Unit 1

Module: Architecture – CPU

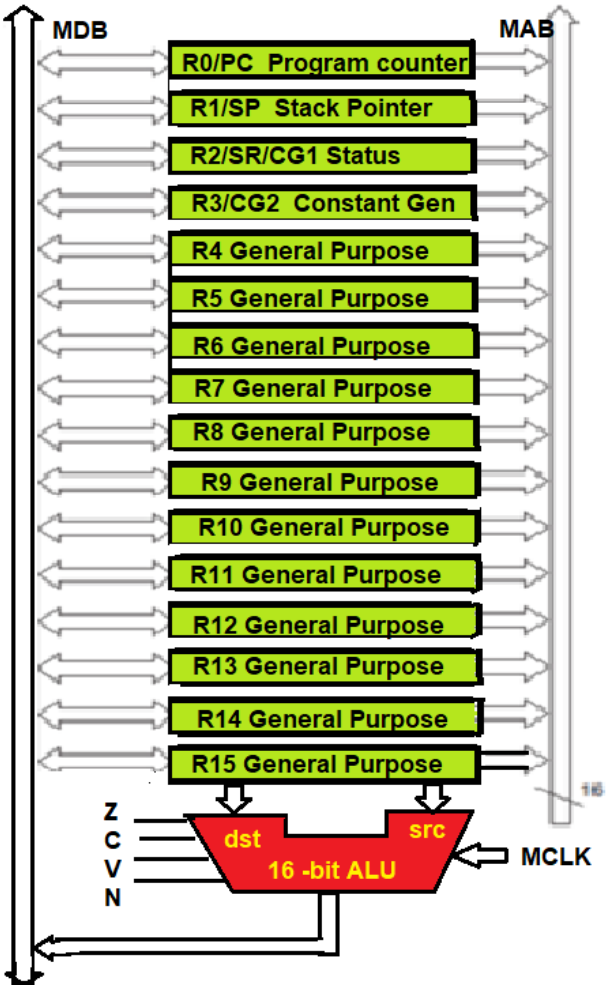
MSP 430 CPU – Introduction

The MSP430 is an ultra-low-power mixed signal microcontroller with a built-in 16-bit timer and ten I/O pins. In addition, it has built-in communication capability using synchronous protocols (UART, SPI or I2C) and a 10/16-bit sigma-delta A/D converter.

The architecture, combined with five low-power modes is optimized to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 1 µs.

The MSP430 has In-System Programmable (ISP) flash memory of up to 256 Kb.

16 – BIT RISC CPU



- Efficient, ultra-low power CPU
- C-compiler friendly
- RISC architecture – 27 core instructions

- 24 emulated instructions
- 7 addressing modes
- Constant generator
- Single-cycle register operations
- Memory-to-memory atomic addressing
- Bit, byte and word processing
- 20-bit addressing on MSP430X for Flash>64KB

CPU Registers – Program Counter (PC)

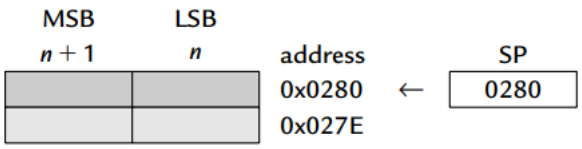
The 16-bit program counter (PC/R0) points to the next instruction to be executed. Each instruction uses an even number of bytes (two, four, or six), and the PC is incremented accordingly. Instruction accesses in the 64-KB address space are performed on word boundaries, and the PC is aligned to even addresses.

CPU Registers - Stack Pointer (SP)

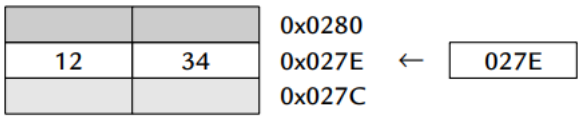
The MSP430 uses R1 as a pointer to the stack in RAM (the stack must reside in RAM). The stack pointer (SP/R1) is used by the CPU to store the return addresses of subroutine calls and interrupts. The SP is initialized into RAM by the user and is aligned to even addresses. At any given time, the stack pointer points to the last value pushed (placed) on the stack. Values are pushed as 16 bit words, and after each push, the stack pointer is decremented by 2.

PUSH and POP operations of Stack

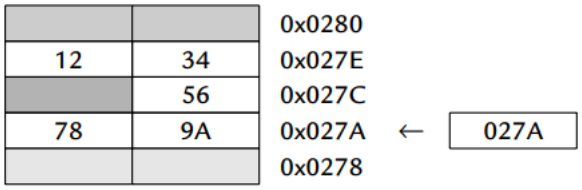
(a) Stack after initialization.



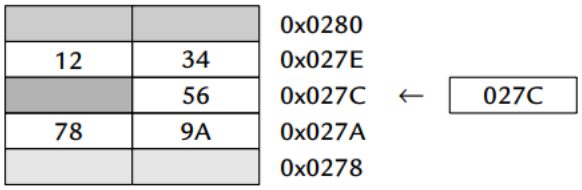
(b) Stack after `push.w #0x1234`.



(c) Stack after `push.b #0x56` followed by `push.w #0x789A`.



(d) Stack after `pop.w R15`.



R2: Status Register (CG1 + SR)

Status Resister (SR)

- The Status Register (SR/R2) 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 of the SR are used to support the constants generator.

15 14 13 12 11 10 9	8	7	6	5	4	3	2	1	0
Reserved for CG1	V	SCG1	SCG0	OSCOFF	CPUOFF	GIE	N	Z	C

Bit		Description
8	V	Overflow bit. V = 1 ⇒ Result of an arithmetic operation overflows the signed-variable range.
7	SCG1	System clock generator 0. SCG1 = 1 ⇒ DCO generator is turned off – if not used for MCLK or SMCLK.
6	SCG0	System clock generator 1. SCG0 = 1 ⇒ FLL+ loop control is turned off.
5	OSCOFF	Oscillator Off. OSCOFF = 1 ⇒ turns off LFXT1 when it is not used for MCLK or SMCLK.
4	CPUOFF	CPU off. CPUOFF = 1 ⇒ disable CPU core.
3	GIE	General interrupt enable. GIE = 1 ⇒ enables maskable interrupts.
2	N	Negative flag. N = 1 ⇒ result of a byte or word operation is negative.
1	Z	Zero flag. Z = 1 ⇒ result of a byte or word operation is 0.
0	C	Carry flag. C = 1 ⇒ result of a byte or word operation produced a carry.

R2/R3: Constant Generator (CG1/CG2)

Depending on the source-register addressing modes (As) value, six commonly used constants can be generated without a code word or code memory access to retrieve them. This is a very powerful feature, which allows the implementation of emulated instructions, for example, instead of implementing a core instruction for an increment, the constant generator is used.

Register	As	Constant	Remarks
R2	00	-	Register mode
R2	01	(0)	Absolute mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	-1, word processing

Depending on the source-register addressing modes (As) value, six commonly used constants can be generated without a code word or code memory access to retrieve them. This is a very

powerful feature, which allows the implementation of emulated instructions, for example, instead of implementing a core instruction for an increment, the constant generator is used.

R4 - R15: General Purpose Registers

These general-purpose registers are used to store data values, address pointers, or index values and can be accessed with byte or word instructions.