GABARITO DA AD1
Organização de Computadores 2019.1

1. (1,0) Considere uma máquina HIPOTÉTICA com arquitetura semelhante àquela apresentada em aula. Pode-se endereçar no máximo 64K células de memória onde cada célula armazena uma palavra e cada instrução tem o tamanho de uma palavra. Todas as instruções desta máquina possuem o mesmo formato: um código de operação, que permite a existência de um valor máximo de 256 códigos, e dois operandos, que indicam endereços de memória.

   a) Qual o tamanho mínimo do CI ?
   CI terá o tamanho necessário para endereçar toda a memória.
   Tamanho da memória = N = 64K células
   CI = log_2 64 K células = ***16*** **bits**

   b) Qual a capacidade máxima da memória em bits ?
   T = M x N => T = tamanho da célula x quantidade de células da MP =>
   quantidade de células da MP (N) = 64 K células
   O tamanho da célula (M) deverá armazenar uma palavra, esta terá o tamanho da instrução
   Cada instrução = codOper + 2 operandos
   CodOper deverá permitir 256 códigos diferentes (instruções) => CodOper = 8 bits
   Operando corresponde a 1 endereço de memória = 16 bits
   Instrução = CodOper + 2 Operandos => Instrução = 8 + 2 × 16 => Tam. instrução = 40 bits
   Tamanho da célula (M) = tamanho da instrução = 40 bits
   T = M x N => T = 40 bits/célula x 64K células => ***T = 2560 K bits ou 320 K bytes***

   c) Qual o tamanho mínimo do REM ?
   REM = Barramento de endereços, este terá a capacidade de endereçar 64K células = N
   N = 64Kcélulas => $N = 2^{16}$ => e = 16 bits
   REM = barramento de endereços = ***16 bits***

   d) Qual o tamanho mínimo do RI ?
   O tamanho mínimo para RI deverá ser o tamanho de uma instrução, esta tem 40 bits
   O tamanho mínimo para RI deverá ser ***40 bits***

   e) Qual o tamanho do barramento de endereços ?
   REM = barramento de endereços ***= 16 bits***

   f) Se a largura do barramento de dados desta máquina for igual à metade do tamanho de uma instrução, como funcionará o ciclo de busca ?
   Serão necessários 2 ciclos de busca para obter uma instrução

2. (0,5) Descreva passo a passo as operações de leitura da memória e de escrita na memória, indicando como os registradores RDM e REM são utilizados e como a unidade de controle gera os sinais necessários.

   Passos de uma operação de leitura
   1) (REM) <- (outro registrador da UCP)
   1.1) O endereço é colocado no barramento de endereços
   2) Sinal de leitura é colocado no barramento de controle
   2.1) Decodificação do endereço e localização da célula na memória
   3) (RDM) <- (MP(REM)) pelo barramento de dados
   4) (outro registrador da UCP) <- (RDM)

   Passos de uma operação de escrita
   1) (REM) <- (outro registrador)
   1.1) O endereço é colocado no barramento de endereços
   2) (RDM) <- (outro registrador)
   2.1) O dado é colocado no barramento de dados
   3) Sinal de escrita é colocado no barramento de controle

3. (1,5) Considere uma máquina HIPOTÉTICA que possa endereçar 512 Mbytes de memória física, utilizando endereço referenciando byte, e que tenha a sua memória organizada em blocos de 32 bytes. Ela possui uma memória cache que pode armazenar 8K blocos, sendo um bloco por linha. Mostre o formato da memória cache, indicando os campos necessários (tag, bloco) e o número de bits para cada campo, e o formato de um endereço da memória principal, indicando os bits que referenciam os campos da cache, para os seguintes mapeamentos:

   a) Mapeamento direto.

Memória Principal

- ⇒ *Tamanho da memória (em bytes) = 512 Mbytes, como 1 célula referencia a 1 byte, temos N = 512 Mcélulas*
- ⇒ *Será organizada em blocos de 32 bytes, como 1 célula = 1 byte, cada bloco = 32 células, K = 32*
- ⇒ *Sendo N o tamanho endereçável da memória e K que é a quantidade de células por blocos temos:*

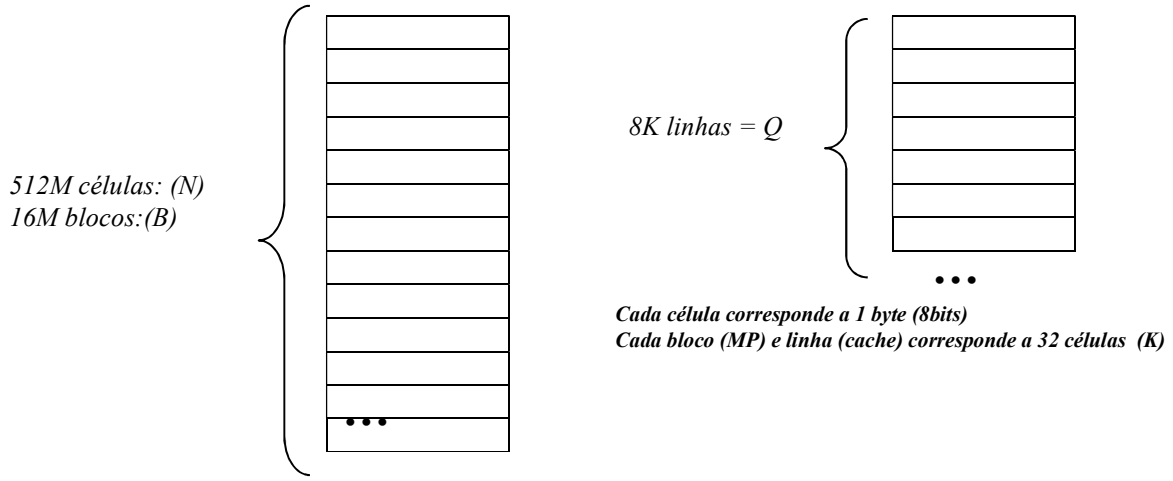
N = 512 Mcélulas e K = 32 células / blocos o total de blocos da MP (B) será:

Total de blocos: B = N / K => B = 512Mcélulas / 32 células/bloco => B = 16M blocos

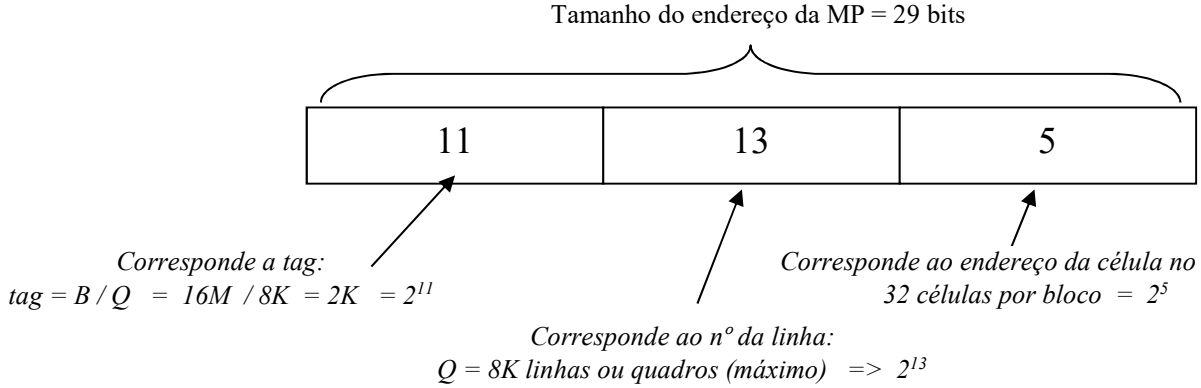
Memória Cache

OBS: O K (quantidade de células/bloco) tem de ser igual a MP.

- ⇒ *Tamanho da memória cache em blocos = 8K linhas que podem armazenar 8K blocos*
- ⇒ *Tamanho da memória cache em células = 8K blocos x 32 células/bloco = 256K células*



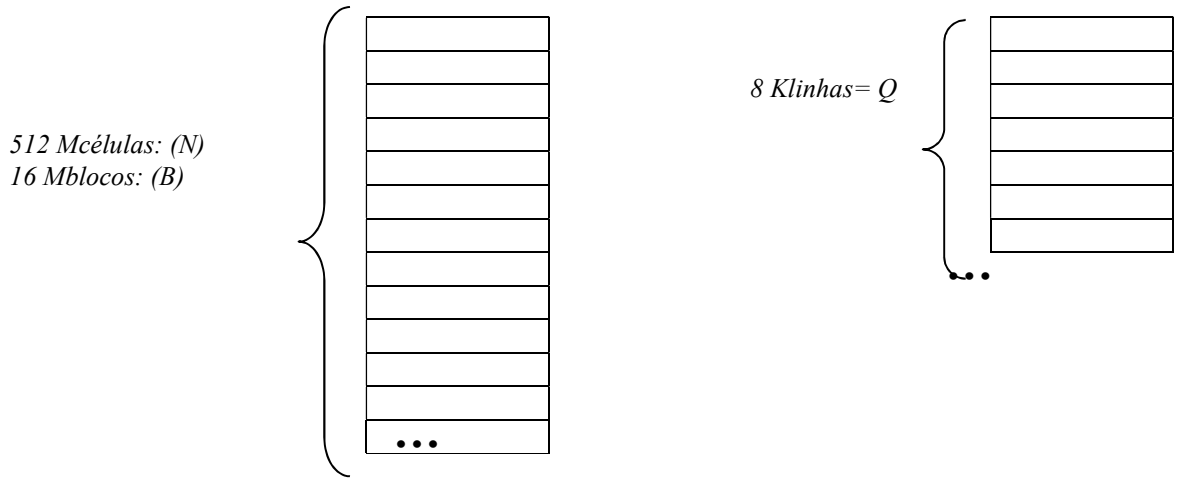
Para endereçarmos toda a MP precisamos da seguinte quantidade de bits (E)
sendo $N = 2^E$ => N = 512M células => $N = 2^{29}$ => E = 29 bits



b) Mapeamento totalmente associativo.

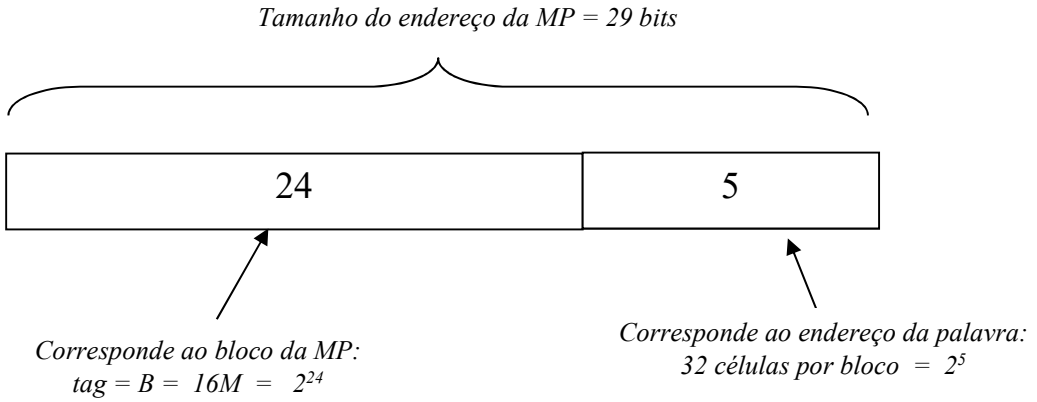
Memória Principal
=> N = 512 Mcélulas
=> K = 32 células por bloco
=> B = 16 Mblocos

Memória Cache
OBS: O K (quantidade de células/bloco) tem de ser igual a MP.
=> Q =8K linhas ou 8K blocos
=> Tamanho da memória cache = 8K células



Para endereçarmos toda a MP precisamos da seguinte quantidade de bits: E = 29 bits

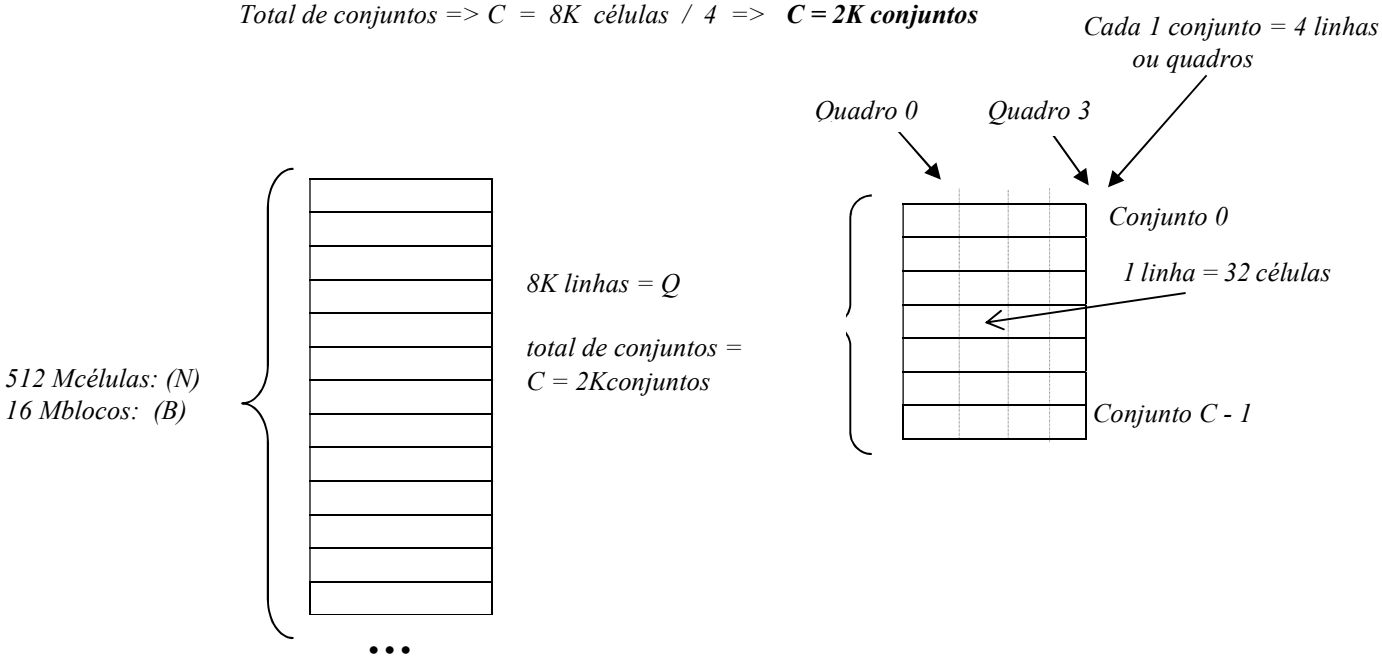
Como o bloco pode ser alocado em qualquer posição da memória cacha a tag indicará qual dos blocos da MP está alocado naquela posição da memória cachê



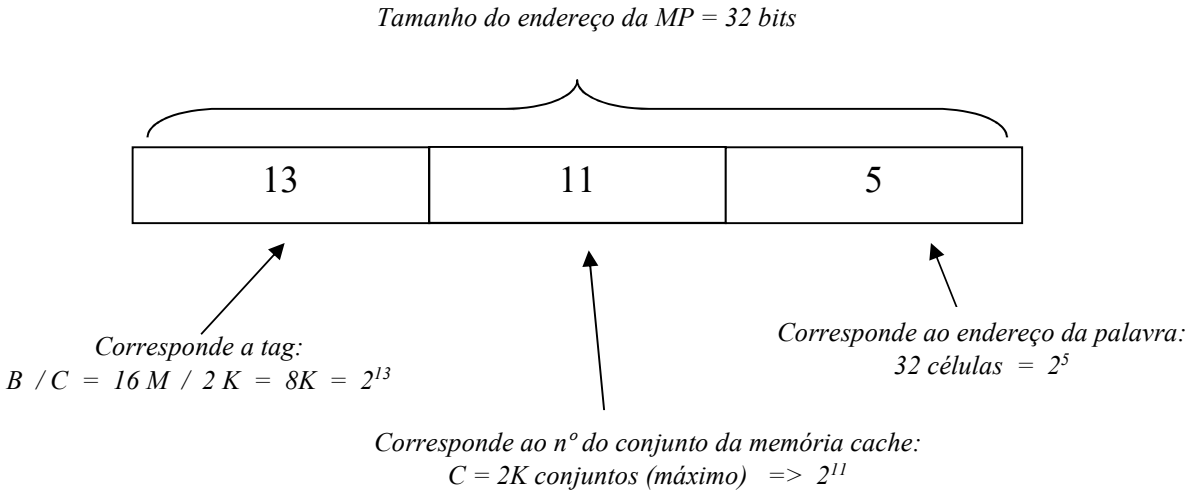
c) Mapeamento associativo por conjunto, onde cada conjunto possui quatro linhas, cada uma de um bloco.

Memória Principal
=> N = 512 Mcélulas
=> K = 32 células por bloco
=> B = 16 Mblocos

Memória Cache
OBS: O K (quantidade de células/bloco) tem de ser igual a MP.
=> Q =8 K linhas ou 8K blocos
=> Tamanho da memória cache = 8K células
=> 1 conjunto = 4 linhas (ou quadros) =>
Total de conjuntos => C = 8K células / 4 => ***C = 2K conjuntos***



Para endereçarmos toda a MP precisamos da seguinte quantidade de bits: E = 32 bits



4. (1,0) Explique em detalhes a organização hierárquica do subsistema de memória nos computadores atuais.

O subsistema de memória é interligado de forma bem estruturada seguindo uma organização hierárquica. Podemos representar essa organização, conforme as características das memórias, na forma de uma pirâmide cujos níveis são descritos a seguir.

No topo da pirâmide temos os registradores, que são pequenas unidades de memória que armazenam dados dentro do núcleo da UCP. Os registradores são dispositivos de maior velocidade com tempo de acesso em torno de 1 ciclo de memória, possui a menor capacidade de armazenamento e também o menor tempo de armazenamento.

Em um nível abaixo temos a memória cache, cuja função é acelerar a velocidade de transferência das informações entre UCP e a memória principal (MP), aumentando, assim, o desempenho do sistema. A UCP procura informações primeiro na cache e caso não as encontre, estas serão transferidas da MP para a cache. A cache possui tempo de acesso menor que a da MP, embora seja superior aos dos registradores. A capacidade da cache é bem inferior à da MP, mas possui um tamanho capaz de armazenar uma apreciável quantidade de informações. O tempo de permanência do dado é menor do que o tempo de duração do programa a que pertence.

Abaixo da memória cache temos a memória básica de um sistema de computação, que é a memória principal (MP). A MP armazena e disponibiliza o programa (e seus dados) à UCP para que este, durante a execução, busque instrução a instrução. A MP é mais lenta que a cache e mais rápida que a memória secundária, possui capacidade bem superior à da cache e os dados ou instruções permanecem na MP enquanto durar a execução do programa.

Finalmente, na base da pirâmide temos a memória secundária (MS), memória auxiliar ou memória de massa, que fornece garantia de armazenamento permanente aso dados e programas dos usuários, mesmo com a perda de energia. Alguns dispositivos são ligados ao barramento de caráter permanente, como o disco rígido, e outros são conectados conforme a necessidade do usuário, como os cartões de memória, CD-ROMs e pen-drives. A MS é a memória mais lenta quando comparada as dos demais níveis, mas possuem a maior capacidade de armazenamento.

5. (1,5) Escreva um programa que utilize as instruções de linguagem de montagem apresentadas na aula 4 para executar o seguinte procedimento. O conteúdo da memória cujo endereço é 10 é lido e verifica-se se o seu valor é menor que 0. Caso seu valor seja menor que 0, o conteúdo de memória cujo endereço é 30 é adicionado ao conteúdo de memória cujo endereço é 10 e o resultado é armazenado no endereço 10. Caso contrário, o conteúdo de memória cujo endereço é 20 é multiplicado por 2 e o resultado é armazenado no endereço 30.
Além de apresentar seu programa escrito em linguagem de montagem, apresente também o programa traduzido para linguagem de máquina.

Obs: neste gabarito os endereços 10,20,30 mencionados no título da questão foram considerados em hexadecimal, entretanto, soluções onde os mesmos endereços tenham sido entendidos como sendo na base 10, também serão considerados quando na correção da AD.

Endereço (hexa)	***Instrução***	***Descrição***	***Linguagem Máquina (bin / hexa)***
00	*LDA 10*	*ACC <- (10)*	*(0001 0001 0000 / 110)*
01	*JN 06*	*se ACC < 0, CI <- 06*	*(0111 0000 0110 / 706)*
02	*LDA 20*	*ACC <- (20)*	*(0001 0010 0000 / 120)*
03	*ADD 20*	*ACC <- ACC + (20)*	*(0011 0010 0000 / 320)*
04	*STR 30*	*(30) <- ACC*	*(0010 0011 0000 / 230)*
05	*HLT*	*Encerra Procedimento*	*(0000 0000 0000 / 000)*
06	*LDA 30*	*ACC <- (30)*	*(0001 0011 0000 / 130)*
07	*ADD 10*	*ACC <- ACC + (10)*	*(0011 0001 0000 / 310)*
08	*STR 10*	*(10) <- ACC*	*(0010 0001 0000 / 210)*
09	*HLT*	*Encerra Procedimento*	*(0000 0000 0000 / 000)*

6. (1,5) Considere uma máquina cujo relógio possui uma frequência de 2,5 GHZ e um programa no qual são executadas 400 instruções desta máquina.

   a) Calcule o tempo de UCP utilizado para executar este programa, considerando que cada instrução é executada em dois ciclos de relógio e a execução de uma instrução só se inicia quando a execução da instrução anterior é finalizada.

   2,5GHz = 2.500.000.000 Hz
   Tempo de um ciclo de relógio = 1/2.500.000.000 = 0,000 000 000 4 seg ou 0,4ns (nanosegundos)
   Tempo de execução de 1 instrução = 2 ciclos de relógio = 0,8ns
   400 instruções ***executadas sequencialmente*** *= 400 × 0,8ns =* ***320ns***

   b) Considere que essa máquina utilize um pipeline de 4 estágios, todos de igual duração. Calcule o tempo máximo que o estágio deve durar para que o tempo de execução do programa seja menor do que o tempo calculado no item anterior

   Tempo total para execução das 400 instruções deverá ser < ***320ns*** *(tempo do item a)*
   Consideremos t como sendo o tempo para execução de um estágio
   Tempo para execução de uma instrução = 4 estágios × t = 4t
   Tempo para execução das demais em ***pipeline*** *= 399 × t = 399t*
   Tempo para execução das 400 instruções = 4t (primeira instrução) + 399t (para as demais) = 403t

   O tempo total para execução do 2º. Caso < tempo total execução do 1º. Caso =>
   403 t < 320ns => t < 320/403 ***=> t < 0,794ns ou*** t_{max} ***= 0,794ns*** *(tempo para um estágio)*

7. (1,5) Faça uma pesquisa no livro "Arquitetura e Organização de Computadores" de William Stallings e descreva os métodos utilizados para lidar com desvios condicionais em arquiteturas que utilizam pipeline.

   O Desvio condicional consiste no principal impedimento para um fluxo constante de instruções em um pipeline de instruções. Diversas abordagens são adotadas para lidar com o desvio condicional, entre elas:

   Múltiplos fluxos: *Consiste em duplicar os estágios iniciais da pipeline para permitir a busca de ambas as instruções, usando assim dois fluxos de instruções. Esta abordagem apresenta problemas como o atraso à contenção de acesso a registradores e à memória, além da entrada de instruções adicionais na pipeline antes que seja tomada a decisão sobre o desvio original.*

   Busca antecipada da instrução-alvo de desvio*. Consiste em buscar antecipadamente tanto a instrução-alvo de desvio quanto a instrução consecutiva ao desvio, no instante em que uma instrução de desvio-condicional é reconhecida. A instrução-alvo é armazenada em um registrador, até que a instrução de desvio seja executada. Seja o desvio tomado ou não, a próxima instrução a ser executada terá sido buscada antecipadamente.*

   Memória de laço de repetição: *Consiste em usar uma pequena memória de alta velocidade, mantida pelo estágio de busca de instrução da pipeline, que é usada para manter as* ***n*** *instruções buscadas mais recentemente, em sequência. Na ocorrência do desvio, será verificado se a instrução-alvo está presente nesta memória.*

   Previsão de desvio*: Consiste em várias técnicas que podem ser usadas para prever se um desvio será tomado ou não, entre as mais comuns estão as seguintes.*

   1) *Prever que o desvio nunca será tomado: esta previsão não depende do histórico de execução de instruções até o momento, não supõe que o desvio não seja tomado e continua a buscar instruções na sequência em que ocorrem no programa.*
   2) *Prever que o desvio sempre será tomado: conforme a anterior diferindo que o desvio será tomado, buscando sempre as próximas instruções a partir do endereço-alvo do desvio.*
   3) *Prever se o desvio será tomado ou não conforme o código de operação: em casos de*

determinados códigos de operação, o processador supõe que o desvio sempre é tomado; em outros, que o desvio nunca é tomado, estudo feito com esta técnica apresenta taxas de sucesso superiores a 75%.

4) *Prever o desvio com base em chaves de desvio tomado e de desvio não tomado: consiste em uma estratégia dinâmica onde um ou mais bits podem estar associados a cada instrução de desvio condicional, usados para refletir a história recente da execução da instrução. Bits estes chamados de chaves de desvio tomado ou de desvio tomado ou de desvio não tomado.*
5) *Prever o desvio com base em uma tabela de histórico de desvios: Esta técnica mantém uma pequena tabela de histórico de instruções de desvio executadas mais recentemente, incluindo um ou mais bits de entrada. O processador pode endereçar essa tabela de maneira associativa, tal como uma memória cache, ou usar bits de mais baixa ordem do endereço das instruções de desvio. A tabela de histórico de desvios é uma pequena memória cache associada ao estágio de busca de instrução da pipeline.*

***Atraso de desvio**: técnica para reordenar automaticamente as instruções de um programa, de modo que instruções de desvio ocorram mais tarde do que ocorrem de fato na seqüência especificada.*

8. (1,5) Considere a máquina apresentada na aula 4. Descreva detalhadamente (do mesmo modo que é apresentado na aula 4) como é realizada a execução das seguintes instruções:

a) STA 300

1. *RI <- (CI)*
2. *CI <- CI + 1*
3. *Decodificação do código de operação*
    a. *Recebe os bits do código de operação*
    b. *Produz sinais para a execução da operação de escrita*
4. *A UC emite sinais para que o valor do campo operando = 300 seja transferido para o REM*
5. *Conteúdo do Acumulador (ACC) é transferido para o RDM (RDM ← ACC)*
6. *A UC ativa a linha WRITE do barramento de controle*
7. *O REM passa o conteúdo para o barramento de endereços*
8. *O RDM passa o conteúdo para o barramento de dados*
9. *A memória grava o dado recebido pelo barramento de dados no endereço enviado através do barramento de endereços*

b) SUB 40

1. *RI ← (CI) , ou seja, RI ← recebe a instrução contida no endereço contido no CI*
2. *CI ← CI + 1*
3. *Decodificação do código de operação*
    a. *recebe os bits do código de operação*
    b. produz sinais para a execução da operação de subtração
4. *Execução da operação*
    a. *A UC emite sinais para que o valor do campo operando (40) seja transferido para a REM*
    b. *A UC emite sinais para que o valor contido no REM seja transferido para o barram*ento de endereços
    c. *A UC ativa a linha READ do barramento de controle*
    d. *Conteúdo da posição da memória, contido no barramento de endereços (40) , é transferido através do barramento de dados para o RDM*
5. *UC emite sinais para transferir conteúdo do acumulador para UAL(1), (UAL ← ACC) - 1o. termo*
    a. *O conteúdo do RDM é transferido para o registrador acumulador (ACC ← RDM)*
6. *UC emite sinais para transferir conteúdo do acumulador para UAL(2). (UAL ← ACC) - 2o. termo*
    a. *A UC emite sinais para a UAL executar da operação se subtração*
    b. *A UC emite sinais para a UAL liberar resultado e armazenar no acumulador (ACC ← UAL)*

c) JN 100

a) RI <- Instrução lida
b) CI <- CI + 1
c) Decodificação do código de operação
- recebe os bits do código de operação
- produz sinais para a execução da operação de salto condicional
d) UC emite sinal para transferir conteúdo acumulador para UAL (UAL <- ACC)
e) Executa operação de comparação
e.1) Resultado = verdadeiro, isto é, ACC < 0
CI <- Operando (CI <- 100)
f) Inicia o procedimento de leitura da instrução contida no endereço que consta em CI