UNIVERSIDADE FEDERAL DE SÃO PAULO
Instituto de Ciência e Tecnologia
Engenharia Biomédica

Sistema de Alarme de Incêndio Inteligente Baseado em Cálculo de Risco em MIPS

Discentes: Nicolas David da Cruz Santos , Davi de Oliveira Custódio, João Augusto Paixão Rocha, Bernardo Friske de Souza

Resumo. *O presente trabalho apresenta a criação de um simulador de alarme de incêndio com uso da arquitetura MIPS. A partir das métricas de temperatura e fumaça, foi elaborado um sistema que faz um cálculo de risco ponderado que informa como saída uma quantificação do risco e o estado do cenário atual, com este variando entre normal, de atenção, de alerta e de evacuação. Para isso, foram implementadas funções, laços, tratamento de entrada, pilha, lista circular, vetores, entre outros recursos da linguagem assembly na arquitetura MIPS. Foi possível concluir que, a partir da linguagem de baixo nível assembly, programas com aplicações reais podem ser implementados e, com associação a sensores externos e um dispositivo que integre hardware e software, podem ter uso efetivo.*

Palavras-chave: Simulador de Alarme de Incêndio. AOC. Hardware. Sistemas Embarcados

1. Introdução

Os sistemas embarcados estão presentes em muitos dispositivos do dia a dia, incluindo sensores e alarmes usados para monitorar ambientes. Para entender melhor como esses sistemas funcionam internamente, é comum usar arquiteturas didáticas como o MIPS, que permite observar de forma clara como operações são feitas diretamente por registradores e instruções de baixo nível (PATTERSON; HENNESSY, 2014).

Neste trabalho, foi feito o desenvolvimento de um simulador de sistema embarcado/IoT implementado em linguagem MIPS Assembly, seguindo as restrições e características de dispositivos de baixo nível. O objetivo do projeto é demonstrar como recursos limitados — como memória reduzida, ausência de bibliotecas de alto nível e necessidade de controle manual da pilha e da lógica interna — influenciam diretamente a construção de sistemas embarcados reais.

O sistema simulado tem como finalidade monitorar temperatura e nível de fumaça, calculando um índice de risco e determinando o estado do ambiente com base nesses parâmetros. O programa também permite ativação manual do alarme e exibição das últimas leituras registradas, implementando assim funcionalidades esperadas em dispositivos de segurança residenciais ou industriais.

2. Fundamentação Teórica

O desenvolvimento de sistemas embarcados baseados em MIPS exige a compreensão de princípios fundamentais da organização de computadores, da arquitetura MIPS e do funcionamento de instruções de baixo nível. A arquitetura MIPS é amplamente utilizada em ensino e pesquisa por sua simplicidade estrutural, formato fixo de instruções e conjunto reduzido de operações, o que a torna adequada para explorar conceitos de pilha, registradores,

memória e chamadas de sistema. No contexto do projeto, é essencial a compreensão dos seguintes elementos teóricos

2.1 Arquitetura MIPS

O MIPS é um processador RISC (Reduced Instruction Set Computer), caracterizado por um conjunto pequeno e eficiente de instruções. Cada instrução possui formato fixo de 32 bits, o que simplifica o pipeline e torna o comportamento do hardware mais previsível. O uso de registradores — como os temporários ($t0–$t9), salvos ($s0–$s7), de argumentos ($a0–$a3) e retorno ($v0–$v1) — é implementado para escrita de funções, chamadas de procedimento e manipulação de dados. Esse tipo de organização favorece a execução eficiente de operações e é amplamente utilizada em arquiteturas reais (STALLINGS, 2010).

Além disso, neste trabalho, a organização da memória é feita em segmentos .data e .text, o que resulta em declaração das variáveis globais, buffers, mensagens no primeiro segmento e o código executável no segundo. Isso facilita a compreensão da estrutura interna do código e do uso das informações.

2.2 Conjunto de Instruções e Syscalls

O programa faz uso de instruções aritméticas (add, mul, div), de controle de fluxo (beq, bgt, j, jal, jr), de manipulação de memória (lw, sw, la, lb) e chamadas ao sistema (syscall). As syscalls fornecidas pelo simulador MARS ou SPIM permitem entrada e saída de dados, leitura de strings e encerramento do programa, criando uma interface simples entre o software e o ambiente de simulação.

2.3 Pilhas, Funções e Vetores

No projeto, a pilha é utilizada para guardar valores temporários, organizar chamadas de funções e manter o estado do programa. A manipulação direta do ponteiro de pilha (usando $s0) mostra como variáveis locais e endereços de retorno são armazenados e restaurados durante a execução, reforçando os princípios de alocação dinâmica e escopo (PATTERSON; HENNESSY, 2005).

As funções desempenham papel essencial na modularização do código, permitindo dividir o programa em partes menores e mais fáceis de entender. Cada chamada de função exige salvar o contexto anterior — como registradores e o endereço de retorno — na pilha, garantindo que o fluxo de execução possa continuar corretamente após a conclusão da sub-rotina. Esse mecanismo ajuda o aluno a compreender como o MIPS controla o fluxo e preserva informações entre diferentes blocos de código (WEBER, 2008).

O projeto também faz uso de vetores, que são armazenados no segmento .data e acessados por meio de endereçamento baseado em deslocamentos. O uso de indexação com multiplicação por 4 (devido ao tamanho de 4 bytes das palavras) apresenta conceitos de acesso sequencial, aritmética de ponteiros e manipulação de estruturas lineares. Esse tipo de operação é fundamental para entender como a memória é organizada.

2.4 Lógica de Sensores e Processamento de Dados

Sistemas embarcados reais frequentemente realizam leitura de sensores, interpretação de sinais e tomada de decisão. O projeto simula esse comportamento ao implementar leitura de dados de temperatura e fumaça, validação de entrada, cálculo de risco baseado em fórmulas simples, armazenamento circular das últimas leituras e acionamento automático ou manual do estado de alarme. Esses mecanismos espelham características comuns em sistemas embarcados reais, como alarmes ambientais, microcontroladores e sistemas de monitoramento.

3. Metodologia

O desenvolvimento do projeto foi conduzido com base na implementação incremental de um sistema simulado na arquitetura MIPS, utilizando o ambiente MARS. Ele consistiu em etapas envolvendo o planejamento lógico, elaboração incremental das funcionalidades, realização de ajustes e adição de detalhes.

No planejamento lógico, inicialmente foram definidos os requisitos do programa, incluindo leitura de valores (temperatura e fumaça), validação de entrada, armazenamento das últimas medições e acionamento de um alarme. Também foi planejado como cada funcionalidade seria distribuída entre funções separadas, visando modularidade e clareza. Outra etapa essencial, foi a definição da fórmula ponderada para o cálculo de risco, que foi formulada a partir de valores típicos de temperatura e níveis de fumaça registrados em cenários reais de operação normal, condições de pré-incêndio e episódios de incêndio. A concentração de fumaça é ponderada com maior peso, considerando tanto os riscos diretos de asfixia quanto sua elevada capacidade de dispersão no ambiente, mesmo na ausência de temperaturas elevadas. A fórmula é apresentada a seguir:

$$RiscoCalculado = \frac{18 \times Temperatura + 32 \times Fumaça}{120}$$

Além disso, é definido o valor 100 como limite para Risco, assim:

$$RiscoResultante = RiscoCalculado,\ se\ RiscoCalculado \leq 100$$

$$RiscoResultante = 100,\ se\ RiscoCalculado > 100$$

Tendo a lógica do nosso programa elaborada, o próximo passo foi a implementação em si. Para isso, o primeiro desafio enfrentado foi a entrada de dados, mais especificamente, como salvaríamos uma entrada que poderia ser tanto caractere como número e que seu tipo não poderia ser conhecido com antecedência. Essa complicação envolve a característica do assembly de exigir a definição do tipo da variável para atribuir um valor à ela, e a solução usada foi receber tanto as letras quanto os números da entrada como caracteres e depois fazer

a eventual conversão entre números e letras quando necessária. Assim, foi feita a função “lendoEntrada”, que recebe a entrada do usuário e faz o direcionamento para a parte do código responsável para tratar cada um dos seguintes casos de entrada:

- caractere “e”: encerrar o programa;
- caractere “b”: ativação manual do alarme;
- caractere “l”: listar últimas leituras feitas;
- caractere numérico: valor da temperatura ou da fumaça;
- outro caractere: entrada inválida.

Para o encerramento do programa, é feita a chamada de sistema usando registrado v0. Para a ativação manual do alarme, move-se para a função “acionarAlarmeManualmente”, que gera como saída o valor de risco 100 e o estado evacuar. Quanto à lista das últimas leituras, tem-se a função “ListarUltimasLeituras”, que verifica se já foram feitas leituras a partir de um registrador de controle a2, e, com a confirmação, executa uma rotina de impressão dos valores de temperatura e fumaça usando buffers circulares que armazenam últimas medidas lidas. Esse armazenamento é feito a partir da própria função “lendoEntrada”, com um loop acionado cada vez que um caractere numérico é encontrado, que converte os caracteres numéricos nos seus números correspondentes usando os valores ASCII, pilhas e as operações de soma e multiplicação para juntar cada dígito do número de acordo com a grandeza, e encaminha os pares de valores temperatura fumaça para a rotina “GravarLeitura”, que salva os valores em um buffer circular “UltimasLeiturasT” para temperatura e outro, “UltimasLeiturasT”, para fumaça. Por fim, para o caso da entrada não ser válida, é chamada a rotina “ErroDeLeitura” caso o problema seja no formato da entrada e “Erro2” caso não tenha nenhum registro de leitura para ser listado com o comando “l”.

Com as entradas gravadas, é possível aplicar o cálculo do risco e dos estados para medições de temperatura e fumaça. Essa etapa é feita na função “CalcularRisco”, que faz uso das operações “add”, “mult” e “div” para a implementar a fórmula e também utiliza a instrução condicional beq para tratar o caso em que o valor calculado ultrapassa o limite permitido, ajustando o resultado para o valor 100. Com o cálculo feito, a função “OutputAlarme” é chamada para definir o estado e realizar a impressão dos resultados. Para isso, são feitas comparações com o valor do risco para definir em qual intervalo ele se encontra: se for menor que 10, o estado é normal; entre 10 (inclusive) e 30, de atenção; entre 30 (inclusive) e 50, de alerta e entre 50 (inclusive) e 100 (inclusive) de evacuação. Ao fim, temos os valores do cálculo de risco e do estado para impressão.

No que diz respeito à organização da memória, é notável que o segmento “.data” é responsável pela declaração das mensagens de interface, as variáveis auxiliares de controle, os estados do alarme e os vetores destinados ao armazenamento sequencial das leituras. Essa etapa permitiu estruturar a memória separando dados estáticos e estruturas auxiliares de maneira coerente com o funcionamento de sistemas embarcados reais. No segmento “.text” é onde ocorre a implementação da lógica do código já mencionada.

Após a implementação das funcionalidades principais, procedeu-se aos testes e ajustes no simulador MARS. O programa foi testado repetidas vezes no MARS, usando entradas diferentes para verificar se o sistema estava validando os dados corretamente, calculando o risco de forma adequada e armazenando as leituras no vetor circular. Esses testes também serviram para ajustar erros e garantir que os níveis do alarme fossem exibidos corretamente.

Para elaboração do código, foi feito o uso de Inteligência Artifical (IA), mais especificamente, o modelo de linguagem do ChatGPT da OpenIA. Sua utilização foi limitada ao aprendizado de recursos da linguagem assembly em MIPS não vistos em aula, como funções, vetores circulares e pilhas. Isto é, o uso de IA resumiu-se somente para aprendizado de como tais recursos funcionam, bem como de que forma podem ser utilizados.

4. Resultados

Com o sistema implementado, foi possível realizar testes para simular o funcionamento do dispositivo embarcado. Observou-se que o programa realiza corretamente o cálculo da função de risco, apresentando como esperado os estados de severidade e os valores de risco. Isso pode ser observado na figura 1, que apresenta a entrada de valores que resultam em cada um dos estados como saída.

```
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
20 5
RISCO: 4/100
ESTADO: normal
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
50 20
RISCO: 12/100
ESTADO: atenção
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
100 100
RISCO: 41/100
ESTADO: alerta
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
150 200
RISCO: 75/100
ESTADO: evacuação
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
250 300
RISCO: 100/100
ESTADO: evacuação
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
```

Figura 1: Exemplos de entradas e saídas para cada estado de risco.

```
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
20 5
RISCO: 4/100
ESTADO: normal
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
50 20
RISCO: 12/100
ESTADO: atenção
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
100 100
RISCO: 41/100
ESTADO: alerta
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
150 200
RISCO: 75/100
ESTADO: evacuação
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
250 300
RISCO: 100/100
ESTADO: evacuação
```

Figura 1: Exemplos de entradas e saídas para cada estado de risco.

Note que, assim como o previsto, o valor do risco não ultrapassa 100, mesmo que o cálculo inicial resulte em valores maiores que 100, como no quinto caso.

Também foi feita a verificação da ativação manual de emergência pelo usuário. Com ela, observou-se como saída o estado de evacuação com valor de risco 100, como apresentado na figura 2.

```
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
b
RISCO: 100/100
ESTADO: evacuação
Alarme ativado manualmente
```

Figura 2: Exemplo de ativação manual de emergência.

```
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
b
RISCO: 100/100
ESTADO: evacuação
Alarme ativado manualmente
```

Figura 2: Exemplo de ativação manual de emergência.

Ademais, foram testadas duas funcionalidades essenciais. A primeira é de listagem das últimas leituras, possível a partir do uso de um buffer circular. O resultado é apresentado na figura 3. A segunda é a resposta à entradas inválidas, ou seja, informe de erro caso as entradas não sigam algum dos formatos especificados ou caso for solicitada a listagem das últimas leituras sem que haja registros prévios. A saída para esses casos é apresentada na figura 4.

```
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
l
(Temperatura | fumaça)
20 5
(Temperatura | fumaça)
50 20
(Temperatura | fumaça)
100 100
(Temperatura | fumaça)
150 200
(Temperatura | fumaça)
250 300
```

Figura 3: Listagem das últimas marcações feitas.


```
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
l
(Temperatura | fumaça)
20 5
(Temperatura | fumaça)
50 20
(Temperatura | fumaça)
100 100
(Temperatura | fumaça)
150 200
(Temperatura | fumaça)
250 300
```

Figura 3: Listagem das últimas marcações feitas.

```
ERRO: O sistema detectou um erro na leitura de dados, verificar sensores.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
a
ERRO: O sistema detectou um erro na leitura de dados, verificar sensores.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
ab
ERRO: O sistema detectou um erro na leitura de dados, verificar sensores.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
440
ERRO: O sistema detectou um erro na leitura de dados, verificar sensores.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
440 200 300
ERRO: O sistema detectou um erro na leitura de dados, verificar sensores.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
l
ERRO: Nenhuma leitura feita.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
```

Figura 4: Detecção de entradas inválidas.

```
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
a
ERRO: O sistema detectou um erro na leitura de dados, verificar sensores.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
ab
ERRO: O sistema detectou um erro na leitura de dados, verificar sensores.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
100
ERRO: O sistema detectou um erro na leitura de dados, verificar sensores.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
100 200 300
ERRO: O sistema detectou um erro na leitura de dados, verificar sensores.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
l
ERRO: Nenhuma leitura feita.
```

Figura 4: Detecção de entradas inválidas.

Por fim, a Figura 5 ilustra o uso do comando “e” para encerrar o programa. Após sua execução, o sistema finaliza imediatamente a aplicação, impedindo a realização de novas entradas de dados pelo usuário.

```
ERRO: Nenhuma leitura feita.
Insira (Temperatura Fumaça) separadas por espaço OU
insira o código 'b' para ativação manual do alarme OU
o código 'l' para listar as ultimas leituras. Digite 'e' para encerrar.
e

-- program is finished running --
```

Figura 5: encerramento do programa com comando “e”.

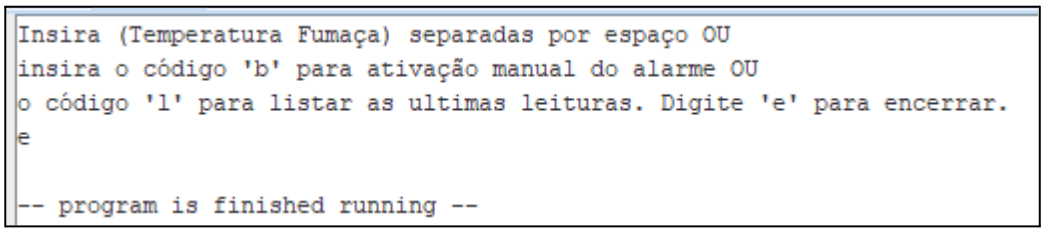

Figura 5: encerramento do programa com comando “e”.

A partir da análise dos resultados, conclui-se que o sistema se mostrou funcional, estabelecendo com sucesso a interação com o usuário, a partir de mensagens exibidas através das chamadas de sistema, com sentenças diretas e simples. Também, foram executadas com êxito as fórmulas de cálculo de risco e do estado de emergência, apresentando a saída na tela para o usuário, assim como a ativação manual. Ademais, foram feitas a validação da entrada, cálculos internos e exibição das mensagens, sem perda de dados e preservando os registradores por meio do uso adequado da pilha, o que permitiu a exibição das últimas leituras e a verificação de entradas válidas

5. Conclusão

O projeto em MIPS ajudou a colocar em prática conceitos importantes da disciplina de Arquitetura e Organização de Computadores. A implementação permitiu entender melhor como a pilha funciona, como os registradores são usados, como a memória é organizada e como o código pode ser dividido em funções. O uso de vetores e o trabalho direto com endereços também ajudaram a compreender como os dados são manipulados em baixo nível.

A simulação do monitoramento de temperatura e fumaça mostrou como sistemas embarcados fazem leituras de sensores, tratam informações e tomam decisões automáticas. Mesmo sendo um projeto didático, ele mostrou como a linguagem assembly pode ser usada para implementar programas com aplicações reais, neste caso, de monitoramento ambiental. Assim, é possível entender o uso dessa abordagem em sistemas funcionais e efetivos do mundo real com a associação de sensores externos e a um dispositivo que integre hardware e software.

Referências Bibliográficas

PATTERSON, David A.; HENNESSY, John L. ***Organização e Projeto de Computadores: A Interface Hardware/Software***. 5. ed. Rio de Janeiro: Elsevier, 2014.

TAVARES, M. J.; BARBOSA, L. C. **Metodologias Ativas no Ensino de Hardware: Uma Revisão Sistemática**. *Revista Brasileira de Informática na Educação*, v. 27, n. 1, p. 112–125, 2019.

STALLINGS, W. **Arquitetura e Organização de Computadores**. 10. ed. São Paulo: Pearson, 2017.

WEBER, R. *Assembly MIPS: Arquitetura e Programação*. 2. ed. Porto Alegre: Sagra Luzzatto, 2008. *(Caso não queira essa referência, posso remover — ela é adequada à parte de funções e pilha.)*