# MONTADOR PARA UMA ARQUITETURA RISC SIMPLES

## AVISO

**Prazos de entrega:** O montador entregue no dia 24/07/2019 (quarta-feira), até às 10h, terá um fator de multiplicação 1,0x, a entrega no dia 25/07/2019 (quinta-feira), até às 10h, terá um fator de multiplicação 0,6x, e o montador entregue no dia 26/07/2019 (sexta-feira), até às 10h, terá um fator de multiplicação 0,3x. Não será aceito entrega após as 10h do dia 26/07/2019 (sexta-feira).

### 1. RESUMO

Este projeto tem como objetivo o desenvolvimento de um montador (*assembler*) para uma linguagem de montagem de uma CPU RISC simples. Um montador é uma ferramenta que converte código em linguagem de montagem (*assembly*) para código em linguagem de máquina e produz o código objeto como saída. O código objeto é um arquivo binário que representa o mapa de memória que será utilizado em um simulador[i] da CPU.

### 2. FUNDAMENTAÇÃO TEÓRICA

### 2.1. ESPECIFICAÇÃO DA CPU SIMPLES

A CPU possui 32 registradores de propósito geral, cada um de 32 bits.

A Tabela 1 apresenta os 32 registradores de propósitos gerais da CPU. O registrador $zero é um *hardwired* e não permite escritas, sempre tem o valor 0 (zero).

**Tabela 1:** Registradores da CPU

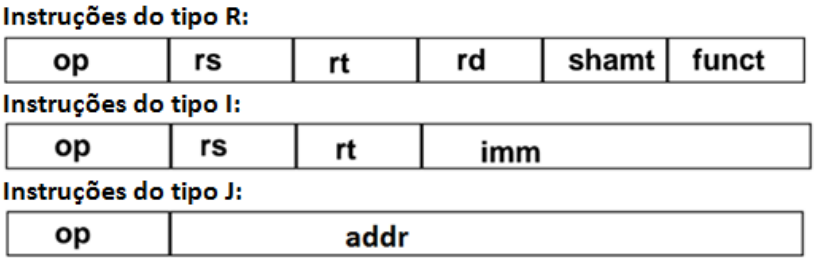
| Número | Nome | Descrição |
|---|---|---|
| 0 | $zero | Valor constante igual a zero |
| 1 | $at | Assembler temporário. |
| 2 - 3 | $v0 - $v1 | Retorno de função. |
| 4 - 7 | $a0 - $a3 | Argumento de função. |
| 8 - 15 | $t0 - $t7 | Variáveis temporárias. |
| 16 - 23 | $s0 - $s7 | Variáveis salvas por uma função. |
| 24 - 25 | $t8 - $t9 | Mais variáveis temporárias. |
| 26 - 27 | $k0 - $k1 | Temporários para o sistema operacional. |
| 28 | $gp | Apontador global. |
| 29 | $sp | Apontador de pilha. |
| 30 | $fp | Apontador de quadro. |
| 31 | $ra | Endereço de retorno. |

#### 2.1.1. FORMATO DAS INSTRUÇÕES

As instruções são divididas em três tipos: R, I e J. Cada instrução começa com um código de operação (OP) de 6 bits. Além do OP, as instruções do tipo R especificam três registradores, um campo de quantidade de mudança e um campo de função; As instruções de tipo I especificam dois registos e um valor imediato de 16 bits; As instruções do tipo J seguem o OP com um alvo de salto de 26 bits. Na Figura 1 estão os três formatos usados para o conjunto de instruções básicas.


Prof. Dr. Max Santana Rolemberg Farias
max.santana@univasf.edu.br

**Figura 1:** Formatos de instruções




**OP** – Operação (zero para o tipo R).
**Rd** – Registrador destino.
**Rs, Rt** – Registradores de operando.
**SHAMT** – Quantidade de deslocamento.
**FUNCT** – Operação a ser executada.
**IMM** – Endereço de memória/Constante numérica.
**PC** – Contador de programa.

### 2.1.2. CONJUNTO DE INSTRUÇÕES

As instruções da CPU são apresentadas na Tabela 2.

**Tabela 2:** conjunto de Instruções

| Instrução | Tipo | Opcode | Funct |
|---|---|---|---|
| **add** | R | 0 | 32 |
| **addi** | I | 8 | - |
| **sub** | R | 0 | 34 |
| **sll** | R | 0 | 0 |
| **srl** | R | 0 | 2 |
| **lui** | I | 15 | - |
| **and** | R | 0 | 36 |
| **andi** | I | 12 | - |
| **or** | R | 0 | 21 |
| **ori** | I | 13 | - |
| **xor** | R | 0 | 38 |
| **xori** | I | 14 | - |
| **slt** | R | 0 | 42 |
| **slti** | I | 10 | - |
| **beq** | I | 4 | - |
| **bne** | I | 5 | - |
| **mul** | R | 0 | 24 |
| **div** | R | 0 | 26 |
| **j** | J | 2 | - |
| **jal** | J | 3 | - |
| **jr** | R | 0 | 8 |
| **lw** | I | 35 | - |
| **sw** | I | 43 | - |

### 2.2. ESPECIFICAÇÃO DO SISTEMA DE MEMÓRIA

A memória usada no projeto do simulador possuirá palavras de 512 bits com endereçamento por 8 bits (512 x 8), isto é, cada palavra tem o tamanho de 8 bits (1 byte). Como isso, a memória do projeto tem a capacidade de armazenamento de 4K (4096 bits), e desta forma essa memória deve ter 12 bits de entradas de endereçamento ($2^{12}$ = 4096 bits).

**OBSERVAÇÂO:** As instruções e os dados serão armazenados na memória usando a estratégia *big-endian*.

# MONTADOR PARA UMA ARQUITETURA RISC SIMPLES

### 3. ESPECIFICAÇÃO DO EXECUTÁVEL

O executável do montador deve aceitar 2 (dois) argumentos, sendo que o primeiro é obrigatório e o segundo é opcional. O primeiro argumento deve conter o nome do arquivo de entrada, ou seja, o nome do arquivo que contém o programa em linguagem de montagem. O segundo argumento, facultativo, é o nome do arquivo de saída a ser gerado pelo montador. Caso o segundo argumento não seja fornecido, você deve criar uma arquivo chamado *memoria.mif*.

Dessa forma, os comandos abaixo são exemplos válidos para a execução do montador:

```
./assenbler programa.asm memoria.mif
```

Nesse primeiro comando o montador lê um arquivo denominado *programa.asm* e modifica o mapa de memória do arquivo *memoria.mif*, e no comando abaixo ele lê o arquivo *programa.asm* e cria um arquivo chamado *memoria.mif*.

```
./assenbler programa.asm
```

### 3. ESPECIFICAÇÃO DO ARQUIVO DE ENTRADA

O arquivo de entrada do montador é um arquivo texto (.asm), onde cada linha deve ter o seguinte formato:

```
[rotulo:] [instrução] [# comentário]
```

Os colchetes determinam elementos opcionais, ou seja, qualquer coisa é opcional, podendo então haver linhas em branco no arquivo de entrada, ou apenas linhas de comentários, ou linhas só com uma instrução.

**OBSERVAÇÂO:** É possível que haja múltiplos espaços em branco no início ou final da linha ou entre elementos.

### 4. ESPECIFICAÇÃO DO ARQUIVO DE SAÍDA

O arquivo de saída do montador (.mif) apresenta um mapa de memória que contém as seguintes informações:

```
WITH = 8;
DEPTH = 512;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT BEGIN
        001     : 00;
        002     : 04;
        003     : 20;
        004     : 03;
        005     : 00;
        006     : 05;
        007     : 20;
        008     : 16;
        009     : 00;
        00A     : 00;

        ...

        00C     : 53;
        00D     : 00;
        1FD     : 00;
        1FE     : 00;
        1FF     : 00;
END;
```

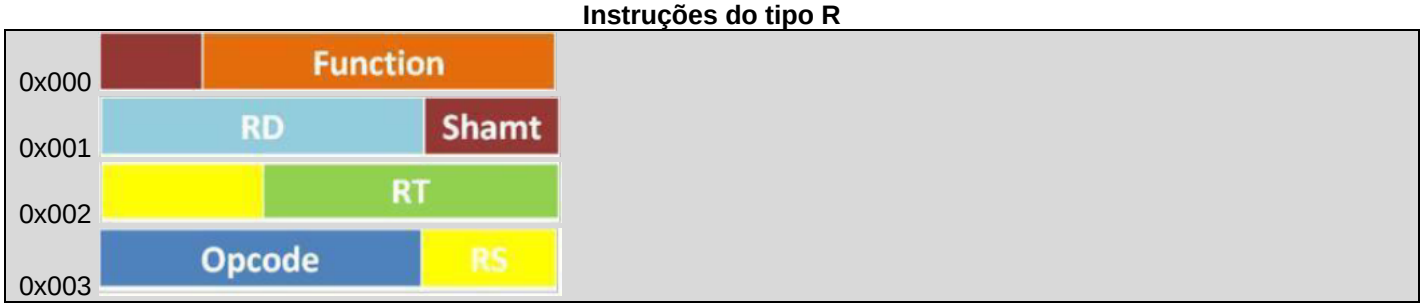
Onde, o endereço de memória é uma sequência de 3 dígitos hexadecimais. Já os dados de cada endereço de memória é uma sequência de 2 dígitos hexadecimais que representa uma palavra, conforme já visto em aula.

**OBSERVAÇÃO:** Não deve haver caracteres extras ou linhas em branco, apenas linhas no formato acima.

### 4.1. CONFIGURAÇÃO DO ARQUIVO

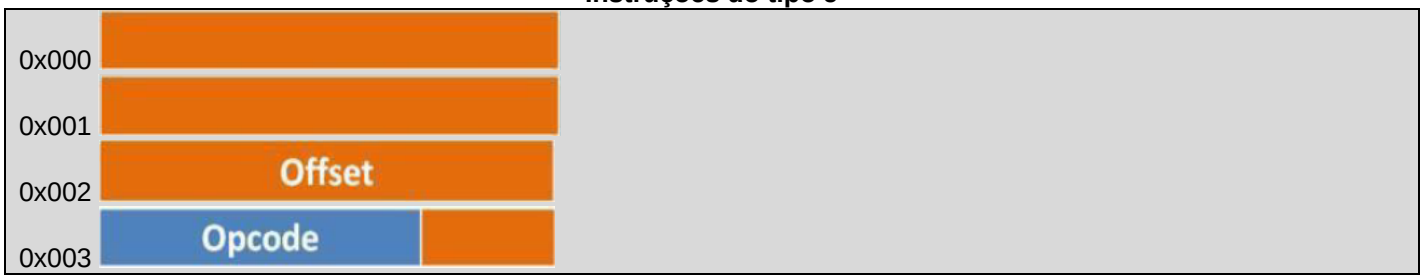
Para os testes no simulador o arquivo (.mif) deverá ser configurado com um conjunto de instruções específicas. Isso é feito com a inserção das instruções nas posições de memória que estão marcadas na margem esquerda de cada linha do arquivo (.mif). Abaixo é mostrado como dispor cada tipo de instrução nas posições de memória.

**Instruções do tipo R**




**Instruções do tipo I**


**Instruções do tipo J**




[i] O simulador da CPU simples (RISC) será desenvolvido durante o curso.