UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM SIMULADOR DE EXECUÇÃO INTERNA DE UMA CPU HIPOTÉTICA

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

MARCELO LUÍS RAMOS

BLUMENAU, JUNHO/2000

2000/1-43

PROTÓTIPO DE UM SIMULADOR DE EXECUÇÃO INTERNA DE UMA CPU HIPOTÉTICA

MARCELO LUÍS RAMOS

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Antônio Carlos Tavares — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Antônio Carlos Tavares

_____

Prof. Maurício Capobianco Lopes

_____

Prof. Miguel A. Wisintainer

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

ACC - Sigla convencionalmente utilizada para um acumulador
AR - Registrador de Endereços
BCD - *Binary Code Decimal* -
CI - Contador de Instrução
CPU - *Central Processing Unit* - Unidade Central de Processamento
MAD - Memória Aritmética Digital
MAL - Memória Aritmética e Lógica
MP - Memória Principal
RI - Registrador de Instrução
UC - Unidade de Controle
ULA - Unidade Aritmética e Lógica
VLSI - *Very Large Scale Integration* – Escala de Integração Muito Grande

RESUMO

Este trabalho tem como objetivo principal é estudar a funcionalidade da unidade de controle (UC) de uma unidade central de processamento (CPU). Para o desenvolvimento do protótipo foi estudado a organização da CPU e suas funções básicas. Para validação do trabalho, construiu-se uma CPU hipotética na qual apresenta-se graficamente a execução das instruções.

ABSTRACT

The main objective of this work is to study the control unit (UC) of a central unit processing (CPU). The organization of CPU and its basic functions were studied for the prototype development. For the validation of this work, a hypothetical CPU was built showing graphically the execution of the instructions.

1 INTRODUÇÃO

Atualmente, a complexidade para análise e compreensão do funcionamento dos circuitos lógicos, obriga ao aluno a fugir da sala de aula para entrar em laboratórios técnicos. Esse é um dos meios encontrados para o melhor aprendizado em disciplinas como Arquitetura de Computadores, entre outras.

A comunicação falada, por melhor que seja, é passível de incompreenssões e ambigüidades, e, deste modo, a comunicação visual, por mais precária que se apresente, é uma ferramenta fundamental no auxílio ao entendimento dos problemas. O ser humano é habituado a aprender através de exemplos práticos que possam demonstrar a resolução dos problemas [DON1991].

Partindo da necessidade de ter-se um ambiente de visualização de uma CPU – unidade central de processamento hipotética, esta foi definida e montada para que algumas de suas funções básicas pudessem ser demonstradas, em substituição aos laboratórios e dando uma melhor forma de aprendizado especificamente para Arquitetura de Computadores.

1.1 MOTIVAÇÃO

A principal motivação para este trabalho foi o desenvolvimento de um protótipo para auxiliar o aprendizado tradicional em disciplinas como Arquitetura de Computadores. Para isso foi utilizada a Linguagem de Programação Delphi 4.0.

1.2 OBJETIVOS

O principal objetivo deste trabalho é o desenvolvimento de um protótipo de execução interna de uma CPU hipotética.

1.3 ORGANIZAÇÃO DO TEXTO

O primeiro capítulo apresenta a motivação para a realização deste trabalho, os seus objetivos e sua organização.

No segundo capítulo são apresentados os conceitos básicos referentes a CPU.

No terceiro capítulo são demonstradas às instruções de máquina e os modos de endereçamento.

O quarto capítulo trata da organização básica de um computador.

O quinto capítulo refere-se ao desenvolvimento do protótipo.

O sexto capítulo é a conclusão deste trabalho.

2 CONCEITOS BÁSICOS

Neste capítulo serão descritos os conceitos básicos referentes a unidade central de processamento ou, em inglês, *central processing unit* (CPU).

2.1 UNIDADE CENTRAL DE PROCESSAMENTO

Será apresentado, a seguir, o componente CPU e sua tarefa no computador.

2.1.1 ELEMENTOS DA CPU E SUAS FUNÇÕES

Em [LAN1974] vê-se que unidade central de processamento é o componente de um sistema de computação que tem por função realizar as operações de processamento nos dados e controlar as atividades dos demais elementos do sistema. Para realizar essas tarefas, pode-se conceitualmente dividir a CPU em duas áreas funcionais distintas: de processamento e de controle.

Na realidade, explica [MAL1985], a CPU é constituída de uma enorme quantidade de pequenos elementos de hardware: registradores de dados, contadores, decodificadores, registradores de endereços, vias de dados, vias de endereços, vias para controle, todos fabricados com tecnologia de semicondutores. Cada um desses elementos possui uma função específica, realizada durante o período de tempo em que uma instrução de máquina é executada e que contribui, em cada caso, para a realização de uma das duas funções básicas já referidas.

Para [MON1996], com a chegada da tecnologia VLSI , os fabricantes de dispositivos semicondutores passaram a construir componentes cada vez mais compactos, menores em tamanho e com muito maior quantidade de elementos e circuitos eletrônicos internos. Com isso, foi possível introduzir em um único *chip* os milhares de elementos necessários ao funcionamento completo de uma CPU.

2.1.1.1 FUNÇÃO DE PROCESSAMENTO

Existem três tipos de operações realizadas pela CPU [MON1996] : operações aritméticas, operações lógicas e teste de sinal de um valor. A função de processamento, é a função responsável pela execução de um dos tipos das operações citadas, ver Figura 1.

Assim, computar ou processar um dado consiste em executar uma operação aritmética ou lógica sobre ele, e essa função é exercida pela área de processamento da CPU.

Para realizar essa tarefa, toda CPU possui um conjunto de elementos de hardware, sendo que o principal elemento é a unidade lógica e aritmética (ULA). A ULA é um aglomerado de circuitos lógicos e componentes eletrônicos que, integrados, realizam operações de soma, subtração, deslocamento, complementação e operações lógicas sobre dados. É o componente responsável pela realização da tarefa fundamental de processar o dado, conforme já mencionado anteriormente. Todos os demais dispositivos de um sistema de computação trabalham para que os dados sejam finalmente levados à ULA e sejam por ela manipulados ([MON1996]).

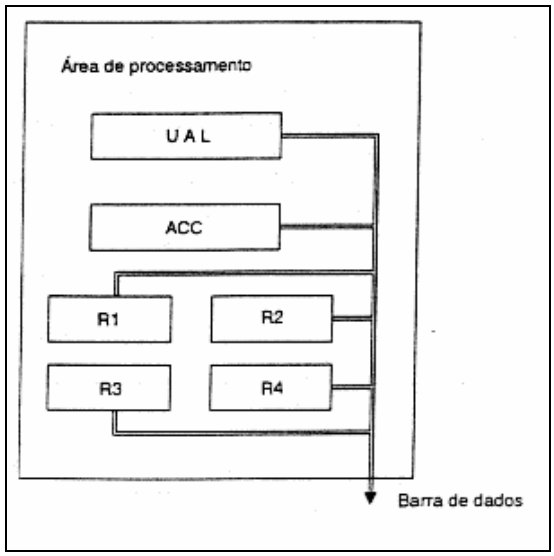
A capacidade de uma CPU, em relação a potência de processamento, é determinada em grande parte, pelas possibilidades de realizar operações embutidas no hardware da ULA. Pequenos e simples processadores têm ULA que pode realizar apenas uma pequena quantidade de operações básicas, como soma e subtração. Operações mais complexas, como multiplicação e divisão são, nesses processadores, realizadas por programas (software), enquanto em processadores mais poderosos, o hardware da ULA é capaz de realizar diretamente as operações de soma, subtração, multiplicação e divisão, com muito mais rapidez do que se fossem realizadas por programa.

Para realizar as operações sobre os dados, a ULA é apoiada por dispositivos de armazenamento, que guardam temporariamente valores a serem transferidos para a ULA ou que recebam o resultado de uma operação aritmética ou lógica que acabou de ser realizada pela ULA. Tratam-se dos registradores, usados em conjunto com a ULA na tarefa de processamento; são o rascunho da ULA ([MON1996]).

Alguns fabricantes denominam esses registradores de acumuladores (ACC), outros chamam apenas de registradores de emprego geral, porque podem receber dados, endereços e índices. Em geral, o tamanho em *bits* dos registradores auxiliares da ULA, bem como, e principalmente, o tamanho dos valores capazes de serem manipulados de uma só vez pela ULA é determinado pelo tamanho em *bits* da palavra.

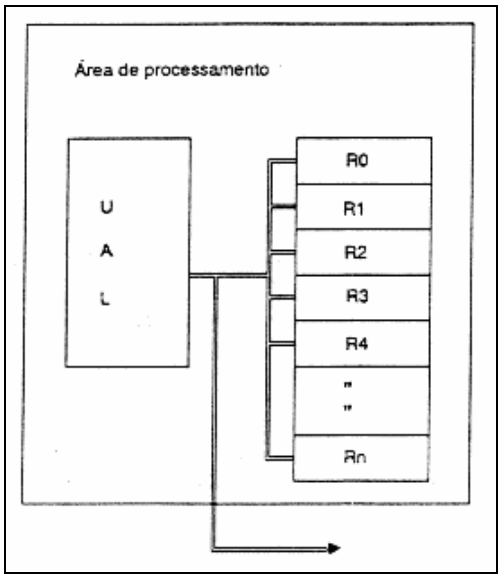
As figuras 1 e 2 mostram esquemas simplificados de uma CPU e suas diferenças de organização interna no que se refere à função de processamento.

Figura 1 - Esquema de uma CPU com emprego de ACC



Fonte: [MON1996]

Figura 2 – Esquema de uma CPU, sem ACC e com emprego de registradores



Fonte: [MON1996]

Inicialmente, pode-se definir a palavra como sendo um conjunto de *bits* que representa uma informação útil. Assim, uma palavra estaria associada ao tipo de interação entre memória principal e CPU, que é individual, informação por informação. Ou seja, a CPU processa

instrução por instrução (cada uma estaria associada a uma palavra), armazena ou recupera número a número (cada um estaria associado a uma palavra), e assim por diante.

Ainda em [MON1996] têm-se que é uma decisão estratégica de projeto definir o tamanho da palavra de uma CPU, pois com essa definição estará sendo estabelecida a sua capacidade de processamento e, de certo modo, a vocação e velocidade do sistema. Se uma CPU for projetada para ter uma palavra de 8 *bits*, significa que:

a) o ACC e demais registradores de dados da CPU terão capacidade para armazenar valores de 8 *bits*;
b) a ULA somente poderá efetuar operações aritméticas e lógicas com valores representados em 8 *bits*;
c) as vias de dados entre a memória principal (MP) e a CPU e internas na CPU terão, em geral, capacidade para transferir valores com 8 *bits*.

Desse modo, se em um programa for necessário somar dois números com, digamos, 16 *bits*, essa operação será realizada em duas etapas: inicialmente, serão adicionadas a 1ª metade de cada número (8 *bits*) e depois a parte restante. Os pequenos e antigos microcomputadores TK-3000, MSX, Cobra-305, Motorola 6800, etc. efetuavam operações aritméticas desse modo ([MON1996]).

Em outros sistemas onde, por exemplo, a palavra é definida com 16 *bits*, têm-se:

a) o ACC, demais registradores e vias de dados permitem o armazenamento e transferência de números de 16 *bits*;
b) a ULA é capaz de efetuar operações aritméticas e lógicas entre valores com 16 *bits*.

Nesses sistemas, a soma de dois números de 16 *bits*, mencionadas no exemplo anterior, será realizada em uma única etapa, em vez de duas etapas, como no caso dos processadores de palavra de 8 *bits*, e é evidente que será realizada em um tempo bem menor. O sistema com palavra de 16 *bits* possui capacidade de processamento (e velocidade) maior do que o sistema de palavra de 8 *bits*.

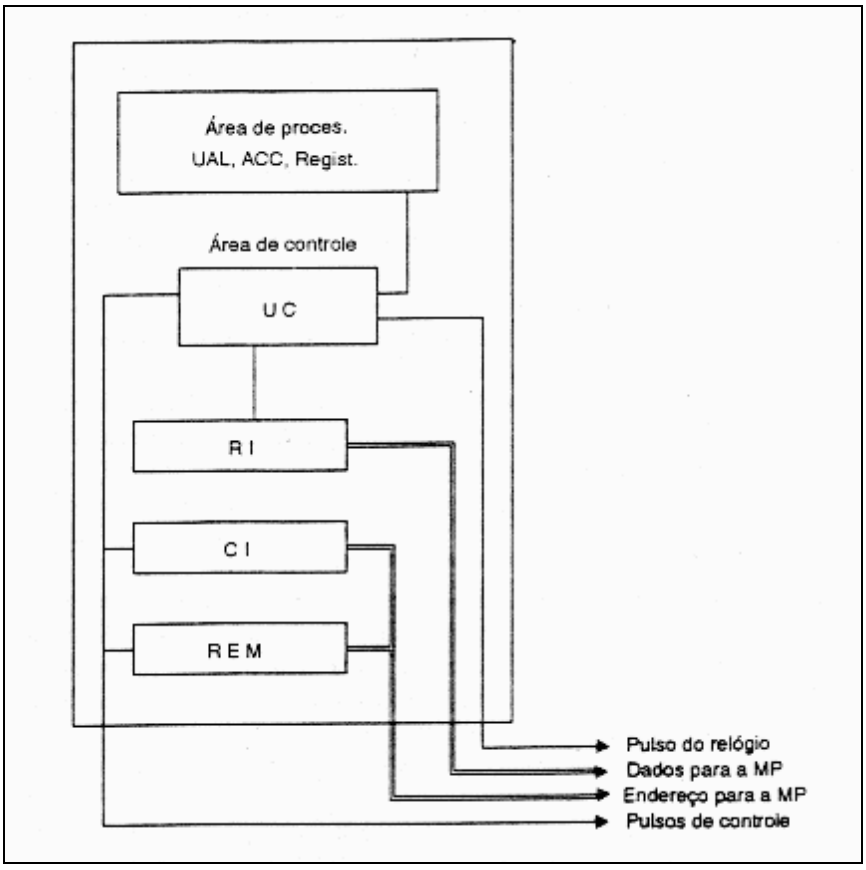
2.1.1.2 FUNÇÃO DE CONTROLE

A Função de Controle da CPU é responsável pelo funcionamento sincronizado de todos os dispositivos de um sistema de computação, a fim de que eles contribuam

adequadamente para a execução de uma instrução de máquina ([MON1996]). Fazem parte da função de controle, entre outros:

a) unidade de controle – UC;
b) contador de instrução – CI;
c) registrador de instrução – RI.

Figura 3 – Elementos principais da área de controle de uma CPU



Fonte: [MON1996]

A UC é constituída de um conjunto de pequenos dispositivos de hardware: decodificadores, contadores, vias de controle, portas lógicas, etc., tendo como elemento principal o relógio (*clock*), responsável pela manutenção do sincronismo de eventos e da correta seqüência de ações para execução de uma operação na CPU. As tarefas básicas da UC são:

a) buscar e transferir para a CPU (armazena no RI) a instrução de máquina a ser executada;

b) interpretar a instrução de modo a identificar qual a operação a ser realizada pela ULA;
c) emitir os sinais de controle e de sincronismo necessários (na seqüência requerida) à execução da operação que acabou de ser interpretada.

Um dos pontos fundamentais do projeto e construção de um sistema de computação digital consiste na correta programação da seqüência de operações necessárias à busca e execução das instruções de máquina pertencentes a um determinado programa. Essa seqüência é efetivada pela UC através da emissão, no tempo determinado, dos sinais de controle e sincronização que iniciam e terminam nas diversas etapas necessárias à realização de um ciclo de instrução ([MON1996]).

Desse modo, a UC é responsável pela atividade adequada do sistema, integrando seus diversos componentes de hardware, enquanto a ULA, efetivamente, executa as operações (sob o comando da UC). Para realizar suas tarefas a UC utiliza os dois registradores já citados: o contador de instrução (CI) e registrador de instrução (RI) ([MON1996]).

O CI tem por função armazenar o endereço da próxima instrução a ser recuperada da MP (ciclo de leitura) para interpretação e execução. O RI tem por função armazenar o código da instrução a ser executada (os *bits* que indicam a operação a ser realizada), que será descrito mais adiante ([MON1996]).

3 INSTRUÇÕES DE MÁQUINA

A seguir será discutida a instrução de máquina e o que nela está envolvido.

3.1 O QUE É UMA INSTRUÇÃO DE MÁQUINA

Para [ZUF1978] a instrução consiste em uma ordem codificada (código de operação), para a UC executar uma operação qualquer sobre dados. No contexto da interpretação de uma instrução, o dado pode ser um valor numérico, um caractere alfabético, um endereço (instrução de desvio).

A operacionalidade de um computador, construído segundo os princípios enunciados pelo Prof. John Von Newmann, em 1945, caracteriza-se por:

a) armazenamento prévio do programa, na memória principal, uma instrução seqüencialmente a outra;
b) execução das instruções, uma de cada vez, pela CPU, buscando (leitura) da memória a próxima instrução a ser executada, bem como os dados que serão manipulados por aquela específica instrução;
c) possibilidade de alteração da seqüência de execução das instruções através de uma instrução apropriada, que determina o desvio da seqüência (a próxima instrução a ser executada não será a que está armazenada na célula imediatamente seguinte);
d) o CI, contendo o endereço da instrução a ser executada em seguida, é o responsável pela manutenção ou alteração da seqüencialidade da execução do programa.

3.2 MODOS DE ENDEREÇAMENTO

Algumas conclusões em relação ao item anterior podem ser efetuadas. O endereçamento de uma instrução (macro) é sempre realizado através do valor armazenado do contador de instrução (CI). Todo ciclo de instrução é iniciado pela transferência da instrução para o RI (usando-se o endereço contido no CI).

A localização do(s) dado(s) pode estar explicitamente indicada na própria instrução, por um ou mais conjuntos de *bits*, denominados campo do operando ou implicitamente (quando o dado está armazenado no acumulador, que não precisa ser endereçado por ser único na unidade central de processamento - CPU).

A existência de vários métodos para localizar um dado que está sendo referenciado em uma instrução se prende à necessidade de dotar os sistemas de computação da necessária flexibilidade no modo de atender diferentes requisitos dos programas.

A seguir serão destacados alguns modos de endereçamento Todos os conceitos a seguir foram vistos em [ZUF1978].

3.2.1 MODO IMEDIATO

O método mais simples e rápido de obter um dado é indicar seu próprio valor no campo operando da instrução, em vez de buscá-lo na memória. A vantagem desse método reside no curto tempo de execução da instrução, pois não gasta ciclo de memória para sua execução, exceto o único requerido para a busca da instrução.

Assim, o dado é transferido da memória juntamente com a instrução (para o RI), visto estar contido no campo operando da instrução. Esse modo, é útil para inicialização de contadores, na operação com constantes matemáticas, para armazenamento de ponteiros em registradores da CPU, etc.

Uma de suas desvantagens reside na limitação do tamanho do campo operando das instruções, o que reduz o valor máximo do dado a ser manipulado. Outra desvantagem é o fato de que, em programas repetidamente executados, com valores de variáveis diferentes a cada execução, esse método acarretaria o trabalho de alteração do valor do campo operando a cada execução.

3.2.2 MODO DIRETO

Nesse método, o valor binário contido no campo operando da instrução indica o endereço de memória onde se localiza o dado.

O endereço pode ser o de uma célula onde o dado está inteiramente contido ou pode indicar o endereço da célula inicial, quando o dado está armazenado em múltiplas células. É também um modo simples de acesso, pois requer apenas uma referência à memória principal para buscar o dado (é porém mais lento que o modo imediato).

3.2.3 MODO INDIRETO

Nesse método, o valor binário do campo operando representa o endereço de uma célula; mas o conteúdo da mesma não é o valor de um dado; é um outro endereço de memória, cujo conteúdo é o valor do dado. Assim, há um duplo endereçamento para o acesso a um dado e, consequentemente, mais ciclos de memória para buscar o dado, comparativamente com os métodos já apresentados.

O endereço intermediário (conteúdo da célula endereçada pelo valor do campo operando) é conhecido como ponteiro, pois indica a localização do dado. Este conceito é largamente empregado em programação.

3.2.4 ENDEREÇAMENTO POR REGISTRADOR

Esse método tem característica semelhante aos modos direto e indireto, exceto que a célula (ou palavra) de memória referenciada na instrução é substituída por um dos registradores da CPU. Com isso, o endereço mencionado na instrução passa a ser o de um dos registradores, e não mais de uma célula da memória principal.

A primeira vantagem, logo observada, consiste no menor número de *bits* necessários para endereçar os registradores, visto que estes existem em muito menor quantidade que as células de memória. Outra, está no próprio emprego do dado, que passa a ser armazenado em um meio (registrador) cujo acesso é muito mais rápido que o acesso à memória.

3.2.5 MODO INDEXADO

Nesse tipo de instrução, o endereço do dado é a soma do valor do campo operando (fixo para todos os elementos de um determinado *array*) e de um valor armazenado em um dos registradores da CPU (normalmente denominado registrador índice). O valor armazenado nesse registrador varia para o acesso a cada elemento (“aponta” para o elemento desejado).

3.2.6 MODO BASE MAIS DESLOCAMENTO

Esse método consiste na utilização de dois campos na instrução (que substituem o campo operando): um com o endereço de um registrador chamado de base, e o outro, com um valor denominado deslocamento, porque contém um valor relativo à primeira instrução.

4 ORGANIZAÇÃO BÁSICA DE UM COMPUTADOR

A enorme variedade de circuitos integrados atualmente existente permite delinear um número muito grande de organizações internas de processadores, dependendo da aplicação específica a que o processador se destina e da eficiência desejada na operação do processador em si ([ZUF1978]).

A necessidade de padronização do equipamento produzido impõe, todavia, limites à liberdade de organização interna e à eficiência de operação, na tentativa de otimização dos custos finais. Com isso, surgem os processadores de uso geral, cuja organização interna sofre relativamente poucas variações e cujo sistema programacional segue sempre as mesmas linhas ([MAL1985]).

Os dados e as instruções de um computador são armazenados em sua memória. Existem diversos níveis de memória. Normalmente as memórias são mais lentas que os demais elementos da Unidade Central de Processamento (CPU). Para reduzir essa disparidade de velocidades utilizam-se de memórias de alta velocidade como elementos de acoplamento na interação direta com a Unidade Lógica e Aritmética (ULA) e Unidade Central (UC). Problemas de custo, contudo, restringem a utilização dessas memórias rápidas. Por isso têm-se um segundo nível de memória de menor velocidade, e de menor custo, que completa o sistema de armazenamento primário do computador. Um terceiro nível de memória é constituído pelo armazenamento secundário, como por exemplo os discos flexíveis, discos rígidos, etc. O emprego de cada um desses tipos depende das finalidades e do tamanho do centro de processamento de dados onde o computador se encontra instalado ([ZUF1978]).

A ULA executa operações aritméticas e lógicas. As operações aritméticas normalmente realizadas são as operações de soma e subtração. A integração em larga escala facilita, todavia, a incorporação de outras operações aritméticas como a multiplicação e a divisão. Microprocessadores monolíticos possuem unidas em uma única pastilha, a ULA e a UC, além dos registradores de trabalhos diversos, cuja finalidade é guardar resultados intermediários, aumentando significativamente a eficiência de operação do sistema. Nos microprocessadores por segmentação de *bits*, a ULA é dividida em pares modulares que, associadas entre si, permitem a construção de um computador de grande porte ([MAL1985]).

A UC retira ordenadamente e interpreta as instruções da memória, executando-as numa segunda fase. Existem dois tipos de projetos de unidades de controle: lógica fixa e controle microprogramado. Na habilidade de a UC executar diferentes instruções provenientes da memória surge a versatilidade do computador, o qual é capaz de executar diferentes seqüências de cálculos ([ZUF1978]).

4.1 DETALHES OPERACIONAIS DE UM PROCESSADOR

Algoritmo é uma lista ou seqüência de procedimentos que, executados em ordem sobre determinados conjuntos de dados, conduz a uma resposta, qualquer que seja o conjunto de dados, desde que esse conjunto pertença ao campo de definição da atuação do algoritmo. Considerando, por exemplo, algoritmos que operam no corpo dos números reais, deve-se fornecer a esses algoritmos dados que sejam números reais ([ZUF1978]).

Fornecido determinado processador de dados, existe um repertório de instruções desse processador definida pelo fabricante da máquina. Se essas instruções estão na forma de um cordão de *bits* que pode ser diretamente interpretado pela máquina, diz-se que há uma instrução em linguagem de máquina ou simplesmente uma instrução de máquina. Quando um algoritmo é colocado em termos de uma seqüência de instruções, diz-se que há um programa. Se as instruções desse programa são instruções de máquina, tem-se um programa em linguagem de máquina. Evidentemente essa forma não é a mais conveniente para o usuário. Definem-se então, termos mnemônicos que são associados às instruções. Quando as instruções são representadas pelo código mnemônico, diz-se que elas se encontram em linguagem de montagem (*assembly language*) ([MAL1985]).

Ao usuário é mais fácil escrever programas em linguagem de montagem do que em linguagem de máquina. Existe, todavia, o problema de passar esse programa de linguagem de montagem para a linguagem de máquina. Essa transformação pode ser realizada automaticamente por um programa montador (assembler). O programa montador opera sobre o programa do usuário em linguagem de montagem, agora denominado de programa fonte, e o transforma num programa objeto que, nesse caso, está em linguagem de máquina. O montador é um programa tradutor ou metaprograma. Um programa tradutor traduz um programa fonte escrito numa linguagem de alto nível para uma linguagem de menor nível ([MON1996]).

Os programas que realizam a tradução dos programas em linguagem de alto nível para linguagens de montagem ou de máquina recebem o nome de compilador ou interpretador, dependendo da maneira que é realizada a tradução. Um compilador processa totalmente o programa fonte traduzindo-o para linguagem de máquina. Só então é executado esse programa (que se transformou em programa objeto).

Já um programa interpretador toma, uma a uma, as instruções do programa fonte, obtém a instrução ou instruções do programa objeto, executa essas instruções e só então se ocupa de nova instrução do programa fonte. A execução das sucessivas instruções de um programa em linguagem de máquina armazenado na memória principal de um processador é comandada pela UC. Essa unidade retira uma a uma as instruções da memória principal, obedecendo a uma ordem estabelecida pelo próprio programa que está sendo executado ([MON1996]).

Posto isso, procura-se estabelecer algumas definições, habitualmente válidas para a maior parte dos processadores existentes. Essas definições relacionam-se à forma de execução das instruções principalmente quanto aos problemas associados à temporização. Diz-se habitualmente porque, a rigor, são possíveis muitas variações em torno desse assunto, de forma que a seqüência básica de execução das instruções aqui estabelecida pode eventualmente se tornar irreconhecível ([MON1996]).

Uma instrução em sua execução é decomposta em uma seqüência de operações muito elementares, que são executadas em um número inteiro de ciclos de relógio (normalmente um). Essas operações elementares que não podem, por isso, ser decompostas em operações mais simples recebem o nome de microoperações. A denominação de tempo de ciclo de instrução ou simplesmente ciclo de instrução é o intervalo de tempo necessário para que a máquina execute essa instrução. Numa dada máquina, o ciclo de instrução depende do tipo particular da instrução que está sendo executada. Existem instruções que tem inclusive seu tempo de execução variável, dependendo de parâmetros do programa em execução ([ZUF1978]).

Na grande maioria dos processadores hoje em operação, a UC divide o ciclo de instrução em duas partes distintas: o ciclo de busca e o ciclo de execução. O ciclo de busca é o tempo consumido pela UC para identificar a instrução que deve ser executada, separá-la das

demais instruções presentes na memória principal e transportá-la para um Registrador de Instrução (RI). Os *bits* do RI são aplicados na UC, fornecendo a essa unidade elementos que irão determinar o comportamento da UC durante o ciclo de execução. Habitualmente o ciclo de busca é idêntico para todas as instruções e deve ocorrer independentemente do programa particular em execução. Por isso é conveniente encarar o ciclo de busca como uma operação "vegetativa" da máquina, que, uma vez iniciado, tem prosseguimento independentemente do conteúdo da memória principal ([ZUF1978]).

A definição do ciclo básico de máquina, ou simplesmente ciclo de máquina, é o intervalo de tempo necessário para a máquina produzir um acesso a determinada localidade da memória principal, obter a informação desejada e, se necessário, no caso de leitura destrutiva, restaurar a informação original no endereço acessado. O ciclo de máquina está intimamente ligado ao ciclo de memória ou tempo de ciclo de memória, que é o menor intervalo de tempo entre duas leituras consecutivas na memória. Dependendo das microoperações auxiliares a serem realizadas no acesso, o ciclo de máquina pode ser de 20 a 40% maior que o tempo de ciclo de memória. Se uma instrução, composta de *n* palavras, o ciclo de busca exige *n* ciclos de máquina para ser completado ([LAN1974]).

Costuma-se admitir o ciclo de máquina como sendo um múltiplo inteiro de ciclos de relógio como 2, 4, 8, 12, 16, determinando, cada ciclo de relógio, um segmento dentro do ciclo de máquina. As coisas, porém, complicam-se um pouco quando não tem-se a necessidade de realizar acesso à memória, como ocorre muitas vezes no ciclo de execução. Nesse caso, pode não haver necessidade de ciclo completo de máquina. Executam-se as operações em determinado número de ciclos de relógio e passamos à etapa seguinte. Dizemos então, que executamos uma fase da instrução. O tempo máximo de duração da execução de uma fase é o ciclo de máquina, e o tempo mínimo é um ciclo de relógio ([LAN1974]).

Concluído o ciclo de busca da instrução, o próximo passo é o ciclo de execução. Novamente dependendo da complexidade da instrução, o ciclo de execução pode durar uma ou várias fases. Por exemplo, uma instrução que opera com os dados em dupla precisão e que deve obter esses dados na memória principal, requer, para seu ciclo de execução, ao menos quatro ciclos de máquina ([LAN1974]).

Embora exista alternância entre ciclo de busca e ciclo de execução na grande maioria dos processadores em operação, essa condição não é absolutamente necessária, sendo possível retirar da memória, de uma vez, grupos de instruções, para só então passar a executá-las.

4.2 PROCESSADORES SERIAIS E PARALELOS

Denominam-se de palavra de máquina a um conjunto de *bits*, normalmente composto por um número fixo de dígitos, que serve de unidade básica para a representação de dados e para a realização das operações lógicas e aritméticas. Em [MON1996], temos como exemplo, um microprocessador de 8 *bits* tem seu duto de dados composto de 8 linhas em paralelo, sua ULA composta de 8 *bits* e seus dados armazenados na memória também em quantidade de 8 *bits*.

Todas as operações em um processador são formadas por seqüências temporais de execução mais simples, de maneira que todos os processadores, a rigor, operam serialmente. Por que existem, então, processadores ditos paralelos ou seriais ? Essa distinção baseia-se, principalmente, no modo pelo qual são processados os dígitos (não necessariamente binários) que compõem a palavra de máquina. Se os dígitos de uma palavra de máquina são processados serialmente um após o outro, trata-se de um processador serial, operando na base numérica em que estão representados os dígitos. Assim, uma máquina com dígitos decimais, representados em um código *BCD*, operando serialmente é um processador serial decimal ([ZUF1978]).

Agora, um processador que opere todos os dígitos de palavra de máquina simultaneamente é um processador paralelo, apesar de realizar operações seqüenciais no tempo. De modo geral, devemos chamar a atenção que, no cômputo velocidade de operação por paralelismo, os processadores seriais são potencialmente mais eficientes que os processadores paralelos, devido a presença dos atrasos imanentes ([ZUF1978]).

4.3 UNIDADE DE CONTROLE

A Unidade de Controle, UC, de um processador deve gerar os valores corretos das variáveis de controle, no instante correto, de forma que o fluxo de dados previsto em cada instrução seja executado de acordo com o previsto. Dependendo do processador, a unidade de

controle pode estar concentrada em uma região do processador ou distribuída por todo o sistema. É possível uma variação contínua de grau de concentração e de distribuição ([MON1996]).

Não é necessário que um sistema possua, também, uma única unidade de controle. Em sistemas de multiprocessamento, podem haver muitos processadores independentes operando cooperativamente. Aqui existe, também, quase uma graduação contínua entre múltiplos processadores totalmente independentes, apenas tendo fixado um protocolo de intercomunicações entre si, múltiplos processadores operando com graus de hierarquia, um processador mestre e processadores escravo até o ponto onde não se consegue distinguir a situação resultante da situação existente num sistema com uma única UC distribuída.

Com o desenvolvimento dos microprocessadores, existe atualmente grande esforço do desenvolvimento de sistemas de grande porte com múltiplas CPUs de microprocessadores, sendo que a principal dificuldade reside na criação de um sistema programacional (*software system*) adequado. De acordo com a filosofia de projeto, costuma-se classificar as unidades de controle em duas categorias: de lógica fixa e de controle microprogramado ([ZUF1978]).

Numa UC do tipo lógica fixa, na síntese das variáveis de controle que comandam as operações do processador, adotam-se os métodos clássicos de síntese e minimização de funções booleanas. Tendo em conta que no estabelecimento dos valores das variáveis de controle de uma UC existem muitas condições irrelevantes, essa etapa é de muita importância na redução de componentes da UC. Uma dificuldade da lógica fixa é que esse não permite facilmente modificações no repertório de instruções após a implementação das variáveis de controle. De fato, modificações implicariam em ressintetizar essas variáveis, o que pode implicar em modificações profundas na implementação.

Numa unidade de controle microprogramada realiza-se simplesmente a tabulação dos valores que devem ser assumidos pelas variáveis de controle e utiliza-se um subsistema de Memória Aritmética e Lógica (MAL) para implementação dessas variáveis. O acréscimo de novas instruções no repertório já existente implica tão somente em se ampliar a memória MAL. O nome microprogramação advém do fato de que cada instrução pode ser descrita por um programa (microprograma), podendo-se, em sua implementação, utilizar os mesmos recursos que existem na programação normal.

A execução da operação ou das operações indicadas por uma instrução é decomposta em um conjunto de operações mais elementares denominadas de microoperações. Habitualmente as microoperações não podem ser subdivididas em partes menores. Como microoperações pode-se entender deslocamentos em registradores, acesso à memória, transporte de palavras de dados de um ponto a outro, etc.

4.3.1 CONTROLE MICROPROGRAMADO

A microprogramação, de acordo com seu idealizador Wilkes, é um processo sistemático de projetar um sistema de computação genérico. Sabe-se que uma instrução de máquina é uma configuração de *bits*, habitualmente constante, interpretada e executada pela UC como uma seqüência de subconjuntos de microoperações microconcorrentes no tempo. É usual o termo microconcorrentes para indicar microoperações que são executadas paralelamente, considerando-se a escala de tempo de operação do relógio. Uma instrução de máquina é o nível mais elementar de instrução numa UC com lógica fixa. Todavia, num controle microprogramado, uma instrução é decomposta em uma cadeia de microinstruções mais elementares ainda. Daí a instrução de máquina passar a ser denominada de macroinstrução ([ZUF1978]).

A microprogramação considera cada instrução de máquina, ou seja, cada macroinstrução, como sendo o resultado de um programa composto de microinstruções, sendo por isso, chamado de microprograma. Como conseqüência disso, torna-se possível utilizar todos os processos de programação no projeto da UC microprogramada. É denominado de microciclo ao intervalo de tempo gasto na busca e execução de uma microinstrução ([LAN1974]).

Considere, inicialmente, um relógio monofásico. Após o ciclo de busca, o código de macroinstrução é enviado ao registrador de códigos de macroinstruções. O conteúdo desse registrador permanece constante durante toda a execução do microprograma. O endereço da próxima microinstrução é fornecido pela própria memória de controle. A ordem de execução das microinstruções no microprograma pode ser mudada acionando-se o multiplex com uma variável de controle externa. Entre as variáveis de controle existe uma que indica o término do microprograma e o início do ciclo de busca de nova macroinstrução ([ZUF1978]).

São classificados dois tipos de microprogramação: não codificada e codificada. Na microprogramação não codificada, cada linha comanda diretamente uma porta, uma injeção em registrador, enfim um elemento de circuito que irá permitir a realização de microoperações. O próximo código de microinstrução é determinado *bit* a *bit* pela memória de controle. Uma vantagem de microprogramação não codificada é a grande versatilidade, pois pode-se controlar independentemente todos os recursos do processador no nível de porta e assim ter possibilidade de estabelecer o maior número possível de microconcorrências, utilizando ao máximo os recursos do microprocessador. Existindo elevado grau de microconcorrência, têm-se maior rapidez na execução dos microprogramas, já que esses terão um pequeno número de passos ([ZUF1978]).

Caso interessante é o limite onde o número de passos é igual a um e a microinstrução coincide com a macroinstrução. Aqui novamente pode se observar o problema do compromisso entre paralelismo e tempo de execução. A maior dificuldade da microprogramação não codificada é exatamente o elevado número de variáveis de controle.

Como no controle não codificado existem recursos que obrigatoriamente devem ser disjuntivos no tempo, é conveniente associar combinações de variáveis a esses recursos. Dessa forma, obtêm-se uma redução do número de variáveis de controle e é evitada a possibilidade de operações com erro resultante do acionamento de duas variáveis de controle que não podem operar simultaneamente. A rigor, num controle codificado, deve-se, uma vez estabelecido o repertório de macroinstruções, levantar quais os conjuntos de microoperações em que nunca ocorre microconcorrência, e neles estabelecer um grupo de variáveis de controle, de forma que, a cada combinação das variáveis de controle, corresponda um elemento do conjunto de microoperações disjuntivas. Embora seja aparentemente difícil a escolha ótima de tais conjuntos, a fixação desses conjuntos normalmente é feita por integrados, utilizados nos diversos pontos do processador. No controle codificado, têm-se portanto, as variáveis de controle associadas em pequenos grupos que irão selecionar uma dentre um conjunto de microoperações disjuntivas no tempo ([ZUF1978]).

Além da microprogramação codificada e não codificada, existe a microprogramação horizontal e vertical. Primeiramente, algumas características da microprogramação horizontal:

a) comprimentos da palavra de microinstrução situados entre 40-120 *bits*, sendo típicos valores entre 70 e 80 variáveis;

b) a maior parte das combinações das variáveis de controle não é utilizada em virtude do elevadíssimo número de combinações possíveis;
c) pouca ou nenhuma classificação de microinstruções dentro de funções, não existindo um repertório básico de microinstruções – habitualmente as microinstruções podem ser criadas de acordo com a necessidade;
d) os campos dentro da palavra de microinstrução contém poucos *bits*, produzindo um nível de codificação relativamente baixo;
e) elevado grau de microconcorrência com pouca necessidade de operação serial;
f) tempo de execução das microinstruções bastante pequeno.

Existem vários modos de melhorar a eficiência de utilização da memória de controle. Por exemplo, em virtude do número variável de microinstruções numa macroinstrução, fixa-se por macroinstrução um número máximo de microinstruções. Não utilizando esse número no máximo, serão desperdiçadas as combinações de *bits* de endereçamento. São possíveis duas formas para contornar o problema, dadas a seguir ([LAN1974]).

Uma primeira forma consiste em admitir um espaço de endereçamento contínuo de forma que o código de macroinstruções represente o primeiro endereço de microinstruções. Terminado o microprograma, o próximo código na ordem crescente representa o novo código de macroinstruções, não havendo distinção entre o campo de endereçamento de microinstruções e de macroinstruções ([LAN1974]).

Nesse esquema aumenta-se continuamente de uma unidade o código da microinstrução presente, tendo em conta que existem, no microprograma, seqüências de microinstruções que podem estar numa ordem crescente de endereços. Com isso, reduz-se o número de variáveis de controle. A execução do microprograma inicia-se com a predisposição do registrador de código de microinstruções, com o código de macroinstruções. Para passar a próxima microinstrução, soma-se um ao conteúdo desse registrador. No salto condicional, realiza-se, através do multiplex, nova predisposição do registrador do código de microinstruções. Esse arranjo tem a vantagem de permitir uma passagem relativamente fácil a qualquer ponto do subsistema MAL, mesmo que esse ponto pertença a outra macroinstrução. Todavia, tem-se uma desvantagem na necessidade de um ligeiro aumento do número de *bits* de codificação das macroinstruções, já que nem todos os códigos serão utilizados para caracterizar diferentes macroinstruções ([ZUF1978]).

Existe uma segunda alternativa, segundo [ZUF1978] de implementar unidades microprogramadas, porém com maiores limitações operacionais que no caso anterior. Nesse caso, é mantida a separação dos códigos de macro e microinstruções, e acrescenta-se um codificador prévio que reduz o número de *bits*. Com isso não é necessário desperdiçar *bits* da macroinstrução. Todavia a síntese do codificador prévio depende do repertório e do tipo de macroinstruções implementados, eliminando a principal vantagem de um sistema microprogramado, que permite a inserção de novas macroinstruções, sem que as macroinstruções já existentes fossem de alguma forma afetadas.

Implicitamente foi admitido, até aqui, sistemas com uma única fase de relógio, de forma que cada microinstrução fosse executada num ciclo de relógio. Sistemas desse tipo são denominados de monofásicos. Entretanto é possível pensar em certa seqüencialização dentro de uma microinstrução, de modo que algumas das microoperações especificadas são realizadas após as outras. Denomina-se de unidades de controle microprogramadas polifásicas quando existirem microinstruções que possam durar dois ou mais ciclos de relógio monofásico, ou então, quando houverem múltiplas fases de relógio.

Evidentemente a insistência em sistemas monofásicos, trará um determinado número *n* de microinstruções num programa, pois obrigatoriamente algumas microoperações devem preceder outras, independentemente do paralelismo existente. Esse número *n* poderá ser consideravelmente reduzido; do contrário, as microinstruções com duração de múltiplos ciclos ou múltiplas fases de relógio, tendo em vista que muitas microoperações simples, que devem ser realizadas obrigatoriamente em seqüência, podem ser incorporadas a uma única microinstrução. Com isso, o número inicial *n* de microinstruções, num microprograma de um sistema com UC microprogramada monofásica, pode ser substancialmente reduzido e a velocidade de execução da macroinstrução aumentada em virtude da redução do número de acessos na memória principal. Têm-se, evidentemente, um limite superior do tamanho de cadeia de microoperações dentro de uma microinstrução, caso contrário, numa situação limite, a microinstrução passaria a coincidir com a macroinstrução e o resultado seria uma UC mais próxima da lógica fixa que do controle microprogramado. A cada sistema particular existe um ponto ótimo de incorporação de microoperações executadas em série, dentro de uma microinstrução.

Existe grande esforço no sentido de otimização do tamanho ideal para as microinstruções, de forma a maximizar as microconcorrências dentro do sistema. Fixadas as microinstruções, o esforço passa a se concentrar na otimização do microprograma em si. Como na microprogramação horizontal tanto a natureza das microinstruções como a natureza dos microprogramas dependem diretamente da organização interna do processador em projeto, não existe unanimidade em torno do assunto, sendo as soluções mais eficientes as heurísticas.

Em contraposição à microprogramação horizontal, dentro do esforço de melhor aproveitamento da memória, foi desenvolvido o conceito de microprogramação vertical. Nesse tipo de microprogramação é definido um repertório de microinstruções com um número máximo de elementos 2^m, sendo cada instrução representada por uma combinação das *m* variáveis. Os microprogramas são implementados dentro do subsistema MAL de controle, com as microinstruções codificadas. Na saída do subsistema MAL têm-se um subsistema que é encarregado de realizar a decodificação das microoperações. Observe-se que há uma importante perda de liberdade na fixação de um repertório máximo de microinstruções que pode redundar em aumento de tamanho dos microprogramas. Porém, sendo fixado um repertório básico de microinstruções, é fácil a construção de programas geradores de microprogramas. O multiplex permite a predisposição do registrador de códigos de micro e macroinstruções e a realização de saltos condicionais dentro do microprograma ([ZUF1978).

Aqui, também admite-se implicitamente um sistema microprogramado monofásico. A utilização de sistemas polifásicos, com microinstruções mais e mais complexas, aproximando-se da macroinstrução, faz com que a UC microprogramada, também nesse caso, aproxime-se da UC de lógica fixa. Considera-se, para um melhor entendimento desse fato um sistema polifásico onde as microinstruções sejam compostas de múltiplos ciclos de um relógio monofásico.

Para maior eficiência do sistema de microprogramação vertical, a utilização de um sistema polifásico é uma necessidade, pois caso contrário, o resultado seria um número muito grande de microinstruções no microprograma. Aqui pode se observar também, que a microprogramação vertical pode ser considerada como o caso limite do controle codificado horizontal, onde todas as variáveis de controle estão codificadas no microprograma. A tendência de introduzir maior complicação nas microinstruções levou alguns autores a

definirem a nanoprogramação como sendo o nível mais elementar da microprogramação, como fora originalmente proposto por Wilkes. A rigor, não existe necessidade de tal codificação das variáveis de controle para que um esquema de microprogramação seja considerado como vertical, bastando apenas ser utilizada uma codificação em nível elevado ([ZUF1978]).

As principais características da microprogramação vertical são:

a) palavra de instrução de 16 a 20 *bits*;
b) microinstruções suficientemente complexas, de modo que são necessários vários ciclos de relógio monofásico para executá-la;
c) campos de palavras de instrução altamente codificados;
d) campos longos na execução de microinstruções;
e) utilização da maior parte das combinações dos *bits* de microinstrução;
f) definição de um repertório básico de microinstruções;
g) perda de contato direto com os recursos de circuitos.

Em uma unidade de controle serial microprogramada, é freqüente a necessidade de repetição da mesma microinstrução por um número predeterminado de vezes. É conveniente que a UC possua recursos para possibilitar essa repetição, sem necessidade de formação de elos no microprograma. Com isso, reduz-se o número de acessos à memória de controle e também o número total de microoperações realizadas por um fator de pelo menos dois.

A troca de microinstrução corrente ocorre quando o conteúdo do contador de repetição de microinstruções for igual a zero. Esse contador é predisposto pela microinstrução quando essa é injetada no registrador de macro e microinstruções, determinando-se assim, o número de repetições.

Um dos aspectos mais controvertidos e interessantes da microprogramação surge substituindo-se o subsistema de controle MAL por um sistema MAD. Nesse caso, o usuário tem possibilidades de escrever seus próprios microprogramas e estabelecer um repertório de macroinstruções. Esta operação é própria para emulação, que tem encontrado crescente aplicação prática ([ZUF1978]).

A integração em larga escala tornou viável a difusão do emprego de microprogramação. A implementação de macroinstruções complexas pode facilmente ser realizada com a microprogramação e, com a redução dos custos dos CIs, a microprogramação tem-se tornado econômica para processadores de menor porte e cujo repertório não necessita obrigatoriamente ser composto de instruções mais elaboradas. A utilização de subsistemas MAD como memórias de controle recebe o nome unidade de controle dinamicamente microprogramável. O subsistema MAD deve ser de acesso extremamente rápido, para permitir acesso rápido às sucessivas microinstruções.

Para encerrar o assunto microprogramação, deve-se chamar a atenção ao fato de que deve ser dedicado especial cuidado à depuração do microprograma (*debug*), isto é, ao processo de verificação detalhado de validade dos cordões de *bits* armazenados na memória de controle. Na medida do possível, deve-se utilizar todo suporte de computação nesses testes, em virtude de ser impraticável em grandes sistemas outro tipo de verificação ([MON1996]).

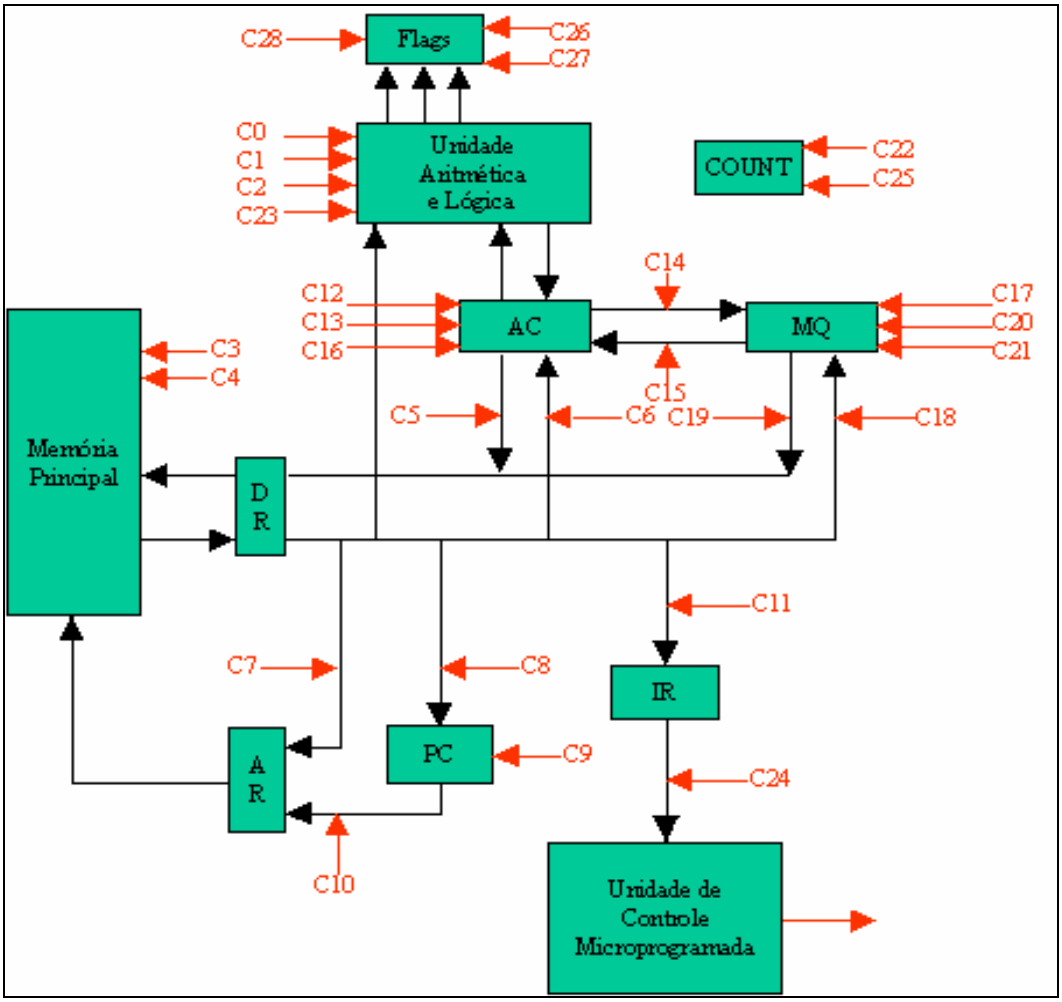
5 PROTÓTIPO

Neste capítulo, serão apresentadas a especificação e a arquitetura utilizada para o desenvolvimento do protótipo, e por fim o funcionamento deste.

5.1 ESPECIFICAÇÃO DA CPU HIPOTÉTICA

O principal objetivo do protótipo é mostrar a seqüência de microinstruções necessárias para a execução das instruções da CPU. A execução deve permitir que o usuário visualize os sinais de controle gerados e para tanto, foi adotada, neste trabalho, a organização apresentada por [HAY1978] (Figuras 4 e 5) como modelo de CPU para a execução das instruções.

Figura 4 - Estrutura de uma CPU estendida



Fonte: [HAY1978]

Figura 5 - Sinais de controle da CPU estendida

Sinal de Controle	Operação Controlada	Comentários
C0	AC ← AC + DR	Executa operação de soma com o conteúdo do DR, sendo que o resultado permanece no ACC
C1	AC ← AC – DR	Executa operação de subtração com o conteúdo do DR, sendo que o resultado permanece no ACC
C2	AC ← $\overline{AC}$	Executa operação do complemento de AC
C3	DR ← M(AR)	Executa uma leitura na memória.
C4	M(AR) ← DR	Executa uma gravação na memória.
C5	DR ← AC	O conteúdo do Acumulador (AC) é transferido para o registrador de dados da memória (DR).
C6	AC ← DR	O conteúdo do registrador de dados da memória (DR) é transferido para o Acumulador.
C7	AR ← DR(ADR)	O conteúdo do campo de endereço da instrução é transferido para o registrador de endereço da memória.
C8	PC ← DR(ADR)	O conteúdo do campo de endereço da instrução é transferido para o registrador contador de programa.
C9	PC ← PC + 1	Incrementa o conteúdo do contador de programa.
C10	AR ← PC	Transfere o conteúdo do registrador contador de programa para o registrador de endereço da memória.
C11	IR ← DR(OP)	Transfere o conteúdo do código de operação da instrução armazenada no registrador de dados da memória para o registrador de instruções.
C12	RIGHT-SHIFT AC	Faz o deslocamento para a direita do conteúdo do Acumulador.
C13	LEFT-SHIFT AC	
C14	RIGHT-SHIFT (AC,MQ)	
C15	LEFT-SHIFT (AC,MQ)	
C16	AC ← 0	Move zero para o Acumulador.
C17	AC(0) ← AC	
C18	MQ ← DR	Move o conteúdo do registrador de dados da memória para o registrador MQ.
C19	DR ← MQ	Move o conteúdo do registrador MQ para o registrador de dados da memória
C20	MQ(n-1) ← 1	Move 1 para o registrador MQ
C21	MQ(n-1) ← 0	Move 0 para o registrador MQ
C22	COUNT ← COUNT + 1	Incrementa o registrador COUNT
C23	AC ← AC – DR	Executa operação de soma com o conteúdo do DR, sendo que o resultado permanece no ACC
C24	μPC ← IR	Move o conteúdo do registrador de instruções para o registrador μPC da unidade de controle.
C25	COUNT ← 0	Move zero para o registrador Count.
C26	V ← 0	Move 0 para V (Indicador de Status)
C27	V ← 1	Move 1 para V (Indicador de Status)
C28	FLAGS ← 0	Zera os flags

Fonte: [HAY1978]

A Figura 5 apresenta os 29 sinais de controle que encontram-se na estrutura da CPU estendida e suas respectivas operações de controle. Para a representação das operações de

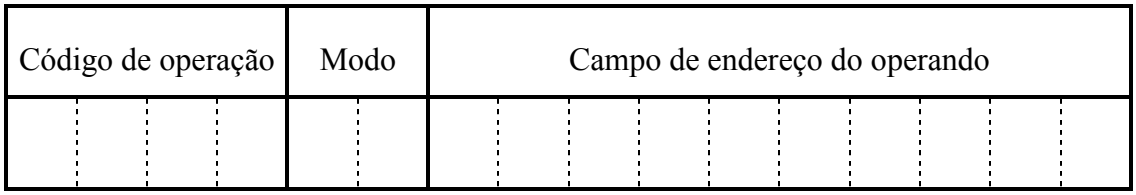
controle utiliza-se a seta para a esquerda (←) que significa uma atribuição, portanto, C3 significa que o registrador de dados (DR) recebe o conteúdo da palavra de memória endereçada pelo registrador de endereços (AR), ou seja, uma operação de leitura na memória.

Este trabalho se baseia em uma CPU hipotética, definida como uma máquina de uso geral, com 1024 *bytes* de memória, palavra de 16 *bits*, conjunto de 16 instruções básicas e 3 modos de endereçamento (imediato, direto e indireto).

5.1.1 FORMATO DA INSTRUÇÃO

As instruções da CPU hipotética tem tamanho fixo de 16 *bits*, conforme apresentado na figura 6. Os 4 *bits* de mais alta ordem armazenam o código de operação da instrução. Os próximos 2 bits armazenam o modo de endereçamento e os 10 *bits* restantes, o campo de endereço do operando. O conteúdo do campo de endereço do operando dependerá do modo de endereçamento especificado na instrução.

Figura 6 - Formato da instrução



5.1.2 MODOS DE ENDEREÇAMENTO

Três são os modos de endereçamento utilizados:

a) direto: neste modo o campo de endereço da instrução especifica o endereço físico do operando da instrução, por exemplo, a instrução *LOAD* VALOR.

b) indireto: neste modo o campo de endereço da instrução contém o endereço de uma palavra de memória que armazena o endereço do operando da instrução, por exemplo, a instrução *LOAD* [VALOR].

c) imediato: neste modo o campo de endereço contém o próprio operando. Neste caso este campo passa a ser o operando da instrução, por exemplo, a instrução *LOAD* #VALOR.

5.1.3 CONJUNTO DE INSTRUÇÕES BÁSICAS

O conjunto de instruções básicas do protótipo é limitado em 16 (2^4). Estas instruções podem ser definidas para tratar programas mais complexos e mostrar todo o funcionamento do mesmo. Esta definição de limite da quantidade de instruções na verdade não ocorre. Isto porque uma instrução para o protótipo é lida através do arquivo de instruções; com isto, podem ser criadas instruções ilimitadas, desde que utilizem as conexões previstas no protótipo.

A seguir será exemplificado como podem ser implementadas as instruções em diferentes modos de acesso. O exemplo ilustra a execução da instrução ADD no modo Direto e Indireto, isto ocorre após a busca da instrução e decodificação da mesma.

No modo direto têm-se a seqüência de conexões C7, C3 e C0. Segue a explicação passo a passo de cada uma dessas conexões e execuções:

a) C7 – AR ← DR(ADR) Transfere o conteúdo do campo de endereço da instrução para o AR;

b) C3 – DR ← M(AR) leitura do operando da instrução;

c) C0 – ACC ← ACC + DR executa a operação aritmética com o conteúdo do DR, sendo que o resultado permanece no ACC.

Para o modo indireto, a seqüência de conexões é C7, C3, C7, C3, C0:

a) C7 – AR ← DR(ADR) Transfere o conteúdo do campo de endereço da instrução para o AR;

b) C3 – DR ← M(AR) leitura do endereço do operando da instrução;

c) C7 – AR ← DR(ADR) Transfere o conteúdo do campo de endereço da instrução para o AR;

d) C3 – DR ← M(AR) leitura do operando da instrução;

e) C0 – ACC ← ACC + DR executa a operação aritmética com o conteúdo do DR, sendo que o resultado permanece no ACC.

Observando a estrutura da CPU hipotética verificamos que o endereço indireto limita-se ao campo de endereço da instrução, já que pode-se transferir apenas o conteúdo do DR(ADR) (ou seja, a parte do endereço) para o AR e não o DR inteiro.

5.1.4 COMPONENTES BÁSICOS

A CPU considerada apresenta os seguintes componentes básicos:

a) acumulador (AC) – é um registrador de uso geral contendo 10 bits. Este registrador é utilizado pelas instruções aritméticas, lógicas e de desvio condicional;

b) registrador de dados (DR) – este registrador contém 16 bits e mantém o conteúdo do dado a ser armazenado na memória em uma operação de escrita (sinal de controle C4) ou o dado transferido da memória em uma operação de leitura (sinal de controle C3). A quantidade de bits deste registrador determina o tamanho da palavra de memória. Neste trabalho definiu-se o tamanho da palavra de memória conforme descrito na seção anterior "formato da instrução".

c) registrador de endereço da memória (AR) – este registrador contém 10 bits e armazena o endereço da célula de memória durante uma operação de acesso à memória (sinais de controle C3 e C4). A quantidade de bits deste registrador determina o tamanho da memória principal (quantidade de palavras, $2^{10} = 1024$).

d) COUNT – registrador, contador;

e) MQ – registrador multiplicador quociente;

f) registrador de instruções (IR) – este registrador mantém o código de operação da instrução durante a sua decodificação e execução.

g) *flags* – este registrador mantém o *status* da CPU. Através da utilização destes bits pode-se testar o resultado de qualquer operação aritmética. Estes *flags* são atualizados a cada operação aritmética ou lógica.

h) memória principal (M) – nela estará armazenado o programa conforme arquitetura de Von Neumann. A memória principal apresenta dois sinais de entrada C3 e C4 para leitura e escrita, respectivamente. Portanto, o sinal C3 fará com que o conteúdo presente em C3 seja utilizado para localizar a palavra de memória a ser transferida para o registrador DR.

i) unidade de controle microprogramada – esta é a responsável pela geração dos sinais de controle da CPU, portanto, é a partir desta que saem todos os sinais de controle.

j) unidade aritmética e lógica – este componente é que executa as operações matemáticas e atualiza os valores dos *flags*.

5.2 ESPECIFICAÇÃO DO PROTÓTIPO

Os programas que podem ser executados no protótipo e sua arquitetura (registradores, conjunto e formato das instruções etc.) terão finalidades didáticas e atingirão este objetivo ao permitir que o usuário selecione, visualize e eventualmente interaja com os diversos níveis de computação e informação que são textual e graficamente mostrados nas várias janelas disponíveis. As seguintes funções estão disponíveis ao usuário:

a) selecionar uma dentre as funções ou programas disponíveis;

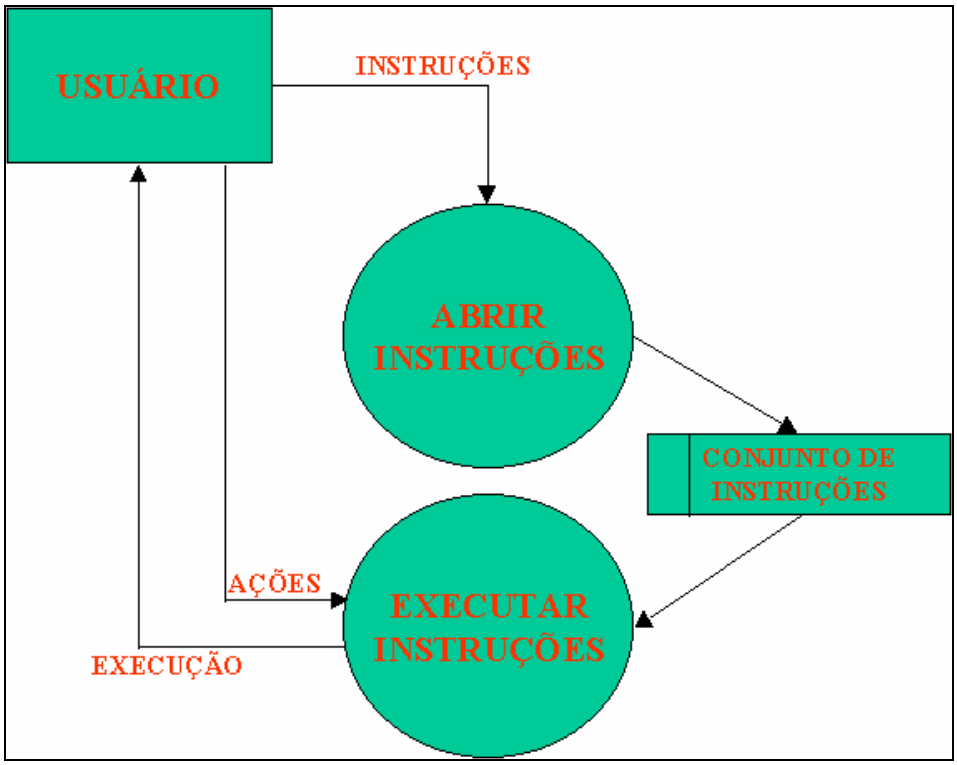
b) provocar um *reset* na CPU;

c) ver detalhadamente, passo a passo, todos os sinais de controle que ocorrem no interior da CPU.

5.2.1 DIAGRAMA DE CONTEXTO

A Figura 7 representa o diagrama de contexto do protótipo.

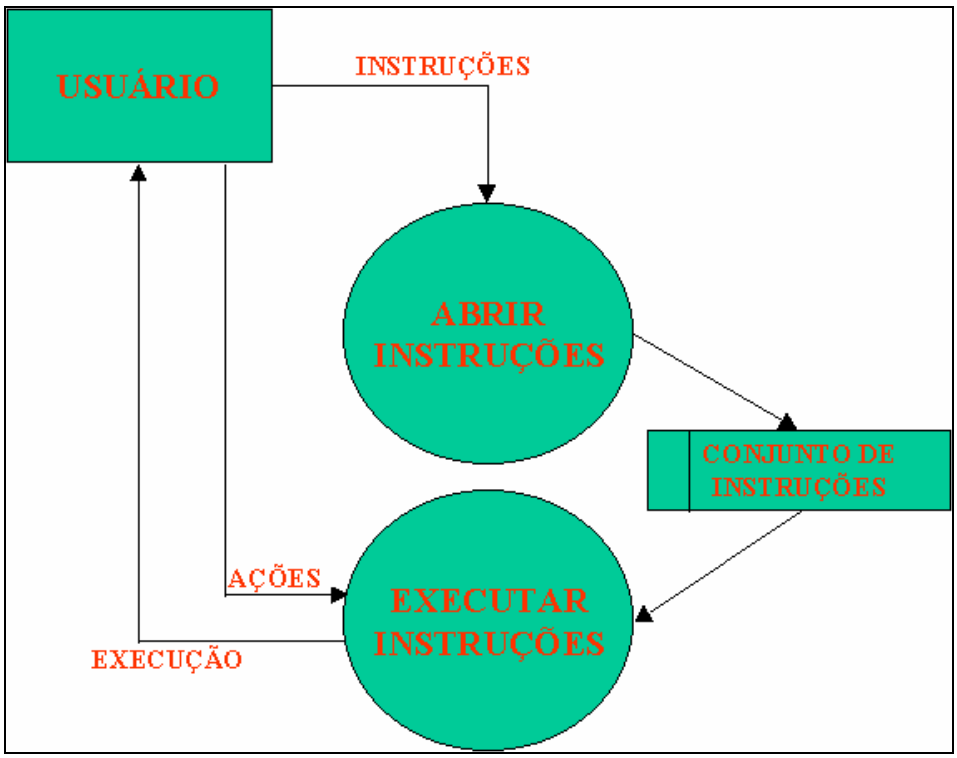
Figura 7 - Diagrama de Contexto



5.2.2 DIAGRAMA NÍVEL 0

A Figura 8 representa o diagrama nível 0.

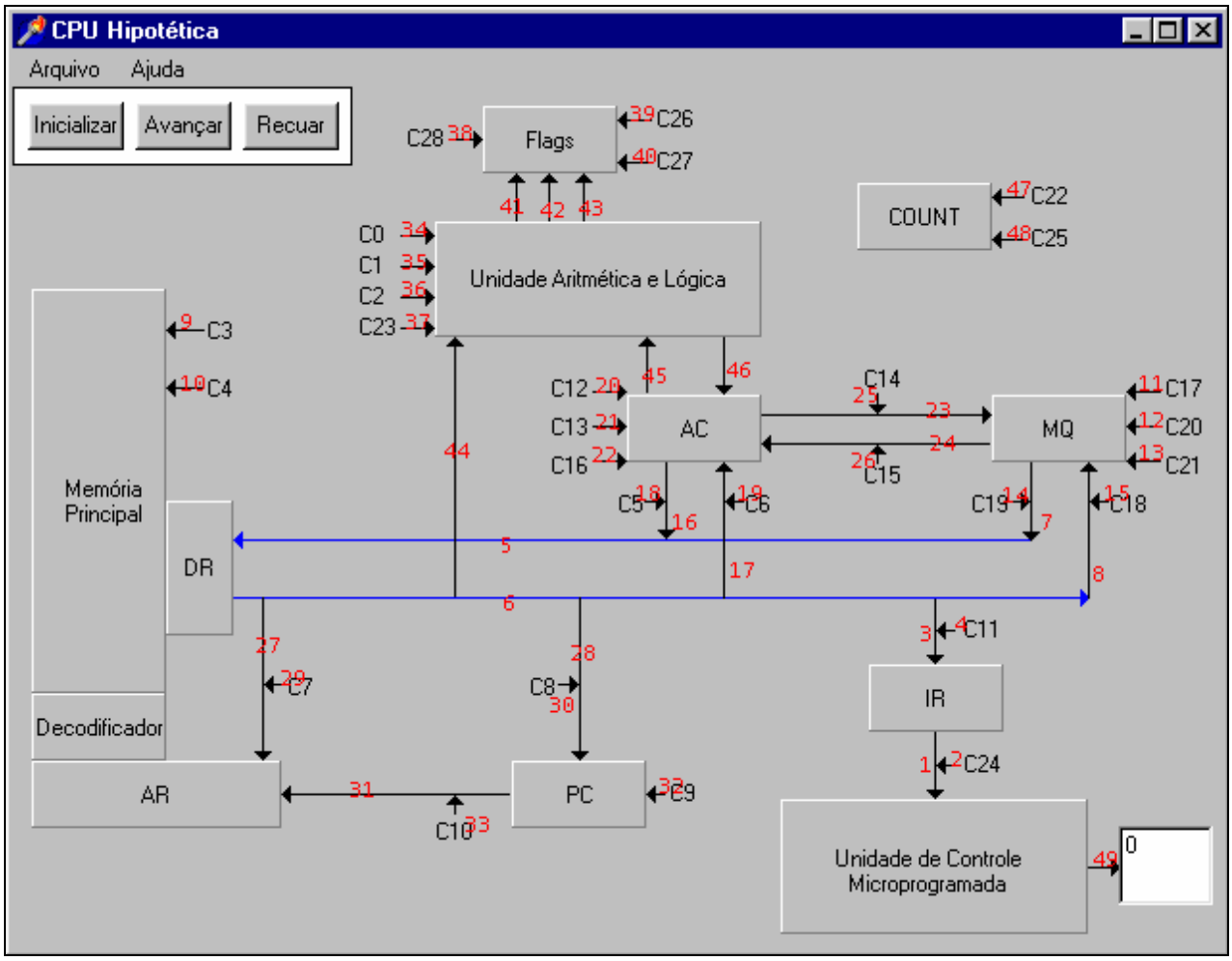
Figura 8 - Diagrama Nível 0



5.2.3 ARQUITETURA DO PROTÓTIPO

Para a construção do protótipo foram utilizados os objetos do tipo "*shape*" do Delphi e representar as unidades já caracterizadas, e 49 retas ou setas para a representação das conexões entre essas unidades, conforme demonstrado na tela principal do protótipo na figura 9. Pode-se observar as 49 retas apresentadas e suas posições com fonte na cor vermelha.

Figura 9 - Disposição das Retas



A utilização neste formato facilitou muito na fabricação do cenário onde se farão as simulações, e também facilitou a implementação do protótipo.

5.2.3.1 ESTRUTURAS DE DADOS

A representação dos sinais de controle é feita através de uma estrutura que contém uma entrada para cada sinal. Dentro de cada uma das entradas existem armazenados os números dos segmentos de reta que representarão as conexões que serão ativadas a partir do sinal de controle correspondente. São 6 segmentos de reta no máximo e um texto explicativo, que será mostrado no momento de execução do sinal. Na Figura 10 está representada esta estrutura de dados.

Figura 10 - Estrutura dos sinais de controle e suas retas

Sinal	Reta 1	Reta 2	Reta 3	Reta 4	Reta 5	Reta 6	Texto
0	34	44	45	46	00	00	AC ← AC + DR
1	35	44	45	46	00	00	AC ← AC – DR
2	36	45	46	00	00	00	AC ← AC
3	09	00	00	00	00	00	DR ← M(AR)
4	10	00	00	00	00	00	M(AR) ← DR
5	18	16	00	00	00	00	DR ← AC
6	19	17	00	00	00	00	AC ← DR
7	29	27	00	00	00	00	AR ← DR(ADR)
8	30	28	00	00	00	00	PC ← DR(ADR)
9	32	00	00	00	00	00	PC ← PC + 1
10	33	31	00	00	00	00	AR ← PC
11	04	03	00	00	00	00	IR ← DR(OP)
12	20	00	00	00	00	00	RIGHT-SHIFT AC
13	21	00	00	00	00	00	LEFT-SHIFT AC
14	25	23	00	00	00	00	RIGHT-SHIFT (AC,MQ)
15	26	24	00	00	00	00	LEFT-SHIFT (AC,MQ)
16	22	00	00	00	00	00	AC ← 0
17	11	00	00	00	00	00	AC(0) ← AC
18	15	08	00	00	00	00	MQ ← DR
19	14	07	00	00	00	00	DR ← MQ
20	12	00	00	00	00	00	MQ(n-1) ← 1
21	13	00	00	00	00	00	MQ(n-1) ← 0
22	47	00	00	00	00	00	COUNT ← COUNT + 1
23	37	44	45	46	00	00	AC ← AC – DR
24	02	01	00	00	00	00	μPC ← IR
25	48	00	00	00	00	00	COUNT ← 0
26	39	00	00	00	00	00	V ← 0
27	40	00	00	00	00	00	V ← 1
28	38	00	00	00	00	00	FLAGS ← 0

Os segmentos de reta, por sua vez, estão representados na estrutura descrita na Figura 11. Para cada segmento de reta, de um total de 49, são armazenados as informações de direção, sentido, ponto inicial (X e Y), tamanho e cor.

Figura 11 - Estrutura das retas e seus componentes

Reta	Direção	Sentido	X1	Y1	Tamanho	Cor
1	V	B	480	335	034	Preto
2	H	E	481	352	010	Preto
3	V	B	480	265	034	Preto
4	H	E	481	282	010	Preto
5	H	E	115	235	415	Azul
6	H	D	115	265	445	Azul

Reta	Direção	Sentido	X1	Y1	Tamanho	Cor
7	V	B	530	195	040	Preto
8	V	C	560	195	071	Preto
9	H	E	080	126	020	Preto
10	H	E	080	156	020	Preto
11	H	E	580	158	018	Preto
12	H	E	580	176	018	Preto
13	H	E	580	194	018	Preto
14	H	D	519	215	010	Preto
15	H	E	561	215	010	Preto
16	V	B	340	195	039	Preto
17	V	C	370	195	071	Preto
18	H	D	329	215	010	Preto
19	H	E	371	215	010	Preto
20	H	D	302	158	017	Preto
21	H	D	302	176	017	Preto
22	H	D	302	194	017	Preto
23	H	D	390	170	119	Preto
24	H	E	390	185	119	Preto
25	V	B	450	159	010	Preto
26	V	C	450	186	010	Preto
27	V	B	130	265	084	Preto
28	V	B	295	265	084	Preto
29	H	E	131	310	010	Preto
30	H	D	284	310	010	Preto
31	H	E	140	367	119	Preto
32	H	E	330	367	010	Preto
33	V	C	230	368	010	Preto
34	H	D	202	077	017	Preto
35	H	D	202	093	017	Preto
36	H	D	202	109	017	Preto
37	H	D	202	125	017	Preto
38	H	D	227	027	017	Preto
39	H	E	315	017	018	Preto
40	H	E	315	039	018	Preto
41	V	C	262	045	045	Preto
42	V	C	279	045	045	Preto
43	V	C	297	045	045	Preto
44	V	C	230	130	135	Preto
45	V	C	330	130	029	Preto
46	V	B	370	130	029	Preto
47	H	E	510	057	017	Preto
48	H	E	510	079	017	Preto
49	H	D	560	405	016	Preto

Todas os segmentos de reta (49) foram representados a partir desta definição. Para o campo **direção** os valores utilizados são “V” para segmentos na vertical e “H” para segmentos na horizontal; para o campo **sentido** os valores utilizados são dependentes também da direção, pois para segmentos na vertical (“V”) utilizam-se os valores “B” para os segmentos que possuem orientação para baixo; “C” para os segmentos que possuem orientação para cima e “ ” (branco) para segmentos sem orientação (reta 9). Finalmente para

os segmentos horizontais (“H”) utilizam-se os valores “D” para os segmentos que possuem orientação para a direita; “E” para os segmentos que possuem orientação para a esquerda e “ ” (branco) para segmentos sem orientação; este campo **sentido** implementa a seta da reta conforme as indicações descritas.

Os campos **X1** e **Y1** correspondem ao primeiro ponto (ou ponto inicial) do segmento na *form* principal da CPU hipotética; onde **X1** representa o ponto inicial horizontalmente na *form* principal, da esquerda para a direita; e **Y1** representa o mesmo ponto no sentido vertical da *form*, de cima para baixo.

O campo tamanho indica o comprimento em relação ao ponto inicial do segmento e o campo cor estabelece a qualidade de cor para o segmento, de acordo com uma codificação pré definida, que será apresentado a seguir. Esta última função foi elaborada para reconhecer o estado das conexões da CPU hipotética. Assumiu-se como padrão o preto para as conexões inativas e o azul para as ativas; com a utilização das demais cores será possível elaborar uma espécie de caminho, isto é, representar um rastro pelas conexões que acabaram de deixar o estado ativo e passaram à inativo.

O armazenamento das instruções programadas é feito a partir de um arquivo com extensão de arquivo CPU, denominado de arquivo de instruções. Este arquivo contém simplesmente as conexões a serem executadas e é lido para outra estrutura, apresentada na Figura 12, que apresenta a seqüência de conexões para busca e decodificação de uma instrução (C10, C3, C11, C24, C9).

Figura 12 - Estrutura das conexões

Seqüência	Conexão
1	110
2	103
3	111
4	124
5	109

Cada registro contém um número que especifica a cor e o número do sinal controle que se deseja executar. No caso acima o primeiro registro contém 110 que significa a cor 1 e o sinal de controle C10. As cores são identificadas através dos códigos relacionados abaixo:

a) 1-azul;

b) 2-vermelho;

c) 3-verde;

d) 4-amarelo;

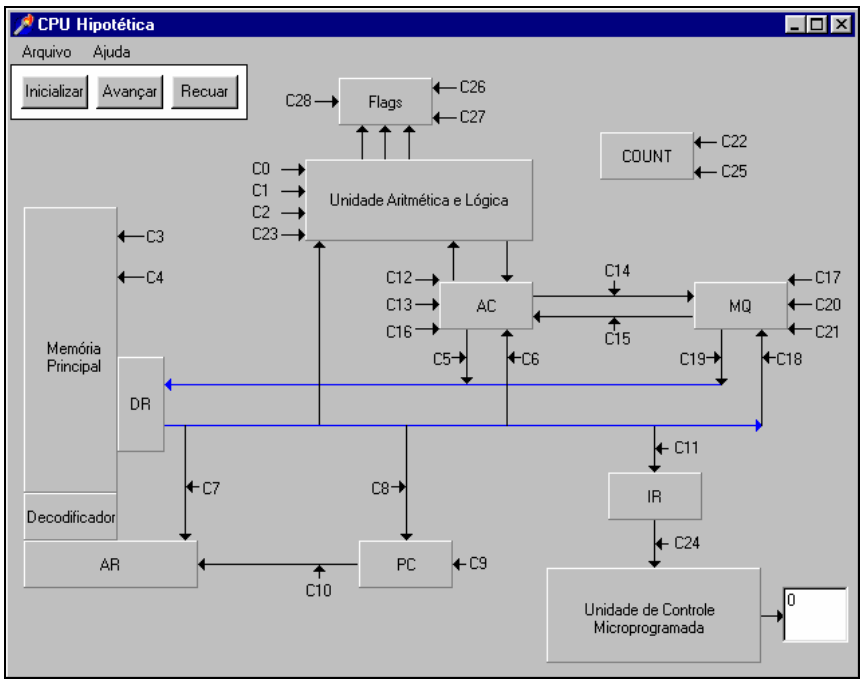
e) 5-branca;

f) 6-cinza.

5.3 FUNCIONAMENTO DO PROTÓTIPO

Basicamente o funcionamento do protótipo dá-se através do incremento (ou decremento) de um índice, sobre qual a próxima (ou anterior) instrução será executada. O controle deste índice é feito através dos botões de controle INICIALIZA (zera o índice), AVANÇAR (incrementa o índice) e RECUAR (decrementa o índice); assim têm-se a oportunidade de verificar, passo a passo, o funcionamento da CPU estendida. Os botões em questão e os menus podem ser verificados na Figura 13, que apresenta a tela principal do protótipo.

Figura 13 - Tela principal do protótipo



A partir do menu ABRIR é possível abrir o arquivo de instruções, que obedece um *layout* de comandos pré definidos, já descrito. O arquivo aberto passa por uma validação de dados.

Estes comandos são na verdade indicadores para as conexões, que por sua vez identificam as retas que serão acionadas no decorrer da execução. O acionamento nada mais é do que a troca de cor do segmento de reta em questão, representando a troca do estado da conexão.

A execução é feita seqüencialmente. Ao se acionarem os botões de ação (INICIALIZAR, AVANÇAR, RECUAR) são acionadas as retas indicadas, similar a utilização de um vídeo cassete e, com isto possibilitando a simulação dos comandos da CPU.

Cada vez que o botão AVANÇAR for executado será acionada uma conexão, independente de a mesma ter uma ou 5 retas, todo este controle é feito pelo protótipo.

O botão RECUAR foi criado para, em sala de aula, ser possível voltar um passo atras e rever o movimento executado, em caso de dúvidas.

O menu AJUDA trata somente uma tela de apresentação do protótipo e identifica o Trabalho de Conclusão de Curso.

5.3.1 EXEMPLO DE UMA INSTRUÇÃO

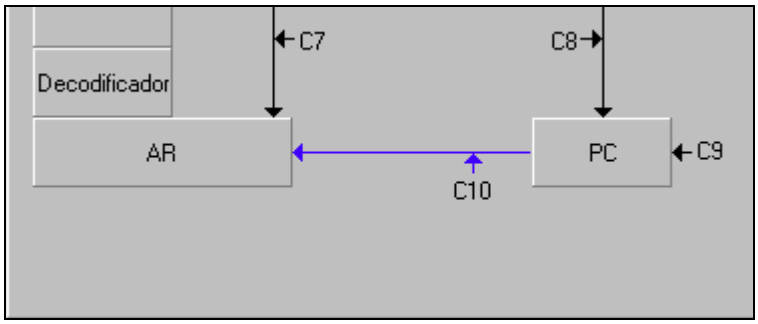
A seguir será descrito passo a passo a execução de busca de uma instrução, para isso o protótipo deve executar os seguintes sinais de controle: C10, C3, C11, C24, C9. Estes sinais devem estar em um arquivo de instruções representado na Figura 14.

Figura 14 - Sinais de controle da busca de uma instrução

Seqüência	Conexão
1	110
2	103
3	111
4	124
5	109

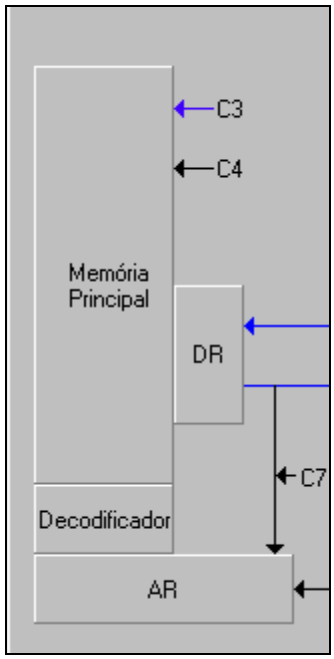
A Figura 15 representa a execução do 1º passo, através da ativação do sinal C10, este sinal habilita a transferência do conteúdo do registrador PC para o registrador AR.

Figura 15 - Exemplo 1º passo



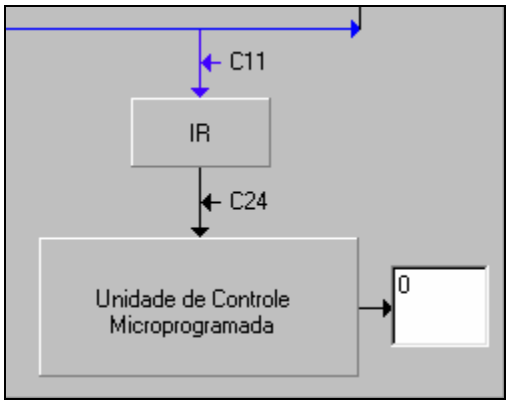
O segundo passo é habilitar o sinal de controle C3 (DR ← M(AR)). Este sinal ativa a memória para leitura. Ao final desta operação o registrador DR terá o conteúdo da palavra de memória endereçado por AR (próxima instrução). A Figura 16 ilustra este passo.

Figura 16 - Exemplo 2º passo



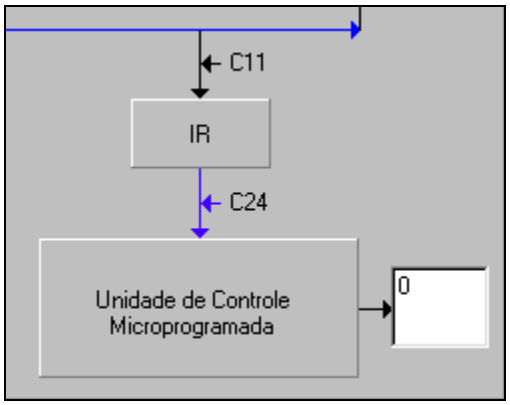
No próximo passo o conteúdo de DR é transferido para o registrador IR, através do sinal de controle C11 – Figura 17.

Figura 17 - Exemplo 3º passo



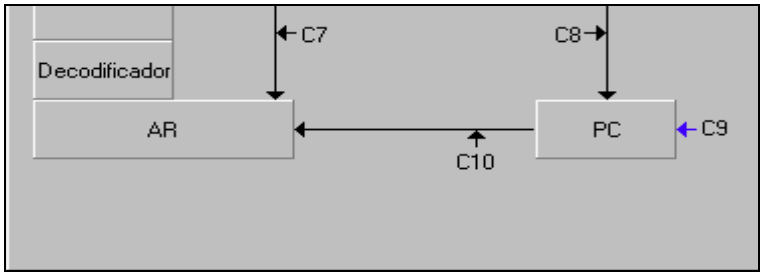
Depois disso o próximo sinal de controle é o C24. A partir deste sinal a unidade de controle microprogramada passa a decodificar a instrução, Figura 18.

Figura 18 - Exemplo 4º passo



O registrador PC deverá ser incrementado no próximo passo, através do sinal de controle C9, Figura 19. Este ciclo poderá ocorrer em paralelo com a operação de decodificação do código de operação da instrução, uma vez que são operações independentes.

Figura 19 - Exemplo 5º passo



6 CONCLUSÃO E EXTENSÃO

Este capítulo contém as conclusões e as possíveis extensões deste trabalho. Deve-se enfatizar que a continuação deste trabalho através de extensões, é sem dúvida de grande importância, para a Arquitetura de Computadores e seus estudantes.

6.1 CONCLUSÃO

Observou-se que com a utilização das estruturas, o protótipo atingiu seu principal objetivo. A animação das retas é possível, através do redesenho das linhas, criando-se a sensação de movimento e permitindo o usuário a imaginar o funcionamento da CPU.

Deve ser lembrado que há abstrações na animação proporcionada pelo protótipo: na realidade conforme têm-se as entradas nos circuitos lógicos, estes produzirão uma saída, independente de o mesmo estar "animado" ou não. Também fez-se propositadamente a animação de apenas um componente por vez, o que não ocorre na realidade.

6.2 EXTENSÃO

Como melhoramento pode-se fazer uma definição mais ampla das instruções a serem utilizadas no protótipo e mostrar o conteúdo de cada um dos registradores em uma janela separada, podendo-se assim compará-los e assim o usuário aluno pode ter um melhor entendimento do que ocorre dentro da CPU.

Outro melhoramento é a possibilidade de ser implementado um interpretador de comandos, para que a seqüencialização dos sinais de controle sejam automaticamente geradas a partir do mouse.

REFERÊNCIAS BIBLIOGRÁFICAS

[CAN1998] CANTU, Marco. **Dominando Delphi 3: a bíblia.** São Paulo (SP) : MakronBooks, 1998.

[COS1993] COSTA, Simone Erbs da. **Estudos de metodologias de prototipação de sistemas e sua aplicabilidade no ambiente Genexus.** Blumenau (SC) : FURB, 1993. (Trabalho de Conclusão de Curso).

[DON1991] DONDIS, Donis A. **Sintaxe da linguagem visual**. São Paulo : Martins Fontes, 1991.

[HAY1978] HAYES, John P.. ***Computer architecture and organization*.** São Paulo (SP) : McGraw-Hill, 1978.

[LAN1974] LANGDON Jr., Glen George; Flegni, Edson. **Projeto de computadores digitais**. São Paulo : Edgard Blücher, 1974, 2 ed.

[MAC1997] MACHADO, Luciano Dozól. **Estudo e aplicação da técnica de prototipação rápida orientada a objetos.** Blumenau (SC) : FURB, 1997. (Trabalho de Conclusão de Curso).

[MAL1985] MALVINO, A. **Microcomputadores e microprocessadores**, São Paulo : McGraw-Hill, 1985.

[MEI1980] MEINADIER, Jean-Pierre. ***Estructura y funcionamiento de los computadores digitales*.** Madrid : AC, 1980.

[MEL1990] MELENDEZ FILHO, Rubem. **Prototipação de sistemas de informações: fundamentos, técnicas e metodologia.** Rio de Janeiro (RJ): LTC - Livros Técnicos e Científicos, 1990.

[MIL1997] MILLER, Tood. **Using Delphi 3**. Indianápolis : Que, 1997.

[MON1996] MONTEIRO, Mario A. **Introdução à organização de computadores.** Rio de Janeiro : LTC, 1996, 3 ed.

[OBR1993] O'BRIEN, Stephen. **Turbo Pascal 6 – completo e total.** São Paulo (SP) : MakronBooks, 1993.

[OSI1997] OSIER, Dan. **Aprenda em 21 dias Delphi 2.0.** Rio de Janeiro (RJ) : Campus, 1997.

[ROS1993] ROSENBORG, Victoria. **Guia de multimídia.** Rio de Janeiro (RJ) : Berkeley Brasil, 1993.

[SWA1996] SWAN, Tom. **Delphi: bíblia do programador**. São Paulo : Berkeley Brasil, 1996.

[ZUF1978] ZUFFO, João Antonio. **Fundamentos da arquitetura e organização de microprocessadores.** São Paulo : Edgard Blücher, 1978.