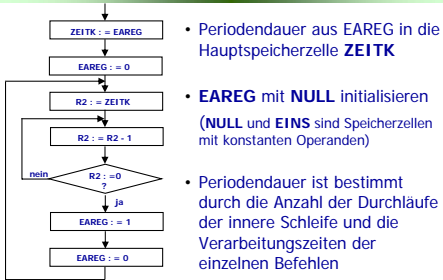


## 2. Übung

### Einführung in die Assemblerprogrammierung

#### MIPS-Assembler

#### Programmablauf in Form eines Flussdiagramms



- Periodendauer aus EAREG in die Hauptspeicherzelle **ZEITK**
- **EAREG** mit **NULL** initialisieren (**NULL** und **EINS** sind Speicherzellen mit konstanten Operanden)
- Periodendauer ist bestimmt durch die Anzahl der Durchläufe der inneren Schleife und die Verarbeitungszeiten der einzelnen Befehle

### Berechnung von ZEITK

Impulsabstand: 0,005 s      Taktzeit: 0,04 µs (25 MHz)

### Maschinencode-Darstellung

Adresse	Maschinencode		Symbol
	Dec. Dual	Hex.	
0	000000	000000	0100 MOVE
1	000001	0000000000000000	8040 EAREG
2	000002	0000000000010111	0017 ZEITK
3	000003	0000000000000000	0100 MOVE
4	000004	0000000000000000	0100 NULL
5	000005	1000000000000000	8000 EAREG
6	000006	0000000000000000	0100 MOVE
7	000007	0000000000010000	0018 NULL
8	000008	0000000000000000	0100 MOVE
9	000009	0000000000000111	0017 ZEITK
10	000010	0000000000000000	020A SUB
11	000011	0000000000000001	0019 EINS
12	000012	0010000100000100	218A CMP
13	000013	0001100000000000	4C30 BNE
14	000014	0000000000000000	000A M2
15	000015	0000000000000000	0100 MOVE
16	000016	0000000000010001	0019 EINS
17	000017	1000000000000000	8000 EAREG
18	000018	0000000000000000	0100 MOVE
19	000019	0000000000010000	0018 NULL
20	000020	1000000000000000	8000 EAREG
21	000021	0000000000000000	0000 JMP
22	000022	0000000000000000	0008 M1
23	000023	XXXXXXXXXXXX	
24	000100	0000000000000000	0000
25	000101	0000000000000001	0001

Symbol	Binärkode
MOVE	0000 0001
SUB	0000 0010
CMP	0010 0001
BNE	0000 1100
JMP	0000 0101

## Einführung in die Assemblerprogrammierung

### Programm-Darstellung

- Symbolische Darstellung
- Maschinencode-Darstellung

### Programm-Übersetzung

- Assemblersprache
- Assembleranweisungen
- Assemblierung

#### Notwendige Befehle zur Lösung

##### MOVE QADR, ZADR

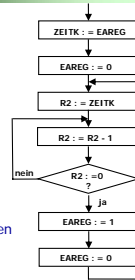
Inhalt von QADR (Quelladresse) nach ZADR (Zieladresse)

##### SUB QADR, ZADR

Subtrahiere den Inhalt von QADR vom Inhalt von ZADR und schreibe das Ergebnis in ZADR

##### CMP ADR1, ADR2

Vergleiche die mit ADR1 und ADR2 adressierten Operanden. Ergebnis in den CC-Bits des Prozessor-Statusregisters



### Maschinencode-Darstellung

#### Symbolische Angaben durch ihre äquivalente binäre Darstellungen ersetzen:

- Zuordnung der symbolischen Operationscodes zu binären Operationscodes liegt durch eine **Zuordnungstabelle** fest:

Symbol	Binärkode
MOVE	0000 0001
SUB	0000 0010
CMP	0010 0001
BNE	0000 1100
JMP	0000 0101

### Programm-Übersetzung (Assemblierung)

Übersetzung kann „per Hand“ erfolgen. Dazu benötigt man:

- die Zuordnungstabelle für die OpCodes und
- die Adresse des ersten Befehls im Speicher, um die Symboltabelle mit den Adresszuordnungen aufstellen zu können.

**Nachteil:** sehr zeitaufwendig und fehleranfällig

Übersetzungsvorgang läuft jedoch nach festen Regeln ab:

➔ **Ausführung durch den Prozessor selbst**

## Programmieraufgabe

### Aufgabenstellung

Es soll ein Impulsgeber mit Hilfe eines µP-Systems aufgebaut werden, der in konstanten Zeitabständen Impulse an eine Ein-/Ausgabeeinheit abgibt

- Festlegen eines Satzes von Maschinenbefehlen zur Lösung dieser Aufgabe
- Erstellung des Programms in der Assemblersprache und in der Maschinensprache

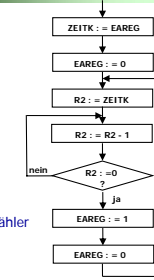
#### Notwendige Befehle zur Lösung

##### BNE SPRADR

**Bedingter Sprung:** lade den Befehlszähler mit der Sprungadresse SPRADR, sofern das Prozessor-Statusregister den Zustand "ungleich" zeigt, sonst wird der nächste Befehl im Programm ausgeführt

##### JMP SPRADR

**Unbedingter Sprung:** lade den Befehlszähler mit der Sprungadresse SPRADR



### Maschinencode-Darstellung

- Die Ersetzung symbolischer Adressen durch numerische Adressen ergibt sich aus der Lage des Programms im Hauptspeicher
- Numerische Adressen von Registern liegen fest.
- Adresse von EAREG ist durch die Adressdecodierung der EA-Einheit vorgegeben (\$8000)

#### Voraussetzung:

Programm belegt den Hauptspeicher ab der Zelle 0

➔ Adresszuordnung als Symboltabelle

### Programm-Übersetzung (Assemblierung)

#### Assembler:

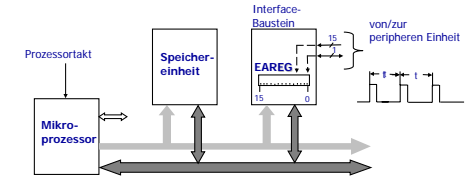
Programm, das einen Quellcode in Assemblersprache in eindeutiger Weise in Maschinencode übersetzt.

Die Regeln zur symbolischen Programmierung ergeben sich aus der Definition einer Assemblersprache

#### Format einer Programmzeile:

Namensfeld (label)	Operationsfeld (OpCode)	Adressfeld (Operanden)	Kommentarfeld
symbolische Adressierung einer Programmzeile	symbolischer OpCode	symbolische Adreßangaben	

## Programmieraufgabe



EA-Einheit mit einem 16 Bit Register **EAREG**  
 Absolute Adresse von EAREG: **\$8000 (32768)**  
 Periodendauer der Impulsfolge: **t**

#### Programm in symbolischer Darstellung

MOVE	EAREG, ZEITK	6
MOVE	NULL, EAREG	6
MOVE	NULL, R0	4
MARKE1	MOVE ZEITK, R2	4
MARKE2	SUB EINS, R2	5
CMP	R0, R2	4
BNE	MARKE2	3
MOVE	EINS, EAREG	6
MOVE	NULL, EAREG	6
JMP	MARKE1	3
ZEITK		
NULL	0	
EINS	1	

Takte/Befehl

### Maschinencode-Darstellung

#### Symboltabelle für unser Programm:

Symbol	Adresse		
	Dual	Dezimal	Hex.
MARKE1	0000 0000 0000 1000	8	0008
MARKE2	0000 0000 0001 1010	10	000A
ZEITK	0000 0000 0001 0111	23	0017
NULL	0000 0000 0001 1000	24	0018
EINS	0000 0000 0001 1001	25	0019
EAREG	1000 0000 0000 0000	32768	8000

MOVE	MARKE1	ZEITK
MOVE	NULL	EAREG
MOVE	NULL	R0
MOVE	ZEITK	R2
SUB	EINS	R2
CMP	R0	R2
BNE	MARKE2	
MOVE	EINS	EAREG
MOVE	NULL	EAREG
JMP	MARKE1	
ZEITK		
NULL	0	
EINS	1	

### Assemblerdirektiven (Assembleranweisungen)

Assemblerdirektiven sind Befehle für den Assembler

- Erzeugung von Konstanten
- Wertzuweisung an Operanden
- Reservierung von Speicherplatz
- Steuerung des Assemblierungsvorgangs
- erzeugen bei der Übersetzung nicht immer Binärkode (im Gegensatz zu Maschinenbefehlen)
- sie unterliegen dem Format einer Programmzeile

## Assemblerdirektiven

[symbol]	ORG	c	origin of program or data
[symbol]	DS	c	define storage
[symbol]	DC	c	define constant
symbol	EQU	c	equate
	END		end of program

### Im Beispiel:

```

ORG 0
EAREG EQU 32678
NULL DC 0
EINS DC 1
ZEITK DS 1
    
```

## Impulsgeber-Programm

Name	Opcode	Adrefangaben	Kommentar
*			Impulsgeberprogramm
EAREG	ORG	0	Speicherbelegung ab Adresse 0
	EQU	32768	Ein-/Ausgabeadresse festlegen
	MOVE	EAREG, ZEITK	Zeitkonstante einlesen
	MOVE	NULL, EAREG	
	MOVE	NULL, R0	
M1	MOVE	ZEITK, R2	Zeitschleife initialisieren
M2	SUB	EINS, R2	
	CMP	R0, R2	Zeitbedingung abfragen
	BNE	M2	
	MOVE	EINS, EAREG	positive Flanke
	MOVE	NULL, EAREG	negative Flanke
	JMP	M1	
*			Beginn des Datenbereichs
ZEITK	DS	1	
NULL	DC	0	
EINS	DC	1	
	END		

Nr.	Adresse	Inhalt	Name	Opcode	Adrefangaben	Kommentar
1	0	*				Impulsgeberprogramm
2	0					
3	0					
4	0	ORG 0				speichern ab Adr. 0
5	0	EQU 32768				E/A-Adr. festlegen
6	0000	EAREG				MOVE EAREG, ZEITK Zeitkonst. einlesen.
7	0003	0100				MOVE NULL, EAREG
8	0006	0108				MOVE NULL, R0
9	0008	010A	M1			MOVE ZEITK, R2 Zeitschleife init.
10	000A	020A	M2			SUB EINS, R2
11	000C	210A				CMP R0, R2 Zeitbed. abfragen
12	000D	0C00				BNE M2
13	000F	0100				MOVE EINS, EAREG positive Flanke
14	0012	0100				MOVE NULL, EAREG negative Flanke
15	0015	0500				JMP M1
16	0018	0000				
17	0017	ZEITK	DS	1		
18	0018	0000	NULL	DC	0	
19	0019	0001	EINS	DC	1	
20		END				

## Assemblierung

### Zwei Phasen:

- Phase:**
  - Adresszuordnung herstellen
  - Fehlerliste anfertigen
- Phase:**
  - Maschinencode erzeugen
  - Symbolische und binäre Auflistung

Beim Erreichen der END-Anweisung müssen alle Symbole definiert sein!

## MIPS-Assembler

### Assemblerprogrammierung mit dem MIPS-Simulator SPIM

- Der SPIM-Simulator (MIPS R2000/R3000-Prozessor)
  - Literatur: Hennessy & Patterson (Anhang A auf der TI-Homepage)
- Programmiermodell des SPIM-Simulators
  - Aufbau eines MIPS-Prozessors
  - Registersatz
  - Speicheraufteilung
- MIPS-Assemblerprogrammierung
  - Syntax der MIPS-Assemblersprache
  - Assemblerdirektiven
  - Adressierungsarten
  - Datenformate
  - Befehlsformate & Befehlsatz

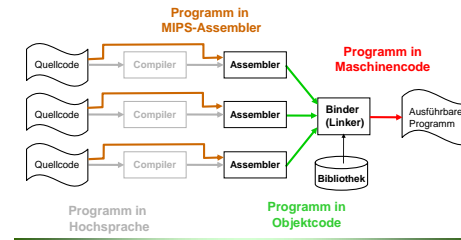
## Warum MIPS?

MIPS (*machine with no interlocked pipe stages*)  
 ≠  
 MIPS (*million instructions per second*)

- Der MIPS R2000 ist ein sehr klar strukturierter RISC Prozessor
- Sauberer und klarer Befehlsatz
- Basis der richtungsweisenden Bücher von Hennessy/Patterson
- MIPS außerhalb von PCs (bei Druckern, Routern, ..) weit verbreitet
- SPIM-Simulator verfügbar
  - Keine Gefährdung des laufenden Systems
  - Bessere Interaktionsmöglichkeiten
  - SPIM läuft auf PCs, und Workstations (Linux/Sun-OS, MacOS, Windows)

## Der SPIM-Simulator

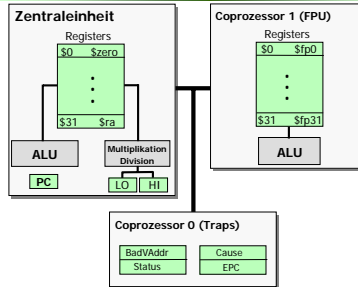
Der SPIM-Simulator ist Assembler, Linker und Debugger in einem Programm.



## Installation und Benutzung

- Simulator abrufbar unter:
  - <http://i16www.ira.uka.de/users/asfour/TI/TI-2/Spim>
- SPIM unterliegt nur der GNU-Lizenz. Die neueste Version ist erhältlich unter:
  - <ftp://ftp.cs.wisc.edu/pub/spim>
- Unix-Versionen: spim.tar.gz, spim.tar.z
- Macintosh-Version: SPIM.sit.bin und SPIM.Hqx.txt
- Windows-Version: spim.zip

## Aufbau des MIPS-Prozessors



## Koprozessoren

Der MIPS-Prozessor besitzt zwei Koprozessoren

- Coprozessor 0 (Traps):** Register enthalten Informationen über den Prozessorstatus und die Ursachen von Unterbrechungen und Ausnahmen
- Coprozessor 1 (FPU) für Gleitkomma-Arithmetik:** Enthält 32 allgemein verwendbare 32-Bit Gleitkomma-Register

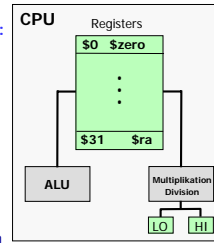
## Registersatz

MIPS ist eine **Lade/Speicher-Architektur:**

- Speicherzugriffe nur über Lade- und Speicher-Befehle.
- Berechnungen erfolgen nur auf Registern

Der MIPS-Prozessor ist eine typische **Register-Register-Maschine** mit 32 allgemein verwendbare Register.

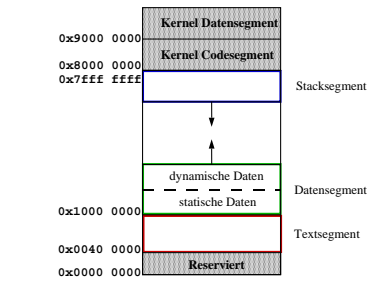
Sie sind durch ein vorangestelltes \$-Zeichen gekennzeichnet und deren Verwendung ist zum Teil durch Konvention festgelegt.



## Registersatz

Name	Nr.	Verwendung
\$k0 – \$k1	\$26 – \$27	Reserviert für das Betriebssystem. Dürfen vom Programmierer <b>nicht</b> verwendet werden.
\$gp	\$28	Beinhaltet einen Zeiger auf die Mitte im statischen Datensegment und wird zum schnellen Laden und Speichern globaler Daten verwendet
\$sp	\$29	Stack-Zeiger: verweist auf die erste freie Speicheradresse des Stacks
\$fp	\$29	Rahmen-Zeiger: für die temporäre Allokierung von Speicherplatz beim Aufruf von Unterprogrammen
\$ra	\$31	enthält die Rücksprungadresse beim Unterprogrammaufruf
PC	-	Befehlszähler
HI, LO	-	64-Bit Resultat einer Multiplikation von Integer-Zahlen bzw. Quotient und Rest einer Integer-Division

## Speicheraufteilung



## Speicheraufteilung

- Textsegment:** beinhaltet den ausführbaren Maschinencode
- Datensegment:** beinhaltet statische und dynamische Daten:
  - Speicherbereiche für **statische** Daten werden vom *Assembler* allokiert. (In C: *globale Variablen*)
  - Speicherbereiche für **dynamische** Daten werden vom *Programm* allokiert. (In C: *void \*malloc(size)*)
- Stacksegment:** beinhaltet lokale Daten und Rücksprungadressen für Unterprogramme
- Kernelsegmente:** beinhalten betriebssystemeigene Daten und Prozeduren