MANUAL DO ALUNO

DISCIPLINA SISTEMAS DIGITAIS E ARQUITETURA DE COMPUTADORES

Módulos 7 e 8

República Democrática de Timor-Leste
Ministério da Educação

FICHA TÉCNICA

TÍTULO
MANUAL DO ALUNO - DISCIPLINA DE SISTEMAS DIGITAIS E ARQUITETURA DE COMPUTADORES
Módulos 7 e 8

AUTOR
BRUNO MORAIS

COLABORAÇÃO DAS EQUIPAS TÉCNICAS TIMORENSES DA DISCIPLINA
XXXXXXX

COLABORAÇÃO TÉCNICA NA REVISÃO

DESIGN E PAGINAÇÃO
UNDESIGN - JOAO PAULO VILHENA
EVOLUA.PT

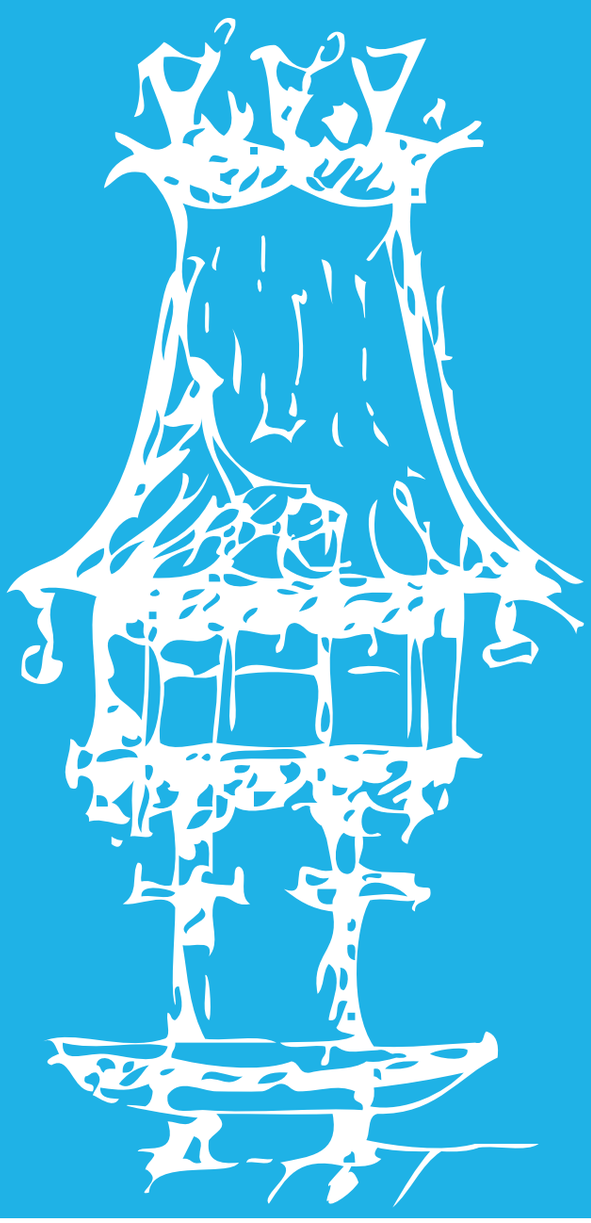
IMPRESSÃO E ACABAMENTO
XXXXXX

ISBN
XXX - XXX - X - XXXXX - X

TIRAGEM
XXXXXXX EXEMPLARES

COORDENAÇÃO GERAL DO PROJETO
MINISTÉRIO DA EDUCAÇÃO DE TIMOR-LESTE
2015

Índice



Arquitetura de Computadores

Módulo 7

Caracterização do Módulo

Apresentação

Neste módulo, é efetuada uma abordagem à forma de organização e funcionamento das arquiteturas de computadores, ao nível da realização e operação interna dos componentes do computador, para o processamento, armazenamento em memória e ações de entrada e saída da informação.

Deverão ser adquiridos conhecimentos teóricos e práticos, estes últimos através da realização de trabalhos laboratoriais sobre a arquitetura de computadores, baseados nos processadores de um PC.

Objetivos de aprendizagem

- Identificar as principais famílias lógicas.
- Conhecer as principais tipos de memória e suas células básicas.
- Avaliar a arquitetura interna de um sistema de um PC.
- Conhecer os diversos tipos de barramentos existentes num PC.
- Conhecer a organização e gestão de memória Principal num PC.

Âmbito de conteúdos

- Principais tipos de memória e identificação das suas células básicas constituintes.
- Introdução histórica aos computadores desde os ábacos e calculadores mecânicos até aos nossos dias. Identificar as principais tendências atuais nomeadamente a nível de comprimento de instruções, estrutura de execução, etc.
- Descrição histórica da evolução do computador PC compatível, salientando as várias evoluções fundamentais desde a placa original até às placas atuais. Identificar quais as principais unidades constituintes e principais evoluções.
- Introdução ao conceito de barramento (*bus*). Descrição e caracterização dos principais tipos de barramentos usados nos PCs.

- Vários tipos de memória usada num PC *(DRAM,SRAM* para as caches, V*RAM e WRAM* para as memórias de vídeo, *EEPROMs*, etc).
- Organização dos bancos de memória de "cache" num PC e comunicação com o PC.
- Organização dos bancos de memória de *DRAM* num PC.
- Evolução histórica da interface vídeo num PC compatível
- Interface com o disco rígido e periféricos.

Famílias Lógicas

Estudamos até o momento as diversas operações lógicas sem nos preocuparmos muito com os elementos utilizados para implementá-las na prática. De fato, ao menor dos circuitos apresentados a título de exemplo, não discutimos com maior profundidade de que maneira e quais os dispositivos utilizados para construímos os circuitos digitais.

As funções lógicas estudadas podem ser implementadas com elementos discretos (relés, díodos, transístores, etc.) ou circuitos integrados (CI's).

Circuitos integrados são dispositivos semicondutores nos quais um único CHIP de silício integra um certo número de resistências, díodos, transístores, etc., número este que pode ser elevadíssimo, ultrapassando a casa dos milhares.

De modo geral, podemos afirmar que os circuitos integrados apresentam, em relação aos com elementos discretos, as seguintes vantagens.

a. Menor tamanho;
b. Menor consumo de potência;
c. Menor custo;
d. Maior velocidade;
e. Maior confiabilidade.

Devido a todas estas vantagens, os circuitos digitais na sua maioria são construídos com CI's.

Família de circuitos integrados

De acordo com a tecnologia de construção, os circuitos integrados são agrupados em famílias, entre as quais podemos citar:

a. RTL- *Resistor Transistor Logic*
b. RCTL- *Resistor Capacitor Transistor Logic*
c. DTL- *Diode Transistor Logic*
d. TTL- *Transistor Transistor Logic*
e. C-Mos- *Complementary Metal Oxide Semiconductor*
f. ECL- *Emmitter Coupled Logic*

As três primeiras são as mais antigas e atualmente já se encontram em desuso. A família ECL é a mais veloz de todas, mas a sua utilização encontra-se restrita a aplicações que requeiram altíssimas velocidades de comutação, devido ao alto custo e às dificuldades que o seu uso apresenta.

Restam assim, as famílias TTL e C-MOS, que são as mais utilizadas atualmente.

Família TTL

Visão geral

A grande maioria dos circuitos integrados TTL pertence às séries 54 e 74, introduzidas originalmente pela *Texas Instruments* e que são hoje um padrão da indústria, fornecidas por diversos fabricantes.

A série 54 é de uso militar e opera na faixa de temperatura de -55°C a +125°C, com uma tensão de alimentação de 5V ±0,5V.

A série 74 é de uso geral, operando na faixa de temperatura de 0°C a +70°C, com alimentação de 5V ± 0,25V.

Há centenas de funções disponíveis nas séries 54/74, abrangendo portas lógicas, flip-flop´s, descodificadores, contadores, etc. Conforme o número de portas presentes num CI, podemos classifica-los como:

- ***SSI (SMALL SCALE INTEGRATION, ou Integração em pequena escala):*** de 1 a 12 portas lógicas;
- ***MSI (MEDIUM SCALE INTEGRATION, ou Integração em média escala):*** de 13 a 99 portas lógicas;
- ***LSI (LARGE SCALE INTEGRATION, ou Integração em grande escala):*** de 100 a 1000 portas lógicas;
- ***VLSI (VERY LARGE SCALE INTEGRATION, ou Integração em escala muito grande):*** acima de 1000 portas lógicas.

Além da série 54/74, que é a mais importante e que possui o maior número de funções disponíveis, existem algumas outras séries como a 4000MTTL da Motorola e a 8200 da Signetics.

Tipos de circuitos integrados TTL

Tendo como referência o circuito de uma porta TTL na versão standart, introduzindo algumas modificações neste circuito, foram desenvolvidas outras versões, todas compatíveis entre si, de modo a obter características melhores em um ou em outro ponto.

Assim, dentro da série 54/74 encontramos cinco versões distintas:

- 54/74 – **STANDART**
- 54/74 L – **LOW POWER** (baixa potência)
- 54/74 H – **HIGH SPEED** (alta velocidade)
- 54/74 S – **SCHOTTKY**
- 54/74 LS – **LOW POWER SCHOTTKY**

A seguir é apresentada uma tabela comparativa com as principais características das 5 versões:

PARÂMETRO	74L	74LS	74	74H	74S
Atraso de propagação (ns)	33	10	10	6	3
Dissipação de potência/Gate (mW)	1	2	10	22	20

Tabela 1:Comparação entre as diferentes versões TTL

A versão standart é a de mais baixo custo e a que possui a maior variedade de funções disponíveis. É importante ressaltar que nem todos os CI´s estão disponíveis em todas as versões.

A versão LOW POWER, indicada pela letra L (por exemplo: 74L00) no nome do CI apresenta o mais baixo consumo de potência, às custas de uma redução de velocidade. Esta versão é indicada nas aplicações onde o baixo consumo seja o fator mais importante, como em equipamentos operados com baterias, e a velocidade requerida não seja muito alta.

A versão HIGH SPEED, indicada pela letra H no nome do CI (por exemplo: 74H00) apresenta uma velocidade maior que a Standart, com um consumo bem mais elevado. Esta versão encontra pouco uso atualmente, sendo substituída pela série S.

A versão SCHOTTKY é a mais veloz de todas. Os circuitos integrados desta versão têm a letra S no seu nome (por exemplo: 74S00); esta versão é indicada em aplicações que se requisitem altas velocidades de comutação.

A versão LOW POWER SCHOTTKY, indicada pelas letras LS no seu nome do CI (por exemplo: 74LS00) é a mais recente de todas e oferece a mesma velocidade da versão standart com um consumo bem menor. Sob este aspeto a versão LS é uma solução quase ótima na maioria dos casos.

A escolha de uma ou outra versão depende dos requisitos particulares de cada circuito. Devem-se ter em conta os fatores de custo, velocidade e consumo de potência, analisando cuidadosamente estes aspetos para se encontrar a solução mais adequada em cada caso.

Interligação dos diversos tipos de TTL

Todas as cinco versões da série 54/74 são compatíveis entre si e operam a partir da mesma alimentação de 5V.

Entretanto, é preciso as características de cada uma para determinar o número máximo de entradas de uma versão que podem ser ligadas à saída de outra.

	74L	74	74LS	74H	74S	Unidade
V_{OL}	**0,3**	**0,4**	**0,5**	**0,4**	**0,5**	**V**
V_{OH}	**2,4**	**2,4**	**2,7**	**2,4**	**2,7**	**V**
I_{OL}	**4,0**	**16,0**	**8,0**	**20,0**	**20,0**	**mA**
I_{OH}	**200**	**400**	**400**	**500**	**1000**	µA
V_{IL}	**0,7**	**0,8**	**0,8**	**0,8**	**0,8**	**V**
V_{IH}	**2,0**	**2,0**	**2,0**	**2,0**	**2,0**	**V**
I_{IL}	**0,2**	**1,6**	**0,4**	**2,0**	**2,0**	**mA**
I_{IH}	**10,0**	**40,0**	**20,0**	**50,0**	**50,0**	µA

Tabela 2: Características das diversas versões da família TTL

Através desta tabela é fácil determinar os FAN-OUT de cada tipo. Lembre-se de que o FANOUT só tem sentido para interligação de CI´s do mesmo tipo.

O FAN-OUT no estado lógico “0” será:

$$I_{OL} / I_{IL}$$

No estado lógico “1” será:

$$I_{OH} / I_{IH}$$

Fazendo os cálculos obtemos a tabela abaixo:

FAN-OUT	74L	74	74LS	74H	74S
ESTADO “0”	22	10	22	10	10
ESTADO “1”	20	10	20	10	20

Tabela 3: FAN-OUT´s para os diversos tipos de TTL.

Família CMOS

A família MOS complementar (CMOS) usa tanto FETs de canal-N quanto canal-P no mesmo circuito, de forma a aproveitar as vantagens de ambas as famílias lógicas. Em geral, os dispositivos de família CMOS são mais rápidos e consomem menos potência que os restantes das famílias MOS. Estas vantagens são contra balanceadas pela extrema complexidade do processo de fabrico dos CI´s CMOS, e pela sua baixa densidade de integração, quando comparados com os MOS. Por tais motivos, o CMOS não pode competir com o MOS em aplicações LSI (*Large Scale Integration*).

No entanto a lógica CMOS tem vindo a crescer de maneira constante nas aplicações MSI (*Medium Scale Integration*), a maioria das vezes destronando a lógica TTL, que é a sua concorrente mais direta. O processo de fabrico dos CMOS é mais simples do que o TTL, possuindo também uma densidade de integração maior, permitindo portanto, que mais circuitos sejam colocados numa mesma área de CHIP, reduzindo em consequência o custo por função lógica. A lógica CMOS só usa uma pequena fração da potência necessária à série TTL com as especificações de consumo mais baixas (série 74L), sendo especialmente apropriada para aplicações que devam ser alimentadas por bateria. Por norma, os dispositivos CMOS são mais lentos do que os TTL, apesar da nova série CMOS de alta velocidade competir em pé de igualdade com as séries TTL 74 e 74LS.

Características das séries CMOS

As séries 4000 e 14000, introduzidas pela RCA e pela Motorola Inc, respetivamente, foram as primeiras séries da família CMOS. A primeira linha de dispositivos 4000 foi lançada com o nome de série 4000B, a qual foi lançada com diversos melhoramentos em relação à série 4000A. O mais importante destes melhoramentos foi a capacidade de suportar correntes mais altas. Ambas as séries são ainda bastante usadas, apesar do aparecimento das novas séries CMOS mais modernas, pelo fato de implementarem diversas funções ainda não disponíveis nas novas séries.

Série 74C

Esta série CMOS é compatível, pino por pino e função por função, com os dispositivos TTL do mesmo número. Muitas das funções disponíveis em TTL estão também disponíveis em CMOS.

Série 74HC (CMOS de alta velocidade)

Esta é uma versão melhorada da série 74C. O principal melhoramento é o tempo de comutação, aproximadamente dez vezes menor que o da série 74C. A velocidade dos dispositivos desta série é comparável com a velocidade dos dispositivos da série TTL 74LS. Outra melhoria na série 74HC é a sua capacidade de suportar altas correntes de saída.

Série 74HCT

Esta também é uma série CMOS de alta velocidade. A principal diferença entre esta série e a 74 HC é o fato de ela ser desenvolvida para ser compatível em termos de tensões com dispositivos da família TTL. Por outras palavras, os dispositivos 74 HCT podem ser alimentados diretamente por saídas de dispositivos TTL. Isto aplica-se às outras séries CMOS.

Tensão de alimentação

As séries 4000 e 74C operam com valores de Vdd na faixa de 3 a 15V, de forma que a regulação da fonte de alimentação não possui um ponto crítico para tais séries. Já as séries 74HC e 74HCT operam com alimentação na faixa de 2 a 6V. Quando dispositivos CMOS e TTL estiverem a ser usados em conjunto, a tensão de alimentação deve ser mantida nos 5V, de modo a que uma única fonte de 5V possa fornecer tanto a tensão Vdd para os dispositivos CMOS que estiverem a operar com fontes de alimentação que fornecem tensões maiores do que 5V, devem ser tomados cuidados especiais para permitir a operação conjunta dos dispositivos CMOS e TTL.

Níveis de tensão

Quando as saídas CMOS só estiverem alimentando entradas CMOS, os níveis de tensão de saída serão muito próximos de 0V para nível baixo e de +Vdd para o nível alto. Isto deve-se à resistência muito alta na entrada dos dispositivos CMOS, que tira pouca corrente da saída CMOS que estiver a alimentar. As especificações para a tensão de entrada nos dois níveis lógicos são expressas como um percentual da tensão de alimentação. A tensão V_{IL} (max.) é 30% de V_{IH} (min) e de 70% de Vdd. Quando um CI CMOS está a trabalhar com Vdd = 5V, este aceitará qualquer tensão de entrada menor do que V_{IL} (max.) = 1,5 V que representa o nível lógico baixo e qualquer tensão acima de VIH (min.) = 3,5V representa o nível alto.

Margens de ruído

As margens de ruído DC para os dispositivos CMOS podem ser determinadas a partir dos dados da tabela 4, as margens de ruído são 1,5V para ambos os níveis lógicos com Vdd= 5V. Isto torna-o a série mais atrativa para aplicações expostas a ruídos.

Potência

A potência para um Vdd de 5V é 2,5nW por porta, é pequena pois existe sempre um Mos-fet a cortar o caminho da corrente. A potência cresce com a frequência devido a

uma corrente transitória que é gerada para carregar a capacitância de carga quando a saída CMOS comuta de baixo para alto.

V_{IL} (max) = 30% Vdd	V_{OL} (max) = 0V
V_{IL} (min) = 70% Vdd	V_{OL} (min) = Vdd
$V_{nh} = V_{oh}(min) - V_{ih}(min)$ = 30% Vdd	$V_{nl} = V_{il}(max) - V_{ol}(max)$ = 30% Vdd

Tabela 4: Níveis de tensão CMOS (série 4000B):

FAN-OUT

As entradas CMOS têm uma resistência extremamente grande por volta de 10 trilhões de ohms que não tira quase corrente nenhuma da fonte de sinal. Cada entrada CMOS apresenta capacitância de 5pF da carga para a terra. Cada carga CMOS acrescida à saída faz com que o atraso de comutação cresça de 3ns. O FAN-OUT vai depender do atraso de propagação máximo aceitável, tipicamente o fan-out é 50 para operações abaixo de 1MHz.

Velocidade de comutação

Apesar do fato de o dispositivo CMOS, tal como o N-MOS e P-MOS ter de alimentar capacitâncias de carga relativamente altas, a sua velocidade de comutação é um pouco mais rápida devido à sua resistência de saída no nível lógico alto e à resistência R_{on} do MOSFET- P cujo valor é 1 KΩ, isto permite um carregamento rápido da capacitância.

Entradas não utilizadas

As entradas de um circuito CMOS não podem nunca ser deixadas desligadas, pois ficariam suscetíveis à ação de ruído ou de cargas estáticas que poderão polarizar facilmente ambos os mosfets, o que aumentaria a potência dissipada e causaria um superaquecimento do circuito.

Cargas estáticas

A alta resistência da entrada de circuitos CMOS torna-os propensos a sofrer a ação de cargas estáticas que podem resultar em tensões grandes o suficiente para romperem o isolamento dielétrico entre a porta e o canal do mosfet. Muitos CMOS tem díodos zeners em cada uma das suas entradas.

	74HC	4000B	74	74S	74LS	74AS	74ALS	ECL
Potencia dissipada por porta (mW)	2,5	1	10	20	2	8	1,2	40
Estática (mW) a 100KHz	0,17	0,1	10	20	2	8	1,2	40
Produto velocidade potencia a 100KHz	1,4	5	90	60	19	13,6	4,8	40
Taxa máxima de clock (Mhz)	40	12	35	12,5	45	200	70	300
Margem de ruído: pior caso (V)	0,9	1,5	0,4	0,3	0,3	0,3	0,4	0,25

Tabela 5: Características das diversas versões da família CMOS.

Questionário

1. Diga que famílias lógicas conhece e quais as que já caíram em desuso.

2. Existem duas series dentro da família lógica TTL. A Serie 54 e a Serie 74. Qual é a diferença entre ambas?

3. Dentro das series 54/74 existem cinco versões. Faça a legenda das mesmas.

   - 54/74 _____
   - 54/74 L _____
   - 54/74 H _____
   - 54/74 S _____
   - 54/74 LS _____

4. Quais são os fatores a ter em conta na escolha de qualquer uma das versões TTL?

5. Todas as cinco versões da série 54/74 são compatíveis entre si e operam a partir da mesma alimentação. Qual é o valor de alimentação?.

6. Quais as principais vantagens dos dispositivos da família CMOS relativamente aos MOS?

7. Qual foi o principal melhoramento da serie 4000B relativamente à serie 4000A?

8. Qual é a principal diferença entre a serie 74HCT e a a serie 74HC?

Evolução da Arquitetura de Computadores

Elementos do sistema computacional

O computador é uma máquina ou dispositivo capaz de executar uma sequência de instruções definidas pelo homem para gerar um determinado resultado, o qual atenda a uma necessidade específica (ex.: realizar cálculos, criar relatórios). Essa sequência de instruções é denominada algoritmo, o qual pode ser definido como um conjunto de regras expressas por uma sequência lógica finita de instruções, que ao serem executadas pelo computador, resolvem um problema específico. Assim, podemos dizer que um ou mais algoritmos compõem o que conhecemos como programa de computador, que no âmbito profissional da área de informática é conhecido como software.

As partes físicas de um computador, tais como: dispositivos de entrada e saída (ex.: monitor, teclado, impressora, webcam), dispositivos de armazenamento (ex. memória volátil e permanente), processador, assim como todo o conjunto de elementos que compõem um computador são chamados de hardware. A Figura 1 apresenta os elementos que compõem o hardware.

Figura 1: Elementos de hardware.

Dessa forma, pode-se dizer que a combinação do *hardware* e do *software* forma o *sistema computacional*.

Computadores analógicos x digitais

Os computadores podem ser classificados em dois tipos principais: analógicos e digitais. Os computadores analógicos não trabalham com números nem com símbolos que representam os números; eles procuram fazer analogia entre quantidades (ex. pesos, quantidade de elementos, níveis de tensão, pressões hidráulicas). Alguns exemplos desse tipo de computador são o Ábaco – que utilizava pequenos carretéis embutidos num pequeno friso de metal para realizar cálculos – ou a régua de cálculo – que utiliza comprimentos de escalas especialmente calibradas para facilitar a multiplicação, a divisão e outras funções.

Pode-se dizer que o computador analógico é uma categoria de computadores em que se utilizam eventos elétricos, mecânicos ou hidráulicos para resolver problemas do homem. Ou seja, tais computadores representam o comportamento de um sistema real utilizando-se para isso grandezas físicas.

Assim podemos dizer que computadores analógicos são normalmente criados para uma finalidade específica, assim como ocorre com a construção de circuitos eletrónicos que implementam sistemas de controlo (ex.: sistemas de segurança, sistemas de controlo de nível). Nesses sistemas, os resultados da computação analógica são utilizados dentro do próprio sistema. Assim, uma pessoa era responsável pela programação e funcionamento desses computadores analógicos, realizando a programação diretamente no *hardware* (ex.: engrenagens, roldanas). No início do século XX as primeiras calculadoras mecânicas, caixas registadoras e máquinas de cálculo em geral foram redesenhadas para utilizar motores elétricos, com a posição das engrenagens representando o estado de uma variável. Exemplos de variáveis utilizadas em computadores analógicos são: a intensidade de uma corrente elétrica numa resistência, o ângulo de giro de uma engrenagem, o nível de água num recipiente.

Diferentemente dos computadores analógicos, que representam números por meio da analogia direta entre quantidades, os computadores digitais resolvem problemas realizando operações diretamente com números, enquanto os analógicos medem. Os computadores digitais resolvem os problemas realizando cálculos e tratando cada número, dígito por dígito. Assim um computador digital é uma máquina projetada para armazenar e manipular informações representadas apenas por algarismos ou dígitos,

que só podem assumir dois valores distintos, 0 e 1, razão pela qual é denominado de computador digital.

Outra grande diferença dessa categoria de computadores é que eles podem resolver problemas através de uma sequência programada de instruções com o mínimo de intervenção humana.

Assim, podemos dizer que o computador digital surgiu como uma solução rápida e com um nível de automação bem mais elevado de realizar grandes computações numéricas. Muitas são as necessidades do homem em termos de computação, especialmente nas áreas de engenharia, além de demonstrações e aplicações teóricas (ex.: cálculo de um fator, progressões aritméticas). Sem o uso da tecnologia, muitos cálculos manuais tornavam-se inviáveis, tanto pelo custo em termos de esforço quanto pelo risco de obter resultados incorretos.

Dessa forma, os computadores digitais foram um passo determinante para o progresso que é possível perceber atualmente em termos de computação. O sonho do homem em realizar cálculos de forma automática, fazendo do computador um dispositivo semelhante ao cérebro humano, mas com capacidades infinitamente maiores do que o ser humano poderia suportar, virou realidade e permite hoje automatizar grande parte das tarefas do ser humano, facilitando sua vida pessoal e profissional.

Evolução tecnológica

Como foi possível perceber na seção anterior, houve uma grande evolução desde o surgimento do computador analógico até o desenvolvimento do computador digital. O que marcou a diferença nessa evolução foram as tecnologias utilizadas na construção de tais computadores, pois, no decorrer dos anos, foram sendo descobertos novos conhecimentos, materiais e dispositivos os quais permitiram a substituição de tecnologias antigas de processamento de informações por novas tecnologias mais eficientes em termos de computação.

Com o surgimento dos primeiros computadores, foi possível classificá-los em gerações, de acordo com as tecnologias utilizadas para a sua produção. A seguir apresentamos as tecnologias utilizadas em cada geração.

Geração zero: computadores mecânicos (1642-1945)

Essa geração foi caracterizada pelos computadores essencialmente analógicos, conforme descritos anteriormente, os quais eram construídos a partir de engrenagens mecânicas e eletromecânicas, operavam em baixa velocidade e eram destinados a resolver problemas específicos. São exemplos dessa geração, além dos já citados anteriormente, o mecanismo de Antikythera, a máquina de Pascal e a máquina das diferenças de Babbage.

Primeira geração: válvulas (1945-1955)

Podemos dizer que a Segunda Guerra Mundial foi o marco do surgimento da computação moderna. Foi nesse contexto que começaram a surgir novas tecnologias mais modernas capazes de substituir os componentes mecânicos utilizados até então nos computadores analógicos, possibilitando o surgimento dos computadores digitais. Esse foi um dos motivos pelos quais os computadores da época ficaram conhecidos como computadores de "primeira geração".

Alguns dos componentes utilizados na produção desses computadores eram os relés, os condensadores e as válvulas, sendo as últimas o mais importante deles. As válvulas possibilitaram cálculos milhares de vezes mais rápidos do que os efetuados com os relés eletromecânicos utilizados inicialmente. A Figura 2 apresenta uma válvula típica.

A entrada de dados e instruções nesses computadores, bem como a sua memória temporária, ocorria frequentemente pela utilização de cartões perfurados. Como os computadores tinham seu funcionamento baseado em válvulas (cuja função básica era controlar o fluxo da corrente, amplificando a tensão que recebe de entrada), normalmente quebravam após algum tempo de uso contínuo em função da queima delas, o que resultava numa falta de confiabilidade, principalmente nos resultados finais. Além de ocupar muito espaço, o seu processamento era lento e o consumo de energia elevado.

Figura 2: Válvula eletrónica

Dentre as primeiras calculadoras e os primeiros computadores (eletrónicos) a utilizarem válvulas, destacamos:

a. ENIAC, na Universidade da Pennsylvania;
b. IBM 603, 604, 701 e SSEC;
c. EDSAC, na Universidade de Cambridge;
d. UNIVAC I, de Eckert e Mauchly.

Algumas das características do *Eletronic Numerical Integrator and Computer* (ENIAC), destacando o efeito do uso de válvulas na construção de computadores (Figura 3):

a. Levou tres anos para ser construído;
b. Funcionava com aproximadamente 19.000 válvulas;
c. Consumia 200 quilowatts;
d. Pesava 30 toneladas;
e. Tinha altura de 5,5m;
f. O seu comprimento era de 25 m;
g. Tinha o tamanho de 150 m².

É possível imaginar a quantidade de energia consumida e o calor produzido por quase 19.000 válvulas?! A finalidade do ENIAC era o cálculo de tabelas de balística para o exército americano. Tratava-se de uma máquina decimal, ou seja, não binária (baseada em 0's e 1's) e a sua programação envolvia a configuração de diversos cabos e interruptores (como é possível observar na Figura 3), podendo levar vários dias.

Figura 3: ENIAC

Modelo de Von Neumann

John Von Neumann foi um matemático natural da Hungria que viveu a maior parte da sua vida nos Estados Unidos. Contribuiu de forma significativa para a evolução dos computadores. As suas contribuições perduram até os dias atuais, sendo que a principal delas foi a construção de um computador sequencial binário de programa armazenado. Podemos dizer que ele propôs os elementos críticos de um sistema computacional, denominado de Modelo de Von Neumann. A arquitetura de computador proposta por esse modelo é composta basicamente por:

a. Uma memória física (para armazenar programas e dados – representados por 0's e 1's);
b. Uma Unidade Aritmética e Lógica (ULA), cuja função é executar operações indicadas pelas instruções de um programa. O seu trabalho é apoiado por diversos registadores (ex.: acumulador);
c. Uma Unidade de Controle (UC), cuja função é buscar um programa na memória, instrução por instrução, e executá-lo sobre os dados de entrada (que também se encontram na memória);
d. Equipamento de entrada e saída.

É importante esclarecer que a ULA e a UC, juntamente com diversos registadores específicos, formam a Unidade Central de Processamento (CPU) do computador.

A Figura 4 apresenta os componentes da arquitetura de Von Neumann descritos acima:

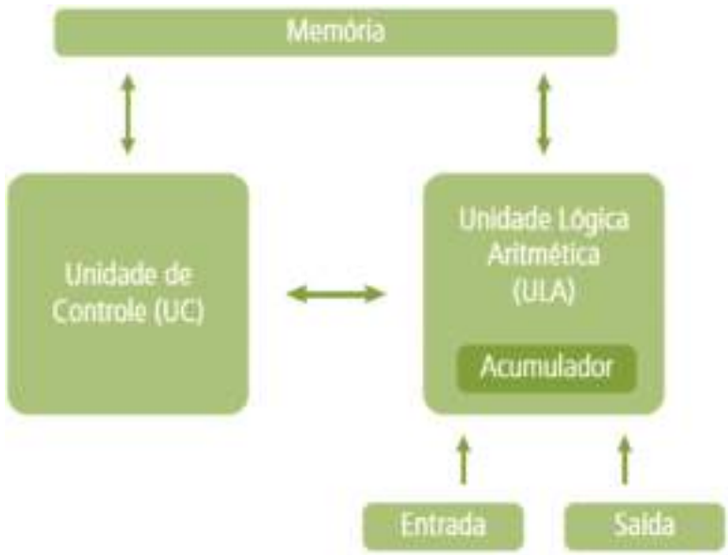


Figura 4: Arquitetura de Von Neumann

Destacamos que a proposta inicial de Von Neumann ainda é utilizada, mas não no seu formato original, pois muitas melhorias foram realizadas visando a obtenção de uma máquina com desempenho cada vez mais elevado, como é o caso das arquiteturas paralelas, que replicam alguns elementos da arquitetura básica de Neumann. Atualmente, muitos pesquisadores têm desenvolvido estudos visando obter uma alternativa a esse padrão, mas ainda não obtiveram sucesso.

Segunda geração: transístores (1955-1965)

Nessa geração, a válvula foi substituída pelo transístor, o qual passou a ser um componente básico na construção de computadores. O transístor foi desenvolvido pelo *Bell Telephones Laboratories em 1948.*

Esse dispositivo reduziu de forma significativa o volume dos computadores e aumentou a sua capacidade de armazenamento. Além disso, o transístor apresentava aquecimento mínimo, baixo consumo de energia e era mais fiável que as válvulas (que queimavam com facilidade). Para termos uma ideia, um transístor apresentava apenas 1/200 (0, 005) do tamanho de uma das primeiras válvulas e consumia menos de 1/100 (0,01) da sua energia.

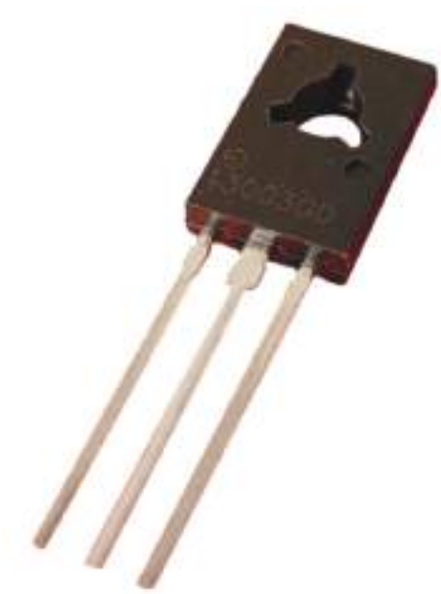


Figura 5: Transístor

A função básica do transístor em circuitos e componentes de um computador é o de um interruptor eletrónico para executar operações lógicas. Existem diversos modelos de transístores, os quais podem possuir características distintas de acordo com a sua aplicação. A Figura 5 apresenta as características físicas de um transístor convencional.

Os materiais utilizados no fabrico do transístor são principalmente: o silício (Si), o germânio (Ge), o gálio (Ga) e alguns óxidos.

Terceira geração: circuitos integrados (1965-1980)

É a partir desta geração que surgem os primeiros circuitos integrados (CI): dispositivos que incorporam inúmeros transístores e outros componentes eletrónicos em formato de miniaturas em um único encapsulamento. Portanto, cada chip é equivalente a inúmeros transístores. Essa tecnologia substituía os transístores, os quais apresentam as seguintes vantagens: maior fiabilidade (não possui partes móveis); muito menores (equipamento mais compacto e mais rápido pela proximidade dos circuitos); baixo consumo de energia (miniaturização dos componentes) e custo de elaboração muito menor. Dessa forma, os computadores passaram a tornar-se mais acessíveis.

A Figura 6 apresenta um circuito integrado.

Figura 6: Circuito Integrado

Distintamente dos computadores das gerações anteriores, a entrada de dados e instruções passaram a ser realizadas por dispositivos de entrada e saída, tais como teclados e monitores. A velocidade do processamento era na ordem de microssegundos. Um dos computadores considerados precursor dessa geração foi o IBM 360, o qual era capaz de realizar 2 milhões de adições por segundo e cerca de 500 mil multiplicações, tornando seus antecessores totalmente obsoletos.

Quarta geração: microprocessadores (1970 - atual)

Há circuitos integrados de diversos tamanhos, tipos e funções, desde os que contêm algumas dezenas de milhares de transístores até circuitos integrados extraordinariamente mais complexos e "inteligentes" – ou seja, capazes de cumprir múltiplas funções de acordo com comandos ou "instruções" a eles fornecidos.

A partir de 1970, as evoluções tecnológicas ocorreram principalmente na miniaturização dos componentes internos dos computadores, entretanto, os avanços ficaram relacionados à escala de integração dos circuitos integrados, ou seja, na quantidade de dispositivos que era possível incluir num único *chip*. A Tabela 6 apresenta as características de cada escala.

Denominação	Complexidade (números de transístores)		
	Interpretação comum	Tanenbaum	Texas Instruments
SSI - Small Scale Integration	10	10	abaixo de 12
MSI - *Medium Scale Integration*	100	10 -100	12 – 99
LSI - *Large Scale Integration*	1000	100 – 100000	100 – 999
VLSI - *Very Large Scale Integration*	10000 – 100000	A partir de 100000	Acima de 1000
ULSI *Ultra Large Scale Integration*	100000 – 1000000	----------	----------
SLSI *Super Large Scale Integration*	1000000 - 10000000	----------	----------

Tabela 6: Escalas de integração

Nesta geração os circuitos passaram a uma larga escala de integração – *Large Scale Integration* (LSI), a partir do desenvolvimento de várias técnicas, e aumentou significativamente o número de componentes num unico *chip*. Como podemos observar na Figura 7, as características físicas de um microprocessador (frente e verso).

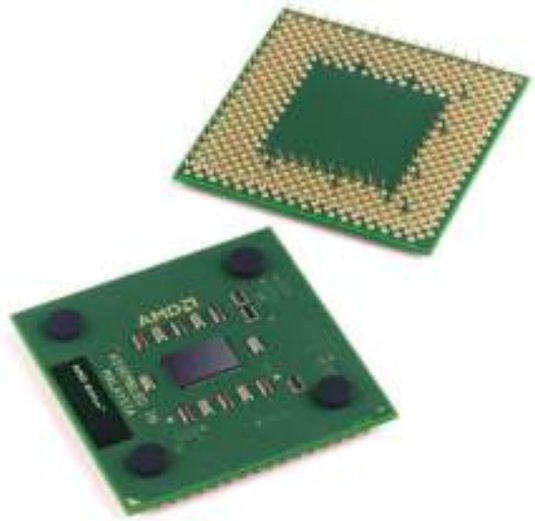


Figura 7: Microprocessador

Em 1970, a INTEL *Corporation* lançou no mercado um tipo novo de circuito integrado: o microprocessador. Os microprocessadores concentravam todos os componentes principais de um computador: a *Central Processing Unit* (CPU) ou Unidade Central de Processamento; controladores de memória e de entrada e saída. Assim, os primeiros computadores ao utilizarem o microprocessador eram denominados "computadores de quarta geração".

O futuro

Atualmente dispomos de computadores extremamente velozes, que apresentam tamanhos cada vez menores; esses avanços são possíveis graças às pesquisas que não cessam, por parte de inúmeras universidades e instituições de pesquisa dispostas a descobrir novas tecnologias. Assim surgiu o que conhecemos por nanotecnologia, ou seja, a capacidade potencial de criar a partir da miniaturização, permitindo, assim, o desenvolvimento de dispositivos miniaturizados para fazer parte de um computador, por exemplo.

Sempre na pesquisa da descoberta de novas tecnologias, neste caso que possibilitem que o computador se torne mais rápido que o seu antecessor (mesmo que o seu antecessor já seja extremamente rápido e possua desempenho além das expectativas dos utilizadores), surgiram novas tecnologias para a construção de computadores não mais baseadas em conceitos digitais (0 e 1) e energia elétrica. Assim, surgiram os computadores óticos: em que feixes de luz poderão cruzar-se num cubo ótico, transportando informação digital.

Os computadores quânticos também estão a ser largamente pesquisados por todo o mundo. Nesse tipo de computador, são os átomos que desempenham o papel dos transístores. Ao contrário dos clássicos *bits* digitais (0 e 1), as menores unidades de informação de um computador quântico podem assumir qualquer valor entre zero e um. Dessa forma, existem previsões bem otimistas de que essa nova tecnologia substitua o silício (matéria-prima dos transístores) em pouco tempo.

Componentes Básicos de um Sistema Computacional

Segundo a arquitetura de Von Neumann, os computadores possuem quatro componentes principais: Unidade Central de Processamento (UCP) – composta pela Unidade Lógica e

Aritmética (ULA) e a Unidade de Controle (UC), a memória e os dispositivos de entrada e saída. Tais componentes são interconectadas por barramentos. E todos esses itens constituem o h*ardware* de um computador (o seu conjunto de componentes físicos), os quais são agrupados em módulos específicos, constituindo a estrutura básica de um computador. A Figura 8 mostra, de forma genérica, essa estrutura.

Cabe esclarecer que quando se fala em processador está-se a falar genericamente da UCP. Muitas pessoas usam a sigla UCP ou CPU para indicar a caixa do computador, o que é erróneo.

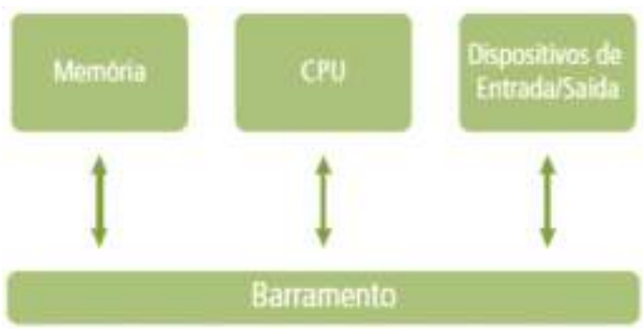


Figura 8: Microprocessador

A seguir serão descritos cada um dos principais componentes de um computador:

a. **UCP:** sigla representativa de Unidade Central de Processamento. Podemos dizer que se trata do componente principal do computador. Algumas pessoas chamam de processador ou microprocessador. É responsável pela execução de dados e instruções armazenadas em memória (código de programas e dados);

b. **Memória:** existem diversos tipos de memória num computador (ex.: RAM (principal), ROM, *cache*, registadores), mas existe uma delas denominada memória principal, a qual é indispensável. A memória principal é tão importante quanto a UCP, pois sem ela não seria possível disponibilizar os programas e os seus dados para o processamento pela CPU. Portanto, a memória é responsável por armazenar todos os programas que executam no computador e os dados que utilizam;

c. Dispositivos de Entrada e Saída (E/S): são dispositivos responsáveis pelas entradas e saídas de dados, ou seja, pelas interações entre o computador e o mundo externo (utilizadores). São exemplos de dispositivos de E/S: monitor de vídeo, teclado, rato, webcam, impressora, entre outros;

d. Barramento: é responsável por interligar todos os componentes listados acima. Trata-se de uma via de comunicação composta por diversos fios ou condutores elétricos por onde circulam os dados manipulados pelo computador.

Questionário

1. Explique com suas palavras no que consiste um sistema computacional.
2. Qual a diferença entre um computador analógico e um computador digital?
3. Os computadores atuais são analógicos ou digitais?
4. O que John Von Neumann significou para a computação?
5. Qual a composição do modelo de Von Neumann e qual a relação desse modelo com os computadores atuais?
6. No que consiste um transístor e qual a sua contribuição para a evolução dos computadores?
7. Qual a composição de um circuito integrado?
8. Em qual das escalas de integração se classificam os microprocessadores?

Barramentos

Os diversos componentes de um computador comunicam-se através de barramentos, os quais se caracterizam como um conjunto de condutores elétricos que interligam os diversos componentes do computador e de circuitos eletrónicos que controlam o fluxo dos *bits*. O barramento conduz de modo sincronizado o fluxo de informações (dados e instruções, endereços e controlos) de um componente para outro ao longo da placa-mãe. O barramento organiza o tráfego de informações observando as necessidades de recursos e as limitações de tempo de cada componente, de forma que não ocorram colisões, ou mesmo, algum componente deixe de ser atendido.

O barramento de um sistema computacional, denominado barramento do sistema, é o caminho por onde trafegam todas as informações dentro do computador. Esse barramento é formado basicamente por três vias específicas: barramento de dados, barramento de endereços e barramento de controlo, conforme mostra a Figura 9.

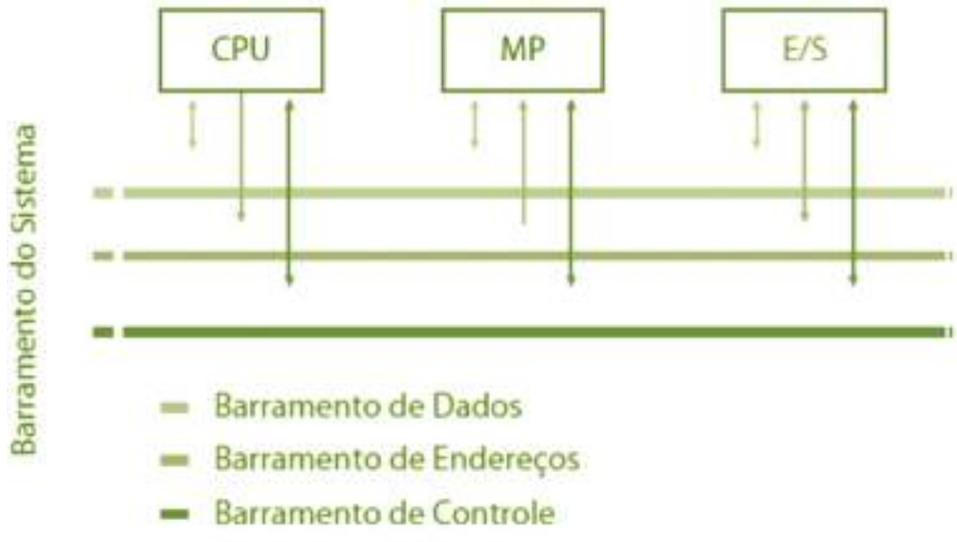


Figura 9: Barramento do sistema

Barramento de dados

Este barramento interliga o RDM - *Registador de dados de memória* (localizado na UCP) à memória principal, para transferência de instruções ou dados a serem executados.

É bidirecional, isto é, ora os sinais percorrem o barramento vindo da UCP para a memória principal (operação de escrita), ora percorrem o caminho inverso (operação de leitura). Possui influência direta no desempenho do sistema, pois, quanto maior a sua largura, maior o número de *bits* (dados) transferidos de cada vez e consequentemente mais rapidamente esses dados chegarão ao seu destino (UCP ou memória).

Os primeiros computadores pessoais (ex.: PC-XT) possuíam barramento de dados de oito vias, ou seja, capaz de transferir oito *bits* de cada vez. Atualmente, conforme a arquitetura do processador, podem existir barramento de dados de 32, 64 ou 128 *bits*.

Barramento de endereços

Interliga o REM ou MAR - *Memory Address Register* (localizado na UCP) à memória principal, para transferência dos *bits* que representam um determinado endereço de memória onde se localiza uma instrução ou dado a ser executado. É unidirecional, visto que somente a UCP aciona a memória principal para a realização de operações de leitura ou escrita. Possui tantas vias de transmissão quantos são os *bits* que representam o valor de um endereço.

Seguindo o exemplo anterior, no antigo PC-XT, este barramento possuía 20 linhas: com isso era possível utilizar endereços de no máximo 20 *bits*. Logo, o maior endereço possível, será:

2^{20}= 1.048.576 *Bytes* = 1 MB

Desta forma, a capacidade de armazenamento da memória RAM poderá ser no máximo 1MB. É possível afirmar que o tamanho do barramento de endereços determina a quantidade máxima de armazenamento de dados que a memória principal pode dispor. Atualmente, os barramentos dispõem de significativa capacidade de armazenamento (ex.: 32,64, 128 *bits*), possibilitando grandes espaços para armazenamento na memória.

Barramento de controlo

Interliga a UCP, mais especificamente a Unidade de Controlo (UC), aos restantes componentes do sistema computacional (memória principal, componentes de entrada e de saída) para passagem de sinais de controlo gerados pelo sistema. São exemplos de

sinais de controlo: leitura e escrita de dados na memória principal, leitura e escrita de componentes de entrada e saída, certificação de transferência de dados – o dispositivo acusa o término da transferência para a UCP, pedido de interrupção, relógio (*clock*) – por onde passam os pulsos de sincronização dos eventos durante o funcionamento do sistema.

É bidirecional, porque a UCP, por exemplo, pode enviar sinais de controlo para a memória principal, como um sinal indicador de que deseja uma operação de leitura ou de escrita, e a memória principal pode enviar sinais do tipo *wait* (espere), para a UCP aguardar o término de uma operação.

Os barramentos partilham as suas vias de comunicação (normalmente fios de cobre) entre diversos componentes neles ligados como mostra a Figura 9. Nesse caso, somente é permitida a passagem de um conjunto de *bits* de cada vez e, por esse motivo, o controlo do barramento torna-se um processo essencial para o funcionamento adequado do sistema.

No modelo apresentado na Figura 9, há um único barramento de dados, endereços e controle interligando todos os componentes do computador. Isso justifica-se pela grande diferença de características dos diversos componentes existentes, principalmente periféricos (ex.: a velocidade de uma transferência de dados de um teclado é muitas vezes menor que a velocidade de transferência de dados de um disco magnético).

Considerando esse fato, os projetistas de sistemas de computação criaram diversos tipos de barramento, apresentando taxas de transferência de bits diferentes e apropriadas às velocidades dos componentes interligados (ex.: UCP, memória, disco rígido, teclado). Nesse caso há uma hierarquia em que os barramentos são organizados e, atualmente os modelos de organização de sistemas de computação adotados pelos fabricantes possuem diferentes tipos de barramentos:

- **Barramento local:** possui maior velocidade de transferência de dados, funcionando normalmente na mesma frequência do relógio do processador. Este barramento costuma interligar o processador aos dispositivos de maior velocidade (visando não atrasar as operações do processador): memória *cache* e memória principal;
- **Barramento do sistema:** podemos dizer que se trata de um barramento opcional, adotado por alguns fabricantes, fazendo com que o barramento local

faça a ligação entre o processador e a memória *cache* e esta se interligue com os módulos de memória principal (RAM) através do chamado barramento do sistema, de modo a não permitir acesso direto do processador à memória principal. Um circuito integrado denominado ponte (*chipset*) sincroniza o aceso entre as memórias;

- **Barramento de expansão:** também chamado de barramento de entrada e de saída (E/S), é responsável por interligar os diversos dispositivos de E/S aos restantes componentes do computador, tais como: monitor de vídeo, impressoras, CD/DVD, etc. Também se utiliza uma ponte para se ligar ao barramento do sistema; as pontes sincronizam as diferentes velocidades dos barramentos.

Considerando a acentuada diferença de velocidade apresentadas pelos diversos dispositivos de E/S existentes atualmente, a indústria de computadores tem desenvolvido alternativas visando maximizar o desempenho nas transferências de dados entre dispositivos (ex.: entre disco e memória principal). A alternativa encontrada foi separar o barramento de expansão (ou de E/S) em dois, sendo um de alta velocidade para dispositivos mais rápidos (ex.: redes, placas gráficas) e outro de menor velocidade para dispositivos mais lentos (ex.: teclado, *modems*, *rato*). A Figura 10 apresenta este conceito.

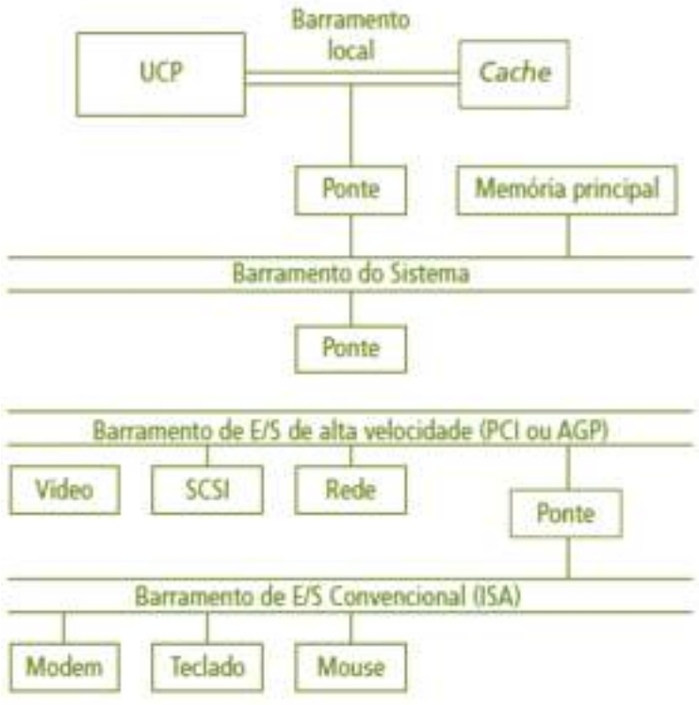


Figura 10: Exemplo de barramento de maior desempenho usado atualmente

Não importa o tipo de barramento, um fator importante que influencia no desempenho do sistema computacional é a largura (tamanho) do barramento, que diz respeito à quantidade de informações (no formato de *bits*) que poderão ser transmitidas simultaneamente por ele. Podemos dizer que é a quantidade de fios ou de vias que um barramento apresenta que vai caracterizar a sua largura.

Quando se fala em quantidade de bits que podem trafegar num barramento, fala-se em taxa de transferência, que revela a medida dessa quantidade, a qual é especificada em bits por segundo (normalmente K *bits*, M bits, etc.).

Conforme discutido anteriormente, cada um dos barramentos permite a sua partilha com os demais componentes do sistema, especialmente o barramento de expansão, que é partilhado com diversos dispositivos de entrada e saída. Para que isso ocorra, é indispensável um mecanismo de controlo de acesso baseado em regras, que garanta que quando um dos dispositivos estiver a utilizar o barramento, os restantes componentes deverão aguardar a sua liberação.

Esse mecanismo de controlo de acesso é denominado protocolo. Isso faz com que um barramento não seja composto somente de fios condutores, mas também de um protocolo.

Os fabricantes de computadores têm procurado uma padronização na definição de protocolos, de forma a evitar que cada um crie o seu próprio, com características diferentes dos demais, o que tornaria muito inflexível o uso de certos componentes (ex.: vídeo, impressoras, discos rígidos) disponibilizados no mercado. Nesse cenário, não importa o fabricante do componente nem as suas características físicas, desde que ele esteja de acordo com os padrões de protocolos definidos pela indústria de computadores. Sendo assim, muitos padrões de barramento de expansão foram desenvolvidos ao longo do tempo, alguns deles já não são mais utilizados. Os mais populares são apresentados a seguir:

- ISA (*Industry Standard Adapter*): desenvolvido pela IBM. Apresenta uma taxa de transferência baixa, mas apesar disso, foi adotado por toda a indústria. Os sistemas atuais não o usam;
- PCI (*Peripheral Component Interconnect*): desenvolvido pela Intel, tornando-se quase um padrão para todo o mercado, como barramento de alta velocidade. Permite transferência de dados em 32 ou 64 *bits* a velocidades de 33 MHz e de 66 MHz. Cada controlador permite cerca de quatro dispositivos;

- USB (*Universal Serial Bus*): tem a característica particular de permitir a ligação de muitos periféricos simultaneamente (pode ligar até 127 dispositivos num barramento – através de uma espécie de centralizador) ao barramento; este, por uma única porta (conetor), liga-se à placa-mãe. Grande parte dos dispositivos USB é desenvolvida com a característica de eles serem ligados ao computador e utilizados logo de seguida, o que é chamado de *plug-and-play*;
- AGP (*Accelerated Graphics Port*): barramento desenvolvido por vários fabricantes, porém liderados pela Intel, com o objetivo de acelerar as transferências de dados do vídeo para a memória principal, especialmente dados em 3D (terceira dimensão), muito utilizados em aplicações gráficas (ex.: jogos);
- PCI *Express* (*Peripheral Component Interconnect Express*): este barramento foi construído por um grupo de empresas denominado PCI-SIG (*Peripheral Component Interconnect Special Interest Group*), composto por empresas como a *Intel*, AMD, IBM, HP e *Microsoft*. Este barramento veio para atender às necessidades por mais velocidade criada pelos novos chips gráficos e tecnologias de rede apresentando altas taxas de transferência. Assim, o PCI e o AGP foram substituídos pelo PCI *Express*. Nesse barramento, a conexão entre dois dispositivos ocorre de modo ponto a ponto (exclusivo) – comunicação serie. Por esse motivo, o PCI *Express* não é considerado um barramento propriamente dito (considerando que barramento é um caminho de dados onde se podem ligar vários dispositivos ao mesmo tempo, partilhando-o). Essa característica é o que faz ser o meio de comunicação entre dois dispositivos de um computador mais rápido atualmente. Até ao momento existiram três versões desse barramento (1.0 – lançado em 2004; 2.0 – lançado em 2007; e o 3.0 – lançado em 2010).

Cada barramento possui um protocolo padrão que é utilizado pela indústria de computadores para a produção de todos os dispositivos de entrada e saída a serem conectados nos diferentes tipos de barramento. Dessa forma, foram desenvolvidos os chamados *slots*. Um *slot* nada mais é do que um orifício ou um encaixe padronizado inserido na placa-mãe dos computadores de maneira a que os diversos dispositivos possam ser encaixados, desde que acolham um dos padrões disponíveis nela.

Assim, nas Figuras 11, 12 e 13 são apresentados alguns slots correspondentes aos padrões de barramentos de expansão listados acima.

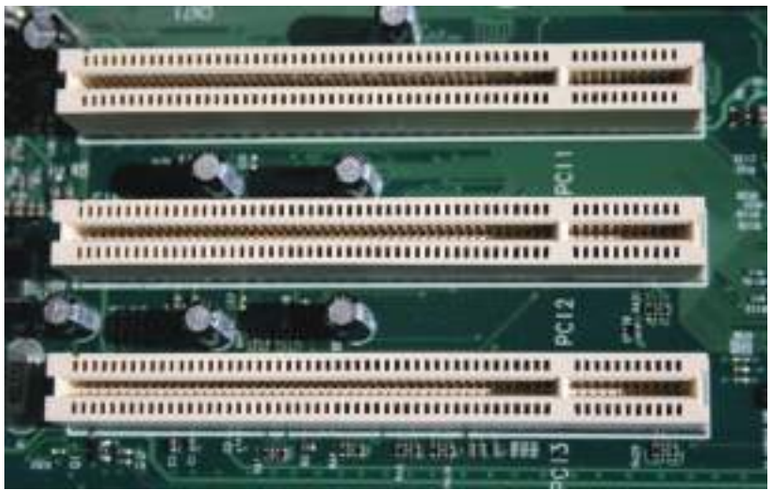


Figura 11: Slot de barramento PCI

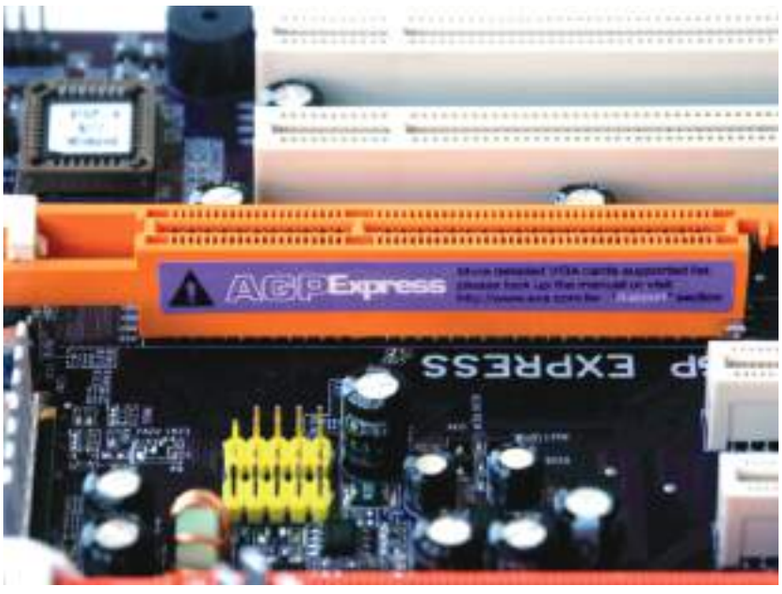


Figura 12: Slot AGP 8x

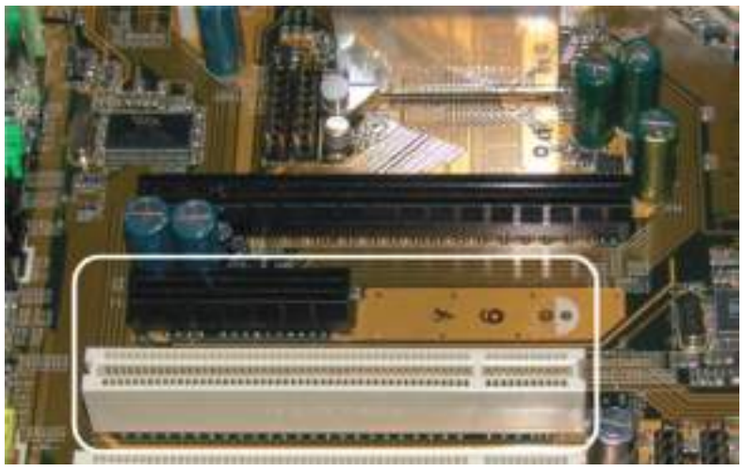


Figura 13: Slots PCI Express 16x (branco) e 4x (preto)

Nesse caso, a principal diferença entre os diversos tipos de barramentos está na quantidade de bits que podem ser transmitidos de cada vez e na frequência de operação utilizada. Os barramentos AGP e PCI Express são considerados os mais rápidos, seguidos pelo barramento PCI original, sendo esses os barramentos mais utilizados em computadores atualmente.

Questionário

1. Diga o que é um barramento.
2. Que tipos de barramentos conhece? Explique pelo menos o funcionamento de um.
3. Que tipos de barramentos de expansão podemos encontrar na arquitetura dos nossos PC's?
4. Faça a legenda das seguintes imagens.

1. 

2. 

5. Qual é o significado das siglas AGP e PCI?
6. Em que é que diferem os vários barramentos, e quais são considerados os mais rápidos?

Subsistema de Memória

Sistema de memória e suas características

De acordo com o modelo de Von Neumann, a função da Unidade Central de Processamento (UCP) ou processador é, essencialmente, capturar dados e instruções que compõem um programa e processá-los, não importando a sua origem ou destino. Mas para que o processador possa executar os programas, os seus dados e instruções devem estar armazenados na memória.

Portanto, a memória dos computadores é um elemento indispensável e tão importante quanto a Unidade Central de Processamento (CPU) ou processador. A memória é um dispositivo que permite ao computador armazenar dados de forma temporária ou permanente. A memória é a parte do computador onde os programas e os dados são armazenados. Sem uma memória na qual os processadores (CPU) possam ler ou escrever informações, o conceito de computador digital com programa armazenado não pode ser implementado.

Para o funcionamento adequado de um computador, é necessário dispor, nele mesmo, de diferentes tipos de memória. Em algumas tarefas, pode ser fundamental que a transferência de dados seja feita da forma mais rápida possível – é o caso das tarefas realizadas pela CPU, onde a velocidade é o fator preponderante, ao passo que a quantidade de *bits* a ser manipulada é muito pequena. Esse tipo de memória deve possuir características diferentes daquele em que a capacidade de armazenamento é mais importante que a sua velocidade de transferência de e para outros dispositivos. Destacamos que a necessidade da existência de vários tipos de memória ocorre em virtude de vários fatores concorrentes, mas principalmente em função do aumento da velocidade das CPUs (a qual é muito maior do que o tempo de acesso da memória) e da capacidade de armazenamento.

Se existisse apenas um tipo de memória, a sua velocidade deveria ser compatível com a da CPU, de modo que esta não ficasse à espera muito tempo por um dado que estivesse a ser transferido. Assim, a CPU manipula um dado em 5 ns, e a memória transfere um dado em 60 ns.

Considerando os diversos tipos de memórias existentes, as quais variam em função de sua tecnologia de fabrico, capacidade de armazenamento, velocidade e custo, pode-se dizer que fica muito difícil projetar um computador utilizando-se apenas um único tipo de memória. Desta forma, o computador possui muitas memórias, as quais se encontram interligadas de forma bem estruturada, constituindo o que é chamado de subsistema de memória, o qual é parte do sistema computacional.

O subsistema de memória é projetado de modo a que os seus componentes sejam organizados hierarquicamente. A Figura 14 apresenta uma pirâmide que mostra a hierarquia das memórias existentes num computador. Observa-se que a base da pirâmide é larga, simbolizando a elevada capacidade de armazenamento, o tempo de uso, a velocidade e o custo de sua tecnologia de construção. Assim, a base da pirâmide representa dispositivos de armazenamento de massa (memória secundária), de baixo custo por *byte* armazenado, mas ao mesmo tempo com baixa velocidade de acesso. A seta na direção do topo indica que quanto mais rápidas forem as memórias, mais elevado será o seu custo em relação à tecnologia e menor a sua capacidade de armazenamento num computador.

Figura 14: Hierarquia de memórias

A Tabela 7 apresenta as características básicas de cada tipo de memória, a seguir.

	Localização	É volátil?	Velocidade	Capacidade de armaze-namento	Custo por bit
Registadores	Processador	Sim	Muito alta (trabalha na velocidade do processa-dor)	Muito baixa (Bytes)	Muito Alto
Cache	Processador	Sim	Alta (traba-lha na velo-cidade do processador)	Baixa (KB)	Alto
Principal	Placa mãe	RAM – sim ROM - não	Depende do tipo de memória instalada	Média (MB)	Medio (tem descido muito)
Secundaria	HD, CD's, etc.	Não	Baixa (lenta)	Alta (GB/TB)	Baixo (tem descido muito)

Tabela 7: Características básicas dos tipos de memória

A seguir apresentamos de forma detalhada cada um dos tipos de memória apresentados na figura 14.

Registadores

A função da memória é a de armazenar dados destinados a serem, em algum momento, utilizados pelo processador. O processador procura dados e instruções de onde estiverem armazenadas e deposita-as temporariamente no seu interior para que possa realizar as operações solicitadas utilizando os restantes componentes (seria análogo à função

memória de uma calculadora). Os dispositivos denominados registadores são os locais onde esse conteúdo fica armazenado.

Assim, o conceito de registador surgiu da necessidade do processador de armazenar temporariamente dados intermediários durante um processamento. Por exemplo, quando um dado resultado de operação precisa de ser armazenado até que o resultado de uma procura da memória esteja disponível para com ele realizar uma nova operação. Os registadores são dispositivos de armazenamento temporário (volátil), localizados no interior do processador (CPU). Por causa da tecnologia utilizada, os registadores são um tipo de memória extremamente rápida e bastante cara. Por esse motivo, a sua disponibilidade em um computador é muito limitada. Cada registador possui capacidade para manter apenas um dado (**uma palavra**).

Em resumo, os registadores, conforme mostra a Figura 14, ficam no topo da pirâmide, o que representa que a sua velocidade de transferência de dados dentro do processador é bastante elevada, em consequência disso, a sua capacidade e armazenamento é baixa e o seu custo é alto.

Memória cache

Considerando a premissa de que o processador precisa procurar dados e instruções na sua memória externa, denominada memória principal, para processá-los e, considerando que a tecnologia desenvolvida para os processadores fez com que se esses dispositivos sejam bem mais rápidos que a memória principal, surgiu a necessidade de diminuir esse atraso gerado pela transferência de dados entre a memória e o processador.

Na busca de soluções para a limitação imposta pela comunicação entre processador e memória, foi desenvolvida uma técnica que consiste na inclusão de um dispositivo de memória entre a memória principal e o processador. Esse dispositivo é denominado memória *cache*. A sua função principal é acelerar a velocidade de transferência das informações entre processador e a memória principal e, com isso, aumentar o desempenho dos sistemas de computação. As memórias *cache* são voláteis, assim como os registadores, pois dependem de energia para manter o seu conteúdo armazenado.

A Figura 15 apresenta um diagrama de blocos de um processador Pentium original, a distribuição da memória cache e a sua relação com a memória principal.

Figura 15: Memória Cache no Pentium

Assim, a memória *cache* é um tipo de memória construída com tecnologias semelhantes às do processador, isso eleva os custos de produção. Para amenizar o fator custo e dispor das vantagens de um sistema computacional com desempenho mais rápido, foram incorporadas ao computador pequenas porções de memória *cache*, localizadas internamente ao processador e entre ele e a memória principal, as quais funcionam como um espelho de parte da memória principal.

Nesse caso, quando o processador solicita um determinado dado e encontra-o na *cache*, não há necessidade de requisitá-lo à memória principal, reduzindo significativamente o tempo de processamento. Ou seja, quanto mais memória *cache* um processador possuir, melhor será o seu desempenho.

A tecnologia de produção da memória cache é SRAM (*Static Random Access Memory*), a qual é bastante diferente das memórias DRAM (*Dynamic Random Access Memory*) – tecnologia da memória principal. A diferença é que nas memórias SRAM não há necessidade de *refresh* ou realimentação constante para que os dados armazenados não sejam perdidos. Isso é possível porque as memórias SRAM utilizam seis transístores (ou quatro transístores e duas resistências) para formar uma célula de memória. Assim, o *refresh* passa a não ser necessário, o que faz com que esse tipo de memória seja mais rápida e consuma menos energia.

Os processadores trabalham, basicamente, com dois tipos de *cache*: *cache* L1 (*Level* 1 ou Nível 1) e *cache* L2 (*Level* 2 ou Nível 2). Normalmente a *cache* L2 é um pouco maior que a L1 e foi implantada quando a *cache* L1 se mostrou insuficiente.

Nas gerações anteriores, a *cache* L1 ficava localizada no interior do processador e a *cache* L2 era externa a ele. Nas gerações de computadores atuais, ambos os tipos ficam localizados dentro do *chip* do processador, sendo que, em muitos casos, a *cache* L1 é dividida em duas partes: "L1 para dados" e "L1 para instruções". De destacar, ainda, que no dependendo da arquitetura do processador, é possível o surgimento de modelos que tenham um terceiro nível de *cache* (L3).

Memória principal

É um tipo de memória indispensável para o funcionamento do computador, à qual o processador pode fazer acesso direto. Além de alocar os dados e instruções de programas a serem manipulados pelo processador, esse tipo de memória dá acesso às memórias secundárias, de forma a disponibilizar dados ao processador.

A memória principal é denominada por memória RAM (*Random Access Memory*), corresponde a um tipo de memória volátil, ou seja, o seu conteúdo fica armazenado enquanto o computador estiver ligado, ao desligar a corrente elétrica, o conteúdo da memória RAM é apagado. Esse é o motivo pelo qual muitas pessoas perdem ficheiros que estão a utilizar quando, por exemplo, alguém toca no cabo ligado à tomada de energia elétrica ou a luz vá a baixo sem aviso prévio. Isso acontece porque ao ocorrerem tais fatos, o ficheiro ainda não tinha sigo guardado em algum tipo de memória permanente (ex.: o disco do computador). A Figura 16 apresenta um pente (módulo) de memória RAM típico.

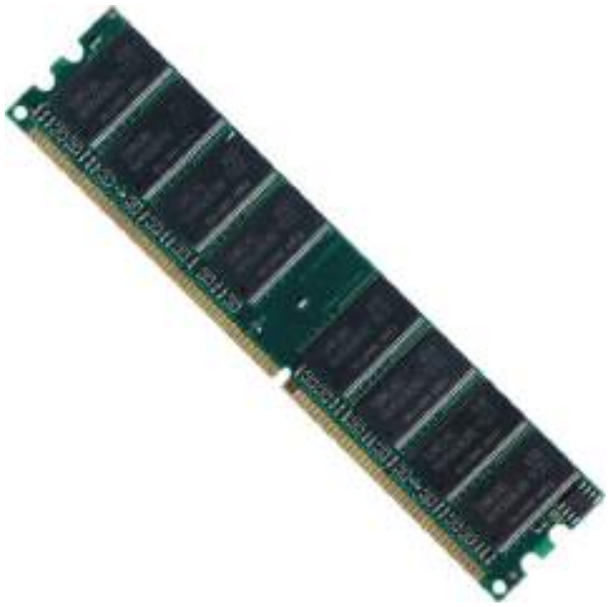


Figura 16: Pente de memória RAM

A memória RAM é denominada genericamente de DRAM (*Dynamic* RAM), ou RAM dinâmica, pelo fato de possuir uma característica chamada refrigeração de memória, que tem a finalidade de manter os dados armazenados enquanto o computador estiver ligado. Essa denominação está ligada à tecnologia utilizada na produção deste tipo de memória, a qual se baseia na utilização de dispositivos semicondutores, mais especificamente condensadores associados a transístores para representar *bits* de dados armazenados. Podemos dizer que é necessário um transístor e um condensador para representar uma célula de memória.

Pelo fato de precisarem ser "refrescadas" ou realimentadas constantemente, as memórias DRAM consomem muitos ciclos do processador para a realimentação, além de consumirem mais energia que outros tipos de memória Por isso, são mais lentas e possuem um custo muito menor e capacidade de armazenamento de dados consideravelmente maior que as memórias estáticas (ex.: *cache*).

Atualmente, podemos contar com muitas opções de padrões de memória RAM, devido à procura constante de uma memória de maior capacidade, maior velocidade de acesso, menor consumo de energia e de tempo de realimentação.

Apresentamos alguns padrões de memória RAM disponíveis mais utilizados atualmente:

a. DDR (*Double Data Rate*): duplicam o desempenho da memória, possibilitando a transferência de dois lotes de dados – entre processador e memória – por ciclo de *clock.*
b. DDR-2: possibilitam a transferência de quatro lotes de dados por ciclo de clock e apresentam menor consumo de energia que a DDR original.
c. DDR-3: transferem oito lotes de dados por ciclo de *clock* e consomem ainda menos energia que as suas versões anteriores.

A memória RAM é comercializada para uso nos computadores no formato de pentes ou módulos de memória, contendo uma determinada quantidade desses recursos. Os pentes de memória podem variar de acordo com as características apresentadas pela memória, especialmente ligadas ao desempenho ou velocidade de transferência. Sendo assim, existem diferentes modelos de módulos de memória disponíveis no mercado. Dentre os que são utilizados atualmente, podem-se citar:

a. Módulos SIMM (*Single In Line Memory Module*): apresentam um pequeno orifício nas linhas de contato. Foram utilizados em memórias FPM e EDO RAM.

Não se encontram disponíveis no mercado atualmente;

b. Módulos DIMM (*Double In Line Memory Module*): não apresentam orifícios nas linhas de contato e apresentam contatos em ambos os lados do módulo. São utilizados atualmente em memórias DDR, DDR2 e DDR3. A Figura 17 apresenta exemplos de módulos DIMM.

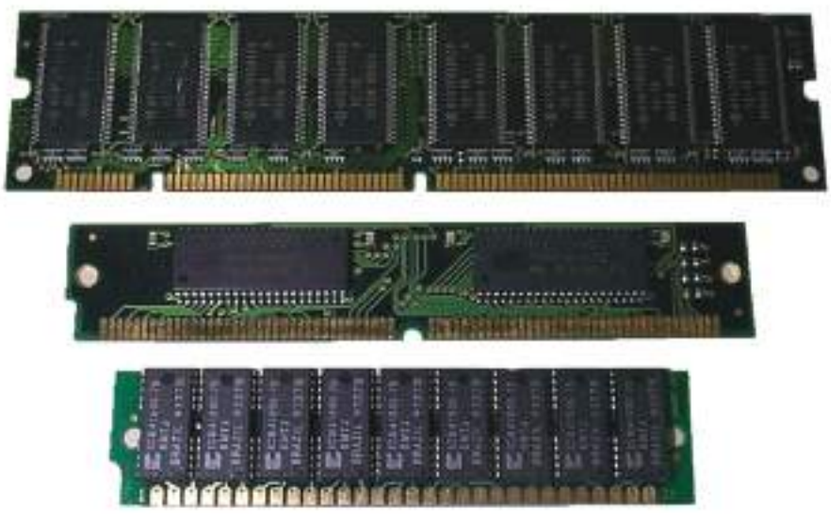


Figura 17: Módulos DIMM

Organização da memória principal

Como a memória principal é o local onde os dados e as instruções de um programa ficam armazenados para serem utilizados pelo processador durante a sua execução, é preciso ficar claro que esse conteúdo precisa estar organizado numa estrutura padrão que permita a identificação do local onde estão armazenados cada um dos seus itens (ex.: uma instrução ou um dado).

Assim, a memória principal encontra-se organizada num conjunto de células, sendo que cada uma delas representa o agrupamento de uma quantidade de *bits*. Cada célula caracteriza uma unidade de armazenamento na memória e possui um endereço único, o qual é utilizado pelo processador para aceder ao seu conteúdo. Portanto, a célula é a menor unidade endereçável em um computador. Podemos dizer ainda que cada célula é constituída de um conjunto de circuitos eletrónicos, baseados em semicondutores, que permitem o armazenamento de valores 0 ou 1, os quais representam um dado ou uma instrução.

A quantidade de *bits* que pode ser armazenada numa célula é definida pelo fabricante. Uma célula contendo N *bits* permite o armazenamento de 2^Ncombinações de valores, o que representará a quantidade de células possíveis na memória. Um tamanho comum de célula adotado pelos fabricantes é 8 *bits* (1 *byte*).

Se for possível armazenar numa memória de 2^N combinações possíveis de células (cada uma delas contendo dados armazenados), então será possível calcular a capacidade de armazenamento da memória principal, da seguinte forma:

- Se N = 9 *bits*, tem-se que 2^9=512 (células de memória);
- Se cada célula pode armazenar 8 *bits*, tem-se que: 512 x 8 = 4KB (4 quilo *byte*) de espaço em memória.

O acesso a cada posição (célula) de memória pode ser feito de modo aleatório, proporcionando grande flexibilidade, graças à sua tecnologia de produção. São duas as operações que podem ser realizadas em uma memória: escrita (*write*) – para o armazenamento de dados na memória – e leitura (*read*) – para a recuperação de dados e instruções armazenados na memória.

O termo palavra também é utilizado ao se tratar de memória de um computador, mas não deve ser confundido com célula, pois palavra é utilizada para definir a unidade de transferência e processamento e o número de *bits* que podem ser transferidos entre a memória principal e a CPU para processamento. Acerca desse tema, podemos afirmar que a memória principal deve ser organizada num conjunto sequencial de palavras, cada uma diretamente acessível pelo processador.

Memória ROM

A memória ROM (*Read Only Memory*) também é considerada uma memória principal, mas apresenta algumas diferenças em relação à memória RAM. A primeira delas é o fato de ser uma memória somente de leitura, ou seja, o seu conteúdo é escrito uma vez e não é mais alterado, apenas consultado. Outra característica das memórias ROM é que elas são do tipo não voláteis, isto é, os dados gravados não são perdidos na ausência de energia elétrica ao dispositivo.

É dito que um *software* que é armazenado numa memória ROM passa a ser chamado de *firmware*. Num computador existem diversos *softwares* desse tipo disponíveis em memórias ROM, pois não podem ser apagados ao desligar o computador e devem ficar disponíveis sempre que for necessário.

Dessa forma, as memórias ROM são aplicadas nos computadores para armazenar três programas principais:

- BIOS (*Basic Input Output System*): ou Sistema Básico de Entrada e Saída, é responsável por indicar ao processador da máquina como operar com os dispositivos básicos de entrada e saída;
- POST (*Power On Self Test*): Autoteste – programa de verificação e teste que se executa após a ligação do computador, realizando diversas ações sobre o *hardware* (ex.: contagem de memória);
- SETUP: Programa que altera os parâmetros armazenados na memória de configuração (CMOS).

As memórias ROM podem ser classificadas em:

- PROM (*Programmable Read-Only Memory*): este é um dos primeiros tipos de memória ROM. A gravação de dados neste tipo é realizada através de aparelhos que trabalham através de uma reação física com elementos elétricos. Uma vez que isso ocorre, os dados gravados na memória PROM não podem ser apagados ou alterados;
- EPROM (*Erasable Programmable Read-Only Memory*): as memórias EPROM têm como principal característica a capacidade de permitir que dados sejam regravados no dispositivo. Isso é feito com o auxílio de um componente que emite luz ultravioleta. Neste processo, os dados gravados precisam ser apagados por completo. Somente após esse procedimento uma nova gravação pode ser realizada;
- EEPROM (*Electrically-Erasable Programmable Read-Only Memory*): este tipo de memória ROM também permite a regravação de dados; no entanto, ao contrário do que acontece com as memórias EPROM, os processos para apagar e gravar dados são feitos eletricamente, fazendo com que não seja necessário mover o dispositivo do seu lugar para um aparelho especial para que a regravação ocorra;

- EAROM (*Electrically-Alterable Programmable Read-Only Memory*): as memórias EAROM podem ser vistas como um tipo de EEPROM. A sua principal característica é o fato de que os dados gravados podem ser alterados aos poucos, razão pela qual esse tipo é geralmente utilizado em aplicações que exigem apenas reescrita parcial de informações;
- *Flash*: as memórias *Flash* também podem ser vistas como um tipo de EEPROM; no entanto, o processo de gravação (e regravação) é muito mais rápido. Além disso, memórias *Flash* são mais duradouras e podem guardar um volume elevado de dados. Trata-se do tipo de memória utilizada nas *pen-drives*;
- CD-ROM, DVD-ROM e afins: essa é uma categoria de discos óticos onde os dados são gravados apenas uma vez, seja de fábrica, como os CDs de músicas, ou com dados próprios do utilizador, quando este efetua a gravação. Há também uma categoria que pode ser comparada ao tipo EEPROM, pois permite a regravação de dados: CD-RW e DVD-RW e afins.

Memória secundária

A memória secundária também é denominada de memória de massa, por possuir uma capacidade de armazenamento muito superior à das outras memórias conforme discutido neste módulo. Outra característica em que difere a memória secundária das outras memórias é o fato de ser permanente (não volátil), ou seja, não perde o conteúdo armazenado caso o computador seja desligado. Por estar na base da pirâmide (Figura 14), apresenta o menor custo por *byte* armazenado.

Este tipo de memória não possui acesso direto pelo processador, há sempre a necessidade do carregamento de dados dos dispositivos de memória secundária para a memória principal, para que então sejam enviados ao processador.

A memória secundária pode ser constituída por diferentes tipos dispositivos, alguns diretamente ligados ao sistema para acesso imediato (ex.: discos rígidos) e outros que podem ser conectados quando desejado (ex.: *pen-drive*, CD/DVD).

Em relação à tecnologia de fabrico, existe uma variedade muito grande de tipos, assim como a variedade de dispositivos que se enquadra nessa categoria de memória. Para

cada dispositivo, existem diferentes tecnologias de produção, não sendo possível aborda-las todas neste módulo.

Questionário

1. Quais são os dois tipos principais de memória? Explique.

2. O que são memórias voláteis e não voláteis? Exemplifique.

3. Qual é a função das memórias principais?

4. O que diferencia uma memória que utiliza a tecnologia DRAM de outra que utiliza a SRAM?

5. Faça uma análise dos diversos tipos de memória em relação aos fatores velocidade e custo.

6. Analise memória *cache* e fale da atuação dela no desempenho do computador.

Bibliografia

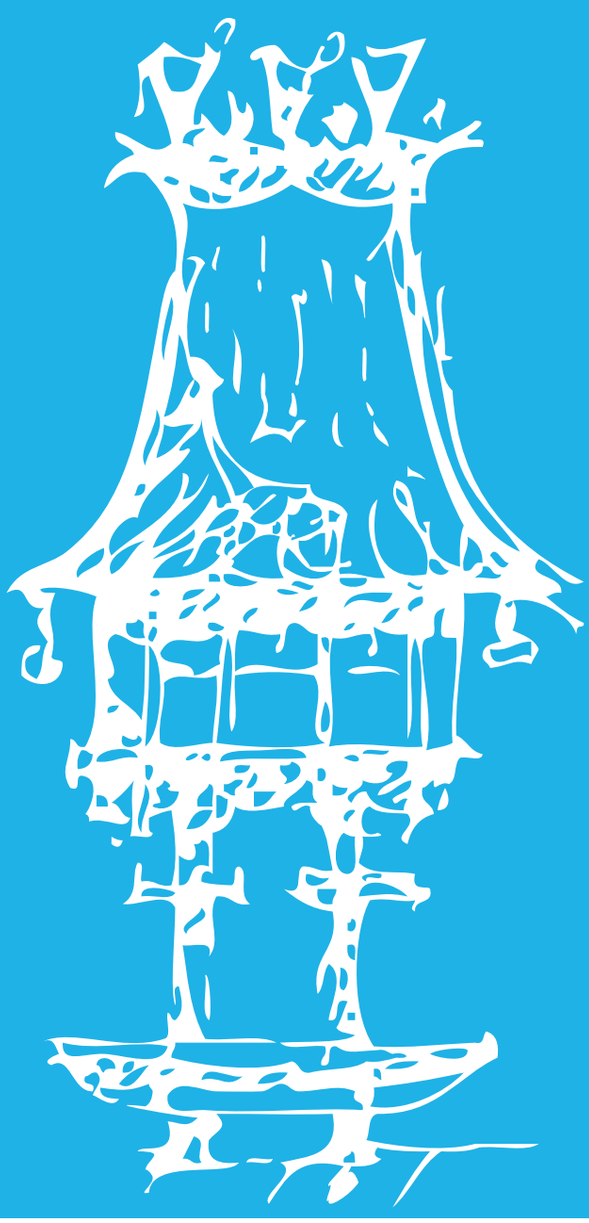
GOUVEIA, José; MAGALHÃES, Alberto, *Curso Técnico de Hardware*. Lisboa: FCA, 2002.

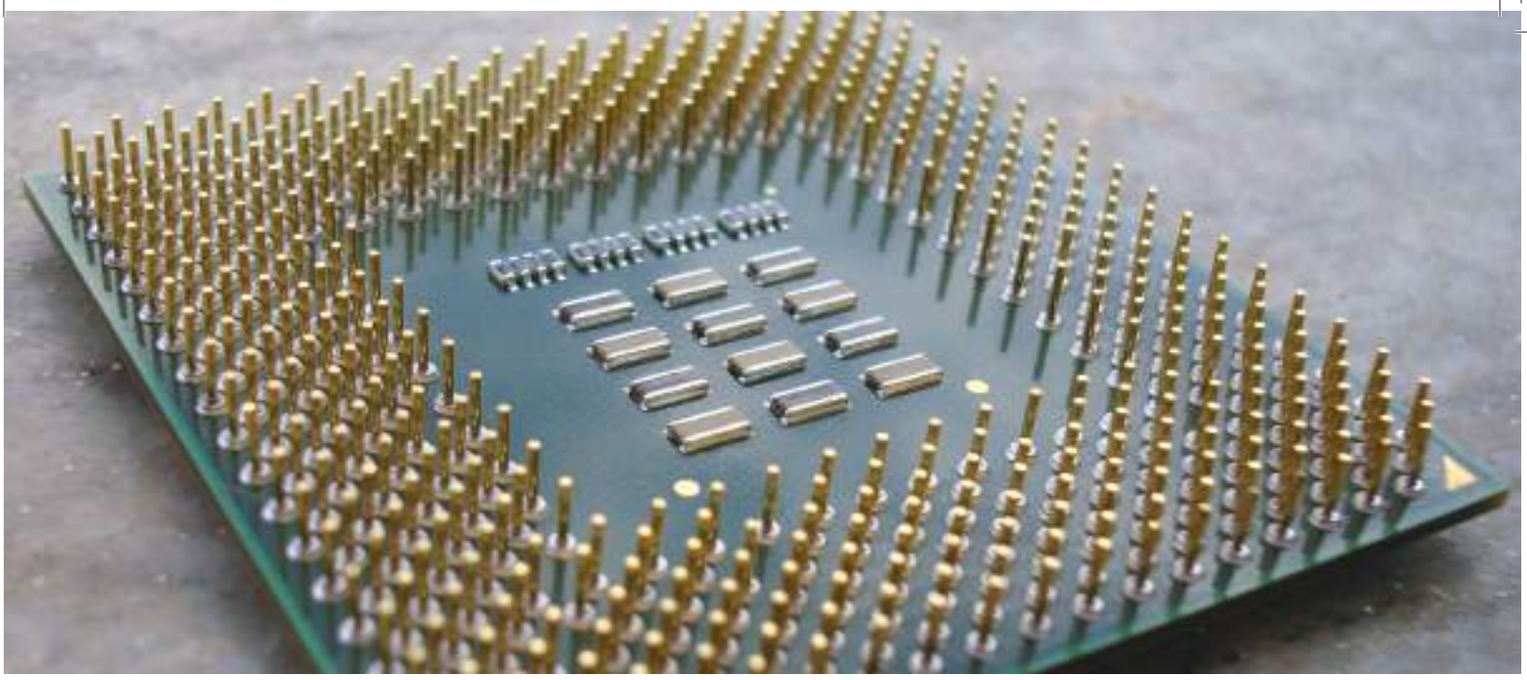
RODRIGUES, Pimenta; ARAÚJO, Mário, *Projecto de Sistemas Digitais*, 2ª ed.. Lisboa: Editorial Presença, sd.

SAMPAIO, A., *Hardware para profissionais.* Lisboa: FCA, sd.

SAMPAIO, A., *Microcomputadores: Circuitos Internos e Programação*. Queluz: Edições EPGE, 1993.

Notas





Arquitetura de Microprocessadores

Módulo 8

Caracterização do Módulo

Apresentação

Pretende-se neste módulo que os alunos adquiram os conhecimentos essenciais e noções sobre a arquitetura de microcomputadores, bem como o desenvolvimento de sistemas com microprocessadores e microcontroladores. Para além disso é abordada a interligação de dispositivos com interface em "*bus*", nomeadamente a ligação de microprocessadores a memórias e periféricos de entrada/saída.

Objetivos de aprendizagem

Conhecer a panorâmica global do mundo dos microprocessadores.

Identificar as principais características de um microprocessador.

Estudar uma arquitetura de um microprocessador.

Estudar o esquema de hardware de um PC, nomeadamente a nível de geração de interrupções, portos de entrada/saída, *Timers*, Geração de Som, Acesso direto aos recursos de imagem do sistema, etc.

Âmbito de conteúdos

Principais componentes de um microprocessador.

Evolução das arquiteturas de microprocessadores.

Arquitetura de um sistema tipo.

Tipos de dados.

Organização de memória.

Tipos de endereçamento.

Ligação com o exterior.

O Processador

Organização do processador

No módulo anterior estudamos os componentes básicos da arquitetura de um computador, segundo o modelo de Von Neumann, cuja proposta vale a pena relembrar no início deste módulo, a qual irá tratar especificamente de um dos elementos dessa arquitetura: o processador. A proposta de Von Neumann para a construção de um computador previa que:

- Codificasse instruções que pudessem ser armazenadas na memória e sugeriu que se usassem cadeias de uns e zeros (binário) para codificá-lo;
- Armazenasse na memória as instruções e todas as informações que fossem necessárias para a execução da tarefa desejada;
- Ao processar o programa, as instruções fossem procuradas diretamente na memória.

A Unidade Central de Processamento (UCP) é responsável pelo processamento e execução de programas armazenados na memória principal, procurando as suas instruções, examinando-as e, então, executando uma após a outra. Cabe lembrar que sempre que houver referências à sigla UCP, estamos a fazer referência ao processador do computador.

A UCP é composta por várias partes distintas, entre elas: registadores, Unidade de Controlo (UC) e Unidade Lógica Aritmética (ULA).

A UCP pode ser dividida em duas categorias funcionais, as quais podem ser chamadas de unidade, conforme segue: *Unidade Funcional de Controlo* e *Unidade Funcional de Processamento*. A Figura 1 apresenta um diagrama funcional básico da UCP, o qual mostra os elementos essenciais para o seu funcionamento.

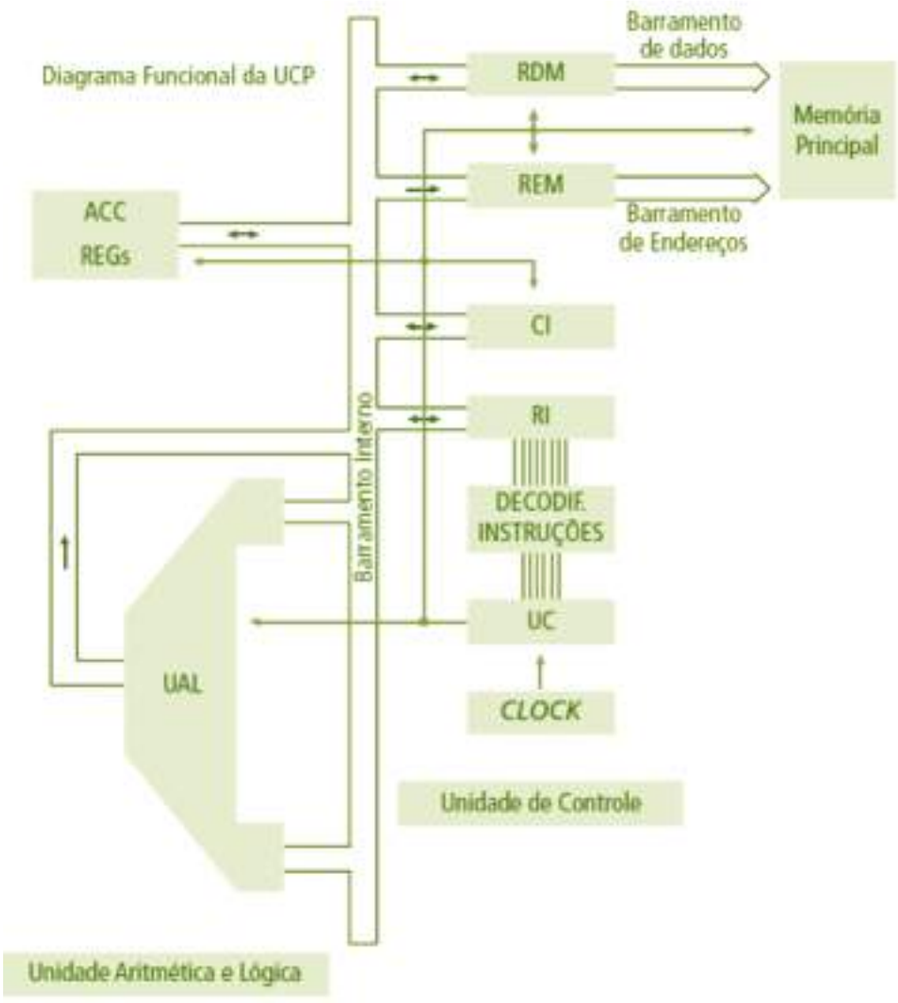


Figura 1: Diagrama funcional da UCP

A Unidade Funcional de Processamento é composta pelos seguintes elementos: Registadores, ACC, ULA. A Unidade Funcional de Controlo é composta pelos seguintes elementos: RDM, REM, CI, RI, Descodificador de Instruções, UC, *Clock* (relógio).

Os componentes do processador são interligados através de um barramento, que consiste num conjunto de fios paralelos que permitem a transmissão de dados, endereços e sinais de controle entre a UCP, memória e dispositivos de entrada e saída. Existem barramentos externos ao processador, cuja função é conectá-lo à memória e aos dispositivos de entrada/saída, além dos barramentos internos à UCP.

A seguir apresentamos de forma detalhada cada um dos elementos que compõem as unidades funcionais da UCP, conforme citados acima.

Questionário

1. Qual a função da UCP?

2. Diga quais são as categorias funcionais da UCP.

3. Como estão ligados os componentes de um processador?

Unidade funcional de processamento

O processamento de dados é a ação de manipular um ou mais valores (dados) em certa sequência de ações, de modo a produzir um resultado útil. Assim podemos dizer que, processar dados é a finalidade do sistema computacional e consiste em executar uma ação, com os dados, que produza algum tipo de resultado. Algumas das tarefas mais comuns da função processamento são: operações aritméticas (somar, subtrair, multiplicar, dividir); operações lógicas (AND, OR, XOR, entre outras) e movimentação de dados entre a UCP e a memória e vice-versa, entre outras.

A seguir serão apresentados os dispositivos que compõem a Unidade Funcional de Processamento (ULA) e registadores, os quais estão em destaque na Figura 2.

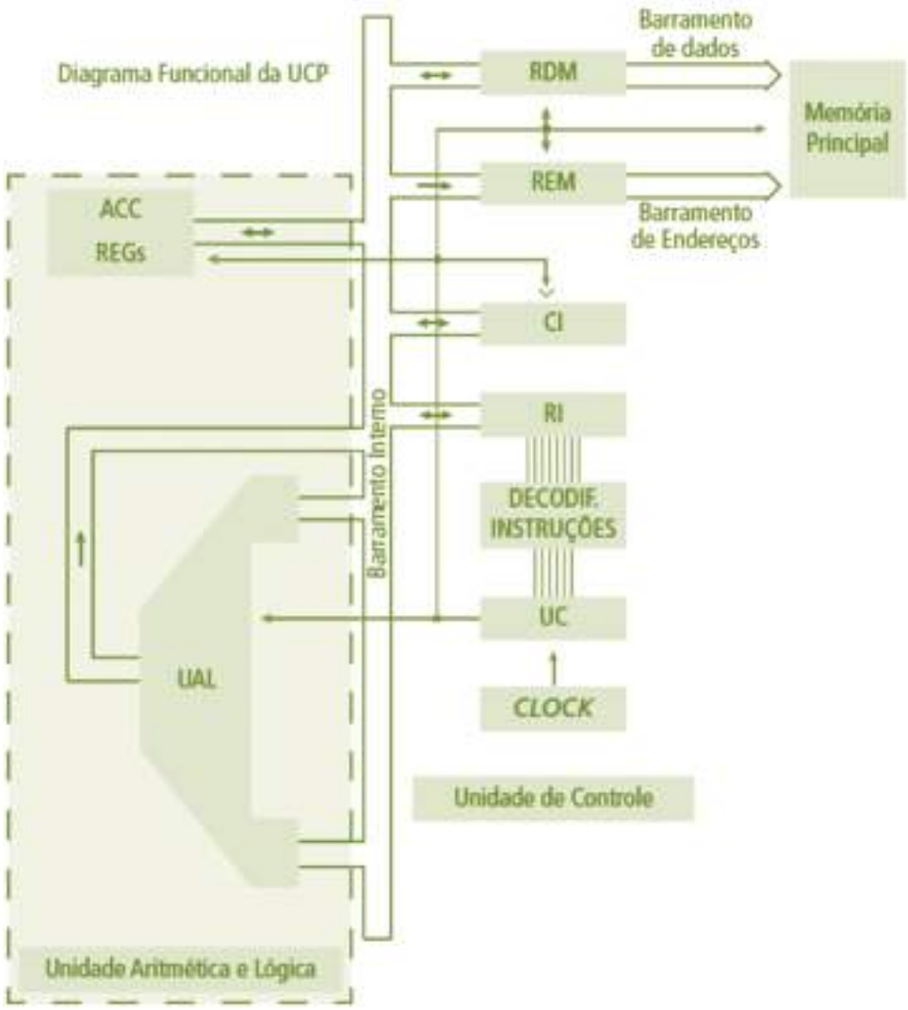


Figura 2: Diagrama funcional da UCP - componentes de processamento

Unidade lógica e aritmética (ULA)

A função efetiva deste dispositivo é a execução das instruções dos programas que se encontram armazenadas na memória. Ao chegarem à UCP, essas instruções são interpretadas e traduzidas em operações matemáticas a serem executadas pela ULA.

Podemos dizer que a ULA é um aglomerado de circuitos lógicos e componentes eletrónicos simples que, integrados, realizam as operações aritméticas e lógicas. São exemplos de operações executadas pela ULA: soma, multiplicação, operações lógicas (AND, OR, NOT, XOR, entre outras), incremento, decremento e operação de complemento.

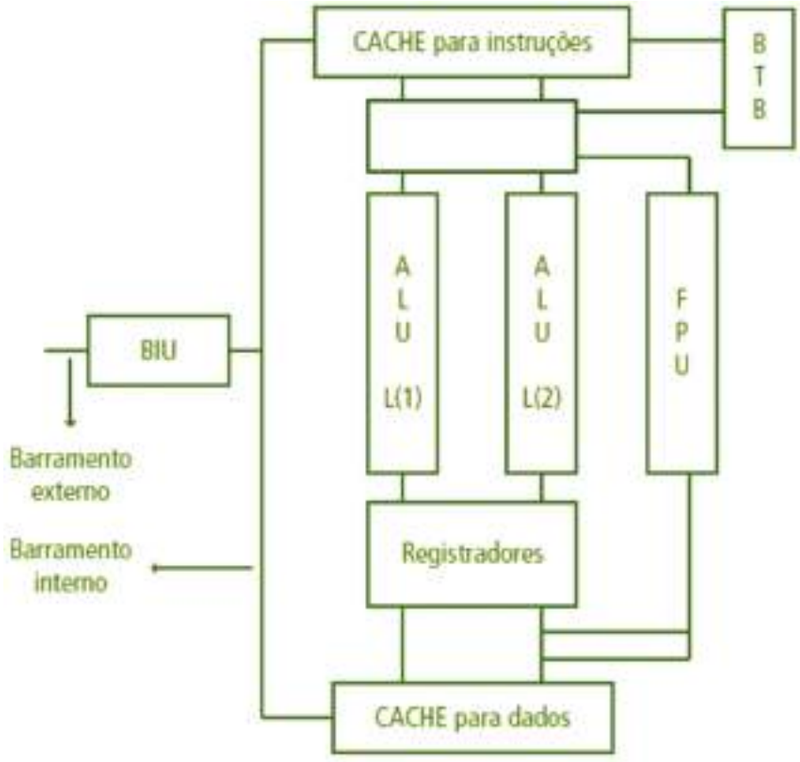


Figura 3: Arquitetura básica da UCP do Pentium original

A ULA é uma pequena parte do circuito integrado da CPU, utilizada em pequenos sistemas, ou pode compreender um considerável conjunto de componentes lógicos de alta velocidade. Isso pode ser constatado em processadores mais modernos, os quais utilizam na sua arquitetura mais de uma ULA, de modo a tornar a execução das instruções mais rápida. Por exemplo: os processadores *Pentium* possuem três ULAs, duas delas para processamento de números inteiros e a terceira para números fracionários, conforme apresenta a Figura 3.

Registadores

São elementos de armazenamento temporário, localizados na UCP, os quais são extremamente rápidos por causa da sua tecnologia de produção (conforme apresentado no módulo anterior). Assim, as UCPs são produzidas com certa quantidade de registadores destinados ao armazenamento de dados que estão a ser utilizados durante o processamento e, portanto, servem de memória auxiliar básica da ULA.

A quantidade e o emprego dos registadores variam bastante de modelo para modelo de processador. Devido à sua tecnologia de construção e por estarem localizados no interior da UCP, são muito caros e, por isso, disponíveis em quantidade limitada.

Os sistemas mais antigos possuíam um registador especial chamado acumulador ou ACC (de *accumulator*), o qual, além de armazenar dados, servia de elemento de ligação entre a ULA e os demais dispositivos da UCP. Nos computadores mais simples é encontrado apenas um acumulador, conforme apresentado na Figura 2. Em arquiteturas mais complexas, vários registadores podem desempenhar as funções de um acumulador, além de haver diversos registadores de dados de uso geral.

Outro fator importante relacionado aos registadores é o tamanho da palavra, a qual está vinculada ao projeto de produção da UCP, correspondendo ao tamanho dos elementos ligados à área de processamento, a ULA e os registadores de dados. A capacidade de processamento de uma UCP, ou seja, a sua velocidade, é bastante influenciada pelo tamanho da palavra. Atualmente há computadores referenciados como tendo uma arquitetura de 32 *bits* ou uma arquitetura de 64 *bits*, o que corresponde ao tamanho da sua palavra.

Questionário:

1. Quais os elementos constituintes da Unidade Funcional de Processamento

2. Das seguintes opções escolha a que ache mais correta.
   a. Podemos dizer que a ULA é um aglomerado de circuitos lógicos e componentes eletrónicos simples que, integrados, realizam as operações aritméticas e lógicas.
   b. Podemos dizer que a ULA é um aglomerado de circuitos lógicos e componentes eletrónicos simples que, integrados, realizam as operações lógicas.
   c. Podemos dizer que a ULA é um aglomerado de circuitos lógicos e componentes eletrónicos simples que, integrados, realizam as operações aritméticas.
   d. Nenhuma das anteriores

3. Os processadores PENTIUM originais possuíam três ULA's. Para que servia cada uma delas?

4. Para que servem os registadores?
   a. São elementos de armazenamento definitivo
   b. São elementos de armazenamento temporário
   c. São elementos de processamento temporário
   d. Nenhuma das anteriores

5. A quantidade e o emprego dos registadores variam bastante de modelo para modelo de processador? Justifique.

Unidade funcional de controlo

Conforme apresentado no início deste módulo, a Unidade Central de Processamento (UCP) é responsável pelo processamento e execução de programas armazenados na memória principal, sendo que a ULA é o elemento da UCP responsável pela execução das operações solicitadas. Dessa forma, a Unidade Funcional de Controlo é responsável pela realização das seguintes atividades:

- Procura a instrução que será executada, armazenando-a num registador da UCP;
- Interpretação das instruções a fim de saber quais operações deverão ser executadas pela ULA (ex.: soma, subtração, comparação) e como realizá-las;
- Geração de sinais de controlo apropriados para a ativação das atividades necessárias à execução propriamente dita da instrução identificada. Esses sinais de controlo são enviados aos diversos componentes do sistema, sejam eles internos à UCP (ex.: a ULA) ou externos (ex.: memória e dispositivos de entrada e saída).

A seguir apresentamos os dispositivos que compõem a Unidade Funcional de Controlo – RDM, REM, CI, RI, Descodificador de Instruções, UC, *Clock*, os quais estão em destaque na Figura 4.

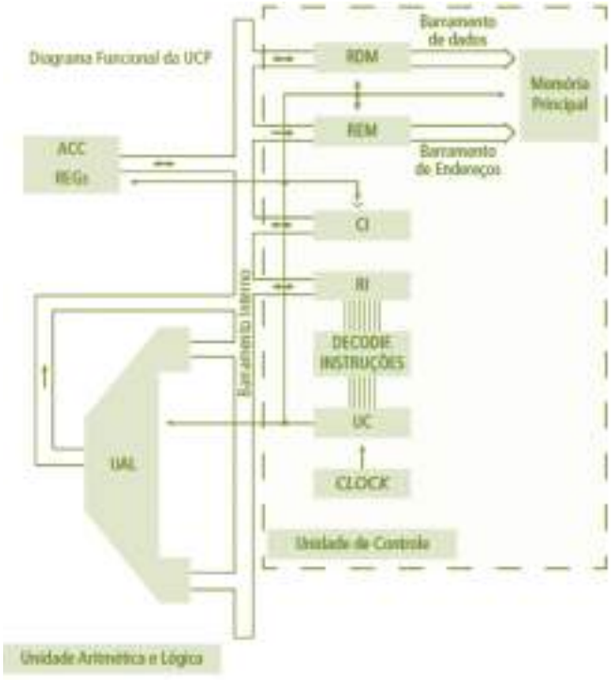


Figura 4: Diagrama funcional da UCP - componentes de controlo

Registador de dados de memória (RDM) e de endereços de memória (REM)

O RDM, também chamado de MBR (*Memory Buffer Register*), é um registador que armazena temporariamente dados (conteúdo de uma ou mais células) que estão a ser transferidos da memória principal para a UCP (numa operação de leitura) ou da UCP para a memória principal (numa operação de escrita). Em seguida, o referido dado é reencaminhado para outro elemento da UCP para processamento ou para uma célula da memória principal, se for um resultado de um processamento. A quantidade de *bits* que pode ser armazenada no RDM é a mesma quantidade suportada pelo barramento de dados.

O REM, também chamado de MAR (*Memory Address Register*), é um registador que armazena temporariamente o endereço de acesso a uma posição de memória, necessário ao se iniciar uma operação de leitura ou de escrita. Em seguida, o referido endereço é encaminhado à controladora da memória, principal identificação e localização da célula desejada. Permite armazenar a mesma quantidade de bits do barramento de endereço.

Contador de instruções (CI)

Este registador é também denominado de *Program Counter* (PC) ou contador de programa. É o CI cujo valor aponta para a próxima instrução a ser procurada da memória a ser executada no processador. Assim, assim que a instrução a ser executada seja procurada da memória principal para a CPU, o sistema automaticamente efetua a modificação do conteúdo do CI de modo a que ele passe a armazenar o endereço da próxima instrução na sequência. Assim, o CI é um registador crucial para o processo de controlo e de sequenciamento da execução dos programas.

Registador de instruções (RI)

Este registador tem a função de armazenar a instrução a ser executada pela UCP. Ao se iniciar um ciclo de instrução, a UC emite sinais de controlo em sequência no tempo, de modo que se processe a realização de um ciclo de leitura para procurar a instrução na

memória. No final do ciclo de leitura a instrução desejada será armazenada no RI, via barramento de dados e RDM. A Figura 5 mostra o RI ligado diretamente ao descodificador de instruções, o qual irá interpretar a instrução e avisar à Unidade de Controlo (UC).

Descodificador de instruções

Cada instrução é uma ordem para que a UCP realize uma determinada operação. Como são muitas instruções, é necessário que cada uma possua uma identificação própria e única, e é função do descodificador de instrução identificar que operação será realizada, correlacionada à instrução cujo código de operação foi descodificado. Assim, o RI irá passar ao descodificador uma sequência de *bits* representando uma instrução a ser executada.

Figura 5: Diagrama de blocos da descodificação numa UCP

Um descodificador possui 2^n saídas, sendo n a quantidade de algarismos binários do valor de entrada. A Figura 5 mostra um diagrama de blocos do processo de descodificação na UCP, no qual o RI passa um código de instrução ao descodificador de tamanho de *4 bits*, que é descodificado (interpretado) e encaminhado à UC para que ela emita os sinais de controlo para os demais elementos da UCP.

O componente descodificador foi incorporado à UCP com o aparecimento das máquinas CISC (Complex Instruction Set Computer) e trata-se de uma categoria de arquitetura de processadores que favorece um conjunto simples e pequeno de instruções de máquinas. Uma instrução de máquina é uma operação básica que o hardware realiza diretamente.

Registador de instruções (RI)

Esses registadores têm como função controlar a execução das instruções e os demais componentes da UCP. Dispositivo que possui a lógica necessária para realizar a movimentação de dados e de instruções da/para a CPU, através de sinais de controlo que emite em instantes de tempo programados.

Os sinais de controlo ocorrem em vários instantes durante o período de realização de um ciclo de instrução e, de modo geral, todos possuem uma duração fixa e igual, originada num gerador de sinais denominado relógio (*clock*).

Relógio (clock)

Podemos definir este dispositivo como um gerador de pulsos, cuja duração é chamada de *ciclo*, e a quantidade de vezes que esse pulso básico se repete num segundo define a unidade de medida do relógio, denominada frequência, a qual também é usada para definir a velocidade na CPU.

A unidade de medida utilizada para a frequência do relógio da UCP é o *hertz* (Hz), que significa um ciclo por segundo.

A cada pulso é realizada uma operação elementar, durante o ciclo de uma instrução (ex.: procura de dados, envio da instrução para o RI, sinal de controlo).

Como os computadores atuais apresentam frequências bastante elevadas, utiliza-se a medida de milhões de ciclos por segundo (mega-*hertz* – MHz) ou bilhões de ciclos por segundo (giga-*hertz* – GHz).

Questionário

1. A unidade funcional de controlo é responsável por três tipos de operações. Diga pelo menos duas delas.

2. Como visto nas durante as aulas, a unidade duncional de controlo é composta por diversos dispositivos. Quais?

3. Qual o significado das siglas RDM e REM? Explique sucintamente a função de ambas.

4. Porque é necessária a utilização de um descodificador de instruções?

5. Qual é a finalidade do clock?

Barramentos

Os diversos componentes de um computador comunicam-se através de barramentos, os quais se caracterizam como um conjunto de condutores elétricos que interligam os diversos componentes do computador e de circuitos eletrónicos que controlam o fluxo dos *bits*. O barramento conduz de modo sincronizado o fluxo de informações (dados e instruções, endereços e controlos) de um componente para outro ao longo da placa-mãe. O barramento organiza o tráfego de informações observando as necessidades de recursos e as limitações de tempo de cada componente, de forma que não ocorram colisões, ou mesmo, algum componente deixe de ser atendido.

O barramento de um sistema computacional, denominado barramento do sistema, é o caminho por onde trafegam todas as informações dentro do computador. Esse barramento é formado basicamente por três vias específicas: barramento de dados, barramento de endereços e barramento de controlo, conforme mostra a Figura 6.

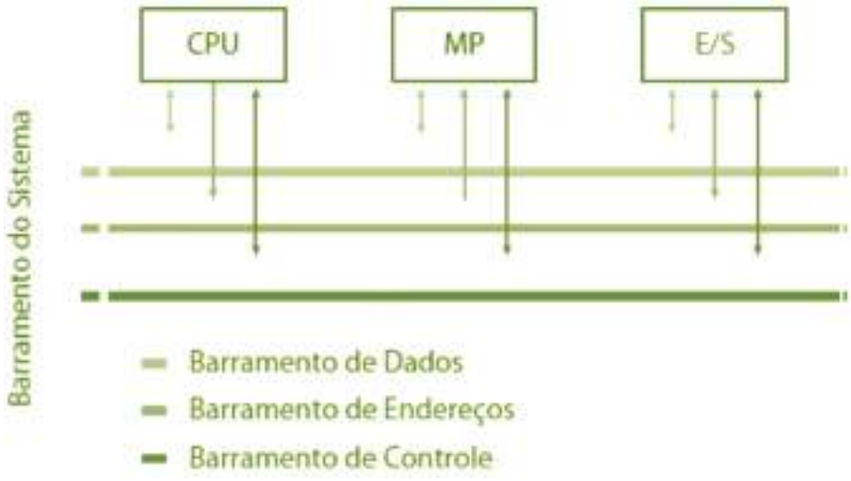


Figura 6: Barramento do sistema

Barramento de dados

Este barramento interliga o RDM - *Registador de dados de memória* (localizado na UCP) à memória principal, para transferência de instruções ou dados a serem executados. É bidirecional, isto é, ora os sinais percorrem o barramento vindo da UCP para a memória principal (operação de escrita), ora percorrem o caminho inverso (operação de leitura). Possui influência direta no desempenho do sistema, pois, quanto maior a sua largura, maior o número de *bits* (dados) transferidos de cada vez e consequentemente mais rapidamente esses dados chegarão ao seu destino (UCP ou memória).

Os primeiros computadores pessoais (ex.: PC-XT) possuíam barramento de dados de oito vias, ou seja, capaz de transferir oito *bits* de cada vez. Atualmente, conforme a arquitetura do processador, podem existir barramento de dados de 32, 64 ou 128 *bits*.

Barramento de endereços

Interliga o REM ou MAR - *Memory Address Register* (localizado na UCP) à memória principal, para transferência dos *bits* que representam um determinado endereço de memória onde se localiza uma instrução ou dado a ser executado. É unidirecional, visto que somente a UCP aciona a memória principal para a realização de operações de leitura ou escrita. Possui tantas vias de transmissão quantos são os *bits* que representam o valor de um endereço.

Seguindo o exemplo anterior, no antigo PC-XT, este barramento possuía 20 linhas: com isso era possível utilizar endereços de no máximo 20 *bits*. Logo, o maior endereço possível, será:

$$2^{20} = 1.048.576 \text{ } Bytes = 1 \text{ MB}$$

Desta forma, a capacidade de armazenamento da memória RAM poderá ser no máximo 1MB. É possível afirmar que o tamanho do barramento de endereços determina a quantidade máxima de armazenamento de dados que a memória principal pode dispor. Atualmente, os barramentos dispõem de significativa capacidade de armazenamento (ex.: 32,64, 128 *bits*), possibilitando grandes espaços para armazenamento na memória.

Barramento de controlo

Interliga a UCP, mais especificamente a Unidade de Controlo (UC), aos restantes componentes do sistema computacional (memória principal, componentes de entrada e de saída) para passagem de sinais de controlo gerados pelo sistema. São exemplos de sinais de controlo: leitura e escrita de dados na memória principal, leitura e escrita de componentes de entrada e saída, certificação de transferência de dados – o dispositivo acusa o término da transferência para a UCP, pedido de interrupção, relógio (*clock*) – por onde passam os pulsos de sincronização dos eventos durante o funcionamento do sistema.

É bidirecional, porque a UCP, por exemplo, pode enviar sinais de controlo para a memória principal, como um sinal indicador de que deseja uma operação de leitura ou de escrita, e a memória principal pode enviar sinais do tipo *wait* (espere), para a UCP aguardar o término de uma operação.

Os barramentos partilham as suas vias de comunicação (normalmente fios de cobre) entre diversos componentes neles ligados como mostra a Figura 6. Nesse caso, somente é permitida a passagem de um conjunto de *bits* de cada vez e, por esse motivo, o controlo do barramento torna-se um processo essencial para o funcionamento adequado do sistema.

No modelo apresentado na Figura 6, há um único barramento de dados, endereços e controle interligando todos os componentes do computador. Isso justifica-se pela grande diferença de características dos diversos componentes existentes, principalmente periféricos (ex.: a velocidade de uma transferência de dados de um teclado é muitas vezes menor que a velocidade de transferência de dados de um disco magnético).

Considerando esse fato, os projetistas de sistemas de computação criaram diversos tipos de barramento, apresentando taxas de transferência de bits diferentes e apropriadas às velocidades dos componentes interligados (ex.: UCP, memória, disco rígido, teclado). Nesse caso há uma hierarquia em que os barramentos são organizados e, atualmente os modelos de organização de sistemas de computação adotados pelos fabricantes possuem diferentes tipos de barramentos:

- **Barramento local:** possui maior velocidade de transferência de dados, funcionando normalmente na mesma frequência do relógio do processador.

Este barramento costuma interligar o processador aos dispositivos de maior velocidade (visando não atrasar as operações do processador): memória *cache* e memória principal;

- **Barramento do sistema:** podemos dizer que se trata de um barramento opcional, adotado por alguns fabricantes, fazendo com que o barramento local faça a ligação entre o processador e a memória *cache* e esta se interligue com os módulos de memória principal (RAM) através do chamado barramento do sistema, de modo a não permitir acesso direto do processador à memória principal. Um circuito integrado denominado ponte (*chipset*) sincroniza o aceso entre as memórias;
- **Barramento de expansão:** também chamado de barramento de entrada e de saída (E/S), é responsável por interligar os diversos dispositivos de E/S aos restantes componentes do computador, tais como: monitor de vídeo, impressoras, CD/DVD, etc. Também se utiliza uma ponte para se ligar ao barramento do sistema; as pontes sincronizam as diferentes velocidades dos barramentos.

Considerando a acentuada diferença de velocidade apresentadas pelos diversos dispositivos de E/S existentes atualmente, a indústria de computadores tem desenvolvido alternativas visando maximizar o desempenho nas transferências de dados entre dispositivos (ex.: entre disco e memória principal). A alternativa encontrada foi separar o barramento de expansão (ou de E/S) em dois, sendo um de alta velocidade para dispositivos mais rápidos (ex.: redes, placas gráficas) e outro de menor velocidade para dispositivos mais lentos (ex.: teclado, *modems*, *rato*). A Figura 7 apresenta este conceito.

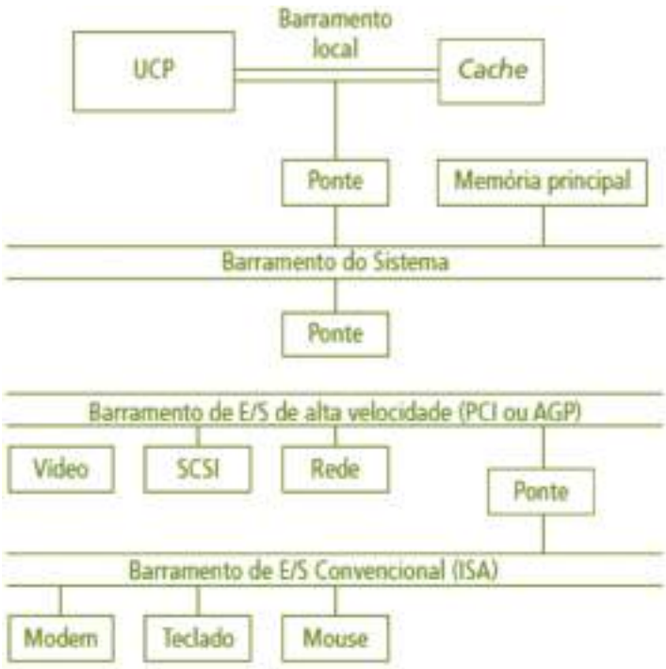


Figura 7: Exemplo de barramento de maior desempenho usado atualmente

Não importa o tipo de barramento, um fator importante que influencia no desempenho do sistema computacional é a largura (tamanho) do barramento, que diz respeito à quantidade de informações (no formato de *bits*) que poderão ser transmitidas simultaneamente por ele. Podemos dizer que é a quantidade de fios ou de vias que um barramento apresenta que vai caracterizar a sua largura.

Quando se fala em quantidade de bits que podem trafegar num barramento, fala-se em taxa de transferência, que revela a medida dessa quantidade, a qual é especificada em bits por segundo (normalmente K *bits*, M bits, etc.).

Conforme discutido anteriormente, cada um dos barramentos permite a sua partilha com os demais componentes do sistema, especialmente o barramento de expansão, que é partilhado com diversos dispositivos de entrada e saída. Para que isso ocorra, é indispensável um mecanismo de controlo de acesso baseado em regras, que garanta que quando um dos dispositivos estiver a utilizar o barramento, os restantes componentes deverão aguardar a sua liberação.

Esse mecanismo de controlo de acesso é denominado protocolo. Isso faz com que um barramento não seja composto somente de fios condutores, mas também de um protocolo.

Os fabricantes de computadores têm procurado uma padronização na definição de protocolos, de forma a evitar que cada um crie o seu próprio, com características diferentes dos demais, o que tornaria muito inflexível o uso de certos componentes (ex.: vídeo, impressoras, discos rígidos) disponibilizados no mercado. Nesse cenário, não importa o fabricante do componente nem as suas características físicas, desde que ele esteja de acordo com os padrões de protocolos definidos pela indústria de computadores. Sendo assim, muitos padrões de barramento de expansão foram desenvolvidos ao longo do tempo, alguns deles já não são mais utilizados. Os mais populares são apresentados a seguir:

- ISA (*Industry Standard Adapter*): desenvolvido pela IBM. Apresenta uma taxa de transferência baixa, mas apesar disso, foi adotado por toda a indústria. Os sistemas atuais não o usam;
- PCI (*Peripheral Component Interconnect*): desenvolvido pela Intel, tornando-se quase um padrão para todo o mercado, como barramento de alta velocidade. Permite transferência de dados em 32 ou 64 *bits* a velocidades de 33 MHz e de 66 MHz. Cada controlador permite cerca de quatro dispositivos;
- USB (*Universal Serial Bus*): tem a característica particular de permitir a ligação de muitos periféricos simultaneamente (pode ligar até 127 dispositivos num barramento – através de uma espécie de centralizador) ao barramento; este, por uma única porta (conetor), liga-se à placa-mãe. Grande parte dos dispositivos USB é desenvolvida com a característica de eles serem ligados ao computador e utilizados logo de seguida, o que é chamado de *plug-and-play*;
- AGP (*Accelerated Graphics Port*): barramento desenvolvido por vários fabricantes, porém liderados pela Intel, com o objetivo de acelerar as transferências de dados do vídeo para a memória principal, especialmente dados em 3D (terceira dimensão), muito utilizados em aplicações gráficas (ex.: jogos);
- PCI *Express* (*Peripheral Component Interconnect Express*): este barramento foi construído por um grupo de empresas denominado PCI-SIG (*Peripheral Component Interconnect Special Interest Group*), composto por empresas como a *Intel*, AMD, IBM, HP e *Microsoft*. Este barramento veio para atender às necessidades por mais velocidade criada pelos novos chips gráficos e

tecnologias de rede apresentando altas taxas de transferência. Assim, o PCI e o AGP foram substituídos pelo PCI *Express*. Nesse barramento, a conexão entre dois dispositivos ocorre de modo ponto a ponto (exclusivo) – comunicação serie. Por esse motivo, o PCI *Express* não é considerado um barramento propriamente dito (considerando que barramento é um caminho de dados onde se podem ligar vários dispositivos ao mesmo tempo, partilhando-o). Essa característica é o que faz ser o meio de comunicação entre dois dispositivos de um computador mais rápido atualmente. Até ao momento existiram três versões desse barramento (1.0 – lançado em 2004; 2.0 – lançado em 2007; e o 3.0 – lançado em 2010).

Cada barramento possui um protocolo padrão que é utilizado pela indústria de computadores para a produção de todos os dispositivos de entrada e saída a serem conectados nos diferentes tipos de barramento. Dessa forma, foram desenvolvidos os chamados *slots*. Um *slot* nada mais é do que um orifício ou um encaixe padronizado inserido na placa-mãe dos computadores de maneira a que os diversos dispositivos possam ser encaixados, desde que acolham um dos padrões disponíveis nela. Assim, nas Figuras 8, 9 e 10 são apresentados alguns slots correspondentes aos padrões de barramentos de expansão listados acima.

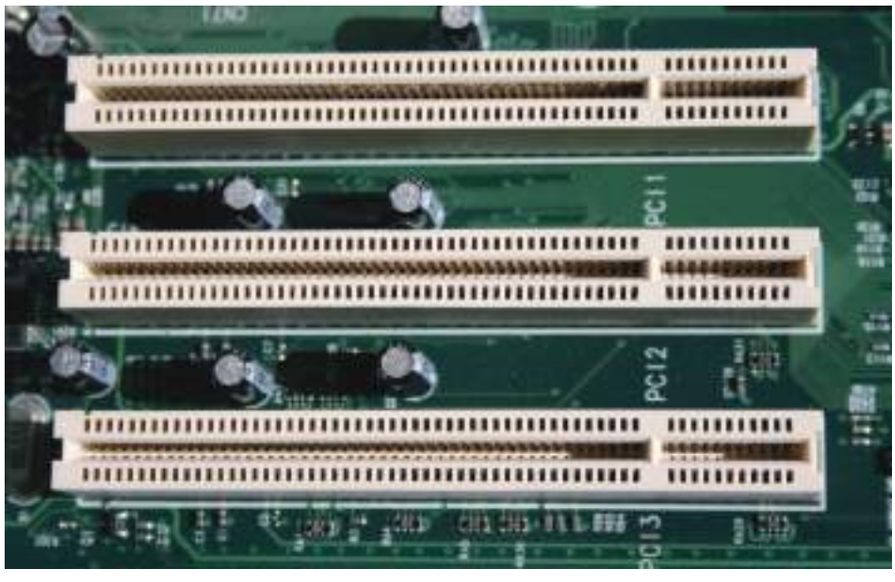


Figura 8: Slot de barramento PCI

Figura 9: Slot AGP 8x

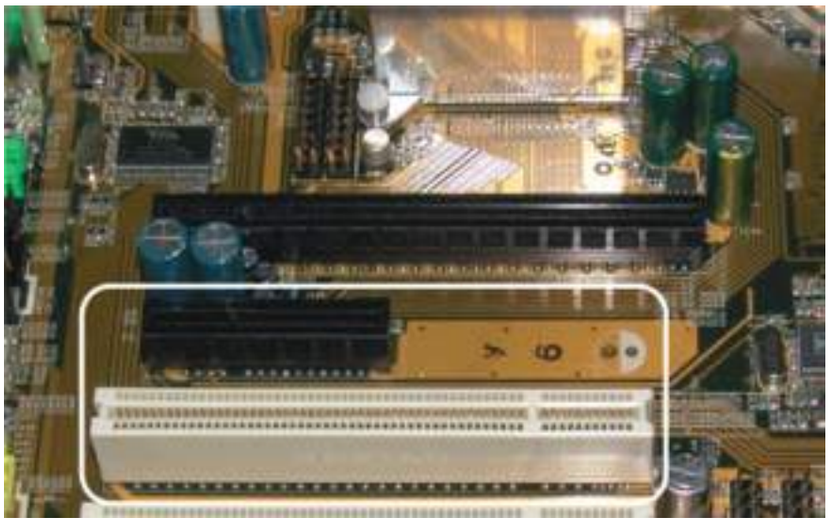


Figura 10: Slots PCI Express 16x (branco) e 4x (preto)

Nesse caso, a principal diferença entre os diversos tipos de barramentos está na quantidade de bits que podem ser transmitidos de cada vez e na frequência de operação utilizada. Os barramentos AGP e PCI Express são considerados os mais rápidos, seguidos pelo barramento PCI original, sendo esses os barramentos mais utilizados em computadores atualmente.

Instruções de Máquina

O termo instrução de máquina já foi mencionado anteriormente mas sem uma explicação efetiva do que realmente significa. Foi demonstrado que a UCP é responsável pela execução de instruções e dados de programas, os quais se encontram armazenados na memória. Além disso, foi dito que uma instrução é uma ordem para que a UCP realize determinada operação (ex.: somar, subtrair, mover um dado de um local para outro, transferir um dado para um dispositivo de saída).

Pode-se dizer que uma máquina pode executar tarefas complicadas e sucessivas se for "instruída" sobre o que fazer e em que sequência isso deve ser feito. Os seres humanos, ao receberem uma instrução (ex.: "trazer a pasta da funcionária Ana"), precisaram de realizar uma série de ações intermediárias, até que a tarefa seja realizada por completo. Então, considerando esse exemplo, seria necessário: localizar o ficheiro em que as pastas de todos os funcionários estão arquivadas; localizar a pasta da funcionária Ana, trazê-la a quem a solicitou.

Da mesma forma, para a máquina (computador) é necessário que cada instrução seja detalhada em pequenas etapas. Isso ocorre porque os computadores são projetados para entender e executar pequenas operações, ou seja, as operações mais básicas (ex.: soma, subtração). Essas pequenas etapas de uma instrução dependem do conjunto de instruções do computador.

Assim, uma instrução de máquina pode ser definida pela formalização de uma operação básica que o *hardware* é capaz de realizar diretamente. Por outras palavras, consiste em transformar instruções mais complexas numa sequência de instruções básicas e compreensíveis pelo processador.

Para exemplificar, vamos considerar um processador com uma ULA capaz de executar a soma ou a multiplicação de dois números (operações básicas), mas não as duas coisas ao mesmo tempo. Agora imaginemos que esse processador precisa executar:

X = A + B*C de uma só vez.

Isso irá gerar a necessidade de transformar essa instrução, considerada complexa para esse processador, numa sequência de instruções mais básicas, da seguinte forma:

- Executar primeiro: T = B*C (sendo que T é um registador ou memória temporária)
- Em seguida realizar a operação: X = A + T

O projeto de um processador é centrado no conjunto de instruções de máquina que se deseja que ele execute, ou seja, do conjunto de operações primitivas que ele poderá executar. Quanto menor e mais simples for o conjunto de instruções, mais rápido é o ciclo de tempo do processador.

Um processador precisa dispor de instruções para:

- Movimentação de dados;
- Aritméticas;
- Lógicas;
- Edição;
- Deslocamento;
- Manipulação de registros de índice;
- Desvio;
- Modificação de memória;
- Formas de ligação à sub-rotina;
- Manipulação de pilha;
- Entrada e saída e de controlo;

Atualmente, há duas tecnologias de projeto de processadores empregues pelos fabricantes de computadores:

- Sistemas com conjunto de instruções complexo (*Complex Instruction Set Computers* - CISC),
- Sistemas com conjunto de instruções reduzido (*Reduced Instruction Set Computers* - RISC).

Ambas as tecnologias serão abordadas em seções subsequentes.

Formato das instruções

Uma instrução é formada basicamente por dois campos:

Código de operação (*Opcode*): um subgrupo de *bits* que identifica a operação a ser realizada pelo processador. É o campo da instrução cujo valor binário identifica a operação a ser realizada, conforme exemplo da Figura 11. Esse valor é a entrada no Decodificador

de Instruções na Unidade de Controle. Cada instrução deverá ter um código único que a identifique.

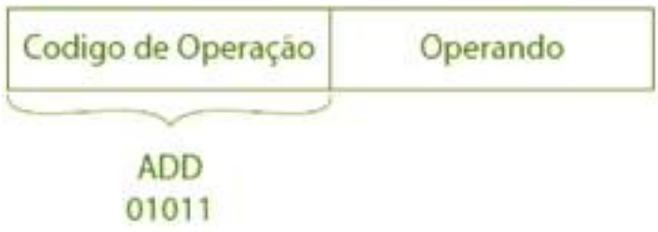


Figura 11: Código de operação de uma Instrução

Operando: um subgrupo de *bits* que identifica o endereço de memória onde está contido o dado que será manipulado, ou pode conter o endereço onde o resultado da operação será armazenado (Figura 12).

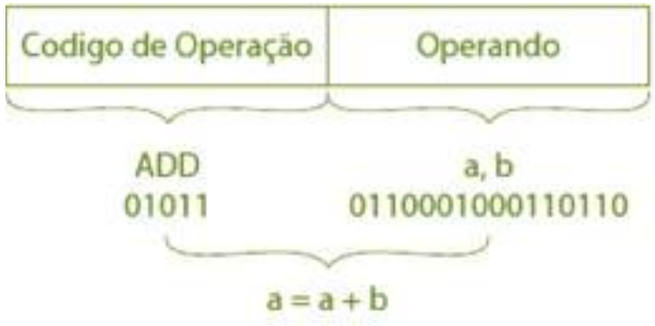


Figura 12: Operando de uma instrução

Ciclo de instrução

A partir da proposta da arquitetura de Von Neumann, da qual os conceitos básicos ainda são válidos, propunha-se que as instruções fossem executadas sequencialmente (a não ser pela ocorrência de um desvio), uma a uma. O contador de instruções indica a sequência de execução, isto é, o CI controla o fluxo de execução das instruções. Na seguir (Figura 13) é ilustrado o ciclo de execução de uma instrução. Em seguida, é descrita em mais detalhes cada uma das fases.

Figura 13: Ciclo de instruções

Fase 1

A UCP procura o código de operação (*Opcode*) na memória principal, o qual está localizado no endereço contido no CI (endereço da próxima instrução a ser executada) e armazena-o no Registador de Instrução (RI): ***RI ← (CI)***

Micro operações da fase 1:

1. A UC lê o conteúdo do CI (endereço da próxima instrução) e coloca o endereço no REM;
2. A UC envia um sinal – via barramento de controlo – à controladora da memória principal para que realize uma operação de leitura;
3. A memória principal lê o endereço que está no REM – via barramento de endereço – e busca o conteúdo da célula referenciada;
4. A memória principal coloca no RDM – via barramento de dados – o conteúdo da célula;

5. A controladora da memória principal envia à UC – via barramento de controlo – o sinal de leitura concluída;
6. A UC transfere o código de operação (conteúdo que está no RDM) ao RI.

Fase 2

O Descodificador de Instrução decodifica (interpreta) o Código de Operação (*Opcode*) contido no RI.

Micro operações da fase 2:

1. O RI envia para o descodificador de instrução os *bits* correspondentes ao *Opcode*;
2. O Descodificador de Instruções determina quantas células a instrução ocupa e identifica a operação a ser realizada;
3. A UC envia um sinal de controlo à ULA informando a operação a ser realizada e incrementa o CI para apontar para a próxima instrução:

   CI → (CI+N), onde, N = nº de células que a próxima instrução ocupa.

Fase 3

A UC procura (se houver) o(s) dado(s) (Operandos): RI ← (Op)

Micro operações da fase 3:

1. A UC envia um sinal – via barramento de controlo – à controladora da memória principal para que realize uma operação de leitura;
2. A memória principal lê o endereço que está no REM – via barramento de endereços – e procura o conteúdo da célula referenciada;
3. A memória coloca no RDM – via barramento de dados – o conteúdo da célula lida;
4. A memória principal envia à UC – via barramento de controlo – um sinal de leitura concluída;

5. A UC transfere o operando (conteúdo do RDM) ao RI (se for um código de operação) ou a um dos registadores internos da UCP (se for um dado).

Obs.: Esta fase repete-se até que sejam trazidos para dentro da UCP todos os operandos necessários à execução da instrução.

Fase 4

A UC comanda a execução da instrução (a operação é executada sobre o(s) dado(s));

Micro operações da fase 4:

1. A ULA executa a instrução sobre os dados disponíveis nos registadores;
2. Ao concluir a operação, a ULA envia um sinal para a UC informando que a execução terminou;
3. A UC identifica o endereço de memória para onde deve ser enviado o resultado da operação e o armazena no REM;
4. A UC autoriza o envio do resultado da operação para o RDM;
5. A UC autoriza a controladora de memória a realizar uma operação de leitura no REM para obter o endereço de memória onde deverá ser escrito o resultado e uma leitura no RDM para obter o resultado a ser escrito na memória.

Fase 5

Se o programa tiver terminado, para, senão, volta à Fase 1.

Arquiteturas RISC e CISC

Até o momento foram apresentadas algumas características das instruções de máquina, destacando que a arquitetura de um processador está fortemente relacionada com o seu conjunto de instruções e que um processador sempre estará preparado para processar instruções básicas (ex.: soma, subtração, comparação).

A presença de outros tipos de instruções depende da orientação da arquitetura (*Complex Instruction Set Computer* – CISC ou *Reduced Instruction Set Computer* – RISC) e se está voltada para aplicações gerais ou específicas.

Arquitetura CISC

A arquitetura CISC (*Complex Instruction Set Computer*) tem conjuntos de instruções grandes, de tamanhos variáveis e com formatos complexos. Processadores CISC são capazes de executar várias centenas de instruções complexas diferentes, sendo extremamente versátil.

Estes processadores possuem uma micro programação, ou seja, um conjunto de códigos e de instruções que são gravados no processador, permitindo-lhe receber as instruções dos programas e executá-las utilizando as instruções contidas na sua micro programação.

A possibilidade de o processador executar instruções complexas facilita a programação de alto nível, diminuindo o código e consequentemente o espaço utilizado em memória pelos programas.

Em contrapartida, temos um menor desempenho devido ao processador ter de executar instruções maiores e mais complexas. O projeto e a construção de processadores CISC tem um custo elevado, por causa da sua alta complexidade e maior número de componentes internos do chip.

Arquiteturas que se basearam em CISC foram: System/360 através da z/Architecture, PDP-11, VAX, Motorola 68k, e x86.

Arquitetura RISC

Para driblar o baixo desempenho dos processadores CISC, foi então criada a arquitetura RISC (*Reduced Instruction Set Computing*), que, ao contrário da arquitetura CISC, suporta algumas poucas instruções simples. Em consequência disso, processadores dessa arquitetura tem construção mais simples e possuem menos componentes no seu chip, diminuindo seu custo final.

Além de baixo custo, também há maior desempenho, pois por haver menos circuitos internos nestes processadores, encontra-se menores temperaturas durante seu funcionamento, possibilitando produzir processadores com clocks ainda maiores. Em 1997 já havia processadores RISC, como os Alpha, operando a 600Mhz.

Contudo, devido a processadores desta arquitetura trabalharem com menos instruções, a programação de alto nível para os mesmos é mais trabalhosa, pois para se usar alguma função complexa encontrada em processadores CISC, deve-se juntar mais de uma instrução simples.

Algumas famílias de processadores que utilizam esta arquitetura são: DEC Alpha, AMD 29k, ARC, ARM, Atmel AVR, Blackfin, MIPS, PA-RISC, Power (inclusive PowerPC), SuperH, e SPARC.

Arquitetura Híbrida

Devido a cada uma das duas arquiteturas terem as suas peculiaridades (como visto na Tabela 1), os fabricantes de processadores decidiram unir ambas numa só, formando uma arquitetura híbrida. Num processador RISC, há um (ou mais) circuito CISC, e vice-versa.

Desta forma, o processador trabalha mais rápido em instruções que requer maior demanda (RISC), devido à maioria das instruções executadas serem de pouca complexidade e, quando necessário, haverá a disponibilidade de se executar instruções complexas, encontradas em processadores CISC.

Atualmente, grande parte dos processadores adota este conceito.

RISC	CISC
Múltiplos conjuntos de registadores, muitas vezes superando 256	Único conjunto de registadores, tipicamente entre 6 e 16 registadores
Três operandos de registadores permitidos por instrução (por ex., *add* R1, R2, R3)	Um ou dois operandos de registadores permitidos por instrução (por ex., *add* R1, R2)
Passagem eficiente de parâmetros por registadores no chip (processador)	Passagem de parâmetros ineficiente através da memória
Instruções de um único ciclo (ex. *load* e*store*)	Instruções de múltiplos ciclos
Controlo *hardwired* (embutido no hardware)	Controlo micro programado
Altamente paralelizado (*pipelined)*	Fracamente paralelizado
Instruções simples e em número reduzido	Muitas instruções complexas
Instruções de tamanho fixo	Instruções de tamanho variável
Complexidade no compilador	Complexidade no código
Apenas instruções *load* e *store* podem aceder à memória	Muitas instruções podem aceder à memória
Poucos modos de endereçamento	Muitos modos de endereçamento

Tabela 1: Características da arquitetura RISC e CISC

Representação de dados

Um computador funciona através da execução sistemática de instruções que o orientam a realizar algum tipo de operação sobre valores (numéricos, alfanuméricos ou lógicos). Esses valores são genericamente conhecidos como dados**.**

Os dados são convertidos internamente num código de armazenamento no formato binário. Para compreender melhor, pode-se considerar o exemplo a seguir:

Para o valor decimal *143*

00110001 (algarismo 1)

00110111 (algarismo 4)

00110011 (algarismo 3)

Qualquer que tenha sido a linguagem de programação utilizada para escrever o programa, ela deverá ser convertida para código-objeto (código binário) e, em seguida, para o código executável (conjunto de códigos de máquina), o qual é gerado pelo compilador da linguagem), conforme a Figura 14. Essa conversão também inclui dados, que deverão ser alterados de modo a estarem numa forma apropriada para utilização pela ULA (ex.: números inteiros ou fracionários). Por exemplo, para efetuar uma soma, a ULA executa, passo a passo, uma série de micro operações (um algoritmo): verificar o sinal dos números, verificar o tipo do número, etc.

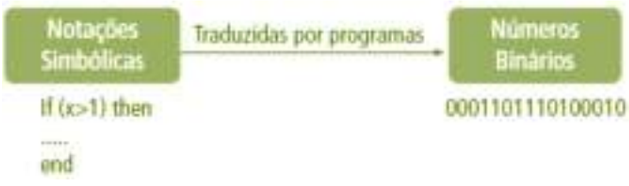


Figura 14: Processo de conversão

Formas de representação

As diferentes formas de representação e respetivos algoritmos de realização das operações matemáticas são muito úteis, pois cada uma tem uma aplicação mais vantajosa que a outra. Cabe ao programador a escolha da forma a ser utilizada pelo sistema, podendo ser explícita ou implícita. Explícita, quando o programador define as variáveis e constantes no seu programa. Implícita, quando é deixado para que o compilador faça a escolha.

Tipos de dados

Definem para o sistema como cada dado deverá ser manipulado, pois conforme citado anteriormente, cada tipo de dado recebe um tratamento diferenciado pelo processador.

Exemplo:

VAR X:=INTEGER;

VAR X:=REAL;

Os termos INTEGER e REAL são interpretados de modo diferente, acarretando alterações significativas tanto no modo de organizar os *bits* que representam um número quanto na sequência de etapas do algoritmo de execução de uma operação aritmética com o número.

De modo geral, as seguintes formas de dados são mais utilizadas nos programas atuais de computadores (formas primitivas, entendidas pelo *hardware*):

- Tipo Caractere: dados sob forma de caractere;
- Tipo Lógico: dados sob forma lógica;
- Tipo Numérico: dados sob forma numérica.

Outras formas mais complexas são permitidas em certas linguagens modernas (como tipo REGISTRO, tipo ARRAY, tipo INDEX, tipo POINTER etc.). No entanto, durante o processo de compilação, os dados acabam sendo convertidos finalmente nas formas primitivas já mencionadas, para que o *hardware* possa executá-las.

Tipo caracter

A representação interna de informações em um computador é realizada através de uma correspondência entre o símbolo da informação e o grupo de algarismos binários (*bits*). Cada símbolo (caractere, número ou símbolo) possui uma identificação específica.

Exemplo: Símbolo “A” ⇒ Algarismos binários “10101101”

Podemos nos questionar como é possível representar, com apenas dois símbolos (0 e 1), todos os caracteres alfabéticos, algarismos decimais, sinais de pontuação, de operações matemáticas, entre outros, necessários à elaboração de um programa de computador.

A resposta para essa pergunta seria: pela utilização do método chamado de codificação, pelo qual cada símbolo da nossa linguagem tem um correspondente grupo de *bits* que identifica univocamente o referido símbolo (caractere).

Existem alguns padrões de codificação previamente definidos, conforme apresentados na tabela 2.

CÓDIGOS CARACTERES	
BCD – Binary Code Decimal	Utiliza 6 bits/caracteres, codificando 64 caracteres.
EBCDIC – Extended Binary Coded Decimal Interchange Code	Exclusivo da IBM, utilizando 8 bits para codificar 256 caracteres.
ASCII – American Standart Code for Information Interchange	ASCII – usado pelos demais fabricantes. Utiliza oito bits/caractere em sua versão extendida, codificando 256 caracteres.
UNICODE	Código que utiliza 16 bits/símbolo, podendo representar 65.536 símbolos diferentes. Pretende codificar em um único código os símbolos de todas as linguagens conhecidas no mundo. Está sendo desenvolvido por um consórcio desde 1991

Tabela 2: Padrões de codificação de caracteres

A utilização de padrões de codificação (ex.: ASCII, Unicode) é o método primário de introdução de informações no computador. As restantes formas de representação de informação (tipos de dados) surgem no decorrer do processo de compilação ou interpretação do programa. O padrão de codificação mais utilizado pela indústria de computadores é o ASCII. A codificação correspondente a esse padrão já é parte do *hardware* (armazenado numa memória do tipo ROM) e é definida pelo próprio fabricante.

A Figura 15 apresenta parte da tabela ASCII como exemplo.

Binário	Decimal	Hexa	Glifo	Binário	Decimal	Hexa	Glifo	Binário	Decimal	Hexa	Glifo
0010 0000	32	20		0010 0000	64	40	@	0010 0000	96	60	
0010 0001	33	21	!	0010 0001	65	41	A	0010 0001	97	61	a
0010 0010	34	22	"	0010 0010	66	42	B	0010 0010	98	62	b
0010 0011	35	23	#	0010 0011	67	43	C	0010 0011	99	63	c
0010 0100	36	24	$	0010 0100	68	44	D	0010 0100	100	64	d
0010 0101	37	25	%	0010 0101	69	45	E	0010 0101	101	65	e
0010 0110	38	26	&	0010 0110	70	46	F	0010 0110	102	66	f
0010 0111	39	27	'	0010 0111	71	47	G	0010 0111	103	67	g
0010 1000	40	28	(	0010 1000	72	48	H	0010 1000	104	68	h
0010 1001	41	29	)	0010 1001	73	49	I	0010 1001	105	69	i
0010 1010	42	2A	*	0010 1010	74	4A	J	0010 1010	106	6A	j
0010 1011	43	2B	+	0010 1011	75	4B	K	0010 1011	107	6B	k
0010 1100	44	2C	,	0010 1100	76	4C	L	0010 1100	108	6C	l
0010 1101	45	2D	-	0010 1101	77	4D	M	0010 1101	109	6D	m
0010 1110	46	2E	.	0010 1110	78	4E	N	0010 1110	110	6E	n
0011 1111	47	2F	/	0011 1111	79	4F	O	0011 1111	111	6F	o
0011 0000	48	30	0	0011 0000	80	50	P	0011 0000	112	70	p

Figura 15: Exemplo parcial da Tabela ASCII

Tipo lógico

Permite a utilização de variáveis que possuem apenas dois valores para representação: Falso (*bit* 0) e Verdadeiro (*bit* 1). As funções de cada operador lógico estão apresentadas no Tabela 3.

PORTA LÓGICA	DEFINIÇÃO
AND	O operador lógico AND é definido de modo a que o resultado da operação com ele será VERDADE se e somente se todas as variáveis de entrada forem VERDADE (=1). Caso contrário, o resultado será FALSO (=0)
OR	O resultado da operação será VERDADE (=1) se um operando (ou variável lógica) ou o outro for verdadeiro. Basta que apenas um dos operandos seja verdadeiro. Caso contrário, o resultado será FALSO (=0). Operadores lógicos OR também são largamente utilizados em lógica digital ou na definição de condições em comandos de decisão de certas linguagens de programação.

NOT	É definido de modo a produzir na saída um resultado de valor oposto (ou inverso) ao da variável de entrada. É usado apenas com uma única variável. Desse modo, se a variável tem o valor 0 (FALSO), o resultado da operação NOT sobre essa variável será 1 (VERDADE), e se a variável for igual a 1 (VERDADE), então o resultado do NOT será 0 (FALSO).
XOE	O operador lógico XOR (EXCLUSIVE-OR) ou OU EXCLUSIVO é definido de modo a prover um resultado VERDADEIRO se apenas uma das variáveis ou operadores for VERDADEIRA. Sendo X=A XOR B, o resultado X será VERDADE se exclusivamente (daí o nome OU EXCLUSIVO) A OU B for VERDADE. Caso ambos sejam "VERDADE" ou ambos "FALSO", então o resultado será FALSO.
NAND	Negação do operador lógico AND.
NOR	Negação do operador lógico OR.

Tabela 3: Operadores lógicos e as suas funcões

Tipo numérico

Como os computadores são elementos binários, a forma mais eficiente de representar números deve ser "binária", isto é, converter o número diretamente de decimal para o seu correspondente valor binário. Deste modo a ULA poderá executar as operações mais rapidamente.

Existem três fatores que devem ser considerados, pois podem acarretar inconvenientes no projeto e na utilização da máquina:

- A representação do sinal do número;
- A representação da vírgula (ou ponto) que separa a parte inteira da fracionária de um número não inteiro;
- A quantidade limite de algarismos possível a ser processada pela ULA.

O problema que consiste do sinal do número pode ser resolvido com o acréscimo de mais um *bit* na representação do número (adicionado à esquerda), como *bit* mais significativo. Esse *bit* adicional indica o sinal do número. A conversão adotada (conforme a tabela 4) é:

- Valor positivo: *bit* de sinal igual a 0;
- Valor negativo: *bit* de sinal igual a 1.

Valor Decimal	Valor Binário
- 47	100101111 (com 9 bits)
+ 47	000101111 (com 9 bits)

Tabela 4: Exemplo

Outro problema reside na forma de representação de números fracionários, por causa da dificuldade de representar a vírgula/ponto internamente, entre a posição de dois *bits*. O que ocorre é que a vírgula não é efetivamente representada, mais sim assumida a sua posição no número e este sendo representado apenas pelos seus algarismos significativos como se fosse inteiro.

Exemplo: 110111,110 ⇒ 110111110

O sistema identifica que quantidade de algarismos é inteira e que a quantidade é fracionária através da escolha entre dois modos de representação e de realização de operações aritméticas:

- Representação em ponto fixo (vírgula fixa);
- Representação em ponto flutuante (vírgula flutuante).

A quantidade de dígitos disponíveis no sistema de computação (processador) para representar números é um problema relevante. Na matemática os números reais existentes são infinitos; no entanto, computadores são máquinas onde as células e registadores possuem tamanho finito mas que têm capacidade de representar uma quantidade finita de números. Deste modo surge o aspeto denominado *overflow* ou estouro da capacidade de representar números. Há também o *underflow*, que se caracteriza por ocorrer um resultado cujo valor é menor que o menor valor representável.

Representação de Instruções

Conforme já mencionamos diversas vezes, o funcionamento básico de um computador está totalmente relacionado às operações primitivas que a UCP (o hardware) é capaz de realizar diretamente.

Sabemos também que as referidas operações primitivas, básicas, têm a sua execução efetivada através da realização, passo a passo, de uma sequência de ações menores (denominadas micro operações). Esta sequência constitui o algoritmo que conduz à execução da operação, o qual chamamos de instrução da máquina para realizar a dita operação.

As UCPs (os processadores) são fabricadas contendo internamente a programação para realização de uma grande quantidade de operações primitivas, cada uma definida pela respetiva instrução de máquina, que é a formalização da operação em si.

Denomina-se *conjunto de instruções (instruction set)* de um processador esta quantidade de instruções que ele pode realizar diretamente.

Neste capítulo pretende-se apresentar um pouco mais de detalhes sobre as referidas instruções de máquina, ampliando assim as informações constantes dos capítulos anteriores. Entre esses detalhes destaca-se de modo importante o formato da instrução, isto é, o significado de cada um dos bits que constitui uma instrução de máquina.

Na figura 16 vemos como são representadas instruções de máquina com um formato-padrão de um operando.

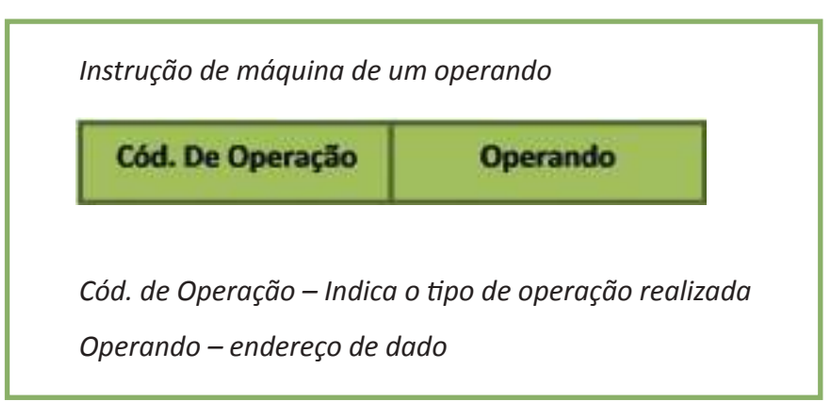


Figura 16: Exemplo de formato de instruções de um operando

No entanto, o formato apresentado na figura não é o único utilizado nos sistemas de computação comerciais. Na realidade, há diversas formas de representar instruções, e vários modos da Unidade de Controlo interpretar a procura do(s) dado(s).

Cada formato de instrução tem características próprias, com as suas vantagens e desvantagens, podendo ser eficaz em certas aplicações e desaconselhável em outras.

Na prática, o conjunto de instruções definido para uma determinada UCP é sempre constituído de uma mistura de formatos diferentes, justamente para permitir a melhor aplicação em cada caso.

É interessante observar que há opiniões divergentes quanto à vantagem de se obter versatilidade com um numeroso conjunto de instruções de máquina.

O problema, neste caso, está na dificuldade dos compiladores em escolher a melhor opção de instrução para cada tipo de aplicação. A tendência é fazer com que os compiladores operem quase sempre com uma pequena quantidade de instruções; com isso, perderiam sentido tantas instruções pouco úteis (ou pouco utilizadas).

É conveniente repetir, neste ponto, alguns dos aspetos básicos sobre o funcionamento de um computador, especificamente referentes aos processadores (Unidade Central de Processamento).

Os processadores são projetados com uma arquitetura definida com o único propósito de realizar operações básicas muito simples, tais como: somar dois valores, subtrair dois valores, mover um valor de um local para outro. A programação da sequência de passos para realizar cada uma das mencionadas operações é inserida no processador durante o processo da sua produção, caracterizando a instrução de máquina.

Como a linguagem utilizada pelas máquinas para realizar suas tarefas é binária, também uma instrução de máquina deve ser representada nessa linguagem e, assim, ser formada por um conjunto de bits, que indica, conforme o seu formato e programação, o que o processador deve realizar (qual a operação) e como realizar (a operação), além de ter que indicar com que dados a operação irá trabalhar (a localização do(s) dado(s)).

O projetista do processador escolhe, então, que operações aquele processador irá realizar e define, para cada uma delas, todos os detalhes de identificação e execução da operação, estabelecendo, assim, o formato de cada instrução de máquina.

Basicamente, uma instrução possui dois elementos:

- O que indica ao processador o que fazer e como fazer — denominado código de operação (C. Op.);
- O que indica ao processador com que dado ou dados a operação irá se realizar — denominado operando (Op.).

Esses elementos são identificados para o processador por um grupo de bits específico, os quais, em conjunto, formam a instrução completa.

Toda instrução de máquina possui um código de operação específico e único para aquela tarefa. Este código, após ser descodificado durante o ciclo de execução da instrução, permitirá que a Unidade de Controlo emita os sinais necessários, e previamente programados, para se efetuar a sequencia de passos de realização da operação indicada.

A quantidade de bits estabelecida para este campo da instrução (C. Op.) define o limite máximo de instruções que o processador poderá executar. Se, por exemplo, um determinado processador possui instruções de máquina, cujo C. Op. é um campo de 6 bits, então, este processador somente poderá realizar 64 instruções diferentes, dado que $2^6 = 64$.

Além do código de operação, as instruções podem conter um ou mais campos denominados operando, cada um deles contendo informação sobre o dado a ser manipulado (o tipo da informação sobre o dado — o seu valor ou o endereço de memória onde localizá-lo). A Figura 17 mostra exemplos de alguns formatos típicos de instruções de máquina.

Para compreender, de modo ordenado, a representação de instruções em sistemas de computação, pode-se efetuar a análise do assunto segundo dois aspetos:

- Quantidade de operandos;
- Modo de interpretação do valor armazenado no campo operando (modo de endereçamento do dado).

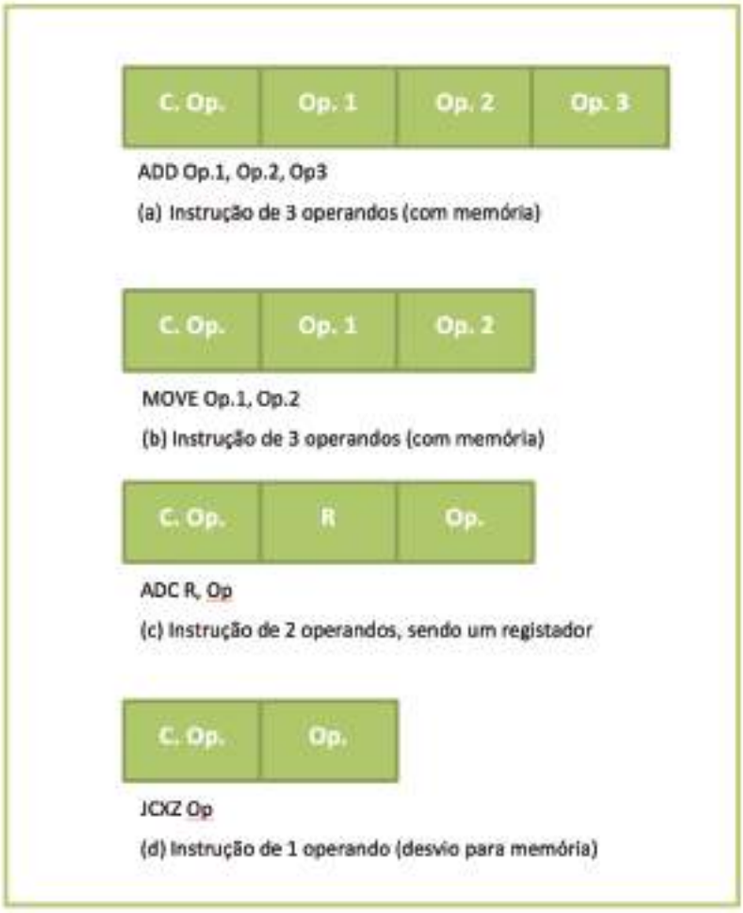


Figura 17: Exemplos de formatos de instruções de máquina

Quantidade de operandos

Desde os primeiros processadores concebidos até os atuais, os projetistas têm definido conjuntos de instruções dos mais variados tipos e formatos, de modo que, mesmo num único e específico processador, as instruções tendem a ter formato diferente, isto é, tamanhos e campos diversos, conforme a operação que a instrução indica.

Assim, instruções que realizam operações aritméticas ou lógicas tendem a ter 2 operandos (embora uma instrução deste tipo devesse possuir 3 operandos para se tomar completa, como veremos mais adiante), algumas vezes 1 apenas (os outros ficam implícitos) e muito raramente 3 operandos. E assim por diante.

Um dos primeiros formatos de instrução idealizados foi incluído no sistema SEAC (Standard Eastern Automatic Computer), que ficou pronto em 1949 e possuía quatro operandos, conforme mostrado na Figura 18.

Tal instrução (que não é mais utilizada — nem a máquina) era completa. Não só possuía indicação explícita da localização de todos os operandos (no caso, é claro, de se tratar de uma instrução que realiza uma operação aritmética), como também já trazia armazenado o endereço da próxima instrução. No caso de um computador com memória de 2K células (endereços) e com instrução possuindo um código de operação de 6 bits (conjunto de 64 possíveis instruções), cada instrução teria um tamanho total de 50 bits.

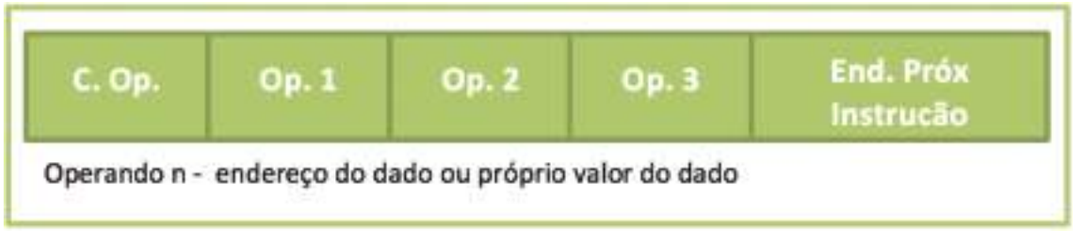


Figura 18: Exemplo de formato de instrução de quatro operandos

Se a memória tem 2K células, então cada endereço seria indicado por um número de 11 bits, pois:

$2^{11} = 2K$

Como cada campo operando contém o endereço de um dado e a instrução possui quatro operandos:

4 (operandos) X 11 (bits de endereços) + 6 (C. Op.) = 50 bits

Essa máquina poderia ter, por exemplo, uma instrução de soma do tipo (em Assembler): ADD X,Y,Z,P, cuja descrição para execução seria: (Z) <- (X) + (Y), sendo P <- endereço da próxima instrução

Essa única instrução permitiria a execução da expressão:

C = A + B

podendo ser representada em linguagem Assembly como:

ADD A,B,C,P

Muitas considerações podem ser feitas a respeito das vantagens relativas à quantidade de operandos desse tipo de instrução, entre as quais podemos citar:

- *Preenchimento* — a instrução possui todos os operandos necessários à realização de uma operação aritmética, dispensando até instruções de desvio incondicional, pois o endereço de desvio consta no campo P;

- *Menor quantidade de instruções em um programa* - em comparação ao uso de instruções com menor quantidade de operandos, como veremos mais adiante.

Mas, apesar dessas vantagens, esse tipo de instrução tem uma grande desvantagem: a ocupação de demasiado de espaço de memória, principalmente se atentarmos para o fato de que grande número de instruções de um programa não precisa de todos os três operandos (praticamente somente as instruções que tratam de operações matemáticas é que poderiam requerer 3 operandos).

Uma instrução de desvio incondicional, por exemplo, precisaria apenas do campo P (que conteria o endereço da próxima instrução, para onde se estaria querendo desviar), restando inúteis 33 bits da instrução (3X 11 bits).

Outras instruções também deixam de usar todos os campos operandos. A instrução que transfere um valor da MP para a UCP (LOAD) precisa apenas de dois campos: um para o endereço do dado e outro para indicar o endereço da próxima instrução. Restariam inúteis dois campos ou 22 bits. Às vezes, este tipo de instrução poderia requerer outro operando, para indicar o destino do dado na UCP, se considerarmos que o local de destino (registador) poderia ser um dentre vários. No exemplo, consideramos que o LOAD seria realizado armazenando o dado em um registador especial e único, o acumulador, prescindindo, assim, de indicação explícita na instrução.

Um dos fatores mais importantes no projeto de uma UCP consiste na escolha do tamanho das instruções.

Essa escolha depende de várias características da máquina, tais como:

- Tamanho da memória;
- Tamanho e organização das células da MP;
- Velocidade de acesso;
- Organização do barramento de dados.

O ponto crucial a ser analisado por ocasião do projeto reside na comparação entre dois fatores, economia de espaço X conjunto completo e poderoso de instruções.

Um bom conjunto de instruções requer muitas instruções (para atender a diferentes tipos de aplicações), o que implica definir muitos códigos de operação (um para cada instrução, é claro) e, consequentemente, mais bits para o campo código de operação.

O aumento da quantidade de bits do campo C. Op. acarreta aumento do tamanho da instrução e, principalmente, aumento da tarefa do descodificador durante a execução do ciclo de instrução

Além disso, um grande conjunto de instruções pode indicar que elas sejam mais completas e, nesse caso, há necessidade de muitos bits na instrução, já que haverá diversos campos de operandos.

No entanto, quanto mais bits a instrução possui, mais memória se consome para armazená-la. Isto vai contra o desejo de economia de espaço de armazenamento, visando à redução de custos (mesmo atualmente, com a redução dos preços dos dispositivos eletrónicos, memória é sempre um elemento caro).

O problema é de tal ordem que há uma corrente de pesquisadores e fabricantes que utiliza outra tecnologia para definir a arquitetura dos processadores, na busca de economia e eficiência nessa área de especificação de conjunto de instruções. Esta arquitetura diferente, conhecida como RISC — Reduced Instruction Set Computer.

Continuando, pode-se obter a desejada economia, sem comprometer a flexibilidade das instruções e, como possível solução do problema, efetuar uma redução na quantidade de operandos nas instruções.

Se, por exemplo, na instrução apresentada Figura 18 fosse retirado um operando, então, os 44 bits seriam usados para três operandos, o que, mantido o mesmo tamanho da palavra, daria para se aceder endereços de 14 bits (2^{14} = 16K) em vez dos 2K anteriores. Além disso, os 2 bits restantes poderiam servir para aumentar o campo código de operação para 8 bits (aumentaria o conjunto de instruções para 256).

Outra possibilidade seria reduzir o tamanho total da instrução, melhorando o uso da memória e permitindo maiores programas.

Na prática, a procura de instruções menores redundou inicialmente na retirada do campo P. Isto foi possível através da conceção de uma técnica mais aperfeiçoada de obter, de forma automática, o endereço da próxima instrução. Esta técnica consistiu na criação de um registador especial na UCP, cuja função indica o endereço da próxima instrução (sendo automaticamente incrementado, está sempre a indicar o novo endereço). Trata-se, de um CI — Contador de Instrução (ou em inglês, MAR — Memory Address Register).

Instruções com Três Operandos

Uma instrução que trata da execução de uma operação aritmética com dois valores requer, naturalmente, a indicação explícita da localização desses valores. Quando eles estão armazenados na MP, o campo operando deve conter, então, o endereço de cada um deles, o que indica a necessidade de 2 campos operandos. Além disso, se se trata de uma soma de valores é natural imaginar que o sistema deve ser orientado para armazenar o resultado em algum local e, assim, deve haver um terceiro campo operando, para indicar o endereço da MP onde será armazenado o resultado.

A Figura 19 apresenta o formato básico de uma instrução de três operandos. Pode-se estabelecer, por exemplo, que os campos Operando 1 e Operando 2 representem o endereço de cada dado utilizado como operando numa operação aritmética ou lógica e que o campo Operando 3 contenha o endereço para armazenamento do resultado dessas operações.

As instruções de três operandos, empregues em operações aritméticas, podem ser do tipo:

ADD A,B,X (X) <- (A) + (B)

SUB A,B,X (X) <- (A) - (B)

MPY A,B,X (X) <- (A) (B)

DIV A,B,X (X) <- (A) (B)

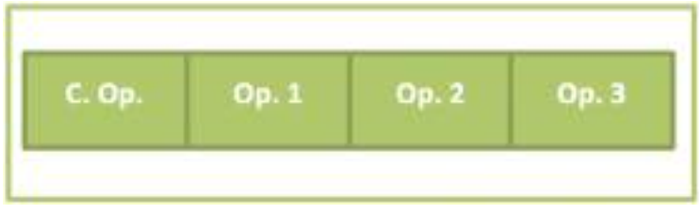


Figura 19: Exemplo de formato de instrução de três operandos

Para exemplificar a sua utilização, consideremos que um programa escrito em linguagem de alto nível contenha o comando apresentado a seguir, o qual calcula o valor de uma expressão algébrica:

X = A * (B + C * D - E/F)

Como resultado do processo de compilação, o referido comando será convertido em instruções de máquina que, em conjunto, representam um programa com resultado idêntico ao do comando já apresentado.

Para simplificar e melhorar o nosso entendimento, vamos utilizar as instruções da linguagem Assembly para montar o programa equivalente, em vez de criarmos instruções em linguagem binária.

A sequência do algoritmo para resolver a equação é a seguinte, considerando as regras matemáticas usuais:

1. Inicialmente, resolver as operações internas aos parênteses.
2. Dentre as operações existentes, a primeira a ser realizada é a multiplicação de C por D (o resultado é armazenado numa variável temporária, T1) e, de seguida, a divisão de E por F (resultado em uma variável temporária, T2) — prioridade dessas operações sobre soma e subtração.
3. Posteriormente, efetua-se a soma de B com T l.
4. Subtrai-se T2 do resultado dessa soma.
5. Finalmente, multiplica-se A por esse último resultado e armazena-se em X.

Considerando as instruções Assembly de 3 operandos, anteriormente definidas, podemos construir o seguinte programa equivalente ao comando exemplificado:

```
MPY   C,D,T1      ; multiplicação de C e D, resultado em T l (item 2)
DIV   E,F,T2      ; divisão de E por F, resultado em T2 (item 2)
ADD   B,T1,X      ; soma de B com T l; resultado em X (item 3)
SUB   X,T2,X      ; subtração entre X e T2, resultado em X (item 4)
MPY   A,X,X       ; multiplicação de A por X, resultado em X (item 5)
```

Utilizaram-se duas variáveis temporárias, T1 e T2. Todas as letras usadas (A, B, C, D, E, F, T1, T2) representam endereços simbólicos de memória. A ordem de execução foi a normal: da esquerda para a direita, de acordo com a prioridade matemática.

Observemos que há operandos com endereços iguais, o que é um desperdício de espaço de memória. Também deve ser observado que o número de instruções é igual ao de operações; isto vai acontecer sempre com instruções de 3 operandos, pois cada uma delas resolve por completo uma operação.

Ainda que se tenha reduzido a quantidade de operandos (de quatro para três), continuamos a consumir demasiado espaço de memória para a efetiva utilização dos operandos (continua a haver muitas instruções que não requerem todos os campos de operandos).

Instruções de máquina de 3 operandos são raramente encontradas em conjuntos de instruções dos atuais processadores existentes no mercado, devido principalmente ao seu grande tamanho.

Instruções com Dois Operandos

No exemplo anterior, pudemos observar que a maioria das instruções exige apenas dois endereços (o outro é repetido). Considerando a importância do problema de economia de espaço de armazenamento, foram criadas instruções com dois campos de operandos, como:

ADD A.B (A) <- (A) + (B)

As restantes operações aritméticas seriam realizadas com instruções de formato igual. Na realidade, o conjunto de instruções aritméticas de 2 operandos poderia ser do tipo a seguir indicado:

ADD Op.1, Op.2 (Op.1) <- (Op1l) + (Op.2)

SUB Op.1, Op.2 (Op.1) <- (Op.1) - (Op.2)

MPY Op.1, Op.2 (Op.1) <- (Op.1) * (Op.2)

DIV Op.1, Op.2 (Op.1) <- (Op.1) / (Op.2)

Nesse caso, o conteúdo da posição de memória, cujo endereço está indicado em Op.1 (valor do primeiro operando) será destruído com o armazenamento, naquele endereço, do resultado da operação. Pode-se evitar, quando necessário, essa destruição, "salvando-se" o valor da variável, antes da execução da instrução.

Esse "salvamento" de variável pode ser realizado por uma nova instrução:

MOVE A,B (A) <- (B)

Com essas instruções de dois operandos, o comando correspondente à equação:

X = A * (B + C * D - E/F)

poderia ser convertido, para execução, no programa Assembler apresentado a seguir, ainda de acordo com a sequência apresentada no item anterior:

```
MPY   C,D     ;multiplicação de C por D, resultado em C (item 2)
DIV   E,F     ;divisão de E por F, resultado em E (item 2)
ADD   B,C     ;soma de B com C, resultado em B (item 3)
SUB   B,E     ;subtração entre B e E, resultado em B (item 4)
MPY   A,B     ;multiplicação de A por B, resultado em A (item 5)
MOVE  X,A     ;armazenamento do resultado final, A, em X
```

Note, agora, que a sequência contém uma instrução a mais que o número de operações da expressão; e, também, podemos observar que foram destruídos os valores armazenados nos endereços correspondentes às variáveis A, B, C e E.

É sempre provável que se empregue, num programa, uma variável mais de uma vez. Para evitar que uma determinada variável tenha o seu valor destruído devido ao armazenamento de um resultado parcial no endereço correspondente, podem-se usar algumas variáveis temporárias e instruções MOVE, com o propósito de preservar todos os valores de variáveis.

Assim, a execução da instrução

```
MPY   C.D
```

acarretaria a destruição do valor da variável C, já que a descrição da instrução orienta a soma do valor armazenado no endereço indicado no campo (Op.1) que, neste caso, é correspondente à variável C, com o valor armazenado no endereço indicado no campo (Op.2), que, no caso, é correspondente à variável D, e que o resultado obtido (valor C + D) seja armazenado na MP no endereço indicado no mesmo campo (Op.1), que era o de C.

Para evitar essa destruição, o programa anterior poderia ser alterado para o seguinte:

```
MOVE  X,C     ;mover cópia de C para o endereço X
MPY   X,D     ;multiplicar X (cópia de C) por D. O resultado será armazenado em X e não
              mais em C e, assim, o valor da variável não é destruído (item 2).
MOVE  T1,E    ;mover cópia de E para o endereço T1
DIV   T1,F    ;dividir T1 (cópia de E) por E. O resultado será armazenado em T1 e não
              mais em E (item 2).
ADD   X,B     ;somar X por B, resultado em X (item 3)
SUB   X,T1    ;subtrair T1 de X, resultado em X (item 4)
MPY   X,A     ;multiplicar A por X, resultado final em X (item 5)
```


Instruções com Um Operando

Considerando as vantagens obtidas com a redução da quantidade de operandos (instruções menores), foram também criadas instruções de apenas um operando.

Com esse tipo, o acumulador (ACC) é empregue como operando implícito (não é necessário especificar o seu endereço na instrução, pois só há um ACC), guardando o valor de um dos dados e, posteriormente, o valor do resultado da operação.

```
ADD   Op.   ACC <- ACC + (Op.)
SUB   Op.   ACC <- ACC - (Op.)
MFY   Op.   ACC <- ACC (Op.)
DIV   Op.   ACC <- ACC (Op.)
```

Com o propósito de permitir a transferência de dados entre o ACC e a MP, foram criadas duas novas instruções:

```
LDA Op.      que significa   ACC <- (Op.)
STA Op.                      (Op.) <- ACC
```

Ainda o mesmo comando apresentado na equação X = A * (B + C * D - E/F) poderia ser convertido no programa Assembler, mostrado a seguir, constituído de instruções de um operando (sem destruirmos valor algum das variáveis).

```
LDA   C
MPY   D
STA   X
LDA   E
D1V   F
STA   T1
LDA   B
ADD   X
SUB   T1
MPY   A
STA   X
```

A comparação entre os diferentes programas (para instruções com 3 operandos, com 2 e com 1 operando) é apresentada na Tabela 5, a qual inclui, também, a quantidade

de memória despendida com cada programa, bem como a quantidade de acessos à memória (para ciclos de leitura ou de escrita) em cada caso.

Para calcular a quantidade de bits gastos em cada programa foram considerados um tamanho de código de operação de 8 bits (para todos os 3 tipos de instrução) e uma MP com capacidade de armazenamento de 1M células, sendo, portanto, cada campo operando de 20 bits (endereço do dado), visto que 2^{20}) = 1M.

A Tabela 5 apresenta o tamanho em bits de cada tipo de instrução, bem como a quantidade de ciclos de memória (de acessos) que são consumidos em cada uma delas.

Os programas apresentados na Tabela 6 indicam que, em termos de gasto de memória, instruções de 3 operandos não servem, razão por que foram abandonados pelos fabricantes há muito tempo. Usualmente, as instruções de dois operandos são mais bem empregues em operações matemáticas (aritméticas e lógicas), utilizando-se também instruções de 1 operando em outros casos, como instruções de desvio.

Instrução de 3 operandos	C.Op. = 8 bits + 3 operandos de 20 bits cada um = 68 bits Ciclos de memória = 4 (um para procurar a instrução e 3 para cada operando)
Instrução de 2 operandos	C.Op. = 8 bits + 2 operandos de 20 bits cada um = 48 bits Cicios de memória = *4* (um para procurar a instrução e 3 para cada operando)
Instrução de 1 operandos	C.Op. = 8 bits + 1 operando de 20 *bits* = *28* bits Ciclos de memória = 2 (um para procurar a instrução e 1 para o operando)

Tabela 5: Tamanho e Consumo de Tempo de Execução de Instruções de 3, de 2 e de 1 Operando

Com instruções de 3 operandos	Com instruções de 2 operandos (sem salvamento)	Com instruções de 2 operandos (com salvamento)	Com instruções de 1 operando

MPY C, D, Tl
DIV E, F, T2
ADD D, T l, X
SUB X, T2, X
MPY A, X, X
	MPY C, D
	DIV E,F
	ADD B, C
	SUB B, E
	MPY A, B
	MOVE X, A
		MOVE X, C
		MPY X, D
		MOVE T l, E
		DIV T l, F
		ADD X, B
		SUB X, Tl
		MPY X, A
			LDA C
			MPY D
			STA X
			LDA E
			DIV F
			STA Tl
			LDA B
			ADD X
			SUB Tl
			MPY A
			STA X

Espaço: 340 bits Tempo: 20 acessos	Espaço: 288 bits Tempo: 24 acessos	Espaço: 336 bits Tempo: 28 acessos	Espaço: 308 bits Tempo: 22 acessos

Tabela 6: Programas Assembly para Solucionar a Equação: X = A (B + C * D — E/F)*

Na verdade, há ainda muita controvérsia em relação ao projeto e implementação do conjunto de instruções de um processador, não existindo, de modo nenhum, unanimidade para os diversos tópicos, tais como tamanho da instrução, formato e significado do campo operando.

Instruções de poucos operandos ocupam menos espaço de memória e tomam o projeto do processador mais simples em virtude das poucas ações que elas induzem a realizar. No entanto, o programa gerado em binário é algumas vezes maior devido ao aumento da quantidade de instruções (não é o caso do exemplo apresentado na Tabela 6).

Instruções de 1 operando são simples e baratas de implementar, porém somente utilizam um único registador (o ACC) e, com isso, reduzem a flexibilidade e velocidade de processamento (a utilização de mais de um registador acelera o processamento devido à velocidade de transferência desses dispositivos).

Instruções com mais de 1 operando podem usar tanto endereços de memória como registadores no seu formato.

Modos de Endereçamento

Já foi descrito o formato básico de instruções de máquina e o ciclo de execução de cada instrução, concluindo que:

- O endereçamento de uma instrução é sempre realizado através do valor armazenado no contador de instrução (CI). Todo ciclo de instrução é iniciado pela transferência da instrução para o RI (usando-se o endereço contido no CI).
- Toda instrução consiste numa ordem codificada (código de operação) para a UCP 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 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 de ser endereçado por ser único na UCP).

Todos os exemplos apresentados até este ponto definiram o campo operando como contendo o endereço da MP onde está localizado o dado referido na instrução. No entanto, essa não é a única maneira de indicar a localização de um dado, havendo outros *modos de endereçamento.*

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

Conforme será demonstrado a seguir, há instruções em que é ineficiente usar o dado armazenado na MP, como, por exemplo, no caso de um contador, o qual tem um valor fixo inicial e, durante a execução do programa, é sistematicamente atualizado. Nesse caso, melhor seria se o referido contador (dado) fosse inicialmente transferido para um registador disponível na UCP e lá permanecesse (sendo diretamente atualizado na UCP) até ao final da execução do programa, em vez de ir da MP para a UCP e vice-versa (para atualização de seu valor), o que acarreta um considerável gasto de tempo para os repetidos ciclos de leitura e gravação.

Por outro lado, a manipulação de vetores acarreta a necessidade de se estabelecer um método eficaz de endereçamento para variáveis que ocupam posições contíguas de memória. E assim por diante. Dentre os diversos modos de endereçamento atualmente empregados, os principais são:

- Imediato;
- Direto;
- Indireto;
- Por registador;
- Indexado;
- Base mais deslocamento.

Modo Imediato

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

Assim, o dado é transferido da memória juntamente com a instrução (para o RI), visto estar contido no campo operando da instrução.

Este modo, denominado imediato, é útil para inicialização de contadores (um valor sempre fixo em toda a execução do mesmo programa); na operação com constantes matemáticas; para armazenamento de ponteiros em registadores da UCP; ou para indicação da quantidade de posições em que um determinado número será deslocado para a direita ou para a esquerda (em operações de multiplicação e divisão).

Uma das suas desvantagens consiste 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 (dado de valor diferente).

Praticamente, todo computador possui uma ou mais instruções que empregam o modo de endereçamento imediato: para instruções de desvio; de movimentação de um dado; para operações aritméticas com uma constante, etc.

Por exemplo, os processadores da família x86 possuem, entre outras, a instrução:

MOV R , Op.,

que pode ser usada da seguinte forma:

MOV AL, 22H (copiar o valor hexadecimal 22 para o registrador AL, sendo o valor de tamanho igual a 1 byte).

Com esta mesma instrução é possível copiar um valor de 32 bits. Por exemplo:

MOV EBX, 33445566H

(copiar o valor hexadecimal 33445566 para o registrador de 32 bits EBX).

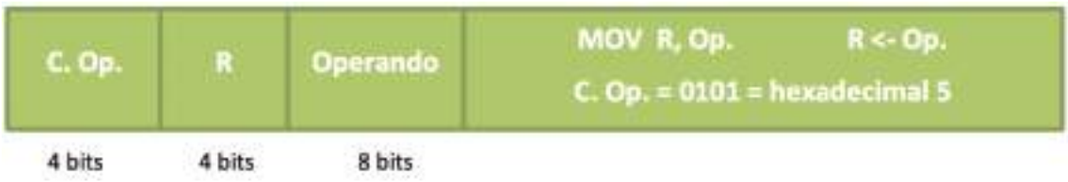
Exemplo 1

JMP Op. CI <- Op.

C. Op. = 1010 = hexadecimal A

Instrução: 101000110101 ou A35 (C. Op. = A e Operando = 35) --- Armazenar o valor 35 no CI.

Exemplo 2



Instrução: 0101001100000111 ou 5307 (C. Op. = 5, R = 3 e Operando = 07) – Armazenar o valor 07 no registador de endereço 3 (R3).

Modo Direto

Nesse método, o valor binário contido no campo operando da instrução indica o endereço de memória onde se localiza o dado. Tem sido o modo empregue nos nossos exemplos anteriores.

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 MP para procurar o dado, sendo, porém, mais lento que o modo imediato, devido naturalmente à referência à memória.

Quando um dado varia de valor a cada execução do programa, a melhor maneira de utilizá-lo é, inicialmente, armazená-lo na MP (do dispositivo de entrada para a memória). O programa, então, usa o dado pelo modo direto, em que a instrução indica apenas o endereço onde ele se localiza.

Uma possível desvantagem desse processo está na limitação de memória a ser usada, conforme o tamanho do campo operando. Isto é, se o campo tiver um tamanho, por exemplo, de 12 bits, com a utilização do modo direto, somente se pode aceder às células de endereço na faixa de 0 a 4095 (decimal), correspondentes aos valores binários 000000000000 a 111111111111.

Atualmente, como o espaço de endereçamento de MP tem vindo a crescer bastante (usa-se MP com espaço de endereçamento da ordem de 32M células, 64MB e até 256MB), não é desejável criar instruções com campo operando de tantos bits — para endereçar 64M células seriam necessários 26 bits para endereço direto.

Os processadores da família Intel x86 possuem instruções no modo direto, uma das quais, do tipo MOV R, Op., que pode ser implementada copiando até 4 células contíguas de memória, 32 bits, para um registador.

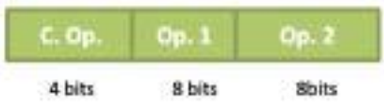
Exemplo 3

a)

C. Op. = 7 LDA Op. ACC <- (Op.)

Após a execução da instrução, o ACC conterá o valor 05A.

b)



C. Op. = B ADD Op.1, Op.2 (Op.1) <- (Op.1) + (Op.2)

Instrução: B55C3B

MP
“
“
“
05A
“
“
“
~~103~~ 15D
“
“
“

Somar o dado de valor binário 000100000011 ou hexadecimal 103 armazenado na célula de endereço hexadecimal 5C (Op. 1) com o dado de valor binário 000001011010 ou hexadecimal 05A armazenado na célula de endereço 3B (Op.2) e armazenar o resultado (15D) na célula de endereço hexadecimal 5C (Op.1).

Modo Indireto

Neste método, o valor binário do campo operando representa o endereço de uma célula; mas o conteúdo da referida célula não é o valor de um dado (como no modo direto), 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 procurar 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 ("aponta" para o dado). O conceito de ponteiro de dado é largamente empregue em programação.

Com esse processo, elimina-se o problema do modo direto, de limitação do valor do endereço do dado, pois estando o endereço armazenado na memória (pode ocupar uma ou mais células), ele se estenderá ao tamanho necessário à representação do maior endereço da MP do sistema de computação em uso.

Exemplo 3

C.Op. = 4 LDA Op. ACC <- ((Op.))

Instrução: 474

	MP
74	05D
	"
5D	1A4
	"

74 é o endereço da célula cujo conteúdo (5D) é o endereço do dado (1A4). Após a execução, o valor 1A4 estará armazenado no ACC.

Há uma variação pouco empregue deste método, em que, em vez de um nível sem direção, são usados múltiplos níveis. Em lugar do valor do dado (caso de apenas um nível sem direção), armazena-se novo endereço, que acede a uma célula, que pode conter outro endereço, e assim sucessivamente.

Uma das implementações desta variação do modo indireto prevê a inclusão de um bit especial na instrução, o qual representa a continuação ou não do endereçamento. Se o valor desse bit for igual a 0, o endereçamento prossegue com novo acesso, enquanto o valor 1 significa o encerramento do processo, isto é, o endereço final do dado.

Este método é atualmente pouco empregue devido à quantidade de acessos à memória e à sua complexidade para programadores em linguagem Assembly.

Um dos possíveis usos (no passado) para o modo indireto é na manutenção de ponteiros de dados. Se tivermos uma relação de dados a serem movimentados para novas posições de memória (caso, por exemplo, de elementos de vetores) usa o modo indireto, basta apenas modificar o valor da célula endereçada no primeiro nível (campo do operando), isto é, modificar o endereço de acesso ao dado, sem alterar o valor do campo operando.

Como já mencionado anteriormente, a grande desvantagem deste método é, obviamente, a maior quantidade de ciclos de memória requerida para completar o ciclo da instrução, pois, para se aceder a um dado no modo indireto, é necessário efetuar dois acessos à memória (um para procurar o endereço do dado e outro para efetivamente procurar o dado).

Ainda sobre o processo de acesso do modo indireto, deve-se ter atenção para o fato de que, embora seja possível localizar qualquer endereço de MP (porque o endereço de efetivo acesso ao dado não está mais contido no campo operando da instrução, como no modo direto), num dado instante somente se pode aceder a endereços até ao limite do campo operando, pois esse, contendo o endereço de acesso ao ponteiro do dado (endereço do dado), limita o tamanho do maior valor de endereço.

A Tabela 7 apresenta um resumo comparativo da definição, vantagens e desvantagens de cada modo de endereçamento.

Modo de endereçamento	Definição	Vantagens	Desvantagens
Imediato	O campo operando contem o dado.	Rapidez na execução da instrução.	Limitação do tamanho do dado. Inadequado para o uso com dados de valor variável.
Direto	O campo operando contem o endereço do dado.	Flexibilidade no acesso a variáveis de valor diferente em cada execução do programa.	Perda de tempo, se o dado é uma constante.
Indireto	O campo operando contem o endereço do dado.	Manuseio de vetores (quando o modo indexado não esta disponível). Uso como “ponteiro”.	Muitos acessos a MP para execução.

Tabela 7: Quadro demonstrativo das características dos modos de endereçamento

Endereçamento por Registrador

Este 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 registadores da UCP. Com isso, o endereço mencionado na instrução passa a ser o de um dos registadores, e não mais de uma célula da MP. A primeira vantagem, logo observada, consiste no menor número de bits necessários para endereçar os registadores, visto que esses existem em muito menor quantidade que as células de memória. Isto reduz o tamanho geral da instrução.

Um computador que tenha uma UCP projetada com 16 registadores requer apenas 4 bits para endereça-los (cada um dos 16 registradores tem um endereço de 4 bits, de 0 a F16, por exemplo); no caso de endereçamento de células de MP, há necessidade de 20 ou mais bits para indicar o endereço de cada uma das células.

Outra vantagem está na própria utilização do dado, que passa a ser armazenado num meio (registador) cujo acesso é muito mais rápido que o acesso à memória.

Para mostrar, de modo mais objetivo, a utilidade e as vantagens do uso de registadores no endereçamento de instruções, vamos considerar a execução do conjunto de instruções mostrado a seguir (pode ser parte de um programa), através de dois modos diferentes: com a utilização do modo de endereçamento por registador e sem utilização desse modo de endereçamento.

```
DO I = 1 TO 100
READ A, B
X = A + B
END
```

O excerto de programa descrito executa 100 vezes o mesmo tipo de ação: ler dois valores e somá-los. Para implementar a sua execução direta é necessário definir uma variável inteira (chamamos de *contador*); após a execução de cada conjunto de instruções de leitura dos dados (Get) e de soma, o contador é incrementado de 1, até atingir o valor 100, quando a execução do excerto de programa se completa.

Em linguagem Assembly teríamos:

```
        GET L       ;ler valor do "loop" (no exemplo o valor é igual a 100)
        LDI 0       ;ACC <- 0
        SUBM L      ;ACC <- ACC - (L), no exemplo o valor é 100
In      STA I       ;(I) <- ACC (inicialmente I = 100)
        JZ Fim      ;se ACC = 0 vá para Fim
        GET         ;A ler valor do dado para o endereço A
        GET B       ;ler valor do dado para o endereço B
        LDA A       ;ACC <- (A)
        ADD B       ;ACC <- ACC + (B)
        STR X       ;(X) <- ACC
        LDA I       ;ACC <- (I)
        INC         ;ACC <- ACC + 1 (no exemplo, estamos a fazer I + 1)
        JMP In      ;vá para In
Fim     HLT         ;parar
```

Como podemos observar, as instruções LDA I e STA I manipulam com a leitura e gravação de cada um dos 100 valores assumidos por I durante a execução do programa; gastam-se, com isso, 200 ciclos de memória (100 de leitura de I — LDA Op. e 100 de gravação do novo valor de I — STA Op.).

Vamos agora executar o mesmo excerto do programa de outra forma. Desta vez vamos utilizar o modo de endereçamento por registador.

O objetivo é armazenar na UCP (num dos registadores disponíveis) o valor inicial de 1 e efetuar a soma I = I + 1, diretamente na UCP (sem necessidade de acesso à MP, visto que o valor de I permanece armazenado num dos registradores da UCP). Esse método irá economizar os 200 acessos gastos pelo processo anterior.

O novo excerto de programa em linguagem Assembly ficaria assim:

```
            LDI R1,1        ;R1 <- 1 (o registador recebe o valor inicial do contador,
                            que é 1)
            LDI R2,100      ;R2 <- 100 (o registador recebe o valor final do contador,
                            que é 100)
In          SUBR R1, R2     ;(R1) <- (R1) - (R2)
            JZ R, Fim       ;se R1 = 0 vá para Fim
            GET A           ;ler valor do dado para o endereço A
            GET B           ;ler valor do dado para o endereço B
            LDA A           ;ACC <- (A)
            ADD B           ;ACC <- ACC + (B)
            STR X           ;(X) <- ACC
            INC R1          ;(R1) <- (R1) + 1 (no exemplo, estamos a fazer I + 1)
            JMP In          ;vá para In
Fim         HLT             ;parar
```

As instruções LDI R1,1 e LD1 R2,100 armazenam em registadores, respetivamente, o valor inicial (1) e o valor final (100) do contador. No exemplo foram indicados os registradores R1 e R2, mas podem ser usados quaisquer registadores disponíveis.

As instruções SUBR R1,R2 e INC R1 serão executadas manipulando-se os valores armazenados dentro da UCP, nos registadores escolhidos, R1 e R2, cujos endereços constam da instrução (campo R); não há, portanto, acesso à MP.

Outra vantagem da utilização do modo de endereçamento por registador — economia de bits nas instruções — pode ser verificada pela seguinte comparação:

Instrução com uso de registadores

Instrução com acesso a 64K células de MP

Há duas maneiras de utilizar o modo de endereçamento por registador:

- Modo por registrador direto;
- Modo por registrador indireto.

No primeiro caso, o registador endereçado na instrução contém o dado a ser manipulado. No outro, o registador referenciado armazena o endereço de uma célula de memória onde se encontra o dado. A instrução conterá, como sempre, o endereço do registador.

A Fig. 21 mostra dois tipos de instruções que empregam o modo de endereçamento por registrador.

Na Fig. 21(a) observa-se que a instrução possui dois campos contendo, cada um, o *endereço* de um registador (um dos 16 possíveis), o qual poderá ter armazenado o dado (direto) ou o endereço da MP, onde estará o dado (indireto).

Na Figura 21(b), a instrução possui dois campos de operando, um dos quais é o endereço de um registador (que contém o dado) e o outro é o endereço de uma célula da MP, que armazena o dado (direto). Pode-se, ainda, ter uma variação desse formato, em que o campo operando pode conter o próprio valor do dado (imediato).

Embora o modo de endereçamento por registador seja vantajoso em vários aspetos, tais como rapidez de execução da instrução e economia de espaço de armazenamento das instruções, essas vantagens nem sempre são aplicáveis. Há certos casos em que não se

observa vantagem alguma no referido método, ocorrendo até desperdício de instruções, o que pode vir a constituir uma desvantagem.

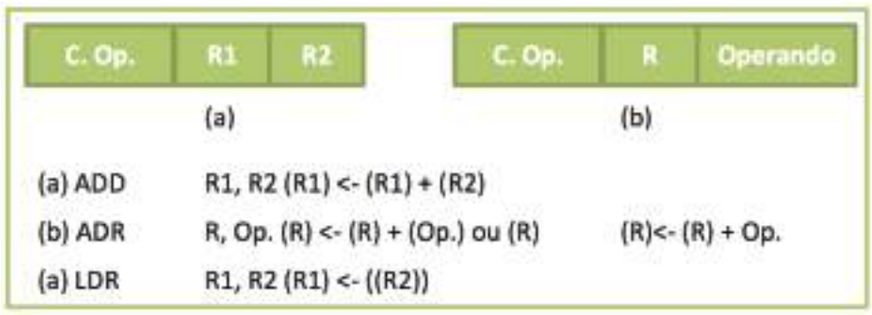


Figura 21: Exemplo de instruções com modo de endereçamento por registador

No exemplo utilizado, as instruções que efetivamente calculam a equação X = A + B não empregaram modo de endereçamento por registrador e sim o modo direto convencional. Não é eficaz usar registador apenas para realizar uma transferência do tipo:

MP -> R -> UAL

Como podemos verificar, os dados representados pelas variáveis A e B estarão, em cada uma das 100 execuções, sempre na MP (são lidos do meio exterior para a MP) e terão que ser passados para a UAL as 100 vezes. Assim, se utilizássemos algum registador para armazenar A ou B, o dispositivo serviria apenas para atrasar a execução da instrução com mais um armazenamento, sem se obter qualquer vantagem.

O uso do registador somente é vantajoso se proporcionar redução de ciclos de memória, o que não era o caso.

Uma outra possível desvantagem da utilização do modo de endereçamento por registrador consiste, em certos casos, na dificuldade em se definir quais os dados que serão armazenados nos registadores e quais permanecerão na MP (por falta de registadores). Isto acontece devido ao reduzido número dos registadores existentes nas UCPs e à grande quantidade de dados manipulados pelos programas.

Modo Indexado

Frequentemente, durante a execução de programas, há necessidade de se manipularem endereços de acesso a elementos de certos tipos especiais de dados. Esses endereços servem, na realidade, de ponteiros para os referidos elementos.

Por exemplo, o acesso aos elementos de um vetor (array) deve considerar que tais elementos são armazenados sequencialmente na memória e que a sua localização pode ser referenciada por um ponteiro (endereço), que é alterado para indicar o elemento desejado (o índice do elemento identifica individualmente cada um).

Portanto, é importante que haja, no conjunto de instruções de máquina, algumas capazes de realizar essas manipulações de endereços, permitindo uma localização dos dados mais rápida e eficiente.

A maioria dos processadores possui uma ou mais dessas instruções; a sua descrição caracteriza um modo de endereçamento denominado *indexado.* Essa denominação advém do fato de que a obtenção do endereço de um dado (elemento de um array) relaciona-se com o seu índice.

Neste tipo de instrução, o endereço do dado é a soma do valor do campo operando (fixo para todos os elementos de um dado array) e de um valor armazenado num dos registradores da UCP (normalmente denominado *registador índice).* O valor armazenado nesse registador varia para o acesso a cada elemento ("aponta para o elemento desejado).

Na verdade, esse modo de endereçamento é uma evolução das técnicas desenvolvidas desde primórdios da computação para manipulação dessas estruturas de dados especiais.

Podemos exemplificar essa assertiva apresentando alguns possíveis métodos, evolutivamente na manipulação de arrays, até atingirmos a eficiência do modo indexado.

Consideremos, por exemplo, a necessidade de, em certo programa, executar-se a seguinte operação sobre três vetores (arrays) de 100 elementos cada:

Prog-1 DO 1 = 1 TO 100

C(I) = A(I) + B(I)

Usando o Modo Direto sem Alterar os Bits que Descrevem as Instruções

Neste caso, haveria necessidade de se escrever instruções para cada uma das 100 operações de soma a serem efetivamente realizadas pela máquina. Exemplificando com as instruções de um operando adotadas nesse texto, teríamos um programa semelhante ao apresentado na Figura 22.

```
LDA   A(1)
ADD   B(1)
STA   C(1)
LDA   A(2)
ADD   B(2)
STA   C(2)
------------
------------
LDA   A(100)
ADD   B(100)
STA   C(100)
HLT
```

Figura 22: Programa 1, em linguagem Assembly.

Evidentemente, esta técnica de programação é ineficiente e trabalhosa, usando o computador apenas como calculadora, já que o programador terá toda a carga de trabalho.

Usando o Modo Direto com Alteração Dinâmica do Conteúdo das Instruções.

O cálculo da soma dos 100 elementos pode ser obtido por um programa com muito menor quantidade de instruções.

Nesse caso, utiliza-se o automatismo do computador para realizar a tarefa de executar 100 vezes as operações.

Por tanto, é necessário que se determine, em tempo de execução, o endereço de cada um dos 100 elementos dos vetores. Como esses elementos estão armazenados sequencialmente na MP, basta existir uma instrução que incremente o valor do campo operando (endereço do dado). Tal instrução executa uma operação aritmética (X = X + 1) sobre um valor binário (X), que é, na realidade, um endereço.

O programa, em Assembly, para resolver o algoritmo definido em Prog-1, seria semelhante ao mostrado na Tabela 8.

Programa Assembly	Programa em linguagem de máquina
T LDA A(1)	11A00
1 ADD B(1)	21A64
2 STA C(1)	31AC8
INC T	8103 A
INC 1	8103B
INC 2	8103C
DCR N	919FF
JNZ T	D103A
END	F0000

Tabela 8: Programa 2, em linguagens Assembly e de máquina

Observe a Fig. 23, na qual é apresentada uma MP com 64K células (cada célula é identificada por um endereço com 4 algarismos hexadecimais), todas com capacidade

de armazenar valores com 20 bits (5 algarismos hexadecimais) e instruções (de um operando), do tamanho da célula e da palavra.

Nessa memória foram armazenados os elementos do vetor A (a partir do endereço hexadecimal 1A00), do vetor B (a partir do endereço hexadecimal 1A64) e, a partir do endereço 1AC8, deverão ser armazenados os elementos do vetor C (resultado da soma). O excerto do programa (Tabela 8) começa a partir do endereço hexadecimal 103A (o conteúdo da célula tem armazenada a instrução cujo valor em hexadecimal é 11A00).

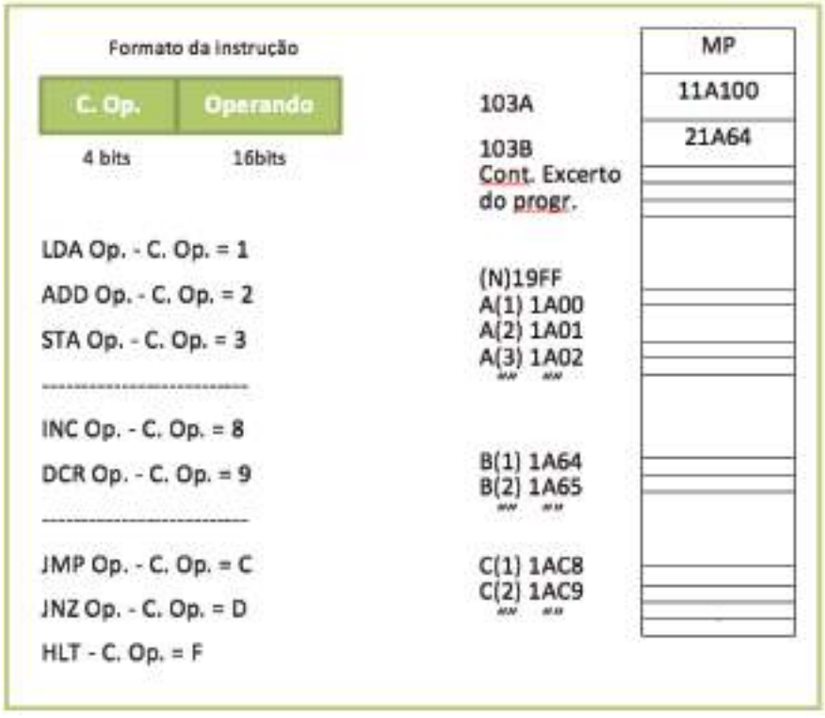


Figura 23: Programa e dados de uma soma de vetores no modo direto.

A primeira execução da instrução INC, código de operação igual a 8 (utilizando o modo de endereçamento imediato), acarreta a alteração do valor do campo operando das instruções LDA, ADD e STA, respetivamente, para: 1A01 (endereço do segundo elemento do vetor A, identificado por A(2)), 1A65 (endereço do elemento B(2) do vetor B) e 1AC9 (endereço do elemento C(2) do vetor C). Na passagem seguinte do loop, os valores seriam novamente alterados (somando 1 ao valor atual) para, respetivamente: 1A02, 1A66 e 1ACA, endereços dos elementos A(3), B(3) e C(3), dos vetores A, B e C e assim sucessivamente, até alcançar-se o endereço dos últimos elementos: A(100), B(100) e C(100).

O método é adequado e vantajoso, na medida em que são elaboradas apenas *nove* instruções (é claro que há outras maneiras de se fazer o mesmo programa), e o trabalho de execução fica por conta da máquina, não do programador.

A desvantagem deste método, porém, reside no fato de que o valor de uma instrução (o campo operando) é alterado durante a execução do programa. Caso houvesse alguma interrupção anormal no meio da execução, seria preciso reinicializar os valores iniciais, bem como a reinicialização dos valores teria que ser realizada em toda nova execução do programa, pois ele termina com os valores de endereço de A(100), B(100) e C(100).

Uso do Modo Indireto

Esta é uma das aplicações do modo indireto, pois o endereço de cada elemento do vetor estará armazenado na MP, numa célula cujo endereço consta no campo operando da instrução.

O processo de obtenção do endereço de cada elemento, durante a execução do programa, consiste na alteração (adicionando-se um valor que aponte para o novo elemento) do conteúdo da posição de memória endereçada, no modo indireto, pelo campo operando da instrução. Assim, não há modificação das instruções em tempo de execução (um dos problemas do método anterior).

A Figura 24 apresenta um exemplo dessa técnica. Os elementos de cada vetor estão armazenados nos mesmos endereços da figura anterior. As instruções LDA, ADD e STA utilizam o modo indireto, enquanto INC usa o modo direto (para incrementar os endereços dos elementos dos vetores).

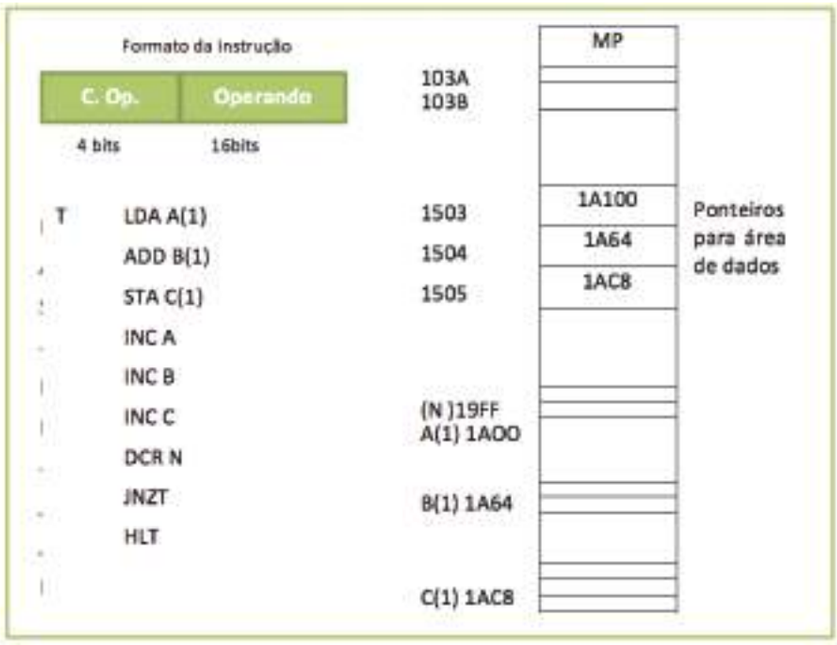


Figura 24: Soma de vetores (programas e dados) usando o modo indireto.

O programa é basicamente semelhante ao do exemplo anterior (modo direto), exceto que a primeira execução das instruções INC somaria 1 aos valores 1A00, 1A64 e 1AC8 (respetivamente, conteúdo das células de endereços 1503, 1504 e 1505). Isso permitiria apontar para os elementos A(2), B(2) e C(2).

Outra diferença fundamental em relação ao método anterior é que, nesse caso, os valores de endereços dos elementos não fazem parte das instruções; estão armazenados na MP e, portanto, as instruções não são alteradas durante a execução do programa.

Essa é uma técnica mais limpa, e eficiente, em termos de programação, embora haja um gasto maior de tempo na execução do programa, devido à maior quantidade de acessos do modo indireto.

Usando o Modo Indexado

Neste modo, o endereço de cada elemento do vetor é obtido pela soma (a soma é efetuada antes da colocação do valor do endereço de acesso no REM) do valor do campo operando da instrução com o valor armazenado num dos registadores da UCP (escolhido como registador-índice). O resultado da soma é o endereço efetivo do dado, o qual será, então, transferido para o REM. A escolha de qual registador será utilizado como registador-índice depende da linguagem que se estiver a usar: caso a linguagem

seja Assembly, é responsabilidade do programador administrar o uso dos registadores da UCP; se, por outro lado, for usada uma linguagem de alto nível, a escolha de uso dos registadores fica por conta do programa compilador.

Para utilizar este modo, é necessário haver instruções que manipulem valores em registadores, tais como:

- Carregar um valor no registrador (armazenar o índice);
- Somar um dado valor ao existente no registrador;
- Desviar para outra instrução, se o valor armazenado no registador for igual a zero (permite sair de um "looping" de execução), e assim por diante.

A grande vantagem da técnica reside na rapidez de execução das instruções de acesso aos dados, visto que a alteração do endereço dos elementos é realizada na própria UCP. Muitos computadores modernos possuem instruções no modo indexado, cujo exemplo de utilização é apresentado nas Figuras 25 e 26 (ainda a mesma soma dos vetores A, B e C, dos métodos anteriores — Prog-1).

```
        MVI (R4), 1
        MVI (R2), 100
T       LDA (R4), 19FF
        ADD (R4), 1A63
        STA (R4), 1AC7
        INC (R4)
        DCR (R2)
        JZR (R2), T
        HLT
```

Figura 25: Programa 3, em linguagem Assembly

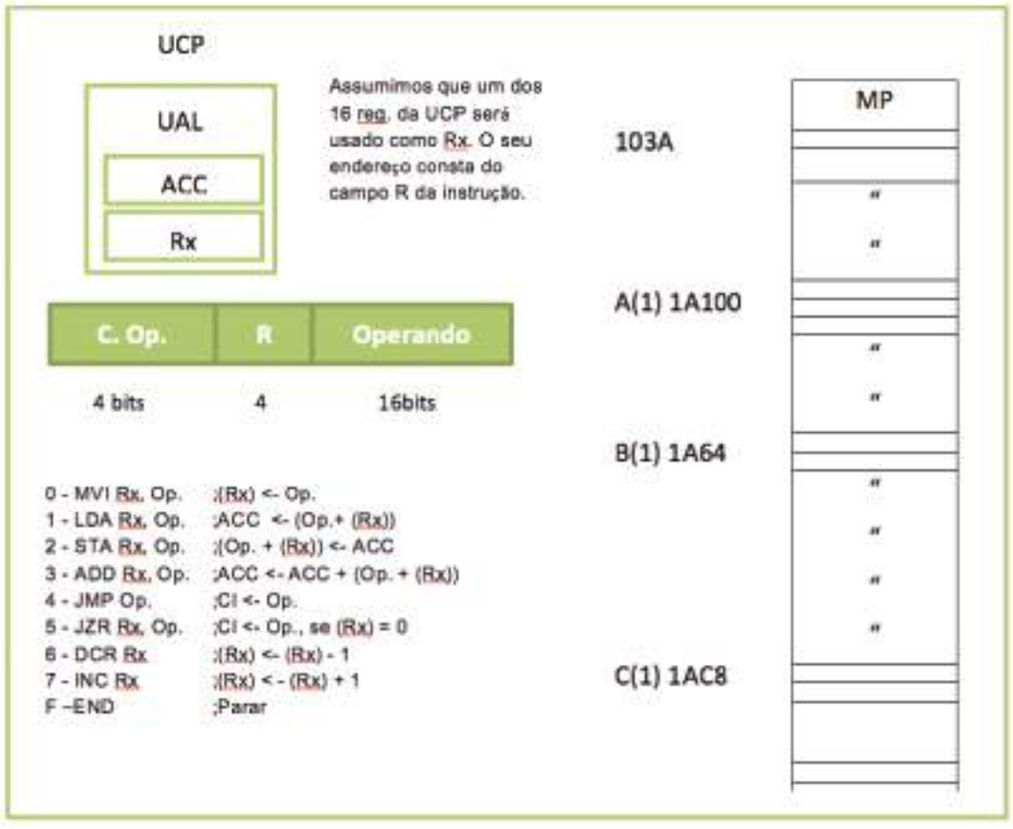


Figura 26: Exemplo de emprego do modo indexado.

Em primeiro lugar, move-se (MVI) o valor inicial do índice para o registador escolhido como Rx (foi escolhido como exemplo o registrador R4) e, assim, teremos a instrução, em linguagem de máquina:

040001_{16}

Em seguida, inicia-se o excerto repetitivo do programa, executando o teste de verificação da saída do looping (quando se atingir a soma do centésimo elemento de cada um dos vetores, A e B).

A soma propriamente dita é obtida pela repetição de:

LDA Rx, Op. ;Começa a carregar A(1)

ADD Rx, Op. ;Somar A e B

STA Rx, Op. ;Armazenar resultado em C

Em primeiro lugar, o endereço de A(1), (1A00), é indicado na instrução pela soma do campo Op. (neste caso seria, por exemplo, o valor hexadecimal 19FF) com o conteúdo de Rx (neste caso, o conteúdo de R4, que deverá ser 0001). O resultado dessa soma,

1A00, será, então, transferido para o REM, iniciando o ciclo de leitura do elemento A(1) do vetor A.

A obtenção do valor do elemento de B(1) do vetor B, cujo endereço em hexadecimal é 1A64, também será realizada pela soma do valor 0001 (armazenado em R4) com o valor 1A63 (valor constante do campo Op. — sempre o mesmo em todas as instruções que manipulem elementos do vetor B), e assim também será efetuado para o cálculo do endereço dos elementos do vetor C.

O programa prossegue, com o incremento do valor armazenado em R4 (Rx), de modo a permitir a soma do elemento A(2) com o elemento B(2). A instrução seria:

INC Rx.

Neste caso, o valor de R4 passaria de 0001 para 0002, o que, somado ao valor do campo Op. apontaria para o segundo elemento de cada vetor.

Em seguida, retorna-se ao ponto inicial, para testar se o loop já foi executado 100 vezes. Caso negativo, a soma é reiniciada: o valor 19FF será somado ao valor 0002 (para endereço de A(2)), bem como 1A63 e 1AC7 com 0002, respetivamente, para endereços de B(2) e C(2). E assim, sucessivamente, até o final.

Modo Base Mais Deslocamento

Este modo de endereçamento tem características semelhantes ao modo indexado, visto que o endereço de acesso a uma célula de memória se obtém pela soma de dois valores, um inserido no campo apropriado da instrução (normalmente denominado campo deslocamento — *displacement)* e o outro valor inserido num determinado registador, denominado registador-base ou registador de segmento.

A diferença entre eles está na aplicação e no propósito do método e, por conseguinte, na forma de implementá-lo. Neste caso, o valor a se manter fixo é o do registador-base/segmento, variando o conteúdo do campo deslocamento em cada instrução, diferentemente do modo indexado, em que o conteúdo do registador é que se altera.

Os processadores da família Intel x86 possuem alguns registadores projetados especificamente com a finalidade de servir como registador de segmento, como os 6 registadores de 16 bits dos processadores Pentium.

Este modo de endereçamento acarreta uma redução do tamanho das instruções (e, com isso, economiza memória), bem como facilita o processo de relocação dinâmica de programas.

A sua escolha decorre de dois fatores:

- Durante a execução de uma grande quantidade de programas, as referências a células de memória, onde se localizam os operandos, normalmente são sequenciais, ocorrendo poucos acessos a outras instruções fora de ordem (exceto desvios);
- A maioria dos programas ocupa um pequeno espaço da MP disponível.

Dessa forma, em vez de ser necessário, em cada instrução, que o campo operando tenha um tamanho correspondente à capacidade total de endereçamento da MP, basta que o endereço desejado seja obtido pela soma de um valor existente em um dos registradores da UCP com um valor contido na instrução.

Por isso o método é chamado de *base mais deslocamento,* consistindo, então, na utilização de dois campos na instrução (que substituem o campo operando): um, com o endereço de um registador (chamado de *base* ou *segmento),* e outro, com um valor denominado *deslocamento* (porque contém um valor relativo — que se desloca em relação — à primeira instrução).

Consideremos, por exemplo, o caso de processadores, nos quais as instruções possuem campo de endereço de registador-base com 4 bits (estipulamos no exemplo que o processador possui 16 registadores e, portanto, será necessário definir-se 16 endereços, um para cada registador) e campo deslocamento, com 12 bits. E que o processador pode endereçar até 16M células (cada endereço linear de memória deverá ter 24 bits).

Nesse caso, podem-se endereçar áreas de 4096 bytes (4K) com um valor armazenado no registador-base, gastando-se apenas 16 bits (4 + 12), ao contrário dos 24 bits necessários para endereçar diretamente todas as células da MP daquele computador (capacidade máxima da MP igual a 16 Mbytes). Economizam-se, assim, 8 bits em cada instrução.

A Figura 27 apresenta um exemplo desse modo de endereçamento, usando quatro registadores-base e 12 bits para o campo deslocamento. Pode-se observar então que, num dado instante, há quatro áreas de endereçamento, cada uma correspondente ao valor armazenado em cada registador-base.

Da descrição dos modos *indexado* e *base mais deslocamento,* podemos observar que o processo de cálculo do efetivo endereço de acesso é idêntico em ambos os modos. A diferença está no conceito de cada um, não na sua implementação.

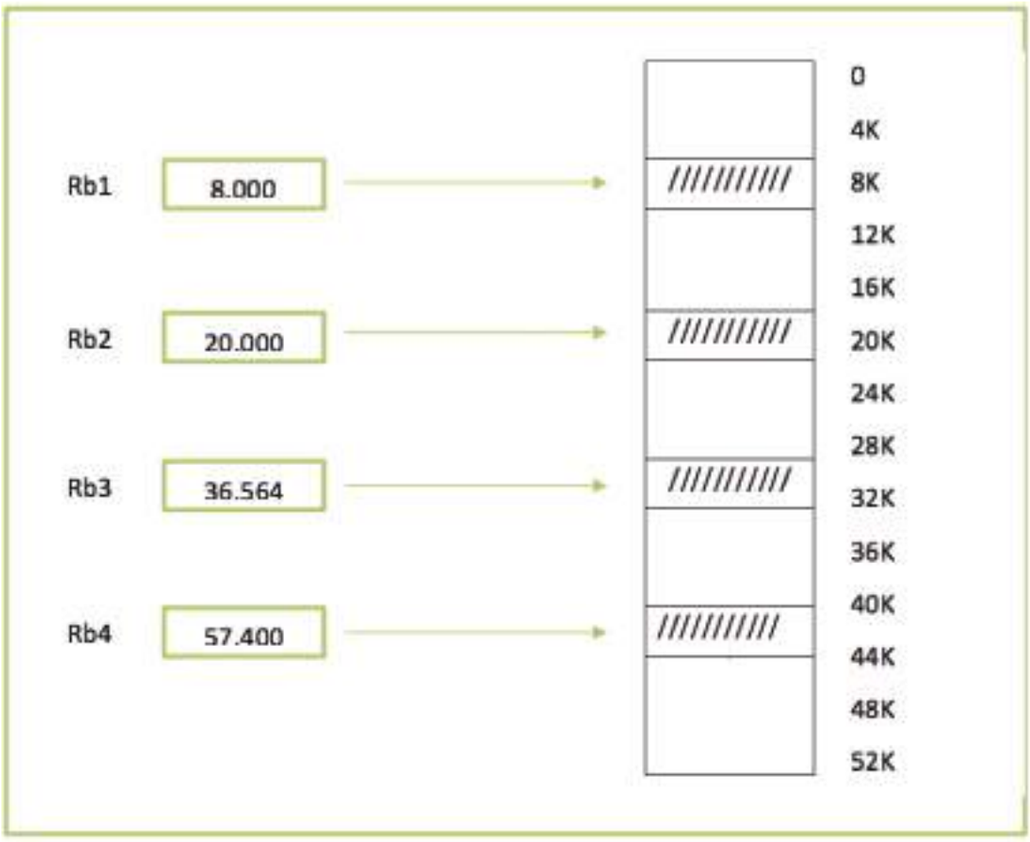


Figura 27: Exemplo de uso do modo base mais deslocamento.

A indexação é empregue quando se deseja aceder a diferentes dados, com alteração do endereço, por incremento (ou decremento) do valor do registador-índice. Quando a modificação de endereço é realizada para relocação de programa, basta uma única alteração do conteúdo do registador-base (no modo base mais deslocamento).

Sobre o que realmente acontece na máquina, podemos observar:

Vários dados são acedidos com diversos valores de registador-índice e um único valor no campo operando;

Vários dados são acedidos com um único valor de registador-base e valores diferentes no campo deslocamento da instrução.

Questionário

1. Cite uma possível vantagem da utilização de instruções com menor quantidade de operandos.

2. Crie um conjunto de instruções de dois operandos, definidas em linguagem Assembly, necessárias para a realização de operações aritméticas e elabore programas para cálculo das seguintes equações:
   a. X = A + (B * (C - A) + (D - E/B) * D)
   b. Y = (A + B * (C — D * (E/(B - F)) + B) * E)

3. Considere as instruções Assembly de um operando utilizadas neste módulo e escreva programas para resolver as equações apresentadas no exercício anterior.

4. Faça o mesmo para instruções de dois operandos.

5. Cite uma aplicação em programa para o modo de endereçamento imediato. Indique uma desvantagem desse modo.

6. Faça o mesmo para o modo de endereçamento direto.

7. Analise os modos de endereçamento direto e direto por registador, estabelecendo diferenças de desempenho, vantagens e desvantagens de cada um.

8. Considere as instruções definidas a seguir (de um operando):

LDA Op.	ACC <- (Op.)	STA Op.	(Op.) <- ACC
ADD Op.	ACC <- ACC + (Op.)	SUB Op.	ACC <- ACC - (Op.)
MPY Op.	ACC <- ACC * (Op.)	DIV Op.	ACC <- ACC/(Op.)

Deduza a equação matemática cuja solução resultou no seguinte programa, criado com estas instruções:

```
LDA A
ADD C
STA X
LDA B
MPY D
SUB E
STA Y
LDA X
ADD Y
DIV F
STA X
```

9. Qual é o objetivo da utilização do modo de endereçamento base mais deslocamento? Qual é a diferença de implementação entre esse modo e o modo indexado?

10. Em um determinado processador, há instruções que usam o modo de endereçamento base mais deslocamento, cada uma possuindo um tamanho de X bits. Desses X bits, a bits identificam o código da operação; b bits especificam o endereço do registador usado como base; c bits são utilizados para o campo deslocamento. Considerando que a barra de endereços possui y bits, que fração da MP pode ser endereçada sem que sejam alterados os conteúdos dos registradores-base existentes nesse processador?

11. Considere um processador que possua as seguintes características:
    - um registrador de 8 bits;
    - um registrador de 16 bits;
    - uma barra de dados de 8 bits;
    - uma barra de endereços de 16 bits.

Defina instruções que permitam ao processador carregar em um registador o conteúdo do endereço dado, adicionar a um registador um valor especificado e carregar no registador A o conteúdo da posição de memória apontada pelo registador B. Descreva cada instrução e caracterize o tipo de endereçamento que ela utiliza. Com essas instruções faça um programa que permita carregar no registrador A o elemento de ordem 1F de uma tabela que começa na posição de memória de endereço 013D e que gasta 8 bits de memória para cada elemento.

12. Considere um computador com UCP constituída de um RI com 24 bits, Cl e REM de 12 bits, UAL, UC e vários registadores de utilização geral. Esse computador possui um conjunto de 256 instruções de formato único, mostrado a seguir, e modos de endereçamento: *direto, indireto* e *por registrador.*

C. Op.	R1	R2	Operando

a. Quantos registadores de utilização geral podem ser endereçados nesse processador?
b. Supondo duas instruções A e B, onde a instrução A acede a MP no modo indireto e a instrução B acede a MP no modo por registador (modalidade indireta), qual delas executa o seu ciclo de instrução mais rápido? Porquê?

Dispositivos de entrada e saída

Introdução a dispositivos de entrada e saída

De acordo com o modelo previsto por Von Neumann e apresentado ao longo das seções anteriores, um computador deve ser capaz de armazenar dados e instruções necessários para a execução de uma tarefa, na memória e no formato binário. Para que possam ser executados, esses dados e instruções são procurados pela UCP diretamente na memória. É na memória também que o resultado desse processamento será disponibilizado. Como esses dados e instruções chegaram à memória? Como o resultado do processamento desse conteúdo mantido na memória é retornado ao utilizador?

A resposta para essas perguntas é que são necessários elementos que permitam a interface do utilizador com o computador, tanto para dar a entrada de dados e instruções quanto para proporcionar a saída de resultados ao utilizador, no formato adequado, tal qual foi solicitado.

Esses elementos de interface podem ser chamados de dispositivos (ou periféricos) de entrada e saída (E/S). São considerados um subsistema de memória, pois fazem parte do sistema maior que é o sistema computacional, onde os dispositivos de E/S compõem o chamado subsistema de E/S, o qual, segundo, deve ser capaz de realizar duas funções:

- Receber ou enviar informações ao meio exterior;
- Converter as informações (de entrada e saída) numa forma compreensível para a máquina (se estiver a receber) ou para o programador ou utilizador (se estiver a enviar).

Dentre os diversos dispositivos de E/S podemos citar: teclado, rato, monitor de vídeo, impressora, *webcam*, modem, dispositivos de armazenamento (ex.: disco rígido, CD/DVD – ROM, *pen-drive*). Estes dispositivos interligam-se à UCP e memória principal através do barramento de expansão e podem ser classificados em duas categorias: entrada (teclado, rato, webcam, modem, disco rígido) e saída (impressoras, disco rígido, monitor de vídeo). Sublinhamos que o disco rígido, assim como outros meios de armazenamento, são dispositivos tanto de entrada como de saída.

Lembramos que todos os estímulos gerados pelos dispositivos de E/S, por exemplo - a pressão de uma tecla, são convertidos em vários sinais elétricos, com diferentes intensidades, podendo representar os valores 0 ou 1.

Podemos destacar algumas observações relevantes que influenciam na comunicação dos dispositivos com a UCP e memória principal. São elas:

- Os dispositivos de E/S apresentam diferentes características, o que tornaria a comunicação entre a UCP e o periférico extremamente complicada, caso esta fosse realizada direta e individualmente (ex.: comunicação direta entre UCP e teclado, entre UCP e vídeo). Isso ocorre em função da grande diferença de velocidade entre UCP e os dispositivos de E/S, além de haver grandes diferenças de velocidade entre os próprios dispositivos, como por exemplo o disco rígido, que é mais rápido que o teclado. A Tabela 9 apresenta alguns exemplos de dispositivos de E/S e a sua velocidade de transmissão de dados aproximada.

Dispositivo	Taxa de Transmissão (KB/s)
Teclado	0,01
Rato	0,02
Impressora Matricial	1
Modem	2 a 8
Impressora Laser	200
Scanner	400
CD-ROM	1000
Rede Local	500 a 6000
Placa Gráfica	60000
Disco Rígido (HD)	2000 a 10000

Tabela 9: Dispositivos de E/S e a sua velocidade de transmissão de dados

Além da velocidade, outro aspeto que diferencia os dispositivos de E/S é a sua forma de comunicação. A comunicação entre o núcleo do computador e os dispositivos de E/S poderia ser classificada em:

- Comunicação serie: a informação pode ser transmitida/recebida, *bit* a *bit*, um a seguir ao outro, conforme a Figura 28.

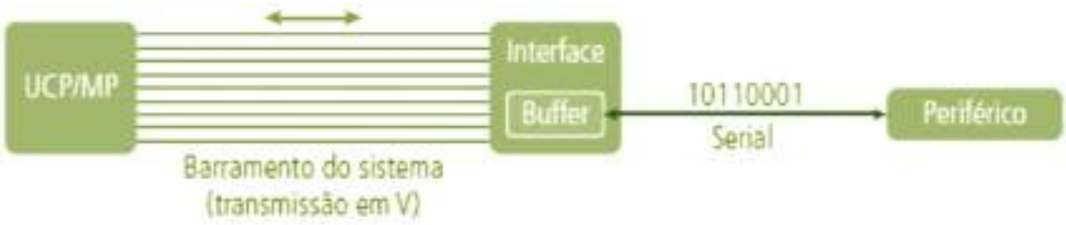


Figura 28: Transmissão serie entre interface e periférico

- Comunicação paralela: a informação pode ser transmitida/recebida em grupos de *bits* de cada vez, isto é, um grupo de bits é transmitido simultaneamente de cada vez, conforme apresenta a Figura 29.

Figura 29: Transmissão paralela entre transmissor (Interface) e recetor (periférico)

Existem diferenças relativas à parte elétrica de geração e interpretação dos sinais de transmissão. Podemos afirmar que, devido a essas diferenças, na prática a UCP não se liga diretamente com cada periférico, mas sim com dispositivos que realizam a "tradução" e a compatibilização das características de um (dispositivo de E/S) para o outro (UCP/MP), além de realizar outras tarefas de controlo.

Os dispositivos possuem algumas denominações. A mais comum seria interface de E/S, mas comercialmente podem ser encontrados como controlador ou adaptador, adicionando-se o nome do dispositivo (ex.: controlador de vídeo, controlador de disco). A função de todos é sempre a mesma: compatibilizar as diferentes características de um periférico e da UCP/memória principal, permitindo um fluxo correto de dados numa

velocidade adequada a ambos os elementos que estão sendo interligados. A Figura 30 apresenta exemplos de interfaces.

Figura 30: Exemplos de controladores de E/S

Metodologias de comunicação entre UCP e dispositivos de E/S

Num sistema de computação há a necessidade de que a UCP se comunique com a memória principal (RAM) e com os dispositivos de E/S (ex.: teclado, *rato*, monitor de vídeo, rede) para a transferência de dados. Semelhante ao que ocorre com a comunicação entre UCP e memória principal, na qual são definidos endereços para cada posição de memória, os quais são referenciados pela UCP, quando se trata de comunicação entre UCP e dispositivos, torna-se necessário que a UCP indique um endereço que corresponda ao periférico em questão.

Cada periférico acoplado ao sistema de computação possui um endereço, o qual é identificado como endereço de porta de E/S, ou seja, cada dispositivo de E/S possui um número de identificação única no sistema computacional onde se encontra. Dessa forma, se o endereço de porta ou endereço de E/S for um número de oito *bits*, significa que poderão ser conectados até 256 (28) dispositivos ao sistema.

Diversas formas de comunicação entre UCP e memória principal foram propostas, as quais sofreram melhorias ao longo do tempo, procurando sempre alcançar uma melhor utilização da UCP e um melhor desempenho para o sistema como um todo. Podem ser destacados três métodos para gerir a entrada e saída:

- *Entrada e saída programada*

Neste método, também chamado de *pooling*, a UCP precisa verificar continuamente se cada um dos dispositivos precisa de atendimento, ou seja, tudo depende da UCP. Por exemplo, se o disco quer transferir algum dado para a memória, a UCP deve ficar dedicada a esse processo de transferência até que esta seja concluída. A grande desvantagem é a subutilização da UCP, a qual, enquanto houver uma operação de transferência de dados, não realiza outras operações de processamento. Este método não já não se utiliza.

- *Entrada e saídas controladas por interrupção.*

Este método possibilita que a UCP não fique presa em espera ocupada até que um dispositivo esteja pronto para realizar a transferência de dados propriamente dita. Assim, a UCP dá início à operação emitindo uma instrução de E/S para a interface ou controlador do dispositivo em questão e, quando o dispositivo estiver pronto para a operação de transferência, recebe uma interrupção a avisar que ela poderá começar. Essa demora para iniciar a transferência, após as instruções da UCP, deve-se à lentidão dos dispositivos de E/S em relação à UCP, problema que é diminuído com o uso de interrupções. Este método sofreu melhorias e também já não é utilizado.

- *Acesso direto à memória (DMA)*

A função do controlador (ou interface) é controlar o seu dispositivo de E/S e manipular para ele o acesso ao barramento. Quando um programa quer dados do disco, por exemplo, ele envia um comando ao controlador de disco, que então emite comandos de procura e outras operações necessárias para que ocorra a transferência.

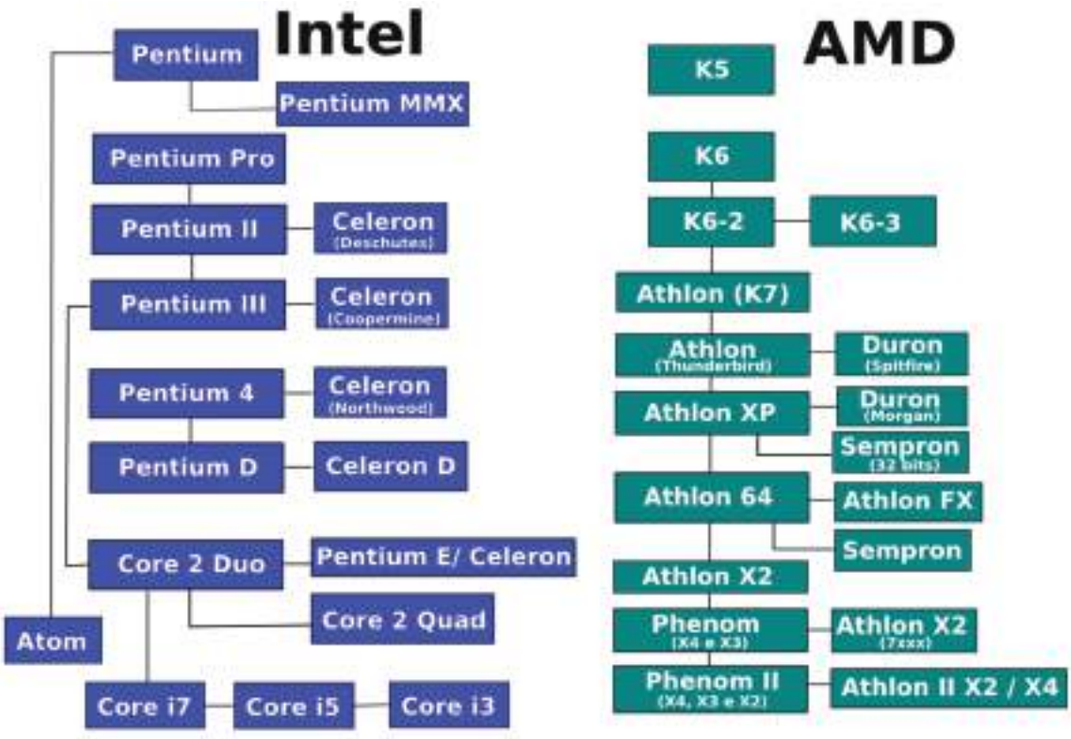
Quando o controlador lê ou escreve dados de/para a memória sem a intervenção da UCP, é dito que ele está a executar um acesso direto à memória (*Direct Memory Access*), conhecido por DMA. Dessa forma, a UCP limita-se a solicitar a transferência para um dispositivo denominado controlador de acesso direto à memória principal (DMA *Controller*), o qual se responsabiliza totalmente pela transferência. A UCP é avisada apenas no início e no final da operação de transferência entre dispositivo e memória principal. Este método é utilizado atualmente pelos computadores.

Destacamos que a UCP apresenta apenas uma linha para receber pedidos externos de interrupções de todos os dispositivos de E/S. Dessa forma, a fim de partilhar essa linha

com os diversos dispositivos existentes num computador, foi desenvolvido o controlador de interrupções, o qual recebe e encaminha, de acordo com a prioridade (definida para evitar conflitos de acesso à UCP), os pedidos de interrupção feitos pelos diferentes dispositivos de entrada/saída. Quando a UCP recebe o pedido de interrupção, verifica com o controlador de interrupções que dispositivo causou o pedido e então efetua o atendimento apropriado à interrupção.

Anexo I

Evolução dos processadores INTEL e AMD



Bibliografia

SAMPAIO, A., *Hardware para profissionais*. Lisboa: FCA, sd.

SAMPAIO, A., Microcomputadores: *Circuitos Internos e Programação*. Queluz: Edições EPGE, 1993.

TOKHEIM, Roger L., *Introdução aos Microprocessadores*. S. Paulo: McGrawHill, 1985.

VELOSO, V. C., *Assembly Puro e Simples*. Amadora: Edições Graficria, 1995.

VERDE, Raul, *Computadores Digitais 2*, Dinalivro, Lisboa, sd.