

Guião II - Níveis de Abstração

1. Edição de programas

Editar os ficheiros **main.c** e **soma.c** com o conteúdo seguinte:

<u>main.c</u>	<u>soma.c</u>
main ()	int accum=0;
{	void soma (int p)
int x;	{
soma (x);	accum += p;
}	}

1.a) Identifique o formato de representação dos ficheiros **.c** usando o comando **file**:

* **file main.c** → main.c: ASCII text

* **file soma.c** → soma.c: ASCII text

Ambos os ficheiros estão no formato **texto**, ou seja, cada elemento incluído nos ficheiros representa um carácter codificado em ASCII.

1.b) Que relação existe entre aqueles dois módulos?

Na função **main()** é chamada a função **soma()**.

2. Ferramentas

Usar o comando **man <util>** para conhecer os utilitários **gcc**, **gdb** e **objdump**.

3. Compilação

Compile os módulos **soma.c** e **main.c** usando o comando:

```
gcc -Wall -O2 -S soma.c main.c
```

3.a) Identifique as diferentes opções usadas no comando acima.

-Wall → opção que ativa todas (*all*) as mensagens de aviso (*Warning*) por parte do **gcc**;

-O2 → opção que indica ao **gcc** para ativar o nível 2 de otimização;

-S → faz com que o processo de compilação termine antes do processo de montagem.

Outras opções que controlam a otimização de código:

-O, -O1 → O compilador tenta reduzir o tamanho do código e o tempo de execução, sem efetuar otimizações que ocupem muito tempo de compilação.

-O2 → Otimizar ainda mais. O **gcc** efetua quase todas as otimizações suportadas e que não exijam ter que assumir um compromisso entre espaço-desempenho. Em comparação com **-O**, esta opção aumenta o tempo de compilação e o desempenho do código gerado.

-O3 → Otimizar ainda mais. Ativa todas as otimizações especificadas por **-O2** e também ativa: **-finline-functions**, **-funswitch-loops**, **-ftree-loop-vectorize**, etc.

-O0 → Reduz o tempo de compilação e permite que a depuração produza os resultados esperados. Esta é a opção por omissão.

3.b) Identifique o formato dos ficheiros produzidos.

```
main.s: ASCII assembler program text
soma.s: ASCII assembler program text
```

Tal como os ficheiros originais também os ficheiros produzidos continuam a estar no formato **texto**. No entanto, uma análise da informação neles contida permite-nos concluir que os mesmos contêm **código assembly IA32**.

3.c) Visualize os ficheiros gerados e interprete o respetivo conteúdo, determinando o nível de abstração implícito.

soma.s

```
.file      "soma.c"
.globl accum
.data
.align     4
.type      accum,@object
.size      accum,4
accum:
    .long   0
.text
.globl soma
.type      soma,@function
soma:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %eax
    addl    %eax, accum
    leave
    ret
...
```

main.s

```
.file      "main.c"
.text
.globl main
.type      main,@function
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    andl    $-16, %esp
    subl    $12, %esp
    pushl   %eax
    call    soma
    leave
    ret
...
```

O código *assembly* é constituído por **mnemónicas** que identificam as operações que o CPU executa (por ex., `movl`, `subl`, `addl`, `pushl`) seguidas da identificação dos operandos fonte e/ou destino (através do seu valor ou da localização).

Os **valores** são normalmente numéricos, e as **localizações** são (i) endereços de memória, (ii) nomes de registos (que começam por "%"), ou (iii) especificação de endereços de memória usando constantes e 1 a 2 nomes de registos entre parênteses.

No código *assembly* também está presente **informação simbólica**:

- As linhas que começam com "." são **diretivas** (comandos) para o programa de montagem e especificam (i) onde começa o bloco de informação que contém o código do módulo ("**.text**"), (ii) onde são definidas as variáveis globais ("**.bss**" ou "**.data**"), ou (iii) que uma determinada sequência de caracteres (nome de função ou variável) é única no programa e foi definida neste módulo, podendo ser acedida a partir de qualquer outro módulo que seja posteriormente ligado a este e que lhe faça referência. Exemplo: "**global <nomeFunc | nomeVar>**".
- As linhas que terminam com ":" são **etiquetas** que indicam
 - a localização de um dado pedaço de código dentro do bloco **.text** ou
 - a localização de uma dada variável dentro do bloco que contém as variáveis globais.
- As **sequências de caracteres no meio do código** de montagem, associadas normalmente ao nome de uma variável ou função, indicam a localização em memória onde ela irá tomar os diversos valores ao longo da execução de um programa (as etiquetas referidas antes).

4. Montagem

Executar o comando: `gcc -Wall -O2 -c soma.s`

4.a) Qual o objetivo de usar a opção `-c` ?

Utilizando a opção `-c` o `gcc` irá **compilar** e fazer a **montagem** do código fonte terminando imediatamente antes de fazer a **ligação** de módulos.

4.b) Identifique o formato do ficheiro resultante.

O ficheiro está em **formato objeto**, em que parte do seu conteúdo é já o **código em linguagem máquina** do IA32.

`soma.o`: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped

4.c) Obteria o mesmo resultado se usasse como entrada o ficheiro `soma.c` ?

```
gcc -Wall -O2 -c soma.c
```

```
gcc -Wall -O2 -c soma.s
```

Os comandos anteriores produzem o mesmo resultado, o ficheiro objeto **`soma.o`**

4.d) Qual o nível de abstração implícito?

O nível de abstração do resultado (`soma.o`) é o do **código objeto**.

4.e) O ficheiro `soma.o` pode ser executado diretamente pela máquina?

`soma.o` é um ficheiro objeto, em que parte do seu conteúdo é já o código do programa em linguagem máquina do IA32. Contudo, olhando para os endereços de cada uma das instruções (que começam no endereço "0") pode concluir-se que o código ainda não está pronto para ser executado, para além de se saber que ainda falta ligar com `main.o` para produzir um programa final. Também não possui código de interação com o sistema operativo.

5. Desmontagem/Disassembly

Executar o comando: `objdump -d soma.o`

Resultado:

```
-----
Disassembly of section .text:
soma.o: file format elf32-i386
00000000 <soma>:
   0:    55                push    %ebp
   1:    89 e5             mov     %esp, %ebp
   3:    8b 45 08          mov     0x8(%ebp), %eax
   6:    01 05 00 00 00 00 add     %eax, 0x0
   c:    c9              leave
   d:    c3              ret
-----
```

5.a) O ficheiro contém informação simbólica?

O ficheiro continua a ter informação simbólica: `<soma>`

5.b) Identifique as diferentes representações das instruções presentes quanto ao tamanho.

<code>push %ebp</code>	ocupa 1 byte
<code>mov %esp, %ebp</code>	ocupa 2 bytes
<code>mov 0x8(%ebp), %eax</code>	ocupa 3 bytes
<code>add %eax, 0x0</code>	ocupa 6 bytes
<code>leave</code>	ocupa 1 byte
<code>ret</code>	ocupa 1 byte

Identifique as diferentes representações das instruções presentes quanto ao formato.

```
push  Reg
mov   regS, regD
mov   Imm(regB), regD
add   Reg, Imm
leave
ret
```

6. Desmontagem/Disassembly

Baseado na visualização do conteúdo do resultado produzido pelo comando:

```
objdump -d soma.o
```

6.a) Que relação existe entre a informação observada acima e a presente no ficheiro `soma.s`?

```
===== soma.s =====
pushl   %ebp
movl    %esp, %ebp
movl    8(%ebp), %eax
addl    %eax, accum
leave
ret

===== disassembly de soma.o =====
0: push   %ebp
1: mov    %esp, %ebp
3: mov    0x8(%ebp), %eax
6: add    %eax, 0x0
C: leave
D: ret
=====
```

A variável global **accum** já não aparece como informação simbólica: foi convertida num número que representa o endereço de memória onde irá ficar alojada. O endereço de memória é "0" e coincidente com o endereço do início do código da função **soma()**. A razão para essa coincidência resulta do facto de o endereço especificado para a variável indicar a sua localização dentro do bloco das variáveis globais (i.e., é a primeira variável global) enquanto que o endereço 0 do código da função **soma()** indica a localização desse código dentro do bloco **.text** do ficheiro.

6.b) Identifique o código relativo à representação da variável **accum**?

```
6: 01 05 00 00 00 00      add %eax, 0x0
```

Apresente razões para o modo de representação usado.

A variável **accum** é representada pelo endereço de memória onde irá ficar alojada ('0' neste caso). O endereço '0' indica a localização relativa dentro do bloco das variáveis globais, ou seja, é a primeira variável global.

7. Desmontagem/Disassembly

Execute o comando: `gdb soma.o`

7.a) Identifique o resultado do comando do `gdb`: `x/23xb soma` (mostra 23 bytes da função `soma` em hexadecimal)

```
0x0 <soma>: 0x55 0x89 0xe5 0x8b 0x45 0x08 0x01 0x05
0x8 <soma+8>: 0x00 0x00 0x00 0x00 0xc9 0xc3 Cannot access
memory at address 0xe
```

7.b) Identifique o resultado do comando do gdb: **disass soma** (*disassembly* da função soma)

```
Dump of assembler code for function soma:
0x00000000 <soma+0>:    push    %ebp
0x00000001 <soma+1>:    mov     %esp,%ebp
0x00000003 <soma+3>:    mov     0x8(%ebp),%eax
0x00000006 <soma+6>:    add     %eax,0x0
0x0000000c <soma+12>:   leave
0x0000000d <soma+13>:   ret
End of assembler dump.
```

8. Ligação e execução de módulos

Execute o comando: **gcc -Wall -O2 -o prog main.c soma.o**

8.a) Identifique o formato do ficheiro resultante, através do comando: **file prog**

prog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.9, dynamically linked (uses shared libs), for GNU/Linux 2.6.9, not stripped

Este ficheiro contém toda a informação necessária para poder ser executado no computador: código em linguagem máquina dos dois módulos, incluindo o código das funções de bibliotecas (do C, do sistema operativo, etc), e os valores inicializados das variáveis globais (por ex., mensagens de erro).

8.b) Como procederia para executar o programa?

./prog

Dado que o caminho para o executável **prog** não se encontra listado na variável de ambiente **\$PATH** é necessário inserir o caminho, neste caso o diretório atual '.'

9. Depuração de programas

Executar o comando: `objdump -d prog`

```
-----
...
08048354 <main>:
8048354: 55                push    %ebp
8048355: 89 e5            mov     %esp,%ebp
8048357: 83 ec 08        sub     $0x8,%esp
804835a: 83 e4 f0        and     $0xfffffffff0,%esp
804835d: 83 ec 0c        sub     $0xc,%esp
8048360: 50              push    %eax
8048361: e8 02 00 00 00  call    8048368 <soma>
8048366: c9              leave
8048367: c3              ret

08048368 <soma>:
8048368: 55                push    %ebp
8048369: 89 e5            mov     %esp,%ebp
804836b: 8b 45 08        mov     0x8(%ebp),%eax
804836e: 01 05 bc 95 04 08 add     %eax,0x80495bc
8048374: c9              leave
8048375: c3              ret
...
-----
```

9.a) Repita o exercício 6 para o resultado obtido pelo comando anterior.

A variável **accum** e o código da função soma deixaram de ser referenciados pelo endereço "0", foram recolocados em endereços distintos (8048368 - **soma**, 80495bc - **accum**), compatíveis com a execução num sistema real.

9.b) Relacione a representação da variável **accum** com os conceitos de little/big-endian.

----- código binário -----	----- código assembly -----
804836e: 01 05 bc 95 04 08	add %eax,0x 080495bc

Analisando lado a lado o código binário e o código *assembly* pode verificar-se que a forma como o endereço de **accum** é guardado em memória corresponde a uma arquitetura com ordenação *little endian*: o LSB (0xbc) é guardado no endereço menor (0x**08048370**) e o MSB (0x**08**) é guardado no endereço maior (0x**08048373**).

10. Depuração de programas

Execute o comando: **`gdb prog`**

10.a1) Identifique o resultado do comando do `gdb`: **`disass soma`**

```
-----  
0x08048368 <soma+0>:      push    %ebp  
0x08048369 <soma+1>:      mov     %esp,%ebp  
0x0804836b <soma+3>:      mov     0x8(%ebp),%eax  
0x0804836e <soma+6>:      add     %eax,0x80495bc  
0x08048374 <soma+12>:     leave  
0x08048375 <soma+13>:     ret  
-----
```

10.a2) Identifique o resultado do comando do `gdb`: **`disass main`**

```
-----  
0x08048354 <main+0>:      push    %ebp  
0x08048355 <main+1>:      mov     %esp,%ebp  
0x08048357 <main+3>:      sub     $0x8,%esp  
0x0804835a <main+6>:      and     $0xffffffff0,%esp  
0x0804835d <main+9>:      sub     $0xc,%esp  
0x08048360 <main+12>:     push    %eax  
0x08048361 <main+13>:     call    0x8048368 <soma>  
0x08048366 <main+18>:     leave  
0x08048367 <main+19>:     ret  
-----
```

10.b) Identifique no código acima a passagem do controlo à função **`soma()`**.

```
0x08048361 <main+13>:  call 0x8048368 <soma>
```

A análise do código do **`main()`** mostra a existência de uma instrução **`call`** seguida de um endereço de memória que corresponde precisamente ao início da função **`soma()`**.

10.c) Execute o programa, usando o comando **`run`** do `gdb`

```
Starting program: /state/partition1/home/aje/aula5/prog
```