

# Relatório de Construção da Agenda de Compromissos: AppointmentAgenda

Luiz Felipe Diniz Costa 13782032

Rafael Jun Morita 10845040

# Índice

<b>1. Introdução</b>	<b>3</b>
<b>2. Solução proposta</b>	<b>3</b>
2.1. Funcionalidades	3
2.2. Questões técnicas	5
<b>3. Processamento da informação</b>	<b>13</b>
3.1. Dados de entrada	13
3.2. Processamento	14
3.3. Dados de saída	14
<b>4. Desafios</b>	<b>15</b>
<b>5. Conclusão</b>	<b>16</b>
<b>6. Código Fonte</b>	<b>17</b>

# 1. Introdução

O presente relatório descreve o desenvolvimento do programa AppointmentAgenda, realizado como parte do trabalho da disciplina de [Organização e Arquitetura de Computadores](#) no Instituto de Ciências Matemáticas e de Computação (ICMC). O objetivo deste projeto foi criar uma ferramenta de gerenciamento de eventos em linguagem Assembly MIPS, proporcionando uma maneira eficiente e organizada de agendar compromissos.

## 2. Solução proposta

O AppointmentAgenda foi projetado visando tornar-se uma solução completa e intuitiva para a gestão de eventos. O objetivo principal foi oferecer aos usuários uma ferramenta completa e de fácil utilização. Dessa forma, a prioridade na construção deste programa foi garantir a ausência de conflitos na agenda, incorporando um sistema inteligente de detecção de sobreposições de horários, assegurando que os eventos agendados não entrem em conflito, mantendo assim a organização e a integridade do planejamento dos eventos.

Essa solução proporciona a flexibilidade necessária para ajustar e remover eventos conforme as demandas de uma agenda dinâmica, garantindo uma experiência eficiente e livre de complicações para os usuários, como será detalhado no próximo tópico.

### 2.1. Funcionalidades

Detalhes do Evento:

Cada evento registrado no AppointmentAgenda contém informações essenciais, como nome, data, hora de início e hora de término. Esses detalhes fornecem uma visão abrangente do cronograma de compromissos.

Resolução de Conflitos:

Implementamos um sistema automático de detecção de conflitos. Se um novo evento entra em conflito com um evento existente em termos de horário, o programa impede

o registro desse novo evento. Isso garante que não haja sobreposições ou conflitos de agendamento.

#### Modificação de Eventos:

Os usuários têm a capacidade de editar e modificar eventos previamente registrados. Essa funcionalidade é essencial para fazer ajustes nos detalhes dos eventos, como alterar o nome, a data ou o horário, proporcionando flexibilidade ao gerenciamento da agenda.

#### Remoção de Eventos:

O programa permite a remoção de eventos. Isso possibilita que eventos desatualizados ou cancelados sejam facilmente removidos da agenda, mantendo-a organizada e precisa.

#### Organização Cronológica:

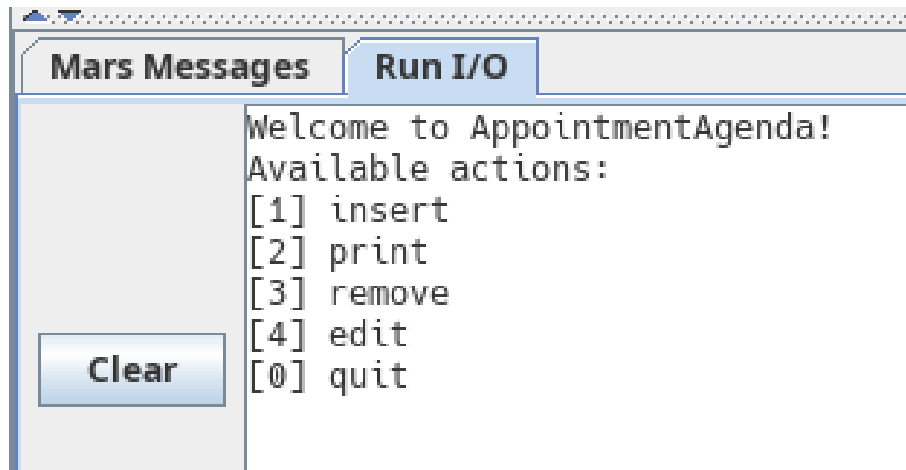
A organização dos eventos é apresentada de forma cronológica, exibindo-os ordenadamente com base na data e hora de realização. Isso proporciona aos usuários uma visualização clara e sequencial de todos os compromissos agendados, facilitando o acompanhamento do cronograma de atividades.

#### Detecção de Inputs Inválidos:

Para garantir a precisão e integridade dos dados inseridos, o sistema é capaz de identificar inputs inválidos. Caso o usuário insira um dia inválido, horas fora do intervalo correto ou configure a hora de término antes da hora de início, a agenda prontamente emite um alerta indicando a entrada inválida. Além disso, inputs que contenham minutos excedendo 59, como por exemplo "10.60", são imediatamente identificados como erro. Essa funcionalidade de detecção contribui significativamente para prevenir inconsistências nos dados, garantindo que apenas informações precisas e pertinentes sejam registradas e mantendo a integridade dos dados armazenados na agenda.

## 2.2. Questões técnicas

Para elaborar o fluxo de trabalho da agenda, foi desenvolvido um menu composto por 5 opções destinadas aos usuários. Esta estrutura pode ser visualizada na imagem a seguir:



Cada opção desse menu representa uma das funções principais do programa, que são chamadas quando o usuário digita o número correspondente, internamente as funções principais podem usar tanto as outras principais como as secundárias no processo de execução da tarefa. Importante ressaltar que como estamos trabalhando com programação de baixo nível, tudo é feito explicitamente por meio de “jumps”.

A função ‘insert’ pede um dado de cada vez ao usuário e armazena-os em variáveis auxiliares.

```
Exemplo da função Insert

# this function inserts a new event
insert:

# printing the event name question
li $v0, 4
la $a0, insert_eventName
syscall

# eventCounter starts in 1, so we need to multiply it by MAX_LENGTH_EVENT_NAME
# to get the correct position in the array
lw $t0, eventCounter
mul $t0, $t0, 50 #MAX_LENGTH_EVENT_NAME

# reading the event name
li $v0, 8
la $a0, aux_eventName
li $a1, 50 #MAX_LENGTH_EVENT_NAME
syscall
```

Aqui, o programa solicita ao usuário que insira o nome do evento, armazenando-o na variável auxiliar 'aux\_eventName'. O mesmo procedimento é realizado para coletar o dia, hora de início e hora de término do evento, sendo que esses valores são verificados para garantir sua validade - dias devem estar entre 1 e 31, horas entre 00:00 e 23:59. Em caso de valores inválidos, o programa interrompe o processo e exibe uma mensagem de erro ao usuário. Uma vez que os valores são armazenados nas variáveis auxiliares, a função 'compareDay' é acionada.

```
Exemplo da função CompareDay

compareDay:

    # using auxCounter set to value 1
    li $t1, 1
    lw $t2, eventCounter

    # compare if the day inserted already exists withing the eventsDay array
    loop_compareDay:

        # if auxCounter is equal to eventCounter, we have compared all days, the
        # value inserted is the biggest
        bge $t1, $t2, exit_compareDay
```

Neste trecho, é inicializado um contador que controla o término do loop. Enquanto esse contador alcança o número de eventos registrados, o loop continua. Durante a iteração, o dia armazenado na variável auxiliar é comparado com o dia de cada evento no array. Ao encontrar dias correspondentes, o programa invoca a função 'compareHour'.

```
Exemplo da função CompareDay

    # if the day inserted($s0) is equal to current day($t4) in the array, we need
    # to compare the hours
    beq $t4, $s0, compareHour

    exit_compareHour:

    # if the day inserted is smaller than any day in the array, we need to insert
    # it in current position
    blt $s0, $t4, sortArray

    # Increment auxCounter
    addi $t1, $t1, 1

    j loop_compareDay
```

Dentro da função ‘compareHour’, ocorre a verificação de possíveis conflitos de horários. Caso não haja conflito, o dia a ser inserido (com todos os seus dados armazenados nas variáveis auxiliares) é inserido de acordo com a ordem cronológica, posicionando-se antes ou depois conforme necessário.

```
Exemplo da função CompareHour

compareHour:
    mul $t5, $t1, 4 #MAX_LENGTH_HOUR

    # store the start time and end time of the event inserted conffliction and atual
    # event
    l.s $f13, aux_eventStartTime
    l.s $f14, aux_eventEndTime
    l.s $f15, eventsStartTime($t5)
    l.s $f16, eventsEndTime($t5)

    # verify if the event inserted conffliction with the atual event
    c.lt.s $f14, $f15 # if aux_eventEndTime < eventsStartTime
    bc1t sortArray # if true we insert in the current position and sort the array

    # if eventsEndTime($t5) < aux_eventStartTime
    c.lt.s $f16, $f13
    # if true we insert in the next position and sort the array
    bc1t exit_compareHour

    c.eq.s $f14, $f15 # if aux_eventEndTime == eventsStartTime($t5)
    bc1t errorInsert # if true we print the errorInput message

    c.eq.s $f16, $f13 # if eventsEndTime($t5) == aux_eventStartTime
    bc1t errorInsert # if true we print the errorInput message
```

Se não houver conflito de dia, o evento é colocado na posição atual se o dia for menor que o dia que está armazenado no array na posição atual.

```
Exemplo da função CompareHour

# if the day inserted is smaller than any day in the array, we need to insert
# it in current position
b.lt $s0, $t4, sortArray
```

Todavia, a função ‘sortArray’ realiza o armazenamento dos dados do evento, provenientes das variáveis auxiliares, na posição apropriada dentro do array. Em seguida, realiza deslocamentos dos elementos conforme a ordem estabelecida.

```
Exemplo da função SortArray

mul $t3, $t1, 4 #MAX_LENGTH_DAY
# store day inserted in the array
sw $s0, eventsDay($t3)
```

Se a inserção ocorrer no meio ou no início do array, todos os eventos a partir da posição atual são deslocados uma posição para a direita.

```
Exemplo da função SortArray

loop_sortArray:

# $t1=auxCounter, $t2=eventCounter, $t3=position in the array, $t4=day in the
# array
beq $t1, $t2, exit_sortArray # if auxCounter is equal to eventCounter, the array
# is sorted
addi $t1, $t1, 1 # Increment auxCounter($t1)
mul $t3, $t1, 4 # position in the array, auxCounter * 4

# event day in the array sorted
lw $t5, eventsDay($t3) # day in the array
sw $t4, eventsDay($t3) # store day in the array in $t4
move $t4, $t5 # move day in the array to $t4
```

Porém, se o ‘compareDay’ passar por todos os elementos sem colisão, o dia a ser inserido será inserido no fim do array.

Neste momento ocorre a inserção:

```
Exemplo da função CompareDay

# if auxCounter is equal to eventCounter, we have compared all days, the value
inserted is the biggest
bge $t1, $t2, exit_compareDay
```



```
Exemplo da função CompareDay:

exit_compareDay:
    # day greater than any event, so we can insert it in the end of the array

    # event day in the array sorted
    sw $s0, eventsDay($t0)

    # event start time in the array sorted
    l.s $f12, aux_eventStartTime
    s.s $f12, eventsStartTime($t0)

    # event end time in the array sorted
    l.s $f13, aux_eventEndTime
    s.s $f13, eventsEndTime($t0)

insertAuxEvent:
    # set t0 appropriate position in the array
    lw $t0, eventCounter
    mul $t0, $t0, 50 #MAX_LENGTH_EVENT_NAME

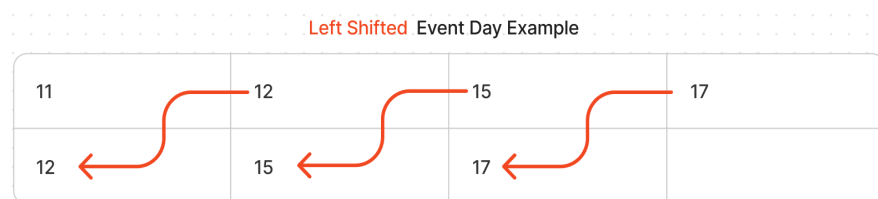
    # aux_eventName into eventsName array
    la $t1, aux_eventName
    la $t2, eventsName($t0)

    # loop to copy the aux_eventName into eventsName
```

Ademais, na remoção o evento a ser removido é escolhido através do seu índice no array, que é impresso junto aos dados de cada evento na função print.

```
Event number: 1
Event name: aniversário
Event day: 13
Event start time: 12.0
Event end time: 15.0
```

A partir do índice indicado, os elementos das posições subsequentes são deslocados uma posição para a esquerda, repetindo-se esse processo até alcançar a última posição dos arrays.



Após deslocados, o contador de eventos é decrementado.

```
Exemplo da função Remove

# decrease eventCounter
lw $t1, eventCounter
subi $t1, $t1, 1
sw $t1, eventCounter
```

Já a função print funciona de uma maneira bastante simples, ela simplesmente cria um loop que a cada interação, passa pelo índice dos arrays imprimindo cada dado dos eventos (o índice, nome, data, e hora de início e fim), quando o índice é igual ao 'eventCounter' todos os eventos foram impressos.

```
Exemplo da função Print

# function of print all events
print:
# using auxCounter set to 1
li $t0, 1
lw $t1, eventCounter
beq $t0, $t1, errorNoEventToPrint

# loop to print all information about events
loop_print:

# if auxCounter is equal to eventCounter, we have printed all events
bge $t0, $t1, exit_print

# Print text of event number
li $v0, 4
la $a0, print_eventNumber
syscall

# print t0
li $v0, 1
move $a0, $t0
syscall
```

Para editar um evento usando nossa função 'edit', procedemos da mesma maneira que selecionamos o elemento a ser removido na função 'remove': através do índice.

```
Exemplo da função Edit

# function to edit an event
edit:
    # if eventCounter is 1, we need to print the errorEdit message
    lw $t0, eventCounter
    beq $t0, 1, errorNoEvent

    # print the event number question
    li $v0, 4
    la $a0, edit_eventNumber
    syscall

    # store the event number in the eventNumber variable
    li $v0, 5
    syscall
    sw $v0, eventNumber
```

A ‘`edit`’ recebe 1, e os dados do dia a ser editado são armazenados em variáveis auxiliares.

```
Exemplo da função Edit

# set edit_flag to 1
lw $t0, edit_flag
addi $t0, $t0, 1
sw $t0, edit_flag

# load the eventday in s0
lw $t0, eventNumber
mul $t0, $t0, 4 #MAX_LENGTH_DAY

lw $s0, eventsDay($t0)
```

Quando armazenados, o dia é mandado para a função ‘`remove`’ com a flag de edição armazenando um valor verdadeiro

```
Exemplo da função Edit

# jumps to remove function with edit_flag set to 1
j edit_remover

edit_insert:
```

O item é removido, e é checado a flag de edição no fim da remoção, se ela for igual a 1 o programa volta para a função ‘`edit`’, no ‘`edit_insert`’.

```
Exemplo da função Edit

# if editor_flag is equal to 1, we need to jump to edit_insert, else we need to
# print the operationSucceeded message
lw $t0, editor_flag
beq $t0, $zero removeSuccess
j edit_insert
```

Em ‘edit\_insert’, o programa pergunta quais dados o usuário deseja alterar, sendo que os dados antigos continuam salvos nas variáveis auxiliares, e se o dado for alterado, ele é sobrescrito na respectiva variável auxiliar.

```
Exemplo da função Edit

# print keep event question
li $v0, 4
la $a0, edit_keepThis
syscall

# reading the option
li $v0, 5
syscall
move $t3, $v0

# if $t3 is equal zero, jump to edit_jumpName
beq $t3, $zero, edit_jumpName

# print what's new question
li $v0, 4
la $a0, edit_whatsNew
syscall

# reading the event name
li $v0, 8
la $a0, aux_eventName
li $a1, 50 #MAX_LENGTH_EVENT_NAME
syscall
https://ray.so/#
edit_jumpName:
```

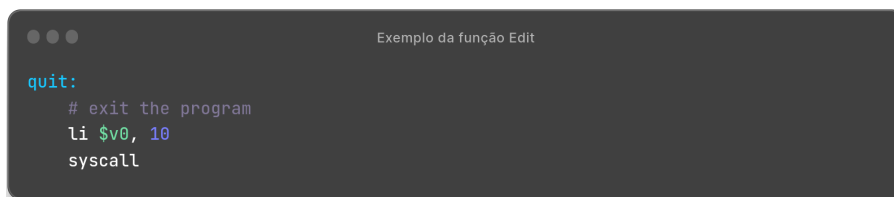
A flag de edição é resetada, e com todos os dados salvos nas variáveis auxiliares, o programa reinsere o evento utilizando o compareDay e sortArray, na mesma lógica da inserção, que já ocorre em ordem, assim toda vez que um evento editado, mesmo que a data e horas forem alterados, o array de eventos continuará ordenado.



```
# reset editor flag
lw $t9, editor_flag
subi $t9, $t9, 1
sw $t9, editor_flag

j compareDay
```

Por fim, a função quit, que chama a syscall com  $\$v0 = 10$  para fechar o programa.



```
quit:
# exit the program
li $v0, 10
syscall
```

## 3. Processamento da informação

### 3.1. Dados de entrada

Ao receber os dados requisitados, a etapa inicial consiste no armazenamento desses dados em variáveis auxiliares. Essa abordagem permite a realização de verificações essenciais antes do momento de inserção efetiva na memória, momento em que o contador de eventos é incrementado de fato.

Cada categoria de informação referente aos eventos é alocada em seu respectivo vetor, permitindo a associação entre os eventos através dos índices correspondentes. Essa correlação é estabelecida mediante a utilização da função secundária "compareDay", a qual se encarrega de resolver possíveis sobreposições entre os eventos registrados. Para alcançar a ordenação dos eventos por dia, crucial para uma visualização otimizada na função de impressão, a função "compareDay" aciona a função "sortArray". Esse procedimento contribui significativamente para um gerenciamento mais eficiente e estruturado dos eventos, resultando em uma agenda organizada e de fácil compreensão. Isso será muito útil na implementação descrita no tópico [3.3](#).

### 3.2. Processamento

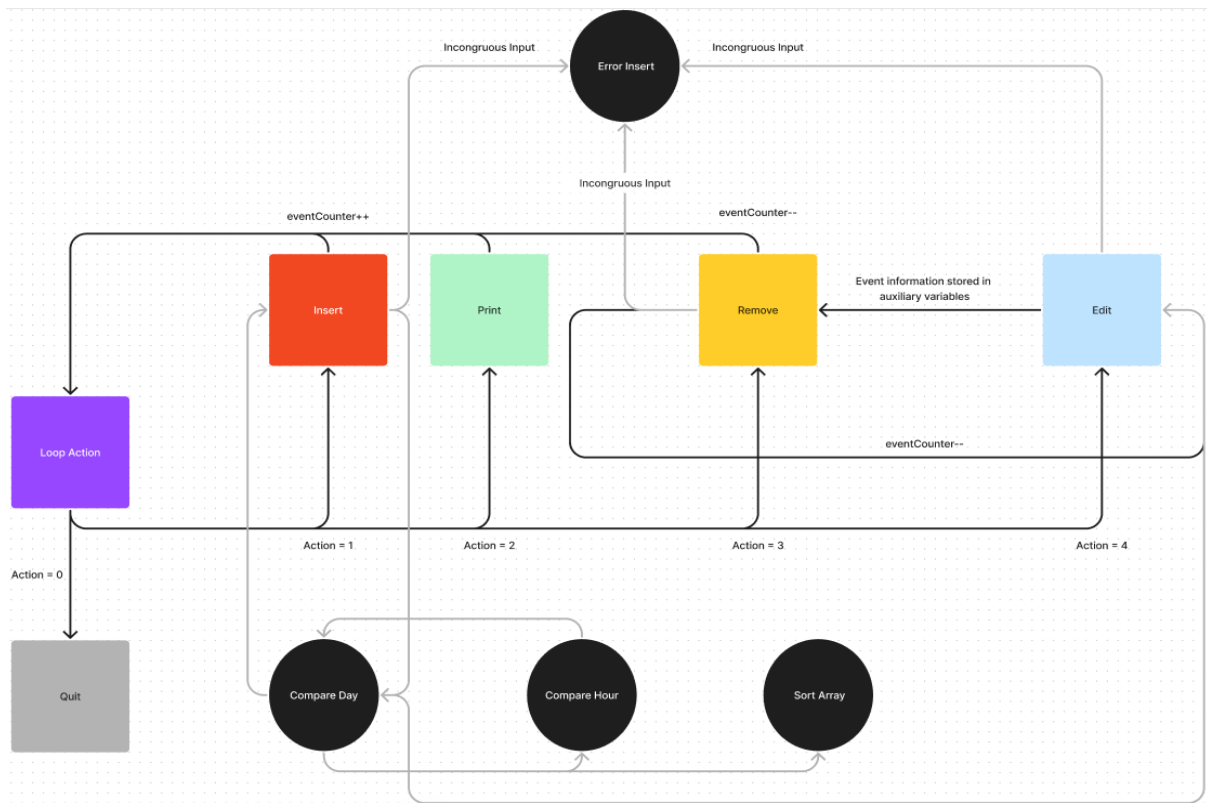
Para realizar a remoção de eventos, é necessário capturar de maneira precisa o evento que se deseja remover. Para alcançar isso, decidimos utilizar o índice do array como identificador único de cada evento. Dessa forma, cada evento é associado a um identificador baseado na sua posição no array. Para efetuar a remoção, é necessário deslocar uma posição para a esquerda todos os elementos à direita do evento que será removido no array. Após esse deslocamento, é feita a redução do contador de eventos, atualizando-o adequadamente.

No caso da função de edição de eventos, o processo foi um pouco mais complexo. Inicialmente, foi crucial capturar todos os dados do evento que se desejava editar, possibilitando ao usuário a opção de manter determinadas informações inalteradas durante a edição. Para realizar a edição, primeiramente acionamos a função de remoção para excluir o evento que será editado. Posteriormente, inserimos novamente o evento editado, proporcionando ao usuário a liberdade de manter ou alterar os dados conforme sua preferência. Essa ação de adicionar novamente o evento foi fundamental para reordenar todos os eventos, caso o usuário tenha alterado informações que impactassem a ordem correta dos eventos armazenados. Isso se dá ao fato de que a ordenação ocorre na inserção de um novo evento, então a forma mais prática de editar um evento foi removendo ele, guardando dados que se manteriam inalterados e chamando a função de inserção com uma flag de edição.

### 3.3. Dados de saída

Por esse motivo, na função print, há uma iteração por todos os elementos de cada array por meio de uma variável auxiliar. Enquanto imprime as informações dos compromissos, essa variável é incrementada. O processo continua até que o contador auxiliar alcance o mesmo valor do contador de eventos, momento em que a iteração é encerrada. Esse mecanismo assegura que todos os compromissos sejam exibidos corretamente e que a impressão seja concluída de forma precisa e abrangente.

Finalizando a descrição das funções principais, temos a função quit, que desempenha exclusivamente o papel de encerrar o programa.



## 4. Desafios

O projeto apresentou vários desafios durante sua execução, sendo os principais:

- 4.1. Captação das entradas do usuário: se mostrou difícil a captação e manuseio de diferentes tipos de dados(int, strings, float) inicialmente. Essa dificuldade causou confusão na inserção dos eventos, por exemplo, quando tratando cada informação em seu devido array e diferentes offsets;
- 4.2. Debug do código: quando testando e o programa apresentava erros, era difícil localizar a origem do problema, e quando localizado por conta de assembly ser um tanto ilegível no sentido de que não é fácil compreender rapidamente o que uma linha ou um conjunto de linhas está fazendo exatamente no programa;

- 4.3. Organizar a agenda em ordem cronológica: foi planejada e executada uma inserção em ordem, pensando que seria o mais simples de implementar. Esse plano apresentou prós e contras, além da dificuldade inicial de manusear quatro arrays diferentes de tipos de dados distintos, considerando de que forma iriam ser tratadas as variáveis auxiliares para cada dado e os tipos de registradores que cada dado precisa, a função `sortArray` ficou extremamente vinculada a função `CompareDay`, que por sua vez era ligada a inserção. Nesse processo era necessário tratar a ordem cronológica dos dias e dentro do mesmo dia checar colisão de horários e ordenar por horários os diferentes arrays. Considerando a complexidade conjunta dessas funções, esse se mostrou o maior desafio do projeto.

## 5. Conclusão

A proposta do `AppointmentAgenda` visa oferecer aos usuários uma ferramenta completa e intuitiva para o gerenciamento de compromissos, demonstrando-se capaz de realizar o agendamento, modificação e remoção de eventos de forma eficiente. A ênfase na prevenção de conflitos por meio de um sistema inteligente de detecção de sobreposições de horários garantiu uma agenda organizada e livre de conflitos, proporcionando aos usuários uma experiência de agendamento tranquila e livre de complicações.

As funcionalidades implementadas, como a organização cronológica dos eventos, a detecção de inputs inválidos e os processos de remoção e edição, foram fundamentais para a precisão e integridade dos dados armazenados. A estruturação do programa com base em arrays e índices permitiu uma gestão eficaz dos eventos, garantindo a ordenação adequada e facilitando a visualização dos compromissos.

O desenvolvimento desse projeto foi um desafio significativo para os integrantes da equipe, sendo a primeira vez que trabalharam com uma linguagem de baixo nível como o `Assembly MIPS`. Lidar com uma problemática complexa, envolvendo considerável manipulação de memória, proporcionou não apenas aprendizado, mas também insights valiosos sobre o funcionamento interno dos sistemas computacionais.



## 6. Código Fonte

Acesse o projeto final, incluindo as instruções de uso e instalação, clicando neste [link](#).