Assembly en interrupts op de MSP430

4.3.3 Status Register (SR)

The 16-bit Status Register (SR, also called R2), used as a source or destination register, can only be used in register mode addressed with word instructions. The remaining combinations of addressing modes are used to support the constant generator. Figure 4-9 shows the SR bits. Do not write 20-bit values to the SR. Unpredictable operation can result.

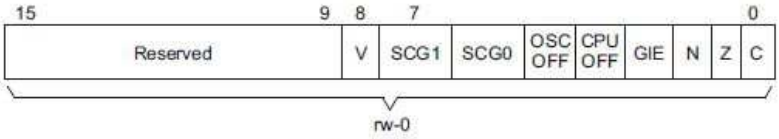

Figure 4-9. SR Bits

Table 4-1 describes the SR bits.

Table 4-1. SR Bit Description

Bit	Description	
Reserved	Reserved	
V	Overflow. 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[1]	System clock generator 1. This bit may be used to enable or disable functions in the clock system depending on the device family; for example, DCO bias enable or disable.	
SCG0[1]	System clock generator 0. This bit may be used to enable or disable functions in the clock system depending on the device family; for example, FLL enable or disable.	
OSCOFF[1]	Oscillator off. When this bit is set, it turns off the LFXT1 crystal oscillator when LFXT1CLK is not used for MCLK or SMCLK.	
CPUOFF[1]	CPU off. When this bit is set, it turns off the CPU.	
GIE	General interrupt enable. When this bit is set, it enables maskable interrupts. When it is reset, all maskable interrupts are disabled.	
N	Negative. This bit is set when the result of an operation is negative and cleared when the result is positive.	
Z	Zero. This bit is set when the result of an operation is 0 and cleared when the result is not 0.	
C	Carry. This bit is set when the result of an operation produced a carry and cleared when no carry occurred.	

(1) The bits CPUOFF, OSCOFF, SCG0, and SCG1 request the system to enter a low-power mode

NOTE: Bit manipulations of the SR should be done by the following instructions: MOV, BIS, and BIC.

Voorbeeld één

Een datastructuur in RAM waarin je snel wil lezen en schrijven.

```
create RWDATA  5 cells allot \ R/W data info

: !DATA     ( x +n -- )     cells rwdata + ! ;  \ High level
: @DATA     ( +n -- x )     cells rwdata + @ ;
```

De code variant gebruikt de indirecte adressering van de MSP430 om tot een efficiënt stukje code te komen.

```
code !DATA  ( x +n -- )
    tos  tos add   rwdata #  tos add
    sp )+  tos ) mov   sp )+  tos mov   next
end-code
code @DATA  ( +n -- x )
    tos  tos add   rwdata #  tos add
    tos )  tos mov   next
end-code
```

Voorbeeld twee

Een low power variant van KEY) als er geen toets ingedrukt is (dat staat in IFG2 bit-0) zet KEY) de processor in een low power mode. Hier is dat LPM2 maar als er ook nog andere "devices" actief zijn kan het zijn dat je minder diep kan gaan slapen. Dan is LPM0 een goed alternatief... De processor gebruikt dan wel meer stroom!

```
\ Code routines for a low power version of KEY
code KEY)       ( -- c )
    tos  sp -) mov          \ Push TOS
    #1  03 & .b bit         \ IFG2  Test RX flag
    =? if,                  \ Not set ?
        #8  sr bic          \ Interrupt off (Important!)
        #1  01 & .b bis     \ IE2  RX interrupt on
        98 #  sr bis        \ Go from AM to LPM2 & intrpt on
\       18 #  sr bis        \ Go from AM to LPM0 & intrpt on
    then,
    66 &  tos .b mov        \ UCA0RXBUF  Get char
    next
end-code

routine RX-INTERRUPT
    #1  01 & .b bic         \ IE2  RS232 RX interrupt off
    F0 #  rp ) bic          \ LPM off
    reti
END-CODE

rx-interrupt    FFEE !vec   \ Set interrupt routine for key)
```

Tips en truuks:

Data van de stack halen	De top staat altijd in TOS
2é item van stack halen	`sp )+ <register> mov ( Etcetera )`
3é item van stack ophalen	`2 sp x) <register> mov`
Data op stack pushen	`tos sp -) mov`
Forth woorden aanroepen	`ip push ' words >body # ip mov next`
Of:	`' words # w mov w )+ pc mov`
Gebruik van variabelen	`variable-naam & moon mov`
en values	`adr value-naam & moon mov`
Constanten	`constantnaam # moon mov`
Macro	`: MACRO-2* [ assembler ] tos tos add [ forth ] ;`
Subroutine aanroepen	`subroutine-naam # call`
De MSP430 en aligned	Let op alle geheugen cell aligned!

Commacode:

Laad commacode.f eerst.
Compileer een code woord, type daarna commacode in.
Bewerk de gegenereerde commacode.

1) Waar in de code een constante staat getal vervangen door de naam van die constante
2) Waar in de code een variabele of value adres staat vervangen door zijn naam
3) Tenslotte de hex-code voor NEXT vervangen door NEXT
   De hex code is: `4F00 ,` of `4536 , 4630 ,`

De commacode stappen:

Voor het genereren van commacode, eerst het code woord assembleren, daarna commacode uitvoeren.

```
@)create RWDATA  5 cells allot \ R/W data info  RWDATA is not
new  OK.0
@)  OK.0
@)code !DATA  ( x +n -- )  !DATA is not new  OK.2
@)    tos  tos add   rwdata #  tos add  OK.2
@)    sp )+  tos ) mov   sp )+  tos mov   next  OK.2
@)end-code  OK.0
@)commacode
```

Stap 1: Code kopiëren

```
code !DATA   5707 ,  5037 ,  208A ,  44B7 ,
0 ,  4437 ,  4F00 ,  end-code
```

Stap 2: Variabele, etc. vervangen

```
code !DATA   5707 ,  5037 ,  rwdata ,  44B7 ,
0 ,  4437 ,  4F00 ,  end-code
```

Stap 3: Next vervangen en klaar.

```
code !DATA   5707 ,  5037 ,  rwdata ,  44B7 ,
0 ,  4437 ,  next  end-code
```

Programmeer opdrachten:

1) Schrijf een code woord dat alleen de lage byte achter laat (u -- lb)

2) Schrijf een code woord dat alleen de hoge byte achter laat (u -- hb)

3) Schrijf een code woord **SPLIT** dat de lage en hoge byte zo achter laat (u -- lb hb)

4) Zet dit om in code: `: >DEV  ( a -- )  FE and  to dev ;`
   En maak er commacode van, **dev** is een value!

Uitkomsten:

```
code LOWBYTE  ( u -- lb )
     #-1  tos .b bia   next
end-code
```

```
code HIGHBYTE ( u -- hb )
     tos swpb   #-1  tos .b bia   next
end-code
```

```
code SPLIT    ( u -- lb hb )
     tos  day mov  tos swpb
     #-1  tos .b bia  #-1  day .b bia
     day  sp -) mov   next
end-code
```

```
value DEV
code >DEV    ( a -- )
     FE #  tos bia
     tos  adr DEV & mov
     sp )+ tos mov   next
end-code
```

```
commacode
code >DEV   F037 ,  FE ,
4782 ,  208A ,  4437 ,  4F00 ,
end-code
```

```
@)adr dev . 208A  OK.0
```

```
code >DEV   F037 ,  FE ,
4782 ,  adr DEV ,  4437 ,  NEXT
end-code
```