

Assembler Workshop

Oliver Knodel

oliver.knodel@tu-dresden.de

Dresden, 16. April 2015



DRESDEN
concept
Exzellenz aus
Wissenschaft
und Kultur

01 Motivation

Warum eigentlich noch Assembler

- Nah an der Hardware und dadurch besseres Verständnis für die Funktionsweise.
- Sicherheitskritische Anwendungen oftmals direkt in Assembler geschrieben.
- Debuggen von Mikrocontrollern häufig auch auf Assemblerebene.
- ...

Agenda

- 01 Motivation
- 02 Hardware
- 03 Assembler
 - Blinkende LED
- 04 Adressierung, Direktiven und Basiskonstrukte
 - Berechnungen im RAM
- 05 Unterprogramme, Interrupts und Timer
 - Blinkende LED mit Unterprogrammen
 - Summe natürlicher Zahlen
 - Blinkende LED mit Timer
 - Programm zur Multiplikation
 - Berechnung der Fakultät
- 06 LCD Controller
 - Text auf dem LCD

*Kennzeichnung
von Aufgaben*



Assembler Workshop Zusatzmaterial:
<http://wwwpub.zih.tu-dresden.de/~knodel/ASM/>

02 Hardware

Texas Instruments MSP430

- 16 Bit RISC Mikrocontroller.
- Klassische Von-Neumann Architektur.
- 16 interne Register.
- Speichergrößen von bis zu 512 kByte Flash und bis zu 66 kByte RAM.
- Taktrate von bis zu 25 MHz.
- Anwendungsbereiche sind Eingebettete Systeme mit äußerst geringem Energieverbrauch (Ultra-Low-Power)

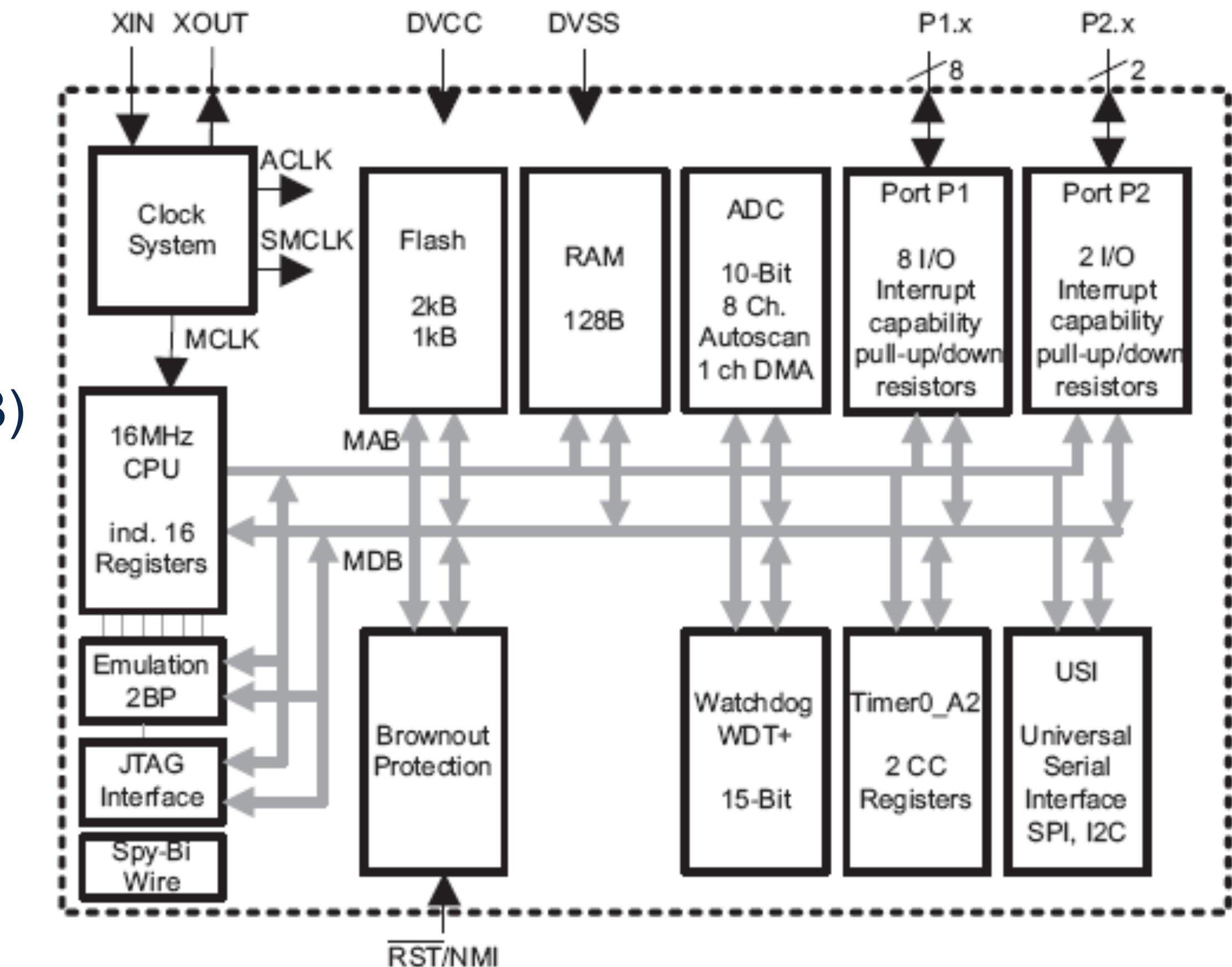


02 Hardware

Grundlegender Aufbau eines MSP430

Integrierte Peripherie und Komponenten:

- Takterzeugung,
- Supply Voltage Supervisor (SVS):
 - Brownout Protection,
 - Reset bei Spannungseinbruch.
- Speicher (Flash+RAM),
- Daten- und Speicherbus (MemoryAB/MDB)
- Timer,
- I/O Ports,
- Watchdog,
- DMA-Controller,
- Multiplizierer,
- Universal Serial Interface (USI):
 - UART,
 - ...



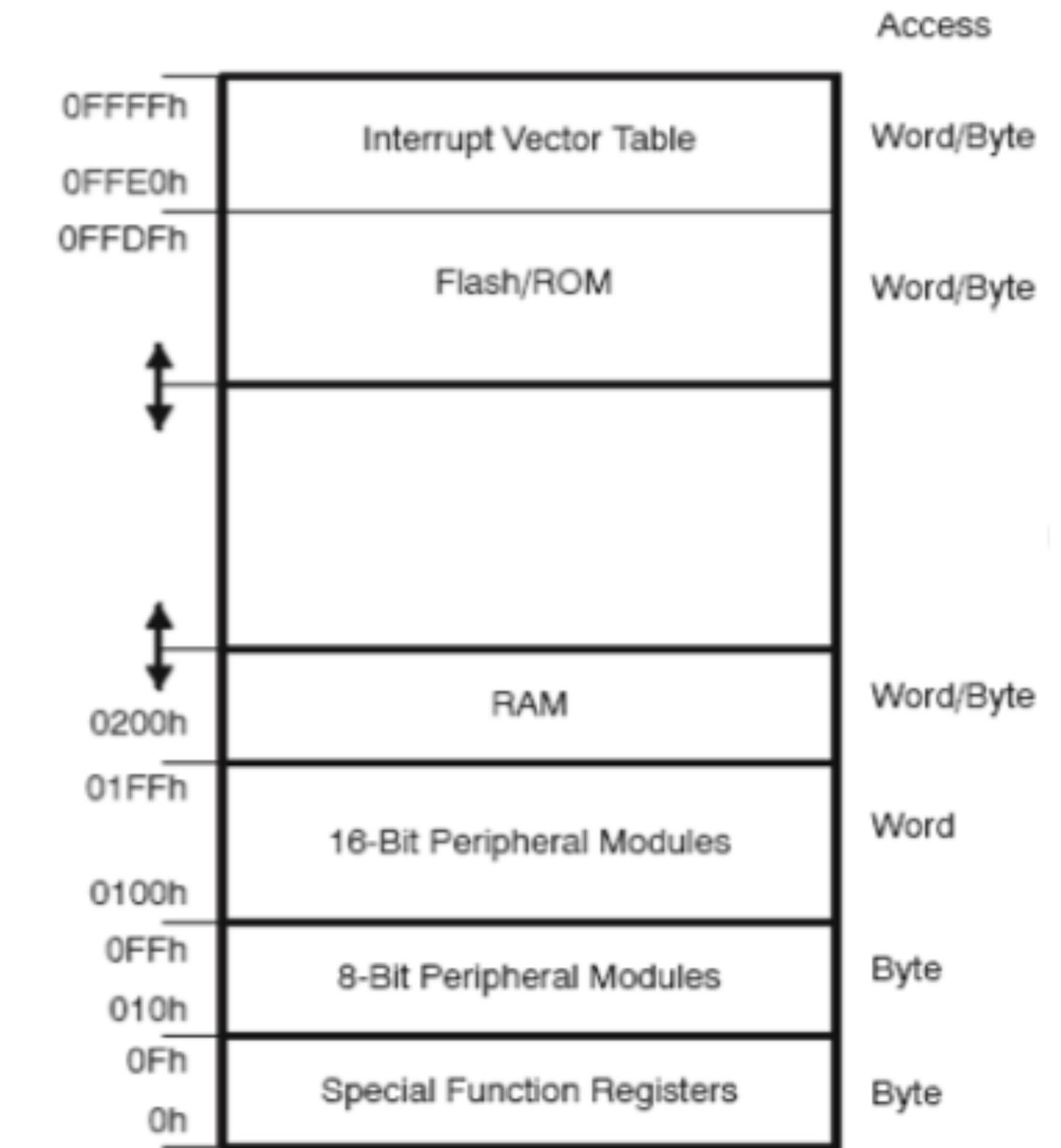
02 Hardware

Besondere Komponenten I – Speicher

- Register, I/O-Register, Peripherie, RAM und Flash sind in **einen** linear adressierbaren Speicher gemapped.
- Das Programm wird im Flash/ROM abgelegt und bleibt auch nach einem Reset erhalten.
- Daten zur Laufzeit werden im RAM abgelegt und sind nach einem Reset nicht mehr vorhanden.

Beispiel:

- Flash: 2 KByte
 - Von 0xF800 bis 0xFFDF
- RAM: 128 Byte
 - Von 0x0200 bis 0x0280



02 Hardware

Besondere Komponenten II – Watchdog (WDC)

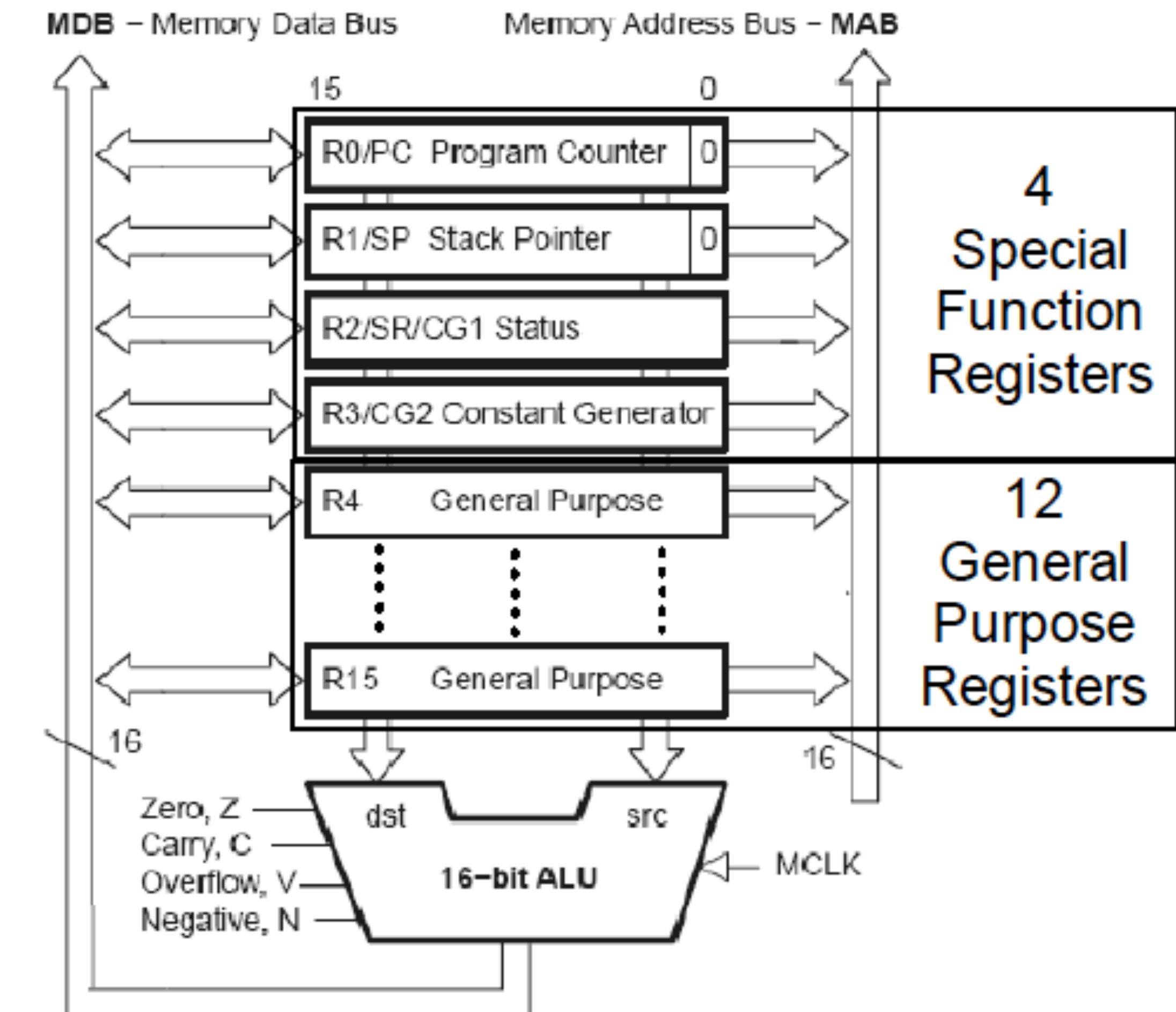
- Überwacht Funktion des Programms,
- Watchdog-Timer wird im Hintergrund inkrementiert,
- Timer wird an bestimmten Punkten des Programms immer wieder auf „0“ zurückgesetzt,
- Reset wenn Programm nicht mehr reagiert bzw. wenn der Timer einen Schwellwert erreicht hat.



02 Hardware

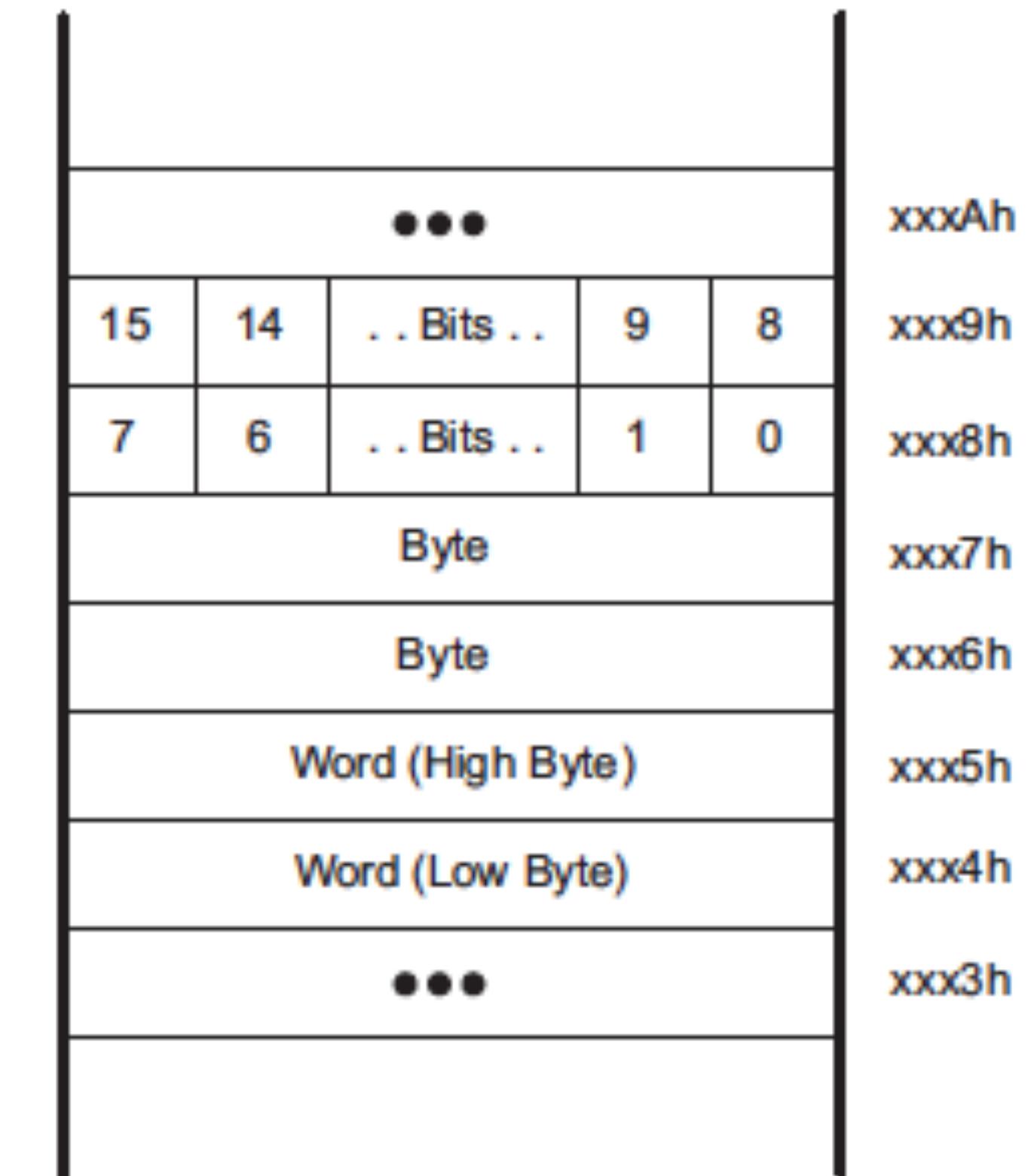
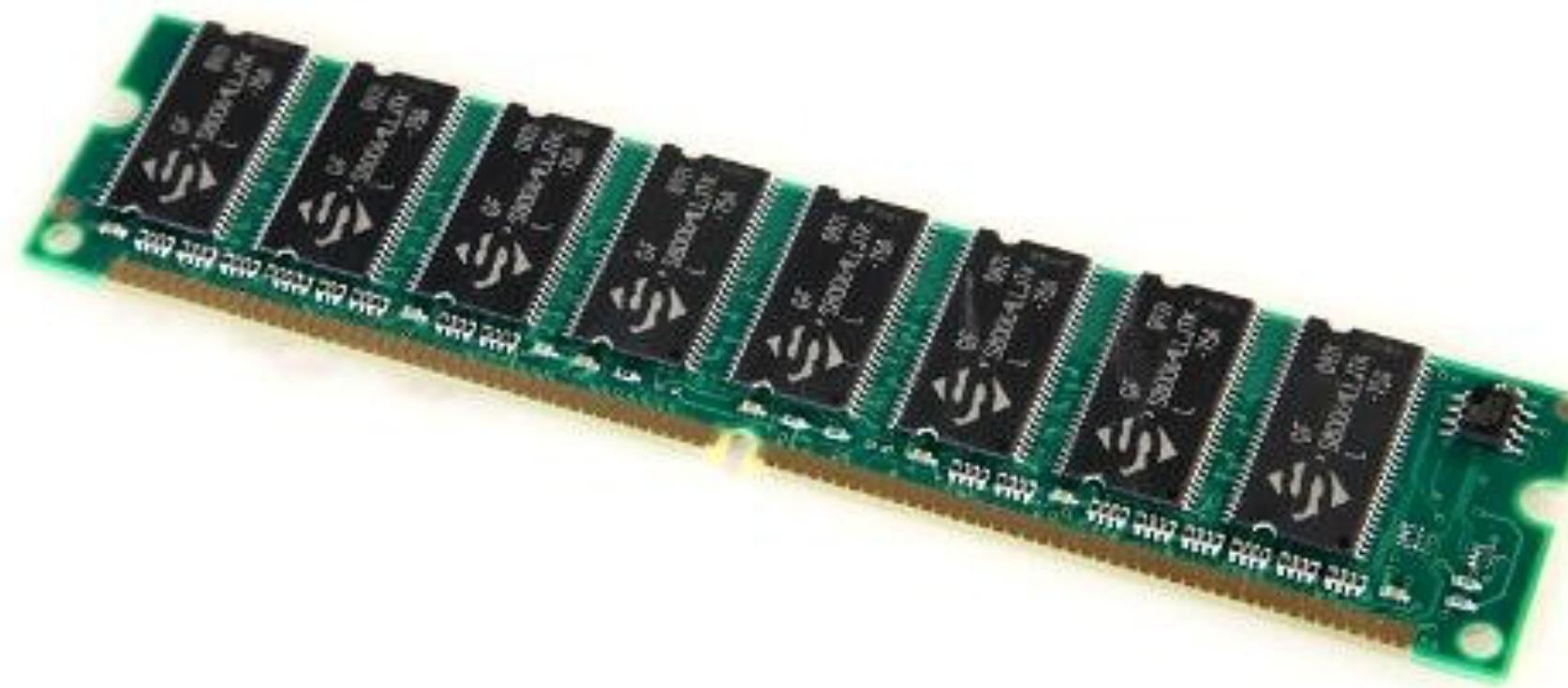
Die eigentliche CPU

- 16-Bit ALU mit 16 internen Registern:
 - Vier Spezialregister,
 - 12 Register für Daten.
- Vier Flags:
 - Zero,
 - Carry,
 - Overflow,
 - Negative.
- Einfache Arithmetische und Logische Operation (siehe Befehlsliste).



02 Hardware Speicherorganisation

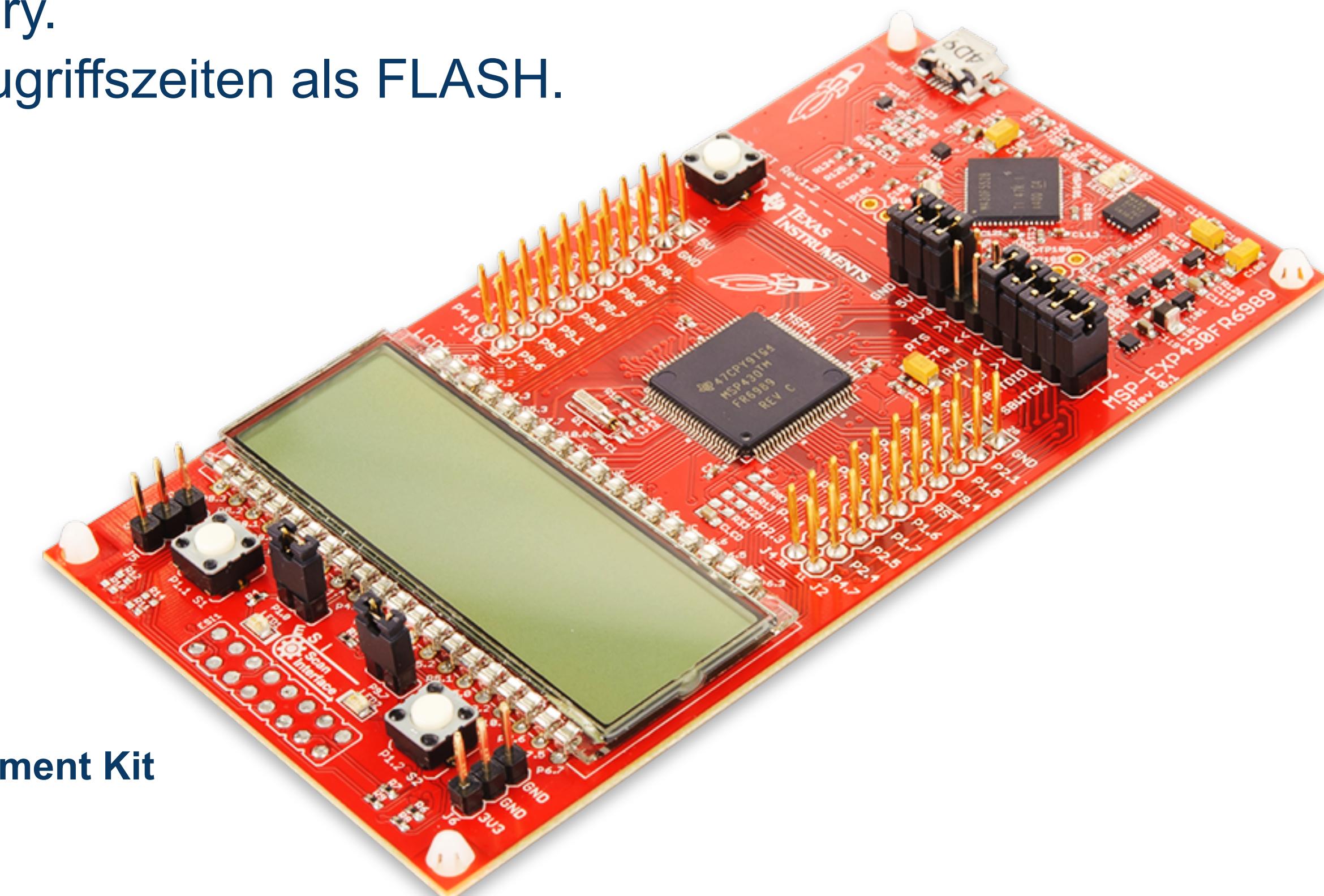
- An jeder Speicherstelle liegt genau ein Byte.
- Die Adressierung erfolgt mit 2 Byte.
- Das höherwertige Byte eines Wortes liegt auf der höheren Speicheradresse.



02 Hardware

Launchpad MSP-EXP430FR6989

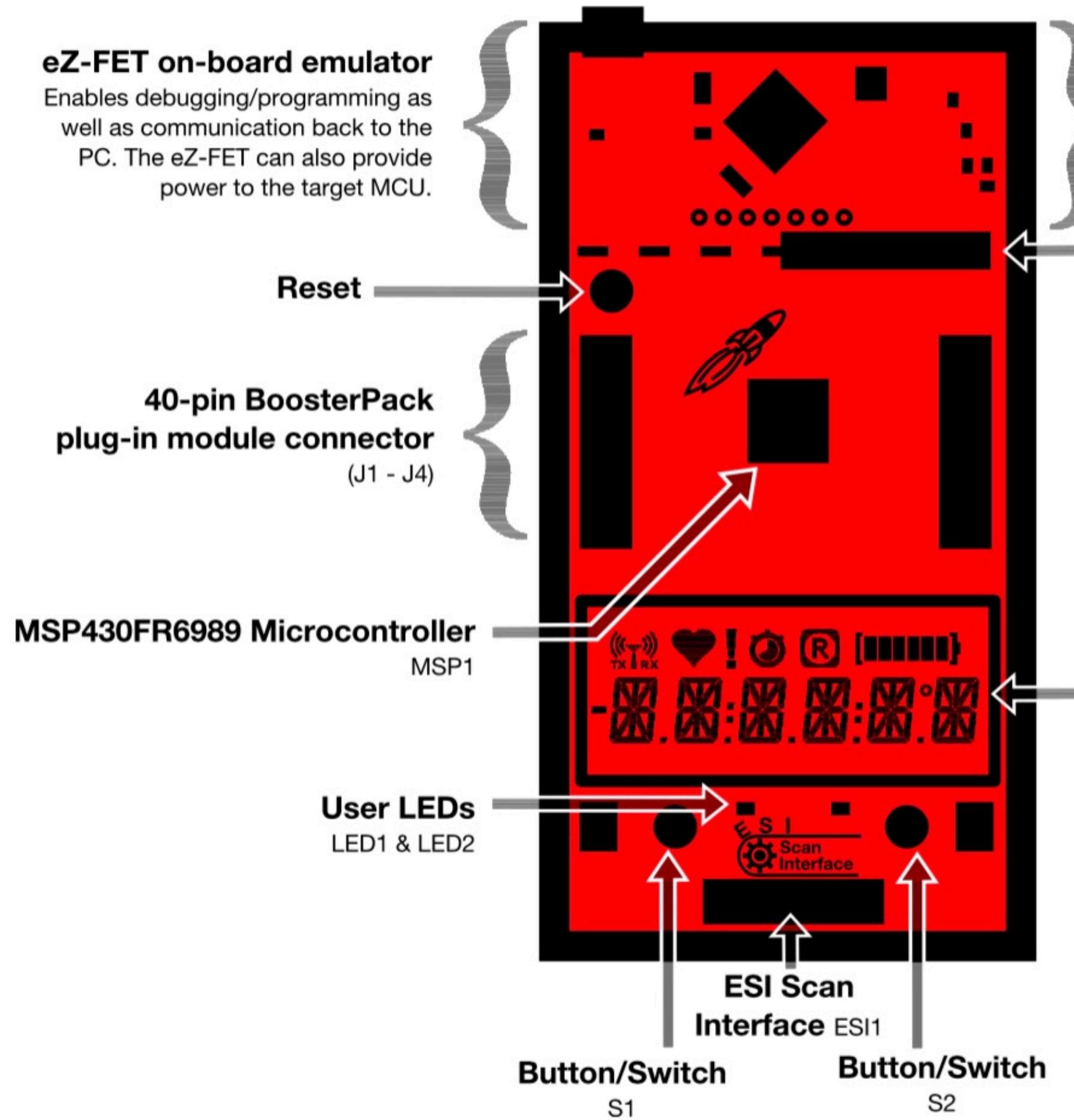
- Microcontroller: MSP430 FR6989 aus der MSP430FRx Serie mit 128 kByte FRAM Speicher:
 - Ferroelectric Random Access Memory.
 - Geringerer Energieverbrauch und Zugriffszeiten als FLASH.
- 40 Pin I/O
- EnergyTrace++
- LCD mit 320 Segmenten
- 2 x LED und
- 2 x Button



User's Guide — MSP430FR6989 LaunchPad™ Development Kit

SLAU627A – May 2015

02 Hardware Das Board im Detail



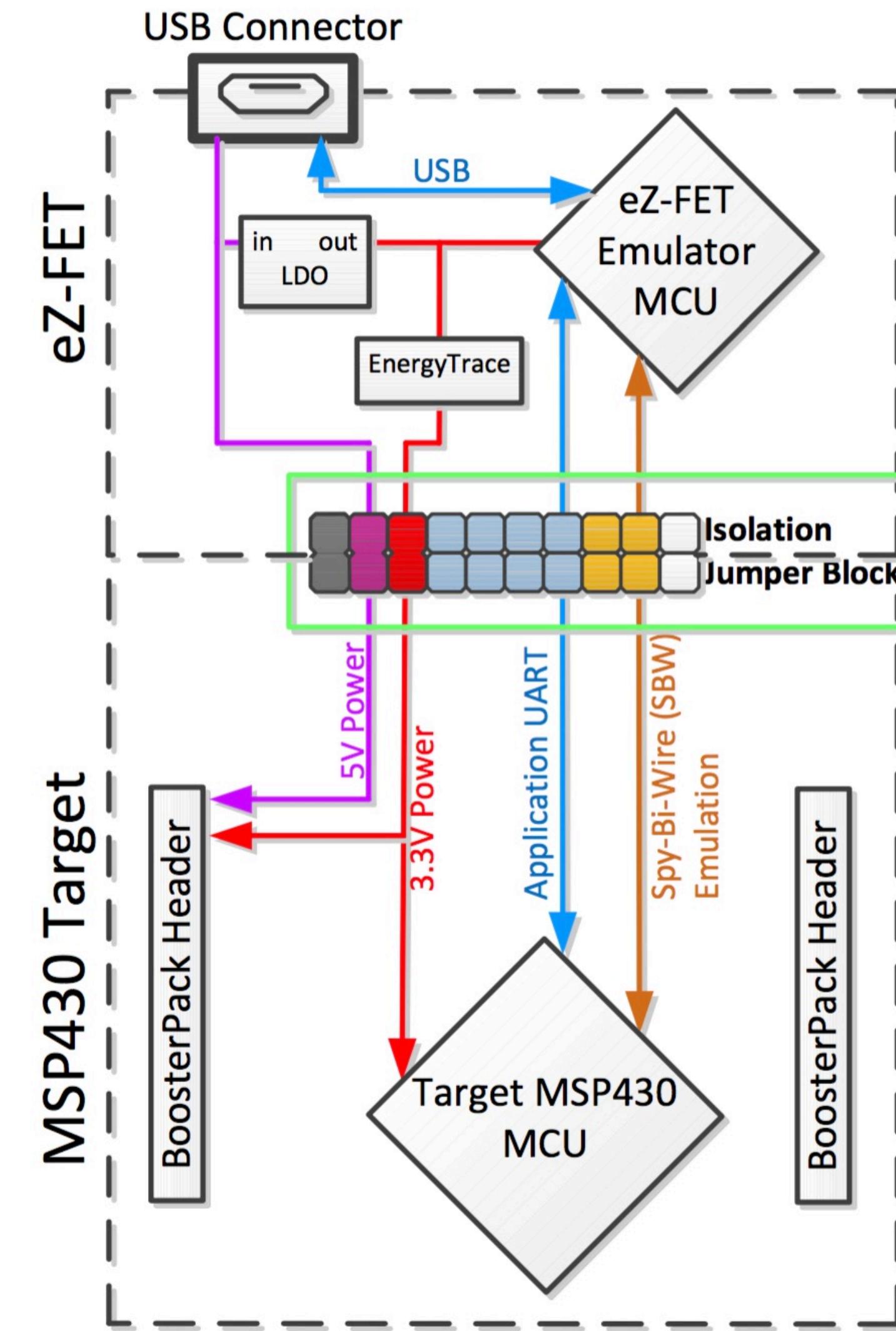
EnergyTrace Technology
Real-time power consumption readings & state updates from the MSP430FR6989 MCU, including CPU and peripheral state are viewable through the EnergyTrace GUI

Jumpers to isolate emulator from target MCU (J101)

- Back-channel UART to PC (RTS, CTS, RXD, TXD)
- Spy-bi-wire debug (SBWTDIO/SBWTC)
- Power (5V, 3V3, & GND)

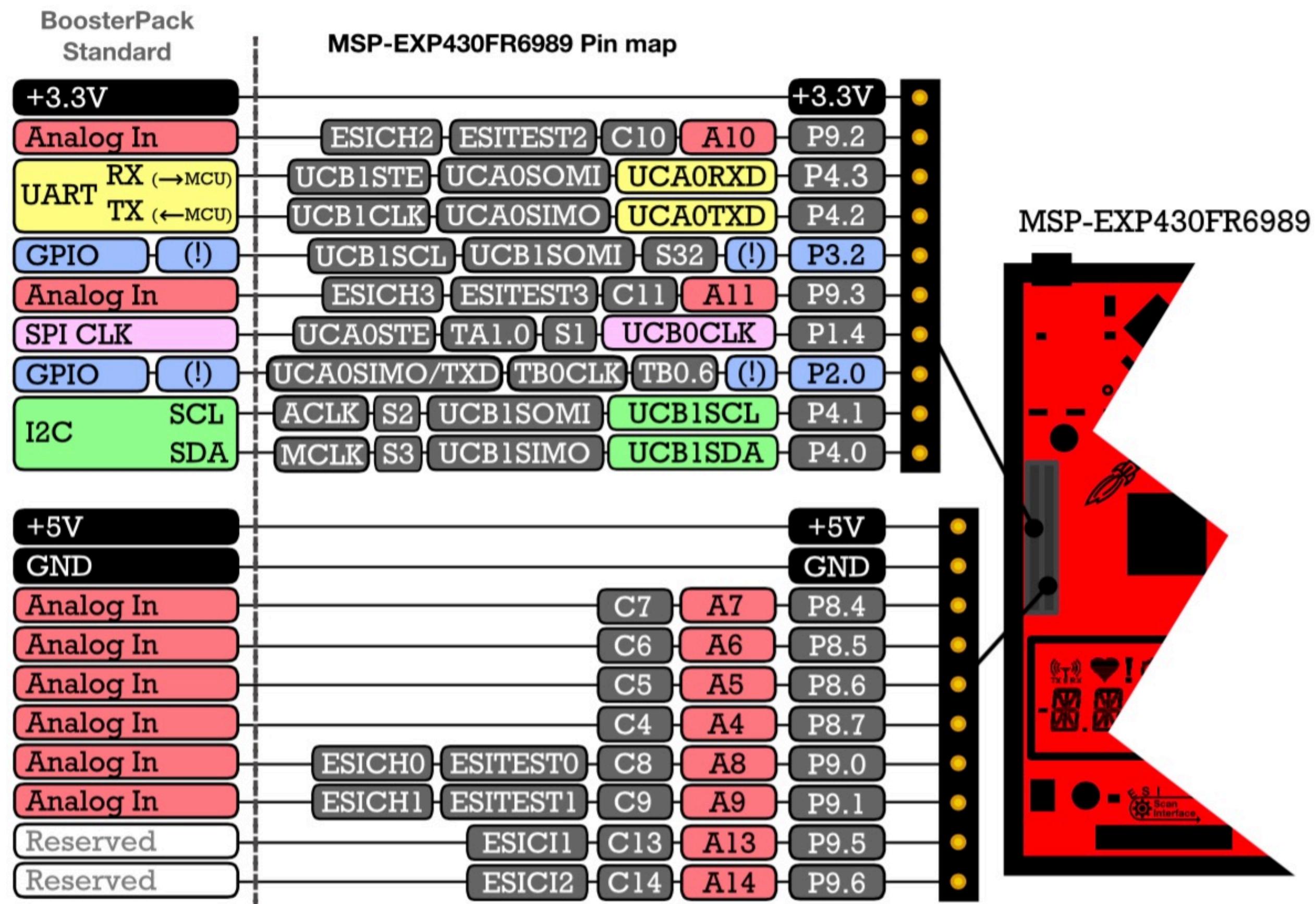
Segmented LCD Display

- 6 alphanumeric characters
- 6 symbols for various applications
- Ultra-low power display



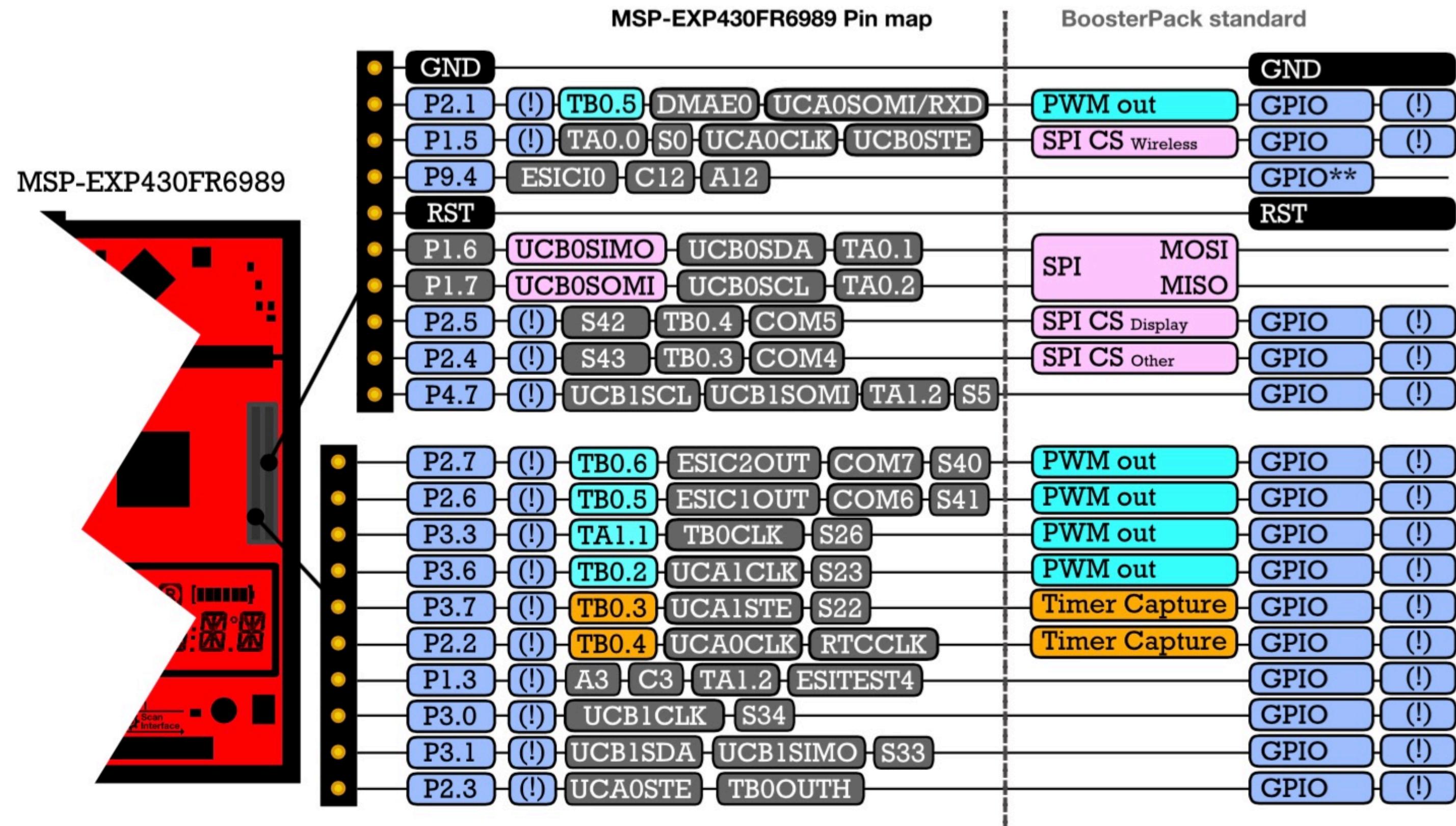
02 Hardware

Das Board im Detail



02 Hardware

Das Board im Detail



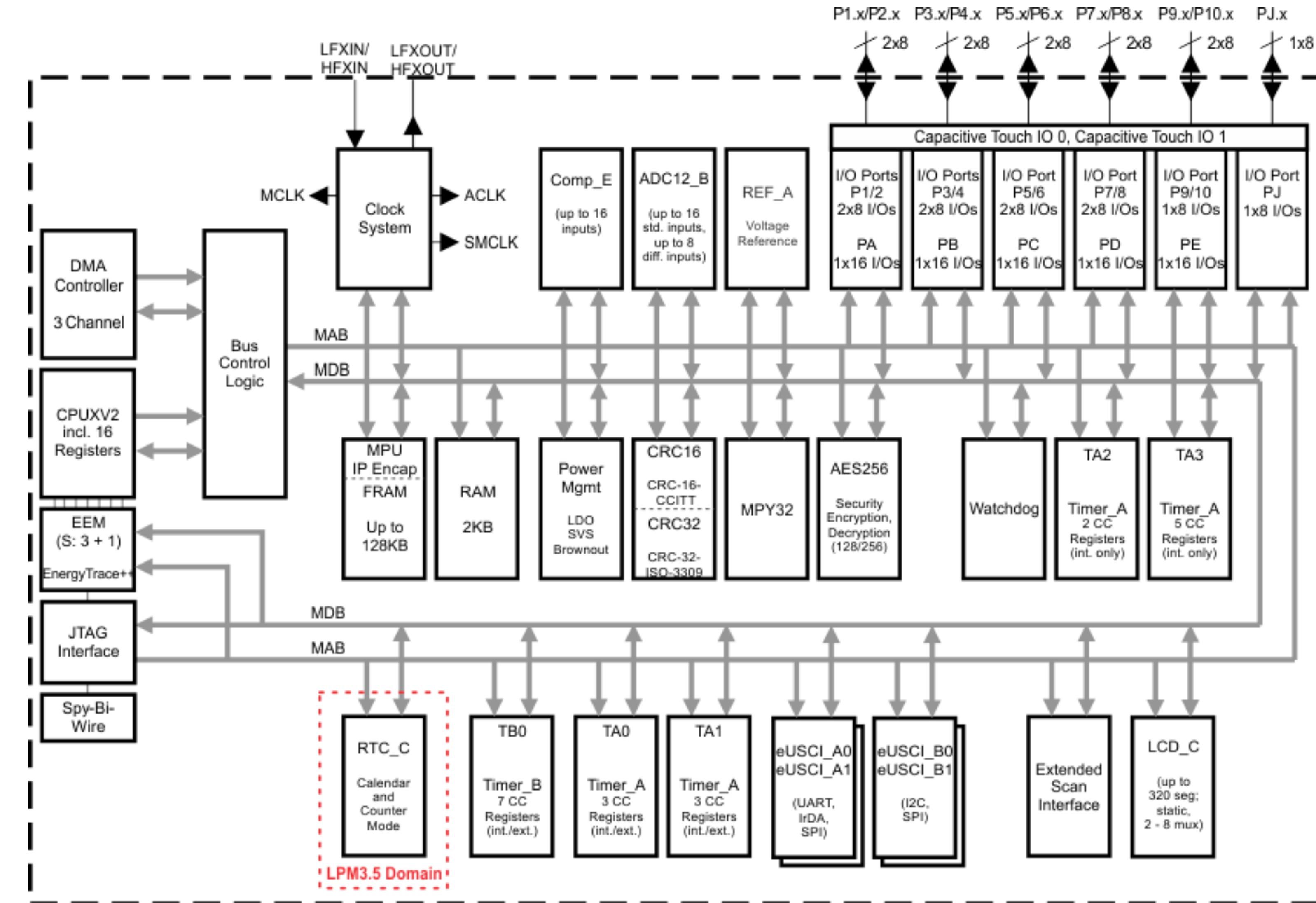
02 Hardware

Der Prozessor im Detail

- 1.8-V to 3.6-V operation
- 16-bit RISC architecture up to 16-MHz system clock and 8-MHz FRAM access
- 100 µA/MHz active mode and 350 nA standby with RTC and 3.7-pF crystal
- Extended Scan Interface
- 16-channel 12-bit ADC
- Comparator
- Five Timers
- Direct memory access
- 256-bit AES
- 83 GPIOs

02 Hardware

Der Prozessor im Detail



Kapitel 03

Assembler

```

MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER PAGE  2

C000          ORG      ROM+$0000 BEGIN MONITOR
C000 BE 00 70  START    LDS      #STACK

*****
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A

0013        RESETA  EQU      %00010011
0011        CTLREG EQU      %00010001

C003 86 13  INITA   LDA A #RESETA  RESET ACIA
C005 B7 80 04          STA A ACIA
C008 86 11          LDA A #CTLREG  SET 8 BITS AND 2 STOP
C00A B7 80 04          STA A ACIA

C00D 7E C0 F1  JMP     SIGNON   GO TO START OF MONITOR

*****
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* DESTROYS: acc A
* CALLS: none
* DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH    LDA A ACIA    GET STATUS
C013 47          ASR A      SHIFT RDRF FLAG INTO CARRY
C014 24 FA          BCC INCH    RECEIVE NOT READY
C016 B6 80 05          LDA A ACIA+1 GET CHAR
C019 84 7F          AND A #\$7F  MASK PARITY
C01B 7E C0 79          JMP OUTCH   ECHO & RTS

*****
* FUNCTION: INHEX - INPUT HEX DIGIT
* INPUT: none
* OUTPUT: Digit in acc A
* CALLS: INCH
* DESTROYS: acc A
* Returns to monitor if not HEX input

C01E 8D F0  INHEX   BSR     INCH    GET A CHAR
C020 81 30          CMP A #'0  ZERO
C022 2B 11          BMI     HEXERR  NOT HEX
C024 81 39          CMP A #'9  NINE
C026 2F 0A          BLE     HEXRTS GOOD HEX
C028 81 41          CMP A #'A
C02A 2B 09          BMI     HEXERR  NOT HEX
C02C 81 46          CMP A #'F
C02E 2E 05          BGT     HEXERR
C030 80 07          SUB A #7  FIX A-F
C032 84 0F          HEXRTS AND A #$0F  CONVERT ASCII TO DIGIT
C034 39          RTS

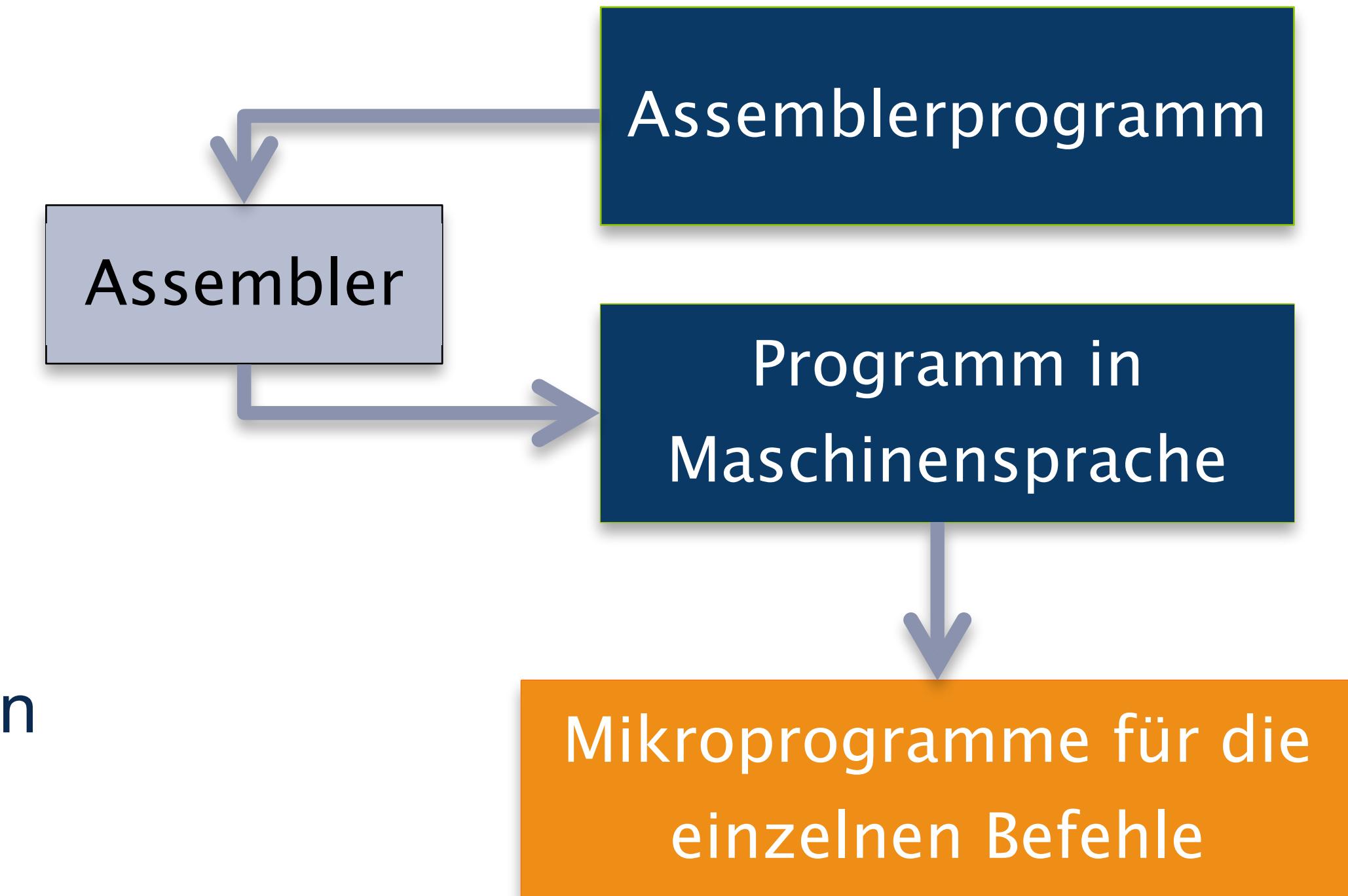
C035 7E C0 AF  HEXERR  JMP     CTRL    RETURN TO CONTROL LOOP

```

03 Assembler

Was ist Assembler?

- Ein Assemblerprogramm ist ein maschinen-nahe Programm.
- Assemblerprogramme sind immer spezifisch auf einen oder wenige Prozessortypen ausgerichtet.
- Der Assembler übersetzt das Assemblerprogramm in die Maschinensprache.
- Konkrete Befehle mit ihrer Kodierung und den endgültigen Speicheradressen.



03 Assembler

Ablauf eines Programms

- I. Befehl von Speicherstelle (\leftarrow Program Counter) laden,
- II. Befehl decodieren,
- III. Speicherstelle vom Folgebefehl ermitteln (\rightarrow Program Counter),
- IV. Ersten Operanden laden,
- V. Zweiten Operanden laden,
- VI. Ausführen,
- VII. Ergebnis in Register oder Speicher schreiben.

03 Assembler

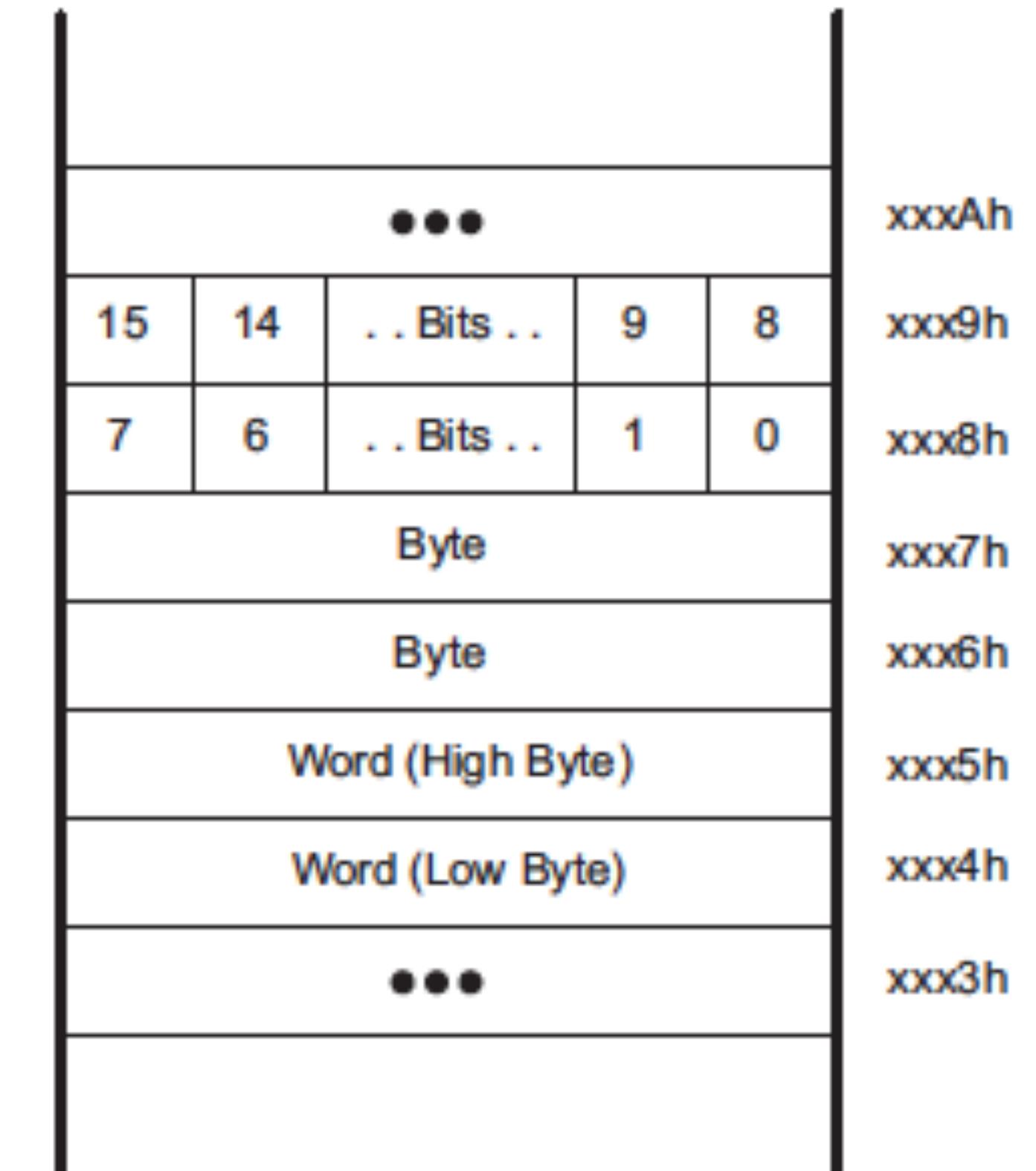
Befehle I

3 Befehlstypen:

- Dual-Operand
- Single-Operand
- Jump-Befehle
- + Befehle zur Verwaltung

Der MSP kann zwischen zwei Verarbeitungsbreiten unterscheiden:

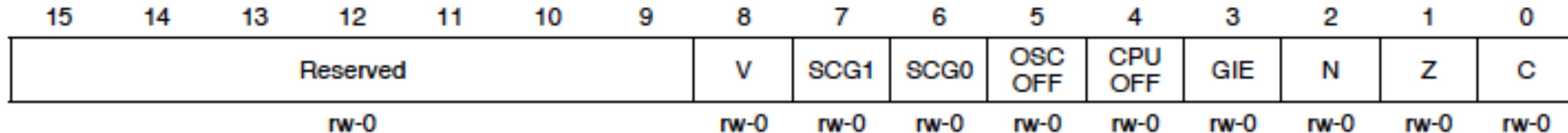
- Bytes (8 Bit) **.b** und
- Wörter (16 Bit) **.w**



Operanden im Speicher

03 Assembler **Befehle II**

Arithmetische Befehle und Logische Operationen verändern Flags im Statusregister:



- C: Carry
- Z: Zero
- N: Negative
- V: Overflow

03 Assembler

Dual-Operand Befehle

Mnemonic		Operation	V	N	Z	C
MOV	src,dst	dst = src	no	no	no	no
ADD	src,dst	dst = src + dst	yes	yes	yes	yes
ADDC	src,dst	dst = src + dst + C	yes	yes	yes	yes
SUB	src,dst	dst = dst - src	yes	yes	yes	yes
SUBC	src,dst	dst = dst - src - C	yes	yes	yes	yes
CMP	src,dst	dst - src	yes	yes	yes	yes
DADD	src,dst	dst = src + dst + C	yes	yes	yes	yes
BIT	src,dst	src & dst	clear	yes	yes	yes
BIC	src,dst	dst = !src & dst	no	no	no	no
BIS	src,dst	dst = src dst	no	no	no	no
XOR	src,dst	dst = src ^ dst	yes	yes	yes	yes
AND	src,dst	dst = src & dst	clear	yes	yes	yes

03 Assembler

Single-Operand Befehle

Mnemonic	Operation	V	N	Z	C
RRC dst	rotate through carry (right)	yes	yes	yes	yes
RRA dst	dst = dst >> 1 (arithmetically)	clear	yes	yes	yes
PUSH src	SP = SP -2 ; @SP = PC + 2	no	no	no	no
SWPB dst	Bits 15 to 8 <-> Bits 7 to 0	no	no	no	
CALL dst	SP = SP -2 ; @SP = PC + 2	no	no	no	no
RET	return from subroutine	no	no	no	no
RETI	return from interrupt	yes	yes	yes	yes
SXT dst	Bit 8 to Bit 15 = Bit 7	clear	yes	yes	yes

03 Assembler **Sprungbefehle**

Mnemonic

JEQ/JZ

Label

JNE/JNZ

Label

JC

Label

JNC

Label

JN

Label

JGE

Label

JL

Label

JMP

Label

Operation

Jump to label if zero bit is set

Jump to label if zero bit is reset

Jumpf to label if carry bit is set

Jump to label if carry bit is reset

Jump to label if negative bit is set

Jump to label if $(N \wedge V) = 0$

Jump to label if $(N \wedge V) = 1$

Jump to label unconditionally

03 Assembler

Spezielle Befehle

Mnemonic	Operation
CLRC	Clear carry bit
CLRN	Clear negative bit
CLRZ	Clear zero bit
SETC	Set carry bit
SETN	Set negative bit
SETZ	Set carry bit
DINT	Disable Interrupt
EINT	Enable Interrupt

03 Assembler

Adressierungsarten (Auswahl)

Register-Register:

- **Mov R5, R6**
⇒ Kopiert den Inhalt von R5 nach R6

Immediate:

- **Mov.w #4AF3h, R6**
- **Mov.b #01010111b, R6**
⇒ Speichert Direktwert in R6 (R6 = 0x4AF3)

Absolute Adressierung

- **Mov &4AF3h, R6**
⇒ Speichert den Wert **an** der Speicherstelle in R6

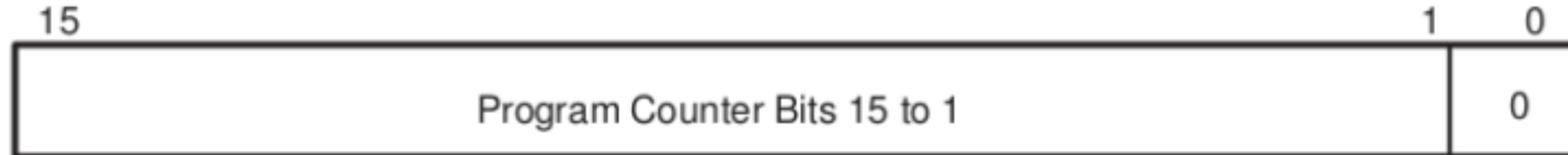
03 Assembler

Wort oder Byte?

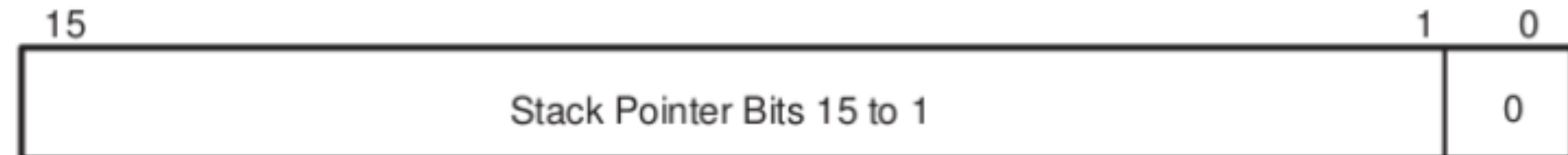
- CPU arbeitet mit Wörtern.
- Hat ein Befehl kein Suffix oder den Suffix **.w** wird mit Wörtern gearbeitet:
 - Speicherzugriffe auf 16-Bit und somit zwei Speicherstellen ausgerichtet,
 - Flags werden durch Wort-Operationen aktualisiert.
- Ist der Suffix **.b** wird mit Bytes gearbeitet:
 - Speicherzugriffe nur auf einzelne Speicherstellen,
 - Flags werden durch Byte-Operationen aktualisiert.

03 Assembler **Register (MSP430)**

R0: Program Counter (Beinhaltet Speicherstelle des nächsten Befehls)



R1: Stack Pointer



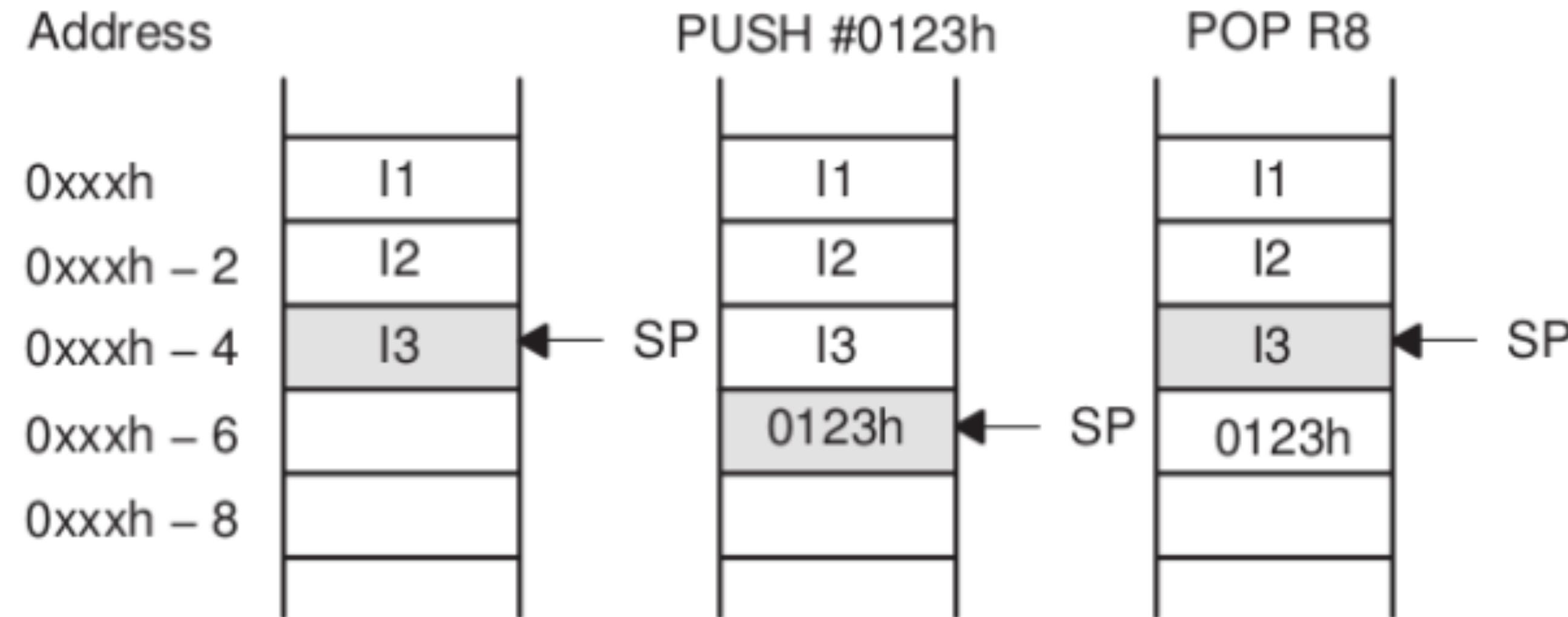
R2: Status Register

R3: Constant Generator

R4-R15: General Purpose Register

03 Assembler **Stack und Unterprogramme**

- Der Stack-Pointer zeigt auf die Speicherstelle für den nächsten Wert und arbeitet nach dem LIFO-Prinzip.
- Auf dem Stack können temporäre Daten gespeichert (**PUSH**) und geladen werden (**POP**).
- Der Stack sichert Rücksprungadressen bei Aufrufen von **Unterprogrammen**.



03 Assembler

Input/Output

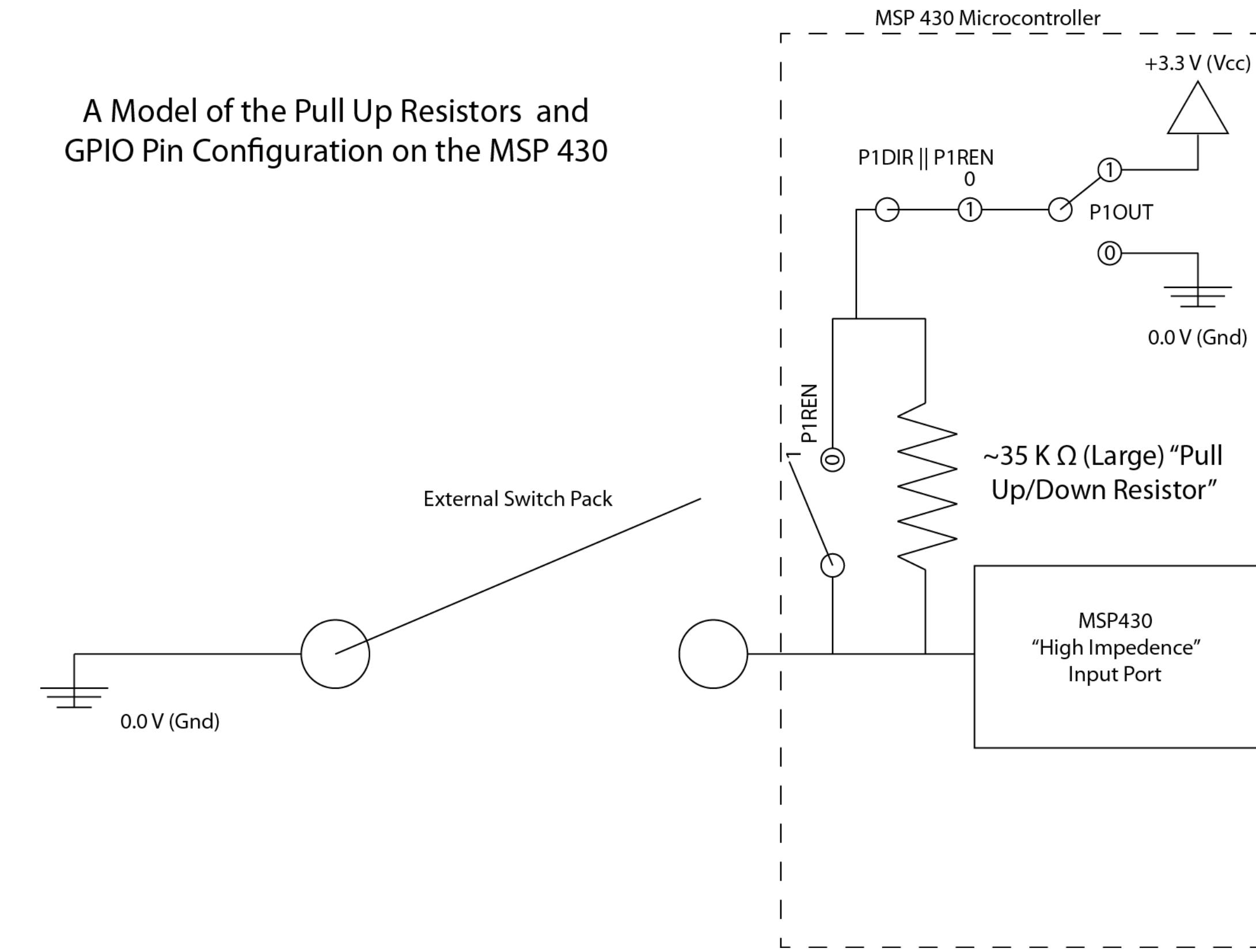
Register:

- **P1DIR**: Richtung (1 steht für Ausgang)
- **P1IN**: Eingangswert
- **P1Out**: Ausgabewert

Wichtige Befehle:

- **BIS** – Bit Set
- **BIC** – Bit Clear
- **BIT** – Bit Test

A Model of the Pull Up Resistors and GPIO Pin Configuration on the MSP 430



03 Assembler

Ein einfaches Beispiel

```

.cdecls C,LIST,"msp430.h" ; cdecls tells assembler to allow the c header
;-----
; Main Code
;-----

        .text                      ; program start
        .global RESET               ; define entry point

RESET      mov.w  #__STACK_END,SP      ; initialize stack pointer
          mov.w  #WDTPW+WDTTHOLD,&WDTCTL ; stop watchdog timer
          mov.w  #0000h,&PM5CTL0
          bis.b  #00000001b,&PADIR    ; make P1.0 and P1.6 output
          bis.b  #00000001b,&PAOUT

Mainloop   ...

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

.sect ".reset" ; MSP430 RESET Vector
.short RESET
;-----
; Stack Pointer definition
;-----

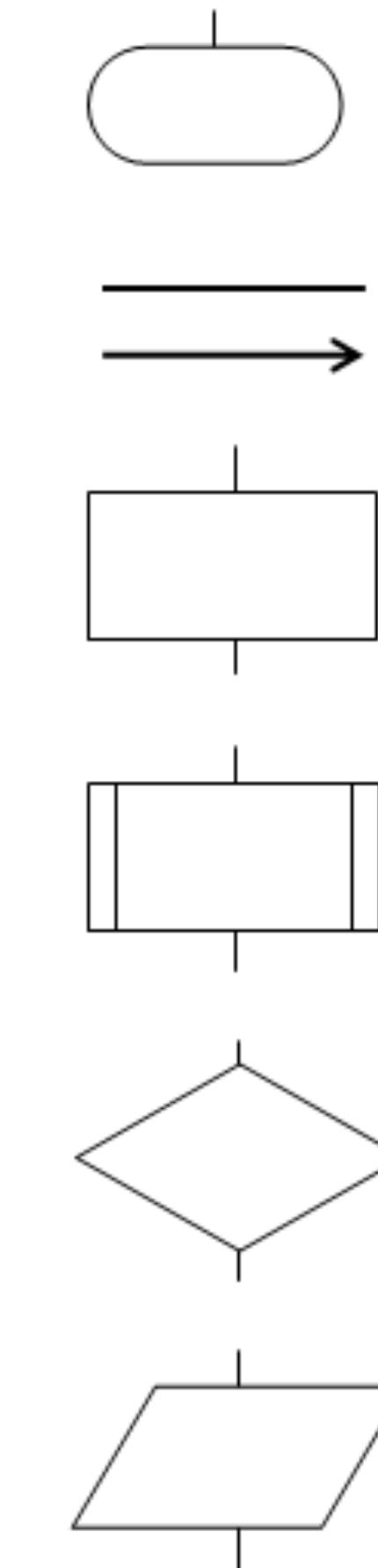
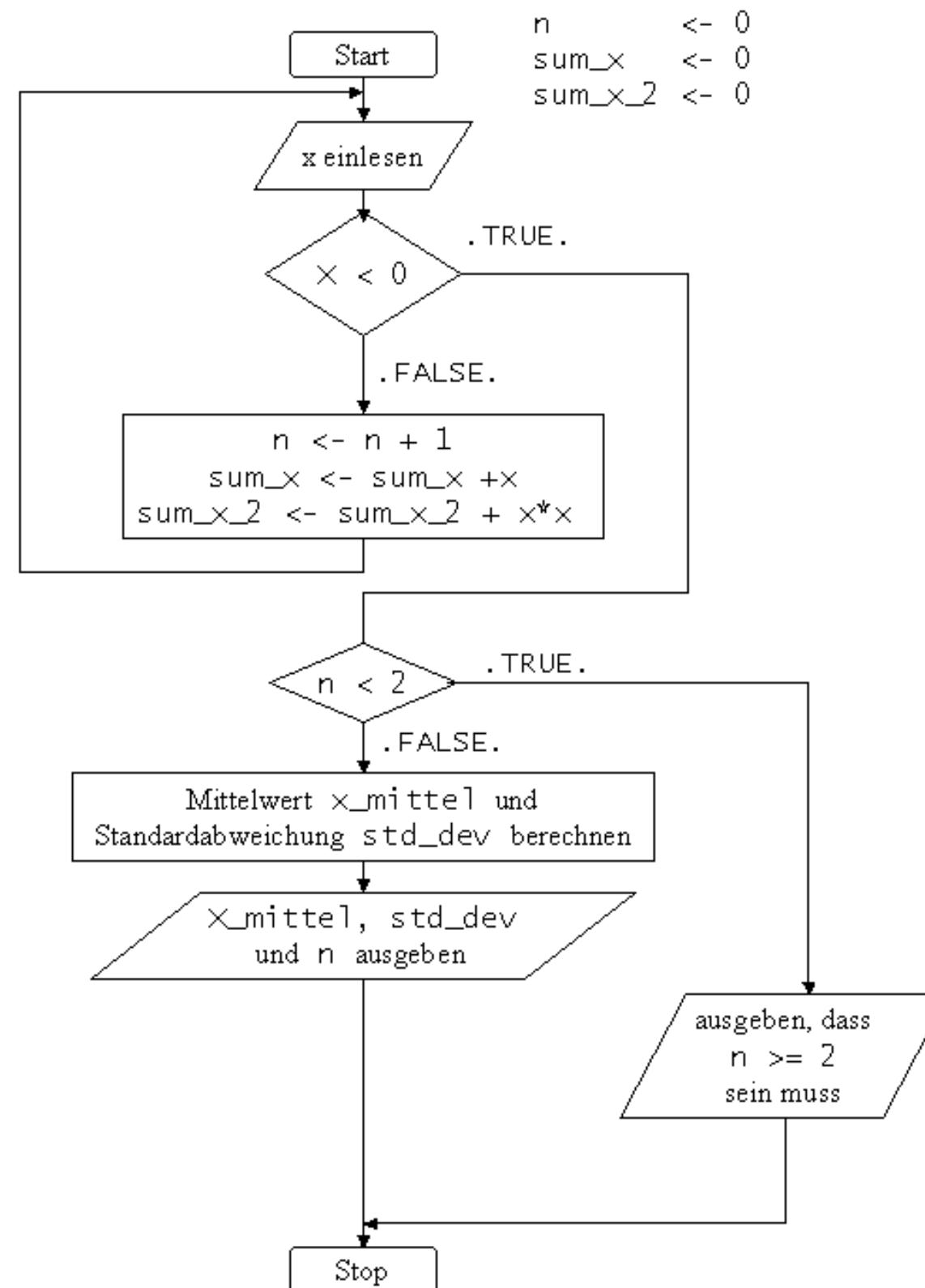
.global __STACK_END
.sect .stack

```

03 Assembler

Programmablaufplan (PAP)

Einfaches Flussdiagramm zur besseren Übersicht über den Algorithmus...



03 Assembler

Erste Schritte auf dem MSP (in Assembler)...

1. Blinkende LED (**blink.asm**):

Die **rote LED** soll in einem Takt von ca. 1Hz blinken.

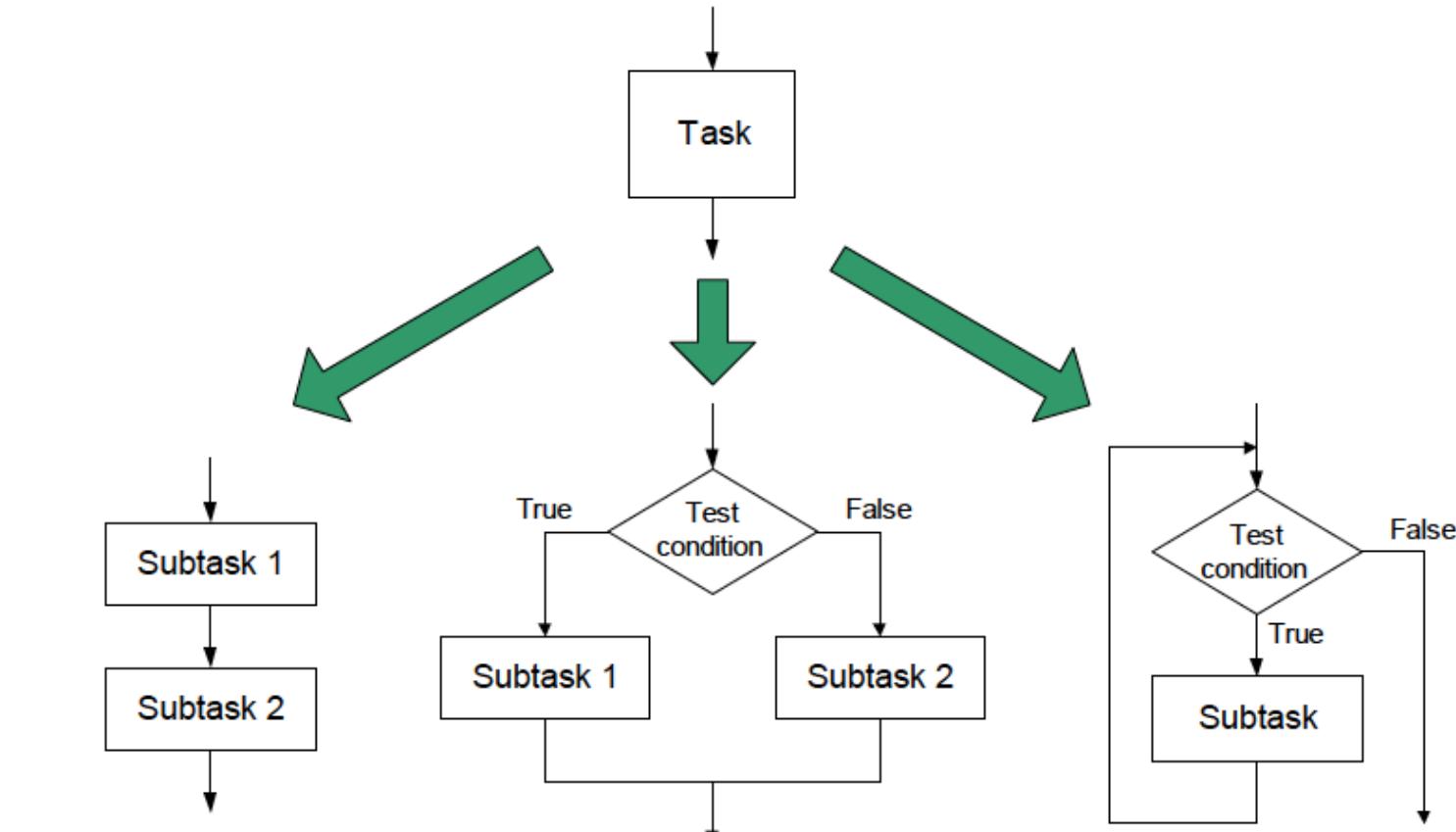
2. Einfache Ein- und Ausgaben (**io.asm**):

Wenn der Taster S2 betätigt wird, leuchtet die grüne LED und wenn der Taster nicht gedrückt wird, leuchtet die rote LED.



Kapitel 04

Adressierung, Direktiven und Basiskonstrukte



04 Adressierung, Direktiven und Basiskonstrukte

Übersicht MSP430 Adressierungen

- Register-Modus:
 - **mov.b R5,R6**
 - Bewegt Inhalt von **R5** nach **R6**.
- Immediate-Modus:
 - **mov.b #FFh,R5**
 - Bewegt **FFh** nach **R5**.
- Index-Modus:
 - **Mov.b 3(R5),R6**
 - Inhalt von Speicheradresse **M(3+R5)** nach **R6**.

04 Adressierung, Direktiven und Basiskonstrukte

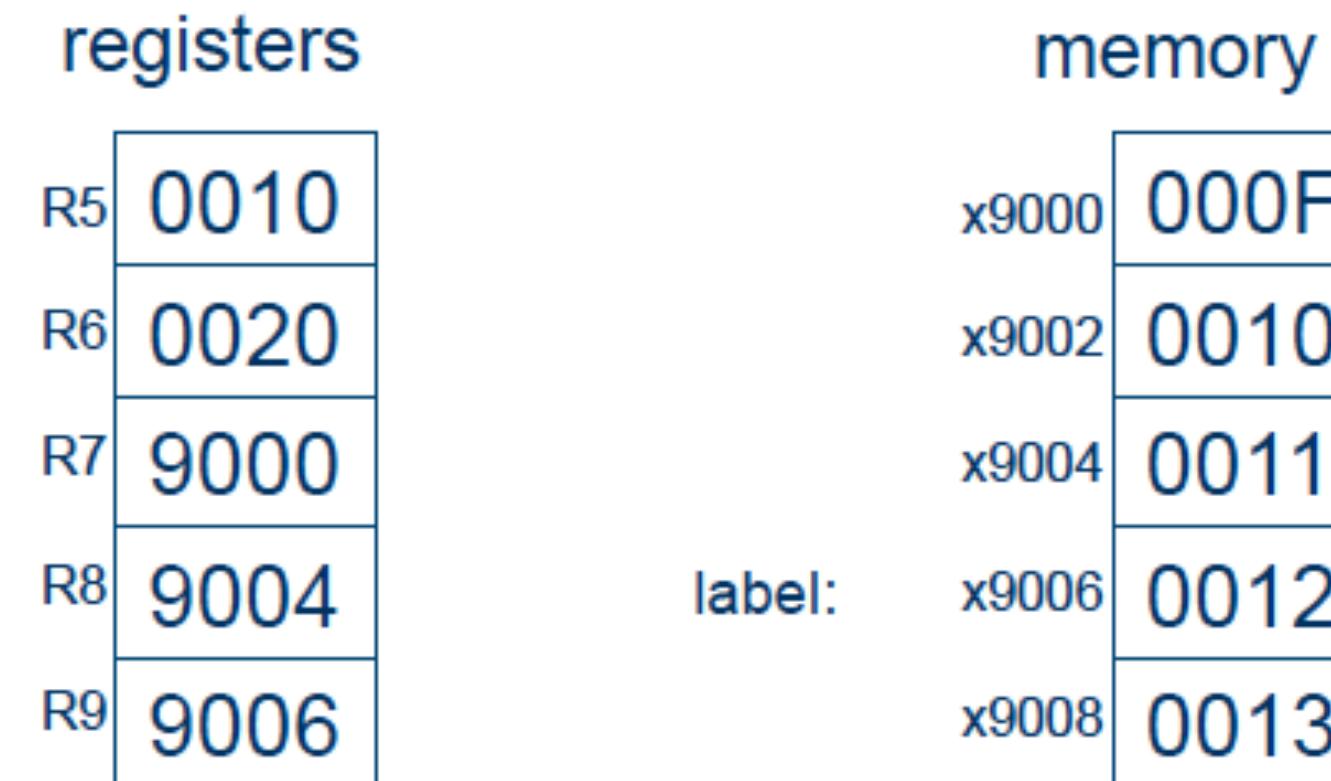
Übersicht MSP430 Adressierungen

- Absoluter Modus:
 - `mov.b &label, R6`
 - Bewegt Inhalt von Speicheradresse **M(label)** nach **R6**.
- Indirekter Modus:
 - `mov.b @R5,R6`
 - Bewegt Inhalt von Speicheradresse **M(R5)** nach **R6**.
- Indirekt und Auto-Inkrement
 - `mov.b @R5+,R6`
 - Bewegt Inhalt von Speicheradresse **M(R5)** nach **R6** und inkrementiert **R5**.

04 Adressierung, Direktiven und Basiskonstrukte

Beispiele Adrssierung

- **Mov.w R5,R6** Inhalt von R5 nach R6: R6=0010.
- **Mov.w @R7,R8** Inhalt R7 ist Speicherstelle: Bewege Inhalt von M(9000) nach R8=000F.
- **Mov.w &9004h, R6** Inhalt von Speicherstelle nach R6: R6=0011.
- **Mov.w 2(R7), R9** Inhalt von R7+2 ist Adresse (9002): R9=0010.
- **Mov.w #9000h,R7** Bewege 9000 nach R7: R7=9000.
- **Mov.w label,R7** Bewegt Inhalt von Adresse 9006 nach R7: R7=0012.
- **Mov.w #label,R8** Bewegt Wert von label nach R8: R8=9006.



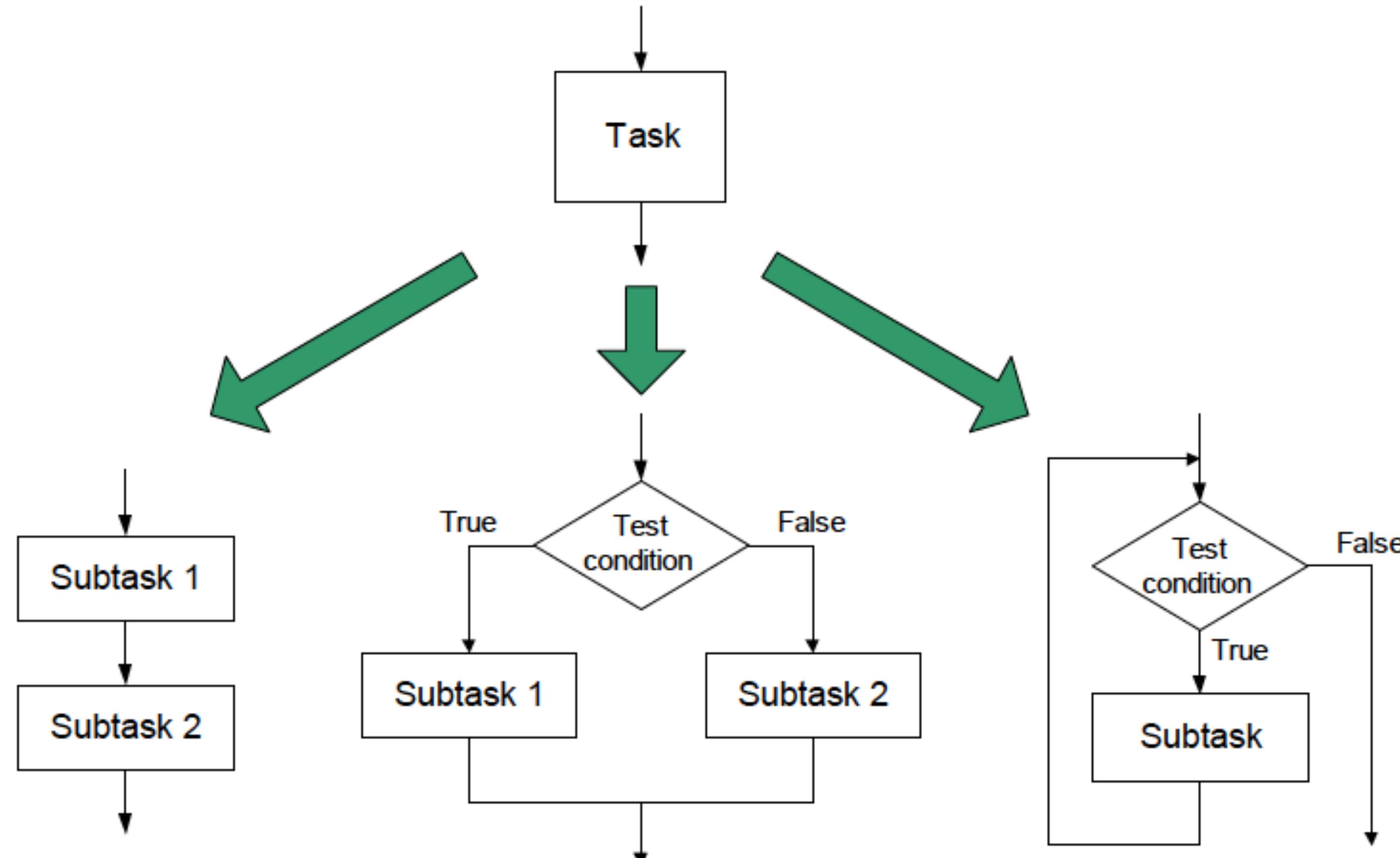
04 Adressierung, Direktiven und Basiskonstrukte

Assemblerdirektiven

- **.text**
 - Folgender Assemblercode wird in den Bereich für ausführbaren Programmcode geschrieben.
- **.end**
 - Ende des Programmes.
- **.sect label**
 - Schreibt folgende Befehle in den Bereich mit dem vordefinierten Namen label.
- **.data**
 - Anfang des Datensegmentes.
- **.global label**
 - Bereich mit dem Namen label ist Einstiegspunkt.
- **.byte wert**
 - Schreibt Bytes (**.byte**), Wörter (**.word**) oder Strings (**.string**) in das Segment.
- **symbol .set wert**
 - Definiert ein Symbol mit einen dazugehörigen Wert.
- **.cdecls [optionen] name**
 - Ermöglicht die Einbindung von C-Headern.

04 Adressierung, Direktiven und Basiskonstrukte

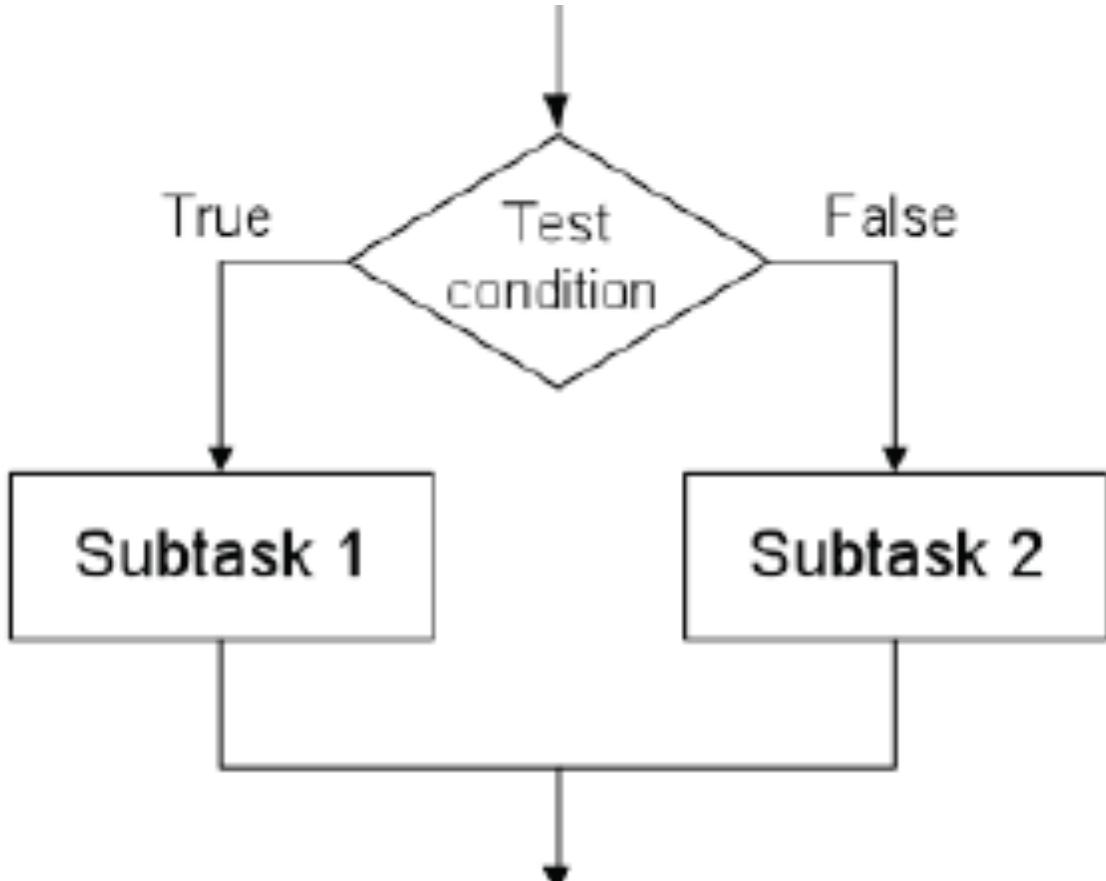
Basiskonstrukte



04 Adressierung, Direktiven und Basiskonstrukte

Basiskonstrukte: If-Then-Else

```
if (Button == 0) {
    GreenOn();
} else {
    GreenOff();
}
```



<code>bit.b #0x08, &P1IN</code> <code>jnz OFF</code>	<code>...</code>
<code>ON bis.b #0x01, &P1OUT</code> <code>jmp NEXT</code>	<code>ON</code>
<code>OFF bic.b #0x01, &P1OUT</code> <code>NEXT . . .</code>	<code>OFF</code>

04 Adressierung, Direktiven und Basiskonstrukte

Basiskonstrukte: While-Loop

```
while (TRUE) {  
    LED_ON();  
    delay();  
    LED_OFF();  
    delay();  
}
```

```
...  
LOOP  bis.b  #0x01, &P1OUT  
      call   #delay  
      bic.b #0x01, &P1OUT  
      call   #delay  
      jmp    LOOP  
...
```

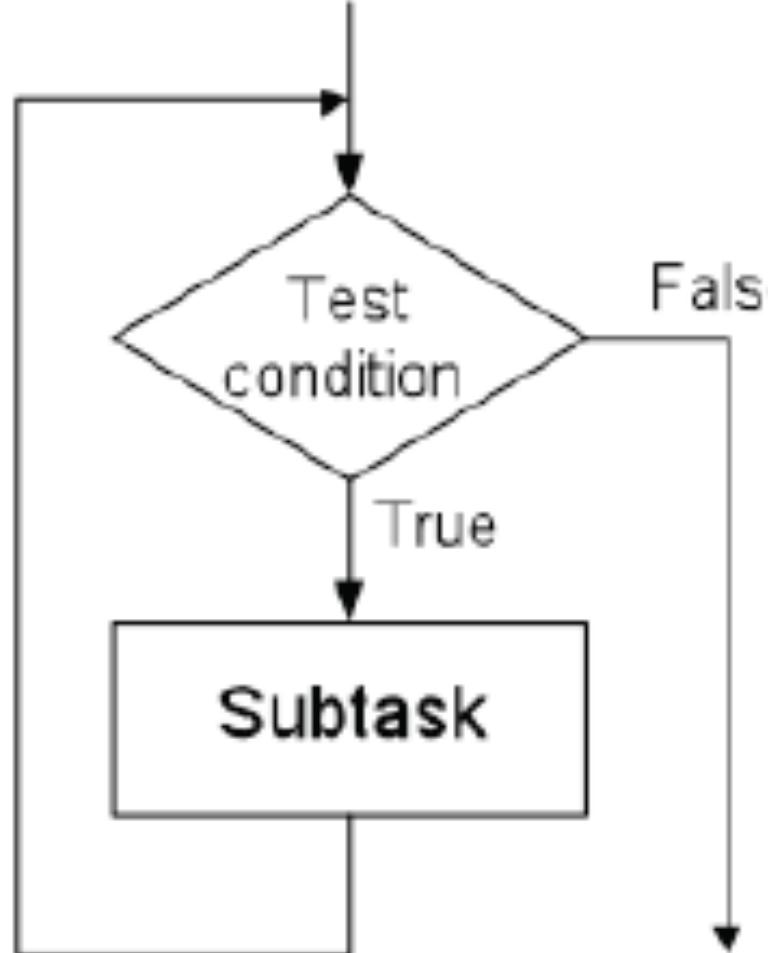
04 Adressierung, Direktiven und Basiskonstrukte

Basiskonstrukte: For-Loop

```

for(int i=0; i<10; i++) {
    LED_ON();
    delay();
    LED_OFF();
    delay();
}

```

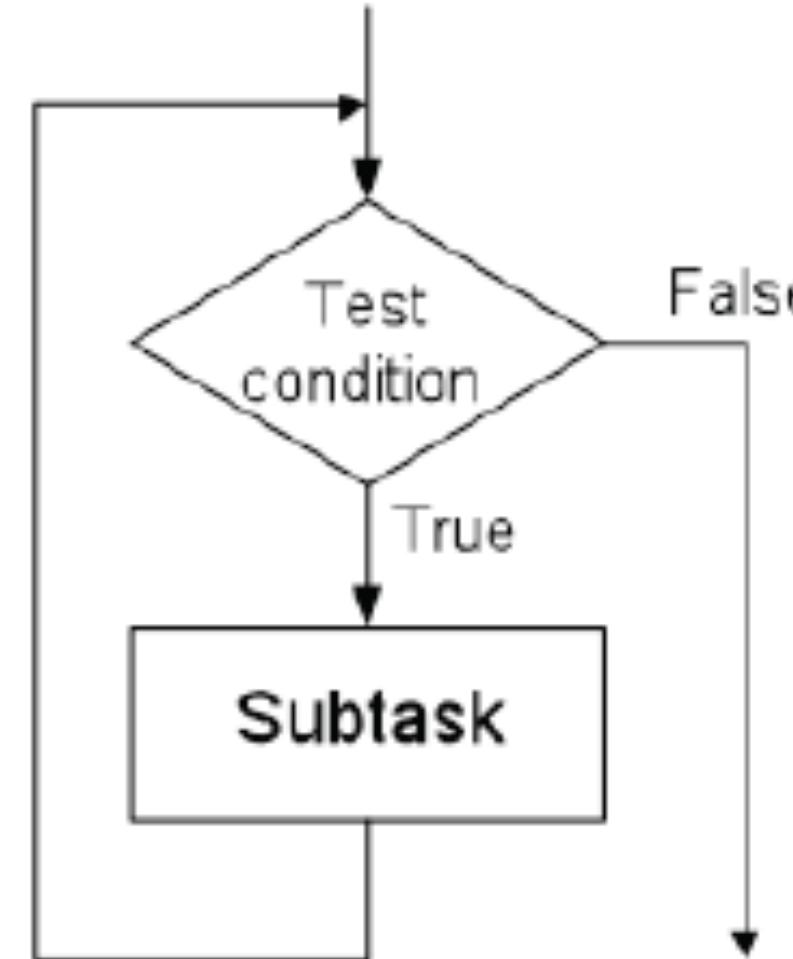


LOOP	<code>... mov.w #0,R15 cmp.w #10,R15 jge DONE bis.b #0x01, &P1OUT call #delay bic.b #0x01, &P1OUT call #delay add.w #1,R15 jmp LOOP ...</code>
DONE	

04 Adressierung, Direktiven und Basiskonstrukte

Basiskonstrukte: For-Loop II

```
for(int i=10; i>0; i--) {
    LED_ON();
    delay();
    LED_OFF();
    delay();
}
```



LOOP

DONE

```
...
mov.w #10,R15
bis.b #0x01, &P1OUT
call #delay
bic.b #0x01, &P1OUT
call #delay
dec.w R15
jnz LOOP
...
```

04 Adressierung, Direktiven und Basiskonstrukte

Aufgaben

Berechnungen im RAM (`calc.asm`):

- Im Speicher sind im Datensegment unter dem Label **bytes** Datenbytes hinterlegt.
- Unter dem Label **count** ist die Anzahl (ein Byte) der Datenbytes gespeichert.
- Das Programm soll die Datenbytes mit sinnvollen Adressierungsarten lesen und aufsummieren.
- Das Ergebnis soll an die Speicheradresse mit dem Label **result** abgelegt werden.
- Liegt ein ausgehender Übertrag vor, ist das Ergebnis falsch und es soll der Wert **0xFF** als Ergebnis geschrieben werden. Des weiteren soll das falsche Ergebnis durch die **rote LED** signalisiert werden. Ist das Ergebnis hingegen korrekt soll die **grüne LED** leuchten.



04 Adressierung, Direktiven und Basiskonstrukte

MSP430 – Adressierungen

Vier Arten der Adressierung für Quelladressen:

- Rs – Register
- x(Rs) — Indexed Register
- @Rs — Register Indirect
- @Rs+ — Indirect Auto-increment

Zwei Arten für Zieladressen:

- Rd — Register
- x(Rd) — Indexed Register

04 Adressierung, Direktiven und Basiskonstrukte

Maschinencode des MSP

```

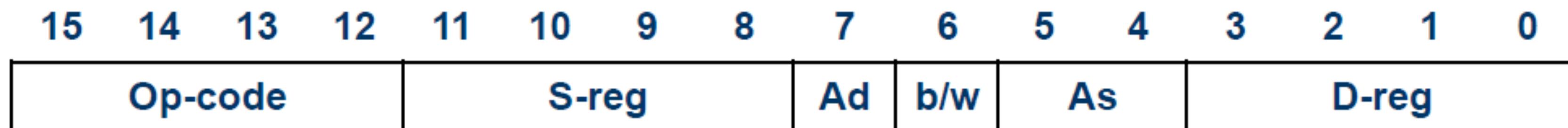
37      daten    .word          0x90D6, 0x145A, 0x0000
0x0200:  90D6 145A 0000
-----
53      Mainloop  mov.w   #daten, R4
        Mainloop:
0xf80a:  4034 0200          MOV.W   #0x0200,R4
54              mov.w   &daten, R5
0xf80e:  4215 0200          MOV.W   &daten,R5
56              mov.w   2(R4), R6
0xf812:  4416 0002          MOV.W   0x0002(R4),R6
57              add.w   R5,R6
0xf816:  5506                ADD.W   R5,R6
59              rra.w   R6
0xf818:  1106                RRA     R6
60              mov.w   R6,4(R4)
0xf81a:  4684 0004          MOV.W   R6,0x0004(R4)
62              jmp     Mainloop
f81e:  3FF5                JMP     (Mainloop)

```

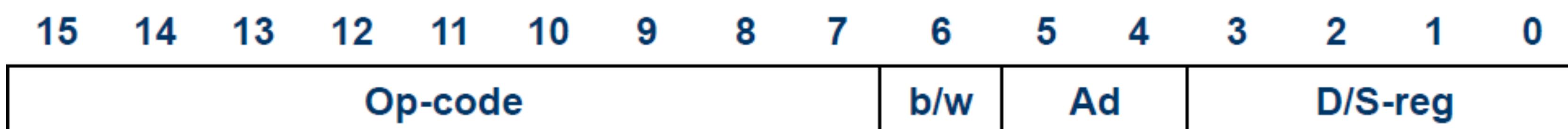
04 Adressierung, Direktiven und Basiskonstrukte

Drei unterschiedliche Befehlsformate

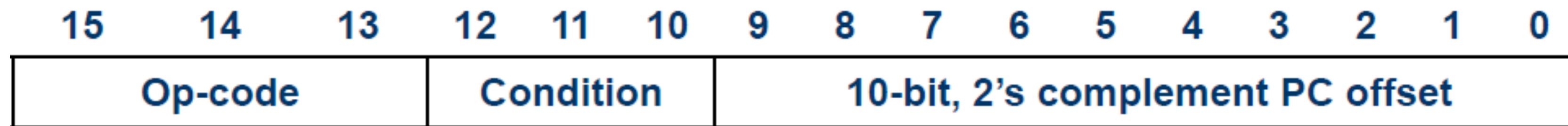
- Befehle mit zwei Operanden:



- Befehle mit einem Operanden:



- Sprungbefehle:



04 Adressierung, Direktiven und Basiskonstrukte

Komponenten eines Befehlswortes

src: Quelloperand nach As und S-reg:

- As: Adressierungsmodus für Quelle
- S-reg: Quellregister

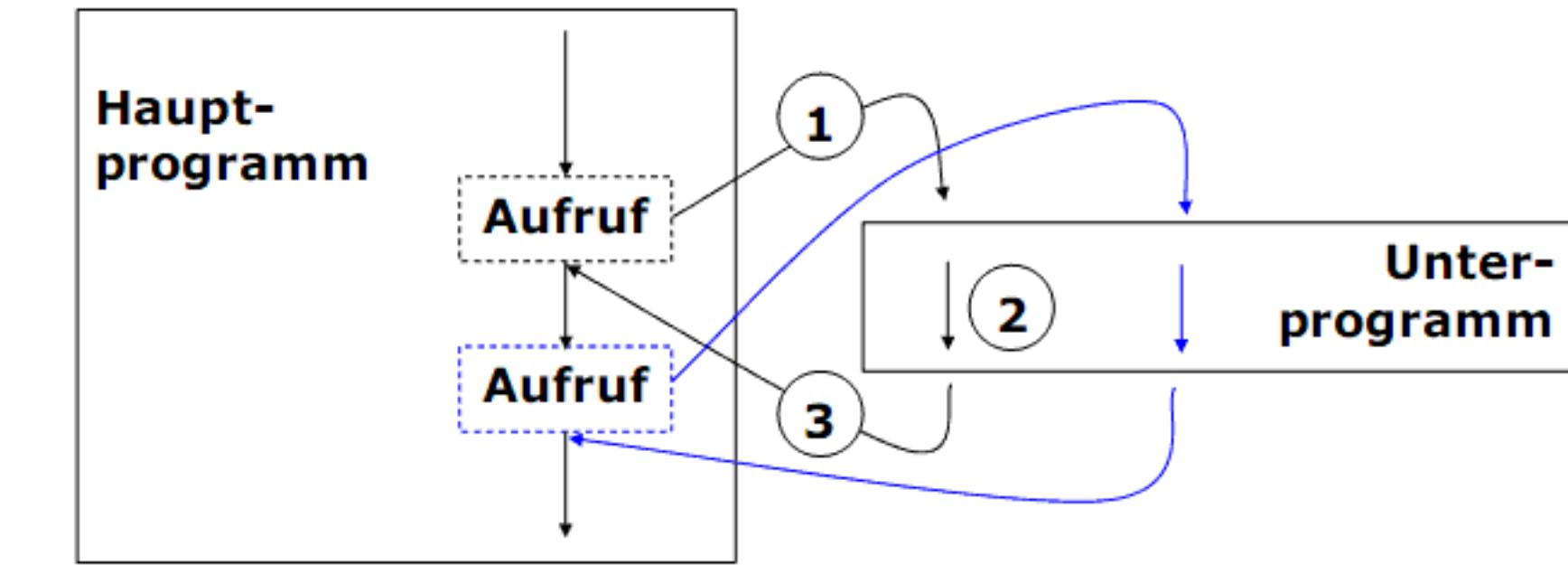
dst: Zieloperand, as defined in Ad and D-reg:

- Ad: Adressierungsmodus für Ziel
- D-reg: Zielregister

b/w: Wort- oder Byte

Kapitel 05

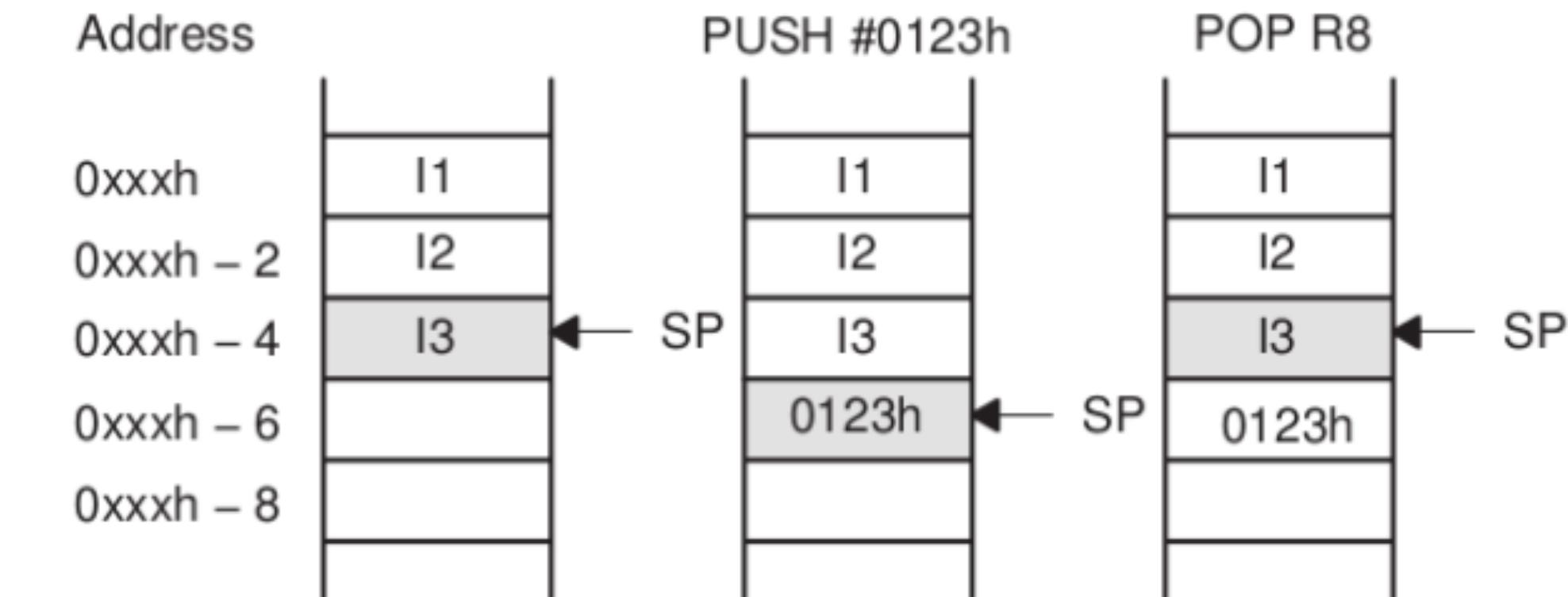
Unterprogramme, Interrupts und Timer



05 Unterprogramme, Interrupts und Timer

Unterprogramme

- Mit dem Befehl **CALL #Label** wird ein Sprung an das Label ausgeführt und der alte Inhalt vom PC wird auf dem Stack gesichert.
- Register die modifiziert werden sollten ebenfalls mit dem **PUSH** Befehl auf dem Stack gesichert werden.
- Am Ende des Programms müssen die gesicherten Register wiederhergestellt mit **POP** werden.
- Danach folgt der Rücksprung mit dem Befehl **RET**.



05 Unterprogramme, Interrupts und Timer

Aufgabe – Funktionen

Blinkende LED mit Unterprogrammen (**blink_call.asm**):

1. Das Programm **blink.asm** ist so zu erweitern, dass die Zeitschleifen über den Aufruf von Unterprogrammen realisiert werden. Dabei ist darauf zu achten, dass die Unterprogramme Registerinhalte, die verändert werden, beim Aufruf sichern und vorm Rücksprung wiederherstellen.
2. Wie kann dem Unterprogramm ein Wert wie zum Beispiel die Zeit (Anzahl der Schleifendurchläufe) übergeben werden?



05 Unterprogramme, Interrupts und Timer

Aufgabe – Summe

1. Summe natürlicher Zahlen (**summe.asm**):

Es ist ein Programm zu entwerfen, welches die Summe der natürlichen Zahlen bis einschließlich einer Zahl n berechnet.

Beispiel: n=5

Ergebnis: 1+2+3+4+5=15

Daten im RAM ablege mit:

```
.data  
daten .word 0x0001, 0x0002, 0x0003, 0x0004, 0x0005
```

Der Wert n ist aus dem Speicher zu lesen und das Ergebnis ebenfalls in den Speicher zu schreiben. Während der Berechnung soll die **rote LED** leuchten, sobald das Ergebnis im Speicher steht soll die grüne LED leuchten.

2. Das Programm ist so zu erweitern, dass es mit Wörtern zu je 2 Byte arbeitet und entsprechend dazu den Befehl **ADDC** nutzt, um so die Verarbeitungsbreite auch auf 32 Bit zu erweitern.



05 Unterprogramme, Interrupts und Timer

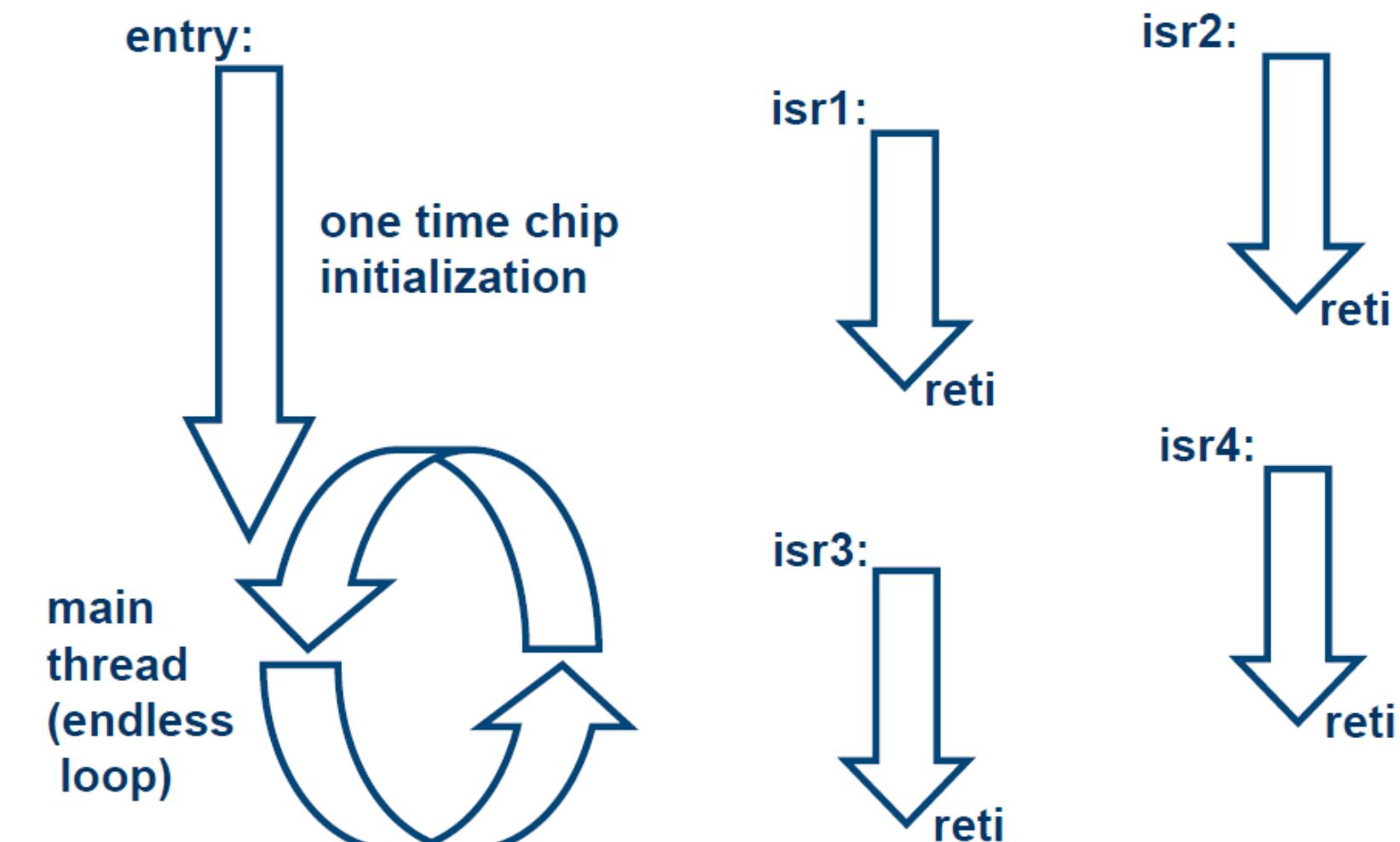
Interrupts I

- Die explizite permanente Abfrage eines Signales wird als **Polling** bezeichnet und erfordert CPU-Laufzeit.
- Der Programmablauf kann durch Signale von außen direkt ohne deren explizite Abfrage unterbrochen werden.
- Diese Unterbrechungen ohne Polling werden als **Interrupts** bezeichnet und können auftreten bei:
 - Resets,
 - Hardwarefehlern,
 - Watchdogs,
 - Timersignalen,
 - Änderung eines Eingangs,
 - ...

05 Unterprogramme, Interrupts und Timer

Interrupts II

- Interrupts können für bestimmte Programmabschnitte deaktiviert werden.
- Es kann nicht vorausgesagt werden, wann und vor allem wo (Position im Programm) ein Interrupt auftritt.
- Interrupts werden für Aktivitäten und Programmabschnitt mit höherer Priorität eingesetzt (z.B. Reset).
- Diese Programmabschnitte, welche auf einen Interrupt reagieren heißen **Interrupt Service Routine – ISR**.
- Ein Interrupt kann eine CPU aus dem Sleep-Modus (Energiesparmodus) aufwecken.



05 Unterprogramme, Interrupts und Timer

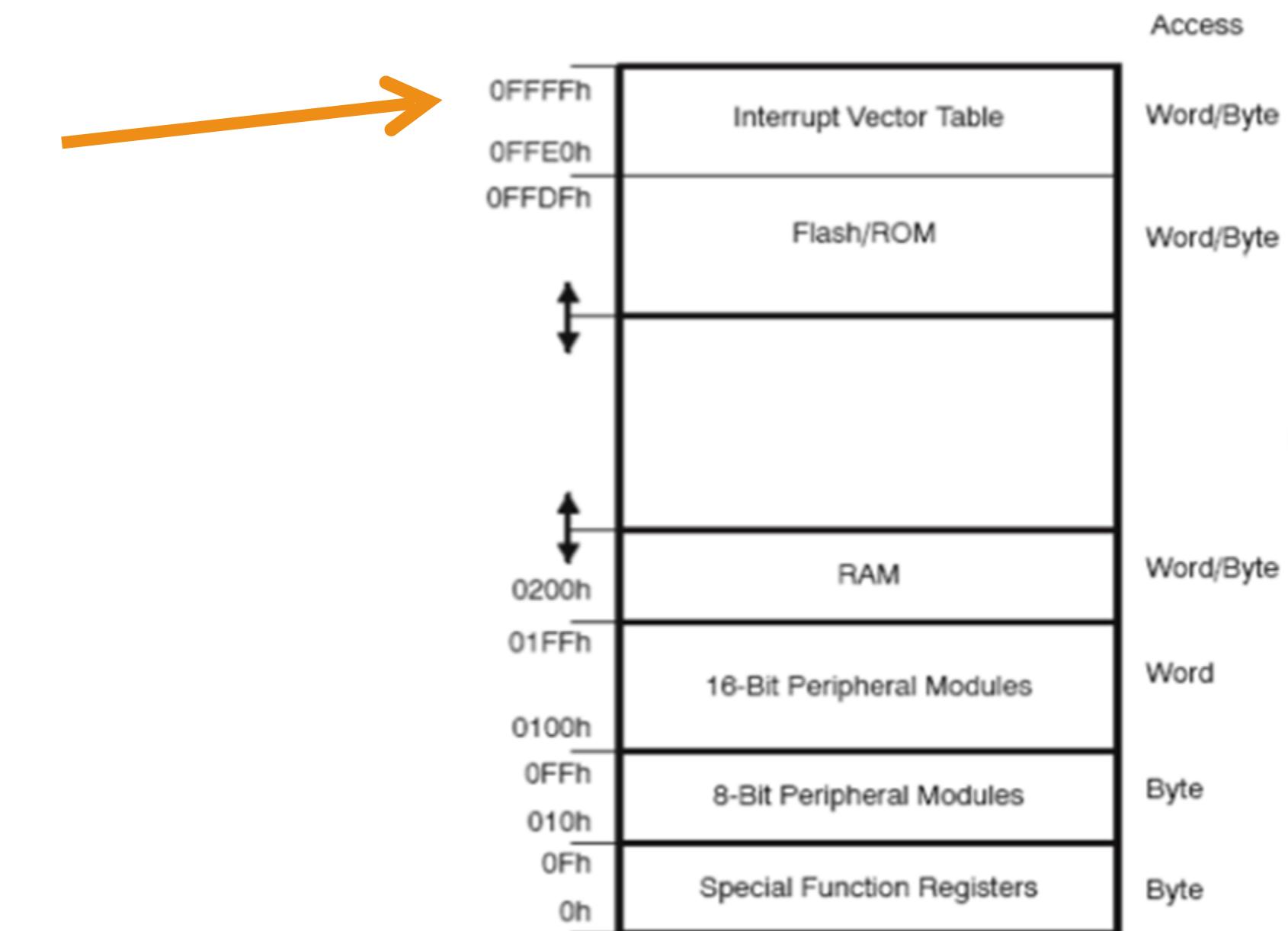
Interrupts III

- Jeder Interrupt hat ein Flag, welches gesetzt wird, sobald der Interrupt eintritt.
- Die meisten Interrupts sind **maskierbar**.
- Ist ein Interrupt maskierbar, kann er aktiviert bzw. deaktiviert werden.

05 Unterprogramme, Interrupts und Timer

Interrupts – Interruptvektor

- Die CPU muss wissen, an welcher Speicherstelle nach einem Interrupt der nächste Befehl zu finden ist, um entsprechend auf den Interrupt reagieren zu können.
- Die Adresse eines Programmes, welches auf eine ISR reagiert, heißt **Interruptvektor**.
- Die Interruptvektoren des MSP430 liegen im **Vector Table** am Ende des Speichers.



05 Unterprogramme, Interrupts und Timer

Interrupts – Sprungziele (MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx)

Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
Reset: power up, external reset watchdog, FRAM password	... WDTIFG FRCTLPW	... Reset	... 0FFEh	... Highest
System NMI: JTAG Mailbox	JMBINIFG, JMBOUTIFG	(Non)maskable	0FFFCh	
User NMI: NMI oscillator fault	... NMIIFG OFIFG	... (Non)maskable (Non)maskable	... 0FFFAh	...
Device specific			0FFF8h	
...		
Watchdog timer	WDTIFG	Maskable
...		
Device specific				
Reserved		Maskable		Lowest

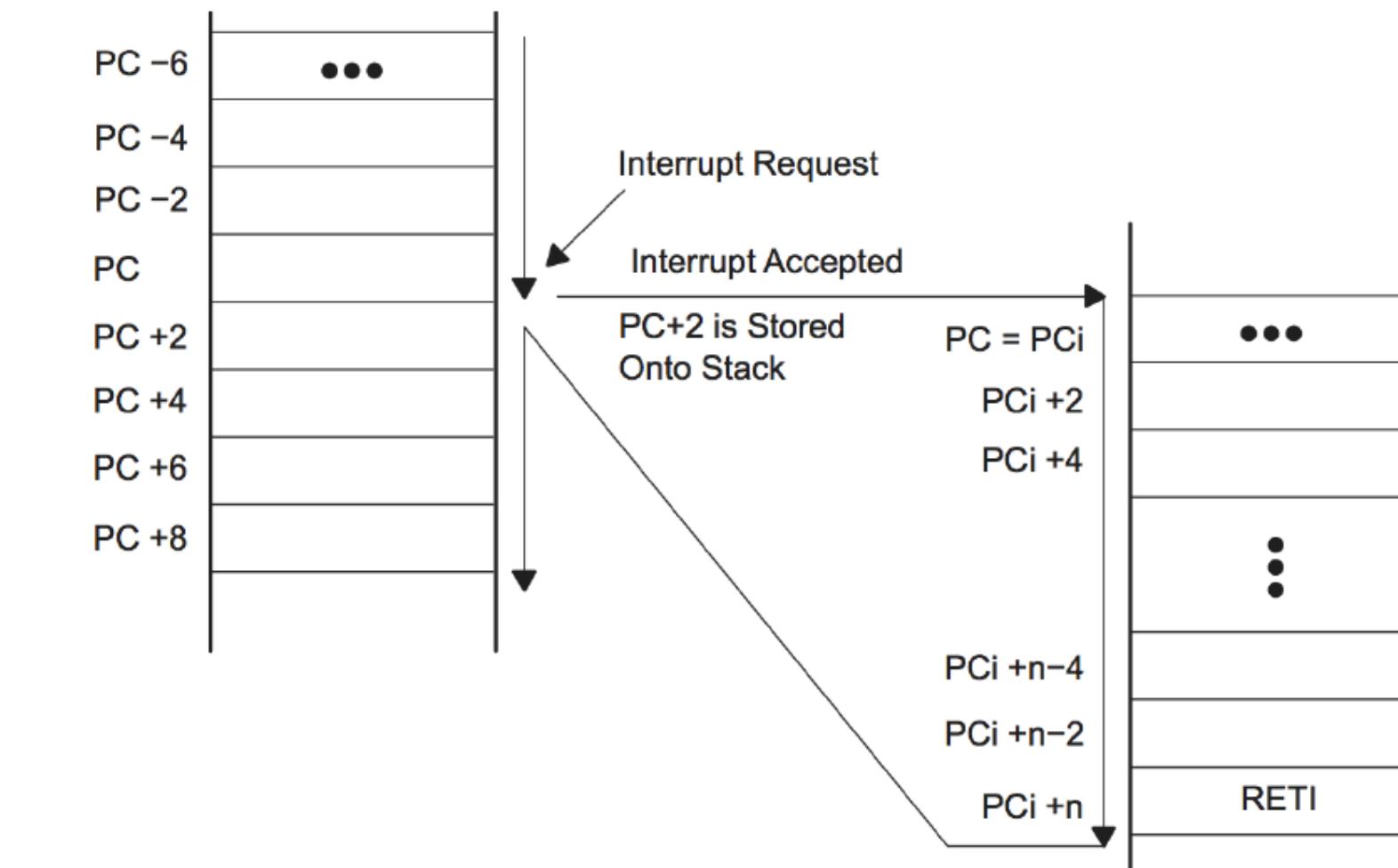
User's Guide — MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family

SLAU367H, S.46 – October 2012

05 Unterprogramme, Interrupts und Timer

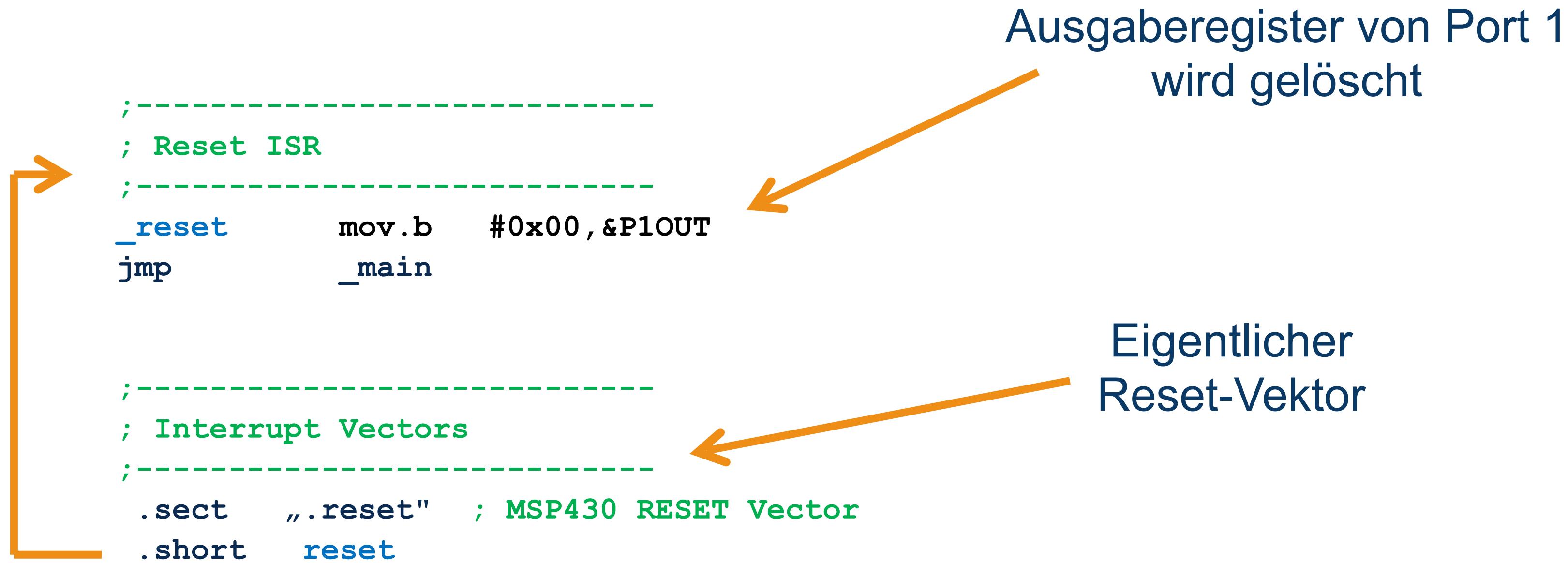
Ablauf eines Interrupts I

- Der Ablauf ist ähnlich zu dem eines Unterprogrammaufrufes:
 - Die Rücksprungadresse wird auf dem Stack gesichert.
 - Der Inhalt des Statusregisters ist ebenfalls auf den Stack zu sichern.
 - Die Rückkehr aus einer ISR erfolgt mit dem Befehl **RETI**.
- Im Gegensatz zu Unterprogrammen werden bei einem Interrupt automatisch das Statusregister gezielt verändert:
 - Der Sleep-Modus der CPU wird deaktiviert und
 - Weitere Interrupts werden deaktiviert.



05 Unterprogramme, Interrupts und Timer

Beispiel Reset



```
;-----  
; Reset ISR  
;  
_reset      mov.b   #0x00, &P1OUT  
jmp   _main  
  
;  
; Interrupt Vectors  
;  
.sect  ..reset" ; MSP430 RESET Vector  
.short _reset
```

05 Unterprogramme, Interrupts und Timer

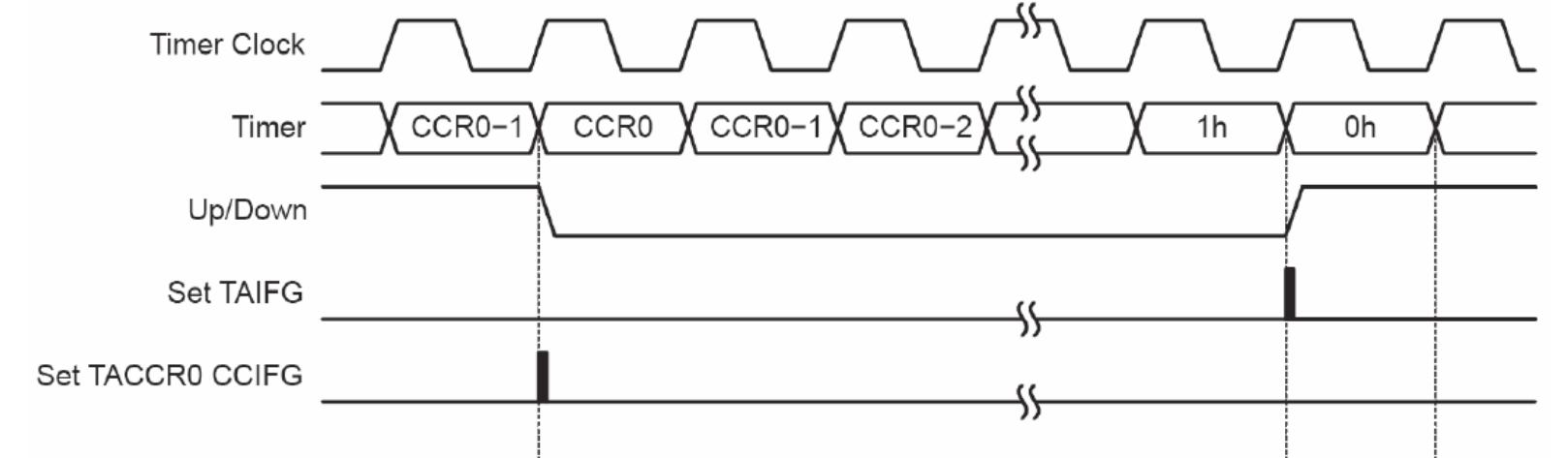
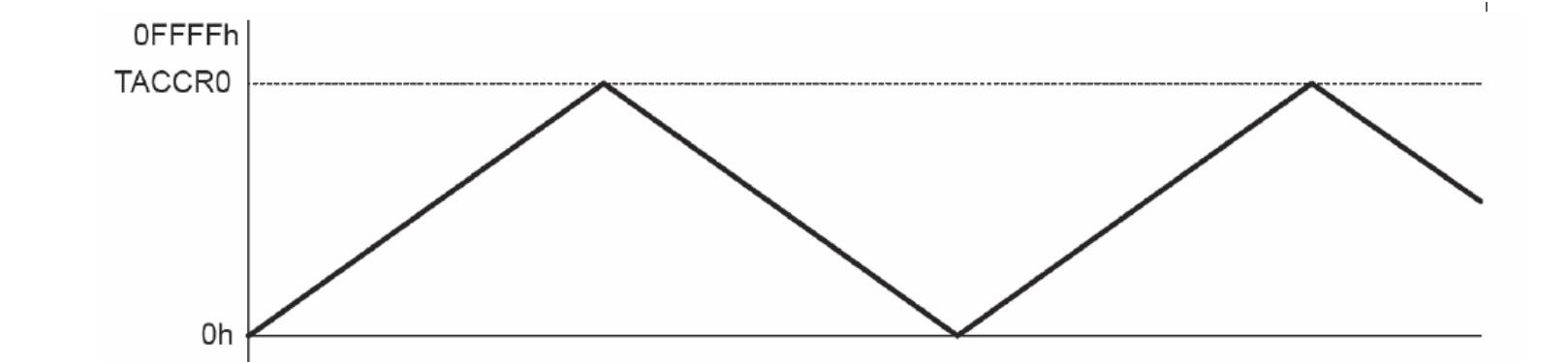
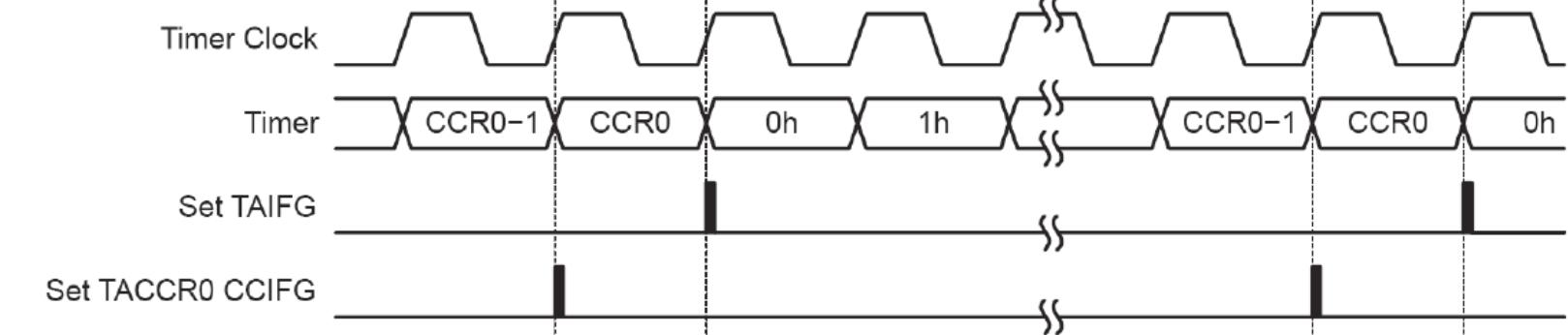
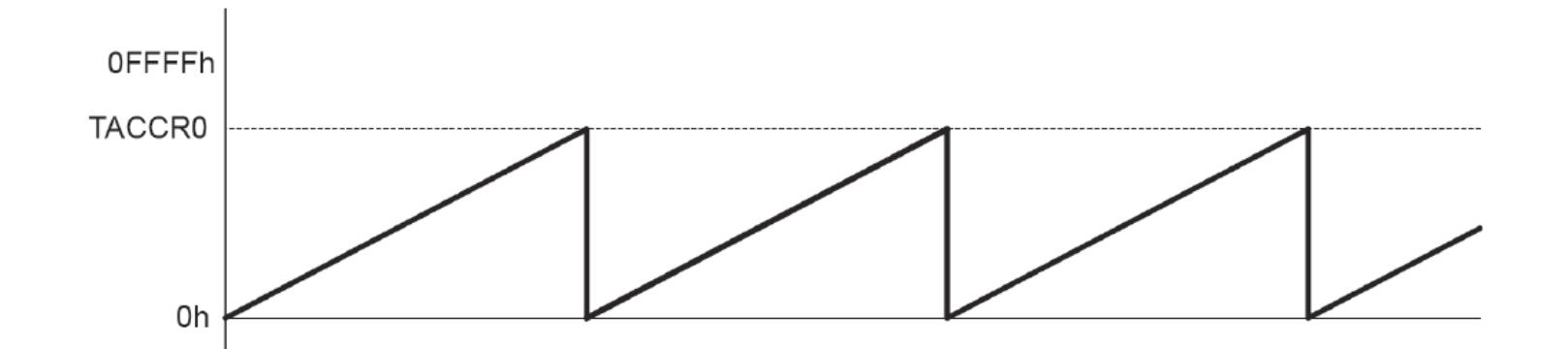
Timer des MSP430

- Der MSP verfügt über fünf Timer.
- Die Komponenten können mit unterschiedlichen auf dem Chip verfügbaren Takten versorgt werden.
- Die Timer verfügen über 16-Bit Register, die mit beliebigen Werten initialisiert werden können und mit dem ausgewählten Takt inkrementiert werden.
- Ist der Wert Null, wird ein Interrupt ausgelöst und das Register wieder mit dem Startwert initialisiert.

05 Unterprogramme, Interrupts und Timer

Modus des Timers

- 4 Operationsarten:
 - Stop,
 - Up,
 - Continuous und
 - Up/Down.
- 3 Vergleichsregister (TACCRx).
- 2 Interrupt-Vektoren:
 - TACCR0 – Vergleichsregister,
 - TAIV – Timer Überlauf.

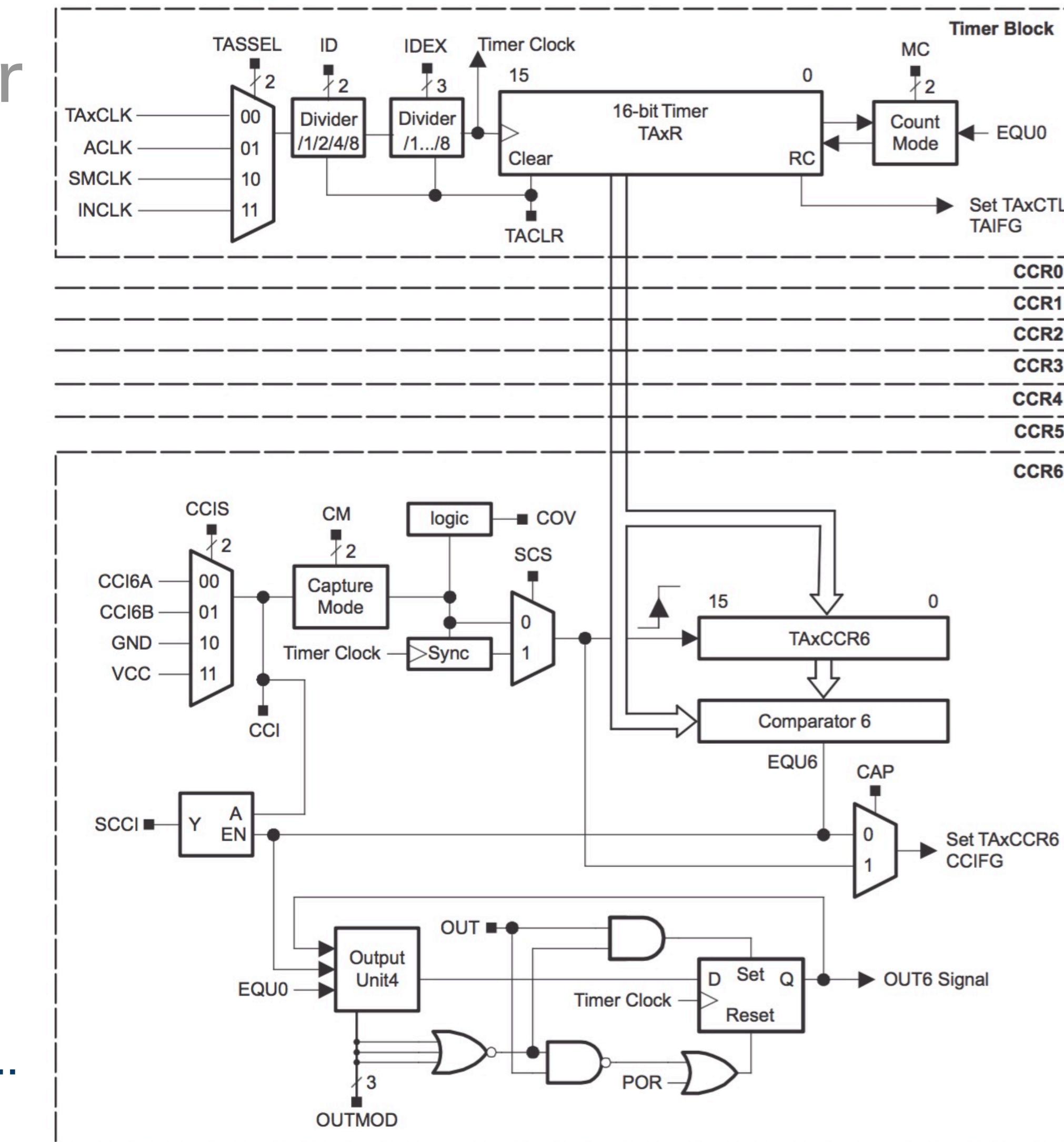


05 Unterprogramme und Timer

Aufbau des Timers

- Zählertakt einstellbar über Eingangstakt **TASSEL** und zwei Dividierer.
- Insgesamt können 7 (**CCR0** ... **CCR6**) Register mit Vergleichswerten geladen werden.
- Wird einer der Werte erreicht wird ein Interrupt ausgelöst.

User's Guide — MSP430FR58xx, MSP430FR59xx, ...
 SLAU367H, S.503 – October 2012



05 Unterprogramme, Interrupts und Timer

Timer - Kontrollregister TACTL

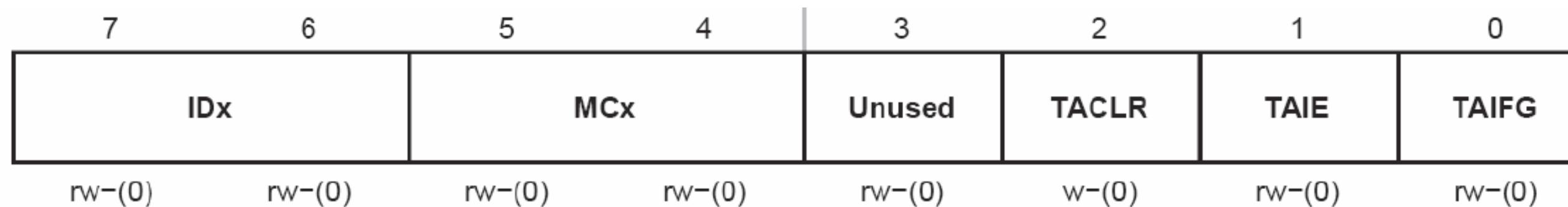


Unused Bits Unused
15-10

TASSELx Bits Timer_A clock source select
9-8 00 TACLK
 01 ACLK
 10 SMCLK
 11 INCLK

05 Unterprogramme, Interrupts und Timer

Timer - Kontrollregister TACTL



IDx	Bits 7-6	Input divider. These bits select the divider for the input clock. 00 /1 01 /2 10 /4 11 /8
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00 Stop mode: the timer is halted 01 Up mode: the timer counts up to TACCR0 10 Continuous mode: the timer counts up to 0FFFFh 11 Up/down mode: the timer counts up to TACCR0 then down to 0000h
Unused	Bit 3	Unused
TACLR	Bit 2	Timer_A clear. Setting this bit resets TAR, the TACLK divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.
TAIE	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0 Interrupt disabled 1 Interrupt enabled
TAIFG	Bit 0	Timer_A interrupt flag 0 No interrupt pending 1 Interrupt pending

05 Unterprogramme, Interrupts und Timer

Beispiel (MSP430G2452)

```

;-----;
; Timer Initialisierung
;-----;

    mov.w  #CCIE,&CCTL0           ; CCR0 interrupt enabled
    mov.w  #0xFFFF,&TACCR0        ; Timer_A Capture Compare
    mov.w  #(TASSEL_2 | MC_3 | ID_2 | TAIE ),&TACTL
                                ; TASSEL:   Timer_A Source SElect
                                ; TASSEL_2: SMCLK (1MHz)
                                ; MC_0:     Halt
                                ; MC_1:     up_mode
                                ; MC_2:     continuous_mode
                                ; MC_3:     up-down_mode
                                ; ID_2:     Verteiler (Input Divider) = 1:4
    bis.w  #GIE,SR

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

    .sect  ".int09"             ; Timer_A0 Vector
    .short TA0 ISR              ; ISR

```

05 Unterprogramme, Interrupts und Timer

Aufgabe – Timerbaustein

Blinkende LED mit Timer (`blink_timer.asm`):

Das Programm `blink_timer.asm` ist so zu erweitern, dass der Wechsel der LED über den durch den Timer ausgelösten Interrupt erfolgt und die Hauptschleife **Mainloop** leer ist:

```
Mainloop    nop
            jmp Mainloop
```



Der Timer soll dabei zur Untersuchung der Funktion für unterschiedlichen Ausgangsfrequenzen initialisiert werden.

05 Unterprogramme, Interrupts und Timer

Aufgabe – Multiplikation

Programm zur Multiplikation (**mult.asm**):

1. Es ist ein Programm zu entwerfen, welches zwei 16-Bit Werte aus dem Speicher miteinander Multipliziert und das Ergebnis wieder im Speicher ablegt. Ist das Ergebnis zu groß soll das durch die **rote LED** signalisiert werden.

2. Die Multiplikation ist in eine Funktion einzubetten, welche 32 Bit Ergebnisse berechnen kann. Die Operanden selbst sollen eine Breite von 16 Bit haben.



05 Unterprogramme, Interrupts und Timer

Aufgabe - Fakultät

Programm zur Berechnung der Fakultät (`fakultaet.asm`):

Es ist ein Programm zu entwerfen, welches die Fakultät einer Zahl n berechnet. Dazu kann das Unterprogramm zur Multiplikation verwendet werden. Alle Berechnungen mit 16 Bit.

Beispiel: $n=5$

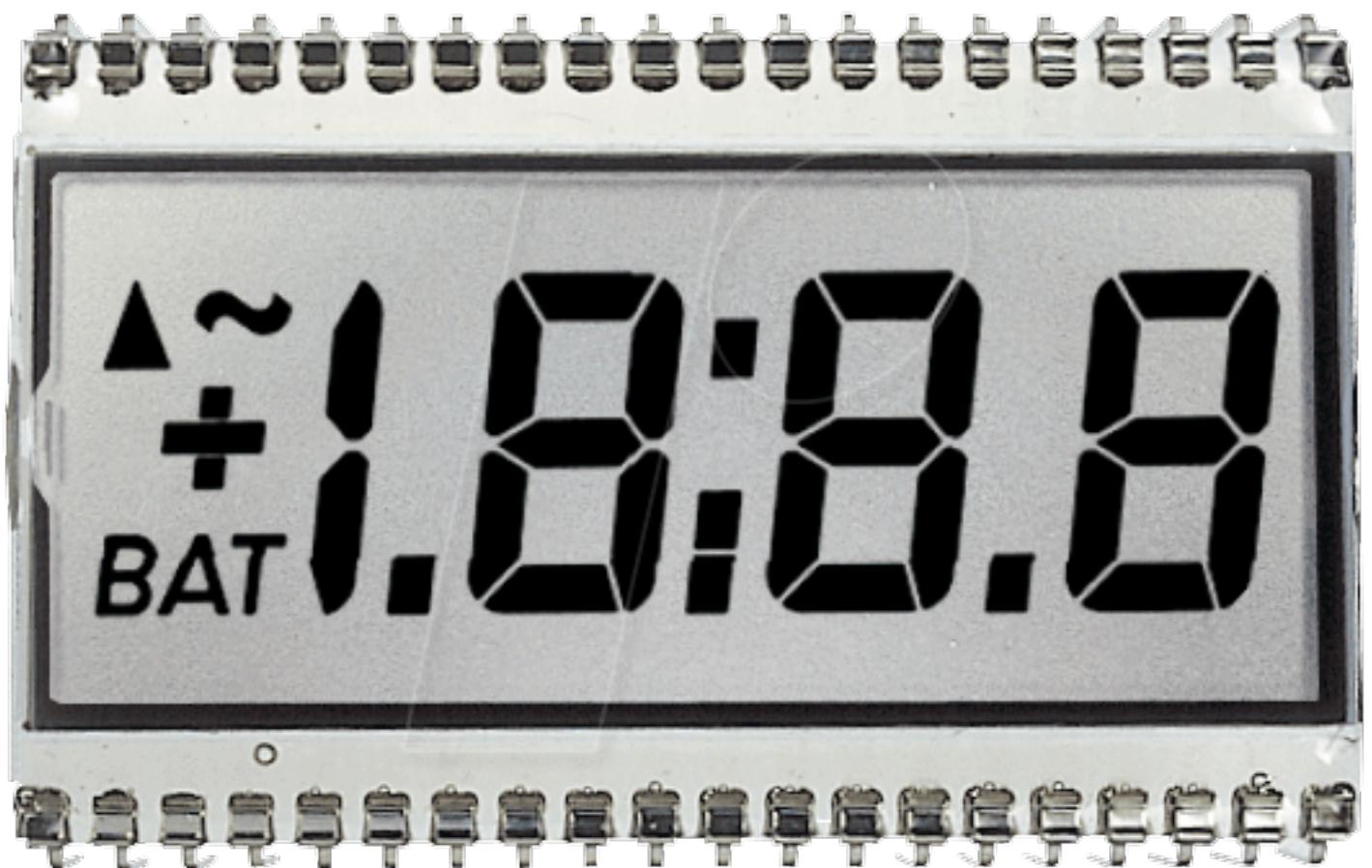
Ergebnis: $1 \times 2 \times 3 \times 4 \times 5 = 120$

- a) Was ist der größte Wert, von dem die Fakultät berechnet werden kann?
- b) Wie viele Befehle müssen für die Berechnung dieses Wertes abgearbeitet werden?



Kapitel 06

LCD Controller

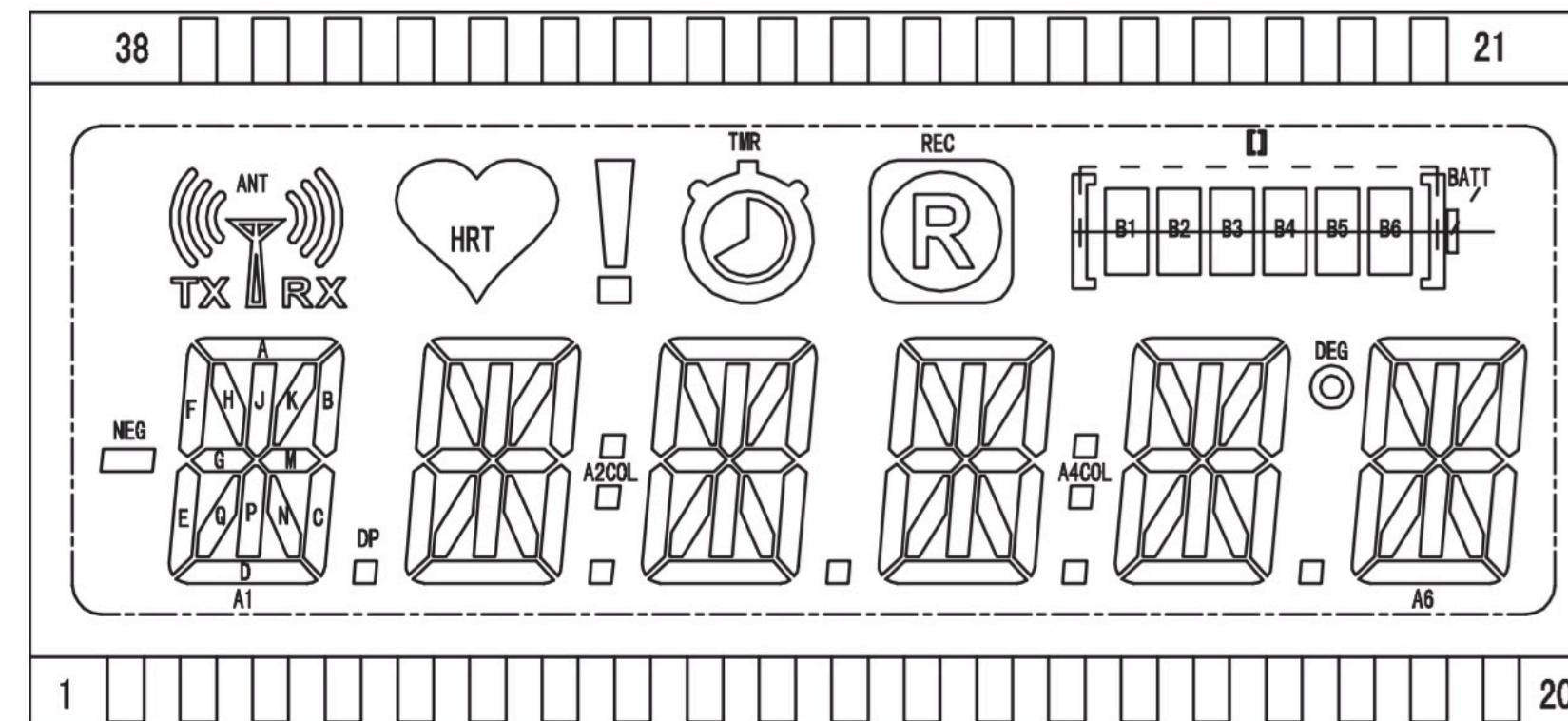


06 LCD Controller

The MSP430FR6989 LaunchPad features an on-board LCD!

Programm mit einfacher Textausgabe (`1cd.asm`):

- a) Anzeigen eines Textes.
- b) Das REC Symbol in einem Takt von 1 Hz blinken lassen.



Literatur

User's Guide — MSP430FR6989 LaunchPadTM Development Kit

SLAU627A – May 2015

<http://www.ti.com/lit/ug/slau627a/slau627a.pdf>

User's Guide — MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family

SLAU367H – October 2012

<http://www.ti.com/lit/ug/slau367h/slau367h.pdf>

Texas Instruments Wiki

http://processors.wiki.ti.com/index.php/Main_Page

Assembler Workshop

<http://wwwpub.zih.tu-dresden.de/~knodel/ASM/>

