Chapter 4

MSP430 Architecture

The objective of this chapter is to provide a comprehensive description of the MSP430 architecture, covering its main characteristics: address space, the Central Processing Unit (CPU), the seven addressing modes and the instruction set. A review is made of the MSP430 assembly language instruction set composed of 27 base op-codes and 24 emulated instructions. It also discusses the techniques used to program in assembly language and the basics of writing an assembly-language program.

Topic **Page**

4.1 Introduction

The types of devices such as microprocessor, microcontroller, processor, digital signal processor (DSP), amongst others, in a certain manner, are related to the same device – the ASIC (Application Specific Integrated Circuit). Each processing device executes instructions, following a determined program applied to the inputs and shares architectural characteristics developed from the first microprocessors created in 1971. In the three decades after the development of the first microprocessor, huge developments and innovations have been made in this engineering field. Any of the terms used at the beginning of this section are correct to define a microprocessor, although each one has different characteristics and applications.

The definition of a microcontroller is somewhat difficult due to the constantly changing nature of the silicon industry. What we today consider a microcontroller with medium capabilities is several orders of magnitude more powerful, than the computer used on the first space missions. Nevertheless, some generalizations can be made as to what characterizes a microcontroller. Typically, microcontrollers are selected for embedded systems projects, i.e., control systems with a limited number of inputs and outputs where the controller is *embedded* into the system.

The programmable SoC (system-on-chip) concept started in 1972 with the 4-bit TMS1000 microcomputer developed by Texas Instruments (TI), and in those days it was ideal for applications such as calculators and ovens. This term was changed to Microcontroller Unit (MCU), which was more descriptive of a typical application. Nowadays, MCUs are at the heart of many physical systems, with higher levels of integration and processing power at lower power consumption.

The following list presents several qualities that define a microcontroller:

- Cost: Usually, the microcontrollers are high-volume, low cost devices;
- Clock frequency: Compared with other devices (microprocessors and DSPs), microcontrollers use a low clock frequency. Microcontrollers today can run up to 100 MHz/ 100 Million Instructions Per Second (MIPS)
- Power consumption: orders of magnitude lower than their DSP and MPU cousins;
- Bits: 4 bits (older devices) to 32 bits devices;
- Memory: Limited available memory, usually less than 1 MByte;
- Input/Output (I/O): Low to high (8-150) pin-out count.

Figure 4-1 shows a microcontroller that meets the above criteria.

Figure 4-1. Microcontroller Texas Instruments MSP430F169.

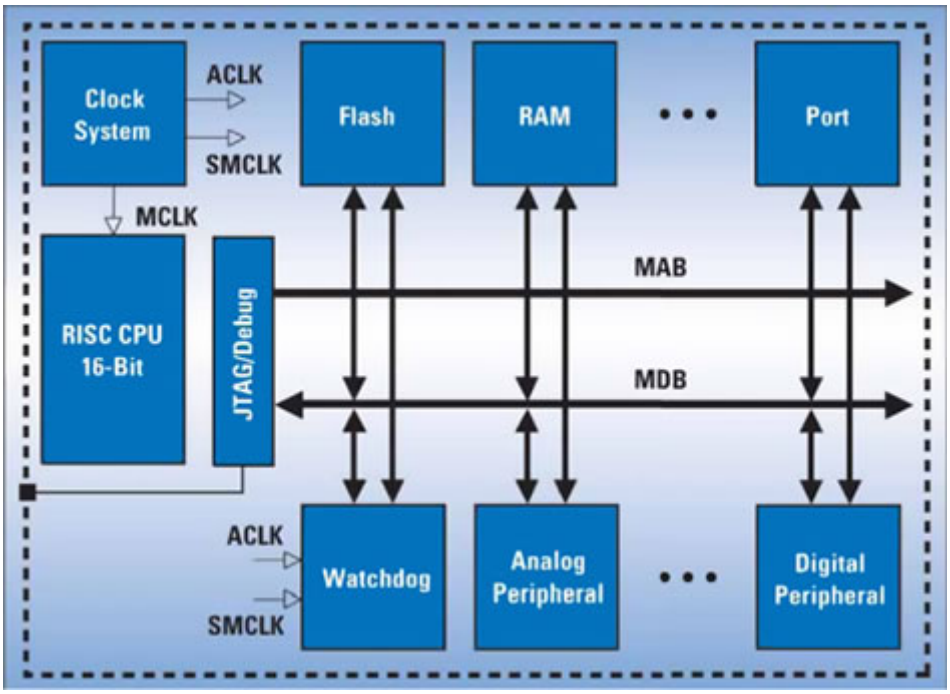
4.2 Main characteristics of a MSP430 microcontroller

Although there are variants in devices in the family, a MSP430 microcontroller can be characterized by:

- Low power consumption:
  - 0.1 μA for RAM data retention;
  - 0.8 μA for real time clock mode operation;
  - 250 μA/MIPS at active operation.
- Low operation voltage (from 1.8 V to 3.6 V).
- < 1 μs clock start-up.
- < 50 nA port leakage.
- Zero-power Brown-Out Reset (BOR).
- On-chip analogue devices:
  - 10/12/16-bit Analogue-to-Digital Converter (ADC);
  - 12-bit dual Digital-to-Analogue Converter (DAC);
  - Comparator-gated timers;
  - Operational Amplifiers (OP Amps);
  - Supply Voltage Supervisor (SVS).
- 16 bit RISC CPU:
  - Instructions processing on either bits, bytes or words;
  - Compact core design reduces power consumption and cost;
  - Compiler efficient;
  - 27 core instructions;
  - 7 addressing modes;
  - Extensive vectored-interrupt capability.
- Flexibility:
  - Up to 256 kB In-System Programmable (ISP) Flash;
  - Up to 100 pin options;
  - USART, I2C, Timers;
  - LCD driver;
  - Embedded emulation.

The microcontroller's performance is directly related to the 16-bit data bus, the 7 addressing modes and the reduced instructions set, which allows a shorter, denser programming code for fast execution. These microcontroller families share a 16-bit CPU (Central Processing Unit) core, RISC[1] type, intelligent peripherals, and flexible clock system that interconnects using a Von Neumann[2] common memory address bus (MAB) and memory data bus (MDB) architecture.

Figure 4-2. MSP430 architecture.



4.3 Address space

All memory, including RAM, Flash/ROM, information memory, special function registers (SFRs), and peripheral registers are mapped into a single, contiguous address space as shown in *Figure 4–3*.

Note: See the device-specific datasheets for specific memory maps. Code access is always performed on even addresses. Data can be accessed as bytes or words.

1 RISC (Reduced Instructions Set Computing) – In this type of configuration, the instructions are reduced to the basic ones, with the objective of providing simpler and faster instruction decoding. The MSP430 only uses 27 physical instructions and 24 emulated instructions.

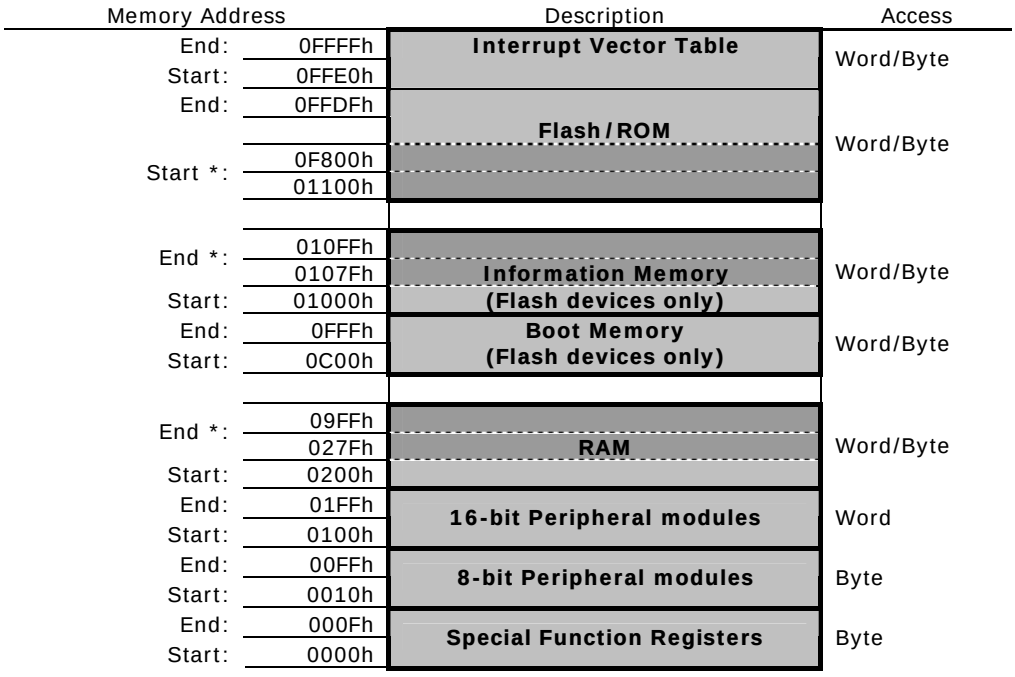
2 Von Neumann architecture - Computational architecture that makes use of only one storage structure to save the data and instructions sets. In the model, the separation of the processing unit storage is implicit. Since the instructions are treated as data, the devices that use this type of architecture can easily modify the instruction, i.e., are programmable.

The MSP430 is available with either Flash or ROM memory types. The memory type is identified by the letter immediately following “MSP430” in the part numbers.

Flash devices: Identified by the letter “F” in the part numbers, having the advantage that the code space can be erased and reprogrammed.

ROM devices: Identified by the letter “C” in the part numbers. They have the advantage of being very inexpensive because they are shipped pre-programmed, which is the best solution for high-volume designs.

Figure 4-3. Memory Map.



* Depending on device family.

For all devices, each memory location is formed by 1 data byte. The CPU is capable of addressing data values either as bytes (8 bits) or words (16 bits). Words are always addressed at an even address, which contain the least significant byte, followed by the next odd address, which contains the most significant byte. For 8-bit operations, the data can be accessed from either odd or even addresses, but for 16-bit operations, the data values can only be accessed from even addresses.

4.3.1 Interrupt vector table

The interrupt vector table is mapped at the very end of memory space (upper 16 words of Flash/ROM), in locations 0FFE0h through to 0FFFEh (see the device-specific datasheets). The priority of the interrupt vector increases with the word address.

Table 4-1. Interrupt vector table for MSP430 families.

Vector Address	Priority	'11xx and '12xx	'13x and '14x	'2xx	'3xx	'4xx
0xFFFE	31, Highest	Hard Reset/ Watchdog	Hard Reset/ Watchdog	Hard Reset/ Watchdog	Hard Reset/ Watchdog	Hard Reset/ Watchdog
0xFFFC	30	Oscillator/ Flash/NMI	Oscillator/ Flash/NMI	Oscillator/ Flash/NMI	Oscillator/ Flash/NMI	Oscillator/ Flash/NMI
0xFFFA	29	Unused	Timer_B	Timer_B (22x2, 22x4, 23x, 24x, 26x only)	Dedicated I/O	Timer_B ('43x and'44x only)
0xFFF8	28	Unused	Timer_B	Timer_B (22x2, 22x4, 23x, 24x only)	Dedicated I/O	Timer_B ('43x and'44x only)
0xFFF6	27	Comparator	Comparator	Comparator_A+ (20x1 only, 21x1, 23x, 24x, 26x)	Unused	Comparator
0xFFF4	26	Watchdog Timer	Watchdog Timer	Watchdog Timer+	Watchdog Timer	Watchdog Timer
0xFFF2	25	Timer_A	USART Rx	Timer_A	Timer_A	USART0 Rx ('43x and'44x only)
0xFFF0	24	Timer_A	USART0 Tx	Timer_A	Timer_A	USART0 Tx ('43x and'44x only)
0xFFEE	23	USART0 Rx ('12xx only)	ADC	USCI Rx (22x2, 22x4, 23x, 24x, 26x only) I2C status (23x, 24x)	USART Rx	ADC ('43x and'44x only)
0xFFEC	22	USART0 Tx ('12xx only)	Timer_A	USCI Tx (22x2, 22x4, 23x, 24x, 26x only) I2C Rx/Tx (23x, 24x, 26x only)	USART Tx	Timer_A
0xFFEA	21	ADC10	Timer_A	ADC10 (20x2 22x2, 22x4 only) ADC12 (23x, 24x, 26x only) SD16_A (20x3 only)	ADC('32x and '33x) Timer/Port ('31x)	Timer_A
0xFFE8	20	Unused	Port 1	USI (20x2, 20x3 only)	Timer/Port ('32x and '33x)	Port 1
0xFFE6	19	Port 2	USART1 Rx	Port P2	Port 2	USART1 Rx ('44x only)
0xFFE4	18	Port 1	USART1 Tx	Port P1	Port 1	USART1 Tx ('44x only)
0xFFE2	17	Unused	Port 2	USCI Rx (23x, 24x, 26x only) I2C status (241x, 261x only)	Basic Timer	Port 2
0xFFE0	16	Unused	Unused	USCI Tx (23x,24x only) I2C Rx/Tx (241x, 261x only)	Port 0	Basic Timer
	15			DMA (241x, 261x only)		
	14			DAC12 (241x, 261 only)		
	13 to 0 Lowest			Reserved		

4.3.2 Flash / ROM

The start address of Flash/ROM depends on the amount of Flash/ROM present on the device. The start address varies between 01100h (60k devices) to 0F800h (2k devices) and always runs to the end of the address space at location 0FFFFh. Flash can be used for both code and data. Word or byte tables can also be stored and read by the program from Flash/ROM.

All code, tables, and hard-coded constants reside in this memory space.

4.3.3 Information memory (*Flash devices only*)

The MSP430 flash devices contain an address space for information memory. It is like an onboard EEPROM, where variables needed for the next power up can be stored during power down. It can also be used as code memory. Flash memory may be written one byte or word at a time, but must be erased in segments. The information memory is divided into two 128-byte segments. The first of these segments is located at addresses 01000h through to 0107Fh (Segment B), and the second is at address 01080h through to 010FFh (Segment A). This is the case in 4xx devices. It is 256 bytes (4 segments of 64 bytes each) in 2xx devices.

4.3.4 Boot memory (*Flash devices only*)

The MSP430 flash devices contain an address space for boot memory, located between addresses 0C00h through to 0FFFh. The “bootstrap loader” is located in this memory space, which is an external interface that can be used to program the flash memory in addition to the JTAG. This memory region is not accessible by other applications, so it cannot be overwritten accidentally. The bootstrap loader performs some of the same functions as the JTAG interface (excepting the security fuse programming), using the TI data structure protocol for UART communication at a fixed data rate of 9600 baud.

4.3.5 RAM

RAM always starts at address 0200h. The end address of RAM depends on the amount of RAM present on the device. RAM is used for both code and data.

4.3.6 Peripheral Modules

Peripheral modules consist of all on-chip peripheral registers that are mapped into the address space. These modules can be accessed with byte or word instructions, depending if the peripheral module is 8-bit or 16-bit respectively. The 16-bit peripheral modules are located in the address space from addresses 0100 through to 01FFh and the 8-bit peripheral modules are mapped into memory from addresses 0010h through to 00FFh.

4.3.7 Special Function Registers (SFRs)

Some peripheral functions are mapped into memory with special dedicated functions. The Special Function Registers (SFRs) are located at memory addresses from 0000h to 000Fh, and are the specific registers for:

- Interrupt enables (locations 0000h and 0001h);
- Interrupt flags (locations 0002h and 0003h);
- Enable flags (locations 0004h and 0005h);

SFRs must be accessed using byte instructions only. See the device-specific data sheets for the applicable SFR bits.

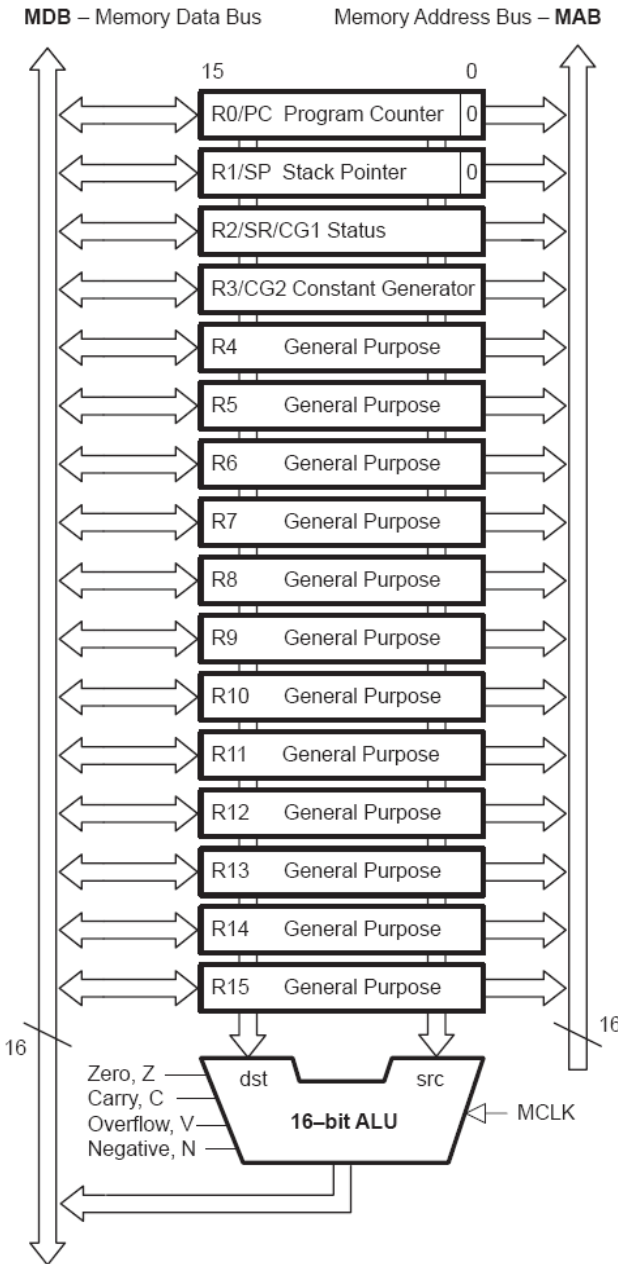
4.4 Central Processing Unit (MSP430 CPU)

The RISC type architecture of the CPU is based on a short instruction set (27 instructions), interconnected by a 3-stage instruction pipeline for instruction decoding. The CPU has a 16-bit ALU, four dedicated registers and twelve working registers, which makes the MSP430 a high performance microcontroller suitable for low power applications. The addition of twelve working general purpose registers saves CPU cycles by allowing the storage of frequently used values and variables instead of using RAM.

The orthogonal instruction set allows the use of any addressing mode for any instruction, which makes programming clear and consistent, with few exceptions, increasing the compiler efficiency for high-level languages such as C.

Note: MSP430 Assembly programming language topics are discussed in Chapter 15.

Figure 4-4. MSP430 CPU block diagram.



4.4.1 Arithmetic Logic Unit (ALU)

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

4.4.2 MSP430 CPU registers

The CPU incorporates sixteen 16-bit registers:

- Four registers (R0, R1, R2 and R3) have dedicated functions;
- There are 12 working registers (R4 to R15) for general use.

R0: Program Counter (PC)

The 16-bit Program Counter (PC/R0) points to the next instruction to be read from memory and executed by the CPU. The Program counter is implemented by the number of bytes used by the instruction (2, 4, or 6 bytes, always even). It is important to remember that the PC is aligned at even addresses, because the instructions are 16 bits, even though the individual memory addresses contain 8-bit values.

R1: Stack Pointer (SP)

The Stack Pointer (SP/R1) is located in R1.

1st: stack can be used by user to store data for later use (instructions: store by PUSH, retrieve by POP);

2nd: stack can be used by user or by compiler for subroutine parameters (PUSH, POP in calling routine; addressed via offset calculation on stack pointer (SP) in called subroutine);

3rd: used by subroutine calls to store the program counter value for return at subroutine's end (RET);

4th: 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.

R2: Status Register (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. See the device-specific data sheets for more details.

SR

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 Registers (CG1/CG2)

Depending of 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.

Table 4-2. Values of the constant generator registers.

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

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.

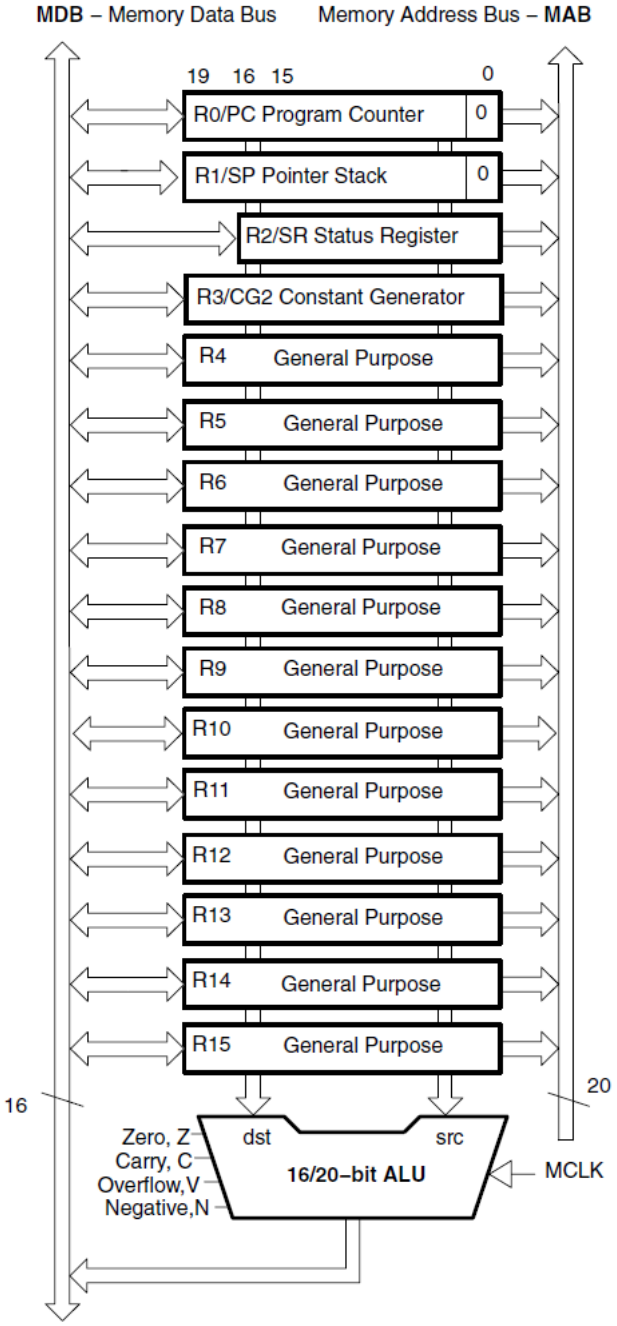
4.5 Central Processing Unit (MSP430X CPU)

4.5.1 Main features of the MSP430X CPU architecture

The MSP430X CPU extends the addressing capabilities of the MSP430 family beyond 64 kB to 1 MB. To achieve this, there are some changes to the addressing modes and two new types of instructions. One type of new instructions allows access to the entire address space, and the other is designed for address calculations.

The MSP430X CPU address bus is 20 bits, but the data bus is still 16 bits. The CPU supports 8-bit, 16-bit and 20-bit memory accesses. Despite these changes, the MSP430X CPU remains compatible with the MSP430 CPU, having a similar number of registers. A block diagram of the MSP430X CPU is shown in the figure below:

Figure 4-5. MSP430X CPU block diagram.



Although the MSP430X CPU structure is similar to that of the MSP430 CPU, there are some differences that will now be discussed.

With the exception of the status register SR, all MSP430X registers are 20 bits. The CPU can now process 20-bit or 16-bit data.

4.5.2 MSP430X CPU registers

❑ R0 (PC) - Program Counter

Has the same function as the MSP430 CPU, although now it has 20 bits.

❑ R1 (SP) - Stack Pointer

Has the same function as the MSP430 CPU, although now it has 20 bits.

❑ R2 (SR) - Status Register

Has the same function as the MSP430 CPU, but still only has 16 bits.

Table 4-3. Description of the SR bits.

Bit	Description	
Reserved	Reserved	
V	Overflow bit. This bit is set when the result of an arithmetic operation overflows the signed-variable range.	
	`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	System clock generator 1. This bit, when set, turns off the DCO dc generator if DCOCLK is not used for MCLK or SMCLK.	
SCG0	System clock generator 0. This bit, when set, turns off the FLL+ loop control.	
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 bit. This bit is set when the result of an operation is negative and cleared when the result is positive.	
Z	Zero bit. This bit is set when the result of an operation is zero and cleared when the result is not zero.	
C	Carry bit. This bit is set when the result of an operation produced a carry and cleared when no carry occurred.	

❑ R2 (CG1) and R3 (CG2) - Constant Generators

The registers R2 and R3 can be used to generate six different constants commonly used in programming, without the need to add an extra 16-bit word of code to the instruction. The constants below are chosen based on the bit (As) of the instruction that selects the addressing mode.

Table 4-4. Values of constant generators.

Register	As	Constant	Remarks
R2	00	-	Register mode
R2	01	(0)	Absolute address 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	FFh, FFFFh, FFFFFh	-1, word processing

Whenever the operand is one of these six constants, the registers are selected automatically. Therefore, when used in constant mode, registers R2 and R3 cannot be addressed explicitly by acting as source registers.

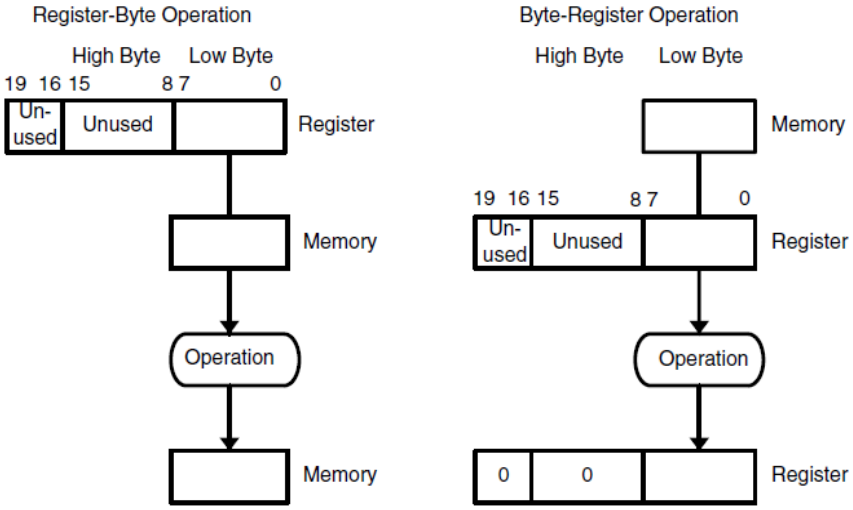
❑ R4-R15 – General-purpose registers

These registers have the same function as the MSP430 CPU, although they now have 20 bits. They can store 8-bit, 16-bit or 20-bit data. Any byte written to one of these registers clears bits 19:8. Any word written to one of these registers clears bits 19:16. The exception to this rule is the instruction SXT, which extends the sign value to fill the 20-bit register.

The following figures illustrate how the operations are conducted for the exchange of information between memory and registers, for the following formats: byte (8 bits), word (16 bits) and address (20 bits).

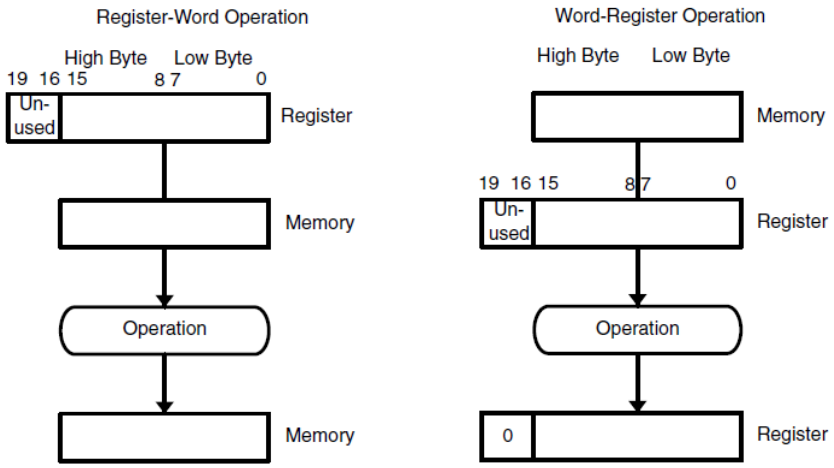
The following figure illustrates the handling of a byte (8 bits) using the suffix .B.

Figure 4-6. Example: Register-Byte/Byte-Register operation.



The following figure illustrates the handling of a word (16-bit) using the suffix `.W`.

Figure 4-7. Example: Register-Word/Word-Register operation.



The following figure illustrates the manipulation of an address (20 bits) using the suffix `.A`.

Figure 4-8. Example: Register - Address-Word operation.

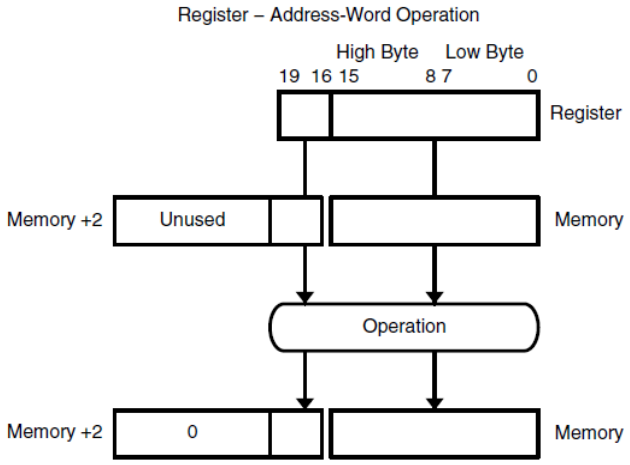
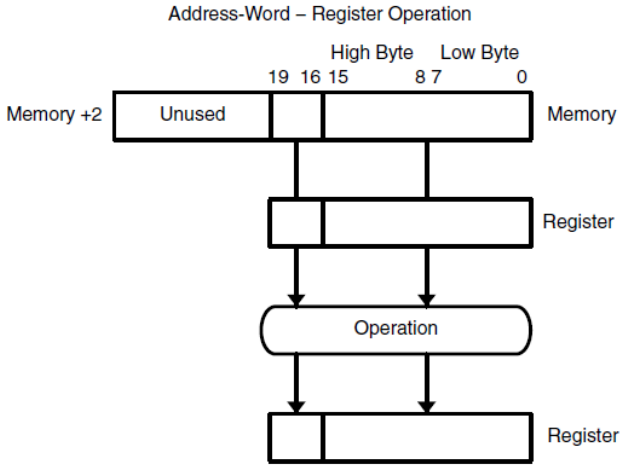


Figure 4-9. Example: Address-Word - Register operation.



All other differences in the addressing modes, instruction set, and other details for the CPUX architecture present in MSP430 devices which have over 64 kB of on chip memory are described in much greater depth is *section 15.3.2*.

4.6 Addressing modes

The MSP430 supports seven addressing modes for the source operand and four addressing modes for the destination operand (see *Table 4-5*). The following sections describe each of the addressing modes, with a brief description, an example and the number of CPU clock cycles required for an instruction, depending on the instruction format and the addressing modes used.

Note: Additional information concerning the MSP430 addressing modes can be found in Chapter 15.

Table 4-5. Source and destination operands, addressing modes.

Mode	Source operand	Destination operand	Description
Register mode	X	X	Single cycle
Indexed mode	X	X	Table processing
Symbolic mode	X	X	Easy to read code, PC relative
Absolute mode	X	X	Directly access any memory location
Indirect register mode	X		Access memory with pointers
Indirect auto increment mode	X		Table processing
Immediate mode	X		Unrestricted constant values

Before describing the addressing modes, it is important to mention the clock cycles required by interrupts and resets.

Table 4-6. Cycles required performing an interrupt or a reset.

Action	Cycles	Length (words)
Return from interrupt	5	1
Interrupt accepted	6	-
Watchdog timer reset	4	-
Hard reset	4	-

4.6.1 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.

Description: Register contents are operands.

Source mode bits: As = 00 (source register defined in the opcode).

Destination mode bit: Ad=0 (destination register defined in the opcode).

Syntax: Rn.

Length: One or two words.

Comment: Valid for source and destination.

Example 1: Move (copy) the contents of source (register R4) to destination (register R5). Register R4 is not affected.

Before operation:	R4=A002h	R5=F50Ah	PC = PC_{pos}
Operation:	MOV R4, R5		
After operation:	R4=A002h	R5=A002h	PC = PC_{pos} + 2

The first operand is in register mode and depending on the second operand mode, the cycles required to complete an instruction will differ. *Table 4-7* shows the cycles required to complete an instruction, depending on the second operand mode.

Table 4-7. Cycles required to perform the instruction in the first operand, in register mode.

Operands	2nd operand mode	Operator	Cycles	Length (words)
2	Register	Any	1*	1
2	Indexed, Symbolic or Absolute	Any	4	2
1	N/A	RRA, RRC, SWPB or SXT	1	1
1	N/A	PUSH	3	1
1	N/A	CALL	4	1

*Register mode instructions where the destination is the Program Counter (PC) require 2 cycles instead of 1.

4.6.2 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.

Description: (Rn + X) points to the operand. X is stored in the next word.

Source mode bits: As = 01 (memory location is defined by the word immediately following the opcode).

Destination mode bit: Ad=1 (memory location is defined by the word immediately following the opcode).

Syntax: X(Rn).

Length: Two or three words.

Comment: Valid for source and destination.

Example 2: Move (copy) the contents at source address (F000h + R5) to destination (register R4).

```
Before operation:   R4=A002h    R5=050Ah    Loc:0xF50A=0123h
Operation:          MOV F000h(R5), R4
After operation:    R4=0123h    R5=050Ah    Loc:0xF50A=0123h
```

Table 4-8. Cycles required to perform an instruction contained in the first operand, using indexed mode.

Operands	2nd operand mode	Operator	Cycles	Length (words)
2	Register	Any	3	2
2	Indexed, Symbolic or Absolute	Any	6	3
1	N/A	RRA, RRC, SWPB or SXT	4	2
1	N/A	CALL or PUSH	5	2

4.6.3 Symbolic mode

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.

Description: (PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.

Source mode bits: As = 01 (memory location is defined by the word immediately following the opcode).

Destination mode bit: Ad=1 (memory location is defined by the word immediately following the opcode).

Syntax: ADDR.

Length: Two or three words.

Comment: Valid for source and destination.

Example 3: Move the content of source address XPT (x pointer) to the destination address YPT (y pointer).

```
Before operation:   XPT=A002h   Location YPT=050Ah
Operation:          MOV XPT, YPT
After operation:    XPT= A002h  Location YPT=A002h
```

Table 4-9. Cycles required to perform an instruction contained in the first operand, in symbolic mode.

Operands	2nd operand mode	Operator	Cycles	Length (words)
2	Register	Any	3	2
2	Indexed, Symbolic or Absolute	Any	6	3
1	N/A	RRA, RRC, SWPB or SXT	4	2
1	N/A	CALL or PUSH	5	2

4.6.4 Absolute mode

Similar to Symbolic mode, with the difference that the label is preceded by "&".

Description: The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.

Source mode bits: As = 01 (memory location is defined by the word immediately following the opcode).

Destination mode bit: Ad=1 (memory location is defined by the word immediately following the opcode).

Syntax: &ADDR.

Length: Two or three words.

Comment: Valid for source and destination.

Example 4: Move the content of source address XPT to the destination address YPT.

```
Before operation:   Location XPT=A002h   Location YPT=050Ah
Operation:          MOV &XPT, &YPT
After operation:    Location XPT= A002h   Location YPT=A002h
```

Table 4-10. Cycles required to perform an instruction contained in the first operand in absolute mode.

Operands	2nd operand mode	Operator	Cycles	Length (words)
2	Register	Any	3	2
2	Indexed, Symbolic or Absolute	Any	6	3
1	N/A	RRA, RRC, SWPB or SXT	4	2
1	N/A	CALL or PUSH	5	2

4.6.5 Indirect register mode

The data word addressed is located in the memory location pointed to by Rn. Indirect mode is not valid for destination operands, but can be emulated with the indexed mode format 0(Rn).

Description: Rn is used as a pointer to the operand.

Source mode bits: As = 10.

Syntax: @Rn.

Length: One or two words.

Comment: Valid only for source operand. The substitute for destination operand is 0(Rn).

Example 5: Move the contents of the source address (contents of R4) to the destination (register R5). Register R4 is not modified.

```
Before operation:   R4=A002h    R5=050Ah    Loc:0xA002=0123h
Operation:          MOV @(R4), R5
After operation:    R4= A002h   R5=0123h    Loc:0xA002=0123h
```

Table 4-11. Cycles required to perform an instruction contained in the first operand, in indirect register mode.

Operands	2nd operand mode	Operator	Cycles	Length (words)
2	Register	Any	2*	1
2	Indexed, Symbolic or Absolute	Any	5	2
1	N/A	RRA, RRC, SWPB or SXT	3	1
1	N/A	CALL or PUSH	4	1

*Indirect register mode instructions where destination is Program Counter (PC) require 3 cycles.

4.6.6 Indirect auto increment mode

Similar to indirect register mode, but with indirect auto increment mode, the operand is incremented as part of the instruction. The format for operands is @Rn+. This is useful for working on blocks of data.

Description: Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for byte instructions and by 2 for word instructions.

Source mode bits: As = 11.

Syntax: @Rn+.

Length: One or two words.

Comment: Valid only for source operand. The substitute for destination operand is 0(Rn) plus second instruction INCD Rn.

Example 6: Move the contents of the source address (contents of R4) to the destination (register R5), then increment the value in register R4 to point to the next word.

```
Before operation:   R4=A002h    R5=050Ah    Loc:0xA002=0123h
Operation:          MOV @R4+, R5
After operation:    R4= A004h   R5=0123h    Loc:0xA002=0123h
```

Table 4-12. Cycles required to perform an instruction contained in the first operand, in indirect auto increment mode.

Operands	2nd operand mode	Operator	Cycles	Length (words)
2	Register	Any	2*	1
2	Indexed, Symbolic or Absolute	Any	5	2
1	N/A	RRA, RRC, SWPB or SXT	3	1
1	N/A	PUSH	4	1
1	N/A	CALL	5	1

*Indirect autoincrement mode instructions where destination is Program Counter (PC) require 3 cycles.

4.6.7 Immediate mode

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

Description: The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

Source mode bits: As = 11.

Syntax: #N.

Length: Two or three words. It is one word less if a constant in CG1 or CG2 can be used.

Comment: Valid only for source operand.

Example 7: Move the immediate constant E2h to the destination (register R5).

Before operation: `R4=A002h R5=050Ah`

Operation: `MOV #E2h, R5`

After operation: `R4= A002h R5=00E2h`

Table 4-13. Cycles required to perform an instruction contained in the first operand, in immediate mode.

Operands	2nd operand mode	Operator	Cycles	Length (words)
2	Register	Any	2*	2
2	Indexed, Symbolic or Absolute	Any	5	3
1	N/A	RRA, RRC, SWPB or SXT	N/A	N/A
1	N/A	PUSH	4	2
1	N/A	CALL	5	2

*Immediate mode instructions where destination is Program Counter (PC) require 3 cycles.

4.7 Instruction set

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.

There are three core-instruction formats:

- Double operand;
- Single operand;
- Program flow control - Jump.

Byte, word and address instructions are accessed using the `.B`, `.W` or `.A` extensions. If the extension is omitted, the instruction is interpreted as a word instruction.

Note: Additional information about the instruction set can be found in Chapter 15.

4.7.1 Double operand instructions

The double operand instruction is formatted as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode				S-Reg				Ad	B/W	As		D-Reg			

Bit		Description
15-12	opcode	
11-8	S-Reg	The working register used for the source operand (src)
7	Ad	The addressing bits responsible for the addressing mode used for the destination operand (dst)
6	B/W	Byte or word operation: B/W=0: word operation; B/W=1: byte operation
5-4	As	The addressing bits responsible for the addressing mode used for the source operand (src)
3-0	D-Reg	The working register used for the destination operand (dst)

Table 4-14 shows the double operand instructions that are not emulated.

Table 4-14. Double operand instructions that are not emulated.

Mnemonic	Operation	Description
Arithmetic instructions		
ADD(.B or .W) src,dst	src+dst→dst	Add source to destination
ADDC(.B or .W) src,dst	src+dst+C→dst	Add source and carry to destination
DADD(.B or .W) src,dst	src+dst+C→dst (dec)	Decimal add source and carry to destination
SUB(.B or .W) src,dst	dst+.not.src+1→dst	Subtract source from destination
SUBC(.B or .W) src,dst	dst+.not.src+C→dst	Subtract source and not carry from destination
Logical and register control instructions		
AND(.B or .W) src,dst	src.and.dst→dst	AND source with destination
BIC(.B or .W) src,dst	.not.src.and.dst→dst	Clear bits in destination
BIS(.B or .W) src,dst	src.or.dst→dst	Set bits in destination
BIT(.B or .W) src,dst	src.and.dst	Test bits in destination
XOR(.B or .W) src,dst	src.xor.dst→dst	XOR source with destination
Data instructions		
CMP(.B or .W) src,dst	dst-src	Compare source to destination
MOV(.B or .W) src,dst	src→dst	Move source to destination

Depending on the double operand instruction result, the status bits may be affected. *Table 4-15* gives the conditions for setting and resetting the status bits.

Table 4-15. Conditions for status bits, depending on the double operand instruction result.

	Status bits			
Mnemonic	**V**	**N**	**Z**	**C**
Arithmetic instructions				
ADD(.B or .W) src,dst	=1, Arithmetic overflow =0, otherwise	=1, negative result =0, if positive	=1, null result =0, otherwise	=1, carry from result =0, if not
ADDC(.B or .W) src,dst	=1, Arithmetic overflow =0, otherwise	=1, negative result =0, if positive	=1, null result =0, otherwise	=1, carry from MSB result =0, if not
DADD(.B or .W) src,dst	-	=1, MSB=1 =0, otherwise	=1, null result =0, otherwise	=1, result > 99(99)
SUB(.B or .W) src,dst	=1, Arithmetic overflow =0, otherwise	=1, negative result =0, if positive	=1, null result =0, otherwise	=1, if no borrow =0, otherwise
SUBC(.B or .W) src,dst	=1, Arithmetic overflow =0, otherwise	=1, negative result =0, if positive	=1, null result =0, otherwise	=1, if no borrow =0, otherwise
Logical and register control instructions				
AND(.B or .W) src,dst	=0	=1, MSB result set =0, if not set	=1, null result =0, otherwise	=1, not zero =0, otherwise
BIC(.B or .W) src,dst	-	-	-	-
BIS(.B or .W) src,dst	-	-	-	-
BIT(.B or .W) src,dst	=0	=1, MSB result set =0, otherwise	=1, null result =0, otherwise	=1, not zero =0, otherwise
XOR(.B or .W) src,dst	=1, both operands negative	=1, MSB result set =0, otherwise	=1, null result, =0, otherwise	=1, not zero =0, otherwise
Data instructions				
CMP(.B or .W) src,dst	=1, Arithmetic overflow =0, otherwise	=1, src>=dst =0, src<dst	=1, src=dst =0, otherwise	=1, carry from MSB result =0, if not
MOV(.B or .W) src,dst	-	-	-	-

4.7.2 Single operand instructions

The single operand instruction is formatted as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode									B/W	Ad		D/S-Reg			

Bit		Description
15-7	opcode	
6	B/W	Byte or word operation: B/W=0: word operation; B/W=1: byte operation
5-4	Ad	The addressing bits responsible for the addressing mode used for the source operand (src)
3-0	D/S-Reg	The working register used for the destination operand (dst) or for the source operand (src)

Table 4-16 shows the single operand instructions that are not emulated.

Table 4-16. Single operand instructions that are not emulated.

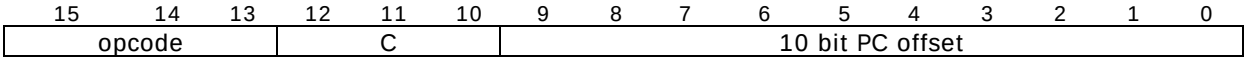
Mnemonic	Operation	Description
Logical and register control instructions		
RRA(.B or .W) dst	MSB→MSB→...LSB→C	Roll destination right
RRC(.B or .W) dst	C→MSB→...LSB→C	Roll destination right through (from) carry
SWPB(.B or .W) dst	Swap bytes	Swap bytes in destination
SXT dst	bit 7→bit 8...bit 15	Sign extend destination
PUSH(.B or .W) src	SP-2→SP, src→@SP	Push source to stack
Program flow control instructions		
CALL(.B or .W) dst	SP-2→SP, PC+2→@SP dst→PC	Subroutine call to destination
RETI	TOS→SR, SP+2→SP TOS→PC, SP+2→SP	Return from interrupt

Table 4-17. Conditions for status bits, depending on the single operand instruction result.

	Status bits			
Mnemonic	**V**	**N**	**Z**	**C**
Logical and register control instructions				
RRA(.B or .W) dst	=0	=1, negative result =0, otherwise	=1, null result, =0, otherwise	Loaded from LSB
RRC(.B or .W) dst	=1, dst positive & C=1 =0, otherwise	=1, negative result =0, otherwise	=1, null result, =0, otherwise	Loaded from LSB
SWPB(.B or .W) dst	-	-	-	-
SXT dst	=0	=1, negative result =0, otherwise	=1, null result, =0, otherwise	=1, not zero =0, otherwise
PUSH(.B or .W) src	-	-	-	-
Data instructions				
CALL(.B or .W) dst	-	-	--	-
RETI	restored from stack	restored from stack	restored from stack	restored from stack

4.7.3 Program flow control - Jumps

The jump instruction is formatted as follows:



Bit		Description
15-13	opcode	
12-10	C	
9-0	PC offset	PCnew = PCold + 2 + PCoffset × 2

Table 4-18 shows the program flow control (jump) instructions that are not emulated.

Table 4-18. Program flow control (jump) instructions.

Mnemonic	Description
Program flow control instructions	
`JEQ/JZ label`	Jump to label if zero flag is set
`JNE/JNZ label`	Jump to label if zero flag is reset
`JC label`	Jump to label if carry flag is set
`JNC label`	Jump to label if carry flag is reset
`JN label`	Jump to label if negative flag is set
`JGE label`	Jump to label if greater than or equal
`JL label`	Jump to label if less than
`JMP label`	Jump to label unconditionally

4.7.4 Emulated instructions

Table 4-19 gives the different emulated instructions. This table also contains the type of operation and the emulated instruction based on the core instructions.

Table 4-19. Emulated instructions.

Mnemonic	Operation	Emulation	Description
Arithmetic instructions			
ADC(.B or .W) dst	dst+C→dst	ADDC(.B or .W) #0,dst	Add carry to destination
DADC(.B or .W) dst	dst+C→dst (decimally)	DADD(.B or .W) #0,dst	Decimal add carry to destination
DEC(.B or .W) dst	dst-1→dst	SUB(.B or .W) #1,dst	Decrement destination
DECD(.B or .W) dst	dst-2→dst	SUB(.B or .W) #2,dst	Decrement destination twice
INC(.B or .W) dst	dst+1→dst	ADD(.B or .W) #1,dst	Increment destination
INCD(.B or .W) dst	dst+2→dst	ADD(.B or .W) #2,dst	Increment destination twice
SBC(.B or .W) dst	dst+0FFFFh+C→dst dst+0FFh→dst	SUBC(.B or .W) #0,dst	Subtract source and borrow /.NOT. carry from dest.
Logical and register control instructions			
INV(.B or .W) dst	.NOT.dst→dst	XOR(.B or .W) #0(FF)FFh,dst	Invert bits in destination
RLA(.B or .W) dst	C←MSB←MSB-1…LSB+1←LSB←0	ADD(.B or .W) dst,dst	Rotate left arithmetically
RLC(.B or .W) dst	C←MSB←MSB-1…LSB+1←LSB←C	ADDC(.B or .W) dst,dst	Rotate left through carry
Data instructions			
CLR(.B or .W) dst	0→dst	MOV(.B or .W) #0,dst	Clear destination
CLRC	0→C	BIC #1,SR	Clear carry flag
CLRN	0→N	BIC #4,SR	Clear negative flag
CLRZ	0→Z	BIC #2,SR	Clear zero flag
POP(.B or .W) dst	@SP→temp SP+2→SP temp→dst	MOV(.B or .W) @SP+,dst	Pop byte/word from stack to destination
SETC	1→C	BIS #1,SR	Set carry flag
SETN	1→N	BIS #4,SR	Set negative flag
SETZ	1→Z	BIS #2,SR	Set zero flag
TST(.B or .W) dst	dst + 0FFFFh + 1 dst + 0FFh + 1	CMP(.B or .W) #0,dst	Test destination
Program flow control			
BR dst	dst→PC	MOV dst,PC	Branch to destination
DINT	0→GIE	BIC #8,SR	Disable (general) interrupts
EINT	1→GIE	BIS #8,SR	Enable (general) interrupts
NOP	None	MOV #0,R3	No operation
RET	@SP→PC SP+2→SP	MOV @SP+,PC	Return from subroutine

Table 4-20. Conditions for status bits, depending on the emulated instruction result.

	Status bits			
Mnemonic	**V**	**N**	**Z**	**C**
Arithmetic instructions				
ADC(.B or .W) dst	=1, Arithmetic overflow =0, otherwise	=1, negative result =0, if positive	=1, null result =0, otherwise	=1, dst from 0FFFFh to 0000 =0, otherwise
DADC(.B or .W) dst	-	=1, MSB=1 =0, otherwise	=1, dst=0 =0, otherwise	=1, dst from 99(99) to 00(00) =0, otherwise
DEC(.B or .W) dst	=1, Arithmetic overflow =0, otherwise	=1, negative result =0, if positive	=1, dst contained 1 =0, otherwise	=1, dst contained 0 =0, otherwise
DECD(.B or .W) dst	=1, Arithmetic overflow =0, otherwise	=1, negative result =0, if positive	=1, dst contained 2 =0, otherwise	=1, dst contained 0 or 1 =0, otherwise
INC(.B or .W) dst	=1, dst contained 07(FF)h =0, otherwise	=1, negative result =0, if positive	=1, dst contained FF(FF)h =0, otherwise	=1, dst contained FF(FF)h =0, otherwise
INCD(.B or .W) dst	=1, dst contained 07(FFE)h =0, otherwise	=1, negative result =0, if positive	=1, dst contained FF(FE)h =0, otherwise	=1, dst contained FF(FF)h or FF(FE)h =0, otherwise
SBC(.B or .W) dst	=1, Arithmetic overflow =0, otherwise	=1, negative result =0, if positive	=1, null result, =0, otherwise	=1, if no borrow =0, otherwise
Logical and register control instructions				
INV(.B or .W) dst	=1, negative initial dst =0, otherwise	=1, negative result =0, if positive	=1, dst contained FF(FF)h =0, otherwise	=1, not zero =0, otherwise
RLA(.B or .W) dst	=1, Arithmetic overflow =0, otherwise	=1, negative result =0, if positive	=1, null result, =0, otherwise	Loaded from MSB
RLC(.B or .W) dst	=1, Arithmetic overflow =0, otherwise	=1, negative result =0, if positive	=1, null result, =0, otherwise	Loaded from MSB
Data instructions				
CLR(.B or .W) dst	-	-	-	-
CLRC	-	-	-	=0
CLRN	-	=0	-	-
CLRZ	-	-	=0	-
POP(.B or .W) dst	-	-	-	-
SETC	-	-	-	=1
SETN	-	=1	-	-
SETZ	-	-	=1	-
TST(.B or .W) dst	=0	=1, dst negative =0, otherwise	=1, dst contains zero =0, otherwise	=1
Program flow control				
BR dst	-	-	-	-
DINT	-	-	-	-
EINT	-	-	-	-
NOP	-	-	-	-
RET	-	-	-	-

4.8 Quiz

1. The number of instructions supported by the MSP430 CPU is:

(a) 27 instructions;

(b) 20 core instructions and 14 emulated instructions;

(c) 27 core instructions and 24 emulated instructions;

(d) 24 core instructions.

2. The MSP430 RISC type CPU is:

(a) Based on a reduced instruction set;

(b) Based on pure pattern matching and absence of instructions;

(c) Based on a complex instruction set;

(d) A CPU without peripheral connections.

3. The von Neumann architecture used by the MSP430:

(a) Has data storage entirely contained within the data processing unit;

(b) Has physically separate storage and signal pathways for instructions and data;

(c) Has a separate bus just for peripherals;

(d) Has program, data memory and peripherals, all sharing a common bus structure.

4. The ALU included in the MSP430 CPU handles:

(a) Addition, subtraction, multiplication and division operations;

(b) Addition, subtraction, comparison and logical (AND, OR, XOR) operations;

(c) Addition, subtraction, multiplication and comparison operations;

(d) Addition, subtraction, multiplication and logical (AND, OR, XOR) operations.

5. The MSP430 CPU incorporates:

(a) 14 registers (2 for dedicated functions and 12 working registers);

(b) 16 registers (6 for dedicated functions and 10 working registers);

(c) 18 registers (4 for dedicated functions and 14 working registers);

(d) 16 registers (4 for dedicated functions and 12 working registers).

6. The Program Counter (PC):

(a) Stores the return addresses of subroutine calls and interrupts;

(b) Points to the next instruction to be read from memory and executed by CPU;

(c) Stores state and control bits;

(d) Points to the next instruction to be written in memory.

7. The result in the Status Register SR = 0x0104 indicates:

(a) Arithmetic operation result overflows the signed-variable range and produced a carry;

(b) Arithmetic operation result overflows the signed-variable range which result is negative, when maskable interrupts are enabled;

(c) Arithmetic operation result is negative and produced a carry;

(d) CPU is disabled and the maskable interrupts are enabled.

8. The MSP430 Status Register (SR) bit:

(a) V is set when the result of a byte or word operation overflows;

(b) Z is set when the result of a byte or word operation is zero;

(c) All of the above;

(d) None of the above.

9. The MSP430 supports on two-address-instructions:

(a) Seven addressing modes for the source operand and three addressing modes for the destination operand;

(b) Six addressing modes for the source operand and four addressing modes for the destination operand;

(c) Seven addressing modes for the source operand and four addressing modes for the destination operand;

(d) Six addressing modes for the source operand and three addressing modes for the destination operand.

Solution: 1. (c); 2. (a); 3. (d); 4. (b); 5. (d); 6. (b); 7. (b); 8. (c); 9. (c)

4.9 FAQs

1. Can I access the data in the register mode?

The data in the register can be accessed using word or byte instructions. If byte instructions are used, the MSB byte is always 0 in the result. The status bits (V, N, Z, C) are modified according to the result of the byte instruction.

2. What are the available destination addresses of any kind (opcode or emulated) instruction?

Destination addresses are valid anywhere in the memory map. It must be ensured that an instruction that modifies the contents of the destination uses a writable address.