FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA
Departamento de Engenharia Electrotécnica

Trabalho Final

Sistemas Lógicos II

Turno P6

13/06/2022

Alunos:

Isaac Matos Furtado Nº 62884

Tiago Miguel Duarte Mendes Nº 63352

Tiago Capelo Monteiro Nº 63368

Índice

1. Introdução 7
2. Descrição dos requisitos 4
3. Apresentação da arquitectura implementada 7
Apresentação de árvore de ficheiros schematics do microcomputador 9
Detalhes de cada componente do módulo 10
Microcomputador 10
ram_rom256 11
Dispo 84 e 52 12
Dispo 115 14
CPU 15
Controlador 17
NCCPU 19
Stack pointer 21
Instruction register 22
Flag register 23
ALU(Arithmetic and logic unit) 24
ALUFlags 26
Portas lógicas 28
Portas lógica ANDS 29
Portas lógicas ORS 30
Portas lógicas XORS 31
Detalhes de registros(A, B, C, D, PC, MAR, TMP) 32
A, B, C, D, PC, MAR, TMP 32
CL_1 33
Detalhes do buffer e buffer8 33
5. Testes e Resultados 34
Teste CLI 35
Teste ALU 35

Teste Dispositivos 82 , 54 37
Teste NCCPU (No controller CPU) e SP 38
Teste do dispositivo 115 39
Teste CPU 39
6. Conclusões 40

1. INTRODUÇÃO

Este trabalho consiste na criação de um microcomputador contendo as memórias ROM e RAM, tendo cada uma ela 128 posições de 1 byte, um dispositivo externo(115) que funciona como interruptor do hardware e outros dois denominados X(84) e Y(52) a enviar e receber informação.

Estes componentes são todos ligados a um microprocessador capaz de interpretar e realizar um conjunto de instruções dadas.

Objetivo deste relatório é observarmos e comentamos sobre o funcionamento de um microprocessador desde da criação de registos básicos até a um componente complexo, capaz de controlar os processos de um computador.

Para tal, o trabalho encontra-se dividido em 5 partes:

1. Identificamos os requisitos necessários para o nosso trabalho, indicando o funcionamento de casa e a sua importância para o sistema
2. Neste ponto, iremos detalhar a arquitetura implementada no sistema e realizar um breve discussão sobre o assunto de quais vantagens e desvantagens temos de o realizar
3. Para simplificar o trabalho, procedemos à criação de diversos módulos que dividem o nosso CPU, aqui iremos explicar e descrever cada.
4. Testagem do trabalho, criar o trabalho e o mesmo não funcionar seria um problema, assim com base nos módulos criados podemos testar individualmente cada parte do trabalho e resolver problemas de maneira rápida e eficaz.
5. Parte Final do trabalho, a conclusão, onde fazemos um breve resumo do projeto, comentado o que aprendemos com o mesmo e o que fomos capazes de realizar indicando as maiores dificuldades da sua realização.

2. DESCRIÇÃO DOS REQUISITOS

Neste capítulo apresenta-se a descrição dos requisitos necessários à realização do trabalho, as funções necessárias e as instruções que devem ser dadas ao microprocessador para as realizar e os requisitos necessários para as poder realizar.

Usando como base o processador 8086 da Intel, iremos construir este processador através de um programa de montagem de aparelhos lógicos/digitais.

O usado para este caso foi o xilinx 9.2i, foram construídos um cpu(seus componentes), 2 periféricos e adicionadas duas memórias(RAM e ROM), ao quais precisamos dar instruções para possibilitar a realização de tarefas do processador que enviará comandos de controlo para os outros dispositivos e é capaz de realizar operações aritméticas e lógicas dentro do mesmo, usando um componente denominado ALU.

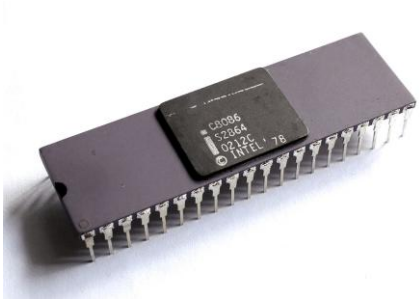


Figura 1- Processador 8086

Precisaremos também de ter atenção ao espaço pretendido para este projeto(8 Bits) e os cuidados a ter para a criação do mesmo os registos devem ser de 8 bits, o fios capazes de levar 8 bits de informação ALU realizar contas de 8 bits, memória adequada para 8 bits , adicionar um controlador capaz de verificar erros e iniciar interrupt quando necessário e controlar a troca de memória entre componentes do Microprocessador.

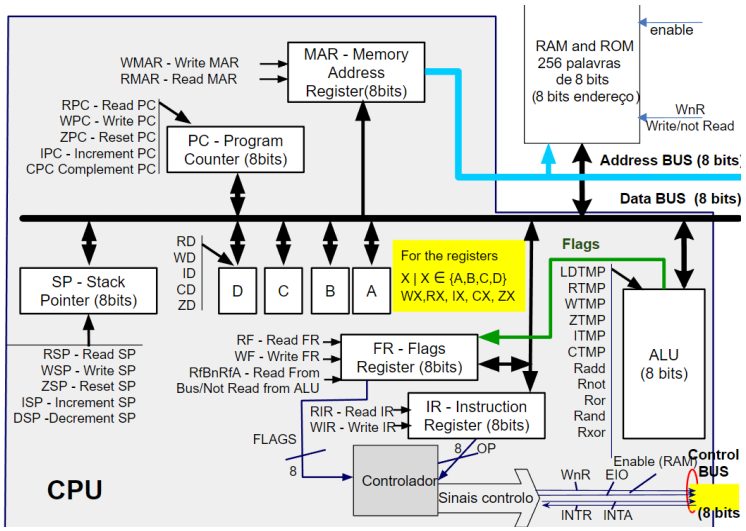


Figura 2 - Arquitetura do Processador(8 Bits)

Principalmente o CPU contém diversos componentes que serão explicados mais tarde no trabalho, mas sendo eles o Memory address register, Program Counter, Stack Pointer, Registos(A,B,C,D), como referido anteriormente Controlador e ALU(Somador, FLAGS, TMP).
E juntado de forma correta para o funcionamento do CPU obtemos a seguinte arquitetura

De seguida, começamos pela criação de máquinas de estado o primeiro tipo de linguagem(linguagem de máquina) que colocadas em certa ordem e com certo tipo de objetivo,

serão denominadas instruções das mesma evoluindo para o segundo tipo de linguagem(Assembly) e poderíamos continuar até chegar a linguagens de programação de alto nível, mas estas não serão necessárias para este trabalho.

As instruções serão guardadas na memória com um certo número de bits(8 neste caso) que serão levadas para o Instruction Register(IR) na fase denominada fetch a que aplicámos neste trabalho são as seguintes:

-ADD A,[addr] - Opcode 00000001 - Coloca no registo A o valor que estiver guardado na posição de memória com endereço [addr]

-TEST A, B - Opcode 00000010 - Através das flags verifica o resultado da operação lógica and entre A e B por bit

-NAND A, B - Opcode 00000011 - Diferente da operação TEST A, B realiza a operação lógica nand e coloca o resultado no registo A.

-MOV X, Y - Leva o que indicado em Y para o correspondente X. Exemplo se X e Y forem um registo então será copiado e levado o valor de Y para X, se X for um endereço [addr] então é levado para a memória de endereço [addr] o valor de Y, se Y for um byte será posto em X o byte

-MOV [addr], A - Opcode 00000100

-MOV B,byte - Opcode 00000101

-MOV A, [B] - Opcode 00000110

-MOV D, A - Opcode 00010011

-MOV A, B - Opcode 00010100

-Mov SP, byte - Opcode 00001111

-Mov A, byte - Opcode 00000000

-PUSH X- Coloca no topo da stack, ou seja no endereço apontado pelo Stack Pointer, e depois decrementa o mesmo, um valor X. Exemplo PUSHF irá colocar no topo da stack o valor guardado nas Flags e o PUSH A vai copiar o valor de A para a topo das flags.Sempre que um PUSH é realizado deve ser também realizado um POP para evitar eventuais problemas.

-POP C- Opcode 00001011 - Ao contrário do PUSH irá buscar ao valor apontado pelo Stack Pointer e coloca o mesmo no registo indicado a seguir ao comando

-IRET - Opcode 00010001 - Com o objetivo de inverter o Push do PC e Flags causados por Interrupt de software ou hardware, incrementa-se o SP e para retornar ao PC e às Flags o seu valor

-JC addr- Opcode 00010011- Ignora ou passar sem usar o endereço indicado

-JPE addr- Opcode 00010110 - Realiza o mesmo que o JC mas neste caso verifica se a flag paridade estiver a 1

-INT X - Interrupt de software. Parando a execução normal e fazendo um push das flags e do pc para a correspondente parte da ram na stack irá realizar as instruções guardadas em X até

parar quando encontrar IRET

-INT addr - Opcode 00010000

-IN A, D - Opcode 00001000 - leva para a A o valor correspondente na memória ao endereço guardado em D

-OUT A, D - Opcode 00001001 - leva o valor de A para o endereçado por D

-CALL addr - Opcode 00001110 - Guarda na posição de memória(stack) apontado pelo Stack Pointer o valor do PC da próxima instrução, decrementa o SP e coloca o PC no addr

-JST addr - Opcode 00010111

Interrupts

-STI - Opcode 00011000 - Vai iniciar um interrupt de Flag

-CLI - Opcode 00011001 - Vai parar um interrupt de Flag

CMP A, B - Opcode 00011010 - Vai subtrair o valor de B ao valor de A e carregar nas flags o resultado.

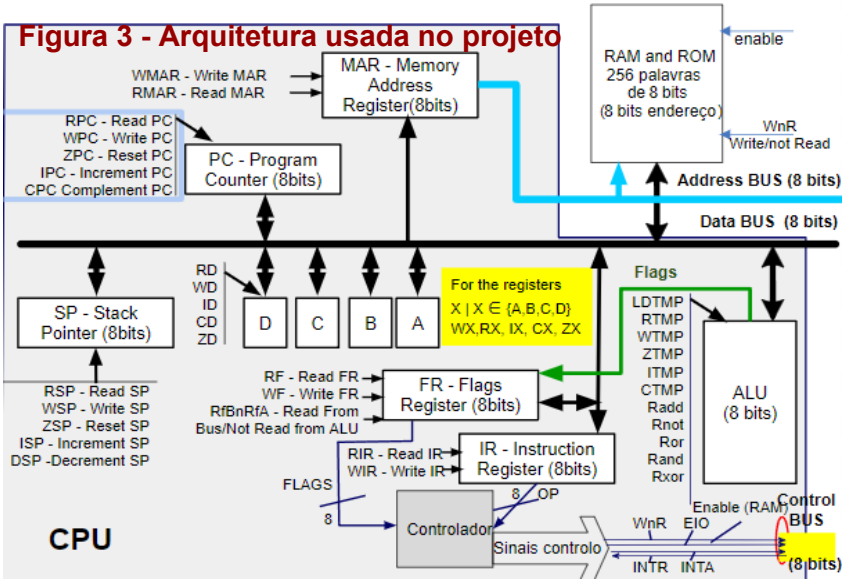
-INT - Sem opcode por ser operação realizada pelo o hardware

-Stop - Opcode 00000000 - Evitando que o programa corra infinitamente temos de ter uma instrução de stop para o mesmo parar quando realizar tudo o que lhe foi pedido

3. APRESENTAÇÃO DA ARQUITECTURA IMPLEMENTADA

Neste capítulo iremos observar a arquitetura implementada e discutir as principais razões do seu uso e a sua adequação para o trabalho que pretendemos realizar como o seu funcionamento e bases

A arquitetura deste projeto baseia-se em um CPU e periféricos como memória, um interrupt (dispositivo 115) e outros dois dispositivos externos de entrada/saída(não representados na imagem).
O CPU está por si dividido em mais 3 unidades sendo estas a ALU(Unidade de aritmética e lógica), o Controlador e registradores



Memórias

Neste microcomputador foram implementadas dois tipos de memórias:

I. RAM(Random Access Memory)
   A. Permite temporariamente guardar programas, dados ou resultados
II. ROM(Read Only Memory)
   A. Guarda programas permanentemente através de código binário

ALU

Unidade Aritmética e Lógica responsável por todas as operações a serem realizadas

Registos

Capazes de guardar pequenas quantidades de dados(endereços, instruções, ou outro tipo de memória) designados de D,C,B,A também podem ser utilizados outros tipos de partes do microprocessador como registros temporários na operação de instruções.

Registos específicos/dedicados

A. Flag Register(FR) - Guarda o Estado Atual do microprocessador e é controlado pelas restrições impostas pela operações aritméticas
B. Program Counter(PC) - Guarda o endereço da próxima instrução que deve ser realizada pelo computador
C. Instruction Register(IR) - Responsável pelo controlo da unidade que guarda a informação a ser realizada pelo computador
D. Stack Pointer(SP) - Aponta(Guarda) a última posição a ser utilizada pelo stack

4. Detalhe dos módulos do Sistema

Neste capítulo apresentaremos a estrutura do nosso trabalho realizado no Xilinx e falaremos de cada um dos módulos criados para o processador e o objetivo na sua criação.

Apresentação de árvore de ficheiros schematics do microcomputador

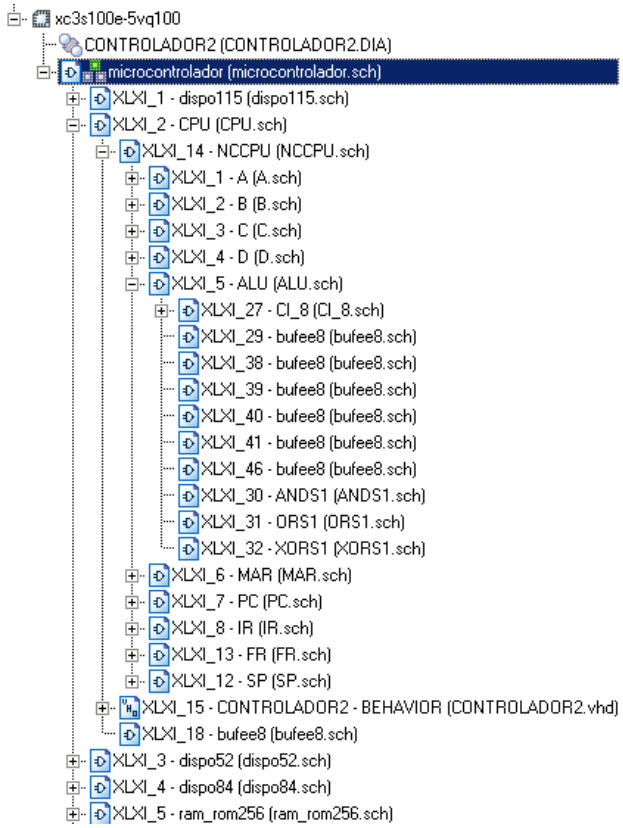


Figura 4 - Árvore de ficheiros geral

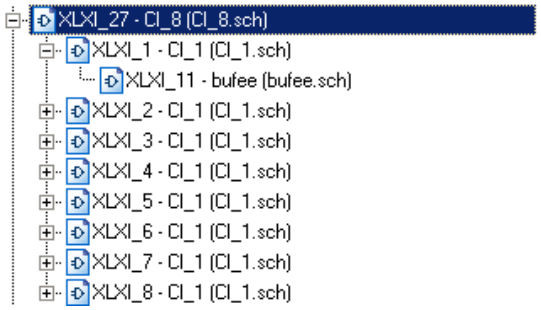


Figura 5 - Árvore de ficheiros do registro

Detalhes de cada componente do módulo

Microcomputador

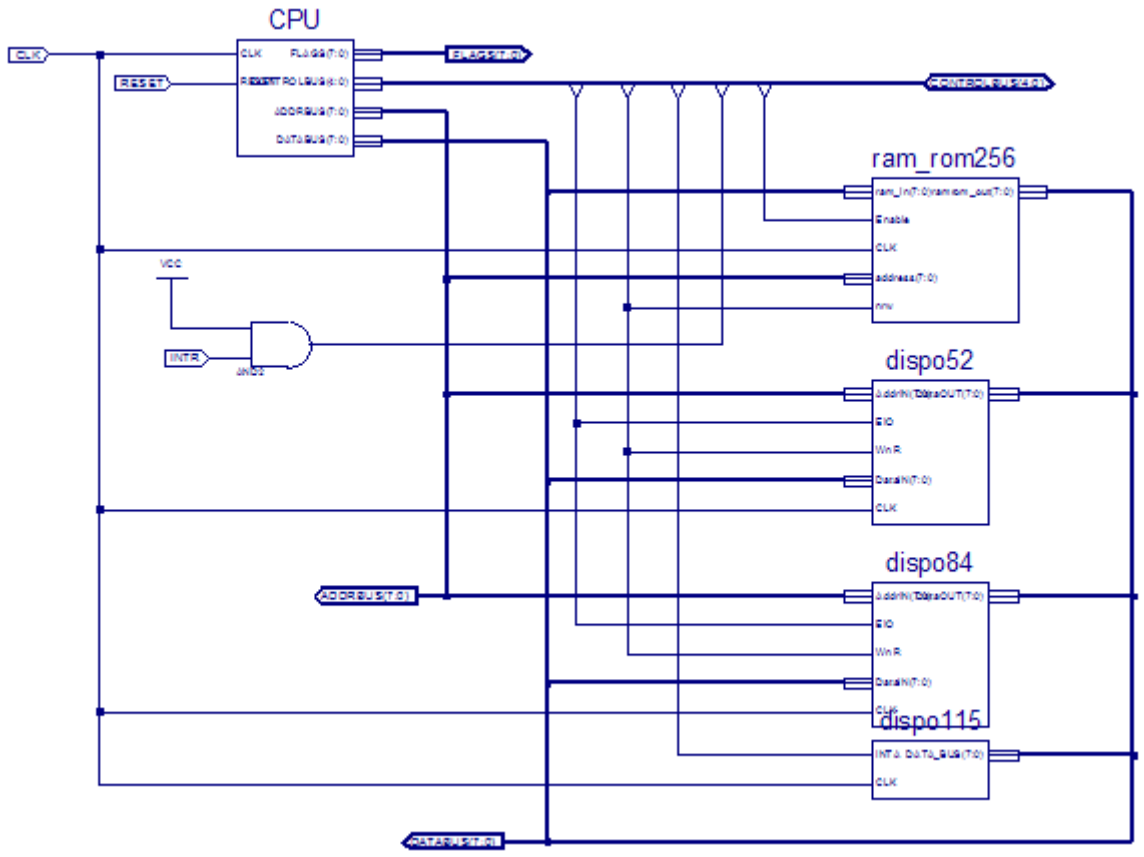


Figura 6 - Desenho de microcomputador

O microcomputador desenvolvido no nosso grupo contém os 3 periféricos, o CPU e a RAM/ROM. Deste modo, este é o ficheiro em que se liga todos os componentes para formar um computador:

- O nanoprocessador, que tem o nome de CPU
- A memória, que tem o nome de ram_rom256
- Os periféricos, com o nome de dispo 115, dispo84 e dispo52.
    - O dispo115 é aquele usado para o interrupt de hardware.
    - O dispo84 e dispo52.

ram_rom256

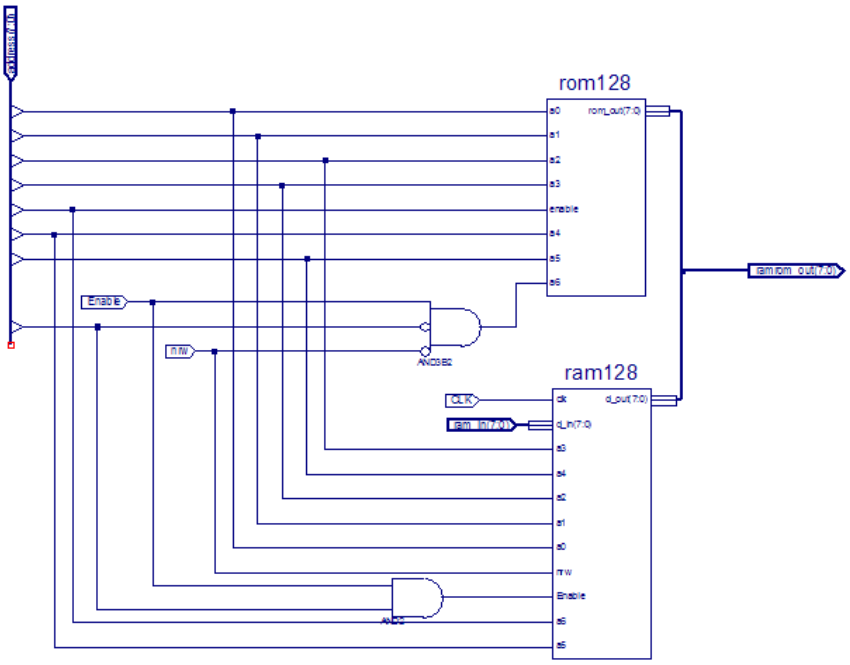


Figura 6 - Desenho de ram_rom256

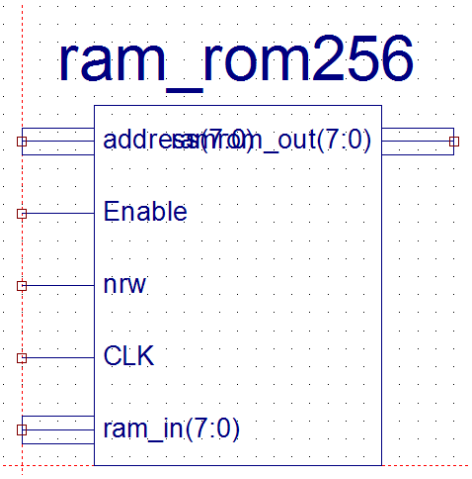


Figura 7 - Símbolo de ram_rom256

A ram_rom256 é a parte do microcomputador que guarda o opcode das instruções que serão processadas pelo controlador. A RAM tem o stack. Que serve para instruções como CALL e RET.

Por fim, guarda também valores que, processados com as instruções, formam um programa.

Dispo 84 e 52

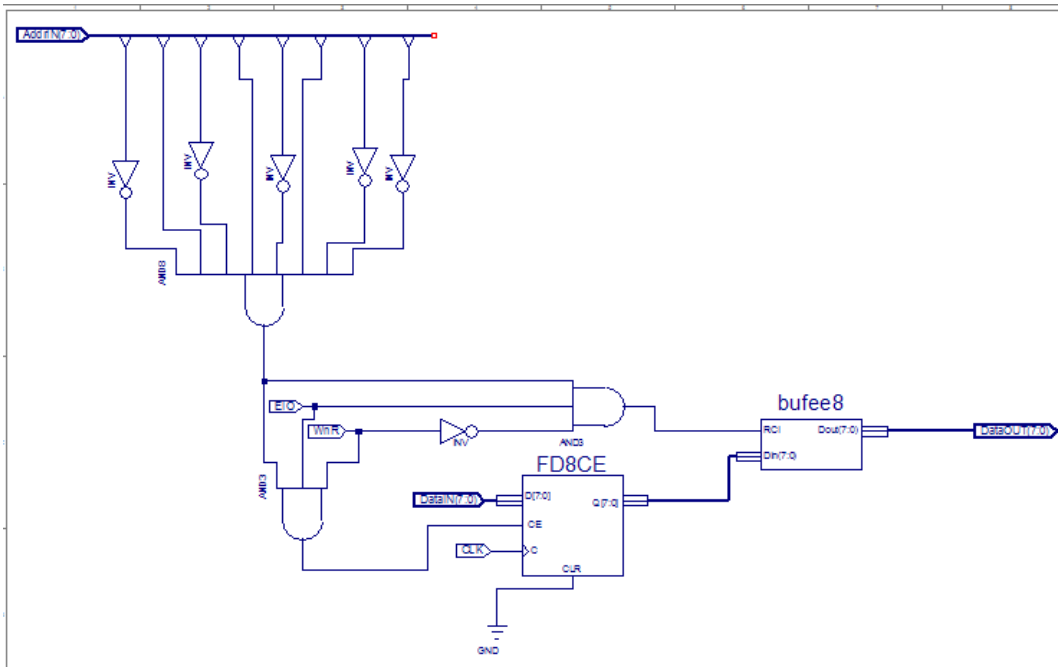


Figura 8 - Desenho de dispo84

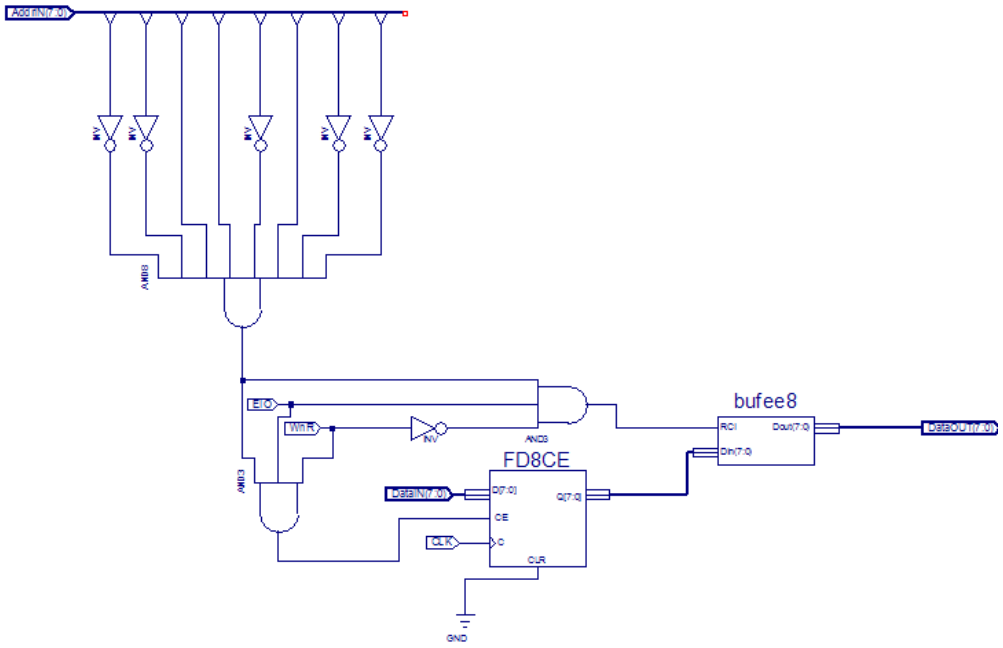


Figura 7 - Desenho de dispo52

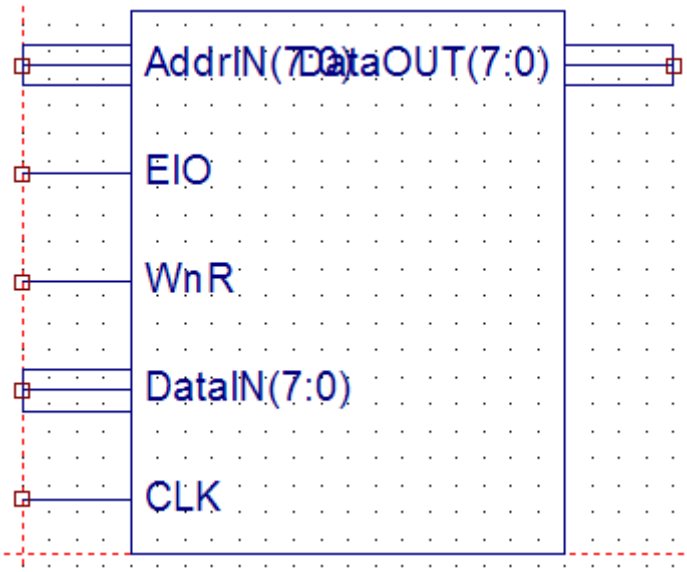


Figura 8 - Símbolo de dispo52 e dispo84

Os dispositivos 52 e 84 são periféricos que podem tanto passar valores para o databus como ter o seu registro guardado valores pelo data bus que vem o CPU.

Para o valor dentro do dispo52 passar para o databus, por exemplo, é necessário:

- O valor do adressbus ser 52 em decimal e o sinal EIO estar ativo.

Para o valor do databus ser guardado no registro do dispo84, é necessário:

- Valor do adressBus ser 84, o sinal EIO(Enable I/O) e WnR(Write not Read) ser ativados.

Dispo 115

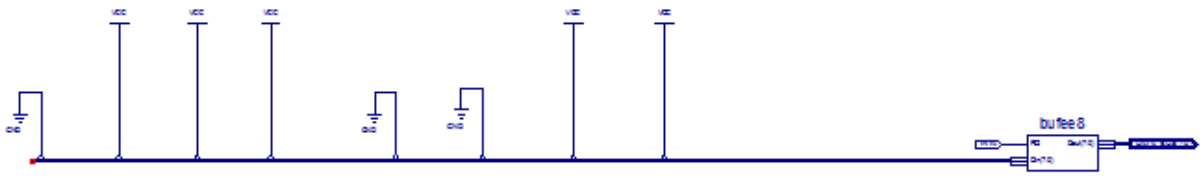


Figura 8 - Desenho de dispo115

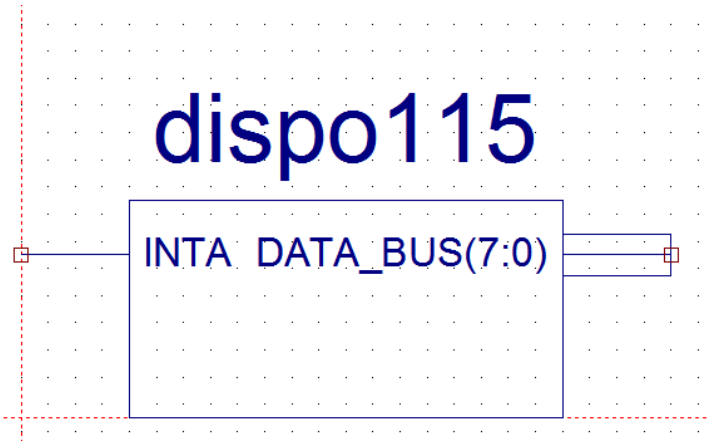


Figura 9 - Símbolo de dispo115

O dispositivo 115 serve apenas para, quando se dá o INTA, o valor que passa no databus será o valor 115 em binário. Que nasce o dispo 115.

CPU

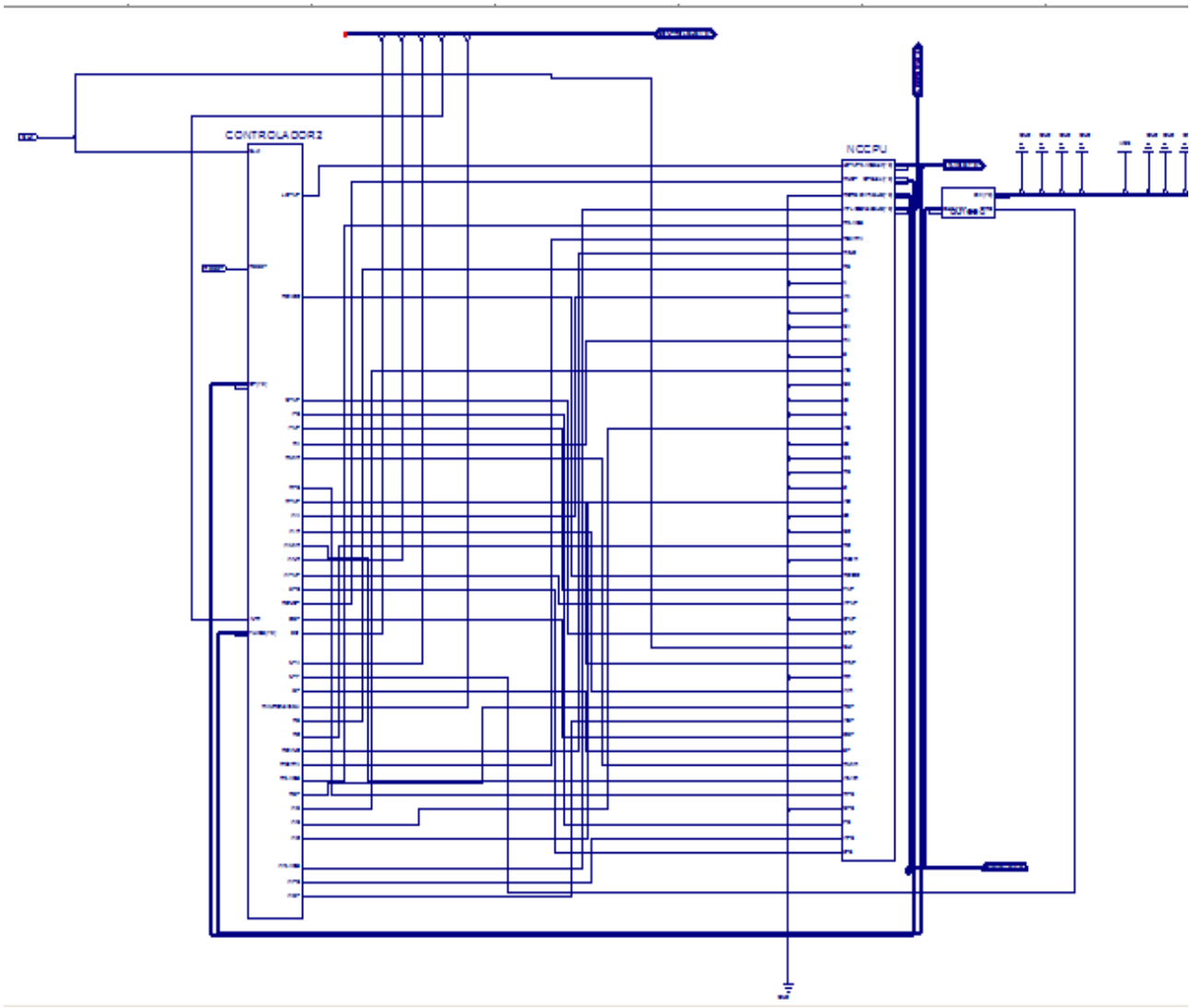


Figura 10 - Desenho de CPU

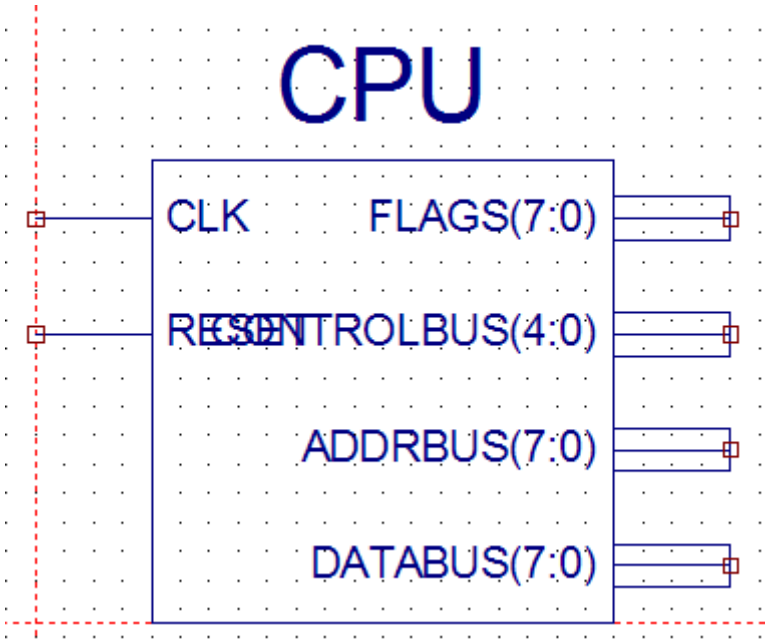


Figura 11 - Símbolo de CPU

- O CPU está composto por dois componentes:
  - NCCPU(No controller central processing unit): A parte toda do CPU sem o controlador).
  - Controlador2: O Controlador do CPU, por outras palavras, o componente do CPU que oriente todas as suas funções. Este é um controlador sequencial, por outras palavras, é um controlador que executa cada instrução de cada vez.

Controlador

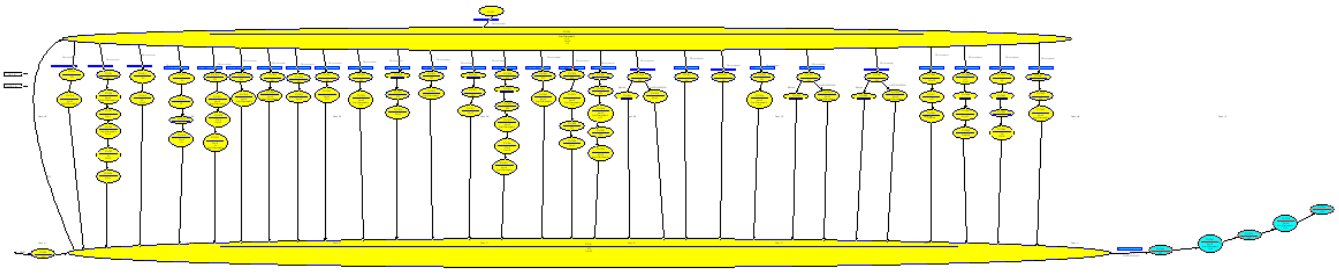


Figura 12 - Máquina de estados de controlador

Figura 13 - Símbolo de máquina de estados de controlador

O controlador é a parte do CPU onde, dependendo do OPCode que vem do instruction register, diferentes conjuntos de sinais para o NCCPU são enviados.

Este conjunto de sinais podem causar alterações na memória e nos registros do CPU.

Assim, um programa é apenas uma combinação de muitos conjuntos de sinais transmitidos do controlador ao NCCPU.

NCCPU

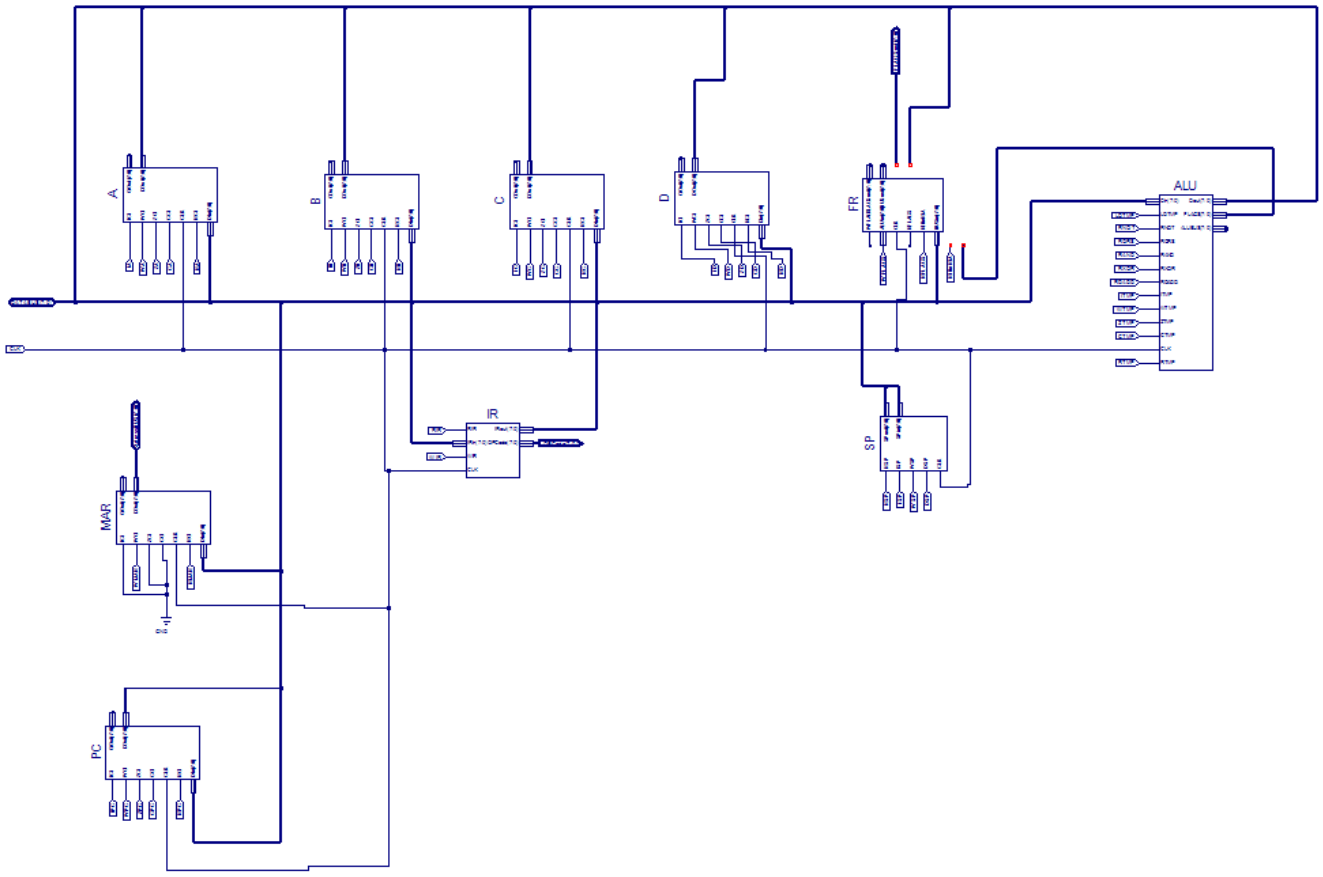


Figura 14 - Desenho do NCCPU(No controller central processing unit)

Esta é a parte do CPU onde ocorre a maior parte do processamento das instruções. Sendo o controlador o componente que as organiza de forma a criar instruções.

O NCCPU contém:

- IR(Instruction register)
- SP(Stack pointer)
- FR(Flag register)
- PC(Program counter)
- MAR(memory address register)
- ALU(Arithmetic and logic unit)
- A, B, C, D(registos)

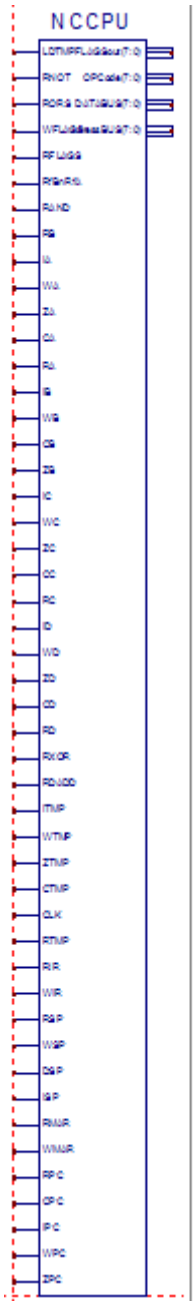


Figura 15 - Símbolo do NCCPU(No controller central processing unit)

-

Stack pointer

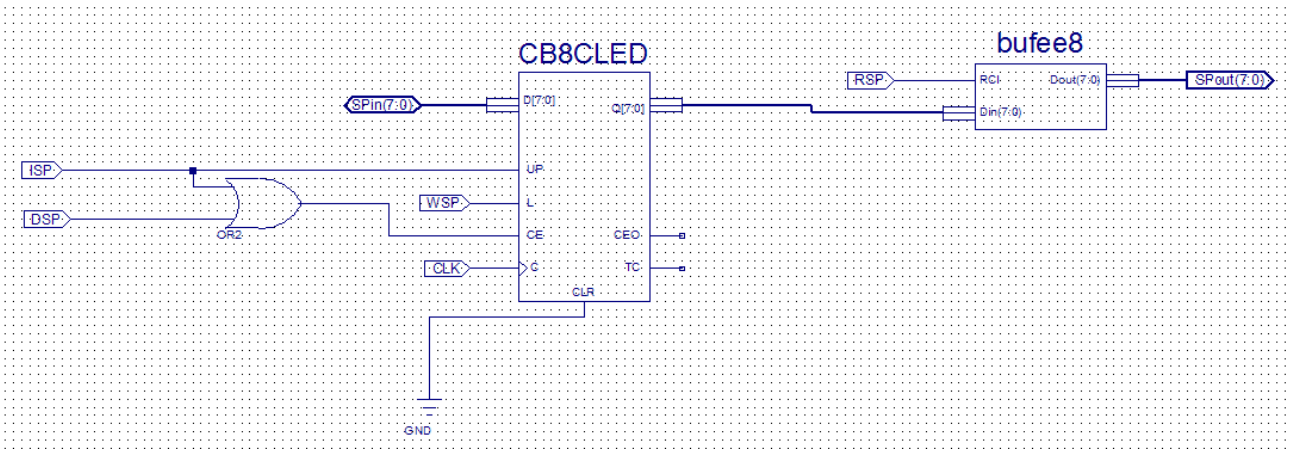


Figura 16 - Desenho do stack pointer

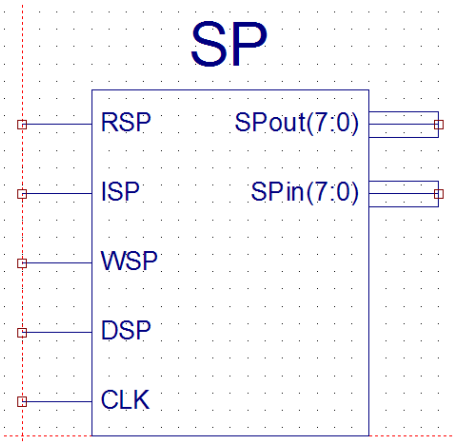


Figura 17 - Símbolo do stack pointer

O stack pointer é a parte do CPU em que se guarda a posição de memória livre no topo do stack.

Quando o stack pointer incrementa, as entradas UP e CE estão ativas.

Quando o stack pointer decrementa, apenas a entrada CE é ativada.

Dessa forma, o registo decrementa o valor que contém lá dentro. Apontando desta forma para a próxima posição de memória livre no topo do stack.

Instruction register

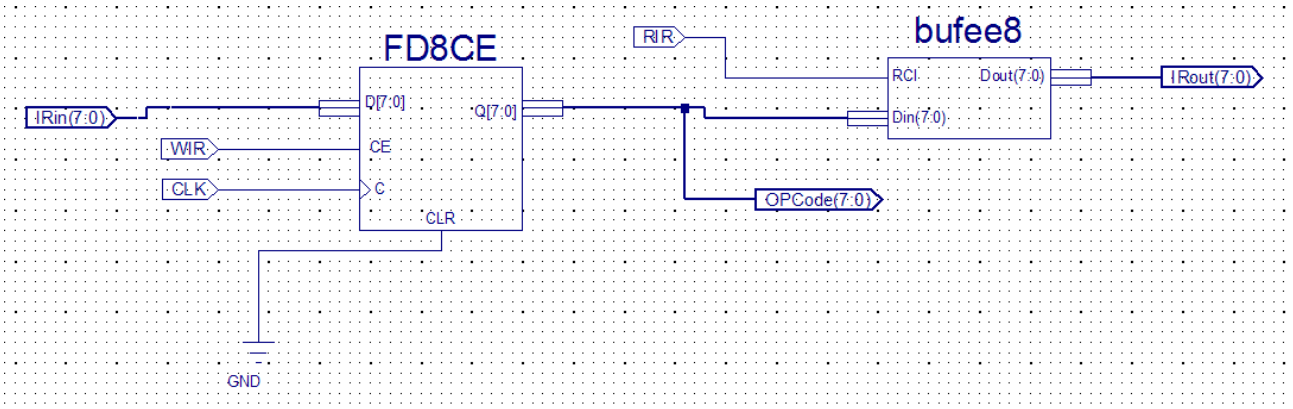


Figura 18 - Desenho do instruction register

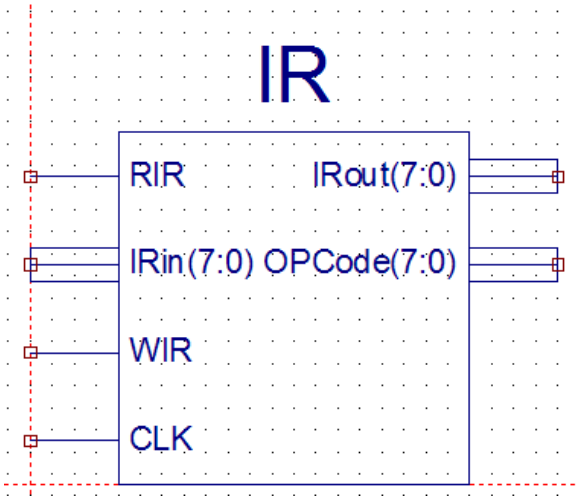


Figura 19 - Símbolo do instruction register

O instruction register(IR) guarda temporariamente o OPcode, que vem da RAM e transmite este ao controlador.

Quando se quer que o IR guarde o valor da instrução, ativa-se o WIR.

Todavia, o IR também pode ser usado como um registro para guardar valores temporariamente. Nesse caso, quando queremos que esses valores temporários sejam escritos no IR, ativamos o WIR.

Quando queremos que o valor temporário vá para o databus, ativamos o RIR. Que serve para evitar curto circuito no databus com os outros componentes.

Flag register

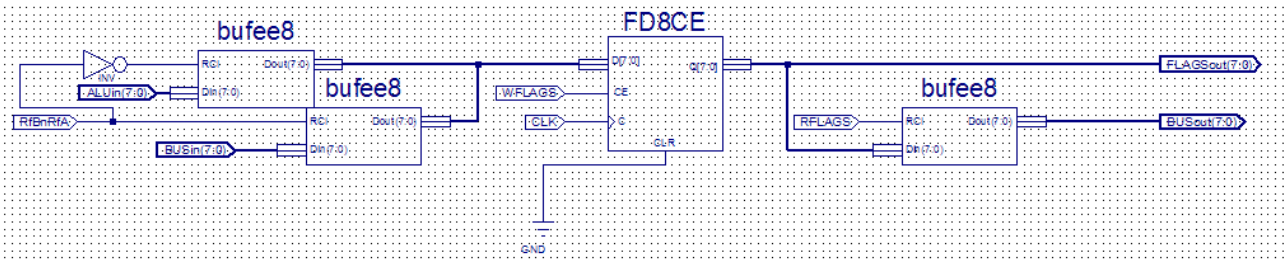


Figura 20 - Desenho do flag register

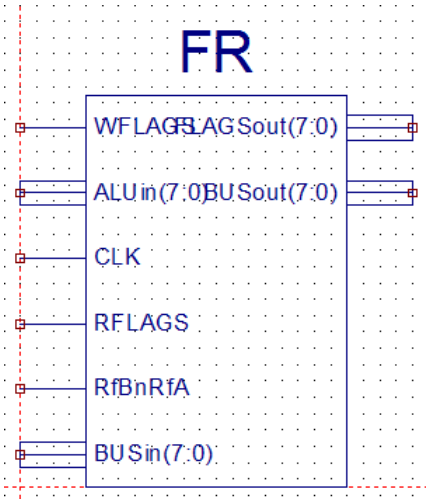


Figura 21 - Símbolo do flag register

O Flag register serve para guardar o valor das flags temporariamente, para, por exemplo, irem para o stack. Podem até mesmo guardar valores temporariamente.

Para guardar um valor temporariamente do FR do databus, ativa-se o sinal RfBnRfA(Read from bus not Read from ALU). Depois, o valor que o dataBus tem é colocado no registo FD8CE na condição que o sinal WFLAGS é ativado.

Exatamente o mesmo acontece quando queremos guardar o valor das FLAGS no FR. Só que desta vez, não adotamos o valor do RfBnRfA. O resto do processo é igual.

Assim, o valor sai para o FLAGS out, que está ligado ao controlador para as instruções de Jump condicionais.

Caso queiramos que o valor vá também para o databus, apenas ativa-se o RFLAGS.

ALU(Arithmetic and logic unit)

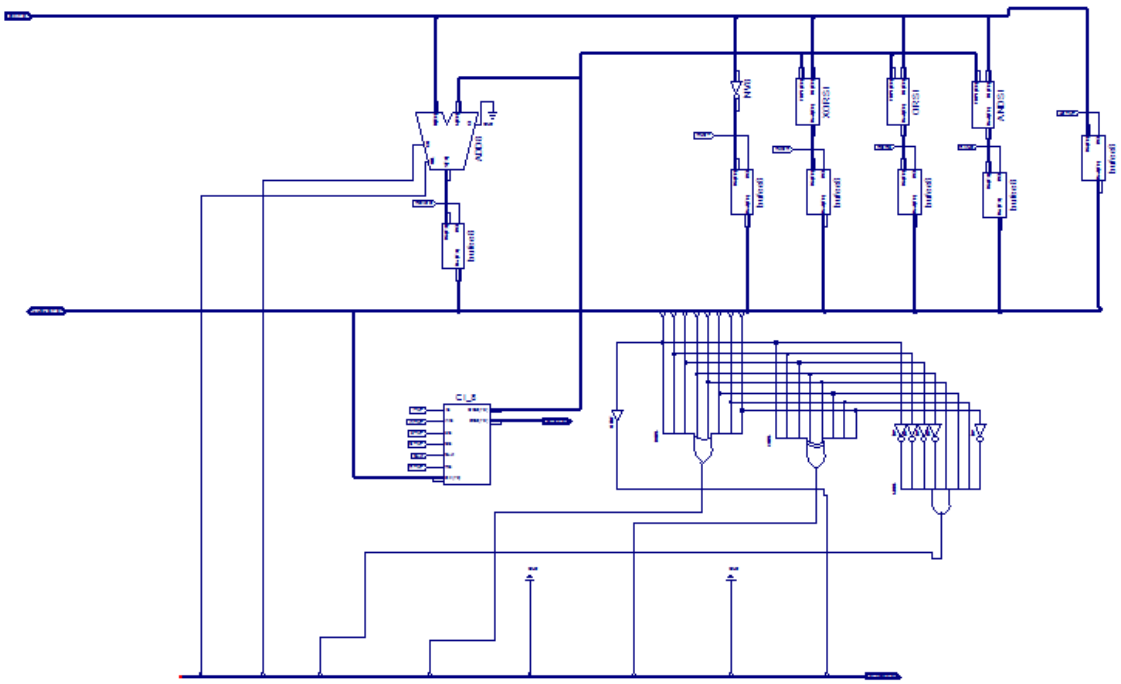


Figura 22 - Desenho do ALU

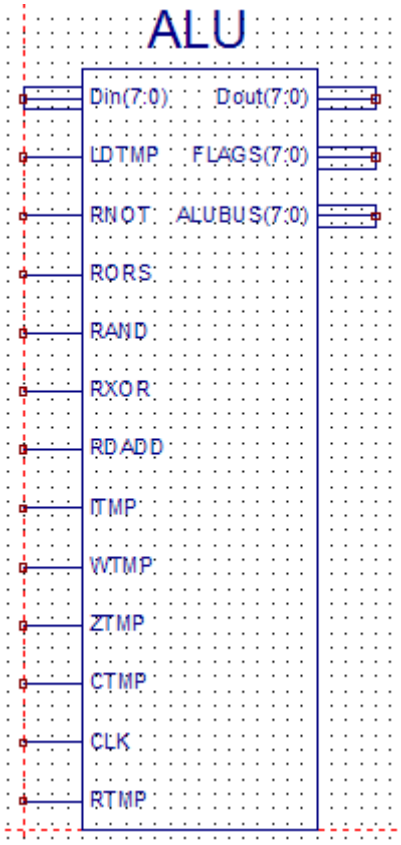


Figura 23 - Símbolo da ALU

A ALU(Arithmetic and logic unit) é a parte do nanoprocessador que realiza as operações aritméticas e lógicas.

As operações aritméticas são efetuadas no somador(ADD8):

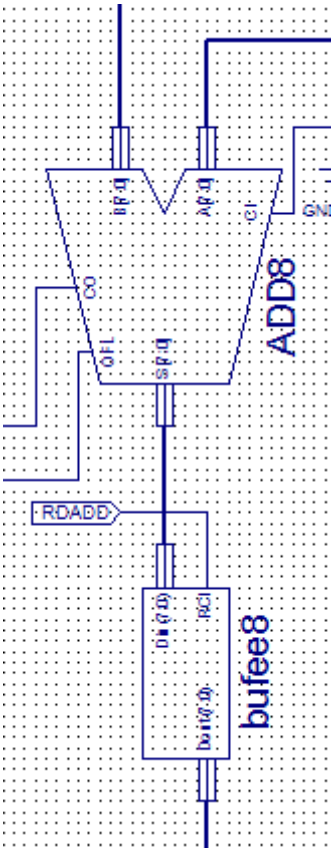


Figura 23 - Secção do somador no desenho da ALU

Este tem uma buffer, que serve para evitar curto circuito no alubus.

A ALU também tem portas lógicas, para efetuar operações lógicas e a parte das ALUflags, para retirar características das operações lógicas e aritméticas.

Para efetuar as operações, a ALU dispõe de um registro(TMP):

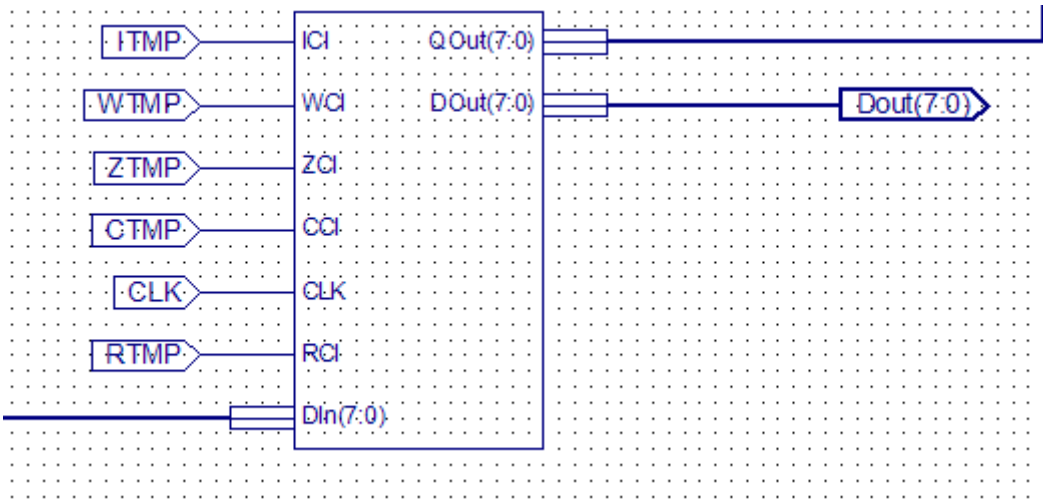


Figura 24 - Secção do registro TMP no desenho da ALU

Este registo também pode ser usado como registo temporário para guardar valores durante o a execução de uma instrução.

Para tal, ativa-se o LDTMP para que o valordo databus vá para o registo.

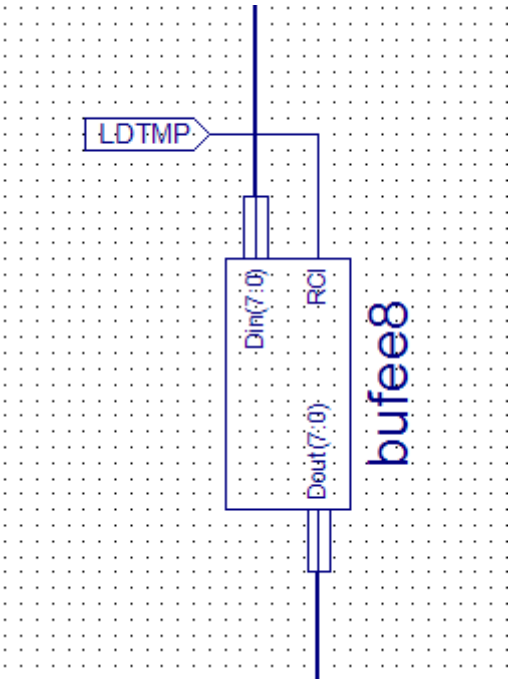


Figura 25 - Secção do buffer LDTMP no desenho da ALU

Depois ativa-se o WTMP para o valor ser escrito no registro TMP

ALUFlags

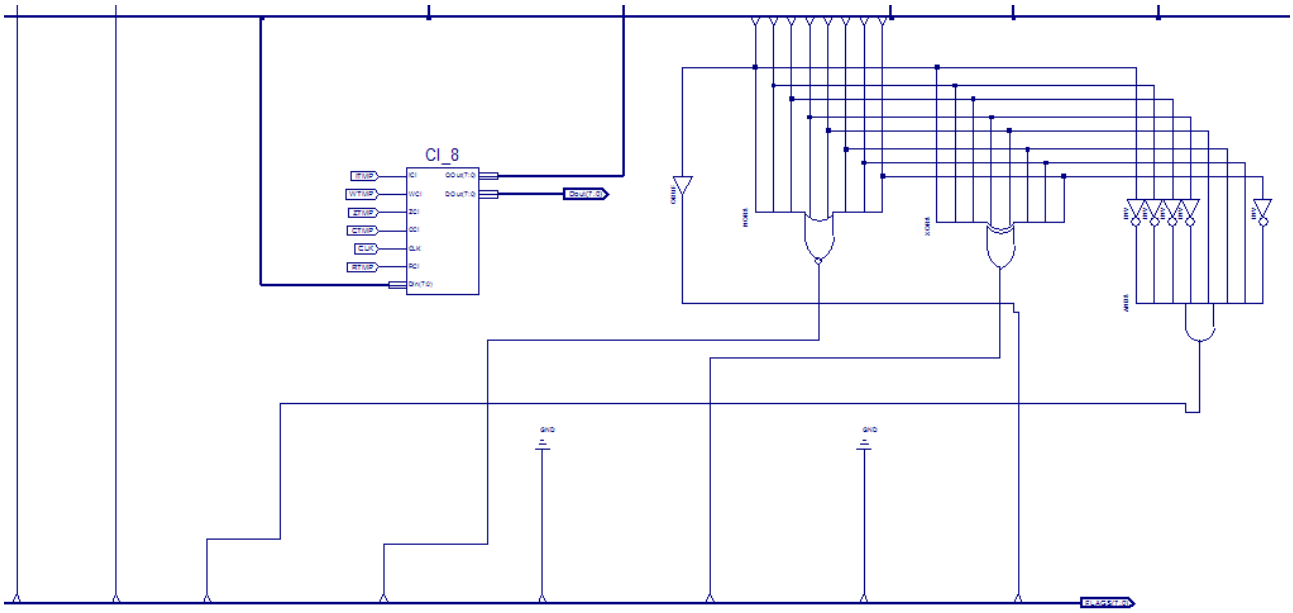


Figura 26 - Secção das ALUflags no desenho da ALU

A ALU Flags tem como objetivo registrar certas características que ocorrem nas operações aritméticas e lógicas.

Dois dos fios vem do somador(ADD 8), o carry(CO) e OFL(overflow)

Outros 4 sinais são:

- Parity(ver se a números de 1 no byte é par)
- Signal(Se o bit mais significativo é 1, retorna 1. Se é 0, retorna 0)
- Zero(Ver se todos os bits do ALUbus é 0)
- student(se a soma dos últimos dígitos dos alunos do grupo for igual ao valor presente do databus)

Colocamos grounds em alguns fios, para garantir que o valor que entra no Flag register é de 8 bits. Para assim não

Portas lógicas

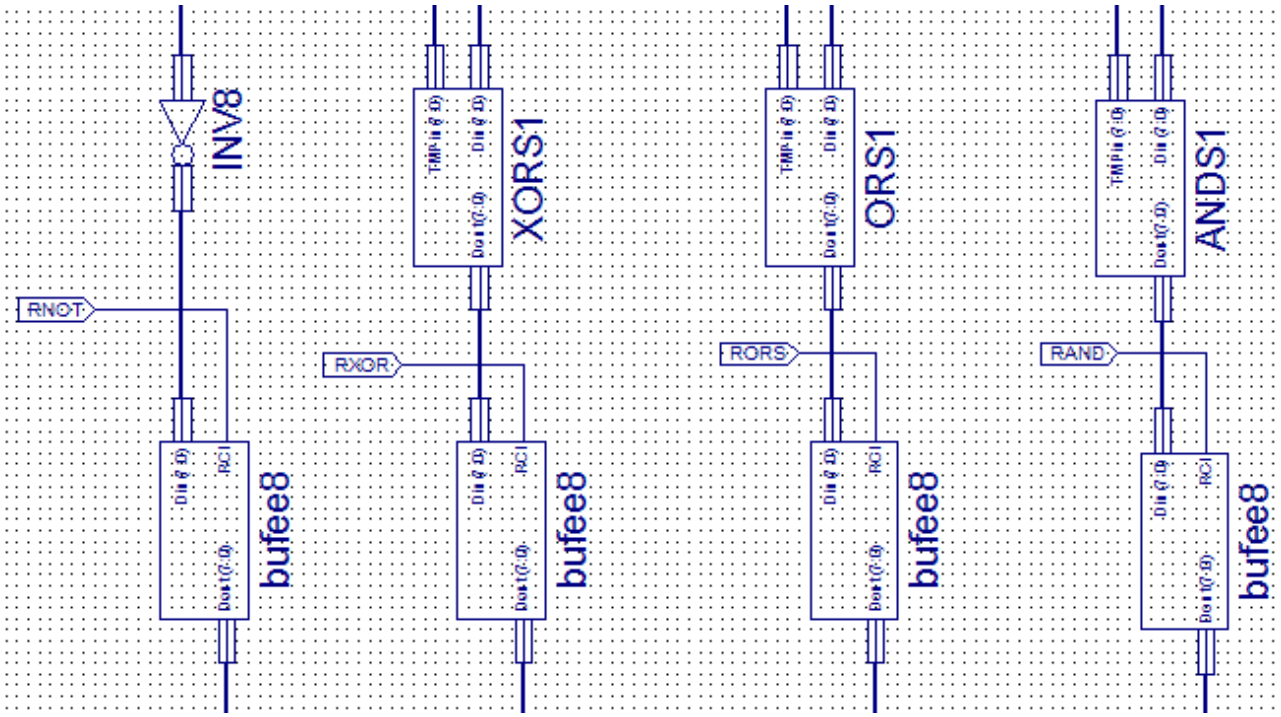


Figura 27 - Secção das portas lógicas no desenho da ALU

As portas lógicas têm como objetivo realizar as operações lógicas nos valores do databus.

Todos estes têm um buffer(bufee8) que serve para evitar um curto circuito dentro do alubus.

Dessa forma, por exemplo, caso queiramos colocar o valor invertido do databus no registo CL_8, é necessário ativar o sinal RNOT, para o valor invertido do databus estar presente no alubus.

Para o colocar no CL_8, ativa-se o sinal WTMP.

Portas lógica ANDS

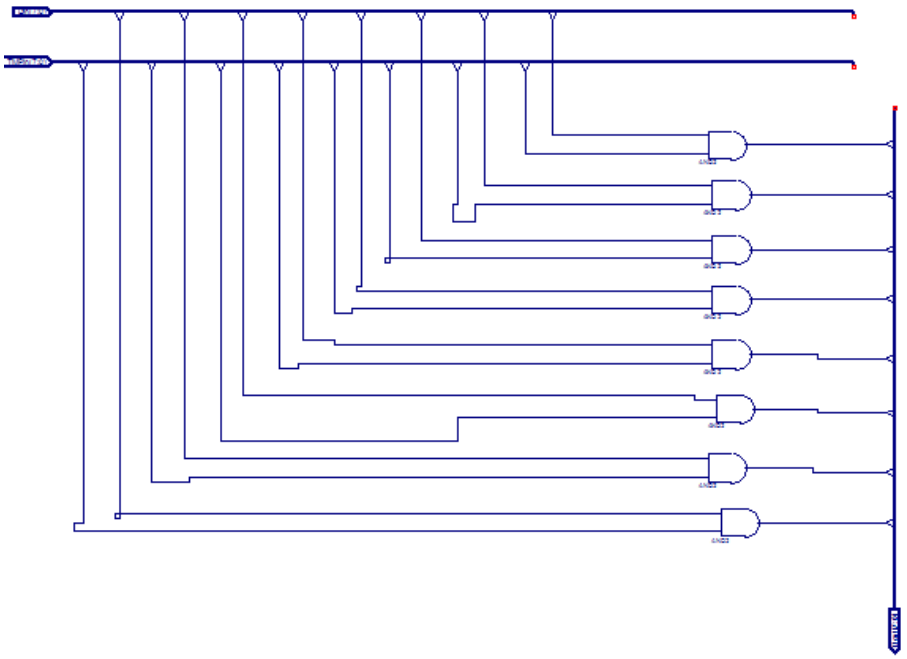


Figura 28 - Desenho da porta lógica ANDS1

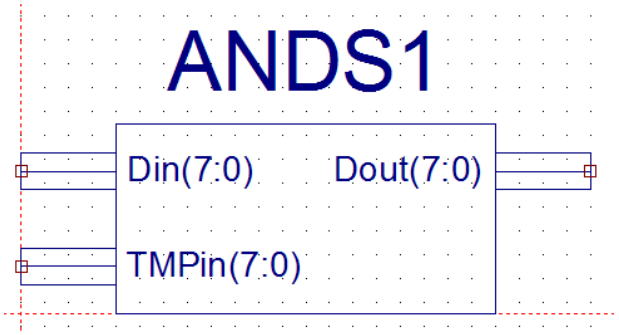


Figura 29 - Símbolo da porta lógica ANDS1

A porta lógica ANDS1 serve para comparar cada bit e fica a 1 quando ambos os bits são a 1.

Portas lógicas ORS

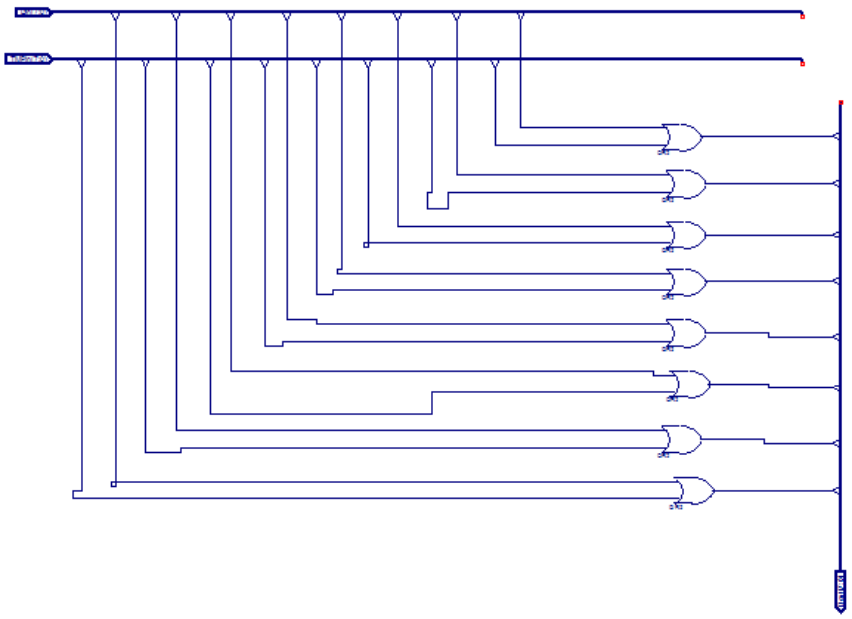


Figura 30 - Desenho da porta lógica ORS

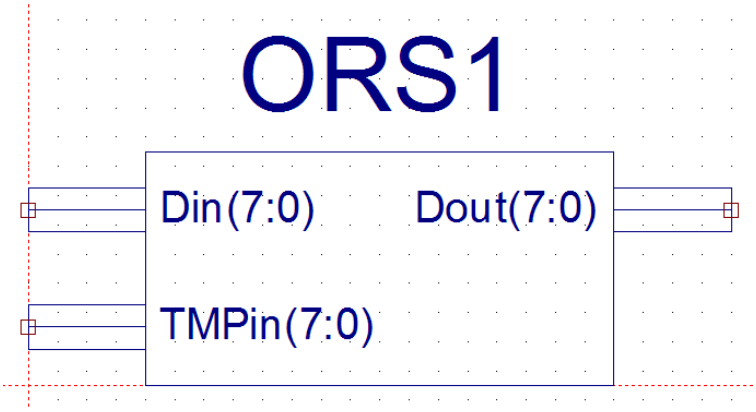


Figura 31 - Símbolo da porta lógica ORS

A porta lógica ORS1 serve para comparar cada bit e fica a 1 quando um dos os bits fica a 1.

Portas lógicas XORS

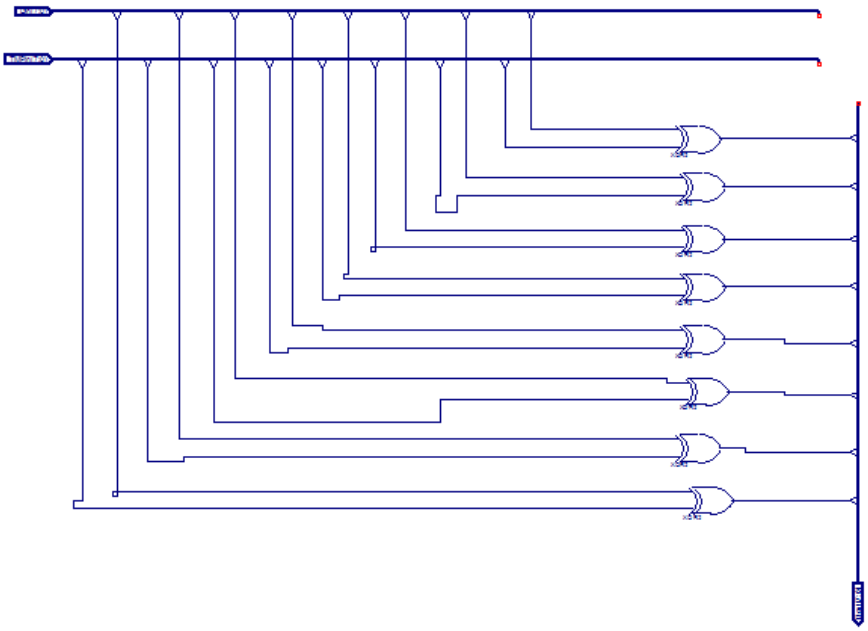


Figura 32 - Desenho da porta lógica XORS

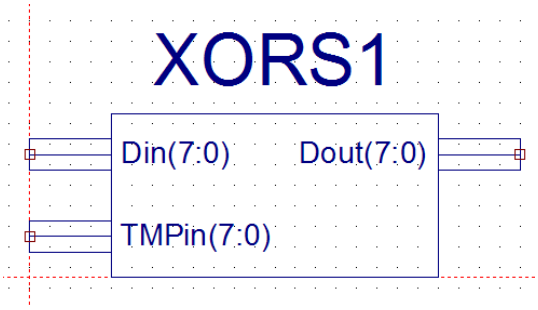


Figura 33 - Símbolo da porta lógica XORS

A porta lógica XORS1 serve para comparar cada bit e fica a 1 quando ambos os bits são diferentes.

Detalhes de registros(A, B, C, D, PC, MAR, TMP)

A, B, C, D, PC, MAR, TMP

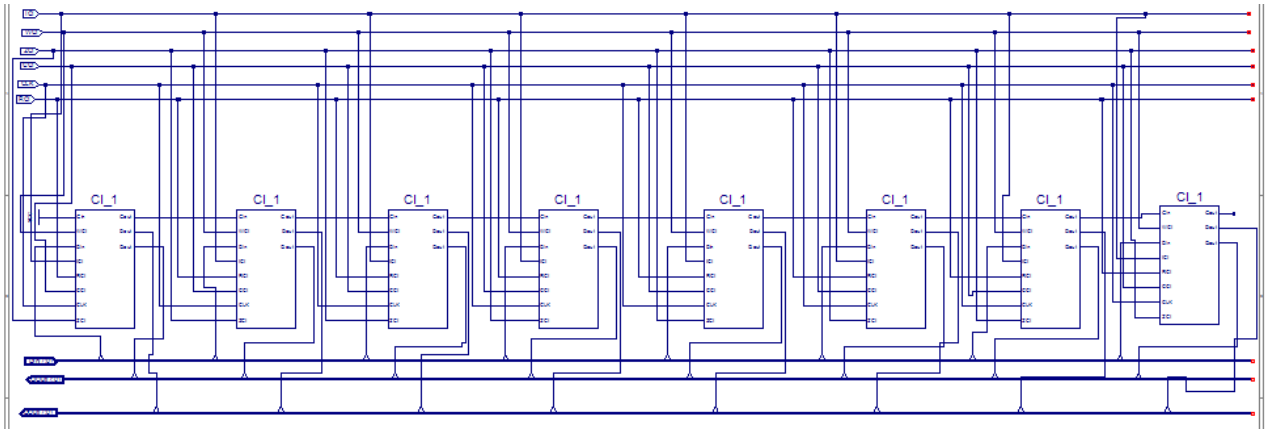


Figura 34 - Desenho de registro 8 bits

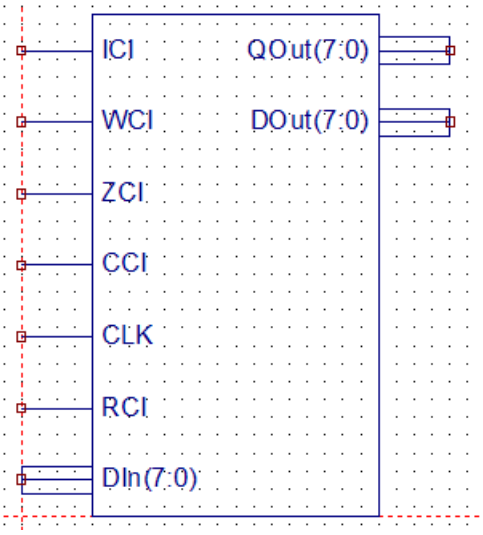


Figura 35 - Símbolo de registro 8 bits

A saída Qout apenas é usada na ALU, para os valores do TMP irem diretamente para as portas lógicas e para o somador.

No caso do MAR, as entradas ICI, ZCI e CCI estão ligadas ao ground. Isto é porque o MAR(memory address register) apenas guarda os valores que vão para o instruction register e os endereços de memória da RAM.

O registo A, B, C, D tem a entrada DIn e DOut ligadas ao databus.

CL_1

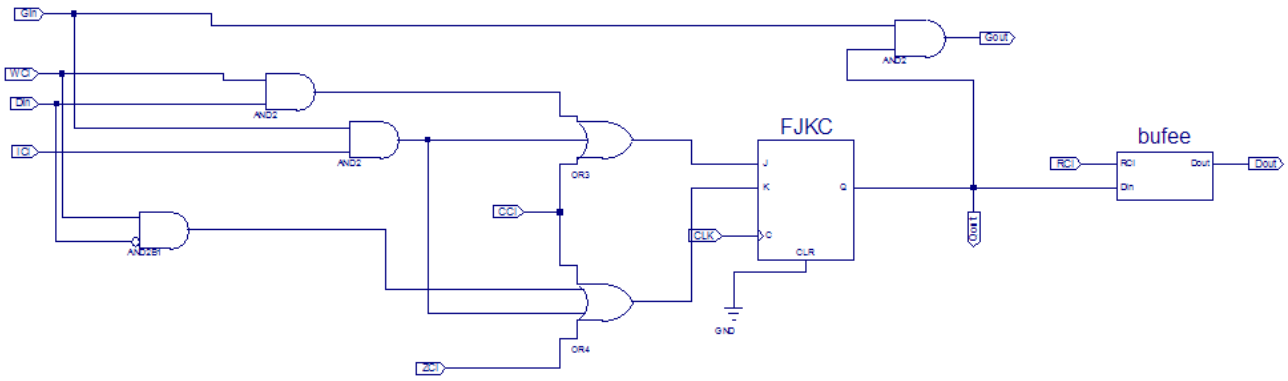


Figura 36 - Desenho de registro 1 bit

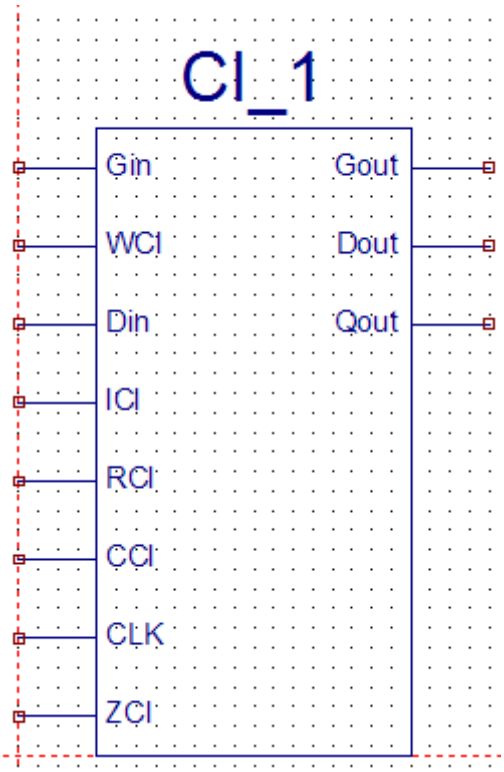


Figura 37 - Símbolo de registro 1 bit

Detalhes do buffer e buffer8

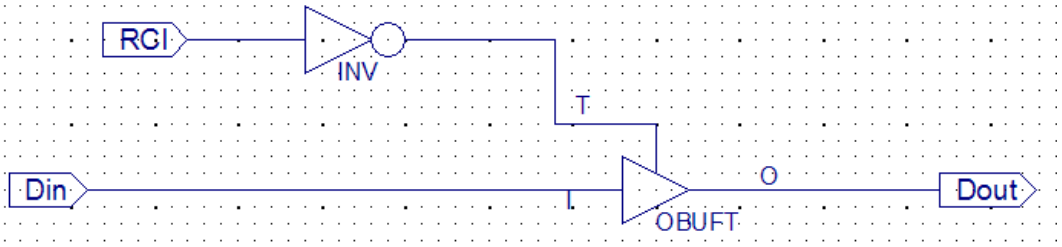


Figura 38 - Desenho de buffer

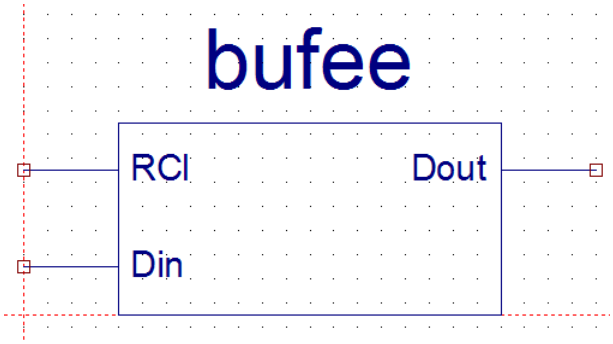


Figura 39 - Símbolo de buffer

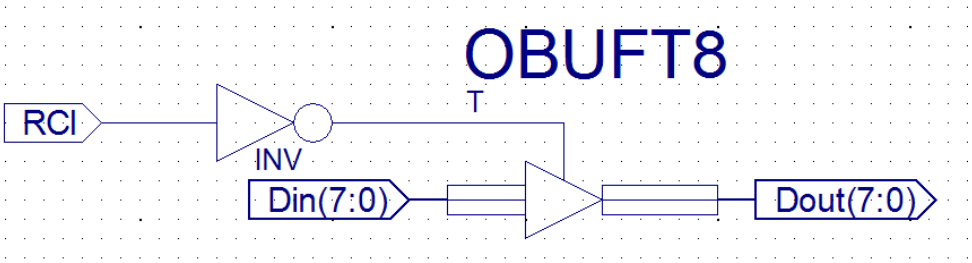


Figura 40 - Desenho de buffer 8 bits

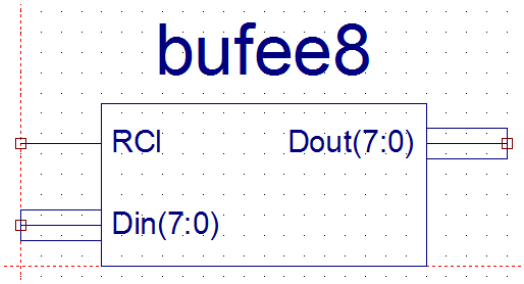


Figura 41 - Símbolo de buffer 8 bits

5. TESTES E RESULTADOS

Neste Capítulo iremos falar sobre os testes e comentar sobre os resultados obtidos nos mesmo se obtivemos o que era esperado ou não ou que fomos capazes de realizar, as componentes funcionais do projeto e erros inesperados.

Teste CLI

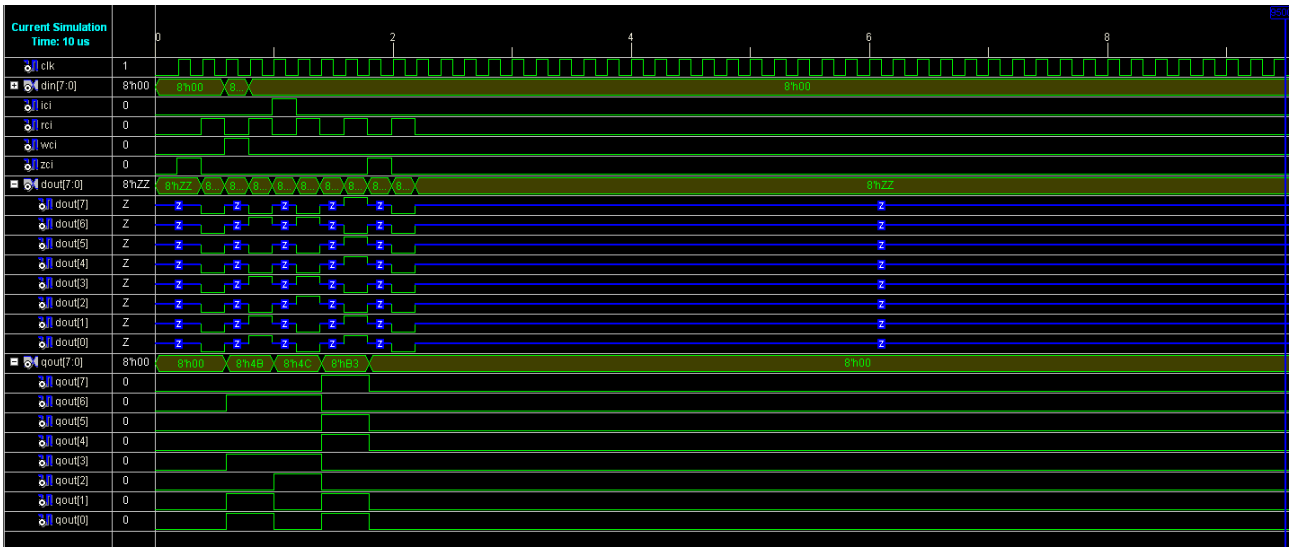


Figura 42 - Teste CLI

Neste teste que nós realizamos com base no indicado no moodle podemos observar o funcionamento do registo com bases nos valores que lhe eram dados ao longo do tempo e assim sendo podemos observar que foi obtido o esperado no mesmo, com o valores a serem dados guardados e depois enviados para os fios corretamente quando pedido, sendo capaz de realizar reset e incrementação

Teste ALU

Teste

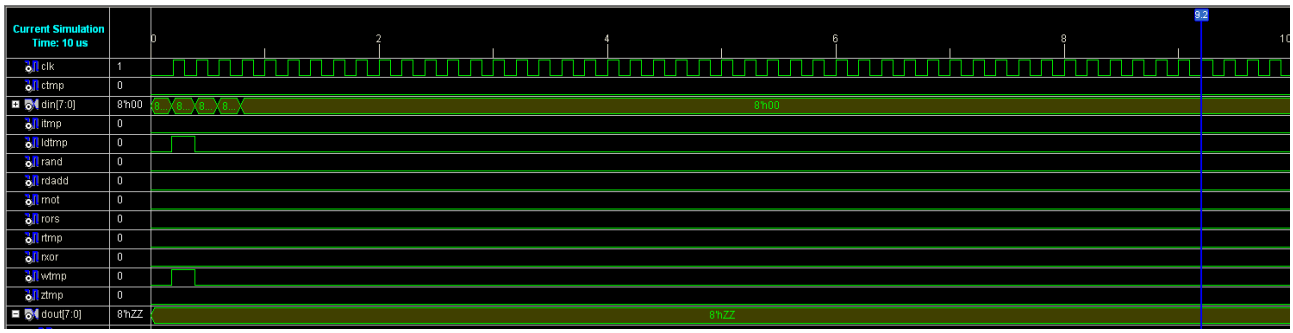


Figura 43 - Teste ALU

Resultado

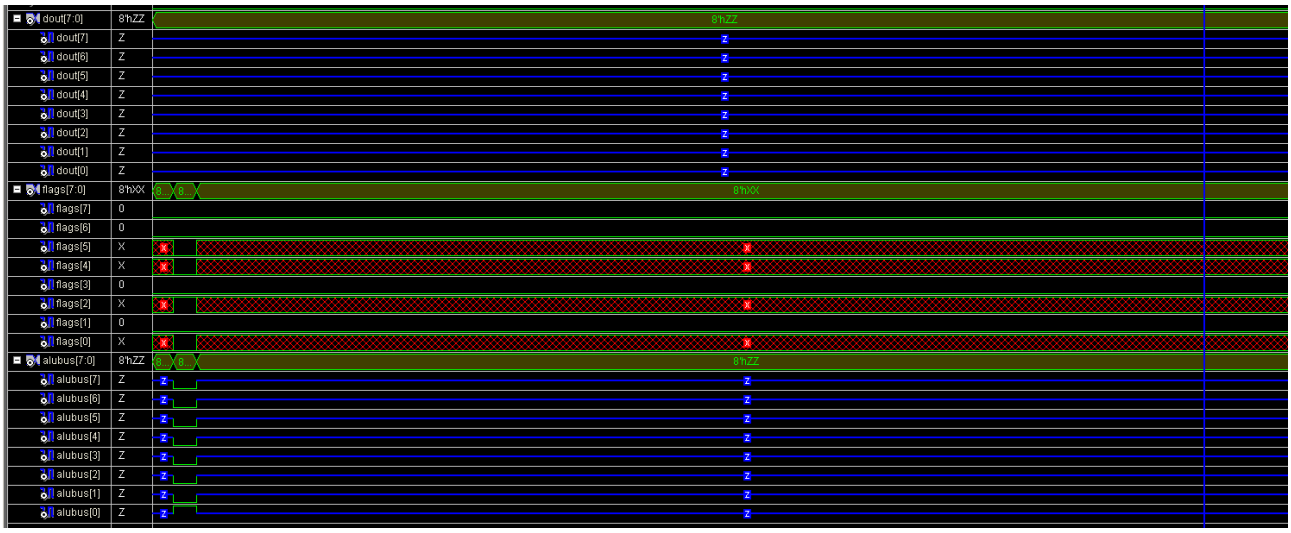


Figura 44 - Resultado teste ALU

Para a realização deste teste demos um valor de um bit à ALU e percebemos que esse valor era recebido pela mesma, as flags corretas ativavam-se, ou seja a flag paridade ativa indicando que o número era ímpar.

Ao realizarmos um conta de somar e uma operação na flag os resultados obtidos foram os esperados também, ao que podemos observar que o funcionamento da nossa flag se encontrava credível e obtemos o valor X(Don't Care) quando a mesma se encontra indefinida.

Teste Dispositivos 82 , 54

As diferenças nestes dois dispositivos encontra-se apenas na sua função de entrada ou seja o resultado é exatamente o mesmo ao que realizamos o teste em apenas um deles já que o outro daria o mesmo alterando os valores do BUS para o respectivo endereço do dispositivo em binário

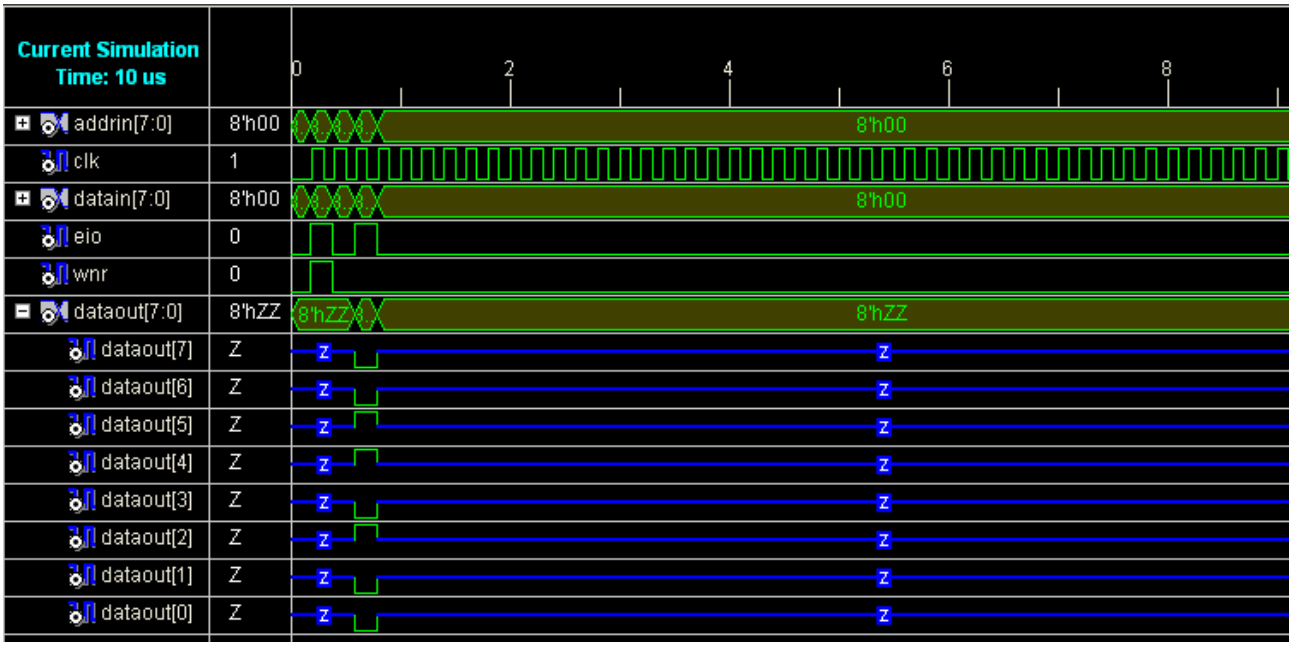


Figura 45 - Teste dispositivo 52

Ao receber o valor 52 em binário com a ativação do dispositivo e do WnR é nos possível escrever um valor no registo, ao voltar a ativar o dispositivo será possível obtermos nos Bus o valor do gravado anteriormente no registo. Provando o funcionamento do dispositivo em questão.

Teste NCCPU (No controller CPU) e SP

Por fim no teste do CPU sem controlador testamos a troca de informação entre os registos A, B e a ALU, um POP ao stack pointer, ao instruction register e à MAR e um teste das flags todos deram os resultados esperados

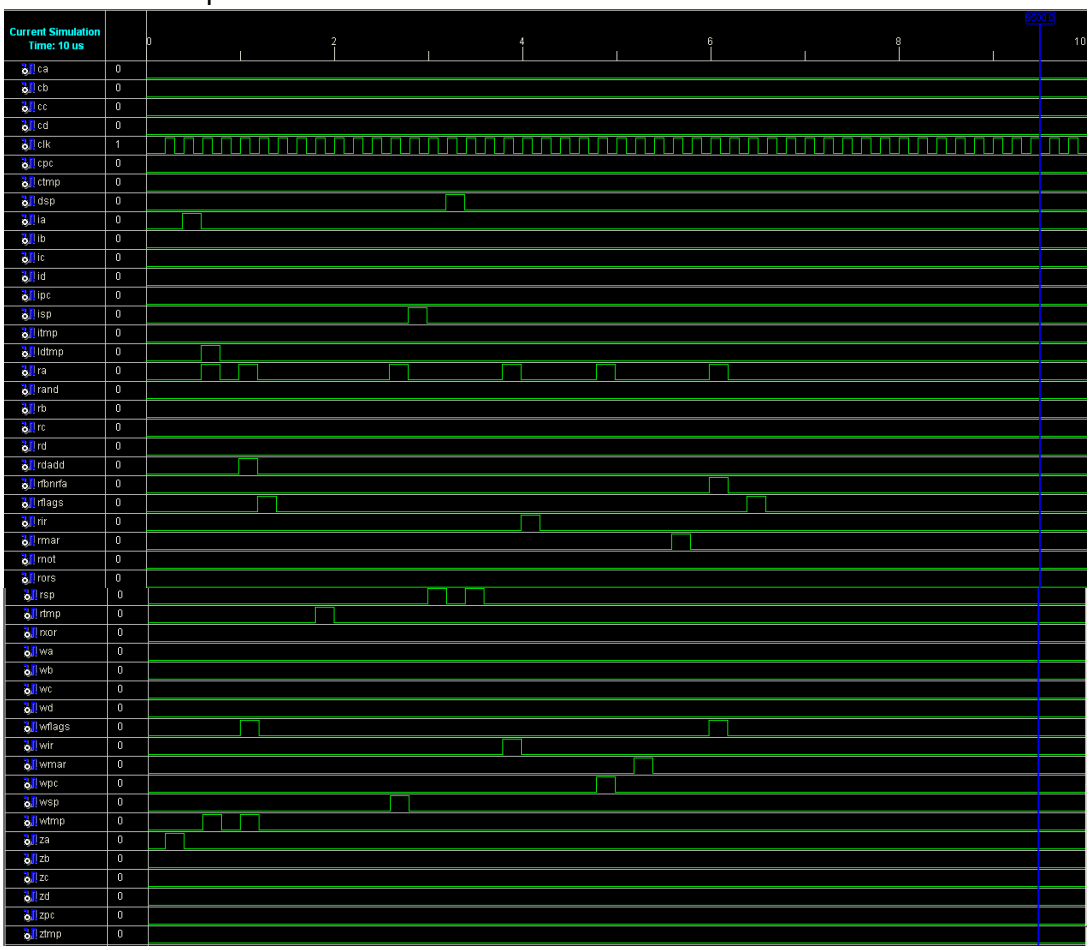


Figura 46 - Teste NCCPU e SP

Resultado

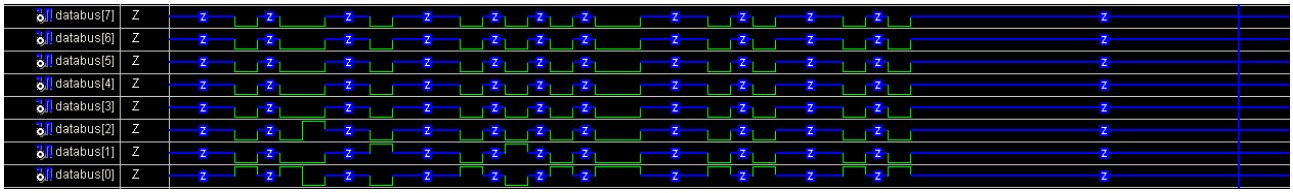


Figura 47 - Resultado teste NCCPU e SP

Observamos os resultados e os valores obtidos confirmando o funcionamento do CPU, ALU e SP, IR , MAR e do Flag Register onde obtermos o resultados obtidos na função que mandamos para cada um dos mesmo.

Teste do dispositivo 115

Ao ativarmos o INTA o resultado esperado acontece ou seja o valor 115 é enviado pelo dispositivo para o DATA BUS, para a realização do teste tivemos de usar um clock, mas o dispositivo em si não necessita do mesmo.

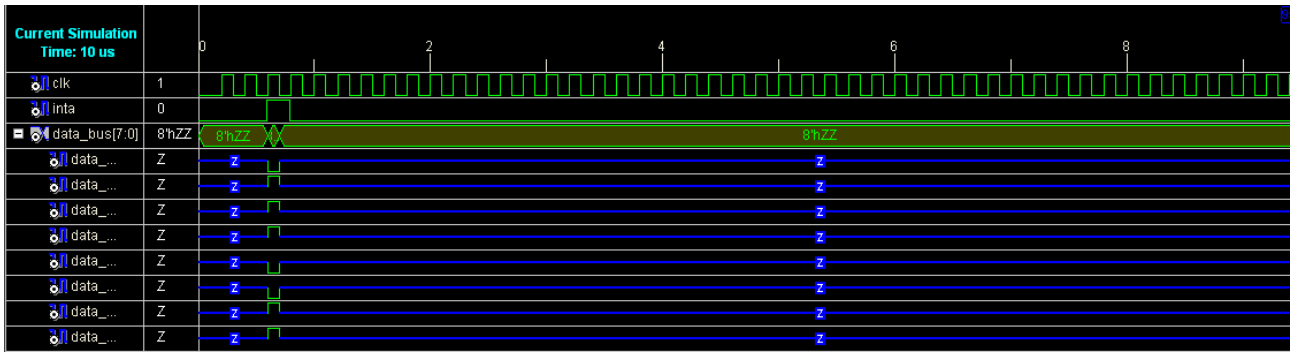


Figura 48 - Teste dispositivo 115

Teste CPU

O teste ao CPU não nos foi possível pois ao realizar o mesmo um erro no data bus é identificado ficando tudo em X impossibilitando a realização de qualquer tipo de instrução ou tipo de pedido direto ao processador.

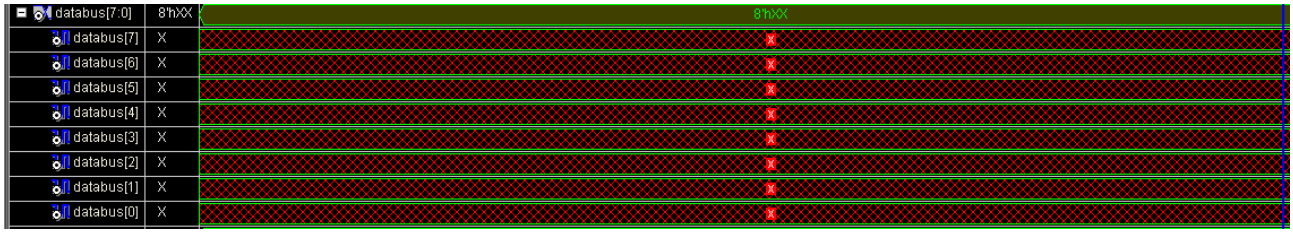


Figura 49- Teste CPU

6. CONCLUSÕES

Neste capítulo final vamos falar resumidamente sobre o trabalho onde identificamos eventuais dificuldades e o procedimento para as ultrapassar, até que parte o trabalho se encontra concluído se o mesmo não se encontrar devidamente completo.

Este trabalho que começou no início do segundo semestre com a criação de simples registos de 8 bits até ao final do mesmo, teve bastantes dificuldades ultrapassadas e momentos de stress devido ao próprio xilinx que travava ou não guardava algo quando lhe era pedido

Durante o trabalho encontramos diversas dificuldades que averiguamos durante a realização de testes como ALU dar um X mas esperávamos Z e com o avançar do trabalho reparamos que o mesmo não era um erro mas já tinha dado imensas dores de cabeça e cansaço.

Os fios foram dos maiores problemas que tivemos já que a realizar os testes certas vezes invés de obtermos o valor pretendido no bit certo obtemos no oposto como exemplo no teste da ALU o valor de A enviado era 1 somado ao valor de B que era um também e realizando uma operação de soma, era esperado obter 00000010 mas obtivemos 01000000.

Tivemos problemas na realização do teste final, a nossa data bus não realizava nenhum dos processos, pois dava X antes de tudo. A criação do controlador também foi complicada diversas momento não sabíamos bem ao que ligar e como criar.

A possibilidade de realizar teste no Xilinx foi de grande ajuda para o trabalho pois podemos identificar e corrigir muitos erros com base no mesmo, dos mais simples aos mais complexos do trabalho.

Na questão de arquitetura o CPU neste momento encontra-se devidamente completo, mas como referido anteriormente a Data Bus apenas apresenta X sendo então incapaz de realizar o teste final, mas todos os outros testes realizados aos componentes deram os resultados esperados para o funcionamento do trabalho.

A capacidade de trabalho de grupo do grupo e estarmos em ordens com aulas quando nos foi dado o trabalho final foi de grande importância para a realização e finalização do mesmo e a rapidez como conseguimos certos erros foi de extrema importância para a finalização do trabalho.