# Introdução aos Sistemas Computacionais

Carlos Antônio Ruggiero

1992 - Última Modificação 2020

# Sumário

# Capítulo 1

# Introdução Histórica. Conceitos Fundamentais

## 1.1 Introdução

A história da computação pode ser contada a partir do trabalho pioneiro do matemático inglês Charles Babbage (1791-1871) que projetou um dispositivo mecânico para cálculos, o qual seria controlado por três cartões perfurados. O projeto era muito ousado para sua época e não pode ser terminado. Porém, muitas das idéias de Babbage foram aproveitadas um século mais tarde com o advento da computação eletrônica.

As primeiras máquinas exibindo as características fundamentais que se encontram em modernos computadores apareceram na década de 40. Os principais centros de pesquisa na área localizavam-se, então, na Inglaterra e Estados Unidos. Para se entender como apareceu o *computador de programa armazenado,* como é chamado o computador moderno, é necessário ter-se uma visão geral da situação mundial anterior aos anos 40.

Desde o começo do século XX, muito trabalho foi efetuado em indústrias e universidades com a finalidade de tornar mais simples a execução de operações que envolviam grande quantidade de cálculos numéricos. Devem ser ressaltados os trabalhos de Herman Hollerith (considerado o fundador da "International Business Machines - IBM") e de James Powers na América. Hollerith desenvolveu máquinas ordenadoras de cartões perfurados que muito ajudaram em censos norte americanos. Máquinas com cartões perfurados eram usadas principalmente em aplicações comerciais e estatísticas mas com o tempo, foram gradativamente sendo utilizadas também para cálculos científicos. Além de máquinas com cartão perfurado, calculadoras mecânicas de diversas formas e procedências achavam-se no mercado.

Um terceiro tipo de máquinas para cálculos era o *Analisador Diferencial* que resolvia equações diferenciais muito mais rapidamente que outras máquinas ou métodos. Na década de 30, vários analisadores diferenciais haviam sido construídos nos Estados Unidos, Inglaterra e Alemanha.

A deflagração da segunda guerra mundial, trouxe uma enorme motivação para a construção de máquinas mais rápidas que pudessem quebrar os códigos de guerra

inimigos. Na Inglaterra, uma enorme máquina com esse propósito foi construída no começo dos anos 40. Ela se chamava *COLOSSUS* e continha 1500 válvulas. O objetivo era quebrar os dois códigos usados pelos alemães, o *Enigma* e o *Geheimschreiber*. Um matemático inglês que se tornaria muito famoso no mundo da computação trabalhou ativamente nesse projeto; seu nome era Alan M. Turing. O sucesso na quebra dos códigos motivou o intenso estudo de computação na Inglaterra no período pós 45.

Também no início dos anos 40, estava-se desenvolvendo uma grande máquina para cálculos numéricos nos EUA, o *ENIAC* que continha 18.000 válvulas. O ENIAC ("Electronic Numerical Integrator and Computer") foi projetado e construído por J. W. Mauchly e J. P. Eckert na escola Moore da Universidade da Pensilvânia. Ele não era propriamente um computador de programa armazenado mas foi o precursor de todos os projetos de computador nos Estados Unidos.

## 1.2 O Computador de Programa Armazenado

Um artigo de Burks, Goldstine e von Neumann escrito em 1946 é considerado como a primeira proposta de um computador de programa armazenado. A arquitetura sugerida ficou conhecida como arquitetura de von Neumann e é aquela utilizada (com algumas modificações) pela grande maioria dos computadores *convencionais* modernos. Von Neumann e sua equipe porém, demoraram para construir esse computador e outros grupos de pesquisa completaram projetos semelhantes em outros lugares.

O primeiro computador de programa armazenado a rodar foi construído na Universidade de Manchester, Inglaterra. O grupo era chefiado por F. C. Williams e Tom Kilburn e iniciou suas atividades em dezembro de 1946. O MARK-1, como era chamado, foi concluído em 1948 e rodou um programa pela primeira vez no dia 21 de junho de 1948 (um programa de fatoração). O MARK-1 já continha algumas características de computadores modernos como a existência de um registrador de índice e a presença simultânea de memória primária (que era uma modificação de um CRT comum, que ficou conhecido como *Willams's tube*) juntamente com memória secundária (implementada como um tambor magnético).

Simultaneamente, Maurice Wilkes chefiava um grupo em Cambridge para construir um outro computador de programa armazenado, o EDVAC (Electronic Discrete Variable Automatic Computer) que executou seu primeiro programa em 6 de maio de 1949. Wilkes ficou conhecido como o primeiro pesquisador a propor um computador microprogramado em 1951.

Duas características básicas distinguem os computadores de programa armazenado de todas as máquinas calculadoras anteriores:

- O armazenamento tanto do programa quanto dos dados é efetuado na mesma memória física.
- O resultado da computação passa a ser totalmente determinístico e o comportamento da máquina é completamente previsível. Isto se deve ao uso de quantidades digitais.

Von Neumann finalmente concluiu seu protótipo em 1950. Um ano mais tarde, Mauchly e Eckert concluíram o UNIVAC-1 que seria o primeiro computador comercial no mercado americano. Anteriormente, a indústria Ferranti na Inglaterra já havia produzido computadores baseados no MARK-1.

A década de 50 veria grandes transformações na tecnologia utilizada para a construção de computadores. A implementação de memória principal deixou de ser feita com tubos eletrostáticos para ser feita com *núcleo de ferrite* (criados por Forrester em 1951 no MIT). Os transistores, criados em 1948 na Bell Laboratories, começaram a ser utilizados a partir dos primeiros anos da década de 50 permitindo o projeto e implementação de computadores muito mais rápidos e confiáveis que os com válvula.

Dependendo da tecnologia utilizada para sua implementação, os computadores são divididos em várias *gerações*. A primeira geração de computadores foi construída com válvulas enquanto computadores da segunda geração utilizaram-se de transistores. No início da década de 60, surge a terceira geração baseada em circuitos integrados (vários componentes compactados numa única pastilha). Posteriormente, com a integração em larga escala (LSI), surge a quarta geração (início da década de 70). Fala-se na quinta geração que, ao contrário das predecessoras, não introduziria grandes modificações na tecnologia mas inovaria a arquitetura e o software radicalmente.

Vários exemplos de computadores de terceira e quarta geração serão estudados no decorrer do curso.

## 1.3 Conceitos Fundamentais

### 1.3.1 Arquitetura, Organização e Implementação

Não é fácil achar-se uma definição para o termo *arquitetura*. Amdahl, Braauw e Brooks, projetistas do sistem IBM/360 definiam o termo como sendo as características de um computador vistas por um programador em "assembler", ou seja, a estrutura conceitual e funcional da máquina e não os detalhes de organização e implementação. Stone, por outro lado, diz que arquitetura é o estudo da organização e interconexão de componentes de sistemas computacionais. Uma definição mais informal é dada por Patterson: "it's the business of designing computer". A primeira dessas definições será adotada mas deve-se ter sempre em mente os outros pontos de vista.

A *organização* de um computador pode ser definida como a "natureza das unidades operacionais que realizam as especificações da arquitetura". Assim, a organização estaria logicamente um nível abaixo da arquitetura preocupando-se mais com detalhes de projeto. A *implementação* então, cuidaria das características físicas e tecnológicas das unidades operacionais da organização, preocupando-se com detalhes como velocidade de comutação de componentes, dimensionamento de hardware, etc. A implementação refere-se ao nível mais baixo do projeto computacional.

### 1.3.2 Níveis de hardware e software em computação

Um computador moderno é composto por um conjunto de circuitos eletrônicos lógicos que aceitam instruções de programação. Essas instruções são pulsos elétricos interpretados como níveis digitais. Deve ser óbvio para o aluno que programar um algoritmo complexo pelo uso de pulsos elétricos é tarefa muito difícil, ou quase impossível.

O ideal seria que o computador aceitasse comandos que seres humanos usam, tais como "some esses dois números" ou "se o nome for xxx, faça isso, senão, faça aquilo". Tal computador, porém, apresentaria uma relação custo benefício muito ruim. O que se faz, então, é "traduzir" uma linguagem parecida com a usada pelos seres humanos, para os pulsos elétricos aceitos pelo computador. A linguagem próxima àquela dos seres humanos é chamada de alto nível, enquanto os pulsos elétricos são conhecidos como linguagem de baixo nível.

A tradução do alto nível para o baixo nível não é feita de uma só vez; de fato, existem vários níveis intermediários e, de certa forma, cada nível intermediário representa uma máquina virtual para o nível imediatamente acima dele.

Assim, a solução de um problema por computador passa por várias etapas ou *níveis*. Da especificação do problema (nível mais alto), até a sequência de chaveamento dos transistores (nível mais baixo):

- O problema deve ser codificado numa linguagem de alto nível. Esse pode ser chamado o nível da aplicação.

- Uma vez codificado, o programa deve ser *compilado*, ou seja, os comandos da linguagem devem ser transformados em instruções executáveis pelo processador. Esse é o nível de linguagem ou compilação.

- O programa é então fornecido ao sistema operacional que se encarregará de prover todos os recursos do sistema necessários para a execução tais como parte do tempo disponível do processador, canais de entrada saída, bibliotecas padrão, etc..

- As instruções de máquina devem ser executadas pelo processador. Aqui normalmente considera-se que os níveis de software acabam e os de hardware começam. As instruções de máquina são *interpretadas*, transformando-se em microinstruções (em computadores microprogramados).

- As microinstruções agem diretamente sobre os circuitos eletrônicos básicos do computador (contadores, relógios, circuitos lógicos, etc).

Nesse curso, o maior interesse será no penúltimo nível o qual é o objeto básico do estudo de arquitetura. Note que o último nível é objeto do estudo de organização e de implementação. Os níveis mais altos são objeto de várias disciplinas de software tais como Sistemas Operacionais, Linguagens de Programação e Estudo de Algorítmos.

### 1.3.3 Arquitetura von Neumann

Num artigo publicado em 1946, Burks, Goldstine e von Neumann sugeriram um computador cuja arquitetura era bastante semelhante às arquiteturas de computadores modernos. A essa arquitetura, deu-se o nome de "Arquitetura de von Neumann" (von Neumann era, dentre os três, o pesquisador mais influente). A figura 1.1 mostra a arquitetura básica proposta por Burks et al.

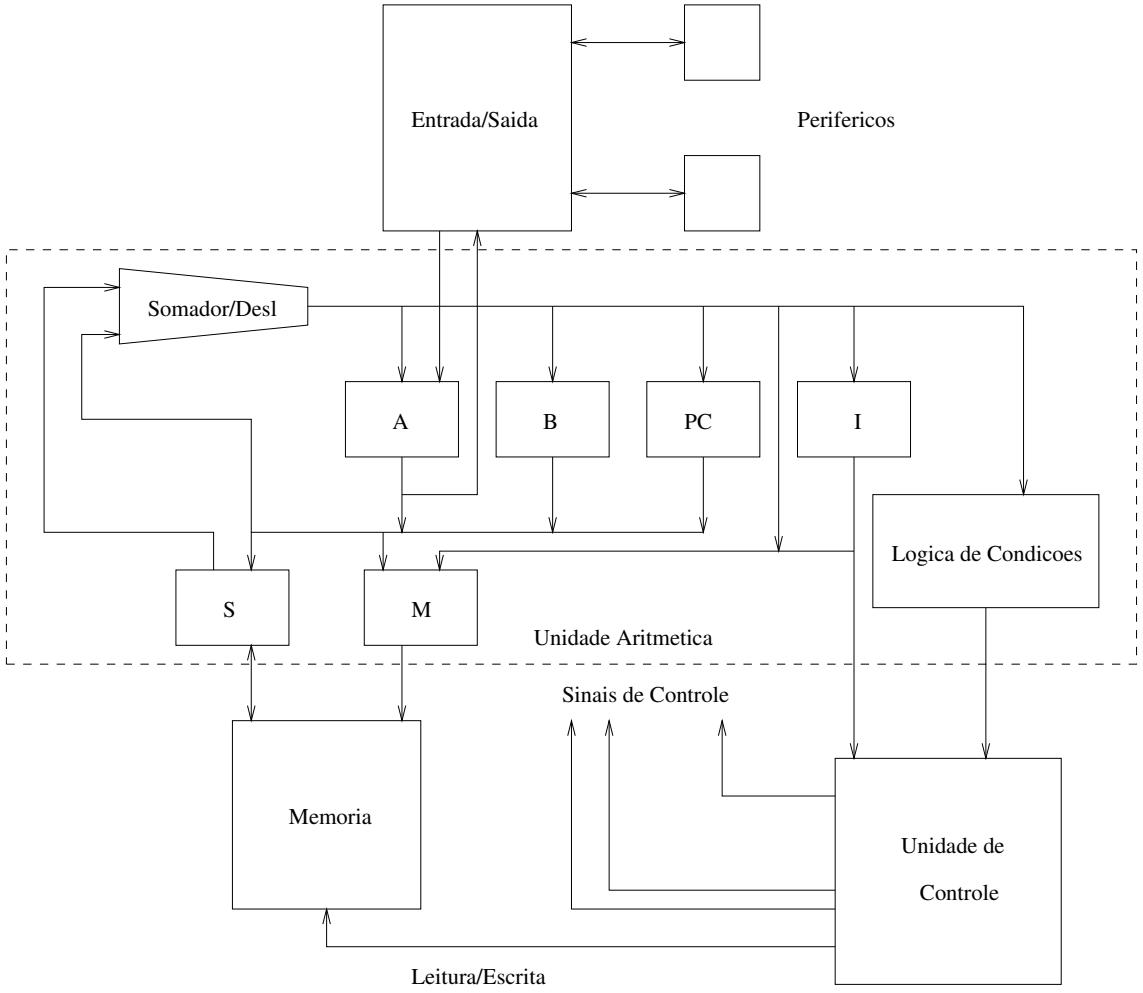



Figura 1.1: Arquitetura Proposta pro Burks et al.

Os *registradores* A e B são utilizados para operações aritméticas. Dados lidos ou a serem escritos na memória são armazenados no registrador S. O registrador M (de memória) contém o *endereço* da posição de memória endereçada pelo processador. O registrador P *aponta* sempre a próxima instrução a ser executada e ficou conhecido como *contador de programa*. O código da instrução sendo executada é armazenado no registrador I, conhecido como registrador de instruções. Note que, segundo a nossa definição de arquitetura, o registrador I (e também S e M) não deveria aparecer na figura 1 pois seu conteúdo não pode ser modificado pela linguagem montadora; ele pertence à *organização* da proposta de von Neumann. Ele

foi incluído, porém, para propiciar um melhor entendimento sobre o funcionamento dessa arquitetura.

A memória proposta continha 4096 posições, cada uma com 40 bits de informação (*palavra*). Desta forma, 12 bits de endereçamento eram necessários (2 elevado a 12 é 4096).

Ao conjunto de registradores, unidade aritmética e unidade de controle, dá-se o nome de *Unidade Central de Processamento-UCP*. Além da UCP e da memória, note a presença de unidades de entrada e saída (*periféricos*).

Quando uma instrução é executada, a seguinte sequência de operações deve ser efetuada:

1. A instrução a ser efetuada é *buscada (fetch)* da memória. A posição de memória é apontada pelo contador de programa. O conteúdo de P é colocado em M e a unidade de controle gera um sinal de *leitura* para a memória que então passa o conteúdo da palavra endereçada para S. A unidade de controle faz com que S transfira o valor lido para o registrador I. O valor de P é então incrementado para apontar a próxima instrução. Essa etapa é chamada de *busca ou "fetch"*.

2. O valor armazenado em I age sobre a unidade de controle que então *executará* a instrução. Como exemplo, suponha que a instrução a ser executada é uma soma, aqui representada por ADD X, onde X representa uma posição de memória. O valor X (que na máquina de Burks et al. vem especificado na própria instrução) é colocado em M e a unidade de controle gera um sinal de leitura da memória. O valor lido é somado com o valor do acumulador pelo somador/deslocador e o resultado é colocado de volta em A. À essa segunda etapa dá-se o nome de fase de *execução*.

Na máquina de Burks et al., os dados eram de 40 bits. Duas instruções eram colocadas numa palavra, cada uma com 20 bits dos quais 8 representavam o código da instrução (*op-code*) enquanto os outros 12 eram o endereço do dado. Veja figura 1.2

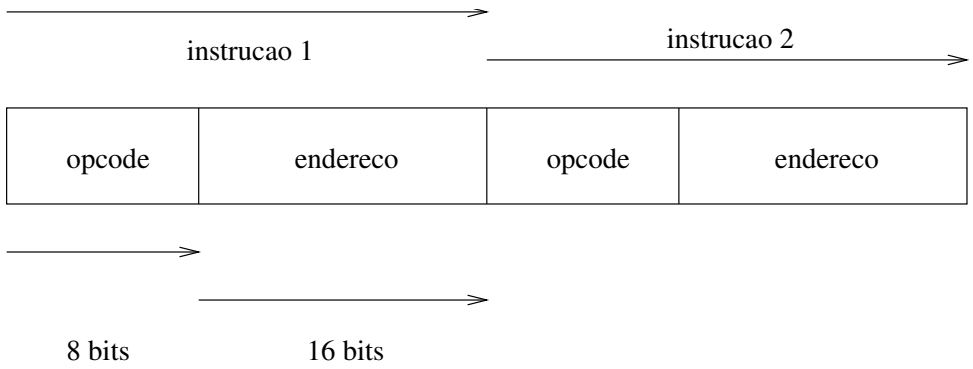



Figura 1.2: Codificação da instrução de von Neumann

As características fundamentais dessa arquitetura são:

- O programa (as instruções) é armazenado na mesma memória que os dados.
- A comunicação com a memória é sobrecarregada (*gargalo de von Neumann*). Note que tanto na fase de busca quanto na de execução, a unidade central de processamento deve ir à memória.

O computador de von Neumann é hoje representado como o mostrado na figura 1.3:

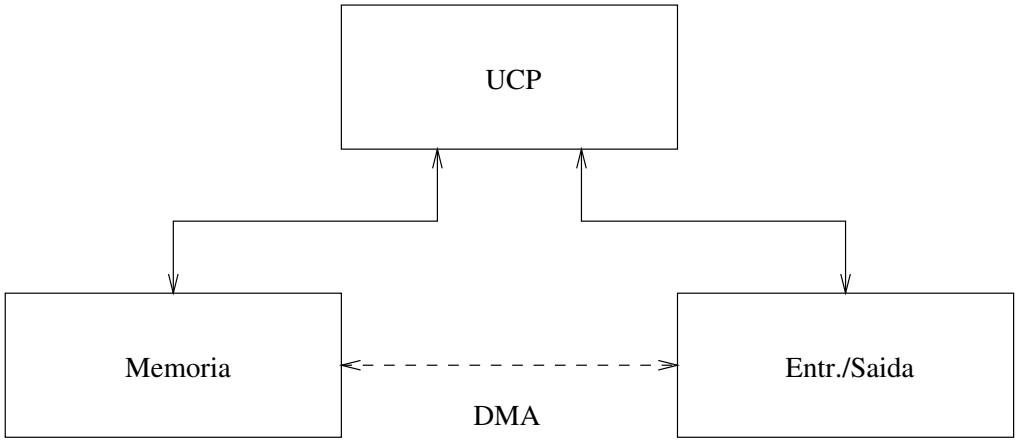



Figura 1.3: Diagrama mostrando a arquitetura de von Neumann

Como exemplo final, seja o segmento de programa mostrado abaixo:

A=B+C
D=E+F

Após passar pelo compilador, o código de máquina será (para o 8085, um microprocessador da INTEL):

```
1000 LDA    2001                100B LDA    2004
1003 MOV    B,A                 100E MOV    B,A
1000 LDA    2002                100F LDA    2005
1000 ADD    B                   1012 ADD    B
1000 STA    2000                1015 STA    2003
```

onde assume-se que A, B, C, D, E e F acham-se nas posições 2000, 2001, ... até 2005..

Uma característica da máquina de Burks et al. que hoje não está mais presente, é que a execução de interações naquela máquina envolvia modificação de instruções durante a execução do programa. De fato, a máquina tinha instruções especializadas em modificar o campo de endereço de outras instruções previamente executadas. Dessa forma, diferentes elementos de uma estrutura de dados poderiam ser utilizados pela mesma instrução. Von Neumann acreditava ser essa uma das grandes vantagens da arquitetura e praticamente todos os computadores da

década de 50 proviam instruções para a *automodificação* de programas. Na década de 60, porém, severas críticas foram feitas a essa característica e a automodificação caiu em desuso. De fato, um programa automodificante é mais difícil de ser reexecutado e pode ser extremamente difícil de ser depurado. Atualmente, a utilização de diferentes elementos de uma estrutura pela mesma instrução é conseguida através de outros mecanismos como o da *indexação* e a automodificação, apesar de ainda possível, é fortemente desencorajada.

### 1.3.4 Elementos de uma Arquitetura

Várias características de uma máquina são importantes no estudo e avaliação de sua arquitetura. Dentre elas, as mais importantes serão analisadas brevemente a seguir. Durante todo o curso, esses elementos serão estudados com mais detalhes.

#### Representação de Dados

Essa característica refere-se ao modo como os valores são representados e tratados. Por exemplo, a maioria dos computadores define o tipo de dados *inteiro*, mas a representação pode variar de máquina para máquina: binária, binária codificada como decimal (BCD), complemento de um, complemento de dois, etc. Também existem várias possibilidades para representação de números em ponto flutuante (números *reais*).

A representação de caracteres também pode ser feita de várias maneiras. Normalmente algum tipo de código que associa um inteiro a um caracter é utilizado. Códigos famosos são o ASC (*American Standard Code*) e o EBCDIC.

#### Tamanho da estrutura de dados básica

A unidade básica de informação é o *bit*. O conjunto de 8 bits é chamado de byte. Cada posição de memória armazena um certo número de bits. Têm-se computadores com memória de 8, 16, 32, 64 bits.

Computadores podem ser classificados segundo o número de bits em cada posição de memória. Uma classificação mais útil se refere ao número de bits do barramento da unidade aritmética.

#### Endereçamento

O endereço de uma posição de memória pode ser formado de várias maneiras diferentes. O endereço *absoluto* dado na instrução como no caso da arquitetura de von Neumann, é apenas uma das possibilidades. Como visto anteriormente, computadores modernos dispõem de complexas regras de formação de endereços de tal forma que a mesma instrução possa utilizar-se de vários elementos de uma estrutura de dados. Nesse caso, fatores externos a própria instrução podem contribuir no cálculo de endereços o que apresenta três grandes vantagens:

- **Facilidade de manipulação de estrutura de dados:** o mesmo código pode processar uma estrutura de dados completa.

- **Relocabilidade:** o programa pode ser carregado a partir de qualquer posição de memória.
- **Proteção:** é possível evitar-se que certas posições de memória sejam inadvertidamente modificadas. É claro que isso não era possível na arquitetura original de von Neumann.

A maneira pela qual um computador executa a *aritmética de endereços* é característica importante de sua arquitetura. Outra característica importante é o número de endereços que podem ser especificados numa única instrução (normalmente aritmética). Assim tem-se máquinas de 0, 1, 2 e 3 endereços.

### Organização de Registro

Desde os primeiros computadores (veja comentário sobre o MARK-1 na seção 1) notou-se que a unidade central de processamento era muito mais rápida que as memórias disponíveis. Assim, o conceito de *hierarquia de memória* foi introduzido. Mesmo a memória principal, porém, é lenta, quando comparada à velocidade da UCP. Esta deve então utilizar-se de "áreas de trabalho" pequenas mas extremamente rápidas. Às características dessa área de trabalho dá-se o nome de *modelo de registro*. Cada posição nessa área de trabalho é chamada de *registro* o qual pode ser especificado de maneira breve e concisa.

Dois tipos de registros podem ser definidos: os de operando e os de controle. Registros de operação são usados para a manipulação de dados do programa que está sendo executado. Registros de controle, por outro lado, são aqueles que mantém endereços e informações sobre o estado da máquina e não sobre os dados manipulados pela máquina. Registros de operando são usados para processamento numérico. Como exemplo de registros de controle tem-se o contador de programa (chamado PC) e os registros de *segmentação* que indicam os espaços de programa e dados no espaço total de memória. Outros tipos de registros serão analisados em aulas subsequentes.

### Conjunto de Instruções

Uma instrução de computador definine a operação que deve ser efetuada sobre um ou vários dados. O conjunto das instruções em sequência definem o que o programa executará.

Vários tipos de instrução são encontrados em computadores. Entre os tipos mais importantes destacam-se:

- **Transporte de Dados:** instruções usadas para mover dados entre posições de memória e registros. Exemplo típico é o MOVE. Assim, para o 8085 tem-se MOV A,B que move o conteúdo do registro B para A.
- **Carregamento:** usadas para trazer dados da memória para registros. A instrução básica nessa categoria é LOAD. Note que a instrução LOAD poderia ter sido colocada no ítem anterior, mas pela sua importância em arquitetura, é conveniente separá-la.

- **Armazenamento:** usadas para gravar dados de registros na memória. O exemplo típico é a instrução STORE que assim como LOAD poderia ser tratada como uma instrução de transporte.

- **Controle de Fluxo:** usadas para alterar a sequência normal de execução de instruções. Exemplos são BRANCH, JUMP, CALLSUB, etc.

- **Aritméticas e lógicas:** executam operações aritméticas e lógicas como soma, subtração, ou-exclusivo, etc. Exemplos são ADD, SUB, XOR.

Instruções são normalmente especificadas por um mnemônico contendo 3 ou 4 letras. Esses mnemônicos constituem o que se convencionou chamar de linguagem *montadora* ou *assembly*. O programa que transforma a linguagem montadora em código binário que o processador possa entender é chamado de *montador* ou *assembler*.

Como exemplo do funcionamento de um montador, considere as duas linhas de código abaixo:

```
X   EQU 2000H
    ORG 1000H
    LDA X
```

Depois da montagem, tem-se:

```
1000    3A
1001    00
1002    20
```

### Interrupções

É muitas vezes necessário mudar-se o fluxo de controle normal de um programa quando da ocorrência *assíncrona* de um evento externo. A essa alteração de fluxo dá-se o nome de *interrupção*.

Vários eventos podem provocar a interrupção de um programa. Alguns exemplos importantes são:

- Um programa de maior prioridade pede a utilização da UCP: esta deverá suspender temporariamente a execução do programa de menor prioridade que continuará sendo executado quando o programa de alta prioridade permitir.

- Um problema sério de hardware é detetado como por exemplo uma falha de memória, falta de energia elétrica, etc.

- O programa tenta executar uma operação ilegal tal como dividir por zero, modificar uma posição de memória protegida, executar uma instrução não definida, etc.

Vários mecanismos de atendimento e tratamento de interrupções serão vistos durante o curso.

### Estado de Controle

Para maior versatilidade e confiabilidade quando da execução de programas, certos computadores só permitem que certas instruções sejam executadas quando o processsador encontra-se no *estado de controle*. A modificação de certas posições de memória é também ilegal quando o processador não se acha nesse estado.

A existência de estados de controle facilita bastante o trabalho dos sistemas operacionais tornando-os mais confiáveis e versáteis.

### Entrada-Saída

Mecanismos de *entrada-saída* definem as maneiras pelas quais um computador se comunica com o meio exterior e principalmente com o ser humano. Unidades de entrada-saída (periféricos) são normalmente muito mais lentas que a UCP e memória e consequentemente merecem um tratamento diferenciado.

# Capítulo 2

# Software e Compiladores

## 2.1 Conceitos Gerais de Software

Dá-se o nome de software ao:

1. Conjunto de Programas aplicativos que rodam na máquina.

2. Conjunto de programas fornecidos junto com o hardware do computador com o objetivo de facilitar a sua utilização.

O software de computadores (principalmente os programas aplicativos) é, na maior parte das vezes, escrito numa *linguagem de programação* de alto nível. Dentre as várias linguagens de alto nível existentes, as mais importantes são:

- FORTRAN (FORmula TRANslation): Criada por J. Backus no final da década de 50 especialmente para processamento numérico. Vários "padrões" de FORTRAN foram criados: FORTRAN 66, FORTRAN 77. É uma linguagem muito popular principalmente entre físicos e engenheiros não diretamente ligados à ciência da computação. FORTRAN têm recebido severas críticas de cientistas da computação. A crítica principal é que o seu uso leva a programas de difícil compreensão e com erros difíceis de serem detectados.

- COBOL: Linguagem antiga especialmente voltada para cálculos financeiros. Não é usada pela comunidade científica.

- C: Linguagem criada nos Laboratórios Bell por Dennis Ritchie no início da década de 70 para a escrita do sistema operacional UNIX. É considerada uma linguagem de sistemas, ou seja, é utilizada para se escrever programas que ajudarão na utilização mais efetiva do hardware. Exemplos: sistemas operacionais, filtros, ordenadores, formatadores, editores, etc.

- PASCAL: Criado por N. Wirth na década de 70, é considerada a linguagem que introduziu o conceito de programação estruturada. O objetivo básico é produzir programas mais confiáveis e de fácil entendimento. É usado principalmente em aplicações não numéricas e no ensino de programação.

- LISP: Criada no início da década de 60 por John McCarthy, tornou-se popular na década de 80 entre pesquisadores em arquitetura e inteligência artificial. É a precursora das linguagens funcionais que supostamente levam à produção de programas mais confiáveis (programas que podem ser provados corretos).

- PROLOG: Criada na década de 70 na França, tenta utilizar as propriedades da lógica em programação. Tornou-se bastante popular entre os pesquisadores de inteligência artificial.

Muitos sistemas operacionais existem atualmente. Alguns sistemas importantes são:

- UNIX: Desenvolvido nos laboratórios Bell por Thompson e Ritchie com a qualidade de portabilidade como principal objetivo, esse sistema operacional tornou-se extremamente popular na década de 80. Deu origem ao Linux e ao FreeBSD atuais.

- VMS (Virtual Memory System): Este sistema surgiu com a arquitetura VAX 11 da Digital Equipment Corporation (DEC). Considerado um sistema bastante sólido, o VMS espalhou-se por todo o mundo (em processadores VAX). O chefe de desenvolvimento do VMS foi David Cutler.

- DOS: Esse sistema operacional tornou-se o padrão para os computadores pessoais PC na década de 90. Foi criado por William Gates, presidente da Microsoft.

- Windows: Sistema projetado pela Microsoft para competir no mercado de SOs. Um projetista importante foi David Cutler.

Finalmente, deve-se lembrar que cada processador tem sua própria linguagem de máquina (linguagem montadora).

O software é uma das áreas de maior polêmica na ciência da computação. Algumas discussões são:

- Linguagem Imperativas (procedurais) x Linguagens Declarativas ou Funcionais: linguagens imperativas são FORTRAN, PASCAL, COBOL, C enquanto PROLOG é declarativa e LISP, funcional. Pesquisadores em computação favorecem LISP e PROLOG, enquanto outros usuários preferem as linguagens imperativas.

- Linguagens não estruturadas (FORTRAN) x Linguagens Estruturadas (PASCAL e C): Pesquisadores em computação preferem PASCAL enquanto outros usuários preferem (FORTRAN).

- Linguagens funcionais x Linguagens declarativas: As opiniões são muito divergentes entre os pesquisadores em arquitetura e inteligência artificial. Entre pesquisadores em arquitetura, uma preocupação grande é quanto ao potencial de paralelismo explorável pela máquina.

- UNIX x VMS x NT: Entre os pesquisadores as opiniões divergem quanto ao melhor sistema operacional em servidores.

## 2.2 Compiladores

Um programa é chamado tradutor se ele toma como entrada um programa escrito numa linguagem A (chamada linguagem *fonte*) e gera um programa numa linguagem B (chamada linguagem *objeto*) funcionalmente equivalente ao programa entrada. Veja figura 2.1.

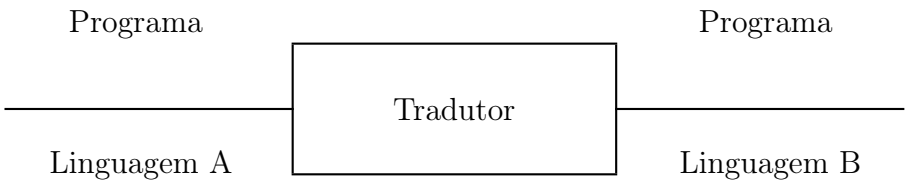



Figura 2.1: Tradutores

Se a linguagem A é de alto nível e B é linguagem de máquina, o tradutor é chamado *compilador*. Veja figura 2.2. Assim:

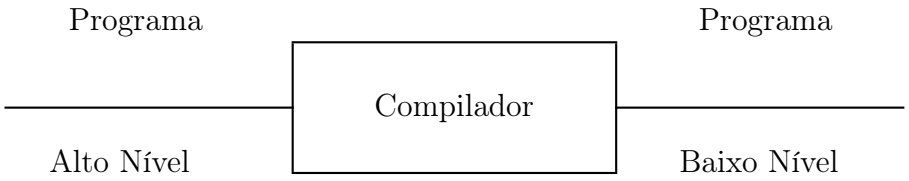



Figura 2.2: Compiladores

O programa em linguagem de máquina pode ser executado no computador. Alguns tradutores tomam o programa de entrada escrito em alto nível e geram um programa em nível *intermediário*. Este último é executado diretamente através de um programa chamado *interpretador*. Linguagens normalmente interpretadas são Java, Python etc. Interpretadores são menores e mais fáceis de escrever que compiladores, mas são muito mais lentos.

### 2.2.1 Fases de Compilação

O processo de compilação é dividido em várias fases:

1. Análise Léxica ("scanner"): classifica as palavras de um programa em várias categorias como palavras-chave (IF, THEN, WHILE, DO, etc), identificadores (X, Volume_Total), operadores (<, =, >, +), delimitadores (vírgula, parêntesis).

2. Análise Sintática ("parser"): junta as palavras em estruturas sintáticas. Assim A*B+C é considerado uma expressão. Várias expressões formam um comando (statement).

3. Gerador de código Intermediário: gera uma sequência de instruções simples num formato pré-definido (que fica próximo da linguagem assembler).

4. Otimizador: tenta melhorar o código produzido no estágio 3.

5. Geração de código: gera finalmente o programa em linguagem de máquina.

Além das fases citadas, têm-se o *gerenciador de tabela* ("table manager") que mantém a "tabela de símbolos" (contendo os nomes usados pelo programa) e a rotina de tratamento de erro ("error-handler").

### 2.2.2 Exemplo de compilação

Como primeiro exemplo, considere o seguinte programa FORTRAN que soma todos os elementos de um vetor e imprime o resultado.

```
      PROGRAM SOMA_VECTOR
      INTEGER X(10)

      ISUM=0
      DO 5 I=1,10
5     X(I)=I
      DO 10 I=1,10
10    ISUM=ISUM+X(I)
      WRITE(5,*)ISUM
      CALL EXIT
      END
```

O compilador FORTRAN para o VAX-11 gera o seguinte código de máquina:

```
.TITLE SOMA_VECTOR
.IDENT 01

    0000 .PSECT $CODE
    0000  SOMA_VECTOR::
    0000 .WORD ^M<IV,R10,R11>
    0002 MOVAL $LOCAL+^X28, R11
    ; 0004
    0009 CLRL R10
    ; 0005
```

```
    000B MOVL #1, R12
    000E  L$1:
    ; 0006
    000E MOVL R12, X-4(R11)[R12]
    0013 AOBLEQ #10, R12, L$1
    ; 0007
    0017 MOVL #1, R12
    001A  L$2:
    ; 0008
    001A ADDL2 X-4(R11)[R12], R10
    001F AOBLEQ #10, R12, L$2
    0023 MOVL R10, ISUM(R11)
    0026 MOVL R12, I(R11)
    ; 0009
    002A PUSHL #5
    002C CALLS #1, FOR$WRITE_SL
    0033 PUSHAL ISUM(R11)
    0035 CALLS #1, FOR$IO_L_R
    003C CALLS #0, FOR$IO_END
    ; 0010
    0043 CALLS #0, FOR$EXIT
    ; 0011
    004A MOVL #1, R0
    004D RET
.END


PROGRAM SECTIONS

    Name  Bytes   Attributes

  0 $CODE                                   78   PIC CON REL LCL   SHR
  EXE   RD NOWRT LONG
  2 $LOCAL                                  48   PIC CON REL LCL NOSHR NOEXE
  RD   WRT LONG

    Total Space Allocated                  126


ENTRY POINTS

    Address  Type  Name

  0-00000000       SOMA_VECTOR
```

```
VARIABLES

    Address  Type  Name              Address  Type  Name

  2-0000002C  I*4  I               2-00000028  I*4  ISUM


ARRAYS

    Address  Type  Name                Bytes  Dimensions

  2-00000000  I*4  X                     40  (10)


LABELS

    Address   Label       Address   Label

      **      5             **      10


FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name

        FOR$EXIT


COMMAND QUALIFIERS

  FORTRAN /LIS/MACHIN ADDVEC

  /CHECK=(NOBOUNDS,OVERFLOW,NOUNDERFLOW)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /STANDARD=(NOSYNTAX,NOSOURCE_FORM)
  /SHOW=(NOPREPROCESSOR,NOINCLUDE,MAP)
  /F77  /NOG_FLOATING  /I4  /OPTIMIZE  /WARNINGS
  /NOD_LINES  /NOCROSS_REFERENCE  /MACHINE_CODE  /CONTINUATIONS=19


COMPILATION STATISTICS

  Run Time:           0.57 seconds
  Elapsed Time:       2.23 seconds
```

```
  Page Faults:        73
```

O scanner separa identificadores (X, I, ISUM), palavras-chave (PROGRAM, INTEGER, DO, WRITE, CALL), operadores (+, =) e delimitadores. Note que o conteúdo da tabela de símbolos é dado na listagem. Note também que o otimizador coloca o valor de ISUM e I em registradores.

Como segundo exemplo de compilação, seja o programa soma.c a seguir, escrito na linguagem C.

```
#include <stdio.h>

main(){
  int sum, i, x[10];

  for (i=1; i<=10; i++) x[i]=i;
  sum =0;
  for (i=1; i<=10; i++) sum=x[i] + sum;
  printf("O resultado: %d\n",sum);
}
```

O comando `cc -S -c soma.c` gera um arquivo soma.s mostrado a seguir (o SO é o FreeBSD 2.1 rodando num pentium):

```
.file "soma.c"
gcc2_compiled.:
___gnu_compiled_c:
.text
LC0:
.ascii "O resultado: %d\12\0"
.align 2
.globl _main
.type  _main,@function
_main:
pushl %ebp
movl %esp,%ebp
subl $48,%esp
call ___main
movl $1,-8(%ebp)
L6:
cmpl $10,-8(%ebp)
jle L9
jmp L7
.align 2,0x90
L9:
movl -8(%ebp),%eax
movl -8(%ebp),%edx
```

```
movl %edx,-48(%ebp,%eax,4)
L8:
incl -8(%ebp)
jmp L6
.align 2,0x90
L7:
movl $0,-4(%ebp)
movl $1,-8(%ebp)
L10:
cmpl $10,-8(%ebp)
jle L13
jmp L11
.align 2,0x90
L13:
movl -8(%ebp),%eax
movl -48(%ebp,%eax,4),%ecx
addl %ecx,-4(%ebp)
L12:
incl -8(%ebp)
jmp L10
.align 2,0x90
L11:
movl -4(%ebp),%eax
pushl %eax
pushl $LC0
call _printf
addl $8,%esp
L5:
leave
ret
Lfe1:
.size  _main,Lfe1-_main
```

# Capítulo 3

# Representação de Dados

Esta característica de arquitetura refere-se ao modo como os valores são representados e tratados. Analisemos primeiramente a aritmética de computadores.

## 3.1 Aritmética de Computadores

No mundo "real" da matemática, a aritmética pode ser definida como uma função de $\mathbb{R}x\mathbb{R}$ em $\mathbb{R}$.

$$f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$

Na computação porém, devido ao número finito de possíveis valores a função torna-se

$$f : M \times M \to M$$

onde M é um subconjunto finito de $\mathbb{R}$.

Devido a M ser finito, é claro que muitos números não podem ser representados exatamente (só aproximadamente) enquanto outros nem podem ser representados (números muito grandes e números muito pequenos).

É preciso ter-se sempre em mente que a aritmética em computadores é uma simplificação da aritmética convencional e consequentemente pode introduzir vários erros (alguns bastante sérios).

## 3.2 Ponto Fixo, Notação Posicional

Uma grande vantagem do sistema decimal utilizado hoje em dia é a "notação posicional". Assim

$$929 = 9 \times 100 + 2 \times 10 + 9 = 9 \times 10^2 + 2 \times 10^1 + 9 \times 10^0$$

Na notação posicional, a contribuição de cada dígito no valor do número depende de sua posição. O primeiro "9" vale 100 vezes mais que o segundo, ou seja, a base ao quadrado.

Um número em ponto fixo pode ser representado por:

$$X = X_{n-1}X_{n-2}...X_0X_{-1}...X_{-m}$$

Na base R, o valor de X seria

$$V(X) = \sum_{i=0}^{n-1} X_i R^i + \sum_{i=-1}^{-m} X_i R^i = \sum_{i=-m}^{n-1} X_i R^i$$

com a condição de que $0 \leq X_i < R$.

Exemplos: (Valor na base 10)

$$V(522)_6 = 2 + 2 \times 6 + 5 \times 6^2 = 2 + 12 + 180 = (194)_{10}$$
$$V(111110)_2 = 1 \times 2 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 = 62_{10}$$
$$V(1.01)_2 = 1 \times 2^0 + 1 \times 2^{-2} = 1 + \tfrac{1}{4} = 1.25_{10}$$
$$V(74)_5 = \text{não válido}$$

Os valores nos cálculos acima podem ser tomados em qualquer base R. Exemplos:

$$V(15)_{10} = 101 \times 1010^0 + 001 \times 1010^1 = 101 + 1010 = 1111$$
$$V(22)_3 = 2 \times 3^0 + 2 \times 3^1 = 2 + 11 = 13_5$$

Para transformar um número de uma base familiar R' para R faz-se:

Parte não fracionária:

$$V(X) = X_0 + X_1R + X_2R^2 + ... + X_{n-1}R^{n-1}$$
$$\tfrac{V(X)}{R} = \tfrac{X_0}{R} + (X_1 + X_2R^1 + ... + X_{n-1}R^{n-2})$$

O resto da divisão V(X) por R é exatamente $X_0$. O quociente $(X_1 + X_2R + ...)$ pode também ser dividido por R e têm-se $X_1$ como o novo resto e assim sucessivamente. Por exemplo:

$$63_{10} \text{ para base } 5$$

$$63 = 12 \times 5 + 3 \Rightarrow X_0 = 3$$

$$12 = 2 \times 5 + 2 \Rightarrow X_1 = 2$$

$$2 = 0 \times 5 + 2 \Rightarrow X_2 = 2$$

$$63_{10} = 223_5$$

Parte fracionária:

$$V(X) = \tfrac{X_{-1}}{R} + \tfrac{X_{-2}}{R^2} + .. + \tfrac{X_{-n}}{R^n}$$

ou

$$R \times V(X) = X_{-1} + \tfrac{X_{-2}}{R} + .. + \tfrac{X_{-n}}{R^{n-1}}$$

$X_{-1}$ não é mais fracionário. Faz-se:

$$F_1 = \frac{X_{-2}}{R} + .. + \frac{X_{-n}}{R^{n-1}}$$
$$R \times V(X) = X_{-1} + F_1$$
$$R \times F_1 = X_{-2} + F_2 \Rightarrow R \times F_{-(i+1)} + F_{i+1}$$

Como exemplo, seja calcular o valor de 0.75 (decimal) em binário.

$$0.75_{10} \times 2 = 1.5 \Rightarrow X_{-1} = 1$$
$$0.5_{10} \times 2 = 1.0 \Rightarrow X_{-2} = 1$$
$$(0.75)_{10} = (0.11)_2$$

Em computação a base mais importante é a binária. Em termos de notação porém, ela é difícil de ser usada. Porém se os dígitos binários (bits) forem agrupados de n em n têm-se a representação em base $2^n$ (Prove!). Exemplo:

$$n = 3\ 011000111010000_2 = 011000111010000 = 30720_8$$

Em hexadecimal

$$011000111010000 = 31C0_{16}$$

As operações aritméticas em bases diferentes da base 10 seguem as mesmas regras da base 10.

## 3.3 Representação de Números Negativos

A representação "natural" seria simplesmente adicionar-se um sinal na frente do número. Essa representação é conhecida como magnitude de sinal. Assim $3 \rightarrow +3$ e têm-se $-3$.

Em cálculos computacionais, a representação em complemento é de mais fácil manipulação.

### 3.3.1 Representação em Complemento

Seja um número de n dígitos base R. Esse número é um entre $R^n$ possibilidades. Se $|b| < R^n/2$ onde b é um número, então convenciona-se:

- Se b é positivo então a representação em R é utilizada.
- Se b é negativo então o número $R^n - b$ é usado.

A aritmética com o uso da representação em complemento fica mais simples. Assim, considere a soma de dois números $b > 0$ e $-c > 0$. As representações são respectivamente, $b$ e $R^n - |c|$. Têm-se:

$$b + R^n - |c| = R^n + b - |c| \text{ se } |b| > |c|$$
e
$$b + R^n - |c| = R^n - (|c| - b) \text{ se } |b| < |c|$$

Se desprezarmos o "carry" no primeiro caso, teremos que o resultado está corretamente representado em complemento. Note que não há necessidade de se saber se o módulo de b é maior ou menor que o de c: o resultado será sempre correto. Como exemplo, considere que R=10 e n=3. Deseja-se 53-101. Assim

$$-101 = 10^3 - 101 = 899$$
$$53 + (-101) = 53 + 899 = 952 = 10^3 - 48$$
$$53 - 101 = -48$$

Essa representação em complemento é chamada de representação em complemento de *dois*.

Na base 2, a representação em complemento de dois é facilmente obtida *complementando-se logicamente* cada bit e somando 1 ao resultado. Outra representação é a de complemento de um onde números negativos são obtidos simplesmente complementando-se logicamente cada bit *sem* se somar um no final. A tabela 3.1 mostra as três representações mencionadas para números binários de 4 bits.

| $b_3b_2b_1b_0$ | Mg Sinal | Compl. 1 | Compl. 2 |
|---|---|---|---|
| 0111 | +7 | +7 | +7 |
| 0110 | +6 | +6 | +6 |
| 0101 | +5 | +5 | +5 |
| 0100 | +4 | +4 | +4 |
| 0011 | +3 | +3 | +3 |
| 0010 | +2 | +2 | +2 |
| 0001 | +1 | +1 | +1 |
| 0000 | +0 | +0 | +0 |
| 1000 | -0 | -7 | -8 |
| 1001 | -1 | -6 | -7 |
| 1010 | -2 | -5 | -6 |
| 1011 | -3 | -4 | -5 |
| 1100 | -4 | -3 | -4 |
| 1101 | -5 | -2 | -3 |
| 1110 | -6 | -1 | -2 |
| 1111 | -7 | -0 | -1 |

Tabela 3.1: Representações de Binários

A notação mais conveniente é a de complemento de dois.

## 3.4 Representação BCD, Binary Coded Decimal

A representação é decimal mas codificada em binário. É muito usada em calculadoras. Exemplo:

$$427_{10} = 0100\ 0010\ 0111$$

A soma em BCD é efetuada em binário (grupos de 4 bits). Se o número resultante da soma é maior que 1001, soma-se seis (0110) a ele e considera-se que *foi um*. Por exemplo:

```
    1
0100     0111    +
0011     0101
-------------
         1100  maior que 1001
         0110
         ----
1000     0010  e vai um
```

## 3.5 Ponto Flutuante

Para representar-se números maiores que o possível em ponto fixo, é necessário utilizar-se ponto flutuante. Na representação em ponto flutuante têm-se uma *mantissa* ou *fração* e um *expoente*. Como exemplo, considere a representação em ponto flutuante no Burroughs B-5500. Fração e expoente são representados em magnitude de sinal.

| Sinal | Sinal do Expoente | Expoente | Fração |
|---|---|---|---|
| 1 bit | 1 bit | 6 bits | 39 bits |
| 0 | 0 | $3_8$ | $000..53_8$ |

O expoente é de oito. Assim, o número no exemplo é:

$$53_8 \times 8^3 = 53000_8 = 22016_{10}$$

# Capítulo 4

# Exemplos de Representação de Dados

## 4.1 Representação de Dados no IBM/370

O sistema IBM 370 apresenta quatro tipos básicos de dados:

- Decimal zona
- Decimal empacotado
- Decimal ponto-fixo
- Decimal ponto-flutuante

### 4.1.1 Decimal Zona

Cada byte representa um dígito. O byte contém 4 bits representando o sinal, que podem ser 1100 ou 1111 (positivos) ou 1101 (negativo), e a representação BCD da magnitude. Assim, o número +1234 seria representado por:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1111 | 0001 | 1111 | 0010 | 1111 | 0011 | 1111 | 0100 |

### 4.1.2 Decimal Empacotado

Um byte contém dois dígitos BCD. O sinal do número é dado no último byte. Por exemplo, o número +1234 é representado por

| | |
|---|---|
| 0000 | 0001 |
| 0010 | 0001 |
| 0100 | 1100 |

### 4.1.3 Binário em Ponto-fixo

Dois tipos de inteiros podem ser representados:

- Half words: 2 bytes são utilizados.
- Full words: 4 bytes são utilizados.

A representação de números negativos é feita em complemento de dois. Assim, -5 em half word é:

| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

O número +25 em full word será:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

### 4.1.4 Ponto flutuante

Três tipos são definidos pela IBM:

- Curto
- Longo
- Extendido

#### Curto

Formado por uma fração com sinal (no bit mais significativo, o bit 0) e por um expoente de 16 representado em excesso de 64.

Por exemplo $23_{10} = 17_{16}$ seria

| S | Expoente | Fração |
|---|---|---|
| 0 | 1000010 | 0001 0111 0000 0000 0000 0000 |
| 0 | 1 7 | 8 31 |

#### Longo

Como o curto só que a fração ocupa do bit 8 ao 63.

#### Extendido

Composto por dois dados longos. Os expoentes e sinais devem ser os mesmos.

## 4.2 Representação de Dados no VAX-11

São definidos 6 tipos de dados no VAX-11:

- Inteiros e ponto flutuante
- Cadeia de Caracteres
- Cadeia Numérica
- Decimais empacotados
- Campos de Bit de largura variável
- Filas

### 4.2.1 Inteiros e Ponto flutuante

São divididos em:

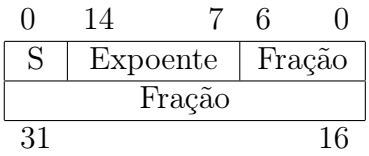
- Byte: composto por oito bits. Pode ser com ou sem sinal; o bit de sinal é o oitavo (mais significativo) quando a representação com sinal é utilizada. Representação de números negativos é feita em complemento de dois.
- Word: composto por 16 bits (dois bytes).
- Longword: 32 bits (4 bytes).
- Quadword: 64 bits (8 bytes).
- Octaword: 128 bits (16 bytes).

Números em ponto flutuante no VAX-11 têm uma parte fracionária com sinal e um expoente de 2. A parte fracionária é sempre um número entre 0.5 e 1.0. Assim, o bit mais significativo é sempre 1 e não é representado. Quatro tipos de números em ponto flutuante podem ser definidos:

- F_Float
- D_Float
- G_Float
- H_Float

**F_Float**

O bit 15 é o do sinal. O expoente é representado em excesso de 128 nos bits de 14 a 7. Bits 6 a 0 e 31 a 16 armazenam a fração. Veja figura abaixo.




Se S=0 e Exp=0, então o número representado é 0.000. O máximo número representável é aproximadamente $1.7 \times 10^{38}$. (Prove!)

**D_Float**

Como o F_Float só que com mais 4 bytes para a fração (bits de 32 a 63 além dos bits de 6 a 0 e 31 a 16).

**G_Float**

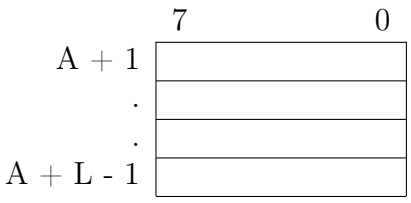
Como o D_Float só que o expoente vai dos bits 14 a 4 representado em excesso de 1024.

**H_Float**

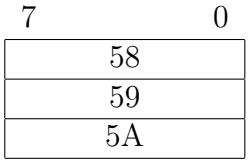
Expoente em excesso de 16384 (bits 14 a 0) e 15 bytes para fração (bits 16 a 127).

### 4.2.2 Cadeia de Caracteres

Sequência de caracteres ASCII especificada pelo endereço inicial (do byte inicial) e o comprimento L.




Como exemplo, a cadeia XYZ pode ser representada por




onde os valores 58, 59 e 5A são os números hexadecimais correspondentes a X, Y e Z no código ASCII.

### 4.2.3 Cadeia Numérica

Uma sequência de caracteres numéricos pode ser representada de diversas formas no VAX-11. Uma das maneiras é o "trailing" onde um sinal é adicionado antes da cadeia (na qual os dígitos estão representados em ASCII). O sinal pode ser $2B_{16}$ que é o positivo ou $2D_{16}$ que é o negativo. Abaixo vê-se a representação do número $-123$.

| | |
|---|---|
| 2 | D |
| 3 | 1 |
| 3 | 2 |
| 3 | 3 |

### 4.2.4 Decimal Empacotado

A sequência de caracteres é armazenada em BCD. Assim dois dígitos podem ser armazenados por byte (um por "nibble"). O sinal positivo é representado por $12_{16}$, $14_{16}$ ou ainda $15_{16}$ enquanto o negativo é $11_{16}$ ou $13_{16}$. Assim, $+123$ é representado por

| 7 4 | 3 0 |
|---|---|
| 0001 | 0010 |
| 0011 | 1100 |

e -12 é:

| 7 4 | 3 0 |
|---|---|
| 0000 | 0001 |
| 0010 | 1011 |

### 4.2.5 Campos de Bit de Largura Variável

Necessitam de três dados para sua completa especificação:

- Endereço base A que é o endereço de um byte.
- Posição de bit P que é um longword (com sinal).
- Tamanho S do campo (um byte sem sinal).

Como exemplo, seja o campo definido por:

- A=B2204C01
- P=29
- S=2

| | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|
| B2204C01 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| B2204C02 | 15 | | | | | | | |
| B2204C03 | 23 | | | | | | | |
| B2204C04 | 31 | 30 | 29 | | | | | |

Assim, o campo é composto pelos bits 5 e 6 do endereço B2204C04 (que são as posições 29 e 30 em relação ao endereço B2204C01).

# Capítulo 5

# Endereçamento e Formato de Instruções

## 5.1 Introdução

Uma instrução pode ser vista como um código de operação seguido de várias especificações de operandos. O número de operandos especificados em cada instrução é uma característica importante da arquitetura de um computador. Muitos operandos por instrução fornece uma certa versatilidade ao programador mas a instrução pode tornar-se muito comprida e complexa.

Para que a instrução não se torne excessivamente comprida, o arquiteto tem duas soluções:

- Reduzir o número de especificações de operandos numa instrução. Assim, quando são necessários mais operandos do que a instrução comporta, utiliza-se mais de uma instrução.
- Tenta-se comprimir o endereço dos operandos ao máximo.

## 5.2 Máquinas de 0, 1, 2 e 3 endereços

Esta classificação se refere ao número de endereços que vêm explicitamente numa instrução. Porém, é difícil dizer-se que uma arquitetura é de n endereços considerando-se que a maioria dos computadores trabalha com diferentes números de operandos dependendo da instrução. Normalmente, utiliza-se uma instrução aritmética como a de soma para classificar-se uma arquitetura.

É importante lembrar também que podem ser distinguidos dois tipos de unidades endereçáveis: a memória e os registradores.

### 5.2.1 Máquinas de 3 endereços

A instrução soma necessita logicamente de três endereços: dois dos operandos a serem somados e um para o resultado. Máquinas que executam plenamente a

soma entre operandos na memória são chamados de máquinas de três endereços. Um exemplo é o VAX 11. Suponha que se queira somar o conteúdo da posição designada por OP1 com o conteúdo da posição apontada por OP2 e colocar o resultado na posição designada por RES. Têm-se:

```
OP1:    .LONG   1
OP2:    .LONG   1
RES:    .LONG   1
         ADDL   OP1, OP2, RES
```

Note como com uma única instrução a soma foi efetuada.

Máquinas de três endereços são extremamente versáteis e eram consideradas "superiores" no final da década de 70. Atualmente porém, com a tendência RISC, essas máquinas já não gozam da mesma popularidade entre os arquitetos.

### 5.2.2 Máquinas de 2 endereços

Como logicamente a operação de soma precisa de 3 endereços, máquinas de dois endereços assumem um dos endereços "implicitamente".

Um dos exemplos de máquina de dois endereços é o PDP-11. Suponha que se quer executar uma soma de OP1 com OP2. Faz-se:

```
ADD OP1,OP2
```

Note que os endereços do segundo operando e do resultado são os mesmos, ou seja, OP2. Se for necessário armazenar o resultado numa posição de memória diferente, uma instrução extra será necessária (como MOV OP2,RES).

Algumas máquinas trabalham com dois endereços mas um deles deve ser um registrador. Assim o microprocessador 68000 da Motorola executa:

```
ADD $1000,D1
```

que soma o conteúdo da posição 1000 com o registrador D1.

Os IBM 360 e 370 também exigem que um dos operandos seja um registrador. Arquiteturas do tipo 68000 e IBM 360 são algumas vezes chamadas de arquiteturas de 1 1/2 endereços.

Em máquinas de 1 1/2 endereços, a soma OP1 + OP2 = RES pode ser escrita como:

```
LOAD    OP1,R1
ADD     OP2,R1
STORE   R1,RES
```

O DEC-10 é um outro exemplo de máquina de um endereço e meio.

### 5.2.3 Máquinas de 1 endereço

Nesse tipo de arquitetura, dois endereços da soma são implícitos. Tipicamente, tem-se que o acumulador (um registrador especial na unidade central de processamento) é tanto um dos operandos quanto o lugar para o resultado. A soma OP1 + OP2 = RES é implementada:

```
LOAD    OP1
ADD     OP2
STORE   RES
```

onde LOAD e STORE são instruções que carregam e armazenam o conteúdo do acumulador. Nessas arquiteturas, o uso de instruções LOAD e STORE é bastante frequente como no caso de arquiteturas 1 1/2. Estas são chamadas de arquiteturas LOAD-STORE.

Algumas arquiteturas exigem que o único endereço da instrução seja também um registrador. Como exemplo, considere a execução de uma soma OP1+OP2=RES no microprocessador 8085. Têm-se:

```
LDA     OP1
MOV     B,A
LDA     OP2
ADD     B
STA     RES
```

### 5.2.4 Máquinas de zero endereços - Máquinas de Pilha

Nessas máquinas TODOS os endereços são implícitos. É usada a estrutura de "pilha" em hardware. Numa máquina de pilha, instruções PUSH e POP são utilizadas para carregar-se e armazenar-se valores na pilha. Assim, a soma OP1+OP2=RES pode ser implementada:

```
PUSH    OP1
PUSH    OP2
ADD
POP     RES
```

Note a ausência de endereços na instrução ADD. Ela soma as duas posições do topo da pilha.

Exemplos de computadores de pilha são os Burroughs B5000, o B6500 e o Hewlett-Packard HP3000.

## 5.3 Formato de Instruções

Vários tipos de codificação de instruções são possíveis. Exemplos importantes são:

1. Burks, Goldstine e von Neumann: Cada posição de memória de 40 bits continha duas instruções de 20 bits sendo 8 bits para código de operação (opcodes) e 12 bits de endereço (havia 4096 posições de memória).

2. CDC 6600 e 7600: os computadores mais rápidos da década de 60. Apresentam dois formatos de instrução (15 bits e 30 bits). O formato da instrução de 15 bits pode ser visto abaixo:

| f | m | i | j | k |
|---|---|---|---|---|
| 3 | 3 | 3 | 3 | 3 |

   Os campos f e m designam o "opcode" da instrução (por exemplo, ADD). Os operandos são j e k (que especificam registradores) e o resultado é colocado em i (também um registrador). No formato de 30 bits, k é substituído por uma constante K de 18 bits.

3. IBM 370: Instruções são compostas de uma, duas ou três "half-words" (16 bits). Os opcodes ocupam sempre os oito bits mais significativos da primeira "half-word". Os outros campos dependem dos vários modos de endereçamento possíveis.

4. Motorola 68000: Os primeiros 16 bits de uma instrução especificam tanto o opcode quanto os modos de endereçamento dos operandos. Extensões de endereçamento fazem com que a instrução possa chegar a 11 palavras.

5. UNIVAC 1107: Cada instrução é acomodada numa posição de memória que contém 36 bits. O formato da instrução é

| f | j | a | x | h | i | u |
|---|---|---|---|---|---|---|
| 6 | 4 | 4 | 4 | 1 | 1 | 16 |

   onde f define o opcode, u é a posição de memória e a especifica o registrador a ser utilizado.

6. VAX 11: Uma instrução é composta de um opcode que é formado por um ou dois bytes e uma série de especificadores de operando (até 6 operandos são possíveis). Assim, a instrução do VAX 11 é de comprimento variável.

## 5.4 Modos de endereçamento

### 5.4.1 Endereçamento Absoluto ou Direto

Nessa forma de endereçamento, a posição de memória é dada diretamente na instrução. Por exemplo, no 8085 existe a instrução:

LDA 2000H

O acumulador será carregado com o conteúdo da posição 2000H. É o modo de endereçamento básico de memória.

### 5.4.2 Endereçamento Imediato

Ao invés do endereço da posição de memória, o próprio dado vem na instrução. Assim no 8085:

MVI C,05H

move o número 05H para o registrador C.

Essa forma de endereçamento é muito útil para se codificar constantes num programa.

### 5.4.3 Indexação

A indexação é especialmente utilizada para se manipular estrutura de dados, tipicamente em "loops".

Assim, o segmento de programa:

```
      DO  10  I=1,10
10    C(I)=A(I)+B(I)
```

sem o uso de indexação (ou indireção como será visto a seguir) só poderá ser implementado como uma sequência de 10 operações de soma ou com instruções automodificantes, como sugeria von Neumann.

A indexação consiste em se somar um offset, normalmente armazenado num registrador, a um valor base dado, por exemplo, na instrução.

Como exemplo, considere a instrução de VAX 11:

```
CLRB    @#^X1012[R2]
```

O endereço 1012 é modificado pelo conteúdo de R2. Suponha que R2=$5_{16}$. Então, o endereço efetivo será:

$$1012_{16} + 5_{16} = 1017_{16}$$

Depois da execução dessa instrução, a posição 1017 da memória (byte) será zerado.

Uma variação interessante do mecanismo de indexação é o autoincremento ou autodecremento do registrador de índice, ou seja, a própria execução da instrução modifica o valor do registrador de índice.

Considere o UNIVAC 1107 e o seu formato de instrução:

| f | j | a | x | h | i | u |
|---|---|---|---|---|---|---|
| 6 | 4 | 4 | 4 | 1 | 1 | 16 |

Os registradores X são compostos por duas partes:

| $X_i$ | $X_m$ |
|---|---|
| 18 | 18 |

Se o campo x da instrução não for zero, o campo $X_m$ do registrador X especificado será somado ao valor de u para se obter o endereço efetivo da memória. Se o bit h estiver em 1, além da operação citada, é executado $X_m \leftarrow X_i + X_m$ onde $X_i$ é interpretado com sinal. Assim, o valor $X_m$ também é modificado.

### 5.4.4 Indireção

Um endereço indireto é aquele que contém o endereço do dado desejado e não o próprio dado. O opcode ou a especificação do operando indicam se a indireção deve ou não ser utilizada. Assim, depois de se calcular o endereço efetivo (com índice por exemplo) o endereço ainda passa por outra etapa para conseguir-se o operando.

O endereçamento indireto pode ser utilizado para implementação de loops (como ?). Pode ser utilizado também para relocabilidade de dados referenciados pelos programas.

Alguns projetistas acreditam que o endereçamento indireto é ineficiente pois mais de um acesso a memória é necessário. Como exemplo de endereçamento indireto, considere novamente o UNIVAC 1107. O bit i na instrução indica se a indireção deve ser efetuada. Têm-se assim, vários níveis de indireção e em cada estágio, nova indexação é possível permitindo cálculos bastante complexos de endereços.

O HP2100 também apresenta indireção em vários níveis.

### 5.4.5 Endereçamento de Página, base e segmentação

Registradores de índice fornecem um offset para o endereço dado numa instrução. Registradores de base fornecem um endereço que deve ser somado com o offset dado na instrução.

Podem-se distinguir dois tipos de espaços de memória de um processador:

- O espaço endereçável diretamente por uma instrução.
- O espaço total passível de ser endereçável pelo processador.

Em muitos processadores, ambos os espaços citados são coincidentes como por exemplo, no VAX-11. Já no microprocessador 8086 da INTEL, o espaço endereçável por uma instrução é limitado em 64K bytes enquanto o espaço total de endereçamento é de 1M byte. Nessa arquitetura, são definidos registradores de base que só alteram o endereço de um operando dentro dos 64K bytes (isto é, a instrução fornece um offset de 64K bytes para um registrador de base com valor máximo de 64K bytes; os 64K bytes do espaço endereçável são considerados circulares). Já registradores de *segmento* são interpretados como endereços de 1M byte (20 bits dos quais só os 16 *mais* significativos são representados sendo os 4 menos significativos sempre considerados com zero). Repare porém, que o endereçamento por *segmento* é conceitualmente idêntico ao por *base*.

Quando o endereço é obtido simplesmente pela concatenação do valor de um registrador com o offset dado na instrução (por exemplo, um registrador contém os oito bits mais significativos e a instrução fornece os oito bits menos significativos), fala-se em endereçamento de página.

O uso de registradores de base, segmentação e paginação são bastante utilizados para se separar as diferentes "seções" de um programa tais como código, dados,

pilha, etc tornando mais fácil o controle de proteção de memória. Assim, o processador pode proibir que um programa modifique os registradores de segmento, tornando o acesso a certas áreas do espaço de endereçamento, impossível.

Deve ser notado porém, que modos de endereçamento como segmentação não são os únicos métodos possíveis de proteção de memória numa arquitetura. Na verdade, a proteção via modo de endereçamento pode ser indesejável porque:

- É interessante permitir-se que qualquer programa tenha acesso a todo (ou quase todo) o espaço de endereçamento.
- Pode-se desejar proibir o acesso a porções de memória menores do que os possíveis pelo uso de registradores de segmento.
- Existe a necessidade de se permitir acesso a uma determinada posição somente para leitura e não para escrita ou vice-versa. Assim, certas posições devem ser acessadas em determinados casos mas não em outros o que é difícil de ser implementado somente com restrições no endereçamento.

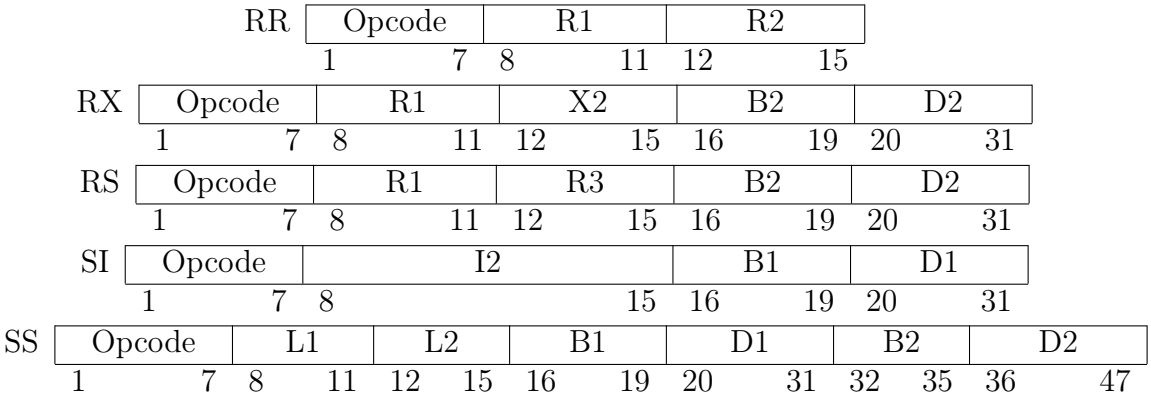
# Capítulo 6

# Exemplos de Modo de Endereçamento

## 6.1 IBM 370

Instruções no IBM 370 podem ser de 5 formatos diferentes designados pelos códigos RR, RX, RS, SI e SS. RR representa operações entre registro e registro, RX, entre registro e posição de memória indexada, RS, entre registro e posição de memória, SI entre posição de memória e dado imediato e SS entre memória e memória.

Os formatos de instrução são:




onde as letras R, X, B, D, L, I representam:

- R: registrador geral da UCP usado como operando
- X: registrador geral da UCP usado como registrador de índice
- B: registrador geral da UCP usado como registrador de base
- D: deslocamento ou "offset"
- L: comprimento em operações sobre dados de comprimento variável
- I: valor imediato

Os registradores gerais da UCP são de 32 bits mas só os 24 bits menos significativos são utilizados na aritmética de endereços.

Como exemplo, considere uma instrução do tipo RX que tem formato assembler:

Op R1,D2(X2,B2)

Um exemplo de tal instrução é o ADD representado por A. Suponha que R1 é o registrador 3, D2=200, X2 é o registrador 4 e B2 é o registrador 5.

Quando da execução da instrução, o conteúdo do registrador 3 é somado ao conteúdo da posição de memória apontada por 200 mais conteúdo do registrador 4 mais conteúdo do registrador 5 (24 bits menos significativos) e o resultado da operação é colocado no registrador 3.

## 6.2 VAX 11

Modos de endereçamento no VAX dividem-se em dois grupos: modos gerais de endereçamento e endereçamento de desvio.

### 6.2.1 Modos gerais de endereçamento

#### Por registrador

O conteúdo do registrador especificado é acessado. O formato é 5Rn onde Rn especifica um dos 16 registradores de uso geral de R0 a R15 (R15 é o PC e não pode ser especificado).

Exemplo: MOVW R1,R2 cujo código é B0 51 52.
Antes: R1=4597AF3B R2=3279245A
Depois: R1=4597AF3B R2=3279AF3B

#### Registrador deferido

O conteúdo do registrador é o endereço do dado. Apresenta um nível de indireção em relação ao endereçamento por registrador. O formato é 6Rn (Rn diferente de PC).

Exemplo: CLRL (R4) cujo código é D4 64
Antes: R4=00002000 2000: AB CD EF 55 32
Depois: R4=00002000 2000: 00 00 00 00 32

#### Autoincremento

O registrador R contém o endereço do operando como no modo deferido, mas depois da execução da instrução o valor de R é incrementado pelo tamanho do operando. O formato é 8Rn.

Exemplo: MOVL (R1)+,R2 cujo código é D0 81 52
Antes: R1=00001010 R2=00000000 00001010: 00 11 22 33 44 55
Depois: R1=00001014 R2=33221100 00001010: 00 11 22 33 44 55

### Autoincremento deferido

O registrador Rn contém o endereço de um "longword" que é o endereço do dado requerido. Depois da execução da instrução, 4 é somado a Rn. O formato é 9Rn.

Exemplo: MOVW @(R1)+,R2 cujo código é B0 91 52
Antes: R1=00001010 R2=00000000 00001010: 00 11 22 33 44 33221100: 34 5F 00
Depois: R1=00001014 R2=00005F34 memórias inalteradas

### Autodecremento

O tamanho do dado é subtraído de Rn que passa então a conter o endereço do dado. O formato é 7Rn (Rn diferente de PC).

Exemplo: MOVL -(R3),R4 cujo código é D0 73 54
Antes: R3=00001018 R4=00000000 00001014: 10 32 54 CE
Depois: R3=00001014 R4=CE543210 memórias inalteradas

### Literal

O dado é especificado na própria instrução. O formato é 0N, 1N, 2N ou 3N onde N é um número hexadecimal qualquer (entre 0 e F). Assim:

| 0 | 0 | Literal |
|---|---|---|
| 7 | 6 | 5 0 |

Exemplo: MOVL S^ #9,R4 cujo código é D0 09 54
Antes: R4=00000000
Depois: R3=00000009

### Deslocamento

O deslocamento dado na instrução é somado com o valor de Rn e o resultado é o endereço do dado. O deslocamento pode ser um byte, uma word ou uma longword. Os formatos são ARn seguido de um byte (deslocamento de byte), CRn seguido de uma word (deslocamento de word) e ERn seguido de uma longword (deslocamento de longword).

Exemplo: MOVB B^ 5(R4),B^ 3(R3) cujo código é 90 A4 05 A3 03
Antes: R4=00001012 R3=00002020 1015: 00 00 06 00 2020: 00 00 00 00
Depois: R4=00001012 R3=00002020 1015: inalterado 2020: 00 00 00 06

### Deslocamento deferido

O deslocamento é adicionada a Rn e o resultado é o endereço do endereço do dado. Os formatos são BRn, DRn, FRn (byte, word e long respectivamente).

Exemplo: INCW @B^ 5(R4) cujo código é B6 B4 05
Antes: R4=00001012 00001017: 88 42 24 68 68244288: 13 57
Depois: R4=00001012 68244288: 14 57

### Indexado

A especificação do endereço conterá pelo menos dois bytes: o primeiro é do formato 4Rn (com Rn diferente de PC) e indica o registrador de índice. O conteúdo desse registrador é multiplicado pelo tamanho do dado e somado ao endereço especificado no segundo byte. O endereçamento do segundo byte pode ser através de qualquer método visto até aqui com exceção do por registrador, o literal ou o próprio indexado.

Exemplo 1: Registrador deferido indexado.
INCW (R2)[R5] cujo código é B6 45 62
Antes: R4=00001012 R5=00000003 00001018: 45 67
Depois: R4=00001012 R5=00000003 00001018: 46 67
Exemplo 2: Autoincremento deferido indexado.
CLRW @(R4)+[R5] cujo código é B4 45 94
Antes: R4=00001012 R5=00000005 00001012: 43 21 08 06 0608214D: 22 33
Depois: R4=00001016 R5=00000005 0608214D: 00 00
Exemplo 3: Deslocamento deferido indexado.
MOVL @Bˆ 14(R1)[R3],R5 cujo código é D0 43 B1 14 55
Antes: R1=00001012 R3=00000004 1026: 11 22 33 44 44332221: 01 23 45 67
Depois: R1=00001012 R3=00000004 R5=67452301

# Capítulo 7

# Exemplos de Modo de Endereçamento (Cont.) e Organização de Registro

## 7.1 Endereçamento via Contador de Programa

Em alguns modos de endereçamento anteriores, o PC pode ser especificado como o registrador utilizado. Esses modos de endereçamento recebem nomes especiais quando o PC é utilizado.

### 7.1.1 Imediato

É o modo autoincremento onde o PC é utilizado como registrador. O formato é $8FH$ seguido do dado imediato.

Exemplo: MOVL #6,R4 cujo código é D08F06000000

Antes: R4=00000000 Depois: R4=00000006

### 7.1.2 Absoluto

É o modo autoincremento deferido com o PC utilizado na especificação de registrador. O dado contido na instrução é o endereço absoluto de uma posição de memória (não importa onde a instrução é executada na memória). O formato é $9FH$ seguido do endereço absoluto.

Exemplo: CLRL @#^ X674533 cujo código é D4 9F 33 45 67 00

Antes: 00 67 45 33: 23 45 72 83 Depois: 00 67 45 33: 00 00 00 00

### 7.1.3 Relativo

É o modo "deslocamento" com o PC como registrador especificado. O deslocamento que segue a especificação de endereço deve ser somado ao PC para se ter o endereço do dado. É útil para se escrever código independente da posição (relocável). O formato é $AFH$, $CFH$ e $EFH$ (deslocamento de byte, word e longword) seguido do deslocamento.

Exemplo: MOVL ^ X2016,R4 cujo código é 1012: D0 CF 00 10 54

Antes: R4=00000000, 2016: 77 00 86 00 Depois: R4=00860077, 2016: 77 00 86 00

### 7.1.4 Relativo deferido

Corresponde ao modo deslocamento deferido com o PC especificado como registrador. É semelhante ao relativo só que o deslocamento + PC fornece não o endereço do dado mas o endereço do endereço do dado. O formato é $BFH$, $DFH$, $FFH$ seguido do deslocamento.

Exemplo: MOVL @^ X2050,R2 cujo código é 2000: D0 BF 4D 52

Antes: R2=00000000 2050: 00 60 00 00 6000: 67 45 23 01 Depois: R2=01234567 e memórias inalteradas

### 7.1.5 Endereçamento de Desvio

Nesse modo de endereçamento, o byte ou word que se segue a instrução é somado com o valor do PC. O resultado é colocado no PC, ou seja, é o endereço da próxima instrução a ser executado.

Exemplo: Programa que vai para NOT se C não contiver um dígito de 0 a 9 em ASCII (C é tratado sem sinal).

```
CMPB    C,#^ A/0/
BLSSU   NOT
CMPB    C,#^ A/9/
BGTRU   NOT
```

## 7.2 Organização de Registro (Registrador)

A arquitetura v. Neumann contém 3 elementos fundamentais: a unidade central de processamento, a memória e dispositivos de entrada/saída. Os programas e dados são armazenados na memória. A comunicação UCP e memória é sobrecarregada e é chamada de gargalo de von Neumann. A comunicação UCP memória deve então ser reduzida ao máximo. Uma das maneiras é manter-se na UCP os dados mais frequentemente referenciados. Estes dados são armazenados em "registradores". Assim, registradores são fundamentais em arquiteturas von Neumann.

Os registradores são locais de armazenamento como as posições de memória; podem ser "endereçados" por uma instrução. Porém, como o número de registradores é muito menor que o número de posições de memória, o número de bits necessários para a sua especificação também é bastante menor. Portanto, o acesso a registradores é mais rápido que o acesso a memória por 3 motivos:

1. Registradores geralmente já se encontram na UCP e nenhuma comunicação com a memória é necessária.

2. Registradores são geralmente implementados com tecnologia bem mais rápida que a utilizada nas memórias

3. Devido aos poucos bits necessários para o endereçamento, a decodificação e execução de instruções que envolvem registradores pode ser muito mais rápida (devido a menor complexidade da unidade de controle).

O estudo da organização de registro envolve:

- Estudo do tamanho e número de registradores (capacidade de armazenamento de dados na UCP).

- Estudo da finalidade de grupos de registradores

O tamanho de registradores é normalmente igual ao tamanho do barramento interno de dados da unidade lógico aritmética. Portanto, o comprimento em bits dos registradores da UCP classificam a arquitetura em arquitetura de 8, 16, 32 (etc..) bits.

Registradores podem ser de vários tipos:

1. Registradores para uso em operações aritméticas também chamado de registradores de operando.

2. Registradores de endereçamento, utilizados na aritmética de endereços para acesso de dados na memória. Nessa categoria incluem-se registradores de índice, de página, de segmento, contador de programa, registrador de manipulação de pilha, etc...

3. Registradores de controle de estado: indicam em que estado se encontra o processador. Por exemplo várias arquiteturas definem um conjunto de flags que revelam se a última instrução teve como resultado um número negativo, ou zero ou ainda se houve "overflow". Também podem indicar em que prioridade de interrupção se encontra o processador (só interrupções de maior prioridade que a atual serão atendidas), se o processador executa instruções no modo passo a passo, etc.

Processadores podem adotar 2 filosofias quanto a utilização desses vários tipos de registradores:

1. Existe um conjunto de registradores gerais que executam as funções dos vários tipos de registradores (ou pelo menos, as funções de registrador de operando e de endereçamento)

2. Vários conjuntos de registradores são definidos, cada um desempenhando uma função específica.

Um exemplo da filosofia 1 é verificado no VAX. Seus 15 registradores são de uso geral e podem ser usados tanto como operandos quanto como registradores de endereçamento (é claro, com algumas restrições).

Os computadores UNIVAC, CDC (Cyber) e os CRAY, por outro lado, utilizam-se de conjuntos de registradores diferentes para cada função.

O IBM 370, como o VAX, apresenta um conjunto de registradores que pode servir como operando e como endereço. No 370, entretanto, registradores especiais são utilizados se o operando for representado em ponto flutuante.

A utilização de registradores de operando varia bastante de máquina para máquina. Em algumas arquiteturas, só é definido um acumulador que passa a representar tanto um operando como o lugar onde o resultado de uma operação aritmética será colocado. Outras arquiteturas apresentam um conjunto geral de registradores e comunicação extensiva dos registradores com a memória. O IBM 370 inclui-se nessa classe.

Algumas arquiteturas exigem que todos os operandos estejam em registradores. É o caso dos CDC e CRAY. Note a diferença entre os dois códigos seguintes:

```
LOAD    R1,Mem1
ADD     R1,Mem2
STORE   R1,Mem3
```

que pode ser rodado num IBM 370 e

```
LOAD    R1,Mem1
LOAD    R2,Mem2
ADD     R1,R2,R3
STORE   R3,Mem3
```

que pode ser rodado num CDC 7600 (com outros opcodes!).

No primeiro caso, note que um só registrador foi suficiente enquanto no segundo pelo menos dois seriam necessários.

Algumas arquiteturas usam registradores diferentes para diferentes tipos de operando. Além do IBM 370 já citado (que usa registradores de ponto flutuante diferentes dos de manipulação de inteiros), tem-se o exemplo do Cray-1 que usa registradores para valores escalares (só contendo um elemento) e valores vetoriais (compostos por vários elementos).

Arquiteturas variam também quanto ao método de carregamento de valores num registrador. Algumas permitem que o conteúdo do registrador seja modificado apenas parcialmente. Assim, um registrador de 32 bits pode ter somente os 8 bits menos significativos modificados no VAX. Nesse computador, assim como no IBM 370 e no 8086, o opcode da instrução é que determina quanto do registrador é modificado. Já no UNIVAC 1100, um dos campos da palavra de instrução (j) determina quantos bits serão alterados.

As arquiteturas CDC e Cray apresentam uma forma diferente de carregamento de valores em registrador. Ao invés de se carregar o registrador de operando, carrega-se um registrador de endereço com o endereço do dado que se deseja. O

registrador de dado associado a esse registrador de endereço (existe uma correspondência biunívoca) é carregado com o dado.

Para facilitar o trabalho do processador em ambientes de multiprogramação, o conceito de conjuntos múltiplos de registradores foi desenvolvido. O conjunto de registradores é replicado várias vezes na UCP e cada programa utiliza-se de um conjunto. Quando o processador muda de programa sendo executado, ele simplesmente começa a utilizar um outro conjunto de registradores. Esse método evita os salvamentos e recuperações dos valores dos registradores cada vez que o processador muda de contexto.

Um exemplo de múltiplos registradores é encontrado em alguns processadores construídos na filosofia RISC, como o RISC I. Este processador construído em Berkeley tem vários conjuntos de registradores e alguns registradores são comuns a mais de um conjunto de registradores facilitando a passagem de parâmetros de uma subrotina a outra. Como a passagem de parâmetros pode representar boa parte do tempo de processamento em certos programas (principalmente nos recursivos; na torre de Hanoi as chamadas de subrotina ocupam 90% do tempo num sistema baseado em 68000).

Uma simplificação do modelo de múltiplos registradores é a utilização de apenas dois conjuntos. Esse modelo é particularmente útil para o tratamento de interrupções pois o salvamento de registradores não é necessário sendo suficiente mudar-se de conjunto de registradores utilizados. Exemplos são o UNIVAC 1100 e o microprocessador Z-80.

### 7.2.1 Pilhas de Operando

Em algumas arquiteturas (Burroughs 5000, 6500 etc) o conjunto de registradores é organizado em forma de pilha de tal forma que somente o registrador apontado pelo registrador "topo da pilha" é acessável num determinado instante.

A execução do comando A=B+C é feita através de

```
1000    PUSH    B
1001    PUSH    C
1002    ADD
1003    PUSH    end A
1004    POP
```

e os registradores ficam

```
depois de 1000          depois de 1001          depois de 1002

 1 -                     1 -                     1 -
 2 -                     2 -   <- topo           2 -
 3 -     <- topo         3 - C                   3 -    <- topo
 4 - B                   4 - B                   4 - A
```

```
depois de 1003                  depois de 1004

1 -                                 1 -
2 -                                 2 -
3 - end A                           3 -
4 - A                               4 -    <- topo
```

Uma grande vantagem da arquitetura por pilha de operando é que ela consegue apoiar eficientemente o processo de compilação (com notação posfixa ou seja, operando operando função). Assim A=B+C*D é compilado para ABCD*+=. A passagem para código de máquina pode ser:

```
PUSH D
PUSH C
MULT
PUSH B
ADD
PUSH end A
POP
```

Uma grande desvantagem do modelo de pilha é que dificulta a exploração de paralelismo. Por exemplo, considere o programa:

```
A=B+C
D=E+F
```

Os dois comandos podem ser executados concorrentemente pois são independentes. Numa máquina de pilha, porém, isto torna-se difícil pela própria organização sequencial da pilha. Por exemplo, considere o código abaixo que corresponde a um possível código de máquina de pilha.

```
PUSH B
PUSH C
PUSH E
PUSH F
ADD
PUSH end D
POP
ADD
PUSH end A
POP
```

A execução paralela dos ADD seria difícil. Numa máquina de 1 1/2 endereço, por outro lado, poderíamos fazer:

```
LOAD R1,B
LOAD R2\c{C}
LOAD R3,E
LOAD R4,F
ADD  R1,R2
ADD  R3,R4
STORE R2,A
STORE R4,D
```

e agora se o hardware apresenta mais de uma unidade de soma, os dois ADD podem ser executados em paralelo.

# Capítulo 8

# Exemplos de Organização de Registradores, Conjunto de Instruções e Ligação a Subrotina

## 8.1 Exemplos de Organização de Registradores

### 8.1.1 VAX

A arquitetura VAX contém 16 registradores de uso geral de 32 bits. Eles são denotados por Rn com n entre 0 e 15. Alguns registradores têm significado especial.

- R15=PC (Program Counter): é usado como contador de programa e consequentemente é modificado a cada instrução.
- R14=SP (Stack Pointer): é usado como ponteiro de pilha e endereçado por várias instruções que fazem uso implícito da pilha como chamadas de subrotina, retorno de subrotina, etc.
- R13=FP (Frame Pointer): aponta para uma estrutura de dados montada na chamada de cada subrotina.
- R12=AP (Argument Pointer): aponta para uma região de memória onde argumentos de subrotina são salvos.
- R6 a R11: registradores de uso geral.
- R0 a R5: registradores de uso geral masa usados implicitamente por certas instruções complexas.

Registradores com alto número são bastante globais sendo necessários ao software e ao hardware. Registradores com baixo número são considerados temporários e nem são salvos na passgem de uma subrotina para outra.

O VAX mantém também um registrador especial de STATUS, o PSL que contém 32 bits. É composto por vários campos como por exemplo:

- bits 3 a 0: mantém "códigos de condição" do processador. Têm-se
  - N (bit 3): é o código de Negativo.
  - Z (bit 3): é o código de Zero.
  - V (bit 3): é o código de Overflow.
  - C (bit 3): é o código de Carry.
- Bits 7 a 4: reservados para "traps"
- Bits 20 a 16: dão o IPL (Interrupt Priority Level) ou seja, o nível de prioridade da interrupção.
- Bits 25 a 24: determinam o modo que o programa está executando (existem 4 modos: kernel, supervisor, executive e user).
- Bit 31: Compatibilidade. Quando em 1, o processador executa as instruções do PDP-11.

### 8.1.2 CRAY-1

Todos os operandos devem estar em registradores. O conjunto de registradores é composto de:

- Um conjunto de 8 registradores de operando de 64 bits para escalares (registradores S).
- Um conjunto de 8 registradores de 64 palavras (registradores V).
- 8 registradores de 24 bits (registradores A).
- 64 registradores de salvamento escalar (24 bits) (registradores T).
- 64 registradores de salvamento de endereço (registradores B).

Existem aproximadamente 4 kbytes (4888 bytes) de registradores com tempo de acesso de 6 ns. Registradores A podem ser usados em indexação, contagem de deslocamentos, etc.

A memória do CRAY-1 é composta por 2 a 24 palavras de 16 bits ou 2 a 22 palavras de 64 bits sendo cada 16 bits chamados de "parcel" (pacote). Alguns registradores de suporte são:

- VM: registrador de 64 bits que mascara os elementos de um vetor com cada bit correspondendo a um elemento.
- P: contador de programa contendo 24 bits.
- BA (Base Address): registrador de base que contém os 18 bits mais significativos de um endereço de 22 bits. Os endereços são somados a BA para se ter o endereço absoluto da memória.

- LA (Limit Address): contém os 18 bits mais significativos de um endereço de 22 bits. Acesso a posições superiores a LA são proibidos. Juntamente com BA, implementa um esquema de proteção de memória.

- Registrador F: contém 9 bits de flag que indicam a ocorrência de algum tipo de erro no processamento.

## 8.2 Conjunto de Instruções

O estudo do conjunto de instruções executáveis por uma máquina é tópico fundamental no estudo de sua arquitetura.

Várias maneiras existem para se classificar as instruções de uma determinada máquina. A classificação mais comum é através do tipo de operação executada. Têm-se:

1. Instruções de transporte de dados: são os LOAD, STORE, MOVE que copiam um dado de uma posição de memória ou registrador para outra posição de memória ou registrador.

2. Instruções aritméticas: realizam as operações aritméticas como soma, subtração, multiplicação e divisão (ADD, MPY, DIV, etc).

3. Instruções Lógicas: realizam operações lógicas sobre um ou dois dados (complemento, ou, e ou exclusivo, etc..).

4. Instruções de deslocamento: Deslocam um dado à esquerda ou à direita por um número determinado de bits.

5. Instruções de Edição: Como as instruções de transporte só que também processam os dados transportados (tipicamente executam operações lógicas com os dados transportados).

6. Instruções de auxílio de "loop". Algumas arquiteturas definem instruções complexas que incrementam ou decrementam posições de memória ou registrador e automaticamente testam uma condição do resultado. Dependendo do teste, o loop é repetido.

7. Instruções de mudança de fluxo: fornecem uma maneira de se alterar o fluxo normal de controle da arquitetura von Neumann. Exemplos típicos são os "Branch", "Jump", etc..

8. Instruções de manipulação de pilha: utilizam-se de uma estrutura de daods tipo LIFO (Last In First Out). A inserção de um dado é conseguida com PUSH e a remoção com POP.

9. Instruções de ligação à subrotina: Instruções de mudança de fluxo como CALL e RET. CALL, em muitas arquiteturas, salva automaticamente o estado do máquina, salva registradores e executa a passagem de parâmetro.

10. Instruções de ES: instruções que diferecionam dados para os periféricos.

11. Instruções de Controle ou privilegiadas: instruções que modificam o estado da máquina. Só podem ser executadas por processos com os necessários privilégios.

### 8.2.1 Simetria no Conjunto de Instruções

Medida da "regularidade" ou da "ortogonalidade" de um conjunto de instruções. Simetria perfeita é atingida quando as mesmas operações são disponíveis para todos os tipos de dados, são disponíveis anto em registradores como na memória e com todos os tipos de endereçamento. As condições de estado também devem ser "setadas" com operações sobre qualquer tipo de dados.

Para muitos arquitetos a simetria é uma grande qualidade do conjunto de instruções. Exemplo de simetria é encontrado no VAX-11. Já no 8086, a assimetria é marcante e em várias instruções somente algums dos registradores podem ser especificados. A simetria traz vantagem para:

- Compilador: que tende a produzir códigos melhores e mais rapidamente.
- Programador em Assembler: o trabalho de entendimento de um conjunto de instruções simétrico é muito mais rápido e fácil que o de um conjunto assimétrico.

## 8.3 Ligação a Subrotina

Um conjunto de instruções agrupados sob um determinado nome, conjunto este que pode ser executado várias vezes durante a execução de um programa (pela simples chamada do nome) é chamado de subrotina ou procedimento. A utilização de subrotinas traz 2 vantagens:

- Código menor: de fato, sem o uso de subrotinas a mesma atividade será codificada várias vezes ocupando muito espaço.
- Códigos escritos com subrotinas estimulam a escrita de programas mais claros e fáceis de "manter".

O mecanismo de passagem de controle e de dados de um programa para uma subrotina é chamado de "ligação a subrotina". A ligação a subrotina envolve:

- Um nome da subrotina pela qual esta pode ser referenciada.
- Um mecanismo de controle de fluxo que permita que o programa transfira o controle para a subrotina e que esta retorne o controle para o programa quando de seu término.
- Um mecanismo de passagem de dados de e para a subrotina
- Convenções de uso de variáveis locais e globais pela subrotina.

### 8.3.1 Instruções de Ligação

Duas instruções básicas devem ser definidas para ligação a subrotina:

- Uma instrução que transfira o controle do programa solicitante para a subrotina. Esta instrução deve salvar de alguma forma o endereço de retorno para onde o fluxo deverá ser transferido quando a subrotina terminar.
- Uma instrução para retornar o controle ao programa principal

A primeira instrução é tipicamente chamada de CALL enquanto a de retorno é chamada de RET. O endereço de retorno salvo pela instrução CALL pode ser armazenado em vários locais:

- Um endereço fixo na memória
- Um endereço na memória especificado pela instrução CALL
- Um registrador fixo
- Um registrador especificado no CALL.

O uso de registradores fixos impede o aninhamento de subrotinas (subrotina A chamando a B que chama a C etc..) além de um determinado nível (dado pelo número de registradores disponíveis para armazenamento de endereços de retorno). Exemplo de uso de registradores fixos é o 8008 da Intel.

Um método utilizado antigamente era o de se armazenar o endereço de retorno na primeira posição da subrotina chamada. Este método foi utilizado na arquitetura do HP2100. Aqui existe também uma dificuldade na implementação de recursividade (quando uma subrotina chama-se a si mesma).

O método mais comum atualmente é a utilização de uma estrutura de dados tipo LIFO para salvamento de endereços de retorno. Assim, o CALL, além do desvio, faz um PUSH do endereço de retorno. A instrução RET executa um POP. A diferença para o PUSH e o POP comuns é que aqui, é sempre o PC o registrador envolvido.

### 8.3.2 Passagem de Parâmetros

Muitas convenções de passagens de parâmetros podem ser adotadas. Um método simples é de se passar utilizando-se dos registradores. Em algumas arquiteturas, instruções que modificam e salvam vários registradores são disponíveis.

A pilha pode também ser utilizada com essa finalidade. Os parâmetros a serem passados são colocados na pilha pelo programa solicitante e retirados pela subrotina.

Algumas arquiteturas utilizam-se de um registrador que aponta para uma área de dados reservada para uma derterminada ativação da subrotina. Esse registrador é chamado de "ativação" e é importante para a implementação de código recursivo.

# Capítulo 9

# Interrupções

## 9.1 Definição

Dá-se o nome de interrupção a uma alteração do fluxo normal de execução de instruções de um programa devido a um evento externo. Este evento é *assíncrono* em relação ao programa sendo executado, ou seja, o tempo exato de sua ocorrência não pode ser previsto pela unidade central de processamento.

Quando uma interrupção é atendida, ou seja, quando a UCP assume que ela é mais importante que o programa sendo executado, uma rotina de "serviço" de interrupção é invocada. Como no caso de uma chamada de subrotina, o PC da próxima instrução deve ser salvo para que seja possível voltar-se ao programa quando a rotina de serviço terminar. É comum também salvar-se o "status" do processador, pois é comum que este seja modificado pelo atendimento à interrupção.

Fontes comuns de interrupção são:

1. Dispositivos de entrada e saída que devam ser atendidos com alta prioridade. Este é o caso "clássico" de pedido de interrupção sendo particularmente importante em sistemas de tempo real. Como exemplo, suponha um computador sendo usado para controle de processo numa indústria química. Suponha que um forno deva ser controlado de tal forma que sua temperatura nunca ultrapasse um determinado limite. Quando um sensor detectar que a temperatura esta se elevando em demasia, ele deve interromper o computador para que este tome as medidas necessárias. A prioridade desta interrupção é alta pois as consequências de uma temperatura muito alta podem ser desastrosas. Note que este evento é assíncrono pois não há como a UCP possa saber quando a temperatura do forno vai se elevar.

2. Problemas graves de hardware. Aqui incluem-se erros de paridade de memória, falha de fonte de alimentação, ou incerteza quanto a validade de certas estruturas de dados fundamentais para o correto funcionamento do sistema operacional (por exemplo, overflow de pilha do sistema).

3. Instruções que não podem ser executadas ou por não terem sentido lógico (como uma divisão por zero, uma multiplicação cujo resultado é maior que o máximo possível de ser armazenado), ou por ser uma instrução privilegiada.