Organição de Computadores 1

Trabalho prático

Montador MIPS – versão simplificada

ANDRÉ ELIAS, 3013
ARTHUR MARCIANO, 3019
JHORANNE AGUIAR, 2618

Florestal, setembro de 2018

SUMÁRIO

Introdução

Um montador traduz um arquivo de instruções em assembly para um arquivo de instruções de máquina binárias e dados binários.
Este trabalho tem como objetivo a aplicação prática dos conhecimentos adquiridos em sala. Ele consiste basicamente da implementação de um montador MIPS (versão simplicada). Para isto foram criadas algumas funções e utilizadas as instruções especificadas na descrição do trabalho e um código escrito na linguagem Python para colocar em prática o montador.

Descrição do Desenvolvimento

O trabalho foi implementado através da linguagem Python, uma vez que ela é uma liguagem de alto nível que, por sua vez, facilitou consideravelmente o desenvolvimento do algoritmo.

Na primeira parte do código tem-se uma função que tem como objetivo verificar qual o registrador está sendo lido e atribui-lo ao seu respectivo valor.

```
5   def retorna_reg(reg):
6       if (reg == '$szero'):
7           return '00000'
8       elif (reg == '$v0'):
9           return '00010'
10      elif (reg == '$v1'):
11          return '00011'
12      elif (reg == '$a0'):
13          return '00100'
14      elif (reg == '$a1'):
15          return '00101'
16      elif (reg == '$a2'):
17          return '00110'
18      elif (reg == '$a3'):
19          return '00111'
20      elif (reg == '$t0'):
21          return '01000'
22      elif (reg == '$t1'):
23          return '01001'
24      elif (reg == '$t2'):
25          return '01010'
26      elif (reg == '$t3'):
27          return '01011'
28      elif (reg == '$t4'):
29          return '01100'
30      elif (reg == '$t5'):
31          return '01101'
32      elif (reg == '$t6'):
33          return '01110'
34      elif (reg == '$t7'):
35          return '01111'
36      elif (reg == '$s0'):
37          return '10000'
38      elif (reg == '$s1'):
39          return '10001'
40      elif (reg == '$s2'):
41          return '10010'
42      elif (reg == '$s3'):
43          return '10011'
44      elif (reg == '$s4'):
45          return '10100'
46      elif (reg == '$s5'):
47          return '10101'
48      elif (reg == '$s6'):
49          return '10110'
50      elif (reg == '$s7'):
51          return '10111'
52      elif (reg == '$t8'):
53          return '11000'
54      elif (reg == '$t9'):
55          return '11001'
56      elif (reg == '$k0'):
57          return '11010'
58      elif (reg == '$k1'):
59          return '11011'
60      elif (reg == '$gp'):
61          return '11100'
62      elif (reg == '$sp'):
63          return '11101'
64      elif (reg == '$fp'):
65          return '11110'
66      elif (reg == '$ra'):
67          return '11111'
68      else:
69          return ''
```

Esta é a segunda função que foi utilizada para converter um número decimal para binário.

```
def converte_binario(conv, bits = 16):
    conv = int(conv)
    if (conv >= 0):
        Nbin = bin(int(conv))
        Nbin = Nbin.replace("0b", "")
        while len(Nbin) != bits:
            if len(Nbin) == bits:
                return Nbin
            Nbin = "0" + Nbin
        return Nbin
    else:
        Nbin = bin(ctypes.c_ushort(conv).value)
        Nbin = Nbin.replace("0b", "")
        return Nbin;
```

Este é um pedaço da última parte do código que, basicamente, irá conferir qual a instrução lida e chamar a função que retorna o número do registrador para cada campo do bitfield.

```
if (bitfield[0] == 'add'):
    rd = retorna_reg(bitfield[1])
    rs = retorna_reg(bitfield[2])
    rt = retorna_reg(bitfield[3])
    arq.write('000000' + str(rs) + str(rt) + str(rd) + '00000' +
    '100000'+ "\n")

elif (bitfield[0] == 'and'):
    rd = retorna_reg(bitfield[1])
    rs = retorna_reg(bitfield[2])
    rt = retorna_reg(bitfield[3])
    arq.write('000000' + str(rs) + str(rt) + str(rd) + '00000' +
    '100100'+ "\n")

elif (bitfield[0] == 'nor'):
    rd = retorna_reg(bitfield[1])
    rs = retorna_reg(bitfield[2])
    rt = retorna_reg(bitfield[3])
    arq.write('000000' + str(rs) + str(rt) + str(rd) + '00000' +
    '100111'+ "\n")

elif (bitfield[0] == 'or'):
    rd = retorna_reg(bitfield[1])
    rs = retorna_reg(bitfield[2])
    rt = retorna_reg(bitfield[3])
    arq.write('000000' + str(rs) + str(rt) + str(rd) + '00000' +
    '100101'+ "\n")
```

Conclusão

Enfim, neste trabalho implementamos um montador MIPS que foi importante para a aplicação dos conhecimentos adquiridos até então. Pode-se, também, aprender e se envolver mais com a disciplina.
Todo o processo de desenvolvimento do trabalho foi importante para poder assimiliar os conceitos, pois a aplicação dos conhecimentos aprendidos em sala foi utilizado para a realização do mesmo.