problem

electrons

problem

algorithm

electrons

problem

algorithm

program

electrons

problem

algorithm

program

runtime system
(VM, OS, MM)

electrons

problem

algorithm

program

runtime system
(VM, OS, MM)

ISA (architecture)

electrons

problem

algorithm

program

runtime system
(VM, OS, MM)

ISA (architecture)

microarchitecture

electrons

problem

algorithm

program

runtime system
(VM, OS, MM)

ISA (architecture)

microarchitecture

logic

electrons

problem

algorithm

program

runtime system
(VM, OS, MM)

ISA (architecture)

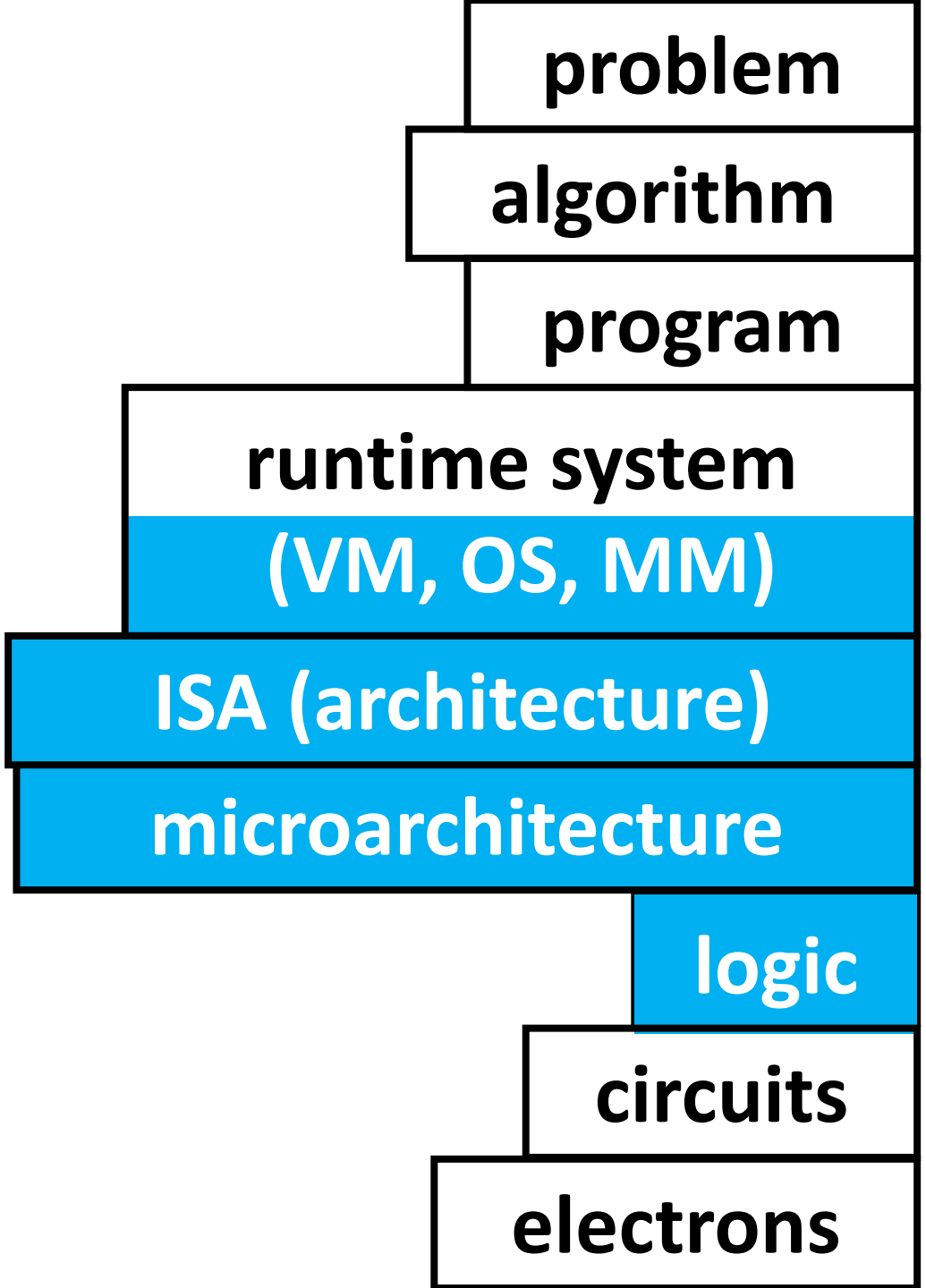
microarchitecture

logic

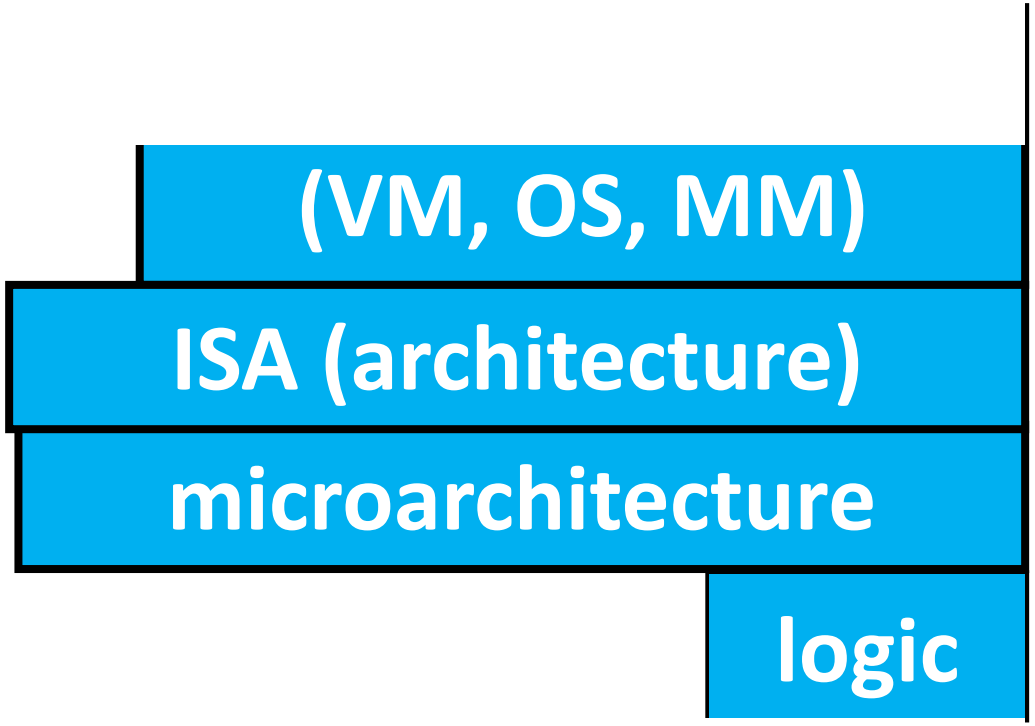
circuits

electrons

Computer System

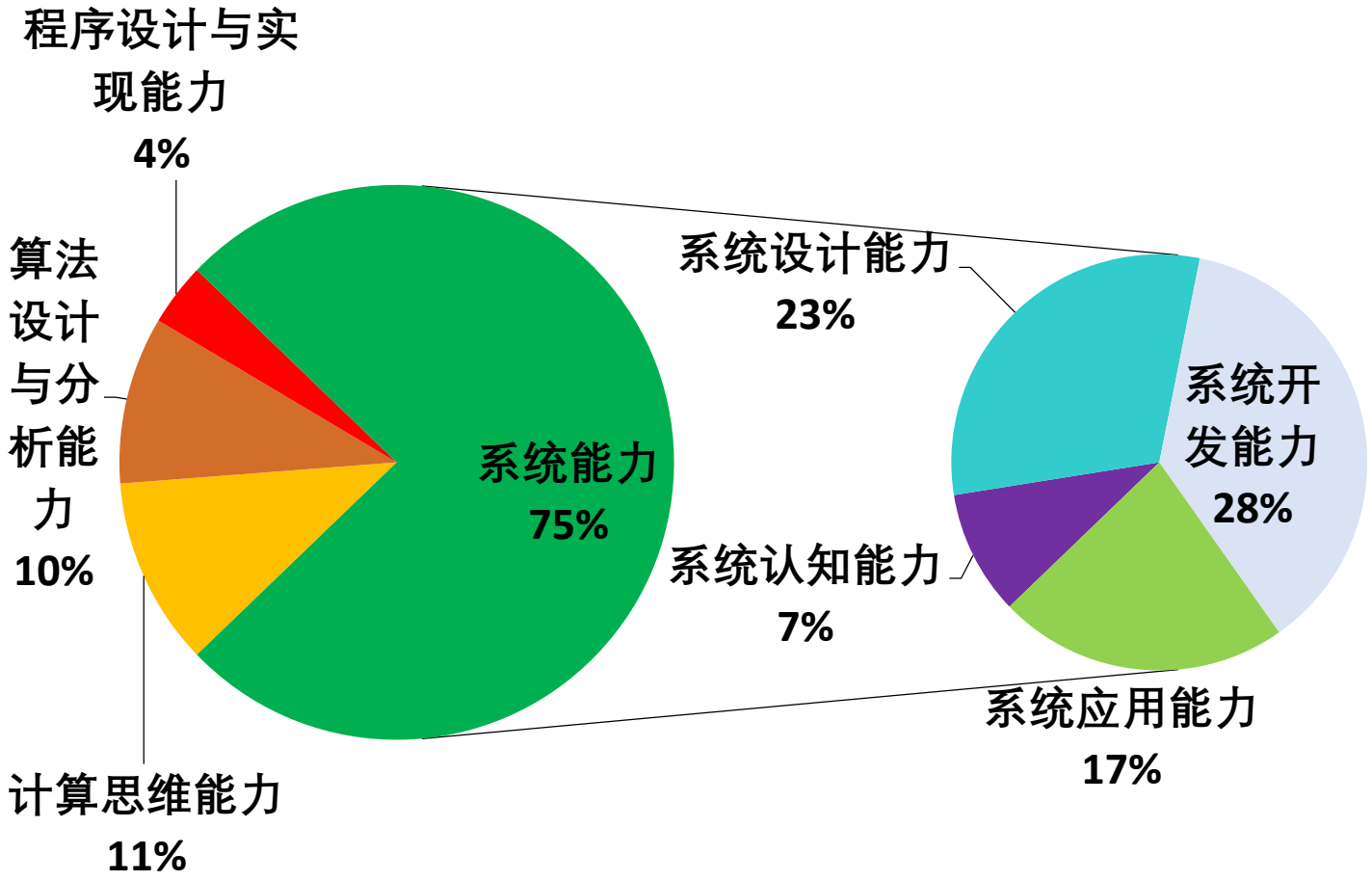


challenging



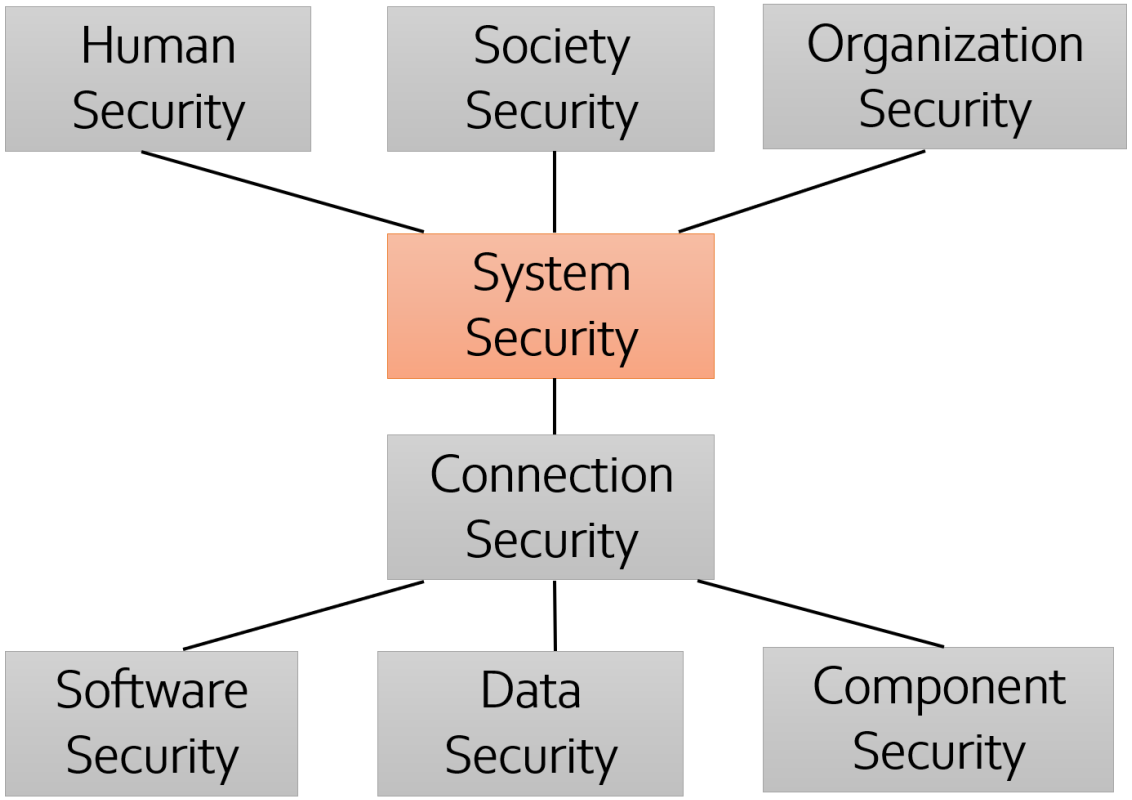
How about computer system competency?

- Computer science and technology
  - Computing Curricula 2020

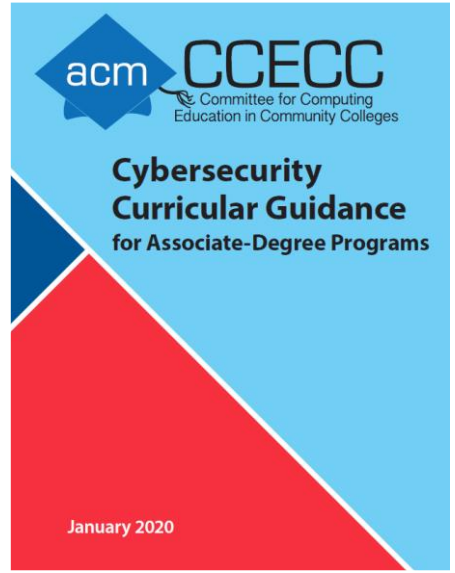


How about computer system competency?

- Cyber Security / Information Security
  - CSEC 2017
  - Cyber2yr 2020



CSEC2017



Cyber2yr2020

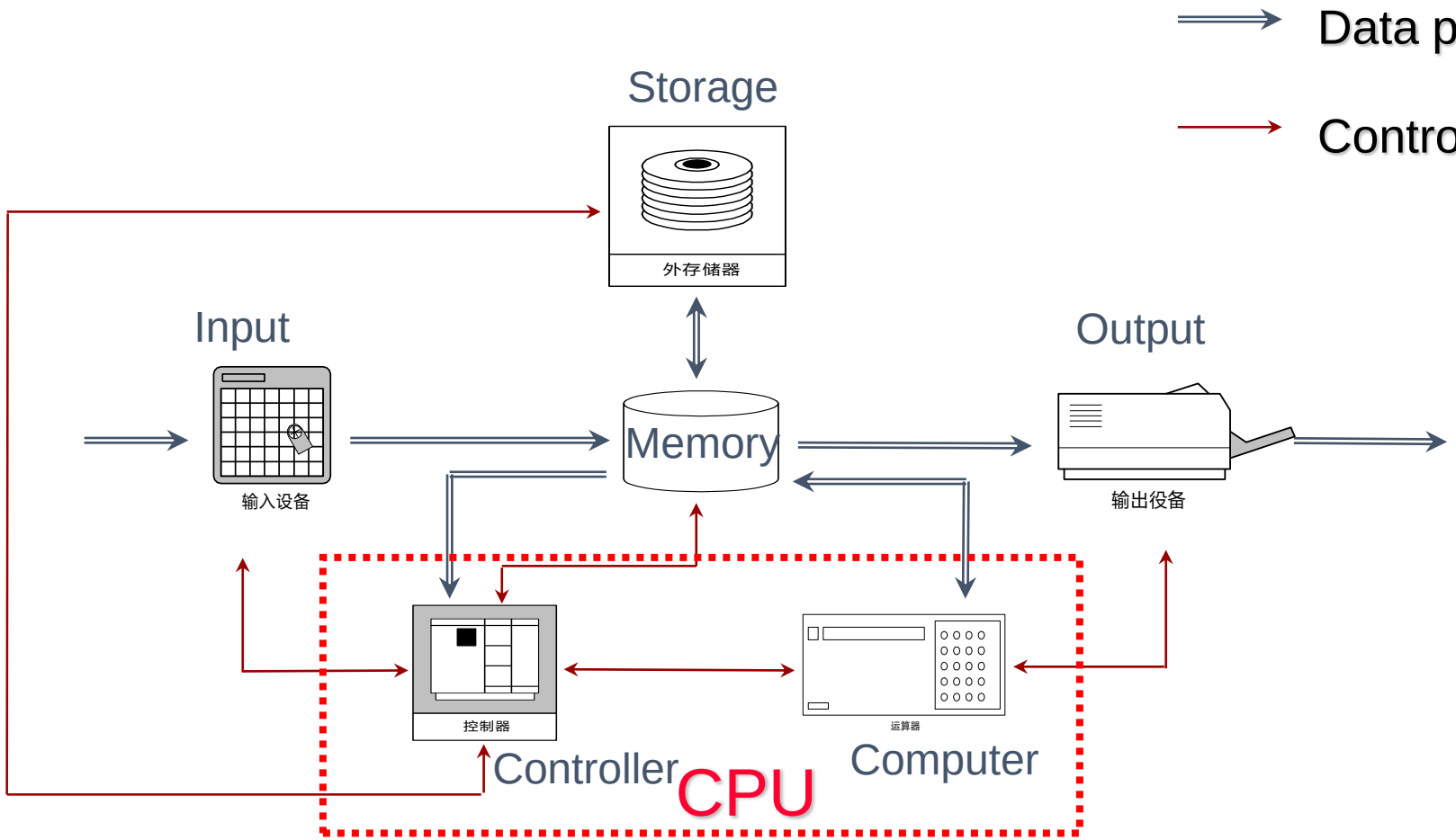
Chapter 1

Fundamentals of computer system

Von Neumann Structure

- Von Neumann structure: data and programs are in memory.
- CPU takes instructions and data from memory for operation and puts the results into memory

Von Neumann Structure



The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

Big Men (1)

- Mateo Valero
  - http://personals.ac.upc.edu/mateo/

- For important contributions to instruction level parallelism and superscalar processor design.

Big Men (2)

- John Leroy Hennessy
- Hennessy is one of the founders of MIPS Computer Systems Inc. as well as Atheros and served as the tenth President of Stanford University.
- Hennessy announced that he would step down in the summer of 2016. He was succeeded as President by Marc Tessier-Lavigne.
- Marc Andreessen called him "the godfather of Silicon Valley."

Big Men (3)

- David Andrew Patterson

- An American computer pioneer and academic who has held the position of professor of computer science at the University of California, Berkeley since 1976.

- He currently is Vice Chair of the Board of Directors of the RISC-V Foundation, and the Pardee Professor of Computer Science, Emeritus at UC Berkeley.

one big thing at a time

Big Men (3)

JOHN L HENNESSY

United States – 2017

CITATION

For pioneering a systematic, quantitative approach to the design and evaluation of computer architectures with enduring impact on the microprocessor industry.

RESEARCH

DAVID PATTERSON

United States – 2017

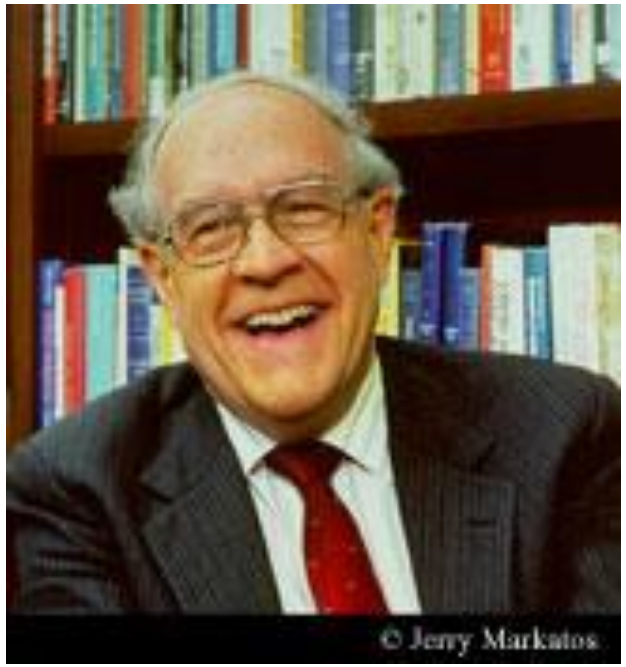
CITATION

For pioneering a systematic, quantitative approach to the design and evaluation of computer architectures with enduring impact on the microprocessor industry.

RESEARCH

- Turing Award for pioneering a systematic, quantitative approach to the design and evaluation of computer architectures with enduring impact on the microprocessor industry.
- Hennessy and Patterson created a systematic and quantitative approach to designing faster, lower power, and reduced instruction set computer (RISC) microprocessors.

Big Men (4)



Frederick P. Brooks

http://www.cs.unc.edu/~brooks/

- 1999 ACM Turing Award
- landmark contributions to computer architecture, operating systems, and software engineering."

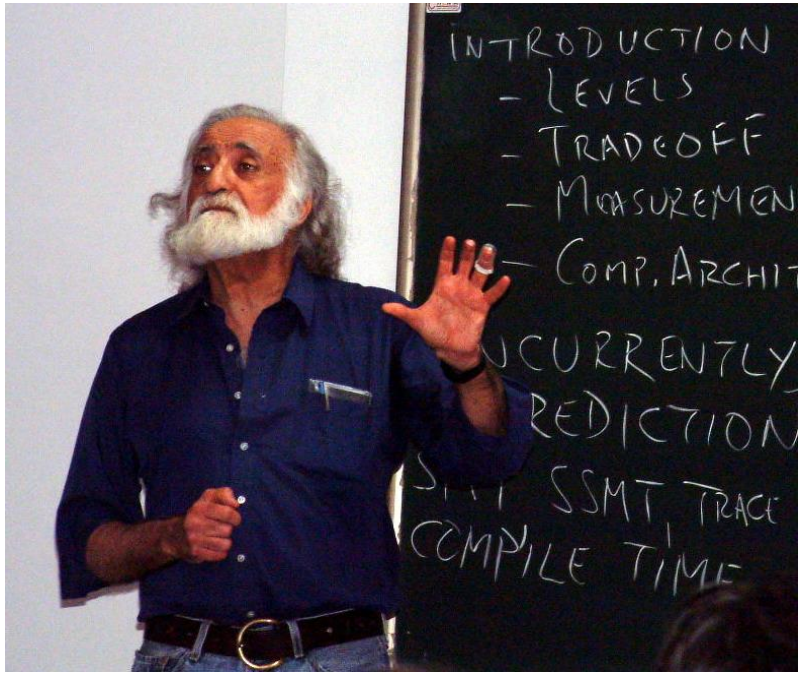
2004 Eckert-Mauchly Award

"For the definition of computer architecture and contributions to the concept of computer families and to the principles of instruction set design; for seminal contributions in instruction sequencing, including interrupt systems and execute instructions; and for contributions to the **IBM 360 instruction set architecture**."

Big Men (5)

- Robert Marco Tomasulo
- A computer scientist, and the inventor of the Tomasulo algorithm.
- Tomasulo was the recipient of the 1997 Eckert–Mauchly Award "or the ingenious Tomasulo algorithm, which enabled out-of-order execution processors to be implemented."

Big Men (6)



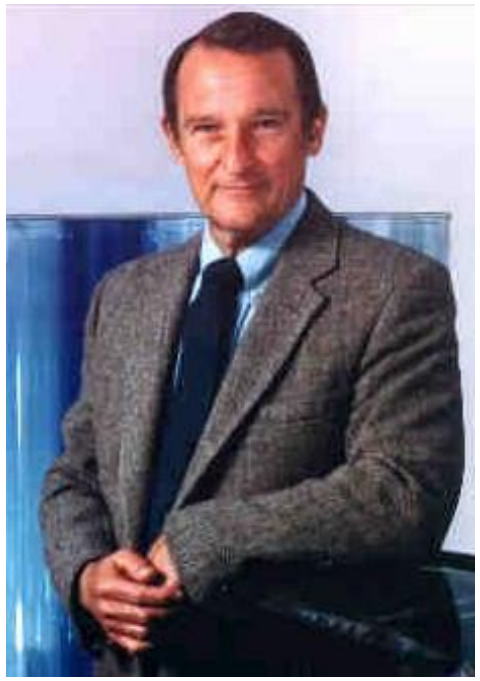
- Yale Patt
- For important contributions to instruction level parallelism and superscalar processor design.

Big Men (7)

- Michael J. Flynn

- http://www.cpe.calpoly.edu/IAB/flynn.html
- For his important and seminal contributions to processor organization and classification, computer arithmetic and performance evaluation.

Big Men (8)



- Seymour Cray
- For a career of achievements that have advanced supercomputing design.
- In 1958, the world's first transistor-based supercomputer was designed and built.
- At the same time, he has made significant contributions to the production of reduced instruction (RISC) high-end microprocessors, and is one of the most important figures in the field of high-performance computers.

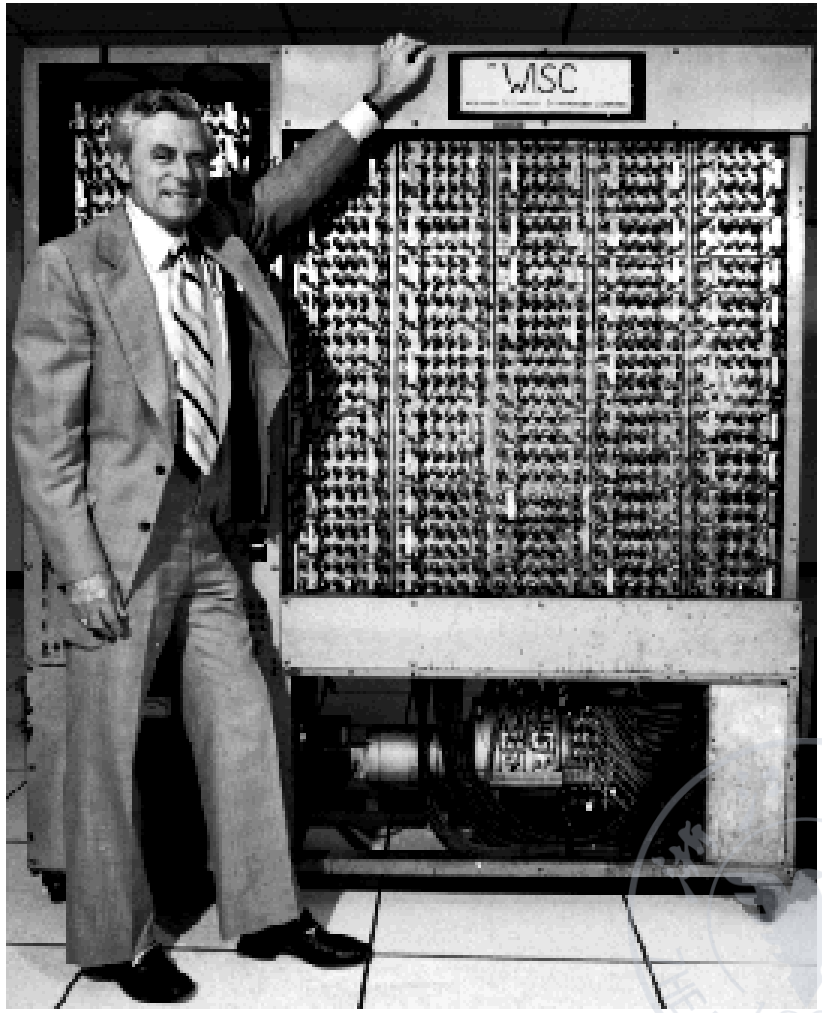
Big Men (9)

- Amdahl, Gene M.

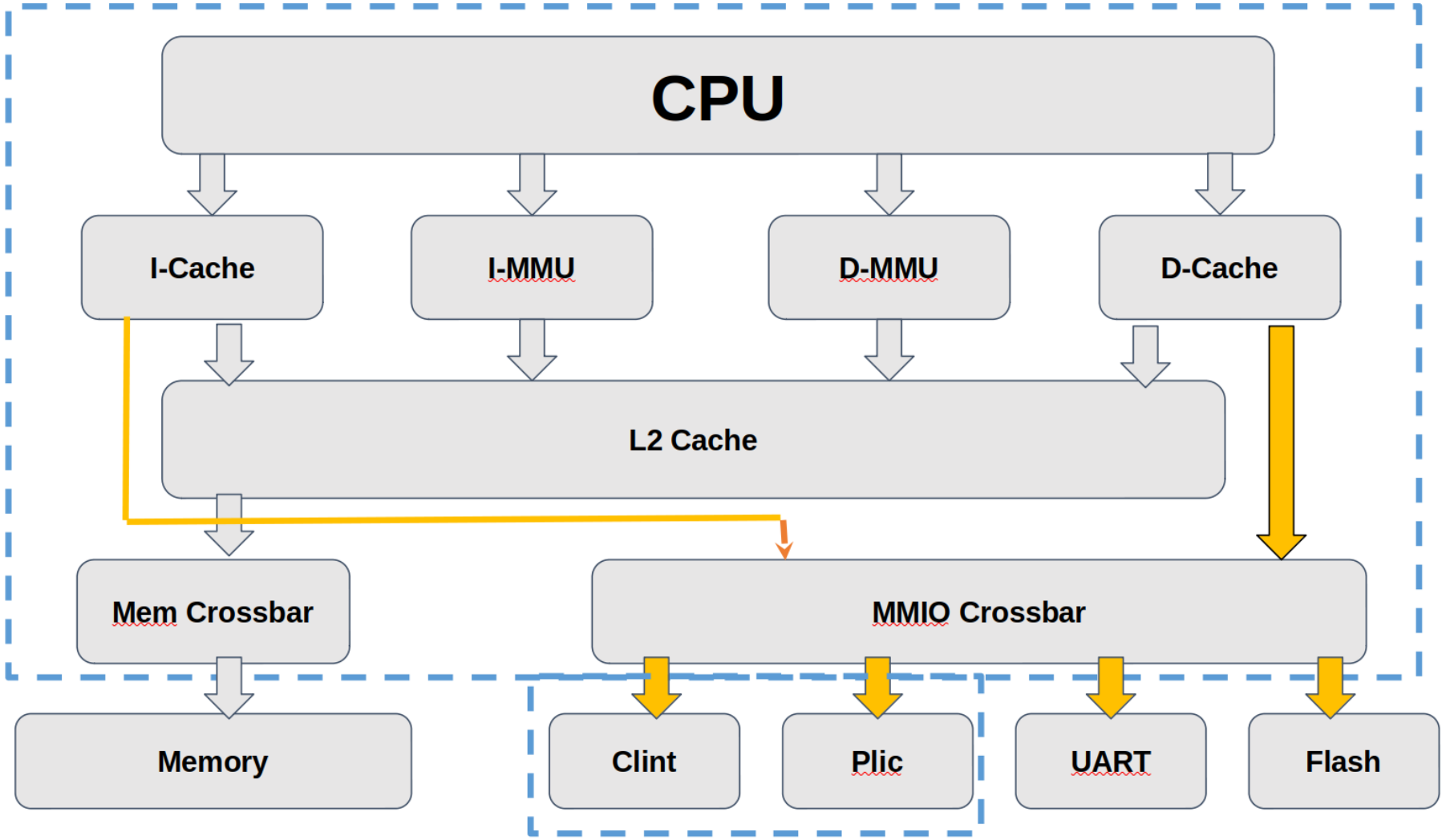
- For outstanding innovations in computer architecture, including pipelining, instruction look-ahead, and cache memory.

In 1975, Dr. Amdahl stands beside the Wisconsin Integrally Synchronized Computer (WISC), which he designed in 1950. It was built in 1952. *(Image courtesy of Dr. Gene M. Amdahl.)*

From Computer Desktop Encyclopedia
Reproduced with permission.
© 1999 Dr. Gene M. Amdahl

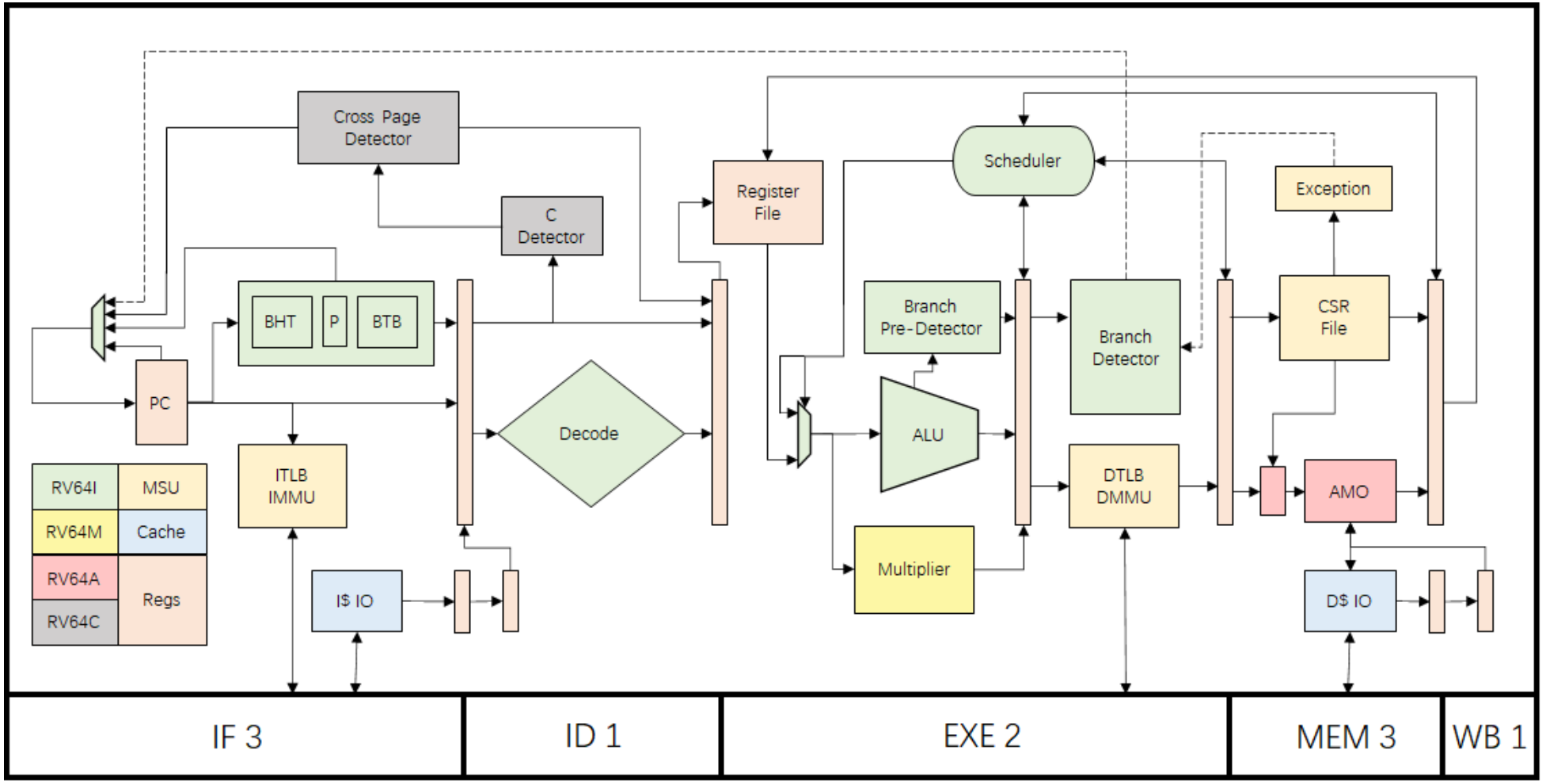


What is more? Your own CPU.



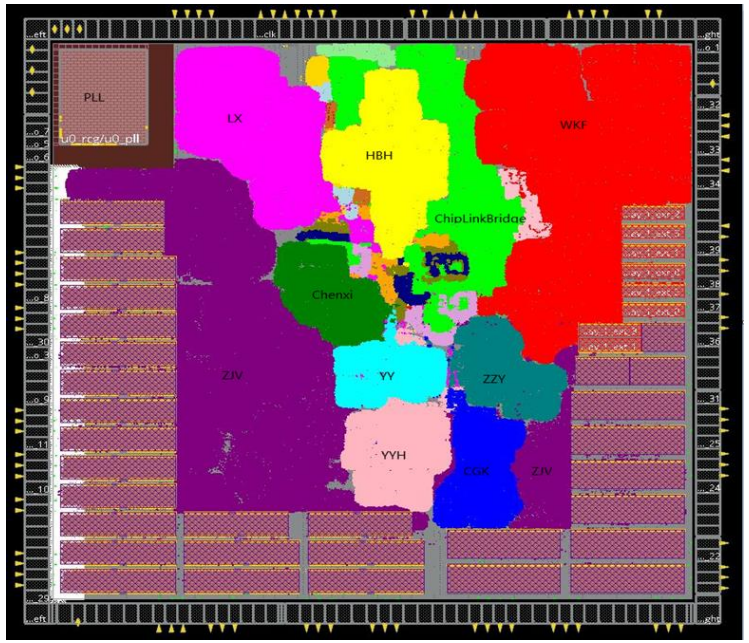
ZJV 1.0 技术指标——微结构设计

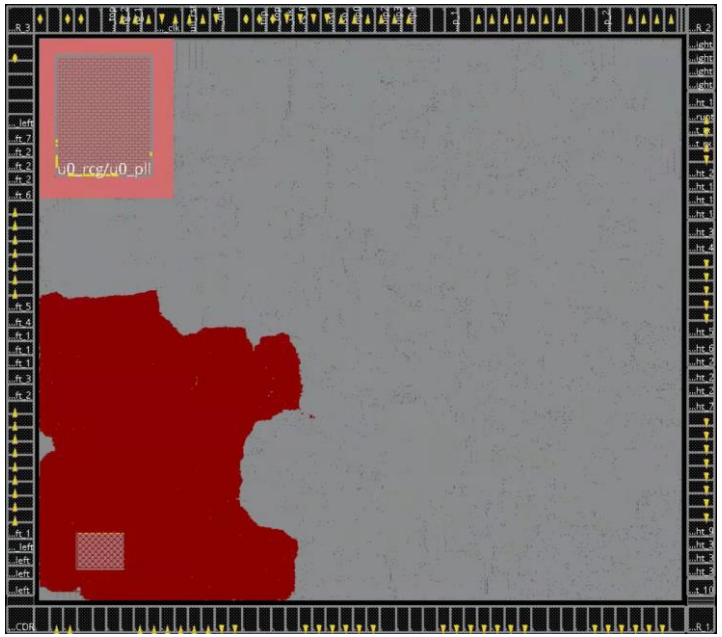
What is more? Your own CPU.



ZJV 1.0 技术指标——微结构设计

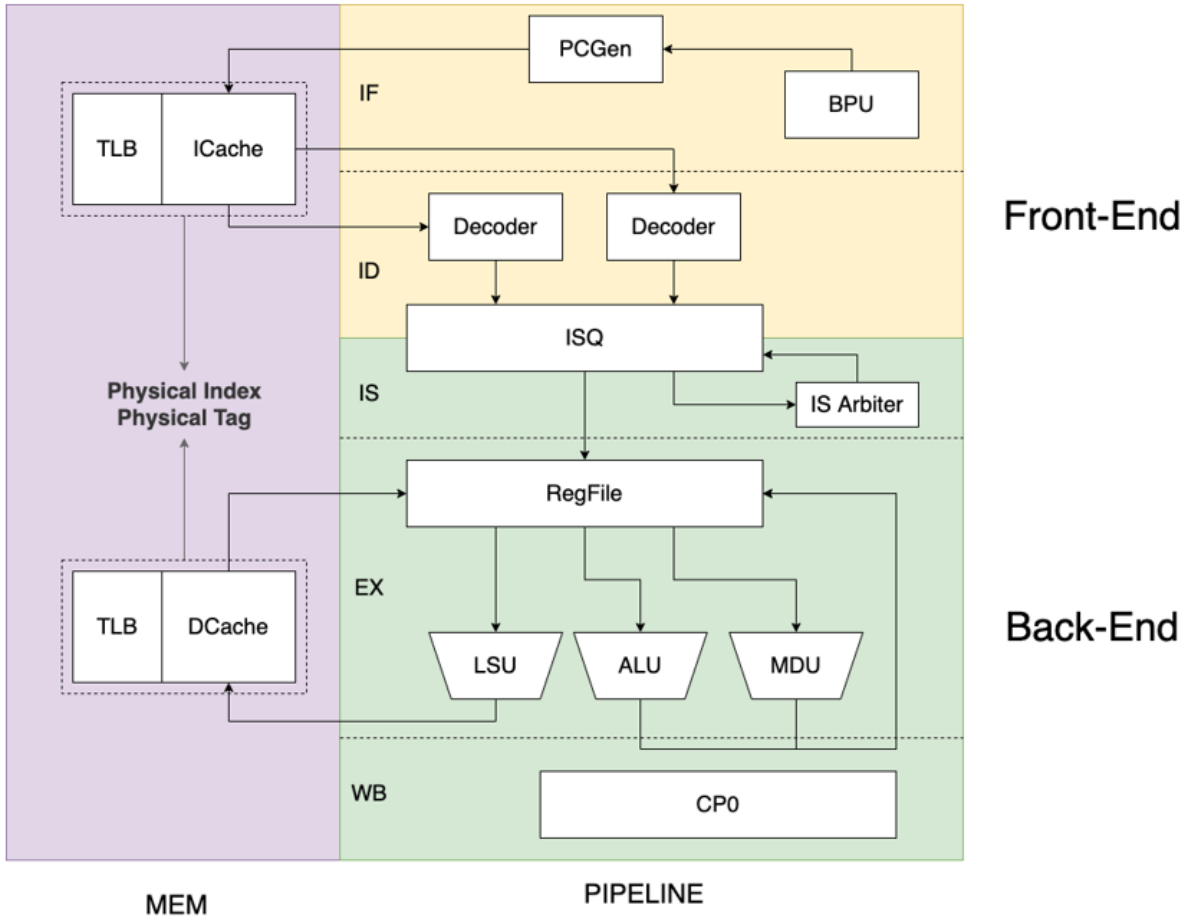
ZJV1.0——流片成果（一生一芯二期）





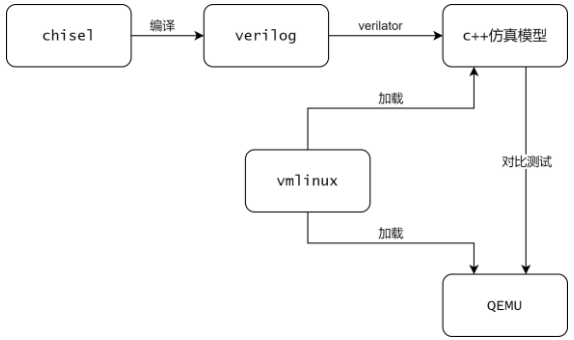
What is more? Your own CPU.

- Amipsel:
  - 五段流水32位处理器
  - 高可配置性
    - 前端（IF/ID）：单/双发射
    - 后段（IS/EX/WB）：最多为三发射
    - TLB、原子指令等
  - Chisel语言开发



What is more? Your own CPU.

- 自主搭建的差分测试框架dtqemu
  - 模拟RAM和串口等外设
  - 同步中断
  - 比对qemu和AMipsel的寄存器
- 便捷Linux的调试



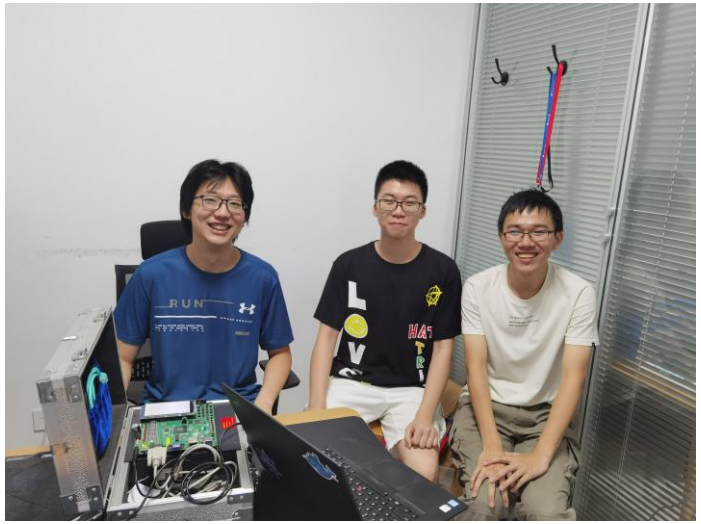
操作系统移植

支持pmon的全部指令运行

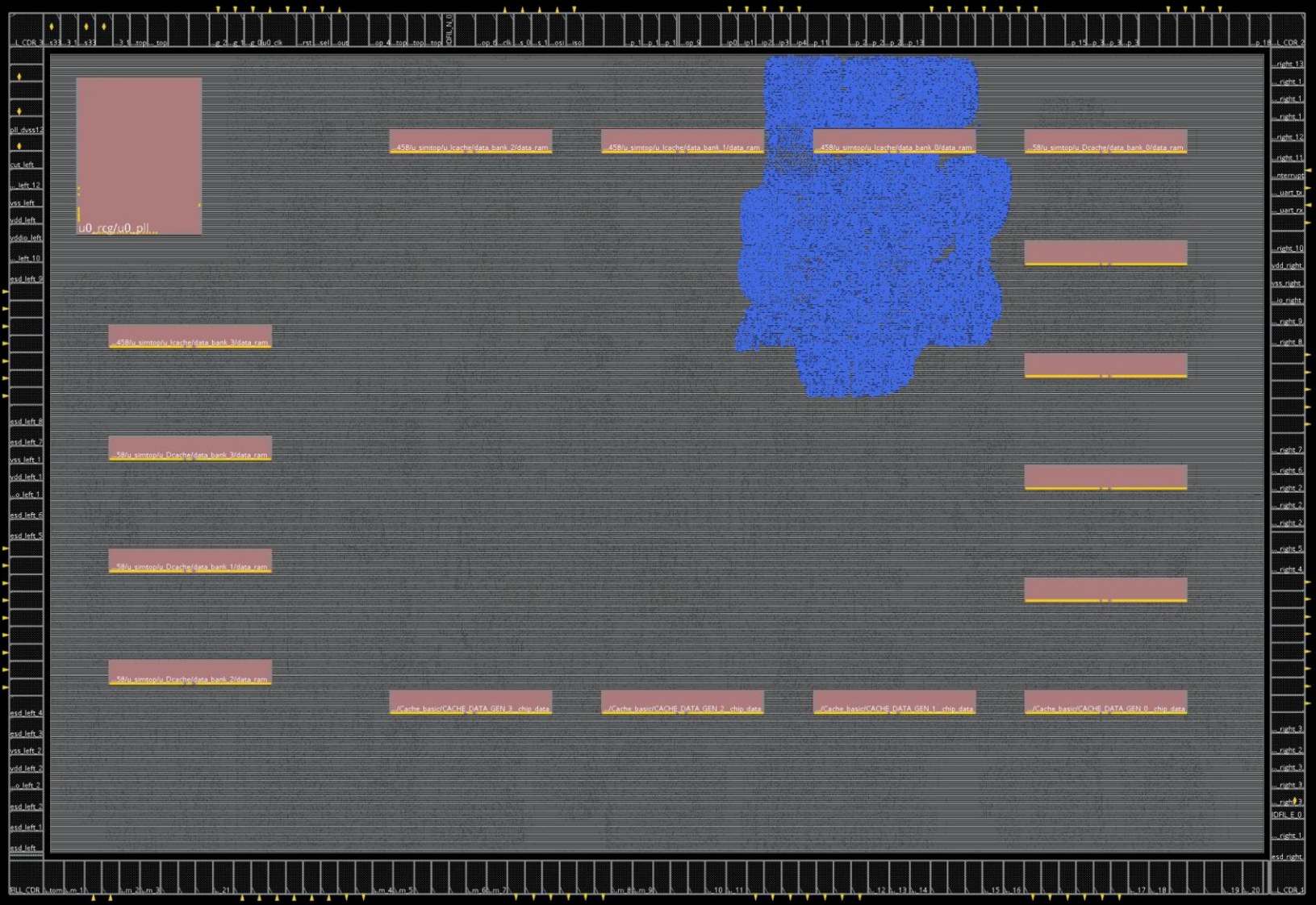
支持ucore的运行

完成Linux的启动流程到用户态

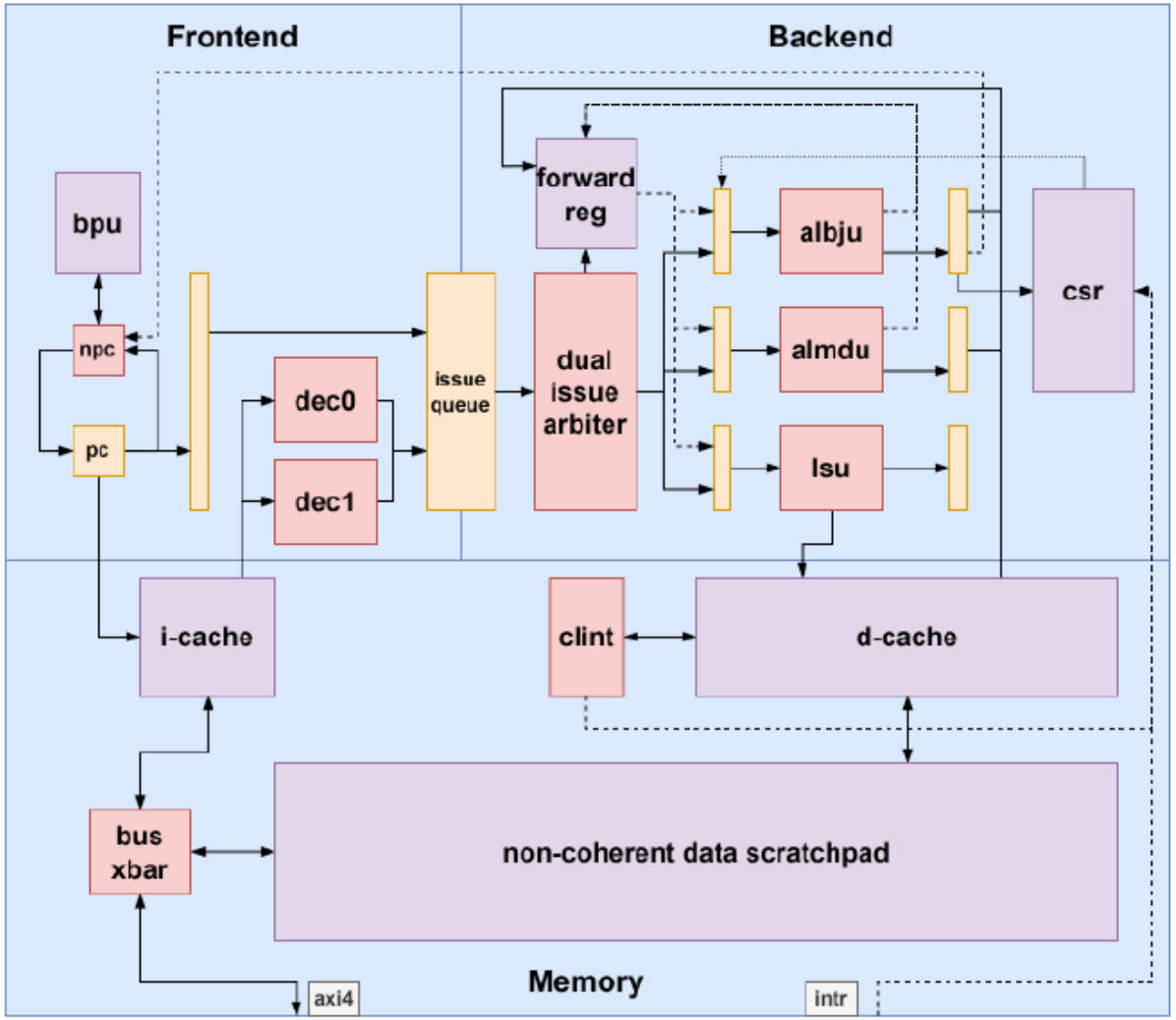
团队赛入围决赛的参赛队				预赛分数		
序号	学校	队伍名称	参赛学生	功能分	性能分	频率MHz
1	浙江大学	amipsel	苑子琦、沈栩捷、石昊洋、李鸿屹	100	83.715	104
2	西北工业大学	西北工业大学二队	江嘉熙、杨益治、魏大昊、甲世东	100	79.030	155
3	西北工业大学	西北工业大学一队	王翰墨、王玉佳、吴奇、杨士欣	100	78.402	150
4	清华大学	琉璃层露队	黄嘉良、刘松铭、杨倚天、余泰来	100	66.563	120
5	济南大学	济南大学一队	刘子臻、张鹏、李川、王钰涵	100	60.729	100
6	北京航空航天大学	北京航空航天大学一队	田韵豪、曹文轩、陈昊	100	58.711	90
7	哈尔滨工业大学（威海）	CommyMIPS	李宗泽、田林、王朝伟、蒋巍威	100	56.701	115
8	北方工业大学	四脸懵逼队	潘智文、梁家宇、兰天翔、黄超	100	55.254	97
9	哈尔滨工业大学（深圳）	芯之怪盗团	朱旗、尚奕扬、邓起源、丁浩卓	100	52.343	85
10	哈尔滨工业大学	0xdead10c	陶子康、代昆、马庄宇	100	52.491	80

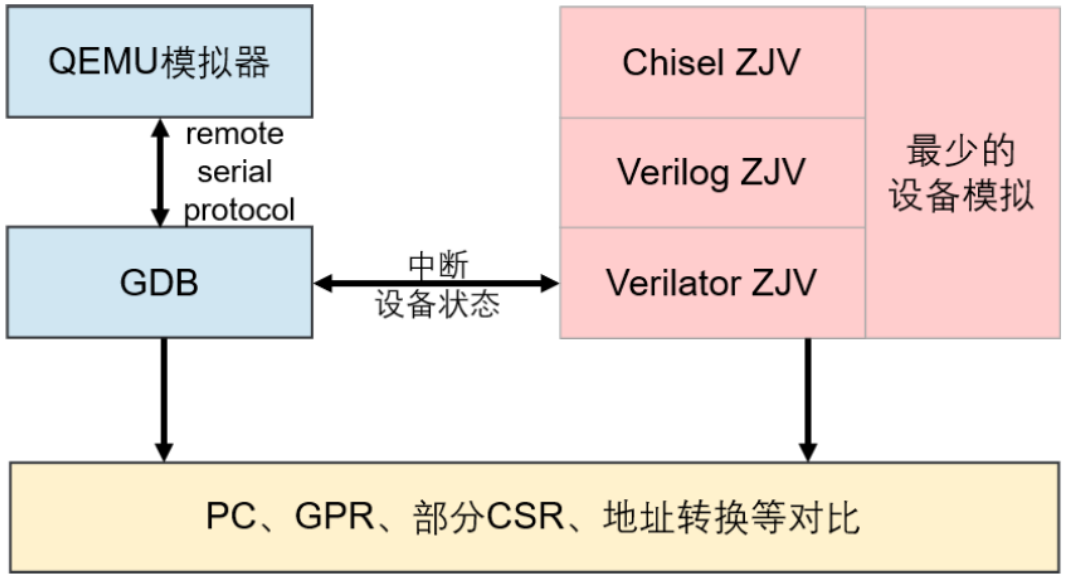


ZJV2.0 ——流片成果（一生一芯三期）



What is more? Your own CPU + system.



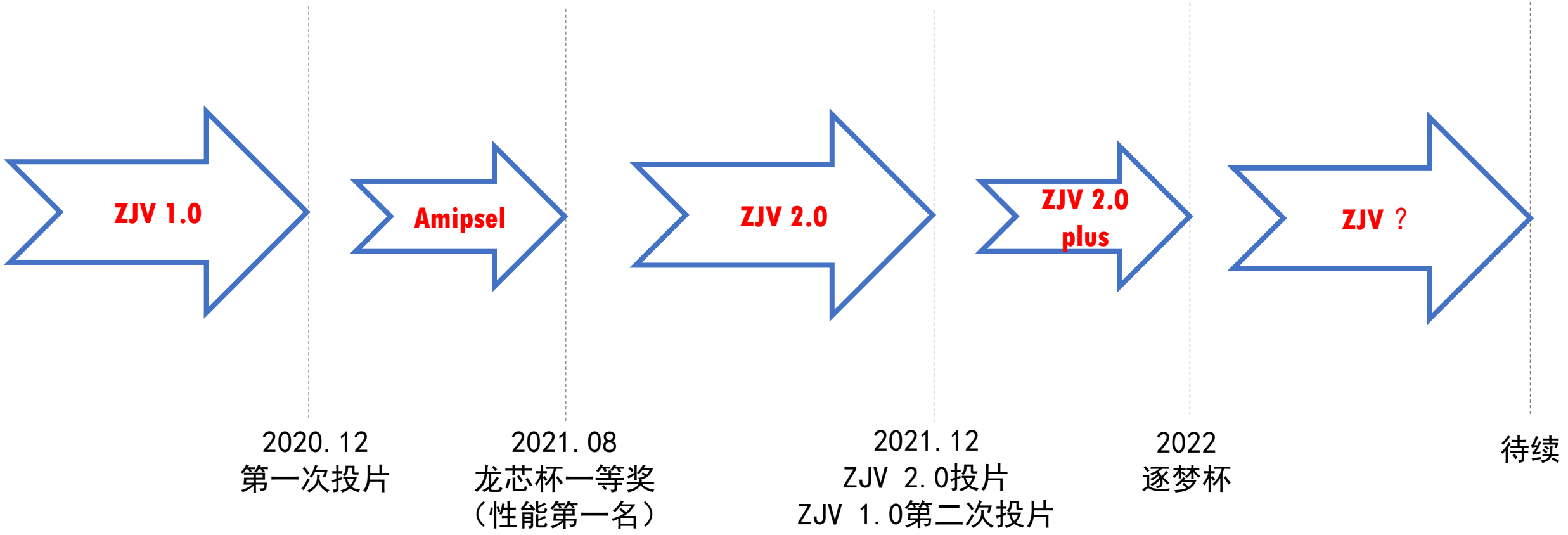


逐梦杯总决赛:

- 自主设计优化，获得1789性能分
- 面向通用系统，启动Linux 发行版
- 着眼安全特性，保护数据完整性和机密性

What is more? Our idea (not only CPU)

ZJV = ZheJiang University + RISC-V



Classes of Computers

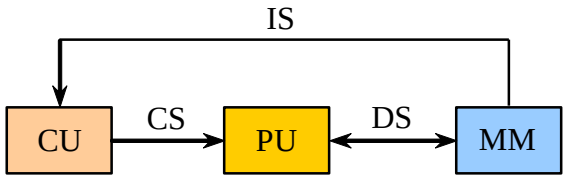
- Desktop computers (or Personal Computers)
  - General purpose, variety of software
  - Emphasize good performance for a single user at relatively low cost.
  - Mostly execute third-party software
- Server computers
  - Emphasize great performance for a few complex applications.
  - Or emphasize reliable performance for many users at once.
  - Greater computing, storage, or network capacity than personal computers.
- Embedded computers
  - Largest class and most diverse.
  - Hidden as components of systems.
  - Stringent power/performance/cost constraints.

Classes of Computers

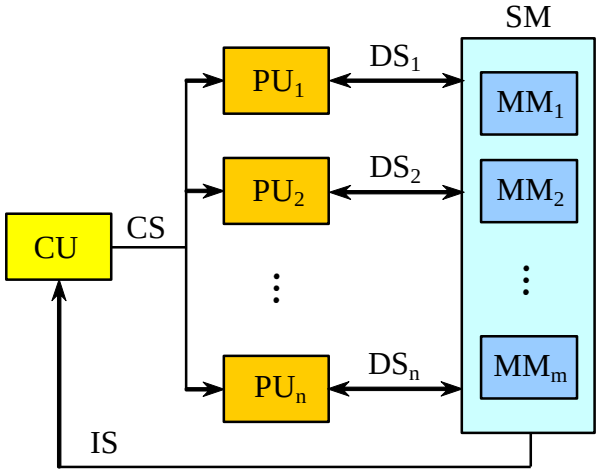
- Personal Mobile Devices
  - Smartphones
  - Tablet/iPad
  - generally have the same design requirements as PCs

- Supercomputer
  - Computer cluster
  - High capacity, performance, reliability
  - Range to building sized

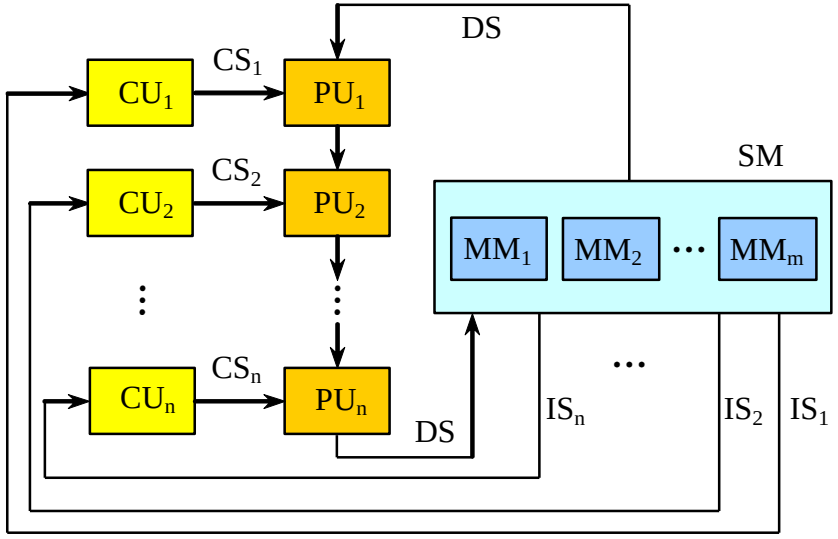
Classed By Flynn



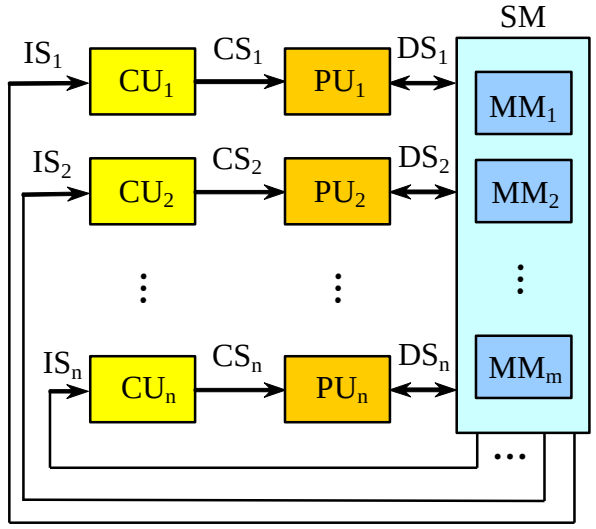
（a）SISD 计算机



（b）SIMD 计算机



（c）MISD 计算机



（d）MIMD 计算机

IS：Instruction stream

DS：Data stream

CS：Control stream

CU：Control unit

PU：Process unit

MM&SM：Memory

What You Will Learn

- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- What is parallel processing

Understanding Performance

- Algorithm
  - Determines number of operations executed
- Programming language, 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 operations are executed

How you will learn?

- Possibility:
  - standing on the shoulder of giants.
- Concepts, Ideas and Principles
- Quantitative approaches
- Hit the problem and right way to solve problem

- As a man sows, so he shall reap.

Summarize

- According to the process of using data, computers are developing in three fields:
  - Speed up processing (parallel)
  - Speed up transmission (accuracy)
  - Increase storage capacity and speed up storage (reliability)
- The central issue discussed and studied in this course is
  - Processing
  - Storage
  - transmission

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
    - User CPU time : CPU time spent in the program itself
    - System CPU time: CPU time spent in the OS, performing tasks on behalf of the program.
  - Different programs are affected differently by CPU and system performance

The improvement of computer architecture

- Improvement of input / output
- The development of memory organization structure
  - Associative memory and associated processor
  - General register group
  - Cache
- Two directions of instruction set development：
  - CISC
  - RISC
- Parallel processing technology
  - How to develop parallelism in traditional machines?
  - Develop parallel technologies at different levels.
  - For example, micro operation level, instruction level, thread level, process level, task level, etc.

Quantitative approaches

CPU performance formula

Amdahl's Law

Measuring Data Size

- bit - Binary digit
- nibble - four bits
- byte - eight bits
- word - four bytes (32 bits) on many embedded/mobile processors and eight bytes (64 bits) on many desktops and servers.
- kibibyte (KiB) [kilobyte (KB)] - 2^{10} (1,024) bytes
- mebibyte (MiB) [megabyte (MB)] - 2^{20} (1,048,576) bytes
- gibibyte (GiB) [gigabyte (GB)] - 2^{30} (1,073,741,824) bytes
- tebibyte (TiB) [terabyte (TB)] - 2^{40} (1,099,511,627,776) bytes
- pebibyte (PiB) [petabyte (PB)] - 2^{50} (1,125,899,906,842,624) bytes

CPU Performance

In order to determine the effect of a design change on the performance experienced by the user, we can use the following relation:

$$CPU\ Execution\ Time = CPU\ Clock\ Cycles \times Clock\ Period$$

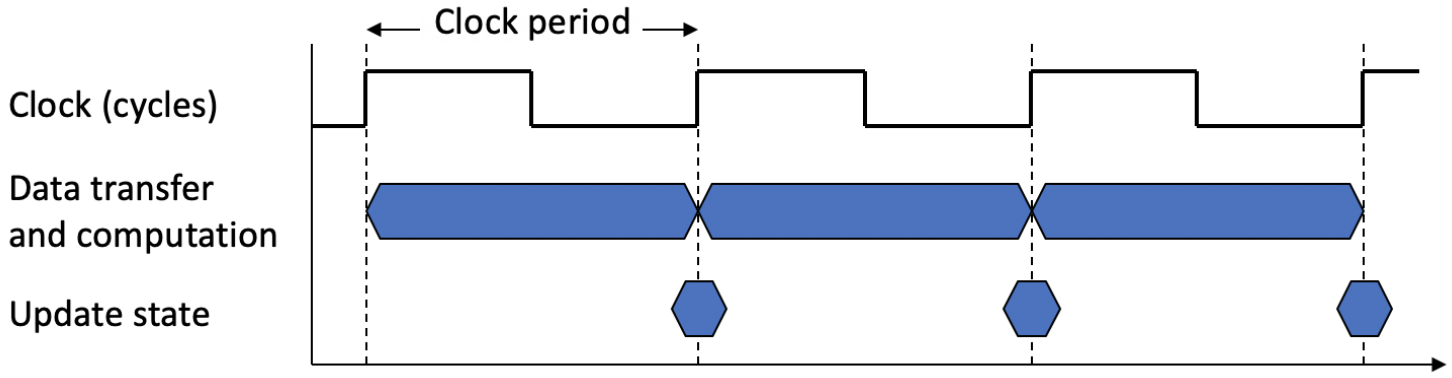
Alternatively,

$$CPU\ Execution\ Time = \frac{CPU\ Clock\ Cycles}{Clock\ Rate}$$

Clearly, we can reduce the execution time of a program by either reducing the number of clock cycles required or the length of each clock cycle.

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

Instruction Count and CPI

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction（CPI）
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

$$CPI = \frac{CPU\ Clock\ Cycles}{Instruction\ Count}$$

$$CPU\ Clock\ Cycles = Instructions\ for\ a\ Program \times Average\ Clock\ Cycles\ Per\ Instruction$$

$$CPU\ Time = Instruction\ Count \times CPI \times Clock\ Period$$

$$CPU\ Time = \frac{Instruction\ Count \times CPI}{Clock\ Rate}$$

Components of CPU performance

The basic components of performance and how each is measured.

Component	Units of Measure
CPU Execution Time for a Program	Seconds for the Program
Instruction Count	Instructions Executed for the Program
Clock Cycles per Instruction	Average Number of Clock Cycles per Instruction
Clock Cycle Time (Clock Period)	Seconds per Clock Cycle

Instruction Count, CPI, and Clock Period combine to form the three important components for determining CPU execution time. *Just analyzing one is not enough!* Performance between two machines can be determined by examining non-identical components.

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
  - Which is faster, and by how much?

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps} \\ \text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}\end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5 Clock Cycles = 2×1 + 1×2 + 2×3 = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6 Clock Cycles = 4×1 + 1×2 + 1×3 = 9
  - Avg. CPI = 9/6 = 1.5

Performance Summary

The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

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

Amdahl's Law

Amdahl's Law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.

Amdahl's Law depends on two factors:

- The fraction of the time the enhancement can be exploited.
- The improvement gained by the enhancement while it is exploited.

$$Improved\ Execution\ Time = \frac{Affected\ Execution\ Time}{Amount\ of\ Improvement} + Unaffected\ Execution\ Time$$

Make the common case fast!

Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{improved} = \frac{T_{affected}}{improvemen\ t\ factor} + T_{unaffected}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

Can't be done!

Amdahl's Law

- The system performance acceleration rate is limited by the percentage of the execution time of the component to the total execution time in the system.
- Amdahl's law defines the *speedup* that can be gained by using a particular feature.

$$\text{Speedup} = \frac{\text{Performance for entire task}_{\text{Using Enhancement}}}{\text{Performance for entire task}_{\text{Without Enhancement}}}$$

$$= \frac{\text{Total Execution Time}_{\text{Without Enhancement}}}{\text{Total Execution Time}_{\text{Using Enhancement}}}$$

ZHEJIANG UNIVERSITY 1897

Amdahl's Law

- Based on the basic idea that:

Execution time $_{new}$ = Execution time $_{can\ not\ be\ enhance}$ + Execution time $_{enhanced}$

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Amdahl's Law

- Improved ratio: In the system before the improvement, the ratio of the execution time of the improvement part to the total execution time.
  - It is always less than or equal to 1.
  - For example: a program that needs to run for 60 seconds has 20 seconds of calculation that can be accelerated,
    Then the ratio is 20/60.
- Component speedup ratio: The multiple that can be improved after some improvements. It is the ratio of the execution time before the improvement to the execution time after the improvement.
  - Under normal circumstances, the component acceleration ratio is greater than 1.
  - For example: if the system is improved, the execution time of the improved part is 2 seconds, Before the improvement, its execution time was 5 seconds, and the component acceleration ratio was 5/2.

Amdahl's Law

- Example 1.1 Increasing the processing speed of a certain function in the computer system to *20 times* the original, but the processing time of this function only accounts for *40%* of the running time of the entire system. After adopting this method to improve performance, how much can the performance of the entire system improve?

Answer:

- Fraction $_{enhanced}$ = 40%
- Speedup $_{enhanced}$ = 20

$$\text{Speedup} = \frac{1}{0.6 + \frac{0.4}{20}} = 1.613$$

Amdahl's Law

- Example 1.2 After a computer system adopts floating-point arithmetic components, the floating-point arithmetic speed is increased by *20 times*, and the overall performance of a certain program of the system is increased by *5 times*. Try to calculate the proportion of the floating-point operations in this program.

Answer:

- Speedup $_{\text{overall}}$ = 5
- Speedup $_{\text{enhanced}}$ = 20

$$\frac{1}{(1 - \text{Fraction}) + \frac{\text{Fraction}}{20}} = 5$$

Fraction = 84.2%

Amdahl's Law

- A decreasing rule for performance improvement
  If only a part of the computing task is improved, the more the improvement, the more limited the overall performance improvement.

- Important inference: If only a part of the whole task is improved and optimized, the speedup obtained will not exceed:

1/ (1-the ratio can be improved)

Great Architecture Ideas

Great Architecture Ideas

- There are 8 great architectural ideas that have been applied in the design of computers for over half a century now.

- As we cover the material of this course, we should stop to think every now and then which ideas are in play and how they are being applied in the current context.

Great Architecture Ideas

- Design for Moore's law.
  - The number of transistors on a chip doubles every 18-24 months.
  - Architects have to anticipate where technology will be when the design of a system is completed.
- Use abstraction to simplify design.
  - Abstraction is used to represent the design at different levels of representation.
  - Lower-level details can be hidden to provide simpler models at higher levels.
- Make the common case fast.
  - Identify the common case and try to improve it.
  - Most cost efficient method to obtain improvements.
- Improve performance via parallelism.
  - Improve performance by performing operations in parallel.
  - There are many levels of parallelism – instruction-level, process-level, etc.

Great Architecture Ideas

- Improve performance via pipelining.
  - Break tasks into stages so that multiple tasks can be simultaneously performed in different stages.
  - Commonly used to improve instruction throughput.
- Improve performance via prediction.
  - Sometime faster to assume a particular result than waiting until the result is known.
  - Known as speculation and is used to guess results of branches.
- Use a hierarchy of memories.
  - Make the fastest, smallest, and most expensive per bit memory the first level accessed and the slowest, largest, and cheapest per bit memory the last level accessed.
  - Allows most of the accesses to be caught at the first level and be able to retain most of the information at the last level.
- Improve dependability via redundancy.
  - Include redundant components that can both detect and often correct failures.
  - Used at many different levels.

Why learn architecture?

Improving a program's performance is not as simple as reducing its memory usage. Modern programmers need to have an understanding of the issues "below the program":

- The parallel nature of processors.
  - How might you speed up your application by introducing parallelism via threading or multiprocessing?
  - How will the compiler translate and rearrange your own instruction-level code to perform instructions in parallel?
- The hierarchical nature of memory.
  - How can you rearrange your memory access patterns to more efficiently read data?
  - Ever heard of *page (cache) coloring*, *false sharing*, *side-channel attacks*?
- The translation of high-level languages into hardware instructions
  - What decisions are made by the compiler in the process of generating instruction-level statements?

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
- 8 great architectural ideas