O nível de microarquitetura

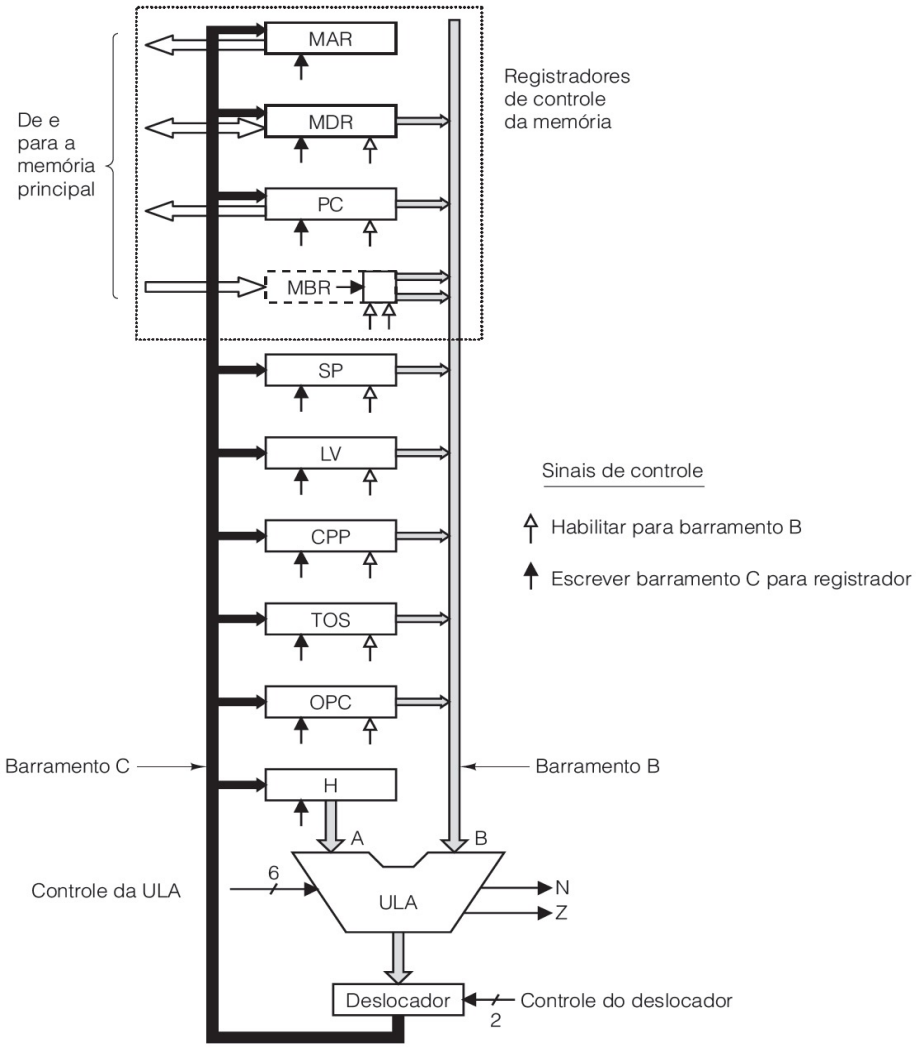
Um exemplo de microarquitetura

- Um modelo conveniente para o projeto de microarquitetura é pensar no projeto como um problema de programação no qual cada instrução no nível ISA é uma função a ser chamada por um programa mestre.

- Nesse modelo, o programa mestre é um laço simples, sem fim, que determina uma função a ser invocada, chama a função e então começa de novo.

- O microprograma tem um conjunto de variáveis denominado **estado** do computador, que pode ser acessado por todas as funções.

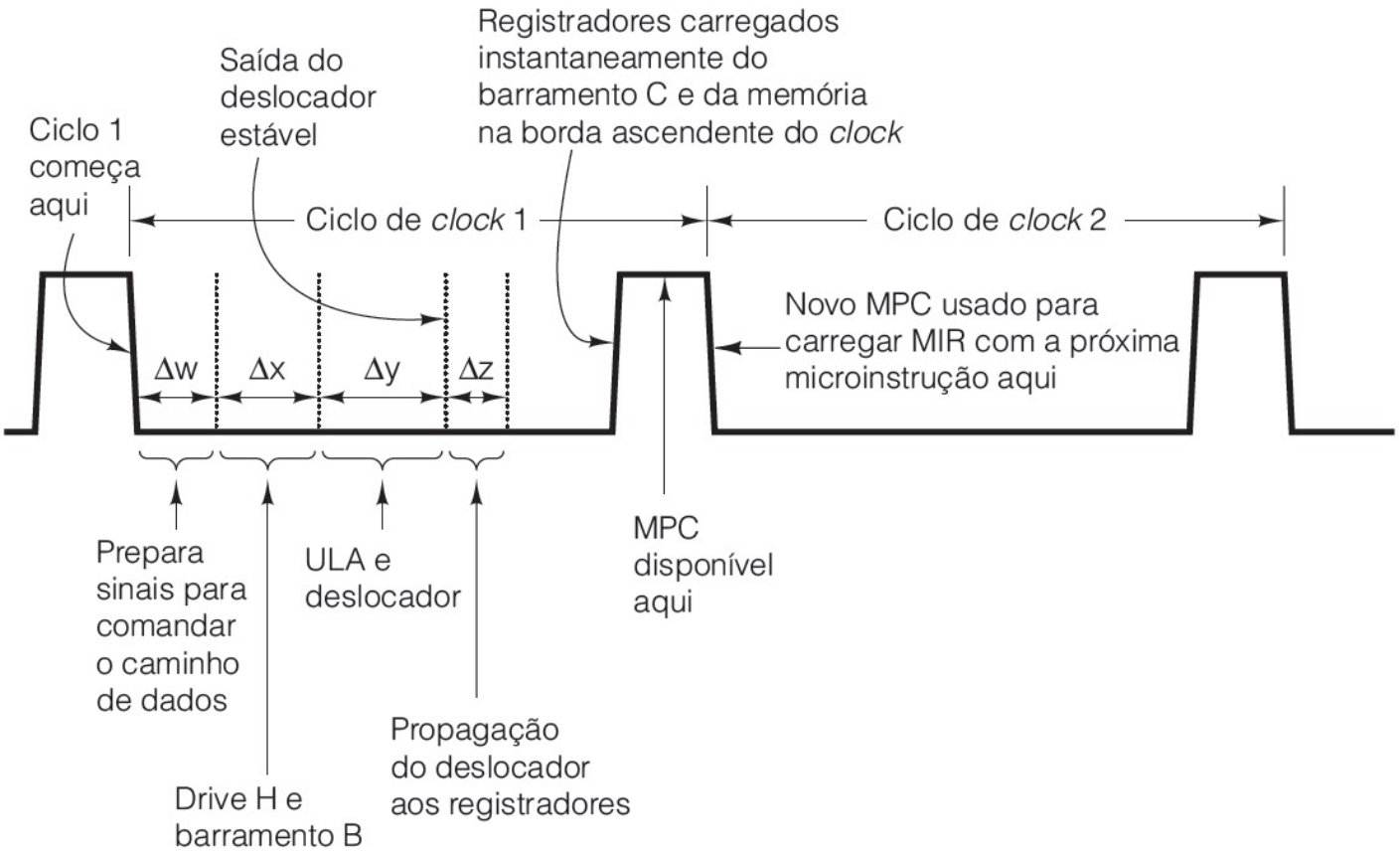
Um exemplo de microarquitetura



O **caminho de dados** é a parte da CPU que contém a ULA, suas entradas e suas saídas.

Um exemplo de microarquitetura

- Diagrama de temporização de um ciclo de caminho de dados.



Um exemplo de microarquitetura

- A porta de 32 bits é controlada por dois registradores, MAR (**Memory Address Register – registrador de endereço de memória**) e MDR (**Memory Data Register – registrador de dados de memória**).

- Para controlar o caminho de dados precisamos de 29 sinais, que podem ser divididos em cinco grupos funcionais, como descreveremos a seguir:

  - 9 sinais para controlar escrita de dados do barramento C para registradores.

  - 9 sinais para controlar habilitação de registradores dirigidos ao barramento B para a entrada da ULA.

Um exemplo de microarquitetura

- O sequenciador deve produzir dois tipos de informação a cada ciclo:

1. O estado de cada sinal de controle no sistema.
2. O endereço da microinstrução que deve ser executada em seguida.

- O item mais importante na parte do controle da máquina é uma memória denominada **armazenamento de controle**.

- Uma vez que o armazenamento de controle é uma memória, ele precisa de seu próprio registrador de endereço de memória e de dados de memória.

Projeto do nível de microarquitetura

- Velocidade pode ser medida de várias maneiras, mas dadas uma tecnologia de circuitos e uma ISA, há três abordagens básicas para aumentar a velocidade de execução:

1. Reduzir o número de ciclos de clock necessários para executar uma instrução.

2. Simplificar a organização de modo que o ciclo de clock possa ser mais curto.

3. Sobrepor a execução de instruções.

Projeto do nível de microarquitetura

- O número de ciclos de *clock* necessários para executar um conjunto de operações é conhecido como **comprimento do caminho**.
- Reduzir o número de ciclos de instrução necessários para buscar instruções requer mais do que apenas um circuito adicional para incrementar o PC.
- Sobrepor a execução de instruções é, de longe, o mais interessante e oferece a melhor oportunidade para drásticos aumentos de velocidade.

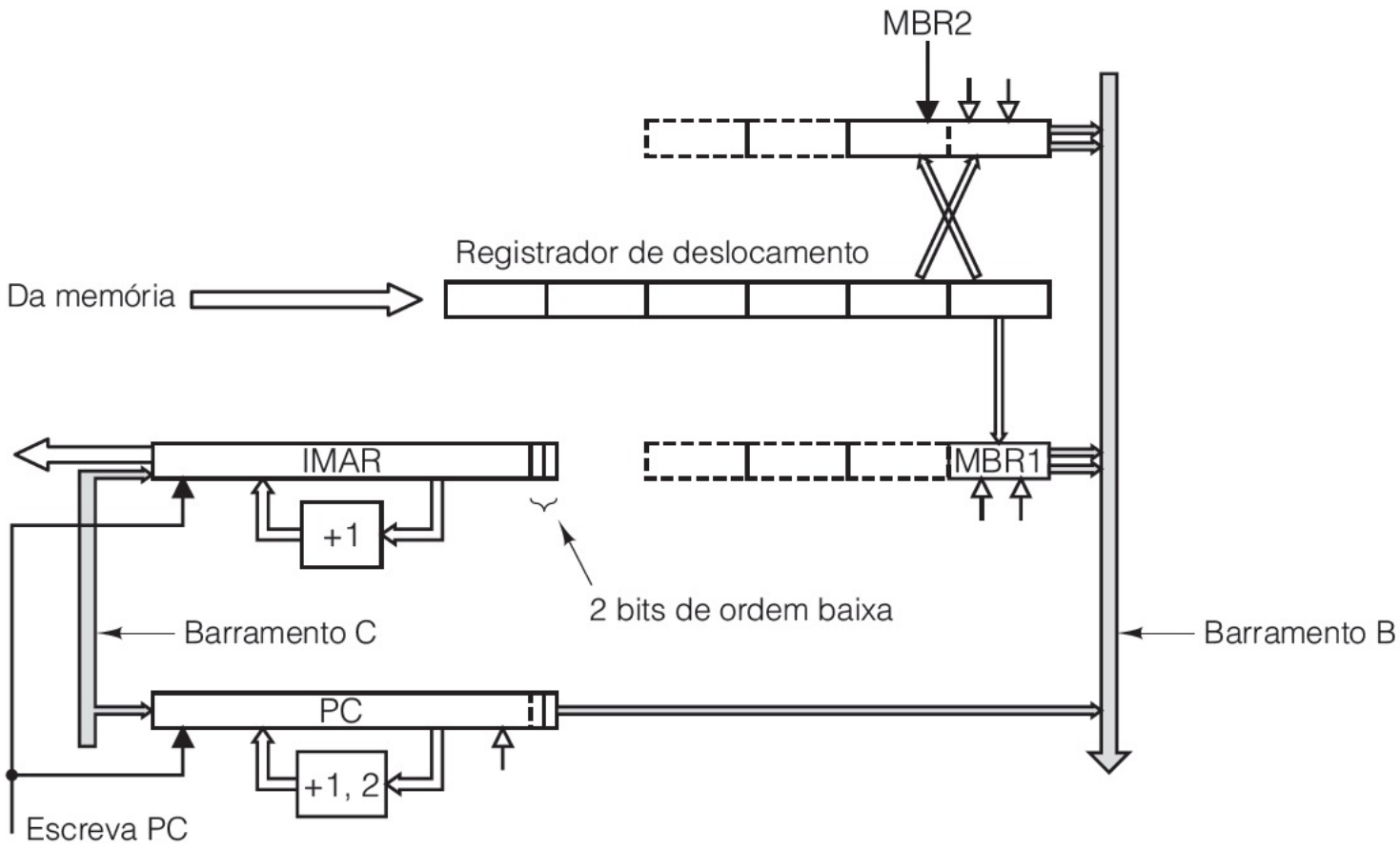
Projeto do nível de microarquitetura

- Para cada instrução, podem ocorrer as seguintes operações:

1. O PC é passado pela ULA e incrementado.
2. O PC é usado para buscar o próximo byte na sequência de instruções.
3. Operandos são lidos da memória.
4. Operandos são escritos para a memória.
5. A ULA efetua um cálculo e os resultados são armazenados de volta.

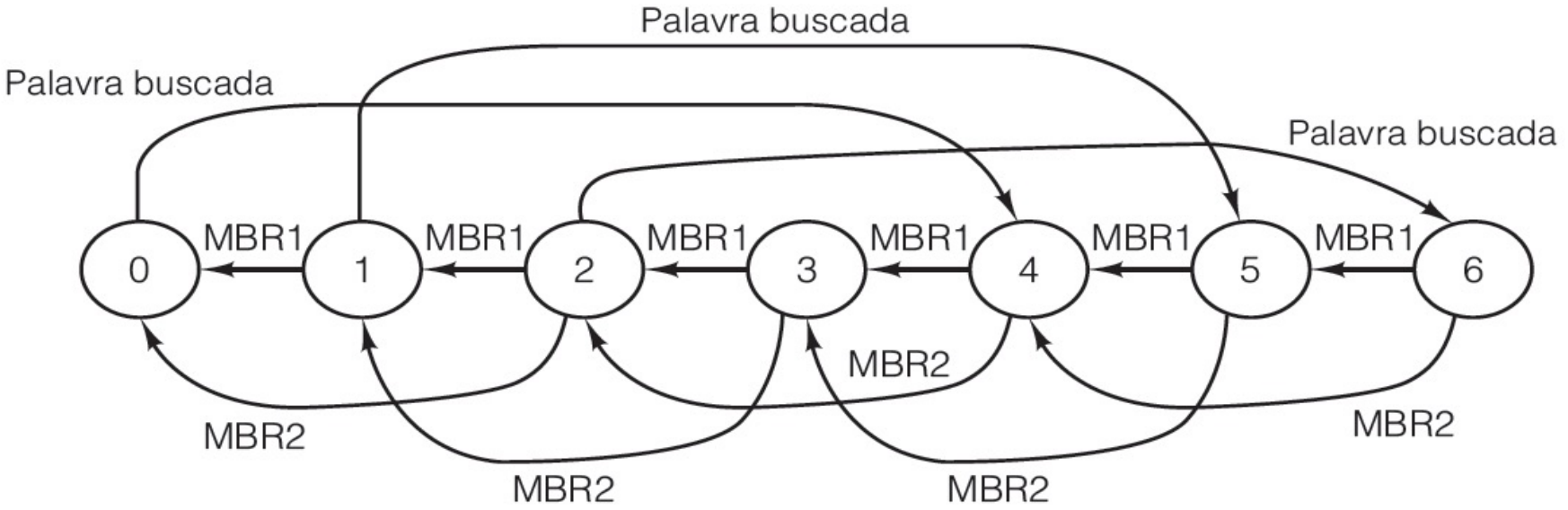
Projeto do nível de microarquitetura

- Unidade de busca para a Mic-1.



Projeto do nível de microarquitetura

- Máquina de estado finito para implementar a IFU.

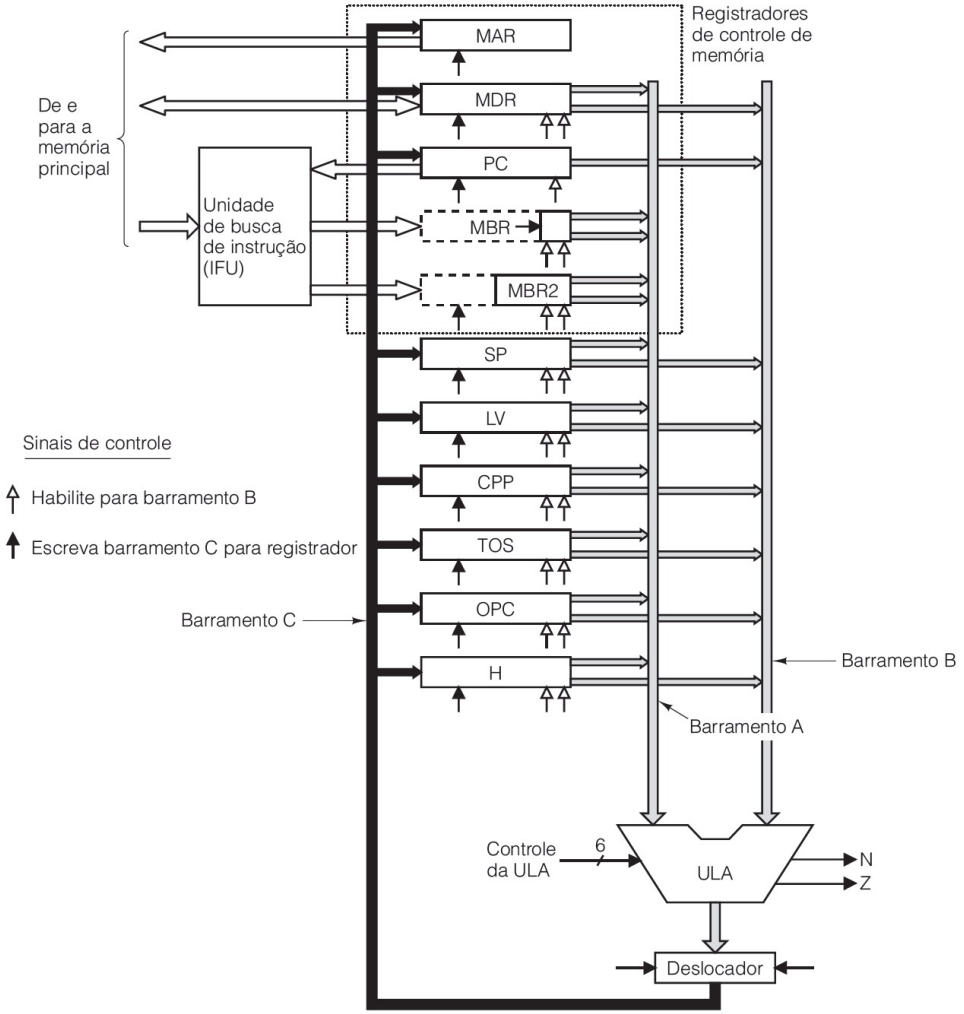


Transições

MBR1: ocorre quando MBR1 é lido
MBR2: ocorre quando MBR2 é lido
Palavra buscada: ocorre quando uma palavra da memória é lida e 4 bytes são colocados no registrador de deslocamento

Projeto do nível de microarquitetura

- A IFU pode reduzir muito o comprimento do caminho da instrução média.
- O caminho de dados para a Mic-2.



Projeto do nível de microarquitetura

- Que tal tentar reduzir o tempo de ciclo?

- Há três componentes importantes no ciclo do caminho de dados propriamente dito:

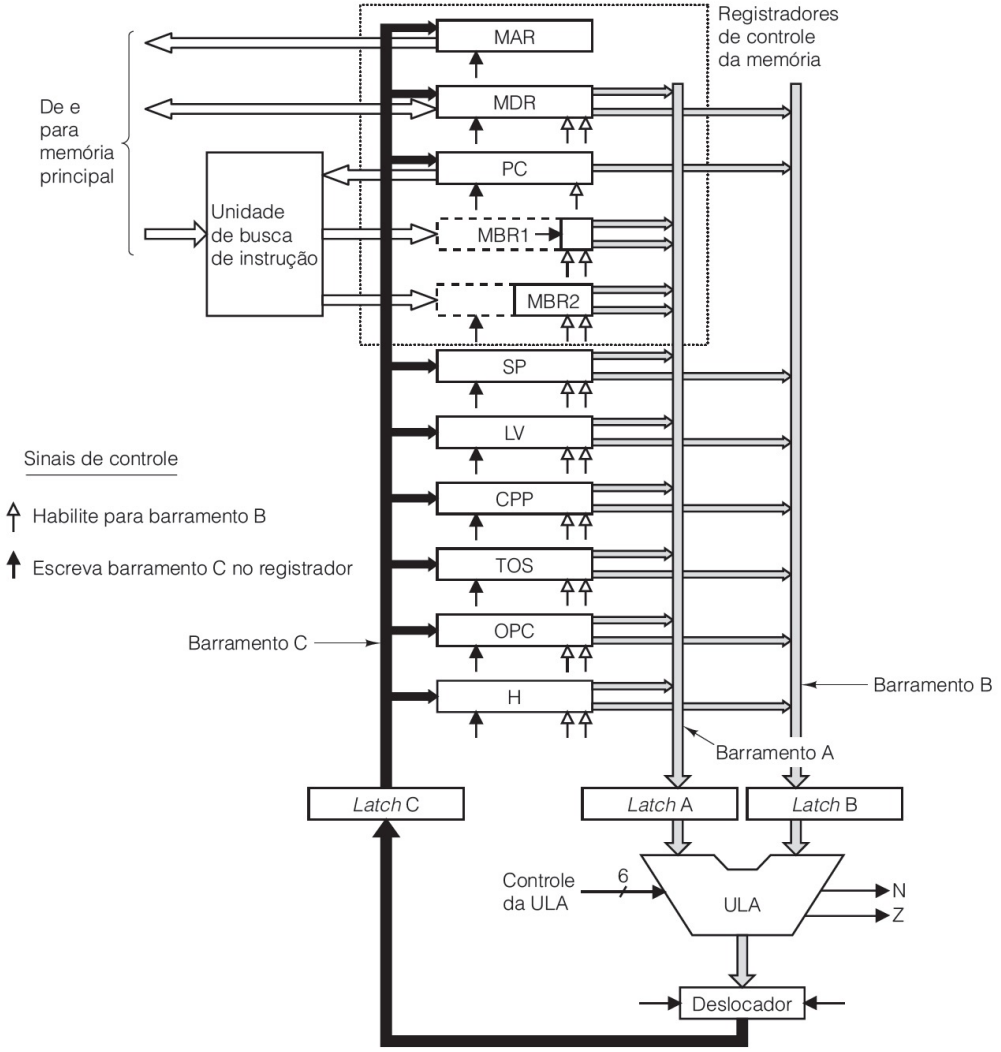
1. O tempo para levar os registradores selecionados até os barramentos A e B.

2. O tempo para que a ULA e o deslocador realizem seu trabalho.

3. O tempo para os resultados voltarem aos registradores e serem armazenados.

Projeto do nível de microarquitetura

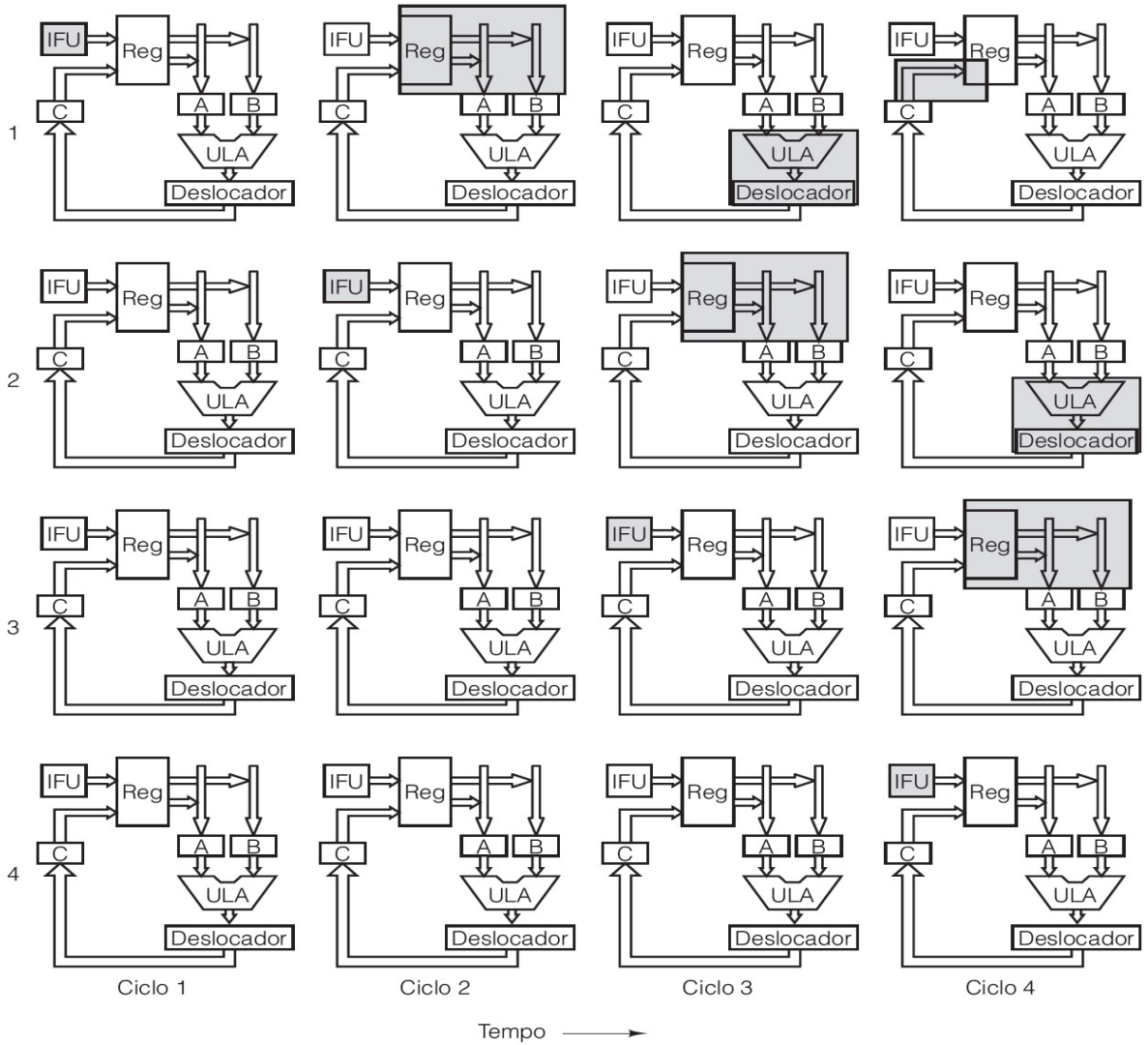
- Na figura ao lado, mostramos uma nova arquitetura e três barramentos, incluindo a IFU, mas com três registradores adicionais, cada um inserido no meio de cada barramento.



Projeto do nível de microarquitetura

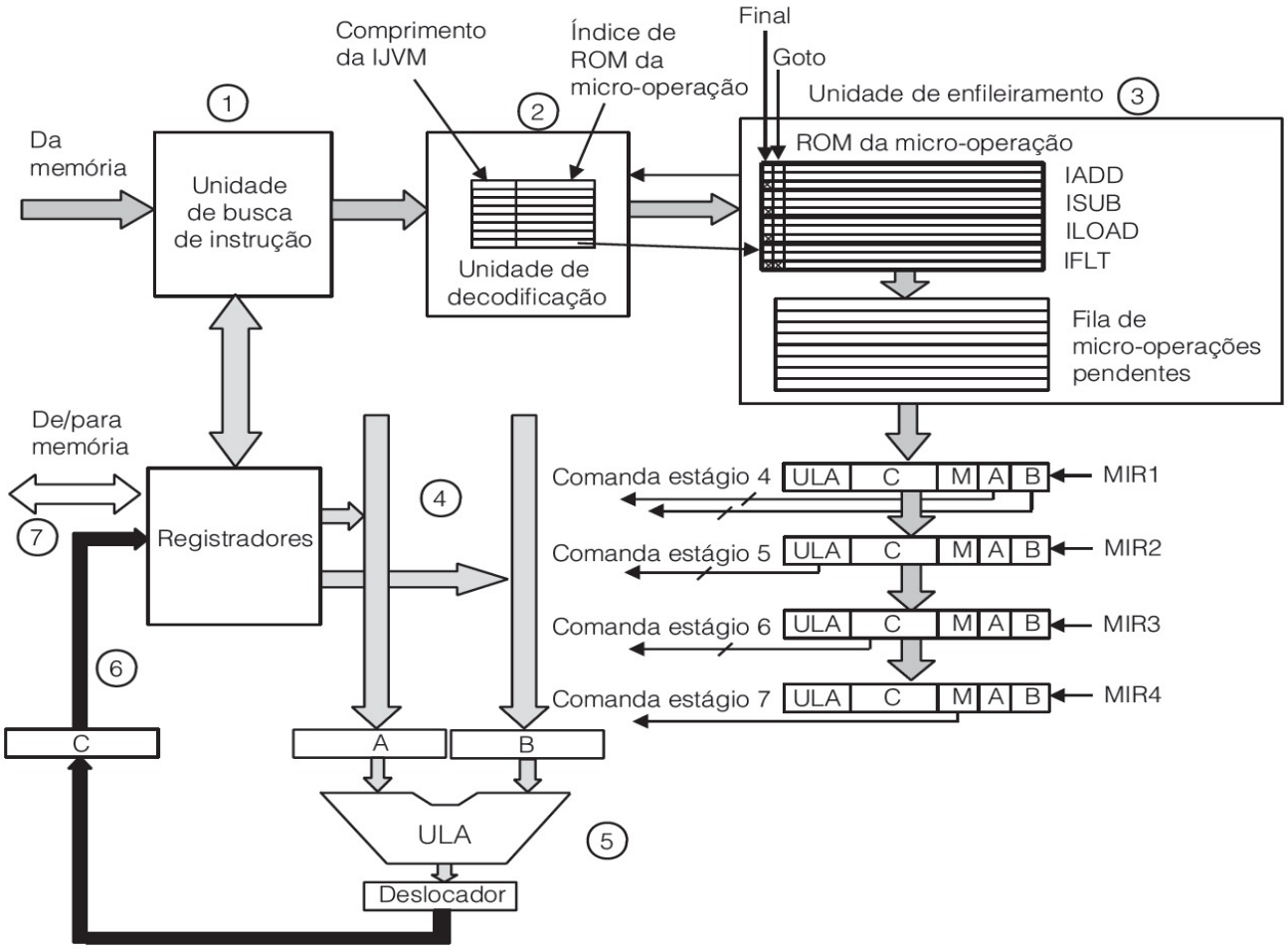
- Embora o programa Mic-3 leve mais ciclos do que o programa Mic-2, ainda assim é mais rápido.
- O *pipeline* deixou a máquina mais rápida, ainda que tivéssemos de protelar uma vez para evitar uma dependência.
- *Pipeline* é uma técnica fundamental em todas as CPUs modernas, portanto, é importante entendê-lo bem.
- A seguir vemos o caminho de dados ilustrado graficamente como um *pipeline*.

Projeto do nível de microarquitetura



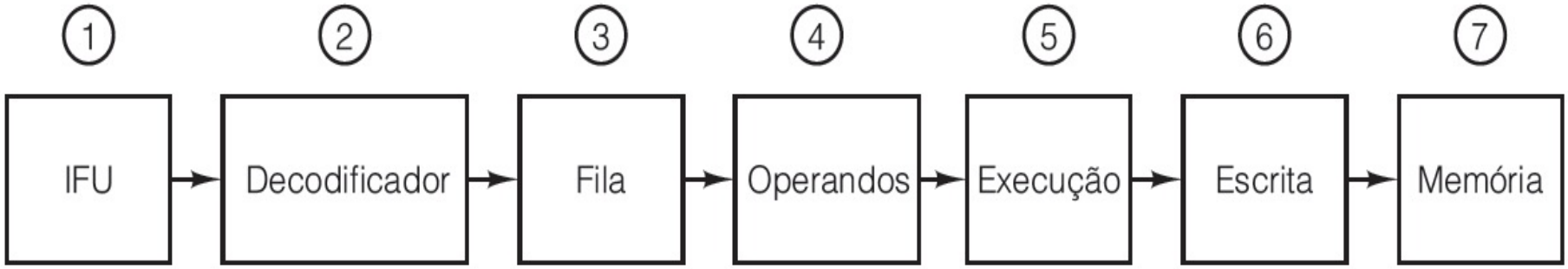
Projeto do nível de microarquitetura

- Nossa última microarquitetura é a Mic-4.



Projeto do nível de microarquitetura

- A Mic-4 tem um projeto de alto *pipelining*, com sete estágios e hardware muito mais complexo.
- O *pipeline* é mostrado em esquema na figura abaixo.



Melhoria de desempenho

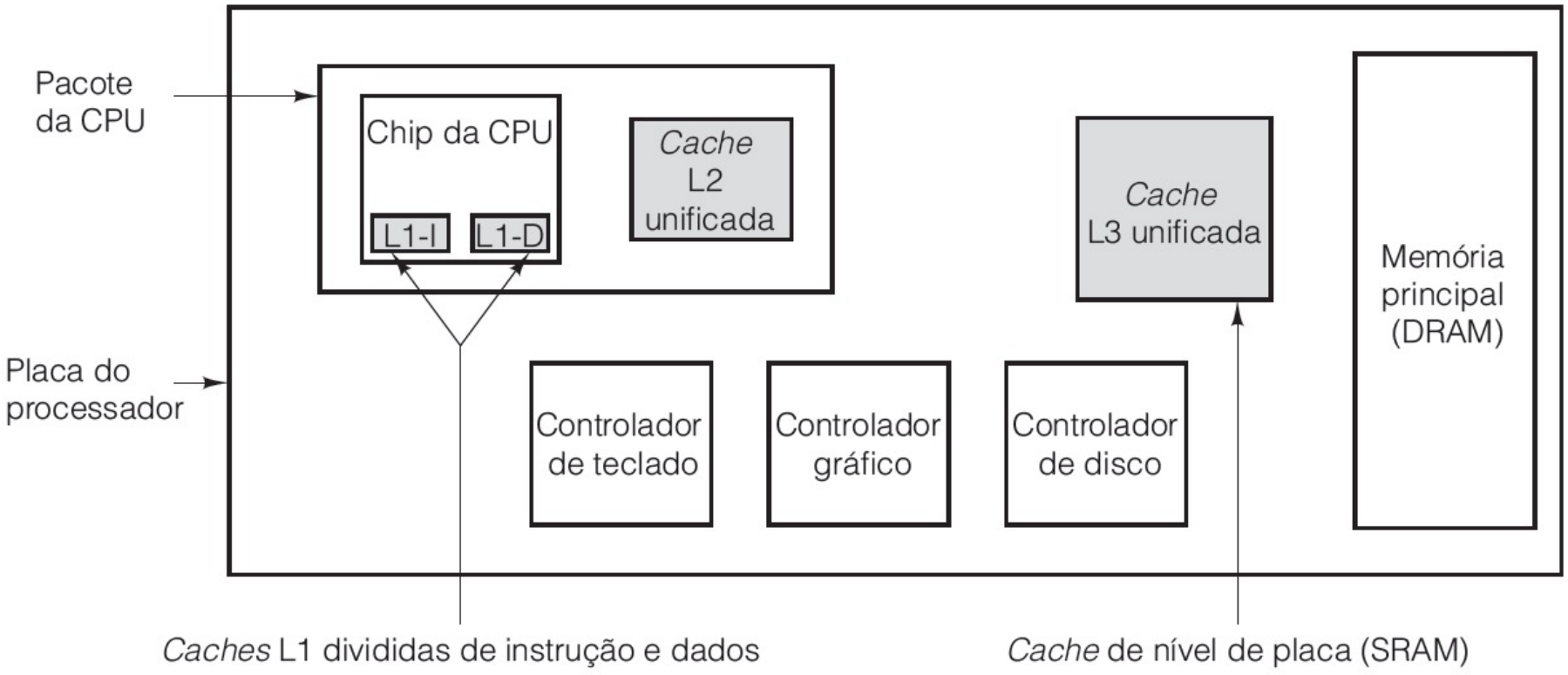
- Melhorias de implementação são modos de construir uma nova CPU ou memória para fazer o sistema funcionar mais rápido sem mudar a arquitetura.
- Um modo de melhorar a implementação é usar um *clock* mais rápido.
- Alguns tipos de melhorias só podem ser feitos com a alteração da arquitetura.
- Para conseguir um desempenho completo, o software tem de ser alterado, ou ao menos recompilado com um novo compilador que aproveita as novas características.

Memória *cache*

- Uma ***cache*** guarda as palavras de memória usadas mais recentemente em uma pequena memória rápida, o que acelera o acesso a elas.
- É possível obter muitos benefícios com *caches* separadas para instruções e dados, algo que muitas vezes denominamos ***cache* dividida**.
- Pode haver três ou mais níveis de cache à medida que se exigem sistemas de memória mais sofisticados.

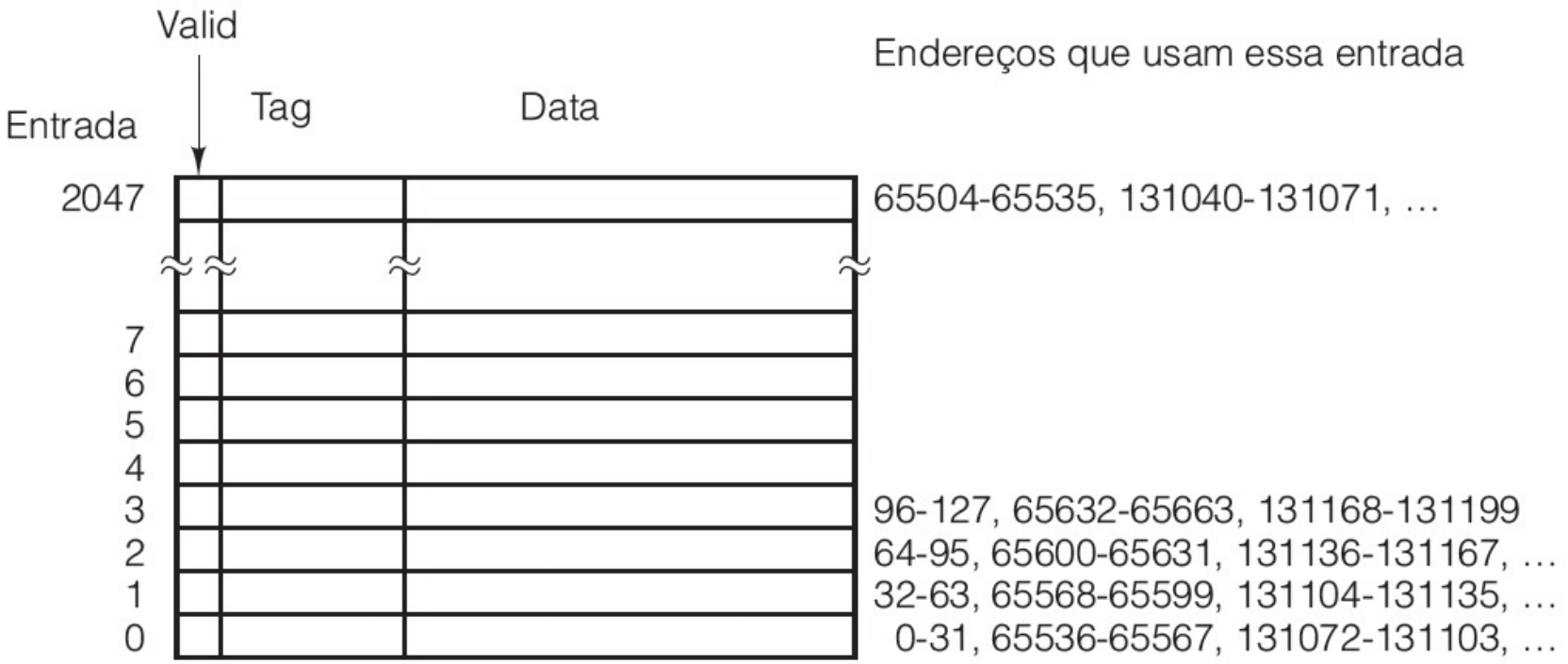
Memória *cache*

- Sistema com três níveis de *cache*.



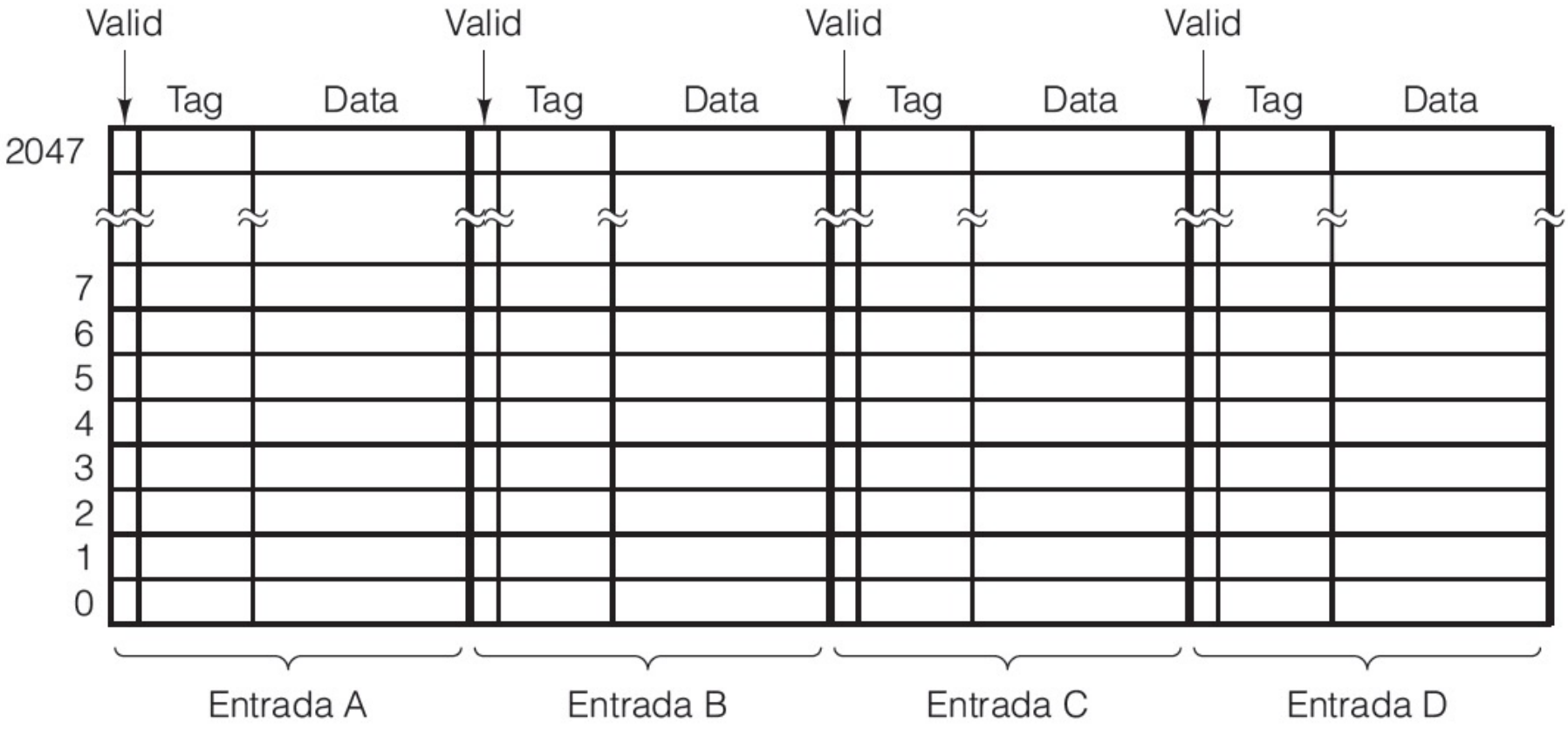
Memória *cache*

- A *cache* mais simples é conhecida como ***cache* de mapeamento direto**.



Memória *cache*

- Uma *cache* associativa de conjunto de quatro vias é ilustrada abaixo:



Previsão de desvio

- Programas não são sequências de código linear – estão repletos de instruções de desvio. Considere:
- Fragmento de programa.

```
if (i == 0)
        k = 1;
else
        k = 2;
```

- Sua tradução para uma linguagem de montagem genérica.

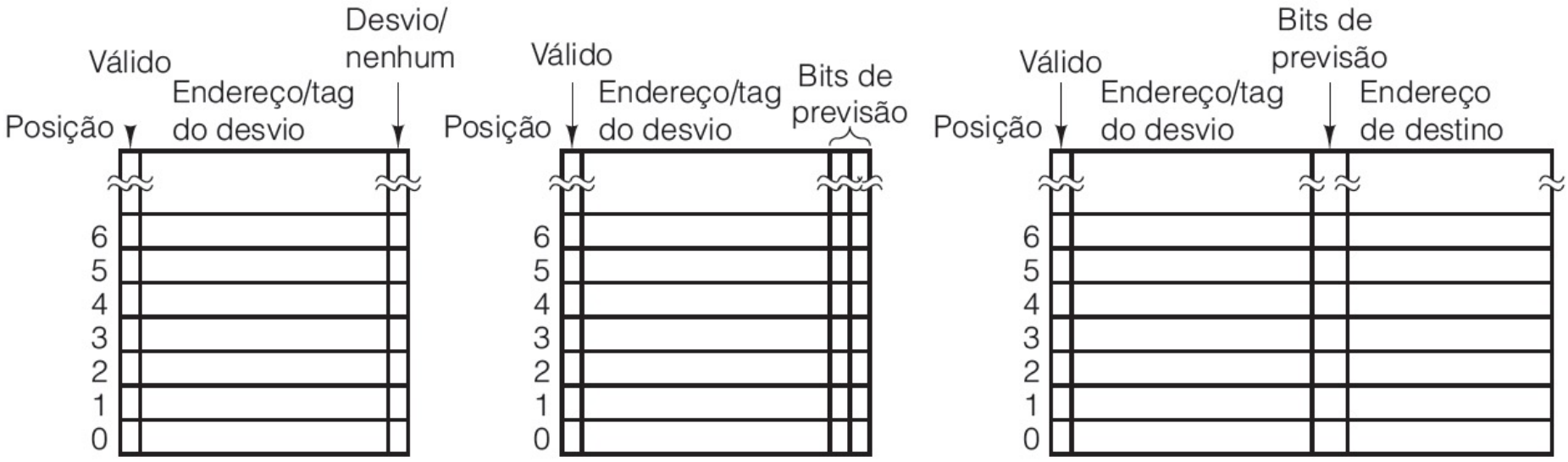
```
        CMP i,0      ; compare i com 0
        BNE Else     ; Desvie se for diferente
Then:   MOV k,1      ; Mova 1 para k
        BR Next      ; Desvio incondicional
Else:   MOV k,2      ; Mova 2 para k
Next:
```

Previsão de desvio

- Devemos observar que duas das cinco instruções são desvios. O problema está na natureza do *pipelining*.
- A posição após um desvio é denominada **posição de retardo** (*delay slot*).
- Por mais que desvios incondicionais sejam irritantes, os desvios condicionais são piores.
- Se um desvio for previsto corretamente, não há nada de especial a fazer.
- A execução apenas continua no endereço de destino.

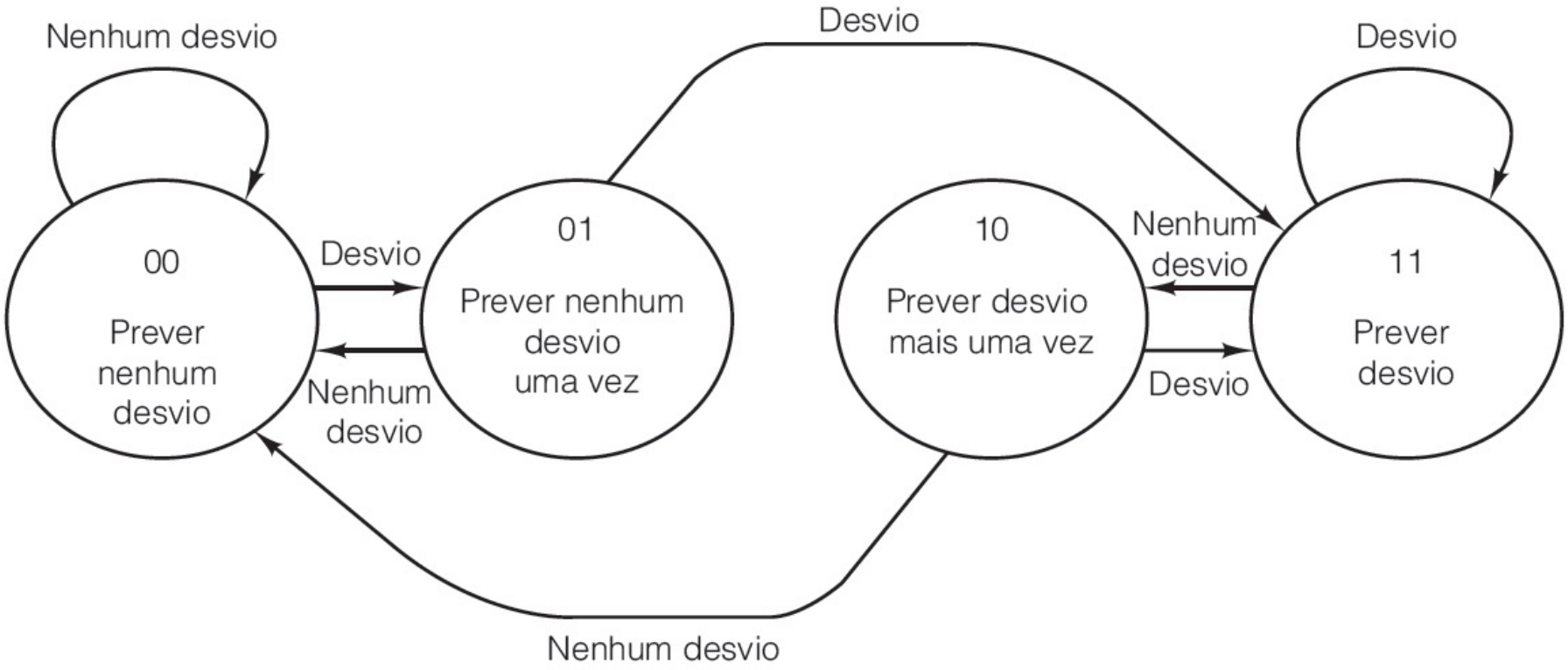
Previsão de desvio

- Histórico de desvio de 1 bit.
- Histórico de desvio de 2 bits.
- Mapeamento entre endereço de instrução de desvio e endereço de destino.



Previsão de desvio

- Máquina de estado finito de 2 bits para previsão de desvio.



Execução fora de ordem e renomeação de registrador

- Introduzimos uma nova técnica: **registrador de renomeação**.
- A sábia unidade de decodificação transfere a utilização de R1 em I6 (ciclo 3) e I7 (ciclo 4) para um registrador secreto, S1, que não é visível para o programador.
- Agora, I6 pode ser emitida ao mesmo tempo em que I5.
- CPUs modernas costumam ter dezenas de registradores secretos para usar com renomeação de registrador.
- Essa técnica pode eliminar dependências WAR e WAW.

Execução especulativa

- Programas de computador podem ser desmembrados em **blocos básicos**, em que cada um consiste em uma sequência linear de código com um ponto de entrada no início e uma saída no final.

- Calculamos as somas dos cubos dos inteiros pares e ímpares até algum limite e as acumulamos em *evensum* e *oddsum*, respectivamente.

- Dentro de cada bloco básico, as técnicas de reordenação da seção anterior funcionam bem.

Execução especulativa

- A execução especulativa introduz alguns problemas interessantes.
- Um deles é que nenhuma das instruções especulativas tem resultados irrevogáveis, porque mais tarde pode-se descobrir que elas não deveriam ter sido executadas.
- O que acontece se uma instrução executada por especulação causar uma exceção?
- Uma solução presente em várias máquinas é inserir uma instrução SPECULATIVE-LOAD que tenta buscar a palavra na *cache*, mas, se ela não estiver lá, desiste.