UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA MECATRÔNICA

PROJETO DE UMA CPU COM 16 INSTRUÇÕES

ELE1717 - Grupo 02 - Projeto - Problema 02

ARTHUR FELIPE RODRIGUES COSTA
ERIKA COSTA ALVES
LUCAS GUALBERTO SANTOS RIBEIRO
LUIZ VITOR CLEMENTINO
RAFAEL DE MEDEIROS MARIZ CAPUANO

Natal, 06 de fevereiro de 2021

Resumo

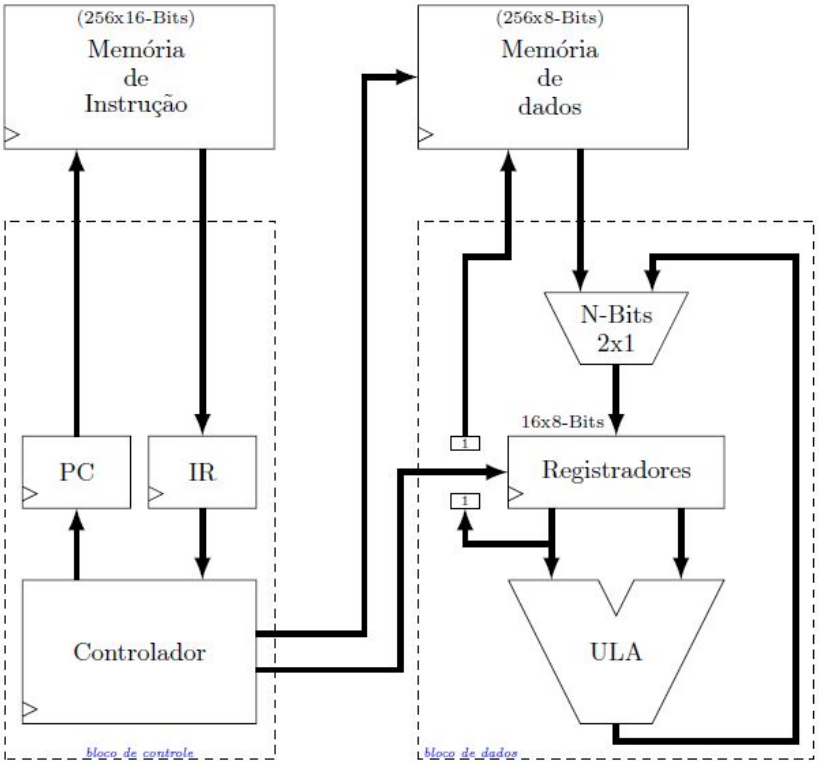
Este relatório tem como objetivo documentar e explicar o projeto de uma unidade central de processamento (CPU,) de uso geral simples, com 16 instruções. O método de projeto RTL é usado para definir as operações de controle e de dados a serem usadas, assim como memórias para armazenar as instruções e os dados.
Palavras-chaves: Memória; ULA; Instruções; Arquitetura de Computadores.

Sumário

1 Introdução

No nosso dia a dia, os processadores de computadores e celulares são os mais próximos que temos de nós, onde os mesmos podem ter milhares de instruções diferentes a serem executadas. O problema proposto trata-se do projeto de um circuito integrado de um processador de propósitos gerais simples, mais comumente chamado de CPU, de uso geral e que executa 16 instruções. O conjunto de instruções guardadas na memória de instrução são chamadas de programas. A figura 1 mostra os principais componentes usados no projeto RTL, a memória de instrução(256x16-bits) armazena as instruções a serem executadas, o contador PC indica qual a próxima instrução a ser executada de acordo com o controlador, o registrador IR armazena os 16 bits referentes a instrução escolhida. Na parte de dados temos uma memória de dados para armazenar os dados a serem tratados de acordo com a instrução, também há uma unidade lógica e aritmética (ULA) para as operações como soma, subtração e operações lógicas. O banco de registradores é usado para guardar os dados a serem operados pela ULA, o multiplexador serve para escolher o resultado de uma operação da ULA ou um dado da memória de dados.

Figura 1 - Estrutura básica do projeto RTL de uma CPU de uso geral



Fonte: material da disciplina

A figura 2 mostra as 16 instruções da CPU, cada instrução armazenada possui 16 bits de largura, onde os 4 bits mais significativos, chamado de opcode, indicam para o controlador qual a instrução a ser executada e os bits restantes indicam um valor de um dado de 8 bits ou um endereço de um registrador a ser usado para leitura ou escrita. As flags Carry e ULA são definidas pelas operações na ULA.

Figura 2 - Conjunto de instruções da CPU

Oper.	Classe	Opcode	4bits	4bits	4bits	Descrição	Carry	ULA
HLT	Controle	0000	-	-	-	PC_{k+1}=PC_k		
LDR	Dados	0001	A	addr[7..4]	addr[3..0]	Reg[A]<=Mem_D[addr]		
STR	Dados	0010	A	addr[7..4]	addr[3..0]	Mem_D[addr]<=Reg[A]		
MOV	Dados	0011	-	B	C	Reg[B]<=Reg[C]		
ADD	ULA	0100	A	B	C	Reg[A]<=Reg[B]+Reg[C]	•	•
SUB	ULA	0101	A	B	C	Reg[A]<=Reg[B]-Reg[C]	•	•
AND	ULA	0110	A	B	C	Reg[A]<=Reg[B] AND Reg[C]	•	•
OR	ULA	0111	A	B	C	Reg[A]<=Reg[B] OR Reg[C]	•	•
NOT	ULA	1000	A	-	C	Reg[A]<=NOT Reg[C]	•	•
XOR	ULA	1001	A	B	C	Reg[A]<=Reg[B] XOR Reg[C]	•	•
CMP	ULA	1010	A	B	C	Reg[A]<=CMP(Reg[B],Reg[C])	•	•
JMP	Salto	1011	-	value[7..4]	value[3..0]	PC_{k+1}=value		
JNC	Salto	1100	-	value[7..4]	value[3..0]	PC_{k+1}=value, if carry=0		
JC	Salto	1101	-	value[7..4]	value[3..0]	PC_{k+1}=value, if carry=1		
JNZ	Salto	1110	-	value[7..4]	value[3..0]	PC_{k+1}=value, if ULA≠0		
JZ	Salto	1111	-	value[7..4]	value[3..0]	PC_{k+1}=value, if ULA=0		

Fonte: Material da disciplina

Foram realizadas pesquisas sobre o funcionamento de uma CPU, assim como o funcionamento de cada instrução. No livro do Vahid (2008), no capítulo oito, há uma explicação básica do funcionamento de uma CPU simples, é possível também encontrar em outros materiais da internet o funcionamento das instruções e sua relação direta com a linguagem Assembly, dado que esta linguagem manipula uma CPU no seu nível mais baixo.

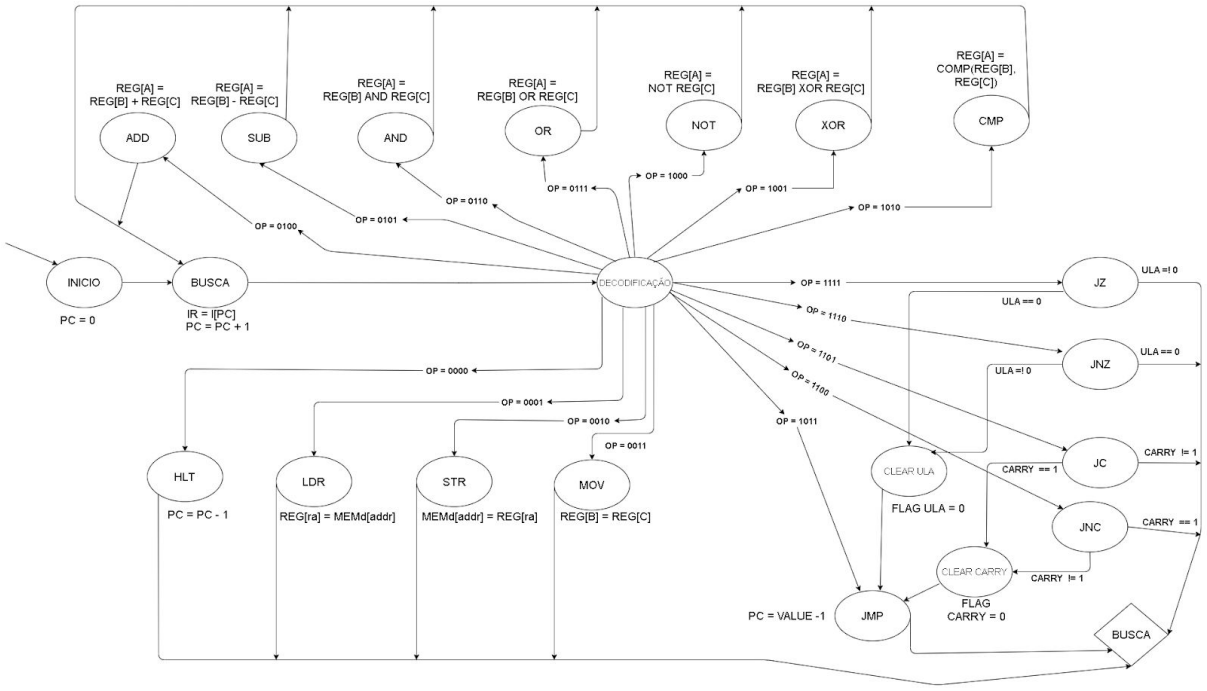
Usando o método de projeto RTL é possível montar um bloco de controle para decodificar as instruções vindas do registrador de instrução(IR) e assim manipular os sinais que controlam as operações no bloco de dados juntamente com a memória de dados, e também para manipular sinais que executam instruções lógicas na própria unidade de controle.

2 Desenvolvimento

Utilizando o método de projeto RTL é necessário dividi-lo em quatro etapas, a primeira etapa a ser mostrada será a construção de uma máquina de estados (MDE) de alto nível, com a descrição do que faz cada estado, a segunda é montar um bloco de dados onde estará a ULA e o banco de registradores, a terceira é juntar o bloco de dados ao bloco de controle, e por último montar a MDE de baixo nível com a explicação do que faz cada sinal de controle.

2.1 Máquina de estados de alto nível

Figura 3 - Máquina de estados de alto nível

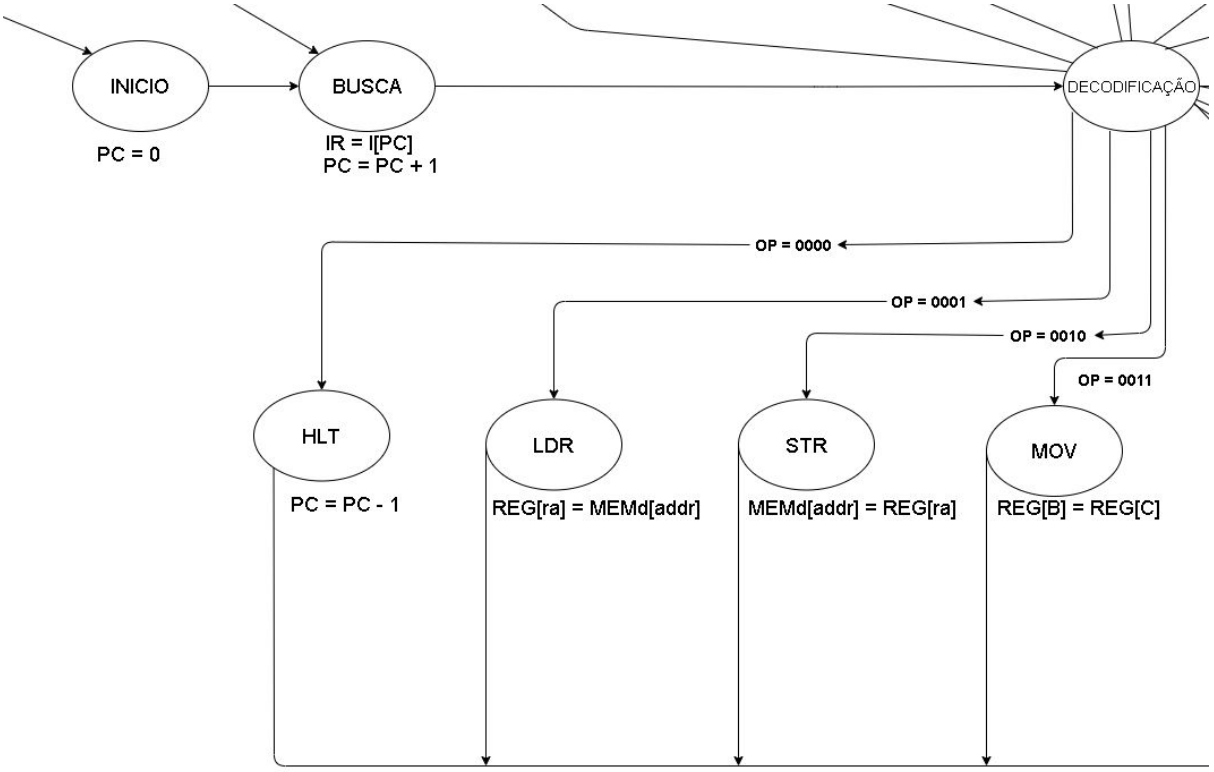


Fonte: Autores

A explicação da MDE de alto nível será dividida em três partes, na primeira estarão os estados iniciais e de dados, na segunda os estados de ULA e na terceira os estados de salto. Todos os estados que realizam operações retornam para o estado BUSCA e assim se forma um loop que busca por instruções a serem executadas até que a CPU seja desligada.

2.1.1 Descrição dos estados iniciais e estados de dados

Figura 4 - Estados iniciais e de dados



Fonte: Autores

- **INÍCIO:** o contador PC é zerado, isso acontece somente neste estado.

- **BUSCA:** o contador PC incrementa o valor um para buscar a próxima instrução e o registrador de instrução guarda os dados referentes à instrução do endereço apontado por PC.

- **DECODIFICAÇÃO:** neste estado o opcode é avaliado para decidir o próximo estado, entre eles estão todos os estados que realizam alguma das 16 instruções.

- **HLT:** executa a instrução HLT - 0000 xxxx xxxx xxxx, a transição é feita para este estado quando o opcode é igual a 0000, parando o processamento da CPU. Este estado não realiza nenhuma ação, a próxima instrução a ser carregada continua sendo a mesma para o estado HLT. A subtração do valor um será explicada depois com detalhes.
  É importante lembrar que as instruções do processador tem 16 bits, e os bits de opcode são sempre referentes à instrução no momento.

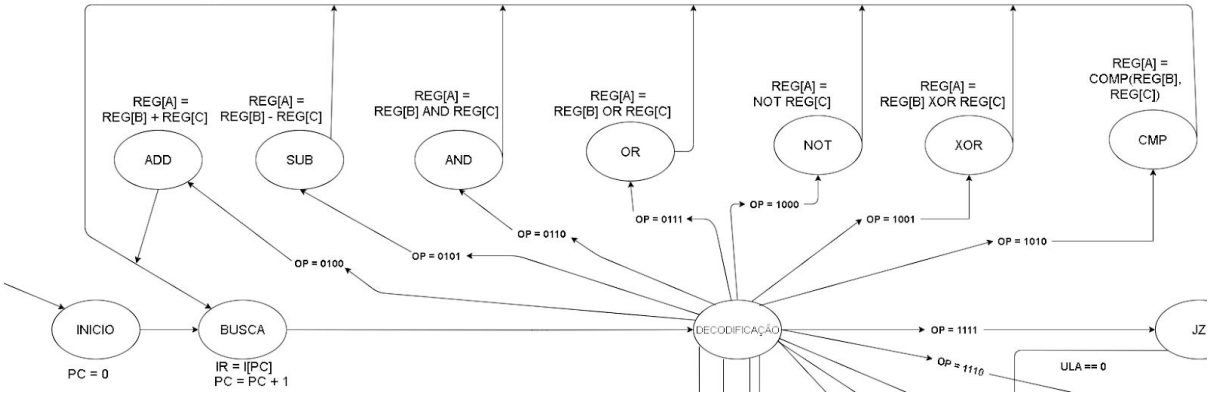
- **LDR:** instrução LDR - 0001 regA[11:8] addr[7:0]: o conteúdo de uma posição da memória de dados, especificada por addr[7:0], é carregado em um registrador do banco de registradores, especificado por regA[11:8]. Conhecida como operação de carga.

- **STR:** instrução STR - 0010 regA[11:8] addr[7:0]: o conteúdo de um registrador do banco de registradores, especificado por regA[11:8], é armazenado em uma posição da memória de dados, especificada por addr[7:0]. Conhecida como operação de armazenamento.

- **MOV:** instrução MOV - 0011 xxxx regB[7:4] regC[3:0]: o conteúdo de um registrador é movido para outro registrador.

2.1.2 Descrição dos estados de ULA

Figura 5 - Estados que utilizam a ULA



Fonte: Autores

Em todos os estados de ULA as flags carry e ULA são modificadas, a flag carry é a saída “vai-um” do somador e subtrator, sendo zero nas operações lógicas e de comparação. A flag ULA indica se o resultado de uma operação aritmética ou lógica é zero ou não.

- **ADD:** instrução ADD - 0100 regA[11:8] regB[7:4] regC[3:0]: os conteúdos de um registrador B e de um registrador C são somados e o resultado é armazenado em um registrador A.
- **SUB:** instrução SUB - 0101 regA[11:8] regB[7:4] regC[3:0]: o valor de um registrador C é subtraído de um registrador B e o resultado armazenado em um registrador A.
- **AND:** instrução AND - 0110 regA[11:8] regB[7:4] regC[3:0]: é feita uma operação AND bit a bit entre os conteúdos de dois registradores e o resultado é carregado em outro registrador.

- **OR:** instrução OR - 0111 regA[11:8] regB[7:4] regC[3:0]: é feita uma operação AND bit a bit entre os conteúdos de dois registradores e o resultado é carregado em outro registrador.

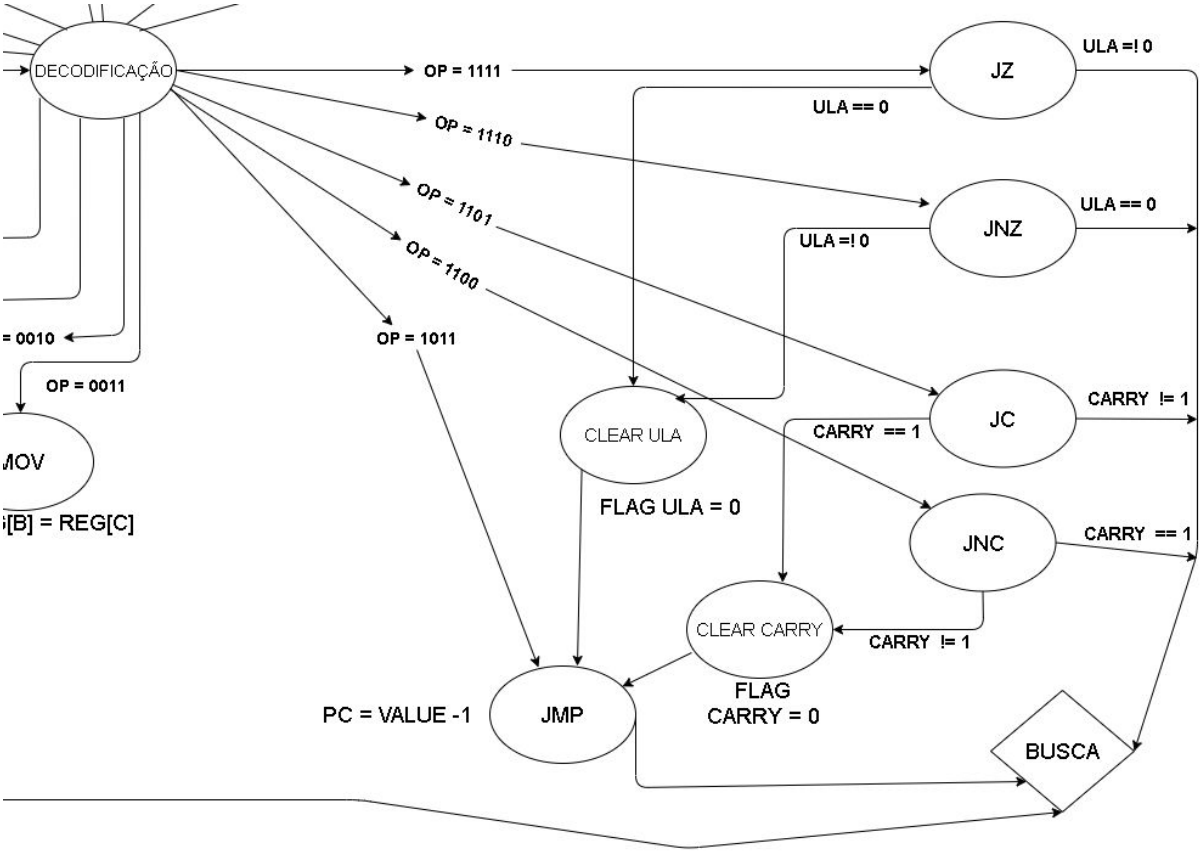
- **NOT:** instrução NOT - 1000 regA[11:8] xxxx regB[3:0]: nega todos os bits do valor de um registrador e armazena o resultado em outro registrador.

- **XOR:** instrução XOR - 1001 regA[11:8] regB[7:4] regC[3:0]: é feita uma operação XOR bit a bit entre os conteúdos de dois registradores e o resultado é carregado em outro registrador.

- **CMP:** instrução CMP - 1010 regA[11:8] regB[7:4] regC[3:0]: compara o valor de dois registradores bit a bit e guarda o resultado em outro registrador.

2.1.3 Descrição dos estados de salto

Figura 6 - Operações de salto



Fonte: Autores

As instruções de salto são usadas para indicar ao processador que a próxima instrução a executar não é a que está imediatamente a seguir mas sim outra, selecionada pelo programa que está na memória. Existem saltos condicionais que ocorrem se uma determinada condição se verificar e saltos incondicionais que ocorrem sempre, sem estarem sujeitos a qualquer condição.

- **JMP:** instrução JMP - 1011 xxxx valor[7:0]: realiza um salto incondicional para uma instrução especificada pelos 8 ultimos bits dos 16 bits de instrução, fazendo com que PC aponte para valor[7:0] e execute a instrução neste endereço de memória.

  Os próximos estados são de salto condicional, dependendo do valor de carry e ULA, se as condições forem satisfeitas é feita uma transição para o estado JMP que salta para a instrução especificada em cada estado de salto condicional.

- **JNC:** instrução JNC - 1100 xxxx valor[7:0]: realiza um salto se carry for igual a zero, apontando PC para o endereço valor[7:0].

- **JC:** instrução JC - 1101 xxxx valor[7:0]: realiza um salto se carry for igual a um, apontando PC para o endereço valor[7:0].

- **JNZ:** instrução JNZ - 1110 xxxx valor[7:0]: realiza um salto se ULA for diferente de zero, apontando PC para o endereço valor[7:0].

- **JZ:** instrução JZ - 1111 xxxx valor[7:0]: realiza um salto se ULA for igual a zero, apontando PC para o endereço valor[7:0].

- **CLEAR ULA:** ativa o clear do flip flop D que mantém o valor da flag ULA.

- **CLEAR CARRY:** ativa o clear do flip flop D que mantém o valor da flag carry.

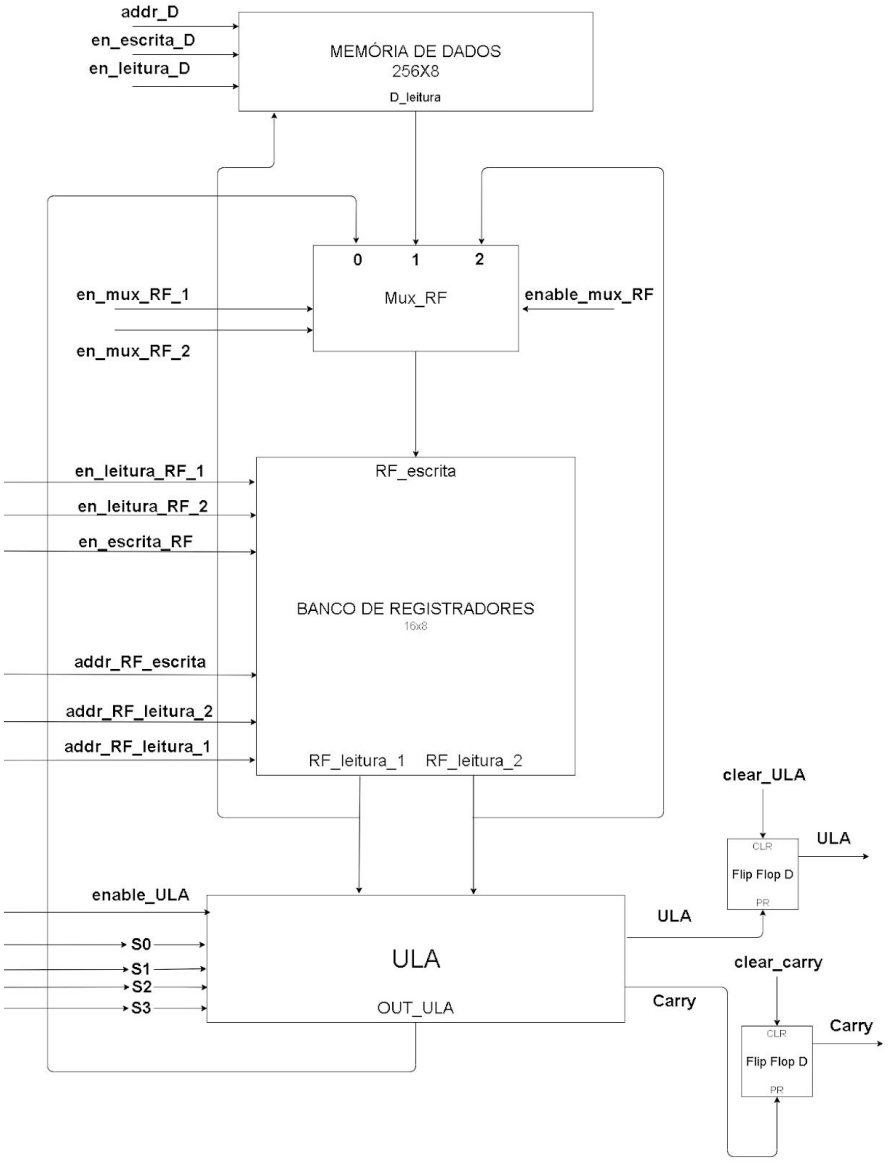
Uma observação importante a ser feita sobre a entrada do contador PC é que esta tanto pode receber a variável value das instruções de salto, como pode receber o valor atual do próprio PC quando no estado HLT. Após carregar PC com o endereço desejado a ser buscado na memória, a MDE volta para o estado BUSCA e incrementa 1 no valor de PC, dessa forma, a próxima instrução a ser buscada não seria o valor desejado mas sim, valor + 1, por isso é necessário decrementar 1 da entrada do contador PC para que isto não ocorra.

2.2 Bloco de dados

Em Vahid (2008) o segundo passo do método de projeto RTL é a construção do bloco operacional ou bloco de dados, e o terceiro passo é a conexão do bloco de dados com o bloco de controle. Nesta seção será apresentado o bloco de dados juntamente com a memória de dados.

- **Memória de dados(256x8 bits):** é necessária para armazenar os dados a serem operados pela ULA, dados esses que serão movidos ou virão do banco de registradores. A entrada addr_D indica o endereço a ser acessado, en_escrita_D habilita a escrita, en_leitura_D habilita a leitura e também há uma entrada e uma saída para os dados de 8 bits.

Figura 7 - Bloco de dados e memória de dados



Fonte: Autores

- **Multiplexador RF:** mux de três entradas que seleciona umas delas para ser entrada do banco de registradores, onde a entrada 0 é a saída da ULA, a entrada 1 é a saída da memória de dados e a entrada 2 é a saída 2 do banco de registradores.
    - ➢ en_mux_RF_1 e en_mux_RF_2 são os sinais seletores do mux, enable_mux_RF serve para ativar ou não o funcionamento do mux.

- **Banco de registradores(16x8 bits):** armazena os dados vindos da memória de dados a serem operados, assim como os resultados destas operações. Tem uma entrada que é selecionada pelo mux descrito acima e duas saídas de dados, onde as saídas serão entradas da ULA, para as operações aritméticas e lógicas serem feitas.
    - ➢ Para a escrita no banco de registradores, o sinal en_escrita_RF habilita a mesma e addr_RF_escrita indica em qual registrador será feita a escrita. A entrada en_leitura_RF_1 habilita a leitura de um registrador na posição que addr_RF_leitura_1 indica, en_leitura_RF_2 e addr_RF_leitura_2 tem a mesma função, mas para a saída 2 do banco de registradores.

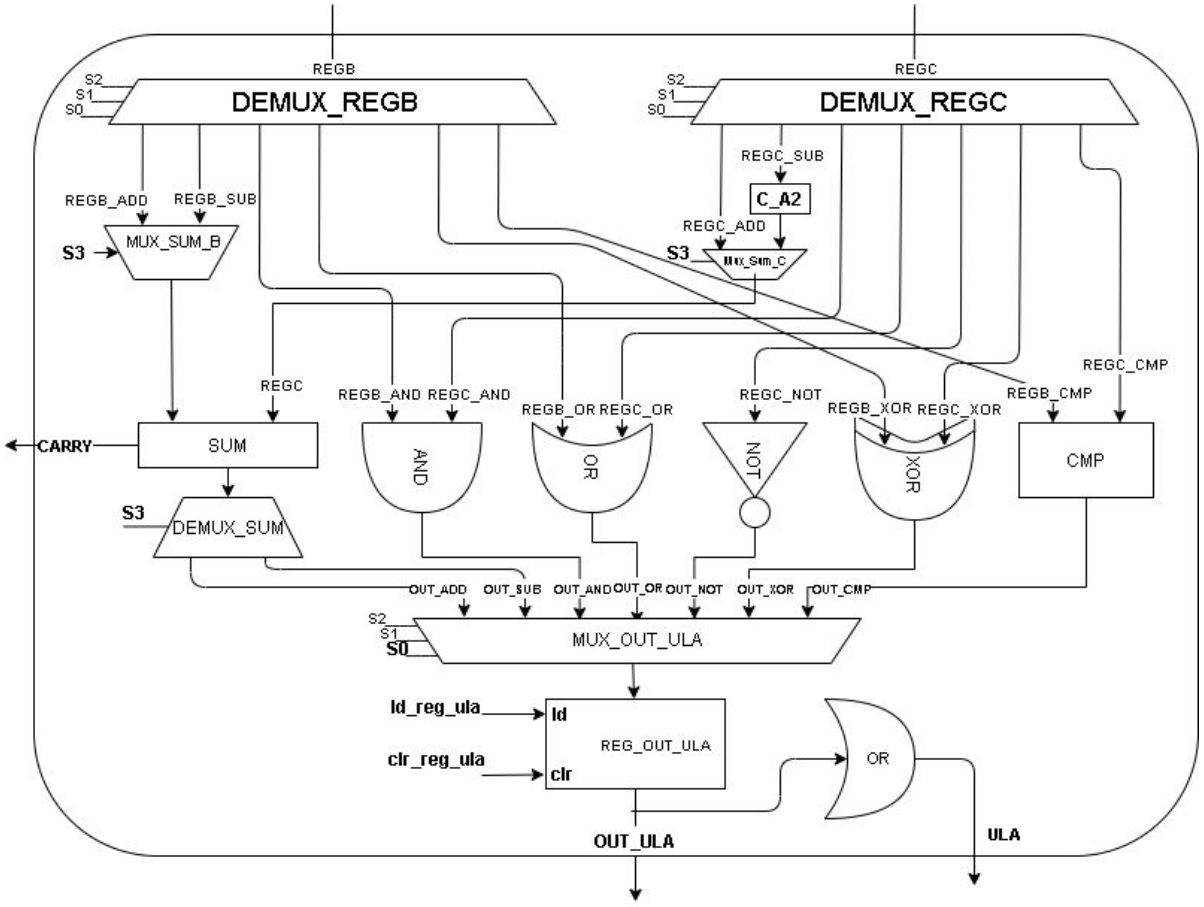
- **Flip Flops D:** há dois flip flops D com presset e clear, sendo que cada entrada presset recebe o valor das flags carry e ULA, estes flip flops mantém o valor das duas flags ativados, e são zerados com o clear quando houver uma instrução de salto.

2.2.1 Unidade Lógica e Aritmética (ULA)

Dentro do bloco da ULA é onde acontecem as operações de soma, subtração, AND, OR, XOR, NOT e de comparação, também é o bloco responsável por modificar as flags de carry e ULA. De acordo com a figura 7 a ULA tem duas entradas de dados vindas do banco de registradores, quatro entradas de seleção S0, S1, S2 e S3 e uma entrada que habilita ou não todos os multiplexadores e demultiplexadores ao mesmo tempo, chamada de enable_ULA. Uma das saídas é o resultado da operação realizada entre os dois dados de entrada, as outras duas são as flags carry e ULA.

De acordo com a figura 8, as duas entradas de dados da ULA passam cada uma por um demultiplexador (demux), que tem o papel de ramificar as duas entradas para a operação escolhida de acordo com os seletores do demux S0, S1 e S2. O sinal S3 é uma combinação lógica dos sinais seletores do demux, combinação essa descrita mais à frente, e que serve como seletor do MUX_SUM_B, MUX_SUM_C e DEMUX_SUM, onde estes três últimos selecionam as entradas para soma ou subtração.

Figura 8 - Unidade Lógica e Aritmética



Fonte: Autores

- **Operação de soma:** uma das entradas do somador vem do MUX_SUM_B, que no caso de soma seleciona a primeira saída do DEMUX_REGB, chamada de REGB_ADD. A outra entrada vem do MUX_SUM_C, que seleciona a primeira saída do DEMUX_REGC, chamada de REGC_ADD. Com o resultado da soma, o bit de “vai-um” do somador é exatamente a flag carry que sai da ULA. Esse resultado passa pelo DEMUX_SUM que seleciona a primeira saída, referente a operação de soma.
- **Operação de subtração:** essa operação usa o mesmo somador usado na operação de soma, com a diferença de que o valor a ser subtraído entra no somador com sinal negativo. A primeira entrada vem do MUX_SUM_B, que no caso de subtração seleciona a segunda saída do DEMUX_REGB, chamada de REGB_SUB. A outra entrada vem do MUX_SUMC, que seleciona a segunda saída do DEMUX_REGC após passar por um bloco de complemento A2 para tornar o número negativo e assim somar um número positivo com um negativo e realizar a subtração.
- **Operação AND:** a terceira saída dos demux de cada entrada vai para uma porta lógica AND, operação esta feita bit a bit.
- **Operação OR:** a quarta saída dos demux de cada entrada vai para uma porta lógica OR, operação esta feita bit a bit.

- **Operação NOT:** uma porta lógica NOT tem como entrada a quinta saída do DEMUX_REGC, denominada REGC_NOT, negando todos os bits desse sinal. Somente a segunda saída do banco de registradores pode ser negada dentro da ULA.
- **Operação XOR:** a primeira entrada da porta lógica XOR vem da quinta saída do DEMUX_REGB, denominada REGB_XOR, a segunda entrada vem da sexta saída do DEMUX_REGC, chamada de REGC_XOR. A operação XOR é feita bit a bit assim como nas outras operações lógicas.
- **Operação de comparação:** usa-se um comparador de igualdade para comparar as duas entradas bit a bit. A primeira entrada vem da sexta saída do DEMUX_REGB, chamada de REGB_CMP, a segunda entrada vem da sétima saída do DEMUX_REGC, chamada de REGC_CMP.

Os sete resultados das sete operações realizadas dentro da ULA são entradas de um multiplexador chamado de MUX_OUT_ULA 7x1 de 8 bits, que através de S0, S1 e S2 seleciona qual resultado vai ser repassado para a saída da ULA. Essa saída do mux é a entrada de um registrador, que mantém o valor da última operação feita, a mesma saída também passa por uma porta lógica OR para fazer a flag ULA ser ou não igual a zero. É especificado no projeto que ocorre a transição para o estado JZ se ULA = 0, significando que o resultado da última operação foi zero.

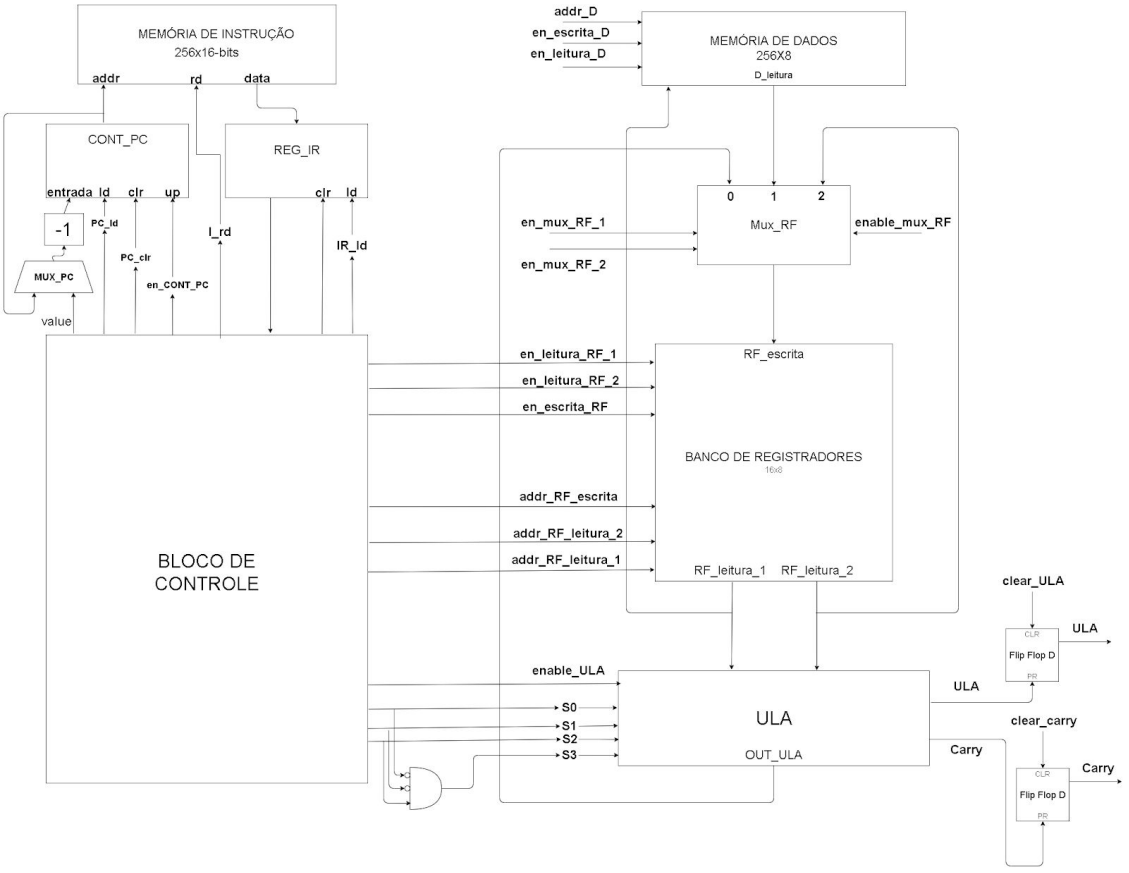
Uma observação importante sobre a ULA é que para todos os componentes com clock dentro da mesma, este clock deverá ser 5 vezes superior ao do resto do circuito, visto que as operações feitas na ULA podem levar mais de um ciclo de relógio para serem realizadas, a escolha de 5 vezes maior é para garantir que todas as operações serão feitas a tempo.

As lógicas de seleção de de cada mux e demux que tem como seletores S0, S1 e S3 estão descritas nas tabelas do anexo A.

2.3 Conexão do bloco de dados com a unidade de controle

O terceiro passo do método de projeto RTL é a conexão do bloco de dados com o bloco de controle, neste caso a parte de controle será não só o bloco de controle em si, mas também o contador PC, o registrador IR e mais dois componentes lógicos que têm relação com a entrada do PC, além da memória de instrução.

Figura 9 - Unidade de controle e bloco de dados

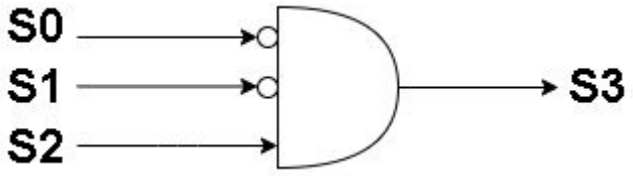


Fonte: Autores

- **Contador PC:** é um contador crescente com carga paralela, pode ter duas entradas possíveis, quando no estado HLT a entrada será o próprio valor atual de PC, quando no estado JMP(estado de salto) a entrada será value. Para a escolha de uma das duas entradas foi usado um multiplexador 2x1, e antes do valor escolhido entrar no contador ele passa por um subtrator de 1 por conta do incremento de 1 no estado BUSCA como explicado anteriormente.
- **Memória de instrução:** cada posição da memória guarda os 16 bits referentes a cada instrução, onde o endereço a ser acessado é apontado por PC. A saída são os 16 bits da instrução escolhida e que é armazenado no registrador IR.
- **Registrador IR:** registrador de instrução usado para armazenar a instrução vinda da memória de instrução, sua saída vai para o bloco de controle para que de acordo com os bits da instrução controla as operações no bloco de dados.

- **Seletor S3:** usado para selecionar a saída do MUX_SUM_B, MUX_SUM_C e DEMUX_SUM, é uma combinação dos seletores S0, S1 e S2, dada por S3 = (S0’ AND S1’ AND S2). Essa combinação é mostrada na figura 10 abaixo.

Figura 10 - Sinal de controle S3

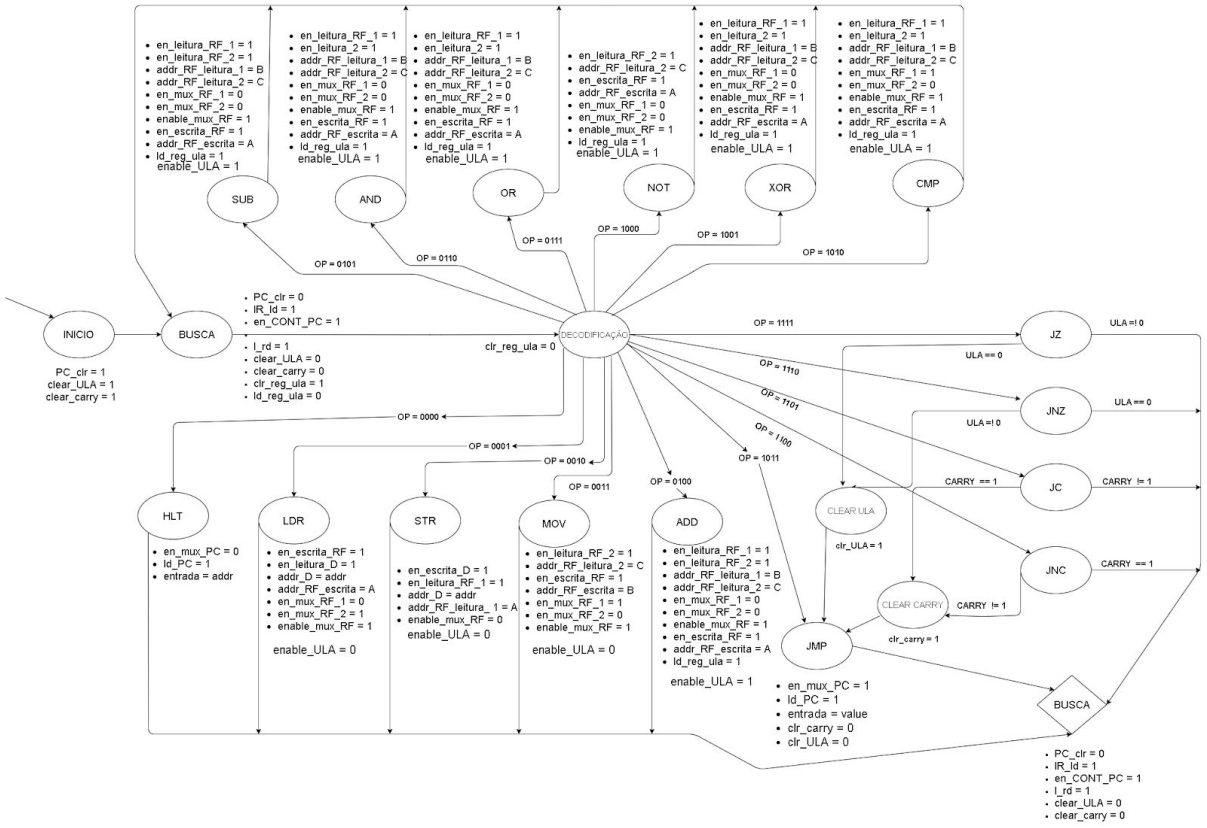


Fonte: Autores

2.4 Máquina de estados de baixo nível

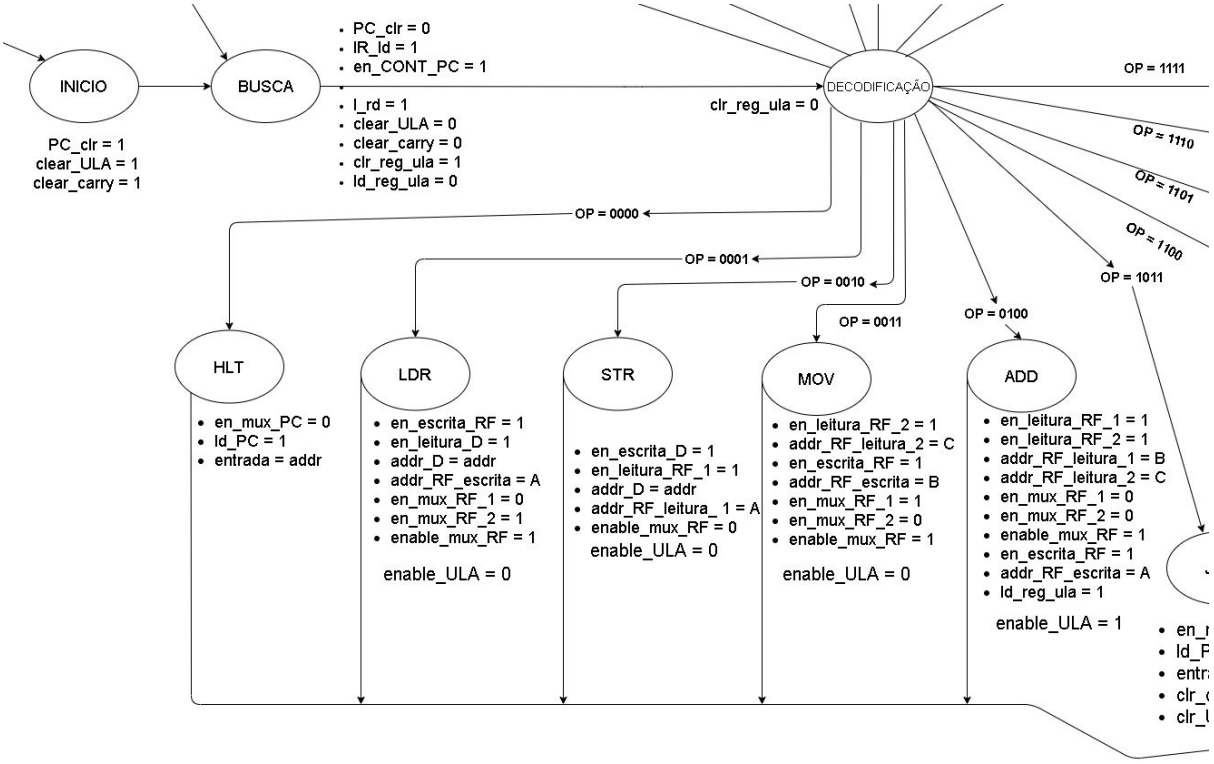
A MDE de baixo nível completa está descrita na figura 10, os sinais de controle de cada estado serão explicados abaixo com mais detalhes, indicando o que cada um faz.

Figura 11 - MDE de baixo nível



Fonte: Autores

Figura 12 - Parte 1 da MDE de baixo nível



Fonte: Autores

- **INÍCIO:**

➔ PC_clr = 1: zera o contador PC para as instruções começarem da posição 0.
➔ clear_ULA = 1: zera o flip-flop D usado para manter o valor da flag ULA.
➔ clear_carry = 1: zera o flip-flop D usado para manter o valor da flag carry.

- **BUSCA:**

➔ PC_clr = 0: desativa o clear do contador PC, pois neste estado ele começa a contagem de PC.
➔ IR_ld = 1: load no registrador de instrução IR com o valor vindo da memória de instrução.
➔ en_CONT_PC = 1: inicia a contagem do contador PC.
➔ I_rd = 1: habilita a leitura na memória de instrução.
➔ clear_ULA = 0: desativa o clear do flip-flop D usado para manter o valor da flag ULA.
➔ clear_carry = 0: desativa o clear do flip-flop D usado para manter o valor da flag carry.
➔ clr_reg_ula = 1: zera o registrador que armazena o resultado da ULA.
➔ ld_reg_ula = 0: desabilita o mesmo registrador.

- **DECODIFICAÇÃO:**

➔ clr_ld_ula = 0: desabilita o clear do registrador na saída da ULA.

- **HLT:**

➔ en_MUX_PC = 0: seleciona a entrada zero do MUX_PC antes do contador PC como sendo a saída do próprio contador PC.

➔ ld_PC = 1: ativa a carga paralela do contador PC carregando-o com o valor na sua entrada.

➔ entrada = addr: a entrada do contador PC recebe o valor vindo da entrada zero do MUX_PC subtraido de.

- **LDR:**

➔ en_escrita_RF = 1: habilita a escrita no banco de registradores.

➔ en_leitura_D = 1: habilita a leitura na memória de dados.

➔ addr_D = addr: a entrada da memória de dados recebe o endereço vindo do bloco de controle a ser acessado de acordo com a instrução.

➔ addr_RF_escrita = A: a entrada de endereço de escrita do banco de registradores recebe o endereço do registrador a ser escrito de acordo com os bits de instrução.

➔ en_mux_RF_1 = 0 e en_mux_RF_2 = 1: seleciona a entrada 1 do MUX_RF, permitindo a passagem do valor que vem da memória de dados para o banco de registradores.

➔ enable_mux_RF = 1: habilita o funcionamento do MUX_RF.

➔ enable_ULA = 0: desabilita o funcionamento de todos os multiplexadores e demultiplexadores na ULA, já que a mesma não é utilizada neste estado.

- **STR:**

➔ en_escrita_D = 1: habilita a escrita na memória de dados.

➔ en_leitura_RF_1 = 1: habilita a leitura de um registrador na saída 1 do banco de registradores.

➔ addr_D = addr: a entrada da memória de dados recebe o endereço vindo do bloco de controle a ser acessado de acordo com a instrução.

➔ addr_RF_leitura _1 = A: o endereço de um registrador é passado a entrada de endereço para leitura 1 do banco de registradores.

➔ enable_mux_RF = 0: desabilita o funcionamento do MUX_RF, já que os dados do registrador vão direto para a memória de dados.

➔ enable_ULA = 0: desabilita o funcionamento de todos os multiplexadores e demultiplexadores na ULA, já que a mesma não é utilizada neste estado.

- **MOV:**

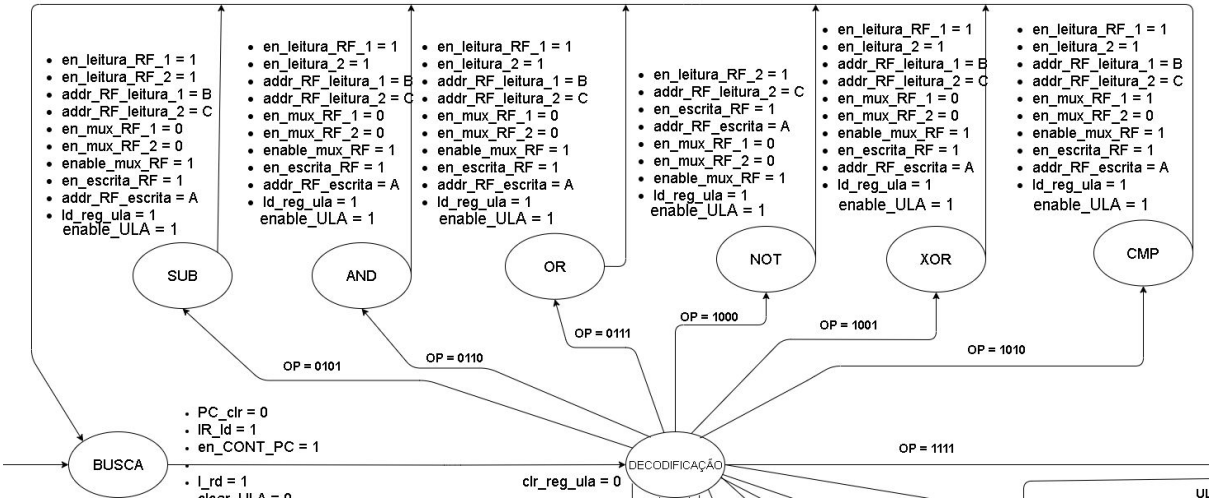
➔ en_leitura_RF_2 = 1: habilita a leitura de um registrador na saída 2 do banco de registradores.

- ➔ addr_RF_leitura_2 = C: o endereço de um registrador é passado a entrada de endereço para leitura 2 do banco de registradores.
- ➔ en_escrita_RF = 1: habilita a escrita no banco de registradores.
- ➔ addr_RF_escrita = B: a entrada de endereço de escrita do banco de registradores recebe o endereço do registrador a ser escrito de acordo com os bits de instrução.
- ➔ en_mux_RF_1 = 1 e en_mux_RF_2 = 0: seleciona a entrada 2 do MUX_RF, permitindo a passagem do valor que vem da saída 2 do banco de registradores.
- ➔ enable_mux_RF = 1: habilita o funcionamento do MUX_RF.
- ➔ enable_ULA = 0: desabilita o funcionamento de todos os multiplexadores e demultiplexadores na ULA, já que a mesma não é utilizada neste estado.

- **ADD:**

- ➔ en_leitura_RF_1 = 1: habilita a leitura de um registrador na saída 1 do banco de registradores.
- ➔ en_leitura_RF_2 = 1: habilita a leitura de um registrador na saída 2 do banco de registradores.
- ➔ addr_RF_leitura_1 = B: o endereço de um registrador(que vem dos bits de instrução) é passado para a entrada de endereço para leitura 1 do banco de registradores.
- ➔ addr_RF_leitura_2 = C: o endereço de outro registrador(que vem dos bits de instrução) é passado para a entrada de endereço para leitura 2 do banco de registradores.
- ➔ en_mux_RF_1 = 0 e en_mux_RF_2 = 0: seleciona a entrada 0 do MUX_RF, permitindo a passagem do resultado que vem da ULA para o banco de registradores.
- ➔ enable_mux_RF = 1: habilita o funcionamento do MUX_RF.
- ➔ en_escrita_RF = 1: habilita a escrita no banco de registradores.
- ➔ addr_RF_escrita = A: A entrada de endereço de escrita do banco de registradores recebe o endereço do registrador a ser escrito com o resultado da ULA.
- ➔ enable_ULA = 1: habilita o funcionamento de todos os multiplexadores e demultiplexadores na ULA.
- ➔ s0 = 0.
- ➔ s1 = 0.
- ➔ s2 = 0.
- ➔ s3 = s0's1's2.

Figura 13 - Parte 2 da MDE de baixo nível



Fonte: Autores

- **SUB:** todos os sinais modificados no estado ADD têm as mesmas modificações neste estado. Com exceção de:
  - ➔ s0 = 0.
  - ➔ s1 = 0.
  - ➔ s2 = 1.
  - ➔ s3 = s0's1's2.

- **AND:** todos os sinais modificados no estado SUB têm as mesmas modificações neste estado. Com exceção de:
  - ➔ s0 = 0.
  - ➔ s1 = 1.
  - ➔ s2 = 0.
  - ➔ s3 = s0's1's2.

- **OR:** todos os sinais modificados no estado AND têm as mesmas modificações neste estado. Com exceção de:
  - ➔ s0 = 0.
  - ➔ s1 = 1.
  - ➔ s2 = 1.
  - ➔ s3 = s0's1's2.

- **XOR:** todos os sinais modificados no estado OR têm as mesmas modificações neste estado. Com exceção de:
  - ➔ s0 = 1.

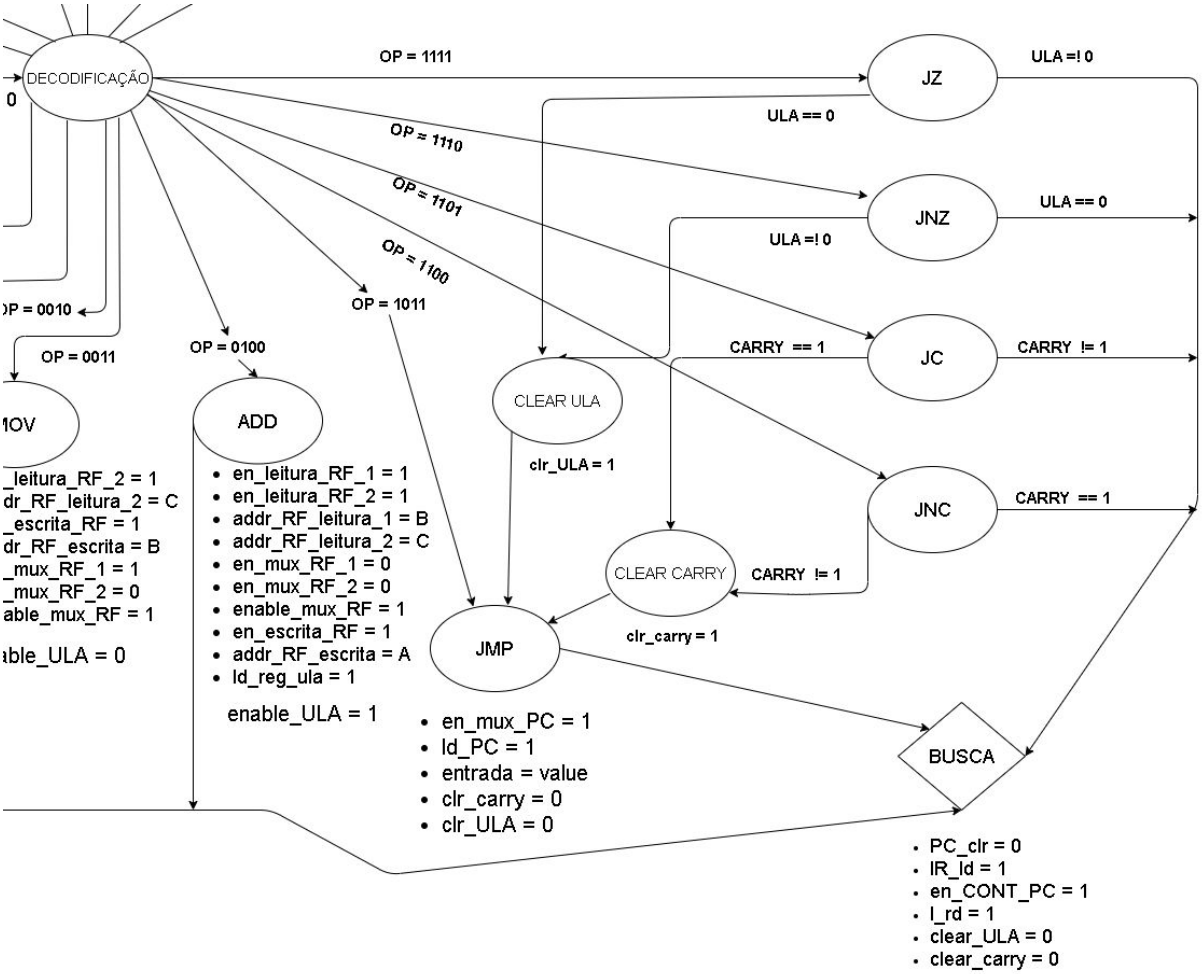
- ➔ s1 = 0.
- ➔ s2 = 1.
- ➔ s3 = s0's1's2.

- **NOT:**

➔ en_leitura_RF_2 = 1: habilita a leitura de um registrador na saída 2 do banco de registradores.
➔ addr_RF_leitura_2 = C: o endereço de um registrador é passado a entrada de endereço para leitura 2 do banco de registradores.
➔ en_escrita_RF = 1: habilita a escrita no banco de registradores.
➔ addr_RF_escrita = A: a entrada de endereço de escrita do banco de registradores recebe o endereço do registrador a ser escrito com o resultado da ULA.
➔ en_mux_RF_1 = 0 e en_mux_RF_2 = 0: seleciona a entrada 0 do MUX_RF, permitindo a passagem do resultado que vem da ULA para o banco de registradores.
➔ s0 = 1.
➔ s1 = 0.
➔ s2 = 0.
➔ s3 = s0's1's2.

- **CMP:** todos os sinais modificados no estado XOR têm as mesmas modificações neste estado. Com exceção de:
  - ➔ s0 = 1.
  - ➔ s1 = 1.
  - ➔ s2 = 0.
  - ➔ s3 = s0's1's2.

Figura 14 - Parte 3 da MDE de baixo nível



Fonte: Autores

- **JMP:**

➔ en_mux_PC = 1: seleciona a entrada 1 do MUX_PC para que o valor do endereço escolhido seja acessado na memória de dados.

➔ ld_PC = 1: ativa a carga paralela do contador PC carregando-o com o valor na sua entrada.

➔ entrada = value: o contador PC é carregado com o valor do endereço a ser acessado.

➔ clr_carry = 0: desabilita o clear do flip-flop D que mantém o valor da flag carry.

➔ clr_ULA = 0: desabilita o clear do flip-flop D que mantém o valor da flag ULA.

- **JZ:** não modifica nenhum sinal de controle.

- **JNZ:** não modifica nenhum sinal de controle.

- **JC:** não modifica nenhum sinal de controle.

- **JNC:** não modifica nenhum sinal de controle.

- **CLEAR ULA:**

➔ clr_ULA = 1: zera o flip-flop D que mantém o valor da flag ULA.

- **CLEAR CARRY:**

➔ clr_carry = 1: zera o flip-flop D que mantém o valor da flag carry.

3 Conclusão

Dadas as pesquisas feitas utilizando as referências de sistemas digitais e arquitetura de computadores, foi possível esquematizar um projeto de uma CPU. Tal projeto de CPU constitui de partes fundamentais, os quais foram comentados neste relatório, estas partes demonstraram ser eficientes para a solução do projeto de uma CPU.

Como fonte principal de pesquisa, o livro de Sistemas Digitais foi imprescindível para a formulação do projeto, tal que boa parte do projeto já estava feita no livro, mas como o projeto deste relatório em específico foi utilizado 16 instruções houve a necessidade de acrescentar mais detalhes para que o projeto se adequa-se às necessidade. E esses ajustes aparentam ser o suficiente para o funcionamento do esquemático. Um grande exemplo disso foi a utilização de registradores para armazenar os valores da flag ULA do carry, para que as operações de jump funcionem como esperado.

Tendo em vista o escopo desse projeto, o mesmo trabalha com programas no nível de código de máquina, que é composto somente por 0s e 1s, e para que a construção de um programa a nível de um humano fosse feito seria necessário um assembler, que é basicamente outro programa que “traduz” símbolos da linguagem assembly para um código de máquina. Sendo assim, este projeto abordou a estrutura mais baixa de uma CPU que foi a montagem de uma unidade de controle e de um bloco de dados, assim como o uso de memórias para armazenar dados e instruções.

Então, por fim, podemos dizer que a arquitetura do projeto ocorreu como previsto nas pesquisas sobre Sistemas Digitais e Arquitetura de Computadores.

Referências

STALLINGS, W. Arquitetura e Organização de Computadores: 8. ed. São Paulo: Pearson Education do Brasil, 2010

VAHID, F. “Digital Design”, 1. ed. Editora Bookman, 2006

Anexos

Anexo A

MUX_RF	
CÓDIGOS	**ENTRADA SELECIONADA**
00	D_leitura
01	RF_leitura_2
10	Saída da ULA
11	XXXXXXXXXX

DEMUX_REGB	
CÓDIGOS	**SAÍDA SELECIONADA**
000	ADD
001	SUB
010	AND
011	OR
100	NOT
101	XOR
110	CMP

DEMUX_REGC	
CÓDIGOS	**SAÍDA SELECIONADA**
000	ADD
001	SUB
010	AND
011	OR
100	NOT
101	XOR
110	CMP

DEMUX_OUT_ULA	
CÓDIGOS	**SAÍDA SELECIONADA**
000	OUT_ADD
001	OUT_SUB
010	OUT_AND
011	OUT_OR
100	OUT_NOT
101	OUT_XOR
110	OUT_CMP

DEMUX_SUM B / DEMUX SUM C/ MUX SUM		
CÓDIGOS		**SAÍDA SELECIONADA**
0	S3 = S0’S1’S2	ADD
1		SUB

Fonte:Elaboração Própria

ANEXO B - Relato Semanal

Líder: Rafael Capuano

A.1 Equipe

Função:	**Nome completo do aluno**
Redator:	Luiz Vitor Clementino
Debatedor:	Erika Costa Alves
Videomaker:	Lucas Gualberto Santos Ribeiro
Auxiliar:	Arthur Felipe Rodrigues Costa

A.2 Problema

O projeto se resume a projetar um processador de 8 bits, de 16 operações,dada nossa experiência em circuitos digitais, as operações aritméticas foram fáceis de projetar.

Em se tratando dos problemas, o problema mais trabalhoso que tivemos que resolver foi basicamente o fato das funções jump que foram pedidas no projeto, fora isso o problema foi razoavelmente simples.

A.3 Registro de Brainstorm

O brainstorm ocorreu continuamente em todas as reuniões que tivemos, as opiniões foram muito similares entre os membros do grupo e a cada nova reunião

foram repassadas as pesquisas de cada membro da equipe, assim como soluções para algumas partes do projeto.

Os debates correram bem, todos os membros do grupo participaram de todas as reuniões, tivemos ideias similares de como resolver o problema dos estados Jump,descritos no relatório.

A.4 Pontos Chaves

O ponto chave do problema se resumia em como projetar a função Jump de forma eficiente,o resultado descrito no relatório foi um consenso chegado pelo grupo.

A.5 Planejamento da pesquisa

A pesquisa foi feita através do livro de Frank Vahid,não foi necessário um planejamento, dado que cada integrante e sua função trabalhou de maneira independente.