MICROPROCESSADORES E MICROCONTROLADORES

FUNÇÕES E SUBROTINAS

Crie programas em módulos - são mais fáceis de escrever, entender, debugar e manter

FUNÇÕES E SUBROTINAS

Crie programas em módulos - são mais fáceis de escrever, entender, debugar e manter

O custo de chamar uma função (operações de jump, armazenamento de registradores com o stack pointer etc.) é geralmente negligível

FUNÇÕES E SUBROTINAS

Crie programas em módulos - são mais fáceis de escrever, entender, debugar e manter

O custo de chamar uma função (operações de jump, armazenamento de registradores com o stack pointer etc.) é geralmente negligível

Funções liberam espaço em RAM de variáveis locais

FUNÇÕES E SUBROTINAS

Crie programas [illegible] mais fáceis de escrever, entend[illegible]ter

O custo de cha[illegible]operações de jump, armazena[illegible]dores com o stack pointer et[illegible]egligível

Funções liberam espaço em RAM de variáveis locais

Não se esqueça de documentar suas funções, para uso posterior!!!

FUNÇÕES E SUBROTINAS

Convenções de uso dos registradores (mspgcc):

Argumentos são alocados da esquerda para a direita, de R15 a R12

FUNÇÕES E SUBROTINAS

Convenções de uso dos registradores (mspgcc):

Argumentos são alocados da esquerda para a direita, de R15 a R12

Para mais de quatro parâmetros, eles são passados pela pilha, comprometendo a performance do código

FUNÇÕES E SUBROTINAS

Convenções de uso dos registradores (mspgcc):

Valor de retorno:

- Tipos char, int e ponteiro para função - R15
- Tipos long e float - R15:R14
- long long - R15:R14:R13:R12
- Resultados com mais de 64 bits - memória

FUNÇÕES E SUBROTINAS

Convenções de uso dos registradores (mspgcc):

Os registradores R0 a R5 não são geralmente utilizados pelo compilador

R12 a R15 não precisam ser guardados na pilha antes de serem usados, a não ser em interrupções

R6 a R11 devem ser sempre guardados

FUNÇÕES E SUBROTINAS

Convenções de uso dos registradores (mspgcc):

Registradores são alocados na ordem {R12-R15, R11-R0}

char, int e ponteiros ocupam um registrador, long e float, dois registradores em ordem Little Endian, e long long, quatro registradores em ordem Little Endian

FUNÇÕES E SUBROTINAS

Convenções de uso dos registradores (mspgcc):

Registradores {R12-R15, R11-R0}

char, int e p ador, long e float, dois registradores em ordem Little Endian, e long long, quatro registradores em ordem Little Endian

ATENÇÃO: outros compiladores seguem convenções diferentes de uso de registradores!!!

FUNÇÕES E SUBROTINAS

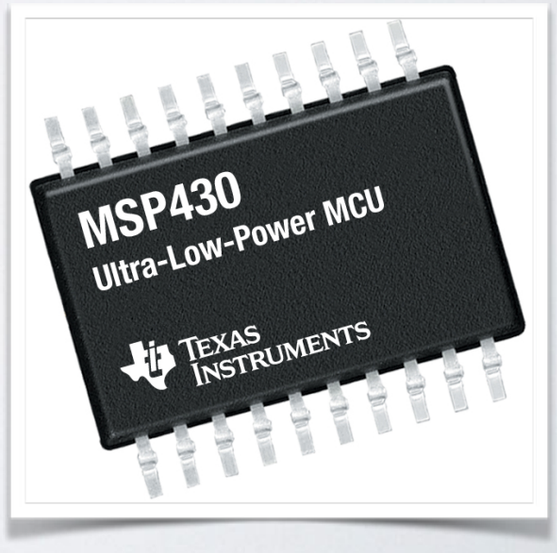
Alocação de variáveis locais

- Registradores
- Posições em RAM
- Pilha

MISTURANDO C E ASSEMBLY

Para acelerar o desenvolvimento:

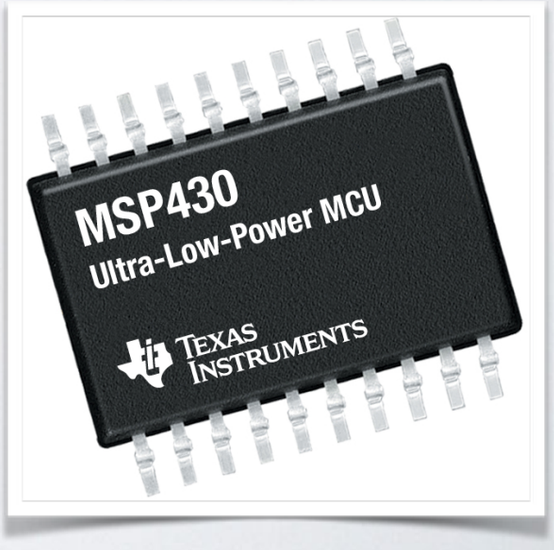
- Código principal em C
- Otimizações em Assembly



MISTURANDO C E ASSEMBLY

1. Procure funções intrínsecas já prontas em <intrinsics.h>

==> _ _swap_bytes()

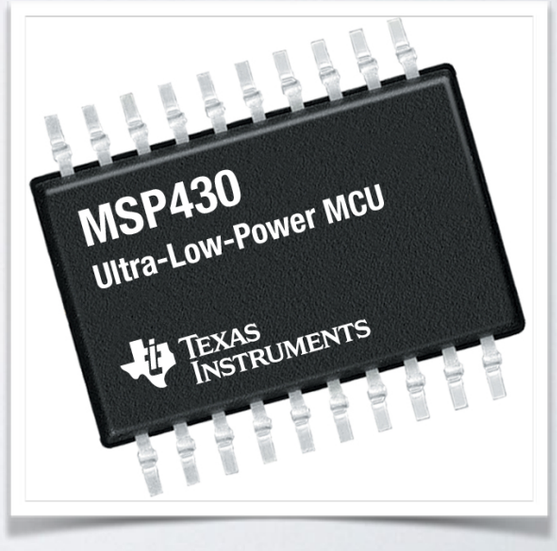


MISTURANDO C E ASSEMBLY

2. Assembly inline

==> asm("mov.b &P1IN,&dest")

CUIDADO aonde você mexe!!!

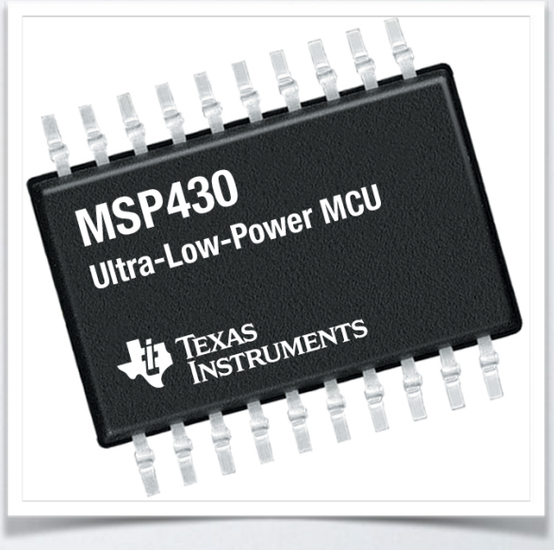


MISTURANDO C E ASSEMBLY

3. Escreva uma subrotina completa e faça a chamada em C

==> Dica: escreva a subrotina em C, compile o código, veja o código Assembly resultante e otimize a partir daí.

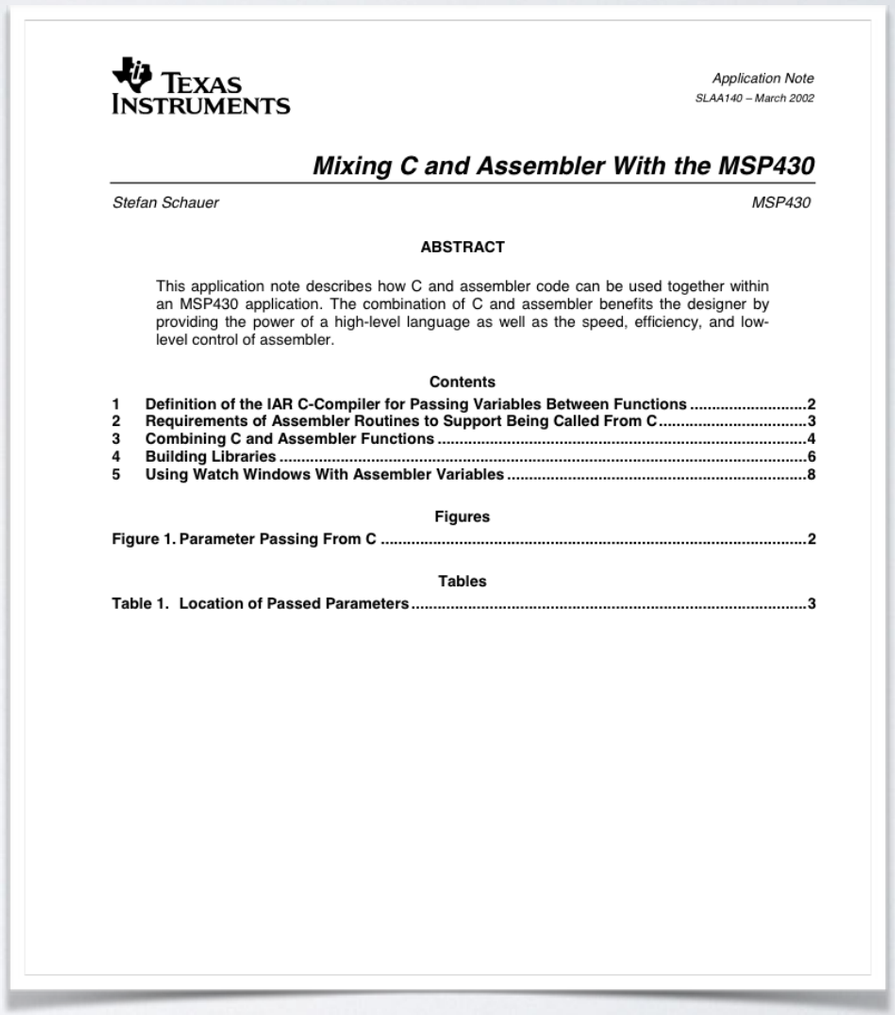
CUIDADO aonde você mexe!!!



MISTURANDO C E ASSEMBLY

Dicas no texto *Mixing C and Assembler with the MSP430*

(disponível online como slaa140.pdf)



TEXAS INSTRUMENTS

Application Note
SLAA140 – March 2002

Mixing C and Assembler With the MSP430

Stefan Schauer *MSP430*

ABSTRACT

This application note describes how C and assembler code can be used together within an MSP430 application. The combination of C and assembler benefits the designer by providing the power of a high-level language as well as the speed, efficiency, and low-level control of assembler.

Contents

Figures

Tables

INTERRUPÇÕES

Funções chamadas por hardware

Chamada imprevisível para a CPU

Pode ser previsível para o programador (você)

INTERRUPÇÕES

Tarefas mais importantes do que o código principal

Tarefas pouco frequentes (evitar pooling)

"Acordar" a CPU

Chamadas a sistemas operacionais

INTERRUPÇÕES

Interrupt Service Routine (ISR)

Por ser imprevisível para a CPU, essa função não pode deixar estragos

Chamada por periféricos e pelo próprio núcleo do MCU (gerador de clock, p. ex.)

INTERRUPÇÕES

Cada ISR tem um flag de lógica positiva

==> Timer_A seta TAIFG no reg. TACTL quando TAR=0. Se TAIE=1, a ISR correspondente é chamada.

INTERRUPÇÕES

Interrupções podem ser mascaráveis: só funcionam quando GIE=1 no SR.

As que não dependem de GIE (não-mascaráveis) também precisam ser habilitadas em registradores

INTERRUPÇÕES

Cada ISR tem um endereço específico, guardado no *vector table* do MSP430

Algumas fontes de interrupção compartilham o endereço

==> TAIFG compartilha o endereço com as interrupções de captura/comparação de canais do Timer_A

INTERRUPÇÕES

ISRs têm prioridades de execução,
para evitar conflitos

Prioridades fixas em hardware,
proporcionais à altura do endereço
(imutáveis)

INTERRUPÇÕES

Table 5. Interrupt Sources, Flags, and Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range(1)	PORIFG RSTIFG WDTIFG KEYV(2)	Reset	0FFFEh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG(2)(3)	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	30
Timer1_A3	TA1CCR0 CCIFG(4)	maskable	0FFFAh	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG(2)(4)	maskable	0FFF8h	28
Comparator_A+	CAIFG(4)	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer0_A3	TA0CCR0 CCIFG(4)	maskable	0FFF2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG (5)(4)	maskable	0FFF0h	24
USCI_A0/USCI_B0 receive USCI_B0 I2C status	UCA0RXIFG, UCB0RXIFG(2)(5)	maskable	0FFEEh	23
USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	UCA0TXIFG, UCB0TXIFG(2)(6)	maskable	0FFECh	22
ADC10 (MSP430G2x53 only)	ADC10IFG(4)	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7(2)(4)	maskable	0FFE6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7(2)(4)	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
See (7)			0FFDEh	15
See (8)			0FFDEh to 0FFC0h	14 to 0, lowest

(1) A reset is generated if the CPU tries to fetch instructions from within the module register memory address range (0h to 01FFh) or from within unused address ranges.
(2) Multiple source flags
(3) (non)-maskable: the individual interrupt-enable bit can disable an interrupt event, but the general interrupt enable cannot.
(4) Interrupt flags are located in the module.
(5) In SPI mode: UCB0RXIFG. In I2C mode: UCALIFG, UCNACKIFG, ICSTTIFG, UCSTPIFG.
(6) In UART/SPI mode: UCB0TXIFG. In I2C mode: UCB0RXIFG, UCB0TXIFG.
(7) This location is used as bootstrap loader security key (BSLSKEY). A 0xAA55 at this location disables the BSL completely. A zero (0h) disables the erasure of the flash if an invalid password is supplied.
(8) The interrupt vectors at addresses 0FFDEh to 0FFC0h are not used in this device and can be used for regular program code if necessary.

MSP430G2553

INTERRUPÇÕES

Interrupções requerem o armazenamento de valores dos registradores da CPU. Pode-se:

- Copiar regs. na pilha (Freescale HCS08);
- Mudar para um segundo conjunto de regs.(Z80);
- Salvar somente o endereço de retorno (PIC16).

INTERRUPÇÕES

MSP430 salva o endereço de retorno e o SR (status register), que controla os modos de baixo consumo.

INTERRUPÇÕES

Exemplo: TAR==0 && TAIE==1 && GIE==1

1. Instrução atual é executada se CPU estiver ativa, ou MCLK é ativado se a CPU estiver inativa.

INTERRUPÇÕES

2. PC é guardado na pilha.

3. SR é guardado na pilha.

4. Interrupção de maior prioridade é chamada.

INTERRUPÇÕES

5. Flag de interrupção é mantida se endereço da interr. for compartilhado, ou flag é zerada se endereço não for compartilhado.

INTERRUPÇÕES

6. SR é zerado: interrupções mascaráveis e modos de baixo consumo são desabilitados.

7. PC obtém endereço da interrupção, e CPU executa ISR.

INTERRUPÇÕES

Este processo toma 6 ciclos de clock. Como a última instrução deve ser executada, considere a instrução mais demorada, que demora mais 6 ciclos.

A pior latência entre o pedido de interrupção e a entrada na ISR é de 12 ciclos.

INTERRUPÇÕES

Se estiver em modo de baixo consumo, a latência é de 6 ciclos + tempo para acionar o MCLK.

INTERRUPÇÕES

Em Assembly, a ISR deve terminar com *reti (return from interrupt)*, que devolve para SR e para o PC os seus valores, que estavam na pilha.

Este processo dura 5 ciclos de clock.

INTERRUPÇÕES

Listing 6.4: Program `timrint1.s43` in assembly language to toggle LEDs using interrupts generated by timer_A in up mode.

```
; timrint1.s43 - toggles LEDs with period of about 1s
; TACCR0 interrupts from timer A with period of about 0.5s
; Timer clock is SMCLK divided by 8, up mode, period 50000
; Olimex 1121STK, LED1,2 active low on P2.3,4
; J H Davies, 2006-09-20; IAR Kickstart version 3.41A
;-------------------------------------------------------------------------
#include <msp430x11x1.h>            ; Header file for this device
;-------------------------------------------------------------------------
; Pins for LED on port 2
LED1        EQU     BIT3
LED2        EQU     BIT4
;-------------------------------------------------------------------------
    RSEG    CSTACK                  ; Create stack (in RAM)
;-------------------------------------------------------------------------
    RSEG    CODE                    ; Program goes in code memory
Reset:                              ; Execution starts here
    mov.w   #SFE(CSTACK),SP         ; Initialize stack pointer
main:                               ; Equivalent to start of main() in C
    mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
    mov.b   #LED2,&P2OUT            ; Preload LED1 on, LED2 off
    bis.b   #LED1|LED2,&P2DIR       ; Set pins with LED1,2 to output

    mov.w   #49999,&TACCR0          ; Period for up mode
    mov.w   #CCIE,&TACCTL0          ; Enable interrupts on Compare 0
    mov.w   #MC_1|ID_3|TASSEL_2|TACLR,&TACTL    ; Set up Timer A
; Up mode, divide clock by 8, clock from SMCLK, clear TAR
    bis.w   #GIE,SR                 ; Enable interrupts (just TACCR0)
    jmp     $                       ; Loop forever; interrupts do all
;-------------------------------------------------------------------------
; Interrupt service routine for TACCR0, called when TAR = TACCR0
; No need to acknowledge interrupt explicitly - done automatically
TA0_ISR:                            ; ISR for TACCR0 CCIFG
    xor.b   #LED1|LED2,&P2OUT       ; Toggle LEDs

    reti                            ; That's all: return from interrupt
;-------------------------------------------------------------------------
    COMMON  INTVEC                  ; Segment for vectors (in Flash)
    ORG     TIMERA0_VECTOR
    DW      TA0_ISR                 ; ISR for TA0 interrupt
    ORG     RESET_VECTOR
    DW      Reset                   ; Address to start execution
    END
```

INTERRUPÇÕES

Listing 6.5: Program `timintC1.c` to toggle LEDs using interrupts generated by channel 0 of Timer_A in up mode.

```
// timintC1.c - toggles LEDs with period of about 1.0s
// Toggle LEDs in ISR using interrupts from timer A CCR0
//   in Up mode with period of about 0.5s
// Timer clock is SMCLK divided by 8, up mode, period 50000
// Olimex 1121STK, LED1,2 active low on P2.3,4
// J H Davies, 2006-10-11; IAR Kickstart version 3.41A
//-----------------------------------------------------------------------
#include <io430x11x1.h>                 // Specific device
#include <intrinsics.h>                 // Intrinsic functions
//-----------------------------------------------------------------------
// Pins for LEDs
#define LED1    BIT3
#define LED2    BIT4
//-----------------------------------------------------------------------
void main (void)
{
    WDTCTL = WDTPW|WDTHOLD;             // Stop watchdog timer
    P2OUT = ~LED1;                      // Preload LED1 on, LED2 off
    P2DIR = LED1|LED2;                  // Set pins with LED1,2 to output
    TACCR0 = 49999;                     // Upper limit of count for TAR
    TACCTL0 = CCIE;                     // Enable interrupts on Compare 0
    TACTL = MC_1|ID_3|TASSEL_2|TACLR;   // Set up and start Timer A
// "Up to CCR0" mode, divide clock by 8, clock from SMCLK, clear timer
    __enable_interrupt();               // Enable interrupts (intrinsic)
    for (;;) {                          // Loop forever doing nothing
    }                                   // Interrupts do the work
}
//-----------------------------------------------------------------------
// Interrupt service routine for Timer A channel 0
//-----------------------------------------------------------------------
#pragma vector = TIMERA0_VECTOR
__interrupt void TA0_ISR (void)
{
        P2OUT ^= LED1|LED2;             // Toggle LEDs
}
```

INTERRUPÇÕES

Dicas para ISRs:

1. Que elas sejam curtas.
2. Desabilite interrupções indesejadas.
3. Defina os vetores de interrupção para debugar o programa.
4. Evite compartilhar dados com main().

MODOS DE BAIXO CONSUMO

Economia de consumo de energia.

Cinco modos, três mais utilizados.

==> Com base no MSP430F2013:

Vcc = 3V, DCO @ 1MHz, LFXT1 @ 32kHz

MODOS DE BAIXO CONSUMO

Modo ativo:

- CPU, clocks e módulos ativos
- Corrente próxima a 300uA, podendo ser reduzida de acordo com Vcc, desde que suporte MCLK

MODOS DE BAIXO CONSUMO

LPM0:

- CPU e MCLK inativos, SMCLK e ACLK ativos
- Corrente próxima a 85uA

MODOS DE BAIXO CONSUMO

LPM3:

- CPU, MCLK, SMCLK e DCO inativos, ACLK ativo
- Corrente próxima a 1uA

MODOS DE BAIXO CONSUMO

LPM4:

- CPU e clocks inativos
- Corrente próxima a 0,1uA

MODOS DE BAIXO CONSUMO

Modos definidos no SR por SCG0,
SCG1, CPUOFF e OSCOFF

Em Assembly:

==> bis.w #LPM3, SR

==> bis.w #GIE | LPM3, SR

Em C:

==> _low_power_mode_3();

==> _BIS_SR(GIE | LPM3_Bits);