Computação paralela:
Dados únicos de instrução única
Dados múltiplos de instrução única
Dados únicos de várias instruções
Múltiplos dados de instrução múltipla

Modo real e modo protegido

1

computação paralela

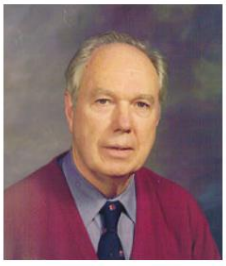
- **A computação paralela** consiste em dividir problemas maiores em partes menores independentes e semelhantes que podem ser executadas simultaneamente. •

Cada parte é subdividida em uma série de instruções.

- As instruções de cada parte são executadas simultaneamente em diferentes CPUs.
- Os sistemas paralelos lidam com o uso simultâneo de vários recursos de computador que podem incluir um único computador com vários processadores se comunicando usando memória compartilhada.

2

Taxonomia de Flynn



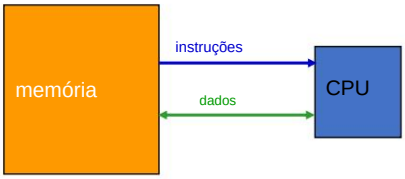
- Michael Flynn em 1966 apresentou uma taxonomia para categorizando diferentes estilos de sistema de computador arquitetura no seguinte artigo:
- ***MJ Flynn, "Sistemas de computação de velocidade muito alta", em Proceedings of the IEEE, vol. 54, nº. 12, pp. 1901-1909, dez. 1966, doi: 10.1109/PROC.1966.5273.***

- **Fluxo de instrução único, fluxo de dados único (SISD)**
- **Fluxo de instrução única, fluxo de dados múltiplos (SIMD)**
- Fluxo de instruções múltiplas, fluxo de dados único (MISD)
- Múltiplos fluxos de instruções, múltiplos fluxos de dados (MIMD).

3

SISD (Single Instruction Single Data)

- Um único fluxo serial de instruções opera sobre dados (princípio clássico de von-Neumann).
- Fluxo de dados sequencial e uma única unidade de processamento para executar o fluxo de dados.



Clássico:
IBM-PC, IBM 370,
DEC Micro-VAX,…

4

SISD (Single Instruction Single Data)

As vantagens: •

Requer menos energia. • Não

há problema de protocolo de comunicação complexo entre vários núcleos.

As desvantagens: • A

velocidade da arquitetura SISD é limitada assim como single-core processadores.

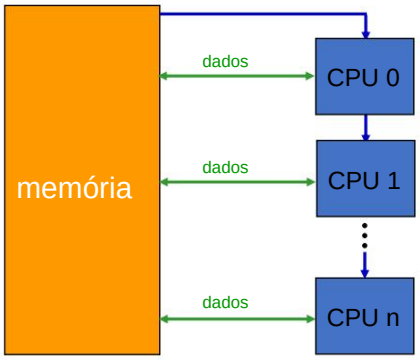
• Não é adequado para aplicações maiores.

5

SIMD (dados múltiplos de instrução única)

• Todos os processadores executam as mesmas instruções em dados diferentes (processador de matriz)

• Sistema multiprocessador executando a mesma instrução em todas as CPUs, mas operando em diferentes fluxos de dados. instruções



Exemplo de processamento de imagem: cada processador opera em uma parte da imagem

6

SIMD (dados múltiplos de instrução única)

As vantagens: • A mesma operação em vários elementos pode ser realizada usando apenas uma instrução. • O rendimento do sistema pode ser aumentado aumentando o número de núcleos do processador.

• A velocidade de processamento é superior à arquitetura SISD.

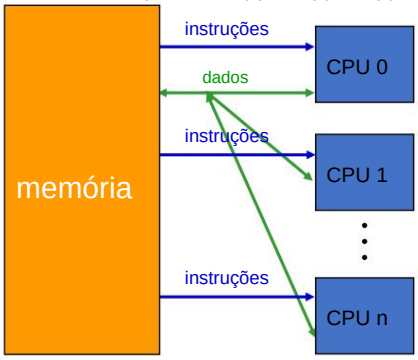
As desvantagens: • Existe uma comunicação complexa entre números de núcleos de processador.

• O custo é maior do que a arquitetura SISD.

7

MISD (Dados Únicos de Instrução Múltipla)

• Todos os processadores executam instruções diferentes nos mesmos dados. • Executa diferentes operações nos mesmos dados. • Exemplo Z = sin(x)+cos(x)+tan(x)

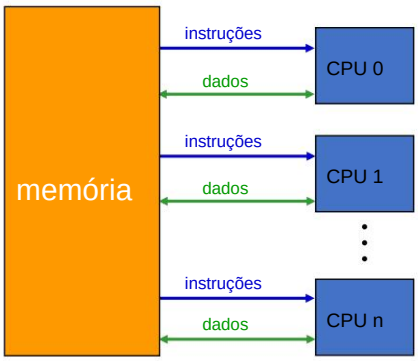


O modelo MISD não é útil na maioria das aplicações. Portanto, poucas máquinas são construídas, mas nenhuma delas está disponível comercialmente.

8

MIMD (Dados Múltiplos de Instrução Múltipla)

- Todos os processadores executam instruções diferentes em dados diferentes
- Uma máquina multiprocessadora que pode executar várias instruções em vários dados.



Clássico:

IBM 3084, Cray-2,

Hoje:

A maioria dos sistemas de computador são muitos processadores de núcleo, têm componentes especializados operando em paralelo etc.

9

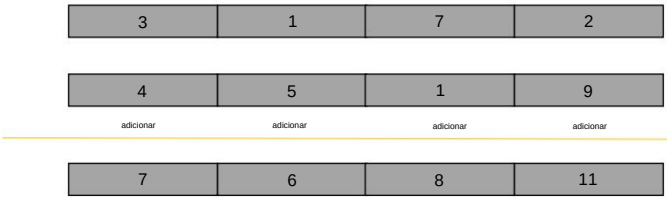
SIMD: dados múltiplos de instrução única

Todas as instruções referidas nas aulas anteriores, utilizam dados únicos para realizar a Ação. O tipo dessas instruções pode ser projetado por **Single Instruction, Single Dados (SISD).**

Isso significa que se você precisar fazer a mesma ação, por exemplo, adicionar, em quatro elementos consecutivos, a instrução add deve ser chamada quatro vezes, uma para cada elemento.

As operações são **serializadas**, o que significa que uma operação é feita após a operação anterior.

O tempo necessário para fazer as quatro operações pode ser quatro vezes maior que o tempo necessidade de uma única operação.



10

SIMD: dados múltiplos de instrução única

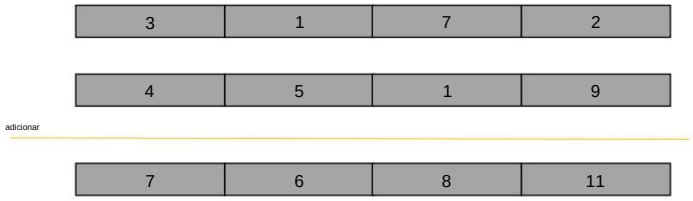
Normalmente, é necessário fazer a mesma operação em vários dados.

O SIMD é uma solução para diminuir o tempo necessário para realizar essas ações

Uma única instrução pode fazer a mesma operação em vários dados

As operações em cada dado são **paralelizadas**, o que significa que todas as operações iniciam e terminar ao mesmo tempo.

O tempo necessário para realizar as quatro operações pode ser o mesmo para realizar uma única Operação.



11

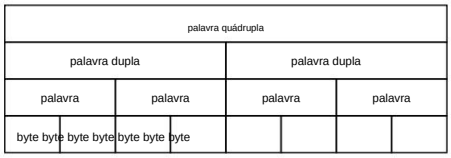
Dados Múltiplos de Instrução Única

SIMD requer registradores MMX, XMM, YMM ou ZMM.

Esses registradores, com tamanho entre 64 bits (MMX) e 512 bits (ZMM), podem representar muitos tipos de dados

Os registradores MMX podem representar 8 bytes, 4 palavras, 2 palavras duplas ou 1 palavra quádrupla

Registro MMX



O significado exato dos dados em cada registrador depende apenas da instrução usado. As instruções são diferentes para cada tipo de dados. Por exemplo, a instrução para adicionar bytes é diferente da instrução para adicionar palavras ou adicionar palavras duplas!

12

Dados Múltiplos de Instrução Única

Os registradores XMM (128 bits) podem representar

- 2 flutuadores duplos
- 4 carros alegóricos
- 2 palavras quádruplas
- 4 palavras duplas
- 8 palavras
- 16 bytes

Registradores XMM (128 bits)

13

Dados Múltiplos de Instrução Única

Registros YMM (256 bits) = XMM *2

Registradores ZMM (512 bits) = YMM*2

14

Dados Múltiplos de Instrução Única
Instruções para mover dados

A instrução mov não pode ser usada com registradores mmx\xmm\ymm\zmm

Existem várias instruções para mover dados.

2 dessas instruções são:

destino movdqa , *fonte*

destino movdqu , *fonte*

Ambas as instruções movem a origem para o destino.

Origem e destino podem ser registros xmm ou memória

A memória não pode ser usada simultaneamente na origem e no destino movdqa

é mais rápido que movdqu, mas movdqa, quando usado com memória, implica que a memória deve ser alinhada em 16 bytes

(o endereço deve ser múltiplo de 16)

15

Dados Múltiplos de Instrução Única

- Certas instruções SIMD, que executam a mesma instrução em vários dados, exigem que o endereço de memória desses dados esteja alinhado a um determinado limite de byte.
  Isso significa efetivamente que o endereço da memória em que seus dados residem precisa ser divisível pelo número de bytes exigidos pela instrução.

- Portanto, o alinhamento de 16 bytes (128 bits) significa que o endereço de memória de seus dados precisa ser um múltiplo de 16.

Por exemplo, 0x00010 seria 16 bytes alinhados, enquanto 0x00011 não seria.

16

Dados Múltiplos de Instrução Única
Exemplos

movdqa xmm0,xmm1; Mova xmm1 para xmm0 movdqa
xmm0,1234; Instrução inválida. Valores não podem ser usados movdqa xmm0,
[esi]; mover 16 bytes de memória iniciados em esi para xmm0 movdqa [esi],xmm0; mover
xmm0 para a memória iniciada em esi

Em ambos os casos, se esi não for múltiplo de 16, ocorrerá um erro de tempo de execução

Nota: Em C você pode forçar o compilador a alinhar os dados usando o seguinte código
__declspec(align(16)) //diretiva para alinhar a próxima variável em 16 bytes char
data_aligned[16]={0,1,2,3,4 ,5,6,7,8,9,10,11,12,13,14,15};

Agora você pode usar a instrução movdqa xmm0,data_aligned;

17

Dados Múltiplos de Instrução Única
Exemplos

movdqu xmm0,xmm1; Mover xmm1 para xmm0

movdqu xmm0,1234; Instrução inválida. Valores não podem ser usados

movdqu xmm0,[esi]; mover 16 bytes de memória iniciados em esi para xmm0

movdqu[esi],xmm0; mover xmm0 para a memória iniciada em esi

em ambos os casos esi pode ter qualquer valor

18

Dados Múltiplos de Instrução Única
Instruções

Existem centenas de instruções para SIMD

Você não precisa saber de memória nenhuma instrução, mas precisa saber como usá-la se tiver acesso a um manual

Existem muitas informações na rede sobre instruções SIMD.

Você pode encontrar o melhor para você…

19

Dados Múltiplos de Instrução Única
Instruções

O nome de cada instrução pode ser representado pelo seguinte diagrama

Fonte: https://upload.wikimedia.org/wikipedia/commons/e/e2/MMX_instrucao.png

20

Dados Múltiplos de Instrução Única
Exemplos

paddb xmm0,xmm1; Adicionar Byte Compactado

Adicione cada um dos 16 bytes do registro xmm0 a xmm1 e armazene o resultado em xmm0

paddw xmm0,xmm1; Adicionar palavra empacotada

Adicione cada uma das 8 palavras do registrador xmm0 a xmm1 e armazene o resultado em xmm0

paddd xmm0,xmm1; Empacotado Adicionar palavra dupla

Adicione cada uma das 4 palavras duplas do registrador xmm0 a xmm1 e armazene o resultado em xmm0

paddq xmm0,xmm1; Empacotado Adicionar palavra quádrupla

Adicione cada uma das 2 palavras quádruplas do registro xmm0 a xmm1 e armazene o resultado em xmm0

21

Dados Múltiplos de Instrução Única
Exemplos

paddsb xmm0,xmm1; Byte de saturação de adição compactado

Adicione cada um dos 16 bytes assinados do registrador xmm0 ao xmm1 e armazene o resultado com saturação em xmm0

paddsw xmm0,xmm1; Empacotado Adicionar Palavra de Saturação

Adicione cada uma das 8 palavras assinadas do registrador xmm0 a xmm1 e armazene o resultado com saturação em xmm0

paddusb xmm0,xmm1; Byte de saturação sem sinal de adição compactado

Adicione cada um dos 16 bytes não assinados do registrador xmm0 a xmm1 e armazene o resultado com saturação em xmm0

paddusw xmm0,xmm1; Empacotado Adicionar palavra de saturação não assinada

Adicione cada uma das 8 palavras sem sinal do registrador xmm0 a xmm1 e armazene o resultado com saturação em xmm0

22

Memória de endereçamento

Até a CPU 80286, apenas um modo, denominado Real Mode, era usado para endereçar a memória

O Modo Real teve dois grandes problemas:

1. Só pode endereçar memória até 1 MB (2^20)
2. Não implementa nenhum tipo de proteção de memória, o que significa que qualquer aplicativo pode ler e/ou escrever em qualquer endereço, mesmo no endereço do sistema operacional

No início, todas as CPUs, mesmo as mais recentes, rodam em Modo Real.

Isso significa que o código inicial no BIOS é executado no modo real.

23

Modo protegido

A CPU 80286 introduz o Modo Protegido.

No entanto, este modo ganhou maior importância apenas no 386, a primeira CPU de 32 bits.

As principais vantagens são:

1. Grande capacidade de endereçamento
2. Proteção de memória por hardware.
   Em teoria, os aplicativos só podem usar sua própria memória **(exceto se o programador for um hacker…)**
3. Definir, em hardware, 4 níveis de privilégios de aplicativos
4. Pode ser usado para implementar **memória virtual**

24

Modo protegido

O endereço é sempre formado por dois registradores, o seletor e o offset

O seletor é representado por registradores de segmento

- CS para representar o segmento de código. O código do programa está sempre no Segmento de Código.
- CS:EIP aponta para a próxima instrução a ser executada
- SS para representar o Segmento da Pilha. A pilha está sempre no segmento de pilha.
- SS:ESP aponta para o elemento no topo da pilha
- DS, ES, FS e GS para representar dados, não definidos na pilha

O deslocamento pode ser representado por quaisquer outros registradores.

25

Modo protegido

Ao programar em *modelo plano* no Windows, todos os registradores de segmento permanecem inalterados.

Neste modo o programador apenas usa o offset para endereçar qualquer byte na memória

A memória máxima que um aplicativo pode usar neste modo é de 4 GB (2^32). – sistemas x86 (32 bits)

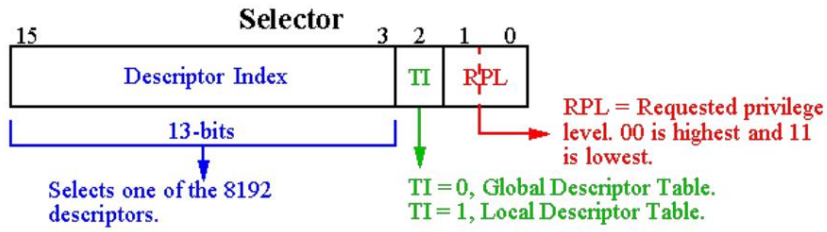
Internamente a CPU **sempre** usa o Seletor correspondente para endereçar qualquer byte ou instrução.

Nota: Modelo de memória plana ou modelo de memória linear refere-se a um paradigma de endereçamento de memória no qual "a memória aparece para o programa como um único espaço de endereço contíguo. A CPU pode endereçar diretamente (e linearmente) todos os locais de memória disponíveis sem ter que recorrer a nenhum tipo de segmentação de memória ou esquemas de paginação.

26

***Registos de Segmento em* Modo Protegido**

• *Interpretação:*



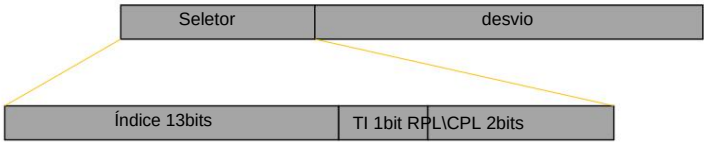
• Índice de Descritores e Índice de Tabelas (TI) :

O índice descritor de 13 bits seleciona um de até 8 K descritores em GDT e LDT, conforme especificado pelo bit TI.
Portanto, esses 14 bits permitem o acesso a descritores de 16K de 8 bytes.

GDT – tabela de descritor global (uma para todos os programas)
LDT – tabela de descritor local (tipicamente um para cada programa)

27

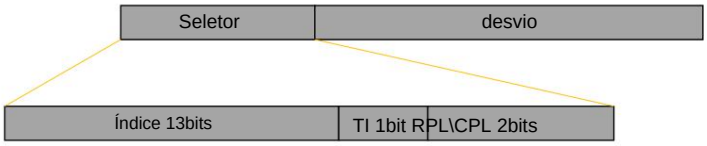
Modo protegido



RPL\CPL representa um dos quatro níveis de privilégio
- 00 o maior nível de privilégio. É o nível para o sistema operacional
- 01 não utilizado no sistema operacional Windows
- 10 não utilizado no sistema operacional Windows
- 11 o nível de privilégio menor. Usado por aplicativos

28

Modo protegido



No caso do endereço formado por CS:EIP, que aponta para a próxima instrução a ser executada, os bits menos significativos de CS (CPL: Current Privilege Level) representa a prioridade do aplicativo

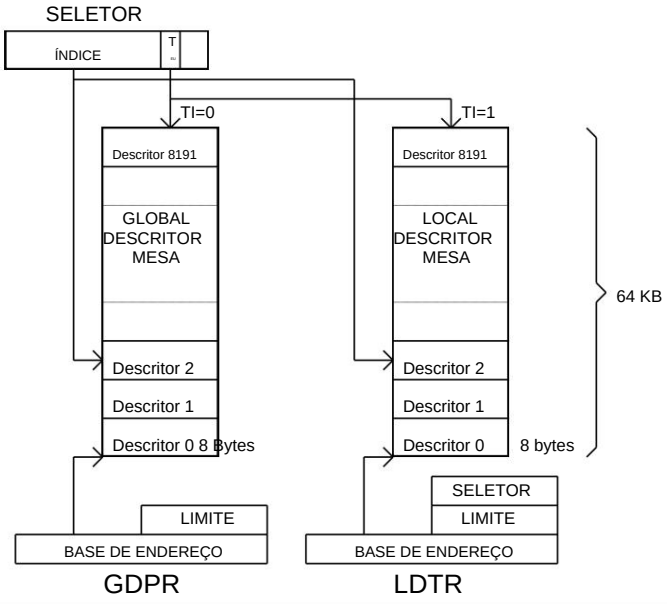
Isso significa que a CPU conhece o privilégio de qualquer instrução e pode executá-la ou não de acordo com seu privilégio

No caso de qualquer outro endereço, que aponte para algum dado, o seletor de bits menos significativos (RPL: Request Privilege Level) representa a prioridade deste dado.

Isso significa que um programa para usar esses dados deve ter, no mínimo, a mesma prioridade dos dados.

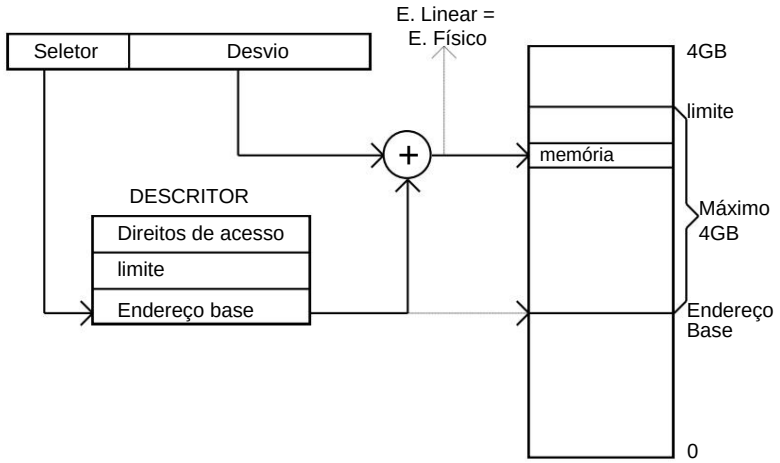
29

Modo protegido



30

Modo protegido



31

Resumindo: Modo Protegido

Cada ponteiro para uma zona de memória tem associado um limite, o que significa que qualquer tentativa de usar qualquer byte fora desse limite é negada.

A memória tem seu próprio nível de privilégio. Apenas programas com o mesmo ou alto nível de privilégio podem ler/escrever esta memória A

memória pode ser definida como leitura/escrita ou apenas

leitura Programas rodando com privilégios de baixo nível não podem mudar os atributos da memória

Somente programas executados com privilégio de alto nível (SO) podem alterar os atributos de memória

Se um programa ganha privilégio de alto nível pode fazer tudo…

32