UNIT-3

MSP430 MICROCONTROLLER:

- The MSP430 MCU is designed specifically for ultra-low-power applications.

Features of MSP430:

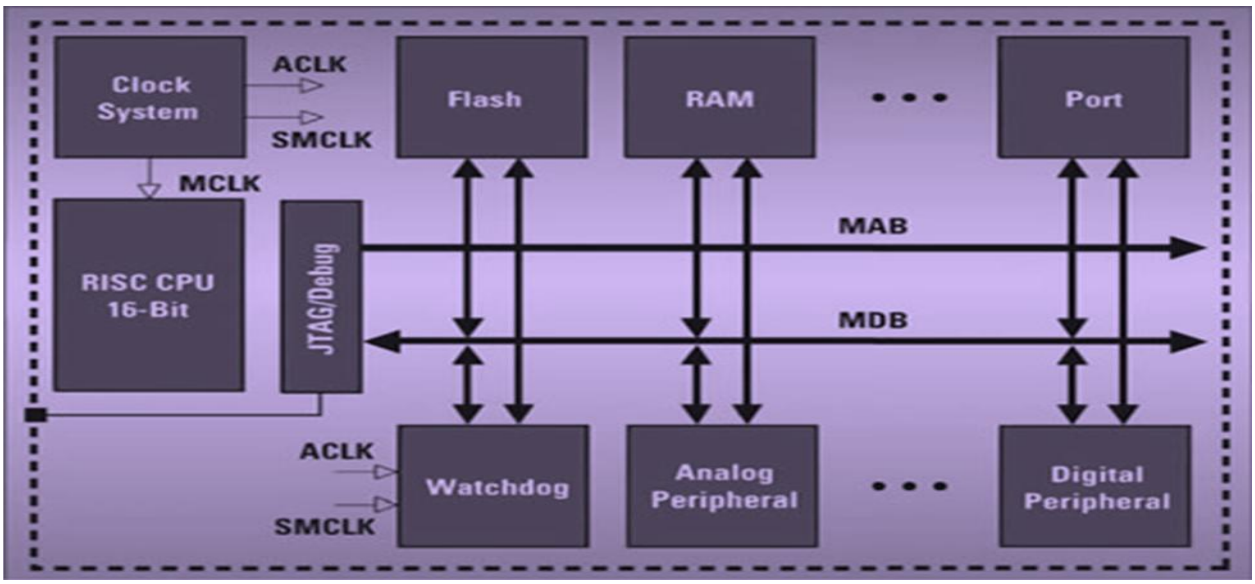
- Ultra-low-power (ULP) architecture and flexible clock system extend battery life
- Flexible clocking system.
- It is having 5 low power modes: LPM 0 – 4.
- Low power consumption:

  0.1 µ A for RAM data Retention,

  0.8 µ A for RTC mode operation

  250 µA /MIPS at active operation.
- Quick wake-up time from standby mode and it depends on no.of on-chip peripherals.
- Low operation voltage (from 1.8 V to 3.6 V).
- Enhanced libraries to benefit several applications such as capacitive touch, metering metrology, low power design and debugging
- Extensive interrupt capability relieves need for polling
- Flexible and powerful processing capabilities.
- Seven source-address modes
- Only 27 core instructions and 24 emulated instructions.
- Prioritized, nested interrupts
- Large register file
- Efficient table processing
- Fast hex-to-decimal conversion

BLOCK DIAGRAM OF MSP430:



- On the left is the CPU and its supporting hardware, including the clock generator. The emulation and JTAG interface are used to communicate with a desktop computer when downloading a program and for debugging.

- These MSP controller families share a 16-bit CPU core, RISC type, intelligent peripherals and flexible clock system that interconnect using a 16-bit Von Neumann common memory address bus (MAB) and memory data bus (MDB) architecture.
- It is having no.of general purpose I/O ports, each of size 8-bit. All the pins can be configured either as input or output to interface digital signal.
- This architecture is rich in on-chip analog and digital peripherals.

Analog Peripherals: A/D Converters, Comparator, LCD Driver, Supply Voltage Supervisor

Digital Peripherals: Watch-dog timer, 16-bit and 8-bit Timers, Hardware Multiplier, Universal Serial Communication Interface (USCI) etc..,

Clock Generator:

The MSP430 addresses the conflicting demands for high performance, low power, and a precise frequency by using three internal clocks, which can be derived from up to four sources. These are the internal clocks, which are the same in all devices:

• Master clock, MCLK, is used by the CPU.

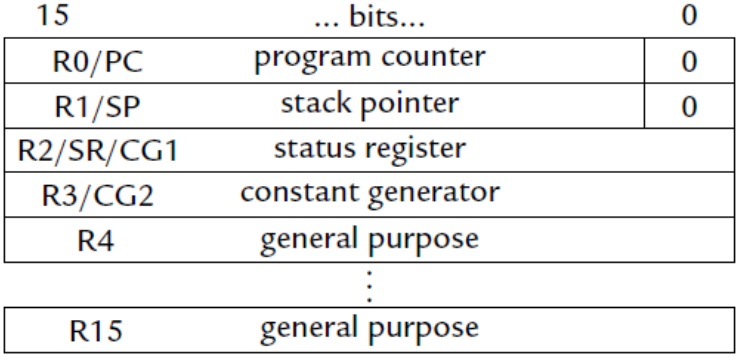
• Subsystem master clock, SMCLK, is distributed to peripherals.

• Auxiliary clock, ACLK, is also distributed to peripherals.

Memory: These devices have flash memory, 2KB to 32KB and 128 bytes to 2KB of RAM. The size of memories varies from family to family.

Central Processing Unit

- 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.
  - The special Purpose Registers are PC (Program Counter), SP (Stack Pointer) , SR (Status Register) and CGx (Constant Generator)



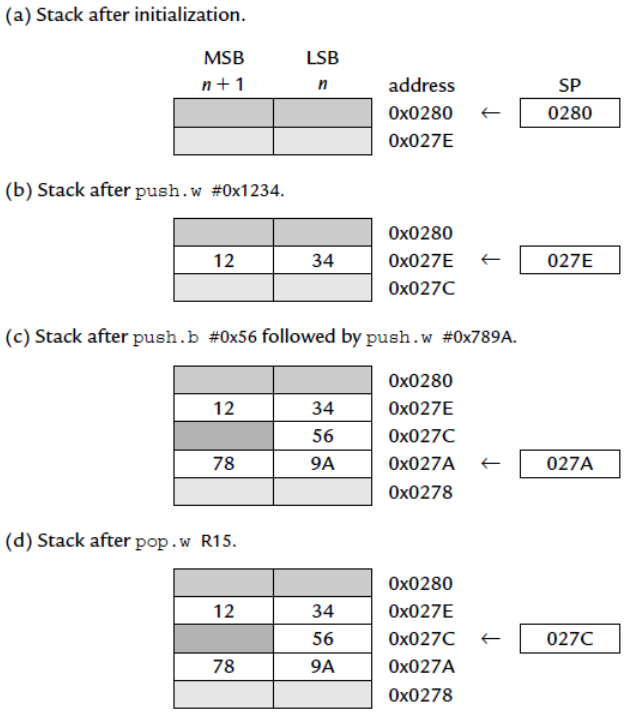
R0 (Program Counter (PC)):

This contains the address of the next instruction to be executed—"points to" the instruction in the code memory.

R1 (Stack Pointer (SP)):

When a subroutine is called the CPU must jump to the subroutine, execute the code there, and finish by returning to the instruction after the call. It must therefore keep track of the contents of the PC before jumping to the subroutine so that it can return afterward. This is done with a *stack*, which is also known as a last in–first out (LIFO) data structure.The stack is allocated at the top of RAM and grows *down* toward low addresses. The

stack pointer holds the address of the top of the stack. The operation of the stack is illustrated in below Figure.



The word 0x1234 has been added or *pushed* on to the stack. The value of SP is first decreased by 2. A word has been removed, pulled or *popped* from the stack into the register R15 the stack pointer is increased by 2.

R2 (Status Register (SR)):

15 … 9	8	7	6	5	4	3	2	1	0
reserved	V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C

The C, Z, N, and V bits are affected by many of the operations performed by the ALU

- The *carry* bit C is set when an extra bit generated from the result of an arithmetic operation.
- The *zero* flag Z is set when the result of an operation is 0. A common application is to check whether two values are equal
- 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 overflow* flag V is set when the result of a signed operation has overflowed.

Enable Interrupts

Setting the *general interrupt enable* or GIE bit enables maskable interrupts. Clearing the bit disables all maskable interrupts. There are also nonmaskable interrupts, which cannot be disabled with GIE.

Control of Low-Power Modes

The CPUOFF, OSCOFF, SCG0, 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 (LPM0 – 4).

R2 and R3 (Constant Generators – CG1 & CG2)

Both R2 and R3 are used to provide the 6 most commonly used constants.

mov.w #0000h,R5 -------- Clears R5 register

add.w #0001h,R6 -------- Increments R6

General instructions shown above wasteful of both memory and time because the values would have to be fetched from memory whenever they were needed.To improve efficiency, if we use constant generators as source operands then the instructions look like

mov.w R3,R5 -------- Clears R5 register

add.w 0(R3),R6 -------- Increments R6 Register

Addressing Mode	R2 (CG1)	R3 (CG2)
Register	––	0000h
Indexed	0000h	0001h
Register Indirect	0004h	0002h
Indirect Auto-increment Register	0008h	FFFFh (–1)

General-Purpose Registers

The remaining 12 registers R4–R15 have no dedicated purpose and may be used as general working registers.

ADDRESSING MODES OF MSP430:

Addressing mode means the way of specifying the operands in an instruction. The MSP430 has seven addressing modes to interact with the CPU registers. The MSP430 supports seven addressing modes for the source operand and four addressing modes for the destination operand. They are

- Register mode
- Indexed mode
- Symbolic mode
- Absolute mode
- Indirect register mode
- Indirect auto increment register mode
- Immediate mode

1. Register Mode

This uses one or two of the registers in the CPU. It is the most straightforward addressing mode and is available for both source and destination.

Ex: MOV.W R4, R5 - Move (copy) the contents of source (register R4) to destination (R5). The registers are specified in the instruction word; no further data are needed. It is also the fastest mode and this instruction takes only 1 cycle. Any of the 16 registers can be used for either source or destination but there are some special cases:

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 is addressed by adding a constant base address to the contents of a CPU register; the value in the register is not changed.

Ex : MOV.B 3(R5), R4

Move (copy) the contents at source address (3+R5) to destination (register R4)

Indexed addressing can be used for the source, destination, or both.

3. Symbolic Mode

Symbolic mode allows the assignment of labels to fixed memory locations, so that those locations can be addressed directly with the assigned label name. For example, suppose that a program uses the variable

LoopCtr, which occupies a word. The following instruction stores the value of LoopCtr in R6 using symbolic mode.

Ex: **mov.w** LoopCtr ,R6 *; load word LoopCtr into R6*

4. Absolute Mode

It is similar to Symbolic mode, with the difference that the label is preceded by "&". The word following the instruction contains the absolute address.

Ex: **mov.b** &P1IN ,R6 *; load byte P1IN into R6*

Where PIN is the absolute address of the register. This addressing mode is used for special function and peripheral registers, whose addresses are fixed in the memory map. This addressing mode can be used for both source and destination operands.

5. Register indirect mode

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

Ex: **mov.w** @R5 ,R6 *; load word from address (R5)=4 into R6*

6. Indirect Autoincrement Register Mode

This is also applicable only for the source operand and the format is specified with a symbol '@' in front of the register and + sign after it, such as @Rn+. Here, Rn register value is used as data pointer and increments the register content after the operation by 1 (for byte operations) or 2 (for word operation).

Ex: **mov.w** @R5+,R6

A word is loaded from address 4 into R6 and the value in R5 is incremented to 6 because a word (2 bytes) was fetched.

This mode cannot be used for the destination.

7. Immediate Mode

In this addressing mode, the immediate data is specified as the operand in the instruction i.e., the data is readily available as a part instruction that will be fetched from the memory for the operation.

Immediate mode is used to assign constant values to registers or memory locations. The immediate data cannot be a destination operand.

Ex: **mov.w** #0900h,R5

INSTRUCTION SET OF MSP430:

The MSP430 instruction set consists of 27 core instructions. Additionally, it supports 24 emulated instructions. The core instructions have unique op-codes decoded by the CPU, while the emulated ones need assemblers and compilers to generate their mnemonics.The instruction set is orthogonal *with* few exceptions, meaning that all addressing modes can be used with all instructions and registers. There are three formats of instruction.

Double operand (Format I): Arithmetic and logical operations with two operands such as add.w src, dst. Both operands must be specified in the instruction.

Single operand (Format II): A mixture of instructions for control or to manipulate a single operand, which is effectively the *source or destination* for the addressing modes.

Jumps: The jump to the destination rather than its absolute address, in other words the offset that must be added to the program counter.The "return from interrupt" instruction reti is unique in requiring no operands. This would usually be described as *inherent* addressing but TI curiously classifies it as Format II without data.

1. 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. This is an excellent feature.

mov.w src ,dst *; move (copy)* *dst = src*

Stack Operations

These push data onto the stack and pop them off using stack pointer. The SP is fixed to be even, so a word of stack space is always consumed, even if only a byte is added.

push.w src *; push data onto stack *--SP = src*

pop.w dst *; pop data off stack dst = *SP++* ---------- *emulated*

2. Arithmetic and Logic Instructions with Two Operands

Binary Arithmetic Instructions with Two Operands

These are fairly standard. The carry bit should be interpreted as "not borrow" for subtraction:

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.w src ,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)*

The compare operation cmp is the same as subtraction sub 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.w dst *; clear dst = 0* ----------*emulated*
dec.w dst *; decrement dst – –* ----------*emulated*
decd.w dst *; double decrement dst -= 2* ---------- *emulated*
inc.w dst *; increment dst++* ---------- *emulated*
incd.w dst *; double increment dst += 2* ----------*emulated*
tst.w dst *; test (compare with 0) (dst - 0)* ---------- *emulated*

The test operation is the special case of comparison with 0. In many processors the clear operation differs from a move with the value 0 because a move sets the flags but a clear does not. This does not apply to the MSP430 because a move does not set the flags.

Logic Instructions with Two Operands

These are not quite the same as in many other processors:

and.w	src, dst	*; bitwise and* *dst &= src*
xor.w	src, dst	*; bitwise exclusive or* *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*

The MSP430 has the usual and and exclusive-OR xor instructions but not an explicit inclusive-OR. The and 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 invinstruction, 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
```

It is emulated using xorand inherits its peculiarity C = ~ Z. Its operand is a destination. It is not the same as changing the sign of a number, which is the twos complement.

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)
```

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
```

3. Shift and Rotate Instructions

Logical shift inserts zeroes for both right and left shifts.

Arithmetic shift inserts zeroes for left shifts but the most significant bit, which carries the sign, is replicated for right shifts.

Rotation does not introduce or lose any bits; bits that are moved out of one end of the register are passed around to the other.

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

Flow of Control

Subroutines, Interrupts, and Branches

These are mainly straightforward but there is a tricky point about addresses:

```
br      src    ;  branch (go to)          PC = src     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 less tricky:

```
jmp    label   ; unconditional jump
```

The conditional jumps are the "decision-making" instructions and test certain bits or combinations in the status register. It is not possible to jump according to the value of any other bits in SR or those in any other register. Typically a bit test instruction bitis used to detect the bit(s) of interest and set up the flags in SR before a jump. Many branches have two names to reflect different usage. For example, it is clearer to use jcif the carry bit is used explicitly—after a rotation, for instance—but jhsis more appropriate after a comparison:

```
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
```

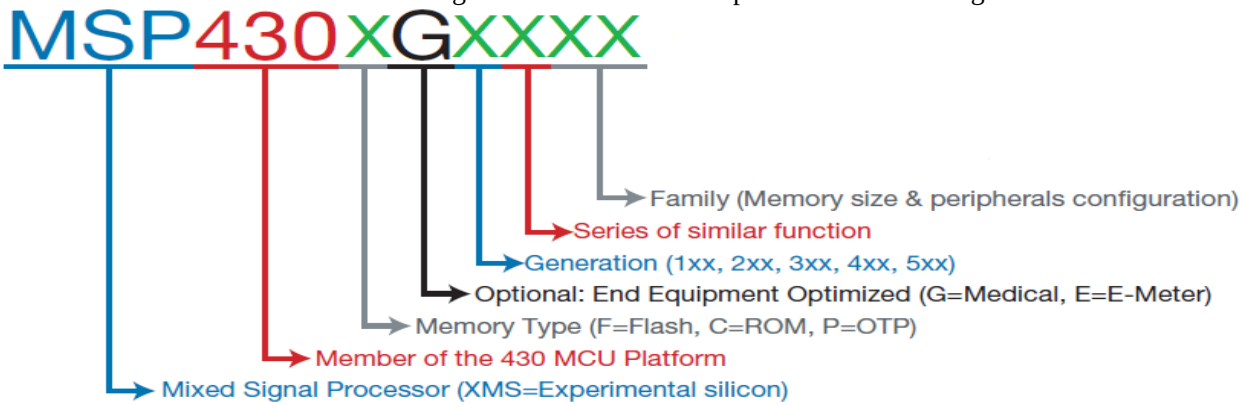
Assume that the "comparison" jumps follow cmp.w src,dst, which sets the flags according to the difference dst-src. Alternatively, tst.wdstsets the flags for dst -0:

```
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
```

(Variants of the MSP430 family viz. MSP430x2x, MSP430x4x, MSP430x5x and their targeted applications):

MSP430 Numbering:

MSP430 devices have numbering such as **MSP430xGxx** provides the following information:



Features of Various MSP430 Families

MSP430x2xx series

Power specification overview, as low as:

- 0.1 µA RAM retention
- 0.3 µA standby mode (VLO)
- 0.7 µA real-time-clock mode
- 220 µA / MIPS active
- Feature ultrafast wakeup from standby mode in less than 1 µs

Device parameters

- Flash options: 1–120 KB
- RAM options: 128 B – 8 KB
- GPIO options: 10, 11, 16, 24, 32, and 48 pins
- ADC options: Slope, 10 & 12bit SAR, 16 & 24bit Sigma Delta
- Other integrated peripherals: operational amplifiers, 12bit DAC, up to 2 16-bit timers, watchdog timer, brownout reset, SVS, USI module (I^2C, SPI), USCI module, DMA, 16×16 multiplier, Comparator_A+, temperature sensor

MSP430x4xx series

Power specification overview, as low as:

- 0.1µA RAM retention
- 0.7 µA real-time-clock mode
- 200 µA / MIPS active
- Features fast wakeup from standby mode in less than 6 µs.

Device parameters:

- Flash/ROM options: 4 – 120 KB
- RAM options: 256 B – 8 KB
- GPIO options: 14, 32, 48, 56, 68, 72, 80 pins
- ADC options: Slope, 10 & 12bit SAR, 16bit Sigma Delta
- Other integrated peripherals: SCAN_IF, ESP430, 12bit DAC, Op Amps, RTC, up to 2 16bit timers, watchdog timer, basic timer, brownout reset, SVS, USART module (UART, SPI), USCI module, LCD Controller, DMA, 16×16 & 32x32 multiplier, Comparator_A, temperature sensor, 8 MIPS CPU Speed

MSP430x5xx series

Power specification overview, as low as:

- 0.1µA RAM retention
- 2.5 µA real-time-clock mode

- 165 µA / MIPS active
- Features fast wakeup from standby mode in less than 5 µs.

Device parameters:

- Flash options: up to 512 KB
- RAM options: up to 66 KB
- ADC options: 10 & 12bit SAR
- GPIO options: 29, 31, 47, 48, 63, 67, 74, 87 pins
- Other integrated peripherals: High resolution PWM, 5 V I/O's, USB, up to 4 16-bit timers, watchdog timer, RealTime Clock, brownout reset, SVS, USCI module, DMA, 32x32 multiplier, Comp B, temperature sensor

Applications:

x2xx-consumer electronics,metering meterology, data logging applications.

x4xx-metering meterology,

x5xx-usb and infrared communications,metering meterology,consumer electronics.

Sample embedded system on MSP430 microcontroller:

An example embedded system is the weighing machine shown in Figure. It includes the following functional blocks:

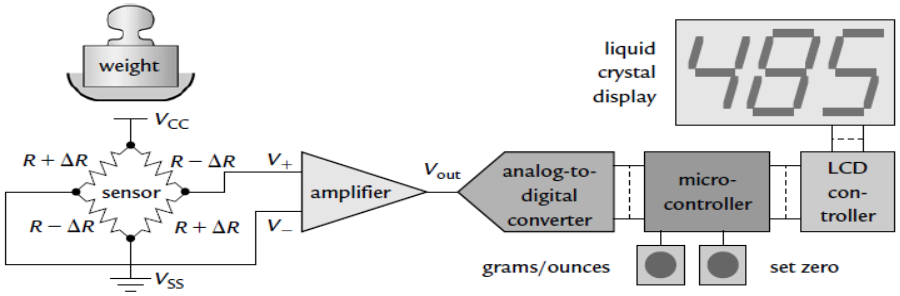
• The sensor has four resistive elements arranged as a Wheatstone bridge. Ideally, this is balanced when there is no load, giving $V+ = V-$. Two of the resistances increase and two decrease when a weight is placed on the scale pan, driving the bridge out of balance.

• A differential amplifier magnifies the difference in voltage between its input terminals, giving $Vout = A(V+-V-)$, where A is the gain.

• The analog output of the amplifier is converted to a binary value in an analog-to-digital converter.

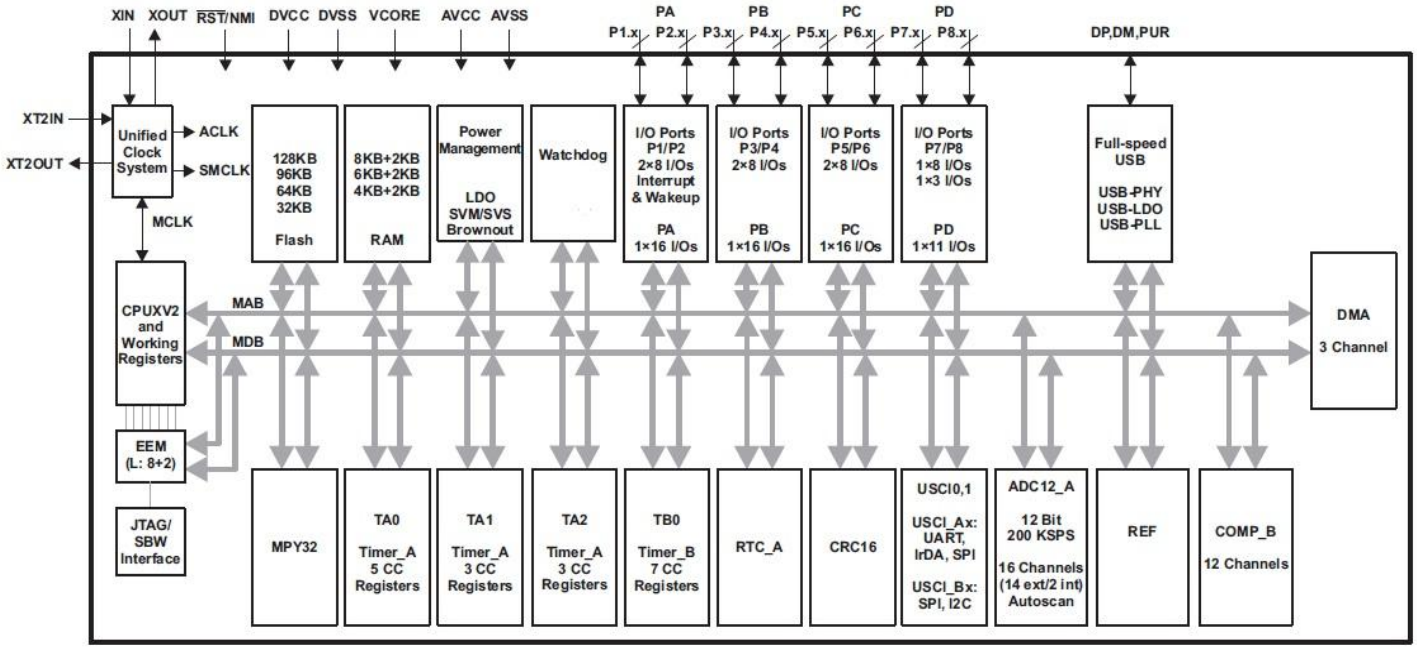
• The microcontroller multiplies the input by an appropriate factor so that the display gives the weight in grams or ounces and subtracts an offset so that the display reads zero when no weight is present. It also reads the buttons and supervises the complete system.

• There is a serial interface between the microcontroller and the liquid crystal display, which has a built-in controller.



This system clearly needs a lot of components, including several integrated circuits. In contrast, the whole system can be constructed from a sensor, an MSP430, a simple LCD without a controller, and a couple of decoupling capacitors. The MSP430x4xx family drives segmented LCDs directly, which eliminates the need for a controller. Several devices contain ADCs with high-resolution, differential inputs, which would work directly from the sensor without the need for an amplifer. The microcontroller can also manage the power drawn by the circuit so that the processor would be switched off when it was not needed and the whole system shut down after a period of inactivity.

MSP430x5xx Series Block Diagram:



CPUX

The MSP430X CPU features include:

- 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.
- 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

On–chip Peripherals(analog and digital):

Memory: These devices have flash memory, up to 512KB and RAM up to 66KB.

Digital I/O Ports:

MSP430x5xx devices may have up to 12 digital I/O ports implemented, P1 to P11 and PJ. Most ports have eight I/O pins, however some ports may contain less. Each I/O pin is individually configurable for input or output direction, and each I/O line can be individually read or written to. All ports have individually configurable pullup or pulldown resistors. Ports P1 and P2 always have interrupt capability. Port pairs P1/P2, P3/P4, P5/P6, P7/P8, etc. are associated with the names PA, PB, PC, PD, etc., respectively.

The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts
- Independent input and output data registers
- Individually configurable pullup or pulldown resistors

POWER MANAGEMENT MODULE (PMM):

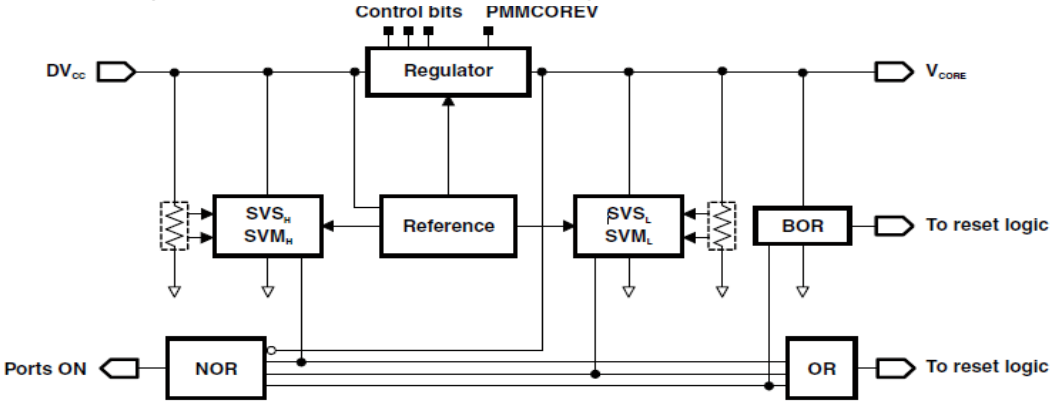
The PMM features include:

- Wide supply voltage (DV_{CC}) range: 1.8 V to 3.6 V
- Core voltage (V_{CORE}) generation: 1.4 V, 1.6 V, 1.8 V, and 1.9 V (typical)

- Brown-out-reset (BOR)
- Supply voltage supervisor(SVS) for DV_{CC} and V_{CORE}
- Supply voltage monitor(SVM) for DV_{CC} and V_{CORE} with eight programmable levels
- Software accessible power-fail conditions
- Software selectable power-on-reset at power-fail condition
- I/O protection at power-fail condition

The main digital logic of the MSP430 device requires a voltage that is lower than the range allowed by DV_{CC}. So, the PMM incorporates an integrated low-dropout voltage regulator (LDO) that generates a secondary core voltage rail, V_{CORE}.

The block diagram of the PMM is shown in below fig.



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.

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 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.

Timers (Timer_A and Timer_B)

Timer is a 16-bit timer/counter with up to seven capture/compare registers. Timer_A & B can support multiple capture/compares, PWM outputs, and interval timing. They also has extensive interrupt capabilities. Timer features include:

- Asynchronous 16-bit timer/counter with four operating modes
- Selectable and configurable clock source
- Up to seven configurable capture/compare registers
- Configurable outputs with PWM capability
- Asynchronous input and output latching
- Interrupt vector register for fast decoding of all Timer_A & B interrupts

Real-Time Clock (RTC)

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). Calendar mode integrates an internal calendar which compensates for months with less than 31 days and includes leap year correction. The RTC also supports flexible alarm functions and offset calibration hardware.

CRC16

The CRC16 module produces a signature based on a sequence of entered data values and can be used for data checking purposes.

ADC12_A (Analog to Digital Convertor)

The ADC12_A module supports fast 12-bit analog-to-digital conversions. The module implements a 12-bit SAR core, sample select control, reference generator and a 16 word conversion-and-control buffer. The

conversion-and-control buffer allows up to 16 independent ADC samples to be converted and stored without any CPU intervention.

DMA Controller

The DMA controller allows movement of data from one memory address to another without CPU intervention. For example, the DMA controller can be used to move data from the ADC12_A conversion memory to RAM. Using the DMA controller can increase the throughput of peripheral modules. The DMA controller reduces system power consumption by allowing the CPU to remain in sleep mode, without having to awaken to move data to or from a peripheral.

Universal Serial Communication Interface (USCI)

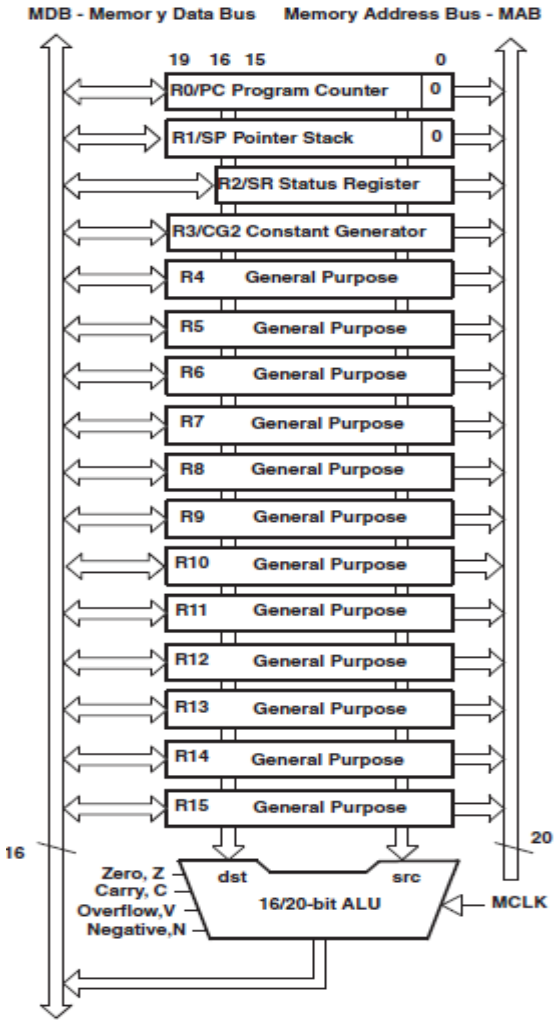
The USCI modules are used for serial data communication. The USCI module supports synchronous communication protocols such as SPI (3-pin or 4-pin) and I2C, and asynchronous communication protocols such as UART, enhanced UART with automatic baudrate detection, and IrDA.

Universal Serial Bus (USB)

The features of the USB module include:

- Fully compliant with the USB 2.0 full-speed specification
  - – Full-speed device (12 Mbps) with integrated USB transceiver (PHY)
  - – Supports control, interrupt, and bulk transfers
  - – Supports USB suspend, resume, and remote wakeup
- A power supply system independent from the PMM system
  - – Integrated 3.3-V LDO regulator with sufficient output to power entire MSP430 and system circuitry from 5-V V_{BUS}

Register Organization of MSP 430X5XX(Register Sets):



The registers in the CPU of the MSP430X can be used for either data or addresses and have therefore

been enlarged to 20 bits as well. The program counter and stack pointer are used only as addresses and are therefore always treated as 20-bit registers. In contrast, the status register has only 16 bits. The constant generator and general-purpose registers can handle 8, 16, or 20-bit numbers.The general functions of the registers are unchanged from the MSP430. For example, the stack pointer should be initialized to the top of RAM before any functions are called.

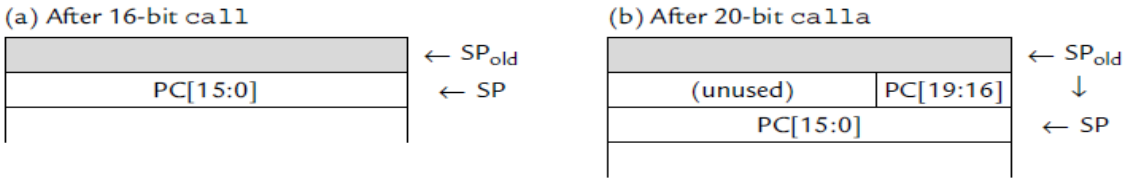
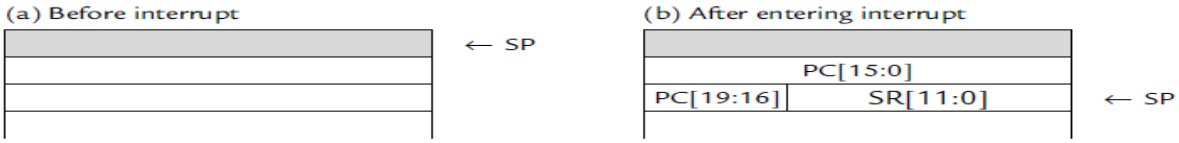


Figure above shows that two 16-bit words are needed to store a 20-bit address on the stack and the same is true for main memory: Address words require 4 bytes of storage, 12 bits of which are unused. This wastes memory and reduces the effective speed of the processor because two cycles are needed to fetch a complete 20-bit address from memory.

Interrupts are handled in a slightly different way from subroutines. Both the status register and program counter are stacked in this case. The upper 7 bits of the status register are not used and the MSP430X therefore takes over the space for the top 4 of them to hold the extra 4 bits of the program counter. It is shown in below figure.



The above implementation saves both memory and time, which is particularly important for interrupts. The '*reti*' instruction has been updated to match. An interrupt service routine must be situated in the lowest 64KB of the address space because vectors hold only 16-bit addresses but the full 20-bit address is stacked to ensure that execution can resume at any address after the interrupt.

Address space:

The MSP430x5xx family has a 1-MB unified memory map with expanded peripheral space over previous families.4KB of memory from 00000h to 00FFFH is reserved for the peripherals. 2KB of memory from 01000h to 017FFH is reserved for the boot memory to store the programs which are required to initialize the device. 512 Bytes of memory from 01800h to 019FFH is reserved for information memory for storing the calibration data , temporary data, intermediate data. Information memory is divided into 4 parts of each 128 Bytes named as infoA, infoB, infoC and infoD. A small amount of RAM memory 16KB is reserved for processing the data.256KB of memory is reserved as code memory for storing the code of the programs.MSP430 has large interrupt capability whose address range is 0FF80H to 0FFFFH i.e., 128 Bytes.
The detailed information of address space is shown in the table below.

Segment	Address	Size
Peripherals	00000h – 00FFFh	4KB
Boot Memory	01000h – 017FFh	2KB
Information memory	01800h – 019FFh	512 Bytes
RAM	01C00h – 05BFFh	16KB
Code memory	05C00h – 45BFFh	256KB
Interrupt vectors	0FF80h – 0FFFFh	128 Bytes