O projeto elaborado do montador possui o bojetivo de receber um código assembly e transformalo em um código binário. Este é um processo de suma importância para um sistema computacional, pois os programas de auto nível são compilados para uma línguagem de baíxo nível e depois transformado em um bytecode que pode ser por uma máquina. Este proecesso de transformação de código para números binário é feito por um montador, ou seja, o objetivo do código elaborado neste projeto.

O programa que irá realizar esta tradução foi feito na línguagem Python (Versão 3) e este foi nomeado de “assembler.py”. O programa estará recebendo um arquivo que possui uam série de instruções do mips, este outro arquivo foi nomeado “assembly_mips.m” que será o código traduzido de assembly (MIPS) para binário. O código do programa em Python possui apenas programação estruturada, visto que a utilização de classes e objetos não seria necessária, pois o paradigma orientado a objeto não afetaria no desenvolvimento/produção do projeto. O programa faz bastante uso de funções de alto nível para a manipulam strings e listas, posto que não seria possível realizar esta aplicação sem estas funções, já que lidamos com um conjuto de strings no código do arquivo “assembly_mips.m”

O código inicialmente faz uma declaração de variáveis do tipo lista (list) que irão possuir o conjunto de registradores, numeração dos registradores, instruções do tipo R, instruções do tipo I, instruções do tipo J, as functions referentes as intruções do tipo R, e os opcodes referentes as instruções do tipo I e J.

Feita as declarações essenciais do projeto, vamos ler o nosso arquivo “assembly_mips.m”. Para que seja possível realizar a leitura de todas as linhas. O código foi feito para realizar um tratamento linha por linha das instruções, então, para isso que isto funcione temos um laço (for) que irá funcinar até que chegue a última linha de código. Feita a leitura da linha do código fazemos a verificação da instrução passada para saber se ela é válida, ou seja, existe em algum conjunto dos tipos de instruções. Conforme o tipo de instrução passada, teremos que realizar um tratamento diferente, dessa forma, teremos que cuidar de uma forma específica cada tipo de instrução, seja ela R, I ou J. Basicamente as operações feitas nestes tratamentos são as mesmas, verificação de registradores, transformação para hexadecimal dos registradores, funt ou opcode referente a instrução, obter endereço da label, realizar shift left. Todas essas operações são feitas no código, mas claro que dependendo da instrução, haverá tratamentos diferentes. Ao terminar de fazer a verificação das instruções e dos registradores, ou seja, obter a function ou opcode da instrução e o número refente ao registrador, o programa irá transformar estas informações em hexadecimal. Feita está transformação será realizada a escrita dos dados (instruçõs e registradores) hexadecimais no arquivo “assembly.m”. Para cada tipo de instrução iremos escrever de uma forma diferente, coforme foi ensinado no livro “Organização e Projeto de Computadores – David A. Patterson e John L. Hennessy”. Ao terminar a escrita de uma linha do código o processo será repetido até que a última linha do código “assembly_mips.s” seja lida.

Feito uma breve explicação e descrição do projeto, podemos executar facilmente o programa. Supondo que usuário já possua o Python3 instalado em sua máquina, fazemos o comando “python3 assembler.py” para seja feita a tradução do arquivo “assembly_mips.s”. Vale lembrar que o código só irá funcionar se os arquivos se encontrarem no mesmo diretório/pasta e se não houver erros de escrita no código. Supondo que o assembly tenha sido escrito de forma errada, o programa irá gerar um código não esperado. Feito todos os passos descritos, teremos a saída do código no arquivo “assembly.m”

OBS: Não coloque um espaço no final no código “assembly.s”, pois até nesta situação o código não irá funcionar.

CÓDIGO DO MONTADOR:

```
#REGISTRADORES

registradores = ['$at', '$v0', '$v1', '$a0', '$a1', '$a3' ,'$t0' , '$t1' , '$t2', '$t3', '$t4', '$t5', '$t6', '$t7', '$s0', '$s1', '$s2', '$s3', '$s4', '$s5', '$s6', '$s7', 't8', '$t9', '$k0', '$k1', '$gp', '$sp', '$fp', '$ra']

#REPRESENTAÇÃO NUMÉRICA DOS REGISTRADORES

registradores_num = ['1', '2', '3', '4', '5', '6' ,'7' , '8' , '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31']

#TIPOS DE INSTRUÇÕES

instruções_R = ['add', 'addu', 'sub', 'subu', 'mult', 'multu', 'div', 'divu', 'mfhi', 'mflo', 'and', 'or', 'xor', 'nor', 'slt'
'sltu', 'sll', 'sllv', 'srl', 'srlv', 'sra', 'srav', 'jr', 'jalr', 'syscall']

instruções_I = ['addi', 'addiu', 'lw', 'lh', 'lhu', 'lb', 'lbu', 'sw', 'sh', 'sb', 'lui', 'ori', 'andi', 'xori', 'slti', 'sltiu', 'beq', 'bne']

instruções_J = ['j', 'jal']

#OPCODE OU FUNCT DAS INSTRUÇÕES

funct_R = ['0x20', '0x21', '0x22', '0x23', '0x18', '0x19', '0xla', '0x1b', '0x10', '0x12', '0x24', '0x25', '0x26', '0x27', '0x2a', '0x2b', '0x00', '0x04', '0x02', '0x06', '0x03', '0x07', '0x08', '0x09', '0x0c']

opcode_I = ['0x08', '0x09', '0x23', '0x21', '0x25', '0x20', '0x24', '0x2b', '0x29', '0x28', '0x0f', '0x0d', '0x0c', '0x0e', '0x0a', '0x0b', '0x04', '0x05']

opcode_J = ['0x02', '0x03']


################################################################################

#ARQUIVOS
arq_aux = open('assembly_mips.s','r')
arq = open('assembly_mips.s','r')

linhas = arq_aux.readlines()
qtd_linhas = len(linhas)

for i in range(qtd_linhas):
        linha_codigo = arq.readline()
        lista = linha_codigo.split()

        label = '1'

        #VERIFICA SE É UMA INSTRUÇÃO DO TIPO R
        for cont in range(24):
```

```
        if(lista[0] == instruções_R[cont]):
                tipo_instrução = "R"
                label = '0'

#VERIFICA SE É UMA INSTRUÇÃO DO TIPO
for cont in range(17):
        if(lista[0] == instruções_I[cont]):
                tipo_instrução = "I"
                label = '0'


#VERIFICA SE É UMA INSTRUÇÃO DO TIPO J
for cont in range(1):
        if(lista[0] == instruções_J[cont]):
                tipo_instrução = "J"
                label = '0'

#VERIFICA SE FOI PASSADO UMA LABEL
if(label == '1'):
        label = str(lista[0])
        condição_label = label.find(':')
        if(condição_label == -1):
                break
        else:
                num_label = qtd_linhas
                num_label = bin(num_label)
                continue


#TRATAMENTO DA INSTRUÇÃO TIPO R
if(tipo_instrução == 'R'):
        instrução = lista[0]
        lista.pop(0)
        lista = str(lista).strip('[]')
        lista = lista.strip('')
        lista = lista.replace("'",'')
        lista = lista.split(',')

        #VERIFICAÇÃO DE REGISTRADORES
        aux1 = 0
        aux2 = 0
        aux3 = 0

        for cont in registradores:
                if(lista[0] == cont):
                        aux1 = 1
                        break

        for cont in registradores:
                if(lista[1] == cont):
                        aux2 = 1
                        break
```

```
for cont in registradores:
	if(lista[2] == cont):
		aux3 = 1
		break


#TRATAMENTO DOS REGISTRADORES

rs = str(lista[1]) #rs
rt = str(lista[2]) #rt
rd = str(lista[0]) #rd

#DESCOBRiR NÚMERO REFERENTE A REGISTRADOR

#REFERENTE AO RS
for i in registradores:
	if(i == rs):
		index = registradores.index(i)
		break

aux = int(registradores_num[index])
rs = aux

#REFERENTE AO RT
for i in registradores:
	if(i == rt):
		index = registradores.index(i)
		break

aux = int(registradores_num[index])
rt = aux

#REFERENTE AO RD
for i in registradores:
	if(i == rd):
		index = registradores.index(i)
		break

aux = int(registradores_num[index])
rd = aux


#TRANSFORMA OS REGISTRADORES EM HEXADECIMAL
rs = bin(rs)
rt = bin(rt)
rd = bin(rd)

#ESCREVE NO ARQUIVO assembly.m
arq_hexadecimal = open('assembly.m', 'a')
arq_hexadecimal.write("00")
arq_hexadecimal.write(rs)
```

```
                arq_hexadecimal.write(rt)
                arq_hexadecimal.write(rd)
                arq_hexadecimal.write("00")
                for i in instruções_R:
                        if(instrução == i):
                                funct_index = instruções_R.index(i)
                arq_hexadecimal.write(funct_R[funct_index])
                arq_hexadecimal.write("\n")
                continue

################################################################################

        #TRATAMENTO DE INSTRUÇÃO TIPO I
        elif(tipo_instrução == 'I'):
                instrução = lista[0]

                #INSTRUÇÕES LW E SW POSSUEM UM TRATAMENTO ESPECIAL
                if(instrução == "lw" or instrução == "sw"):
                        lista.pop(0)
                        lista = str(lista)
                        lista = lista.replace(',',' ')
                        lista = lista.replace('(',")
                        lista = lista.replace(')',")
                        lista = lista.replace('[',")
                        lista = lista.replace(']',")
                        lista = lista.replace("",")
                        lista = lista.replace("",")
                        lista = lista.replace("$",' $')
                        lista = lista.split()


                        #VERIFICAÇÃO DE REGISTRADORES
                        aux1 = 0
                        aux2 = 0

                        for cont in registradores:
                                if(lista[0] == cont):
                                        aux1 = 1
                                        break


                        for cont in registradores:
                                if(lista[2] == cont):
                                        aux2 = 1
                                        break


                        #TRATAMENTO DE REGISTRADORES
                        rt = str(lista[0]) #rt
                        rd = str(lista[2]) #rd

                        #DESCOBRIR NÚMERO REFERENTE AO REGISTRADOR
```

```
#REFERENTE AO RT
for i in registradores:
	if(i == rt):
		index = registradores.index(i)
		break

aux = int(registradores_num[index])
rt = aux

#REFERENTE AO RD
for i in registradores:
	if(i == rd):
		index = registradores.index(i)
		break

aux = int(registradores_num[index])
rd = aux

#RECEBE O ENDEREÇO
adress = 4*int(lista[1])


#TRANSFORMA OS REGISTRADORES E O ENDEREÇO EM HEXADECIMAL
rt = bin(rt)
rd = bin(rd)
adress = bin(adress)

#ESCREVE NO ARQUIVO
arq_hexadecimal = open('assembly.m', 'a')
if(tipo_instrução == 'I'):
	for i in instruções_I:
		if(instrução == i):
			funct_index = instruções_I.index(i)
	arq_hexadecimal.write(opcode_I[funct_index])
	arq_hexadecimal.write(rt)
	arq_hexadecimal.write(rd)
	arq_hexadecimal.write(adress)
	arq_hexadecimal.write("\n")
	continue

#TRATAMENTO REFERENTE AS OUTRAS INSTRUÇÕES DO TIPO I
else:
	lista.pop(0)
	lista = str(lista)
	lista = lista.replace(',',' ')
	lista = lista.replace("'","")
	lista = lista.replace("'","")
	lista = lista.replace("[","")
	lista = lista.replace("]","")
	lista = lista.split()
```

```
aux1 = 0
aux2 = 0

#VERIFICAÇÃO DE REGISTRADORES
for cont in registradores:
	if(lista[0] == cont):
		aux1 = 1
		break


for cont in registradores:
	if(lista[1] == cont):
		aux2 = 1
		break

#TRATAMENTO DE REGISTRADORES
rt = str(lista[0]) #rt
rd = str(lista[1]) #rd

for i in registradores:
	if(i == rt):
		index = registradores.index(i)
		break

aux = int(registradores_num[index])
rt = aux

#REFERENTE AO RD
for i in registradores:
	if(i == rd):
		index = registradores.index(i)
		break

aux = int(registradores_num[index])
rd = aux

#CONSTANTE PASSADA PELO USUÁRIO
constant = int(lista[2])


#TRANSFORMA REGISTRADORES E CONSTANTE EM HEXADECIMAL

rt = bin(rt)
rd = bin(rd)
constant = bin(constant)

#ESCREVE NO ARQUIVO
arq_hexadecimal = open('assembly.m', 'a')
for i in instruções_I:
	if(instrução == i):
		funct_index = instruções_I.index(i)
```

```
                    arq_hexadecimal.write(opcode_I[funct_index])
                    arq_hexadecimal.write(rt)
                    arq_hexadecimal.write(rd)
                    arq_hexadecimal.write(constant)
                    arq_hexadecimal.write("\n")
                    continue

#########################################################################################

        #INSTRUÇÃO DO TIPO J
        else:
                #COMO A LABEL JÁ FOI DECLARADA BASTA ESCREVER O OPCDODE + ENDEREÇO PASSADO
                #ESCREVE NO ARQUIVO
                arq_hexadecimal = open('assembly.m', 'a')
                instrução = lista[0]
                for i in instruções_J:
                        if(instrução == i):
                                funct_index = instruções_J.index(i)
                arq_hexadecimal.write(opcode_J[funct_index])
                arq_hexadecimal.write(num_label)
                arq_hexadecimal.write("\n")
                continue


#########################################################################################
```

Exemplos de Entradas e Saídas:

assembly_mips.s
subu $t0,$s2,$a0
addi $a2,$s2,50
EXIT:
add $t0,$s0,$a0
lw $a0,20($t0)
j EXIT
beq $a0,$s0,20

assembler.m
000b100010b1000b111000x23
0x080b1110b100010b110010
000b11110b1000b111000x20
0x230b1000b1110b1010000
0x020b111
0x040b1000b11110b10100

assembly_mips.s
add $s0,$a1,$t0
multu $t3,$a1,$a0
addi $t0,$zero,100

assembler.m
000b1010b1110b1111000x20
000b1010b1000b1010000x19
0x080b1110b1110b1100100

assembly_mips.s
lw $s0,20($a0)
sw $s1,40($a1)
multu $t3,$a1,$a2
subu $t3,$zero,Ss0

assembler.m
0x230b11110b1000b1010000
0x2b0b100000b1010b10100000
000b1010b1010b1010000x19
000b10100b10100b1010000x23