Parallel computing:
Single instruction single data
Single instruction multiple data
Multiple instruction single data
Multiple instruction multiple data

Real mode and Protected mode

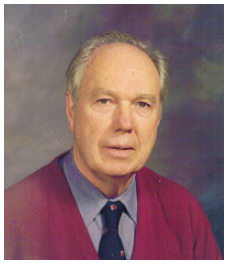
1

Parallel computing

- **Parallel computing** consists of breaking down larger problems into smaller independent and similar parts that can be executed simultaneously.
- Each part is further broken down to a series of instructions.
- Instructions from each part execute simultaneously on different CPUs.
- Parallel systems deal with the simultaneous use of multiple computer resources that can include a single computer with multiple processors communicating using shared memory.

2

Flynn's taxonomy



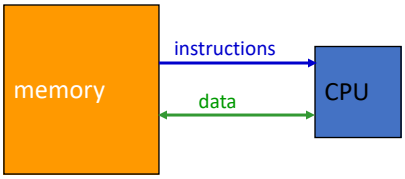
- Michael Flynn in 1966 presented a taxonomy for categorizing different styles of computer system architecture in the following paper:
- ***M. J. Flynn, "Very high-speed computing systems," in Proceedings of the IEEE, vol. 54, no. 12, pp. 1901-1909, Dec. 1966, doi: 10.1109/PROC.1966.5273.***

- **Single instruction stream, single data stream (SISD)**
- **Single instruction stream, multiple data stream (SIMD)**
- Multiple instruction stream, single data stream (MISD)
- Multiple instruction stream, multiple data stream (MIMD).

3

SISD (Single Instruction Single Data)

- A single serial stream of instructions operates on data (classical von-Neumann principle).
- Sequential data stream and one single processing unit to execute the data stream.



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

4

SISD (Single Instruction Single Data)

The advantages:

- It requires less power.
- There is no issue of complex communication protocol between multiple cores.

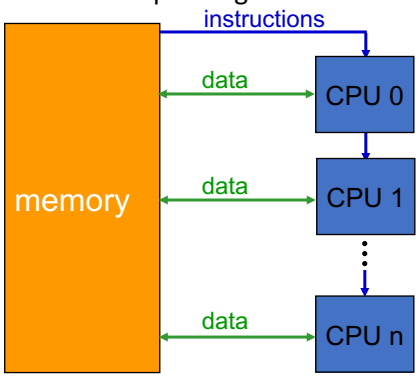
The disadvantages:

- The speed of SISD architecture is limited just like single-core processors.
- It is not suitable for larger applications.

5

SIMD (Single Instruction Multiple Data)

- All processors perform the same instructions on different data (array processor)
- Multiprocessor system executing the same instruction on all the CPUs but operating on different data streams.



Example image processing: each processor operates on a part of the picture

6

SIMD (Single Instruction Multiple Data)

The advantages:

- Same operation on multiple elements can be performed using one instruction only.
- Throughput of the system can be increased by increasing the number of cores of the processor.
- Processing speed is higher than SISD architecture.

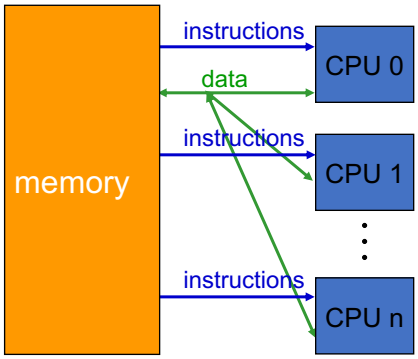
The disadvantages:

- There is complex communication between numbers of cores of processor.
- The cost is higher than SISD architecture.

7

MISD (Multiple Instruction Single Data)

- All processors perform different instructions on same data.
- Performs different operations on the same data.
- Example Z = sin(x)+cos(x)+tan(x)

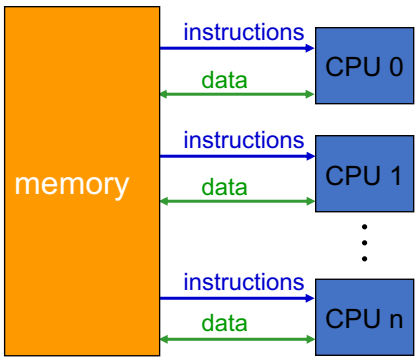


The MISD model are not useful in most of the applications. Therefore, few machines are built, but none of them are available commercially.

8

MIMD (Multiple Instruction Multiple Data)

- All processors perform different instructions on different data
- A multiprocessor machine which can execute multiple instructions on multiple data.



Classical:
IBM 3084, Cray-2,

Today:

Most computer systems are many core processors, have specialized components operating in parallel etc.

9

SIMD: Single Instruction Multiple Data

All the instructions referred in the previous classes, use single data to perform the action. The type of these instructions can be designed by **Single Instruction, Single Data (SISD).**

This means that if you need make the same action, for example an add, in four consecutives elements, the add instruction must be call four times, one for each element.

The operations are **serialized**, which means that one operation is made after the previous operation.

The time need to make the four operations can be four times more than the time need for one single operation.

3	1	7	2
4	5	1	9
add	add	add	add
7	6	8	11

10

SIMD: Single Instruction Multiple Data

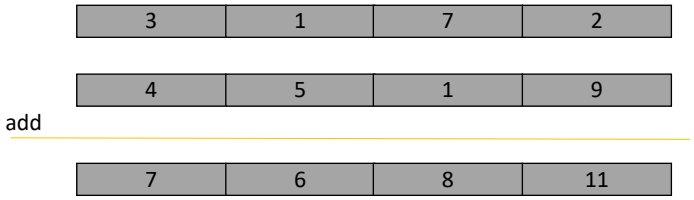
Usually, it is necessary to make the same operation in multiple data.

SIMD is a solution to decrease the time needed to make these actions

One single instruction can make the same operation in multiple data

The operations in each data are **parallelized** which means that all operations start and finish at the same time.

The time need to perform the four operations can be the same to perform one single operation.



11

Single Instruction Multiple Data

SIMD requires MMX, XMM, YMM or ZMM registers.

These registers, with size between 64bits (MMX) and 512bits (ZMM), can represent many types of data

The MMX registers can represent 8 bytes, 4 words, 2 double words or 1 quad word

MMX Register

quadword							
doubleword				doubleword			
word		word		word		word	
byte	byte	byte	byte	byte	byte	byte	byte

The exact meaning of the data in each register is just dependent of the instruction used. Instructions are different for each data type. For example, the instruction to add bytes is different than instruction to add words or to add double words!

12

Single Instruction Multiple Data

The XMM registers (128bits) can represent

- 2 double floats
- 4 floats
- 2 quad words
- 4 double words
- 8 words
- 16 bytes

XMM registers(128 bits)

<table>
<tr><td colspan="8">double float</td><td colspan="8">double float</td></tr>
<tr><td colspan="4">float</td><td colspan="4">float</td><td colspan="4">float</td><td colspan="4">float</td></tr>
<tr><td colspan="8">quadword</td><td colspan="8">quadword</td></tr>
<tr><td colspan="4">doubleword</td><td colspan="4">doubleword</td><td colspan="4">doubleword</td><td colspan="4">doubleword</td></tr>
<tr><td colspan="2">word</td><td colspan="2">word</td><td colspan="2">word</td><td colspan="2">word</td><td colspan="2">word</td><td colspan="2">word</td><td colspan="2">word</td><td colspan="2">word</td></tr>
<tr><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td><td>byte</td></tr>
</table>

13

Single Instruction Multiple Data

YMM registers (256 bits) = XMM *2

ZMM registers (512 bits) = YMM*2

14

Single Instruction Multiple Data
Instructions to move data

The instruction mov can't be used with registers mmx\xmm\ymm\zmm

There are various instructions to move data.

2 of these instructions are:

movdqa *destination, source*

movdqu *destination, source*

Both instructions move the source to destination.

Source and destination can be xmm registers or memory

Memory can't be used simultaneously in source and destination

movdqa is faster than movdqu, but movdqa, when used with memory, implies that memory must be aligned at 16 bytes

(the address must be multiple of 16)

15

Single Instruction Multiple Data

- Certain SIMD instructions, which perform the same instruction on multiple data, require that the memory address of this data is aligned to a certain byte boundary. This effectively means that the address of the memory your data resides in needs to be divisible by the number of bytes required by the instruction.

- So, the alignment of 16 bytes (128 bits) means the memory address of your data needs to be a multiple of 16.

E.g. 0x00010 would be 16 byte aligned, while 0x00011 would not be.

16

Single Instruction Multiple Data Examples

movdqa xmm0,xmm1; Move xmm1 to xmm0
movdqa xmm0,1234; Invalid instruction. Values cant be used
movdqa xmm0,[esi]; move 16 bytes of memory started at esi to xmm0
movdqa [esi],xmm0; move xmm0 to memory started at esi

In both case if esi isn't multiple of 16 a run time error occur

Note: In C you can force the compiler to align data using the following code
__declspec(align(16)) //directive to align the next variable at 16 bytes
char data_aligned[16]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

Now you can use the instruction movdqa xmm0,data_aligned;

17

Single Instruction Multiple Data Examples

movdqu xmm0,xmm1; Move xmm1 to xmm0
movdqu xmm0,1234; Invalide instruction. Values cant be used
movdqu xmm0,[esi]; move 16 bytes of memory started at esi to xmm0
movdqu [esi],xmm0; move xmm0 to memory started at esi
in both cases esi can have any value

18

Single Instruction Multiple Data Instructions

There are hundreds of instructions for SIMD

You don't need to know by memory any instruction, but you need to know how to use it if you have access to a manual

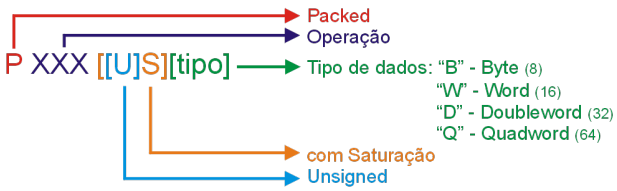
There are many information in the net about SIMD instructions.

You can find the best for you…

19

Single Instruction Multiple Data Instructions

The name of each instruction can be represented by the following diagram



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

20

Single Instruction Multiple Data Examples

paddb xmm0,xmm1; Packed Add Byte
Add each of 16 bytes from register xmm0 to xmm1 and store the result in xmm0

paddw xmm0,xmm1; Packed Add Word
Add each of 8 word from register xmm0 to xmm1 and store the result in xmm0

paddd xmm0,xmm1; Packed Add Double word
Add each of 4 double word from register xmm0 to xmm1 and store the result in xmm0

paddq xmm0,xmm1; Packed Add Quad word
Add each of 2 quad word from register xmm0 to xmm1 and store the result in xmm0

21

Single Instruction Multiple Data Examples

paddsb xmm0,xmm1; Packed Add Saturation Byte
Add each of 16 signed bytes from register xmm0 to xmm1 and store the result with saturation in xmm0

paddsw xmm0,xmm1; Packed Add Saturation Word
Add each of 8 signed word from register xmm0 to xmm1 and store the result with saturation in xmm0

paddusb xmm0,xmm1; Packed Add Unsigned Saturation Byte
Add each of 16 unsigned bytes from register xmm0 to xmm1 and store the result with saturation in xmm0

paddusw xmm0,xmm1; Packed Add Unsigned Saturation Word
Add each of 8 unsigned words from register xmm0 to xmm1 and store the result with saturation in xmm0

22

Addressing memory

Until the 80286 CPU, only one mode, named Real Mode, was used to address memory

Real Mode had two big problems:

1. Only can address memory until 1MB (2^20)
2. Doesn't implement any kind of memory protection, which means that any application can read and\or write in any address, even in the address of the operation system

At start, all CPUs, even the most recent, runs in Real Mode.

This means that the initial code in the BIOS runs in Real Mode.

23

Protected Mode

The 80286 CPU introduce the Protected Mode.

However, this mode gain increased importance just in the 386, the first 32 bits CPU.

The main advantages are:

1. Large addressing capacity
2. Protection of memory by hardware.
   In theory, the applications just can use your own memory **(except if the programmer is a hacker…)**
3. Define, in hardware, 4 levels of applications privileges
4. Can be used to implement **virtual memory**

24

Protected Mode

The address is always formed by two registers, the selector and the offset

The selector is represented by segment registers

- CS to represent the Code Segment. The program code is always in the Code Segment.
- CS:EIP points to the next instruction to be executed
- SS to represent the Stack Segment. The stack is always in the Stack Segment.
- SS:ESP points to the element in the stack top
- DS, ES, FS and GS to represent data, not defined in the stack

The offset can be represented by any other registers.

25

Protected Mode

When programming in *flat model* in Windows, all the segment registers remains unchanged.

In this mode the programmer just use the offset to address any byte in memory

The max memory that an application can use in this mode is 4GB (2^32). – x86 systems (32 bits)

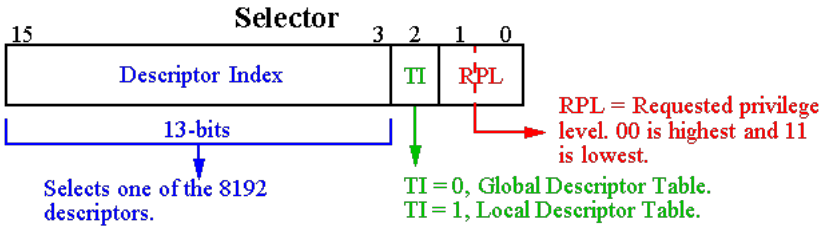
Internally the CPU **always** use the correspondent Selector to address any byte or instruction.

Note: Flat memory model or linear memory model refers to a memory addressing paradigm in which "memory appears to the program as a single contiguous address space. The CPU can directly (and linearly) address all the available memory locations without having to resort to any sort of memory segmentation or paging schemes.

26

***Segment Registers in* Protected Mode**

• *Interpretation:*



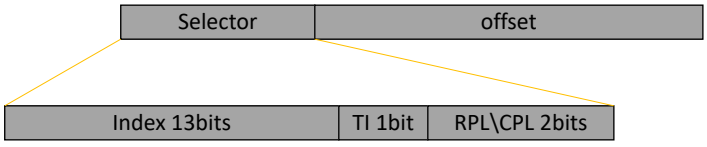
•**Descriptor Index and Table Index (TI)** *:*

The 13-bit descriptor index selects one of up to 8K descriptors in either the GDT and LDT, as specified by the TI bit.
Therefore, these 14 bits allows access to 16K 8-byte descriptors.

GDT – global descriptor table (one for all programs)
LDT – local descriptor table (typically one for each program)

27

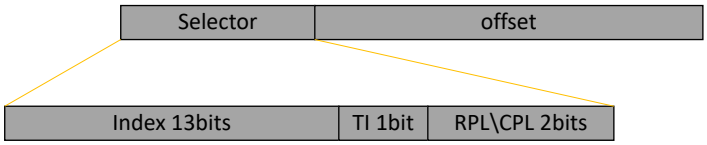
Protected Mode



RPL\CPL represents one of the four levels of privilege
- 00 the most privilege level. Is the level for the Operating System
- 01 unused in Windows Operating System
- 10 unused in Windows Operating System
- 11 the less privilege level. Used by applications

28

Protected Mode



In the case of the address formed by CS:EIP, that points to the next instruction to be executed, the CS less significant bits
(CPL: Current Privilege Level) represents the application priority

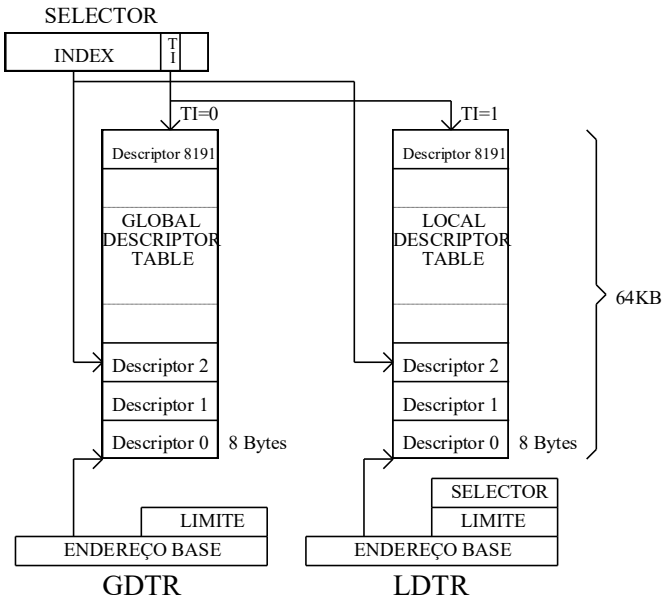
This means that CPU know the privilege for any instruction, and can execute it or not according her privilege

In the case of any other address, that points to any data, the selector less significant bits (RPL: Request Privilege Level) represents the priority of this data.

This means that a program to use this data must have, at least, the same priority of the data.

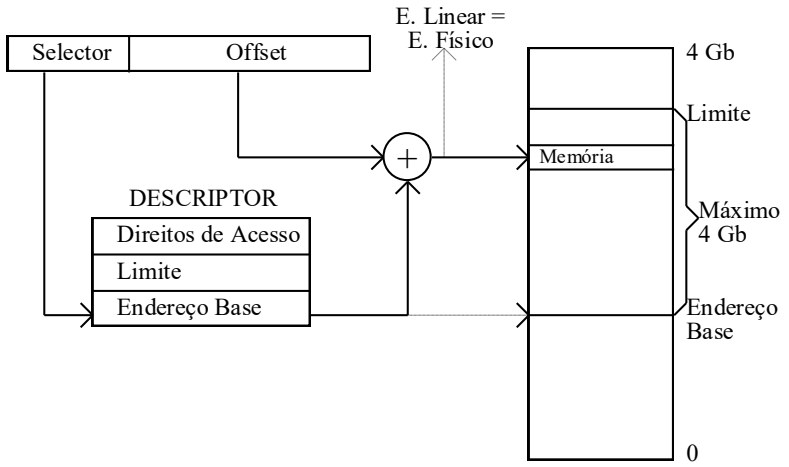
29

Protected Mode



30

Protected Mode



31

In summary: Protected Mode

Each pointer to a memory zone have associate a limit, which means that any attempt to use any byte outside this limit is denied.

Memory have your own privilege level. Only programs with the same or high level of privilege can read\write this memory

Memory can be defined to be read\write or just read

Programs running at low level privilege can't change the memory attributes

Only programs that run at high level privilege (OS) can change the memory attributes

If a program gain high level privilege can do everything…

32