[MAC0211] Laboratório de Programação I
Aula 3
Sistemas de Numeração (Continuação)
Linguagem de Montagem

Kelly Rosa Braghetto

DCC–IME–USP

5 de março de 2013

Sistemas de numeração

O ENIAC usava o sistema de numeração decimal. Depois dele, todos os computadores eletrônicos usam em seus cálculos aritméticos o sistema de numeração binário.

Sistema decimal (base 10)

- Usa dez dígitos distintos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- É um sistema posicional
  - Valor de um dígito depende da posição em que ele se encontra no conjunto de dígitos que representa uma quantidade
  - O valor total do número é a soma dos valores relativos de cada dígito

	735			573	
700	30	5	500	70	3
$7x10^2$	$3x10^1$	$5x10^0$	$5x10^2$	$7x10^1$	$3x10^0$

Sistema binário (base 2)

- Usa dois dígitos distintos (0, 1)
- Estrutura de pesos dos números binários:

$$\ldots 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0,\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}\ 2^{-5}\ldots$$

Conversão de binário para decimal

Exemplo: $(111001,1)_2$

$$\begin{aligned} &= (1x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1})_{10} \\ &= (32 + 16 + 8 + 1 + 0,5)_{10} \\ &= (57,5)_{10} \end{aligned}$$

Conversão de decimal para binário

Exemplo:

$$(57,3125)_{10} = (111001,0101)_2$$

Parte inteira – Método das divisões sucessivas

$$\begin{array}{ccccccl}
57 & \div & 2 & = & 28 & \text{com resto} & 1 \quad \rightarrow \text{bit } \textbf{menos} \text{ significativo} \\
28 & \div & 2 & = & 14 & \text{com resto} & 0 \\
14 & \div & 2 & = & 7 & \text{com resto} & 0 \\
7 & \div & 2 & = & 3 & \text{com resto} & 1 \\
3 & \div & 2 & = & 1 & \text{com resto} & 1 \\
1 & \div & 2 & = & 0 & \text{com resto} & 1 \quad \rightarrow \text{bit } \textbf{mais} \text{ significativo}
\end{array}$$

Tomando-se os restos na ordem inversa da que foram gerados, temos o número 111001.

Logo, temos que $(57)_{10} = (111001)_2$.

Conversão de decimal para binário

Exemplo:

$$(57,3125)_{10} = (111001,0101)_2$$

Parte fracionária – Método das multiplicações sucessivas

0,3125	$\times$	2	=	0,	625	$\rightarrow$ bit **mais** significativo
0,625	$\times$	2	=	1,	25	
0,25	$\times$	2	=	0,	5	
0,5	$\times$	2	=	1,	0	$\rightarrow$ bit **menos** significativo

Tomando-se os restos na ordem em que foram gerados, temos o número 0101.

Logo, temos que $(0,3125)_{10} = (0,0101)_2$.

Aritmética binária

Soma

$0 + 0 = 0$
$0 + 1 = 1 + 0 = 1$
$1 + 1 = 10$

Exemplo: $1111 + 11100 = 101011$

$$\begin{array}{cccccc} & 1 & 1 & & & \\ & & 1 & 1 & 1 & 1 \\ + & 1 & 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 & 1 & 1 \end{array}$$

Aritmética binária

Subtração

0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
10 - 1 = 1

Exemplo: 10001 + 1110 = 101011

$$
\begin{array}{cccccc}
 & & 1 & 1 & & \\
 & 0 & \cancel{10} & \cancel{10} & 10 & \\
 & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & 1 \\
- & & 1 & 1 & 1 & 0 \\
\hline
 & 0 & 0 & 0 & 1 & 1
\end{array}
$$

Conversão de binário para decimal

Outro exemplo:

$$(1010101010101110)_2$$

parte 1 (8 bits)	parte 2 (8 bits)
10101010	10101110
$(170)_{10}$	$(174)_{10}$

Logo, temos que $(1010101010101110)_2 = (170 \times 2^8 + 174)_{10} = (170 \times 256 + 174)_{10} = (43694)_{10}$.

Organização da memória de um computador

- A memória é organizada como "retângulos" de bits
- Cada retângulo é chamado de palavra
- Transferências de dados de/para a memória são feitas de 1 (ou mais) palavra(s) por vez
- Palavras na memória de um computador são numeradas consecutivamente, iniciando em 0; dizemos que esses números são os endereços das palavras
- Os endereços das palavras são usados pelos processadores, nas operações de transferência de dados de/para a memória
- Capacidade de uma memória = número de palavras $\times$ tamanho da palavra
- Computadores com processadores Intel usam palavras de 8 bits

Organização da memória de um computador

- 1 byte = 8 bits
- O número de palavras na memória de um computador geralmente é uma potência grande de 2, ou um múltiplo menor de uma dessas potências
- É conveniente o uso de símbolos/prefixos especiais para denotar essas potências:

	Valor Exato	Símbolo	Prefixo	Valor Aprox.
2^{10}	1 024	k	kilo	mil
2^{20}	1 048 576	M	mega	milhão
2^{30}	1 073 741 824	G	giga	bilhão
2^{40}	1 099 511 627 776	T	tera	trilhão

Um "parênteses" sobre símbolos/prefixos

IEC prefixos binários			SI prefixos decimais		
Valor	Símbolo	Prefixo	Valor	Símbolo	Prefixo
2^{10}	Ki	kibi	10^3	k	kilo
2^{20}	Mi	mebi	10^6	M	mega
2^{30}	Gi	gibi	10^9	G	giga
2^{40}	Ti	tebi	10^{12}	T	tera
2^{50}	Pi	pebi	10^{15}	P	peta

- IEC – International Electrotechnical Commission
- SI – International System of Units
- Considerando esses prefixos, 1 kibibyte (KiB) $\neq$ 1 kilobyte (kB)

Sistema hexadecimal (base 16)

- Usa 16 dígitos distintos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)
- Estrutura de pesos dos números binários:

$$\ldots 16^5\ 16^4\ 16^3\ 16^2\ 16^1\ 16^0,\ 16^{-1}\ 16^{-2}\ 16^{-3}\ 16^{-4}\ 16^{-5} \ldots$$

Razões para aprendê-lo

- Endereços de memória são números muito grandes $\rightarrow$ representação hexadecimal é mais "curta"
- Depuradores de código geralmente exibem os valores contidos nos registradores em hexadecimal; é útil sabermos verificar a aritmética de valores em hexadecimal sem a necessidade de convertê-los para a base 10

Sistema hexadecimal (base 16)

Exemplo de conversão de hexadecimal para decimal

$$\begin{aligned}(14D)_{16} &= (1 \times 16^2 + 4 \times 16^1 + 13 \times 16^0)_{10} \\ &= (256 + 64 + 13)_{10} \\ &= (333)_{10}\end{aligned}$$

Exemplo de conversão de decimal para hexadecimal

$$\begin{array}{ccccccc} 1000 & \div & 16 & = & 62 & \text{com resto} & 8 \\ 62 & \div & 16 & = & 3 & \text{com resto} & 14 = \text{E} \\ 3 & \div & 16 & = & 0 & \text{com resto} & 3 \end{array}$$

Tomando-se os restos na ordem inversa da que foram gerados, temos o número 3E8.
Logo, temos que $(1000)_{10} = (3E8)_{16}$.

Relação entre a base binária e a base hexadecimal

Exemplo de conversão de decimal para binário

$$
\begin{array}{rcccrll}
1000 & \div & 2 & = & 500 & \text{com resto} & 0 \\
500 & \div & 2 & = & 250 & \text{com resto} & 0 \\
250 & \div & 2 & = & 125 & \text{com resto} & 0 \\
125 & \div & 2 & = & 62 & \text{com resto} & 1 \\
62 & \div & 2 & = & 31 & \text{com resto} & 0 \\
31 & \div & 2 & = & 15 & \text{com resto} & 1 \\
15 & \div & 2 & = & 7 & \text{com resto} & 1 \\
7 & \div & 2 & = & 3 & \text{com resto} & 1 \\
3 & \div & 2 & = & 1 & \text{com resto} & 1 \\
1 & \div & 2 & = & 0 & \text{com resto} & 1
\end{array}
$$

Tomando-se os restos <u>na ordem inversa da que foram gerados</u>, temos o número 1111101000.

Logo, temos que $(1000)_{10} = (1111101000)_2$.

Relação entre a base binária e a base hexadecimal

- Dividir por 2 quatro vezes equivale a dividir por 16 uma vez
- Se agruparmos os dígitos do número binário quatro a quatro, veremos a seguinte relação:

Número na base decimal:		1000	
Número na base binária:	11	1110	1000
Número na base hexadecimal:	3	E	8

- Assim, podemos usar o sistema hexadecimal como uma forma "mais legível" do binário
- Com dois dígitos em hexadecimal representamos 1 byte
- Outro exemplo:

Binário :	1011	0010	1001	0101	0000	0111	1010	1000	1000
Hexadecimal :	B	2	9	5	0	7	A	8	8

Aritmética hexadecimal

Exemplo: 47BC + A78 = 5234

$$
\begin{array}{ccccc}
 & 1 & 1 & 1 & \\
 & 4 & 7 & B & C \\
+ & & A & 7 & 8 \\
\hline
 & 5 & 2 & 3 & 4
\end{array}
$$

"Colinha":

- $(C + 8)_{16} = (12 + 8)_{10} = (20)_{10} = (14)_{16}$
- $(1 + B + 7)_{16} = (1 + 11 + 7)_{10} = (19)_{10} = (13)_{16}$
- $(1 + 7 + A)_{16} = (1 + 7 + 10)_{10} = (18)_{10} = (12)_{16}$
- $(1 + 4)_{16} = (5)_{16}$

Aritmética hexadecimal

Exemplo: 47BC - A4E = 3D6E

$$
\begin{array}{ccccc}
 & 3 & 17 & A & 1C \\
 & \cancel{4} & \cancel{7} & \cancel{B} & \cancel{C} \\
- & & A & 4 & E \\
\hline
 & 3 & D & 6 & E
\end{array}
$$

"Colinha":

- $(1C - E)_{16} = (28 - 14)_{10} = (14)_{10} = (E)_{16}$
- $(A - 4)_{16} = (10 - 4)_{10} = (6)_{10} = (6)_{16}$
- $(17 - A)_{16} = (23 - 10)_{10} = (13)_{10} = (D)_{16}$
- $(3 - 0)_{16} = (3)_{16}$

Números negativos em binário

Representação sinal-e-magnitude

- Bit mais significativo representa o sinal do número
  - 0 – número positivo
  - 1 – número negativo
- Exemplo: $(0101)_2 = (5)_{10}$ e $(1101)_2 = (-5)_{10}$

Problema: a soma fica complicada para o computador

"Algoritmo" para a soma:

- Caso 1 – os dois números são positivos: basta somá-los
- Caso 1 – os dois números são negativos: remova os sinais dos números, some-os e depois coloque o sinal de menos no resultado
- Caso 3 – um número é positivo e outro negativo: subtraia o de menor magnitude do de maior; se o de maior magnitude tem um sinal de menos, então coloque o sinal de menos no resultado

Números negativos em binário

Complemento de 2

- Usada nos computadores
- Facilita a soma: não é preciso se preocupar se o número é positivo ou negativo... basta somá-los
- Funcionamento "análogo" ao do odômetro
- Exemplo: $(4 + (-7))_{10} = (0100 + 1001)_2 = (1101)_2 = (-3)_{10}$

Decimal	Binário (4 bits) em Complemento de 2
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Números negativos em binário

Conversão de binário "puro" para Complemento de 2

- Passo 1: inverter os bits (= trocar zeros por uns e uns por zeros)
- Passo 2: somar 1 ao número resultante da inversão

Obs.: os mesmos passos valem para converter de complemento de 2 para binário puro.

Exemplos

Decimal	Binário puro	Complemento de 2 (8 bits)
-108	-01101100	10010100

$(108)_{10} =$ $(01101100)_2$
Depois da inversão: 10010011
Depois de + 1 : 10010100

Números negativos em binário

Exemplo: soma/subtração com complemento de dois

$(109 - 108)_{10} = (109 + (-108))_{10}$

$(109)_{10} \quad = 01101101$
$(-108)_{10} = 10010100$

$$
\begin{array}{rr}
 & 01101101 \\
+ & 10010100 \\
\hline
\cancel{1} & 00000001
\end{array}
$$

bit de
"carry"

Lembrete: arquitetura da família x86

Registradores de propósito geral

- A (acumulador)
- B (base)
- C (contador)
- D (dados)
- processador 8086 (16 bits): AX (AH,AL), BX (BH,BL), CX (CH,CL), DX (DH,DL), SP, BP, SI, DI
- processador 80386 (32 bits): EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI
- processador Intel x86-64 e AMD64 (64 bits): RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8–15

Linguagem de Montagem

Estrutura geral das instruções

Cada linha de um programa em linguagem de montagem é composto por 4 campos:

- **rótulo** (*label*): "nomeia" os blocos do programa. São usados nos saltos. Devem ser alfanuméricos começando por letras
- **mnemônico**: especifica uma instrução (ex.: MOV, ADD, ...)
- **operando(s)**: objeto(s) sobre o qual(is) a instrução opera. Quando uma instrução possui mais de um operando, eles devem vir separados por vírgulas. Nem toda instrução tem um operando
- **comentário**: documenta o código. É iniciado por um ponto-e-vírgula. É permitido que uma linha tenha somente o campo de comentário. (Obs.: comentários são particularmente importantes em linguagem de montagem!)

Linguagem de montagem

Exemplo de programa

[Rótulo:]	[Mnemônico]	[Operando]	[;Comentário]
	MOV	CX, 5	; inicializa contador com 5
inicio:	MOV	AX, 25h	; inicializa AX com 25h
	ADD	AX,AX	; AX <– AX + AX
	DEC	CX	; contador <– contador - 1
	JNZ	inicio	

Comando para transferência de dados: **MOV**

Copia o valor do segundo operando no primeiro operando.
O conteúdo do segundo operando permanece inalterado.

Formatos

- **MOV** ***reg*,*reg*|*mem*|*const***
- **MOV** ***mem*,*reg*|*const***

Operandos

- *reg* – um registrador de propósito geral
- *mem* – posição de memória (pode ser indicada por meio de uma constante, como `[1000]`, `ou por meio de um registrador, como [EBX]`)
- *const* – valor *constante*

Comando para transferência de dados: **MOV**

Exemplos

Correto	**Incorreto**	**Problema**
MOV AH,-14	MOV AL,999	; 999 não cabe em 8 bits
MOV AX,36H		
MOV AL,'A'	MOV EBX,DX	; não possuem o mesmo
MOV EAX,EBX		; tamanho
MOV BX,1000		
MOV AX,[EBX]		
MOV AX,[1000]		
MOV AX,[1000+EBX]		
MOV [1000],AX	MOV [1000],[EBX]	; não há MOV direto
MOV [1000],36H		; entre memórias

Considerações sobre o uso de memória como operando

Casos de não ambiguidade no tamanho

Acontecem quando a instrução envolve um operando do tipo *mem* e outro do tipo *reg*.
Neste caso, o número de palavras manipuladas na memória é determinado pelo tamanho de *reg*.
Exemplo: a instrução

MOV AX, [1000]

copia 2 palavras da memória (posições 1000 e 1001) porque o registrador AX é de 16 bits.

Considerações sobre o uso de memória como operando

Casos de ambiguidade no tamanho

Acontecem quando a instrução envolve um operando do tipo *mem* e outro do tipo *const*. Exemplo:

MOV [EBX], 5

Neste caso, o número de palavras manipuladas na memória pode ser determinado de duas maneiras:

1. a arquitetura do processador determina a quantidade de bits *default* (16 bits, 32 bits, 64 bits)
2. uso de notação para determinar o quantidade de bytes manipulados. Exemplo:

```
MOV   BYTE [EBX],5    ; BYTE para designar 8 bits
MOV   WORD [EBX],5    ; WORD para designar 16 bits
MOV   DWORD [EBX],5   ; DWORD para designar 32 bits
```

Um “parênteses”: Convenções de notação

Soluções para problemas de ambiguidade

- Problema-exemplo 1: **50** pode ser um número em notação decimal ou hexadecimal
- Solução: usar sufixos que determinam o sistema de numeração. Por exemplo, **50D** designa um número decimal, enquanto **50H** é hexadecimal (**10B** é binário)
- Problema-exemplo2 (consequência da solução anterior): **AH, BH, CH e DH** designam números hexadecimais, mas também são nomes de registradores
- Solução: na linguagem de montagem, fazer com que todos os números hexadecimais sejam também iniciados por um dígito em $0, 1, \ldots, 9$ [1]. Por exemplo, **0AH** designa o número hexadecimal A e não o registrador AH

[1] Na linguagem C, números hexadecimais são precedidos por “0x”

Comando para troca de dados: **XCGH**

Troca os valores dos operandos (ou seja, faz o primeiro receber o valor do segundo e o segundo receber o valor do primeiro).
Os operandos precisam ser do mesmo tamanho.

Formatos

- **XCGH** ***reg,reg|mem***
- **XCGH** ***mem,reg***

Exemplos

XCHG AH,BL
XCHG AH,[BL]
XCHG [EBX],AH

Instruções aritméticas – soma: **ADD**

Soma o valor do segundo operando ao valor do primeiro, armazenando o resultado no primeiro operando.
O valor do segundo operando permanece inalterado.

Formato

- **ADD** *reg,reg|mem|const*

Exemplos

ADD BL,10 ; BL <– BL + 10
ADD BL,AL ; BL <– BL + AL
ADD BL,[1000] ; BL <– BL + [1000]

Instruções aritméticas – subtração: **SUB**

Subtrai o valor do segundo operando do valor do primeiro, armazenando o resultado no primeiro operando.
O valor do segundo operando permanece inalterado.

Formato

- **SUB** *reg,reg|mem|const*

Exemplos

```
SUB    BL,10       ; BL <– BL - 10
SUB    BL,AL       ; BL <– BL - AL
SUB    BL,[1000]   ; BL <– BL - [1000]
```

Instruções aritméticas – incremento e decremento: **INC** e **DEC**

Incrementa ou decrementa o valor do operando em 1.

Formato

- **INC** ***reg|mem***
- **DEC** ***reg|mem***

Exemplos

INC	CX	$\longleftrightarrow$	ADD	CX, 1
DEC	CX	$\longleftrightarrow$	SUB	CX, 1

Bibliografia e materiais recomendados

- Capítulos 3, 4 e 6 do livro *Linux Assembly Language Programming*, de B. Neveln
- Livro *The Art of Assembly Language Programming*, de R. Hyde
  `http://cs.smith.edu/~thiebaut/ArtOfAssembly/artofasm.html`
- Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
  `http://www.ime.usp.br/~kon/MAC211`

Cenas dos próximos capítulos...

- Mais instruções em linguagem de montagem
- Estrutura de um programa em linguagem de montagem
- Montadores
- Primeiro programa completo em linguagem de montagem