



Berner Fachhochschule  
Haute école spécialisée bernoise  
Bern University of Applied Sciences

# Hardwarenahe Softwareentwicklung

## Assembler-Programmierung

V5.1, ©2023 roger.weber@bfh.ch

# Lernziele

Sie sind in der Lage:

- ▶ Vor- und Nachteile der Assembler-Programmierung zu erklären.
- ▶ Den Entwicklungsablauf eines Assembler-Projektes zu verstehen.



# Inhaltsverzeichnis

1. Assembler-Eigenschaften
2. Aufbau einer Assembler-Datei
3. Assembler-Syntax
4. Entwicklungsablauf

# Assembler-Eigenschaften



# Eigenschaften

- ▶ Die wichtigsten Eigenschaften der Assembler-Programmierung sind:
  - ▶ Hardwarenahe Programmiersprache.
  - ▶ Syntax ist abhängig von der CPU.



- ▶ Was ergeben sich daraus für Vor- und Nachteile?

# Sprachebenen

 Maschinennähe	Nimm eine Flasche Bier aus dem Kühlschrank	Natürliche Sprache	 Abstraktion
	Bier - 1	Pseudocode	
	Bier = Bier - 1;	Hochsprache	
	LDR r0,=bier LDR r1,[r0] SUB r1,r1,#1 STR r1,[r0]	Assembler	
	0x4802 0100'1000'0000'0010 0x6801 0110'1000'0000'0001 0x3901 0011'1001'0000'0001 0x6001 0110'0000'0000'0001	Maschinencode	

## Aufbau einer Assembler-Datei

# Aufbau einer Assembler-Datei

Eine Assembler-Datei wird beeinflusst durch:

- ▶ Assembler-Syntax  
→ abhängig von der gewählten CPU
- ▶ Assembler-Direktiven  
→ abhängig von der Entwicklungsumgebung (z.Bsp. GNU)
- ▶ Programmierrichtlinien  
→ abhängig von der Firma



## Aufbau einer Assembler-Datei

```

*****
*
* Project           : Project name
*
*
*
*
* Program/Module    : Module name
* File name         : File.s
* Version           : 1.00
* Created           : dd.mm.yyyy
* Author            : My name
*
* -----
* Description        : A short description of the module
*
*
*
* Modifications / History:
* Author    date      description
* m-n      dd.mm.yyyy  ...
*****

```

Header  
(optional)

```

/*****
* PUBLIC DECLARATIONS
*****/

```

Exported Symbols  
(optional)

```

*****
* EXTERNAL DECLARATIONS *
*****
.extern      extsub      @imported symbol

```

Imported Symbols  
(optional)

```

/*****
* INCLUDE DEFINITIONS
*****/

#include "registers.h" @include controller regs

```

Included Files  
(optional)

```

/*****
* EQUATE DEFINITIONS
*****/
.equ      MAX,0x1000      @maximum allowable size

```

EQU  
(optional)

```

/*****
* VARIABLE AND CONSTANT DEFINITIONS
*****/

```

### Variables & Constants (optional)

```
.data
myvar:space      4               @space for 1 Word (32-Bit)

/*****
```

```

* CODE
/*****
      .text

```

```
mysub: LDR    r0,=myvar    @ load address of variable myvar
      MOV    r1,#MAX      @ load value MAX to r1
      STR    r1,[r0]      @ store value in variable myvar
      BL     extsub       @ call subroutine extsub
      MOV    pc,lr        @ return from subroutine
      end
```

Code

```

/*****
* CODE
*****/

.text

mysub: LDR    r0,=myvar    @ load address of variable myvar
      MOV    r1,#MAX      @ load value MAX to r1
      STR    r1,[r0]      @ store value in variable myvar
      BL     extsub       @ call subroutine extsub
      MOV    pc,lr        @ return from subroutine
      .end

```

# Aufbau einer Assembler-Zeile

Labelfield	Operation / Instruction	Operands	Comment
main:	MOV	r1, #MAX	@ load maximal value

## Labelfeld:

- ▶ Erste Spalte
- ▶ Endet mit ":"
- ▶ Symbolische Adresse für Subroutinen oder Label.

## Operation / Instruktion:

- ▶ CPU-Instruktion oder Assembler-Direktive.

# Aufbau einer Assembler-Zeile

Labelfield	Operation / Instruction	Operands	Comment
main:	MOV	r1,#MAX	@ load maximal value

## Operanden:

- ▶ Operanden von Instruktionen oder Assembler-Direktiven.
- ▶ ARM-CPU: 1, 2 oder 3 Operanden

B	loop	@ Beispiel 1 Operand
MOV	pc,lr	@ Beispiel 2 Operanden
SUB	r0,r1,r2	@ Beispiel 3 Operanden

## Kommentar:

```
/* Blockkommentar ueber  
   mehrere Zeilen */  
# Zeilenkommentar  
LDR r0,=op1 @ Zeilenendkommentar
```

# Assembler-Syntax

# Symbole

Symbole sind Namen für Variablen, Subroutinen, Labeln usw.

Folgende Zeichen sind zugelassen:

- ▶ Buchstaben a..z, A..Z (keine Umlaute)
- ▶ Ziffern 0..9
- ▶ Underscore “\_”
- ▶ Punkt “.”

Folgende Regeln gelten:

- ▶ Das erste Zeichen muss ein Buchstabe sein.
- ▶ Symbole können beliebig lang sein, alle Zeichen sind signifikant.
- ▶ Es wird zwischen Gross- und Kleinschreibung unterschieden.

# Konstanten

Typ	Kurzbezeichnung	Präfix	Beispiel
Zeichenkette	String	"	"Hallo"
Buchstabe	Char	'	'a'
Numerisch	Hex	0x	0x7FF
Numerisch	Bin	0b	0b0111011
Numerisch	Oct	0	0123
Numerisch	Dez	kein Präfix	123

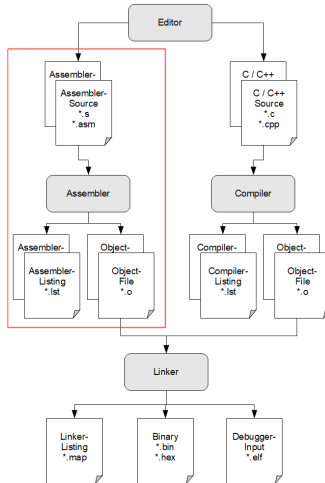
Beispiele:

```
string: .asciz "hallo"    @ label string
LDR     r0,=4              @ r0 = 4
LDR     r1,=0x12345678     @ r1 = 0x12345678
```

# Entwicklungsablauf

# Ablauf einer Assembler-Entwicklung

Viele verwendete Tools sind dieselben wie für die C-Entwicklung. Neu ist der Assembler-Pfad.





- ▶ Übersetzt den Assembler-Sourcecode (ASCII-Quelldatei mit der Endung “.s“ oder “.asm“) in ein Objekt-File (“.o“).
- ▶ CPU-spezifisch
- ▶ Erzeugt optional ein Assembler-Listing (“.lis“).
- ▶ Cross-Assembler (Target und Entwicklungsplattform sind unterschiedlich).

# Beispiel Assembler-Listing

```
.global main
.thumb
.text

main:      MOV    r0,#1           @ r0 = 1
80001dc:    2001           mov r0,#1
loop:      LDR    r1,=0x12345678  @ r1 = 0x12345678
80001de:    4901           ldr r1,[pc,#4]
          ADD    r0,r0,r1        @ Beide Werte addieren, r0 = r0 + r1
80001e0:    1840           add r0,r0,r1
          B      loop           @ Endlosschleife, branch to loop
80001e2:    e7fc           b.n 80001de <loop>
80001e4:    12345678        .word 0x12345678
```