MARS 2

Wiederholung

- Programm besteht aus Operatoren und Operanden
- C/JAVA: Operatoren = { if(), else, +, -, while, ...}

  Operanden = Variablen und Zahlen
- Assembler: Operatoren = Befehle = {add, lw, sw, beq,…}

  Operanden = Register = {$t1, $ra, $s3,…}

Register (Operanden)

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

Befehle (Operatoren)

Kategorie	Befehl	Beispiel	Bedeutung
Arithmetische Befehle	add	`add $s1,$s2,$s3`	`$s1 = $s2+$s3`
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2-$s3`
	add immediate	`addi $s1,$s2,100`	`$s1 = $s2+100`
Datentransport	load word	`lw $s1,100($s2)`	`$s1 = Mem[$s2+100]`
	store word	`sw $s1,100($s2)`	`Mem[$s2+100] = $s1`
Verzweigung	branch on equal	beq $s1,$s2,25	if ($s1==$s2) PC = PC + 4 + 100
	branch on not equal	bne $s1,$s2,25	if ($s1!=$s2) PC = PC + 4 + 100

```
int count_negatives(int table[], int n)
{
    int count = 0;
    int i;

    for (i=0; i < n; i++) {
        if (table[i] < 0) {
            count++;
        }
    }

    return count;
}
```

Pseudobefehle

- Befehle, die aus mehreren Befehlen bestehen
  - blt, bge, li
- Warum?
  - Entwurfsprinzipien
  - Befehlssatz begrenzt halten

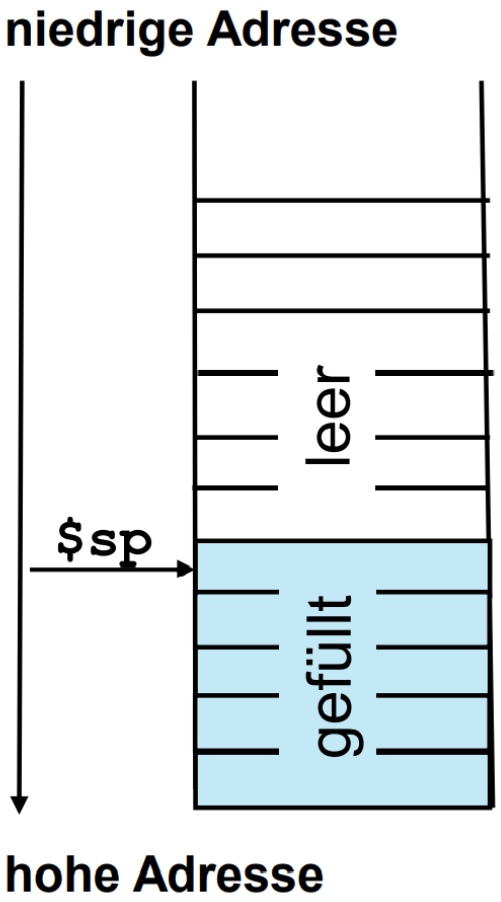
4 Entwurfsprinzipien

- Simplicity favors regularity
  (Einfachheit begünstigt Regelmäßigkeit)
- Smaller is faster
  (Kleiner ist schneller)
- Make the common case fast
  (Optimiere den häufig vorkommenden Fall)
- Good design demands compromises
  (Ein guter Entwurf fordert Kompromisse)
  - oder: Sei nicht dogmatisch

Multiplikation

- Das Multiplizieren von zwei 32-Bit Zahlen kann ein 64-Bit Produkt ergeben.
  - Neue Register: Hi und Lo
    - Hi beinhaltet die 32 höchstwertigsten Bits des 64-Bit Produkts.
    - Lo beinhaltet die 32 niederwertigsten Bits.
  - MIPS-Befehle:
    - mult $s2,$s3 # Hi#Lo = $s2x$s3
    - Move from lo: mflo $s1 # $s1 = Lo
    - Move from hi: mfhi $s1 # $s1 = Hi
- Pseudo-Instruktion: mul $s1,$s2,$s3
- Reale Umsetzung: mult $s2,$s3
- mflo $s1

Stack



Ablegen von `$s0 und $s1`:

```
addi  $sp,$sp,-8
sw    $s0,4($sp)
sw    $s1,0($sp)
```