Organização de Computadores Digitais - 5954008

1. Introdução

Prof. Luiz Otavio Murta Jr

Local: Depto. de Computação e Matemática (FFCLRP/USP)

Principais Tópicos

1. Introdução

- 1.1. Organização
- 1.2. Breve História da Computação
- 1.3. Informação Digital
  - 1.3.1. Introdução
  - 1.3.2. Codificação da Informação
  - 1.3.3. Sistemas Numéricos
  - 1.3.4. Operações Aritméticas com Números Binários

1.2.9. Projeto para CPUs modernas

- Agora que já se passaram mais de duas décadas desde que as primeiras máquinas RISC foram lançadas.
- Certos princípios de projeto passaram a ser aceitos como um bom modo de projetar computadores, dado o estado atual da tecnologia de hardware.
- Se ocorrer uma importante mudança na tecnologia.
  - por exemplo, se, de repente, um novo processo de fabricação fizer o ciclo de memória ficar dez vezes mais rápido do que o tempo de ciclo da CPU.
  - todas as apostas perdem.
- Assim, os projetistas de máquinas devem estar sempre de olho nas mudanças tecnológicas que possam afetar o equilíbrio entre os componentes.
- Dito isso, há um conjunto de princípios de projeto, às vezes denominados princípios de projeto RISC, que os arquitetos de CPUs de uso geral se esforçam por seguir.

1.2.9. Projeto para CPUs modernas

- Limitações externas, como a exigência de compatibilidade com alguma arquitetura existente,
  - muitas vezes exigem uma solução de conciliação de tempos em tempos,
  - mas esses princípios são metas que a maioria dos projetistas se esforça para cumprir.
- Todas as instruções são executadas diretamente por hardware
- Todas as instruções comuns são executadas diretamente pelo hardware – não são interpretadas por microinstruções.
- Eliminar um nível de interpretação dá alta velocidade à maioria das instruções.
- No caso de computadores que executam conjuntos de instruções CISC, as instruções mais complexas podem ser subdivididas em partes separadas que então podem ser executadas como uma sequência de microinstruções.

1.2.9. Projeto para CPUs modernas

- Essa etapa extra torna a máquina mais lenta, porém, para instruções que ocorrem com menos frequência, isso pode ser aceitável.
- Computadores modernos recorrem a muitos truques para maximizar seu desempenho, entre os quais o principal é tentar iniciar o máximo possível de instruções por segundo.
- Afinal, se você puder emitir 500 milhões de instruções por segundo, terá construído um processador de 500 MIPS, não importa quanto tempo elas realmente levem para ser concluídas.
  - (MIPS quer dizer Milhões de Instruções Por Segundo. O processador MIPS recebeu esse nome como um trocadilho desse acrônimo. Oficialmente, ele significa Microprocessor without Interlocked Pipeline Stages – microprocessador sem estágios paralelos de interbloqueio.)

1.2.9. Projeto para CPUs modernas

- Esse princípio sugere que o paralelismo pode desempenhar um importante papel na melhoria do desempenho, uma vez que emitir grandes quantidades de instruções lentas em curto intervalo de tempo só é possível se várias instruções puderem ser executadas ao mesmo tempo.
- Embora as instruções sempre sejam encontradas na ordem do programa, nem sempre elas são executadas nessa mesma ordem (porque algum recurso necessário pode estar ocupado) e não precisam terminar na ordem do programa.
  - É claro que, se a instrução 1 estabelece um registrador e a instrução 2 usa esse registrador, deve-se tomar muito cuidado para garantir que a instrução 2 não leia o registrador até que ele contenha o valor correto.
- Fazer isso funcionar direito requer muito controle, mas possibilita ganhos de desempenho por executar várias instruções ao mesmo tempo.

1.2.9. Projeto para CPUs modernas

- Um limite crítico para a taxa de emissão de instruções é a decodificação de instruções individuais para determinar quais recursos elas necessitam.
- Qualquer coisa que possa ajudar nesse processo é útil. Isso inclui fazer instruções regulares, de comprimento fixo, com um pequeno número de campos.
- Quanto menor o número de formatos diferentes para as instruções, melhor.
- Um dos modos mais simples de subdividir operações em etapas separadas é requerer que os operandos para a maioria das instruções venham de registradores da CPU e a eles retornem.

1.2.9. Projeto para CPUs modernas

- A operação de movimentação de operandos da memória para registradores pode ser executada em instruções separadas.
- Uma vez que o acesso à memória pode levar um longo tempo, e que o atraso é imprevisível, o melhor é sobrepor essas instruções a outras se elas nada fizerem exceto movimentar operandos entre registradores e memória.
- Essa observação significa que somente instruções LOAD e STORE devem referenciar a memória.
- Todas as outras devem operar apenas em registradores.

1.2.10. Paralelismo no nível de instrução

- Arquitetos de computadores estão sempre se esforçando para melhorar o desempenho das máquinas que projetam.
- Fazer os chips funcionarem com maior rapidez aumentando suas velocidades de clock é um modo, mas, para cada novo projeto, há um limite para o que é possível fazer por força bruta naquele momento da História.
- Por conseguinte, grande parte dos arquitetos de computadores busca o paralelismo (fazer duas ou mais coisas ao mesmo tempo) como um meio de conseguir desempenho ainda melhor para dada velocidade de clock.
- O paralelismo tem duas formas gerais, a saber, no nível de instrução e no nível de processador.
  - Na primeira, o paralelismo é explorado dentro de instruções individuais para obter da máquina mais instruções por segundo.
  - Na última, várias CPUs trabalham juntas no mesmo problema. Cada abordagem tem seus próprios méritos.
-

1.2.10. Paralelismo no nível de instrução

- **Pipelining (paralelismo)**
- Há anos sabe-se que o processo de buscar instruções na memória é um grande gargalo na velocidade de execução da instrução.
- Para amenizar esse problema, os computadores, desde o IBM Stretch (1959), tinham a capacidade de buscar instruções na memória antecipadamente, de maneira que estivessem presentes quando necessárias.
- Essas instruções eram armazenadas em um conjunto de registradores denominado buffer de busca antecipada (ou prefetch buffer).
- Desse modo, quando necessária, uma instrução podia ser apanhada no buffer de busca antecipada, em vez de esperar pela conclusão de uma leitura da memória.

1.2.10. Paralelismo no nível de instrução

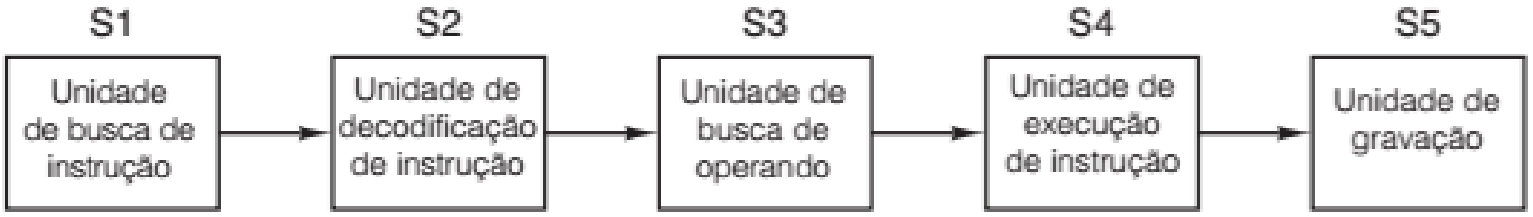
- Na verdade, a busca antecipada divide a execução da instrução em duas partes: a busca e a execução propriamente dita.
- O conceito de pipeline (paralelismo, canalização) amplia muito mais essa estratégia.
- Em vez de dividir a execução da instrução em apenas duas partes, muitas vezes ela é dividida em muitas partes:
  - uma dúzia ou mais,
  - cada uma manipulada por uma parte dedicada do hardware,
  - e todas elas podem executar em paralelo.

1.2.10. Paralelismo no nível de instrução

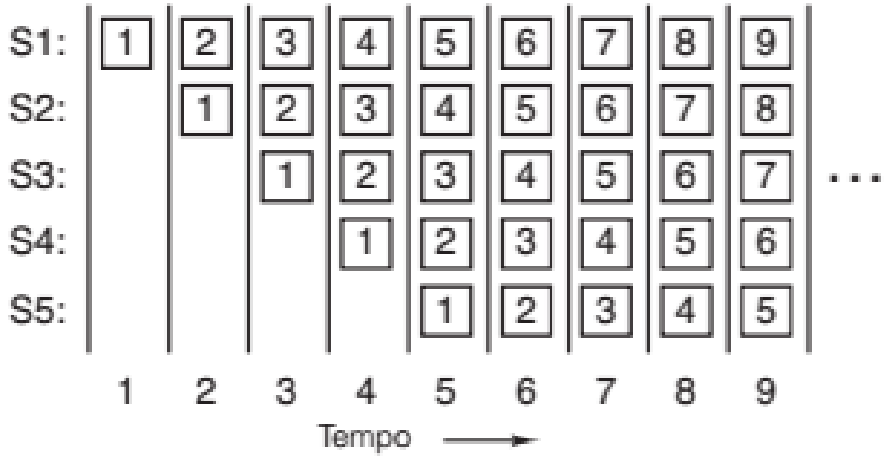
- A figura a seguir ilustra um pipeline com cinco unidades, também denominadas estágios.
- **O estágio 1** busca a instrução na memória e a coloca em um buffer até que ela seja necessária.
- **O estágio 2** decodifica a instrução, determina seu tipo e de quais operandos ela necessita.
- **O estágio 3** localiza e busca os operandos, seja nos registradores, seja na memória.
- **O estágio 4** é que realiza o trabalho de executar a instrução, normalmente fazendo os operandos passarem pelo caminho de dados.
- Por fim, o estágio 5 escreve o resultado de volta no registrador adequado.

1.2.10. Paralelismo no nível de instrução

- (a) Pipeline de cinco estágios. (b) Estado de cada estágio como uma função do tempo. São ilustrados nove ciclos de clock.



(a)



(b)

1.2.10. Paralelismo no nível de instrução

- O pipeline em função do tempo:
  - durante o ciclo de clock 1, o estágio S1 está trabalhando na instrução 1, buscando-a na memória.
  - durante o ciclo 2, o estágio S2 decodifica a instrução 1, enquanto o estágio S1 busca a instrução 2.
  - durante o ciclo 3, o estágio S3 busca os operandos para a instrução 1, o estágio S2 decodifica a instrução 2 e o estágio S1 busca a terceira instrução.
  - durante o ciclo 4, o estágio S4 executa a instrução 1, S3 busca os operandos para a instrução 2, S2 decodifica a instrução 3 e S1 busca a instrução 4.
  - durante o ciclo 5, S5 escreve (grava) o resultado da instrução 1 de volta ao registrador, enquanto os outros estágios trabalham nas instruções seguintes.
- Vamos considerar uma analogia para esclarecer melhor o conceito de pipelining.

1.2.10. Paralelismo no nível de instrução

- Imagine uma fábrica de bolos na qual a operação de produção dos bolos e a operação da embalagem para expedição são separadas.
- Suponha que o departamento de expedição tenha uma longa esteira transportadora ao longo da qual trabalham cinco funcionários (unidades de processamento).
  - A cada 10 segundos (o ciclo de clock), o funcionário 1 coloca uma embalagem de bolo vazia na esteira.
  - A caixa é transportada até o funcionário 2, que coloca um bolo dentro dela.
  - Um pouco mais tarde, a caixa chega à estação do funcionário 3, onde é fechada e selada.
  - Em seguida, prossegue até o funcionário 4, que coloca uma etiqueta na embalagem.
  - Por fim, o funcionário 5 retira a caixa da esteira e a coloca em um grande contêiner que mais tarde será despachado para um supermercado.

1.2.10. Paralelismo no nível de instrução

- Em termos gerais, esse é o modo como um pipeline de computador também funciona:
  - cada instrução (bolo) passa por diversos estágios de processamento antes de aparecer já concluída na extremidade final.
- Voltando ao nosso pipeline da Figura, suponha que o tempo de ciclo dessa máquina seja 2 ns.
- Sendo assim, uma instrução leva 10 ns para percorrer todo o caminho do pipeline de cinco estágios.
- À primeira vista, como uma instrução demora 10 ns, parece que a máquina poderia funcionar em 100 MIPS, mas, na verdade, ela funciona muito melhor do que isso.
- A cada ciclo de clock (2 ns), uma nova instrução é concluída, portanto, a velocidade real de processamento é 500 MIPS, e não 100 MIPS.

1.2.10. Paralelismo no nível de instrução

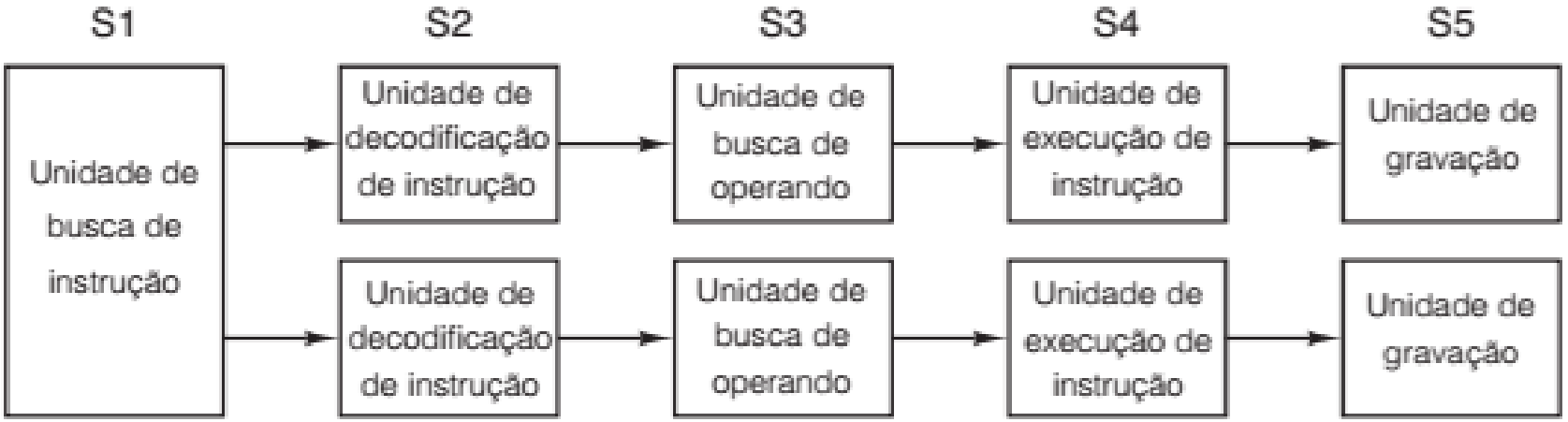
- O pipelining permite um compromisso entre latência (o tempo que demora para executar uma instrução) e largura de banda de processador (quantos MIPS a CPU tem).
- Com um tempo de ciclo de T ns e *n* estágios no pipeline, a latência é *n*T ns porque cada instrução passa por *n* estágios, cada um dos quais demora T ns.
- Visto que uma instrução é concluída a cada ciclo de clock e que há 10^9 /T ciclos de clock por segundo, o número de instruções executadas por segundo é 10^9/T.
- Por exemplo, se T = 2 ns, 500 milhões de instruções são executadas a cada segundo.
- Para obter o número de MIPS, temos de dividir a taxa de execução de instrução por 1 milhão para obter (10^9 /T)/10^6 = 1.000/T MIPS.

1.2.11. Arquiteturas superescalares

- Se um pipeline é bom, então certamente dois pipelines são ainda melhores.
- Um projeto possível para uma CPU com dois pipelines.
- Nesse caso, uma única unidade de busca de instruções busca pares de instruções ao mesmo tempo e coloca cada uma delas em seu próprio pipeline, completo com sua própria ULA para operação paralela.
- Para poder executar em paralelo, as duas instruções não devem ter conflito de utilização de recursos e nenhuma deve depender do resultado da outra.
- Assim como em um pipeline único, ou o compilador deve garantir que essa situação aconteça, ou os conflitos deverão ser detectados e eliminados durante a execução usando hardware extra.

1.2.11. Arquiteturas superescalares

- Pipelines duplos de cinco estágios com uma unidade de busca de instrução em comum.



1.2.11. Arquiteturas superescalares

- Embora pipelines, simples ou duplos, sejam usados em sua maioria em máquinas RISC (o 386 e seus antecessores não tinham nenhum), a partir do 486 a Intel começou a acrescentar pipelines de dados em suas CPUs.
- O 486 tinha um pipeline e o Pentium original tinha pipelines de cinco estágios mais ou menos como os da Figura.
- Embora a exata divisão do trabalho entre os estágios 2 e 3 (denominados decode-1 e decode-2).
- O pipeline principal, denominado **pipeline u**, podia executar uma instrução Pentium qualquer.
- O segundo, denominado **pipeline v**, podia executar apenas instruções com números inteiros
  - e também uma instrução simples de ponto flutuante – FXCH.

1.2.11. Arquiteturas superescalares

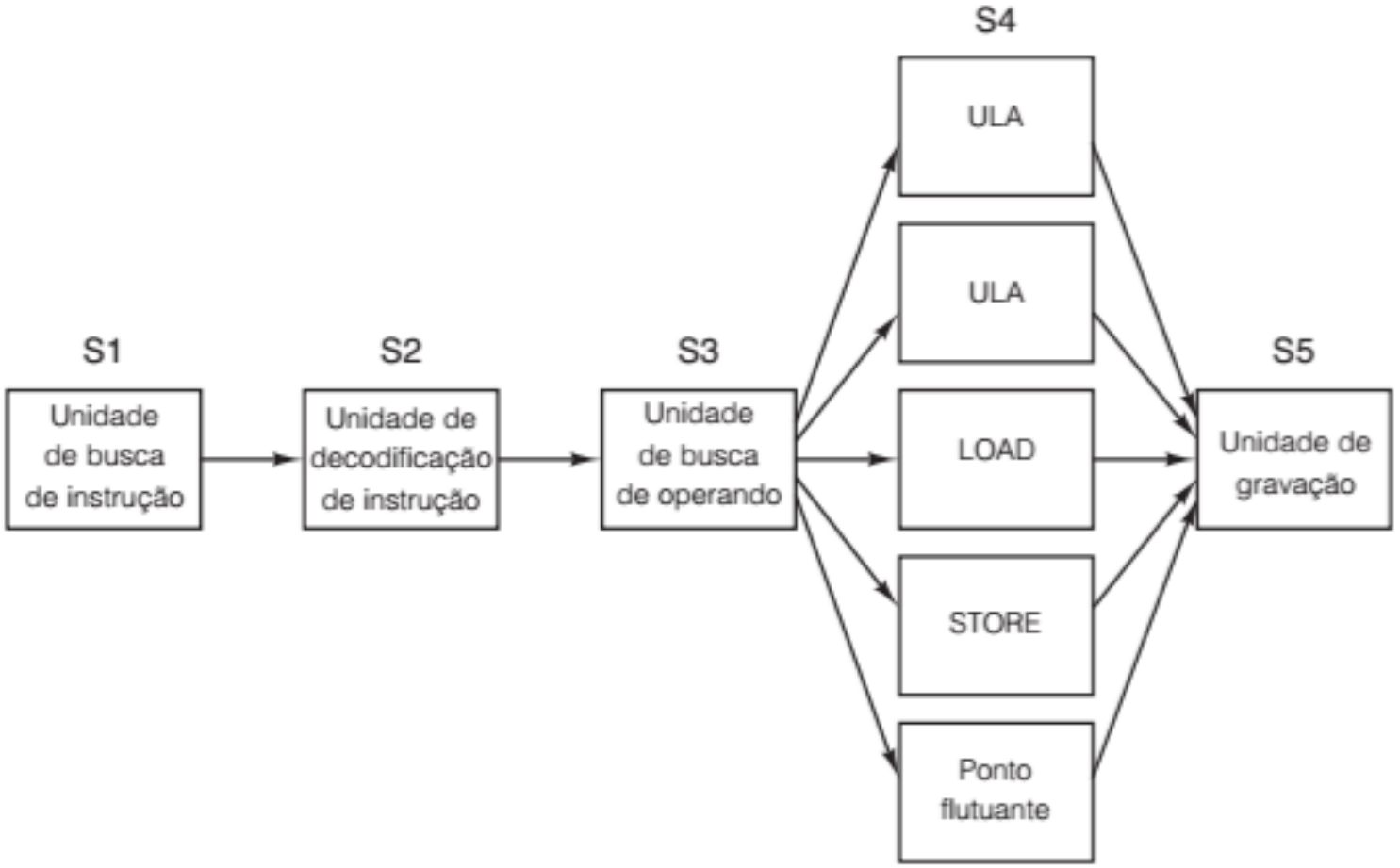
- Regras fixas determinavam se um par de instruções era compatível e, portanto, se elas podiam ser executadas em paralelo.
- Se as instruções em um par não fossem simples o suficiente ou se fossem incompatíveis, somente a primeira era executada (no pipeline u).
- A segunda era retida para fazer par com a instrução seguinte. Instruções eram sempre executadas em ordem.
- Assim, os compiladores específicos para Pentium que produziam pares compatíveis podiam produzir programas de execução mais rápidos do que compiladores mais antigos.
- Medições mostraram que um código de execução Pentium otimizado para ele era exatamente duas vezes mais rápido para programas de inteiros do que um 486 que executava à mesma velocidade de clock (Pountain, 1993).

1.2.11. Arquiteturas superescalares

- Esse ganho podia ser atribuído inteiramente ao segundo pipeline.
- Passar para quatro pipelines era concebível, mas exigiria duplicar muito hardware
  - (cientistas da computação, ao contrário de especialistas em folclore, não acreditam no número três).
- Em vez disso, uma abordagem diferente é utilizada em CPUs de topo de linha.
- A ideia básica é ter apenas um único pipeline, mas lhe dar várias unidades funcionais, conforme mostra a Figura próxima.
- Por exemplo, a arquitetura Intel Core tem uma estrutura semelhante à dessa figura, que será discutida mais tarde.
- O termo arquitetura superescalar foi cunhado para essa técnica em 1987 (Agerwala e Cocke, 1987).

1.2.11. Arquiteturas superescalares

- Processador superescalar com cinco unidades funcionais.



1.2.11. Arquiteturas superescalares

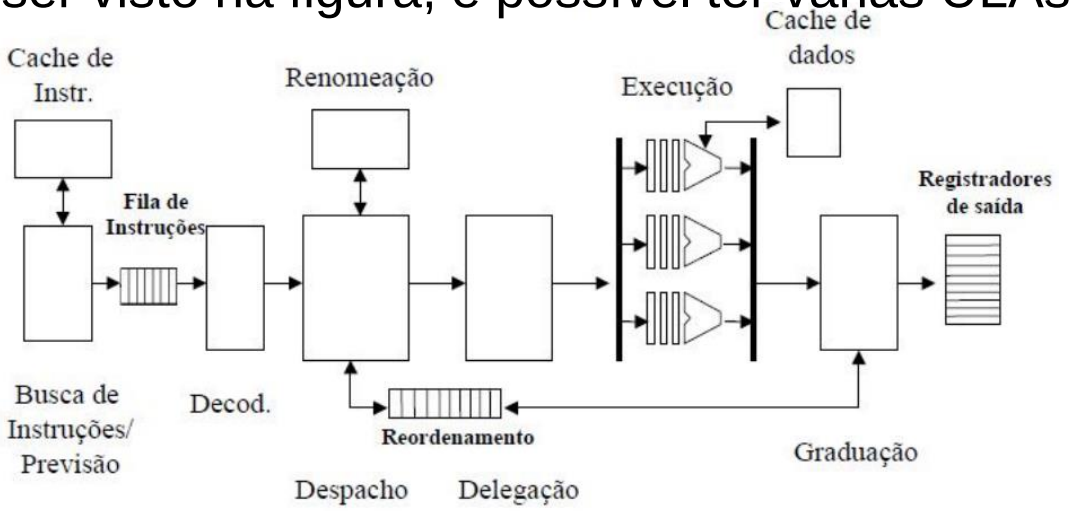
- A definição de “superescalar” evoluiu um pouco ao longo do tempo.
- Agora, ela é usada para descrever processadores que emitem múltiplas instruções – frequentemente, quatro ou seis – em um único ciclo de clock.
- Claro que uma CPU superescalar deve ter várias unidades funcionais para passar todas essas instruções.
- Uma vez que, em geral, os processadores superescalares têm um só pipeline, tendem a ser parecidos com os da Figura.
- Usando essa definição, o 6600 não era tecnicamente um computador superescalar, pois emitia apenas uma instrução por ciclo.

1.2.11. Arquiteturas superescalares

- A diferença conceitual entre:
  - uma CPU com um clock de 100 ns que executa uma instrução a cada ciclo para um grupo de unidades funcionais
  - e uma CPU com um clock de 400 ns que executa quatro instruções por ciclo para o mesmo grupo de unidades funcionais
  - é muito pequena.
- A ideia fundamental é que a taxa final é muito mais alta do que a taxa de execução, sendo a carga de trabalho distribuída entre um conjunto de unidades funcionais.
- Implícito à ideia de um processador superescalar é que o estágio S3 pode emitir instruções com rapidez muito maior do que o estágio S4 é capaz de executá-las.

1.2.11. Arquiteturas superescalares

- Se o estágio S3 executasse uma instrução a cada 10 ns e todas as unidades funcionais pudessem realizar seu trabalho em 10 ns, nunca mais do que uma unidade estaria ocupada ao mesmo tempo, o que negaria todo o raciocínio.
- Na verdade, grande parte das unidades funcionais no estágio 4 leva um tempo bem maior do que um ciclo de clock para executar, decerto as que acessam memória ou efetuam aritmética de ponto flutuante.
- Como pode ser visto na figura, é possível ter várias ULAs no estágio S4.



1.2.12. Computadores paralelos

- Um número substancial de problemas em domínios de cálculo como ciências físicas, engenharia e gráficos de computador envolve laços e matrizes, ou então tem estrutura de alta regularidade.
- Muitas vezes, os mesmos cálculos são efetuados em muitos conjuntos diferentes de dados ao mesmo tempo.
- A regularidade e a estrutura desses programas os tornam alvos especialmente fáceis para aceleração por meio de execução paralela.
- Há dois métodos que têm sido usados para executar esses programas altamente regulares de modo rápido e eficaz:
  - processadores SIMD,
  - e processadores vetoriais.

1.2.12. Computadores paralelos

- Embora esses dois esquemas guardem notáveis semelhanças na maioria de seus aspectos,
  - o primeiro deles é considerado um computador paralelo,
  - o segundo é considerado uma extensão de um processador único.
- Computadores paralelos de dados encontraram muitas aplicações bem-sucedidas como consequência de sua notável eficiência.
- Eles são capazes de produzir poder de computação significativo com menos transistores do que os métodos alternativos.
- Gordon Moore (da lei de Moore) observou que o silício custa cerca de 1 bilhão de dólares por acre (4.047 m^2).
- Assim, quanto mais poder de computação puder ser espremido desse acre de silício, mais dinheiro uma empresa de computador poderá obter vendendo silício.

1.2.12. Computadores paralelos

- Os processadores paralelos de dados são um dos meios mais eficientes de espremer o desempenho do silício.
- Como todos os processadores estão rodando a mesma instrução, o sistema só precisa de um “cérebro” controlando o computador.
- Em consequência, o processador só precisa de um estágio de busca, um estágio de decodificação e um conjunto de lógica de controle.
- Essa é uma enorme economia no silício, que dá aos computadores paralelos uma grande vantagem sobre outros processadores, desde que o software que eles estejam rodando seja altamente regular, com bastante paralelismo.

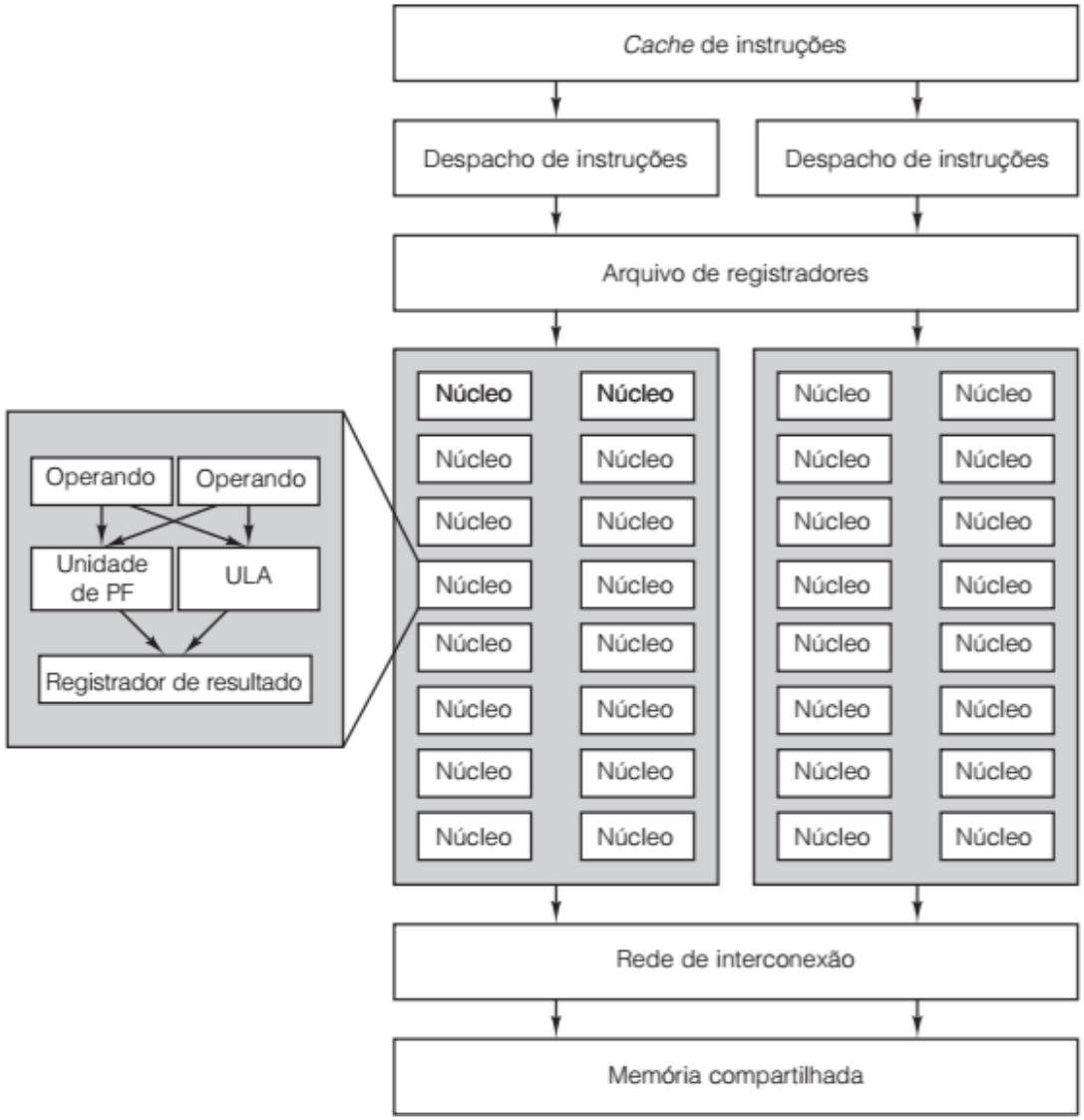
1.2.12. Computadores paralelos

- Um processador SIMD (Single Instruction-stream Multiple Data-stream, ou fluxo único de instruções, fluxo múltiplo de dados) consiste em:
  - um grande número de processadores idênticos,
  - que efetuam a mesma sequência de instruções sobre diferentes conjuntos de dados.
- O primeiro processador SIMD do mundo foi o ILLIAC IV da Universidade de Illinois (Bouknight et al., 1972).
- O projeto original do ILLIAC IV consistia em quatro quadrantes, cada um deles com uma grade quadrada de 8 × 8 elementos de processador/memória.
- Uma única unidade de controle por quadrante transmitia uma única instrução a todos os processadores, que era executada no mesmo passo por todos eles, cada um usando seus próprios dados de sua própria memória.

1.2.12. Computadores paralelos

- Por causa de um excesso de custo, somente um quadrante de 50 megaflops (milhões de operações de ponto flutuante por segundo) foi construído;
  - se a construção da máquina inteira de 1 gigaflop tivesse sido concluída,
  - ela teria duplicado a capacidade de computação do mundo inteiro.
- As modernas unidades de processamento de gráficos (GPUs) contam bastante com o processamento SIMD para fornecer poder computacional maciço com poucos transistores.
- O processamento de gráficos foi apropriado para processadores SIMD porque a maioria dos algoritmos é altamente regular, com operações repetidas sobre pixels, vértices, texturas e arestas.

1.2.12. Computadores paralelos



1.2.12. Computadores paralelos

- A Figura mostra o processador SIMD no núcleo da GPU Fermi da Nvidia.
- A GPU Fermi contém até 16 multiprocessadores de fluxo (com memória compartilhada – SM) SIMD, com cada multiprocessador contendo 32 processadores SIMD.
- A cada ciclo, o escalonador seleciona dois threads para executar no processador SIMD.
- A próxima instrução de cada thread é então executada em até 16 processadores SIMD, embora possivelmente menos se não houver paralelismo de dados suficiente.

1.2.12. Computadores paralelos

- Se cada thread for capaz de realizar 16 operações por ciclo, um núcleo GPU Fermi totalmente carregado com 32 multiprocessadores realizará incríveis 512 operações por ciclo.
- Esse é um feito impressionante, considerando que uma CPU quad-core de uso geral com tamanho semelhante lutaria para conseguir 1/32 desse processamento.
- Para um programador, um processador vetorial se parece muito com um processador SIMD.
- Assim como um processador SIMD, ele é muito eficiente para executar uma sequência de operações em pares de elementos de dados.

1.2.12. Computadores paralelos

- Diferente de um processador SIMD, todas as operações de adição são efetuadas em uma única unidade funcional, de alto grau de paralelismo.
- A Cray Research, empresa fundada por Seymour Cray, produziu muitos processadores vetoriais, começando com o Cray-1 em 1974 e continuando até os modelos atuais.
- Processadores SIMD, bem como processadores vetoriais, trabalham com matrizes de dados.
- Ambos executam instruções únicas que, por exemplo, somam os elementos aos pares para dois vetores.

1.2.12. Computadores paralelos

- Porém, enquanto o processador SIMD faz isso com tantos somadores quantos forem os elementos do vetor,
  - o processador vetorial tem o conceito de um registrador vetorial,
  - que consiste em um conjunto de registradores convencionais que podem ser carregados com base na memória em uma única instrução que,
  - na verdade, os carrega serialmente com base na memória.
- Então, uma instrução de adição vetorial efetua as adições a partir dos elementos de dois desses vetores, alimentando-os em um somador com paralelismo (pipelined) com base em dois registradores vetoriais.

1.2.13. Computadores paralelos

- O resultado do somador é outro vetor, que pode ser armazenado em um registrador vetorial ou usado diretamente como um operando para outra operação vetorial.
- As instruções SSE (Streaming SIMD Extension) disponíveis na arquitetura Intel Core utilizam esse modelo de execução para agilizar o cálculo altamente regular, como multimídia e software científico.
- Nesse aspecto particular, o ILLIAC IV é um dos ancestrais da arquitetura Intel Core.

1.2.14. Multiprocessadores

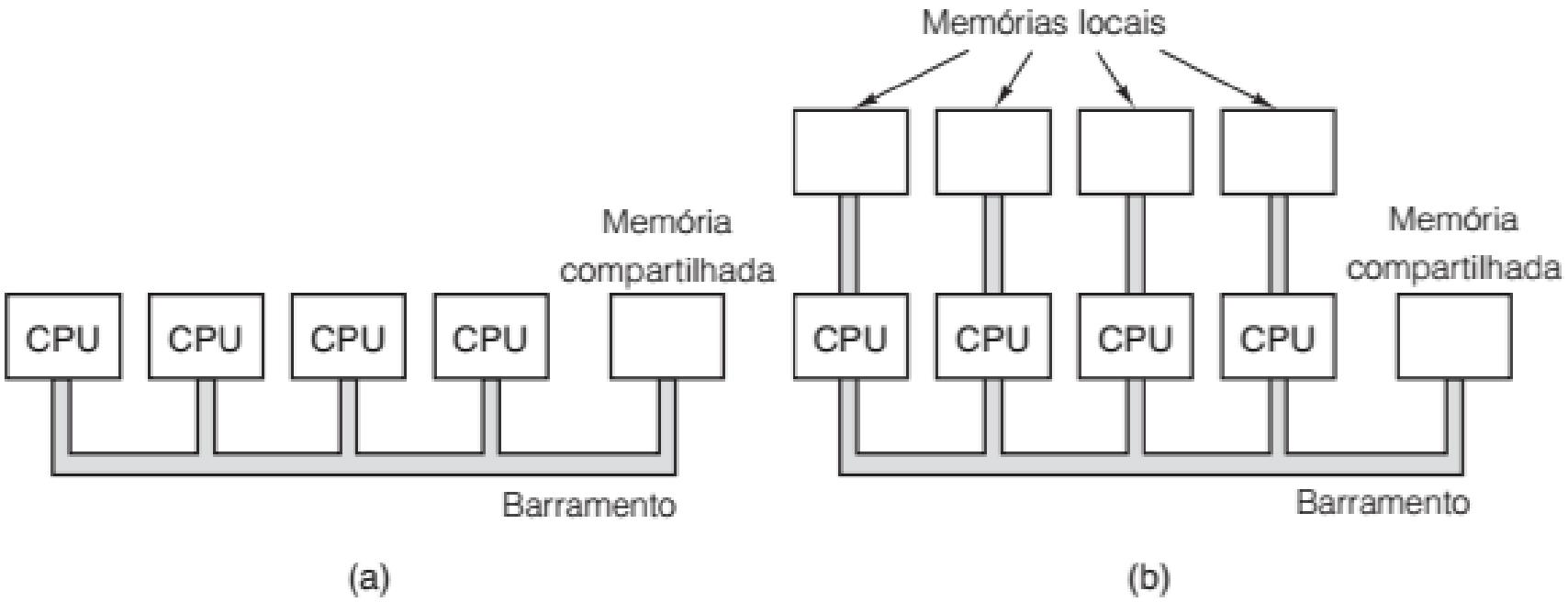
- Os elementos de processamento em um processador SIMD não são CPUs independentes,
  - uma vez que há uma só unidade de controle compartilhada por todos eles.
- Nosso primeiro sistema paralelo com CPUs totalmente desenvolvidas é o multiprocessador,
  - um sistema com mais de uma CPU que compartilha uma memória em comum, como um grupo de pessoas que, dentro de uma sala de aula, compartilha um quadro em comum.
- Uma vez que cada CPU pode ler ou escrever em qualquer parte da memória, elas devem se coordenar (em software) para evitar que uma atrapalhe a outra.

1.2.14. Multiprocessadores

- Quando duas ou mais CPUs têm a capacidade de interagir de perto, como é o caso dos multiprocessadores, diz-se que elas são fortemente acopladas.
- Há vários esquemas de implementação possíveis. O mais simples é um barramento único com várias CPUs e uma memória, todas ligadas nele.
- Um diagrama desse tipo de multiprocessador de barramento único é mostrado na Figura.

1.2.14. Multiprocessadores

- (a) Multiprocessador de barramento único.
- (b) Multicomputador com memórias locais.



1.2.14. Multiprocessadores

- Com um grande número de processadores velozes tentando acessar a memória pelo mesmo barramento, surgirão conflitos.
- Projetistas de multiprocessadores apresentaram vários esquemas para reduzir essa disputa e melhorar o desempenho.
- Um desses esquemas, mostrado na Figura (b), dá a cada processador um pouco de memória local só dele, que não é acessível para os outros.
- Essa memória pode ser usada para o código de programa e para os itens de dados que não precisam ser compartilhados.

1.2.14. Multiprocessadores

- O acesso a essa memória privada não usa o barramento principal, o que reduz muito o tráfego no barramento.
- Multiprocessadores têm a vantagem sobre outros tipos de computadores paralelos: é fácil trabalhar com o modelo de programação de uma única memória compartilhada.
- Por exemplo,
  - um programa que procura células cancerosas na foto de algum tecido, tirada por um microscópio.
  - a fotografia digitalizada poderia ser mantida na memória em comum, sendo cada processador designado para caçar essas células em alguma região.
- Uma vez que cada processador tem acesso a toda a memória, estudar a célula que começa em sua região designada mas atravessa a fronteira da próxima região não é problema.

Questões

- Considere a operação de uma máquina de Von Neuman. Suponha que carregar os registradores de entrada da ULA leve 5 ns, executar a ULA demore 10 ns e armazenar o resultado de volta no registrador de rascunho tome 5 ns. Qual é o número máximo de MIPS de que essa máquina é capaz na ausência de paralelismo (pipelining)?
- Qual é a finalidade da etapa “Alterar o contador de programa para que aponte para a próxima instrução” na lista da execução da instução? O que aconteceria se essa etapa fosse omitida?
- No computador 1, o tempo de execução de todas as instruções é 10 ns. No computador 2, o tempo de execução é de 5 ns. Você pode afirmar com certeza que o computador 2 é mais rápido? Discuta sua resposta.

Questões

- Imagine que você está projetando um computador de um só chip para um sistema embutido. O chip conterá toda sua memória e executará à mesma velocidade da CPU sem penalidade de acesso. Diga por que as seguintes estratégiassão tão importantes (admitindo que ainda se deseje alto desempenho) :
  - Todas as instruções são executadas diretamente por hardware
  - Maximize a taxa de execução das instruções
  - Instruções devem ser fáceis de decodificar
  - Somente LOAD e STORE devem referenciar a memória
  - Providencie muitos registradores