



UNIVERSITÄT  
DES  
SAARLANDES

# PROGRAMMIERUNG 2 - SS16

## PROJEKT 1 - MIPS-POTPOURRI

---

Autoren: Gregory Stock    Jan Baumeister

27. April 2016

Universität des Saarlandes

# GRUNDLAGEN

---

`git config` dient der Konfiguration von Git Repositories

- `--global` richtet die globale Konfiguration ein
  - `user.name` Name des Nutzers
  - `user.email` Email des Nutzers

`git config` dient der Konfiguration von Git Repositories

- `--global` richtet die globale Konfiguration ein
  - `user.name` Name des Nutzers
  - `user.email` Email des Nutzers

## Beispiel

```
git config --global user.name 'Konrad Klug'
```

Wir können das Projekt mit `git clone` unter folgender URL beziehen:

```
https://prog2scm.cdl.uni-saarland.de/git/project1/<username>
```

`<username>` = Euer Benutzername auf der Prog2-Website

Wir können das Projekt mit `git clone` unter folgender URL beziehen:

```
https://prog2scm.cdl.uni-saarland.de/git/project1/<username>
```

<username> = Euer Benutzername auf der Prog2-Website

## Beispiel

```
git clone https://.../project1/s8konrad project1
```

Wir können das Projekt mit `git clone` unter folgender URL beziehen:

```
https://prog2scm.cdl.uni-saarland.de/git/project1/<username>
```

`<username>` = Euer Benutzername auf der Prog2-Website

## Achtung!

Die Repositories sind nur innerhalb des Uninetzes erreichbar. Von außerhalb kann man eine VPN-Verbindung zum Uninetz einrichten.

Eine Anleitung steht auf der Website unter [Software](#).

# MARS MIPS SIMULATOR

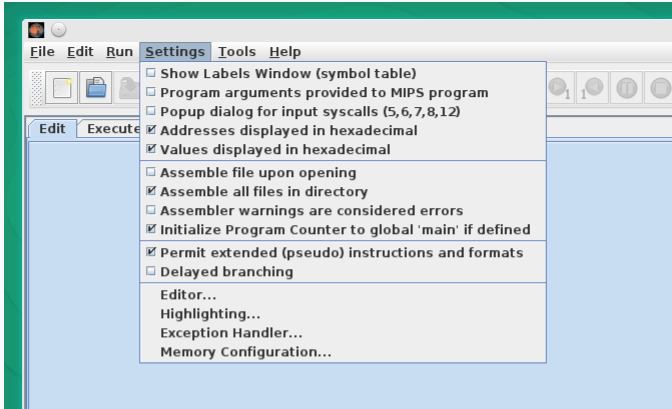
---



# MARS EINSTELLUNGEN

## Achtung

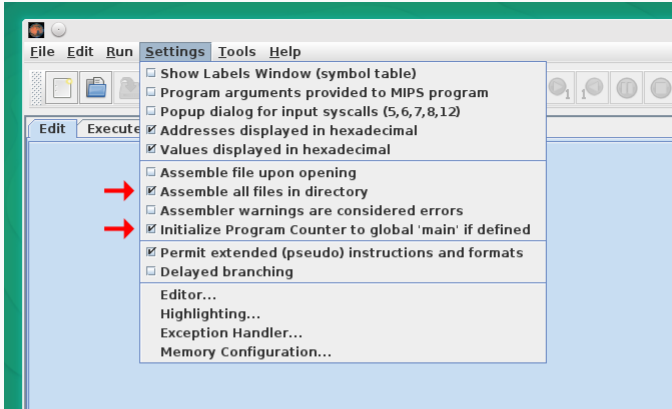
Wir müssen zwei Einstellungen anpassen.



# MARS EINSTELLUNGEN

## Achtung

Wir müssen zwei Einstellungen anpassen.



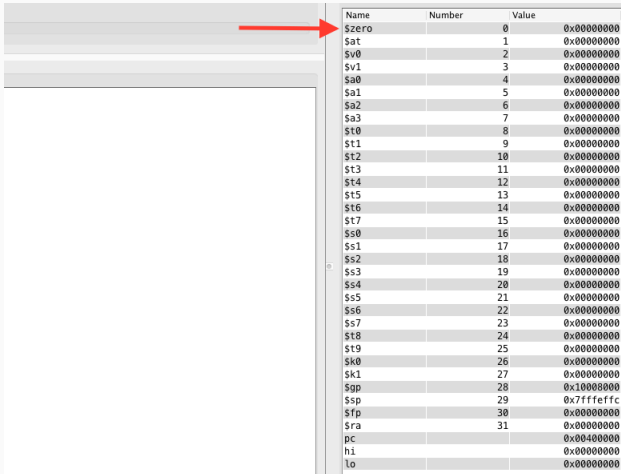
The screenshot shows the MARS 4.3 Help window. The 'Help' menu is open, and the 'MIPS' tab is selected. The 'Load & Store addressing modes, pseudo instructions' section is highlighted in green. Below this, a table lists various instructions and their descriptions.

Basic Instructions	Extended (pseudo) Instructions	Directives	Syscalls	Exceptions
abs.d \$f2,\$f4	Floating point absolute value double precision : Set \$f2 to absolute value of \$f4			
abs.s \$f0,\$f1	Floating point absolute value single precision : Set \$f0 to absolute value of \$f1			
add \$t1,\$t2,\$t3	Addition with overflow : set \$t1 to (\$t2 plus \$t3)			
add.d \$f2,\$f4,\$f6	Floating point addition double precision : Set \$f2 to double-precision sum of \$f4 and \$f6			
add.s \$f0,\$f1,\$f3	Floating point addition single precision : Set \$f0 to single-precision sum of \$f1 and \$f3			
addi \$t1,\$t2,-100	Addition immediate with overflow : set \$t1 to (\$t2 plus signed 16-bit immediate -100)			
addiu \$t1,\$t2,-100	Addition immediate unsigned without overflow : set \$t1 to (\$t2 plus signed 16-bit immediate -100)			
addu \$t1,\$t2,\$t3	Addition unsigned without overflow : set \$t1 to (\$t2 plus \$t3), no overflow			
and \$t1,\$t2,\$t3	Bitwise AND : Set \$t1 to bitwise AND of \$t2 and \$t3			
andi \$t1,\$t2,100	Bitwise AND immediate : Set \$t1 to bitwise AND of \$t2 and zero-extended immediate 100			
bc1f 1,label	Branch if specified FP condition flag false (BC1F, not BCLF) : If Coprocessor 1 condition flag 0 false, branch to label			
bc1f label	Branch if FP condition flag 0 false (BC1F, not BCLF) : If Coprocessor 1 condition flag 0 false, branch to label			
bc1t 1,label	Branch if specified FP condition flag true (BC1T, not BCLT) : If Coprocessor 1 condition flag 0 true, branch to label			
bc1t label	Branch if FP condition flag 0 true (BC1T, not BCLT) : If Coprocessor 1 condition flag 0 true, branch to label			
beq \$t1,\$t2,label	Branch if equal : Branch to statement at label's address if \$t1 and \$t2 are equal			

## Bedeutung der einzelnen Register

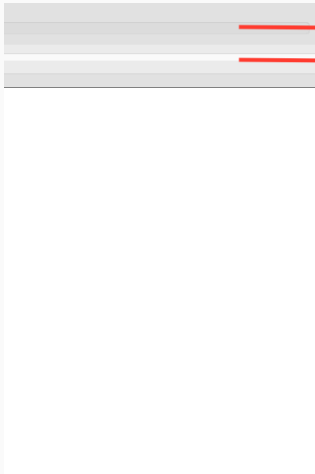
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

## Bedeutung der einzelnen Register



Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

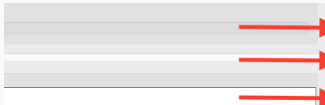
## Bedeutung der einzelnen Register



The diagram on the left shows a stack of horizontal bars representing memory or register allocation. Two red arrows point from these bars to the first two rows of the register table on the right, specifically to the '\$zero' and '\$v0' registers.

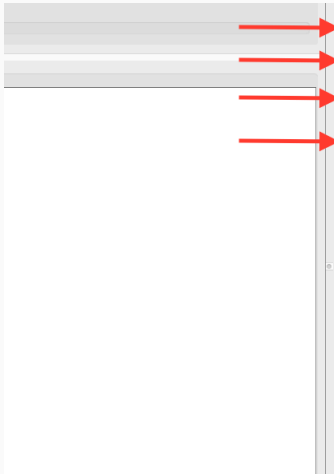
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

## Bedeutung der einzelnen Register



Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

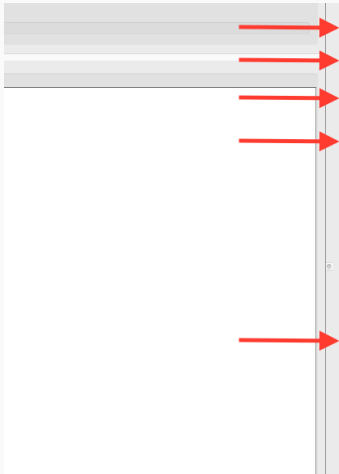
## Bedeutung der einzelnen Register



Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

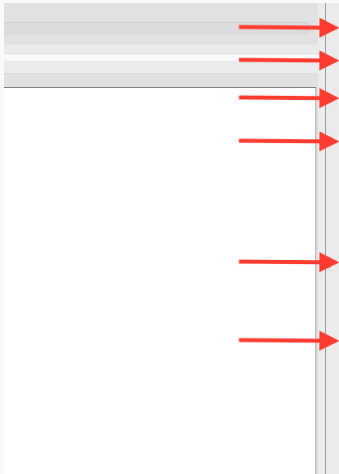


## Bedeutung der einzelnen Register



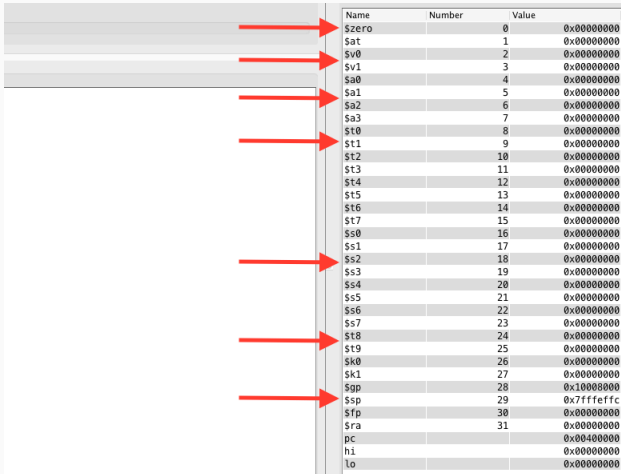
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

## Bedeutung der einzelnen Register



Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

## Bedeutung der einzelnen Register



The diagram illustrates the meaning of individual registers. On the left, a vertical column of gray boxes represents memory locations. Red arrows point from these boxes to specific registers in the table on the right. The table lists registers by name, number, and value.

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

# ÜBUNG LINKEDLIST

Wir möchten in MIPS eine verkettete Liste von Zahlen ausgeben.

```
.data
L1:
    .word L2
    .word 1
L2:
    .word L3
    .word 2
L3:
    .word L4
    .word 3
L4:
    .word L5
    .word 4
L5:
    .word 0
    .word 5
```

# ÜBUNG LINKEDLIST - LÖSUNG

## main.s

```
.data
L1:
.word L2
.word 1
L2:
.word L3
.word 2
L3:
.word L4
.word 3
L4:
.word L5
.word 4
L5:
.word 0
.word 5

.text
.globl main

main:
    la    $a0 L1
    jal   print_list
    li    $v0 10
    syscall
```

## print.s

```
.text
.globl print_list

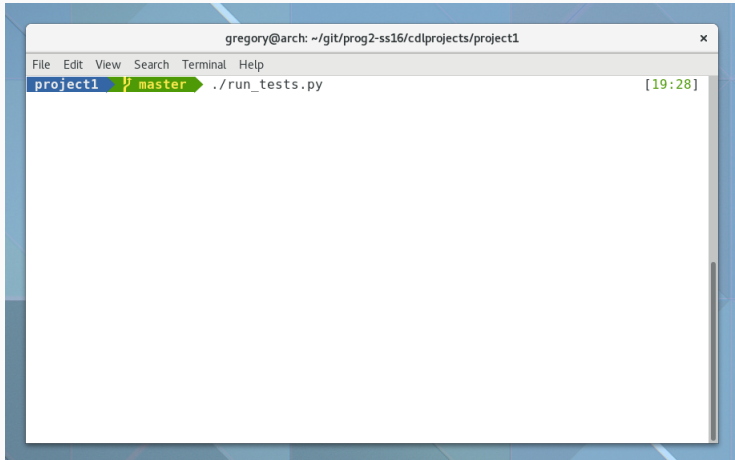
print_list:
    move $t0 $a0           # $a0 Anfangsadresse der Liste

print_loop:
    lw    $t1 4($t0)       # lade aktuelle Zahl
    move $a0 $t1
    li    $v0 1            # gebe Zahl auf Konsole aus
    syscall
    li    $a0 32           # gebe Leerzeichen auf Konsole aus
    li    $v0 11
    syscall
    lw    $t0 ($t0)        # lade die neue Adresse
    bnez $t0 print_loop
end:
    jr    $ra
```

Wir können in unserem Projektordner mit `./run_tests.py` die Public Tests ausführen.

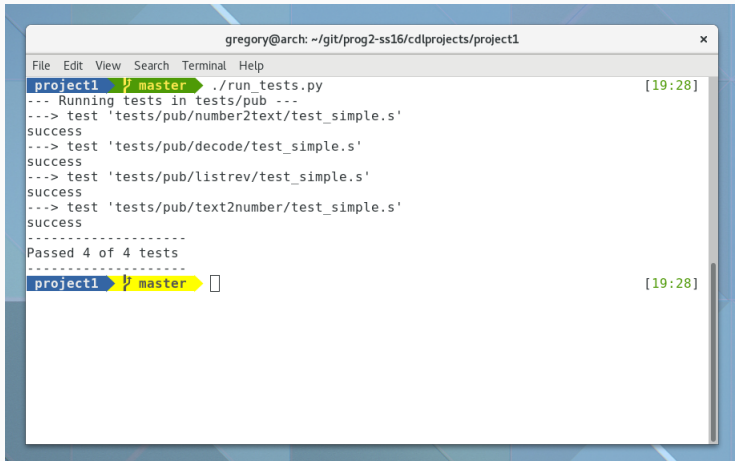
# PUBLIC TESTS

Wir können in unserem Projektordner mit `./run_tests.py` die Public Tests ausführen.



# PUBLIC TESTS

Wir können in unserem Projektordner mit `./run_tests.py` die Public Tests ausführen.



```
gregory@arch: ~/git/prog2-ss16/cdlprojects/project1
File Edit View Search Terminal Help
project1 master ./run_tests.py [19:28]
--- Running tests in tests/pub ---
---> test 'tests/pub/number2text/test_simple.s'
success
---> test 'tests/pub/decode/test_simple.s'
success
---> test 'tests/pub/listrev/test_simple.s'
success
---> test 'tests/pub/text2number/test_simple.s'
success
-----
Passed 4 of 4 tests
-----
project1 master [19:28]
```



FRAGEN?

# MIPS-POTPOURRI

---

## Zahl zu Text

- `$a0` enthält Adresse genau hinter dem Ende des Zielpuffers
- `$a1` enthält die umzuwandelnde Zahl
- `$a2` enthält die Basis im Intervall  $[2, 36]$ , in die die Zahl konvertiert werden soll
- `$v0` soll am Ende die Adresse des ersten Zeichens enthalten

## Zahl zu Text

- `$a0` enthält Adresse genau hinter dem Ende des Zielpuffers
- `$a1` enthält die umzuwandelnde Zahl
- `$a2` enthält die Basis im Intervall  $[2, 36]$ , in die die Zahl konvertiert werden soll
- `$v0` soll am Ende die Adresse des ersten Zeichens enthalten

## Beispiel

- `$a0` = 0x1000
- `$a1` = 90
- `$a2` = 16

# AUFGABE 1 - BEISPIEL

## Zahl zu Text

- $\$a0$  enthält Adresse genau hinter dem Ende des Zielpuffers
- $\$a1$  enthält die umzuwandelnde Zahl
- $\$a2$  enthält die Basis im Intervall  $[2, 36]$ , in die die Zahl konvertiert werden soll
- $\$v0$  soll am Ende die Adresse des ersten Zeichens enthalten

## Beispiel

- $\$a0 = 0x1000$                        $\langle 90 \rangle_{10} \longrightarrow \langle 5A \rangle_{16} \longrightarrow "5A\backslash 0"$
- $\$a1 = 90$
- $\$a2 = 16$

### Text zu Zahl

- `$a0` enthält die Adresse des ersten Zeichens der Textdarstellung
- `$a1` enthält die Basis der gegebenen Zahl im Intervall  $[2, 36]$
- `$v0` soll am Ende die umgewandelte Zahl enthalten

### Text zu Zahl

- `$a0` enthält die Adresse des ersten Zeichens der Textdarstellung
- `$a1` enthält die Basis der gegebenen Zahl im Intervall [2, 36]
- `$v0` soll am Ende die umgewandelte Zahl enthalten

### Beispiel

- `$a0` = 0x1000
- `$a1` = 16

## AUFGABE 2 - BEISPIEL

### Text zu Zahl

- $\$a0$  enthält die Adresse des ersten Zeichens der Textdarstellung
- $\$a1$  enthält die Basis der gegebenen Zahl im Intervall  $[2, 36]$
- $\$v0$  soll am Ende die umgewandelte Zahl enthalten

### Beispiel

- $\$a0 = 0x1000$
- $\$a1 = 16$

Adresse	Hex	Dez	ASCII
0x1000	35	53	5
0x1001	41	65	A
0x1002	0	0	\0

"5A\0"  $\rightarrow \langle 5A \rangle_{16} \rightarrow \langle 90 \rangle_{10}$



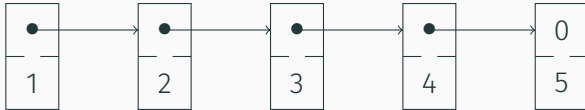
$$\text{NUMBER2TEXT}(\text{TEXT2NUMBER}(x)) = x$$

$$\text{TEXT2NUMBER}(\text{NUMBER2TEXT}(x)) = x$$

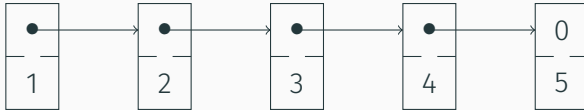
### Destruktives Listenreversieren

- `$a0` enthält die Adresse des ersten Elements der Liste
- `$v0` soll am Ende die Adresse des neuen ersten Elements enthalten

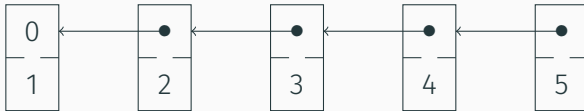
### Beispiel (Fortsetzung)



### Beispiel (Fortsetzung)



Wir reversieren die Liste:



## AUFGABE 4 - BEISPIEL

Adresse	Wert	Binär
0x10010000	0x46DAA2B5	01 00 01   10 11 01   10 10 10   10 00 10   10 11 01   01
0x10010004	0xCA28920B	11 00   10 10 00   10 10 00   10 01 00   10 00 00   10 11
0x10010008	0x70A62078	01   11 00 00   10 10 01   10 00 10   00 00 01   11 10 00
0x1001000C	0x00000000	00 00 00   00 00 00   00 00 00   00 00 00   00 00 00   00

## AUFGABE 4 - BEISPIEL

Adresse	Wert	Binär
0x10010000	0x46DAA2B5	01 00 01   10 11 01   10 10 10   10 00 10   10 11 01   01
0x10010004	0xCA28920B	11 00   10 10 00   10 10 00   10 01 00   10 00 00   10 11
0x10010008	0x70A62078	01   11 00 00   10 10 01   10 00 10   00 00 01   11 10 00
0x1001000C	0x00000000	00 00 00   00 00 00   00 00 00   00 00 00   00 00 00   00

Speicherauszug:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x46daa2b5	0xca28920b	0x70a62078	0x00000000

## AUFGABE 4 - BEISPIEL

Adresse	Wert	Binär
0x10010000	0x46DAA2B5	01 00 01   10 11 01   10 10 10   10 00 10   10 11 01   01
0x10010004	0xCA28920B	11 00   10 10 00   10 10 00   10 01 00   10 00 00   10 11
0x10010008	0x70A62078	01   11 00 00   10 10 01   10 00 10   00 00 01   11 10 00
0x1001000C	0x00000000	00 00 00   00 00 00   00 00 00   00 00 00   00 00 00   00

Speicherauszug:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x46daa2b5	0xca28920b	0x70a62078	0x00000000

ASCII-Darstellung:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	F . . .	. ( . \v	p . x	\0 \0 \0 \0

## AUFGABE 4 - EXTRAHIEREN

Wir haben das Wort `0x46DAA2B5` in Register `$t0` geladen.  
Nun möchten wir die orange markierten Bits extrahieren.

Anweisung	Inhalt Register \$t0	Hex
	01 00 01 10 11 01 10 10 10 10 00 10 10 11 01 01	0x46DAA2B5
<code>sll \$t0 \$t0 12</code>	10 10 10 10 00 10 10 11 01 01 00 00 00 00 00 00	0xAA2B5000
<code>srl \$t0 \$t0 26</code>	00 00 00 00 00 00 00 00 00 00 00 00 00 10 10 10	0x0000002A



FRAGEN?

BEI PROBLEMEN NUTZT DEN  
ASKBOT ODER KOMMT IN DIE  
OFFICE-HOURS!