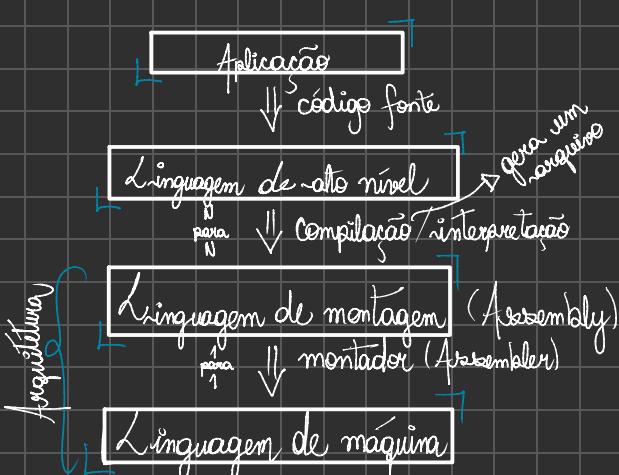
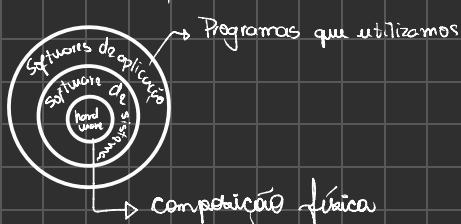


Arquitetura de um computador

Compreende a projeto estrutural de um computador, ou seja, com componentes lógicos que determinam seu funcionamento.

Um sistema computacional atualmente compreende três camadas



A arquitetura de um computador, definida unidas suas instruções, é chamada de Arquitetura do Conjunto de Instruções (ISA). O ISA pode ser classificado como RISC ou CISC.

RISC → Reduced Instruction Set Computer

Possui poucas instruções, simples e uso menor padrão

CISC → Complex Instruction Set Computer

Possui centenas de instruções em seu conjunto, capazes de executar uma grande diversidade de operações

Instruções	RISC	CISC
Projeto	- menor gte - mais simples e padronizadas	- maior gte - mais complexas
Execução	- centrado no software	- centrado no hardware

Linguagem de montagem Assembly

↳ Arquitetura MIPS 32 bits

↳ Simulador: SPIM

↳ spim -f código_spim

↳ windows: baixe o spim e adicione a pasta que contém o arquivo spim.exe à variável de ambiente Path.

↳ linux: apt install spim.

↳ Ambiente ideal: VSCode + SPIM + WSL.

Operações aritméticas

• add a,b,c # a = b + c

 | comentários

- Todas as instruções aritméticas seguem este formato:

• sub

• mul

• div

Quais são os variáveis? Registradores (32 bits) → tamanho de um int.

Registradores são unidades de memória de acesso imediato que ficam dentro do processador.

Um processador MIPS de 32 bits possui 32 registradores em sua unidade principal, que opera com inteiros.

Usando registradores disponíveis para uso só:

Notações

\$t0 a \$t7 8 regis.

\$s0 a \$st 16-23 8 regis.

\$t8 a \$t9 24-25 2 regis.

18 regis.

→ não muito relevantes

Temporário

Salvo

Temporário

Os 14 registradores restantes são de uso reservado, veremos alguns adiante

Ex: Código alto nível,

$$f = (g+h) - (i+j);$$

↳ compilação

add a,g,h

add b,i,j

sub f,a,b

add \$t0,\$s1,\$s2 # t0 = g+h

add \$t1,\$s3,\$s4 # t1 = i+j

sub \$s0,\$t0,\$t1 # f = t0 - t1

Como fazer um Olá Mundo em Assembly?

Um programa em assembly MIPS possui o seguinte esqueleto:

```
.data
    # seção de dados
.text
    # seção de código
main:
    # rotulo
```

A execução do código começa na linha rotulada por "main" e segue linha após linha (salvo se houver um desvio). A esse paradigma chamamos "sequencial".

→ Tipos de dados

- word w1, w2, ..., wn → dado de 32 bits
- byte b1, b2, ..., bn → dados de 8 bits;
- ascii str → cadeia de caracteres ASCII terminados pelo caractere nulo. Ex:

```
.data
msg: .ascii "Olá Mundo"
x: word 110
```

.text

```
main:
    li $v0, 4
    la $a0, msg
    syscall
```

li \$v0, 10

syscall

→ Chamadas ao sistema (system call)

Existem 4 tipos de sistema:

- Escrita na saída padrão;
- Leitura da entrada padrão;
- Alocarção de memória;
- Encerramento do processo.

Para executar uma chamada ao sistema, precisamos fazer o seguinte:

- Carregar no registrador \$v0 o código da chamada;
- Carregar os argumentos nos registradores \$a0 a \$a3 (opcional, depende da operação);
- Executar usando a instrução syscall;
- Colher o retorno (opcional. Nos registradores \$v0 e \$v1).

Representação de zero, 0

Para representar o 0, há um registrador especial chamado "zero". Ex:

add \$t0, \$zero, \$t0 # copia o valor de \$t0 para \$t0

→ Instruções imediatas

Variáveis de algumas instruções terminadas mnémico com i, que trabalha com dois registradores + uma constante (nessa ordem). Ex:
addi \$t0, \$t0, 1 # - incrementa o valor de \$t0 em uma unidade --> \$t0 = \$t0 + 1

→ Pseudoinstruções

Estas instruções simplificadas que nós pertencem à ISA e que são traduzidas pelo compilador para instruções 3 pseudoinstruções (tabelas):

- li reg, const # → copia a const para o registrador. #li = "load immediate"
- move reg1, reg2 # → copia o conteúdo de reg2 para reg1.
- la reg, label # → corrige o endereço de memória associado ao rotulo label para o registrador

Imprimindo "Olá Mundo/n"

```
.data
msg: .ascii "Olá Mundo/n" # grava o "Olá Mundo" no registrador "msg"
```

.text

main:

```
li $v0, 4 # syscall 4 imprime a string matela
la $a0, msg
syscall
```

```
li $v0, 10 # syscall 10 encerra
syscall
```

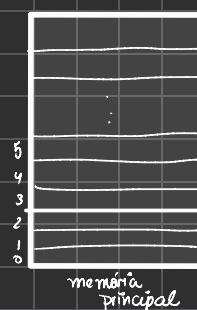
20/09

Redondo o código

spin ← código spin



Instruções de acesso à memória



1 byte = 8 bits

$$1 \text{ KiB} = 2^{10} \text{ B}$$

$$1 \text{ MiB} = 2^{20} \text{ B}$$

$$1 \text{ GiB} = 2^{30} \text{ B}$$

lw reg1, const (reg2) → li o conteúdo do reg1 na memória no end reg2 + const
↳ load word (leitura)

sw reg1, const (reg2) → salva o conteúdo de reg1 na memória no end reg2 + const
↳ store word (escrita)

OBS: Os dados sempre são salvos em endereços múltiplos de 4. Isso se chama restrição de alinhamento.

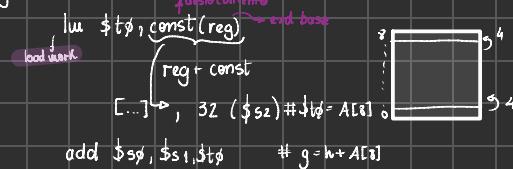
OBS: Para utilizar as instruções de acesso à memória, o espaço na memória deve estar alocado. Há duas formas de alocar espaço na memória:

- ① Declarações em .data
- ② Syscall 9.

Ex: Código em C

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

g em $\$sp$, h em $\$s_1$ e end base de A em $\$s_2$.



Ex:

$$A[12] = h + A[8]$$

lhu \$t0, 32(\$s2)

add \$s0, \$s1, \$t0

sll \$s0, 48(\$s2)

Sistema numérico

- ① Posicional

Ex: 100

O mesmo símbolo possui valores distintos a depender da posição que ocupa.

- ② Possui base = qtd. de símbolos p/ representação

Considerando um número

$$X = x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0$$

Considerando uma base b

$$X = x_{n-1}b^{n-1} + x_{n-2}b^{n-2} + \dots + x_2b^2 + x_1b + x_0$$

Ex: 101001 na base 2.

$$x = 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 41$$

Representação decimal (binária)

Ex: 0010101 = 21

4	Bit de sinal: bit mais significativo
0	
+	

Bit de sinal: bit mais significativo
0+
1-

Exercício: relembrar outras formas de representação de números com sinal em particular

↳ bit de sinal (magnitude)

↳ complemento a 1

↳ hexa

2.2.09

Representação numérica

↳ Possui base

↳ É opcional

Processador MIPS 32 possui registradores de 32 bits (4 bytes). Qual a capacidade de representação?

* Números naem real

Menor: 00000...00 } unsigned

Maior: 11111...11 } int
32 bits

$$\rightarrow 2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^{31}$$

$$= 2^0(2^{32}-1) = 2^{32}-1 = 4.294.967.295$$

2-1

↳ 0 até maior número

* Números com sinal

X ----- real

Menor 000...000 } int

Maior 111...111 } 31 bits

$$\rightarrow 2^{31}-1 = -2.147.483.648 a 2.147.483.647$$

OBS: O MIPS 32 não implementa o long, mas a ideia seria usar os registradores, o que nos daria:

- unsigned long: 0 a $2^{64}-1$

- long: - 2^{63} a $2^{63}-1$

OBS 2: No Assembly MIPS, os inteiros interpretam o binário como um número com sinal. Algumas possuem variantes terminadas por "u" que interpretam os inteiros com sinal real.
↳ wr addu

Representação das inteiros em linguagem de máquina

As instruções em assembly MIPS vão traduzidas para o binário e seguem 3 formatos padrão: Tipo R, Tipo I e tipo J.

* Formato tipo R (3 registradores)

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

op: código da operação $\xrightarrow{\text{Tipo}}$ tabeladoz
funct: código aritmético

rs e rt: registradores operandos (nenhuma ordem)

rd: registrador de destino

shamt: tamanho do deslocamento

Exemplo:

add \$t0, \$s1, \$s2



Tipo R	\$s1	\$s2	\$t0	\emptyset	add

\emptyset	17	18	8	\emptyset	32

Binário de 32 bits

* Formato Tipo I (2 regos + 1 const)

op	rs	rt	constante / endereço
6 bits	5 bits	5 bits	16 bits

Exemplo:

addi \$t0, \$s1, 21

Tipo I	\$t0	\$s1	21

\emptyset	8	16	21

Binário de 32 bits

OBS: ① No Tipo I, uma constante varia de -2^{15} a $2^{15}-1$

↳ para operações imediatas, vai precisar de uma constante maior, deve-se realizar na memória (data) e carregar num registrador usando lw

↳ Para lw e lh, replica porque na lógica né send. base + offset
deslocamento

Instruções de lógicas de deslocamento

① Deslocamentos \rightarrow Tipo R

↳ esquerda \$t0 \$t0 \$s0 \emptyset

valift left rd re logid de bits \rightarrow shamt (valift = amount)
logical

↳ direita \$t0 \$t0 \$s0, 10

valift right

logical

Lógicas

• and \$t0, \$s0, \$s1: faz o "et" lógico bit a bit entre \$s0 e \$s1 e resulta em \$t0

• or \$t0, \$s0, \$s1: "ou" lógico bit a bit.

• nor \$t0, \$s0, zero: "ou" lógico bit a bit se resulta zero.

• xor \$t0, \$s0, \$s1: "xor" lógico bit a bit.

Assembly - linguagem de programação
Assembler - compilador

Operações:

- Load
- Store
- Move

Registradores

\$zero - 0

\$v0, \$v1 - retornam resultados das funções

\$a0, \$a1, \$a2, \$a3 - argumentos de função

\$ra - retorna o endereço de uma função

\$t1 a \$t9 - temporárias, que podem ser modificadas por funções

\$s1 a \$s9 - salvores.

Comandos li \$v0

Comando	Significado
li \$v0, 1	imprimir inteiro
li \$v0, 2	imprimir float
li \$v0, 3	imprimir double
li \$v0, 4	imprimir String ou char
li \$v0, 5	ler inteiro
li \$v0, 6	ler float
li \$v0, 7	ler double
li \$v0, 8	ler String ou char
li \$v0, 10	encerrar programa principal

.data → área para dados da memória
msg .ascii "Olá, mundo!"

.text → área para instruções do programa
li \$v0, 4
la \$a0, msg
syscall

Imprimindo um char

.data
caractere byte 'A'

.text
li \$v0, 4
la \$a0, caractere
syscall

} finalizar o programa

Impressão de inteiros

.data

idade: .word 56

.text

li \$v0, 1

lui \$a0, idade

syscall

li \$v0, 10

syscall

Soma de inteiros

add \$t0, \$t1, \$t2 # t0 = t1 + t2

addi \$t0, \$s1, 15 # t0 = t1 + 15

.text

li \$t0, 75

li \$t1, 25 → addi \$t2, \$zero, 25
(-25)

add \$s0, \$t0, \$t1

addi \$s1, \$s0, 36

Subtração de inteiros

sub \$t0, \$t1, \$t2 # t0 = t1 - t2

subi \$t0, \$t1, 15 # t0 = t1 - 15

.text

li \$t0, 75

li \$t1, 25 → addi \$t2, \$zero, 25
(-25)

sub \$t2, \$t0, \$t1

subi \$t3, \$t2, 10
(-10)

Multiplicação de inteiros

mul \$s0, \$t0, \$t1 # s0 = t0 * t1

↳ resultado em um inteiro

.text

li \$t0, 12

addi \$t1, \$zero, 10

mul \$s0, \$t0, \$t1

li \$v0, 1

move \$a0, \$s0

syscall

Multiplicar por potências de dois

- Basta realizar a operação de shift left que significa mover os bits para a esquerda
- Se movermos os bits de um número binário uma casa para a esquerda, multiplicaremos por 2.

$$2 = 4$$

:

$$n = 2^n$$

sll \$t0,\$t1,n # faz shift left de t1 n casas para a esquerda

.text

li \$t0,12

addi \$t1,\$zero,10

sll \$s1,\$t1,10 # multiplicar 10 por 2^{10} .

↳ mais barato

Divisão de inteiros

div \$t0,\$t1 # realiza a divisão inteira t0/t1

a parte inteira vai para t0

o resto vai para t1

mflo \$s0 # move o conteúdo de t0 para s0.

mfhi \$s1 # move o conteúdo de t1 para s1

.text

li \$t0,32

li \$t1,5

div \$t0,\$t1;

mflo \$s0

mfhi \$s1

shift right

srl \$s2,\$t0,2 # 32/4

srl \$s2,\$t1,2

↳ parte inteira

Leritura de inteiros

· li \$v0,5

.data

saudacao: .asciz "Fomiga sua idade:"

saida: .asciz "Sua idade é: "

.text

li \$v0,4

la \$a0,saudacao

syscall

li \$v0,6

syscall

move \$t0,\$v0

li \$v0,4

la \$a0,saida

syscall

li \$v0,1

move \$a0,\$t0

syscall

Leritura de strings

· Em .data, declararmos a string e seu tamanho

- usaremos .space para o número de bytes

.text

- realizaremos a instrução li \$v0,8

- la \$a0, < variável RAM >

- la \$a1, < numeroBytesLidos >

- Ao dar syscall, o valor lido fica armazenado em \$a0

.data

pergunta: .asciiz "Qual é o seu nome? "

saudacao: .asciiz "Olá, "

nome: .space 25

.text

#impressão da pergunta

li \$v0,4

la \$a0,pergunta

syscall

#leitura do nome

li \$v0,8

la \$a0,nome

la \$a1,25

syscall

#mostra a saudação

li \$v0,4

la \$a0,saudacao

syscall

#impressão do nome

li \$v0,4

la \$a0,nome

syscall

Pontos flutuantes

- Números representados em casas decimais
- float
 - 32 bits
 - 1 registrador
- double
 - 64 bits
 - 2 registradores
- \$f0 a \$f31 : co-processador 1

```
.data
    PI: .float 3.141592
.text
    li $v0, 2 #sistema, prepare-se pra imprimir um float
    lwcl $f12, PI #no caso dos float, os registradores estão
    #no co-processador 1 (cp1)
    #SEMPRE devemos colocar o valor do float em $f12, ou
    #o valor correto não é impresso
    syscall
```

Para ler:

```
li $v0, 6
syscall
```

```
.data
    msg: .asciiz "Forneça um número decimal: "
    zero: .float 0.0
.text
    #imprimindo mensagem para o usuário
    li $v0, 4
    la $a0, msg
    syscall

    #lendo o número
    li $v0, 6
    syscall #valor lido estará em $f0

    lwcl $f1, zero
    add.s $f12, $f1, $f0

    #imprimindo o número
    li $v0, 2
    syscall
```

- Double

- registradores parecidos
- valor fica em \$f0

```
.data
    msg: .asciiz "Forneça um número decimal: "
    zero: .double 0.0
.text
    #imprimindo mensagem para o usuário
    li $v0, 4
    la $a0, msg
    syscall

    #lendo o número
    li $v0, 7
    syscall #valor lido estará em $f0

    ldc1 $f2, zero
    add.d $f12, $f2, $f0

    #imprimindo o número
    li $v0, 3
    syscall
```

Condicionais

COMANDOS CONDICIONAIS

Comando	Significado	Pronúncia
beq \$t1, \$t2, label	Se \$t1 for igual a \$t2, execute a partir do rótulo label	branch if equal
bne \$t1, \$t2, label	Se \$t1 for diferente de \$t2, execute a partir do rótulo label	branch if not equal
blt \$t1, \$t2, label	Se \$t1 for menor que \$t2, execute a partir do rótulo label	branch if less than
bgt \$t1, \$t2, label	Se \$t1 for maior que \$t2, execute a partir do rótulo label	branch if greater than
ble \$t1, \$t2, label	Se \$t1 for menor ou igual a \$t2, execute a partir do rótulo label	branch if less or equal
bge \$t1, \$t2, label	Se \$t1 for maior ou igual a \$t2, execute a partir do rótulo label	branch if greater or equal

Escreva um programa que informa se um número é par ou ímpar

```
.data
    msg: .asciiz "Forneça um número: "
    par: .asciiz "O número é par."
    impar: .asciiz "O número é ímpar."

.text
    #imprimindo mensagem para o usuário
    li $v0, 4
    la $a0, msg
    syscall

    #ler o número
    li $v0, 5
    syscall

    li $t0, 2
    div $v0, $t0

    mfhi $t1 #possui o resto da divisão por 2

    beq $t1, $zero, imprimePar
    li $v0, 4
    la $a0, impar
    syscall
    imprimePar:
    li $v0, 4
    la $a0, par
    syscall
```

* não existe else.

laços de repetição while

Os laços de repetição em Assembly são combinações de Ifs e jumps. Para implementar um loop, teremos pelo menos dois rótulos: um para manter no loop, e outro para sair dele.

Ex:

while:

#comandos

saída:

#comandos

```
.text
    move $t0, $zero #$t0 será o iterador i

    while:
        beq $t0, 10, saída
        addi $t0, $t0, 1
        j while
    saída:
        #imprime o valor de i
        li $v0, 1
        move $a0, $t0
        syscall
```

```
.text
#ler o numero
li $v0, 5
syscall

move $t0, $v0 #valor lido

move $t1, $zero

laco:
    bgt $t1, $t0, lcaFora

    #imprimir $t1
    li $v0, 1
    move $a0, $t1
    syscall

    #imprimir espaço em branco
    li $v0, 4
    la $a0, espaco
    syscall

    addi $t1, $t1, 1
    j laco

espaco: .space 1
```