Organização de Computadores

Arquitetura Cleópatra (Versão 3.0)

Profs. Ney Calazans e Fernando Moraes

Última alteração: 19/08/2004

Sumário:

1. ESPECIFICAÇÃO DA ARQUITETURA 2
   1.1 Conjunto de Registradores 2
   1.2 Conjunto de Instruções 3
      1.2.1 Observações 5
   1.3 Modos de Endereçamento (ME) 5
   1.4 Classes e Formatos de Instruções 6
      1.4.1 Formato das instruções sem operando explícito 6
      1.4.2 Formato das instruções com 1 operando explícito 6
      1.4.3 Observação e questão 7
   1.5 Linguagem de Montagem (Assembly Language) - Sintaxe 7
      1.5.1 Exemplo de programa e exercício proposto 9
2. ORGANIZAÇÃO PARA A ARQUITETURA 11
   2.1 A Interface de Entrada e Saída do Processador Cleópatra 11
   2.2 A Organização Interna do Processador Cleópatra Bloco de Dados e Bloco de Controle 13
3. BLOCO DE DADOS 16
   3.1 Estrutura Interna do Bloco de Dados 16
   3.2 A Implementação de Instruções 18
   3.3 Microssimulação do Bloco de Dados 18
      3.3.1 Programa exemplo 19
      3.3.2 Valores esperados no barramento de dados e nos sinais de controle por ciclo 19
      3.3.3 Uma linguagem de Micromontagem 19
   3.4 Resultados da Microssimulação 21
   3.5 Diagrama (Parcial) do Microcódigo executado pelo Bloco de Dados 22
4. BLOCO DE CONTROLE 24
   4.1 Introdução 24
   4.2 Interface de Entrada e Saída do Bloco de Controle 24
   4.3 Bloco de Controle - Implementação com Máquina de Estados 26
   4.4 Bloco de Controle - Implementação Microprogramada 28
      4.4.1 Estrutura interna da μROM 29
      4.4.2 Interface externa do seqüenciador 30
      4.4.3 Estrutura interna do seqüenciador 31

1. ESPECIFICAÇÃO DA ARQUITETURA

A arquitetura Cleópatra é um processador de 8 bits em todos os sentidos, tamanho dos registradores de trabalho, tamanho dos barramentos de dados e de endereços e tamanho de palavra da memória principal. Trata-se de uma arquitetura didática, que servirá para introduzir os conceitos gerais de organização de computadores, sem contudo se envolver com tópicos avançados presentes em praticamente qualquer processador atual, tais como “pipeline”, memórias “cache”, etc.

A maior limitação da arquitetura Cleópatra reside em seu barramento de endereços, que possui apenas 8 bits, o que determina um mapa de memória de apenas 256 posições disponíveis para armazenar programas e dados[1]. O motivo de impor tal limitação é tão-somente facilitar o processo didático, pois durante o aprendizado de organização de computadores não é estritamente necessário escrever programas de grande porte ou lidar com grandes volumes de dados. As características restantes da arquitetura podem ser ainda hoje encontradas em processadores comerciais, tais como microcontroladores para uso em eletrônica embarcada de 8/16 bits da Intel (e.g. o 80C51) e Motorola (e.g. o 68HC11), entre outros.

Em termos de estrutura, trata-se de uma arquitetura muito simples, similar às introduzidas pela primeira vez na década de 1970, baseadas em um registrador de trabalho único, o Acumulador, fonte e destino da maioria das operações realizadas sobre dados. Embora possua um número reduzido de instruções (apenas 14), não se deve confundir esta arquitetura com arquiteturas comumente denominadas RISC (do inglês “Reduced Instruction Set Computer” ou computador com conjunto de instruções reduzido), cuja caracterização será feita oportunamente. Justamente ao contrário, Cleópatra se adapta mais ao conceito oposto, de arquiteturas CISC (do inglês “Complex Instruction Set Computer” ou computador com conjunto de instruções complexo), embora seu conjunto de instruções seja tudo menos complexo[2].

A seguir, apresenta-se a especificação completa da arquitetura Cleópatra, listando seu conjunto de registradores e biestáveis qualificadores, na Seção 1.1, descrevendo o conjunto de instruções, na Seção 1.2, os modos de endereçamento disponíveis, na Seção 1.3 e as classes e formatos de instrução na Seção 1.4. Adicionalmente, introduz-se uma linguagem de montagem simples para facilitar a descrição textual de programas, bem como a representação de dados, na Seção 1.5.

1.1 Conjunto de Registradores

Nesta Seção, definem-se os registradores do processador Cleópatra que são de relevantes do ponto de vista **arquitetural**, ou seja, cuja existência necessita ser considerada em algum momento por programadores em linguagem de montagem. Outros registradores podem ser e serão necessários do ponto de vista **organizacional**, para implementar o processador, mas estes serão introduzidos mais tarde, quando for o momento oportuno, neste documento. Os registradores da arquitetura Cleópatra são divididos em três classes, mostradas a seguir.

Registrador Primário de Dados:

1 Registrador Acumulador — AC

Registradores de Controle:

1 Contador de Programa — PC

1 Registrador de Subrotina — RS

[1] Esta limitação pode ser superada através de software e hardware adequados, externos ao processador, mas a discussão deste assunto foge ao escopo do presente documento.

[2] Vê-se assim como as nomenclaturas RISC e CISC são em si inadequadas para descrever plenamente os conceitos a elas subjacentes. Mesmo assim, continua-se a usá-las por motivos históricos.

Qualificadores:

4 Biestáveis de Estado -	*Negativo*	N
	Zero	Z
	Vai-um	C
	Transbordo	V

1.2 Conjunto de Instruções

Instruções de uma arquitetura são ordens para o hardware realizar alguma tarefa. Cada instrução consiste de um conjunto de ações executadas pelo processador em um certo período de tempo. Estas ações podem incidir sobre os registradores de dados e/ou de controle, sobre os qualificadores, sobre a memória principal ou sobre o sistema de controle do processador como um todo. Nas arquiteturas von Neumann mais tradicionais, cada instrução é executada individualmente pelo processador, sem que nenhuma outra instrução possa ser executada ao mesmo tempo que esta.

Na Tabela 1.1 constam todas as instruções que a arquitetura Cleópatra é capaz de executar. Qualquer programa executado por esta arquitetura consiste de seqüências de instruções retiradas desta Tabela. Cada instrução é especificada por três informações:

- um **mnemônico** ou seja, um texto simbólico que lembra a função desempenhada pela instrução, e que pode ser usado para descrever programas que podem ser executados na arquitetura. Deve-se notar na tabela que algumas instruções possuem um operando e outras não. Um **operando** é um dado a ser usado na execução, e que expressa, sob alguma codificação informações como números naturais ou inteiros, caracteres, vetores de bits, endereços de posições de memória, etc.;

- uma **codificação de instrução**, que consiste em um ou em um conjunto de códigos binários que distinguem a instrução de todas as demais da arquitetura. Por exemplo, a instrução NOT é designada por dois códigos possíveis, 0000 ou 0001, pois seu código é 000x. Por outro lado, a instrução AND possui o código 0111. Obviamente, códigos de instruções distintas não podem ser iguais, nem sequer ter uma intersecção não nula, no caso da especificação desta possuir inespecificações (em inglês “don´t cares”), representados pelo caracter “**-**”;

- uma **descrição**, que estabelece a semântica associada à instrução.

Uma observação importante na descrição das instruções diz respeito ao seqüenciamento de execução destas. No paradigma de computador de programa armazenado, também conhecido como modelo de von Neumann, se as instruções não determinarem nada em contrário, a execução de instruções é seqüencial, no ordem em que estas aparecem no programa. Existe sempre uma classe de instruções em qualquer linguagem e/ou arquitetura cuja operação consiste justamente em controlar o fluxo de execução das instruções restantes do programa. São as chamadas **instruções de controle de fluxo de execução**, podendo constituir-se de saltos incondicionais e condicionais, repetição de trechos (laços, ou em inglês “loops”), etc. Na Cleópatra, todas as instruções cujo mnemônico inicia com a letra “J” (do inglês “jump”, saltar), acrescidas das instruções RTS e HLT pertencem à classe de instruções de controle de fluxo de execução. Note-se que as instruções de salto condicional (JN, JZ, JC e JV) possuem um funcionamento que varia de acordo com o valor armazenado em algum biestável de estado no momento da execução do salto. O salto pode ser executado ou não. Deve-se perceber que, como o registrador PC aponta sempre para a próxima instrução a ser executada, um salto é implementado pela escrita de um novo valor no PC (mudando qual a próxima instrução a ser executada). No caso de salto não executado por uma instrução de salto condicional, existe uma menção ditando que o PC é incrementado de 1. Ora, embora não

explicitado para as instruções restantes da Cleópatra, deve-se ter em mente que esta ação acontece para todas as instruções (inclusive as de controle de fluxo de execução como JMP, RTS e HLT), visando implementar o modelo de execução seqüencial assumido por omissão no modelo de programação empregado em todos os processadores existentes. Em resumo, o incremento do registrador PC é um **efeito secundário** da execução de qualquer instrução. O fato de este efeito estar explicitado para instruções de salto condicional serve apenas para reforçar as duas possibilidades de execução possíveis para estas instruções.

Tabela 1.1 – Instruções da arquitetura Cleópatra.

Descrição da Instrução	Mnemônico	Código	Descrição
Inversão do acumulador	NOT	000-	Complementa (ou seja, inverte) todos os bits de AC.
Escrita na memória	STA **operando**	001-	Armazena dado em AC na posição de memória especificada por **operando**.
Leitura da memória para o acumulador	LDA **operando**	0100	Carrega AC com conteúdos de memória da posição especificada por **operando**.
Adição de valor a valor do acumulador	ADD **operando**	0101	Adiciona dado em AC a conteúdo da memória dada por **operando**, colocando o resultado em AC.
Ou de valor com valor do acumulador	OR **operando**	0110	Faz OU bit a bit de AC com conteúdo da memória dada por **operando**, colocando o resultado em AC.
E de valor com valor do acumulador	AND **operando**	0111	Faz E bit a bit de AC com conteúdo da memória dada por **operando**, colocando o resultado em AC.
Salto incondicional	JMP **operando**	1000	PC recebe valor especificado por **operando**.
Salto condicionado ao valor do qualificador de vai-um	JC **operando**	1001	Se C=1, PC recebe valor dado por **operando**. Senão, PC← PC+2.
Salto condicionado ao valor do qualificador de transbordo aritmético	JV **operando**	1110	Se V=1, PC recebe valor dado por **operando**. Senão, PC← PC+2.
Salto condicionado ao valor do qualificador de sinal	JN **operando**	1010	Se N=1, PC recebe valor dado por **operando**. Senão, PC← PC+2.
Salto condicionado ao valor do qualificador zero	JZ **operando**	1011	Se Z=1, PC recebe valor dado por **operando**. Senão, PC← PC+2.
Salto incondicional para subrotina	JSR **operando**	1100	RS recebe conteúdo de PC, enquanto que PC recebe valor especificado por **operando**.
Retorno de subrotina	RTS	1101	PC recebe conteúdos de RS.
Parada do Processador	HLT	1111	Suspende a realização dos ciclos de busca, decodificação e execução.

Todas as instruções da arquitetura Cleópatra possuem pelo menos dois operandos, exceto a instrução HLT que não possui nenhum. Contudo, visando simplificar a execução de instruções (e portanto a *implementação em hardware da arquitetura*, ou seja, a *organização do processador*), nem todos os operandos de cada instrução são explicitamente selecionáveis pelo programador em linguagem de montagem (ou mesmo pelo programador em linguagem objeto). A maioria dos operandos das instruções é especificada de forma implícita. Por exemplo, uma instrução ADD possui três operandos, os dois operandos fonte e o operando destino, mas apenas um dos operandos

fonte é *selecionável* pelo programador, os outros dois sendo determinados pela arquitetura como sendo o registrador AC, de forma implícita.

1.2.1 Observações

1. Na codificação de instruções, uma posição marcada com "–" indica que o valor do bit em questão é irrelevante, podendo ser 0 ou 1;
2. **operando** - operando que depende do modo de endereçamento para ter seu valor e semântica determinados;
3. Instruções lógicas (NOT, AND e OR) e LDA - afetam os códigos de condição N e Z;
4. ADD - Afeta N, Z, C e V;
5. Instruções restantes (STA, JMP, JN, JZ, JC, JV, JSR, RTS HLT) - não alteram códigos de condição;
6. Instruções sem operando (NOT, RTS e HLT) - estas instruções operam sobre dados implícitos ou não têm operandos explícitos;
7. Instruções de desvio (JMP, JN, JZ, JC, JV, JSR) e STA - nestas, os modos de endereçamento direto e imediato são idênticos (possuem o mesmo efeito). Para uma definição do que é modo de endereçamento, ver Seção seguinte.

1.3 Modos de Endereçamento (ME)

Toda instrução que trabalha sobre alguma informação que varia ao longo do tempo de execução ou entre execuções de um mesmo programa requer a especificação destas informações sob a forma de operandos. A interpretação dos operandos depende da maneira como o hardware foi concebido. Por exemplo, uma dada instrução que soma dois números pode pressupor que os números estão especificados dentro da própria instrução. Em português, poder-se-ia ter uma instrução *"Some 5+5 e coloque o resultado no registrador Acumulador"*. Uma maneira mais flexível de realizar a mesma instrução seria algo como dizer *"Some X+Y e coloque o resultado no registrador Acumulador"*, supondo que X e Y identificam valores variáveis que podem mudar a cad execução distinta da dita instrução. Num computador, isto pode ser representado como uma instrução que soma os conteúdos de duas posições de memória e coloca o resultado no Acumulador. Ambas instruções realizam o mesmo processamento se X e Y corresponderem a posições de memória contendo 5, mas a segunda forma é certamente mais flexível do ponto de vista de programação. Às diferentes maneiras de se recuperar operandos para uma dada instrução dá-se o nome de **modos de endereçamento**. Note-se que, de um ponto de vista estrito, modo de endereçamento não é uma característica de uma instrução, mas de cada operando de uma instrução. Na Tabela 1.2 aparecem os modos de endereçamento disponíveis na arquitetura Cleópatra.

Tabela 1.2 – Modos de endereçamento da arquitetura Cleópatra.

Denominação	Codificação	Descrição
Imediato	00	Operando é o próprio dado de 8 bits armazenado segunda palavra de memória da instrução.
Direto	01	Operando é endereço de memória de 8 bits onde se encontra o dado.
Indireto	10	Operando é endereço de memória de 8 bits onde se encontra o endereço do dado.
Relativo	11	Operando é deslocamento de 8 bits em complemento de 2, a ser somado ao valor atual do PC, gerando assim o endereço do dado.

Os modos de endereçamento são descritos por um nome, um código binário e uma descrição da semântica de cada modo disponível. Não está mostrado, mas os operandos podem ainda ser especificados através do modo de endereçamento **implícito**, que consiste em embutir no próprio código da instrução a identificação do operando, conforme discutido na Seção anterior.

1.4 Classes e Formatos de Instruções

Na arquitetura Cleópatra, divide-se as 14 instruções, quanto ao funcionamento, em 3 classes:

- Instruções de Movimento de Dados – É a classe mais repetidamente executada na prática, sobretudo em arquiteturas com poucos registradores, e inclui as instruções LDA e STA;
- Instruções Lógicas ou Aritméticas – É a única classe cuja execução pode produzir transformações de dados, sendo portanto fundamental em qualquer arquitetura. Na Cleópatra, a classe inclui as instruções lógicas (NOT, AND e OR) e a instrução aritmética ADD;
- Instruções de Controle do Fluxo de Execução – É a classe que permite comandar a ordem de execução das instruções restantes, sendo assim útil para acrescentar flexibilidade e tornar programas mais compactos. Inclui o desvio incondicional (JMP) os desvios condicionais (JN, JZ, JC e JV), as instruções de desvio e retorno de subrotina (JSR e RTS) e a instrução HLT.

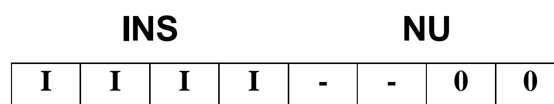
A arquitetura Cleópatra foi projetada com base no conceito de que todas as operações de manipulação de dados usam de alguma forma o Acumulador AC. Esta simplificação arquitetural permite que existam apenas dois formatos de instrução:

- Instruções sem operando explícito;
- Instruções com 1 operando explícito.

Instruções sem operando explícito ocupam apenas 1 byte (= uma palavra), enquanto que instruções com 1 operando explícito ocupam dois bytes (= duas palavras) da memória de programa. Abaixo, descreve-se cada um dos formatos separadamente.

1.4.1 Formato das instruções sem operando explícito

Neste formato, os quatro primeiros bits contêm o código instrução ou operação (em inglês, "operation code", ou OPCODE), INS. Os dois bits seguintes podem assumir qualquer valor, e não são usados, indicado por NU. Estes bits são usados no outro formato descrito abaixo. Os dois últimos bits da primeira palavra de uma instrução são sempre 0, e são igualmente ignorados. Note-se que este formato serve para especificar apenas três das instruções da arquitetura Cleópatra (NOT, RTS e HLT). Graficamente, o formato é o seguinte:



1.4.2 Formato das instruções com 1 operando explícito

Neste formato, a primeira palavra é similar ao formato anterior, onde os quatro primeiros bits contêm o código da operação (operation code, ou OPCODE), INS, e os dois últimos são sempre 0. Contudo, os bits seguintes ao campo INS especificam o Modo de Endereçamento (em

inglês “addressing mode”) ME, a ser usado, de acordo com a codificação fornecida na Tabela 1.2. Como já foi mencionado antes, o modo de endereçamento indica a forma de recuperar o dado. Na Cleópatra, o dado é explicitado pelo programador em linguagem de montagem na segunda palavra de instrução, designada **operando**. Graficamente, o formato de instruções com 1 operando explícito é o seguinte:

I	I	I	I	A	A	0	0
Operando							

1.4.3 Observação e questão

O número real de instruções distintas executáveis por uma arquitetura é uma função não apenas do número de códigos de operação, mas igualmente dos modos de endereçamento. Idealmente, o número de instruções distintas seria o produto do número de códigos de operação distintos (na Cleópatra, 14) pelo número de modos de endereçamento distintos (na Cleópatra, 4). Como algumas instruções não usam todos os modos de endereçamento, este valor ideal (na Cleópatra 14X4=56 instruções) quase nunca é alcançado.

1. Exercício: Calcule o número efetivo de combinações distintas de OPCODE e ME que correspondem à execução de ações distintas pela arquitetura Cleópatra. Use as Tabelas 1.1 e 1.2, e as observações da Seção 1.2 e as informações da presente Seção. (A resposta correta é 40. Justifique este valor).

1.5 Linguagem de Montagem (Assembly Language) - Sintaxe

Programar um processador consiste em utilizar o conjunto de instruções por este disponibilizado e especificar uma ordem de aplicação de um certo número de cada uma destas instruções com o objetivo de processar informações. Um programa é sempre descrito através de uma linguagem formal. Para especificar um programa que vá ser executado em uma determinada arquitetura existem dois métodos básicos:

- Especificar os códigos binários de instruções, modos de endereçamento e operandos para cada instrução a executar, e em seguida juntar cada um dos vetores binários assim obtidos em um vetor binário único que pode ser armazenado em memória e fornecido ao processador como um programa a executar. Isto se chama de **programação em linguagem de máquina**, e foi a primeira forma surgida de programação de computadores eletrônicos, sendo tediosa e muito propensa a erros. O programa gerado por este processo chama-se **código objeto** ou **código executável**;

- Especificar instruções usando mnemônicos, símbolos tipográficos especiais para designar os modos de endereçamento e formas simbólicas de representar operandos, tais como números em diversas bases, caracteres ou vetores de caracteres (“strings”), e em seguida juntar cada instrução assim especificada em um texto com uma dada sintaxe que traduz simbolicamente a semântica de execução de instruções do processador em questão. Isto se chama de **programação em linguagem de montagem**, e foi a primeira forma obtida de superar os maiores problemas da programação em linguagem de máquina. O programa em linguagem de montagem gerado pelo processo aqui descrito é também denominado **código fonte**. Obviamente, programas em linguagem de montagem não são executáveis pelo processador. É necessário antes traduzir o texto gerado para a linguagem de máquina descrita na especificação da arquitetura do processador. O trabalho de tradução sendo sistemático, tedioso e propenso a erros, é bem realizado por

programas de computador, os denominados **montadores** (em inglês **"assemblers"**), cujo nome deriva do fato destes "montarem" o código objeto a partir do código fonte;

Um aspecto importante é a relação entre linguagens de alto nível tais como C, Pascal, Basic e as linguagens de máquina e de montagem. Primeiro, as linguagens de alto nível são altamente independentes da arquitetura do processador, o que não ocorre com as linguagens de montagem e de máquina. Segundo, a tradução de programas em linguagem de alto nível para código objeto é muito complexa (muito mais que a tradução de código fonte em linguagem de montagem para código objeto), fazendo com que os programas tradutores gerem código objeto que nem sempre é o mais eficiente. Os tradutores de linguagem de alto nível se denominam genericamente de **compiladores**, e normalmente realizam o processo de tradução em vários passos, entre os quais quase sempre está a geração de um programa em linguagem de montagem, seguido da montagem deste código fonte. Ou seja, um montador funciona como um dos passos realizados por um compilador.

Nesta Seção, introduz-se uma proposta de linguagem de montagem para representar programas e dados para a arquitetura Cleópatra. A linguagem será definida usando um formato similar ao formalismo BNF (do inglês "Backus-Naur Form"), de uso corrente na especificação de linguagens. Neste formalismo, textos entre < e > representam estruturas da linguagem definidas por alguma regra. Texto fora destes delimitadores corresponde a conjuntos de caracteres ASCII que devem ser digitados exatamente como mostrado. As regras para cada estrutura da linguagem serão apresentadas informalmente, em português, como segue:

- Um programa nesta linguagem será uma seqüência de linhas, cada uma destas podendo conter um ou mias instâncias dos seguintes elementos léxicos:
    - Uma instrução da arquitetura;
    - Uma especificação de operando (se aplicável);
    - Uma especificação de rótulo, usando como referência simbólica no programa a posições da memória de programa ou da memória de dados;
    - Um comentário;
    - Caracteres TAB e espaços.
- Uma linha contendo uma instrução é formada por um rótulo opcional, seguido por um mnemônico obrigatório, seguido ou não de um operando (a existência deste depende do mnemônico usado), seguido de um comentário opcional;
- Os códigos de operação (do inglês, "operation code" ou "opcode") são palavras reservadas, correspondendo aos 14 mnemônicos mencionados acima, no item Conjunto de Instruções.
- Rótulos podem denotar posições da memória de programa, quando precederem algum dos 14 mnemônicos, ou da memória de dados, quando precederem a diretiva "DB"; a diretiva DB (do inglês "Define Byte") funciona de forma similar a declarações em linguagens de alto nível, tendo como efeito a reserva de uma posição de memória para conter a informação no campo de operando da diretiva e designada no programa pelo nome que corresponde ao rótulo que precede a diretiva, se este existir; cuidado, diretivas <u>não são</u> instruções da arquitetura, mas ordens e informações de controle a serem usadas pelo programa montador na geração do código objeto a partir do código fonte;
- Rótulos são especificados por qualquer texto que inicia com letra, e que é diferente de um mnemônico, e sucedido pelo caracter especial ":";
- Comentários são iniciados por ";". O restante da linha após este caracter é ignorado;

- O formato para os operandos é o seguinte:

  #<rótulo> ou #<número>

  < rótulo > ou <número>

  < rótulo >,I ou < número>,I

  < rótulo >,R ou < número>,R

- Os símbolos "#", "I" e "R" indicam o modo de endereçamento a usar na interpretação do operando, correspondendo a Imediato, Indireto e Relativo, respectivamente. O modo de endereçamento deve ser explicitamente fornecido, exceto no caso de endereçamento direto que é o modo usado por omissão. <rótulo> representa qualquer rótulo como especificado acima, sem o caracter delimitador ":". <número> representa um número em um de três formatos, hexadecimal, binário ou decimal, cada um respeitando respectivamente as seguintes notações: <*numhexa*>H / <*numbin*>B / <*numdec*>

- <*numhexa*> é uma seqüência de dois três caracteres do conjunto {0,1,2,3,4,5,6,7,8,9, 0,A,B,C,D,E,F}, representando um número hexadecimal representável em 8 bits. Caso o primeiro caracter seja uma letra, deve-se fazer preceder o número hexadecimal de um 0. Neste caso, o número de caracteres pode chegar a 3 (o objetivo é evitar a confusão entre rótulos e números pelo programa montador);

- <*numbin*> é um conjunto de até oito elementos do conjunto {0,1}, ou seja, um número binário;

- <*numdec*> é ou um conjunto de até três de caracteres do conjunto {0,1,2,3,4,5,6,7,8,9,0}, representando um número decimal entre 0 e 255, ou um conjunto de até três caracteres deste mesmo conjunto precedido do caracter "-", representando um número decimal negativo entre -1 e -128. Se houver omissão de um identificador de formato de representação, assume-se que o número é representado em decimal;

- <*rótulo*> corresponde sempre a um endereço de memória, seja da área de programa, seja da área de dados. A interpretação semântica de porções do texto em linguagem de montagem contendo um rótulo precedido do caracter # depende da instrução sendo executada. Do ponto de vista geral, preceder um rótulo pelo caracter # indica que o operando da instrução é o próprio valor associado ao rótulo, e não o valor armazenado na posição de memória cujo endereço é o valor do rótulo.

- Quando se programa em linguagem de montagem, da mesma forma que em programação em linguagens de alto nível, é sempre necessário especificar não apenas as instruções a executar, mas também os dados sobre os quais se trabalhará. Para especificar os dados e separar dados de programas, é necessário usar um conjunto de *diretivas* de montagem, onde diretivas podem ser definidas como instruções para o processo de montagem, e não para o processo de execução de programas, como é o caso das *instruções* do processador. Usaremos apenas 6 diretivas no que segue, DB para reservar posições de memória e inicializá-las com valores (DB é análoga à forma mais simples de declaração de variáveis em linguagens de alto nível), ORG para especificar o endereço de memória a partir do qual prosseguirá a montagem das linhas seguintes do código fonte em linguagem de montagem (útil para localizar dados e instruções em regiões específicas de memória), e os pares .CODE e .ENDCODE e .DATA e .ENDDATA, para especificar início e fim de segmentos de programas e dados, respectivamente.

1.5.1 Exemplo de programa e exercício proposto

Apresenta-se agora um exemplo de programa e sua interpretação. No programa, os endereços da memória de programa e de dados e o código objeto correspondente a cada linha foram colocados

como comentário.

1. Exemplo: O programa abaixo executa um algoritmo irrelevante, apenas serve para ilustrar o formato da linguagem de montagem, a tradução de linguagem de montagem para linguagem de máquina (em hexadecimal) e o funcionamento dos modos de endereçamento. Note que o número das linhas não faz parte do programa, está indicado apenas para fins didáticos.

Explanação: O programa inicia carregando o registrador acumulador AC com o conteúdo da posição 90 de memória na linha 1 (correspondendo ao rótulo END1, inicializado com 30 no programa pela diretiva DB na linha 11); a este dado, a segunda instrução soma o conteúdo da posição de memória 93 (correspondendo ao conteúdo da posição de memória associada ao rótulo END4, acessada indiretamente a partir do rótulo END2), colocando o resultado da soma de volta no acumulador, obtendo o dado via endereçamento indireto; a terceira instrução realiza o armazenamento do conteúdo de AC na posição de memória END3 (endereço 92, inicialmente contendo 00). Após, o mesmo dado é transformado pelo zeramento de seus quatro bits mais significativos, pela instrução da linha 6, e o resultado volta à AC. Se o resultado desta operação for 0, o programa pára, senão AC tem todos os seus bits invertidos e só aí o programa pára. Identifique se com os dados fornecidos abaixo a linha 8 é executada ou não.

Memória de Programa

```
   RótuloMnem  Operando        End   Opcode   Oper
1  .CODE                   ; Designa o início de um trecho da memória de programa
2         ORG  #00H        ; Indica que a montagem deve passar ao endereço 00H
3  INIT:  LDA  END1        ;    00    44       90
4         ADD  END2,I      ;    02    58       91
5         STA  END3        ;    04    24       92
6         AND  #0FH        ;    06    70       0F
7         JZ   FIM,R       ;    08    BC       01
8         NOT              ;    0A    00
9  FIM:   HLT              ;    0B    F0
10 .ENDCODE                ; Designa o fim de um trecho da memória de programa
```

Memória de Dados

```
   RótuloMnem  Operando        End   Dado
11 .DATA                   ; Designa o início de um trecho da memória de programa
12        ORG  #90H        ; Indica que a montagem deve passar ao endereço 90H
13 END1:DB     #30H        ;    90    30
14 END2:DB     #END4       ;    91    93
15 END3:DB     #00H        ;    92    00
16 END4:DB     #5BH        ;    93    5B
17 .ENDDATA                ; Designa o fim de um trecho da memória de dados
```

2. Exercício: Faça um programa para somar dois vetores de n posições, onde n está armazenado numa posição de memória ao iniciar o programa.

3. Exercício: Na arquitetura Cleópatra existem algumas instruções muito comuns em processadores comerciais, tais como a subtração de dois números inteiros e a operação OU-Exclusivo de dois vetores de bits. Implemente estas operações como subrotinas em linguagem de montagem Cleópatra supondo que as rotinas trabalham sobre dois operandos de 8 bits, um contido no registrador AC e outro numa posição pré-fixada de memória, retornando ambas o resultado no registrador AC.

2. ORGANIZAÇÃO PARA A ARQUITETURA

Existem inúmeras maneiras de se implementar a especificação introduzida na Seção anterior. O objetivo da Seção presente é propor uma maneira de implementar arquitetura Cleópatra, ou seja, partir da arquitetura Cleópatra, (a visão do programador em linguagem de montagem) e criar uma Organização Cleópatra (a visão do projetista de hardware).

2.1 A Interface de Entrada e Saída do Processador Cleópatra

Antes de tudo, é necessário estabelecer a relação entre o ambiente onde funcionará o processador e o próprio processador. De acordo com o modelo de von Neumann, processadores de programa armazenado interagem com a memória de dados e a memória de endereços, além de interagirem com dispositivos de entrada e de saída. Preocupar-nos-emos aqui apenas com a relação entre processador e memória[3]. Estabelecer a relação entre processador e memória consiste em definir precisamente (bit a bit, ou melhor, fio a fio) as entradas e saídas do processador para realizar a interface com a memória e com o mundo externo, o que é mostrado na Figura 2.1. A seguir, discutem-se as escolhas realizadas.

A proposta da Figura tem como objetivo simplificar ao máximo a interface entre o processador Cleópatra de um lado, e a memória e o mundo externo de outro. O mundo externo fornece três sinais de entrada: (*i*) o sinal de relógio **ck** (em inglês, “clock”), que define a cadência de realização das operações pelo hardware; (*ii*) o sinal de reinicialização **reset**, destinado a colocar o processador em um estado inicial conhecido (visto se tratar de um circuito seqüencial, como ver-se-á adiante); (*iii*) o sinal **hold**, que indica que o processador deve suspender a execução de ciclos de acesso à memória até que este sinal seja desativado. O sinal **hold** pode ser gerado por memórias ou dispositivos de entrada e saída lentos, para impedir que o processador leia um dado do seu barramento de entrada (ver abaixo a definição deste último) sem que a memória tenha tido tempo de produzir o dado solicitado, ou para fazer com que o processador mantenha um dado no seu barramento de saída para que um dispositivo ou memória lentos tenham tempo adicional para adquirir este dado do barramento. O mundo externo recebe do processador apenas um sinal, denominado **halt**, indicando que Cleópatra não está executando instruções no momento, um sinalizador, por exemplo, de fim de execução de um programa.

A interface processador-memória é mais complexa, pois implica troca de informações e acesso a diferentes posições do chamado mapa de memória. É necessário ter em mente que do ponto de vista funcional, uma memória de computador pode ser vista como uma tabela linear (ou, de modo equivalente, como um vetor), onde se pode ter acesso a apenas um elemento de cada vez, para escrita ou para leitura. Assim, a interface memória-processador consiste de:

- um conjunto de fios que especifica a posição da memória onde o processador deseja fazer acesso, o **barramento de endereços**. No caso do processador Cleópatra, da especificação retira-se que este barramento é de 8 fios, designando um mapa de memória de 256 posições. Assume-se aqui que a primeira posição da memória corresponde à configuração de 8 bits em zero 00000000_2 (ou em hexadecimal 00H), e que a última

[3] Na realidade, esta abordagem é hoje quase que geral, pois na maioria dos processadores modernos (os da Intel são uma exceção importante), faz acesso a todo dispositivo de entrada e/ou saída através de posições de memória reservadas para seu uso, no que costuma chamar de “Entrada e Saída Mapeada em Memória”, assunto a ser visto em detalhe nas disciplinas de Arquitetura de Computadores. A maior vantagem desta abordagem é que a arquitetura não precisa ter instruções especiais para fazer entrada e saída, podendo usar as mesmas instruções de acesso à memória (vários processadores da Intel possuem operações IN e OUT para fazer acesso à dispositivos de entrada e/ou saída.). De fato, mesmo em máquinas com processadores da Intel, os dispositivos de entrada e saída são mapeados em memória, descartando-se o uso das operações específicas de entrada e saída, que restam como um anacronismo arquitetural (salvo exceções especiais).

posição corresponde à configuração de 8 bits em um 11111111_2 (ou em hexadecimal 0FFH). As posições intermediárias seguem a convenção natural de números binários sem sinal. Usando uma notação similar a de vetores em linguagens de programação, designar-se-á este barramento de **ADDRESS[7:0]**;

- um fio para designar quando o processador está (ou não) fazendo acesso à memória, designado **ce** (do inglês "chip enable", significando habilita o ou um dos circuitos integrados ou "chips" de memória). Adota-se aqui a convenção que se este sinal estiver em 1 o processador está fazendo acesso à memória, se ele está em 0, não;
- um fio para designar que operação (se alguma está sendo realizada) o processador está realizando sobre a memória, designado **rw** (do inglês "read/write", leitura/escrita). Adota-se aqui a convenção que 1 neste sinal indica uma operação de leitura e 0 uma operação de escrita. Note-se que a interpretação deste sinal depende diretamente do valor que o sinal **ce** possui no momento. Se **ce** é 0, o valor de **rw** é irrelevante, senão este designa o tipo de operação.

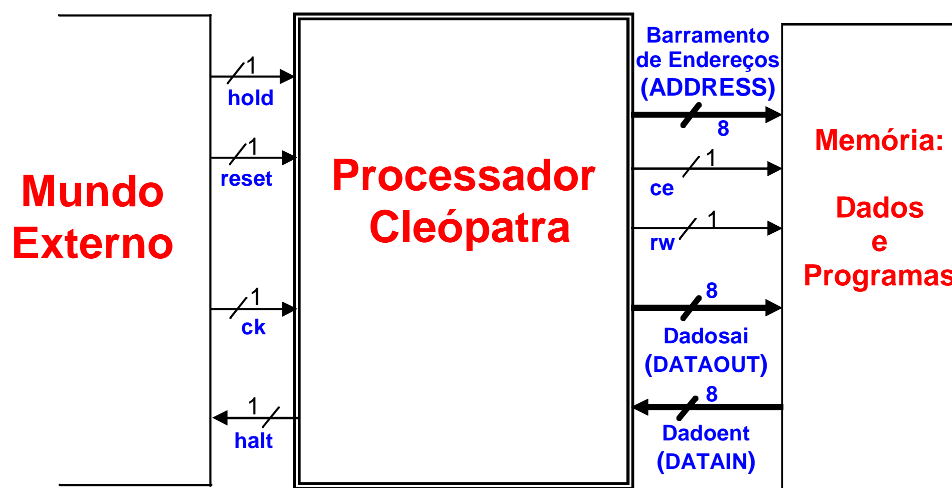


Figura 2.1 – O Processador Cleópatra com seu Ambiente (Memória e Mundo Externo).

- finalmente, dois conjuntos de fios para transportar a informação entre o processador e a memória, os **barramentos de dados**. No caso do processador Cleópatra, da especificação retira-se que estes barramentos são de 8 fios, designando uma palavra de memória de tamanho 8 bits. Usando uma notação similar à de vetores em linguagens de programação, designar-se-ão estes barramentos de **DATAIN[7:0]** e **DATAOUT[7:0]**.

Algumas organizações são propostas com um único barramento de dados bidirecional. O motivo desta escolha é simplificar a interface, reduzindo não apenas o número de sinais (fios) que conduzem dados entre o processador e memória, mas também o número de sinais de controle e simplificar o protocolo de comunicação entre memória e processador. Edições anteriores à versão 3.0 da organização Cleópatra usavam um barramento de dados único bidirecional. A mudança a partir da V3.0 decorre da facilidade de implementar uma organização sem fios bidirecionais internos, uma vez que o objetivo é implementar o subsistema processador-memória todo em um mesmo circuito integrado. Note-se que, como conseqüência desta mudança, alguns recursos, tais como o simulador do processador Cleópatra, podem ainda usar a notação antiga, onde o barramento de dados é único e bidirecional.

2.2 A Organização Interna do Processador Cleópatra Bloco de Dados e Bloco de Controle

Dada a interface definida na Seção anterior, é necessário agora estruturar a implementação das entranhas do processador Cleópatra. Seguindo fielmente o modelo de von Neumann, divide-se o processador em dois blocos que se comunicam, um para processar informação útil (os dados), denominado **bloco de dados** (BD), e outro para comandar as ações do primeiro bloco, de forma a conduzir a realização das operações de forma ordenada e correta, denominado por isto mesmo de **bloco de controle** (BC). Assim como a memória é um bloco passivo a mercê do processador, internamente, o bloco de dados é um escravo à serviço do bloco de controle. A Figura 2.2 ilustra a organização interna do processador e sua relação com a memória e o mundo externo. A Figura 2.2 requer explicação adicional, sobretudo no que tange os sinais de interface entre o bloco de dados e o bloco de controle.

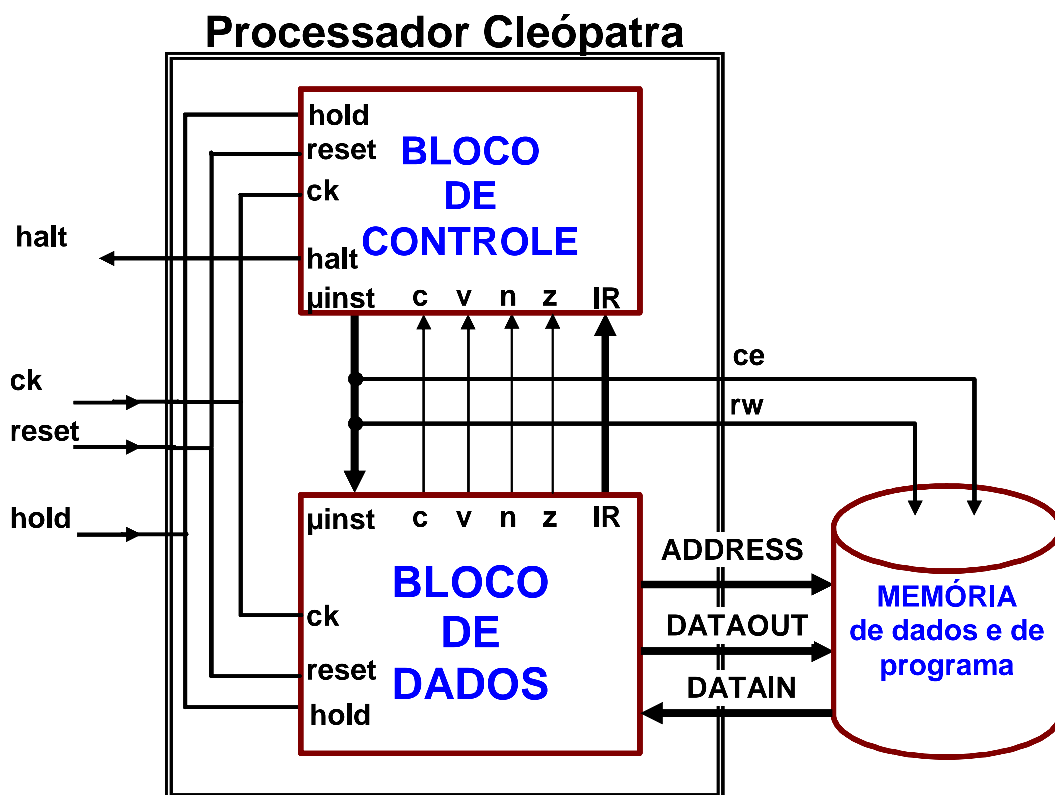


Figura 2.2 – Diagrama de blocos da arquitetura Cleópatra.

Primeiro, de acordo com o modelo de von Neumann, os sinais que vão do bloco de controle para o bloco de dados, denominados **comandos**, têm como objetivo especificar cada uma das ações realizadas no bloco de dados, que são:

- a operação que a ALU (*Arithmetic Logic Unit*) realiza a cada período de relógio;
- a conexão de saídas/entrada de registradores a entradas/saída da ALU;
- a habilitação de escrita de valores em registradores e/ou biestáveis de estado;
- o controle do valor a ser lido/escrito na memória.

Todos estes sinais são codificados em um conjunto de fios (mais um barramento)

denominado de **palavra de microinstrução**, ou simplesmente **microinstrução** (uma justificativa desta denominação é dada na Seção 4, durante a discussão detalhada do bloco de controle). Nesta proposta de organização para a arquitetura Cleópatra, este barramento será de 13 bits (justificados e definidos mais adiante).

Segundo, ainda de acordo com o modelo de von Neumann, os sinais que vão do bloco de dados para o bloco de controle, denominados **qualificadores**, têm como objetivo informar ao bloco de controle eventualidades ocorridas durante a realização das operações solicitadas, no caso:

- a ocorrência de vai-um na última casa de uma operação aritmética realizada pela ALU, o sinal denominado **c**. A interpretação é direta, se o sinal for 1, houve vai-um na última operação aritmética realizada pela ALU, caso contrário não houve vai-um;
- a ocorrência de transbordo[4] durante uma operação aritmética realizada pela ALU, o sinal denominado **v**. A interpretação é direta, se o sinal for 1, houve transbordo na última operação aritmética realizada pela ALU, caso contrário não houve transbordo;
- a ocorrência de um resultado igual a zero em uma operação realizada pela ALU, o sinal denominado **z**. A interpretação também é direta, embora às vezes gere uma certa confusão. Se o sinal for 1, a última operação que o afetou gerou zero como resultado, senão, o resultado foi diferente de zero;
- o sinal do resultado de uma operação realizada pela ALU, denominado **n**. A interpretação é a seguinte. Se o sinal for 1, a última operação que o afetou gerou um resultado negativo (representação em complemento de 2), senão o resultado é positivo.

A observação da Figura 2.2 leva à conclusão de que existe ainda um outro conjunto de sinais indo do bloco de dados para o bloco de controle, denominado **IR** (do inglês, "Instruction Register", ou Registrador de Instruções) Na realidade, este conjunto de sinais não são qualificadores e derivam de uma decisão de implementação que desvia-se do modelo geral de von Neumann. No modelo, toda a informação de controle flui pelo bloco de controle, e os registradores que transportam informação de controle como o **PC** (do inglês, "Program Counter", ou Contador de Programa), o **IR** e o **MAR** (do inglês, "Memory Address Register", ou Registrador de Endereços de Memória), estão conceitualmente no mesmo bloco. Contudo, em vários momentos durante a execução de programas, é necessário realizar operações aritméticas com estes registradores. Por exemplo, a cada ciclo de busca incrementa-se o **PC**; durante operações com modo de endereçamento relativo, os conteúdos do **PC** devem ser somados a um operando que vem da memória para gerar o endereço do dado. Assim, existem duas opções para organizar o processador:

- primeiro, seguir a risca o modelo de von Neumann, acrescentando capacidade para o bloco de controle realizar operações lógico-aritméticas, ou seja, ter uma ALU em cada um dos blocos. Esta é uma solução eficaz em termos de tempo de execução, mas cara em termos de hardware, usada em processadores modernos;
- segundo, migrar os registradores de controle para o bloco de dados e usar a mesma ALU para operações sobre dados e informação de controle, seqüencializando as operações da ALU. Esta é uma solução barata e lenta, usada em processadores de baixos custo e/ou desempenho.

Aqui, escolhe-se a solução de mais baixo custo. Assim, os conteúdos do **IR**, que designam a instrução a ser executada, devem ser passados do bloco de dados para o bloco de controle, para que este gere as microinstruções que implementam passo a passo cada instrução do programa em execução.

[4] Se a diferenciação entre vai-um e transbordo não for dominada, revisar representações de números com sinal em hardware, em particular a definição de representação de números em complemento de 2, assunto de Circuitos Digitais.

Em resumo:

- O Processador da proposta é composto por dois blocos (seguindo o modelo de von Neumann):
    1) Bloco de Dados, ou Parte Operativa, ou Caminho de Dados
    2) Bloco de Controle, ou Parte de Controle, ou Unidade de Controle
- Dados e programas são armazenados na memória externa
- Os sinais externos do processador são:
    1) **ck**, **reset** (entradas): o sinal de relógio **ck** é responsável pelo sincronismo da CPU e o sinal **reset** pela inicialização do processador
    2) **ce**, **rw, halt** (saídas): controlam a leitura/escrita na memória externa. **ce** quando em 1 indica para a memória que o processador está desejando realizar um acesso a algum conteúdo desta última. A linha **rw** indica o tipo de acesso (**rw**=1 indica leitura, **rw**=0 indica escrita). O sinal halt indica que o processador parou de executar instruções (devido à execução de uma instrução HALT).
    3) 2 barramentos de dados, **DATAIN e DATAOUT**: transportam dados de para e da memória respectivamente (ambos barramentos são de 8 bits)
    4) barramento de endereços **ADDRESS** (saída): endereço uma determinada palavra na memória (o barramento de endereços é de 8 bits)
- Comunicação entre o bloco de dados e bloco de controle (sinais internos ao processador):
    1) **µinst**: palavra de microinstrução, contém **13** sinais: 3 para escrita nos registradores, 3 para seleção de registrador, 3 para seleção da operação da ALU, **ce** (chip-enable), **rw** (read-write), **lcv** para habilitar carga no registrador de vai-um/transbordo e **lnz** para habilitar carga no registrador de negativo/zero[5].
    2) **IR**: instrução corrente, os conteúdos do registrador de instruções, **IR**.
    3) **c**, **v**, **n**, **z**: qualificadores (em inglês, "flags") gerados pela ALU do Bloco de Dados e armazenados em biestáveis.
- A Seção 3 deste documento explora a implementação do Bloco de Dados, conforme descrito nesta proposta.
- A Seção 4 deste documento explora a implementação do Bloco de Controle, conforme descrito nesta proposta.

[5] Os sinais **lcv** e **lnz** comandam simultaneamente a escrita em dois biestáveis (**c** e **v** / **n** e **z**, respectivamente), pois a análise da execução de cada instrução durante o projeto da organização Cleópatra indicou que sempre que é feita uma escrita em um dos elementos de cada um dos pares, é também feita no outro membro do par, e sempre que não há uma escrita em um também não a há no outro, tornando desnecessário se ter sinais separados de habilitação de escrita nos quatro elementos.

3. BLOCO DE DADOS

3.1 Estrutura Interna do Bloco de Dados

Existem inúmeras maneiras de se implementar o Bloco de Dados. A Figura 3.1 apresenta uma das possíveis propostas de implementação.

Figura 3.1 – Bloco de Dados da Arquitetura Cleópatra.

A primeira tarefa é estabelecer os registradores do Bloco de Dados.

Embora os registradores de controle situem-se conceitualmente no Bloco de Controle (na visão arquitetural), é interessante para a organização colocá-los no Bloco de Dados. O motivo desta decisão é aumentar a eficiência da implementação, pois várias operações aritméticas devem ser realizadas sobre registradores de controle, por exemplo durante o uso de endereçamento relativo. Assim, duas opções se colocam para viabilizar que os registradores de controle pertençam

fisicamente ao Bloco de Controle: adicionar uma ALU no Bloco de Controle para lidar com estas situações, ou acrescentar na interface entre Bloco de Controle e Bloco de Dados as entradas e saídas de cada registrador de controle que necessite operações aritméticas (no caso, pelo menos o Contador de Programa). Ambas soluções são caras em termos de hardware (ou ter duas ALUs na organização, ou acrescentar 16 fios à interface Bloco de Dados/Bloco de Controle). Assim, optou-se por mover todos os registradores de controle para o Bloco de Dados. Os registradores do Bloco de Dados incluem então todos os registradores da arquitetura (ver especificação na Seção 1 acima). Além destes, alguns registradores adicionais são necessários para implementar a organização. Acrescentou-se o mínimo de registradores, os dois que provêem acesso à memória (para endereços o MAR e para dados o MDR) e o fundamental registrador de instruções IR.

Uma segunda tarefa é codificar os comandos que o bloco de controle gera para o bloco de dados. Estabelecemos, arbitrariamente, as seguintes convenções.

Tabela 3.1 – Códigos de operação da Unidade Lógico-Aritmética.

Denominação	Codificação	Descrição
Soma	000	Soma o que está em BUS_A com o que está em BUS_B e coloca o resultado em BUS_C. Gera N, Z, C e V.
Inc	001	Toma o que está em BUS_A, soma à constante 1 e coloca o resultado em BUS_C (ignora BUS_B). Gera N, Z, C e V.
Não	010	Toma o que está em BUS_A, inverte todos os bits e coloca o resultado BUS_C (ignora BUS_B). Gera N e Z.
PassaB	100	Toma o que está em BUS_B e coloca em BUS_C (ignora BUS_A). Gera N e Z.
Ou	101	Toma o que está em BUS_A, faz o Ou bit a bit com o que está em BUS_B e coloca o resultado em BUS_C. Gera N e Z.
E	110	Toma o que está em BUS_A, faz o E bit a bit com o que está em BUS_B e coloca o resultado em BUS_C. Gera N e Z.
PassaA	111	Toma o que está em BUS_A e coloca em BUS_C (ignora BUS_B). Gera N e Z.

Tabela 3.2 – Códigos de escrita nos Registradores do BD.

Denominação	Codificação	Descrição
MAR	000	Escreve no registrador MAR.
MDR	001	Escreve no registrador MDR.
IR	010	Escreve no registrador IR.
PC	011	Escreve no registrador PC.
AC	100	Escreve no registrador AC.
RS	101	Escreve no registrador RS.
PC_MDR	110	Escreve simultaneamente nos registradores MDR e PC.
NULL	111	Não escreve em nenhum registrador.

Tabela 3.3 – Códigos de leitura dos Registradores do BD para BUS_A e BUS_B.

Denominação	Codificação	Descrição
NULL	000	Não coloca nada em BUS_A nem em BUS_B.
MDR	001	Lê o registrador MDR para BUS_B. Não coloca nada em BUS_A.
IR	010	Lê o registrador IR para BUS_B. Não coloca nada em BUS_A.
PC	011	Lê o registrador PC para BUS_A. Não coloca nada em BUS_B.
AC	100	Lê o registrador AC para BUS_A. Não coloca nada em BUS_B.
RS	101	Lê o registrador RS para BUS_A. Não coloca nada em BUS_B.
AC_MDR	110	Lê o registrador AC para BUS_A. Lê o registrador MDR para BUS_B.
PC_MDR	111	Lê o registrador PC para BUS_A. Lê o registrador MDR para BUS_B.

Uma terceira tarefa é estabelecer a configuração geral do Bloco de Dados, o que envolve definir as formas de interconexão entre registradores e a ALU e a forma de acesso à memória. A

escolha de uso de registradores como MAR e MDR define a estrutura de acesso à memória. Todo dado lido da memória passa por dentro do MDR e possui como endereço os conteúdos do MAR. Com relação à interconexão ALU/registradores, escolhemos a configuração mais óbvia que é prover três barramentos de acesso de oito bits: dois para levar os conteúdos de registradores para as entradas da ALU (**BUS_A** e **BUS_B** na Figura 3.1) e um para levar o resultado da operação de volta ao conjunto de registradores, também chamado de **banco de registradores**.

3.2 A Implementação de Instruções

Sabe-se que é possível realizar diversas operações simultâneas em hardware. Uma instrução de qualquer arquitetura é executada realizando diversas operações. Embora seja possível (em tese) implementar todas as operações requeridas por qualquer instrução de qualquer arquitetura simultaneamente (acrescentando hardware de forma a realizar todas as operações desta instrução ao mesmo tempo), em geral tal abordagem conduz a uma organização do processador extremamente cara, inviabilizando sua construção como um produto comercial. Para viabilizar o hardware do processador, seqüencializam-se as instruções, decompondo sua execução nas chamadas fases de execução. Normalmente, as fases de execução são três:

- Fase de busca (em inglês, "fetch") – é a leitura do código da instrução da memória de programa;
- Fase de decodificação (em inglês, "decode") – segue-se à primeira, identifica a instrução;
- Fase de execução (em inglês, "execute") – sucede as duas anteriores, e compreende a realização da instrução propriamente dita.

Mesmo a decomposição de instruções em fases pode não ser suficiente para viabilizar a construção de processadores economicamente viáveis que implementem as fases cada uma em um período de relógio. Assim, cada fase pode ser decomposta em unidades mais simples, executáveis cada uma em exatamente um período de relógio. A estas unidades dá-se o nome de **microinstruções**. Ou seja, define-se microinstrução como um conjunto de operações que uma determinada organização pode executar simultaneamente em um período de relógio. Cada uma das operações é coerentemente denominada de **microoperação**. Após detalhar todas as instruções em fases, as fases em microinstruções e as microinstruções em microoperações, o trabalho do projetista de hardware fica completamente definido, e a implementação pode ser iniciada. Um conjunto de todas as microinstruções que definem completamente a operação da arquitetura constitui um **microprograma**. Cada **instrução** de um **programa** em linguagem de montagem pode ser detalhada em um trecho deste microprograma. O objetivo da Seção 3.3 é detalhar um trecho de programa em linguagem de montagem em um trecho de microprograma equivalente, que é executado pelo hardware da organização Cleópatra. A Seção 4 dedica-se a estudar o microprograma completo para executar qualquer programa, composto por qualquer número de instruções.

3.3 Microssimulação do Bloco de Dados

A tabela abaixo ilustra um exemplo de programa a ser executado no Bloco de Dados. Este programa insere o valor 33H no AC, somando-o em seguida com o conteúdo da posição de memória 83H (inicializado com 0D6H), colocando o resultado na posição 18H (posição inicializada com 00H).

3.3.1 Programa exemplo

Endereço de memória	Instruções	Código objeto	
		Em Binário	Em Hexadecimal
0 e 1	LDA #33H	01000000 00110011	40 33
2 e 3	ADD 83,D	01010100 10000011	54 83
4 e 5	STA 18H,D	00100100 00011000	24 18
Endereço	Dados	Dados em Binário	Dados em Hexa
18	00H	00000000	00
…	…	…	…
83	0D6H	11010110	D6

3.3.2 Valores esperados no barramento de dados e nos sinais de controle por ciclo

```
datain  dataout write_reg read_reg ALU_op  ce  rw  lnz  lcv

  --      --      0          3       7      0   0   0    0  ; busca LDA #
  40      --      6          3       1      1   1   0    0
  --      --      2          1       4      0   0   0    0

  --      --      0          3       7      0   0   0    0  ; executa LDA #
  33      --      6          3       1      1   1   0    0
  --      --      4          1       4      0   0   1    0

  --      --      0          3       7      0   0   0    0  ; busca ADD ,D
  54      --      6          3       1      1   1   0    0
  --      --      2          1       4      0   0   0    0

  --      --      0          3       7      0   0   0    0  ; executa ADD ,D
  83      --      6          3       1      1   1   0    0
  --      --      0          1       4      0   0   0    0
  D6      --      1          0       7      1   1   0    0
  --      --      4          6       0      0   0   1    1

  --      --      0          3       7      0   0   0    0  ; busca STA ,D
  24      --      6          3       1      1   1   0    0
  --      --      2          1       4      0   0   0    0

  --      --      0          3       7      0   0   0    0  ; executa STA ,D
  18      --      6          3       1      1   1   0    0
  --      --      0          1       4      0   0   0    0
  --      09      7          4       7      1   0   0    0
```

3.3.3 Uma linguagem de Micromontagem

Para descrever o detalhamento da execução de instruções do código objeto, introduz-se agora uma linguagem muito simples que está diretamente relacionada com ações do hardware. Cada comando desta linguagem de montagem corresponde a uma ou a um conjunto muito pequeno de microoperações. Um conjunto de microoperações separados pelo caractere vírgula e terminado pelo caractere ponto corresponde a uma microinstrução, ou seja, a um conjunto de microoperações executadas em paralelo em exatamente um período de relógio. A linguagem de micromontagem, está descrita esquematicamente na Tabela 3.4 abaixo. Nela, Reg_i representa qualquer dos registradores da arquitetura Cleópatra (AC, PC, RS, IR, MDR ou MAR), PMEM(Reg_i) significa o conteúdo da posição de memória cujo endereço está contido no registrador Reg_i, CMD indica qualquer um dos comandos do Bloco de Controle para o Bloco de Dados (ALU_OP, WRITE_REG, READ_REG, LCV, LNZ, CE e RW), e **op** significa qualquer um dos operadores da arquitetura

Cleópatra (não, e, ou, soma). Várias das gerações de comandos são implícitas, para evitar uma linguagem inutilmente detalhada. Por exemplo uma escrita na memória não detalha o acionamento dos comandos CE e RW. Uma observação importante é que os comandos de 1 bit têm um estado ativo (na Cleópatra sempre correspondem ao valor 1) e um estado inativo. Outra observação é que se deve atentar para a diferença existente entre comandos que estão ativos por exatamente um ciclo de relógio e outros, cujo efeito dura até que outro comando altere seu efeito, como ocorre no caso de escrita em registradores.

Tabela 3.4 – Comandos da Linguagem de Micromontagem.

Sintaxe	Descrição da microoperação ou microinstrução
$Reg_1 \leftarrow Reg_2$	Transfere (ou seja, copia) os conteúdos de Reg_1 para Reg_2. Ao final, ambos possuem o mesmo dado.
$Reg_1 \leftarrow$ PMEM (Reg_2)	Escreve (ou seja, copia) os conteúdos da posição de memória cujo endereço está em Reg_2 em Reg_1.
Reg++	Soma 1 aos conteúdos atuais de Reg e coloca o resultado de volta em Reg.
CMD=XXX	Gera o comando de n bits CMD com o valor XXX por exatamente um ciclo de relógio.
$Reg_1 \leftarrow$ **op** Reg_2	Realiza a operação unária **op** sobre os conteúdos de Reg_2 e coloca o resultado em Reg_1.
$Reg_1 \leftarrow Reg_2$ **op** Reg_3	Realiza a operação binária **op** sobre os conteúdos de Reg_2 e Reg_3 e coloca o resultado em Reg_1.
PMEM(Reg_1) $\leftarrow Reg_2$	Escreve (ou seja, copia) os conteúdos de Reg_2 na posição de memória cujo endereço está em Reg_1.
$\mu op1$, $\mu op2$, ..., μopn.	Uma microinstrução, ou seja, um conjunto de n microoperações executadas em exatamente 1 ciclo de relógio.

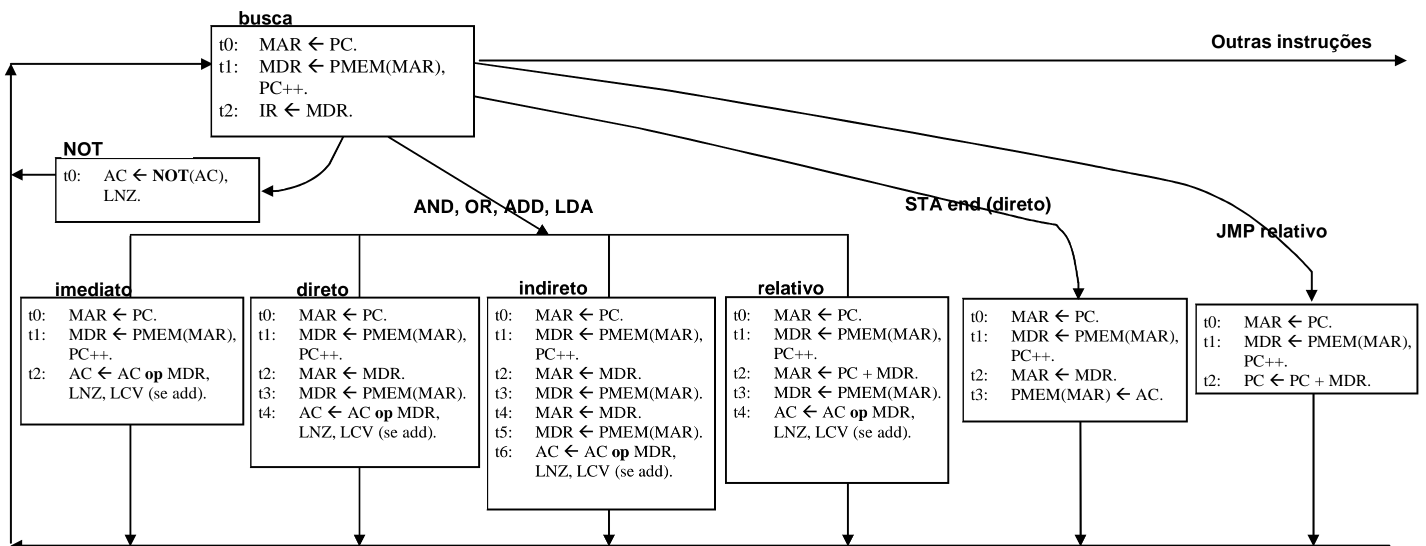
A seguir, usar-se esta linguagem para descrever a execução do programa da Seção 3.3.1.

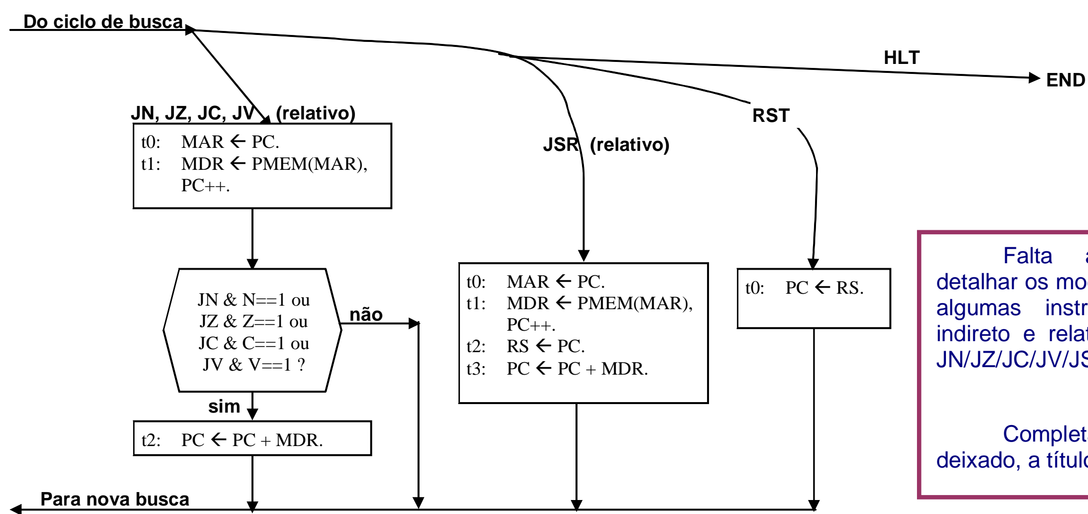
3.4 Resultados da Microssimulação

A tabela abaixo descreve a execução do trecho de programa exemplo descrito na Seção 3.3.1 a cada ciclo de relógio. Ela mostra o valor de sinais de controle gerados pelo Bloco de Controle (colunas write_reg a lcv), a informação nos barramentos de dados (DIN e DOUT) e de endereços (ADDRESS), os conteúdos de registradores do Bloco de Dados (colunas ADDRESS a RS), os conteúdos dos flip-flops qualificadores (coluna N Z C V) e a microinstrução executada a cada ciclo de relógio (coluna Microcódigo). ***Importante***: o efeito na memória da dados e instruções está sendo percebido via os barramentos de dados (se CE=0, -- indica que dado no barramento é qualquer e irrelevante). Os valores nos registradores são aqueles obtidos após a execução da microinstrução. O microcódigo traduz, em linguagem de micromontagem a operação indicada pelos sinais gerados no BC. Em menos da metade dos ciclos de relógio se faz acesso à memória. A última coluna à direita reúne microinstruções em ciclos de máquina: busca e execução. Todos os dados na tabela abaixo estão em hexadecimal, exceto o índice do sinal de relógio CK. Assume-se que registradores e biestáveis de qualificadores estão inicialmente zerados. ***Obs:*** a segunda linha da *busca* realiza 2 operações em paralelo. As linhas no final da primeira e segunda fases de execução de instrução acionam a escrita dos biestáveis qualificadores. Em vermelho aparecem os valores alterados pela microinstrução.

CK	DIN	DOUT	ALU _op	write _reg	read _reg	lnz	lcv	ce	rw	ADDR ESS	MDR	IR	PC	AC	RS	N Z C V	Microcódigo	*Ciclo de Máquina*
0	--	--	7	0	3	0	0	0	0	00	00	00	00	00	00	0 0 0 0	MAR ← PC.	**Busca LDA ,#**
1	**40**	--	1	6	3	0	0	1	1	00	40	00	01	00	00	0 0 0 0	MDR ← PMEM(MAR), PC++.	
2	--	--	4	2	1	0	0	0	0	00	40	40	01	00	00	0 0 0 0	IR ←MDR.	
3	--	--	7	0	3	0	0	0	0	01	40	40	01	00	00	0 0 0 0	MAR ← PC.	**Execução LDA ,#**
4	**33**	--	1	6	3	0	0	1	1	01	33	40	02	00	00	0 0 0 0	MDR ←PMEM(MAR), PC++.	
5	--	--	4	4	1	1	0	0	0	01	33	40	02	33	00	0 0 0 0	AC ← MDR, LNZ.	
6	--	--	7	0	3	0	0	0	0	02	33	40	02	33	00	0 0 0 0	MAR ← PC.	**Busca ADD ,D**
7	**54**	--	1	6	3	0	0	1	1	02	54	40	03	33	00	0 0 0 0	MDR ← PMEM(MAR), PC++.	
8	--	--	4	2	1	0	0	0	0	02	54	54	03	33	00	0 0 0 0	IR ←MDR.	
9	--	--	7	0	3	0	0	0	0	03	54	54	03	33	00	0 0 0 0	MAR ← PC.	**Execução ADD ,D**
10	**83**	--	1	6	3	0	0	1	1	03	83	54	04	33	00	0 0 0 0	MDR ←PMEM(MAR), PC++.	
11	--	--	4	0	1	0	0	0	0	83	83	54	04	33	00	0 0 0 0	MAR ← MDR.	
12	**D6**	--	7	1	0	0	0	1	1	83	D6	54	04	33	00	0 0 0 0	MDR ←PMEM(MAR).	
13	--	--	0	4	6	1	1	0	0	83	D6	54	04	09	00	0 0 1 0	AC ← AC + MDR, LNZ, LCV.	
14	--	--	7	0	3	0	0	0	0	04	D6	54	04	09	00	0 0 1 0	MAR ← PC.	**Busca STA ,D**
15	**24**	--	1	6	3	0	0	1	1	04	24	54	05	09	00	0 0 1 0	MDR ← PMEM(MAR), PC++.	
16	--	--	4	2	1	0	0	0	0	04	24	24	05	09	00	0 0 1 0	IR ←MDR.	
17	--	--	7	0	3	0	0	0	0	05	24	24	05	09	00	0 0 1 0	MAR ← PC.	**Execução STA ,D**
18	**18**	--	1	6	3	0	0	1	1	05	18	24	06	09	00	0 0 1 0	MDR ←PMEM(MAR), PC++.	
19	--	--	4	0	1	0	0	0	0	18	18	24	06	09	00	0 0 1 0	MAR ← MDR.	
20	**09**	**09**	7	7	4	0	0	1	0	18	18	24	06	09	00	0 0 1 0	PMEM(MAR) ← AC.	

3.5 Diagrama (Parcial) do Microcódigo executado pelo Bloco de Dados





Falta ainda neste fluxograma detalhar os modos de endereçamento para algumas instruções. Entre elas: STA indireto e relativo, JMP direto e indireto; JN/JZ/JC/JV/JSR direto e indireto.

Completar estas instruções é deixado, a título de exercício, aos alunos.

4. BLOCO DE CONTROLE

4.1 Introdução

O objetivo desta Seção é introduzir duas propostas diferentes de organização do Bloco de Controle da arquitetura Cleópatra, pressupondo que a organização do Bloco de Dados é aquela proposta na Seção 3 deste documento. Note-se que estas não são as únicas organizações possíveis, nem mesmo as melhores. O critério didático foi usado para guiar as propostas. Estas são suficientemente detalhadas para permitir a implementação completa das versões do Bloco de Controle em questão, detalhando inclusive a codificação binária de comandos e sinais.

Inicialmente, é necessário perceber que esta organização é bem mais complexa que a do Bloco de Dados, ou pelo menos, de estrutura bem menos intuitiva que este.

Um Bloco de Controle implementado como um circuito síncrono segue normalmente o modelo geral denominado Máquina de Estados Finita Síncrona (MEF ou FSM, do inglês *Synchronous Finite State Machine*), que pode ser representado pelo desenho da Figura 4.1.

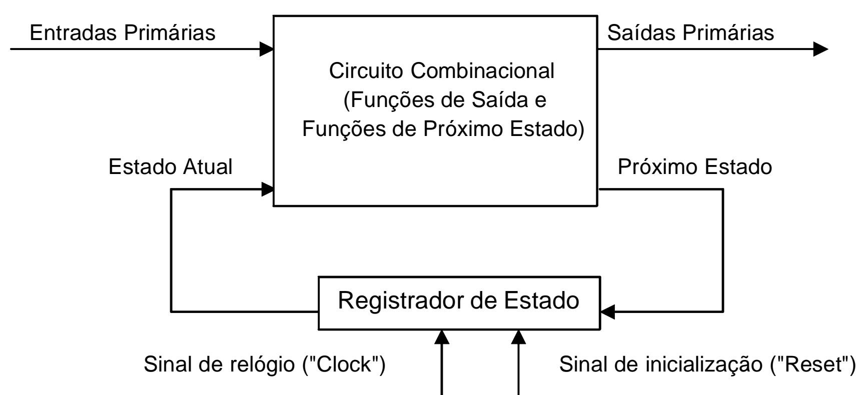


Figura 4.1 - O modelo Máquina de Estados Finita Síncrona.

Este modelo é composto de um bloco combinacional que computa as *funções de saída* e as *funções de próximo estado* de um controlador, no caso o Bloco de Controle do processador Cleópatra. O estado da máquina é armazenado em um registrador a cada novo período de relógio. Ao mesmo tempo, o circuito combinacional computa novas saídas e o novo próximo estado.

4.2 Interface de Entrada e Saída do Bloco de Controle

Ver-se-á nesta Seção a estrutura e uma descrição funcional do comportamento dos sinais de entrada e saída do Bloco de Controle. As Seções 4.3 e 4.4 detalham as duas implementações alternativas do Bloco de Controle, uma usando máquinas de estado e outra usando microprogramação.

As entradas primárias do Bloco de Controle provêm do Bloco de Dados, e são transportadas por 11 fios:

- 6 fios transportam a parte mais significativa do Registrador de Instruções IR, os quatro primeiros contendo o código de operação da instrução corrente, e os dois últimos contendo o código do modo de endereçamento da mesma instrução;

- 4 fios que são os qualificadores gerados pelos flip-flops conectados à saída da Unidade Lógico-

Aritmética (ALU) do Bloco de Dados. A saída destes elementos biestáveis serve para o Bloco de Controle tomar decisões. No caso da máquina Cleópatra, estes são usados apenas nas instruções de salto condicional JN, JZ, JC e JV;

- 1 fio corresponde ao sinal de controle HOLD. Este, quando em '1' indica que o acesso à memória atualmente em curso deve ser prolongado por um número de ciclos de relógio definido pela duração do sinal ativo, o acesso sendo concluído apenas um no ciclo de relógio após a desativação de HOLD.

As saídas primárias CE e RW, por outro lado, vão não apenas para o Bloco de Dados, mas igualmente para a Memória Principal, controlando seu funcionamento, ou seja, as operações de escrita e leitura. Em particular, devido à forma como foi concebido o Bloco de Dados, os sinais que vão para a Memória Principal são igualmente aproveitados neste. O número total de saídas primárias do Bloco de Controle é estabelecido pelos comandos a serem gerados para controlar a Memória Principal e o Bloco de Dados. Lembrando que o Bloco de Dados contém os registradores MAR e MDR, sabe-se que os Barramentos de Dados e o de Endereços são parte do Bloco de Dados. Isto ocorre mesmo que o MAR esteja, conceitualmente no Bloco de Controle, o que é definição arquitetural, mudada por questões de desempenho e facilidade de implementação no momento de criar a organização da máquina (ver Seção 3.1). Assim, ao Bloco de Controle resta a tarefa de gerar os dois sinais de comando de acesso à memória, RW (RW=1 indica uma operação de leitura da memória e RW=0 indica uma operação de escrita) e CE (*Chip Enable*, também chamado de *Strobe*, STR, ou *Chip Select*, CS), que indica o momento exato da realização da operação especificada na linha RW. Outra saída primária é o sinal HALT, que indica, quando ativo (HALT='1'), que o processador executou a instrução HLT e como conseqüência parou de executar o ciclo eterno de busca/decodificação/execução.

Além das três saídas primárias, o Bloco de Controle fornece outras 11 saídas, para controlar os seguintes recursos (para mais informações, consultar a definição do Bloco de Dados):

- 3 fios para determinar a operação a ser realizada pela ALU, denominados coletivamente de ALU_OP. ALU_OP pode assumir 7 códigos possíveis, batizados: ADD (Código binário 000 para ALU_OP[2:0]), INC (Código binário 001), NOT (Código binário 010), PSB (Código binário 100), OR (Código binário 101), AND (Código binário 110) e PSA (Código binário 111), e um dos códigos (Código binário 011) não é jamais gerado, ou seja, não é usado;

- 3 fios para determinar o registrador a ser carregado com o valor atualmente na saída da ALU, denominados coletivamente de WRITE_REG. WRITE_REG pode assumir a cada instante1 de 8 códigos possíveis, codificados com o mesmo nome dos registradores: MAR (Código binário 000 para WRITE_REG[2:0]), MDR (Código binário 001), IR (Código binário 010), PC (Código binário 011), AC (Código binário 100), RS (Código binário 101). Um caso especial a ser considerado é a ativação simultânea de duas escritas. O valor MDPC (Código binário 110), corresponde a escritas no registrador MDR, com o dado vindo da memória (pois este pode ser escrito com um dado que não provém da ALU, ao contrário de todos os outros registradores), e no PC. Finalmente existe o código NOW (Código binário 111), que indica que nenhuma escrita deve ser realizada;

- 3 fios para determinar o(s) registrador(es) a ser(em) conectados aos barramentos BUS_A e BUS_B, denominados coletivamente de READ_REG. READ_REG pode assumir também 1 de 8 códigos possíveis. Note-se que o controle simultâneo para estes barramentos pode gerar até dois comandos simultâneos de leitura, decodificados separadamente pelo Bloco de Dados. Os códigos gerados pelo controle e sua interpretação são: MDR (Código binário 001), que conecta a saída do registrador MDR a BUS_B, IR (Código binário 010), que conecta a saída do registrador IR a BUS_B, PC (Código binário 011), que conecta a saída do registrador PC a BUS_A, AC (Código binário 100), que conecta a saída do registrador AC a BUS_A, RS

(Código binário 101), que conecta a saída do registrador IR a BUS_A, MDAC (Código binário 110), que conecta a saída do registrador MDR a BUS_B e a saída de AC a BUS_A, e MDPC (Código binário 111), que conecta a saída do registrador MDR a BUS_B e a saída de PC ao BUS_A. Um caso especial é a operação NOP (Código binário 000), que nunca é usado;

- 1 fio para comandar a escrita dos biestáveis que armazenam os qualificadores C e V (C=0 indica não ocorrência de vai-um e C=1 indica ocorrência para a última instrução que o afetou, enquanto que V=0 indica a não ocorrência de transbordo e V=1 indica ocorrência para a última instrução que o afetou), denominado LCV (LCV=1 indica que a próxima borda de relógio deve carregar novos valores nos biestáveis C e V). Percebe-se que a única instrução da arquitetura Cleópatra que afeta estes biestáveis é ADD. Nenhum outro uso da ALU afeta estes "flags".

- 1 fio para comandar simultaneamente a escrita dos biestáveis que armazenam os qualificadores N (N=0 indica resultado positivo e N=1 indica resultado negativo para a última instrução que o afetou) e Z (Z=0 indica resultado não nulo e Z=1 indica resultado nulo para a última instrução que o afetou), denominado LNZ (LNZ=1 indica que o relógio deve carregar um novo valor nos biestáveis N e Z). Perceba-se que devido à especificação da arquitetura Cleópatra, nunca é necessário realizar uma operação de escrita sobre N em separado de Z. Lembra-se aqui que das catorze instruções da arquitetura Cleópatra. apenas as que eventualmente alteram o conteúdo do registrador Acumulador (AC) afetam os qualificadores N e Z (NOT, LDA, ADD, OR e AND).

A interface de entrada e saída do Bloco de Controle aparece na Figura 4.2.

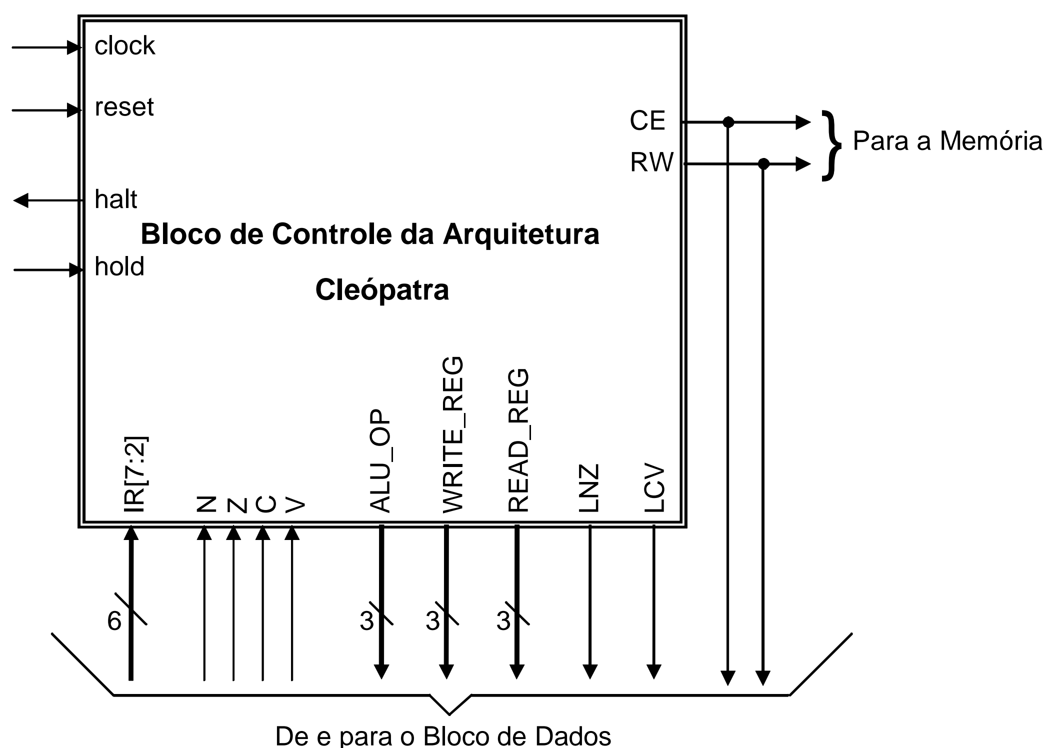


Figura 4.2 - Interface de Entrada e Saída do Bloco de Controle da Arquitetura Cleópatra.

4.3 Bloco de Controle - Implementação com Máquina de Estados

Dada a Interface de Entrada e Saída descrita na Seção anterior, pode-se apresentar a estrutura interna do Bloco de Controle quando implementado como uma máquina de estados finita tradicional.

Como descrito antes, cada instrução do processador é executada em três fases, busca, decodificação e execução. Cada fase exige certo número de ciclos de relógio para ser executada, número este determinado pela estrutura do Bloco de Dados vista na Seção 3 deste documento. Conforme visto durante a explicação de microssimulações, a busca leva três ciclos de relógio para ser executada. No terceiro destes ciclos também é executada a decodificação de instruções. A fase de execução dura uma quantidade variável de ciclos, que depende da instrução e do modo de endereçamento específicos.

A Figura 4.3 descreve uma implementação da máquina de estados para o Bloco de Controle do Processador Cleópatra sob a forma de uma máquina de Mealy. A implementação possui 15 estados.

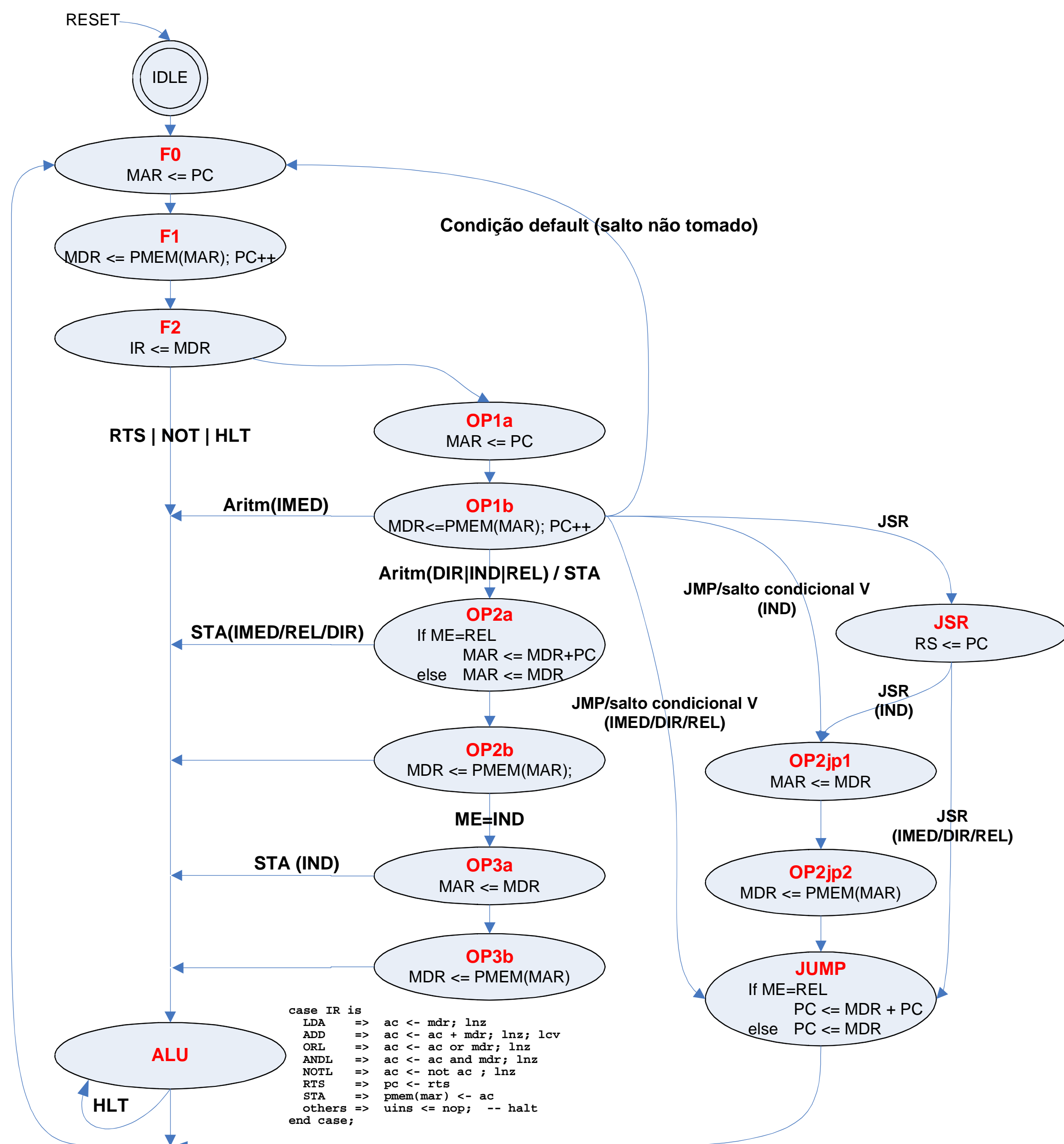


Figura 4.3 – Uma implementação do bloco de Controle do processador Cleópatra sob a forma de uma máquina de estados finita. Todos os testes realizados para percorrer uma dada aresta do grafo que define esta máquina de estados são realizados sobre o valor do registrador de instruções IR. A implementação pressupõe que a transição de estados se dá a cada borda de subida do sinal de relógio. Considerações do efeito da ativação do sinal **hold** na máquina de estados foram omitidas. Assume-se que a ativação deste

suspende a evolução de estados, mantendo a máquina no estado corrente até sua desativação.

O primeiro estado desta máquina (**IDLE**) serve apenas para garantir a sincronização da operação de inicialização do processador. Ele força que o primeiro estado relevante de operação, o primeiro ciclo da busca, aconteça a partir da primeira borda de subida de relógio que suceder à desativação do sinal de **reset** (ou seja, **reset** ir para '0').

Seguem-se os três estados de busca de instrução, executados sempre em seqüência (**F0**, **F1**, **F2**). Cada um destes executa incondicionalmente uma microinstrução. A seguir ocorre a decodificação, que detecta, a partir do valor escrito em IR no estado **F2** que rumo tomar durante a fase de execução da instrução. Há dois caminhos possíveis, um para instruções sem operando (RTS, NOT e HLT) e um para as instruções com operando (demais).

No primeiro caminho, o único estado antes da finalização da instrução é o estado **ALU**, após o que há a volta para o ciclo de busca. A exceção é a instrução HLT, que neste estado faz o processador entrar em laço, do qual este só pode escapar através da geração de um sinal externo de inicialização pelo mundo externo ao processador (reset). O estado **ALU** é também usado pelas instruções com operando, o que se pode notar pela reconvergência do segundo caminho na representação da máquina de estados. Neste estado, o valor do IR é testado para determinar a operação específica a realizar (ver comando VHDL **case** ao lado do estado, na Figura).

O segundo caminho começa com dois estados que servem para realizar a busca do operando (**OP1a**, **OP1b**), sendo estes comuns a todas as instruções que passam por tal caminho. Após o segundo destes estados (**OP1b**), existe uma tomada de decisão sobre qual dentre 6 sub-caminhos seguir: um caminho para instruções aritméticas e LDA com modo imediato (direto para estado **ALU**); um para as operações lógicas/aritméticas e LDA com modos de endereçamento diferentes de imediato e para a instrução STA com qualquer modo de endereçamento (seqüência que começa com estado **OP2a**); um para saltos condicionais onde a condição de salto não é verdadeira (volta para a busca em **F0**); um para executar a instrução JSR (**JSR**); um para executar saltos incondicionais (JMP) e saltos condicionais com condição de salto verdadeira onde o modo de endereçamento usado é indireto (seqüência de estados iniciando no estado **OP2jp1**); e um caminho para executar saltos incondicionais (JMP) e saltos condicionais com condição de salto verdadeira onde o modo de endereçamento usado é diferente do indireto (estado **JUMP**).

A partir da Figura 4.3, das explicações incompletas acima e do diagrama da Seção 3.5, é possível compreender completamente o funcionamento desta implementação do Bloco de Controle. Um detalhe importante omitido na Figura 4.3 é que em qualquer dos estados da máquina a evolução de estados pode ser suspensa mediante acionamento do sinal externo **hold**. Isto poderia ser considerado na Figura acrescentado um laço em todos os estados da máquina saindo do estado e entrando nele mesmo, com um rótulo **hold**='1'. Isto não foi feito para evitar deixar o diagrama pouco claro.

4.4 Bloco de Controle - Implementação Microprogramada

A partir da Interface de Entrada e Saída, pode-se discutir a estrutura interna do Bloco de Controle. A cada período de relógio, um novo conjunto de comandos deve ser gerado no Bloco de Controle para comandar a Memória Principal e o Bloco de Dados. Isto está de acordo com o comportamento de uma Máquina de Estados Finita, que a cada relógio gera um conjunto de saídas e realiza uma transição de estado, e ambos, saída e próximo estado são determinados unicamente pelo estado atual (armazenado no registrador de estados) e pelas entradas atuais.

Lembrando a nomenclatura para instruções de uma arquitetura, já mencionada na Seção 3, temos que:

- uma *instrução* de uma arquitetura é um conjunto de ciclos de máquina;

- um *ciclo de máquina* é composto de um conjunto de microinstruções;

- uma *microinstrução* é um conjunto de microcomandos realizados em paralelo durante exatamente um período de relógio;

- um *microcomando* é uma operação unitária realizada pelo hardware em um único período de relógio.

Exemplos de microcomandos são uma escrita em um registrador ou a realização de uma operação pela ALU, ou ainda a conexão de uma saída de registrador a um barramento do tipo *tristate*. Exemplo de microinstrução é uma leitura da memória (pois envolve vários microcomandos, todos realizados em paralelo durante exatamente um período de relógio: uma ativação de sinal de operação - RW, um comando de execução, CE, e uma escrita em um registrador, MDR). Um exemplo de ciclo de máquina é o ciclo de busca, que leva vários períodos de relógio, cada um executando uma série de microcomandos em paralelo. Exemplo de instrução é LDA da arquitetura Cleópatra, que toma vários ciclos de máquina (busca, decodificação e execução), cada um necessitando de vários períodos de relógio para ser executado.

Um circuito combinacional pode ser implementado como uma memória ROM, onde cada linha da memória corresponde a um conjunto de saídas simultâneas do circuito, determinadas exclusivamente pelo valor instantâneo das entradas. As entradas de um circuito combinacional correspondem desta forma a endereços da memória ROM. Esta será a organização empregada para construir o Bloco de Controle da máquina Cleópatra. Todos os diferentes conjuntos de saídas (comandos) são armazenados em uma ROM, e geram-se endereços na ordem correta para implementar os ciclos de máquina (Busca/Decodificação/Execução) de cada instrução da arquitetura. Ou seja, em cada linha da ROM armazena-se uma microinstrução, que corresponde a um conjunto de valores para ativar (ou não) um conjunto de microcomandos a cada período de relógio.

A Figura 4.4 detalha a estrutura interna do Bloco de Controle, usando uma ROM, denominada microROM ou µROM. Além da µROM, existe um bloco denominado Seqüenciador, um subsistema que calcula o próximo endereço da µROM a cada período de clock. Note que toda a "inteligência" do Bloco de Controle está na organização dos conteúdos da µROM e no comportamento do seqüenciador.

4.4.1 Estrutura interna da µROM

Estuda-se agora a estrutura da µROM. Nota-se que os conteúdos desta estão divididos em campos de bits, cada campo correspondendo a um conjunto de colunas da µROM. À exceção das duas colunas de 1 bit mais à esquerda, todos os campos são saídas do Bloco de Controle, 11 deles para o Bloco de Dados, e 2 (RW e CE) para o Bloco de Dados e para a Memória Principal.

Todos os campos da µROM são de um bit, exceto ALU_OP, WRITE_REG e READ_REG, que são de três bits cada, de acordo com a definição dos comandos do Bloco de Dados.

Os pressupostos de organização da ROM são os seguintes:

- As microinstruções que implementam o ciclo de busca de instrução estão localizadas seqüencialmente a partir do endereço 0 da ROM, para ser mais exato nos endereços 0, 1 e 2, uma vez que durante a microssimulação do Bloco de Dados, já descobriu-se que são necessários exatos 3 períodos de clock (portanto 3 microinstruções) para buscar qualquer instrução;

- No último período de relógio do ciclo de busca, é feita a decodificação. Logo, nenhuma microinstrução separada é necessária para este ciclo de máquina;

- Existem 14 instruções na arquitetura Cleópatra. Da microssimulação, sabe-se que as instruções

gastam de 1 a 7 períodos de relógio para realizar o ciclo de execução, nunca mais do que isto. As instruções sem operando explícito levam todas apenas um período de clock na fase de execução. As instruções de salto e a instrução STA não usam o modo de endereçamento imediato, pois ou ele não faz sentido, ou é tratado da mesma forma que o modo direto;

Cada modo de endereçamento implica um funcionamento distinto da fase de execução de uma instrução. Cada instrução tem de 0 a 4 modos de endereçamento, e assim pode-se dimensionar o número de linhas da µROM por uma análise de pior caso: Número de linhas máximo da µROM = 3 (instruções sem operando) + 3 (ciclos de busca e decodificação) + 7*7*3 (Saltos e STA) + 4*7*4= 265 linhas. Assim, precisa-se no pior caso de 9 bits de endereço para a saída do seqüenciador, e uma µROM de 265 posições. Os números exatos são 8 bits e 136 posições.

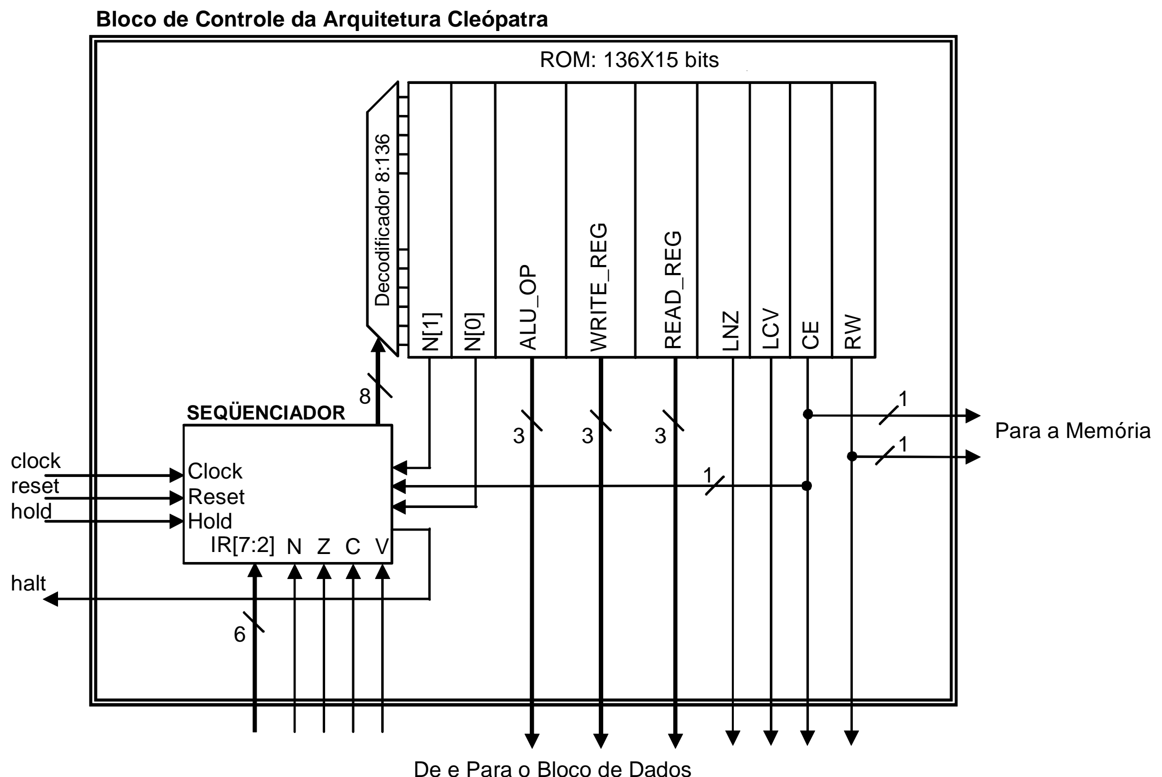


Figura 4.4 - Estrutura Interna do Bloco de Controle da Arquitetura Cleópatra.

4.4.2 Interface externa do seqüenciador

Volta-se agora a atenção para a interface externa do seqüenciador. Alguns dos sinais desta interface requerem explicação adicional. A função do seqüenciador, como já foi dito, é gerar os endereços para a µROM a cada período de relógio, de forma a estabelecer a ordem de execução de microinstruções armazenadas nesta. Para tanto, note-se que o seqüenciador recebe nas suas entradas todas as informações necessárias para computar a próxima linha da µROM a ser selecionada:

- Os seis bits mais significativos do registrador IR, indicando a instrução e o modo de endereçamento a empregar;
- Os flags N, Z, C, V usados para tomadas de decisão no caso das instruções JN, JZ, JC, e JV, respectivamente;
- Os dois sinais de *controle de execução de próxima microinstrução*, N[1:0], provenientes da

µROM, e o sinal de CE.

Com os dois sinais de controle N[1:0], é possível especificar 4 operações de controle distintas:

- Incrementar o endereço (INC, código binário 00) - a operação mais comum, pois as microinstruções que devem ser executadas em seqüência estão, no mais das vezes, localizadas em posições contíguas da µROM;
- Saltar para o endereço 0 (JZER, código binário 01) - operação gerada sempre que um ciclo de execução de instrução termina, garantindo que a próxima tarefa a executar é um ciclo de busca de nova instrução;
- Saltar para um determinado endereço da µROM (LOAD, código binário 10) - usado exclusivamente no último período de relógio do ciclo de busca, para determinar que o próximo endereço da µROM é aquele ditado pelos conteúdos do IR (código da operação e modo de endereçamento). Corresponde ao ciclo de decodificação de instruções ilustrado pela Figura na Seção 3.5 (as múltiplas setas derivando do ciclo de busca);
- Saltar condicionalmente para um determinado endereço da µROM (JCOND, código binário 11) - usado exclusivamente para instruções que testam qualificadores vindos do Bloco de Controle e executam condicionalmente uma microinstrução (JN , JZ, JC e JV).

O sinal CE é parte da interface do seqüenciador para permitir que o sinal HOLD possa comandar a duração de ciclos de acesso à memória e entrada e saída externos. Quando se está fazendo acesso ao mapa de memória (CE='1'), se HOLD='1' o ciclo de acesso e estendido pelo número de ciclos em que o sinal fica ativo.

A única observação que resta fazer antes de mostrar a estrutura interna do seqüenciador diz respeito a instruções de salto condicional (JN, JZ, JC e JV). Estas precisam de um tratamento especial. Com base na combinação de conteúdos do IR e dos qualificadores, deve-se computar uma informação binária do tipo *"Salta"* ou *"Não-salta"*. Por exemplo, se o qualificador Z está em 1 e a instrução em execução é JZ, a informação binária deve ser "Salta". Ao todo, existem quatro situações óbvias em que a informação *"Salta"* é gerada e *"Não-salta"* é gerada em todas as outras situações possíveis. A microssimulação da parte operativa para estas instruções deve tornar óbvio como gerenciar o tratamento de saltos condicionais.

4.4.3 Estrutura interna do seqüenciador

A Figura 4.5 detalha a estrutura interna do seqüenciador em nível de blocos. Todos os sinais mostrados são de um bit, exceto quando explicitamente anotado de forma diferente sobre o sinal.

Pode-se dividir o seqüenciador em duas partes principais:

- o registrador de 8 bits comandado pelo relógio, que guarda o endereço da µROM ao qual está sendo feito acesso no momento, denominado na Figura de µPC. Este registrador corresponde ao registrador de estado do modelo FSM. Toda a lógica combinacional do seqüenciador mais as duas colunas N[1:0] da µROM e o decodificador desta correspondem às funções de próximo estado do mesmo modelo O decodificador e o resto da µROM implementa as funções de saída;
- uma lógica combinacional responsável pelo cálculo do próximo endereço da µROM a ser carregado no µPC no próximo período de relógio;

O funcionamento do seqüenciador é simples, pelo menos em princípio. A cada período de relógio, um novo µendereço é calculado e carregado no µPC, indicando a posição da µROM que

contém a próxima microinstrução a ser executada pelo Bloco de Dados.

Seqüenciador do Bloco de Controle da Arquitetura Cleópatra

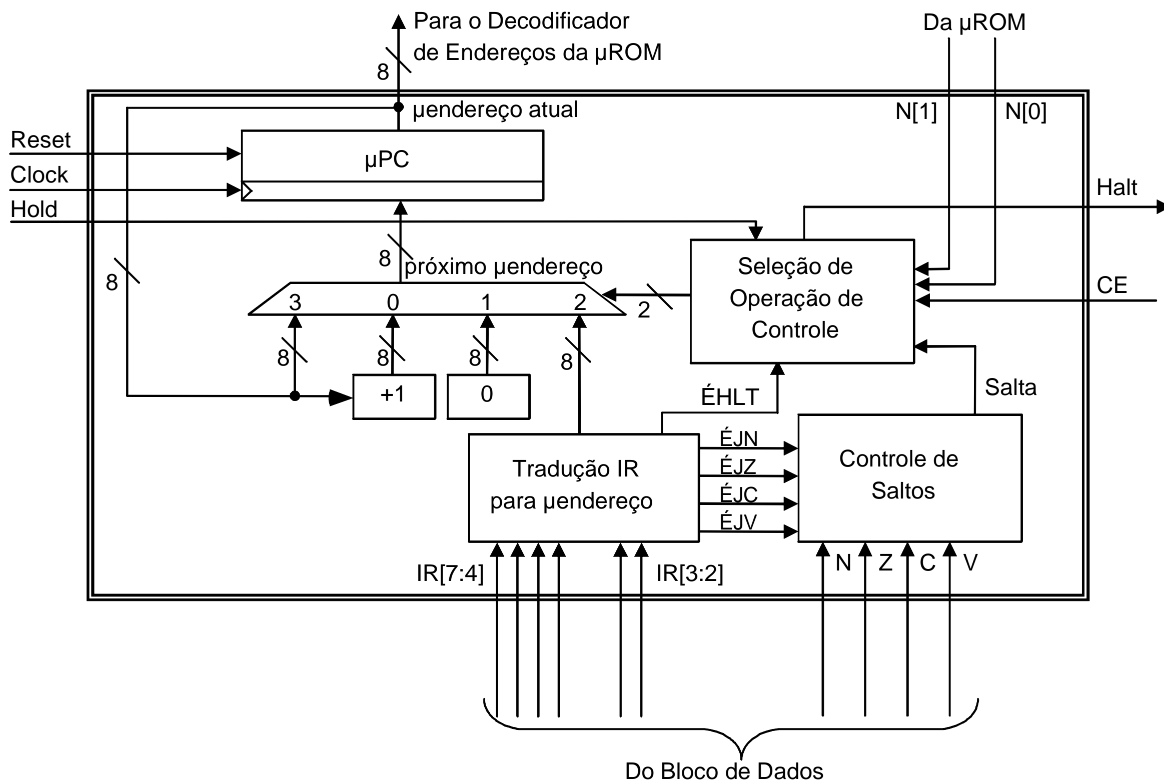


Figura 4.5 - Estrutura Interna do Seqüenciador do Bloco de Controle da Arquitetura Cleópatra.

A única parte um pouco mais complexa é justamente a lógica de cálculo do próximo µendereço. Veja-se agora como se dá esta geração. Inicialmente, percebe-se que há quatro possíveis fontes de próximo µendereço:

- O µendereço atual incrementado de uma unidade, gerado pelo bloco incrementador denominado "+1" na Figura 4.5 (conectado na entrada 0 do multiplexador 4*8:1*8) que recebe como entrada o µendereço atual;
- A constante 0, gerada internamente (cuja saída está conectada à entrada 1 do multiplexador);
- Um novo µendereço, gerado pelo bloco denominado "Tradução IR para µendereço" (cuja saída está conectada à entrada 2 do multiplexador), que recebe como entradas o código da instrução e o modo de endereçamento, contidos atualmente no registrador IR, localizado no Bloco de Dados;
- O µendereço atual, tomado da saída do µPC (conectado na entrada 3 do multiplexador).

A escolha de qual destes quatro µendereços carregar a cada relógio no µPC é realizada pelos dois blocos mais à direita da Figura 4.5, denominados "Seleção de Operação de Controle" e "Controle de Saltos". O funcionamento destes dois blocos é o seguinte. O Bloco "Controle de Saltos" gera um único bit na sua saída, o sinal "Salta" que informa, quando em 1, que a instrução corrente é um salto condicional (JN, JZ, JC ou JV) e que o qualificador correspondente (N, Z, C, ou V, respectivamente) está ativado, indicando que o salto deve ser realizado. Para compreender o

funcionamento do Bloco “Seleção de Operação de Controle”, é necessário detalhar a fase final do ciclo de execução de saltos condicionais. Nestas instruções existe uma microinstrução que deve ser realizada condicionalmente, qual seja, a carga de um novo valor no registrador PC (ou seja, realizar o salto). Esta é sempre a última microinstrução antes de terminar o ciclo de execução. Logo, a penúltima microinstrução desta fase, para saltos condicionais, deve gerar o sinal de controle N[1:0]=11, ou seja, JCOND. Neste caso, se o sinal “Salta” está em 1, o µPC deve ser incrementado, pela seleção da entrada 0 do multiplexador, passando à execução da última microinstrução da fase de execução do salto condicional. Entretanto, se “Salta” está em 0, a última microinstrução não deve ser executada e o ciclo de execução deve terminar aí mesmo, pela seleção da entrada 1 do multiplexador, que zerará o µPC no próximo relógio. Um caso mais especial é criado pela instrução HLT. O sinal ÉHLT, quando ativado, se sobrepõe a todos os sinais de controle (N[0:1], e Salta), gerando a seleção da entrada 3 do multiplexador, carregando o mesmo valor anterior de volta no µPC. O efeito líquido desta atitude é fazer o Bloco de Controle entrar em um “loop” eterno, executando uma microinstrução inócua, que é justamente o que a instrução HLT prevê.

Uma outra observação diz respeito aos sinais ÉHLT, ÉJN, ÉJZ, ÉJC e ÉJV. Cada um destes sinais é gerado de forma muito simples, usando um reconhecedor de código de operação, implementável como uma porta AND de 4 entradas e alguns inversores. Por exemplo, a instrução JZ tem como código de operação 1011, logo o sinal ÉJZ é gerado por uma porta AND de quatro entradas onde estão conectados os sinais IR[7:4] e o sinal IR[6] passa por um inversor antes de entrar na porta AND. Ou seja, apenas quando IR[7:4] for exatamente o vetor 1011, indicando que a instrução é efetivamente JZ (com qualquer modo de endereçamento) o sinal ÉJZ estará em 1. Raciocínio idêntico funciona para HLT, JN, JC e JV, mudando apenas o número e a posição dos inversores.

Se quatro portas lógicas AND são suficientes para gerar estas saídas do Bloco “Tradução IR para µendereço”, o mesmo não vale para a saída de µendereço do bloco. Na realidade, trata-se de um codificador cuja organização depende dos conteúdos da µROM. Apresenta-se agora, via alguns estudos de caso, como se determina o comportamento deste codificador. A Tabela 1 abaixo dá a lista dos µendereços de início de cada combinação instrução/modo de endereçamento (quando este último é pertinente), e deve ser usada para a realização do bloco “Tradução IR para µendereço”, que nada mais será que um conjunto de 8 funções de chaveamento que dependem de 6 entradas[6]. Pode-se empregar o diagrama da seção 3.5 para entender a discussão abaixo.

Já foi dito acima, e verificou-se na microssimulação, que o ciclo de busca toma três microinstruções, que devem ocupar as três primeiras posições da µROM (endereços 0, 1 e 2). Logo é natural que o endereço 3 da µROM contenha o início da fase de execução da primeira das instruções. Seguindo a ordem da especificação da arquitetura, a primeira instrução seria NOT (código binário 000X XX00). Logo, ao receber como entrada um dos vetores 0000 ou 0001 nas linhas IR[7:4], o bloco “Tradução IR para µendereço” deve gerar o µendereço 0000011 (3 em hexadecimal). Todo o ciclo de execução do NOT pode ser realizado em um único período de relógio, ao final do qual o seqüenciador seleciona a constante 0 para carregar no µPC.

Logo, a próxima instrução, STA (Código binário 001X), pode ter sua fase de execução inserida a partir da posição 4. Contudo, STA usa modos de endereçamento. Logo, ela corresponderia a 4 instruções distintas, não a apenas a uma como no caso da instrução NOT. Contudo, STA imediato não faz sentido, ou ela pode ser tratada da mesma forma que STA direto. Logo, existem 3 tipos de STA: direto/imediato, indireto e relativo. Pela microssimulação, já se sabe que STA direto toma 4 períodos de relógio, ocupando assim os endereços 4 a 7 da µROM. Ou seja, os quatro vetores 001X0X nas linhas IR[7:2] devem produzir o µendereço 0000100 na saída do codificador. Para o LDA indireto, o ciclo de execução inicia no endereço 8 e dura 6 períodos de relógio, e assim por diante, conforme o diagrama da Seção 3.5.

[6] Deve estar claro que este fato implica que o número exato de microinstruções necessárias para cada ciclo de execução de cada instrução já foi calculado, ou seja, que os autores deste documento já implementaram a µROM completamente.

Uma última observação sobre os dados da Tabela 4.1 é que os endereços de início das instruções JN, JZ, JC e JV coincidem, para cada modo de endereçamento. Este fato curioso se deve à estrutura particular do seqüenciador, que calcula se um salto condicional deve ser tomado ou não, independente do tipo de salto condicional, simplesmente computando esta informação a partir dos conteúdos do IR e dos qualificadores provenientes do Bloco de Dados. Isto economiza hardware, pois a microROM precisa ter menos linhas distintas.

Tabela 4.1 - Endereços de início das microinstruções dos ciclos de execução de cada instrução na µROM.

Instrução	µendereço de Início	Instrução	µendereço de Início
NOT	03H	JMP # ,D	62H
STA # ,D	04H	JMP ,I	65H
STA ,I	08H	JMP ,R	6AH
STA ,R	0EH	JC # ,D	6DH
LDA #	12H	JC ,I	70H
LDA ,D	15H	JC ,R	75H
LDA ,I	1AH	JN # ,D	6DH
LDA ,R	21H	JN ,I	70H
ADD #	26H	JN ,R	75H
ADD ,D	29H	JZ # ,D	6DH
ADD ,I	2EH	JZ ,I	70H
ADD ,R	35H	JZ ,R	75H
OR #	3AH	JSR # ,D	78H
OR ,D	3DH	JSR ,I	7CH
OR ,I	42H	JSR ,R	82H
OR ,R	49H	RTS	86H
AND #	4EH	JV # ,D	6DH
AND ,D	51H	JV ,I	70H
AND ,I	56H	JV ,R	75H
AND ,R	5DH	HLT	87H