Computer Architecture *

Jens Lechtenbörger

IT Systems, Summer Term 2024

This presentation on computer architectures concludes part 1 of IT Systems. It presents the von Neumann architecture and Moore's law as concepts of general importance, before it completes the Hack computer architecture in particular.

1 Introduction

Let us look at the core of our topic, its learning objectives, followed by a recap.

1.1 Today's Core Question

- How do the previously built chips fit together in a computer architecture?
  - Based on Chapter 5 of (Nisan and Schocken 2005)

In this presentation, we solve the puzzle of building a computer by properly arranging all previously built parts.

1.2 Learning Objectives

- Explain von Neumann architecture and discuss its principles, in general and in relation to Hack
- Discuss implications of Moore's law
- Build, test, and analyze Hack computer
  - Trace execution of programs (e.g., program counter, register and memory contents)

After working through this presentation you should be able to discuss the von Neumann architecture and its principles as well as implications of Moore's law.
Besides, you will be able to build, test, and analyze the Hack computer.

1.3 Retrieval Practice

- How to control the ALU?
- What is the Memory Hierarchy?
- What are the major components of the Hack Computer?

*This PDF document is an inferior version of an OER in HTML format; free/libre Org mode source repository.

- How do binary Hack machine instructions look like?

Please take a brief break and write down answers to these questions, without using previous class material.

Agenda

The agenda of this presentation is as follows. After this introduction, we revisit the von Neumann architecture with more details.

Then, you will see Moore's law and its implications and limitations.

Finally, we end our tour of building the Hack computer by using all previously built chips.

2 Von Neumann Architecture

As mentioned already, we usually build computers according to an architecture proposed by John von Neumann.

2.1 Sketch of Von Neumann Architecture

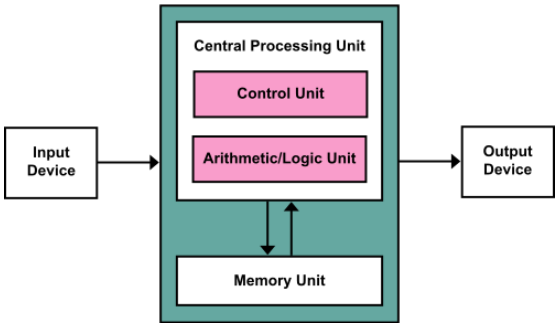

Figure 1: ("von Neumann Architecture" by Kapooht under CC BY-SA 3.0; converted from Wikimedia Commons)

- Design proposed in (von Neumann 1945)

The von Neumann architecture is a computing model that describes a fundamental design of digital computers. It consists of the following main components: The Central Processing Unit with Arithmetic Logic Unit and Control Unit operates on Memory and interacts with Input and Output devices.

- Note: ROM is **not** part of von Neumann architecture
  - Actually, Harvard architecture is variant of Hack with separate memories for data and instructions
  - Details are not important for us

Note that the von Neumann architecture includes just one memory unit for data and instructions, while Hack contains separate units for memory and instructions. Actually, the variant of the von Neumann architecture with separate memories for data and instructions is known as Harvard architecture.

2.2 Von Neumann Principles

- Major principles
  - **Stored program** concept
    - Instructions stored in memory, just as data, modifiable
    - General-purpose, programmable

    The **stored program** concept is probably the most important aspect of the von Neumann architecture, as it distinguishes machines built and programmed for a single purpose from general-purpose, programmable computers.

    In the stored program concept, memory contains data **and** instructions, which makes computers general-purpose and programmable, as programs can be easily modified and executed.

    Without further precautions, instructions can be used as data and also be modified. In general, self-modifying code is harder to read, so we rarely use this feature. However, attackers may exploit this feature to inject their code into vulnerable systems.

  - Single CPU
    - Singe-Instruction Single-Data (SISD) principle
    - (In contrast to parallelism)

    The **single** CPU executes instructions **sequentially** on single pieces of data, following what is known as **Single-Instruction Single-Data** model. Several variants of this model exist that offer parallelism, e.g., for single instructions that operate on multiple pieces of data or the case of multiple instructions operating on multiple pieces of data in parallel.

    Actually, modern processors provide different types of parallelism for improved performance.

  - **Von Neumann Bottleneck**
    - Fast CPU fetches every instruction and data over single bus from slow memory
    - **Memory wall** according to (Wulf and McKee 1995)
      - CPU speeds increase much more than RAM speeds
      - System performance bounded by memory speed
    - Caching and multi-threading as mitigation
      - Threads to be explained in OS part

    The **von Neumann bottleneck** refers to the limitation imposed by the differences in speed of CPU and memory: The fast CPU fetches every instruction and data from slow memory over a single bus. This leads to a **memory wall** phenomenon, a term coined in the paper mentioned here, where the CPU may not be able to operate at full speed as it frequently needs to wait for data from memory. This problem has increased over time as CPU speeds increased at a much faster rate than RAM speeds.

    Mitigations for this bottleneck include techniques like caching and multi-threading. In fact, multi-threading by itself does nothing to improve the speed difference, but it may enable the CPU to work on a different thread of execution while the thread executed so far is stalled by a memory access.

2.3 Fetch-Decode-Execute Cycle

- Use program counter to **fetch** instruction from memory

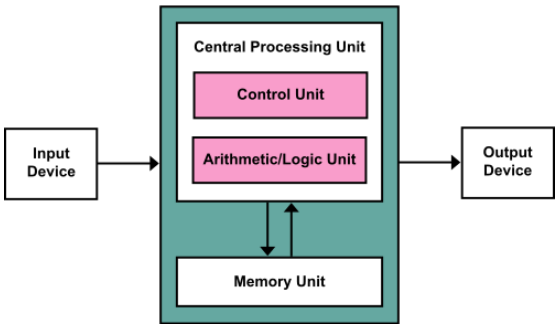

Figure 2: Figure under CC BY-SA 3.0

- Control unit **decodes** bits of instruction to determine operation
- CPU **executes** operation
  - Maybe load data
  - Perform (ALU) operation
  - Maybe write result
  - Update program counter

As explained in the context of the Hack architecture already, modern processors execute architecture-specific machine instructions in a sequence of steps: First, a special register of the processor, called program counter, contains the address of the next instruction to execute. Using that address, the processor fetches an instruction from memory. Then, a control unit decodes the bits of the instruction to figure out what to do. Afterwards, the processor may need to load some data, it executes an operation, probably using the ALU, and it may have to write the result of the operation somewhere. Last, it updates the program counter.

3 Moore's Law

- Published in (Moore 1965)

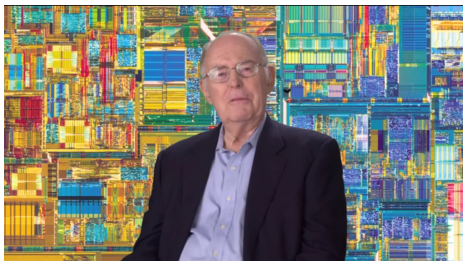

Figure 3: Figure under CC BY-SA 3.0 Deed

  - "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year [...]. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at

least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000.

I believe that such a large circuit can be built on a single wafer."

– Later corrected to "double every two years"

  * **Exponential** improvements

Gordon Moore, a co-founder of Intel, published a prediction in 1965, which should later become famous as Moore's law. You see an essential quote here.

Slightly rephrasing the quote, he predicted that, at the **same cost**, every year about twice as complex chips could be produced. In popular interpretation, chips would be twice as powerful at the same cost every year, thanks to continuing miniaturization.

Later, this prediction was corrected, and used as marketing and engineering goal by Intel, to produce twice as powerful chips every two years.

Note that repeated multiplication by 2 over decades embodies an **exponential** growth, which is probably a unique achievement among technical innovations across domains. Consequently, this law has had profound implications for the computer industry, driving rapid advancements in computing power and miniaturization while reducing costs. In particular, the success of ever more powerful computers turned out to be a driver for innovation across various fields and industries.

3.1 Sample Numbers

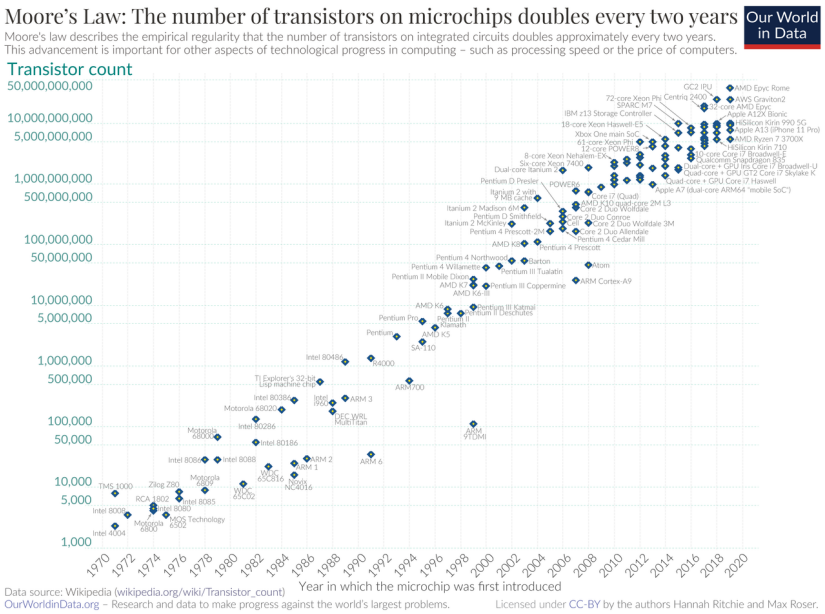

Figure 4: "Moore's Law" by Hannah Ritchie and Max Roser under CC BY 4.0; from Wikimedia Commons

This graph based on Wikipedia data shows the increase of transistor counts following Moore's law in the past decades.

Note that a lot of data points are close to a straight line. Given the logarithmic scale of the y-axis, the graph indicates an exponential increase in the number of transistors over time, up to billions of transistors per chip today.

3.2 Future Perspectives

- Chip's "speed" doubled every two years
  - Due to continued miniaturization
    - Smaller transistors need less power, switch faster
      For decades, progress in miniaturization led to smaller and faster transistors, more of which were integrated into individual chips.
    - Until limits of physics reached
      - Intel Skylake (2015) transistors are around 100 atoms across
      - Leak current, heat, quantum effects
    - To limit energy usage, only parts of a chip are powered-on
      - Powered-off areas referred to as "dark silicon" (Esmaeilzadeh et al. 2011)

      However, at the current nanometer scale of transistors, limits of physics make continued miniaturization unlikely. For example, at this nanoscale, quantum tunneling occurs, where electrons pass barriers that should not be passable under classical mechanics. Such effects degrade the performance of transistors.
      Besides, to keep energy usage under control, increasingly larger fractions of all transistors on a chip need to be powered-off at any point in time. These powered-off parts are referred to as "dark silicon". Clearly, powered-off parts do not contribute to the processor's performance.
  - Options according to (Lundstrom and Alam 2022) (beyond class)
    - 2D nanoelectronics, 3D terascale integration, functional integration

    The paper cited here discusses three potential ways forward in view of Moore's law. Although these topics are beyond our class goals, you may still want to read the short paper: First, classical, two-dimensional, nanoscale fabrication processes for transistors will eventually hit the quantum tunneling limit, preventing further miniaturization.
    Second, the transistor count can be increased by three-dimensional stacking of logic, memory, and power chips, which comes with its own set of challenges.
    Third, deviating from the von Neumann architecture, special-purpose, application-specific processing units may be used much more frequently throughout the computer and its input and output devices. Thus, the load on CPUs would be reduced, as they receive processed information instead of raw data.
    In essence, a further increase in the number of transistors is unlikely to result from further nanoscale miniaturization, but from "terascale electronics" in the form of three-dimensional stacking and added functionality.

3.3 Parallel Programming

- For decades, individual CPUs became more powerful
  - Now, CPUs contain more cores
  - Need parallel programming to make programs faster
    Thanks to Moore's law, we have grown used to ever more powerful, faster CPUs. However, we now see CPUs that contain more and more cores. Basically, you can think of a core as a separate CPU, of about the same speed. Thus, a multi-core CPU essentially embeds several CPUs.
    Importantly, a classical, single-threaded program will only run on a single of these cores, without any benefit in terms of speed.
    To take advantage of multiple cores, parallel programming is necessary.

- OS Topics
  - **Processes** and **threads**
    - Processes are programs in execution
      - Each process can have multiple threads of execution
      - Each core can execute a different thread
      - Needs to be programmed
  - **Concurrency** and **mutual exclusion**
    - **Protect** shared data structures (e.g., via locks)
      - Otherwise, data structures will be corrupted (cf. locking and update anomalies in database systems)
    - Programmers must learn this

In the operating systems part of the course, we will address selected topics related to parallel programming such as processes and threads. Some aspects are mentioned here.

4 Hack Computer

Let us now build the hack computer.

4.1 Hack Input/Output Devices

- Recall: Memory-mapped I/O

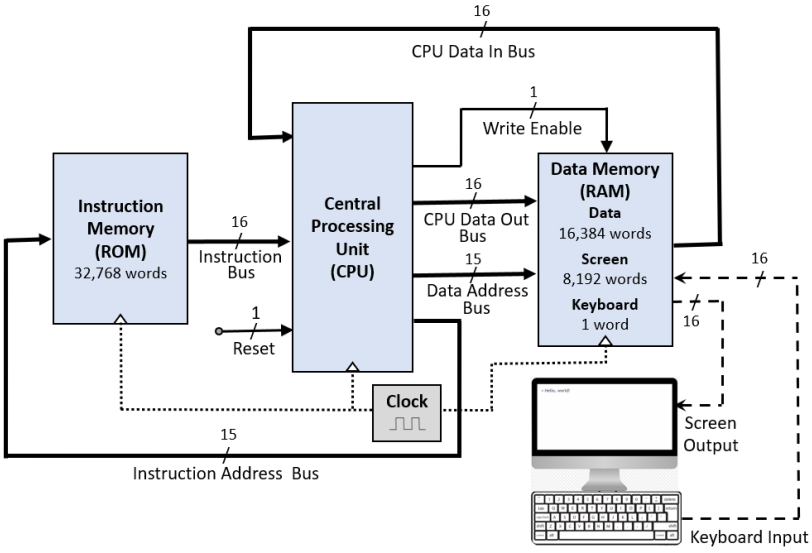

Figure 5: "Hack Computer" by user52174 under CC BY-SA 4.0; from StackExchange

Recall that the Hack computer contains a screen as output device and a keyboard as input device. Both devices are represented with memory maps, which abstract away details of the underlying hardware.

- Screen
  - Built-in chip `Screen`
    - Acts like RAM8K (with `in[16]`, `address[13]`, `load`, `out[16]`)
    - Bit patterns control pixels on screen

  The screen comes as builtin chip that acts just like an appropriately sized RAM chip. Each bit in that RAM area controls one pixel on screen.
- Keyboard
  - Built-in chip `Keyboard`
    - No input, only output `out[16]`, reflects currently pressed key

  The keyboard comes as builtin chip that only produces a single word as output. This word is a code for the currently pressed key, 0 if no key is pressed.

4.2 Hack Data Memory

- Specification in `Memory.hdl`

```
CHIP Memory {
    IN in[16], load, address[15];
    OUT out[16];
Parts:
...
}
```

  - Output `out` with contents of memory location `address`
  - If `load` then store `in` at `address`
  - Different `address` ranges
    - 0-16383: Access to `RAM16K`
    - 16384-24575: Access to `Screen`
    - 24576: Access to `Keyboard`
    - Larger values are invalid

Data memory in the Hack computer consists of usual RAM and of the memory maps for screen and keyboard. The address ranges for the different components are repeated here.

4.3 Hack Instruction Memory

- Recall `ROM32K`

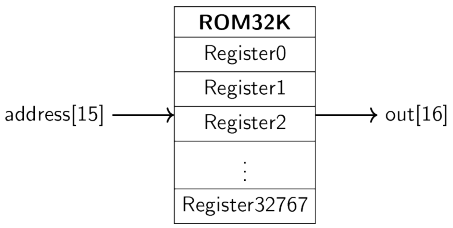

Figure 6: Figure under CC0 1.0

Instruction memory in the Hack computer just consists of a single ROM chip.

4.4 Hack CPU Chip

- Specification via machine language and `CPU.hdl`

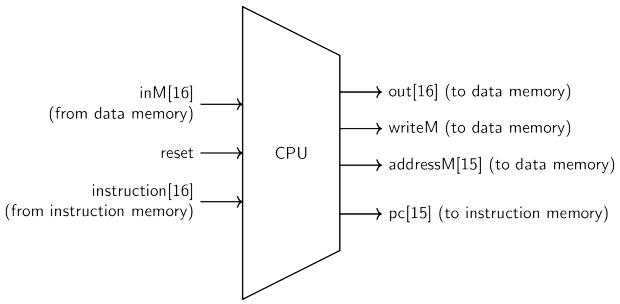

Figure 7: Figure under CC0 1.0

  - Details of machine language include:
    - Three registers: A, D, PC
    - Implicit memory location M (addressed by A register)

```
CHIP CPU {
IN  inM[16],         // M value input  (M = contents of Memory[A])
    instruction[16], // Instruction for execution
    reset;           // Signals whether to re-start the current
     // program (reset=1) or continue executing
     // the current program (reset=0).

OUT outM[16],        // M value output
    writeM,          // Write into M?
    addressM[15],    // Address in data memory (for M)
    pc[15];          // address of next instruction
PARTS: ...}
```

The CPU for the Hack computer is a complex chip whose characteristics are shown here. Importantly, the machine language defines what the CPU is able to do. In particular, it defines the number of available registers and a method to access data memory.

The CPU may receive three pieces of input: First, incoming data from the implicit memory location M. Second, the next instruction to execute from ROM. Third, a reset bit to reset the program counter.

The CPU produces four pieces of output, out of which three define whether and where what should be written to data memory via the implicit memory location M.

In addition, the contents of the program counter define the address for the next instruction in ROM.

We investigate details of the CPU's inner workings next.

4.4.1 ALU with Registers

- Recall C instruction
  - 1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3

(when a=0) comp	c1	c2	c3	c4	c5	c6	(when a=1) comp
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
−1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
−D	0	0	1	1	1	1	
−A	1	1	0	0	1	1	−M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D−1	0	0	1	1	1	0	
A−1	1	1	0	0	1	0	M−1
D+A	0	0	0	0	1	0	D+M
D−A	0	1	0	0	1	1	D−M
A−D	0	0	0	1	1	1	M−D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

Figure 8: Operation Bits of Hack C-Instructions

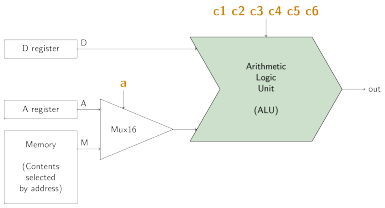

Figure 9: Figure under CC0 1.0

Recall that the ALU is a major component of the CPU and that six control bits determine the ALU's operation. Moreover, C-instructions contain those six bits. An additional seventh bit, the a-bit, controls the operands of the ALU.

As visualized here, the D-register is wired as first input into the ALU.

With the a-bit as selector, a Mux provides the choice between A-register and memory contents as second input into the ALU: If the a-bit is set, M is chosen instead of A.

4.4.2 Proposed CPU Implementation

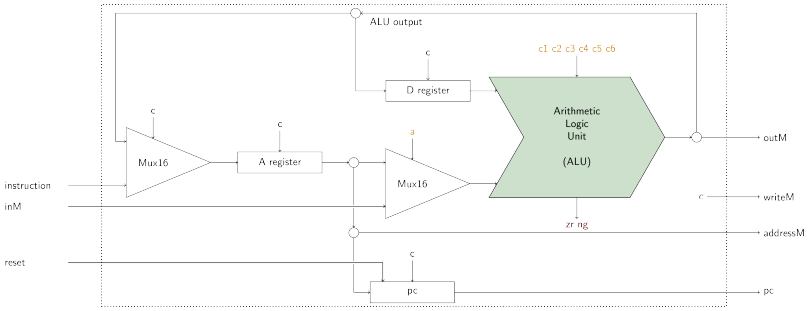

Figure 10: Figure under CC0 1.0

- Incomplete picture
  - Several inputs "c" denote control inputs
    - E.g., load bits for registers, selector for Mux
    - Computed by additional control logic, revisited in class
      - Control unit of von Neumann architecture
      - Decoding of instruction

This picture suggests a possible sketch for the inner workings of the Hack CPU. Note that the parts from the previous slide are embedded here.

In addition, registers and program counter have control inputs, whose values depend on the currently executed instruction. Towards that end, the control unit of the von Neumann architecture decodes the current instruction and routes necessary control information to each component. In the case of Hack, when implementing the CPU, we need to add suitable control logic.

We will discuss some aspects of this control logic in class.

4.5 Hack Computer Chip

- Specification in `Computer.hdl`

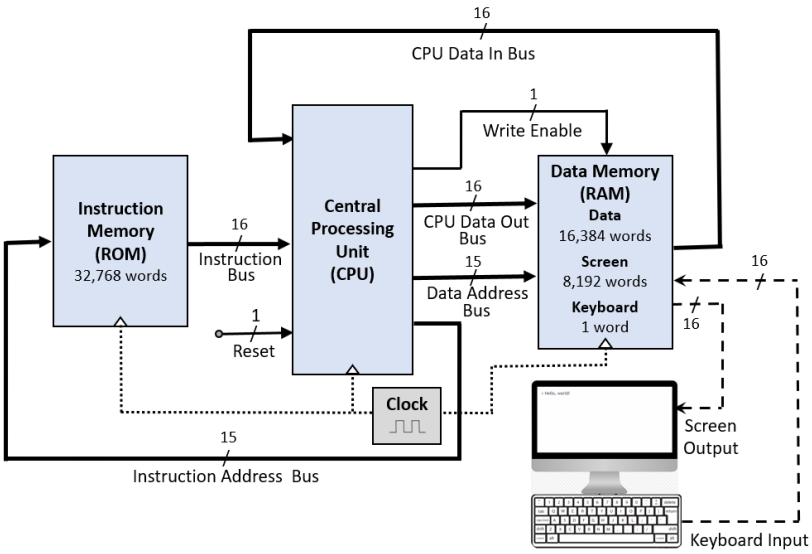

Figure 11: Figure under CC BY-SA 4.0

  - Hack computer, including `CPU`, `Memory` (with `RAM16K`, `Screen`, `Keyboard`), `ROM32K`
  - Input `reset` allows to (re-) start execution of program in ROM

- Build in Project 5

Once `CPU` and `Memory` are built, assembly of the Hack computer is rather straightforward. It just embeds three parts, namely CPU, Memory, and ROM.

5 Conclusions

Let us conclude.

5.1 Summary

- Von Neumann architecture as blueprint for programmable, general-purpose computers
  - Bottleneck from memory access, memory wall
  - Hack as variant with ROM
- Moore's law predicts exponential growth of computer power
  - Physical limits ahead
  - Parallel programming skills required

This presentation contains general ideas related to the von Neumann architecture and to Moore's law as fundamental aspects of modern computers.

- Project 5: Hack computer, end of Part 1 of IT Systems

  - Few components: CPU, data memory, instruction memory
    * CPU executes machine instructions
    * Memory-mapped I/O for screen and keyboard

Building the Hack computer ends part 1 of IT Systems. Now, we are in control of a simple variant of the von Neumann architecture with only three components. Of these, the CPU is certainly the most complex piece. Overall, the Hack computer can execute programs in its own machine language, where keyboard and screen are available as memory-mapped input and output devices.

5.2 Outlook

- In part 2, Operating Systems, we revisit I/O
- Programmed I/O (**polling**) vs **interrupt-driven** I/O
  - You "polled" when accessing the keyboard in Hack
    * Poll: "Has a key been pressed yet?"
      · If not, **wait in loop** → Waste of CPU time
  - Interrupt: Additional input from I/O device to CPU
    * Via pin or bus
      · Notification to CPU: "Hey, someone pressed key X here. Please act."
      · If no key, no unnecessary processing time
      · However, interrupt **overhead**; to be discussed

In part 2 of IT Systems, we discuss Operating Systems. In that context, we also revisit input output processing. We contrast programmed I/O, which is called polling, and interrupt-driven I/O.

When accessing the keyboard in Hack, you engaged in polling, which entails repeatedly asking: Has a key been pressed yet?

If not, the system waits in a loop, resulting in a waste of CPU time.

On the other hand, interrupt-driven I/O involves additional input from the device to the CPU, typically through a pin or bus. The device notifies the CPU, saying: Hey, someone pressed key X here. Please act.

With interrupts, there is no unnecessary processing time if no key is pressed. However, there is an overhead associated with interrupts, which we will discuss later.

5.3 Beyond Class

- There is much more to computer architecture, e.g.:
  - Optimizations
    * Parallelism, e.g., pipelining with speculative execution, hyper-threading, (heterogeneous) multi-core
    * Special-purpose processors, e.g., graphics, machine learning, networking
  - Variety
    * Bus variants
    * Embedded computers
  - Non-traditional architectures

* E.g., quantum, neuromorphic, adiabatic, biological

Computer architecture encompasses much more beyond what has been discussed, in particular various optimizations. These optimizations include parallelism, such as pipelining, hyperthreading, and multi-core processing, as well as speculative execution. Moreover, there are special-purpose processors designed for tasks like graphics rendering, machine learning, and networking.

Additionally, computer architecture presents a wide variety of implementations, including different bus variants and embedded computing systems.

Finally, research and business explore non-traditional computing architectures, for which you find sample hyperlinks on the slide.

5.4 Self-Study-Tasks

- Taking all other parts for granted, implement `Computer.hdl`
  - Part of Project 5
  - (In class, we proceed backwards, with `Memory` and `CPU`)

Please take a break and implement the Hack computer.

Also think about the proposed CPU implementation and data memory. Do questions arise that should be discussed in class?

Bibliography

Esmaeilzadeh, Hadi, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. "Dark Silicon and the End of Multicore Scaling." In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 365–76. https://doi.org/10.1145/2000064.2000108.

Lundstrom, Mark S., and Muhammad A. Alam. 2022. "Moore's Law: The Journey Ahead." *Science* 378 (6621): 722–23. https://www.science.org/doi/abs/10.1126/science.ade2191.

Moore, Gordon E. 1965. "Cramming More Components onto Integrated Circuits." *Electronics* 38 (8). https://www.computerhistory.org/collections/catalog/102770822.

Neumann, John von. 1945. "First Draft of a Report on the EDVAC." University of Pennsylvania. https://web.mit.edu/STS.035/www/PDFs/edvac.pdf.

Nisan, Noam, and Shimon Schocken. 2005. *The Elements of Computing Systems: Building a Modern Computer from First Principles*. The MIT Press. https://www.nand2tetris.org/.

Wulf, Wm. A., and Sally A. McKee. 1995. "Hitting the Memory Wall: Implications of the Obvious." *Sigarch Comput. Archit. News* 23 (1): 20–24. https://doi.org/10.1145/216585.216588.

The bibliography contains references used in this presentation.

License Information