Einführung in Assembler und Programmablaufpläne (PAP)

1. ARM Assemblerprogramme[1]

Befehl	Syntax	Funktion
ADD	ADD R_D, R_N, Operand²	Addieren von 2 oder mehr Registern. Ergebnis wird in R_D gespeichert.
BIC	BIC R_D, R_N, Operand²	AND-Operation auf R_N mit Komplementärwert von Operand² Ergebnis wird in R_D gespeichert.
CMP	CMP R_N, Operand²	Vergleich auf Gleichheit. Bei Gleichheit wird Zero-Flag (Z=1) gesetzt.
EOR	EOR R_D, R_N, Operand²	Bitweises Exklusiv-Oder von R_N und Operand²
LDR	LDR R_D, [R_N]	1. Lade Register R_D mit Inhalt von Adresse aus R_N
	LDR R_D, =Offset	2. Lade R_D mit Wert von Offset (Speicheradresse)
MOV	MOV R_D, Operand²	Move (Wert in Register schreiben) Wert von Operand² ins Register R_D
MOV32	MOV32 R_D, Operand²	Move (Wert in Register schreiben) Wert von Operand² ins Register R_D (freie Wertewahl)
ORR	ORR R_D, R_N, Operand²	Bitweises Oder von R_N und Operand² Ergebnis wird in R_D gespeichert.
SBC	SBC R_D, R_N, Operand²	Subtraktion von 2 oder mehr Registern Ergebnis wird in R_D gespeichert.
STR	STR R_N, [R_D]	Store Wert von R_N als Inhalt von Adresse aus Rd.
SUBS	SUBS R_D, R_N, Operand²	Subtraktion von 2 oder mehr Registern. Ergebnis wird in R_D gespeichert. Zusätzlich vergleicht der Befehl das Register R_D auf den Wert 0. Bei Wert 0 wird im APSR-Register das Zero-Flag auf 1 gesetzt.

[R_D] = Zielregister

[R_N] = Quellregister (mit Operand[1])

Operand² = Weiteres Quellregister ODER Konstante (z.B.: #3, #0x34, …)

[1] Auszug des ARM-Befehlssatzes [https://www.heyrick.co.uk/assembler/qfinder.html]

2. Direktiven (Auszug)[2]

Befehl	Syntax	Funktion
ALIGN	ALIGN	ALIGN dient der Angleichung von Speicheradressen.
AREA	AREA SEGMENT, Attribut1, …	Bereichsdefinition (Segment, Code/Daten, Write/Read-Only)
END	END	Definiert Ende der Datei
ENTRY	ENTRY	Definiert Einstiegspunkt des Programms
EQU	Bezeichner EQU WERT	Definiert Bezeichner mit Wert. An jeder Stelle im Code wo der Bezeichner verwendet wird, wird der Bezeichner durch den Wert ersetzt.
EXPORT	EXPORT Symbol	Symbol wird exportiert. Linker weiß wo sich Funktion befindet, so dass andere Dateien auf die Funktionen in dieser Datei zugreifen können.
IMPORT	IMPORT Symbol	Importiert Symbol. Nun kann die Funktion Symbol genutzt werden.
THUMB	THUMB	THUMB-Mode Anweisung, ARM Cortex M4 kann nur THUMB-Instruktionen ausführen

3. Kommentare in Assembler

Kommentare beginnen mit einem Semikolon. Bis zum Ende der Zeile wird alles als Kommentar vom Assembler betrachtet und ignoriert. Mehrzeilige Kommentare müssen in jeder Zeile mit einem Semikolon beginnen.

Beispiele:

```
MOV R4, #0 ; Dies ist ein einfacher Kommentar
; Ein weiterer Kommentar
MOV R5, #3 ; Noch ein Kommentar
```

[2] Auszug vom ARM Info Center [http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/Chedcbai.html]

4. Symbole

Symbole werden vom Linker und vom Debugger benötigt. Der Linker liest Symbole aus Eingangsdateien, um zu wissen, wo welche Funktion definiert wird. Der Debugger benötigt Symbole, um zu wissen, wo er sich im Programm befindet.

Ein „Label" ist ein Symbol. Ein Label ist der Name der Funktion (Sprungadresse).

Bezeichner, die mit dem Befehl EQU festgelegt werden sind absolute Symbole.

Der Linker unterscheidet zwischen ***relocatable Symbols (Labels)*** und ***absolute Symbols (Bezeichner)***.

5. Sprungbefehle[3]

Befehl	**Syntax**	**Funktion**
B	B Label	Springe (Branch) zu Label.
BEQ	BEQ Label	Springe (Branch) zu Label, wenn EQ (Equal), das bedeutet, wenn ein Vergleich durch den CMP-Befehl positiv war. Zero-Flag im APSR-Register = 1
BL	BL Label	Springe (Branch) zu Label. Schreibt Adresse der nächsten Instruktion ins LR (Link-Register).
BX	BX R_M	Springe (Branch) zu Adresse in Register. Aber indirekt. Beispiel: BX LR
BNE	BNE Label	Springe (Branch) zu Label wenn Zero-Flag im APSR-Register = 0 (Vergleich war negativ)

[R_M] = Register mit Sprungadresse

Link Register:

Register r14 is used as the subroutine Link Register (LR). Register r14 receives the return address when a Branch with Link (BL or BLX) instruction is executed.

[3] Auszug aus ARM Info Center
[http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0211h/ch02s08s01.html]

6. Quellcode

```
1.    ; Defines aus lm4f120h5qr.h:
2.    GPIO_PORTA_DATA_R   EQU 0x400043FC ; GPIO PORT A - Data Register
3.    GPIO_PORTA_DEN_R    EQU 0x4000451C ; GPIO PORT A  - Digital-Enable Register
4.    GPIO_PORTA_DIR_R    EQU 0x40004400 ; GPIO PORT A -  DIRECTION-Register
5.    SYSCTL_RCGC2_R      EQU 0x400FE108 ; System Control Register: Clock-Aktivierung
6.    SYSCTL_RCGC2_GPIOA  EQU 0x00000001 ; Clock-Bitmuster für  Port A
7.
8.    ; eigene Defines
9.    COUNTER EQU 0xFFFFF ; Blinky-Delay
10.   PA7     EQU 0x80    ; Bitmuster für Port PA7 (grüne LED auf Boosterpack)
11.   ; Assembler-Direktive -> .text-Bereich
12.    THUMB
13.    AREA    |.text|, CODE, READONLY, ALIGN=2
14.    EXPORT  Blinky
15.    ENTRY
16.
17.   ; Funktionen
18.   Blinky
19.    ; Als erstes Clock auf Port A aktivieren
20.    LDR R1, =SYSCTL_RCGC2_R
21.    LDR R0, [R1]
22.    ORR R0, R0, #SYSCTL_RCGC2_GPIOA
23.    STR R0, [R1]
24.
25.    ; Setze Port PA7 (0x80) als Ausgang
26.    LDR R1, =GPIO_PORTA_DIR_R
27.           LDR R0, [R1]
28.           ORR R0, R0, #PA7
29.           STR R0, [R1]
30.
31.         ; Setze Port PA7 als Digital-Port
32.           LDR R1, =GPIO_PORTA_DEN_R
33.           LDR R0, [R1]
34.           ORR R0, R0, #PA7
35.           STR R0, [R1]
36.    MOV32 R8, #COUNTER
37.    MOV R4, #0
38.    BL Toggle
39.
40.
41.   Toggle
42.       ; Toggle die grüne LED
43.       LDR R1, =GPIO_PORTA_DATA_R
44.       LDR R0, [R1]
45.       EOR R0, #PA7
46.       STR R0, [R1]
47.
48.       BL delay
49.       B Toggle
50.
51.   delay MOV32 R8, #COUNTER
52.
53.   dl1
54.       SUBS R8, #1
55.       BNE dl1
56.       BX LR
57.
58.       ALIGN
59.       END
```

7. Programmablaufplan

Abbildung 1: Kontrollpunkt

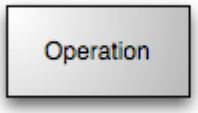


Abbildung 2: Tätigkeit

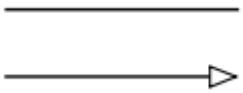


Abbildung 3: Verbindungselement

Abbildung 4: Subtätigkeit

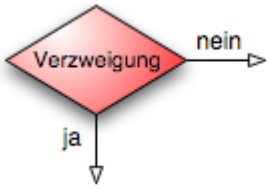


Abbildung 5: Verzweigung

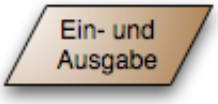


Abbildung 6: Ein- oder Ausgabe

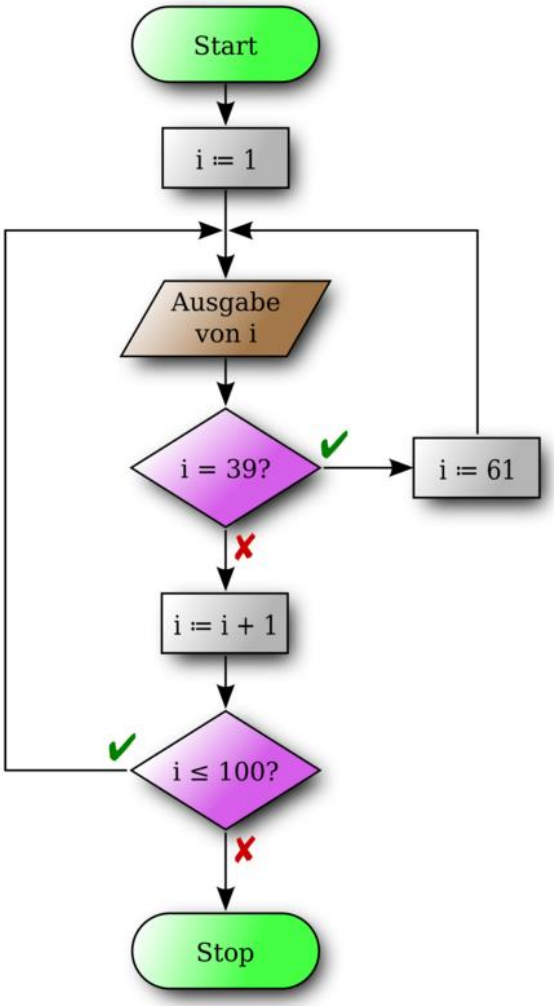


Abbildung 8: PAP Beispiel1

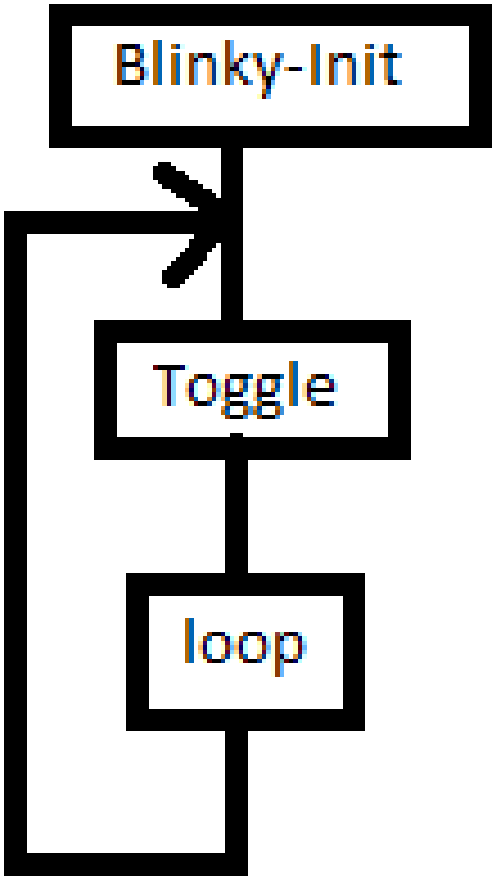


Abbildung 7: PAP Bespiel 'Blinky'

Ostfalia – HAW | Fakultät Informatik | Bachelor – Rechnerstrukturen Labor

8. Programmablauf / Sprungbefehle

Programmablauf:

C

```
1  int main() {
2    doNothing();
3  }
   void funktion2() {
   }
```

Ein C-Programm startet seinen Programmablauf in der Funktion main(). In diesem Beispiel erfolgt kein Aufruf von funktion2(); Somit wird funktion2() niemals aufgerufen.

Assembler

```
1  Blinky
2    LDR R0, #3
3
4  Funktion1
5    LDR R1, #4
6
7  Funktion2
8    LDR R2, #2
```

Ein Assembler-Programm läuft Zeile für Zeile durch, falls kein Sprungbefehl erfolgt.

Branch-Befehl ***B***

C

```
1  int main() {
2    funktion2();
6  }
3  void funktion2() {
4    doNothing();
5  }
```

In der Hauptroutine main() wird funktion2 aufgerufen. Nach Abarbeitung von funktion2 springt das Programm an die Stelle nach Aufruf von funktion2().

Assembler

```
1  Blinky
2    LDR R0, #3
3    B Funktion2
     Funktion1
     LDR R1, #4
4  Funktion2
5    LDR R2, #2
```

In der Funktion Blinky wird Funktion2 aufgerufen.

Das Programm springt zu Funktion2. Nach Abarbeitung beendet sich das Programm.

Branch-Befehl ***BL/BX***

C

1	int main() {
2	funktion2();
6	}
3	void funktion2() {
4	doNothing();
5	}

In der Hauptroutine main() wird funktion2 aufgerufen.

springt das Programm an die Stelle

nach Aufruf von funktion2().

Assembler

1	Blinky
2	LDR R0, #3
3	BL Funktion2
7	B Ende
4	Funktion2
5	LDR R1, #4
6	BX LR
8	Ende

Diesmal wird in Schritt 3 die Funktion2 mit

dem Befehl BL aufgerufen. Dieser speichert die Adresse des nächsten Befehls im LR-Register.
In Schritt 6 wird der BX-Sprungbefehl benutzt.
Dieser springt an die Adresse, die im Link-Register (LR) gespeichert ist. Somit erfolgt der Ablauf des

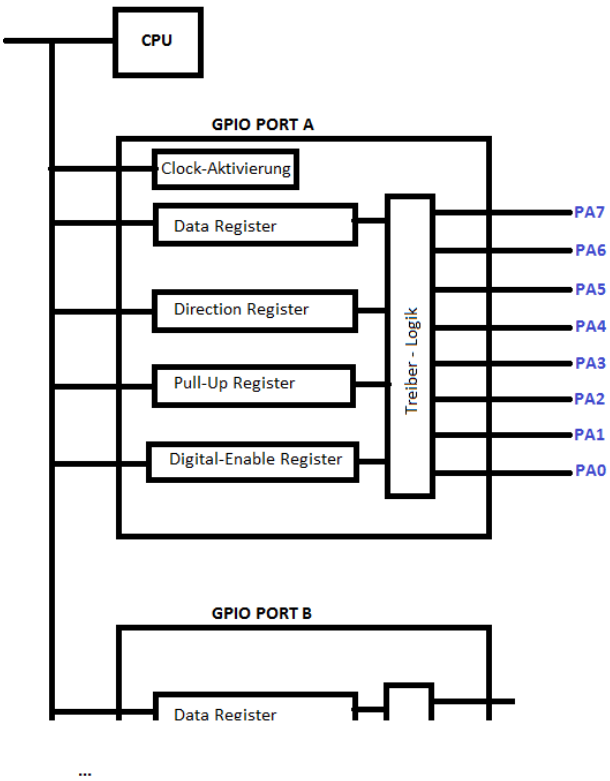
Programms wie im C-Beispiel.

9. Funktionsweise der Experimentierplatine

9.1. LED

Anschluss der LED an Prozessor-Pins **PA3, PA6, PA7** über GPIO-Port A:

Die RGB-LED ist über 3 GPIO (General Purpose Input /Output) -Leitungen mit der CPU verbunden. Das Experimentierboard besitzt mehrere GPIO-Ports (A-F). Jeder dieser Ports besitzt 4-8 Ausgangspins. Diese Pins können unterschiedlich angesprochen werden.



Entweder

als Ein-/Ausgabe-Pin

oder

als Analog-/Digital-Pin

Die Funktion des Ports ergibt sich aus den Werten **mehrerer** Register:

Data, Digital-Enable, Direction und Pull – Up/-Down.

Der Treiber-Logik-Baustein schaltet je nach Einstellung dieser Register. Um einen Port zu aktivieren, muss das passende Bitmuster SYSCTL_RCGC2_GPIO**X** im Clock-Aktivierungs-Register SYSCTL_RCGC2_R gesetzt werden.

Data/Direction/Pull-Up/Digital-Enable sind 32-Bit Register. Allerdings werden nur die untersten 8 Bits benutzt:

Bit	7	6	5	4	3	2	1	0
Wert								

Um nun die grüne LED-Leuchte als digitalen Ausgangspin anzusprechen, müssen Bitmuster in Registern gesetzt werden. Da die LED-Leuchte ein digitaler Ausgang ist, muss im Direction-Register das Bitmuster für die LED gesetzt werden, ebenso im Digital-Enable-Register. Um nun die LED einzuschalten muss man das Bitmuster **0x80** für den Port **PA7** im Data-Register setzen. 0x80 ist binär 10000000:

Bit	7	6	5	4	3	2	1	0
Wert	1	0	0	0	0	0	0	0

Bit 7 ist gesetzt -> Also wird PA7 auf HIGH gesetzt und die LED leuchtet Grün.

9.2. Stack

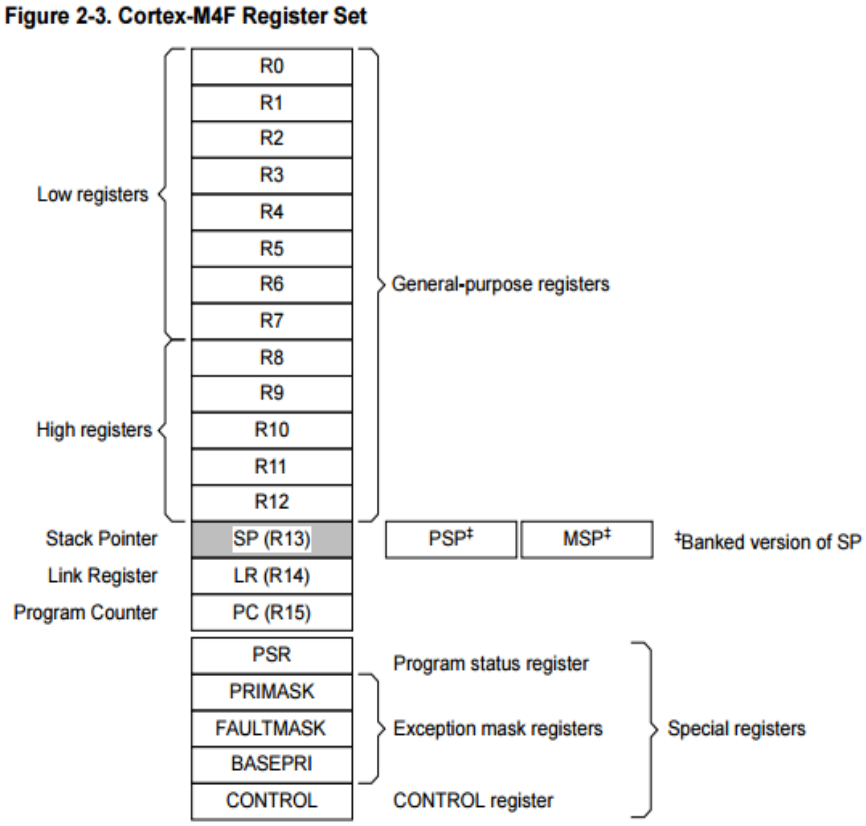


Abbildung 9: Registerset (Auch im Debugmodus einsehbar)

Bitmuster ergibt sich aus den addierten Bitmustern der benötigten Ports SYSCTL_RCGC2_GPIOX. Beispiel:

```
21: ; Als erstes Clock auf Port A aktivieren
22: LDR R1, =SYSCTL_RCGC2_R
23: LDR R0, [R1]
24: ORR R0, R0, #SYSCTL_RCGC2_GPIOA | SYSCTL_RCGC2_GPIOB
25: STR R0, [R1]
```

1. Direction des Ports setzen (Input/Output) -> passendes Bitmuster in GPIO_PORTX_DIR_R setzen.
Das Bitmuster ergibt sich aus den gewünschten Pins (z.B 0x80 für Pin7 als Ausgang). Beispiel:

```
27: ; Setze Port PA7 (0x80) als Ausgang
28: LDR R1, =GPIO_PORTA_DIR_R
29: LDR R0, [R1]
30: ORR R0, R0, #PA7
31: STR R0, [R1]
```

2. Digital-Enable bei Digitalports setzen -> passendes Bitmuster im GPIO_PORTX_DEN_R - Register setzen.
Wird ein Pin als Digitalpin genutzt, muss das Bitmuster des Pins /der Pins in diesem Register gesetzt werden. Beispiel:

```
33: ; Setze Port PA7 als Digital-Port
34: LDR R1, =GPIO_PORTA_DEN_R
35: LDR R0, [R1]
36: ORR R0, R0, #PA7
37: STR R0, [R1]
```

4. evtl. Pull-Up-Widerstand setzen (Button) -> passendes Bitmuster in GPIO_PORTX_PUR_R - Register setzen.
Beispiel:

```
; Setze Pull-Up-Widerstand für Port X Pin Y (Bitmuster #XYZ)
LDR R1, =GPIO_PORTX_PUR_R
LDR R0, [R1]
ORR R0, R0, #XYZ
STR R0, [R1]
```