

Aula 12

Arquiteturas RISC

RISC

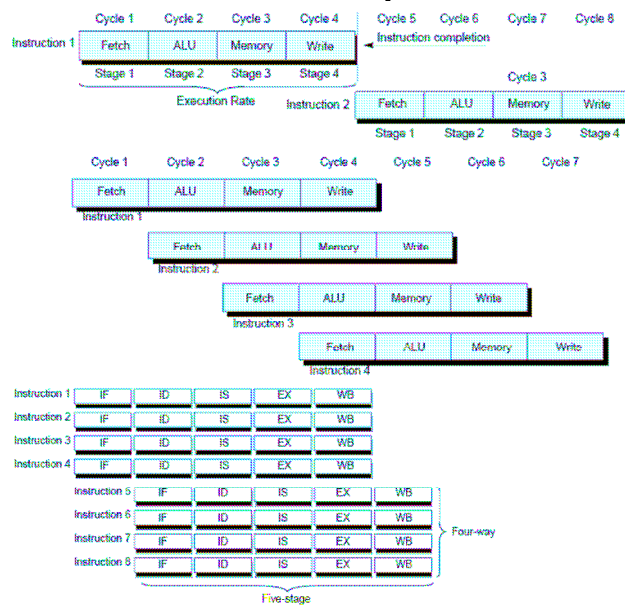
- RISC- reduced instruction set computer
- “RISC: any computer announced after 1985”

Steven Przybylski –Trabalhou no projeto do MIPS (Stanford)

Processadores RISC

- Baseado no princípio “mais simples = mais rápido”
- Conjunto de instrução simples auxilia no desenvolvimento de processadores mais rápidos
- Passar a tarefa de otimização para o software
 - O desenvolvedor e o compilador tem mais tempo para otimizações
 - Instruções complexas podem ser implementadas utilizando simples instruções (multiplicação usando somas)

Pipeline



Conjunto de instruções

Acumulador:

1 address add A $acc \leftarrow acc + mem[A]$

Pilha (Java):

Proposito geral:

2 address add A B $EA(A) \leftarrow EA(A) + EA(B)$

3 address add A B C $EA(A) \leftarrow EA(B) + EA(C)$

Load/Store:

3 address add Ra Rb Rc $Ra \leftarrow Rb + Rc$

load Ra Rb $Ra \leftarrow mem[Rb]$

store Ra Rb $mem[Rb] \leftarrow Ra$

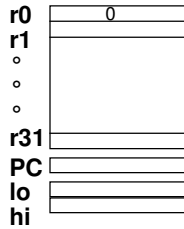
Processadores RISC- Exemplos

	DLX	MIPS I	PA-RISC	PowerPC	SPARC V8
Date announced	1990	1986	1986	1993	1987
Instruction size (bits)	32	32	32	32	32
Address space (size, model)	32 bits, flat	32 bits, flat	48 bits, segmented	32 bits, flat	32 bits, flat
Data alignment	Aligned	Aligned	Aligned	Unaligned	Aligned
Data addressing modes	2	2	5	4	2
Protection	Page	Page	Page	Page	Page
Minimum page size	4 KB	4 KB	4 KB	4 KB	8 KB
I/O	Memory mapped	Memory mapped	Memory mapped	Memory mapped	Memory mapped
Integer registers (number, model, size)	31 GPR × 32 bits	31 GPR × 32 bits	31 GPR × 32 bits	31 GPR × 32 bits	31 GPR × 32 bits
Separate floating-point registers	32 × 32 or 16 × 64 bits	16 × 32 or 16 × 64 bits	16 × 32 or 16 × 64 bits	16 × 32 or 16 × 64 bits	16 × 32 or 16 × 64 bits
Floating-point format	IEEE 754 single, double	IEEE 754 single, double	IEEE 754 single, double	IEEE 754 single, double	IEEE 754 single, double

	IBM 360/370	Intel 8086	Motorola 68000	DEC VAX
Date announced	1964/1970	1978	1980	1977
Instruction size(s) (bits)	16,32,48	8,16,24,32,40,48	16,32,48,64,80	8,16,24,32,...,432
Addressing (size, model)	24 bits, flat/31 bits, flat	4+16 bits, segmented	24 bits, flat	32 bits, flat
Data aligned?	Yes 360/ No 370	No	16-bit aligned	No
Data addressing modes	2/3	5	9	≥ 14
Protection	Page	None	Optional	Page
Page size	2 KB & 4 KB	—	0.25 to 32 KB	0.5 KB
I/O	Opcode	Opcode	Memory mapped	Memory mapped
Integer registers (size, model, number)	16 GPR × 32 bits	8 dedicated data × 16 bits	8 data & 8 address × 32 bits	15 GPR × 32 bits
Separate floating-point registers	4 × 64 bits	Optional: 8 × 80 bits	Optional: 8 × 80 bits	0
Floating-point format	IBM (floating hexadecimal)	IEEE 754 single, double, extended	IEEE 754 single, double, extended	DEC

Fonte: Hennessy & Patterson. Computer Architecture: A quantitative approach

MIPS R3000



Espaço de endereçamento 2^{32} x bytes

Registradores 31 x 32-bit GPRs
(R0=0)

Lógicas e aritméticas

Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU,

Addl, AddIU, SLTI, SLTIU, Andl, Orl, Xorl, LUI

SLL, SRL, SRA, SLLV, SRLV, SRAV

Acesso a memória

LB, LBU, LH, LHU, LW, LWL, LWR

SB, SH, SW, SWL, SWR

Controle

J, JAL, JR, JALR

BEq, BNE, BLEZ, BGTZ, BLTZ, BGEZ, BLTZAL, BGEZAL

Formato de instruções

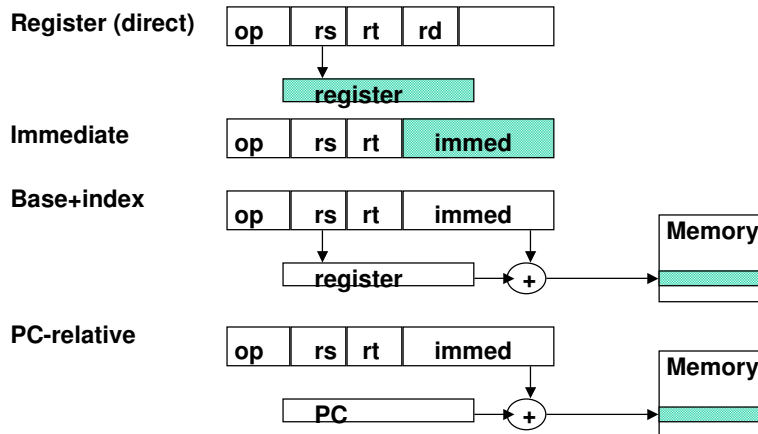


• Tamanho do código => instruções de tamanho variável

• Desempenho => tamanho fixo

- Decodificação simples

MIPS – Modos de endereçamento



Acessos a memória

- Dados de 16 bits, alinhados em endereços múltiplos de 2
- Dados de 32 bits, alinhados em endereços múltiplos de 4
- Carga de dados imediatos de 32 bits
 - `li $4, 0x1234567`
 - É executado como:
 - `lui $4, 0x1234` # carrega na parte alta o valor 0x1234
 - `ori $4, $0, 0x5678` # OR

Slot de desvios e carga

- Para utilizar o máximo do pipeline, e ocultar as penalidades de um desvio ou carga de dados
- “delayed branch slot” e “delayed load slot”: a instrução subsequente a um desvio ou load/store é sempre executada
- Ex:


```
..
lw $3, val1
nop
..
```

MIPS

- R0, \$0 = ## zero
- R2, \$2 = ## return value / system call number
- R4, \$4 = ## 1st argument
- R5, \$5 = ## 2nd argument
- R6, \$6 = ## 3rd argument
- R7, \$7 = ## 4th argument
- R29, \$29 = ## stack pointer
- R30, \$30 = ## frame pointer
- R31, \$31 = ## return addr

Nome	Número	Uso	Preservado em chamadas?
\$zero	0	Constante 0	n.d
\$v0-\$v1	2-3	Resultados e avaliações de expressões	Não
\$a0-\$a3	4-7	Argumentos	Sim
\$t0-\$t7	8-15	Temporários	Não
\$s0-\$s7	16-23	Salvos	Sim
\$t8-\$t9	24-25	Temporários	Não
\$gp	28	Ponteiro global	Sim
\$sp	29	Ponteiro para pilha	Sim
\$fp	30	Ponteiro para frame	Sim
\$ra	31	Endereço de retorno	Sim

PCSpim

- Simulador baseado no conjunto de instruções do MIPS
- Não é necessário gerar o executável, pois o simulador lê o código assembler e gera a memória de código e de dados
- Download:
<http://www.cs.wisc.edu/~larus/spim.html>

Exemplo

```
.data
val1: .word 1000
val2: .word 1500
vector: .space 400 # 100 words

.text
lw $3, val1
nop
lw $4, val2
nop

bgt $3, $4, fim
nop
sw $0, val1 # armazena o valor 1 no endereço val1
nop

fim:
jr $31
nop
```

Exemplo

```
• .data
• val1: .word 1000
• val2: .word 1500
• vector: .space 400

• .text
• .globl main
• main:

• li $10, 100
• add $4, $0, $0

• LC0:
•
• sw $10, vector($4) # armazena o valor do registrador $10 no endereço val1
• nop
• addi $10, $10, -1
• addi $4, $4, 0x4 # soma o registrador $4 com o valor 0x4 e armazena o resultado em $4
• bne $0, $10, LC0
• nop

•
• fim:
• jr $31
• nop
```

Exemplo

```
• .data
• str: .asciiz "Teste"

• .text
• .globl main

• main:
• li $2, 0x1
• add $2, $2, $2

• la $a0, str # arg1
• li $v0, 4
• syscall

• jr $31
• nop
```

Syscall

- #define PRINT_INT_SYSCALL 1
- #define PRINT_FLOAT_SYSCALL 2
- #define PRINT_DOUBLE_SYSCALL 3
- #define PRINT_STRING_SYSCALL 4

- #define READ_INT_SYSCALL 5
- #define READ_FLOAT_SYSCALL 6
- #define READ_DOUBLE_SYSCALL 7
- #define READ_STRING_SYSCALL 8

- #define SBRK_SYSCALL 9

- #define EXIT_SYSCALL 10

- #define PRINT_CHARACTER_SYSCALL 11
- #define READ_CHARACTER_SYSCALL 12

- #define OPEN_SYSCALL 13
- #define READ_SYSCALL 14
- #define WRITE_SYSCALL 15
- #define CLOSE_SYSCALL 16

- #define EXIT2_SYSCALL 17

Subrotinas

main:

...

jal Rotina

nop

...

Rotina:

...

...

jr \$31

nop

Nome	Numero	Uso	Preservado em chamadas?
Szero	0	Constante 0	n.d
Sv0-Sv1	2-3	Resultados e avaliações de expressões	Não
Sa0-Sa3	4-7	Argumentos	Sim
St0-St7	8-15	Temporários	Não
Ss0-Sv7	16-23	Salvos	Sim
St8-St9	24-25	Temporários	Não
Sgp	28	Ponteiro global	Sim
Ssp	29	Ponteiro para pilha	Sim
Sfp	30	Ponteiro para frame	Sim
Sra	31	Endereço de retorno	Sim

Exemplo

- pread:

```
la $a0,prompt # print string
```

```
li $v0,4 # service 4
```

```
syscall li $v0,5 # read int into $v0 syscall  
# service 5
```

```
jr $31 # return
```

```
nop # branch delay slot
```

```
.data
```

```
prompt: .asciiz "Enter an integer"
```