MSP430 Addressing Modes

Ad	As	d/s	Register	Syntax	Description
0	00	*ds*	*n ≠ 3*	**R*n***	Register direct. The operand is the contents of R*n*. A_d=0
1	01	*ds*	*n ≠ 0, 2, 3*	**x(R*n*)**	Indexed. The operand is in memory at address R*n*+*x*.
-	10	*s*	*n ≠ 0, 2, 3*	**@R*n***	Register indirect. The operand is in memory at the address held in R*n*.
-	11	*s*	*n ≠ 0, 2, 3*	**@R*n*+**	Indirect auto-increment. As above, then the register is incremented by 1 or 2.
			Addressing modes using R0 (PC)		
1	01	ds	0 (PC)	***LABEL***	Symbolic. x(PC) The operand is in memory at address PC+*x*.
-	11	s	0 (PC)	***#x***	Immediate. @PC+ The operand is the next word in the instruction stream.
			Addressing modes using R2 (SR) and R3 (CG), special-case decoding		
1	01	ds	2 (SR)	***&LABEL***	Absolute. The operand is in memory at address *x*.
-	10	s	2 (SR)	**#4**	Constant. The operand is the constant 4.
-	11	s	2 (SR)	**#8**	Constant. The operand is the constant 8.
-	00	s	3 (CG)	**#0**	Constant. The operand is the constant 0.
-	01	s	3 (CG)	**#1**	Constant. The operand is the constant 1. There is no index word.
-	10	s	3 (CG)	**#2**	Constant. The operand is the constant 2.
-	11	s	3 (CG)	**#–1**	Constant. The operand is the constant –1.

MSP430 Instruction Set

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction
0	**0**	**0**	**1**	**0**	**0**	**opcode**			**B/W**	**As**		**register**				**Single-operand arithmetic**
0	0	0	1	0	0	0	0	0	B/W	As		register				**RRC** Rotate right through carry
0	0	0	1	0	0	0	0	1	0	As		register				**SWPB** Swap bytes
0	0	0	1	0	0	0	1	0	B/W	As		register				**RRA** Rotate right arithmetic
0	0	0	1	0	0	0	1	1	0	As		register				**SXT** Sign extend byte to word
0	0	0	1	0	0	1	0	0	B/W	As		register				**PUSH** Push value onto stack
0	0	0	1	0	0	1	0	1	0	As		register				**CALL** Subroutine call; push PC and move source to PC
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	**RETI** Return from interrupt; pop SR then pop PC

0	**0**	**1**	**condition**			**10-bit signed offset**	**Conditional jump; PC = PC + 2×offset**
0	0	1	0	0	0	10-bit signed offset	**JNE/JNZ** Jump if not equal/zero
0	0	1	0	0	1	10-bit signed offset	**JEQ/JZ** Jump if equal/zero
0	0	1	0	1	0	10-bit signed offset	**JNC/JLO** Jump if no carry/lower
0	0	1	0	1	1	10-bit signed offset	**JC/JHS** Jump if carry/higher or same
0	0	1	1	0	0	10-bit signed offset	**JN** Jump if negative
0	0	1	1	0	1	10-bit signed offset	**JGE** Jump if greater or equal
0	0	1	1	1	0	10-bit signed offset	**JL** Jump if less
0	0	1	1	1	1	10-bit signed offset	**JMP** Jump (unconditionally)

opcode				**source**	**Ad**	**B/W**	**As**	**destination**	**Two-operand arithmetic**
0	1	0	0	source	Ad	B/W	As	destination	**MOV** Move source to destination
0	1	0	1	source	Ad	B/W	As	destination	**ADD** Add source to destination
0	1	1	0	source	Ad	B/W	As	destination	**ADDC** Add source and carry to destination
0	1	1	1	source	Ad	B/W	As	destination	**SUBC** Subtract source from destination (with carry)
1	0	0	0	source	Ad	B/W	As	destination	**SUB** Subtract source from destination
1	0	0	1	source	Ad	B/W	As	destination	**CMP** Compare (pretend to subtract) source from destination
1	0	1	0	source	Ad	B/W	As	destination	**DADD** Decimal add source to destination (with carry)
1	0	1	1	source	Ad	B/W	As	destination	**BIT** Test bits of source AND destination
1	1	0	0	source	Ad	B/W	As	destination	**BIC** Bit clear (dest &= ~src)
1	1	0	1	source	Ad	B/W	As	destination	**BIS** Bit set (logical OR)
1	1	1	0	source	Ad	B/W	As	destination	**XOR** Exclusive or source with destination
1	1	1	1	source	Ad	B/W	As	destination	**AND** Logical AND source with destination (dest &= src)

Special Registers: PC (Program Counter)=*R0*; SP (Stack Pointer)=*R1*; SR (Status Register)=*R2*; CG (Constants Generator)=*R3*;

* changes based on op
\- not affected

No.	Emulated	Mnemonic	Operand(s)	Description		V	N	Z	C
1	⊙	ADC(.B)	dst	Add C to destination	dst+C→dst	*	*	*	*
2		ADD(.B)	src,dst	Add source to destination	src+dst→dst	*	*	*	*
3		ADDC(.B)	src,dst	Add source and C to destination	src+dst+C→dst	*	*	*	*
4		AND(.B)	src,dst	AND source and destination	src .and. dst → dst	0	*	*	*
5		BIC(.B)	src,dst	Clear bits in destination	not.src .and. dst → dst	-	-	-	-
6		BIS(.B)	src,dst	Set bits in destination	src .or. dst → dst	-	-	-	-
7		BIT(.B)	src,dst	Test bits in destination	src .and. dst	0	*	*	*
8	⊙	BR	dst	Branch to destination	dst → PC	-	-	-	-
9		CALL	dst	Call destination	PC+2 → stack, dst → PC	-	-	-	-
10	⊙	CLR(.B)	dst	Clear destination	0 → dst	-	-	-	-
11	⊙	CLRC		Clear C	0→C	-	-	-	0
12	⊙	CLRN		Clear N	0→N	-	0	-	-
13	⊙	CLRZ		Clear Z	0→Z	-	-	0	-
14		CMP(.B)	src,dst	Compare source and destination	dst - src	*	*	*	*
15	⊙	DADC(.B)	dst	Add C decimally to destination	dst + C → dst (decimally)	*	*	*	*
16		DADD(.B)	src,dst	Add source and C decimally to dst	src + dst + C → dst (decimally)	*	*	*	*
17	⊙	DEC(.B)	dst	Decrement destination	dst - 1 → dst	*	*	*	*
18	⊙	DECD(.B)	dst	Double-decrement destination	dst - 2 → dst	*	*	*	*
19	⊙	DINT		Disable interrupts	0 → GIE	-	-	-	-
20	⊙	EINT		Enable interrupts	1 → GIE	-	-	-	-
21	⊙	INC(.B)	dst	Increment destination	dst +1 → dst	*	*	*	*
22	⊙	INCD(.B)	dst	Double-increment destination	dst+2 → dst	*	*	*	*
23	⊙	INV(.B)	dst	Invert destination	.not.dst → dst	*	*	*	*
24		JC/JHS	label	Jump if C set/Jump if higher or same		-	-	-	-
25		JEQ/JZ	label	Jump if equal/Jump if Z set		-	-	-	-
26		JGE	label	Jump if greater or equal		-	-	-	-
27		JL	label	Jump if less		-	-	-	-
28		JMP	label	Jump	PC + 2 × offset → PC	-	-	-	-
29		JN	label	Jump if N set		-	-	-	-
30		JNC/JLO	label	Jump if C not set/Jump if lower		-	-	-	-
31		JNE/JNZ	label	Jump if not equal/Jump if Z not set		-	-	-	-
32		MOV(.B)	src,dst	Move source to destination	src → dst	-	-	-	-
33	⊙	NOP		No operation		-	-	-	-
34	⊙	POP(.B)	dst	Pop item from stack to destination	@SP → dst, SP+2 → SP	-	-	-	-
35		PUSH(.B)	src	Push source onto stack	SP - 2 → SP, src → @SP	-	-	-	-
36	⊙	RET		Return from subroutine	@SP → PC, SP + 2 → SP	-	-	-	-
37		RETI		Return from interrupt		*	*	*	*
38	⊙	RLA(.B)	dst	Rotate left arithmetically		*	*	*	*
39	⊙	RLC(.B)	dst	Rotate left through C		*	*	*	*
40		RRA(.B)	dst	Rotate right arithmetically		0	*	*	*
41		RRC(.B)	dst	Rotate right through C		*	*	*	*
42	⊙	SBC(.B)	dst	Subtract not(C) from destination	dst + 0FFFFh + C → dst	*	*	*	*
43	⊙	SETC		Set C	1→C	-	-	-	1
44	⊙	SETN		Set N	1→N	-	1	-	-
45	⊙	SETZ		Set Z	1→Z	-	-	1	-
46		SUB(.B)	src,dst	Subtract source from destination	dst + .not.src + 1 → dst	*	*	*	*
47		SUBC(.B)	src,dst	Subtract source and not(C) from dst	dst + .not.src + C → dst	*	*	*	*
48		SWPB	dst	Swap bytes		-	-	-	-
49		SXT	dst	Extend sign		0	*	*	*
50	⊙	TST(.B)	dst	Test destination	dst+0FFFFh+1	0	*	*	1
51		XOR(.B)	src,dst	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

MSP430 Emulated Instructions

Mnemonic	Operation	Emulation	Description
Arithmetic Instructions			
ADC(.B or .W) dst	**dst+C→dst**	**ADDC(.B or .W) #0,dst**	Add carry to destination
DADC(.B or .W) dst	**dst+C→dst (decimally)**	**DADD(.B or .W) #0,dst**	Decimal add carry to destination
DEC(.B or .W) dst	**dst-1→dst**	**SUB(.B or .W) #1,dst**	Decrement destination
DECD(.B or .W) dst	**dst-2→dst**	**SUB(.B or .W) #2,dst**	Decrement destination twice
INC(.B or .W) dst	**dst+1→dst**	**ADD(.B or .W) #1,dst**	Increment destination
INCD(.B or .W) dst	**dst+2→dst**	**ADD(.B or .W) #2,dst**	Increment destination twice
SBC(.B or .W) dst	**dst+0FFFFh+C→dst** **dst+0FFh→dst**	**SUBC(.B or .W) #0,dst**	Subtract source and borrow /.NOT. carry from dest.

Mnemonic	Operation	Emulation	Description
Logical and Register Control Instructions			
INV(.B or .W) dst	**.NOT.dst→dst**	**XOR(.B or .W) #0(FF)FFh,dst**	Invert bits in destination
RLA(.B or .W) dst	**C←MSB←MSB-1** **LSB+1←LSB←0**	**ADD(.B or .W) dst,dst**	Rotate left arithmetically
RLC(.B or .W) dst	**C←MSB←MSB-1** **LSB+1←LSB←C**	**ADDC(.B or .W) dst,dst**	Rotate left through carry
Program Flow Control			
BR dst	**dst→PC**	**MOV dst,PC**	Branch to destination
DINT	**0→GIE**	**BIC #8,SR**	Disable (general) interrupts
EINT	**1→GIE**	**BIS #8,SR**	Enable (general) interrupts
NOP	**None**	**MOV #0,R3**	No operation
RET	**@SP→PC SP+2→SP**	**MOV @SP+,PC**	Return from subroutine

Mnemonic	Operation	Emulation	Description
Data Instructions			
CLR(.B or .W) dst	**0→dst**	**MOV(.B or .W) #0,dst**	Clear destination
CLRC	**0→C**	**BIC #1,SR**	Clear carry flag
CLRN	**0→N**	**BIC #4,SR**	Clear negative flag
CLRZ	**0→Z**	**BIC #2,SR**	Clear zero flag
POP(.B or .W) dst	**@SP→temp** **SP+2→SP** **temp→dst**	**MOV(.B or .W) @SP+,dst**	Pop byte/word from stack to destination
SETC	**1→C**	**BIS #1,SR**	Set carry flag
SETN	**1→N**	**BIS #4,SR**	Set negative flag
SETZ	**1→Z**	**BIS #2,SR**	Set zero flag
TST(.B or .W) dst	**dst + 0FFFFh + 1** **dst + 0FFh + 1**	**CMP(.B or .W) #0,dst**	Test destination

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx																
4xxx																
8xxx																
Cxxx																
1xxx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14xx																
18xx																
1Cxx																
20xx	JNE/JNZ															
24xx	JEQ/JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															

asm code 0(Rn) = @Rn when source	machine code	opcode	source Reg	Ad (dest)	B/~W	As (Src)	destination	* IF NEEDED Additional Data 1	* IF NEEDED Additional Data 2
mov.w R5, R6	4506	0 1 0 0	5	0	0	0 0	6	-	-
add.b R4, R8	5448	0 1 0 1	4	0	1	0 0	8	-	-
bit.w @R7, R12	B72C	1 0 1 1	7	0	0	1 0	C		
bit.b 0(R7), R15	B76F	1 0 1 1	7	0	1	1 0	F		
bic.b 2(R8), 0(R8)	C8D8 0002 0000	1 1 0 0	8	1	1	0 1	8	0002	0000
bic.w 2(R8), 4(R7)	C897 0002 0004	1 1 0 0	8	1	0	0 1	7		
bis.w #0xAAAA, R11	D03B AAAA	1 1 0 1	0	0	0	1 1	B	AAAA	
rra.b R10	114A	0 0 0 1	0001	0	1	0 0	A		
rrc.w R11	100B	0 0 0 1	0000	0	0	0 0	B		
jmp *LABEL*	3 [11xx xxx xxxx]	0 0 1 1	11xx	x	x	x x	(offset - Δ inst. words)		
cmp R13, 2(R8)	9D88 0002	1 0 0 1	D	1	0	0 0	8	0002	
jnz *LABEL*	2 [00xx xxxx xxxx]	0 0 1 0	00xx	x	x	x x	(offset - Δ inst. words)		
rla.b R10	5A4A								
add.b R10, R10	🡸 (emulated code) 🡹	0 1 0 1	A	0	1	0 0	A		
rlc.w R11	6B0B								
addc.w R11, R11	🡸 (emulated code) 🡹	0 1 1 0	B	0	0	0 0	B		
inv.w R10	E33A								
xor.w #0xFFFF, R10	🡸 (emulated code) 🡹	1 1 1 0	3	0	0	1 1	A		
inc.w R11	531B								
add.w #0x0001, R11	🡸 (emulated code) 🡹	0 1 0 1	3	0	0	0 1	B		

SR Bit — *After the operation is complete, the SR bits are set based on these conditions*

V — oVerflow: "signed" overflow - leading digit (MSB) switched from (+) to (-) or from (-) to (+)

examples in binary

add.w	*inc.w*	*incd.b*	*add.b*
0100 0000 0000 0000	0111 1111 1111 1111	0111 1110	1011 0000
+0111 0100 0010 1011			+ 1000 1000
1011 0100 0010 1011	1000 0000 0000 0000	0000 0000 1000 0000	0000 0000 0011 1000

N — Negative: The leading digit (MSB) is 1 - the number "negative" if you are using signed numbers

examples in binary

cmp.w	*inv.w*	*dec.b*	*sub.b*
0000 0010 0101 0101	0101 0000 1111 1010	0000 0000 0000 0000	0000 0101
1111 1100 1110 0000			- 0000 0111
1111 1111 0011 0101	1010 1111 0000 0101	0000 0000 1111 1111	0000 0000 1111 1110

Z — Zero: All the bits are 0

examples in binary

sub.w	*rra.w*	*and.b*	*xor.b*
0000 0010 0101 0101	0000 0000 0000 0001	1010 1010	0010 1101
-0000 0010 0101 0101		0101 0101	0010 1101
0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000

C — Carry: "unsigned" overflow - there was not enough space - a digit got knocked off ***either end***

examples in binary

add.w	*rra.w*	*rlc.b*	*rrc.b*
1000 0000 0000 0000	0000 0100 0000 0001	0000 0000 1000 0010	0000 0000 1101 1011
1000 0000 0000 0001			
(1) 0000 0000 0000 0001	0000 0010 0000 0000 (1)	(1) 0000 0000 0000 0100	0000 0000 0110 1101 (1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled, SMCLK, ACLK are active
0	1	0	1	LPM1	CPU, MCLK are disabled. DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active.
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled. DC generator remains enabled. ACLK is active.
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled. DC generator disabled. ACLK is active.
1	1	1	1	LPM4	CPU and all clocks disabled

Table 15-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0

BRCLK Frequency [Hz]	Baud Rate [Baud]	UCBRx	UCBRSx	UCBRFx	Maximum TX Error [%]		Maximum RX Error [%]	
1,000,000	9600	104	1	0	-0.5	0.6	-0.9	1.2
1,000,000	19200	52	0	0	-1.8	0	-2.6	0.9
1,000,000	38400	26	0	0	-1.8	0	-3.6	1.8
1,000,000	56000	17	7	0	-4.8	0.8	-8.0	3.2
1,000,000	115200	8	6	0	-7.8	6.4	-9.7	16.1
1,000,000	128000	7	7	0	-10.4	6.4	-18.0	11.6
1,000,000	256000	3	7	0	-29.6	0	-43.6	5.2

Table 15-5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1

BRCLK Frequency [Hz]	Baud Rate [Baud]	UCBRx	UCBRSx	UCBRFx	Maximum TX Error [%]		Maximum RX Error [%]	
1,000,000	9600	6	0	8	-1.8	0	-2.2	0.4
1,000,000	19200	3	0	4	-1.8	0	-2.6	0.9
1,000,000	57600	1	7	0	-34.4	0	-33.4	0

15.4.1 UCAxCTL0, USCI_Ax Control Register 0

7	6	5	4	3	2	1	0
UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCPEN Bit 7 ***Parity enable***
0 Parity disabled.
1 Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.

UCPAR Bit 6 ***Parity select. UCPAR is not used when parity is disabled.***
0 Odd parity *1 Even parity*

UCMSB Bit 5 ***MSB first select. Controls the direction of the receive and transmit shift register.***
0 LSB first *1 MSB first*

UC7BIT Bit 4 ***Character length. Selects 7-bit or 8-bit character length.***
0 8-bit data *1 7-bit data*

UCSPB Bit 3 ***Stop bit select. Number of stop bits.***
0 One stop bit *1 Two stop bits*

UCMODEx Bits 2-1 ***USCI mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0.***
00 UART mode
01 Idle-line multiprocessor mode
10 Address-bit multiprocessor mode
11 UART mode with automatic baud rate detection

UCSYNC Bit 0 ***Synchronous mode enable***
0 Asynchronous mode *1 Synchronous mode*

15.4.2 UCAxCTL1, USCI_Ax Control Register 1

7	6	5	4	3	2	1	0
UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

UCSSELx Bits 7-6 ***USCI clock source select. These bits select the BRCLK source clock.***
00 UCLK *01 ACLK*
10 SMCLK *11 SMCLK*

UCRXEIE Bit 5 ***Receive erroneous-character interrupt-enable***
0 Erroneous characters rejected and UCAxRXIFG is not set
1 Erroneous characters received will set UCAxRXIFG

UCBRKIE Bit 4 ***Receive break character interrupt-enable***
0 Received break characters do not set UCAxRXIFG.
1 Received break characters set UCAxRXIFG.

UCDORM Bit 3 ***Dormant. Puts USCI into sleep mode.***
0 Not dormant. All received characters will set UCAxRXIFG.
1 Dormant. Only characters that are preceded by an idle-line or with address bit set will set UCAxRXIFG. In UART mode with automatic baud rate detection only the combination of a break and synch field will set UCAxRXIFG.

UCTXADDR Bit 2 ***Transmit address. Next frame to be transmitted will be marked as address depending on the selected multiprocessor mode.***
0 Next frame transmitted is data
1 Next frame transmitted is an address

UCTXBRK Bit 1 ***Transmit break. Transmits a break with the next write to the transmit buffer. In UART mode with automatic baud rate detection 055h must be written into UCAxTXBUF to generate the required break/synch fields. Otherwise 0h must be written into the transmit buffer.***
0 Next frame transmitted is not a break
1 Next frame transmitted is a break or a break/synch

UCSWRST Bit 0 ***Software reset enable***
0 Disabled. USCI reset released for operation.
1 Enabled. USCI logic held in reset state.

15.4.3 UCAxBR0, USCI_Ax Baud Rate Control Register 0 &
15.4.4 UCAxBR1, USCI_Ax Baud Rate Control Register 1

7	6	5	4	3	2	1	0
UCBRx							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx 7-0 *Clock prescaler setting of the Baud rate generator. The 16-bit value of (UCAxBR0 + UCAxBR1 × 256) forms the prescaler value.*

15.4.5 UCAxMCTL, USCI_Ax Modulation Control Register

7	6	5	4	3	2	1	0
UCBRFx				UCBRSx			UCOS16
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCBRFx Bits 7-4 ***First modulation stage select.*** *These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. Table 15-3 shows the modulation pattern.*

UCBRSx Bits 3-1 ***Second modulation stage select.*** *These bits determine the modulation pattern for BITCLK. Table 15-2 shows the modulation pattern.*

UCOS16 Bit 0 ***Oversampling mode enabled***
0 Disabled *1 Enabled*

15.4.7 UCAxRXBUF, USCI_Ax Receive Buffer Register

7	6	5	4	3	2	1	0
UCRXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCRXBUFx Bits 7-0 ***The receive-data buffer is user accessible and contains the last received character from the receive shift register.*** *Reading UCAxRXBUF resets the receive-error bits, the UCADDR or UCIDLE bit, and UCAxRXIFG. In 7-bit data mode, UCAxRXBUF is LSB justified and the MSB is always reset.*

15.4.8 UCAxTXBUF, USCI_Ax Transmit Buffer Register

7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCTXBUFx Bits 7-0 ***The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted on UCAxTXD.*** *Writing to the transmit data buffer clears UCAxTXIFG. The MSB of UCAxTXBUF is not used for 7-bit data and is reset.*

15.4.12 IE2, Interrupt Enable Register 2

7	6	5	4	3	2	1	0
						UCA0TXIE	UCA0RXIE
						rw-0	rw-0

Bits 7-2 *These bits may be used by other modules (see the device-specific data sheet).*

UCA0TXIE Bit 1 ***USCI_A0 transmit interrupt enable***
0 Interrupt disabled *1 Interrupt enabled*

UCA0RXIE Bit 0 ***USCI_A0 receive interrupt enable***
0 Interrupt disabled *1 Interrupt enabled*

15.4.13 IFG2, Interrupt Flag Register 2

7	6	5	4	3	2	1	0
						UCA0TXIFG	UCA0RXIFG
						rw-1	rw-0

Bits 7-2 *These bits may be used by other modules (see the device-specific data sheet).*

UCA0TXIE Bit 1 *USCI_A0 transmit interrupt flag. UCA0TXIFG is set when UCA0TXBUF is empty.*
0 No interrupt pending ***1 Interrupt pending***

UCA0RXIE Bit 0 *USCI_A0 receive interrupt flag. UCA0RXIFG is set when UCA0RXBUF has received a complete character.*
0 No interrupt pending ***1 Interrupt pending***

6.7 What Happens when an Interrupt Is Requested?

SP-2; PC→SP.
SP-2; SR→SP

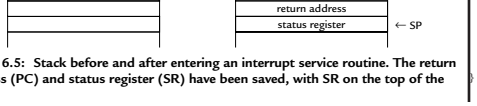


Figure 6.5: Stack before and after entering an interrupt service routine. The return address (PC) and status register (SR) have been saved, with SR on the top of the stack.

Serial_MSP.c

```
void serial_init(void) {
    P1SEL = 0x04;                       // Select UART as the pin function
    P1SEL2 = 0x04;
    UCA0CTL1 |= UCSWRST;                // Disable UART module for configuration
    UCA0CTL0 = 0x00;                    // No parity, LSB first, 8-bit data, 1 stop bit, UART, Asynchronous
    UCA0CTL1 |= 0xC0;                   // SMCLK source, keep in reset state
    UCA0BR0 = 104;                      // 9600 Baud rate - Assumes 1 MHz clocks (1000000 / 9600 = 104)
    UCA0MCTL = 0x02;                    // 2nd Stage modulation = 1, Oversampling off
    IE2 = 0x00;                         // Interrupts disabled
    UCA0CTL1 &= ~UCSWRST;               // Enable UART module
}
void serial_charTX(char c) {
  while( !(IFG2 & UCA0TXIFG) );         // Wait until the transmit buffer is empty
  UCA0TXBUF = c;                        // Send the character through the Xmit buffer
}
void clock_init(void) {
    DCOCTL = 0x00;
    BCSCTL1 = CALBC1_1MHZ;              // Calibrate to 1 MHz
    DCOCTL = CALDCO_1MHZ;
}
```

mainTX.c

```
#include <msp430g2553.h>
#include "serial_msp.h"
void main(void) {
    WDTCTL = WDTHOLD + WDTPW;  // Stop Watchdog timer
    P1DIR |= BIT0 + BIT6;      // Configure LED pins as outputs
    P1OUT &= ~(BIT0 + BIT6);   // Turn off LEDs
    P2DIR &= ~(BIT0 + BIT1 + BIT2);  // Make button pins as inputs
    clock_init();              // Set clock frequency to 1 MHz
    serial_init();             // Initialize UART module
    while(1) {
        if( (P2IN & BIT2) == BIT2) {
            serial_charTX('r');
            P1OUT |= (BIT0 + BIT6);     // Turn on BOTH LEDs
            __delay_cycles(1000000);
        }
        if( (P2IN & BIT1) == BIT1) {
            serial_charTX('i');
            P1OUT |= BIT6;              // Turn On green LED
            P1OUT &= ~BIT0;             // Turn Off red LED
            __delay_cycles(1000000);
        }
        if( (P2IN & BIT0) == BIT0) {
            serial_charTX('d');
            P1OUT |= BIT0;              // Turn On red LED
            P1OUT &= ~BIT6;             // Turn Off green LED
            __delay_cycles(1000000);
        } else {
            P1OUT &= ~(BIT0 + BIT6);    // Turn on LEDs
        }
    }
}
```

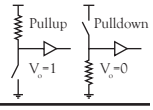
Serial_MSP_Interrupt.c

```
void serial_init_inter(void) {
    P1SEL = 0x02;
    P1SEL2 = 0x02;
    UCA0CTL1 |= UCSWRST;                // Disable UART module for configuration
    UCA0CTL0 = 0x00;                    // No parity, LSB first, 8-bit data, 1 stop bit, UART, Asynchronous
    UCA0CTL1 |= 0xc0;                   // SMCLK source, keep in reset state
    UCA0BR0 = 104;                      // 9600 Baud rate  - Assumes 1 MHz clock
    UCA0MCTL = 0x02;                    // 2nd Stage modulation = 1, Oversampling off
    UCA0CTL1 &= ~UCSWRST;               // Enable UART module
}
char serial_charRX(void) {
    while( !(IFG2 & UCA0RXIFG) );       // Wait until a character has been received
    return UCA0RXBUF;                   // Return received character
}
void clock_init(void) {
    DCOCTL = 0x00;
    BCSCTL1 = CALBC1_1MHZ;      // Calibrate to 1 MHz
    DCOCTL = CALDCO_1MHZ;
    BCSCTL1 |= XT2OFF + DIVA_0;
    BCSCTL3 = XT2S_0 + LFXT1S_2 + XCAP_1;
}
```

mainRX.c

```
#include <msp430g2553.h>
#include "Serial_MSP_Interrupt.h"
void main(void) {
    WDTCTL = WDTHOLD + WDTPW;  // Stop Watchdog timer
    P1DIR |= BIT0 + BIT6;      // Configure LED pins as outputs
    P1OUT &= ~(BIT0 + BIT6);   // Turn off LEDs
    clock_init();              // Set clock frequency to 1 MHz
    serial_init_inter();       // Initialize UART module
    IFG2 &= ~(UCA0RXIFG);
    IE2 |= UCA0RXIE;           // Enable UART interrupt
    __bis_SR_register(GIE);
    while(1) { }
}

#pragma vector = USCIAB0RX_VECTOR //interrupt vector routine
__interrupt void RX_Function(void) {
    char c;
    while( !(IFG2 & UCA0RXIFG) ); // This should make the CPU wait since the flag should be set already
    c = UCA0RXBUF;                // Store received character
    switch(c) {
        case 'r':                 // Reset the counter
            P1OUT |= (BIT0 + BIT6);   // Turn off LEDs
            break;
        case 'i':
            P1OUT |= BIT6;            // Turn On green LED
            P1OUT &= ~BIT0;           // Turn Off red LED
            break;
        case 'd':
            P1OUT |= BIT0;            // Turn On red LED
            P1OUT &= ~BIT6;           // Turn Off green LED
            break;
        default:
            P1OUT &= ~(BIT0 + BIT6);  // Turn off LEDs
            break;
    }
}
```



ADC_MSP.c

```
void adc_init() {     // Initialize ADC
    P1DIR &= 0x0001;                    // Set Pins as inputs
    ADC10CTL0 = ADC10ON + ADC10SHT_0 + SREF_0;  // configure ADC module
    ADC10CTL1 = CONSEQ_0 + ADC10SSEL_0 + ADC10DIV_0 + SHS_0 + INCH_0;
    ADC10AE0 = 0x0001;                  // Enable analog function in pins
}
int adc_measureAll(void) {              // Return the sum of all ADC channels
    return (adc_measure(X_AXIS) + adc_measure(Y_AXIS) + adc_measure(Z_AXIS));
}
int adc_measure(int channel) {          // measure selected ADC Channel: X_AXIS, Y_AXIS, Z_AXIS
    ADC10CTL0 &= ~ENC;                  // Disable ADC
    // Configure the MUX channel to sample
    ADC10CTL1 = CONSEQ_0 + ADC10SSEL_0 + ADC10DIV_0 + SHS_0 + (channel << 12);
    ADC10CTL0 |= ENC + ADC10SC;         // Enable ADC and start conversion
    while ( (ADC10CTL0 & ADC10IFG) == 0);   // Wait until the conversion is finished
    ADC10CTL0 &= ~ENC;                  // Disable ADC before retrieving the conversion from memory
    return ADC10MEM;                    // Return measurement value
}
```

mainADC.c

```
#include <msp430g2553.h>
#include "adc_msp.h"
#include "lcd_lib.h"
void main(void) {
    WDTCTL = WDTPW + WDTHOLD;// Stop Watchdog
    adc_init(); //initialize A to D converter
    lcd_init(); //initialize LCD
    P1DIR |= BIT6;
    send_cmd(0x0C); //disable the blinking cursor
    while(1) {
        int x;
        x=adc_measure(0); //use 0 for channel 0 (P1.0)
        if(x>=512){
            P1OUT |= BIT6;
        } else {
            P1OUT &= ~BIT6;
        }
        delay_ms(250);
        gotoXy(0,0);
        write_int(x);
        write_string(" Value");
        gotoXy(0, 1); // Move cursor to 2nd row
        write_int(x);
        write_string(" Value");
    }
}
```

C → asm:

```
while(i!=3)
{
    do_it;
}
```

```
        go_back CMP   #0x03, J(R7)
                JEQ   done
                CALL  #do_it
                JMP   go_back
done
        . . .
do_it:
```

```
for(i=0; i<64;i++)
{
    do_rep;
}
```

```
        go_back CLR   J(R7)
                MOV   J(R7), R5
                CMP   #0x64, R5
                JGE   done
                CALL  #do_rep
                INC   J(R7)
                JMP   go_back
done
        . . .
do_rep:
```

22.3.1 ADC10CTL0, ADC10 Control Register 0

15	14	13	12	11	10	9	8
SREFx			ADC10SHTx		ADC10SR	REFOUT	REFBURST
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ENC = 0

SREFx Bits 15-13 Select reference.
000 V_{R+} = V_{CC} and V_{R-} = V_{SS}
001 V_{R+} = V_{REF+} and V_{R-} = V_{SS}
010 V_{R+} = V_{eREF+} and V_{R-} = V_{SS}. Devices with V_{eREF+} only.
011 V_{R+} = Buffered V_{eREF+} and V_{R-} = V_{SS}. Devices with V_{eREF+} pin only.
100 V_{R+} = V_{CC} and V_{R-} = V_{REF-} / V_{eREF-}. Devices with V_{eREF-} pin only.
101 V_{R+} = V_{REF+} and V_{R-} = V_{REF-} / V_{eREF-}. Devices with V_{eREF-} pin only.
110 V_{R+} = V_{eREF+} and V_{R-} = V_{REF-} / V_{eREF-}. Devices with V_{eREF+} pins only.
111 V_{R+} = Buffered V_{eREF+} and V_{R-} = V_{REF-} / V_{eREF-}. Devices with V_{eREF+} pins only.

ADC10SHTx Bits 12-11 ADC10 sample-and-hold time
00 4 × ADC10CLKs
01 8 × ADC10CLKs
10 16 × ADC10CLKs
11 64 × ADC10CLKs

ADC10SR Bit 10 ADC10 sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer.
0 Reference buffer supports up to ~200 ksps
1 Reference buffer supports up to ~50 ksps

REFOUT Bit 9 Reference output
0 Reference output off
1 Reference output on. Devices with V_{eREF+} / V_{REF+} pin only.

REFBURST Bit 8 Reference burst.
0 Reference buffer on continuously
1 Reference buffer on only during sample-and-conversion

MSC Bit 7 Multiple sample and conversion. Valid only for sequence or repeated modes.
0 The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion.
1 The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed

REF2_5V Bit 6 Reference-generator voltage. REFON must also be set.
0 1.5 V
1 2.5 V

REFON Bit 5 Reference generator on
0 Reference off
1 Reference on

ADC10ON Bit 4 ADC10 on
0 ADC10 off
1 ADC10 on

ADC10IE Bit 3 ADC10 interrupt enable
0 Interrupt disabled
1 Interrupt enabled

ADC10IFG Bit 2 ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed.
0 No interrupt pending
1 Interrupt pending

ENC Bit 1 Enable conversion
0 ADC10 disabled
1 ADC10 enabled

ADC10SC Bit 0 Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically.
0 No sample-and-conversion start
1 Start sample-and-conversion

22.3.2 ADC10CTL1, ADC10 Control Register 1

15	14	13	12	11	10	9	8
INCHx				SHSx		ADC10DF	ISSH
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10DIVx			ADC10SSELx		CONSEQx		ADC10BUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-0

Can be modified only when ENC = 0

INCHx Bits 15-12 Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions. Only available ADC channels should be selected. See device specific datasheet.
0000 A0
0001 A1
0010 A2
0011 A3
0100 A4
0101 A5
0110 A6
0111 A7
1000 V_{eREF-}
1001 V_{REF+}/V_{eREF-}
1010 Temperature sensor
1011 (V_{CC} - V_{SS}) / 2
1100 (V_{CC} - V_{SS}) / 2, A12 on MSP430F22xx devices
1101 (V_{CC} - V_{SS}) / 2, A13 on MSP430F22xx devices
1110 (V_{CC} - V_{SS}) / 2, A14 on MSP430F22xx devices
1111 (V_{CC} - V_{SS}) / 2, A15 on MSP430F22xx devices

SHSx Bits 11-10 Sample-and-hold source select.
00 ADC10SC bit
01 Timer_A.OUT1[1]
10 Timer_A.OUT0[1]
11 Timer_A.OUT2 (Timer_A.OUT1 on MSP430F20x0, MSP430G2x31, and MSP430G2x30 devices)[1]

ADC10DF Bit 9 ADC10 data format
0 Straight binary
1 2s complement

ISSH Bit 8 Invert signal sample-and-hold
0 The sample-input signal is not inverted.
1 The sample-input signal is inverted.

ADC10DIVx Bits 7-5 ADC10 clock divider
000 /1
001 /2
010 /3
011 /4
100 /5
101 /6
110 /7
111 /8

ADC10SSELx Bits 4-3 ADC10 clock source select
00 ADC10OSC
01 ACLK
10 MCLK
11 SMCLK

CONSEQx Bits 2-1 Conversion sequence mode select
00 Single-channel-single-conversion
01 Sequence-of-channels
10 Repeat-single-channel
11 Repeat-sequence-of-channels

ADC10BUSY Bit 0 ADC10 busy. This bit indicates an active sample or conversion operation
0 No operation is active.
1 A sequence, sample, or conversion is active.

[1] Timer triggers are from Timer0_A if more than one timer module exists on the device.

C → asm: (I, K = integer; index for stack frame pointer)

```
if (b>=5&&c==2) {
    do_this;
} else {
    do_that;
}
```

```
        CMP   #0x05, J(R7)
        JL    do_that
        CMP   #0x02, K(R7)
        JNE   do_that
        JMP   do_this
do_that
        . . .
        JMP   done
do_this
        . . .
        JMP   done
done
```

```
switch (myByte) {
    case a:
        do_A;
        break;
    case b:
        do_B;
        break;
    default:
        do_D;
        break;
}
```

```
        CMP   #a, &myByte
        JNE   sw_01
        CALL  #do_A
        JMP   sw_end
sw_01:
        CMP   #b, &myByte
        JNE   sw_def
        CALL  #do_B
        JMP   sw_end
sw_def:
        CALL  #do_D
        JMP   sw_end
sw_end:
```

```
int i;
for(i=10; i>0; i--)
{
    do_dot();
    delay();
    do_dash();
    delay();
}
```

```
        .bss  i, 2 ;2 bytes global variable i
        mov.w #10,&i
fl_ck:  call  #do_dot
        call  #delay
        call  #do_dash
        call  #delay
        dec   &i
        jnz   fl_ck
for_done:
```