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 (**M**ember **G**roup **T**echnical **S**taff)

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

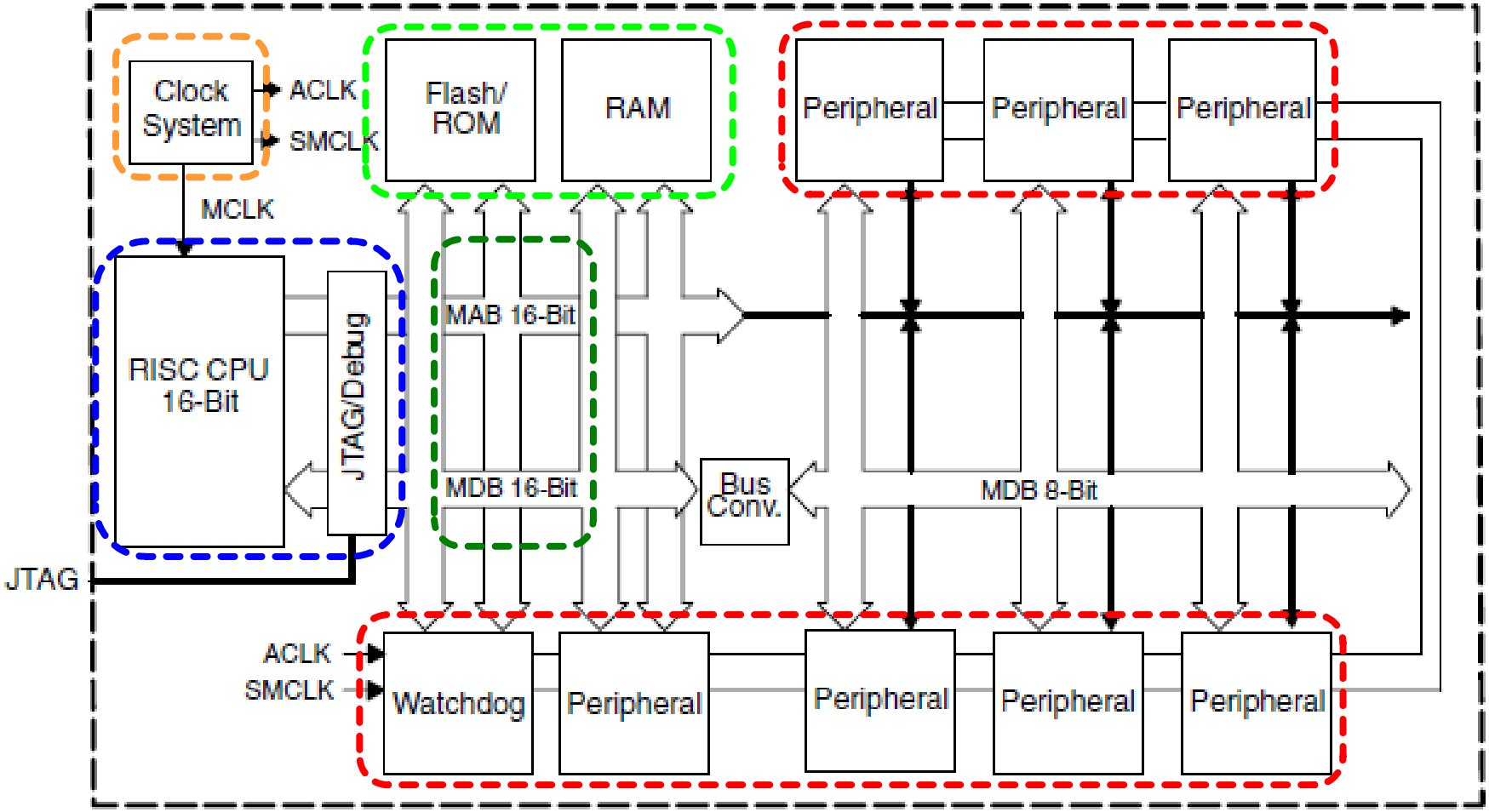
- Embedded systems
- Embedded system design cycle
- Need for Low-Power embedded systems
- Power Aware Architecture
- Power saving techniques
- Choosing a suitable microcontroller

Agenda for this session

Agenda for this session

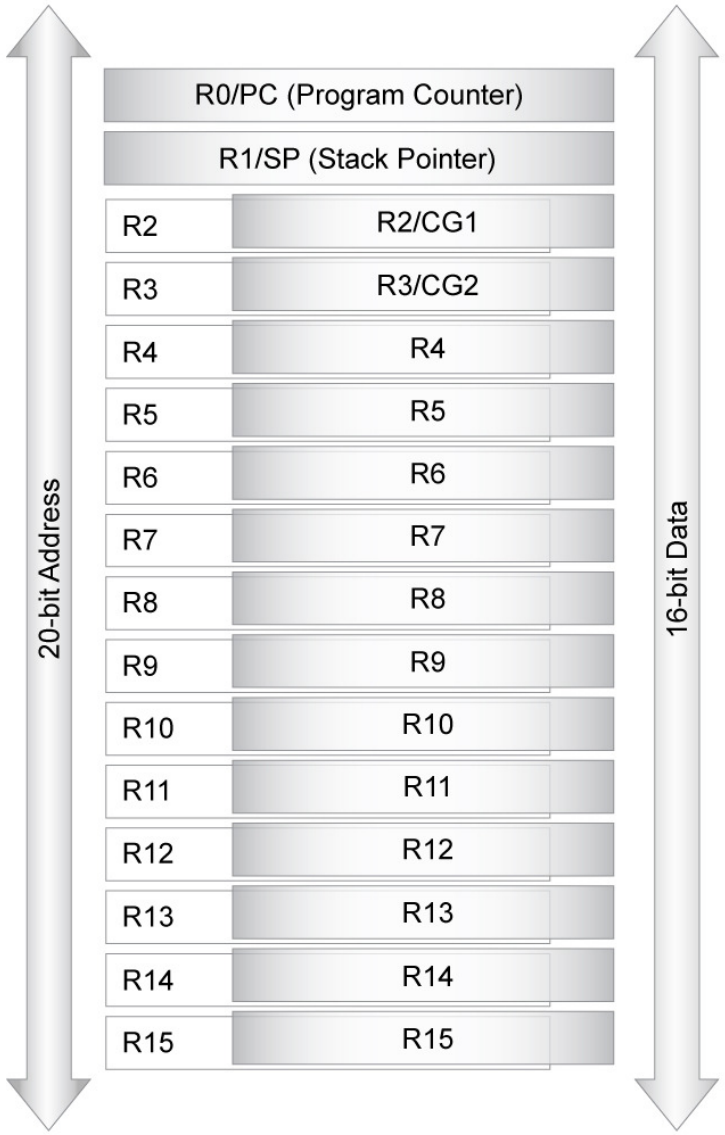
- MSP430 Architecture
- Instruction Set
- Compiler Friendly Features
- Memory Sub-System
- Clock System

MSP430 Architecture

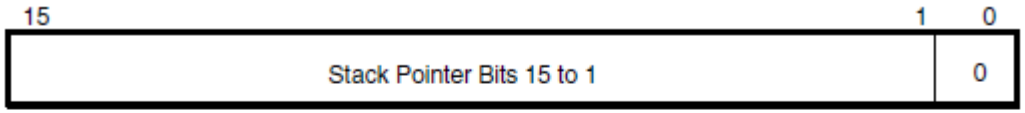


16-bit RISC CPU

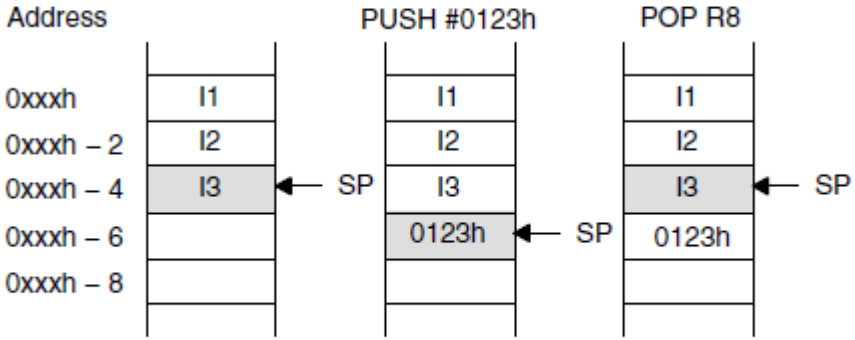
- Efficient, ultra-low power CPU
- C-compiler friendly
- RISC architecture
  - 27 core instructions
  - 24 emulated instructions
  - 7 addressing modes
  - Constant generator
- Single-cycle register operations
- Memory-to-memory atomic addressing
- Bit, byte and word processing
- 20-bit addressing on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



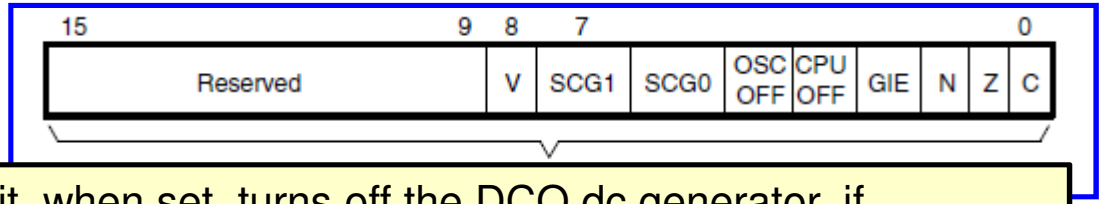
Stack Behavior: Push and Pop




```
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

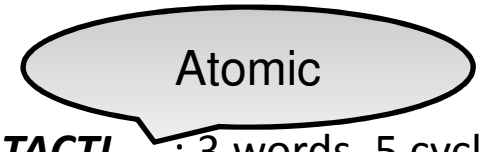
- RISC

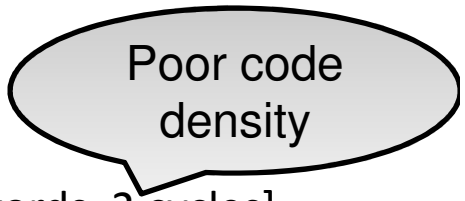
MSP430

  - 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]

- Why not single cycle execution

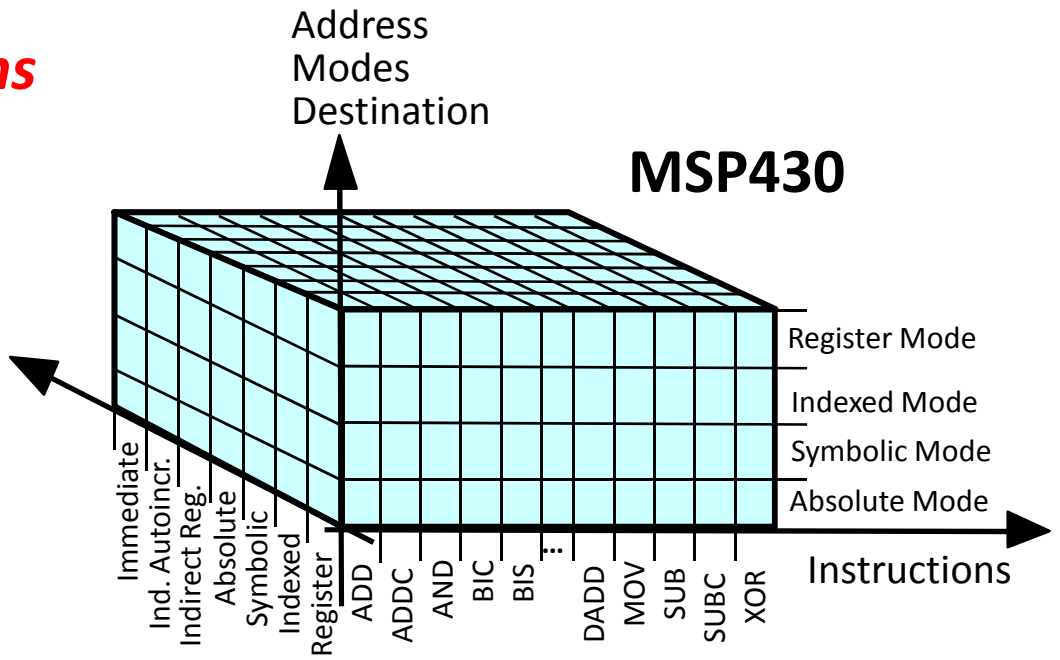
To support orthogonal instruction set

Orthogonal Architecture

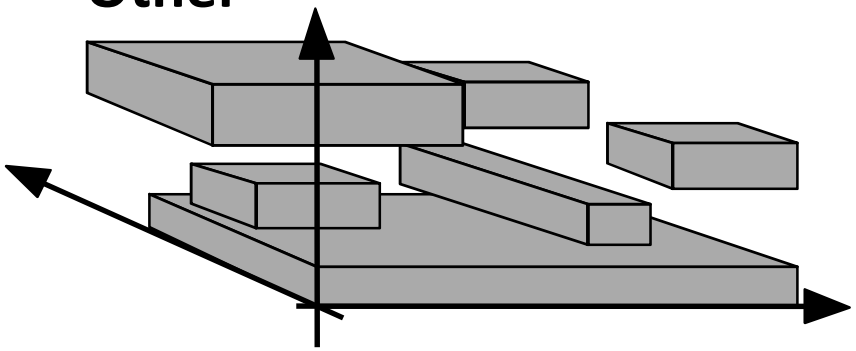
Consistent With No Exceptions

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

Register Mode

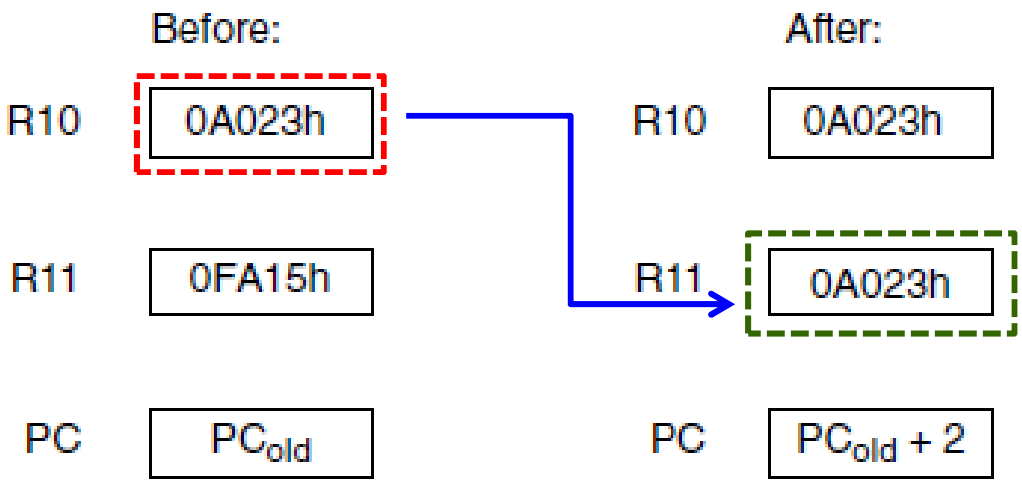
Assembler Code	Content of ROM
`MOV R10,R11`	`MOV R10,R11`

Length: One or two words

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

Comment: Valid for source and destination

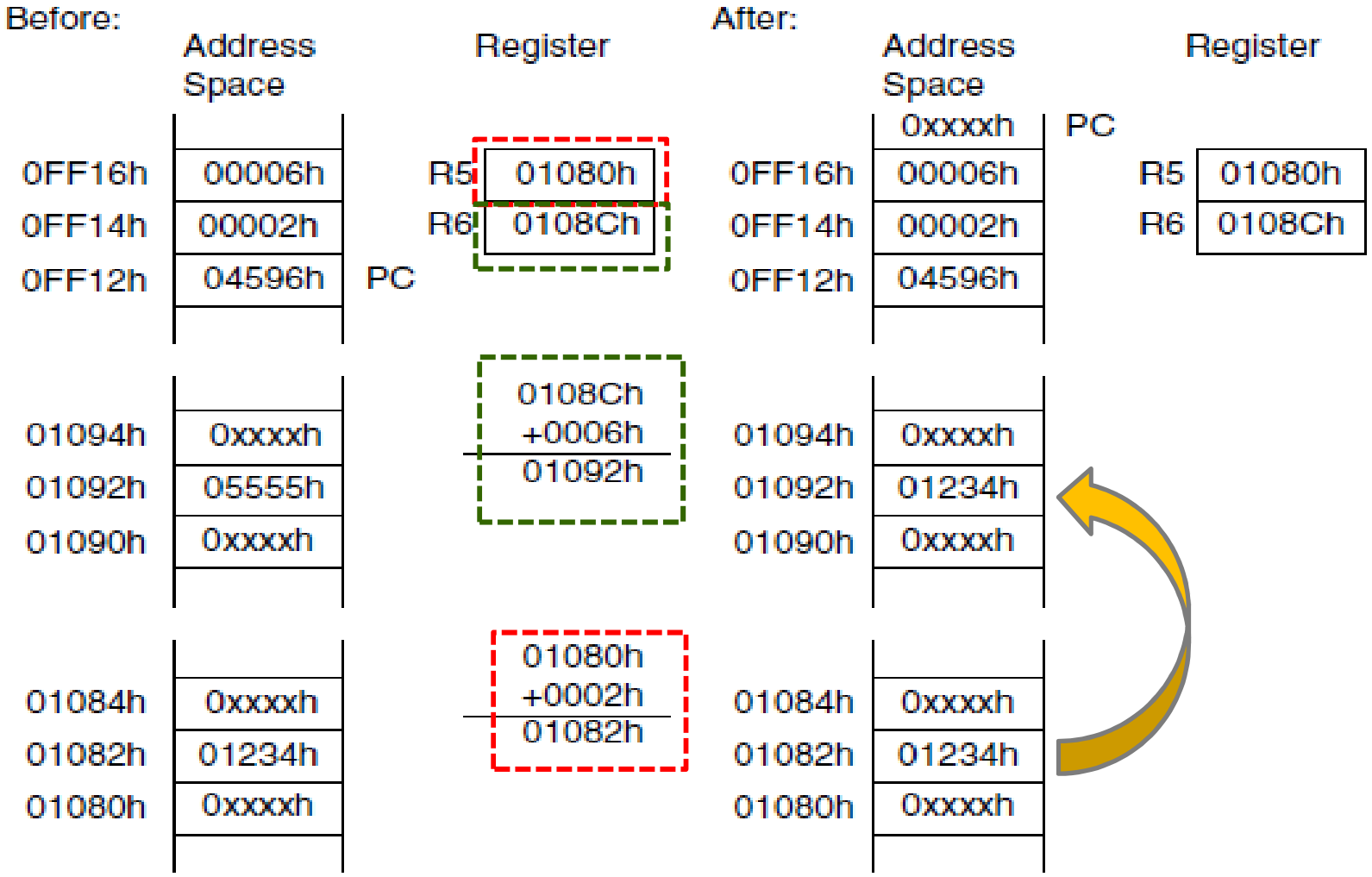
Example: `MOV R10,R11`



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)



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
		PC
0FF16h	011FEh	
0FF14h	0F102h	
0FF12h	04090h	
0F018h	0xxxxh	
0F016h	0A123h	
0F014h	0xxxxh	
01116h	0xxxxh	
01114h	05555h	
01112h	0xxxxh	

0FF14h
+0F102h
0F016h

0FF16h
+011FEh
01114h

After:

Address Space		Register
	0xxxxh	PC
0FF16h	011FEh	
0FF14h	0F102h	
0FF12h	04090h	
0F018h	0xxxxh	
0F016h	0A123h	
0F014h	0xxxxh	
01116h	0xxxxh	
01114h	0A123h	
01112h	0xxxxh	

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
0F018h	0xxxxh	
0F016h	0A123h	
0F014h	0xxxxh	
01116h	0xxxxh	
01114h	01234h	
01112h	0xxxxh	

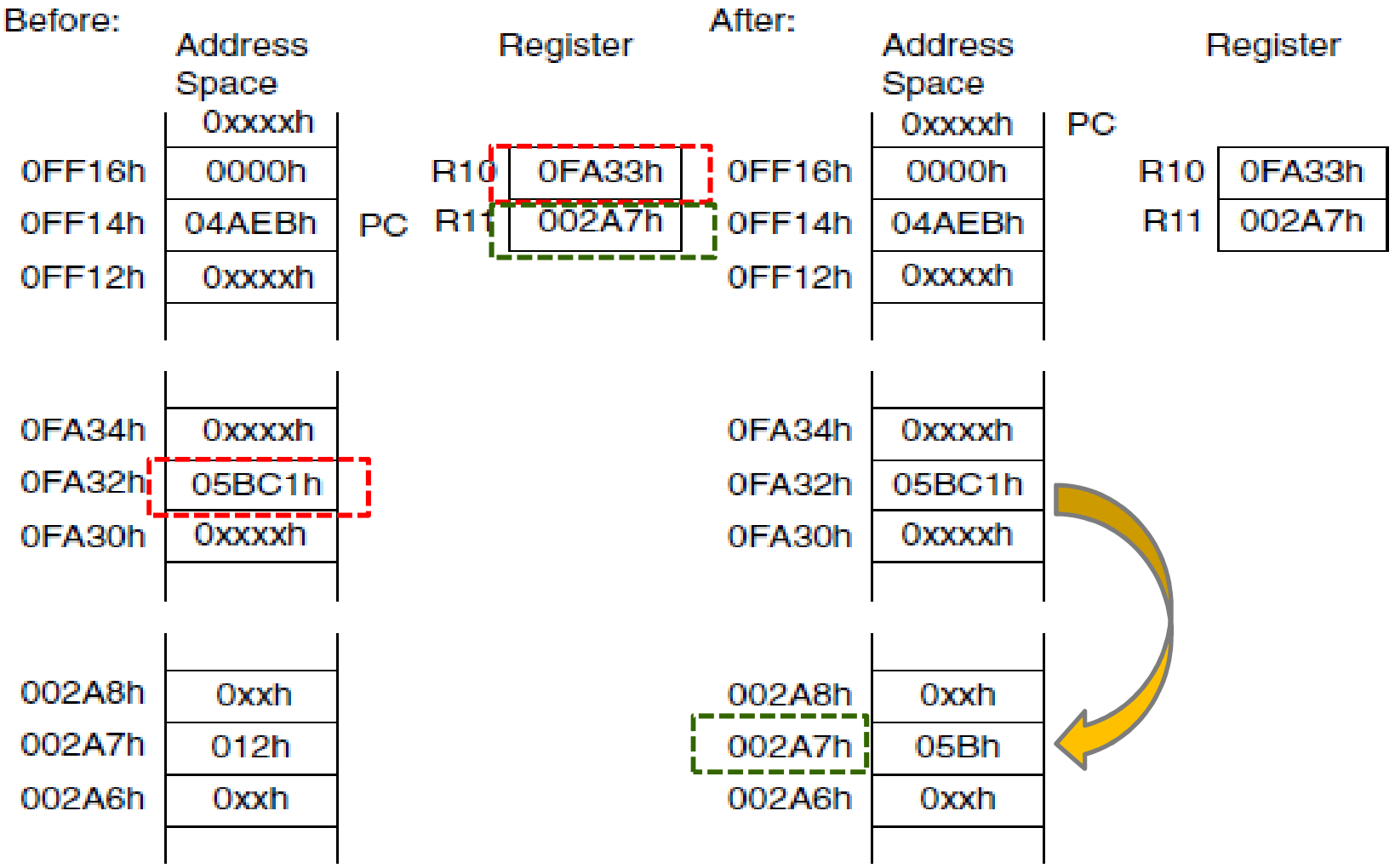
After:

Address Space		Register
	0xxxxh	PC
0FF16h	01114h	
0FF14h	0F016h	
0FF12h	04292h	
0F018h	0xxxxh	
0F016h	0A123h	
0F014h	0xxxxh	
01116h	0xxxxh	
01114h	0A123h	
01112h	0xxxxh	

Indirect Register Mode

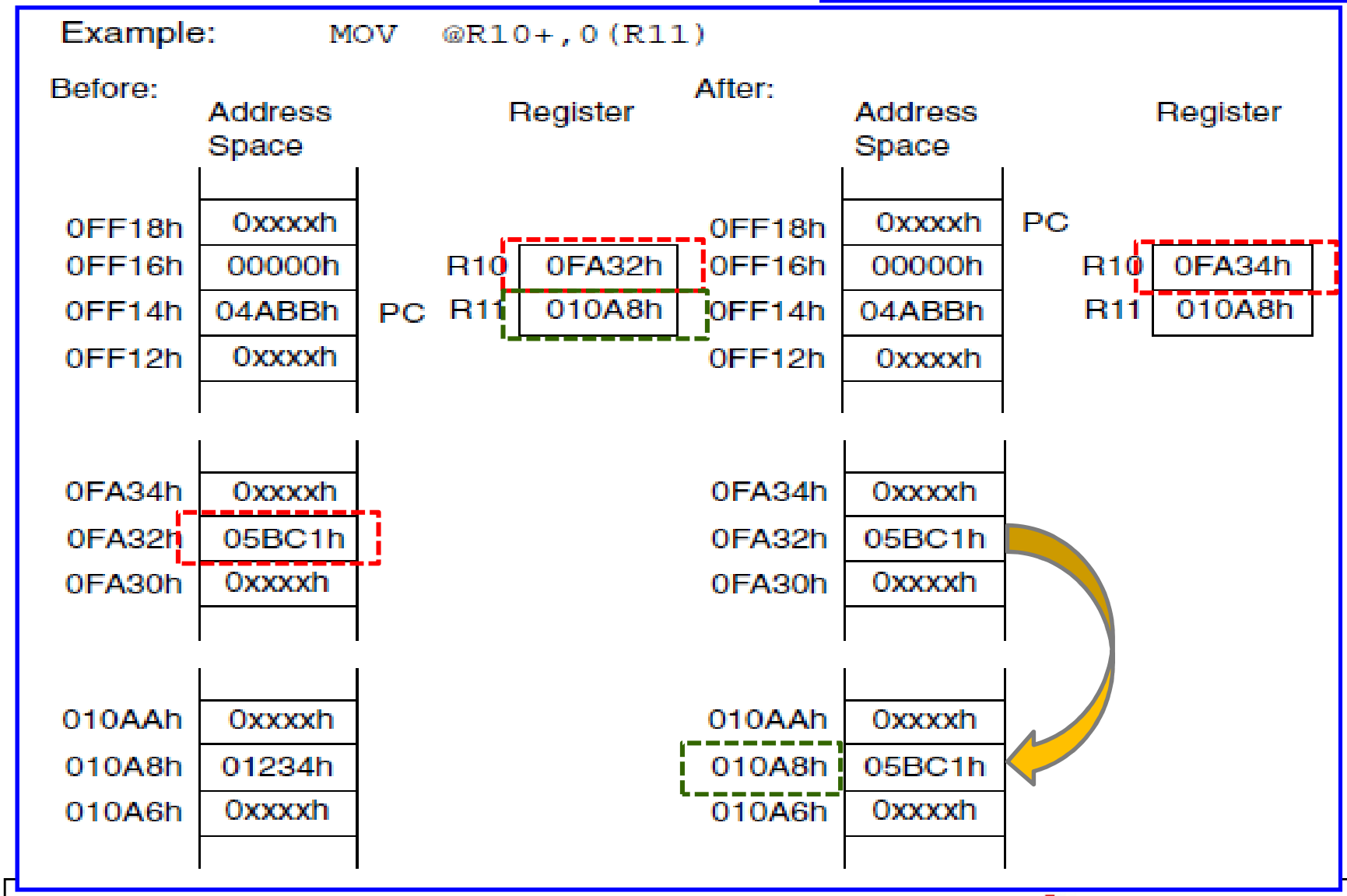
Assembler Code	Content of ROM
MOV @R10,0(R11)	MOV @R10,0(R11)

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



Indirect Auto Increment Mode

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



Immediate Mode

Assembler Code	Content of ROM
MOV #45h,TONI	MOV @PC+,X(PC)
	45
	X = TONI – PC

Example: MOV #45h,TONI

Before:

	Address Space	Register
0FF16h	01192h	
0FF14h	00045h	
0FF12h	040B0h	PC

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

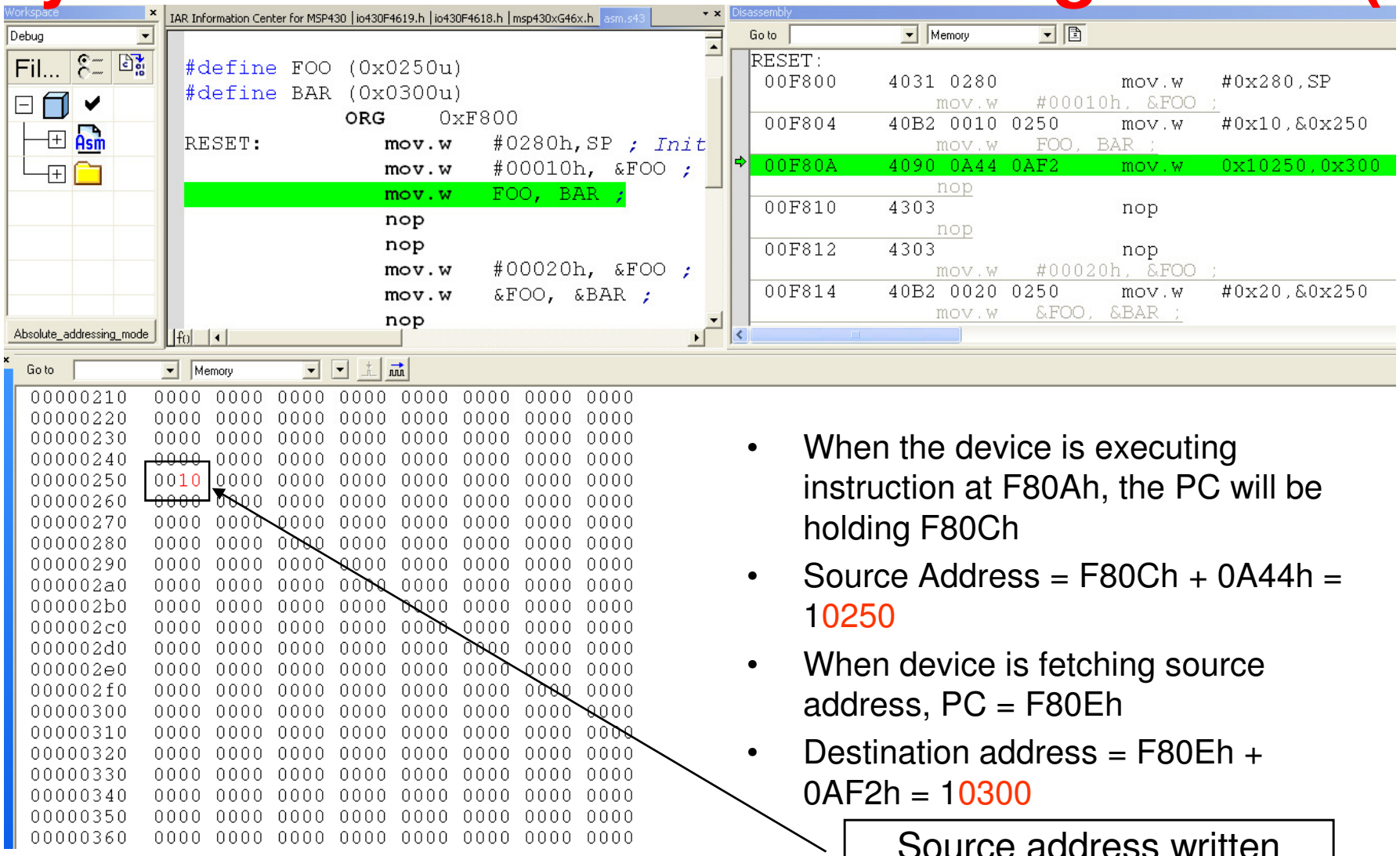
0FF16h
+01192h
010A8h

After:

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

010AAh	0xxxxh
010A8h	00045h
010A6h	0xxxxh

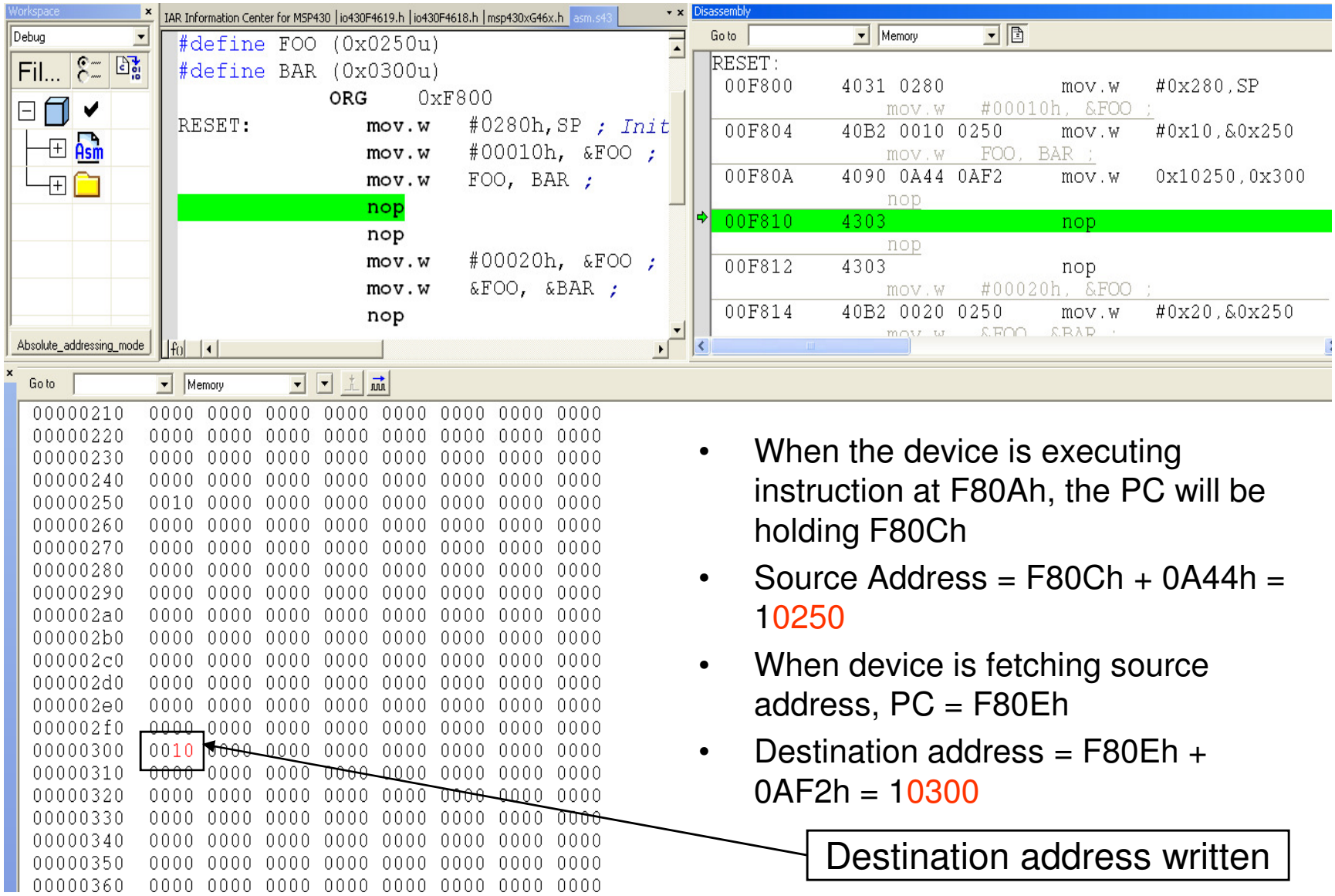
Symbolic/Absolute Addressing Modes(1)



- 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

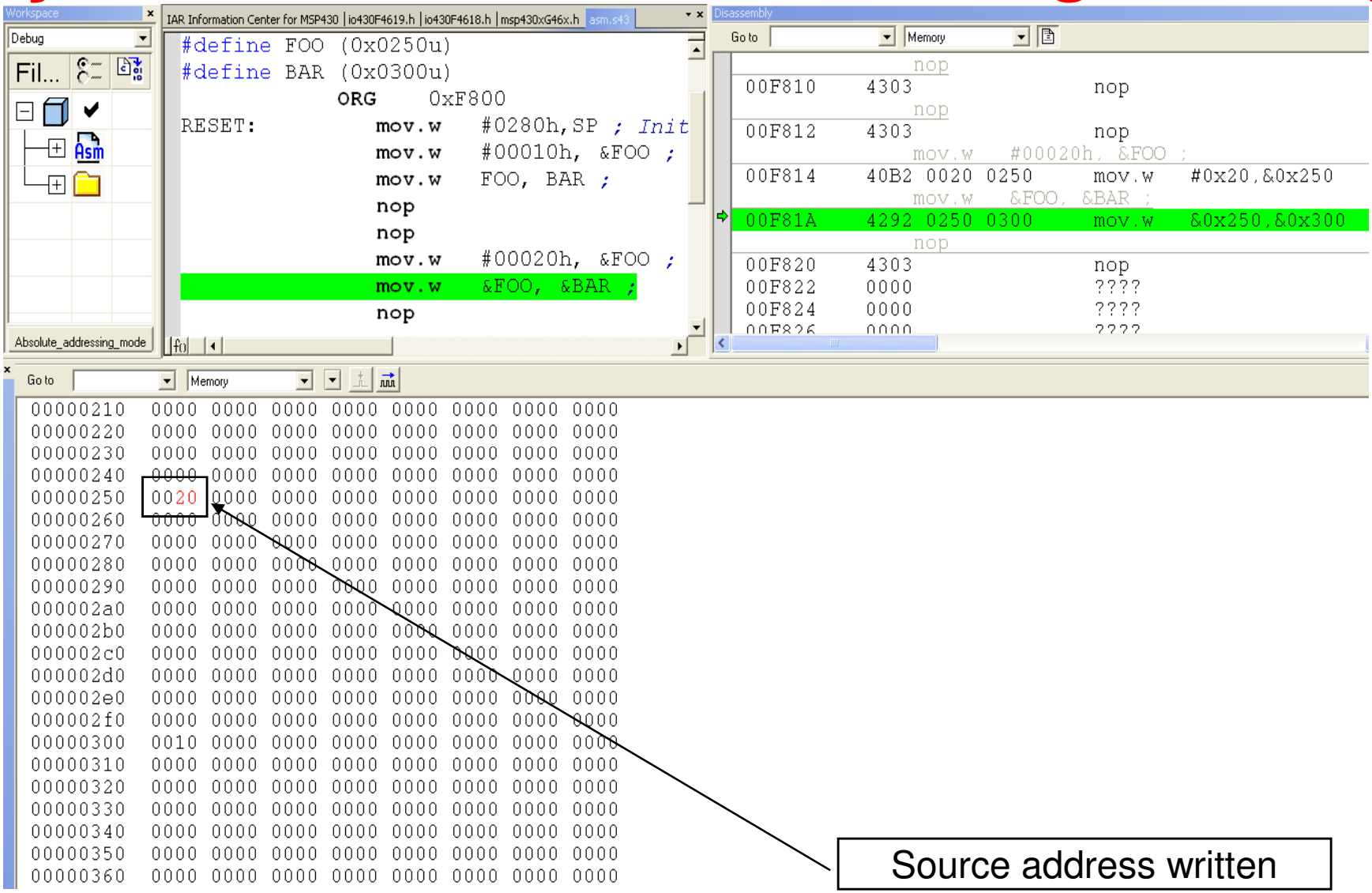
Symbolic/Absolute Addressing Modes(2)



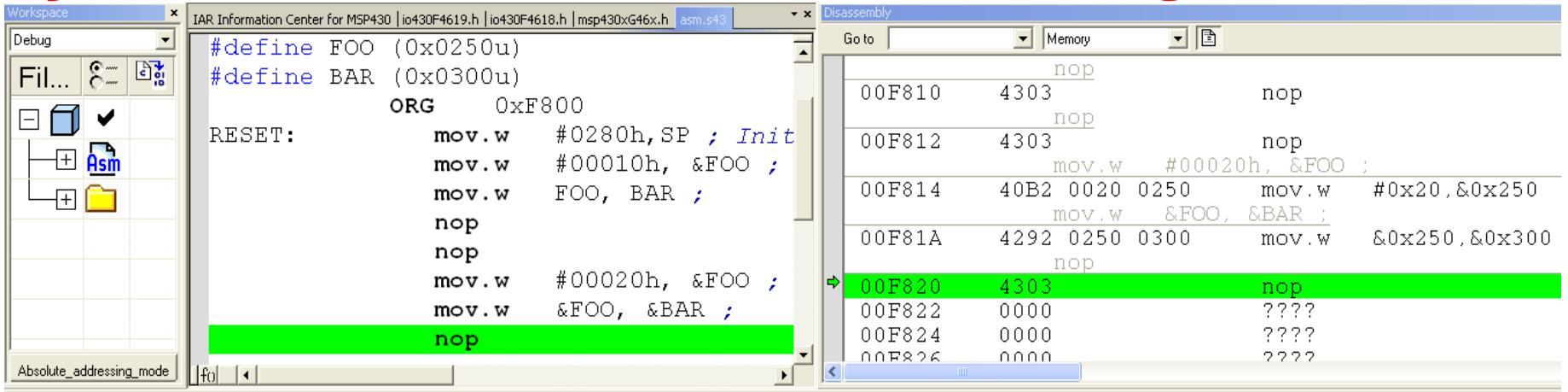
- When the device is executing instruction at F80Ah, the PC will be holding F80Ch
- Source Address = F80Ch + 0A44h = 10250
- When device is fetching source address, PC = F80Eh
- Destination address = F80Eh + 0AF2h = 10300

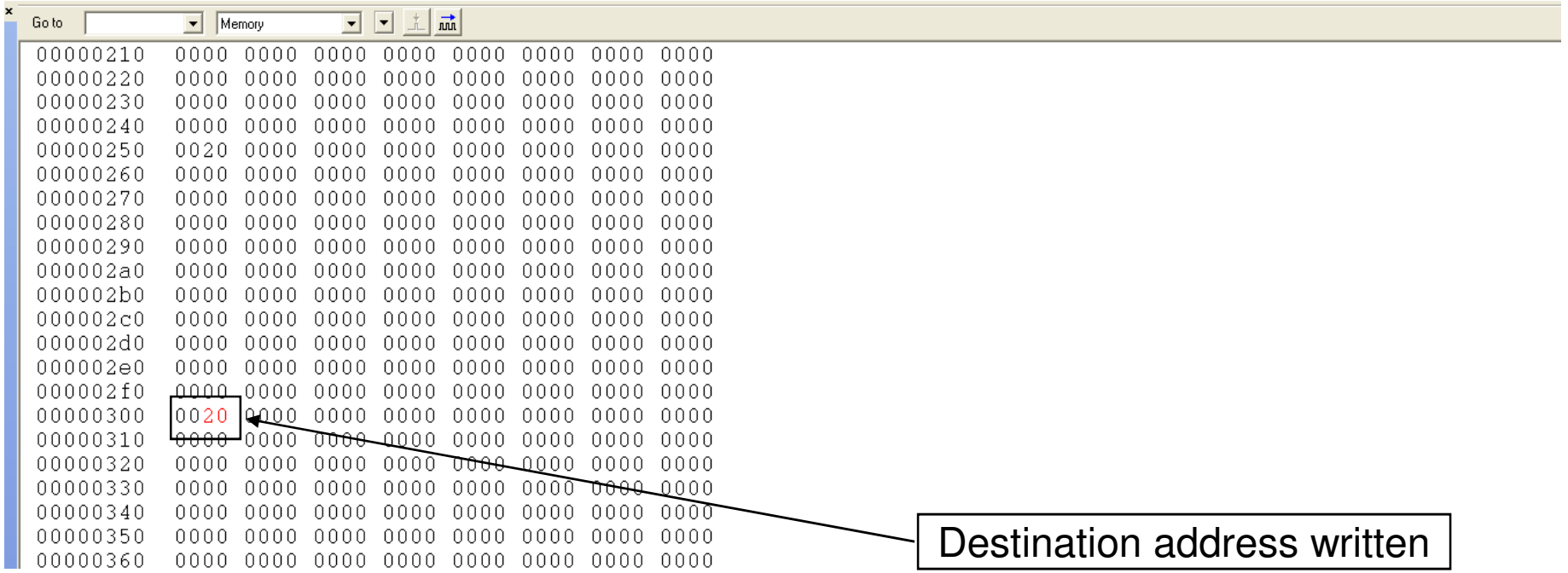
Destination address written

Symbolic/Absolute Addressing Modes(3)



Symbolic/Absolute Addressing Modes(4)

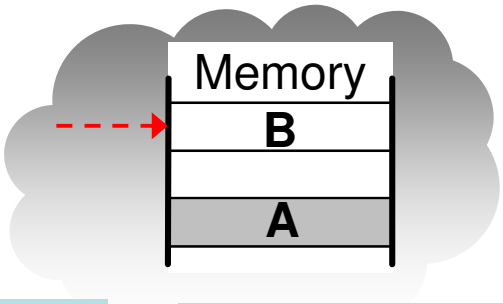




Destination address written

Atomic Addressing

B=B+A

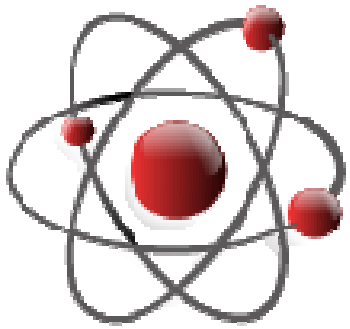



```
; Pure RISC
push    R5
ld      R5,A
add     R5,B
st      B,R5
pop     R5
```

```
; MSP430

add     A,B
```

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

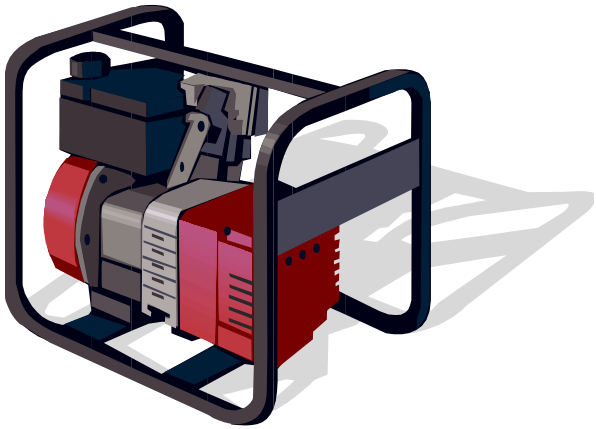


Constant Generator

```
4314            mov.w   #0002h,R4       ; With CG

40341234        mov.w   #1234h,R4       ; Without CG
```

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



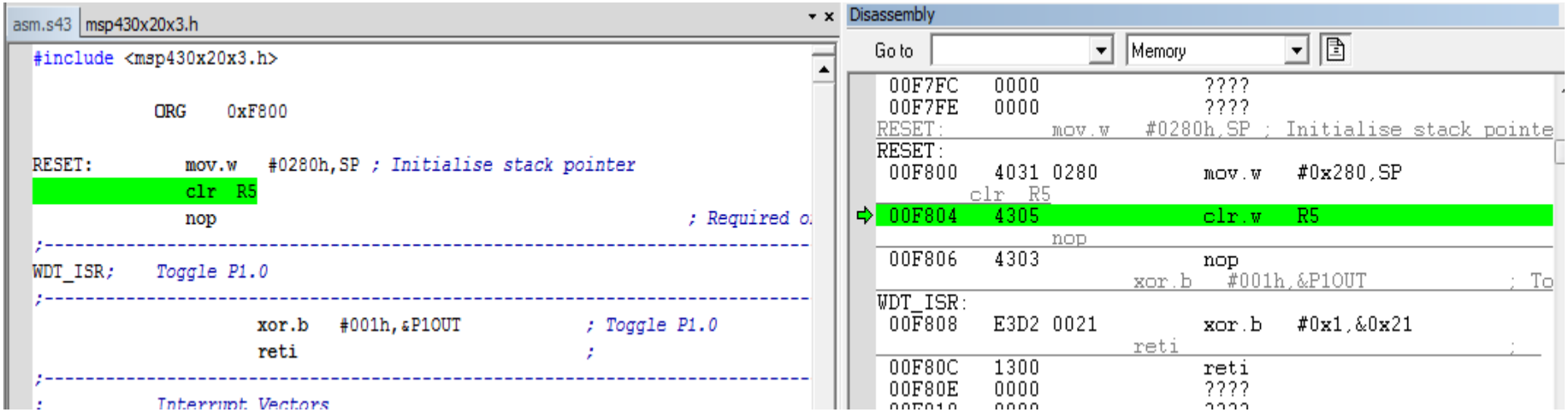
24 Emulated Instructions

```
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



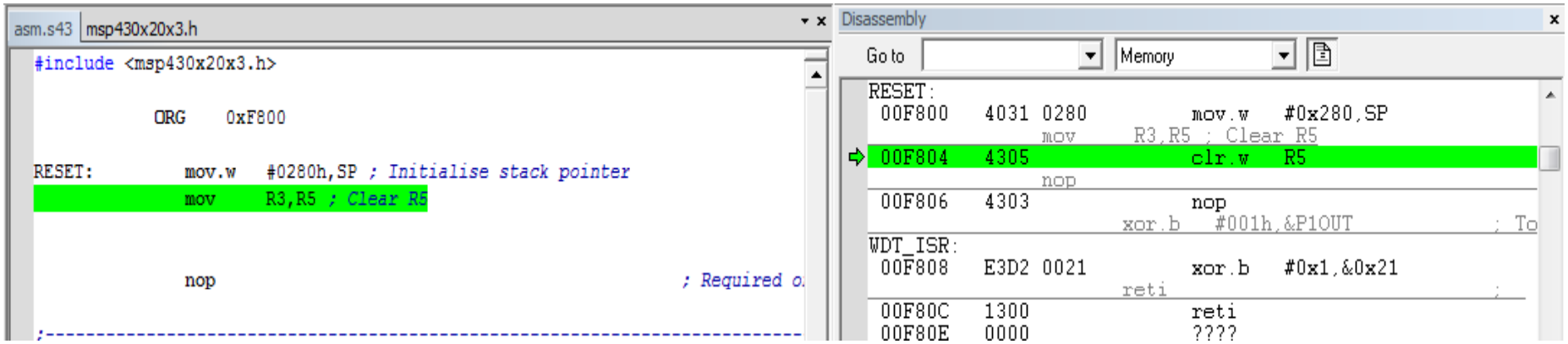
Instruction code: 0x4305

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

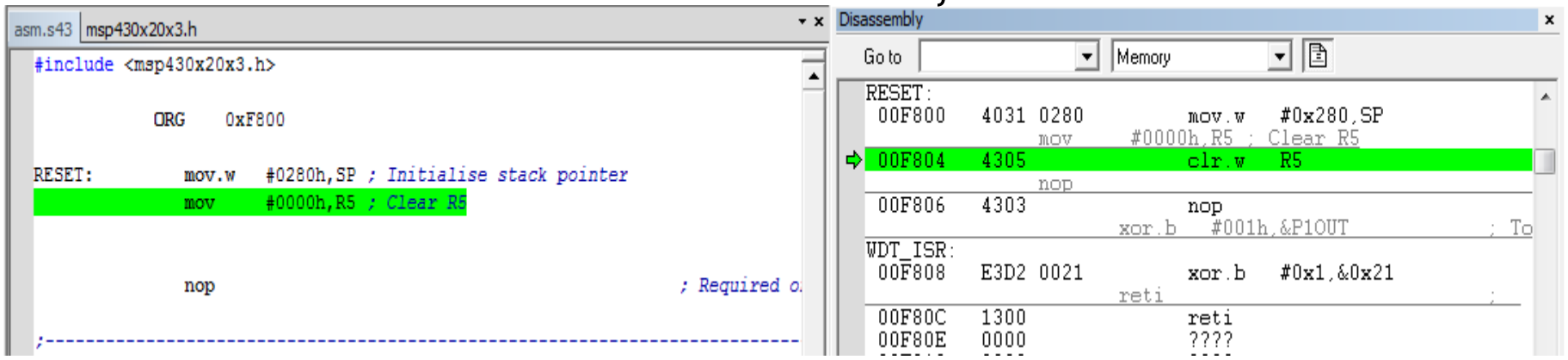
This instruction is equivalent to using the instruction MOV R3, R5 where R3 contains the value #0.

Emulated Instruction(Contd)

MOV R3, R5



MOV #0000h, R5



Three Assembly Instruction Formats

Format I
Source and Destination

```
add.w   R4,R5              ; R4+R5=R5 xxxx
add.b   R4,R5              ; R4+R5=R5 00xx
```

Format II
Destination Only

```
rlc.w   R4
rlc.b   R4
```

Format III
8(Un)conditional Jumps

```
jmp     Loop_1             ; Goto Loop_1
```

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)	jl	**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     0FFFEh              ; MSP430 RESET Vector
        DW      RESET               ;
        ORG     0FFF4h              ; WDT Vector
        DW      WDT_ISR             ;
        END
```

Using C

```
#include <msp430f2013.h>

void main (void)
{
//    WDTCTL = WDTPW | WDTTMSEL | WDTCNTL ; // 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()

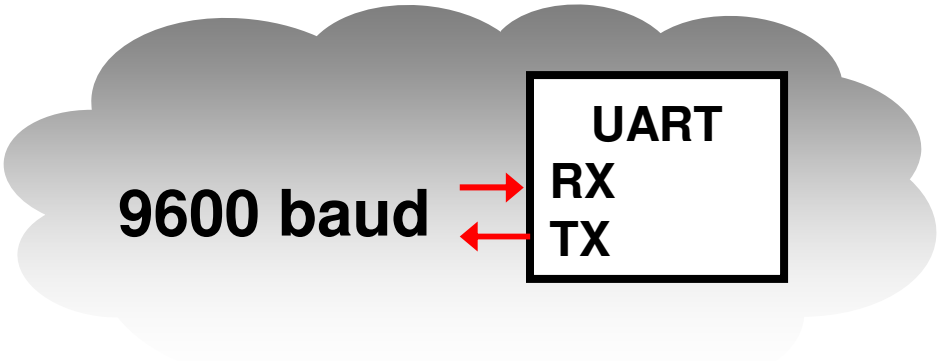
Mixing C and Assembly

- Check first to see whether an intrinsic function is available (see intrinsics.h) Ex : *__swap_bytes( )* calls the swpb instruction

- 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

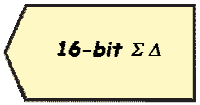
BasicTimer1

USART

USCI

USI

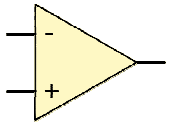
Comparator_A



ΣΔ16

ADC10/12

DAC12

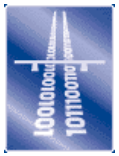


OP-Amps

Scan IF

Hardware Multiplier

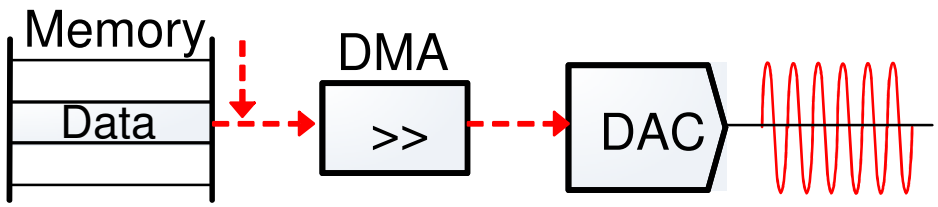
LCD



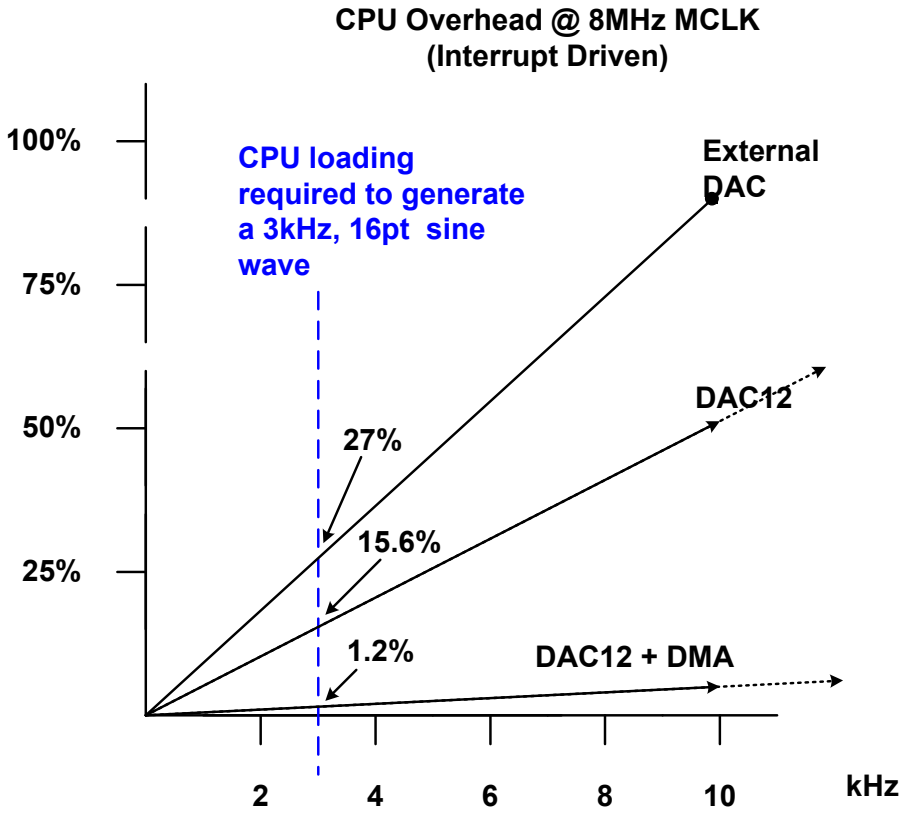
DMA

EEM

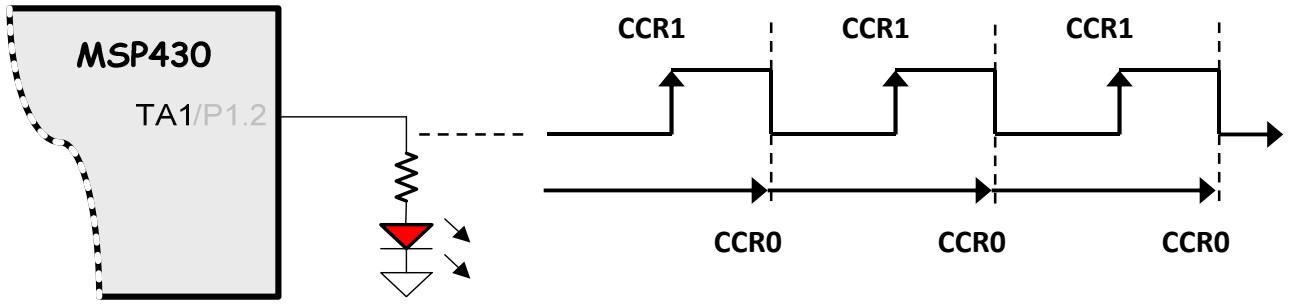
Intelligent Peripheral Performance



- 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
```

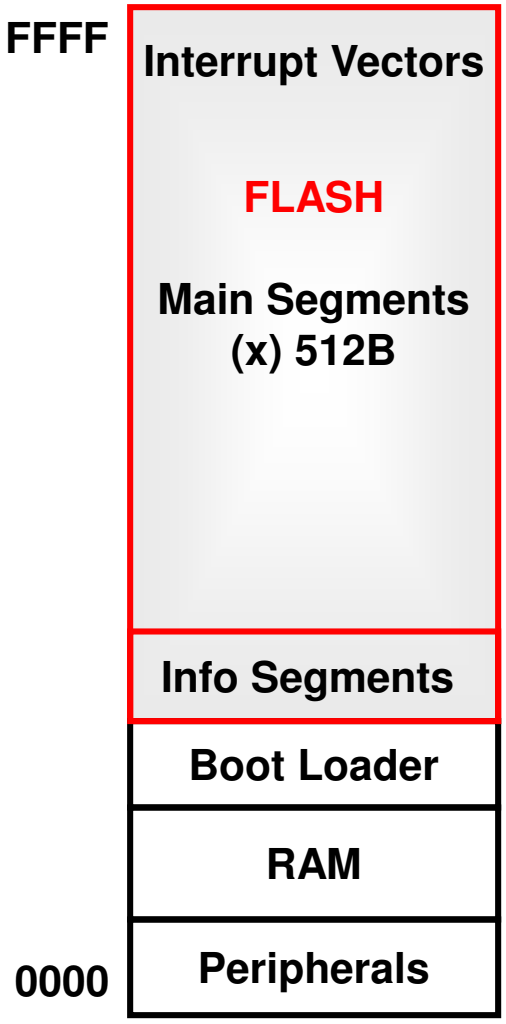
100% CPU overhead

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

Unified Memory Map

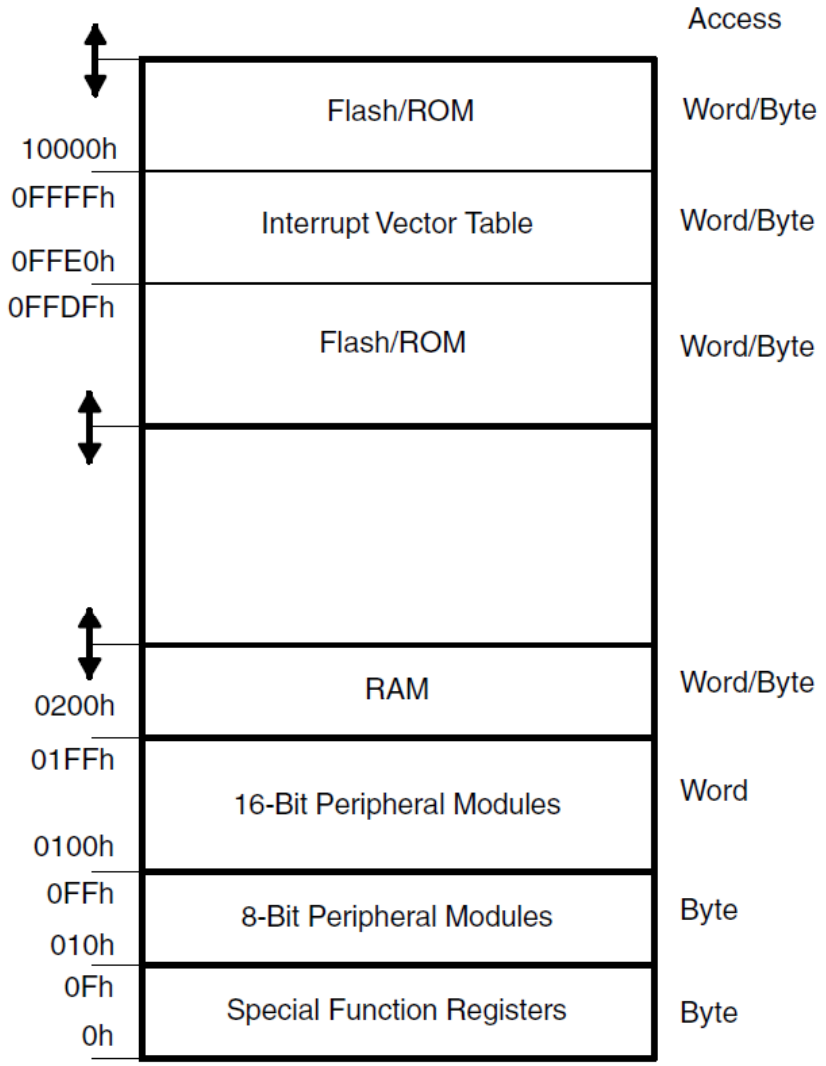
- Absolutely no paging
- Supports code agility
- In System Programmable (ISP) Flash
  - Self programming
  - JTAG
  - Bootloader

```
// 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)

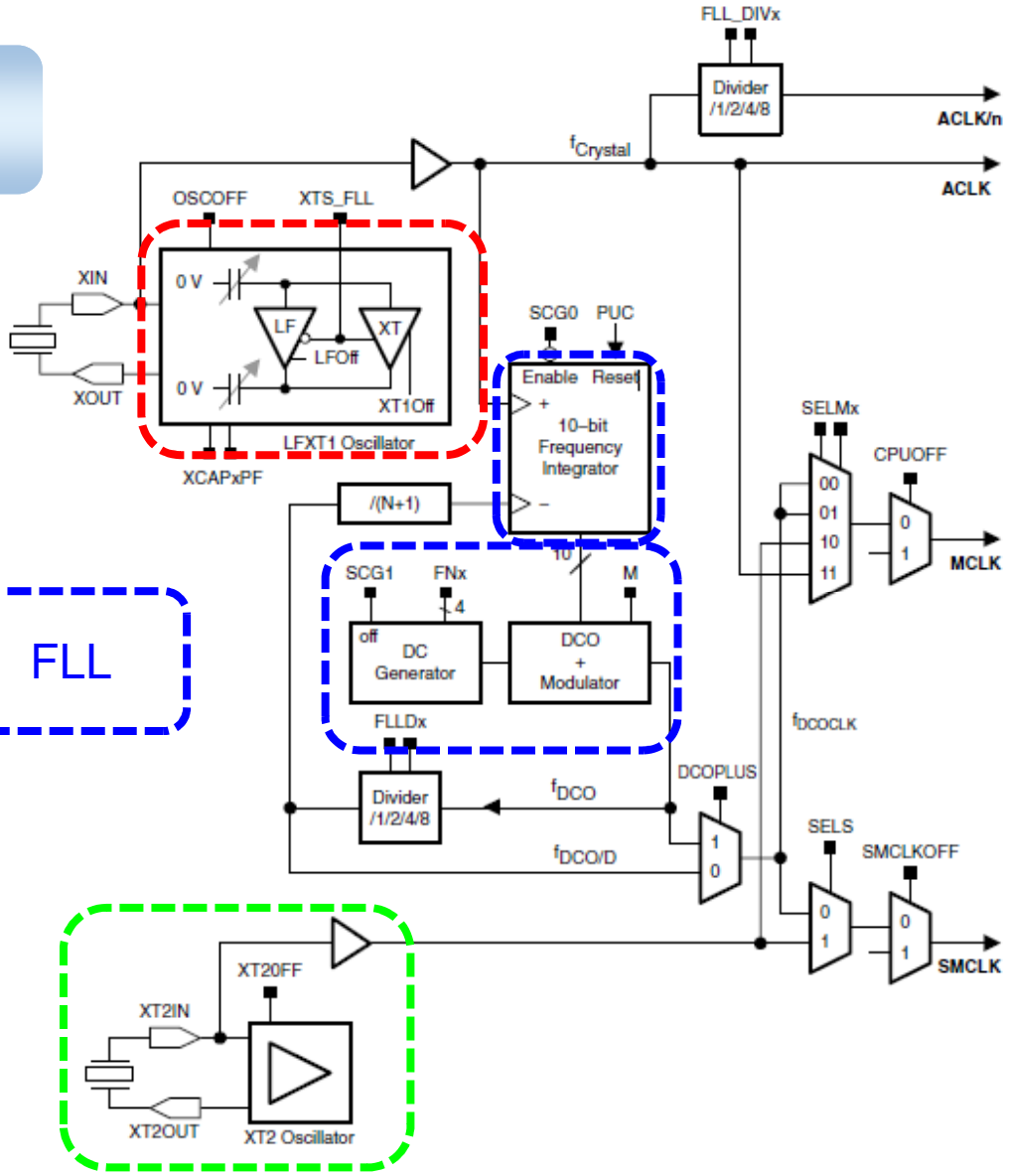
Check Device Specific Datasheets for details

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

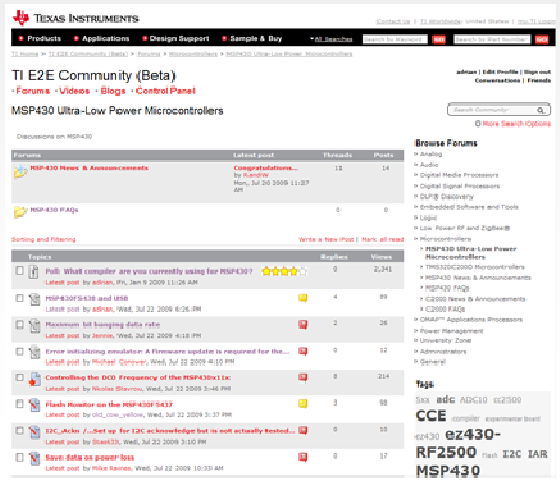
- User's Guides
- Datasheets
- TI Community Forum
- 100+ Application Reports
- 1000+ Code Examples
- Product Brochure
- MCU Selection Tool
- Latest Tool Software
- 3rd Party Listing
- Silicon Errata

www.ti.com/msp430

Extensive Community Support

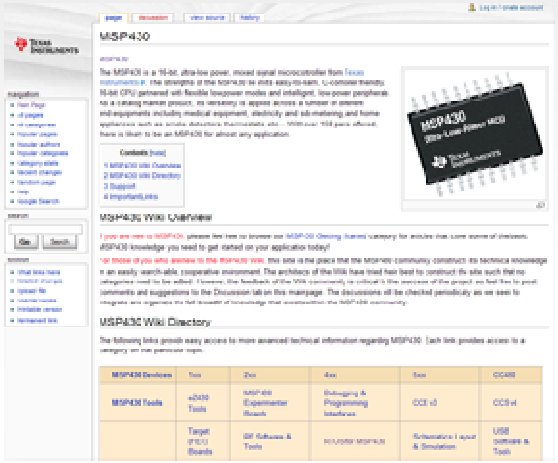
E2E Community

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

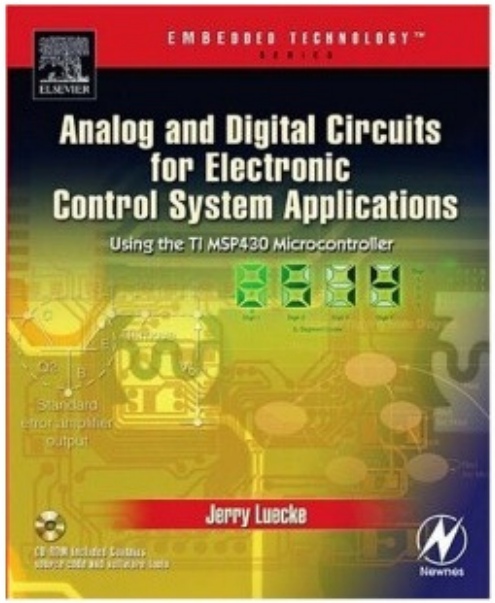


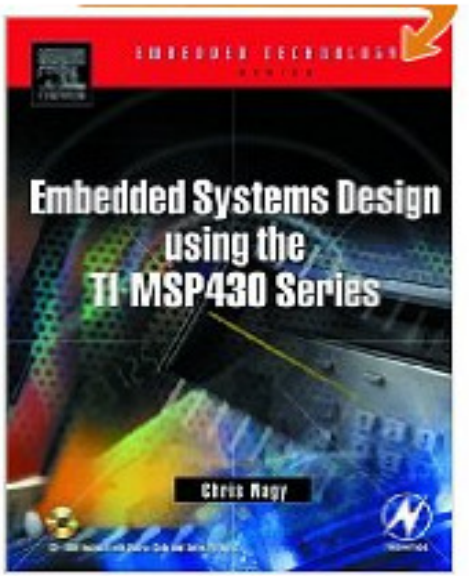
Processor Wiki

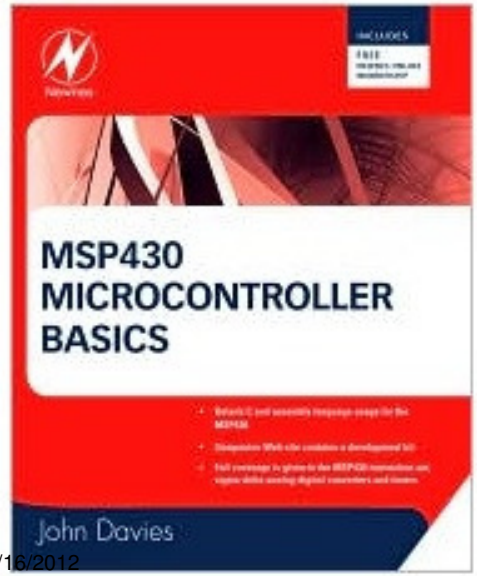
- Growing collection of technical wiki articles
- Tips & tricks, common pitfalls, and design ideas
- http://wiki.msp430.com

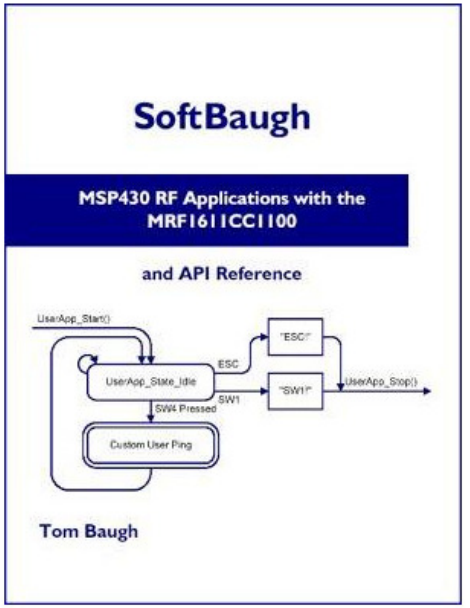


BOOKS


EMBEDDED TECHNOLOGY™
Analog and Digital Circuits for Electronic Control System Applications
Using the TI MSP430 Microcontroller
Jerry Luecke
Newnes


Embedded Systems Design using the TI MSP430 Series
Chris Nagy


MSP430 MICROCONTROLLER BASICS
John Davies


SoftBaugh
MSP430 RF Applications with the MRF1611CC1100
and API Reference
UserApp_State_Idle
ESC
"ESC!"
SW1
"SW1!"
SW4 Pressed
Custom User Ping
UserApp_Stop()
Tom Baugh

Q & A