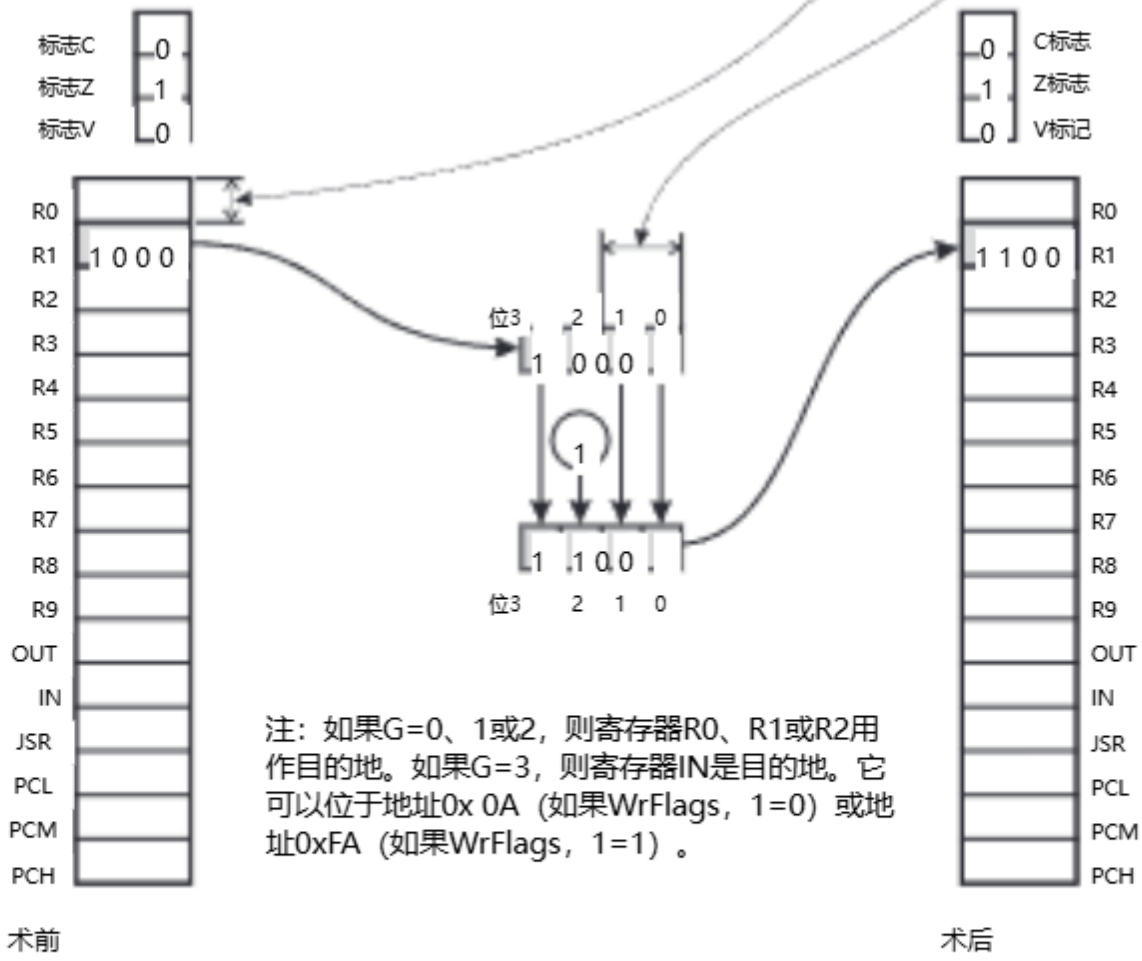
编码方式：



“0000 1010”位是BSET RG，M操作码。“NNNN”位是文字N

范例：

BSET R1.2



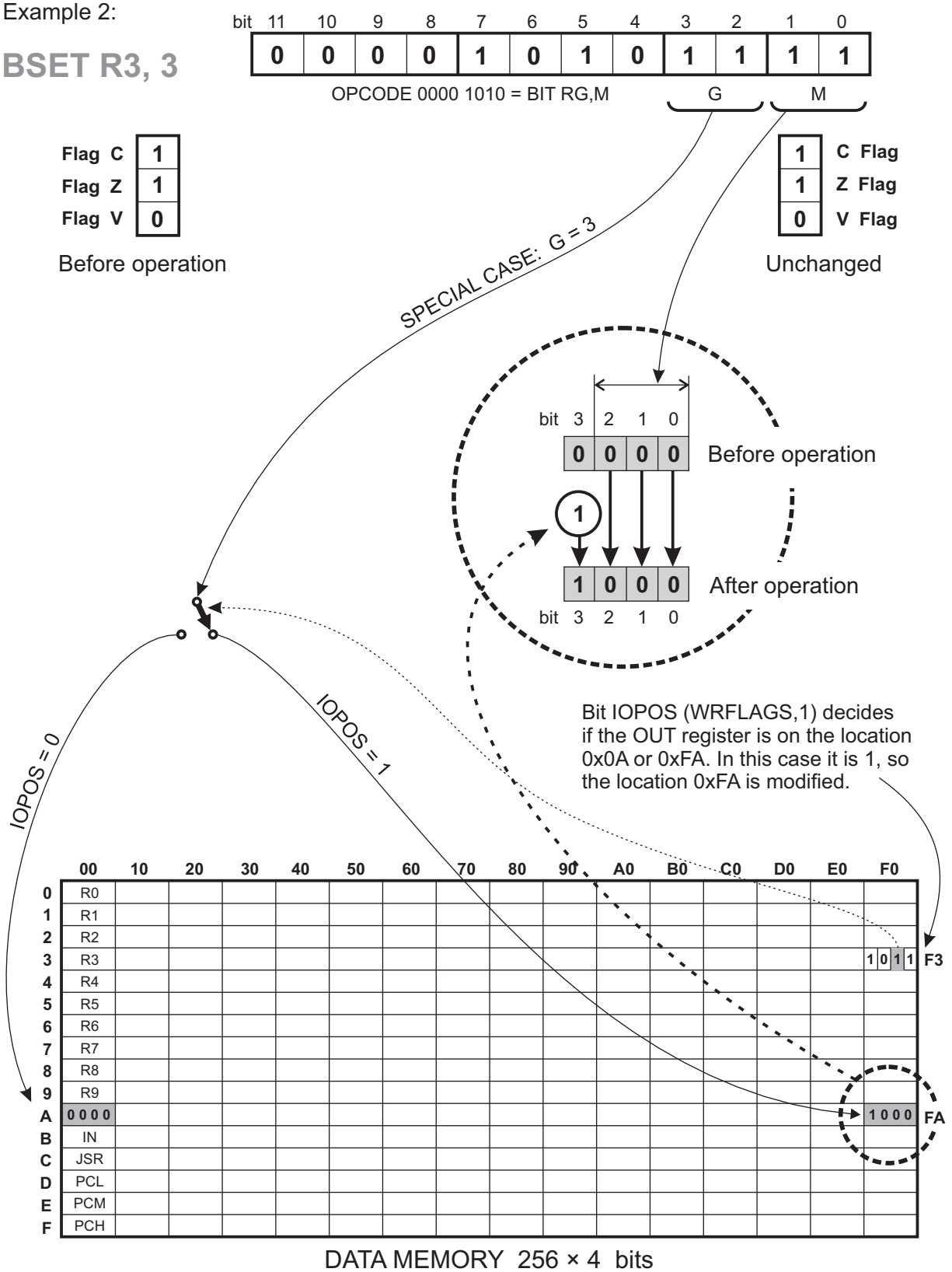
注：如果G=0、1或2，则寄存器R0、R1或R2用作目的地。如果G=3，则寄存器IN是目的地。它可以位于地址0x 0A（如果WrFlags，1=0）或地址0xFA（如果WrFlags，1=1）。

BSET RG,M

Set bit M in register RG (CONTINUED)

Example 2:

BSET R3, 3



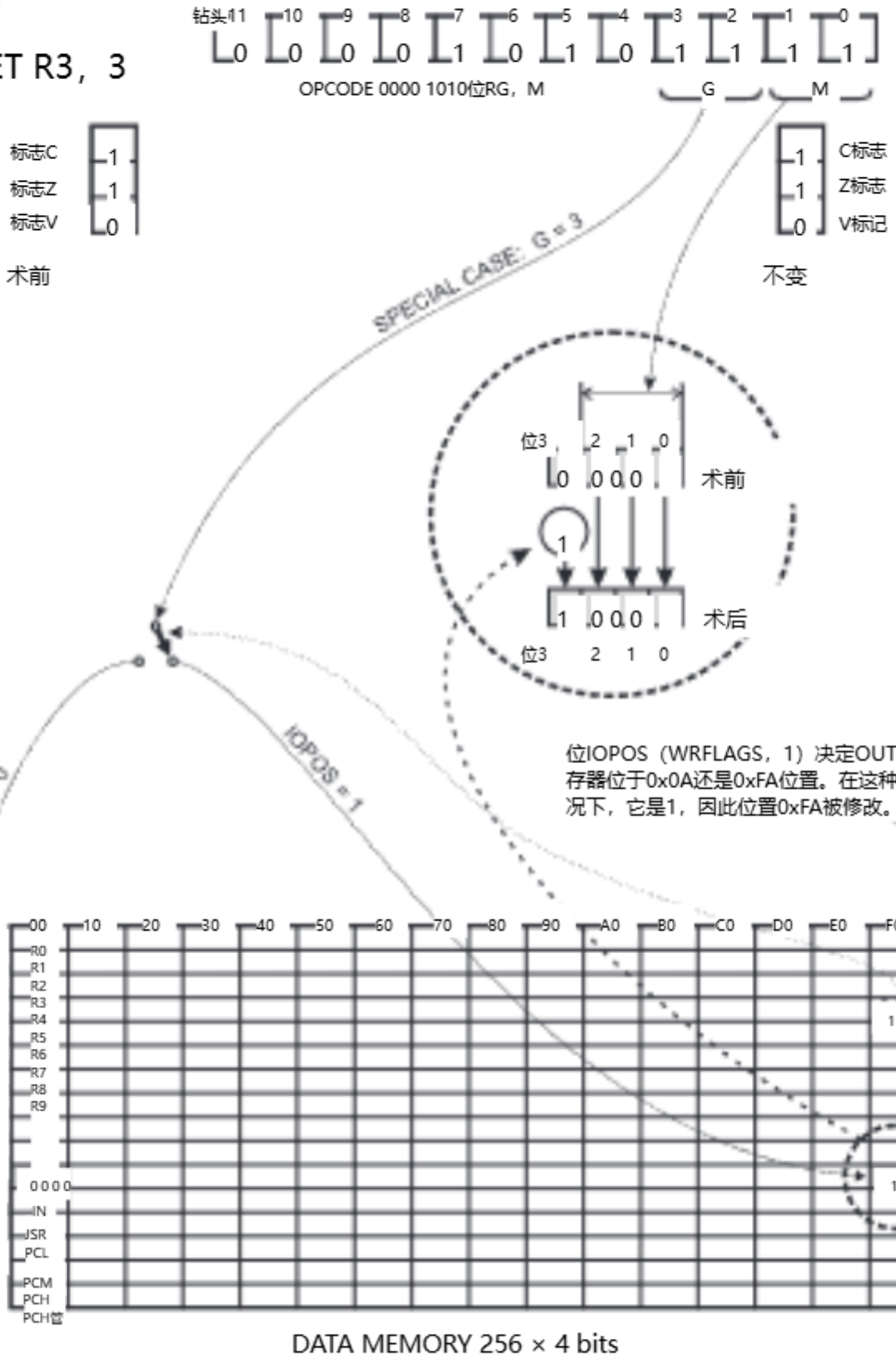
DATA MEMORY 256 × 4 bits

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

BSET RG，M —Set bit M in register RG（continued）（注册号）

例二：

BSET R3，3



DATA MEMORY 256 × 4 bits

注意：寄存器WRFLAGS在手册《特殊功能寄存器》中有介绍

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

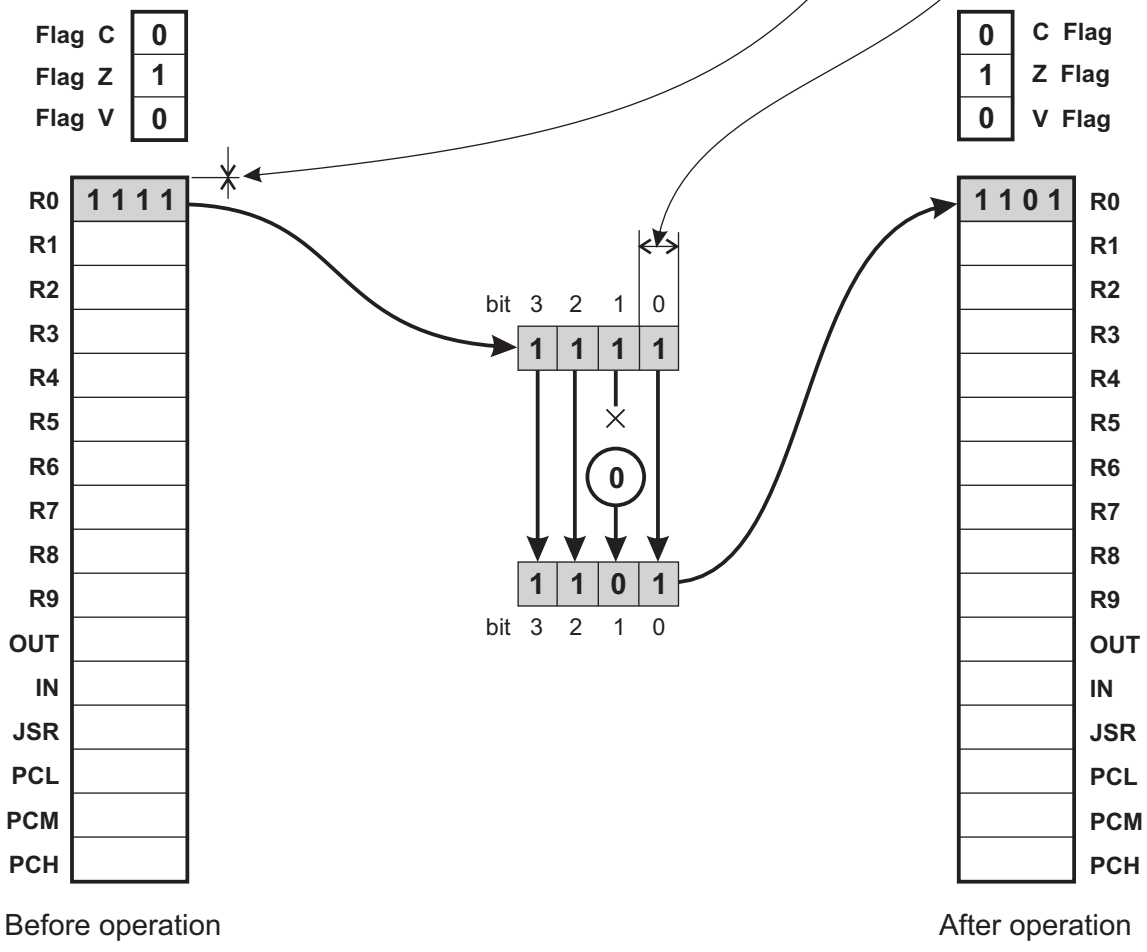
The "0000 1011" bits are the BCLR RG,M opcode
The "NNNN" bits are literal N

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

OPCODE 0000 1011 = BCLR RG,M — G — M



BagrRG，M

清除寄存器RG中的位M

语法：{label}贝鲁 RG，M

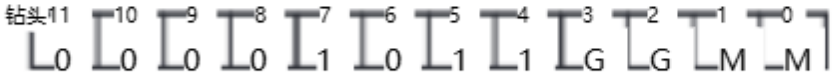
操作数：RG ∈ [R0| R1| R2| RS] N ∈ 0| 1 | 2 |3或0| 1 | 2 |S

操作方式：<bit> ← 0

产品描述：清除RG寻址的寄存器中的位#N。

受影响的旗帜：一个也没有。

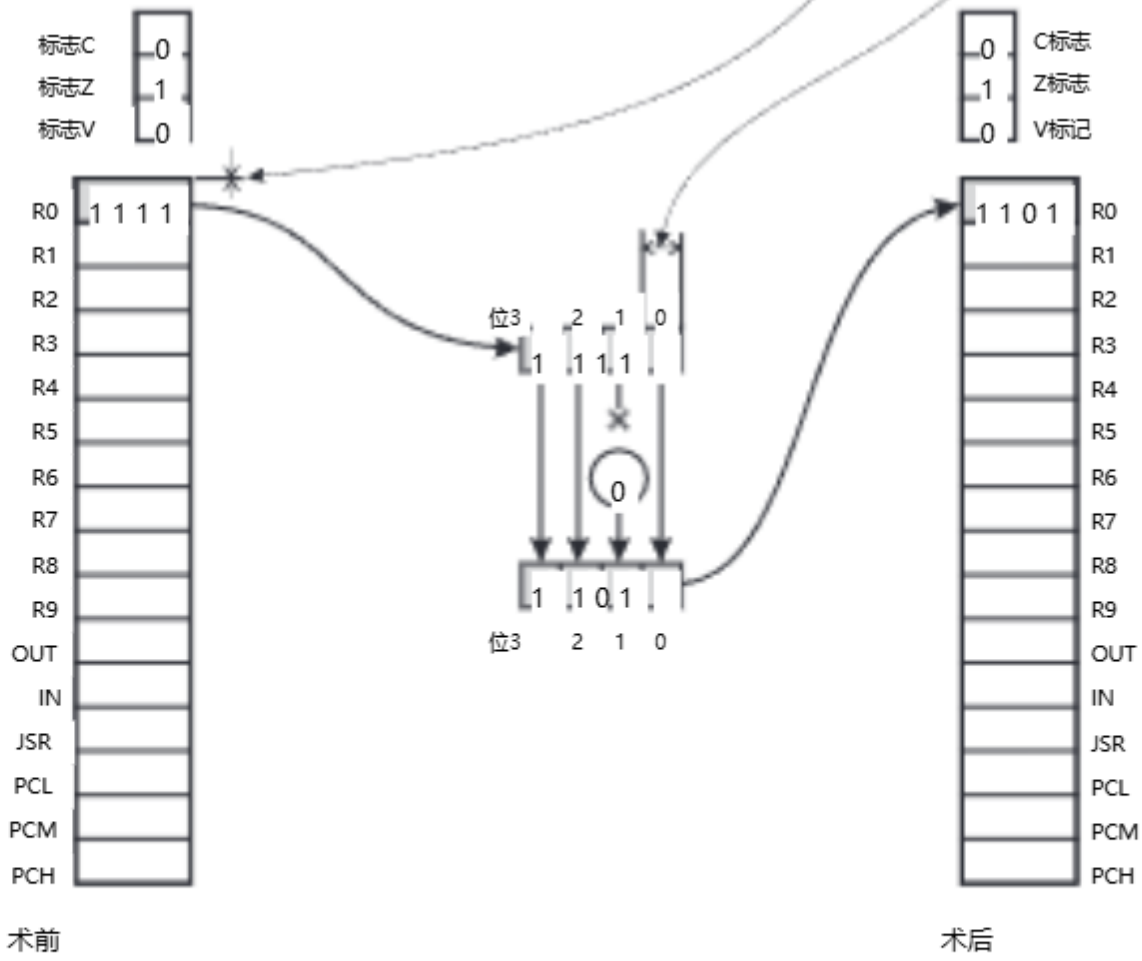
编码方式：



“0000 1011”位是BARNRG，M操作码。“NNNN”位是文字N

范例：

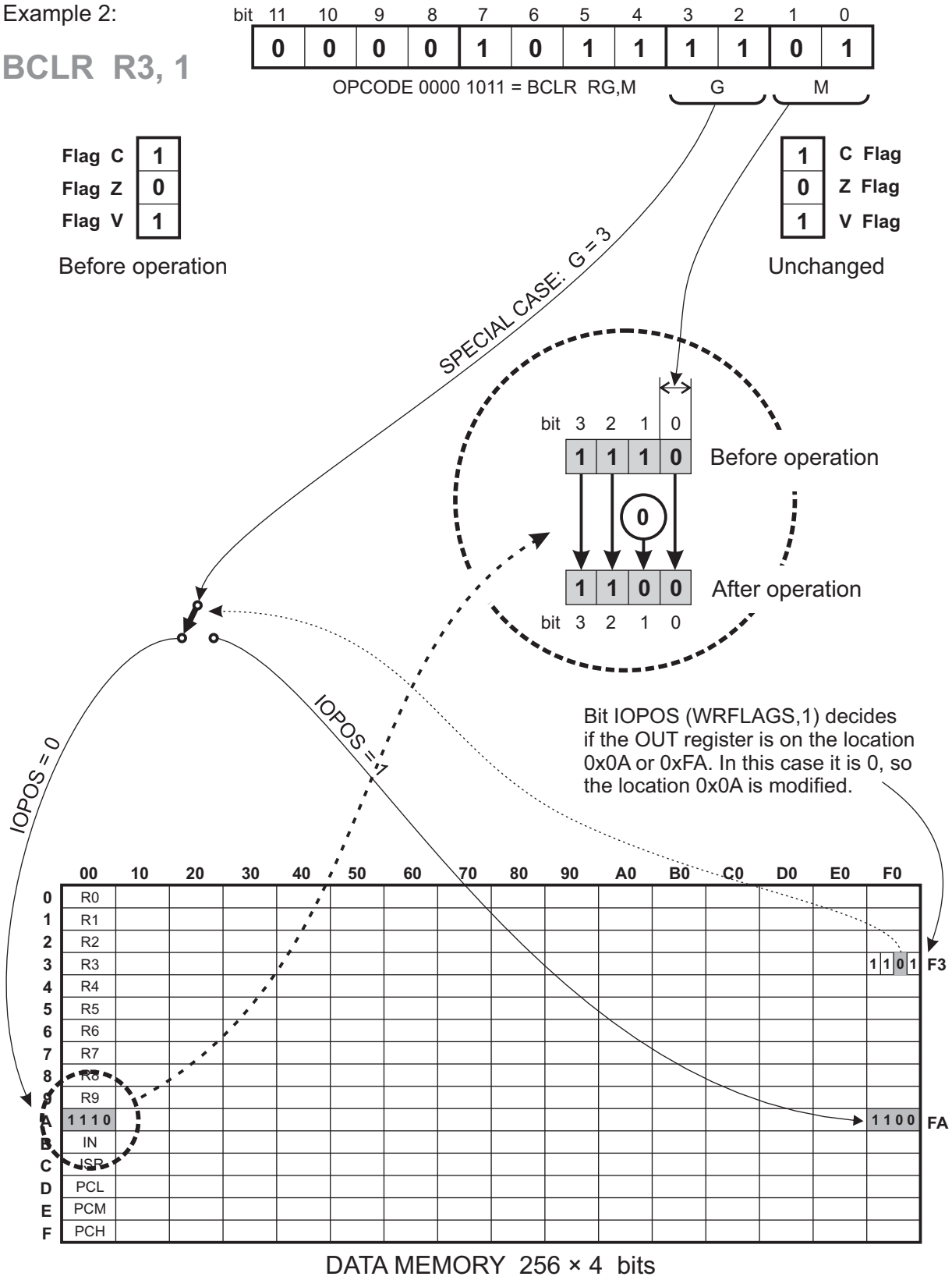
BARR0，1



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

Example 2:

BCLR R3, 1



DATA MEMORY 256 × 4 bits

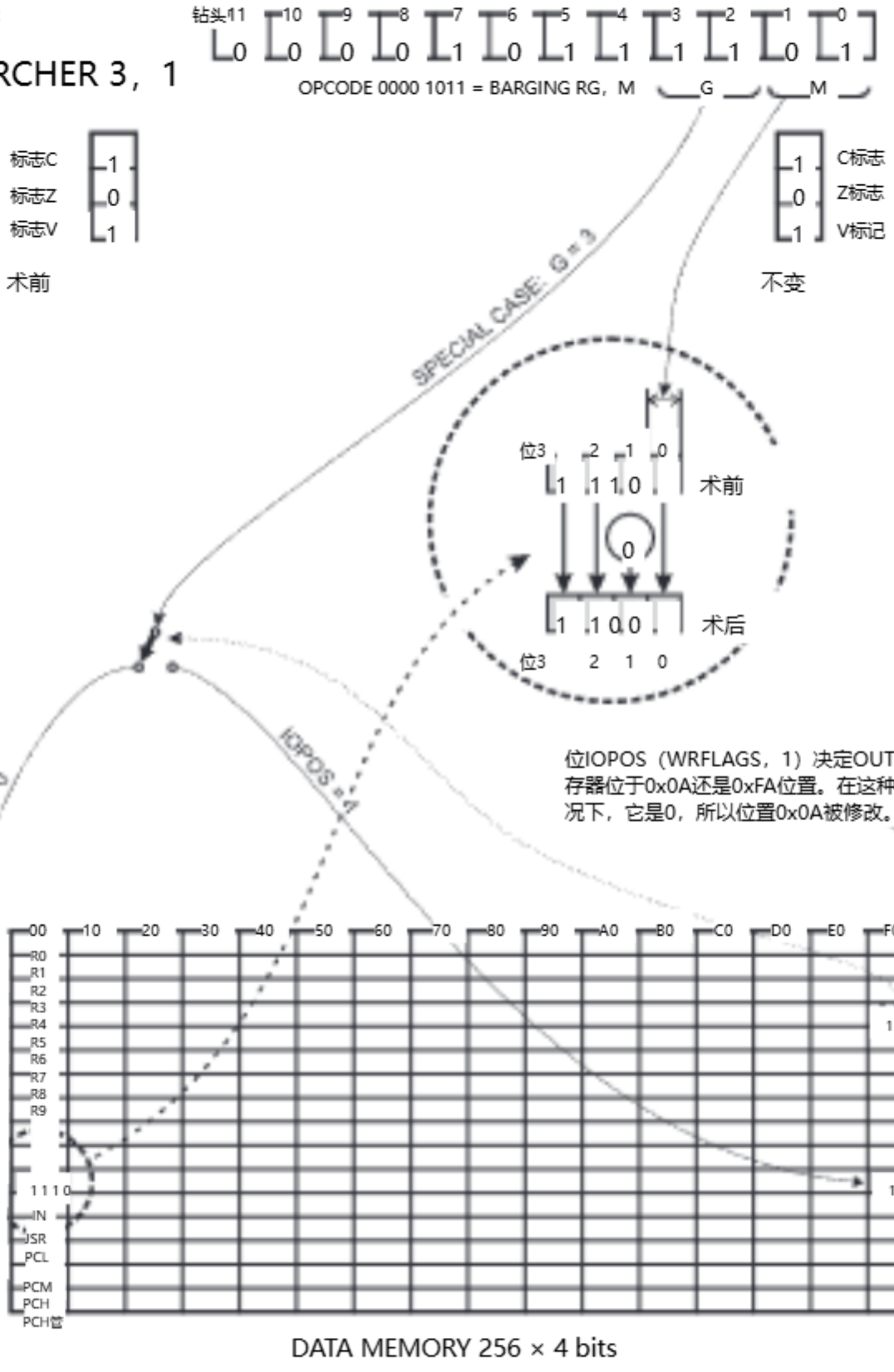
Note: Registers WRFLAGS is described in the manual Special Function Registers

BagrRG，M

清除寄存器RG中的位M（续）

例二：

BARCHER 3，1



DATA MEMORY 256 × 4 bits

注意：寄存器WRFLAGS在手册《特殊功能寄存器》中有介绍

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

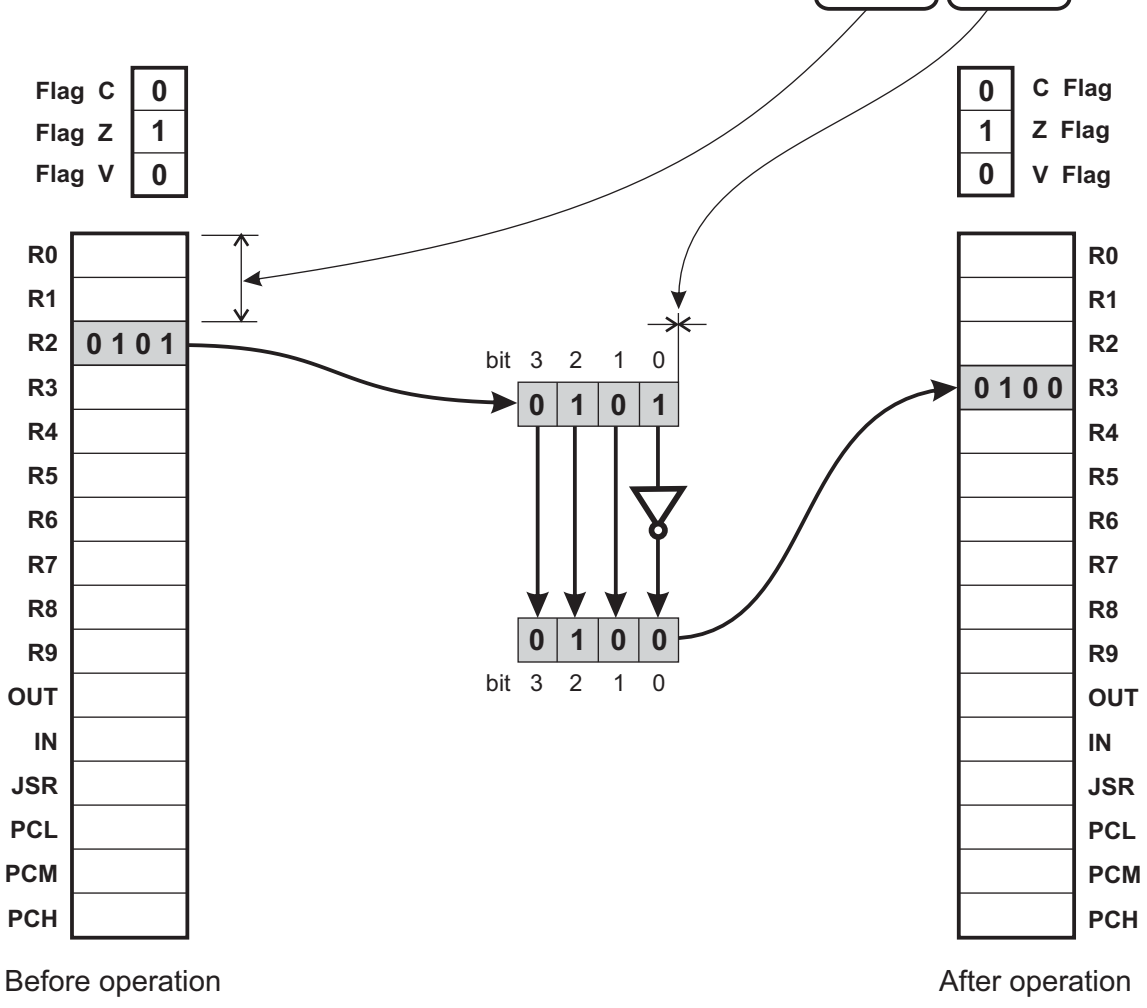
The "0000 1100" bits are the BTG RG,M opcode
The "NNNN" bits are literal N

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 G M



BTG RG，M ——切换寄存器RG中的位M

语法： {label} BTG RG，M

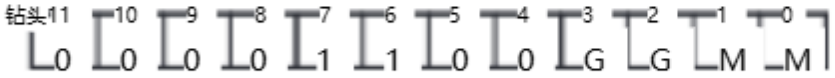
操作数： RG ∈ [R0| R1| R2| RS] N ∈ 0| 1 | 2 |3或0| 1 | 2 |S

操作方式： ← -

产品描述： 反转RG寻址的寄存器中的位#N。

受影响的旗帜： 一个也没有。

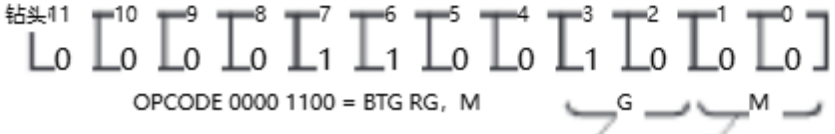
编码方式：

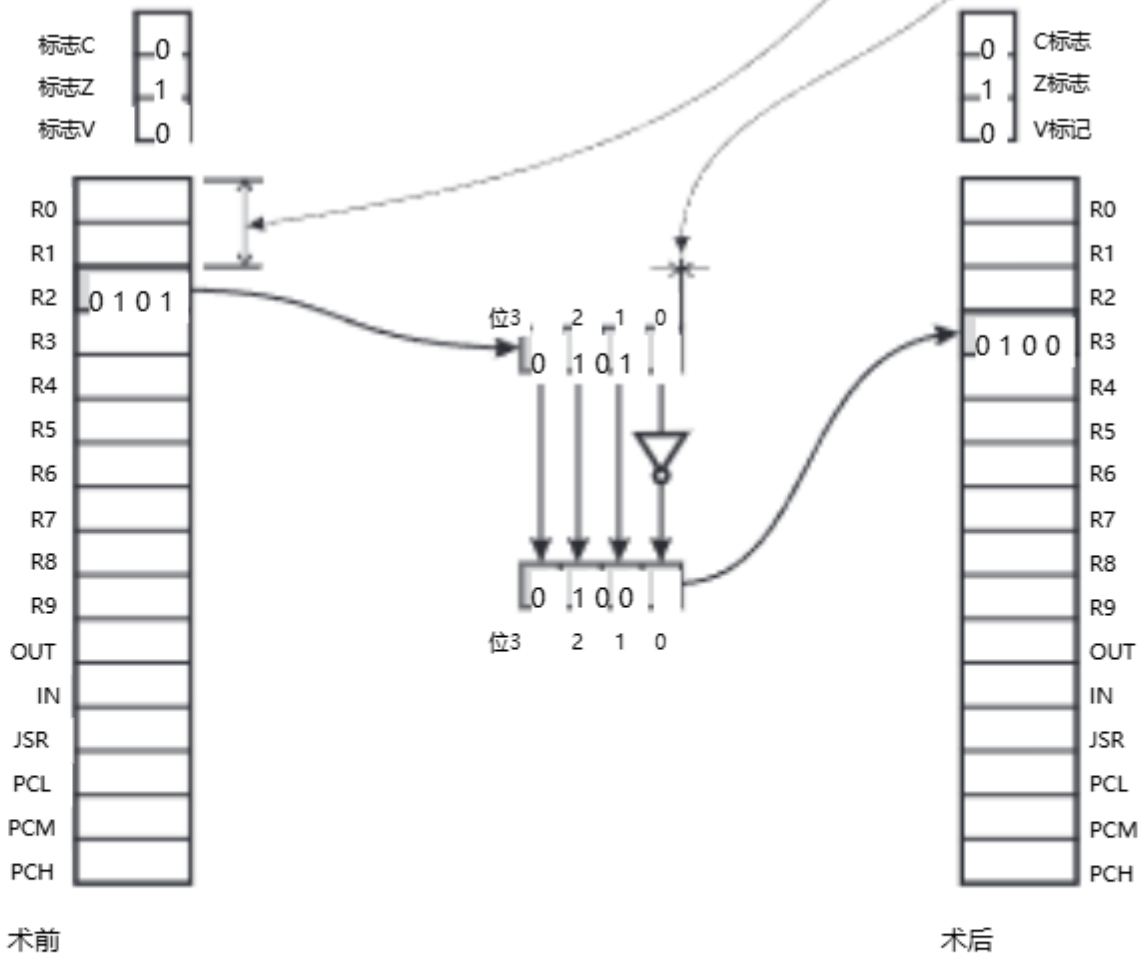


“0000 1100”位是BTG RG，M操作码。 “NNNN”位是文字N

范例：

BTG R2，0



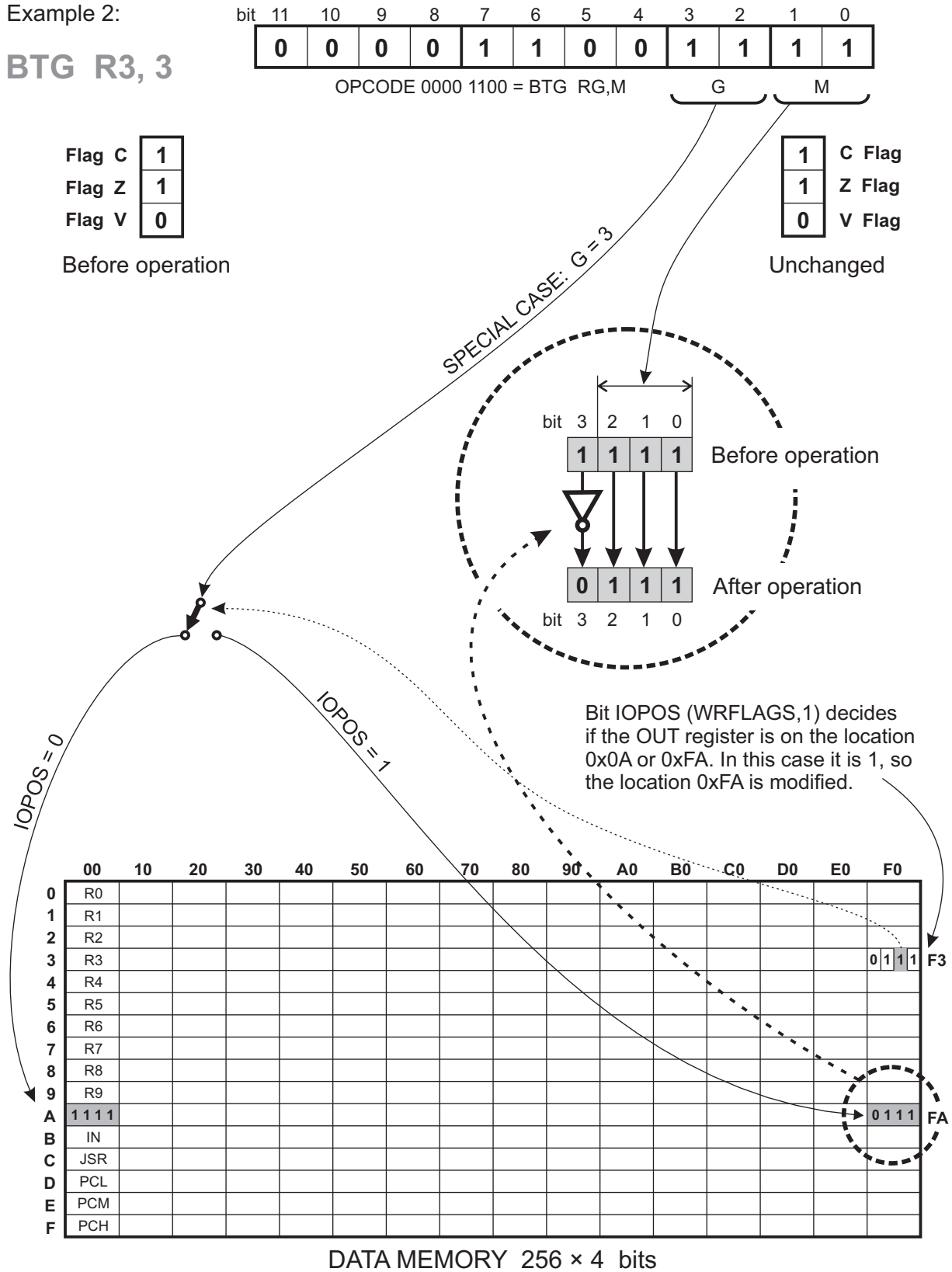


BTG RG,M

Toggle bit M in register RG (CONTINUED)

Example 2:

BTG R3, 3



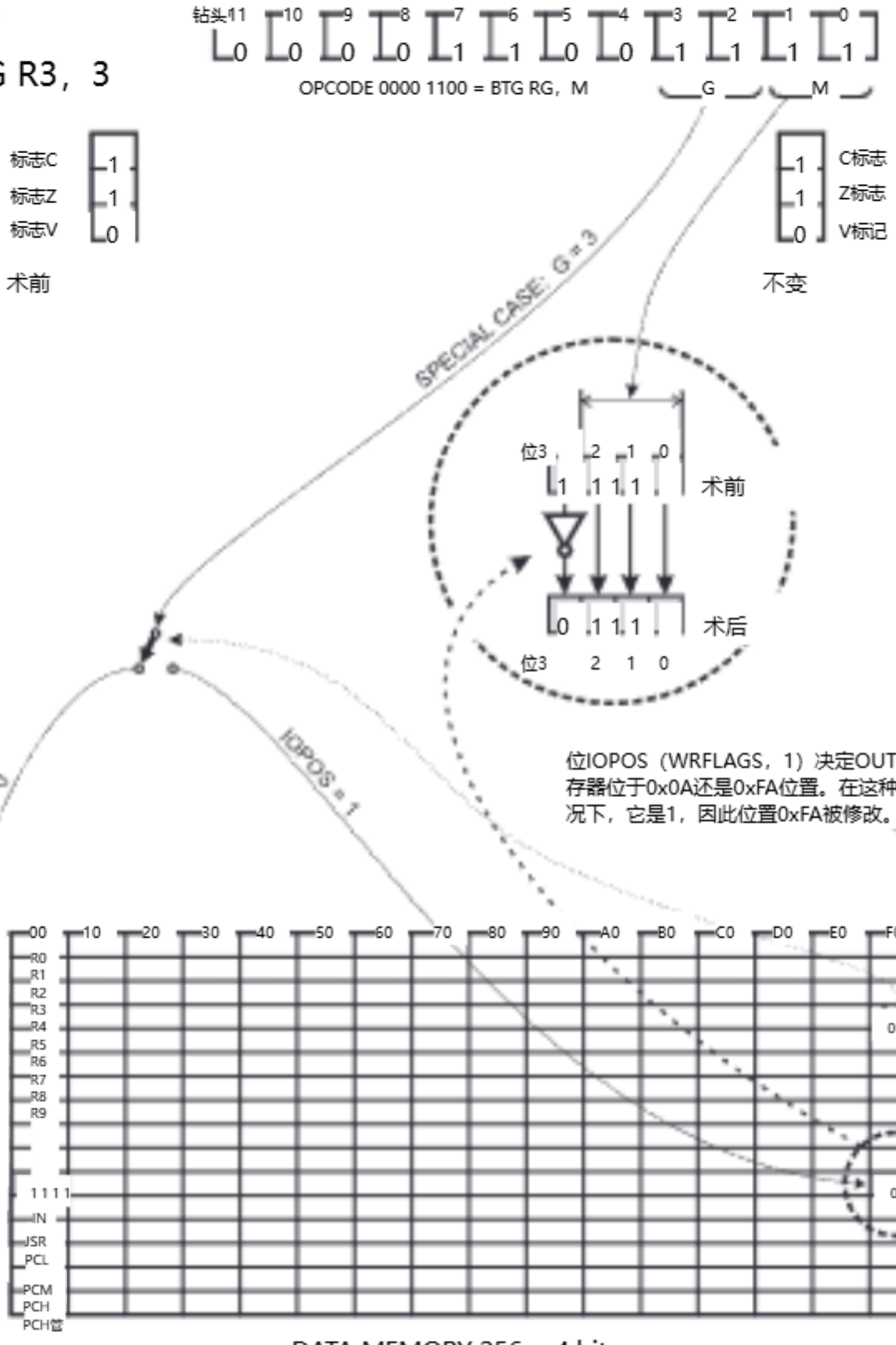
DATA MEMORY 256 × 4 bits

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

BTG RG，M ——在寄存器RG中切换位M（续）

例二：

BTG R3，3



DATA MEMORY 256 × 4 bits

注意：寄存器WRFLAGS在手册《特殊功能寄存器》中有介绍

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

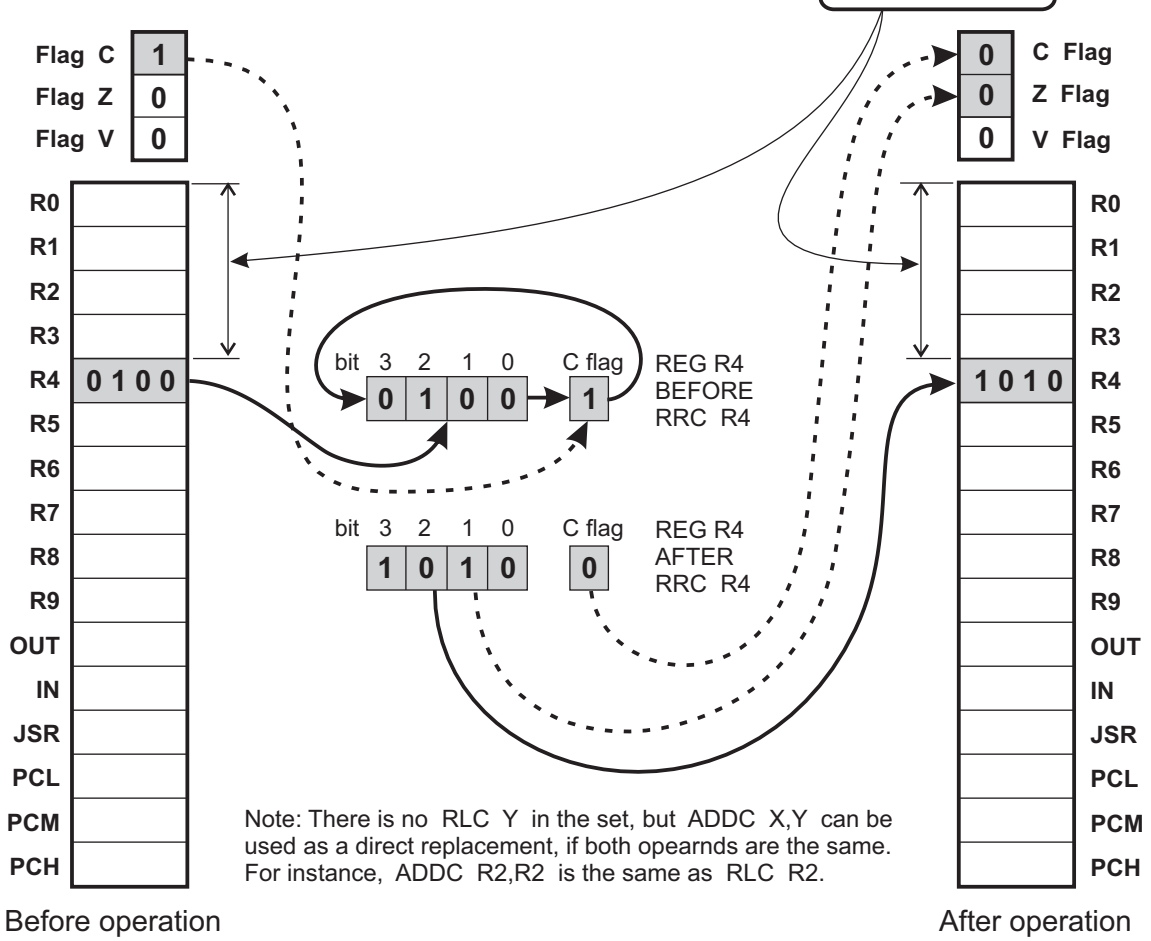
Example:

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.

Before operation

After operation

RRC RY

向右旋转通过携带寄存器RY

语法：{label} RRC RY

操作数：RY ∈ [R0. R15]

操作方式：(C)← (RY0)， (RY3) ← (C)， (RY2) ← (RY3)， (RY1) ← (RY2)， (RY0) ← (RY1)

产品描述：通过进位标志将寄存器Y的内容向右旋转一位，并将结果放回寄存器RY。进位标志被移位到寄存器RY的最高有效位，然后用寄存器RY的最低有效位重写。

受影响的旗帜：寄存器RY的位0被复制到标志C。
如果操作后结果=0000，则设置Z。否则，重置Z

编码方式：

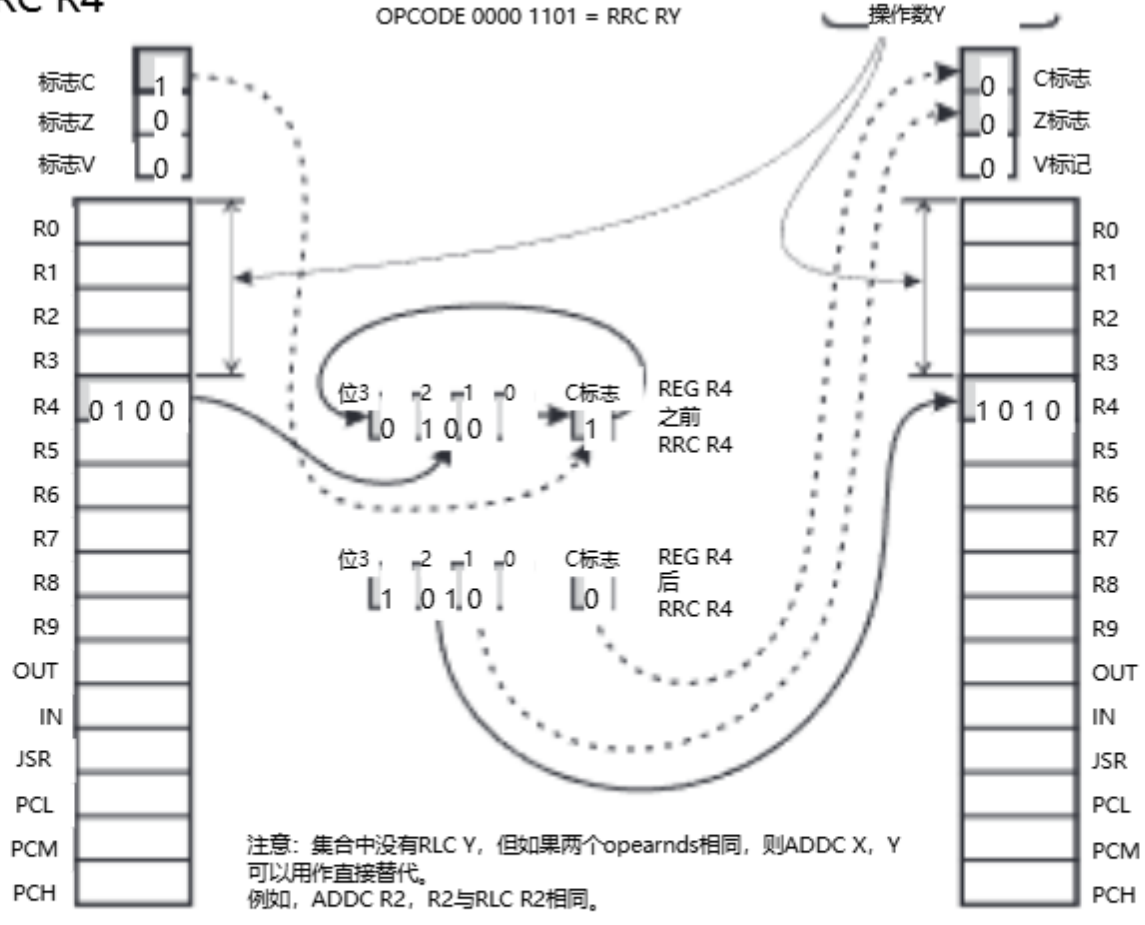
钻头11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	Y	Y	Y	Y

“0000 1101”位是RRC RY操作码。“YYYY”位选择操作数Y

范例：

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

RRC R4



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

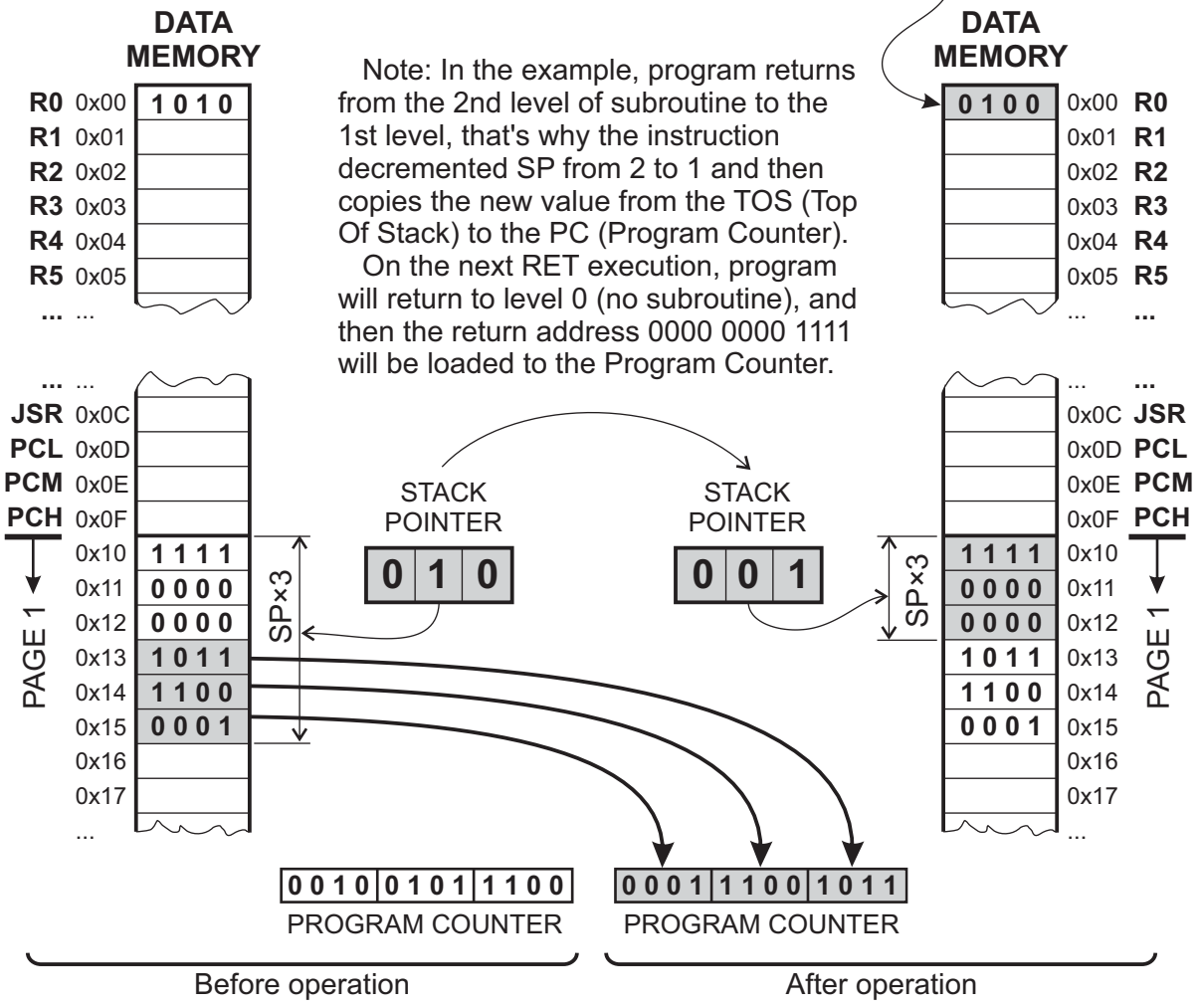
Example:

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.

RET R0，N ——从子例程返回，文字在寄存器R0中

语法： {label} RET R0,N

操作数： R0
N ∈ 0. 15

操作方式： (R0)← N
← SP—1 PC系列<11 id=5>← （SP × 3 + 2），
（SP × 3 + 1），（SP × 3）

产品描述： 用文字值N加载R 0并从堆栈中弹出PC

受影响的旗帜： 一个也没有。

编码方式：

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

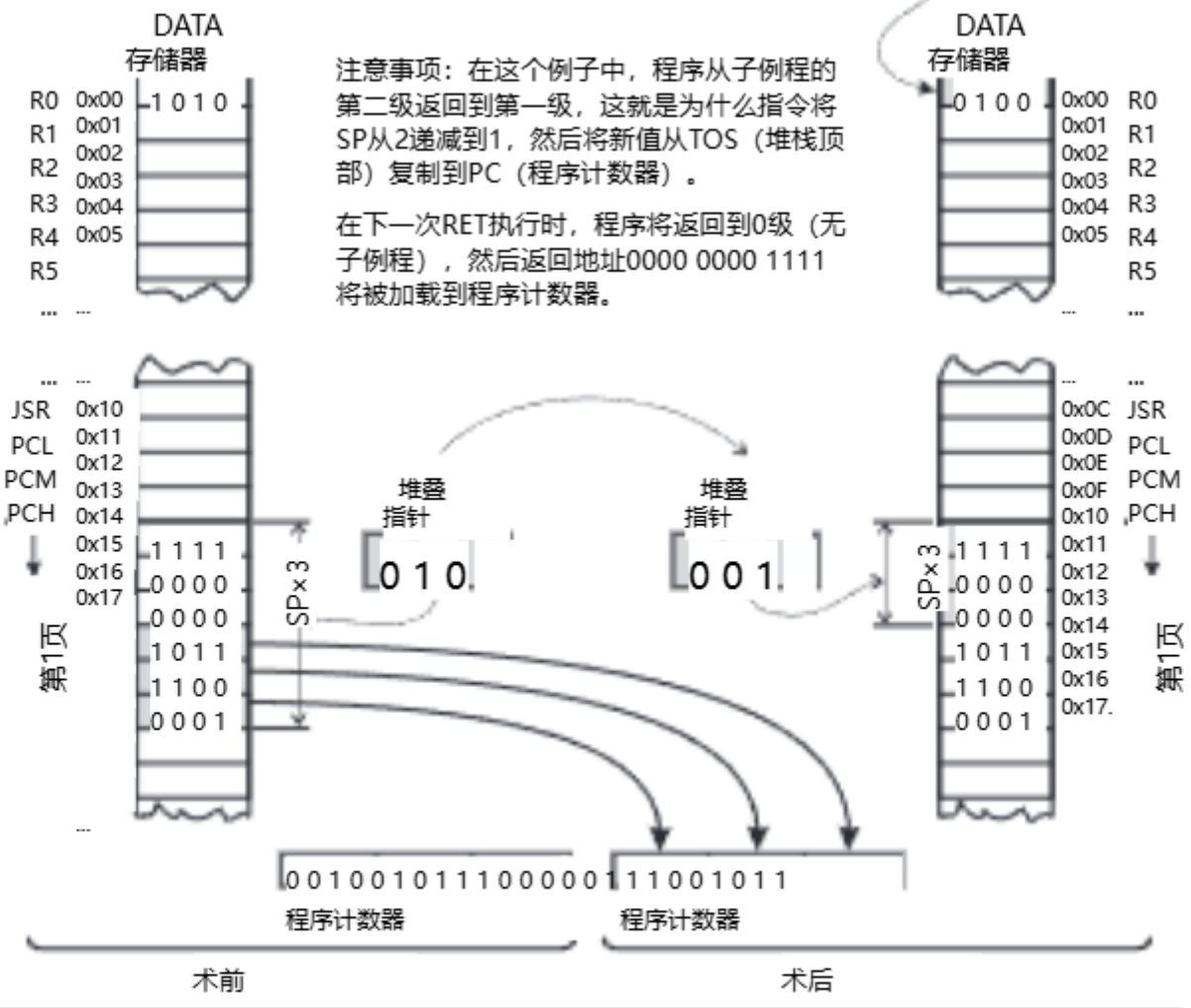
“0000 1110”位是RET操作码“NNNN”位
是文字N

范例：

RET R0，#4

钻头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　　文字N



注意事项：在这个例子中，程序从子例程的第二级返回到第一级，这就是为什么指令将SP从2递减到1，然后将新值从TOS（堆栈顶部）复制到PC（程序计数器）。

在下一次RET执行时，程序将返回到0级（无子例程），然后返回地址0000 0000 1111将被加载到程序计数器。

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

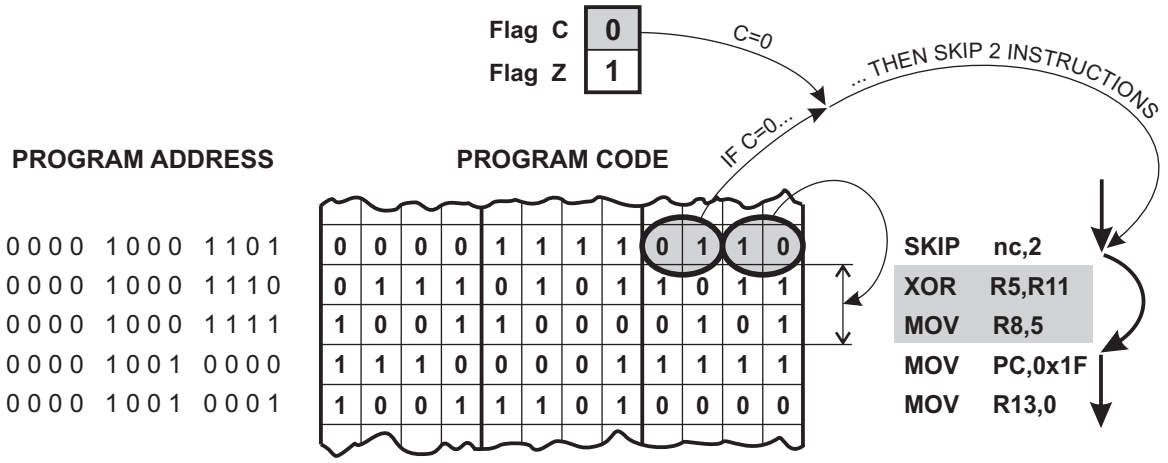
STEP COUNT M	SKIP INSTRUCTIONS
0 0	4
0 1	1
1 0	2
1 1	3

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.

跳过F、M

有条件地跳过下M条指令

语法： {label}跳过 z| NZ| C| nc，M

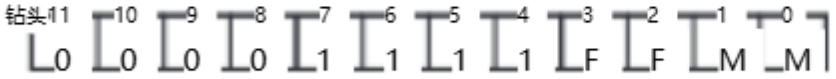
操作数： F ∈ z| NZ| C| nc M ∈ 0. 3（特殊情况：M=0表示M=4）

操作方式： 如果条件=真，则（PC）←（PC+M）

产品描述： 如果condition=true，则跳过接下来的M条指令。
特殊情况：如果M=0，则跳过4条指令。

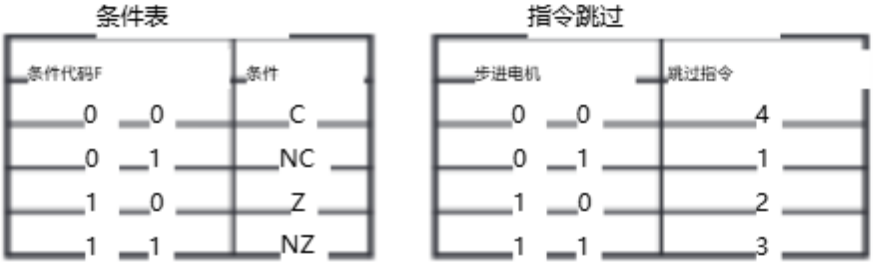
受影响的旗帜： 一个也没有。

编码方式：



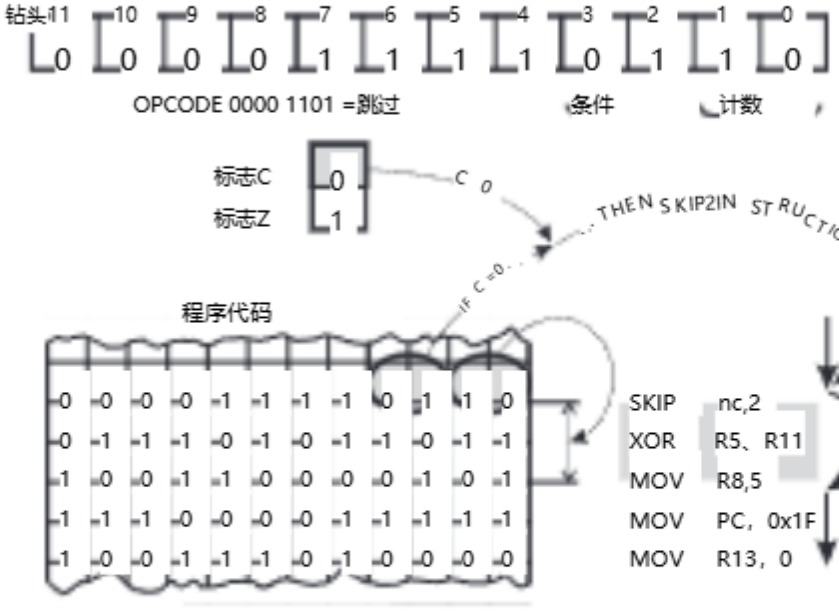
“0000 1111”位是SKIP操作码。“FF”位是条件码（参见条件表）。“MM”位是跳过步数（“00”=“4”）

条件/跳过编码：



范例：

SKIP nc，#2



注意事项：在本例中，标志C =0，因此指令SKIP nc，2导致程序跳过地址0000 1000 1110和0000 1000 1111上的两条指令。程序继续在地址0000 1001 0000执行。