

# Assembler RISC

T.Mainzer, kiv.zcu.cz

## RISC

RISC, neboli Reduced Instruction Set Computer - koncepce procesorů s redukováným souborem instrukcí (vs. CISC, neboli Complex Instruction Set Computer, "bohatý" instrukčním soubor)

Redukování instrukční sady na minimum, což umožní rychlé provádění instrukcí.

RISC	CISC
Málo jednodušších instrukcí, typicky jednotaktové, snadnější pipelining	Mnoho instrukcí, často komplexních a vícetaktových
Rychlý, velký kód (mnoho instrukcí)	Pomalejší, kratší kód
Instrukce stejně dlouhé, omezené množství formátů	Mnoho instrukčních a adresních formátů
Pracuje se s registry, LOAD/STORE samostatné spec. instrukce	Složité adresní režimy v mnoha instrukcích
Plocha čipu je využita pro registry	Plocha čipu je využita pro komplexní instrukce
Kompilátor řeší závislosti instrukcí	„Jednodušší“ kompilátor

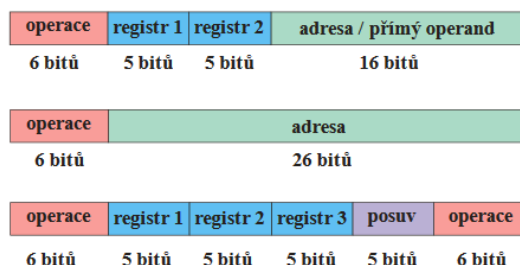
## MIPS

MIPS je procesor typu RISC (MIPS =microprocessor without Interlocked Pipeline Stages, tj. procesor bez automaticky organizované (blokové) [pipeline](#))

- instrukce mají délku 32b, registry jsou 32b, je jich 32
- využívá se pipeline
- LOAD/STORE architektura
- koprocessor pro pohyblivou řádovou čárku

### Formáty instrukcí:

- I (immediate) – přímý operand (číslo)
- J (jump) - skok
- R (register) – práce s registry



### Formát I

Obsahuje přímý operand – konstantu, adresní offset, např.

LW \$1, -0x0200(\$2) # \$1 ← [\$2-0x200] ..do reg.1 obsah adresy reg.2-konstanta 0x200

ADDIU \$1,\$2,0x1234 # \$1 ← \$2 + 0x1234 .. do reg.1 součet reg.2+konstanta 0x1234

## Formát J

Formát pro skokové instrukce, 26bit offset pro adresu (posouvá se o 2 bity doleva (zarovnání na 4B), tj. Rozsah skoku je  $2^{28}=256\text{MB}$ )

Příklad: J lab\_1 #skoci na navesti lab\_1 (relativni adresa)

Pro absolutni skok (32bitu) se používá instr. JR a registr, např. JR \$1

## Formát R

Obsahuje pole pro určení tří registrů (source, target, destination), pole pro délku posuvu a rozšířený operační kód

např. ADD \$1,\$2,\$3 # \$1  $\leftarrow$  \$2+\$3

## Pseudoinstrukce

Zpřehlednují zápis kódu (z hlediska programátora „rozšiřují“ instrukční soubor). Překladač je nahrazuje jednou či více instrukcemi.

pseudoinstrukce	Instrukce
MOVE \$1,\$2 # \$2 $\leftarrow$ \$1	ADDU \$2,\$0,\$1 # \$2 $\leftarrow$ \$1 + \$0 (pozn. \$0=zero register, vrací hodnotu 0)
LW \$10, var_1 # \$1 $\leftarrow$ [0x10018] (pozn. LW = loadWord = načte 32 bitů z paměti)	LUI \$1,0x1001 # \$1 $\leftarrow$ 0x1001 LW \$10,8(\$1) # \$10 $\leftarrow$ [\$1+8] (pozn. Do reg.\$10 zapisuje hodnotu z paměti. Adresace je složena jako bazova adresa (0x10010) + offset (8))

## Pipeline (proudové/řetězové zpracování)

MIPS ma 5 úrovnovou pipeline

IF – instruction fetch – načtení instrukce z paměti

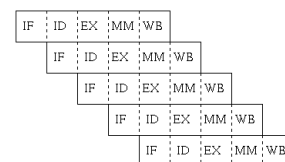
ID – instruction decode - dekodování instrukce a čtení registru

EX – execute – provedení operace / spočtení adresy

MM – memory - přístup k operandu do paměti

WB – write back – zpětný zápis výsledku do registru

A 5-segment instruction pipeline



IF: instruction fetching  
ID: instruction decoding  
EX: instruction execution  
MM: memory accessing  
WB: write back to registers

## Důsledky

### zpoždění skokových instrukcí

Zamýšlený program	Důsledek pipeline	Možná úprava	Možná úprava
Instrukce0 J label #skok instrukce1 instrukce2 ... label: instrukceA	<i>Jelikož skok (na návěší label) se provede až ve fázi EX) bude za instrukcí skoku provedena ještě jedna instrukce</i>	J label Instrukce0 instrukce1 instrukce2 ... label: instrukceA	Instrukce0 J label instrukceA instrukce1 instrukce2 ... label:

## zpoždění uložení výsledku / načítání z paměti atp.

Zamýšlený program LW \$10, var ADDI \$11, \$10, \$9	Důsledek pipeline uložení var do registru \$10 se provede ve fázi WB instr.LW a načtení z registru ve fázi ID instr.ADDI – tj. pokud má instr.ADDI použít zamýšlenou hodnotu je třeba vložit instrukce (ideálně jiné než NOPy)	Možná úprava LW \$10, var NOP NOP ADDI \$11, \$10, \$9
---	---	--

## Assembler MIPS

### Registry

- 32x 32bitů, kromě čísla registrů (např. \$29) je možné používat alias názvy (např. \$sp)

Číslo registru	Alias název	Použití
0	\$zero	dummys registr
1	\$at	rezervováno pro assembler
2	\$v0	návratové hodnoty funkcí
3	\$v1	
4	\$a0	předávání argumentů
...	...	
7	\$a3	
8	\$t0	neukládané registry
...	...	
15	\$t7	

Číslo registru	Alias název	Použití
16	\$s0	ukládané registry
...	...	
23	\$s7	
24	\$t8	neukládané registry
25	\$t9	
26	\$k0	rezervováno pro O.S.
27	\$k1	
28	\$gp	global pointer
29	\$sp	stack pointer
30	\$fp	frame pointer
31	\$ra	return address

### První program

<pre>.data #datový segment var1: .word 0x11223344 #operand1, 32b var2: .word 0xaabbcc00 #operand2, 32b var3: .word 0x00 #operand3 (výsledek), 32b .text #kodový segment .globl main #main = vstupní bod programu, musí být global lw \$t0, var1 #načtení operandu1 lw \$t1, var2 #načtení operandu2 nop #nop kvůli pipeline nop addi \$t2, \$t1, \$t0 # \$t2 ← \$t1 + \$t0 sw \$t2, var3 #uložení výsledku do var3 j \$ra #navratový skok do "systemu" nop #nop za skokem kvůli pipeline</pre>	
--	--

### Direktivy překladače (výběr)

.align n	Zarovnání v paměti na hranici 2 <sup>n</sup>
.space n	Rezervování místa n byte
.ascii str .asciiz str	Řetězec ascii – nezakončen/zakončen nulovým znakem

.byte b1,b2,...bn .half h1,h2,...hn .word w1,w2,...wn	Celé číslo – 1B,2B,4B
.text .data .ktext .kdata	Označení segmentů – kód, data, kód jádra, data jádra

## Systemové služby

V simulátoru SPIM lze použít simulované volání systémových služeb vhodné pro uživatelský vstup/výstup.

<pre> .data #datový segment str1: .asciiz "Ahoj\n" ... .text #kodový segment ... li \$v0,4      #číslo služby do \$v0 la \$a0,str1   #adresa textu do \$a0 (obecné parametry služby do \$a0-\$a2) syscall      #volání systémové služby nop ... </pre>	
--	--

## Manipulace se stringy/řetězci

### Spočítání počtu znaků v řetězci

<pre> .data str1: .asciiz "Ahoj\n" ... .text #kodový segment ... la \$t0,str1   #adresa textu do \$t0 (začátek pole znaku) li \$t1,0      #čítá počet znaku na 0 loop: lb \$t2,0(\$t0) #nahrání znaku (1byte) z adresy \$t0+0 beqz \$t2,end  #je-li t2 nulové (=konec řetězce) skok na návěst end add \$t0,\$t0,1 #inkrementuj adresu (\$t0 ← \$t0+1) add \$t1,\$t1,1 #inkrementuj počítadlo znaku j loop        #opakuji hledání nop end: ... </pre>	
---	--

### Převod na malá písmena

<pre> .data str1: .asciiz "HalLo wORld" eol:  .asciiz "\n" .text #kodový segment .globl main main: la \$t0,str1   #adresa textu do \$t0 (začátek pole znaku) loop: lb \$t2,0(\$t0) #nahrání znaku (1byte) z adresy \$t0+0 do \$t2 beqz \$t2,end  #je-li t2 nulové (=konec řetězce) skok na návěst end  li \$t1,'A'    #constant 'A' bge \$t2,\$t1,upa #skok na upa když \$t2 větší nebo rovno 'A' (\$t1) j next        #else continue upa: li \$t1,'Z'    #constant 'Z'      ble \$t2,\$t1,upz #skok když menší nebo rovno </pre>	
---	--

```

j next
upz:  add $t2,#t2,32    #inkrementuj znak o 32 = preved velke na male
      sb $t2,($t0)      #uloz zmeneny znak zpet
next:
      add $t0,1         #inkrementuj ukazatel
      j loop            #opakuj
      nop
end:   li $v0,4
      la $a0,str1
      syscall           #vypis vysledek
      li $v0,4
      la $a0,eol
      syscall           #odradkuj
      j $ra             #navratovy skok do "systemu"
      nop

```

## Simulátor SPIM

Simulátor SPIM ve verzi pro Windows s názvem **QTspim** lze najít na adrese <http://sourceforge.net/projects/spimsimulator/files/>.

V případě zájmu lze najít i [další verze pro jiné operační systémy](#) .

Na stejné adrese je ke stažení i [přehledný manuál k simulátoru SPIM](#), včetně instrukčního souboru procesoru MIPS

V nastavení Simulator-Settings lze vypnout/zapnout volby související s pipeline (povolování/zakazování zpoždění):

