

Lecture 2

MSP430 Architecture

Atul Lele

Ramakrishna Reddy

About me

- Education
 - B.E. in Electronics and Telecommunication, PICT, Pune (2000)
- Professional experience
 - Working with Texas Instruments India Pvt Ltd for last 11 years
 - Expertise in:
 - ASM based verification, Analog and Mixed Signal Simulations, Synthesis, Static Timing Analysis, Low Power Design.
 - Current Role
 - Design Architect (Member Group Technical Staff)

Outline

- Summary from previous session/s.
- Agenda for this session.
 - MSP430 Architecture
 - Instruction Set
 - Compiler Friendly Features
 - Memory Sub-System
 - Clock System
- Wrap-Up.
- Q&A

Summary from previous session/s

Summary from previous session/s

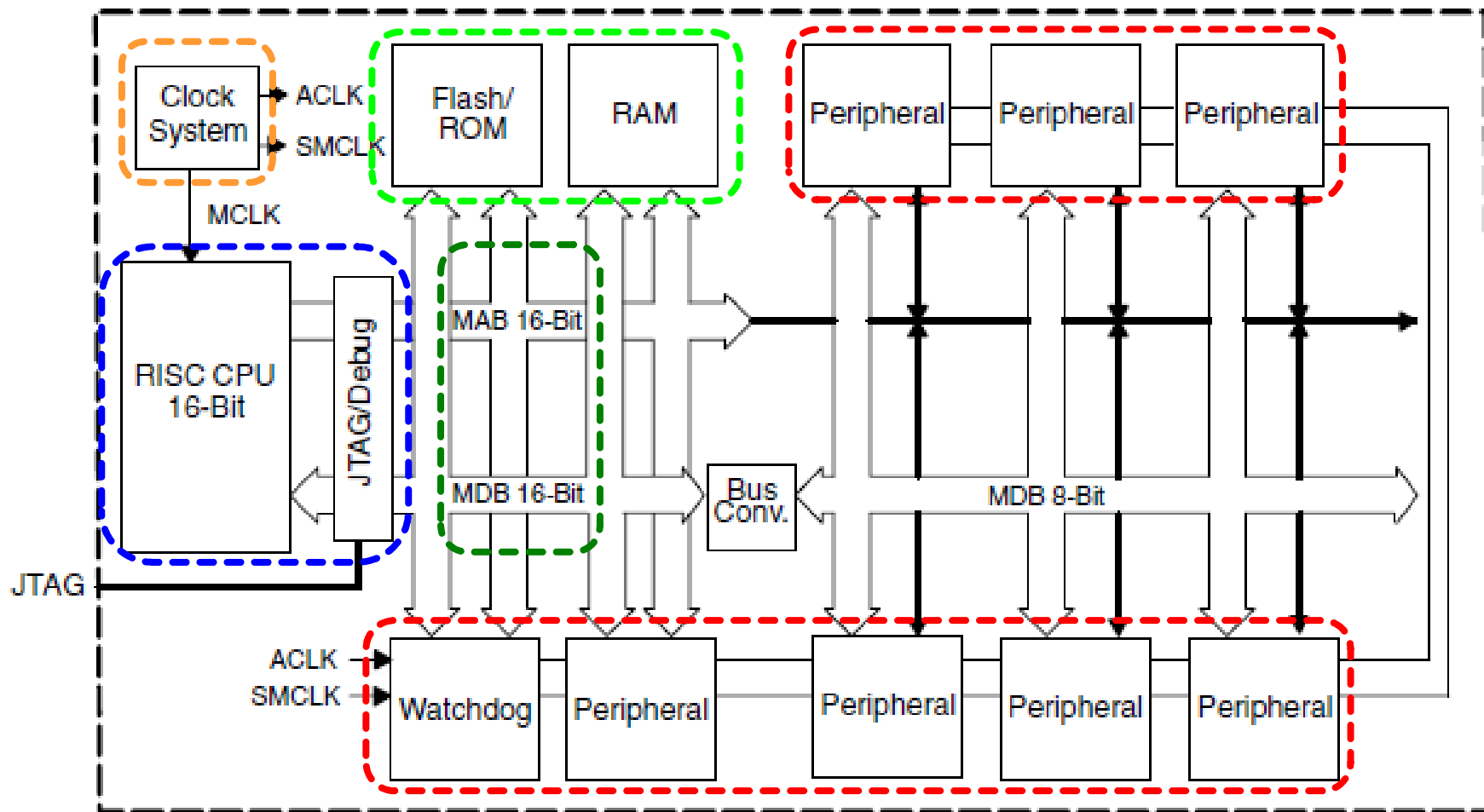
- Embedded systems
- Embedded system design cycle
- Need for Low-Power embedded systems
- Power Aware Architecture
- Power saving techniques
- Choosing a suitable microcontroller

Agenda for this session

Agenda for this session

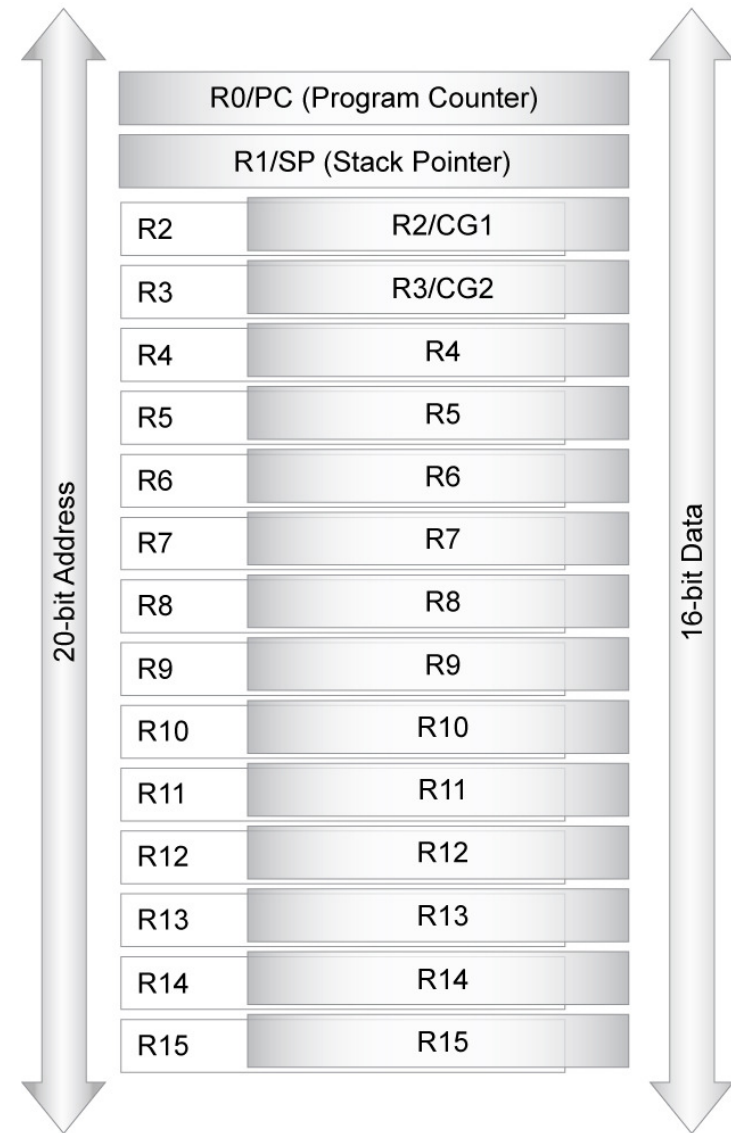
- MSP430 Architecture
- Instruction Set
- Compiler Friendly Features
- Memory Sub-System
- Clock System

MSP430 Architecture

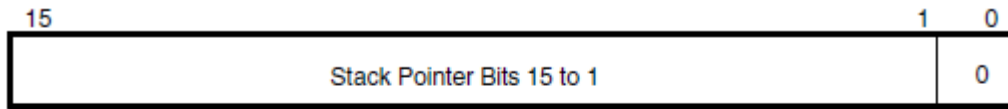


16-bit RISC CPU

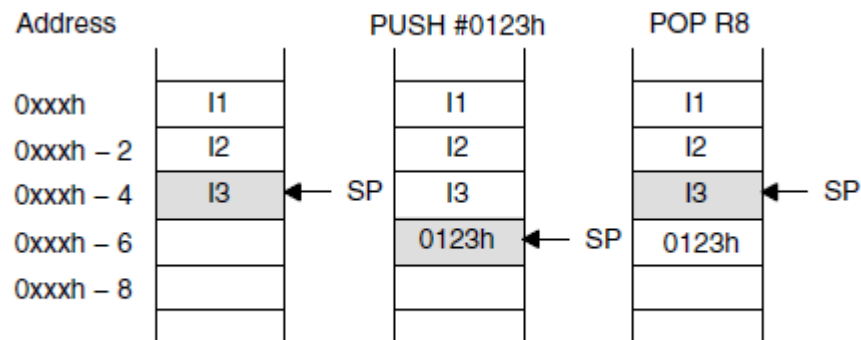
- Efficient, ultra-low power CPU
- C-compiler friendly
- RISC architecture
 - 27 core instructions
 - 24 emulated instructions
 - 7 addressing modes
 - Constant generator
- Single-cycle register operations
- Memory-to-memory atomic addressing
- Bit, byte and word processing
- 20-bit addressing on MSP430X for Flash >64KB



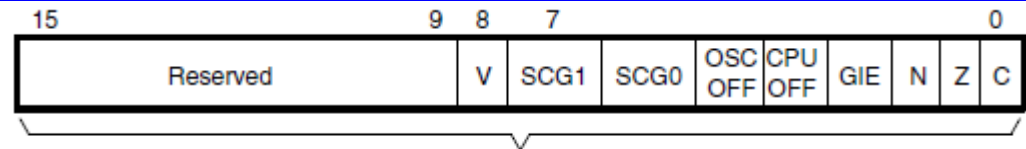
Stack Behavior: Push and Pop



```
MOV    2(SP),R6 ; Item I2 -> R6
MOV    R7,0(SP) ; Overwrite TOS with R7
PUSH   #0123h   ; Put 0123h onto TOS
POP    R8       ; R8 = 0123h
```



Status Register



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 use 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.

V Overflow bit. This bit is set when the result of an arithmetic operation overflows the signed-variable range.

ADD(.B),ADDC(.B) Set when: Positive + Positive = Negative, Negative + Negative = Positive
SUB(.B),SUBC(.B),CMP(.B) Set when: Positive – Negative = Negative. Negative – Positive = Positive

N Negative bit. This bit is set when the result of a byte or word operation is negative and cleared when the result is not negative.

Word operation: N is set to the value of bit 15 of the result

Byte operation: N is set to the value of bit 7 of the result

Z Zero bit. This bit is set when the result of a byte or word operation is 0 and cleared when the result is not 0.

C Carry bit. This bit is set when the result of a byte or word operation produced a carry and cleared when no carry occurred.

Is the MSP430 a RISC

- RISC

MSP430

- Small set of general purpose instructions
- Large bank of general purpose registers
- Load-store architecture
- Single-cycle execution



Is the MSP430 a RISC(Contd)

- Why not load-store architecture

MSP430 : Ex : ***bic.w #MC0 | MC1, &TACTL*** ; 3 words, 5 cycles

Pure RISC : Ex : ***load.w #TACTL, R4*** ; load address of TACTL [2 words, 2 cycles]
load.w @R4, R5 ; load value of TACTL [1 word, 2 cycles]
load.w #MC0 | MC1, R6 ; load immediate operand [2 words, 2 cycles]
bic.w R6, R5 ; perform operation [1 word, 1 cycle]
store.w R5, @R4 ; store result for TACTL [1 word, 2 cycles]

Atomic

Poor code density

- Why not single cycle execution

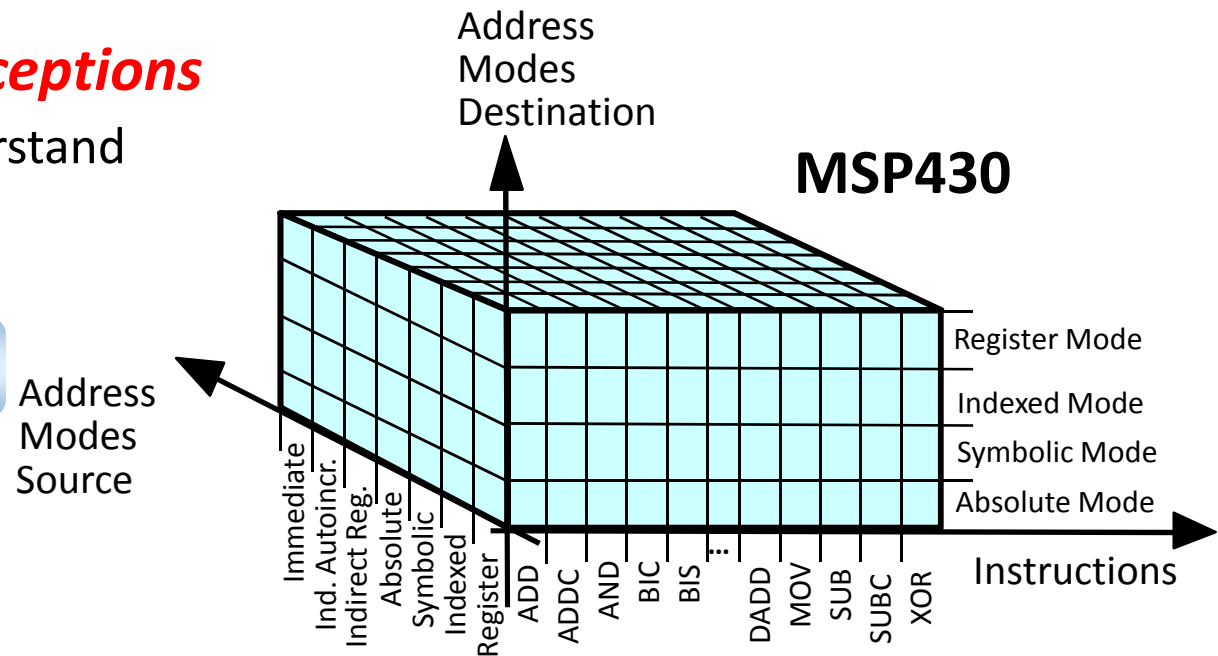
To support orthogonal instruction set

Orthogonal Architecture

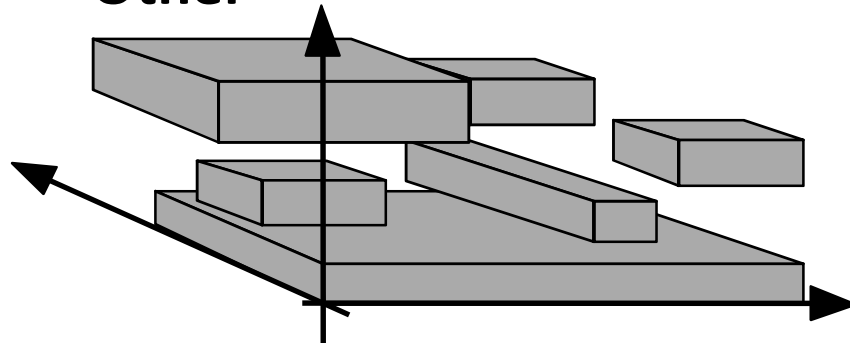
Consistent With No Exceptions

- Clear and easy to understand
- No special instructions
- Compiler efficient

Compiler Friendly



Other



- Special instructions to learn
- Complicated
- Inefficient

Bytes, Words And CPU Registers

16-bit addition

			Code/Cycles
5405	add.w	R4, R5	; 1/1
529202000202	add.w	&0200, &0202	; 3/6

8-bit addition

5445	add.b	R4, R5	; 1/1
52D202000202	add.b	&0200, &0202	; 3/6

- Use CPU registers for calculations and dedicated variables
- Same code size for word or byte
- Use word operations when possible



Seven Addressing Modes

Register Mode	<code>mov.w R10, R11</code> Single cycle
Indexed Mode	<code>mov.w 2(R5), 6(R6)</code> Table processing
Symbolic Mode	<code>mov.w EDE, TONI</code> Easy to read code, PC relative
Absolute Mode	<code>mov.w &EDE, &TONI</code> Directly access any memory
Indirect Register Mode	<code>mov.w @R10, 0(R11)</code> Access memory with pointers
Indirect Autoincrement	<code>mov.w @R10+, 0(R11)</code> Table processing
Immediate Mode	<code>mov.w #45h, &TONI</code> Unrestricted constant values



Register Mode

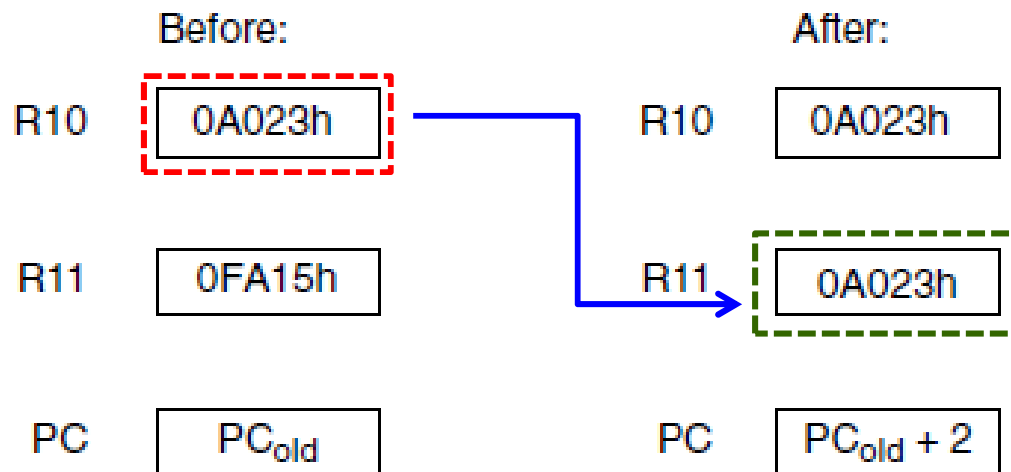
Assembler Code	Content of ROM
MOV R10,R11	MOV R10,R11

Length: One or two words

Operation: Move the content of R10 to R11. R10 is not affected.

Comment: Valid for source and destination

Example: MOV R10,R11



Indexed Mode

Assembler Code	Content of ROM
MOV 2 (R5) , 6 (R6)	MOV X (R5) , Y (R6)
	X = 2
	Y = 6

Example: MOV 2 (R5) , 6 (R6)

Before:

	Address Space
0FF16h	00006h
0FF14h	00002h
0FF12h	04596h
01094h	0xxxxh
01092h	05555h
01090h	0xxxxh
01084h	0xxxxh
01082h	01234h
01080h	0xxxxh

Register

R5	01080h
R6	0108Ch

PC

0108Ch +0006h 01092h

01080h +0002h 01082h

After:

	Address Space
0FF16h	0xxxxh
0FF14h	00006h
0FF12h	00002h
01094h	0xxxxh
01092h	01234h
01090h	0xxxxh
01084h	0xxxxh
01082h	01234h
01080h	0xxxxh

Register

R5	01080h
R6	0108Ch



Symbolic Mode

Assembler Code	Content of ROM
MOV EDE,TONI	MOV X(PC),Y(PC) X = EDE - PC Y = TONI - PC

Example:

MOV	EDE,TONI	;Source address	EDE = 0F016h
		;Dest. address	TONI=01114h

Before:

Address Space	Register
0FF16h	011FEh
0FF14h	0F102h
0FF12h	04090h

After:

Address Space	Register
0FF16h	011FEh
0FF14h	0F102h
0FF12h	04090h

PC Update:

Address Space	Register
0F018h	0xxxxh
0F016h	0A123h
0F014h	0xxxxh

Instruction Memory:

Address Space	Register
01116h	0xxxxh
01114h	05555h
01112h	0xxxxh

Calculation:

$$0FF14h + 0F102h = 0F016h$$

$$0FF16h + 011FEh = 01114h$$

Final State:

Address Space	Register
0F018h	0xxxxh
0F016h	0A123h
0F014h	0xxxxh

Instruction Memory:

Address Space	Register
01116h	0xxxxh
01114h	0A123h
01112h	0xxxxh

A yellow arrow points from the 0A123h value in the Instruction Memory to the 0A123h value in the PC register.

Absolute Mode

Assembler Code	Content of ROM
MOV &EDE,&TONI	MOV X(0),Y(0) X = EDE Y = TONI

Example:

```
MOV    &EDE,&TONI ;Source address EDE=0F016h,  
                ;dest. address TONI=01114h
```

Before:

Address Space	Register
0FF16h	01114h
0FF14h	0F016h
0FF12h	04292h

PC

After:

Address Space	Register
0FF16h	01114h
0FF14h	0F016h
0FF12h	04292h

PC

0F018h: 0xxxxh

0F016h: 0A123h

0F014h: 0xxxxh

01116h: 0xxxxh

01114h: 01234h

01112h: 0xxxxh

Indirect Register Mode

Assembler Code	Content of ROM
MOV @R10,0(R11)	MOV @R10,0(R11)

Example: MOV.B @R10,0(R11)

Before:

Address Space
0xxxxh
0FF16h 0000h
0FF14h 04AEBh
0FF12h 0xxxxh

0FA34h	0xxxxh
0FA32h	05BC1h
0FA30h	0xxxxh

002A8h	0xxh
002A7h	012h
002A6h	0xxh

Register

R10	0FA33h
PC R11	002A7h

After:

Address Space
0xxxxh
0FF16h 0000h
0FF14h 04AEBh
0FF12h 0xxxxh

0FA34h	0xxxxh
0FA32h	05BC1h
0FA30h	0xxxxh

002A8h	0xxh
002A7h	05Bh
002A6h	0xxh

Register

R10	0FA33h
R11	002A7h



Indirect Auto Increment Mode

Assembler Code	Content of ROM
MOV @R10+, 0 (R11)	MOV @R10+, 0 (R11)

Example: MOV @R10+, 0 (R11)

Before:

Address Space	Register
0FF18h	0xxxxh
0FF16h	00000h
0FF14h	04ABBh
0FF12h	0xxxxh
0FA34h	0xxxxh
0FA32h	05BC1h
0FA30h	0xxxxh
010AAh	0xxxxh
010A8h	01234h
010A6h	0xxxxh

Register

R10	0FA32h
PC R11	010A8h

After:

Address Space	Register
0FF18h	0xxxxh
0FF16h	00000h
0FF14h	04ABBh
0FF12h	0xxxxh
0FA34h	0xxxxh
0FA32h	05BC1h
0FA30h	0xxxxh
010AAh	0xxxxh
010A8h	05BC1h
010A6h	0xxxxh

PC

R10	0FA34h
R11	010A8h



Immediate Mode

Assembler Code	Content of ROM
MOV #45h, TONI	MOV @PC+, X (PC)
	45
	$X = \text{TONI} - \text{PC}$

Example: MOV #45h, TONI

Before:

Address
Space

Register

After:

Address
Space

Register

0FF16h	01192h	PC
0FF14h	00045h	
0FF12h	040B0h	

010AAh	0xxxxh
010A8h	01234h
010A6h	0xxxxh

0FF18h	0xxxxh	PC
0FF16h	01192h	
0FF14h	00045h	
0FF12h	040B0h	

0FF16h		010AAh	0xxxxh
+01192h		010A8h	00045h
010A8h		010A6h	0xxxxh

Symbolic/Absolute Addressing Modes(1)

The screenshot displays the IAR Embedded Workbench interface for an MSP430 microcontroller. The main window shows the assembly code for a program. The assembly window is set to 'Absolute addressing mode'. The code defines two constants, FOO (0x0250) and BAR (0x0300), and then executes a series of instructions. The instruction at address 00F80A is highlighted in green, showing a move operation from FOO to BAR. The disassembly window shows the same instruction with its parameters: 00F80A 4090 0A44 0AF2 mov.w 0x10250,0x300. The memory dump window shows the memory contents, with the value 0010 at address 0000250 highlighted in red. An arrow points from the text 'Source address written' to this value.

```
#define FOO (0x0250u)
#define BAR (0x0300u)
ORG 0xF800
RESET:
    mov.w #0280h, SP ; Init
    mov.w #00010h, &FOO ;
    mov.w FOO, BAR ;
    nop
    nop
    mov.w #00020h, &FOO ;
    mov.w &FOO, &BAR ;
    nop
```

Disassembly:

```
RESET:
00F800 4031 0280      mov.w #0x280, SP
          mov.w #00010h, &FOO ;
00F804 40B2 0010 0250  mov.w #0x10, &0x250
          mov.w FOO, BAR ;
→ 00F80A 4090 0A44 0AF2  mov.w 0x10250,0x300
          nop
00F810 4303      nop
          nop
00F812 4303      nop
          mov.w #00020h, &FOO ;
00F814 40B2 0020 0250  mov.w #0x20, &0x250
          mov.w &FOO, &BAR ;
```

Memory Dump:

00000210	0000	0000	0000	0000	0000	0000	0000	0000
00000220	0000	0000	0000	0000	0000	0000	0000	0000
00000230	0000	0000	0000	0000	0000	0000	0000	0000
00000240	0000	0000	0000	0000	0000	0000	0000	0000
00000250	0010	0000	0000	0000	0000	0000	0000	0000
00000260	0000	0000	0000	0000	0000	0000	0000	0000
00000270	0000	0000	0000	0000	0000	0000	0000	0000
00000280	0000	0000	0000	0000	0000	0000	0000	0000
00000290	0000	0000	0000	0000	0000	0000	0000	0000
000002a0	0000	0000	0000	0000	0000	0000	0000	0000
000002b0	0000	0000	0000	0000	0000	0000	0000	0000
000002c0	0000	0000	0000	0000	0000	0000	0000	0000
000002d0	0000	0000	0000	0000	0000	0000	0000	0000
000002e0	0000	0000	0000	0000	0000	0000	0000	0000
000002f0	0000	0000	0000	0000	0000	0000	0000	0000
00000300	0000	0000	0000	0000	0000	0000	0000	0000
00000310	0000	0000	0000	0000	0000	0000	0000	0000
00000320	0000	0000	0000	0000	0000	0000	0000	0000
00000330	0000	0000	0000	0000	0000	0000	0000	0000
00000340	0000	0000	0000	0000	0000	0000	0000	0000
00000350	0000	0000	0000	0000	0000	0000	0000	0000
00000360	0000	0000	0000	0000	0000	0000	0000	0000

- When the device is executing instruction at F80Ah, the PC will be holding F80Ch
- Source Address = F80Ch + 0A44h = 10250
- When device is fetching source address, PC = F80Eh
- Destination address = F80Eh + 0AF2h = 10300

Source address written

Symbolic/Absolute Addressing Modes(2)

The screenshot displays the IAR Information Center for MSP430, showing the assembly code and its disassembly. The assembly code defines two constants, FOO (0x0250) and BAR (0x0300), and a RESET section. The disassembly shows the corresponding machine code. A memory dump at the bottom shows the state of memory, with a box highlighting the value 0010 at address 0000300, which is pointed to by a line from the disassembly window.

Assembly Code:

```
#define FOO (0x0250u)
#define BAR (0x0300u)

ORG 0xF800

RESET:
    mov.w #0280h, SP ; Init
    mov.w #00010h, &FOO ;
    mov.w FOO, BAR ;
    nop
    nop
    mov.w #00020h, &FOO ;
    mov.w &FOO, &BAR ;
    nop
```

Disassembly:

```
RESET:
00F800 4031 0280      mov.w #0x280, SP
          mov.w #00010h, &FOO ;
00F804 40B2 0010 0250  mov.w #0x10, &0x250
          mov.w FOO, BAR ;
00F80A 4090 0A44 0AF2  mov.w 0x10250, 0x300
          nop
→ 00F810 4303      nop
          nop
00F812 4303      nop
          mov.w #00020h, &FOO ;
00F814 40B2 0020 0250  mov.w #0x20, &0x250
          mov.w &FOO, &BAR ;
```

Memory Dump:

00000210	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000220	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000230	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000240	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000250	0010	0000	0000	0000	0000	0000	0000	0000	0000
00000260	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000270	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000280	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000290	0000	0000	0000	0000	0000	0000	0000	0000	0000
000002a0	0000	0000	0000	0000	0000	0000	0000	0000	0000
000002b0	0000	0000	0000	0000	0000	0000	0000	0000	0000
000002c0	0000	0000	0000	0000	0000	0000	0000	0000	0000
000002d0	0000	0000	0000	0000	0000	0000	0000	0000	0000
000002e0	0000	0000	0000	0000	0000	0000	0000	0000	0000
000002f0	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000300	0010	0000	0000	0000	0000	0000	0000	0000	0000
00000310	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000320	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000330	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000340	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000350	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000360	0000	0000	0000	0000	0000	0000	0000	0000	0000

Annotations:

- When the device is executing instruction at F80Ah, the PC will be holding F80Ch
- Source Address = F80Ch + 0A44h = 10250
- When device is fetching source address, PC = F80Eh
- Destination address = F80Eh + 0AF2h = 10300

Destination address written

Symbolic/Absolute Addressing Modes(3)

The screenshot displays the IAR IDE interface for an MSP430 microcontroller. The main window shows the assembly code for a program. The code defines two constants, FOO (0x0250) and BAR (0x0300), and then moves values into them. The memory dump window shows the resulting values in memory, with a callout indicating the source address 0020 is written.

Assembly Code:

```
#define FOO (0x0250u)
#define BAR (0x0300u)
ORG 0xF800
RESET:
    mov.w #0280h, SP ; Init
    mov.w #00010h, &FOO ;
    mov.w FOO, BAR ;
    nop
    nop
    mov.w #00020h, &FOO ;
    mov.w &FOO, &BAR ;
    nop
```

Disassembly:

Address	OpCode	Comment
00F810	4303	nop
00F812	4303	nop
00F814	40B2 0020 0250	mov.w #00020h, &FOO ;
00F81A	4292 0250 0300	mov.w &FOO, &BAR ;
00F820	4303	nop
00F822	0000	????
00F824	0000	????
00F826	0000	????

Memory Dump:

Address	Value
00000210	0000 0000 0000 0000 0000 0000 0000 0000
00000220	0000 0000 0000 0000 0000 0000 0000 0000
00000230	0000 0000 0000 0000 0000 0000 0000 0000
00000240	0000 0000 0000 0000 0000 0000 0000 0000
00000250	0020 0000 0000 0000 0000 0000 0000 0000
00000260	0000 0000 0000 0000 0000 0000 0000 0000
00000270	0000 0000 0000 0000 0000 0000 0000 0000
00000280	0000 0000 0000 0000 0000 0000 0000 0000
00000290	0000 0000 0000 0000 0000 0000 0000 0000
000002a0	0000 0000 0000 0000 0000 0000 0000 0000
000002b0	0000 0000 0000 0000 0000 0000 0000 0000
000002c0	0000 0000 0000 0000 0000 0000 0000 0000
000002d0	0000 0000 0000 0000 0000 0000 0000 0000
000002e0	0000 0000 0000 0000 0000 0000 0000 0000
000002f0	0000 0000 0000 0000 0000 0000 0000 0000
00000300	0010 0000 0000 0000 0000 0000 0000 0000
00000310	0000 0000 0000 0000 0000 0000 0000 0000
00000320	0000 0000 0000 0000 0000 0000 0000 0000
00000330	0000 0000 0000 0000 0000 0000 0000 0000
00000340	0000 0000 0000 0000 0000 0000 0000 0000
00000350	0000 0000 0000 0000 0000 0000 0000 0000
00000360	0000 0000 0000 0000 0000 0000 0000 0000

Source address written

Symbolic/Absolute Addressing Modes(4)

The screenshot displays the IAR Embedded Workbench interface for an MSP430 microcontroller. The main window shows assembly code with two macros defined: `FOO (0x0250u)` and `BAR (0x0300u)`. The code starts at address `0xF800` and includes instructions like `mov.w #0280h, SP`, `mov.w #00010h, &FOO`, `mov.w FOO, BAR`, and `nop`. A green highlight is placed on a `nop` instruction at the bottom of the code window.

On the right, the Disassembly window shows the corresponding machine code. It lists instructions with their addresses and hex values. A green highlight is placed on the instruction at address `00F820`, which is `4303 nop`.

At the bottom, a memory dump window shows a hex dump of memory starting from address `00000210` to `00000360`. The value `0020` at address `00000300` is highlighted with a red box. An arrow points from this box to a text box labeled "Destination address written".

Workspace: IAR Information Center for MSP430 | io430F4619.h | io430F4618.h | msp430xG46x.h | asm.s43

Debug: Fil... Asm

Absolute_addressing_mode

Go to: Memory

Disassembly: Go to: Memory

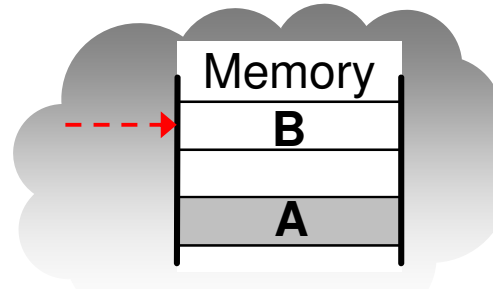
00F810 4303 nop
00F812 4303 nop
00F814 40B2 0020 0250 mov.w #0x20,&0x250
00F81A 4292 0250 0300 mov.w &0x250,&0x300
00F820 4303 nop
00F822 0000 ????
00F824 0000 ????
00F826 0000 ????

00000210 0000 0000 0000 0000 0000 0000 0000 0000
00000220 0000 0000 0000 0000 0000 0000 0000 0000
00000230 0000 0000 0000 0000 0000 0000 0000 0000
00000240 0000 0000 0000 0000 0000 0000 0000 0000
00000250 0020 0000 0000 0000 0000 0000 0000 0000
00000260 0000 0000 0000 0000 0000 0000 0000 0000
00000270 0000 0000 0000 0000 0000 0000 0000 0000
00000280 0000 0000 0000 0000 0000 0000 0000 0000
00000290 0000 0000 0000 0000 0000 0000 0000 0000
000002a0 0000 0000 0000 0000 0000 0000 0000 0000
000002b0 0000 0000 0000 0000 0000 0000 0000 0000
000002c0 0000 0000 0000 0000 0000 0000 0000 0000
000002d0 0000 0000 0000 0000 0000 0000 0000 0000
000002e0 0000 0000 0000 0000 0000 0000 0000 0000
000002f0 0000 0000 0000 0000 0000 0000 0000 0000
00000300 0020 0000 0000 0000 0000 0000 0000 0000
00000310 0000 0000 0000 0000 0000 0000 0000 0000
00000320 0000 0000 0000 0000 0000 0000 0000 0000
00000330 0000 0000 0000 0000 0000 0000 0000 0000
00000340 0000 0000 0000 0000 0000 0000 0000 0000
00000350 0000 0000 0000 0000 0000 0000 0000 0000
00000360 0000 0000 0000 0000 0000 0000 0000 0000

Destination address written

Atomic Addressing

B=B+A

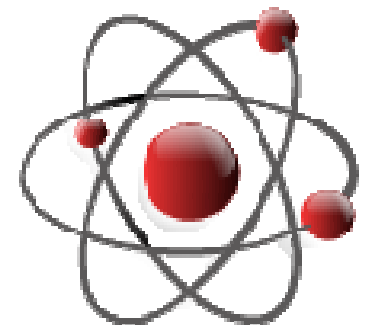


```
; Pure RISC  
push    R5  
ld      R5, A  
add     R5, B  
st      B, R5  
pop     R5
```

```
; MSP430
```

```
add     A, B
```

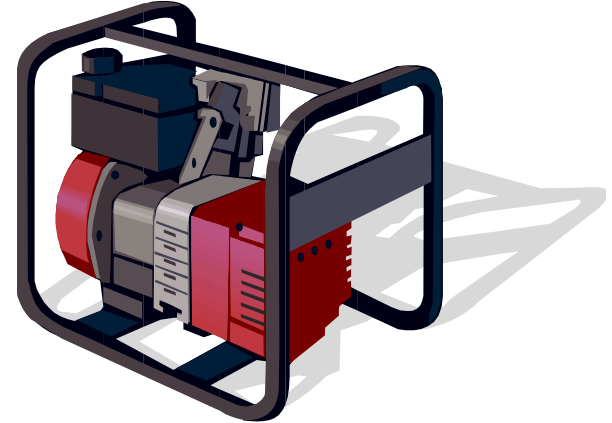
- Non-interruptible memory-to-memory operations
- Useable with complete instruction set



Constant Generator

<u>4</u> 314	mov.w	#0002h, R4	; With CG
4 <u>0</u> 341234	mov.w	#1234h, R4	; Without CG

- ◆ Immediate values -1,0,1,2,4,8 generated in hardware
- ◆ Reduces code size and cycles
- ◆ Completely automatic



24 Emulated Instructions

4130	ret	; Return (emulated)
4130	mov.w @SP+,PC	; Core instruction

- ◆ Easier to understand - no code size or speed penalty
- ◆ Replaced by assembler with core instructions
- ◆ Completely automatic

Emulated Instruction

```

asm.s43 msp430x20x3.h
#include <msp430x20x3.h>

ORG 0xF800

RESET:    mov.w #0280h, SP ; Initialise stack pointer
          clr R5
          nop ; Required o

;-----
WDT_ISR;  Toggle P1.0
;-----
          xor.b #001h, &P1OUT ; Toggle P1.0
          reti ;

;-----
          Interrupt Vectors
;-----

Disassembly
Go to Memory
00F7FC 0000 ???
00F7FE 0000 ???
RESET: mov.w #0280h, SP ; Initialise stack pointer
RESET: 00F800 4031 0280 mov.w #0x280, SP
          clr R5
→ 00F804 4305 clr.w R5
          nop
00F806 4303 nop
          xor.b #001h, &P1OUT ; To
WDT_ISR: 00F808 E3D2 0021 xor.b #0x1, &0x21
          reti ;
00F80C 1300 reti
00F80E 0000 ???
00F810 0000 ???
  
```

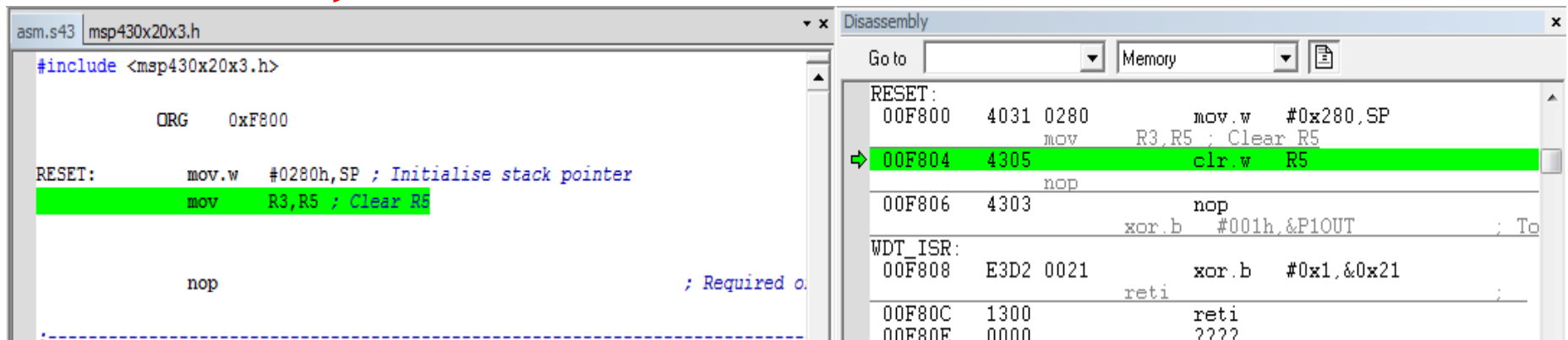
Instruction code: 0x4305

Op-code	S-reg	Ad	B/W	As	D-reg
0 1 0 0	0 0 1 1	0	0	0 0	0 1 0 1
MOV	R3	Register	16 Bits	Register	R5

This instruction is equivalent to using the instruction **MOV R3, R5** where R3 contains the value #0.

Emulated Instruction(Contd)

MOV R3, R5



The screenshot shows an IDE with two windows. The left window, titled 'asm.s43 msp430x20x3.h', contains assembly code. The right window, titled 'Disassembly', shows the disassembly of the code. The instruction 'MOV R3, R5' is highlighted in green in both windows.

```
#include <msp430x20x3.h>

ORG 0xF800

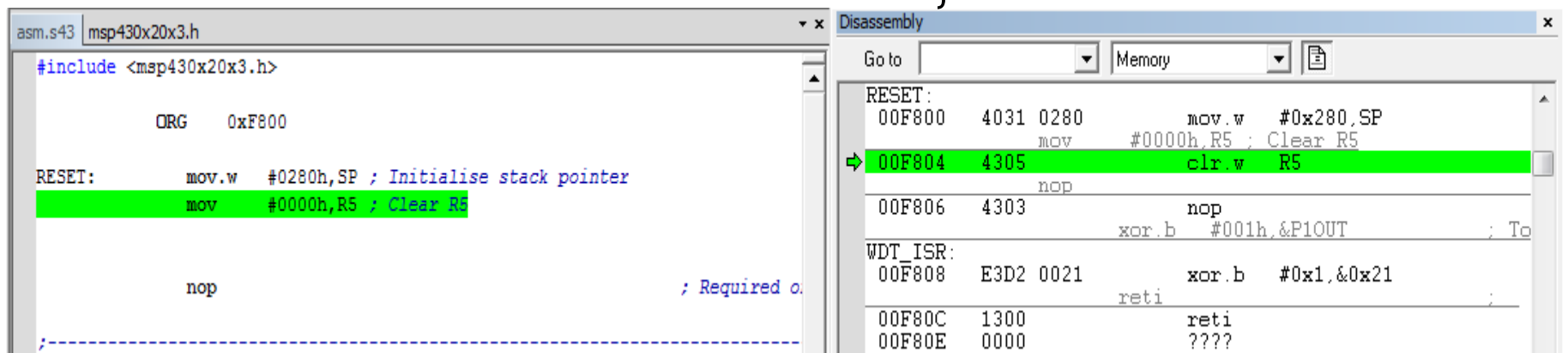
RESET: mov.w #0280h, SP ; Initialise stack pointer
      mov R3, R5 ; Clear R5

      nop ; Required o
```

Disassembly:

Address	Offset	Op-Code	Instruction
00F800	4031 0280	mov.w	#0x280, SP
00F804	4305	mov	R3, R5 ; Clear R5
00F806	4303	nop	
00F808	E3D2 0021	xor.b	#001h, &P1OUT ; To
00F80C	1300	reti	
00F80E	0000	???	

MOV #0000h, R5



The screenshot shows an IDE with two windows. The left window, titled 'asm.s43 msp430x20x3.h', contains assembly code. The right window, titled 'Disassembly', shows the disassembly of the code. The instruction 'MOV #0000h, R5' is highlighted in green in both windows.

```
#include <msp430x20x3.h>

ORG 0xF800

RESET: mov.w #0280h, SP ; Initialise stack pointer
      mov #0000h, R5 ; Clear R5

      nop ; Required o
```

Disassembly:

Address	Offset	Op-Code	Instruction
00F800	4031 0280	mov.w	#0x280, SP
00F804	4305	mov	#0000h, R5 ; Clear R5
00F806	4303	nop	
00F808	E3D2 0021	xor.b	#001h, &P1OUT ; To
00F80C	1300	reti	
00F80E	0000	???	

Three Assembly Instruction Formats

Format I

Source and Destination

<code>add.w</code>	<code>R4, R5</code>	<code>; R4+R5=R5</code>	<code>xxxx</code>
<code>add.b</code>	<code>R4, R5</code>	<code>; R4+R5=R5</code>	<code>00xx</code>

Format II

Destination Only

<code>rlc.w</code>	<code>R4</code>
<code>rlc.b</code>	<code>R4</code>

Format III

8(Un)conditional Jumps

<code>jmp</code>	<code>Loop_1</code>	<code>; Goto Loop_1</code>
------------------	---------------------	----------------------------

51 Total Assembly Instructions

Format I Source, Destination	Format II Single Operand	Format III +/- 9bit Offset	Support
add(.b)	br	jmp	clrc
addc(.b)	call	jc	setc
and(.b)	swpb	jnc	clrz
bic(.b)	sxt	jeq	setz
bis(.b)	push(.b)	jne	clrn
bit(.b)	pop(.b)	jge	setn
cmp(.b)	rra(.b)	j1	dint
dadd(.b)	rrc(.b)	jn	eint
mov(.b)	inv(.b)		nop
sub(.b)	inc(.b)		ret
subc(.b)	incd(.b)		reti
xor(.b)	dec(.b)		
	decd(.b)		
	adc(.b)		
	sbc(.b)		
	clr(.b)		
	dadc(.b)		
	rla(.b)		
	rlc(.b)		
	tst(.b)		

Bold type denotes emulated instructions

Using Assembly

```
#include <msp430f2013.h>
    ORG 0xF800
RESET:    mov.w #0280h,SP ; Initialise stack pointer
SetupWDT  mov.w #WDT_MDLY_32,&WDTCTL ; WDT ~30ms interval timer
          bis.b #WDTIE,&IE1          ; Enable WDT interrupt
SetupP1    bis.b #001h,&P1DIR          ; P1.0 output

Mainloop   bis.w #CPUOFF+GIE,SR        ; CPU off, enable interrupts
          nop                          ; Required only for debugger
;-----
WDT_ISR;   Toggle P1.0
;-----
          xor.b #001h,&P1OUT          ; Toggle P1.0
          reti                          ;
;-----
;      Interrupt Vectors
;-----
    ORG 0xFFFEh          ; MSP430 RESET Vector
    DW  RESET             ;
    ORG 0xFFF4h          ; WDT Vector
    DW  WDT_ISR           ;
    END
```

Using C

```
#include <msp430f2013.h>

void main (void)
{
    // WDTCTL = WDTPW | WDTTMSEL | WDTCTL ; // WDT ~30ms interval timer
    WDTCTL = WDT_MDLY_32;                // Set Watchdog Timer interval to ~30ms
    IE1 |= WDTIE;                        // Enable WDT interrupt
    P1DIR |= 0x01;                       // Set P1.0 to output direction

    __bis_SR_register(LPM0_bits + GIE);    // Enter LPM0 w/ interrupt
}

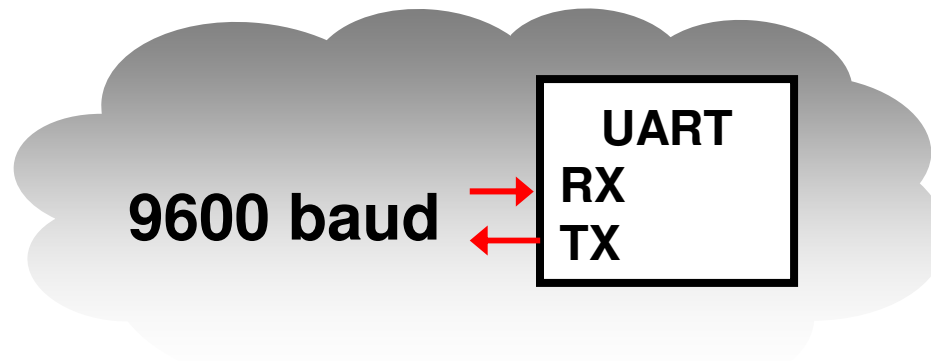
// Watchdog Timer interrupt service routine
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void)
{
    P1OUT ^= 0x01;                       // Toggle P1.0 using exclusive-OR
}
```

__BIS_SR(LPM0_bits+GIE) = _low_power_mode_0()

Mixing C and Assembly

- Check first to see whether an intrinsic function is available (see `intrinsics.h`) Ex :
`__swap_bytes()` calls the `swpb` instruction
 ← Note double underscore
- Inline assembly , `asm("mov.b &P1IN, &dest")`
- Write a complete subroutine in assembly and call it from C

Interrupts Control Program Flow



```
// Polling UART Receive  
for (;;)   
    {  
    while (!(IFG2&URXIFG0));  
    TXBUF0 = RXBUF0;  
    }
```

100% CPU Load

```
// UART Receive Interrupt  
#pragma vector=UART_VECTOR  
__interrupt void rx (void)  
{  
    TXBUF0 = RXBUF0;  
}
```

0.1% CPU Load

MSP430 Peripherals



WDT



Timer_A



Timer_B



BasicTimer1



USART



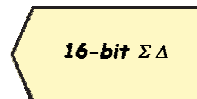
USCI



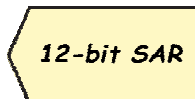
USI



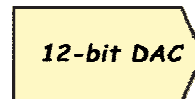
Comparator_A



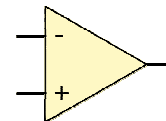
$\Sigma\Delta 16$



ADC10/12



DAC12



OP-Amps



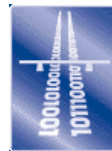
Scan IF



Hardware
Multiplier



LCD

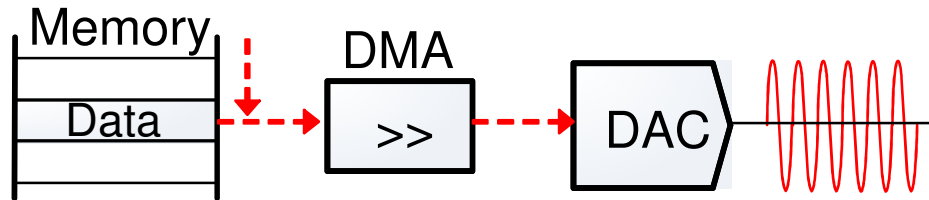


DMA

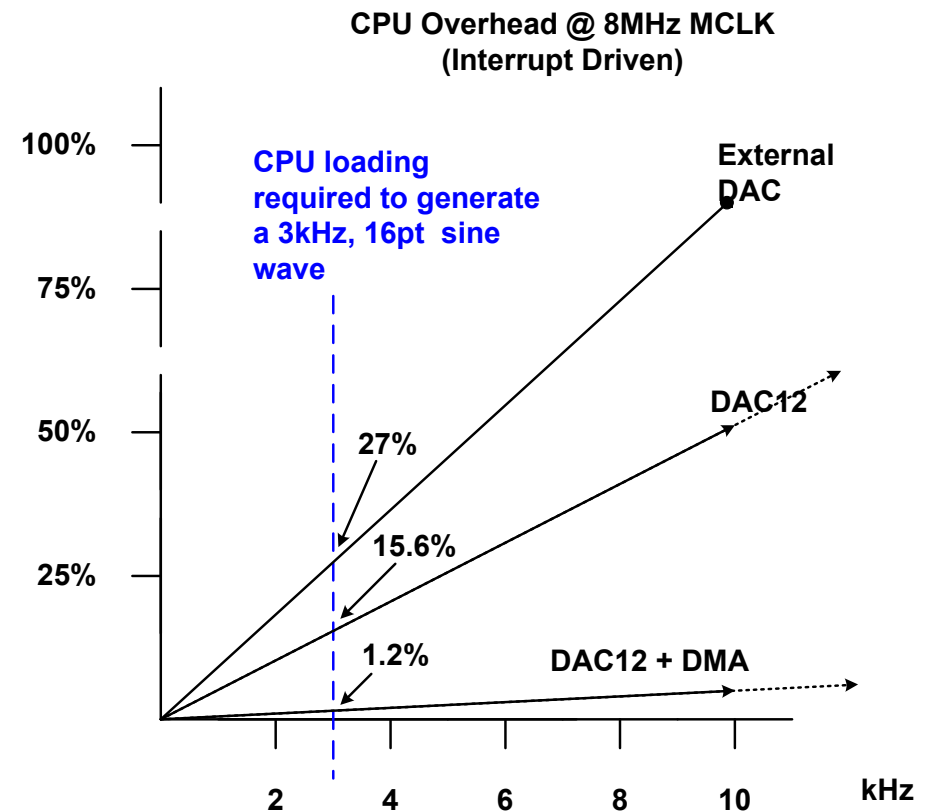


EEM

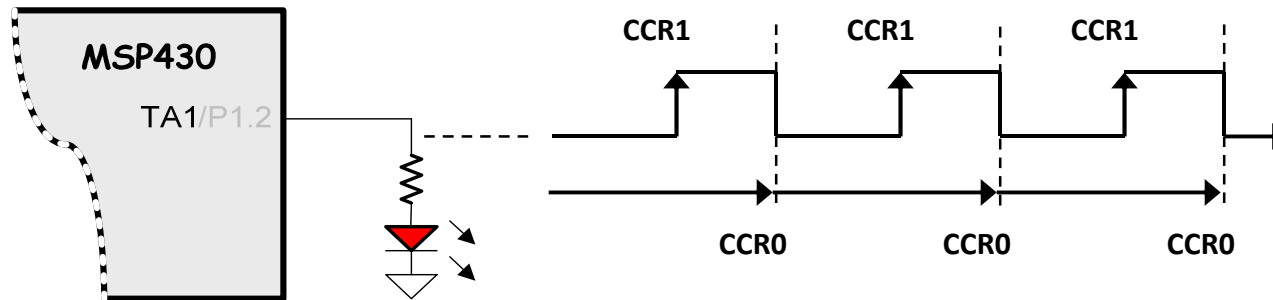
Intelligent Peripheral Performance



- Increased system flexibility
- No code execution required
- Lower power
- Higher efficiency



Replace Software With Hardware



**Zero CPU
overhead**

```
CCR0 = Period;           // Setup timer
CCTL1 = OUTMOD0_3;
CCR1 = Duty;
TACTL = TASSEL1 + MC0;   // Start timer
_BIS_SR(CPUOFF);         // No CPU
```

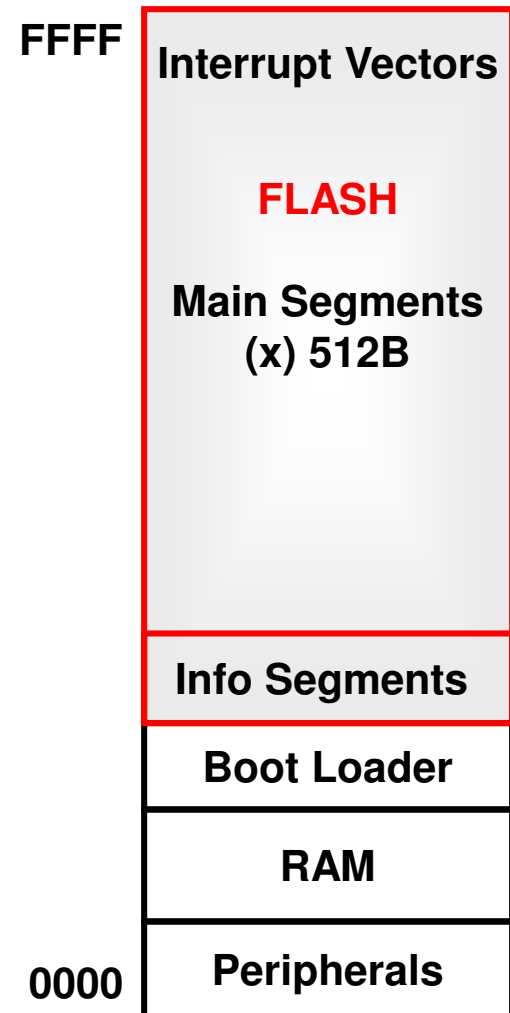
**100% CPU
overhead**

```
for (;;) {
    P1OUT |= 0x04;        // Set
    delay1();
    P1OUT &= ~0x40;       // Reset
    delay2();
}
```

Unified Memory Map

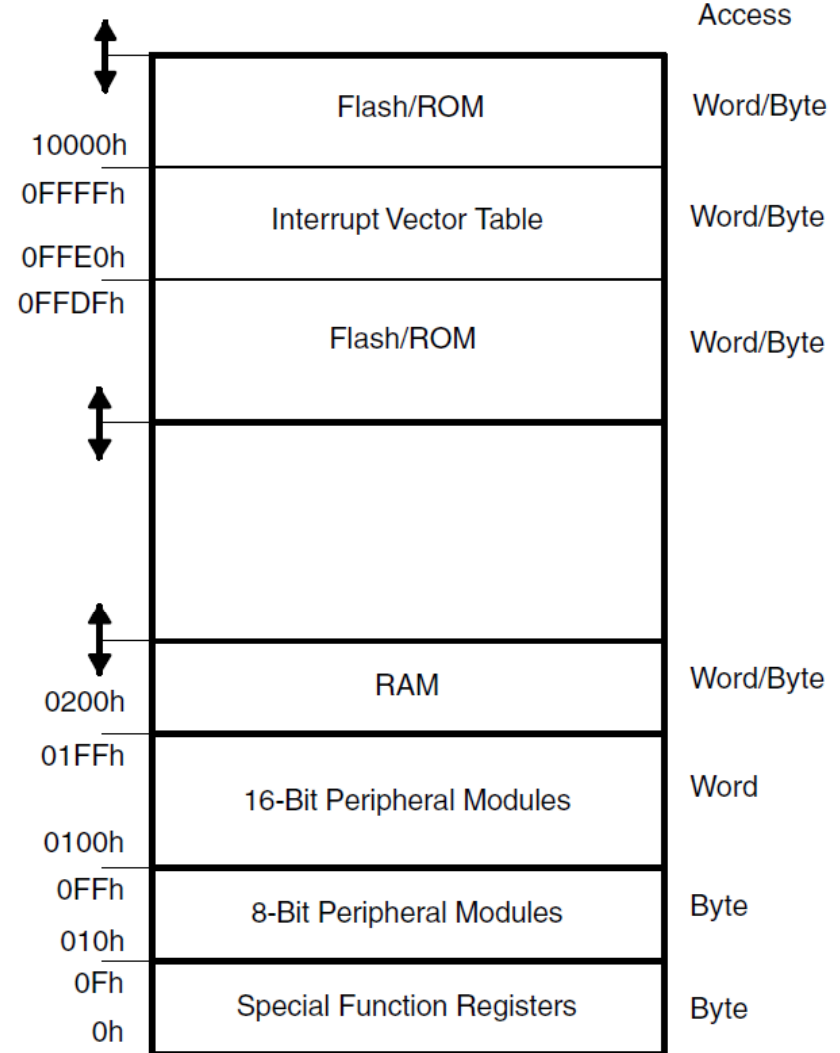
- Absolutely no paging
- Supports code agility
- In System Programmable (ISP) Flash
 - Self programming
 - JTAG
 - Bootloader

```
// Flash In System Programming
FCTL3 = FWKEY;           // Unlock
FCTL1 = FWKEY | WRT;     // Enable
*(unsigned int *)0xFC00 = 0x1234;
```



Memory Map

**Check Device Specific
Datasheets for details**



Clock System (1)

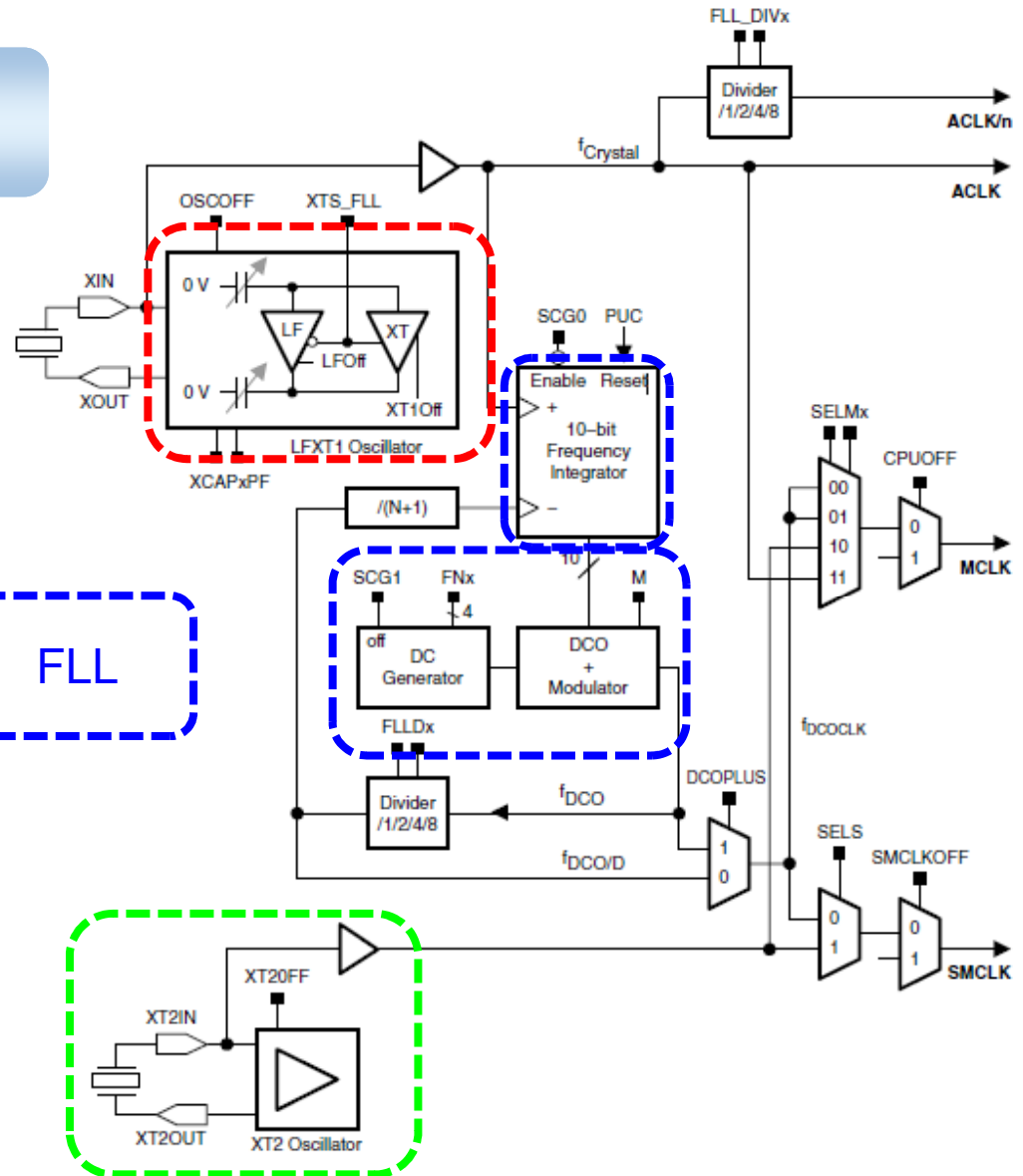
Check Device Specific Datasheets for details

LFXT Oscillator

DCO

FLL

XT2 Oscillator



Clock System (2)

The FLL+ clock module includes two or three clock sources:

- [LFXT1CLK](#): Low-frequency/high-frequency oscillator that can be used either with low-frequency 32768-Hz watch crystals or standard crystals or resonators in the 450-kHz to 8-MHz range. See the device-specific data sheet for details.
- [XT2CLK](#): Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 450-kHz to 16-MHz range.
- [DCOCLK](#): Internal digitally controlled oscillator (DCO) with RC-type characteristics, stabilized by the FLL.
- [VLOCLK](#): Internal very low power, low frequency oscillator with 12-kHz typical frequency.

Clock System (3)

- ACLK: Auxiliary clock. The ACLK is software selectable as LFXT1CLK or VLOCLK as clock source. ACLK is software selectable for individual peripheral modules.
- ACLK/n: Buffered output of the ACLK. The ACLK/n is ACLK divided by 1,2,4, or 8 and used externally only.
- MCLK: Master clock. MCLK is software selectable as LFXT1CLK,
- VLOCLK, XT2CLK (if available), or DCOCLK. MCLK can be divided by 1, 2, 4, or 8 within the FLL block. MCLK is used by the CPU and system.
- SMCLK: Sub-main clock. SMCLK is software selectable as XT2CLK (if available) or DCOCLK. SMCLK is software selectable for individual peripheral modules

Clock System (4)

- Frequency Ranges
- Low Power Modes
- Fault Flags
- Fail Safe Operation

Wrap-Up

Wrap-Up

- MSP430 Architecture
- Instruction Set
- Compiler Friendly Features
- Memory Sub-System
- Clock System

MSP430 Resources

- User's Guides
- Datasheets
- [TI Community Forum](#)
- 100+ Application Reports
- [1000+ Code Examples](#)
- Product Brochure
- [MCU Selection Tool](#)
- Latest Tool Software
- 3rd Party Listing
- Silicon Errata

```
C:\Documents and Settings\d0869271\Local Settings\Temp\wz2370\MCU\CCDev\msp430
File Edit Search View Format Language Settings Macro Run TextFX Plugins
msp430f543x_rtm01_01
// MSP430F543x Demo - Comparator A, Poll input CA0, result in
// Description: The Voltage at pin CA0 (Vcompa) is compared to
// voltage of 0.5*Vcc. LED is toggled when Vcompa crosses the
//
//
// MSP430F543x
//
//
//
//
//
//
// M. Seaman/ P. Thanigai
// Texas Instruments Inc.
// September 2008
// Built with IAR Embedded Workbench
//
#include "msp430x5xx.h"

void main(void)
{
    WDCTL = WDTPW+WDTHOLD;
    P1CTL = XCAP14PF;

    CACTL1 = CAON+CAREF_2+CARSE
    CACTL2 = P0CA0;
    CAPD |= CAPD6;
    P4DIR |= 0x40;
    P2SEL |= BIT6;
    while(1)
    {
        if (CACTL2&0x01)
        {
            P4OUT |= 0x40;
        }
    }
}
```



MSP430F543x, MSP430F541x
MSP430F543xA, MSP430F541xA
MSP430F543x, MSP430F541x

MIXED SIGNAL MICROCONTROLLER

MSP430x5xx Family

User's Guide

MSP430 Ultra-Low-Power Microcontrollers



MCUs around the clock
It's always 430 somewhere

www.ti.com/msp430

2/16/2012

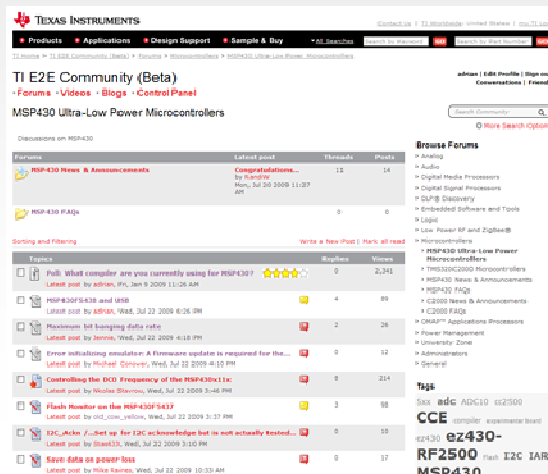
50



Extensive Community Support

E2E Community

- Videos, Blogs, Forums
- Extensive community support and idea exchange
- Global customer support
- <http://e2e.ti.com>

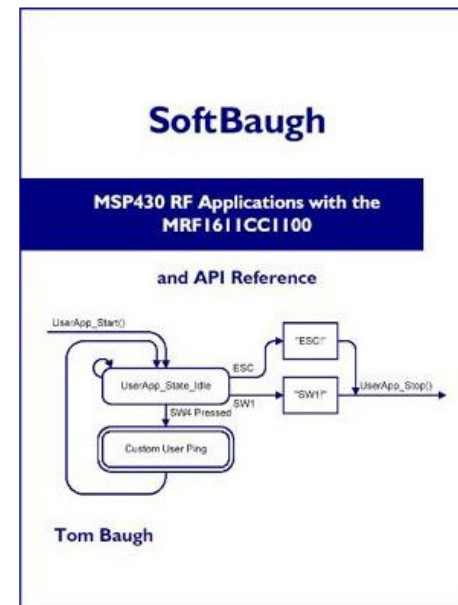
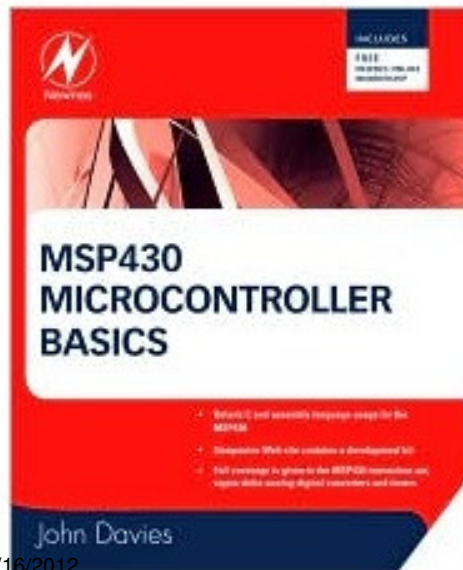
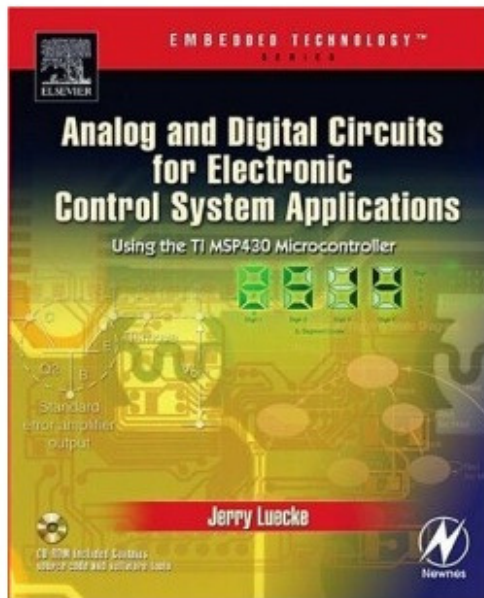


Processor Wiki

- Growing collection of technical wiki articles
- Tips & tricks, common pitfalls, and design ideas
- <http://wiki.msp430.com>



BOOKS



2/16/2012

52

Q & A