Design de Computadores

Projeto 1 – Entrega Intermediária Contador

Rafael Lima

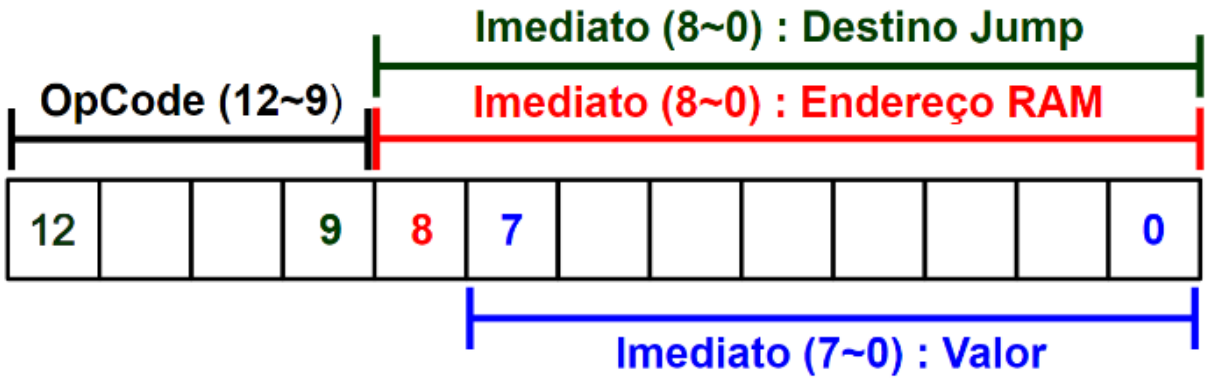
Caroline Chaim de Lima Carneiro

Arquitetura do processador

A arquitetura do processador utilizada nesse projeto é baseada em acumulador, onde o resultado de qualquer operação será sempre armazenado dentro de um único acumulador. Nesse tipo de arquitetura, as instruções possuem apenas um endereço que pode endereçar as linhas da memória RAM, as linhas da memoria ROM, ou endereçar um imediato que por sua vez será armazenado no acumulador.

Nesse projeto as instruções são compostas de 4 bits dedicados ao OpCode, onde é definido qual o tipo de operação que será usada e 9 bits dedicado ao endereço, indicando que o limite total de linhas em cada memória será $2^9 = 512$.

A imagem abaixo mostra a alocação de bits.



Formato da Instrução

Para escrita da instrução dentro da ROM, é utilizado o seguinte padrão como mostrado na imagem abaixo.

```
tmp(0) := x"0" & '0' & x"00";       -- comentário
```

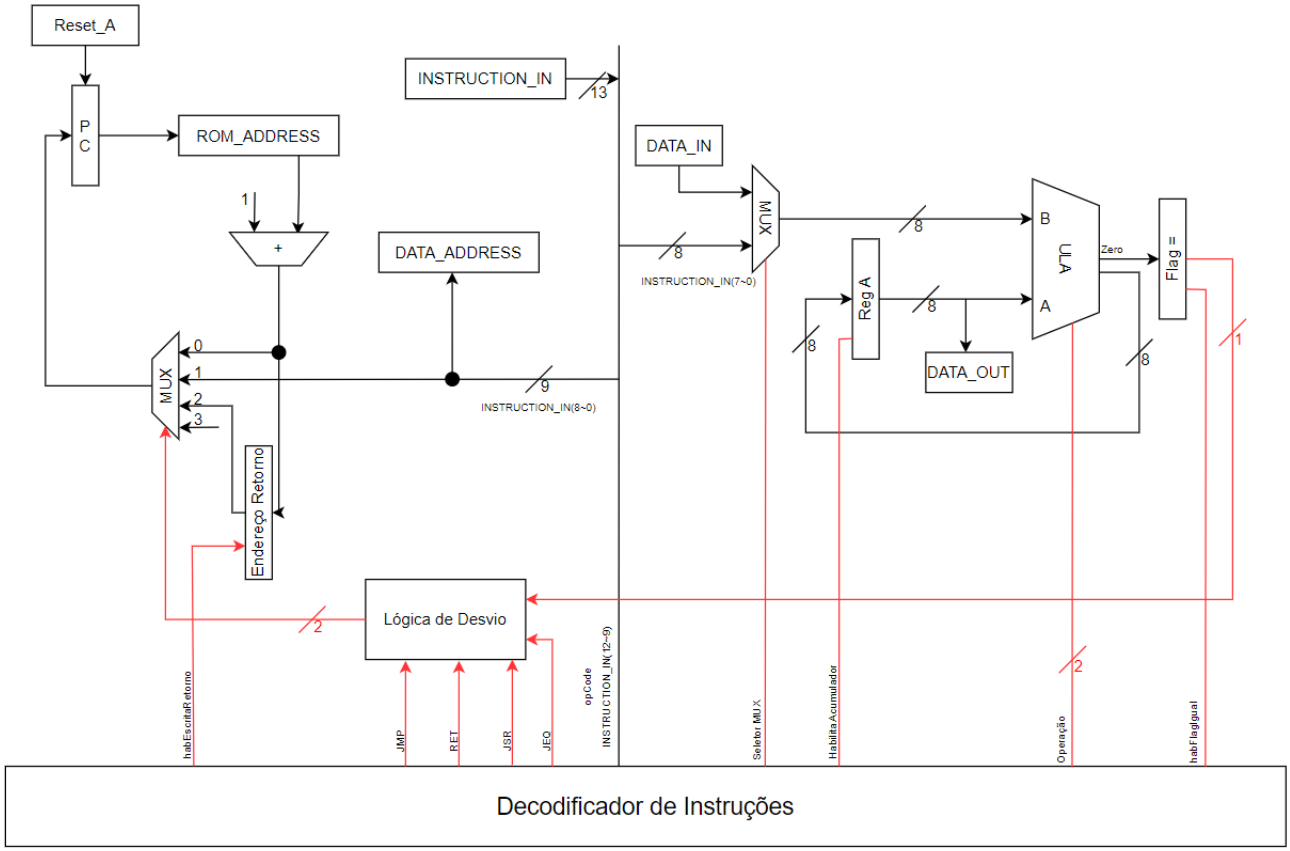
Nela conseguimos observar que tmp(x) indica em qual posição de linha a instrução está, o que é essencial para instruções como JSR, JMP e JEQ que precisam ler esse valor para aplicar sua funcionalidade.

Dentro de x”0”, como está traduzido em Hexadecimal, conseguimos escrever valores de 4 bits representando os OpCodes do nosso processador

Dentro de &”0” e x”00” escrevemos os 9 bits alocados para os endereços de memória e imediato.

Instrução	Mnemônico	Hexadecimal
Sem operação	NOP	x"0"
Carrega valor da memória, dos botões ou das chaves para o acumulador	LDA	x"1"
Soma valor do acumulador com valor da memória, e guarda no acumulador	SOMA	x"2"
Subtrai do valor do acumulador o valor da memória, e guarda no acumulador	SUB	x"3"
Carrega valor do imediato no acumulador	LDI	x"4"
Carrega valor do acumulador na memória, nos displays hexadecimais ou nos LEDs	STA	x"5"
Desvia a execução	JMP	x"6"
Desvia a execução se condição for cumprida	JEQ	x"7"
Compara valor do acumulador com valor da memória	CEQ	x"8"
Desvia a execução para sub rotina	JSR	x"9"
Retorna a execução da sub rotina	RET	x"A"

processador como um todo compõe no total de 11 componentes como mostrado na imagem abaixo.



Sobre esse fluxo, temos que levar em consideração que Reset_A, ROM_ADDRESS, INSTRUCTION_IN, DATA_ADDRESS, DATA_IN e DATA_OUT representam conexções com periféricos externos além de Rd e Wr que não foram representados no diagama, que tambem são perféricos externos bastante utilizados no nosso programa.

O mapa da memória do projeto, com as conexões dos periféricos mostrados anteriormente foi feito com base no exemplo disponibilizado no site da disciplina:

Endereço em Decimal	Periférico	Largura dos Dados	Tipo de Acesso	Bloco (Página) de Memória
0	RAM (Valor das Unidades)	8 bits	Leitura/Escrita	0
1	RAM (Valor das Dezenas)	8 bits	Leitura/Escrita	0
2	RAM (Valor das Centenas)	8 bits	Leitura/Escrita	0

3	RAM (Valor dos Milhares)	8 bits	Leitura/Escrita	0
4	RAM (Valor das Dezenas de Milhares)	8 bits	Leitura/Escrita	0
5	RAM (Valor das Centenas de Milhares)	8 bits	Leitura/Escrita	0
6	RAM (Limite das Unidades)	8 bits	Leitura/Escrita	0
7	RAM (Limite das Dezenas)	8 bits	Leitura/Escrita	0
8	RAM (Limite das Centenas)	8 bits	Leitura/Escrita	0
9	RAM (Limite dos Milhares)	8 bits	Leitura/Escrita	0
10	RAM (Limite das Dezenas de Milhares)	8 bits	Leitura/Escrita	0
11	RAM (Limite das Centenas de Milhares)	8 bits	Leitura/Escrita	0
12 ~ 17	RAM	8 bits	Leitura/Escrita	0
18	RAM (Flag Verifica Limite)	8 bits	Leitura/Escrita	0
19	RAM (Flag Inibir Contagem)	8 bits	Leitura/Escrita	0
20	RAM (Constante 0)	8 bits	Leitura/Escrita	0
21	RAM (Constante 1)	8 bits	Leitura/Escrita	0
22	RAM (Constante 10)	8 bits	Leitura/Escrita	0
23 ~ 63	RAM	8 bits	Leitura/Escrita	0
64 ~ 127	Reservado	–	–	1
128 ~ 191	Reservado	–	–	2
192 ~ 255	Reservado	–	–	3
256	LEDR0 ~ LEDR7	8 bits	Escrita	4
257	LEDR8	1 bit	Escrita	4
258	LEDR9	1 bit	Escrita	4
259 ~ 287	Reservado	–	–	4
288	HEX0	4 bits	Escrita	4
289	HEX1	4 bits	Escrita	4
290	HEX2	4 bits	Escrita	4
291	HEX3	4 bits	Escrita	4
292	HEX4	4 bits	Escrita	4
293	HEX5	4 bits	Escrita	4
294 ~ 319	Reservado	–	–	4
320	SW0 ~ SW7	8 bits	Leitura	5
321	SW8	1 bit	Leitura	5
322	SW9	1 bit	Leitura	5
323 ~ 351	Reservado	–	–	5
352	KEY0	1 bit	Leitura	5

353	KEY1	1 bit	Leitura	5
357 ~ 383	Reservado	–	–	5
384 ~ 447	Reservado	–	–	6
448 ~ 509	Reservado	–	–	7
510	Limpa Leitura KEY1	–	Escrita	7
511	Limpa Leitura KEY0	–	Escrita	7

Funcionamento do Projeto

O projeto do contador tem dois modos principais de funcionamento, progressivo e regressivo. O modo de operação é definida pelo switch 9. No modo de contagem progressiva, ativado quando o switch 9 está desligado, o Key 0 incrementa o valor do cantador em 1 a cada clique, e no modo de contagem regressiva, ativado quando o switch 9 está ligado, decrementa a contagem de um em um.

Fonte do Programa em Assembly:

O nosso codigo escrito em assembly, impementada dentro da ROM, possui no total de 146 linhas. Para auxiliar no desenvolvimento desse codigo nós modificamos um codigo criado pelo Marclo Mello para melhor nos benificiar.

```
"""
- Criado em 07/Fevereiro/2022
- Atualizado em 19/04/2023

@autor: Marco Mello e Paulo Santos
@autor_versao_modificada: Rafael Lima

Regras:

1) O Arquivo ASM.txt não pode conter linhas iniciadas com caracteres ' ' ou '\n')
2) Linhas somente com comentários são excluídas
3) Instruções sem comentário no arquivo ASM receberão como comentário no arquivo
BIN a própria instrução
"""

inputASM = 'ASM.txt' #Arquivo de entrada de contém o assembly
outputBIN = 'BIN.txt' #Arquivo de saída que contém o binário formatado para VHDL
outputMIF = 'initROM.mif' #Arquivo de saída que contém o binário formatado para
.mif

noveBits = True;
```

```
#definição dos mnemônicos e seus
#respectivo OPCODEs (em Hexadecimal)
mne =   {
       "NOP":   "0",
       "LDA":   "1",
       "SOMA":  "2",
       "SUB":   "3",
       "LDI":   "4",
       "STA":   "5",
       "JMP":   "6",
       "JEQ":   "7",
       "CEQ":   "8",
       "JSR":   "9",
       "RET":   "A",
}

#Converte o valor após o caractere arroba '@'
#em um valor hexadecimal de 2 dígitos (8 bits)
def  converteArroba(line):
    line = line.split('@')
    line[1] = hex(int(line[1]))[2:].upper().zfill(2)
    line = ''.join(line)
    return line

#Converte o valor após o caractere arroba '@'
#em um valor hexadecimal de 2 dígitos (8 bits) e...
#concatena com o bit de habilita
def  converteArroba9bits(line):
    line = line.split('@')
    if(int(line[1]) > 255 ):
        line[1] = str(int(line[1]) - 256)
        line[1] = hex(int(line[1]))[2:].upper().zfill(2)
        line[1] = "\" & '1' & x\"" + line[1]
    else:
        line[1] = hex(int(line[1]))[2:].upper().zfill(2)
        line[1] = "\" & '0' & x\"" + line[1]
    line = ''.join(line)
    return line

#Converte o valor após o caractere cifrão'$'
#em um valor hexadecimal de 2 dígitos (8 bits)
def  converteCifrao(line):
    line = line.split('$')
    line[1] = hex(int(line[1]))[2:].upper().zfill(2)
    line = ''.join(line)
```

```
    return line

#Converte o valor após o caractere arroba '$'
#em um valor hexadecimal de 2 dígitos (8 bits) e...
#concatena com o bit de habilita
def  converteCifrao9bits(line):
    line = line.split('$')
    if(int(line[1]) > 255 ):
        line[1] = str(int(line[1]) - 256)
        line[1] = hex(int(line[1]))[2:].upper().zfill(2)
        line[1] = "\" & '1' & x\"" + line[1]
    else:
        line[1] = hex(int(line[1]))[2:].upper().zfill(2)
        line[1] = "\" & '0' & x\"" + line[1]
    line = ''.join(line)
    return line

#Define a string que representa o comentário
#a partir do caractere cerquilha '#'
def defineComentario(line):
    if '#' in line:
        line = line.split('#')
        line = line[0] + "\t#" + line[1]
        return line
    else:
        return line

#Remove o comentário a partir do caractere cerquilha '#',
#deixando apenas a instrução
def defineInstrucao(line):
    line = line.split('#')
    line = line[0]
    return line

#Consulta o dicionário e "converte" o mnemônico em
#seu respectivo valor em hexadecimal
def trataMnemonico(line):
    line = line.replace("\n", "") #Remove o caracter de final de linha
    line = line.replace("\t", "") #Remove o caracter de tabulacao
    line = line.split(' ')
    line[0] = mne[line[0]]
    line = "".join(line)
    return line

def labels(lines):
```

```
    i = 0;
    unused_lines = []
    constants = {}
    labels = {}
    while i < len(lines):
        line = lines[i]
        if line.startswith("#") or line.startswith(" ") or line.startswith("\n"):
            unused_lines.append(i)
        elif "=" in line:
            const = line.split("=")[0]
            const = const.replace(' ', '')
            value = line.split("=")[1]
            value = value.replace(' ', '')
            value = value.replace('\n','')
            constants[const] = value
            unused_lines.append(i)
        elif line.endswith(":\n"):
            label = line.replace(":\n", '')
            labels[label] = i - len(unused_lines)
            unused_lines.append(i)
        i+=1

    return constants, labels, unused_lines


with open(inputASM, "r") as f: #Abre o arquivo ASM
    lines = f.readlines() #Verifica a quantidade de linhas


with open(outputBIN, "w+") as f:  #Abre o destino BIN

    cont = 0 #Cria uma variável para contagem

    #primeira passagem troca labels por numeros


    consts, labels, unused_lines = labels(lines)
    i = 0
    clean_lines = []
    while i < len(lines):
        line = lines[i]
        for const in consts:
            if const in line:
                lines[i] = line.replace(const,str(consts[const]))
        for label in labels:
```

```
            if label in line:
                lines[i] = line.replace(label,str(labels[label]))
        i+=1

    i = 0
    while i < len(unused_lines):
        line_to_remove = unused_lines[i] - i
        lines.pop(line_to_remove)
        i+=1

    print(unused_lines)

    for line in lines:
        print(line)


    for line in lines:

        #Verifica se a linha começa com alguns caracteres invalidos ('\n' ou ' '
ou '#')
        if (line.startswith('\n') or line.startswith(' ') or
line.startswith('#')):
            line = line.replace("\n", "")
            print("-- Sintaxe invalida" + ' na Linha: ' + ' --> (' + line + ')')
#Print apenas para debug

        #Se a linha for válida para conversão, executa
        else:

            #Exemplo de linha => 1. JSR @14 #comentario1
            comentarioLine = defineComentario(line).replace("\n","") #Define o
comentário da linha. Ex: #comentario1
            instrucaoLine = defineInstrucao(line).replace("\n","") #Define a
instrução. Ex: JSR @14

            instrucaoLine = trataMnemonico(instrucaoLine) #Trata o mnemonico.
Ex(JSR @14): x"9" @14

            if '@' in instrucaoLine: #Se encontrar o caractere arroba '@'
                if noveBits == False:
                    instrucaoLine = converteArroba(instrucaoLine) #converte o
número após o caractere Ex(JSR @14): x"9" x"0E"
                else:
                    instrucaoLine = converteArroba9bits(instrucaoLine) #converte
o número após o caractere Ex(JSR @14): x"9" x"0E"
```

```
            elif '$' in instrucaoLine: #Se encontrar o caractere cifrao '$'
                if noveBits == False:
                    instrucaoLine = converteCifrao(instrucaoLine) #converte o
número após o caractere Ex(LDI $5): x"4" x"05"
                else:
                    instrucaoLine = converteCifrao9bits(instrucaoLine) #converte
o número após o caractere Ex(LDI $5): x"4" x"05"

            else: #Senão, se a instrução nao possuir nenhum imediato, ou seja,
nao conter '@' ou '$'
                instrucaoLine = instrucaoLine.replace("\n", "") #Remove a quebra
de linha
                if noveBits == False:
                    instrucaoLine = instrucaoLine + '00' #Acrescenta o valor
x"00". Ex(RET): x"A" x"00"
                else:
                    instrucaoLine = instrucaoLine + "\" & " + "\'0\' & " +
"x\"00" #Acrescenta o valor x"00". Ex(RET): x"A" x"00"


            line = 'tmp(' + str(cont) + ') := x"' + instrucaoLine + '";\t-- ' +
comentarioLine + '\n'  #Formata para o arquivo BIN

                     #Entrada => 1. JSR @14 #comentario1

                     #Saída =>   1. tmp(0) := x"90E"; -- JSR @14  #comentario1

            cont+=1 #Incrementa a variável de contagem, utilizada para
incrementar as posições de memória no VHDL
            f.write(line) #Escreve no arquivo BIN.txt

            print(line,end = '') #Print apenas para debug



##############################
##############################
#Conversão para arquivo .mif
##############################
##############################
'''
with open(outputMIF, "r") as f: #Abre o arquivo .mif
    headerMIF = f.readlines() #Faz a leitura das linhas do arquivo,
                              #para fazer a aquisição do header
```

```
with open(outputBIN, "r") as f: #Abre o arquivo BIN
    lines = f.readlines() #Faz a leitura das linhas do arquivo


with open(outputMIF, "w") as f:  #Abre o destino .mif novamente
                                 #agora para preenchê-lo com o pograma

    cont = 0 #Cria uma variável para contagem

    ########################################
    #Preenche com o header lido anteriormente
    for lineHeader in headerMIF:
        if cont < 21:           #Contagem das linhas de cabeçalho
            f.write(lineHeader) #Escreve no arquivo se saída .mif o cabeçalho (21
linhas)
        cont = cont + 1         #Incrementa varíavel de contagem
   #--------------------------------------
   ########################################

    for line in lines: #Varre as linhas do código formatado para a ROM (VHDL)

        replacements = [('t', ''), ('m', ''), ('p', ''), ('(', ''), (')', ''),
('=', ''), ('x', ''), ('"', '')] #Define os caracteres que serão excluídos

        ########################################
        #Remove os caracteres que foram definidos
        for char, replacement in replacements:
            if char in line:
                line = line.replace(char, replacement)
        #--------------------------------------
        ########################################

        line = line.split('#') #Remove o comentário da linha

        if "\n" in line[0]:
            line = line[0]
        else:
            line = line[0] + '\n' #Insere a quebra de linha ('\n') caso não tenha

        f.write(line) #Escreve no arquivo initROM.mif
    f.write("END;") #Acrescente o indicador de finalização da memória. (END;)
'''
```