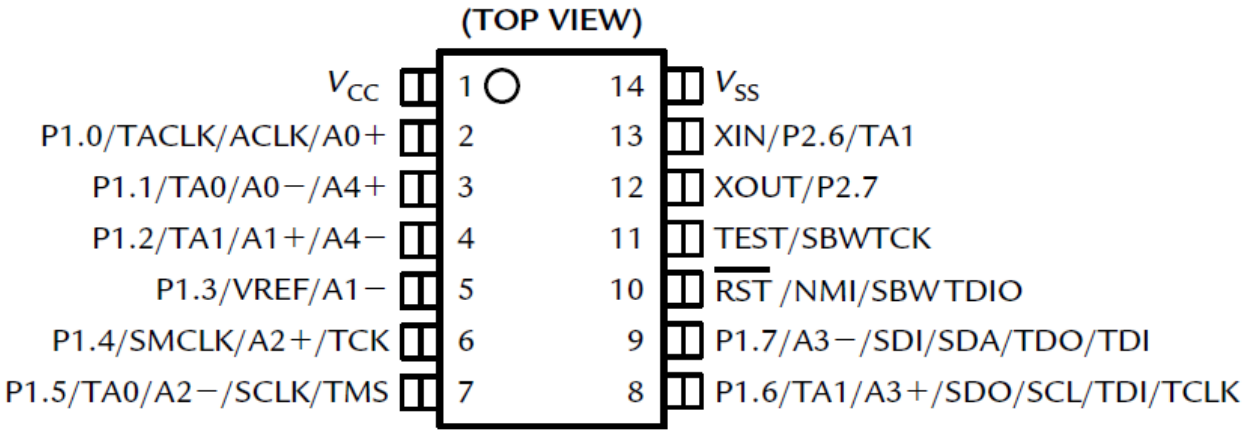
UNIT – 3

Low Power RISC MSP430:

Features:

- It is introduced in the late 1990s by Texas Instruments
- It is a 16-bit RISC Based Microcontroller with Von-Neumann Architecture
- It is Low cost and Low power consumption
- It is suitable for low-power and portable applications
- Its CPU is small and efficient, with a large number of registers
- It has set of intelligent peripherals like I/O Ports, Timers, ADC, DAC, Flexible Clock and USCI-I2C, SPI, UART
- It has 16-bit Data bus and 16-bit Address bus
- It can address 64 KB memory with Flash ROM and RAM
- It has 16 Registers in its CPU and each register is 16-bits wide can be used for either data or address
- It has only 27 Instructions
- It has 7 addressing modes
- Its CPU can run at16 MHz
- It has several low-power modes of operation
- It operates at Low voltage i.e. from 1.8V to 3.6V
- It is extremely easy to put the device into a low-power mode. No special instruction is needed**:** The mode is controlled by bits in the status register. TheMSP430 is awakened by an interrupt and returns automatically to its low-power mode after handling the interrupt.
- It can wake from a standby mode rapidly, perform its tasks, and return to a low-power mode.
- A wide range of peripherals is available, many of which can run autonomously without the CPU for most of the time.
- Many portable devices include liquid crystal displays, which the MSP430 can drive directly.
- It has Prioritized nested interrupts
- Ultralow-power optimization extends battery life
  - – 0.1 µA for RAM Data Retention
  - – 0.8 µA for RTC mode
  - – 250 µA/MIPS for active mode
- Zero-power Brown-Out -Reset (BOR)
- The order of storing the bytes in memory is little endian

Pin Diagram of MSP430F2003 and F2013:

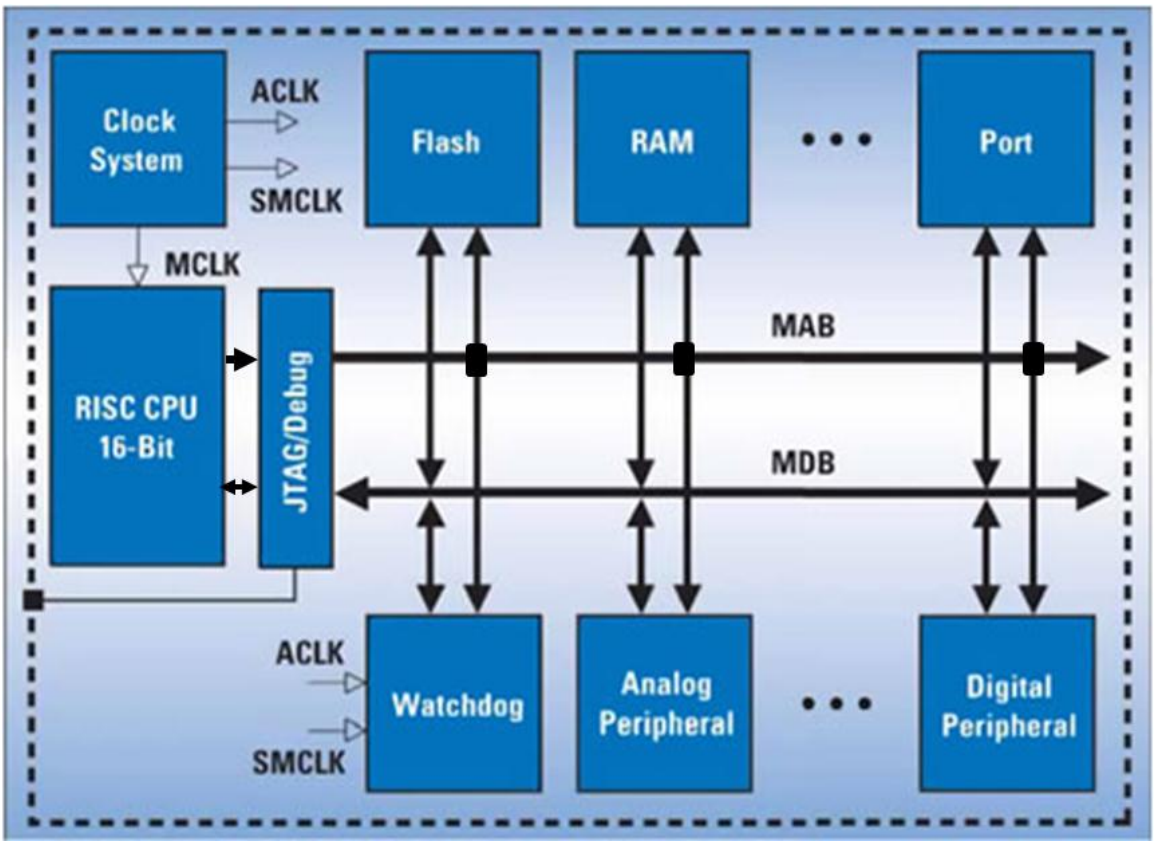


Pin-out of the MSP430F2003 and F2013

- V_{CC} and V_{SS} are the power supply voltage and ground pins for the whole device
- P1.0–P1.7, P2.6, and P2.7 are for digital input and output, grouped into ports P1and P2.
- TACLK, TA0, and TA1 are associated with Timer_A; TACLK can be used as the clock input to the timer, while TA0 and TA1 can be either inputs or outputs. These can be used on several pins because of the importance of the timer.
- A0−, A0+, and so on, up to A4±, are inputs to the analog-to-digital converter. It has four differential channels, each of which has negative and positive inputs. VREF is the reference voltage for the converter.
- ACLK and SMCLK are outputs for the microcontroller's clock signals. These can be used to supply a clock to external components or for diagnostic purposes.
- SCLK, SDO, and SCL are used for the universal serial interface, which communicates with external devices using the serial peripheral interface (SPI) or inter-integrated circuit (I^2C) bus.
- XIN and XOUT are the connections for a crystal, which can be used to provide an accurate, stable clock frequency.
- $\overline{RST}$ is an active low reset signal. Active low means that it remains high near V_{CC} for normal operation and is brought low near V_{SS} to reset the chip.
- NMI is the non-maskable interrupt input, which allows an external signal to interrupt the normal operation of the program.
- TCK, TMS, TCLK, TDI, TDO, and TEST form the full JTAG interface, used to program and debug the device.

- SBWTDIO and SBWTCK provide the Spy-Bi-Wire interface, an alternative to theusual JTAG connection that saves pins.

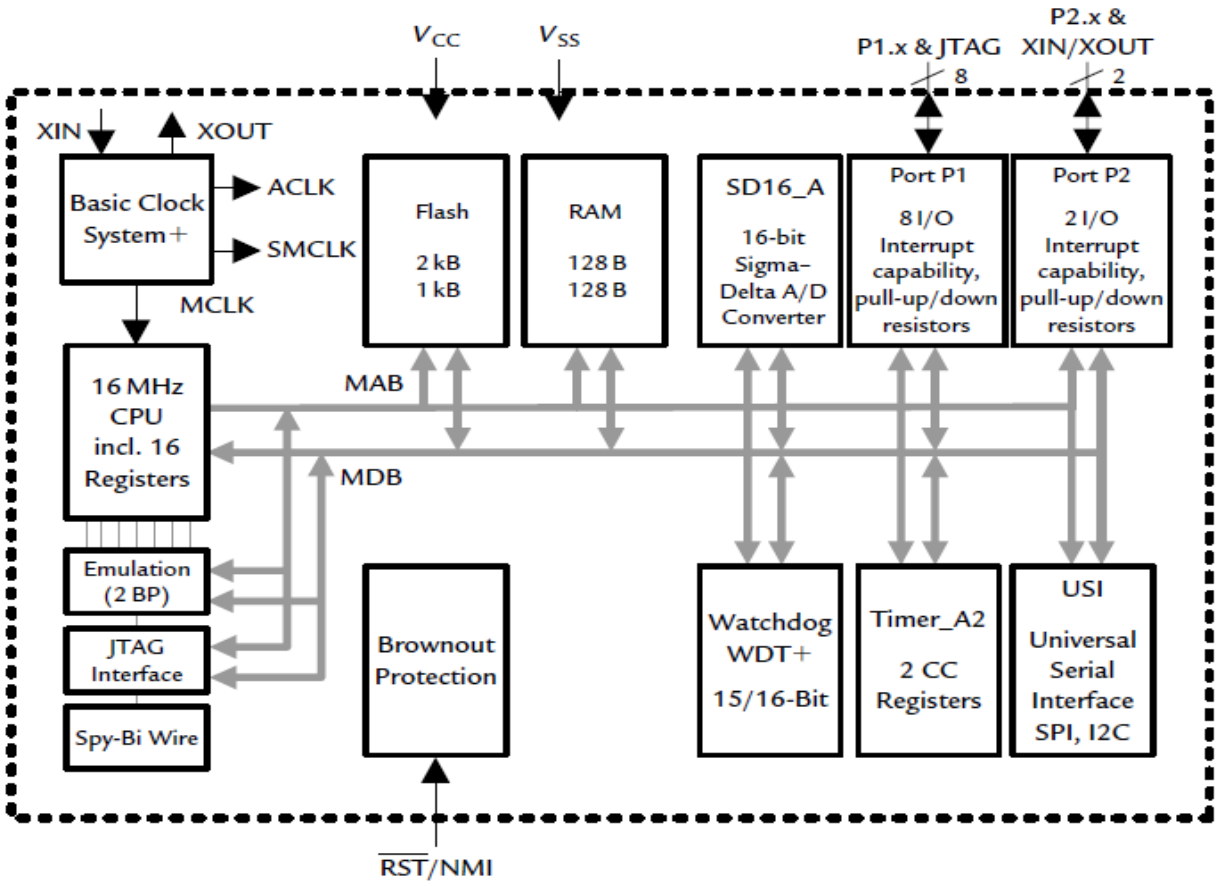
General Block Diagram of MSP430:



Architecture of MSP430F2003 and F2013:

The Architecture of MSP430F2003 and F2013 in shown below:

- On the left are the CPU and its supporting hardware, including the clock generator. The emulation, JTAG interface and Spy-Bi-Wire are used to communicate with a desktop computer when downloading a program and for debugging.
- The main blocks are linked by the memory address bus (MAB) and memory data bus (MDB).
- These devices have flash memory, 1KB in the F2003 or 2KB in the F2013, and128 bytes of RAM.
- The brownout protection comes into action if the supply voltage drops to a dangerous level.
- There are ground and power supply connections. Ground is labeled V_{SS} and is taken to define 0V. The supply connection is V_{CC}.A range of 1.8–3.6V is specified for the F2013.
- Six blocks are shown for peripheral functions (there are many more in larger devices). All MSP430s include input/output ports, Timer_A, and a watchdog



Block diagram of the MSP430F2003 and F2013

timer. The universal serial interface (USI) and sigma–delta analog-to-digital converter (SD16_A) are particular features of this device.

Memory Organization:

- MSP430 consists of 64K memory which includes Flash/ROM and RAM
- The memory data bus is 16 bits wide and can transfer either a word of 16 bits or a byte of 8 bits.
- Memory Addresses are 16-bit
- Bytes may be accessed at any address but words need more care.
- Even Address access for word
- The address of a word is defined to be the address of the byte with the lower address, which must be even. Thus the two bytes at 0x0200 and 0x0201 can be considered as a valid word with address 0x0200, which may be fetched in a single cycle of the bus.
- On the other hand, it is not possible to treat the two bytes at 0x0201 and 0x0202 as a single word because their address would be 0x0201, which is odd and therefore invalid. These two bytes straddle the boundary of two words.

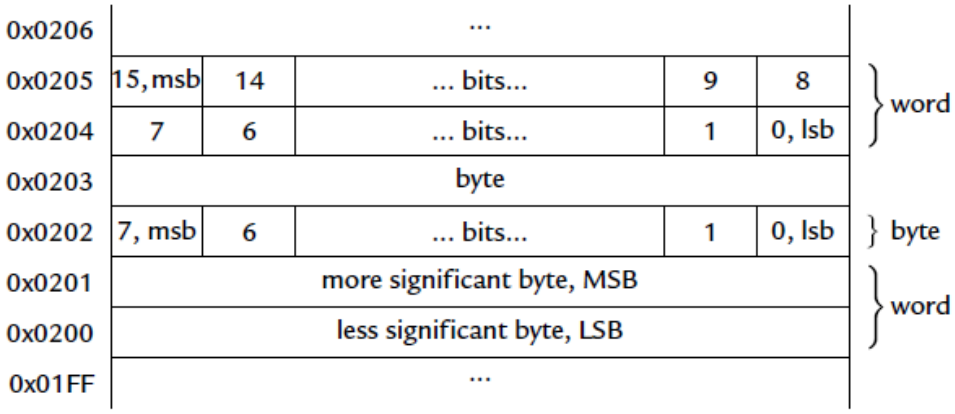


Figure: Ordering of bits, bytes, and words in Memory

Little-endian ordering may appear more logical but has one awkward outcome. A debugger usually displays the contents of memory by showing the value of each byte by default. Addresses increase from left to right across each line. This means that the low-order byte is displayed first, followed by the high-order byte. Thus our value of **0x1234** is displayed as **34 12**. It is easy to be puzzled by this. A simple solution is to display the contents of memory in words instead.

The following figure shows the **Memory Map of the F2013:**

Address	Type of memory
0xFFFF 0xFFC0	interrupt and reset vector table
0xFFBF 0xF800	flash code memory (lower boundary varies)
0xF7FF 0x1100	
0x10FF 0x1000	flash information memory
0x0FFF 0x0C00	*bootstrap loader* (*not in F20xx*)
0x0BFF 0x0280	
0x027F 0x0200	RAM (upper boundary varies)
0x01FF 0x0100	peripheral registers with word access
0x00FF 0x0010	peripheral registers with byte access
0x000F 0x0000	special function registers (byte access)

Figure: Memory map of the MSP430F2013

Here is a brief description of each region:

- **Special function registers:** Mostly concerned with enabling functions of some modules and enabling and signaling interrupts from peripherals.
- **Peripheral registers with byte access and Peripheral registers with word access:** Provide the main communication between the CPU and peripherals. Some must be accessed as words and others as bytes. They are grouped in this way to avoid wasting addresses
- **Random access memory:** Used for variables. This always starts at address 0x0200 and the upper limit depends on the size of the RAM. The F2013 has 128 B.
- **Bootstrap loader:** Contains a program to communicate using a standard serial protocol, often with the COM port of a PC. This can be used to program the chip. All MSP430s had a bootstrap loader until the F20xx.
- **Information memory:** A 256B block of flash memory that is intended for storage of nonvolatile data. This might include serial numbers to identify equipment—an address for a network, for instance—or variables that should be retained even when power is removed.
- **Flash Code memory:** Holds the program, including the executable code itself and any constant data. The F2013 has 2KB but the F2003 only 1KB.
- **Interrupt and reset vectors:** Used to handle "exceptions," when normal operation of the processor is interrupted or when the device is reset. This table was smaller and started at 0xFFE0

Central Processing Unit:

The central processing unit (CPU) executes the instructions stored in memory. It steps through the instructions in the sequence in which they are stored in memory until it encounters a branch or when an exception occurs (interrupt or reset). It includes the arithmetic logic unit (ALU), which performs computation, a set of 16 registers designated R0–R15 and the logic needed to decode the instructions and implement them. The CPU can run at a maximum clock frequency f_{MCLK} of 16MHz in the MSP430F2xx family.

Registers of MSP430 CPU:

The CPU of MSP 430 includes a 16-bit ALU and a set of 16 Registers R0 – R15. In these registers four are special Purpose and 12 are general purpose registers. All the registers can be addressed in the same way.

The special Purpose Registers are:

PC (Program Counter), SP (Stack Pointer), SR (Status Register), CGx (Constant Generator)

The MSP430 CPU includes an arithmetic logic unit (ALU) that handles addition, subtraction, comparison and logical (AND, XOR) operations. ALU operations can affect the overflow, zero, negative, and carry flags in the status register.
The following figure shows the register organization of MSP430 CPU.

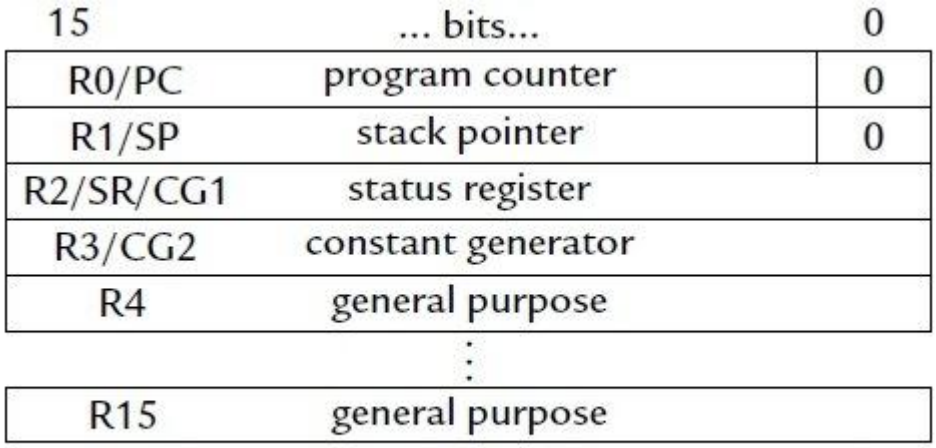


Figure: Registers in the CPU of the MSP430

R0: Program Counter (PC):

This contains the address of the next instruction to be executed. The Program counter is incremented by 2. It is important to note that the PC is aligned at even addresses, because the instructions are 1-3 words.

Subroutines and interrupts also modify the PC but in these cases the previous value (Next line of current instruction which is executing) is saved on the stack and restored later.

R1: Stack Pointer (SP):

- The Stack Pointer (SP/R1) is located in R1.
- The Stack Pointer holds the address of the top of the stack
- Stack can be used by user to store data for later use(instructions: store by PUSH, retrieve by POP)
- The stack pointer is used by the CPU to store the return addresses of subroutine calls and interrupts. It uses a pre-decrement, post-increment scheme.
- The stack is allocated at top of RAM and grows down towards the low address. SP holds the address of top of the stack.
- The stack pointer holds the address of the most recently added word
- Stack can be used by subroutine calls to store the program counter value for return at subroutine's end (RET)

- Used by interrupt - system stores the actual PC value first, then the actual status register content (on top of stack) on return from interrupt (RETI) the system get the same status as just before the interrupt happened (as long as none has changed the value on TOS) and the same program counter value from stack.

The operation of the stack is illustrated in below Figure.

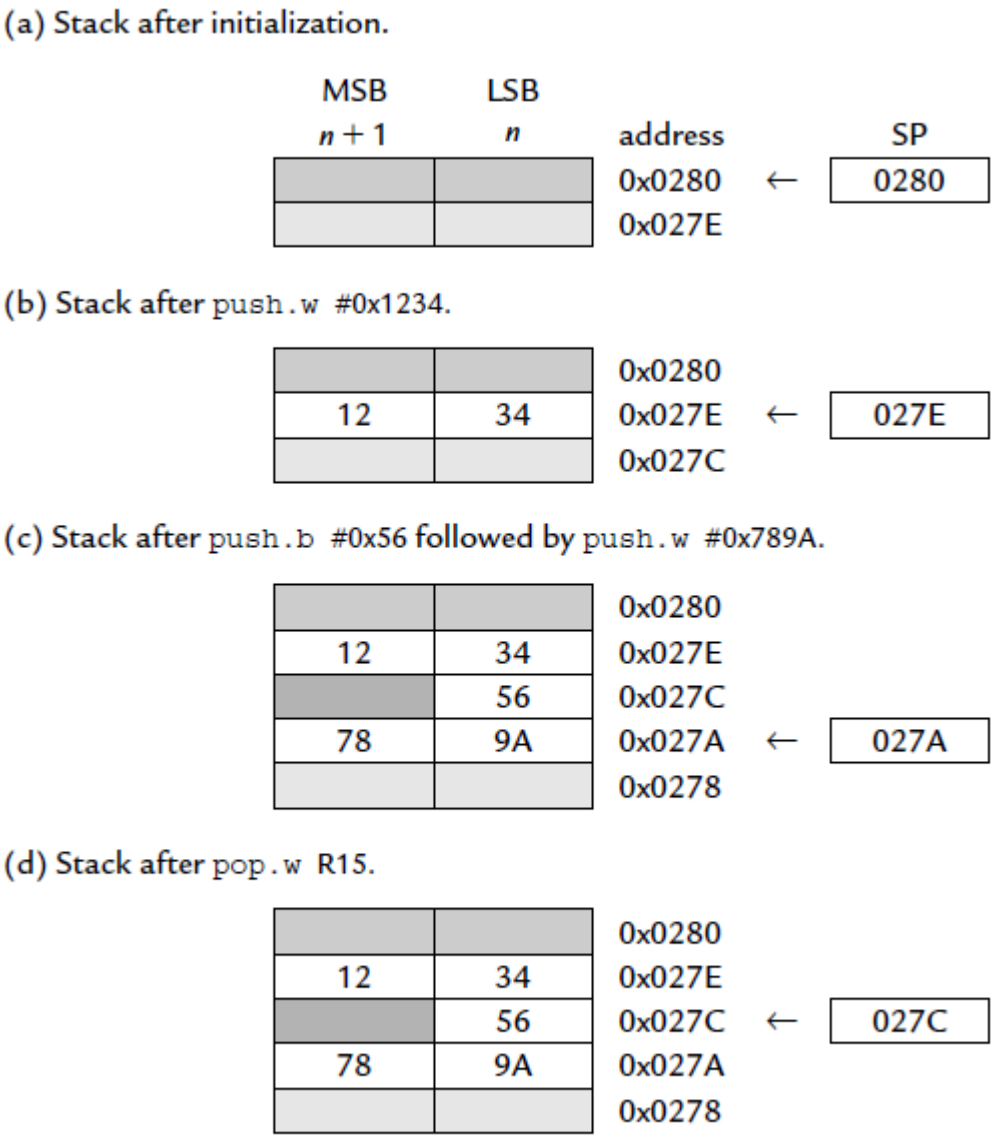


Figure: Operation of the stack in the MSP430F2013, whose RAM lies from 0x0200 to 0x027F

Note: For programs written in C, the compiler initializes the stack automatically as part of the startup code, which runs silently before the program starts, but you must initialize SP yourself in assembly language.

R2: Status Register (SR):

The Status Register (SR/R2) is a 16 bit register, and it stores the state and control bits. The system flags are changed automatically by the CPU depending on the result of an operation in a register. The reserved bits are not used in the

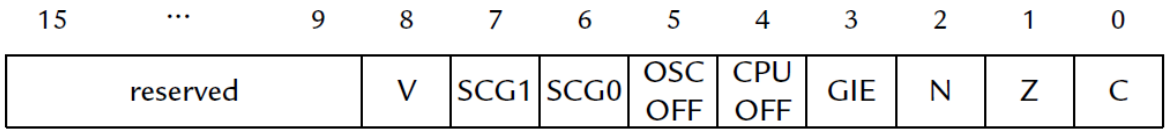


Figure: Individual bits in the status register

MSP430.

- The **Carry flag C** is set when the result of an arithmetic operation is too large to fit in the space allocated. In other words, an overflow occurred.
- The **Zero flag Z** is set when the result of an operation is 0.
- The **Negative flag N** is made equal to the msb of the result, which indicates a negative number if the values are signed.
- The **Signed Over Flow flag V** is set when the result of a signed operation has overflowed, even though a carry may not be generated

❖ Remember that a byte can hold the values 0 to 0xFF if it is unsigned or −0x80 to 0x7F if it is signed.

- **Enable Interrupts:** Setting the **General Interrupt Enable–GIE** bit enables maskable interrupts, provided that the individual sources of interrupts have themselves been enabled. Clearing the bit disables all maskable interrupts. There are also non-maskable interrupts, which cannot be disabled with GIE.
- **Control of Low-Power Modes:** The **CPUOFF, OSCOFF, SCG0 (**System Clock Generator**),** and **SCG1** bits control the mode of operation of the MCU. All systems are fully operational when all bits are clear. Setting combinations of these bits puts the device into one of its low-power modes

R2/R3: Constant Generator Registers (CG1/CG2):

This provides the six most frequently used values so that they need not be fetched from memory whenever they are needed. It uses both R2 and R3 to provide a range of useful values by using the CPU's addressing modes.

R4 - R15: General–Purpose Registers:

The remaining 12 registers R4–R15 have no dedicated purpose and may be used as general working registers. They may be used for either data or addresses because both are 16-bit values, which simplify the operation significantly.
The following figure shows the MSP430 CPU Block Diagram.

The CPU features include:

- RISC architecture with 27 instructions and 7 addressing modes.
- Orthogonal architecture with every instruction usable with every addressing mode.
- Full register access including program counter, status registers, and stack pointer.
- Single-cycle registers operations.
- Large 16-bit register file reduces fetches to memory.
- 16-bit address bus allows direct access and branching throughout entire memory range.
- 16-bit data bus allows direct manipulation of word-wide arguments.
- Constant generator provides six most used immediate values and reduces code size.
- Direct memory-to-memory transfers without intermediate register holding.
- Word and byte addressing and instruction formats.

➢ An **orthogonal** instruction set is an instruction set architecture where all instruction types can use all addressing modes. It is "orthogonal" in the sense that the instruction type and the addressing mode vary independently.

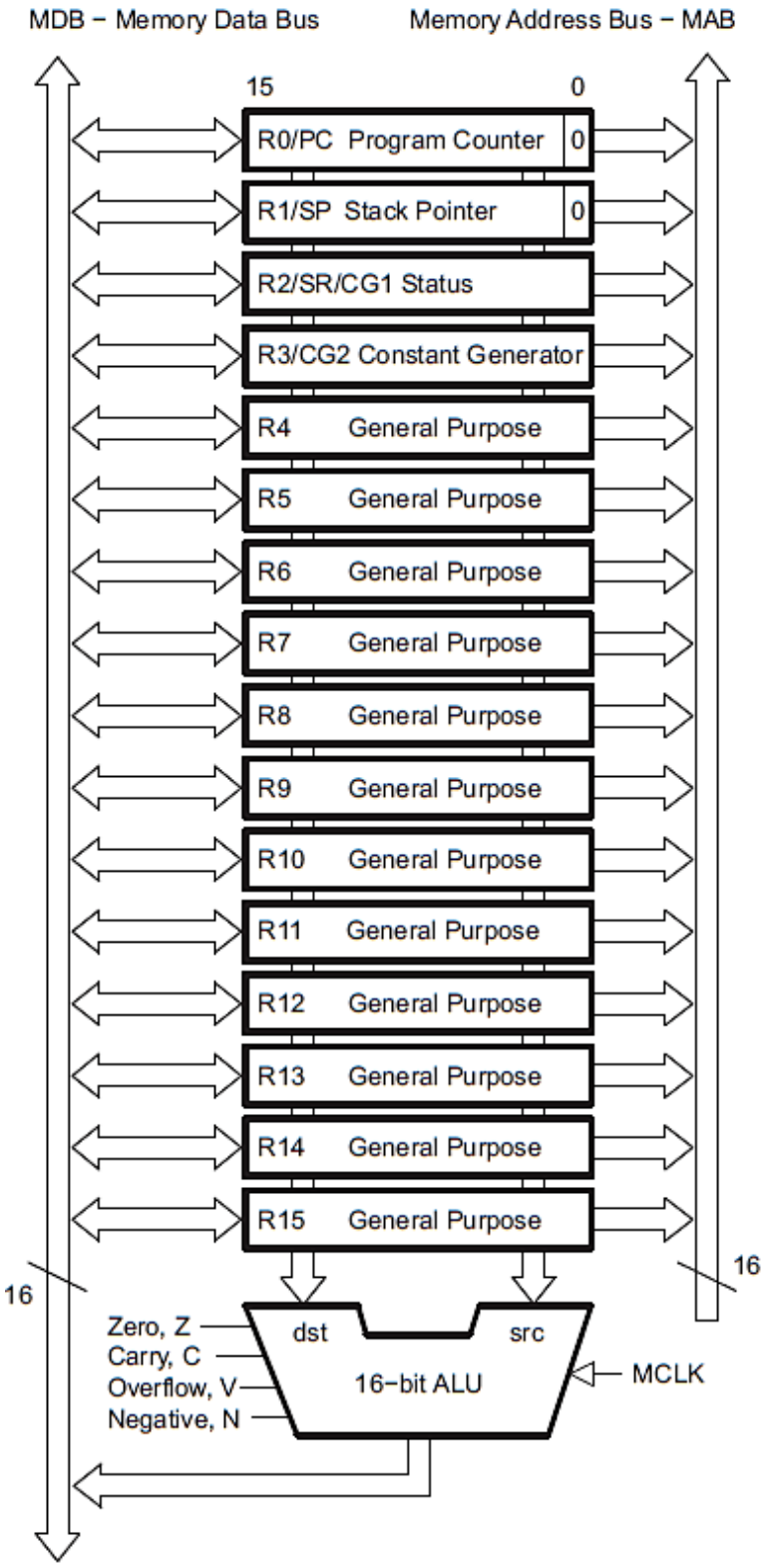


Figure: CPU Block Diagram

Addressing modes:

The MSP430 supports seven addressing modes. They are:

1) Register mode
2) Indexed mode
3) Symbolic mode
4) Absolute mode
5) Indirect register mode
6) Indirect auto increment mode
7) Immediate mode

1) Register Mode:

Register mode operations work directly on the processor registers, R4 through R15, or on special function registers, such as the program counter or status register. They are very efficient in terms of both instruction speed and code space.

Ex: MOV.b R4, R5 *; move (copy) byte from R5 to R6*

MOV.w R4, R5*; move (copy) word from R5 to R6*

Operation: Move (copy) the contents of source (register R4) to destination (register R5), Register R4 is not affected. **.b →** for byte operation& **.w →** for word operation

2) Indexed mode:

The Indexed mode commands are formatted as X(Rn), where X is a constant and Rn is one of the CPU registers. The absolute memory location X+Rn is addressed.

Indexed mode addressing is useful for applications such as lookup tables

Ex: MOV.b 4(R5), 6(R6) ; move data from address **4 + (R5)**

; to address **6 + (R6)**

Operation: Move the contents of the source address (contents of R5 + 4) to the destination address (contents of R6 + 6). Thesource and destination registers (R5 and R6) are not affected.

3) Symbolic mode(PC Relative):

Symbolic mode allows the assignment of labels to fixed memory locations, so that those locations can be addressed. This is useful for the development of embedded programs.

In this case the program counter PC is used as the base address, so the constant is the offset to the data from the PC. TI calls this the symbolic mode although it is usually described as PC-relative addressing. It is used by writing the symbol for a memory location without any prefix.

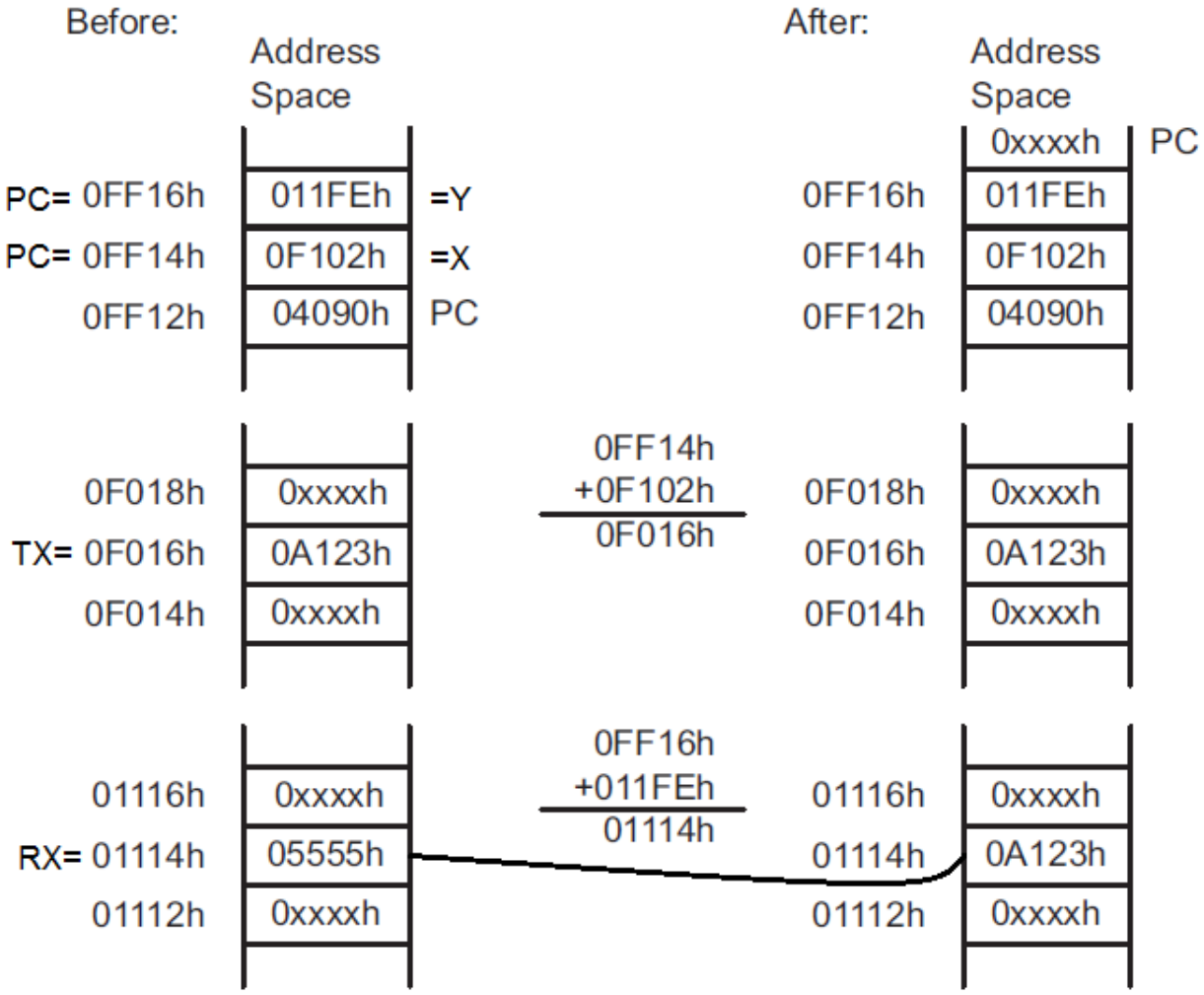
Ex: MOV.w TX, RX ; move data from src address **TX**
; to dst address **RX**

The assembler replaces the above instruction by the indexed form

MOV X(PC),Y(PC) ; where X=TX-PC ⇨ TX=PC+X **&**
; Y=RX-PC ⇨ RX=PC+Y

Operation: Move the contents of the source address TX (contents of PC + X) to the destination address RX (contents of PC + Y). The words after the instruction contains the differences between the PC and the source address i.e. X or destination address i.e. Y. The assembler computes and inserts offsets X and Y automatically.

Ex: MOV.wTX, RX ; Src. address TX = 0F016h
;Dst. address RX = 01114h



4) Absolute mode:

Similar to Symbolic mode, with the difference that the label is preceded by "**&**". This mode is used for special function and peripheral registers, whose addresses are fixed in the memory map.

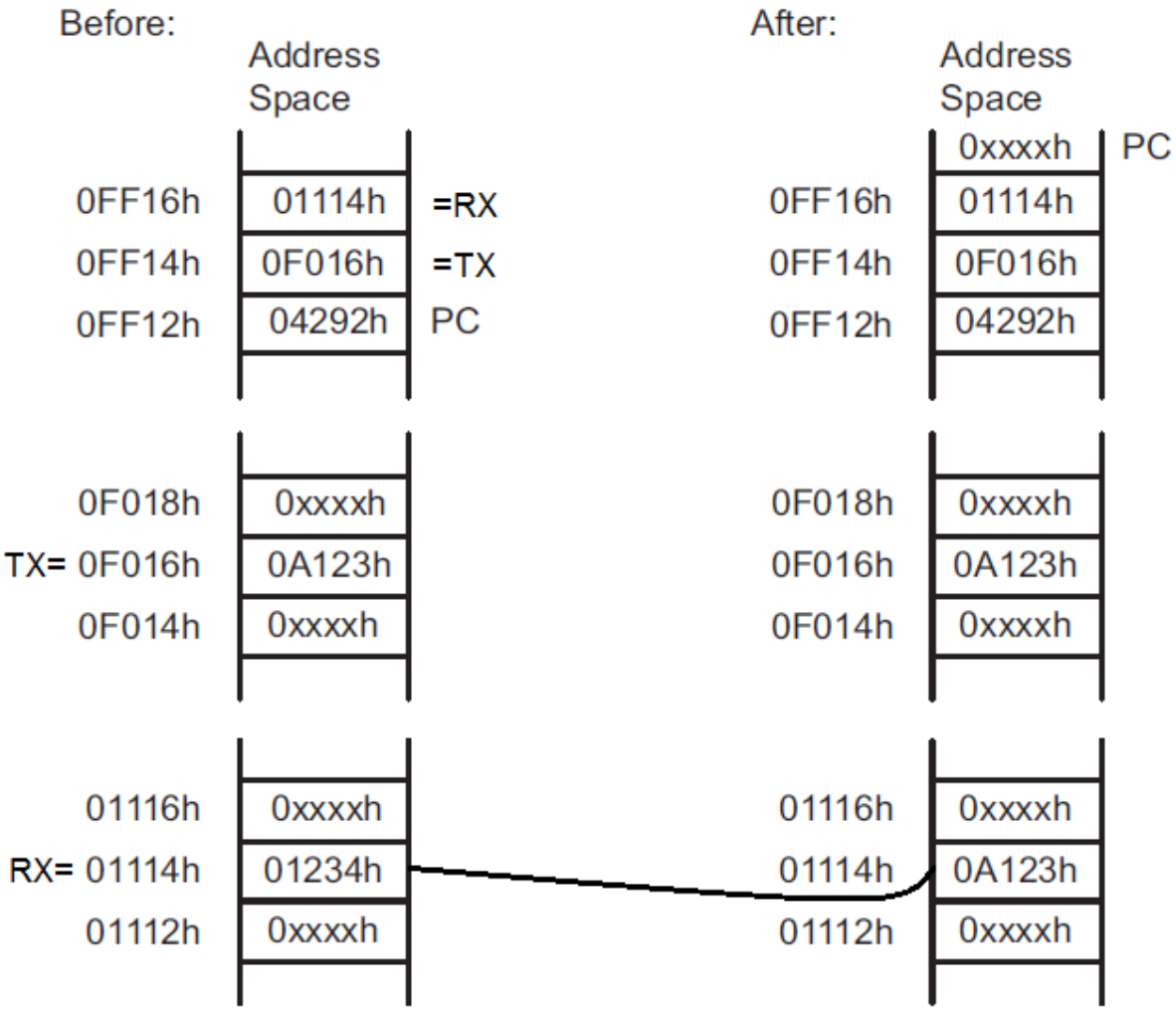
Ex: MOV.w &TX, &RX ; move data from src address **TX**
; to dst address **RX**

The assembler replaces the above instruction by the indexed form

MOV X(SR),Y(SR)⇨ MOV X(0),Y(0)
; Where X=TX-0⇨ TX=X → Absolute Address**&**
; Y=RX-0⇨ RX=Y→ Absolute Address

Operation: Move the contents of the source address TX to the destination address RX. The words after the instruction contain the absolute address of the source and destination addresses.

Ex: MOV.w&TX, &RX ; Src. address TX = 0F016h
;Dst. addressRX = 01114h



Indirect register mode:

This is available only for the source and is shown by the symbol @ in front of a register,such as @R5. It means that the contents of R5 are used as the address of the operand. In other words, R5 holds a pointer rather than a value.

Indirect addressing cannot be used for the destination so indexed addressing must be used instead Indirect addressing i.e. the substitute for destination operand is 0(Rd).

Ex: MOV.w @R10, 0(R11)

Operation: Move the contents of the source address (contents of R10) to the destination address (contents of R11). The registers are not modified.

6) Indirect Autoincrement Mode:

Again this is available only for the source and is shown by the symbol **@** in front of a register with a **+** sign after it, such as @Rn+. It uses the value in Rn as a

pointer and automatically increments it afterward by 1 for a byte operation or by 2 for a word operation after the fetch.

Ex: MOV.w @R10+, 0(R11)

Operation: Move the contents of the source address (contents of R10) to the destination address (contents of R11). After that R10 is incremented by 1 for a byte operation, or 2 for a word operation.

7) Immediate Mode:

Immediate mode is used to assign constant values to registers or memory locations.

Ex: MOV.b #45h, R5

Operation: Move the immediate constant 45h to the destination (register R5).

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.
- The instruction set is orthogonal withfew exceptions, meaning that all addressing modes can be used with all instructions andregisters.

Movement Instructions:

There is only the one **"mov"** instruction to move data. It can address all of memory as either source or destination, including both registers in the CPU and the whole memory map.

```
mov.wsrc, dst        ; move (copy)        dst = src
```

Stack Operations

These instructions either push data onto the stack or pop them off.

```
push.wsrc      ; push data onto stack--SP = src
pop.w dst              ; pop data off stack.   dst = SP++ emulated
```

The pop operation is emulated using post-increment addressing but push requires a special instruction because pre-decrement addressing is not available.

1) Arithmetic and Logic Instructions:

Arithmetic Instructions with Two Operands

add.w src ,dst	; add	dst += src
addc.w src ,dst	; add with carry	dst += (src + C)
adc.w dst	; add carry bit	dst += C **emulated**
sub.w src ,dst	; subtract	dst -= src
subc.wsrc ,dst	; subtract with borrow	dst -= (src + ~C)
sbc.w dst	; subtract borrow bit	dst -= ~C **emulated**
cmp.w src ,dst	; compare, set flags only	(dst - src)

Note: The compare operation **"cmp"** is the same as subtraction except that only the bits in SR are affected; the result is not written back to the destination.

Arithmetic Instructions with One Operand

All these are emulated, which means that the operand is always a destination:

clr.wdst	; clear	dst = 0 **emulated**
dec.wdst	; decrement	dst -- **emulated**
decd.wdst	; double decrement	dst -= 2 **emulated**
inc.wdst	; increment	dst++ **emulated**
incd.wdst	; double increment	dst += 2 **emulated**
tst.wdst	; test (compare with 0) (dst - 0)	**emulated**

Decimal Arithmetic

These instructions are used when operands are binary-coded decimal (BCD) rather than ordinary binary values.

dadd.w src ,dst	; decimal add with carry	dst += src + C
dadc.w dst	; decimal add carry bit	dst += C **emulated**

Logic Instructions with Two Operands

and.w src ,dst	; bitwise and	dst &= src
xor.w src ,dst	; bitwise xor	dst ^= src
bit.w src ,dst	; bitwise test, set flags only	(dst & src)
bis.w src ,dst	; bit set	dst |= src
bic.w src ,dst	; bit clear	dst &= ~src

Note: The **and** &**bitwise test** operations are identical except that bit is only a test and does not change its destination.

Logic Instructions with One Operand

There is only one of these, the invert **"inv"** instruction, also known as ones complement, which changes all bits of 0 to 1 and those of 1 to 0:

inv.w dst	; invert bits	dst = ~dst	**emulated**

Byte Manipulation

These instructions do not need a suffix because the size of the operands is fixed:

swpb src	; swap upper and lower bytes (word only)
sxt src	; extend sign of lower byte (word only)

- The swap bytes instruction "swpb" swaps the two bytes in a word.
- The sign extend instruction "sxt" is used to convert a signed byte into a signed word.

Operations on Bits in Status Register

There is a set of emulated instructions to set or clear the four lowest bits in the status register, those that can be masked using the constant generator:

clrc	; clear carry bit c = 0	**emulated**
clrn	; clear negative bit n = 0	**emulated**
clrz	; clear zero bit z = 0	**emulated**
setc	; set carry bit c = 1	**emulated**
setn	; set negative bit n = 1	**emulated**
setz	; set zero bit z = 1	**emulated**
dint	; disable general interrupts GIE=0	**emulated**
eint	; enable general interrupts GIE =1	**emulated**

2) Shift and Rotate Instructions:

There are three types of shifts
(i) logicalshift (ii) arithmetic shift (iii) rotation.

- **Logical shift** inserts zeroes for both right and left shifts.
- **Arithmetic shift** inserts zero for left shifts at **lsb**but for the right shifts the **msb** is replicated.
- **Rotation** does not introduce or lose any bits; bits that are moved out of one end ofthe register are passed around to the other.

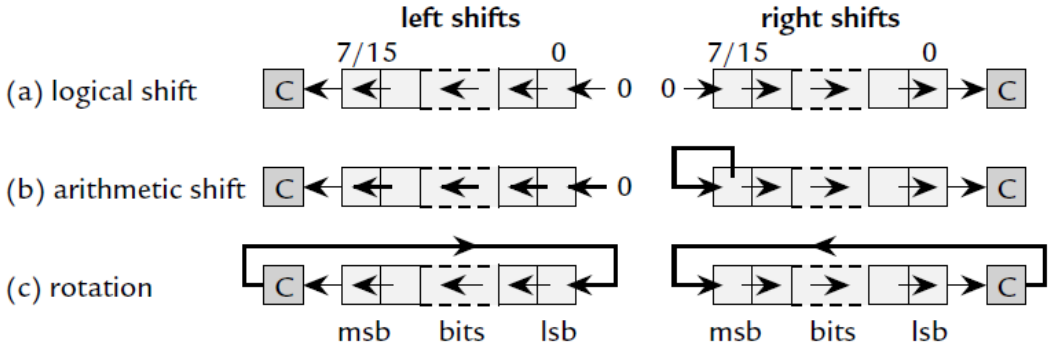


Figure: Left and right logical shifts, arithmetic shifts, and rotations on an 8- or 16-bit register.

- The MSP430 has arithmetic shifts and rotations, all of which use the carry bit.
  The right-shifts are native instructions but the left shifts are emulated

rla dst	; arithmetic shift left	**emulated**
rra src	; arithmetic shift right	
rlc dst	; rotate left through carry	**emulated**
rrc src	; rotate right through carry	

3) Flow of Control:

Subroutines, Interrupts, and Branches

brdst	; branch (go to)	PC = dst	**emulated**
call src	; call subroutine		
ret	; return from subroutine		**emulated**
reti	; return from interrupt		
nop	; no operation (consumes single cycle)		**emulated**

Jumps ⇨Unconditional and Conditional

- The unconditional jump instruction is

```
jmp label          ; unconditional jump
```

- **jmp** fits in a single word, including the offset, but its range is limited to about±1KB from the current location.
- **br** can go anywhere in the address space and use any addressing mode but isslower and requires an extra word of program storage.

- The conditional jumps are the “decision-making” instructions and test certain bits or combinations in the status register.

```
jc label    ; jump if carry set,      C = 1   same as jhs
jnc label   ; jump if carry not set,  C = 0   same as jlo
jn label    ; jump if negative,               N = 1
jz label    ; jump if zero,           Z = 1   same as jeq
jnz label   ; jump if nonzero,                Z = 0   same as jne

jeq label   ; jump if equal,                  dst = src       same as jz
jne label   ; jump if not equal,    dst!= src same as jnz
jhs label   ; jump if higher or same,         dst>= src same as jc
jlo label   ; jump if lower,                  dst < src same as jnc

jge label   ; jump if greater or equal,       dst >= src      signed values
jl(t) label ; jump if less than,              dst < src       signed values
```

Many branches have two names to reflect different usage. For example, it is clearer to use**jc** if the carry bit is used explicitly—after a rotation, for instance—but **jhs** is more appropriate after a comparison.

Assume that the “comparison” jumps follow **cmp.wsrc,dst**, which sets the flags according to the difference **dst-src**. Alternatively, **tst.wdst** sets the flags for **dst – 0.**

Both mnemonics jl and jlt are used. It is up to the programmer to select the correct instruction. For example, suppose that two bytes contain 0x99 and 0x01. They are related by 0x99 > 0x01 if the values are unsigned but 0x99 < 0x01 if they

are signed, twos complement numbers because 0x99 is the representation of −0x67.

The following table shows the list of 27 core instructions of MSP430:

S.No.	Mnemonic	S-Reg, D- Reg	Operation	Status Bits			
				V	N	Z	C
1	MOV	src,dst	src → dst	-	-	-	-
2	ADD	src,dst	src + dst → dst	*	*	*	*
3	ADDC	src,dst	src + dst + C → dst	*	*	*	*
4	SUB	src,dst	dst + .not.src + 1 → dst	*	*	*	*
5	SUBC	src,dst	dst + .not.src + C → dst	*	*	*	*
6	CMP	src,dst	dst → src	*	*	*	*
7	DADD	src,dst	src + dst + C → dst (decimally)	*	*	*	*
8	BIT	src,dst	src .and. dst	0	*	*	Z
9	BIC	src,dst	not src .and. dst → dst	-	-	-	-
10	BIS	src,dst	src .or. dst → dst	-	-	-	-
11	XOR	src,dst	src .xor. dst → dst	*	*	*	Z
12	AND	src,dst	src .and. dst → dst	0	*	*	Z
13	RRC	dst	C → MSB →.....LSB → C	*	*	*	*
14	RRA	dst	MSB → MSB →....LSB → C	0	*	*	*
15	PUSH	src	SP - 2 → SP, src → SP	–	–	–	–
16	SWPB	dst	bit 15...bit 8 ↔ bit 7...bit 0	–	–	–	–
17	CALL	dst	Call subroutine in lower 64KB	–	–	–	–
18	RETI		TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SP	*	*	*	*
19	SXT	dst	Register mode: bit 7 → bit 8...bit 19 Other modes: bit 7 → bit 8...bit 15	0	*	*	Z
20	JEQ/JZ	Label	Jump to label if zero bit is set	Status bits are not affected			
21	JNE/JNZ	Label	Jump to label if zero bit is reset				
22	JC	Label	Jump to label if carry bit is set				
23	JNC	Label	Jump to label if carry bit is reset				
24	JN	Label	Jump to label if negative bit is set				
25	JGE	Label	Jump to label if (N .XOR. V) = 0				
26	JL	Label	Jump to label if (N .XOR. V) = 1				

27	JMP	Label	Jump to label unconditionally	

Note:

*=Statusbitisaffected.

–=Statusbitisnotaffected.

0=Statusbitiscleared.

1=Statusbitisset.

The following table shows the list of 24 Emulated Instructions:

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 a core instruction. There is no code or performance penalty for using emulated instructions.

S.No.	Instruction	Explanation	Emulation	Status Bits			
				V	N	Z	C
1	ADC dst	Add Carry to dst	ADDC #0,dst	*	*	*	*
2	BR dst	Branch indirectly dst	MOV dst,PC	–	–	–	–
3	CLR dst	Clear dst	MOV #0,dst	–	–	–	–
4	CLRC	Clear Carry bit	BIC #1,SR	–	–	–	0
5	CLRN	Clear Negative bit	BIC #4,SR	–	0	–	–
6	CLRZ	Clear Zero bit	BIC #2,SR	–	–	0	–
7	DADC dst	Add Carry to dst decimally	DADD #0,dst	*	*	*	*
8	DEC dst	Decrement dst by 1	SUB #1,dst	*	*	*	*
9	DECD dst	Decrement dst by 2	SUB #2,dst	*	*	*	*
10	DINT	Disable interrupt	BIC #8,SR	–	–	–	–
11	EINT	Enable interrupt	BIS #8,SR	–	–	–	–
12	INC dst	Increment dst by 1	ADD #1,dst	*	*	*	*
13	INCD dst	Increment dst by 2	ADD #2,dst	*	*	*	*
14	INV dst	Invert dst	XOR #–1,dst	*	*	*	*
15	NOP	No operation	MOV R3,R3	–	–	–	–
16	POP dst	Pop operand from stack	MOV @SP+,dst	–	–	–	–
17	RET	Return from subroutine	MOV @SP+,PC	–	–	–	–
18	RLA dst	Shift left dst arithmetically	ADD dst,dst	*	*	*	*
19	RLC dst	Shift left dst logically through Carry	ADDC dst,dst	*	*	*	*
20	SBC dst	Subtract Carry from dst	SUBC #0,dst	*	*	*	*
21	SETC	Set Carry bit	BIS #1,SR	–	–	–	1
22	SETN	Set Negative bit	BIS #4,SR	–	1	–	–
23	SETZ	Set Zero bit	BIS #2,SR	–	–	1	–
24	TST dst	Test dst (compare with 0)	CMP #0,dst	0	*	*	1

Note:

*=Statusbitisaffected.

–=Statusbitisnotaffected.

0=Statusbitiscleared.

1=Statusbitisset.

Instruction Formats:

There are three core-instruction formats:

1) Double operand (Format I)
2) Single operand (Format II)
3) Jump (Format III)

Note: The Instruction Formats can be used to find the Machine codes manually for assembly language instructions

- **opcode-** is the operation code
- **src-**The source operand defined by As and S-Reg
- **dst-** The destination operand defined by Ad and D-Reg
- **As** (2 bits-addressing bits) gives the mode of addressing for the source, which has four basic modes.
- **Ad** (1 bit-addressing bits) similarly gives mode of addressing for the destination, which has only two basic modes.
- **S-Reg** and **D-Reg** specify the CPU registers associated with the source and destination, the registers either contain the data or addresses.
- **B/W** (1 bit) Byte or Word operation:
  - 0: word operation, 1: byte operation

As Bits		*Addressing Mode*
0	**0**	**Register**
0	**1**	**Indexed**
1	**0**	**Indirect Reg.**
1	**1**	**Indirect Auto-Increment /Immediate**

Ad Bit	*Addressing Mode*
0	**Register**
1	**Indexed**

Double-Operand (Format I) Instructions:

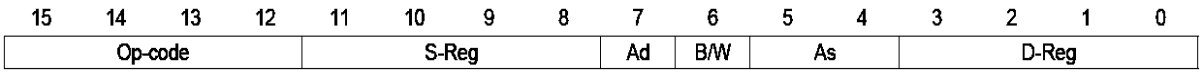


Figure: Double Operand Instruction Format

Single-Operand (Format II) Instructions:

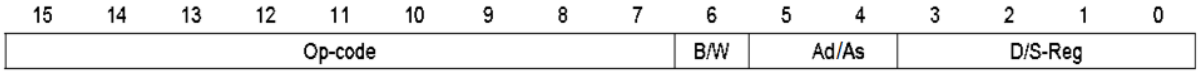


Figure: Single Operand Instruction Format

Jump Instruction Format:

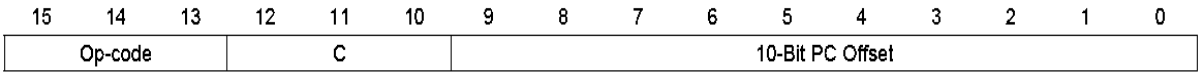


Figure: Jump Instruction Format

Here is an example of a move from register to register with the resulting machine code:

```
MOV.w R5, R6          ; 4506
```

The instruction can be broken into its fields of opcode = 4, S-reg = 5, Ad = 0, B/W = 0,As= 0, D-reg = 6. What do these mean?

- ⇨ The opcode of 4 represents a move.
- ⇨ The bit B/W = 0 shows that the operand is a word.
- ⇨ The addressing mode for the source is As = 0, which is register. The register is S-reg = 5, which is R5 as expected.
- ⇨ Similarly, the addressing mode for the destination is Ad = 0, which again means register. The register is D-reg = 6 = R6.

Here is another example addition rather than a move:

```
ADD.w R5, R6          ; 5506
```

The machine code is identical except for the opcode which is 5 rather than 4. The specification of the operands is unchanged. This is because of the orthogonality: All instructions use the same addressing modes.

Let us move an immediate value instead of a register:

```
MOV.w #5, R6          ; 4036 0005
```

Now there are two words. The fields of the instruction are opcode = 4, S-reg = 0, Ad = 0, B/W = 0, As = 3 = 11b, D-reg = 6. The difference is in the specification of the source, which means immediate operand. The register is S-Reg = 0. The value itself is contained in the second word in the machine code.

The following table shows the **opcodes** for core instructions:

OPCODE (HEX)	CORE INSTRUCTION
4	MOV
5	ADD
6	ADDC
7	SUB

8	SUBC
9	CMP
A	DADD
B	BIT
C	BIC
D	BIS
E	XOR
F	AND

OPCODE (BINARY)	**CORE INSTRUCTION**
000100000	RRC
000100001	SWPB
000100010	RRA
000100011	SXT
000100100	PUSH
000100101	CALL
000100110	RETI

OPCODE (BINARY)	**CONDITION (C) (BINARY)**	**CORE INSTRUCTION**
001	000	JNE/JNZ
001	001	JEQ/JZ
001	010	JNC/JLO
001	011	JC/JHS
001	100	JN
001	101	JGE
001	110	JL
001	111	JMP

Instruction Timing:

- It takes one cycle to fetch the instruction word itself. This is all if both source and destination are in CPU registers.
- One more cycle is needed to fetch the source if it is given indirectly as @Rn or @Rn+, in which case the address is already in the CPU. This includes immediate data.
- Alternatively, two more cycles are needed if one of the indexed modes is used. The first is to fetch the base address, which is added to the value in a CPU register to get the address of the source. A second cycle is necessary to fetch the operand itself. This includes absolute and symbolic modes.
- Two more cycles are needed to fetch the destination in the same way if it is indexed.
- A final cycle is needed to write the destination back to memory if required; no allowance is needed for a register in the CPU.

Table: Number of MCLK cycles required for typical instructions. It applies only to logical and arithmetic instructions and when the destination is not PC.

Format I Destination	Source		
	Rs	@Rs, @Rs+	S(Rs)
Rd	1	2	3
D(Rd)	4	5	6
Format II	1	3	4

(a) Two operands (Format I), destination is register.

`add.w Rs, Rd`	`add.w @Rs, Rd`	`add.w S(Rs), Rd`
fetch instruction	fetch instruction	fetch instruction
	fetch source @Rs	fetch S
		fetch source S(Rs)

(b) Two operands (Format I), destination is indexed.

`add.w Rs, D(Rd)`	`add.w @Rs, D(Rd)`	`add.w S(Rs), D(Rd)`
fetch instruction	fetch instruction	fetch instruction
fetch D	fetch source @Rs	fetch S
fetch destination D(Rd)	fetch D	fetch source S(Rs)
write destination D(Rd)	fetch destination D(Rd)	fetch D
	write destination D(Rd)	fetch destination D(Rd)
		write destination D(Rd)

(c) One operand (Format II)

`rra.w Rs`	`rra.w @Rs`	`rra.w S(Rs)`
fetch instruction	fetch instruction	fetch instruction
	fetch source @Rs	fetch S
	write source @Rs	fetch source S(Rs)
		write source S(Rs)

Figure: Cycle-by-cycle operation of typical instructions, showing the traffic with memory.

Variants of the MSP430 Family:

MSP430x1xx:

- Provides a wide range of general purpose devices from simple versions to complete systems for processing signals
- There is a broad selection of peripherals and some include a hardware multiplier, which can be used as rudimentary digital signal processor
- Packages have 20–64 pins

MSP430x2xx:

- Introduced in 2005.
- CPU can run at 16 MHz, double the speed of earlier devices, while consuming only half the current at the same speed.
- 14 pin PDIP package.
- Pull-up or pull-down resistors are provided on the inputs to reduce the number of external components needed.
- Even the smallest,14-pin devices offer a 16-bit sigma–delta ADC

MSP430x3xx:

- The original family, which includes drivers for LCDs. It is now obsolescent.

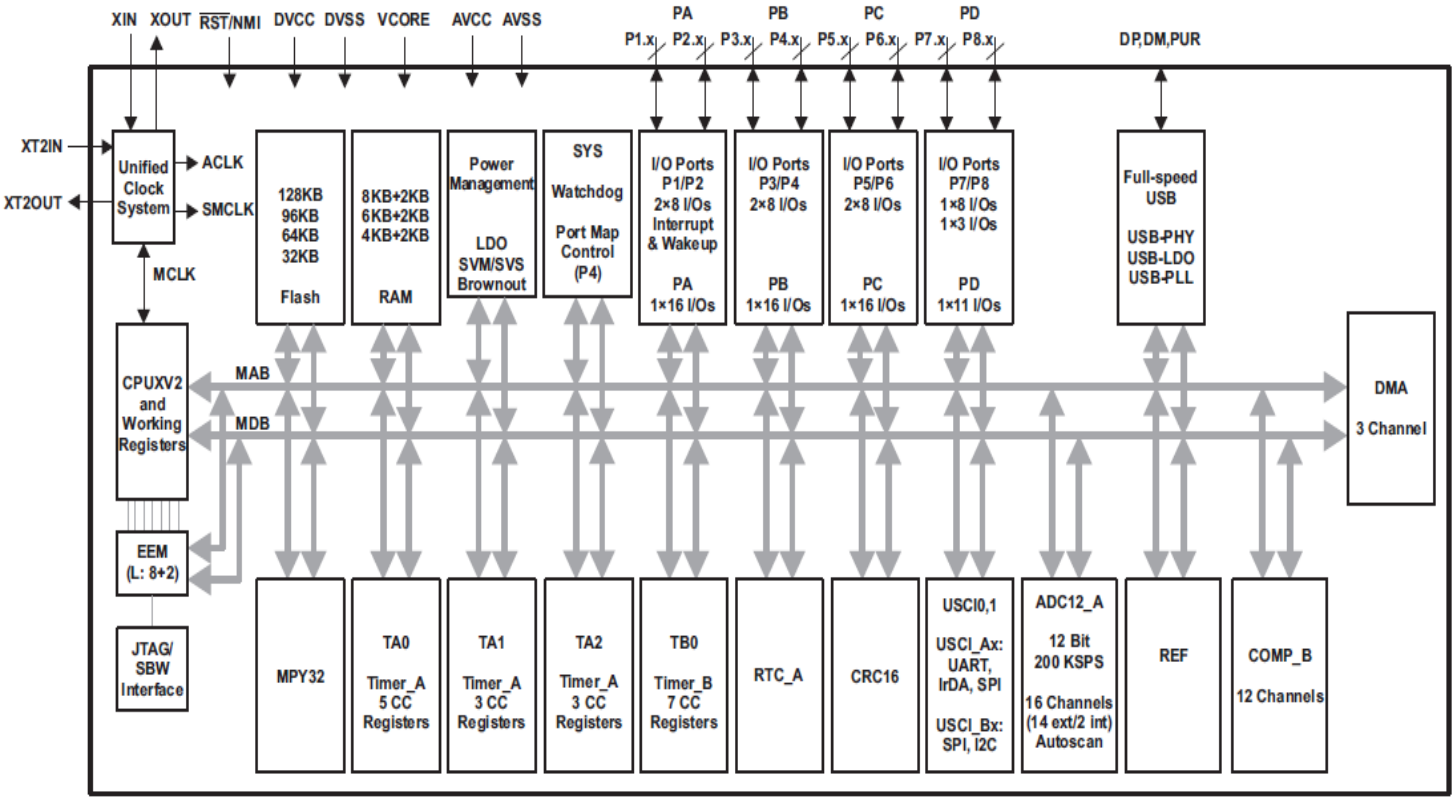
MSP430x4xx:

- Can drive LCDs with up to 160 segments. Many of them are ASSPs (application-specific standard product), but there are general-purpose devices as well. Their packages have 48–113 pins, many of which are needed for the LCD.

MSP430x5xx:

- It is Next Generation of MSP430 Family
- Advanced Ultra Low Power features
- Increased Performance, Functionality and ease-of-use
- Significantly longer battery life
- It contains almost all Peripherals like PORTS (P1-P8), ADC, DAC, TIMER_A0-2 & B0, DMA, COMPARATOR, USCI: UART, SPI, I2C IRDA, USB etc.
- Lowest Active Current/MHz:
  - <200uA/MHz

MSP430x5xx Series Block Diagram:



CPUX:

The MSP430X CPU is RISC architecture with 51 instructions and 7 addressing modes. It is integrated with 16 registers (each 20-bit wide except SR) that provide reduced instruction execution time. The register-to-register operation execution time is one cycle of the CPU clock. Peripherals are connected to the CPU using data, address, and control buses, and can be handled with all instructions. It has 20-bit address bus allows direct access and branching throughout the entire memory range without paging.

JTAG (Joint Test Action Group):

The MSP430 family supports the standard JTAG interface which requires four signals for sending and receiving data. The JTAG signals are shared with general-purpose I/O. It is used to program and debug the device.

SBW (Spy-Bi-Wire) Interface:

In addition to the standard JTAG interface, the MSP430 family supports the two wire Spy-Bi-Wire interface. Spy-Bi-Wire can be used to interface with MSP430 development tools and device programmers.

Flash Memory:

The flash memory can be programmed through the JTAG port, Spy-Bi-Wire (SBW), the BSL, or in-system by the CPU. The CPU can perform single-byte, single-word, and long-word writes to the flash memory.

The RAM is made up of n sectors. Each sector can be completely powered down to save leakage; however; all data is lost.

RAM has 5 sectors. The size of a each sector is 2KB. In that 5 sectors one is for USB & RAM (Both) and remaining 4 sectors only for RAM.

Peripherals:

Peripherals are connected to the CPUX through data, address, and control buses. Peripherals can be handled using all instructions.

On-Chip Peripherals (Analog and Digital):

- Digital I/O PORTs (GPIO)
- Port Mapping Controller
- Power Management Module (PMM)
- Hardware Multiplier (MPY32)
- Real-Time Clock (RTC_A)
- Watchdog Timer (WDT_A)
- System Module (SYS)
- DMA Controller
- Universal Serial Communication Interface (USCI: UART Mode, SPI Mode, I2C Mode)
- Timers (TA0, TA1, TA2, TB0)
- Comparator_B
- Analog to Digital Convertor (ADC12_A)
- Cyclic Redundancy Check (CRC16)
- Universal Serial Bus (USB)
- Embedded Emulation Module (EEM)

Digital I/O PORTs:

- There are up to eight 8-I/O ports P1- P8 each Port is 8 bit wide
- All individual I/O bits are independently programmable.
- Any combination of input, output, and interrupt conditions is possible.
- Pull-up or Pull-down on all ports is programmable.
- Read and write access to port-control registers is supported by all instructions.
- Ports can be accessed byte-wise (P1 through P8) or word-wise in pairs (PA through PD).
- Independent input and output data registers

Port Mapping Controller:

The port mapping controller allows the flexible and reconfigurable mapping of digital functions to port P4.

Power Management Module (PMM):

- The PMM includes an integrated voltage regulator that supplies the core voltage to the device and contains programmable output levels to provide for power optimization.
- The PMM also includes supply voltage supervisor (SVS) and supply voltage monitoring (SVM) circuitry, as well as brownout protection.
- The brownout circuit is implemented to provide the proper internal reset signal to the device during power on and power off.
- The SVS and SVM circuitry detects if the supply voltage drops below a user-selectable level and supports both supply voltage supervision (SVS) (the device is automatically reset) and supply voltage monitoring (SVM) (the device is not automatically reset).

Hardware Multiplier:

- The multiplication operation is supported by a dedicated peripheral module. The module performs operations with 32-, 24-, 16-, and 8-bit operands. The module supports signed and unsigned multiplication as well as signed and unsigned multiply-and-accumulate operations.

Real-Time Clock (RTC_A):

- The RTC_A module can be used as a general-purpose 32-bit counter (counter mode) or as an integrated real-time clock (RTC) (calendar mode).
- In counter mode, the RTC_A also includes two independent 8-bit timers that can be cascaded to form a 16-bit timer/counter. Both timers can be read and written by software.
- Calendar mode integrates an internal calendar which compensates for months with less than 31 days and includes leap year correction. The RTC_A also supports flexible alarm functions and offset calibration hardware.

Watchdog Timer (WDT_A):

The primary function of the WDT_A module is to perform a controlled system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can be configured as an interval timer and can generate interrupts at selected time intervals.

System Module (SYS):

The SYS module handles many of the system functions within the device.

These include power-on reset and power-up clear handling, NMI source selection and management, reset interrupt vector generators, bootstrap loader entry mechanisms, and configuration management.

DMA Controller:

The DMA controller allows movement of data from one memory address to another without CPU intervention.

Universal Serial Communication Interface (USCI: UART Mode, SPI Mode, I2C Mode):

The USCI modules are used for serial data communication. The USCI module supports synchronous communication protocols such as SPI (3-pin or 4-pin) and I2C, and asynchronous communication protocols such as UART, enhanced UART with automatic baud rate detection, and IrDA.

Timers (TA0, TA1, TA2, TB0):

Timers are 16-bit timer and counter (Timer_A/B type) with 5/3/3/7 capture/compare registers. It can support multiple capture/compare registers, PWM outputs, and interval timing. It also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Comparator_B:

The primary function of the Comparator_B module is to support precision slope analog-to-digital conversions, battery voltage supervision, and monitoring of external analog signals.

ADC12_A:

The ADC12_A module supports fast 12-bit analog-to-digital conversions. It has 16 independent channels to be convert and store without any CPU intervention.

CRC16:

A Cyclic Redundancy Check (CRC) is an error-detecting code commonly used in digital networks and storage devices for data errors checking purpose

REF Voltage Reference:

The REF voltage module generates the Reference Voltage which is used by ADC as a reference mark or point to convert analog voltage from 0v to REF voltage.

Universal Serial Bus (USB):

The USB module is a fully integrated USB interface that is compliant with the USB 2.0 specification. The module supports full-speed operation of control, interrupt, and bulk transfers. The module includes an integrated LDO, PHY, and PLL.

Embedded Emulation Module (EEM):

The EEM supports real-time in-system debugging.

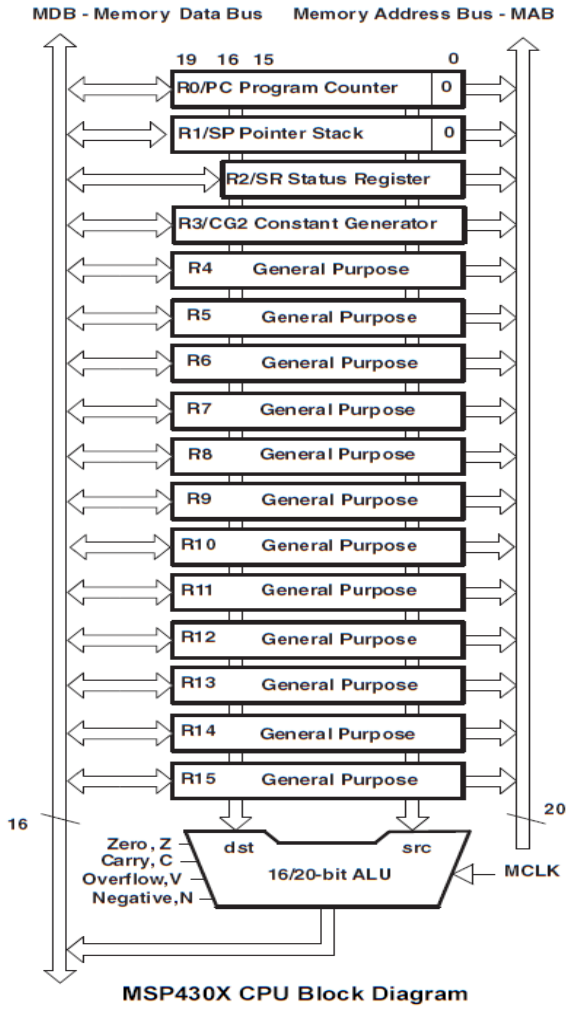
Features of EEM:

- Eight hardware triggers or breakpoints on memory access
- Two hardware triggers or breakpoints on CPU register write access
- Up to 10 hardware triggers can be combined to form complex triggers or breakpoints

- Two cycle counters
- Sequencer
- State storage
- Clock control on module level

MSP430X CPU (CPUX) – Features:

The MSP430X CPU features include:



MSP430X CPU Block Diagram

- RISC architecture
- Orthogonal architecture
- Full register access including program counter, status register and stack pointer
- Single-cycle register operations
- Large register file reduces fetches to memory
- It has 51 instructions
- 20-bit address bus allows direct access and branching throughout the entire memory range without paging
- 16-bit data bus allows direct manipulation of word-wide arguments
- Constant generator provides the six most often used immediate values and reduces code *size*
- *Direct memory-to-memory transfers without intermediate register* holding
- Byte, word, and 20-bit address-word addressing.

➢ An **orthogonal** instruction set is an instruction set architecture where all instruction types can use all addressing modes. It is "orthogonal" in the sense that the instruction type and the addressing mode vary independently.

Registers of MSP430 CPUX:

The CPUX of MSP 430 includes a 16/20-bit ALU and a set of 16 Registers R0 – R15. In these registers four are special Purpose and 12 are general purpose registers. All the registers can be addressed in the same way.

The special Purpose Registers are:

PC (Program Counter), SP (Stack Pointer), SR (Status Register), CGx (Constant Generator)

The MSP430 CPU includes an arithmetic logic unit (ALU) that handles addition, subtraction, comparison and logical (AND, XOR) operations. ALU operations can affect the overflow, zero, negative, and carry flags in the status register.

The following figure shows the register organization of MSP430 CPUX.

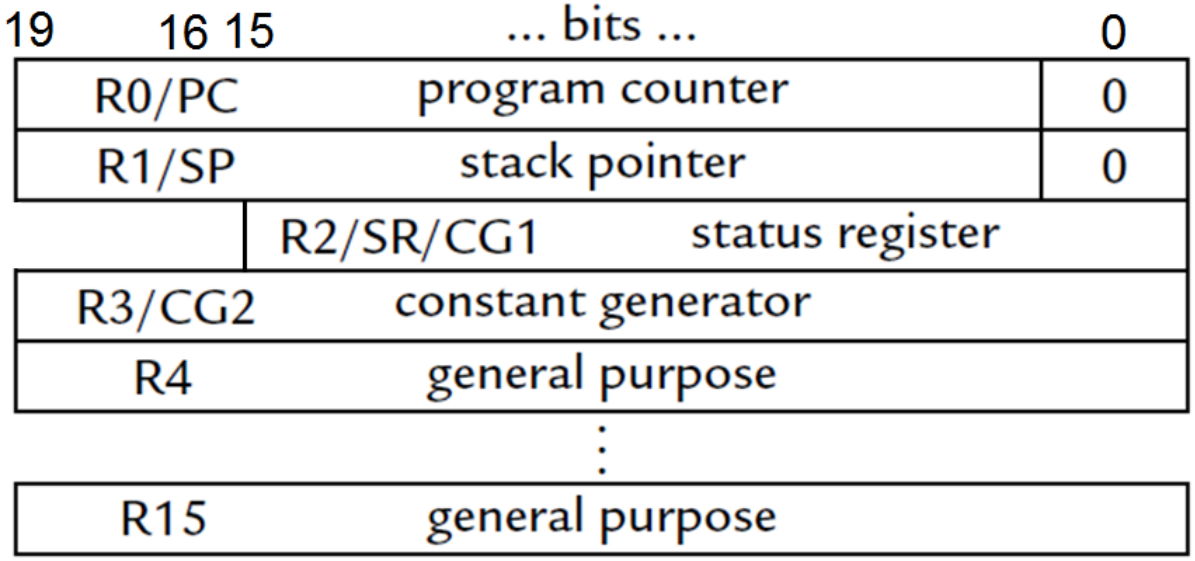
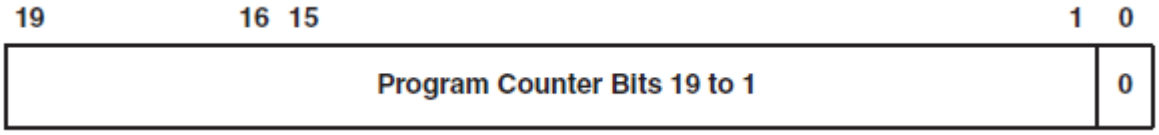


Figure: Registers in the CPUX of MSP430x5xx

R0: Program Counter (PC):

The 20-bit PC (PC/R0) points to the next instruction to be executed. Each instruction uses an evennumber of bytes (2, 4, 6, or 8 bytes), and the PC is incremented accordingly. Instruction accesses areperformed on word boundaries, and the PC is aligned to even addresses.Following Figure shows the PC structure.

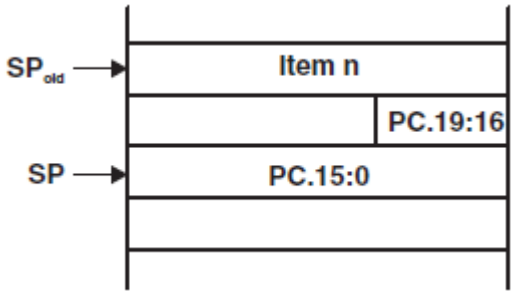
Subroutines and interrupts also modify the PC but in these cases the previous value (Next line of current instruction which is executing) is saved on the stack and



restored later.

Figure: Program Counter

The PC is automatically stored on the stack with CALL (or CALLA) instructions and during an interruptservice routine. Following Figure shows the storage of the PC with the return address after a CALLA instruction.A CALL instruction stores only bits 15:0



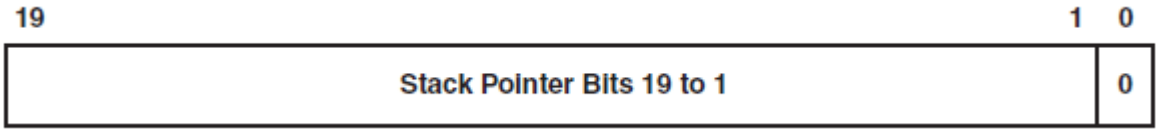
of the PC.

Figure: PC Storage on the Stack for CALLA

The RETA instruction restores bits 19:0 of the PC and adds 4 to the stack pointer (SP). The RET instruction restores bits 15:0 to the PC and adds 2 to the SP.

R1: Stack Pointer (SP):

The 20-bit SP (SP/R1) is used by the CPU to store the return addresses of subroutine calls and interrupts. It uses a predecrement, postincrement scheme. In addition, the SP can be used by software with all instructions and addressing modes.



Following Figure shows the SP. The SP is initialized into RAM by the user,and is always aligned to even addresses.

Figure: Stack Pointer

The Following Figure shows the stack usage.

```
PUSH    #0123h              ; Put 0123h on stack
POP     R8          ; R8 = 0123h
```

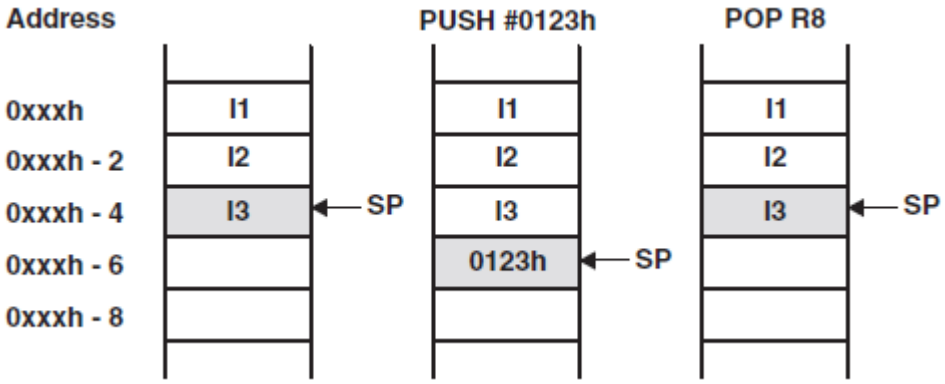
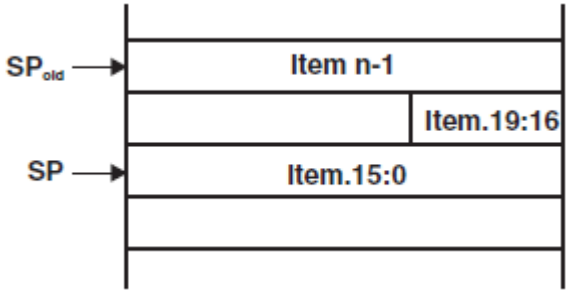


Figure: Stack Usage



The following Figure shows the stack usage when 20-bit address words arepushed.

Figure: PUSHX.A Format on the Stack

Note: For programs written in C, the compiler initializes the stack automatically as part of the startup code, which runs silently before the program starts, but you must initialize SP yourself in assembly language.

R2: Status Register (SR):

The 16-bit SR (SR/R2), used as a source or destination register, can only be used in register modeaddressed with word instructions. The remaining combinations of addressing modes are used to supportthe constant generator. Figure 4-9 shows the SR bits. Do not write 20-bit values to the SR. Unpredictable operation can result.

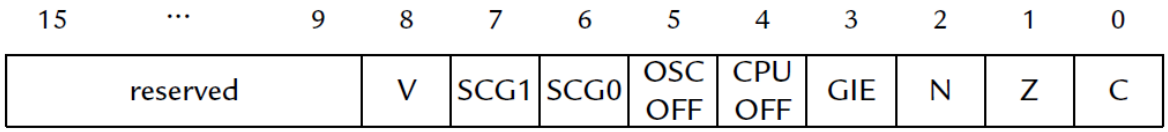


Figure: Individual bits in the status register

The reserved bits are not used in the MSP430.

Table: SR Bit Description

Bit	Description	
Reserved	Reserved	
V	Overflow.Thisbitissetwhentheresultofanarithmeticoperationoverflowsthesigned-variablerange.	
	ADD(.B), ADDX(.B,.A), ADDC(.B), ADDCX(.B.A), ADDA	Set when: positive+ positive=negative negative+ negative=positive otherwise reset
	SUB(.B), SUBX(.B,.A), SUBC(.B), SUBCX(.B,.A), SUBA, CMP(.B), CMPX(.B,.A), CMPA	Set when: positive–negative=negative negative–positive=positive otherwise reset
SCG1	Systemclockgenerator1.Thisbitmaybetoenable/disablefunctionsintheclocksystemdependingonthedevice family; for example, DCO bias enable/disable	
SCG0	Systemclockgenerator0.Thisbitmaybeusedtoenable/disablefunctionsintheclocksystem dependingonthe device family; for example, FLL disable/enable	
OSCOFF	Oscillator off. This bit, when set, turns off the LFXT1 crystal oscillator when LFXT1CLK is not used for MCLK or SMCLK.	
CPUOFF	CPU off. This bit, when set, turns off the CPU.	
GIE	General interrupt enable. This bit, when set, enables maskable interrupts. When reset, all maskable interrupts are disabled.	
N	Negative.Thisbitissetwhentheresultofanoperationisnegativeandclearedwhentheresult ispositive.	

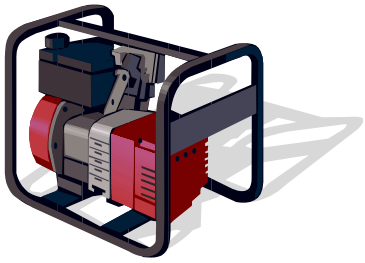
Z	Zero.Thisbitissetwhentheresultofanoperationis0andclearedwhentheresultisnot0.
C	Carry.Thisbitissetwhentheresultofanoperationproducedacarryandclearedwhennocarryoccurred.

R2/R3: Constant Generator Registers (CG1/CG2):

Six commonly-used constants are generated with the constant generator registers R2

Table: Values of Constant Generators CG1, CG2

Register	As	Constant	Remarks
R2	00	–	Register mode
R2	01	(0)	Absolute address mode
R2	10	00004h	+4,bitprocessing
R2	11	00008h	+8,bitprocessing
R3	00	00000h	0,wordprocessing
R3	01	00001h	+1
R3	10	00002h	+2,bitprocessing
R3	11	FFh,FFFFh,FFFFFh	–1,wordprocessing



```
4314            mov.w #0002h, R4   ; With CG
4034 1234       mov.w #1234h, R4   ; Without CG
```

- Constant (Immediate) values -1,0,1,2,4,8 generated in hardware

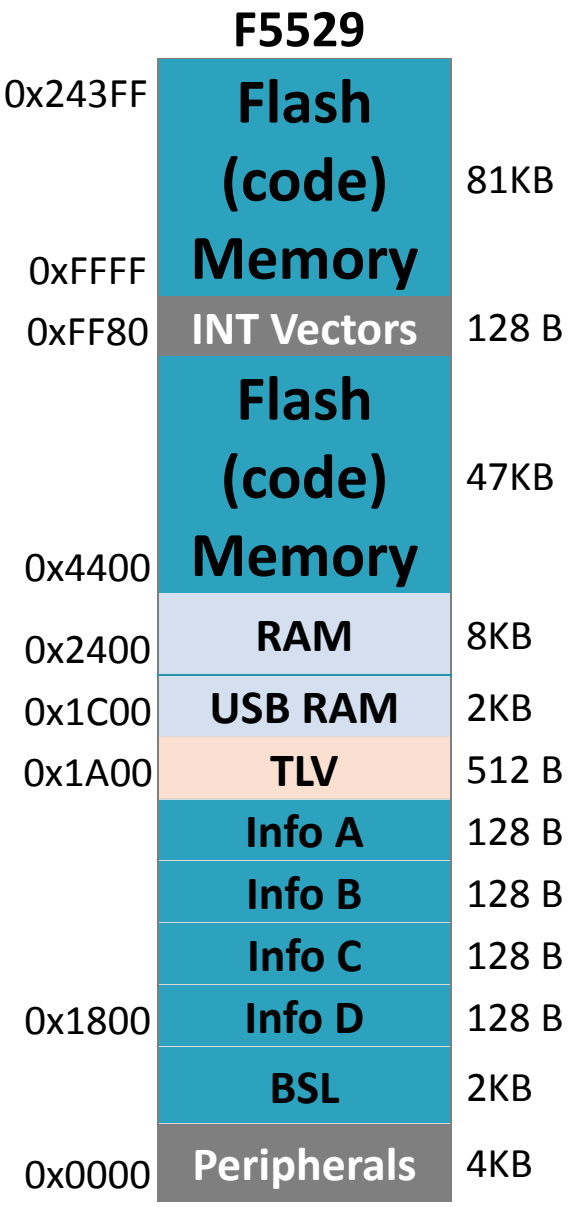
The constant generator advantages are:

- No special instructions required
- No additional code word for the six constants
- No code memory access required to retrieve the constant
- Reduces code size and cycles
- Completely Automatic

R4 - R15: General–Purpose Registers:

The remaining 12 registers R4–R15 have no dedicated purpose and may be used as general working registers. They may be used for either data or addresses because both are 16-bit values, which simplify the operation significantly.

Address Space (Memory Organization):



MEMORY TYPE		MSP430F5529 MSP430F5528 MSP430F5519
Unused Memory		0FFFFFh–024400h
Interrupt Vector	Size	128 B 00FFFFh–00FF80h
Flash (code) Memory	Total Size	128KB
	Bank D	32KB 0243FFh–01C400h
	Bank C	32KB 01C3FFh–014400h
	Bank B	32KB 0143FFh–00C400h
	Bank A	32KB 00C3FFh–004400h
RAM	Sector 3	2KB 0043FFh–003C00h
	Sector 2	2KB 003BFFh–003400h
	Sector 1	2KB 0033FFh–002C00h
	Sector 0	2KB 002BFFh–002400h
USB RAM#	Sector 7	2KB 0023FFh–001C00h
Information Memory (flash)	Info A	128 B 0019FFh–001980h
	Info B	128 B 00197Fh–001900h
	Info C	128 B 0018FFh–001880h
	Info D	128 B 00187Fh–001800h
Bootstrap loader (BSL) memory (flash)	BSL 3	512 B 0017FFh–001600h
	BSL 2	512 B 0015FFh–001400h
	BSL 1	512 B 0013FFh–001200h
	BSL 0	512 B 0011FFh–001000h
Peripherals	Size	4KB 000FFFh–0h

1MB

- **Info** – Information Memory (flash)
- **TLV** – Contents of the Device Descriptor **Tag Length Value** (TLV)
- **BSL**– Bootstrap Loader Memory (flash)

Sample Embedded System onMSP430 Microcontroller:

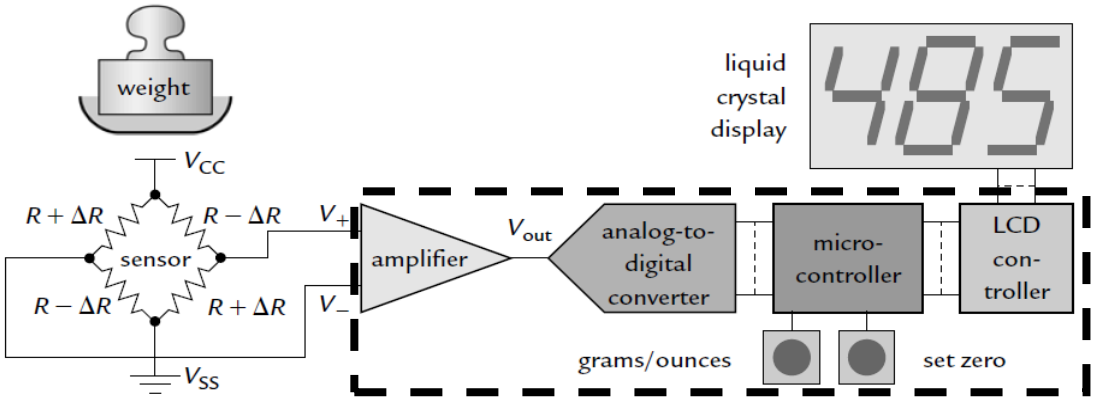


Fig: Weighing Machine with a liquid crystal display, broken down intoindividual functions.

*V*out = *A*(*V*+ − *V*−), where *A* is the gain

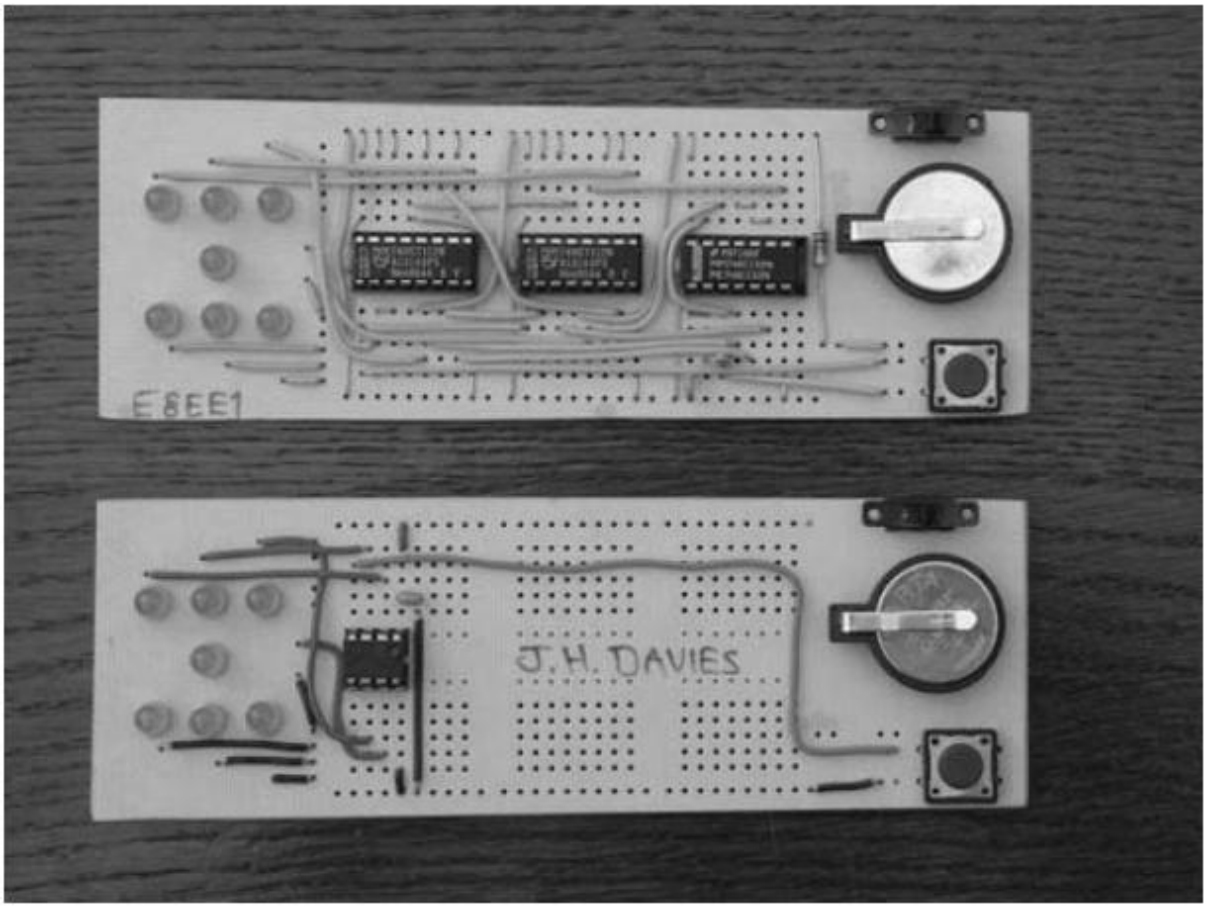


Figure: Electronic Dice built using (top) *JK* flip-flops and gates and (bottom) an eight-pin microcontroller.