CSE 331 / EEE 332 / ETE 332/EEE 453 Microprocessor Interfacing and Embedded System

DR. SHAIKH ASIF MAHMOOD
PROFESSOR, EEE, BUET

What we will learn?

- How a microprocessor is programmed
- Instruction Set Architecture(ISA) or just Architecture
- How a microprocessor is designed
- Logic level
- Component level

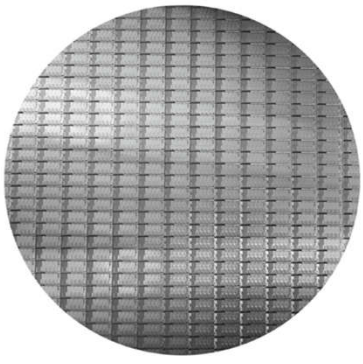
(Commonly known as microarchitecture)

What's inside a chip?

- The processor integrated circuit inside the A5 (Apple) package.
- The size of chip is 12.1 by 10.1 mm

How are chips made?

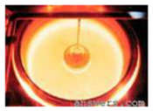
- Core i7 chips on a 12 inch wafer
- How are these chips made?
  - Essentially from **very pure** *sand!*
    - Make wafer from Sand (silicon)
    - Then fabricate chips on that wafer



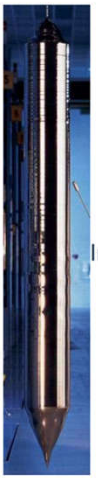
- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

Making the wafer

Pure Sand



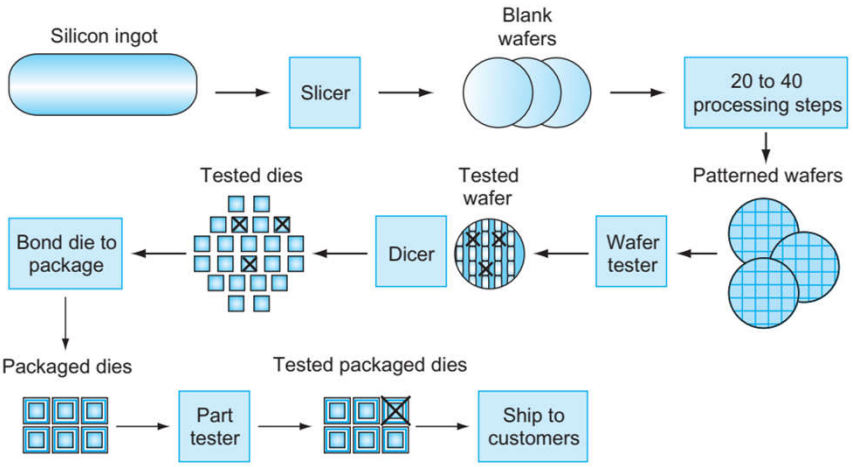
Melt in furnace



Ingot formation

Cut and polish

Chip manufacturing process



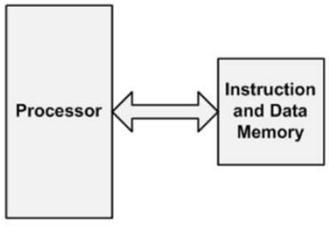
Computer Architecture

- The microprocessors are everywhere: mobile phone, TV, smart TV, laptop, smart watch, motor vehicles, airplanes, Xbox, PS, etc.
- Depending on the application, the architecture and microarchitecture of the processors can be different
- Two types of computer architecture,
  - Von Neumann architecture,
  - Harvard architecture

Harvard Architecture and Von Neumann Architecture

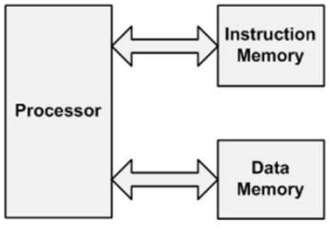
Von-Neumann

Instructions and data are stored in the same memory.



Harvard

Data and instructions are stored into separate memories.



Levels of Abstractior

- **Architecture**
  - A set of specifications that allows developers to write software and firmware
  - These include the ***instruction set***.
- **Microarchitecture**
  - The logical organization of the inner structure of the computer
- **Hardware or Implementation**
  - The realization or the physical structure, i.e., logic design and chip packaging

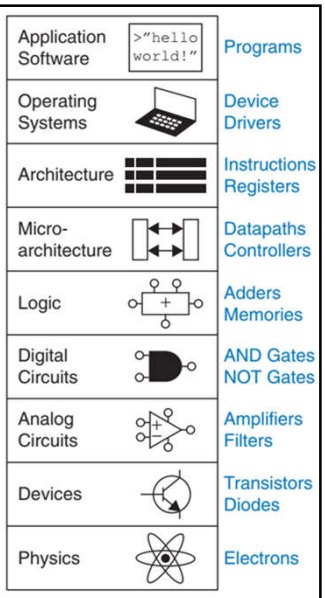
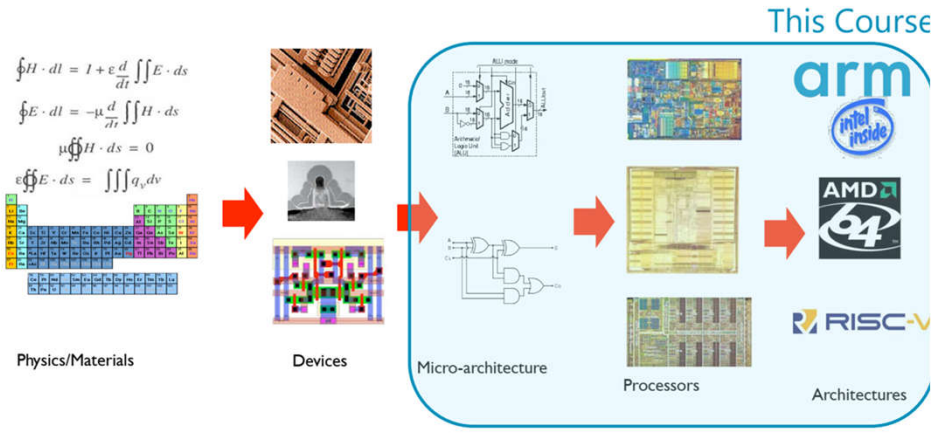


Figure 1.1 Levels of abstraction for an electronic computing system

Another view of Abstractions



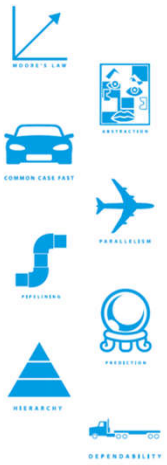
Slide Courtesy DSMC

Abstractions

- One of the great ideas to improve design is abstraction.
- One of the most important abstractions is the interface between the hardware and the lowest-level software.
- Because of its importance, it is given a special name: the instruction set architecture, or simply architecture, of a computer.

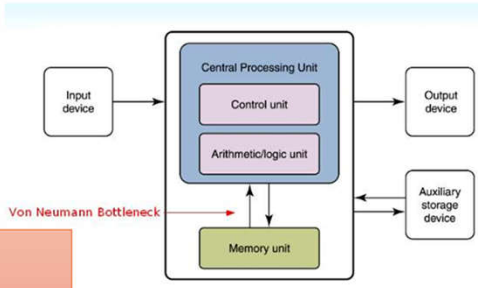
8 Great Ideas for Comp. Arch.

- Design for ***Moore's Law***
- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy



Stored Program Computer/Concept
(Von Neumann Arch.)

- Instructions and data are stored together in memory (in binary form).
- The program is then fetched from memory an instruction at a time and executed



Questions…
- How are the program represented?
- How do we implement an algorithm in the computer?
- How does a computer interpret a program?

Representing Programs

- We need some basic building blocks -- call them "instructions"
- What does "execute a program" mean?
- What instructions do we need?
- What should instructions look like?
- Is it enough to just specify the instructions?
- How complex should an instruction be?

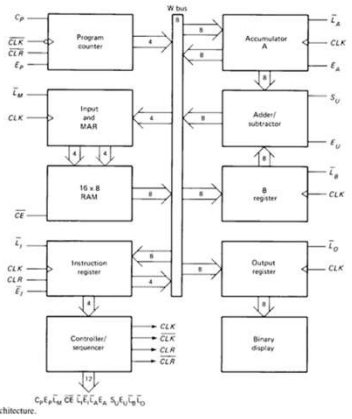
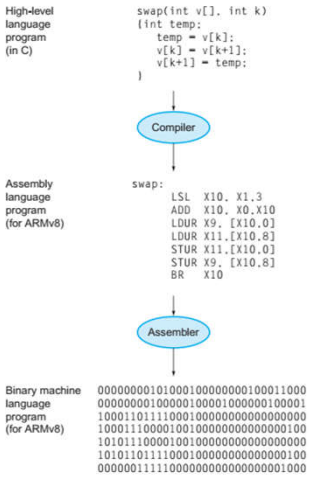


Fig. 10-1 SAP-1 architecture.

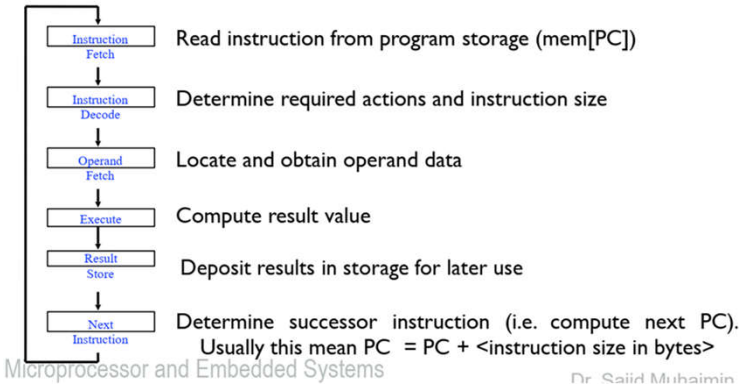
Levels of Programming Code

- **High-level language**
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- **Assembly language**
  - Textual representation of instructions
- **Hardware representation**
  - Binary digits (bits)
  - Encoded instructions and data



Program Execution

- This is the algorithm for a stored-program computer
- The Program Counter (PC) is the key



What instructions do we need?

- Basic operations are a good choice.
- Motivated by the programs people write.
- Math: Add, subtract, multiply, bit-wise operations
- Control: branches, jumps, and function calls.
- Data access: Load and store.
- The exact set of operations depends on many, many things
- Application domain, hardware trade-offs, performance, power, complexity requirements.

What should instructions look like?

- They will be numbers -- i.e., strings of bits
- It is easiest if they are all the same size, say 32 bits
- We can break up these bits into “fields” – like members in a class or struct.
- This sets some limits
- On the number of different instructions we can have
- On the range of values any field of the instruction can specify

Is specifying the instructions sufficient?

- No! We also must know what the instructions operate on.
- This is called the "Architectural State" of the machine

Registers -- a few named data values that instructions can operate on

Memory -- a much larger array of bytes that is available for storing values.

How big is memory? 32 bits or 64 bits of addressing.

- 64 is the standard today for desktops and mobiles.
- 32 for lower-end phones
- 16/32 for embedded processors

How complex should instructions be?

- More complexity
  - More different instruction types are required.
  - Increased design and verification costs.
  - More complex hardware.
  - More difficult to use -- What's the right instruction in this context?
- Less complexity
  - Programs will require more instructions -- poor code density
  - Programs can be more difficult for humans to understand

"Instruction Set Architecture (ISA) or simply "Architecture"

- An ISA is "the agreed-upon interface between all the software that runs on the machine and the hardware that executes it."
- The "contract" between software and hardware
- Functional definition of operations, modes, and storage locations supported by hardware
- Precise description of how to invoke, and access them
- Same ISA or Architecture can be implemented by different microarchitecture, hardware designs

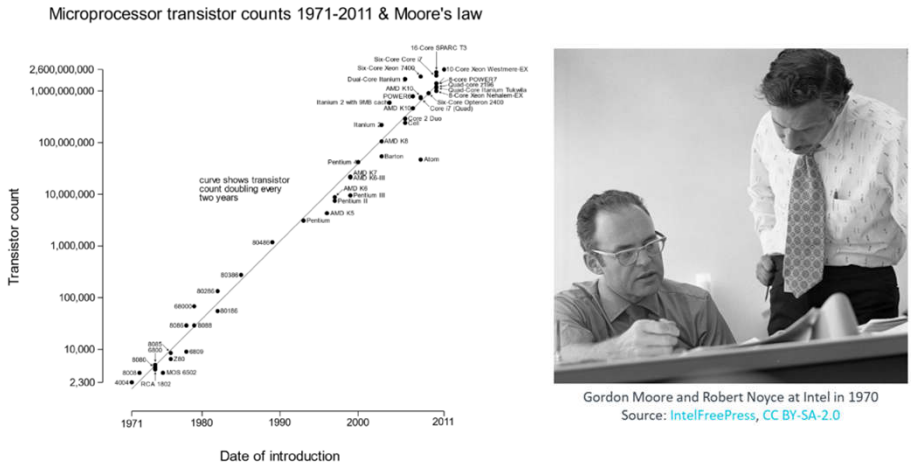
Historical Performance Trends

- By 1985, it was possible to integrate a complete microprocessor onto a single die or "chip."
- As fabrication technology improved, and transistors got smaller, the performance of a single core improved quickly.
- Performance improved at the rate of 52% per year for nearly 20 years (measured using SPEC benchmark data).

Moore's law

- Moore's Law predicted that the number of transistors we can integrate onto a chip, for the same cost (or minimal increase), doubles every 2 years.
- **1985 –Intel 386**: 275K transistors, die size =43 mm^2
- **2002 –Intel Pentium 4**: 42M transistors, die size = 217 mm^2
- **2024- Intel i9 9900K** – 4.2 Billion transistors!

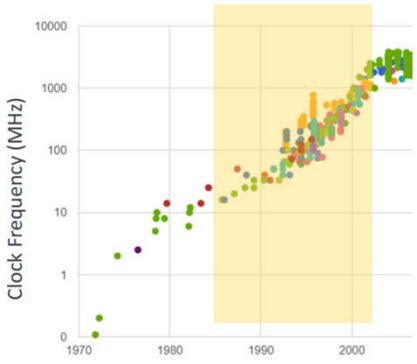
Moore's law



Gordon Moore and Robert Noyce at Intel in 1970
Source: IntelFreePress, CC BY-SA-2.0

Historical Performance Trends

- From 1985 to 2002, performance improved by ~ 800 times.
- Over time, technology scaling provided much greater numbers of faster and lower-power transistors.
- Clock frequency improved quickly between 1985 and 2002.



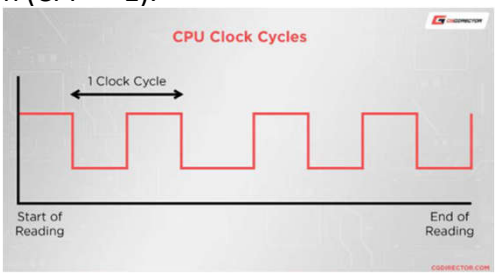
A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz. *Clock Frequency*, Stanford CPU DB. Accessed on Nov. 5, 2019. [Online]. Available: http://cpudb.stanford.edu/visualize/clock_frequency

Historical Performance Trends

- The "iron law" of processor performance:

• CPU Time = instructions executed x clocks per instruction (CPI) x clock cycle time (period)

- Early machines were limited by transistor count. As a result, they often required multiple clock cycles to execute each instruction (CPI >> 1).



Example (Comparing Performance)

Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

- Answer

We know that each computer executes the same number of instructions for the program; let's call this number I. First, find the number of processor clock cycles for each computer:

$$\text{CPU clock cycles}_A = I \times 2.0$$
$$\text{CPU clock cycles}_B = I \times 1.2$$

Example (Comparing Performance)

- Answer

Now we can compute the CPU time for each computer:

$$\text{CPU time}_A = \text{CPU clock cycles}_A \times \text{Clock cycle time}$$
$$= I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}$$

Likewise, for B:

$$\text{CPU time}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

Clearly, computer A is faster. The amount faster is given by the ratio of the execution times:

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

We can conclude that computer A is 1.2 times as fast as computer B for this program.

Example (Comparing Code Segments)

A compiler designer is trying to decide between two code sequences for a computer. The hardware designers have supplied the following facts:

	CPI for each instruction class		
	A	B	C
CPI	1	2	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

Code sequence	Instruction counts for each instruction class		
	A	B	C
1	2	1	2
2	4	1	1

Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

Example (Comparing Code Segments)

- Answer

Sequence 1 executes 2 + 1 + 2 = 5 instructions. Sequence 2 executes 4 + 1 + 1 = 6 instructions. Therefore, sequence 1 executes fewer instructions.

We can use the equation for CPU clock cycles based on instruction count and CPI to find the total number of clock cycles for each sequence:

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{C}_i)$$

This yields

$$\text{CPU clock cycles}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ cycles}$$
$$\text{CPU clock cycles}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ cycles}$$

So code sequence 2 is faster, even though it executes one extra instruction. Since code sequence 2 takes fewer overall clock cycles but has more instructions, it must have a lower CPI. The CPI values can be computed by

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

$$\text{CPI}_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2.0$$

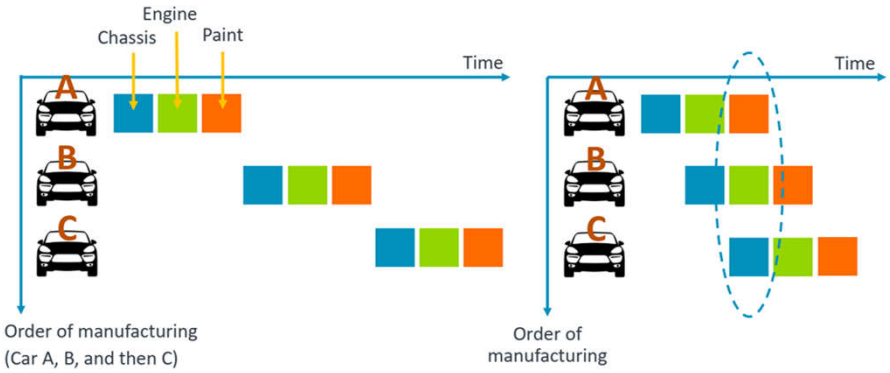
$$\text{CPI}_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

Clocks Per Instruction (CPI)

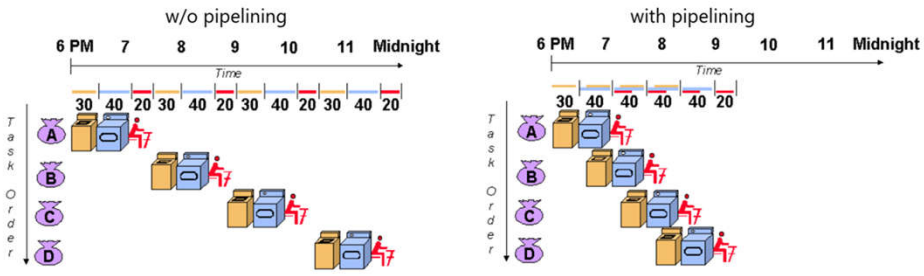
- As transistor budgets improved, we could aim to get closer to a CPI of 1.
- Eventually, the industry was also able to fetch and execute multiple instructions per clock cycle. This reduced CPI to below 1.
- When we fetch and execute multiple instructions together, we often refer to **Instructions Per Cycle (IPC)**, which is 1/CPI.
- For instructions to be executed at the same time, they must be independent.
- Again, growing transistor budgets were exploited to help find and exploit this **Instruction-Level Parallelism (ILP)**.

Parallelism and Pipelining

- Pipelining exploits the potential parallelism among instructions.



Pipelining and Parallelism (Another example)

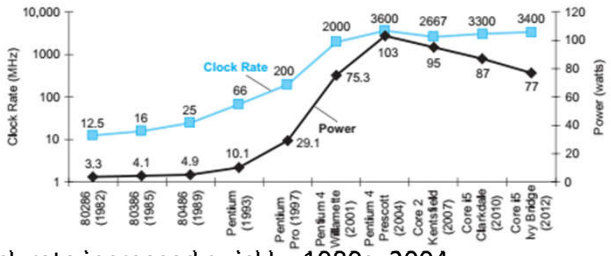


https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/pipelining/index.html#:~:text=We%20could%20put%20the%20the,the%20third%20and%20fourth%20loads.

Reaching Limits of Performance Gains

- Slowing Single core Performance Gains
- The limits of pipelining
- The limits of Instruction-Level Parallelism (ILP)
- The performance of on-chip wiring: wire-delays -> Logic delay
- Power consumption

Power Consumption Trend



- Clock rate increased quickly: 1980s- 2004
- Then slowed down! Why?
- We hit a "Power Wall"

$$Power \propto 1/2 \times Capacitive\,load \times Voltage^2 \times Frequency\,switched$$

Example (Relative Power)

Suppose we developed a new, simpler processor that has 85% of the capacitive load of the more complex older processor. Further, assume that it can adjust voltage so that it can reduce voltage 15% compared to processor B, which results in a 15% shrink in frequency. What is the impact on dynamic power?

- Answer

$$\frac{\text{Power}_{\text{new}}}{\text{Power}_{\text{old}}} = \frac{(\text{Capacitive load} \times 0.85) \times (\text{Voltage} \times 0.85)^2 \times (\text{Frequency switched} \times 0.85)}{\text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}}$$

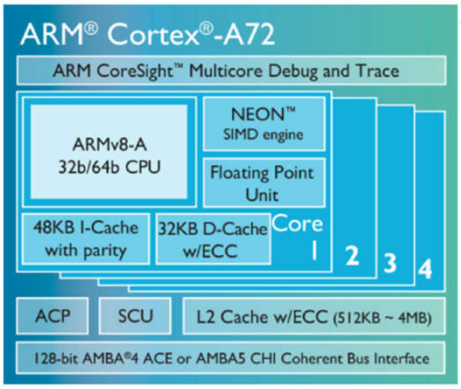
Thus the power ratio is

$$0.85^4 = 0.52$$

Hence, the new processor uses about half the power of the old processor.

Multicore Processors

- Eventually, it made sense to shift from single-core to multicore designs.
- From ~2005, multicore designs became mainstream.
- The number of cores on a single chip increased over time.
- Individual cores were designed to be as power efficient as possible.



e.g., 4 x Arm Cortex-A72 processors, each with their own L1 caches and a shared L2 cache

Multicore Processors

Exploiting multiple cores comes with its own set of challenges and limitations:

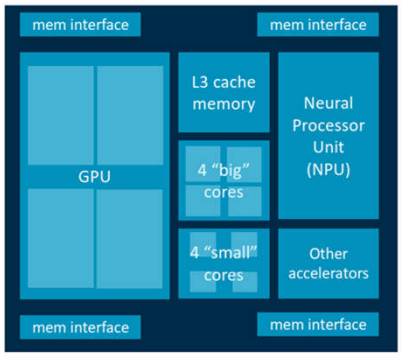
- Power consumption may still limit performance.
- We need to write scalable and correct parallel programs to exploit them.
- We might not be able to find enough parallel threads to take advantage of our cores.
- On-chip and off-chip communication will limit performance gains.
- Off-chip bandwidth is limited and may throttle our many cores.
- Cores also need to communicate to maintain a coherent view of memory

Further Specialization (Accelerators)

- Today, we often need to look beyond general-purpose programmable processors to meet our design goals.
- We trade flexibility for efficiency.
- We remove the ability to run all programs and design for a narrow workload, perhaps even a single algorithm.
- These "accelerators" can be 10-1000x better than a general-purpose solution in terms of power and performance.

Recent SoC Designs

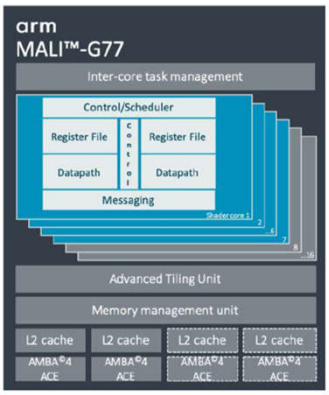
- A modern mobile phone SoC (2019) may contain more than 7 billion transistors.
- It will integrate:
  - Multiple processor cores
  - A GPU
  - A large number of specialized accelerators
  - Large amounts of on-chip memory
  - High bandwidth interfaces to off-chip memory



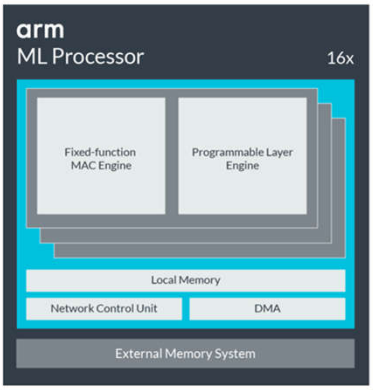
A high-level block diagram of a mobile phone SoC

 arm

Processors for specific applications



Graphics Processing Unit (GPU)



Neural Processor Unit (NPU)

Limits to specialization

- There are costs associated with designing each new accelerator.
- The chip, or "ASIC," produced may only be competitive in a smaller target market, reducing profitability.
- Specialization reduces flexibility.
- The logic invested in specialized accelerators is no longer general-purpose.
- Algorithm changes may render specialized hardware obsolete.
- Once we've specialized, further gains may be difficult to achieve.
- Specialization isn't immune to the concept of diminishing returns.

Trends in Computer Architecture

Time

Approach	Outcome
Early computers	Gains from bit-level parallelism
Pipelining	+ Instruction-level parallelism
Multicore/GPUs	+ Thread-level parallelism/data-level parallelism
Greater integration (large SoCs), heterogeneity, and specialization	+ Accelerator-level parallelism

Note: Memory hierarchy developments have also been significant. The memory hierarchy typically consumes a large fraction of the transistor budget.

The Future – The End of Moore's Law?

- The end of Moore's Law has been predicted many times.
- Scaling has perhaps slowed in recent years, but transistor density continues to improve.
- Eventually, 2D scaling will have to slow down.
- We are ultimately limited by the size of atoms!
- Where next?
- Going 3D - Future designs may take advantage of multiple layers of transistors on a single chip.
- Note: the gains are linear rather than exponential.
- Better packaging and integration technologies (e.g., chip stacking)
- New types of memory (phase-change memory, STT-RAM, etc)
- New materials and devices (nanowire, nanosheet transistors, etc)