INSTITUTO FEDERAL
SANTA CATARINA

Microprocessadores

Hugo Marcondes
hugo.marcondes@ifsc.edu.br

Aula 01

Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina

O que é uma Arquitetura de Computadores

INSTITUTO FEDERAL
SANTA CATARINA

Aplicação

O "GAP" é muito grande para ser tratado de uma só vez !

Arquitetura de computadores é o projeto de camadas de abstração/implementação que nos permite executar aplicações de processamento de dados de forma eficiente utilizando tecnologias manufaturadas

Física

O que é uma Arquitetura de Computadores

INSTITUTO FEDERAL
SANTA CATARINA

- Aplicação
- Algoritmos
- Linguagens de Programação
- Sistemas Operacionais
- ISA
- Microarquitetura
- Nível RTL
- Portas Lógicas
- Circuitos
- Dispositivos
- Física

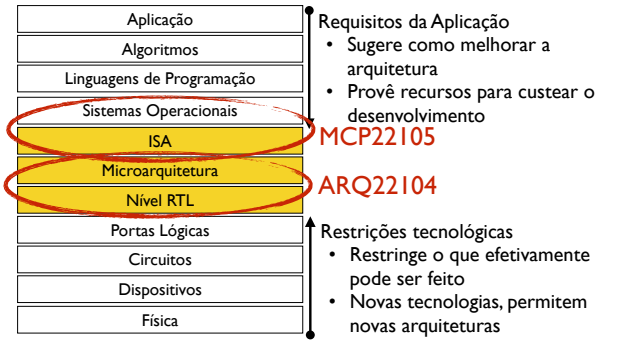
O que é uma Arquitetura de Computadores

INSTITUTO FEDERAL
SANTA CATARINA

- Aplicação
- Algoritmos
- Linguagens de Programação
- Sistemas Operacionais
- ISA
- Microarquitetura
- Nível RTL
- Portas Lógicas
- Circuitos
- Dispositivos
- Física

} Arquitetura de Computadores

O que é uma Arquitetura de Computadores



Modelos de programação

- Computadores são máquinas para a realização de cálculos
  - Conceito de programa armazenado
  - Sequência de instruções, operando dados
- Diversos modelos de programação
  - Registradores
    - Banco de Registradores
    - Acumuladores
  - Pilha

Modelos de Programação

- Banco de Registradores
  - Principalmente máquinas RISC / Load Store
- Acumuladores
  - Poucos registradores
  - Geralmente específicos de máquinas CISC
- Máquinas de Pilha
  - Dados presentes na memória
  - Empilhados, operações são aplicadas aos dois elementos no topo da pilha.

Linguagem de máquina

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101111000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

- Calcula e imprime a soma dos quadrados dos inteiros entre 0 e 100

Linguagem de máquina x montagem

- Linguagem de máquina ("machine language")
  - Representação binária das instruções de um processador
    - Campos codificados em binário
    - 000000 10001 10010 01000 00000 100000
- Linguagem de montagem ("assembly language")
  - Representação simbólica da linguagem de máquina
    - Mnemônicos associados aos campos
    - add $t0, $s1, $s2

A linguagem de montagem

- Principais elementos
  - Mnemônicos de operações
    - add, sub, etc ...
  - Mnemônicos de registradores
    - $t0
  - Mnemônicos de endereços
    - Rótulos / Labels
    - Símbolos

Linguagem de máquina

```
#include <stdio.h>

int
main (int argc, char *argv[])
{
    int i;
    int sum = 0;

    for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
    printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

- Calcula e imprime a soma dos quadrados dos inteiros entre 0 e 100

Linguagem de máquina

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra

        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

- Calcula e imprime a soma dos quadrados dos inteiros entre 0 e 100

Linguagem de máquina

```
addiu    $29, $29, -32
sw       $31, 20($29)
sw       $4, 32($29)
sw       $5, 36($29)
sw       $0, 24($29)
sw       $0, 28($29)
lw       $14, 28($29)
lw       $24, 24($29)
multu    $14, $14
addiu    $8, $14, 1
slti     $1, $8, 101
sw $8,   28($29)
mflo     $15
addu     $25, $24, $15
bne      $1, $0, -9
sw       $25, 24($29)
lui      $4, 4096
lw       $5, 24($29)
jal      1048812
addiu    $4, $4, 1072
lw       $31, 20($29)
addiu    $29, $29, 32
jr       $31
move     $2, $0
```

- Calcula e imprime a soma dos quadrados dos inteiros entre 0 e 100

Linguagem de máquina

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101111000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101111001000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

- Calcula e imprime a soma dos quadrados dos inteiros entre 0 e 100

Revisitando a Arquitetura MIPS

- MIPS (Microprocessor without Interlocked Pipeline Stages) é uma arquitetura de microprocessador.
  - 1980 - Berkeley
    - David Patterson e Carlo Séquin
    - Processador RISC
  - 1981- Stanford
    - John Hennessy, aprimora o RISC e cria-se o MIPS.
    - Introduziu de forma eficiente o uso de pipelines para o paralelismo de instruções.

MIPS

- O MIPS foi um dos µProcessadores mais comercializados do mundo. Ainda é muito empregado e vendido.
  - Arquitetura é licenciada para diversos fabricantes de chip
  - Home entertainment, networking, communication
- É extremamente didático e possui uma arquitetura elegante, por isso o seu estudo em sistemas digitais e arquitetura de computadores.



MIPS TECHNOLOGIES

"To command a computer's hardware, you must speak its language. The words of a computer's language are called instructions, and its vocabulary is called an instruction set"
- David Patterson

MIPS Assembly

- ISA simples e didático
  - Muito similar a ARM (12 bilhões de unidades vendidas em 2014)
- Poucas instruções, instruções simples
- Acesso a memória via LOAD e STORE
- Operações ULA, apenas com registradores (3 operandos)
- Instruções de tamanho fixo (32 bits)
- Poucos modos de endereçamento de dados
- Instruções "compare-and-branch"

Banco de Registradores

- Banco com 32 registradores
  - Palavras de 32 bits (4 bytes)

Número	Nome	Uso
0	$zero	Contém o valor 0 (zero)
1	$at	Temporário do Montador
2-3	$v0-$v1	Retorno de função
4-7	$a0-$a3	Argumentos
8-15	**$t0-$t7**	**Temporários**
16-23	**$s0-$s7**	**Temporários Salvos**
24-25	**$t8-$t9**	**Temporários**
26-27	$k0-$k1	Reservado para o kernel
28	$gp	Global Pointer
29	$sp	Stack Pointer
30	$fp	Frame Pointer
31	$ra	Return Address

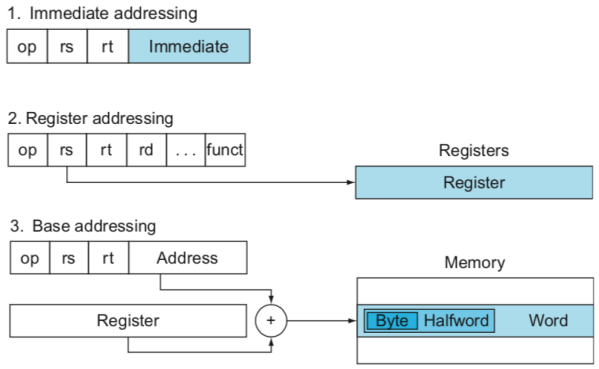
Operadores em memória

- Memória principal é utilizada para dados compostos
  - Arrays, estruturas, dados dinâmicos
- Para aplicar operações lógicas
  - Carregar valores da memória em registradores
  - Armazenar o resultado do registrador na memória
- Memória é endereçada por byte
  - Cada endereço corresponde a um byte (8-bits)
- Palavras são alinhadas na memória
  - Endereços devem ser múltiplos de 4
- Organização dos dados na memória
  - Big Endian
  - Little Endian

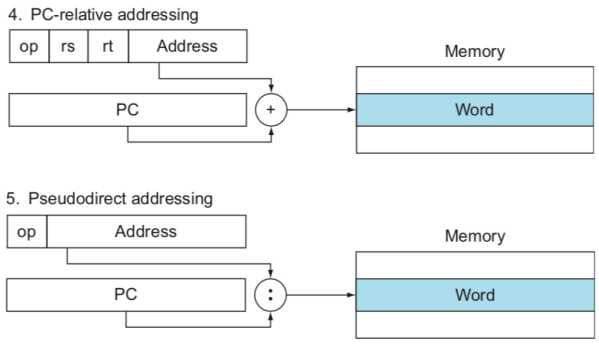
Modos de Endereçamento do MIPS

- **Imediato**
  - O operando é uma constante dentro da instrução
- **Registrador**
  - O operando está no banco de registradores
- **Registrador Base + Índice**
  - O operando está na memória. O endereço de memória do operando é formado pelo valor do registrador base somado ao valor de índice
- **Relativo ao PC**
  - O endereço alvo é a soma do valor do PC com uma constante presente na instrução
- **Pseudodireto**
  - O endereço alvo é composto por uma constante de 26 bits concatenada com o bits mais significativos do PC.

Modos de Endereçamento do MIPS



Modos de Endereçamento do MIPS



Exemplo 1: Operandos em Memória

- código C:

```
g = h + A[8];
```

  - g está em $s1, h em $s2, endereço base de A em $s3

- Código MIPS:
  - Índice 8 requer um offset de 32
    - 4 bytes por palavra !

```
lw  $t0, 32($s3)    # load word
add $s1, $s2, $t0
```

offset

base register

Exemplo 2: Operandos em Memória

- código C:

```
A[12] = h + A[8];
```

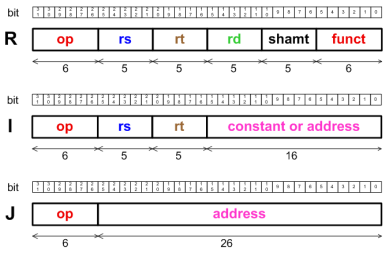
  - h está em $s2, endereço base de A em $s3

- Código MIPS:
  - Índice 8 requer um offset de 32

```
lw  $t0, 32($s3)    # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)    # store word
```


Instruções do MIPS

- Tamanho fixo de 32 bits
  - Três formatos de instrução



add $t0, $s1, $s2

lw $t0, 32 ($s1)
addi $s3, $s3, 4

j desvio

Instruções do tipo R

- Instruções que operam apenas com registradores

op (opcode) = 0, código que diz qual instrução deve ser executada
rs (*source register*) – registrador fonte
rt (*target register*) – registrador alvo
rd (*destination register*) registrador destino
shamt (*shift amount*) – valor usado em instruções de deslocamento
funct (função) – código de função para operações lógico/aritméticas

O formato *assembly* é:
<instrução> <regDestino><regFonte><regTarget>

Exemplo: **add $s0, $s1 $s2** # s0 = s1 + s2
add $16, $17 $18

Código de Máquina: 0x02328020
0b000000 10001 10010 10000 00000 100000

Instruções do tipo I

- Instruções que operam com um valor imediato

O formato *assembly* para instruções lógico/aritméticas é:
<instrução> <regTarget><regFonte><imediato>

Exemplo: **addi $s0, $zero, 42** # s0 = 0 + 42
addi $16, $0, 0x002A

Código de Máquina: 0x2010002A
0b001000 00000 10000 0000000000101010

O formato *assembly* para instrução de desvio condicional é:
<beq/bne> <regFonte><regTarget><end/label>
if(r1==r2) then PC ← PC + 4 + imediato

Exemplo: **beq $s3, $s4, *label*** # se $s3==$s4 o programa desvia para o end. apontado por *label*
instrução # beq $19,$20, 0x0000
***label*: instrução**

Código de Máquina: 0x12740001
0b000100 10011 10100 0000000000000001

Instruções do tipo I

I	op	rs	rt	constant or address

O formato *assembly* para instruções *load word* e *store word, de acesso a memória, é*:

<lw> <regDestino> <offset><end/regFonte> (regDestino ← mem[end + offset])
<sw> <regFonte> <offset><end/regDestino> (mem[end + offset] ← regFonte)

Exemplo: **lw $t0, 12($s3)** # t0 = mem[12+ valor em $s3]
lw $8, 0x000C($19)
Código de Máquina: 0x8E680000C
0b100011 10011 01000 0000000000001100

Exemplo: **sw $t0, 48($s4)** # mem[48+ valor em $s4] = $t0
sw $8, 0x0030($20)
Código de Máquina: 0xAE880030
0b101011 10100 01000 0000000000110000

Instruções do tipo J

- Instruções para saltos incondicionais longos

J	op	address

O formato *assembly* para instruções *Jump* é:
<instrução> <end/label>

Exemplo: #laço infinito com incremento de $s1
loop: **addi $s1, $s1, 1** # s1 = s1 + 1
j *loop* # j 0x00400000
o endereço depende da posição do código no #programa, ele é devidamente adaptado pelo #compilador
Código de Máquina: 0x08100000
0b000010 00000100000000000000000000

No *Jump* a execução do programa é desviada para qualquer posição de memória indicada pelo endereço de desvio.

MIPS Instruction Set

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Three operands; overflow detected
	add immediate	`addi $s1,$s2,100`	$s1 = $s2 + 100	+ constant; overflow detected
	add unsigned	`addu $s1,$s2,$s3`	$s1 = $s2 + $s3	Three operands; overflow undetected
	subtract unsigned	`subu $s1,$s2,$s3`	$s1 = $s2 – $s3	Three operands; overflow undetected
	add immediate unsigned	`addiu $s1,$s2,100`	$s1 = $s2 + 100	+ constant; overflow undetected
	move from coprocessor register	`mfc0 $s1,$epc`	$s1 = $epc	Copy Exception PC + special regs
	multiply	`mult $s2,$s3`	Hi, Lo = $s2 × $s3	64-bit signed product in Hi, Lo
	multiply unsigned	`multu $s2,$s3`	Hi, Lo = $s2 × $s3	64-bit unsigned product in Hi, Lo
	divide	`div $s2,$s3`	Lo = $s2 / $s3, Hi = $s2 mod $s3	Lo = quotient, Hi = remainder
	divide unsigned	`divu $s2,$s3`	Lo = $s2 / $s3, Hi = $s2 mod $s3	Unsigned quotient and remainder
	move from Hi	`mfhi $s1`	$s1 = Hi	Used to get copy of Hi
	move from Lo	`mflo $s1`	$s1 = Lo	Used to get copy of Lo
Data transfer	load word	`lw $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	`sw $s1,20($s2)`	Memory[$s2 + 20] = $s1	Word from register to memory
	load half unsigned	`lhu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	`sh $s1,20($s2)`	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte unsigned	`lbu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	`sb $s1,20($s2)`	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	`ll $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store conditional word	`sc $s1,20($s2)`	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half atomic swap
	load upper immediate	`lui $s1,100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits

MIPS Instruction Set

Category	Instruction	Example	Meaning	Comments
Logical	AND	`AND $s1,$s2,$s3`	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	OR	`OR $s1,$s2,$s3`	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	NOR	`NOR $s1,$s2,$s3`	$s1 = ~ ($s2 |$s3)	Three reg. operands; bit-by-bit NOR
	AND immediate	`ANDi $s1,$s2,100`	$s1 = $s2 & 100	Bit-by-bit AND with constant
	OR immediate	`ORi $s1,$s2,100`	$s1 = $s2 | 100	Bit-by-bit OR with constant
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10	Shift left by constant
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10	Shift right by constant
Condi-tional branch	branch on equal	`beq $s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1,$s2,25`	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	`slti $s1,$s2,100`	if ($s2 < 100) $s1 = 1; else $s1=0	Compare < constant; two's complement
	set less than unsigned	`sltu $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1=0	Compare less than; natural numbers
	set less than immediate unsigned	`sltiu $s1,$s2,100`	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; natural numbers
Uncondi-tional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to $ra	For switch, procedure return
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000	For procedure call

Exemplo: Formato de instruções

INSTITUTO FEDERAL
SANTA CATARINA

MIPS machine language

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
and	R	0	18	19	17	0	36	and $s1,$s2,$s3
or	R	0	18	19	17	0	37	or $s1,$s2,$s3
nor	R	0	18	19	17	0	39	nor $s1,$s2,$s3
andi	I	12	18	17	100			andi $s1,$s2,100
ori	I	13	18	17	100			ori $s1,$s2,100
sll	R	0	0	18	17	10	0	sll $s1,$s2,10
srl	R	0	0	18	17	10	2	srl $s1,$s2,10
beq	I	4	17	18	25			beq $s1,$s2,100
bne	I	5	17	18	25			bne $s1,$s2,100
slt	R	0	18	19	17	0	42	slt $s1,$s2,$s3
j	J	2	2500					j 10000
jr	R	0	31	0	0	0	8	jr $ra
jal	J	3	2500					jal
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer, branch format

MARS: MIPS Assembler and Runtime Simulator

INSTITUTO FEDERAL
SANTA CATARINA