

# Montador RISC-V Assembly Simplificado

Marcus Eduardo-5779

Matheus Junio-5382

## Resumo

Neste trabalho, mostramos a conversão de um conjunto de instruções em assembly para linguagem de máquina, isto feito por linguagem de programação.

## 1 Informações Gerais

Para a montagem do montador, utilizamos a linguagem de programação Python 3, devido ao grande campo e grande facilidade de lidar com a mesma. Vale ressaltar que as respostas geradas pelo código são salvas em arquivos e também são mostradas ao usuário no terminal de execução. O montador suporta as seguintes instruções : LW, SW, ADD, XOR, ADDI, SLL, BNE, convertendo-as em linguagem de máquina.

## 2 Funcionamento

O código realiza em seu início um mapeamento de todas as instruções previamente definidas para nosso grupo, posteriormente o código utiliza algumas funções para abrir o arquivo, ler o arquivo, e a partir daí, pelo mapeamento feito no início realizar a conversão para linguagem de máquina (binária). Vale ressaltar que a conversão é feita linha por linha, ou seja, a cada linha lida no arquivo, é chamada uma função de conversão para a mesma. Posteriormente, é exibido no terminal os zeros e uns, que foram obtidos através da conversão da linguagem assembly.

```
add x2, x0, x1
sll x1, x2, x2
or x2, x2, x1
andi x2, x1, 16
addi x3, x2, -243
sw x2, 0(x1)
lw x3, 0(x2)
bne x2, x3, 8
```

Figure 1: Exemplo de código assembly.

## 3 Como executar

Para executar o arquivo, o usuário deve baixar o projeto do github(link abaixo), o usuário também deve ter instalado em seu computador o Python 3, o mesmo é capaz de rodar apenas pelo terminal, mas recomendamos uma IDE (recomendado o VScode, utilizado durante a confecção do código).

### 3.1 Executando

Na IDE ou no terminal, o usuário irá rodar o código, o mesmo pedirá todo o caminho do arquivo .asm dessa forma: `python3 tp-nacif-oc.py [arquivo.asm]`  
EX: (windows) `python C:/Users/mathe/OneDrive/workspace/tp-riscv/tp.py C:/Users/mathe/OneDrive/workspace/tp-riscv/ex1`. Ao final disso o código irá realizar a conversão.

```
PS C:\Users\mathe> python C:\Users\mathe\OneDrive\workspace\tp_riscv\tp_nacif_oc.py C:\Users\mathe\OneDrive\workspace\tp_riscv\ex1
0000000000100000000000100110011
000000000100001000100010110011
1111000011010001000000010010011
0000000001100010100001000110011
00000000010010010000010010011
000000010100001101000100010011
0000000111000011000100010010011
000000010100001101000100010011
executado em: 0.0015037059783935547 s
```

Figure 2: Exemplo execução.

### 3.2 Resultados em um arquivo

Caso o usuário digite "-f", e deverá indicar todo caminho até o arquivo onde os resultados serão armazenados, dessa forma: `python3 tp-nacif-oc.py [arquivo.asm] -f [arquivo resultado]`  
EX: (windows) `python C:/Users/mathe/OneDrive/workspace/tp-riscv/tp.py C:/Users/mathe/OneDrive/workspace/tp-riscv/ex1 -f C:/Users/mathe/OneDrive/workspace/tp-riscv/saida`.

```
PS C:\Users\mathe> python C:\Users\mathe\OneDrive\workspace\tp_riscv\tp_nacif_oc.py C:\Users\mathe\OneDrive\workspace\tp_riscv\ex1 -f C:\Users\mathe\OneDrive\workspace\tp_riscv\saida
executado em: 0.006025552749633789 s
```

Figure 3: Exemplo de colagem para armazenamento no arquivo.

### 3.3 Modo Debug

O código também conta com um modo debug, acionado quando apertado "-d" no terminal, exibindo algumas informações ao usuário, isso deverá ser digitado ao fim de qualquer operação listada acima.

```

PS C:\Users\mathe> python C:\Users\mathe\OneDrive\workspace\tp_riscv\tp_nacif_oc.py C:\Users\mathe\OneDrive\workspace\tp_riscv\ex1 -d
00000000 00001 00000 000 00010 0110011
00000000 00010 00010 001 00001 0110011
111100001101 00010 000 00011 0010011
00000000 00011 00010 100 00100 0110011
000000000001 00010 010 00001 0010011
000000001010 00011 010 00010 0010011
000000011100 00011 00010 001 0010011
000000001010 00011 010 00010 0010011
executado em: 0.0009999275207519531 s

```

Figure 4: Exemplo de debug

## 4 Pontos Importantes

### 4.1 Conversor

Função responsável por converter cada linha do arquivo para linguagem de máquina, sendo uma das principais funções do programa, será possível visualizar toda a função no link ao fim do arquivo.

```

def processarLinhaPrincipal(linha, debug=False, extended=False):
    if linha == "":
        return
    espaco = ""
    if debug:
        espaco = " "
    inst = ""
    completo = []
    res = ""
    for digito in linha:
        if digito != " " and digito != ",":
            inst += digito
        elif inst:
            completo.append(inst)
            inst = ""

```

Figure 5: Função de conversão (breve).

### 4.2 Main

É responsável por todo funcionamento do código, já que a mesma faz a abertura e leitura do arquivo, além de realizar a chamada da função de conversão para as linhas do arquivo, a mesma ainda é capaz de escrever no arquivo os resultados em um arquivo caso o usuário deseje. Segue abaixo um trecho breve da função principal.

```

def main():
    if len(sys.argv) >= 4:
        if os.path.isfile(sys.argv[3]):
            os.remove(sys.argv[3])
        with open(sys.argv[1], "w") as file:
            alll = file.readlines()
            if "-de" in sys.argv[-1]:
                for linha in alll:
                    processarLinhaPrincipal(linha.strip("\n"), True, True)
                return
            if "-d" in sys.argv[-1]:
                for linha in alll:
                    processarLinhaPrincipal(linha.strip("\n"), True)
                return
            for linha in alll:
                processarLinhaPrincipal(linha.strip("\n"))
    start = time.time()

```

Figure 6: Main.

### 4.3 Outras funções

O código contém outras funções, com papel crucial, mas que serão detalhadas de formas mais simples por aqui. A função que converte decimais em complemento de dois também se faz presente no código, assim como a função de converter os registradores, já que a função de conversão realiza apenas as de instrução.

```
def funcComplementoDeDois(numeroParametro,bits=12):  
    numeroParametro = int(numeroParametro)  
    if numeroParametro < 0:  
        return bin((1 << bits) + numeroParametro)[2:]  
    else:  
        numeroParametro = bin(numeroParametro)[2:]  
        c = bits  
        for i in numeroParametro:  
            c-=1  
        return c*'0'+str(numeroParametro)
```

Figure 7: Função de complemento de 2.

```
def reg(reg):  
    reg = int(reg.strip('x'), 16)  
    c = 5  
    for i in range(len(bin(reg)[2:])):  
        c-=1  
    return c*'0'+bin(reg)[2:]
```

Figure 8: Função de registradores.

### 4.4 Extras

Como extra, nosso código mostra no terminal o tempo de execução do mesmo, e também foi adicionada a instrução "SUB" ao conversor.

## 5 Resultados

A partir da conversão destas funções o código gera um numero binário que é exibido do terminal, e caso seja solicitado pelo usuário o mesmo será armazenado em um arquivo, finalizando assim, o que foi proposto neste trabalho prático.

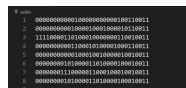


Figure 9: Exemplo de saída(arquivo).

## 6 Links

Github para visualização do código completo e seu download: [https://github.com/matheus-junio-da-silva/TP1\\_OC1\\_PYTHON](https://github.com/matheus-junio-da-silva/TP1_OC1_PYTHON).