

# Arithmetik mit 8-Bit Zahlen

1. Studieren Sie die Befehle ADD und SUB im Freescale "HCS08 Family Reference Manual" und interpretieren Sie die beiden Folien „Carry und Overflow beim ADD/SUB-Befehl“.

**ADD** Add without Carry

Operation  
 $A \leftarrow (A) + (M)$

Description  
 Adds the contents of M to the contents of A and places the result in A

Condition Codes and Boolean Formulae

V	H	I	N	Z	C
1	1	1	1	1	1

V:  $A7 \& M7 \& R7 \mid \overline{A7 \& M7 \& R7}$   
 Set if a two's complement overflow resulted from the operation; cleared otherwise

H:  $A3 \& M3 \mid M3 \& R3 \mid R3 \& A3$   
 Set if there was a carry from bit 3; cleared otherwise

N: R7  
 Set if MSB of result is 1; cleared otherwise

Z:  $R7 \& R6 \& R5 \& R4 \& R3 \& R2 \& R1 \& R0$   
 Set if result is \$00; cleared otherwise

C:  $A7 \& M7 \mid M7 \& R7 \mid R7 \& A7$   
 Set if there was a carry from the MSB of the result; cleared otherwise

**ADD SUB** Subtract

Operation  
 $A \leftarrow (A) - (M)$

Description  
 Subtracts the contents of M from A and places the result in A

Condition Codes and Boolean Formulae

V	H	I	N	Z	C
1	1	1	1	1	1

V:  $A7 \& M7 \& R7 \mid \overline{A7 \& M7 \& R7}$   
 Set if a two's complement overflow resulted from the operation; cleared otherwise. Literally read, an overflow condition occurs if a positive number is subtracted from a negative number with a positive result, or, if a negative number is subtracted from a positive number with a negative result.

N: R7  
 Set if MSB of result is 1; cleared otherwise

Z:  $R7 \& R6 \& R5 \& R4 \& R3 \& R2 \& R1 \& R0$   
 Set if result is \$00; cleared otherwise

C:  $\overline{A7 \& M7} \mid M7 \& R7 \mid R7 \& \overline{A7}$   
 Set if the unsigned value of the contents of memory is larger than the unsigned value of the accumulator; cleared otherwise

Block 5 Assembler Programmier Techniken

## Carry (3 Fälle, oben) und Overflow (2 Fälle, unten) beim ADD-Befehl

Legend:  
 — A(OP1)  
 — M(OP2)  
 — R(Resultat)

Condition Codes:  
 C:  $A7 \& M7 \mid M7 \& R7 \mid A7 \& R7$   
 V:  $\overline{A7 \& M7 \& R7} \mid A7 \& M7 \& R7$   
 N: R7  
 Z:  $R7 \& R6 \& R5 \& R4 \& R3 \& R2 \& R1 \& R0$

Block 5 Assembler Programmier Techniken

## Carry (3 Fälle, oben) und Overflow (2 Fälle, unten) beim SUB-Befehl

Legend:  
 — A(OP1)  
 — M(OP2)  
 — R(Resultat)

Condition Codes:  
 C:  $\overline{A7 \& M7} \mid M7 \& R7 \mid \overline{A7 \& R7}$   
 V:  $A7 \& M7 \& R7 \mid \overline{A7 \& M7 \& R7}$   
 N: R7  
 Z:  $R7 \& R6 \& R5 \& R4 \& R3 \& R2 \& R1 \& R0$

2. Schreiben Sie einen Programmteil, welcher 2 Operanden von den Adressen \$00C0 und \$00C1 einliesst und nach einer noch anzugebenden arithmetischen Verknüpfung (Aufgabe 1.3 und 1.4) das Resultat ins X-Register schreibt und die aktuellen Werte der Flags VNZC auf 4 LEDs wie folgt ausgibt:

C → PTF0 = LED B FL

Z → PTF1 = LED R FL

V → PTE7 = LED B FR

N → PTF2 = LED R FR

*Die LED sind schön 1:1 wie die Flags im CCR sind. D.h. man muss die Flags / Bits nicht schieben und kann sie direkt auf den Port F resp. E ausgeben.*

```
; Rear-LED ON, damit sichtbar ist, dass der MC-Car noch läuft (Save Battery)
;-----
                LDA    #$04                ; LED Rear On --> uP still running ;- )
                STA    PTFDD
                STA    PTFD

; Start des Benutzer-Codes
;-----

userMain: LDA    #$07
          STA    PTFDD                ; Pin0-3, Port F DataDirection = Output
          LDA    #$80
          STA    PTFDD                ; Pin 7, Port E DataDirection

          LDA    #$FF                ; All LED OFF
          STA    PTED
          STA    PTFD

          JMP    AUFGABE1            ; Jump to Aufgabe X

;----- Aufgabe 4.1 -----
AUFGABE1: LDA    #$00                ; Default-Wert 00 in Speicher C0 und C1 speichern
          STA    $00C0
          STA    $00C1

Loop:    LDA    $00C0                ; Load OP1
          ADD    $00C1                ; ADD OP2
          ;SUB    $00C1                ; SUB OP2
          TAX
          TPA                        ; Result -> X // Transfer Accumulator to X
          ; Transfer Processor Status Byte to Accumulator
          ; (CCR -> Akku)
          COMA                        ; Invertiere Akku (CCR) because of low-active LED
          STA    PTED                ; V -> PTE7
          STA    PTFD                ; NZC -> PTF2..0
          BRA    Loop                ; Endlosschleife
```

3. Benutzen Sie den unter 1.2 vorbereiteten Programmteil, um die Flagsetzung beim ADD Befehl zu analysieren. Legen Sie dazu vor dem Programmablauf zweckmässige Operanden auf die Adressen \$00C0 und \$00C1 (z.B mit dem Debugger Befehl >mem). Verifizieren Sie die erwarteten Flags auf der Anzeige der roten und blauen FrontLEDs und stellen Sie den Bezug zu den Zahlenkreisdiagrammen auf den Folien her.

*siehe oben!*

4. Machen Sie dasselbe wie in Aufgabe 1.3 für den Befehl SUB.

*siehe oben!*

## Bitmanipulationen

1. Schreiben Sie ein kleines Programm, welches das Carry-Flag im CCR mittels Bit-Maskierung löscht! Kontrollieren Sie Ihr Programm, indem Sie das Carry-Flag zuerst mit >reg setzen.

```

TPA
AND #$FE
TAP

; Rear-LED ON, damit sichtbar ist, dass der MC-Car noch läuft (Save Battery)
;-----
LDA    #$04                ; LED Rear On --> uP still running ;- )
STA    PTDDD
STA    PTDD

; Start des Benutzer-Codes
;-----

userMain: LDA    #$07
          STA    PTFDD        ; Pin0-3, Port F DataDirection = Output
          LDA    #$80
          STA    PTEDD        ; Pin 7, Port E DataDirection

          LDA    #$FF        ; All LED OFF
          STA    PTED
          STA    PTFD

          JMP    AUFGABE2      ; Jump to Aufgabe 1 / 2 or 3

;----- Aufgabe 4.2 -----
; Carry-Flag im CCR mittels Bit-Maskierung löschen
; vorher Flag setzen, z.B. mittels Debugger Shell: reg SR=0x01
AUFGABE2: NOP                ; set Carry-Flag now --> SR=0x01
          TPA                ; CCR -> Akku (Processor Status Byte)
          AND    #$FE        ; Clear Carry-Flag (Bit0) %00000001
          TAP                ; Akku -> CCR

          ; Overflow-Flag mittels Bit-Maskierung setzen: reg SR=0x00
          NOP                ; set Overflow-Flag now --> SR=0x00
          TPA                ; CCR -> Akku
          ORA    #$80        ; Set Overflow-Flag (Bit7) %10000000
          TAP                ; Akku -> CCR

          ; direkte setzen des Carry-Flag
          SEC                ; Set Carry-Flag

          ; direkte löschen des Carry-Flag
          CLC                ; Clear Carry-Flag

EndLoop2: BRA    AUFGABE2      ; Endlos-Loop

```

2. Schreiben Sie ein kleines Programm, welches das Overflow-Flag mittels Bit-Maskierung setzt!

*siehe oben!*

```

TPA
ORA #$80
TAP

```

3. Es gibt einige Befehle, welches das direkte Setzen (Sxx von SET) bzw. Löschen (Cxx von CLEAR) zulassen. Wie heissen die Befehle, welche dasselbe wie unter Aufgabe 2.1 und 2.2 tun?

```

SEC    ; Set Carry-Flag
CLC    ; Clear Carry-Flag

```

## BRANCH-Befehle

### 1. Informieren Sie sich im HSLU\_HCS08\_Programming\_Guide über die BRANCH-Befehle.

```

; BCC      Branch if Carry Bit Clear (C = 0)
; BCS      Branch if Carry Bit Set (C = 1)
; BNE      Branch if Not Equal (Z = 0)
; BEQ      Branch if Equal (Z = 1)
; BHCC     Branch if Half Carry Bit Clear (H = 0)
; BHCS     Branch if Half Carry Bit Set (H = 1)
; BMC      Branch if Interrupt Mask Clear (I = 0)
; BMS      Branch if Interrupt Mask Set (I = 1)
; BPL      Branch if Plus (N = 0)
; BMI      Branch if Minus (N = 1)
; BRCLR n  Branch if Bit n in Memory Clear
; BRSET n  Branch if Bit n in Memory Set

; BGE      Branch if Greater Than or Equal To (N and V = 0)
; BGT      Branch if Greater Than (Z or (N and V) = 0)
; BHI      Branch if Higher (C or Z = 0)
; BHS      Branch if Higher or Same (Same as BCC)
; BLE      Branch if Less Than or Equal To ( Z or (N and V) = 1)
; BLO      Branch if Lower (C = 1)
; BLS      Branch if Lower or Same (C or Z = 1)
; BLT      Branch if Less Than (signed) (N and V = 1)

```

Wie heissen und wie funktionieren die Branch-Befehle mit denen man gleichzeitig maskieren kann?

**BRCLR n, Addr, Label** ; Verzweigt nach Label, wenn Bit n der Speicherstelle  
; an Adresse Addr nicht gesetzt ist (Addr nur DIR)

**BRSET n, Addr, Label** ; Verzweigt nach Label, wenn Bit n der Speicherstelle  
; an Adresse Addr gesetzt ist (Addr nur DIR)

### 2. Schreiben Sie ein kleines Programm, welches mittels eines Compare-Befehls zwei 8-Bit Operanden vergleicht und dann mittels verschiedener BRANCH Befehle verzweigt. Machen Sie sich so die Bedeutung verschiedener Branch Befehle klar (BCS, BHI, etc.).

Signalisation: Kein Sprung Schalte LED B FL ein (PTF0)

Sprung Schalte LED B FR ein (PTE7)

```

;----- Aufgabe 4.3 -----
AUFGABE3: LDA  #$FF                ; All LED OFF
            STA  PTED
            STA  PTFD

            LDA  #$0F                ; Load $0F into Akku
            CMP  #$0F                ; Compare with $0F
            BEQ  ISEQUAL              ; Branch if Equal -> JMPEQUAL (Z = 0)
NOTEQUAL: ; not Equal -> Schalte LED B FL ein (PTF0)
            LDA  #$FE                ; LED Low-Active! 01 -> FE
            STA  PTFD
            BRA  AUFGABE3            ; Loop
ISEQUAL: ; Equal -> Schalte LED B FR ein (PTE7)
            LDA  #$7F                ; LED Low-Active! 80 -> 7F
            STA  PTED
            BRA  AUFGABE3            ; Loop

```