Arquitectura de Computadores I - Módulo 1

Níveis conceptuais

1. Introdução

Pretende-se com esta aula prática que os alunos entendam os vários níveis de abstracção envolvidos no processo de desenvolvimento de *software* e compreendam as representações usadas em cada nível.

Para atingir este objectivo o aluno deve desenvolver um pequeno programa em C, constituído por 2 módulos, e, usando as ferramentas do Linux `gcc`, `gdb` e `objdump`, acompanhar e visualizar as várias fases desse processo.

2. Linguagem de alto nível

Escreva em C, usando o editor de texto que considerar mais adequado, os 2 módulos apresentados na tabela 1.

main.c	**soma.c**

`int total=0;`
`main () {`
`  int i;`
 	
`  i = 10 ;`
`  soma (i); }`
	`extern int total;`
	
	`void soma (int p)`
	`{`
	`  total += p;`
	`}`

Questão 1 – Qual o tamanho da cada um dos ficheiros?

Questão 2 – Em que formato está representada a informação contida nestes ficheiros?

3. Compilação

Por compilação entende-se a conversão do programa escrito numa linguagem de alto nível para o nível do *assembly*. Note que apesar de a maior parte dos compiladores permitir, com uma única linha de comando, passar directamente do nível da linguagem de alto nível para o nível máquina, na realidade estão a ser executados 3 programas distintos, correspondentes a 3 fases diferentes: compilação, montagem (*assembler*) e *linker*.

Compile o módulo `soma.c` e `main.c` usando o scomandos

```
gcc –O2 –S soma.c
gcc –O2 –S main.c
```

A opçãoO2 indica ao compilador para usar o nível dois de optimização do código, enquanto a opção S indica que deve gerar apenas o código *assembly*. Estes comandos geram os ficheiros `soma.s` e `main.s`.

Questão 3 – Em que formato está representada a informação contida nestes ficheiros?

Questão 4 – Usando um programa adequado visualize o conteúdo de `soma.s` e `main.s`. Encontra informação simbólica? Qual?

Questão 5 – Este programa pode ser executado directamente pela máquina? Em que nível de abstracção nos encontramos?

4. Montagem (Assembler)

Use os comandos

```
gcc –O2 –c soma.c
gcc –O2 –c main.c
```

para gerar o código binário correspondente aos módulos `soma.c` e `main.c`. O código binário não pode ser visualizado usando um editor de texto, pois o formato da informação já não é ASCII. Para visualizar o conteúdo do ficheiro usa-se um *debugger* (depurador) fornecido com o Linux. Execute a seguinte sequência de comandos:

```
> gdb soma.o
(gdb)x/18xb soma
```

Questão 6 – O que representam os valores que está a visualizar?

Questão 7 – Este programa pode ser executado directamente pela máquina? Em que nível de abstracção nos encontramos?

O conteúdo dos ficheiros objecto pode ser visualizado usando *disassemblers*. Execute os comandos

```
objdump –d soma.o
objdump –d main.o
```

Questão 8 – Este programa contem informação simbólica?

Questão 9 – Como está representada a variável `total`? Porque razão é ela representada desta forma?

Questão 10 – Quantas instruções tem a função soma? Quantos bytes ocupa? Quais são as instruções mais curtas e mais longas?

5. Linker

Para gerar o programa executável é necessário ligar os dois módulos entre si e com quaisquer outras bibliotecas de funções que sejam utilizadas, assim como acrescentar código que lida com o Sistema Operativo. Este é o papel de *linker*. Execute o comando

```
gcc –O2 –o prog main.c soma.o
```

Questão 11 – O resultado da execução deste comando é colocado no ficheiro `prog`. Qual o formato da informação aí contida? Este ficheiro pode ser executado directamente pela máquina?

Visualize o conteúdo deste ficheiro e guarde-o num ficheiro de texto usando o comando

```
objdump –d prog > prog.dump
```

Localize no ficheiro `prog.dump` a função `soma`.

Questão 12 – Como está representada a variável `total`?

Questão 13 – Porque ordem são armazenados na memória os 4 bytes correspondentes ao endereço de `accum`? *Little-endian* ou *big-endian*?

Questão 14 – Como é que a função `main` passa o controlo (invoca) a função `soma`?