Fundamentos de programação em *assembly*

João Canas Ferreira

Fevereiro 2014

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Assuntos

Organização básica de um microcomputador

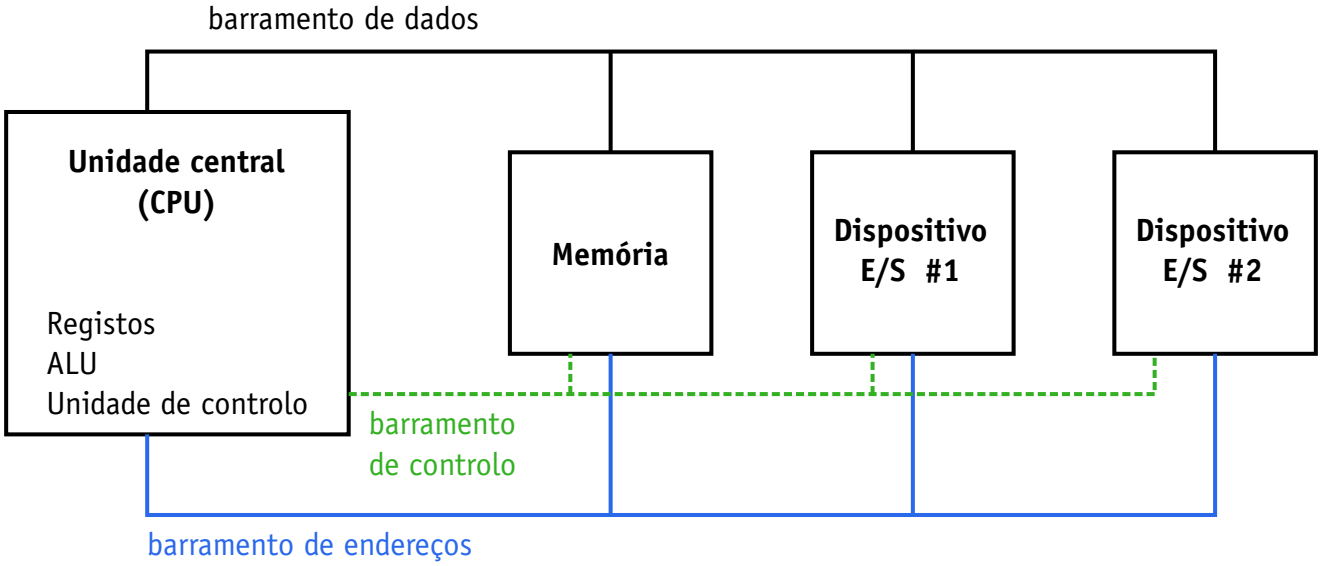
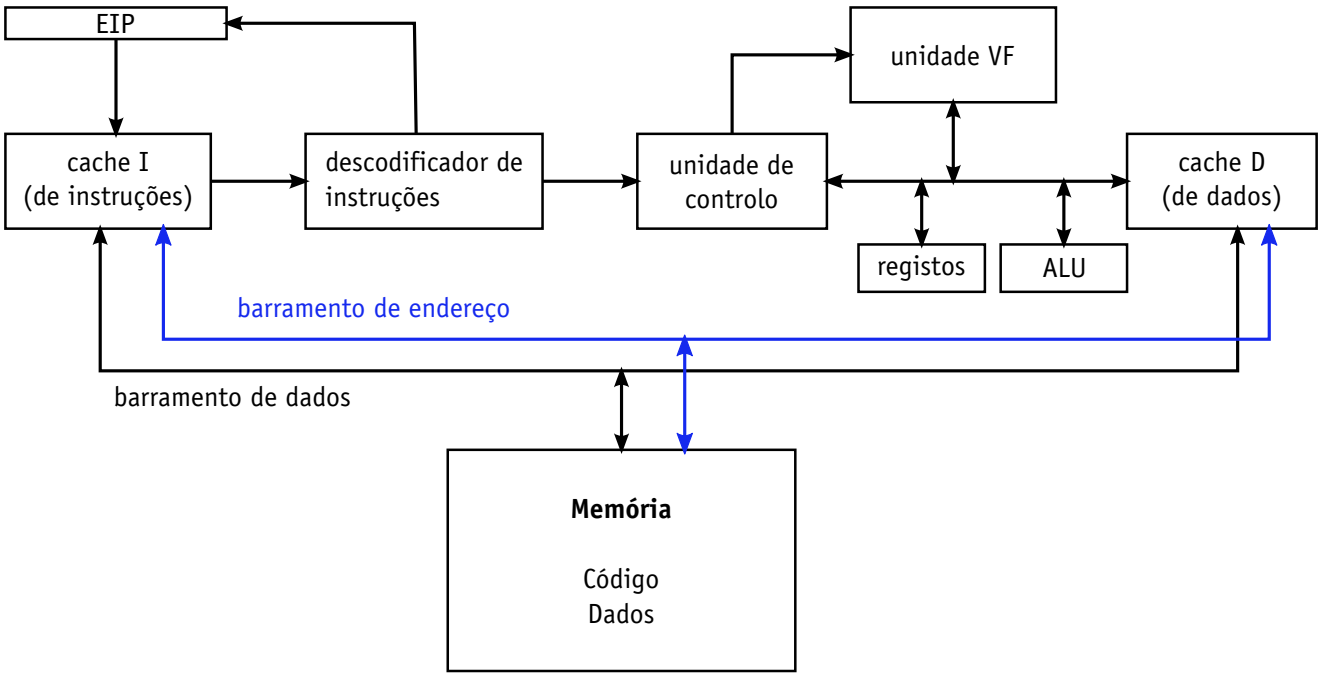
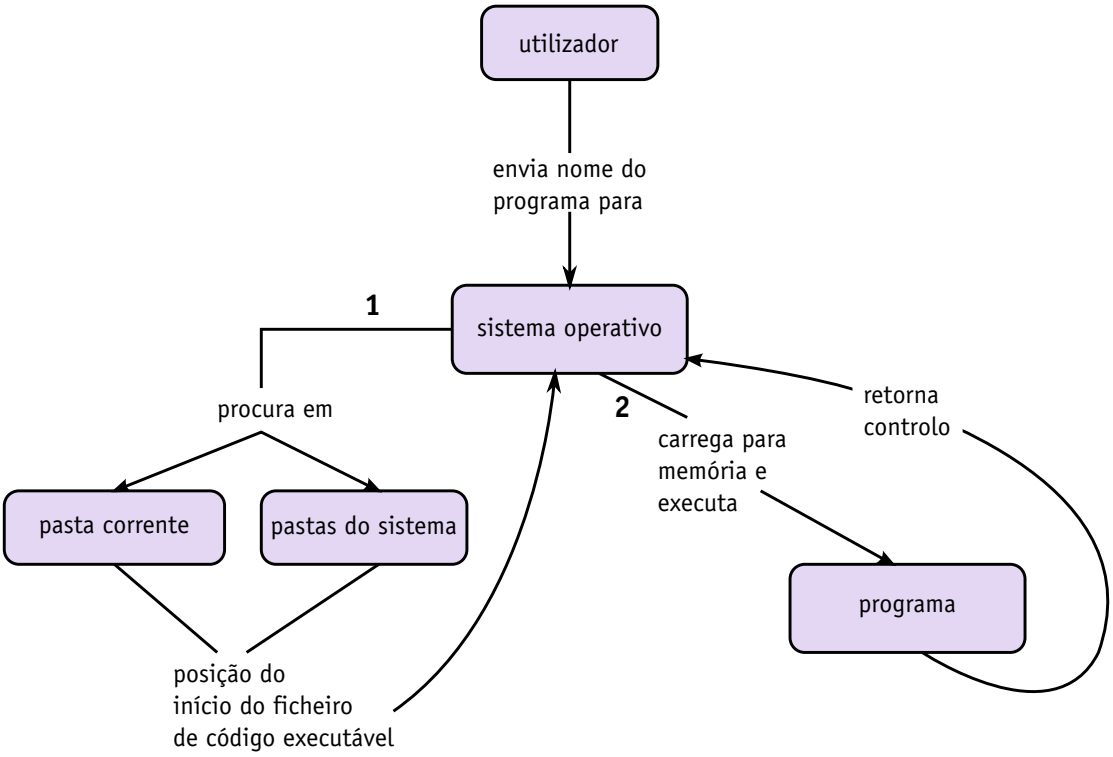


Diagrama de blocos simplificado de um CPU



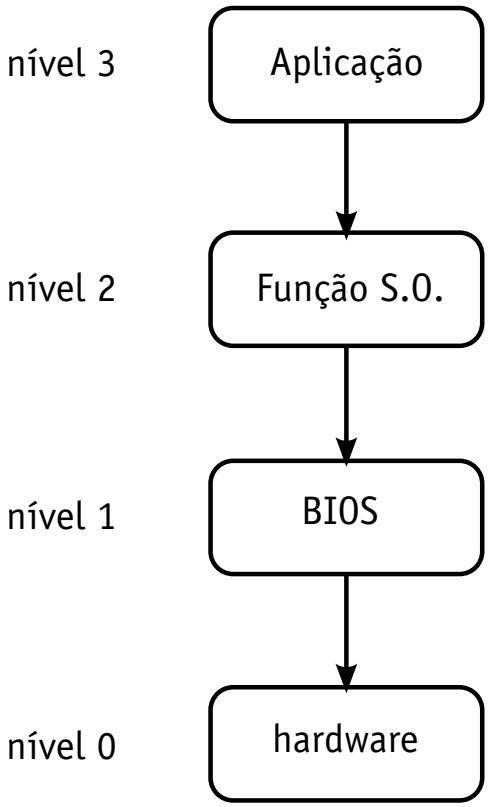
Execução de um programa



Sistema operativo multitarefa

- Windows e Linux são sistemas operativos *multitarefa*:
  - O sistema operativo pode ter múltiplos programas em execução simultânea.
  - Processo = programa em execução
  - Escalonador atribui a cada processo uma parte do tempo de CPU.
  - Troca rápida entre processos cria ilusão de execução simultânea.
  - Processador deve fornecer suporte para troca rápida entre processos.
  - Processador com N núcleos pode ter N processos em execução *realmente* simultânea.
  - Num sistema que executa M processos (M>N), os M processo devem ser "distribuídos" pelos N núcleos.
  - Cada núcleo troca rapidamente entre os processos que lhe estão atribuídos.

Operações de entrada/saída



- Nível 3: Funções de linguagens de alto nível (C,Scheme); portáveis, convenientes, nem sempre rápidas. Exemplo: `printf, scanf`.
- Nível 2: Interface do S.O.: *Application Programming Interface (API)*; mais capacidades, muitos detalhes. Exemplo: `ExitProcess`.
- Nível 1: BIOS. Controladores em software *(drivers)* que comunicam diretamente com dispositivos. S.O. geralmente impede o acesso direto a este nível (segurança).
- Linguagem assembly permite recorrer aos serviços de qualquer nível.

Assuntos

Modos de funcionamento

Processadores IA-32 têm três modos básicos de funcionamento:

Protegido Modo "nativo" de 32 bits, com todas as instruções disponíveis. Processos usam áreas de memória separadas e o CPU impede acessos fora da área atribuída. Um processo não pode interferir com outro. Usa endereços de 32 bits (4 GB).

- **Virtual-8086** Permite executar programas de modo real em modo protegido (com restrições que impedem um programa de afetar outros).

Real Implementa o ambiente de programação do processador 8086 (com alguns extras). Permite controlo total do hardware. Usa endereços de 20 bits (1 MB). No arranque, o CPU está em modo real.

Gestão de sistema Modo especial para implementação de funções de sistema (gestão de consumo, segurança, etc). Usado para customizar o sistema pelos fabricantes.

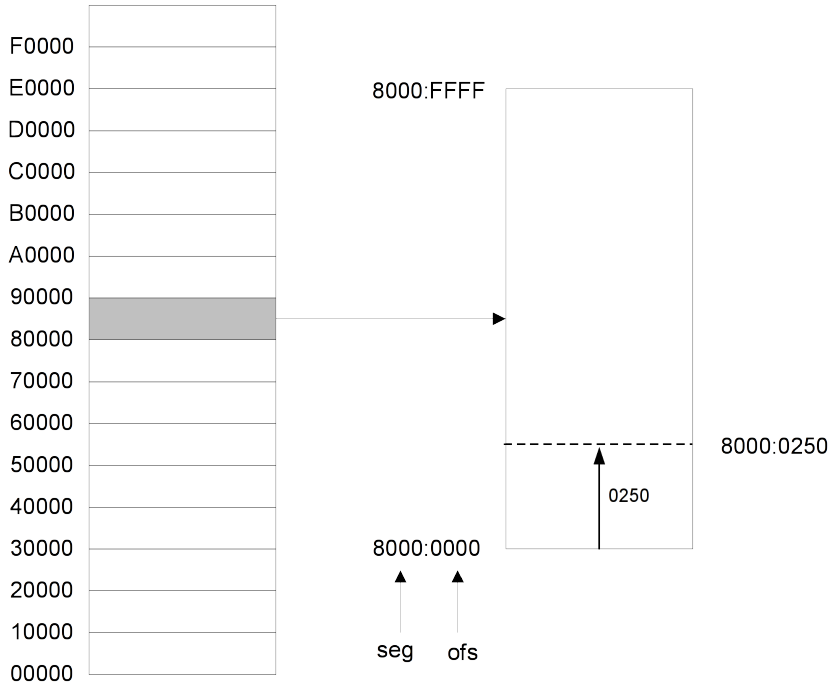
Endereçamento em modo real

- Gama de endereços: 0-FFFFFh
- Problema: Registos de 16 bits
- Solução: Memória segmentada em zonas de 64 KB (segmentos)
- Endereço tem 2 partes: segmento (16 bits) e deslocamento (16 bits)
- Representação segmento:deslocamento 8000:1234
- O segmento é especificado em registos especiais
- O deslocamento é incluído nas instruções
- Segmentos principais: código (registo CS), dados (DS), pilha (SS).

Cálculo do endereço linear (em hexadecimal):

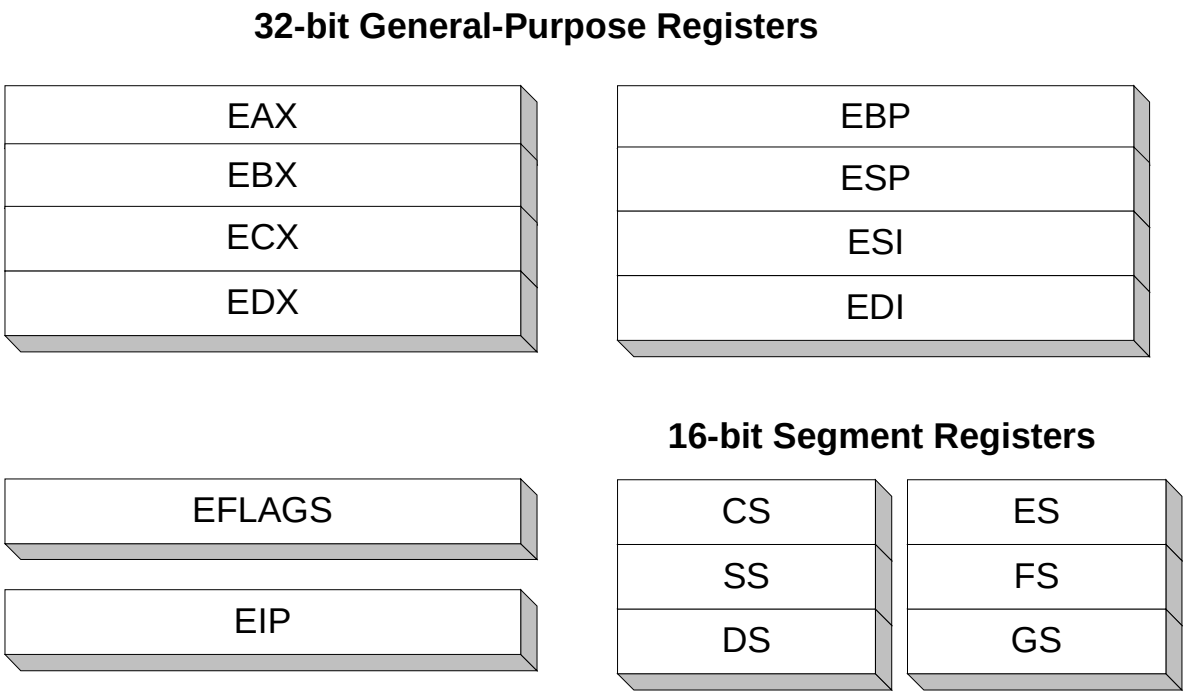
Segmento:	8 0 0 0
Deslocamento:	+ 1 2 3 4
Resultado:	8 1 2 3 4

Segmentos



Fonte: [Irvine10]

Registos

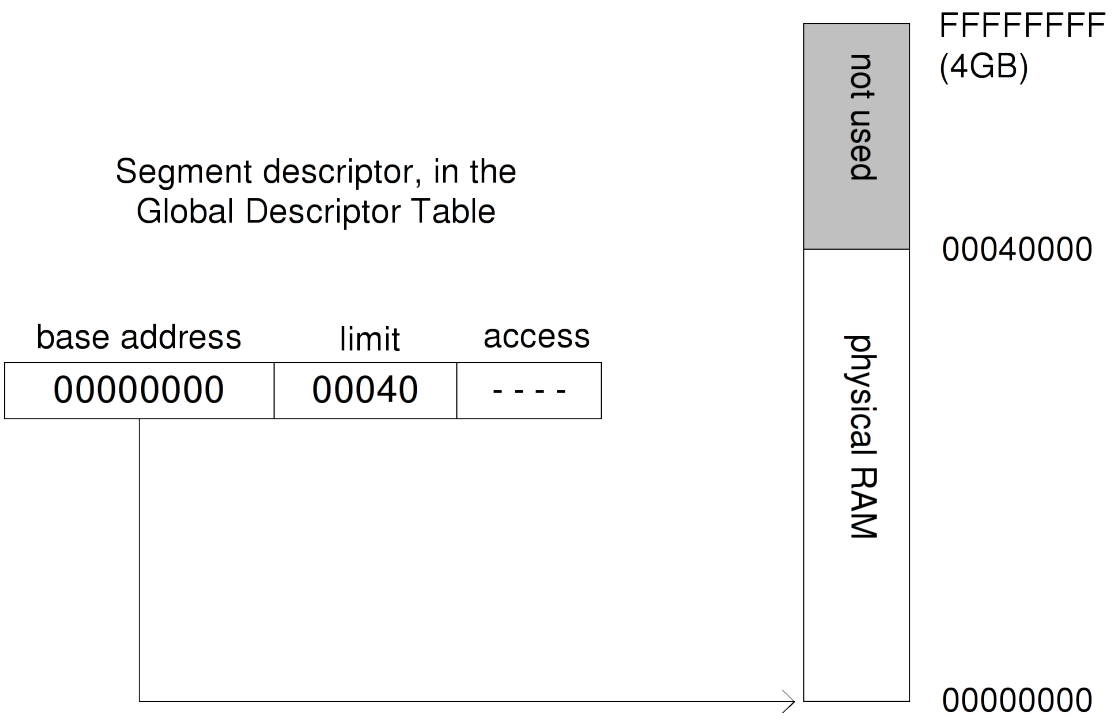


Fonte: [Irvine10]

Endereçamento em modo protegido (32 bits)

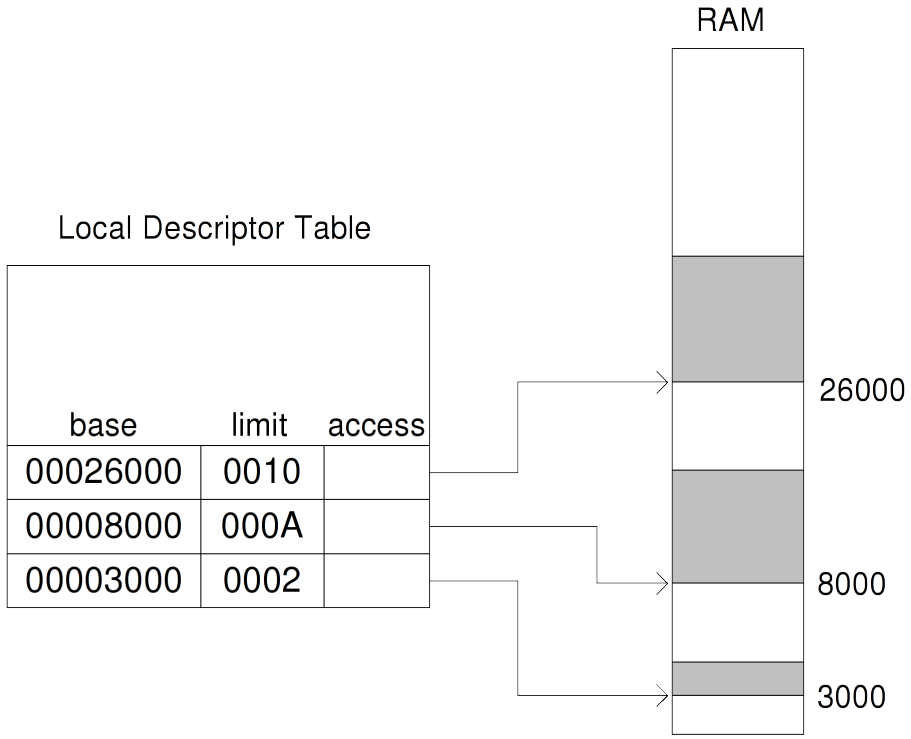
- Segmentos podem ter 4 GiB
- Registos de segmentos contêm índice para uma tabela de descritores
- Dois modelos:
  - *Flat:* (liso) Todos os segmentos têm 4 GiB e estão "sobrepostos"
    É este o modelo que usamos: espaço único de 4 GiB
    Usa uma tabela de descritores globais (GDT).
  - *Multi-segmento:* Programa é composto por segmentos separados (tabela de descritores locais—LDT) [pouco usado]
- Cada descritor especifica:
  1. endereço base
  2. limite (a multiplicar por 1000h [= 4096d])
  3. modos de acesso (leitura, leitura/escrita, permissões de acesso)

Modelo com espaço de endereçamento liso



Fonte: [Irvine10]

Modelo com múltiplos segmentos



Fonte: [Irvine10]

Elementos lexicais

- Constantes (números inteiros)
- Expressões de números inteiros
- Carateres e cadeias de carateres constantes
- Identificadores e palavras reservadas
- Diretivas e instruções
- Etiquetas
- Mnemónicas e operandos
- Comentários

Constantes e expressões constantes

- Bases: binário (b), decimal (d), hexadecimal (h), octal (o)
- Exemplos: 30d, 64h, 1101b, 0a2h
- Expressões com valores inteiros: constantes e operadores

Operator	Name	Precedence Level
( )	parentheses	1
+,-	unary plus, minus	2
*,/	multiply, divide	3
MOD	modulus	3
+,-	add, subtract	4

Expression	Value
16 / 5	3
-(3 + 4) * (6 - 1)	-35
-3 + 4 * 6 - 1	20
25 mod 3	1

Fonte: [Irvine10]

- Carateres: "A" ou 'x' (1 byte)
- Cadeias (sequências): "Viva!" ou 'Viva!' (5 bytes)

Identificadores

- Formato de identificadores
  - 1-247 carateres
  - primeira caráter pode ser letra, _ , ?, $ ou @ (evitar)
  - carateres seguintes também podem incluir dígitos

rotina _Nome123 $rotina @val

- Identificadores são usados para
  - variáveis
  - etiquetas
  - rotinas e argumentos de rotinas
  - macro-procedimentos (não usados em MPCP)
- Palavras reservadas não podem ser usadas como identificadores:
  - mnemónicas (nomes de instruções: `ADD`, `SUB`, `JMP`)
  - diretivas (ex., .data, .code)
  - tipos (ex., dword, DWORD, byte)
  - operadores (ex., type, lengthof)

Etiquetas

- Etiquetas servem para identificar posições em memória
  - Correspondem a endereços (deslocamento) de dados ou instruções
- Etiquetas seguem as regras dos identificadores
- Para dados:
  - não podem ser repetidas
  - não terminam com dois pontos :
- Em código:
  - indicam o destino dos saltos
  - terminam com dois pontos :

Diretivas e instruções

Diretivas

- Diretivas são comandos para o programa *assembler*
  - Não fazem parte do conjunto de instruções
  - São usadas para informação adicional: declarar rotinas, definir áreas de código ou dados, selecionar o modelo de memória, etc.
  - Variam de *assembler* para *assembler*

Instruções

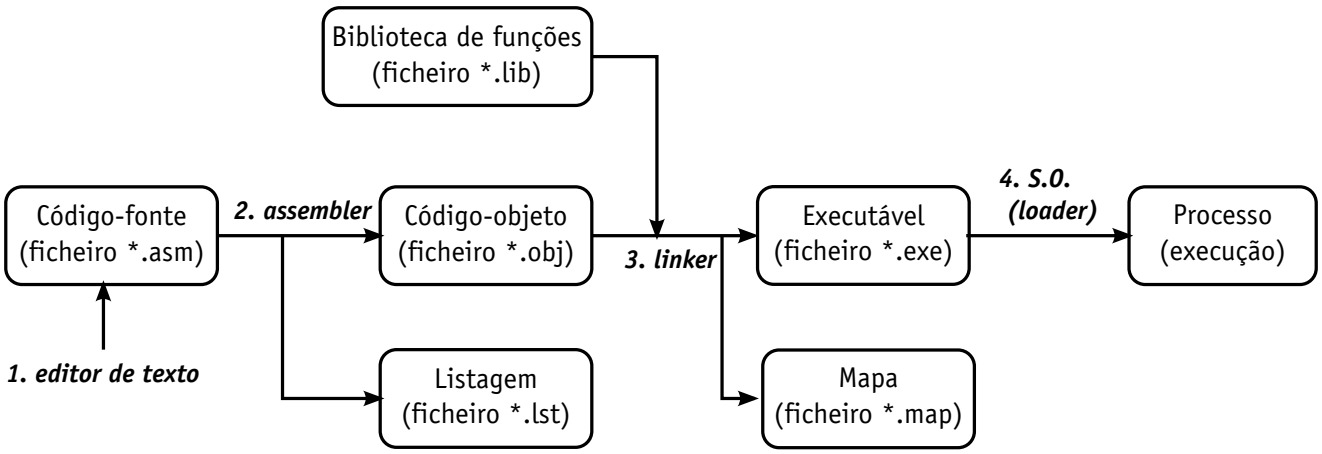
- Instruções são convertidas em código-máquina
- Executadas pelo CPU (em tempo de execução)
- Uma instrução contém:
  - etiqueta (opcional)
  - mnemónica (obrigatória)
  - operando(s) (dependendo da instrução)
  - comentário (opcional, começa por ponto e vírgula)
- Operandos são:
  - constantes ou expressões constantes
  - registos
  - posições de memória (incluindo etiquetas de dados)

Um programa completo

```
include mpcp.inc
.data
valores   DWORD 10, 15, 20
msg       BYTE   "Total = %d", 13,10 ,0     ; CR+LF

.code
main:     xor    eax, eax   ;inicializar a 0
          mov    ecx, 3     ; contador
          mov    edi, offset valores    ; endereço
ciclo:    jecxz  fim     ; saltar se ecx=0
          add    eax, [edi]     ; adicionar valor apontado por EDI
          add    edi, 4
          dec    ecx     ; atualizar contador
          jmp    ciclo     ; repetir
fim:      invoke printf, offset msg, eax      ; chamar sub-rotina
          invoke _getch     ; apenas em Windows
          invoke ExitProcess, 0     ; apenas em Windows
end main
```

Etapas da produção de um programa



- Sempre que o código-fonte é alterado, repetir passos 2–4
- **Listing file** Listagem detalhada do código gerado
- **Map file** Listagem com informação sobre utilização de memória
- **Object file** Ficheiro com código-objecto (código-máquina)
- **Executable file** Ficheiro com programa executável
- **Passo 4** Execução do programa

Listagem do programa

```
00000000                    .data
00000000  0000000A          valores DWORD 10, 15, 20
          0000000F
          00000014
          ...
00000000                    .code
00000000  33 C0             start:  xor eax, eax
00000002  B9 00000003       mov ecx, 3
00000007  BF 00000000       mov edi, offset valores
0000000C  E3 08             ciclo:  jecxz fim
0000000E  03 07             add eax, [edi]
00000010  83 C7 04          add edi, 4
00000013  49                dec ecx
00000014  EB F6             jmp ciclo
```

Definição de dados

- Tipos intrínsecos (pré-definidos) de dados
  - BYTE, SBYTE
    8-bit unsigned integer; 8-bit signed integer
  - WORD, SWORD
    16-bit unsigned, signed integer
  - DWORD, SDWORD
    32-bit unsigned, signed integer
- A definição de dados *reserva memória* e associa-lhe uma etiqueta.
  - Formato: etiqueta diretiva valor_inicial, valor_inicial, ...
  - Exemplo: `val1 BYTE 10`
  - Todos os valores iniciais são convertidos para dados em binário
- Sequências (vetores) de valores
  - `lista BYTE 10, 21, 11, 15`
  - `lista2 BYTE 10 DUP (50)` 10 elementos inicializados a 50
  - `lista3 BYTE 10 DUP (?)` 10 elementos não-inicializados

Exemplos de definição de dados

As definições de dados globais são colocadas em secções `.data`

- Carateres são bytes:
  `byte1 BYTE 'A'` ; valor do código ASCII
- Também se pode usar DB (Define Byte):
  `byte1 DB 'A'`
- Cadeia de carateres:
  `pergunta BYTE 'Nome?'` ; sequência de 5 bytes
- Cadeia de carateres terminada por 0:
  `mensagem BYTE 'Viva!',0` ; sequência de 6 bytes
- `word2 SWORD -32768` ; inteiro 16 bits, com sinal
- `val4 SDWORD -3,-2,-1,0,1` ; seq. de inteiros 32 bits, com sinal

Definições de dados não inicializados podem ir para secções `.data?`
Vantagem: O ficheiro executável final fica mais pequeno (porque não inclui os valores iniciais).

Utilização de etiquetas de dados

A etiqueta representa o endereço da primeira posição de memória atribuída ao valor.

```
            .data
val1        DWORD 50
val2        DWORD 100, 101
            .code
            ...
            mov    eax, val1          ; EAX=50
            mov    ebx, [val1]        ; EBX=50
            mov    ecx, val2          ; ECX=100
            mov    edx, [val2+4]      ; EDX=101
            mov    edx, val2[4]       ; o mesmo
            mov    edx, 4[val2]       ; o mesmo
            mov    edx, 4+val2        ; o mesmo
```

Uma etiqueta não é uma variável!

Operadores para dados

- Operador OFFSET: distância de uma etiqueta ao início do segmento
- Operador TYPE: tamanho (em bytes) de um elemento da definição
- Operador LENGTHOF: número de elementos de uma definição de dados
- Operador SIZEOF: número de bytes reservados para os dados
  - equivalente a multiplicar LENGTHOF por TYPE

```
            .data
val1        DWORD 150, 160, 170
            .code
            mov    edi, OFFSET val1
            mov    esi, [edi]         ; ESI=150
            mov    ecx, LENGTHOF val1 ; ECX=3
            mov    edx, SIZEOF val1   ; EDX=12
            mov    ebx, TYPE val1     ; EBX=4
```

Definição de constantes simbólicas

- Utilização de constantes simbólicas torna programas mais legíveis
- Também facilita a realização de alterações
- Constantes numéricas com a diretiva **=**: `Tamanho = 100`
- Também pode ser usado a diretiva EQU: `Tamanho EQU 100`
- Definições feitas com **=** podem ser redefinidas.

```
NumElem   = 10
ValInit   = 1
          .data
val1      DWORD NumElem dup(0)
          .code
          mov    edi, OFFSET val1
          mov    ecx, NumElem
ciclo:    jecxz  fim
          mov    dword ptr [edi], ValInit
          dec    ecx
          add    edi, 4
          jmp    ciclo
fim:      ...
```


Algumas funções disponíveis

- As seguintes funções devem ser usadas com `invoke`.
- A declaração das funções está feita no ficheiro **`mpcp.inc`**

- **printf, formato, arg2,...**
  imprime a cadeia de carateres **formato** (especificada pelo seu endereço) substituindo cada subsequência iniciada por **%**
  - %d: número inteiro em decimal; %x: em hexadecimal (dword)
  - %s: cadeia de carateres (endereço – dword)
  - %c: caráter (byte)
- **scanf, formato, arg2,...**
  lê dados especificados pela cadeia de carateres **formato** (como em printf) e coloca-os nas posições indicadas pelos argumentos `arg2,...`, que devem representar endereços
- **_getch** espera por uma tecla
- **ExitProcess, code** termina o programa
  (especifica código de retorno: 0=sucesso)

- Mais documentação sobre funções disponíveis: `http://goo.gl/bPb10`
`http://www.cplusplus.com/reference/cstdlib/`

Referências

Irvine10 Kip Irvine
Assembly Language for x86 Processors (6th edition)
Pearson Education, 2010