Assembly RISC-V

Como Programar Nisso?

Introdução

github.com/thaleslim/aulas

“Quem você acha que é”

Thales Menezes

- Bacharelando em Ciência da Computação (17.1)
- Estudante ávido de Python 3 e Assembly’s (16.2/17.1 - ...)
- Tutor de Introdução à Ciência da Computação (18.2 - ...)
- Monitor de Introdução à Sistemas Computacionais (19.1 - ...)

“Para que Assembly?”

Mitos

- Assembly é difícil de ler/escrever/aprender/depurar

- Com compiladores tão bons, aprender Assembly é desnecessário

- A era do assembly já passou, hoje em dia recursos* não faltam

- Assembly não é portátil

* Memória para processamento e espaço em disco.

“Para que Assembly?”

- Assembly é rápido

- Assembly é compacto

- Assembly permite que tirar melhor proveito do hardware*

- Assembly ajuda a entender melhor os programas e abstrações

“Porque RISC-V?”

“O RISC-V é uma ISA recente, iniciada do zero, minimalista e aberta, informada por erros de ISA anteriores”

- ISA recente e aberta (*Free & Open Software*)

- Atual, academicamente falando (MIPS 1985 | RISC-V 2010/11)

- Legível (dentro do seu contexto)

- Arquitetura Modular (RV32I)
  Vs Incremental (Compatibilidade Retroativa)

- Veloz a baixo custo

RISC vs CISC

RISC:
Reduced Instruction Set Computer

- Processador com poucas instruções
- Apenas instruções simples
- “Rápidas” e “Compactas”
- Ex.: ARM, MIPS, RISC-V

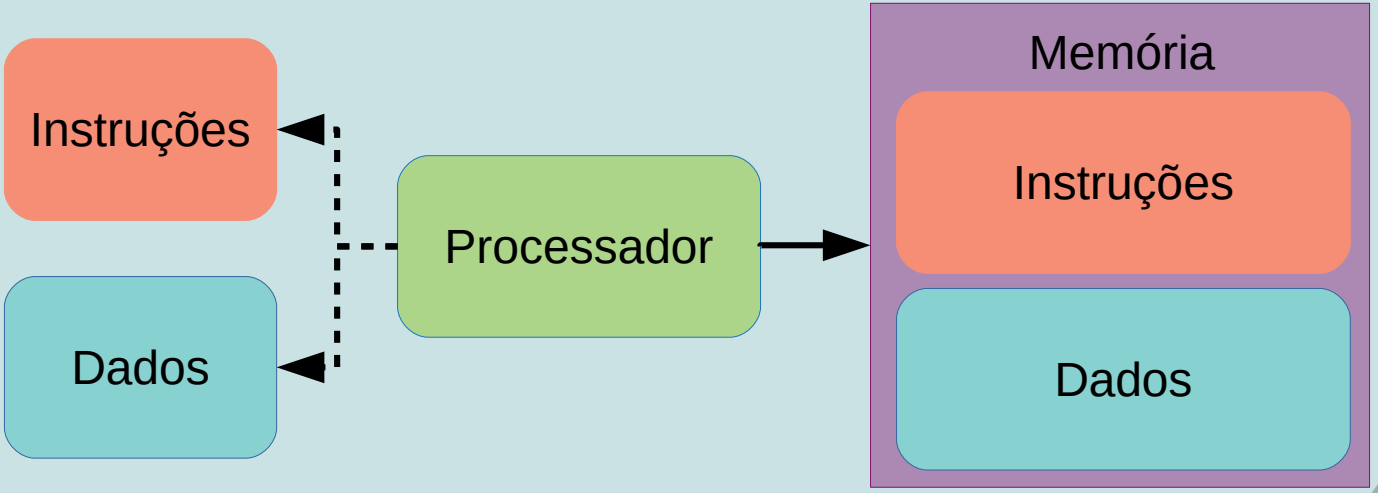
CISC:
Complex Instruction Set Computer

- Processador com muitas instruções
- Contém instruções simples e complexas
- “Lentas” e “Grandes”
- Ex.: x86, x64

Harvard vs Programa Armazenado

Harvard
(Como a faculdade)

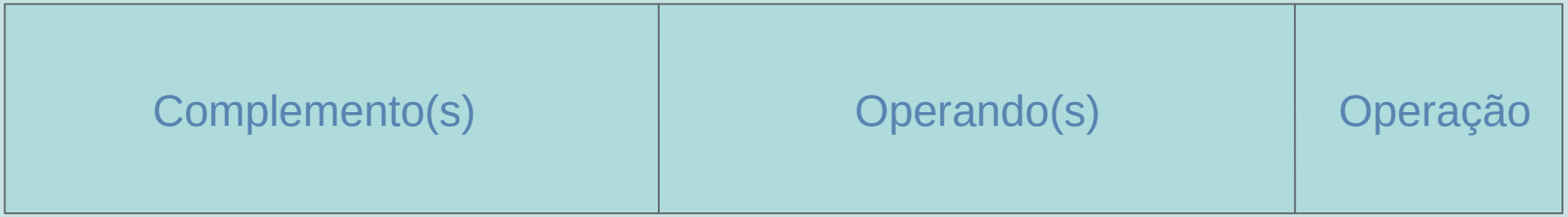
Programa Armazenado
(Equipe Prof. Von Neumann)



Comandos

Um programa pode ser visto como uma série de comandos armazenados na memória, sendo indistinguíveis de um dado.

Como representar um comando?



Glossário

- Labels:

  Nomes para endereços da memória (Ponteiros)

- Diretivas:

  Palavras reservadas usadas na tradução e montagem do código; não existem para o processador.

Módulo 1:
Sintaxe Primitiva

Principais Blocos

- Defines
- Dados
- Instruções

"Uma linguagem não é considerada grandiosa somente pela beleza da sua implementação, mas, principalmente, pela riqueza da sua documentação"

Rodrigo Bonifácio

Dentro do ambiente RARS,
pressione F1 para abrir a ajuda

Armazenar Dados

- Imbutido na Instrução (Valor Imediato):
  Acesso rápido, melhor otimização possível.

- Registradores:
  Acesso rápido; porém quantidade limitada.

- Memória de Dados (RAM):
  Maior, porém mais lenta.

Banco de Registradores

Os principais

- zero: valor constante 0 (*ground*)
- t0~t6: registradores temporários
- s0~s11: registradores salvos
- a0~a7: argumentos/resultados de funções (a0 e a1)
- ra: endereço de retorno (*return address*)
- sp: ponteiro da pilha (*stack pointer*)

"Se todo mundo respeitar a convenção o projeto sai sem complicação"

```
addi t0, t0, 4
# t0 = t0 + 4
```

Dados

Declarar variáveis e outras informações

- **.data <address>**

  Área “reservada” para dados

  Simboliza que os valores à partir dele estarão inseridos no primeiro endereço disponível na área de dados (default) ou <address> (argumento opcional).

Dados

Declarar variáveis e outras informações

```
int num;
int t0 = 10;
```


Dados

Declarar variáveis e outras informações

```
.data
texto:  .string      “Hello World”
hum:    .word        0xFFFFFFFF  # 32 bits
meio:   .halfword    0xFFFF      # 16 bits = ½ .word
char:   .byte        0xFF        #   8 bits = 1 byte
# Podemos colocar mais de um valor do mesmo tipo, separando
# com vírgulas ou espaços
numbers:   .word  4, 8 12        # numbers[] = { 4, 8, 12 }
# Podemos deixar um “espaço vazio”
void:         .space 20  # 20 bytes reservados
fpoint:       .float   4.0
big_fpoint: .double 8.25
```

<label> <diretiva> valor

Dados

Declarar variáveis e outras informações

- **Diretivas:**

.string e .asciz	→ sequência de characteres + null
.word	→ número inteiro de 32 bits
.halfword	→ número inteiro de 16 bits (½ .word)
.byte	→ número inteiro de 8 bits (1 byte)
.float	→ número de ponto flutuante de precisão simples
.double	→ número de ponto flutuante de precisão dupla
.space <bytes>	→ reserva um número de bytes

Dados

Declarar variáveis e outras informações

```
int numbers[][] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```


Dados

Declarar variáveis e outras informações

Solução 1: vetor

```
.data
# matriz: numbers[][] = { {1,2,3}, {4,5,6}, {7, 8, 9} }

numbers:    .word  1, 2, 3      # numbers[0] = { 1, 2, 3 }
                   4, 5, 6      # numbers[1] = { 4, 5, 6 }
               .word  7  8  9   # numbers[2] = { 7, 8, 9 }

# Acesso numbers[x][y]
#                          &numbers + ( y*largura + x )
```

Dados

Declarar variáveis e outras informações

Solução 2: ponteiros

```
.data
# matriz: numbers[][] = { {1,2,3}, {4,5,6}, {7, 8, 9} }

numbers_0:   .word  1  2  3      # numbers[0] = { 1, 2, 3 }

numbers_1:   .word  4  5  6      # numbers[1] = { 4, 5, 6 }

numbers_2:   .word  7  8  9      # numbers[2] = { 7, 8, 9 }

numbers:     .word number_0, numbers_1, numbers_2
```


Instruções

Compõem a lógica do programa

- **.text <address>**

  Área “reservada” para instruções do processador

  Simboliza que os valores à partir dele estarão inseridos no primeiro endereço disponível na área de instruções (default) ou <address> (argumento opcional).

Instruções

Compõem a lógica do programa

```
num = 4
t0 = num
t1 = 0 + 10
t0 += t1
a0 = 0 + t0
print(a0)
```

Instruções

Compõem a lógica do programa

```
int num = 4;

int main(){

        int t0 = &num;
        t0 = * (int *) t0;
        int t1 = 0 + 10;
        t0 += t1;
        int a0 = 0 + t0;
        printf(“%d”, a0);

        return 0;
}
```

Instruções

Compõem a lógica do programa

```
.data
num:  .word 4

.text
      la t0, num              # t0 = &num
      lw t0, (t0)             # t0 = num = 4
      addi  t1, zero, 10      # t1 = 0 + 10 = 10
      # li t1, 10
      add  t0, t0, t1         # t0 = 4 + t1
      add  a0, zero, t0       # a0 = 0 + t0 = t0
      # mv a0, t0
      li  a7, 1               # a7 = 1 NOTE: pseudoinstrução
      ecall                   # interrupção de sistema
```

operação registradorRetorno, [operandos e valores]

Instruções

Compõem a lógica do programa

- **Soma e Consulta à Memória:**

add t0,t1,t2 → t0 = t1 + t2 (Add operation)

addi t0,t1, Imm → t0 = t1 + Imm (Add with Immediate)

la t0, label → t0 = &label; (Load Address)

Carrega o endereço apontado por label para o registrador

lw t0, Imm (t1) → t0 = mem[t1 + Imm] (Load Word)

Carrega, em t0, o valor contido no endereço de t1 + Immediate

Immediate (Imm) = Número inteiro de 12 bits (complemento 2)

Instruções

Compõem a lógica do programa

- **Pseudoinstruções :**

Comandos que não são compreendidos pelo processador, são traduzidas para instruções válidas durante a compilação.

li t0, Imm → t0 = Imm (Load Immediate) → addi t0, zero, Imm

mv t0, t1 → t0 = t1 (Move) → add t0, t1, zero

Instruções

Compõem a lógica do programa

- **Interrupções de sistema:**

  Temos à disposição alguns serviços de sistema implementados:

  Para executar esses serviços precisamos:

  - Carregar o número do serviço desejado no registrador a7
  - Carregar os argumentos necessários (a0,a1,...)
  - Instrução ecall → chamada/interrupção do sistema
  - Recuperar resultados, caso existam

Instruções

Compõem a lógica do programa

- **Interrupções de sistema:**
  - Ler e imprimir valores através do RUN I/O (“Terminal”)
  - Encerrar o programa
  - Pausar a execução do programa
  - Tocar notas musicais
  - Manipular arquivos
  - Gerar números aleatórios

```
.text
        li a0, 4    # a0 = 4
        li a7, 1    # código imprime int
        ecall       # system call
```

FIM Módulo 1: Exercícios

Exercícios de Fixação

1) Faça um programa que calcule a média entre dois números

Input:
2 inteiros em linhas separadas
Output:
1 único número inteiro, média arredondada para baixo

Exemplo:

INPUT	OUTPUT
2 3	2

Exercícios de Fixação

2) Faça um programa que recebe um texto, até 50 caracteres, e o imprime no RUN I/O

Input:
1 única linha de texto
Output:
1 única linha de texto

Exemplo:

INPUT	OUTPUT
oi	oi
RISC-V	RISC-V

The End ...  For now ▷

Thales Menezes

GitHub @thaleslim

Referências Bibliográficas

- RARS: RISC-V Assembler Runtime Simulator

Buscar por *Latest Release*, clicar em *Assets* e <rars>.jar

- Guia Prático RISC-V: Atlas de uma arquitetura aberta
- The Art of Assembly Language

Referências Bibliográficas

- RARS: RISC-V Assembler Runtime Simulator

Buscar por *Latest Release*, clicar em *Assets* e <rars>.jar

- Guia Prático RISC-V: Atlas de uma arquitetura aberta
- The Art of Assembly Language