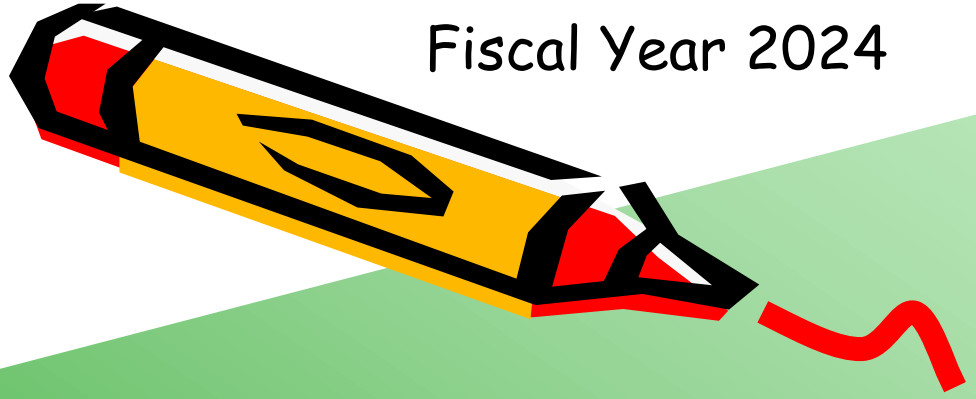


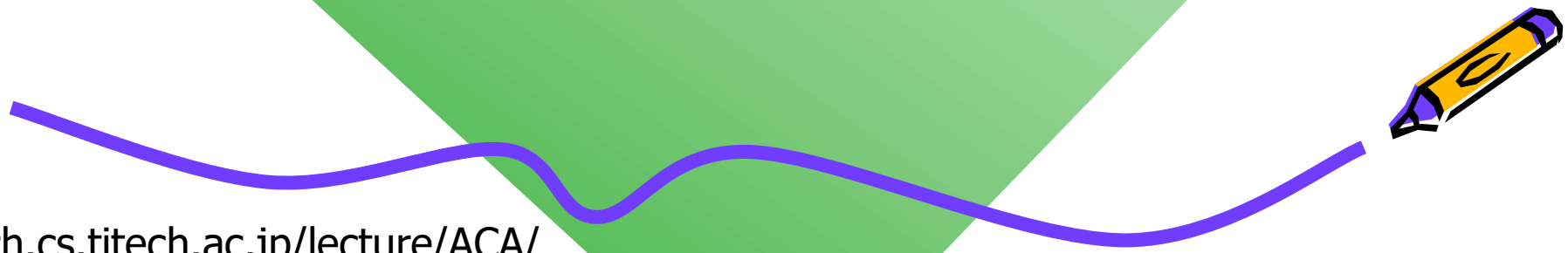
Fiscal Year 2024

Ver. 2024-12-09a

Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

1. Design and Analysis of Computer Systems



www.arch.cs.titech.ac.jp/lecture/ACA/
Room No. W8E-308, Lecture (Face-to-face)
Mon 13:30-15:10, Thr 13:30-15:10

Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Syllabus (1/3)

Course description and aims

This course aims to provide students with cutting-edge technologies and future trends of computer architecture with focusing on a microprocessor which plays an important role in the downsizing, personalization, and improvement of performance and power consumption of computer systems such as PCs, personal mobile devices, and embedded systems.
In this course, first, along with important concepts of computer architecture, students will learn from instruction set architectures to mechanisms for extracting instruction level parallelism used in out-of-order superscalar processors. After that, students will learn mechanisms for exploiting thread level parallelism adopted in multi-processors and multi-core processors.

Student learning outcomes

By taking this course, students will learn:
(1) Basic principles for building today's high-performance computer systems
(2) Mechanisms for extracting instruction level parallelism used in high-performance microprocessors
(3) Methods for exploiting thread level parallelism adopted in multi-processors and multi-core processors
(4) New inter-relationship between software and hardware

Keywords

Computer Architecture, Processor, Embedded System, multi-processor, multi-core processor

Competencies that will be developed

✓ Specialist skills	Intercultural skills	Communication skills	Critical thinking skills	Practical and/or problem-solving skills

Class flow

Before coming to class, students should read the course schedule and check what topics will be covered. Required learning should be completed outside of the classroom for preparation and review purposes.

Syllabus (2/3)

Textbook(s)

John L. Hennessy, David A. Patterson. Computer Architecture A Quantitative Approach, Fifth Edition. Morgan Kaufmann Publishers Inc., 2012

Reference books, course materials, etc.

William James Dally, Brian Patrick Towles. Principles and Practices of Interconnection Networks. Morgan Kaufman Publishers Inc., 2004.

Assessment criteria and methods

Students will be assessed on their understanding of instruction level parallelism, multi-processor, and thread level parallelism. Students' course scores are based on the mid-term report and assignments (40%), and the final report (60%).

Related courses

CSC.T363 : Computer Architecture
CSC.T341 : Computer Logic Design

Prerequisites (i.e., required knowledge, skills, courses, etc.)

No prerequisites are necessary, but enrollment in the related courses is desirable.

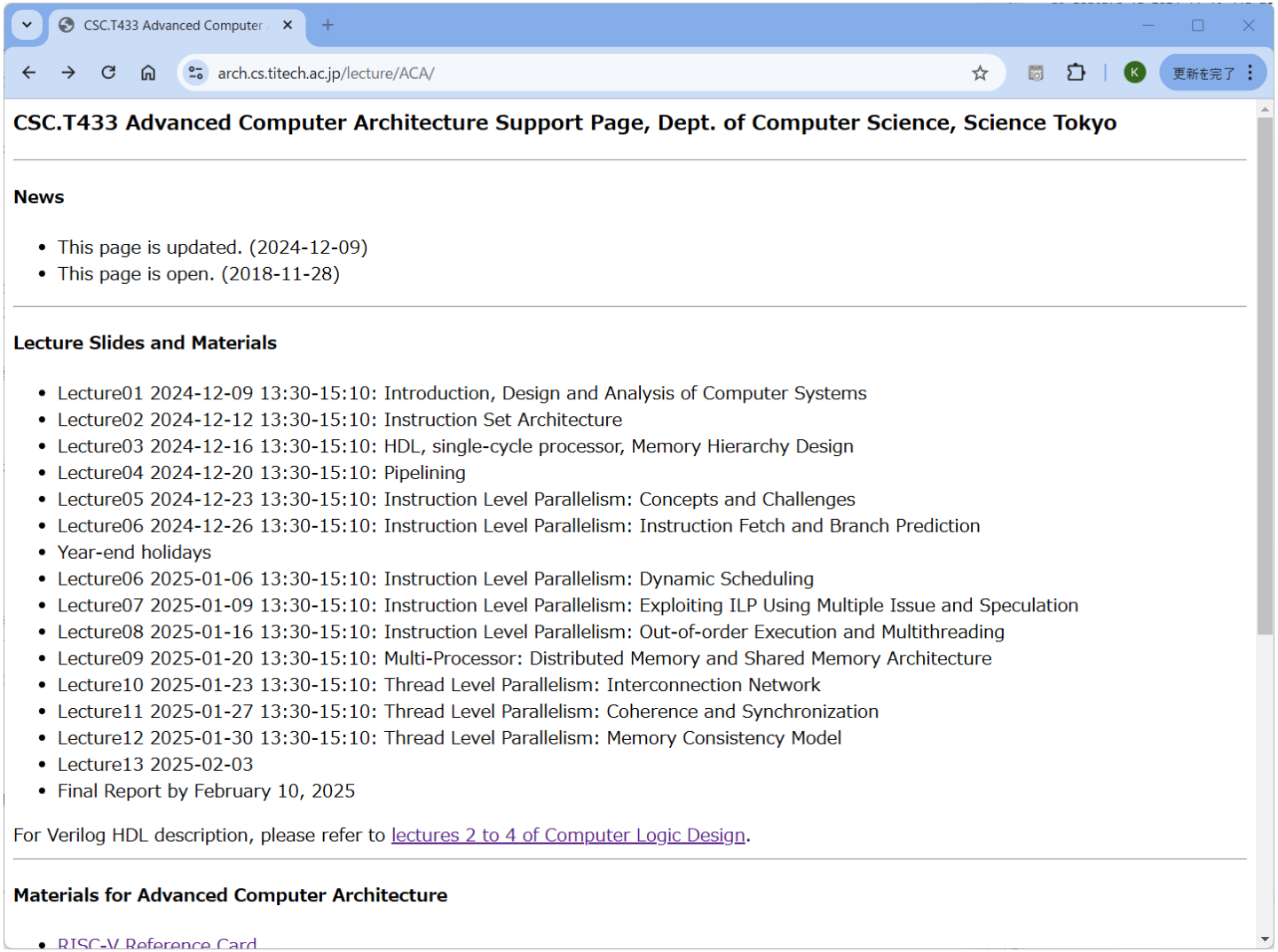
Contact information (e-mail and phone) **Notice : Please replace from "[at]" to "@"(half-width character).**

Kise Kenji: kise[at]c.titech.ac.jp

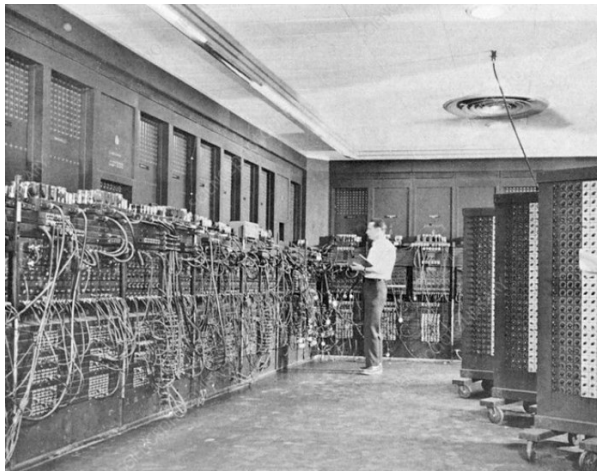
Office hours

Contact by e-mail in advance to schedule an appointment.

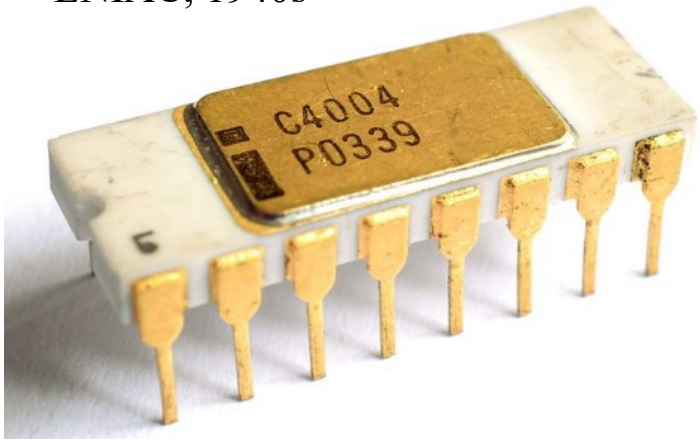
Support page https://www.arch.cs.titech.ac.jp/lecture/ACA/

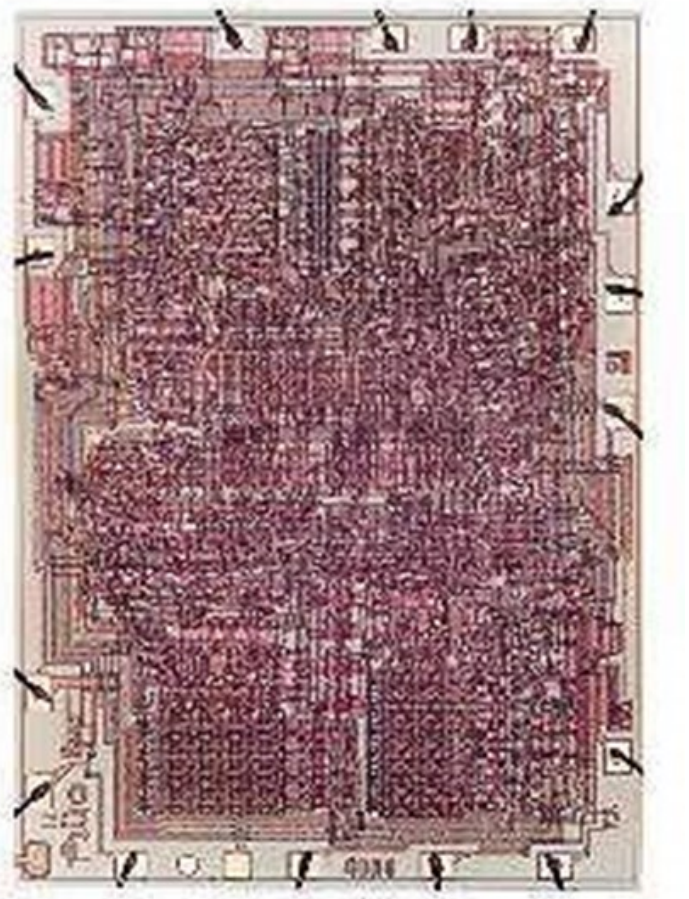


The birth of microprocessors in 1971



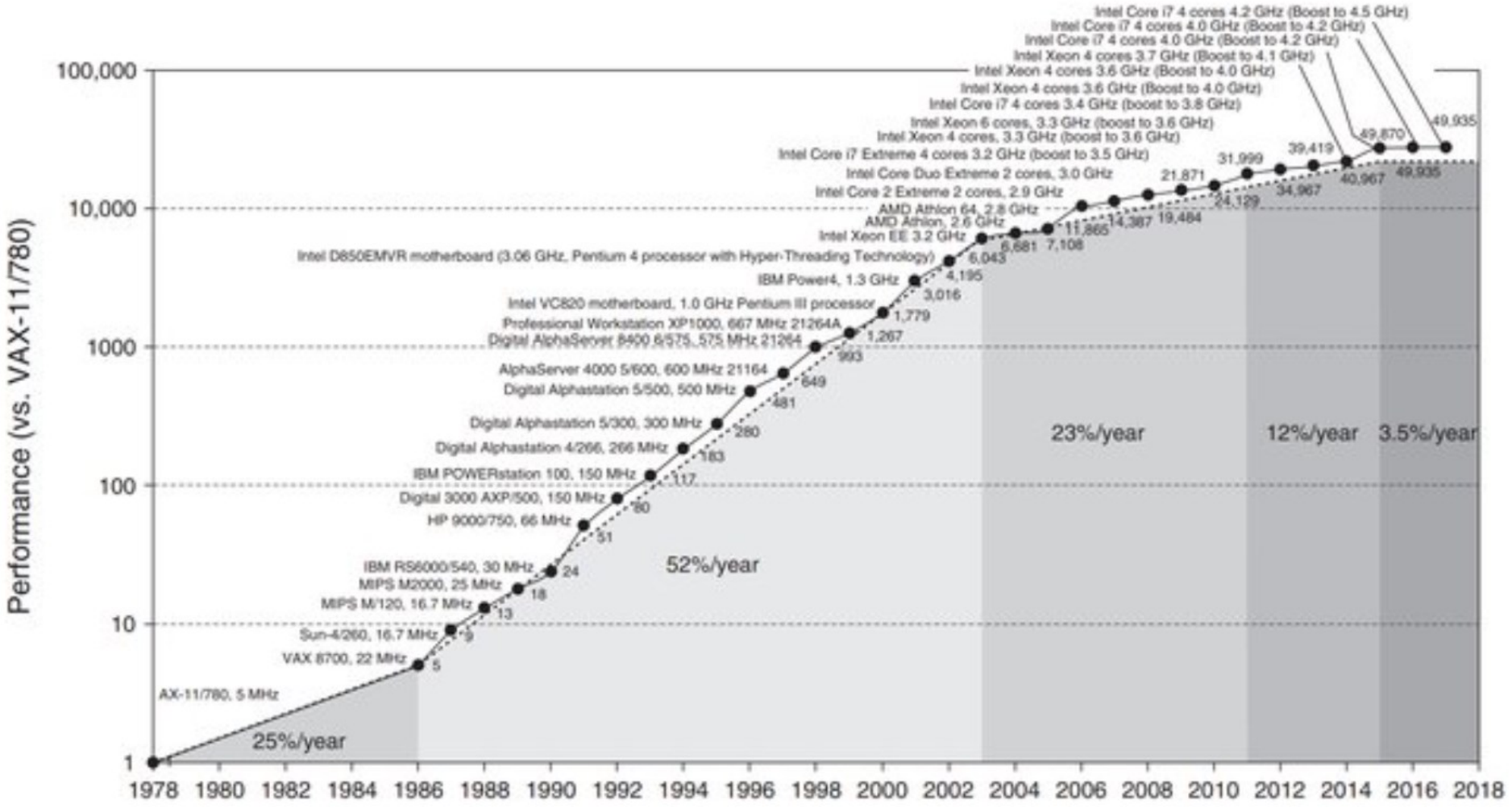
ENIAC, 1940s





Name	Year	# of transistors
Intel 4004	1971	2,250

Growth in processor performance



Program in C and RISC-V assembly code

```c
int main(){
    int i, sum=1;
    for(i=1; i<10; i++) sum = sum + i;
    return sum;
}
```

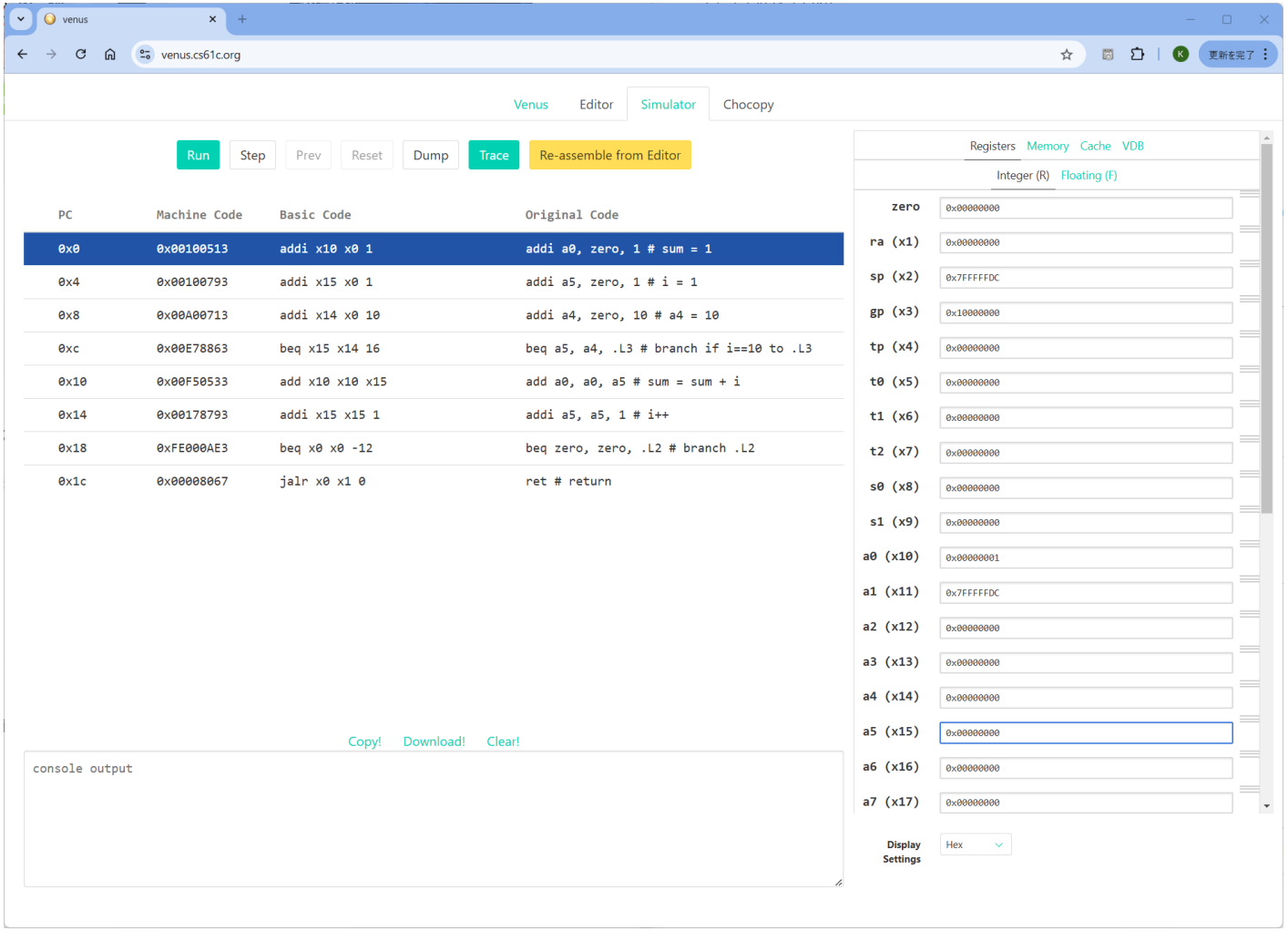
main1.c

main1.s

```
main:
    addi    a0, zero, 1      # sum = 1
    addi    a5, zero, 1      # i = 1
    addi    a4, zero, 10     # a4 = 10
.L2:
    beq     a5, a4, .L3      # branch if i==10 to .L3
    add     a0, a0, a5       # sum = sum + i
    addi    a5, a5, 1        # i++
    beq     zero, zero, .L2  # branch .L2
.L3:
    ret                      # return
```

Venus RISC-V editor and simulator

- https://venus.cs61c.org/



Defining performance and speedup

- Normally interested in reducing
  - Execution time (response time) – the time between the start and the completion of a program
  - Performance is the inverse of execution time.

$$\text{performance}_X = 1 \,/\, \text{execution\_time}_X$$

  - Thus, to maximize performance, need to minimize execution time

If X is n times faster than Y, then

$$\frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution\_time}_Y}{\text{execution\_time}_X} = n$$

Performance Factors

- Want to distinguish elapsed time and the time spent on our task
- CPU execution time (CPU time) : time the CPU spends working on a task
  - Does not include time waiting for I/O or running other programs

$$\text{CPU execution time for a program} = \text{\# CPU clock cycles for a program} \times \text{clock cycle time}$$

or

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock rate}}$$

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{F}$$

Clock rate or Frequency F is the inverse of clock cycle time

Performance Factors

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{F}$$

Performance is the inverse of CPU execution time

Performance for a program executing N instructions = F x 1 / # CPU clock cycles for a program

Multiply both sides if the equation by N

Performance_x_N = F x N / # CPU clock cycles for a program

- Performance = F x IPC
  - F : frequency (clock rate)
  - IPC : executed instructions per cycle (N / clock cycles)

- The performance can be improved by increasing either F or IPC

The past, present, and future of the world's most important device

EDITOR'S NOTE

BY HARRY GOLDSTEIN

The Device That Changed Everything

Transistors are civilization's invisible infrastructure

This replica of the original point-contact transistor is on display outside *IEEE Spectrum*'s conference rooms.

I was roaming around the *IEEE Spectrum* office a couple of months ago, looking at the display cases the IEEE History Center has installed in the corridor that runs along the conference rooms at 3 Park. They feature photos of illustrious engineers, plaques for IEEE milestones, and a handful of vintage electronics and memorabilia including an original Sony Walkman, an Edison Mazda lightbulb, and an RCA Radiotron vacuum tube. And, to my utter surprise and delight, a replica of the first point-contact transistor invented by John Bardeen, Walter Brattain, and William Shockley 75 years ago this month.

I dashed over to our photography director, Randi Klett, and startled her with my excitement, which, when she saw my discovery, she understood: We needed a picture of that replica, which she expertly shot and now accompanies this column.

What amazed me most besides the fact that the very thing this issue is devoted to was here with us? I'd passed by it countless times and never noticed it, even though it is tens of billions times the size of one of today's transistors. In fact, each of us is surrounded by billions, if not trillions of transistors, none of which are visible to the naked eye. It is a testament to imagination and ingenuity of three generations of electronics engineers who took the (by today's standards) mammoth point-contact transistor and shrunk it down to the point where transistors are so ubiquitous that civilization as we know it would not exist without them.

The best explanation of the point-contact transistor is in Bardeen's 1956 Nobel Prize lecture, but even that left out important details.

Of course, this wouldn't be a *Spectrum* special issue if we didn't tell you how the original point-contact transistor worked, something that even the inventors seemed a little fuzzy on. According to our editorial director for content development, Glenn Zorpette, the best explanation of the point-contact transistor is in Bardeen's 1956 Nobel Prize lecture, but even that left out important details, which Zorpette explores in classic *Spectrum* style in "The First Transistor and How It Worked," on page 24.

And while we're celebrating this historic accomplishment, Senior Editor Samuel K. Moore, who covers semiconductors for *Spectrum* and curated this special issue, looks at what the transistor might be like when it turns 100. For "The Transistor of 2047" [p. 38], Moore talked to the leading lights of semiconductor engineering, many of them IEEE Fellows, to get a glimpse of a future where transistors are stacked on top of each other and are made of increasingly exotic 2D materials, even as the OG of transistor materials, germanium, is poised for a comeback.

When I was talking to Moore a few weeks ago about this issue, he mentioned that he's attending his favorite conference just as this issue comes out, the 68th edition of IEEE's International Electron Devices Meeting, in San Francisco. The mind-bending advances that emerge from that conference always get him excited about the engineering feats occurring in today's labs and on tomorrow's production lines. This year he's most excited about new devices that combine computing capability with memory to speed machine learning. Who knows, maybe the transistor of 2047 will make its debut there, too. ■

PORTRAIT BY SERGIO ALBIAC; RANDI KLETT

IEEE Spectrum, December 2022

Moore's Law

- Moore's law is the observation that the number of transistors in a dense integrated circuit doubles about every two years. The observation is named after Gordon Moore, the co-founder of Fairchild Semiconductor and Intel, whose 1965 paper described a doubling every year in the number of components per integrated circuit, and projected this rate of growth would continue for at least another decade. In 1975, looking forward to the next decade, he revised the forecast to doubling every two years. The period is often quoted as 18 months because of a prediction by Intel executive David House (being a combination of the effect of more transistors and the transistors being faster).

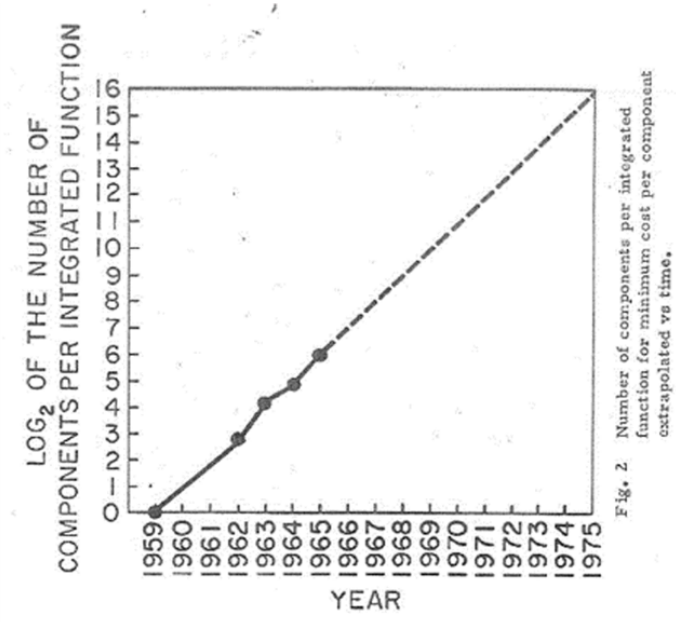


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

WIKIPEDIA

Moore's Law

VISUALIZING PROGRESS

If transistors were people

If the transistors in a microprocessor were represented by people, the following timeline gives an idea of the pace of Moore's Law.

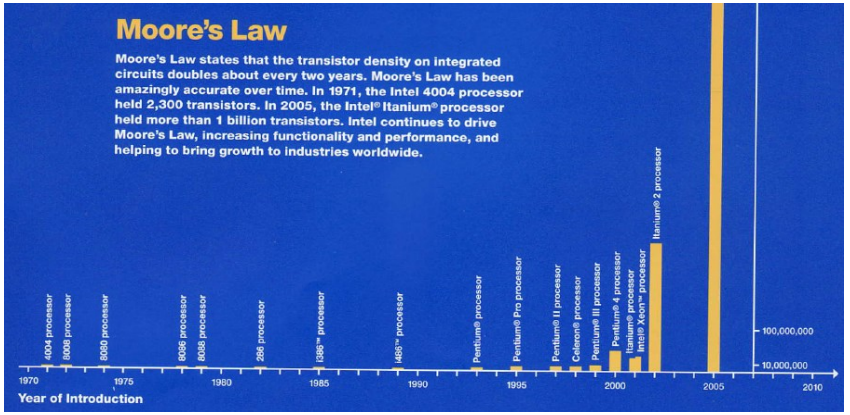
2,300
Average music hall capacity

134,000
Large stadium capacity

32 Million
Population of Tokyo

1.3 Billion
Population of China

Now imagine that those 1.3 billion people could fit onstage in the original music hall. That's the scale of Moore's Law.



Moore's Law

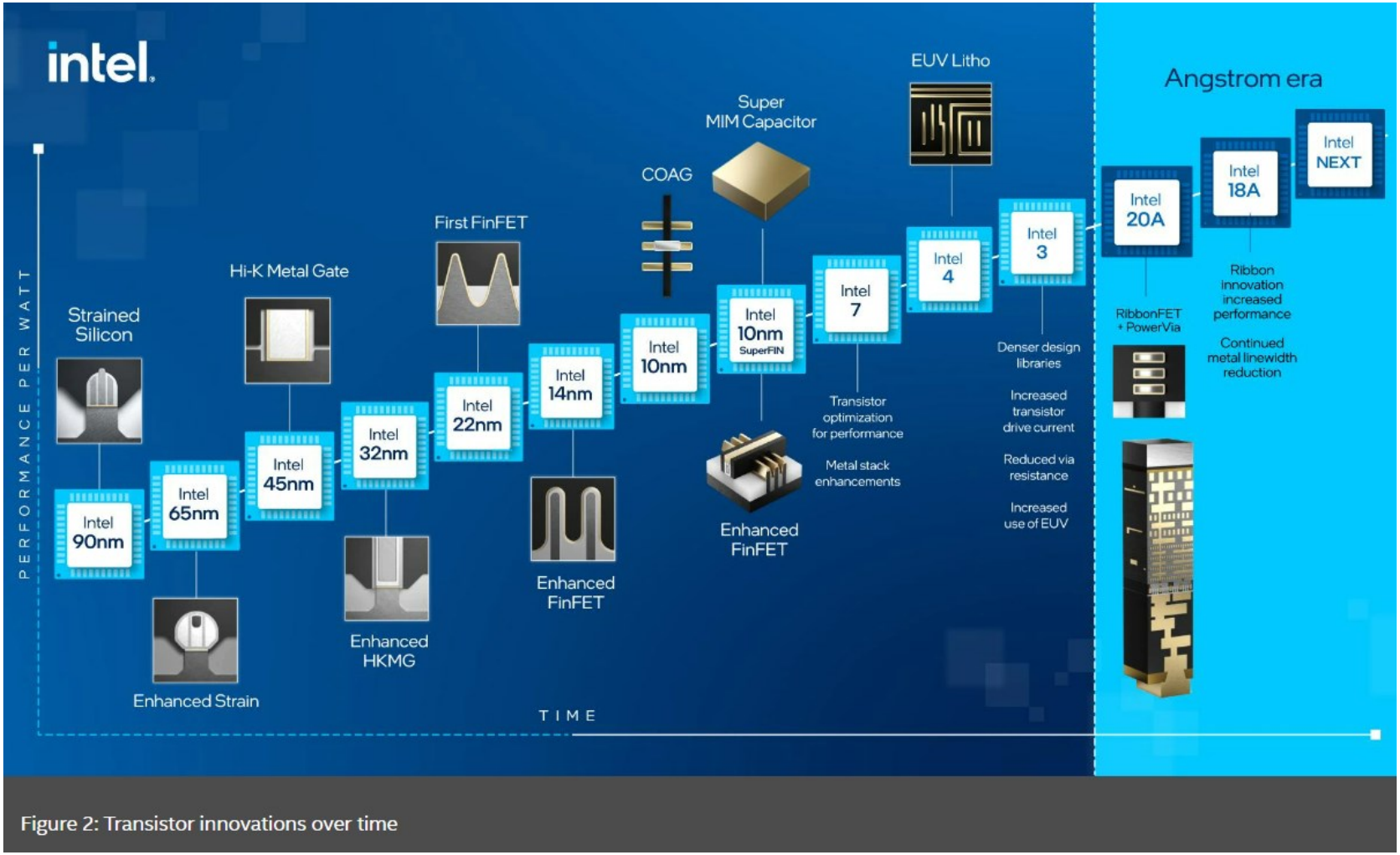
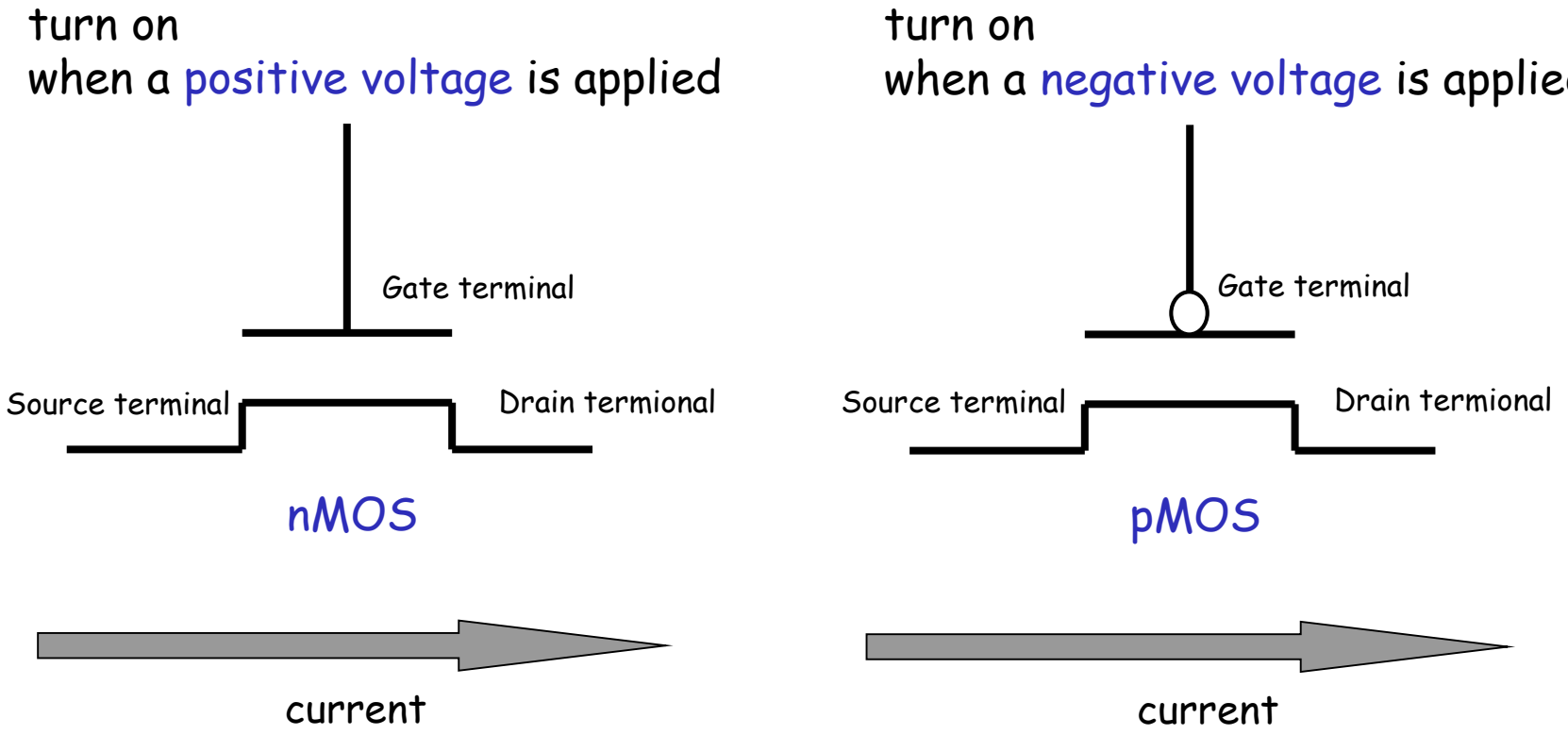


Figure 2: Transistor innovations over time

https://www.intel.com/content/www/us/en/newsroom/opinion/moore-law-now-and-in-the-future.html

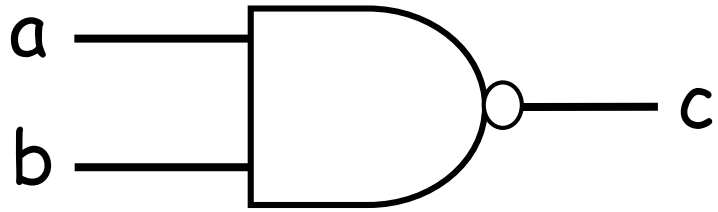
Transistor

- In an nMOS transistor, when a positive voltage is applied to the gate terminal relative to the source terminal, it creates an electric field that attracts electrons towards the gate. This forms a conductive channel between the source and drain terminals allowing current to flow through.



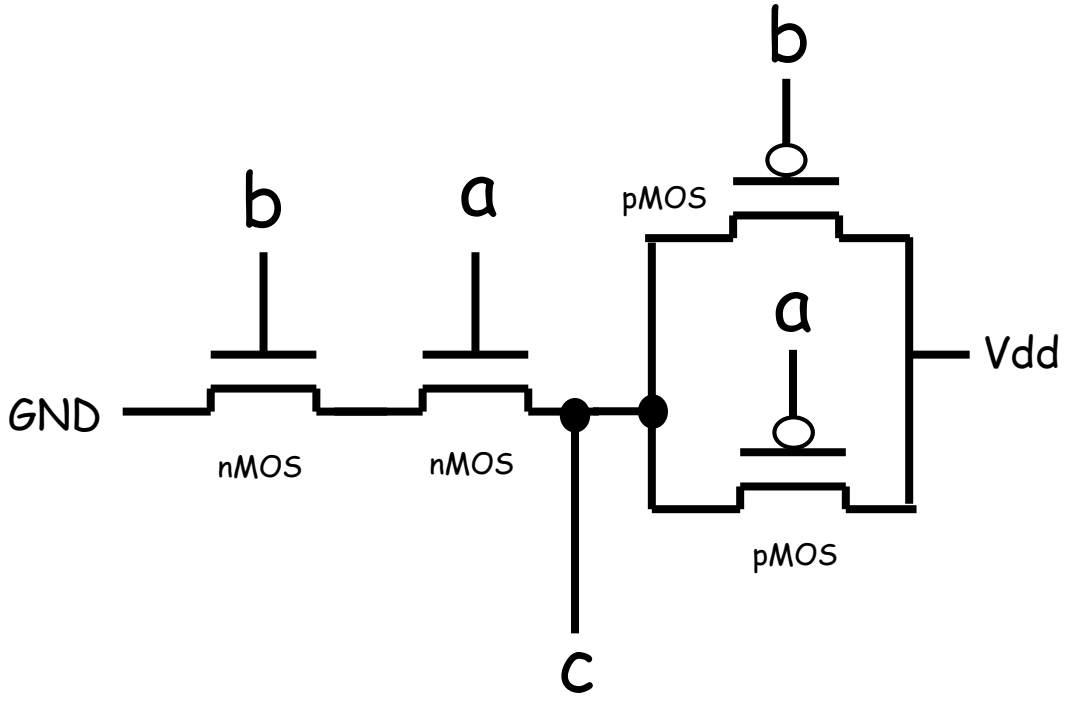
Transistor and Gate

NAND gate



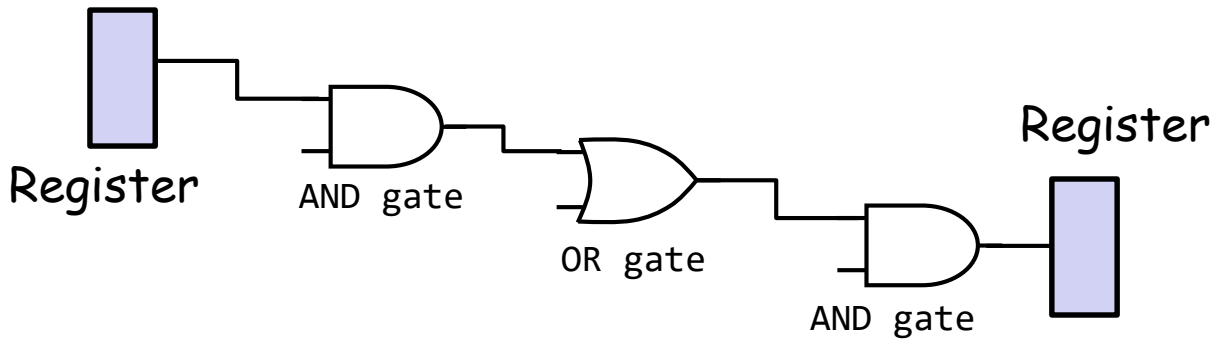
Truth table of NAND gate

a	b	c
0	0	1
1	0	1
0	1	1
1	1	0



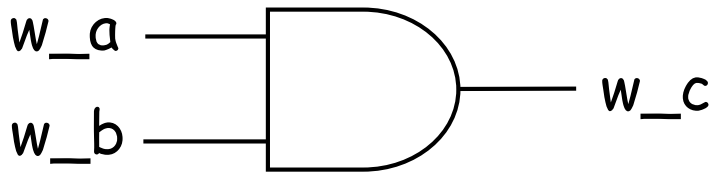
Clock rate F is mainly determined by

- Switching speed of gates (transistors)
- The number of levels of gates
  - The maximum number of gates cascaded in series in any combinational logics.
  - In this example, the number of levels of gates is 3.
- Wiring delay and fanout



Sample circuit 1 and Verilog HDL code 1

- AND gate



Truth table of AND gate

w_a	w_b	w_c
0	0	0
0	1	0
1	0	0
1	1	1

```
module addgate (
    input  wire w_a,
    input  wire w_b,
    output wire w_c
);
    assign w_c = w_a & w_b;
endmodule
```

Sample circuit 1 and Verilog HDL code 2

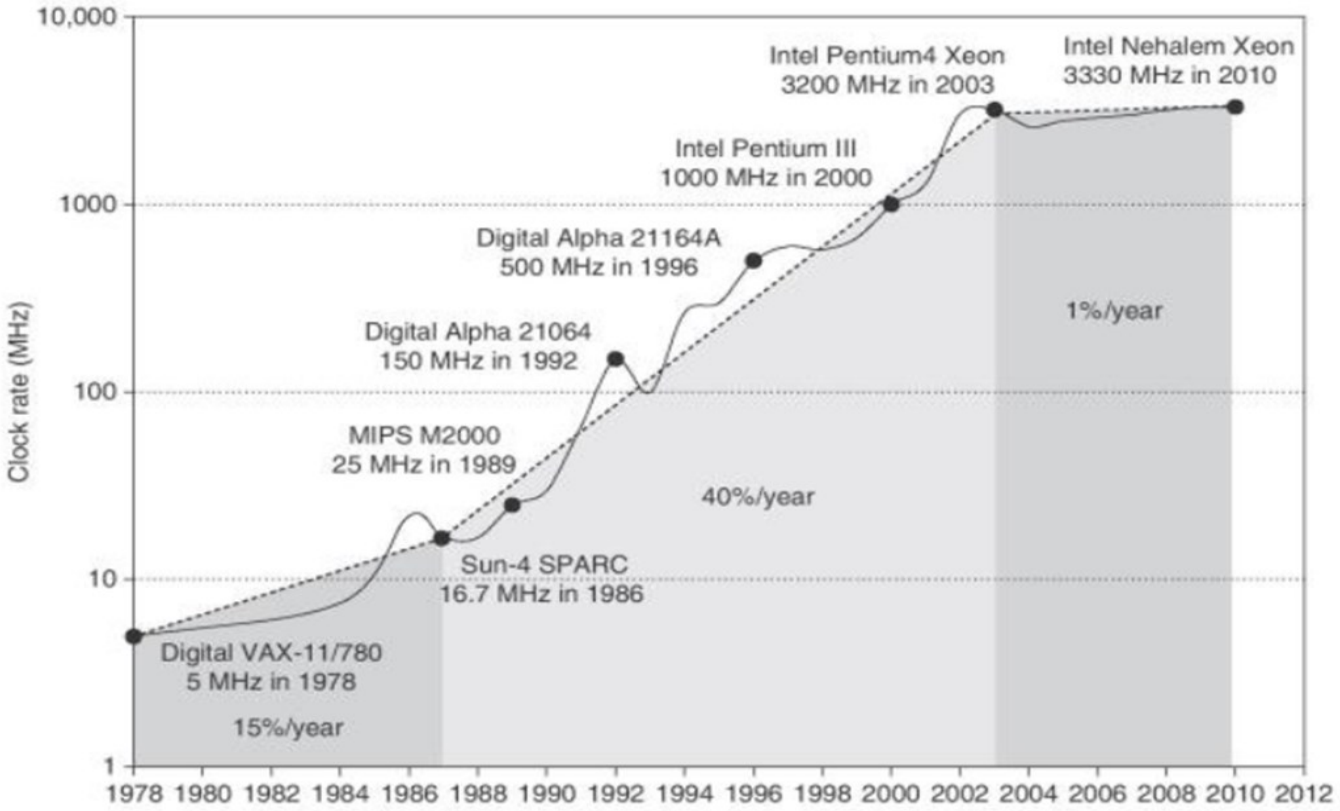
```
module top();
    wire w_c;
    reg r_a = 0;
    reg r_b = 0;
    initial #10 r_a = 1;
    initial #20 r_b = 1;
    always #1 $display("%d: %d %d %d", $time, r_a, r_b, w_c);
    initial #30 $finish();
    addgate m1(r_a, r_b, w_c);
endmodule

module addgate (
    input  wire w_a,
    input  wire w_b,
    output wire w_c
);
    assign #5 w_c =  w_a & w_b;
endmodule
```

```
 1: 0 0 x
 2: 0 0 x
 3: 0 0 x
 4: 0 0 x
 5: 0 0 0
 6: 0 0 0
 7: 0 0 0
 8: 0 0 0
 9: 0 0 0
10: 1 0 0
11: 1 0 0
12: 1 0 0
13: 1 0 0
14: 1 0 0
15: 1 0 0
16: 1 0 0
17: 1 0 0
18: 1 0 0
19: 1 0 0
20: 1 1 0
21: 1 1 0
22: 1 1 0
23: 1 1 0
24: 1 1 0
25: 1 1 1
26: 1 1 1
27: 1 1 1
28: 1 1 1
29: 1 1 1
30: 1 1 1
```

Growth in clock rate F of microprocessors

Intel 4004 clocked at 740KHz in 1971



From CAQA 5th edition

13th Generation Intel® Core™ i9 Processors

Products formerly Raptor Lake

Desktop

i9-13900K

Launched

Q4'22

Intel 7

$589.00 - $599.00

CPU Specifications

Total Cores	24
# of Performance-cores	8
# of Efficient-cores	16
Total Threads	32
Max Turbo Frequency	5.80 GHz
Intel® Thermal Velocity Boost Frequency	5.80 GHz
Intel® Turbo Boost Max Technology 3.0 Frequency ‡	5.70 GHz
Performance-core Max Turbo Frequency	5.40 GHz
Efficient-core Max Turbo Frequency	4.30 GHz
Performance-core Base Frequency	3.00 GHz
Efficient-core Base Frequency	2.20 GHz

Syllabus (3/3)

Course schedule/Required learning		
	Course schedule	**Required learning**
Class 1	Design and Analysis of Computer Systems	Understand the basic of design and analysis of computer systems.
Class 2	Instruction Set Architecture	Understand the examples of instruction set architectures
Class 3	Memory Hierarchy Design	Understand the organization of memory hierarchy designs
Class 4	Pipelining	Understand the idea and organization of pipelining
Class 5	Instruction Level Parallelism: Concepts and Challenges	Understand the idea and requirements for exploiting instruction level parallelism
Class 6	Instruction Level Parallelism: Instruction Fetch and Branch Prediction	Understand the organization of instruction fetch and branch predictions to exploit instruction level parallelism
Class 7	Instruction Level Parallelism: Advanced Techniques for Branch Prediction	Understand the advanced techniques for branch prediction to exploit instruction level parallelism
Class 8	Instruction Level Parallelism: Dynamic Scheduling	Understand the dynamic scheduling to exploit instruction level parallelism
Class 9	Instruction Level Parallelism: Exploiting ILP Using Multiple Issue and Speculation	Understand the multiple issue mechanism and speculation to exploit instruction level parallelism
Class 10	Instruction Level Parallelism: Out-of-order Execution and Multithreading	Understand the out-of-order execution and multithreading to exploit instruction level parallelism

From multi-core era to many-core era

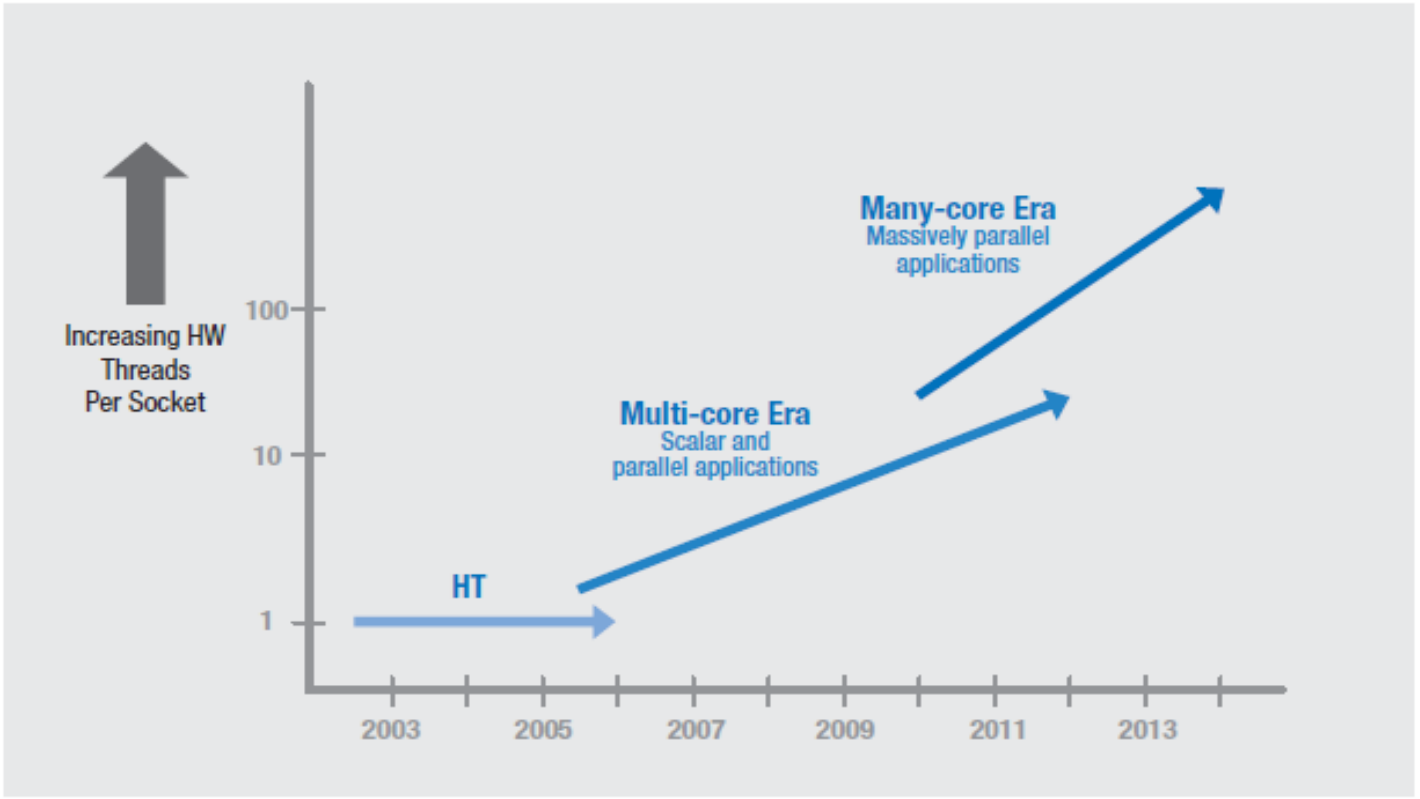


Figure 1: Current and expected eras of Intel® processor architectures

Platform 2015: Intel® Processor and Platform Evolution for the Next Decade, 2005

Pollack's Rule

- Pollack's Rule states that microprocessor performance increase due to microarchitecture advances is roughly proportional to the square root of the increase in complexity. Complexity in this context means processor logic, i.e. its area.

WIKIPEDIA

From multi-core era to many-core era

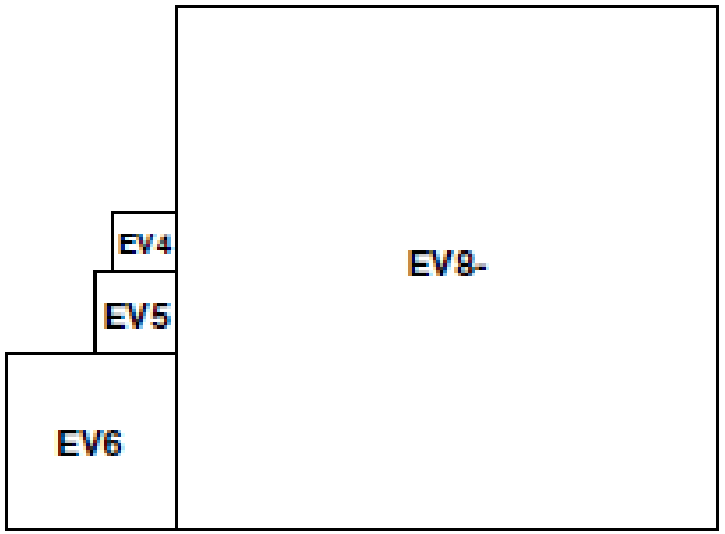
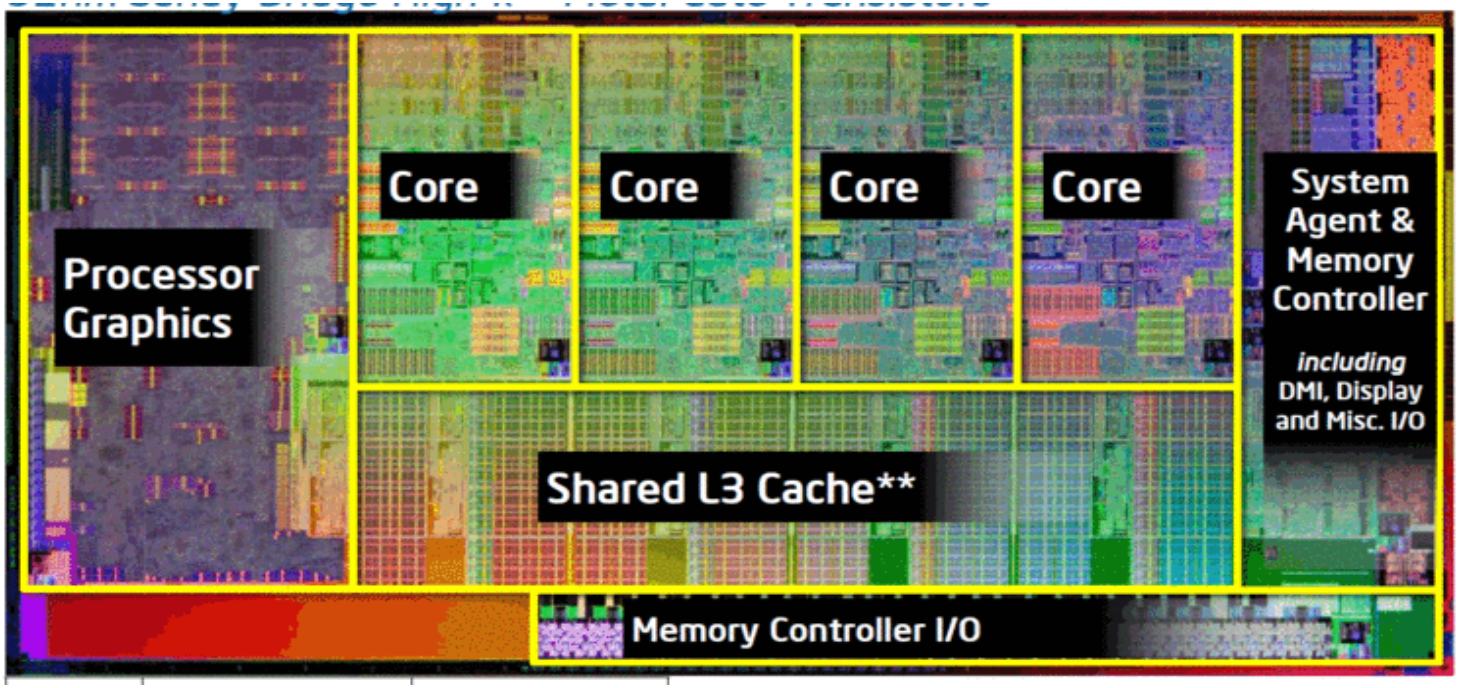


Figure 1. Relative sizes of the cores used in the study

Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction, MICRO-36

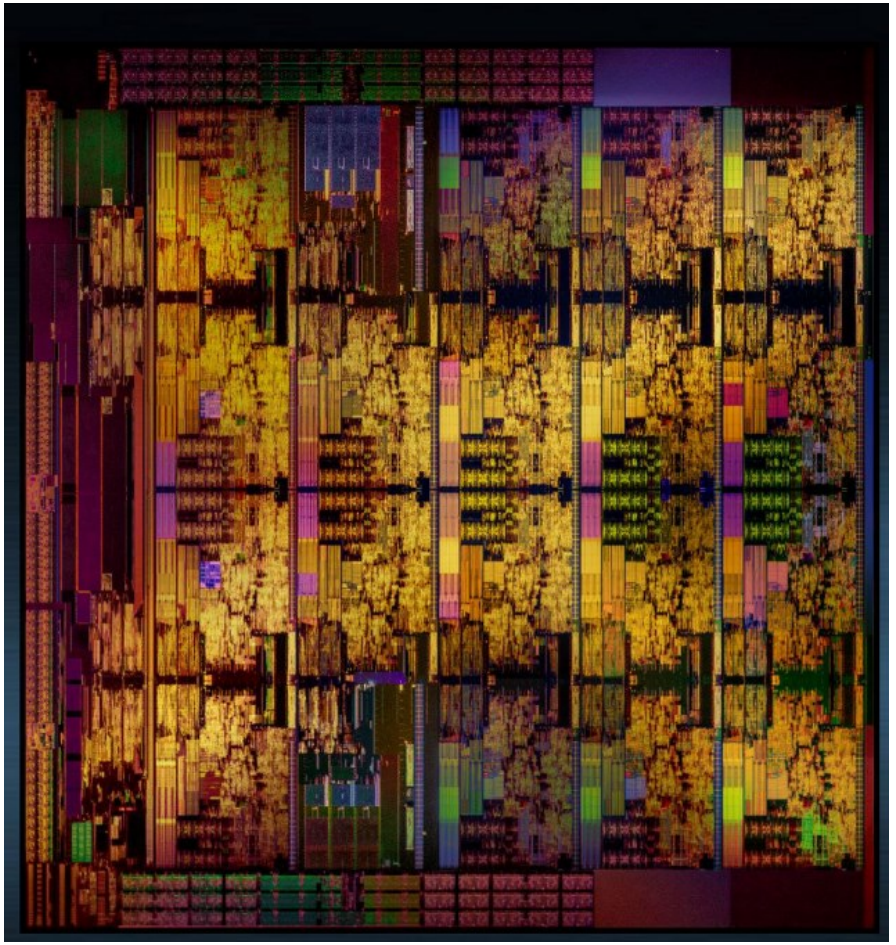
Intel Sandy Bridge, January 2011

- 4 core



Intel Skylake-X, Core i9-7980XE, 2017

- 18 core



Syllabus (3/3)

Course schedule/Required learning		
	Course schedule	**Required learning**
Class 1	Design and Analysis of Computer Systems	Understand the basic of design and analysis of computer systems.
Class 2	Instruction Set Architecture	Understand the examples of instruction set architectures
Class 3	Memory Hierarchy Design	Understand the organization of memory hierarchy designs
Class 4	Pipelining	Understand the idea and organization of pipelining
Class 5	Instruction Level Parallelism: Concepts and Challenges	Understand the idea and requirements for exploiting instruction level parallelism
Class 6	Instruction Level Parallelism: Instruction Fetch and Branch Prediction	Understand the organization of instruction fetch and branch predictions to exploit instruction level parallelism
Class 7	Instruction Level Parallelism: Advanced Techniques for Branch Prediction	Understand the advanced techniques for branch prediction to exploit instruction level parallelism
Class 8	Instruction Level Parallelism: Dynamic Scheduling	Understand the dynamic scheduling to exploit instruction level parallelism
Class 9	Instruction Level Parallelism: Exploiting ILP Using Multiple Issue and Speculation	Understand the multiple issue mechanism and speculation to exploit instruction level parallelism
Class 10	Instruction Level Parallelism: Out-of-order Execution and Multithreading	Understand the out-of-order execution and multithreading to exploit instruction level parallelism
Class 11	Multi-Processor: Distributed Memory and Shared Memory Architecture	Understand the distributed memory and shared memory architecture for multi-processors
Class 12	Thread Level Parallelism: Coherence and Synchronization	Understand the coherence and synchronization for thread level parallelism
Class 13	Thread Level Parallelism: Memory Consistency Model	Understand the memory consistency model for thread level parallelism
Class 14	Thread Level Parallelism: Interconnection Network and Man-core Processors	Understand the interconnection network and many-core processors for thread level parallelism

Adaptive Computing Research Initiative (ACRi)

- The aim
  - Aiming to develop the high-performance Adaptive Computing Systems that utilize FPGAs
  - Working out to distribute the FPGA-related technologies, including our developed systems, as an outreach activity for research results
- Main research theme
  1. Development for FPGA accelerator to speed up processing of **AI** etc.
  2. Development for FPGA accelerators and FPGA systems for **IoT**.
- Activity
  - Establishment Date: April 1st 2020
  - Activity period : First period of 3 years

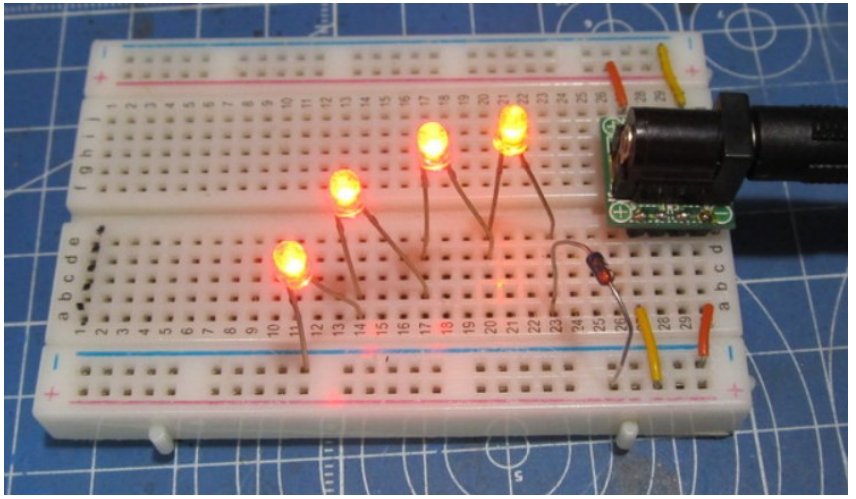
The Adaptive Computing Research Initiative is an organization to seek out and research ways to utilize FPGAs.

Please apply for your user account on this site today

- https://gw.acri.c.titech.ac.jp/wp/manual/apply-for-account

Discussion: software and hardware

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



Hardware to light up some LEDs