

CSE101 · Week 2

Computer Architecture

Prof. Dr. Mehmet Göktürk
Gebze Technical University · Fall 2026

CPU, registers, memory and the stored-program concept

VizMachine instructions and program execution

I/O, buses, contention and DMA

Handshaking, communication and parallel execution

Central Processing Unit (CPU) or processor

Arithmetic/Logic unit versus Control unit

Registers

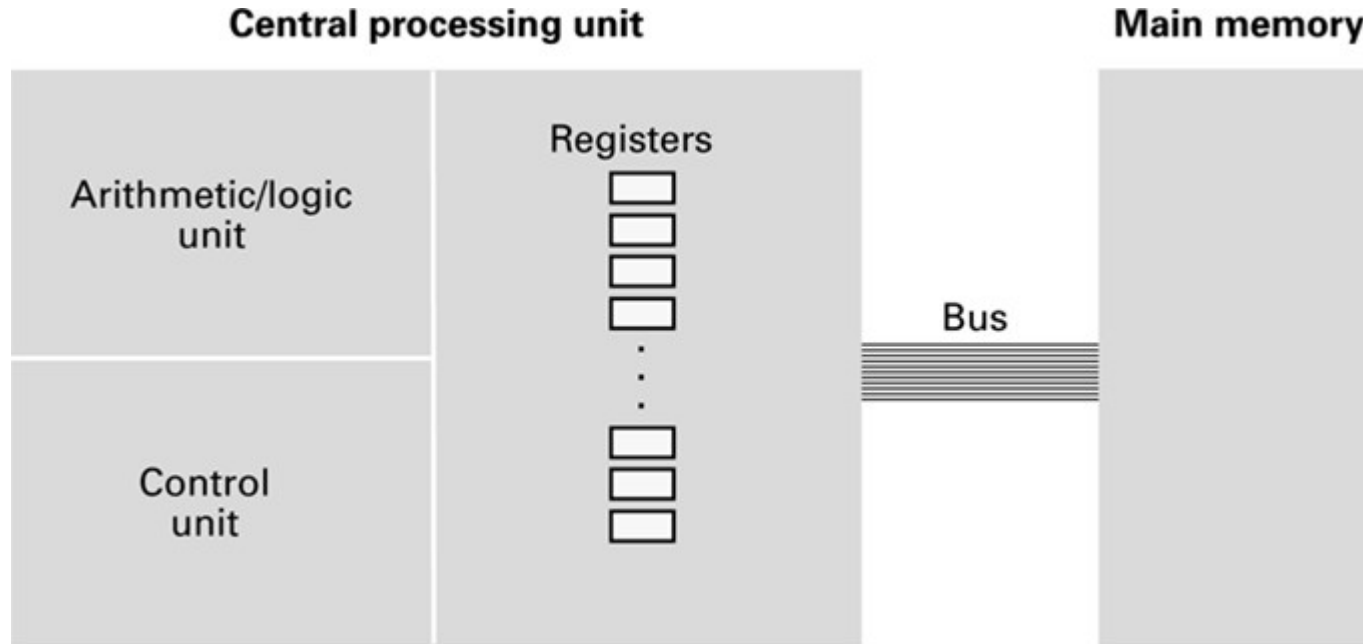
General purpose

Special purpose

Bus

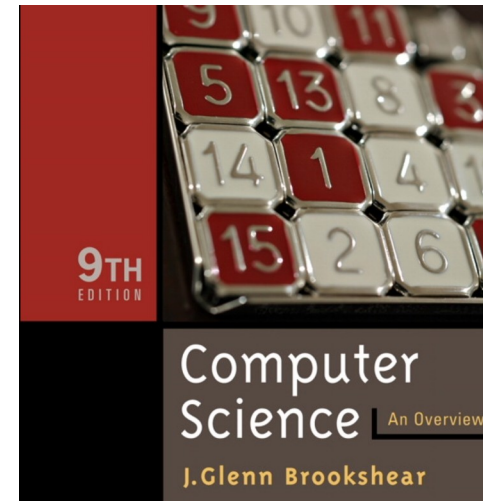
Motherboard

CPU and main memory

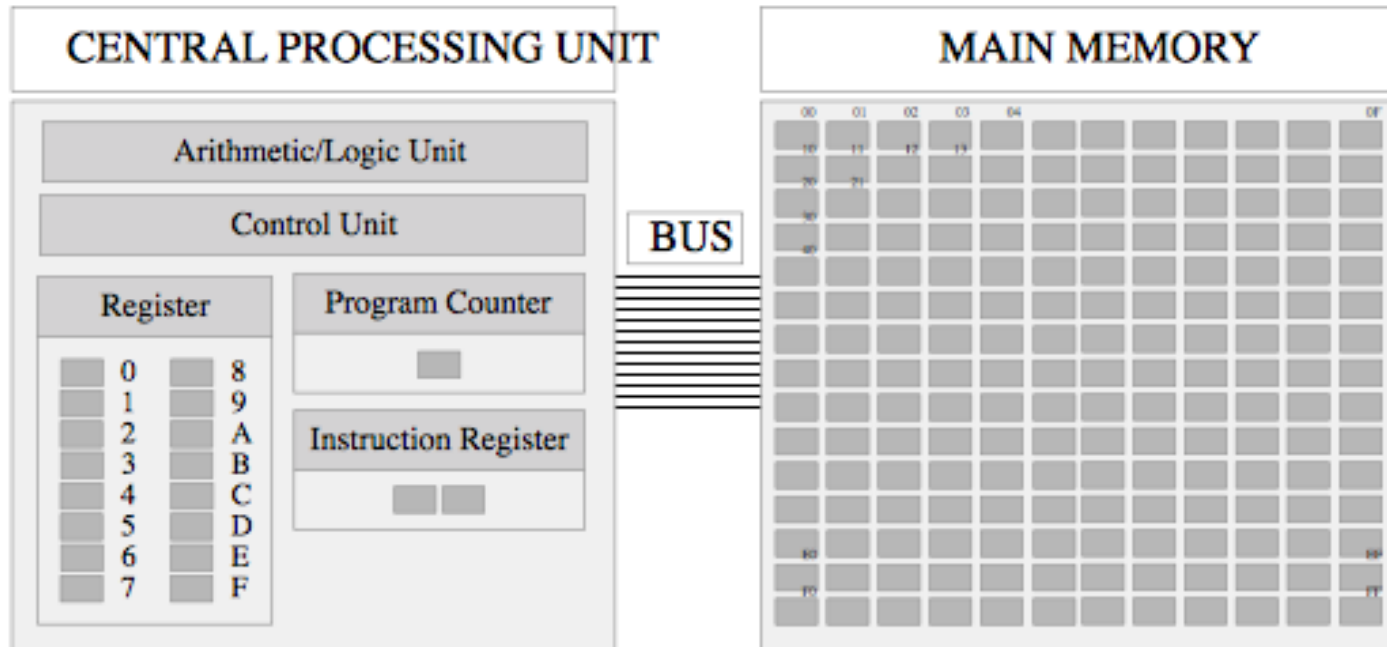


Brookshear Machine

- A small hypothetical computer for learning machine language
- Introduced in Brookshear's textbook
- VizMachine lets us inspect and run this model



Architecture



16 general registers and 256 one-byte memory cells. Each instruction occupies two bytes.

Instructions used in our examples

Encoding	Operation
1RXY	Load R from memory cell XY
2RXY	Load literal XY into R
3RXY	Store R in memory cell XY
5RST	Integer add: R gets $S + T$
7RST / 8RST / 9RST	Bitwise OR / AND / XOR
BRXY	Jump to XY if R equals R0
C000	Halt

R, S and T name registers; XY is one byte written as two hexadecimal digits.

Stored Program Concept

A program can be encoded as bit patterns and stored in main memory. From there, the CPU can then extract the instructions and execute them. In turn, the program to be executed can be altered easily.

Machine instruction: An instruction (or command) encoded as a bit pattern recognizable by the CPU

Machine language: The set of all instructions recognized by a machine

RISC and CISC

RISC designs emphasize regular instructions and register operations.

Arm uses separate load/store instructions for memory access.

CISC designs, such as x86, include more varied instruction forms.

Instruction-set style alone does not determine processor speed.

Machine Instruction Types

Data Transfer: copy data from one location to another

Arithmetic/Logic: use existing bit patterns to compute a new bit pattern

Control: direct the execution of the program

Adding values stored in memory

- Step 1.** Get one of the values to be added from memory and place it in a register.
- Step 2.** Get the other value to be added from memory and place it in another register.
- Step 3.** Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.
- Step 4.** Store the result in memory.
- Step 5.** Stop.

Division as an algorithm

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

Step 3. If this second value is zero, JUMP to Step 6.

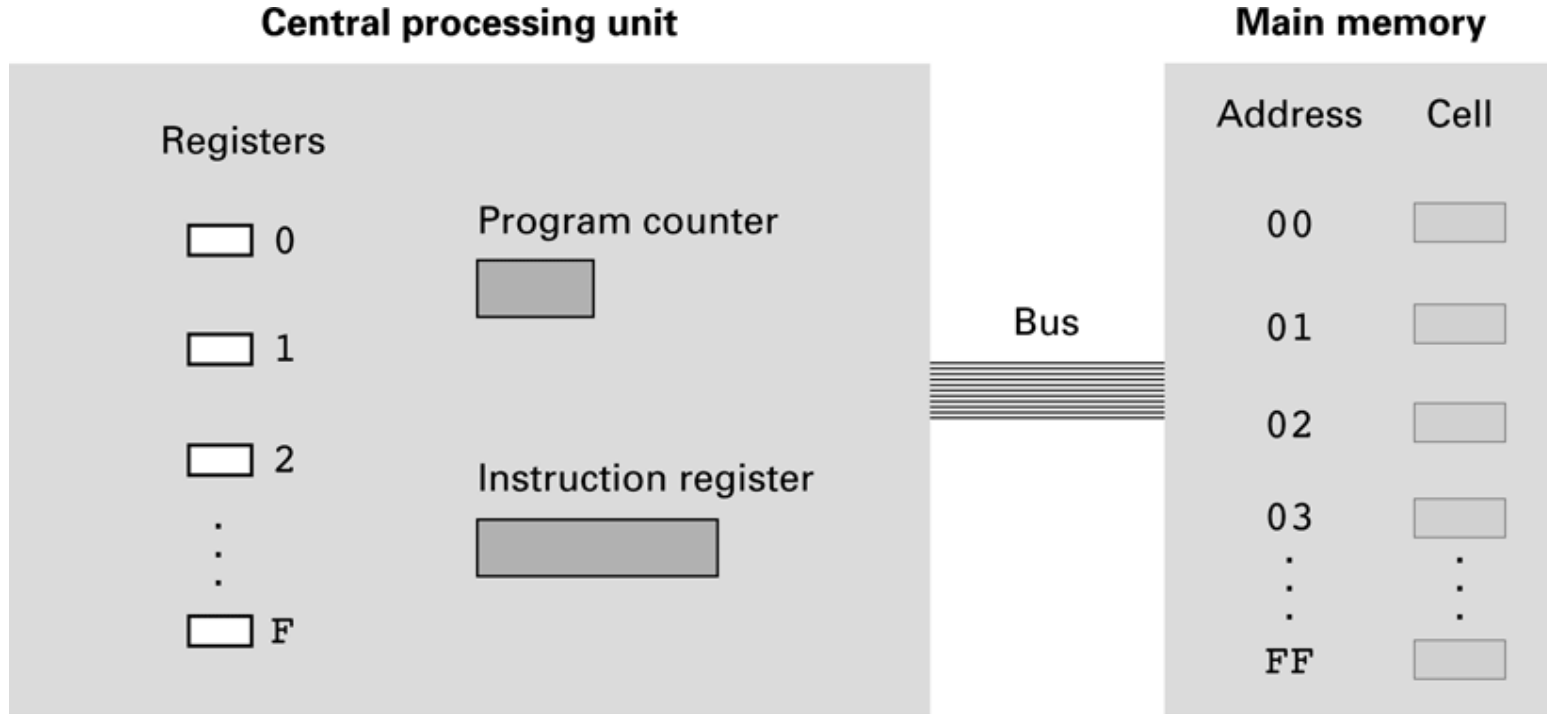
Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

Conceptual algorithm: the basic Brookshear instruction set has no DIVIDE instruction.

The Brookshear machine



Parts of a Machine Instruction

Op-code: Specifies which operation to execute

Operand: Gives more detailed information about the operation

Interpretation of operand varies depending on op-code

VizMachine for this lecture

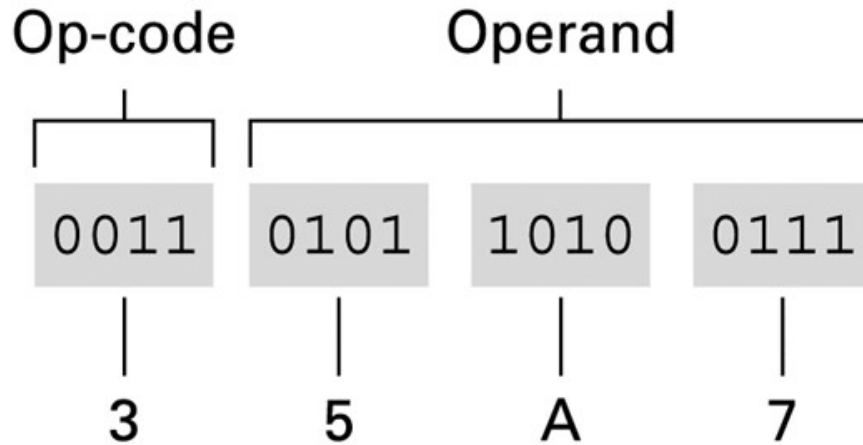
Use the course-provided VizMachine version.

Locate memory, registers, the program counter and instruction register.

Enter hexadecimal instruction bytes and initial data.

Step through the program and compare the machine state with your prediction.

Instruction encoding



Actual bit pattern (16 bits)

Hexadecimal form (4 digits)

Decoding 35A7



Op-code 3 means to store the contents of a register in a memory cell.

This part of the operand identifies the address of the memory cell that is to receive data.

This part of the operand identifies the register whose contents are to be stored.

An encoded addition program

Instruction	Operation
156C	R5 gets M[6C]
166D	R6 gets M[6D]
5056	R0 gets R5 + R6
306E	M[6E] gets R0
C000	Halt

The same encoded program is loaded into memory and traced on the following slides.

Program Execution

Controlled by two special-purpose registers

Program counter: address of next instruction

Instruction register: current instruction

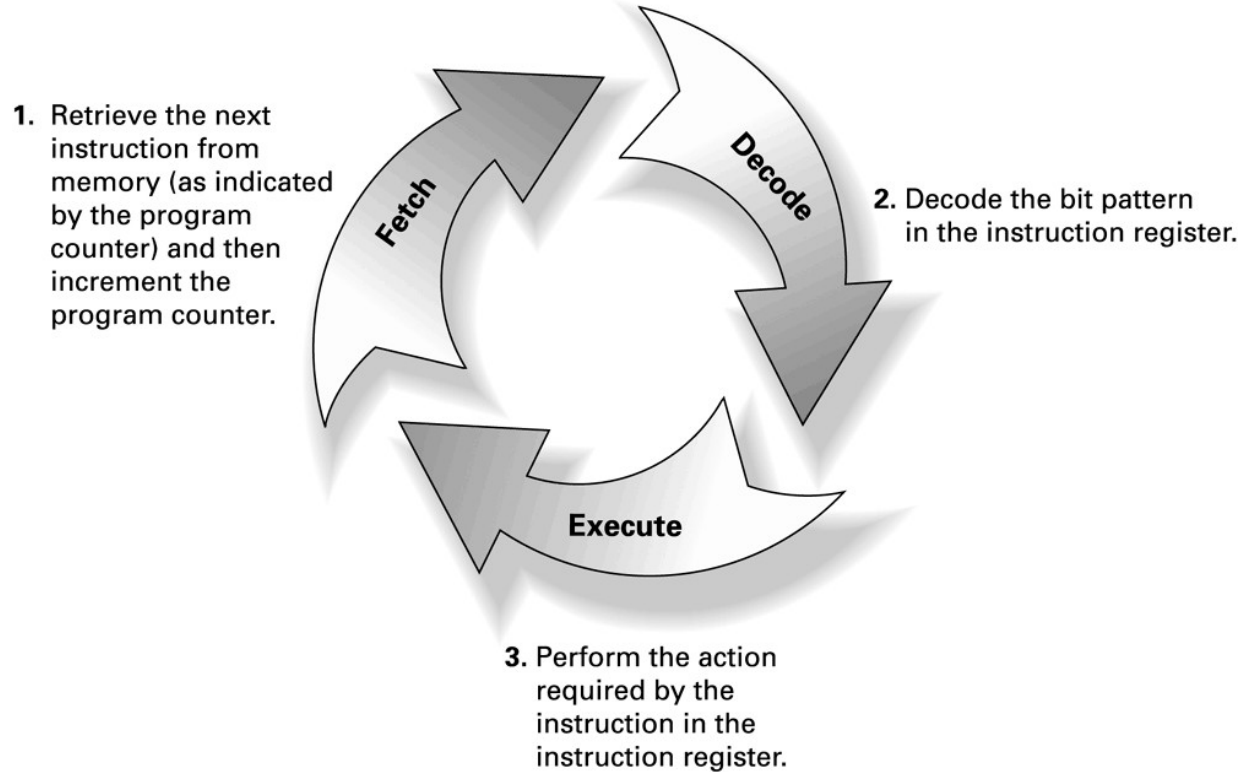
Machine Cycle

Fetch

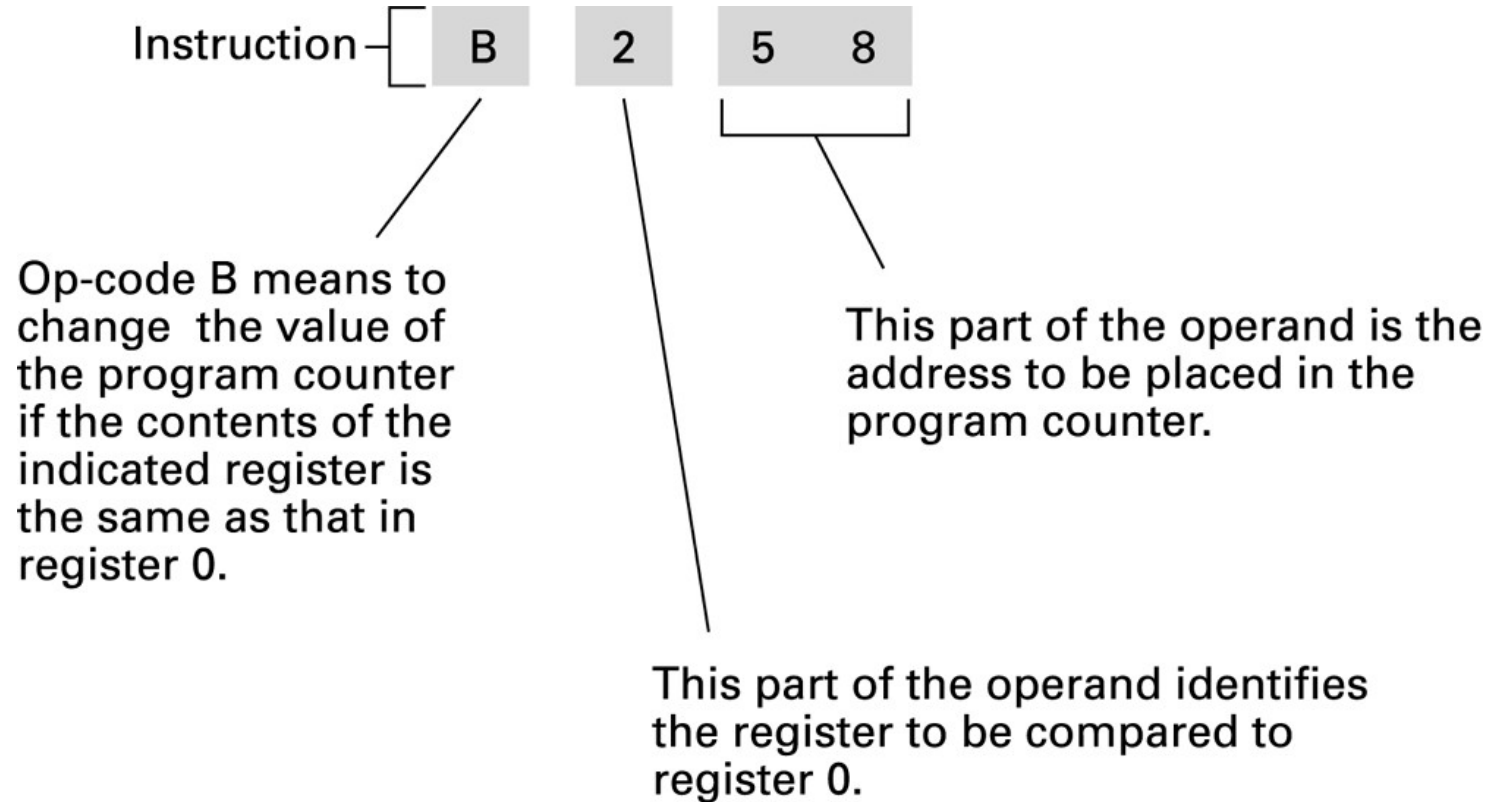
Decode

Execute

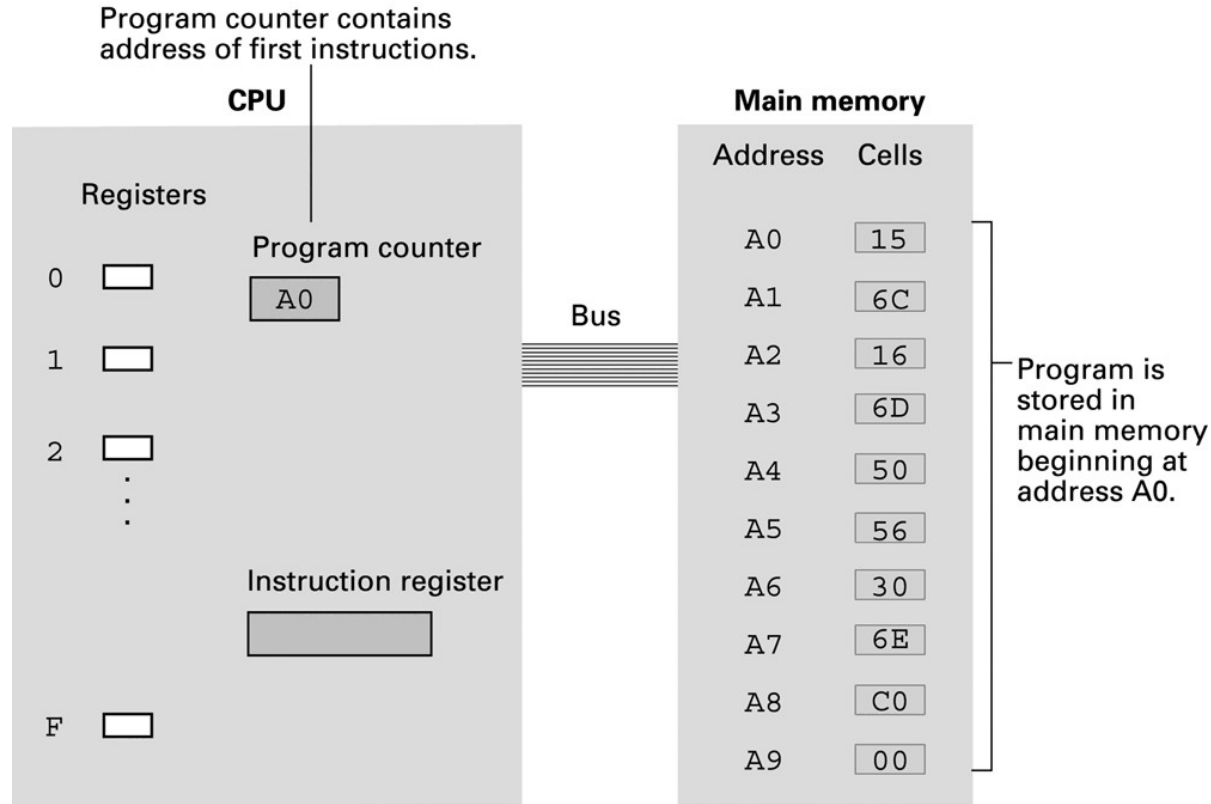
The machine cycle



Conditional branch B258



A program in main memory



Running the addition program

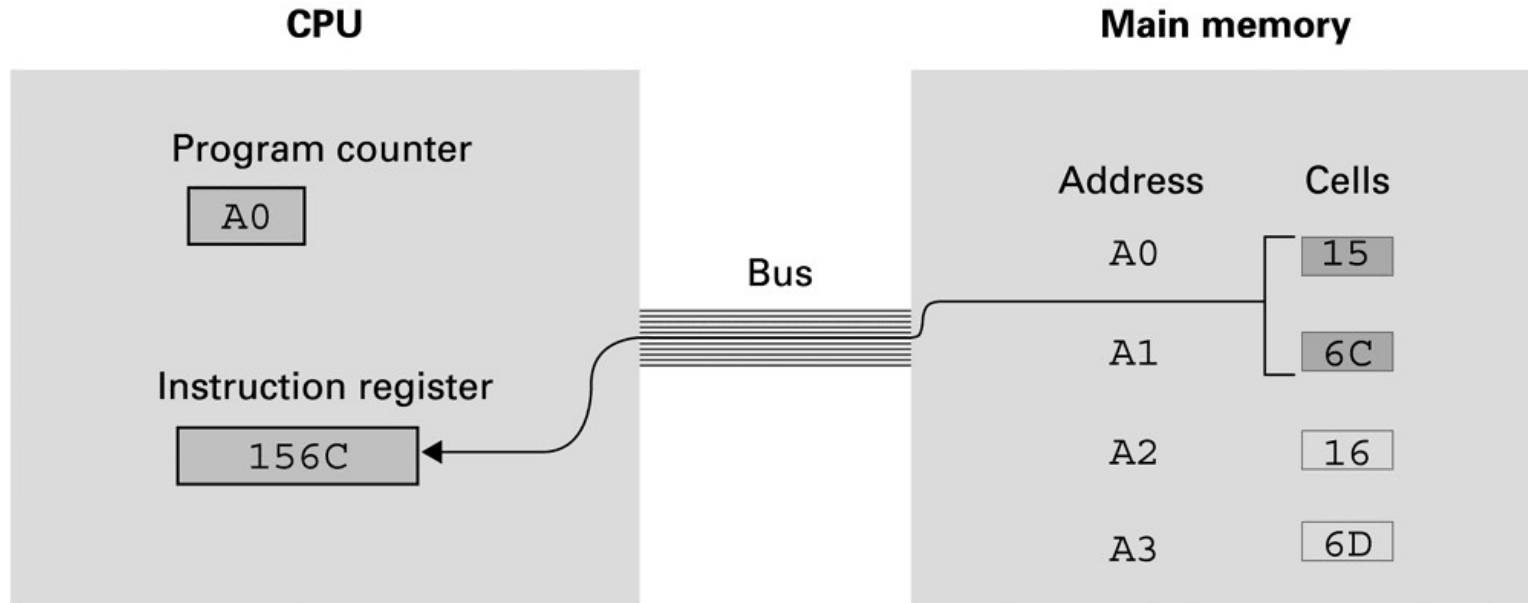
Load 156C 166D 5056 306E C000 into A0 through A9.

Set M[6C] = 05 and M[6D] = 03. Start with M[6E] = 00.

Set PC = A0. Begin with general registers at 00.

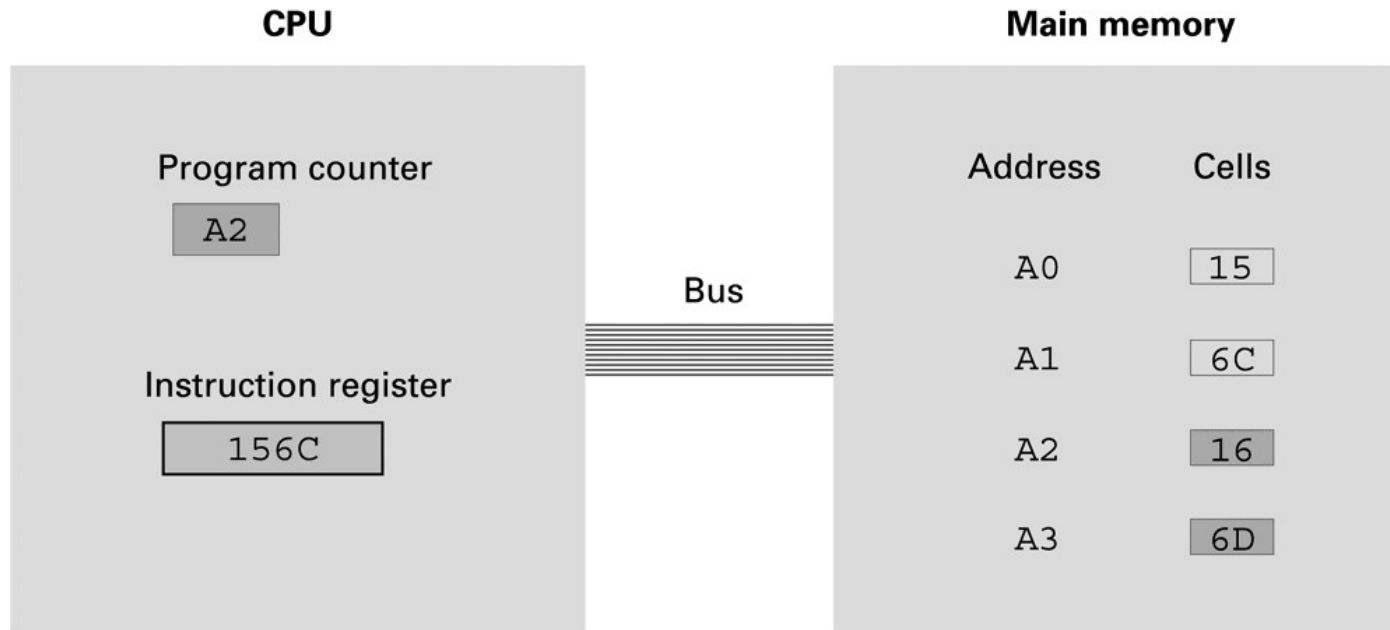
Each instruction uses two bytes. Predict the result before running.

Fetching an instruction



- a. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.

Advancing the program counter



b. Then the program counter is incremented so that it points to the next instruction.

Execution trace

Executed	Next PC	R5	R6	R0	M[6E]
156C	A2	05	00	00	00
166D	A4	05	03	00	00
5056	A6	05	03	08	00
306E	A8	05	03	08	08
C000	AA	05	03	08	08

All entries are hexadecimal. This VizMachine core advances PC before executing HALT.

The same program with new input

Keep all five instructions unchanged.

Set $M[6C] = 09$ and $M[6D] = 04$. Reset PC to A0.

Predict the result, then step through the program.

Result: $R0 = 0D$ and $M[6E] = 0D$. The program now computes $9 + 4$.

Memory load and literal load

Instruction	Meaning	R5 after execution
156C	Read M[6C]	09 if M[6C] = 09
256C	Use literal byte 6C	6C, regardless of M[6C]

The opcode changes the meaning of the final two digits. All displayed values are hex.

A small coding change

Modify the addition program so that its result goes to cell 70.

Which instruction changes? Do the LOAD instructions change?

Answer: replace 306E with 3070. Keep both LOADs and ADD unchanged.

Run with inputs 09 and 04. M[70] becomes 0D.

Copying two bytes with the CPU

Address	Instruction	Effect
A0	1150	R1 gets M[50]
A2	3160	M[60] gets R1
A4	1151	R1 gets M[51]
A6	3161	M[61] gets R1
A8	C000	Halt

Set $M[50] = 2A$ and $M[51] = 7C$. Start at A0. The CPU executes every load and store.

Copy results and the hardware cost

After the program: $M[60] = 2A$ and $M[61] = 7C$.

Source cells 50 and 51 retain their contents.

Each byte copy takes a LOAD and a STORE, plus instruction fetches.

How much CPU work would copying thousands of bytes require?

A branch changes the next address

Before B258 at address 20	Next PC	Reason
R0 = 03, R2 = 03	58	The register values match
R0 = 03, R2 = 04	22	Continue with the next instruction

B258 compares R2 with R0. It does not compare R2 with the literal value 58.

Arithmetic/Logic Operations

Logic: AND, OR and XOR. Masks select or change bits.

Shifts and rotations rearrange bit positions.

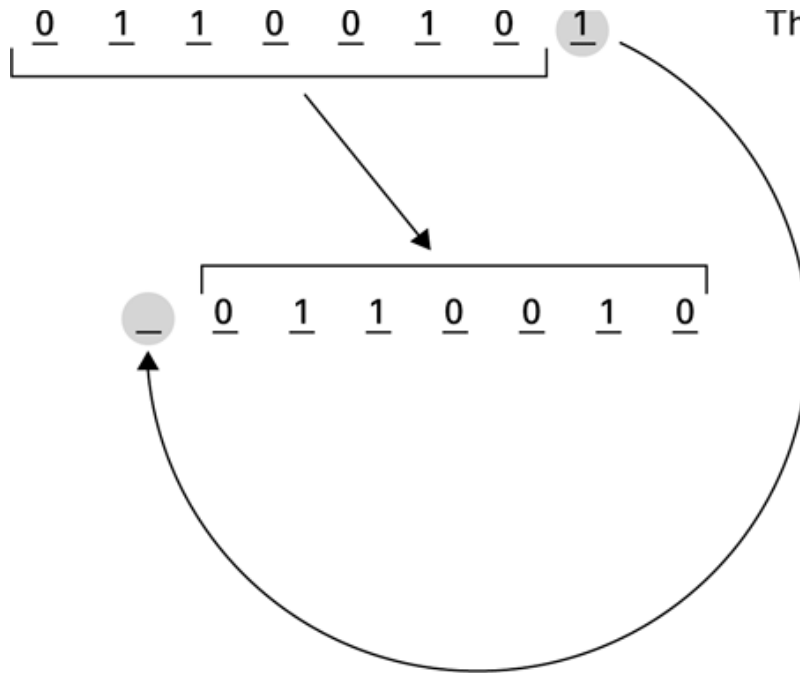
Arithmetic operations depend on the instruction set.

Integer and floating-point encodings need different arithmetic rules.

Historical arithmetic hardware: the available operations depend on the processor.



Rotate 65 one bit to the right



The original bit pattern

The bits move one position to the right. The rightmost bit "falls off" the end and is placed in the hole at the other end.

1 0 1 1 0 0 1 0

The final bit pattern

Bit masking in VizMachine

Instruction	Effect	Result
219A	R1 gets 9A	1001 1010
220F	R2 gets 0F	0000 1111
8312	R3 gets R1 AND R2	0000 1010 = 0A
3360	Store R3 at 60	M[60] = 0A
C000	Halt	Execution stops

Load at A0 and start PC there. A mask can isolate selected bits in a status byte.

Controller: An intermediary apparatus that handles communication between the computer and a device

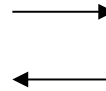
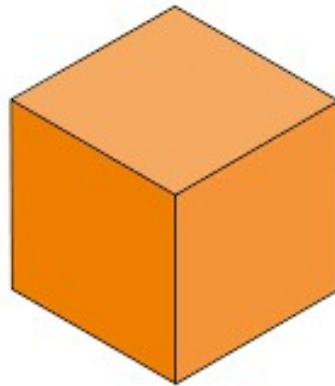
Specialized controllers for each type of device

General purpose controllers (USB and FireWire)

Port: The point at which a device connects to a computer

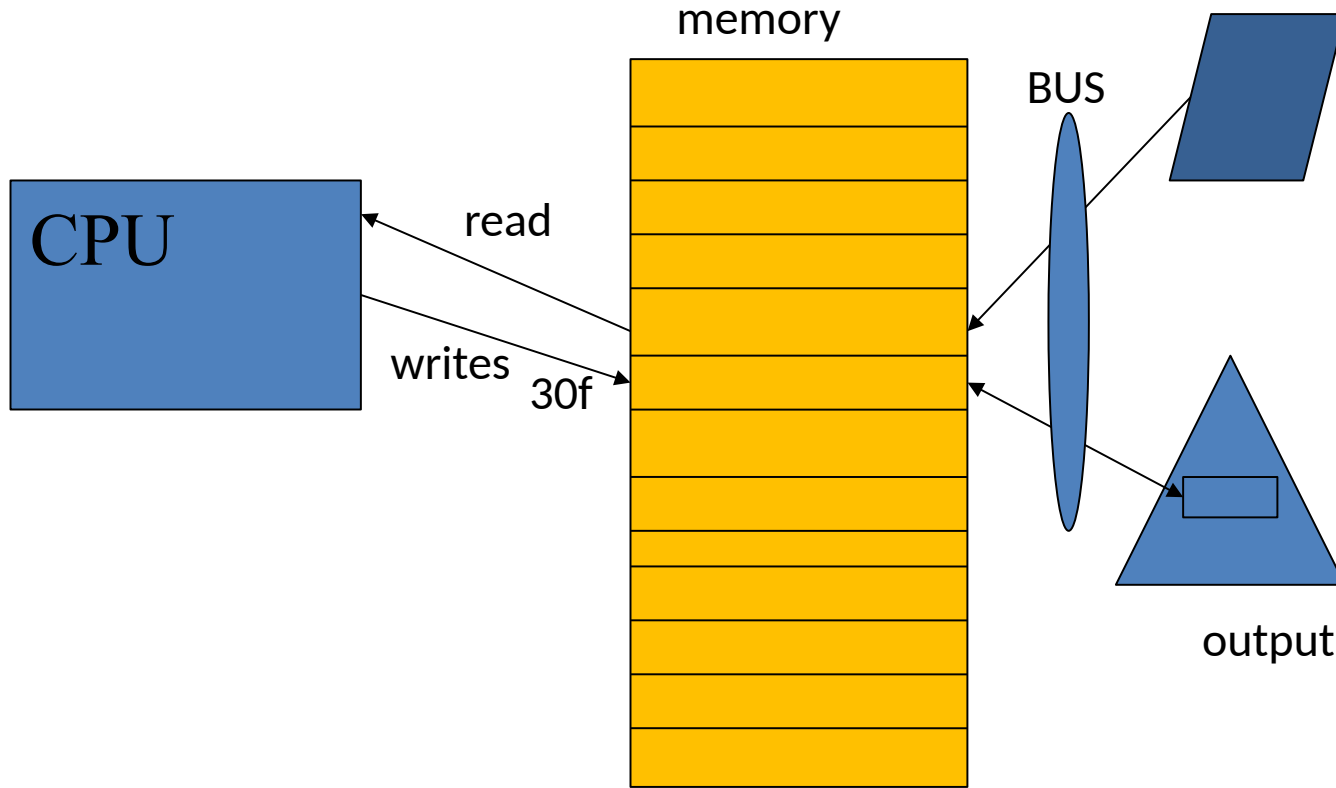
Memory-mapped I/O: CPU communicates with peripheral devices as though they were memory cells

- Is this a computer



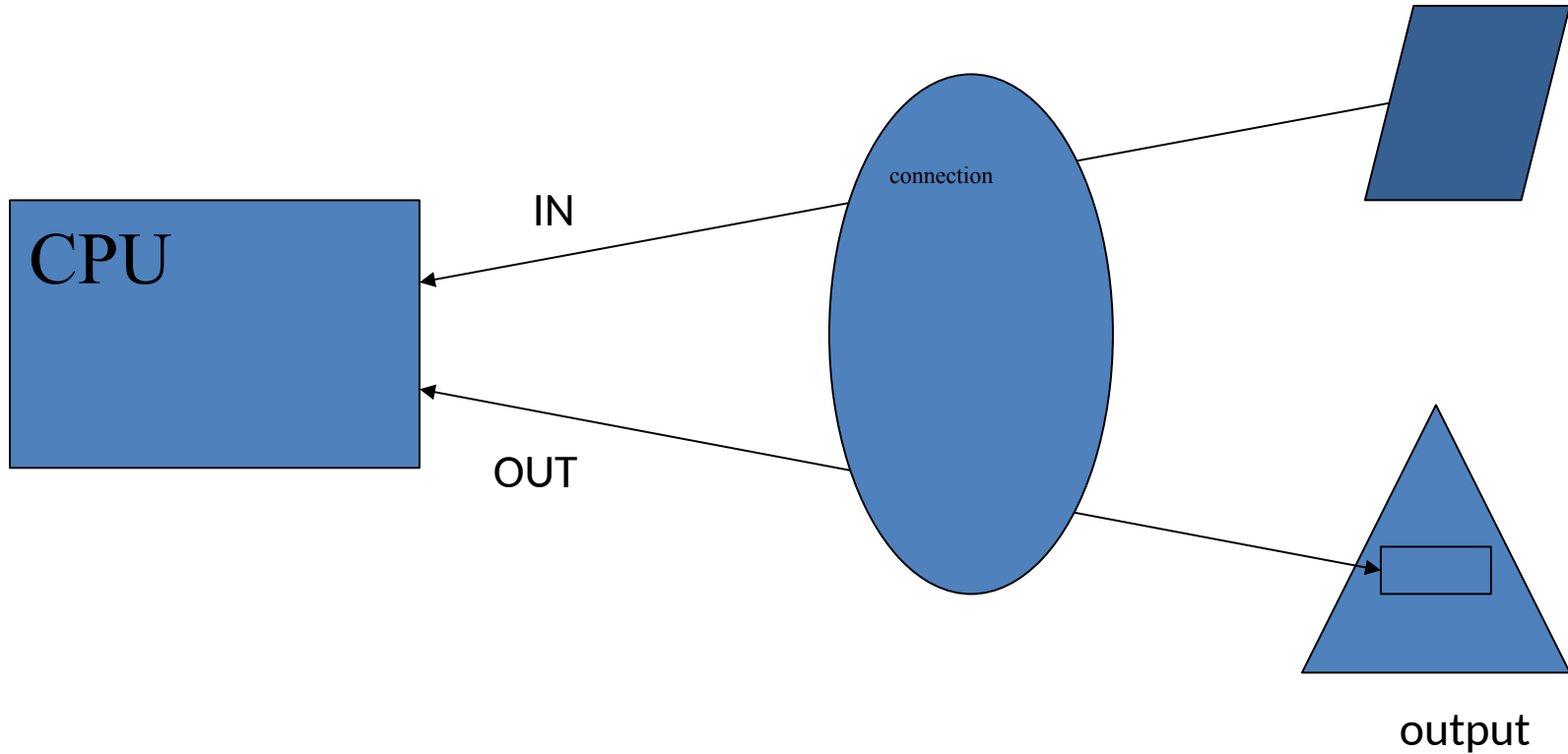
Input and output is
necessary

Memory mapped I/O

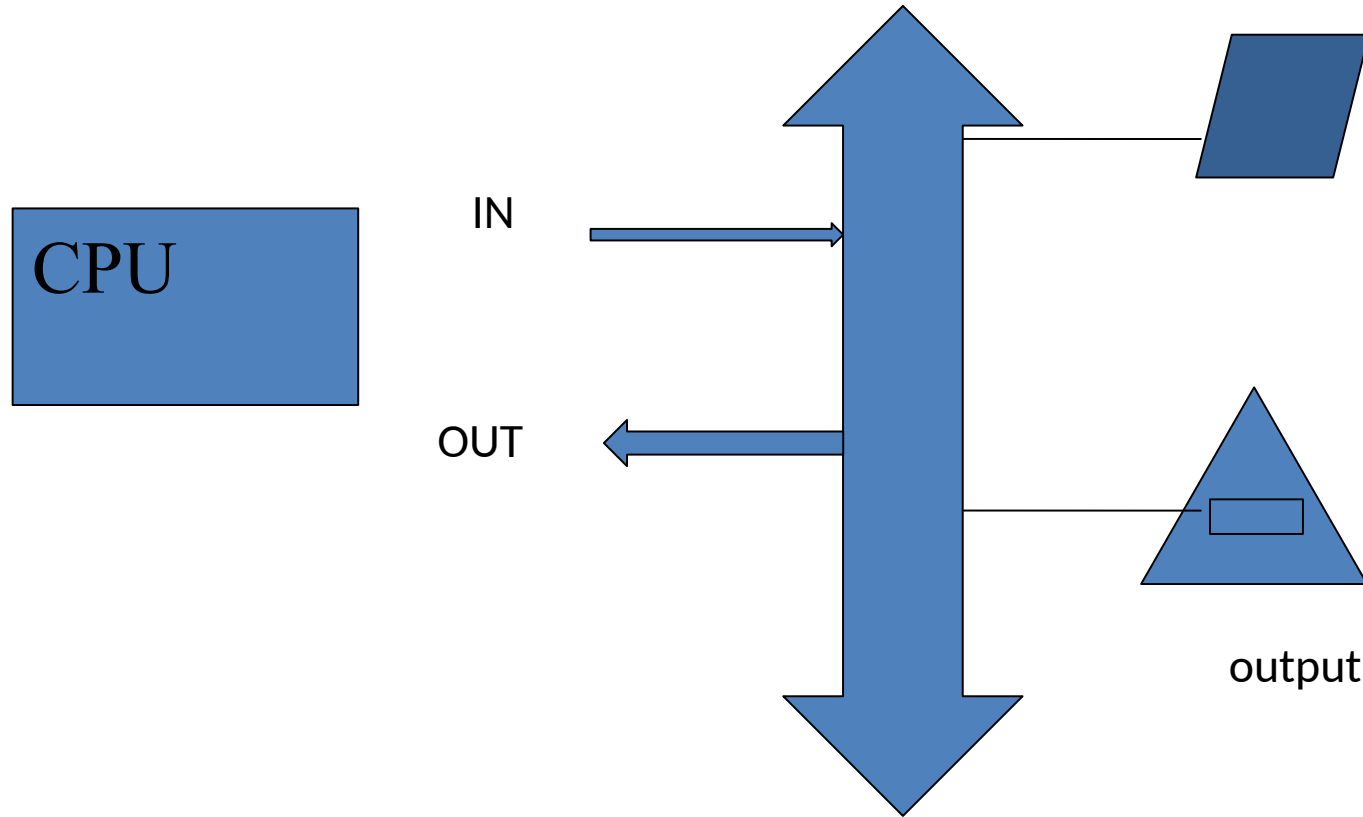


Some addresses select device registers. A read or write can interact with hardware.

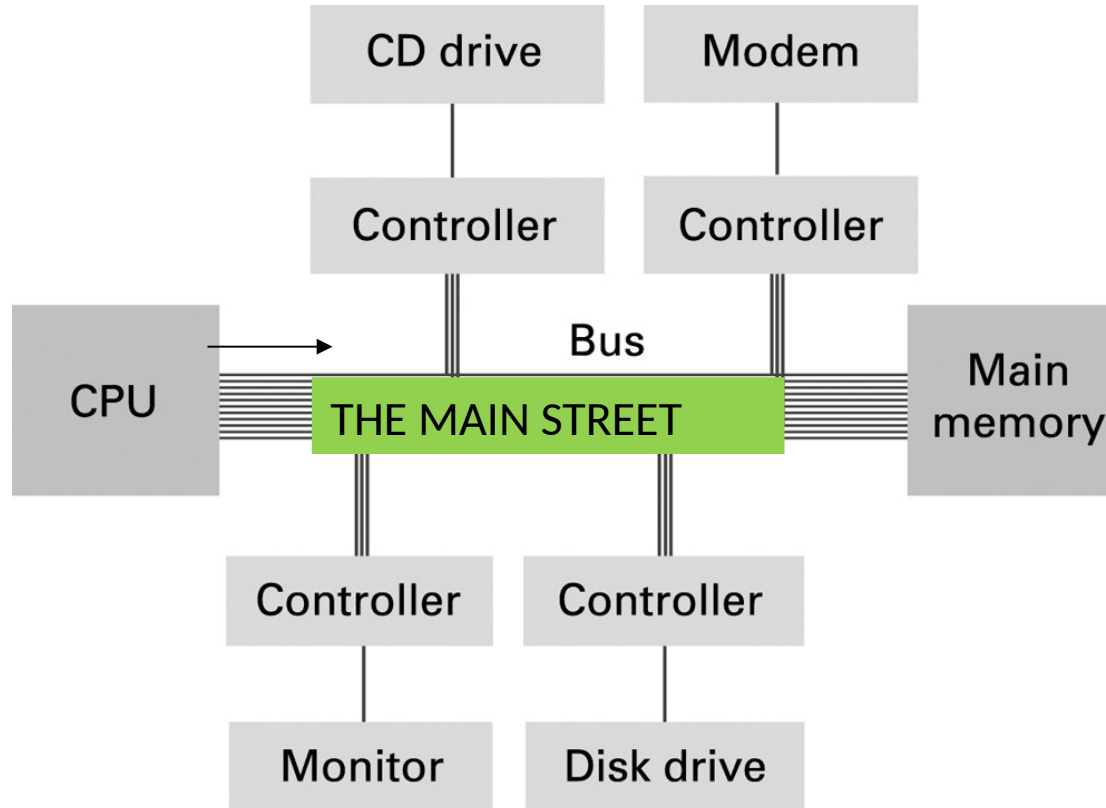
PORT mapped I/O



PORT mapped I/O



Controllers on a shared bus

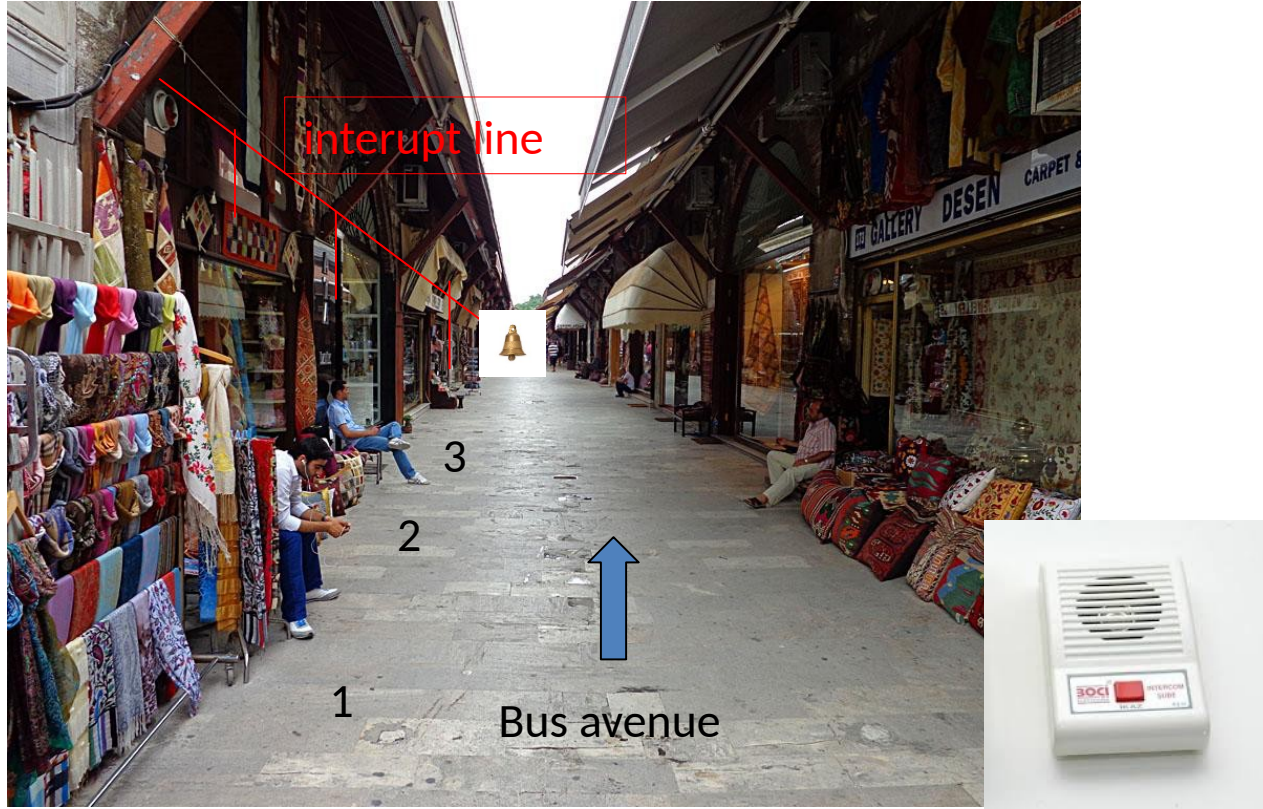


The bus as a shared street



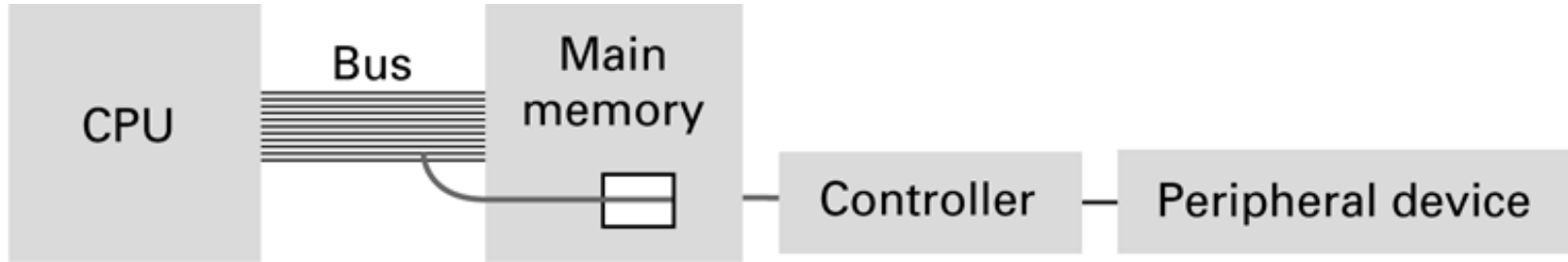
Shared access: many shops use one street to request and receive service.

Requests and the shared street



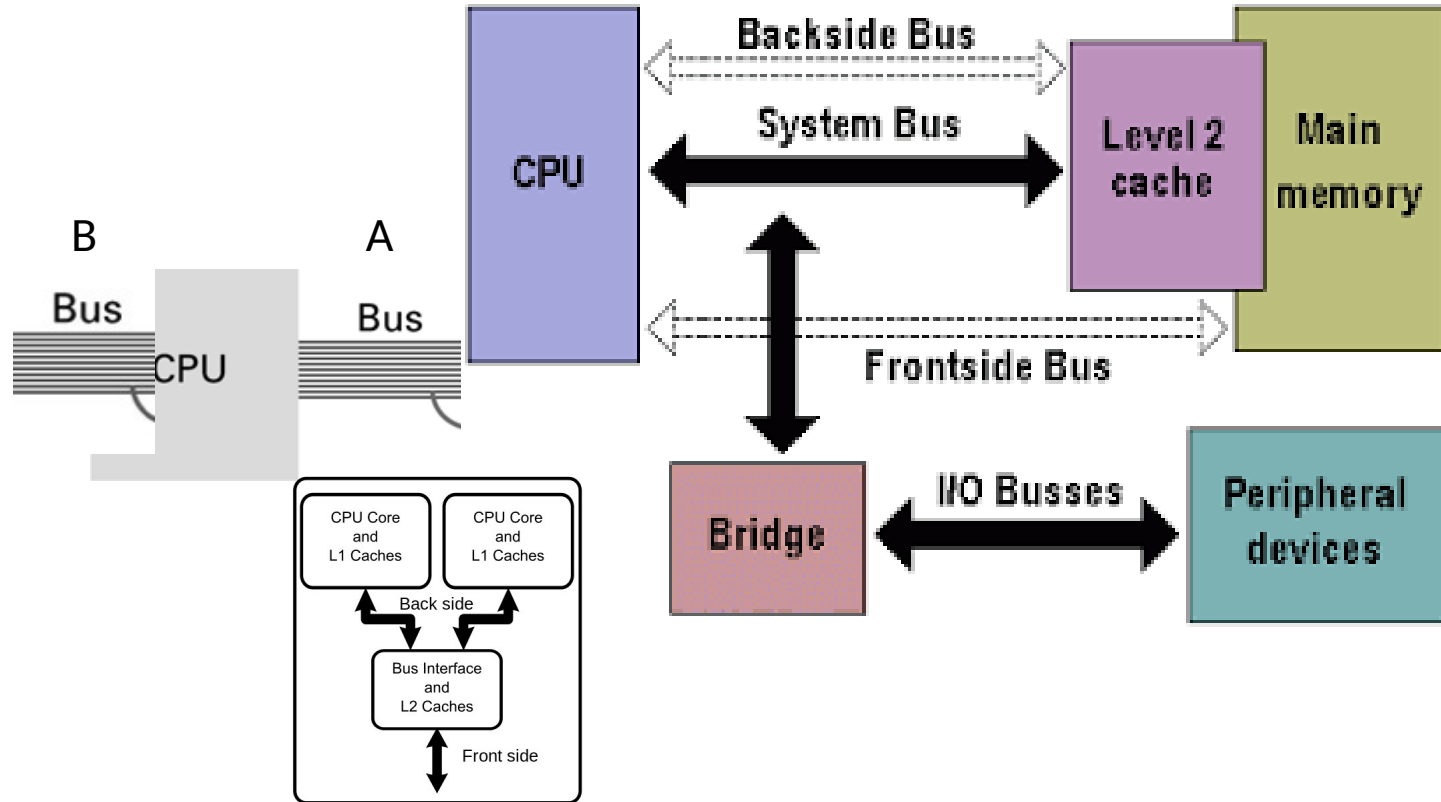
A request signals that service is needed. The shared path still has limited capacity.

Memory-mapped I/O



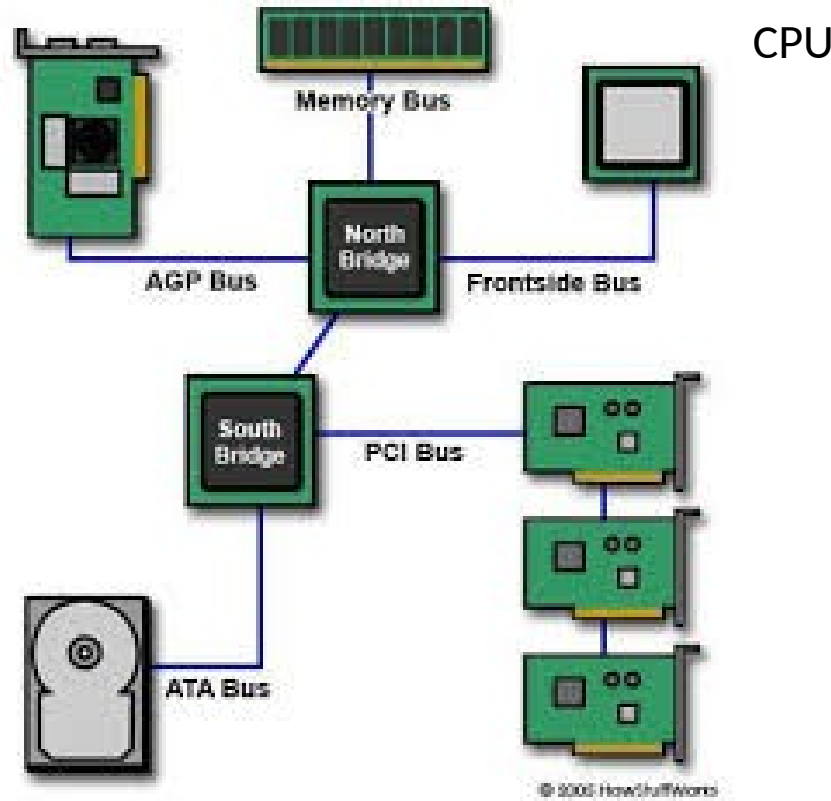
The mapped region represents device registers in the address space, rather than ordinary RAM.

Caches and bus connections



Historical bus arrangement: caches and separate paths reduce repeated main-memory traffic.

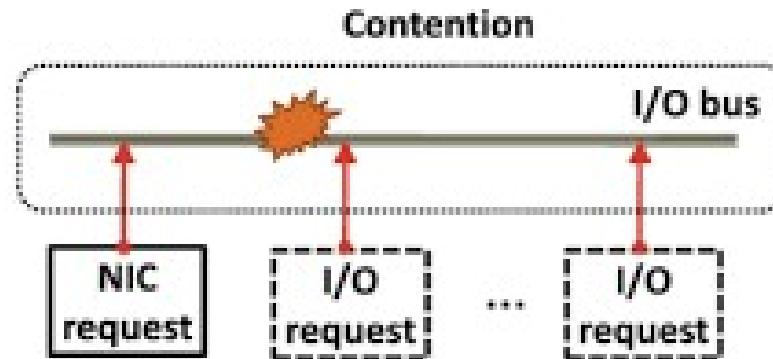
Multiple buses



Historical PC example: several buses connect components with different roles and speeds.

Bus contention

- ????



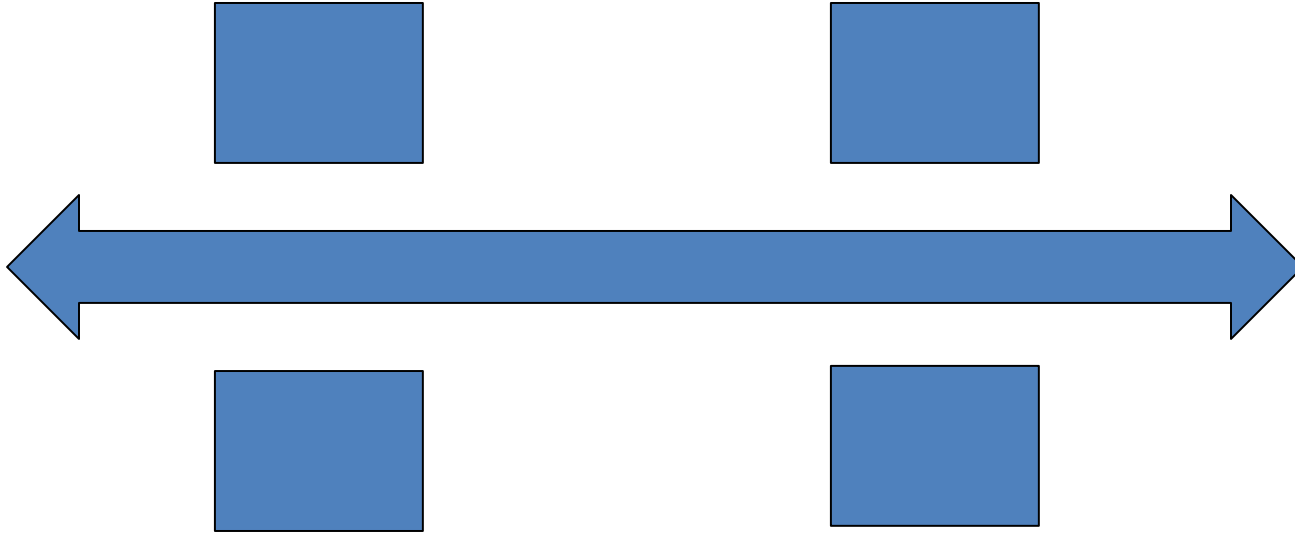
Multiple requesters compete for a shared resource. Arbitration decides who uses it next.

Direct memory access (DMA): Main memory access by a controller over the bus

Von Neumann Bottleneck: Insufficient bus speed impedes performance

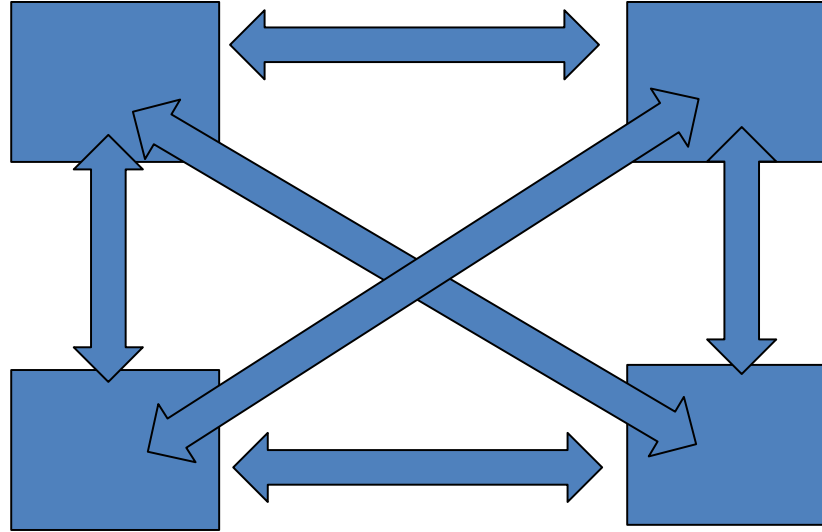
Handshaking: The process of coordinating the transfer of data between components

A shared communication path



One shared path limits simultaneous transfers.

More paths, more connections



Extra connections can allow more simultaneous traffic, at a hardware cost.

Direct memory access

- **DMA: a controller transfers data to or from main memory.**

Arrange
direct
delivery



The CPU arranges the transfer. A controller moves the data without a CPU copy for every byte.

Why real systems use DMA

Our byte-copy program made the CPU load and store each byte.

With DMA, software supplies a buffer and transfer details to a controller.

The controller transfers data while the CPU can perform other work.

Software checks completion and synchronizes access to the buffer.

Handshaking

- Synchronization !!

Half handshake

Ali: Send me pl
Veli: Here you go



Request and acknowledgement coordinate sender and receiver.

Full handshake

Ali: send me pl
Veli: here you go
Ali says: thank you
Veli says: no prob

Strobing

Veli: get this !!

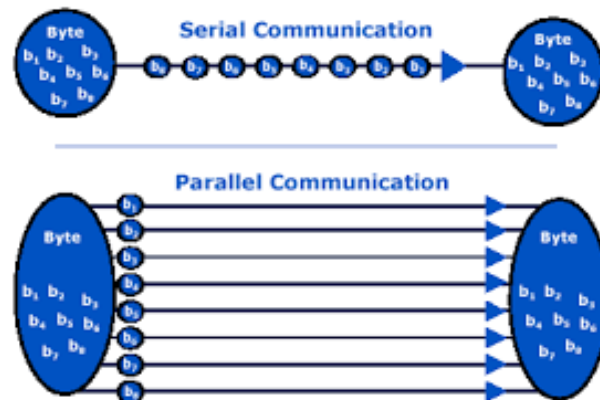
Handshaking with a buffer



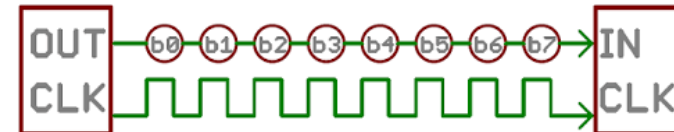
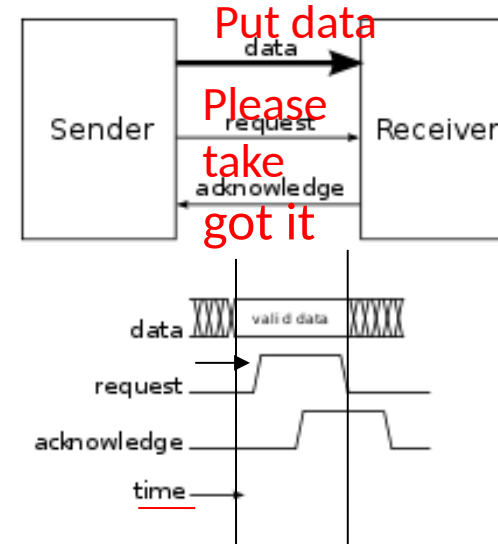
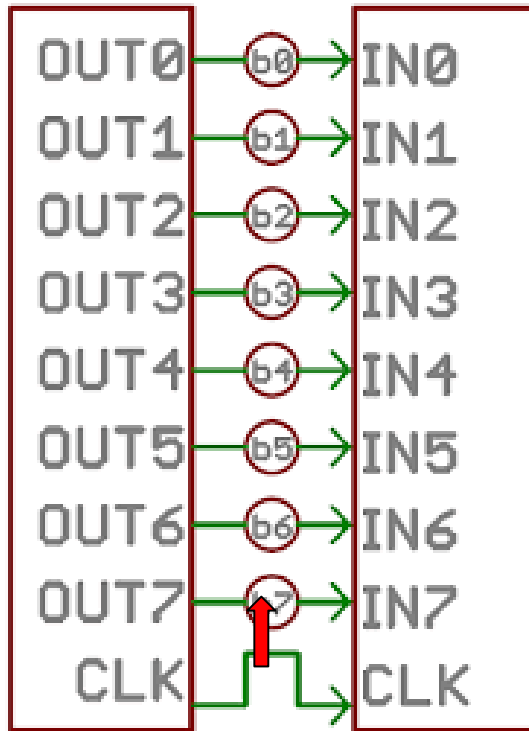
A buffer absorbs temporary differences in arrival and consumption rates.

Parallel Communication: Several communication paths transfer bits simultaneously.

Serial Communication: Bits are transferred one after the other over a single communication path.



Clocked and asynchronous transfers

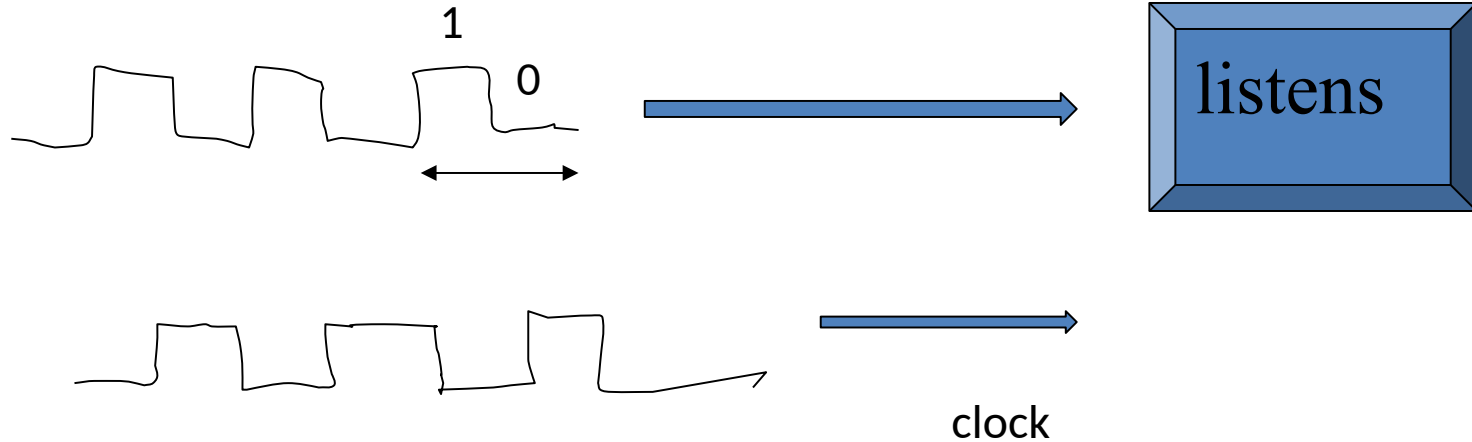


Detecting incoming data



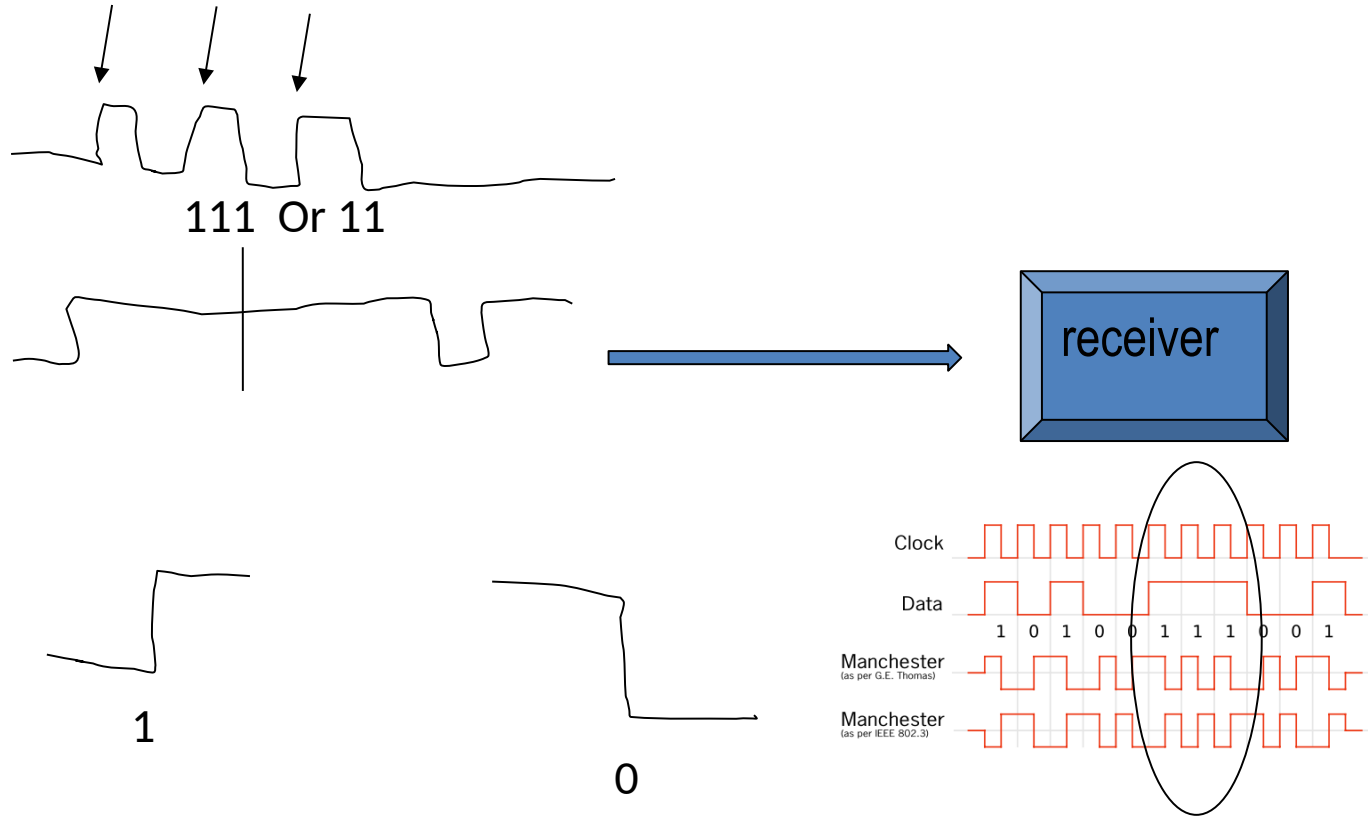
The receiver needs a way to identify when new information arrives.

Data and a separate clock



A timing signal helps the receiver determine when to sample the data.

Manchester coding



A transition within each bit interval carries timing information.

Data communication rates

b/s or bps means bits per second. B/s means bytes per second.

1 kb/s = 1,000 b/s. 1 Mb/s = 1,000,000 b/s.

An ideal 8 Mb/s link carries 1 MB/s before protocol overhead.

Bandwidth is capacity. Achieved throughput can be lower.

Technologies to increase throughput:

Pipelining: Overlap steps of the machine cycle

Parallel Processing: Use multiple processors simultaneously

SISD: No parallel processing

MIMD: Different programs, different data

SIMD: Same program, different data

Architecture: the ideas to retain

VizMachine makes instructions, registers and changing state visible.

Devices require communication, coordination and temporary storage.

Buses have limited capacity. DMA reduces CPU work for data transfers.

Caches, pipelining and parallel execution address different performance limits.