Relatório de Projeto: Simulador de Estufa Inteligente (IoT)

TAB

Gabriela Dobbert Sanches - 163760

Rafael de Carvalho Morel -

Thiago Azevedo Zanini - 148357

https://github.com/gabidsanches/monitoramento-Plantio

1. Resumo

Neste trabalho, desenvolvemos um simulador de "Estufa Inteligente" utilizando a linguagem Assembly MIPS. O objetivo principal foi criar um sistema capaz de atuar como um dispositivo embarcado (IoT) que monitora o microclima de uma plantação. O simulador lê dados de sensores (Umidade, Temperatura e pH), processa essas informações criando um histórico (média móvel) e toma decisões autônomas, como ligar a irrigação ou controlar o ciclo de luz (fotoperíodo) dependendo da fase de crescimento da planta. O projeto nos permitiu aplicar na prática o gerenciamento de registradores, memória e fluxo de execução em baixo nível.

2. Introdução

A escolha do tema "Estufa Inteligente" (baseado na sugestão de Central Climática) surgiu do interesse do grupo por aplicações de IoT no agro. Em um cenário real, microcontroladores simples precisam gerenciar múltiplas variáveis ambientais com eficiência e rapidez. Escolhemos simular esse ambiente porque ele exige uma lógica de controle contínua (loops infinitos) e respostas imediatas a sensores, o que é um cenário bom para a proposta do trabalho.

Este projeto foi fundamental para solidificar os conceitos de AOC. Ao contrário das linguagens de alto nível, onde muitas operações são mais intuitivas, aqui tivemos que lidar diretamente com a organização da CPU. Tivemos que gerenciar manualmente quais registradores usar (**`$t`** vs **`$s`**), entender como os dados são alinhados na memória (**`.word`**) e, principalmente, como proteger o contexto da execução ao usar a pilha (**`$sp`**) durante chamadas de função.

Ao longo do desenvolvimento tivemos algumas dificuldades, tais como: o uso correto de ponteiros e o manuseio da pilha foram complexos na prática, assim como a implementação de algoritmos matemáticos para o cálculo da média móvel e a conversão de horas por ciclo usando apenas inteiros. Tivemos, ainda, que gerenciar manualmente a escassez de registradores disponíveis, decidindo criteriosamente o uso de $t ou $s para garantir a integridade da execução.

---

3. Descrição Geral do Sistema

3.1 Objetivos do simulador

O simulador funciona como o firmware da estufa. Seu propósito é garantir que a planta receba as condições ideais de cultivo. Ele não apenas reage ao "agora", mas também entende um ciclo diário e a tendência dos dados (através da média de umidade),tambem automatiza a ativação de funções, simulando sensores.

3.2 Funcionalidades implementadas

O sistema conta com as seguintes funcionalidades:

- **Configuração de Cultivo:** O usuário define se a planta está na fase "Vegetativa" (precisa de 18h de luz) ou "Floração" (precisa de 12h de luz).

  Fase Vegetativa: Recomenda-se um ciclo de 18 horas de luz e 6 horas de escuridão (ou até 20/4) para promover um crescimento vigoroso da planta.

  Fase de Floração e Frutificação: Um ciclo de 12 horas de luz e 12 horas de escuridão é comumente utilizado para induzir e manter a produção de flores e frutos.

- **Simulação Temporal:** O programa recebe o numero de ciclos de leitura e divide ao longo de 24h.
- **Controle de Fotoperíodo:** Acionamento automático da lâmpada de cultivo baseada no horário simulado e no estagio da planta.
- **Leitura de Sensores e Histórico:** Entrada de dados simulada para Umidade, Temperatura e pH, com armazenamento das últimas 5 leituras da umidade em vetor.
- **Processamento de Sinais:** Cálculo de Média Móvel para a umidade.
- **Alertas e Atuação:** Acionamento de Bomba, Ventoinha e Corretor de pH baseado em limites de segurança e ciclo de luminosidade.

3.3 Arquitetura geral do programa

O código é estruturado em um loop principal podendo ser infinito (ou definido pelo usuário), simulando o ciclo de clock de um microcontrolador:

1. **Setup:** Define constantes e pede configurações iniciais.
2. **Loop Principal:**
   - Calcula o horário atual.
   - Verifica e atualiza o estado da luz.
   - Lê sensores via `syscall`.
   - Atualiza o vetor circular na memória.

    - Chama a função `calcular_media` (módulo separado).
    - Compara resultados e exibe alertas.
3. **Saída:** Exibe o status formatado em ASCII a cada ciclo.

---

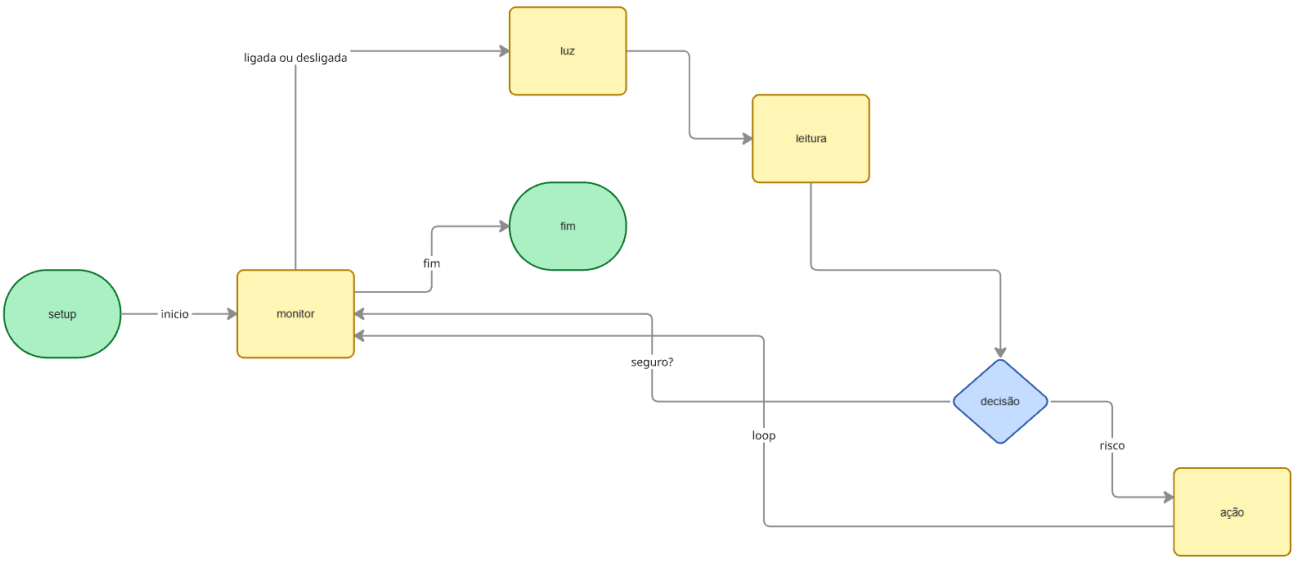
4. Máquina de Estados

4.1 Estados do sistema e transições

Podemos descrever o comportamento da estufa através de estados bem definidos. O sistema transita entre eles dependendo das entradas dos sensores e do contador de tempo:

1. **SETUP (Configuração):** Estado inicial onde os parâmetros de fase e duração são definidos. Transita para MONITORAMENTO automaticamente.
2. **MONITORAMENTO (ligado/desligado):** Estado padrão. O sistema verifica o horário para decidir entre os sub-estados "Luz Ligada" ou "Luz Desligada".
3. **ANÁLISE:** O sistema processa os dados (cálculo de média).
4. **ATUAÇÃO (Alerta):** Se algum parâmetro (Média Umidade, Temp, pH) violar os limites (**`.data`**), o sistema entra temporariamente neste estado para ativar os atuadores (Bomba, Ventoinha, etc.) e retorna ao monitoramento.

4.2 Diagrama de estados



---

5. Estruturas de Dados

Para gerenciar o histórico de umidade, implementamos um **Vetor Circular**.

- **Organização na Memória:** Definimos o vetor **`historico_umid`** com espaço para 5 inteiros (**`.word`**), totalizando 20 bytes.
- **Endereçamento:** Como o MIPS endereça por bytes, não podíamos apenas somar 1 ao índice. Usamos a lógica de `Endereço` **`Base + (Índice * 4)`**. Para fazer isso de forma eficiente, utilizamos a instrução **`sll`** (Shift Left Logical) com deslocamento de 2 bits.
- **Circularidade:** Para que o vetor guardasse sempre os últimos 5 valores sem estourar a memória, usamos um índice que vai de 0 a 4. Quando chega em 5, ele "dá a volta" para 0. Implementamos isso capturando o resto da divisão (`mfhi`) do contador pelo tamanho do vetor.

Além do vetor, utilizamos variáveis globais na seção `.data` para definir os limites de segurança (ex: **`LIMITE_SECO`, `PH_MIN`**), facilitando a calibração do sistema sem mexer no código lógico.

---

6. Funções principais

6.1 Função: `calcular_media`

Esta é a função mais complexa do projeto, responsável por implementar o algoritmo de média móvel que filtra oscilações nas leituras de umidade. Ela foi desenhada para ser uma sub-rotina independente, chamável por **`jal`** (Jump and Link), o que exigiu um cuidado especial com o gerenciamento de contexto na pilha.

Parâmetros de Entrada e Saída:

- **Entrada (`$a0`):** Recebe o endereço base do vetor **`historico_umid`**. Isso permite que a função seja genérica; se tivéssemos outros vetores, poderíamos reutilizar a mesma função.
- **Entrada (`$a1`):** Recebe o tamanho do vetor (fixo em 5).
- **Saída (`$v0`):** Retorna a média aritmética inteira calculada.

Gerenciamento da Pilha: Como a função utiliza registradores que devem ser preservados (`$s`) para realizar a soma, e como a instrução `jal` sobrescreve o endereço de retorno no registrador **`$ra`**, o uso da pilha foi obrigatório.

1. **Início:** No início da função, decrementamos o ponteiro da pilha (`$sp`) em 16 bytes.
2. **Salvamento:** Armazenamos na pilha os valores atuais de `$s0`, `$s1` e `$ra`. Isso "congela" o estado da função `main`.
    - **`$s0`**: Usado como acumulador da soma.
    - **`$s1`**: Usado como contador do loop da média.
    - **`$ra`**: Endereço para voltar ao loop principal.
3. **Restauração:** Antes de retornar (`jr $ra`), fazemos o processo inverso: carregamos os valores da pilha de volta para os registradores e incrementamos `$sp` em 16 bytes, restaurando o contexto original.

Lógica Algorítmica: A função executa um laço de repetição (loop) que vai de 0 até o tamanho do vetor (`$a1`).

- **Cálculo de Endereço Eficiente:** Para acessar a posição correta na memória, não podemos apenas somar o índice ao endereço base, pois cada palavra (**`.word`**) ocupa 4 bytes. Utilizamos a instrução **`sll $t0, $s1, 2`** (Shift Left Logical) para multiplicar o índice por 4. Optamos por **`sll`** em vez de **`mul`** por ser uma operação nativa e mais rápida em processadores RISC, que executa o deslocamento de bits para a esquerda, tivemos muita dificuldade nisso, então esse método foi sugerido pela IA.
- **Acesso à Memória:** O endereço final é calculado somando o deslocamento ao endereço base (**`add $t1, $a0, $t0`**). Em seguida, a instrução **`lw`** (Load Word) carrega o valor armazenado no vetor para um registrador temporário.
- **Acumulação e Divisão:** O valor lido é somado ao acumulador **`$s0`**. Ao final do loop, utilizamos a instrução **`div $s0, $a1`** (Soma / Tamanho) e movemos para **`$v0`** usando **`mflo`**.

6.2 Bloco Lógico: Atualização do Vetor

A ideia do vetor é fazer ele sempre salvar as 5 leituras mais recentes, porem todos os valores se iniciam com 50% de umidade.

Lógica de Ponteiros: O sistema mantém uma variável global **`indice_atual`** na memória.

1. **Cálculo do Offset:** Semelhante à função de média, lemos o índice atual e multiplicamos por 4 (**`sll`**) para encontrar o byte exato onde gravar a nova leitura.
2. **Escrita na Memória:** A leitura do sensor de umidade (que estava em **`$s0`**) é gravada na posição calculada usando **`sw`**. Isso sobrescreve qualquer valor antigo que estivesse lá.

Aritmética Modular para Circularidade: A nossa dificuldade aqui aqui foi fazer o índice voltar a 0 após chegar a 4 (tamanho 5). Em linguagens de alto nível, usaríamos o operador **%**.

- No MIPS, implementamos isso através da instrução **`div $t0, $t3`** (onde **`$t0`** é o índice incrementado e **`$t3`** é o tamanho 5).
- Apenas o resto da divisão é importante. O processador armazena o resto da divisão no registrador **`HI`**.
- Utilizamos `mfhi` **`$t0`** para capturar esse resto.
    - Exemplo: Se o índice for 4 e somarmos 1 -> 5. 5 dividido por 5 dá resto 0. O índice volta para o início automaticamente.

6.3 Lógica: Controle de Fotoperíodo e Tempo

Essa parte do código é responsável pela simulação de tempo, transformando os ciclos de leitura de sensores em intervalos dentro de 24h. Foi aqui que aplicamos a lógica matemática auxiliada pela IA.

Conversão de Escala (Regra de Três): Precisávamos mapear o contador do loop (**`i`**), que vai de 0 até o Total de Ciclos definido pelo usuário, para uma escala de 0 a 1440 (total de minutos em 24h).

- A fórmula implementada em Assembly foi: **`Minutos = (CicloAtual * 1440) / TotalCiclos`**.
- Utilizamos a instrução **`mul`** para multiplicar o contador atual pela constante 1440 (carregada em **`$t1`**). O resultado é um número grande, mas que cabe em um registrador de 32 bits (desde que o total de ciclos não seja muito grande).
- Em seguida, a instrução **`div`** divide esse produto pelo total de ciclos armazenado em **`$s6`**. O resultado (**`mflo`**) é o minuto exato do dia simulado.

Formatação para hora e minuto: Para exibir o horário no formato legível "14:30h", realizamos uma segunda etapa de processamento:

1. Dividimos os minutos totais por 60.
2. O quociente (**`mflo`**) representa as **Horas**.
3. O resto (**`mfhi`**) representa os **Minutos**.
4. Implementamos um desvio condicional (**`blt`**) para verificar se os minutos são menores que 10. Se forem, o código desvia para um pequeno bloco que imprime o caractere '0' antes do número, para que fique "14:05" em vez de "14:5".

Decisão acionamento da luz: Com o horário calculado, o sistema compara este valor com a variável de configuração da fase da planta (armazenada em **`$s5`**).

- Se a planta está na fase vegetativa, **`$s5`** vale 1080 (18 horas).
- A instrução **`ble $t2, $s5, luz_on`** verifica: "O tempo atual é menor ou igual a 1080?".
- Se **sim**, o fluxo desvia para o rótulo **`luz_on`**, que imprime o status "LIGADA".
- Se **não**, o fluxo segue naturalmente para o rótulo **`luz_off`**, desligando a luz.

6.4 Bloco Lógico: Sistema de Alertas e Atuação

Esse bloco do código serve para simular a ativação de ventoinhas, bomba d'agua e corretor de pH através de sensores. Ele é executado ao final de cada ciclo, após todas as leituras e cálculos. Sua função é comparar os dados processados com os limites de segurança definidos na seção **`.data`.**

Uso de Flags de Estado: Para evitar múltiplos **`syscalls`** de "Status OK" quando algo está errado, utilizamos um registrador temporário (**`$t9`**) como uma "Flag de Erro".

- No início da verificação, `$t9` é zerado.
- Se qualquer anomalia for detectada (seca, calor ou pH ruim), `$t9` recebe o valor 1.
- Ao final das verificações, uma instrução `bne $t9, $zero, proxima_iteracao` verifica se a flag foi alterada. Se foi (flag != 0), o sistema pula a mensagem de "Status Normal" e vai direto para o próximo ciclo. Isso garante que o usuário veja apenas os alertas críticos quando eles ocorrem.

Lógica da média calculada: Um ponto crucial desta função é que ela não avalia a umidade instantânea (`$s0`), mas sim a média calculada (`$s3`). Se o sensor oscilar rapidamente entre 39 e 41, a bomba não fica ligando e desligando freneticamente, pois a média suaviza essa variação. Já para Temperatura e pH, utilizamos os valores instantâneos (`$s1` e `$s2`), pois são parâmetros críticos onde a ação imediata é necessária para que a planta não seja prejudicada.

---

7. Uso de IA Generativa

7.1 Ferramentas de IA utilizadas

Utilizamos ferramentas de IA generativa (Gemini) de para auxiliar na lógica matemática da simulação das horas no ciclo de monitoramento.

7.2 Como a IA foi utilizada no projeto

Nossa principal dificuldade foi criar uma fórmula matemática que transformasse um contador de loops em um formato de horas e minutos proporcional ao total de ciclos escolhido pelo usuário. Pedimos para a IA uma lógica matemática para essa conversão.

7.3 Tabela de trechos gerados pela IA

Local do código	**Trecho gerado pela IA (Conceito)**	**Explicação escrita pelo grupo**	**Ajustes feitos**
Main Loop (Cálculo de Tempo)	Fórmula de Regra de Três: **Minutos = (Ciclo * 1440) / Total**	A IA forneceu a lógica de que o dia tem 1440 minutos e que devemos multiplicar o ciclo atual por esse valor e dividir pelo total definido pelo usuário. O resultado no registrador **LO** da divisão é o minuto atual.	A IA sugeriu usar registradores **$t0** e **$t1** para o cálculo. Porém, nós já estávamos usando **$t0** para controlar o índice do vetor circular. Trocamos os registradores para evitar conflito e adicionamos a divisão por 60 manual para separar horas e minutos na exibição.

Main Loop (Decisão de Luz)	Estrutura **if/else** para comparar tempo: **ble $tempo, $limite, on**	O código compara se o minuto atual é menor que o limite de luz da fase (Vegetativa/Floração). Se for, pula para ligar a luz.	Adaptamos os nomes dos labels para o padrão do nosso código (**luz_on**, **luz_off**) e integramos essa lógica dentro do nosso loop principal, que não existia no snippet da IA.
Função Calcular Média (Gestão da Pilha)	Estrutura de salvamento de contexto e otimização de endereçamento via sll (Shift Left Logical).	A gente não estava conseguindo imprimir os valores corretamente e nem armazenar, a IA indicou que o uso do ponteiro $ra e os registradores salvos estavam errados, alem de sugerir o uso de sll para multiplicar o índice por 4 bytes, evitando o uso de mul.	A IA sugeriu substituir a multiplicação tradicional (por 4) pelo sll para resolver a dificuldade que tivemos com o cálculo de endereços de memória Implementamos essa lógica no loop de soma e ajustamos o tamanho da pilha (16 bytes) para acomodar exatamente os registradores usados

7.4 O que tivemos que corrigir no código gerado pela IA

O principal problema no código sugerido pela IA foi a gestão de registradores. O snippet gerado assumia que poderia usar qualquer registrador temporário (**`$t`**). No entanto, no nosso código completo, esses registradores mantinham endereços de memória do vetor. Se tivéssemos usado o código da IA diretamente, ele teria sobrescrito o ponteiro do vetor, causando erro de segmentação ou gravação em memória errada. Tivemos que analisar o fluxo de dados e alocar registradores livres manualmente.

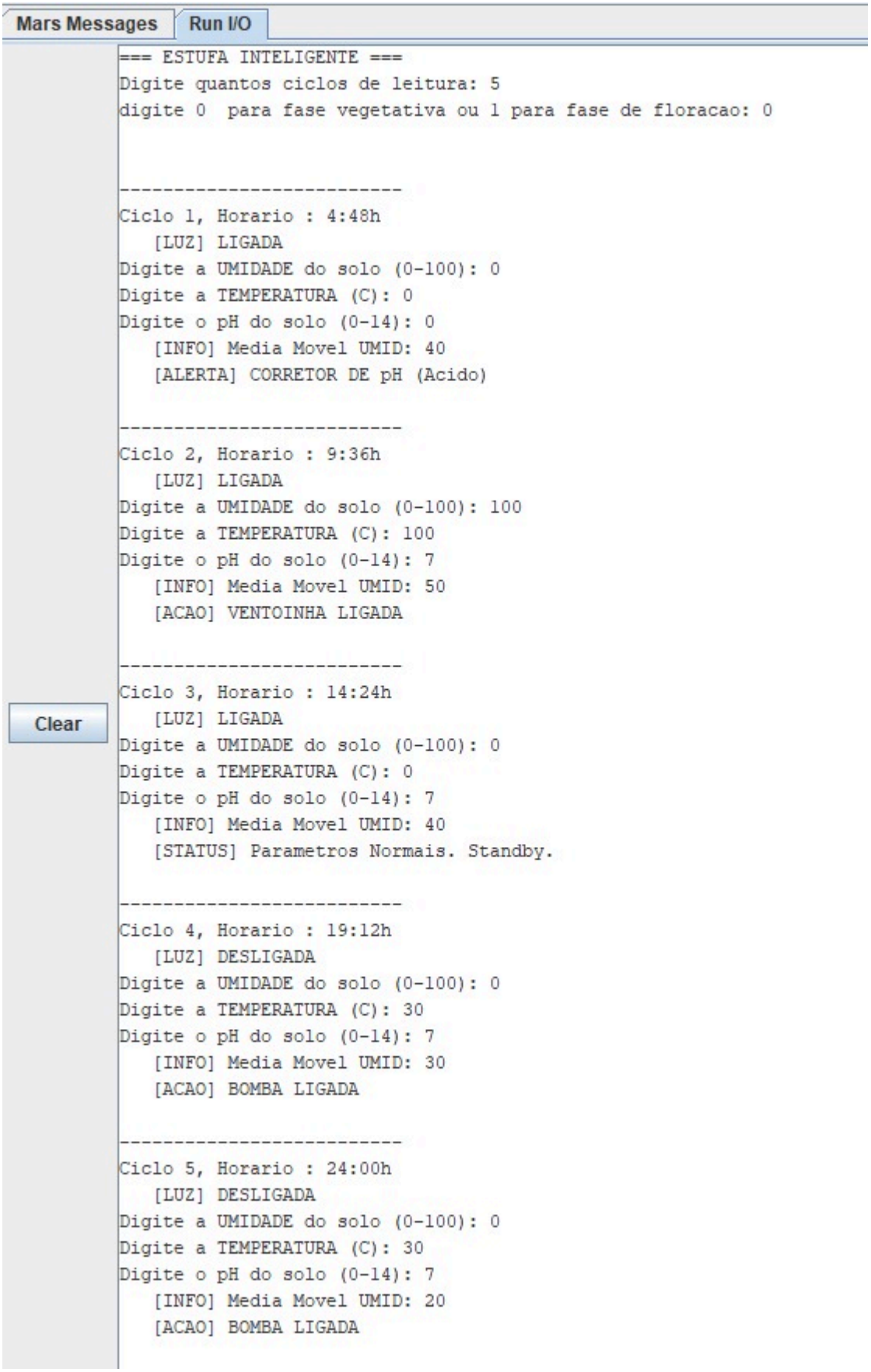
8. Testes e Resultados

8.1 Casos de teste utilizados

Para validar o sistema, utilizamos os seguintes cenários no simulador MARS:

1. **Caso Típico (Monitoramento):** Configuramos 5 ciclos. Inserimos valores normais (Umidade 60, Temp 25).
    - *Resultado:* O sistema mostrou o horário avançando e o status "Standby".
2. **Caso de Atuação (Seca):** Inserimos umidade baixa (30) repetidamente.
    - *Resultado:* Nos primeiros ciclos a bomba não ligou (pois a média ainda estava alta). Após o 3º ciclo, a média caiu abaixo de 40 e a mensagem **`[ACAO] BOMBA LIGADA`** apareceu, validando a lógica de média móvel.
3. **Caso de Fotoperíodo (Fim do dia):** Configuramos para fase de Floração (12h luz).
    - *Resultado:* Quando o horário simulado passou de 12:00h, o status da luz mudou automaticamente de "LIGADA" para "DESLIGADA".

8.2 Screenshots ou trechos de saída



8.3 Discussão dos resultados

O sistema comportou-se como esperado. A interface permitiu visualizar claramente a transição de estados e o funcionamento do "relógio" virtual. A média móvel provou ser eficaz para evitar instabilidade no controle da bomba.

---

9. Conclusão

O projeto foi importante para entendermos os detalhes por trás das linguagens de alto nível. Aprendemos na prática a importância do alinhamento de memória e a disciplina necessária para manipular a pilha sem corromper a execução do programa.

O uso da IA foi importante para a lógica matemática de tempo. A implementação da arquitetura, fluxo de dados e manipulação de hardware simulado exigiu total compreensão nossa, apesar de termos utilizado a IA para auxiliar na escolha de algumas funções para otimização do código (como `bll`).

O projeto foi divertido de fazer, pois conseguimos visualizar uma funcionalidade prática do cotidiano através de um código MIPS, aplicado à embarcados ou IoT.

---

10. Referências

- Material de Aula: Arquitetura e Organização de Computadores - Profa. Denise Stringhini.
- Documentação do Simulador MARS (Missouri State University).
- GREEN POWER. **Fotoperíodo**. Disponível em: https://greenpower.net.br/blog/fotoperiodo/.
- HOW does the stack work in assembly language? **Stack Overflow**, 2009. Disponível em: https://stackoverflow.com/questions/556714/how-does-the-stack-work-in-assembly-language.
- MIPS Assembly functions, what should be the structure (example swap). **Stack Overflow**, 27 abr. 2022. Disponível em: https://stackoverflow.com/questions/72035117/mips-assembly-functions-what-should-be-the-structure-example-swap.