Programação C/Assembly

João Canas Ferreira

Março 2013

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Assembly e linguagens de alto nível

Para quê combinar *assembly* com linguagens de alto nível?

- Utilização de linguagem de alto nível facilita desenvolvimento:
  - mais alto nível de abstração
  - tempo de desenvolvimento mais reduzido
  - maior facilidade de depuração
  - independente do CPU
- Utilização de linguagem *assembly:*
  - permite aproveitar todas as características do CPU
  - permite obter programas mais curtos e mais rápidos
  - permite aceder a instruções que não têm equivalência no modelo de programação
    - gestão de memória virtual
    - gestão de *cache*
    - registos de monitorização de desempenho
    - atendimento de alguns tipos de interrupções

Como combinar C++ e linguagem assembly?

- Utilizar *assembly* embutido *(inline)* no código C++ (não *standard*):

```
int power2(unsigned int num, unsigned int power)
{
    __asm
    {
        mov eax, num
        mov ecx, power
        shl eax, cl
    }
}
```

- Compilação separada
  - rotinas numa linguagem podem invocar rotinas na outra
  - programa principal em C/C++ (o mais habitual)
  - programa principal em *assembly*

Utilização de sub-rotinas em assembly a partir de C/C++

Aspetos a considerar

- compilar separadamente cada módulo com ferramenta apropriada
- criar o executável com o compilador de C/C++
  - garante a utilização das bibliotecas apropriadas
- declarar sub-rotinas de *assembly* em C/C++
- rotinas de assembly devem usar convenção de invocação `cdecl`
  - semelhante a `stdcall`
  - diferença principal: o código que invoca a rotina é responsável por remover os parâmetros da pilha
  - `invoke` (masm) trata automaticamente dos detalhes

Exemplos seguintes

- Windows 32 bits
- Ferramentas de Windows SDK 7.1 (as mesmas que Visual Studio 2010)
- Compilador de C/C++: **`cl`**
- *Assembler:* **`ml`**

Exemplo 1: programa principal (1/2)

```
#include <cstdlib>
#include <ctime>
#include <iostream>
#include "pesquisa.h"
using namespace std;

int main()
{
  const unsigned TAM=10000;
  const unsigned N_REPET=1000000;
  long vector[TAM];
  long valor;
  clock_t startTime, endTime;
  bool encontrado;

  for(int i=0; i<TAM; i++)
    vector[i]=rand();
```


Exemplo 1: programa principal (2/2)

```
  startTime = clock();
  for(int n=0; n<N_REPET; n++)
    encontrado=PesquisaVector(valor,vector,TAM);
  endTime = clock();
  cout << "Tempo␣gasto␣(C++):␣"
       << double(endTime-startTime)/CLOCKS_PER_SEC
       << "␣segundos.␣Encontrado=" << encontrado << endl;

  // Versao ASM
  startTime = clock();
  for( int n=0; n<N_REPET; n++)
    encontrado=AsmPesquisaVector( valor, vector, TAM);
  endTime = clock();
  cout << "Tempo␣gasto␣(ASM):␣"
       << double(endTime-startTime)/CLOCKS_PER_SEC
       << "␣segundos.␣Encontrado=" << encontrado << endl;
  return EXIT_SUCCESS;
}
```

Exemplo 1: sub-rotina em C/C++

```
extern "C" {
  bool AsmPesquisaVector(long n,long vector[],
                         long nElem);
  bool PesquisaVector(long n,long vector[],
                      long nElem);
}
```

```
#include "pesquisa.h"
bool PesquisaVector(long valor, long vector[],
                    long nElem)
{
   for(int i=0; i<nElem; i++) {
        if (vector[i]==valor)
    return true;
  }
  return false;
}
```


Exemplo 1: sub-rotina em assembly (manual)

```
   .586
   .model flat,C
   .code
AsmPesquisaVector PROC USES EDI valor:DWORD,
                  vectPtr:PTR DWORD, nElem:DWORD
    mov     eax, valor
    mov     ecx, nElem
    mov     edi, vectPtr
    repne   scasd
    jz      verdade
    mov     al,0                ; falso
    jmp     fim
verdade:
    mov     al, 1               ; verdade
fim:
    ret
AsmPesquisaVector ENDP
END
```

Exemplo 1: compilação e execução

Compilação com Microsoft Compiler e Masm

```
cl /c /EHsc main1.cpp     ⇒     main1.obj
cl /c pesquisa.cpp        ⇒     pesquisa.obj
ml /c AsmPesquisa.asm     ⇒     AsmPesquisa.obj
cl main1.obj pesquisa.obj AsmPesquisa.obj /link /out:out1.exe
```

Execução

```
D:\masm-proj\mixed>out1
Valor central: 14341
Valor a procurar: 14341
Aguarde...
Tempo gasto (C++): 42.432 segundos. Encontrado=1
Tempo gasto (ASM): 10.467 segundos. Encontrado=1

D:\masm-proj\mixed>out1_opt
Valor central: 14341
Valor a procurar: 14341
Aguarde...
Tempo gasto (C++): 10.545 segundos. Encontrado=1
Tempo gasto (ASM): 10.327 segundos. Encontrado=1
```

Otimização:

cl /c /Ox pesquisa.cpp

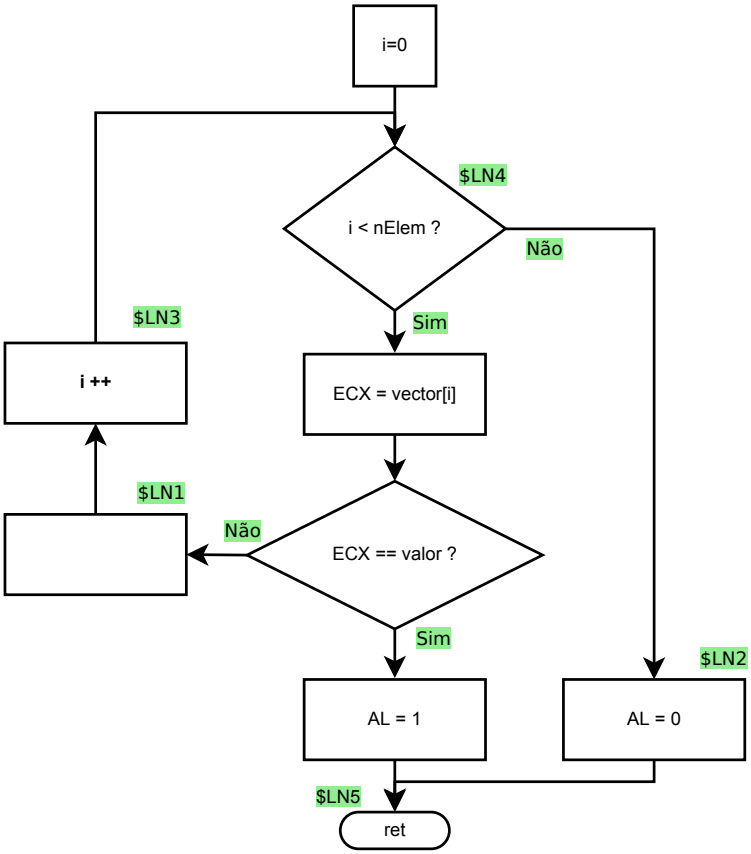
Para obter código gerado:

cl /c /Ox /Fa pesquisa.cpp
⇒ pesquisa.asm

Exemplo 1: código gerado (parcial)

```
_i$2543 = -4                    ; size = 4
_valor$ = 8                     ; size = 4
_vector$ = 12                   ; size = 4
_nElem$ = 16                    ; size = 4
_PesquisaVector PROC
; File d:\masm-proj\mixed\pesquisa.cpp
; Line 4
  push  ebp
  mov ebp, esp
  push  ecx
; Line 5
  mov DWORD PTR _i$2543[ebp], 0
  jmp SHORT $LN4@PesquisaVe
$LN3@PesquisaVe:
  mov eax, DWORD PTR _i$2543[ebp]
  add eax, 1
  mov DWORD PTR _i$2543[ebp], eax
$LN4@PesquisaVe:
```

Diagrama de fluxo



Exemplo 1: código gerado com otimização (simplificado)

```
_valor$ = 8
_vector$ = 12
_nElem$ = 16

_PesquisaVector PROC
  mov  ecx,_nElem$[esp-4]
  xor  eax, eax
  push esi
  test ecx, ecx
  jle  $LN2@PesquisaVe
  mov  edx,_vector$[esp]
  mov  esi,_valor$[esp]

$LL4@PesquisaVe:
  cmp  [edx+eax*4], esi
  je   $LN8@PesquisaVe
  inc  eax
  cmp  eax, ecx
  jl   $LL4@PesquisaVe

$LN2@PesquisaVe:
  xor  al, al
  pop  esi
  ret  0

$LN8@PesquisaVe:
  mov  al, 1
  pop  esi
  ret  0
_PesquisaVector ENDP
```

Exemplo 2: Maior divisor comum (GCD)

```
// C++
startTime = clock();
for( int n = 0; n < N_REPETE; n++)
  res = Gcd(1234*27, 7837485*27);
endTime = clock();
cout << "Tempo␣decorrido␣(C++):␣"
     << double(endTime - startTime) / CLOCKS_PER_SEC
     << "␣segundos.␣Resultado=" << res << endl;
// assembly
startTime = clock();
for( int n = 0; n < N_REPETE; n++)
  res = AsmGcd(1234*27, 7837485*27);
endTime = clock();
cout << "Tempo␣decorrido␣(ASM):␣"
      << double(endTime - startTime) / CLOCKS_PER_SEC
     << "␣segundos.␣Resultado="  << res << endl;
return EXIT_SUCCESS;
```


Exemplo 2: sub-rotina em C

```
#include "gcd.h"

int Gcd (int a,  int b)
{
    if (a == 0 && b == 0)
        b = 1;
    else if (b == 0)
        b = a;
    else if (a != 0)
        while (a != b)
            if (a <b)
                b -= a;
            else
                a -= b;
    return b;
}
```

Fonte: http://www.azillionmonkeys.com/qed/asmexample.html

Exemplo 2: execução

- Resultado da execução:

```
D:\masm-proj\mixed>out4
Aguarde...
Tempo decorrido (C++): 92.149 segundos. Resultado=27
Tempo decorrido (ASM): 26.739 segundos. Resultado=27

D:\masm-proj\mixed>out4_opt
Aguarde...
Tempo decorrido (C++): 39.358 segundos. Resultado=27
Tempo decorrido (ASM): 26.442 segundos. Resultado=27
```

- A versão em *assembly* é mais rápida que a versão C++ otimizada

Exemplo 2: sub-rotina em assembly

```
    .686
    .model flat,C
    .code
AsmGcd PROC a: SDWORD, b: SDWORD
       mov   eax, a
       mov   edx, b
gcd:   neg   eax
       je    L3
L1:    neg   eax
       xchg  eax,edx
L2:    sub   eax,edx
       jg    L2
       jne   L1
L3:    add   eax,edx
       jne   L4
       inc   eax
L4:    ret
AsmGcd ENDP
```

Fonte: P. Hsieh
http://www.azillionmonkeys.com/qed/asmexample.html