

# Aquário Inteligente: Simulador de Dispositivo IoT em Assembly MIPS

Ana Beatriz R. Garcia (168.480)<sup>1</sup> Bruno de A. Correia (168.522)<sup>2</sup>  
Leonardo A. Araújo (168.873)<sup>3</sup>

<sup>1</sup> Instituto de Ciência e Tecnologia – Universidade Federal de São Paulo (UNIFESP)  
CEP 12247-014 – São José dos Campos – SP – Brasil

ana.garcia29@unifesp.br, bcorreia@unifesp.br,  
arazo.leonardo@unifesp.br

Link para repositório github: <https://github.com/Arazoleo/Aqu-rio-Inteligente>

**Resumo.** *Este trabalho apresenta o desenvolvimento de um simulador de dispositivo embarcado IoT (Internet of Things) em Assembly MIPS, implementando um sistema de monitoramento e controle de aquário inteligente. O projeto utiliza o simulador SPIM e implementa funcionalidades como leitura de sensores (temperatura, pH, oxigênio dissolvido e luminosidade), controle automático de atuadores (aquecedor, aerador e iluminação), histórico de leituras com buffer circular, cálculo de média móvel com detecção de tendência, e índice de qualidade da água. O sistema demonstra conceitos fundamentais de arquitetura de computadores, incluindo uso de pilha para gerenciamento de chamadas de função, manipulação de vetores, operações de ponto flutuante, e implementação de algoritmos clássicos em baixo nível.*

## 1. Introdução

O desenvolvimento de sistemas embarcados e dispositivos IoT representa uma área de crescente importância na computação atual. A compreensão de como esses sistemas funcionam em nível de hardware e software de baixo nível se torna fundamental para profissionais da área de computação. Este projeto apresenta a implementação de um simulador de aquário inteligente em Assembly MIPS, demonstrando como conceitos de arquitetura de computadores podem ser aplicados em cenários práticos de automação.

O aquário inteligente é um exemplo representativo de sistema IoT, pois integra múltiplos sensores para monitoramento ambiental, atuadores para controle automático, e lógica de processamento para tomada de decisões baseada em regras. O tema permite explorar diversos aspectos da programação em baixo nível, incluindo manipulação de dados numéricos (inteiros e ponto flutuante), estruturas de dados (vetores), controle de fluxo, e modularização através de funções.

O projeto foi desenvolvido utilizando o simulador SPIM, que implementa a arquitetura MIPS32, permitindo a execução e depuração de código Assembly em ambiente controlado. A implementação demonstra o uso adequado de convenções de chamada de função, gerenciamento de pilha, e utilização do coprocessador de ponto flutuante para operações com números decimais.

## 2. Arquitetura do Sistema

### 2.1. Visão Geral

O sistema Aquário Inteligente é organizado em módulos que simulam os componentes de um dispositivo IoT real:

- **Módulo de Sensores:** Simula a leitura de quatro tipos de sensores: temperatura (°C), pH, oxigênio dissolvido (mg/L) e luminosidade (%).
- **Módulo de Atuadores:** Controla três dispositivos: aquecedor, aerador e sistema de iluminação, com estados ON/OFF baseados em regras automáticas.
- **Módulo de Histórico:** Implementa um buffer circular para armazenamento das últimas 10 leituras de temperatura.
- **Módulo de Análise:** Calcula média móvel, detecta tendências (subindo/descendo/estável) e computa índice de qualidade da água.
- **Interface de Usuário:** Menu interativo ASCII com opções de operação manual, automática e visualização de dados.

### 2.2. Estrutura de Dados

O programa utiliza a seção `.data` para armazenar variáveis e estruturas de dados:

#### Variáveis de Sensores:

- `temp_atual`: Valor float da temperatura atual
- `pH_atual`: Valor float do pH atual
- `o2_atual`: Valor float do oxigênio dissolvido
- `luz_atual`: Valor inteiro (word) da luminosidade

#### Buffer Circular:

```
1 historico_temp: .space 40      # 10 floats x 4 bytes
2 indice_hist:   .word 0        # indice atual (0-9)
3 total_leituras: .word 0        # contador total
```

### 2.3. Tecnologias Utilizadas

**Simulador:** SPIM (MIPS32 Simulator) versão 9.1.24, executado via linha de comando em ambiente macOS.

**Arquitetura:** MIPS32 com coprocessador de ponto flutuante (FPU - Coprocessor 1).

#### Syscalls Utilizadas:

- Syscall 1: Impressão de inteiro
- Syscall 2: Impressão de float
- Syscall 4: Impressão de string
- Syscall 5: Leitura de inteiro
- Syscall 6: Leitura de float
- Syscall 10: Encerramento do programa

### 3. Funcionalidades Implementadas

#### 3.1. Menu Principal

O sistema apresenta um menu interativo com seis opções:

1. **Ler sensores manualmente:** Permite entrada de valores pelo usuário
2. **Modo automático:** Executa N leituras simuladas (1-10)
3. **Ver histórico de temperatura:** Exibe últimas leituras e média móvel
4. **Ver índice de qualidade:** Calcula e exibe pontuação 0-100
5. **Situação atual:** Exporta relatório completo do sistema
6. **Sair:** Encerra o programa

#### 3.2. Leitura de Sensores

A função `ler_sensores` solicita entrada do usuário para cada sensor:

```
1 ler_sensores:
2     addi $sp, $sp, -4          # prologo: reserva pilha
3     sw $ra, 0($sp)            # salva endereco de retorno
4
5     li $v0, 6                  # syscall 6 = ler float
6     syscall
7     s.s $f0, temp_atual        # salva temperatura
8
9     ...
10
11    jal adicionar_ao_historico  # salva no buffer
12
13    lw $ra, 0($sp)             # epilogo: restaura $ra
14    addi $sp, $sp, 4
15    jr $ra                     # retorna
```

#### 3.3. Exibição de Interface

A função `exibir_interface` mostra o painel de status com valores e classificação:

```
=====
TEMP: 26.0 C [OK]
pH:   7.0 [OK]
O2:   6.5 mg/L [OK]
LUZ:  50% [OK]
=====
```

A classificação é determinada comparando cada valor com seus limites usando instruções de ponto flutuante:

```
1 l.s $f0, temp_atual          # carrega temperatura
2 l.s $f2, temp_min            # carrega limite minimo
3 c.lt.s $f0, $f2              # compara: temp < min?
4 bclt exibir_temp_critico     # se verdadeiro, CRITICO
```

### 3.4. Controle de Atuadores

O sistema implementa regras automáticas para controle dos dispositivos:

Sensor	Condição	Atuador	Ação
Temperatura	$< 24^{\circ}\text{C}$	Aquecedor	Liga
Temperatura	$\geq 24^{\circ}\text{C}$	Aquecedor	Desliga
Oxigênio	$< 5.0\text{ mg/L}$	Aerador	Liga
Oxigênio	$\geq 5.0\text{ mg/L}$	Aerador	Desliga
Luminosidade	$< 30\%$	Iluminação	Liga
Luminosidade	$\geq 30\%$	Iluminação	Desliga

**Tabela 1. Regras de controle automático dos atuadores**

### 3.5. Verificação de Alertas

A função `verificar_alertas` exibe mensagens de aviso quando valores estão fora dos limites aceitáveis:

- Temperatura baixa: risco de peixes lentos
- Temperatura alta: risco de estresse térmico
- pH ácido ou alcalino: risco de vida para peixes
- Oxigênio crítico: risco de asfixia
- Luminosidade inadequada: risco para vegetação ou proliferação de algas

### 3.6. Histórico com Buffer Circular

O histórico de temperatura utiliza um vetor de 10 posições implementado como buffer circular:

```
1 adicionar_ao_historico:
2     la $t0, historico_temp    # endereco base
3     lw $t1, indice_hist      # indice atual (0-9)
4
5     sll $t2, $t1, 2          # offset = indice * 4
6     add $t0, $t0, $t2        # endereco = base + offset
7
8     l.s $f0, temp_atual
9     s.s $f0, 0($t0)          # armazena no vetor
10
11     addi $t1, $t1, 1          # incrementa indice
12     li $t3, 10
13     blt $t1, $t3, salvar      # se < 10, salva
14     li $t1, 0                # senao, volta para 0
15 salvar:
16     sw $t1, indice_hist
```

A instrução `sll $t2, $t1, 2` (Shift Left Logical) multiplica o índice por 4, pois cada float ocupa 4 bytes.

### 3.7. Cálculo de Média Móvel

A função `calcular_media_movel` implementa o algoritmo de média aritmética das últimas N leituras:

```
1 calcular_media_movel:
2     mtc1 $zero, $f4          # acumulador = 0.0
3
4 soma_loop:
5     bge $t1, $t5, dividir    # se i >= n, divide
6
7     sll $t2, $t1, 2
8     add $t3, $t0, $t2
9     l.s $f6, 0($t3)          # carrega valor[i]
10    add.s $f4, $f4, $f6       # soma += valor[i]
11
12    addi $t1, $t1, 1
13    j soma_loop
14
15 dividir:
16    mtc1 $t5, $f8             # n como inteiro
17    cvt.s.w $f8, $f8          # converte para float
18    div.s $f4, $f4, $f8       # media = soma / n
```

### 3.8. Detecção de Tendência

Após calcular a média móvel, o sistema detecta a tendência comparando o valor atual com a média:

- **SUBINDO:** temperatura atual  $> média + 0.5^{\circ}C$
- **DESCENDO:** temperatura atual  $< média - 0.5^{\circ}C$
- **ESTÁVEL:** diferença dentro de  $\pm 0.5^{\circ}C$

### 3.9. Índice de Qualidade da Água

O índice de qualidade varia de 0 a 100 pontos, com 25 pontos atribuídos para cada parâmetro dentro do range ideal:

Parâmetro OK	Pontos
Temperatura (24–28°C)	+25
pH (6.5–8.0)	+25
Oxigênio ( $\geq 5.0\text{ mg/L}$ )	+25
Luminosidade (30–70%)	+25
<b>Total Máximo</b>	<b>100</b>

**Tabela 2. Sistema de pontuação do índice de qualidade**

A classificação final é:

- 100 pontos: ÓTIMO
- 75-99 pontos: BOM
- 50-74 pontos: REGULAR
- 0-49 pontos: RUIM

### 3.10. Modo Automático

O modo automático simula leituras sequenciais com valores que variam gradualmente:

- Temperatura: oscila entre 22°C e 30°C (variação de  $\pm 0.5^{\circ}\text{C}$  por iteração)
- pH: varia junto com a temperatura
- Oxigênio: varia inversamente
- Luminosidade: incrementa de 5% em 5%, reiniciando em 30% ao passar de 80%

## 4. Requisitos Técnicos Atendidos

### 4.1. Mínimo de 3 Funções

O projeto implementa 10 funções além da main:

1. ler\_sensores: Leitura de valores dos sensores
2. exibir\_interface: Exibição do painel de status
3. verificar\_alertas: Verificação e exibição de alertas
4. controlar\_atuadores: Controle automático de dispositivos
5. adicionar\_ao\_historico: Armazenamento no buffer circular
6. exibir\_historico: Exibição do histórico de temperaturas
7. calcular\_media\_movel: Cálculo da média das leituras
8. calcular\_qualidade: Cálculo do índice de qualidade
9. exibir\_qualidade: Exibição da pontuação e classificação
10. exportar\_log: Exportação de relatório completo

### 4.2. Uso de Pilha

Todas as funções utilizam a pilha para preservar o endereço de retorno (\$ra):

```
1 funcao:
2     addi $sp, $sp, -4    # reserva 4 bytes na pilha
3     sw $ra, 0($sp)      # salva $ra
4
5     ... corpo da funcao ...
6
7     lw $ra, 0($sp)      # restaura $ra
8     addi $sp, $sp, 4    # libera pilha
9     jr $ra              # retorna
```

A função calcular\_qualidade também preserva o registrador \$s0:

```
1 calcular_qualidade:
2     addi $sp, $sp, -8    # reserva 8 bytes
3     sw $ra, 4($sp)      # salva $ra em sp+4
4     sw $s0, 0($sp)      # salva $s0 em sp+0
5
6     li $s0, 0           # usa $s0 como acumulador
7
8     ... calcula pontuacao ...
9
10    lw $s0, 0($sp)       # restaura $s0
11    lw $ra, 4($sp)       # restaura $ra
12    addi $sp, $sp, 8     # libera pilha
13    jr $ra
```

### 4.3. Uso de Vetor

O histórico de temperatura utiliza um vetor de 10 elementos:

```
1 historico_temp: .space 40    # 10 floats x 4 bytes
```

O acesso ao vetor é feito através de cálculo de endereço:

```
1 la $t0, historico_temp    # endereço base
2 sll $t2, $t1, 2           # offset = indice * 4
3 add $t0, $t0, $t2         # endereço = base + offset
4 l.s $f0, 0($t0)          # carrega elemento
```

### 4.4. Algoritmo Não Trivial

O projeto implementa dois algoritmos não triviais:

**Média Móvel:** Soma de N valores com divisão para cálculo da média, utilizando conversão entre tipos inteiro e ponto flutuante.

**Buffer Circular:** Estrutura de dados que reutiliza posições do vetor quando atinge o limite máximo, permitindo armazenamento eficiente das últimas N leituras.

### 4.5. Interface ASCII

O sistema apresenta interface baseada em texto com:

- Menu principal com opções numeradas
- Painéis formatados com bordas
- Status visual [OK], [ATENÇÃO], [CRÍTICO]
- Mensagens de alerta descritivas
- Relatório de situação completo

## 5. Uso de IA Generativa

### 5.1. Abordagem Adotada

Durante o desenvolvimento deste projeto, **não foram utilizados trechos de código gerados diretamente por IA generativa**. Todo o código Assembly MIPS foi escrito manualmente, garantindo total compreensão de cada instrução implementada.

A única aplicação de IA generativa foi de natureza **conceitual e educacional**, especificamente para esclarecer dúvidas sobre operações com valores de ponto flutuante, que eram desconhecidas antes do projeto.

### 5.2. Orientações Conceituais Obtidas

A IA foi consultada para explicar os seguintes conceitos relacionados ao coprocessador de ponto flutuante (FPU) do MIPS:

- **Registadores \$f0-\$f31:** Entendimento de que o MIPS possui um conjunto separado de registradores para operações com números decimais, acessados através do coprocessador 1.

- **Sufixo .s nas instruções:** Compreensão de que o sufixo .s indica operações com precisão simples (*single precision*, 32 bits), diferenciando de .d para precisão dupla.
- **Instrução mtc1:** Esclarecimento sobre a necessidade de usar mtc1 (Move To Coprocessor 1) para transferir valores entre registradores inteiros e de ponto flutuante.
- **Conversão de tipos:** Explicação sobre o uso de cvt.s.w para converter valores inteiros (word) para float, necessário para operações como divisão na média móvel.
- **Comparações de float:** Orientação sobre o uso de c.lt.s para comparação seguido de bclt/bclf para desvio condicional, diferente do padrão usado com inteiros.
- **Syscalls 2 e 6:** Informação sobre as syscalls específicas para impressão e leitura de valores float, que utilizam \$f12 e \$f0 respectivamente.

### 5.3. Justificativa

A consulta conceitual foi necessária pois o material didático da disciplina focava principalmente em operações com inteiros. As operações de ponto flutuante, embora documentadas no manual do MIPS, possuem sintaxe e comportamento distintos que não eram familiares ao desenvolvedor.

Após compreender os conceitos, todo o código foi implementado manualmente, aplicando o conhecimento adquirido às necessidades específicas do projeto (leitura de sensores, cálculo de média móvel, comparações de limites, etc.).

### 5.4. Não Houve Correções de Código Gerado por IA

Como não foram utilizados trechos de código gerados por IA, não houve necessidade de correções relacionadas a erros típicos de geração automática (instruções inválidas, registradores inexistentes, sintaxe incorreta, etc.). Todos os erros encontrados durante o desenvolvimento foram erros de digitação ou lógica do próprio desenvolvedor, corrigidos durante o processo normal de depuração.

## 6. Registradores Utilizados

Registrador	Uso no Projeto
\$ra	Endereço de retorno de funções
\$sp	Ponteiro da pilha
\$t0-\$t6	Valores temporários, índices, contadores
\$s0, \$s1	Valores preservados entre chamadas (contadores do modo auto)
\$a0	Argumento para syscalls (string ou inteiro)
\$v0	Código da syscall / valor de retorno
\$f0-\$f26	Operações de ponto flutuante
\$f12	Argumento para impressão de float (syscall 2)

**Tabela 3. Registradores utilizados no projeto**



### 7. Instruções de Ponto Flutuante Utilizadas

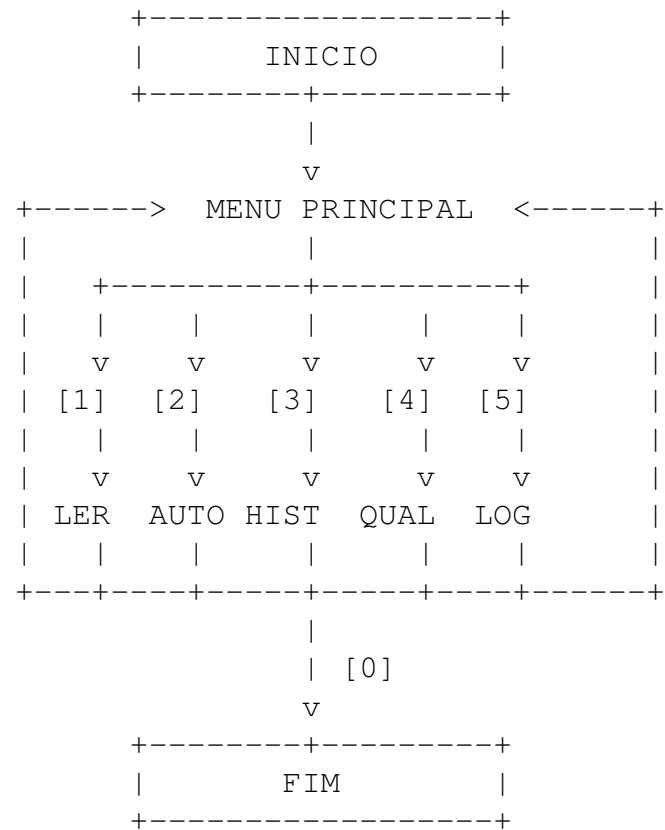
Instrução	Descrição
l.s \$f0, var	Carrega float da memória
s.s \$f0, var	Armazena float na memória
li.s \$f0, 1.5	Carrega constante float
add.s \$f0, \$f2, \$f4	Soma de floats
sub.s \$f0, \$f2, \$f4	Subtração de floats
div.s \$f0, \$f2, \$f4	Divisão de floats
neg.s \$f0, \$f2	Negação (inverte sinal)
c.lt.s \$f0, \$f2	Compara se menor
bclt label	Branch se comparação verdadeira
bclf label	Branch se comparação falsa
mtc1 \$t0, \$f0	Move inteiro para float (bits)
cvt.s.w \$f0, \$f0	Converte word para single

Tabela 4. Instruções de ponto flutuante utilizadas

### 8. Diagrama de Estados

O sistema opera como uma máquina de estados com o menu principal como estado cen-

tral:



### 9. Resultados e Testes

O programa foi testado com diversas combinações de entrada:

### **Teste 1 - Valores Normais:**

- Entrada: Temp=26°C, pH=7.0, O2=6.5mg/L, Luz=50%
- Resultado: Todos [OK], Qualidade 100/100, ÓTIMO

### **Teste 2 - Valores Críticos:**

- Entrada: Temp=20°C, pH=5.0, O2=3.0mg/L, Luz=10%
- Resultado: Todos [CRÍTICO], alertas exibidos, atuadores ligados

### **Teste 3 - Modo Automático:**

- Entrada: 10 leituras automáticas
- Resultado: Valores oscilando corretamente, histórico preenchido, média móvel calculada

## **10. Conclusão**

Este trabalho apresentou o desenvolvimento de um simulador de aquário inteligente em Assembly MIPS, demonstrando a aplicação prática de conceitos de arquitetura de computadores em um contexto de sistemas embarcados IoT.

A IA generativa serviu como ferramenta para compreensão de conceitos não estudados até então de forma rápida. Sendo assim, o uso com parcimônia de recursos vindos de LLM (*Large Language Models*) pode se tornar um fator positivo, ajudando no aprendizado e na praticidade.

Trabalhos futuros podem incluir: implementação de validação de entrada, adição de mais sensores e atuadores, simulação de tempo real com delays, e integração com arquivo para persistência de dados (syscalls 13-15).

## **Referências**

- [1] PATTERSON, D. A.; HENNESSY, J. L. *Computer Organization and Design: The Hardware/Software Interface*. 5th ed. San Francisco: Morgan Kaufmann, 2017.
- [2] LARUS, J. R. *SPIM: A MIPS32 Simulator*. Disponível em: <https://spimsimulator.sourceforge.net/>. Acesso em: 2024.
- [3] VOLLMAR, K.; SANDERSON, P. *MARS: MIPS Assembler and Runtime Simulator*. Disponível em: <https://courses.missouristate.edu/KenVollmar/MARS/>. Acesso em: 2024.
- [4] SWEETMAN, D. *See MIPS Run*. 2nd ed. San Francisco: Morgan Kaufmann, 2007.