ARM: ARM mimarisi, RISC mimarisi kategorisinde olup isminin uzun açılımı Acorn RISC Machine şeklindedir.



There were processors inside these big telephones which does only one job 🡪 telecommunication

Microprocessors are faster than microcontrollers.

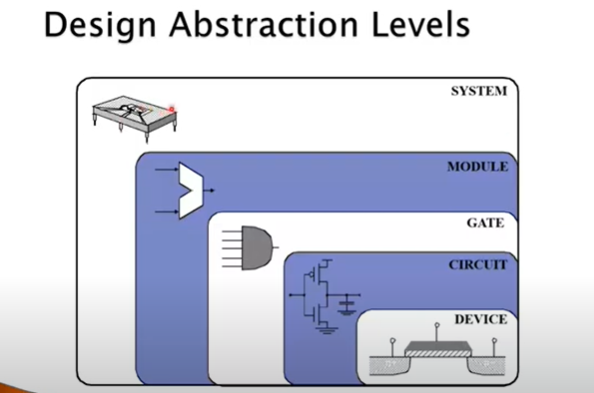
1 transistor speed is about 100 GHz.

A picture containing text, electronics

Description automatically generatedThere is OS (Handling i/o, managing memory and storage, scheduling tasks & sharing resources) and compiler under Systems software.

What do we need for a CPU:

1. Transfer data
2. Arithmetic operations
3. Comparisons
4. Jump operation
   1. Jump to else if condition is false
   2. Jump to function and jump back where (we should keep this place) it is called
5. Logical operations



System – module – gate layers are in this course.

A picture containing text

Description automatically generated**Components of a Computer**

Same components for all kinds of computer

I/O includes:

* UI devices 🡪 keyboard, mouse, etc.

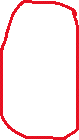
processor core

* Storage
* Network adapters

Any sequential circuit is made up of 2 components: control unit (made up of FSM) and datapath.



Control unit sends control signals to the datapath so that it controls the datapath.



Inside datapath, we have a path for any instruction. Datapath enables any instruction execution. Selection of paths is done by CU. CU selects which path inside datapath will be activated and which instruction will be executed.

CU orders the datapath.

ALU is inside the datapath.

Control unit is designed using Finite State Machine (not necessarily).

Combinational: hafıza yok

Sequential: hafıza var

Bilgisayar 🡪 Sequential

Level 1 cache is closest to CPU. CPU only communicates with this cache. Levels 2 and 3 are too slow for the CPU.

Computer is not embedded.

You need some smart circuit embedded inside smart system. Computer is already smart. Bilgisayarı başka sisteme gömersen embedded olur. Mouse is embedded.

We have microcontroller in mouse bc microprocessor needs huge energy.

Diagram

Description automatically generated

When CPU doesn’t find what is need in Level 1 cache, Level 1 cache go ask Level 2. Then Level 2 cache (if it doesn’t have the data) asks Level 3. If Level 3 also doesn’t have the data, then we go to main memory. This step is done through northbridge.

**HISTORY OF COMPUTERS**

AL-JAZARI 🡪 Earliest programmable analog computer in 1206

BABBAGE 🡪 Difference engine, the first mechanical computer (calculator), 1832

Z1 🡪 First computer before ENIAC

ENIAC 🡪 Built in WW2, first general purpose computer, used for computing artillery firing tables. All tasks of OS (load program to register, etc.) was handling by humans.

FIRST TRANSISTOR 🡪 Bell labs, All of 3 architectures got Nobel, 1948

UNIVAC I 🡪 First commercial computer, 1951

Moore’s Law 🡪 Gordon Moore noted that the number of transistors on a chip doubled every 18 to 24 months, 1965. This law is continued until 2010s.

We started to make transistors:

* Consume less power
* Consume less space
* Faster

Table

Description automatically generated

Uniprocessor Performance

After 2010s, technology improvement constrained by power, instruction level parallelism, memory latency

In CMOS IC technology:

* Power = Capacitive load x x Frequency
  + There is nothing inside digital circuits other than transistors.
  + Capacitive load is undesired. Smaller transistors have smaller capacitive load. But when we have smaller transistors, we put more transistors in our chip so we get more capacitive load. So we can’t decrease capacitive load.
  + Power consumption and heat is directly correlated.
* You must get rid of heat to avoid transistors to reduce speed.
* We can decrease the frequency to decrease the power but increase in frequency is our aim, so you can’t reduce freqeuency.
* You should decrease voltage but it reached to limit. We can’t decrease it under 0 bc then we can’t discriminate 0 and 1.

As a solution they said that, from now on, we will not use single core processors.

İlk başta programlar multicore’a göre tasarlanmadığından bir core kullanılıyor, bir süre sonra diğer core’a geçiliyordu ancak bu bir performans sağlamadı.

Power’ın yarısı devrenin çalışması için, diğer yarısı transistörler tarafından ısıya dönüşüyor. Soğutucu olmazsa Windows açılana kadar processor 100 dereceye ulaşır. Problem küçük alanın çok ısınması.

4 çekirdeğin 8 çekirdek olmasındansa 4 çekirdeğin her birinin daha hızlı olması daha iyi.

Power wall:

* can’t reduce voltage further
* can’t remove more heat

Parallelism is the solution. That’s why we have cores.

**MULTIPROCESSORS**

Multicore microprocessors

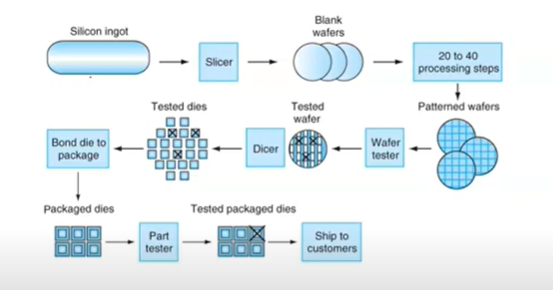
* More than one processor per chip

Requires explicitly parallel programming

* Compare with instruction level parallelism
  + Hardware executes multiple instructions at once
  + Hidden from the programmer
* Hard to do
  + Programming for performance
  + Load balancing
  + Optimizing communication and synchronization

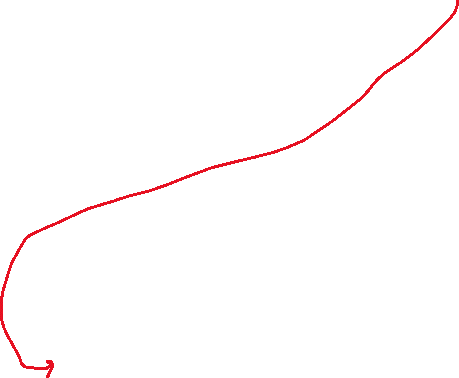
All programs are naturally sequeantially so this is why parallelism is hard to do

**MANUFACTORING**



A picture containing text

Description automatically generated



dies = yonga

bining = aynı işlemlerden geçen 2 son ürünün (çip) farklı (hızlarda) olması

There are core and cache in the chip.

Chart, bubble chart

Description automatically generated

**INTEGRATED CIRCUIT COST**

Cost per die =

Yield: amount of chips you have created that will be sold to the customer

Dies per wafer Wafer area / Die area

Yield =

If you sell 14 nm transistor your yield can be 70% and if 5nm which is newer technology your yield can be 10%. This means you can only sell 10% of the manufactured chips.

Nonlinear relation to are and defect rate

* Wafer cost and area are fixed
* Defect rate determined by manufacturing process
* Die area determined by architecture and circuit design

Table

Description automatically generated

Wafer cost depends on nanometer technology and masks (to construct transistors on silicon) you use.

**PERFORMANCE**

Algorithm determines number of operations executed

Programming languages, compiler, architecture determine number of machine instructions executed per operation

Processor and memory system determine how fast instructions are executed

I/O system including OS determines how fast I/O operatations are executed

**8 GREAT IDEAS**

1. Design for Moore’s Law
2. Use abstraction to simplify design
   1. Start with top level, divide your design at different parts, for each part get down 1 level perform your design at that level
3. Make the common case fast
4. Performance via parallelism
5. Performance via pipelining
6. Performance via prediction
7. Hierarchy of memories
   1. Small memory faster, Big memory slower
8. Dependability via redundancy

RESPONSE TIME: How long it takes to do a task

THROUGHPUT: Total work done per unit time – in one second (e.g., tasks/transactions/… per hour)

2 most famous performance metrics:

* speed
* throughput (# of instructions you can execute in specific amount of time)

You can increase the throughput without increasing the speed. Even you can increase the throughput after you decrease the speed. With parallelization. We cant increase the speed anymore so we try to increase the throughput.

If you can increase the speed, you increase the throughput and response time.

1 yerine 2 core olunca performans 2’ye katlanır diye bir şey yok. Taskleri ikiye ayırmak kolay değil.

Relative Performance

Performance =

X is n time faster than Y:



Example: time taken to run a program:

* 10s on A, 15s on B
* Execution time B / Execution time A = 15s / 10s = 1.5
* So A is 1.5 times faster than B

Measuring Execution Time

Elapsed time

* Total response time, including all aspects
  + Processing, I/O, OS overhead, idle time
* Determines system performance
* Difficult to compute theoratically
  + Easier to compute using benchmarks
  + For example, antutu benchmark executes some tasks and accordingly gives a score to your performance

CPU time

* Time spent processing a given job
  + Discounts I/O time, other jobs’ shares
* Comprises user CPU time and system CPU time
* Different programs are affected differently by CPU and system performance

Basic Measurement Metrics

Comparing machines

* Metrics
  + Execution time (CPU)
  + Throughput (CPU)
  + CPU time – same as execution time
  + MIPS – millions of instructions per second
    - not very good because we don’t know if instruction is complex or not
  + MFLOPS – millions of floating point operations per second
    - floating point instructions are executed in different units bc floating point arithmetic is different than integer arithmetic and floating point requires more time

Comparing machines using sets of programs

* Arithmetic mean, weighted arithmetic mean
* Benchmarks

Best metric for performance is execution time

CPU Clocking

Operation of digital hardware governed by a constant-rate clock

Diagram

Description automatically generated

Clock period (cycle): duration of clock cycle

* At each clock period, part of a task is finished, we don’t finish whole instruction
  + In a single cycle machine, at each clock cycle an instruction is executed
* e.g., 250ps = 0.25ns = 250 x 10-12 s

Clock frequency (rate): cycles per second

* e.g., 4.0GHz = 4000MHz = 4.0 x 109 Hz

Architecturelar aynıysa clock rate daha düşük olsa da response time, throughput daha yüksek olabilir.

How many cycles are required for a program?

Could assume number of cycles = number of instructions (Correct for single cycle processors) 🡪 But this is incorrect. Different instructions may require different amount of cycles on different machines. Even for 1 cycle, we might have more than 1 instructions executed in modern processors.

If you don’t use superscalar architecture (Hardware is executing multiple instructions at the same time. In order to do that, they have multiple ALUs. You have different copies of same hardware), # of cycles > # of instructions generally. We use pipeline but we use different parts of the hardware for executing more than 1 instructions. For instance we use ALU for 1 instruction, we use memory for another instruction, we use another part of the datapath for another instruction.

Chart, box and whisker chart

Description automatically generated

Different numbers of cycles for different instructions

Chart, box and whisker chart

Description automatically generated

Division takes more time than addition (takes more cycle)

Floating point operations take longer than integer ones

Accessing memory takes more time than accessing registers

A given program will require:

* some number of machine instructions
* some number of clock cycles
* some number of seconds

We have a vocabulary that relates these quantities:

* clock cycle time (seconds per cycle)
* clock rate (cycles per second)
* CPI (cycles per instruction)

# of clock cycles x length of the clock period in terms of seconds = execution time (# of seconds)

Execution time = # of instructions x CPI x CPU cycle time

* # of instructions x CPI = toplam program için ne kadar cycle gerektiği

Instruction sayısına bakarken loopların kaç kere çalışacağına da bakmak gerekir. Loop 10 kere dönüyorsa ve içinde 1 instruction varsa 10 instruction diye sayarız.

CPI can be floating point number:

* 1 instruction requires 2 cycles
* 2 instructions require 1 cycle
* CPI is 3/2 = 1.5

Computing CPU time 🡪 The time to execute a given program can be computed as:

* CPU time = CPU clock cycles x Clock cycle time

Since clock cycle time and clock rate are reciprocals:

* CPU time = CPU clock cycles / clock rate

The number of CPU clock cycles can be determined by

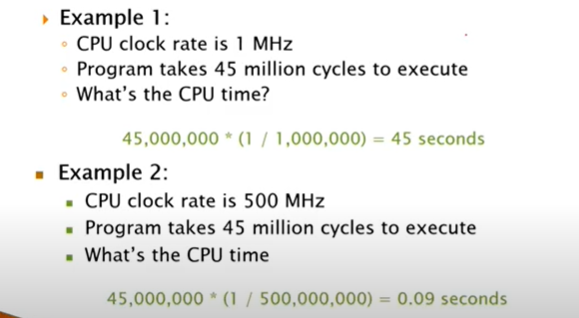
* CPU clock cycles = (instructions/program) x (clock cycles/instruction) = instruction count x CPI
  + program: çalıştırdığım program sayısı, genelde 1

CPU time = instruction count x CPI x clock cycle time = instruction count x CPI / clock rate

Graphical user interface, application, website

Description automatically generated

CPU TIME EXAMPLES



Table

Description automatically generated

EX1 = IC1 x PERIOD = 10 x PERIOD  
EX2 = IC2 x PERIOD = 9 x PERIOD

If they are executed on the same machine, PERIODs are same so second program is faster than first program 10/9 times.

Program 1 is more complex than program 2 bc it needs more CPI.

Ayrıca 1. programda C’den yani en kompleks işlemden (en kompleks çünkü CPI’ı en fazla) daha fazla var.

Computing CPI

The CPI is the average number of cycles per instruction.

If for each instruction type, we know its frequency and number of cycles need to execute it, we can compute the overall CPI as follows:

* CPI = (CPI x F)

For example:

Table

Description automatically generated

2.2 is actual CPI for my program.

You can always assume number of instructions if you don’t know number of instructions. At the end when you are computing CPI, the result will be divided by number of instructions and number of instructions will be cancelled out with nominator:

I 🡪 total number of instructions in my program

* 0.5 x I 🡪 total # of ALU instructions 🡪 0.5 x I x 1(# of cycles) 🡪 total # of cycles required by ALU instructions
* 0.2 x I 🡪 total # of Load instructions 🡪 0.2 x I x 5 🡪 total # of cycles required by Load instructions
* 0.1 x I 🡪 total # of Store instructions 🡪 0.1 x I x 3 🡪 total # of cycles required by Store instructions
* 0.2 x I 🡪 total # of Branch instructions 🡪 0.2 x I x 2 🡪 total # of cycles required by Branch instructions
* CYCLES : 0.5I + 1I + 0.3I + 0.4I = 2.2I 🡪 total cycles required by my program
* CPI = 2.2I/I = 2.2

WITHOUT KNOWING THE # OF INSTRUCTIONS, WE CAN COMPUTE THE CPI IF WE KNOW THE PERCENTAGES OF INSTRUCTIONS INSIDE OUR PROGRAM.

PERFORMANCE

Performance is determined by execution time

Do you think any of the variables is sufficient enough to determine computer performance?

* # of cycles to execute program?
* # of instructions in program?
* # of cycles per second?
* average # of cycles per instruction?
* average # of instructions per second?

It is not true to think that one of the variables is indicative of performance.

Best metric to represent the performance of processor is execution time.

Bu zamana kadar datada vs. yaptığımız O(n) li analizler instruction sayısıyla ilgiliydi. Number of cycles for your program and clock rate of the processor both depend on the processor architecture. We will see this side.

EXAMPLES

Text

Description automatically generated

Text

Description automatically generated

Can these 2 programs executed in 2 machines be the same program with same instructions (and same # of instructions) and even tough they are the same programs, can their CPIs be different? (tek çekirdek mimarisi)

* Aynı program, aynı instructionlardan oluşuyor, instruction countları aynı. Farklı makinelerde CPI ları farklı olabilir mi?
  + CPI = total cycle / total instruction -----> paydalar aynı
  + Instructionların çalışma süreleri, kaç cycle sürecekleri vs. hep architecture a bağlıdır.
  + Dolayısıyla iki program birebir aynı da olsa iki makine aynı instruction set architecture a da sahip olsa CPI lar farklı çıkabilir. Bunun sebebi tasarımdaki farklılıktır.

Execution\_M1 = 2.8 x I x 1/50MHz

# of cycles in my program for M1

= 2.8/1.6

Execution\_M2 = 3.2 x I x 1/100MHz

M2 is faster than M1 by 1.75 times.

Problems with Arithmetic Mean

Applications do not have the same probability of being run

For example, 2 machines timed on 2 benchmarks:

Table

Description automatically generated

Poor Performance Metrics

Marketing metrics for computer performance included MIPS and MFLOPS

MIPS: millions of instructions per second

* MIPS = instructon count / (execution time x 106)
* For example, a program that executes 3 million instructions in 2 seconds has a MIPS rating of 1.5
* Advantage: Easy to understand and measure
* Disadvantages: May not reflect actual performance, since simple instructions do better

MFLOPS: millions of floating point operations per second

MIPS EXAMPLE

2 different compilers are being tested for a 500 MHz machine with 3 different classes of instructions: Class A, Class B, Class C, which require 1, 2, and 3 cycles (respectively). Both compilers are used to produce code for a large piece of software.

The 1st compiler’s code uses:

* 5 billions Class A instructions,
* 1 billion Class B instructions,
* 1 billion Class C instructions.

The 2nd compiler’s code uses:

* 10 billions Class A instructions,
* 1 billion Class B instructions,
* 1 billion Class C instructions.

Which sequence will be faster according to execution time?

Which sequence will be faster according to MIPS?

* 1st compiler uses
  + 5x1 billion cycle A instructions
  + 1x2 billion cycle B instructions
  + 1x3 billion cycle C instructions
  + Total of 10 billion cycles, each cycle is 1/500x106 = (1000 x 10-9) / 500 = 2 ns
    - So execution time is 10 x 109 x 2 x 10-9 = 20 seconds
    - In 20 seconds, I can execute 10 billion cycles
    - We have 7 billion instructions and 20 seconds so:
      * 7x109 / 20 = 0.35 x 109 instructions executed in 1 second
      * 350 x 106 instructions in 1 second so MIPS rating is 350
* 2nd compiler uses
  + 10x1 billion cycle A instructions
  + 1x2 billion cycle B instructions
  + 1x3 billion cycle C instructions
  + Total of 15 billion cycles, each cycle is 2 ns
    - So execution time is 15 x 109 x 2 x 10-9 = 30 seconds
    - In 30 seconds, I can execute 15 billion cycles
    - We have 12 billion instructions and 30 seconds so:
      * 12x109 / 30 = 0.4 x 109 instructions executed in 1 second
      * 400 x 106 instructions in 1 second so MIPS rating is 400
* We can say 1st compiler is 1.5 times faster than 2nd compiler because machines are same and it requires less cycles.
* According to MIPS rating, 2nd compiler is faster.

2 misleading points about MIPS

* For the same processor, we can get different MIPS ratings
* A program with better execution time can result in a bad MIPS rating

Hesaplaması kolay diye MIPS kullanılıyor.

Aslında clock rate de tek başına misleading. Clock rate düşük ama her instruction 1 cycle da bitiyor olabilir. Clock rate çok yüksek ama her instruction 10 cycledan fazla sürüyor olabilir.

Table

Description automatically generated

Performance Summary

The two main measure of performance are:

* execution time: time to do the task
* throughput: number of tasks completed per unit time

Performance and execution time are reciprocals, increasing performance, decreases execution time

The time to execute a given program can be computed as:

* CPU time = instruction count x CPI x clock cycle time
* CPU time = instruction count x CPI / clock rate

These factors are affected by compiler technology, the instruction set architecture, the machine organization, and the underlying technology

When trying to improve performance, look at what occurs frequently 🡪 make the common case fast

1 instruction cannot be executed in floating point cycles, but CPI can be floating point number.

EXAMPLE

Assume that a program runs in 100 seconds on a machine, with multiply operations responsible for 80 seconds. How much do I have to improve the speed of multiplication if I want my program to run 2 times faster?

* I want my program to run in 50 seconds. I will only improve the multiplication part.
* 80 seconds 🡪 30 seconds
  + 80/30 = 2.67 times improvement is needed on speed of multiplication
* Eğer çarpma 20 saniye, diğerleri 80 saniye alsaydı, sadece çarpmayı geliştirerek performansı 2 katına çıkartamazdık.

Graphical user interface, application

Description automatically generated

Estimating Performance Improvements

Assume a processor currently requires 10 seconds to execute a program and processor performance improves by 50% per year (1.5x)

By what factor does processor performance improve in 5 years?

(1+0.5)5 = 7.59

How long will it take a processor to execute the program after 5 years?

ExTimenew = 10/7.59 = 1.32 seconds

SPEC CPU Benchmark

Programs used to measure performance

* Supposedly typical of actual workload

Standard Performance Evaluation Corp (SPEC)

* Develops benchmarks for CPU, I/O, Web, Compiler, AI, …

SPEC benchmarks are repeated by 5-10 years period bc new machines arrived and new compilers so they need to be repeated

SPEC CPU2006

* Elapsed time to execute a selection of programs
  + Negligible I/O, so focuses on CPU performance
* Normalize relative to reference machine
* Summarize as geometric mean of performance ratios
  + CINT2006 (integer) and CFP2006 (floating-point)
    - C means programs are written in C

Table

Description automatically generated

SPECratio is hwo much faster is your machine according to reference machine (older technology).

Geometric mean:

* 🡪 n numbers in root

They use geometric mean because (x and y programs have same importance):

|  |  |  |
| --- | --- | --- |
|  | PROGRAM X | PROGRAM Y |
| MACHINE A | 12 | 2 |
| MACHINE B | 6 | 4 |

average of machine A : 14/2 = 7  
average of machine B : 10/2 = 5

Actually both machines performance should be same bc in program x machine A has twice worse but in program y machine A is twice better.

Geometric mean of machine A :   
Geometric mean of machine B :

So their performance is same.

Summary of Performance Evaluation

Response time and throughput

CPU time = # instr x CPI x cycle time

MIPS and MFLOPS poor performance metrics

SPEC benchmarks to measure CPU performance

Concluding Remarks

Cost/performance is improving

* Due to underlying technology development

Hierarchical layers of abstraction

* In both hardware and software

Instruction set architecture

* The hardware/software interface

Execution time: the best performance measure

Power is a limiting factor

* Use parallelism to improve performance