10º Exercício Prático
Desenvolvido no Laboratório

Objetivo

Comparar instruções semelhantes em linguagem de montagem e verificar sua influência no tempo de execução.

Materiais

1. Compilador GCC
2. Arquivo SoRestaAzul.c
3. Imagem Lapis.ppm

Introdução

As instruções em linguagem de montagem de um processador tem tempos de execução diferentes. Esses tempos dependem da construção em hardware que precisam de um número de ciclos de *clock* para serem realizadas. Como muitas instruções realizam operações diferentes a escolha das instruções com menor tempo de execução influenciam no tempo de execução do programa.

O exercício de hoje consiste em separar uma cor primária de uma imagem utilizando um modo bem simples e ingênuo. Neste modo considera-se que, em *pixel* de uma imagem RGB, se um canal *x* tem valor maior que a média dos outros dois canais, então *x* é a cor do *pixel*.

Desenvolvimento

O programa a ser otimizado gera uma imagem copiando apenas os *pixels* de cor azul, os outros *pixels* são colocados como preto. As otimizações são em linguagem de montagem, a primeira apresenta o código base, na segunda altera apenas uma instrução substituindo uma instrução mais simples e rápida e, finalmente, na última, o tamanho dos registradores são alterados de forma que as instruções alteradas consigam ser executadas mais rapidamente.

Executando o programa sem otimização

1. Compile o programa SoRestaAzul.c
   - gcc -masm=intel -g -O2 SoRestaAzul.c -o SoRestaAzul
2. Execute o programa.
   - ./SoRestaAzul

   Qual foi o tempo observado na execução?

   Tempo 1: _____ s

Otimização usando linguagem de montagem

Nesta otimização os dois for's foram mantidos em linguagem C, mas o processamento de cada *pixel* foi escrito em linguagem de montagem. Cada um dos canais são lidos em transformados pela instrução **`movzx`** em dois bytes para comportar a soma dos dois canais R e G.

1. Abra o arquivo SoRestaAzul.c em um editor de texto
2. Altere a função processa() como código abaixo

```
void processa(struct Pixel img[ALTU_IMG][LARG_IMG],
              struct Pixel imgSai[ALTU_IMG][LARG_IMG]) {
  int i, j;
  for (i = 0; i < ALTU_IMG; i++) {
    for (j = 0; j < LARG_IMG; j++) {
      asm("   movzx ax,  byte ptr %[imgr]  \n"
          "   movzx bx,  byte ptr %[imgg]  \n"
          "   movzx cx,  byte ptr %[imgb]  \n"
          "   add   ax,  bx                \n"
          "   mov   ebx, 0                 \n"
          "   shr   ax,  1                 \n" // divide por 2
          "   cmp   ax,  cx                \n"
          "   jg    rot%=                  \n"
          "   mov   ebx, dword ptr %[imgr] \n"
          "rot%=:                          \n"
          "   mov %[imgSai], ebx           \n"
          :
          : [imgr] "m"(img[i][j].r), [imgg] "m"(img[i][j].g),
            [imgb] "m"(img[i][j].b), [imgSai] "m"(imgSai[i][j])
          : "eax", "ebx", "ecx");
    }
  }
}
```

Qual foi o tempo observado na execução?

Tempo 2: _____ s

Substituindo uma operação por outra mais rápida

Instruções que usam apenas registradores, sem outro tipo de dado, tem maior possibilidade de executarem em menor tempo. Nesta otimização, trocamos apenas a instrução **`mov ebx, 0`** pela instrução **`xor ebx, ebx`** , como ela é mais simples e operando apenas registradores, ela é executada mais rapidamente.

1. Altere a função processa() como código abaixo que opera com inteiros:

```
void processa(struct Pixel img[ALTU_IMG][LARG_IMG],
              struct Pixel imgSai[ALTU_IMG][LARG_IMG]) {
  int i, j;
  for (i = 0; i < ALTU_IMG; i++) {
    for (j = 0; j < LARG_IMG; j++) {
```

```
      asm("   movzx ax,  byte ptr %[imgr]  \n"
          "   movzx bx,  byte ptr %[imgg]  \n"
          "   movzx cx,  byte ptr %[imgb]  \n"
          "   add   ax,  bx                \n"
          "   xor   ebx, ebx               \n"
          "   shr   ax,  1                 \n"
          "   cmp   ax,  cx                \n"
          "   jg    rot%=                  \n"
          "   mov   ebx, dword ptr %[imgr] \n"
          "rot%=:                          \n"
          "   mov %[imgSai], ebx           \n"
          :
          : [imgr] "m"(img[i][j].r), [imgg] "m"(img[i][j].g),
            [imgb] "m"(img[i][j].b), [imgSai] "m"(imgSai[i][j])
          : "eax", "ebx", "ecx");
    }
  }
}
```

Qual foi o tempo observado na execução?

Tempo 3: _____ s

Alterando o tamanho dos registradores

Instruções com número menor de byte de *opcode* e com registradores menores (com menos bits) têm potencial para serem executadas mais rapidamente, porém, não é o que acontece nesse código. As instruções de 32 bits são otimizadas para serem executadas mais rapidamente nos processadores modernos usados em computadores pessoais. No programa abaixo as instruções que utilizam registradores de 16 bits foram substituídas por instruções que usam registradores de 32 bits. Faça a alteração para verificar esta afirmação:

1. Altere a função processa() como código abaixo:

```
void processa(struct Pixel img[ALTU_IMG][LARG_IMG],
              struct Pixel imgSai[ALTU_IMG][LARG_IMG]) {
  int i, j;
  for (i = 0; i < ALTU_IMG; i++) {
    for (j = 0; j < LARG_IMG; j++) {
      asm("   movzx eax, byte ptr %[imgr]  \n"
          "   movzx ebx, byte ptr %[imgg]  \n"
          "   movzx ecx, byte ptr %[imgb]  \n"
          "   add   eax, ebx               \n"
          "   xor   ebx, ebx               \n"
          "   shr   eax, 1                 \n"
          "   cmp   eax, ecx               \n"
          "   jg    rot%=                  \n"
          "   mov   ebx, dword ptr %[imgr] \n"
          "rot%=:                          \n"
          "   mov %[imgSai], ebx           \n"
          :
```

```
            : [imgr] "m"(img[i][j].r), [imgg] "m"(img[i][j].g),
              [imgb] "m"(img[i][j].b), [imgSai] "m"(imgSai[i][j])
            : "eax", "ebx", "ecx");
      }
    }
  }
```

Qual foi o tempo observado na execução?

Tempo 4: _____ s

O compilador “sabe” qual instrução tem menor tempo de execução, assim quando ele transforma o código em linguagem C para linguagem de máquina ele usa as melhores instruções. A vantagem do humano escrever o código em linguagem de montagem é fazer operações que não são permitidas ou reconhecidas pelo compilador. Observe que o tempo 4 bem é inferior ao tempo 1, o compilador otimiza bem o código e frequentemente supera o ser humano, mas no caso, ele não percebe que escrever 4 *bytes* a cada *pixel* não afeta a imagem de saída, mas reduz o tempo de execução. A otimização apresentada em linguagem de montagem neste laboratório escreve 4 bytes por vez, ou seja, RGB do *pixel* atual e mais um *byte* de sujeira no próximo *pixel*, porém isso não é um problema pois este *byte* de sujeira será sobreposto na escrita do próximo *pixel*.

Analisando os resultados

Envie a avaliação dos resultados como descrito no arquivo “Avaliacao Dos Resultados.pdf”.

Conclusão

Escolher as melhores instruções e registradores pode ter influência no tempo de execução de um programa.