MSP430 Instruction Set

Double Operand Instructions

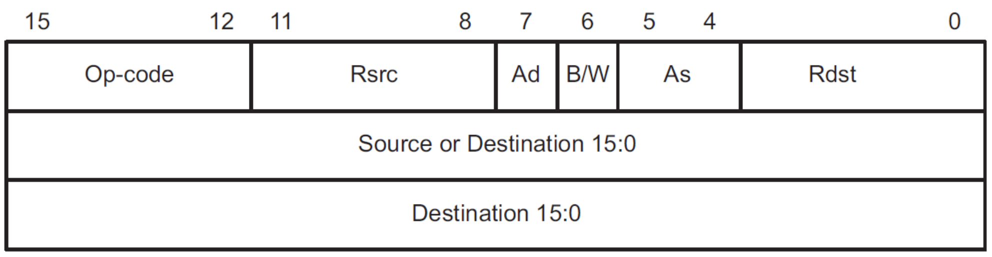


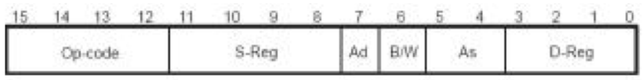
Figure 6-22. MSP430 Double-Operand Instruction Format

Table 6-4. MSP430 Double-Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits[1] V	N	Z	C
MOV(.B)	src,dst	src → dst	–	–	–	–
ADD(.B)	src,dst	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	src + dst + C → dst	*	*	*	*
SUB(.B)	src,dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	dst + .not.src + C → dst	*	*	*	*
CMP(.B)	src,dst	dst - src	*	*	*	*
DADD(.B)	src,dst	src + dst + C → dst (decimally)	*	*	*	*
BIT(.B)	src,dst	src .and. dst	0	*	*	$\overline{Z}$
BIC(.B)	src,dst	.not.src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	src .or. dst → dst	–	–	–	–
XOR(.B)	src,dst	src .xor. dst → dst	*	*	*	$\overline{Z}$
AND(.B)	src,dst	src .and. dst → dst	0	*	*	$\overline{Z}$

[1] * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

MSP430: 16, 16-bit registers
R0 - Program counter
R1 - Stack pointer (SP)
R2 - Status register (SR)
R3 - Constant generator



As	Ad	Addressing Mode	Syntax	Description
00	0	Register Mode +0	Rn	Register contents are operand
01	1	Indexed Mode +1	X(Rn)	(Rn + X) points to the operand. X is stored in the next word
01	1	Symbolic Mode +1	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed Mode X(PC) is used
01	1	Absolute Mode +1	&ADDR	The word following the instruction contains the absolute address.
10	-	Indirect Register Mode +0	@Rn	Rn is used as a pointer to the operand
11	-	Indirect Autoincrement +0	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards
11	-	Immediate Mode +1	#N	The word following the instruction contains the immediate constant N. Indirect Autoincrement Mode @PC+ is used

Single Operand Instructions

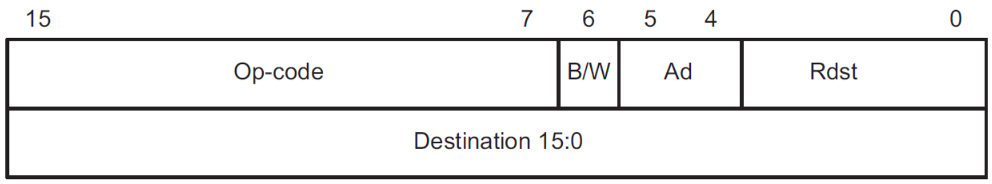


Figure 6-23. MSP430 Single-Operand Instructions

Table 6-5. MSP430 Single-Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits[1]			
			V	N	Z	C
`RRC(.B)`	dst	C → MSB →.......LSB → C	0	*	*	*
`RRA(.B)`	dst	MSB → MSB →....LSB → C	0	*	*	*
`PUSH(.B)`	src	SP - 2 → SP, src → SP	–	–	–	–
`SWPB`	dst	bit 15...bit 8 ↔ bit 7...bit 0	–	–	–	–
`CALL`	dst	Call subroutine in lower 64KB	–	–	–	–
`RETI`		TOS → SR, SP + 2 → SP	*	*	*	*
		TOS → PC,SP + 2 → SP				
`SXT`	dst	Register mode: bit 7 → bit 8...bit 19 Other modes: bit 7 → bit 8...bit 15	0	*	*	$\overline{Z}$

[1] * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

Jump Instructions

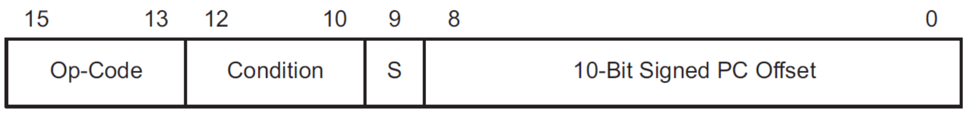
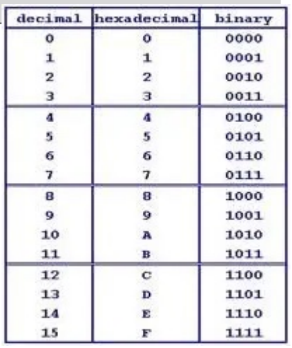


Figure 6-24. Format of Conditional Jump Instructions

Table 6-6. Conditional Jump Instructions

Mnemonic	S-Reg, D-Reg	Operation
`JEQ, JZ`	Label	Jump to label if zero bit is set
`JNE, JNZ`	Label	Jump to label if zero bit is reset
`JC`	Label	Jump to label if carry bit is set
`JNC`	Label	Jump to label if carry bit is reset
`JN`	Label	Jump to label if negative bit is set
`JGE`	Label	Jump to label if (N .XOR. V) = 0
`JL`	Label	Jump to label if (N .XOR. V) = 1
`JMP`	Label	Jump to label unconditionally



decimal	hexadecimal	binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Emulated Instructions

Table 6-7. Emulated Instructions

Instruction	Explanation	Emulation	Status Bits[1]			
			V	N	Z	C
`ADC(.B) dst`	Add Carry to dst	`ADDC(.B) #0,dst`	*	*	*	*
`BR dst`	Branch indirectly dst	`MOV dst,PC`	–	–	–	–
`CLR(.B) dst`	Clear dst	`MOV(.B) #0,dst`	–	–	–	–
`CLRC`	Clear Carry bit	`BIC #1,SR`	–	–	–	0
`CLRN`	Clear Negative bit	`BIC #4,SR`	–	0	–	–
`CLRZ`	Clear Zero bit	`BIC #2,SR`	–	–	0	–
`DADC(.B) dst`	Add Carry to dst decimally	`DADD(.B) #0,dst`	*	*	*	*
`DEC(.B) dst`	Decrement dst by 1	`SUB(.B) #1,dst`	*	*	*	*
`DECD(.B) dst`	Decrement dst by 2	`SUB(.B) #2,dst`	*	*	*	*
`DINT`	Disable interrupt	`BIC #8,SR`	–	–	–	–
`EINT`	Enable interrupt	`BIS #8,SR`	–	–	–	–
`INC(.B) dst`	Increment dst by 1	`ADD(.B) #1,dst`	*	*	*	*
`INCD(.B) dst`	Increment dst by 2	`ADD(.B) #2,dst`	*	*	*	*
`INV(.B) dst`	Invert dst	`XOR(.B) #-1,dst`	*	*	*	*
`NOP`	No operation	`MOV R3,R3`	–	–	–	–
`POP dst`	Pop operand from stack	`MOV @SP+,dst`	–	–	–	–
`RET`	Return from subroutine	`MOV @SP+,PC`	–	–	–	–
`RLA(.B) dst`	Shift left dst arithmetically	`ADD(.B) dst,dst`	*	*	*	*
`RLC(.B) dst`	Shift left dst logically through Carry	`ADDC(.B) dst,dst`	*	*	*	*
`SBC(.B) dst`	Subtract Carry from dst	`SUBC(.B) #0,dst`	*	*	*	*
`SETC`	Set Carry bit	`BIS #1,SR`	–	–	–	1
`SETN`	Set Negative bit	`BIS #4,SR`	–	1	–	–
`SETZ`	Set Zero bit	`BIS #2,SR`	–	–	1	–
`TST(.B) dst`	Test dst (compare with 0)	`CMP(.B) #0,dst`	0	*	*	1

[1] * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

Table 6-7. Emulated Instructions (continued)