

Unbeantwortete Fragen

`.set (noat|at)`

Warnung des Assemblers über die Benutzung des `$at`-Registers ein- oder ausschalten

```
.set noat           # Warnung ausschalten
lw $at, at_save
.set at            # Warnung einschalten
```

`sbrk` (`sbrk` - Linux man page)

- Increments the program's data space by increment bytes
- Datensegement vergrößern



Beispiel: Arithmetische Befehle

- Alle Befehle haben 3 Operanden
- Operand-Reihenfolge ist fest (Zielregister zuerst)
- Operanden einer arithmetischen Operation **müssen** in Registern stehen. Alle Register sind 32 Bit breit

Beispiele:

C-Code: $A = B + C$
MIPS-Code: `add $s0, $s1, $s2`

C-Code: $A = B + C + D;$
 $E = F - A;$
MIPS-Code: `add $t0, $s1, $s2`
 `add $s0, $t0, $s3`
 `sub $s4, $s5, $s0`



Befehlssatz

Logische Befehle

- logisches AND `and rd,rs,rt` `andi rd,rs,imm`
- logisches NOR `nor rd,rs,rt`
- logische Invertierung `not,rdest,rsrc`
- logisches XOR `xor rd,rs,rt` `xori rd,rs,imm`
- logisches OR `or rd,rs,rt` `ori rd,rs,imm`
- bitweise Rotieren `rol/ror rdest,rsrc1,rsrc2`
- bitweise Schieben `sll rd,rs,imm (imm = distance)`
`sllv rd,rs,rt`
`sra rd,rs,imm (imm = distance)`
`srlv rd,rs,rs`



Befehlssatz

Befehle zum Laden von Konstanten

- Laden einer Konstante in ein Register

`li rdest,imm`

`lui rdest,imm`

Vergleichsbefehle

- Vergleich zweier Register

`slt/sltu rd,rs,rt`

`slti/sltiu rd,rs,imm`

`seq rdest,rsrc1,rsrc2`

`sge/sgeu rdest,rsrc1,rsrc2`

`sgt/sgtu rdest,rsrc1,rsrc2`

`sle/sleu rdest,rsrc1,rsrc2`

`sne rdest,rsrc1,rsrc2`

- Vergleich eines Registers mit Null



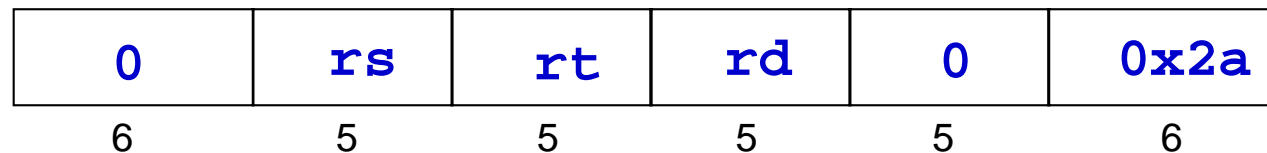
Vergleichsbefehle

```
if $s1 < $s2 then
    $t0 = 1
else
    $t0 = 0
```

} `slt $t0, $s1, $s2`

`slt rd, rs, rt`

slt: set less than



Befehlssatz

Kontrollflussbefehle

- unbedingtes Verzweigen zu einer Adresse
`j target b label`
- unbedingtes Verzweigen zu einer Adresse und sichern der nachfolgenden Adresse (für Unterprogramme)
`jal target`
- Verzweigen, wenn Bedingungs-Flag eines Coprozessors wahr/falsch ist
`bczt label bczf label`
- Verzweigen, wenn ein Register größer/kleiner als ein anderes Register ist
`bgt rsrc1,rsrc2,label bgtu rsrc1,rsrc2,label`
`blt rsrc1,rsrc2,label bltu rsrc1,rsrc2,label`
(auch `bge, bgeu, ble, bleu`)



Befehlssatz

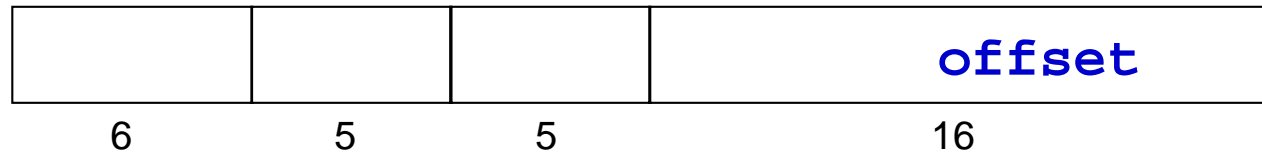
Kontrollflussbefehle

- Verzweigen, wenn zwei Register gleich/ungleich sind
`beq rs,rt,label` `bnq rs,rt,label`
- Verzweigen, wenn ein Register größer/kleiner als Null ist
`bgtz rs, label` `bltz rs,label`
(auch `bgez, blez`)
- Verzweigen, wenn ein Register gleich/ungleich Null ist
`beqz rsrc, label` `bnez rs,label`



Kontrollflussbefehle

Branch



Offset: 16-bit, vorzeichenbehaftet

→ $2^{15}-1$ Befehle vorwärts und 2^{15} rückwärts

jump



26-bit Adress-Feld



Kontrollflussbefehle

Einstufige Verzweigung:

Eine einstufige Verzweigung wird durch einen bedingten Sprung realisiert

if-Anweisung

```
if ( register_s1 == 0 )
{
    if-Anweisungen
}
next-Teil;
```

Assembler-Code

```
beqz $s1, marke1
j    marke2

marke1: { ..... } if-Anweisungen
        { ..... }
        { ..... }

marke2: ..... next-Teil
```



Kontrollflussbefehle

Zweistellige Verzweigung:

if-else-Anweisung

```
if (register_s1 == 0)
{
    if-Anweisungen
}
else
{
    else-Anweisungen
}
next;
```

Assembler-Code

```
beqz $s1, marke1
{
    .....
} else-Anweisungen
j     marke2

marke1: {
    .....
} if-Anweisungen

marke2: ..... next-Teil
```



Kontrollflussbefehle

Bedingte Verzweigung

```
bne $t0, $t1, Label
```

```
beq $t0, $t1, Label
```

```
if (i==j)  
    h = i + j;
```

```
bne $s0, $s1, Label  
add $s3, $s0, $s1
```

```
Label:    ....
```



Kontrollflussbefehle

Unbedingte Verzweigung

`j label`

```
if (i!=j)
    h=i+j;
else
    h=i-j;

beq $s4, $s5, Lab1
add $s3, $s4, $s5
j Lab2
Lab1: sub $s3, $s4, $s5
Lab2: ...
```



Befehlssatz

Lade- und Speicherbefehle

- Laden einer Adresse

`la rdest, address`

- Laden und Speichern eines Bytes, Halbwortes, Wortes und Doppelwort

`lb rt, address` `sb rt, address`

`lbu rt, address`

`lh rt, address` `sh rt, address`

`lhu rt, address`

`lw rt, address` `sw rt, address`

`ld rt, address` `sd rt, address`

- Laden und Speichern von Koprozessor-Registern

`lwcx rt, address` `swcx rt, address` (z=1, FPU)



Befehlssatz

Lade- und Speicherbefehle:

- Laden und Speichern von/an nicht-ausgerichtete Adressen

`lwl rt, address`

`lwr rt, address`

`ulh rdest, address`

`ulhu rdest, address`

`ulw rdest, address`

`ulhu rdest, address`

`ush rsrc, address`

`usw rsrc, address`

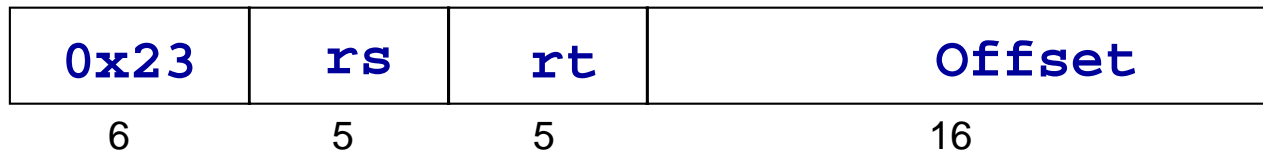


Lade und Speicherbefehle

- Laden/Speichern von Bytes, Halbwörter und Wörter

lw rt, address

Lade das 32-Bit Wort an der Adresse **address** ins Register **rt**



sw rt, address

Speichere das 32-Bit Wort im Register **rt** an der Adresse **address**



Beispiel

Lade- und Speicher-Befehle :

C-Code: `A[8] = h + A[8];`

MIPS code:
`lw $t0, 32($s3)`
`add $t0, $s2, $t0`
`sw $t0, 32($s3)`



Unterscheid zwischen **lb** und **lbu**

```
.data
result: .word 0x89abcdef 0x79abcdef

.text

# Start des Hauptprogrammes

        .globl main
main:   ...
        lbu $a0, result
        # $a0 enthaelt 0x00000089 0x00000079
        ...
        lb $a0, result
        # $a0 enthaelt 0xffffffff89 0x00000079
        ...
        jr $ra
```



MIPS-Speichermodell "Big-Endian"

Big-Endian:

Höchstwertigstes Byte liegt an kleinster Adresse

```
result:      .word 0x89abcdef
```

```
lbu $a0, result
```

```
lbu $a1, result+1
```

```
lbu $a2, result+2
```

```
lbu $a3, result+3
```

```
# $a0 enthaelt 0x89
```

```
# $a1 enthaelt 0xab
```

```
# $a2 enthaelt 0xcd
```

```
# $a3 enthaelt 0xef
```

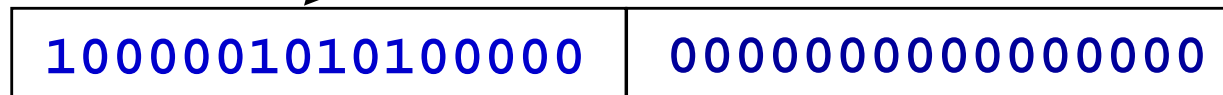
In SPIM wird die Konvention des unterliegenden Rechners verwendet.



Laden von 32-Bit Operanden

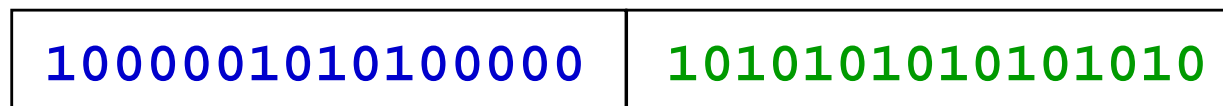
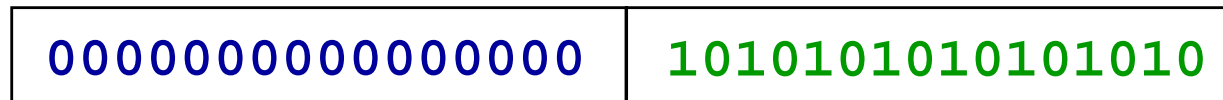
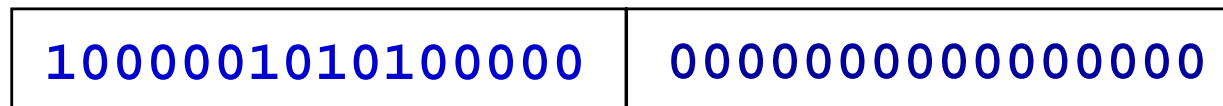
10000010101000001010101010101010 → \$t0

lui \$t0, 1000001010100000 (load upper immediate)



mit Nullen ausfüllen

ori \$t0, \$t0, 1010101010101010



Der globale Zeiger \$gp

Lade-Befehl (Pseudoinstruktion):

```
lw rt, address                lw $v0, 0x10008000
Adresse liegt im Datensegment
```

Bisherige Lösung:

```
lui $at, 0x1000                lui $at, 0x1000
lw rt, Offset($at)            lw $v0, 0x8000($at)
```

Offset := vier niedrigstwertige Stellen von adresse

Bessere Lösung:

```
lw rt, Offset($gp)            lw $v0, 0($gp)
```

Der globale Zeiger \$gp enthält immer den Wert **1000 8000**₁₆

→ Zugriff auf die Adressen **1000 0000**₁₆ bis **1001 0000**₁₆

Befehlssatz

Transportbefehle:

- Verschieben eines Registerinhalts in ein anderes Register

```
move rdest, rsrc
```

- Laden des Registers HI oder LO in ein allgemeines Register

```
mfhi rd          mflo rd
```

```
mthi rs          mtlo rs
```

- Verschieben von/nach Registern in Koprozessoren

```
mfcz rt, rd      mtcz rd, rt
```

```
mfc1 rt, rd      mtcl rd, rt
```

```
mfcl.d rdest, frsrc1
```

(frsrc1 und frsrc1+1 in die CPU-Register rdest und rdest+1)



Befehlssatz

Befehle für Fließkomma-Arithmetik:

➤ Arithmetik

`abs.d fd, fs`

`abs.s fd, fs`

`add.d fd, fs, ft`

`add.s fd, fs, ft`

`sub.d fd, fs, ft`

`sub.s fd, fs, ft`

`mul.d fd, fs, ft`

`mul.s fd, fs, ft`

`div.d fd, fs, ft`

`div.s fd, fs, ft`

➤ Vergleiche

`c.eq.d fs, ft`

`c.eq.s fs, ft`

`c.le.d fs, ft`

`c.le.s fs, ft`

`c.lt.d fs, ft`

`c.lt.s fs, ft`

➤ Laden und Speichern

`l.d fdest, address`

`l.s fdest, address`

`s.d fdest, address`

`s.s fdest, address`



Befehlssatz

Unterbrechungs- und Ausnahmebehandlung:

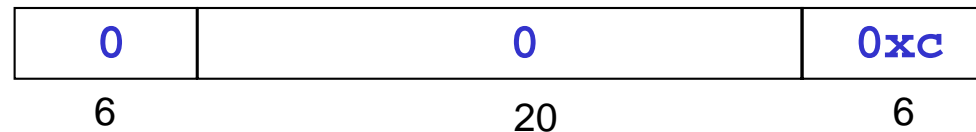
- Wiederherstellen des Status-Registers nach einer Unterbrechung

rfe



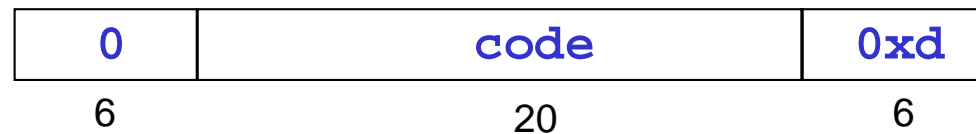
- Systemaufruf

syscall



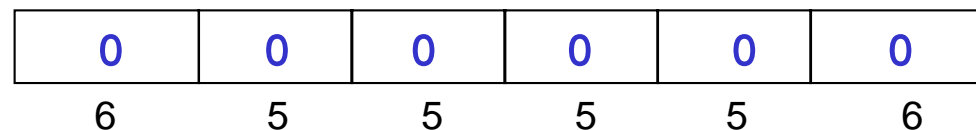
- Software-Unterbrechung

break code



- Dummy-Operation

nop



Wichtige MIPS Befehle

□ Arithmetik:

- `add rd, rs, rt` bzw. `addi rt, rs, imm`
- `mul rdest, rsrc1, src2`
- `sub rd, rs, rt`

□ Verzweigungen

- `slt rd, rs, rt` bzw. `slti rd, rs, imm`
- `beq rs, rt, label`
- `bne rs, rt, label`

□ Sprünge

- `j target`
- `jal target`
- `jr rs`

□ Lade-Speicherbefehle (Transportbefehle)

- `lui rt, imm`
- `lw rt, address`
- `sw rt, address`



Ersetzung von Pseudoinstruktionen

Pseudoinstruktion	wird ersetzt durch
<code>move Rd, Rs</code>	<code>addu Rd, \$0, Rs</code>
<code>neg Rd, Rs</code>	<code>sub Rd, \$0, Rs</code>
<code>b sym</code>	<code>bgez \$0, sym</code>
<code>li Rd, Imm</code>	<code>ori Rd, \$0, Imm</code>
<code>la Rd, sym</code>	<code>lui \$at, [sym / 10000₁₆] ori Rd, \$at, sym & FFFF₁₆</code>
<code>l.d Fd, sym</code>	<code>lui \$at, [sym / 10000₁₆] lwcl Fd, sym & FFFF₁₆(\$at) lui \$at, [sym / 10000₁₆] lwcl Fd + 1, sym & FFFF₁₆(\$at)</code>



Ersetzung von Pseudoinstruktionen

Pseudoinstruktion	wird ersetzt durch
<code>bge Ra, Rb, sym</code>	<code>slt \$at, Ra, Rb</code> <code>beq \$at, \$0, sym</code>
<code>abs Rd, Rs</code>	<code>addu Rd, \$0, Rs</code> <code>bgez Rs, lbl</code> <code>sub Rd, \$0, Rs</code> <code>lbl:</code>
<code>rol Rd, Rs, dist</code>	<code>srl \$at, Rs, 32 - dist</code> <code>sll \$at, Rd, Rs, dist</code> <code>or Rd, Rd, \$at</code>
<code>rem Rd, Ra, Rb</code>	<code>bne Rb, \$0, lbl</code> <code>break 0</code> <code>lbl: div Ra, Rb</code> <code>mfhi Rd</code>
<code>nop</code>	<code>or \$0, \$0, \$0</code>



Programmiertechniken

C:

```
if( ) {...}
  else { if( ) {...}
         else { if( ) {...}
                else {...}
              }
        }
      }
```

C:

```
switch (note)
{
  case1:    1-Anweisungen
            break;
  case2:    2-Anweisungen
            break;
  ...
  ...
  ...
  case6:    6-Anweisungen
            break;
  default:  Fehlermeldung
            break;
}
```



Programmiertechniken

```
# Note steht in der
# Variablen note

lw    $t0, note

li    $t1, 1
li    $t2, 2
     ....
li    $t5, 5

beq   $t0, $t1, marke1
beq   $t0, $t2, marke2
     ....
     ....
     ....

beq   $t0, $t5, marke5
     .... # default

b     weiter
```

```
marke1:  ....           # Note 1
         ....
         b weiter

marke2:  ....           # Note 2
         ....
         b weiter

         .
         .
         .
         .

marke5:  ....           # Note 5
         ....
         b weiter

weiter:  ....           # Mat.-Nr. ....
         ....           # hat die Note:
```



Programmiertechniken

c-Code

```
int x = 12;  
int y = 34;  
x = x + y;
```

MIPS-Assembler

```
addi $t1, $zero, 12  
addi $t2, $zero, 34  
add  $t1, $t1, $t2
```

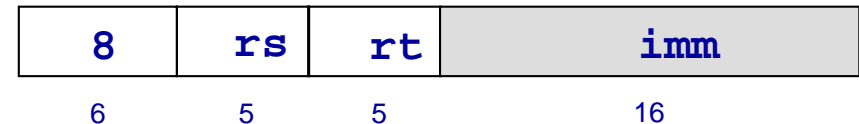


Programmiertechniken

Register \$t1 und \$t2 in den Speicher ab
der Adresse 0x1000 0004

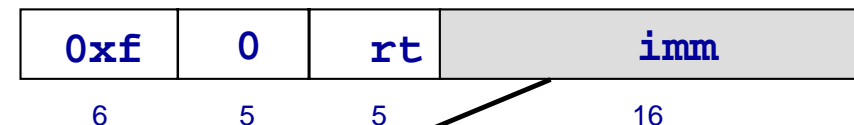
```
addi $s1, $zero, 0x1000 0004
```

```
addi rt,rs,imm
```



```
lui $s1, 0x1000
```

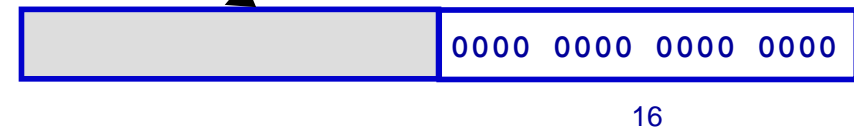
```
lui rt,rs,imm
```



```
ori $s1, 0x0004
```

```
sw $t1, 0($s1)
```

\$s1



```
sw $t2, 4($s1)
```



Programmiertechniken

c-Code

```
if (a < b)
    x = b;
else
    x = a;
```

MIPS-Code

```
lw $t0, a           # initialisiere $t0
lw $t1, b           # initialisiere $t1
slt $t2, $t0, $t1   # ($t0 < $t1)?
beq $t2, $zero, else # wenn die Bedingung nicht erfuehlt
                    # ist, gehe zu else
add $t3, $t1, $zero # es gilt: ($t0 < $t1)
j cont
else: add $t3, $t0, $zero # es gilt ($t0 >= $t1)
      # move $t0 to $t3 (result)

cont: ...
```



Programmieretechniken

C- Code

```
x[0] = x[1] + x[2];
```

MIPS Assemblercode:

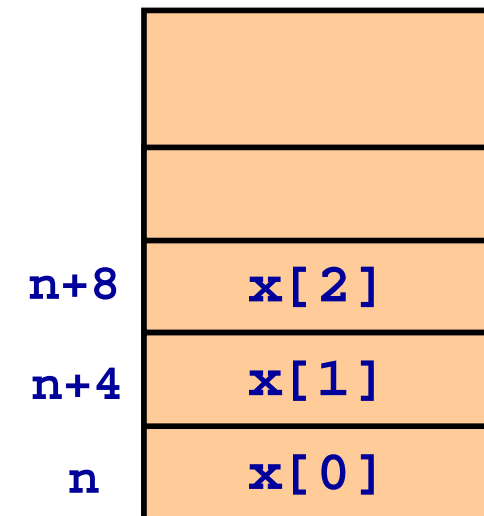
- Annahme: Startadresse n steht bereits in \$t1

```
lw $t2, 4($t1)
```

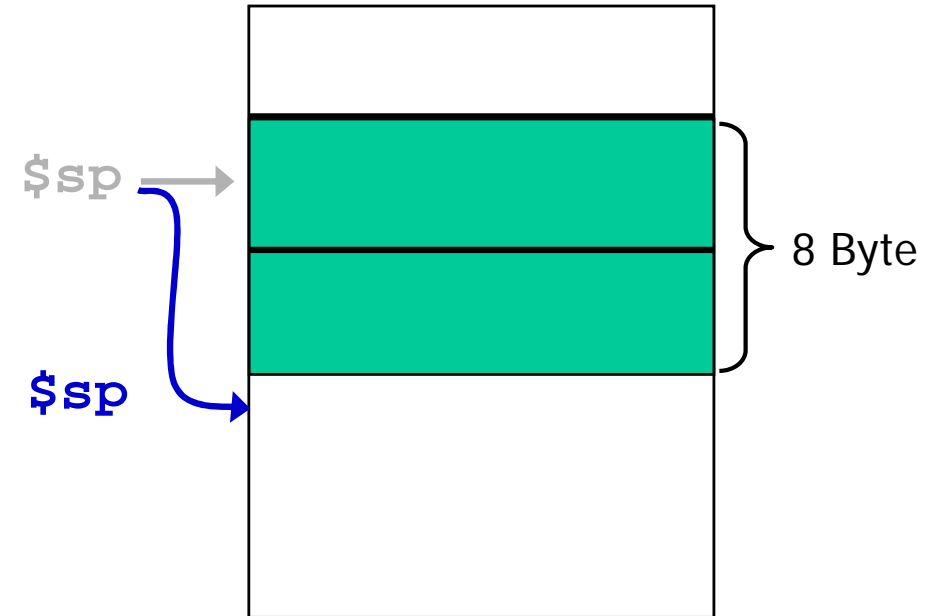
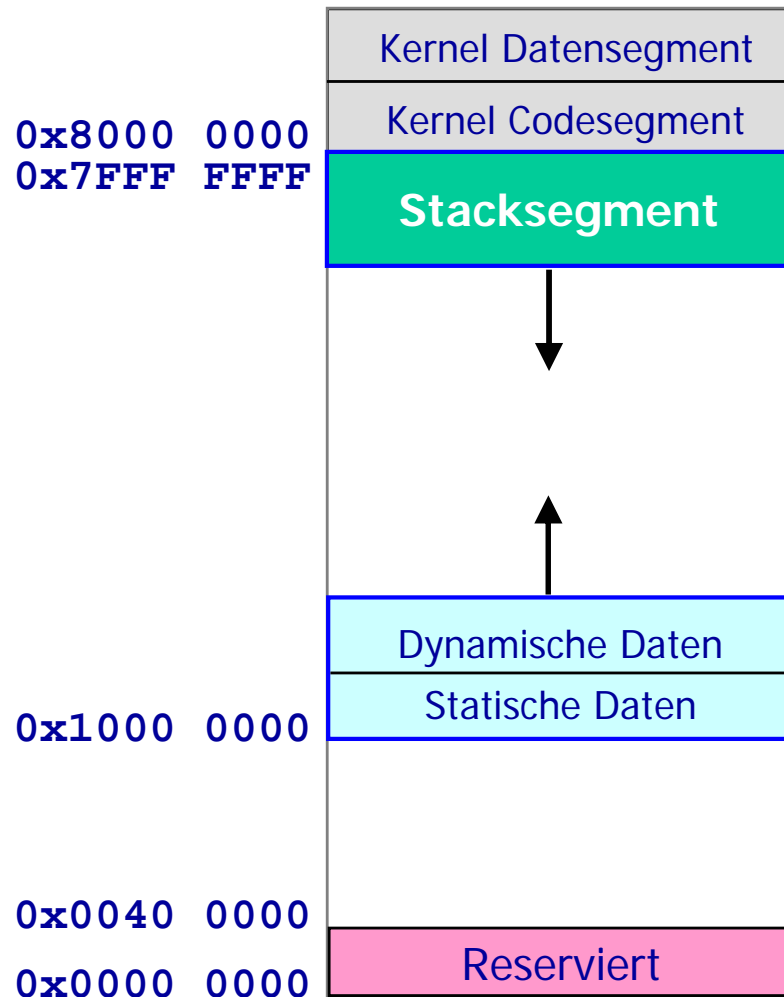
```
lw $t3, 8($t1)
```

```
add $t4, $t2, $t3
```

```
sw $t4, 0($t1)
```



Stackprogrammierung

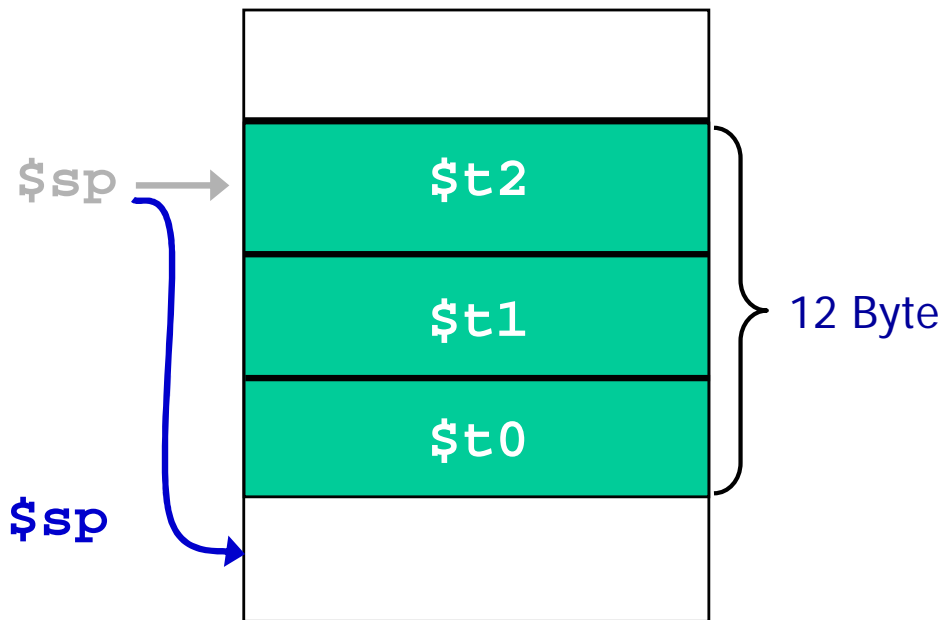


```
subu  $sp, $sp, 8
sw    $t0, 4($sp)
sw    $t1, 8($sp)
```

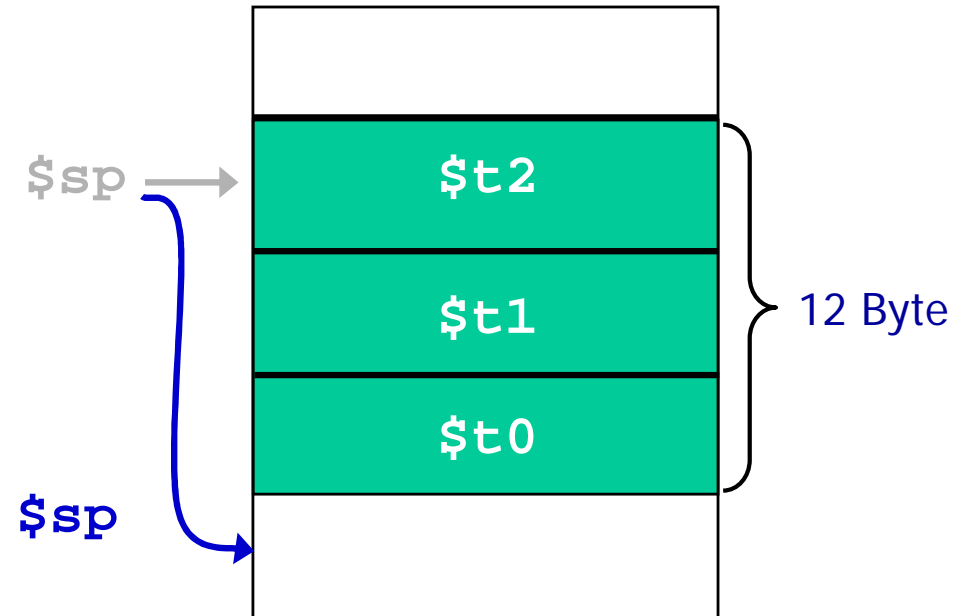


Stackprogrammierung

□ Lesen vom Stack:



```
subu  $sp, $sp, 12
sw    $t0, 4($sp)
sw    $t1, 8($sp)
sw    $t2, 12($sp)
```



```
lw    $t0, 4($sp)
lw    $t1, 8($sp)
lw    $t0, 12($sp)
addi  $sp, $sp, 12
```



Unterprogrammaufrufe

Beispiel (c-Code)

```
int min(int a, int b)
{
    if (a < b) return a;
    else return b;
}

main()
{
    int x = 123; int y = 456;
    int z = min (x,y);
}
```



Unterprogrammaufrufe

MIPS-Assembler:

```
    j min          # Sprung zu min
    ...          # Befehl nach Berechnung von min

min:  ...        # code fuer min(a,b)
    ...
    j ??        # zurueck zum Hauptprogramm
```

Probleme:

- Rücksprungadresse sichern und wiederherstellen
- Parameterübergabe

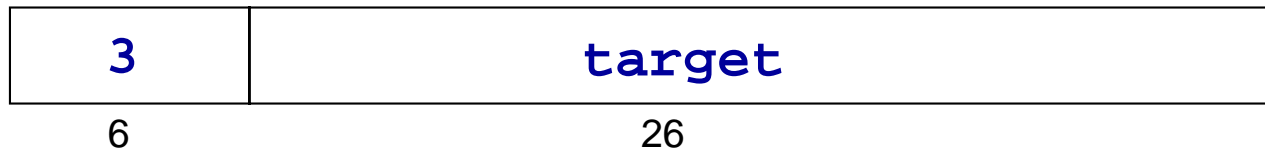


Unterprogrammaufrufe

- **Rücksprungadresse:** MIPS unterstützt den Funktionsaufruf durch speziellen Befehl

jal ("jump and link")

jal target



Springe zu target. Speichere die Adresse des nächsten Befehls in \$ra



Unterprogrammaufrufe

□ Parameterübergabe:

Konventionen zur Verwendung der Register, um Fehler zu vermeiden

- \$a0, ..., \$a3 Register für Argumente des Unterprogramms
- \$v0, \$v1 Register für Funktionswert bzw. für Ausgabeparameter des Unterprogramms
- \$t0-\$t9: Register können im Unterprogramm überschrieben werden
- \$s0-\$s7: Register dürfen nicht überschrieben werden



Unterprogrammaufrufe

```
lw $s0, x           # initialisiere $s0 mit x=123
lw $s1, y           # initialisiere $s1 mit y=456
add $a0, $s0, $zero # kopiere $s0 in Register $a0
                    # fuer Parameteruebergabe
add $a1, $s1, $zero # entsprechend $s1 in $a1
jal min             # springe zur Funktion min
add $s2, $v0, $zero # kopiere Funktionswert in
                    # Ergebnisregister
sw $s2, z           # speichere Ergebnis ab
...
```



Unterprogrammaufrufe

```
min:    add $t0, $zero, $a0      # initialisiere $t0 mit
      # Parameter a
      add $t1, $zero, $a1      # initialisiere $t1 mit
      # Parameter b
      slt $t2, $t1, $t0        # ($t1 < $t0)?
      beq $t2, $zero, else      # wenn die Bedingung nicht
      # erfuehlt ist, gehe zu else
      add $v0, $t1, $zero      # es gilt: ($t1 < $t0)
      j ende                    #

else:   add $v0, $t0, $zero      # es gilt ($t1 >= $t0)
      # move $t0 to $v0 (result)

ende:   j $ra                   # Ruecksprung
```



Unterprogrammaufrufe

- ❑ Regeln oder Konventionen zur Verwendung von **Registern Kellerspeicher (Stack)** bei einem Unterprogrammaufruf
- ❑ Informationen bezüglich des Unterprogramms und des unterbrochenen Programms werden in einem **Rahmen (Frame)** auf dem Stack gespeichert. Diese können:
 - Argumente des Unterprogramms
 - Lokale Variablen des Unterprogramms
 - Registerinhalte, die vom Unterprogramm nicht verändert werden dürfen



Unterprogrammaufrufe

Ein Argument ist eine Integer- oder Fließkomma-Zahl oder ein Zeiger auf eine größere Datenstruktur (z. B. Zeichenkette)

Aufgaben des Aufrufers:

- *Temporäre* Register **\$t0** bis **\$t9** sichern, falls gültige Daten darin enthalten sind
- Übergabe der ersten vier Argumente in den Registern **\$a0** bis **\$a3** (ggf. als Zeiger auf das Argument)
- Register **\$a0** bis **\$a3** sichern, falls die Argumente später noch gebraucht werden
- Übergabe weiterer Argumente auf dem Stack



Unterprogrammaufrufe

Aufruf eines Unterprogramms an der Adresse <addr>
mit dem Befehl

jal <addr>

Die Adresse des nachfolgenden Befehls wird im Register
\$ra gespeichert.

Informationen bezüglich des Unterprogramms werden in
einem Rahmen auf dem Stapel gespeichert.

**Rahmengröße := (Anzahl der Argumente +
Anzahl der zu sichernden Register +
Anzahl der lokalen Variablen) x 4**

(Zu sichernde Fließkommazahlen mit doppelter Genauigkeit benötigen
acht Bytes Speicherplatz)



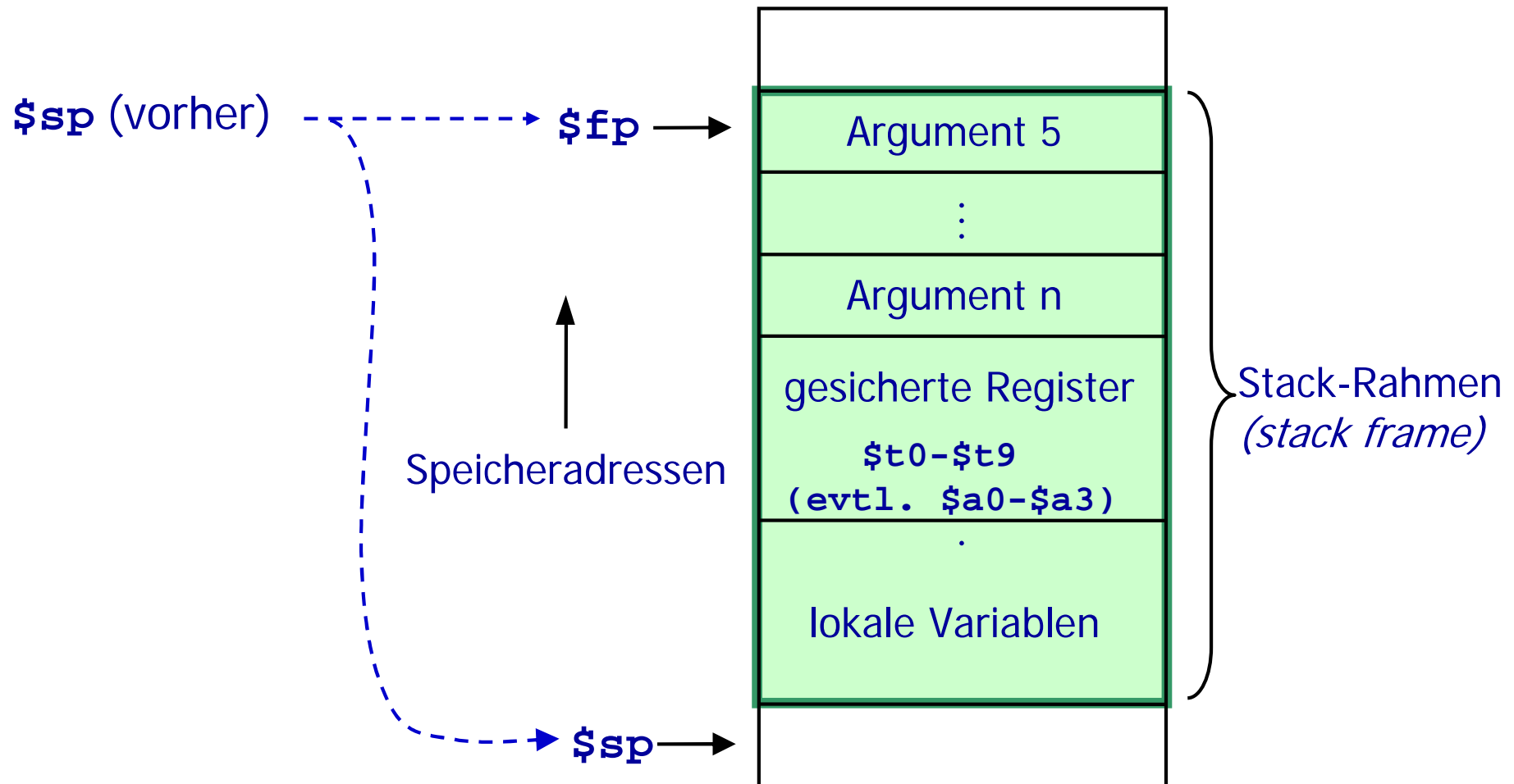
Unterprogrammaufrufe

Aufgaben des Aufgerufenen:

- Reservieren von Speicherplatz für einen neuen Rahmen durch Subtraktion der Rahmengröße von Stack-Pointer (**\$sp**)
- Sichern der Register **\$s0** bis **\$s7**, falls diese im Unterprogramm verwendet werden
- Sichern des Rücksprungadressenregisters **\$ra**, falls das Unterprogramm weitere Unterprogramme aufruft
- Sichern der Rahmenzeigers **\$fp**
- Anlegen eines neuen Rahmenzeigers durch Addition der Rahmengröße zum Stapelzeiger



Stack-Rahmen (*stack frame*)



Einfacher Unterprogrammaufruf

```
main:      jal subroutine
```

```
subroutine: # keine weiteren  
            # Unterprogrammaufrufe  
  
            # für lokale Variablen  
            # nur $t0 bis $t9  
  
            jr $ra
```



Beispiel für einen Unterprogrammaufruf

```
# Hauptprogramm
        .globl main
main:    subu $sp, $sp, 8
        sw $ra, 4($sp)
        sw $fp, 8($sp)
        addu $fp, $sp, 8

        jal subroutine

        lw $ra, 4($sp)
        lw $fp, 8($sp)
        addu $sp, $sp, 8
        jr $ra
```

**Unterprogrammaufrufe
mit jal-Befehl !**

```
.text
subroutine: subu $sp, $sp, 32
           s.d $f2, 4($sp)
           s.d $f4, 12($sp)
           sw $s0, 20($sp)
           sw $s1, 24($sp)
           sw $ra, 28($sp)
           sw $fp, 32($sp)
           addu $fp, $sp, 32

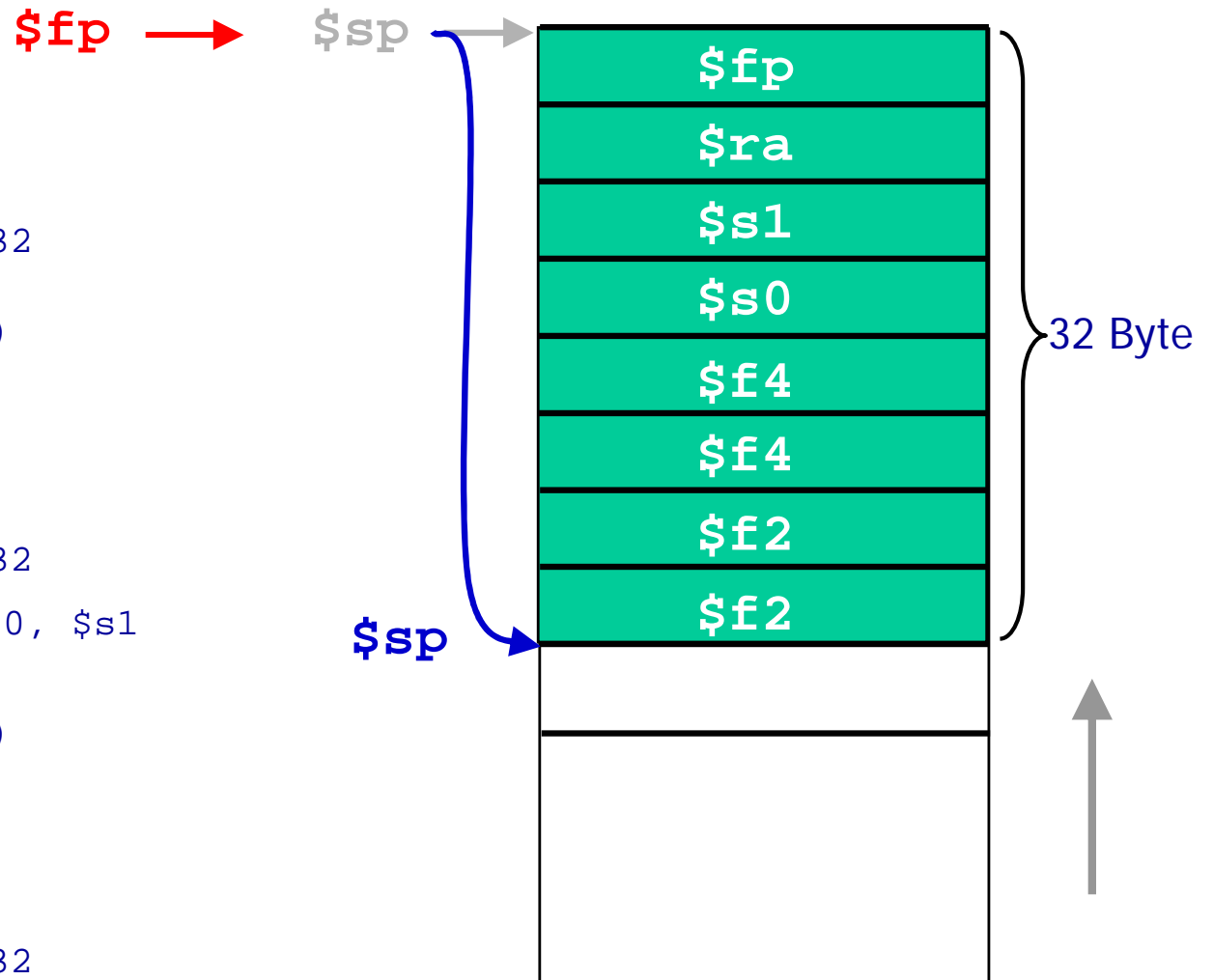
# Veraenderung von $fp2,$fp4,$s0,$s1
           l.d $f2, 4($sp)
           l.d $f4, 12($sp)
           lw $s0, 20($sp)
           lw $s1, 24($sp)
           lw $ra, 28($sp)
           lw $fp, 32($sp)
           addu $sp, $sp, 32
           jr $ra
```



Beispiel für einen Unterprogrammaufruf

```
.text
subroutine:  subu $sp, $sp, 32
             s.d $f2, 4($sp)
             s.d $f4, 12($sp)
             sw $s0, 20($sp)
             sw $s1, 24($sp)
             sw $ra, 28($sp)
             sw $fp, 32($sp)
             addu $fp, $sp, 32

# Veraenderung von FP2, FP4, $s0, $s1
             l.d $f2, 4($sp)
             l.d $f4, 12($sp)
             lw $s0, 20($sp)
             lw $s1, 24($sp)
             lw $ra, 28($sp)
             lw $fp, 32($sp)
             addu $sp, $sp, 32
             jr $ra
```



Rekursive Unterprogrammaufrufe

□ Problem:

- Rücksprungadresse im Register `$ra` wird bei wiederholtem Unterprogrammaufruf immer wieder überschrieben
- Entsprechendes gilt für lokale Daten in Registern

□ Lösung:

- Sichern von Rücksprungadressen und lokalen Daten auf einem Stack



Rekursive Unterprogrammaufrufe

c-Code:

```
main ()
{
    printf („Die Fakultaet von 10 ist: %d\n“, fakultaet(10));
}

int fakultaet (int n)
{
    if (n<1)
        return (1);
    else
        return (n*fakultaet(n-1));
}
```



Rekursive Unterprogrammaufrufe

MIPS-Code:

```
fakultaet: subu $sp, $sp, 8           # Stack-Frame von 8 Bytes
            sw $ra, 4($sp)           # Rucksprungadresse
            sw $a0, 0($sp)           # Argument(n) sichern

            slt $t0, $a0, 1           # (n < 1)? lade das Argument
            beqz $t0, $zero, L1       # falls n >=1 gilt, gehe zu L1

            add $v0, $zero, 1         # return 1

            lw $ra, 4($sp)           # Rucksprungadresse wiederherstellen
            addu $sp, $sp, 8          # Frame-Stack loeschen
            jr $ra                    # gehe zur Rucksprungadresse
```



Rekursiver Unterprogrammaufruf

```
L1:    sub $a0, $a0, 1        # verwende n-1 als Argument
      jal fakultaet        # rufe mit n-1 auf

      lw $a0, 0($sp)       # hole Argument n wieder vom Stack
      lw $ra, 4($sp)       # hole Ruecksprungadresse vom Stack
      add $sp, $sp, 8      # Stack-Frame loeschen

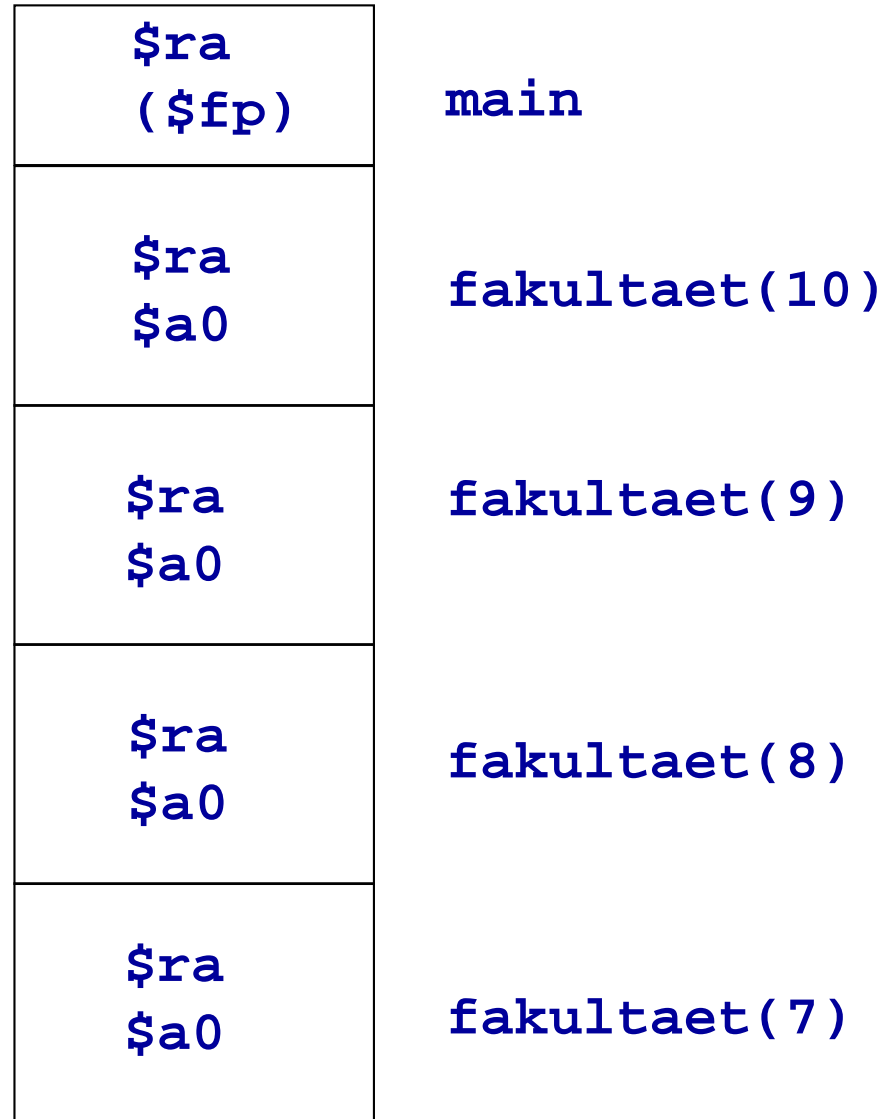
      mul $v0, $a0, $v0    # return n*fak(n-1)

      jr $ra              # Ruecksprung
```



Beispiel 2

Stack



Unterprogrammaufruf

Aufrufendes Programm



Unterprogramm



Ausnahme- und Unterbrechungsbehandlung

Ausnahme:

jede unerwartete Änderung im Kontrollfluss

Beispiel: ungültiger Befehl

Unterbrechung:

Ausnahme, die durch einen äußeren Einfluss verursacht wurde

Beispiel: Drücken einer Taste



MIPS - Ausnahmen

Code	Beschreibung
0	externe Unterbrechung
4	fehlerhafte Adresse beim Laden oder Befehl-Holen
5	fehlerhafte Adresse beim Speichern
6	Busfehler beim Befehl-Holen
7	Busfehler beim Laden oder Speichern
8	Systemaufruf
9	Programmunterbrechung (breakpoint) zur Fehlersuche
10	reservierter Befehl
12	arithmetischer Überlauf bei Ganzzahlberechnungen
14	ungültiges Fließkomma-Ergebnis
15	Division durch Null
16	Überlauf bei Fließkomma-Berechnung
17	Unterlauf bei Fließkomma-Berechnung

} Software



Beispiele für Ausnahmen

Externe Unterbrechung:

- Drücken einer Taste
- Maus wurde bewegt
- Timer ist abgelaufen
- Peripheriegerät (z. B. Drucker) ist fertig
- serielle Schnittstelle hat ein Wort empfangen

Fehlerhafte Adresse:

```
                .data
strng:          .asciiz "arglwuz = "
result:        .space 23

                .text
                .globl main
main:           ...
                lw $a0, result
```



Beispiele für Ausnahmen

Busfehler:

- Time-Out
- Paritätsfehler
- ungültige Port- oder Speicher-Adressen

Programmunterbrechung (breakpoint):

Ausgabe von Register- oder Speicherinhalten oder des Prozessorstatus zur Fehlersuche

Systemaufruf

Software-Interrupt zur Ausführung von Unterprogrammen des Betriebssystems (vergleichbar mit INT n beim 8086)



Beispiele für Ausnahmen

Reservierter Befehl:

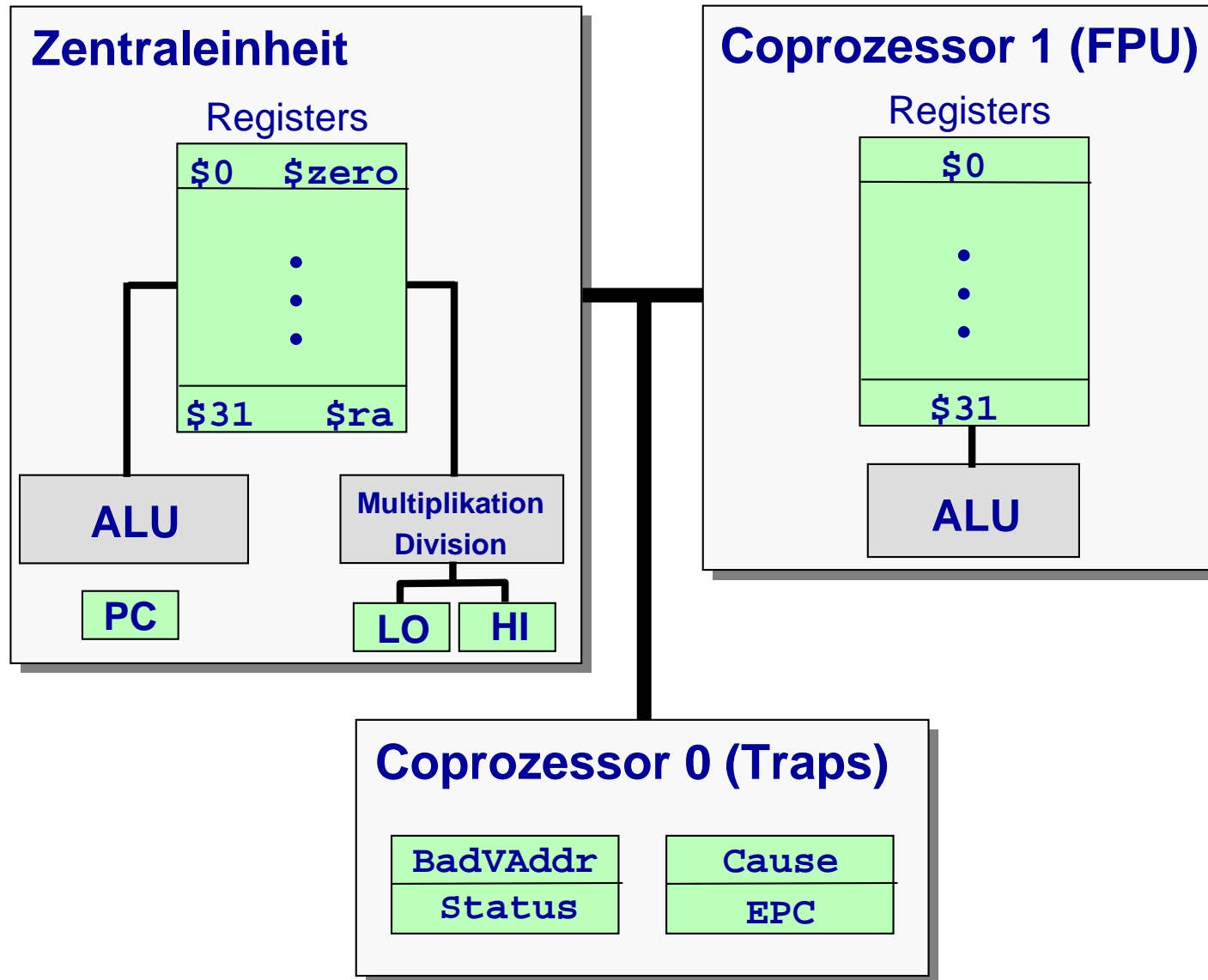
Es wurde versucht, einen Befehl auszuführen, dessen Bits 31...26 nicht als Opcode definiert sind

Arithmetischer Überlauf bei Ganzzahlberechnungen:

Ergebnis einer ADD, ADDI oder SUB Instruktion lässt sich nicht als 32-Bit Zweierkomplement darstellen



Aufbau des MIPS-Prozessors



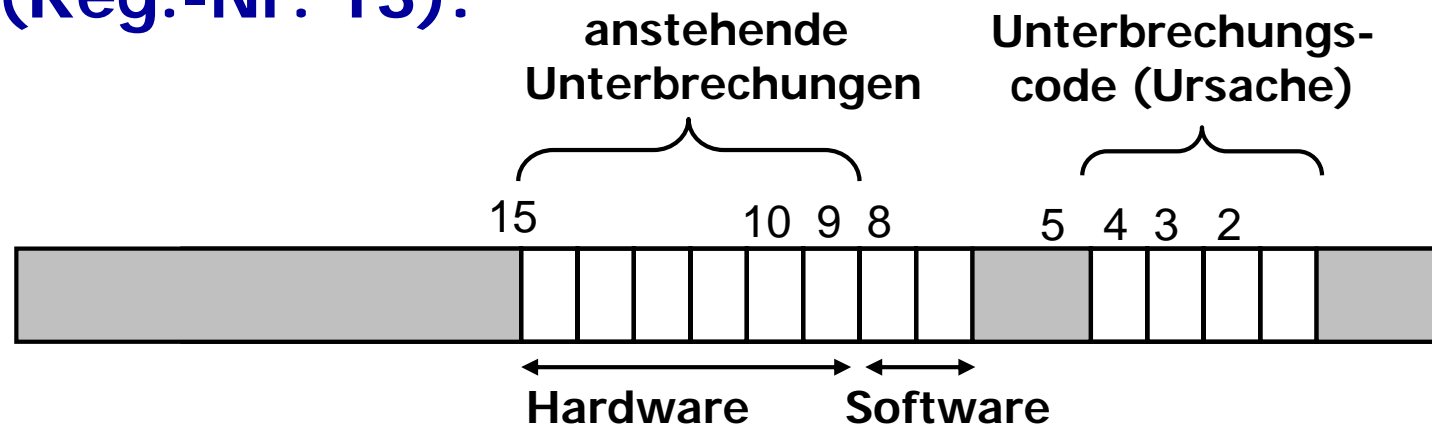
Register des Coprozessors 0

Coprozessor 0 enthält vier Register zur Behandlung von Ausnahmen und Unterbrechungen:

➤ **BadVAddr (Reg.-Nr. 8):**

Falls die Unterbrechung durch einen Speicherzugriff verursacht wurde, ist hier die Speicheradresse enthalten

➤ **Cause (Reg.-Nr. 13):**

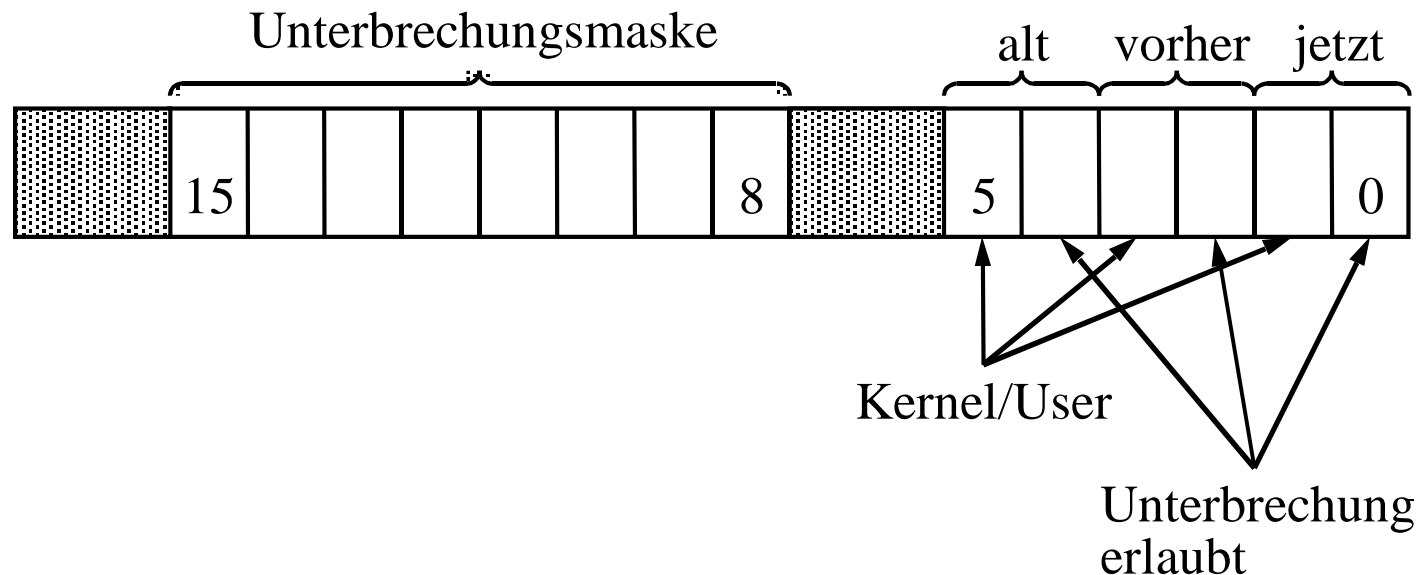


Register des Coprozessors 0

➤ **EPC (Reg.-Nr. 14):**

Speicheradresse, in der sich der Befehl befindet, der zur Unterbrechung geführt hat

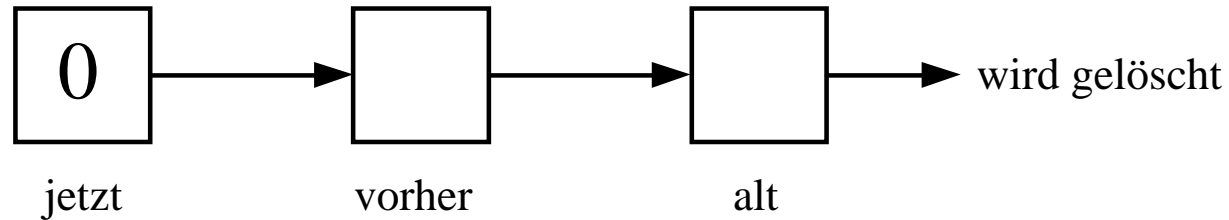
➤ **Status (Reg.-Nr. 12):**



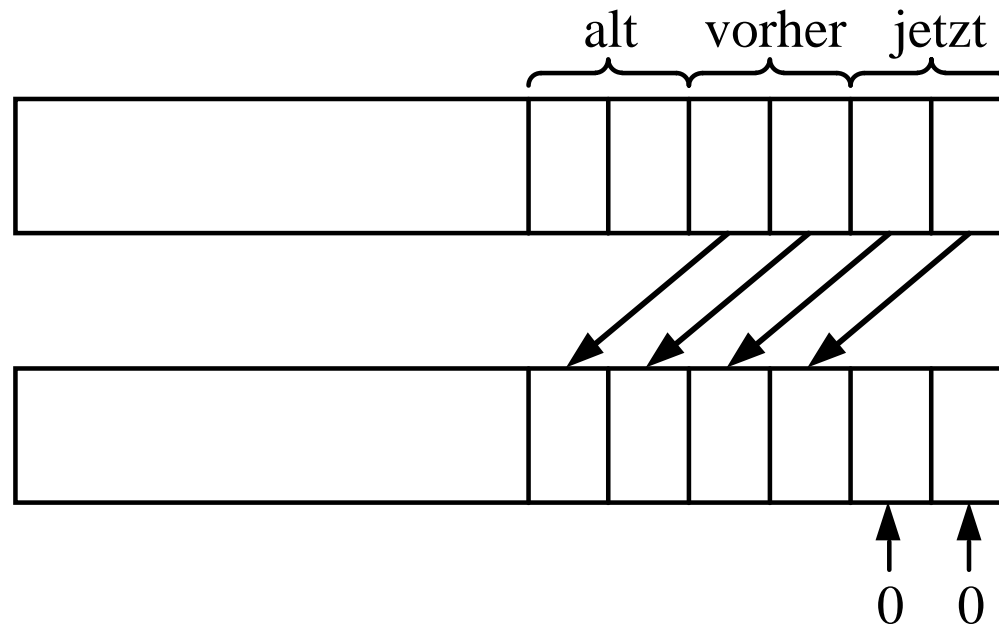
Die Unterbrechungsmaske bestimmt, welche Unterbrechungen zugelassen sind

Register des Coprozessors 0

Speichern der Kernel/User- und Interrupt-Enable-Bits:

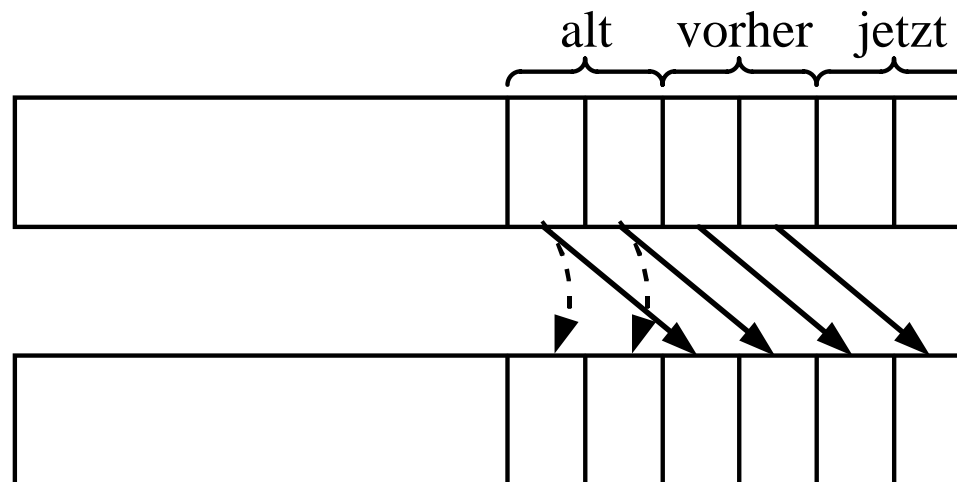


Beim Auftreten einer Unterbrechung:



Register des Coprozessors 0

Rücksprung aus einer Unterbrechung (rfe Instruktion):



Ausnahmebehandlung (Trap Handler)

Bei Ausnahme oder Unterbrechung erfolgt Sprung nach Adresse $8000\ 0080_{16}$

```
.ktext 0x80000080
```

Benutzung des Stapels zur Sicherung von Daten nicht erlaubt, daher Sicherung von Registern im Kerneldatensegment

```
sw $a0, a0_save  
sw $a1, a1_save  
sw $ra, ra_save
```

Register \$at sichern

```
.set noat  
sw $at, at_save  
.set at
```



Ausnahmebehandlung (Trap Handler)

Laden des Cause- und EPC-Registers

```
mfc0 $k0, $13  
mfc0 $k1, $14
```

Unterbrechungen ignorieren

```
bgt $k0, 0x44, fertig
```

Spezifische Maßnahmen

```
... (Benutzung von $a0 und $a1)  
jal print_excp
```



Abschluss der Ausnahmebehandlung

```
fertig:    lw $a0, a0_save
           lw $a1, a1_save
           lw $ra, ra_save
           .set noat
           lw $at, at_save
           .set at
```

Statusregister wiederherstellen

```
rfe
```

Fehlerhafte Instruktion soll nicht nochmals ausgeführt werden

```
addiu $k1, $k1, 4
```



Abschluss der Ausnahmebehandlung

Zurück zum Hauptprogramm (nur falls nicht Ausnahme 6)

```
jr $k1
```

Kerneldatensegment

```
                .kdata  
a0_save:        .word 0  
a1_save:        .word 0  
ra_save:        .word 0  
at_save:        .word 0
```

