Organização de computadores – TP I

Luiz Felipe Couto Gontijo – 2018054524

Universidade Federal de Minas Gerais

Objetivo:

A partir da entrada de um número de até quatro dígitos, o objetivo do trabalho é desenhar esse número (respeitando a forma dos caracteres de displays de LCD) no Bitmap Display do Mars. Portanto, o programa deve ser feito em MIPS assembly.

Implementação:

O código implementado para a realização do trabalho pode ser dividido em partes:

- Leitura do número digitado pelo usuário.
- Verificação dos dígitos e de quantos são.
- Desenho na tela.

➔ Leitura do número

Após o início do primeiro loop (na *label loop*), todos os registradores que são utilizados são zerados. Após isso, é feita a chamada de sistema para ler o número. Caso o número seja menor que 0, é feito um jump para a *label exit*, que finaliza a execução do programa. Caso contrário, esse valor é passado para o registrador $s7.

➔ Verificação dos dígitos e de quantos são

Para verificar quais são os dígitos, é feita a sucessiva divisão do número digitado por 10. Quando o quociente for menor que zero, essa etapa acaba e é dado o comando de jump para a *label imprime_primeiro.* Os restos(que correspondem aos dígitos) são guardados em registradores do tipo 's' . Toda vez que é feita a divisão por 10, o contador de dígitos($v0) aumenta de um.

➔ Desenho na tela

Parte principal do programa. Para realizar o desenho, primeiro foram definidas duas funções, rectangleVertical e rectangleHorizontal, que desenham um retângulo vertical e horizontal, respectivamente. Ambas as funções recebem 4 parâmetros:

- $a0 é o x_min (borda da esquerda)

- $a1 é a largura (deve ser positiva)

- $a2 é o y_min (borda superior)

- $a3 é altura (deve ser positiva)

A partir daí, é definido um registrador temporário para desenhar da esquerda para a direita($t3). A cada iteração, esse registrador aumenta seu valor de 4(para ir para o próximo pixel) e só para quando atinge a largura máxima. Daí, passamos para a próxima linha(definindo $a2 para a próxima linha), redefinimos $t3 e repetimos o processo. Isso é feito até se atinja a altura previamente definida(ou seja, $a2 = $a3). Para desenhar o retângulo horizontal, basta trocar os valores da largura e altura antes de começar o processo. Desse modo, temos as funções que desenham os retângulos que vão compor cada um dos algarismos.

Para desenhar os dígitos de 0 a 9, foi definido uma label para cada um deles(*label desenha_*número**). A *desenha_um* por exemplo, desenha primeiro um retângulo vertical, depois redefini os valores dos argumentos das funções de *rectangle*(para desenharmos o retângulo em outro local) e chamamos novamente a função de desenhar um retângulo vertical.

```
jal rectangleVertical      #Chama a função para desenhar um retângulo vertical
    addi $s0,$s0,0
    addi $s2,$s2,63
    move $a0,$s0           Redefinimos os parâmetros que vamos
    move $a1,$s1           passar para a função de desenho.
    move $a2,$s2
    move $a3,$s3
jal rectangleVertical      #Desenhamos novamente um retângulo vertical, agora
                           embaixo do anterior.
```

Fazemos isso para todos os algarismos, e cada um deles possui sua própria configuração de desenho, mas sempre seguindo alguns padrões para que tudo fique bem alinhado.

Desse modo, agora é possível desenhar os algarismos separados. Para fazer com que saibamos onde começar a desenhar cada algarismo do número (no caso do número possuir mais de um digito) foram definidas mais quatro *labels – desenha_primeiro, desenha_segundo, desenha_terceiro, desenha_quarto.* Cada uma delas se refere a um dos dígitos que o número **pode** possuir(da direita para a esquerda). Quando entramos na *label desenha_primeiro* , verificamos qual o digito correspondente ao primeiro algarismo (que está armazenado em um dos registradores desde a etapa de leitura), e daí chamamos a função para desenha-lo. Após isso, verificamos se a quantidade de dígitos é = 1, e, caso seja, voltamos ao loop inicial para ler o próximo número. Caso contrário, vamos para a *label desenha_segundo*. Nela, são redefinidos os argumentos que desenharão futuramente os retângulos, para que eles sejam desenhados à esquerda do digito desenhado anteriormente. Então desenhamos o algarismo, verificamos a quantidade de dígitos e, caso não seja = 2, vamos para a *desenha_terceiro.* Fazemos

esse processo até que se atinja o número de dígitos ou até que cheguemos ao *desenha_quarto*, que com certeza será o último. Voltamos então, para o loop inicial para ler o próximo número.

Assim, temos todos os algarismos do número desenhados na tela, e portanto, temos o número!

➔ Limpando a tela

Para realizar o processo de limpeza da tela, ou seja, fazer com a tela fique toda preta novamente e pronta para desenhar o próximo número, chamamos a função *cleanScreen*, que simplesmente desenha quatro oitos pretos (nos mesmos padrões das outras funções de desenho) exatamente nas posições onde estão os dígitos do número que acabou de ser desenhado. Dessa forma, independente do número que foi desenhado anteriormente, cobrimos ele de preto, limpando a tela.

Alguns testes e conclusão:

O teste realizado baseia-se no tempo de execução e de resposta do software para diferentes tipos de entradas. Lembrando que para cada desenho de número, também temos uma limpeza de tela(que desenha 4 oitos pretos).

- Para o algarismo menos custoso para desenhar, o algarismo número 1,(que possui apenas dois segmentos de "retângulo") leva 0.2 segundos para ser desenhado.
- O algarismo mais custoso, o oito, leva 0.22 segundos. A diferença é bem pequena(praticamente imperceptível) para o caso anterior.
- Para o último teste, desenhamos o número mais custoso, que é o 8888. Ele também não destoa dos demais testes, completando em 0.24 segundos.

É plausível concluir que o algoritmo utilizado para realizar o trabalho é eficiente, e cumpre o seu objetivo. Porém, devido a estrutura da linguagem MIPS Assembly, que aceita apenas 3 'argumentos/variáveis/registradores' por comando, o código ficou bastante extenso (no que diz respeito ao número de linhas).

Bibliografia:

https://virtual.ufmg.br/20191/mod/folder/view.php?id=33381 – (slides da disciplina)

Organização e Projeto de Computadores, David A. Patterson, John Hennessy

https://gustavus.edu/mcs/max/courses/F2013/MCS-284/labs/lab2/rectangle.asm