Sistemas de Informação
Disciplina: Arquitetura e Organização de Computadores - 1º Período
Professor: José Maurício S. Pinheiro

AULA 6: Máquinas Multiníveis

1. Estruturas de Dados

Dado é a matéria-prima obtida na etapa de coleta (entrada) e informação é o resultado obtido pelo tratamento destes dados (saída) conforme representa a Figura 1. Esse "tratamento" feito pelo computador é o processamento, realizado através de programas que representam uma sequência de instruções.

Figura 1 - Dados e informação

Os tipos de dados podem ser classificados como:

- Escalares;
- Números (inteiros e ponto-flutuante);
- Caracteres (ASCII e EBCEDIC);
- Lógicos;
- Estruturas Estáticas (Vetores, matrizes e registros);
- Estruturas Dinâmicas (todas baseadas em ponteiros).

Para entender os tipos de dados em nível de arquitetura, deve-se ter em mente a distribuição em níveis mostrada na Figura 2:

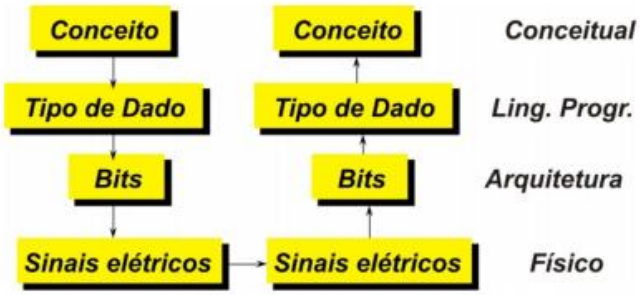

Figura 2 - Tipos de dados em diferentes níveis

Os dados inteiros podem ser representados de três diferentes formas:

- **Sinal-magnitude** – onde o bit de mais alta ordem é zero, se o número for positivo e um, se for negativo;

- **Complemento de 1** – no número negativo todos os bits são invertidos;
- **Complemento de 2** – é o complemento a 1 com soma de mais um ao final do processo de inversão.

Atualmente os computadores usam a representação negativa na forma de complemento de 2.
Os dados lógicos podem ser representados usando uma palavra inteira da arquitetura ou apenas um bit de uma dada palavra. Os números de ponto flutuante são representados usando notação científica, onde se tem alguns bits para o sinal, outros para o expoente e os demais para a mantissa ou fração. A limitação desses valores foi definida pelo IEEE (*Institute of Electrical and Electronics Engineers*), entidade internacional de padronização. A escala de representação do IEEE segue a estrutura da Figura 3:

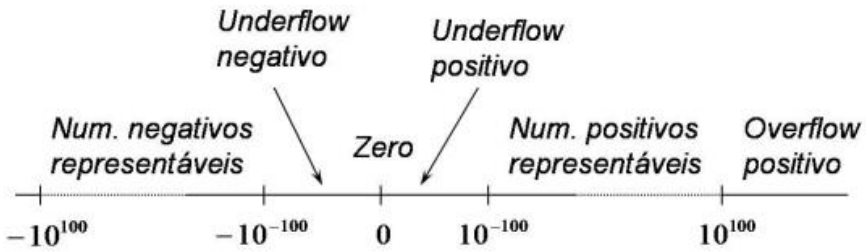

Figura 3 - Escala de representação IEEE

O computador possui uma linguagem própria (linguagem binária ou linguagem de máquina) para receber ordens. No entanto, programar em linguagem binária não é tarefa fácil para seres humanos. Para tornar a programação possível, foram desenvolvidas linguagens de alto nível, ou seja, mais próximas do entendimento humano, chamadas genericamente de linguagens de programação.
Com o mesmo objetivo de aproximar os seres humanos da linguagem dos computadores é usada uma divisão em camadas das arquiteturas de computadores. Assim, quanto mais camadas tiver uma arquitetura, mais próxima da linguagem humana será a linguagem de alto nível deste computador.

2. Níveis das Arquiteturas de Computadores

Inicialmente são determinados os componentes eletrônicos dos circuitos que vão compor as portas lógicas e demais circuitos digitais, sendo este conhecido por nível 0 (zero). Esses circuitos são organizados em forma de “pacotes” para compor o hardware dos computadores, são os chamados circuitos digitais. Classificados em nível 1 (um), os circuitos digitais são usados para compor as implementações práticas de todas as funções e mapeamentos usados na teoria dos circuitos digitais.
No nível 2 (dois), temos o projeto do hardware e do software, pois aqui se define o conjunto de instruções que a CPU será capaz de reconhecer, que tipo de processamento o computador será capaz de realizar, dentre outros.

No nível 3 (três) se encontra o Sistema Operacional, programa capaz de controlar todo o funcionamento do computador, tanto em nível de software, quanto em nível de hardware.
No nível 4 (quatro) está a linguagem de montagem ou assembly. Essa linguagem permite ao programador ter acesso a funcionalidades do computador que não seriam permitidas pelas chamadas linguagens de programação de alto nível. São programações necessárias de se executar diretamente no hardware ou mais intimamente com o sistema operacional. Não confundir com assembler, que é o programa utilizado para executar os códigos fontes criados em assembly. O assembler também é conhecido por montador, exatamente por fazer a execução da linguagem de montagem.
Finalmente no nível 5 (cinco), tem-se a linguagem de alto nível, patamar onde se encontram linguagens como Pascal, Delphi, Java e outras.
Os diferentes níveis na arquitetura dos computadores são apresentados na Figura 4:

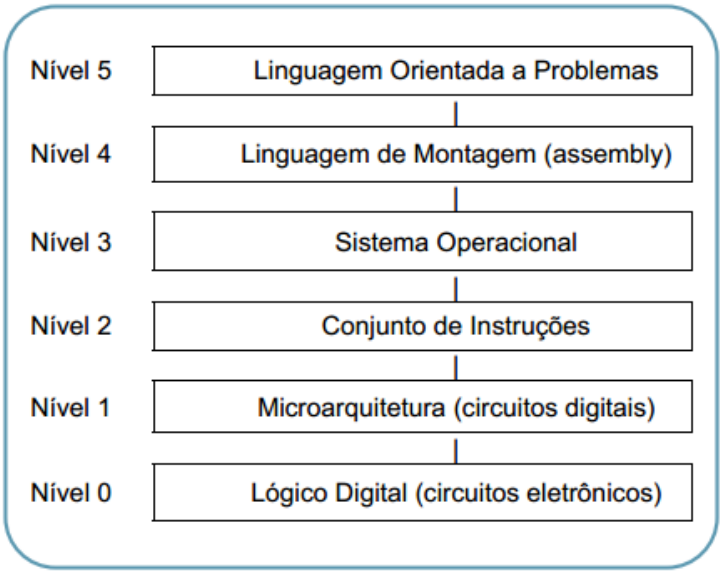

Figura 4 - Níveis das arquiteturas de computadores

3. Linguagens de Programação

Uma linguagem de programação pode ser definida como sendo um conjunto limitado de instruções (vocabulário), associado a um conjunto de regras (sintaxe) que define como as instruções podem ser associadas, ou seja, como se podem compor os programas para a resolução de um determinado problema. As linguagens de programação podem ser classificadas em níveis de linguagens, sendo que as de nível mais baixo estão mais próximas da linguagem interpretada pelo processador e mais distantes das linguagens naturais.

3.1. Linguagem de Máquina

Na linguagem de máquina, a representação dos dados e das operações (instruções) que constituem um programa é baseada no sistema binário, que é

a forma compreendida e executada pelo hardware do sistema. Torna-se inviável escrever ou ler um programa codificado na forma de uma *string* de bits.

3.2. Linguagem de Alto Nível

As linguagens de alto nível são assim denominadas por apresentarem uma sintaxe mais próxima da linguagem natural, utilizada no dia-a-dia, fazendo uso de palavras reservadas, extraídas do vocabulário corrente (com READ, WRITE, TYPE, etc.) e permitem a manipulação dos dados nas mais diversas formas (números inteiros, reais, vetores, etc.), enquanto a linguagem Assembly trabalha com bits, bytes, palavras armazenadas em memória.
A passagem de um programa escrito em linguagem de alto nível para o programa em linguagem de máquina é bem mais complexa, comparada com a linguagem Assembly. Essa passagem é feita utilizando compiladores e ligadores. Um programa escrito em linguagem de alto nível pode, teoricamente, ser usado em qualquer máquina, bastando escolher o compilador correspondente, o que não acontece com um programa escrito em Assembly.
São linguagens de alto nível: Pascal, C, C++, PHP, SQL, Python, C#, Java, etc.

3.3. Compiladores e Ligadores

O código escrito (editado) em uma linguagem de alto nível deve ser convertido em linguagem binária para que possa ser executado pelo processador. Essa conversão se dá através de compiladores e ligadores (link-editor ou linker), conforme mostra a Figura 5:

- **Compiladores:** têm como função traduzir um programa escrito em uma linguagem de alto nível em código binário. O arquivo resultante é chamado código objeto;
- **Ligadores:** ou link-editores, têm como função agregar módulos em um único programa, inserindo informações de relocação de endereços e referência entre os módulos. O arquivo resultante é chamado código executável.

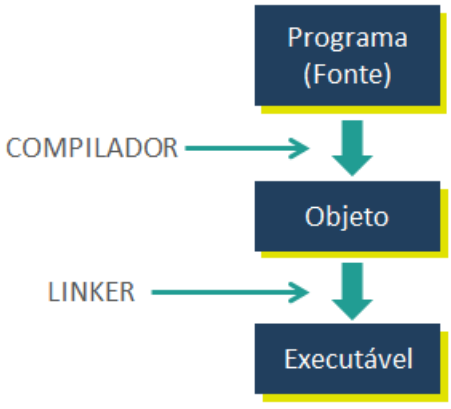

Figura 5 - Compiladores e ligadores

Quando um programa é escrito diz-se que tem o código-fonte, que deverá ser submetido a um “processo de tradução” para linguagem de máquina. A esse processo de tradução dá-se o nome de compilação, interpretação ou montagem, dependendo da linguagem original onde o código foi escrito. O processo de compilação transforma o código-fonte em executável, ou binário.
O processo de interpretação é diferente e mais comum em linguagens mais antigas e em sistemas de formatação de textos como a web da Internet. Esse método faz todos os passos do anterior, porém não gera um binário: a cada linha interpretada a instrução é enviada de imediato para a CPU executar. Isso torna o processo mais lento e obriga o usuário a ter o código fonte disponível para usar o programa.
A linguagem JAVA, por exemplo, tem seu momento de compilação, porém o produto dessa compilação não é um binário finalizado, e sim, um binário genérico chamado *bytecode*, que numa tentativa de ser universal para qualquer arquitetura, deve ser interpretado pela máquina virtual Java (JVM).

3.4. Linguagem Assembly

A linguagem de máquina de cada processador é acompanhada de uma versão “legível” da linguagem de máquina que é a chamada linguagem simbólica Assembly. Simbólica, pois esta linguagem não é composta de números binários como na linguagem de máquina. A linguagem Assembly é, na realidade, uma versão legível da linguagem de máquina. Ela utiliza palavras abreviadas, chamadas de mnemônicos, indicando a operação a ser realizada pelo processador. A linguagem Assembly é orientada para a máquina (ou melhor, para processador), é necessário conhecer a estrutura do processador para poder programar em Assembly. Essa linguagem utiliza instruções de baixo nível que operam diretamente com registros e memórias, ou seja, as instruções são diretamente executadas pelo processador.
A passagem de um programa escrito em Assembly para a linguagem de máquina é quase sempre direta, não envolvendo muito processamento. Essa passagem é chamada de Montagem, e o programa que realiza esta operação é chamado de montador (Assembler). Muitas vezes, ouvimos "linguagem Assembler". É um erro muito difundido. Na realidade, Assembly é o nome da linguagem e Assembler é o montador.
Os códigos montados são aqueles que foram escritos originalmente em Assembly, numa tentativa de fazer um programa que seja mais íntimo da arquitetura. Geralmente os drivers de dispositivos periféricos são montados. Algumas linguagens compiladas geram o chamado código intermediário, que na verdade é um arquivo escrito em assembly que o chamado link-editor do compilador trata de executar também um montador para transformar este assembly em binário executável.
Porém, um código ao ser transformado leva também todas as suas estruturas de dados para o modo binário. Aí nasce um problema: essas estruturas na máquina original vão ocupar alguns endereços de memória determinados pelos espaços livres para alocação de novos programas nessa máquina onde o código foi gerado e cada programa executa em uma partição de memória específica. Para não haver uma invasão, deve-se ocupar

dois registradores no gerenciamento dessas áreas, um contém a base onde começa a partição e o outro contém o limite do tamanho da partição. Sendo assim, teoricamente não existe possibilidade de haver invasões. Cada endereço gerado para uso desse programa é baseado no endereço da base e respeita o endereço limite. Quando o limite é atingido, acontece o erro de falta de memória. Com relação às linguagens de alto nível:

Vantagens:

- Permite acesso direto ao programa de máquina. Um programa escrito em linguagem Assembly pode ser até 300% menor e mais rápido que um programa compilado;
- A linguagem permite o controle total do hardware.

Desvantagens:

- A linguagem apresenta um número muito reduzido de instruções;
- O programador deve conhecer muito bem a máquina;
- O programa Assembly não é muito legível, precisa ser bem documentado;
- O programa Assembly não é portável (pode ser usado apenas em um tipo de computador).
- É portável apenas dentro de uma família de processadores;
- A programação Assembly tem um custo de desenvolvimento maior.

3.5. Aplicações da Linguagem Assembly

Dentre as aplicações da linguagem assembly, destacam-se:

- Controle de processos com resposta em tempo real - Nesse tipo de aplicação, o processador deve executar um conjunto de instruções em um tempo limitado;
- Comunicação e transferência de dados - Nesse tipo de aplicação é utilizada a linguagem Assembly, devido à possibilidade de acessar diretamente o hardware;
- Otimização de subtarefas da programação de alto nível - Um programa não precisa somente ser escrito em linguagem Assembly ou linguagem de alto nível. Podemos ter programas de alto nível com subtarefas escritas em Assembly, para o caso de tarefas tempo real ou para a programação do hardware do computador.

3.6. Otimização dos Compiladores

Um programa escrito em linguagem de alto nível não explicita os registradores que serão utilizados, ao invés disso, faz referências simbólicas aos valores através das variáveis criadas. Também não explicita se o valor associado à variável será guardado num registrador ou na memória. Como os registradores são mais rápidos que a memória, é desejável que sejam mais utilizados.

Cada variável passível de ser guardada em um registrador, recebe um tratamento especial. O compilador pode criar um número ilimitado de "registradores virtuais" para armazená-las e, a partir daí, compartilhar o registrador real de acordo com alguma técnica específica. Claro, isso é feito dentro da limitação da CPU e muitas variáveis vão obrigatoriamente para a memória.
Os compiladores mantêm essa filosofia de forma transparente para o programador, mas alguns, como é o caso da linguagem C, oferecem a possibilidade de o programador decidir quais variáveis devem, sempre que possível, serem armazenadas em registradores, para tanto, ao declarar uma variável, ao invés de escrever "**int soma**", escreve-se "**register int soma**", obrigando então, o compilador a priorizar os registradores para aquela variável.

4. Repertório de Instruções – Programação Assembly

Esse é o nome dado ao conjunto de instruções que a máquina reconhece e executa. As instruções podem ser classificadas em uma das categorias a seguir:

- Leitura/Escrita em memória;
- Operações lógicas e aritméticas sobre dados;
- Controle da sequência de execução;
- Entrada/Saída

As instruções seguem um formato rígido para serem reconhecidas pela CPU, com *opcode* e operandos necessários, a ilustração é feita na Figura 6.

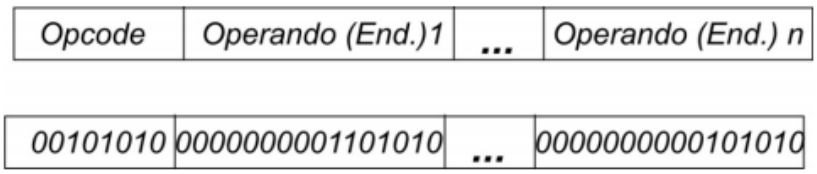

Figura 6 - Formato das instruções

A representação interna das instruções segue o padrão de bits mostrado na figura anterior. Por outro lado, a representação em nível de programação é feita com mnemônicos, ou seja, palavras que representam as instruções. Como, por exemplo, **ADD R1,A,** onde soma o conteúdo do registrador **R1** com o conteúdo de uma variável chamada **A**.
Programar em assembly em uma máquina real significa ter acesso a algumas funcionalidades até em níveis mais baixos que o sistema operacional. As instruções de assembly são criadas de acordo com a arquitetura da máquina e podem ser baseadas em um registrador de uso geral, nesse caso conhecidas como assembly de acumulador, ou baseadas em dois registradores ou ainda baseadas em mais registradores tendo assim três parâmetros.
A tabela 1 mostra a classificação dos tipos de assembly:

Tabela 1 - Tipos de assembly

Num de Endereços	Mnemônicos	Interpretação
3	OP A,B,C	A ← B OP C
2	OP A,B	A ← A OP B
1	OP A	AC ← AC OP A

Ao executar o assembly, a CPU segue um padrão de desempacotamento e reempacotamento dos dados conforme pode ser visto na Figura 7:

Figura 7 - Níveis de execução assembly

- **Movimentação de dado** - cópia ou remoção de dados entre a CPU e a memória;
- **Transformação de formato** - movimentação de bits através de deslocamento e rotação;
- **Transformação de código** - quando se faz necessário converter um código em outro por motivos de críticas de dados ou mesmo para compatibilizar dispositivos que a princípio usam códigos incompatíveis entre si.

Um exemplo prático de mudança de código envolve os programas onde se deseja implementar uma crítica de dados para saber se o usuário digitou um valor numérico em um campo que deve trabalhar com números. Para isso pode ser usada uma *string* (expressão contendo qualquer caractere alfanumérico, formando ou não palavras).
Usa-se a entrada em *string* para permitir a digitação de qualquer formato de dados. Posteriormente, checam-se os valores digitados se correspondem aos dígitos numéricos e finalmente se faz a transformação da *string* digitada em valor numérico para ser usado pelo programa. Essa transformação pertence ao grupo de instruções de transformação de código.

4.1. Instruções de Transferência de Dados

As instruções de transferência de dados movem dados do computador de um

local para outro, sem a necessidade de se modificar o conteúdo dos dados. As transferências mais comuns ocorrem entre a memória e os registradores do processador, entre os registradores do processador e dispositivos de entrada e saída, e entre os próprios registradores do processador.
A Tabela 2 fornece uma lista com oito instruções de transferência de dados, usadas em muitos sistemas computacionais.

- Instrução ***load*** - usada na maioria das vezes para designar uma transferência entre a memória e um registrador do processador, geralmente um acumulador;
- Instrução ***store*** - designa uma transferência de um registrador do processador para a memória;
- Instrução ***move*** - usada em computadores com múltiplos registradores de CPU para designar uma transferência de um registrador para outro. Também é usada para transferência de dados entre os registradores da CPU e a memória, ou entre duas palavras de memória;
- Instrução ***exchange*** - realiza a troca de informação entre dois registradores ou um registrador e uma palavra de memória;
- Instruções ***input*** e ***output*** - transferem dados entre os registradores do processador e terminais de entrada ou saída;
- Instruções ***push*** e ***pop*** - transferem dados entre registradores do processador e uma pilha da memória.

Tabela 2 - Instruções de transferência de dados

Nome	Mnemônico
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Para que seja possível escrever programas em linguagem assembly para um computador é necessário saber os tipos de instruções disponíveis e também ter familiaridade com os modos de endereçamento usados no computador em particular.
A Tabela 3 mostra a convenção recomendada para linguagem assembly e a transferência realizada de fato em cada caso.

Tabela 3 - Operações de transferência

Modo	Convenção Assembly	Transferência de Registradores
Endereçamento Direto	LD ADR	AC ← M[ADR]
Endereçamento Indireto	LD @ADR	AC ← M[M[ADR]]
Endereçamento Relativo	LD $ADR	AC ← M[PC + ADR]
Operando Imediato	LD #NBR	AC ← NBR
Endereçamento Indexado	LD ADR(X)	AC ← M[ADR + XR]
Registrador	LD R1	AC ← R1
Registrador Indireto	LD (R1)	AC ← M[R1]
Auto Incremento	LD (R1)+	AC ← M[R1]; R1 ← R1 + 1

- ADR significa endereço e NBR significa um número ou operando;
- X é um índice, R1 é um registrador do processador, e AC é o registrador acumulador;
- O caractere @ simboliza um endereço indireto;
- O caractere $ antes de um endereço faz com que o mesmo seja relativo ao contador de programa PC;
- O caractere # precede o operando em uma instrução de modo imediato;
- Um registrador, quando colocado entre parênteses após o endereço simbólico, reconhece uma instrução de modo indexado;
- O modo registrador é simbolizado por fornecer o nome de um registrador do processador. No modo registrador indireto, o nome do registrador que contém o endereço de memória deve estar entre parênteses;
- O modo de auto incremento se distingue do modo registrador indireto pela adição de um “+” após o registrador entre parênteses;
- O modo de auto decremento é similar ao modo incremental, mas utiliza um “-” no lugar do “+”;

4.2. Controle de Fluxo

Em linguagens de programação é comum e necessário fazer uso de controles de fluxo para resolver alguns problemas de lógica que envolvem as diversas programações no computador. São comuns desvios baseados em estruturas do tipo **se...então...senão**, estes conhecidos como desvios condicionais. Menos comuns hoje em dia, mas não sem importância, são os desvios incondicionais, baseados em estruturas do tipo **go to** e **labels**.
No assembly os desvios incondicionais são usados para sair de laços e de outras partes do programa de acordo com a lógica em implementação. Os desvios condicionais em assembly testam o valor de um registrador especial chamado flags, ele sempre armazena um código após uma execução de comparação ou de operação aritmética. A Tabela 4 apresenta os valores armazenados nos flags e seus significados.

Tabela 4 - Valores dos flags

Valor (2 bits)	Significado
00	Valores iguais
01	Valor1 MAIOR QUE valor2
10	Valor1 MENOR QUE valor2
11	Overflow

É preciso lembrar que overflow nos flags ocorre sempre que o valor resultante for grande ou pequeno demais para ser armazenado na estrutura escolhida ou ainda quando for feita uma comparação de dois valores impossíveis de serem comparados, tais como uma string e um valor em ponto flutuante.
Resumidamente, pode-se trabalhar com o assembly apresentado na Tabela 5, feito para uma máquina hipotética de quatro registradores de uso geral.

Tabela 5 - Instruções do assembly

Intrução	Opcode	Descrição	Tam. instrução
NOP	0000	No operation	1
LDA reg,end	0001	Carrega var de mem em reg	2
STA reg,end	0010	Armazena reg em var de mem	2
ADD reg,end	0011	Reg ← Reg + mem	2
SUB reg,end	0100	Reg ← Reg – mem	2
AND reg,end	0101	Reg ← Reg and mem	2
NOT reg	0110	Reg ← not Reg	2
CMP reg1,reg2	0111	Flags ← reg1 comp reg2	2
JMP endr	1000	Desvia para o label endr	2
JPC cond end	1001	Desvio condicional*	2

4.3. Modos de Endereçamento

Um operando pode estar em diversas localizações na arquitetura tais como: na memória, no registrador, em um ponteiro, entre outras. Para cada forma de se especificar o operando na instrução assembly, existe um modo de endereçamento.

- **Endereçamento Imediato** - O operando é especificado diretamente na instrução, na forma de uma constante. Exemplo: ADD R1,#A (a constante A será adicionada ao conteúdo de R1);

- **Endereçamento Direto** - O endereço do operando na memória é especificado na instrução. Exemplo: ADD end1, end2 [end1] Å [end1] + [end2];
- **Endereçamento de Registradores** - Apenas registradores são referenciados nas instruções. Exemplo: ADD R3,R5;
- **Endereçamento Indireto** - O endereço referenciado na instrução, na verdade, contém o endereço do operando real armazenado. Exemplo: ADD R1,(R3). R3 aponta para o endereço do operando real (esse modo implementa os ponteiros);
- **Endereçamento Indexado** - Esse modo de endereçamento é usado para operar vetores e matrizes. A instrução contém o endereço base do *array* (estrutura de dados que armazena uma coleção de elementos de tal forma que cada um dos elementos possa ser identificado por, pelo menos, um índice ou uma chave) e o deslocamento para mudar de célula. Exemplo: ADD R1, [R2]end. R1 deverá armazenar o somatório dos valores armazenados na matriz.