

TPC7 e Guião Laboratorial

Resolução dos exercícios

1 Ciclo For

Uma forma de se analisar o código de um ficheiro executável (e para o qual não se tenha acesso ao ficheiro fonte em HLL que lhe deu origem) consiste em (i) desmontar o ficheiro binário para a versão *assembly* e depois (ii) tentar inverter o processo de compilação e produzir código C que pareça “natural” a um programador de C. Por exemplo, não queremos código com instruções `goto`, uma vez que estas são raramente usadas em C; e muito provavelmente não usaríamos também aqui a instrução `do-while`.

Este exercício obriga-nos a pensar no processo inverso da compilação num dado enquadramento: no modo como os ciclos `for` são traduzidos.

- Rotina...
- Ver alínea seguinte...
- A partir do ficheiro executável que foi disponibilizado, `m_contaN`, é possível desmontá-lo para *assembly*, localizar a parte de código simbólico correspondente à função `contaN` e ainda distinguir as partes de inicialização e término (da função), do corpo da função (a parte pertinente neste exercício).
O código simbólico da função deverá ter um aspecto semelhante ao seguinte (este código inclui já uma anotação introduzida manualmente, bem como as etiquetas):

contaN:

```

0x080483f4 <contaN+0>: push    %ebp                ; inicializa função:  salvag/ frame pointer antigo
0x080483f5 <contaN+1>: mov     %esp, %ebp                ;                cria um novo frame pointer
0x080483f7 <contaN+3>: push    %esi                ;                salvaguarda registo %esi
0x080483f8 <contaN+4>: push    %ebx                ;                salvaguarda registo %ebx
0x080483f9 <contaN+5>: mov     0x8(%ebp), %esi      ; coloca apontador p/ início array cadeia em %esi
0x080483fc <contaN+8>: mov     0xc(%ebp), %ecx      ; inicializa com c a var controlo de ciclo (em %ecx)
0x080483ff <contaN+11>: mov     (%ecx,%esi,1), %dl    ; coloca o char, na posição c da cadeia, em %dl
0x08048402 <contaN+14>: xor     %ebx, %ebx          ; inicializa %ebx a 0 (reg alocado à var local result)
0x08048404 <contaN+16>: test    %dl, %dl            ; testa se o char é 0 (caráter “null” em ASCII)
0x08048406 <contaN+18>: je      0x08048420<contaN+34> ; je equivale a jz, ie, salta p/ fim cadeia se 0
.L7:
0x08048408 <contaN+20>: lea     -0x30(%edx), %eax     ; subtrai 48 (ASCII: 0) a %dl e põe em %al
0x0804840b <contaN+23>: cmp     $0x9, %al           ; compara %al com 9
0x0804840d <contaN+25>: ja      0x08048416<contaN+34> ; salta se valor >9, i.e., não é dígito em ASCII
0x0804840f <contaN+27>: movsb   %dl, %eax           ; põe o char lido em %eax estendido p/ 32b c/ sinal
0x08048412 <contaN+30>: lea     -0x30(%eax,%ebx), %ebx ; soma o dígito e acumula o valor em result
.L4:
0x08048416 <contaN+34>: inc     %ecx                ; atualiza o valor da posição na cadeia
0x08048417 <contaN+35>: mov     (%ecx,%esi,1), %al    ; coloca o char na nova posição cadeia, em %al
0x0804841a <contaN+38>: test    %al, %al            ; novo teste: se char é 0 (caráter “null” em ASCII)
0x0804841c <contaN+40>: mov     %al, %dl            ; transfere o char p/ %dl (não afecta bits de cond.)
0x0804841e <contaN+42>: jne     0x08048408<contaN+20> ; salta p/ início ciclo se não é fim da cadeia
.L9:
0x08048420 <contaN+44>: mov     %ebx, %eax           ; função devolve em %eax o  $\Sigma$  calculado (result)
0x08048422 <contaN+46>: popl    %ebx                ; término da função: recupera registo %ebx
0x08048423 <contaN+47>: popl    %esi                ;                recupera registo %esi
0x08048424 <contaN+48>: leave   ;                recupera SP e FP antigo
0x08048425 <contaN+49>: ret     ;                recupera IP e regressa

```

De acordo com as anotações já introduzidas no código, as identificações pedidas na alínea **c)** são agora fáceis de resolver:

- sendo o `result` o somatório de dígitos a devolver, é só procurar a inicialização a zero de um registo e, se este não for o `%eax`, (neste caso é o `%ebx`) então procurar no fim uma instrução que copie o valor em `%ebx` para `%eax`;
- a variável `i` deveria ser inicializado com `c` e incrementada dentro do ciclo `for`, e isso acontece com o registo `%ecx`;
- sabendo que os argumentos se encontram na *stack* à distância de 8 e 12 *bytes* do valor apontado pelo *frame pointer* (em `%ebp`), é fácil de ver para que registos foram copiados;
- o ciclo `for` deverá terminar quando o carácter lido for *null* (e também deverá ser garantido que o ciclo não é executado se o único carácter da *cadeia* for também *null*); assim o código *assembly* deverá conter 2 expressões de teste semelhantes (estão nas linhas 7 e 9 no teste antes de entrar no ciclo, e 17 e 18 dentro do ciclo `for`);
- atualização da variável `i`: procurar por uma instrução que faça o incremento de `%ecx`;
- Consultando uma tabela ASCII fica-se a saber que os algarismos 0 a 9 são codificados por `0x30` a `0x39`, ou seja de 48 a 57 (em decimal);
- a atualização do somatório (na variável `result`) deveria ser feita pela expressão `result = result + ASCII_carácter_lido_estendido_para_32_bits - 48`; embora as instruções aritméticas do IA-32 não permitam especificar mais que 2 operandos, a expressividade da instrução `lea` talvez permita... Procurem no código!

Pode-se ainda ver que:

- as expressões de teste do ciclo `for` têm como objetivo verificar se o carácter lido é o fim da cadeia ou não, i.e., se `cadeia[c]=0`;
- há ainda uma outra expressão de teste associada a uma típica estrutura de `if...then` para decidir quando somar os caracteres (quando forem algarismos...); esta expressão tem como objetivo verificar se o carácter lido – o valor de 8 bits que está no registo `%al` – subtraído de 48 é menor ou igual a 9, o que corresponde a verificar se é o código ASCII de um algarismo ou não, i.e., se `cadeia[c]>='0' && cadeia[c]<='9'`.

- d)** Com base no código anotado da alínea anterior (e comentários que se seguiram), e sabendo a estrutura habitual do código gerado por um compilador com um nível médio de otimização, é possível chegar-se ao seguinte código original em C:

```

1 int contaN(char *cadeia, int c)
2 {
3     int i;
4     int result=0;
5     for (i = c; cadeia[i]!='\0' ; i++)
6         if (cadeia[i] >= '0' && cadeia[i] <= '9')
7             result +=(cadeia[i]-'0');
8     return result;
9 }
```