```
FUN:    sw $ra,-4($sp)
        sw $s0,-8($sp) # $s0 für C
        sw $s1,-12($sp) # $s1 für j
        sw $s2,-16($sp) # $s2 für i
        sw $s3,-20($sp) # $s3 für m
        addi $sp,$sp,-20
        move $s0,$a0
        move $s1,$a1
        move $s2,$a3
        addi $s3,$a2,9
        move $a0,$s2
        move $a1,$s1
        jal V
        add $s3,$s3,$v0
        move $a0,$s1
        move $a1,$s2
        jal X
        add $s3,$s3,$v0
        sll $t0,$s3,2
        add $t0,$t0,$s0
        lw $v0,0($t0)
        addi $sp,$sp,20
        lw $ra,-4($sp)
        lw $s0,-8($sp) # $s0 für C
        lw $s1,-12($sp) # $s1 für j
        lw $s2,-16($sp) # $s2 für i
        lw $s3,-20($sp) # $s3 für m
        jr $ra
```

Rechnerarchitektur (AIN 2)

SoSe 2021

Kapitel 2

Befehle: Die Sprache des Rechners

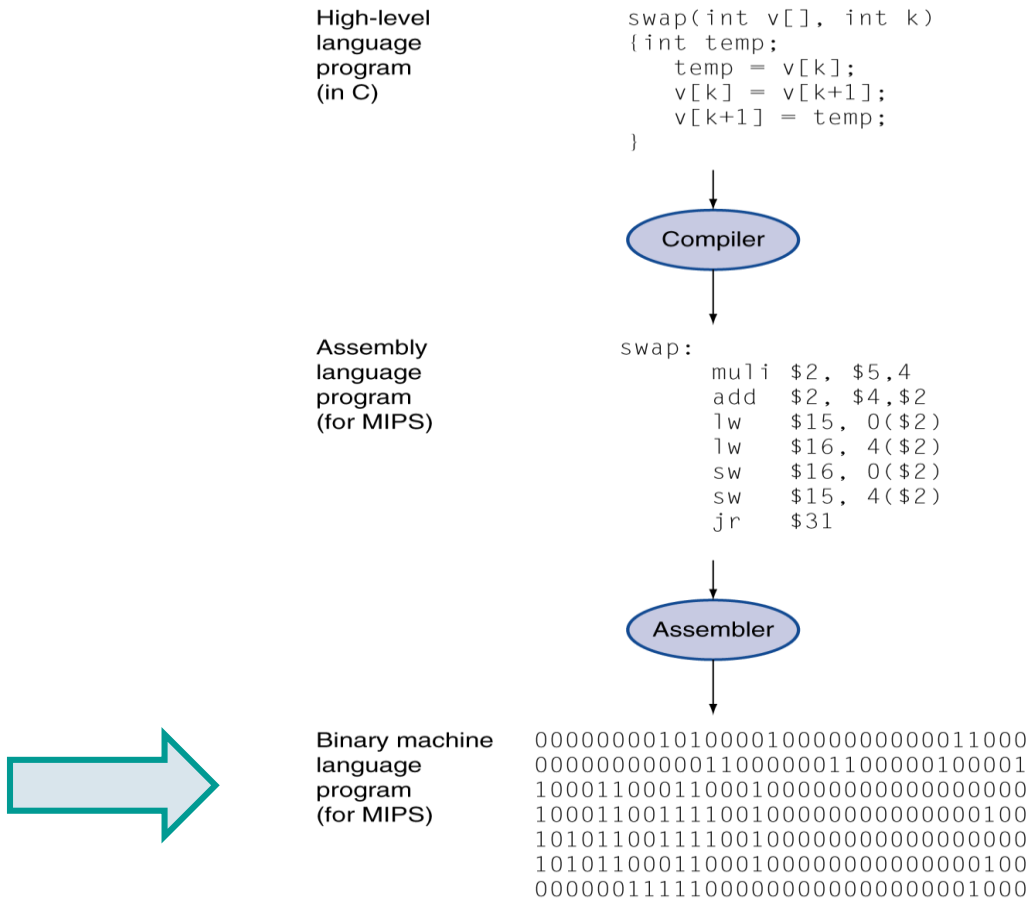
Prof. Dr.-Ing. Michael Blaich
mblaich@htwg-konstanz.de

Vorlesungsinhalt

Programmcode auf verschiedenen Ebenen

Compiler und Assembler bieten Abstraktionsebenen, die die Programmierung erleichtern

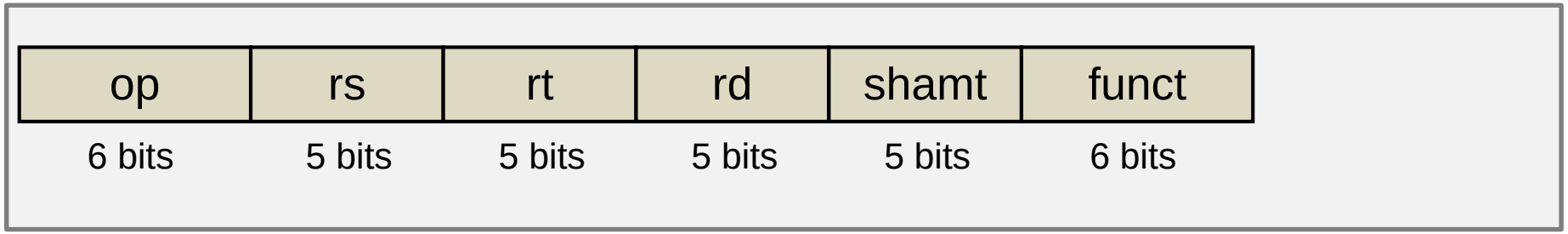
- Hochsprache
  - Komfortable Programmierung
- Assemblersprache
  - Lesbare Maschinensprachebefehle
- Maschinensprache
  - Binär kodierte Befehle und Daten



Instruktionsformat für Addition

Instruktionsformat nennt sich R-Format oder Register-Format

- Instruktionslänge in 32 Bit



Felder

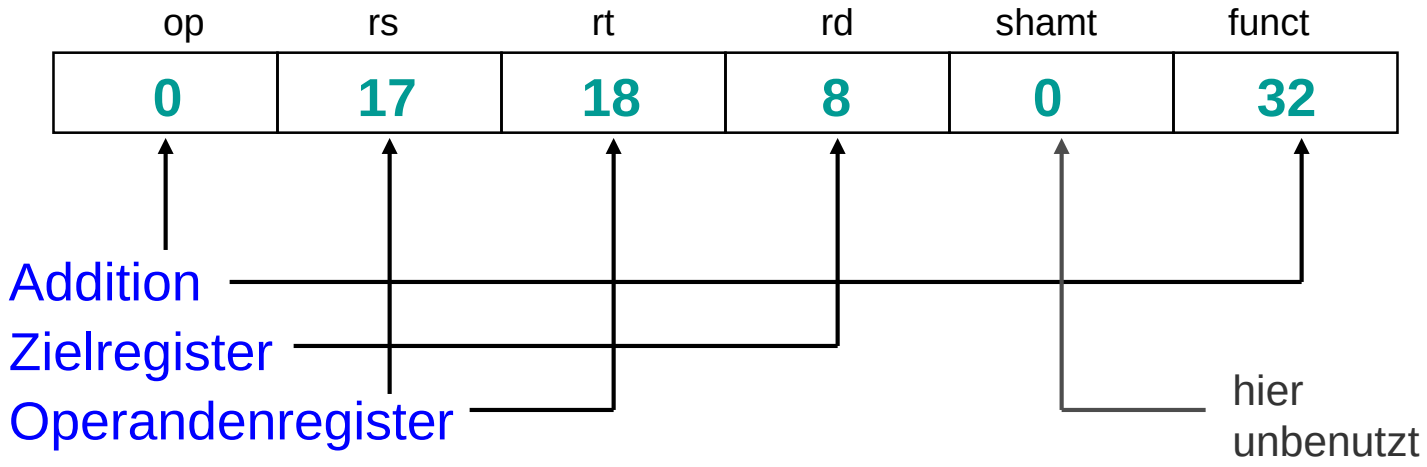
- op: Basisoperation (operation code, opcode)
- rs: Register des ersten Quelloperanden (register source)
- rt: Register des zweiten Quelloperanden (register target)
- rd: Zielregister (register destination)
- shamt: shift amount (00000 for now) → näheres zu shift Befehlen in 2.4
- funct: Funktionscode (function code) spezielle Variante der Basisoperation

Beispiel: Repräsentation eines Additionsbefehls

MIPS-Assembler

```
add $t0, $s1, $s2
```

Dezimaldarstellung



Binärdarstellung

000000	10001	10010	01000	00000	100000
6 Bit	5 Bit	5 Bit	5 Bit	5 Bit	6 Bit

Name	Nr
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23

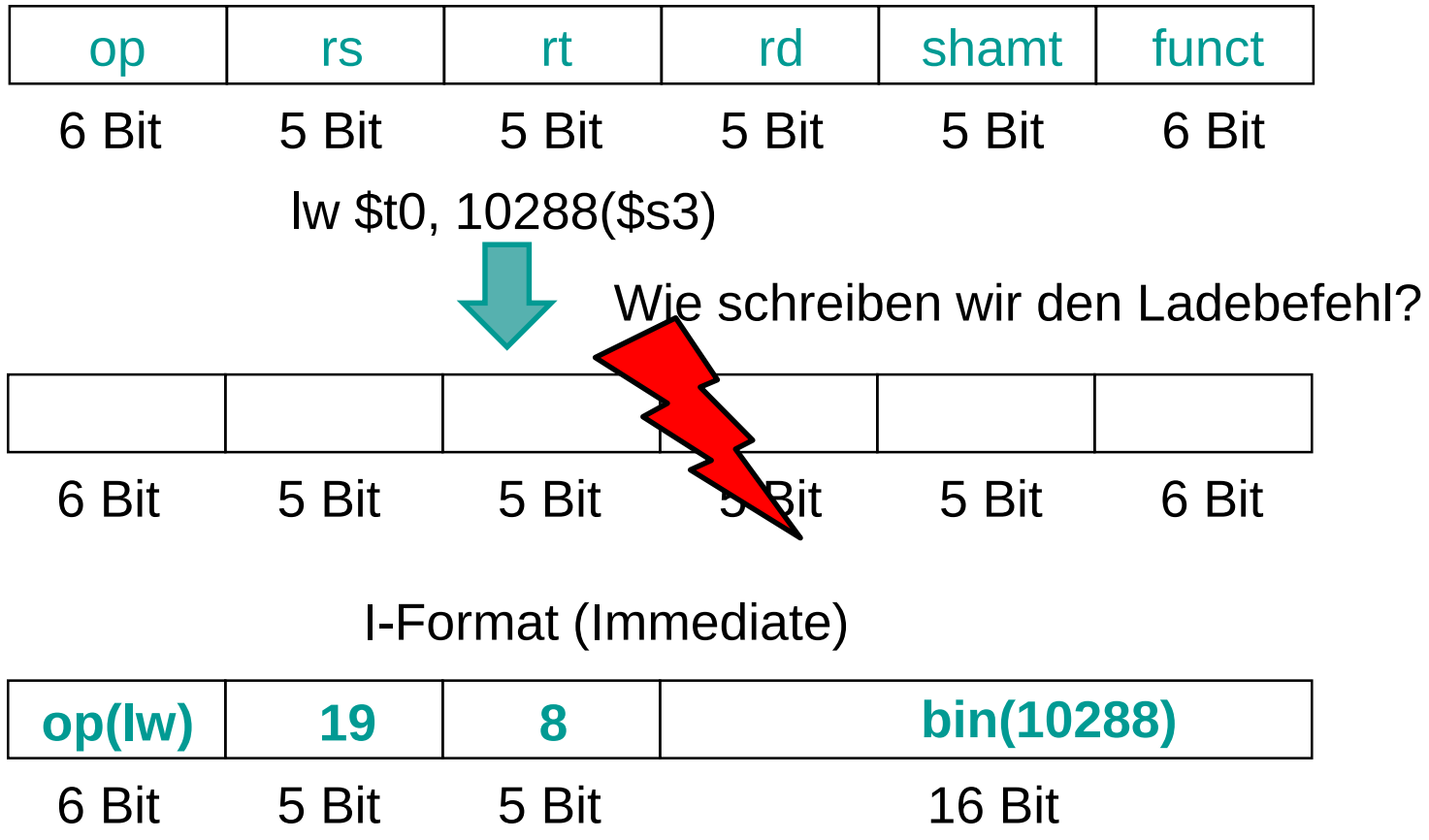
Name	Nr
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

Vorlesungsinhalt

MIPS Instruktionsformate

„R“-format passt nicht für alle Instruktionen

- Unterschiedliche Instruktionsformate benötigt.

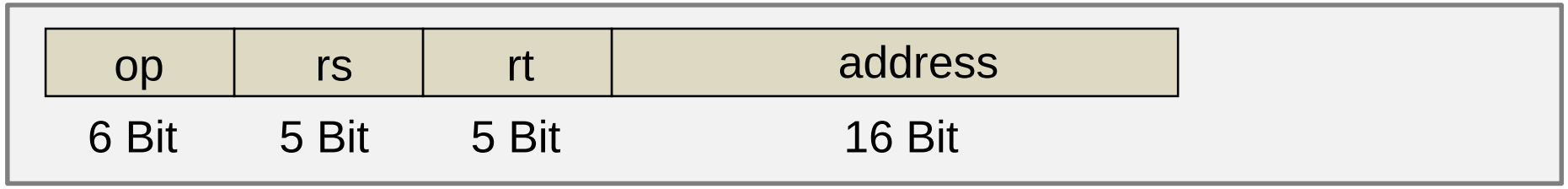


Name	Nr
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23

Name	Nr
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

MIPS Instruktionsformat „I“ (Immediate)

Instruktionsformat „I“ (für immediate = direkt) oder I-Format wird für Datentransferbefehle und für Addieren von Konstanten verwendet



Felder

- op: Basisoperation (operation code, opcode)
- rs: Register des ersten Quelloperanden / der Basisadresse
- rt: Zielregister
- address: Konstante oder Adressoffset

Fragen:
1. Was ist die größtmögliche Konstante (addi)?
2. Was ist der größtmögliche Adressoffset (lw)?

Übersicht der bisherigen Instruktionen

Feststellung: alle Formate haben die gleiche Länge: 32 Bit

Instruktion	Format	op	rs	rt	rd	shamt	funct
`add`	R	0	reg	reg	reg	0	32
`sub`	R	0	reg	reg	reg	0	34
`addi` *(immediate)*	I	8	reg	reg	constant		
`lw` *(load word)*	I	35	reg	reg	offset		
`sw` *(store word)*	I	43	reg	reg	offset		
		6 Bit	5 Bit	5 Bit	5 Bit	5 Bit	6 Bit

16 Bit

Beispiel

$t1: Basisadresse von A, $s2: Wert von h

Was ist der Assembler-Code?

A[300]=h+A[300]

Maschinen-Code (dezimal):

op	rs	rt	rd/shamt/funct or address		
35	9	8	1200		
0	8	18	8	0	32
43	9	8	1200		

Format der Instruktionen:

Instruktion	Format	op	rs	rt	rd/shamt/funct or address		
add	R	0	reg	reg	reg	0	32
lw	I	35	reg	reg	offset		
sw	I	43	reg	reg	offset		

Name	Nr
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23

Name	Nr
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

Vorlesungsinhalt

Kapitel 1: Grundlegende Ideen, Technologien, Komponenten

Kapitel 2: Befehle: Die Sprache des Rechners

- 2.1 Befehlssatz: Was ist das?
- 2.2 Befehle des MIPS Befehlssatzes
- 2.3 Darstellungen von Befehlen im Rechner
- **2.4 Logische Operationen**
- 2.5 Kontrollstrukturen
- 2.6 MIPS Assembler und MARS Simulator
- 2.7 Weitere MIPS-Befehle
- 2.8 Compiler, Assembler, Linker, Loader
- 2.9 Andere Befehlssätze
- 2.10 Zusammenfassung

Logische Operationen (Bit Arithmetik)

Bit-Arithmetik: rechnen mit einzelnen Bits und nicht mit der ganzen Zahl

- Logische Operatoren
  - AND
  - OR
  - NOR
  - XOR
  - NOT

```
AND (0110 1001,0100 0010)

        0110 1001
AND     0100 0010
        0100 0000
```

```
OR (0110 1001,0100 0010)

        0110 1001
OR      0100 0010
        0110 1011
```

```
NOR (0110 1001,0100 0010)

        0110 1001
NOR     0100 0010
        1001 0100
```

```
XOR (0110 1001,0100 0010)

        0110 1001
XOR     0100 0010
        0010 1011
```

```
NOT (0100 0010)

NOT     0100 0010
        1011 1101
```

MIPS: AND, OR, NOR, XOR

Instruktionen in MIPS

Instruktion (R-Format)	Kommentar
`and $t0,$t1,$t2`	# $t0=$t1 AND $t2
`or  $t0,$t1,$t2`	# $t0=$t1 OR $t2
`nor $t0,$t1,$t2`	# $t0=$t1 NOR $t2
`xor $t0,$t1,$t2`	# $t0=$t1 XOR $t2

Instruktion (I-Format)	Kommentar
`andi $t0,$t1,0111`	# $t0=$t1 AND 0111
`ori  $t0,$t1,1100`	# $t0=$t1 OR 1100
`xori $t0,$t1,1100`	# $t0=$t1 XOR 1100

Häufiger Gebrauch:

- **OR** Bit in ein Wort einfügen
  (z.B: Byte an letzte Stelle kopieren)
- **AND** Maskieren von Bits
  (Bits auswählen, andere auf „0“)

- Beispiel „Bits Maskieren mit AND“

$t2	0000 0000 0000 0000 0000 1101 1100 0000
$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	0000 0000 0000 0000 0000 1100 0000 0000

NOT

Es gibt kein NOT in MIPS!!!

- NOT kann als NOR realisiert werden
- Schlanker Befehlssatz: keine unnötigen Befehle

```
NOT  (0100 0010)

NOT     0100 0010
        1011 1101
```

```
NOR  (0100 0010,0000 0000)
        0000 0000
NOR     0100 0010
        1011 1101
```

- als logische Funktion

```
NOT(a)=NOR(a,0)
```

- als MIPS Instruktion

Instruktion (R-Format)	Kommentar
`nor $t0,$t1,$zero`	# $t0=NOT($t1)

Logischer und arithmetischer Shift

Shift: Verschiebung der Bits um „n“ Stellen

- Arithmetischer Rechts-Shift: Einfügen von „1“ bei negativen Zahlen

- Logischer Shift

```
RIGHT   (1110 1001,4)
        0000 1110

RIGHT   (0010 1001,4)
        0000 0010

LEFT    (1110 1001,4)
        1001 0000
```

- Arithmetischer Shift

```
RIGHT   (1110 1001,4)
        1111 1110

RIGHT   (0010 1001,4)
        0000 0010

LEFT    (1110 1001,4)
        1001 0000
```

Im Befehlssatz nicht benötigt. Einfügen von “niedrigwertigen Einsen von rechts” macht keinen Sinn.

MIPS: Shift

Instruktionen in MIPS

Instruktion	Kommentar
`sll  $t0,$t1,4`	# $t0=$t1 << 4
`srl  $t0,$t1,4`	# $t0=$t1 >> 4
`sllv $t0,$t1,$t2`	# $t0=$t1 << $t2
`srlv $t0,$t1,$t2`	# $t0=$t1 >> $t2
`sra  $t0,$t1,4`	# $t0=$t1 >> 4 (arithmetisch)
`srav  $t0,$t1,$t2`	# $t0=$t1 >> $t2 (arithmetisch)

Shifts können teure Multiplikationen ersetzen

- sll $t0,$t1,4: $t0=$t1*16
- sll (shift left logical)
- sllv (shift left logical variable)

Maschinen-Code (dezimal):

	op	rs	rt	rd	shamt	funct
allgemein	0	x	reg	reg	bits	Funktion
`sll  $t0,$t1,4`	0	x	9	8	4	0
`srav  $t0,$t1,$t2`	0	10	9	8	x	7

Name	Nr
$t0	8
$t1	9
$t2	10

Übersicht der Shift-Operationen

- Bit-weise Operationen haben alle opcode „0“
- Unterscheidung durch Funktionsidentifikatoren 0-7

Befehl	B_{31-26}	B_{25-21}	B_{20-16}	B_{15-11}	B_{10-6}	B_{5-0}
	opcode	register s	register t	register d	shift amount	function
sll $rd, $rs, c	00000	unused	-	-	c	000 000
sllv $rd, $rt, $rs	00000	-	-	-	00000	000 100
srl $rd, $rs, c	00000	unused	-	-	c	000 010
srlv $rd, $rt, $rs	00000	-	-	-	00000	000 110
sra $rd, $rs, c	00000	unused	-	-	c	000 011
srav $rd, $rt, $rs	00000	-	-	-	00000	000 111

Übersicht der bit-weisen Operationen

- Einfügen/Extrahieren von Bytes in Worte mit Shift und AND/OR
- Multiplikationen können durch bit-weise Operationen ersetzt werden

	Instruktion	Bedeutung
Shift	**sll** *rd, rs, shamt*	Register rd = Register rs logisch links um den Wert shamt geshiftet.
	sllv *rd, rt, rs*	Register rd = Register rs logisch links um den in Register rs gespeicherten Wert geshiftet.
	srl *rd, rs, shamt*	Register rd = Register rs logisch rechts um den Wert shamt geshiftet.
	srlv *rd, rt, rs*	Register rd = Register rs logisch rechts um den in Register rs gespeicherten Wert geshiftet.
	sra *rd, rs, shamt*	Register rd = Register rs arithmetisch rechts um den Wert shamt geshiftet.
	srav *rd, rt, rs*	Register rd = Register rs arithmetisch rechts um den in Register rs gespeicherten Wert geshiftet.
Logische Verknüpfung	**and** *rd, rs, rt*	Register rd = Register rs AND Register rt.
	or *rd, rs, rt*	Register rd = Register rs AND Register rt.
	nor *rd, rs, rt*	Register rd = Register rs AND Register rt.
	xor *rd, rs, rt*	Register rd = Register rsAND Register rt.
	andi *rt, rs, imm*	Register rt = Register rs AND Konstante imm
	ori *rt, rs, imm*	Register rt = Register rs AND Konstante imm
	xori *rt, rs, imm*	Register rt = Register rs AND Konstante imm

Verständnisaufgabe 1

MIPS Assemblercode für die folgende Funktion:

- $s1=erstes (niedrigwertigstes) Byte von (4*NOT($s1 AND $s2)

Benötigte Befehle: and, nor, sll

Quiz

```
ori     $s0, $zero, 4
lw      $t1, 4($zero)
lw      $t2, -4($t1)
sub     $t3, $s0, $t1
sra     $t2, $t2, 3
sllv    $t3, $t3, $t2
and     $t4, $t1, $t3
or      $t5, $t4, $t3
sw      $t5, 4($t5)
sb      $t5, 9($zero)
```

Was passiert?

Hauptspeicher zu Beginn:

Adresse	Inhalt (Word)
	⋮
12	33
8	6
4	8
0	97563

Hauptspeicher am Ende:

Adresse	Inhalt (Word)
	⋮
12	
8	
4	
0	