UFPB-CI-DSC
Disciplina: Arquitetura de Computadores I
Prof. Augusto de Holanda B. M. Tavares
13 de Março de 2025

# Avaliação da 3ª unidade - Projeto

O objeto deste projeto é o estudo e a implementação de uma microarquitetura correspondente a uma versão modificada da Máquina Mic-1 apresentada em sala de aula. Isso se dá na forma de uma máquina virtual em linguaguem de alto nível, ou seja, é esperado que ao final do processo o grupo apresente e saiba explicar o funcionamento de um código capaz de receber algumas das instruções da ISA da IJVM e fornecer as saídas correspondentes, explicitando as modificações que cada passo da execução as instruções realizam na memória e nos registradores.

## Etapa 1

Considere a ULA apresentada na Figura 2 abaixo:

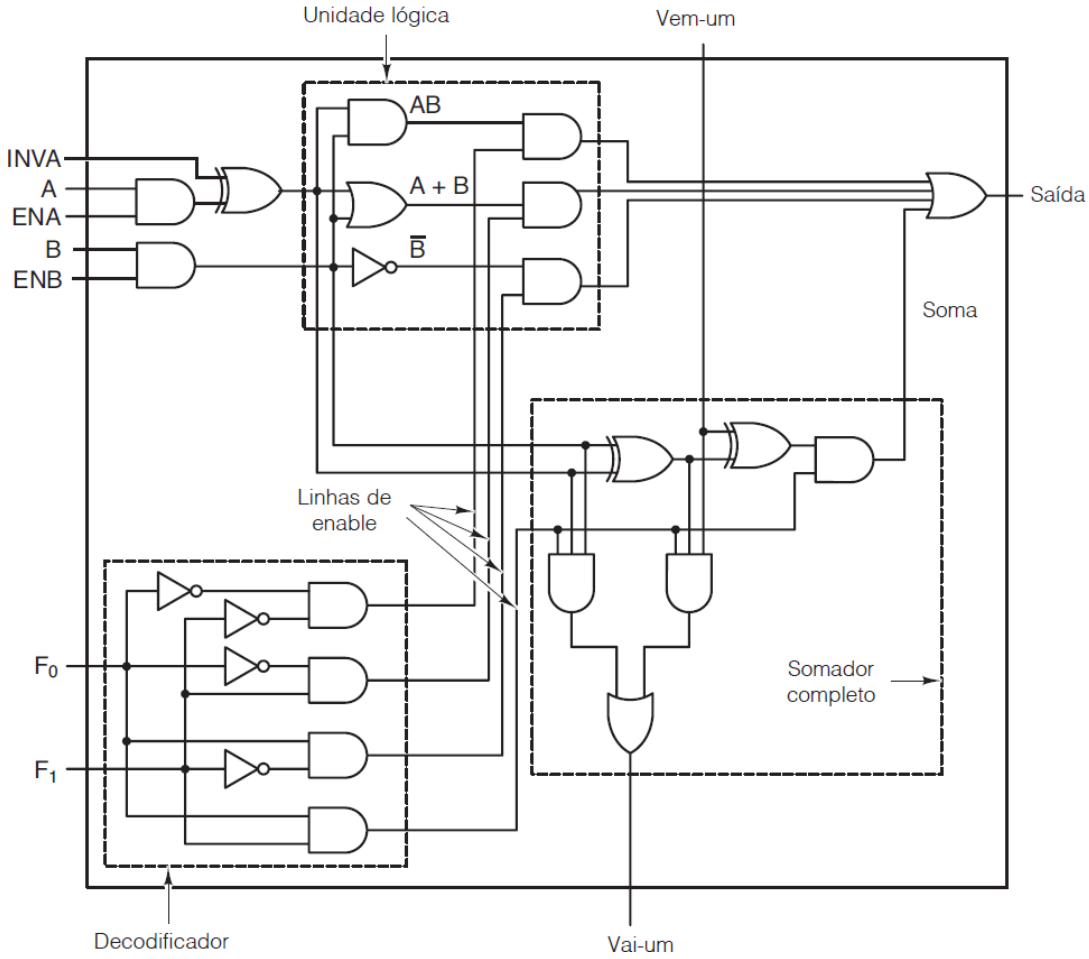


Figura 1: ULA da Mic-1

Esta ULA recebe instruções de uma combinação de sinais de entrada da forma:

| F0 | F1 | ENA | ENB | INVA | INC |
|---|---|---|---|---|---|
| $X_0$ | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ |

A partir da lógica dos circuitos que formam a ULA e dos sinais de entrada o sinal de saída $S$ e o valor do Vai-um são definidos. O sinal $INC$ se igual a 1 força o valor do *bit* de vem-um, somando um ao resultado. Por exemplo, a combinação de sinais abaixo implementa a soma aritmética dos sinais nas entradas A e B, ambas habilitadas.

| F0 | F1 | ENA | ENB | INVA | INC |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 |

Assim, se $A = 1$, $B = 1$ e Vem-um = 0, o sinal de saída e o vai-um serão:

$$S = A \oplus B = 1 \oplus 1 = 0, \text{ Vai-um} = 1 \tag{1}$$

A partir disto, construa um código em linguagem de alto nível que atenda as seguintes especificações.

- O código deve ser capaz de ler e executar uma sequência de instruções para a ULA apresentada acima a partir de um arquivo .txt.
- Cada linha do arquivo de instruções para a ULA conterá uma única palavra de 6 *bits* correspondente a uma instrução. A palavra estará organizada de acordo com o que foi descrito acima.
- Esta palavra deve ser armazenada em uma variável que representa o registrador de instrução (IR), e uma outra variável deve atuar como o contador de programa (PC), definindo que linha do programa está sendo executada.
- A cada linha do programa (arquivo .txt) executada, devem ser anotados em um arquivo de *log*: o valor de IR, o valor de PC, o valor de A, o valor de B, o valor de S e o valor do Vai-um.
- O arquivo *programa_etapa1.txt* fornece um conjunto de instruções para teste do código escrito para esta etapa do projeto, e o arquivo *saída_etapa1.txt* fornece um exemplo do log contendo as saídas esperadas.

## Etapa 2

O objetivo desta etapa é a implementação do caminho de dados da Mic-1 a partir da ULA da atividade anterior:

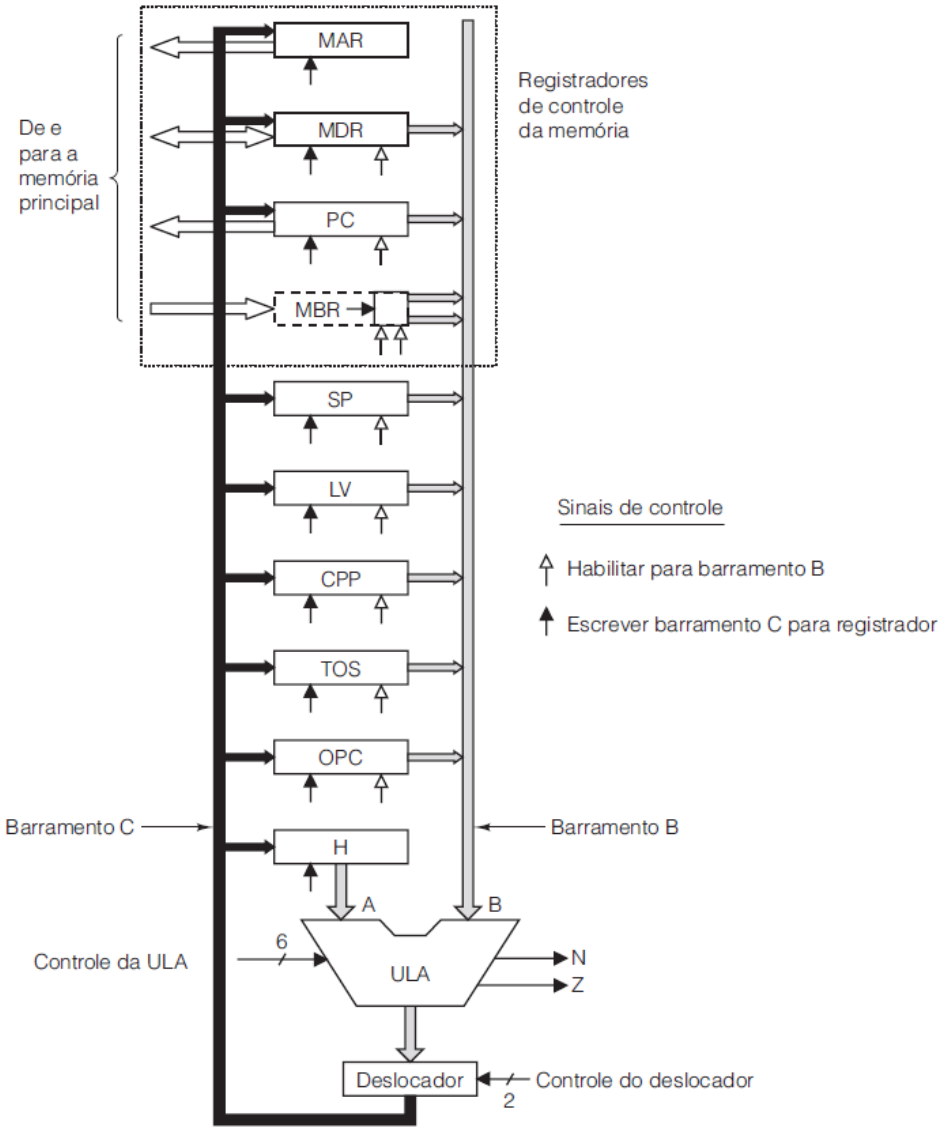


Figura 2: Caminho de dados da Mic-1

**Tarefa 1**

Modifique a ULA da atividade anterior para operar com uma palavra de 8 *bits* como os seus sinais de controle, sendo a forma da palavra dada por:

| SLL8 | SRA1 | F0 | F1 | ENA | ENB | INVA | INC |
|---|---|---|---|---|---|---|---|
| $X_0$ | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $X_7$ |

Há dois novos sinais de controle na ULA: SLL8 e SRA1.

- O sinal SLL8 ativa uma operação de deslocamento **lógico** para a esquerda que desloca a saída $S$ em 8 *bits*.

- O sinal SRA1 ativa uma operação de deslocamento **aritmético** para a direita em 1 *bit*.
- Adicione duas saídas a ULA: N e Z. Z assume nível lógico alto quando a saída é zero, e N assume nível alto quando a saída é negativa. Portanto, a ULA passaria a ter como saídas a saída deslocada $S_d$, o *bit* de Vai-um, N e Z.
- Use os arquivos *programa_etapa2_tarefa1.txt* e *saída_etapa2_tarefa1.txt* para verificar o funcionamento do seu código.

O processo do deslocador ocorre **depois** da saída da ULA ser calculada. Ou seja, em uma operação de soma, a saída $S = A + B$ seria deslocada em 1 *bit* se SRA1 estivesse alto. Ainda SLL8 e SRA1 nunca tem nível lógico alto ao mesmo tempo.

**Tarefa 2**

Implemente um conjunto de dez variáveis, correspondentes aos registradores da Mic-1:

- Os registradores de 32 *bits*: H, OPC, TOS, CPP, LV, SP, PC, MDR e MAR.
- O registrador de 8 *bits* MBR.

Em seguida, implemente as seguintes funções lógicas:

- Um decodificador de 4 *bits* que habilita um de 9 registradores a comandar o barramento B da entrada B da ULA.
- Um seletor de 9 *bits* que habilita um ou mais de 9 registradores acima a serem escritos com o valor na saída da ULA.

Para o caso do decodificador de 4 *bits*, deve ser seguida a convenção abaixo para a relação entre os registradores e a saída habilitada no decodificador:

| Registrador | OPC | TOS | CPP | LV | SP | MBRU | MBR | PC | MDR |
|---|---|---|---|---|---|---|---|---|---|
| Saída habilitada | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Por exemplo, se a entrada do decodificador é $0010_2 = 2_{10}$, o registrador MBR, de número 2, será habilitado para comandar o barramento B. Note que MBR é um registrador de 8 *bits*, e que há duas possibilidades dele comandar o barramento B: MBR e MBRU. Quando MBR é selecionado, o *bit* mais alto de MBR, que é o bit de sinal, deve ser utilizado para preencher a palavra de 8 *bits* até que está tenha os 32 *bits* necessários. Quando MBRU é selecionado, a palavra deve ser preenchida até 32 *bits* utilizando zeros. Para os demais registradores o valor armazenado será de 32 *bits* naturalmente, de modo que basta passar o valor armazenado.

Para o caso do seletor, a convenção é:

| Registrador | H | OPC | TOS | CPP | LV | SP | PC | MDR | MAR |
|---|---|---|---|---|---|---|---|---|---|
| *Bit* | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Por exemplo, se a entrada do seletor for 100001000, os registradores H (*bit* 8) e SP (*bit* 3) estarão habilitados para escrita no barramento C.

A lógica descrita visa implementar a seguinte característica do caminho de dados: a entrada B da ULA terá o valor do registrador habilitado pelo decodificador de 4 *bits*, enquanto a saída da ULA irá sobrescrever o valor dos registradores habilitados pelo seletor de 9 *bits*. A ULA deve ser conectada aos registradores acima da seguinte maneira:

- A entrada A da ULA deve ter o valor armazenado no registrador H.
- A entrada B da ULA deve ter o valor armazenado no registrador que está habilitado para comandar o barramento B.
- A saída deslocada $S_d$ deve sobrescrever o valor em todos os registradores habilitados para escrita no barramento C. Isto deve ocorrer apenas **depois** que a ULA operou os valores nas entradas A e B da instrução atual:

A partir deste ponto, o seu código deve ser capaz de receber palavras de 21 *bits* como instruções, com o seguinte arranjo:

| Controle da ULA | Controle do barramento C | Controle do barramento B |
|---|---|---|
| 8 *bits* | 9 *bits* | 4 *bits* |

Considere, por exemplo, a instrução abaixo:

$$001101001010000000000 \tag{2}$$

Os bits em azul definem que o registrador $MDR$ está comandando o barramento B. Logo, a entrada $B$ da ULA será $B = MDR$. Os *bits* em vermelho definem que a ULA executará uma operação do tipo $S_d = B$. Como $B = MDR$, tem-se que $S_d = MDR$. Por fim, os *bits* em verde definem que os registradores H e TOS estão habilitados para escrita, de modo que o valor da ULA passará para eles, ou seja, ao final da operação tem-se que $H = MDR$ e $TOS = MDR$.

Agora, considere que em seguida seja passada a seguinte instrução para a máquina:

$$001111000000000100100 \tag{3}$$

Os bits em azul definem que o registrador $LV$ está comandando o barramento B. Logo, a entrada $B$ da ULA será $B = LV$. Os *bits* em vermelho definem que a ULA executará uma operação do tipo $S_d = A + B$. Lembre-se que a entrada A da ULA **sempre** é igual ao valor armazenado no registrador H, que é o valor do MDR do ciclo da instrução anterior, ou seja $A = MDR$. Como $B = LV$, tem-se que $S_d = MDR + LV$. Por fim, os *bits* em verde definem que o registrador $MDR$ está habilitado para escrita, de modo que o valor da ULA passará para eles, ou seja, ao final da operação tem-se que $MDR = MDR + LV$.

Em outras palavras, essa sequência de duas instruções executa a operação de somar o valor armazenado em MDR com o valor armazenado em LV e guardar o resultado em MDR.

Para testar o código é necessário definir o estado inicial dos registradores do sistema. Um exemplo pode ser encontrado no arquivo *registradores_etapa2_tarefa2.txt*. O programa

é uma sequência de instruções de 21 *bits* como as descritas acima, tal como o arquivo *programa_etapa2_tarefa2.txt.*. O código do projeto até este ponto deve ler o arquivo de programa e executar a sequência de instruções presentes neste. A saída deve ser um *log* contendo as seguintes informações para cada instrução executada:

- O valor de todos os registradores: H, OPS, TOS, CPP, LV, SP, MBR, PC, MDR, e MAR no início e no fim de cada instrução.
- O registrador que está comandando o barramento B.
- Os registradores que estão habilitados para escrita no barramento C.
- Um registrador de instruções IR com a instrução sendo executada.

Note que ao contrário da etapa anterior, o valor de PC não está mais armazenando cada linha do código. Ainda, depois da primeira instrução, o valor dos registradores no início de um ciclo será igual ao valor dos registradores no final do ciclo anterior. Utilizando os dois arquivos fornecidos, o teste desta tarefa precisa apenas executar as linhas em sequência, e um exemplo de saída esperada pode ser visto em *saída_etapa2_tarefa2.txt*.

## Etapa 3

Nesta etapa será implementado o acesso à memória da Mic-1. Também serão abordadas as instruções que manipulam a pilha.

### Tarefa 1

Nesta tarefa devem ser implementados os *bits* de comando da memória da Mic-1. Sendo assim, uma palavra de *bit* correspondente a uma microinstrução passará a ter 23 *bits*, da forma:

| ULA | Barramento C | Memória | Barramento B |
|---|---|---|---|
| 8 *bits* | 9 *bits* | 2 *bits* | 4 *bits* |

Os 2 *bits* de memória são organizados na forma:

| WRITE | READ |
|---|---|
| $X_1$ | $X_0$ |

O funcionamento da memória ocorre a partir de dois arquivos: há um arquivo *.txt* para a memória de dados, contendo 8 endereços (ou linhas), cada uma contendo uma palavra de dados de 32 *bits*, e um arquivo denominado *.txt* que é a memória de instruções, contendo uma lista de instruções de tamanho (número de linhas) arbitrário, onde cada linha contém uma das palavras de 23 *bits* que funciona como uma instrução da Mic-1.

Excetuado um caso especial, que será descrito na Tarefa 2, apenas um dos *bits* WRITE ou READ estará em nível lógico alto de cada vez. O processo que cada um deles desencadeia quando alto é:

- WRITE: O valor contido no registrador MDR é escrito na linha do arquivo *dados.txt* correspondente ao endereço apontado pelo valor de MAR.
- READ: O valor contido na linha do arquivo *dados.txt* correspondente ao endereço apontado pelo valor de MAR é copiado para o registrador MDR.

**Os processos de escrita e leitura da memória (WRITE e READ, respetivamente) devem ocorrer apenas após a saída da ULA ter sido calculada e escrita nos registradores habilitados no barramento C.**

Considere o código de instrução abaixo:

$$001111000000000101001000 \tag{4}$$

Os bits em azul definem que o registrador $LV$ está comandando o barramento B. Logo, a entrada $B$ da ULA será $B = LV$. Os *bits* em vermelho definem que a ULA executará uma operação do tipo $S_d = A + B$. Lembre-se que a entrada A da ULA **sempre** é igual ao valor armazenado no registrador H. Como $B = LV$, tem-se que $S_d = H + LV$. Por fim, os *bits* em verde definem que o registrador $MDR$ está habilitado para escrita, de modo que o valor da ULA passará para ele, ou seja, ao final da operação tem-se que $MDR = H + LV$. Olhando para os bits de memória, em laranja, está habilitada uma operação de escrita. Sendo assim, após MDR ter sido escrito, a linha do arquivo .txt da memória de dados correspondente ao valor de MAR será preenchida com o valor armazenado em MDR, que neste caso é $H + LV$.

Agora, considere:

$$001101010000010010100100 \tag{5}$$

Na instrução acima, os *bits* de controle da ULA especificam que a operação a ser realizada será $S_d = B + 1$. Neste caso, o barramento $B$ está sendo comandado por $SP$, de modo que $S_d = SP + 1$. O resultado será escrito nos registradores $SP$ e $MAR$, ou seja, o valor de SP é incrementado em 1 e o resultado é copiado para MAR. Por fim, os *bits* em laranja especificam uma operação de leitura, de modo que a palavra de 32 *bits* contida no endereço de memória especificado por MAR será copiada para o registrador MDR.

Ao executar as instruções o código deve gerar um *log* contendo as seguintes informações para cada instrução executada:

- O valor de todos os registradores: H, OPS, TOS, CPP, LV, SP, MBR, PC, MDR, e MAR no início e no fim de cada microinstrução.
- O registrador que está comandando o barramento B.
- Os registradores que estão habilitados para escrita no barramento C.
- Os valores contidos nas linhas de memória de dados após a execução de cada microinstrução.

O código pode ser testado utilizando os arquivos *dados_etapa3_tarefa1.txt*, *microinstruções_etapa3_tarefa1.txt* e *registradores_etapa3_tarefa1.txt*, e um exemplo de saída pode ser visto em *saída_etapa3_tarefa1.txt*

**Entregável**

Uma instrução da IJVM é formada por uma sequência de microinstruções da Mic-1, onde cada uma das microinstruções é uma palavra de 23 *bits*. Por exemplo, considere a sequência de microinstruções abaixo:

```
H = LV
H = H+1
MAR = H; rd
MAR = SP = SP+1; wr
TOS = MDR
```

Nesta sequência são executadas as seguintes micro-instruções:

1. O valor de LV, que é a base do quadro de variáveis locais é armazenado em H.
2. O valor de H é incrementado e passado para MAR, e é realizada uma leitura da memória. Sendo assim, o registrador MDR passará a ter o valor do dado contido no endereço H+1, que da operação anterior resulta em LV+1, que é a segunda variável do quadro local de variáveis.
3. O endereço de topo da pilha SP é incrementado, e o seu valor é passado para MAR. É realizada uma operação que escrita, de modo que o valor contido em MDR será escrito para o endereço em MAR, que é o novo topo da pilha.
4. O registrador TOS, que armazena o valor do topo da pilha, recebe o valor de MDR.

A sequência de microinstruções acima executa a instrução ILOAD x, com $x = 1$ neste caso, em que a variável no endereço 1 acima da base do quadro de variáveis locais será carregada no topo da pilha. A linguagem acima pode ser traduzida em cinco códigos de 23 *bits*, correspondentes aos sinais de controle da Mic-1 modificada implementada até este ponto. Por exemplo, a terceira micro instrução $MAR = H; rd$ seria:

$$00111000100000000010000 \tag{6}$$

Veja que uma instrução ILOAD 3 seria equivalente a:

```
H = LV
H = H+1
H = H+1
H = H+1
MAR = H; rd
MAR = SP = SP+1; wr
TOS = MDR
```

E ILOAD 0 seria:

```
H = LV
MAR = H; rd
MAR = SP = SP+1; wr
TOS = MDR
```

De modo que ao traduzir uma instrução ILOAD x em microinstruções é necessário haver um dinamismo no número de microinstruções que incrementam o valor de H, de acordo com o argumento x da instrução.

Uma outra instrução é a DUP, em que o valor no topo da pilha é duplicado, dada por:

```
MAR = SP = SP+1;
MDR = TOS; wr
```

Neste caso, tem-se que:

1. O valor de SP, que aponta para o topo da pilha, é incrementado, e passado para MAR.
2. O valor do topo de pilha antigo TOS (que será igual ao novo, pois é uma operação de duplicação) é passado para MDR. É realizada uma escrita na memória.

Por fim, tem-se a instrução BIPUSH *byte*, que carrega um dado arbitrário de um *byte* no topo da pilha. Esta instrução possui uma diferença relativa as demais, já que umas das microinstruções deve ser gerada dinamicamente:

```
SP = MAR = SP+1
fetch
MDR = TOS = H; wr
```

Na sequência acima, todas as instruções menos a segunda são similares em sua implementação ao que foi visto nas instruções ILOAD x e DUP. A segunda instrução, que é apenas um fetch, pode parecer estranha, mas é meramente uma particularidade da arquitetura para que seja implementada a possibilidade de carregar um *byte* com dados arbitrários no MBR. Esta instrução contendo apenas um *fetch* é um caso especial, e o seu código será dado por:

$$xxxxxxxx000000001110000 \tag{7}$$

O indicativo desta instrução especial é o valor lógico alto nos *bits* WRITE e READ simultaneamente. Os *bits* em vermelho representados por $x$ correspondem a uma palavra de 8 *bits* qualquer. Quando a instrução acima ocorre, os 8 *bits* são passados para o MBR e o valor de MBR é passado para o registrador H, isto é, $H = MBR$ **sem** passar pela ULA. O preenchimento para que a palavra em H tenha 32 *bits* deve ser feito com zeros. Esta atribuição é uma diferença significativa da Mic-1 apresentada no livro, mas tem o propósito de simplificar a arquitetura para os propósitos deste projeto. Consequentemente, os 8 *bits* deste caso especial devem ter o valor do argumento da instrução BIPUSH *byte*. Desta maneira, um dado arbitrário é carregado no registrador H. Após isto, o fluxo de execução segue normalmente.

Esta particularidade gera uma diferença na instrução BIPUSH relativa as demais: quando esta é traduzida para uma sequência de microinstruções de 23 *bits* a segunda microsinstrução deve ter os seus 8 primeiros *bits* definidos pelo argumento da instrução.

**O entregável, então, é um código que reconhece as instruções ILOAD x, DUP, e BIPUSH *byte* em um documento .txt, interpreta estas instruções em uma sequência de microinstruções e as executa na Mic-1 modificada implementada nas etapas anteriores do projeto.** Para a avaliação, será fornecido um estado inicial para todos os registradores do sistema e para a memória de dados, como nos arquivos *dados_etapa3_tarefa1.txt* e *registradores_etapa3_tarefa1.txt*.

Em seguida, será fornecida uma sequência de instruções como as descritas acima em um arquivo *instruções.txt*. Um exemplo seria:

```
BIPUSH 00110011
DUP
ILOAD 1
```

O código do projeto deve ler o arquivo e executar a sequência de instruções presentes neste, traduzindo cada uma destas para a sequência apropriada de microinstruções. A saída de cada instrução será analisada, e o projeto será avaliado com base na proximidade com o resultado esperado para a execução das instruções.

Para tanto, o código deve, ao executar as instruções, necessariamente gerar um *log* contendo as seguintes informações para cada instrução executada:

- O valor de todos os registradores: H, OPS, TOS, CPP, LV, SP, MBR, PC, MDR, e MAR no início e no fim de cada microinstrução.
- O registrador que está comandando o barramento B.
- Os registradores que estão habilitados para escrita no barramento C.
- Os valores contidos nas linhas de memória de *dados.txt* após a execução de cada instrução.

A legibilidade do *log* e a facilidade de entrada dos dados para avaliação serão levadas em consideração na avaliação.