POLITECNICO
DI TORINO

Recap Computer Architectures (02LSEOV)

Jacopo Nasi
Computer Engineer
Politecnico di Torino

I Period - 2017/2018

20 ottobre 2017

Indice

License

Acknowledgments

Questo breve riepilogo non ha alcuno scopo se non quello di agevolare lo studio di me stesso, se vi fosse di aiuto siete liberi di usarlo.
Le fonti su cui mi sono basato sono quelle relative al corso offerto (**Computer Architectures (02LSEOV)**) dal Politecnico di Torino durante l'anno accademico 2017/2018.
Non mi assumo nessuna responsabilità in merito ad errori o qualsiasi altra cosa. Fatene buon uso!

1 Introduction Computer Design

1.1 Computer Evolution

The first general-purpose computer was created in the late 40s. What now we can buy for 500$ is equivalent (performance) to what could be bought for about $1M in 85'.
During the years the performance growth was not linear, as you can see in figure 1, during the first 10 years the annual increase was around 25-30%/year, from the late 80s to the 2000 the growth is increased around 50%/year and, in the last few years it decrease to the 22%. Why this change during the increase?
The manufacturers have found a lot of physical problem related to the creation

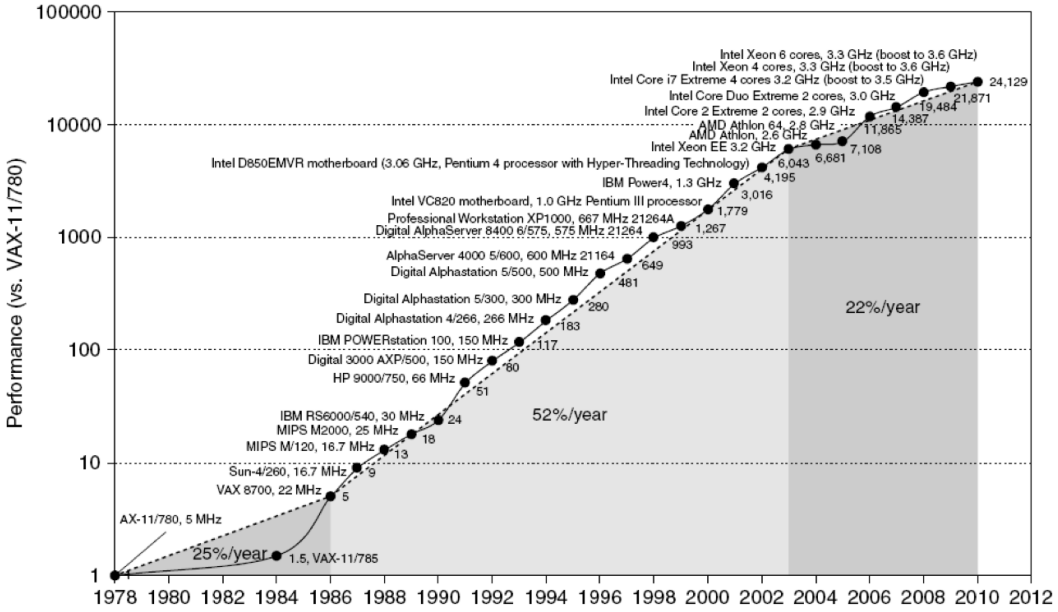

Figura 1: CPUs Growth over years

of new products, this problem are mainly related to:

- Power-Issue.
- Lower instruction-level parallelism.
- Unchanged memory lantecy.

in fact, since 2004, the major industry have changed the conceptual ways to desing processors, switching from single to multi-core architectures. We can say that, in anytime, this growth is incredible an is due to improvements in technology, microprocessor architecture and software development. Since the multi core introduction the major prefers to investe on multicore system rather than develop faster CPU.

1.2 Designing

There are 5 main market areas:

- **Personal Mobile Device** (PMD): Smarthphone, tablet. They are focused in energy efficency and real-time app.
- **Desktop Computing**: From PC to workstation and the main pourpose is optimize the price-performance ratio.
- **Server**: Larger-scale and more reliable computing services.
- **Cluster - WAS**:Emphasis on availability, price-performance and power consumption.
- **Embedded Computers**: Fastest growing portion of PCs market. All special-purpose computer-based application, from cheap to high-end processors.

There are to **Classes of Parallelism**:

- **Data-Level** (DLP): Many data items that can be operated on at the same.
- **Task-Level** (TaskLP): Many task of a work can operate independently.

The first solution allow the processor to split the data operation over multiple cores, you can for example divide an array of n elements over 4 core and, if T is the computational time needed for the entire array, the final time will be T/4 plus a little time for the reunion of the data. It split the one task on different data.
The TLP instead is able to manage multiple task over the same data, this is the common behaviour of the actual system (pipelining techniques).
There are different Parallel Architectures:

- **Instruction-level** (ILP): The is modestly use the data-level parallelism.
- **Vector and GPU**: It exploits DLP.
- **Thread-level** (TLP): It exploits DLP and TaskLP.
- **Request-level** (RLP): Exploits parallelism among decoupled (not-related) tasks.

The designing of a new computer involve important analysis to the main pourpose of it, you need to study which attributes are important for the new machine and you need to maximizes performance and matching cost and power constraints. During the last decade the PC design took advantage of architectural and technology improvements, the performance increase is more than a

factor of 15 on what would have been obtained by relying solely on technology.

The **Moore's Law** says that: *The number of transistors that can be integrated into a single chip doubles every 18/24 months.* Until now the law has worked, as you can see in the figure 2 and, probably, it will work for other time.

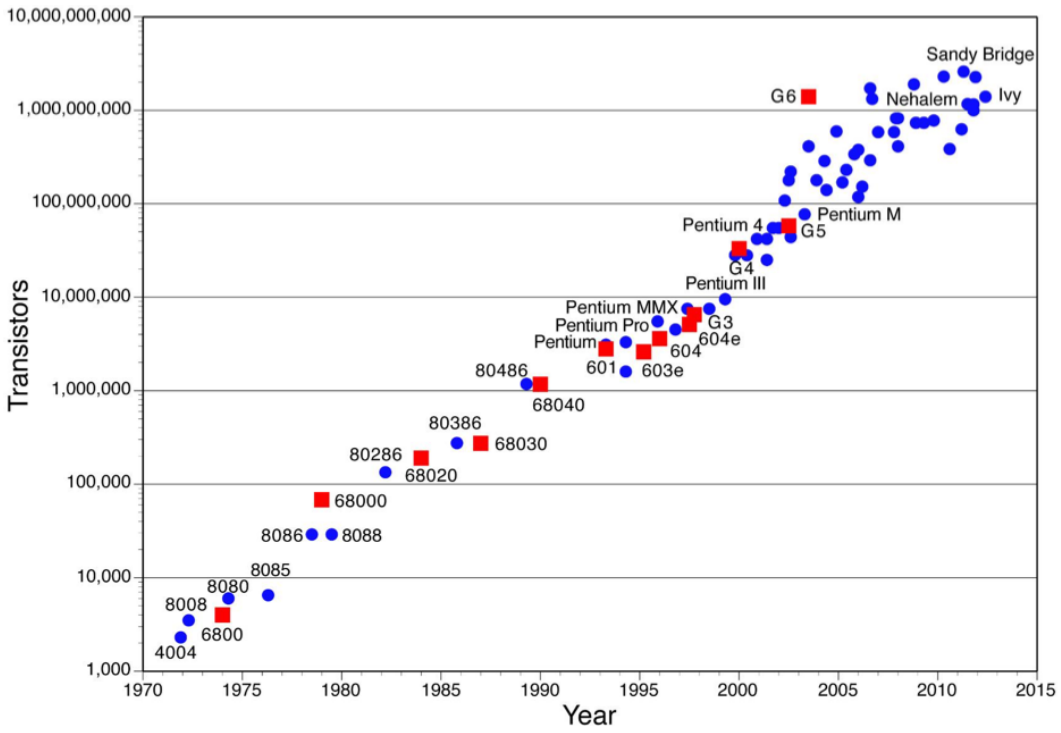

Figura 2: Transistors growth on CPUs

Cost During the evaluation of the IC manufacturing cost it is important not to forget the impact of yeild, the percentage of products that pass the test phase. The production process for every product undergoes an evolution which normally leads to an improvement in yield (learning curve). The cost decrease is due to yield increase. Probably the most important part of the manufacturing cost is related to the validation and testing procedures.

Other designing problem The continuos increase of the system complexity and device integration leads to porblem with power consumption, for the Power and for the Energy (manly for portable devices). Until now the greater power consumption contribution is due to the trasnsistor switching phase, related to the physical formulas the solution applied is trying to reduce the voltage.
Another important factor is the **Dependability** that is the quality of the

system to deliver a correct service, is traditionally very high, but it can be lowered by software or hardware bugs both during the production or in designing phase. During this years the safety-critical areas where the microprocessor have relevant importance are increased, initialy only space, avionics and nuclear, but now we have rail-road, automotive, biomedical, telecom, ecc...
The dependability is often measured using:

- **Mean Time To Failure** (MTTF): 1 failure in one billion hours.
- **Mean Time Between Failures** (MTBF)
- **Mean Time To Repair** (MTTR)

This three measures are related by the formula: $MTBF = MTTF + MTTR$

Performance they be analyzed under multiple point of view, *Time between start and completion of an operation, Total amount of work done in time unit, ecc....* The UNIX system provide 4 different values:

- Elapsed time
- CPU Time
  - User
  - System

The evaluation of often performed letting work a computer and observing its behavior. Unfortunately, the choice of the application severely affects the performance and is not too easy looking the result in a correct way. The main solution is using benchmarks to compare different system running the same operation and production comparable times of execution. The benchmark sets are normally composed of:

- Kernels
- Program Fragments
- Applications

There are multiples solution to analyze the performance one of the most common is the **Amdahl's law** and it based to the comparisons with the older version of the same product. The speedup is calculated by this formula:

$$Speedup = \frac{Performance_{enhancement}}{Performance_{NOenhancement}} \tag{1}$$

the value depends on two factors:

- $Fraction_{Enhanced}$: the fraction of the computation time that take advantages of the enhancement.

- $Speedup_{Enhanced}$: the suze of the enhancement on the part it affects.

A more complete formula is:

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{Enhanced}) + \frac{Fraction_{Enhanced}}{Speedup_{Enhanced}}} \tag{2}$$

an example could be useful:

An enhancement makes one machine 10 times faster for 40% of the programs the machine runs. Which is the overall speedup?
Fraction = 0.4, Speedup = 10.

$$Speedup_{overall} = \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = 1.56 \tag{3}$$

Another important designing evaluation is measuring the time required to execute a program, the are severals approaches:

- Observing the real system: Not easy to be evaluated.
- Simulation: It could be really expensive.
- CPU Equation.

the latest solution use a provided formula to evaluate the CPU time:

$$CPU_{Time} = CLK_{Time} * \sum_{i=1}^{n} CPI_i * IC_i \tag{4}$$

where:

- CPI_i: Number of clock cycles required by instruction i (depends on hardware and instr set).
- IC_i: Number of times instruction i is executed in the program (depends instruction set and compiler).
- CLK_{Time}: Inverse of clock frequency (depends technology).

in the pipelined processor, CPI_i may vary for a given instruction, therefore the evaluation becomes much harder.

2 Instruction Set Principles

2.1 Introduction

The Instruction Set Architecture (ISA) is hoe the computer is seen by the programmer or the compiler. There are different kind of design and they can be selected by different characteristics. The CPUs are often classified according to the type of their internal storage:

- Stack: Is the simplest one it can't accessing memory during operations.
- Accumlator: Similar to 8051 can access the memory during operation.
- Registers
  - Register-memory: Similar to 8086, have a 16-bit register.
  - Register-Register (load-store)
  - Memory-memory (no real cases)

Nowadays all processors are General-Purpose Register (GPR) this because registers are faster than memoery and are easier for a compiler to use. The CPUs are also classifiable by the number of operant per ALU instruction (2 or 3) and number of memory operand per ALU.

2.2 Memory

The memory has a fundamental work in the CPU world, there are several different type of it and of course different type to how accessing it.
There are many different **memory addresses** that can be implemented:

- **Little Endian**: Byte with lower address at the least significant position. The addresses of the data is that of the LSB.
- **Big Endian**: Byte with lower address at the most significant position. The addess of the data is that of the MSB.
- **Aligned**: Allowing only aligned accesses to memory could is a limitation in terms of performance.
- **Misaligned**: This solution require hardware and performance overhead.

A memory position can be accessed in different ways:

- **Register**: (ADD R4, R3) when a value is in a register.
- **Immediate**: (ADD R4, #3) for constants.
- **Displacement**: (ADD R4, 100(R1)) accessing local variables.
- **Deferred or Indirect**: (ADD R4, (R1)) accessing using a pointer or a computed address.
- **Indexed**: (ADD R3, (R1 + R2)) useful in array addressing: R1=array base, R2=index.
- **Direct or Absolute**: (ADD R1, (1001)) useful for accessing static data.

- **Indirect**: (ADD R1, @(R3)) if R3 is the addr. of a pointer p, then the mode is like p.
- **Autoincrement or decrement**: (ADD R1, (R2)±) useful for stepping through array within a loop.

Is important to choosing the correct addressing mode, one can obtain some important consequences, from reduing the number of instruction to increasing the architecture.
There are a lot of instruction that can be done, looking at the 80x86 frequency we can see that most most used are *Load, Conditional Branch, Compare and Store* that are around the 70% of the total operation performed.
The instruction set encoding depends on which instruction compose the instruction set and which addressing modes are supported. When a high number of addressing modes is supported, and address specifier is used to specified the addressing mode. When the number of addressing modes is low, they can be encoded together with the opcode. The different encoding are:

- **Variable**: Any number of operands, operation with variable length, lower performance and minimum code size.
- **Fixed**: Fixed number of operands, need address specifier, fixed instruction length, maximum performance but larger code size.
- **Hybrid**: Multiple format specified by the opcode and it allows a trading-off between code size and performance.

Assembly-level programs are now produced by compilers only. The CPU designer and the compiler writer must interact and cooperate. A crucial part taken by the compiler is the optimization of the register using, it easier to solve it when the number of register is higher (>16).

3 Pipelining

3.1 Introduction

Pipelining is an a solution to execute multiple instructions at the same time overlapping it. The different units (also called *stage* or *segment*) are completing different part of different instruction in parallel. The figure 3 is an example of this techniques. The **throughput** of a pipelined processor is the number of instructions which exit the pipeline in the time unit. All the pipeline stages are synchronized (they proceed to executing the new task all together); the necessary time for executing one step is called machine cycle and, normally, corresponds to one clock cycle. Of course the duration of a machine cycle is determined by the slowest stage. CPI means clock cycles per instruction. The ideal pipeline have all stages perfectly balanced, in this way the $TRP_{pipelined} = TRP_{unpipelined} * n$ where n is the number of pipeline stages.

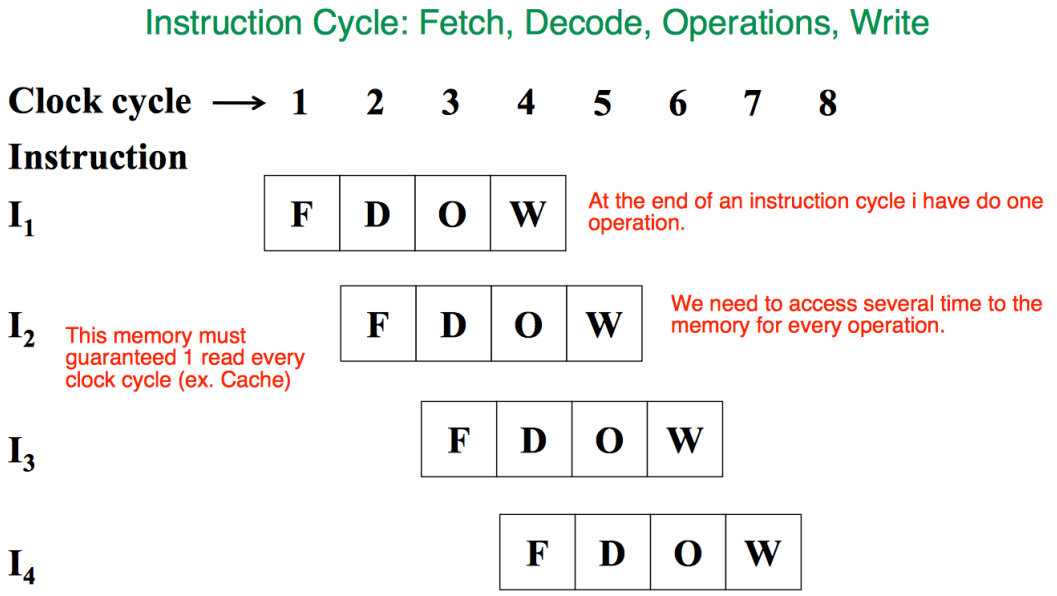

Figura 3: Pipeling example

3.2 Behaviour

The execution of each instruction may be composed of at most five clock cycles:

- **IF**: Fetch the instruction from the memory address of the PC and save it in the IR.
- **ID**: Decode the instruction just fetched.
- **EX**: Execute the instruction.
- **MEM**: Memory access and brach completion.
- **WB**: Write-back.

with the unpipelined processor all instruction require 5 CC, the only optimization to reduce the average CPI is completing the ALU instruction during the MEM cycle, hardware resources could be optimized avoiding duplications.
The pipelined version instead can be more powerful beacause a new instrcution is started at each clock cycle and different resources work on different instruction at the same time. There are several consideration to keep in mind during the development of this units; at every clock cycle, each resource can be used for one purpose only, this means that:

- Separate instruction and data memories must be used.
- The register file is used in two stages: for reading (second half of CC) in ID stage and for writing (first half od CC) in WB stage.

- The PC (*Program Counter*) must be changed in the IF stage, What about branches?

A little view of the system behaviour in figure 4.

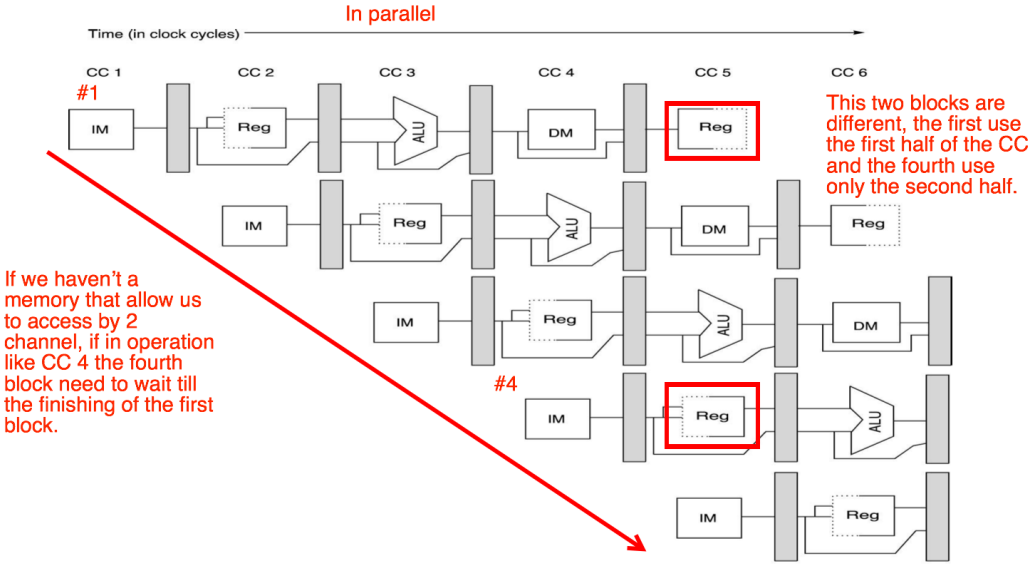

Figura 4: Evolution in time of Pipeling architecture

Performance The first puorpose of this architectural type is the performance increase without making single instruction faster. The single instruction are maded slower for the overhead introduced by the pipeline control. The limitations of a pipeline come from the necessity of balanced stages and pipeling overhead (pipeline register delay and clock skew).