



UNIVERSIDADE LUSÓFONA
de Humanidades e Tecnologias
Humani nihil alienum

Sebenta

ARQUITECTURA DE COMPUTADORES

2006/2007

Paulo Matos

Curso: Engenharia Informática
Cadeira: Arquitectura de Computadores
Ano curricular: 1º
Semestre: 1º
Docente: Alexandre Pereira

Programa do ano lectivo: 2004/2005

Apresentação

Esta cadeira compreende uma vertente teórica e outra prática. Do ponto de vista teórico, são abordadas as questões da estrutura e organização dos computadores digitais. A vertente prática explora a programação em linguagem máquina, como forma de ligar os conceitos teóricos de base.

Objectivos

Entender o funcionamento dos computadores, de uma perspectiva microscópica até uma perspectiva macroscópica. Reconhecimento das potencialidades e limitações de um computador. Entendimento da forma como interagem os diversos elementos da estrutura/arquitectura de um computador.

No final da cadeira, os alunos devem ser capazes de: reconhecer os diversos blocos que compõem um computador digital; diferenciar os diferentes tipos de arquitecturas possíveis; identificar as limitações do hardware e as suas consequências ao nível do software. Devem também ser capazes de elaborar algoritmos para resolver problemas em linguagem máquina.

Condições de obtenção de aprovação à disciplina

A avaliação é feita por intermédio de testes escritos e trabalhos práticos. Os alunos são aprovados com a nota final de 10 valores.

Modelo de avaliação

A avaliação tem duas componentes: avaliação teórica – 2 testes realizados a meio e no fim das aulas, com uma ponderação total de 60% na nota final (30% cada); avaliação prática – um trabalho prático, a realizar em grupo, com uma ponderação de 40% na nota final.

Materiais de apoio e bibliografia

Hayes, John P. – "Computer Architecture and Organization", McGraw Hill, 1978

INTEL – *IA-32 Intel® Architecture Software Developer's Manual, Volume 1: Basic Architecture* [Em linha]. Colorado Springs : INTEL, 2004. Disponível na WWW:
<<ftp://download.intel.com/design/Pentium4/manuals/25366514.pdf>>.

INTEL – *IA-32 Intel® Architecture Software Developer's Manual, Volume 2A: Instruction Set Reference, A-M* [Em linha]. Colorado Springs : INTEL, 2004. Disponível na WWW:
<<ftp://download.intel.com/design/Pentium4/manuals/25366614.pdf>>.

INTEL – *IA-32 Intel® Architecture Software Developer's Manual, Volume 2B: Instruction Set Reference, N-Z* [Em linha]. Colorado Springs : INTEL, 2004. Disponível na WWW: <<ftp://download.intel.com/design/Pentium4/manuals/25366714.pdf>>.

INTEL – *IA-32 Intel® Architecture Software Developer's Manual, Volume 3: System Programming Guide* [Em linha]. Colorado Springs : INTEL, 2004. Disponível na WWW: <<ftp://download.intel.com/design/Pentium4/manuals/25366814.pdf>>.

PEREIRA, Alexandre – *Arquitetura de computadores* [Em linha]. Almada : Mediateca, Lda., 2003. Disponível na WWW: <<http://escola.mediateca.info/ac/>>.

Tanenbaum, Andrew S. – "Structured Computer Organization", Prentice-Hall, 1990

Conteúdos Programáticos

SISTEMAS DE NUMERAÇÃO

- N.1 - INTRODUÇÃO
- N.2 - CONVERSÃO DA BASE B PARA A BASE 10
- N.3 - CONVERSÃO DA BASE 10 PARA A BASE B
 - N.3.1 - Método da divisão-multiplicação
 - N.3.2 - Método da subtracção
 - N.3.3 - Capacidade de numeração
- N.4 - SISTEMA BINÁRIO, OCTAL E HEXADECIMAL
 - N.4.1 - Relação entre as bases 8 e 2
 - N.4.2 - Relação entre as bases 16 e 2
- N.5 - OPERAÇÕES ARITMÉTICAS EM BASES DIFERENTES DE 10
 - N.5.1 - Adição
 - N.5.2 - Subtracção
 - N.5.3 - Multiplicação
 - N.5.4 - Divisão

ARQUITECTURAS DE COMPUTADORES

- I.1 - PRINCIPAIS TIPOS DE ORGANIZAÇÃO
 - I.1.1 - MONOPROCESSAMENTO
 - I.1.2 - PROCESSAMENTO COM OPERAÇÕES MÚLTIPLAS
 - I.1.2.1 - SIMD - Paralelismo de array
 - I.1.2.2 - MISD - Paralelismo de pipeline
 - I.1.2.3 - MIMD - Paralelismo funcional
- I.2 - MEMÓRIA
 - I.2.1 - Memória partilhada
 - I.2.2 - Memória privada
- I.3 - Redes de interligação
 - I.3.1 - Crossbar
 - I.3.2 - Hipercubo
 - I.3.3 - Hiperbus
- I.4 - ORGANIZAÇÃO DE SISTEMAS
 - I.4.1 - Redes de computadores
 - I.4.2 - Computação distribuída
- I.5 - MÁQUINAS MULTI-NÍVEL

A UNIDADE CENTRAL DE PROCESSAMENTO

- P.1 - ORGANIZAÇÃO DA CPU
- P.2 - REPRESENTAÇÃO DA INFORMAÇÃO
 - P.2.1 - Dados não numéricos
 - P.2.2 - Dados numéricos
 - P.2.2.1 - Nomenclatura
 - P.2.2.2 - Formatos dos números
 - P.2.2.2.1 - Vírgula fixa
 - P.2.2.2.2 - Vírgula flutuante
 - P.2.3 - Formatos de instruções (máquina)
- P.3 - UNIDADES ARITMÉTICAS E LÓGICAS
 - P.3.1 - Adição, Subtração e Operações lógicas
 - P.3.2 - Multiplicação
 - P.3.5 - Operações em vírgula flutuante
- P.4 - UNIDADES DE CONTROLE MICROPROGRAMADAS
 - P.4.1 - Introdução
 - P.4.2 - Modelo de WILKES
 - P.4.3 - Comprimento da microinstrução
 - P.4.4 - Organização de uma unidade de controle microprogramada
 - P.4.5 - Escrita de microprogramas
 - P.4.6 - Exemplo de CPU microprogramada
 - P.4.7 - O processador HP 21 MX

MEMÓRIA

- M.1 - HIERARQUIA DE MEMÓRIAS NUM COMPUTADOR DIGITAL
- M.2 - TECNOLOGIA
 - M.2.1 - Características das principais memórias
 - M.2.2 - Memórias de acesso aleatório
 - M.2.3 - Memórias de acesso sequencial
- M.3 - MEMÓRIA CENTRAL
 - M.3.1 - Generalidades
 - M.3.2 - Protecções no acesso a memória central
- M.4 - ORGANIZAÇÃO DE MEMÓRIA DE ALTA VELOCIDADE
 - M.4.1 - Memórias paralelas
 - M.4.2 - Memórias paralelas em multiprocessadores e processadores paralelos
 - M.4.3 - Memórias cache
 - M.4.3.1 - Configuração de um sistema com memória cache
 - M.4.3.1.1 - Organização sectorial
 - M.4.3.1.2 - Organização de correspondência directa (Direct Mapping)
 - M.4.3.2 - Eficiência no acesso à cache
 - M.4.4 - Memórias associativas
- M.5 - PROCESSAMENTO DE ENDEREÇOS
 - M.5.1 - Sistema mono-programado
 - M.5.2 - Sistemas multi-programados
 - M.5.2.1 - Sistema de partições
 - M.5.2.2 - Paginação
 - M.5.2.3 - Memória virtual baseada em paginação
 - M.5.2.4 - Memória virtual segmentada

A FAMÍLIA 8086/Pentium

- X.1 - INTRODUÇÃO
- X.2 - A ARQUITECTURA DOS PROCESSADORES
 - X.2.1 - Registos gerais

- X.2.2 - Registos de segmento e apontador de instrução
- X.2.3 - Flags
- X.3 - MEMÓRIA
 - X.3.1 - Organização
 - X.3.2 - Segmentação/Paginação
 - X.3.3 - Geração de endereços físicos
 - X.3.4 - Implementação da Stack (pilha)
- X.4 - INSTRUÇÕES
 - X.4.1 - Instruções de transferência de dados
 - X.4.2 - Instruções aritméticas
 - X.4.3 - Instruções de manipulação de bits
 - X.4.4 - Instruções sobre strings
 - X.4.5 - Instruções de transferência de controlo
 - X.4.6 - Instruções de controlo do processador
- X.5 - MODOS DE ENDEREÇAMENTO
 - X.5.1 - Operandos em registo e imediato
 - X.5.2 - Endereçamento directo
 - X.5.3 - Endereçamento indirecto
 - X.5.4 - Endereçamento com registo de base
 - X.5.5 - Endereçamento indexado
 - X.5.6 - Endereçamento indexado e com registo de base
 - X.5.7 - Endereçamento de strings
 - X.5.8 - Endereçamento de portos de I/O
- X.6 - PROGRAMAÇÃO EM ASSEMBLY 386/Pentium
 - X.6.1 - Definição de dados
 - X.6.2 - Directivas
 - X.6.3 - Procedimentos
 - X.6.4 - Strings

Software:

- Sistema operativo Linux
- NASM – Assemblador do S. O. Linux (incluído no sistema)

Hardware:

- Computadores

Sistemas de numeração em base 2, 8 e 16

Conversão de base (b) para base (10)

$$325_{(10)} = 3 \times 100 + 2 \times 10 + 5 \times 1 = 325_{(10)}$$

10^2	10^1	10^0
3	2	5

Contagem em base 10

0
1
2
3
4
5
6
7
8
9
10
11

Alfabeto em base 10 = {0,1,2,3,4,5,6,7,8,9}

Contagem em base 2

0
1
10
11
100
101
110

Alfabeto em base 2 = {0,1}

2^2	2^1	2^0
1	0	1

$$101_{(2)} = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = 5_{(10)}$$

Exercício:

Converter $10011101_{(2)}$ para base 10

Solução:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	1	1	0	1

$$\begin{aligned} 10011101_{(2)} &= 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 \\ &\quad + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 2^7 + 2^4 + 2^3 + 2^2 + 2^0 \\ &= 128 + 16 + 8 + 4 + 1 \\ &= 157_{(10)} \end{aligned}$$

Contagem em base 8

0
1
2
3
4
5
6
7
10
11
12
13

Alfabeto em base 8 = {0,1,2,3,4,5,6,7}

8^2	8^1	8^0
5	2	4

$$524_{(8)} = 5 \times 8^2 + 2 \times 8^1 + 4 \times 8^0 = 320 + 16 + 4 = 340_{(10)}$$

Contagem em base 16

0
1
2
3
4
5
6

Alfabeto em base 16 = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

6
7
8
9
A
B
C
D
E
F
10
11
12
13

16^2	16^1	16^0
2	E	3

$$2E3_{(16)} = 2 \times 16^2 + 14 \times 16^1 + 3 \times 16^0 = 512 + 224 + 3 = 739_{(10)}$$

Conversão de base (10) para base (b)

Número em base (b): $a_n a_{n-1} a_{n-2} \dots a_2 a_1 a_0$

O mesmo número em base 10:

$$a_n \times b^n + a_{n-1} \times b^{n-1} + a_{n-2} \times b^{n-2} \dots + a_2 \times b^2 + a_1 \times b^1 + a_0 \times b^0$$

Dividir os coeficientes sucessivamente por b, obtêm-se os a_i .

$$(a_n b^n + a_{n-1} b^{n-1} + a_{n-2} b^{n-2} \dots + a_2 b^2 + a_1 b^1 + a_0 b^0) / b$$

Resultado 1ª divisão:

$$\text{Cociente: } a_n b^{n-1} + a_{n-1} b^{n-2} + a_{n-2} b^{n-3} \dots + a_2 b^1 + a_1 b^0$$

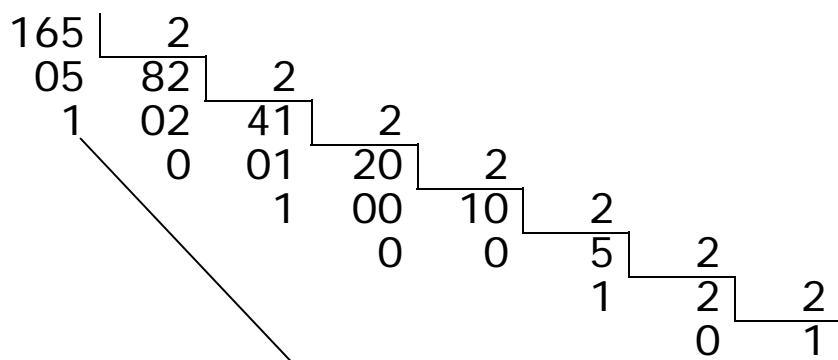
$$\text{Resto: } a_0$$

Resultado 2ª divisão:

$$\text{Cociente: } a_n b^{n-2} + a_{n-1} b^{n-3} + a_{n-2} b^{n-4} \dots + a_2 b^0$$

$$\text{Resto: } a_1$$

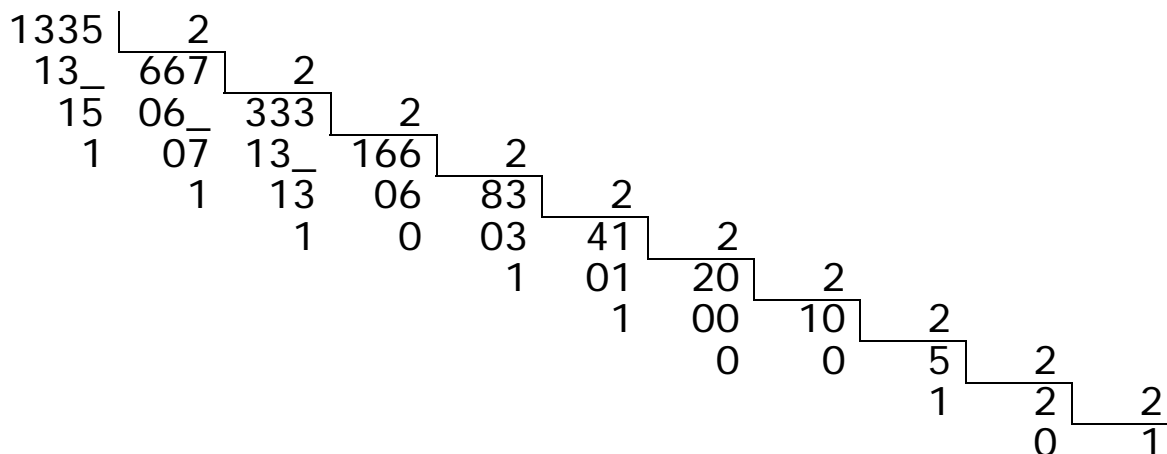
Exemplo: Converter para base 2 o número $165_{(10)}$



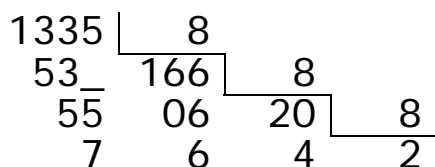
Resultado: $10100101_{(2)}$

Exercícios:

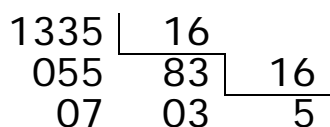
Converter o número $1335_{(10)}$ para base 2, 8 e 16.



Em base 2: $10100110111_{(2)}$



Em base 2: $2467_{(8)}$



Em base 16: $537_{(16)}$

Conversões entre base 2 e bases 8 e 16

Conversão entre base 2 e 8

$$a_{3n+2}2^{3n+2} + a_{3n+1}2^{3n+1} + a_{3n}2^{3n} \dots + a_52^5 + a_42^4 + a_32^3 + a_22^2 + a_12^1 + a_02^0$$

Juntar em grupos de 3 parcelas, a começar pela direita:

$$(a_{3n+2}2^{3n+2} + a_{3n+1}2^{3n+1} + a_{3n}2^{3n}) \dots + (a_52^5 + a_42^4 + a_32^3) + (a_22^2 + a_12^1 + a_02^0)$$

Em cada grupo, factorizar potências de 2:

$$(a_{3n+2}2^2 + a_{3n+1}2^1 + a_{3n}2^0) \cdot 2^{3n} \dots + (a_52^2 + a_42^1 + a_32^0) \cdot 2^3 + (a_22^2 + a_12^1 + a_02^0) \cdot 2^0$$

Passar potências de 2 factorizadas a potências de 8:

$$(a_{3n+2}2^2 + a_{3n+1}2^1 + a_{3n}2^0) \cdot 8^n \dots + (a_52^2 + a_42^1 + a_32^0) \cdot 8^1 + (a_22^2 + a_12^1 + a_02^0) \cdot 8^0$$

Somar as parcelas dentro de parêntesis:

$$x_n8^n \dots + x_18^1 + x_08^0$$

em que x_i é um número de 0 a 7, segundo a tabela:

(8)	(2)
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Converter de base 2 para base 8: $11011010110_{(2)}$

011	011	010	110
3	3	2	6

$$11011010110_{(2)} = 3326_{(8)}$$

Para converter de base 2 para base 16, usar a tabela:

(16)	(2)
0	0000

1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Converter de base 2 para base 16: $11011010110_{(2)}$

0110	1101	0110
6	D	6

$$11011010110_{(2)} = 6D6_{(16)}$$

Converter de base 16 para base 8: $AB9C3_{(16)}$

Passar primeiro para base 2:

A	B	9	C	3
1010	1011	1001	1100	0011

$$AB9C3_{(16)} = 10101011100111000011_{(2)}$$

Converter de base 2 para base 8:

010	101	011	100	111	000	011
2	5	3	4	7	0	3

$$AB9C3_{(16)} = 2534703_{(8)}$$

Números fracionários

Converter base (b) para base (10)

$$165,23_{(10)} = 1 \times 10^2 + 6 \times 10^1 + 5 \times 10^0 + 2 \times 10^{-1} + 3 \times 10^{-2}$$

10^2	10^1	10^0	10^{-1}	10^{-2}
1	6	5	2	3

Converter para base 10: $10110,101_{(2)}$

Lembrar que: $b^{-n} = 1/b^n$

$$2^{-3} = \frac{1}{2^3} = \frac{1}{8} = 0,125$$

$$\begin{aligned} 10110,101_{(2)} &= 2^4 + 2^2 + 2^1 + 2^{-1} + 2^{-3} \\ &= 16 + 4 + 2 + 0,5 + 0,125 \\ &= 22,625_{(10)} \end{aligned}$$

Converter para base 10: $375,34_{(8)}$

$$\begin{aligned} 375,34_{(8)} &= 3 \times 8^2 + 7 \times 8^1 + 5 \times 8^0 + 3 \times 8^{-1} + 4 \times 8^{-2} \\ &= 192 + 56 + 5 + 0,375 + 0,0625 \\ &= 253,4375_{(10)} \end{aligned}$$

$$\begin{aligned} 3 \times 8^{-1} &= \frac{3}{8^1} = \frac{3}{8} = 0,375 \\ 4 \times 8^{-2} &= \frac{4}{8^2} = \frac{4}{64} = \frac{1}{16} = 0,0625 \end{aligned}$$

Converter para base 10: $1A7,C8_{(16)}$

$$\begin{aligned} 1A7,C8_{(16)} &= 1 \times 16^2 + 10 \times 16^1 + 7 \times 16^0 + 12 \times 16^{-1} + 8 \times 16^{-2} \\ &= 256 + 160 + 7 + 0,75 + 0,03125 \\ &= 423,78125_{(10)} \end{aligned}$$

$$\begin{aligned} 12 \times 16^{-1} &= \frac{12}{16^1} = \frac{3}{4} = 0,75 \\ 8 \times 16^{-2} &= \frac{8}{16^2} = \frac{1}{16 \times 2} = \frac{1}{32} = 0,03125 \end{aligned}$$

Converter base (10) para base (b)

Número em base (b): $0, a_{-1}a_{-2}...a_{-n}$

O mesmo número em base 10:

$$a_{-1} \times b^{-1} + a_{-2} \times b^{-2} + \dots + a_{-n} \times b^{-n}$$

Multiplicar a parte fracionária dos produtos sucessivamente por b, obtêm-se os a_i .

$$(a_{-1} \times b^{-1} + a_{-2} \times b^{-2} + \dots + a_{-n} \times b^{-n}) \times b$$

Resultado da 1ª multiplicação:

$$a_{-1} \times b^0 + a_{-2} \times b^{-1} + \dots + a_{-n} \times b^{-n+1}$$

0 que corresponde ao número:

$a_{-1}, a_{-2} \dots a_{-n}$

Depois multiplica-se apenas a parte fracionária:

$0, a_{-2} \dots a_{-n}$

Ou seja:

$$(a_{-2} \times b^{-1} + \dots + a_{-n} \times b^{-n+1}) \times b$$

Resultado da 2ª multiplicação:

$$a_{-2} \times b^0 + a_{-3} \times b^{-1} + \dots + a_{-n} \times b^{-n+2}$$

0 que corresponde ao número:

$a_{-2}, a_{-3} \dots a_{-n}$

Por fim constrói-se o número, com base nos a_i .

Resultado final: $0, a_{-1} a_{-2} \dots a_{-n}$

Exemplo: Passar $0,5625_{(10)}$ para base 2

$0,5625 \times 2 = 1,125$	extraí-se um 1 (inteiro)
$0,125 \times 2 = 0,25$	extraí-se um 0 (inteiro)
$0,25 \times 2 = 0,5$	extraí-se um 0 (inteiro)
$0,5 \times 2 = 1,0$	extraí-se um 1 (inteiro)

$$0,5625_{(10)} = 0,1001_{(2)}$$

Exercício: Passar $12,3125_{(10)}$ para base 2

12		2		
0		6		2
		0		3
				1
				2
				1

$$12_{(10)} = 1100_{(2)}$$

$0,3125 \times 2 = 0,625$
$0,625 \times 2 = 1,25$
$0,25 \times 2 = 0,5$
$0,5 \times 2 = 1,0$

$$12,3125_{(10)} = 1100,0101_{(2)}$$

Converter base (2) para base (8) ou (16) e vice-versa

Converter para base 8: $11101,10101_{(2)}$

011	101	101	010
3	5	5	2

$$11101,10101_{(2)} = 35,52_{(8)}$$

Converter para base 16: $11101,10101_{(2)}$

0001	1101	1010	1000
1	D	A	8

$$11101,10101_{(2)} = 1D,A8_{(16)}$$

Soma em base 10

$$\begin{array}{r}
 \text{1} \quad \text{1} \quad \leftarrow \text{Transporte da coluna anterior} \\
 + \quad 1 \quad 5 \quad 6 \quad (10) \\
 \quad \quad 6 \quad 7 \quad (10) \\
 \hline
 \quad \quad 2 \quad 2 \quad 3 \quad (10)
 \end{array}$$

Soma em base 2

Tabela de soma de 2 bits

$$\begin{array}{r}
 0 \quad 0 \quad 1 \quad 1 \\
 + \quad 0 \quad + \quad 1 \quad + \quad 0 \quad + \quad 1 \\
 \hline
 0 \quad 1 \quad 1 \quad \text{1} \quad 0
 \end{array}$$

Vai um (a somar à coluna seguinte)

Exemplo:

$$\begin{array}{r}
 \text{1} \quad \text{1} \quad \text{1} \quad \leftarrow \text{Transporte da coluna anterior} \\
 + \quad 1 \quad 0 \quad 1 \quad 1 \quad (2) \\
 \quad \quad 1 \quad 0 \quad 1 \quad (2) \\
 \hline
 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad (2)
 \end{array}$$

$$\begin{array}{l}
 2^3 + 2^1 + 2^0 = 11_{(10)} \\
 2^2 + 2^0 = 5_{(10)} \\
 2^4 = 16_{(10)}
 \end{array}$$

Exercício:

$$\begin{array}{r}
 \text{1} \quad \text{1} \quad \text{1} \quad \text{1} \quad \text{1} \quad \leftarrow \text{Transporte da coluna anterior} \\
 + \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad (2) \\
 \quad \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad (2) \\
 \hline
 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad (2)
 \end{array}$$

$$\begin{array}{l}
 2^5 + 2^3 + 2^1 + 2^0 = 43_{(10)} \\
 2^4 + 2^3 + 2^2 + 2^0 = 29_{(10)} \\
 2^6 + 2^3 = 72_{(10)}
 \end{array}$$

Soma em base 8

$$\begin{array}{r}
 \text{1} \quad \text{1} \quad \leftarrow \text{Transporte da coluna anterior} \\
 + \quad 1 \quad 5 \quad 6 \quad (8) \\
 \quad \quad 6 \quad 7 \quad (8) \\
 \hline
 \quad \quad 2 \quad 4 \quad 5 \quad (8)
 \end{array}$$

Soma em base 16

$$\begin{array}{r}
 \begin{array}{c} 1 \quad 1 \\ + \quad 1 \quad A \quad E \\ \hline 2 \quad 1 \quad 5 \end{array} \quad \begin{array}{l} (16) \\ (16) \\ (16) \end{array}
 \end{array}
 \quad \leftarrow \text{Transporte da coluna anterior}$$

Subtração em base 10

$$\begin{array}{r}
 \begin{array}{c} 1 \\ - \quad 1 \quad 5 \quad 6 \\ \hline 1 \quad 1 \quad 9 \end{array} \quad \begin{array}{l} (10) \\ (10) \\ (10) \end{array}
 \end{array}
 \quad \leftarrow \text{Transporte da coluna anterior}$$

Subtração em base 2

Tabela da subtração de 2 bits

$$\begin{array}{r}
 \begin{array}{c} 0 \\ - 0 \\ \hline 0 \end{array} \quad
 \begin{array}{c} 0 \\ - 1 \\ \hline 1 \end{array} \quad
 \begin{array}{c} 1 \\ - 0 \\ \hline 1 \end{array} \quad
 \begin{array}{c} 1 \\ - 1 \\ \hline 0 \end{array}
 \end{array}$$

Vai um (a subtrair na coluna seguinte)

Exemplo:

$$\begin{array}{r}
 \begin{array}{c} 1 \\ - \quad 1 \quad 0 \quad 1 \quad 1 \\ \hline 0 \quad 1 \quad 1 \quad 0 \end{array} \quad \begin{array}{l} (2) \\ (2) \\ (2) \end{array}
 \end{array}
 \quad \leftarrow \text{Transporte da coluna anterior}$$

$$\begin{array}{l}
 2^3 + 2^1 + 2^0 = 11_{(10)} \\
 2^2 + 2^0 = 5_{(10)} \\
 2^2 + 2^1 = 6_{(10)}
 \end{array}$$

Exercício:

$$\begin{array}{r}
 \begin{array}{c} 1 \quad 1 \quad 1 \\ - \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \\ \hline 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \end{array} \quad \begin{array}{l} (2) \\ (2) \\ (2) \end{array}
 \end{array}
 \quad \leftarrow \text{Transporte da coluna anterior}$$

$$\begin{array}{l}
 2^5 + 2^3 + 2^1 + 2^0 = 43_{(10)} \\
 2^4 + 2^3 + 2^2 + 2^0 = 29_{(10)} \\
 2^3 + 2^2 + 2^1 = 14_{(10)}
 \end{array}$$

Subtracção em base 8

$$\begin{array}{r}
 1 \quad \leftarrow \text{Transporte da coluna anterior} \\
 1 \ 5 \ 6 \quad (8) \quad 1x8^2 + 5x8^1 + 6x8^0 = 110_{(10)} \\
 - \quad 6 \ 4 \quad (8) \quad 6x8^1 + 4x8^0 = 52_{(10)} \\
 \hline
 0 \ 7 \ 2 \quad (8) \quad 7x8^1 + 2x8^0 = 58_{(10)}
 \end{array}$$

Exercí ci o:

	1	1	1	1		← Transporte da coluna anterior
	1	0	0	5	6	(8)
-		2	7	5	7	(8)
	0	5	0	7	7	(8)

Subtracção em base 16

	1	1	← Transporte da col una anterior
A	5	3	(16)
-	5	4	(16)
	9	F	F (16)

Exercí ci o:

	1	1	1	← Transporte da coluna anterior
A	A	0	3	(16)
-	B	F	4	(16)
	9	E	0	F (16)

Multiplicação em base 10

Exempl o:

		1	5	4	(10)
x			2	3	(10)
<hr/>					
		4	6	2	
+	3	0	8		
<hr/>					
	3	5	4	2	

$$\begin{aligned} 154 \times 23 &= \\ 154 \times (20 + 3) &= \\ 154 \times 20 + 154 \times 3 &= \\ 154 \times 2 \times 10 + 154 \times 3 & \end{aligned}$$

Multiplicação em base 2

Exemplo:

$$\begin{array}{r}
 \begin{array}{cccccc}
 & & 1 & 0 & 1 & 1 \\
 x & & & 1 & 0 & 1 \\
 \hline
 & & 1 & 0 & 1 & 1 \\
 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 1 & 1 & & \\
 \hline
 1 & 1 & 0 & 1 & 1 & 1
 \end{array}
 \quad
 \begin{array}{l}
 2^3 + 2^1 + 2^0 = 11_{(10)} \\
 2^2 + 2^0 = 5_{(10)} \\
 2^5 + 2^4 + 2^2 + 2^1 + 2^0 = 55_{(10)}
 \end{array}
 \end{array}$$

Exercício:

$$\begin{array}{r}
 \begin{array}{cccccccc}
 & & & & 1 & 1 & 1 & 0 & 1 & 1 \\
 x & & & & & 1 & 0 & 1 & 1 & 1 \\
 \hline
 & & & & 1 & 1 & 1 & 0 & 1 & 1 \\
 & & & 1 & 1 & 1 & 0 & 1 & 1 & \\
 & & 1 & 1 & 1 & 0 & 1 & 1 & & \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & \\
 1 & 1 & 1 & 0 & 1 & 1 & & & & \\
 \hline
 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1
 \end{array}
 \end{array}$$

Transportes da coluna anterior

Divisão em base 10

Exemplo:

$$\begin{array}{r}
 154 \overline{) 13} \\
 \underline{-13} \\
 24 \\
 \underline{-13} \\
 11
 \end{array}$$

Divisão em base 2

Exemplo:

$$\begin{array}{r|l} 1011101 & 110 \\ -110 & 1111 \\ \hline 0101\mathbf{1} & \\ -110 & \\ \hline 0101\mathbf{0} & \\ -110 & \\ \hline 0100\mathbf{1} & \\ -110 & \\ \hline 0011 & \end{array}$$

Exercício:

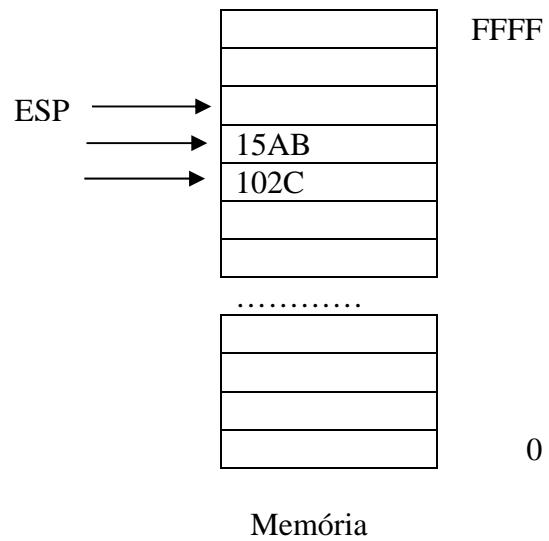
$$\begin{array}{r|l} 1100011 & 1011 \\ -1011 & 1001 \\ \hline 0001\mathbf{011} & \\ -1011 & \\ \hline 0000 & \end{array}$$

Uso explícito da STACK

```
mov eax, 15AB
mov ebx, 102C

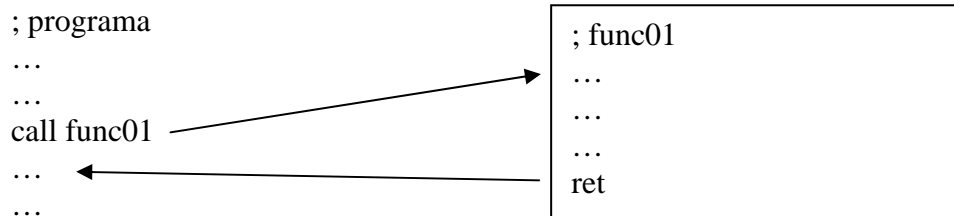
push eax
push ebx

pop ecx
; ecx fica com 102C
pop edx
; edx fica com 15AB
```



Uso implícito da STACK

CALL usa PUSH
RET usa POP



CALL faz push do endereço de retorno e salta para a função.
RET faz pop do endereço de retorno e volta ao programa.

Instruções aritméticas

Adição

V1 dd 8

```
mov eax, 23
mov ebx, 7
add eax, ebx ; eax fica com o valor 30
add ebx, -2 ; ebx fica com 5
add [V1], 4 ; V1 fica com 12
add eax, [V1] ; eax fica com 42
```

adc eax, ebx ; equivalente a: $eax = eax + ebx + CF$

inc eax ; equivalente a: $eax = eax + 1$

Subtracção

sbb eax, ebx ; equivalente a: $eax = eax - ebx - CF$

dec eax ; equivalente a: $eax = eax - 1$

neg eax ; equivalente a: $eax = -eax$

Multiplicação

Os registos eax e edx são usados implicitamente.

eax – multiplicador

edx : eax – resultado

Neste exemplo o ebx é o multiplicando (mas podia ser outro registo qualquer):

```
mov eax, 4
```

```
mov ebx, 7
```

```
mul ebx
```

O resultado fica em edx : eax.

Neste caso, ficam com os seguintes valores:

edx – 0

eax – 28

Divisão

Os registos eax e edx são usados implicitamente.

edx : eax – dividendo

eax – cociente

edx – resto

Neste exemplo o ebx é o divisor (mas podia ser outro registo qualquer):

```
mov edx, 0
```

```
mov eax, 35
```

```
mov ebx, 8
```

```
div ebx
```

O resultado desta divisão é:

eax – 4

edx – 3

CBW – Converte byte em word

```
mov al, -5
mov bx, 1024
add bx, al ; erro! registos de tamanhos diferentes
```

```
mov al, -5
mov bx, 1024
cbw
add ax, bx
```

Justificação:

O número -5 é representado, em binário com 8 bits, da seguinte forma: 11111011

5=00000101 -5=11111010 (em complemento para 1) -5=11111011 (em complemento para 2)
--

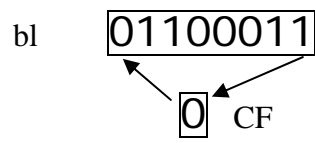
O mesmo número -5 é representado, em binário com 16 bits, da seguinte forma:
1111111111111011

A instrução CBW converte o número 11111011 em 1111111111111011.

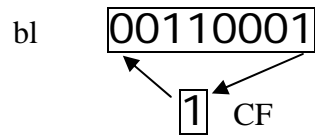
Instruções de manipulação de bits

al	10110101	
bl	01100011	
and al, bl	00100001	
or al, bl	11110111	
xor al, bl	11010110	
not al	01001010	
shl bl, 1	11000110	0 bit mais à esquerda perde-se e entra 0 pela direita
shl al, 1	01101010	0 bit mais à esquerda perde-se e entra 0 pela direita
shr al, 1	01011010	0 bit mais à direita perde-se e entra 0 pela esquerda
sar al, 1	11011010	0 bit mais à direita perde-se e entra o sinal pela esquerda
sar bl, 1	00110001	0 bit mais à direita perde-se e entra o sinal pela esquerda
rol bl, 1	11000110	0 bit que sai pela esquerda entra pela direita
ror al, 1	11011010	0 bit que sai direita pela esquerda
rcr bl, 1	00110001	Supor que CF=0 antes da instrução. No final, fica CF=1

Explicação do rcr bl,1:



Após instrução



Instruções Assembly para manipular strings

Byte	Word	Doubleword
movsb	movsw	movsd
scasb	scasw	scasd
cmps	cmpsw	cmpsd
lods	lodsw	lodsd
stos	stosw	stosd

ASCII – código de 7 bits, estendido a 8 bits

ASCII – American Standard Committee for Information Interchange

UNICODE	UTF-8
	UTF-16
	UTF-32

Com os prefixos REP é possível executar as instruções anteriores repetidas vezes.

Exemplo: **rep movsb**

Move um string completa da origem para o destino

Completa – nº de repetições é colocado no ECX

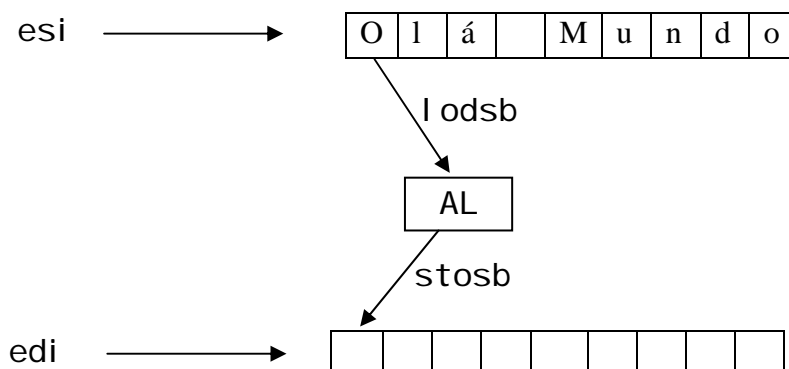
Origem – endereço que está em ESI

Destino – endereço que está em EDI

Funcionamento das instruções lodsb e stosb

Exemplo:

```
frase      db  "Olá Mundo"
destino     resb 80
```



Se a flag de direcção (DF) for igual a 0:

Cada vez que o lodsb é executado, o registo esi é incrementado 1 unidade.

Cada vez que o stosb é executado, o registo edi é incrementado 1 unidade.

Se a flag de direcção (DF) for igual a 1:

Cada vez que o lodsb é executado, o registo esi é decrementado 1 unidade.

Cada vez que o stosb é executado, o registo edi é decrementado 1 unidade.

Controlo de fluxo

Condicional tipo IF

Exemplo: calcular o valor absoluto de um número

```
    mov  eax, numero
    cmp  eax, 0
    jge  posi ti vo
    neg  eax
posi ti vo:
    mov  numero, eax
```

Iteração tipo WHILE

Exemplo: somar os 10 primeiros números

```
    mov  soma, 0
    mov  eax, 1
ci cl o:
    cmp  eax, 10
    jg   fi m
    add  soma, eax
    inc  eax
    jmp  ci cl o
fi m:
```

Iteração tipo FOR

Exemplo: o mesmo

```
    mov  soma, 0
    mov  ecx, 10
ci cl o:
    add  soma, ecx
    loop ci cl o
```

Ler números do teclado

Ler números em hexadecimal

Os números são lidos do teclado como sequências de caracteres (strings de texto)

Tabela ASCII

Símbolo	Código (10)	Código (16)
0	48	30
1	49	31
2	50	32
3	51	33
...
9	57	39
...
A	65	41
B	66	42
C	67	43
...
Z	90	5A
...
a	97	61
b	98	62
c	99	63
...
z	122	7A

Ex:

Se digitar o número "A32B50FE" no teclado, o programa recebe-o como string, ou seja:

01000001 00110011 00110010 01000010 00110101 00110000
01000110 01000101

No entanto, queríamos recebê-lo como o número A32B50FE₍₁₆₎
ou 1010 0011 0010 1011 0101 0000 1111 1110₍₂₎

Processamento da entrada (transformar texto no número correspondente)

Recebemos ('A') 01000001 mas queremos 1010
Recebemos ('3') 00110011 mas queremos 0011
Recebemos ('2') 00110010 mas queremos 0010
Recebemos ('B') 01000010 mas queremos 1011

Para os números: descartar os 4 bits mais significativos:

Recebemos ('3') 00110011 → 0011
Recebemos ('2') 00110010 → 0010

Para as letras: descartar os 4 bits mais significativos e somar 9:

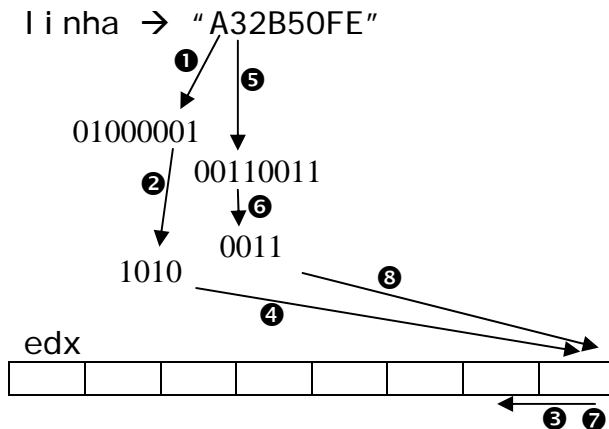
Recebemos ('A') 01000001 → 0001₍₂₎ + 9₍₁₀₎ = 1010₍₂₎
Recebemos ('B') 01000010 → 0010₍₂₎ + 9₍₁₀₎ = 1011₍₂₎

Com minúsculas:

Recebemos ('a') 01100001 $\rightarrow 0001_{(2)} + 9_{(10)} = 1010_{(2)}$
Recebemos ('b') 01100010 $\rightarrow 0010_{(2)} + 9_{(10)} = 1011_{(2)}$

Programa em Assembly para efectuar esta conversão:

Cada símbolo (letra/carácter) processado vai ser guardado no edx:



```
section .data  
num      dd  0  
num_lidos dd  0
```

```
section .bss  
linha     resb 9
```

```
section .text
```

```
; ler número do teclado para a variável linha  
; decrementar "num_lidos" numa unidade  
; converter "linha" para "num"
```

```
    xor     edx, edx ; apaga edx  
    xor     eax, eax ; apaga eax  
    mov     esi, linha  
    cld  
    mov     ecx, [num_lidos]  
converte:  
    lodsb  
    cmp     al, '9'  
    jbe     algarismo  
    add     al, 9  
algarismo:  
    and     al, 0xF  
    shl     edx, 4  
    or      edx, eax  
    loop    converge  
    mov     [num], edx
```

Escrever números no ecrã

Escrever números em hexadecimal

Os números são têm que ser escritos como sequências de caracteres (strings de texto)

Ex:

Supondo que temos o número 0xA32B50FE no programa e que o queremos imprimir no ecrã, temos que o transformar na string correspondente.

Temos o número A32B50FE₍₁₆₎ ou 1010 0011 0010 1011 0101 0000 1111 1110₍₂₎

Mas queremos:

01000001 00110011 00110010 01000010 00110101 00110000
01000110 01000101

Processamento da saída (transformar número no texto correspondente)

Recebemos (A) 1010 mas queremos 01000001
Recebemos (3) 0011 mas queremos 00110011
Recebemos (2) 0010 mas queremos 00110010
Recebemos (B) 1011 mas queremos 01000010

Para os números: adicionar os 4 bits mais significativos:

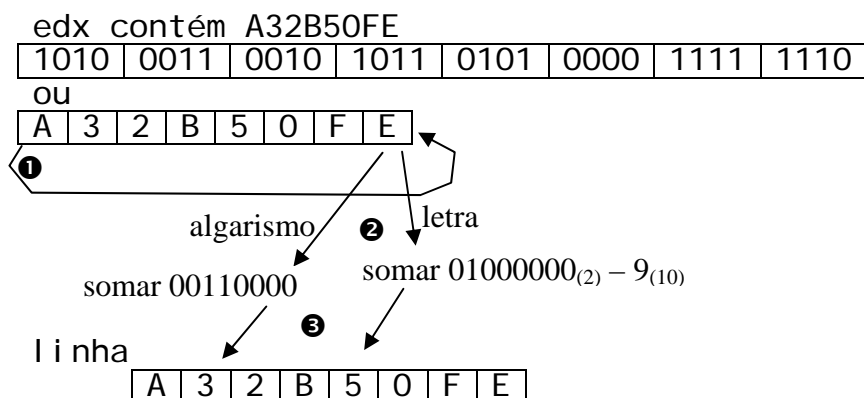
Recebemos (3) 0011 → 00110011
Recebemos (2) 0010 → 00110010

Para as letras: subtrair 9 e adicionar os 4 bits mais significativos:

Recebemos (A) 1010₍₂₎ - 9₍₁₀₎ = 0001₍₂₎ → 01000001
Recebemos (B) 1011₍₂₎ - 9₍₁₀₎ = 0010₍₂₎ → 01000010

Programa em Assembly para efectuar esta conversão:

Cada 4 bits do edx vão dar origem a uma letra/símbolo:



```
section .data
num      dd    0xA32B50FE
```

```
section .bss
linha    resb 9
```

```
section .text
```

```
    mov     eax, ds
    mov     es, eax
    ...
    mov     edi, linha
    cld
    mov     edx, [num]
    mov     ecx, 8
converte2:
    rol     edx, 4
    mov     eax, edx
    and     eax, 0xF
    cmp     eax, 9
    jbe     algarismo2
    sub     eax, 9
    add     eax, 0x40
    jmp     proximo
algarismo2:
    add     eax, 0x30
proximo:
    stosb
    loop    converte2

    ; imprimir string
    ; sair do programa
```

Funções em ASM

Convenções de chamada de funções e passagem de parâmetros

Supor a seguinte função em C, e respectiva chamada:

```
int soma (int x, int y) {  
    return x + y;  
}
```

...

```
res = soma(n1, n2);
```

A função soma em C é compilada para o seguinte código máquina:

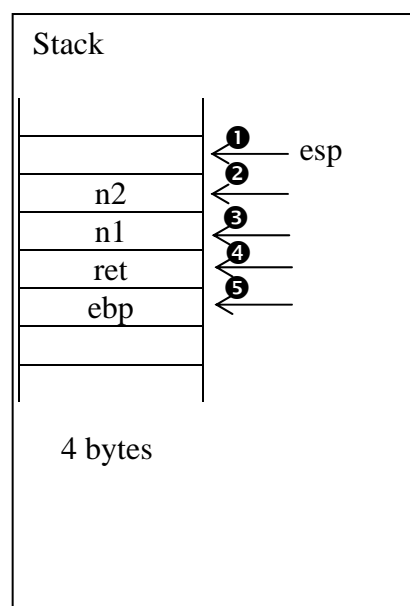
```
n1    dd    5  
n2    dd    3
```

soma:

```
push ebp          ; esp=pos. 5  
mov  ebp, esp  
; Vou buscar o n1  
mov  eax, dword[ebp+8]  
; Vou buscar o n2  
add  eax, dword[ebp+12]  
pop  ebp          ; esp=pos. 4  
ret              ; esp=pos. 3
```

Os parâmetros da função (n1, n2) são passados por stack

```
push  dword[n2]    ; esp=pos. 2  
push  dword[n1]    ; esp=pos. 3  
call  soma         ; esp=pos. 4  
add   esp, 8       ; esp=pos. 1  
mov   [res], eax
```



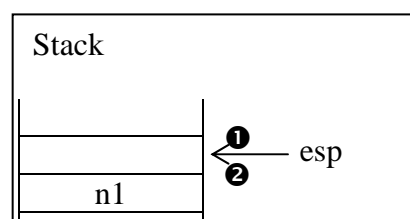
Supor a seguinte função em Pascal, e respectiva chamada:

```
function soma (x, y: integer): integer  
begin  
    soma := x + y;  
end;
```

...

```
res := soma(n1, n2);
```

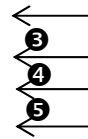
A função soma em Pascal é compilada para o seguinte código máquina:



```
n1    dd    5
n2    dd    3
```

soma:

```
    push ebp                ; esp=pos. 5
    mov  ebp, esp
    ; Vou buscar o n1
    mov  eax, dword[ebp+12]
    ; Vou buscar o n2
    add  eax, dword[ebp+8]
    pop  ebp                ; esp=pos. 4
    ret  8                  ; esp=pos. 1
```



4 bytes

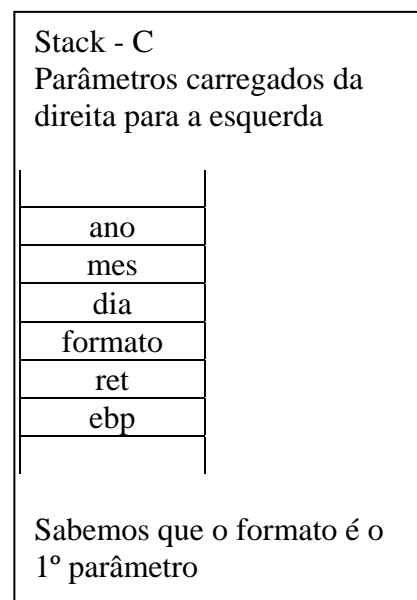
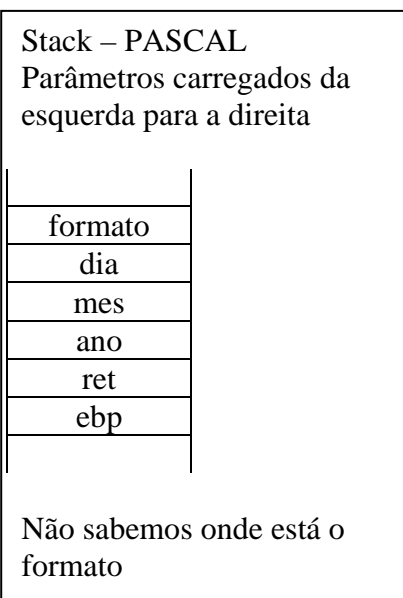
Os parâmetros da função (n1, n2) são passados por stack

```
    push dword[n1]          ; esp=pos. 2
    push dword[n2]          ; esp=pos. 3
    call soma               ; esp=pos. 4
    mov  [res], eax
```

Funções em C com parâmetros variáveis:

```
int printf(char * formato, ...);
```

```
printf("%d-%d-%d", dia, mes, ano);
```



Multiplicação a 4 bits no circuito multiplicador

Multiplicando: 1011

Multiplicador: 0101

A	Q	Cont	M	Comentários
0000	010 1	0	1011	Q(0)=0? Não → A=A+M
1011	0101	0		Right Shift (A, Q)
0101	1010	0		Cont=3? Não → Cont++
0101	101 0	1		Q(0)=0? Sim → Right Shift (A, Q)
0010	1101	1		Cont=3? Não → Cont++
0010	110 1	2		Q(0)=0? Não → A=A+M
1101	1101	2		Right Shift (A, Q)
0110	1110	2		Cont=3? Não → Cont++
0110	111 0	3		Q(0)=0? Sim → Right Shift (A, Q)
0011	0111	3		Cont=3? Sim → FIM
0011	0111			

Cálculos auxiliares:

$$\begin{array}{r}
 \text{M} \quad \quad \quad 1 \ 0 \ 1 \ 1 \quad (2) \\
 \text{A} \quad \quad \quad + \ 0 \ 0 \ 1 \ 0 \quad (2) \\
 \hline
 \text{A+M} \quad \quad 1 \ 1 \ 0 \ 1 \quad (2)
 \end{array}$$

Exercício: Multiplicar os seguintes números a 4 bits

Multiplicando: 1001

Multiplicador: 0110

A	Q	Cont	M	Comentários
0000	011 0	0	1001	Q(0)=0? Sim → Right Shift (A, Q)
0000	0011	0		Cont=3? Não → Cont++
0000	001 1	1		Q(0)=0? Não → A=A+M
1001	0011	1		Right Shift (A, Q)
0100	1001	1		Cont=3? Não → Cont++
0100	100 1	2		Q(0)=0? Não → A=A+M
1101	1001	2		Right Shift (A, Q)
0110	1100	2		Cont=3? Não → Cont++
0110	110 0	3		Q(0)=0? Sim → Right Shift (A, Q)
0011	0110	3		Cont=3? Sim → FIM
0011	0110	← Resultado final		

Representação de números negativos

Os números negativos podem ser representados segundo um dos formatos seguintes (entre outros que veremos à frente):

- Sinal e módulo
- Complemento para 1
- Complemento para 2

Sinal e módulo

O bit mais significativo é o sinal:

- 0 representa o sinal +
- 1 representa o sinal -

Exemplo com números de 4 bits:

- 0101 é o número +5, o primeiro 0 é o sinal (+) e 101 vale 5 no sistema binário de contagem natural
- 1101 é o número -5, o primeiro 1 é o sinal (-) e 101 vale 5 no sistema binário de contagem natural

Problemas:

00 tem duas representações: 0000 e 1000, ou seja (+0 e -0)

Complemento para 1

Os negativos obtêm-se a partir dos positivos, trocando os bits um a um (de 0 para 1, e de 1 para 0).

Exemplos:

- 0101 é positivo e vale +5
- 1010 é negativo e vale -5
- 01110011 é positivo e vale +115 (sistema binário de contagem natural)
- 10001100 é negativo e vale -115

Quando o bit mais significativo é 0, o número é positivo.
Quando o bit mais significativo é 1, o número é negativo.

Exercício:

Quanto vale o número, de 8 bits, 11100110, que está em complemento para 1?

Resolução:

O número 11100110 é negativo, logo vamos obter o seu simétrico: 00011001, que é positivo e vale $25_{(10)}$.
Sendo assim, 11100110 vale $-25_{(10)}$.

Problemas:

0 0 tem duas representações: 0000 e 1111, ou seja (+0 e -0)

Desvantagens:

Ao somar-se ou subtrair-se números em C2, o resultado NÃO vem logo correcto: é necessário corrigi-lo somando o Carry (vai -um).

Ex.: Somar -2 com -4

Este 1 (carry) tem que ser somado ao resultado.	1101	-2 em C1
	+1011	-4 em C1
	1000	
	+ 1	Soma o carry
	1001	-6 em C1

Complemento para 2

O complemento para 2, de um número, obtém-se complementando o número para 1 e somando-lhe 1.

Quando o bit mais significativo é 0, o número é positivo.
Quando o bit mais significativo é 1, o número é negativo.

Exemplo, com 4 bits:

O número 0101 é positivo e vale 5.
O seu complemento para 2 é:

0101	5 ₍₂₎
1010	C1(5)
+ 1	
1011	C2(5), ou seja, -5

Exercício:

Sendo 1011 um número negativo em complemento para 2, indique quanto vale (sugestão: obtenha o seu simétrico).

1011	
0100	C1
+ 1	
0101	C2(1011)

Resposta:

Se o complemento para 2 de 1011 é 0101, que é positivo e vale 5, então 1011 é negativo e vale -5.

Vantagens:

0 0 só tem uma representação. Prova:

Este 1 perde-se porque os números têm apenas 4 bits.	0000	
	1111	C1
	+ 1	
	0000	C2(1011)

Ou seja, o simétrico de 0000, em complemento para 2, é 0000.

Vantagens (2):

Ao somar-se ou subtrair-se números em C2, o resultado vem logo correcto sem ser necessária qualquer correcção.

Ex.: Somar -7 com 3

$$\begin{array}{r} 1001 \\ +0011 \\ \hline 1100 \end{array} \quad \begin{array}{l} -7 \text{ em C2} \\ 3 \\ -4 \text{ em C2} \end{array}$$

Ex.: Somar -2 com -4

Este 1 perde-se porque os números têm apenas 4 bits.

$$\begin{array}{r} 1110 \\ +1100 \\ \hline 1010 \end{array} \quad \begin{array}{l} -2 \text{ em C2} \\ -4 \text{ em C2} \\ -6 \text{ em C2} \end{array}$$

Tabela-resumo

(Exemplo para números de 4 bits)

Deci mal	SM	C1	C2
-8	-	-	1000
-7	1111	1000	1001
-6	1110	1001	1010
-5	1101	1010	1011
-4	1100	1011	1100
-3	1011	1100	1101
-2	1010	1101	1110
-1	1001	1110	1111
0	0000	0000	0000
	1000	1111	
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	0101	0101
6	0110	0110	0110
7	0111	0111	0111

Exercí ci o:

Consi dere o seguinte número em C2: 10001001.

Represente-o no formato de sinal e módul o.

$$\begin{array}{r} 10001001 \\ 01110110 \quad C1 \\ + \quad 1 \\ \hline 01110111 \quad C2 \end{array}$$

O simétrico de 10001001 (negativo) é 01110111 (posi ti vo). Logo, a representação de 10001001 em SM é 11110111.

Divisão a 4 bits no circuito divisor

Dividendo: 00110100

Divisor: 0101

A	Q	Cont	M	Comentários
0011	0100	0	0101	LS(A, Q)
0110	100	0		A=A-M
0001	100	0		A<0? Não: Q(0)=1
0001	1001	0		Cont=3? N: Cont++
0001	1001	1		LS(A, Q)
0011	001	1		A=A-M
1110	001	1		A<0? Sim: Q(0)=0
1110	0010	1		A=A+M
0011	0010	1		Cont=3? N: Cont++
0011	0010	2		LS(A, Q)
0110	010	2		A=A-M
0001	010	2		A<0? Não: Q(0)=1
0001	0101	2		Cont=3? N: Cont++
0001	0101	3		LS(A, Q)
0010	101	3		A=A-M
1101	101	3		A<0? Sim: Q(0)=0
1101	1010	3		A=A+M
0010	1010	3		Cont=3? S: FIM

Cálculos auxiliares:

A		0	1	1	0	(2)
M	-	0	1	0	1	(2)
A-M		0	0	0	1	(2)
A		0	0	1	1	(2)
M	-	0	1	0	1	(2)
A-M	1	1	1	1	0	(2)
A		1	1	1	0	(2)
M	+	0	1	0	1	(2)
A-M	1	0	0	1	1	(2)
A		0	0	1	0	(2)
M	-	0	1	0	1	(2)
A-M	1	1	1	0	1	(2)

1. Apresentação

Pretende-se realizar um programa escrito em linguagem ASSEMBLY 386 que efectue as quatro operações aritméticas básicas sobre dois operandos inteiros, com n bits de comprimento, em que $32 \leq n \leq 256$, correspondendo a um número de **doublewords** entre 1 e 8.

Os negativos são representados no formato de complemento para 2.

2. Estrutura

No início o programa deverá pedir ao utilizador o número de **doublewords** com que os operandos irão ser representados (entre 1 e 8).

Depois, o programa deverá aceitar os 2 operandos (no formato anteriormente apresentado, **em hexadecimal**) e o operador (+, -, *, /), a partir do teclado, sendo o resultado mostrado no ecrã (monitor) no mesmo formato.

O programa deverá detectar e indicar no ecrã situações de "overflow".

O programa poderá ter uma estrutura livre assim como a interface homem-máquina. No entanto aconselha-se os alunos a fazer a introdução dos dados e operador em Polish Reverse Notation (Notação Polaca). Exemplo de soma de 2 inteiros em notação polaca:

23 <ENTER>	(1º operando)
15 <ENTER>	(2º operando)
+ <ENTER>	(operador)
38	(resultado)

3. Grupos

O trabalho pode ser realizado em grupos de 3 alunos no máximo.

4. Relatório

Do relatório deve constar a listagem do programa, o modo como foi estruturado (fluxograma) e uma explicação simplificada da forma como os operandos são decompostos e tratados para se obterem os resultados.

5. Data de entrega e discussão

Data de entrega e discussão: dia marcado para a realização da frequência desta cadeira.