Unidade Central de Processamento

Prof. Marcos Quinet
Universidade Federal Fluminense – UFF
Instituto de Ciência e Tecnologia

Unidade Central de Processamento

- O processador é o componente principal de um sistema computacional, responsável pela realização das operações de **processamento** e de **controle**, durante a execução de um programa

- Um programa, para ser efetivamente executado pelo processador, deve ser constituído de uma série de instruções de máquina. Para que a execução tenha início, as instruções devem ser armazenadas na memória principal

- Ao nos referirmos ao processador, é comum empregarmos os termos UCP ou CPU

Tarefas da UCP

- As funções da UCP são:
  - Buscar na memória a instrução a ser executada;
  - Interpretar que operação a instrução está explicitando;
  - Buscar os dados onde estiverem armazenados;
  - Executar efetivamente a operação com os dados e armazenar o resultado no local definido pela instrução;
  - Reiniciar o processo, buscando a próxima instrução.

Ciclo de Instrução

- Estas etapas compõem o que se denomina um *ciclo de instrução*. Este ciclo se repete indefinidamente até que o sistema seja desligado, ou ocorra algum tipo de erro, ou seja encontrada uma instrução de parada

- Em outras palavras, a UCP é projetada e fabricada com o propósito único de executar sucessivamente pequenas operações, na ordem definida pela organização do programa

Funções da UCP

- As atividades realizadas pela UCP podem ser divididas em duas grandes categorias funcionais: função de processamento e função de controle

  - A função de processamento se encarrega de realizar as atividades relacionadas com a efetiva execução de uma operação, ou seja, processar;

  - A função de controle é exercida pelos componentes da UCP que se encarregam das atividades de busca, interpretação e controle da execução das instruções, bem como do controle da ação dos demais componentes do sistema de computação (memória, entrada/saída).

Funções da UCP

- Anteriormente, foi apresentado o conceito do que significa processamento de dados, a ação de manipular um ou mais valores (dados) em uma certa sequência de passos, de modo a produzir um resultado útil. Este resultado muda conforme o tipo de operação realizada

Funções da UCP

- Processar o dado significa executar com ele uma ação que produza algum tipo de resultado
- Entre as tarefas comuns a esta função (processamento) podem ser citadas as que realizam:

  - Operações aritméticas;
  - Operações lógicas;
  - Movimentação de dados;
  - Desvios;
  - Operações de entrada ou saída.

ULA – Unidade Lógica e Aritmética

- O dispositivo principal desta área de atividades de uma CPU é chamado de ALU ou ULA (*Unidade Lógica e Aritmética*)

- Os demais componentes relacionados com a função processamento são os registradores, que servem para armazenar dados (ou para guardar resultados) a serem utilizados pela ULA. A interligação entre estes componentes é efetuada pelo barramento interno da UCP

ULA

- A ULA é o dispositivo da UCP que executa realmente as operações matemáticas com os dados, como por exemplo operações de deslocamento, incremento, decremento e complemento, além das operações aritméticas e lógicas já apresentadas

- Tais operações podem utilizar um (operações unárias) ou dois (operações binárias) valores e, por isso, a ULA possui dois registradores de entrada

Componentes da UCP

- A ULA é um aglomerado de circuitos lógicos e componentes eletrônicos simples que, integrados, realizam as operações citadas

- A seção de controle é formada basicamente pela Unidade de Controle (UC) e pelo registrador de instrução (IR)

- A seção de processamento é formada pela unidade lógica e aritmética e por diversos registradores

Registradores

- São pequenas unidades de memória, implementadas na UCP, com as seguintes características:
  - Tempo de acesso muito baixo, algo em torno de 10 a 20 nanossegundos
  - Capacidade de armazenar um único dado (instrução, dado ou endereço), fazendo, assim, com que sua capacidade seja de uma palavra, cujo tamanho varia de arcordo com a arquitetura do sistema

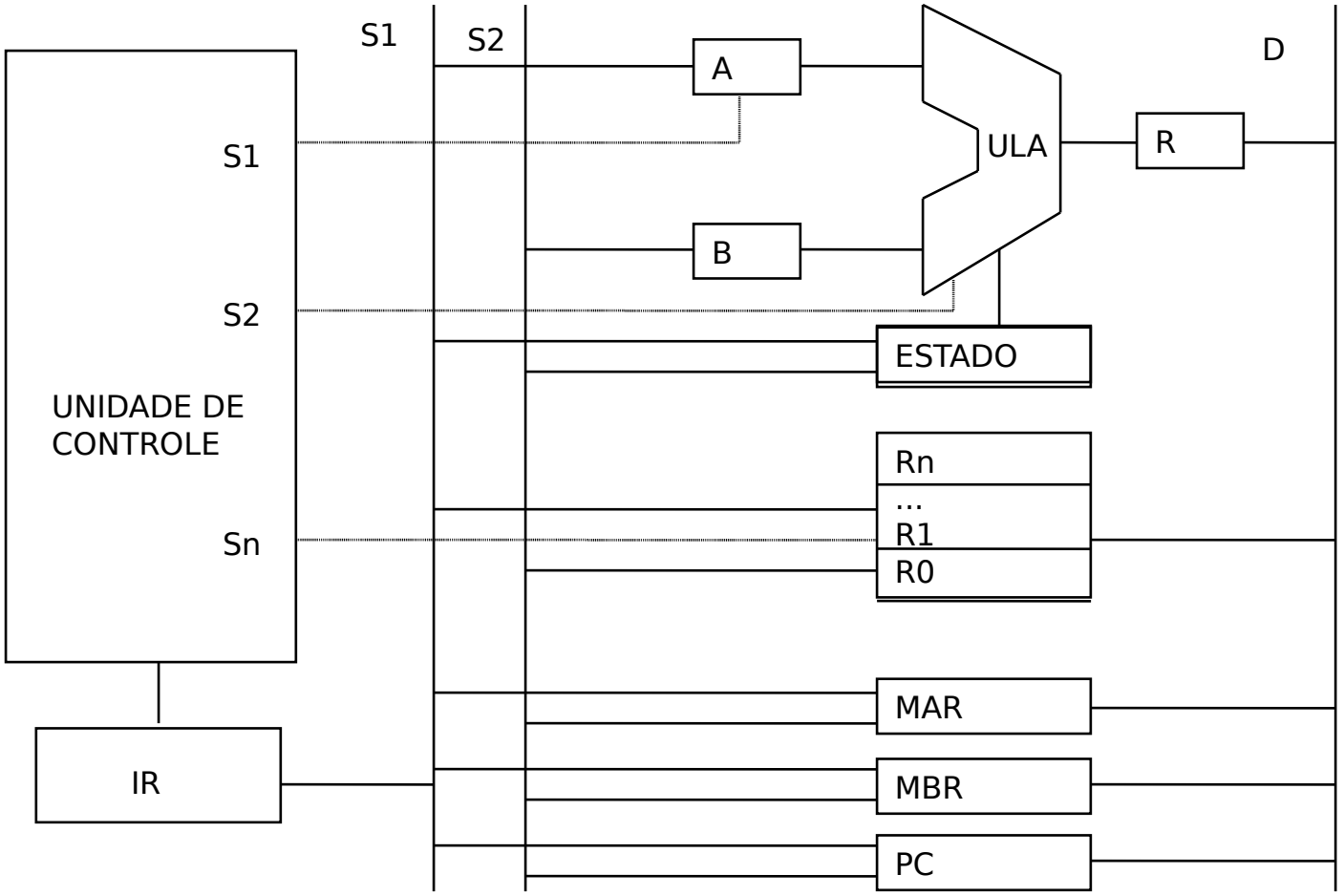
Registradores

- Volatilidade - registradores são memórias de semicondutores e, portanto, necessitam de energia elétrica para funcionarem. Assim, registradores são memórias voláteis

- Temporariedade - registradores de dados armazenam os dados apenas o tempo necessário para sua utilização na ULA, ou seja, um período muito curto

- Custo - devido à tecnologia mais avançada de sua fabricação, os registradores consistem no tipo de memória de maior custo

Seções de Controle e Processamento



Seção de Processamento

Registradores:

- A, B, R: registradores temporários que armazenam, respectivamente, os valores a serem operados e o valor resultante
- R0 ... Rn: registradores de dados
- MAR: endereço da locação de memória onde será feito o acesso
- MBR: armazena temporariamente a informação transferida de ou para a locação de memória endereçada pelo MAR
- PC: contador de programa, contém o endereço da locação de memória onde se encontra a próxima instrução a ser executada
- IR: registrador de instruções
- ESTADO: guarda informações sobre o resultado produzido pela ULA. Exemplo: o bit n é 1 se o resultado for nulo e 0 se for não-nulo

Seção de Processamento

- Além dos registradores de dados, a UCP possui sempre outros registradores (que não participam diretamente da função processamento), com funções específicas ou que funcionam para a área de controle.

- Entre estes registradores podemos destacar:
  - Registrador de instrução (*IR – Instruction Register*)
  - Contador de programa (*PC – Program Counter*)
  - Registrador de endereços de memória (*MAR – Memory Address Register*)
  - Registrador de dados de memória (*MBR – Memory Buffer Register*)

Seção de Processamento

- *Registrador de Dados da Memória* (*MBR*) – registrador que armazena temporariamente a informação que está sendo transferida da MP para a UCP (leitura) ou da UCP para a MP (escrita). Em seguida, a referida informação é reencaminhada para outro elemento da UCP para processamento ou para uma célula da MP, conforme o tipo da operação de transferência
-
- *Registrador de Endereços da Memória* (*MAR*) – registrador que armazena temporariamente o endereço de acesso a uma posição de memória, ao se iniciar uma operação de leitura ou de escrita. Em seguida, o referido endereço é encaminhado à área de controle da MP para decodificação e localização da célula desejada

Seção de Processamento

- Os barramentos internos de comunicação estão representados em S1, S2 e D. Os dois primeiros permitem a transferência de dados dos registradores para a ULA. O barramento D permite a transferência do resultado produzido pela ULA, temporariamente armazenado no registrador R, para outro registrador

Seção de Processamento

- Uma arquitetura de *n* bits é quando todas as operações da ULA podem ser realizadas sobre operandos de até *n* bits

- Normalmente em uma arquitetura de *n* bits, os registradores de dados e os barramentos internos também são de *n* bits, permitindo que os dados sejam armazenados e transferidos da forma mais eficiente possível

Seção de Processamento

- Para que um dado possa ser transferido para a ULA, é necessário que ele permaneça, mesmo que por um breve instante, armazenado em algum registrador.

- Além disso, o resultado de uma operação realizada na ULA deve ser armazenado temporariamente, de modo que possa ser reutilizado mais adiante (por outra instrução) ou apenas para ser, em seguida, transferido para a memória

Seção de Processamento

- Para atender a estes propósitos, a UCP é fabricada com uma certa quantidade de registradores, destinados ao armazenamento de dados. Servem, pois, de memória auxiliar da ULA. Há sistemas nos quais um desses registradores, denominado *acumulador* (ACC), além de armazenar dados, serve de elemento de ligação da ULA com os demais dispositivos da UCP

Seção de Controle

- Todas as operações básicas que ocorrem dentro da seção de processamento são comandadas pela seção de controle. Ao efetuar a busca da instrução, a unidade de controle interpreta a instrução de modo a identificas quais as operações básicas que devem ser realizadas e ativa sinais de controle (S1, S2, ..., S*n*) que fazem uma operação básica de fato acontecer

- Em outras palavras, a seção de controle é projetada para entender o quê fazer, como fazer e comandar quem vai fazer no momento adequado

Unidade de Controle

- É o dispositivo mais complexo da UCP. Além de possuir a lógica necessária para realizar a movimentação de dados e instruções de e para a UCP, esse dispositivo controla a ação da ULA

- Os sinais de controle emitidos pela UC ocorrem em vários instantes durante o período de realização de um ciclo de instrução e, de modo geral, todos possuem uma duração fixa e igual, originada em um gerador de sinais usualmente conhecido como relógio (*clock*)

Registrador de Instruções - IR

- É o registrador que tem a função específica de armazenar a instrução a ser executada pela UCP. Ao se iniciar um ciclo de instrução, a UC emite um sinal de controle que acarretará a realização de um ciclo de leitura para buscar a instrução na memória (uma cópia dela), e que, via barramento de dados e MAR, será armazenada no IR

Execução de Instruções

- Como já vimos, toda operação feita pela UCP, como uma operação aritmética, manipulação de um arquivo (*open, write, close*) ou operação lógica é composta por conjuntos de operações mais simples, chamadas de operações básicas

- Instruções são representadas através de mnemônicos, que são representações dos códigos de operação, e referências a endereços de memória

- Como exemplo de como uma instrução é executada, vejamos a operação ADD (R1, R2, R3), ou seja, a soma do conteúdo armazenado no registrador R1 com o conteúdo do registrador R2, cujo resultado será armazenado em R3

Execução de Instruções

- A execução desta instrução requer as seguintes operações básicas:

1. Transferência do conteúdo do registrador de dados R1 para o registrador temporário A;
2. Transferência do conteúdo do registrador de dados R2 para o registrador temporário B;
3. Adição dos dados armazenados nos registradores A e B e armazenamento do resultado no registrador R;
4. Transferência do conteúdo do registrador R para o registrador R3.

Execução de Instruções

Operação básica	Sinal de controle	Descrição da operação básica
(1)(2)	s1, s2	Coloca o conteúdo de R1, R2 para os barramentos S1, S2, respectivamente
	s3, s4	Armazena a informação presente nos barramentos S1, S2 em A, B, respectivamente
(3)	s5	Seleciona a operação de soma na ALU
	s6	Armazena o resultado produzido pela ALU em R
(4)	s7	Coloca o conteúdo de R para o barramento D
	s8	Armazena a informação presente no barramento D em R3

Execução de Instruções

Conjunto de Instruções do Processadores

- O conjunto de instruções é um dos pontos centrais na arquitetura de um processador. Vários aspectos na definição e implementação da arquitetura são influenciados pelas características do conjunto de instruções

- O conjunto de instruções utilizadas afeta não somente o projeto da seção de processamento: a estrutura e complexidade da unidade de controle é determinada diretamente pelas características do conjunto de instruções

Linguagens de programação: Hierarquia

- Surgimento da Linguagem de alto nível
  - objetivo: tornar a comunicação com o computador mais simples e com menos instruções do que a linguagem de montagem;
  - mais distante da máquina;
  - o programador não precisa se preocupar com o tipo de CPU ou de memória onde o programa será executado
  - Exs.: Fortran, Pascal, C;
  - necessidade de uma conversão para instruções de máquina.

PROCESSO DE COMPILAÇÃO

Linguagem de Montagem

- Em geral, os programas são desenvolvidos em uma linguagem de alto nível, com Pascal, C, ou Java. O compilador traduz o programa de alto nível em uma sequência de instruções de processador

- Desta tradução resulta o programa em linguagem de montagem (*assembly language*). A linguagem de montagem é uma forma de representar textualmente as instruções oferecidas pela arquitetura, cada uma com uma linguagem de montagem em particular

Linguagem de Montagem

- No programa em linguagem de montagem, as instruções são representadas através de “abreviações”, chamadas de mnemônicos, que associam o nome da instrução à sua função, como por exemplo:

ADD	**Adição**
SUB	**Subtração**
MULT	**Multiplicação**
DIV	**Divisão**
LW	**Carregar dados da memória**
SW	**Armazenar dados na memória**

Linguagem de Montagem

- O programa em linguagem de montagem é convertido para um programa em código objeto pelo montador (*assembler*)

- O montador traduz diretamente uma instrução da forma textual para a forma de código binário. É sob a forma binária que a instrução é carregada na memória e interpretada pelo processador

Modulação de Programas

- Programas complexos são normalmente estruturados em módulos. Cada módulo é compilado separadamente e submetido ao montador, gerando diversos módulos em código objeto. Estes módulos são reunidos pelo ligador (*linker*), resultando finalmente no programa executável que é carregado na memória

Compilando para *assembly*

Código em C

```
int sum(int x, int y)
{
  int t = x+y;
  return t;
}
```

Assembly gerado

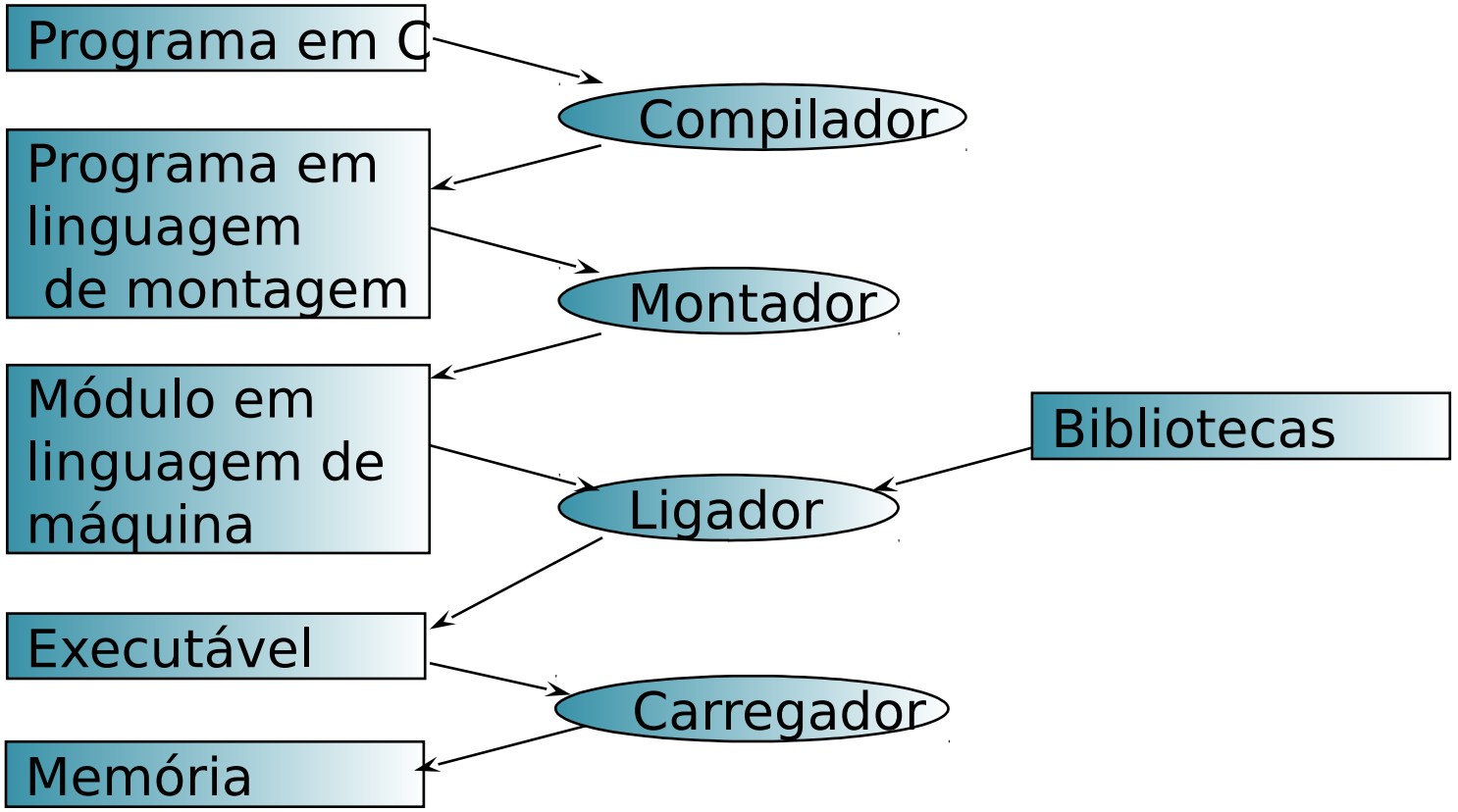
```
_sum:
    pushl %ebp
    movl %esp,%ebp
    movl 12(%ebp),%eax
    addl 8(%ebp),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

Obtido com o comando

`gcc -O -S code.c`

Produz arquivo `code.s`

Processo de Compilação e Execução



Ligadores e Carregadores

- Código objeto não é suficiente para ser executado
- Tarefas a serem executadas
  - Resolução de referências simbólicas (ligação)
  - Ajuste de endereços (relocação)
  - Alocação de espaço em memória (alocação)
  - Colocação das instruções e dados fisicamente na memória (carga)
- Código Objeto
  - Absoluto: os endereços constantes do código são endereços reais de memória
  - Relocável: os endereços são relativos ao início do programa, transformando-se em endereços reais apenas na execução

  Ä mais flexível e mais utilizado!

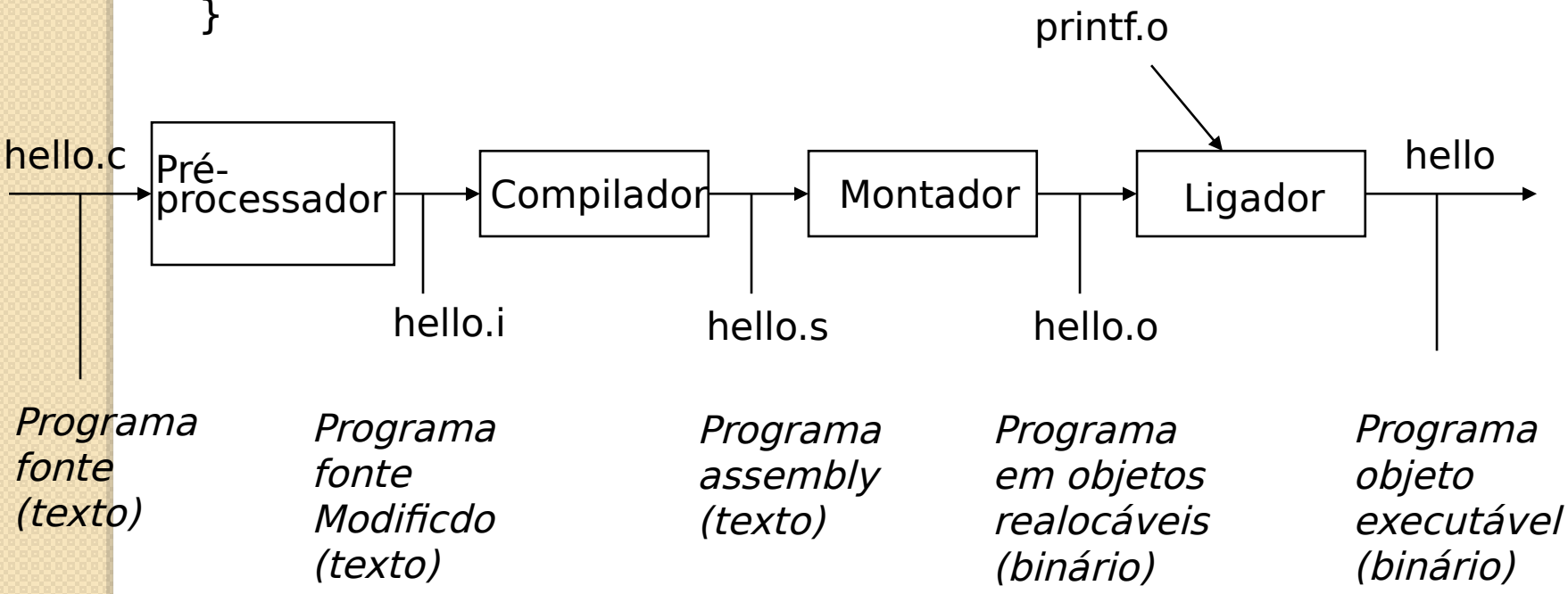
Ligador

- Rotinas comuns são agrupadas em bibliotecas
- Ligador resolve as referências externas
  - Agrega o código objeto das bibliotecas a serem utilizadas, criando um único módulo de carga
  - Substitui chamadas a procedimentos por seus endereços
- Relocação
  - União dos espaços de endereçamento de cada módulo objeto
- Gera código executável (ou módulo de carga)
- DLL (*Dynamic Link Library*)
  - Os procedimentos só são ligados em tempo de execução
  - Ligador dinâmico

Execução de um programa

- Código em C para programa hello.c

```
#include <stdio.h>
int main()
{
      printf("hello, world\n")
}
```

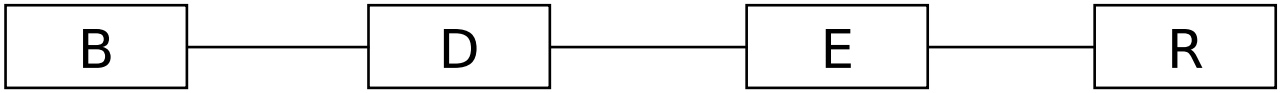


Técnicas Especiais

Pipelining

- No processo básico de execução de instruções, estas são executadas sequencialmente, ou seja, a execução de uma nova instrução inicia-se somente quanto a execução da instrução anterior é completada
- Na técnica de *pipelining*, permite-se que várias instruções sejam executadas simultaneamente, pois os passo da execução são realizados por unidades independentes, denominadas estágios do *pipeline*
- O exemplo a seguir representa um pipeline de quatro estágios

Pipelining

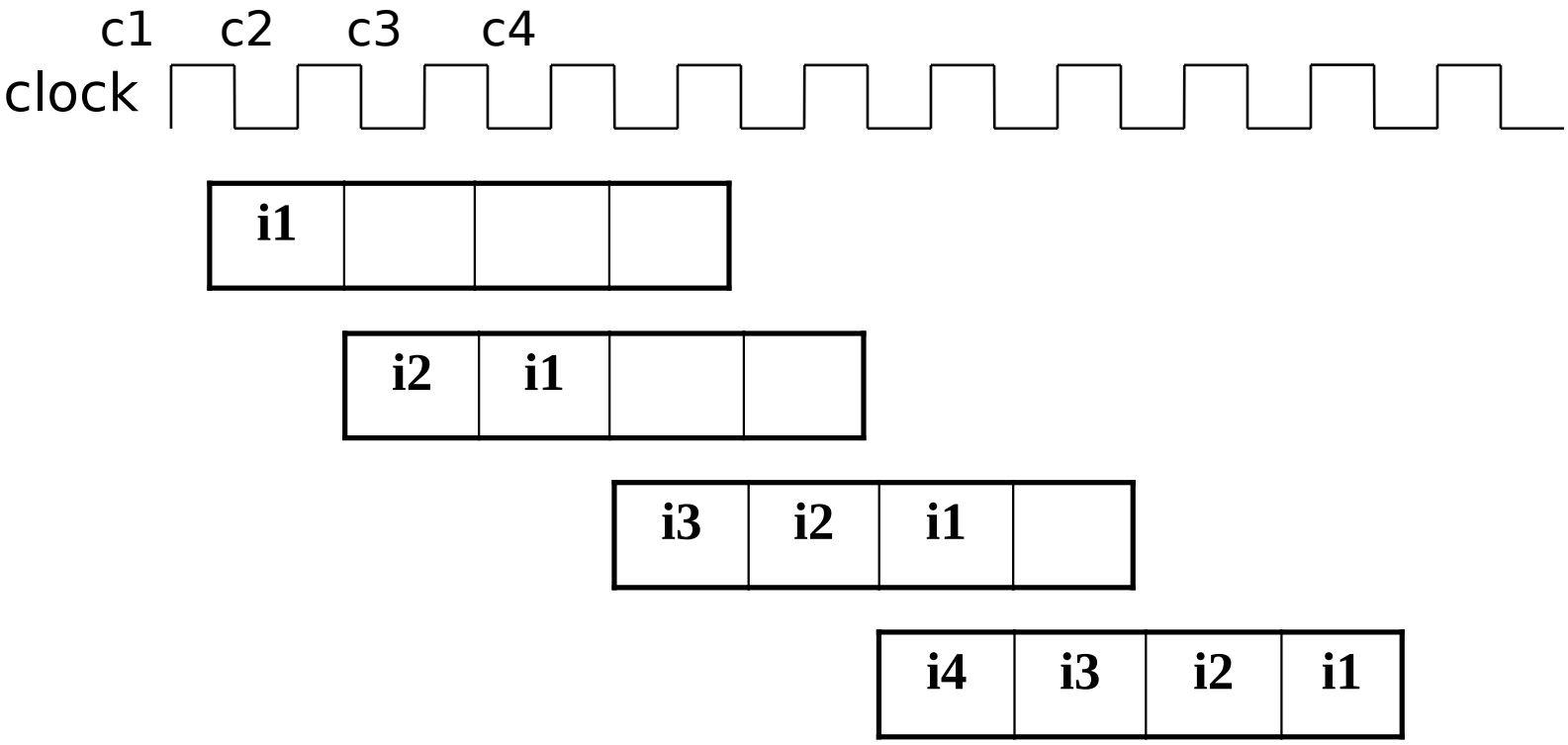


Estágios:

B: busca
D:decodificação
E: execução
R: resultado

A execução de uma instrução inicia-se pelo estágio B, sendo completada no estágio R. Em condições normais, uma instrução avança para o estágio seguinte a cada novo ciclo de clock.

Pipelining



Pipelining

- No ciclo c1, a instrução i1 é buscada no estágio B
- No ciclo c2, a instrução i1 é decodificada no estágio D, enquanto o estágio B busca uma nova instrução, i2
- No ciclo c3, o estágio E executa a instrução i1, ao mesmo tempo que que o estágio D decodifica a instrução i2 e o estágio B busca a instrução i3
- No ciclo c4, o resultado da instrução i1 é armazenado pelo estágio R, as instruções i2 e i3 avançam para o próximo estágio e o estágio B busca a instrução i4

Pipelining

- Novas instruções entram no *pipeline* antes que a execução das instruções anteriores seja completada

- Quando o *pipeline* encontra-se cheio, várias instruções estão sendo executadas em paralelo, uma em cada estágio do *pipeline*

- Na realidade, o aspecto mais importante na técnica de *pipeline* é que uma instrução seja completada em um ciclo de *clock*. Em uma arquitetura sequencial, a execução de uma instrução consome vários ciclos de *clock*, fazendo que o número médio de ciclos esteja bem acima da média de 1 ciclo/operação obtida com o uso da técnica de *pipelining*

Pipelining

- As arquiteturas modernas voltadas para aplicações de alto desempenho utilizam a técnica de *pipelining*, sendo que suas instruções são implementadas de forma a apresentarem um fator de 1 ciclo/instrução

- Apesar de conceitualmente simples, o uso de *pipelining* encontra alguns problemas na prática. A técnica funciona se houver uma continuidade no fluxo de instruções, o que muitas vezes não acontece, por exemplo, quando existe uma dependência de dados entre duas instruções

Pipelining

- Uma dependência entre duas instruções *i* e *j* existe quando um dos operandos da instrução *j* é o resultado da instrução *i* anterior. Se estas duas instruções forem executadas simultaneamente no *pipeline*, pode acontecer que a instrução *i* ainda não tenha produzido seu resultado no momento que a instrução *j* lê seus operandos, resultando em uma execução incorreta, se não existir uma forma de controle

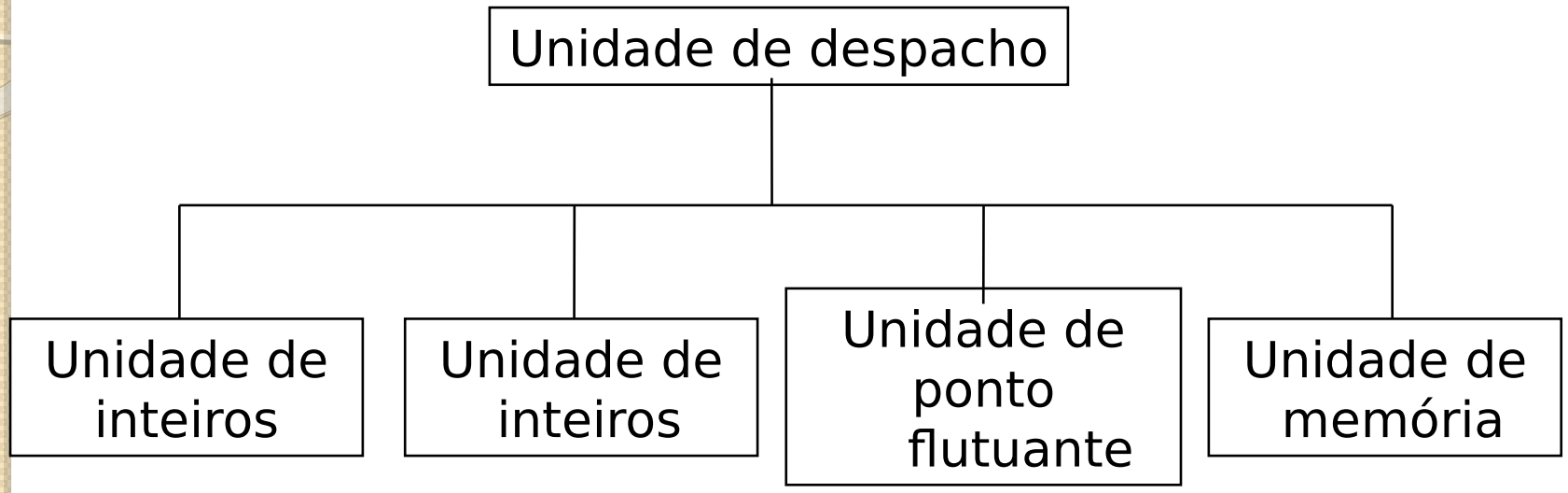
Pipelining

- A solução mais simples seria paralisar a execução de *j* quando seus operando são buscados, até que a operação *i* seja completada. No entanto, esta paralisação parcial do *pipeline* resulta em uma descontinuidade no fluxo de informações, elevando o número médio de ciclos por instrução

Arquiteturas super-escalares

- Com o *pipelining*, apenas uma instrução é completada por ciclo, resultando em um IPC (*instructions per cicle*) máximo de 1 ciclo/instrução. Seria possível aumentar o desempenho de um sistema se o fator ipc fosse elevado para acima desta média
- Uma arquitetura super-escalar opera de forma que mais de uma instrução possa ser completada a cada ciclo, através de múltiplas unidades funcionais independentes, que executam instruções em paralelo
- A cada ciclo, múltiplas instruções podem ser enviadas (despachadas) para a execução nestas unidades funcionais

Arquiteturas super-escalares



- Neste modelo teórico, até quatro instruções podem ser completadas por ciclo. A cada ciclo, a unidade de despacho busca e decodifica um certo número de instruções, e verifica quais destas instruções podem ser despachadas para as unidades funcionais

Arquiteturas super-escalares

- Normalmente uma instrução é despachada para a unidade funcional se ela encontra-se disponível e não existe dependência de dados entre esta instrução e uma outra ainda em execução

- Décadas de 70 e 80: a principal limitação do hardware estava na capacidade de armazenamento e na velocidade da memória principal

- Devido a estas características, era desejável que o código executável de um programa tivesse tamanho reduzido, pois programas maiores teriam um maior tempo de execução devido ao maior número de instruções acessadas na memória

Arquiteturas super-escalares

- Desejava-se reduzir o tamanho do código não somente para economizar o espaço, mas também obter-se um melhor desempenho

- Linguagens de alto nível eram consideradas ineficientes em termos de espaço e tempo de execução, porém, a complexidade das aplicações foi tornando proibitivo o uso de programação *assembly*

- Este foi o motivo para o desenvolvimento de arquiteturas que suportariam o uso de linguagens de alto nível, sendo que sua execução seria tão eficiente quanto a de programas escritos em *assembly*

Arquiteturas super-escalares

- Surgimento das arquiteturas para linguagens de alto nível (HLLC – *High-Level Language Computer architectures*)
  - Ex: arquitetura CISC (*Complex Instruction Set Computers*)

- O conjunto de instruções de diferentes arquiteturas baseiam-se no nível de funcionalidade; nas de alto nível, cada instrução realiza um grande número de operações

Alto nível de funcionalidade: ADD A, B, C

Baixo nível de funcionalidade:

```
LOAD    A, R1
LOAD    B, R2
ADD     R1, R2, R3
STORE   R3, C
```

Arquiteturas super-escalares

- Ambos os casos implementam a operação C = A + B, em diferentes níveis de funcionalidade

- Em uma linguagem de baixo nível de funcionalidade, são realizadas quatro operações: o carregamento das variáveis A e B em registradores, a adição propriamente dita e o armazenamento do resultado em C; em uma linguagem de alto nível de funcionalidade, os operandos são acessados diretamente na memória, é feita a adição e o resultado já é armazenado na memória

Arquiteturas super-escalares

- Linguagens de alto nível e conjuntos de instruções normalmente apresentam níveis de funcionalidade diferentes. A estas diferenças foi dado o nome de lacuna semântica. Em arquiteturas complexas, a abordagem foi de elevar o nível de funcionalidade do conjunto de instruções, diminuindo a lacuna semântica

Arquiteturas super-escalares

- Uma arquitetura com uma pequena lacuna semântica (ou seja, um conjunto de instruções com alto nível de funcionalidade) apresenta dois benefícios:
  - Programas mais eficientes em termos de espaço ocupado na memória (menor número de instruções)
  - Comandos de alto nível com um menor número de instruções resultam em um menor número de acessos a memória para busca de instruções
- Na prática, foram observadas algumas desvantagens no uso de arquiteturas complexas, que puseram em dúvida a filosofia proposta, são elas:

Arquiteturas super-escalares

- Eficácia dos programas: em alguns casos, instruções complexas possuíam um tempo de execução elevado, até mesmo maior do que uma sequência de operações simples que realiza a mesma tarefa

- Utilização de instruções: apenas uma pequena parcela das instruções oferecidas era realmente utilizada (ex.: IBM 370 tinha 183 instruções disponíveis; em 99% dos casos eram utilizadas apenas 48)

- Efeito sobre o desempenho: o aumento do nível de funcionalidade possui um efeito negativo sobre o desempenho, por exemplo, é extremamente difícil implementar uma UC com lógica aleatória, a única alternativa é o uso da microprogramação

Arquiteturas RISC

- As desvantagens das arquiteturas complexas foram o principal fator que motivaram o surgimento da filosofia RISC (*Reduced Instruction Set Computers*)

- Outro fator relevante foram os avanços nas tecnologias de fabricação de memórias, que tiveram suas velocidades de acesso aumentadas e os custos muito reduzidos

- O princípio básico das arquiteturas RISC é a simplicidade das instruções, que são frequentemente usadas na codificação de programas de alto nível. Instruções complexas são incluídas na arquitetura somente se atenderem ambos os casos a seguir:

Arquiteturas RISC

- A implementação de instruções mais complexas somente será permitida se forem realmente significativas, ou seja, ser útil na codificação de diferentes tipos de programas
- O ganho final de desempenho deve ser maior do que eventuais perdas no tempo de execução das instruções simples que ocorram com o acréscimo da instrução complexa
- Caso um destes critérios não seja atendido, é preferível a implementação da instrução complexa através de uma sequência de operações simples

Arquiteturas RISC

- São características comuns a todas as arquiteturas RISC:
  - Implementação com lógica aleatória;
  - *Pipeline* de instruções;
  - Arquitetura registrador-registrador: todos os operandos a serem utilizados em operações aritméticas e lógicas encontram-se em registradores, o que diminui a complexidade da unidade de controle;
  - Regularidade no formato das instruções: todos os códigos possuem o mesmo tamanho, igual ao de uma palavra da memória, o que permite o acesso em um único ciclo a uma instrução completa;

Sistemas Paralelos

- Até recentemente, todas as arquiteturas citadas eram caracterizadas por existir apenas um processador acoplado a uma memória principal

- Um sistema paralelo é aquele onde existem vários processadores e módulos de memória, que se comunicam através de uma certa estrutura de interconexão

- M. J. Flynn estabeleceu, em 1966, uma taxonomia para sistemas paralelos

Sistemas Paralelos

- A classificação de Flynn baseia-se em dois fatores: o número de fluxos de instruções e de fluxos de dados existentes no sistema

- Um fluxo de instrução é uma sequência de instruções endereçadas pelo contador de programa de um elemento processador. Se um sistema possui *n* contadores de programa, este sistema é capaz de processar *n* fluxos de instrução distintos

- Um fluxo de dados corresponde a um conjunto de dados que é manipulado por um elemento processador

Sistemas Paralelos

- SISD – *Single Instruction, Single Data Stream*: um único processador executa uma única sequência de instruções, utilizando dados armazenados em uma única memória. Todos os sistemas processados pertencem a esta categoria

- SIMD – *Single Instruction, Multiple Data Stream*: uma única instrução de máquina controla a execução simultânea de um certo número de elementos de processamento, cada elemento operando sobre um dado pertencente a um fluxo de dados diferente
  - Ex.: supercomputadores vetoriais, que realizam em paralelo uma mesma operação sobre múltiplos elementos de um vetor

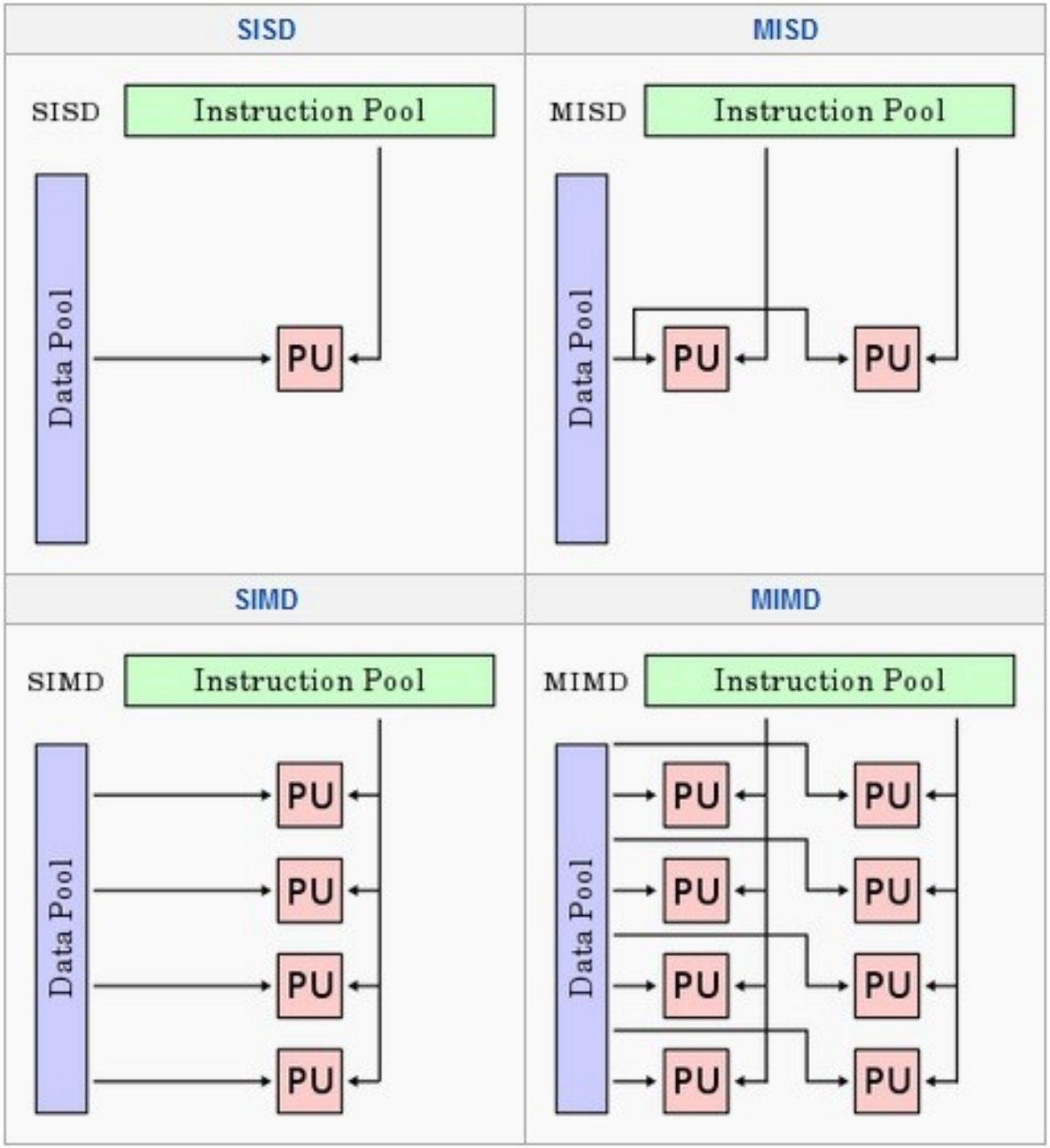
Sistemas Paralelos

- MISD – *Multiple Instruction, Single Data*: uma única sequência de dados é transmitida para um conjunto de processadores, cada um dos quais executa uma sequência de operações diferentes. Na prática, ainda não existe nenhuma estrutura deste tipo implementada

- MIMD – *Multiple Instruction, Multiple Data*: um conjunto de processadores executa simultaneamente sequências diferentes de instruções, sobre conjuntos de dados distintos. Sistemas de memória compartilhada ou distribuída (como clusters) são classificados desta forma

Sistemas Paralelos

Classificação	Nº de fluxos de instrução	Nº de fluxos de dados	Exemplos de sistemas reais
SISD	1	1	sistemas convencionais
SIMD	1	Múltiplos	supercomputadores vetoriais
MISD	múltiplos	1	não existe
MIMD	múltiplos	múltiplos	multiprocessadores


SISD
SISD
Instruction Pool
Data Pool
PU
MISD
MISD
Instruction Pool
Data Pool
PU
PU
SIMD
SIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
MIMD
MIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
PU
PU
PU
PU