**Trabalho 1 - Organização e Arquitetura de Computadores**

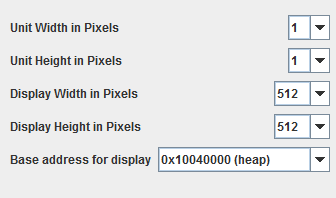
**Nome:** Miguel Barreto Rezende Marques de Freitas

**Matrícula:** 12/0130424

**Turma:** C

**Objetivos:** Este trabalho visa a prática de assembly MIPS e a ambientação com o simulador MARS, fazendo um programa que leia uma imagem bitmap colorida em padrão RGB de um arquivo e carregue no Display Bitmap do MARS, podendo eliminar (zerar) totalmente cada componente de cor separadamente.

**Configuração do Display Bitmap:**



**O display Bitmap tem que estar configurado nessas condições e conectado para que o programa funcione.**

**Módulos e rotinas implementados:**

* **main:**

É onde começa a execução do programa e mantém o loop de execução deste, fazendo chamada a outras subrotinas como **menu** e **switch**. Cuida de receber o retorno da rotina **menu** e passar o argumento para a rotina **switch**, e mantém um fluxo de controle que repete o menu enquanto for necessário. As chamadas de subrotinas são feitas com a instrução *jal*, guardando o endereço PC + 4 no registrador *$ra*.

* **menu:**

Não recebe nenhum argumento, mas oferece um valor de retorno de acordo com a entrada do usuário. Esta rotina faz chamadas de sistema para mostrar as opções na tela e para receber a escolha do usuário como entrada. Retorna o valor informado pelo usuário como retorno em $v0, e volta para a instrução seguinte depois de sua chamada utilizando *jr $ra* para voltar para o endereço armazenado PC + 4.

* **switch:**

Recebe de argumento em $a0 o valor da escolha do usuário, e então desvia o fluxo do programa através de um salto condicional de acordo com o valor do argumento, utilizando a instrução *beq*. O valor de $ra é atualizado depois que a rotina **switch** é chamada na **main** através da instrução *jal*, guardando o endereço da instrução seguinte.

* **abre\_arq** e **load\_img**:

Por motivos de organização, a rotina foi separada nesses dois labels, porém na prática **load\_img** é a continuação direta de **abre\_arq**. O arquivo *lena.bmp* é aberto pelo programa em **abre\_arq**, desviando o fluxo para a rotina **erro** caso algo dê errado. Se der certo, mensagens de confirmação aparecem na tela e a execução continua para **load\_img**, onde ocorre toda a leitura do arquivo, conversão para o formato de pixel de 4 bytes do MARS e carregamento da imagem na região *heap* de memória associada ao display bitmap, mostrando a imagem na tela.

No começo de **load\_img** se atribui ao registrador *$s2* o endereço base da heap, ao registrador *$s1* o endereço base do espaço de memória de 3 bytes chamado *buffer* e ao registrador *$s3* o número 262144, que é o número de pixels da imagem (512x512). Depois disso, as configurações iniciais de leitura de arquivo são feitas, e então se entra em um loop que ocorrerá 262144 vezes, percorrendo todos os pixels da imagem.

No começo do loop (representado por um label chamado *loop*) é feita a chamada de sistema para ler 3 bytes com informações RGB de um pixel do arquivo, depois cada byte desses é carregado em um registrador temporário diferente através da instrução *lb (load byte)*. Desta forma, ficam os três registradores *$t1*, *$t2* e *$t3*, respectivamente com as informações: 0x000000RR, 0x000000GG e 0x000000BB. Através do uso da instrução *sll*, é feito um *shift left* de 16 bits no *$t1* para que este fique 0x00RR0000 e um *shift left* de 8 bits no *$t2*para que este fique 0x0000GG00. Depois disso, os três registradores são somados (através de dois usos da instrução *add*) em um registrador *$t4*, que então passa a armazenar as informações no formato 0x00RRGGBB, adequado para o MARS. Com as informações necessárias para o pixel em uma word de 4 bytes armazenada em *$t4*, é usada a instrução *sw (store word)* para armazenar o valor de *$t4* na *heap*.

Ao fim do loop, são adicionados 4 bytes ao endereço base da *heap* com a instrução *addi*, para que se avance para o próximo pixel na próxima iteração, e então se adiciona -1 ao contador decrescente armazenado em *$s3*, e é feita um teste condicional com a instrução *bne* que pula de novo para o label do *loop* enquanto o contador não chegar a 0.

Quando o loop acaba, se chega no comando de retorno *$jr $ra*, que vai voltar o fluxo do código para a **main**, logo após a chamada da rotina **switch**.

* **elimina:**

O algoritmo da rotina **elimina** é utilizado para todos os casos, e a rotina recebe de argumento em *$a0* a constante a ser utilizada para zerar a componente em questão em cada um dos pixels através de um *AND*, sendo com o imediato 0x0000ffff para zerar a componente vermelha, 0x00ff00ff para zerar a componente verde e 0x00ffff00 para zerar a componente azul. O argumento a ser mandado para a função é de acordo com o fluxo de controle que a rotina **switch** faz em função da escolha do usuário na rotina **menu**.

No começo do loop da rotina **elimina** é carregado um pixel do arquivo de 4 bytes em *$t5*, e depois é feito um *and* de *$*t5 com *$a0* para zerar a componente correspondente, e depois é armazenado armazenando novamente em *$t5*. Após isso, o valor atualizado de *$t5*, com a componente correspondente zerada, é carregado novamente na *heap* através da instrução *sw*.

Ao fim da iteração, com usos da instrução *addi*, 4 é adicionado ao registrador que guarda o endereço da *heap* para ir para o próximo pixel e -1 é adicionado para decrescer o contador, e uma comparação com *bne* é feita para voltar para o começo do loop se o contador ainda não estiver zerado.

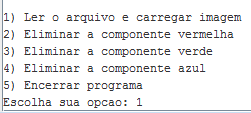
Quando o loop acaba, a imagem já está totalmente atualizada no Display Bitmap com a componente de cor a menos. O comando de retorno *$jr $ra* é usado para voltar para a **main**, na instrução que fica logo após a chamada do **switch**.

* **encerra:**

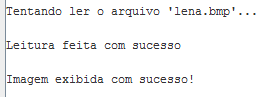
É a rotina que encerra a execução do programa. Mostra uma mensagem de despedida e agradecimentos ao usuário na tela e então faz uma chamada de sistema que encerra o programa.

* **erro:**

Mostra uma mensagem de erro na tela e pula para a instrução **encerra**. Esta rotina só é acionada quando há algum erro de leitura do arquivo *lena.bmp*.



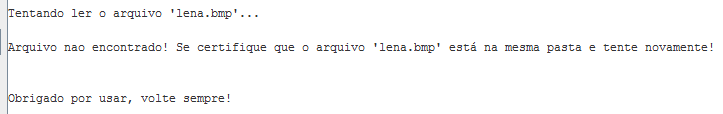
Interação com o usuário através do menu.



Mensagens de sucesso na tela que aparecem à medida que as coisas vão dando certo.



Mensagem que aparece antes do programa ser encerrado.



Sequência do que ocorre quando há erro de leitura do arquivo.

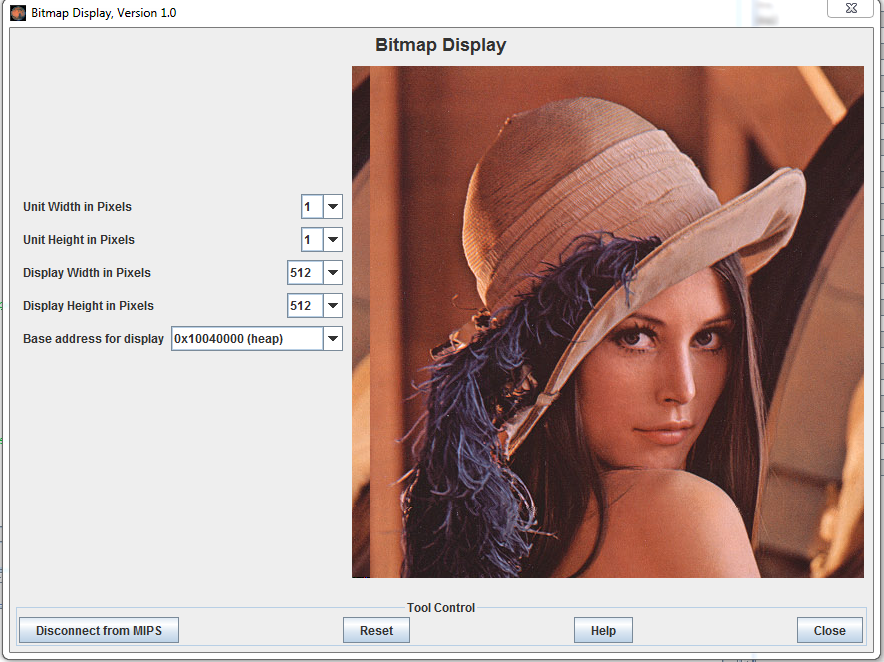


Imagem original exibida no Display Bitmap.

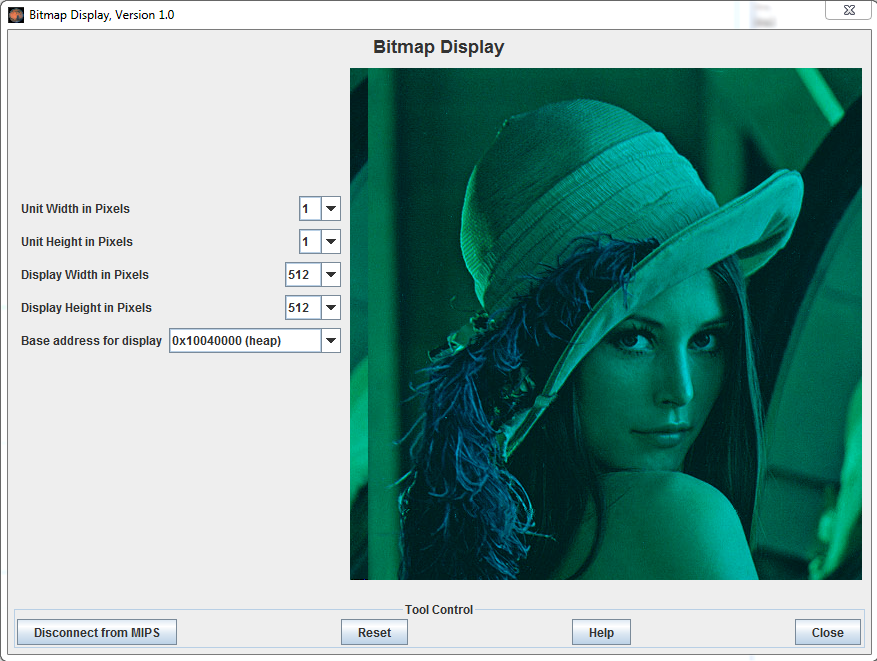


Imagem com apenas a componente vermelha eliminada.

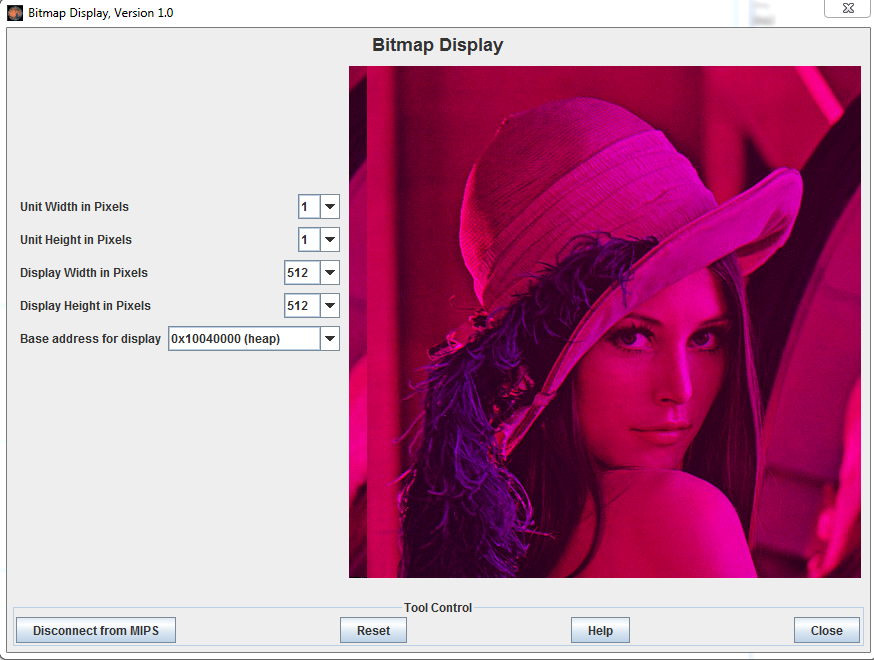


Imagem com apenas a componente verde eliminada.

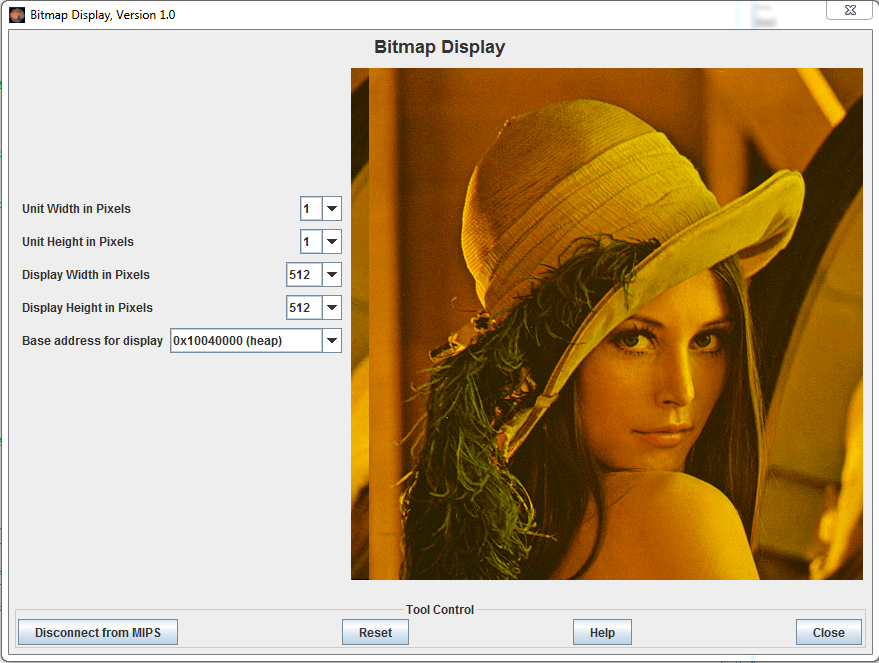


Imagem com apenas a componente azul eliminada.

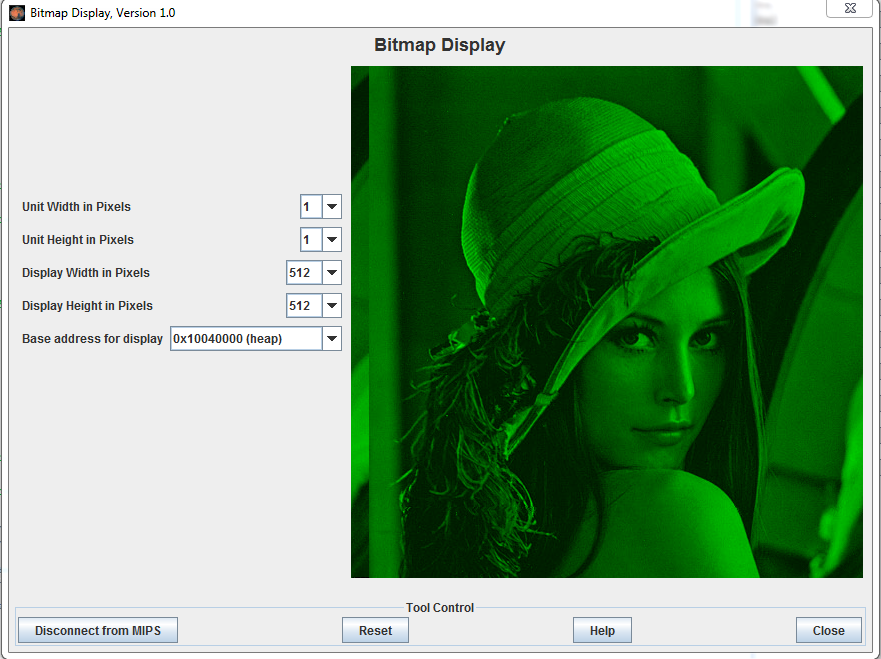


Imagem com as componentes azul e vermelha eliminadas, sobrando apenas a verde.

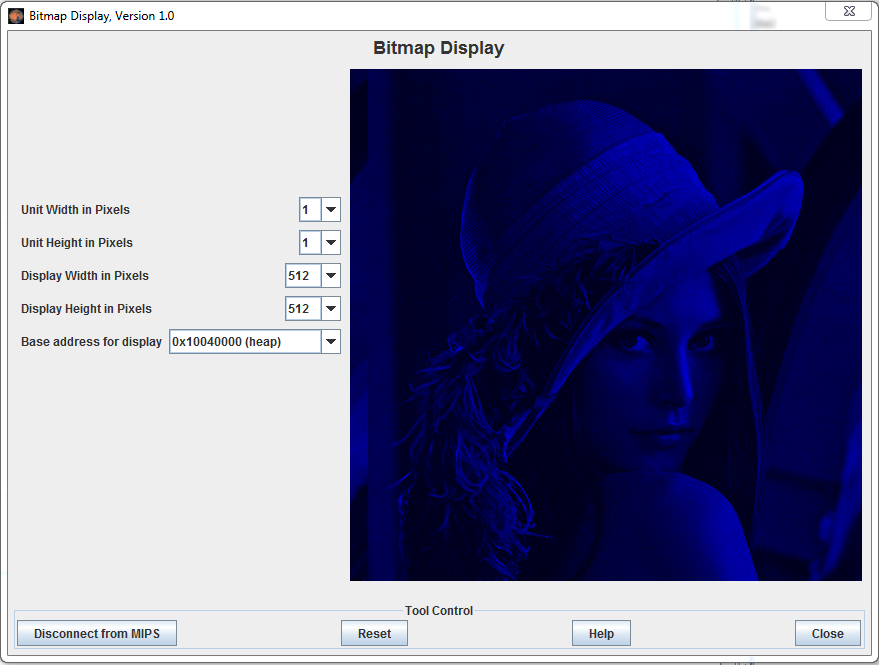


Imagem com as componentes vermelha e verde eliminadas, sobrando apenas a azul.

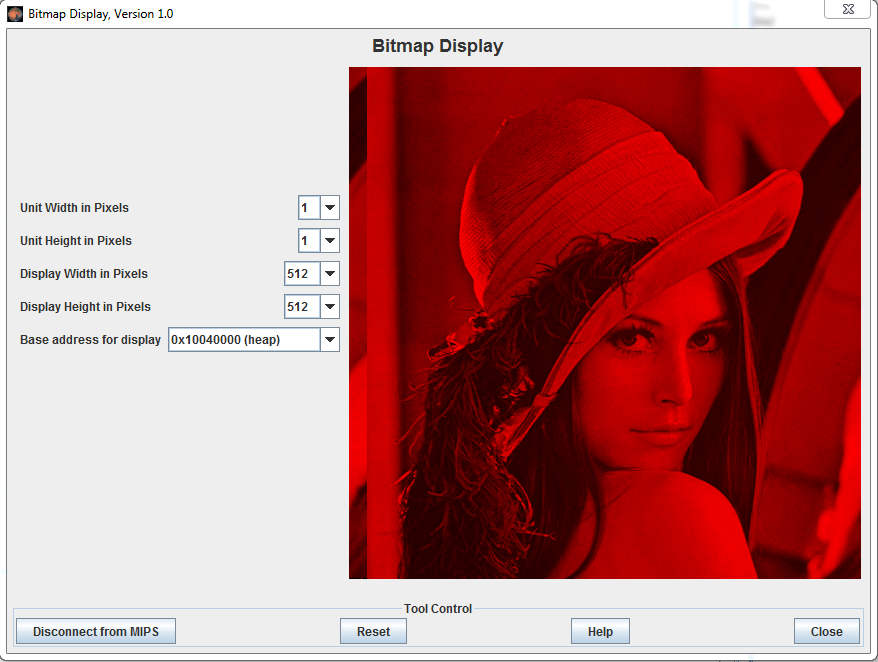


Imagem com as componentes azul e verde eliminadas, sobrando apenas a vermelha.