**Relatório EP 2**

**EACH-USP**

**ORGANIZAÇÃO DE COMPUTADORES DIGITAIS**

**AUTOR:**

**Gabriel faria de Oliveira 10403210**

**São Paulo**

**2020**

**Objetivo:** Implementar um programa que compila algumas instruções MIPS, que serão listadas abaixo, e interpreta os comando em linguagem de máquina, mostrando as mudanças nos componentes da arquitetura, cuja implementação também faz parte do EP, de uma forma interativa.

**Implementação:**

O EP foi implementado usando a IDE NetBeans para codificar (com um conjunto de 13 classes que contêm as funções) e montar a interface (a partir de um Jframe e componentes dentro dele). Ela permite que um código em MIPS, que pode ser escrito nela mesma ou através de um arquivo TXT (passando o caminho dele para o interface), seja compilado e suas mudanças na arquitetura possam ser observadas de uma forma interativa. As instruções suportadas pela interface são as seguintes: add (entre registradores ou um registrador e um imediato de 19 ou 32 bits), sub (entre registradores ou um registrador e um imediato de 19 ou 32 bits), slt (entre registradores ou um registrador e um imediato de 19 ou 32 bits), li (de um valor de 19 ou 32 bits), lw (de um algo declarado no .data ou de um endereço em algum registrador com algum deslocamento), sw (em um algo declarado no .data ou em um endereço em algum registrador com algum deslocamento), move (entre registradores), beq (entre registradores ou um registrador e um imediato de 19 ou 32 bits), bne (entre registradores ou um registrador e um imediato de 19 ou 32 bits) e j (para alguma linha do micro programa na forma de um imediato de 19 bits). Essas instruções, depois de compiladas, vão para o painel EXECUTAR, no qual é possível ver como elas foram quebradas, enquanto no MICROPROGRAMA é possível ver cada ciclo delas no próprio microprograma. Também é possível ver os registradores e os endereços de memória alterando seus valores conforme o decorrer do programa e, assim, a interface atende aos objetivos do EP.

**O compilador:**

Todas as funções a respeito dele estão na classe COMPILADOR e ele é responsável por checar a sintaxe e transformar o código de assembly para binário. Para checar se o código segue o padrão MIPS, a classe utiliza o método “String.matches(String)”, ou seja, ela verifica se cada linha dele bate com algum padrão do conjunto que está na classe regexMIPS. Se a sintaxe (que tem alguns exemplos no apêndice) estiver correta, ela começa a compilar seguindo o modelo abaixo:

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente

A instruções de dividem da seguinte forma: add, sub e slt entre registradores no tipo R; lui, ori, li, beq, bne, sw, lw e as instruções já mencionadas (porém trabalhando com imediatos e talvez sendo necessário as quebrar em lui e ori) no tipo I; e j no tipo J. No tipo R, o REG1 e o REG3 são fontes de dados enquanto o REG2 é o destino. Já no tipo I, o IMEDIATO é um valor de 19 BITS (podendo ser uma endereço ou um valor), o REG2 sempre é uma fonte da dado e o REG1 pode ser um destino (no caso do sw, li, add, sub, slt e o move) ou também uma fonte de dado, como nas demais operações. Por fim, no tipo J há somente um endereço de 19 BITS de pulo (cujo o tamanho em bits será explicado mais a frente) e só é usado pela instrução j.

Mais especificamente, o compilador divide tudo em códigos específicos para transformar em binário. Para a maioria dos registradores não há códigos pois eles só são ativados no microprograma, mas para aqueles que precisam eles são os seguintes:

* nenhum: cujo nome na microinstrução fica “nan” e o código é “0000”.
* zero: cujo nome na microinstrução fica “$zero” e o código é “0001”.
* T1: cujo nome na microinstrução fica “$t1” e o código é “0010”.
* T2: cujo nome na microinstrução fica “$t2” e o código é “0011”.
* T3: cujo nome na microinstrução fica “$t3” e o código é “0100”.
* T4: cujo nome na microinstrução fica “$t4” e o código é “0101”.
* S1: cujo nome na microinstrução fica “$s1” e o código é “0110”.
* S2: cujo nome na microinstrução fica “$s2” e o código é “0111”.
* S3: cujo nome na microinstrução fica “$s3” e o código é “1000”.
* S4: cujo nome na microinstrução fica “$s4” e o código é “1001”.
* SP: cujo nome na microinstrução fica “$sp” e o código é “1010”.
* FP: cujo nome na microinstrução fica “$fp” e o código é “1011”.

Também existem o X e o buffer da ULA, cujos nomes na microinstrução ficam, respectivamente, “ula-X” e “ula-BFR” e os códigos são, respectivamente, “1100” e “1101”. Eles são usados somente pela instrução lui e ori quando uma outra instrução precisa usar um valor imediato que ocupa mais que 19 bits.

Também existem os códigos de operação, que são os seguintes:

* noop: representa a falta de operação e o código é “00000”.
* addr: representa uma soma entre dois registradores e o código é “11111”.
* addi19: representa uma soma entre um registrador e um imediato de 19 bits e o código é “01111”.
* addi32: representa uma soma entre um registrador e um imediato de 32 bits e o código é “10111”.
* subr: representa uma subtração entre dois registradores e o código é “00111”.
* subi19: representa uma subtração entre um registrador e um imediato de 19 bits e o código é “11011”.
* subi32: representa uma subtração entre um registrador e um imediato de 32 bits e o código é “01011”.
* move: representa uma transferência entre registradores e o código é “10011”.
* li19: representa carregar um dado de 19 bits em um registrador e o código é “00011”.
* li32: representa carregar de um dado de 32 bits em um registrador e o código é “11101”.
* lwr: representa um uma leitura na memória de um endereço em um registrador somado a um deslocamento e o código é “01101”.
* lwl: representa uma leitura de algo declarado no .data e o código é “10101”.
* swr: representa uma escrita na memória em um endereço em um registrador somado a um deslocamento e o código é “00101”.
* swl: representa uma escrita em algo declarado no .data e o código é “11001”.
* sltr: representa colocar o valor 1 em um registrador se o primeiro dos outros for menor que o segundo dos outros e o código é “01001”.
* slti19: representa colocar o valor 1 em um registrador se o outro for menor que um imediato de 19 bits e o código é “10001”.
* slti32: representa colocar o valor 1 em um registrador se o outro for menor que um imediato de 32 bits e o código é “00001”.
* beqr: representa pular para uma linha se os registradores tiverem os mesmo valores e o código é “11110”.
* beqi: representa pular para uma linha se o registrador tiver o mesmo valor que um imediato (de 19 ou 32 bits) e o código é “01110”.
* bner: representa pular para uma linha se os registradores não tiverem os mesmo valores e o código é “10110”.
* bnei: representa pular para uma linha se o registrador não tiver o mesmo valor que um imediato (de 19 ou 32 bits) e o código é “00110”.
* j: representa pular para uma linha incondicionalmente e o código é “11010”.
* lui: representa carregar os primeiros 13 bits em um registrador e o código é “01010”.
* ori: representa carregar os últimos 19 bits em um registrador e o código é “10010”

Vale lembrar que o li32 nunca é usado, pois ele é sempre é trocado por um lui e um ori, e a instruções de branch condicional sempre carregam um imediato usando um lui e um ori. Também é importante dizer que o branch só pode ser feito para uma linha que cabe em 19 bits.

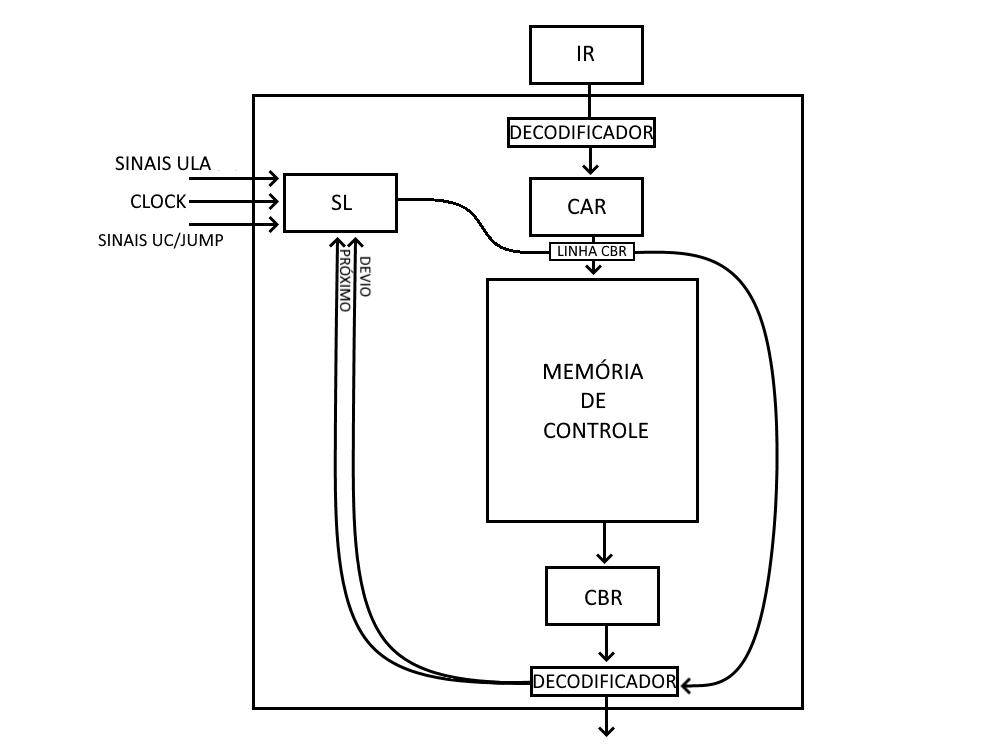
**A arquitetura:**

**Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente**

Ela é composta pelo clock (CLOCK), a unidade de controle (UC), a unidade logica aritmética (ULA), a memória (MEMÓRIA), 17 registradores, 3 buffers (1 na ULA e 2 na MEMÓRIA). Cada registradores e buffer é tem sua devida porta aberta seguindo os comandos da UC, que, além do própria porta, dá um sinal de 2 bits a mais (sendo 00 para receber ou mandar um dado de 32 bits, 10 para os primeiros 13 bits, 01 para os 19 últimos bits e 11 para um imediato ou endereço). Já a ULA, que é ativada em todo ciclo, recebe um sinal de 3 bits da UC, sendo o primeiro bit para fazer alguma operação naquele ciclo (0 para não e 1 para sim), o segundo para usar o bit de sinal ou não (0 para não e 1 para sim) e o terceiro para saber a operação (0 para soma e 1 para subtração). A MEMÓRIA também é ativada todo ciclo e recebe um sinal de 2 bits, sendo o primeiro para fazer alguma operação naquele ciclo (0 para não e 1 para sim) e o segundo para fazer saber a operação (0 para leitura e 1 para escrita).

A UC também tem sinais de controle e uma estrutura interna que é a seguinte:



Ela tem um sinal de controle que vem do próprio microprograma, com o intuito de saber quando usar a linha 0, a no SL ou no CAR para pegar o devida linha no microprograma. Esse sinal tem 2 bits (sendo 00 para esperar o próximo clock e usar o linha 0, 11 para também esperar o próximo clock e mas usar o CAR, 01 usar o SL naquele ciclo ou 10 para usar o CAR naquele ciclo). Quando o devida linha é escolhida e colocada no CBR, alterações são feitas, se baseando na atual linha que foi usada para saber em qual microinstrução a UC está e usando o IR para saber quais portas abrir se necessário. Após isso, são mandados os sinais de controle, as unidade são ativadas pela ARQUITETURA e o SL recebe a linha imediatamente depois e a caso ocorra desvio, o que é determinado através do sinal de jump de 2 bits da UC (00 para não pular, 10 para pular condicionalmente e 11 para pular incondicionalmente) que também vem o próprio micropograma. Além disso, há o sinal de 2 bits do barramento (sendo o primeiro bit para colocar o dado na palavra de controle no barramento interno e o segundo para colocar ele no barramento externo), e um sinal especial de 1 bit do IR (sendo 0 para um imediato e 1 para um endereço de jump), que é utilizado quando ele vai mandar algum dado para o barramento.

Por fim, há o funcionamento da ULA e da MEMÓRIA que são, como já dito, são ativados em todos os ciclos, mas só funcionam quando ativados pela UC. O primeiro vai pegar o binário de 32 bits no X e no buffer, somar ou subtrair usando o bit de sinal ou não e colocar no AC mandando um sinal de 3 bits para a UC, sendo o primeiro para dizer se deu overflow (0) ou não (1), o segundo para dizer se o sinal deu positivo (0) ou negativo (1) e o terceiro para dizer de seu zero (1) ou não (0). Já o segundo, se ele tiver que funcionar no ciclo atual, vai pegar o endereço de 19 bits no buffer, ler da memória e colocar no buffer de saída ou pegar o endereço de 19 bits no buffer auxiliar de endereço e escrever na memória o dado de 32 bits no buffer.

Tudo isso é regido por um microprograma (cujo cabeçalho está no fim do relatório) e CLOCK que tem um pico a cada 7 ciclos, o que dá espaço para a operação mais demorada, a sw, completar todas as microinstruções e, assim, gerar um execução de todas as instruções programadas.

**Ressalvas:**

Algumas pontos vão ser destacados nesse tópico e eles são relevantes para entender o funcionamento dele e suas restrições. Primeiramente, é necessário saber que os endereços podem ocupar no máximo 19 bits, pois os pulos não são feitos usando labels e, na verdade, o número da linha alvo e a memória está programada para usar endereços virtuais de 19 bits, multiplicando eles por 32 para chegar no endereço físico. Ela também tem o comportamento de primeiro receber o endereço e depois o dado caso uma escrita vai ocorrer. Por fim, é importante destacar que a arquitetura assume que cada microinstrução dura um ciclo de clock e primeiro os comandos serão mandados e depois as unidades serão ativadas.

Mais especificamente existem alguns detalhes de implementação que, por falta de literatura e tempo, não foram finalizados. Alguns casos disso são os seguintes:

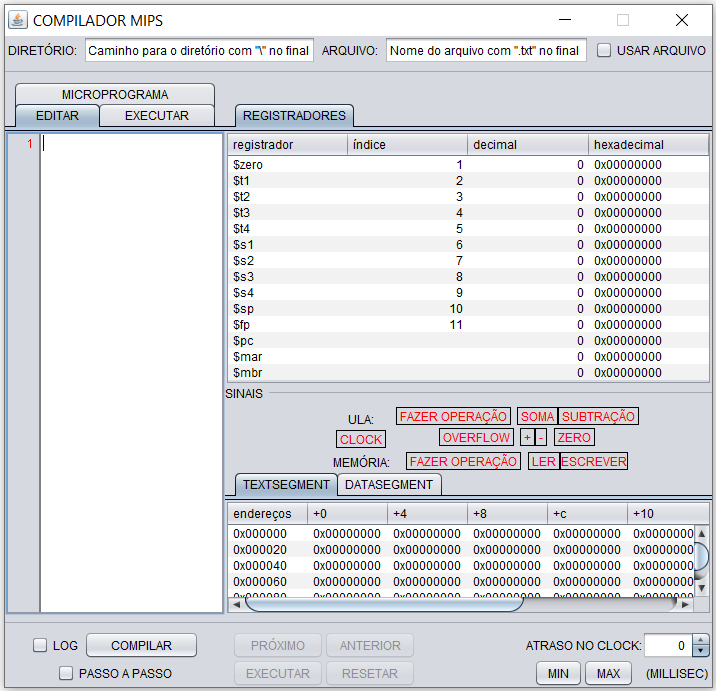
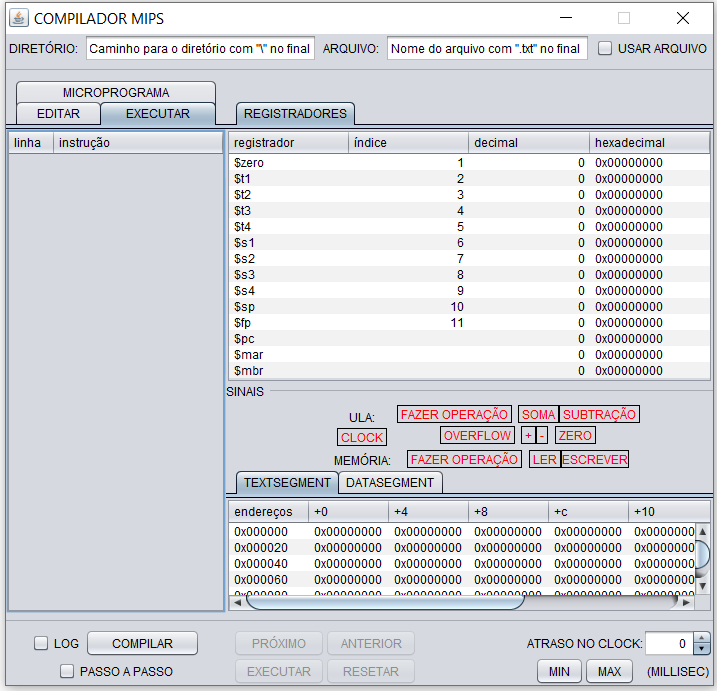
* a única forma de declarar dados é a .word
* a instrução slt somente coloca o valor 1, caso as condições sejam aceitas, e não coloca 0 caso contrário.
* O atraso que é possível colocar no clock não é exatamente na unidade milissegundos e, assim, a noção de tempo no atraso não está exatamente correta.
* A memória não é resetada, por decisão de projeto, quando o código é recompilado, o que pode gerar alguma dúvida durante a execução do código que foi alterado e recompilado.
* O pulo não é por label e, na verdade, é por linha, o que exige atenção quando mudar o código, pois, caso ele seja alterado, uma instrução que é alvo de algum jump corre o risco de trocar de linha.
* Para conseguir pular para uma linha, é necessário que ela tenha uma instrução nela, ou seja, que ele tenha uma das operações suportadas pela interface, se não o código não será compilado.
* Para ter acesso a um arranjo declarado no .data sem usar a label dele, é necessário saber ou ter o valor do endereço carregado em algum registrador de antemão, para usar o lw ou sw com um valor de registrador somado a um deslocamento.
* Ao usar o botão resetar, a memória não é resetada e isso torna necessário recompilar caso alguma mudança no memória tenha sido feita.

**Como compilar:**

Para compilar o EP há duas possibilidades que são as seguintes: abrir o projeto usando NetBeans e compilar através dele, uma vez que a interface MIPS é um JFrame criado na IDE e todas as classes estão projetadas para funcionar conjuntamente. Para abrir o EP no NetBeans é necessário tem a IDE instalada, ir em “File”, clicar em “Open Project”, ir até a pasta em que foi descompactado o zip e abrir a pasta calculadora, que é o projeto.

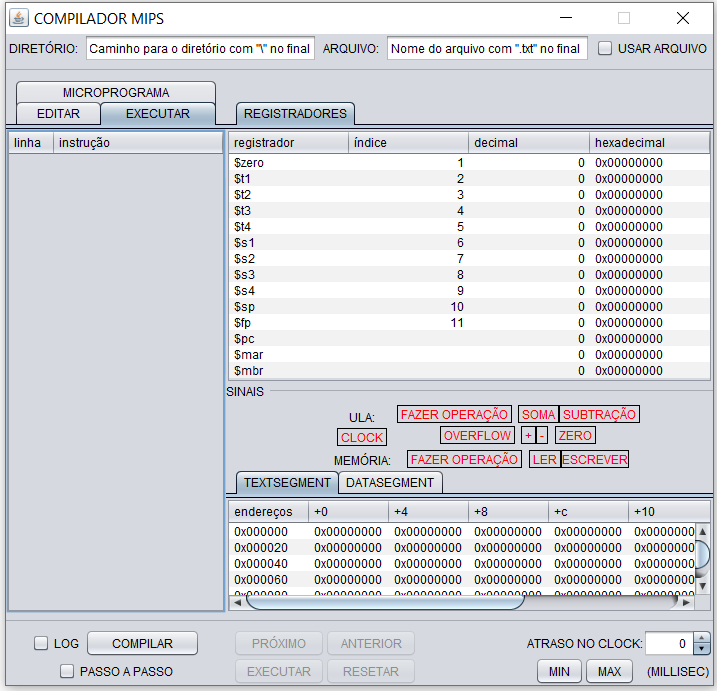
**Como executar:**

Clicando no arquivo “MIPS.jar”, a interface será aberta e os sequintes componentes estarão nela:

****Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Essas são as telas entre as quais é possivel naveger clicando nas diferente abas em que, respectivamente, é possivel editar o código, acomponhar as instruções geradas pelo compilador e o próprio microprograma. Na primeira e no segunda as mudanças vicam visiveis quando uma diferente linha é selecionada (fica destacada em azul).



Nesses campos fica o caminho para algum arquivo TXT que pode ser utilizado no lugar da area de código da interface. Para isso, é necessário inserir o caminho corretamente (preenchendo o diretório com um “\” no final e colocando o nome do arquivo mais “.txt”) e a checkbox “USAR ARQUIVO” deve estar selecionada.

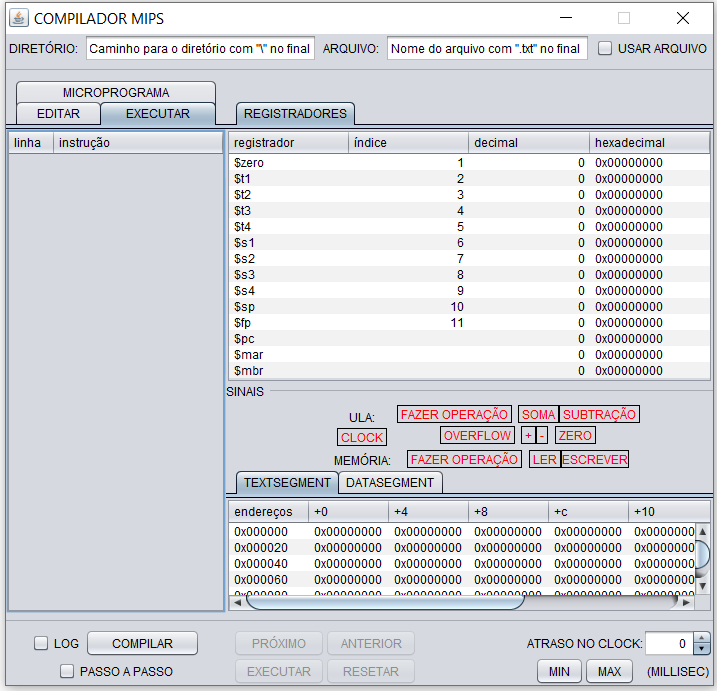
Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Aqui é onde código é compilado, clicando no botão compilar, e a execução pode ser acompanhada, clicando nos botões “PROXIMO”, “EXECUTAR” e “RESETAR”. Também é possível ter um log da compilação e da sintaxe, clicando na checkbox “LOG” antes de compilar, ou escolher entre uma execução passo a passo, clicando o “PASSO A PASSO” ante de compilar ou ter uma execução automática. Ela pode ser feita na maior velocidade possível ou mais lentamente usando o “ATRASO NO CLOCK”.



Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamenteTela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Nessas tabelas é possivel observar as mudanças nos valores nos endereços de memória e nos sinais de controle essenciais para a coordenação dos componentes da arquitetura. Quando algum enredeço de memória for alterado, é possivel ver ele mudando de valor em hexadecimal, enquanto os sinais sinais ficam vermelho quando foram ‘0’ e verdes quando forem ‘1’.

**Testes:**

Os teste, como esse EP se trata de um compilador/interpretador/interface para partes das funcionalidades da linguagem MIPS, se basearam códigos de teste, como loops, pegar e colocar um dado na memória e mexer com os registradores (Exemplos de testes estão no apêndice). Alguns testes de código são os seguintes:

**Apêndice:**

* Cabeçalho microprograma:

|  |  |
| --- | --- |
| Linha 0  Ciclo de Busca | A porta de saída de dados do PC é aberta  A porta de entrada de dados do MAR é aberta  A porta de entrada do X é aberta |
| Linha 1 | A porta de saída do MAR é aberta  A porta de entrado do buffer da ULA é aberta  A porta de entrada do buffer da MEMORIA é aberta  O sinal 100 vai para a ULA (fazer soma sem usar bit de sinal)  O sinal 10 vai para a MEMÓRIA (fazer leitura)  O sinal 1 vai para o barramento interno (dado da PL no barramento interno) |
| Linha 2 | A porta de saída do buffer da MEMORIA é aberta  A porta de saída do AC é aberta  A porta de entrada (dado externo) do MBR é aberta  A porta de entrada do PC é aberta |
| Linha 3 | A porta de saída (dado interno) do MBR é aberta  A porta de entrada do IR é aberta  O sinal 11 vai para a UC (esperar o próximo pico do clock e usar o CAR) |
| Linha 4  addr | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 5 | A porta de entrada do buffer da ULA é aberta  Um sinal de pulo incondicional (11) vai para a UC  O sinal 110 vai para a ULA (fazer soma usando bit de sinal)  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 6  addi19 | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 7 | A porta de saída de dado imediato do IR é aberto  A porta de entrada do buffer da ULA é aberta  O sinal 11 vai para os registradores (vão receber um dado imediato)  O sinal 110 vai para a ULA (fazer soma usando bit de sinal)  Um sinal de pulo incondicional (11) vai para a UC |
| Linha 8  addi32 | A porta de entrada do X é aberta  O sinal 110 vai para a ULA (fazer soma usando bit de sinal)  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 9 | A porta de saída do AC é aberta  (A UC faz alterações no CBR para abrir alguma porta)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 10  subr | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 11 | A porta de entrada do buffer da ULA é aberta  Um sinal de pulo incondicional (11) vai para a UC  O sinal 111 vai para a ULA (fazer subtração usando bit de sinal)  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 12  subi19 | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 13 | A porta de saída de dado imediato do IR é aberto  A porta de entrada do buffer da ULA é aberta  O sinal 11 vai para os registradores (vão receber um dado imediato)  O sinal 111 vai para a ULA (fazer subtração usando bit de sinal)  Um sinal de pulo incondicional (11) vai para a UC |
| Linha 14  subi32 | A porta de entrada do X é aberta  O sinal 111 vai para a ULA (fazer subtração usando bit de sinal)  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 15 | A porta de saída do AC é aberta  (A UC faz alterações no CBR para abrir alguma porta)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 16  move | (A UC faz alterações no CBR para abrir alguma porta)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 17  li19 | A porta de saída de dado imediato do IR é aberto  (A UC faz alterações no CBR para abrir alguma porta)  O sinal 11 vai para os registradores (vão receber um dado imediato)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 18  lwr | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 19 | A porta de saída de dado imediato do IR é aberta  A porta de entrada do buffer da ULA é aberta  O sinal 11 vai para os registradores (vão receber um dado imediato)  O sinal 110 vai para a ULA (fazer soma usando bit de sinal) |
| Linha 20 | A porta de saída do AC é aberta  A porta de entrada de dados do MAR é aberta  Um sinal de pulo incondicional (11) vai para a UC |
| Linha 21  lwl | A porta de entrada de dados do MAR é aberta  A porta de saída de dado endereço do IR é aberta  O sinal 11 vai para os registradores (vão receber um dado imediato) |
| Linha 22 | A porta de saída de dados do MAR é aberta  A porta de entrada do buffer da MEMORIA é aberta  O sinal 10 vai para a MEMÓRIA (fazer leitura) |
| Linha 23 | A porta de saída do buffer da MEMORIA é aberta  A porta de entrada (dado externo) do MBR é aberta |
| Linha 24 | A porta de saída (dado interno) do MBR é aberta  (A UC faz alterações no CBR para abrir alguma porta)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 25  swr | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 26 | A porta de saída de dado imediato do IR é aberta  A porta de entrada do buffer da ULA é aberta  O sinal 11 vai para os registradores (vão receber um dado imediato)  O sinal 110 vai para a ULA (fazer soma usando bit de sinal) |
| Linha 27 | A porta de saída do AC é aberta  A porta de entrada de dados do MAR é aberta  Um sinal de pulo incondicional (11) vai para a UC |
| Linha 28  swl | A porta de entrada de dados do MAR é aberta  A porta de saída de dado endereço do IR é aberta  O sinal 11 vai para os registradores (vão receber um dado imediato) |
| Linha 29 | A porta de saída de dados do MAR é aberta  A porta de entrada do buffer da MEMORIA é aberta |
| Linha 30 | A portas de entrada (dado interno) do MBR é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 31 | A portas de saída (dado externo) do MBR é aberta  A porta de entrada do buffer da MEMORIA é aberta  O sinal 11 vai para a MEMÓRIA (fazer escrita)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 32  sltr | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 33 | A porta de entrada do buffer da ULA é aberta  Um sinal de pulo incondicional vai para a UC  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 34  slti19 | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 35 | A porta de saída de dado imediato do IR é aberta  A porta de entrada do buffer da ULA é aberta  O sinal 11 vai para os registradores (vão receber um dado imediato)  Um sinal de pulo incondicional (11) vai para a UC |
| Linha 36  slti32 | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 37 | O sinal 111 vai para a ULA (fazer subtração usando bit de sinal)  Um sinal de pulo condicional vai para a UC |
| Linha 38 | O sinal 1 vai para o barramento interno (dado da PL no barramento interno)  (A UC faz alterações no CBR para abrir alguma porta)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 39  beqr | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 40 | A porta de entrada do buffer da ULA é aberta  Um sinal de pulo incondicional vai para a UC  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 41  beqi | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 42 | O sinal 111 vai para a ULA (fazer subtração usando bit de sinal)  Um sinal de pulo condicional vai para a UC |
| Linha 43 | A porta de saída de dado endereço do IR é aberta  A porta de entrada do PC é aberta  O sinal 11 vai para os registradores (vão receber um dado imediato)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 44  bner | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 45 | A porta de entrada do buffer da ULA é aberta  Um sinal de pulo incondicional vai para a UC  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 46  bnei | A porta de entrada do X é aberta  (A UC faz alterações no CBR para abrir alguma porta) |
| Linha 47 | O sinal 111 vai para a ULA (fazer subtração usando bit de sinal)  Um sinal de pulo condicional vai para a UC |
| Linha 48 | A porta de saída de dado endereço do IR é aberta  A porta de entrada do PC é aberta  O sinal 11 vai para os registradores (vão receber um dado imediato)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 49  lui | A porta de saída de dado imediato do IR é aberta  O sinal 10 vai para os registradores (vão receber os primeiros 13 bits)  (A UC faz alterações no CBR para abrir alguma porta)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 50  ori | A porta de saída de dado imediato do IR é aberta  O sinal 01 vai para os registradores (vão receber os últimos 19 bits)  (A UC faz alterações no CBR para abrir alguma porta)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 51  j | A porta de saída de dado endereço do IR é aberta  A porta de entrada do PC é aberta  O sinal 11 vai para os registradores (vão receber um dado imediato)  O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 52  Noop-  Instrução | O sinal 00 vai para a UC (esperar o próximo pico do clock e usar a linha 0) |
| Linha 53  Noop-  ciclo | O sinal 11 vai para a UC (esperar o próximo pico do clock e usar a CAR) |

Obs: A omissão de algum componente, como a ULA, em qualquer linha do cabeçalho quer dizer que esse componente não foi ativado ou sofreu alguma alteração; A omissão de algum sinal de controle para a UC quer dizer que ela vai usar a linha imediatamente depois no microprograma e não vai fazer desvio; A omissão de algum sinal de controle para os registradores quer dizer que eles vão receber um dado inteiro de 32 bits.

* Alguns teste da interface:

|  |  |  |
| --- | --- | --- |
| Teste 1 | Teste 2 | Teste 3 |
| .text  li $s1,5  li $t1,0  add $t1,$t1,1  li $t2,0  slt $t2,$t1,$s1  beq $t2,$zero,9  j 4  li $t1,0 | .data  arranjo: .word 4,3,1,5,2  .text  li $s1, 131072  li $s2, 131088  beq $s1, $s2, 21  move $t1, $s1  beq $t1, $s2, 19  lw $t2, ($t1)  lw $t3, 4($t1)  li $t4,0  slt $t4,$t2,$t3  bne $t4,$zero, 17  sw $t3, ($t1)  sw $t2, 4($t1)  add $t1,$t1,4  j 9  sub $s2, $s2, 4  j 7  li $s1, 0 | .text  li $s1, 131072  li $t1,0  li $t2,1  li $t3, 0  li $t4, 10  beq $t3,$t4,15  sw $t1, ($s1)  move $s2,$t2  add $t2,$t2,$t1  move $t1,$s2  add $t3,$t3,1  add $s1,$s1,4  j 7  li $s1,0 |
| Loop que soma 1 em $t1 até ele não seja mais menos que $s1, e no final coloca 0 no $t1 | loop que ordena um arranjo de 5 posições | Loop que escreve na memória os primeiros 10 números da sequência de fibonacci |

* Sintaxe correta:

Uma sintaxe correta exige que a instrução seja uma das suportadas pela interface e que siga as seguintes regras: haja um espaço entre o nome da instrução e o restante das partes e também pode haver um espaço ou não depois na virgula, mas nunca mais de um espaço. Alguns exemplos são os seguintes:

1. add $s1,$s2,$s3 ou add $s1, $s2, $s3 (correto)
2. slt $s1,$s2,5 ou slt $s1, $s2, 5 (correto)
3. move $s1,$s2 ou move $s1, $s2 (correto)
4. li 10 (correto)

Além disso é necessário que só tenha um .data e um .text, sendo que o primeiro ainda deve aparecer antes do segundo.

**Referências:**

Converter número Decimal para Binário, Octal e Hexadecimal. Clevert. Disponível em: <https://clevert.com.br/t/pt-br/base-convert>. Acesso em: 29 de abr. de 2020.

Text Component Line Number. HUSACCT. Disponível em: <https://github.com/HUSACCT/HUSACCT/blob/master/src/husacct/control/presentation/codeviewer/TextLineNumber.java>. Acessado em: -- de -- de 2020

Instrução IF Simples no MIPS. Elaine C. Gatto. Disponível em: <https://www.embarcados.com.br/instrucao-if-simples-no-mips>. Acesso em: 25 de mai. de 2020

MIPS Instruction Reference. AUTOR OU ORGANIZAÇÃO, 1998. Disponível em: <http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html>. Acesso em: 28 de abr. de 2020

Detalhamento da Compilação de Procedimentos no MIPS. Elaine C. Gatto. Disponível em: <https://www.embarcados.com.br/compilacao-de-procedimentos-no-mips>. Acesso em: 1 de jul. de 2020

Confusion between virtual and physical adresses. StackOverflow, 2018. Disponível em: <https://stackoverflow.com/questions/50486367/in-mips-tlb-confusion-between-virtual-and-physical-addresses>. Acesso em: 15 de jul. de 2020

Microprogramação. Siang Wun Song, 2001. Disponível em: <https://www.ime.usp.br/~song/mac412/oc/node46.html#:~:text=M%C3%A1quinas%20mais%20modernas%2C%20muitas%20vezes,as%20fam%C3%ADlias%20Intel%20e%20Motorola.>. Acesso em: 1 de jun. de 2020