Today

- ICQs and Review
- Combinational Logic
- Sequential Logic
- ISA Design

Announcements and Reminders

Reminders:

- Asmt 1 Due Tomorrow Sept 5 at 6pm.
- Submit via gradescope link in Canvas.

Office Hours

See calendar on Canvas

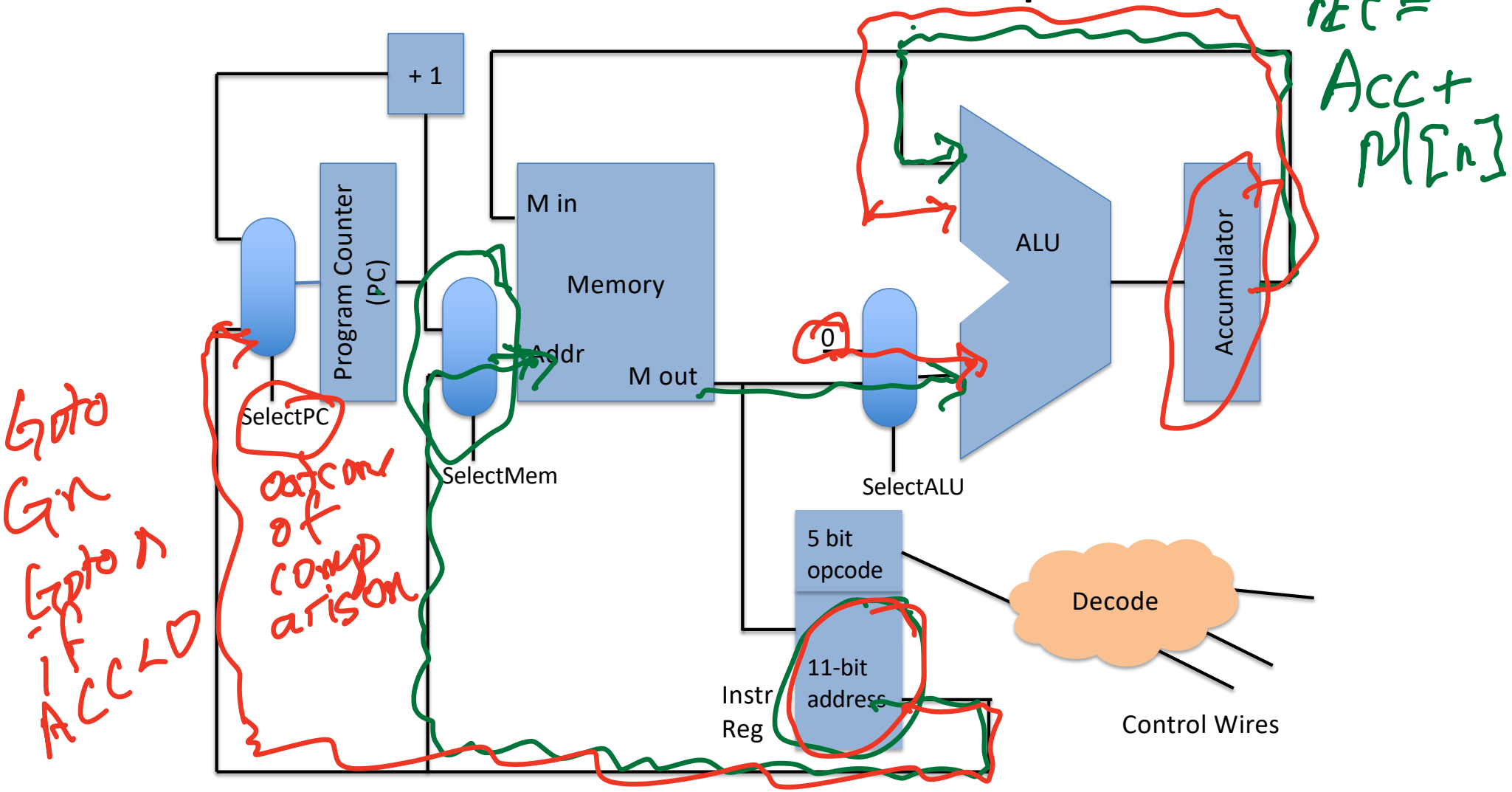
Project 1

- Due Oct 3 at 6pm
- Overview during Tuesday's lecture
- Groups of 2
  - Find partners on Canvas thread
  - OR, we'll facilitate after upcoming lecture

ICQs from last time: Thank you!

- *Several responses of this form:*
  - *"During my time here I have mainly focused on the software side of computing. I am eager to learn more about happens at a lower level and how different implementations can impact performance."*
- *Interest in:*
  - *Hardware in general.*
  - *GPUs, Parallelism, Quantum,*
  - *Getting better at programming and applications*
  - *AI in chips*
  - *Under the hood look at at how a computer work*
  - *Learn about computer architecture, specifically on security since that is a field that I am interested in.*

Last Time: EDSAC Datapath



Programming the EDSAC

- Maurice Wilkes video with subroutine library. Pioneer computers Part 2: 25:00 EDSAC Starts. 27:00: Great film of what it looked like to program… Gordon Bell as narrator.
- https://www.youtube.com/watch?v=wsirYCAocZk

Combinational Logic

Textbook Appendix A

Logic Design

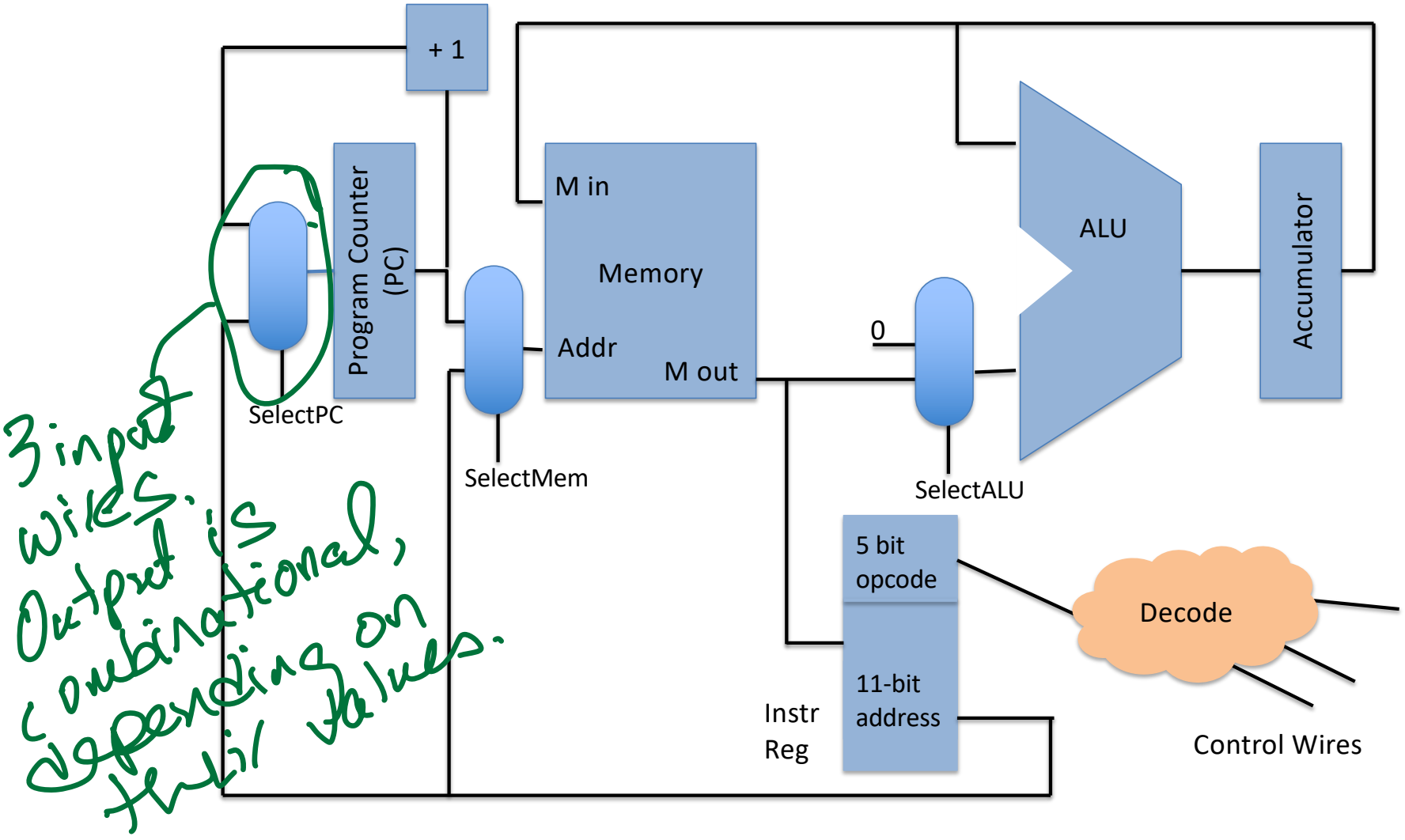
- A ***Combinational Circuit*** is a digital circuit whose outputs depend solely on the present combination of the circuit input's values.
- Examples:
  - Buzzer for when you open the car door with the keys in the ignition

- light switch
- smoke detector

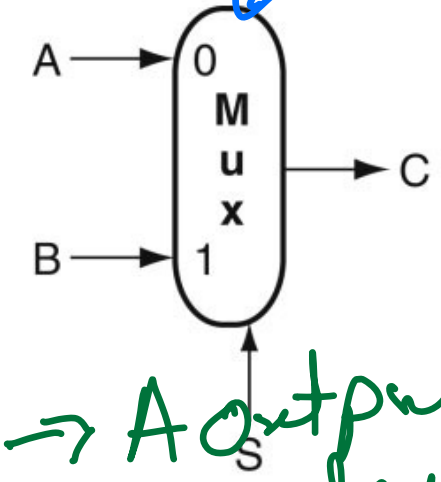
Example: EDSAC ISA from last time…

- Focus on multiplexer that determines where the next PC value comes from. How can we design that? (Depends both on opcode value and also on the 0 coming out of the ALU after comparsion.)

Last Time: EDSAC Datapath



Multiplexer Innards



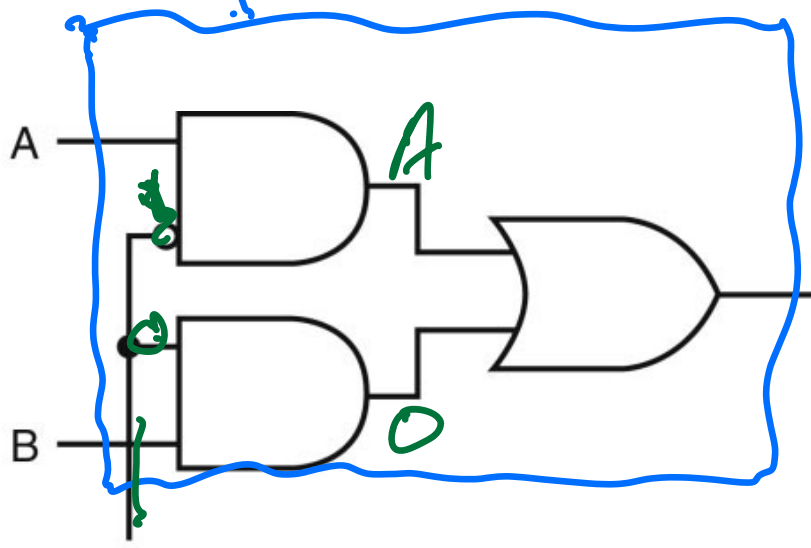
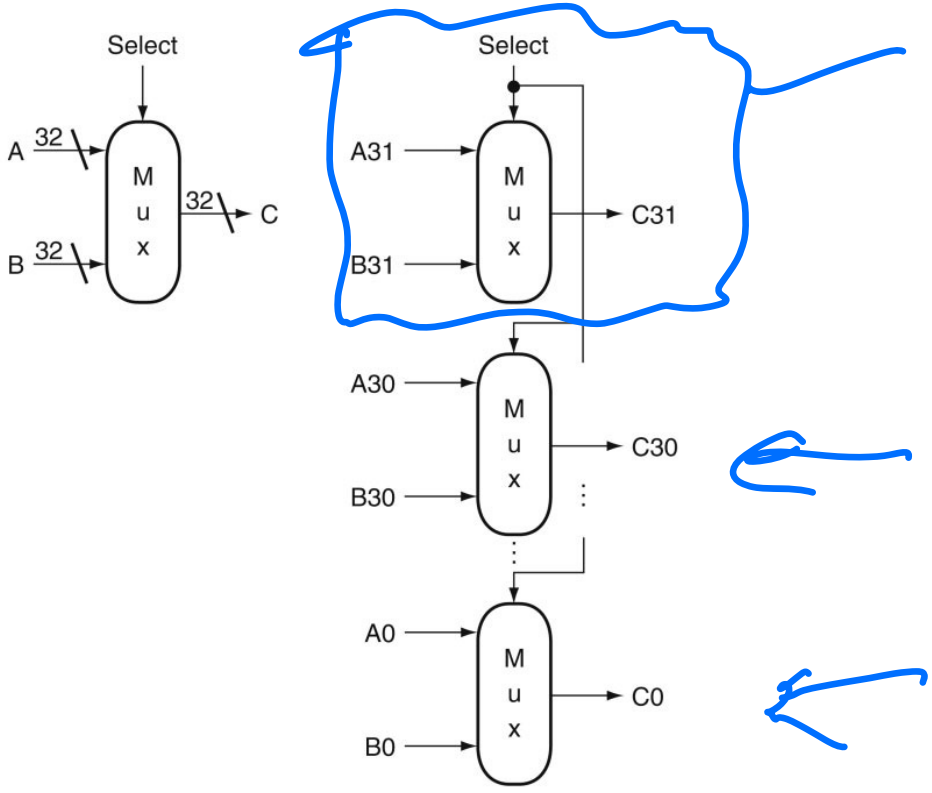


FIGURE B.3.2 A two-input multiplexor on the left and its implementation with gates on the right. The multiplexor has two data inputs (*A* and *B*), which are labeled *0* and *1*, and one selector input (*S*), as well as an output *C*. Implementing multiplexors in Verilog requires a little more work, especially when they are wider than two inputs. We show how to do this beginning on page B-23.

Selecting for multi-bit values



a. A 32-bit wide 2-to-1 multiplexor

b. The 32-bit wide multiplexor is actually an array of 32 1-bit multiplexors

1 bit multiplexor for bit 31

bit 30

bit 0

FIGURE B.3.6 A multiplexor is arrayed 32 times to perform a selection between two 32-bit inputs. Note that there is still only one data selection signal used for all 32 1-bit multiplexors.

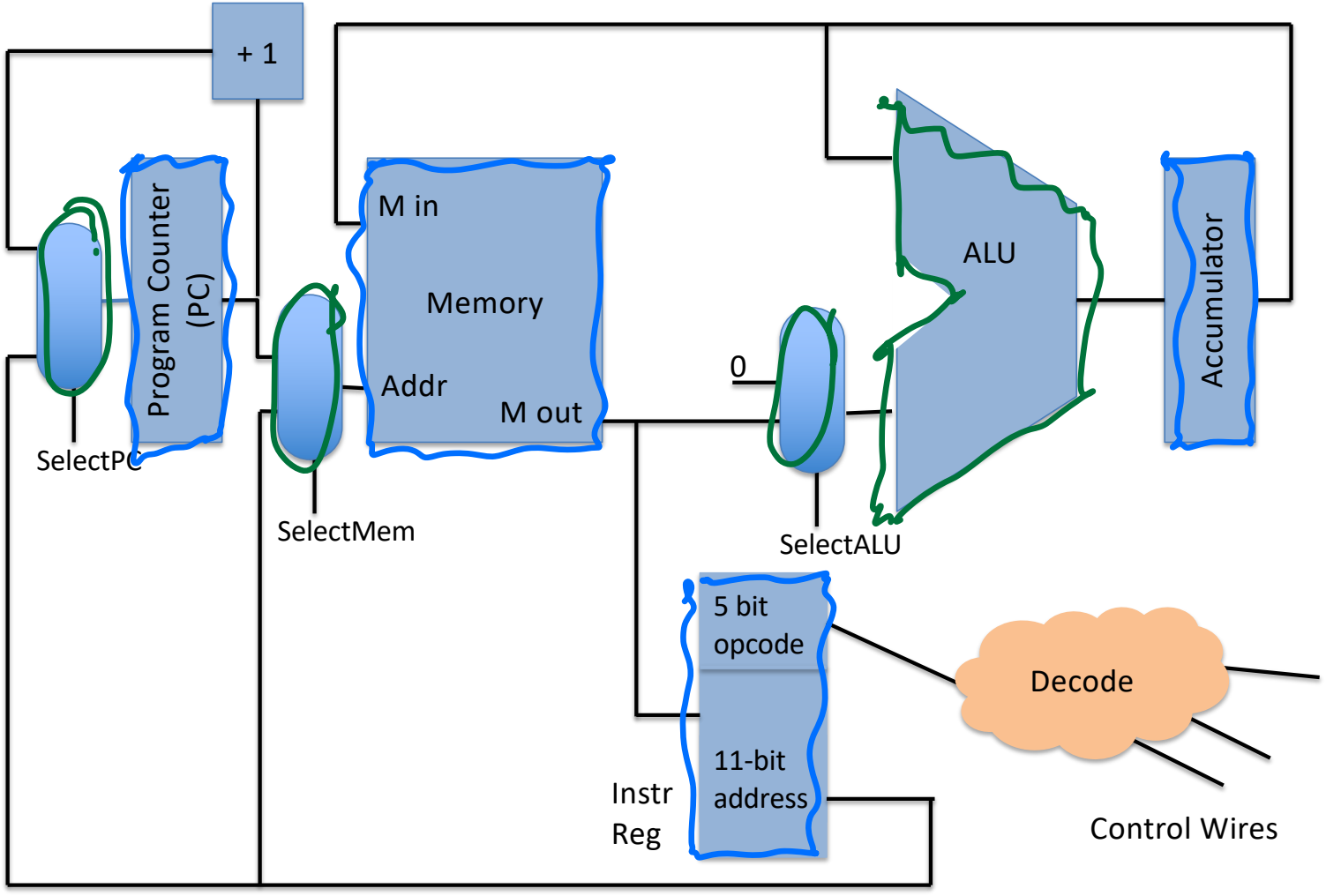
Sequential Circuits and State Storage

Textbook Appendix A

Definitions

- Sequential circuit: Circuit whose Output depends not just on present inputs (as in combinational circuit), but on *past* inputs or on *past sequence* of inputs
- Examples from real life?
  - Pin code door entry
  - Simon - colors in order
  - traffic lights
- State elements: The set of internal storage elements that characterize the current point of execution. If a computer lost power but we were able to restore these elements with the same values they had before, then we could restart execution.

EDSAC Datapath: State Storage or Combinational



Hardware Devices for Holding State

- ***Latch:*** A Memory element in which the output is equal to the value of the stored state inside the element and the state is changed whenever the appropriate inputs change and the clock is asserted.
- ***Flip-flop****:* A memory element for which the output is equal to the value of the stored state inside the element, and for which the internal state is changed ONLY ON A CLOCK EDGE.
- ***D Flip-Flop***: A flip-flop with one data input that stores the value of that input signal in the internal memory when the clock edge occurs.
- ***Register***: A bank of flip-flops, one per bit.

Clock signals

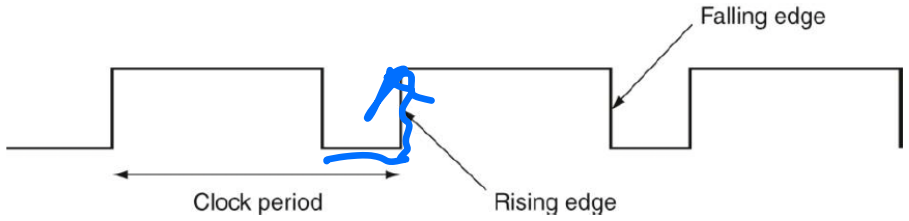


Figure A.7.1 A clock signal oscillates between high and low values. The clock period is the time for one full cycle. In an edge-triggered design, either the rising or falling edge of the clock is active and causes state to be changed.

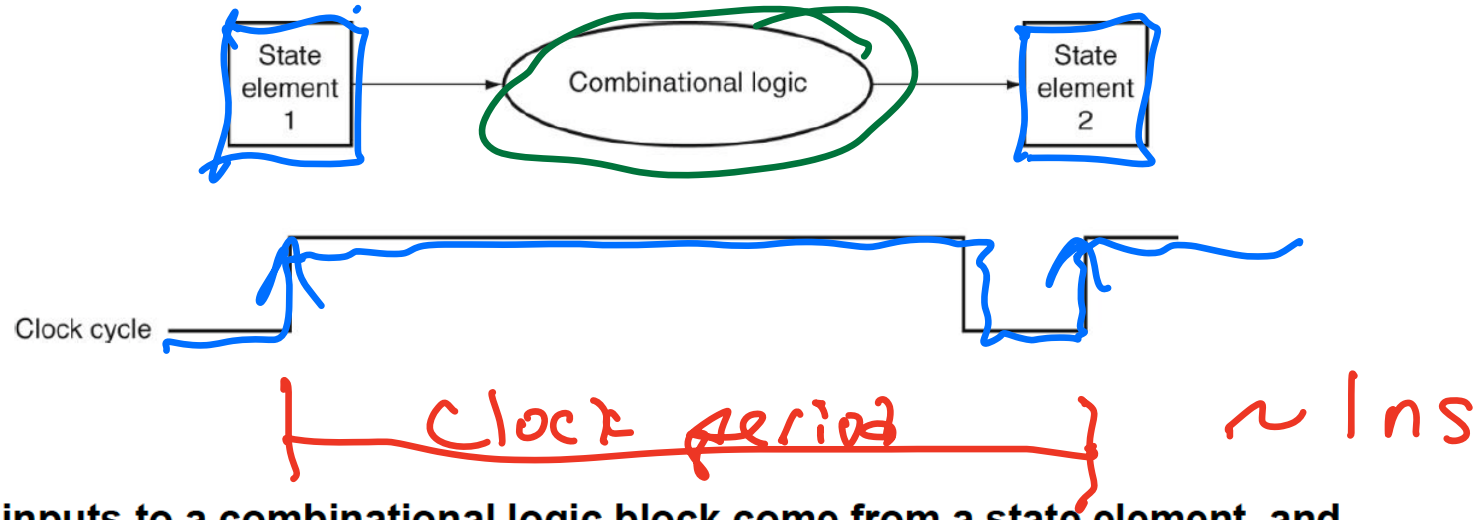
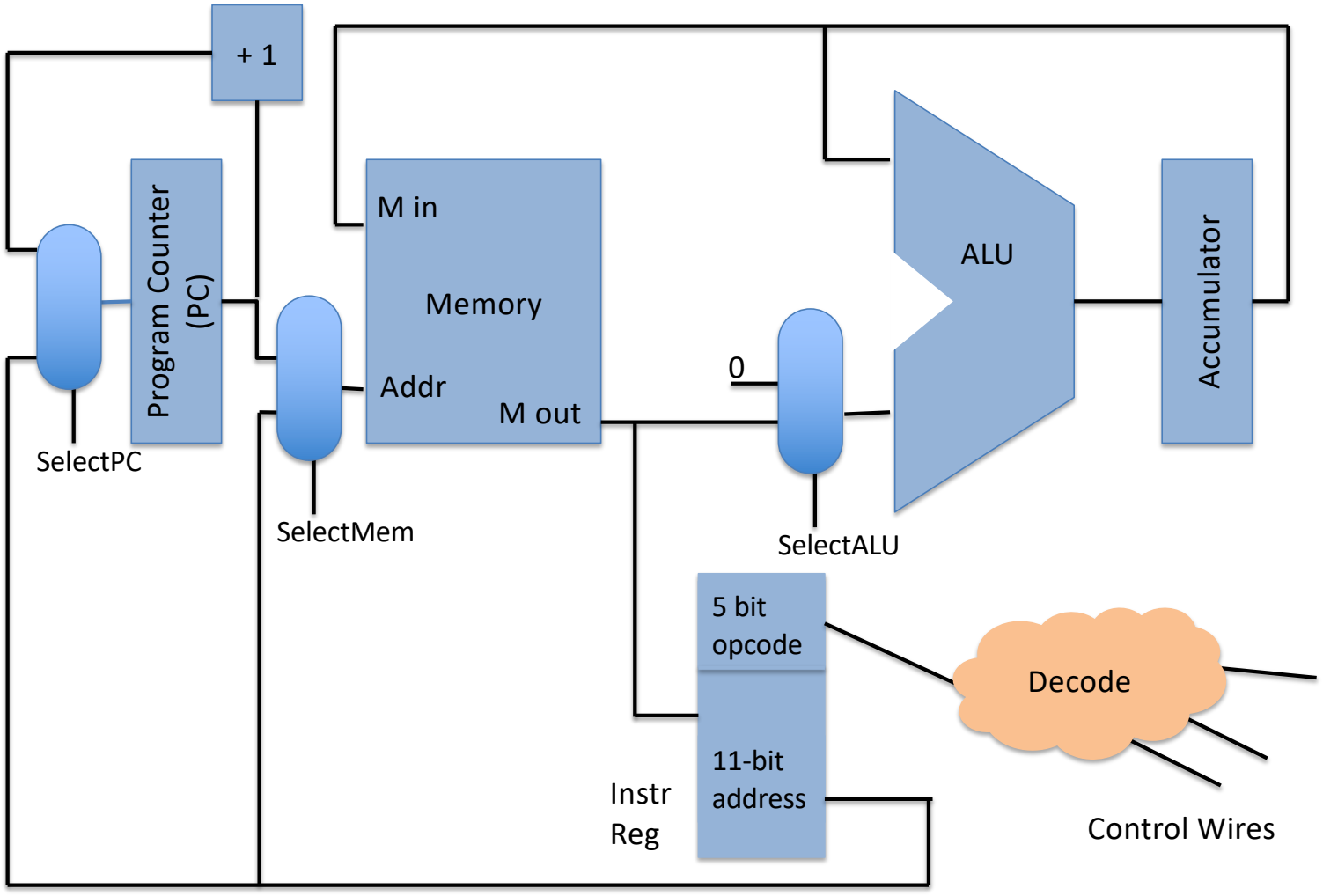


Figure A.7.2 The inputs to a combinational logic block come from a state element, and the outputs are written into a state element. The clock edge determines when the contents of the state elements are updated.

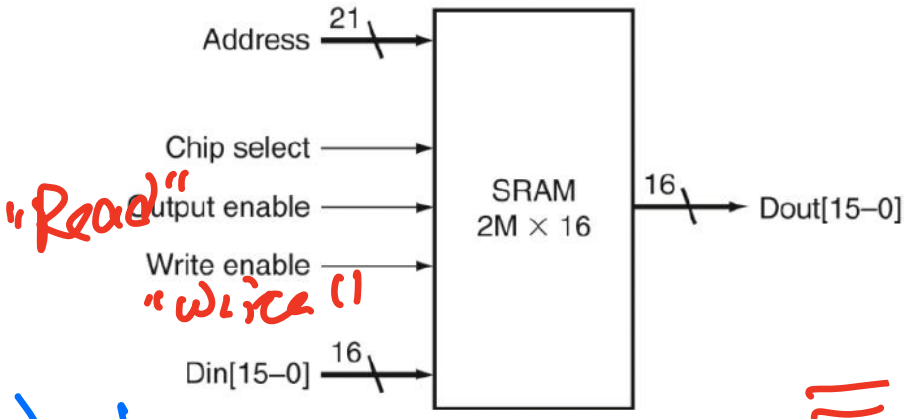
EDSAC Datapath



How does memory work?

Read from Memory:
Addr in
Data out

Write to memory
Addr in + Data In



#entries = 2^{21} entries
2M entries
x 16 bits per entry
= 4MB of memory

Figure A.9.1 A 32K × 8 SRAM showing the 21 address lines (32K = 2^{15}) and 16 data inputs, the three control lines, and the 16 data outputs.

Fundamental Steps to Executing a Program…

- Repeat {
  - Fetch an instruction
  - Decode the instruction
  - Execute the instruction

  }

Instructions: Language of the Computer

Chapter 2

Instruction Set

- The repertoire of instructions of a computer
- Different computers have different instruction sets
  - But with many aspects in common
- Early computers had very simple instruction sets
  - Simplified implementation
- Modern computers range from simple to complex instruction sets.
  - VAX : Canonical Complex ISA – even had instructions that executed a loop of memory copies as a single assembly instruction!
- Sometimes starting simple and gathering complexity as they age

Instruction Set Design

- *"We are guided by a consistent and sound principle in judging the merits of any idea. We wish to incorporate into the machine---in the form of circuits---only such logical concepts as are either necessary to have a complete system or highly convenient because of the frequency with which they occur and the influence they exert in the relevant mathematical situations"*
- <From the original Von Neumann Report, 1945>

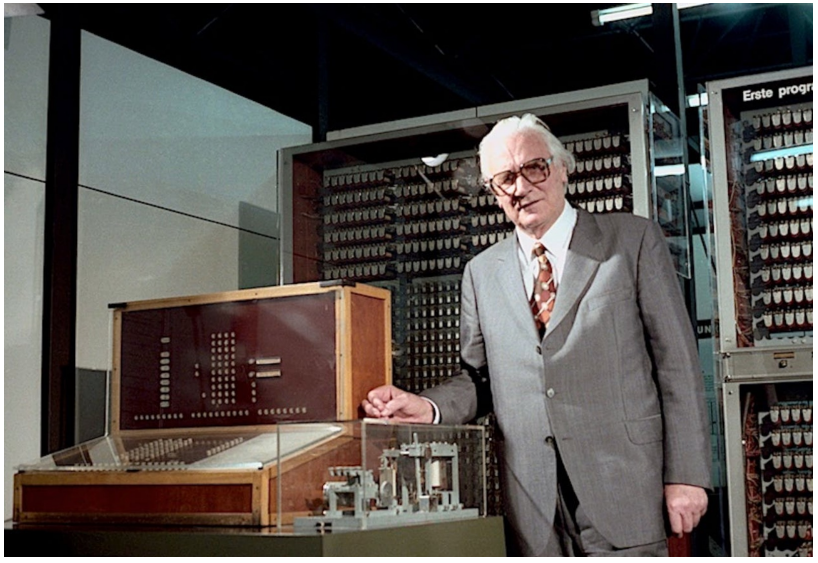
- 50 years later, Patterson & Hennessy: "Make the common case fast…"

ISAs you've heard of? Worked With?

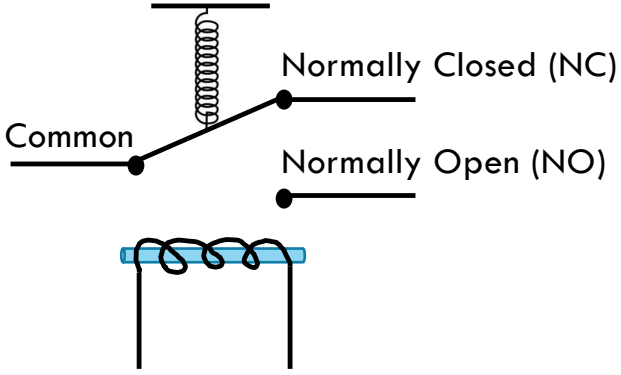
Programmability: Fully automatic digital computer with program stored on punched film. Not Turing-Complete.
Automation: Electronic relays

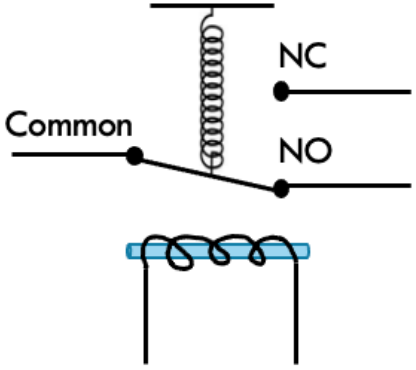
Konrad Zuse's Z3 (1935)

- Konrad Zuse, a German engineer and inventor, made the radical decision to build a fully binary machine [1].
- Used telephone relays and binary representation
- Z3 computer (1935-41) had 2,600 relays, capable of performing floating-point arithmetic.



Previously, internal values were represented in base 10.



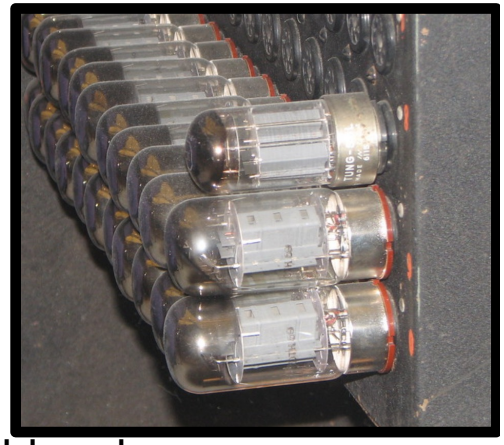


[1] Rojas, Raúl, et al. "The reconstruction of Konrad Zuse's Z3." *IEEE Annals of the History of Computing* 27.3 (2005): 23-32.

Programmability: First programmable, electronic, general-purpose computer.
Automation: Thousands of Vacuum tubes

ENIAC (1946)

ENIAC



- ENIAC: Electronic Numerical Integrator and Computer
- 1st working programmable electronic computer (1946)
- 18,000 vacuum tubes, 7200 crystal diodes, … , and 5 million hand-soldered connections
- Composed of 40 panels
- Team Led by Eckert and Mauchly:

The 18,000 vacuum tube ENIAC proved that electronic technology could be used to make fast, powerful, reliable, general-purpose computers. Its speed of computation was three orders of magnitude greater than the speed obtainable with electromechanical technology, and it solved problems hitherto beyond the

Programmability: First programmable, electronic, general-purpose computer. Now, first electronic stored-program computer.**
Automation: Thousands of Vacuum tubes

ENIAC (May 6, 1949)

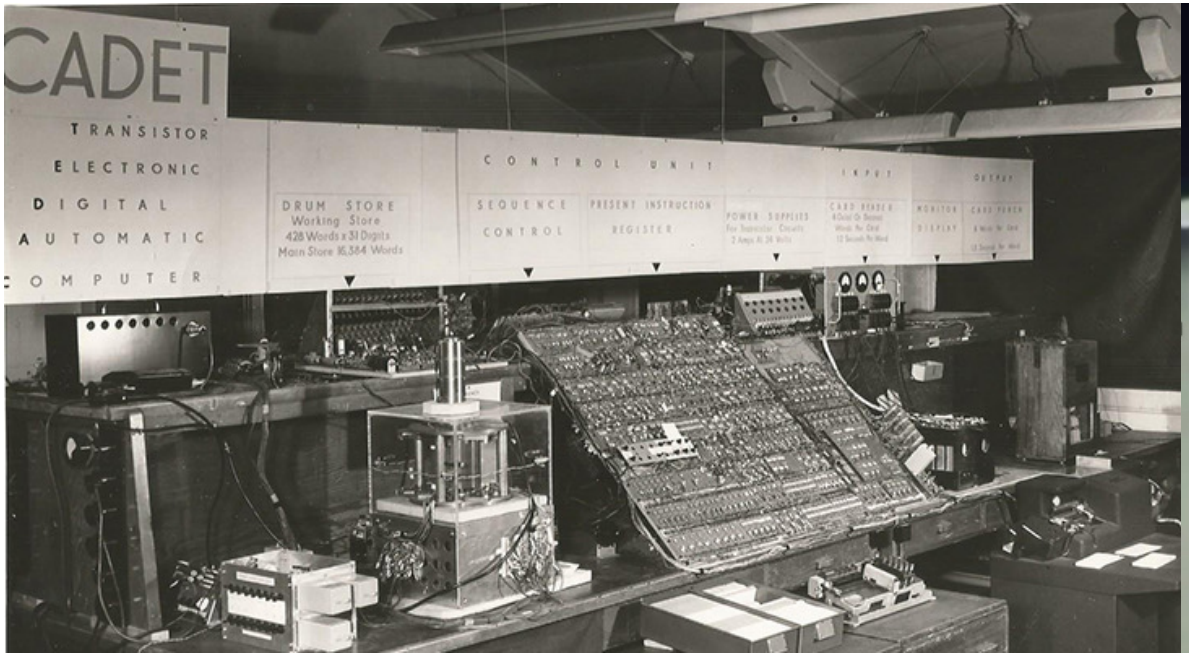
- ENIAC: Electronic Numerical Integrator and Computer
- 1st working programmable electronic computer (1946)
- 1st working stored-program programmable electronic computer (1949)**
- University of Pennsylvania (funded by US Army)

** ALMOST: Manchester Mark I was first Electronic Stored-Program Computer in April 1949.

Programmability: First Stored-program, electronic, general-purpose transistorized computer.
Automation: Thousands of TRANSISTORS!

Harwell CADET (1953)

- The first fully transistorized computer (probably)

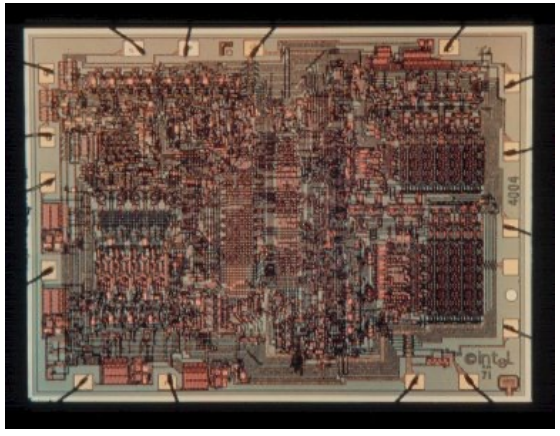


https://youtu.be/V9xUQWo4vN0

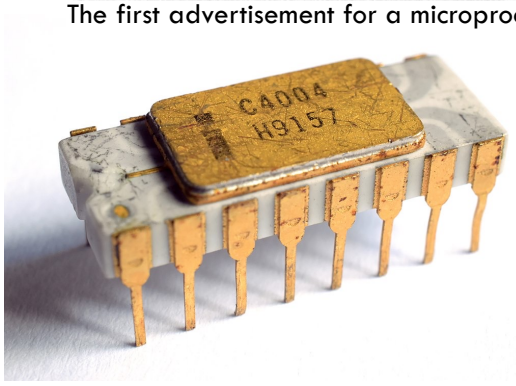
Programmability: First microprocessor (stored-program, electronic, general-purpose) (Architecture: Not x86.)
Automation: Thousands of transistors, one chip!

Intel 4004 (1971)

- First **microprocessor**
- 2,250 transistors, 108Khz, 12 mm^2
- 2 designers



The first advertisement for a microprocessor in Electronic News in November 1971. [1]



COS 126: TOY

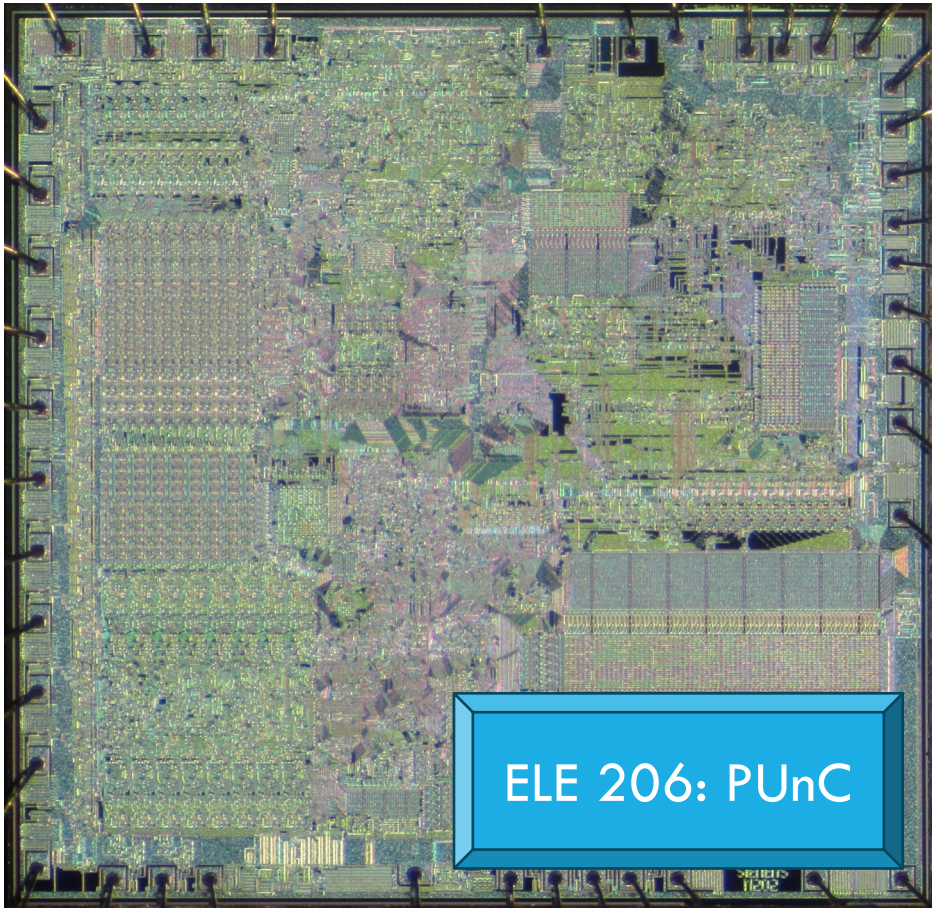
[1] Noyce, Robert N., and Marcian E. Hoff. "A history of microprocessor development at Intel." IEEE Micro 1.1 (1981): 8-21.

Programmability: Really the first microprocessor (stored-program, electronic, general-purpose) (Architecture: x86)
Automation: Tens of Thousands of transistors, one chip!

Intel 8086 (1979)

- Introduced the **x86 architecture** used today
- 29,000 transistors
- 33 mm^2
- 5 MHz
- Original IBM PC based on 8088

We will learn about instruction set architectures (ISAs) in this course.

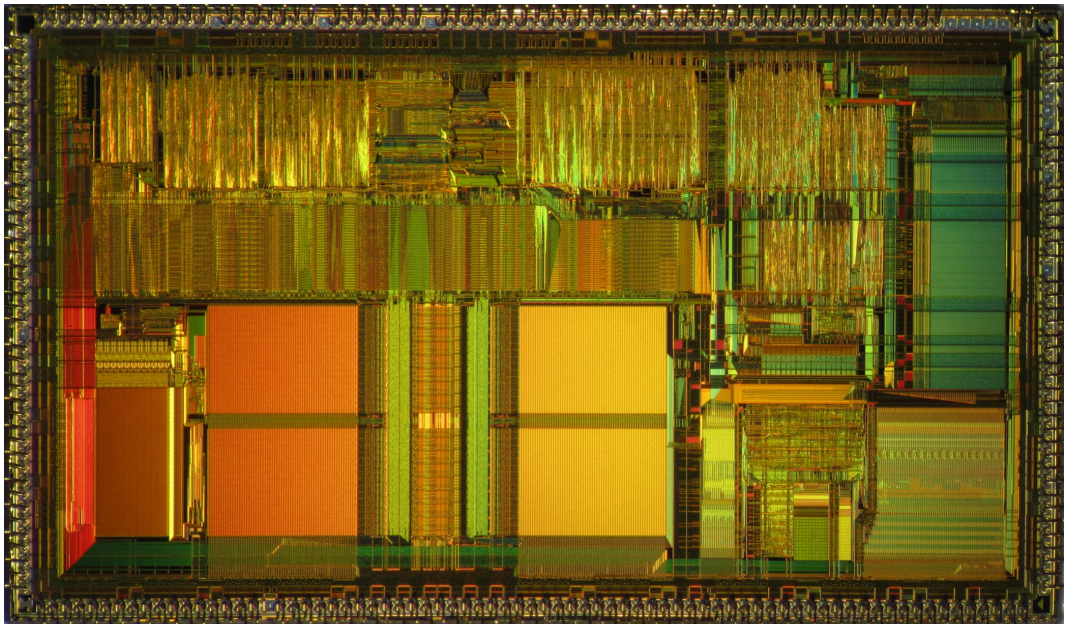


8086 CPU Die

Programmability: Same program binaries as predecessor, just a lot faster! (Architecture: x86 with 32-bit)
Automation: More than one million transistors on one chip!

Intel 80486 (1989)

- First pipelined x86
- First with cache
- 1,200,000 transistors
- 81 mm^2
- 25 MHz



80486 CPU Die

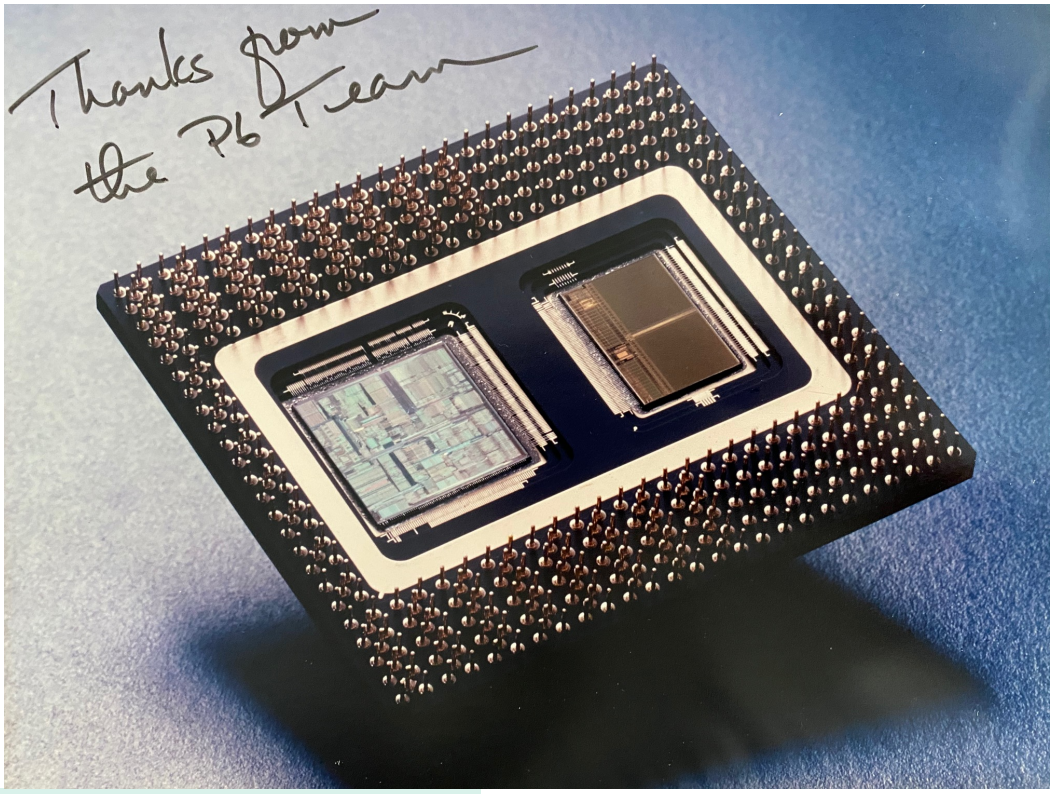
We will learn about pipelining and caching in this course.

COS217's View

Programmability: Same program binaries as predecessor, just a lot faster! (Architecture: still x86)
Automation: Millions of transistors per chip!

Intel Pentium Pro (1995)

- First super-pipelined x86
- First aggressive superscalar
  (First superscalar x86 was Pentium)
- First on-package cache (1MB)
- 7,500,000 transistors
- 200 mm2
- 150-200 MHz



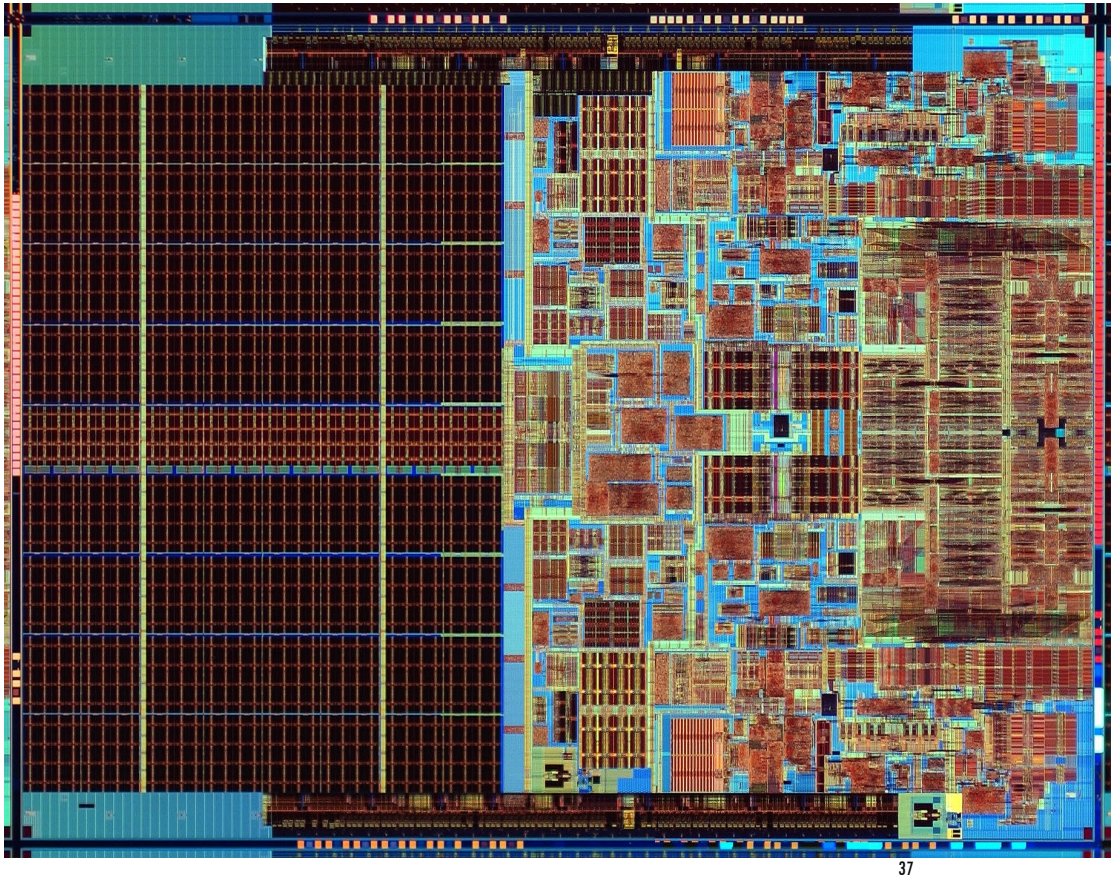
We will learn about caches, super-pipelining, and superscalar in this course.

Programmability: Same program binaries as predecessor, just a little faster! (Architecture: x86-64 with multicore)
Automation: Hundreds of Millions of transistors per chip!

Intel Core 2 Duo (2006)

- Early multicore processor (2 cores)
- 239,000,000 transistors
- 143 mm2
- 1.6 GHz – 3.16 GHz

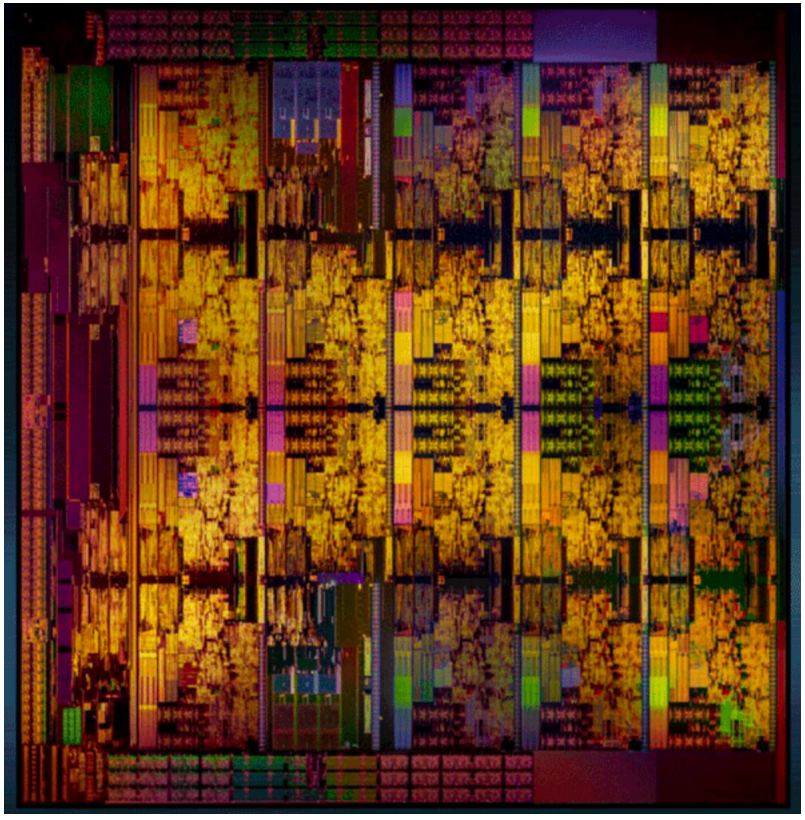
We will learn about multicore in this course.



Programmability: Same program binaries as predecessor, just a little faster! (Architcture: Same x86, but more cores)
Automation: Billions of transistors per chip!

Intel Core i9 i9-10900X X-series (2020)

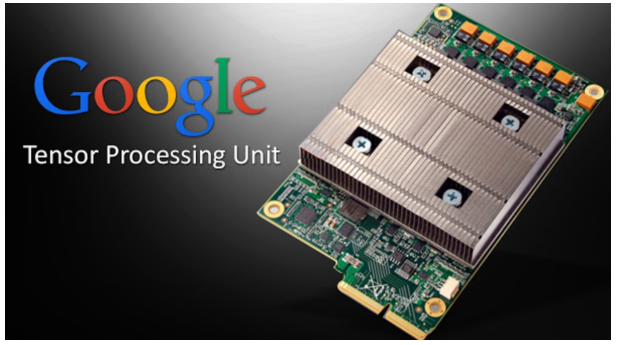
- Modern multicore processor (10 cores)
- 3,500,000,000 transistors (estimated)
- 3.7 GHz – 4.5 GHz



We will learn about the "multicore-problem" in this course.

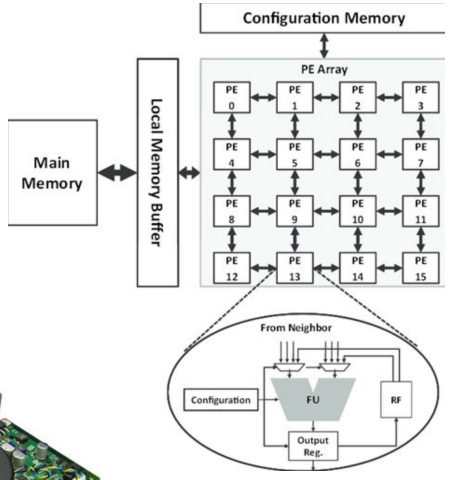
Programmability: Generally faster (not always). Must rewrite your source code. Often limited/not general purpose.
Automation: Many transistors per chip.

Special Purpose/Accelerators



Examples:

- TPU (Tensor Processing Unit): AI Accelerator ASIC for neural network ML.
- FPGA (Field-Programmable Gate Array): Configure lumps of logic
- CGRA (Coarse Grain Reconfigurable Architecture): Configure processing elements and state
- GPGPU (General-Purpose Computing on Graphics Processing Units): Utilize the parallel nature of graphics processing to perform general-purpose computations.



We touch on these and prepare you to learn more.

Instructions: Language of the Computer

Chapter 2

ISA as Contract…

- The **ISA** is the specification for how the machine can be programmed.
- **Assembly code** is the assembly language representation of a program is sort of human readable, but is mostly generated by compilers today.
- **Machine code** is the binary representation of an assembly language program. Machine readable, and largely a 1-to-1 translation of the assembly code.
- The compiler (or the assembly language programmer) MUST FOLLOW THE Given formats. The machine (the processor) only knows how to understand things expressed in that specific format.
- **Beyond the formats**: ISA also specifies other ”rules of the road” such as memory consistency models aka ordering rules for loads and stores.

The RISC-V Instruction Set

- Used as the example throughout the book
- Spearheaded by UC Berkeley as an open-source instruction set to support research and innovation without closed implementation or licensing fees. (eg ARM)
- Now managed by the RISC-V Foundation (riscv.org)
- Although initially created for open-source researcher use, now broadly used by many companies in many different computing implementations: Applications in consumer electronics, network/storage equipment, cameras, printers, …
- Typical of many modern ISAs
  - See RISC-V Reference Data tear-out card (“Green sheet”), and Appendix material

Arithmetic Operations

- Add and subtract, three operands
  - Two sources and one destination

```
add a, b, c   // a gets b + c
```

- All arithmetic operations have this form
- *Design Principle 1:* Simplicity favors regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost

Register Operands

- Arithmetic instructions use register operands

- RISC-V has a 32 × 64-bit register file
  - Use for frequently accessed data
  - 64-bit data is called a “doubleword”
    - 32 x 64-bit general purpose registers x0 to x31
  - 32-bit data is called a “word”

- *Design Principle 2:* Smaller is faster
  - c.f. main memory: millions of locations

RISC-V Registers

- x0: the constant value 0
- x1: return address
- x2: stack pointer
- x3: global pointer
- x4: thread pointer
- x5 – x7, x28 – x31: temporaries
- x8: frame pointer
- x9, x18 – x27: saved registers
- x10 – x11: function arguments/results
- x12 – x17: function arguments

32 total

Register Operand Example

- C code:

```
f = (g + h) - (i + j);
```

  - f, …, j in x19, x20, …, x23

- Compiled RISC-V code:

```
add x5, x20, x21
add x6, x22, x23
sub x19, x5, x6
```

Memory Operands

- Main memory used for composite data
  - Arrays, structures, dynamic data
- To apply arithmetic operations
  - Load values from memory into registers
  - Store result from register to memory
- Memory is byte addressed
  - Each address identifies an 8-bit byte
- RISC-V is Little Endian
  - Least-significant byte at least address of a word
  - *c.f.* Big Endian: most-significant byte at least address
- RISC-V does not require words to be aligned in memory
  - Unlike some other ISAs

Memory Operand Example

- C code:

  ```
  A[12] = h + A[8];
  ```

  – h in x21, base address of A in x22

- Compiled RISC-V code:

  – Index 8 requires offset of 64

    - 8 bytes per doubleword

```
ld   x9, 64(x22)
add  x9, x21, x9
sd   x9, 96(x22)
```

Registers vs. Memory

- Registers are faster to access than memory
- Operating on memory data requires loads and stores
  - More instructions to be executed
- Compiler must use registers for variables as much as possible
  - Only spill to memory for less frequently used variables
  - Register optimization is important!

Immediate Operands

- Constant data specified in an instruction

```
addi x22, x22, 4
```

- Make the common case fast
  - Small constants are common
  - Immediate operand avoids a load instruction

Unsigned Binary Integers

- Given an n-bit number

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: 0 to $+2^n - 1$
- Example
  - $0000\ 0000 \ldots 0000\ 1011_2$
    $= 0 + \ldots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
    $= 0 + \ldots + 8 + 0 + 2 + 1 = 11_{10}$
- Using 64 bits: 0 to +18,446,774,073,709,551,615

2s-Complement Signed Integers

- Given an n-bit number

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: -2^{n-1} to $+2^{n-1} - 1$
- Example
  - $1111\ 1111\ \ldots\ 1111\ 1100_2$
    $= -1 \times 2^{31} + 1 \times 2^{30} + \ldots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$
- Using 64 bits: −9,223,372,036,854,775,808
  to 9,223,372,036,854,775,807

2s-Complement Signed Integers

- Bit 63 is sign bit
  - 1 for negative numbers
  - 0 for non-negative numbers
- $-(-2^{n-1})$ can't be represented
- Non-negative numbers have the same unsigned and 2s-complement representation
- Some specific numbers
  - 0: 0000 0000 … 0000
  - –1: 1111 1111 … 1111
  - Most-negative: 1000 0000 … 0000
  - Most-positive: 0111 1111 … 1111

Signed Negation

- Complement and add 1
  - Complement means 1 → 0, 0 → 1

$$x + \bar{x} = 1111...111_2 = -1$$

$$\bar{x} + 1 = -x$$

- Example: negate +2
  - +2 = 0000 0000 … 0010_{two}
  - –2 = 1111 1111 … 1101_{two} + 1
    = 1111 1111 … 1110_{two}

Sign Extension

- Representing a number using more bits
  - Preserve the numeric value
- Replicate the sign bit to the left
  - c.f. unsigned values: extend with 0s
- Examples: 8-bit to 16-bit
  - +2: 0000 0010 => 0000 0000 0000 0010
  - –2: 1111 1110 => 1111 1111 1111 1110

- In RISC-V instruction set
  - `lb`: sign-extend loaded byte
  - `lbu`: zero-extend loaded byte

Design Variation Exercise

- Suppose you were one of the designers of the original RISC-V instruction set. You have an idea:
- Your idea is to make program code fit into smaller memories by having some of the instructions be encoded into just 16-bits, while most of the rest of the instructions still use 32-bits.

1) Propose a format for these instructions and propose 3 specific new instructions to use it.

2) Be ready to take a side:

For the 16-bit formats or

Against this addition---why might it be a good or bad idea?

Summary and Questions?

Please fill out ICQ Form here!

https://tinyurl.com/Fa25ICQ.

Architecture Credos

- Simplicity favors regularity
- Smaller is faster
- Make the common case fast.
- Good design demands good compromises

Acknowledgements

- Fall, 2025 Course slides include materials from:
  - Margaret Martonosi and David August, Princeton
  - Doug Clark, Princeton (retired)
  - Kelly Shaw, Williams College
  - Carole-Jean Wu, META

Thank you!