****

**UNIVERSIDADE FEDERAL DE SÃO PAULO**

**INSTITUTO DE CIÊNCIA E TECNOLOGIA**

Felipe Fernandes Mendonça

Guilherme Boccomino Marselha

**TUTORIAL PARA ROBÔ SEGUIDOR DE LINHA**

**SÃO JOSÉ DOS CAMPOS**

**2019**

Felipe Fernandes Mendonça

Guilherme Marselha

**TUTORIAL PARA ROBÔ SEGUIDOR DE LINHA**

Trabalho de graduação apresentado ao Bacharelado Interdisciplinar em Ciência e Tecnologia, como parte dos requisitos necessários à obtenção da avaliação do orientador.

**Professor(a):** Prof.ª Dr.ª Denise Stringhini

**Disciplina:** Arquitetura e Organização de Computadores

**SÃO JOSÉ DOS CAMPOS**

**2019**

1. **Introdução**

Os robôs seguidores de linha são robôs pré-programados que tem como função detectar onde existe um caminho em uma superfície, ou seja, um caminho em uma linha. Ele consegue fazer isso através da leitura de sensores.

Robôs seguidores de linha têm como objetivo percorrer, de maneira autônoma, uma trajetória definida. Esses robôs geralmente utilizam sensores infravermelhos com a função de detectar a posição do robô sobre a linha. Um controlador decide com base nessa informação os comandos apropriados para que o robô continue a seguir a linha, o que significa que ele trabalha com um mecanismo de realimentação, formando um efetivo e simples sistema fechado.

1. **Objetivo**

Este trabalho tem como objetivo trazer a simulação da atividade do robô seguidor de linha, utilizando-se do software MARS (MIPS *Assembler and Runtime Simulator*) e a arquitetura MIPS32.

O mesmo descreve um simulador para uma plataforma de seguidor de linha, com o intuito de facilitar os testes neste tipo de sistema. Simuladores, de maneira geral, possuem o benefício de facilitar a prática de uma determinada circunstância, o que favorece o desenvolvimento e progresso daquela área.

1. **Metodologia**

A ferramenta MARS é um programa desenvolvido em Java e a partir do release 4.0, contém 155 instruções básicas da arquitetura MIPS, aproximadamente 370 pseudo-instruções, 17 funções *syscalls* para o console e entrada e saída de dados, outras 22 funções *syscalls* para outros usos como o *MIDI output*, para as saídas de mídia, geração de números aleatórios e afins. Graças a essas e outras funcionalidades, tornou-se possível a execução dos diversos passos evidenciados abaixo a fim de reproduzir a ação do robô.

1. **Inicializando o MARS**

O software está disponível de graça no site de sua desenvolvedora, a Universidade de Missouri, nos Estados Unidos. Após realizar o download, é necessário executá-lo com o Java, já que o mesmo foi desenvolvido nessa linguagem, conforme citado acima.

Após abri-lo, uma tela semelhante a abaixo irá aparecer, trata-se da interface do programa.

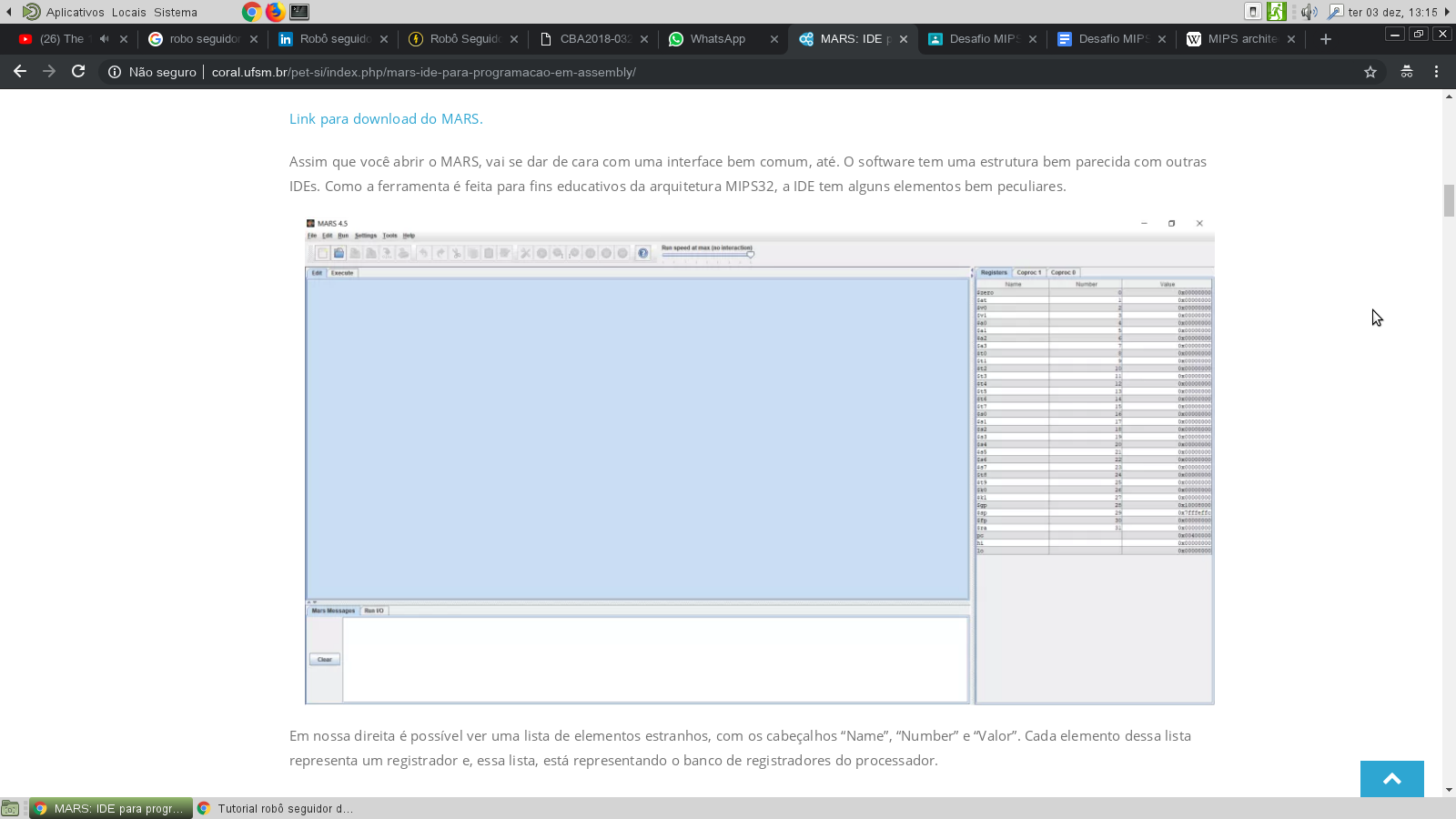


Figura 1 - Interface do software MARS

É necessário criar um arquivo e salvá-lo para iniciar as atividades no programa.

1. **Grid da Tela**

Dando início a programação em si da atividade, iniciou-se programando os loops das linhas que o robô tem que encontrar o grid da tela.

Contextualizando, grid é qualquer conjunto de linhas que sejam traçadas com o único objetivo de guiar ou dar forma a um design, independentemente do processo que foi usado. É um elemento técnico-formal formado por um conjunto de linhas auxiliares na vertical e horizontal, ou de retângulos, que proporcionam uma estrutura para construção das nossas peças de comunicação visual, gráfico ou web, auxiliando na ordenação, distribuição, alinhamento e dimensão dos elementos gráficos (imagens, textos, formas). A função principal do Grid é organizar as informações dentro de uma estrutura.

Baseando-se nessas informações, foi delimitado a área por onde o robô faria seu percurso.

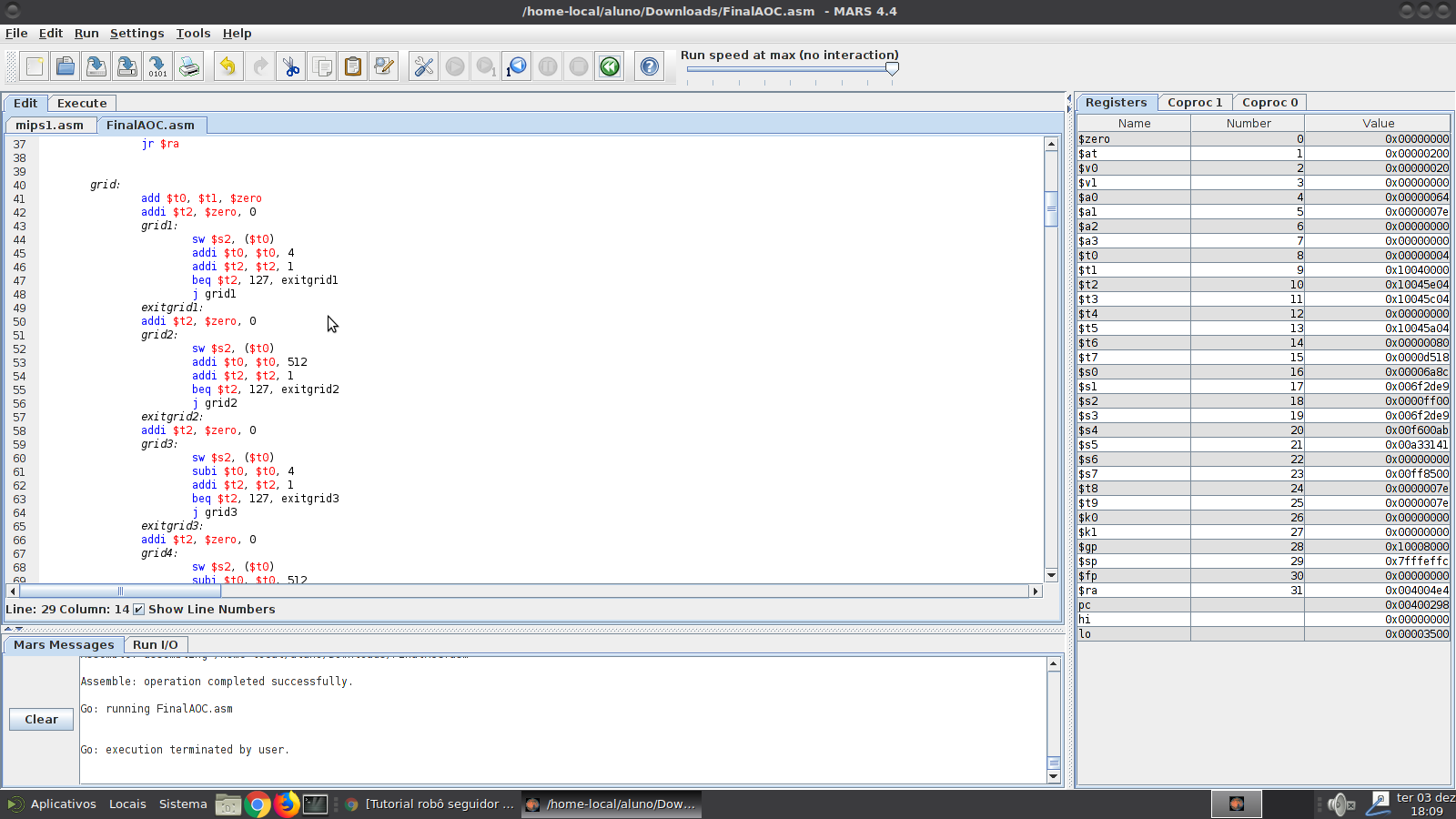


Figura 2 - Bloco que representa o grid

Nesta função, desenhou-se o grid utilizando 4 loops (estão mostrados apenas dois pois os restantes possuem a mesma lógica, apenas com parâmetros diferentes). Utilizamos a variável t0 para pintar o bitmap e a variável t2 como contador. O primeiro loop repete-se 127 vezes e move o bit t0 para a direita pintando assim a barra horizontal superior. O segundo loop move o bit para baixo, pintando a barra vertical direita. Os loops restantes movem o bit para esquerda e depois para cima e assim, o grid é finalizado.

1. **Inicialização e Movimentação do robô**

A movimentação do robô serviu de base para todo o resto das ações, sendo este o principal bloco do programa.

Utilizando-se do método de apagar o último pixel feito pelo movimento do robô ao mesmo tempo em que pinta o próximo com o intuito de dar impressão de movimento, realizou-se o deslocamento do mesmo.

Executou-se isso horizontalmente e, ao encontrar uma parede, o robô descia e mudava sua posição. Ao chegar no final, o mesmo subia e recomeçava a procura até encontrar uma linha ao seu redor, para ver o próximo ponto comandado a fim de continuar a pintura das linhas.

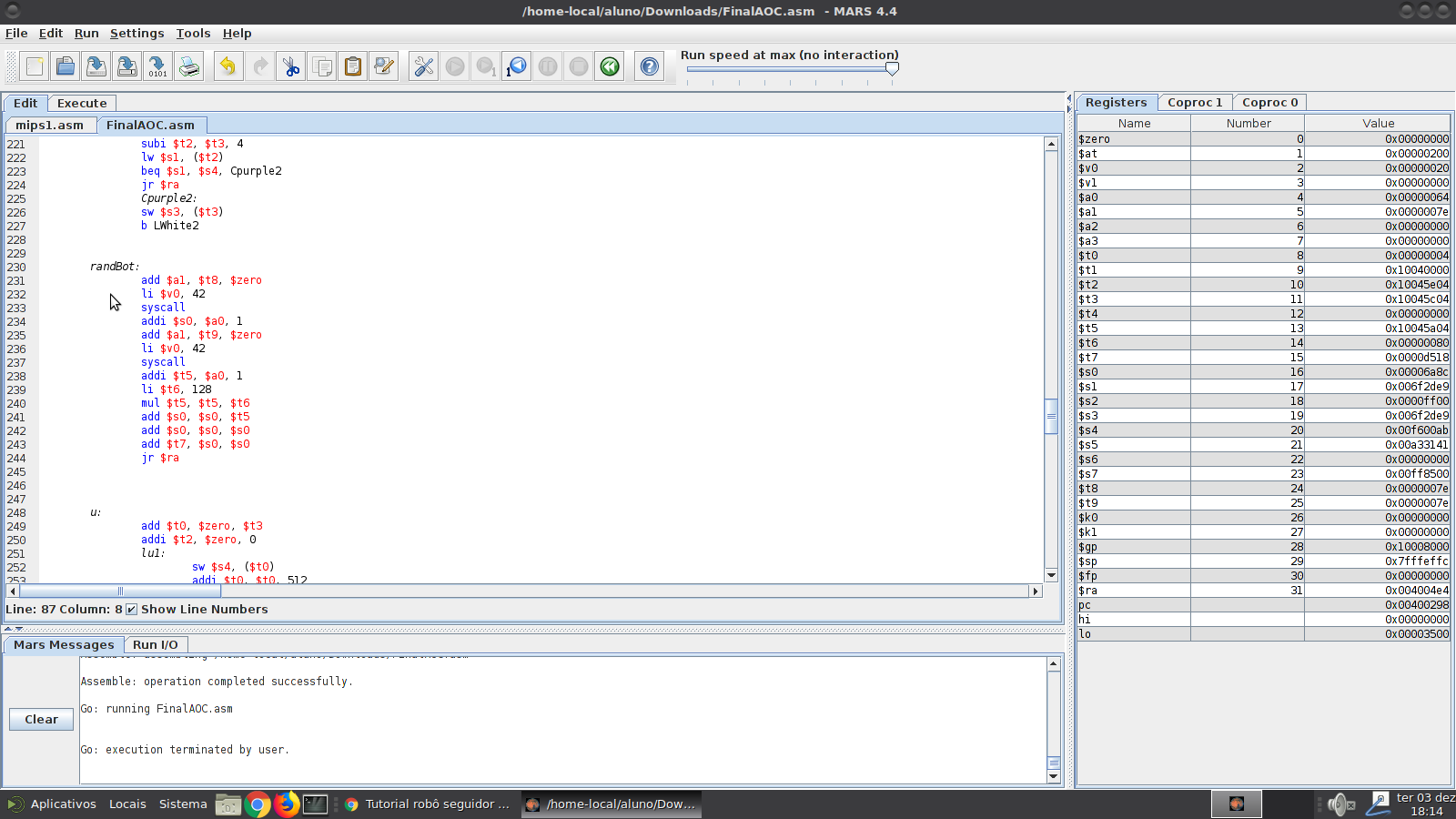


Figura 3 - A randomização do movimento do robô

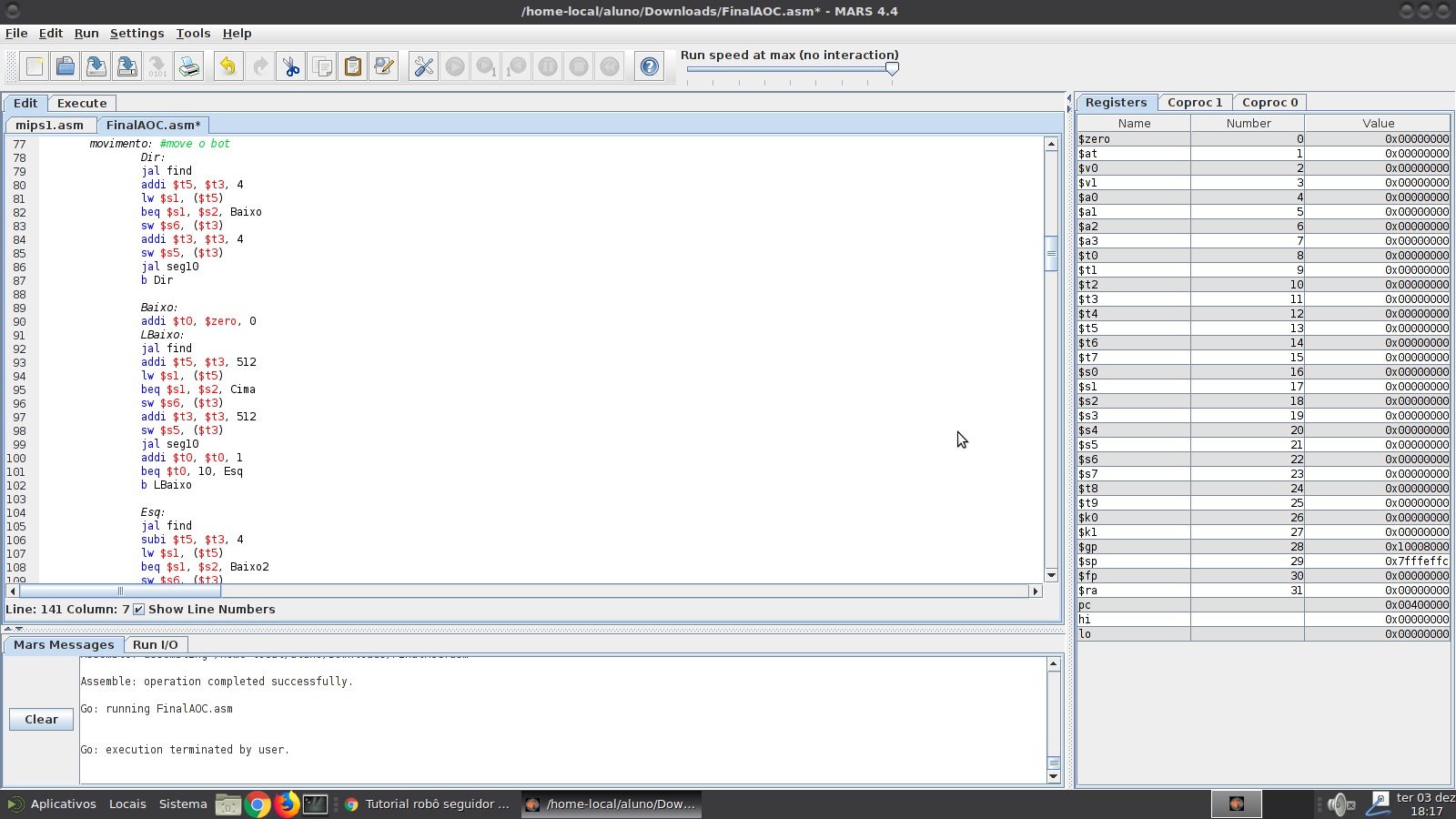


Figura 4 - Comandos de movimentação do robô (direita e baixo)

Na função Dir, tem-se um loop para movimentar o bit para a direita bit a bit com um intervalo de 10 milissegundos para melhorar a visualização. Além disso, utilizamos a função find a cada movimento para procurar ao redor do bit se a linha está presente. A função Baixo é semelhante, porém movimenta o bit 10 posições para baixo ao encontrar uma parede vertical do grid.

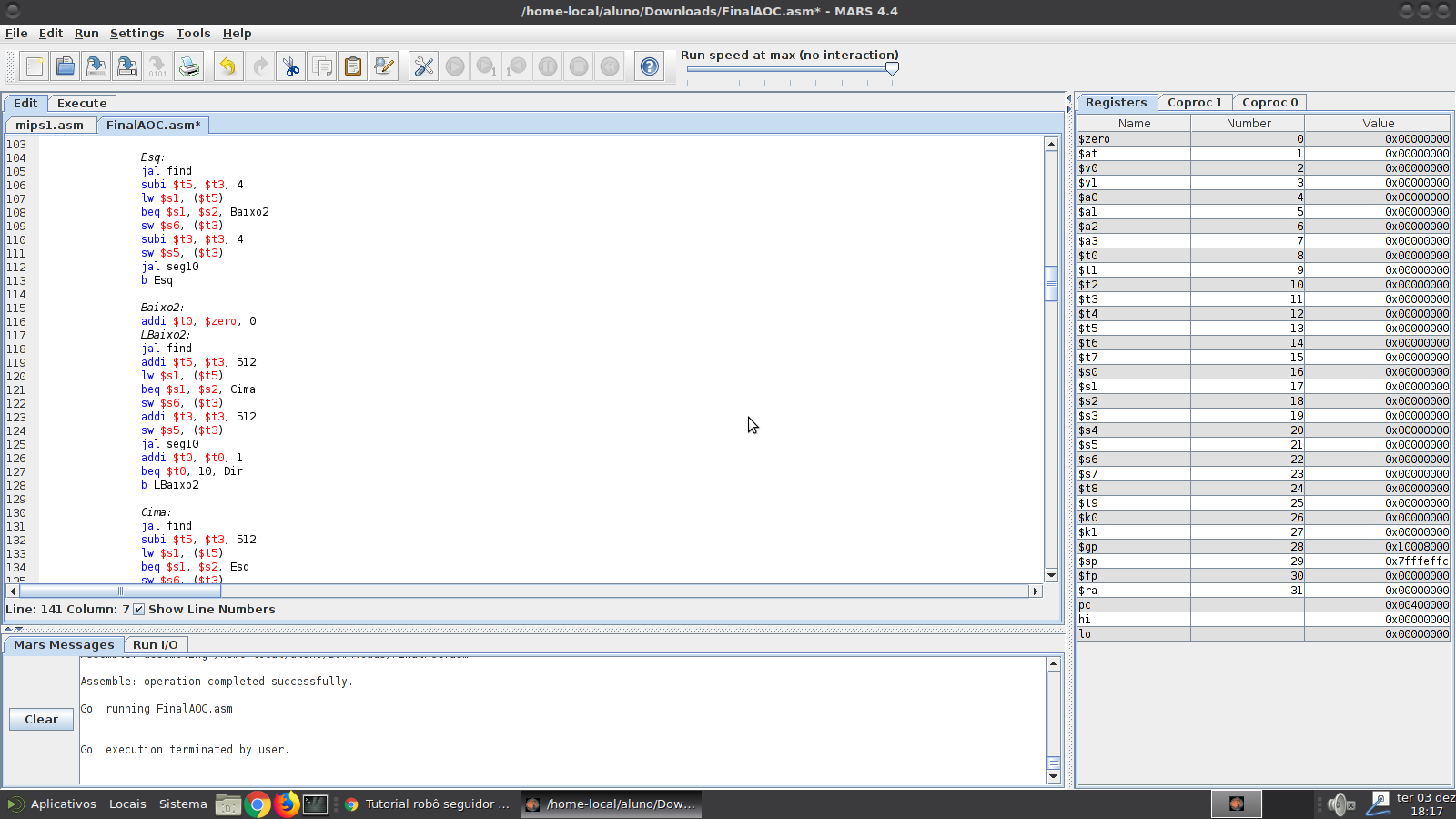


Figura 5 - Movimentação para a esquerda e baixo

Na função Esq, tem-se exatamente o mesmo funcionamento da função Dir, porém voltado para a esquerda. Já a função LBaixo2 é semelhante a função Baixo e trata da colisão do robô na horizontal.

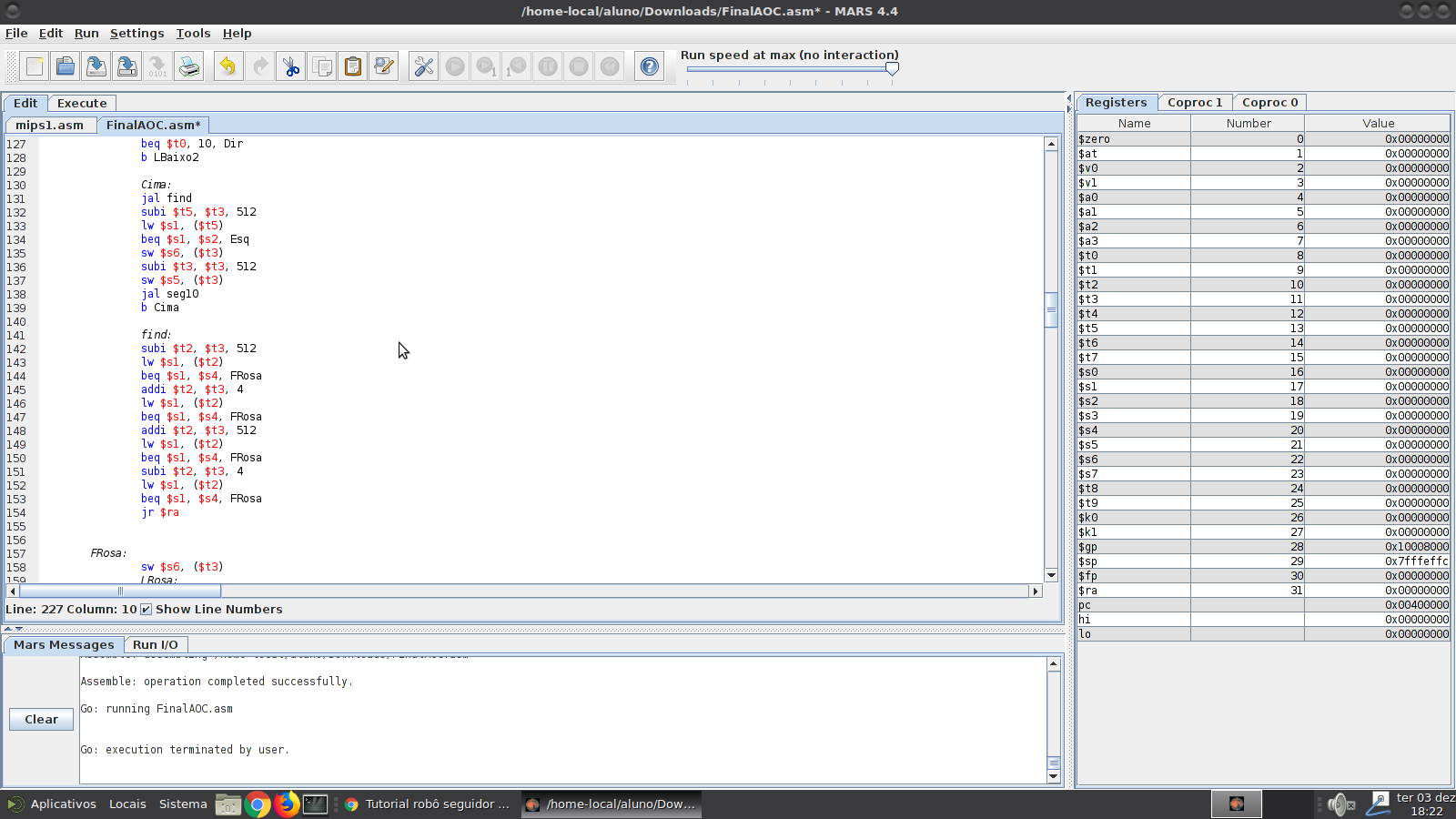


Figura 6 - Movimentação para cima e tentativa de achar a linha

A função Cima inicia ao encontrar a parede horizontal inferior. Nesta função, movimenta-se o bit até encontrar o grid horizontal superior e inicia-se o movimento horizontal novamente. A função find procura ao redor do bit se a cor rosa está presente para então começar a segui-lá.

1. **Colisões com o grid**

Baseando-se agora nas partes já feitas acima, as colisões advindas do movimento foram tratadas de forma que ao encontrar a margem, o robô mudasse sua direção, isto é, quando atingisse a linha superior, desceria e, analogamente, ao alcançar a linha inferior, subiria, recomeçando o movimento nas duas situações.

1. **Programando o comportamento quanto as colisões**

O programa foi estabelecido de forma que a procura pela variável pintada seria sempre a base do movimento, a fim de manter sempre o robô na linha. Sendo assim, ao identificar uma variável colorida, o mesmo direcionava-se a ela.

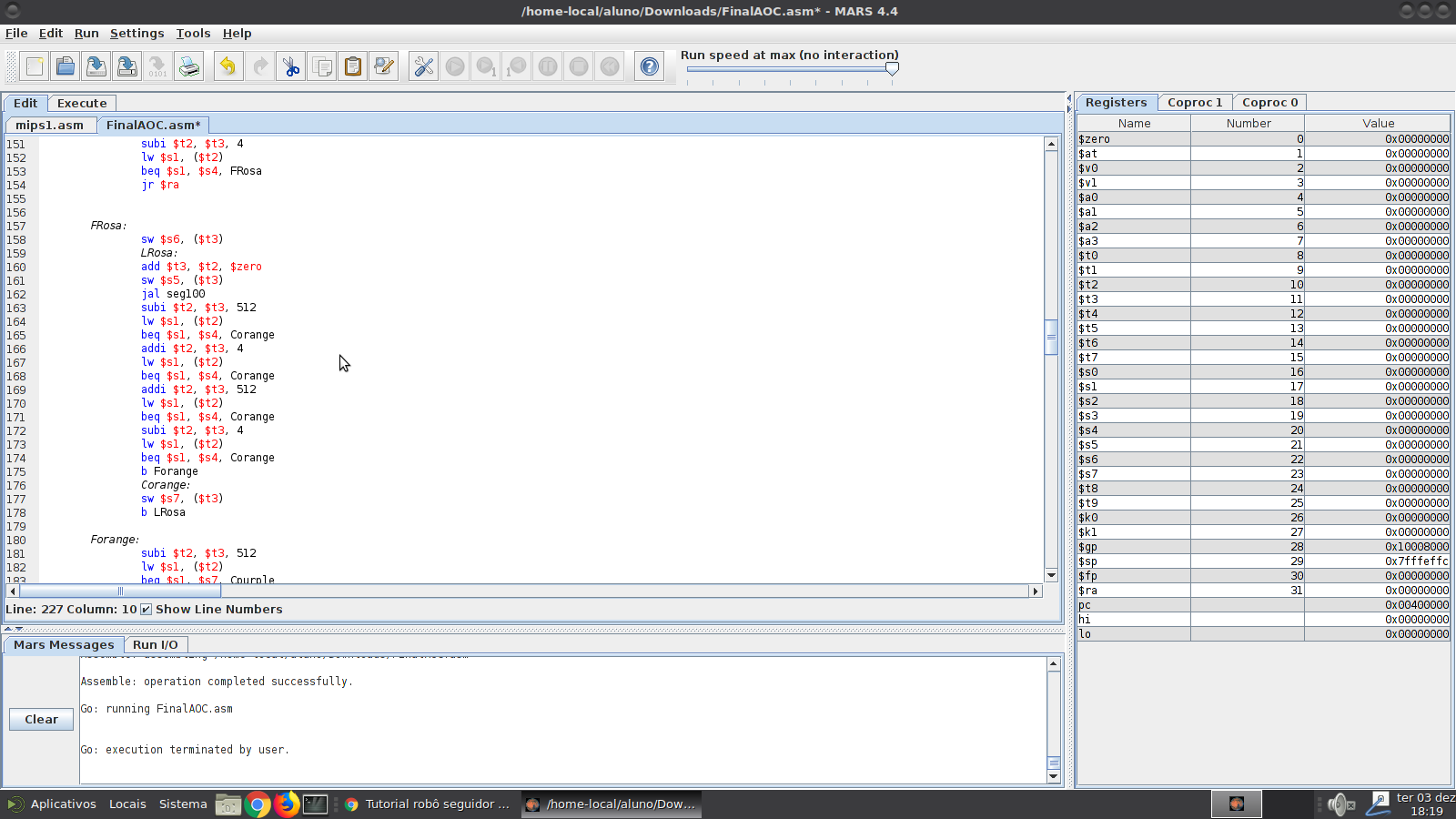


Figura 7 - Comandos para encontrar o primeiro bit de cor rosa

A função FRosa é utilizada para manter o bit seguindo a linha rosa e marcar a posição anterior de laranja.

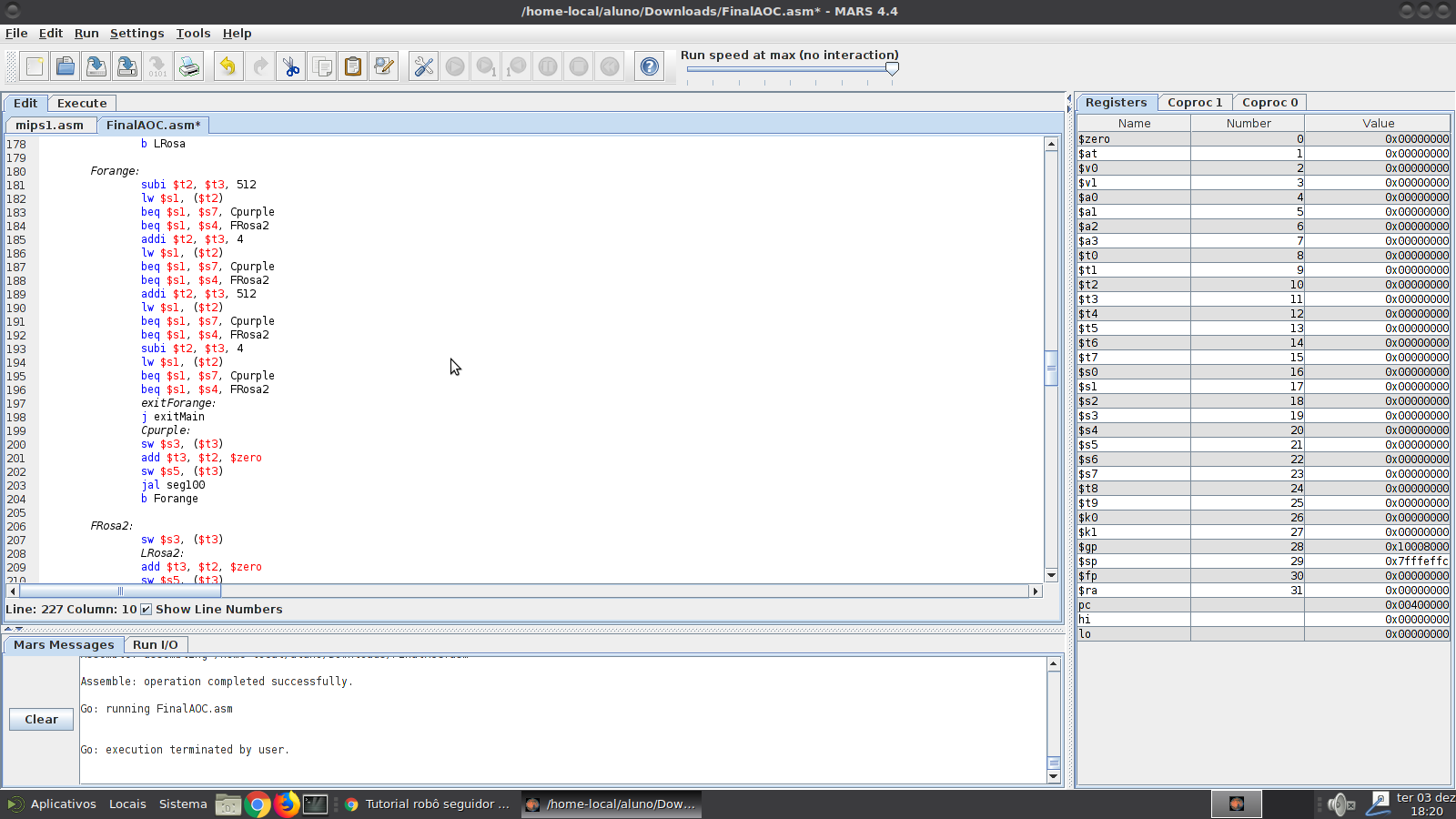


Figura 8 - Comandos para realizar o 1º ciclo de marcação na cor laranja

A função Forange é utilizada para achar a cor laranja após a primeira passagem e, na sequência, marcar a posição anterior de roxo.

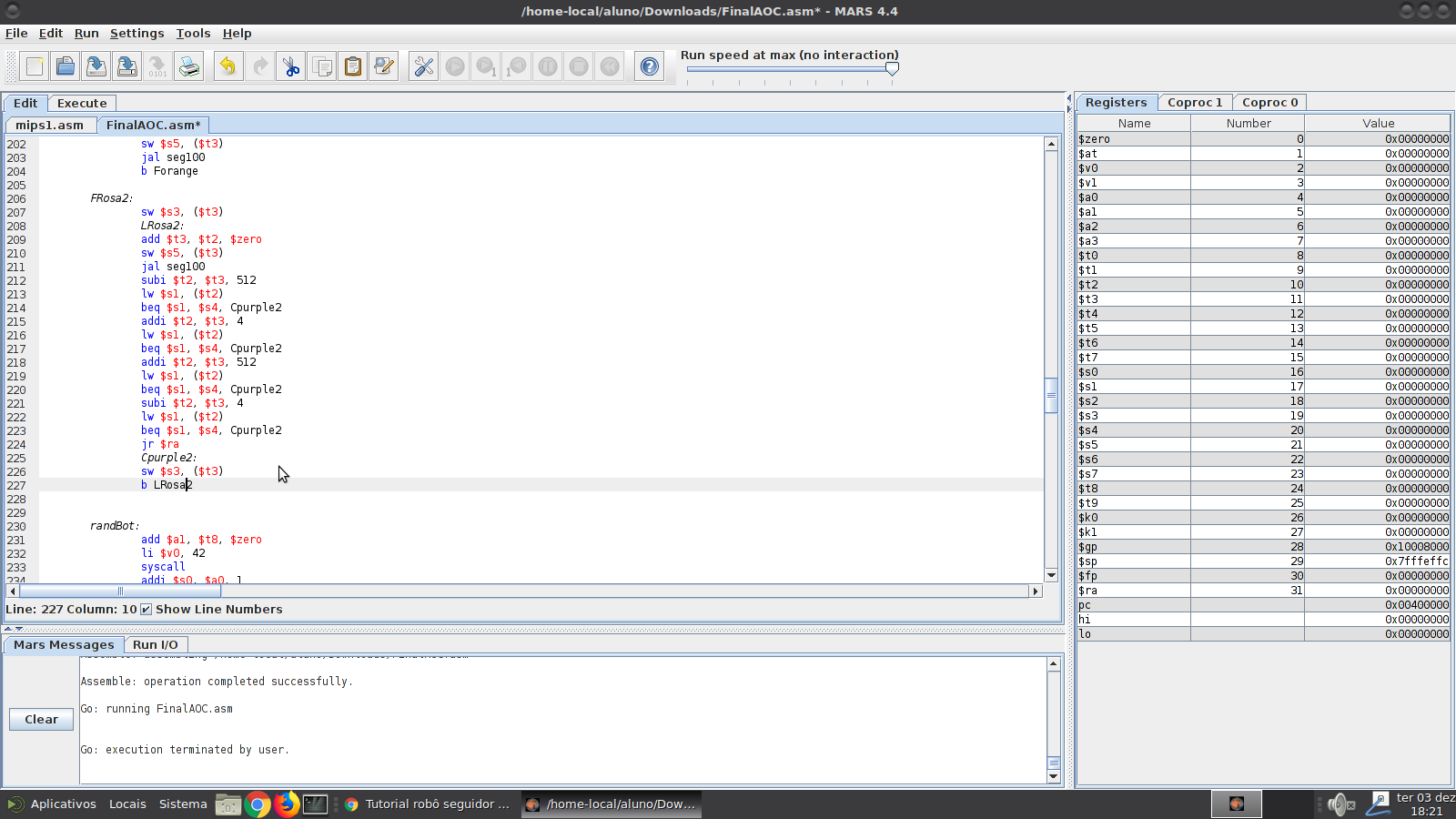


Figura 8 - Comandos para realizar o 2º ciclo de marcação na cor roxa

A função FRosa2 é utilizada para, ao chegar em um bit rosa ainda não marcado (caso o bit encontre a linha em uma posição diferente do início), pintar o bit anterior de roxo, assim, marcando a linha em sua totalidade.

1. **Configurando o Bitmap Display**

O Bitmap Display é uma janela para exibição gráfica baseada em pontos. Suas principais características são as seguintes:

* Um ponto é conjunto de 1 ou mais *pixels*. O tamanho de um ponto é definido pelos parâmetros de *Unit Width in Pixels* e *Unit Heigth in Pixels* que definem, respectivamente, a largura do ponto em *pixels* e a altura do ponto em *pixels*.
* Resolução configurável (*default* 512 x 256*pixels*).
* Cada ponto é representado por uma palavra de 32 bits (4 bytes), no formato RGB, um *byte* para cada cor. *Red*(vermelho) = 0x00FF0000, *Green*(verde) = 0x0000FF00, *Blue*(azul) = 0x000000FF. O *byte* mais significativo é ignorado.
* Essa ferramenta utiliza um *display* gráfico mapeado em memória, isto é, cada endereço de memória corresponde a um ponto.
* O endereço base de memória do Bitmap Display é configurável. Por exemplo, pode ser utilizado como endereço base de memória: *global data, global pointer, static data, heap, memory map.*
* Por *default*, o endereço base de memória aponta para o ponto superior esquerdo da tela.
* O desenho de um pixel na tela é realizado pela escrita de uma palavra contendo a descrição de sua cor RGB na posição de memória correspondente.

Seguindo a orientação da professora, as configurações do Bitmap Display foram as seguintes:

**Configuração de memória (aba “Tools → Memory Configuration”):** padrão

**Configuração do Bitmap Display:**

* Unidade de largura em pixels (*Unit Width in Pixels*): 4
* Unidade de altura em pixels (*Unit Heigth in Pixels*): 4
* Largura do display (*Display Width in Pixels*): 512
* Altura do display (*Display Heigth in Pixels*): 512
* Endereço base do display (*Base address for display*): 0x10040000 (heap)

Com isso, o Bitmap Display foi exibido em uma matriz de 512x512 pixels, com cada ponto com tamanho de 4 pixels de altura por 4 pixels de largura.

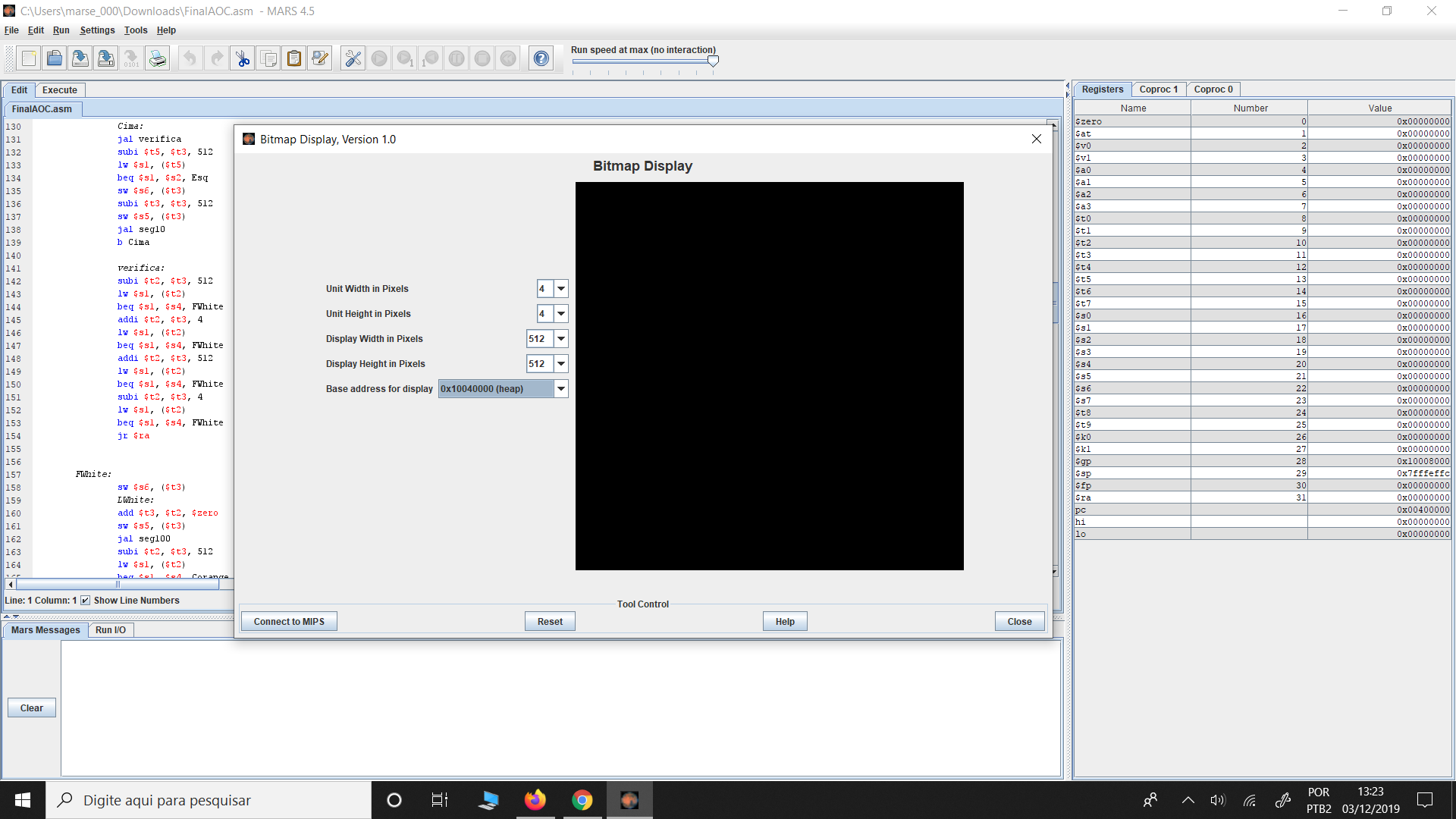


Figura 2 - Bitmap Display no MARS

1. **Conectando-se ao MIPS com Bitmap Display**

Após a configuração do Bitmap Display, conectou-se ao MIPS, deu-se “assemble” com o Bitmap aberto e executou-se o programa.

Para interromper o programa, clicou-se em “stop”, seguido de “reset” no Bitmap e executou-se novamente clicando em “run”.

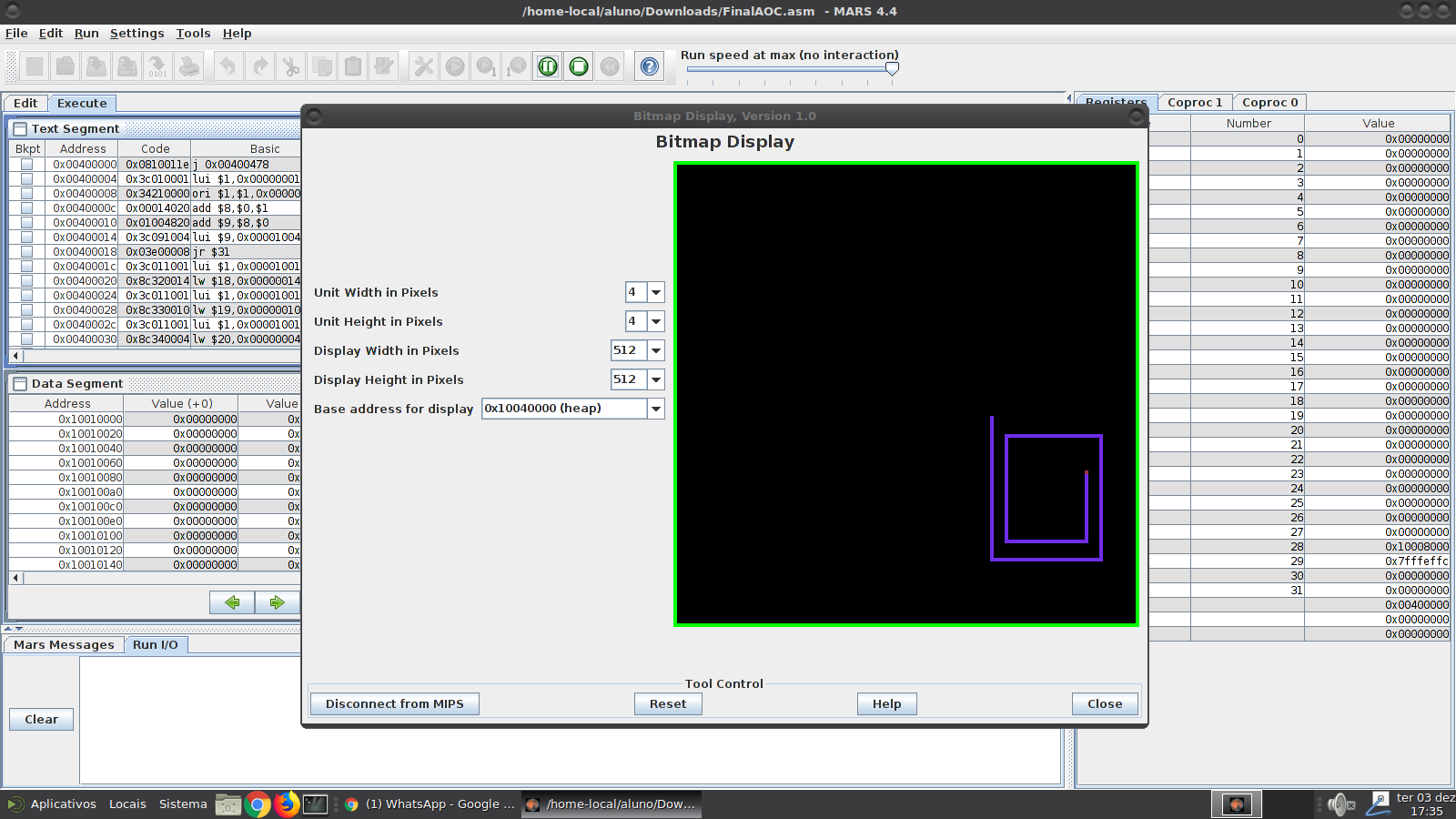


Figura 3 - O robô percorrendo o caminho selecionado

**4. Conclusão**

A simulação, após executada, mostrou-se muito satisfatória, visto que, se comparada com a realidade, reproduz fielmente - com suas limitações - o que observamos pessoalmente, concluindo que, as representações em softwares podem facilitar, agilizar e representar diversas situações e atividades.

O sistema nessa versão já permite o teste de códigos do tipo seguidor de linha, porém, ainda é necessário fazer uma modelagem mais precisa do sistema físico para que se permita a otimização do controle. Fenômenos como inércia, atrito, etc. precisam ser modelados e implementados no simulador.

**5. Referências Bibliográficas**

SIMULADOR DE CÓDIGO ABERTO PARA A PLATAFORMA POLOLU 3PI. UFPE. Disponível em: <https://ssl4799.websiteseguro.com/swge5/PROCEEDINGS/PDF/CBA2018-0321.pdf>. Acesso em: 30 nov. 2019.

ROBÔ SEGUIDOR DE LINHA. Robocore Tecnologia. Disponível em:<https://www.robocore.net/tutoriais/robo-seguidor-de-linha>. Acesso em: 30 nov. 2019.

ROBÔ SEGUIDOR DE LINHA. Isvaldo Souza. Disponível em:<https://pt.slideshare.net/Miojex360/rob-seguidor-de-linha>. Acesso em: 30 nov. 2019.

MARS: IDE PARA PROGRAMAÇÃO EM ASSEMBLY. PET-SI, UFSM. Disponível em:<http://coral.ufsm.br/pet-si/index.php/mars-ide-para-programacao-em-assembly/>. Acesso em: 30 nov. 2019.