Hochschule Bremerhaven

Fachbereich II – Management und Informationssysteme

Informatik

Modul Rechnerarchitektur

Protokoll zu Aufgabenblatt 07

von

Kasem Rashrash Matrikel-Nr. 41398
Mahdie Sabbagh Matrikel-Nr. 41176

19.06.24

Inhaltsverzeichnis

1 Aufgabe01

In dieser Aufgabe mussten zwei verschiedene Assembler-Programme geschrieben werden, die die Taktanzahlen entsprechen, die wir berechnet mit der gegebenen Rechnung.

T = 37
S = 1

Erstmal rechnen wir die gegebene Rechnung aus:
a) mit P1 := (282823 mod (4 * 37 + 5 * 1)) + 128 = 207 Takten
b) mit P2 := (728441 mod (6 * 37 + 7 * 1)) + 8192 = 8413 Takten

1.1 Für a = 207 Takten

a)

block_a_start

ldi R16, 0x45 ; 1 Takt -Initiliasierung 0x45 = 69

LoopLabel:
dec R16 ; 1 Takt -dekrementiert den Wert in R16 um 1 –i
brne LoopLabel ; nicht Null = 2 Takte, Null = 1 Takt

block_a_ende

Taktenanzahl:
Initiliaisierung (ldi) + jeden Durchlauf der Schleife (dec + brne) * Gesamt durchlauf der Schleife (69 – 1) + letzter Durchlauf der Schleife
1 + (1 + 2) * (69 – 1) + 2 = 1 + (3*68) + 2 = 204 + 3 = 207

Fazit: Das Programm benötigt insgesamt 207 Taktzyklen.

1.2 Für b = 8413 Takten

b)

block_b_start:

ldi R16 , 0x9 ; 1 Takt //R16 = 9
ldi R17 , 0x0 ; 1 Takt //R17 = 0

LoopLabel:
.EQU, cl = 262 ; 0 Takt // c1 = 262
ldi R18 , 0 ; 1 Takt // R18 = 0
ldi R19, c1 ; 1 Takt // R19 = 263

LoopLabel_1:
inc R18 ; 1 Takt // R18 = 0 + 1
cp R18, R19 ; 1 Takt // R18 == R19
brne LoopLabel_1 ; solange die werte nicht gleich sind -> 2 Takten; falls gleich => 1 Takt

dec R16 ; 1 Takt
cp R16,R17 ; 1 Takt
brne LoopLabel ; solange R16 ungleich R17 -> 2 Takten ; falls gleich => 1 Takt

ldi R20, 5 ; 1 Takt

NOP_Loop:
NOP ; 1 Takt
dec R20 ; 1 Takt
brne NOP_Loop ; 2 Takte, letzter Durchlauf 1 Takt

END_NOP-Loop:

NOP ; 5 Takte
NOP
NOP
NOP
NOP

block_b_ende

Taktanzahl:
Initialisierung + 9 + Initialisierung + LoopLabel (.EQU + Initialisierung + Initialisierung) + LoopLabel_1 (Increment + Compare Immediate + Branch if not equal) + NOP-Schleife

1 + 1 + (1 + 1 + 0 + (1 +1 + 2) * (262 – 1) + 2) + (1 + 1 + 2) * (9 – 1) + 2 + 25
= 2 + (1044 + 1 + 1 + 2) * (9 – 1)) + 2 + ((4 * 4) + 3) + 1 + 5
= 2 + (1048 * 8) + 27
= 8388 + 25 = 8413 Takte

2 Aufgabe02

Für die Aufgabe 02 standen uns der Labor-Rechner, das Oszilloskop und unser Code aus Aufgabe 01 zur Verfügung. Zunächst haben wir die README-Datei gelesen. Anschließend haben wir unseren Code aus Aufgabe 01 übernommen und den Code-Abschnitt für S=1 hinzugefügt. Wir haben diesen in den Mustercode eingebettet und sind dann Schritt für Schritt die Anzahl der Takte durchgegangen, um sie so anzupassen, dass am Ende dieselbe Anzahl von Takten erreicht wird.

1) Mit dem Befehl make überprüften wir, ob der Code fehlerfrei ist und ausgeführt werden kann.
2) Mit make main-sim wurde dann PORT D mitgeschnitten, sodass Pin-Änderungen auf PORT D in der Datei test-output.vcd aufgezeichnet wurden. Anschließend öffneten wir diese Datei mit dem Befehl gtkwave test-output.vcd auf dem Laborrechner und überprüft, ob die Zeiten eingehalten wurden.
3) Schließlich führten wir make main-firm aus und maßen die Zeit an Pin 4 von PORT D mit dem Oszilloskop, um dies erneut zu überprüfen. Das Signal wurde mithilfe des ATtiny-Mikrocontrollers ausgegeben.

2.1 Fall a = 207 Takten

block_a_start:

main1:

ldi R17, 0 ; 1 Takt
ldi R16, 47 ; 1 Takt
nop
nop

nop

LoopLabel:

dec R16 ; 1 Takt
cp R16,R17 ; 1 Takt
brne LoopLabel ; 2 Takte, falls ungleich Null – bei Null = 1 Takt

sbi PORTD, PD4 ; 2 Takte
cbi PORTD, PD3 ; 2 Takte
cbi PORTD, PD2 ; 2 Takte
sbi PORTD, PD3 ; 2 Takte
sbi PORTD, PD2 ; 2 Takte
nop ; 1 Takt
cbi PORTD, PD4 ; 2 Takte
rjmp main1 ; 2 Takte

block_a_ende:

Für die ersten 64 Iterationen:
46 * (1 Takt(dec) + 1 Takt(cp) + 2 Takte(brne)) = 46 * 4 = 184 Takte

Für die letzte Iteration:
1 Iteration * (1 Takt (dec) + 1 Takt(cp) 1 Takt(brne)) = 3 + 184 = 187
2(Initialisierung) + 187 = 189 Takte
NOPs:
1 + 1 + 1 = 3 Takten

Code-Abschnitt-Port:
2+2+2+2+2+1+2+2= 15 Takten

Gesamttakte:
189 + 15 + 3 = 207

2.2 Fall b = 8413 Takten

block_b_start

```
main2:
ldi R16 , 0x8 ; 1 Takt
ldi R17 , 0x0 ; 1 Takt

LoopLabel:

.EQU cl, 255 ; 0 Takt
ldi R18 , 0 ; 1 Takt
ldi R19, cl ; 1 Takt

LoopLabel_1:

inc R18 ; 1 Takt ,R18 = 0 + 1
cp R18, R19 ; 1 Takt ,R18 == R19
brne LoopLabel_1 ; solange die werte nicht gleich sind -> 2 Takten; falls gleich => 1
Takt

dec R16 ; 1 Takt
cp R16,R17 ; 1 Takt
brne LoopLabel ; solange R16 ungleich R17 -> 2 Takten ; falls gleich => 1 Takt

ldi R30, 0 ; 1 Takt
ldi R29, 44 ; 1 Takt

LoopLabel_2:

inc R30;
cp R30, R29 ; 0 == 44
brne LoopLabel_2 ; solange R16 ungleich R17


ldi R20, 8 ; 1 Takt
NOP_Loop:
dec R20 ; 1 Takt
brne NOP_Loop ; 2 Takte, letzter Durchlauf 1 Takt

sbi PORTD, PD4 ; 2 Takte
cbi PORTD, PD3 ; 2 Takte
cbi PORTD, PD2 ; 2 Takte
sbi PORTD, PD3 ; 2 Takte
sbi PORTD, PD2 ; 2 Takte
nop ; 1 Takt
cbi PORTD, PD4 ; 2 Takte
rjmp main2 ; 2 Takte
```

block_b_ende

Initialisierung + 9 + Initialisierung + LoopLabel (.EQU + Initialisierung + Initialisierung) + LoopLabel_1 (Increment + Compare Immediate + Branch if not equal) + LoopLabel_2 + NOP-Schleife + Code-Abschnitt-Port

Neue angepasste Gesamttakte:
2 + 8174 + 174 + 48 + 15 = 8413 Takte

Fazit
Letztendlich haben unser Code angepasst und es hat am Ende geklappt.
Wir benutzten den Code-Abschnitt der dargestellt für S = 1, da unser Teamnummer ungerade ist. Erstmal haben wir die Datei README vom Mustercode gelesen. Danach betteten wir unser Programm in die betrieb.S Datei ein und liesen das Programm mit make main-sim simulieren.
Für Aufgabe 2 haben wir Unterverzeichnisse erstellt in unserem Docker. aprog/ für Fall a und bprog/ für Fall b.
Problem: Der Wert 262 konnte nicht gespeichert werden in unserem Code am Anfang, da es außerhalb des Bereichs für 8-Bit-Konstanten liegt. In AVR-Assembler können die meisten Befehle nur mit 8-Bit-Konstanten arbeiten, was bedeutet, dass die Werte im Bereich von 0 bis 255 liegen sollten.
Lösung: .EQU cl, 262 umgeändert in .EQU cl, 255 und dann das Programm an die richtigen Taktenanzahl angepasst.

3 Aufgabe03

In der folgenden Aufgabe mussten wir Funktionen erstellen und die Code-Abschnitte in unser Programm so einbetten, dass am Ende die Gesamtanzahl der Takte dieselbe ist oder angepasst wird wie in Aufgabe 1.

Vorgehen:

1) Zunächst haben wir uns die Folien angeschaut sowie die README-Datei gelesen und sind wie in Aufgabe1 vorgegangen.
2) Für jeden Codeabschnitt haben wir eine Funktion aufgebaut und diese mit den in der Vorlesung gezeigten Befehlen uns angeschaut. Die Ausführungszeiten sollen inklusi-

ve call, ret und den sbi, cbi-Befehlen wie in Aufgabe 1 sein.
3) Den Code haben wir in das entsprechende Unterverzeichnis eingebettet, kompiliert, simuliert und programmiert.
4) Danach haben wir mit dem Oszilloskop die Zeit zwischen zwei Spitzen gemessen und den Code entsprechend angepasst.

Wenn man make main-sim ausführt, wird eine Datei namens test-output.vcd erzeugt, die wichtig für unsere Messungen ist. Diese Datei können wir mit gtkwave öffnen. In gtkwave haben wir dann die Zeit zwischen zwei Spitzen gemessen und festgestellt, dass die Anzahl der Takte um 7 gestiegen ist, da RCALL 3 Takte und RET 4 Takte benötigt.

3.1 Fall a = 207 Takten

a)

betrieb.S

StartA:

rcall aufgabe3a ; 3 Takten (für den Aufruf) + 189 Takten (für die Ausführung der Funktion)
sbi PORTD,PD4 ; 2 Takten
cbi PORTD,PD3 ; 2 Takten
cbi PORTD,PD2 ; 2 Takten
sbi PORTD,PD3 ; 2 Takten
sbi PORTD,PD2 ; 2 Takten
nop ; 1 Takt
cbi PORTD,PD4 ; 2 Takten
rjmp StartA ; 2 Takten

funktion.S

.GLOBAL aufgabe3a
.TYPE aufgabe3a, @function

aufgabe3a:

block_a_start:

ldi R17,0 ; 1 Takt
ldi R16,46 ; 1 Takt

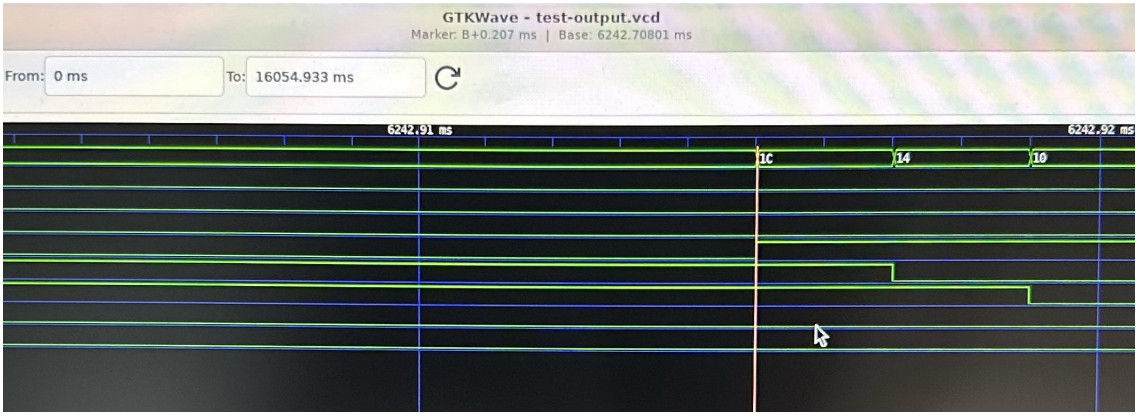
Blockinnen:
dec R16 ; 1 Takt
cp R16,R17 ; 1 Takt
brne Blockinnen ; 2 Takten > wenn der Sprung ausgeführt wird, 1 Takt > wenn der Sprung nicht ausgeführt wird

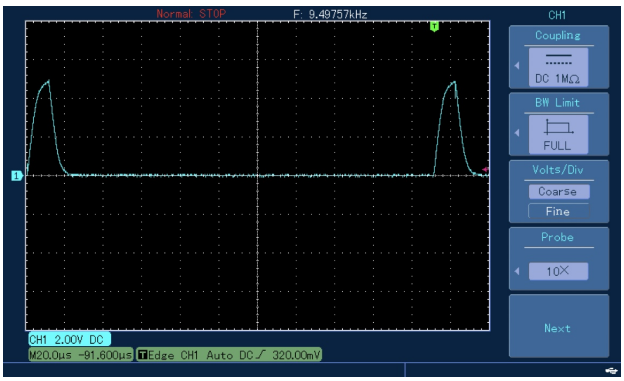
block_a_ende:

ret ; 4 Takten
.SIZE aufgabe3a, .-aufgabe3a

- Die Schleife wird 46 Mal ausgeführt:
- Für die ersten 45 Wiederholungen:
- 45 * (dec; 1 Takt + cp; 1 Takt + brne; 1 Takt) = 3 Takten = 180 Takten
- Für die letzte Wiederholung
- dec; 1 Takt + cp; 1 Takt + brne; 1 Takt = 3 Takte
- Gesamt Anzahl an Takten der Schleife: 183 Takten

Gesamtzahl der Takten:
2 (Initialisierung) + 183 (Schleife) + 4 (Rückkehr) = 189 Takten
189 + (3+2+2+2+2+2+1+2+2) = 189 + 18 = 207





3.2 Fall b = 8413 Takten

b)

betrieb.S

StartB:

```
rcall aufgabe3a ; 3 Takten
sbi PORTD,PD4 ; 2 Takten
cbi PORTD,PD3 ; 2 Takten
cbi PORTD,PD2 ; 2 Takten
sbi PORTD,PD3 ; 2 Takten
sbi PORTD,PD2 ; 2 Takten
nop ; 1 Takt
cbi PORTD,PD4 ; 2 Takten
rjmp StartB ; 2 Takten
```

funktion.S:

```
.GLOBAL aufgabe3b
.TYPE aufgabe3b, @function
```

```
aufgabe3b:


block_b_start:


main2:


ldi R16, 0x8; 1 Takt
ldi R17, 0x0; 1 Takt


LoopLabel:
.EQU cl, 255
ldi R18, 0; 1 Takt
ldi R19, cl; 1 Takt


LoopLabel_1:
inc R18; 1 Takt
cp R18, R19; 1 Takt
brne LoopLabel_1; 2 Takten, außer beim letzten Durchlauf 1 Takt



dec R16; 1 Takt
cp R16, R17; 1 Takt
brne LoopLabel; 2 Takten, außer beim letzten Durchlauf 1 Takt
ldi R30, 0; 1 Takt
ldi R29, 47; 1 Takt

LoopLabel_2:
inc R30; 1 Takt
cp R30, R29; 1 Takt
brne LoopLabel_2; 2 Takten, außer beim letzten Durchlauf 1 Takt

ldi R20, 22; 1 Takt

block_b_ende:

ret; 4 Takten
.SIZE aufgabe3b, .-aufgabe3b
```

- Die innere Schleife wird 255 mal durchgeführt
- 254 mal: 254 * (1+1+2) = 254 * 4 = 1016 Takten
- 1 mal: 1+1+1=3 Takten
- Gesamt: 1016+3= 1019 Takten

(Äußere Schleife)

- Die äußere Schleife wird 8 mal durchgeführt

376 -7 mal: 7 * (1+1+1+1+2+1019) = 7 * 1025 = 7175 Takten

- 1 mal: 1+1+1+1+1+1019 = 1024 Takten
- Gesamt: 7175 + 1024 = 8199 Takten

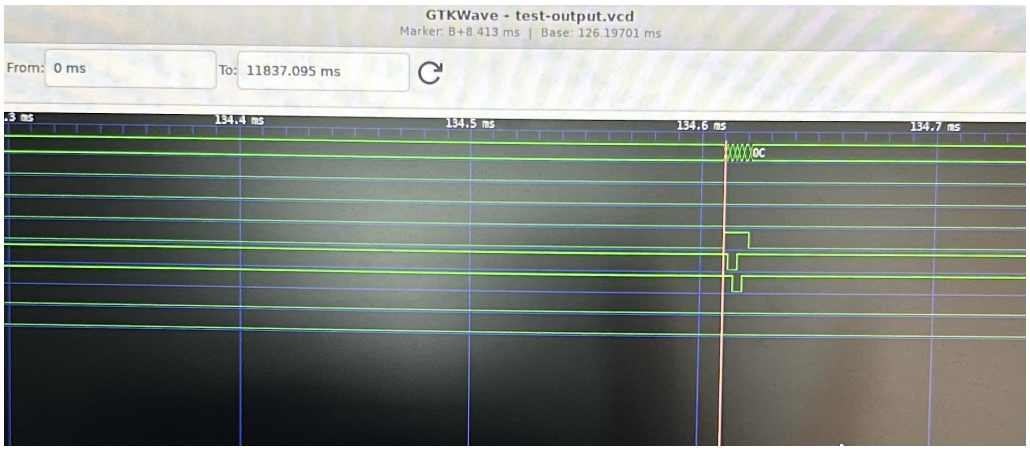
(Zweite Schleife)

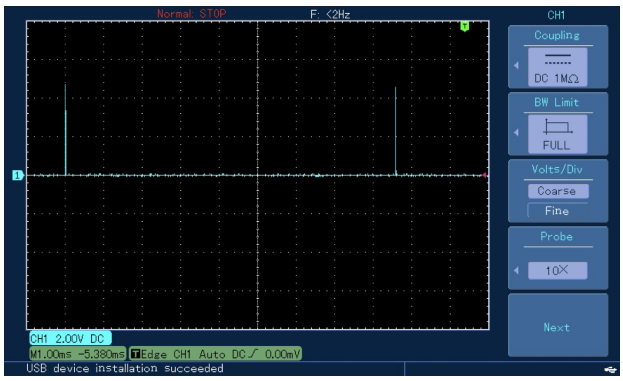
- Die zweite schleife wird 47 mal durchgeführt
- 46 mal: 46 * (1+1+2) = 46 * 4 = 184 Takten
- 1 mal: 1+1+1 = 3 Takten
- Gesamt: 184 + 3 = 187 Takten
- Gesamt für „aufgabe3b“

1+1+8199+1+1+187+1+4 = 8395 Takten

- Gesamt Taktenanzahl mit StartB:

3 (rcall aufgabe3b) + 8395 (aufgabe3b) + 2 (sbi PORTD,PD4) + 2 (cbi PORTD,PD3) + 2 (cbi PORTD,PD2) + 2 (sbi PORTD,PD3) + 2 (sbi PORTD,PD2) + 1 (nop) + 2 (cbi PORTD,PD4) + 2 (rjmp StartB) = 8413 Takte





4 Aufgabe04

In Aufgabe 4 mussten wir die Zero- und Carry-Werte mittels arithmetischer Befehle erzeugen und verschachtelte Schleifen in eine Schleife umwandeln, in dem wir adc und subc verwenden. Am Ende haben wir auch unser Code so angepasst, sodass der Taktanzahl derselbe bleibt wie vorher berechnet. Dann haben wir den Code wieder in den Mustercode eingebettet und dann simuliert und gemessen.

4.1 Fall a = 207 Takten

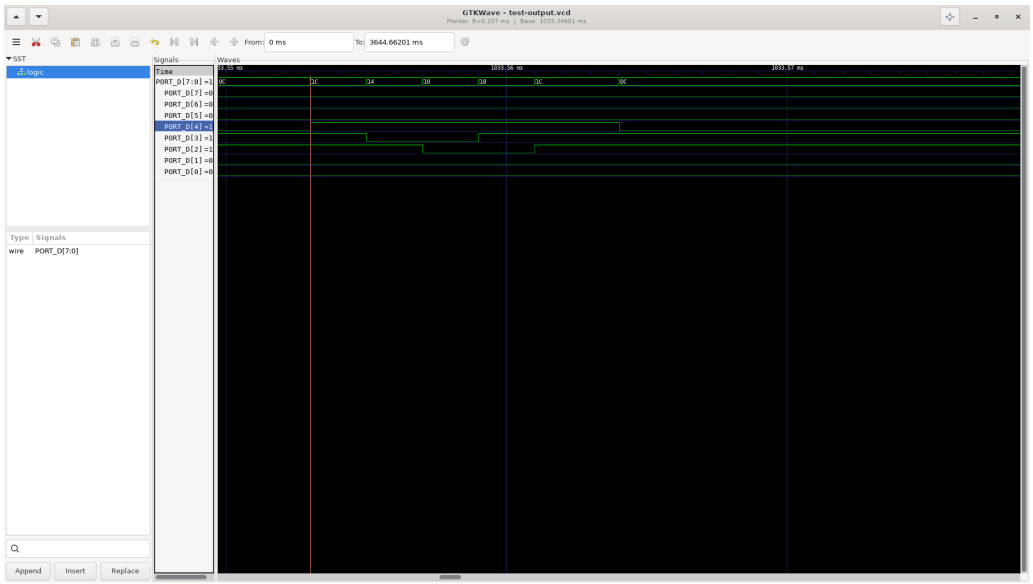
betrieb. S:

StartA:

```
rcall ex4a ; 3 Takt
sbi PORTD,PD4
cbi PORTD,PD3
cbi PORTD,PD2
sbi PORTD,PD3
sbi PORTD,PD2
nop
cbi PORTD,PD4
rjmp StartA
```

funktion.S:

```
.GLOBAL ex4a
.TYPE ex4a, @function
ex4a:

ldi R16, 0
ldi R17, 1
nop
nop
nop

block_a_start:
ldi R18, 45 ; 1 Takt

loop:
adc R18,R17 ; addiert den Wert von R17 zu R18 mit CarryBit
dec R18 ; dekrementiert den Wert von R18 um 1
brpl loop ; (Branch if plus) Springe zurück zu loop, wenn R18 > 0 2 Takte, sonst 1 Takt

block_a_ende:

ret ; 4 Takte

.SIZE ex4a, .-ex4a
```



4.2 Fall b = 8413 Takten

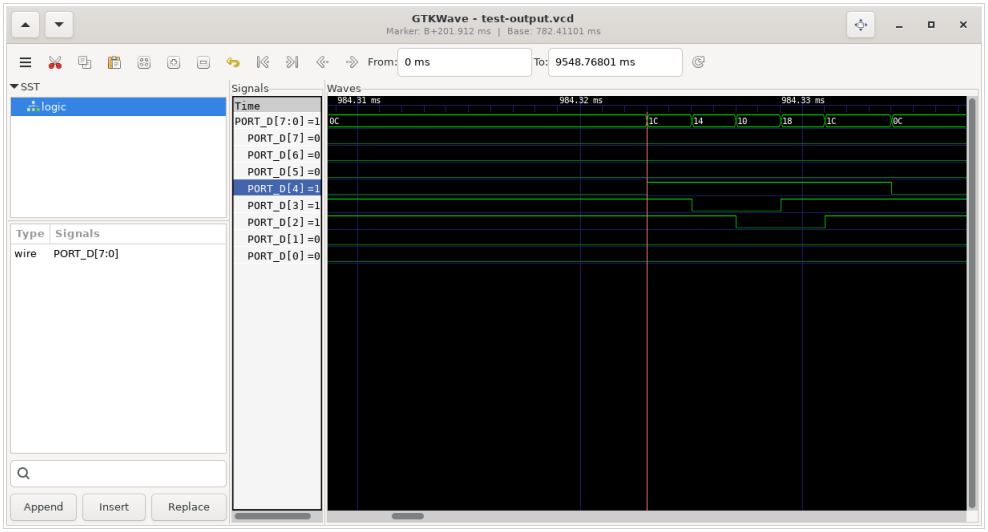
betrieb.S:

StartB:

```
rcall ex4a ; 3 Takte
sbi PORTD,PD4
cbi PORTD,PD3
cbi PORTD,PD2
sbi PORTD,PD3
sbi PORTD,PD2
nop
cbi PORTD,PD4
rjmp StartB
```

funktion.S:

```
.GLOBAL ex4b
.TYPE ex4b, @function
ex4b:

ldi R16, 1 ; 1 T
ldi R17, 0 ; 1 T

block_b_start:
ldi R18, 247 ; 1 T
ldi R20, 207 ; 1 T

Loop:

LargeLoop:
add R20,R16 , 1 T
adc R18,R17 , 1 T
brne Loop ; 2 T , sonst 1

ex4b_end:
ret ; 4 Takte

.SIZE ex4b, .-ex4b
```



5 Zusammenfassung

Zusammengefasst haben wir gelernt, wie man mit Assembler richtig arbeitet. Wir haben viele Codes geschrieben bis es genau exakt geklappt hat, wie es vorgesehen ist. Wir haben aber auch herausgestellt , dass es zeitintensiv ist, da man nicht direkt auf die vorgegebene Taktenanzahl kommt. Man muss ausprobieren und Zahlen ändern bis man auf eine richtige Lösung kommt. Probleme tauchen auf, manche Befehle laufen nicht so wie sein sollten und man stößt immer auf Fehler. Jedoch lernten wir auf unserem Lernweg, das man am Anfang immer Schwierigkeiten hat, aber im Lernprozess langsam verschwinden. Wir können sagen, dass wir viel gelernt haben über die Verbindung zwischen das praktische und theoretische. Als Quellen benutzten wir nur die Vorlesungen von Herrn Lipskoch sowie das auf Elli hochgeladene Manual und die README-Datei.