INSTRUCTION SET

Revision 2 Oct. 12. 2022

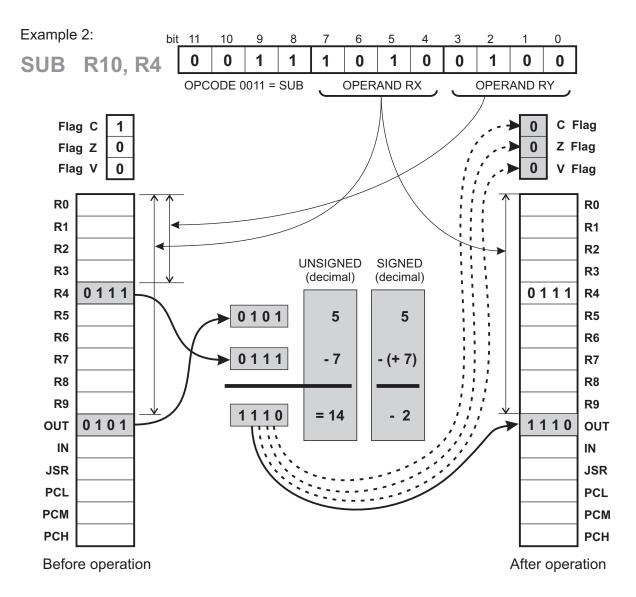
CODING

		MNEMONIC						CODE										
	OPC			FLAGS			OPCODE			OPERAND X OPCODE			OPERAND Y					
	1	ADD	RX,RY	٧	Z	С	0	0	0	1	X	X	X	X	Y	Y	Υ	Υ
•	2	ADC	RX,RY	٧	Z	С	0	0	1	0	X	X	X	X	Y	Y	Y	Υ
•	3	SUB	RX,RY	٧	Z	С	0	0	1	1	X	X	X	X	Υ	Y	Υ	Υ
•	4	SBB	RX,RY	٧	Z	С	0	1	0	0	X	X	X	X	Y	Y	Y	Υ
•	5	OR	RX,RY		Z		0	1	0	1	X	X	X	X	Y	Υ	Y	Υ
•	6	AND	RX,RY		Z		0	1	1	0	X	X	X	X	Y	Y	Y	Υ
•	7	XOR	RX,RY		Z		0	1	1	1	X	X	X	X	Y	Y	Y	Υ
•	8	MOV	RX,RY				1	0	0	0	X	X	X	X	Y	Y	Υ	Υ
	9	MOV	RX,#N				1	0	0	1	X	X	X	X	N	N	N	N
	Α	MOV	[XY],R0				1	0	1	0	X	X	X	X	Y	Y	Υ	Υ
	В	MOV	R0,[XY]				1	0	1	1	X	X	X	X	Y	Y	Υ	Υ
	С	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
	Е	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	СР	R0,N	٧	Z	С	0	0	0	0	0	0	0	0	N	N	N	N
	01	ADD	R0,N	٧	Z	С	0	0	0	0	0	0	0	1	N	N	N	N
•	02	INC	RY		Z	С	0	0	0	0	0	0	1	0	Υ	Y	Y	Υ
	03	DEC	RY		Z	С	0	0	0	0	0	0	1	1	Y	Y	Y	Υ
	04	DSZ	RY				0	0	0	0	0	1	0	0	Y	Y	Y	Υ
	05	OR	R0,N		Z	С	0	0	0	0	0	1	0	1	N	N	N	N
	06	AND	R0,N		Z	С	0	0	0	0	0	1	1	0	N	Ν	N	N
	07	XOR	R0,N		Z	С	0	0	0	0	0	1	1	1	N	Ν	N	N
	08	EXR	N				0	0	0	0	1	0	0	0	N	N	N	N
	09	BIT	RG,M		Z	С	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	С	0	0	0	0	1	1	0	1	Υ	Y	Y	Υ
	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".

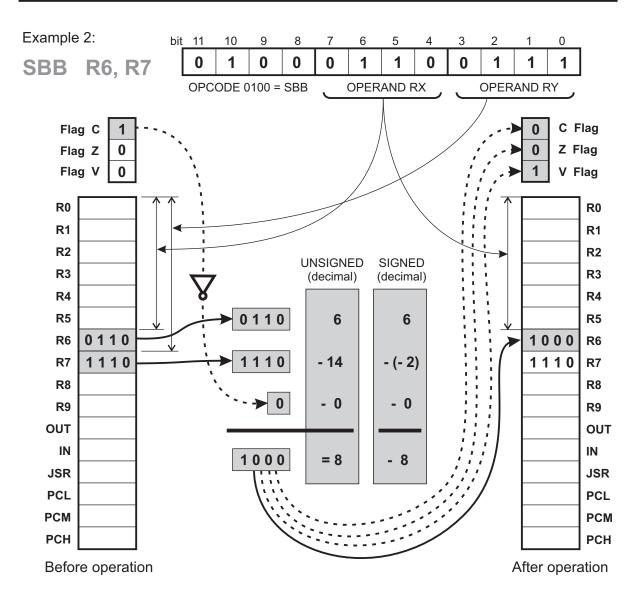
{label} ADD RX, Syntax: $RX \in [R0...R15]$ Operands: RY ∈ [R0...R15] Operation: $(RX) \leftarrow (RX) + (RY)$ Description: Add the contents of the register RY to the contents of the register RX and place the result in the register RX. Register direct addressing must be used for RX and RY. If there is the overflow (if (RX)+(RY)>15), set C. Otherwise, Flags affected: 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. bit 11 Encoding: 0 0 0 1 X X X Y X Υ The "0001" bits are the ADD RX,RY opcode The "XXXX" bits select the operand RX The "YYYY" bits select the operand RY bit 11 10 0 Example: 0 0 1 0 0 1 0 0 0 0 0 0 R2, R0 ADD OPCODE 0001 = ADD OPERAND RX **OPERAND RY** C Flag 1 Flag C 0 0 Z Flag Flag Z Flag V 0 V Flag 1011 R0 1011 R0 UNSIGNED **SIGNED** R1 R1 (decimal) (decimal) 0010 0111 R2 R2 1011 11 - 5 R3 R3 R4 R4 0111 + 7 + 7 R5 R5 R6 R6 R7 R7 0010 = 2 2 R8 R8 R9 R9 OUT OUT IN IN Note: Carry flag is set (C=1), which means that the result **JSR JSR** is not correct for unsigned operation. But the V flag is PCL **PCL** reset, so the result (2) is correct for signed operation. PCM PCM Note: If both operands are the same, instruction ADD **PCH PCH** RX,RY can be used as a substitute for the instruction SL RY (Shift RY Left) Before operation After operation Syntax: {label} ADC RX, RY Operands: $RX \in [R0...R15]$ RY ∈ [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. bit 11 10 **Encoding:** 0 X X Υ The "0010" bits are the ADC RX,RY opcode The "XXXX" bits select the operand RX The "YYYY" bits select the operand RY bit 11 10 6 Example: 0 1 0 0 0 0 1 0 1 1 0 1 R1, R7 **ADC** OPCODE 0010 = ADC OPERAND RX **OPERAND RY** C Flag Flag C 0 Z Flag Flag Z Flag V 0 V Flag R0 R0 UNSIGNED **SIGNED** 0100 1000 R1 (decimal) (decimal) R1 R2 R2 0100 4 4 R3 R3 R4 R4 1011 + 11 + (- 5) R5 R5 R6 R6 + 1 1011 1110 R7 R7 R8 R8 R9 R9 0000 = 0- 0 OUT OUT IN IN **JSR JSR PCL PCL** PCM PCM Hint: If both operands are the same, instruction ADC RX,RY can be used as a substitute for the instruction **PCH PCH** RLC RY (Rotate Left RY Through Carry) Before operation After operation

Syntax: {label} SUB RX, RY Operands: $RX \in [R0...R15]$ 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. 10 bit 11 Encoding: 0 0 1 1 X X X X 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: bit 11 10 7 0 1 0 1 1 0 0 1 0 0 SUB R6, R2 OPCODE 0011 = SUB OPERAND RX OPERAND RY C Flag Flag C 0 Z Flag Flag Z 0 Flag V 0 0 V Flag R0 R0 **UNSIGNED SIGNED** R1 R1 (decimal) (decimal) 1001 R2 1001 R2 R3 R3 1111 15 - 1 R4 R4 R5 R5 1001 - 9 - (- 7) 1111 0110 R6 R6 R7 R7 0110 = 6 6 R8 R8 R9 R9 OUT OUT IN IN **JSR JSR** Note: For subtraction, Carry flag is called Borrow, and it PCL **PCL** is actually inverse Carry. So, No-Carry (if C=0) means "Borrow", and Carry (C=1) means "No Borrow". In this **PCM PCM** example the resulting C=1, so the result is correct in PCH **PCH** unsigned representation. Flag V is reset, which means Before operation After operation that the result is correct in signed representation also.



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.

Syntax: {label} SBB RX, Operands: $RX \in [R0...R15]$ RY ∈ [R0...R15] Operation: $(RX) \leftarrow (RX) - (RY) - (\overline{C})$ 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. 10 bit 11 Encoding: 0 1 0 0 X X X X 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: bit 11 10 1 0 0 1 1 0 0 1 1 0 SBB R5, R3 OPCODE 0100 = SBB OPERAND RX OPERAND RY C Flag Flag C Z Flag Flag Z 0 0 Flag V V Flag 0 R0 R0 UNSIGNED **SIGNED** R1 R1 (decimal) (decimal) R2 R2 0011 R3 1001 R3 1110 14 2 R4 R4 R5 1110 0110 R5 0011 -(+3)R6 R6 R7 R7 - 1 R8 R8 R9 R9 1010 = 10OUT OUT IN IN **JSR JSR PCL PCL** Note: For subtraction, Carry flag is called Borrow, and it PCM **PCM** is inverse Carry. In this example the resulting flags are **PCH PCH** C=1 and V=0, which means that the result is correct both for unsigned and in signed representations. Before operation After operation



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

{label} OR RX, Syntax: $RX \in [R0...R15]$ Operands: RY ∈ [R0...R15] Operation: $(RX) \leftarrow (RX) .OR. (RY)$ 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 bit 11 Encoding: X 0 1 0 1 X 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: bit 11 10 1 0 0 0 0 0 1 1 1 0 OR R0, R7 OPCODE 0101 = OR OPERAND RX OPERAND RY C Flag Flag C 0 0 Flag Z 1 **Z** Flag 0 Flag V 0 V Flag R0 0101 1101 R0 R1 R1 R2 R2 R3 R3 R4 R4 R5 R5 0101 5 R6 R6 1101 1101 R7 R7 1100 .OR. 12 R8 R8 R9 R9 OUT OUT 1101 = 13 IN IN **JSR JSR** PCL **PCL PCM PCM** PCH **PCH** Before operation After operation

AND RX,RY

Logical AND registers RX and RY

{label} AND RX, RY Syntax: $RX \in [R0...R15]$ Operands: RY ∈ [R0...R15] Operation: $(RX) \leftarrow (RX)$.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 bit 11 Encoding: 0 1 1 0 X X X Υ The "0110" bits are the AND RX,RY opcode The "XXXX" bits select the operand RX The "YYYY" bits select the operand RY Example: bit 11 10 1 1 1 0 0 0 1 R13, R12 AND OPCODE 0110 = AND OPERAND RX OPERAND RY C Flag Flag C 0 0 Z Flag Flag Z 1 0 Flag V 0 0 V Flag R0 R0 R1 R1 R2 R2 R3 R3 R4 R4 R5 R5 R6 R6 R7 R7 R8 R8 1110 14 R9 R9 1110 0110 OUT OUT 0111 .AND. 7 0111 0111 IN IN **JSR JSR** PCL **PCL** 0110 PCM **PCM** PCH **PCH** Before operation After operation

XOR RX,RY

Exclusive OR registers RX and RY

{label} XOR RX, RY Syntax: $RX \in [R0...R15]$ Operands: RY ∈ [R0...R15] Operation: $(RX) \leftarrow (RX) . XOR. (RY)$ 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 0 1 The "0111" bits are the XOR RX,RY opcode The "XXXX" bits select the operand RX The "YYYY" bits select the operand RY bit 11 10 0 Example: 1 1 1 1 0 0 0 0 0 1 1 0 XOR R8, R3 OPCODE 0111 = XOR OPERAND X OPERAND Y C Flag Flag C 0 0 Z Flag Flag Z 1 0 Flag V 0 V Flag 0 R0 R0 R1 R1 R2 R2 R3 1100 1100 R3 R4 R4 R5 R5 0110 6 R6 R6 R7 R7 1100 .XOR.12 0110 1010 R8 R8 R9 R9 OUT OUT 1010 = 10 IN IN **JSR JSR PCL PCL** PCM PCM **PCH PCH** Before operation After operation

{label} MOV RX, RY Syntax: $RX \in [R0...R15]$ Operands: RY ∈ [R0...R15] Operation: $(RX) \leftarrow (RY)$ Move the 4-bit contents from the register RY to the register Description: RX. Register direct addressing must be used for RX and RY. Flags affected: None. bit 11 10 Encoding: 1 X X 0 0 X X The "1000" bits are the MOV RX,RY opcode The "XXXX" bits select the operand RX The "YYYY" bits select the operand RY Example: bit 11 0 0 1 0 0 0 1 1 0 1 0 1 MOV R6, R5 OPCODE 1000 = MOV OPERAND RX OPERAND RY C Flag Flag C 0 0 **Z** Flag 0 0 Flag Z V Flag Flag V 0 R0 R0 R1 R1 R2 R2 R3 R3 R4 R4 0010 R5 0010 R5 1110 0010 R6 R6 R7 R7 Note: Contents of the source operand has NOT changed. This rule is valid for all instructions; only R8 R8 the destination register is modified by the R9 R9 operation. OUT OUT IN IN **JSR JSR** Note: If the instruction MOV RX,RY has the register JSR (0x0C) or PCL (0x0D) as the **PCL PCL** destination, then Subroutine Call or Program **PCM PCM** Jump will be executed. Please read the main PCH **PCH** User's Manual. After operation Before operation

{label} MOV RX, N Syntax: $RX \in [R0...R15]$ Operands: $N \in 0...15$ Operation: $(RX) \leftarrow N$ Description: Move the 4-bit literal to the register RX. Register direct addressing must be used for RX Flags affected: None. bit 11 10 Encoding: 1 X Ν The "1001" bits are the MOV RX,N opcode The "XXXX" bits select the operand RX The "NNNN" bits are the literal N bit 11 Example: 0 0 1 0 0 0 1 1 1 1 MOV R9.7 OPERAND RX OPCODE 1001 = MOV RX,N LITERAL N Flag C 0 C Flag Z Flag Flag Z 0 Flag V V Flag 0 R0 R0 R1 R1 R2 R2 R3 R3 R4 R4 R5 R5 R6 R6 R7 R7 R8 R8 1010 0111 R9 R9 OUT OUT IN IN Note: If the instruction MOV RX,N has the register **JSR** JSR JSR (0x0C) or PCL (0x0D) as the destination, PCL **PCL** then Subroutine Call or Program Jump will be **PCM PCM** executed. Please read the main User's Manual. PCH **PCH** 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)

{label} MOV [XY], R0 Syntax: RX ∈ [R0...R15] Operands: RY ∈ [R0...R15] Operation: $((RX):(RY)) \leftarrow (R0)$ 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. bit 11 9 8 5 3 2 10 6 0 **Encoding:** 1 1 0 X X Υ Υ 0 X X The "1010" bits are the MOV [XY],R0 opcode The "XXXX" bits select the operand RX The "YYYY" bits select the operand RY Example: bit 11 10 3 1 0 1 0 1 0 0 1 1 0 0 0 MOV [R6:R4],R0 OPCODE 1010 OPERAND RX OPERAND RY = MOV [XY],R0 C Flag Flag C 0 0 **Z** Flag Flag Z 0 0 Flag V 0 V Flag R0 1111 **DATA DATA** R1 **MEMORY MEMORY** R2 0x00 0x00 0x01 0x01 R3 0x02 0x02 0010 R4 0x03 0x03 0x04 0x04 R5 0x05 0x05 0010 R6 0x06 0x06 0x07 0x07 Bit R7 0x08 0x08 0001 0x090x09 R8 0x0A 0x0A 0001 Bit R9 7 0x0B 0x0B 0x0C 0x0C 0001 0010 OUT 0x0D 0x0D DATA MEMORY 0x0E IN 0x0E **ADDRESS** 0x0F 0x0F **JSR** 0x10 0x10 0x11 0x11 **PCL** 0101 0x12 0x12 **PCM** 0x13 0x13 0x14 0x14 **PCH** 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

{label} MOV R0, [XY] Syntax: RX ∈ [R0...R15] Operands: RY ∈ [R0...R15] Operation: $(R0) \leftarrow ((RX):(RY))$ 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. bit 11 10 **Encoding:** 1 0 1 X X Υ 1 X The "1011" bits are the MOV R0,[XY] opcode The "XXXX" bits select the operand RX The "YYYY" bits select the operand RY Example: 10 0 0 1 1 1 0 0 0 1 1 1 MOV R2, [HR7] OPCODE 1011 OPERAND RX OPERAND RY = MOV R0,[XY]C Flag Flag C 0 0 Z Flag Flag Z 0 0 Flag V V Flag R0 0011 R0 1110 **DATA** R1 R1 **MEMORY** R2 R2 0x00 0x01 R3 R3 0x02 R4 R4 0101 0101 0x03 0x04 R5 R5 3 0x05 0101 R6 R6 1010 1010 R7 R7 1010 0x52 R8 R8 0x53 R9 R9 0 0x54 0x55 0101 1010 OUT OUT 0x56 **DATA MEMORY** IN IN 0x57 **ADDRESS** 0x58 **JSR JSR** 0x59 PCL 1110 0x5A **PCL** 0x5B **PCM PCM** 0x5C 0x5D PCH **PCH** Before operation Unchanged After operation

MOV [NN],R0

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

{label} MOV [NN], R0 Syntax: $NN \in [0...255]$ Operands: $(NN) \leftarrow (R0)$ Operation: Move the 4-bit contents of register R0 to data Description: memory addressed by unsigned literal [NN]. None. Flags affected: **Encoding:** bit 11 10 1 1 0 0 N Ν Ν Ν Ν The "1100" bits are the MOV [NN],R0 opcode The "NNNNNNNN" bits are unsigned literal NN Example: bit 11 10 0 1 0 0 1 1 0 0 1 0 0 0 MOV [0x19], R0 1 OPCODE 1100 LITERAL NN = MOV [NN],R0 C Flag Flag C Z Flag 0 0 Flag Z Flag V 0 0 V Flag **DATA DATA** R0 1001 **MEMORY MEMORY** R1 0x00 0x00 0x01 0x01 R2 0x02 0x02 R3 0x03 0x03 0x04 0x04 R4 0x05 0x05 0x06 0x06 R5 0x07 0x07 R6 0x08 0x08 0x09 0x09 R7 DATA MEMORY ADDRESS 0x0A 0x0A R8 0x0B 0x0B 0x0C 0x0C R9 0x0D 0x0D 0x0E OUT 0x0E 0x0F 0x0F IN 0x10 0x10 0x11 0x11 JSR 0x12 0x12 **PCL** 0x13 0x13 0x14 0x14 **PCM** 0x15 0x15 0x16 **PCH** 0x16 0x17 0x17 0x18 Unchanged 0x18 0111 0x19 1001 0x19 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

{label} MOV R0, [NN] Syntax: $NN \in [0...255]$ Operands: $(R0) \leftarrow (NN)$ Operation: Move the 4-bit contents of data memory addressed by Description: unsigned literal NN to register R0. Register direct addressing must be used for R0. None. Flags affected: bit 11 10 9 5 3 0 **Encoding:** 1 1 0 1 Ν N Ν Ν Ν Ν Ν Ν The "1101" bits are the MOV R0,[NN] opcode The "NNNNNNN" bits are unsigned literal NN Example: bit 11 10 9 8 7 6 5 3 2 0 1 1 0 1 1 1 1 0 0 0 1 0 MOV R0, [0xE2] LITERAL NN OPCODE 1101 = MOV R0,[NN] C Flag Flag C 0 0 Z Flag Flag Z 0 0 Flag V 0 0 V Flag R0 0011 1110 R0 **DATA** R1 R1 **MEMORY** R2 R2 0x00 0x01 R3 R3 0x02 DATA MEMORY ADDRESS R4 R4 0x03 0x04 R5 R5 0x05 R6 R6 R7 R7 0xDC R8 R8 0xDD R9 R9 0xDE 0xDF OUT OUT 0xE0 IN IN 0xE 1110 0xE2 **JSR JSR** 0xE3 0xE4 **PCL PCL** 0xE5 **PCM PCM** 0xE6 0xE7 PCH PCH Unchanged Before operation After operation Note: R0 is the only register that can be used in this instruction.

{label} LPC NN Syntax: $NN \in [0...255]$ Operands: 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. 10 bit 11 Encoding: 1 1 0 Ν 1 Ν Ν Ν Ν The "1110" bits are the LPC #NN opcode The "NNNNNNN" bits are literal #NN Example: bit 11 0 1 1 1 0 0 1 1 0 0 0 1 MOV PC,0x31 OPCODE 1110 = LPC LITERAL NN HIGH LITERAL NN LOW C Flag Flag C 0 0 **Z** Flag Flag Z 0 0 Flag V 0 V Flag R0 R0 R1 R1 R2 R2 ALWAYS POINTS TO PCM, PCH R3 R3 R4 R4 R5 R5 R6 R6 R7 R7 R8 R8 R9 R9 OUT OUT IN IN **JSR JSR PCL PCL** 0001 0110 PCM **PCM** 1010 0011 PCH PCH After operation Before operation

JR NN

Jump relative

Syntax: {label} JR NN

Operands: $NN \in [-128...+127]$

Operation: Program Counter ← Program Counter + NN signed

Description: Add signed integer value NN to the Program Counter

Flags affected: None.

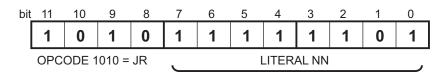
Encoding: bit 11 10 9 8 7 6 5 4 3 2 1 0

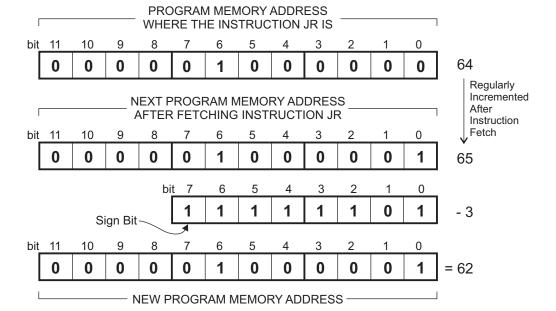
Encoding: 1 0 1 0 N N N N N N N N N

The "1010" bits are the JR NN opcode
The "NNNNNNNNN" 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.

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.

{label} CP R0, N Syntax: R0 Operands: N ∈ 0...15 (R0) - N, set flags only Operation: Compute unsigned R0 – N and update the C and Z flags. Description: The result of the subtraction is not stored. If (R0) > N or (R0) = N, set C. Otherwise, reset C. Flags affected: If (R0) = N, set Z. Otherwise, reset Z. bit 11 10 Encoding: 0 0 Ν N The "0000 0000" bits are the CP R0,N opcode The "NNNN" bits are literal N Example: bit 11 10 0 0 0 0 0 0 0 0 0 1 1 CP R0, 5 OPCODE 0000 0000 = CP R0,N LITERAL N C Flag Flag C Flag Z 1 **Z** Flag Flag V 0 V Flag 0101 R0 0101 5 0101 R0 R1 R1 0101 R2 R2 R3 R3 R4 R4 0000 = 0R5 R5 R6 R6 R7 R7 R8 R8 R9 R9 Note: Instruction CP performs unsigned OUT OUT subtraction, but the result is NOT stored, so IN IN operand R0 is unchanged after operation. **JSR** JSR Only flags C and Z are affected. PCL If flag Z is set after operation, it means that **PCL** R0 = N, and if flag C is set after operation, **PCM PCM** it means that R0 >= N. PCH **PCH** Before operation After operation

Syntax: {label} ADD R0, N

Operands: R0

N ∈ 0...15

Operation: $(R0) \leftarrow (R0) + N$

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: bit 11 10 0 0 0 0 0 0 1 1 0 R0, 14 ADD OPCODE 0000 0001 = ADD R0,N LITERAL N C Flag Flag C 0 1 1 **Z** Flag Flag Z 0 Flag V 0 0 V Flag 0000 0010 R0 0010 2 R0 R1 R1 1110 R2 R2 R3 R3 R4 R4 0000 = 16 R5 R5 0000 with C set R6 R6 R7 R7 R8 R8 R9 R9 OUT OUT IN IN **JSR JSR** PCL **PCL** PCM **PCM** PCH Before operation After operation

Syntax: {label} INC RY

Operand: $RY \in [R0...R15]$

Operation: $(RY) \leftarrow (RY)+1$

Description: Add 1 to the contents of the 4-bit register RY and place

the result back into the register RY.

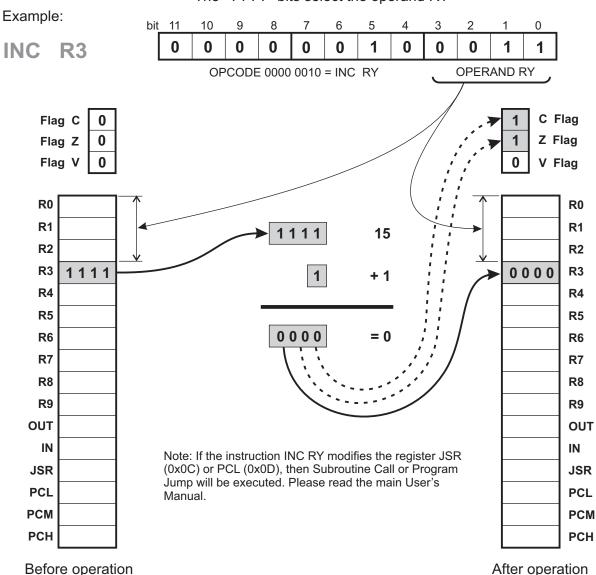
Flags affected: Flag C is not affected.

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 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



Syntax: {label} DEC RY

Operand: $RY \in [R0...R15]$

Operation: $(RY) \leftarrow (RY)-1$

Description: Subtract 1 from the contents of the 4-bit register RY and

place the result back into the register RY.

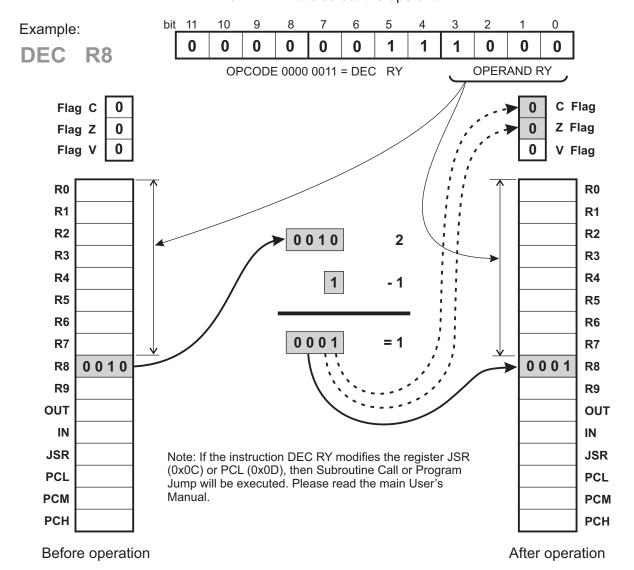
Flags affected: If result=0000 after operation, set flags Z and C. Otherwise,

reset flags Z and C.

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

0 0 0 0 0 1 1 Y Y Y Y

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





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

{label} DSZ RY Syntax: RY ∈ [R0...R15] Operands: $(RY) \leftarrow (RY)$ -1, if result is =0, then $PC \leftarrow PC$ +1 Operation: Subtract 1 from the contents of the 4-bit register RY and Description: if result is =0, increment Program Counter by 1. Flags affected: None. 10 bit 11 Encoding: 0 0 The "0000 0100" bits are the DSZ RY opcode The "YYYY" bits are operand Y Example: bit 11 0 0 0 0 0 1 0 0 0 0 1 1 DSZ R3 OPCODE 0000 0100 = DSZ RY OPERAND RY C Flag 0 Flag C **Z** Flag Flag Z 0 0 Flag V V Flag 0001 1 R0 R0 1 - 1 R1 R1 R2 R2 0000 = 0R3 R3 0001 0000 IF RESULT=0. THEN SKIP ONE INSTAL Note: Although it employs subtraction, this instruction does not affect flags. **PROGRAM ADDRESS PROGRAM CODE** 1010 1000 0000 DSZ 0 0 0 0 1 0 0 0 0 1 1 R3 1010 1000 0001 1 1 1 1 1 0 1 1 1 1 JR -17 1 0 0 0 1010 1000 0010 0 1 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.

{label} OR R0, N Syntax: R0 Operands: N ∈ 0...15 $(R0) \leftarrow (R0) .OR. N, C \leftarrow 1$ Operation: Compute the logical inclusive OR operation of the 4-bit Description: register R0 with the 4-bit literal value N and place the result back into the register R0. Flag C is unconditionally set Flags affected: If result=0000 after operation, set Z. Otherwise, reset Z **Encoding:** bit 11 10 0 0 0 Ν Ν Ν Ν The "0000 0101" bits are the OR R0,N opcode The "NNNN" bits are literal N Example: bit 11 10 0 0 0 0 1 0 0 1 0 OR R0, 6 OPCODE 0000 0101 = OR R0,N LITERAL N C Flag Flag C 1 Flag Z 0 **Z** Flag 1 Flag V 0 0 V Flag 0011 0111 R0 0011 3 R0 R1 R1 R2 0110 .OR. 6 R2 R3 R3 R4 R4 0111 R5 R5 R6 R6 R7 R7 R8 R8 R9 R9 OUT **OUT** IN IN Note: This instruction can be used as a direct replacement for **JSR JSR** the non-existent instruction SET C. In that case, contents of PCL register R0 will be preserved if the literal value is 0000. **PCL** However, the previous flag Z contents will be lost. **PCM PCM PCH PCH**

After operation

Before operation

{label} AND R0, N Syntax: R0 Operands: N ∈ 0...15 $(R0) \leftarrow (R0)$.AND. N, C $\leftarrow 0$ Operation: Compute the logical inclusive AND operation of the 4-bit Description: register R0 with the 4-bit literal value N and place the result back into the register R0. Flag C is unconditionally reset Flags affected: If result=0000 after operation, set Z. Otherwise, reset Z **Encoding:** bit 11 10 0 0 0 Ν Ν Ν Ν The "0000 0110" bits are the AND R0,N opcode The "NNNN" bits are literal N Example: bit 11 10 0 0 0 0 1 1 0 1 0 0 AND R0, 6 OPCODE 0000 0110 = AND R0,N LITERAL N C Flag Flag C 0 1 Flag Z 1 0 **Z** Flag 0 Flag V 0 V Flag 1100 1000 R0 1100 12 R0 R1 R1 1010 .AND. 10 R2 R2 R3 R3 R4 R4 1000 R5 R5 R6 R6 R7 R7 R8 R8 R9 R9 OUT OUT IN IN 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.

JSR

PCL

PCM

PCH

After operation

Before operation

JSR

PCL

PCM

PCH

{label} XOR R0, N Syntax: R0 Operands: N ∈ 0...15 $(R0) \leftarrow (R0) .XOR. N, C \leftarrow \neg C$ Operation: Compute the logical exclusive OR operation of the 4-bit Description: register (R0) with the 4-bit literal value N and place the result back into the register R0. Flag C is unconditionally toggled (complemented). Flags affected: If result=0000 after operation, set Z. Otherwise, reset Z. **Encoding:** bit 11 10 0 0 0 Ν Ν Ν Ν The "0000 0111" bits are the XOR R0,N opcode The "NNNN" bits are literal N Example: bit 11 10 0 0 0 0 1 0 0 1 XOR R0, 3 OPCODE 0000 0111 = XOR R0,N LITERAL N C Flag Flag C 0 Flag Z 1 0 **Z** Flag Flag V 0 0 V Flag R0 1001 1001 1010 9 R0 R1 R1 0011 .XOR. 3 R2 R2 R3 R3 R4 R4 1010 = 10 R5 R5 R6 R6 R7 R7 R8 R8 R9 R9 OUT OUT

Note: This instruction can be used as a direct replacement for

However, the previous flag Z contents will be lost.

the non-existent instruction TOGGLE C. In that case, contents of register R0 will be preserved if the literal value is #0000.

IN

JSR

PCL

PCM

PCH

After operation

Before operation

IN

JSR

PCL

PCM

PCH

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

 $N \in 0...15$ (Special case: N=0 means N=16)

Operation: $(R0)...(RN) \leftrightarrow (0xE0)...(0xEN)$

Description: Exchange N registers from the page 0 with registers

from the alternate set in page 14. Special case: if N=0,

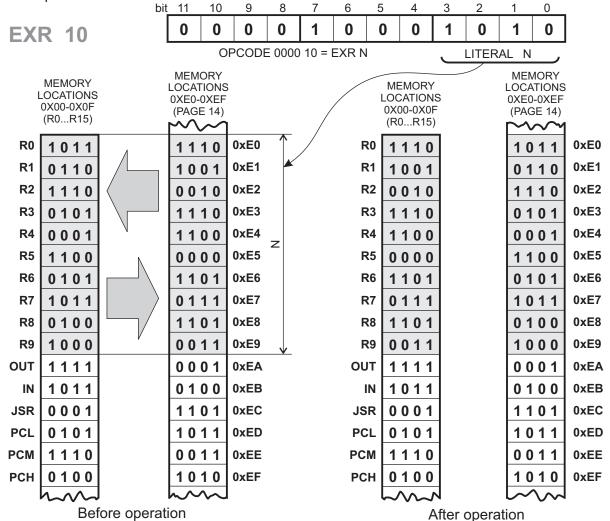
then 16 registers will be exchanged

Flags affected: None.

bit 11 10 4 **Encoding:** 0 0 0 0 1 0 0 0 Ν Ν Ν Ν

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

Example:



Note: Register R0 (data memory location 0x00) is always exchanged with memory location 0xE0, and then, if N \neq 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

{label} BIT RG, M Syntax: $RG \in [R0|R1|R2|RS]$ Operands: $N \in 0 | 1 | 2 | 3$ or 0 | 1 | 2 | SOperation: 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. 11 10 0 bit Encoding: 0 0 1 G G M 0 M The "0000 1001" bits are the BIT RG,M opcode The "NNNN" bits are literal N Example: bit 11 10 0 0 0 1 0 0 1 0 1 BIT R2, 3 OPCODE 0000 1001 = BIT RG,M G Μ C Flag Flag C 0 0 Flag Z 1 0 **Z** Flag V Flag Flag V 0 R0 R0 R1 R1 1101 1101 R2 R2 bit R3 R3 0 1 R4 R4 R5 R5 R6

> 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.

R6

R7

R8

R9

IN

OUT

JSR

PCL

PCM

PCH

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).

Before operation After operation

R7

R8

R9

OUT

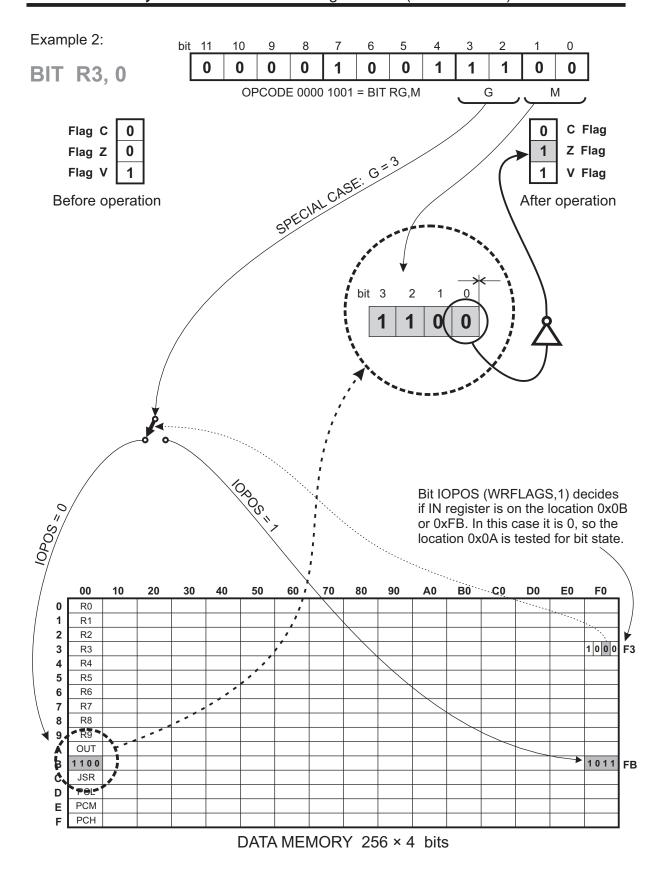
JSR

PCL

PCM

PCH

IN



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

Set bit M in register RG {label} BSET RG, M Syntax: $RG \in [R0|R1|R2|RS]$ Operands: $N \in 0 | 1 | 2 | 3$ or 0 | 1 | 2 | SOperation:
bit> ← 1 Description: Set bit N in register addressed by RG. Flags affected: None. bit 11 10 Encoding: 0 G G M The "0000 1010" bits are the BSET RG,M opcode The "NNNN" bits are literal N Example: bit 11 10 1 1 1 1 0 0 0 0 0 0 0 0 BSET R1, 2 OPCODE 0000 1010 = BSET RG,M G M C Flag Flag C Z Flag Flag Z 1 1 Flag V 0 V Flag R0 R0 1100 1000 R1 R1 R2 R2 bit 3 2 1 R3 R3 1 0 R4 R4 R5 R5 R6 R6 R7 R7 0 R8 R8 R9 R9 OUT **OUT** IN IN Note: If G=0, 1 or 2, then register R0, R1 or **JSR JSR** R2 is used as the destination. If G=3, then **PCL PCL** register IN is the destination. It can be either **PCM** at the address 0x0A (if WrFlags,1=0) or at **PCM**

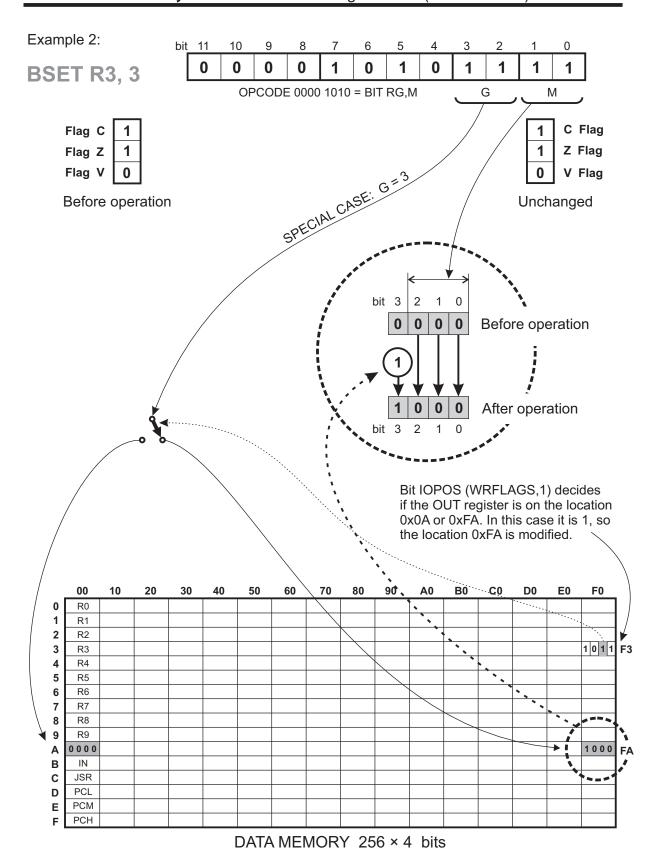
the address 0xFA (if WrFlags, 1=1).

PCH

After operation

Before operation

PCH



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

Syntax: {label} BCLR RG, M

Operands: $RG \in [R0 | R1 | R2 | RS]$

 $N \in 0 | 1 | 2 | 3$ or 0 | 1 | 2 | S

Operation: $\langle bit \rangle \leftarrow 0$

Description: Clear bit #N in register addressed by RG.

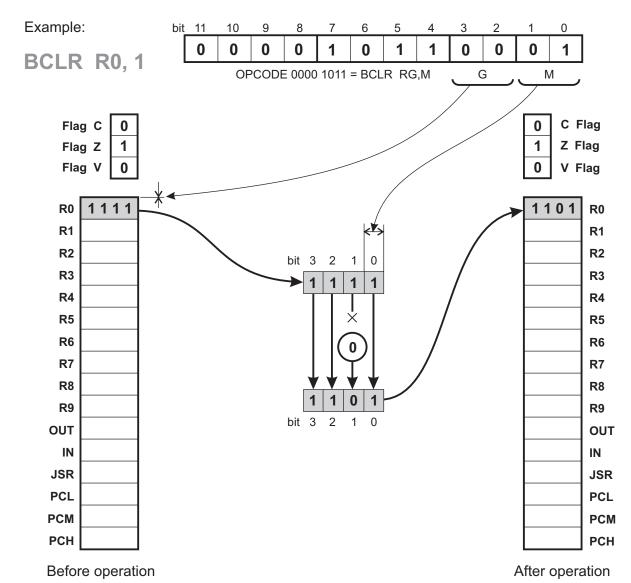
Flags affected: None.

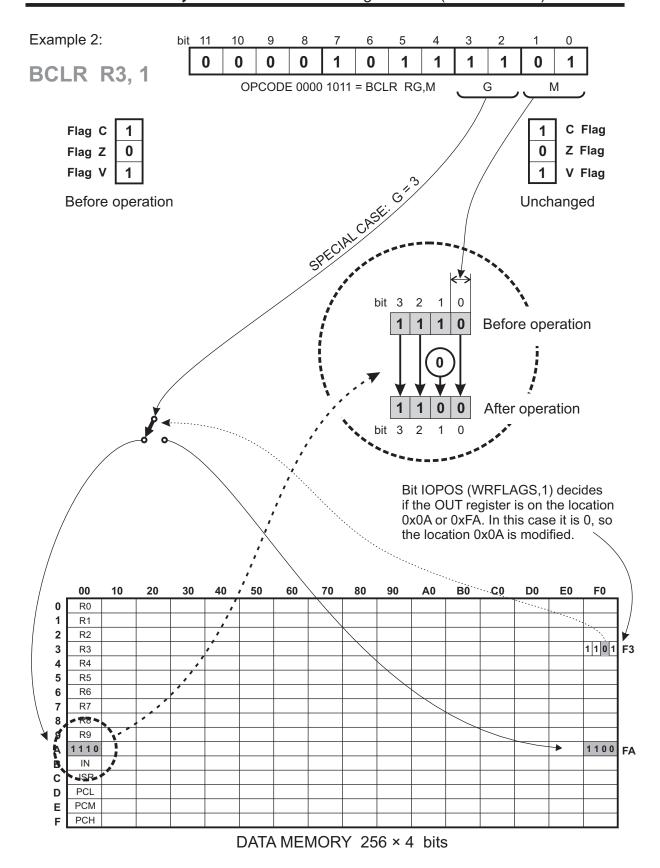
Encoding: bit 11 10 9 8 7 6 5 4 3 2 1 0

0 0 0 1 0 1 1 G G M M

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

The "NNNN" bits are literal N





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

Syntax: {label} BTG RG, M

Operands: $RG \in [R0 | R1 | R2 | RS]$

 $N \in 0 | 1 | 2 | 3$ or 0 | 1 | 2 | S

Operation: <bit> ← -<bit>

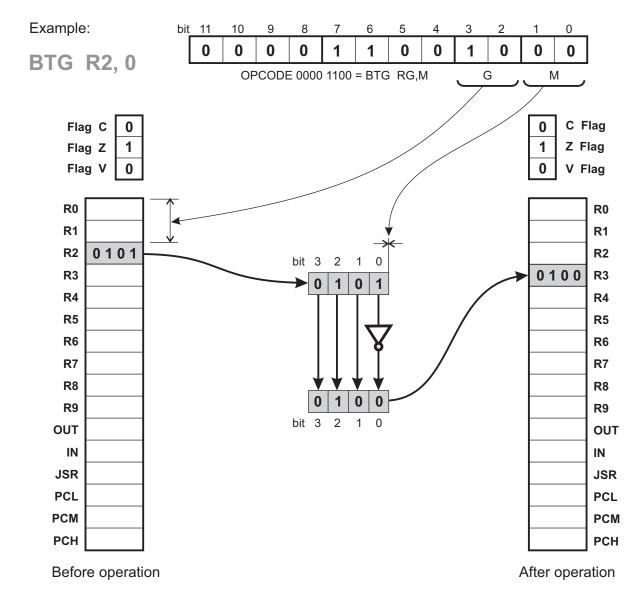
Description: Invert bit #N in register addressed by RG.

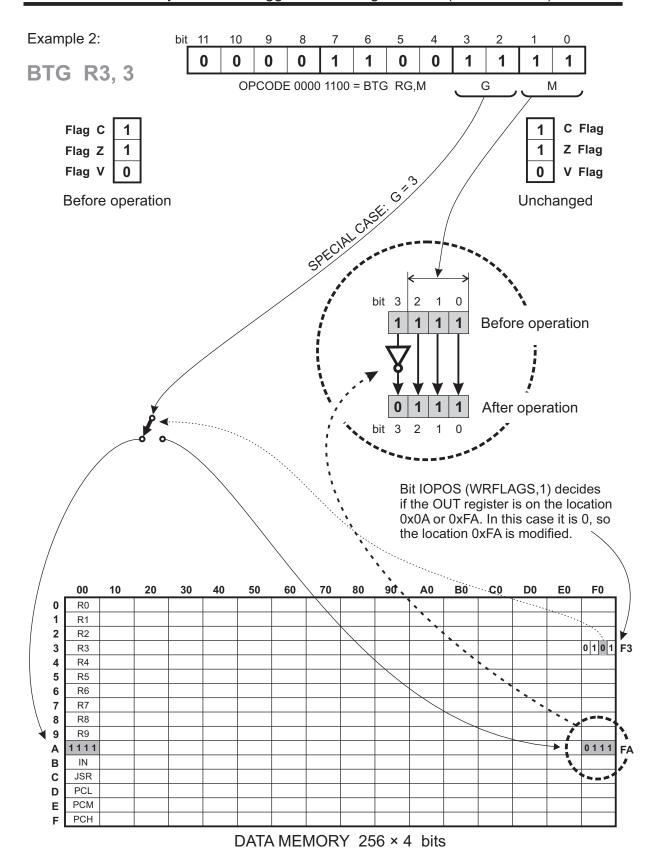
Flags affected: None.

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

The "0000 1100" bits are the BTG $\,$ RG,M opcode

The "NNNN" bits are literal N





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

Syntax: {label} RRC RY

Operand: $RY \in [R0...R15]$

Operation: $(C) \leftarrow (RY0), (RY3) \leftarrow (C), (RY2) \leftarrow (RY3), (RY1) \leftarrow (RY2),$

(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.

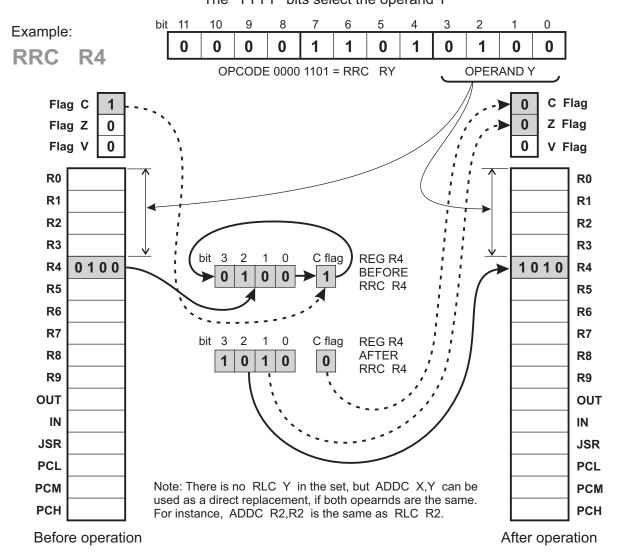
Flags affected: Bit 0 of the register RY is copied to Flag C.

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 1 1 0 1 Y Y Y

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



{label} RET R0,N Syntax: R0 Operands: N ∈ 0...15 $(R0) \leftarrow ((PC))$ Operation: $PC < 11...0 > \leftarrow ((SP \times 3 + 2), (SP \times 3 + 1), (SP \times 3))$ SP ← SP-1 Load R0 with literal value N and pop PC from stack Description: Flags affected: None. bit 11 10 9 0 Encoding: 0 0 1 Ν Ν 0 0 0 Ν Ν The "0000 1110" bits are the RET opcode The "NNNN" bits are literal N Example: bit 11 10 0 0 1 1 0 0 1 0 0 0 RET R0, #4 OPCODE 0000 1110 = RET LITERAL N **DATA DATA MEMORY MEMORY** Note: In the example, program returns **R0** 0x00 1010 0100 0x00 **R0** from the 2th level of subroutine to the **R1** 0x01 0x01 **R1** 1st level, that's why the instruction **R2** 0x02 0x02 **R2** decremented SP from 2 to 1. On the R3 0x03 0x03 R3 next RET execution, when program **R4** 0x04 0x04 **R4** returns to the level 0 (no subroutine). 0x05 **R5 R5** 0x05 return address 0000 0000 1111 will be loaded to the Program Counter. JSR 0x0C 0x0C JSR PCL 0x0D 0x0D PCL **STACK STACK** 0x0E PCM PCM 0x0E **POINTER POINTER** PCH 0x0F 0x0F PCH 0x10 0x10 1111 1111 0x11 $0 \ 0 \ 0 \ 0$ 0000 0x11 0000 0000 0x12 0x12 1011 1011 0x13 0x13 1100 0x14 1100 0x14 0001 0001 0x15 0x15 0x16 0x16 0x17 0x17 001001011100 000111001011 PROGRAM COUNTER PROGRAM COUNTER Before operation After operation

SKIP F, M

Skip next M instructions conditionally

{label} SKIP z|nz|c|nc, M Syntax:

 $F \in z \mid nz \mid c \mid nc$ Operands:

 $M \in 0...3$ (Special case: M=0 means M=4)

Operation: If condition=true, $(PC) \leftarrow (PC+M)$

Description: If condition=true, skip the next M instructions.

Special case: if M=0, then skip 4 instructions.

Flags affected: None.

10 bit 11 Encoding: F 0 0 0 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

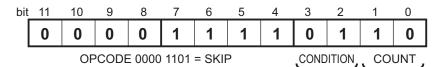
Instructions skipped

CONDITIO	CONDITION	
0	0	С
0	1	NC
1	0	Z
1	1	NZ

STEP COL	JNT M	SKIP INSTRUCTIONS
0	0	4
0	1	1
1	0	2
1	1	3

Example:

SKIP nc, #2



THEN SKIP 2 INSTRUCTIONS Flag Z **PROGRAM CODE PROGRAM ADDRESS** 0000 1000 1101 0 0 0 1 1 1 **SKIP** nc,2 0000 1000 1110 1 1 1 0 1 0 XOR R5,R11 1 0 0000 1000 1111 0 0 1 0 0 0 0 1 MOV R8,5 0000 1001 0000 1 1 1 0 0 0 1 1 1 1 1 MOV PC,0x1F 0 0000 1001 0001 1 MOV R13,0 0 1 0

Flag C

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.