Organização de Computadores

Apostila 6

2011

CAPÍTULO

6 REPRESENTAÇÃO DAS INSTRUÇÕES

Assuntos

UD VI – Representação das Instruções

- 6.1 Introdução
- 6.2 Quantidade de Operandos
- 6.3 Modos de endereçamento

Objetivos

- Conhecer a quantidade de Operandos das Instruções;
- Conhecer os modos de endereçamento; e
- Saber realizar operações com Instruções de Máquina

5.1 Introdução

Conforme já mencionamos diversas vezes nos capítulos anteriores, o funcionamento básico de um computador está totalmente relacionado às operações primitivas que a UCP (o hardware) é capaz de realizar diretamente. Sabemos também que as referidas operações primitivas, básicas, têm sua execução efetivada através da realização, passo a passo, de uma seqüência de ações menores (denominadas microoperações). Esta seqüência constitui o algoritmo que conduz à execução da operação, o qual chamamos de instrução da máquina para realizar a dita operação.

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

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

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

Como visto anteriormente as Instruções de Máquina dividem-se em:

- ✔ **Código de Operação** – indica o tipo de operação a ser realizada
- ✔ **Campos Operando** – endereços dos dados

Existem vários formatos de instruções e cada uma delas com vantagens e desvantagens. Na verdade as CPU utilizam-se de uma mistura delas para permitir sua melhor utilização. A quantidade de instruções e seus formatos definidos para os processadores, além da tecnologia de materiais e arquitetura, resumem na estratégia de fabricação que diferencia cada processador de mercado, de acordo com seus fabricantes.

O problema é que compiladores (iremos ver esse assunto no próximo módulo) têm dificuldade em escolher a melhor opção de formatos de instruções para cada tipo de aplicação.

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

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

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

É conveniente repetir, neste ponto, alguns dos aspectos básicos sobre o funcionamento de um computador, especificamente referentes aos processadores (Unidade Central de Processamento), todos eles já apresentados no **Módulo V.**

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

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

Pode-se analisar a representação das instruções sob dois aspectos:

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

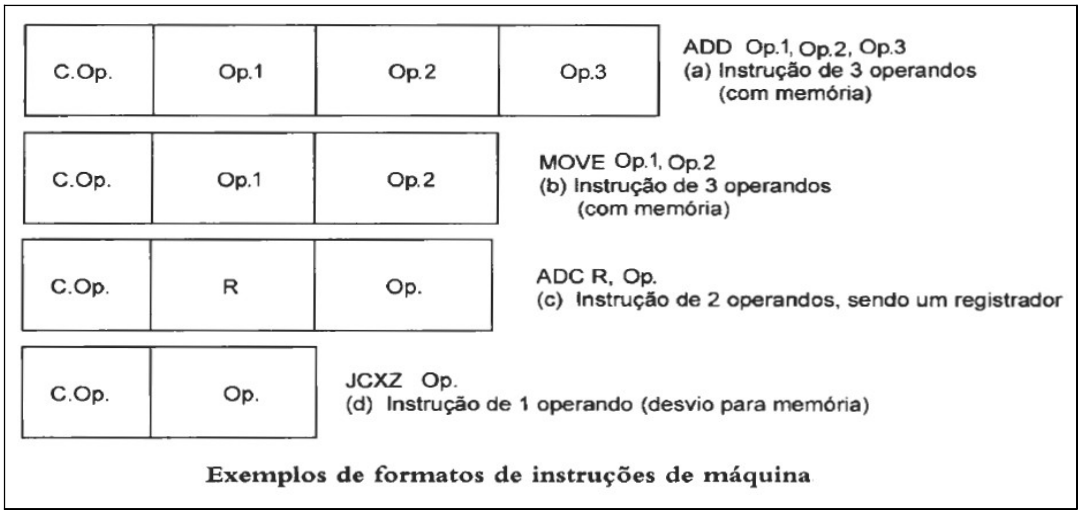
Toda instrução de máquina possui um código de operação específico e único para aquela tarefa. Este código, após ser decodificado durante o ciclo de execução da instrução, permitirá que a Unidade de Controle emita os sinais necessários, e previamente programados, para se efetivar a seqüência de passos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

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

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

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

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



Exemplos de formatos de instruções de máquina

No Projeto de Processadores, a definição do tamanho das Instruções de Máquina depende:

- ✔ do tamanho da memória
- ✔ do tamanho e organização das células da memória
- ✔ da velocidade de acesso da memória
- ✔ da organização do barramento de dados

5.2 Quantidade de Operandos

Um dos primeiros formatos de instrução possuía quatro operandos, sendo o último o endereço da próxima instrução (não é mais utilizada). Era uma instrução completa.

Neste formato poder-se-ia representar a seguinte instrução ASSEMBLER:

ADD X, Y, Z, P onde,

(Z) ← (X) + (Y), sendo P o endereço da próxima instrução. Isso poder-se-ia executar a expressão: C=A+B. Em Assembly: **ADD A,B,C,P**

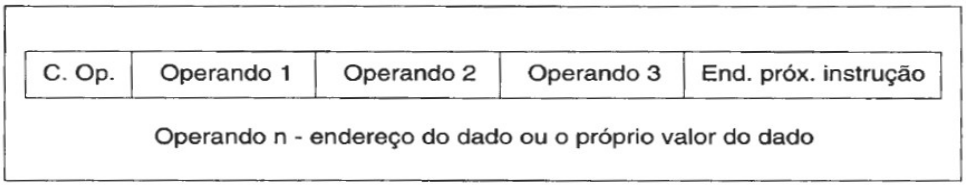
Desde os primeiros processadores concebidos até os atuais Pentium, Power PC, K7, Alpha e outros, os projetistas têm definido conjuntos de instruções dos mais variados tipos e formatos, de modo que, mesmo em um único e específico processador, as instruções tendem a ter formato diferente, isto é, tamanhos e campos diversos, conforme a operação que a instrução indica.

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

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

Tal instrução (que não é mais utilizada — nem a máquina) era completa. Não só possuía indicação explícita da localização de todos os operandos (no caso, é claro, de se tratar de uma instrução que realiza uma operação aritmética), como também já trazia armazenado o endereço da próxima instrução.

No caso de um computador com memória de 2K células (endereços) e com instrução possuindo um código de operação de 6 bits (conjunto de 64 possíveis instruções), cada instrução teria um tamanho total de 50 bits.



Exemplo de formato de instrução de quatro operandos

Vantagens desse formato de instrução:

Completeza: possui todos os valores para operação sem necessidade de instrução de desvio incondicional pois o campo P consta seu endereço

Menor quantidade de instruções em um programa: pelo simples argumento apresentado acima.

Desvantagem:

Uma grande desvantagem é a ocupação demasiada de espaço de memória. Muitas instruções não precisam de 3 operandos de dados.

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

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

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

Ponto crucial ou dilema: Economia de espaço x Conjunto completo e poderoso de instruções.

Quanto mais instruções maior será o total de bits necessários para o Código de Operação aumentando o tamanho da instrução, isso requer do decodificador maior tarefa de operação. Por serem completas requerem mais operandos aumentando o total de bits da instrução e consequentemente consomem mais memória.

Neste contexto temos as 2 arquiteturas de processadores: **RISC** e **CISC**.

Nota-se que deve-se buscar economia de memória, por ser cara, e assim usar menos operandos nas instruções, ou reduzir o tamanho total das instruções como ocorre com a arquitetura RISC.

Fruto dessa necessidade criou-se o componente CI-PC do processador para retirar esse encargo da Instrução de Máquina, reduzindo seu tamanho.

Exemplo:

Execução da Equação: X=A*(B+C*D-E/F)

Quantidade Operandos	Tamanho do C. Op.	Tamanho do Operando	Total da Instrução	Ciclos de Memória
3	8 bits	20 bits cada	68 bits	4
2	8 bits	20 bits cada	48 bits	4
1	8 bits	20 bits	28 bits	2

(Tamanho e consumo – 3, 2 e 1 Operando)

Programa Assembly para: X=A*(B+C*D-E/F)

Instrução com 3 Operandos	Instrução com 2 Operandos (sem salvamento)	Instrução com 2 Operando (com salvamento)	Instrução com 1 Operando

MPY C, D, T1
DIV E, F, T2
ADD B, T1, X
SUB X, T2,X
MPY A, X, X
	MPY C, D
	DIV E, F
	ADD B, C
	SUB B, E
	MPY X, A
	MOVE X, A
		MOVE X, C
		MPY X, D
		MOVE T1, E
		DIV T1, F
		ADD X, B
		SUB X, T1
		MPY X, A
			LDA C
			MPY D
			STA X
			LDA E
			DIV F
			STA T1
			LDA B
			ADD X
			SUB T1
			MPY A
			STA X

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

5.2.1 Instruções com 3 campos operandos

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

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

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

ADD A,B,X (X) ← (A) + (B)

SUB A,B,X (X) ← (A) − (B)

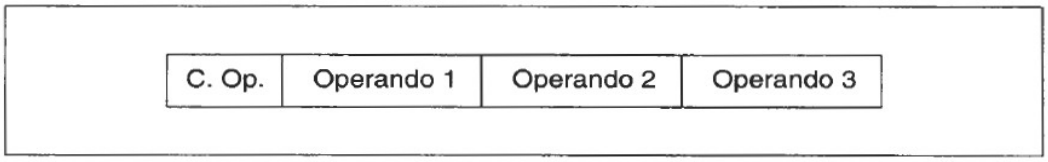
MPY A,B,X (X) ← (A) (B)

DIV A,B,X (X) ← (A) (B)

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

X = A * (B + C * D − E/F)

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



Exemplo de formato de instrução de três operandos

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

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

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

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

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

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

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

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

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

5.2.2 Instruções com 2 campos operandos

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

ADD A,B (A) ← (A) + (B)

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

ADD	Op.1,Op.2	(Op.1) ← (Op.1) + (Op.2)
SUB	Op.1,Op.2	(Op.1) ← (Op.1) − (Op.2)
MPY	Op.1,Op.2	(Op.1) ← (Op.1) * (Op.2)
DIV	Op.1,Op.2	(Op.1) ← (Op.1) /(Op.2)

5.2.3 vInstruções com 1 campo operando

Considerando as vantagens obtidas com a redução da quantidade de operandos (instruções menores), foram também criadas instruções de apenas um operando (instruções do tipo usado no Cap. 5).

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

ADD Op.	ACC ← ACC + (Op.)
SUB Op.	ACC ← ACC − (Op.)
MPY Op.	ACC ← ACC (Op.)
DIV Op.	ACC ← ACC (Op.)

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

LDA Op.	que significa	ACC ← (Op.)
STA Op.		(Op.) ← ACC

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

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

MOVE A,B (A) ← (B)

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

Com essas instruções de dois operandos, o comando correspondente à **Equação** poderia ser convertido, para execução, no programa Assembler apresentado a seguir, ainda de acordo com a seqüência apresentada no item anterior:

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

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

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

Assim, a execução da instrução

```
MPY     C,D
```

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

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

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

Considerações:

Há muita controvérsia em implementação do conjunto de instruções em projetos de processadores. Não há unanimidade na definição do tamanho, formato e significado do campo operando das instruções.

Instruções com poucos operandos ocupam menos memória e tornam-se mais simples, porém aumentam a quantidade de instruções (programa em binário).

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

Instruções de poucos operandos ocupam menos espaço de memória e tornam o projeto do processador mais simples em virtude das poucas ações que elas induzem a realizar. No entanto, o programa gerado em binário é algumas vezes maior devido ao aumento da quantidade de instruções (não é o caso do exemplo apresentado na **Tabela apresentada com os 3 modos de representação das Instruções**).

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

Instruções com mais de 1 operando podem usar tanto endereços de memória como registradores em seu formato, outro item de discussão.

5.3 Modos de Endereçamento

A execução do ciclo de instrução resulta em:

- ✔ O endereço da próxima instrução está no CI-PC
- ✔ O início do ciclo resume em transferir a instrução corrente para o RI

Toda instrução possui um Código de Operação (C.Op.).

A localização dos dados (endereço) para uso pela instrução pode estar nos campos operandos ou no próprio ACC (valor efetivo). Essa operação de localização dos dados resume o modo de endereçamento – existem vários modos de se obtê-lo.

Quanto ao formato básico das instruções de máquina e o ciclo de instrução, conclui-se:

a) O endereçamento de uma instrução é sempre realizado através do valor armazenado no contador de instrução (CI). Todo ciclo de instrução é iniciado pela transferência da instrução para o RI (usando-se o endereço contido no CI).

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

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

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

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

Principais Modos de Endereçamento:

- ✔ Imediato
- ✔ Direto
- ✔ Indireto
- ✔ Por registrador
- ✔ Indexado
- ✔ Base mais deslocamento

1) Modo Imediato

Definição:

O dado é transferido da memória junto com a instrução. O valor do dado está no campo operando.

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

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

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

Uma de suas desvantagens consiste na limitação do tamanho do campo operando das instruções, o que reduz o valor máximo do dado a ser manipulado.

Outra desvantagem é o fato de que, em programas repetidamente executados, com valores de variáveis diferentes a cada execução, esse método acarretaria o trabalho de alteração do valor do campo operando a cada execução (dado de valor diferente).

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

Exemplo 1

Seja a operação **JMP Op.** que é definida para carregar o CI com o valor existente no campo Operando, com a seguinte notação: **CI ← Op.**

Seja o **C. Op. = 10** para o valor correspondente à operação em questão.

Sabemos que 10 = A, na base 16, e que equivale a operação JMP, como dito acima.

Seja ainda o **dado = 00110101** = 35, na base 16.

Logo a Instrução de Máquina, nesse formato de apenas um campo operando, será escrita assim:

A35, na base 16 ou **101000110101** na base 2 (observe a quantidade de bits para C. Op e Operando.)

Como resultado da execução dessa operação teremos o CI carregado com o valor igual a **35**, na base 16 ou **00110101**, na base 2.

Exemplo 2:

C. Op.	R	Operando	MOV R, Op. R ← Op. C. Op. = 0101 = hexadecimal 5
4 bits	4 bits	8 bits	

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

Vantagens desse modo

- ✔ Método mais rápido para execução da instrução
- ✔ Método mais simples e rápido de obter o dado
- ✔ Como o valor do dado já vem na instrução, isso reduz o tempo de execução da instrução (menor quantidade de acessos à memória)

Desvantagens desse modo

- ✔ Consiste na limitação do valor máximo do dado a ser manipulado (restrito ao tamanho do operando)
- ✔ Em programas repetidamente executados, onde possuem valores de variáveis diferentes, acarretam no trabalho de alteração do valor do campo operando a cada execução

Aplicação

Útil para inicialização de contadores (constantes matemáticas; indicação de ponteiros em registradores; operação de deslocamento esquerdo ou direito; etc)

Praticamente todo computador possui uma ou mais instruções que empregam esse modo de endereçamento para:

- ✔ **Instruções de desvio;**
- ✔ **Movimentação de dado**
- ✔ **Operações aritméticas com uma constante**

2) Modo Direto

Definição:

O campo operando da instrução indica o endereço de memória onde se localiza o dado (já estudado antes):

- ✔ **Pode ser o de uma célula (dado está inteiramente contido na célula);**
- ✔ **Pode indicar o endereço da célula inicial (dado em múltiplas células)**

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

O endereço pode ser o de uma célula onde o dado está inteiramente contido ou pode indicar o endereço da célula inicial, quando o dado está armazenado em múltiplas células.

É também um modo simples de acesso, pois requer apenas uma referência à MP para buscar o dado, sendo, porém, mais lento que o modo imediato, devido naturalmente à referência à memória.

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

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

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

Exemplo:

C. Op. 4 bits	Operando 8 bits

M P	
3A	9F
3B	5A
3C	FD

C. Op. = **7** na base 16 => LDA Op. executa: ACC ← Op.

Operando = **3B** na base 16

Instrução: **73B** (para instruções com formato de apenas um campo operando)

ACC ← (3B) (Acesso à memória para obter o valor do dado)

ACC ← 5A

Vantagens

Quando o dado varia de valor a cada execução do programa, a melhor maneira é armazená-lo na memória. O programa usa o dado pelo modo direto.

É também um modo simples de acesso pois requer apenas uma referência à memória para buscar o dado

Desvantagens

É mais lento que o Modo Imediato pelo fato de fazer referência à memória.

Limitação de memória a ser usada, conforme o tamanho do campo operando.

Ex: operando com 12 bits, só podem acessar células com endereços de 0 a 4095 (2^12).

As memórias atualmente possuem espaços de endereçamento bem maiores (64 a 256 MB). Não é desejável criar operando de tantos bits. Para endereçar 64 MB seriam necessários 26 bits para endereço direto.

3) Modo Indireto

Definição:

O valor binário do campo operando representa o endereço de uma célula, mas o conteúdo dessa célula não representa o valor do dado (como no modo direto), mas um outro endereço de memória, cujo conteúdo é o valor do dado.

Nesse método, o valor binário do campo operando representa o endereço de uma célula; mas o conteúdo da referida célula não é o valor de um dado (como no modo direto), é um outro endereço de memória, cujo conteúdo é o valor do dado.

Assim, há um duplo endereçamento para o acesso a um dado e, conseqüentemente, mais ciclos de memória para buscar o dado, comparativamente com os métodos já apresentados.

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

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

Exemplo:

C. Op. 4 bits	Operando 8 bits

M P	
3A	3C
3B	5A
3C	FD

Seja o C. Op. = **7** (base 16) para indicar a operação: LDA Op. que sua execução resulta em: ACC ← ((Op.)) - carregar o acumulador com o valor.

Obs: nesta notação, cada parêntesis representa o endereço da memória, logo aqui teremos o endereço do endereço da memória – conforme preconiza o modo de endereçamento indireto.

Seja o valor do Operando = **3A** na base 16

Logo a instrução para esse formato será: **73A**

ACC ← ((3A)) (Acesso à MP para obter o valor do endereço do endereço do dado)

ACC ← (3C) (Acesso à MP para obter o valor do endereço do dado)

ACC ← FD (Acesso à MP para obter o valor do dado)

Assim o ACC será carregado com o valor: **FD**

Vantagens

Esse processo elimina o problema do endereçamento direto (limitação do valor do endereço do dado), pois estando o endereço armazenado na memória, (pode ocupar uma ou mais células), ele se estenderá ao tamanho necessário à representação do maior endereço da memória.

Facilidade no manuseio de vetores (quando o modo indexado não está disponível).

Flexibilidade no uso como “ponteiro”.

Desvantagens

Como há um duplo endereçamento, mais ciclos de memória são necessários para buscar o dado, portanto mais lento que os dois modos anteriores.

Apesar de poder acessar qualquer endereço da memória, limita-se também ao tamanho do campo operando.

Aplicação

Este modo é conhecido como *ponteiro* (era muito usado em programação), “aponta” para o dado.

É pouco empregado atualmente face à quantidade de acesso à memória e complexidade para os programadores em Assembly.

Pausa!

Considerações quanto aos três modos de endereçamento já apresentados:

Observações

1) Há dois métodos de indicação do modo de endereçamento de instruções:
   a) Cada código de operação estabelece não só o tipo da instrução como também o modo de endereçamento.
   b) A instrução possui um campo específico para indicar o modo de endereçamento; nesse campo, consta um código binário correspondente ao modo desejado.
   todo.
2) Comparando-se as características dos três modos de endereçamento apresentados, pode-se observar que:
   a) O modo *imediato* não requer acesso à MP para buscar o dado (exceto, é claro, para a busca da instrução); o modo *direto* requer um acesso e o modo *indireto*, pelo menos dois acessos para busca do dado na MP.
   b) Quanto ao tempo de execução das instruções, as que usam o modo *imediato* são mais rápidas, seguidas das que usam o modo *direto* e, finalmente, as que usam o modo *indireto* são executadas mais lentamente. A velocidade de execução é diretamente proporcional à quantidade de acessos despendida em cada ciclo da instrução.

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

Quadro demonstrativo das características dos modos de endereçamento

4) Endereçamento por Registrador

Definição

Tem característica semelhante aos modos direto e indireto, exceto que a célula (ou palavra) de memória referenciada na instrução é substituída por um dos registradores da CPU.

Com isso o endereço mencionado na instrução passa a ser o de um dos registradores, e não mais de uma célula da memória.

Exemplo:

C. Op. 4 bits	R 1 bit	Operando 8 bits

Registradores	
1	3C
2	5A
3	FD

Seja o C. Op. = **9** (base 16) indicando a seguinte operação: **ADR R,Op.** e que sua execução resume em: **(R) ← (R) + (Op.)** - carregar no registrador a soma do conteúdo existente no registrador e o valor do campo operando da instrução.

Seja o valor do Operando = **05** (base 16)

Então a Instrução de Máquina, neste formato de instrução será: **9R2,05**

(R2) ← (R2) + 05

(R2) ← 5A + 05

(R2) ← 5F

Como resultado desta operação teremos carregado no registrador 2 o valor **2F**.

Vantagens

Menor número de bits necessários para endereçar os registradores (menor quantidade que as células de memória). Isto reduz o tamanho geral da instrução.

Se uma CPU possui 16 registradores, para endereçá-los bastar apenas 4 bits (de 0 a F) na base 16. Para o caso de células de memória haveria necessidade de 20 ou mais bits.

O dado está armazenado em um meio (registrador) cujo acesso é muito mais rápido que o acesso à memória, uma vez que ele já se encontra dentro do próprio processador.

Desvantagens

Uso limitado em face de haver poucos registradores na CPU.

As vantagens apresentadas anteriormente nem sempre são aplicáveis. Há casos em que seu uso poderá representar desperdício de instruções.

Na operação **X = A + B** (uso do modo direto).

Não é eficaz fazer a transferência MP → Registrador → ULA. Se fosse usado o modo registrador ele só serviria para atrasar a execução da instrução já que os valores A e B são lidos da memória e repassados à ULA (código completo adiante).

Aplicação

Trechos de programas que são repetitivos e que necessitam do uso de contadores torna seu uso eficiente.

Exemplo:

Imagine o trecho de programa com uso de contador, a seguir:

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

Podem ser usados nos **modos**:

- ✔ Por **registrador direto** (o valor do registrador é o dado)
- ✔ Por **registrador indireto** (o valor do registrador é o endereço da memória)

5) Modo Indexado

Definição:

O endereço do dado é a soma do valor do campo operando (fixo para todos elementos de um vetor ou array) e de um valor armazenado em um registrador da CPU (**registrador índice**). O valor do registador varia para o acesso a cada elemento (“aponta” para o elemento desejado do vetor).

Adota-se o princípio que os elementos de um array são armazenados de forma sequencial na memória e a localização de um certo valor pode ser referenciada por um ponteiro.

Essa denominação advém do fato de que a obtenção do endereço de um dado (do array) relaciona-se com seu índice.

A escolha do registrador a ser utilizado como registrador-índice depende da linguagem a ser empregada:

- ✔ **Para o Assembly é de responsabilidade do programador**
- ✔ **Para linguagens de alto nível fica a cargo do compilador**

Para uso deste modo de endereçamento é necessário haver instruções que manipulem valores em registradores como:

- ✔ Carregar o valor no registrador (índice)
- ✔ Somar valor ao do registrador (incremento)
- ✔ Desviar para outra instrução quando o valor do registrador é zero

Exemplo:

C. Op. 4 bits	Registrador-Índice 1 bit	Operando 8 bits
	(valor variável)	(valor fixo)

Registradores	
1	00
2	5A
3	FD

Seguindo-se o mesmo raciocínio, seja o valor do **C. Op. = 7** (base 16) como sendo o que indica a operação: **LDA R,Op.** que sua execução resume-se em: **(R) ← (R)+(Op.)**

E seja o valor do Operando = **18AC** (base 16)

Assim, a Instrução de Máquina, para este formato de um operando será: **7R2,18AC**

Representando a execução desta instrução teremos:

(R2) ← (R2) + 18AC, que resulta em **(R2) ← 00+18AC**

INC (R2), logo **R2 ← R2 + 1**, que resulta **(R2) ← 00+01**

Ao ser executado a instrução **C. Op. = 7, R2** valerá **18AD**

Vantagens

Para operações com array necessita-se menor quantidade de instruções e para essa operações agilizaria a execução do trecho do programa.

Rapidez na execução das instruções de acesso a dados uma vez que a alteração do endereço dos elementos é realizada na própria CPU.

Desvantagem

Usado apenas para determinados tipos de dados (limitação) com ganho substancial de rapidez. Em outros caso torna-se ineficiente.

Aplicação

Durante a execução de programas, há a necessidade de se manipular endereços de acesso a elementos de certos tipos especiais de dados, como os vetores ou arrays.

Muitos computadores modernos possuem instruções no modo indexado para esse fim.

É empregada quando se deseja acessar diferentes dados, com alteração do endereço, por incremento do valor do registrador-índice, ou seja: Vários dados são acessados com diversos valores de registrador-índice e um único valor no campo operando.

6) Modo Base Mais Deslocamento

Definição:

Tem características semelhantes ao modo indexado, pois o endereço de acesso a uma célula resulta em soma de valores. Um valor é inserido num campo apropriado da instrução (**campo deslocamento**) e o outro inserido em um registrador denominado **registrador-base** ou **registrador de segmento**.

Sua implementação resume-se no fato de que o valor fixo é o do registrador-base, diferentemente do modo indexado, e variando o valor do campo deslocamento em cada instrução.

Exemplo:

C. Op. 4 bits	Registrador-Base 1 bit	Campo Deslocamento 8 bits
	(valor fixo)	(valor variável)

Registradores	
1	00
2	5A
3	FD

Operação inversa da realizada no Modo Indexado.

Vantagens

Redução do tamanho das instruções (economia de memória).

Facilita o processo de relocação dinâmica de programas, isto é:

- ✔ Na execução de grande quantidade de programas, as referências às células de memória, onde se localizam os operandos, normalmente são sequenciais, ocorrendo poucos acessos a outras instruções fora de ordem (exceto desvios).
- ✔ A maioria dos programas ocupa um pequeno espaço da MP disponível.
- ✔ Assim, basta que o endereço desejado seja obtido com a soma do valor existente no registrador e um valor contido na instrução

Aplicação

Usado quando a modificação de endereço é realizada para relocação de programa, isso se dá com uma única alteração do conteúdo do registrador-base, ou seja:

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

Considerações Finais

É importante conhecermos todos os modos de endereçamento, mas particularmente sabermos aplicar, pelo menos os três primeiros, isto é: Imediato, direto e indireto, inclusive fazermos operações com estes modos.

Entre no Fórum da disciplina para fazermos uma prática com estes 3 modos e entendermos as suas diferenças, vantagens e desvantagens.

Nos encontraremos lá!

Fim