TPC9 e Guião Laboratorial

Resolução

Pretende-se com esta sessão laboratorial recuperar a informação perdida num ficheiro que continha o código *assembly* da função `int soma_grandes (int n, int *a)` escrevendo, em *assembly*, a parte que foi "danificada".

O código "danificado" desta função percorria os `n` primeiros elementos do vetor de inteiros `a` e adicionava todos os maiores que 1000; terminava devolvendo o valor dessa adição.

Para recuperar este código em *assembly* sugere-se a seguinte metodologia:

- escrever um possível algoritmo da função numa notação/linguagem de alto nível;
- preencher a tabela de alocação de registos a argumentos, variáveis locais e eventuais variáveis de caráter mais temporário, que seja necessário armazenar;
- desenhar o quadro de ativação (*stack frame*) da função;
- escrever o código que substitui a parte "danificada".

1. Um possível algoritmo

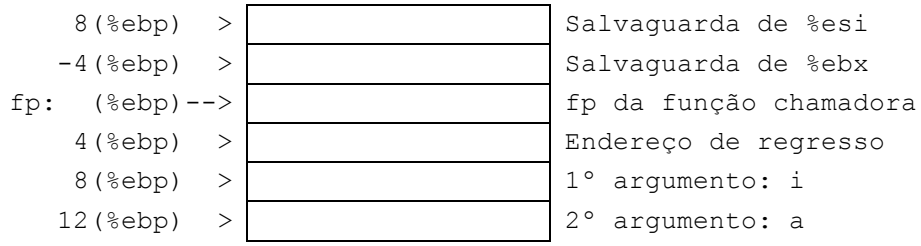
```
int soma_grandes (int n, int *a) {
  int i, r=0;

  for (i=0 ; i<n ; i++) {
    if (a[i] > 1000) r += a[i];
  }
  return r;
}
```

2. Alocação de registos

Variável	Reg	Obs.
`r`	`%eax`	Como se trata do valor a devolver pela função, e como o mecanismo utilizado para o fazer é o registo `%eax`, usa-se desde logo este registo
`i`	`%ecx`	Escolha arbitrária
`a`	`%ebx`	Escolha arbitrária (NOTA: registo *callee saved*)
`n`	`%esi`	Escolha arbitrária (NOTA: registo *callee saved*)
`a[i]`	`%edx`	Escolha arbitrária

3. Quadro de ativação



Algumas considerações:

- eventuais valores associados a registos *caller saved* não são apresentados porque não há hipótese de saber quais os registos que a função que chama `soma_grandes()` eventualmente salvaguarda; quanto aos *callee saved* a ordem vai depender do código...
- as variáveis locais serão armazenadas em registos e não na *stack*;
- os conteúdos das células poderão ser obtidos com o `gdb`.

4. Um código provável da função

```
soma_grandes:

  pushl   %ebp
  movl    %esp, %ebp

  pushl   %ebx                        # salvaguarda os registos ebx
  pushl   %esi                        #                       e esi

  movl    8(%ebp), %esi               # %esi = n (1º arg)
  movl    12(%ebp), %ebx              # %ebx = *a (2ª arg)

  xorl    %eax, %eax                  # r = 0
  xorl    %ecx, %ecx                  # i = 0

TESTE:
  cmpl    %esi, %ecx
  jge     FIM_CICLO                   # se i >= n sai do ciclo for

  movl    (%ebx, %ecx, 4), %edx       # %edx = a[i]
  cmpl    $1000, %edx
  jle     FIM_IF                      # se a[i] <= 1000 salta por cima da adição

  addl    %edx, %eax                  # r += a[i]

FIM_IF:
  incl    %ecx                        # i++
  jmp     TESTE                       # repete o ciclo for

FIM_CICLO:
  popl    %esi                        # recupera os registos esi
  popl    %ebx                        #                     e ebx

  leave
  ret
```