Arquitetura de Computadores

MiEI – 2016/17

DI-FCT/UNL

AC - 2016/17 1

 A equipa

 *Teóricas*

Vitor Duarte

 *Prá,cas*

João Lourenço

Hervé Paulino

Cecília Gomes

Vitor Duarte

AC - 2016/17 2

1

Informações

● **Tudo está no CLIP. NÃO ESQUECER!**

■Horário, objeTvos, programa, bibliografia, avaliação, datas de testes, etc..........

● Atenção aos avisos

● Pode sempre contactar os docentes

■Pessoalmente

■piazza.com/fct.unl.pt/spring2017/ac/home

■email

AC - 2016/17 3

AC - Resumo

● O tema:

■Compreender funcionamento dos computadores e como estes conseguem executar os programas

■Estudar a organização e funcionamento do CPU, da memória e dos periféricos

● Pré-requisitos:

■Saber as matérias de IP e SL

■E ainda: **ter curiosidade e não ser preguiçoso/a** ● Esforço:

■9 créditos ECTS 🡪 ≈ **15h/semana**

AC - 2016/17 4

2

Esforço: O que diz o CLIP…

2hx13sem 

3hx13sem

**10hx12sem**

AC - 2016/17 5

Teóricas e PráTcas ?

● Teóricas:

■Apresentar os conceitos e discuTr o porquê, vantagens e desvantagens

● PráTcas:

■PraTcar o uso dos conceitos e experimentar 🡪 compreender melhor, saber fazer, …

● Testes/exame: avaliam ambas as componentes AC - 2016/17 6

3

Bibliografia

● Computer Systems, a programmer’s 

perspecTve, **2.Ed.**, Bryant &

O’Hallaron

● The C Programming Language, 2.Ed.,

Kernighan & Ritchie 

● Acetatos, apontamentos, manuais, ...

● outros documentos, *sites*, ...

AC - 2016/17 7

Avaliação

● Obter >= 8,5 nas aulas práTcas dá frequência ■Nota da práTca depende de trabalhos individuais e um mini-projecto em grupo de 2

■Se freq. em 2015/16 está dispensado (anterior não) ● Fazer 2 testes ou fazer exame >= 8,5

● Nota final:

■80% testes ou exame + 20% práTca

● Qualquer fraude leva ao chumbo de **todos** os envolvidos

AC - 2016/17 8

4

Como reprovar (escolha uma! ;-)

● Não acompanhar as teóricas

■Julgar que percebem tudo só pelos acetatos

● Não acompanhar as práTcas

■Julgar que basta ver os programas já feitos

● Estudar a matéria das aulas, fazer os exercícios e Trar todas as dúvidas (sem recorrer aos docentes), etc... na véspera do prazo do trabalho, do teste ou do exame

AC - 2016/17 9

No estudo...

● **É necessário trabalhar BEM! ...**

 **... para não trabalhar muito.**

● Não se atrase. Comece cedo, para ter o máximo de ajuda dos docentes.

● Recomenda-se, concentração

■Evitar distrações (Facebook, email, sms, conversa fiada, ...)

■Use várias estratégias de estudo (não leia só os slides) ●Ler, escrever, falar, ouvir, pensar, ver/rever, ...

AC - 2016/17 10

5

ObjecTvos

● Geral:

■Conhecimento!...

■...competência e produTvidade

■...vantagem compeTTva no mercado de trabalho ● Específicos de AC:

■Organização e funcionamento interno dos vários componentes do computador

■Níveis de abstração, interfaces de programação, mecanismos de execução, incluindo os suportados pelo *hardware*

■Diferente SO, diferentes linguagens, linha de comandos AC - 2016/17 11

AC no curso (u.c. relacionadas)

● 1º semestre:

■Introdução à Programação

■Sistemas Lógicos

● 2º semestre:

■**Arquitetura de Computadores**

● Seguintes:

■Fund. Sistemas de Operação, Ling. Ambientes de Programação, Redes de Computadores, Concorrência e Paralelismo,...

■etc…

~~■~~etc…

■etc…

AC - 2016/17 12

6

Programa de AC

● Componentes e organização do computador ■Arquitectura de Von Neumann

● Organização interna do processador (CPU) ■*Instruc,on Set Architecture*. A linguagem máquina e o *assembly*

● Sistema de entradas e saídas, e os Tpos de periféricos ● Hierarquia das unidades de memória

● Metodologia e práTca laboratorial:

■Programação em C e *assembly*, baseada na arquitetura do Tpo PC

AC - 2016/17 13

Da programação à execução

● Java – modelo de programação e execução

*E no meio? Como transformar, traduzir,*

*interpretar, executar...??*

*• Vários níveis de abstração*

**?**

*• Cada nível de abstração oferece uma interface*

*• Facilita o nível superior e usa o nível inferior*

● Sistemas Lógicos/Electrónica – modelo de execução AC - 2016/17 14

7

Níveis nos sistemas informáTcos ● Num sistema de computação: Aplicações

Interface com o computador

linguagens bibliotecas *sistema de operação*

Computador *(hardware)*

s

o

da

D

e

s

a

m

a

r

g

o

r

P

*)*

*e*

*r*

*a*

*w*

*tf*

*o*

*s*

*(*

● *Podemos detalhar em mais níveis …*

AC - 2016/17 15

O tema

● Principais aTvidades num sistema de

computação:

■Computação

■Armazenamento

■Comunicação

● Suportadas pelo hardware ■Via uma interface:

*Instruc,on Set Architecture*

*software*

Unidades Processadoras

(ISA)

*)*

*e*

*r*

*a*

*w*

*dr*

*a*

*h(*

Sistemas de Comunicação Unidades de Armazenamento

AC - 2016/17 16

8

Porquê AC …

● O Eng. InformáTco tem de:

■Cumprir as especificações definidas

■Correção: produzir resultados corretos

■Desempenho: reduzir o tempo de execução ■Compromissos Custo/Realização: esforço de desenvolvimento vs. Tempo vs. custos

■Dominar os produtos e as tecnologias

AC - 2016/17 17

Para …

● Compreender melhor as abstrações e funcionalidades existentes nos vários níveis ● Desenvolver aplicações mais eficientes e fiáveis ■Ao saber mais sobre o sistema que executa a aplicação, pode ser um melhor programador:

●Tirar melhor parTdo do computador e dos vários níveis ●Escrever programas mais fiáveis e rápidos

● Preparar para as cadeiras posteriores

AC - 2016/17 18

9

Cada nível oferece

● Interface

■oferece um conjunto de ”operações"

■este define a "linguagem" e abstrações suportadas ● Transparência

■esconde do nível superior a implementação das suas operações/abstrações

■oferece uma nova máquina (virtual)

AC - 2016/17 19

Execução de programas por tradução

● Os programas são transformados por outros programas

**Programa**

**nível N**

**tradutor**

**Programa**

**nível N-1**

AC - 2016/17 20

10

Execução de programas por interpretação

**Programa**

**nível N**

**interpretador**

**das instruções**

**de nível N-1**

● o *hardware* do computador é sempre o interpretador final

AC - 2016/17 21

Interpretação: Exemplo do Java

**Ficheiro**

**com o código fonte Compilador**

**Ficheiro**

**com o código "máquina-JVM"**

**Biblioteca**

**com o código das classes Java**

**Interpretador**

***programa***

***implementando a JVM "java"***

***Executado***

***pelo computador***

AC - 2016/17 22

11

Passos |picos para execução naTva no *hardware*

**Ficheiro**

**com o código fonte**

**Compilador**

**Ficheiro**

**com o código**

**máquina**

**Biblioteca**

**com o código**

**das funções da**

**linguagem**

Mais MoTvação

**Editor de texto**

**Programa**

**em execução**

**Carregador do SO**

**Ligador Ficheiro**

**executável**

AC - 2016/17 23

● Muitas disciplinas fazem ênfase na abstração ■Exemplos: Tipos de dados abstractos, Classes/Objetos, Análise de complexidade

● A abstração é boa, mas convém não esquecer a realidade!

● Estas abstrações têm limitações

■Especialmente quando há “*bugs*” ou o desempenho não saTsfaz

■É preciso compreender as implementações que as suportam

AC - 2016/17 24

12

Exemplo #1

É preciso compreender o funcionamento do computador ● Provavelmente, nunca irão escrever um programa em “*assembly*”

■Os compiladores são muito melhores (mais pacientes) que nós ● Conhecer “*assembly*” é crucial para perceber o modelo de execução ao nível da máquina

● Tal ajuda a compreender:

■Comportamento de programas na presença de “*bugs*”

● O modelo da linguagem de alto nível deixa de ser aplicável ■Aumento do desempenho de um programa

● Perceber os moTvos da ineficiência de um programa

■Implementação de *so?ware* de sistema

● Compiladores, linguagens e bibliotecas

● Sistema de operação

AC - 2016/17 25

Exemplo #2

**int**’s não são inteiros, **float**’s não são reais ● Exemplos:

■x2 ≥ 0?

●float’s: Sim ... (em princípio ☺)

●int’s:

■ 40000 × 40000 🡪 1600000000

■ 50000 × 50000 🡪 2500000000 ?

■Será (x + y) + z = x + (y + z)?

●int’s: Sim ... (em princípio ☺)

●float’s:

■ (1020 + (-1020)) + 3,14 🡪 3,14

■ 1020 + (-1020 + 3,14) 🡪 3,14 ?

AC - 2016/17 26

13

A importância das memórias

A memória é importante

● A memória não é ilimitada

■Pode ter de ser reservada e gerida

■Muitas aplicações dependem da quanTdade e do desempenho da memória

● O desempenho da memória não é uniforme ■Depende da memória “*Cache*”, da memória real e da memória virtual. Tal pode afectar imenso a execução de um programa ■A adaptação do programa às caracterísTcas do sistema de memória pode levar a grandes aumentos de velocidade ● Os “*bugs*” relacionados com referências à memória são especialmente perniciosos

■Exemplo: em C e C++ o programador pode corromper a memória

■Os efeitos aparecem muitas vezes longe no espaço e no tempo AC - 2016/17 27

Exemplo #3

● Exemplo de um “*bug*” numa referência a memória

main() 

{

 float d = 3.14;

 float a[2];

 a[2] = -1.0; */\* Referencia "out of bound" \*/*

printf("d = %f\n", d);

 exit(0);

}

**Possíveis resultados:**

**-1.000000 3.140000** *Outros…*

**(O programa pode também terminar com um erro: “*Segmentation fault*.” ou “Abort”. O comportamento varia com a arquitetura, o SO e o compilador. )**

AC - 2016/17 28

14

Hierarquia de níveis de memória

● Quanto maior é capacidade maior é o tempo de acesso

*Smaller, faster,*

*and*

*costlier*

L1:

L0:

Registers On-chip L1

*CPU registers hold words retrieved from cache memory.*

*L1 cache holds cache lines*

*(per byte) memory devices*

*Larger, slower,*

L2:

L3:

cache (SRAM)

Off-chip L2

cache (SRAM)

Main memory (DRAM)

*retrieved from the L2 cache.*

*L2 cache holds cache lines retrieved from memory.*

*Main memory holds disk*

*blocks retrieved from local disks.*

*and*

*cheaper (per byte) memory*

L4:

Local secondary storage (local disks)

*Local disks hold files retrieved from disks on remote network servers.*

*devices*Remote secondary storage

L5:

(distributed file systems, Web servers)

AC - 2016/17 29

Exemplo #4

O desempenho de um programa não depende só da complexidade do algoritmo

● Facilmente se têm tempos de execução com relações de 100:1 dependendo da forma como o código é escrito ■É possível opTmizar a múlTplos níveis: algoritmo, representações dos dados, funções e ciclos

● É preciso compreender o sistema para poder opTmizar eficazmente os programas

■Como é que os programas são compilados e executados ■Como medir o desempenho de um programa e idenTficar os gargalos (*boBlenecks*)

■Como melhorar o desempenho sem destruir a modularidade e a generalidade do código

AC - 2016/17 30

15

Exemplos do desempenho

● Desempenhos diferentes de acordo com o padrão de acesso à memória

● Somar todos os elementos de uma matriz: ■MúlTplas formas de percorrer a matriz (**n**x**n**). Exemplos:

/\* ij \*/ 

sum = 0.0;

for (i=0; i<n; i++) { for (j=0; j<n; j++) { sum += a[i][j];

 }

}

/\* ji \*/ 

sum = 0.0;

for (j=0; j<n; j++) {

 for (i=0; i<n; i++) {

sum += a[i][j];

 }

}

AC - 2016/17 31

Desempenho (somas/s para nxn)

600000000

2E+09



1.8E+09

500000000 1.6E+09

1.4E+09

400000000 1.2E+09

300000000 1E+09

800000000

200000000 600000000

400000000 100000000

200000000

0

0























0 200 400 600 800 1000 1200 0 200 400 600 800 1000 1200

Java ji Java ji Java ij Java ij C ji

C ij

AC - 2016/17 32

16

Exemplo #5

Os computadores fazem mais do que executar instruções de um programa

● Enviam e recebem dados para/de periféricos ■O sistema de entradas/saídas (I/O) é críTco para a fiabilidade e o desempenho de um programa

● Os programas comunicam

■Na mesma máquina

■Comunicam através de redes

■Muitos questões aparecem quando se usa uma rede ●ATvidades simultâneas e independentes

●Comunicação não fiável

●Máquinas envolvidas podem usar representações de dados diferentes

●As questões de desempenho e os erros complicam-se

AC - 2016/17 33

Exemplo #6

Toda a informação no sistema é um conjunto de bits

● O mesmo conjunto de bits pode ser:

■Um inteiro

■Um real representado em vírgula flutuante

■Uma sequência de caracteres

■Um conjunto de instruções máquina

● Só se disTngue pela forma como os bits são interpretados

AC - 2016/17 34

17

O código fonte do programa hello.c

#include <stdio.h>

int main()

{

printf(“hello,world\n”);

}

O programa foi criado por um editor de texto e salvo num ficheiro *hello.c*. O ficheiro com código fonte é uma sequência de bits, organizados em bytes. Interpretando o seu conteúdo obtemos:

cada byte interpretando como inteiro:

35 105 110 99 108 117 100 101 32 60 115 116 100 105 111 46 104 62 10 105 ... cada byte interpretando como carácter (norma ISO/ASCII):

# i n c l u d e < s t d i o . h > i n t … Mas ainda não é um programa executável pelo computador!

AC - 2016/17 35

Transformações da linguagem C

● Os programas são transformados por outros programas

● Exemplo da compilação de programas em C: **cc –o hello hello.c**

libc.a *Biblioteca de C*

printf.o

*(arquivo de binários)*

**hello.c hello**

*compilador*

(cc)

*Programa*

*Fonte C*

*(texto)*

hello.o *Linker* (ld)

*Ficheiro*

*objecto*

*(binário)*

*Programa executável (binário)*

AC - 2016/17 36

18

Hello.o

● Interpretando o conteúdo de hello.o obtemos para main: ■Inteiro hex: 55 89 e5 83 e4 f0 83 ec 10 c7 04 24 ■Inteiro dec: 85 -119 -27 -125 -28 -16 -125 -20 16 -57 4 36 ■interpretando como caracteres (norma ISO/ASCII):

● U ? ? ? ? ? ? ? ? ? ? ? $

■Interpretando como instruções (ia32):

 main:

0: 55 push %ebp

 1: 89 e5 mov %esp,%ebp 3: 83 e4 f0 and $0xfffffff0,%esp

6: 83 ec 10 sub $0x10,%esp

9: c7 04 24 00 00 00 00 movl $0x0,(%esp)

10: e8 fc ff ff ff call 11 <main+0x11>

15: b8 00 00 00 00 mov $0x0,%eax

1a: c9 leave

1b: c3 ret

AC - 2016/17 37

Transformações da linguagem Java

● Java é compilado para um código máquina virtual **javac Hello.java**

● Para executar indica-se o nome da classe um programa que interpreta esse código

**java Hello**

println

*Biblioteca*

*de Java*

*(arquivo de*

*binários)*

*Este vai procurar as*

Hello.java

*Programa*

*Fonte C*

*(texto)*

compilador (javac)

Hello.class *interpretador* (java)

*Ficheiro class*

*(binário JVM) Programa executável*

*(binário)*

*classes necessárias. Interpreta e compila à medida que*

*executa.*

AC - 2016/17 38

19

Grandes conceitos na Eng. InformáTca

● Abstração / camadas e interfaces

● OpTmizar os casos mais comuns

● Hierarquia de memórias / latências

● Desempenho por previsão / localidade temporal e espacial

● Desempenho pelo paralelismo e/ou *pipelining* ● Redundância / lidar com falhas

AC - 2016/17 39

20