UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

ELE1717 - SISTEMAS DIGITAIS

PROCESSADOR 16 INSTRUÇÕES

PROJETO DE PROCESSADOR DE USO GERAL

Discente:

ALISSON GABRIEL
KAIKE CASTRO
MATEUS DE ASSIS SILVA
WILLIAN MOURA

Docente:
SAMAHERNI MORAIS

Natal - RN
2021

RESUMO

O presente trabalho visa o desenvolvimento esquemático de um processador de 16 instruções em conformidade com orientações impostas pelo Problema 02. O circuito integrado foi construído a partir do projeto de Nível de Transferência entre Registradores (RTL - inglês : *Register Transfer Level*), desta forma obtém-se uma melhor organização dos elementos de entrada/saída, além de favorecer um compreensão detalhada de cada processo. Foi elaborado uma máquina de estados de alto nível que implementa todos os estados e sinais que orientam a construção e simulação desse processador. No bloco de operação, foi detalhado como deu-se da unidade lógica e arimética que atende o problema posto, com isso todas as setes operações podem ser realizadas. Portanto, esse projeto de processador trará um noção do comportamento de um circuito integrado amplamente utilizado em computadores, *smartphones* e entre outros equipamentos eletrônicos.

Sumário

Lista de Figuras

1 Introdução

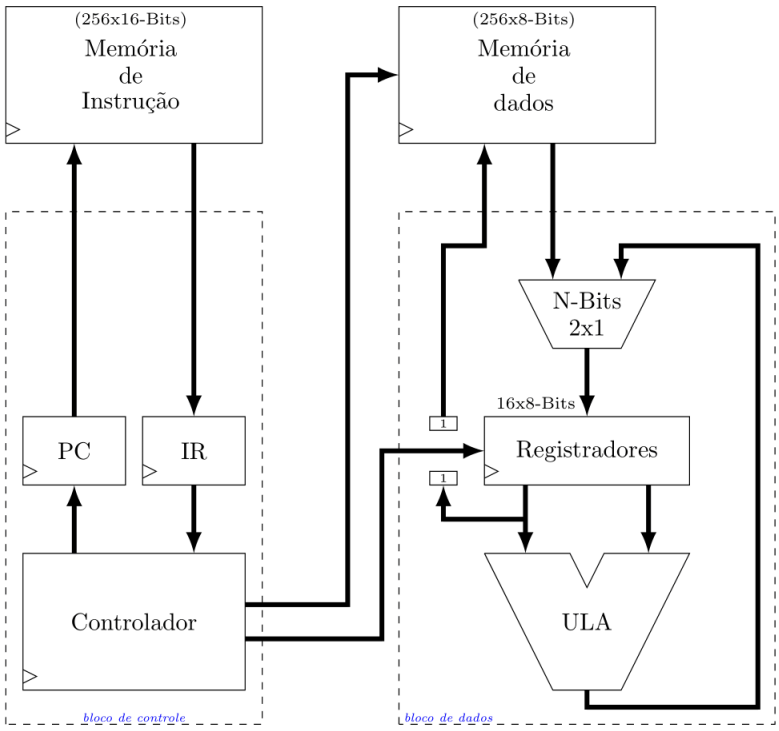
O projeto da primeira e segunda semanas se tratava de um processador de propósito único cujo foco era a tarefa de controlar uma Secretária Eletrônica. Apesar de energeticamente eficientes e rápidos, esses processadores de processo único não são flexíveis (não podem ser utilizados em outras funções) e são construídos apenas para aplicações específicas.

Essa semana o projeto objetiva a construção de um processador de uso geral (programável). Nesse tipo de processador, a tarefa de processamento fica numa memória específica denominada memória de instrução, ao invés de ser implementada no próprio circuito digital.

Os processadores programáveis são circuitos eletrônicos os quais são, do ponto de vista físico, rápidos e eficientes. A estrutura básica de um processador é composta por registradores de controle/estado que podem estar ou não visíveis para o usuário, desta forma armazenando números de ponto fixo ou flutuante, endereços e ponteiros de segmento caracterizando assim o tipo de uso: geral e específico.

O circuito integrado projetado para este trabalho, cuja função é executar 16 instruções, possui uma arquitetura conforme a Figura 1. Este processador de uso geral realizará as rotinas pré-definidas descritas na Tabela 1.

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



Fonte: material suporte

Tabela 1: Conjunto de instruções da CPU de uso geral.

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 suporte

2 Desenvolvimento

Segundo o livro *Arquitetura e Organização de Computadores* (Stallings [3], Capítulo 12), para fazer rotinas, deve estar claro que o processador precisa armazenar alguns dados temporariamente. Ele deve lembrar a posição da última instrução executante para que possa saber onde obter a próxima instrução a ser executada. O bloco de controle do processador precisa armazenar instruções e dados temporariamente enquanto uma instrução está sendo executada. Em outras palavras, o processador precisa de uma pequena memória interna, chamada memória de instrução.

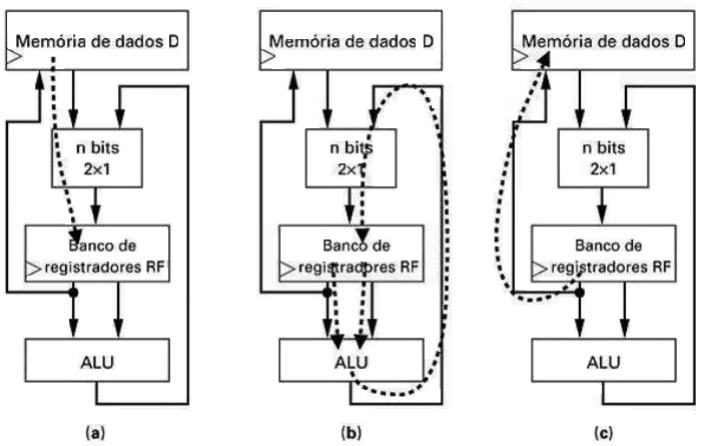
O bloco de controle possui dois registradores, os quais são fundamentais para um processador: o registrador Contador de Programa (PC - *inglês: Program Counter*) e o Registrador de Instruções (IR - *inglês: Instruction Register*). O PC é um contador crescente que aponta para a instrução corrente. Estas estão em sequência, começando com PC = 0, de modo que a instrução em I[0] representa a primeira instrução do programa. O IR, por sua vez, é um registrador local que armazena a instrução lida recentemente.

O controlador realizará a leitura da instrução da memória de instruções para então a executar no bloco operacional. A execução de qualquer programa passa por estágios, que são:

- **Busca:** O processador lê uma instrução da memória (registrador, cache, memória principal).
- **Decodificação:** A instrução é interpretada para determinar qual ação é requerida.
- **Execução:** Realizar uma instrução pode requerer efetuar alguma operação aritmética ou lógica com os dados.

No bloco operacional, o principal componente é a Unidade Lógica e Aritmética (ALU - *inglês: Arithmetic Logic Unit*), responsável por realizar os cálculos necessários (ou processamento de dados). Outro elemento a se mencionar é a memória de dados que armazena todos os dados que um processador poderá acessar, como as informações de entrada e saída. Para processar esses dados será necessário carregar os dados da memória em um banco de registrador que está no bloco operacional. A Unidade de Controle controla a movimentação de dados e das instruções que entram e saem do processador, além de controlar a operação de ALU. Portanto, a dinâmica de funcionamento do operacional segue o fluxo da Figura 17 (VAHID, 2009, p. 442) [4].

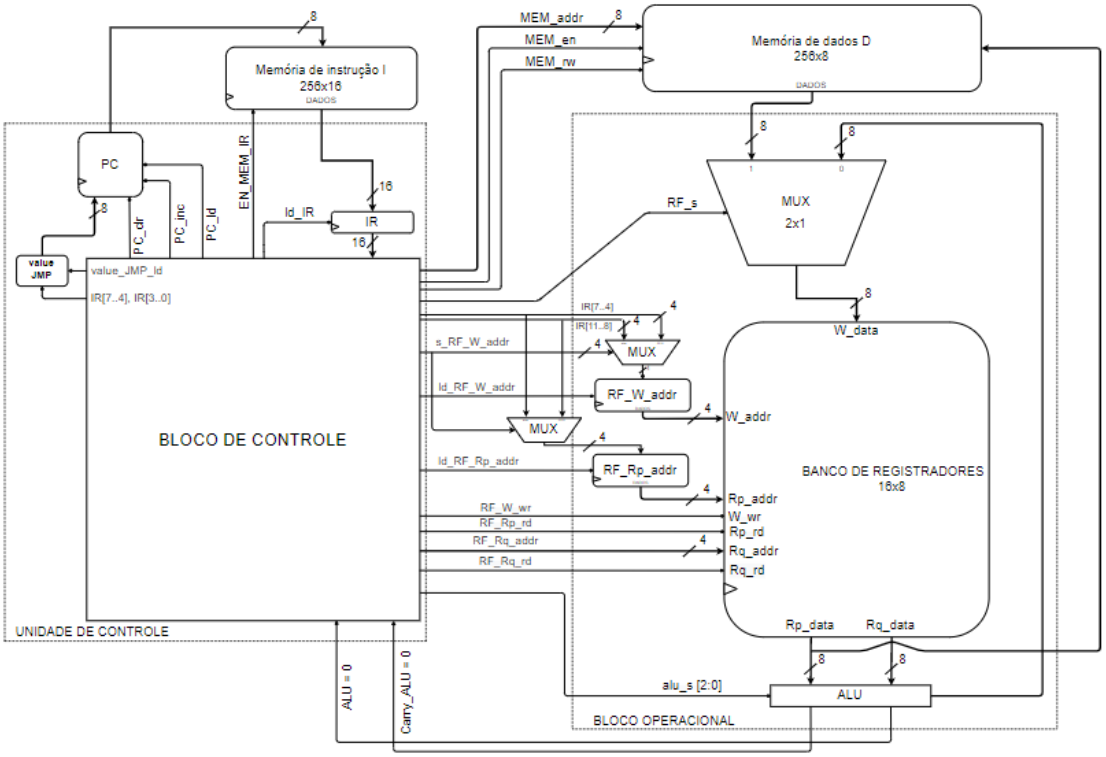
Figura 2: Operações Básicas do Bloco Operacional: (a) Carga (leitura), (b) operações (transformações), (c) armazenamento (escrita).



Fonte: Vahid, Sistemas Digitais (2009)

A Figura 3 mostra a visão geral do processador de 16 instruções. Nela se vê o Bloco de Controle, melhor compreendido com a Máquina de Estados (MDE) descrito ao longo do tópico Unidade de Controle, a seguir.

Figura 3: Projeto em RTL



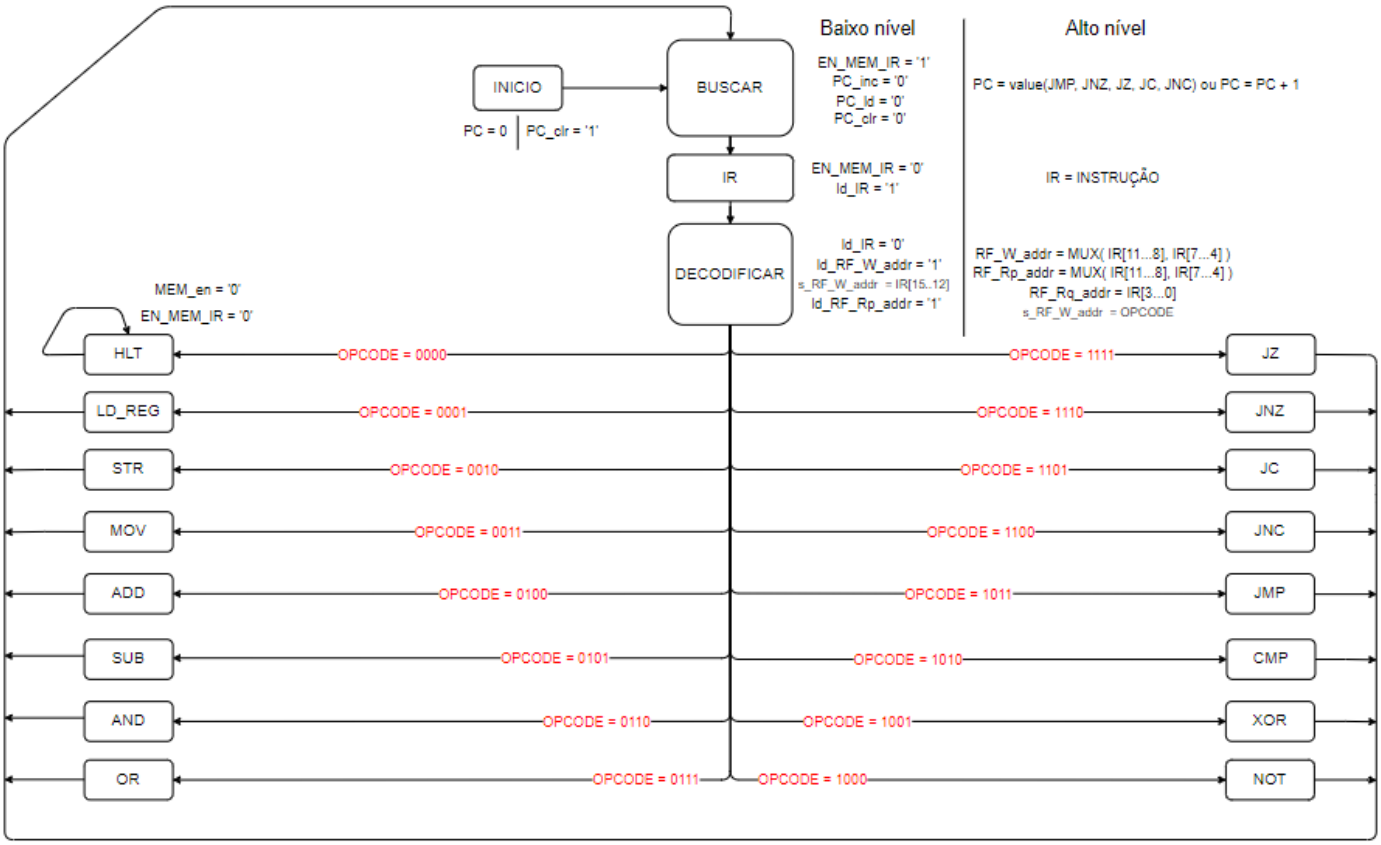
Fonte: os autores

2.1 Unidade de Controle

A Máquina de Estados (MDE) de alto nível, mostrada na Figura 4, melhor detalha o funcionamento do processador. Assuma que os OPCODEs são *nibbles* (valores binários de 4 dígitos) que especificam operações em uma das seguintes categorias gerais: operações aritméticas e lógicas; movimentação de dados entre dois registradores, registrador e memória, ou dois locais de memória; e controle.

Assim, no estado **INICIO** o PC é zerado e, em seguida, é direcionado ao estado **BUSCAR** no qual se transfere a instrução da memória de instruções para sua saída. Para isso, é liberado o acesso para memória em EN_MEM_IR = '1' e no próximo ciclo de clock é armazenado a instrução no registrador de instrução (IR). A seguir em **DECODIFICAR**, a instrução será interpretada para determina qual será a operação, o que acontece em um ciclo de relógio. Nesse mesmo estado, ld_RF_W_addr = '1' e ld_RF_Rp_addr = '1' são sinais que habilitam a alocação do endereço de entrada do bloco de registradores e sua primeria saída nos registradores RF_W_addr e RF_Rp_addr, respectivamente. O valor RF_Rq_addr é referenciado diretamente, através de barramento, do valor do IR, IR[3..0] especificadamente. O sinal s_RF_W_addr recebe o OPCODE e controla os dois multiplexadores MUX para a seleção dos endereços de entrada do banco de registradores e sua saída Rp, pois, quando a máquina está no estado **MOV**, o RF_W_addr receberá o valor de B (IR[7..4]) de acordo com o roteiro, e não o de A (IR[11..8]), como no estado **LD_REG**; enquanto o RF_Rp_addr receberá o valor A, no estado **STR** e nos demais, receberá o valor B. Após cada estado de execução de uma operação, a máquina de estado retorna ao estado **BUSCAR**, salvo no estado **HTL**, onde o fluxo de obtenção de operações é interrompido e há a inviabilização da manipulação das memórias de instrução e de dados.

Figura 4: Máquina de Estados de Alto Nível



Fonte: os autores

É necessário visualizar que em todos os estados após **DECODIFICAR** haverá a mudança de nível lógico nos sinais ld_RF_W_addr e ld_Rp_W_addr para ’0’, pois não é mais necessária, na instrução atual, a aplicação de entrada nos registradores RF_W_addr e RF_Rp_addr.

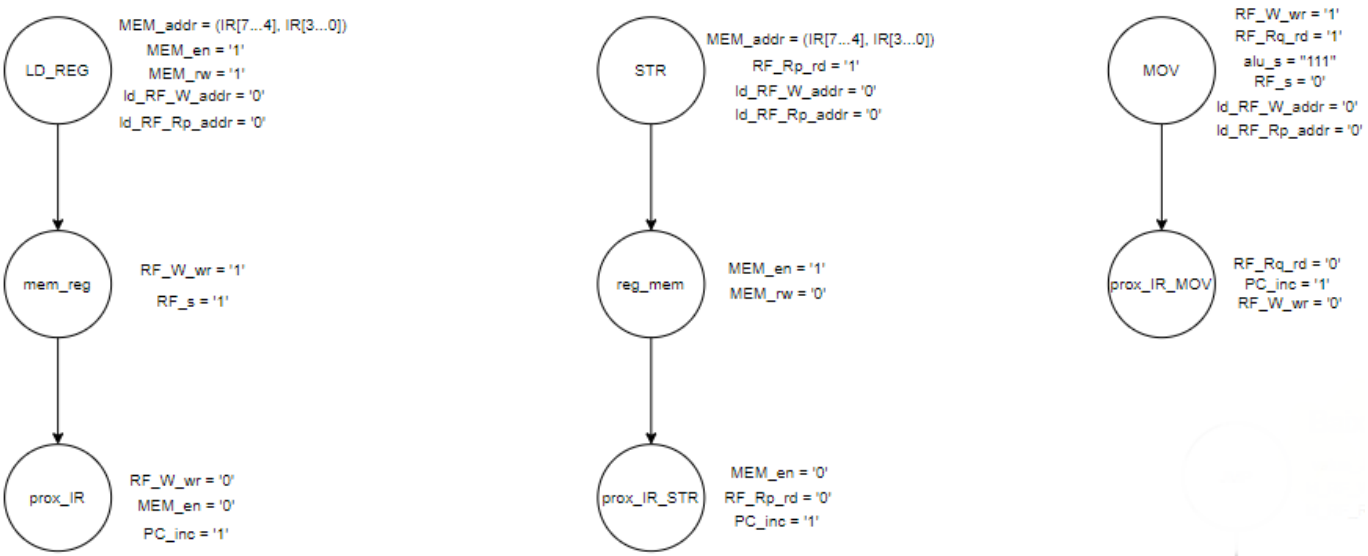
Na Figura 5, considere que ao final de cada pequena máquina de estados, selecionada pelo opcode da instrução (IR[15..12]), há o retorno para o estado **BUSCAR**.

No estado **LD_REG**, é permitido o acesso à memória de dados através do sinal MEM_en e a leitura da memória é habilitada através do sinal MEM_rw, ambos indo para nível lógico alto. Além disso, o sinal ld_RF_W_addr é forçado para o nível lógico baixo. Também é visível que o endereço da memória de dados é apresentado como sendo parte do barramento advindo do registrador de instrução, sendo a junção dos trechos B e C, IR[7..4] e IR[3..0] respectivamente. Em seguida, há o estado **mem_reg** que habilita o acesso e a escrita no banco de registradores pelo sinal (RF_W_wr = ’1’) e seleciona o mux (RF_s = ’1’) para que o valor da memória de dados seja direcionado ao W_data. Ao final desta instrução, o estado **prox_IR** desabilita o *enable* da memória e a escrita no banco de registradores, além de incrementar o contador de instrução por meio do sinal PC_inc.

No estado **STR**, é habilitada a leitura do registrador Rp banco de registradores (RF_Rp_rd = ’1’). Já o estado **reg_mem** habilita a escrita na memória por meio dos sinais MEM_en que vai para nível alto e MEM_rw que vai para nível baixo, liberando a escrita na memória no próximo pulso do relógio. Ao final, o estado **prox_IR_STR** desabilita a memória e a saída Rp do banco de registradores, além de incrementar o contador de instrução.

No estado **MOV**, é habilitada a leitura da saída do registrador Rq banco de registradores (RF_Rq_rd = ’1’), a seleção da operação da ALU é forçada para que a entrada Rq saia sem alteração de valor (alu_s = "111") e a seleção do mux de entrada do banco de registradores é alterada para receber o valor de saída da ALU (RF_s = ’0’), assim como é acionado o modo de escrita no bloco devido à mudança do sinal RF_W_wr indo para nível lógico alto. Ao final, o estado **prox_IR_MOV** desabilita a memória e a saída Rq do banco de registradores e o modo de escrita (RF_W_wr = ’0’), além de incrementar o contador de instrução.

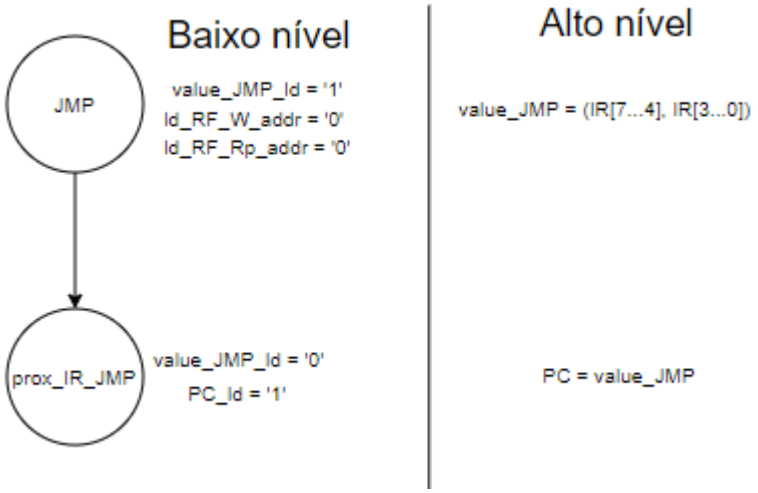
Figura 5: Estados LD_REG, STR, MOV



Fonte: os autores

Na Figura 6, pode-se ver que há a descrição de sinais de baixo nível e a ilustração da função do estado em alto nível. No estado **JMP**, é atribuído o nível lógico alto ao sinal value_JMP_ld e o contrário ocorre com os sinais ld_RF_W_addr e ld_Rp_W_addr. Na transição dos estados para o próximo estado, há inserção do valor da instrução IR[7..4] e IR[3..0] que compõem o endereço da próxima instrução no registrador value_JMP. No estado **prox_IR_JMP**, o sinail value_JMP_ld é colocado em nível lógico baixo e o sinal PC_ld é posto em nível lógico alto, forçando que, no próximo pulso do relógio, o valor no registrador value_JMP seja inserido no contador de instruções PC e não o incremento como é executado em outros estados.

Figura 6: Estado de salto



Fonte: os autores

Na Figura 7 (a) e (b), os estados apresentam comportamentos parecidos mas com condições de transição diferentes, visto que tratam de situações diferentes. Os estados **JZ**, **JNZ**, **JC** e **JNC** desativam os sinais ld_RF_W_addr e ld_Rp_W_addr responsáveis pelos endereços de entrada e saída do banco de registrador.

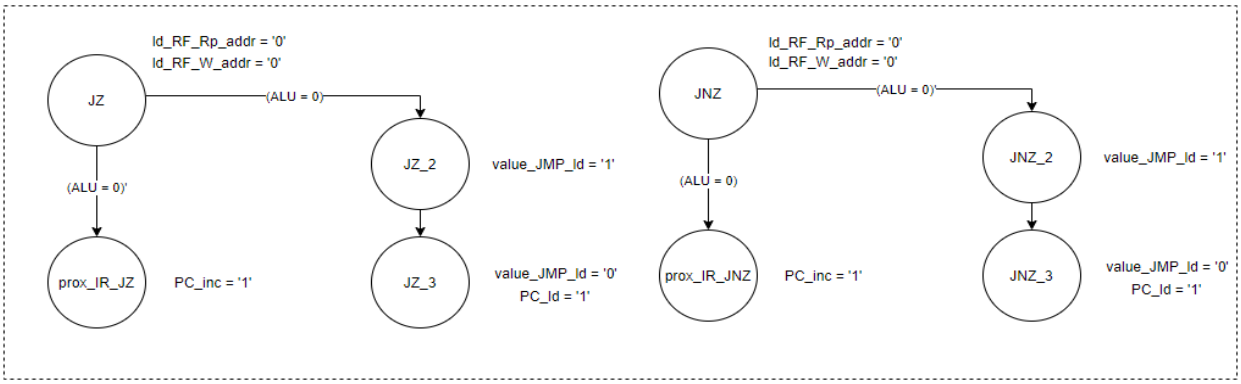
Na Figura 7 (a), é feita a comparação do valor da ALU. No estado **JZ** caso seja igual a zero, o sinal value_JMP_ld_ é ativado em **JZ_2**, carregando o valor da próxima instrução no registrador value_JMP contido nos IR[7..4] e IR[3..0]. Em seguida, após um ciclo de clock em **JZ_3**, o sinal PC_ld é alterado para nível lógico alto, possibilitando a aplicação do valor do registrador value_JMP no contador de instruções PC. Caso o valor da ALU não seja nulo, no estado **JZ**, a máquina vai para o estado **prox_IR_JZ** e então incrementa o contador PC.

No estado **JNZ**, acontece o contrário. Caso o valor da ALU seja nulo, o sinal value_JMP_ld_ é ativado em **JNZ_2**, carregando o valor da próxima instrução no registrador value_JMP. Logo após um ciclo de clock, em **JNZ_3**, o sinal PC_ld é alterado para nível lógico alto. Caso não seja nulo, a máquina vai para o estado **prox_IR_JNZ** e então incrementa o contador PC.

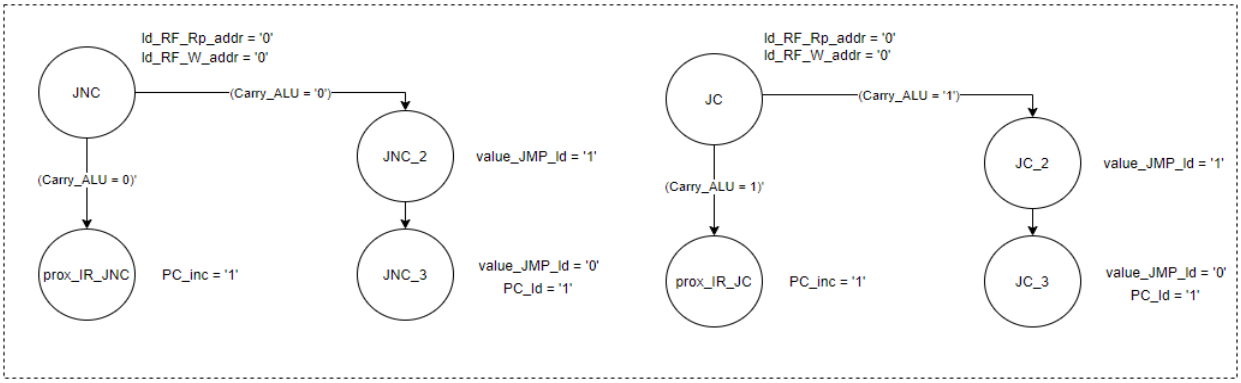
Na Figura 7 (b), ocorre o mesma coisa, porém, o sinal o qual será comparado com zero é o *carry* da ALU, caso haja *overflow*, o *carry* vai estar com nível lógico alto e então, do estado **JC**, a máquina irá para o estado **JC_2** onde é feito mesmo procedimento do estado **JNZ_2** e em seguida ao estado **JC_3** que procede da mesma forma que o estado **JNZ_3**. Em contrapartida, se não houver *overflow*, o estado depois do **JC** será **prox_IR_JC** onde o sinal PC_inc irá incrementar o contador PC.

No estado **JNC**, se o *carry* for nulo, não houver *overflow*, os próximos estados serão **JNC_2** e **JNC_3** que executam os mesmos processos dos estados **JC_2** e **JC_3**. Caso haja *overflow*, o próximo estado será **prox_IR_JNC** onde o sinal PC_inc irá incrementar o contador PC.

Figura 7: Estados de salto condicional



(a)

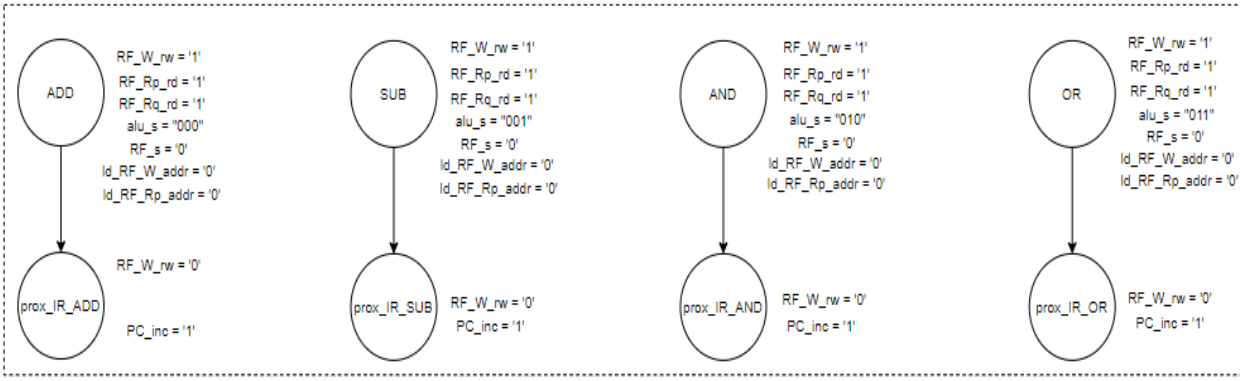


(b)

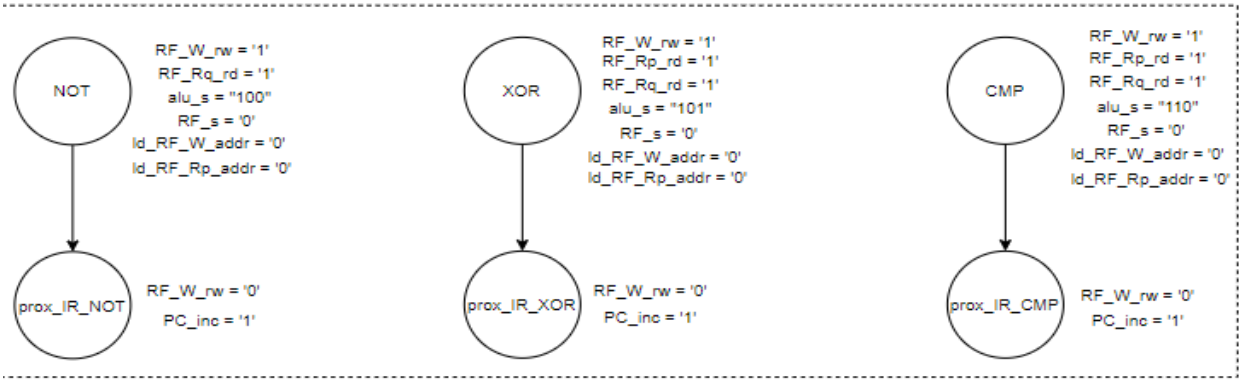
Fonte: os autores

Na Figura 8, mostra todas as operações que envolvem a ALU. Inicialmente é ativado o modo de escrita do banco de registradores, devido à necessidade de gravação do resultado da ALU no banco, e suas saidas também são acionadas. Entretanto, no estado **NOT** como é uma operação de apenas um dado foi necessário apenas ativar a saída Rp_data. Além disso, o seletor multiplexador MUX 2X1 (RF_s) de entrada é forçado para ’0’ com intuito de receber os dados da ALU no banco, o sinal de operação da ALU (alu_s) é alterado de acordo com a escolha da saída de sua saída, conforme a operação que está sendo executada, como alu_s = "001", por exemplo, atribuindo o resultado da adição à saída da ALU. Também são desativados os sinais ld_RF_W_addr e ld_Rp_W_addr. No último estado de cada máquina de instrução da ALU, é desativado o modo de escrita do banco de registradores e é executado o incremento do contador de instruções PC.

Figura 8: Operações na ALU



(a)



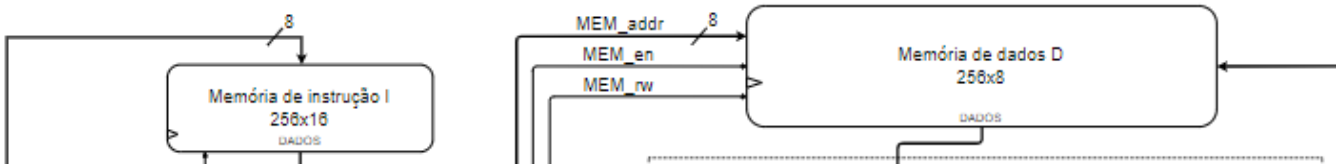
(b)

Fonte: os autores

Note que, na Figura 9, a memória de instrução é onde se armazena o programa desejado e foi determinado que seu tamanho é de 256 palavras contendo 16 bits cada, ou seja, 256x16. Para o projeto, foi escolhida uma memória ROM para atuar na memória de instruções.

Na memória de dados, são possíveis 256 endereços de dados contendo 8 bits cada. O tipo de memória escolhida foi a RAM devido a facilidade de implementação.

Figura 9: Memórias do projeto



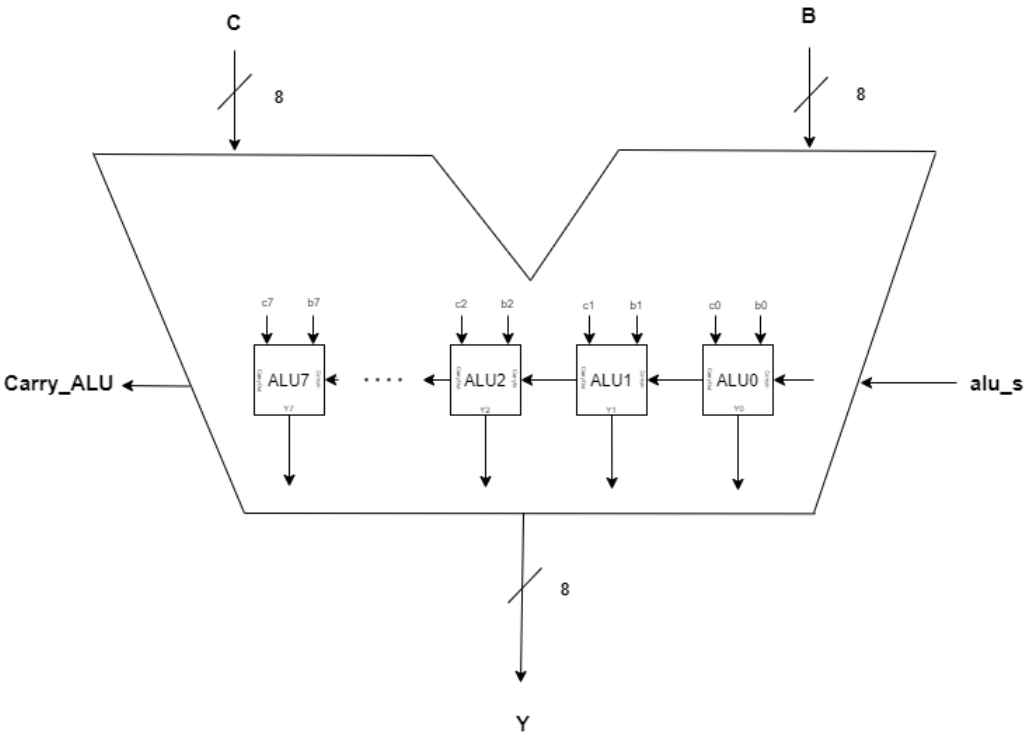
Fonte: os autores

2.2 Unidade Operacional

A ALU é um componente capaz de executar diversas operações aritméticas e lógicas com duas entradas de dados. Neste trabalho, foi construída uma ALU que realiza sete operações as quais são: ADD (SOMA), SUB (SUBTRAÇÃO), AND, OR, NOT, XOR e CMP (COMPARAÇÃO) de igualdade.

Na Figura 10 mostra o componente do tipo ALU em que há duas entradas de oito bits cada **B** e **C** que correspondem Rq_data e Rp_data, respectivamente. A entrada **alu_s** (3 bits) é responsável por fazer a seleção dos multiplexadores interno da ALU. A saída **Y** da ALU são 8 bits e **carry_ALU** que indica sobrecarga aritmética. Por fim, nesta mesma figura há oito blocos (ALU0, ALU1, ALU2, .., ALU7) chamados de extensores que computam bit a bit as entradas de dados.

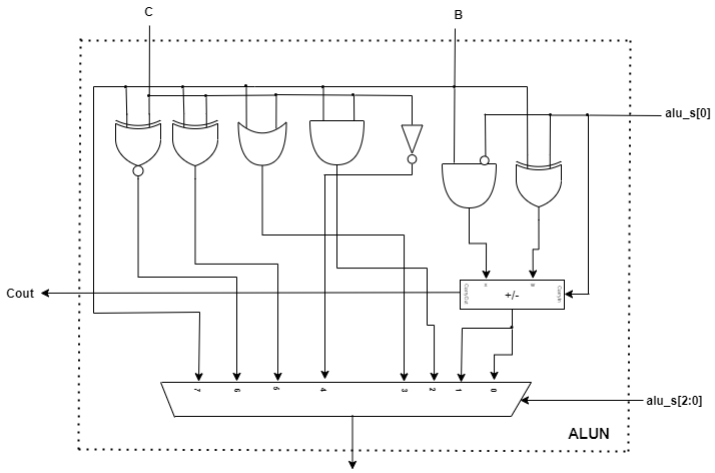
Figura 10: Design da Unidade Lógica e Aritmética



Fonte: os autores

Na Figura 11 mostra o bloco funcional chamado também de extensor. Nele, pode ser visto as sete operações solicitadas no trabalho. O comparador utilizado foi de 1 bit com porta xnor e pode ser consultado no material [1].Neste bloco, há um somador de 1 bit que pode ser modificado para um subtrator, uma vez que, implementado esse circuito alternativo com as portas AND e XOR na entrada do somador. O bit menos significativo alu_s[0] = 0 (soma) e 1 e alu_s[0] = 1 (subtrai). Na Tabela 2, mostra os bits de seleção em tabela-verdade bem como a respectiva função. Para melhor entendimento consultar o material [2].

Figura 11: Bloco Funcional da ALU



Fonte: os autores

Tabela 2: Lógica de Seleção das Operações

alu_s[2]	alu_s[1]	alu_s[0]	Função
0	0	0	ADD
0	0	1	SUB
0	1	0	AND
0	1	1	OR
1	0	0	NOT
1	0	1	XOR
1	1	0	CMP
1	1	1	MOV

Fonte: os autores

Há outros componentes que compõem a unidade operacional como o multiplexador de 2x1 que é responsável por carregar dados de 8 bits tanto da memória de dados quanto da ALU no banco de registradores, possui um bit de seleção chamado RF_s. Existe também, outro multiplexador 2x1 associado com um registrador chamado **RF_W_addr** que controla o endereço de escrita no banco.

A memória de dados que foi utilizado nesse trabalho possui um tamanho de 256 palavras de 8 bits cada e guarda todos os dados de entrada e saída que o processador pode para acessar.

3 Conclusão

O projeto do processador de 16 instruções seguiu todas orientações solicitadas no trabalho do Problema 02, bem como o desenvolvimento de cada instrução da Tabela 1. O livro Sistemas Digitais e Arquitetura e Organização de Computadores foram fundamentais para compreensão de todos os elementos de um processador de uso geral e auxiliou na criação de diversos componentes como a ALU. O estudo acerca dos processadores programáveis resultou na construção de um circuito integrado compacto e funcional, desta forma a implementação desse projeto terá um material que auxiliará a simulação corretamente de um processador de 16 instruções. Portanto, para trabalhos futuros que visem aumentar o número de instruções, pode-se realizar as modificações nesse projeto, visto que foi mostrado o RTL assim como a explicação de cada estado e componente.

4 Referência

[1] Liana Duenha. *Circuitos Combinacionais*. URL: http://www.facom.ufms.br/~lianaduenha/sites/default/files/part06c.pdf. (acessado: 04.01.2021).

[2] Mohsen Imani. *CSE140: Components and Design Techniques for Digital Systems*. URL: https://cseweb.ucsd.edu/classes/su17_2/cse140-a/lectures/CSE140_Imani_slide9.pdf. (acessado: 04.01.2021).

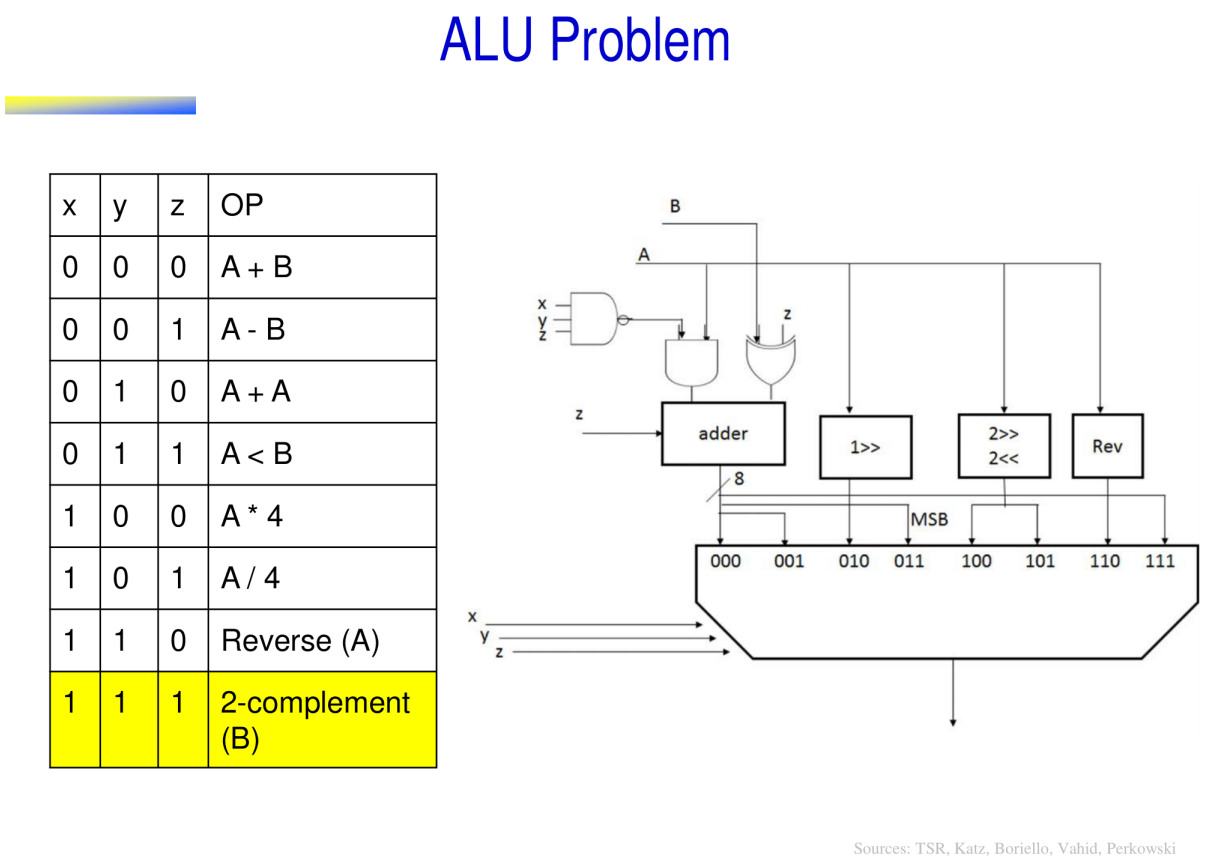
[3] William Stallings. *Arquitetura e Organização de Computadores 8a Edição*. 2010.

[4] Frank Vahid. *Sistemas Digitais*. Bookman Editora, 2009.

5 Apêndice

LINK CIRCUITO COMPLETO DO PROJETO NO: DRAW IO.
LINK RELATÓRIO OVERLEAF PARA EDIÇÃO DE IMPLEMENTAÇÃO.

Figura 12: Estrução de operação da ALU

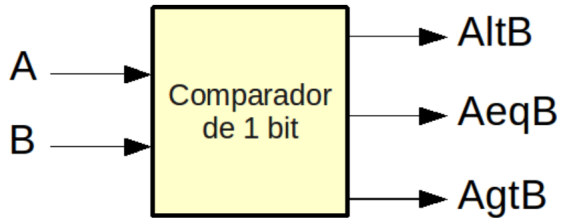


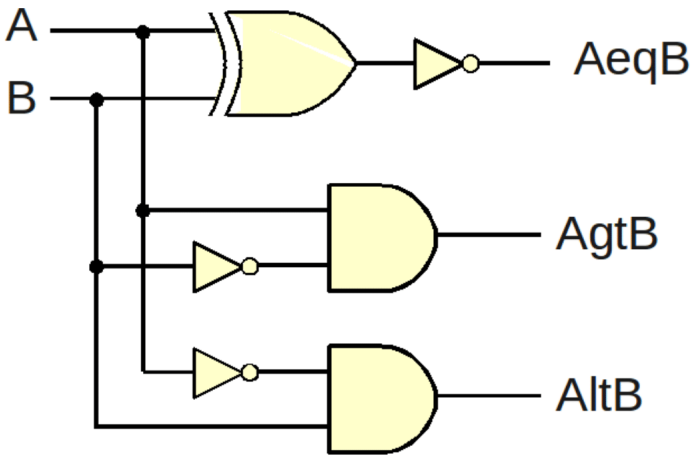
Fonte: Slides from Tajana Simunic Rosing

Figura 13: Comparador de 1 bit

Comparador de Magnitude de Dados de 1 bit

- $AeqB = \overline{A \oplus B}$
- $AgtB = A \bullet \overline{B}$
- $AltB = \overline{A} \bullet B$

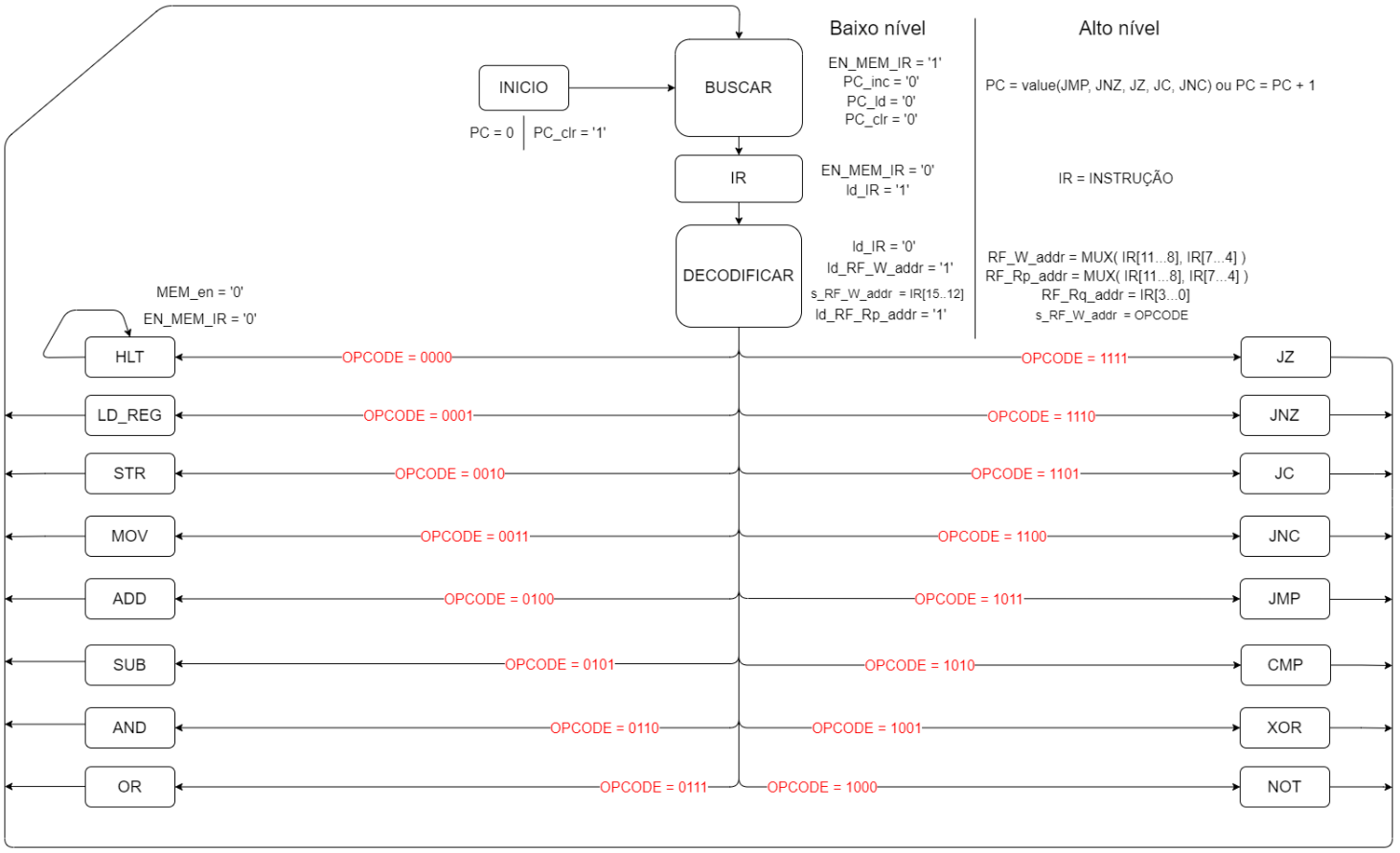




9

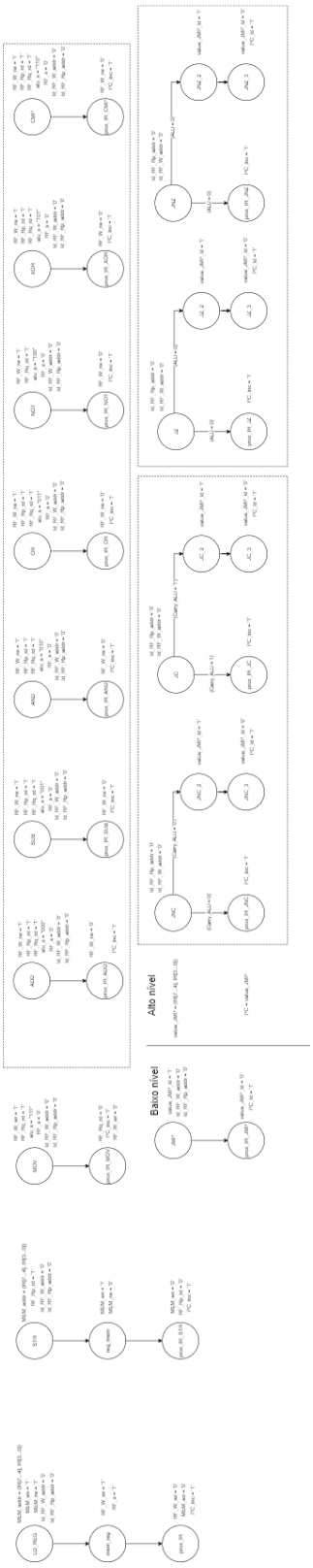
Fonte: Facom UFMS

Figura 14: MDE



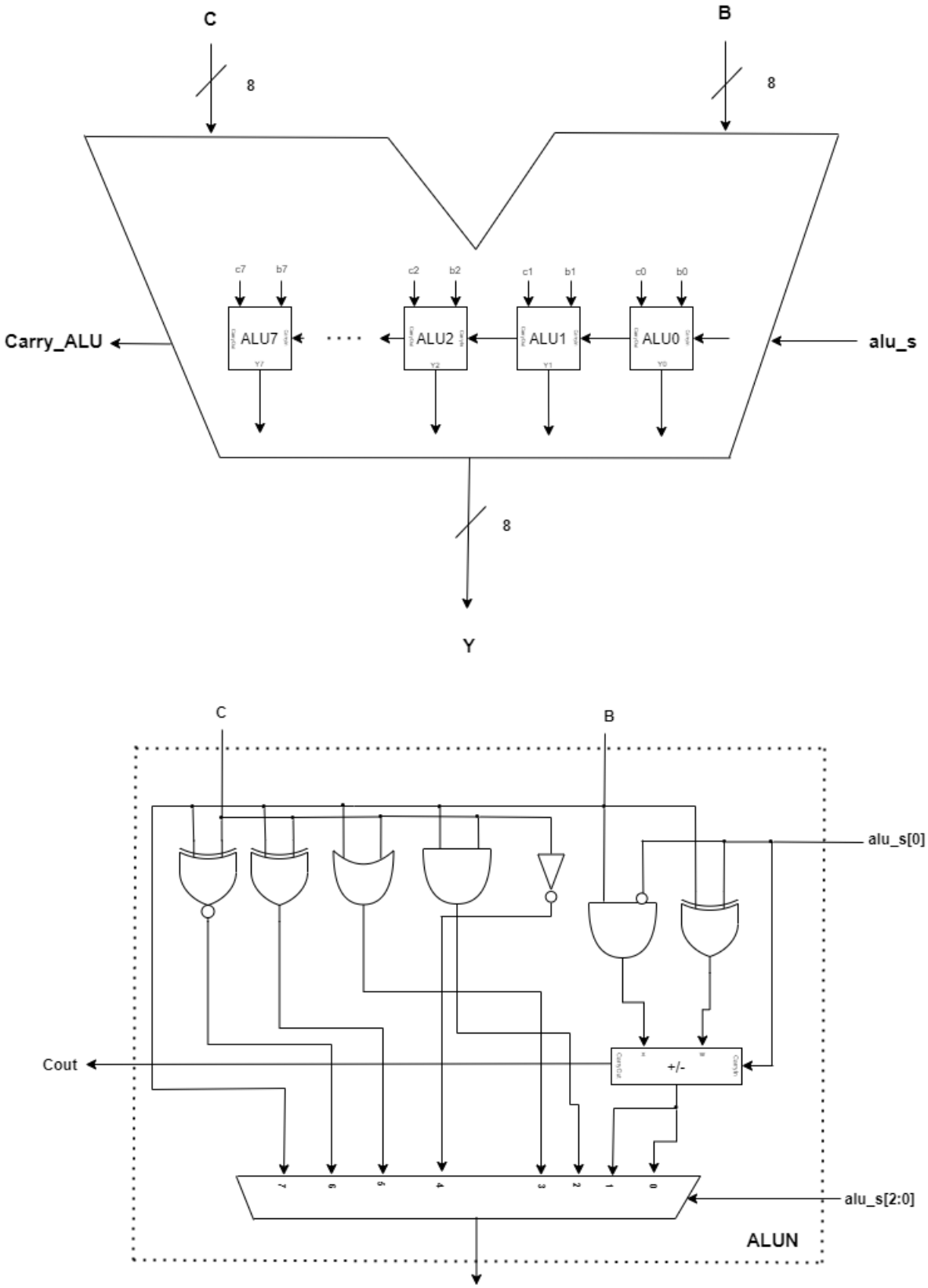
Fonte: os autores

Figura 15: MDE- CADA ESTADO



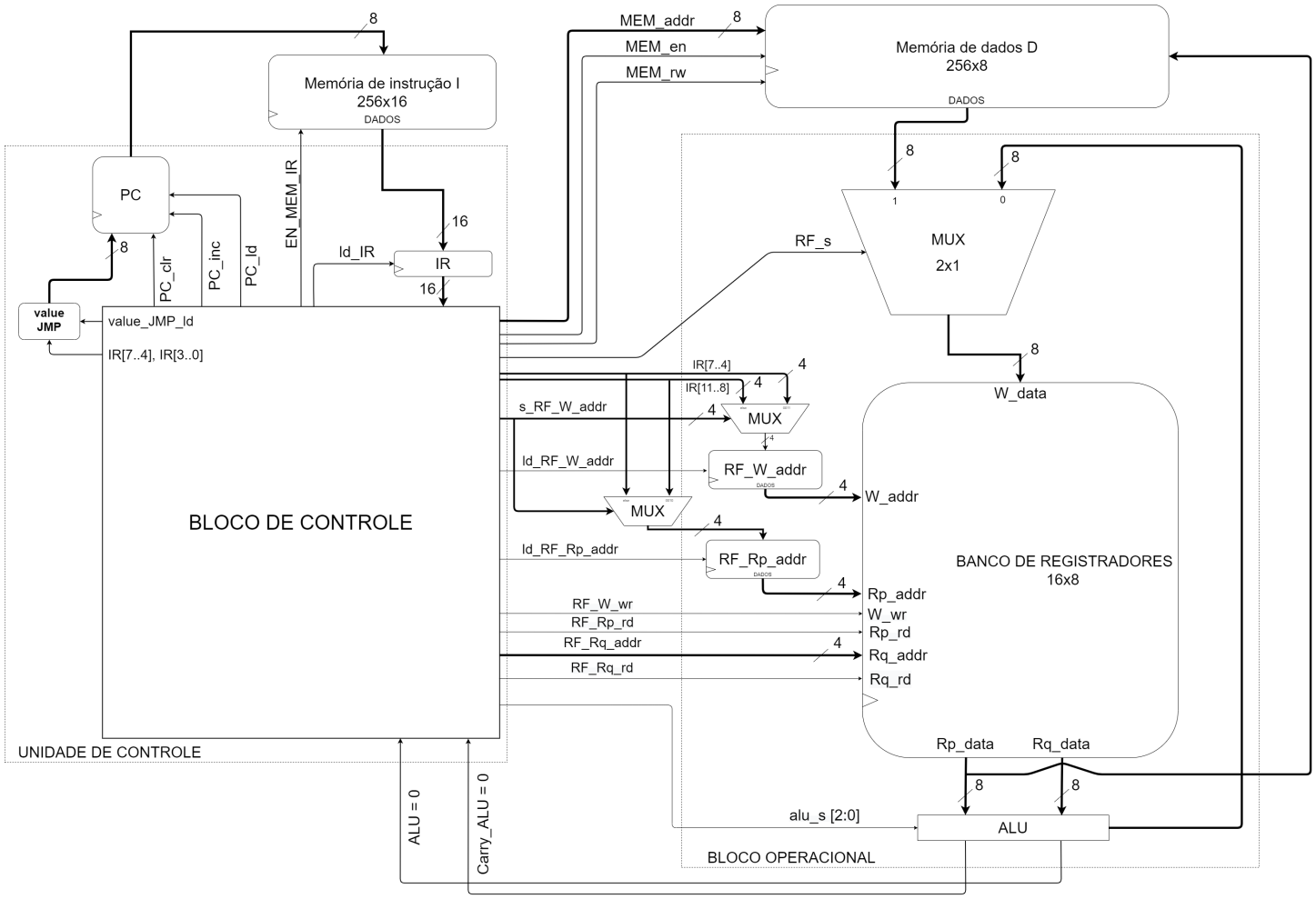
Fonte: os autores

Figura 16: ALU



Fonte: os autores

Figura 17: ALU



Fonte: os autores

6 ANEXO A - Relato semanal

Líder: Mateus de Assis Silva

6.1 A.1 Equipe

Função no grupo:	Nome completo do aluno
Redator:	Kaike Castro Carvalho
Debatedor:	Alisson Gabriel Lucas da Silva
Videomaker:	Willian Moura Gondim de Freitas

6.2 A.2 Defina o problema

O problema apresentado na terceira semana do curso de Sistemas Digitais se refere ao projeto de uma Unidade Central de Processamento (CPU) de uso geral de dezesseis (16) instruções. Trata-se de um circuito integrado capaz de executar autonomamente um conjunto de rotinas pré-definidas na memória de instrução. Tal capacidade deverá ser atingida através da manipulação de memórias, multiplexadores e uma Unidade Lógica e Aritmética (ULA) pelo Bloco Controlador.

6.3 A.3 Registro do brainstorming

Ao longo das reuniões muitas ideias foram propostas. Algumas delas se referiam a como gerenciar o grupo em si. Por exemplo, propôs realizar duas reuniões diárias (a fim de otimizar o tempo ao torná-las curtas e otimizar o trabalho ao torná-las frequentes), além de gravá-las. Além disso, concordou-se em separar as atividades de pesquisa entre os membros, de forma a otimizar o trabalho. No que diz respeito à elaboração do projeto, propôs-se uma proposta de MDE. Ela se apresentou funcional e o grupo decidiu prosseguir com a ideia. Ademais, aproveitou-se a estrutura básica do projeto RTL de uma CPU de uso geral encontrada nas orientações do segundo projeto e nela novos blocos operacionais foram adicionados.

6.4 A.4 Pontos-chave

As ideias propostas ao longo das reuniões levaram à definição de pontos chaves, os quais se encontram elencados abaixo:

- Funcionamento da ALU;

- Sincronicidade das memórias;
- Como a palavra-instrução (binária) é decodificada? É armazenada num registrador ou disponibilizada no barramento?
- Foi-se decidido que a Memória de Instrução deve ser uma ROM.
- A apresentação da MDE deverá ser "resumida". Os estados mais detalhados se encontram desenhados separadamente para simplificar a compreensão;
- A operação de HLT (halt - parada) não leva a estado algum;
- É permitida a passagem dos bits de B através da ULA, com o código 111;
- A operação de comparação é realizada apenas de igualdade bit a bit.

6.5 A.5 Questões de pesquisa

- Como uma ALU funciona?
- Como um processador de uso geral funciona?

6.6 A.6 Planejamento da pesquisa

Para o planejamento da pesquisa, na primeira reunião foi realizada uma discussão geral referente à compreensão individual de cada membro da equipe sobre o tópico em questão (CPU). Aqueles que compreendiam de forma superficial o conteúdo foram orientados a buscarem informações em referências bibliográficas, como os livros-texto das disciplinas de Sistemas Digitais e Arquitetura de Computadores. Àqueles que já tinham conhecimento da matéria analisada foi orientada a busca de conceitos mais específicos. Por exemplo, os alunos que já haviam cursado a disciplina de Arquitetura de Computadores avançaram na pesquisa de componentes operacionais (como a ULA) e os estados necessários para as operações da CPU.