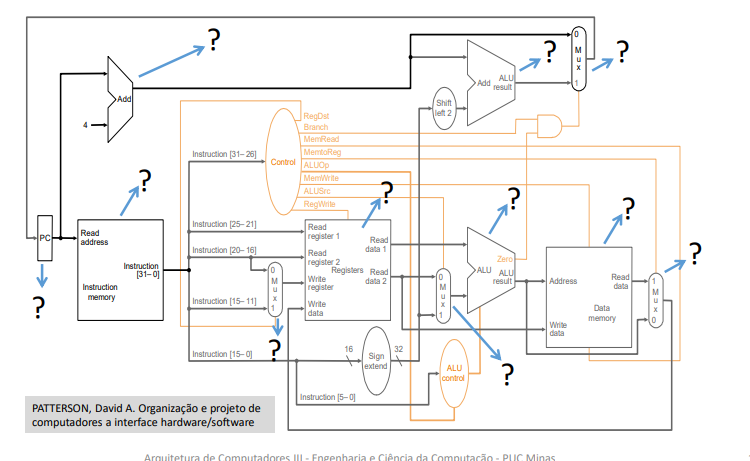
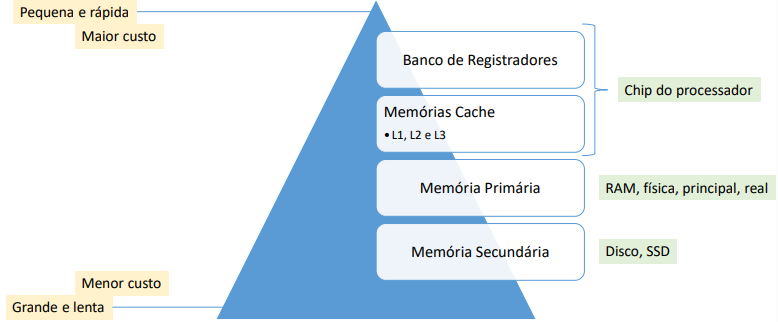
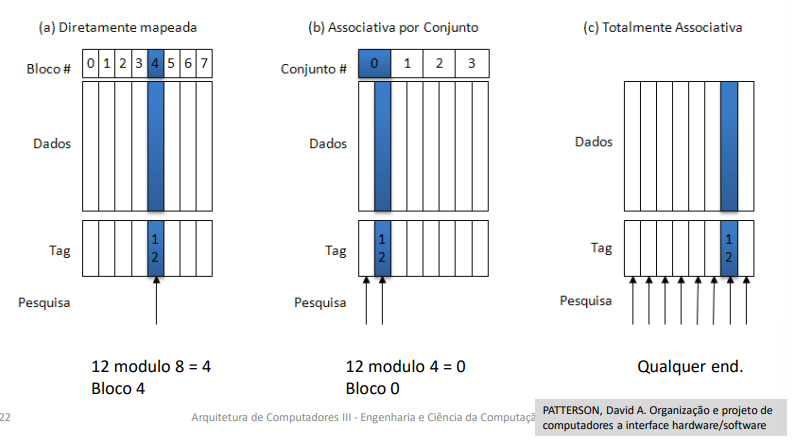
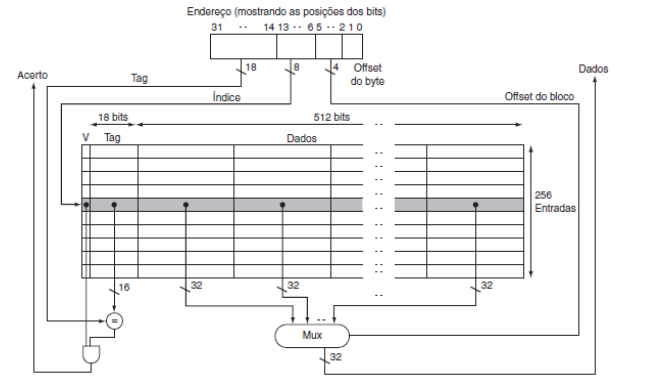
19/02/2024

* Memórias cache estão dentro do processador
* Teoricamente é tudo memória , porém não chamamos assim
* Hierarquia de memória:
  + Queremos que seja pequena rápida e menor custo – Quanto menor mais rápido e mais caro
  + Memória cache:
    - Níveis: L1,L2,L3 – L1 menor que L2 , que é menor que L3
    - **Cópia da principal,** se não tiver na cache, vai pra principal
    - Politica de substituição: para ver quem fica e quem sai da cache
  + Processador trabalha em memória principal, usando a memória cache tbm.
  + Cada nível tem cópia do nível inferior
  + Tipos memória:
    - RAM – Random access
    - ROM – Read Only
    - PROM – ROM programável
      * EPROM – apagável com radiação ultravioleta
      * EEPROM – apagável por sinais elétricos
    - Registradores
  + Célula RAM estática x dinâmica:
    - SRAM - Estática 🡪 Capaz de manter os bits de dados armazenados apenas enquanto a fonte de alimentação estiver conectada ao circuito. Equivalente ao flip-flop. Mais cara
    - DRAM - Dinâmica 🡪 São mais simples e menos área de chip. Uso de transistores e capacitores. Mais lentas. Demandam refresh de memória para manter dados armazenados. Mais barato
* Registradores:
  + Elemento lógico utilizado para armazenar uma palavra binária de n-bits
  + Cascata de flip flop tipo D
* Se tivermos processador de apenas 32 bits estamos limitados a 4GB de RAM

21/02/2024

* Hierarquia de Memória:
  + Como é gerenciada:
    - Registradores 🡪 memoria, feito pelo compilador
    - Cache 🡪 memoria principal, feito pelo hardware, pois quem demanda já é um software
    - Memoria principal 🡪 disco, feito pelo hardware, SO e pelo programador
  + **Princípio da localidade**:
    - Espacial 🡪 se um dado é referenciado, seus vizinhos tendem a ser referenciados logo
      * Normalmente quando acessamos a primeira posição do vetor , também iremos acessar o resto
    - Temporal 🡪 um dado referenciado , tende a ser referenciado novamente
    - Como explorar o princípio de localidade numa hierarquia de memória?
    - Localidade Temporal => Mantenha itens de dados mais recentemente acessados nos níveis da hierarquia mais próximos do processador
    - Localidade Espacial => Mova blocos de palavras contíguas para os níveis da hierarquia mais próximos do processador
  + Organizações de Memória Cache:
    - O processador gera o endereço de memoria e envia para o cache
    - O cache deve:
      * Verificar se tem copia da posição de memória correspondente
      * Se tem, encontra posição da cache
      * Se não, traz contrudo da memoria principal e copia para cache
    - Mapeamento entre endereços de memoria principal e cache 🡪 executado em hardware
    - Estratégias de mapeamento da cache:
      * mapeamento completamente associativo – usado para cache – olha para memoria como um grande conjunto
      * mapeamento direto – 1 way
      *  mapeamento set-associativo
  + Cache:
    - Características:
    - Pouco espaço de armazenamento
    - Alto custo financeiro
    - Baixo tempo de acesso
    - Conceitos
    - Palavra: conjunto de um ou mais bytes.
    - Bloco: conjunto de uma ou mais palavras (unidade da cache)
    - Bit de Válido: indica se o dado ou bloco está válido
    - Tag ou rótulo: parte do endereço de uma palavra na memória principal
    - Slot ou linha: cada linha de uma cache, que pode armazenar um ou mais blocos dependendo da organização da cachem, onde guarda os dados , tudo é chamdo de via.
    - Comparador: compara a tag de um endereço de uma palavra, com as tags dos endereços armazenados no cache
    - Temos um comparador para cada via

26/02/2024

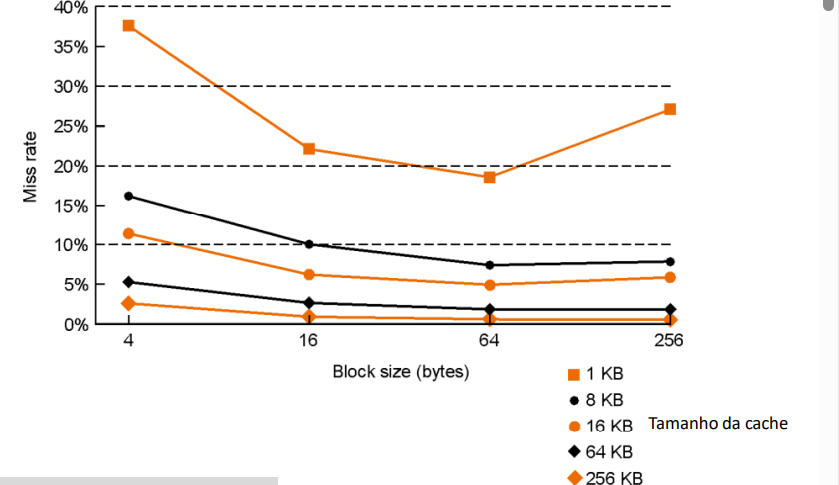
* Tamanho da linha tirando vantagem de localidade espacial, ou seja , preciso do próximo endereço já se encontra na cache
* Offset de bloco 🡪 Para definir qual palavra vc quer (4 bits)
* Offset do byte 🡪 (2 bits – pode ser diferente)

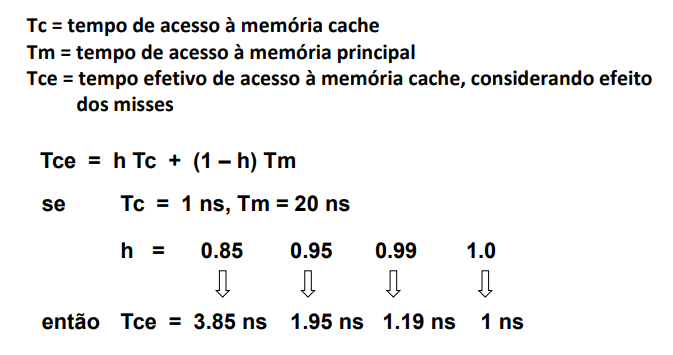
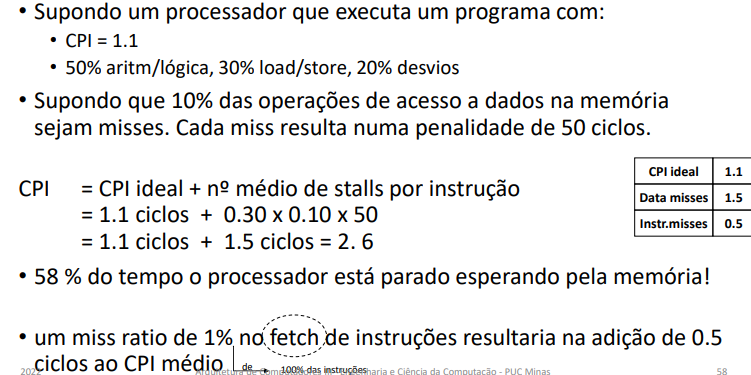
Se tivermos 8 bits de índice temos 2^8 conjuntos – no completamente associativo não temos índice. pois temos apenas um bloco

* Tamanho do bloco = tamanho da linha
* O que normalmente temos é uma diferença na organização da cache e armazenamento
* Quando temos uma associativa por conjunto 4 vias:
  + Armazenamento flexível – menos conflito de dados
  + Cache complexa e maior
  + Gasta mais tempo que mapeamento direto para achar dado
* Mapeamento direto:
  + Pouco localizado temporal, pois toda hora estamos substituindo o dado
  + Não é uma boa opção para cache
* Mapeamento completamente associativo:
  + Armazenamento muito flexível
  + Gasta muito tempo para achar dados
* Quando o cache estiver sem espaço, qual bloco será substituído:
  + Mapeamento direto 🡪 o bloco que estiver no slot
  + Outros 🡪 Política de substituição:
    - LRU 🡪 bloco menos recentemente utilizado. Endereço vai para frente da lista
    - LFU 🡪 substituir o bloco menos frequentemente utilizado. Contador incrementa quando é acessado
    - FIFO 🡪 substituir o primeiro que entrou na cache
    - Aleatório
* Como manter a coerência com a memória principal?
  + Write – through 🡪 escreve na cache e na memória principal
  + Write-back 🡪 Somente em cache, porém quando atualizar escreve na principal

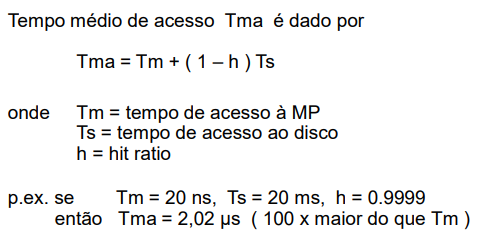
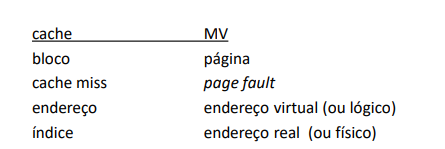
28/02/2024

* Cache hit e cache miss:
  + Miss 🡪 falta na cache, ou seja, o dado procurado ainda tem que ser buscado não memoria principal
  + Hit 🡪 tempo de acesso ao nível superior que consiste em tempo de acesso + tempo para determinar hit/miss
  + Métrica: taxa de acerto = hit / acessos
  + Tipos:
    - Compulsório 🡪 primeiro acesso
    - Conflito 🡪 as linhas de memória acessando o mesmo conjunto da cache conjunto associativa ou mesma linha da cache com mapeamento direto
    - Capacidade 🡪 cache não pode conter todas as linhas acessadas pelo programa. No mapeamento direto não faz sentido
    - Invalidação 🡪 outro processo atualiza memória. Bit de válido
  + OBS: Na hora de ver localidade: Usei no passado vou usar no futuro? Espacial
  + Se existe na cache e não na memória principal, esse bloco deve ser invalidado
* Tamanho da linha vs miss ratio:
  + À medida que aumento o tamanho do bloco há uma queda na taxa de miss, **assim explorando a temporalidade espacial**. Entretanto, ao chegar em 256 KB existe um aumento na miss rate, pois ao aumentar o bloco **diminuímos os conjuntos, assim gerando mais conflitos, explorando a temporalidade temporal**.
  + tempo médio de acesso = Hit Time x (1 - Miss Ratio) + Miss Penalty x Miss Ratio

04/03/2024

* Quantos bits tem a cache no total?
  + Cache com mapeamento direto , com 64 kb de dados, linha com balata de 32 bits e endereços de 32 bits
    - Em uma memória 64kb 🡪 16k palavras
    - Como uma linha com UMA palavra - Se encontramos todas as palavras encontramos todos os blocos
    - Assim: 2^14 x (32 + 32 -14 -2 +1) – 32 – 14 – 2 = TAG
    - TAG + índice(expoente das palavras) + validade \* blocos
  + Supondo cache com mapeamento direto, com 16 kB de dados, blocos de 4 palavras, sendo cada palavra de 32 bits e endereços de 32 bits
    - Quando tivermos mais palavras por bloco falta o offset do bloco
    - TAG = 32 – 10 – 2 – 2
    - 2^10 \* (128 + 18 + 1) 147 \* 2^10 = 147kbits / 8 – 18.4 kB
  + E se for conjunto associativo de 2 vias?
    - 2^12 palavras / 2^2 bloco de 4 palavras – 2^10 blocos
    - Entretanto, agora , pega a quantidade de linhas(blocos) e divide por quantidade de vias – 2^10 / 2 🡪 2^9
    - TAG = 32 – 9 – 2 – 2
    - 2^9 \* (128+19+1) + (128+19+1)
  + E o completamente associativo?
    - Indice zerado
    - Tag maior
* Impacto no desempenho:

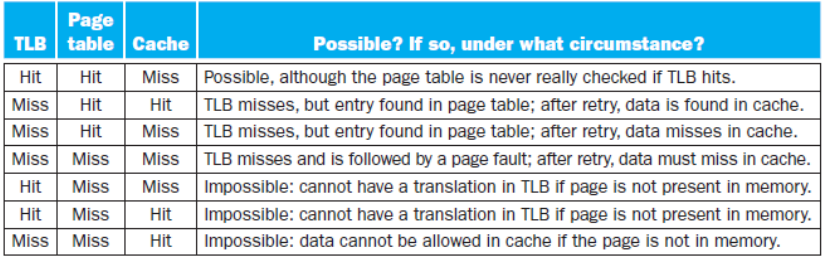
06/03/2024 – Memória Virtual

* Mempria principal:
  + Capacidade limitada
  + Tempo de acesso entre 10 e 20 ns
* Memoria secundaria:
  + Capacidade limitada, mas muito maior
  + Tempo de latência entre 10 e 30 ms
* Problema:
  + Seja um compitador de 32 kbytes de memória principal
  + Como podemos:
    - Rodar programas que usam mais que 32 kbytes
    - Permitir que vários usuários usem o computador
    - Executar vários programas ao mesmo tempo
* Solução:
  + Memória virtual 🡪 técnica que nos permite ver a memoria principal como uma cache de grande capacidade de armazenamento
  + mecanismo automático de gerência de memória, que traz automaticamente para a MP os blocos de informação (do disco) necessários
  + usuário tem a impressão de trabalhar com uma memória única, do tamanho da memória secundária, mas com tempo de acesso próximo do tempo da MP
  + Tempo de acesso:
* Por que MV é diferente de cache:
  + Miss penalty é MUITO maior.
  + Logo:
  + miss ratio precisa ser bem menor do que em cache
  + alta penalidade do miss => necessário buscar blocos maiores em disco
  + princípio de localidade opera sobre blocos maiores de dados ou instruções e leva a hit ratios bem mais elevados
  + Mapeamento totalmente associativo das páginas
  + misses são tratados por software (há tempo disponivel)
  + técnica de escrita write-through não é uma opção. Usa-se write-back.
* Termonologia:
  + Endereço virtual (lógivo) 🡪 gerado pelo programa
  + Endereço real(físico) 🡪 endereço na memoria principal
* Unidade de Gerenciamento de Memroia:
  + MMU(Memory Management Unit)
    - Gerencia de hierarquia de memoria
    - Proteção de memoria
    - Usualmente integrada dentro do microprocessador
    - Faz o mapeamento do virtual para real

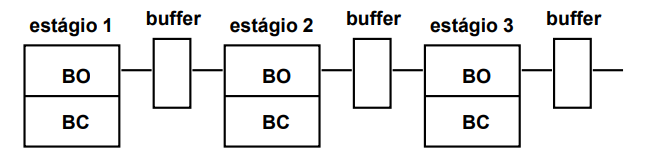
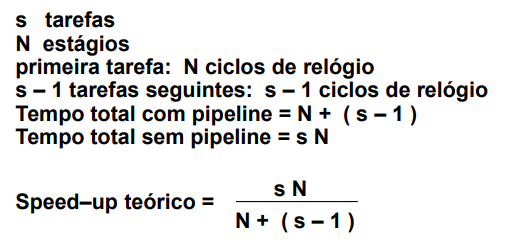
11/03/2024

* Paginação:
  + Mecanismo simples para tradução de endereções virtuais em reais e para gerenciamento do espaço da memoria
  + Espaços de memória real e virtual divididos em blocos chamados de páginas:
    - 4 kbytes a 16 kbytes
  + Endereços virtuais e reais divididos em dois campos:
    - Endereço página
    - Endereço linha (ou bloco)
  + É apenas uma função de mapeamento dos endereços virtuais para reais na memória principal (Quadro ou frames)
  + Página tem o mesmo tamanho seja real ou virtual
  + Page fault ocorre quando a página virtual não está na memória principal 🡪 mapeamento completamente associativo, mais eficiente, ajuda a diminuir alta penalidade dos page faults
  + Page tables:
    - Guardam a correspondência entre páginas virtuais e reais
    - Permitem translação de endereços
    - Estrutura de dados que guarda essa correspondência
    - Cada processo tem a sua
    - S O responsável pela alocação física para o espaço virtual
    - Hardware tem um registrador para o início da tabela – SO atualiza ao iniciar um novo processo
    - Cache da tabela de páginas🡪 algoritmo de substituição para selecionar página da memória principal
* Substituição de páginas:
  + FIFO – First in first out
  + Memória com 3 frames
  + LRU – Last recently used
  + LFU 🡪 FIFO como critério de desempate
  + OPT 🡪 substitui a página que não será usada por um tempo mais longo (prever futuro)

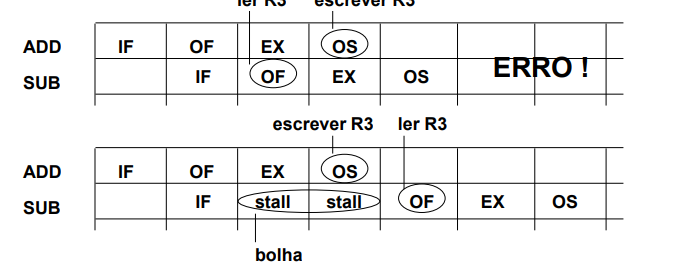
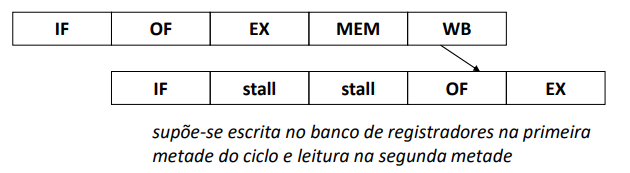
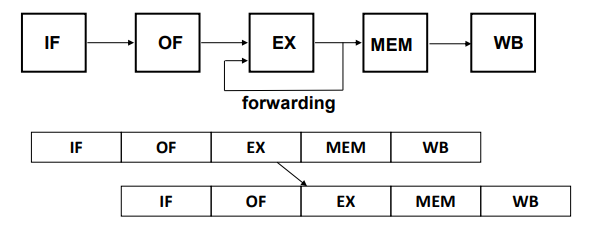
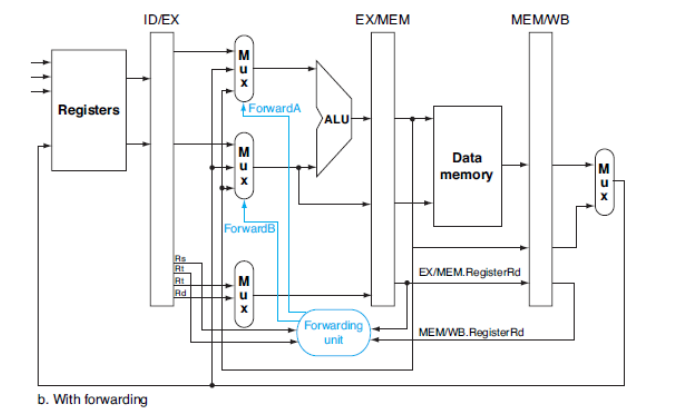
13/03/2024

* Translation look-aside buffer (TLB):
  + Como se fosse uma cache da tabela de página
  + Tamanho excessivo da main memory translation table
  + Se ficar na memoria principal 🡪 dois acessos á memoria a cada cache miss
  + Implementado em hardware
  + Traduz endereços virtuais para reais
  + Páginas do working set 🡪
  + Cache da MMTT
  + Completamente associativo
  + Acerto 🡪 leva para memória primaria
  + Miss 🡪 leva para tabela de páginas
* TBL para endereço real para acessar a cache
* Cache física x virtual
  + Física 🡪 precisamos de um endereço real/ físico para acessar a cache
  + Virtual 🡪 usaremos um endereço virtual para acessar a cache – mais rápido, porém iremos precisar de limpar cache na troca de contexto
* Disco:
  + Tempos:
    - Busca
    - Latência rotacional
    - Tempo de transferência
    - Disco 🡪 tempo de busca + latência + transferência

03/04/2024 Pipeline

* Paralelismo 🡪 precisamos de múltiplos recursos
  + Exemplo supermercado e lavanderia
* Pipeline:
  + O conceito pode ser aplicado em varias áreas
  + Objetivo: Aumento de desempenho
    - Divisão de tarefa em N estágios
    - N tarefas executadas e paralelo, em cada estágio
  + Pipe line cheio é quando tem tarefas em todos os estágios
  + Dependência eh um problema:
    - Solução: segurar a instrução até o WB da que depende, porém ira gerar estágios ocioso 🡪 desempenho péssimo
  + Existe um bloco operacional e de controle para cada estágio, necessidade de buffer entre estágios
  + Pipeline aritmético:
    - EX: soma em ponto flutuante executada em 4 estagios
    - Clock será o estagio mais demorado
    - O primeiro estagio leva todo tempo e os seguintes não
  + Desempenho:
  + Problemas no desempenho:
    - Como dividir todas instruções num mesmo conjunto de estágios?
    - Como obter estágios com tempo similares?
    - Conflito de memoria
    - Dependência de dados
    - Instruções de desvio
  + Se o desvio ocorre , pipeline precisa ser esvaziado
  + Técnicas de tratamento de desvios condicionais:
    - Executar os dois caminhos do desvio – buffers pararelos de instruç~eos
    - Prever sentido do desvio: predição estática e dinâmica
    - Elimar o problema – Delayed Branch

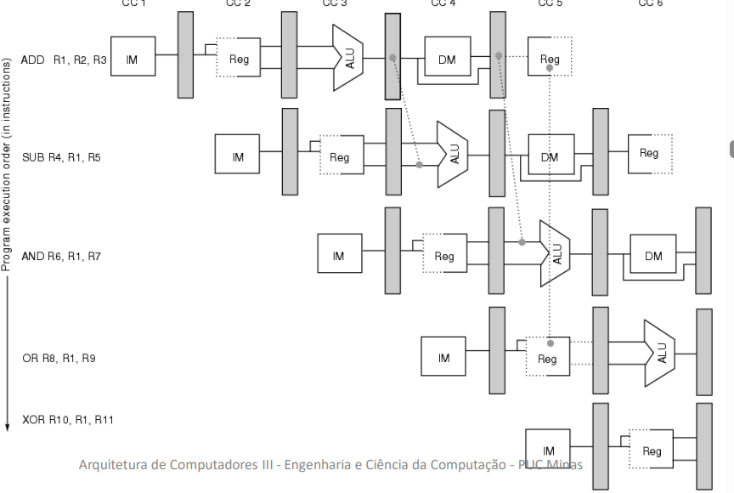
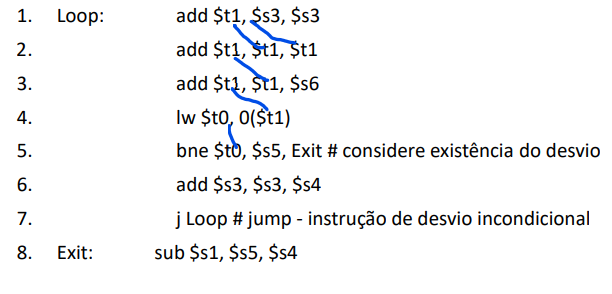
08/04/2024 – Pipeline segunda parte

* Problema: instruções consecutivas podem fazer acesso aos mesmos operandos – pode ter dependências
* Tipos de dependências:
  + Verdadeira
    - Instrução 2 depende de R3 calculado pela instrução 1
    - Pipeline precisa ser parado durantes certo número de ciclos
    - Leitura após escrita
  + Antidepedencia
    - Escrita após leitura
    - não é um problema em pipelines onde a ordem de execução das instruções é mantida
  + De saída
    - Escrita após escrita
* Pipeline interlock:
  + Bit de tag nos registradores
  + Manter sequencia correta de leituras e escritas
  + Tag 0 🡪 leitura indisponível
* Adiantamento de dados:
  + Divisão do ciclo em etapas. Ex WB e OF no mesmo tempo
* Adiantamento de dados 🡪 Caminho interno dentro do pipeline entre saida e entrada – nem sempre é possível fazer sem bolha

Terceira parte:

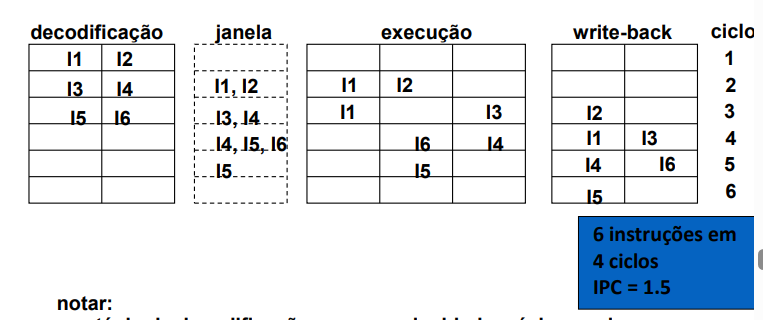
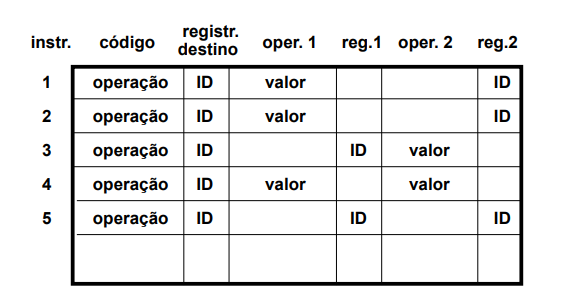
* Adiantamento de dados com load gera bolha , se load escreve

10/04/2024

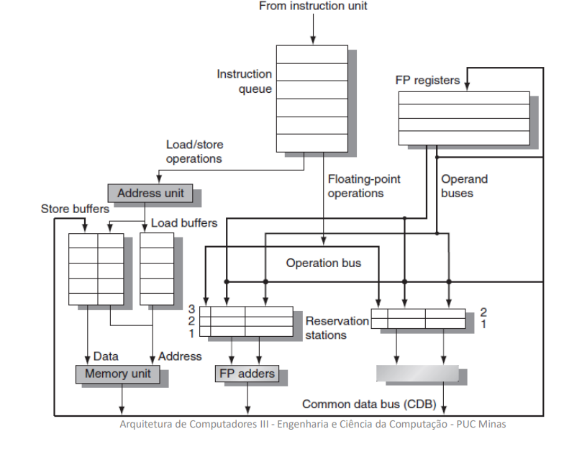
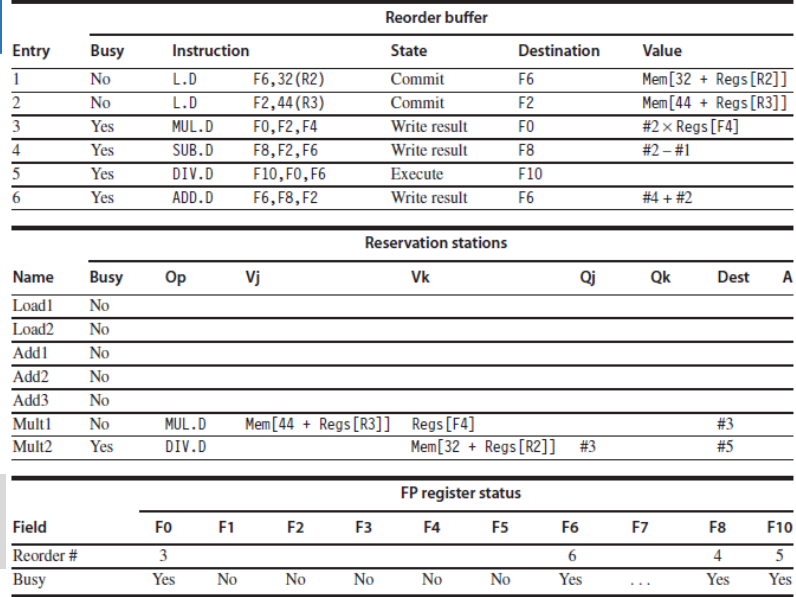
* Como gerar bolha:
  + Sinais de controle 0 – vai acontecer nada
  + Hazard detection unit – gera sinais de controle
* Se tiver uma linha tracejada pra esquerda vai dar erro , por causa da dependência
* Se tiver linha tracejada , iremos precisar do adiantamento de dados e gerar uma bolha. **Sem adiantamento seria necessário 2 bolhas se tiver escrita e leitura no mesmo ciclio , se não sera 3**
* Instruções de desvio:
  + Beq esta na 3 ciclo – então 2 instruções serão descartadas 🡪 **depende de quando for a tomada de decisão**
  + IF flush limpa barreira e serve para o descarte de instruções
* Exercício 1:
  + 1 – Marcar dependências verdadeiras:
  + Quantos ciclos gastam?
    - 5 ciclos 0 bolhas – 1 instrução
    - 1 ciclo 0 bolhas
    - 1 ciclo 0 bolhas
    - 1 ciclo 0 bolhas 🡪 como add é antes do lw não tem bolha
    - 1 ciclo 1 bolha
    - Não executaremos ADD E J 🡪 descartaremos ambos. Duas bolhas
    - 0 ciclo 1 bolha 🡪 ADD
    - 0 ciclo 1 bolha 🡪 J
    - 1 ciclo 0 bolha

Ciclos 13 – Qtd-Instrução: 6 – CPI 13/6

29/04/2024

* Dispacho fora de ordem e terminação fora de ordem:
  + Decodificação de instruções é congelada quando instrução cria.
  + Processador não tem capacidade de look ahead além da instrução que casou problema
* Janela de decodificação:
  + Não para de alimentar janela de instruções
  + Cada coluna é uma unidade funcional
* Janela de instruções centralizada:
  + Buffer que armazena todas as instruções pendentes pra execução

06/05/2024

* Erro predição de desvio -- superescalar:
  + Limpa o buffer a partir da instrução de desvio – ordenado
* Algoritmo de Tomasulo:
  + Melhoria do pipeline super escalar
  + Estação de reserva 🡪 podemos usar seu nome para renomear
    - Economia de armazenamento
    - FP register = Registrador de ponto flutuante
    - É possível ter sem o buffer de reordenamento
    - QJ e QK 🡪 qual momento que vai produzir o resultado
    - Tabela de estações não tem ordem
    - Vj e Vk 🡪 valores do operandos fonte
    - A 🡪 Valor imediato e dps endereço calculado
    - Register status – Banco de registradores 🡪 De qual estação vem o valor daquele registrador
    - Se tiver instrução de desvio devemos apagar todo buffer. Pois teremos que ler de todo jeito
* Arquitetura de Suporte Multithreading
  + Não é programação multi thread
  + Se tivermos **um** núcleo com suporte multi thread – podemos usar multi thread
  + Tipos:
    - Vertical – IMT/BMT
      * Não possui execução simultânea
      * Vantagem: Podemos fazer em um escalar – Não precisamos preocupar com dependência
    - Horizontal – SMT
      * Execução simultânea
  + Arquitetura
  + Diagrama

    Descrição gerada automaticamenteDiferenças:
    - Sem IMT – vê apenas um thread
    - Com IMT – ve 4 threads – pc e register banks
  + Para processador um thread e um processo é a mesma coisa, ou seja, seu processo e escalonamento é o mesmo

13/05/2024

* Suporte a múltiplas threads:
  + CPI 🡪 IPC ( superescalar)
  + Vazão de instruções(superescalar) 🡪 Vazão de threads(SMT)
  + Olusão de mais de um núcleo de processador
  + Não existe esvaziamento de pipeline comum no BMT
  + Não há atraso na execução de threads – comum no IMT e BMT
  + Problemas:
    - Tamanho da arquitetura
    - Banco de registradores muito grande pra guaradar vários contextos
    - Divisão de recursos e equilíbrio de desempenho
    - Conflitos de cache sem degradação de desempenho
* IMT e BMT 🡪 entrelaçamento de threads
* Preciso de 1 ciclio para selecionar a thread

27/05/2024 – Redes em chip

* GPU:
  + Quantidade razoável de núcleos
  + Projetada para processamento gráfico
  + Operação vetorial
  + IA executada na GPU, demanda de processamento em GPU
  + Núcleo com pouca memória
* Barramento:
  + Transferência de dados entre componentes na mesma máquina ou máquinas diferentes
* Processador many core:
  + O problema está no fio para interconectar o núcleo
  + Solução:
    - Rede em chip curto
      * Barramento e chave Cross bar
      * Importante pois ela que vai dar escalabilidade
      * Composta por roteadores
      * Possui pacotes de rede
      * Trabalha com protocolo de roteamento
      * QoS
      * Tolerante a falhas

03/06/2024

* Barramento e chave cross bar não pode ser uma interconexão global – pois não conseguimos ligar todos os núcleos
* Cada repetidor serve como um replicador, revigora sinal
* Nas redes em chip cada roteador tem um arbitro – escolhe a saída
* Protocolos:
  + Garantir entrega de dados
  + Confiabilidade da rede
  + Melhor rota e desempenho