Computador HIPO

Para introduzirmos as noções básicas de como funciona um computador, empregaremos um modelo imaginário (hipotético) que denominaremos de *computador hipo*. O funcionamento desse modelo tem as características básicas do funcionamento de um computador real embora seja muito mais simples e didático.

1 Introdução

Inicialmente vamos apresentar as unidades fundamentais de um computador:

- *Unidade Central de Processamento* ou CPU: é a unidade que controla o funcionamento do computador e executa as *instruções*. Ela contém o *apontador de instruções* e o *acumulador*, que serão explicados adiante.
- *Memória Central*: é composta por unidades chamadas *posições de memória*. Nos computadores cada posição de memória é indicada por um *endereço*. No HIPO (daqui em diante abreviaremos "Computador Hipo" por Hipo) esse endereço é um número de 00 a 99 (dois algarismos). Cada posição de memória pode conter um número. Esse número é representado por um sinal ($+$ ou $-$) seguido de 4 algarismos decimais (por exemplo, $+0002, +1992, -1991$). Doravante denominaremos um sinal e 4 algarismos decimais de *inteiro*.
- *Acumulador*: é uma memória (chamada "registrador") particular da CPU. Como qualquer outra posicao de memória, o acumulador pode conter um *inteiro*.
- *Instrução*: é uma "operação básica" que o computador é capaz de executar. Uma instrução descreve uma ação a ser executada pela CPU. Cada instrução é *codificada* de forma a ser representada por um *inteiro*.
- *Apontador de Instrução*: é utilizado para indicar à CPU qual o endereço da próxima instrução a ser executada.
- *Teclado*: é a unidade que recebe dados do exterior do computador e os transmite à CPU.

- *Impressora*: como o nome diz, é a unidade que imprime dados no papel; esses dados são fornecidos pela CPU.
- *Programa*: é uma seqüência de instruções armazenada na memória.

2 Funcionamento do HIPO

Para descrever o funcionamento do HIPO, vamos dar as regras básicas de funcionamento do componente principal, a CPU. As regras são:

1. Ler a instrução que está na posição de memória indicada pelo apontador de instrução.
2. Mudar o apontador de instrução para a posição de memória seguinte.
3. Interpretar e executar a instrução lida.

Dadas as regras básicas, precisamos *carregar um programa* na memória e fazer com que o apontador de instruções aponte para a primeira instrução do programa. Observação: "carregar" um programa significa colocar cada instrução do mesmo em uma posição de memória. Feito isso, podemos acionar a CPU fazendo com que ela siga as regras básicas e execute o programa dado.

Pergunta: o que aconteceria se trocássemos as regras 2 e 3 de lugar?

3 Codificação de Instruções

Infelizmente o HIPO não reconhece frases do tipo "copie o conteúdo do Acumulador para o Endereço 30". Por isso precisamos escrever numa linguagem que este possa entender, i.e. em *linguagem de máquina*. Para este computador, uma instrução será codificada por um *inteiro*. Por convenção todas as instruções serão codificadas por *inteiros* positivos (i.e. com sinal +).

3.1 Como fazer a codificação das instruções

Suponha que temos que passar para a "linguagem do HIPO" a instrução "Copie o conteúdo do Acumulador para o Endereço 30":

Em primeiro lugar, utilizamos o sinal +. Os dois primeiros algarismos correspondem à instrução, ou seja, à operação a ser executada pela CPU. Na secção 4 listamos os códigos de todas as instruções do HIPO. O código da instrução "copie o conteúdo do acumulador no endereço EE" é o número 12.

Os dois últimos algarismos do código correspondem ao endereço de uma posição de memória, que neste caso é o 30. Logo a instrução acima é codificada, em linguagem de máquina, como

$$+1230$$

Vamos a um outro exemplo: A instrucao é "Se o conteúdo do Acumulador for menor que zero, desvie para o endereço 11".

- Sinal + (por convenção)
- Os dois primeiros algarismos correspondem a essa instrução de desvio: 54 (confira na tabela)
- Os dois últimos algarismos indicam o endereço referenciado: 11

Portanto a codificação desta instrução na linguagem HIPO é +5411.

exercícios

1. Quais são as vantagens a as desvantagens de se codificar as instruções desta maneira.
2. Qual é o número máximo de instruções que cabem na memória do computador?
3. É possível diferenciar uma instrução de um dado? Por quê?

4 Instruções do computador hipo

Cada instrução é composta de um código mais um endereço relativo a esse código, da seguinte maneira: **+IIEE** Onde **II** é o código de uma das instruções abaixo, e **EE** é o endereço ao qual ela se refere ($00 \leq EE \leq 99$). A seguir, [EE] indica o conteúdo de EE, [AC] indica o conteúdo do acumulador, AI indica o apontador de instrucao e (XXX) indica o código simbólico abreviado.

11: (CEA) Copie o conteúdo do endereço EE no acumulador. (AC recebe [EE]).

12: (CAE) Copie o conteúdo do acumulador no endereço EE. (EE recebe [AC])

21: (SOM) Some o conteúdo do endereço EE com o conteúdo do acumulador e guarde o resultado no acumulador. (AC recebe [AC] + [EE])

22: (SUB) Subtraia o conteúdo do endereço EE do conteúdo do acumulador e guarde o resultado no acumulador. (AC recebe [AC] - [EE])

23: (MUL) Multiplique o conteúdo do endereço EE com o conteúdo do acumulador e guarde o resultado no acumulador. (AC recebe [AC] * [EE])

24: (DIV) Divide o conteúdo do acumulador pelo conteúdo do endereço EE e guarde o resultado no acumulador. (AC recebe [AC] / [EE])

25: (MOD) [AC] recebe o resto da divisão [AC] / [EE].

31: (LER) Leia um número e guarde-o no endereço EE. (EE recebe o valor lido)

41: (IMP) Imprima o conteúdo do endereço EE.

50: (NOP) Nenhuma operação é efetuada.

51: (DES) Desvie a execução para o endereço EE, i.e. AI recebe EE.

52: (DPO) Se o conteúdo do acumulador for maior do que zero, desvie a execução para o endereço EE. (Se $[AC] > 0$, AI recebe EE).

53: (DPZ) Se o conteúdo do acumulador for maior ou igual a zero, desvie a execução para o endereço EE. (Se $[AC] \geq 0$, AI recebe EE).

54: (DNE) Se o conteúdo do acumulador for menor do que zero, desvie a execução para o endereço EE. (Se $[AC] < 0$, AI recebe EE.)

55: (DNZ) Se o conteúdo do acumulador for menor ou igual a zero, desvie a execução para o endereço EE. (Se $[AC] \leq 0$, AI recebe EE).

56: (DDZ) Se o conteúdo do acumulador for diferente de zero, desvie a execução para o endereço EE. (Se $[AC] \neq 0$, AI recebe EE).

57: (DZZ) Se o conteúdo do acumulador for igual a zero, desvie a execução para o endereço EE. (Se $[AC] = 0$, AI recebe EE).

58: (DDF) Se o conteudo do acumulador for diferente de infinito, desvie a execução para o endereço EE. (Se $[AC] \neq$ INF, AI recebe EE).

59: (DFF) Se o conteudo do acumulador for infinito, desvie a execução para o endereço EE. (Se $[AC] =$ INF, AI recebe EE).

61: (ADE) Desloque os digitos do acumulador uma posição à esquerda, desprezando o digito mais significativo.

62: (ADD) Desloque os digitos do acumulador uma posição à direita, desprezando o digito menos significativo.

70: (PAR) Pare a execução do programa. OBS.: Esta instrução deve ser executada para encerrar a execução do programa.

5 PROBLEMA 1

Dada uma seqüência de números inteiros não negativos, imprimir a sua soma. A seqüência é seguida de um número negativo. Exemplo de sequência de entrada: +0100, + 0015, +0000, +0007, -0002.

Programa 1:

Instruções por extenso	Endereço	Linguagem de máquina do hipo	Linguagem simbólica (de montagem)
Copie o conteúdo do endereço 30 no acumulador	01	+1130	CEA zero
Copie o conteúdo do acumulador no endereço 40	02	+1240	CAE soma
Leia um número e coloque no endereço 45	03	+3145	**leia:** LER num
Imprima o conteúdo do endereço 45	04	+4145	IMP num
Copie o conteúdo do endereço 45 no acumulador	05	+1145	CEA num
Se o conteúdo do acumulador for menor que zero, desvie para o endereço 11	06	+5411	DNE fim
Copie o conteúdo do endereço 40 no acumulador	07	+1140	CEA soma
Some o conteúdo do endereço 45 com o conteúdo do acumulador e guarde no acumulador	08	+2145	SOM num
Copie o conteúdo do acumulador no endereço 40	09	+1240	CAE soma
Desvie para o endereço 03	10	+5103	DES leia
Imprima o conteúdo do endereço 40	11	+4140	**fim:** IMP soma
Pare	12	+7000	PAR
	30	+0000	**zero:**
	40		**soma:**
	45		**num:**

A quarta coluna da tabela acima representa o programa escrito em uma *linguagem de montagem* ou *assemby*. Escrever um programa em *assembly* é bem mais fácil que escrevê-lo diretamente em linguagem de máquina. Poderíamos inclusive escrever um *programa montador*, ou *assembler*, que se encarregue de traduzir *assembly*, para linguagem de máquina (i.e. um programa que gere o conteúdo da terceira coluna a partir do conteúdo da segunda e da quarta colunas).

6 Fluxograma

Damos a seguir uma outra representação simbólica do programa 1, a de diagrama de blocos, ou fluxograma. Seu significado deve ser evidente.

End.	Linguagem de máquina	rótulos	Linguagem simbólica	Fluxograma
01	+1130		CEA zero	Soma $\leftarrow$ 0
02	+1240		CAE soma	
03	+3145	**leia**	LER num	Ler num
04	+4145		IMP num	Imprimir [num]
05	+1145		CEA num	
06	+5411		DNE fim	Se [num] $<$ 0
07	+1140		CEA soma	
08	+2145		SOM num	
09	+1240		CAE soma	soma $\leftarrow$ [soma]+[num]
10	+5103		DES leia	
11	+4140	**fim**	IMP soma	Imprimir [soma]
12	+7000		PAR	Pare
30	+0000	**zero**		
40		**soma**		
45		**num**		

7 Representações Internas

7.1 Letras

Apesar dos computadores só lidarem com números, eles podem armazenar informações que, a princípio, não são números. Por exemplo, como representaríamos a letra "A" no computador ? Como guardaríamos um nome (por ex. "CARLOS DA SILVA")? Podemos representar objetos como números, desde que tenhamos uma tabela de tradução que diga qual número é associado a qual objeto. Vamos ver como representar caracteres na memória do computador :

instalamos no computador uma tabela assim : A = +0001, B = +0002,..., Z = +0026, 0 = +0027, 1 = +0028, ... , 9 = +0036, a = +0037, b = +0038, ... Daí, podemos "traduzir" qualquer caractere para um número e colocá-lo na memória do computador.

Como faremos para representar seqüências de caracteres? A maneira mais simples é colocar cada letra em uma posição da memória em seqüência, por exemplo : "CARLOS" ficaria assim :
C,A,R,L,O,S = +0003, +0001, +0018, +0012, +0015, +0019

7.2 Overflow

Vamos agora discutir alguns detalhes adicionais sobre a representação de números inteiros.

Em primeiro lugar notemos que há duas maneiras possíveis de representar o número zero, a saber: +0000 e -0000. Por convenção, fica estabelecido que zero é representado por +0000.

Em segundo lugar falta estabelecer o resultado de uma operação aritmética entre dois *inteiros* cujo resultado não pode ser expresso como um *inteiro*. Por exemplo se somarmos +9991 e +0010, o "resultado" (10001) não é mais representável como um *inteiro* (pois *inteiros* sendo constituídos de um sinal e 4 algarismos decimais só podem representar números inteiros entre -9999 e +9999). Casos deste tipo são chamados de *overflow* (transbordamento). Neste caso convencionamos que a CPU retorna o código -0000, que fica sendo nossa representação de infinito (INF), ou de qualquer número inteiro menor que -9999 ou maior que +9999. Também por convenção, o resultado de qualquer operação aritmética envolvendo INF tem como resultado INF; e para as operações de desvio, INF (-0000) não é igual, nem maior, nem menor que zero (+0000).

exemplos:

(-9920 - 80) + 5000 = INF + 5000 = INF.
(-9920 + 5000) - 80 = -4920 - 80 = -5000.

7.3 HIPO-II

Ao multiplicarmos dois inteiros de 4 digitos (não nulos), teremos sempre overflow! Isto motiva um modelo melhorado do HIPO, o HIPO-II. No HIPO-II o acumulador tem "precisão dupla", i.e. o acumumulador pode conter um número representado por um sinal (+ ou −) seguidos de 8 algarismos decimais (por exemplo, $+00000002, +12345678, -12345678$). Doravante denominaremos um sinal e 8 algarismos decimais de *inteiro longo*. Repare que qualquer produto de 2 inteiros pode ser representado como um inteiro longo.

Ao Copiarmos o conteúdo do endereço EE no acumulador (instrução CEA), o sinal e os 4 digitos menos significativos (à direita) de AC recebem [EE]. Assim o inteiro +1998 será copiado para o acumulador como o inteiro longo +00001998.

Ao Copiarmos o conteúdo do acumulador no endereço EE (instrução CAE), [EE] recebe o sinal e os 4 digitos menos significativos (à direita) de AC. Assim o inteiro logo +12345678 em AC será copiado para a posição de memória EE como o inteiro +5678 .

7.4 Ponto Flutuante

O HIPO só guarda números inteiros na memória, o que faríamos para guardar números fracionários? Lembrem-se que um número no padrão HIPO, um *inteiro*, é um sinal seguido de quatro algarismos decimais, como +1234, -3141, ou generalizando, ±DDDD

Agora, nada nos impede de imaginar um ponto no meio deste número, por exemplo assim: ±DD.DD Isso corresponderia a olharmos +1234 na memória como o número 12.34. Assim podemos representar números de -99.99 até +99.99 na memória do HIPO. Nada nos impede de imaginar este ponto em posições diferentes: ±DDD.D ou ±D.DDD , etc. Este método só é bom quando os números têm o mesmo "tamanho", ou seja, estão mais ou menos próximos. Porém, não poderíamos fazer a conta $1000 \times 2.2 = 2200$ usando a representação de ponto fixo no HIPO (pois o ponto precisaria estar num lugar diferente em cada número). Esta conta seria possível se tivéssemos um ponto móvel ("ponto flutuante" em gíria de computação).

Uma maneira (existem várias outras) de representar um número com ponto flutuante no HIPO é colocar em uma posição de memória a mantissa (os algarismos do número), e noutra colocar o expoente (a posição do ponto). Um número com ponto flutuante no formato HIPO será doravante denominado *flutuante*.

Para colocar um número no formato *flutuante* primeiro o escrevemos em notação científica: $230 = 2.30 \times 10^2 = 2.3E + 2$. Depois codificamos o número no formato: ±MMMM ±EEEE onde ±MMMM é a mantissa; no nosso exemplo: +2300 ; e ±EEEE é o expoente, no nosso exemplo: +0002.

Note que imaginamos um ponto fixo aps o primeiro dígito da mantissa, dígito este sempre diferente de zero (±M.MMM). Outro exemplo: $0.00000032 = 3.2E - 7$. aqui a mantissa é +3200 e o expoente é -0007.

Este truque de colocarmos um número em duas posições de memória é bastante comum (normalmente os números são colocados em várias posições de memória) e não é muito difícil programarmos o computador para fazer contas corretamente com *flutuantes*. No entanto, este método de representar números tem limitações : 12345 não pode ser representado exatamente, nem $\frac{1}{3} = 0.333\ldots$, $\sqrt{2} = 1.414213562\ldots$ ou $\pi = 3.14159265\ldots$.

Uma operação aritimética (+,-,∗,/) entre dois *flutuantes* é denominada um FLOP. O numero de FLOPs que um computador consegue realizar por segundo é uma das medidas mais importantes de seu desempenho. Para aumentar seu desempenho o computador tem muitas vezes uma unidade de processamento auxiliar, sob comando da CPU, especializada em FLOPs: é a FPU. Como curiosidade, compare a FLOPagem (FLOPs/segundo) de alguns computadores no mercado:

Computador	CPU	FPU	FLOPagem
PC-XT	Intel 8086	sem FPU	$1E3$
PC-XT	Intel 8086	Intel 8087	$50E3$
PC-AT	Intel i486 com FPU incorporada		$5E6$
PC-AT	Placa com coprocessador i860		$50E6$
CRAY-1	proprietária		$50E6$
CRAY-XMP	proprietária		$200E6$
Touchstone Delta	528 CPUs i860		$10E9$

Estas medidas de desempenho são freqüentemente dadas em *FLOPs, onde * indica a inicial de: Kilo = 10^3, Mega = 10^6, Giga = 10^9 ou Tera = 10^{12}.

exercícios

1. Traduza para o formato numérico do computador HIPO: **a)** "A", "Abacaxi"; **b)** 0.123, 23.22, $1,66 * 10^{-24}$.
2. Faça um programa no HIPO-II para somar dois *flutuantes*.
3. Faça um programa no HIPO-II para multiplicar dois *flutuantes*.

8 Bases de Numeração

Considere os seguintes números: 19 e 91. Embora os dois números contenham os mesmos dígitos (1 e 9) sabemos que eles são diferentes e, além disso, que o segundo número é maior que o primeiro (pois eles têm o mesmo número de dígitos e o dígito mais à esquerda de 91 é maior que o dígito mais à esquerda de 19).

O sistema decimal é um sistema de numeração que usa a *base de numeração 10* , como o próprio nome indica. Cada numero é escrito usando-se *os dígitos 0, 1, 2, 3, 4, 5 , 6, 7, 8 e 9* e a cada algarismo deste número corresponde um valor numérico que depende da posição desse algarismo no número. Assim, por exemplo, o número 1992 é uma notação abreviada do seguinte:

$$1992 = 1000+900+90+2 = 1*1000+9*100+9*10+2 = 1*10^3+9*10^2+9*10^1+2*10^0$$

Formalizando este conceito de base de numeração, em uma base qualquer B (no caso decimal, $B = 10$), um número N é representado explicitando-se a base por

$$N = (d_m d_{m-1} ... d_1 d_0)_B$$

onde d_m,..., d_1 e d_0 são seus dígitos, representados pelos algarismos 0 a $B-1$. O valor de N é dado por

$$N = d_m * B^m + d_{m-1} * B^{m-1} + ... + d_1 * B^1 + d_0 * B^0$$

Os computadores são construídos utilizando-se, por causa de fatores tecnológicos, um sistema *binário* de representação interna de números e de caracteres. Assim, cada dígito de uma "palavra" pode conter apenas os algarismos 0 e 1. Neste sistema de numeração que usa apenas dois algarismos, estes são chamados de *bit* (do Inglês, "binary digit"). No entanto, a grande maioria dos computadores usa um agrupamento de 8 bits como unidade de informação, i.e. cada posição de memória contem 8 *bits*. Esses agrupamentos de 8 bits são chamados de *Byte*, e podem representar até $2^8 = 256$ números ou caracteres.

Observação : No caso do HIPO, convencionamos utilizar o sistema decimal para manter a simplicidade do modelo.

Exemplo: $(1101)_2$ é um número cujo valor é

$$(1101) = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 8 + 4 + 0 + 1 = 13$$

Outras bases de numeração bastante utilizadas em computação são:

- *octal*, ou base 8, que usa os dígitos $0, 1, 2, 3, 4, 5, 6, 7$.
- *hexadecimal*, ou base 16, que usa os dígitos $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, f$ onde os algarismos $a \ldots f$ valem, respectivamente, $10 \ldots 15$.

exercícios

1. Converter para a base decimal os seguintes números binários: 1111, 10101, 00101.
2. Represente em base binária os seguintes números decimais: 8, 16, 15, 20.
3. Deduza uma regra de conversão de números decimais para números binários. Sugestão: divida sucessivamente o número decimal por 2 e veja o resto de cada divisão.
4. Usando a regra deduzida no exercício anterior, converta para a base binária os seguintes números decimais: 20, 21, 73, 512, 1984. Verifique o resultado convertendo os números binários para a base 10, empregando a fórmula de potência já vista.
5. Converta para a base 10 os seguintes números: $(2101)_3$, $(1765)_8$, $(1a2f)_{16}$

Bibliografia

A idéia de definir um modelo didático de computador (como o HIPO) é bastante antiga e difundida. O exemplo mais famoso desta prática é o computador MIX apresentado em [Knuth69]. A presente apostila contém, com pequenas alterações, material de [Setzer84] e [Setzer91].

[Knuth69]. D.E.Knuth, *The Art of Computer Programming Vol.1*, Addison-Wesley Publishing Company, Readind.

[Setzer84]. W.W.Setzer e R.Terada, *Introdução à Computação e à Construção de Algorítmos.* IME-USP, São Paulo.

[Setzer90] W.W.Setzer, *Apostila do Dia da Computação.* IME-USP, São Paulo.