Universidade do Minho
Escola de Engenharia

MIETI :: Métodos de Programação II

2015/16

Práticas Laboratoriais

Módulo 1 (v1)

António Esteves

Fevereiro 2016

Execução de programas num computador ***(1)***

Níveis de abstração:

- nível das linguagens HLL (***H****igh* ***L****evel* ***L****anguages*)
  - É o nível das linguagens convencionais de programação.
  - Em formato **texto**.
  - Exemplos:
    - **imperativas e OO** (Basic, Fortran, C, C++, Java, ...).
    - **funcionais** (Lisp, Haskell, ...).
    - **lógicas** (Prolog, ...).
- nível da linguagem *assembly*
  - É uma linguagem **intermédia.**
  - Representa os comandos do CPU em formato **texto.**
- nível da linguagem máquina
  - É uma linguagem de **comandos.**
  - É **específica** de cada CPU ou duma família de CPU's.
  - Em formato **binário.**

Execução de programas num computador (2)

```
int t = x+y;
```

- **Código C**
  - somar 2 inteiros *x* e *y* e guardar resultado em *t*

```
addl 8(%ebp),%eax
```

- ***Código assembly***
  - somar 2 inteiros de 4-*bytes*
  - idêntico à expressão *x+=y*
  - operandos estão:
    - x: no registo EAX
    - y: na memória (posição EBP+8)

```
0x401046: 03 45 08
```

- **Código *objeto/máquina***
  - instrução com 3-*bytes*
  - na posição de memória 0x401046
  - partes dos bits indicam: que é uma soma, o código do EAX, o código do EBP, o valor 8

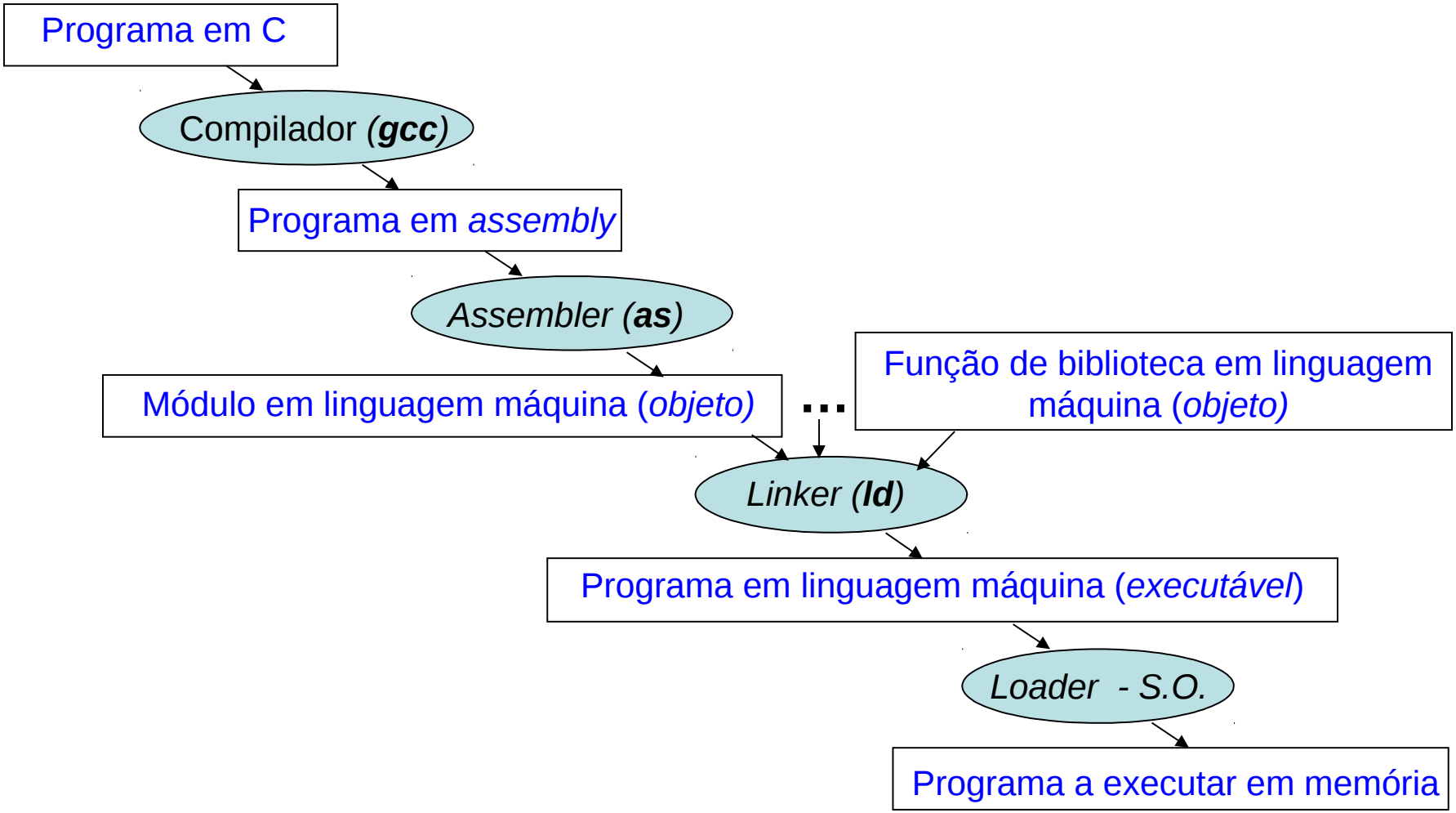
Execução de programas num computador (3)

Mecanismos de conversão (para comandos do CPU):

- **Compilador (gcc)**
  - traduz um programa de um nível de abstração para outro inferior
  - converte um ficheiro com código C (texto) para outro com instruções *assembly* (texto); as instruções são específicas dum tipo de CPU;
- ***Assembler (as)***
  - converte as instruções *assembly* para um ficheiro com código *objeto* (binário)
- ***Linker (ld)***
  - combina/liga um ou mais ficheiros objeto, gerados a partir do código fonte ou bibliotecas, num único ficheiro executável.
- O **gcc** pode ser instruído para efetuar todos os passo da conversão usando um único comando: compilar, gerar código objeto e gerar código executável.
- **Interpretador**
  - analisa, uma a uma, as instruções de um programa em HLL, e:
    - **gera código** em linguagem máquina para essa instrução, **e**
    - **executa** esse código.

Execução de programas num computador (4)

Passos envolvidos na passagem de um programa numa HLL até à sua execução:



Execução de programas num computador (5)

Exemplo – Crie dois ficheiros C com o seguinte conteúdo

main.c

```
#include <stdio.h>
void swap(int *);
int main() {
    int buf[2] = {11, 22};
    printf (“INICIO: [0] = %d [1] = %d\n”,buf[0],buf[1]);
    swap(buf);
    printf (“FINAL:  [0] = %d [1] = %d\n”,buf[0],buf[1]);
    return 0; }
```

swap.c

```
void swap(int *buf) {
    int temp;
    temp   = buf[1];
    buf[1] = buf[0];
    buf[0] = temp; }
```

Execução de programas num computador (6)

Converter este exemplo de C para executável em 3 etapas:

- Compilar o ficheiro main.c (swap.c) para ***assembly***; emite avisos durante a compilação e guarda o resultado no ficheiro main.s (swap.s)

  gcc -Wall -S main.c -o main.s

  gcc -Wall -S swap.c -o swap.s

- Converter os ficheiros *assembly* hello.s e swap.s para código **objeto** (ficheiros hello.o e swap.o)

  as main.s -o main.o

  as swap.s -o swap.o

- Efetuar o ***linking*** para produzir um ficheiro troca1 como o resultado da ligação dos ficheiros objeto main.o, swap.o e a biblioteca libc.a

  ld -o troca1 main.o swap.o -lc --entry main

- O ficheiro ./troca1 é executável?

Alternativa: Usar o **gcc** para efetuar as 3 etapas com um único comando (o **gcc** recorre indiretamente a **as** e **ld**)

gcc -Wall main.c swap.c -lc -o troca2

- O ficheiro ./troca2 é executável?

Tutorial sobre Code:Blocks

- Efetuar o tutorial sobre utilização do IDE Code:Blocks para compilar e depurar programas em C
- O enunciado do tutorial encontra-se na Blackboard
- O Code:Blocks pode ser instalado usando a aplicação **Ubuntu Software Center**, indo à seção **Developper Tools** e depois à subseção **IDEs**
- Em alternativa, pode descarregar-se o Code:Blocks a partir de: **http://www.codeblocks.org**

Exercício 1: Compilar e depurar o módulo Euclides das aulas teóricas (1)

- **Ficheiro com o código fonte euclid.c**

A função **mdc()** calcula o máximo divisor comum

```
int mdc ( int a, int b) {
    if (a<b) { /* se (a < b) e' preciso fazer a troca de a com b */
        int t = a; a = b; b = t;
    }
    While (b != 0) {
        int temp = a%b;
        a = b;
        b = temp;
    }
    return a;
}
```

Exercício 1: Compilar e depurar o módulo Euclides das aulas teóricas (2)

- **Ficheiro com o código fonte euclid.c** (continuação)

A função ext_euclid() implementa o algoritmo de Euclides estendido.
Este algoritmo calcula x e y tais que: a*x + b*y = mdc(a, b)

```
d0=a ,                  x0=1 ,      y0=0
d1=b ,                  x1=0 ,      y1=1
Enquanto(d1 ≠ 0) fazer
    q = ⌊d0/d1⌋
    d2 = d1 ,           x2=x1 ,     y2=y1
    d1 = d0-q*d1
    x1 = x0-q*x1
    y1 = y0-q*y1
    d0 = d2 ,           x0=x2 ,     y0=y2
fimEnquanto
Devolve {d,x,y} = {d0,x0,y0}
```

Exercício 1: Compilar e depurar o módulo Euclides das aulas teóricas *(3)*

- **Ficheiro *header* euclid.h**

Contém a assinatura das funções *mdc()* e *ext_euclid()*

```
/* Assegurar que incluímos o ficheiro header apenas uma vez */
#ifndef __EUCLID_H__
#define __EUCLID_H__
/* variáveis globais (definidas em euclid.c) */
extern int x, y;
/* calcular o mdc */
int mdc ( int a, int b);
/* calcular d = mdc(a,b) e resolver a equação ax+by=d */
int ext _euclid ( int a, int b, int *x, int *y);
#endif
```

***Exercício 1**: Compilar e depurar o módulo Euclides das aulas teóricas (4)*

- Implementar em C a função main(), com a funcionalidade descrita a seguir, e incluí-la no ficheiro main_euclid.c
- Vamos chamar as funções mdc() e ext_euclid(), definidas no ficheiro fonte euclid.c

- Incluir o ficheiro *header* euclid.h
- Ler da consola *(stdin)* os valores de **a** e **b**
- Calcular **d** = mdc(a,b) com o algoritmo de Euclides:

  d = mdc(a, b);
- Apresentar na consola *(stdout)* o valor de **d**
- Calcular **d**=mdc(a,b), **x** e **y** com algoritmo de Euclides estendido:

  d = ext_euclid(a, b, &x, &y);

  Os resultados ficam em d e nas variáveis x e y
- Apresentar na consola os valores de **d**, **x** e **y**.

Exercício 1: *Compilar e depurar o módulo Euclides das aulas teóricas* ***(5)***

- Compilar o ficheiro main_euclid.c
- Compilar o ficheiro euclid.c onde são definidas as funções mdc() e ext_euclid()
- Ligar os 2 ficheiros objeto gerados de modo a obter um único ficheiro executável:

  gcc -g -O0 -Wall main_euclid.c euclid.c -o main_euclid
- Depurar o programa no Code::Blocks e correr o executável na consola usando o comando:

  ./main_euclid

***Exercício 2**: programa com parâmetros*

- Escrever em C um programa com parâmetros que funciona como uma calculadora simples

- Escrever um programa com parâmetros significa que o executável que vamos obter a partir dele vai aceitar que lhe passemos valores quando o executarmos na consola

- Supondo que o executável obtido com a compilação do programa se chama calcula, pretende-se que a calculadora funcione executando o seguinte comando na consola:

  ./calcula op1 op op2

  - op1, op2 → são os valores (inteiros) envolvidos na operação
  - op → operação a efetuar sobre os valores (+,-,x,/)

***Exercício 3**: tipos de dados definidos pelo utilizador*

- Neste exercício vamos utilizar o *typedef* para definir 2 tipos de dados estruturados, correspondentes a um **ponto** e a uma **janela** no ecrã:
  - **Ponto** → uma estrutura contendo dois membros inteiros **x** e **y**
  - **Janela** → uma estrutura contendo como membros:
    - **ul** e **lr** do tipo Ponto, correspondentes aos cantos superior esquerdo e inferior direito;
    - **area** um float, correspondendo à área da janela.
- Escrever e testar um programa que use estes tipos de dados. O programa deve ter, além do main(), funções para:
  - Definir os valores x e y de um Ponto, com a seguinte assinatura:

    void defPonto(Ponto *P, int x, int y);
  - Calcular a área duma Janela:

    void areaJanela(Janela *J);
  - Definir os valores dos 2 cantos de uma Janela (ul e lr) e a área:

    void defJanela(Janela *J, Ponto pUL, Ponto pLR);
  - Deslocar uma Janela de um valor (dx, dy):

    void moverJanela(Janela *J, int dx, int dy);

***Exercício 4**: estruturas, acesso a ficheiros e alocação dinâmica de memória*

- Desenvolver um programa que permite ler os dados relativos aos empregados duma empresa, a partir dum ficheiro (lista.txt), para memória e depois visualizar os dados de um empregado à escolha do utilizador.

- O ficheiro lista.txt está em formato texto e possui a seguinte estrutura:

  - A primeira linha do ficheiro contém o número de empregados contidos no ficheiro.
  - As restantes linhas contêm os dados dos empregados, um por linha, com o seguinte formato:

    ```
    PrimeiroNome SegundoNome Idade
    ```

***Exercício 4**: estruturas, acesso a ficheiros, alocação dinâmica de memória*

Condicionantes para a resolução do exercício:

- Definir uma estrutura struct empregado, que contenha 3 membros adequados para guardar os 3 campos relativos a um empregado.
- Definir um novo tipo de dados Empregado à custa de struct empregado.
- Reservar espaço dinamicamente para um *array* de elementos do tipo Empregado, onde serão guardados os dados de todos os empregados a ler do ficheiro.
- Utilizar uma função **PrintEmpregado(Empregado *e, int p)** que escreve no ecrã os dados do empregado localizado na posição **p** da *array* de Empregados **e**, em que **p** é escolhido pelo utilizador.

***Exercício 5**: estruturas, acesso a ficheiros e alocação dinâmica de memória*

- Escrever um programa em C que permite editar e guardar num ficheiro uma pauta de notas duma disciplina.
- Para isso o programa deve:
  - usar uma estrutura **Aluno** onde se pode guardar os dados dum aluno:
    - número (um inteiro)
    - nome (uma string)
    - nota (um real)
  - pedir ao utilizador o número de alunos (**N**) a incluir na pauta
  - reservar espaço de memória para os **N** Aluno's
  - pedir os dados dos **N** alunos e guardá-los na memória reservada
  - guardar a pauta de notas num ficheiro com nome à escolha do utilizador, usando o seguinte formato:

***Exercício 5**: estruturas, acesso a ficheiros e alocação dinâmica de memória*

```
N
numero1  nome1 nota1
numero2  nome2 nota2
 ...      ...   ...
numeroN  nomeN notaN
```

- Usar funções para:
  - ler os dados dos **N** alunos e guardá-los em memória
  - guardar os dados dos **N** alunos num ficheiro
  - mostrar os dados de um aluno, dado o seu número, no ecrã.

***Exercício 6**: estruturas, acesso a ficheiros e alocação dinâmica de memória*

- Desenvolver um programa em C que permite ler os dados relativos a um conjunto de pontos no espaço tridimensional, a partir dum ficheiro texto pontos.dat, para memória e calcular a distância máxima entre 2 pontos.

- O ficheiro pontos.dat está em formato texto, usa o carater espaço em branco para separar os valores, e possui a seguinte estrutura:

  - O primeiro valor do ficheiro é um inteiro (**n**) que representa o número de pontos contidos no ficheiro.
  - O resto do ficheiro é uma sequência de *double*'*s,* representando as coordenadas Z,Y,X para os pontos **1** a **n**:

    $coordenadaZ_1$ $coordenadaY_1$ $coordenadaX_1$ $coordenadaZ_2$
    $coordenadaY_2$ $coordenadaX_2$ ... $coordenadaZ_n$
    $coordenadaY_n$ $coordenadaX_n$

***Exercício 6**: estruturas, acesso a ficheiros e alocação dinâmica de memória*

Condicionantes para a resolução do exercício:

- Definir uma estrutura struct ponto3d, que permite guardar as coordenadas **z**,**y**,**x** dum ponto
- Definir um novo tipo de dados Ponto3D à custa de struct ponto3d
- Reservar espaço dinamicamente para um *array* de elementos do tipo Ponto3D, onde serão guardados os dados de todos os pontos a ler do ficheiro
- Utilizar uma função double **CalcDistMaxima**(Ponto3D ***p,** int **n**), que calcula a distância máxima entre quaisquer 2 pontos do *array*. Os argumentos da função são o apontador para o *array* de pontos (***p***) e o número de pontos do *array* (***n***). Devolve a distância máxima
- Distância entre 2 pontos $\mathbf{P_i}$ e $\mathbf{P_j} \rightarrow \mathbf{D_{i,j}}=\sqrt{[(z_i-z_j)^2+(y_i-y_j)^2+(x_i-x_j)^2]}$