Design de Computadores - Insper - 2023 S2

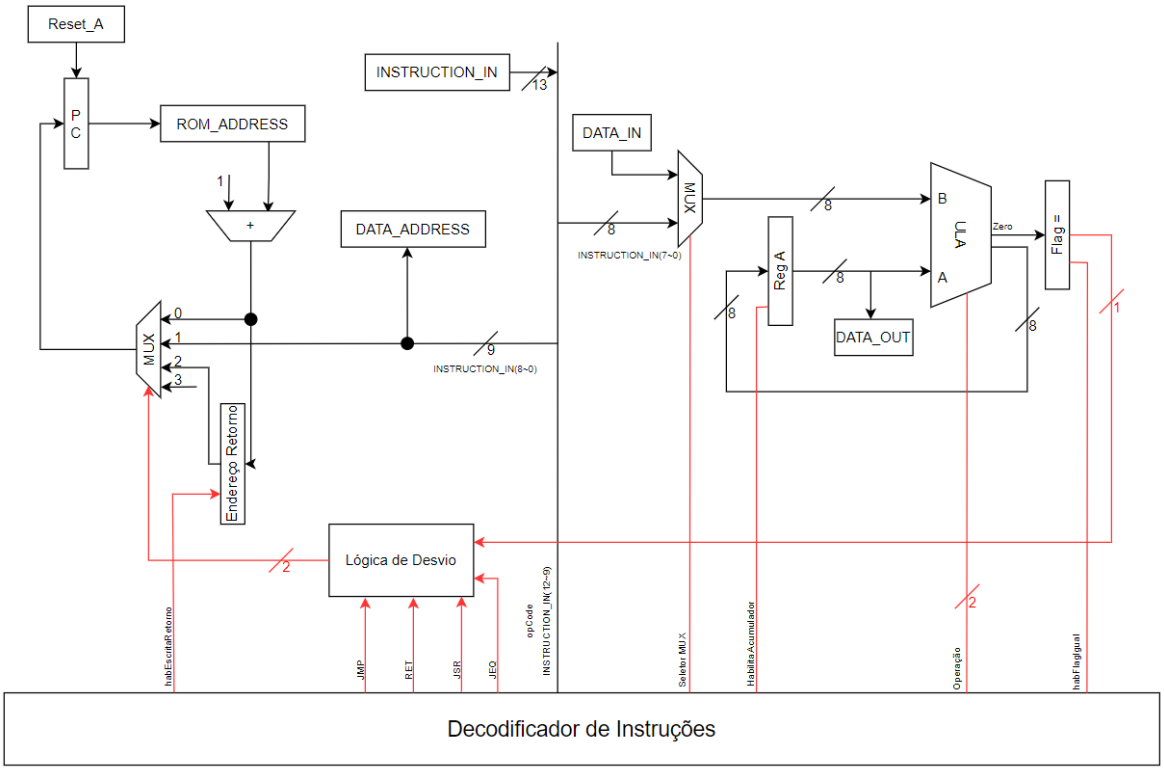
Projeto 1 – Entrega Intermediária

Rafael Lima

Caroline Chaim de Lima Carneiro

Arquitetura do processador

Nessa entrega intermediaria do cotador, escolhemos usar um processador baseada em acumuladores e memoria ram, onde o resultado de qualquer operacao e sempre armazenado em um unico acumulador e nos usamos com frequencia a nossa memoria RAM para armazenar outras informacoes. Essa arquitetura foi escolhida por ter sido trabalhada em aula.



Total de Instrucoes e sua sintexe

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

O contador desse projeto tem X linhas de instruções, sendo usados 10 tipos de instrução cuja estrutura segue o seguinte padrão:

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

Sendo que as intrucoes seguem o seguinte tipo de comportamto e definicao dentro da nosso processador

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"

Fonte do Programa em Assembly:

Por fim, segue o programa usado para converter a programação em assembly para uma linguagem que o Quartus consiga interpretar, baseado no exemplo disponibilizado por Marco Mello:

```
"""

Criado em 07/Fevereiro/2022
```

```
Modificado em 21/Abril/2023
@autor_original: Marco Mello e Paulo Santos
@autor_versão_modificada: Tiago Seixas e Caroline Chaim
Regras:
1) O Arquivo ASM.txt não pode conter linhas iniciadas com caractere ' ' ou '\n')
2) Linhas somente com comentários são excluídas e linhas vazias
3) Instruções sem comentário no arquivo ASM receberão como comentário no arquivo
BIN a própria instrução
4) O comentário deve ter o seguinte espaçamento da instrução (copie se
necessário): "            ", caso contrário ele interpreta o espaço com sendo
parte do nome da instrução
5) Exemplo de código invalido na versão modificada:
                            0.___JSR .pulo0 #comentario1            << Invalido (
o código não acha destino de pulo "pulo0 ", apenas destino "pulo0")
                            1.___JMP .pulo1         #comentario3
                            2.___pulo3: JEQ .pulo2
                            3.___pulo4: NOP
                            4.___NOP
                            5.___pulo1: LDI $5
                            6.___STA $0
                            7.___CEQ @0
                            8.___JMP .pulo3         #comentario4
                            9.___pulo2: NOP
                            10.__LDI $4
                            11.__CEQ @0
                            12.__JEQ .pulo4
                            13.__pulo5: JMP .pulo5
                            14.__pulo0: NOP
                            15.__RET
                            16.__#comentario5           << Invalido ( Linha
somente com comentário )
                            17.___                       << Invalido ( Linha
vazia )

6) Exemplo de código válido na versão modificada (Arquivo ASM.txt):
                            0.___JSR .pulo0         #comentario1
                            1.___JMP .pulo1         #comentario3
                            2.___pulo3: JEQ .pulo2
                            3.___pulo4: NOP
                            4.___NOP
                            5.___pulo1: LDI $5
                            6.___STA $0
                            7.___CEQ @0
                            8.___JMP .pulo3         #comentario4
                            9.___pulo2: NOP
```

```
                            10.__LDI $4
                            11.__CEQ @0
                            12.__JEQ .pulo4
                            13.__pulo5: JMP .pulo5
                            14.__pulo0: NOP
                            15.__RET

7) Resultado do código válido (Arquivo BIN.txt):
                            0.___tmp(0) := x"9" & '0' & x"0E";   -- JSR
.pulo0                           #comentario1
                            1.___tmp(1) := x"6" & '0' & x"05";   -- JMP
.pulo1                           #comentario3
                            2.___tmp(2) := x"7" & '0' & x"09";   -- pulo3: JEQ
.pulo2
                            3.___tmp(3) := x"0" & '0' & x"00";           --
pulo4: NOP
                            4.___tmp(4) := x"0" & '0' & x"00";           -- NOP
                            5.___tmp(5) := x"4" & '0' & x"05";   -- pulo1: LDI $5
                            6.___tmp(6) := x"5" & '0' & x"00";   -- STA $0
                            7.___tmp(7) := x"8" & '0' & x"00";   -- CEQ @0
                            8.___tmp(8) := x"6" & '0' & x"02";   -- JMP
.pulo3                           #comentario4
                            9.___tmp(9) := x"0" & '0' & x"00";           --
pulo2: NOP
                            10.__tmp(10) := x"4" & '0' & x"04";  -- LDI $4
                            11.__tmp(11) := x"8" & '0' & x"00";  -- CEQ @0
                            12.__tmp(12) := x"7" & '0' & x"03";  -- JEQ .pulo4
                            13.__tmp(13) := x"6" & '0' & x"0D";  -- pulo5: JMP
.pulo5
                            14.__tmp(14) := x"0" & '0' & x"00";          --
pulo0: NOP
                            15.__tmp(15) := x"A" & '0' & x"00";          -- RET
"""



assembly = 'ASM.txt' #Arquivo de entrada de contem o assembly
destinoBIN = 'BIN.txt' #Arquivo de saída que contem o binário formatado para VHDL

#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",
}

def entradaSalto(line):
    line = line.split('.')
    destino = line[1]
    destino = destino.replace("\t", "")
    return destino

def achaSaidaSalto(line):
    line = line.split(':')
    destino = line[0]
    return destino

def mudaSaidaSalto(line):
    line = line.split(':')
    del line[0]
    line[0] = line[0][1:]
    return line[0]

#Converte o valor após o caractere arroba '@'
#em um valor hexadecimal de 2 dígitos (8 bits)
def  converteArroba(line):
    line = line.split('@')
    line.append(line[1])
    line[1] = int(line[1])
    if line[1] < 256 :
        line[1] = "'0'"
        line[2] = hex(int(line[2]))[2:].upper().zfill(2)
    else:
        line[1] = "'1'"
        line[2] = hex(int(line[2])-256)[2:].upper().zfill(2)
    line0 = "{}".format(line[0])
    line1 = line[1]
    line2 = line[2]
    #line = ''.join(line)
    return line0, line1, line2
```

```
#Converte o valor após o caractere cifrão'$'
#em um valor hexadecimal de 2 dígitos (8 bits)
def  converteCifrao(line):
    line = line.split('$')
    line.append(line[1])
    line[1] = int(line[1])
    if line[1] < 256 :
        line[1] = "'0'"
        line[2] = hex(int(line[2]))[2:].upper().zfill(2)
    else:
        line[1] = "'1'"
        line[2] = hex(int(line[2])-256)[2:].upper().zfill(2)
    line0 = "{}".format(line[0])
    line1 = line[1]
    line2 = line[2]
    #line = ''.join(line)
    return line0, line1, line2

#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
```

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


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

    dicionario_labels = {}

    cont = 0 #Cria uma variável para contagem

    cont2 = 0 #Cria uma segunda variável para contagem

    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", "")

        #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

            if ':' in instrucaoLine:
                destino = achaSaidaSalto(instrucaoLine)
                dicionario_labels[destino] = str(cont2)

            cont2+=1 #Incrementa a segunda variável de contagem, utilizada para
incrementar adicionar os labels ao dicionario

    #print(dicionario_labels) #imprime o dicionario para checar

    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

            if ':' in instrucaoLine:
                instrucaoLine = mudaSaidaSalto(instrucaoLine)

            if '.' in instrucaoLine:
                destino = entradaSalto(instrucaoLine)
                str_destino_replace = '.'+destino
                str_destino_nova = '@'+dicionario_labels[destino]
                instrucaoLine = instrucaoLine.replace(str_destino_replace,
str_destino_nova)

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

            if '@' in instrucaoLine: #Se encontrar o caractere arroba '@'
                instrucaoLine0, instrucaoLine1, instrucaoLine2 =
converteArroba(instrucaoLine) #converte o número após o caractere Ex(JSR @14):
x"9" x"0E"
                line = 'tmp(' + str(cont) + ') := x"' + instrucaoLine0 + '" & '+
instrucaoLine1 +' & x"' + instrucaoLine2 + '";\t-- ' + comentarioLine +
'\n'  #Formata para o arquivo BIN

                    #Entrada => 1. JSR @14 #comentario1

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

            elif '$' in instrucaoLine: #Se encontrar o caractere cifrao '$'
                instrucaoLine0, instrucaoLine1, instrucaoLine2 =
converteCifrao(instrucaoLine) #converte o número após o caractere Ex(LDI $5):
x"4" x"05"
                line = 'tmp(' + str(cont) + ') := x"' + instrucaoLine0 + '" & '+
instrucaoLine1 +' & x"' + instrucaoLine2 + '";\t-- ' + comentarioLine +
'\n'  #Formata para o arquivo BIN
```

```
                    #Entrada => 1. JSR @14 #comentario1

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

            else: #Senão, se a instrução nao possuir nenhum imediator, ou seja,
nao conter '@' ou '$'
                instrucaoLine = instrucaoLine.replace("\n", "") #Remove a quebra
de linha
                #instrucaoLine = instrucaoLine + '00' #Acrescenta o valor x"00".
Ex(RET): x"A" x"00"
                bit_0 = "'0'"
                line = 'tmp(' + str(cont) + ') := x"' + instrucaoLine + '" & '+
bit_0 +' & x"00";        \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
```

Fontes externas utilizadas:

O site da disciplina, que pode ser acessado selecionando Atividades na aba Conteúdos de Design de Computadores, no Blackboard.

O projeto de Assembler disponibilizado no github por Marco Mello:
https://github.com/Insper/DesignComputadores/tree/master/AssemblerASM_BIN_VHDL