UNIVERSIDADE DO VALE DO ITAJAÍ
ESCOLA POLITÉCNICA
SISTEMAS PARA INTERNET

AIRTON BORGES
BRUNO DIAS

Avaliação 03: Memória Principal e Memória Virtual

Itajaí/SC
31/10/2023

AIRTON BORGES
BRUNO DIAS

Avaliação 03: Memória Principal e Memória Virtual

Relatório referente ao trabalho de Avaliação 03: Memória Principal e Memória Virtual, requerido na disciplina de Sistemas Operacionais, Curso de Sistemas Para Internet ofertado pela Escola Politécnica da Universidade do Vale do Itajaí

Professor: Prof. Felipe Viel

Itajaí/SC

31/10/2023

RESUMO

Airton Borges, Bruno Dias, **Trabalho 03**: Memória Principal e Memória Virtual. 31/10/2023. Trabalho avaliativo– Tecnólogo em Sistemas Para Internet – Escola Politécnica - Universidade do Vale do Itajaí, 2023.

Trabalho referente à média 2 da disciplina de Sistemas Operacionais, que trata tradução de memória virtual para memória física. O objetivo é fornecer um endereço de memória virtual, e encontrar seu número de página e o deslocamento dentro dos arquivos fornecidos, que age como uma simulação da memória física.

Devem ser informados a quantidade de bits do sistema operacional, e o tamanho do deslocamento de página, e então, o endereço virtual. A tradução da memória é feita pelo processo de extração do número de página que é encontrado nos bits mais significantes do endereço fornecido, e do deslocamento de página, que é encontrado nos bits menos significativos.

Neste documento, é relatado os algoritmos relacionados a tradução de memória física e virtual, tanto em sistemas operacionais que suportam registradores de 16 ou 32 bits. O código foi feito em C#, e as manipulações de bit foram feitas usando os operadores bitwise fornecidos pela linguagem.

1. Sumário

2. ENUNCIADO

1.1 Projeto

Suponha que um sistema tenha um endereço virtual de tamanho entre 16 bits à 32 bits com deslocamento na página de 256 b à 4 Kb. Escreva um programa que receba um endereço virtual (em decimal) na linha de comando ou leitura do arquivo addresses.txt faça com que ele produza o número da página e o deslocamento do endereço fornecido, sendo que essa posição indica qual a posição que será lido do arquivo data_memory.txt. Você irá encontrar esses arquivos no GitHub da disciplina, mais especificamente na pasta Memory (link repositório).

Por exemplo, seu programa seria executado da seguinte forma:

./virtual_memory_translate.exe 19986

Ou:

./virtual_memory_translate.exe addresses.txt

Seu programa produzirá:

O endereço 19986 contém: o número da página = 4 o deslocamento = 3602 o Valor lido: 50 (exemplo)

No caso, o número em binário é 0100 1110 0001 0010, sendo que 0100 diz respeito à página e 1110 0001 0010 diz respeito ao deslocamento na página. Você consegue conferir isso com a calculadora do Windows/Linux no modo programador. Para manipular os números em nível de bit, é recomendado usar os operadores bitwise (bit- a-bit) da linguagem escolhida. No caso o exemplo apresentado é para 16 bits. No caso de 32 bits, haveria mais 16 bits a esquerda (mais significativo) referentes ao número de páginas, 0000 0000 0000 0000 0100 1110 0001 0010, porém, ainda será traduzido para página 4 e deslocamento 3602.

Escrever este programa exigirá o uso do tipo de dados apropriado para armazenar 16 a 32 bits (short ou int). É recomendado que você também use tipos de dados sem sinal. Além disso, para endereços de 32 bits deve ser possível usar paginação hierárquica de 2 níveis mantendo 4 Kb, com cada nível tendo 10 bits de tamanho.

Para a implementação do código, você pode fazer um fork do repositório da disciplina no GitHub e usar o codespaces do GitHub para a implementação, onde ele irá executar o Visual Code em uma distro Linux Ubuntu com 2 núcleos e 8 GB de memória principal. Mas onde será executado seus códigos fica a critério do(s) aluno(s).

3. EXPLICAÇÃO E CONTEXTO DA APLICAÇÃO

Trabalho referente à média 2 da disciplina de Sistemas Operacionais, que trata tradução de memória virtual para memória física. O objetivo é fornecer um endereço de memória virtual, e encontrar seu número de página e o deslocamento dentro dos arquivos fornecidos, que age como uma simulação da memória física.

Devem ser informados a quantidade de bits do sistema operacional, e o tamanho do deslocamento de página, e então, o endereço virtual. A tradução da memória é feita pelo processo de extração do número de página que é encontrado nos bits mais significativos do endereço fornecido, e do deslocamento de página, que é encontrado nos bits menos significativos.

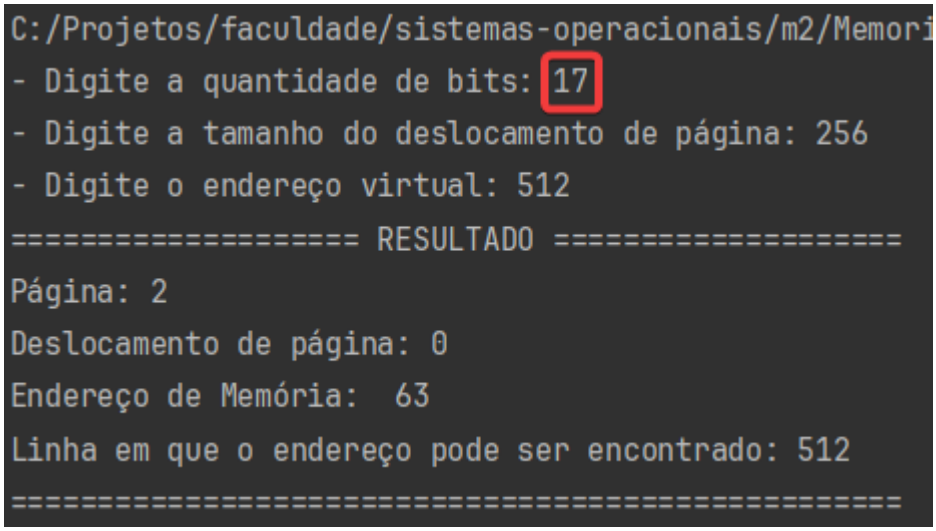
Neste documento, é relatado os algoritmos relacionados a tradução de memória física e virtual, tanto em sistemas operacionais que suportam registradores entre 16 e 32 bits. O código foi feito em C#, e as manipulações de bit foram feitas usando os operadores bitwise fornecidos pela linguagem.

4. RESULTADOS OBTIDOS COM A SIMULAÇÃO

4.1 Obtendo informações

Ao iniciar o programa, são requisitadas as informações relacionadas ao número de bits do sistema operacional, e caso o usuário digite um número entre 16 e 31, serão requisitadas as seguintes informações:

Informações 16 bits



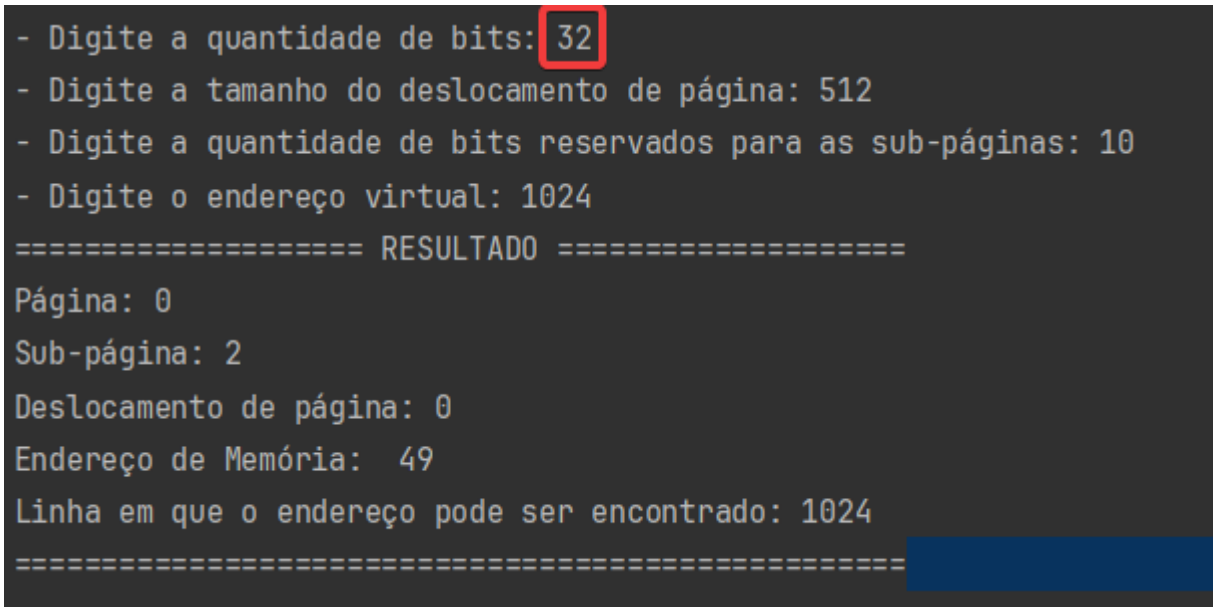
- Tamanho do deslocamento de página.
- Endereço virtual.

E como resultado, será informado:

- Número da Página.
- Deslocamento da página.
- Endereço de memória encontrado.
- E por fim, o número da linha no arquivo data_memory.txt onde esse endereço pode ser encontrado.

Caso o usuário digite 32, serão requeridas as seguintes informações:

Informações 32 bits



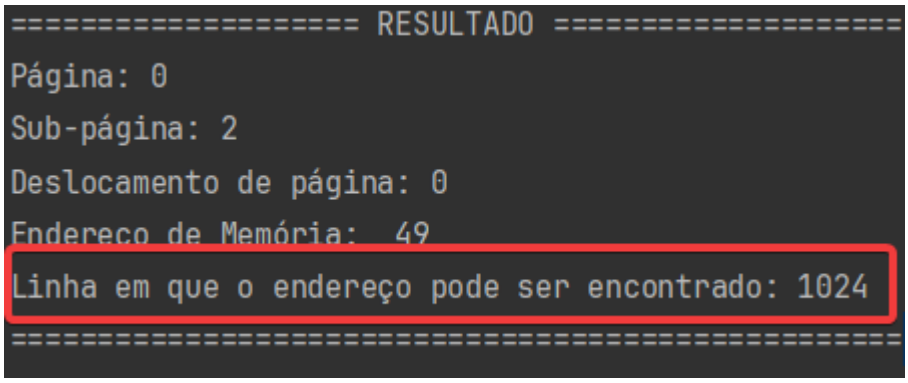
- Tamanho do deslocamento de página.
- Quantidade de bits reservados para o endereçamento das subpáginas.
- Endereço virtual.

E como resultado, são mostrados:

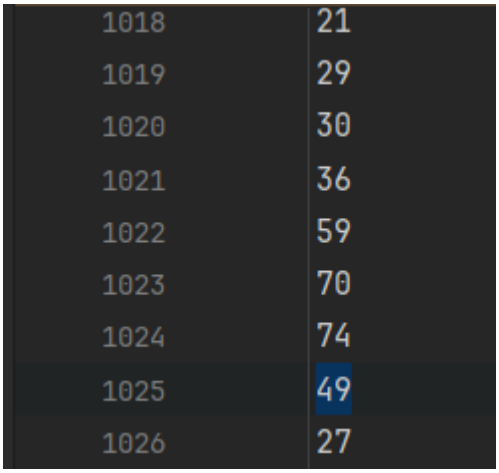
- O número da página.
- O número da subpágina.
- O endereço de memória.
- E por fim, o número da linha no arquivo data_adresses.txt onde esse endereço pode ser encontrado.

Para ambos os casos é fornecido o número da linha onde o registro de memória pode ser encontrado para checagem do resultado.

Linha onde o endereço pode ser encontrado



Endereço "Físico"



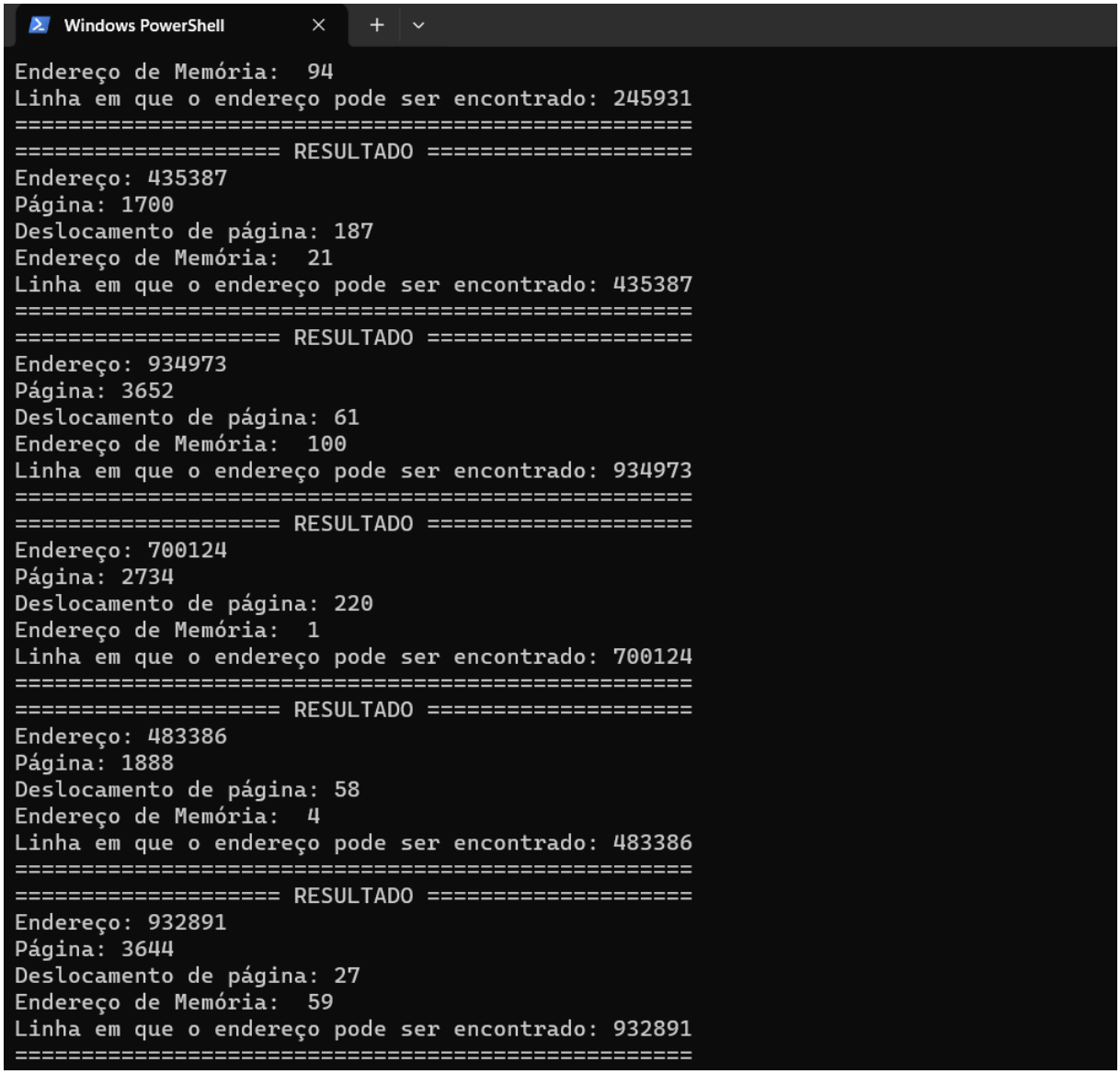
- *Nota: O número da linha é 1025 e não 1024 pois a contagem de linhas começa em 1, não em 0.*

Também é possível passar como argumento do executável, um arquivo de endereços

Arquivo como argumento

Serão mostradas todas as informações de endereçamento referentes a cada registro em cada linha do arquivo, dependendo da quantidade de bits e do número de páginas informadas.

Resultado arquivo como argumento



5. IMPLEMENTAÇÃO

Após obter o número de bits dos registradores de memória do sistema operacional, o programa pode seguir dois caminhos, a implementação correspondente aos 16 bits, ou aos 32 bits. Segue a explicação de ambas.

5.1 Implementação 16 bits

Após obter o input, é necessário obter o número da página e o deslocamento, para fazer isso, são utilizados dois métodos, **ObterNumeroDaPagina** e **ObterDeslocamento:**

ObterNumeroDaPagina e ObterDeslocamento

```
95      if (quantidadeBits >= 16 && quantidadeBits < 32)
96      {
97          Console.Write("- Digite a tamanho do deslocamento de página: ");
98          var tamanhoDeslocamentoPaginas = int.Parse(Console.ReadLine() ?? throw new Exception(message: "Digite um valor válido"));
99          Console.Write("- Digite o endereço virtual: ");
100         var quantidadeBitsPagina = (int)Math.Log2(tamanhoDeslocamentoPaginas);
101
102         var paginas = LerArquivo16b("data_memory.txt");
103         var enderecos = new List<uint>();
104
105         if (args.Length == 0)
106         {
107             Console.Write("- Digite o endereço virtual: ");
108             enderecos.Add(item: uint.Parse(Console.ReadLine() ?? throw new Exception(message: "Digite um valor válido")));
109         }
110         else
111             enderecos = LerEnderecos(pCaminho: "addresses_32b.txt");
112
113         enderecos.ForEach(pEndereco => 
114         {
115             numeroPagina = ObterNumeroDaPagina(quantidadeBitsPagina, pEndereco);
116             deslocamentoPagina = ObterDeslocamento(quantidadeBitsPagina, pEndereco);
```

A implementação dos dois os métodos requere a quantidade de bits referente ao deslocamento de página, então, ela é calculada obtendo o logaritmo de base 2 do tamanho do deslocamento de página informado pelo usuário. A seguir, a descrição de ambos.

5.1.1 ObterNumeroDaPagina

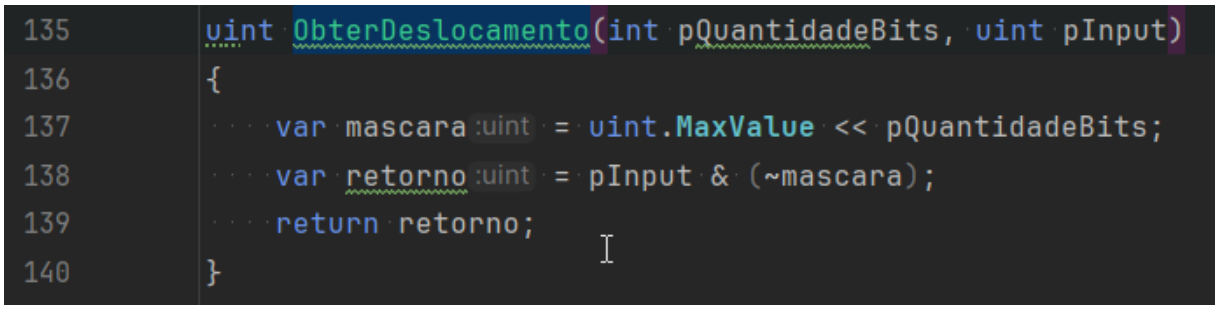
Código ObterNumeroDaPagina

```
152     uint ObterNumeroDaPagina(int pQuantidadeBits, uint pInput)
153     {
154         var mascara :uint = uint.MaxValue << pQuantidadeBits;
155         var bitsMaisSignificativos :uint = mascara & pInput;
156         var retorno :uint = bitsMaisSignificativos >> pQuantidadeBits;
157         return retorno;
158     }
```

- Primeiro, é obtida uma máscara referente aos bits mais significativos, fazendo um bit shift para a esquerda com a quantidade de bits reservados para o deslocamento de página.
- Ou seja, se a quantidade de bits for 256, que é equivalente a 2^8, serão deslocados 8 bits a esquerda do valor máximo de um uint, e teremos o resultado: (...)**1111 1111 0000 0000**. (*Nota: Serão descritos os estados de apenas 16 bits para simplificar a explicação, porém, o uint no c# tem 32 bits*).
- Após isso, essa máscara é aplicada no input por meio do operador lógico AND, que obtém apenas os bits levantados em ambos os lados da operação, que serão os bits mais significativos do input.
- No nosso caso, supondo que o input seja 257, o bit 8 e o bit 0 estarão levantados: **0000 0001 0000 0001**. Ao aplicar a máscara, apenas o bit 8 será mantido na variável **bitsMaisSignificativos**.
- Então, é usado o operador de right shift para trazer os bits mais significativos para o início do uint, assim, obtendo 1 como o número de página.

5.1.2 ObterDeslocamento

Código ObterDeslocamento

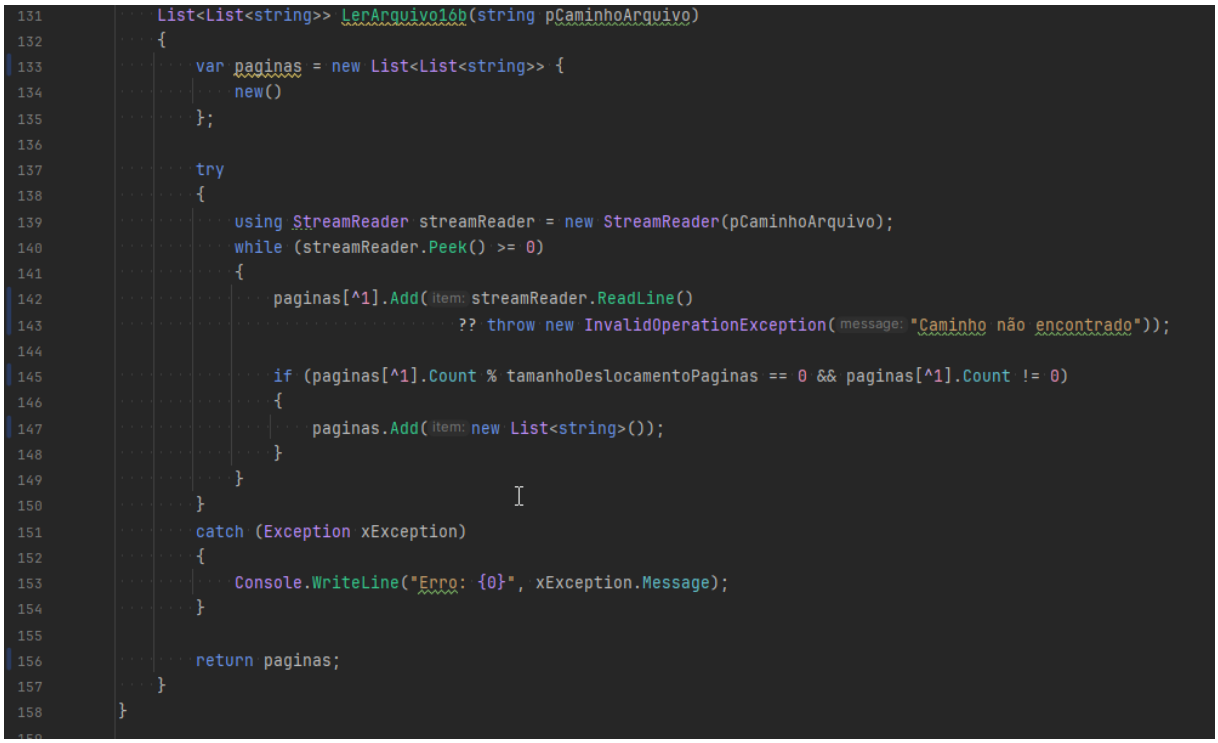


- É utilizada a mesma máscara do passo anterior, porém, é aplicado o operador de complemento ~ para obter uma versão invertida dela, assim, temos: **0000 0000 1111 1111**.
- Ao aplicar essa máscara com a operação AND no input, os bits menos significativos são mantidos, e seguindo o exemplo anterior onde o input é 257, obtemos **0000 0000 0000 0001**, ou seja, 1 como deslocamento dentro da página.

Dessa maneira, é possível obter o número da página e o deslocamento a partir do endereço virtual, porém, é preciso uma simulação da tabela de páginas para poder utilizar esses dados para encontrar o endereço físico no arquivo **data_memory.txt.**

5.1.3 LerArquivo16b

Código LerArquivo16b



```
List<List<string>> LerArquivo16b(string pCaminhoArquivo)
{
    var paginas = new List<List<string>> {
        new()
    };

    try
    {
        using StreamReader streamReader = new StreamReader(pCaminhoArquivo);
        while (streamReader.Peek() >= 0)
        {
            paginas[^1].Add(item: streamReader.ReadLine()
                                ?? throw new InvalidOperationException(message: "Caminho não encontrado"));

            if (paginas[^1].Count % tamanhoDeslocamentoPaginas == 0 && paginas[^1].Count != 0)
            {
                paginas.Add(item: new List<string>());
            }
        }
    }
    catch (Exception xException)
    {
        Console.WriteLine("Erro: {0}", xException.Message);
    }

    return paginas;
}
}
```

- Primeiro, existe um loop que lê cada linha, até o final do arquivo.
- E depois, é inicializado um **List<List<String>>,** onde a lista de listas de string é uma simulação da tabela de páginas, e a lista de string é uma simulação dos endereços em uma única página.
- Então, é preenchida a “primeira página” com cada linha do arquivo, até que ela seja preenchida com um número de registros igual ao tamanho do deslocamento de página.
- Após isso, é criada uma “página”, e o processo começa novamente, até que todos os endereços estejam mapeados.

Após esse mapeamento, é possível acessar o endereço de memória de maneira “similar” ao sistema operacional usando a tabela de páginas, informando um número de página como primeiro índice da matriz, e o segundo índice como deslocamento de página.

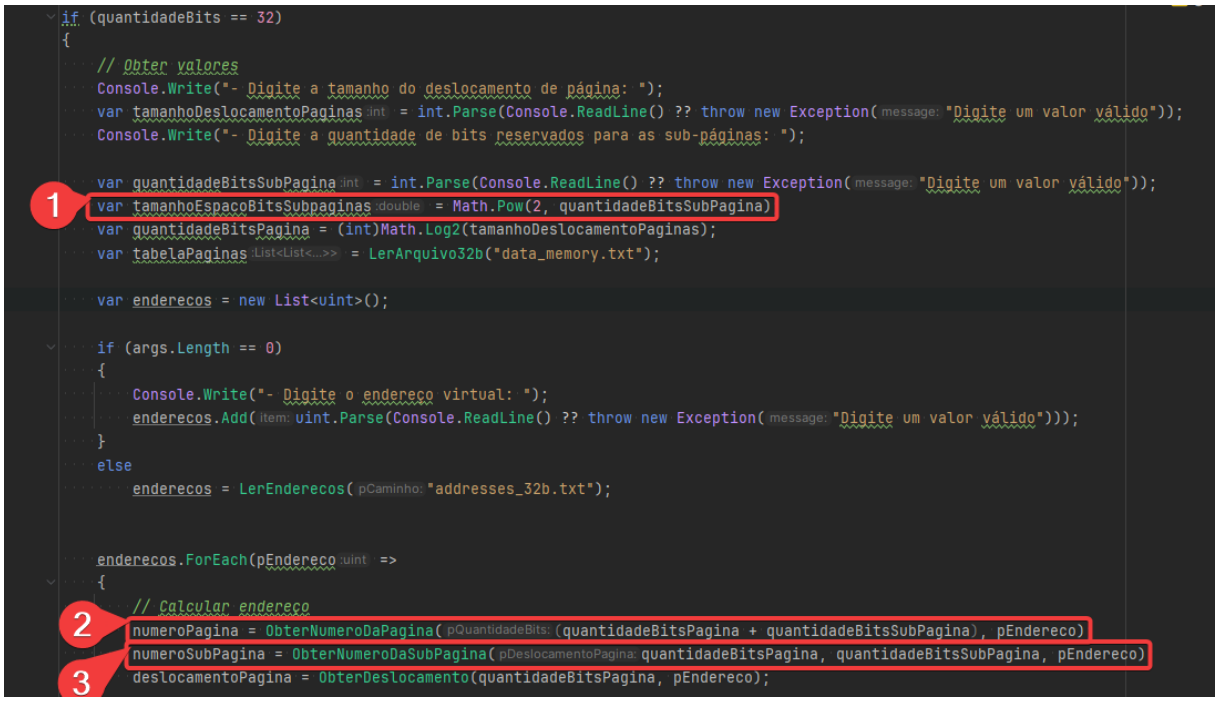
Obtendo o endereço físico

```
118         Console.WriteLine("==================== RESULTADO ====================");
119         var enderecoMemoria :string = paginas[(int)numeroPagina][(int)deslocamentoPagina];
120         var numeroLinha :long = tamanhoDeslocamentoPaginas * numeroPagina + deslocamentoPagina;
121
122         Console.WriteLine($"Endereco: {pEndereco}");
123         Console.WriteLine($"Página: {numeroPagina}");
124         Console.WriteLine("Deslocamento de página: " + deslocamentoPagina);
125         Console.WriteLine("Endereço de Memória:   " + enderecoMemoria);
126         Console.WriteLine("Linha em que o endereço pode ser encontrado: " + numeroLinha);
127         Console.WriteLine("=====================================================");
128     });
```

5.2 Implementação com 32 bits

A implementação com 16 bits é muito semelhante à anterior, porém, com algumas diferenças, agora, necessitamos trabalhar com a hierarquia de páginas que vem com a maior quantidade de bits.

Código principal 32 bits




```
if (quantidadeBits == 32)
{
    // Obter valores
    Console.Write("- Digite a tamanho do deslocamento de página: ");
    var tamanhoDeslocamentoPaginas = int.Parse(Console.ReadLine() ?? throw new Exception(message: "Digite um valor válido"));
    Console.Write("- Digite a quantidade de bits reservados para as sub-páginas: ");

    var quantidadeBitsSubPagina = int.Parse(Console.ReadLine() ?? throw new Exception(message: "Digite um valor válido"));
    var tamanhoEspacoBitsSubpaginas = Math.Pow(2, quantidadeBitsSubPagina)
    var quantidadeBitsPagina = (int)Math.Log2(tamanhoDeslocamentoPaginas);
    var tabelaPaginas = LerArquivo32b("data_memory.txt");

    var enderecos = new List<uint>();

    if (args.Length == 0)
    {
        Console.Write("- Digite o endereco virtual: ");
        enderecos.Add(item: uint.Parse(Console.ReadLine() ?? throw new Exception(message: "Digite um valor válido")));
    }
    else
        enderecos = LerEnderecos(pCaminho: "addresses_32b.txt");


    enderecos.ForEach(pEndereco => 
    {
        // Calcular endereço
        numeroPagina = ObterNumeroDaPagina(pQuantidadeBits: (quantidadeBitsPagina + quantidadeBitsSubPagina), pEndereco)
        numeroSubPagina = ObterNumeroDaSubPagina(pDeslocamentoPagina: quantidadeBitsPagina, quantidadeBitsSubPagina, pEndereco)
        deslocamentoPagina = ObterDeslocamento(quantidadeBitsPagina, pEndereco);
```

1. Dessa vez, também precisamos saber o tamanho do espaço que será reservado para as subpáginas, para poder mapear as subpáginas necessárias dentro da simulação da tabela de páginas. Isso será mais bem explicado em LerArquivo32b.
2. Agora, para obter o número da página, é passado a quantidade de bits ocupado pelo deslocamento de página, junto com a quantidade de bits da subpágina. O funcionamento interno do método é o mesmo explicado anteriormente em ObterNumeroDaPagina.
3. E é necessário um novo método para obter o deslocamento de página, que será mais bem explicado no tópico a seguir: ObterNumeroDaSubPagina.

5.2.1 ObterNumeroDaSubPagina

Código ObterNumeroDaSubPagina

```
143     uint ObterNumeroDaSubPagina(int pDeslocamentoPagina, int pQuantidadeBitsSubPagina, uint pInput)
144     {
145         var mascaraPagina = uint.MaxValue << pDeslocamentoPagina + pQuantidadeBitsSubPagina;
146         var mascaraSubPaginaComDeslocamento = ~mascaraPagina;
147         var mascaraSubPagina = (uint.MaxValue << pDeslocamentoPagina) & mascaraSubPaginaComDeslocamento;
148
149         var numeroFinalSubPagina = (mascaraSubPagina & pInput) >> pDeslocamentoPagina;
150         return numeroFinalSubPagina;
151     }
```

- Primeiro, obtemos uma máscara correspondente aos bits do deslocamento de página, e aos bits da subpágina. Ao inverter essa máscara, obtemos uma máscara correspondente ao número de bits menos significativos. Como exemplo, digamos que o deslocamento de página seja 4096, e sejam alocados 10 bits para a subpágina:
  - **0000 0000 0011 1111 1111 1111 1111 1111**
- Após isso, é preciso uma máscara que represente os bits mais significativos, para isso, apenas aplicamos um bit shift correspondente ao número de bits correspondentes ao deslocamento de página, no nosso caso, 12:
  - **1111 1111 1111 1111 1111 0000 0000 0000**
- Ao aplicar uma operação AND nas duas máscaras, obtemos enfim, a máscara referente à subpágina:
  - **0000 0000 0011 1111 1111 0000 0000 0000**
- Ao aplicar essa máscara no input, que pode ser por exemplo, 4097, obtemos:
  - **0000 0000 0000 0000 0001 0000 0000 0001 (*Input*)**
  - **0000 0000 0011 1111 1111 0000 0000 0000 (*Máscara*)**
  - **0000 0000 0000 0000 0001 0000 0000 0000 (*Resultado*)**
- Por fim, aplicamos um bit shift de correspondente a quantidade de bits referente ao deslocamento de página, para então obter o número da subpágina, que no neste exemplo, será **1**.

5.2.2 LerArquivo32b

Código LerArquivo32b

```
List<List<List<string>>> LerArquivo32b(string pCaminhoArquivo)
{
    var paginas = new List<List<List<string>>> {
        new()
        {
            new List<string>()
        }
    };

    try
    {
        using StreamReader streamReader = new StreamReader(pCaminhoArquivo);
        while (streamReader.Peek() >= 0)
        {
            paginas[^1][^1].Add(item: streamReader.ReadLine() ?? throw new InvalidOperationException(message: "Caminho não encontrado"));

            if (paginas[^1].Count % tamanhoEspacoBitsSubpaginas == 0 && paginas[^1][^1].Count == tamanhoDeslocamentoPaginas)
            {
                paginas.Add(item: new List<List<string>>()
                {
                    new ()
                });
            }
            if (paginas[^1][^1].Count % tamanhoDeslocamentoPaginas == 0 && paginas[^1][^1].Count != 0 )
            {
                paginas[^1].Add(item: new List<string>());
            }
        }
    }
    catch (Exception xException)
    {
        Console.WriteLine("Erro: {0}", xException.Message);
    }

    return paginas;
}
```

- Similar ao LerArquivo16b, existe uma simulação de uma tabela de página, porém, dessa vez, ela é dividida em dois níveis.
- Para representar este novo nível, foi criada uma matriz tridimensional, onde o primeiro índice representa a página externa, o segundo índice representa a subpágina, e o terceiro índice representa o deslocamento de memória.
- A lista responsável por simular as subpáginas, é preenchida com um número de registros igual ao tamanho em bits recebido como parâmetro do usuário, onde esse tamanho é o expoente de uma potência de base 2, que informa a quantidade máxima de registros na subpágina, por exemplo:
    - 10 bits para a subpágina = 1024 endereços em uma subpágina

Uso do índice tridimensional

```
32      Console.WriteLine("==================== RESULTADO ====================");
33      var enderecoMemoria :string  = tlb[(int)numeroPagina][(int)numeroSubPagina][(int)deslocamentoPagina];
34      var numeroLinha :double  = (numeroPagina * tamanhoEspacoBitsSubpaginas * tamanhoDeslocamentoPaginas)
35                  + (tamanhoDeslocamentoPaginas * numeroSubPagina + deslocamentoPagina);
36
37      Console.WriteLine($"Página: {numeroPagina}");
38      Console.WriteLine($"Sub-página: {numeroSubPagina}");
39      Console.WriteLine("Deslocamento de página: " + deslocamentoPagina);
40      Console.WriteLine("Endereço de Memória:  " + enderecoMemoria);
41      Console.WriteLine("Linha em que o endereço pode ser encontrado: " + numeroLinha);
42      Console.WriteLine("====================================================");
```

6. Considerações finais

Por meio deste trabalho foi possível entender e aplicar técnicas de tradução de endereçamento de memória, assim fixando o seu funcionamento e conseguindo entender na prática como a MMU trabalha, é claro, que com uma simulação, não é possível chegar ao nível de complexidade que ocorre dentro do chip, mas é possível ter uma visão básica dos algoritmos e processos usados para lidar com os endereços virtuais e físicos.