Davi Friggi - 176479

Gustavo Tapajós - 177063

Rafael Figueredo - 176609

https://github.com/dFriggi/Simulador-de-Semaforo-Inteligente

Relatório do Simulador de Dispositivo Embarcado/IoT

Resumo

O objetivo deste projeto é desenvolver um simulador de semáforo inteligente em linguagem Assembly MIPS para demonstrar conceitos fundamentais de programação dessa linguagem. A ideia central do simulador é ser semelhante a semáforos que ajustam seus tempos de duração baseado no fluxo de veículos, calculando a média móvel dos últimos cinco ciclos para determinar o tempo do semáforo aberto, com transições para amarelo e vermelho, com visualização em tempo real no Bitmap display. As principais funcionalidades são: leitura de input do usuário para simular o número de carros, um vetor histórico com índice circular, média móvel para o cálculo do tempo, funções dedicadas para o bitmap display e um contador regressivo para simular segundos reais. O que foi alcançado é uma implementação funcional que simula corretamente o funcionamento de semáforos inteligentes e que exibe graficamente as alterações de estado do programa.

Introdução

O contexto do tema escolhido, um simulador de semáforo inteligente, é inspirado em desafios do dia a dia da vida urbana relacionados ao tráfego e ao seu gerenciamento, no qual sistemas de controle que se adaptem à situação são essenciais para otimizar o fluxo de veículos, reduzindo congestionamentos e melhorando a segurança nas diversas vias, especialmente em cidades com alta densidade populacional. O que motivou a escolha dessa simulação foi a dificuldade encontrada no caminho para o Campus UNIFESP de São José dos Campos, pois muitos alunos levam muito tempo para chegar até a faculdade, algo que poderia ser resolvido com a implementação de semáforos inteligentes pela cidade.

A relação do projeto com a disciplina de Organização de Computadores é direta: a programação em Assembly MIPS é um exemplo de desenvolvimento de baixo nível, onde o programador gerencia diretamente a alocação de memória e instruções otimizadas para o processador RISC. Foram utilizados conceitos de arquitetura, como o uso de registradores dedicados como $t (temporários), $s (preservados para variáveis persistentes), $a (parâmetros de entrada) e $v (retornos), que ilustram o modelo de registradores de uso geral. Além disso, há a manipulação de registradores por meio de operações como shifts lógicos (sll para multiplicação por 4 em índices de array), divisões (div e mflo para média móvel) e manipulação da pilha ($sp para prólogos/epílogos). Por fim, o conceito de máquina de estados finitos na implementação do

semáforo, com três estados distintos (verde, amarelo, vermelho) transitando de forma baseada em condições do sistema (tempo decorrido e fluxo de carros).

Descrição Geral do Sistema

Objetivos do Simulador

O programa deve simular corretamente o funcionamento de um semáforo inteligente. Nesse sentido, o semáforo inicia no estado pare (vermelho). Após a leitura do sensor, há a alocação do valor no vetor histórico implementado, seguido do cálculo da média móvel, o qual vai indicar qual o tempo que o semáforo deve ficar no estado aberto (verde). Após isso, o envio do valor para o contador, que vai ser decrementado, e a alteração do semáforo para o estado aberto (verde). Quando o contador zerar, o estado do semáforo muda para atenção (amarelo) e o contador simula 5 segundos, que ao terminarem, o semáforo muda para o estado pare novamente, e aguarda uma próxima leitura do sensor.

Funcionalidades implementadas

Foi implementado a leitura simulada do sensor de fluxo de carros, atualização automática de estados do programa, gerenciamento de vetor histórico com índice circular, cálculo da média móvel, contador decrescente e configuração do Bitmap Display, que mostra os valores do contador e as cores atuais do semáforo.

Arquitetura geral

As principais funções e módulos são a função loop (que gerencia o menu), função de alocar o valor no vetor histórico dos carros, função de cálculo da média móvel, função do contador e as funções do Bitmap Display que exibem o semáforo e a visualização do contador.

Primeiramente, há o input do usuário para simular o sensor de fluxo de carros, o qual é alocado no vetor de histórico de fluxo de carros. Nesse sentido, utilizamos um vetor com 5 posições de memória para armazenar os valores lidos pelo sensor, juntamente com um índice circular, o qual, após a utilização das 5 posições do vetor, começa a preenchê-lo novamente pela primeira posição. Em seguida, há o cálculo do tempo por meio da média móvel , o qual, em decorrência de um vetor com índice circular, foi feito de maneira simples, somando os 5 valores do sensor e dividindo pelo tamanho (5).

Nessa perspectiva, após o cálculo da média móvel, o valor calculado (tempo) é enviado para o contador decrescente e para o alterador do semáforo, que muda para “aberto” (verde) e simula o decremento do tempo recebendo um input de caracter Enter do teclado, para tornar o modelo semelhante à passagem dos segundos. Ao término do contador, o semáforo muda para “atenção” (amarelo) e utiliza um contador decrescente com o número 5 para simular 5 segundos com o sinal amarelo. Por fim, o semáforo volta para “pare” (vermelho) e aguarda pela próxima leitura do sensor, para realizar novamente todo o processo. Todos os processos que necessitam de visualização foram feitos utilizando o Bitmap Display, ou seja, a mudança de cores do semáforo e os valores do contador.

Máquina de Estados

Estado: aberto

No estado aberto, o sistema já leu o input do sensor de fluxo, já alocou o espaço no vetor corretamente e já fez o cálculo do tempo por meio da média móvel. Nesse estado, o semáforo está com a cor verde e o valor de tempo calculado é o que determina a sua duração. Nele, há o funcionamento de apenas funções de contador, as quais fazem o decremento do valor e alteram o Bitmap Display conforme o decremento do próprio contador.

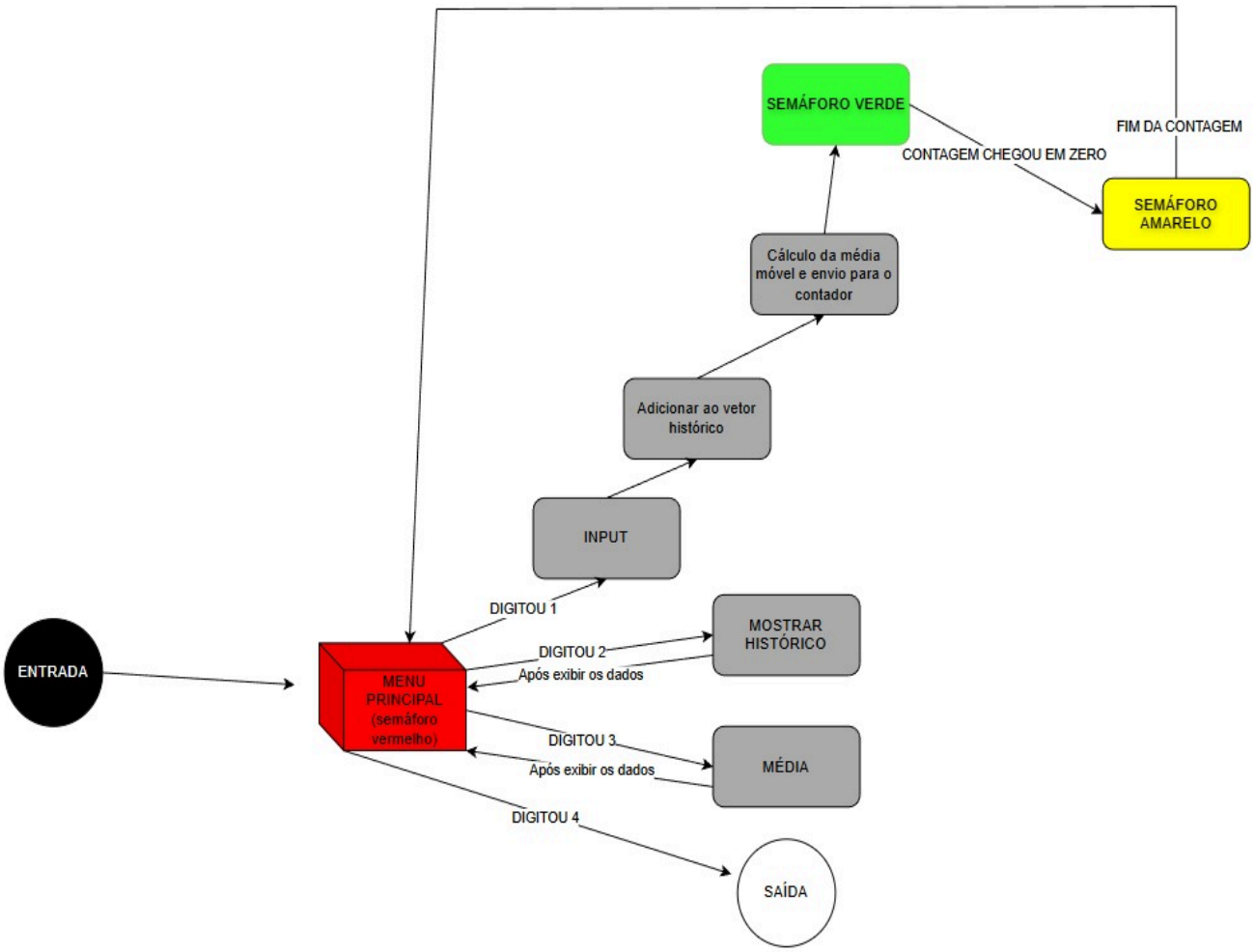
Estado: atenção

No estado de atenção, que vem logo após o estado aberto, o semáforo está com a cor amarela e em uma “preparação” para o próximo ciclo do programa. Além disso, o contador muda para o valor 5, simulando 5 segundos de sinal amarelo, valor esse que é mostrado também pelo Bitmap Display,

Estado: pare

No estado pare, o programa ou acabou de ser iniciado, ou já encerrou um ciclo completo. Aqui, o semáforo muda para a cor vermelha, e o programa aguarda novamente o input do valor para o sensor de fluxo. Durante esse estado, e após o input, são realizados a alocação da leitura do sensor no vetor histórico de carros e o cálculo do tempo por meio da média móvel.

Diagrama de Estados



Estrutura de dados

O principal vetor é o VETOR_HISTORICO, declarado como .word seguido de cinco valores inteiros (ex.: .word 5, 12, 8, 15, 10), que representam o histórico de leituras de fluxo de veículos dos últimos 5 ciclos do programa, com tamanho fixo de 5 elementos para o cálculo da média

móvel. Esse vetor utiliza um índice circular que é gerenciado pela variável INDICE_VETOR (.word), que avança modularmente conforme o cálculo: índice = (índice + 1) % 5 para sobrescrever o valor mais antigo, com o principal objetivo de evitar overflow e manter apenas os últimos cinco dados relevantes. O acesso é feito conforme convenção de endereço base + deslocamento, onde o endereço de um elemento é calculado como base + (índice × 4), que foi implementado utilizando shift lógico (sll $t1, $t0, 2) para multiplicação por 4 (tamanho de cada word).

O Bitmap Display opera implicitamente como uma matriz 256×64 de pixels, mapeada na memória a partir do endereço base 0x10008000 (DISPLAY_ADRESS: .word 0x10008000), onde cada pixel é uma word de 4 bytes. Dessa forma o display é “pintado” conforme o andamento do programa.

A representação do sensor de tráfego é simulada pela entrada do usuário via syscall 5 (li $v0, 5; syscall), que captura o número de carros como um valor em $v0, armazenado no vetor histórico.

A organização geral dos dados na memória segue a convenção MIPS de seção .data para variáveis globais, com constantes como cores (.word 0x00FF0000) e strings (.asciiz "\n" para QUEBRA), agrupadas logicamente para facilitar acesso via la (load address) e lw (load word). Por exemplo, variáveis de controle como ESTADO_ATUAL_SEMAFORO (.word 0) são escalares simples para estados finitos e contadores, enquanto MEDIA_MOVEL (.word 5) armazena o resultado da divisão (mflo $v0).

Funções principais

loop:

Entrada: não ocorre

Registradores: são utilizados $t2, $t3 e $t4 para o funcionamento do menu, os quais armazenam os valores 2, 3 e 4 e os comandos de branch. O $a1 é utilizado para passar o valor 0 para a função seleciona_cor, a qual muda o semáforo para vermelho. O $v0 é utilizado para a leitura do input, o qual é passado para o $a0.

Pilha: não ocorre

Retorno: não ocorre

Lógica: primeiramente torna o semáforo vermelho, após isso armazena os valores nos $t, abre o menu para o usuário escolher: seguir com mais um ciclo, ver o vetor histórico atual, verificar a conta da média móvel ou encerrar o programa. Se escolher seguir com o ciclo, há a leitura do input.

adiciona_carro_historico:

Entrada: input do sensor

Registradores: $t0 para o endereço do vetor , $t1 para o índice do vetor, $t2 para o sll ($t2, $t1, 2), $t3 para o cálculo do endereço correto dentro do vetor ($t2 + $t0).

Pilha: não ocorre

Retorno: não ocorre, porém o VETOR_HISTORICO está com seus valores atualizados

Lógica: primeiramente, são passados os endereços do VETOR_HISTORICO e INDICE_VETOR, com isso é realizado o cálculo correto do endereço dentro do vetor e o valor de $a0 (input) é inserido na posição. Após isso o indice é incrementado em 1, e armazena-se o resto da divisão do indice por 5 de volta no INDICE_VETOR

media_movel:

Entrada: não ocorre

Registradores: $s1 para o endereço do vetor, $s0 para a soma dos valores, $t0 para a quantidade de iterações. $t1, $t2 e $t3 para o pegar o valor dentro do vetor no índice. $t5 e $t6 para o cálculo da divisão final por 5

Pilha: $sp com 3 posições para salvar o $s0, $s1 e o $ra no início da função

Retorno: valor da soma dos valores do vetor dividido por 5

Lógica: primeiramente, são salvos na pilha os valores de $s0, $s1 e $ra. Após isso, é passado o endereço do vetor para $s1. Em seguida é chamado o loop da média, que soma todos os valores do vetor, que são enviados para o fim da média que calcula a divisão da soma por 5, restaura $s0, $s1 e $ra e retorna o valor resultante da divisão.

contador:

Entrada: retorno da média móvel

Registradores: $s0 para o retorno da média, $s3 e $s4 para as funções do Bitmap, todos os $t para comparação para o uso de branchs que desenham números específicos. $s2 e $a3 para a divisão, para obter o valor das unidades e das dezenas.

Pilha: $sp com 4 posições, salvando o $ra, $s0, o $ra ad função que chama o Bitmap das unidades e das dezenas, separadamente.

Retorno: não ocorre

Lógica: primeiramente, é passado o valor calculado da média. É separado o dígito da unidade e da dezena e é enviado para as funções de Bitmap para visualização dos valores. Essa funções salvam

$ra para voltar para o contador principal. Após isso, o contador é decrementado, e ocorre mais um ciclo, até que se encerre.

Uso de Inteligência Artificial

Foi utilizado o Gemini 3.0 PRO para sugestões de melhorias e auxílio no cálculo. Porém, toda a implementação foi feita pelo grupo.

Local	Geração da IA	Explicação
Função de adicionar_vetor_histórico e funcionamento do vetor	Sugestão de Indíce Circular para o vetor de histórico de carros	Método utilizado para considerarmos somente as últimas 5 leituras do vetor histórico de carros, de maneira que fosse substituida a leitura mais “antiga”
Função da média móvel e o cálculo envolvido	Sugestão do cálculo para o ajuste de tempo do semáforo	Cálculo sugerido foi o de somar todos os 5 valores do vetor histórico e dividir por 5, utilizando o algoritmo não trivial da média móvel.
Funções primárias como loop adicionar_vetor_historico, media_movel e contador.	Verificação de funcionamento dos primeiros passo com a IA	Essa verificação ocorria pelo fato de o display e a parte visual não estarem prontas. O foco era o funcionamento lógico, portanto, essas aferições eram feitas usando a IA.
Funções de média móvel e adicionar vetor histórico	Sugestão do Shift Left Logical para calcular a posição * 4 (sll $t2, $t1, 2)	Sugestão de trocar um mult com 4 por um sll de 2 bits, para fazer a multiplicação do índice posição * 4, garantindo o correto acesso ao vetor.

O que corrigimos

A inteligência artificial havia “corrigido” a função adiciona_carro_historico e adicionou um processo desnecessário de salvar o $ra em $sp. Esse processo é desnecessário por conta de não haver outra chamada de função para que o $ra fosse sobrescrito, portanto essa correção salvou espaço no stack pointer.

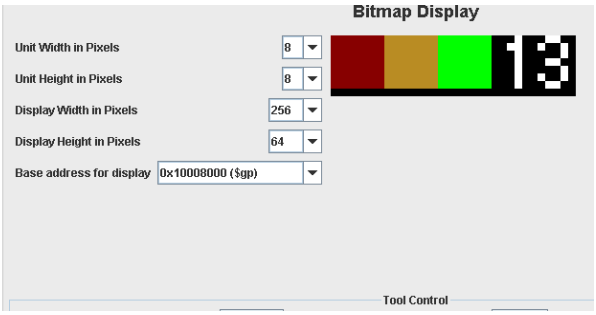
Teste e Resultados

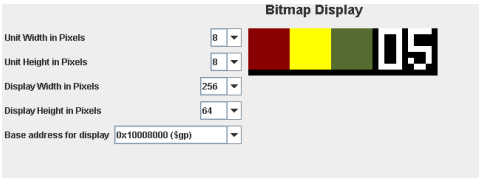
Caso típico

Simulação da inserção de um valor comum para o sensor (20)

```
Adicione a quantidade de carros lida pelo sensor (simulação): Selecione o processo desejado:
 1 - Seguir com o processo
 2 - Mostrar histórico últimos 5 semáforos
 3 - Mostrar média móvel atual
 4 - Encerrar

1
Adicione a quantidade de carros lida pelo sensor (simulação): 20
Quantidade de carros no vetor: 20
Soma atual: 20
Quantidade de carros no vetor: 12
Soma atual: 32
Quantidade de carros no vetor: 8
Soma atual: 40
Quantidade de carros no vetor: 15
Soma atual: 55
Quantidade de carros no vetor: 10
Soma atual: 65
65/5 = 13
```





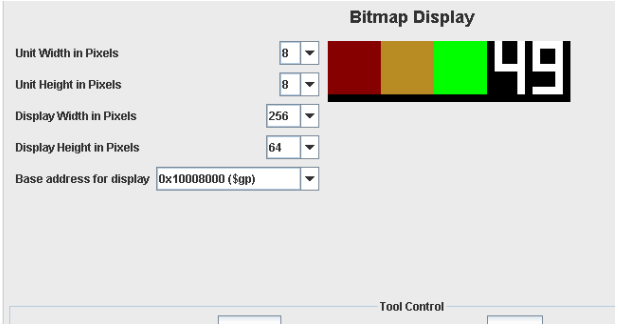
A partir desse ponto, o valor decrementa a cada Enter do teclado. Ao final, mostra 5 segundos e atenção e volta para o estado pare.

Caso adverso

Simulação da inserção de um valor comum para o sensor (200)

```
Selecione o processo desejado:
 1 - Seguir com o processo
 2 - Mostrar histórico últimos 5 semáforos
 3 - Mostrar média móvel atual
 4 - Encerrar

1
Adicione a quantidade de carros lida pelo sensor (simulação): 200
Quantidade de carros no vetor: 200
Soma atual: 200
Quantidade de carros no vetor: 12
Soma atual: 212
Quantidade de carros no vetor: 8
Soma atual: 220
Quantidade de carros no vetor: 15
Soma atual: 235
Quantidade de carros no vetor: 10
Soma atual: 245
245/5 = 49
```



Caso estrutural

Por conta da estrutura de índice circular utilizado, o vetor sempre estará cheio mas nunca terá overflow, pois um valor é sobrescrito em todos os ciclos.

```
Selecione o processo desejado:
 1 - Seguir com o processo
 2 - Mostrar histórico últimos 5 semáforos
 3 - Mostrar média móvel atual
 4 - Encerrar

2
Vetor com o histórico dos últimos 5 valores lidos pelo sensor: 5, 12, 8, 15, 10
```

Discussão de resultados

O resultado obtido foi bastante satisfatório. Por conta da dificuldade do Bitmap, foi trabalhoso porém não tão complexo de ser feito, apesar de todas as implementações. O diferencial é a visualização em tempo real do semáforo e do contador, e a implementação de um vetor com índice circular, que impede o overflow.

Conclusão

Por fim, com esse projeto pudemos aprender na prática muito sobre a matéria de AOC em si. Foi possível ver implementações que refletiam assuntos vistos em aula, mas de maneira mais visível e interativa. O principal ponto foi o aprimoramento não só na linguagem MIPS, mas também de toda lógica de processamento de um programa de baixo nível.

Os principais desafios enfrentados foram as implementações utilizando vetor, pois era uma matéria recente e que custou algumas pesquisas e horas de teste “bruto”. Além disso, a utilização do Bitmap Display foi bastante desafiadora por ser uma ferramenta muito trabalhosa de se utilizar, porém o resultado valeu a pena.

A inteligência artificial teve um papel muito importante de guia e de “tester” para o programa. Sua principal função foi a de sugerir melhores regras de negócio, melhores implementações e a de verificar o funcionamento do programa enquanto ainda não havia um display visual para testes reais.

Uma possível extensão mais complicada do programa seria a implementação de um semáforo de pedestres que também segue o mesmo princípio do semáforo já implementado, porém deve funcionar de maneira alternada.

Referências

Gemini 3.0 Pro

Documentação interna do Mars

https://www.youtube.com/watch?v=k0vSvJB8k3I