

Department of Electrical and Computer Engineering

College of Engineering and Applied Sciences

WESTERN MICHIGAN UNIVERSITY



# ECE 2510 Introduction to Microprocessors



## TI MSP430 Notes

Dr. Bradley J. Bazuin

Associate Professor

Department of Electrical and Computer Engineering

College of Engineering and Applied Sciences

# MSP430

---

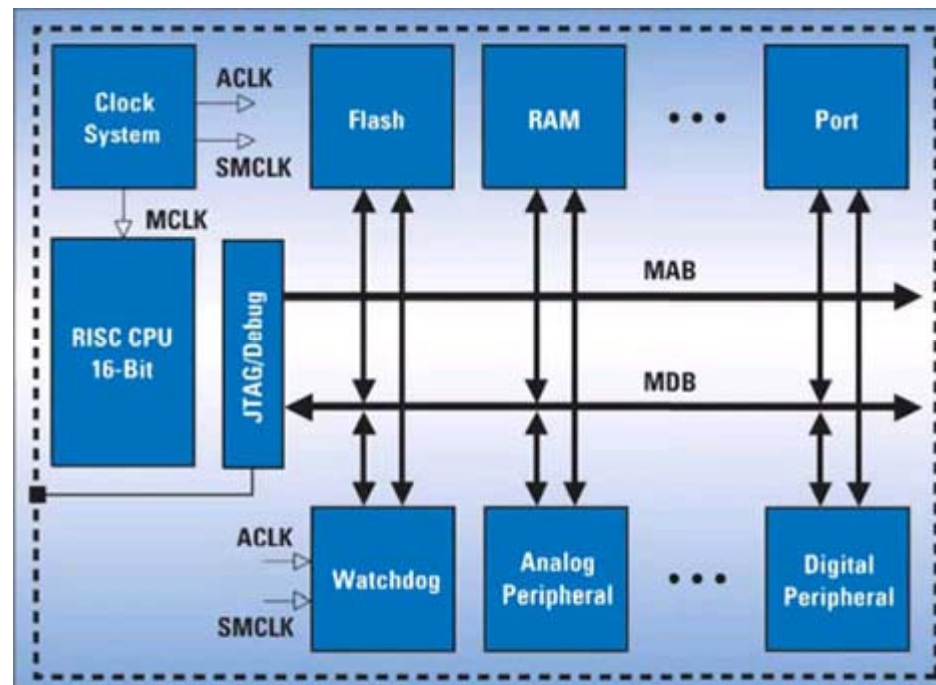
- What is the MSP430?

The MSP430 family of ultra-low-power 16-bit RISC mixed-signal processors from Texas Instruments (TI) provides the ultimate solution for battery-powered measurement applications. Using leadership in both mixed-signal and digital technologies, TI has created the MSP430 which enables system designers to simultaneously interface to analog signals, sensors and digital components while maintaining unmatched low power.

Typical applications include utility metering, portable instrumentation, intelligent sensing, and consumer electronics.

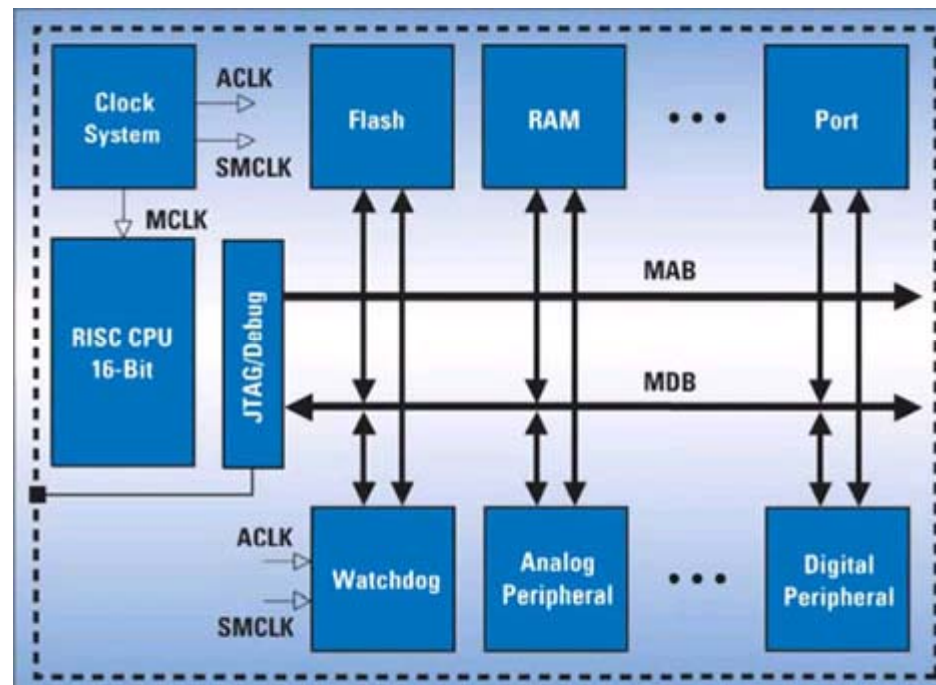
# Basic Architecture

- 16-bit microcontroller
  - RISC Architecture – lots of registers, simple load store memory access, simple instruction that can be executed in a single clock cycle (smaller baby-steps than the HC12!?)



# Memory Mapped Peripherals

- I/O or peripheral devices
  - Both analog (ADC and DAC) and digital functions
- Clocking System



# MSP430 Memory and Peripherals

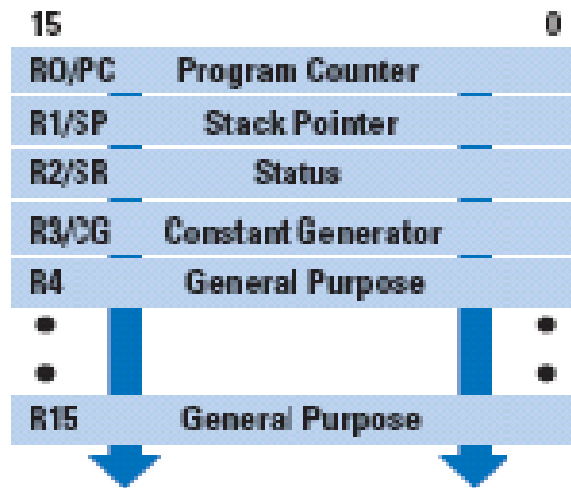
---

- Device Configuration
  - 1-KB to 120-KB ISP Flash
  - RAM up to 10 KB
  - 14- to 100-pin options
- Integrated Peripherals
  - 10-/12-bit SAR ADC
  - 16-bit Sigma Delta ADC
  - 12-bit DAC
  - Comparator
  - LCD driver
  - Supply Voltage Supervisor (SVS)
  - Operational amplifiers
  - 16-bit and 8-bit timers
  - Watchdog timer
  - UART/LIN
  - I2C
  - SPI
  - IrDA
  - Hardware multiplier
  - DMA controller
  - Temperature sensor

# RISC Architecture

---

- 16 fully addressable, single-cycle 16-bit CPU registers
- 27 easy-to-understand instructions and
- Seven consistent-addressing modes.



*The MSP430 CPU core with sixteen 16-bit registers, 27 single-cycle instructions and seven addressing modes results in higher processing efficiency and code density.*

# MSP430 Family

---

- Multiple devices with different numbers of pins and peripherals
  - # MSP430x1xx
    - The MSP430x1xx are Flash/ ROM based MCUs offer 1.8V to 3.6V operation, up to 60kB, 8MIPs with Basic Clock. This family of MCUs offers a wide range of capabilities from a simple low power controller with a comparator, to complete systems on a chip including high-performance data converters, interfaces and multiplier.
  - # MSP430F2xx
    - The new ultra-low-power MSP430F2xx family increases performance up to 16 MHz. Additional enhancements of MSP430F2xx, include an integrated  $\pm 1\%$  on-chip very lowpower oscillator, software-selectable, internal pullup/pull-down resistors and increased number of analog inputs. The in-system programmable Flash has also been improved with smaller 64-byte segments and a lower 2.2-V programming voltage allowing the elimination of external EEPROMs in most systems.
    - Available in low-pin count options.
  - # MSP430x4xx
    - The ultra-low-power MSP430x4xx devices offer 1.8V-3.6V operation, up to 120kB/ Flash/ ROM 8MIPS with FLL + SVS along with an integrated LCD controller for low power metering and medical applications. Several devices offer application-based peripherals to provide single-chip solutions for flow and electricity metering.
  - # MSP430x3xx
    - MSP430x3xx devices are an older family of ROM/OTP devices offering 2.5V-5.5V operation, up to 32kB, 4MIPS and FLL.

# Software Development Tools

---

Multiple “flavors” based on price and application:

- Flash Emulation Tools
  - The Flash Emulation Tool (FET) supports complete in-system development and is available for all Flash devices. Programming, assembler/C-source level debug, single stepping, multiple hardware breakpoints, full-speed operation and peripheral access are all fully supported in-system using JTAG.
- eZ430-F2013 Development Tool
  - eZ430-F2013 Development Tool Designing with the world’s lowest powered MCU just got even easier with the new eZ430-F2013 complete development tool for only \$20!
- Integrated Development Environments
  - Texas Instruments and third party developers offer Integrated Development Environments (IDE) to program all MSP430 devices.

# TI Software Development Tools

---

Part Number	Contents Include	Devices	Price
IAR-KICKSTART	Free 4KB IDE - IAR Embedded Workbench Kickstart	All	Free
MSP-CCE430	Free 8KB IDE - Code Composer Essentials	All	Free
MSP-CCE430PRO	Full Version IDE - Code Composer Essentials Professional	All	\$499
IAR-UPDATE	Free Update for IAR Embedded Workbench (Kickstart, Baseline, Full)	All	Free
MSP430TOOLARCHIVE	Downloads for Previous Software Versions	All	Free

# Flash Emulation Tools

---

- Flash Emulation Tools
  - The Flash Emulation Tool (FET) supports complete in-system development and is available for all Flash devices. Programming, assembler/C-source level debug, single stepping, multiple hardware breakpoints, full-speed operation and peripheral access are all fully supported in-system using JTAG.
  - The FET kit comes with a programming interface, a target board and a kickstart version of an integrated development environment – everything you need to complete an entire project.
  - View Tools selection (INSERT LINK) pages to pick the tool right for your design. Flash Emulation tools vary by interface type (USB or parallel port) and by which device target board is included (14-pin to 100-pin options).

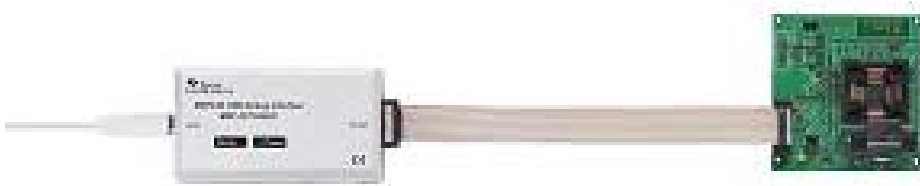
# eZ430-F2013 Development Tool

- eZ430-F2013 Development Tool
  - eZ430-F2013 Development Tool Designing with the world's lowest powered MCU just got even easier with the new eZ430-F2013 complete development tool for only \$20!
  - The platform provides all needed hardware and software in a portable USB stick enclosure. The eZ430-F2013 uses the included IAR IDE, providing full emulation with the option of designing a stand alone system or detaching the removable MSP430F2013 target board to integrate into an existing design.
  - The F20xx combines 16 MIPS performance, less than 1 microamp standby current, with your choice of performance analog converters. For more information visit [www.ti.com/ez430](http://www.ti.com/ez430).



# Hardware Starter Kits

Part Number	Port	Contents Include	Pin Count	Devices Supported	Price
eZ430-F2013	USB	USB stick Interface and target board		MSP430F20xx	\$20
eZ430-RF2500	USB	USB stick interface and two 2.4GHz wireless target boards		Multiple	\$49
MSP-FET430U14	USB	Interface and 14-pin target board w/ socket	14-pin	PW (TSSOP)	\$149
MSP-FET430U28	USB	Interface and 28-pin target board w/socket	20-/28-pin	DW (SOIC)	\$149
MSP-FET430U23x0	USB	Interface and MSP430F23x0 40-pin target board w/socket	40-pin	MSP430F23x0 RHA (QFN)	\$149
MSP-FET430U38	USB	Interface and 38-pin target board w/socket	38-pin	DA (TSSOP)	\$149
MSP-FET430U48	USB	Interface and 48-pin target board w/socket	48-pin	DL (SSOP)	\$149
MSP-FET430U64	USB	Interface and 64-pin target board w/socket	64-pin	PM (QFP)	\$149
MSP-FET430U80	USB	Interface and 80-pin target board w/socket	80-pin	PN (QFP)	\$149
MSP-FET430U100	USB	Interface and 100-pin target board w/socket	100-pin	PZ (QFP)	\$149



PC Interface, Module, Board



USB Stick

# MSP430x1xx Family Key Features

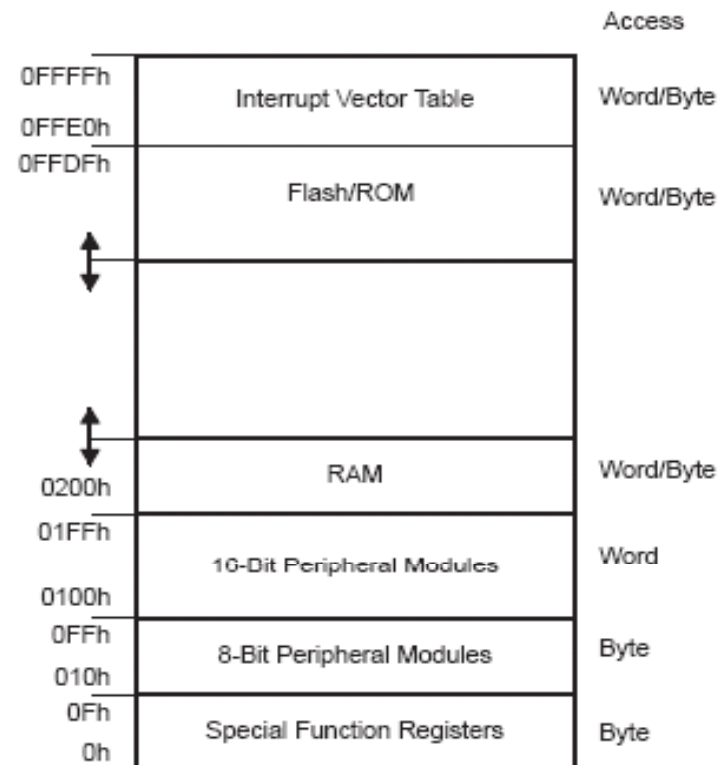
---

- Ultralow-power architecture extends battery life
  - 0.1- $\mu$ A RAM retention
  - 0.8- $\mu$ A real-time clock mode
  - 250- $\mu$ A / MIPS active
- High-performance analog ideal for precision measurement
  - 12-bit or 10-bit ADC — 200 ksps, temperature sensor, VRef
  - 12-bit dual-DAC
  - Comparator-gated timers for measuring resistive elements
  - Supply voltage supervisor
- 16-bit RISC CPU
  - Large register file eliminates working file bottleneck
  - Compact core design reduces power consumption and cost
  - Optimized for modern high-level programming
  - Only 27 core instructions and seven addressing modes
  - Extensive vectored-interrupt capability
- In-system programmable Flash permits flexible code changes, field upgrades and data logging

# Memory Mapped I/O

- Memory and I/O
  - Byte and Word access
  - Dedicated peripherals in 1 addresses
  - RAM
  - Flash Memory and ROM
  - Interrupt Vector Table

Figure 1-2. Memory Map

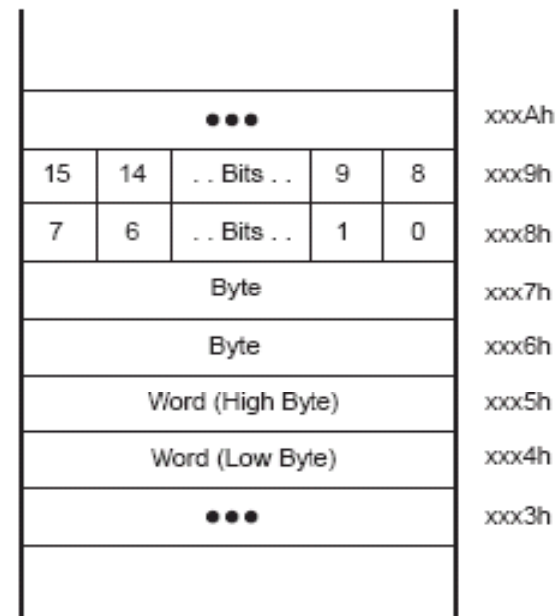


Similar to the HC12

# Memory Access: Bits, Bytes, and Words

Figure 1–3. Bits, Bytes, and Words in a Byte-Organized Memory

- Bytes are located at even or odd addresses.
- Words are only located at even addresses as shown in Figure 1–3. When using word instructions, only even addresses may be used. The low byte of a word is always an even address. The high byte is at the next odd address.



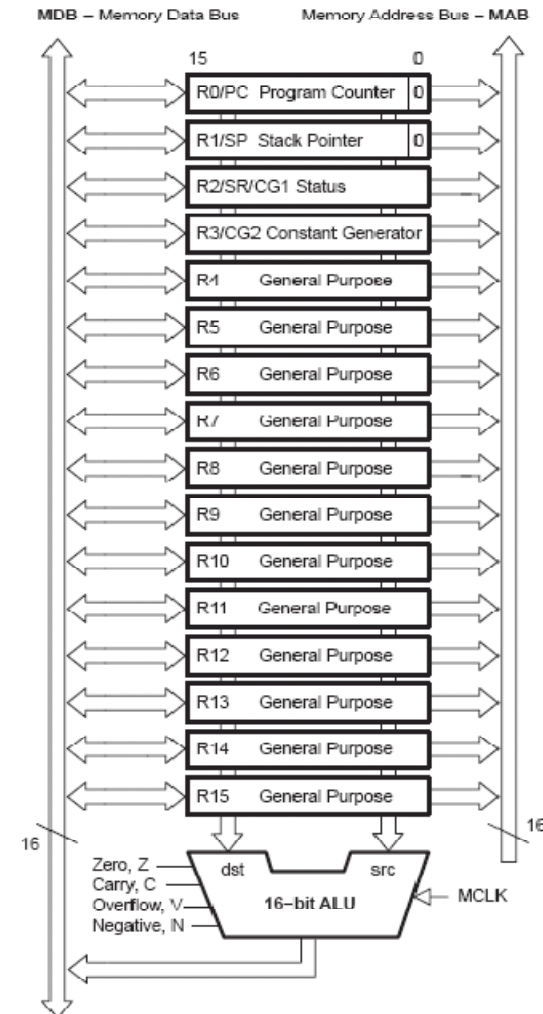
HC12 (1) does not force even boundaries for words and (2) high bytes is lowest address

# Registers and ALU

Figure 3-1. CPU Block Diagram

- Register sources to the ALU
  - General purpose
  - PC, SP, and Status Registers
- Register destination from the ALU
  - When two sources are used, one must also be the destination!
- The ALU provides normal condition codes for branching
  - Z, C, V, and N

Completely Different than HC12!



# Addressing Modes

Table 3–3. Source/Destination Operand Addressing Modes

- Seven addressing modes for the source operand and four addressing modes for the destination operand can address the complete address space with no exceptions. The bit numbers in Table 3–3 describe the contents of the As (source) and Ad (destination) mode bits.

As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.
10/–	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/–	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions and by 2 for .W instructions.
11/–	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

Different but easier  
than the HC12!

# Thinking RISC Addressing

---

- Simplistic View
  - Move data from memory to a register
  - Perform operations from register sources to a register destination
  - Move data from register to memory
- Indirect Addressing
  - Used for sources and not destinations
  - Result of opcode stored in a register
- Indexed, Symbolic and Absolute
  - Opcode followed by values, absolute addresses or offsets

See Section 3.3

# Instruction Set

---

- The complete MSP430 instruction set consists of 27 core instructions and 24 emulated instructions.
  - The core instructions are instructions that have unique op-codes decoded by the CPU.
  - The emulated instructions are instructions that make code easier to write and read, but do not have op-codes themselves, instead they are replaced automatically by the assembler with an equivalent core instruction. There is no code or performance penalty for using emulated instruction.
- There are three core-instruction formats:
  - Dual-operand
  - Single-operand
  - Jump

# RISC Instruction Opcodes

---

- RISC uses single word instructions
  - Limits addressing mode selection and jump range

Figure 3–9. Double Operand Instruction Format

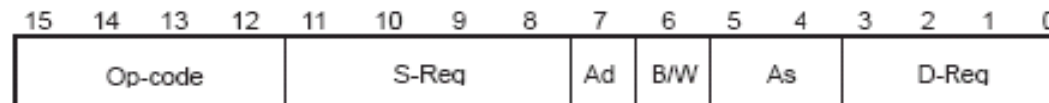


Figure 3–10. Single Operand Instruction Format

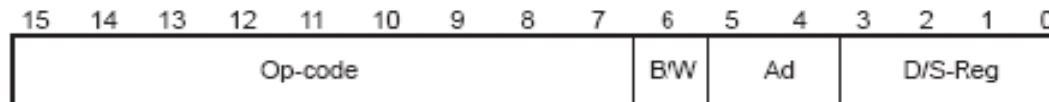
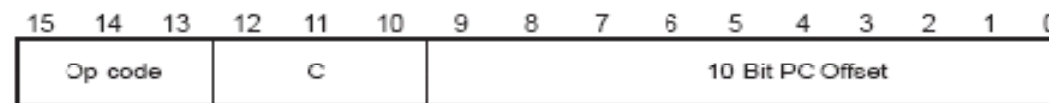


Figure 3–11. Jump Instruction Format



# Dual Operand Instructions

Table 3–11. Double Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			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	*	*	*
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	*	*	*	*
AND(.B)	src, dst	src .and. dst → dst	0	*	*	*

- \* The status bit is affected
- The status bit is not affected
- 0 The status bit is cleared
- 1 The status bit is set

**Note: Instructions CMP and SUB**

The instructions CMP and SUB are identical except for the storage of the result. The same is true for the BIT and AND instructions.

.B for byte wide processing

<http://www.ti.com/litv/pdf/slau049f>

slau049f: MSP430x1xx Family User's Guide

# Single Operand Instructions

Table 3–12. Single Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
RRC (.B)	dst	C → MSB →.....LSB → C	*	*	*	*
RRA (.B)	dst	MSB → MSB →....LSB → C	0	*	*	*
PUSH (.B)	src	SP - 2 → SP, src → @SP	-	-	-	-
SWPB	dst	Swap bytes	-	-	-	-
CALL	dst	SP - 2 → SP, PC+2 → @SP dst → PC	-	-	-	-
RETI		TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SP	*	*	*	*
SXT	dst	Bit 7 → Bit 8.....Bit 15	0	*	*	*

- \* The status bit is affected
- The status bit is not affected
- 0 The status bit is cleared
- 1 The status bit is set

All addressing modes are possible for the `CALL` instruction. If the symbolic mode (`ADDRESS`), the immediate mode (`#N`), the absolute mode (`&LDC`) or the indexed mode `x(RN)` is used, the word that follows contains the address information.

<http://www.ti.com/litv/pdf/slau049f>

slau049f: MSP430x1xx Family  
User's Guide

# Jump Instructions

---

Table 3–13. 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

Conditional jumps support program branching relative to the PC and do not affect the status bits. The possible jump range is from –511 to +512 words relative to the PC value at the jump instruction. The 10-bit program-counter offset is treated as a signed 10-bit value that is doubled and added to the program counter:

$$PC_{\text{new}} = PC_{\text{old}} + 2 + PC_{\text{offset}} \times 2$$

# Subroutine Calls

CALL	Subroutine		
<b>Syntax</b>	CALL	dst	
<b>Operation</b>	dst	-> tmp	dst is evaluated and stored
	SP - 2	-> SP	
	PC	-> @SP	PC updated to TOS
	tmp	-> PC	dst saved to PC

Examples for all addressing modes are given.

CALL	#EXEC	; Call on label EXEC or immediate address (e.g. #0A4h) ; SP-2 → SP, PC+2 → @SP, @PC+ → PC	CALL	@R5+	; Call on the address contained in the word ; pointed to by R5 and increment pointer in R5. ; The next time—S/W flow uses R5 pointer— ; it can alter the program execution due to ; access to next address in a table pointed to by R5 ; SP-2 → SP, PC+2 → @SP, @R5 → PC ; Indirect, indirect R5 with autoincrement
CALL	EXEC	; Call on the address contained in EXEC ; SP-2 → SP, PC+2 → @SP, X(PC) → PC ; Indirect address	CALL	X(R5)	; Call on the address contained in the address pointed ; to by R5 + X (e.g. table with address starting at X) ; X can be an address or a label ; SP-2 → SP, PC+2 → @SP, X(R5) → PC ; Indirect, indirect R5 + X
CALL	&EXEC	; Call on the address contained in absolute address ; EXEC ; SP-2 → SP, PC+2 → @SP, X(0) → PC ; Indirect address			
CALL	R5	; Call on the address contained in R5 ; SP-2 → SP, PC+2 → @SP, R5 → PC ; Indirect R5			
CALL	@R5	; Call on the address contained in the word ; pointed to by R5 ; SP-2 → SP, PC+2 → @SP, @R5 → PC ; Indirect, indirect R5			

# Full Instruction Set (1 of 2)

Table 3–17. MSP430 Instruction Set

Mnemonic		Description		V	N	Z	C
ADC (.B)†	dst	Add C to destination	dst + C → dst	x	x	x	x
ADD (.B)	src, dst	Add source to destination	src + dst → dst	x	x	x	x
ADDC (.B)	src, dst	Add source and C to destination	src + dst + C → dst	x	x	x	x
AND (.B)	src, dst	AND source and destination	src .and. dst → dst	0	x	x	x
BIC (.B)	src, dst	Clear bits in destination	.not.src .and. dst → dst	-	-	-	-
BIS (.B)	src, dst	Set bits in destination	src .or. dst → dst	-	-	-	-
BIT (.B)	src, dst	Test bits in destination	src .and. dst	0	x	x	x
BR†	dst	Branch to destination	dst → PC	-	-	-	-
CALL	dst	Call destination	PC+2 → stack, dst → PC	-	-	-	-
CLR (.B)†	dst	Clear destination	0 → dst	-	-	-	-
CLRC†		Clear C	0 → C	-	-	-	0
CLRNT†		Clear N	0 → N	-	0	-	-
CLRZ†		Clear Z	0 → Z	-	-	0	-
CMP (.B)	src, dst	Compare source and destination	dst - src	x	x	x	x
DADC (.B)†	dst	Add C decimally to destination	dst + C → dst (decimally)	x	x	x	x
DADD (.B)	src, dst	Add source and C decimally to dst.	src + dst + C → dst (decimally)	x	x	x	x
DEC (.B)†	dst	Decrement destination	dst - 1 → dst	x	x	x	x
DECD (.B)†	dst	Double-decrement destination	dst - 2 → dst	x	x	x	x
DINT†		Disable interrupts	0 → GIE	-	-	-	-
EINT†		Enable interrupts	1 → GIE	-	-	-	-
INC (.B)†	dst	Increment destination	dst + 1 → dst	x	x	x	x
INCD (.B)†	dst	Double-increment destination	dst+2 → dst	x	x	x	x
INV (.B)†	dst	Invert destination	.not.dst → dst	x	x	x	x
JC/JHS	label	Jump if C set/Jump if higher or same		-	-	-	-
JEQ/JZ	label	Jump if equal/Jump if Z set		-	-	-	-

# Full Instruction Set (2 of 2)

JGE	label	Jump if greater or equal		-	-	-	-
JL	label	Jump if less		-	-	-	-
JMP	label	Jump	PC + 2 x offset → PC	-	-	-	-
JN	label	Jump if N set		-	-	-	-
JNC/JLO	label	Jump if C not set/Jump if lower		-	-	-	-
JNE/JNZ	label	Jump if not equal/Jump if Z not set		-	-	-	-
MOV(.B)	src, dst	Move source to destination	src → dst	-	-	-	-
NOPT		No operation		-	-	-	-
POP(.B)†	dst	Pop item from stack to destination	@SP → dst, SP+2 → SP	-	-	-	-
PUSH(.B)	src	Push source onto stack	SP - 2 → SP, src → @SP	-	-	-	-
RET†		Return from subroutine	@SP → PC, SP + 2 → SP	-	-	-	-
RETI		Return from interrupt		x	x	x	x
RLA(.B)†	dst	Rotate left arithmetically		x	x	x	x
RLC(.B)†	dst	Rotate left through C		x	x	x	x
RRA(.B)	dst	Rotate right arithmetically		0	x	x	x
RRC(.B)	dst	Rotate right through C		x	x	x	x
SBC(.B)†	dst	Subtract not(C) from destination	dst + 0FFFFh + C → dst	x	x	x	x
SETC†		Set C	1 → C	-	-	-	1
SETN†		Set N	1 → N	-	1	-	-
SETZ†		Set Z	1 → C	-	-	1	-
SUB(.B)	src, dst	Subtract source from destination	dst + .not.src + 1 → dst	x	x	x	x
SUBC(.B)	src, dst	Subtract source and not(C) from dst.	dst + .not.src + C → dst	x	x	x	x
SWPB	dst	Swap bytes		-	-	-	-
SXT	dst	Extend sign		0	x	x	x
TST(.B)†	dst	Test destination	dst + 0FFFFh + 1	0	x	x	1
XOR(.B)	src, dst	Exclusive OR source and destination	src .xor. dst → dst	x	x	x	x

† Emulated Instruction

# Interrupts

---

- There are three types of interrupts:
  - System reset
  - (Non)-maskable NMI
  - Maskable
- The interrupt priorities are fixed and defined by the arrangement of the modules in the connection chain as shown in Figure 2–4. The nearer a module is to the CPU/NMIRS, the higher the priority. Interrupt priorities determine what interrupt is taken when more than one interrupt is pending simultaneously.

# Non-Maskable Interrupts

---

- (Non)-maskable NMI interrupts are not masked by the general interrupt enable bit (GIE), but are enabled by individual interrupt enable bits (NMIIE, ACCVIE, OFIE).
  - When a NMI interrupt is accepted, all NMI interrupt enable bits are automatically reset.
  - Program execution begins at the address stored in the (non)-maskable interrupt vector, 0FFFCh.
  - User software must set the required NMI interrupt enable bits for the interrupt to be re-enabled.
- A (non)-maskable NMI interrupt can be generated by three sources:
  - An edge on the RST/NMI pin when configured in NMI mode
  - An oscillator fault occurs
  - An access violation to the flash memory

# Maskable Interrupt

---

- Maskable interrupts are caused by peripherals with interrupt capability including the watchdog timer overflow in interval-timer mode.
  - Each maskable interrupt source can be disabled individually by an interrupt enable bit, or
  - all maskable interrupts can be disabled by the general interrupt enable (GIE) bit in the status register (SR).

# Interrupt Processing

---

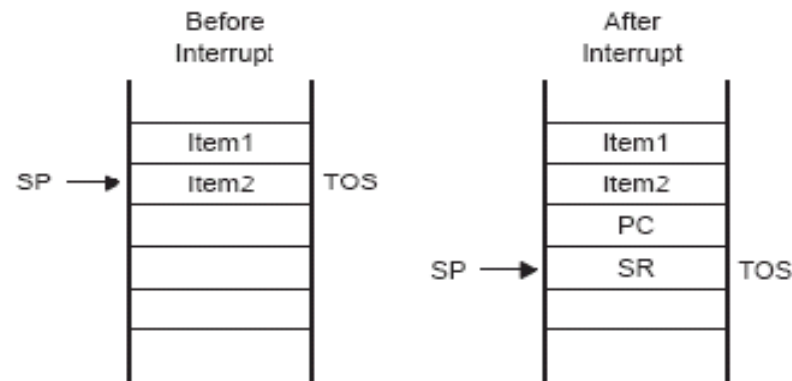
- The interrupt latency is 6 cycles, starting with the acceptance of an interrupt request, and lasting until the start of execution of the first instruction of the interrupt-service routine, as shown in Figure 2–7. The interrupt logic executes the following:
  - 1) Any currently executing instruction is completed.
  - 2) The PC, which points to the next instruction, is pushed onto the stack.
  - 3) The SR is pushed onto the stack.
  - 4) The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
  - 5) The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.
  - 6) The SR is cleared. This terminates any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.
  - 7) The content of the **interrupt vector** is loaded into the PC: the program continues with the **interrupt service routine** at that address.

# Interrupt Stack Operation

---

- Since there are so many registers, they are not automatically pushed onto the stack!
- Only the PC and the status register are pushed
  - If you want more, do it yourself

Figure 2-7. Interrupt Processing



# Interrupt Vectors

Table 2–1. Interrupt Sources, Flags, and Vectors

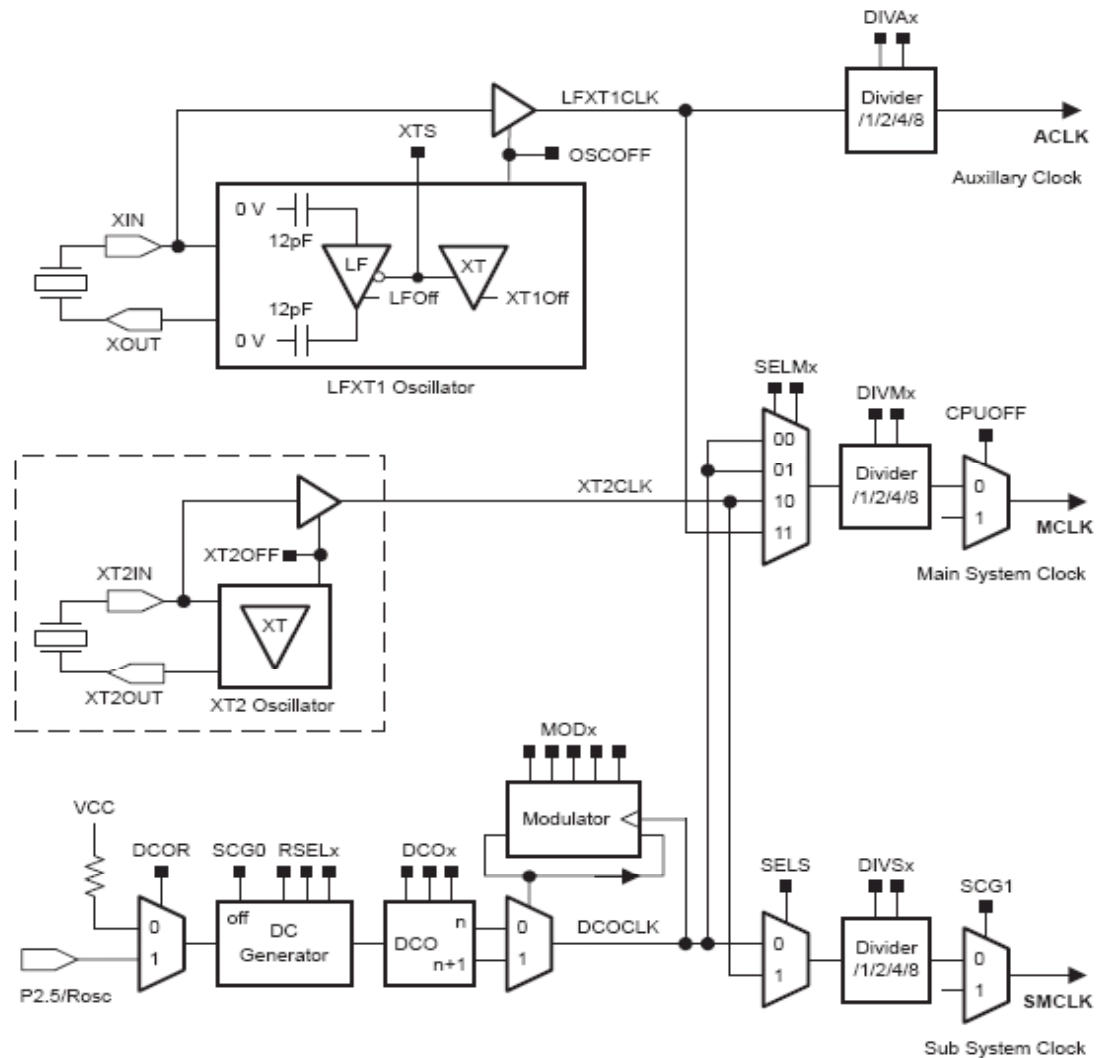
INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up, external reset, watchdog, flash password	WDTIFG KEYV	Reset	0FFFEh	15, highest
NMI, oscillator fault, flash memory access violation	NMIIFG OFIFG ACCVIFG	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	14
device-specific			0FFFAh	13
device-specific			0FFF8h	12
device-specific			0FFF6h	11
Watchdog timer	WDTIFG	maskable	0FFF4h	10
device-specific			0FFF2h	9
device-specific			0FFF0h	8
device-specific			0FFEh	7
device-specific			0FFECh	6
device-specific			0FFEAh	5
device-specific			0FFE8h	4
device-specific			0FFE6h	3
device-specific			0FFE4h	2
device-specific			0FFE2h	1
device-specific			0FFE0h	0, lowest

<http://www.ti.com/litv/pdf/slau049f>

slau049f: MSP430x1xx Family  
User's Guide

# Clock Module

Figure 4-1. Basic Clock Block Diagram



<http://www.ti.com/litv/pdf/slau049f>

slau049f: MSP430x1xx Family  
User's Guide

# Clock Module Capability

---

- The basic clock module includes two or three clock sources:
  - LFXT1CLK: Low-frequency/high-frequency oscillator that can be used either with low-frequency 32768-Hz watch crystals, or standard crystals or resonators in the 450-kHz to 8-MHz range.
  - XT2CLK: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 450-kHz to 8-MHz range.
  - DCOCLK: Internal digitally controlled oscillator (DCO) with RC-type characteristics.
- Three clock signals are available from the basic clock module:
  - ACLK: Auxiliary clock. The ACLK is the buffered LFXT1CLK clock source divided by 1, 2, 4, or 8. ACLK is software selectable for individual peripheral modules.
  - MCLK: Master clock. MCLK is software selectable as LFXT1CLK, XT2CLK (if available), or DCOCLK. MCLK is divided by 1, 2, 4, or 8. MCLK is used by the CPU and system.
  - SMCLK: Sub-main clock. SMCLK is software selectable as LFXT1CLK, XT2CLK (if available on-chip), or DCOCLK. SMCLK is divided by 1, 2, 4, or 8. SMCLK is software selectable for individual peripheral modules.