

## **MARS (MIPS Assembler and Runtime Simulator)**

---

# **Documentação de versão**

---

IDE para Programação em Linguagem Assembly - MIPS

14 de Dezembro de 2017

Fernando dos Santos Soares  
Igor Luciano Magro  
Leonardo Correia  
Maíra Baptista de Almeida

## **Nota**

MARS (MIPS Assembler and Runtime Simulator) é uma IDE para programação em linguagem assembly, destinada ao uso educacional, de código livre. As mudanças abordadas neste documento são destinadas ao uso acadêmico, sem qualquer intenção para fins comerciais. A divulgação e uso desta ferramenta são livres, podendo ser manipulados de acordo com as notas dos autores Pete Sanderson e Kenneth Vollmar, da Universidade *Missouri State*.

## Sumário

1	Funcionalidades adicionadas.....	2
1.1	Atualização manual do bitmap display .....	2
1.2	Criação de cache para o bitmap display.....	3
1.3	Reprodutor de áudio.....	5
2	Aprimoramentos da versão 4.5.....	8
2.1	Bitmap Display .....	8
2.2	Keyboard and Display MMIO Simulator .....	10
3	Correção de Bugs .....	12
3.1	Deadlock .....	12

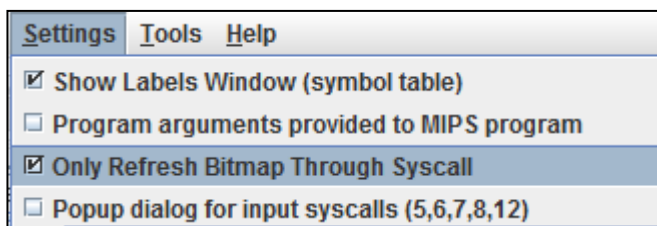
# 1 Funcionalidades adicionadas

## 1.1 Atualização manual do bitmap display

Até então, ao se utilizar a ferramenta do bitmap display, o MARS tenta sincronizar cada mudança ocorrida na memória, com a visualização que foi configurada no bitmap. Desta forma, quando começamos a pintar muitas coisas na tela, percebemos um efeito não muito natural na imagem, que oscila entre as cores que estão sendo trocadas rapidamente.

A solução foi a implementação de uma chamada para o sistema, para que o programador possa escolher quando ele deseja que a tela seja atualizada. Sendo assim, ele pode desenhar diversas coisas de uma vez, e só então, solicitar a atualização, o que torna muito mais suave as transições de desenho e tela.

É importante ressaltar que esta modificação é completamente opcional, ou seja, foi desenvolvido na interface, uma opção para que o usuário possa optar por usar ou não esta funcionalidade.



Ao acessarmos o menu “Settings”, temos agora uma opção chamada “*Only Refresh Bitmap Through Syscall*”, na qual, quando ativada, o MARS não fará mais a atualização do bitmap para cada mudança

na memória, e sim, irá aguardar uma chamada de sistema para que isso ocorra. Caso esta opção esteja desmarcada, irá seguir o comportamento normal do MARS, e a chamada de sistema, se for feita, não surtirá nenhum efeito.

Para o uso desta funcionalidade o programador deve:

```
li $v0, 39
syscall
```

Na seção de “Help” do MARS, foi incluída a documentação do syscall 39, referente a atualização manual do bitmap display.

Basic Instructions	Extended (pseudo) Instructions	Directives	Syscalls	Exceptions	Macros
		\$a1 = ID of cache			
refresh bitmap	39		Refresh Manually Bitmap. Enable the Option "Only Refresh Bitmap Through Syscall" in Settings Menu		

## 1.2 Criação de cache para o bitmap display

Esta funcionalidade é interessante principalmente para aqueles que desejam utilizar uma resolução um pouco mais alta para implementar algo no bitmap display. Isso, pois quanto maior a resolução escolhida, mais blocos de pixels existem na tela para serem pintados, e quando nos deparamos com um cenário em que precisamos trocar completamente o que está na tela, ter muitos blocos pode ser um problema em questão de velocidade.

A solução foi a criação de um cache para o display, ou seja, o código em assembly não precisa desenhar as mesmas coisas, toda vez que executar o programa. Basta solicitar uma chamada de sistema para que o que está na memória/tela seja salvo num arquivo, que pode ser rapidamente lido quando necessário. Isso torna as transições de tela quase que instantâneas, e novamente, é uma funcionalidade opcional, que pode ou não ser aderida pelo programador.

Temos então, um exemplo de implementação utilizando o cache:

```
li $v0, 38
li $a0, 1
li $a1, 14
syscall
```

Vemos, que o syscall 38 será destinado para o cache do bitmap, sendo que temos dois argumentos:

- \$a0: deve ser enviado o tipo da ação a ser feita, no caso enviar o número 1 significa “SET”, ou seja, criar um cache.
- \$a1: deve ser enviado um ID que represente o cache, no exemplo acima, estamos criando um cache com ID = 14. Sendo assim, podemos mais tarde recuperar algum cache através deste ID que estamos determinando agora.

Temos agora, um exemplo de recuperação de um cache:







```
li $v0, 38
li $a0, 0
li $a1, 14
syscall
```

Neste exemplo, continuamos com syscall 38, mas agora o valor de \$a0 é 0, o que significa “GET”, ou seja, recuperar um cache, que no caso, estamos recuperando o cache com ID = 14.

Caso tentemos recuperar um cache inexistente, o registrador \$v0 é definido como -1, logo podemos fazer verificações como, se não existir cache, faça algo, e assim por diante.

Outro ponto a ser ressaltado, é que ao realizar o syscall para recuperar o cache, caso o cache realmente exista, ele é colocado instantaneamente na tela, não precisando de outras chamadas.

Internamente, é criada uma pasta com o nome de “cache”, localizada na mesma pasta em que o MARS foi executado. Desta forma, é possível apagar manualmente estes arquivos, se necessário. Cada arquivo tem o nome do ID enviado via código.

 0.bitmapCache	03/12/2017 06:47	Arquivo BITMAPCACHE	2.186 KB
 1.bitmapCache	14/12/2017 04:34	Arquivo BITMAPCACHE	2.166 KB
 2.bitmapCache	03/12/2017 17:56	Arquivo BITMAPCACHE	2.162 KB
 3.bitmapCache	03/12/2017 17:56	Arquivo BITMAPCACHE	2.162 KB
 4.bitmapCache	03/12/2017 17:56	Arquivo BITMAPCACHE	2.162 KB
 5.bitmapCache	03/12/2017 17:56	Arquivo BITMAPCACHE	2.163 KB

Na seção de “Help” do MARS, foi incluída a documentação do syscall 38, referente a manipulação de cache.

Basic Instructions	Extended (pseudo) Instructions	Directives	Syscalls	Exceptions	Macros
		volume			
bitmap cache	38	\$a0 = type of action type 0 : GET type 1 : SET  \$a1 = ID of cache	Create cache of all bitmap display, the cache is stored in folder "cache", localized in the same folder where Mars was executed.  If cache does not exists, \$v0 is setted to -1		

### 1.3 Reprodutor de áudio

Executar áudio no MARS pode ser uma tarefa um pouco trabalhosa, isso porquê o MARS só reproduz alguns tons e notas musicais, que podem ser combinados e formar uma música. Existem até algoritmos para poder intensificar esse uso nos formatos MIDI, e poder tocar alguma variedade de áudios no MARS.

A solução foi a implementação de uma funcionalidade, que faz com que a maioria dos arquivos de som .wav sejam executados de maneira mais simplificada, bastando seguir alguns passos para isso:

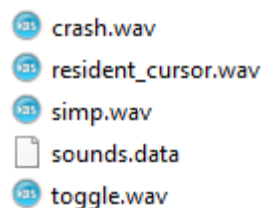
- Primeiramente, ir até a pasta na qual o MARS foi executado, e criar uma pasta chamada de “sounds”, nesta pasta deverão conter os arquivos .wav a serem executados.
- Ainda na pasta “sounds”, um arquivo de configuração que atrelará cada música a um ID deve ser criado, este arquivo deve se chamar “sounds.data”, e deve seguir a seguinte estrutura (exemplo):

```
0=simp.wav  
1=crash.wav  
2=toggle.wav  
3=resident_cursor.wav
```

Ou seja, deve ser escrito primeiramente, um ID que identificará o áudio, seguido de um sinal de “=”, e depois o nome do arquivo a ser executado, sem espaços.

Neste arquivo podem conter comentários para indicar o que significa cada ID. Para escrever comentários, basta colocar o símbolo de “#” antes da linha, assim como já ocorre com o editor do MARS.

Segue um exemplo de como ficaria a pasta “sounds”:



Lembrando que esta pasta deve estar localizada na mesma pasta em que o MARS está sendo executado. O formato .wav é o padrão para reprodução de áudio, mas vale ressaltar que nem todos os tipos de .wav são aceitos.

Tendo configurado a estrutura externa da funcionalidade, podemos então tratar de como ela deve ser chamada pelo MARS.

Segue um exemplo para reprodução de um arquivo de música:

```
li $v0, 37
li $a0, 2
li $a1, 0
li $a2, 1
li $a3, 0
syscall
```

Vemos que o syscall 37 é o syscall reservado para esta funcionalidade, na qual podem ser enviados até 4 argumentos para a manipulação do reprodutor, sendo:

- \$a0: deve ser enviado o ID que foi previamente estabelecido no arquivo “sounds.data”, no caso, é enviado o ID 2, que corresponde ao arquivo “toggle.wav”
- \$a1: deve ser enviado o tipo da ação a ser feita no reprodutor, temos três tipos de ação que podem ser enviadas:
  - \$a1 = 0 significa “PLAY SOUND”, na qual, no exemplo, será executado o arquivo com o ID = 2, sendo “toggle.wav”.
  - \$a1 = 1 significa “STOP SOUND”, na qual será interrompida a execução da música com o ID estabelecido.
  - \$a1 = 2 significa “STOP ALL SOUNDS”, este parâmetro não se relaciona com um ID de música, na qual quando for chamado, serão interrompidas todas as execuções correntes de áudio.
- \$a2: deve ser enviado 0 ou 1, na qual é o parâmetro para identificar se um áudio será executado repetidamente, ou não. Enviar 0 significa que o áudio será executado uma única vez, e enviar 1 significa que o áudio será repetido continuamente.
- \$a3: deve ser enviado, um valor que representará um acréscimo ou decréscimo em decibéis do áudio. Enviar 0 significa que o volume do som original será preservado, e enviar algum inteiro positivo, ou negativo, fará a alteração no volume do arquivo a ser executado.

Segue um exemplo para interromper a execução de um áudio:

```
li $v0, 37
li $a0, 2
li $a1, 1
li $a2, 0
li $a3, 0
syscall
```

No exemplo ao lado, um áudio com ID = 2 será interrompido na execução. Note que os parâmetros \$a2 e \$a3 não serão considerados para este caso, e portanto, são opcionais.

Na seção de “Help” do MARS, foi incluída a documentação do syscall 37, referente a reprodução de áudio.

Basic Instructions	Extended (pseudo) Instructions	Directives	Syscalls	Exceptions	Macros
unsigned					
sound player	37	<p>\$a0 = music Id</p> <p>\$a1 = type of action  type 0 : PLAY SOUND  type 1 : STOP SOUND  type 2 : STOP ALL SOUNDS</p> <p>\$a2 = loop (optional)  1 for repeat sound continually, 0 for play once</p> <p>\$a3 = change volume in decibels (optional)  increase or decrease volume</p>	<p>Sound player, play sounds in .wav format (not all .wav formats)</p> <p>Put your songs in the "sounds" folder, which must be created in the same folder where Mars was run.</p> <p>A "sounds.data" file should be created in "sounds" folder, containing in each line:</p> <p>YOUR_MUSIC_ID_NUMBER=YOUR_SONG_NAME.wav</p> <p>Music ID must be defined in file "sounds.data"</p>		



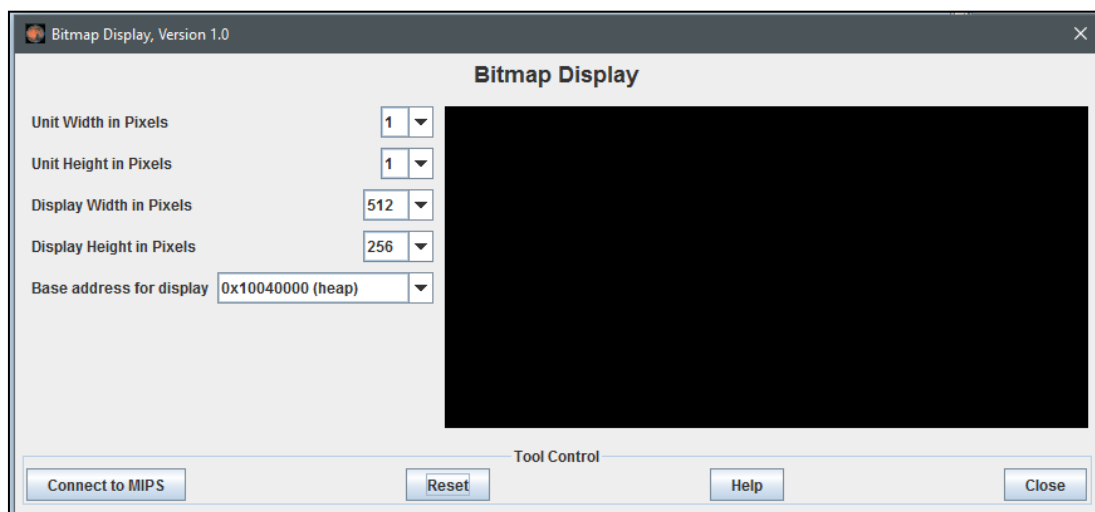
## 2 Aprimoramentos da versão 4.5

Foram alteradas algumas características de algumas ferramentas do MARS para melhorias de interação do usuário, e melhor visualização das informações.

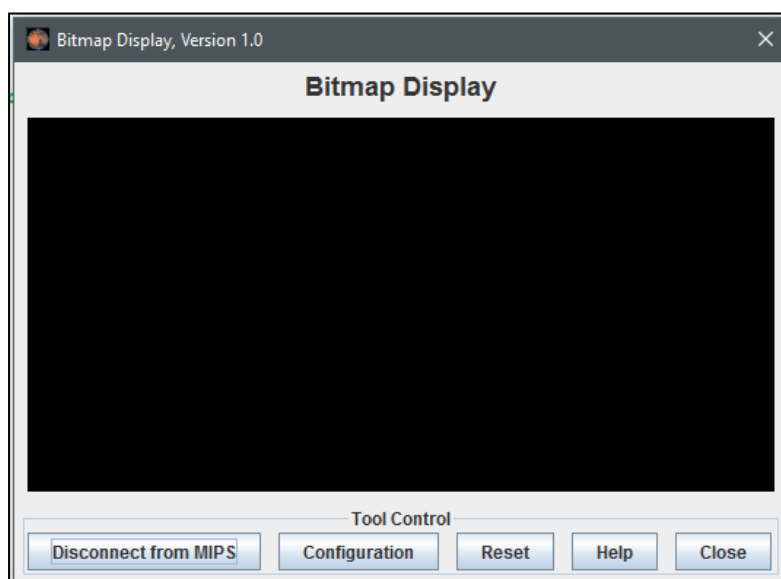
### 2.1 Bitmap Display

Um ponto que atrapalhava em alguns momentos manipular a ferramenta bitmap display, era que existiam componentes de configuração que compartilhavam a tela com o bitmap, o que ocupava muito espaço na tela, principalmente se o bitmap detém de resoluções um pouco grandes.

Abaixo podemos ver como era o bitmap display na versão 4.5 do MARS:

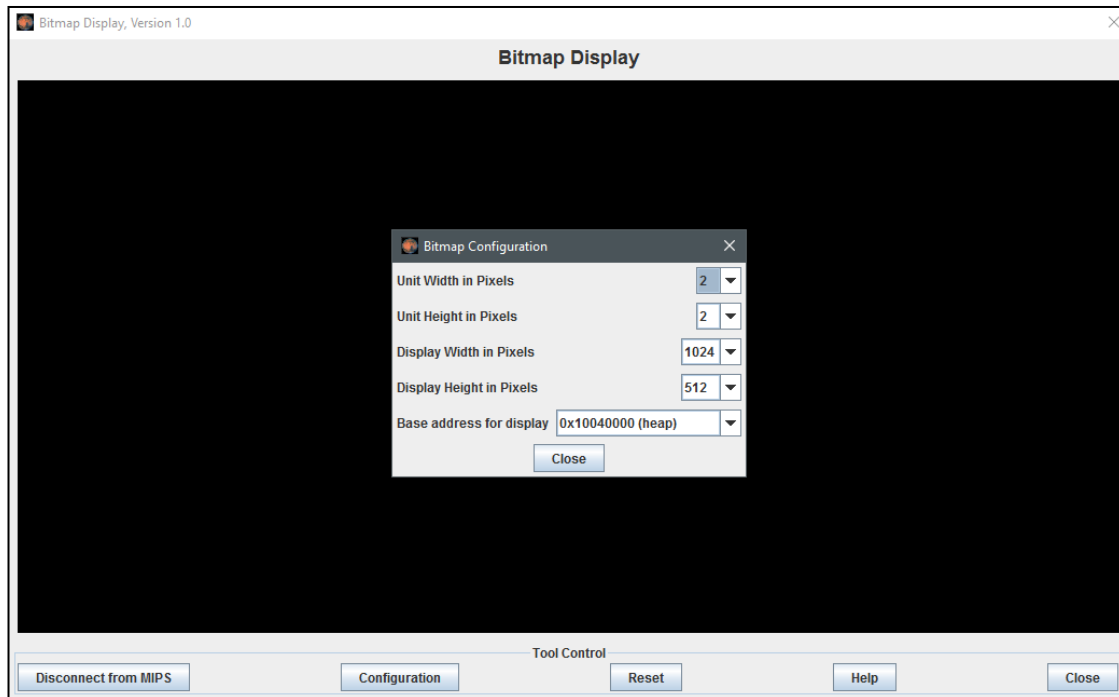


Note que boa parte da tela é tomada pelo menu de configuração do bitmap display, o que impossibilitava o uso de resoluções maiores em monitores não tão robustos. Com as alterações feitas, temos o seguinte resultado:



Note que o menu lateral foi removido, e agora temos a presença de um novo botão na seção “Tool Control”, denominado de “Configuration”. Agora, o menu de configuração fica numa janela separada do bitmap, o que economiza espaço e proporciona o usuário lidar com resoluções de display maiores.

Segue um modelo da nova janela, já numa resolução um pouco mais alta:



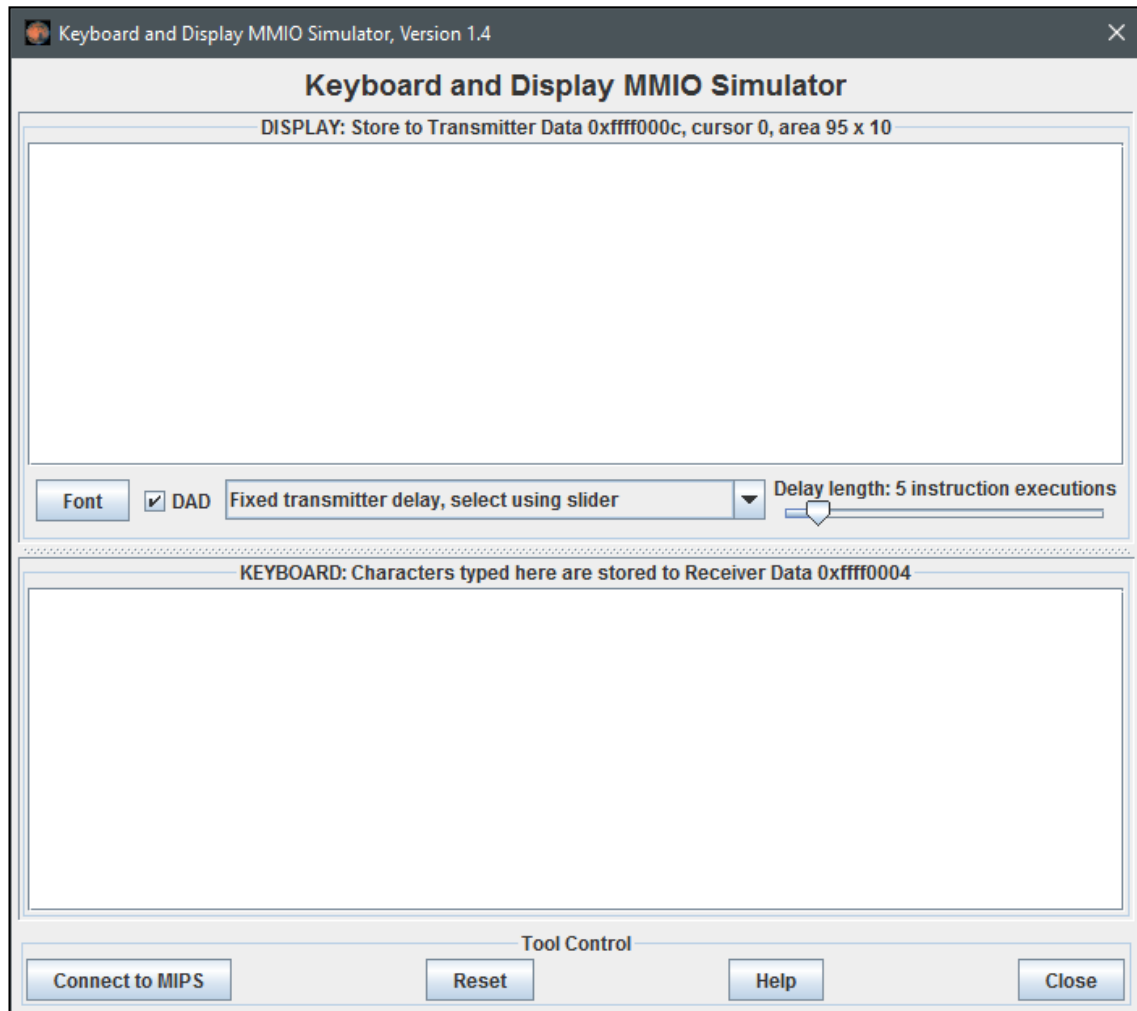
Outro detalhe, é que cada vez que você iniciava o bitmap display, você tinha de configurá-lo da maneira que desejava. Nesta nova versão, se foi implementado preferências de usuário, na qual o MARS irá lembrar a última configuração do bitmap, mesmo que você desligue o seu computador.

Por fim, sempre que o bitmap era iniciado, era necessário se conectar com o MIPS, agora esta conexão é automática, e a cada mudança de configuração a conexão é reestabelecida, não precisando mais interagir com esta conexão. (Ainda é possível conectar e desconectar manualmente, caso seja necessário).

## 2.2 Keyboard and Display MMIO Simulator

A ferramenta mais utilizada para interagir com o input do bitmap display também foi levemente alterada. Um ponto que também atrapalha no keyboard é seu tamanho, pois são duas ferramentas imersas numa só.

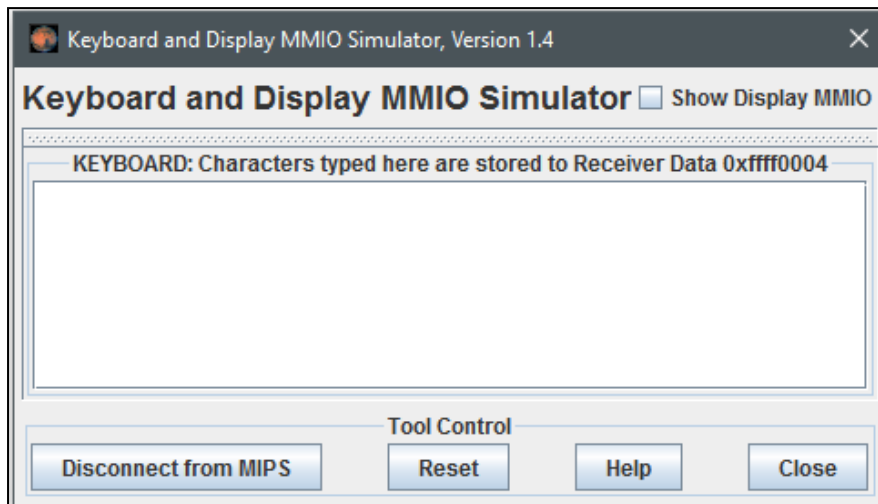
Abaixo, podemos ver a ferramenta na versão 4.5 do MARS:



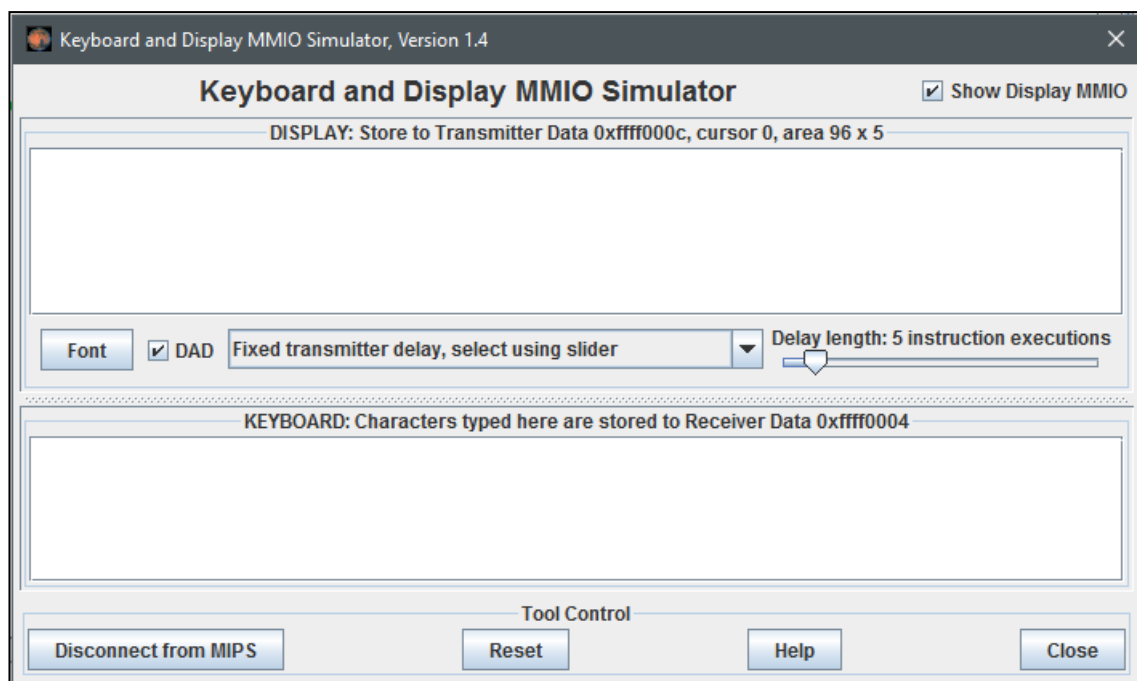
Note a existência das duas ferramentas, em que a primeira é relacionada ao cursor e display, e na maioria das vezes, não é tão utilizada quanto a segunda, que é um keyboard que lê as teclas inseridas e escreve numa posição específica de memória.

Aqui, a mudança principal foi deixar o menu da primeira ferramenta oculto, e inserido um botão de seleção para que ele possa acessado quando necessário.

Segue o modelo de como ficou a nova versão da ferramenta, na qual o tamanho da janela também foi diminuído para ocupar menos espaço na tela:



Note que agora existe do lado superior direito a opção “Show Display MMIO”, em que ao selecioná-la, a ferramenta oculta retorna a tela, como podemos ver abaixo:



Outro detalhe, é que assim como o bitmap display, agora não é mais necessário fazer a conexão manualmente com o MIPS, ela é feita de maneira automática ao abrir a ferramenta.

## 3 Correção de Bugs

### 3.1 Deadlock

Um problema conhecido do MARS na versão 4.5 é referente ao uso simultâneo de duas ferramentas, sendo estas as duas já tratadas neste documento, o bitmap display e o keyboard, da ferramenta “Keyboard and Display MMIO Simulator”.

O problema segue quando estamos escrevendo no keyboard e observando as mudanças no bitmap display, existe um bug que pode ser melhor explicado se for feita uma análise um pouco mais minuciosa de como o MARS trabalha com essa concorrência. O MARS é implementado em Java, e existem várias threads rodando quando utilizamos a IDE. Ao abrir as duas ferramentas e utilizarmos elas de maneira concorrente, ocorre um erro chamado de “deadlock”, na qual uma thread para continuar seu processo, espera a finalização de uma outra, mas por sua vez, esta outra thread também espera a finalização da primeira, ou seja, uma thread depende da outra, e ambas nunca continuarão seu fluxo por essa dependência.

Este fato é meramente uma inconsistência do MARS, desta forma, a nova versão do MARS contempla uma correção, para evitar este caso excepcional de quando ocorre uma dependência mútua de threads.

Esta mudança foi feita na ferramenta do “Keyboard and Display MMIO Simulator”, na qual podemos ver abaixo como era na versão 4.5 do MARS:

```
if (notice.getAddress() == RECEIVER_DATA && notice.getAccessType() == AccessNotice.READ)
{
    updateMMIOControl(RECEIVER_CONTROL, readyBitCleared(RECEIVER_CONTROL));
}
```

Na nova versão, foi implementada uma solução com base num artigo publicado referente a este problema no site “Confect’s Codex”, artigo “MARS MIPS Simulator Lockup Hackfix”:

```
if (notice.getAddress() == RECEIVER_DATA && notice.getAccessType() == AccessNotice.READ)
{
    //This call caused a deadlock sometimes when using the keyboard and display simulator.
    //The synchronized block below simply plucks out the necessary call from
    //updateMMIOControlAndData, which is where this call would take us in the end.
    //-----
    //updateMMIOControl(RECEIVER_CONTROL, readyBitCleared(RECEIVER_CONTROL));
    //-----
    synchronized (Globals.memoryAndRegistersLock)
    {
        try
        {
            Globals.memory.setRawWord(RECEIVER_CONTROL, 0);
        }
        catch (AddressErrorException e)
        {
            e.printStackTrace();
        }
    }
    //-----
}
```