Trabalho para casa nº 3

Funções e procedimentos: IA32

1. Introdução

Pretende-se com este trabalho que o aluno seja capaz de interpretar o *assembly* gerado pelo `gcc` para a arquitectura IA32, nomeadamente no que respeita à invocação de funções e criação da *stack frame*.

2. Linguagem de alto nível

Escreva em C, usando o editor de texto que considerar mais adequado, o seguinte programa:

prog.c

```
typedef struct {
  char S;
  int idade;
} Telem;

int copia (Telem *o, Telem *d, int n)
{
  int ind=0;

  for ( ; n>0 ; n--)
  {
        if (o[n-1].S==1)
        {
              d[ind].S=1;
              d[ind].idade=o[n-1].idade;
              ind++;
        }
  }
  return (ind);
}
```

Note que a função `copia()` copia os elementos do array `o` que tiverem o campo `S`=1 para o array `d`. O parâmetro `n` indica o número de elementos de `o`.

3. Compilação

Compile o programa `prog.c` usando o comando

`gcc -S prog.c`

e copie o ficheiro `prog.s` para `prog.O0.s`

Questão 1 – Qual o factor de escala para os *arrays* do tipo de dados `Telem`?

Questão 2 – Quais os deslocamentos relativamente a `%ebp` de cada um dos parâmetros e variáveis locais?

Questão 3 – Descreva os mecanismos usados para calcular os endereços de `o[n-1].S`, `o[n-1].idade`, `d[ind].S` e `d[ind].idade`.

Compile de novo o programa `prog.c` usando o comando

`gcc –O1 –S prog.c`

e copie o ficheiro `prog.s` para `prog.O1.s`

Questão 4 – Quais as optimizações introduzidas pelo compilador relativamente à versão anterior?

Questão 5 – Descreva os mecanismos usados para calcular os endereços de `o[n-1].S`, `o[n-1].idade`, `d[ind].S` e `d[ind].idade`.

Questão 6 – Porque é que os registos `%edi`, `%esi` e `%ebx` são guardados na *stack* no início da função e restaurados no fim, não se fazendo o mesmo para os registos `%eax`, `%ecx` e `%edx`?

4. Linguagem de alto nível

Escreva em C, usando o editor de texto que considerar mais adequado, o seguinte programa:

prog1.c

```
typedef struct {
  char S;
  int idade;
} Telem;

int copia (Telem *o, Telem *d)
{
  int ind=0, n;

  for (n=0 ; n<5 ; n++)
  {
        if (o[n].S==1)
        {
              d[ind].S=1;
              d[ind].idade=o[n].idade;
              ind++;
        }
  }
  return (ind);
}
```

Note que a grande diferença entre esta versão da função `copia()` e aquela apresentada na secção 2 é que o número de elementos a examinar do array `o` é fixo e conhecido em tempo de compilação.

5. Compilação

Compile o programa `prog.c` usando o comando

`gcc –S –O1 prog1.c`

e copie o ficheiro `prog1.s` para `prog1.O1.s`

Compile de novo o programa `prog.c` usando o comando

`gcc –S –O1 –funroll-all-loops prog1.c`

e copie o ficheiro `prog1.s` para `prog1.unroll.s`

Questão 7 – Quais as optimizações introduzidas pelo compilador na versão com loop-unroll relativamente à outra versão?

Questão 8 – Consegue encontrar a variável `n` no código *assembly*? Como é que o seu papel é implementado?

Questão 9 – Quais lhe parecem ser as vantagens desta técnica? A sua eficácia aumenta ou diminui com o número de iterações do ciclo?

Questão 10 – Qual a maior desvantagem desta técnica?

NOTA: A secção 5.8 do livro "Computer Systems: A Programmer's Perspective", do Bryant and Hallaron apresenta um estudo mais elaborado da técnica de *loop-unrolling*, mesmo quando o número de iterações do ciclo não é conhecido em tempo de compilação. A 5.10.1 do mesmo livro e a secção 6.8 do livro "Computer Organization & Design: The Hardware/Software Interface" do Patterson and Hennessy descrevem uma técnica complementar, conhecida como *loop splitting*, para o caso de processadores com organizações superescalares e com *pipeline*.