PARTE 3

ORGANIZAÇÃO DE UM COMPUTADOR DIGITAL E A LINGUAGEM DE MONTAGEM

Na parte 1 deste livro, vimos o que é um computador digital, definimos hardware e software. Na parte 2, estudamos o como os elementos que compõe o processador são projetados. Iremos agora estudar como um computador digital é formado, ou seja, quais seus componentes, suas funções e seu funcionamento quando da execução de uma instrução. São conceitos imprescindíveis em programação de um modo geral, mas principalmente utilizando uma linguagem de montagem. Por fim, mostraremos conceitos básicos de um linguagem de montagem, preparando para os estudos de casos que virão a seguir.

CAPÍTULO 11 – ORGANIZAÇÃO DE UM COMPUTADOR DIGITAL

Objetivos do Capítulo

Ao final deste capítulo estaremos aptos a:

- entender como um processador digital é composto e quais as funções de suas unidades;
- entender a organização de qualquer processador que por ventura nos depararmos.

APRESENTAÇÃO

Estamos tão acostumados a utilizar um computador que não paramos para analisar como ele funciona. Aqui vamos apresentar as unidades que compõem os computadores, o funcionamento de cada uma delas e como se interagem.

A Unidade Central de Processamento age como se fosse nosso cérebro, processando e armazenando as informações, as unidades de entrada e saída como se fossem nossos sentidos e a unidade de controle como sendo nosso sistema nervoso central.

Vamos então, começar a estudar as unidades e seu funcionamento.

FUNDAMENTOS

11.1 Organização Básica de um Computador Digital

Um computador digital é composto basicamente por 3 unidades:

- Sistema de Memória
- Unidade de Central de Processamento – UCP
- Unidade de Entrada e Saída (E/S)

A figura 11.1 nos mostra esta organização.

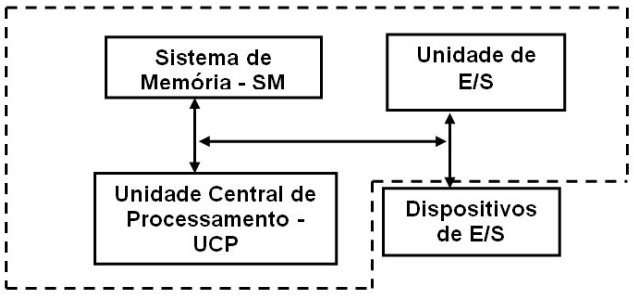


Figura 11.1 – Organização Básica de um Computador Digital

O Sistema de Memória é responsável pelo armazenamento das informações que serão processadas pela Unidade Central de Processamento – UCP. A Unidade de E/S é responsável pelo fluxo das informações entre a SM ou UCP e os Dispositivos de Entrada e Saída. Os Dispositivos de E/S são responsáveis por possibilitar a interação entre o usuário e o computador.

11.2 Sistema de Memória

A memória de um computador é o local onde as informações são armazenadas. Na realidade não temos uma memória e sim um Sistema de Memória, que é composto por vários tipos de memórias, cada um com sua função. A figura 11.2 mostra o Sistema de Memória utilizado nos computadores.

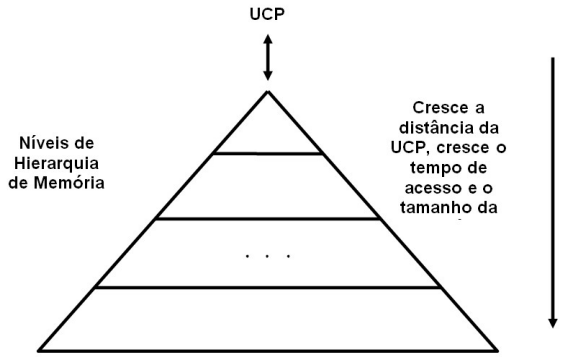


Figura 11.2 – Sistema Hierárquico de Memória

Este sistema é dito hierárquico porque é composto por níveis, onde cada um destes níveis corresponde a uma ou mais memórias com características específicas. Por exemplo, em um sistema com 3 níveis de memória (figura 11.3), o nível mais "próximo" à UCP, são os registradores internos a esta Unidade. Os registradores são as memórias com tempo de acesso mais próximos à a velocidade de processamento da UCP, com custo de armazenamento por bit maior e consequentemente de menor capacidade de armazenamento. O próximo nível corresponde à memória principal que é mais lenta que os registradores, porém com custo de armazenamento menor e com maior capacidade. O último nível corresponde à memória secundária, ou seja, o disco magnético que, por sua vez, que é mais lento, com custo de armazenamento menor e com maior capacidade, se comparada com a memória principal.

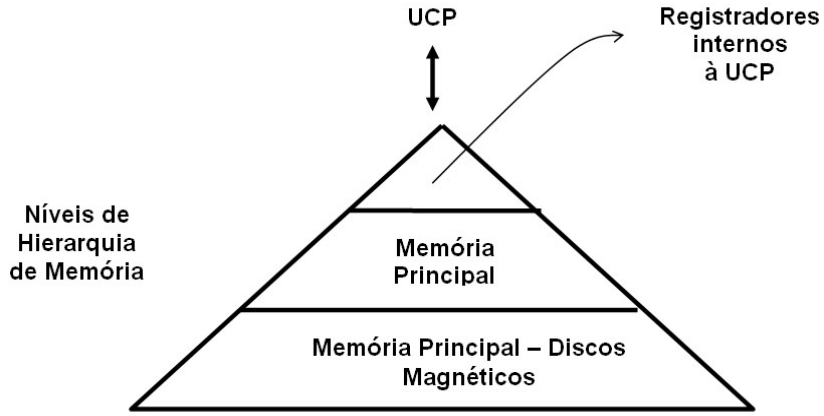


Figura 11.3 – Sistema Hierárquico de Memória com 3 níveis

Nos sistemas de memória, podemos ainda encontrar mais um nível, entre os registradores da UCP e a Memória principal, chamado de memória *cache*. Pela definição de Sistema Hierárquico, podemos concluir que as memórias caches são mais rápidas (e mais caras) que as principais e mais lentas (com menor custo) que os registradores da UCP.

Memórias semicondutoras

As memórias semicondutoras são memórias que utilizam os materiais semicondutores, por exemplo, o silício, em sua construção. Podemos dividir este tipo de memória em 3 categorias, memórias RAM, ROM e FLASH:

a. Memória de acesso aleatório – RAM (*Random Access Memory*)

As memórias tipo RAM são memórias voláteis, onde na ausência da tensão de alimentação, os valores armazenados são perdidos. Temos dois tipos de memória RAM:

- Memórias RAM estáticas – SRAM (*Static Random Access Memory*): seu conteúdo só é alterado quando sobrescrevemos outro valor ou quando

a tensão de alimentação é desligada. São exemplos deste tipo de memória, os registradores internos da UCP e a memória *cache*.

- Memórias RAM Dinâmicas DRAM (*Dynamic Random Access Memory*): Seu conteúdo tem que ser, periodicamente, reescrito para que não se perca. Este processo é chamado de *refresh* de memória. O nível de hierarquia de memória chamada de memória principal é um exemplo de memórias DRAM.

b. Memória somente de escrita – ROM (*Read Only Memory*)

As memórias tipo ROM são memórias não voláteis, ou seja, mesmo na ausência da tensão de alimentação, os valores armazenados são mantidos. Temos 4 tipos de memória ROM:

- Memórias Somente de Leitura – ROM (*Read Only Memory*): são memórias onde o conteúdo é gravado pelo fabricante da memória. Não é possível alterar as informações armazenadas após a sua gravação (figura 11.4).

Figura 11.4 – Circuitos Integrados (*chips*) de Memória ROM.

- Memórias Somente de Leitura, Programáveis - PROM (*Programmable Read Only Memory*): são memórias ROM, onde a gravação do conteúdo é feita

pelo usuário, com um dispositivo especial (figura 11.5). Não é possível alterar as informações armazenadas após a sua gravação.

Figura 11.5 – Programador de Memórias PROM.

- Memórias Somente de Leitura, Programáveis e Apagáveis - EPROM (Erasable *Programmable Read Only Memory*): são memórias ROM Programáveis, onde a gravação do conteúdo é feita pelo usuário, como as memórias PROM, e seu conteúdo pode ser apagado, aplicando-se um feixe de luz ultravioleta sobre ela (figura 11.6).

Figura 11.6 – Circuito Integrado (*chip*) de Memória EPROM.

- Memórias Somente de Leitura, Programáveis e Eletricamente Apagáveis - EEPROM (Electrical Erasable *Programmable Read Only Memory*): são memórias ROM Programáveis e Apagáveis, onde a gravação do conteúdo é feita pelo usuário, e seu conteúdo pode ser apagado, eletricamente, aplicando um determinado nível de tensão em um dos pinos do circuito integrado.

c. Memórias *FLASH*

São memórias semicondutoras, não voláteis, onde é possível escrever e ler, sem a necessidade de dispositivos especiais. Atualmente substituem, com vantagens, as memórias da família ROM. Podemos dizer que são semelhantes às memórias EEPROM, pois podemos ler e apagar seu conteúdo, mas que também se comportam como memórias RAM não voláteis. O processo de gravação e leitura é feito por blocos de múltiplos conteúdos. Um exemplo bem conhecido de utilização de memórias *Flash* são os *Pen Drives* (figura 11.7).

Figura 11.7 – Memória *Flash* em um *Pen Drive*

d. Memórias Magnéticas

Os representantes mais importantes deste tipo de memória são os discos rígidos. Os discos rígidos são organizados em pratos, trilhas e setores, como podemos ver na figura 11.8. Esta organização permite que os dados possam ser gravados e recuperados. Os pratos são revestidos de uma camada de material magnético, onde uma cabeça de gravação-leitura magnetiza uma região para armazenar um bit ou lê o campo magnético existente na região para ler o bit. O conjunto destas regiões nós chamamos de setor. O conjunto de setores, equidistantes do centro do prato, forma a trilha e o conjunto de trilhas forma o prato.

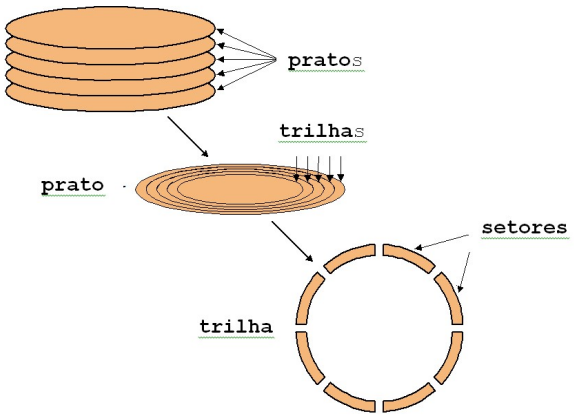


Figura 11.8 – Organização de um Disco Magnético.
(PATTERSON, David A. e HENNESSY, John L.)

Na figura 11.9 podemos ver um disco rígido, comumente utilizado, por dentro.

Figura 11.9 – Vista interna de um disco rígido

Os discos magnéticos são classificados, dentro da hierarquia de memória, como sendo memória secundária, também chamada de memória de massa.

Outros tipos de memórias magnéticas são as fitas e cartuchos magnéticos, que ainda são utilizados em cópias de segurança para sistemas de grande porte.

e. Memórias Ópticas

Outro meio de armazenamento utilizado são os discos ópticos, os CDs e os DVDs. Diferentemente de um disco rígido, os discos ópticos têm suas trilhas em espiral, como mostra a figura 11.10.

Figura 11.10 – Trilhas espirais de um disco óptico.

Organização das memórias em endereços e conteúdos

Uma memória de acesso aleatório precisa ter um rótulo (ou identificador) para cada posição, para podermos, de maneira rápida e eficiente, acessar (ler ou escrever) um conteúdo nesta posição. Este identificador é chamado de endereço e indica a posição onde está armazenado o conteúdo. A figura 11.11 mostra esta organização, para uma memória de 4MB (4×2^{20} Bytes). Note que os endereços, para esta memória, estão no intervalo de 0 a (4M -1).

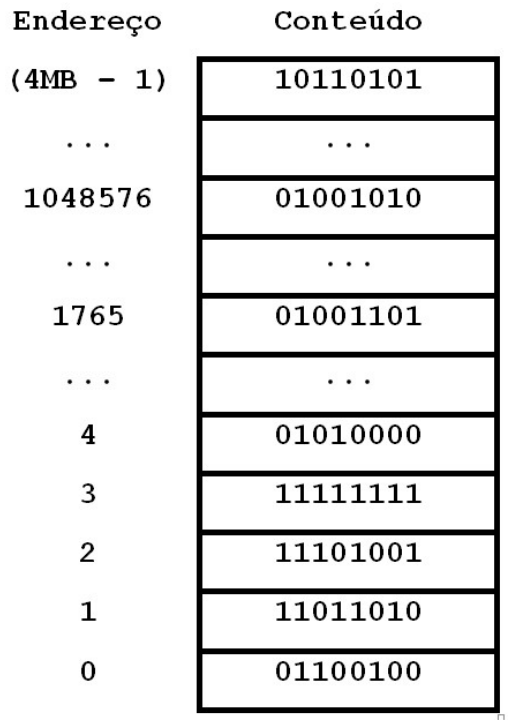


Figura 11.11 – Organização de uma memória em endereço e conteúdo.

Para ler uma informação da memória temos que fornecer o respectivo endereço para memória, e então, ela nos fornece a informação que está armazenada neste endereço. Para escrever uma informação, devemos fornecer o endereço e a informação, para que a memória armazene esta informação na posição indicada pelo endereço.

11.3 Unidade Central de Processamento

Utilizando uma analogia com o corpo humano, a Unidade Central de Processamento – UCP (ou *CPU – Central Processing Unit*) ou simplesmente processador, é o cérebro do computador. Na UCP as informações são processadas. Para possibilitar o processamento, ela deve conter registradores para o armazenamento das informações e dos resultados decorrentes do processamento, uma unidade para realizar os cálculos necessários e uma unidade que coordene todas as operações dentro da

UCP. A figura 11.12 mostra uma organização geral de uma UCP, com sua interação com o sistema de memória e a unidade de E/S.

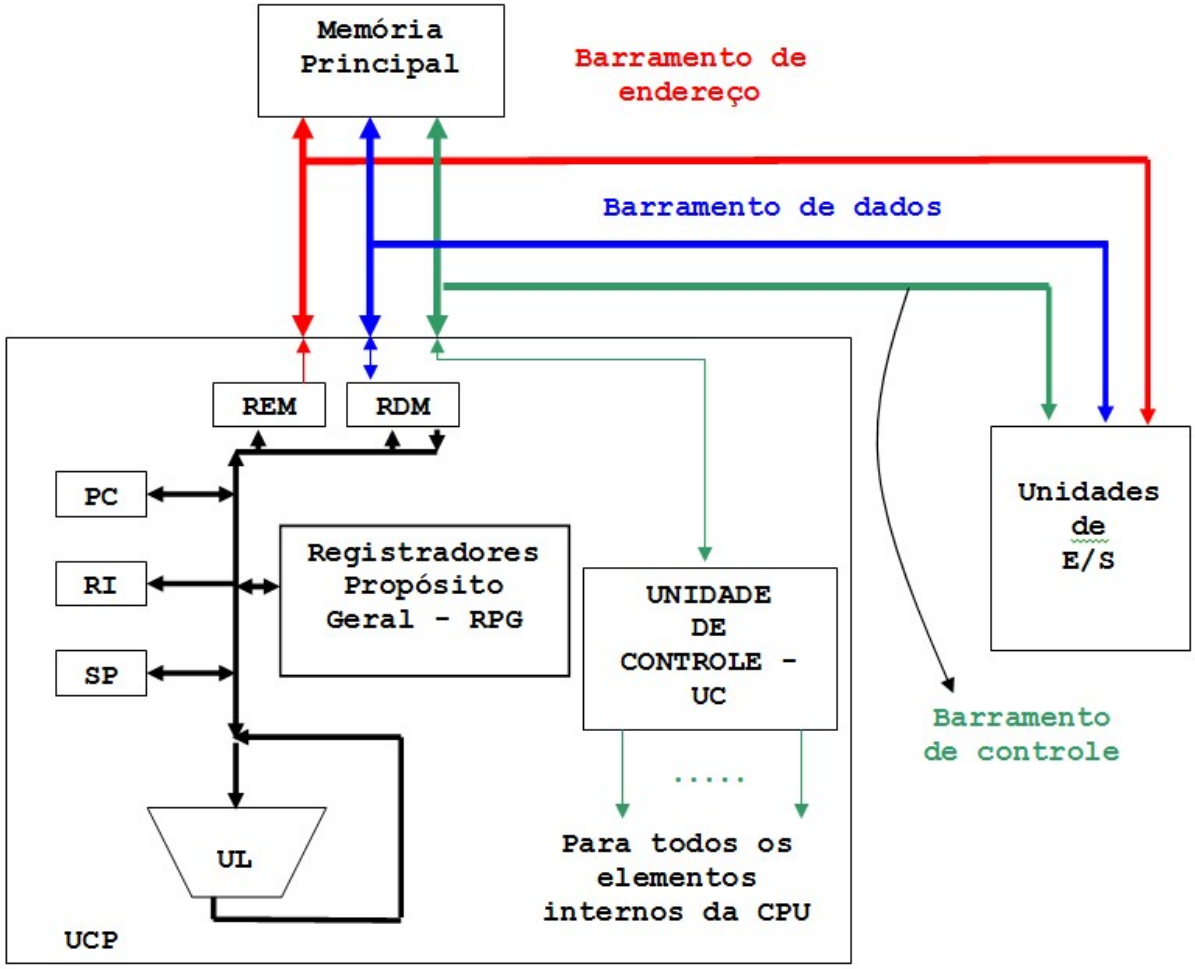


Figura 11.12 – Organização Geral de uma Unidade Central de Processamento – UCP e sua interação com o Sistema de Memória e a Unidade de E/S.

A UCP é formada basicamente por:

- **Registradores de Propósito Geral – RPG**

  Os Registradores de Propósito Geral têm a função de armazenar os dados que serão processados pela UCP. Por exemplo, se queremos somar

dois valores armazenados na memória, primeiro temos que ler os dois números da memória e armazená-los em dois dos Registradores de Propósito Geral para depois efetuarmos a soma.
O número de Registradores de Propósito Geral e seus nomes dependem do processador a ser utilizado.

- **Registradores Específicos**

Como o nome já está dizendo, estes registradores têm uma função específica, diferente da função dos Registradores de Propósito Geral. Geralmente os processadores têm os seguintes Registradores Específicos:

- Registrador de Endereços de Memória – REM

  Este registrador tem a função de guardar um endereço de memória que será acessado, por uma instrução que tenha a função de leitura ou escrita.

- Registrador de Dados de Memória – RDM

  Este registrador guarda um dado lido da memória ou um dado a ser escrito na memória.

- Registrador de Instrução – RI

  Este registrador tem a função de armazenar a instrução lida da memória, até o fim de sua execução.

- Contador de Programa – PC (*Program Counter*).

  Este registrador tem sempre armazenado o endereço de uma instrução que será executada. Outro nome utilizado para este registrador é Apontador de Instrução – IP (*Instruct Pointer*).

- Apontador de Pilha – SP (*Stack Pointer*).

A Pilha é uma organização lógica de armazenamento, implementadas fisicamente na memória principal. O registrador SP guarda sempre o endereço do último dado armazenado na pilha, ou seja, o endereço do topo da pilha.

- **Unidade de Controle – UC**

Se utilizarmos a analogia dos computadores com o corpo humano, a Unidade de Controle será o Sistema Nervoso Central. Ela é responsável por controlar todas as atividades dentro de um computador, através da geração de sinais elétricos (sinais de controle) para as unidades internas e externas à UCP. Estes sinais são gerados decorrentes da instrução que se quer executar.

- **Unidade Lógica e Aritmética – ULA**

Todas as instruções que o processador executa tem operações lógicas e/ou aritméticas. A Unidade Lógica e Aritmética tem a função de executar estas operações.

- **Barramento Interno**

O Barramento Interno é um conjunto de vias (fios) que interligam as unidades internas da UCP.

- **Barramentos Externos**

As ligações entre a UCP, o Sistema de Memória e as Unidades de E/S também são feitas através de barramentos. Temos 3 tipos de barramentos:

  - **Barramento de Dados**

    É o meio físico por onde trafegam as informações que serão de manipuladas.

  - **Barramento de Endereços**

São os caminhos que permitem com que os endereços possam ser enviados para a memória ou para a unidade de E/S.

- **Barramento de Controle**

  Transportam os sinais de controle gerados pela Unidade de Controle até as demais unidades externa à UCP, como a memória ou a unidade de E/S.

Para exemplificar a utilização destes 3 barramentos, suponha a leitura de um dado da memória. A memória sabe que é uma operação de leitura porque a UC envia um sinal de controle de leitura através do Barramento de Controle. O endereço da localização de memória a ser acessada é enviado pelo Barramento de Endereços e o dado chega à UCP pelo Barramento de Dados.

11.4 Unidade de Entrada e Saída (E/S)

As unidades de Entrada e Saída são responsáveis pela comunicação entre o computador e os periféricos ou dispositivos de Entrada e Saída. Podemos classificar as Unidades de E/S em 3 tipos:

- Interfaces
- Canais de E/S
- Processadores de E/S

- **Interfaces**

As interfaces de E/S são circuitos que compatibilizam o formato de dados utilizado pelo computador e o utilizado pelo dispositivo. Esta compatibilização inclui desde como os dados são transmitidos, serialmente ou paralelamente, até a velocidade de transmissão e níveis elétricos do sinal. Estes parâmetros são chamados de protocolo de comunicação. As interfaces são as Unidades de E/S com menor

“inteligência”, pois o grande responsável por gerenciar esta transmissão é a UCP.

Interface Serial

A figura 11.13 mostra a organização básica de uma interface serial. O dado é tratado dentro da UCP de forma paralela. Se quisermos transferir para um dispositivo que utiliza o dado de forma serial, a interface tem serializar os bits de dados e vice-versa. Assim para enviar um dado para um dispositivo serial, inicialmente ele é armazenado em um buffer de transmissão que por sua vez transfere para o registrador de transmissão que serializa e envia o dado.

Chamamos de *buffer* o conjunto de registradores que armazenam temporariamente os dados que estão sendo transferidos, pois as velocidades da UCP e dos dispositivos são diferentes e não podemos ter perda de dados.

Na recepção de um dado de um dispositivo serial, o caminho inverso é feito, o dado é armazenado no registrado de recepção, e quando a palavra estiver completa, ela é transferida para o *buffer* de recepção, deixando o dado disponível para a UCP.

O exemplo mais conhecido de uma interface serial é a interface padrão RS-232.

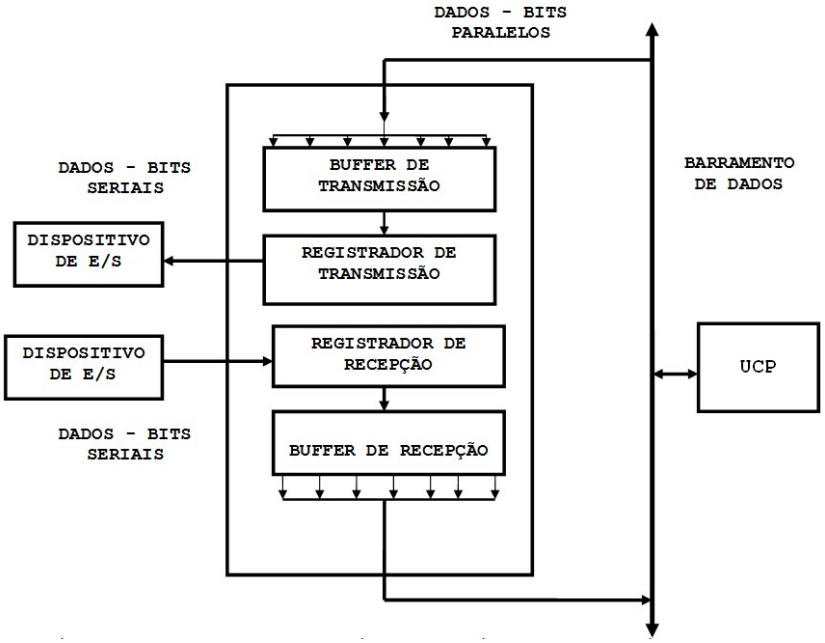


Figura 11.13 – Interface Serial e sua ligação com a Unidade Central de Processamento e Dispositivos de E/S

Interface Paralela

Na figura 11.14 temos o esquema de uma interface paralela. Neste caso, como ela é ligada a dispositivos paralelos, não há a necessidade de serializar os dados, apenas compatibilizar os parâmetros dos protocolos. Portanto o dado paralelo é armazenado no buffer de transmissão, ou de recepção se for uma operação de entrada de dados, e enviado ao dispositivo, ou à UCP.

A interface padrão RS-488 é um dos exemplos deste tipo de interface.

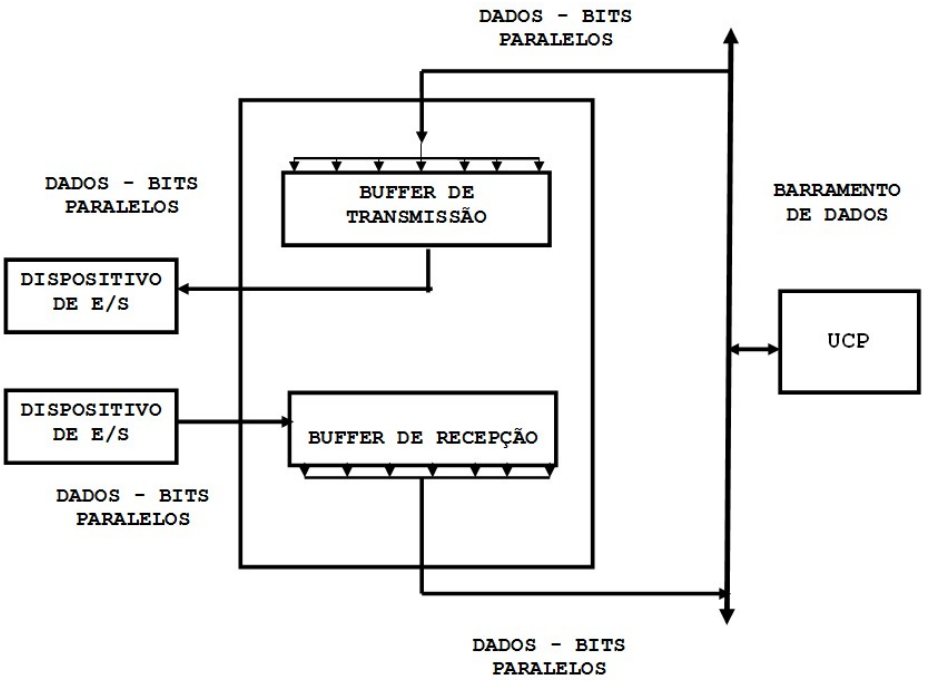


Figura 11.14 – Interface Paralela e sua ligação com a Unidade Central de Processamento e Dispositivos de E/S

- **Canais de Entrada e Saída – E/S**

Os chamados canais de E/S são circuitos com maior capacidade de processamento. Eles permitem com que a UCP tenha menor trabalho no gerenciamento de E/S de dados. Os controladores de Acesso Direto à Memória (*DMA – Direct Access Memory*) são exemplos destas unidades.
Os Controladores DMA permitem com que dados sejam transferidos, diretamente entre o disco e a memória, em blocos, sendo que o controle das transferências é feito por ele. A UCP só tem a responsabilidade de iniciar o processo de transferência. Na figura 11.15 temos o esquema básico de um controlador DMA.

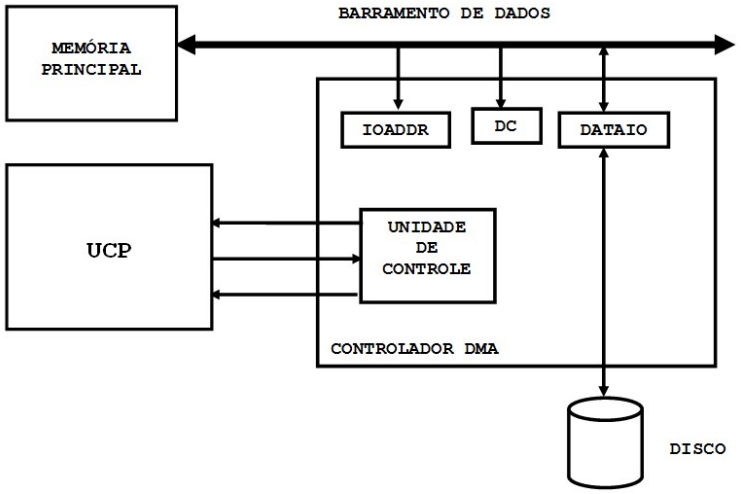


Figura 11.15 – Controlador de Acesso Direto à Memória – DMA e sua ligação com a Unidade Central de Processamento e a Memória Principal

O registrador IOADDR tem a função de guardar o endereço de memória que será acessado. O contador DC inicialmente tem o número de palavras que serão transferidas. A medida que é feito a transferência, este contador é decrementado e quando chegar a zero, a operação DMA é finalizada. O DATAIO é um *buffer*, uma vez que os tempos de acesso ao disco e à memória são diferentes. A unidade de controle é responsável pelo controle da operação de transferência e se comunica com a UCP através de sinais de controle.

- **Processadores de Entrada e Saída – E/S**

Um processador de E/S é uma UCP com um conjunto de instruções específico para operações de entrada e saída. É utilizado em sistemas de computação onde existe a necessidade de se ter grande volume de dados sendo transferidos entre o meio externo e o computador. Os grandes usuários deste tipo de unidade são os computadores com multiprocessadores utilizados em processamento paralelo, como por exemplo, os computadores vetoriais. Na verdade chamamos de processadores de E/S o conjunto formado por esta UCP específica para E/S e mais as interfaces e os controladores DMA (figura 11.16).

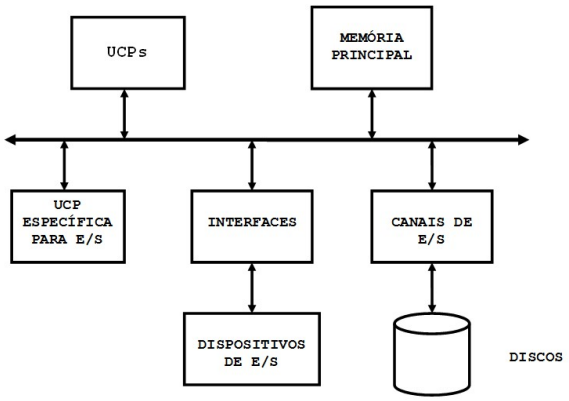


Figura 11.16 – Processador de E/S e sua ligação com a Unidade Central de Processamento e dispositivos de E/S.

Dispositivos de Entrada e Saída - E/S

Os dispositivos de E/S são equipamentos que permitem com que o mundo exterior se comunique com um sistema computacional. Estes dispositivos podem ser classificados, segundo alguns autores, levando-se em conta o comportamento (entrada, saída, entrada e saída ou de armazenamento) e com quem ele interage, com o ser humano ou com outra máquina.

A tabela 11.1 mostra alguns dos dispositivos mais comuns e sua classificação.

Dispositivos	Comportamento	Parceiro
Teclado	Entrada	Humano
Mouse	Entrada	Humano
Microfone	Entrada	Humano
Scanner	Entrada	Humano
Alto falante	Saída	Humano
Impressora	Saída	Humano
Monitor de vídeo	Saída	Humano
Modem	Saída ou Entrada	Máquina
Disco Óptico	Armazenamento	Máquina
Fita Magnética	Armazenamento	Máquina
Disco Magnético	Armazenamento	Máquina

Tabela 11.1 – Classificação dos Dispositivos de Entrada e Saída (PATTERSON, David A. e HENNESSY, John L.)

Resumo

Apresentamos neste capítulo a organização de um computador digital, apresentando também o conceito de memória hierárquica e os tipos de memória existentes, assim como as unidades responsáveis pela comunicação com o mundo externo. No próximo capítulo, mostraremos como uma instrução, em linguagem de máquina é executada pela UCP.

CAPÍTULO 12 – A EXECUÇÃO DE UMA INSTRUÇÃO E A LINGUAGEM DE MONTAGEM

Objetivos do Capítulo

Ao fim deste capítulo o leitor estará apto a:
- entender como uma instrução é executada por um processador.
- aprender conceitos que serão imprescindíveis na programação em linguagem de montagem.
- a escrever programas mais eficientes tanto quanto ao tempo de execução como ao seu tamanho.

FUNDAMENTOS

12.1 A Unidade Central de Processamento e a execução de um programa

No Capítulo 1 foi visto que um programa em linguagem de alto nível para se executado tem que ser traduzido para um programa na linguagem de máquina (código executável) e armazenado na memória principal. Vamos então mostrar como cada instrução em linguagem de máquina, deste código, é executada por uma UCP.

Uma UCP hipotética

Para trabalharmos com um processador, em uma linguagem de baixo nível, precisamos conhecer, principalmente, a organização da UCP, o formato das instruções e o conjunto de instruções deste processador.
Tomaremos como base a organização da UCP mostrada na figura 12.1. Trata-se de um processador hipotético, que trabalha com 4 registradores de propósito geral: R0, R1, R2 e R3. Além destes registradores, dois outros registradores auxiliares existem para possibilitar a correta

utilização da Unidade Lógica e Aritmética, são eles os registradores Y e Z. O registrador Y armazenará temporariamente um dos operandos que será usado pela ULA, e o registrador Z armazenará o resultado de uma operação lógica ou aritmética.

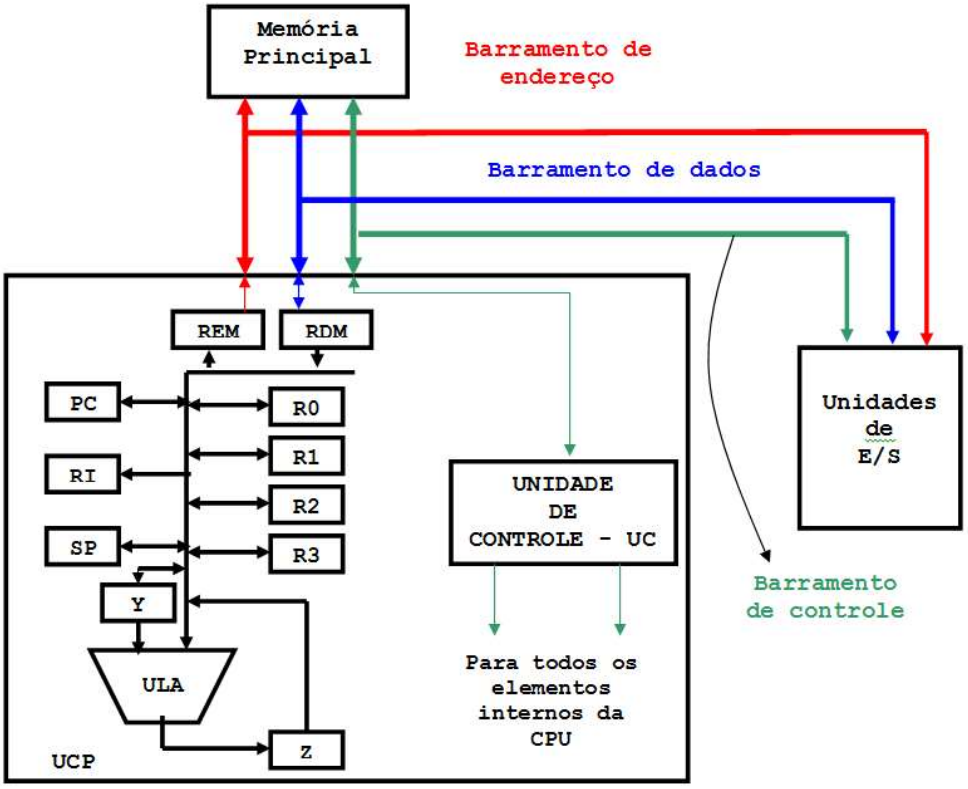


Figura 12.1 – Unidade Central de Processamento - UCP Hipotética.

O conjunto de instruções, desta UCP hipotética, tem dois formatos de instruções. O formato I (figura 12.2) que se refere às instruções com apenas uma palavra de 8 bits, onde o código de operação da instrução (*opcode*) tem 4 bits, os operandos têm dois campos de 2 bits (reg1 e reg2), para definir os registradores que serão manipulados pela instrução.

O formato II (figura 12.3) é utilizado pelas instruções de duas palavras. A primeira palavra de 8 bits tem o código de operação da instrução (*opcode*) com 4 bits, o operando com um campo de 2 bits (reg), para representar o registrador que será manipulado pela instrução,

junto com o outro operando de 8 bits, que é a segunda palavra. Este operando pode ser um número (imediato) ou um endereço de memória.

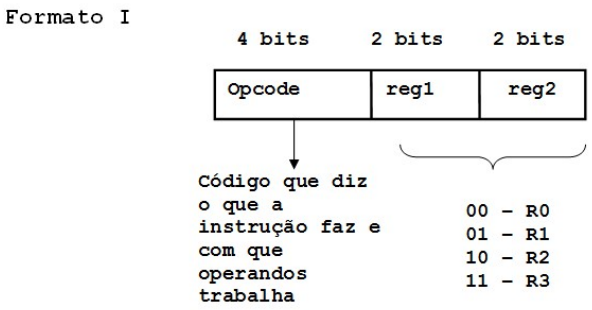


Figura 12.2 – Formato tipo I das Instruções da UCP hipotética.

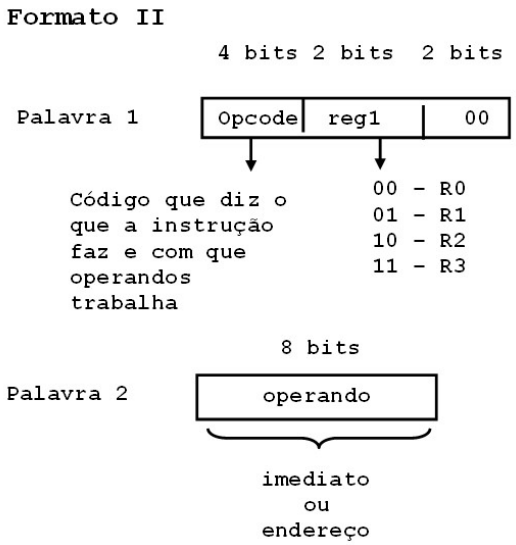


Figura 12.3 – Formato tipo II das Instruções da UCP hipotética.

O conjunto de instruções deste processador consiste em 4 classes de instruções: movimentação de dados, operações lógicas e aritméticas, instruções que manipulam a pilha e de controle de fluxo de execução.

Mnemônico	Operandos	Opcode	Significado	Formato
Instruções de Movimentação de Dados				
MOV	Reg1,Reg2	0000	Reg1 ←Reg2	I
MOV	Reg,imed	1000	Reg ← imed	II
MOV	Reg,[end]	1001	Reg ← [end]	II
MOV	[end],Reg	1010	[end] ← Reg	II
Instruções Aritméticas e Lógicas				
ADD	Reg1,Reg2	0001	Reg1 ← Reg1 + Reg2	I
ADD	Reg,imed	1011	Reg ← Reg + imed	II
SUB	Reg1,Reg2	0010	Reg1 ← Reg1 - Reg2	I
SUB	Reg,imed	1100	Reg ← Reg – imed	II
AND	Reg1,Reg2	0011	Reg1 ← Reg1 *e* Reg2	I
AND	Reg,imed	1101	Reg ← Reg *e* imed	I
OR	Reg1,Reg2	0100	Reg1 ← Reg1 *ou* Reg2	I
Instruções de Manipulação de Pilha				
PUSH	Reg	0101	SP-- , [SP] ← Reg	I
POP	Reg	0110	Reg ←[SP], SP++	I
Instruções de Controle de Fluxo de Execução				
JMP	end	1110	PC ← end	II
CALL	end	1111	SP--,[SP]←PC,PC←end	II
RET	---	0111	PC ←[SP] , SP++	I

Tabela 12.2 – Conjunto de Instruções da UCP Hipotética.

A tabela 12.2 lista o Conjunto de Instruções desta UCP hipotética. A penúltima coluna nos mostra o significado da instrução e a última o seu formato.

O ciclo de execução de uma instrução

Chamamos de ciclo de execução de uma instrução, as etapas que a UCP faz para executar uma instrução em código de máquina. O ciclo é geralmente formado por 4 etapas:

a. Busca da Instrução na Memória (*Fetch* da Instrução)

A busca da instrução na memória principal, chamada de *fetch* da instrução, consiste em ler a posição da memória apontada pelo PC, armazenar a instrução lida no registrador RI e atualizar o conteúdo de PC, para que no próximo ciclo ele possa apontar para a próxima instrução. Esquematicamente, utilizando a nossa UCP hipotética temos:

```
REM ← PC        ; REM recebe o conteúdo de PC
read            ; a UC gera o sinal de leitura
RDM ← [REM]     ; RDM recebe o conteúdo do endereço que está em
                ; REM
PC ← PC + n     ; atualiza PC para que ele aponte para a próxima
                ; palavra, que pode ser o operando desta
                ; instrução (formato II) ou outra instrução
                ; (esta instrução tem o formato I)
```

Como para ler um conteúdo de memória temos que enviar o endereço da posição a ser lida, para a memória. Como é uma instrução que iremos ler, o endereço está armazenado em PC, portanto temos que colocar o valor de PC em REM. Depois a Unidade de Controle envia um sinal de leitura (*read*) para a memória saber que operação será feita (leitura ou escrita). Quando a informação lida vem da memória, ela é armazenada em RDM. Como vimos anteriormente, REM e RDM são os registradores que estão diretamente ligados nos barramentos externos de endereços e dados, respectivamente. Depois o PC é atualizado,

somando-se um número **n** que é o número de bytes da palavra do processador.

b. Decodificação da Instrução.

Após o *fetch* da instrução, ela está armazenada em RDM. Precisamos agora, armazená-la em RI, para podermos decodificá-la. Decodificar uma instrução é saber o que a instrução faz (que operação ela faz) e com que ela faz (quais são seus operandos). Esta decodificação é feita pela Unidade de Controle. Após a decodificação, a UC sabe quais sinais que tem que gerar para executar a instrução. No nosso exemplo, temos os seguintes passos:

```
RI ← RDM        ; RI recebe a instrução lida, que está em RDM
decodificação   ; a UC faz a decodificação da instrução
```

c. Leitura dos Operandos (Busca dos operandos)

Se a instrução tiver um operando que ainda está na memória (no caso de nossa UCP isso só ocorre para as instruções de Formato II), ele terá que ler este operando para poder executá-la. Como operando é a segunda palavra da instrução, ele está armazenado no próximo endereço ao da primeira palavra da instrução. Portanto que aponta para ele é o PC. Na nossa UCP ficaria:

```
REM ← PC        ; REM recebe o endereço do operando que está
                ; em PC
read            ; a UC gera o sinal de leitura
RDM ← [REM]     ; RDM recebe o conteúdo do endereço que está
                ; em REM
PC ← PC + n     ; atualiza PC para que ele aponte para a próxima
```

```
                ; palavra, que pode ser o operando desta
                ; instrução (formato II) ou outra instrução
                ; (esta instrução tem o formato I)
```

Como era de se esperar os passos são os mesmos do *fetch* da instrução, pois ambos são leituras de memória.

d. Execução

Após a decodificação e a busca de operando (se houver necessidade), a UC está pronta para executar a instrução. Executar a instrução significa fazer o que o *opcode* indica (depois da decodificação) com os operandos armazenados na UCP. As etapas da execução naturalmente dependem da instrução.

Exemplos de execução das instruções da UCP hipotética

Mostraremos, esquematicamente, a execução de todas as instruções do conjunto de instruções (tabela 12.2) da nossa UCP hipotética.

- **MOV registrador1,registrador2**

Para esta instrução, vamos supor:

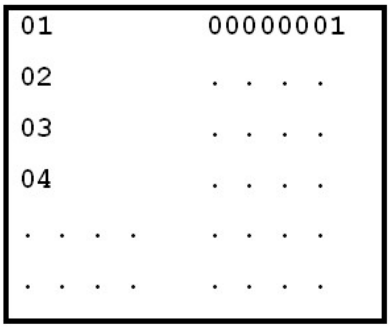
```
                MOV R0,R1       ;R0 ← R1
```

É uma instrução que teria o seguinte formato:

```
                        00000001
```

Os quatros primeiros bits correspondem ao *opcode* da instrução **MOV registrador,número** (ver tabela 12.2), os dois seguintes estariam endereçando o registrador de propósito geral **R0** e os dois últimos o registrador **R1.**

Suponha que a instrução está armazenada na memória da seguinte forma:



Isto significa que a instrução está armazenada no endereço 1. Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
Read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 00000001.
PC ← PC + 1       ; o PC é atualizado com o endereço da próxima
                  ; instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ; para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 0000 (os quatro bits mais significativos) corresponde a instrução de movimentação de

dados, e que o destino é o registrador R0 (os dois bits seguintes), e a origem é o registrador R1.

Busca de Operando

Neste caso, como a instrução é do formato I, não há busca de operando

Execução

```
R0 ← R1      ; O valor armazenado em R1 será copiado para R0.
```

- **MOV registrador,imediato**

Para esta instrução, vamos supor:

```
MOV R0,3          ;R0 ← 3
```

É uma instrução que teria o seguinte formato:

```
10000000
00000011
```

Os quatros primeiros bits correspondem ao *opcode* da instrução **MOV registrador, imediato** (ver tabela 12.2), os dois seguintes estariam endereçando o registrador de propósito geral **R0** e os dois últimos neste caso não são considerados.
Suponha que a instrução está armazenada na memória da seguinte forma:

01	10000000
02	00000011
03	. . . .
04	. . . .
. . . .	. . . .
. . . .	. . . .

Isto significa que a instrução está armazenada no endereço 1 de memória e o operando 3 no endereço 2.
Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
Read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 10000000.
PC ← PC + 1       ; o PC é atualizado com o endereço do operando
                  ; desta instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ;para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 1000 (os quatro bits mais significativos) é da instrução de movimentação de dados, e que o destino é o registrador R0 (os dois bits seguintes), e o operando está armazenado no próximo endereço da memória.

Busca de Operando

```
REM ← PC          ; REM recebe o endereço 2 para leitura.
Read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 2 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 00000011.
PC ← PC + 1       ; o PC é atualizado com o endereço do operando
                  ; desta instrução, neste caso 3.
```

Execução

```
R0 ← RDM          ; O valor 3 foi lido e armazenado em RDM,
                  ;portanto fazemos a transferência para o R0.
```

- **MOV registrador,[endereço]**

Esta é uma instrução de leitura de memória. Para exemplificar, usaremos:

```
MOV R2,[8]        ;R2 ← [8]
```

É uma instrução que teria o seguinte formato:

```
10011000
00001000
```

Os quatros primeiros bits correspondem ao *opcode* da instrução **MOV registrador,[endereço]** (ver tabela 12.2), os dois seguintes estariam endereçando o registrador de propósito geral **R2** e os dois últimos neste caso não são considerados.
Suponha que a instrução está armazenada na memória da seguinte forma:

01	10011000
02	00001000
03	. . . .
. . . .	. . . .
08	00001001
. . . .	. . . .

Isto significa que a instrução está armazenada no endereço 1 de memória e o operando 8 no endereço 2. Diferentemente do exemplo anterior, neste caso o operando é o endereço do número que será armazenado em R2.

Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
Read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 10011000.
PC ← PC + 1       ; o PC é atualizado com o endereço do operando
                  ; desta instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ; para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 1001 (os quatro bits mais significativos) é da instrução de movimentação de dados, e que o destino é o registrador R2 (os dois bits seguintes), e o operando é o endereço do dado que irá ser armazenado no registrador.

Busca de Operando

```
REM ← PC          ; REM recebe o endereço 2 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 2 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
```

```
                    ; 00001000.
PC ← PC + 1         ; o PC é atualizado com o endereço do operando
                    ; desta instrução, neste caso 3.
```

Execução

```
REM ← RDM           ; o valor lido 8 é armazenado em REM
read                ; a UC gera o sinal de leitura
RDM ← [REM]         ; O RDM neste caso recebe o valor 9, que
                    ; é o conteúdo do endereço 8
R2 ← RDM            ; R2 recebe o conteúdo de RDM, neste caso 9.
```

- **MOV [endereço],registrador**

Esta é uma instrução de escrita de memória. Para exemplificar, usaremos:

```
MOV [8],R3          ;[8] ← R3
```

É uma instrução que teria o seguinte formato:

```
10101100
00001000
```

Os quatros primeiros bits correspondem ao *opcode* da instrução **MOV [endereço],registrador** (ver tabela 12.2), os dois seguintes estariam endereçando o registrador de propósito geral **R3** e os dois últimos neste caso não são considerados.
Suponha que a instrução está armazenada na memória da seguinte forma:

01	10101100
02	00001000
03	. . . .
. . . .	. . . .
08	. . . .
. . . .	. . . .

Isto significa que a instrução está armazenada no endereço 1 de memória e o operando 8 no endereço 2.
Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 10101100.
PC ← PC + 1       ; o PC é atualizado com o endereço do operando
                  ; desta instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ; para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 1010 (os quatro bits mais significativos) é da instrução de movimentação de dados, e que o destino é a memória, no endereço 8 que é o operando e a origem é o registrador de propósito geral R3 indicado pelos dois bits que estão depois dos bits de *opcode*.

Busca de Operando

```
REM ← PC          ; REM recebe o endereço 2 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 2 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 00001000.
PC ← PC + 1       ; o PC é atualizado com o endereço da próxima
                  ; instrução, neste caso 3.
```

Execução

```
REM ← RDM         ; O endereço 8 foi lido e
RDM ← R3          ; armazenado em RDM, portanto fazemos a
                  ; transferência para o REM, do valor a ser
                  ; escrito em RDM.
write             ; A UC gera o sinal write para a memória.
[REM] ← RDM       ; A escrita é feita.
```

- **ADD registrador1,registrador2**

Para esta instrução, usaremos o exemplo:

```
ADD R0,R1       ;R0 ← R0 + R1
```

É uma instrução que teria o seguinte formato:

```
00010001
```

Os quatros primeiros bits correspondem ao *opcode* da instrução **ADD registrador,registrador** (ver tabela 12.2), os dois seguintes

estariam endereçando o registrador de propósito geral **R0** e os dois últimos o registrador **R1**.
Suponha que a instrução está armazenada na memória da seguinte forma:

01	00010001
02	. . . .
03	. . . .
04	. . . .
. . . .	. . . .
. . . .	. . . .

Isto significa que a instrução está armazenada no endereço 1.
Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 00010001.
PC ← PC + 1       ; o PC é atualizado com o endereço da próxima
                  ; instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ; para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 0001 (os quatro bits mais significativos) é da instrução de soma, e que o destino é o registrador R0 (os dois bits seguintes), e a origem são os registradores R0 e R1.

Busca de Operando

Neste caso, como a instrução é do formato I, não há busca de operando

Execução

```
Y ← R0          ; O valor armazenado em R0 e transferido para Y.
                ; Y sempre vai armazenar um dos valores que serão
                ; usados pela ULA em suas operações.
Z ← Y + R1      ; O resultado da operação é armazenado em Z.
R0 ← Z          ; O conteúdo de Z é copiado para R0.
```

Este exemplo serve como base para todas as instruções que utilizam a ULA (SUB, AND e OR), além da ADD, em operações entre dois registradores.

- **AND registrador,imediato**

Usaremos a seguinte instrução:

```
AND R0,3        ; R0 ← R0 and 3
```

É uma instrução que teria o seguinte formato:

```
11010000
00000011
```

Os quatros primeiros bits correspondem ao *opcode* da instrução **AND registrador,imediato** (ver tabela 12.2), os dois seguintes estariam endereçando o registrador de propósito geral **R0** e os dois últimos neste caso não são considerados.
Suponha que a instrução está armazenada na memória da seguinte forma:

01	11010000
02	00000011
03	. . . .
04	. . . .
. . . .	. . . .
. . . .	. . . .

Isto significa que a instrução está armazenada no endereço 1 de memória e o operando 3 no endereço 2.

Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 11010000.
PC ← PC + 1       ; o PC é atualizado com o endereço do operando
                  ; desta instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ; para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 1101 (os quatro bits mais significativos) é da instrução lógica (AND), cujos termos são o conteúdo do registrador R0 (os dois bits seguintes), e o operando está armazenado no próximo endereço da memória. O resultado será armazenado no próprio R0.

Busca de Operando

```
REM ← PC          ; REM recebe o endereço 2 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 2 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 00000011.
PC ← PC + 1       ; o PC é atualizado com o endereço da próxima
                  ; instrução, neste caso 3.
```

Execução

```
Y ← R0            ; O valor armazenado em R0 e transferido para Y.
                  ; Y sempre vai armazenar um dos valores que serão
                  ; usados pela ULA em suas operações.
Z ← Y and RDM     ; O outro valor foi lido e está em RDM. O
                  ; resultado da operação é armazenado em Z.
R0 ← Z            ; O conteúdo de Z é copiado para R0.
```

Este exemplo serve como base para todas as instruções que utilizam a ULA (ADD, SUB e OR), além da AND, em operações entre um registrador e um imediato.

- **PUSH registrador**

Para esta instrução, vamos supor:

```
PUSH R0                ; SP ← SP-1 e [SP] ← R0
```

É uma instrução que teria o seguinte formato:

```
01010000
```

Os quatros primeiros bits correspondem ao *opcode* da instrução **PUSH registrador** (ver tabela 12.2), os dois seguintes estariam endereçando o registrador de propósito geral **R0** e os dois últimos não são considerados.
Suponha que a instrução está armazenada na memória da seguinte forma:

01	01010000
02	. . . .
03	. . . .
04	. . . .
. . . .	. . . .
. . . .	. . . .

Isto significa que a instrução está armazenada no endereço 1.

Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 01010000.
PC ← PC + 1       ; o PC é atualizado com o endereço da próxima
                  ; instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ; para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 0101 (os quatro bits mais significativos) é da instrução escrita na pilha, ou seja, uma instrução que escreve na memória, no topo da pilha. Para isto iremos primeiro fazer com que o SP aponte para uma posição livre da pilha (o novo topo), subtraindo 1 posição atual. O valor a ser empilhado é, neste caso, o registrador R0 (os dois bits seguintes).

Busca de Operando

Neste caso, como a instrução é do formato I, não há busca de operando

Execução

```
SP ← SP-1          ; O endereço do topo da pilha é atualizado.
REM ← SP           ; É armazenado em REM, pois será feito uma
                   ; escrita na memória.
RDM ← R0           ; O valor a ser escrito é armazenado em RDM.
write              ; A UC gera o sinal write para a memória.
[REM] ← RDM        ; A escrita é feita.
```

- **POP registrador**

Para esta instrução, vamos supor:

```
POP R1          ; R1← [SP] e SP ← SP+1
```

É uma instrução que teria o seguinte formato:

```
01100100
```

Os quatros primeiros bits correspondem ao *opcode* da instrução POP registrador (ver tabela 12.2), os dois seguintes estariam

endereçando o registrador de propósito geral **R1** e os dois últimos não são considerados.
Suponha que a instrução está armazenada na memória da seguinte forma:

01	01100100
02	. . . .
03	. . . .
04	. . . .
. . . .	. . . .
. . . .	. . . .

Isto significa que a instrução está armazenada no endereço 1.
Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 01100100.
PC ← PC + 1       ; o PC é atualizado com o endereço da próxima
                  ; instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ;para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 0110 (os quatro bits mais significativos) é da instrução leitura na pilha, ou seja, uma instrução que lê da memória, no topo da pilha. O valor a ser lido é, neste caso, armazenado no registrador R1 (os dois bits seguintes).

Depois da leitura é necessário somar 1 ao SP, para que ele aponte para o novo topo.

Busca de Operando

Neste caso, como a instrução é do formato I, não há busca de operando

Execução

```
REM ← SP          ; É armazenado em REM, pois será feito uma
                  ; leitura da memória.
SP ← SP+1         ; O endereço do topo da pilha é atualizado.
read              ; A UC gera o sinal read para a memória.
RDM ←[REM]        ; A leitura é feita.
R1 ← RDM          ; O valor lido é armazenado em R1
```

- **JMP endereço**

Para esta instrução, vamos supor:

```
JMP 8          ; PC ← 8;
```

É uma instrução que teria o seguinte formato:

```
11100000
00001000
```

Os quatros primeiros bits correspondem ao *opcode* da instrução **JMP endereço** (ver tabela 12.2), os demais, da primeira palavra não são considerados. A segunda palavra contém o endereço para onde se quer desviar, ou seja, o endereço da próxima instrução que será executada. Suponha que a instrução está armazenada na memória da seguinte forma:

01	11100000
02	00001000
03	. . . .
04	. . . .
. . . .	. . . .
. . . .	. . . .

Isto significa que a instrução está armazenada no endereço 1 e o operando no endereço 2.

Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 11100000.
PC ← PC + 1       ; o PC é atualizado com o endereço do operando
                  ; desta instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ; para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 1110 (os quatro bits mais significativos) é da instrução desvio, ou seja, uma instrução que alterará o PC com o valor do operando.

Busca de Operando

```
REM ← PC          ; REM recebe o endereço 2 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 2 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 00001000.
PC ← PC + 1       ; o PC é atualizado com o endereço da próxima
                  ; instrução, neste caso 3, embora a
                  ; execução dessa instrução irá modificá-lo.
```

Execução

```
PC ← RDM          ; O endereço a ser armazenado em PC, como já foi
                  ; lido, está em RDM. Portanto temos que copiá-lo
                  ; em PC.
```

- **CALL endereço**

Para esta instrução, vamos supor:

```
CALL 8          ; SP ← SP-1, [SP] ← PC e PC ← 8;
```

É uma instrução que teria o seguinte formato:

```
11110000
00001000
```

Os quatros primeiros bits correspondem ao *opcode* da instrução **CALL endereço** (ver tabela 12.2), os demais, da primeira palavra não são considerados. A segunda palavra contém o endereço para onde se quer desviar, ou seja, o endereço da próxima instrução que será executada, que é a primeira instrução de uma sub-rotina (procedimento ou função). A segunda palavra da instrução é o endereço do procedimento.

Suponha que a instrução está armazenada na memória da seguinte forma:

01	11110000
02	00001000
03	. . . .
04	. . . .
. . . .	. . . .
. . . .	. . . .

Isto significa que a instrução está armazenada no endereço 1 e o operando no endereço 2.

Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 11110000.
PC ← PC + 1       ; o PC é atualizado com o endereço do operando
                  ; desta instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ; para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 1111 (os quatro bits mais significativos) é da instrução desvio, ou seja, uma instrução que alterará o PC com o valor do operando. Mas como temos que armazenar o endereço de retorno (PC atual)

Busca de Operando

```
REM ← PC          ; REM recebe o endereço 2 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 2 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 00001000.
PC ← PC + 1       ; o PC é atualizado com o endereço da próxima
                  ; esta instrução, neste caso 3 embora a
                  ; execução dessa instrução irá modificá-lo.
```

Execução

```
Y ← RDM           ; Armazenamos, temporariamente o endereço lido.
SP ← SP-1         ; O endereço do topo da pilha é atualizado.
REM ← SP          ; É armazenado em REM, pois será feito uma
                  ; escrita na memória.
RDM ← PC          ; O valor de PC vai para RDM.
write             ; A UC gera o sinal write para a memória.
[REM] ← RDM       ; O endereço de retorno é empilhado.
PC ← Y            ; O endereço do procedimento é armazenado em PC.
```

- **RET**

Para esta instrução, vamos supor:

```
RET       ; PC← [SP]; SP ← SP+1
```

É uma instrução que teria o seguinte formato:

```
01110000
```

Os quatros primeiros bits correspondem ao *opcode* da instrução **RET** (ver tabela 12.2), os demais não são considerados.
Suponha que a instrução está armazenada na memória da seguinte forma:

01	01110000
02	. . . .
03	. . . .
04	. . . .
. . . .	. . . .
. . . .	. . . .

Isto significa que a instrução está armazenada no endereço 1.
Esquematicamente teremos então:

Fetch da Instrução

```
REM ← PC          ; REM recebe o endereço 1 para leitura.
read              ; A UC gera o sinal read para a memória.
RDM ← [REM]       ; O conteúdo do endereço 1 é lido e enviado pela
                  ; memória para o RDM. Portanto RDM recebe
                  ; 01110000.
PC ← PC + 1       ; o PC é atualizado com o endereço da próxima
                  ; instrução, neste caso 2.
```

Decodificação

```
RI ← RDM          ; em RDM temos a instrução e armazenamos em RI
                  ;para a decodificação.
```

Na decodificação a UC verifica que o *opcode* 0111 (os quatro bits mais significativos) é da instrução retorno de sub-rotina, ou seja, uma instrução que lê da memória, no topo da pilha o novo valor de PC. Depois da leitura é necessário somar 1 ao SP, para que ele aponte para o novo topo.

Busca de Operando

Neste caso, como a instrução é do formato I, não há busca de operando.

Execução

```
REM ← SP          ; É armazenado em REM, pois será feito uma
                  ; leitura da memória.
SP ← SP+1         ; O endereço do topo da pilha é atualizado.
read              ; A UC gera o sinal read para a memória.
RDM ←[REM]        ; A leitura é feita.
PC ← RDM          ; O valor lido é armazenado em PC
```

12.2 A linguagem de Montagem – *ASSEMBLY*

Como já vimos, a linguagem de montagem é o resultado da tradução de uma linguagem de alto nível, antes de se gerar o código executável (linguagem de máquina). Cada instrução em linguagem de montagem corresponde apenas uma instrução em linguagem de máquina, sendo que cada instrução em linguagem de máquina corresponde a um ou a mais bytes.

Como a programação em linguagem de máquina é tediosa e suscetível a erros, pois só trabalhamos com 0s e 1s, fica mais fácil trabalhar com a linguagem de montagem.

Poderíamos nos questionar o porquê de utilizar a linguagem de montagem se temos as linguagens de alto nível. Algumas razões são:

- Podemos desenvolver rotinas mais eficientes que as traduzidas pelos compiladores.
- As operações de escrita e leitura em posições específicas de memória e em portas de E/S são mais simples de se fazer.

- As partes críticas de um programa podem ser re-escritas em Linguagem de Montagem para torná-las mais eficientes.
- Com a Linguagem Montagem temos a noção de como o computador executa suas tarefas.

Linguagem de montagem: repertório de instruções

As instruções de uma Linguagem de Montagem podem ser agrupadas em:

- Instruções de entrada e saída: São as instruções de leitura e escrita, tanto em memória como em unidades de E/S, que em alguns processadores são classificadas como de movimentação de dados.
- Instruções de movimentação de dados: São instruções que permitem com que os dados possam ser copiados de um local para outro, ou simplesmente trocados de local.
- Instruções aritméticas e lógicas: São as instruções que permitem com que as operações aritméticas (soma, subtração, multiplicação e divisão) e lógicas (AND, OR, NOT e XOR) sejam executadas.
- Instruções de desvios: São instruções que permitem a construção de estruturas comumente utilizadas em linguagem de alto nível, como os *loops*, os desvios incondicionais, os desvios condicionais (se maior, se menor, se igual, se maior ou igual, se menor ou igual, e outros) e as chamadas de procedimentos.
- Instruções de deslocamento e rotação: São instruções que permitem a manipulação dos bits de uma palavra de dados.

Observações:

1. Além das instruções, temos também as pseudo-instruções ou macros, que são implementações simples de funções

específicas. No processo de montagem, estas pseudo-instruções que são traduzidas pelo conjunto de instruções que a compõe.

2. As diretivas são instruções para os montadores, não gerando linguagem de máquina para o processador.

Elementos básicos da linguagem de montagem.

Os elementos necessários para escrever um programa em linguagem de montagem são:

- Rótulo (*Label*): é um símbolo necessário para identificação de endereços no código fonte, usados principalmente para saltos. Devem ser alfanuméricos começando por letras.
- Mnemônico: é o símbolo que especifica o tipo de instrução.
- Operandos: são os registradores, imediatos ou endereços que a instrução irá manipular.
- Comentário: forma de documentar a natureza da idéia codificada.

Exemplo de linhas de programa *assembly* do 80x86:

```
 [Rótulo]        [Mnemônico]     [Operando(s)]          [Comentário]
salto:           MOV             AX,25           ; inicializa AX com 25h
                 ADD             AX,AX           ; AX <-- AX + AX
                 DEC             cont            ; cont é uma variável
                                                 ; cont ← cont - 1
                 JNZ             salto           ; se cont é diferente de
                                                 ; zero, salta para o
                                                 ; rótulo salto
```

Exercícios

Parte 3

1. Mostre a organização básica de um computador digital, descrevendo as unidades que o compõe e suas funções.

2. O que é um sistema hierárquico de memória?

3. Diferencie os tipos de memórias semicondutoras e exemplifique.

4. O que é memória secundária? Exemplifique.

5. Quais são as unidades que compõem a Unidade Central de Processamento? Explique as funções de cada uma delas.

6. Quais os tipos de Unidade de Entrada e Saída e quais as suas diferenças? O que a difere dos Dispositivos de Entrada e Saída?

7. Que tipos de dispositivos existem e como podemos classificá-los? Exemplifique.

8. Explique e esquematize o ciclo de execução de uma instrução.

9. Usando a UCP Hipotética da Figura Exercício 9 mostre a execução de cada instrução do Conjunto de Instruções.

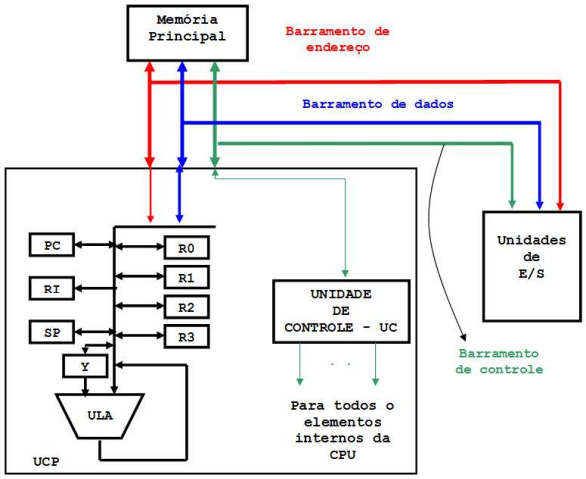


Figura Exercício 9 – Unidade Central de Processamento - UCP Hipotética.