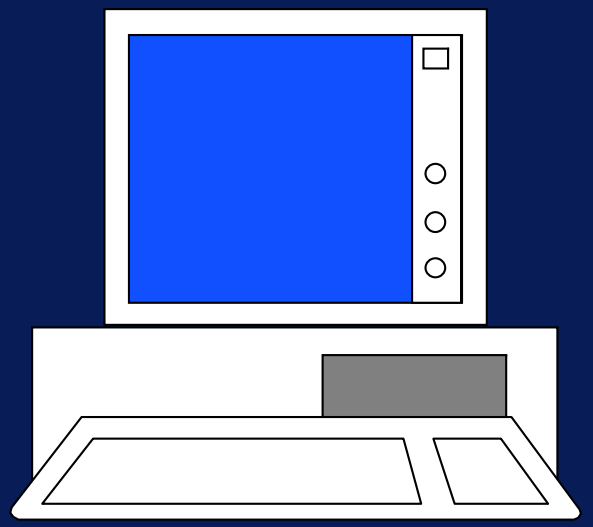
An Introduction to Computer Systems

David Vernon

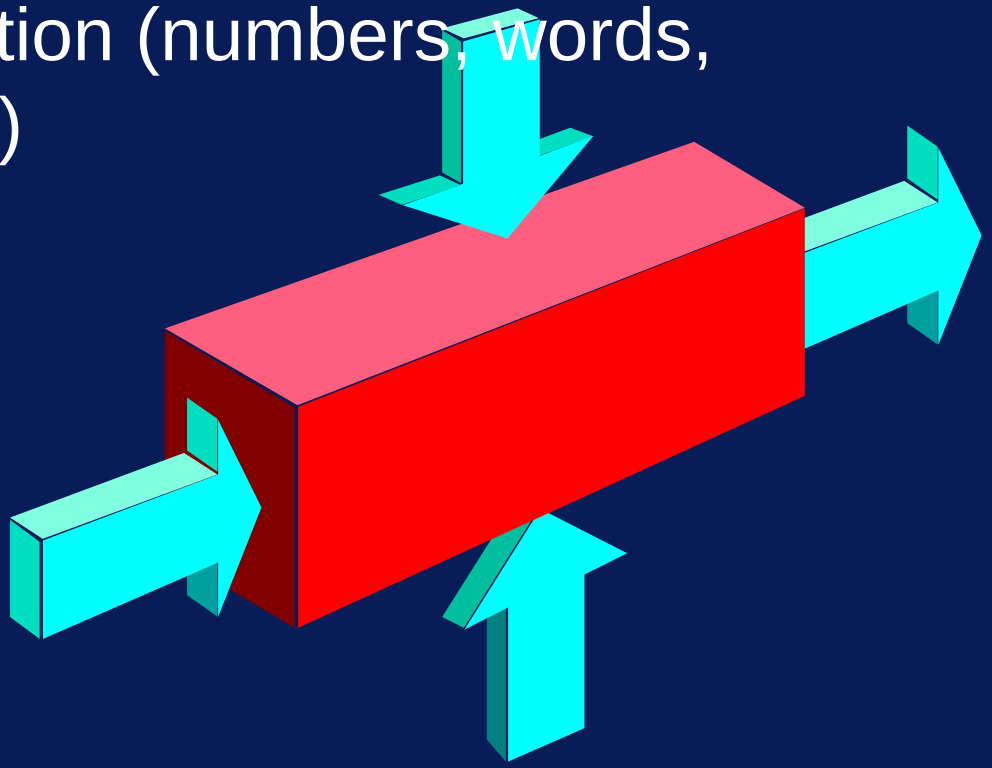
A Computer ....

- takes input
- processes it according to stored instructions
- produces results as output

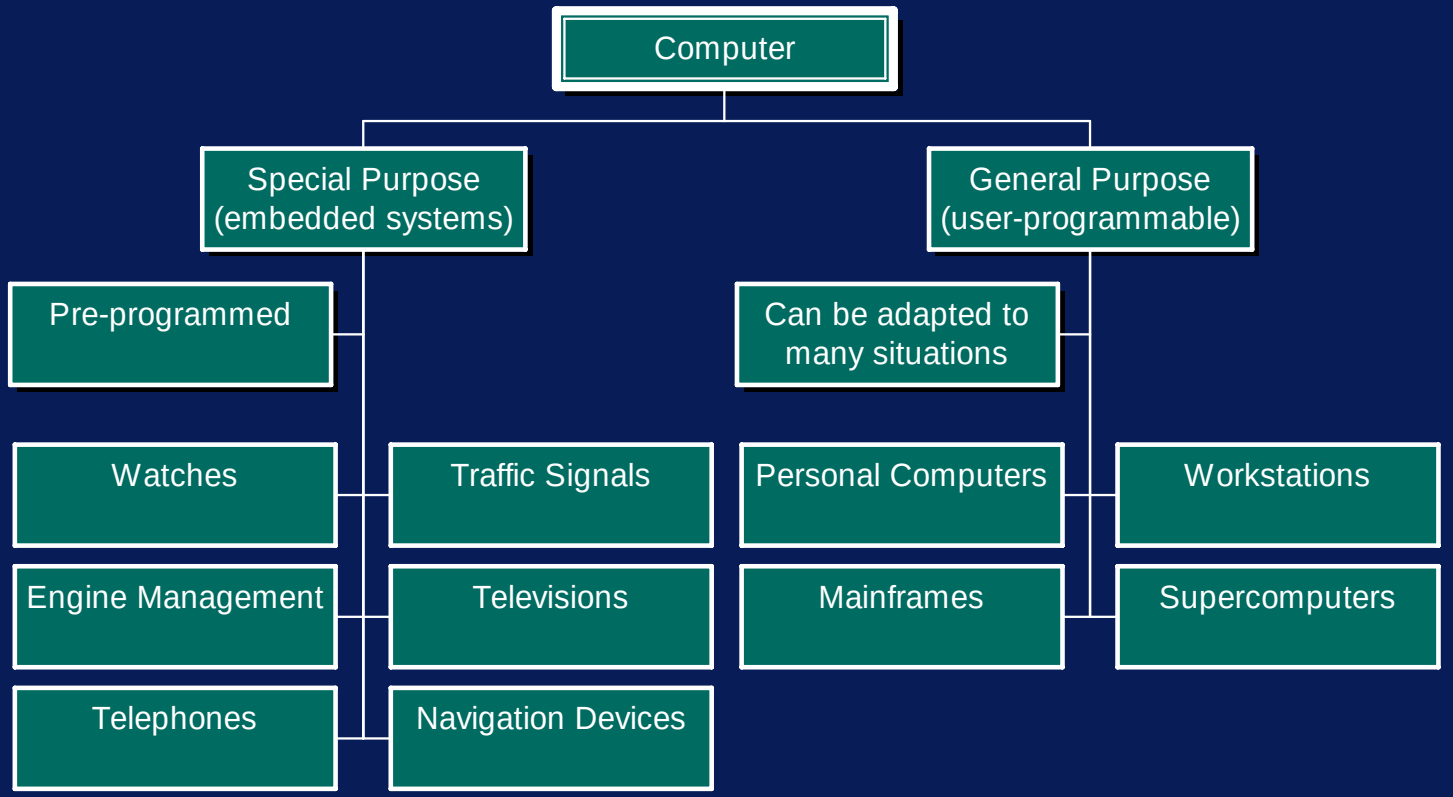


Key Concepts

- Input: Data
- Instructions: Software, Programs
- Output: Information (numbers, words, sounds, images)



Types of Computer



Data *vs* Information

- A
- 2, 4, 23, 30, 31, 36

- A
  - your grade in the exam
- 2, 4, 23, 30, 31, 36
  - Next week's Lotto numbers

Key Concepts

- Codes
  - Data and information can be represented as electrical signals (e.g. Morse code)
  - A code is a set of symbols (such as dots and dashes in Morse code) that represents another set of symbols,
    - » such as the letters of the alphabet,
    - » or integers or real numbers,
    - » or light in an image,
    - » for the tone of a violin

Key Concepts

- A circuit is an inter-connected set of electronic components that perform a function
- Integrated Circuits (ICs)
  - Combinations of thousands of circuits built on tiny pieces of silicon called chips

Key Concepts

- Binary signal (two state signal)
  - Data with two states
  - off & on
  - low voltage & high voltage
  - 0v & 5v

Key Concepts

- Bit
  - Single Binary Digit
  - Can have value 0 or 1, and nothing else
  - A bit is the smallest possible unit of information in a computer

Key Concepts

- Groups of bits can represent data or information
  - 1 bit - 2 alternatives
  - 2 bits - 4 alternatives
  - 3 bits - 8 alternatives
  - 4 bits - 16 alternatives
  - n bits - 2^{n} alternativies
  - 8bits - 2^{8} = 256 alternatives
  - a group of 8 bits is called a byte

4 Bits

- 0000
- 0001
- 0010
- 0011
- 0100
- 0101
- 0110
- 0111
- 1000
- 1001
- 1010
- 1011
- 1100
- 1101
- 1110
- 1111

Key Concepts

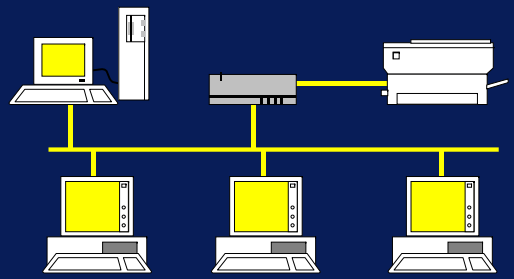
- Information System
  - A system that takes data, stores and processes it, and provides information as an output
  - An IS is a computer in use
  - The amount of data can be vast

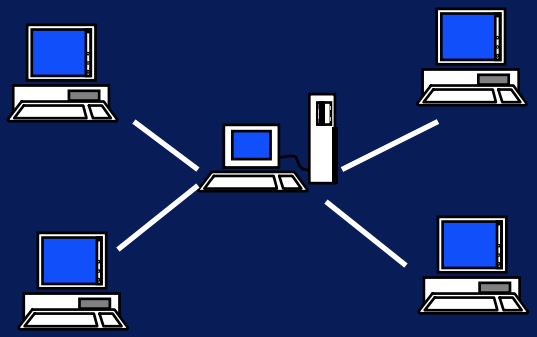
Key Concepts

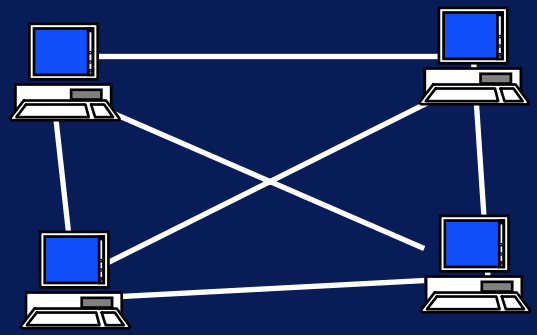
- Communication System
  - Communication: the transfer of meaningful information
  - Comprises
    - » a sender (transmitter)
    - » a channel over which to send the data
    - » a receiver

Key Concepts

- Network
  - Two and usually more communication devices connected together
  - Many connection topologies







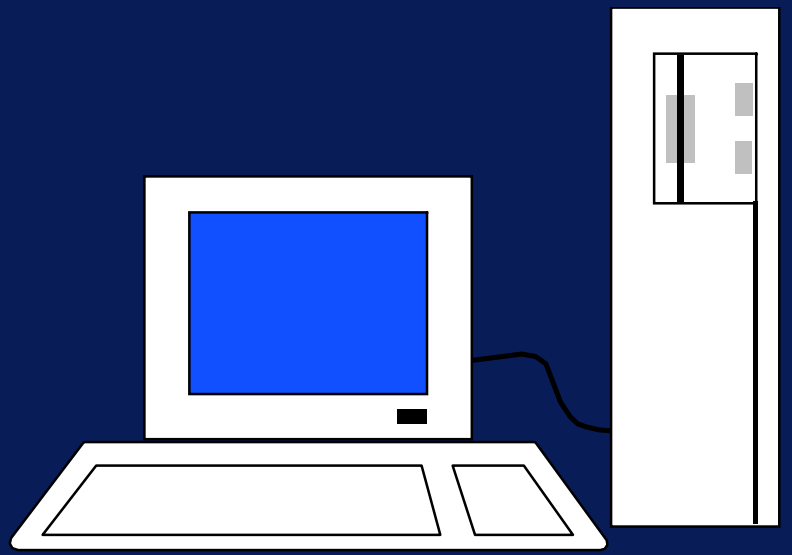
Key Concepts

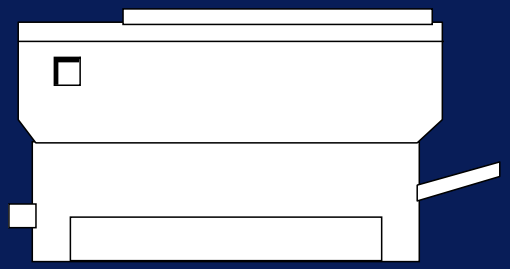
- Hardware
  - The physical (electronic and mechanical) parts of a computer or information system
- Software
  - The programs that control the operation of the computer system

Components of Computer Systems

Components of Computer Systems

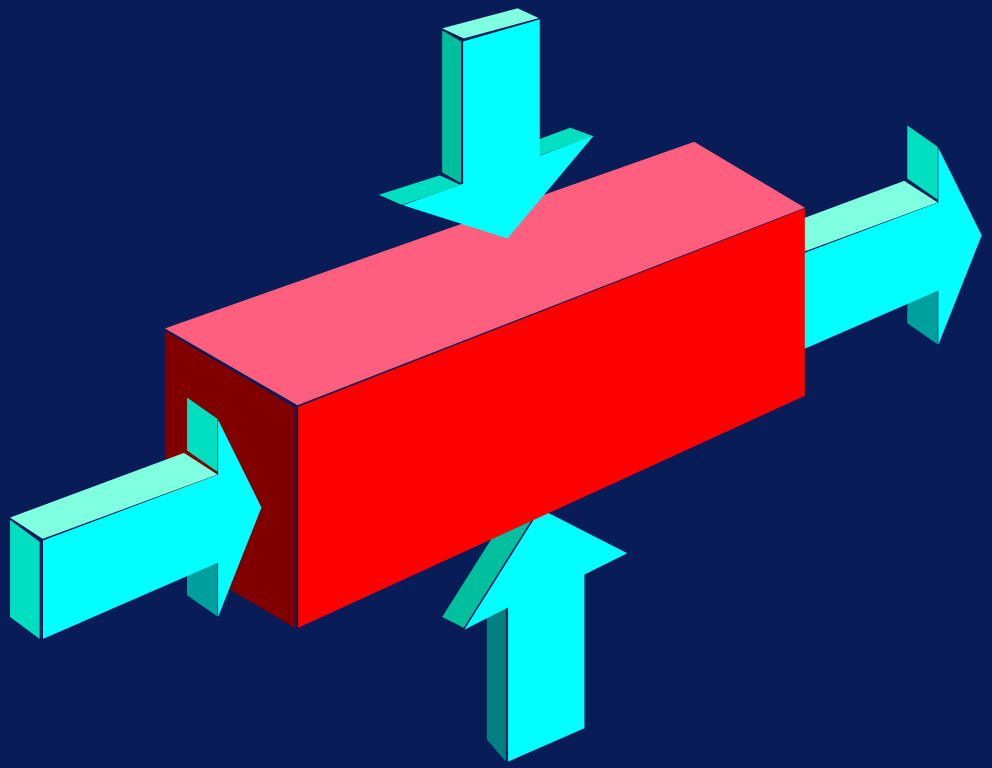
- Keyboard
- Display
- System Unit
- Storage
- Printer



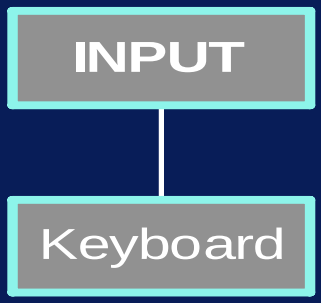


Key Components

- More Formally:
  - Input
  - Output
  - Storage
  - Processor

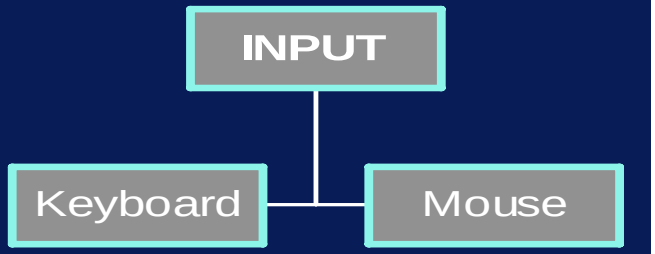


Input Systems

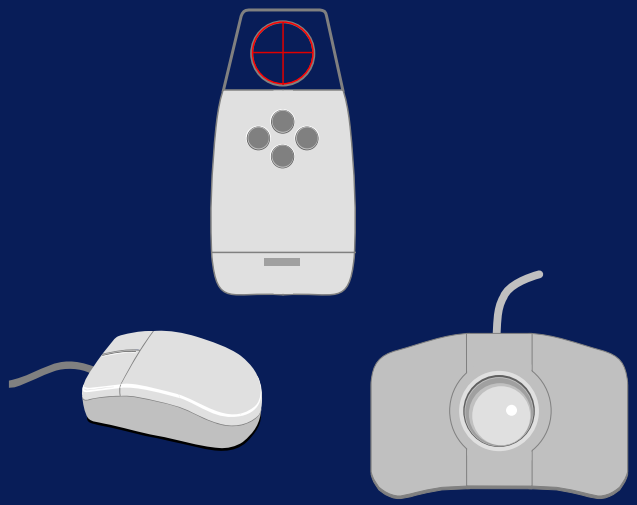


- Keyboard
  - » Most common input device
  - » QWERTY

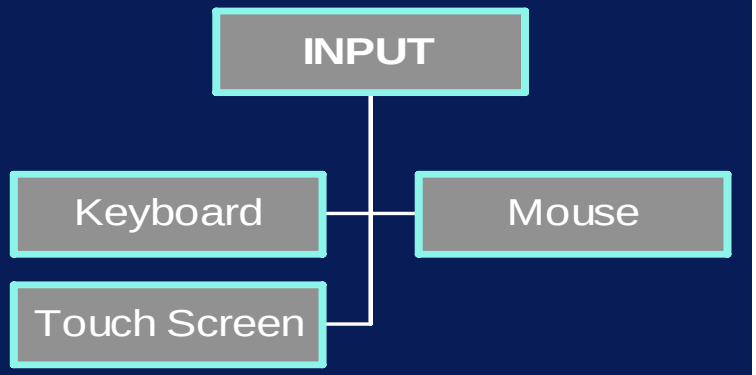
Input Systems



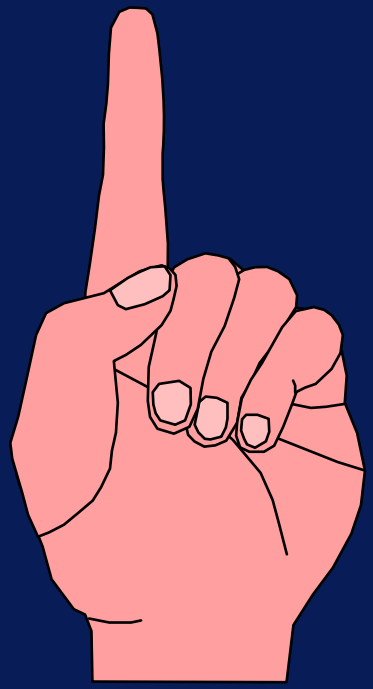
- Mouse
  - » Cursor manipulation device
  - » Trackball

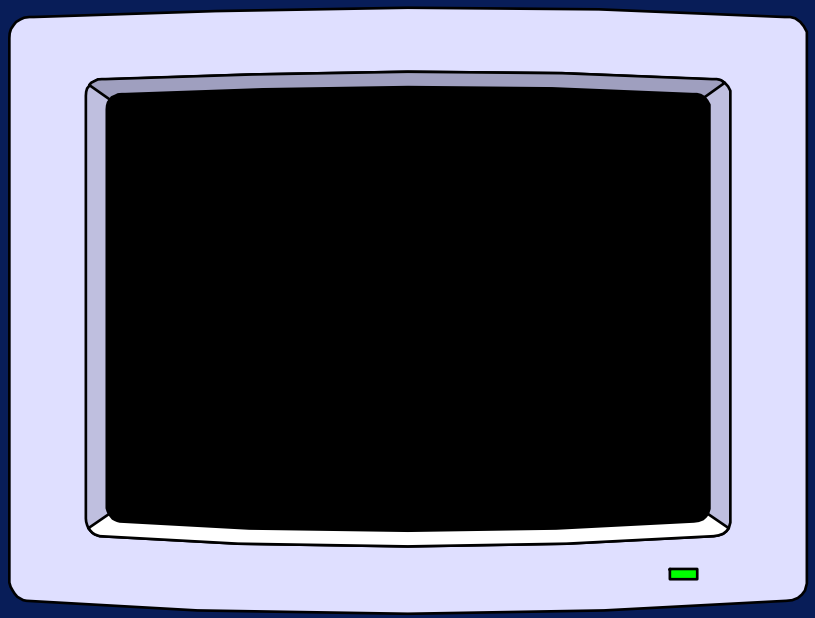


Input Systems

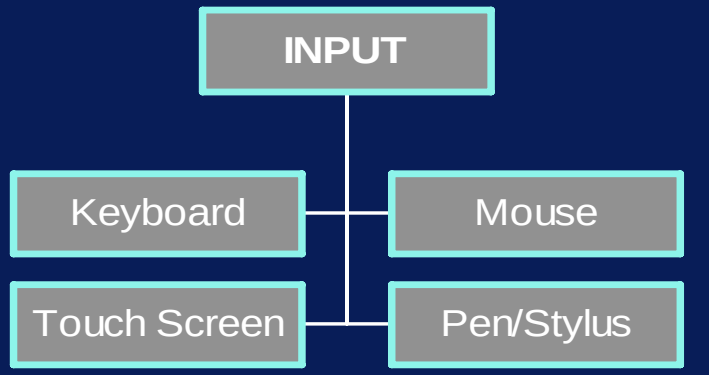


- Touch Screens

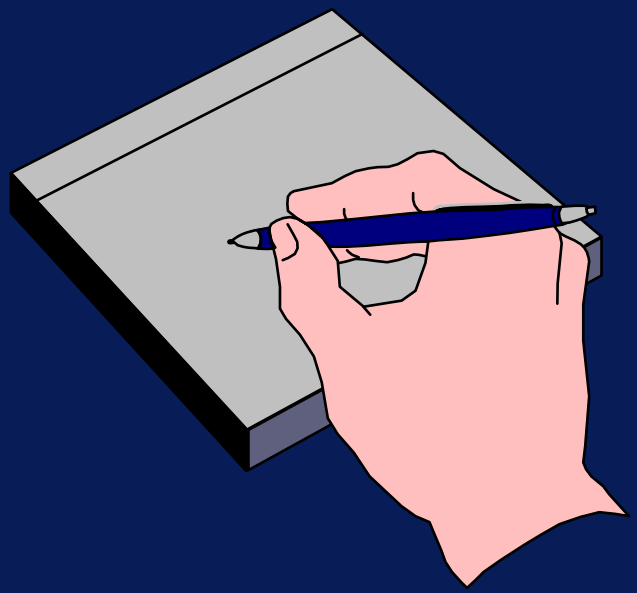


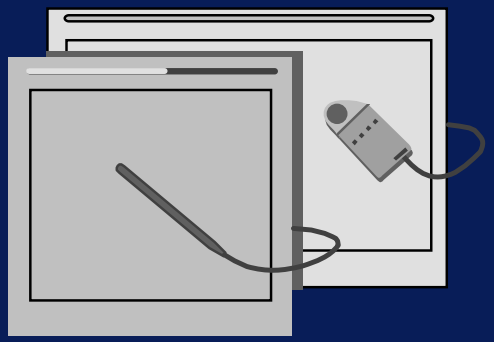


Input Systems

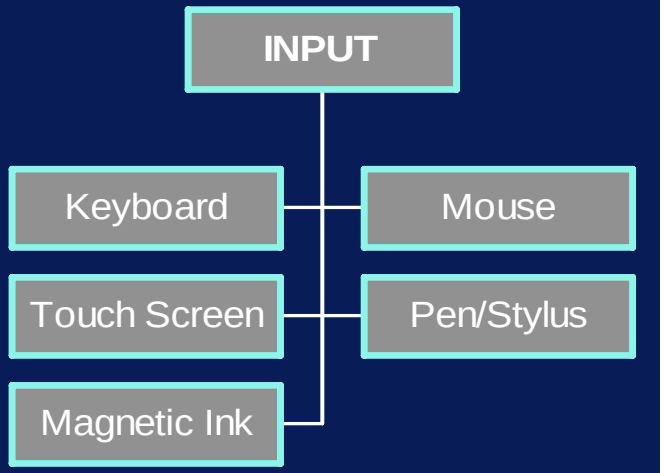


- Pens
- Stylus



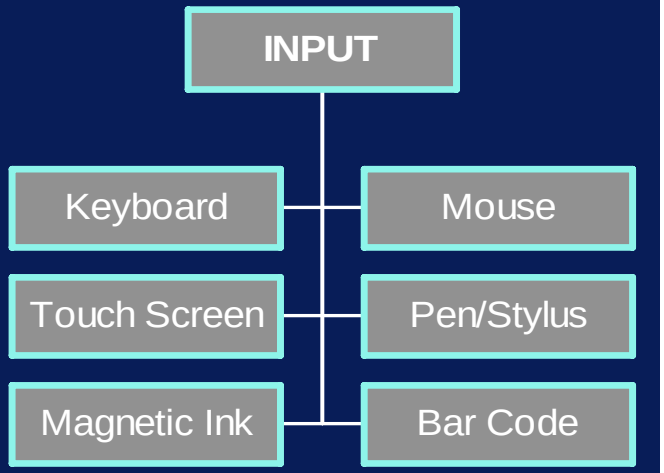


Input Systems

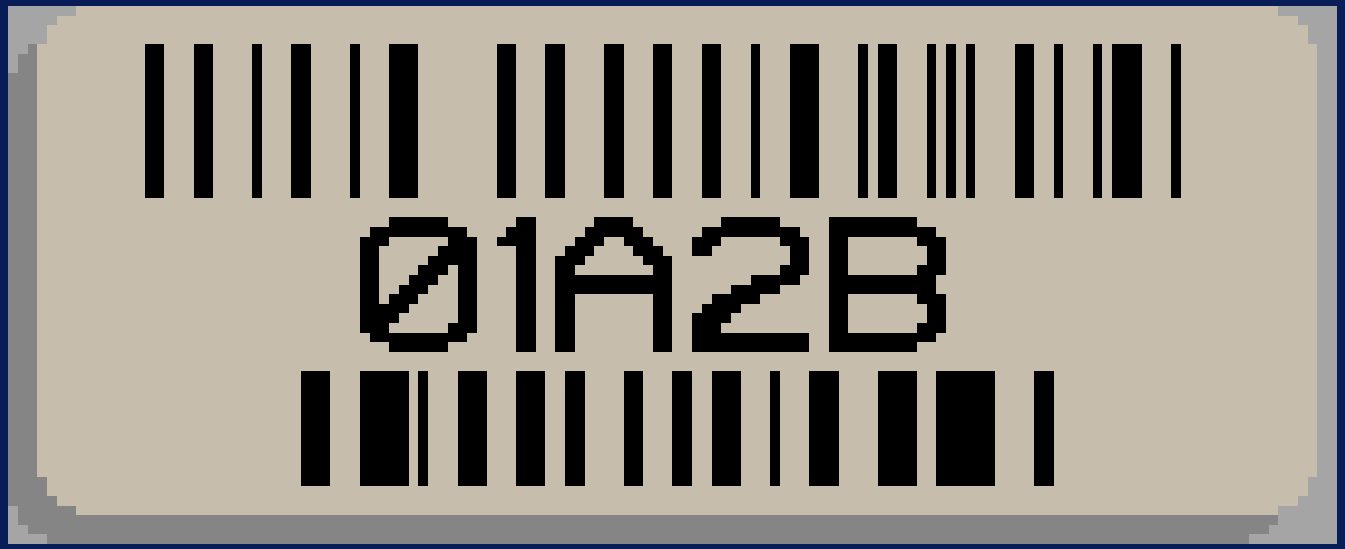


- Magnetic Ink Character Recognition (MICR)

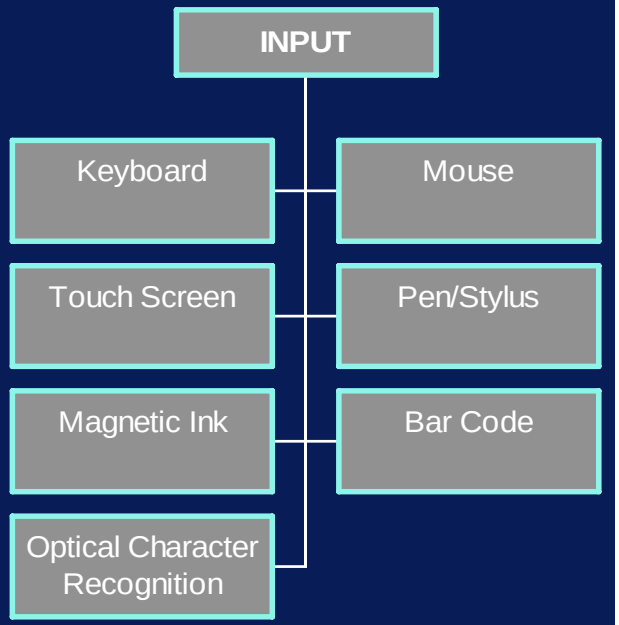
Input Systems



- Bar Code Readers

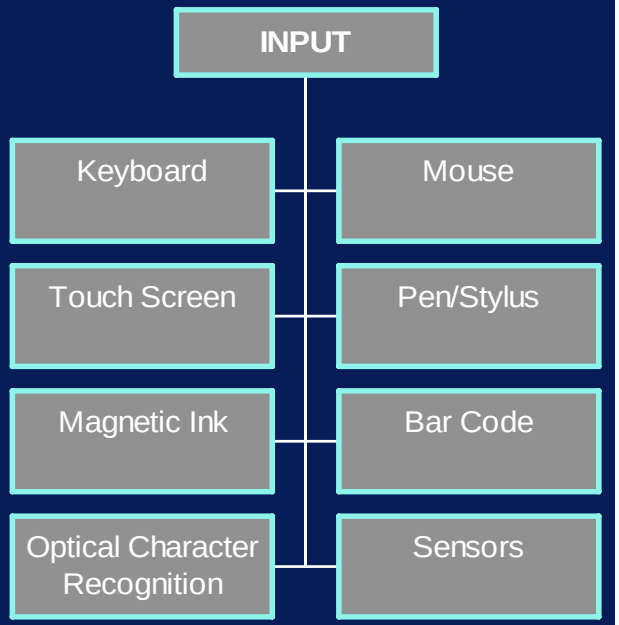


Input Systems

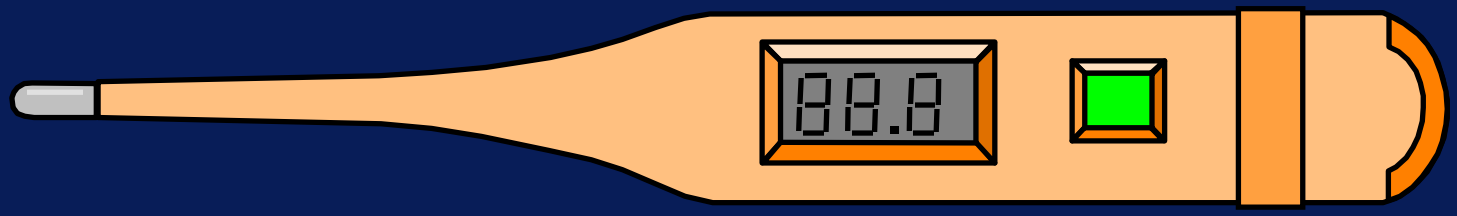


- Optical Character Recognition systems
- Book readers for the blind
- Automated input of text
- Can do typewritten text and handwritten block capital
- Problems with cursive handwriting recognition

Input Systems

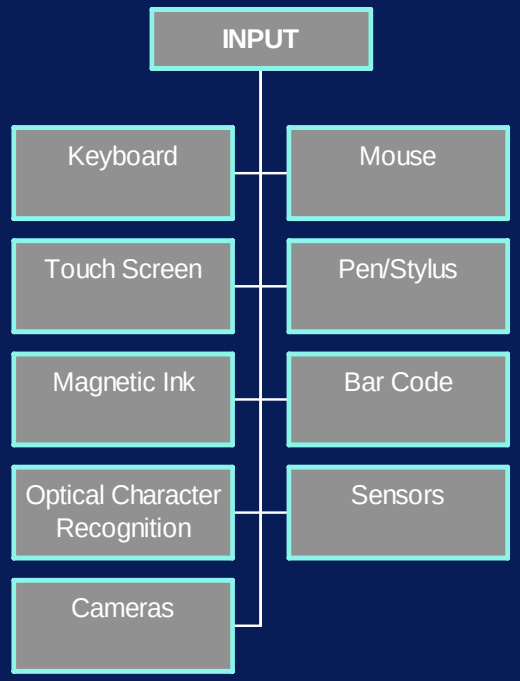


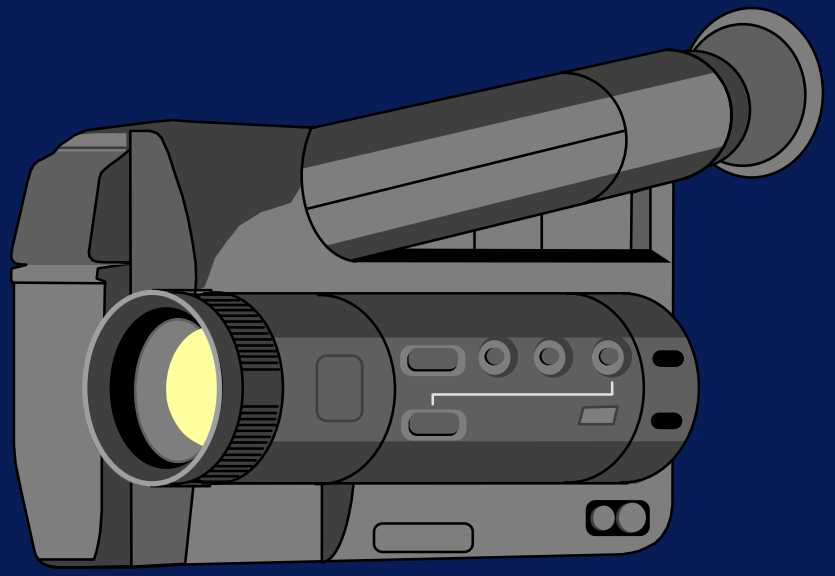
- Sensors
  - Digital thermometers
  - Accelerometers
  - Strain gauges (weighing scales)
  - ....



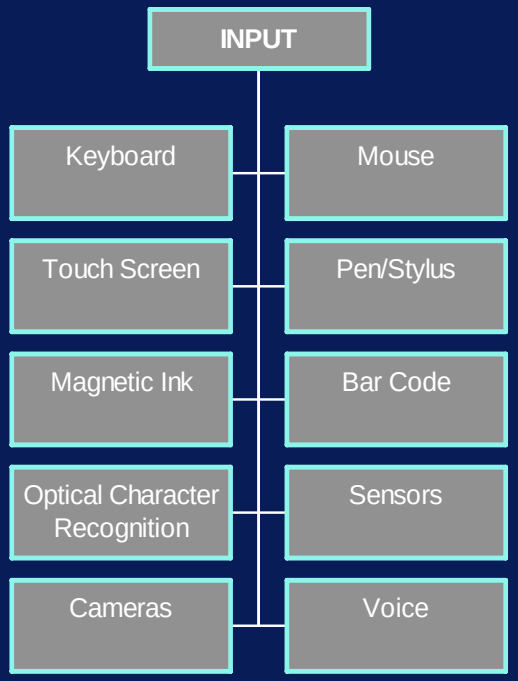
Input Systems

- Camera Systems
  - Surveillance and monitoring
  - Visual inspection
  - Robot guidance
  - Video conferencing

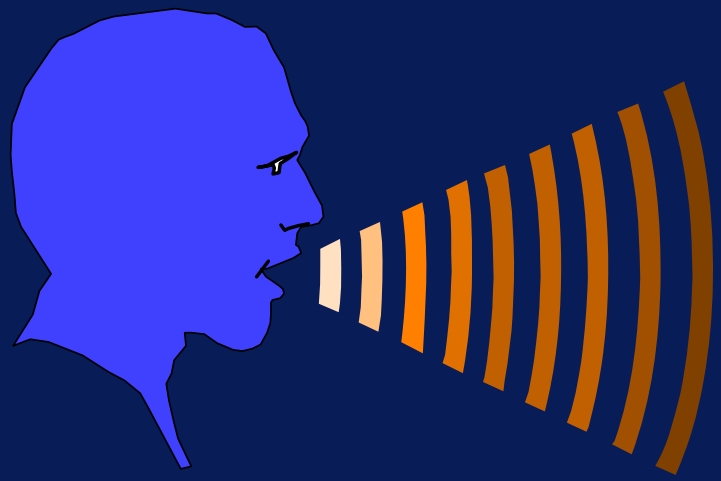


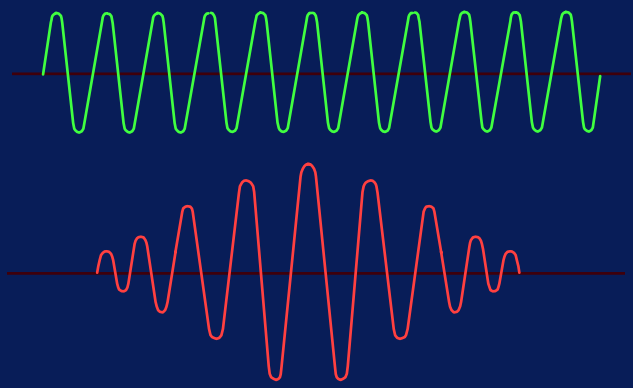


Input Systems



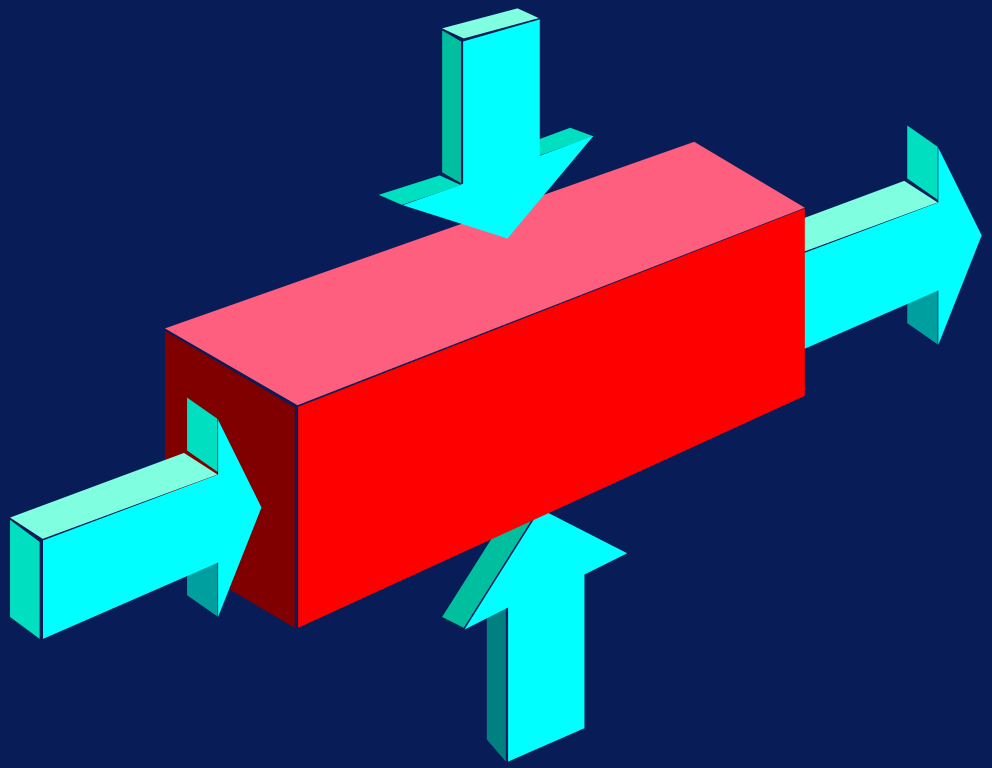
- Voice
  - Voice recognition
  - Hands-free car-phones
  - Assistance for the disabled



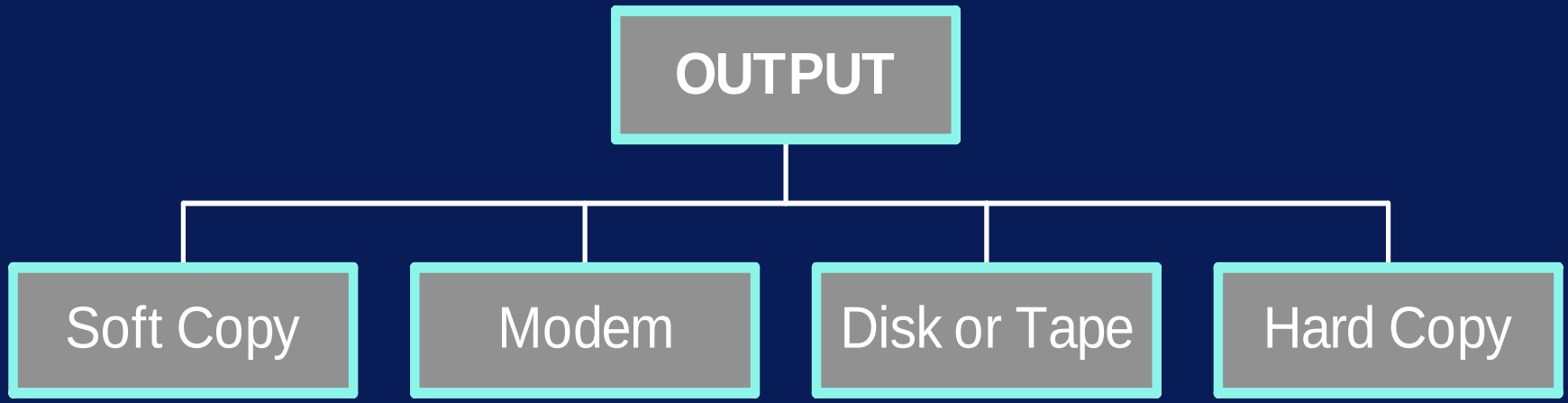


Key Components

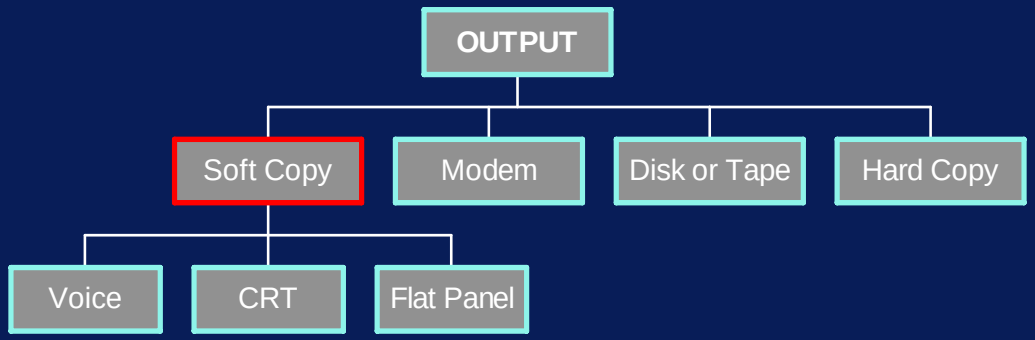
- Input
- Output
- Storage
- Processor



Output Systems

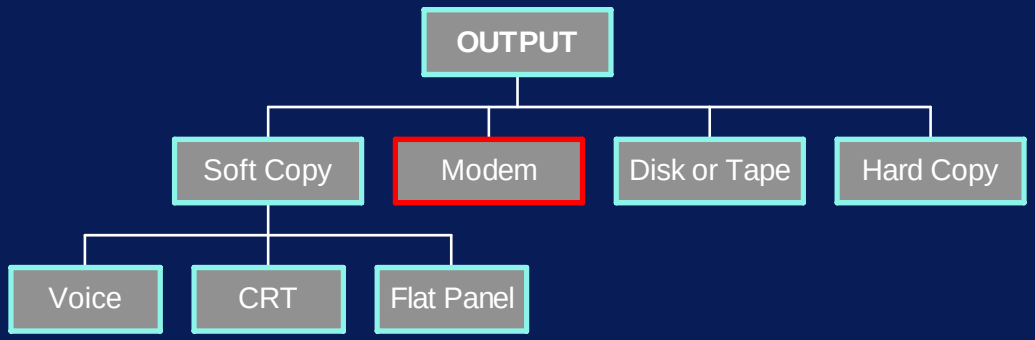


Output Systems

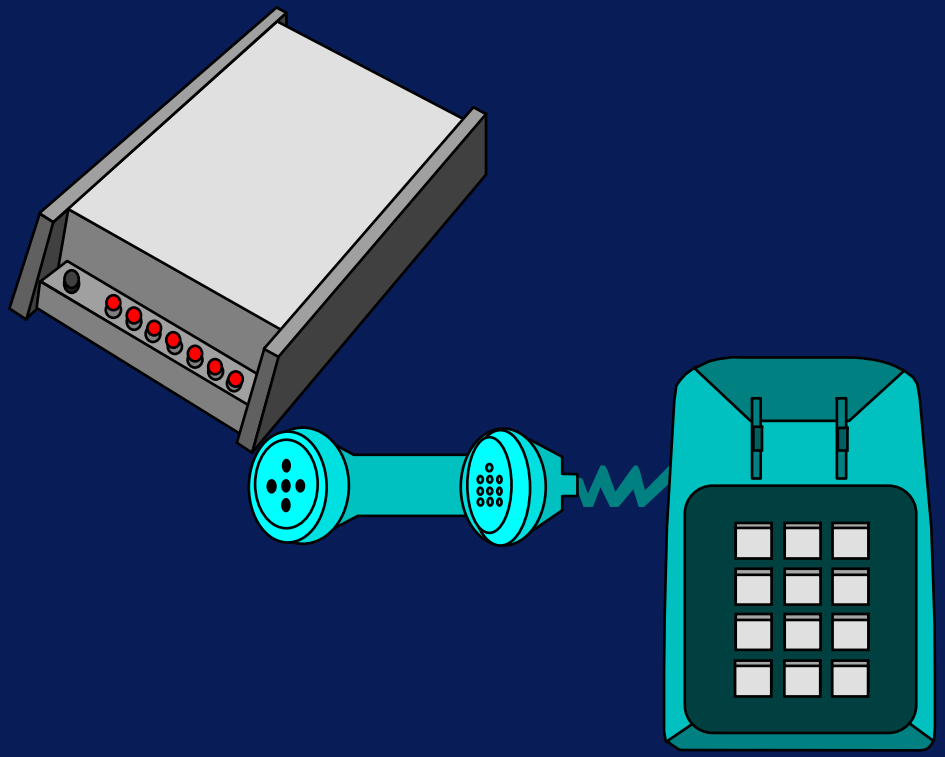


- Soft Copy
  - » Voice synthesis
  - » Music
  - » CRT (Cathode Ray Tube)
  - » LCD (Liquid Crystal Display)

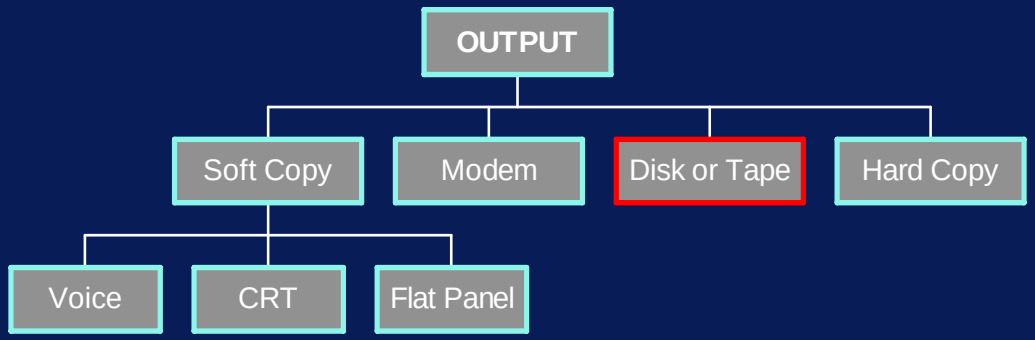
Output Systems



- Modems
  - Modulator-Demodulator
  - Allows computers to communicate over telephone lines

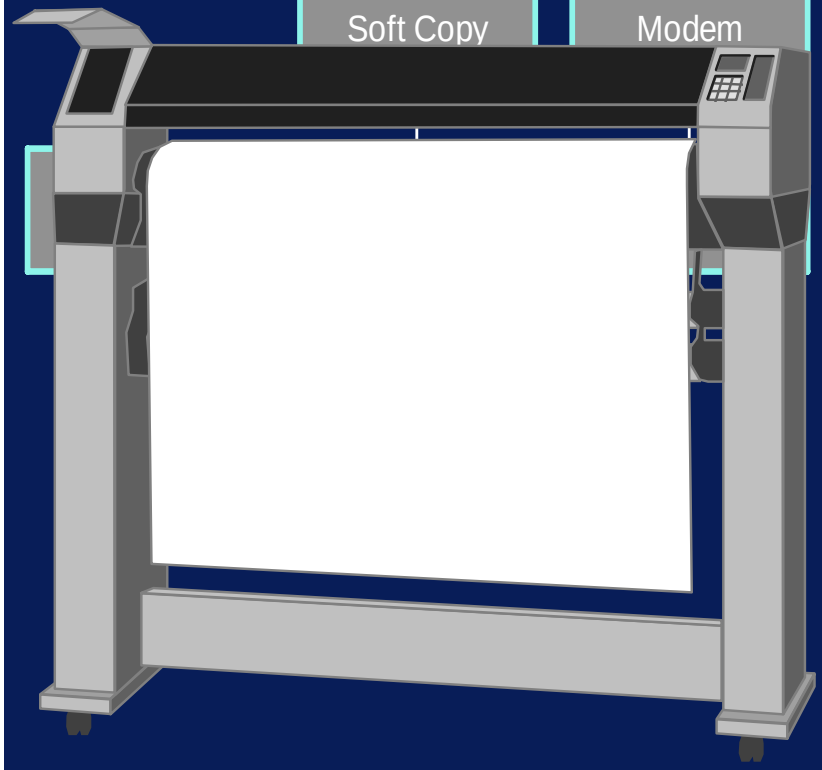


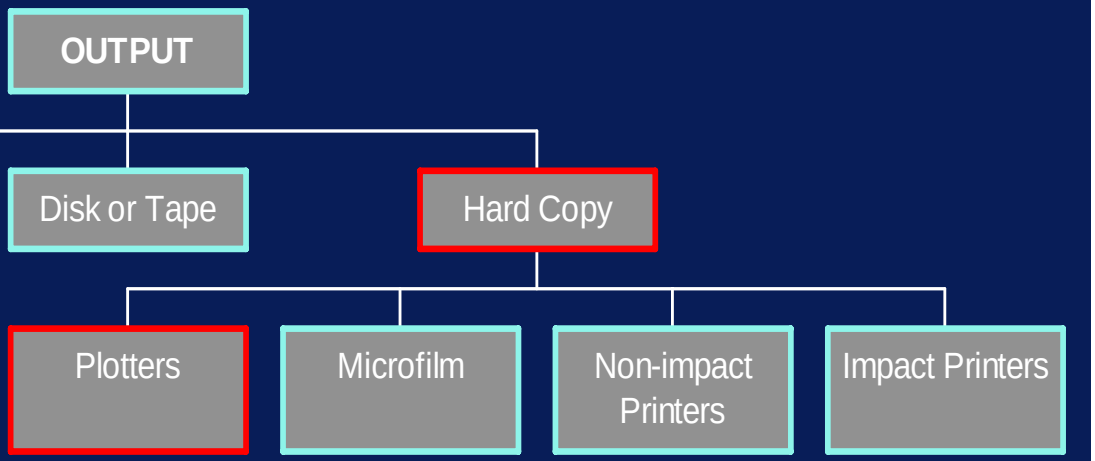
Output Systems



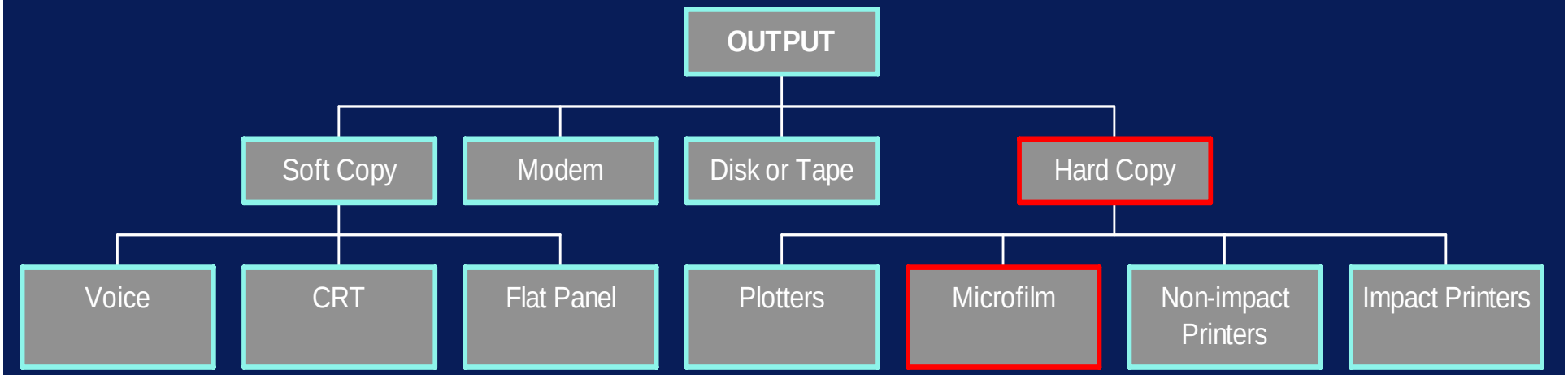
- Disks
  - Magnetic
    - » Floppy
    - » Hard disk
  - Optical
- Storage Devices

Output Systems

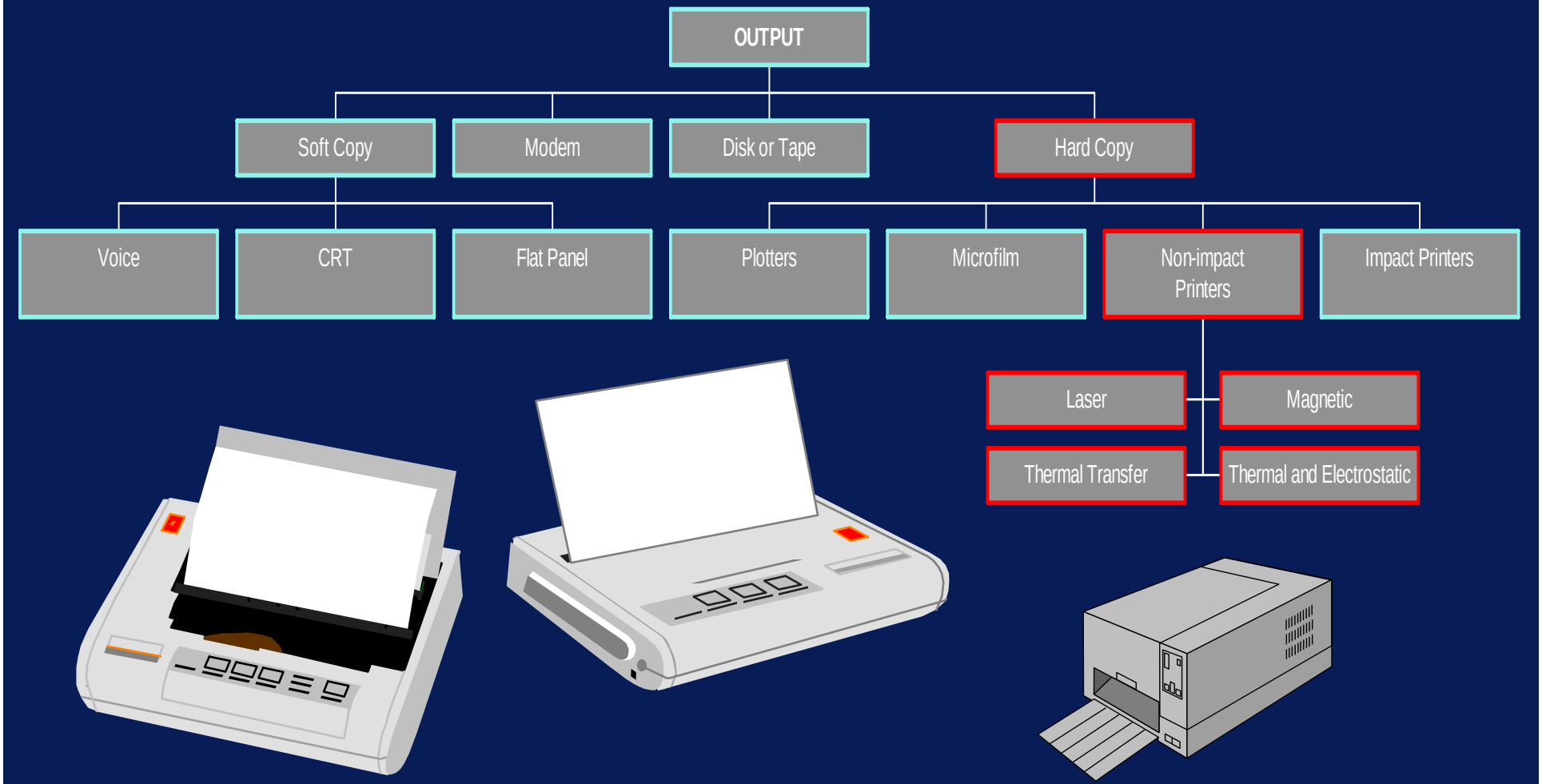




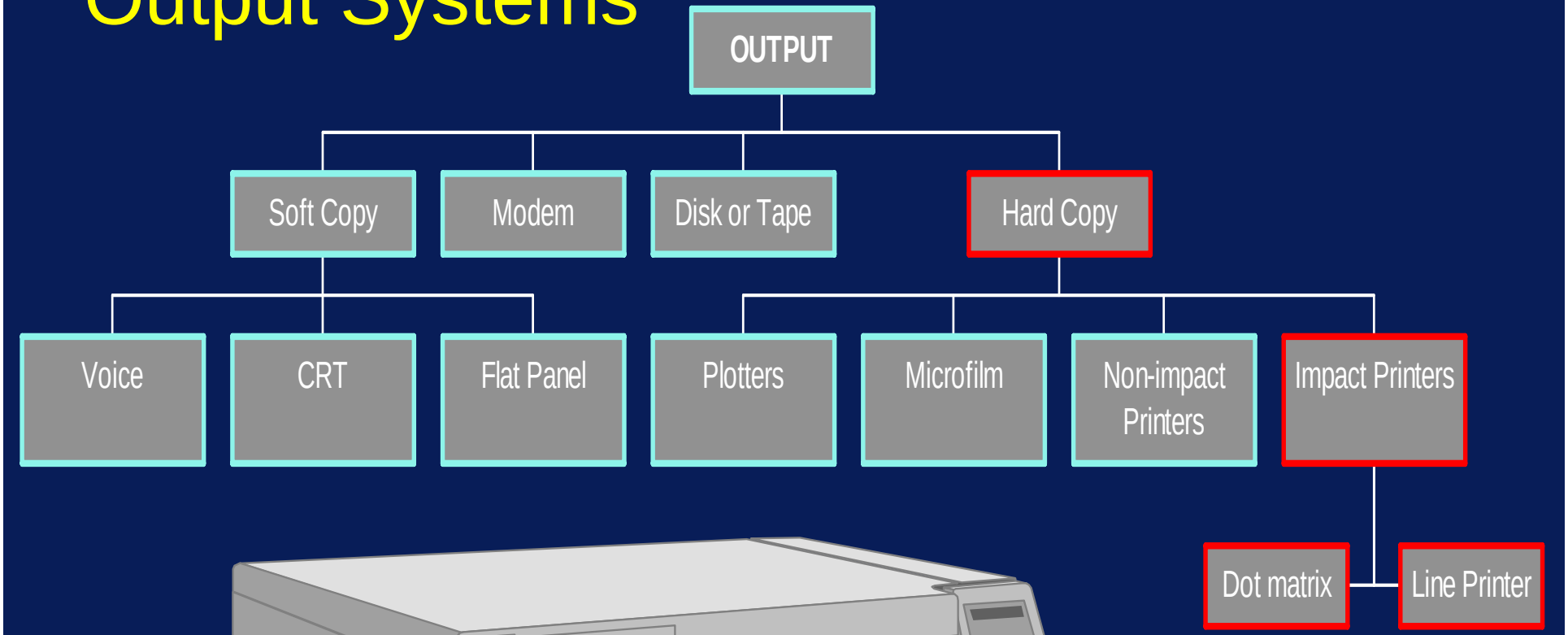
Output Systems

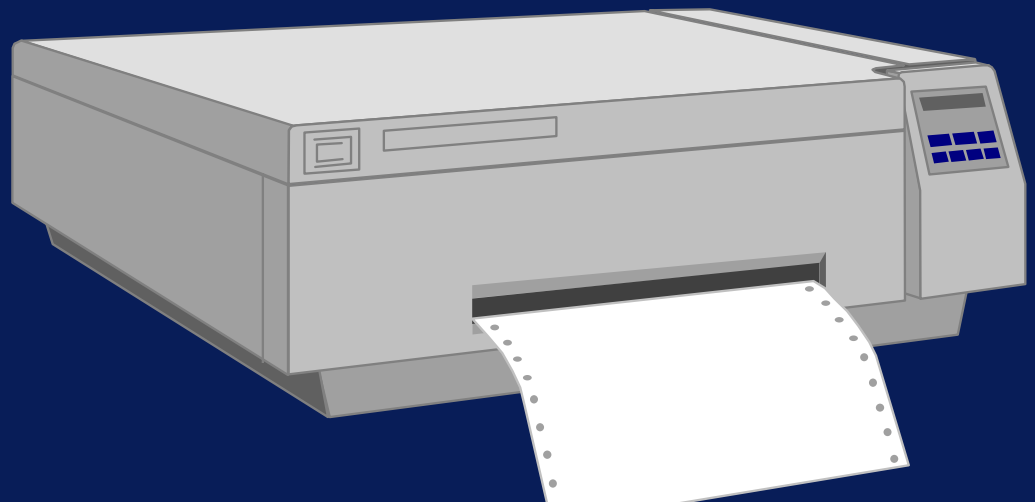


Output Systems



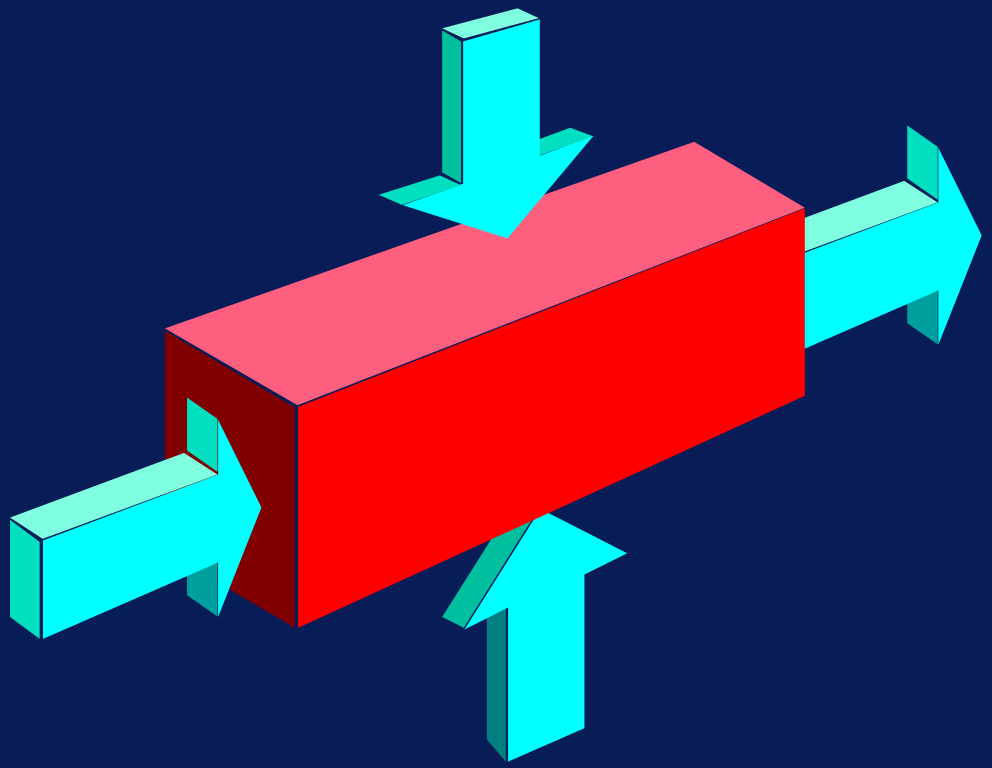
Output Systems



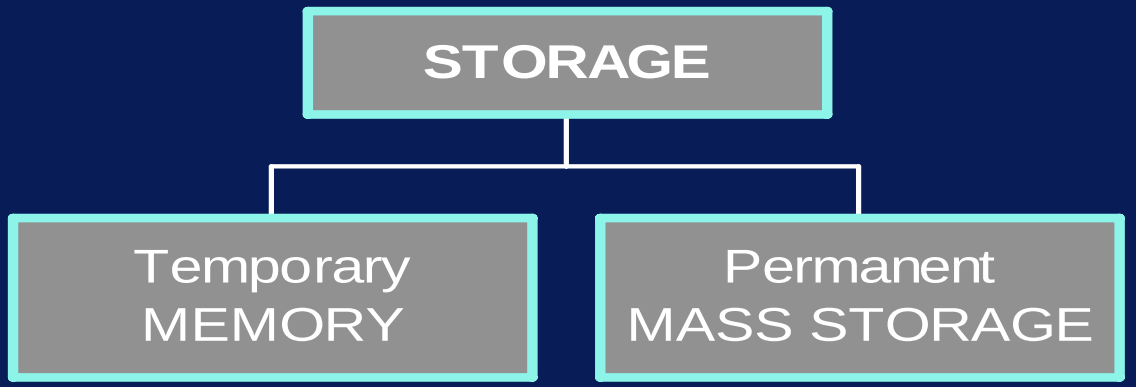


Key Components

- Input
- Output
- Storage
- Processor

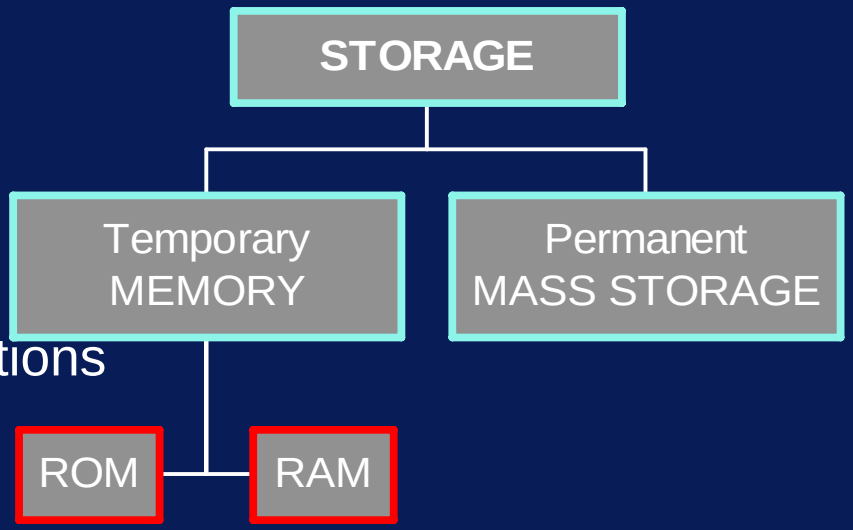


Storage Systems



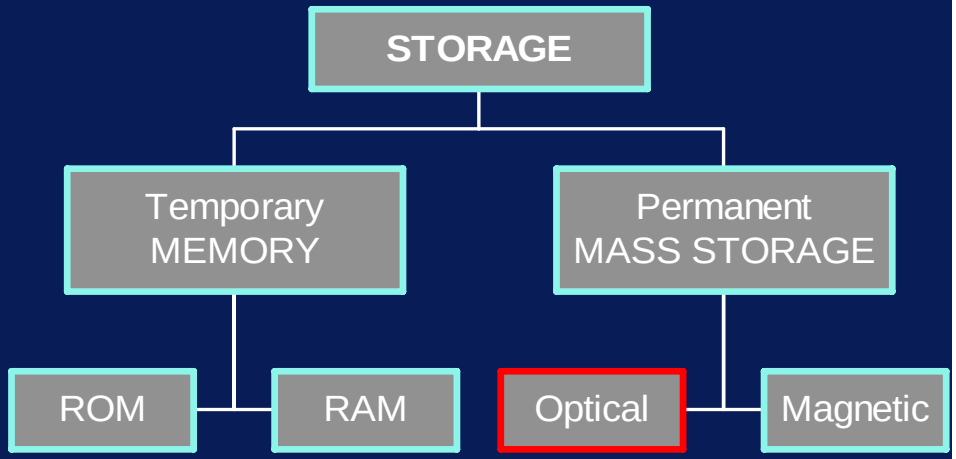
- Units of Storage
  - 1 bit
  - 8 bits = 1 byte
  - 1kbyte = 2^{10} = 1024 bytes
  - 1Mbyte = 2^{20} = 1048576 bytes

Storage Systems

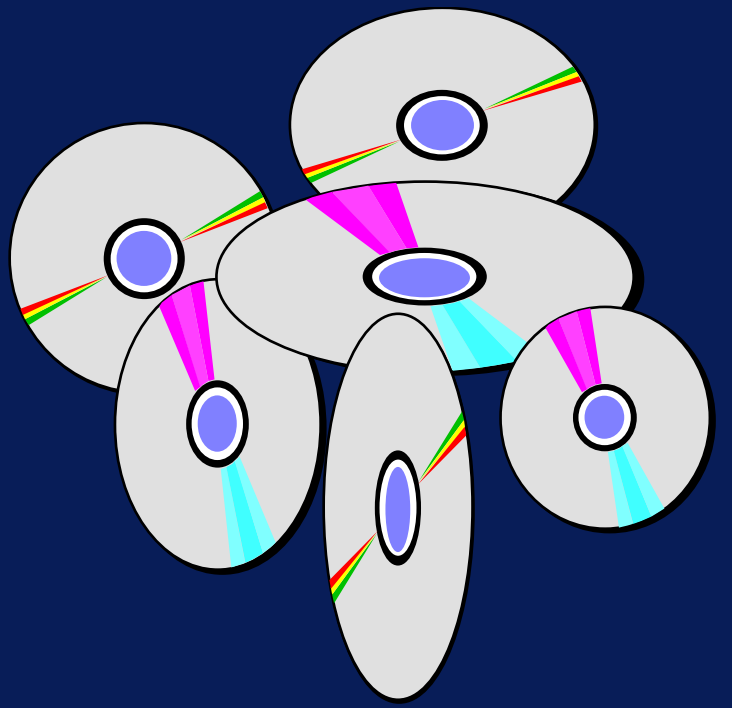


- Memory
  - Stores the bits and bytes (instructions and data)
  - ROM - Read Only Memory
    - » Non-volatile
    - » Won't disappear when power is off
  - RAM - Random Access Memory
    - » Read/Write Memory
    - » Volatile
    - » SIMMs (Single Inline Memory Modules):
      4 Mbytes in a stick of chewing gum

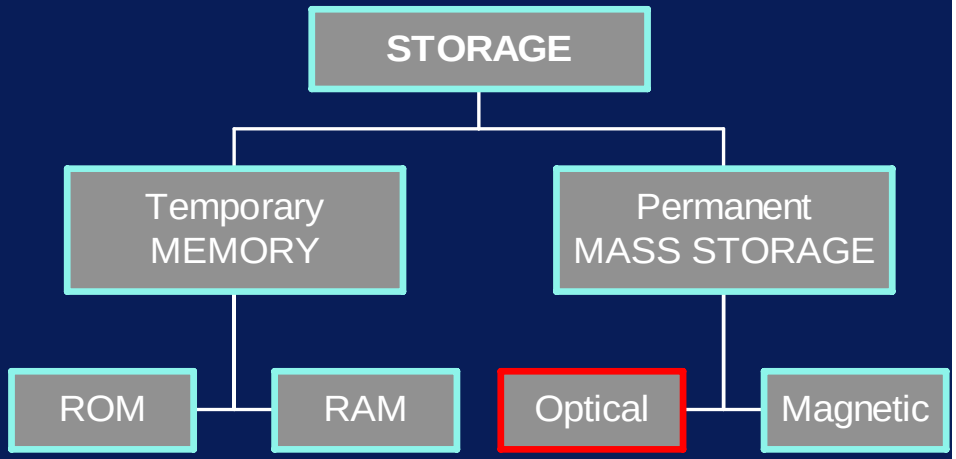
Storage Systems



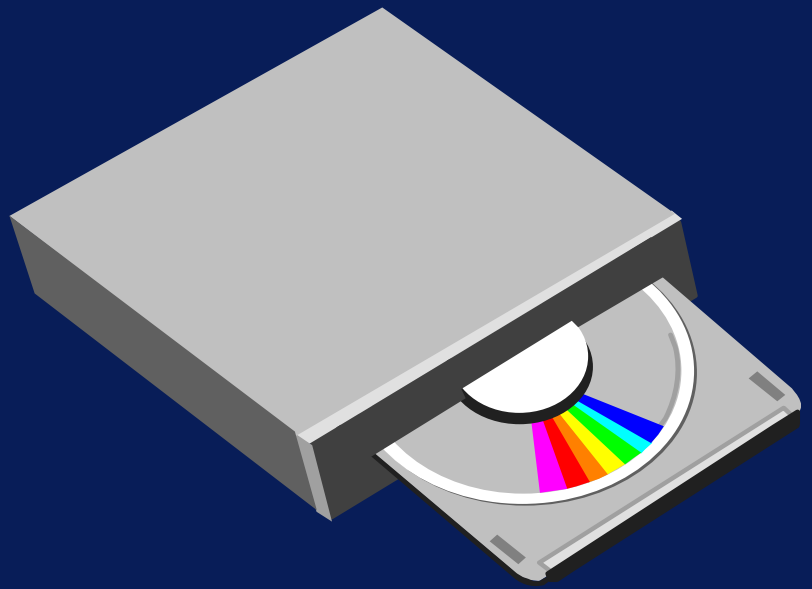
- Optical Disks
  - 15,000 tracks per inch
  - Digital code read by laser
  - 650 Mbytes in a 4.75” plastic platter
  - CD ROM; WORM; Erasable Disks



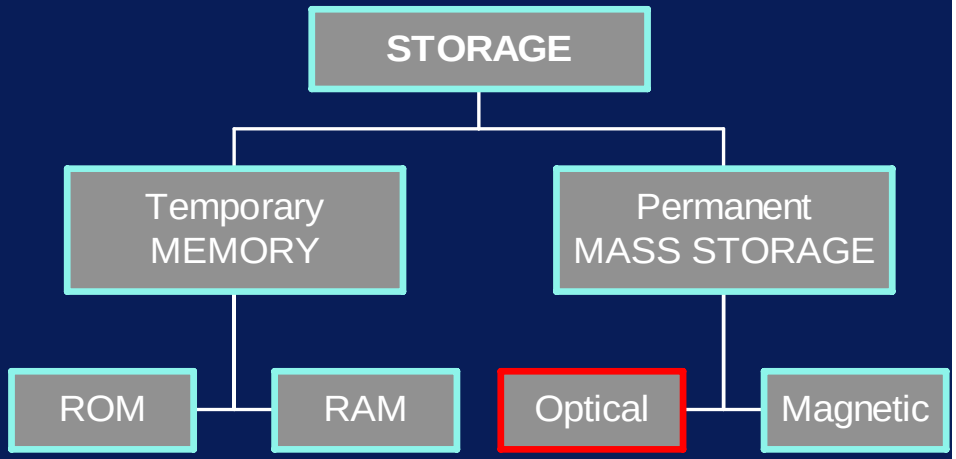
Storage Systems



- CD ROM
  - CD Read Only Memory
  - 12cm optical disk
  - Capable of storing 72 minutes of VHS quality video using MPEG compression

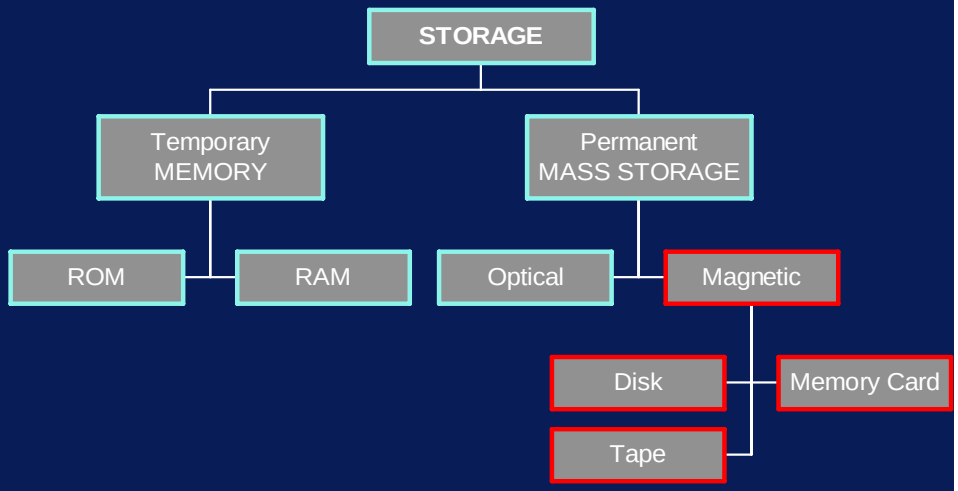


Storage Systems

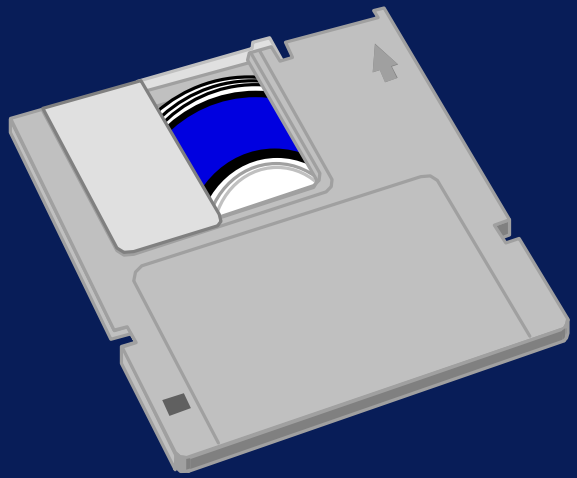


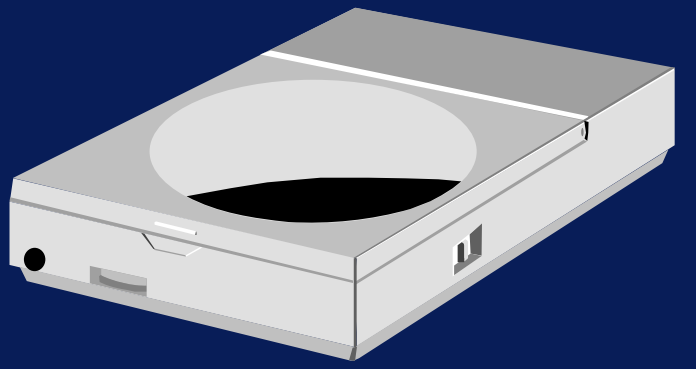
- Write-One Read_Mostly CDs (WORMS)
  - Powerful laser burns in the digital code
  - Not erasable
  - Lowe power laser reads the digital pattern
- Eraseable CD
  - Lasers read and write inofrmation
  - Also use a magnetic material
  - To write: a laser beam heats a tiny spot and a magnetic field is applied to reverse the magnetic polarity

Storage Systems

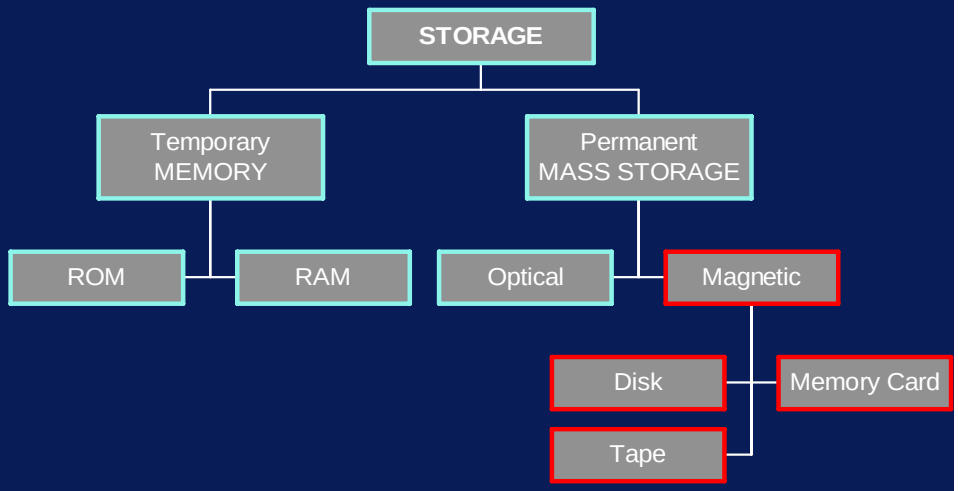


- Magnetic Disk
  - A circular platter coated with magnetic material
- Floppy Disk
  - 3.5”; 360kbyte to 2.88Mbytes (1.44 is common)
- Hard Disk
  - 1.3”, 1.8”, 2.5”, 3.5”, 5.25”; 120Mbytes to over 6 Gigabyte (6 Gbyte)

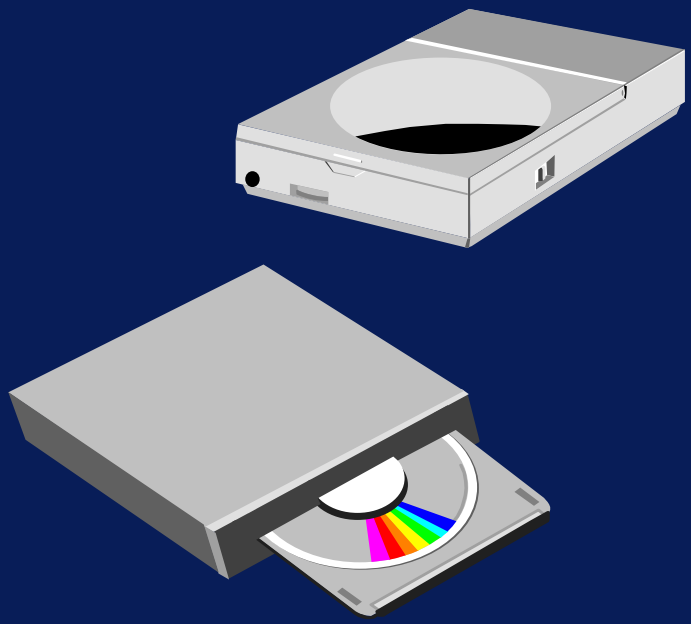




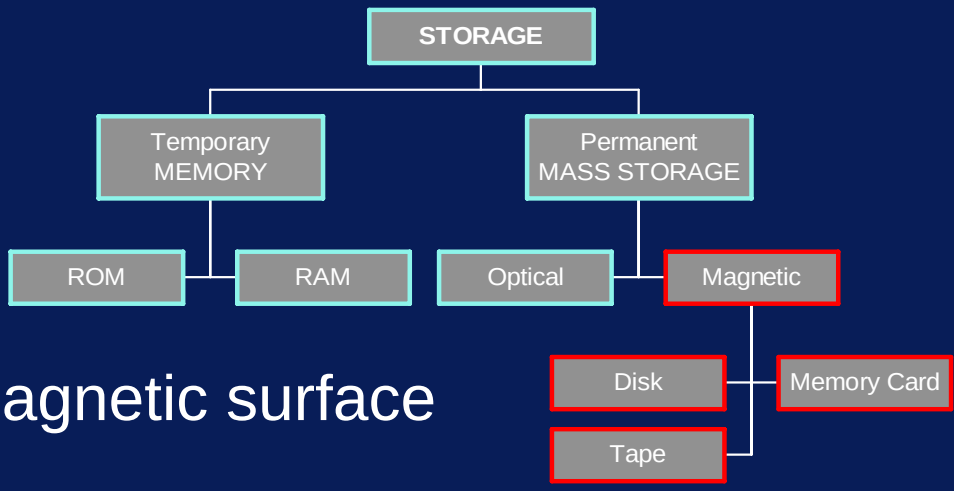
Storage Systems



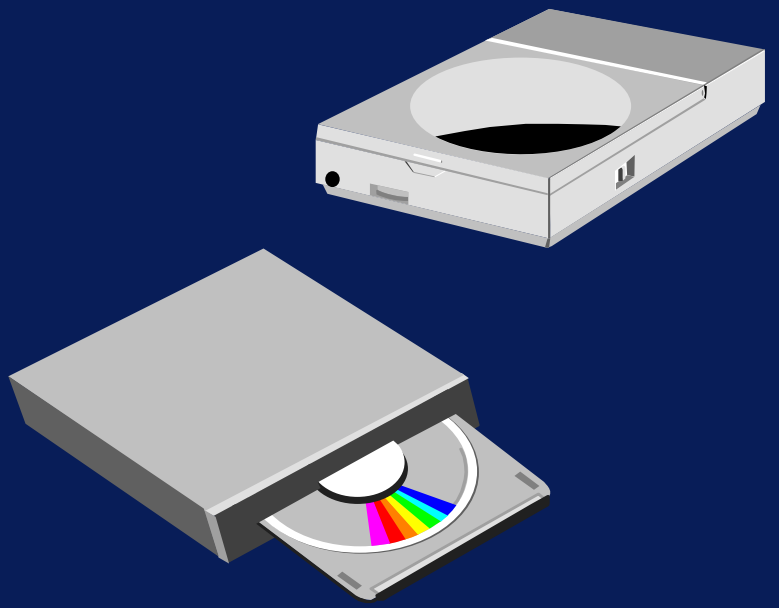
- 40 Gbyte hard disk
  - 20,000,000 pages of text
- 650 Mbyte CD
  - 325,000 pages of text
- 17 Gbyte DVD
  - 8,500,000 pages of text



Storage Systems



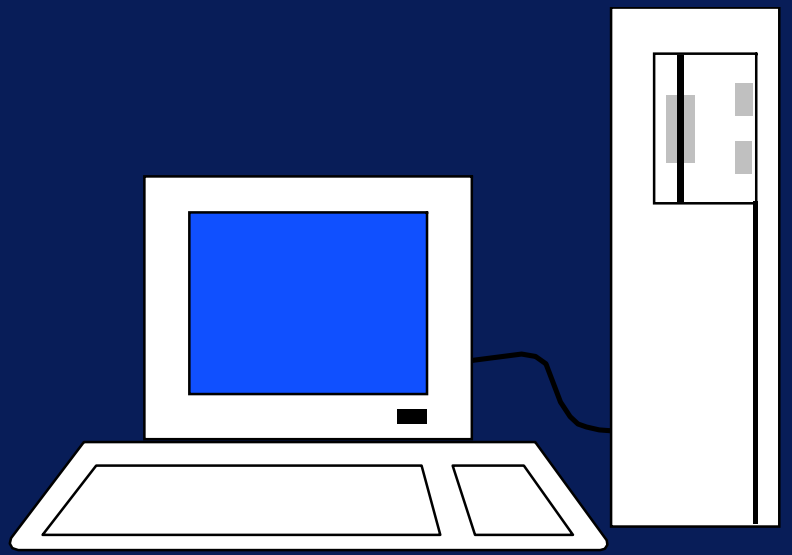
- Height of Read Head above magnetic surface
  - 2 millionths of an inch
- Smoke Particle
  - 250 millionths of an inch
- Fingerprint
  - 620 millionths of an inch
- Dust particle
  - 1500 millionths of an inch
- Human hair
  - 3000 millionths of an inch

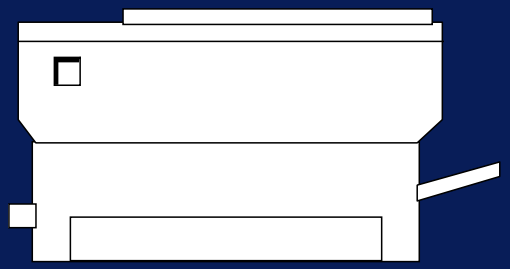


The Processor: Hardware & Software

Components of Computer Systems

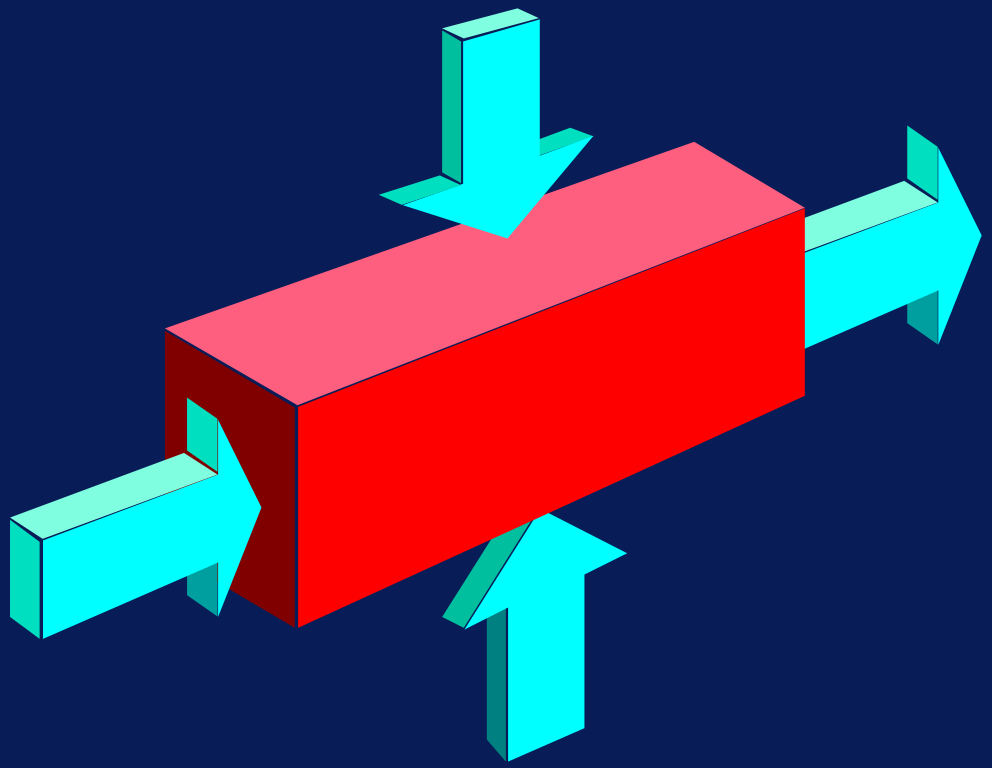
- Keyboard
- Display
- System Unit
- Storage
- Printer



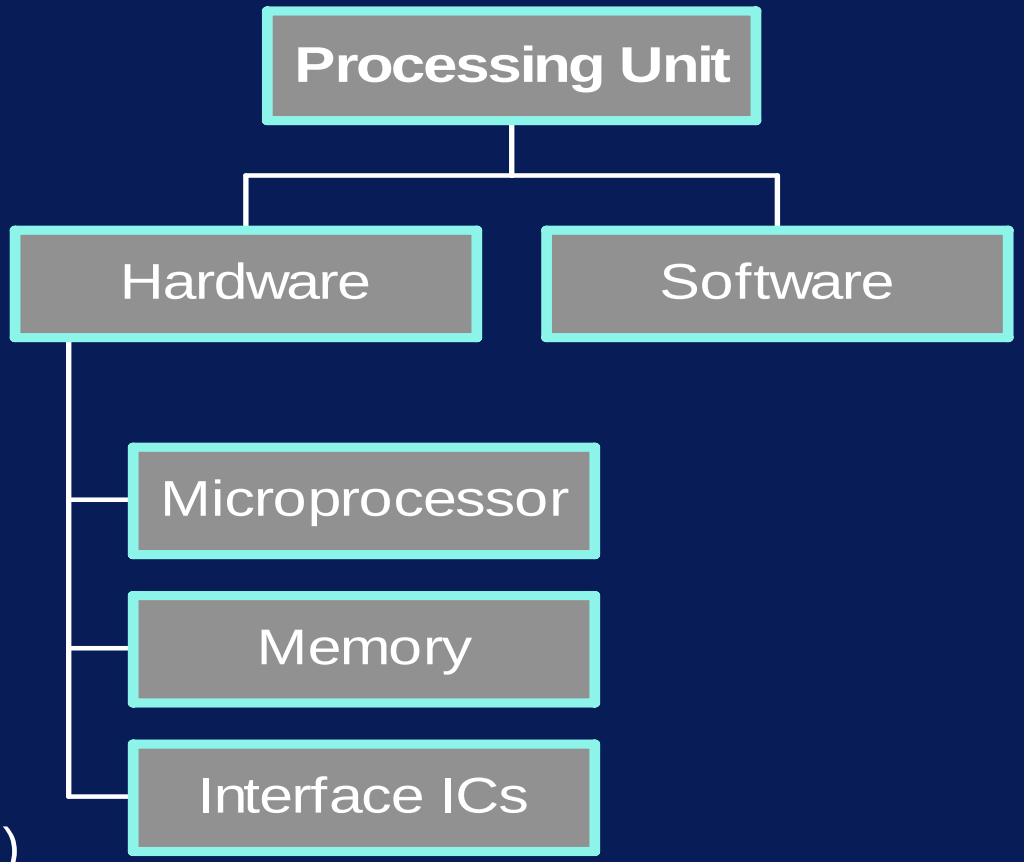


Key Components

- Input
- Output
- Storage
- Processor



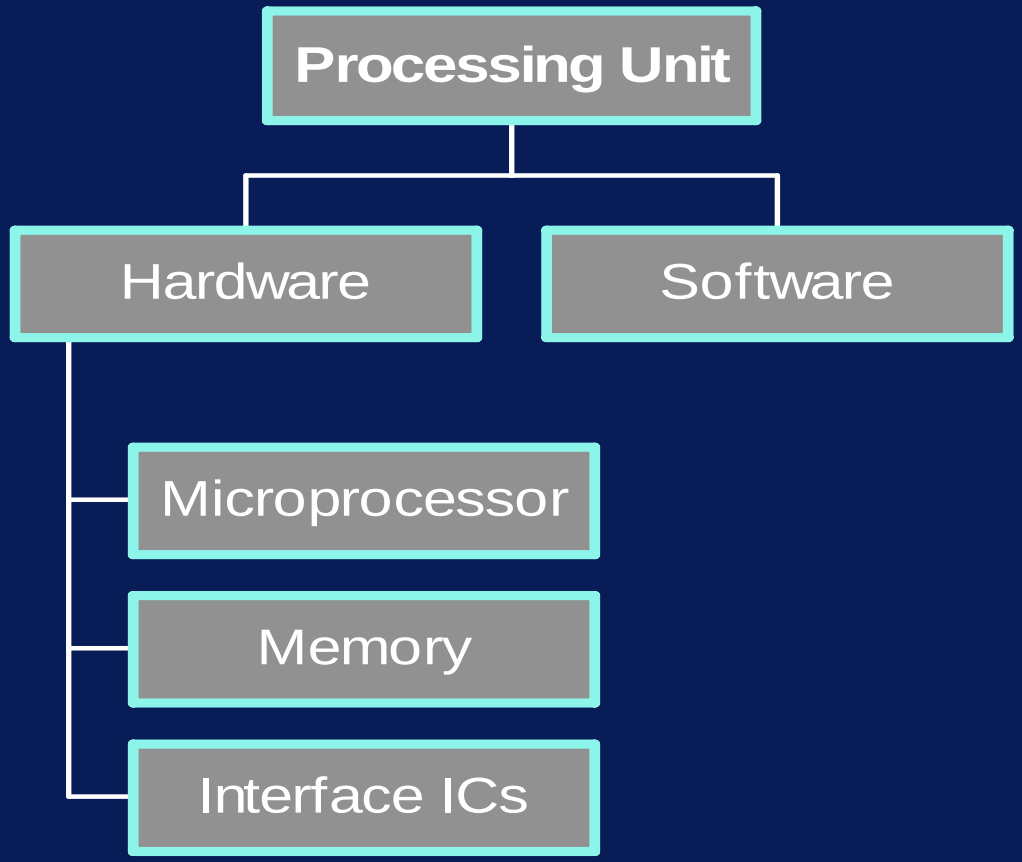
Hardware



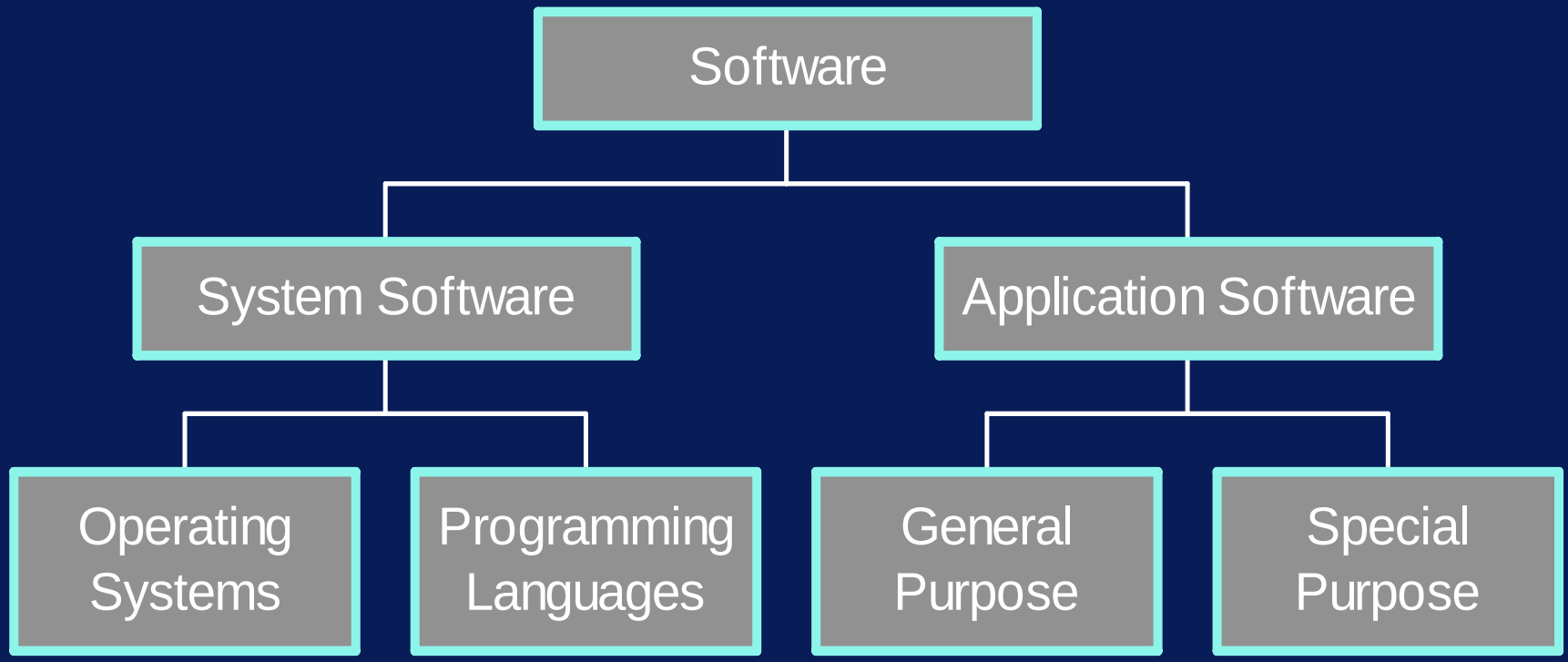
- Microprocessor
  - Effects computation
  - Intel 80486, 80586
  - Motorola 68040
  - PowerPC, MIPS, Alpha, Sparc
  - Clock speeds 50-600MHz (+)
- Memory
  - Storage
- Interface ICs
  - communication with other devices

Hardware

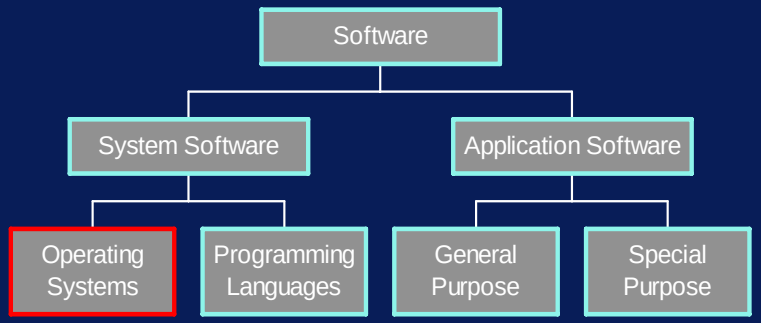
- Much more to come on the topic of hardware shortly



Software

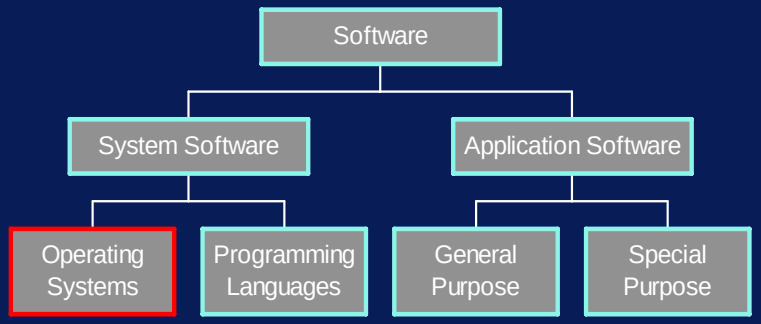


Operating Systems



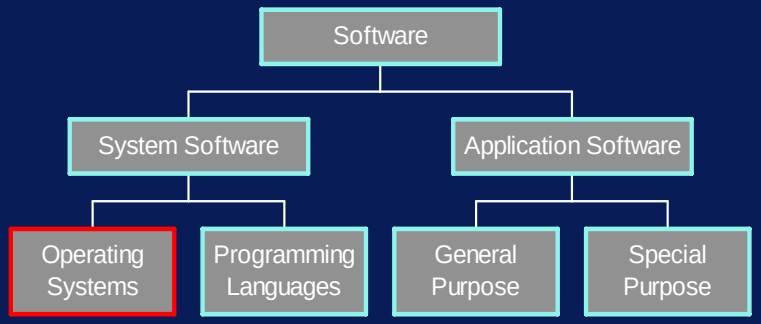
- User interfaces
  - Software which is responsible for passing information to and from the person using the program (the user)
  - Communicates with and controls the computer
  - Three types of user interface:
    - » Graphic user interfaces
    - » Menu driven interfaces
    - » Command driven interfaces

Operating Systems



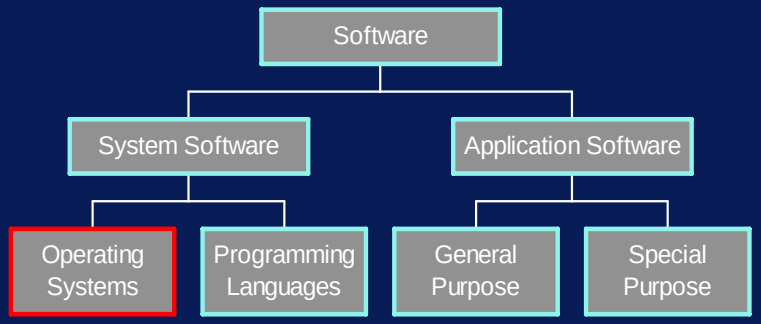
- Graphic User Interfaces (GUIs)
  - Pictures, graphic symbols (icons), to represent commands
  - Windows: a way of ‘looking in’ on several applications at once

Operating Systems



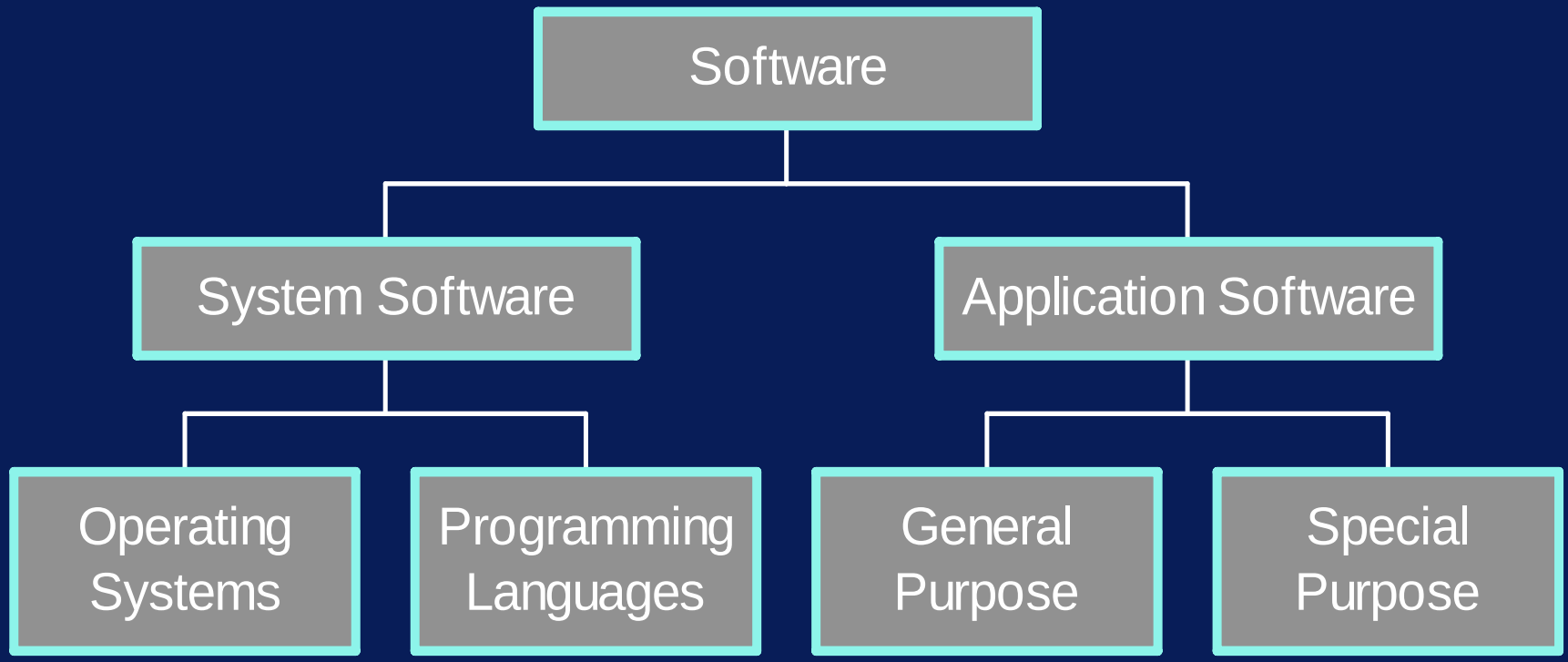
- Menu-driven interfaces
  - Menu bar
  - Pull-down menu for choices

Operating Systems



- Command-driven interfaces
  - A (system) prompt
  - User types in single letter, word, line which is translated into an instruction for the computer
  - For example: cp source destination
  - Need to be very familiar with the syntax (grammar) of the command language

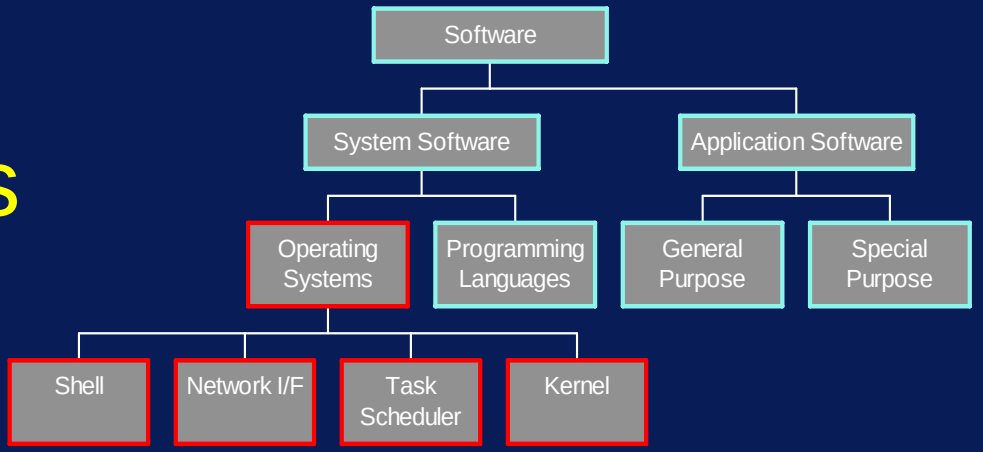
Software



Operating Systems

- Operating System is the software that manages the overall operation of the computer system
- Main purpose is to support application programs
- Hide details of devices from application programs

Operating Systems



- Shell (or user interface)
- Network interface:
  coordinate multiple tasks in a single computer
- Task scheduler
  coordination of multiple tasks in a single computer
- Kernel
  - Software which ties the hardware to the software, and
  - manages the flow of information to and from disks, printers, keyboards, ... all I/O devices

Operating Systems

- File Handling
  - Collection of information (stored on disk)
  - Disks need to be formatted to allow them to store information
  - OS manages location of files on disk
  - OS performs I/O to disk
  - OS checks and corrects errors on disk I/O

Operating Systems

- Device Drivers
  - Programs which handle the various hardware devices, e.g., mouse, keyboard, CD, video, etc.
  - For example, an application wants to print a document
    - » It call the operating system
    - » which sends the information to the device driver together with instructions
    - » and the printer driver handles all the control of the printer

Operating Systems

- Single tasking OS
  - Runs only one application at a time
- Multi-Tasking OS
  - More than one application can be active at any one time

Operating Systems

- DOS (Disk Operating System)
  - Single-tasking
  - Command-driven
  - Huge number of applications written for DOS
  - Does not require powerful computer
  - No network services
  - No multimedia extensions
  - Designed for the Intel 80x86 processor

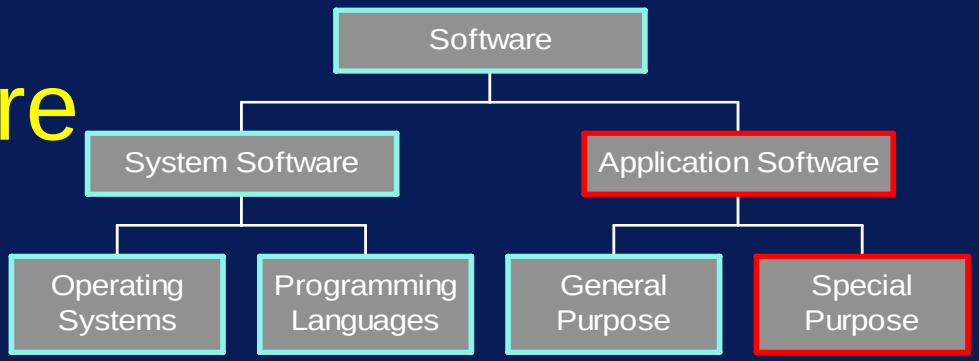
Operating Systems

- Windows
  - GUI
  - Can run DOS programs
  - Has network services
  - Has multimedia extensions
  - Requires large amounts of memory, disk space, powerful processor
  - Designed for the Intel 80X86 processors

Operating Systems

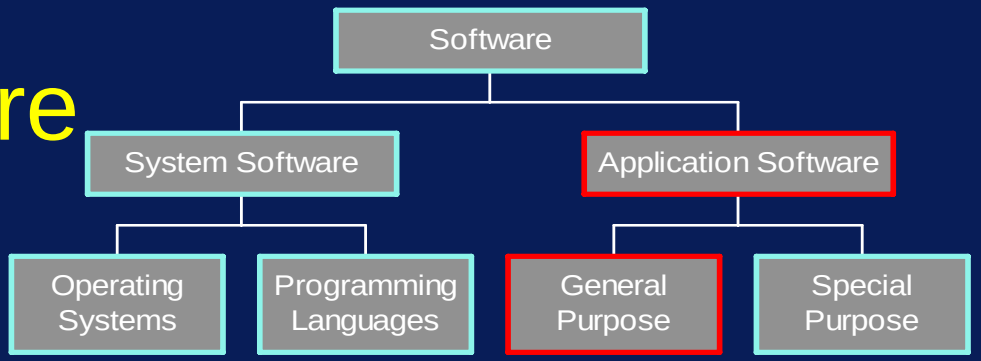
- Macintosh OS
  - Multi-tasking
  - GUI called finder
  - Very easy to use
  - Very graphically oriented
  - Has network services
  - Has multimedia extensions
  - Designed for the Motorola and PowerPC processors

Application Software



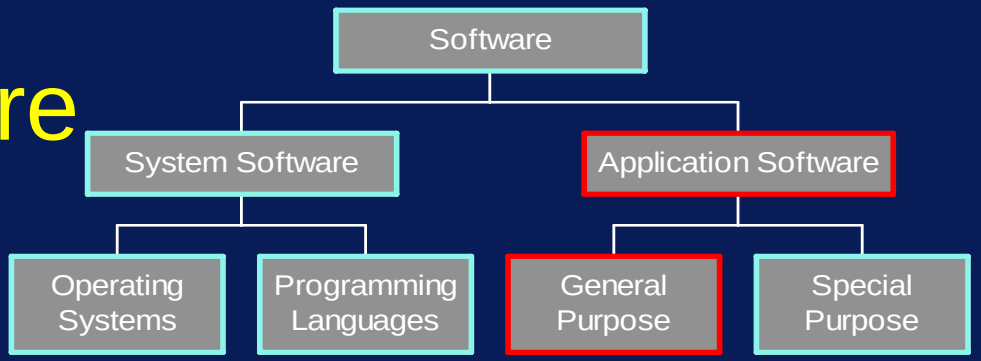
- Special Purpose
  - Payroll
  - Accounting
  - Book-Keeping
  - Entertainment
  - Statistical Analysis

Application Software



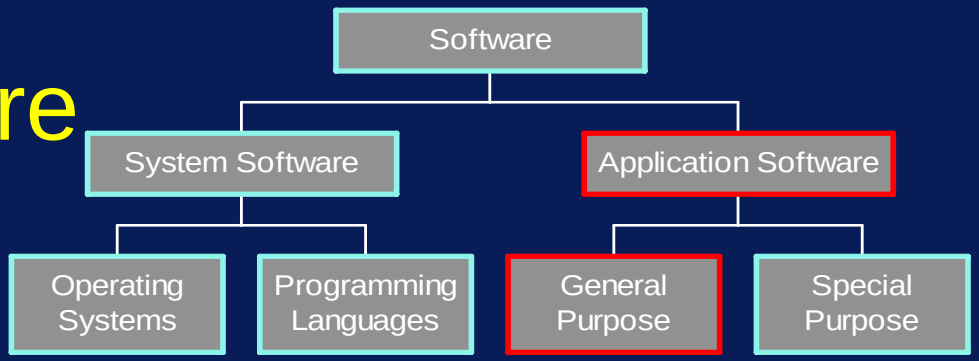
- General Purpose
  - Word Processing (e.g. MS Word)
  - Desktop Publishing (e.g. Quark Xpress)
  - Spreadsheets (e.g. MS Excel)
  - Databases (e.g. MS Access)
  - Graphics (e.g. MS Powerpoint)
  - E-mail (e.g. MS Mail)
  - Internet Browsers (e.g. Firefox, Explorer)

Application Software



- Integrated Software
  - Goal: effective sharing of information between all applications
  - For example: MS Office: Excel, Word, Powerpoint, Access can all use each other's data directly

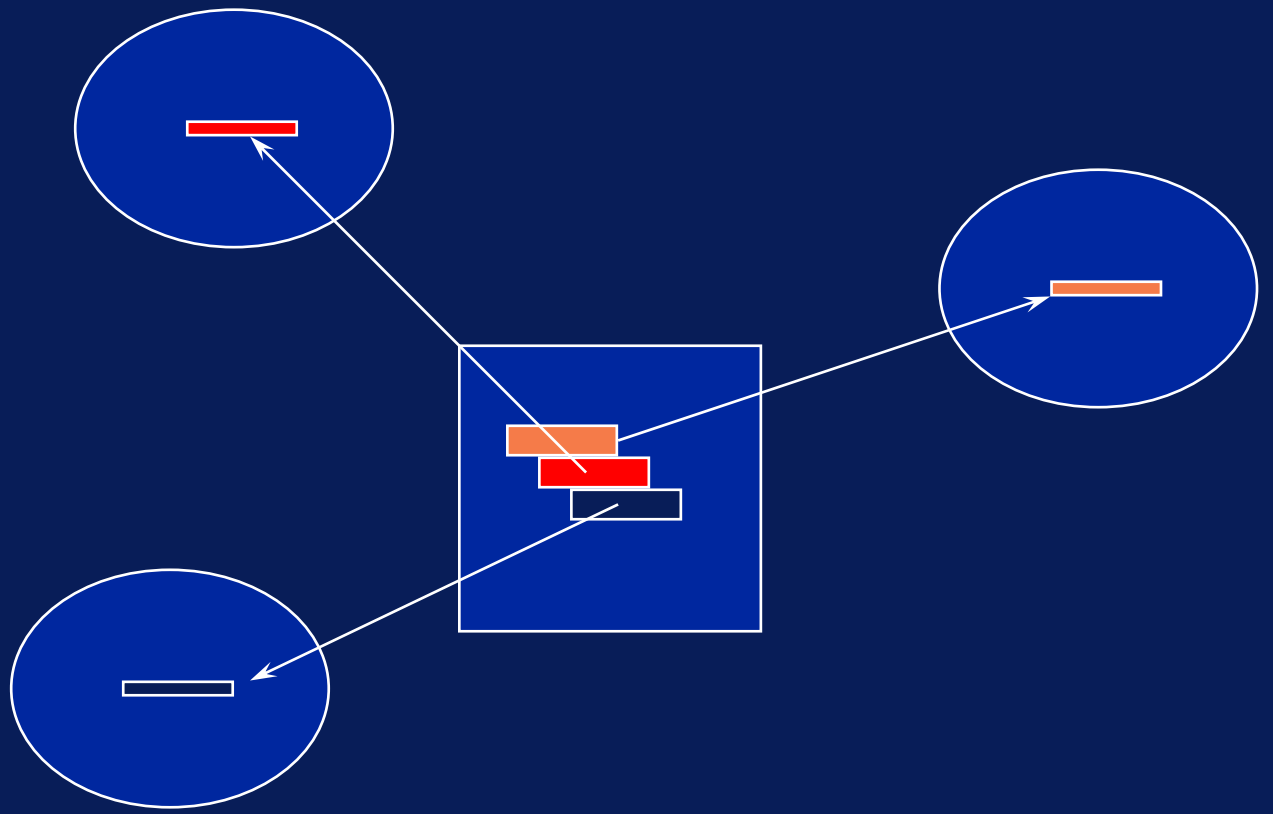
Application Software



- Integrated Software
  - Object Linking & Embedding (OLE)
  - Information is stored in one location only
  - Reference is made to it from another application
  - This reference is known as a link
  - Don’t actually make a copy (cf. hypertext, multimedia, WWW)

Application Software

- Object Linking & Embedding (OLE)



Operation of Processor and Memory

The Processor

- The processor is a functional unit that interprets and carries out instructions
- Also called a Central Processing Unit (CPU)
- Every processor has a unique set of operations
- LOAD
- ADD
- STORE

The Processor

- This set of operation is called the instruction set
- Also referred to as machine instructions
- The binary language in which they are written is called machine language

The Processor

- An instruction comprises
  - operator (specifies function)
  - operands - (data to be operated on)
- For example, the ADD operator requires two operands
  - Must know WHAT the two numbers are
  - Must know WHERE the two numbers are

The Processor

- If the data (e.g. the number to be added) is in memory, then the operand is called an address
- ADD *num1 num2*
- *num1* could be a number or it could be the address of a number in memory (i.e. where the number is stored)

Machine Language Instruction Set

Category	Example
Arithmetic	Add, subtract, multiply, divide
Logic	And, or, not, exclusive or
Program Control	branching, subroutines
Data Movement	Move, load, store
Input/Output	Read, Write

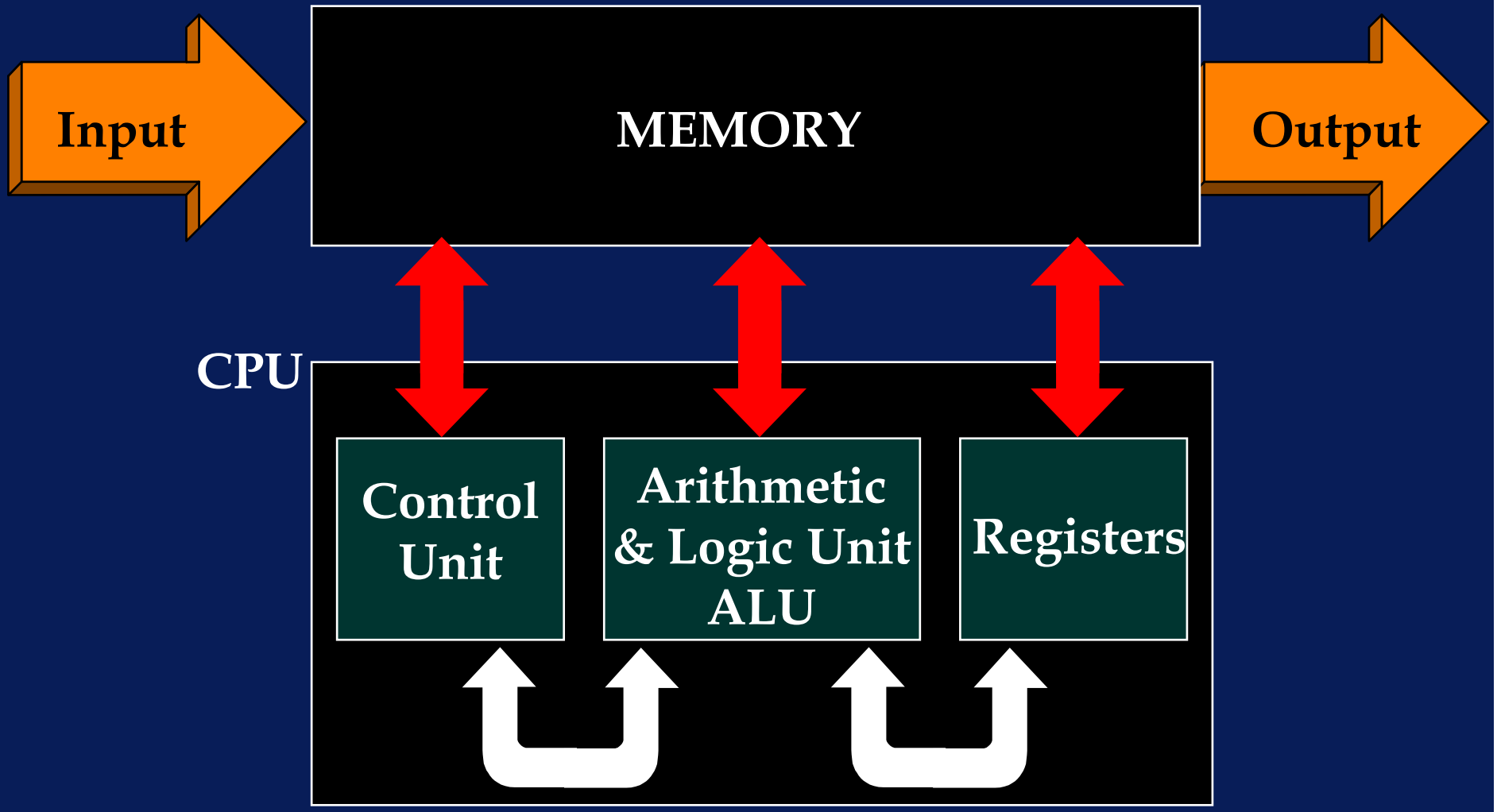
The Processor

- The processor's job is to
  - retrieve instructions from memory
  - retrieve data (operands) from memory
  - perform the operation
  - (maybe store the result in memory)
  - retrieve the next instruction ....

The Processor

- This step-by-step operation is repeated over and over at speeds measured in millionths of a second
- The CLOCK governs the speed: each step must wait until the clock ‘ticks’ to begin
- a 300 MHz processor will use a clock which ticks 300 000 000 times a second

The Components of a Processor



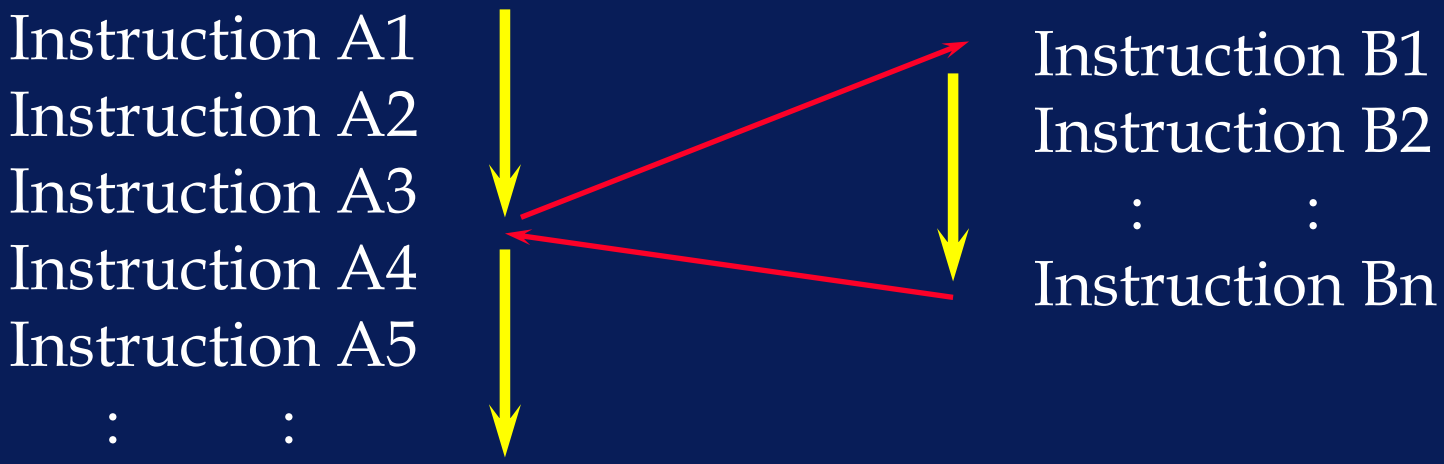
The Control Unit

- Supervises the operation of the processor
- Makes connections between the various components
- Invokes the operation of each component
- Can be interrupted!

The Control Unit

- An interrupt
  - is a signal
  - which tells the control unit to suspend execution of its present sequence of instructions (A)
  - and to transfer to another sequence (B)
  - resuming the original sequence (A) when finished with (B)

An Interrupt



EXECUTE instructions

BRANCH to new set of instructions

The Control Unit

- Receives instructions from memory
- Decodes them (determines their type)
- Breaks each instruction into a sequence of individual actions
  (more on this later)
- And, in so doing, controls the operation of the computer.

The Arithmetic & Logic Unit

- ALU
- Provided the computer with its computational capabilities
- Data are brought to the ALU by the control unit
- ALU performs the required operation

The Arithmetic & Logic Unit

- Arithmetic operations
  - addition, subtraction, multiplication, division
- Logic operations
  - make a comparison (CMP a, b)
  - and take action as a result (BEQ same)

Registers

- Register: a storage location inside the processor
- Control unit registers:
  - current instruction
  - location of next instruction to be executed
  - operands of the instruction
- ALU registers:
  - store data items
  - store results

Registers

- Word size (or word length)
  - Size of the operand register
  - Also used to describe the size of the pathways to and from the processor and between the components of the processor
- 16 to 64 bits word lengths are common
- 32 bit processor ... the operand registers of a processor are 32 bits wide (long!)

Specialized Processors

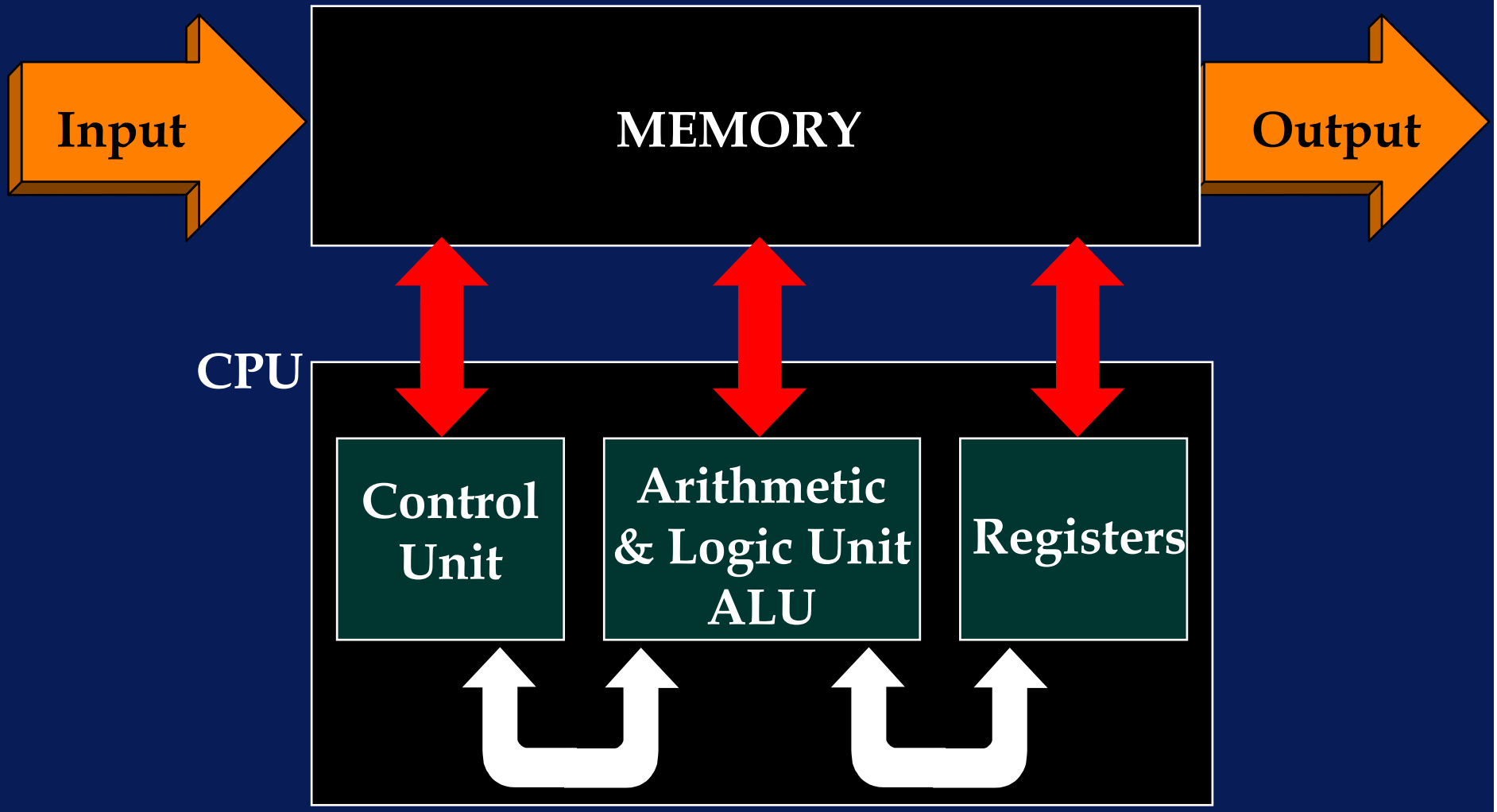
- DSP - Digital Signal Processors
  - Image processing; sound, speech
- Math co-processors
  - Real number arithmetic
- ASICs - Application-Specific Integrated Circuits
  - Microwave contoller
  - Engine management controller

The Operation of the Processor

A Simple Accumulator-Based CPU

(Von Neumann Computer)

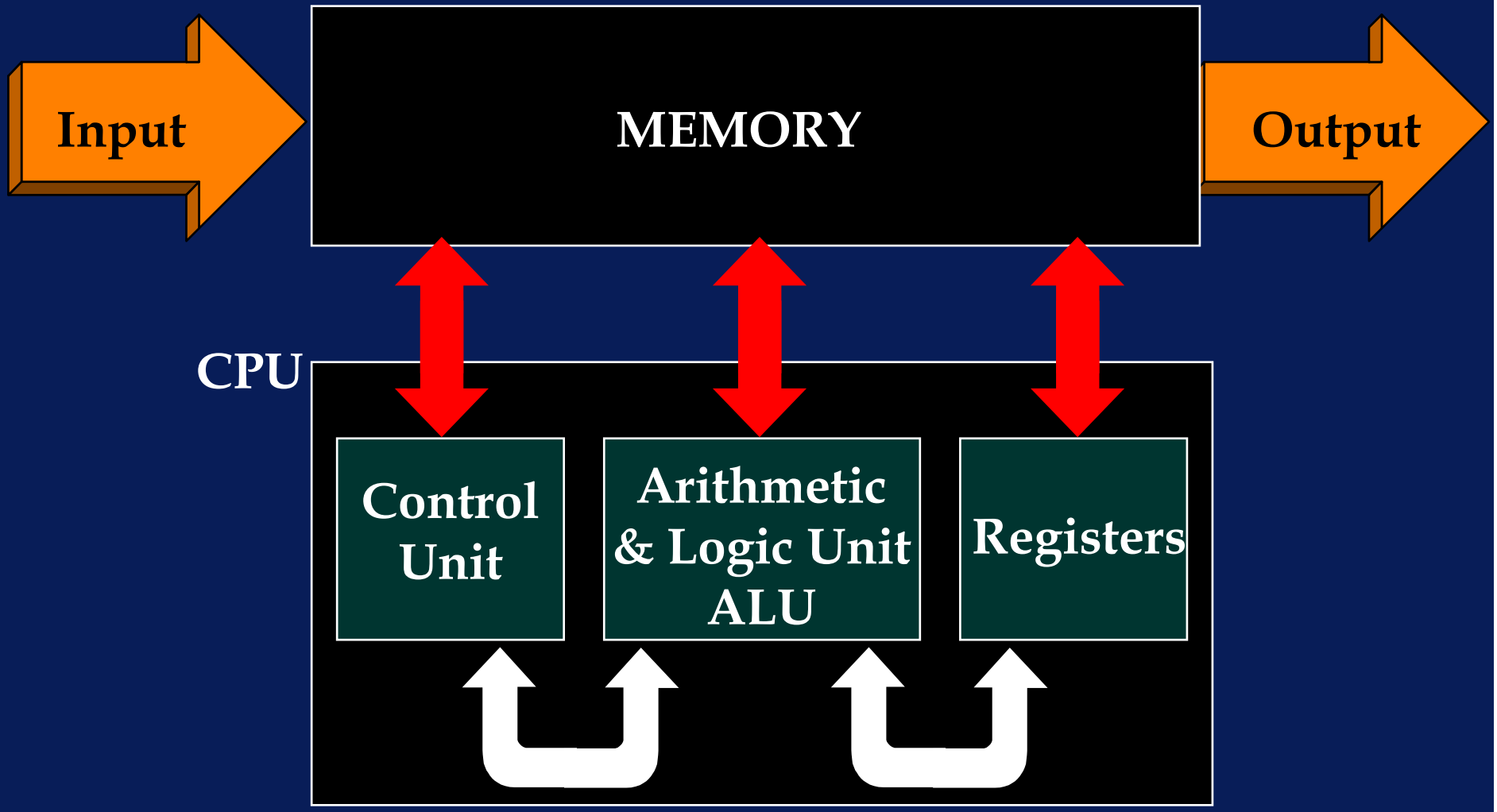
The Components of a Processor



Main Components

- (Program) Control Unit - PCU
  - Coordinates all other units in the computer
  - Organizes movement of data from/to I/O, memory, registers.
  - Directs ALU, specifically to indicate the operations to be performed
  - The control unit operates according to the stored program, receiving and executing its instructions one at a time

The Components of a Processor



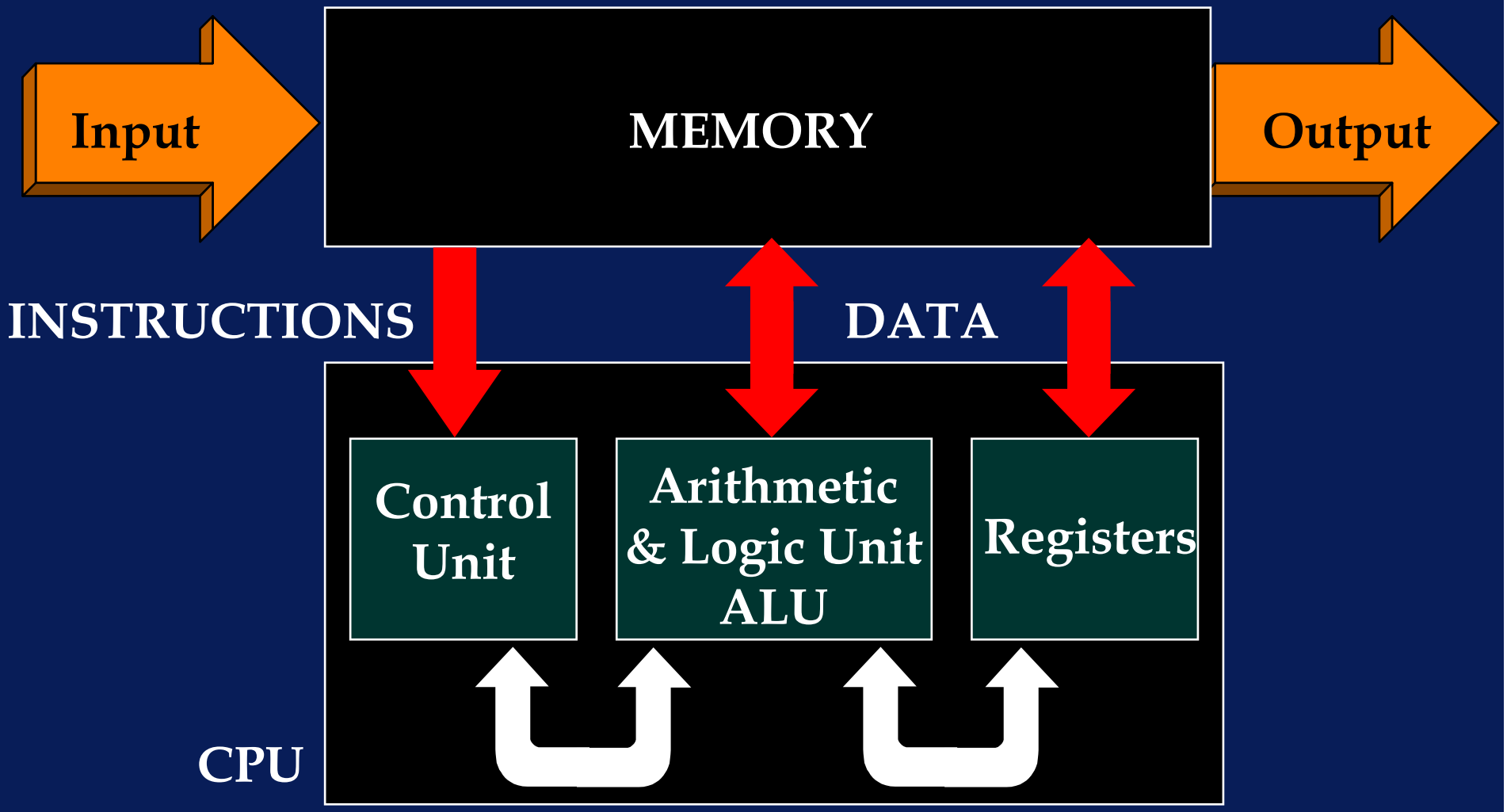
Main Components

- Arithmetic & Logic Unit - ALU
  - All computations are performed in this unit
  - ALU comprises adders, counters, and registers
  - Numerical operations (+ - / x)
  - Logical operations (AND, OR, program branching)

Main Components

- Register
  - Capable of receiving data, holding it, and transferring it as directed by the control unit
- Adder
  - Receives data from two or more sources, performs the arithmetic, and sends the results to a register
- Counter
  - Counts the number of times an operation is performed

The Components of a Processor



Some Key Points

- Instructions are coded as a sequence of binary digits
- Data are coded as a sequence of binary digits
- Registers are simply physical devices which allows these codes to be stored
- Memory is just the same

The Operation of a Processor

- How does a computer evaluate a simple assignment statement?
- For example:

  A := B + C

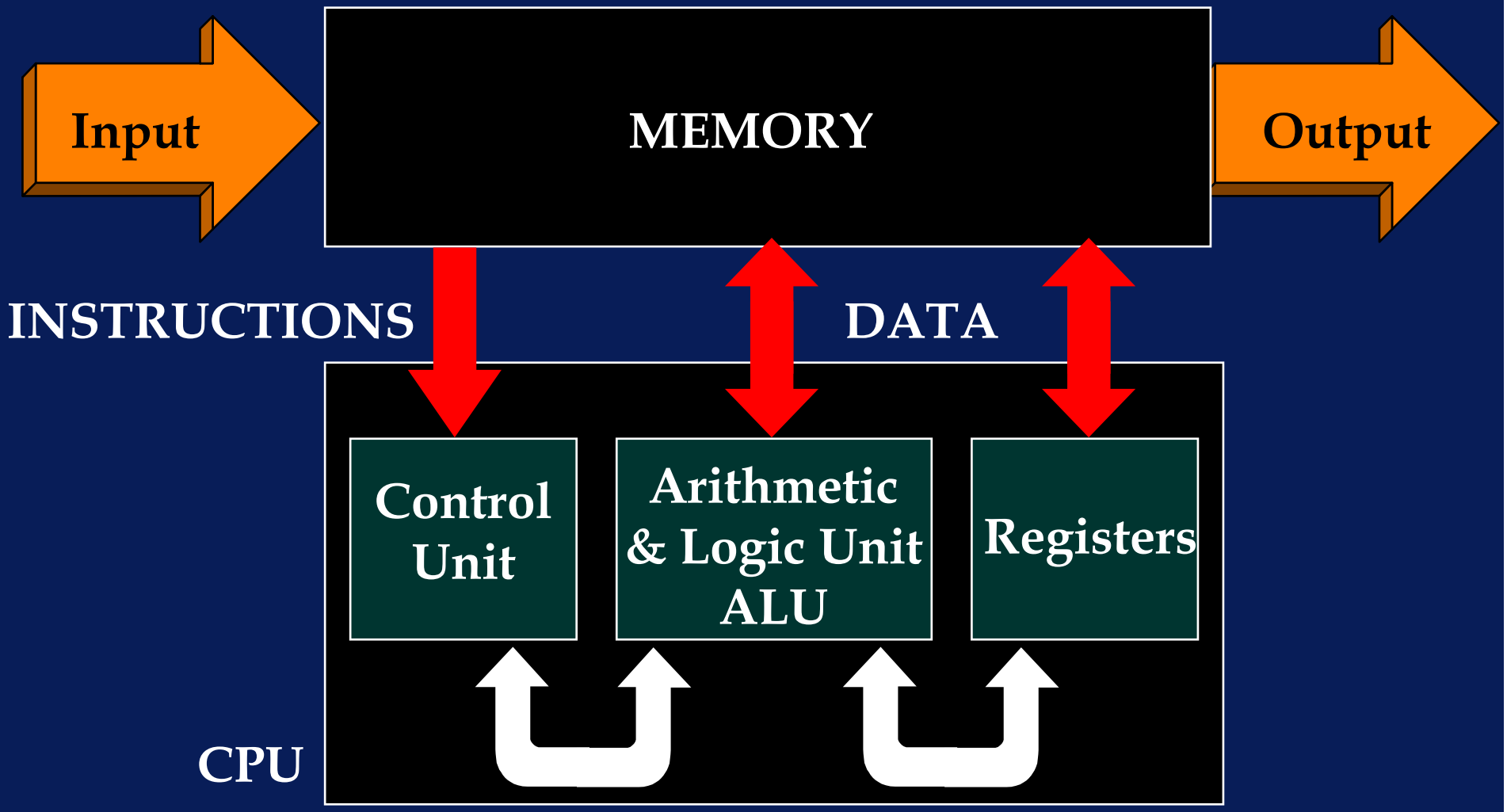
The Operation of a Processor

- A := B + C
- A computer can’t evaluate this directly (because it’s not written in a way which matches the structure of the computer’s physical architecture)
- First, this must be translated into a sequence of instructions which the does match the computer architecture

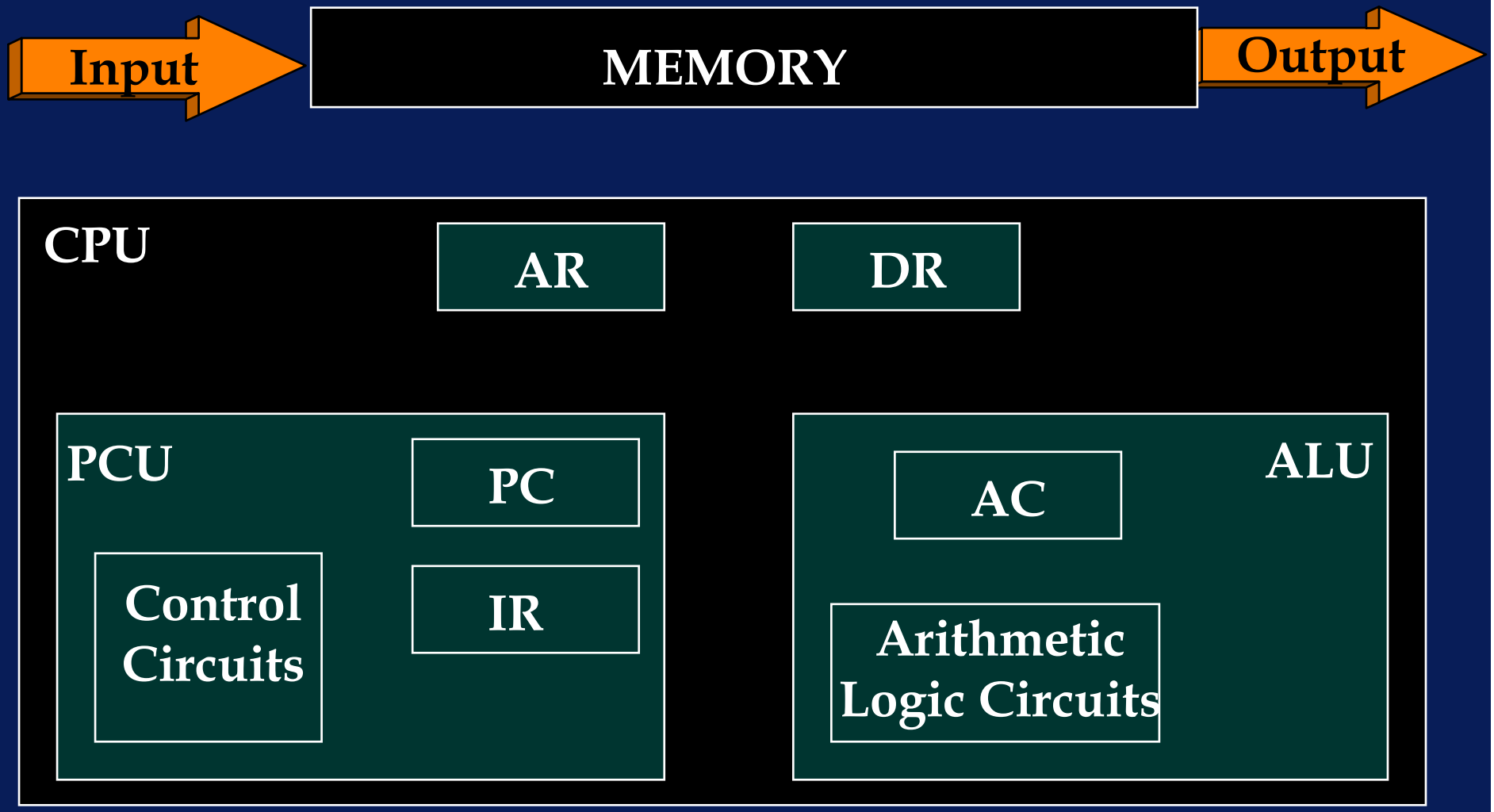
The Operation of a Processor

- A := B + C
- So ... we need a detailed processor architecture (i.e. a machine)
- and a matching language (machine language or assembly language)
  - machine language when it's written as a binary code
  - assembly language when it's written symbolically.

The Components of a Processor



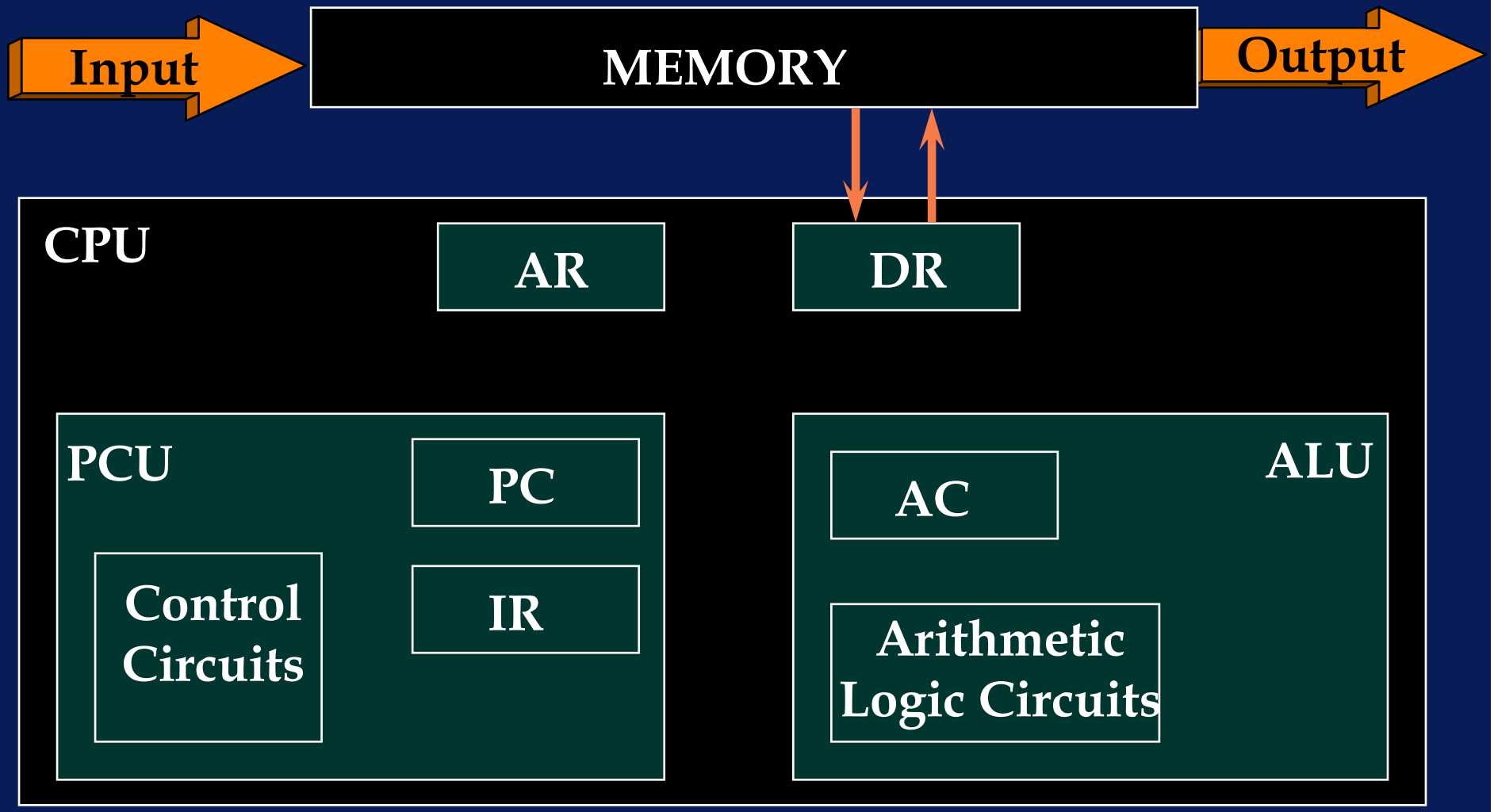
The Components of a Processor



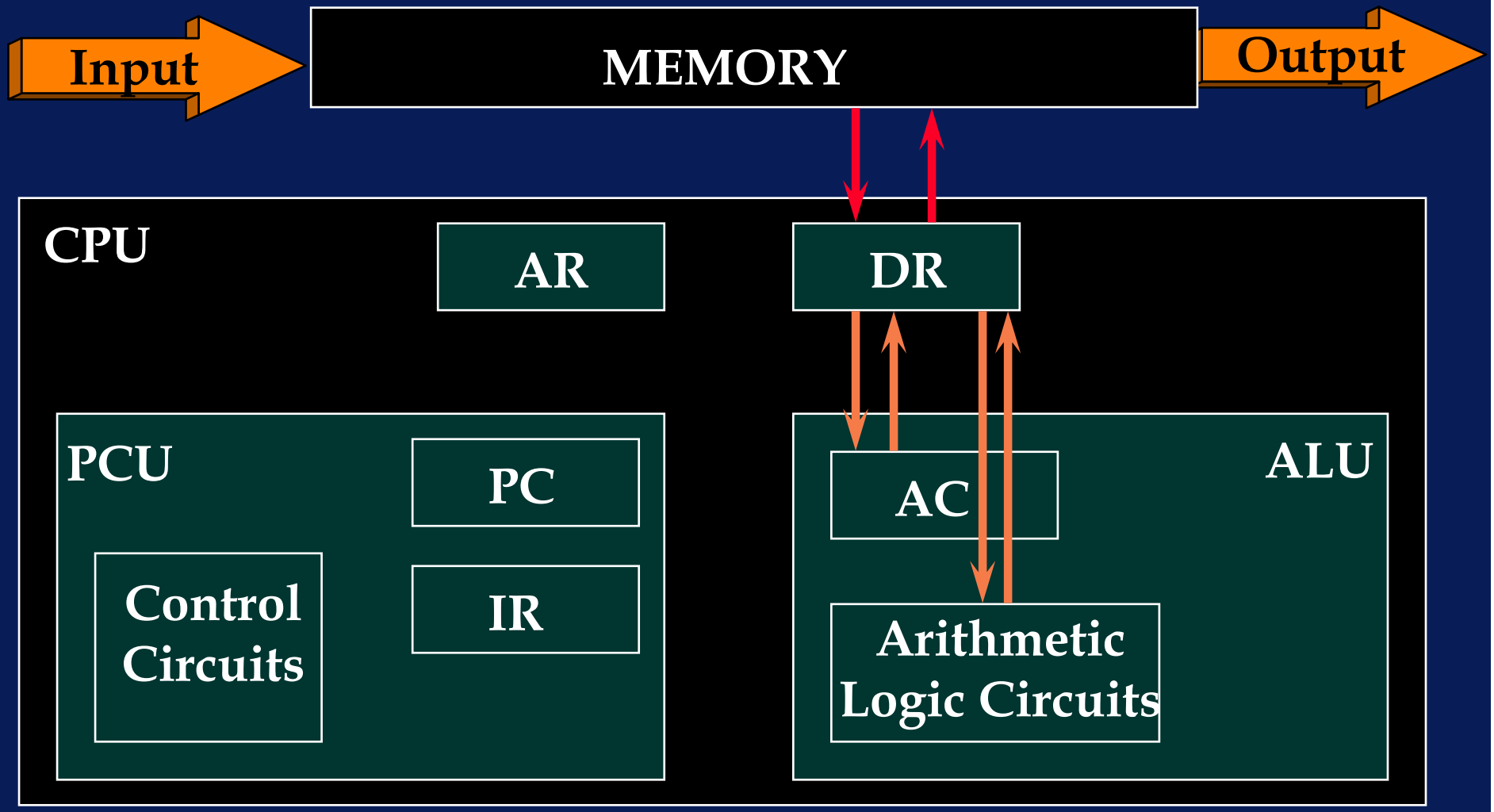
The Components of a Processor

- DR - Data Register
- AR - Address Register
- AC - Accumulator
- PC - Progam Counter
- IR - Instruction Register
- ALU - Arithmetic Logic Unit
- PCU - Program Control Unit

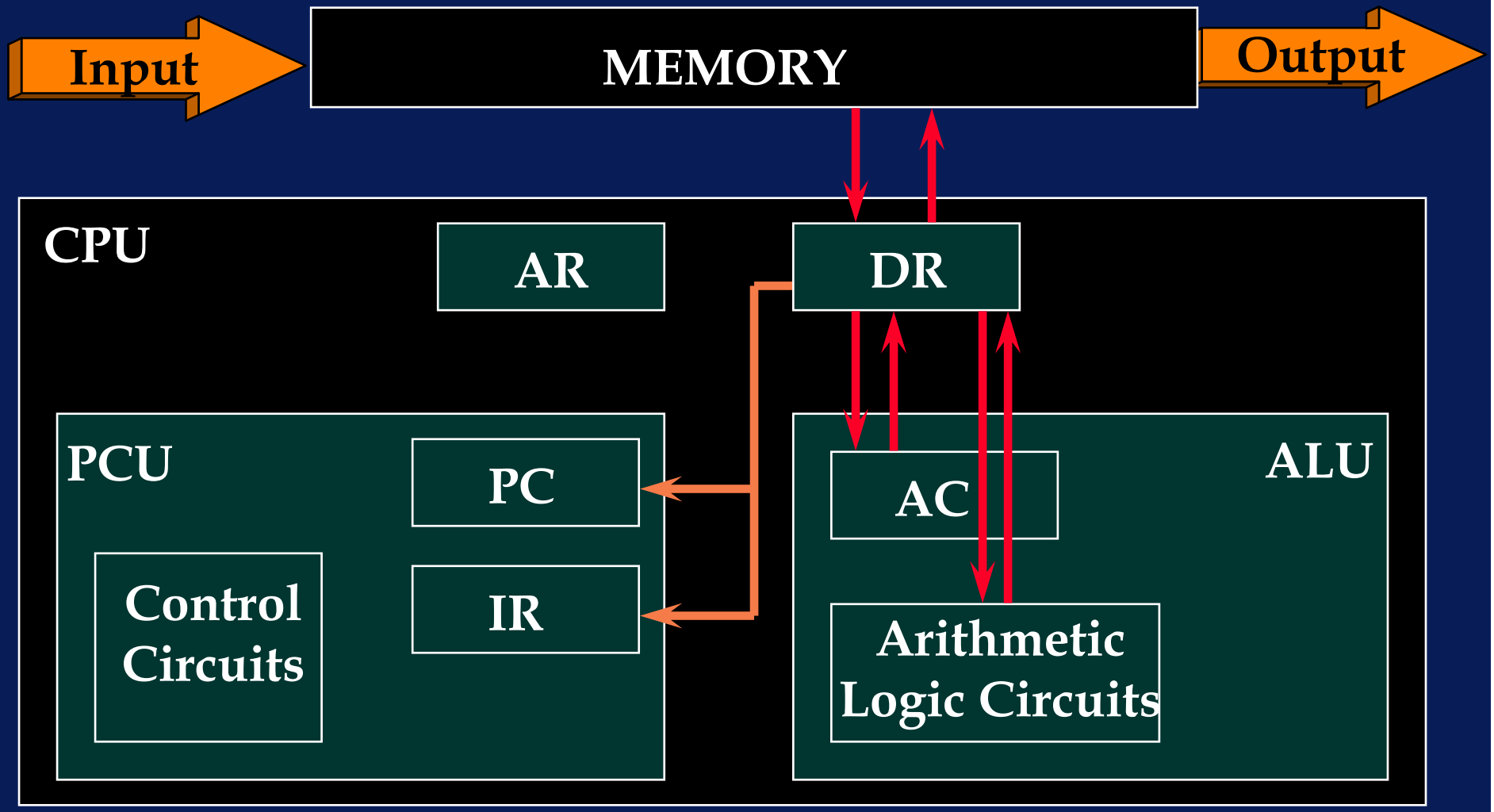
The Components of a Processor



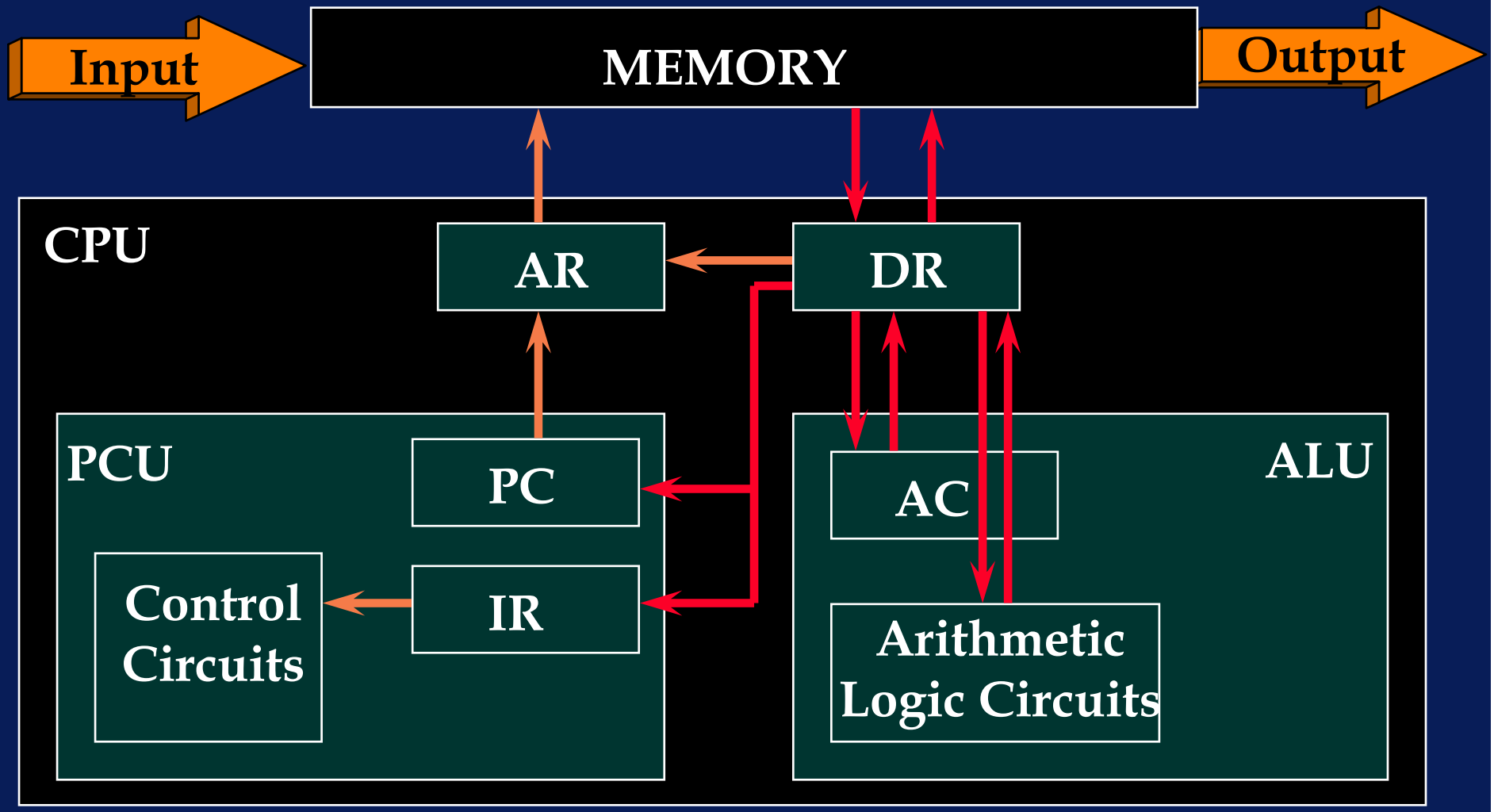
The Components of a Processor



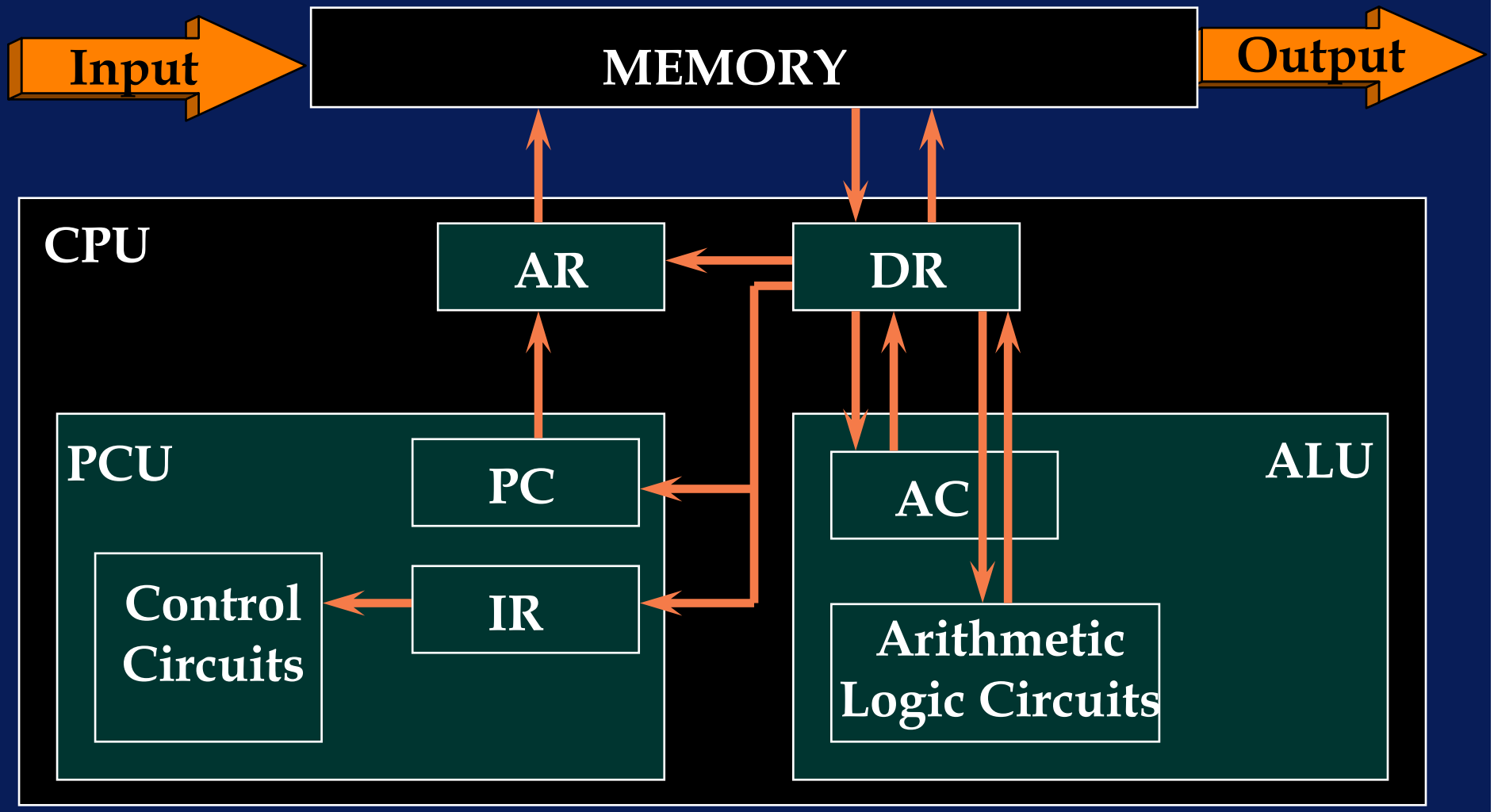
The Components of a Processor



The Components of a Processor



The Components of a Processor



Instruction Format

- An instruction is typically divided into 2 fields

- This is already simplified situation and, in general, one would expect
  - opcode/operand
  - opcode/address (which may vary in size)

Instruction Format

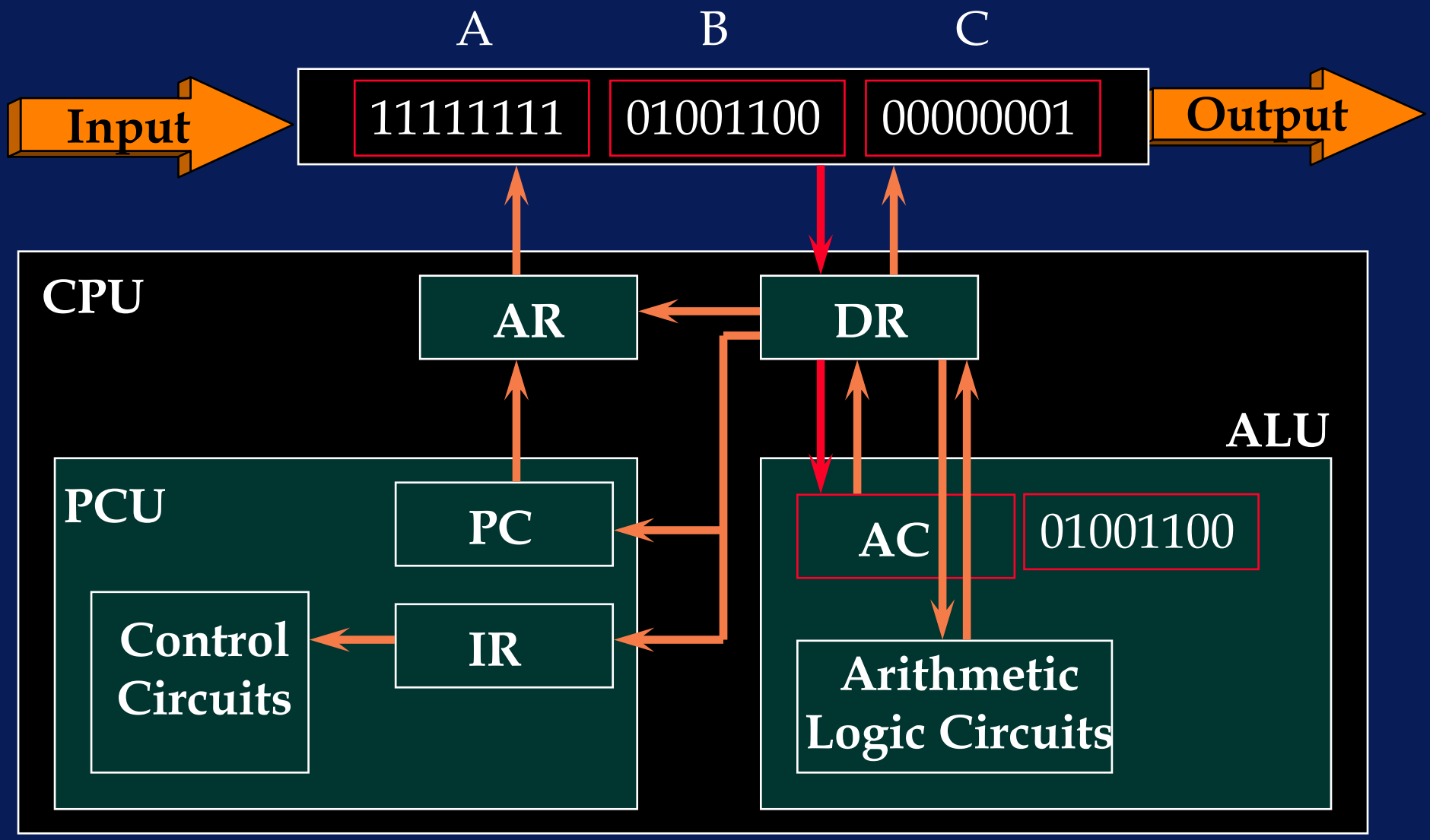
- Load X
  - puts contents of memory location X into the accumulator
- Add T
  - Add contents of memory at location T to the contents of the accumulator
- Store Y
  - Put contents of accumulator into memory at location Y

Evaluate an Assignment

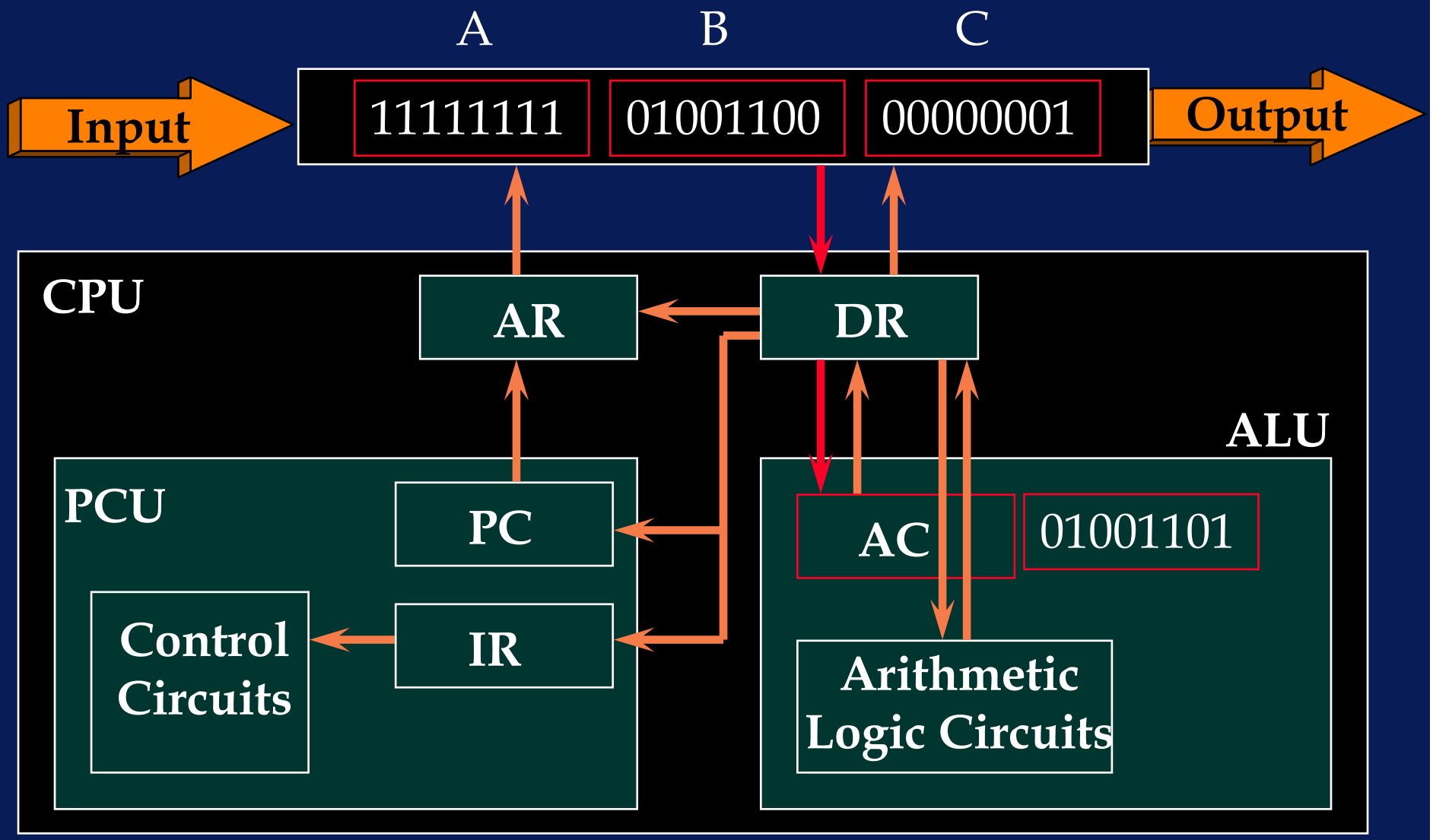
- A := B + C

- Load B
- Add C
- Store A

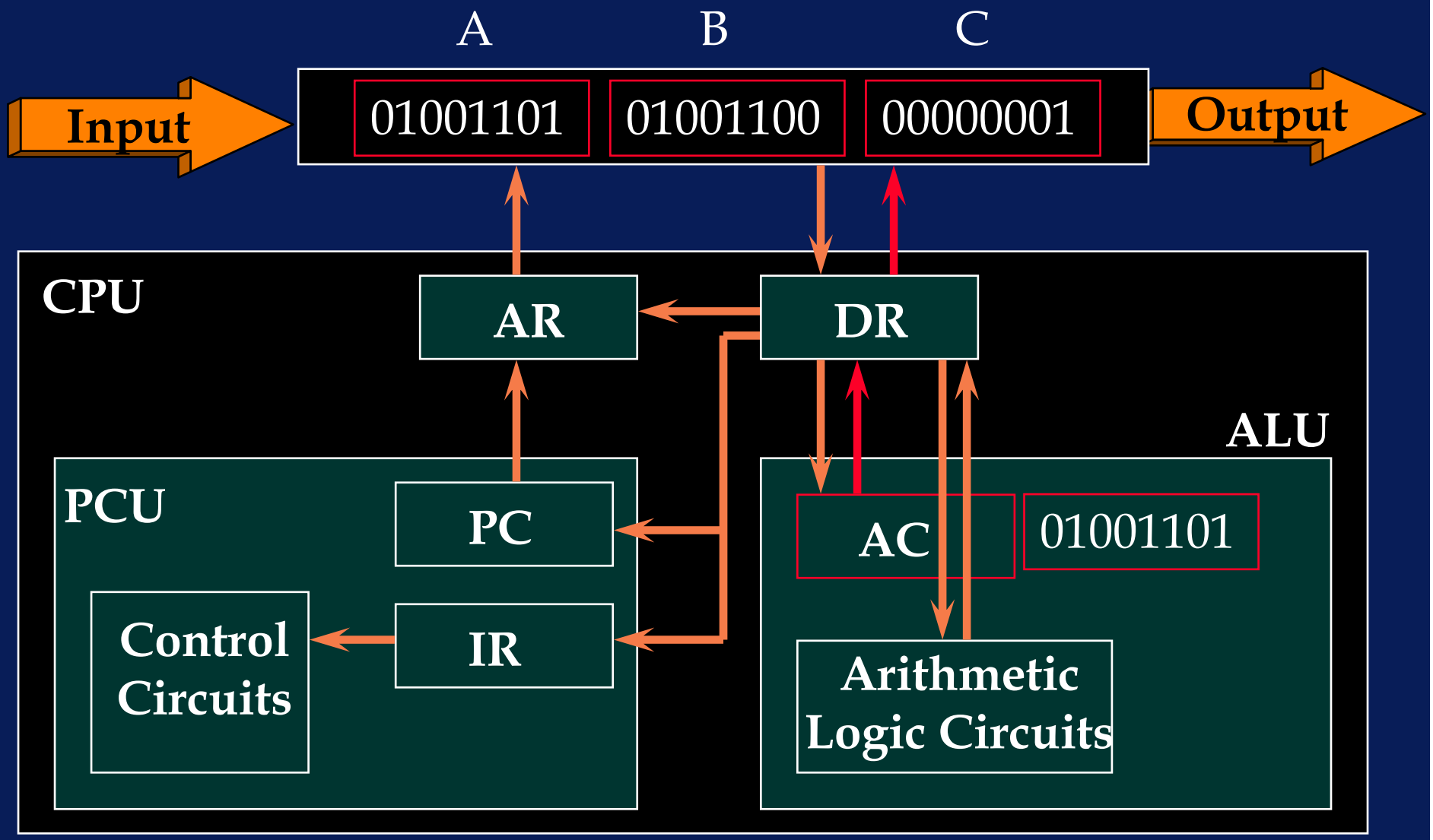
Load B



Add C

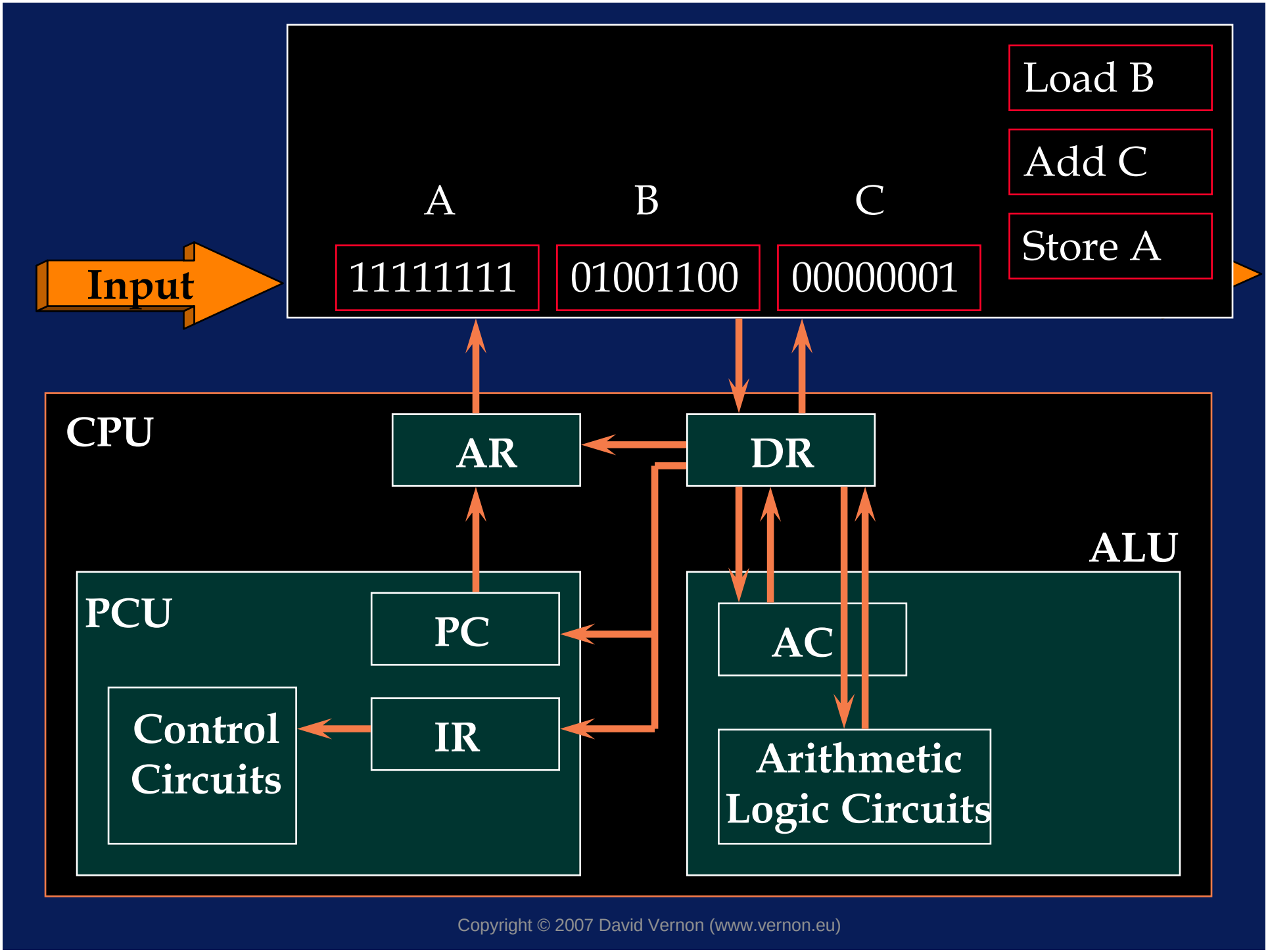


Store A



But .....

- The instructions are stored in memory also!



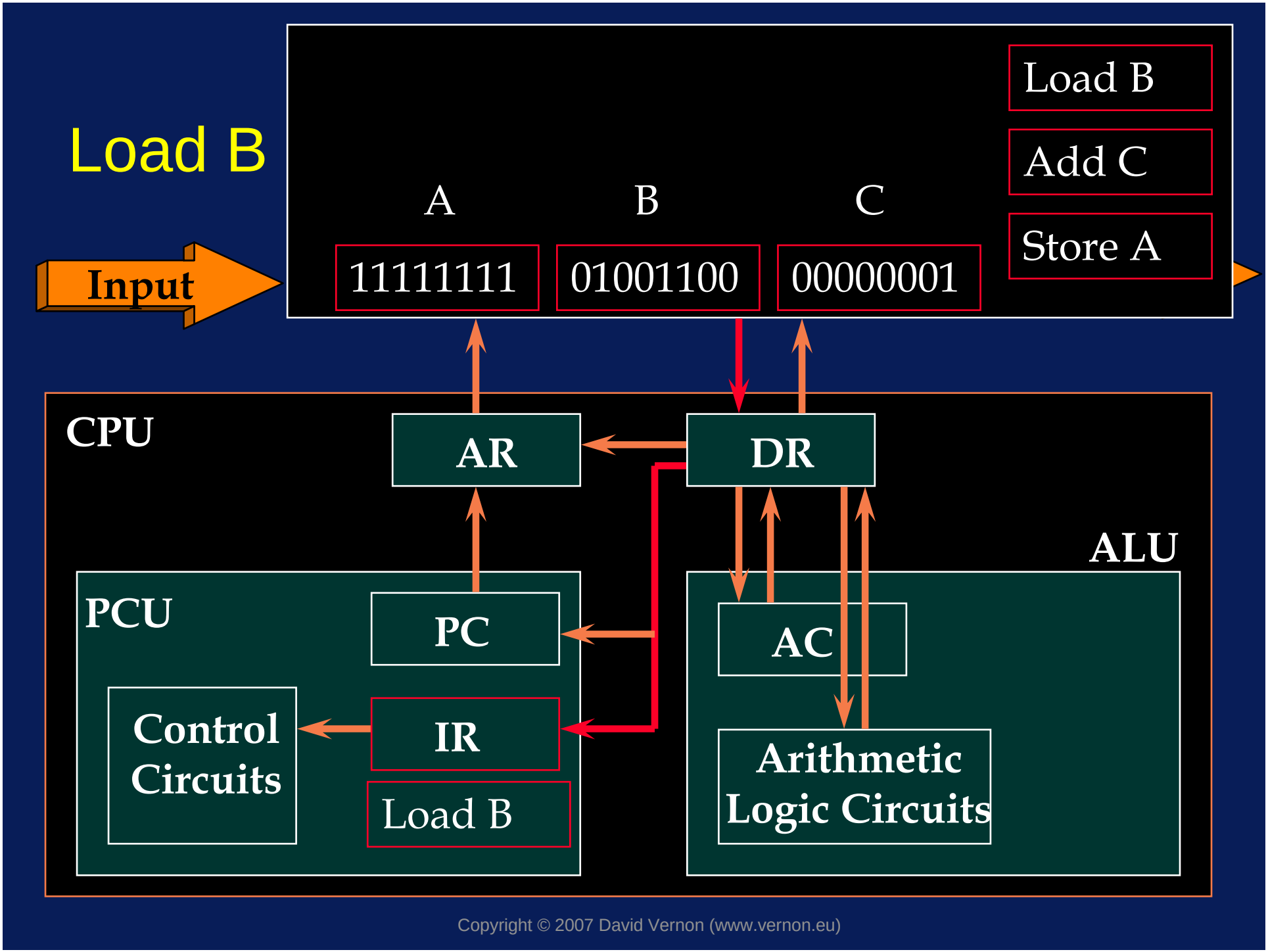
So .....

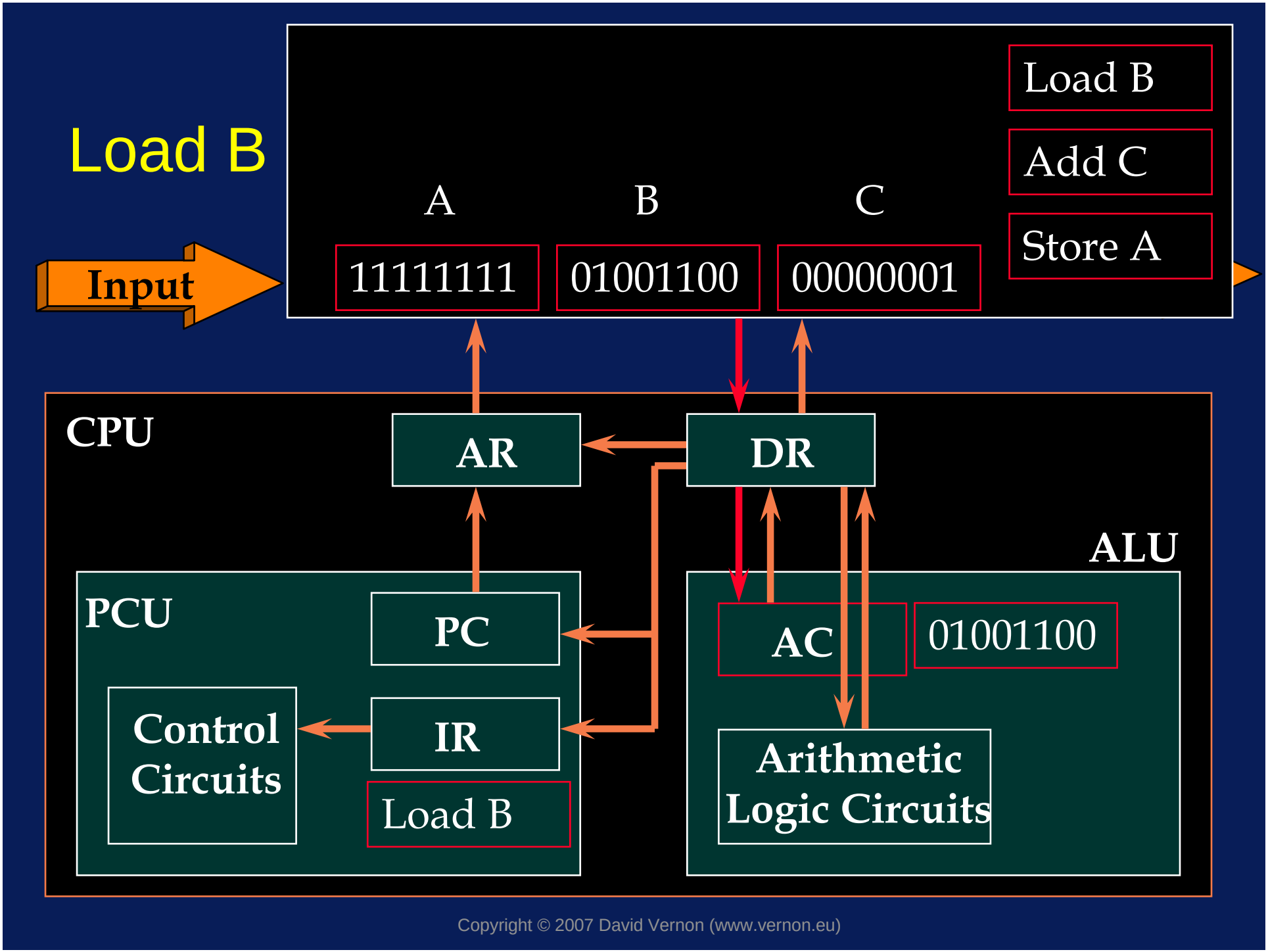
- It works more like this ...

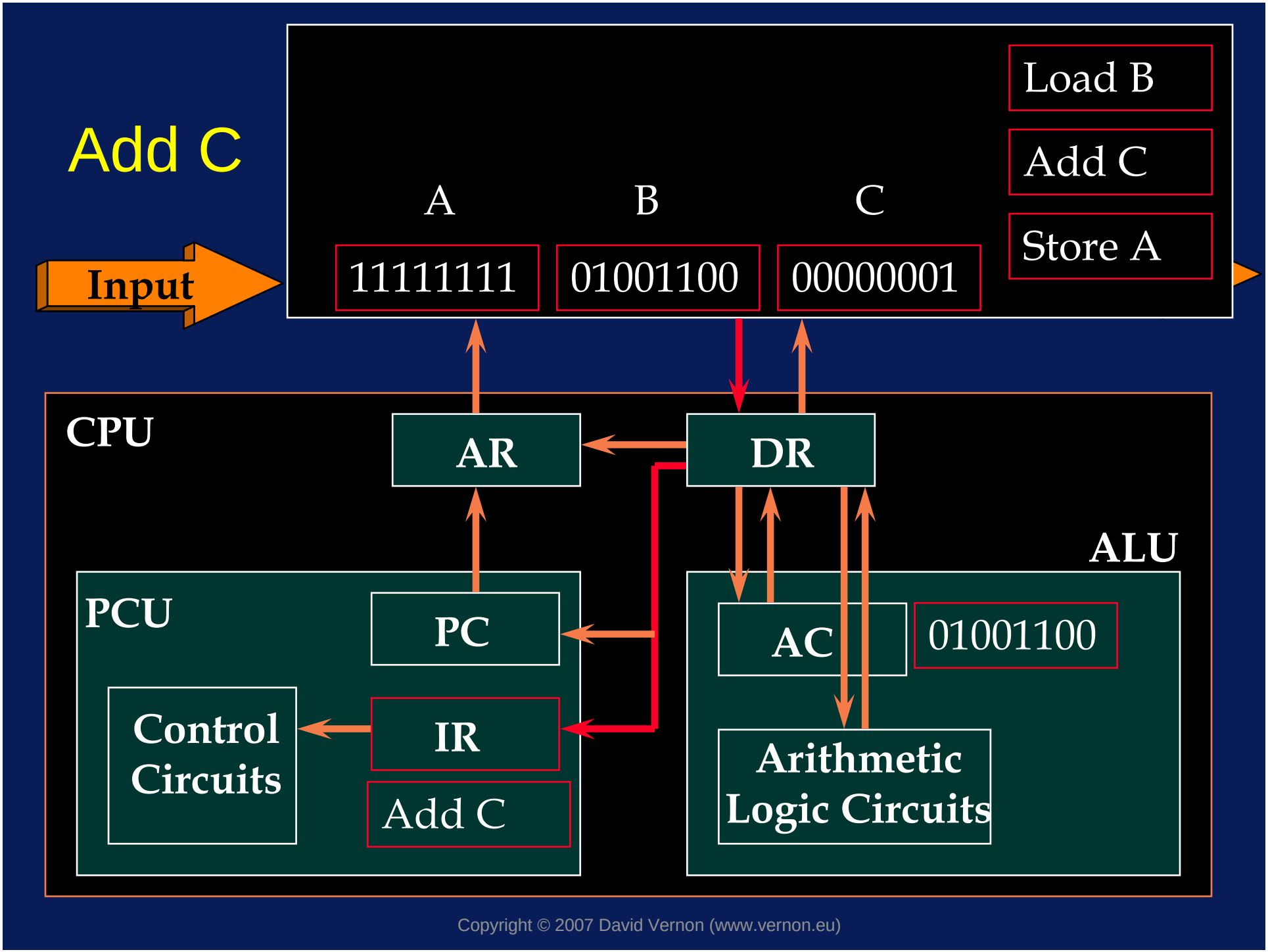
Evaluate an Assignment

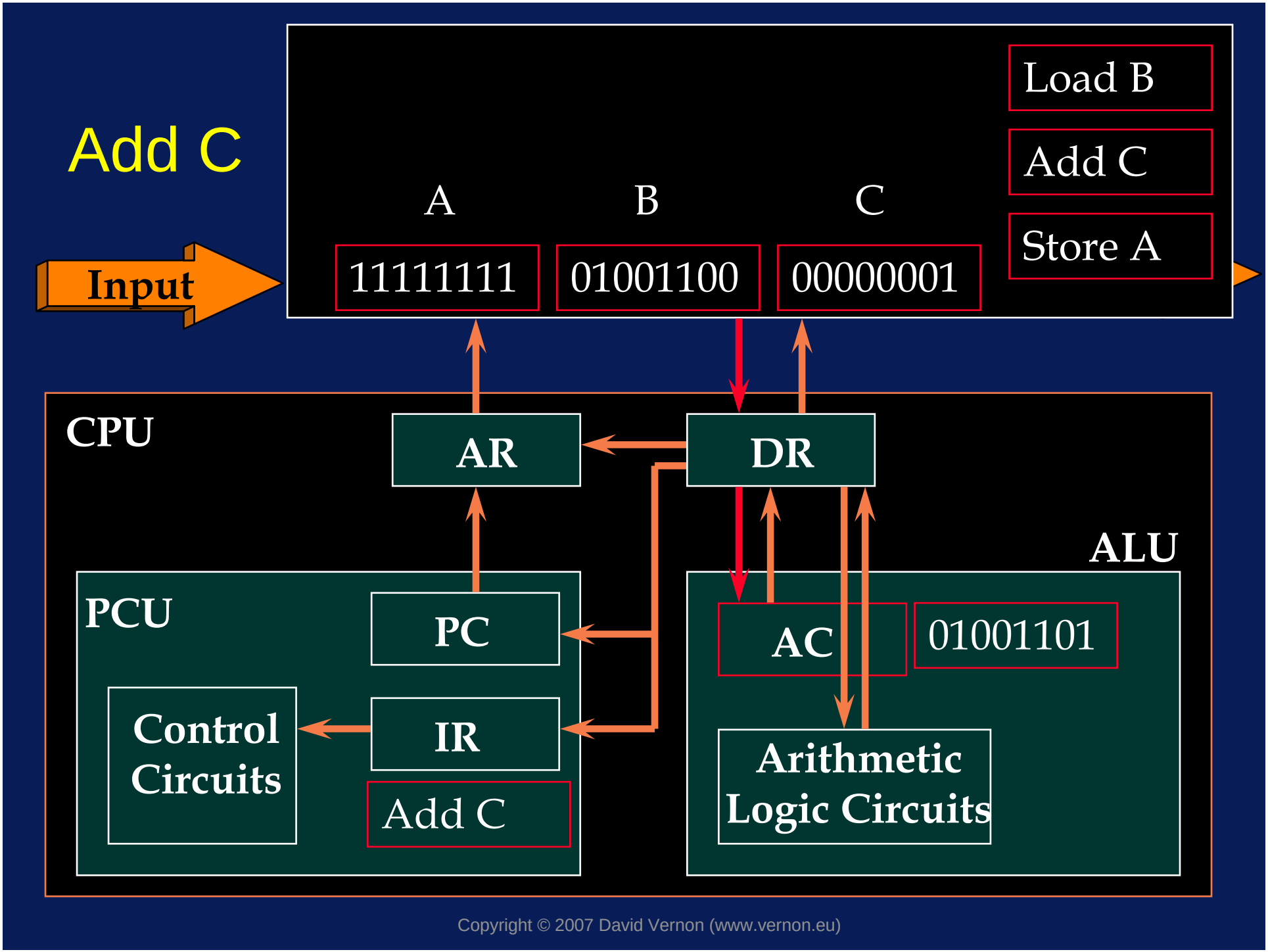
- A := B + C

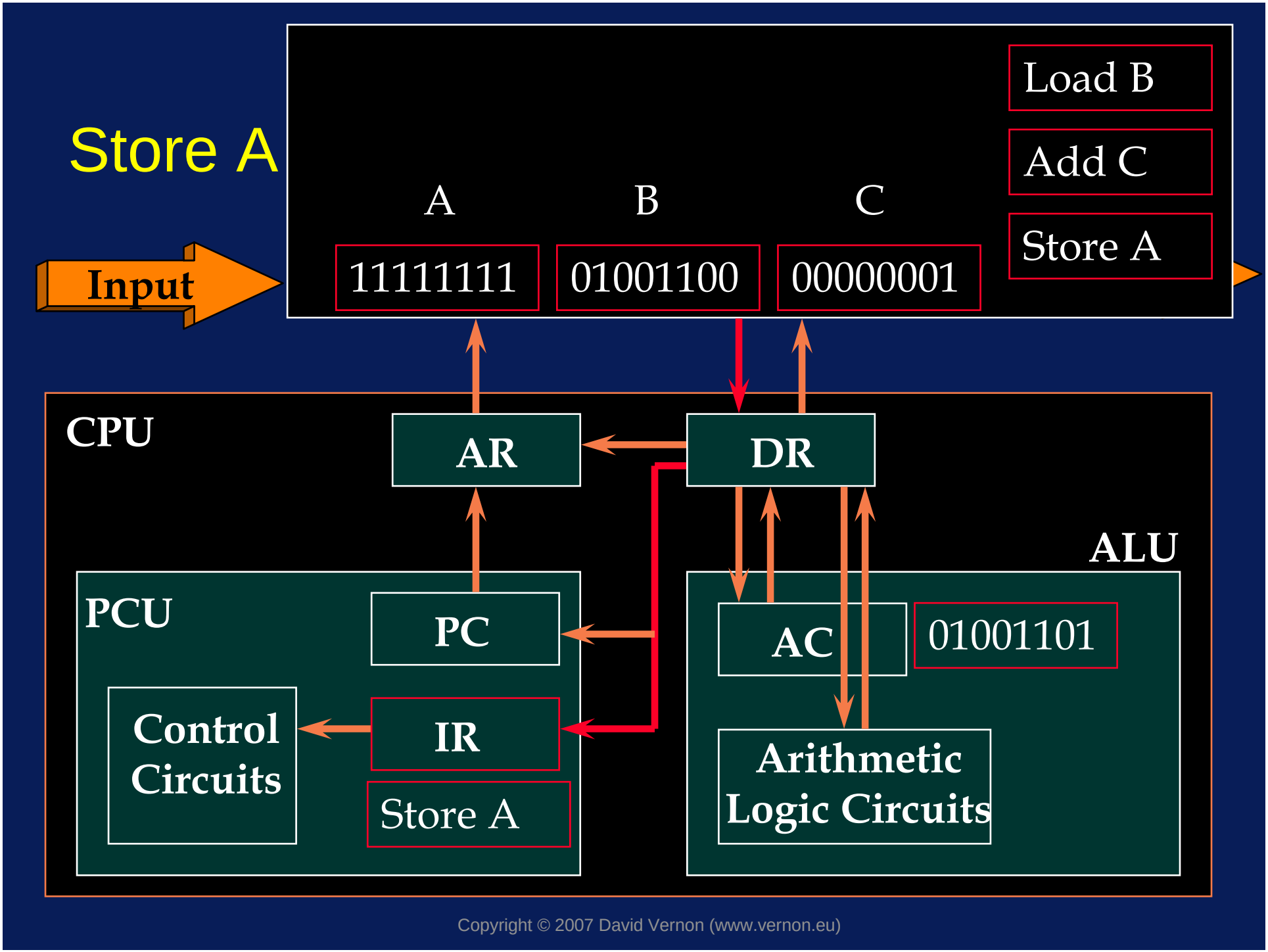
- Load B
- Add C
- Store A

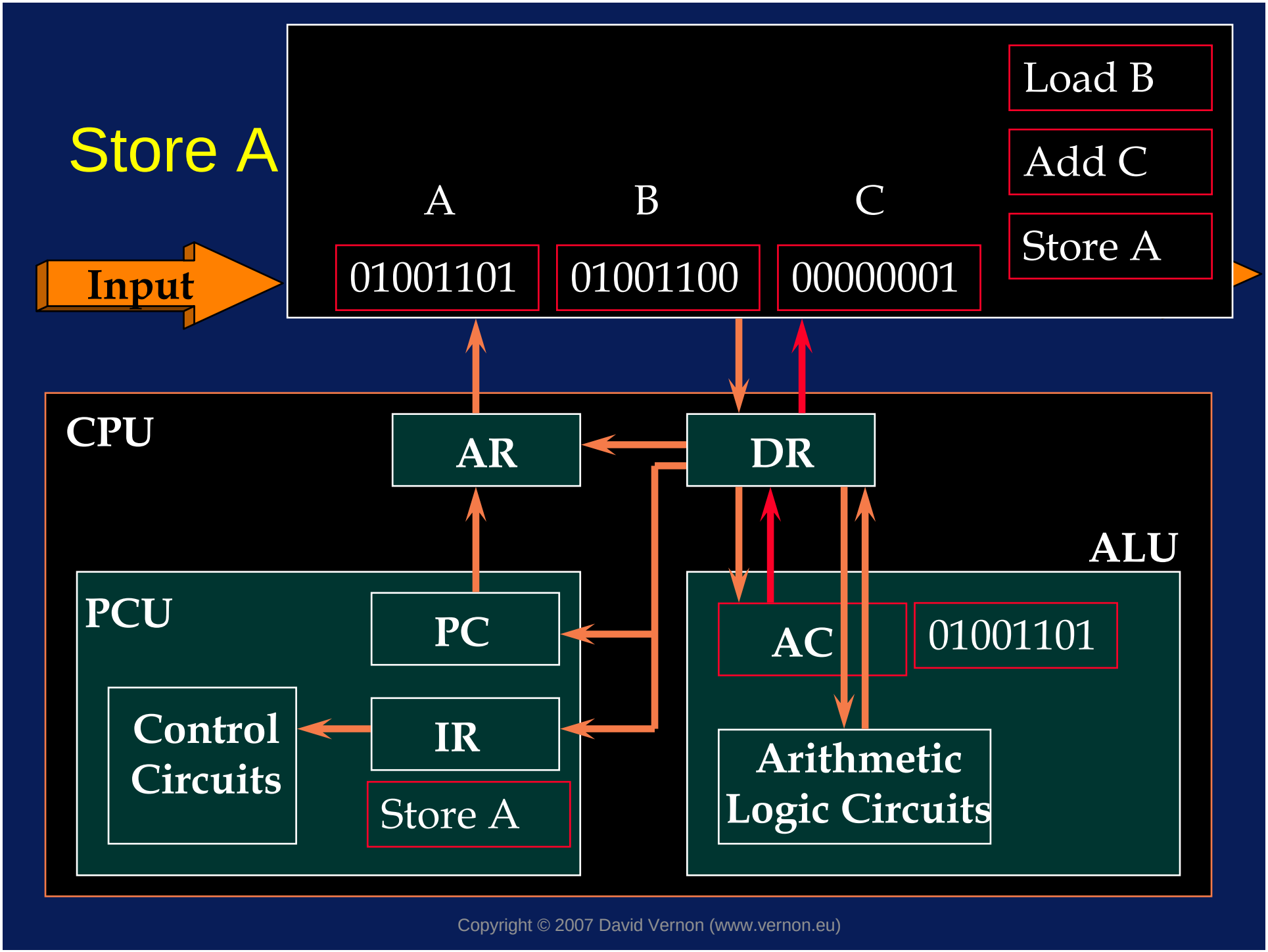


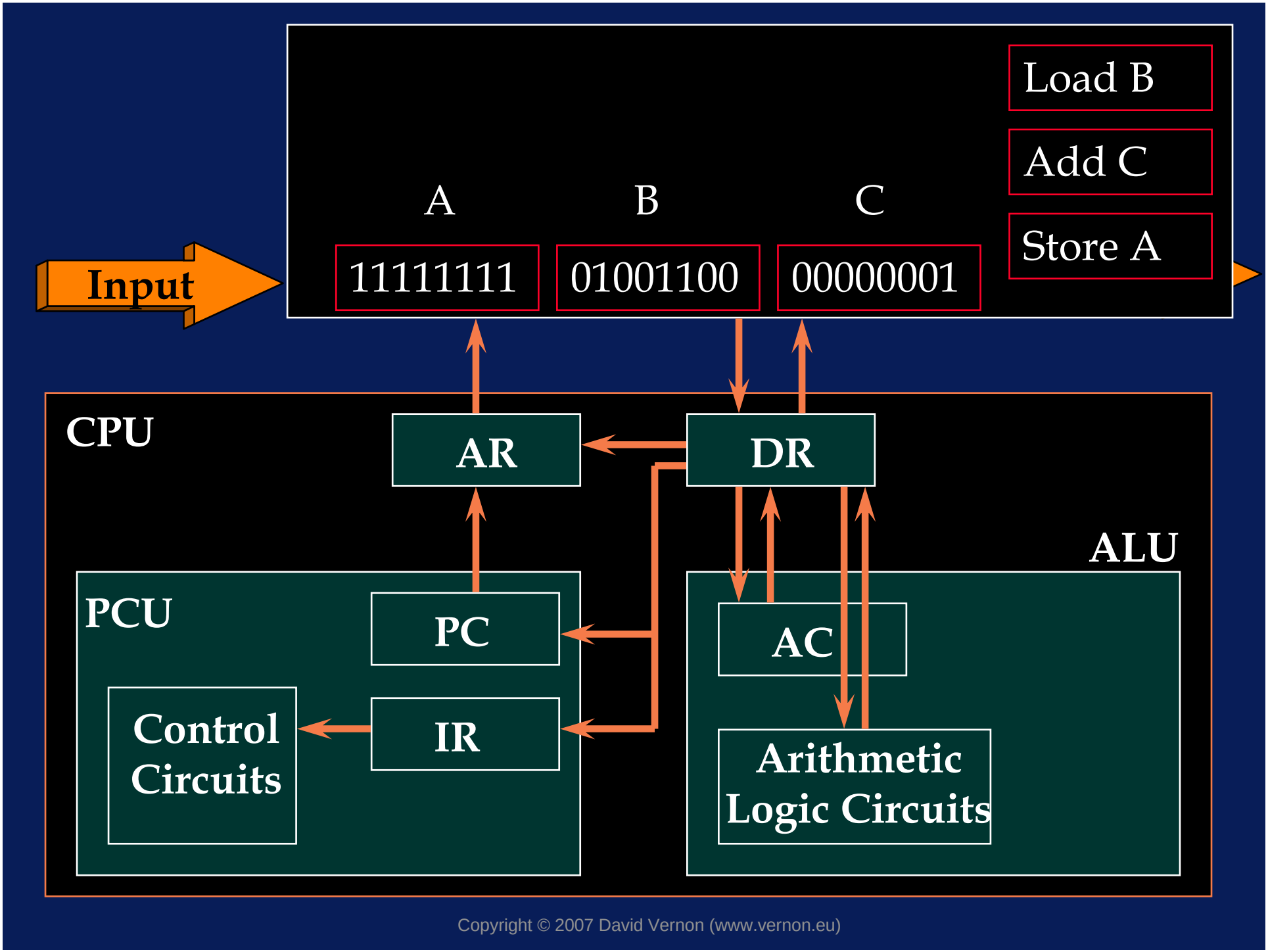












Operation of the Processor

- The primary function of a processor is to execute sequences of instructions stored in main memory

- Instruction Cycle
  - Fetch Cycle
  - Execute Cycle

Instruction Cycle

- Fetch Cycle
  - Fetch instruction from memory
- Execute Cycle
  - Decode instruction
  - Fetch required operands
  - Perform operation

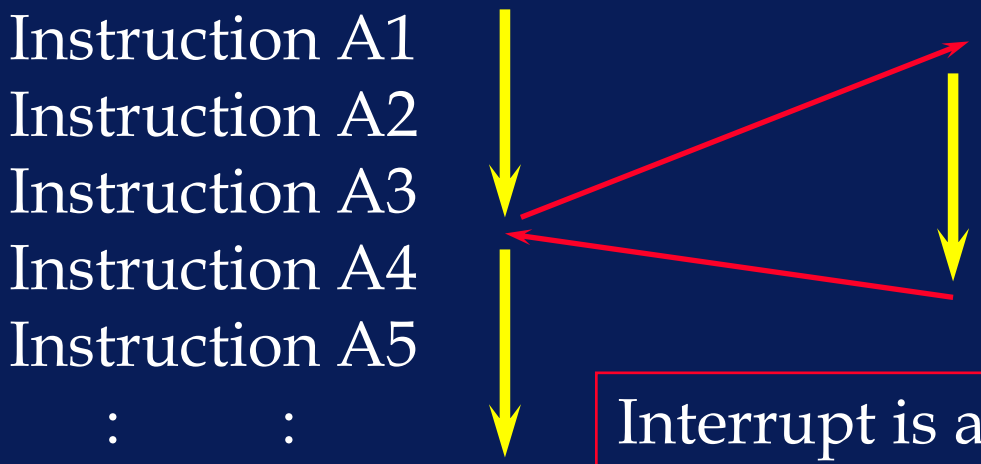
Instruction Cycle

- Instruction cycle comprises a sequence of micro-operations each of which involves a transfer of data to/from registers

Instruction Cycle

- In addition to executing instructions the CPU supervises other system component usually via special control lines
- It controls I/O operations
  (either directly or indirectly)
- Since I/O is a relatively infrequent event, I/O devices are usually ignored until they actively request service from the CPU via an interrupt

An Interrupt

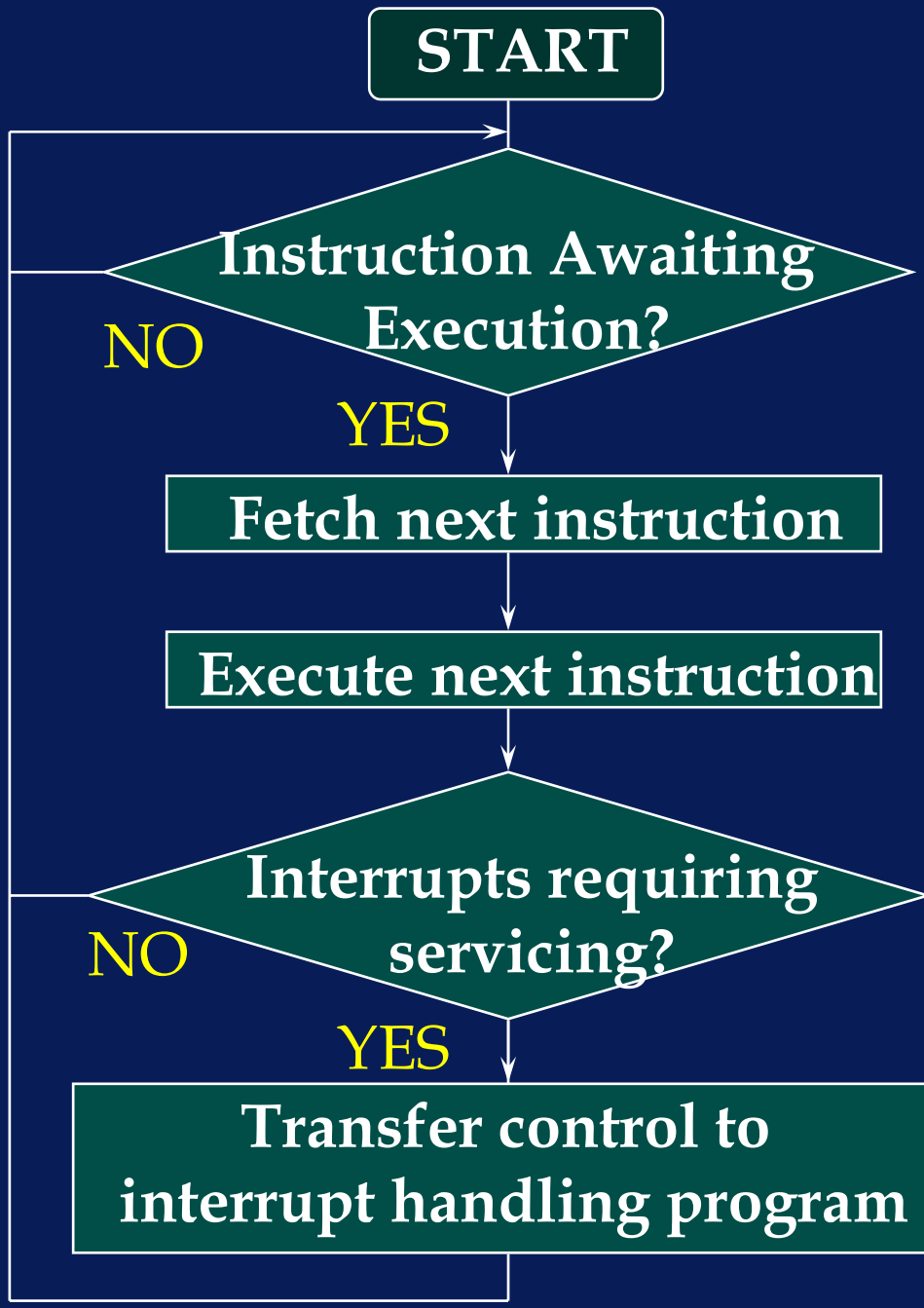


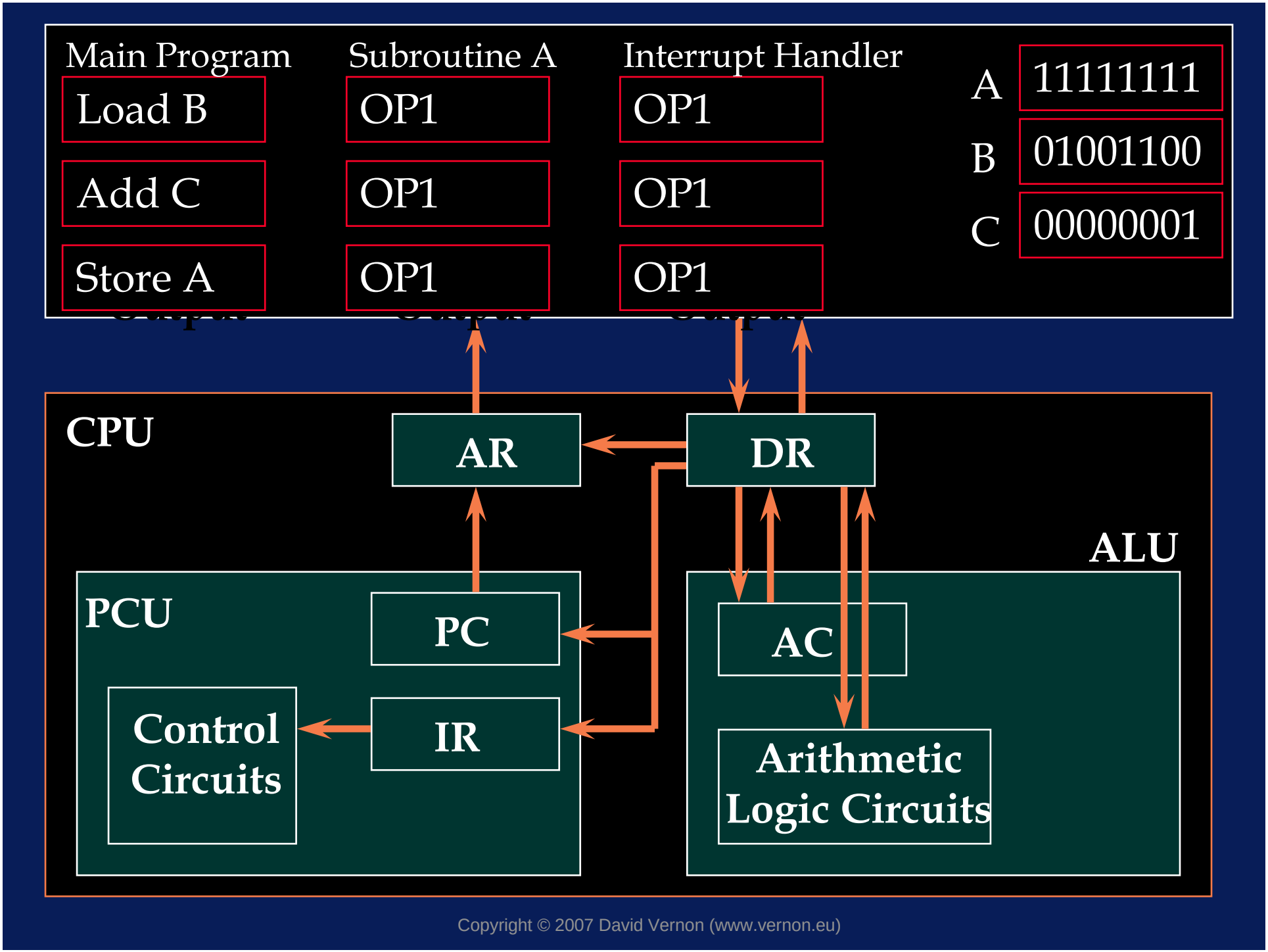
Interrupt is activated by an electronic signal

EXECUTE instructions

BRANCH to new set of instructions

Instruction Cycle





Register Transfer Language

- Storage locations, in CPU and memory, are referred to by an acronym
- AC
  - Accumulator; main operand register of ALU

Register Transfer Language

- DR
  - Data Register; acts as a buffer between CPU and main memory.
  - It is used as an input operand register with accumulator to facilitate operations of the form AC $\leftarrow$ f(AC, DR)

Register Transfer Language

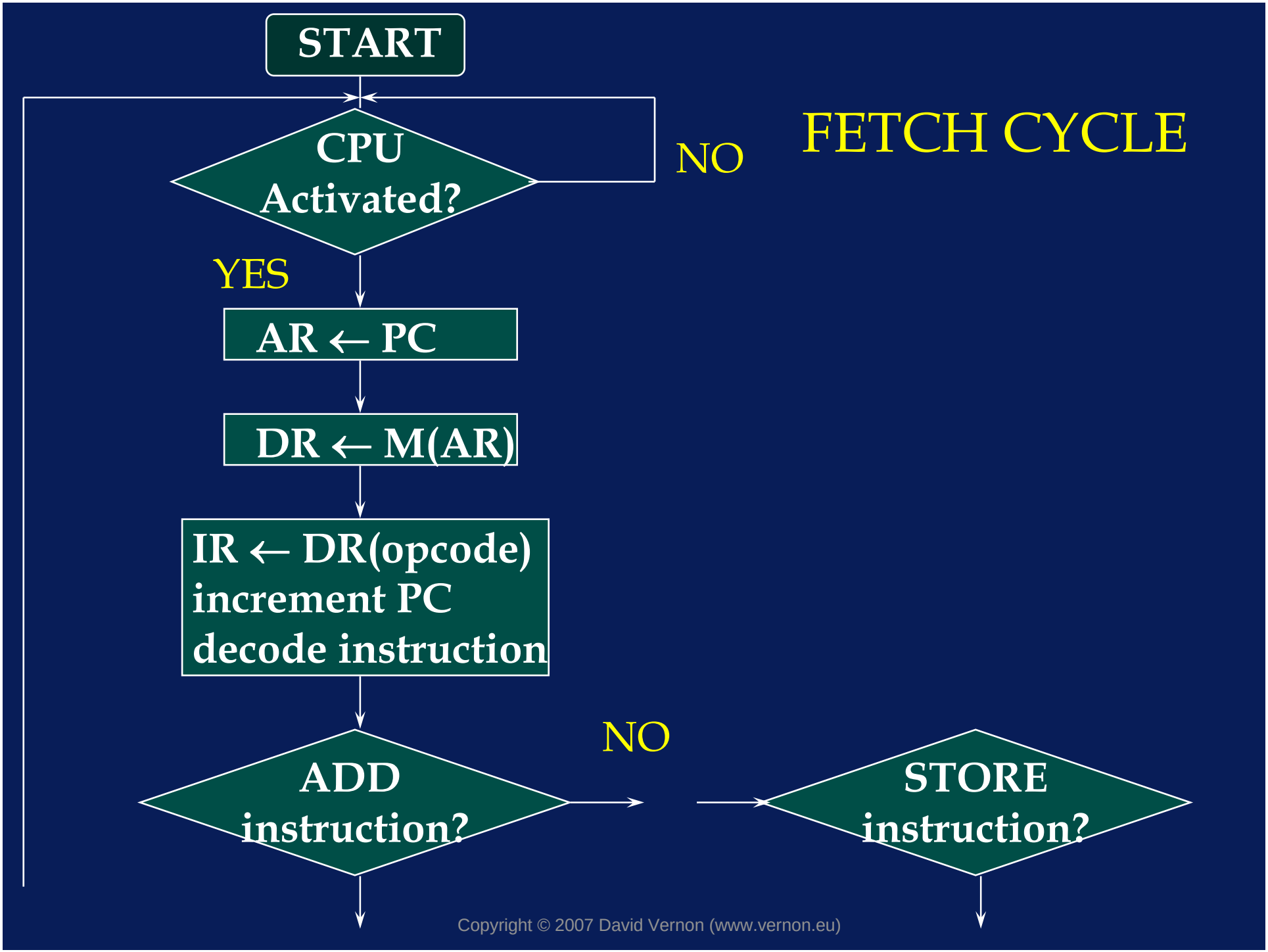
- PC
  - Program Counter
  - Stores address of the next instruction to be executed
- IR
  - Instruction Register
  - Holds the opcode of the current instruction

Register Transfer Language

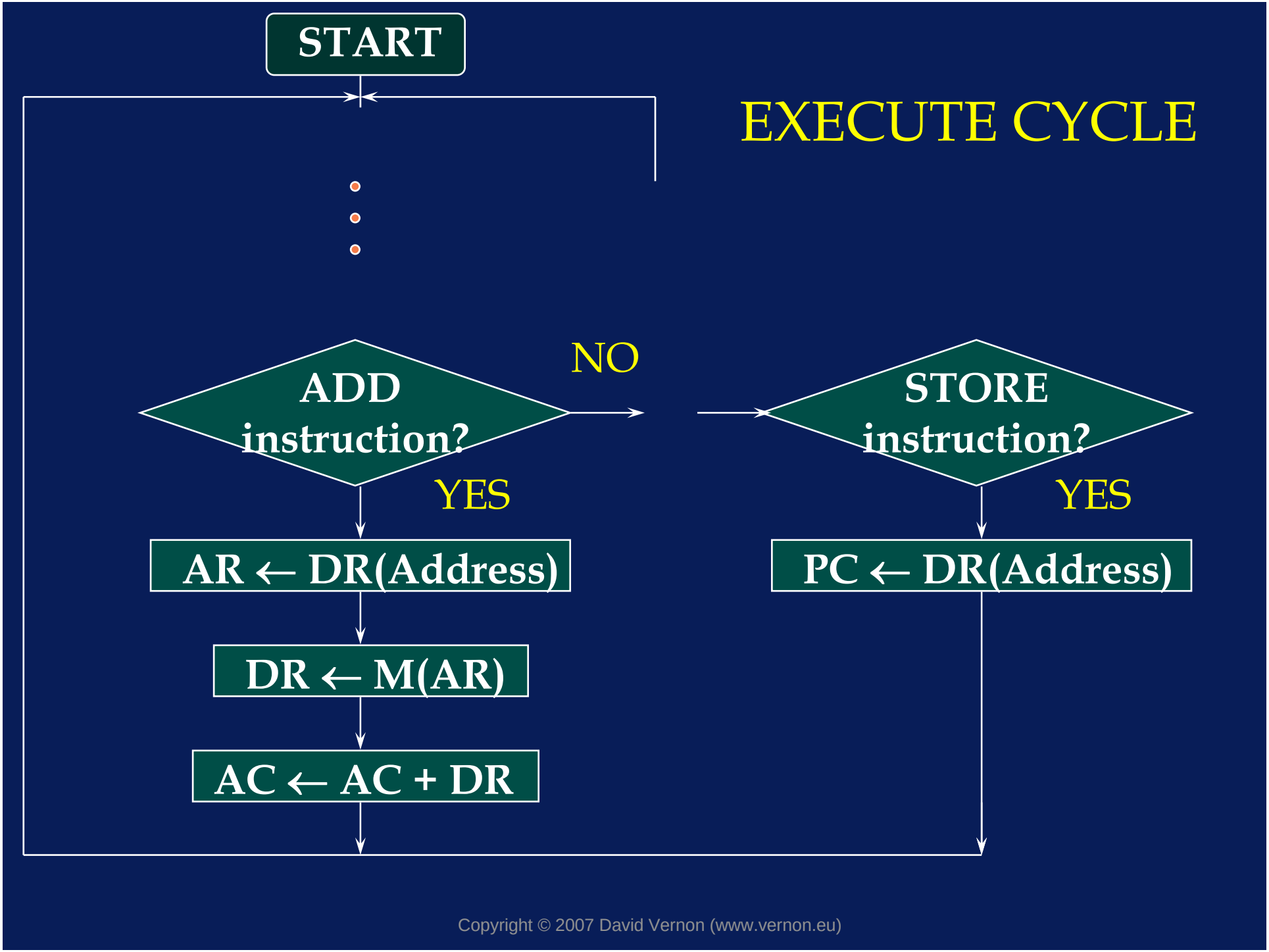
- AR
  - Address Register
  - Holds the memory address of an operand

Register Transfer Language

- A←B
  - transfer contents of storage location B to A (copy operation)
- A←M(ADR)
  - transfer contents of memory at location ADR to location A



EXECUTE CYCLE



Evaluate an Assignment

- A := B + C
  - Load B
  - Add C
  - Store A

A := B + C

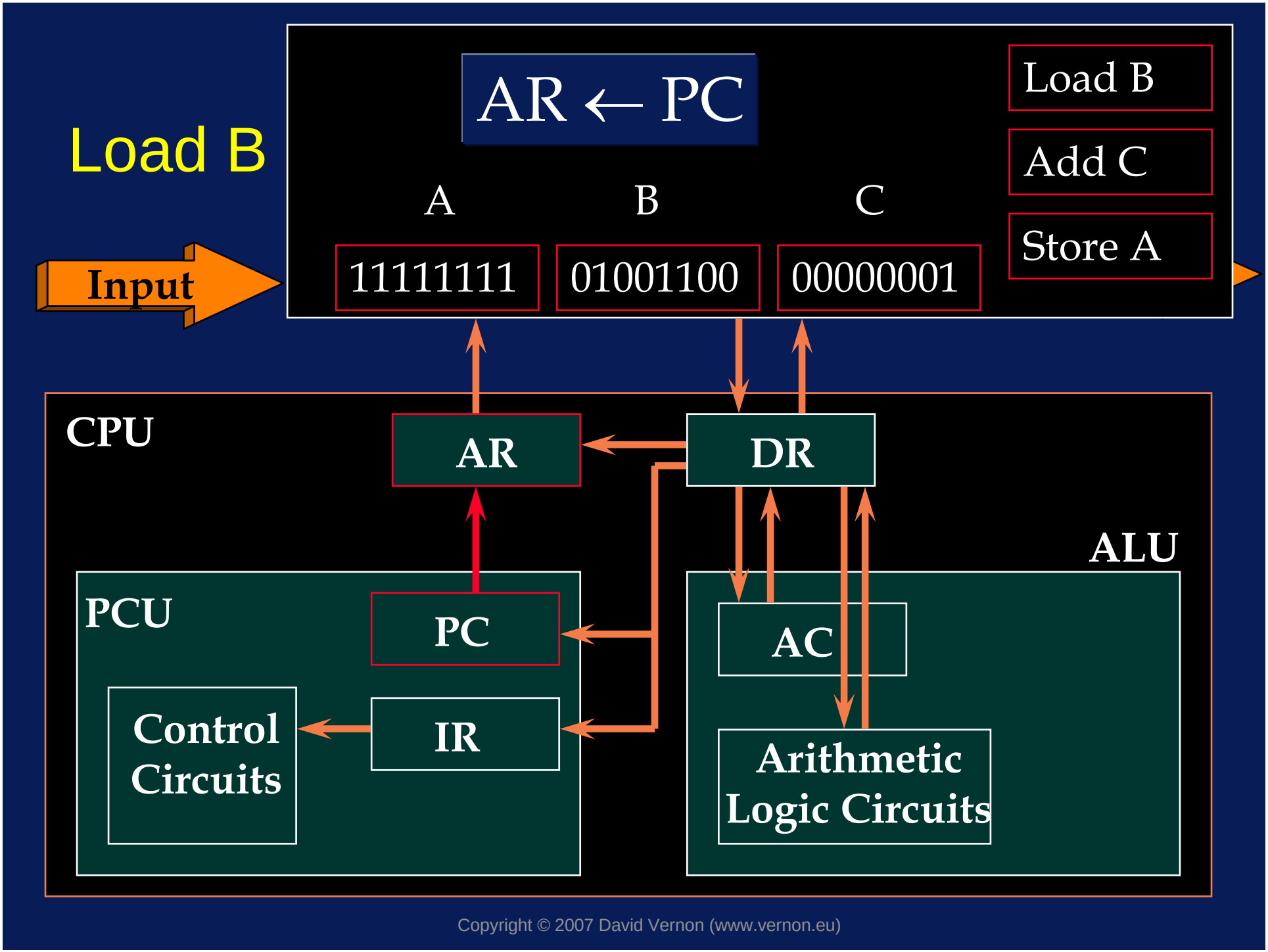
- Load B
  - AR ← PC
  - DR ← M(AR)
  - IR← DR(opcode)
  - Increment PC
  - Decode instruction in IR
  - AR ← DR(address)
  - DR ← M(AR)
  - AC ← DR

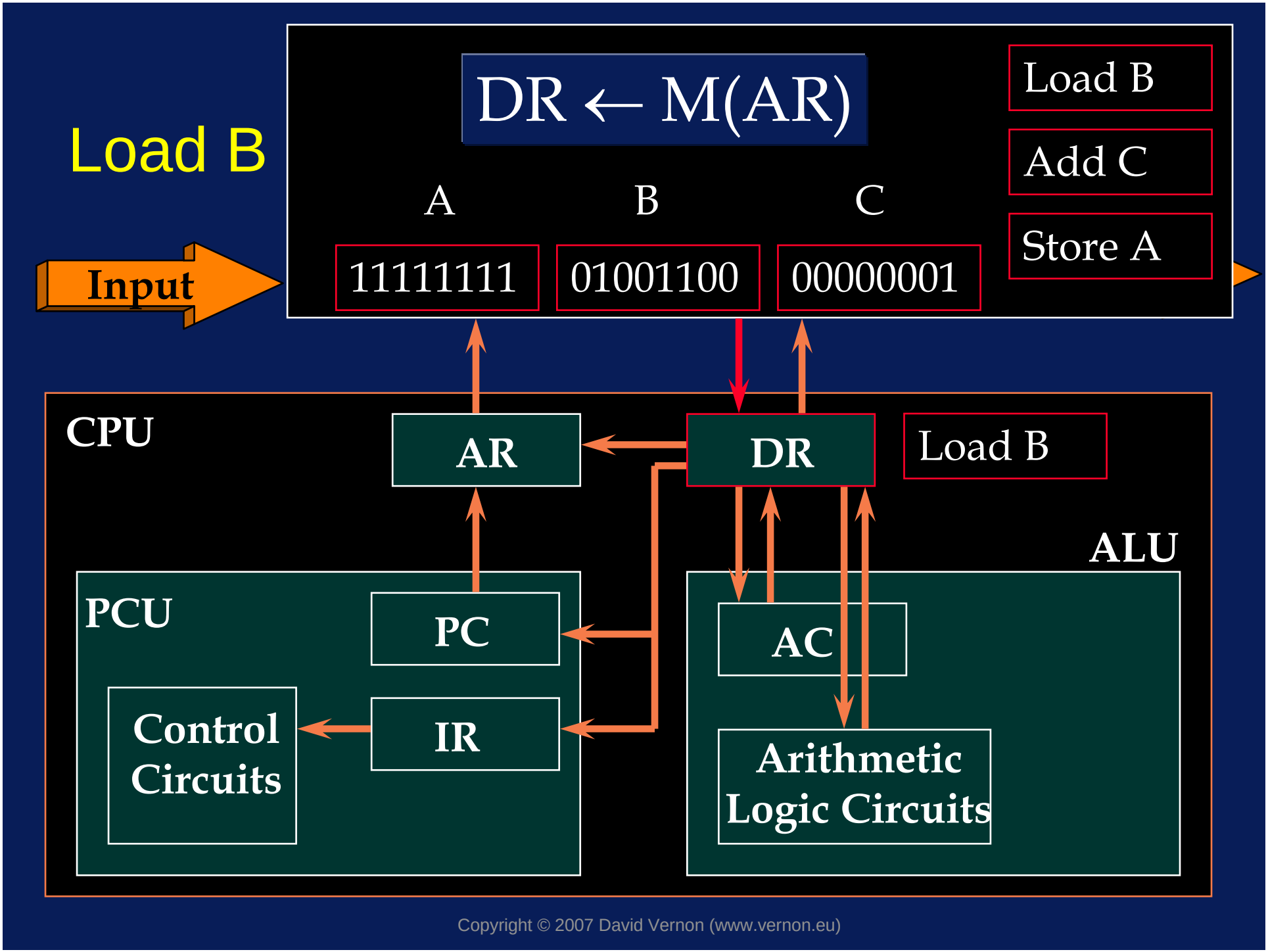
A := B + C

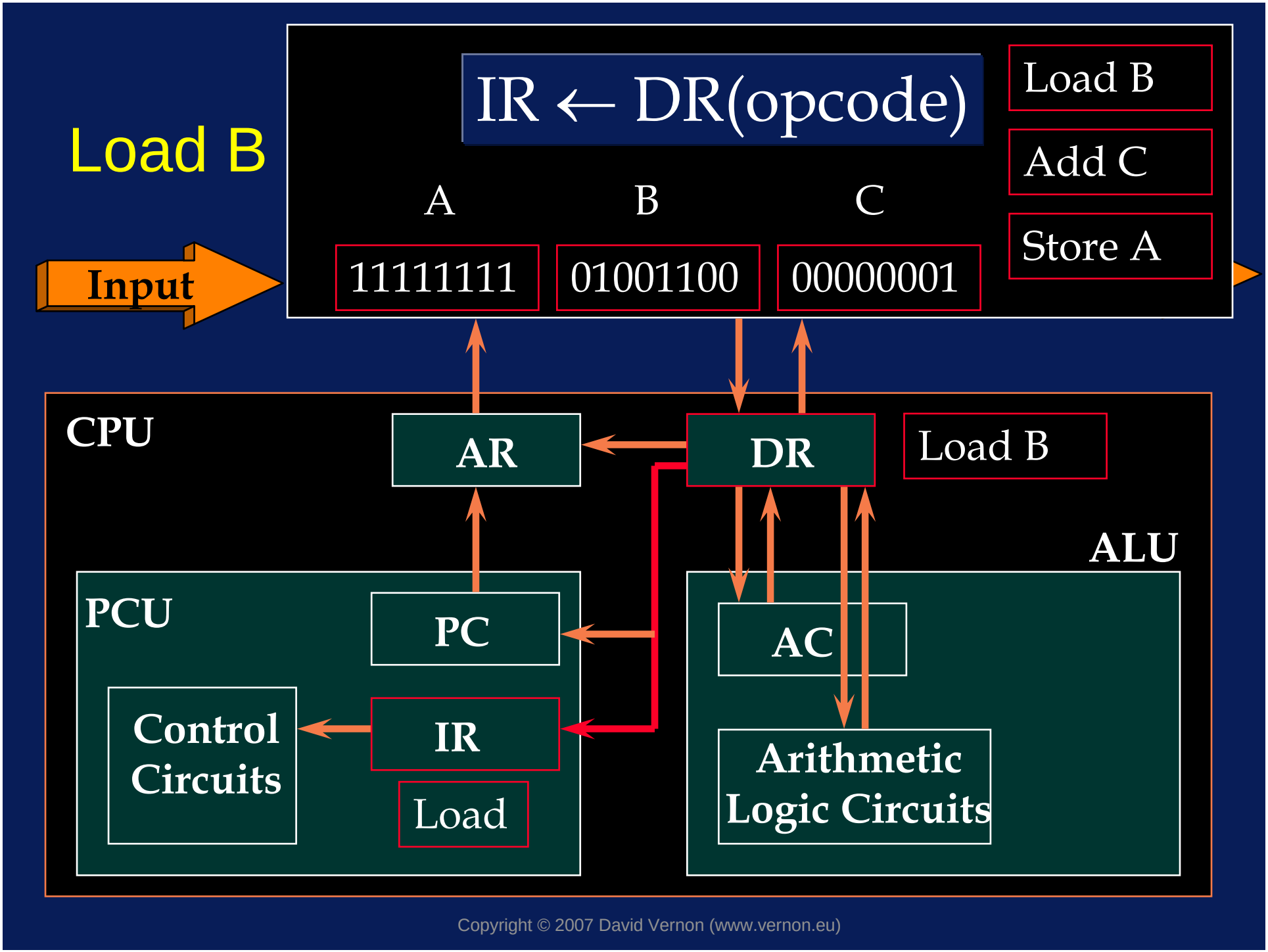
- Add C
  - AR ← PC
  - DR ← M(AR)
  - IR← DR(opcode)
  - Increment PC
  - Decode instruction in IR
  - AR ← DR(address)
  - DR ← M(AR)
  - AC ← AC + DR

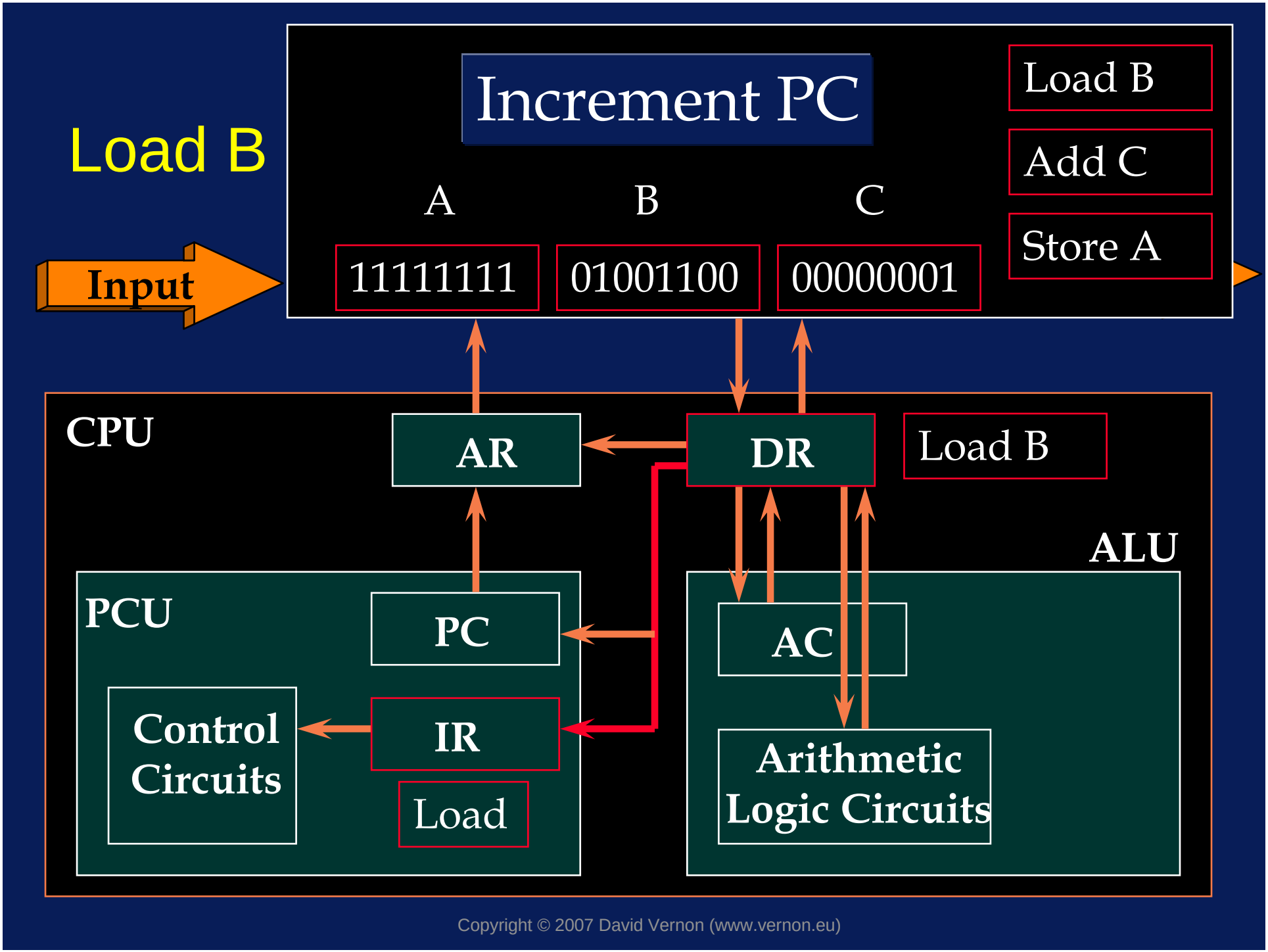
A := B + C

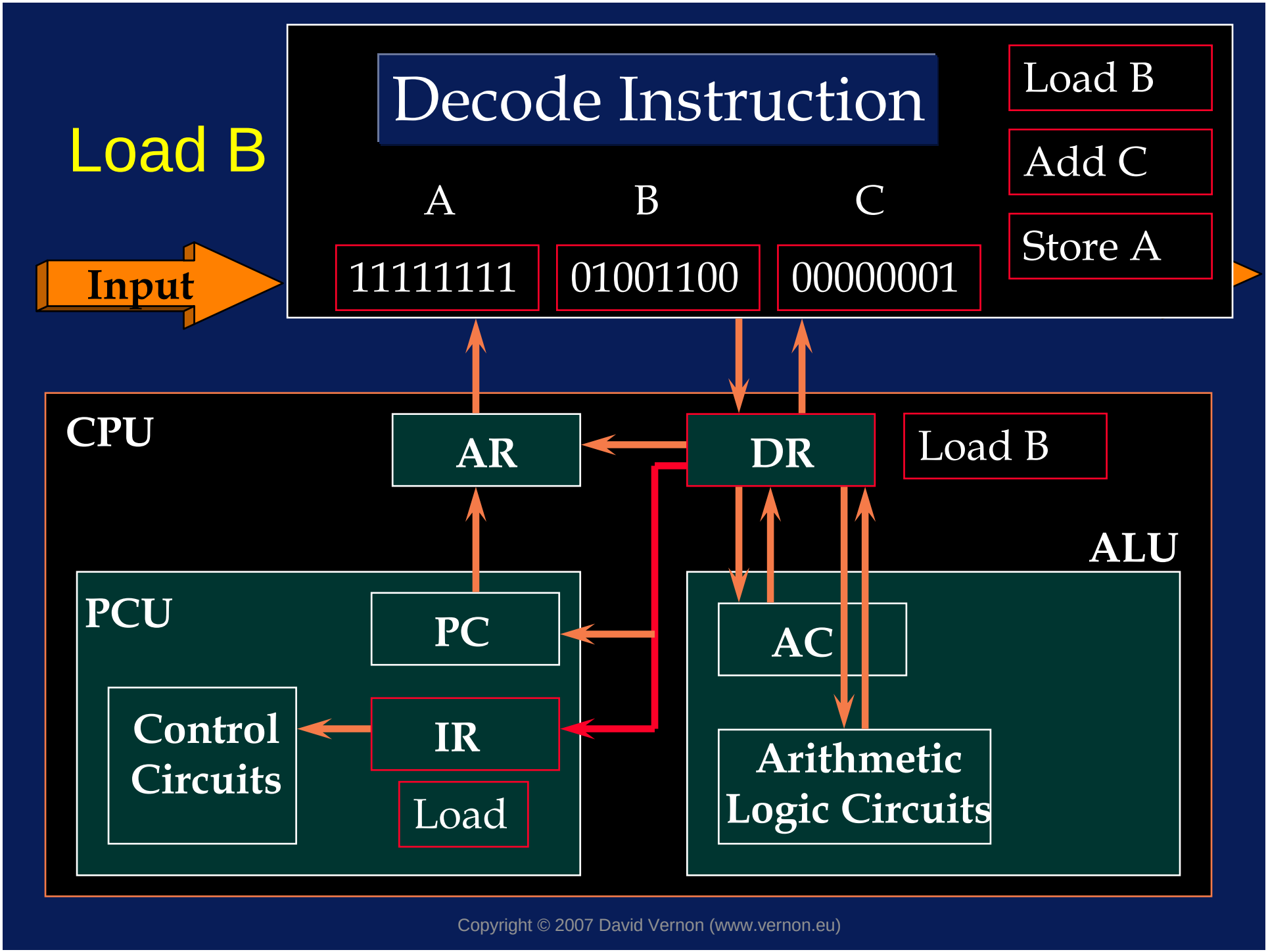
- Store A
  - AR ← PC
  - DR ← M(AR)
  - IR← DR(opcode)
  - Increment PC
  - Decode instruction in IR
  - AR ← DR(address)
  - DR ← AC
  - M(AR) ← DR

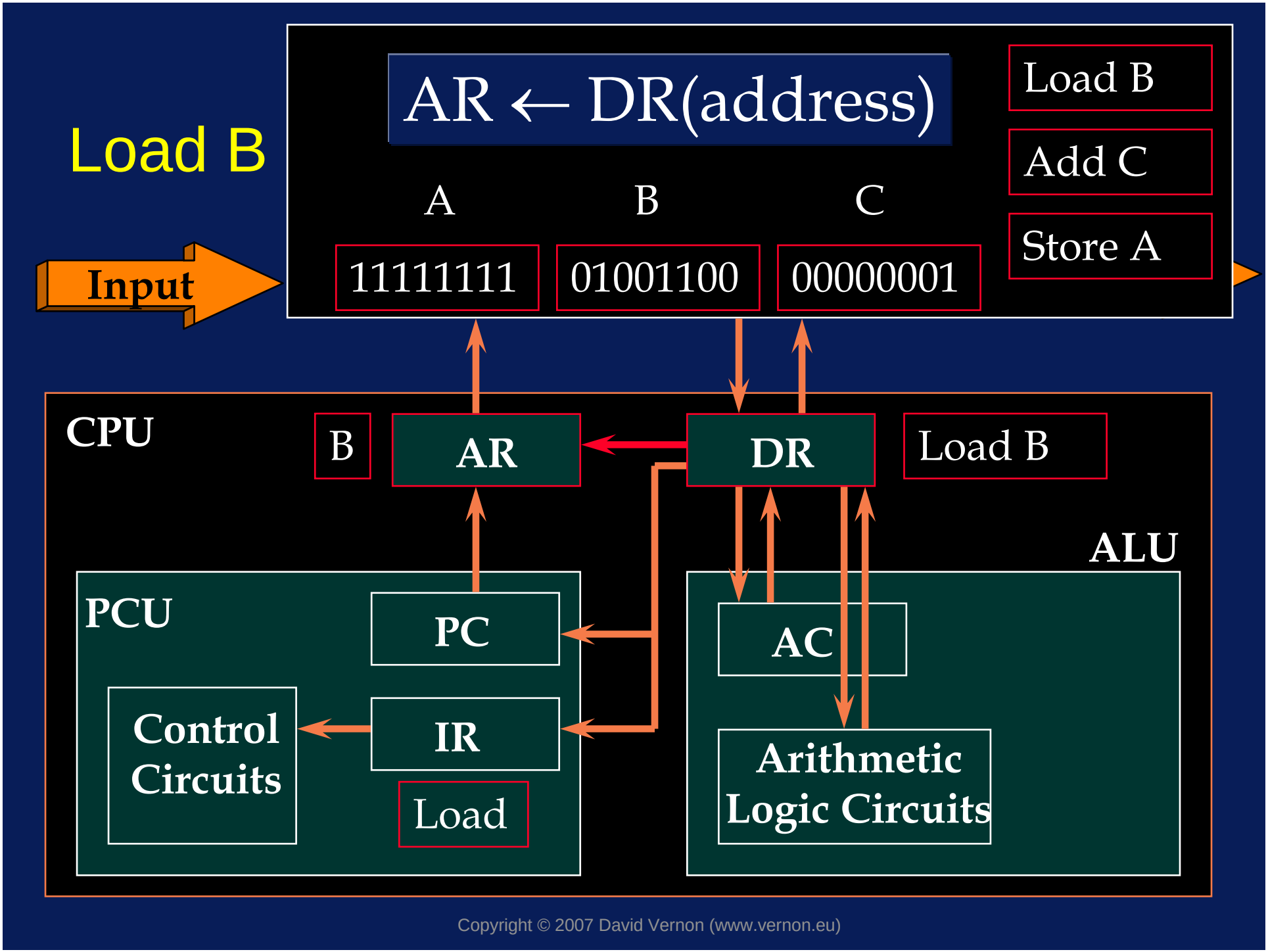


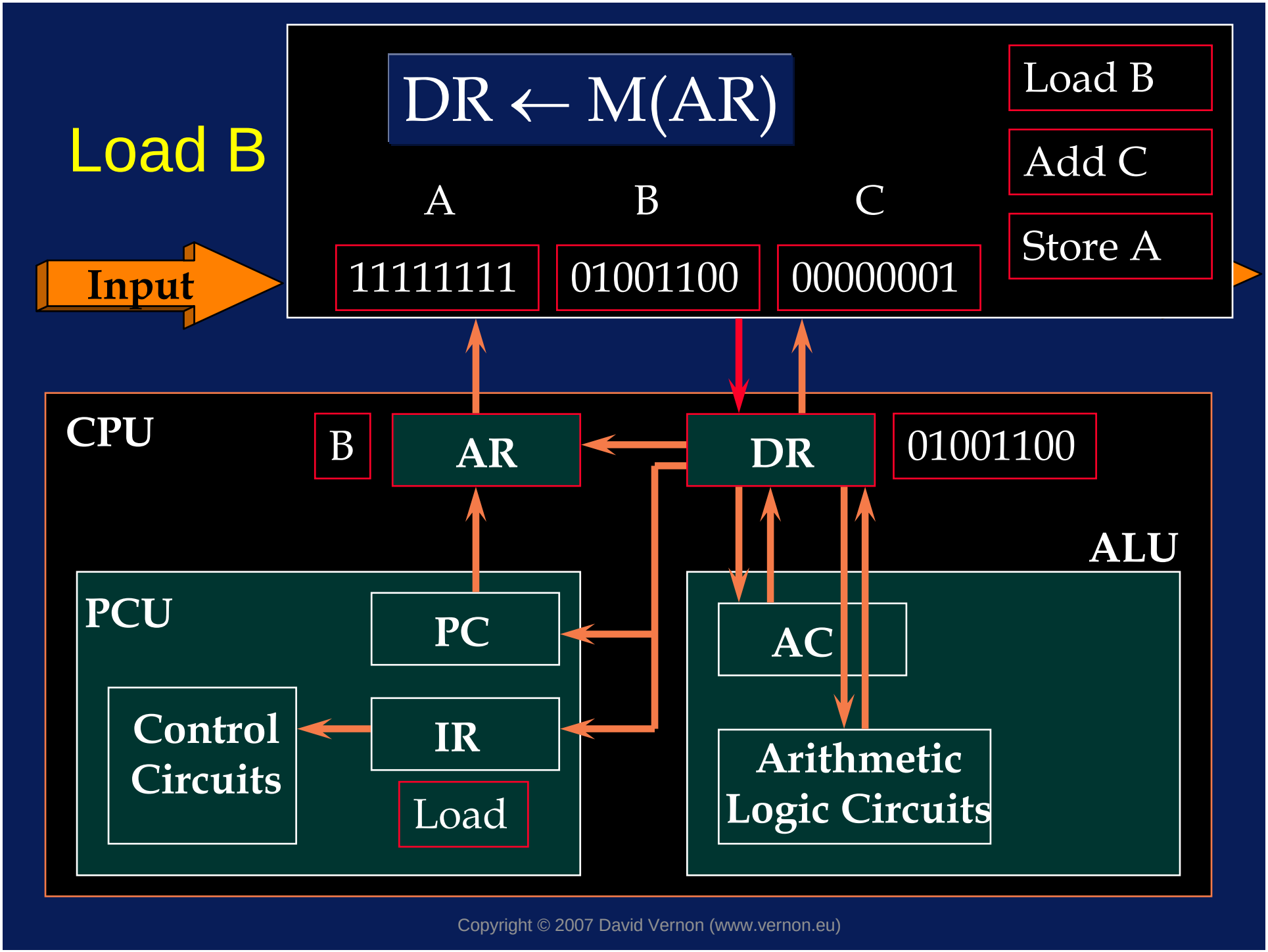


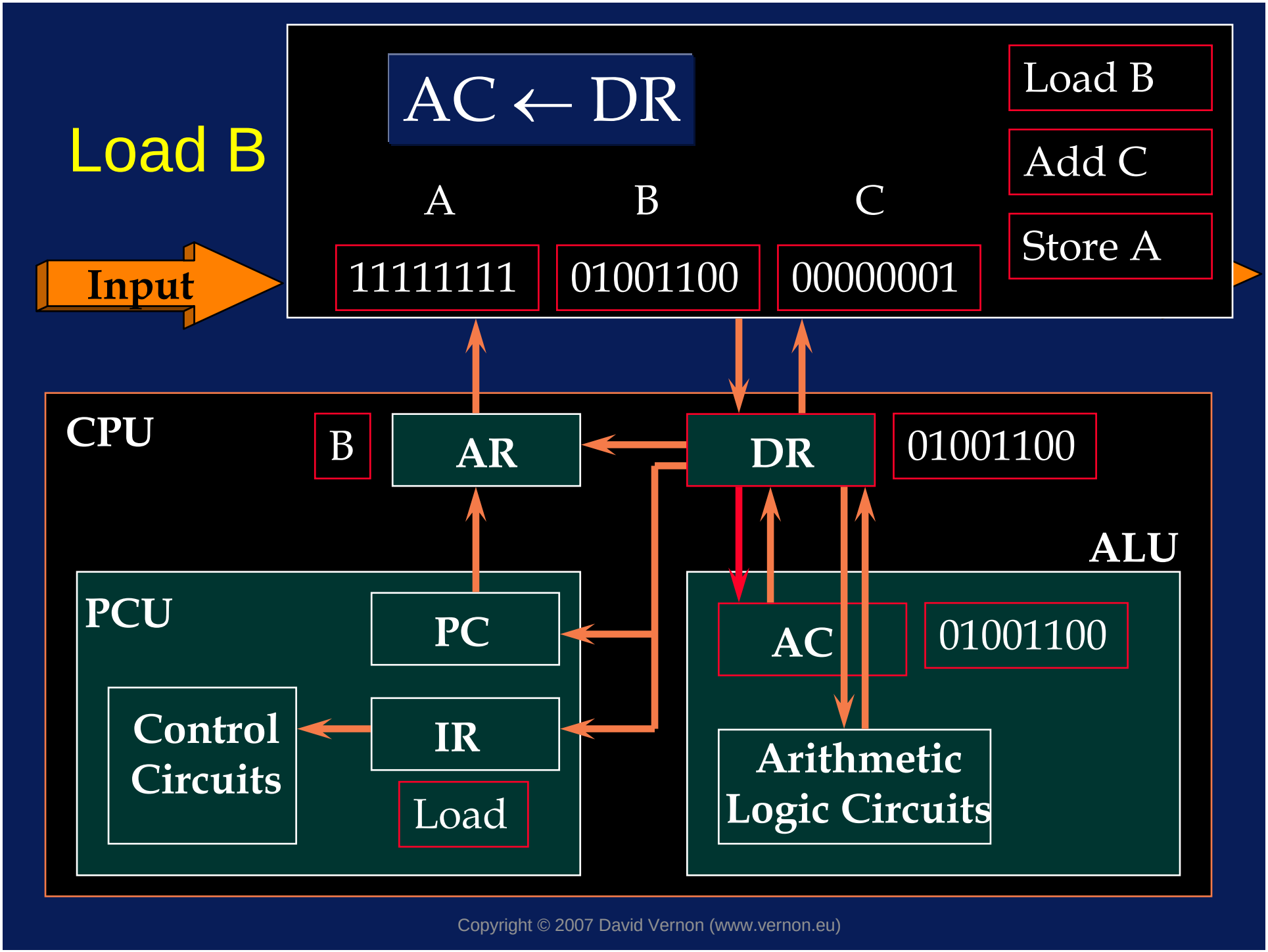












Extensions to the Basic Organization
and
Binary Number Representations

Extensions

- Additional addressable registers for storing operands and addresses
- If these are multipurpose, we have what is called a General Register Organization
- Sometimes special additional registers are provided for the purpose of memory address construction (e.g. index register)

Extensions

- The capabilities of the ALU circuits can be extended to include multiplication and division
- The ALU can process floating point (real) numbers as well as integers
- Additional registers can be provided for storing instructions (instruction buffer)

Extensions

- Special circuitry to facilitate temporary transfer to subroutines or interrupt handling programs and recovery of original status of interrupted program on returning from interrup handler

  e.g. the use of a ‘push-down stack’ implies that we need only a special-purpose ‘stack pointer’ register

Extensions

- Parallel processing

  Simultaneous processing of two or more distinct instructions or data streams

Information Representation

- Types of data
  - Text
  - Numbers
    - » Integers
    - » Reals (floating point numbers)

Text

- ASCII code (American Standard Committee on Information Interchange)
- A unique 8-bit binary code for each character:
  - A-Z, a-z, 1-9, .,¬!”£$$%^&*()_+
  - Special unprintable characters such as the ENTER key (CR for carriage return)

Numbers

- Binary number representation of integers
- If we save one bit to signify positive (+) or negative (-), then an n-bit binary word can represent integers in the range

$-2^{n-1}-1 \;..\; +2^{n-1}$

Numbers

- For example, a 16-bit binary number can represent integers in the range

$-2^{16-1}-1 \;..\; +2^{16-1} = -2^{15}-1 \;..\; +2^{15}$

-32,767 .. +32,768

Numbers

- A 16-bit binary number

0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1

Numbers

- A 16-bit binary number

```
15 14 13 12  11 10  9  8   7  6  5  4   3  2  1  0
 0  0  0  0   0  0  0  1   1  0  0  1   1  1  1  1
```

- $= 1 \times 2^{8} + 1 \times 2^{7} + 1 \times 2^{4} + 1 \times 2^{3} + 1 \times 2^{2} + 1 \times 2^{1} + 1 \times 2^{0}$
- $= 256 + 128 + 16 + 8 + 4 + 2 + 1$
- $= 415$

Numbers

- If we let the most significant bit (MSB) signify positive or negative numbers
  (1 for negative; 0 for positive)
- Then
  +9 = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
  -9 = 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
- +9 + (-9) = 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0
- Which is NOT zero .... a problem!

Numbers

- So we need a different representation for negative numbers
- Called 2s-complement
- Take the 1s-complement of the positive number:

  0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1

  becomes

  1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0

Numbers

- Note that the addition of the 1s-complement and the original number is not zero

```
0 0 0 0  0 0 0 0  0 0 0 0  1 0 0 1  +
1 1 1 1  1 1 1 1  1 1 1 1  0 1 1 0
-----------------------------------
1 1 1 1  1 1 1 1  1 1 1 1  1 1 1 1
```

Numbers

- To get the 2s-complement, add 1

```
 1 1 1 1  1 1 1 1  1 1 1 1  0 1 1 0  +
                            1
-------------------------------------
 1 1 1 1  1 1 1 1  1 1 1 1  0 1 1 1
```

Numbers

- Now do the addition:

```
0 0 0 0  0 0 0 0  0 0 0 0  1 0 0 1  +
1 1 1 1  1 1 1 1  1 1 1 1  0 1 1 1
-----------------------------------
0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0
```

Numbers

- Another example:

+1 + (-1)

```
0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 1 +
1 1 1 1  1 1 1 1  1 1 1 1  1 1 1 1
-----------------------------------
0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0
```

Numbers

- Short-hand for all these 1s and 0s
- HEX notation
- Each group of 4 bits represents a number in the range 0 - 15

Numbers

0 0 0 0 =	0
0 0 0 1 =	1
0 0 1 0 =	2
0 0 1 1 =	3
0 1 0 0 =	4
0 1 0 1 =	5
0 1 1 0 =	6
0 1 1 1 =	7
1 0 0 0 =	8
1 0 0 1 =	9
1 0 1 0 =	A
1 0 1 1 =	B
1 1 0 0 =	C
1 0 0 1 =	D
1 1 1 0 =	E
1 1 1 1 =	F

Numbers

- Thus:

0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1	0009
1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1	FFF7
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0000

Numbers

- And:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0001
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	FFFF
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0000

Numbers

- Hex is used as a notation for any sequence of bits (e.g. ASCII characters require just two hex digits)

Digital Design

Design Hierarchy

- Many digital systems can be divided into three design levels that form a well-defined hierarchy

Design Hierarchy

- The Architecture Level
  High-level concerned with overall system management
- The Logic Level
  Intermediate level concerned with the technical details of the system
- The Physical Level
  Low level concerned with the details needed to manufacture or assemble the system

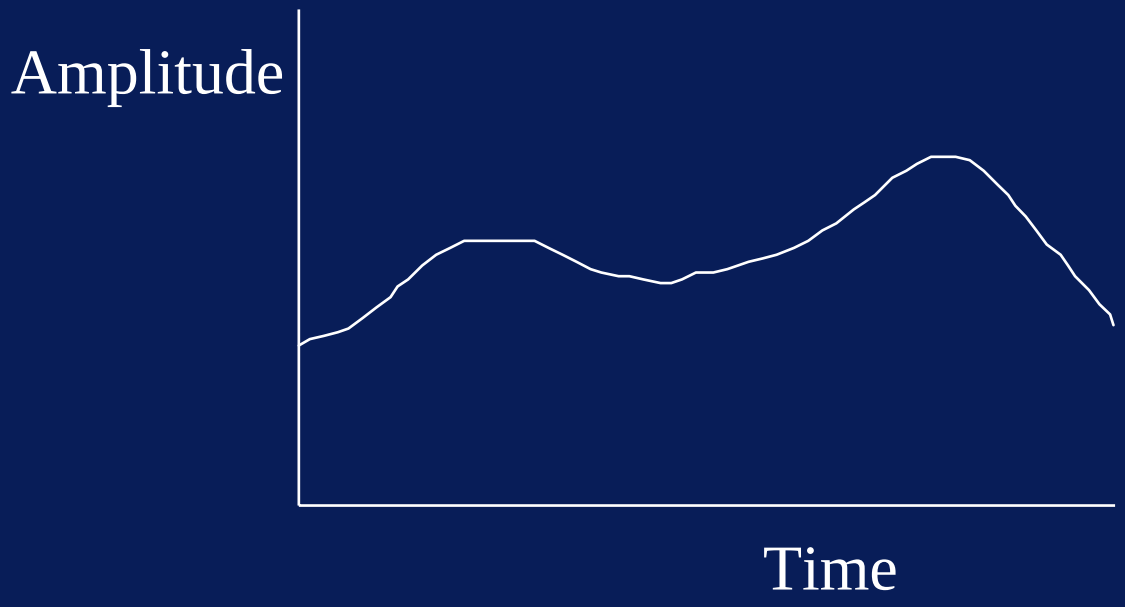
Design Hierarchy

- We have already studied the architecture level
- Now we will address the logic level
- At the logic level, there are two classes of digital system
  - Combinational - digital systems without memory
  - Sequential - digital systems with memory

Analogue and Digital Signals

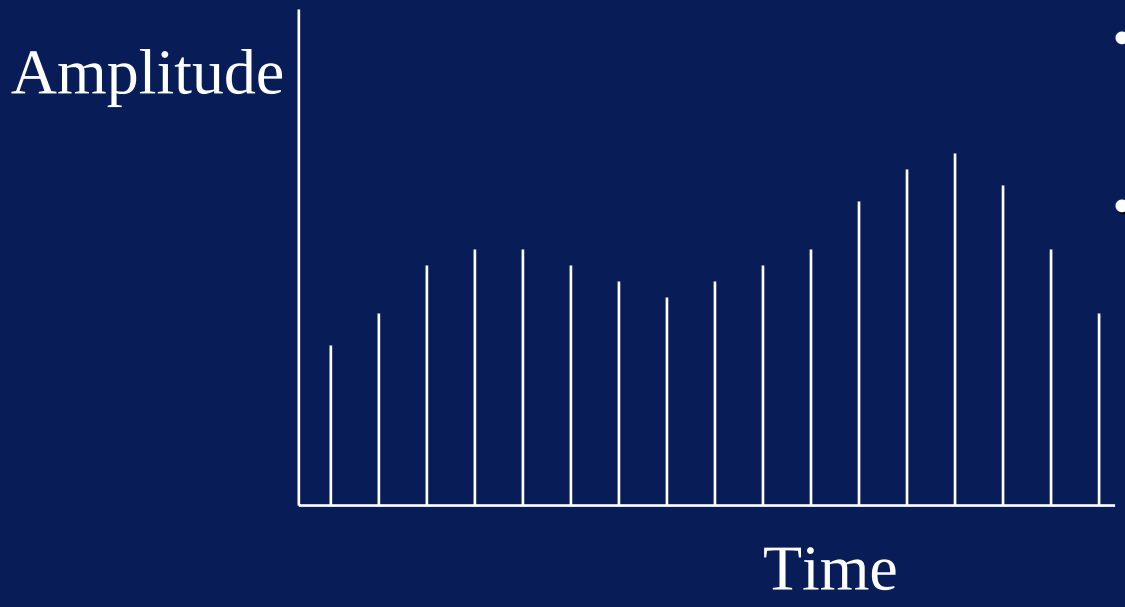
- An analogue signal can have any value within certain operating limits
- For example, in a (common emitter) amplifier, the output (O/P) can have any value between 0v and 10v.
- A digital signal can only have a fixed number of values within certain tolerances

Analogue and Digital Signals



- An analogue signal
- The amplitude is defined at all moments in time

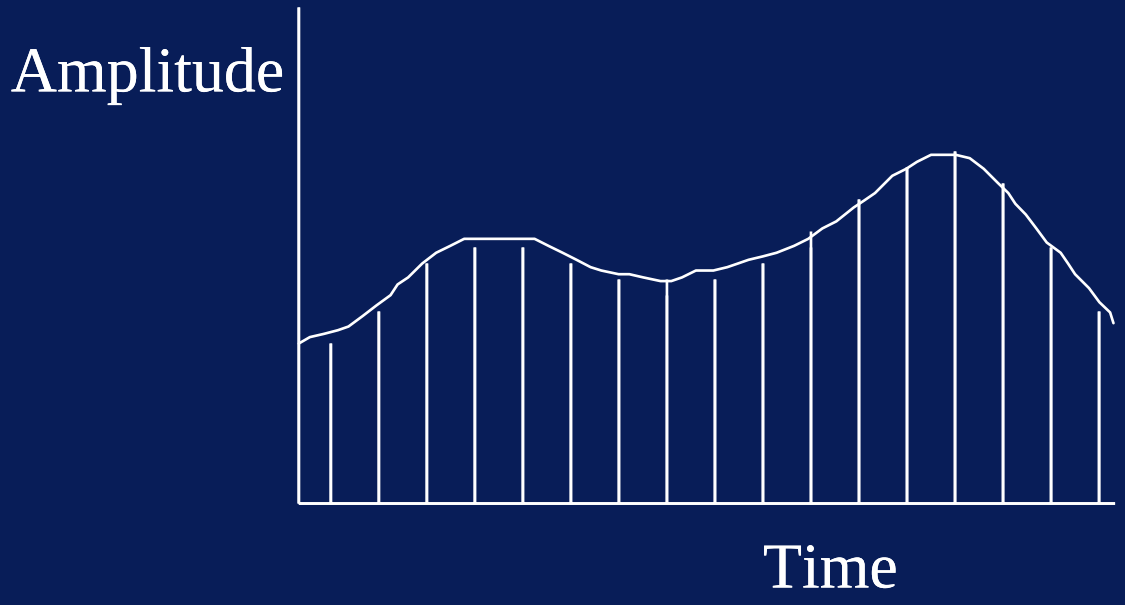
Analogue and Digital Signals



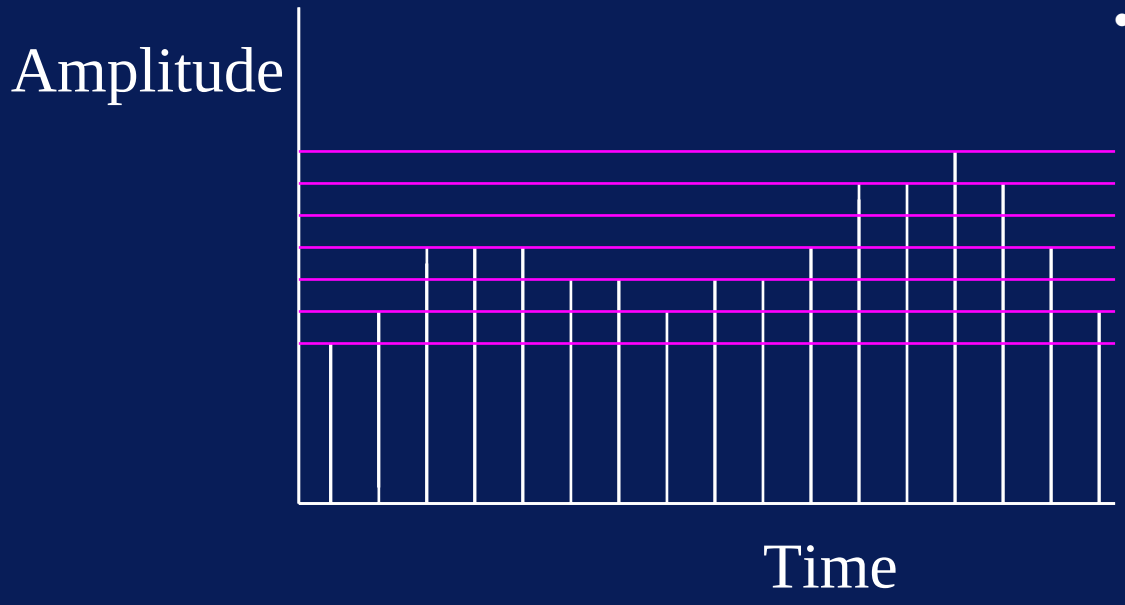
- A digital signal
- It is a sampled version of the analogue signal
- Only defined at certain discrete times
- DISCRETE TIME SIGNAL

Analogue and Digital Signals

- A digital signal is a sampled version of the analogue signal



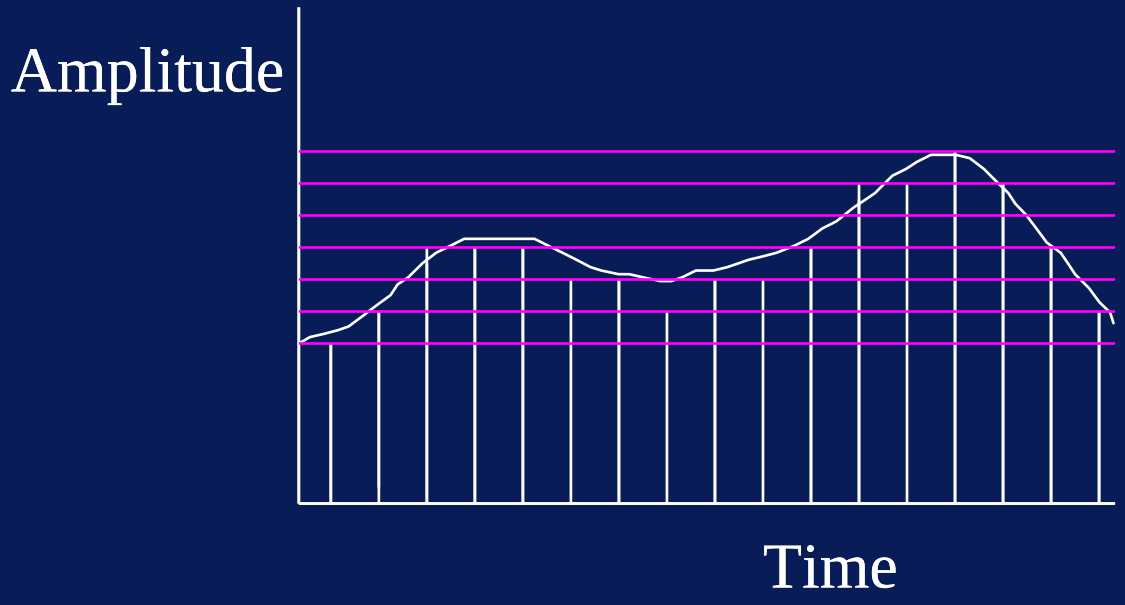
Analogue and Digital Signals



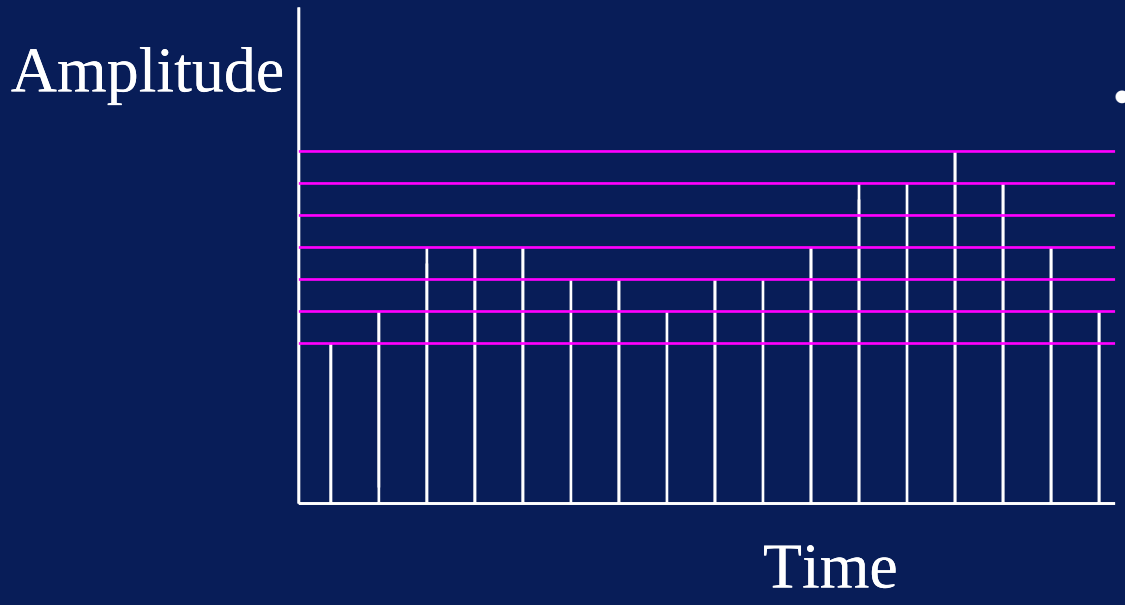
- The amplitude may also be restricted to take on discrete values only
- In which case it is said to be quantized

Analogue and Digital Signals

- Quantization introduces errors which depend on the step size or the resolution



Analogue and Digital Signals



- Signals (voltages or currents) which are samples and quantized are said to be DIGITAL
- They can be represented by a sequence of numbers

Analogue and Digital Signals

- Calculation with numbers is usually done in base 10 arithmetic
- Easier to effect machine computation in base 2 or binary notation
- We can also use base 2 or binary notation to represent logic values: TRUE and FALSE
- Manipulation of these (digital) logic values is subject to the laws of logic as set out in the formal rules of Boolean algebra

Boolean Algebra

- Definition: a logic variable *x* can have only one of two possible values or states

  x = TRUE

  x = FALSE
- In binary notation, we can say

  x = TRUE = 1

  x = FALSE = 0
- This is called positive logic or high-true logic

Boolean Algebra

- We could also say

  x = TRUE = 0

  x = FALSE = 1
- This is called negative logic or low-true logic
- Usually we use the positive logic convention

Boolean Algebra

- Electrically,
  - 1 is represented by a more positive voltage than zero and
  - 0 is represented by zero volts

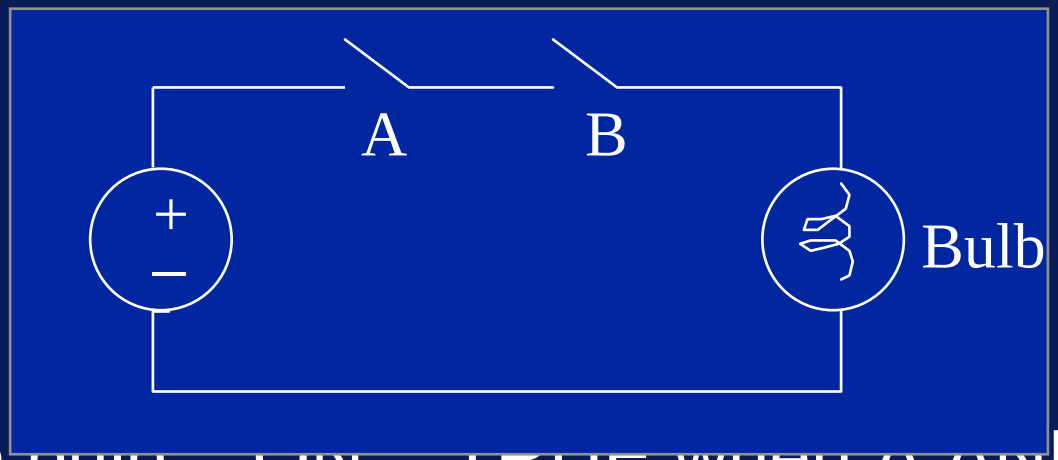
- *x* = TRUE = 1 = 5 volts
  x = FALSE = 0 = 0 volts

Logic Gates

- Logic gates are switching circuits that perform certain simple operations on binary signals
- These operations are chosen to facilitate the implementation of useful functions

Logic Gates

- The AND logic operation
- Consider the following circuit



- The bulb – ON – TRUE when A AND B are TRUE (i.e. closed)

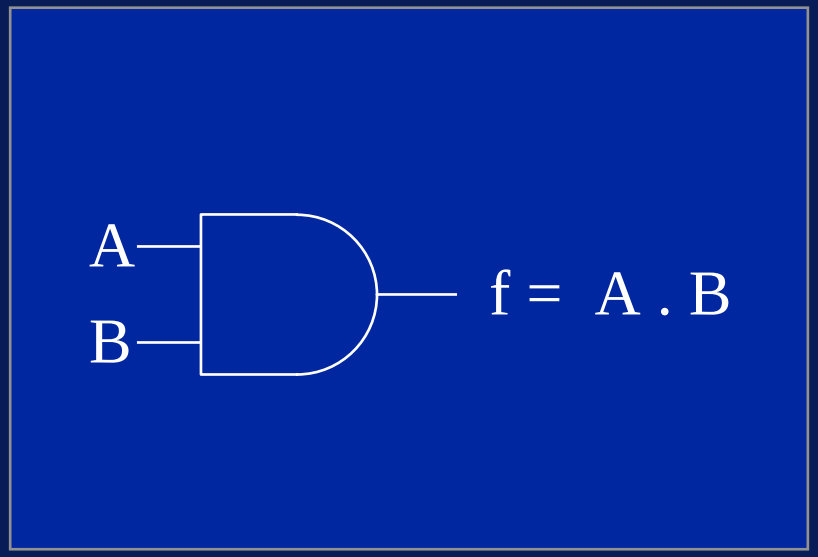
Logic Gates

- The AND Truth Table

A	B	f = A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Logic Gates

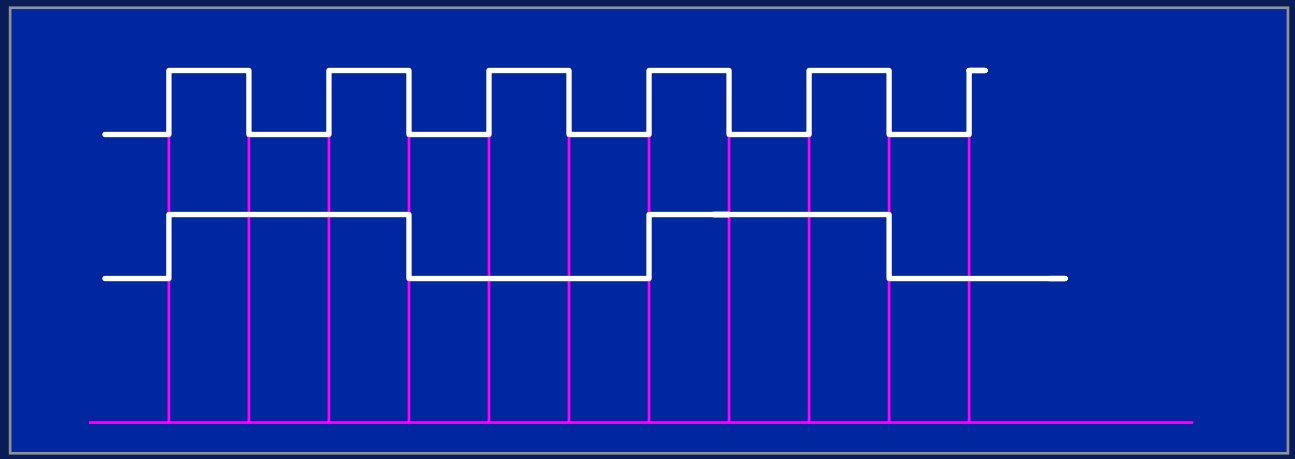
- The AND Gate



A and B are variables and note the use of the . to denote AND

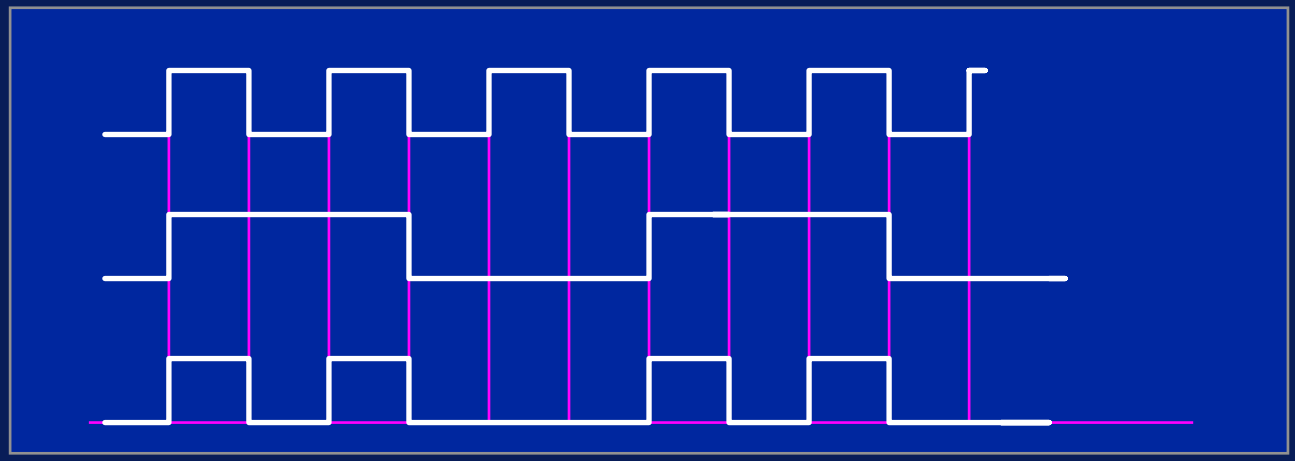
Logic Gates

- The AND Gate - An example
- Determine the output waveform when the input waveforms A and B are applied to the two inputs of an AND gate



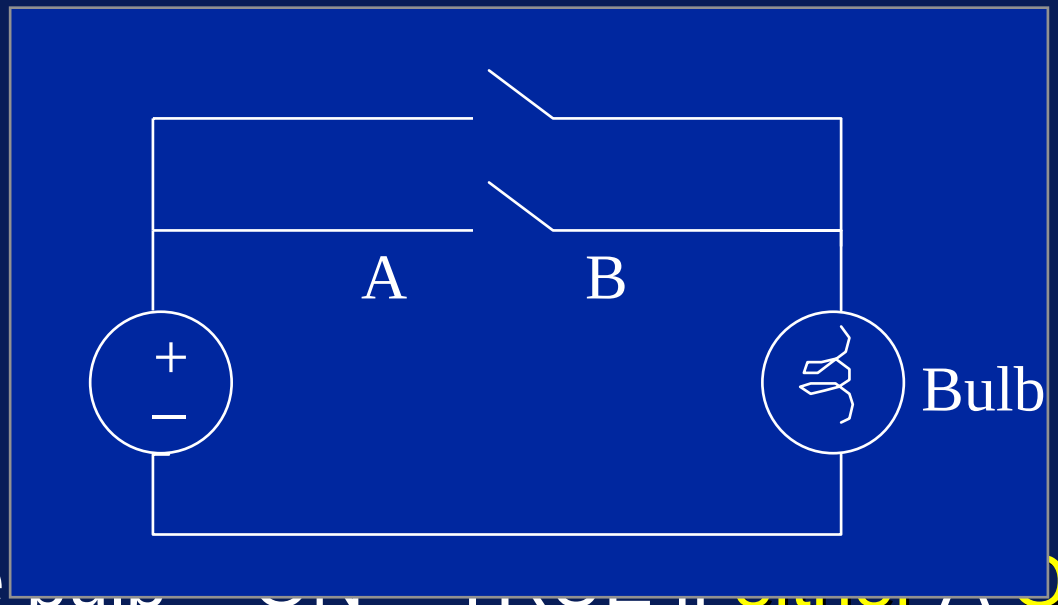
Logic Gates

- The AND Gate - An example
- Determine the output waveform when the input waveforms A and B are applied to the two inputs of an AND gate



Logic Gates

- The OR logic operation



- The bulb is ON = TRUE if either A OR B are TRUE (i.e. closed)

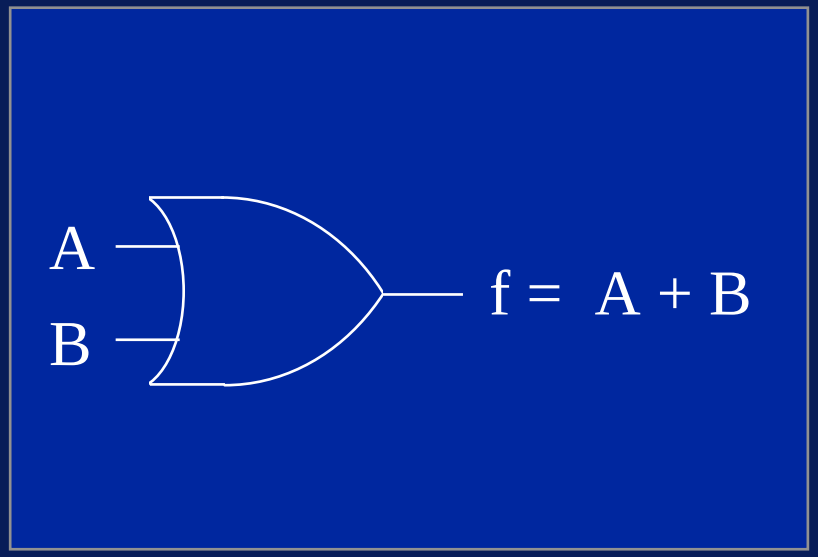
Logic Gates

- The OR Truth Table

A	B	f = A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Logic Gates

- The OR Gate



A and B are variables and note the use of the + to denote OR

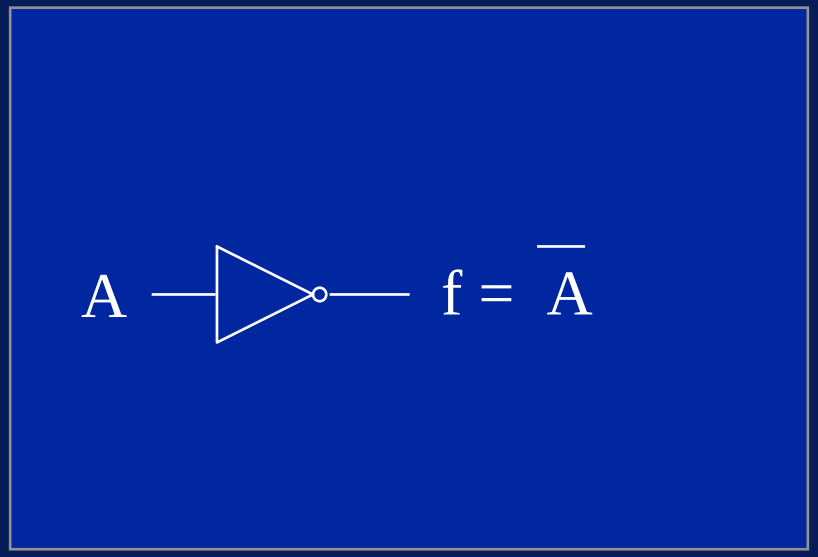
Logic Gates

- The NOT Truth Table

A	f = NOT A
0	1
1	0

Logic Gates

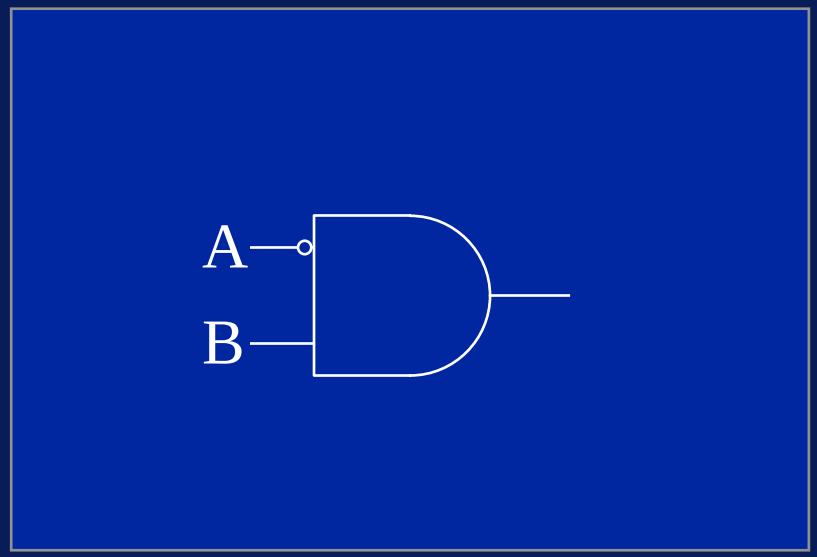
- The NOT Gate



Note the use of the bar over the A to denote NOT

Logic Gates

- Sometimes a ‘bubble’ is used to indicate inversion



Logic Gates

- In fact it is simpler to manufacture the combination NOT AND and NOT OR than it is to deal with AND and OR
- NOT AND becomes NAND
- NOT OR becomes NOR

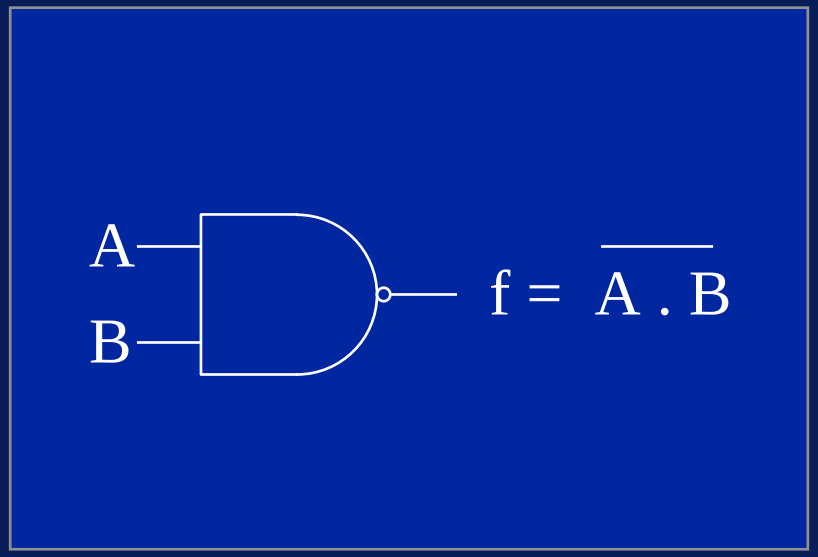
Logic Gates

- The NAND Truth Table

A	B	f = A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

Logic Gates

- The NAND Gate



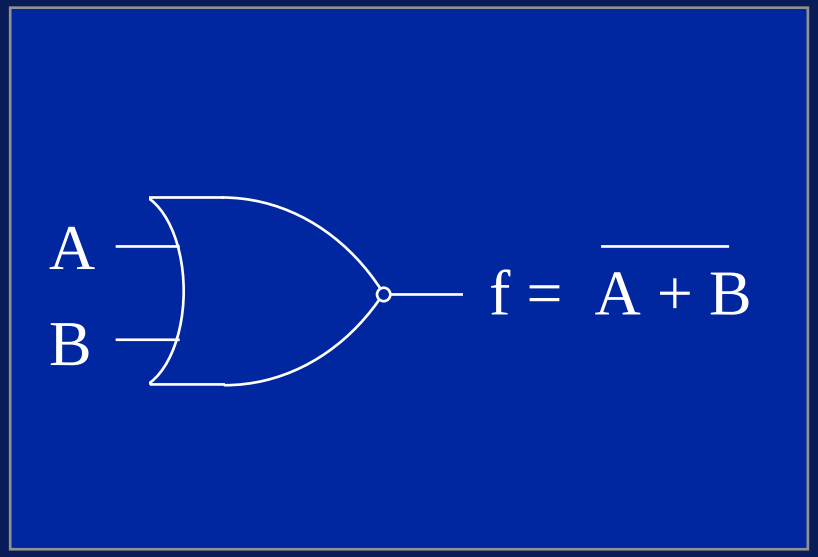
Logic Gates

- The NOR Truth Table

A	B	f = A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

Logic Gates

- The NOR Gate



Logic Gates

- The EXCLUSIVE OR Truth Table

f = A XOR B

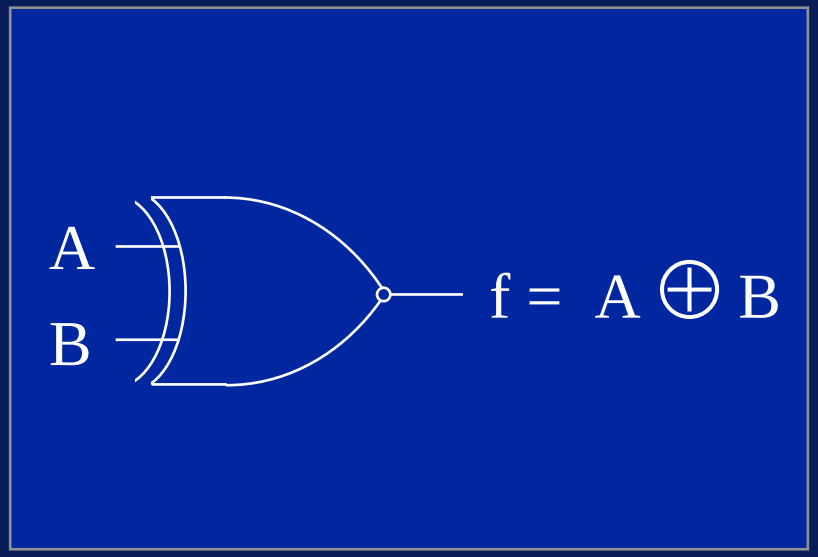
= $A \oplus B$

= $A\overline{B} + \overline{A}B$

A	B	f = A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Logic Gates

- The XOR Gate



Logic Gates

- The EXCLUSIVE NOR Truth Table

f = NOT (A XOR B)

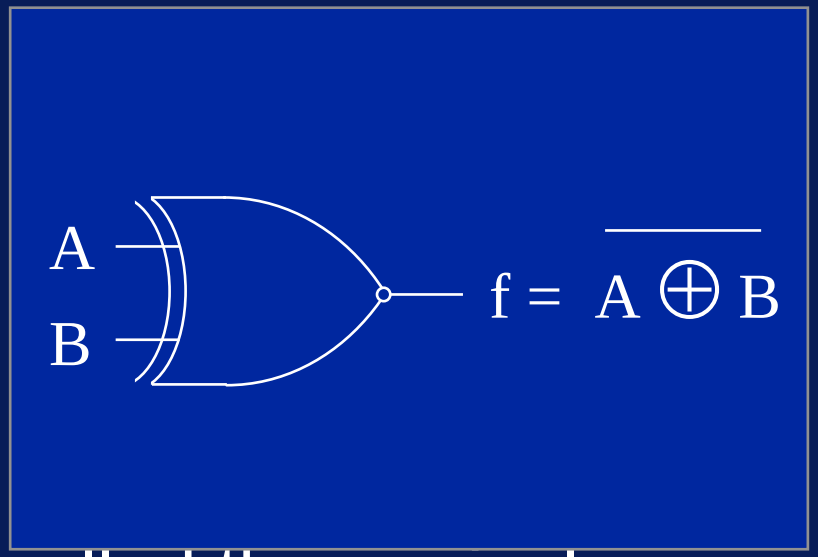
$= \overline{A \oplus B}$

$= \overline{AB} + AB$

A	B	f = A XOR B
0	0	1
0	1	0
1	0	0
1	1	1

Logic Gates

- The EXCLUSIVE NOR Gate



This is called the equivalence gate

Rules and Laws of Boolean Algebra

- Operations on Boolean variables are defined by rules and laws, the most important of which are presented here
- Commutative Law

  $A \cdot B = B \cdot A$

  $A + B = B + A$
- This states that the order of the variables is unimportant

Rules and Laws of Boolean Algebra

- Associative Law

  $A \cdot (B \cdot C) = A \cdot (B \cdot C)$

  $A + (B + C) = A + (B + C)$

- This states that the grouping of the variables is unimportant

Rules and Laws of Boolean Algebra

- Distributive Law

  $A . (B + C) = A . B + A . C$

- This states that we can remove the parenthesis by ‘multiplying through’

- The above laws are the same as in ordinary algebra, where ‘+’ and ‘.’ are interpreted as addition and multiplication

Rules and Laws of Boolean Algebra

- Basic rules involving one variable:

  $A + 0 = A \quad A \cdot 0 = 0$
  $A + 1 = 1 \quad A \cdot 1 = A$
  $A + A = A \quad A \cdot A = A$
  $A + \overline{A} = 1 \quad A + A = 0$

- It should be noted that $\overline{\overline{A}} = A$

Rules and Laws of Boolean Algebra

- An informal proof of each of these rules is easily accomplished by taking advantage of the fact that the variable can have only two possible values
- For example, rule 2:
  $A + 1 = 1$
- If $A = 0$ then $0 + 1 = 1$
- If $A = 1$ then $1 + 1 = 1$

Rules and Laws of Boolean Algebra

- Some useful theorems

$A + A.B = A$

$A + \overline{A}.B = A + B$

$A.B + A.\overline{B} = A$

$A(A + B) = A$

$A(\overline{A} + B) = A.B$

$(A+B)(A+\overline{B}) = A$

These may be proved in a similar manner

Rules and Laws of Boolean Algebra

- For example: $A + A.\overline{B} = A+B$

A	B	$\overline{A}$	$\overline{A}.B$	$A+\overline{A}.B$	A+B
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

Rules and Laws of Boolean Algebra

- Finally, we come to DeMorgan's Laws which are particularly useful when dealing with NAND and NOR logic
- They are stated as follows

DeMorgan's Laws (1)

- $\overline{A + B} = \overline{A} \cdot \overline{B}$
- Read as:
  - NOT (A OR B) = NOT A AND NOT B
  - A NOR B = NOT A AND NOT B
- Relates NOT, OR, and AND
- Can be extended to any number of variables

$$\overline{A + B + C \ldots} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \ldots$$

DeMorgan's Laws (1)

- $\overline{A + B} = \overline{A} \cdot \overline{B}$

A	B	A + B	$\overline{A + B}$	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0	0				
0	1	1				
1	0	1				
1	1	1				

DeMorgan's Laws (1)

- $\overline{A + B} = \overline{A} \cdot \overline{B}$

A	B	A + B	$\overline{A + B}$	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0	0	1			
0	1	1	0			
1	0	1	0			
1	1	1	0			

DeMorgan's Laws (1)

- $\overline{A + B} = \overline{A} \cdot \overline{B}$

A	B	A + B	$\overline{A + B}$	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0	0	1	1		
0	1	1	0	1		
1	0	1	0	0		
1	1	1	0	0		

DeMorgan's Laws (1)

- $\overline{A + B} = \overline{A} \cdot \overline{B}$

A	B	A + B	$\overline{A + B}$	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0	0	1	1	1	
0	1	1	0	1	0	
1	0	1	0	0	1	
1	1	1	0	0	0	

DeMorgan's Laws (1)

- $\overline{A + B} = \overline{A} \cdot \overline{B}$

A	B	A + B	$\overline{A + B}$	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

DeMorgan's Laws (2)

- $\overline{A \cdot B} = \overline{A} + \overline{B}$
- Read as:
  - NOT (A AND B) = NOT A OR NOT B
  - A NAND B = NOT A OR NOT B
- Relates NOT, OR, and AND
- Can be extended to any number of variables

$$\overline{A \cdot B \cdot C \ldots} = \overline{A} + \overline{B} + \overline{C} \ldots .$$

DeMorgan's Laws (2)

- $\overline{A \cdot B} = \overline{A} + \overline{B}$

A	B	$A \cdot B$	$\overline{A \cdot B}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	0				
0	1	0				
1	0	0				
1	1	1				

DeMorgan's Laws (2)

- $\overline{A \cdot B} = \overline{A} + \overline{B}$

A	B	$A \cdot B$	$\overline{A \cdot B}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	0	1			
0	1	0	1			
1	0	0	1			
1	1	1	0			

DeMorgan's Laws (2)

- $\overline{A \cdot B} = \overline{A} + \overline{B}$

A	B	A . B	$\overline{A \cdot B}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	0	1	1		
0	1	0	1	1		
1	0	0	1	0		
1	1	1	0	0		

DeMorgan's Laws (2)

- $\overline{A \cdot B} = \overline{A} + \overline{B}$

A	B	$A \cdot B$	$\overline{A \cdot B}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	0	1	1	1	
0	1	0	1	1	0	
1	0	0	1	0	1	
1	1	1	0	0	0	

DeMorgan's Laws (2)

- $\overline{A \cdot B} = \overline{A} + \overline{B}$

A	B	A . B	$\overline{A \cdot B}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

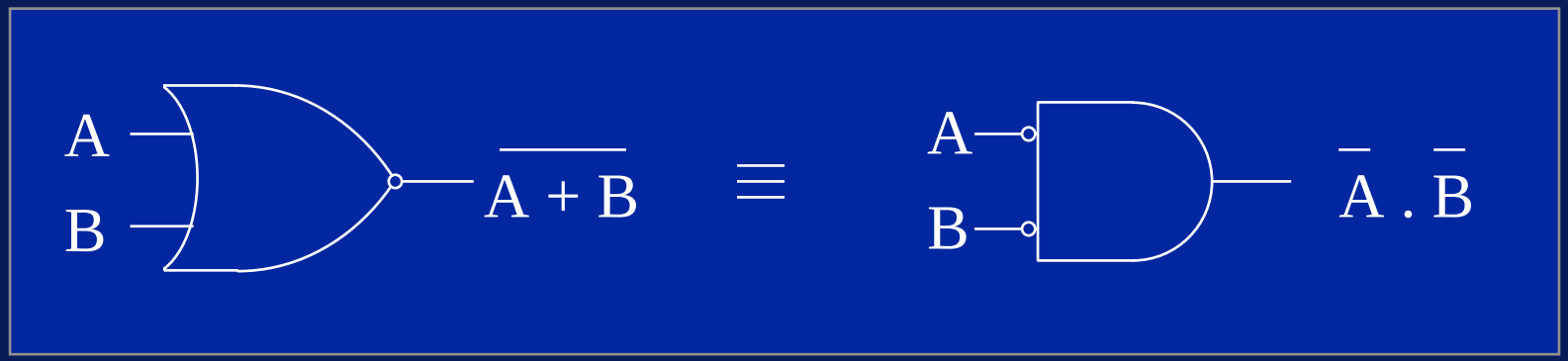
DeMorgan's Laws

- $\overline{A + B} = \overline{A} . \overline{B}$
- Let A be 'I won the Lotto'
- Let B be 'I'm happy'
- The the first DeMorgan Law tell us that:
  NOT (I won the Lotto OR I'm happy)
  is the same as
  NOT(I won the lotto) AND NOT(I'm happy)
  [or:
  I didn't win the lotto and I'm not happy]

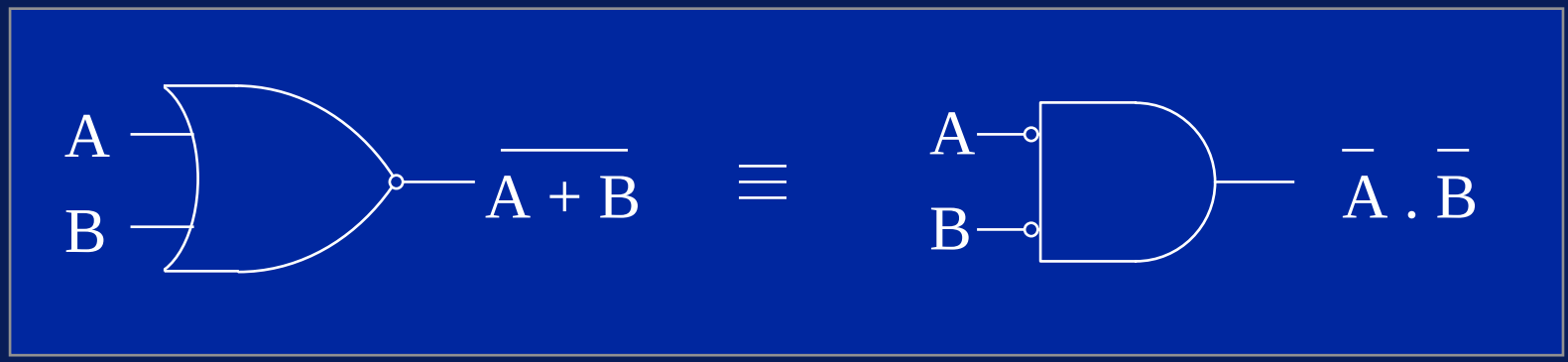
DeMorgan's Laws

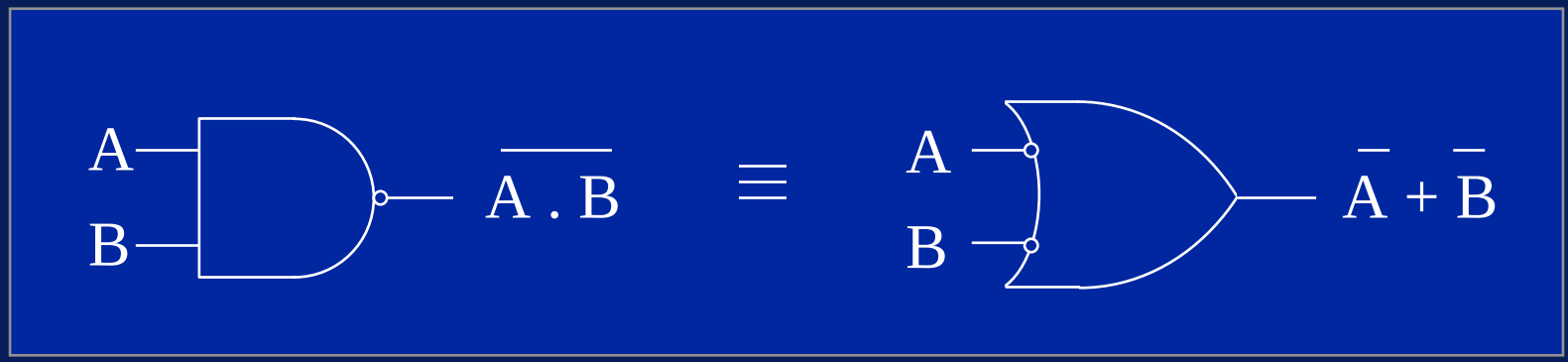
- $\overline{A \cdot B} = \overline{A} + \overline{B}$
- Let A be 'I won the Lotto'
- Let B be 'I'm happy'
- The the second DeMorgan Law tell us that:
  NOT (I won the Lotto AND I'm happy)
  is the same as
  NOT(I won the lotto) OR NOT(I'm happy)
  [or:
  I didn't win the lotto OR I'm not happy]

DeMorgan's Laws



DeMorgan's Laws

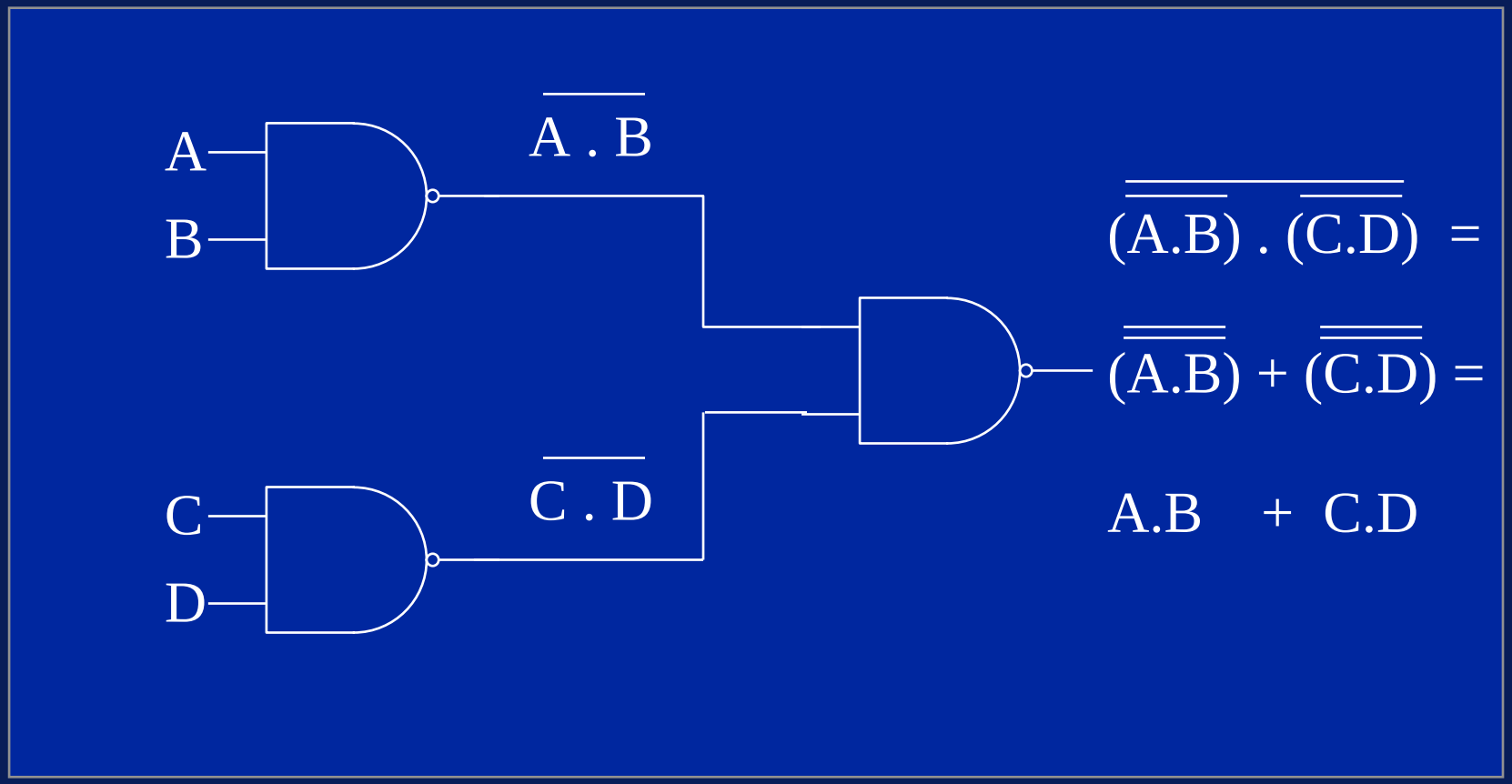




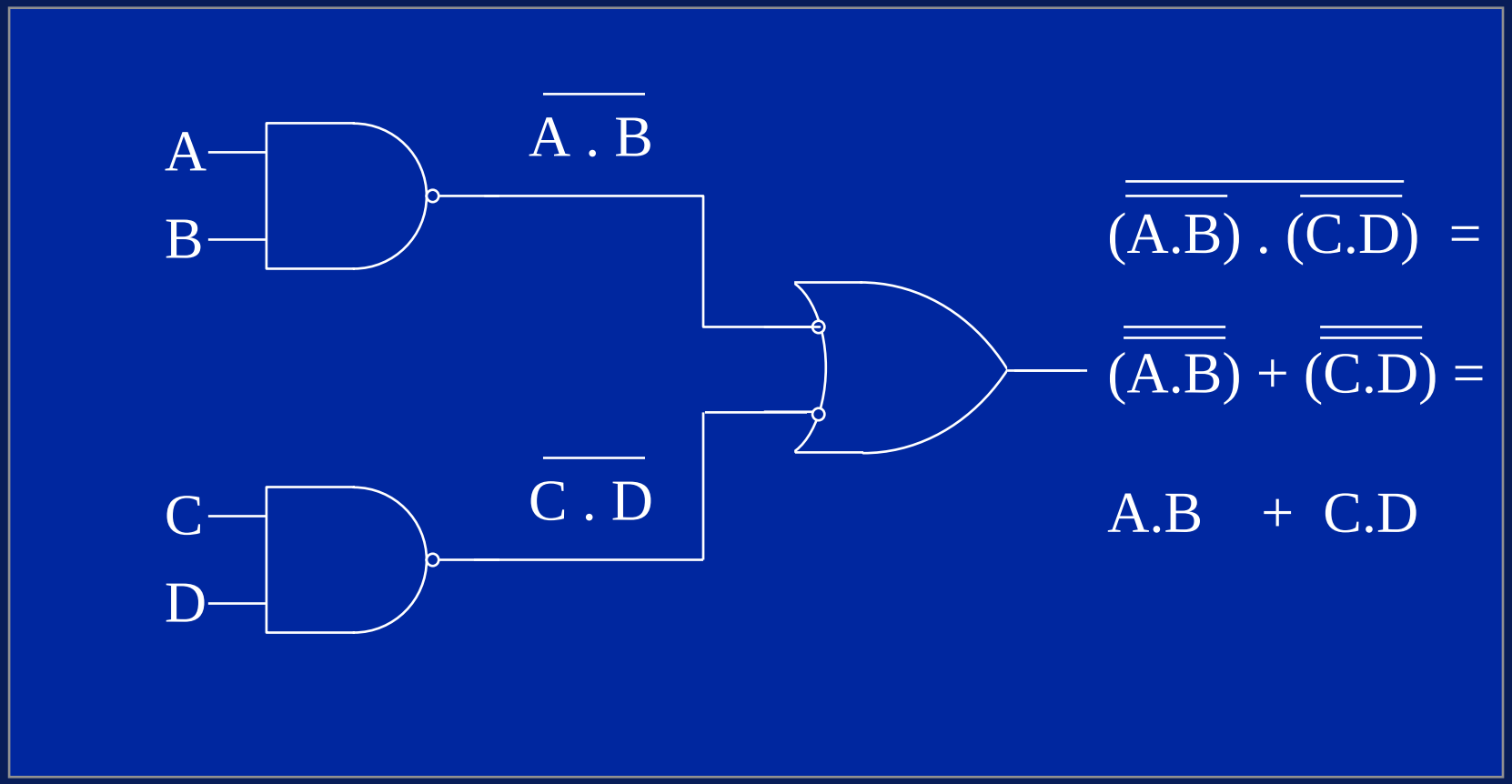
DeMorgan's Laws

- Taking the NAND gate as an example, we can derive effective AND-OR gating although physically we are using only one type of gate
- For example f = A.B + C.D

DeMorgan's Laws



DeMorgan's Laws



DeMorgan's Laws

- This is referred to as NAND/NAND gating
- Any logic equation may be implemented using NAND gates only
- Thus NAND gates may be regarded as univeral gates
- The same is true for NOR gates

DeMorgan's Laws

- Other advantages of using NAND or NOR gating are:
- Simplest and cheapest to fabricate
- Fastest operating speed
- Lowest power dissipation

Simplification of Expressions using Boolean Algebra

- It is important to minimise Boolean functions as this often brings about a reduction in the number of gates or inputs that are needed
- For example: consider

  AB + A(B+C) + B(B+C)

Simplification of Expressions using Boolean Algebra

- AB + A(B+C) + B(B+C)
- AB + AB + AC + BB + BC {distribution}
- AB + AC + BB + BC {X + X = X }
- AB + AC + B + BC {X . X = X }
- AB + B.1 + BC + AC {X . 1 = X }
- B(A+1+C) + AC {distribution}
- B.1 + AC {1 + X = 1 }
- B + AC {X . 1 = X }

Simplification of Expressions using Boolean Algebra

- Consider:
  (A$\underline{B}$(C + BD) + $\underline{AB}$)C
- (A$\underline{B}$C + A$\underline{B}$BD + $\underline{AB}$)C {distribution}
- A$\underline{B}$CC + A$\underline{B}$BCD + $\underline{AB}$C {distribution}
- A$\underline{B}$C + A$\underline{B}$BCD + $\underline{AB}$C {X.X = X }
- A$\underline{B}$C + A$\underline{0}$CD + $\underline{AB}$C {X.$\underline{X}$ = 0 }
- A$\underline{B}$C + $\underline{AB}$C {0.X = 0 }
- $\underline{B}$C(A + A) {distribution}
- $\underline{B}$C {X+X = 1 }

Simplification of Expressions using Boolean Algebra

- In general:
  ‘Multiply out and collect common terms’
- Exactly as you would do when simplifying ordinary algebraic expressions

Expression of Problems using Boolean Algebra

- Most ‘real’ problems are defined using a sentential structure.
- It is therefore necessary to translate such sentences into Boolean equations if we are to derive a digital circuit to give a Boolean result.

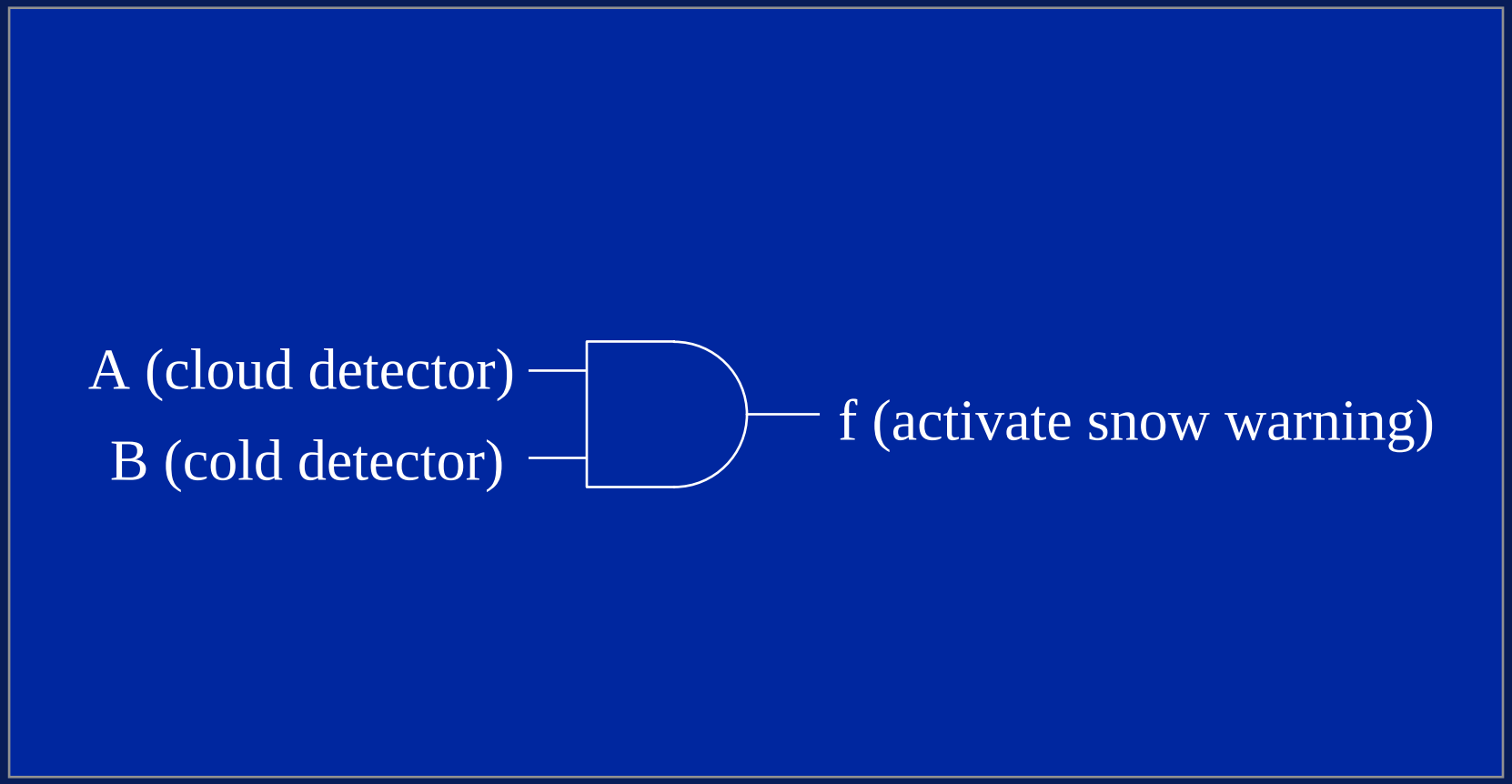
Expression of Problems using Boolean Algebra

- For example
  It will snow IF it is cloudy AND it is cold.
- The ‘IF’ and the ‘AND’ divide the sentence into different phrases

Let:

- f = ‘it will snow’ = 1, if true; 0 if false
- A = ‘it is cloudy’ = 1, if true; 0 if false
- B = ‘it is cold’ = 1, if true’ 0 if false
- So f = A.B

Corresponding Digital Circuit



Expression of Problems using Boolean Algebra

A slightly more complicated example:

- An alarm circuit is to be designed which will operate as follows
- The alarm will ring IF the alarm switch is on AND the door is open, or IF it is after 6pm AND the window is open

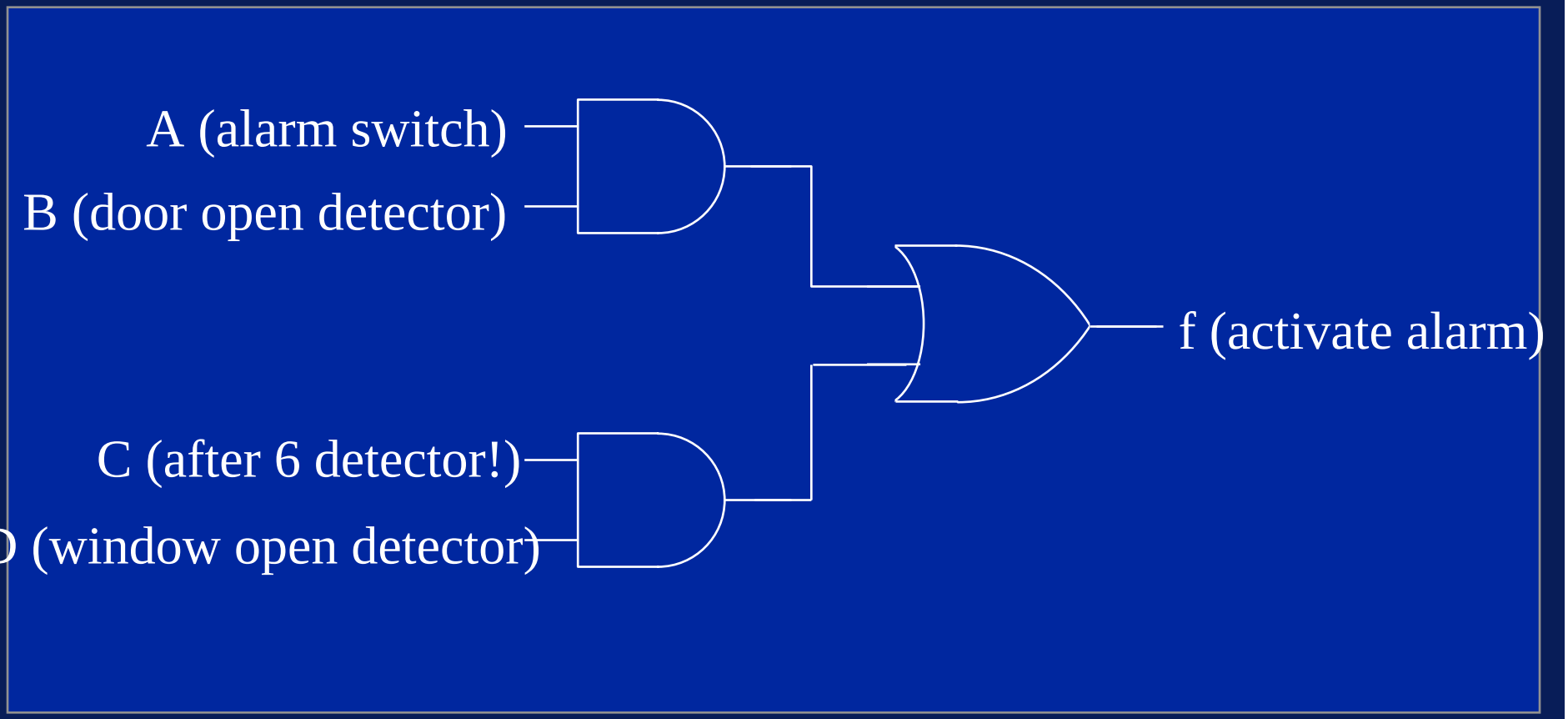
Expression of Problems using Boolean Algebra

- Let’s give the clauses some labels, just as before
- The alarm will ring (f)
  - IF the alarm switch is on (A)
  - AND the door is open (B),
  - or IF it is after 6pm (C)
  - AND the window is open (D)

Expression of Problems using Boolean Algebra

- f = ‘the alarm will ring’ = 1, if true; 0 if false
- A = ‘the alarm switch is on’ = 1, if true; 0 if false
- B = ‘the door switch is open’ = 1, if true; 0 if false
- C = ‘it is after 6 pm’ = 1, if true; 0 if false
- D = ‘the window switch is open’ = 1, if true; 0 if false
- So f = A.B + C.D
- If f = 1, then the alarm will ring!

Corresponding Digital Circuit



Expression of Problems using Boolean Algebra

- For more complicated problems, we define the problem coherently by constructing a truth table
- We'll introduce the idea for this simple example and then go on to use it in a more complicated example

Expression of Problems using Boolean Algebra

- If we have a problem (or a Boolean expression) with four variables, then our truth table will look like this:

Expression of Problems using Boolean Algebra

A	B	C	D
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Expression of Problems using Boolean Algebra

- Each combination of the logical variables A, B, C, and D make a 4-bit binary number in the range 0-15
- let’s number each row with the equivalent decimal number

Expression of Problems using Boolean Algebra

Row	A	B	C	D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Expression of Problems using Boolean Algebra

- We could also add the equivalent Boolean expression
- For example:

  0010 is equivalent to $\bar{A}.B.\bar{C}.\bar{D}$

- (in the following we will leave out the . for AND and just write $A\bar{B}\bar{C}\bar{D}$)

Expression of Problems using Boolean Algebra

Row	A	B	C	D	
0	0	0	0	0	$\bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$
1	0	0	0	1	$\bar{A}\,\bar{B}\,\bar{C}\,D$
2	0	0	1	0	$\bar{A}\,\bar{B}\,C\,\bar{D}$
3	0	0	1	1	$\bar{A}\,\bar{B}\,C\,D$
4	0	1	0	0	$\bar{A}\,B\,\bar{C}\,\bar{D}$
5	0	1	0	1	$\bar{A}\,B\,\bar{C}\,D$
6	0	1	1	0	$\bar{A}\,B\,C\,\bar{D}$
7	0	1	1	1	$\bar{A}\,B\,C\,D$
8	1	0	0	0	$A\,\bar{B}\,\bar{C}\,\bar{D}$
9	1	0	0	1	$A\,\bar{B}\,\bar{C}\,D$
10	1	0	1	0	$A\,\bar{B}\,C\,\bar{D}$
11	1	0	1	1	$A\,\bar{B}\,C\,D$
12	1	1	0	0	$A\,B\,\bar{C}\,\bar{D}$
13	1	1	0	1	$A\,B\,\bar{C}\,D$
14	1	1	1	0	$A\,B\,C\,\bar{D}$
15	1	1	1	1	$A\,B\,C\,D$

Expression of Problems using Boolean Algebra

- We call each of these product terms a MINTERM
- Note that each minterm contains each input variable in turn
- We can express any Boolean expression in minterm form
- If f is expressed this way, we say it is in
  - ‘sum of products’ form
  - 1st Canonical Form

Expression of Problems using Boolean Algebra

Row	A	B	C	D		Minterm
0	0	0	0	0	$\bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$	m0
1	0	0	0	1	$\bar{A}\,\bar{B}\,\bar{C}\,D$	m1
2	0	0	1	0	$\bar{A}\,\bar{B}\,C\,\bar{D}$	m2
3	0	0	1	1	$\bar{A}\,\bar{B}\,C\,D$	m3
4	0	1	0	0	$\bar{A}\,B\,\bar{C}\,\bar{D}$	m4
5	0	1	0	1	$\bar{A}\,B\,\bar{C}\,D$	m5
6	0	1	1	0	$\bar{A}\,B\,C\,\bar{D}$	m6
7	0	1	1	1	$\bar{A}\,B\,C\,D$	m7
8	1	0	0	0	$A\,\bar{B}\,\bar{C}\,\bar{D}$	m8
9	1	0	0	1	$A\,\bar{B}\,\bar{C}\,D$	m9
10	1	0	1	0	$A\,\bar{B}\,C\,\bar{D}$	m10
11	1	0	1	1	$A\,\bar{B}\,C\,D$	m11
12	1	1	0	0	$A\,B\,\bar{C}\,\bar{D}$	m12
13	1	1	0	1	$A\,B\,\bar{C}\,D$	m13
14	1	1	1	0	$A\,B\,C\,\bar{D}$	m14
15	1	1	1	1	$A\,B\,C\,D$	m15

Expression of Problems using Boolean Algebra

- For example, in the case of the alarm circuit, we have:
- A = ‘the alarm switch is on’
- B = ‘the door switch is open’
- C = ‘it is after 6 pm’
- D = ‘the window switch is open’
- f = 1 = if A(=1) . B(=1) + C(=1) . D(=1)
- If f = 1, then the alarm will ring!

Expression of Problems using Boolean Algebra

- Then
  f = m3 + m7 + m11 + m12 + m13 + m14 + m15
- Why?
- Because m3, m7, m11, and m15 are the minterm expressions when both C and D are 1
- And m12, m13, and m14 are the minterm expressions when both A and B are 1
- And the alarm should ring if any of these expressions occur, i.e., if f = AB + CD

Expression of Problems using Boolean Algebra

Row	A	B	C	D		Minterm
0	0	0	0	0	$\bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$	m0
1	0	0	0	1	$\bar{A}\,\bar{B}\,\bar{C}\,D$	m1
2	0	0	1	0	$\bar{A}\,\bar{B}\,C\,\bar{D}$	m2
3	0	0	1	1	$\bar{A}\,\bar{B}\,C\,D$	m3
4	0	1	0	0	$\bar{A}\,B\,\bar{C}\,\bar{D}$	m4
5	0	1	0	1	$\bar{A}\,B\,\bar{C}\,D$	m5
6	0	1	1	0	$\bar{A}\,B\,C\,\bar{D}$	m6
7	0	1	1	1	$\bar{A}\,B\,C\,D$	m7
8	1	0	0	0	$A\,\bar{B}\,\bar{C}\,\bar{D}$	m8
9	1	0	0	1	$A\,\bar{B}\,\bar{C}\,D$	m9
10	1	0	1	0	$A\,\bar{B}\,C\,\bar{D}$	m10
11	1	0	1	1	$A\,\bar{B}\,C\,D$	m11
12	1	1	0	0	$A\,B\,\bar{C}\,\bar{D}$	m12
13	1	1	0	1	$A\,B\,\bar{C}\,D$	m13
14	1	1	1	0	$A\,B\,C\,\bar{D}$	m14
15	1	1	1	1	$A\,B\,C\,D$	m15

The Director's Dilemma and the Digital Doctor

Example courtesy of

Dr. D. J. Furlong
Department of Electronic and Electrical Engineering
Trinity College, Dublin
Ireland

The Director's Dilemma and the Digital Doctor

- The Cast:
  - Director of Public Health Systems
  - Dr. Logik
- The Scenario:
  - Overcrowded Public Health Clinic with many obviously ailing clients looking for a diagnosis ... Is it the dreaded Boolean virus? Or just Digital Flu? Or maybe just epidemic paranoia ....

The Director's Dilemma and the Digital Doctor

Director:

Well, we're going to have to do something ...

I mean there's thousands of them out there either sick or just plain worried that they might be going to come down with some of the symptoms ...

You're the expert ...

The Director's Dilemma and the Digital Doctor

Dr. Logik:

Ja, Ja, Ja, ... I know, but I'm needed at all the other clinics too, you know.

Look, why don't you get those clever engineering or computer science types of yours to build you an automated diagnostic system

The Director's Dilemma and the Digital Doctor

Dr. Logik:

so that all these good people can just enter their particular symptoms, if they have any,

and be told electronically which medication, if any, to take ...

Ja? Very simple, really.

The Director's Dilemma and the Digital Doctor

Director:

Well, I suppose it would be something .. but what if this computer scientist gets it wrong?

I mean, if these people actually have this Boolean Virus then, - OK, we just prescribe Neominterm and they'll get over it.

But if we give them Neominterm and they just have Digital Flu, or nothing at all, then they'll die.

It's lethal stuff, you know

The Director's Dilemma and the Digital Doctor

Director:

And if they have in fact got the Boolean Virus and we don't give them Neominterm then they'll croak it too ...

And the symptoms are very similar ...

I mean the chances of getting it wrong are so great and the consequences so drastic I really feel we have to have an expert like yourself here on site to check ...

The Director's Dilemma and the Digital Doctor

Dr. Logik:

Look, it's quite impossible that I should stay here and examine all these people.

Calm down ... the problem is not that difficult to deal with.

You're just panicking. Relax ... I'll start things off for your computer scientist, but then I really must dash ...

The Director's Dilemma and the Digital Doctor

Dr. Logik:

You see, it's like this.

There are four symptoms: chills, rash, bloodshot eyes, and a fever.

Now, anybody who hasn't got any of these symptoms doesn't have either Boolean Virus or Digital Flu.

OK? Ja.

The Director's Dilemma and the Digital Doctor

Dr. Logik:

Anybody who has NO chill but HAS some of the other symptoms is just suffering from Digital Flu,

so give them some DeMorgan salts and send them home.

Ja? Good.

The Director's Dilemma and the Digital Doctor

Dr. Logik:

Now, if they DO have a chill,

then you've got to look at the combination of symptoms ...

Ja? OK!

If there is a chill and a rash only, then it's Digital Flu again.

The Director's Dilemma and the Digital Doctor

Dr. Logik:

However, if it's a chill by itself

or a chill any any other combination of rash, bloodshot eyes, and fever,

then they've got the Boolean Virus for sure.

Only Neominterm can save them then ...

The Director's Dilemma and the Digital Doctor

Director:

That may be all very straightforward to you but how is my computer scientist going to design a foolproof system to distinguish Digital Flu from Boolean Virus?

The Director's Dilemma and the Digital Doctor

Dr. Logik:

Look ... I'm late as it is, but all that is required is a truth table with inputs and outputs like this

The Director's Dilemma and the Digital Doctor

Chills	Rash	Bloodshot Eyes	Fever	Boolean Virus	Digital Flu

The Director's Dilemma and the Digital Doctor

Dr. Logik:

The outputs will be the Boolean Virus and Digital Flu indicators ... say a few little LEDs, Ja?

Just hook them up to the system, along with the input symptom switches - one each for Chills, Rash, Bloodshot Eyes, and Fever, Ja?

Then all the patients have to do is move the switch to True if they have a particular symptom, for False if not ...

The Director's Dilemma and the Digital Doctor

Dr. Logik:

Then the system will do the rest and either the Boolean Virus LED or the Digital Flu LED will go on - unless of course they don't have any of these symptoms in which case they're just scared and don't have either the Boolean Virus or the Digital Flu

The Director's Dilemma and the Digital Doctor

Dr. Logik:

So,

If Boolean Virus THEN please take some Neominterm,

ELSE IF Digital Flu THEN please take some DeMorgan Salts,

ElSE just go home.

Ja? That's the way you computer people like to put things, isn't it? Ja? Simple!

The Director's Dilemma and the Digital Doctor

Director:

Ja ... I mean Yes ...

Dr. Logik:

Right then. I'm off. Good luck ... Oh ja, you might need this ...

Director:

A mirror? I don't follow you, Doctor.

The Director's Dilemma and the Digital Doctor

Dr. Logik:

The bloodshot eyes ... They'll have to be able to examine their eyes, now won't they? Adieu ...

Exeunt Dr. Logik ... Director rings Department of Computer Science

Director:

Can you send over a Computer Scientist right away, please?

The Director's Dilemma and the Digital Doctor

Enter Computer Scientist with puzzled look.

Director:

Ah, yes ... now look ... we need a system and it's got to work as follows ...

Director explains problem and gives Computer Scientist Dr. Logik's truth table sketch and mirror ...

The Director's Dilemma and the Digital Doctor

Computer Scientist:

Yes, but as far as I remember the only logic gates we have in stock are NAND gates ...

Director:

Well, if I remember my college course in digital logic, that shouldn't necessarily be a problem, now should it?

The Director's Dilemma and the Digital Doctor

Computer Scientist:

No? I mean No! er ... Yes, right ... Well, it depends ... Let's see what's in stores. I don't think there's very many of them at that ...

Computer Scientist checks his laptop database ...

The Director's Dilemma and the Digital Doctor

Computer Scientist:

6 two-input, 2 three-input, and 1 four-input NAND gates .. That's all! Power supply ... yep, Switches .. yep. LEDs ... yep. Box ... yep. Well, I'll just have to see if I can make it work with that lot.

Director:

Please do! We're depending on you ...

The Director's Dilemma and the Digital Doctor

Will the Computer Scientist be successful?

Will the Director have to send for the National Guard to control the by now tense and growing crowd looking for diagnosis?

Stay tuned!

The Director's Dilemma
Key Facts

- There are four symptoms:
  - chills
  - rash
  - bloodshot eyes
  - fever
- There are two ailments:
  - Boolean Virus
  - Digital Flu

The Director's Dilemma
Key Facts

- Anybody who hasn't got any of these symptoms doesn't have either Boolean Virus or Digital Flu.
- Anybody who has NO chill but HAS some of the other symptoms is just suffering from Digital Flu,
- If there is a chill and a rash only, then it's Digital Flu again.
- it's a chill by itself or a chill any any other combination of rash, bloodshot eyes, and fever,

  then they've got the Boolean Virus

The Director's Dilemma and the Digital Doctor

- Available Equipment
  - 6 two-input NAND gates
  - 2 three-input NAND gates
  - 1 four-input NAND gate

The Director's Dilemma and the Digital Doctor

- We will address the problem in three way, just to compare efficiency and effectiveness:
- Straightforward (naive) implementation of the condition in gates
- Efficient implementation of the conditions in gates by simplifying the expressions
- Function minimization procedure using minterms and Karnaugh maps

The Director's Dilemma
Key Facts

- There are four symptoms:
  - (A) chills
  - (B) rash
  - (C) bloodshot eyes
  - (D) fever
- There are two ailments:
  - (f1) Boolean Virus
  - (f2) Digital Flu

The Director's Dilemma
Key Facts

- Anybody who hasn't got any of these symptoms doesn't have either Boolean Virus or Digital Flu.
- NOT (A OR B OR C OR D)
- ~~A + B + C + D~~
- Anybody who has NO chill but HAS some of the other symptoms is just suffering from Digital Flu
- f2 = $\overline{A}$. (B + C + D)

The Director's Dilemma
Key Facts

- If there is a chill and a rash only, then it's Digital Flu again.
- f2 = A . B . $\underline{C}$. $\underline{D}$
- it's a chill by itself or a chill and any other combination of rash, bloodshot eyes, and fever,
  (But not the Digital Flu combination of chill and rash only - NB)

  then they've got the Boolean Virus
- f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ +A. (B + C + D) . $\underline{(B . C . D)}$

The Director's Dilemma
Key Facts

- f1 = f1 = A . $\overline{B}$. $\overline{C}$. $\overline{D}$ +A. (B + C + D) . $\overline{(B . C . D)}$
  f2 = $\overline{A}$. (B + C + D)
  f2 = A . B . $\overline{C}$. $\overline{D}$

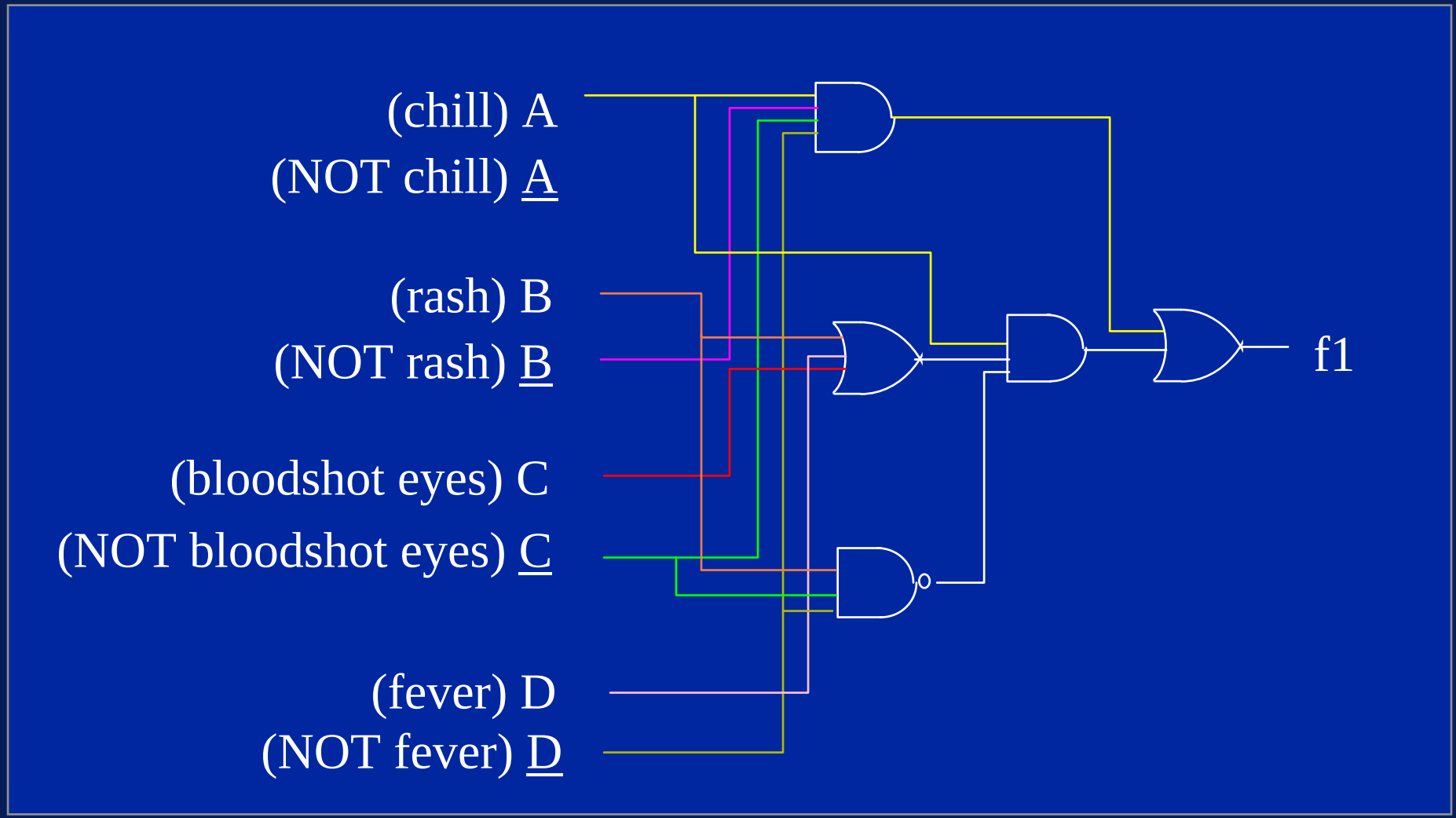
- f1 = A . $\overline{B}$. $\overline{C}$. $\overline{D}$ +A. (B + C + D) . (B . ~~$\overline{C}$. $\overline{D}$~~)
  f2 = $\overline{A}$. (B + C + D) + A . B . $\overline{C}$. $\overline{D}$

The Director's Dilemma
Key Facts

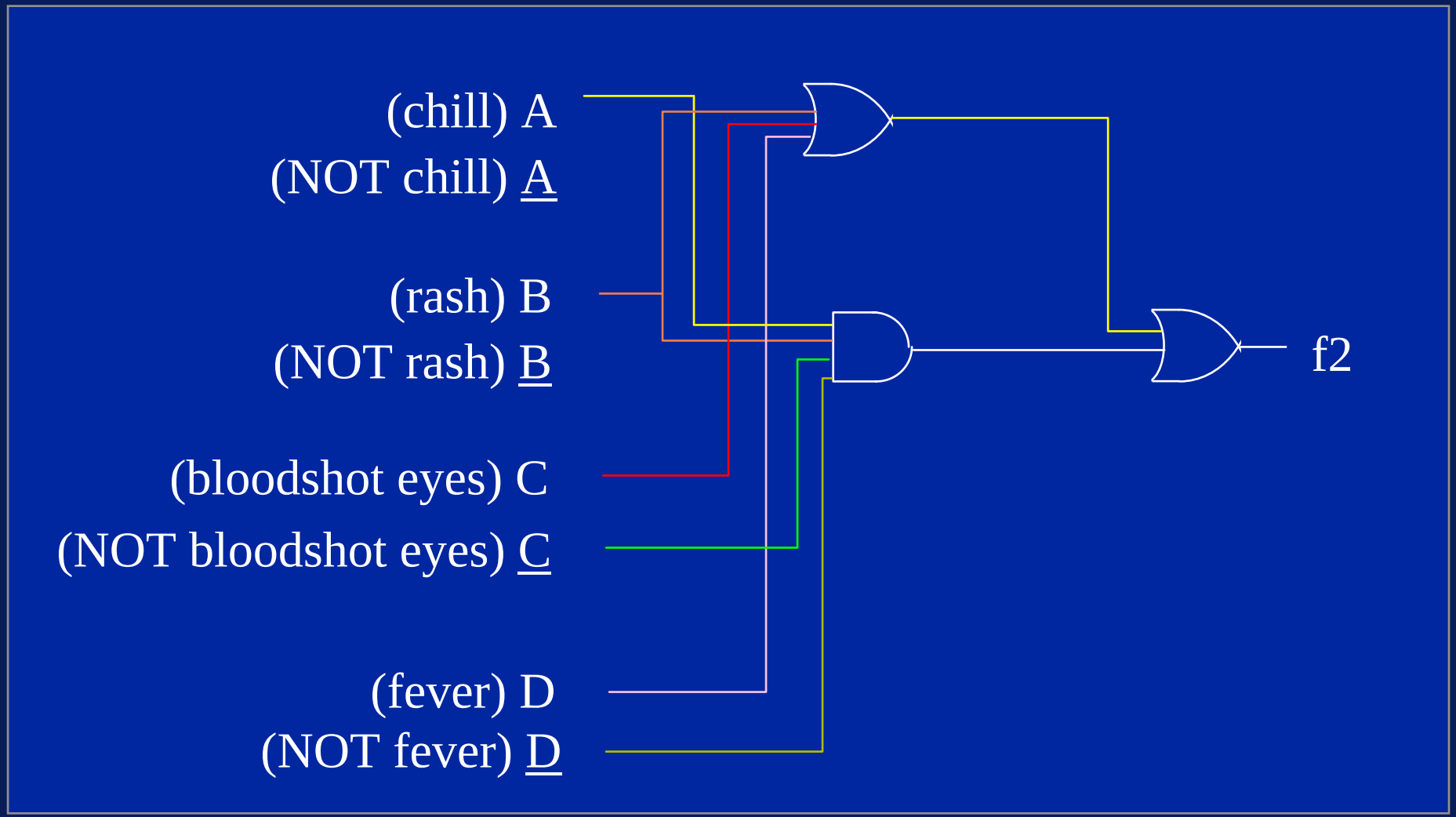
- Our first implementation of this will be a direct 'gating'of these two Boolean expressions
- Note that in all of the following implementations, we will assume that both the value of an input variable (e.g. A or 'chill') and its logical inverse (e.g. $\overline{A}$ or 'NOT chill') are available

f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ +A. (B + C + D) . (B . $\underline{C}$. $\underline{D}$)

Corresponding Digital Circuit



f2 = A . (B + C + D) + A . B . C . D Corresponding Digital Circuit



The Director's Dilemma Simplified Implementation

- These two logic circuits are complicated!
- Can we do any better?
- Let's try to simplify the expressions.

The Director's Dilemma
Simplified Implementation

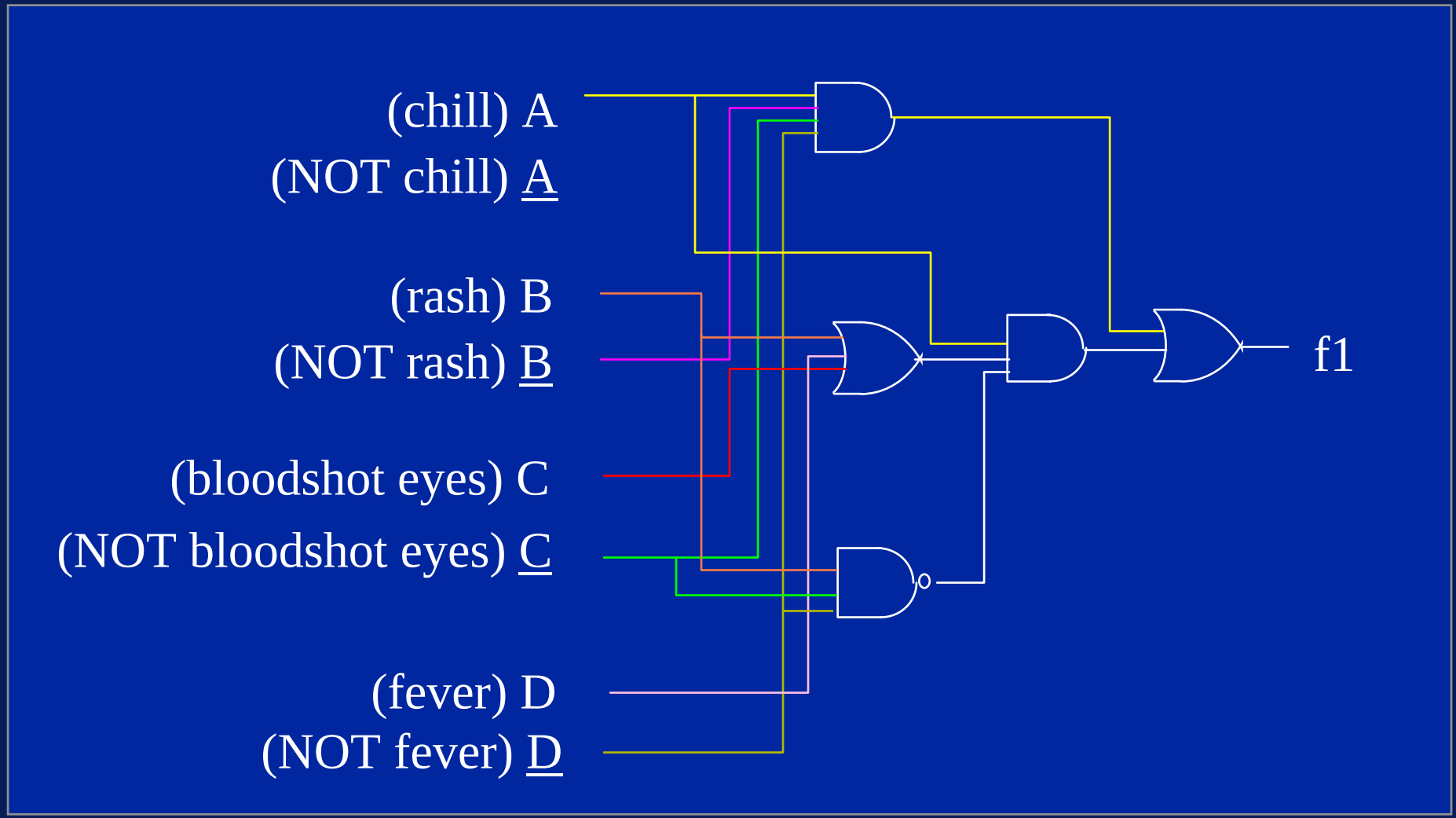
- $f1 = A \,.\, \underline{B} \,.\, \underline{C} \,.\, \underline{D} + A.\,(\,B + C + D) \,.\, (B \,.\, \underline{C} \,.\, \underline{D})$
- $f1 = A \,.\, \underline{B} \,.\, \underline{C} \,.\, \underline{D} + (A.B + A.C + A.D).(\underline{B} + C + D)$
- $f1 = A \,.\, \underline{B} \,.\, \underline{C} \,.\, \underline{D} + A.B.\underline{B} + A.C.\underline{B} + A.D.\underline{B} + A.B.C + A.C.C + A.D.C + A.B.D + A.C.D + A.D.D$
- $f1 = A \,.\, \underline{B} \,.\, \underline{C} \,.\, \underline{D} + A.0 + A.C.\underline{B} + A.D.\underline{B} + A.B.C + A.C + A.D.C + A.B.D + A.C.D + A.D$

The Director's Dilemma
Simplified Implementation

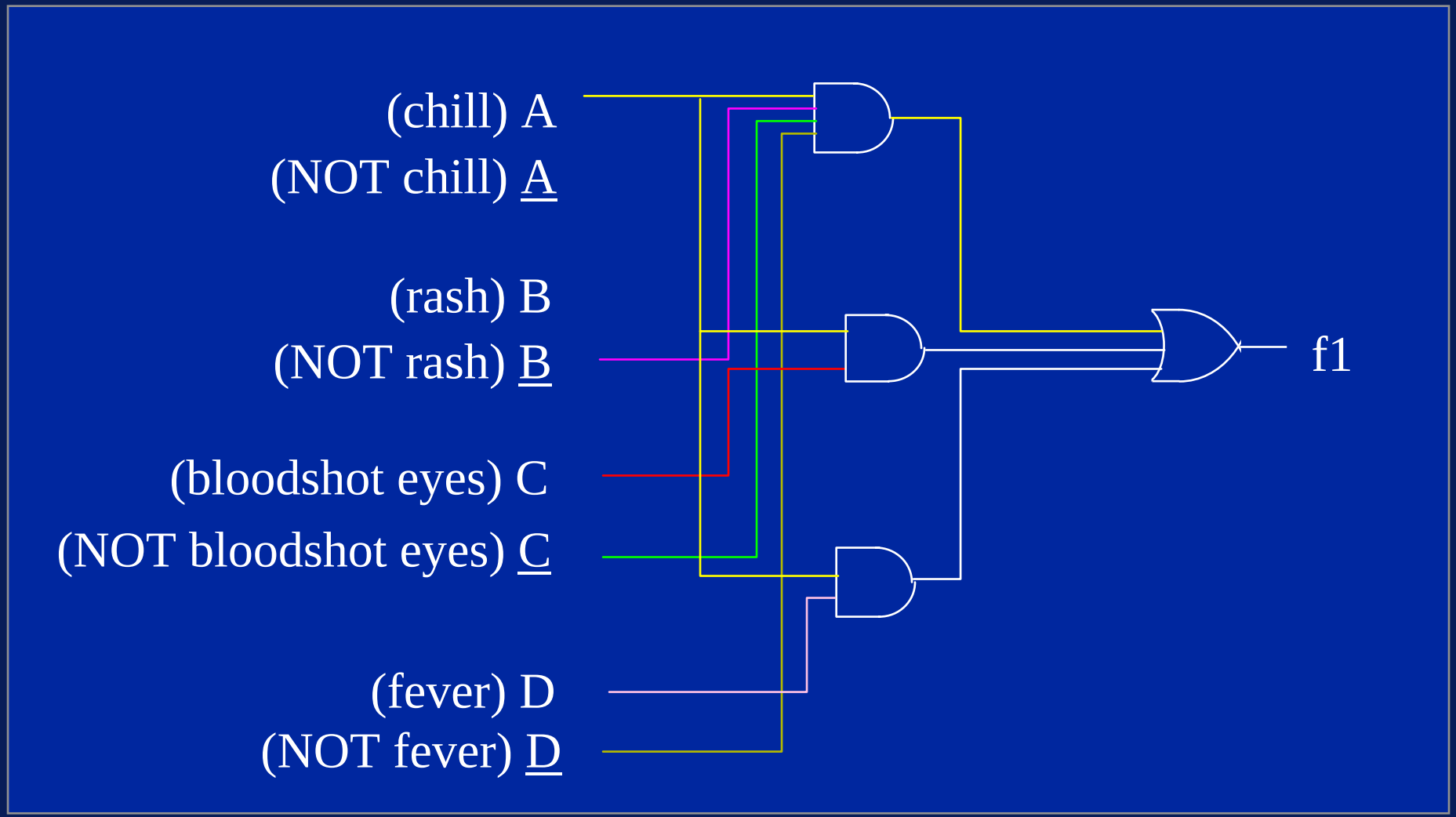
- f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ + 0 + A.(C.$\underline{B}$ + D.$\underline{B}$ + B.C + C + D.C + B.D + C.D + D)
- f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ + A.(C.$\underline{B}$ + D.$\underline{B}$ + B.C + C + D.C + B.D + D)
- f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ + A.(C.($\underline{B}$ + B) + C + D.($\underline{B}$ + C + B + 1))
- f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ + A.(C.1 + C + D.1)
- f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ + A.(C + D)
- f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ + A.C + A.D

f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ +A. (B + C + D) . (B . $\underline{C}$. $\underline{D}$)

Corresponding Digital Circuit



f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ + A.C + A.D Corresponding Digital Circuit



The Director's Dilemma Simplified Implementation

- That simplification was hard work! Is there an easier way?
- Yes!
- We use truth-tables, minterms, and a simplification method known as Karnaugh maps

Anybody who hasn't got any of these symptoms doesn't have either Boolean Virus or Digital Flu

Chills	Rash	Bloodshot Eyes	Fever	Boolean Virus	Digital Flu
0	0	0	0	0	0

Anybody who has NO chill but HAS some of the other symptoms is just suffering from Digital Flu

Chills	Rash	Bloodshot Eyes	Fever	Boolean Virus	Digital Flu
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1

The Director's Dilemma
Key Facts

- If there is a chill and a rash only, then it's Digital Flu again.
- It's a chill by itself or a chill and any other combination of rash, bloodshot eyes, and fever,
  (But not the Digital Flu combination of chill and rash only - NB)

  then they've got the Boolean Virus

If there is a chill and a rash only, then it's Digital Flu again ..... chill and other combinations - BV

Chills	Rash	Bloodshot Eyes	Fever	Boolean Virus	Digital Flu
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	0

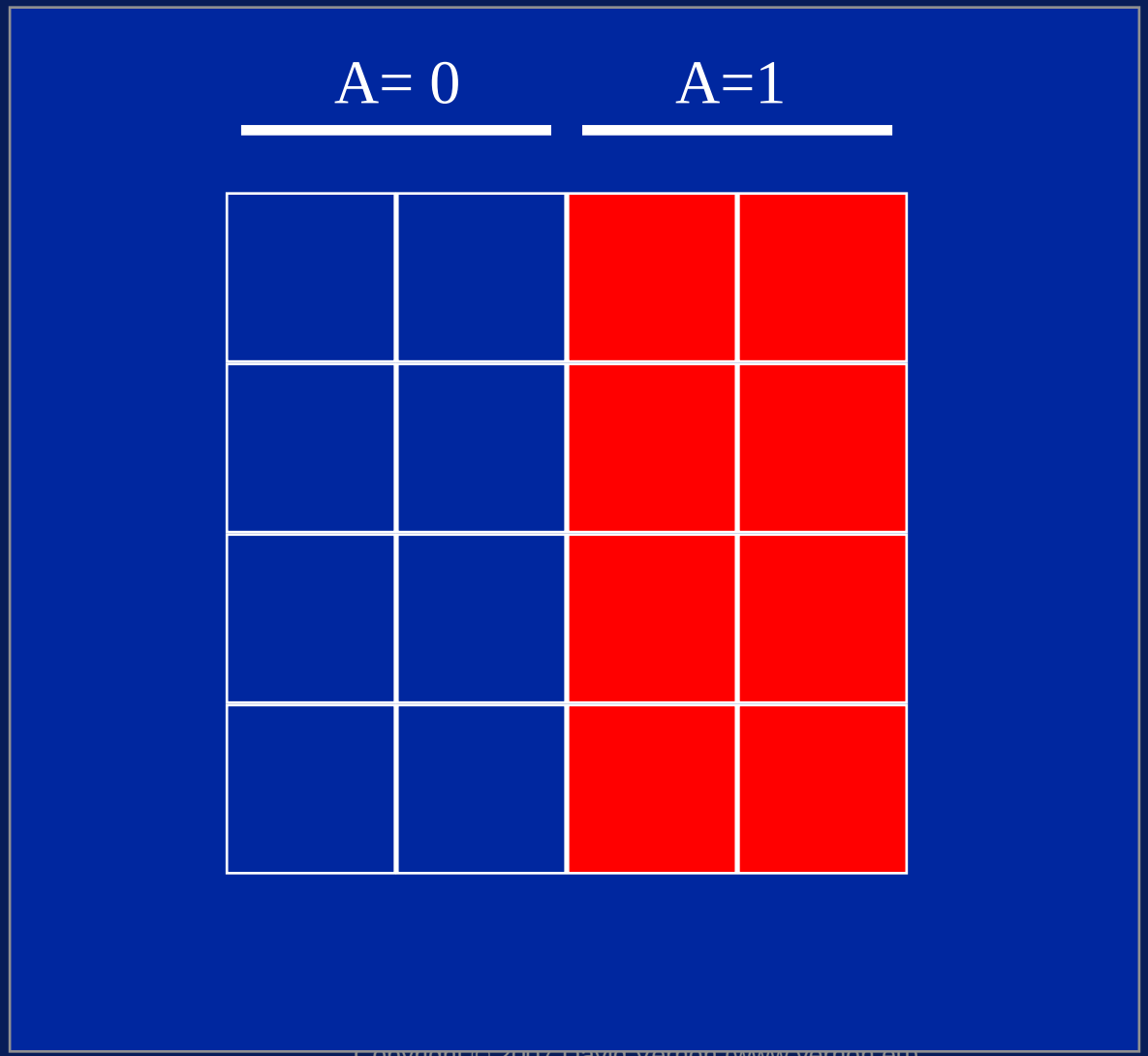
Simplification using Minterms and Karnaugh Maps

Row	A	B	C	D	f1	f2	Minterm
0	0	0	0	0	0	1	m0
1	0	0	0	1	0	1	m1
2	0	0	1	0	0	1	m2
3	0	0	1	1	0	1	m3
4	0	1	0	0	0	1	m4
5	0	1	0	1	0	1	m5
6	0	1	1	0	0	1	m6
7	0	1	1	1	0	1	m7
8	1	0	0	0	1	0	m8
9	1	0	0	1	1	0	m9
10	1	0	1	0	1	0	m10
11	1	0	1	1	1	0	m11
12	1	1	0	0	0	1	m12
13	1	1	0	1	1	0	m13
14	1	1	1	0	1	0	m14
15	1	1	1	1	1	0	m15

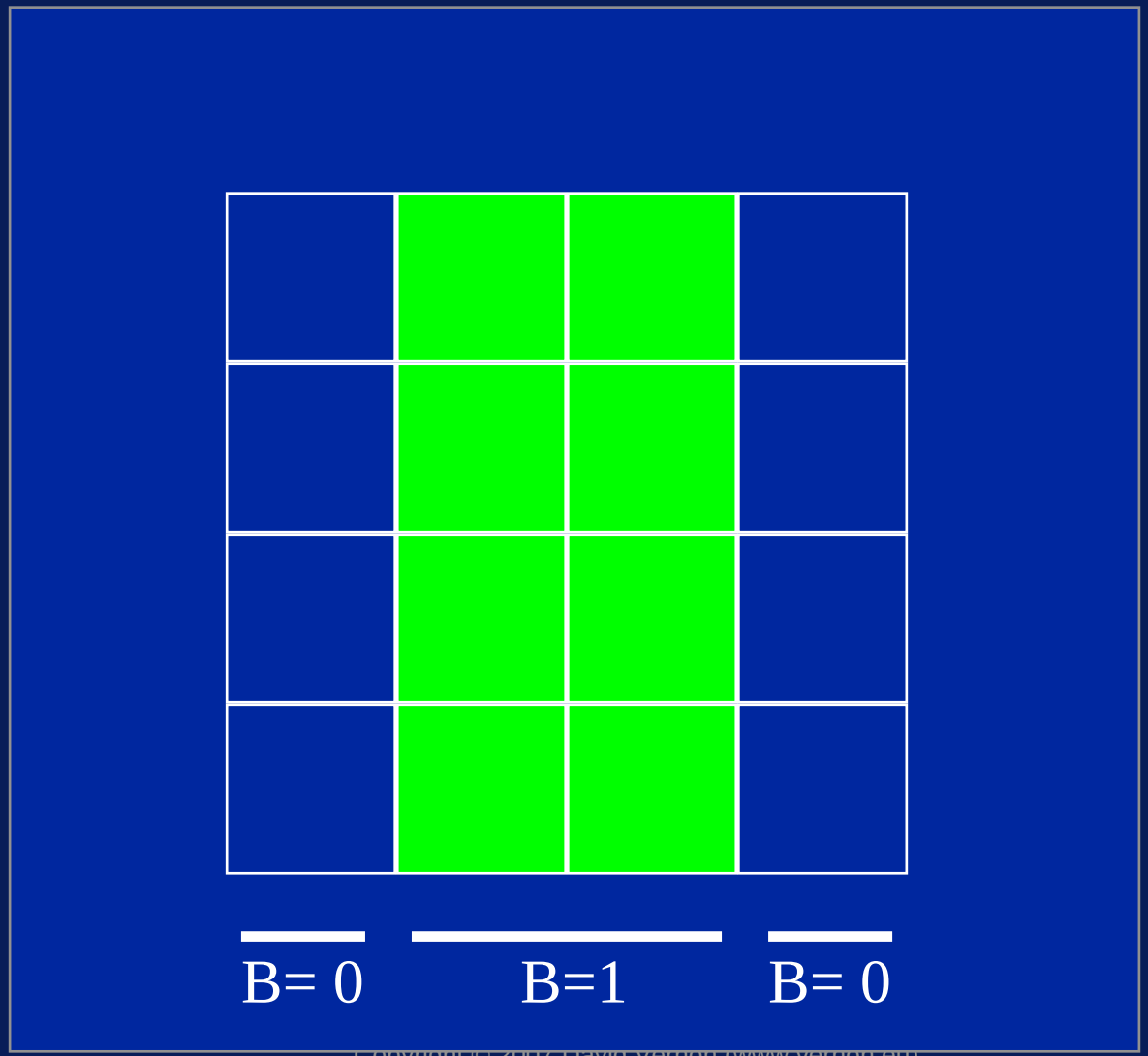
Simplification using Minterms and Karnaugh Maps

- A Karnaugh Map is simply another form of truth table
- Entry of each minterm
- Arranged in a 2-D array
- Each variable ‘blocks in’ half of the array
- Different half for each variable
- With a 4-variable expression, we know there are 16 possible combinations or minterms

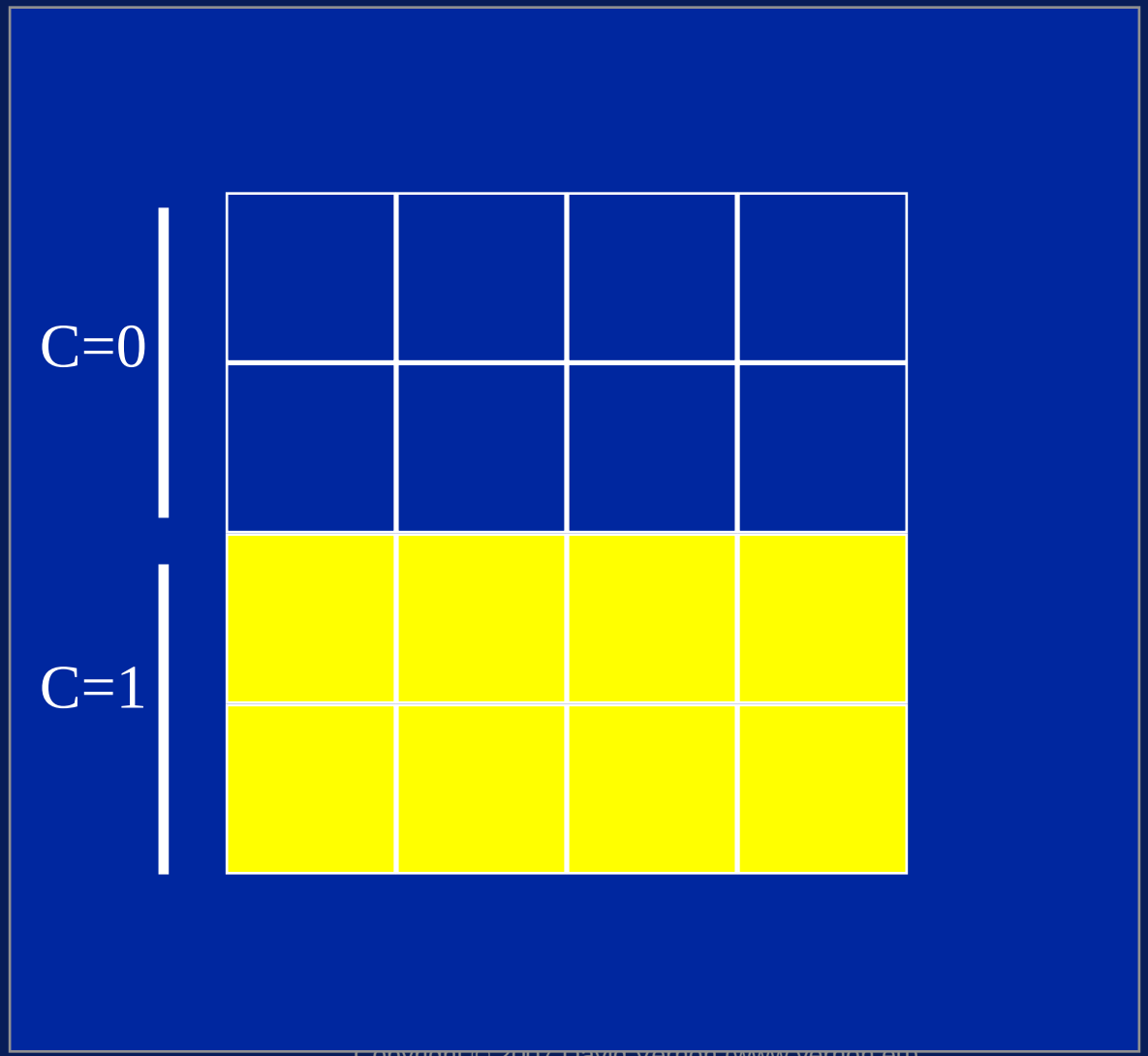
Simplification using Minterms and Karnaugh Maps



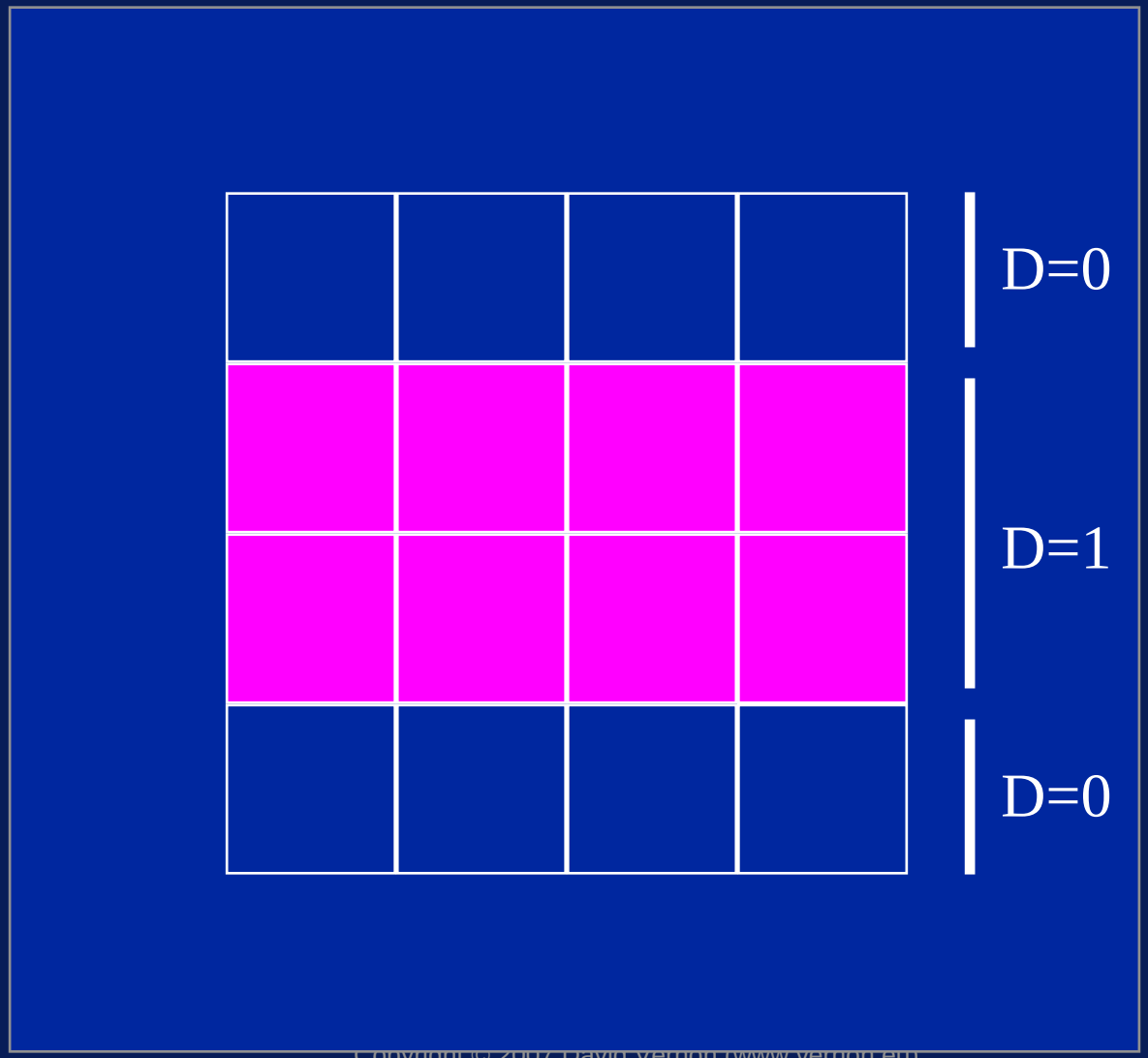
Simplification using Minterms and Karnaugh Maps



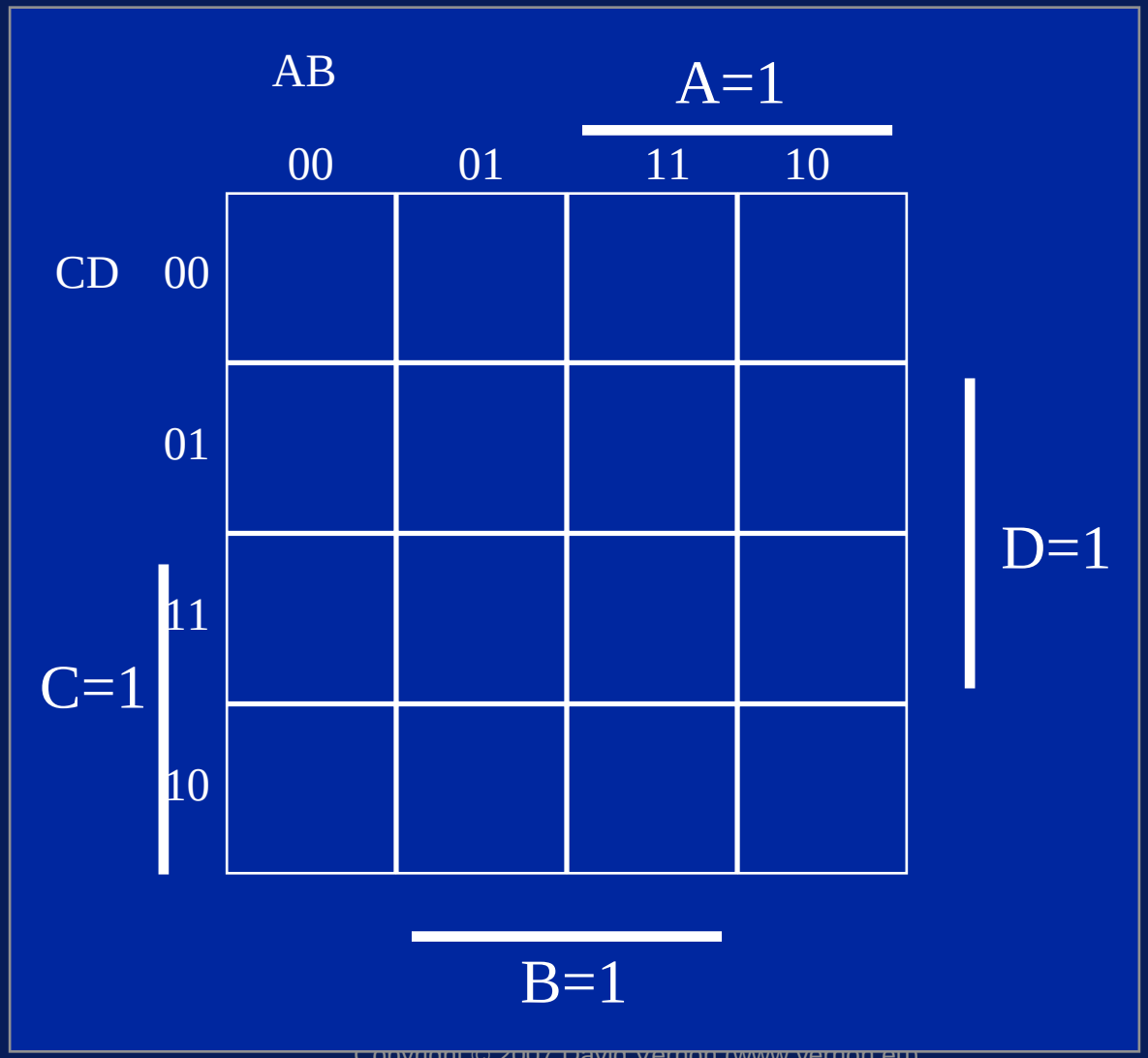
Simplification using Minterms and Karnaugh Maps



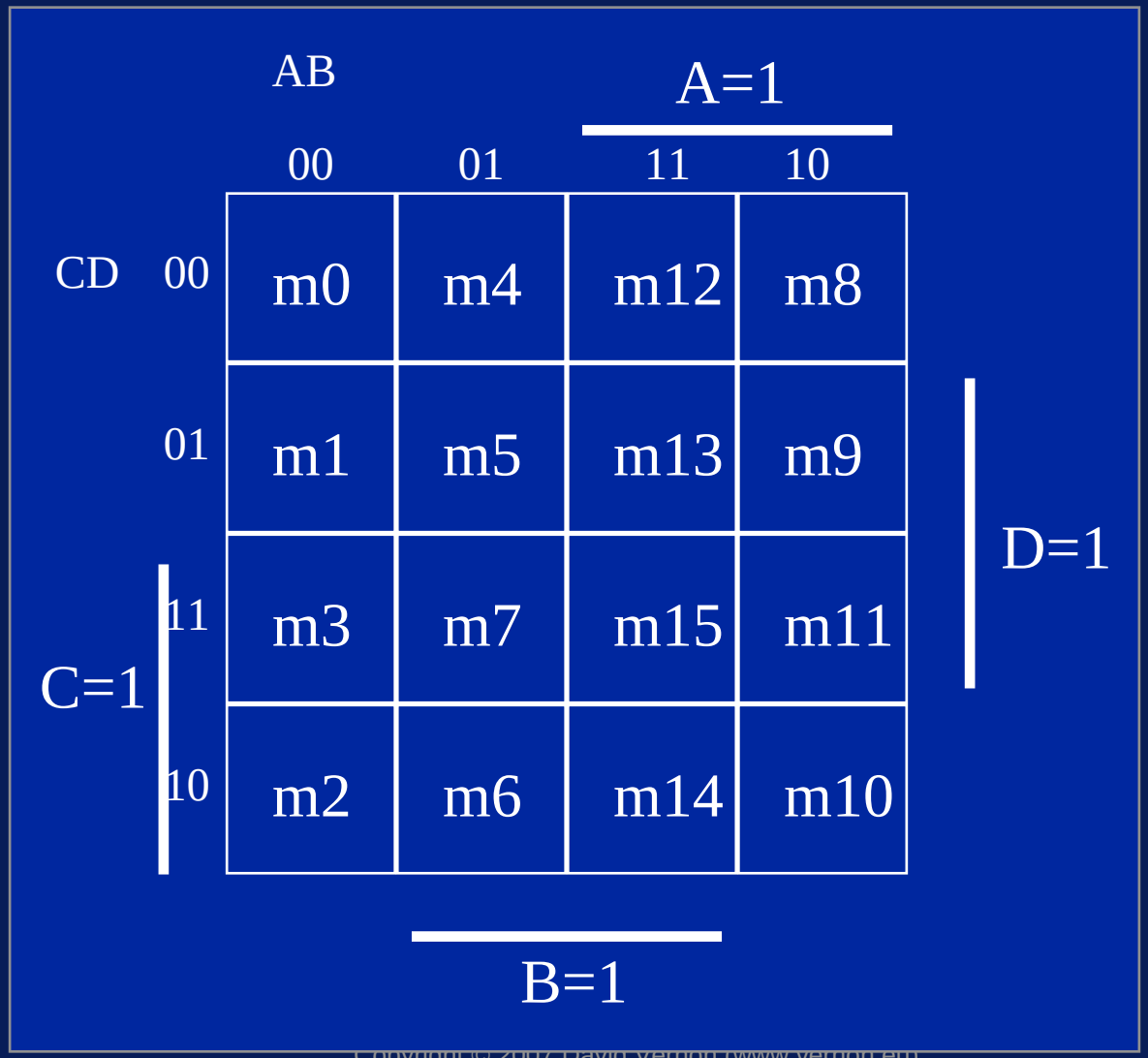
Simplification using Minterms and Karnaugh Maps



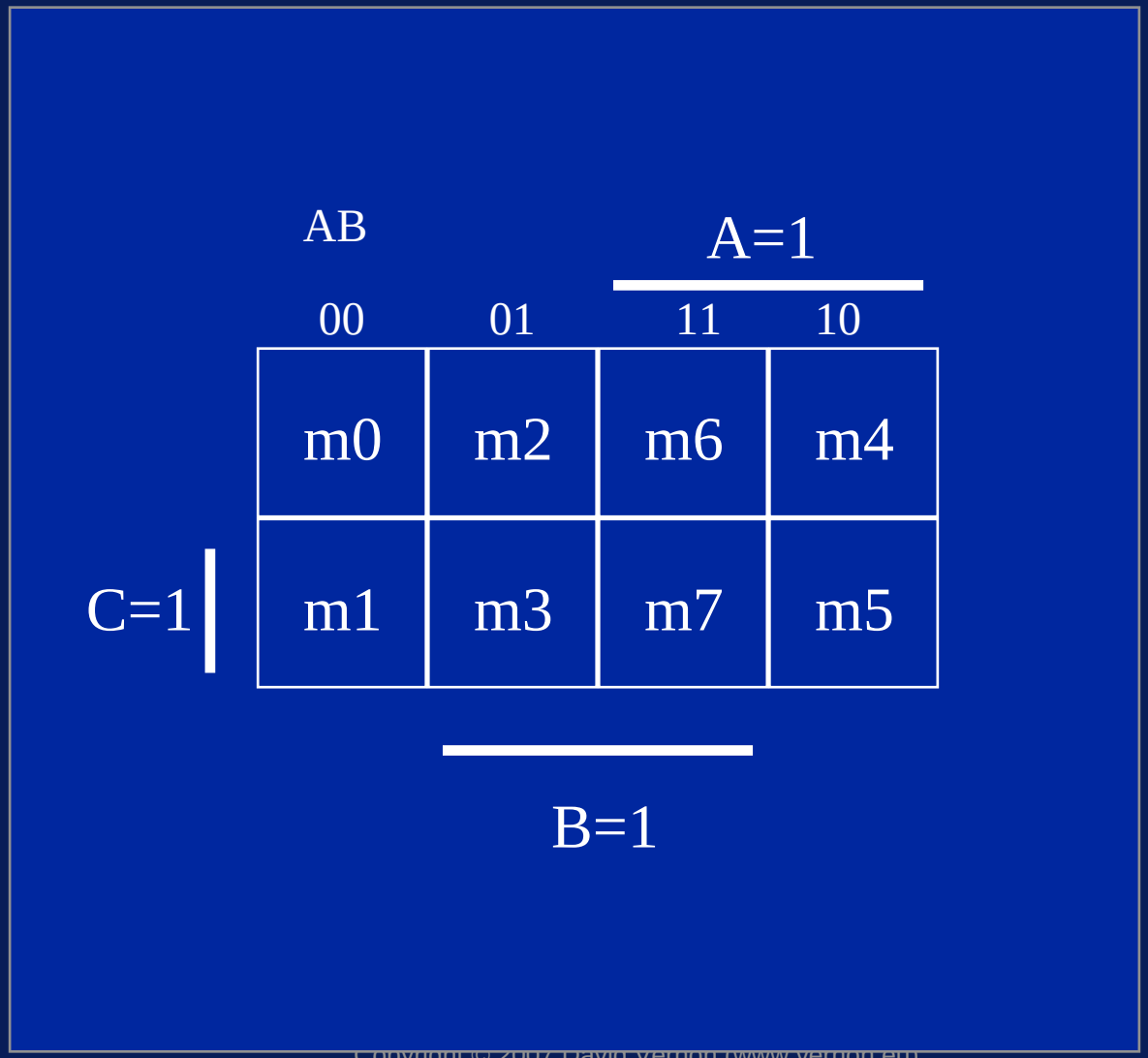
Simplification using Minterms and Karnaugh Maps



Simplification using Minterms and Karnaugh Maps



Aside: 3-Variable Karnaugh Map



Simplification using Minterms and Karnaugh Maps

- To simplify a Boolean expression
- Express it as a (conventional) truth table
- Identify the minterms that are ‘TRUE’
- Identify the minterms that are ‘FALSE’
- Mark them as such in the Karnaugh Map
- And then .....

Simplification using Minterms and Karnaugh Maps

- Identify groups of adjacent ‘1’s in the Karnaugh Map
  - Note that two squares are adjacent if they share a boundary (this includes the top and bottom edges and the left and right edges: top IS adjacent to bottom and left IS adjecent to right).
  - For example, minterm 11 is adjacent to minterm 3
- Try to get groups that are as large as possible(in blocks of 1, 2, 4, 8, ...)

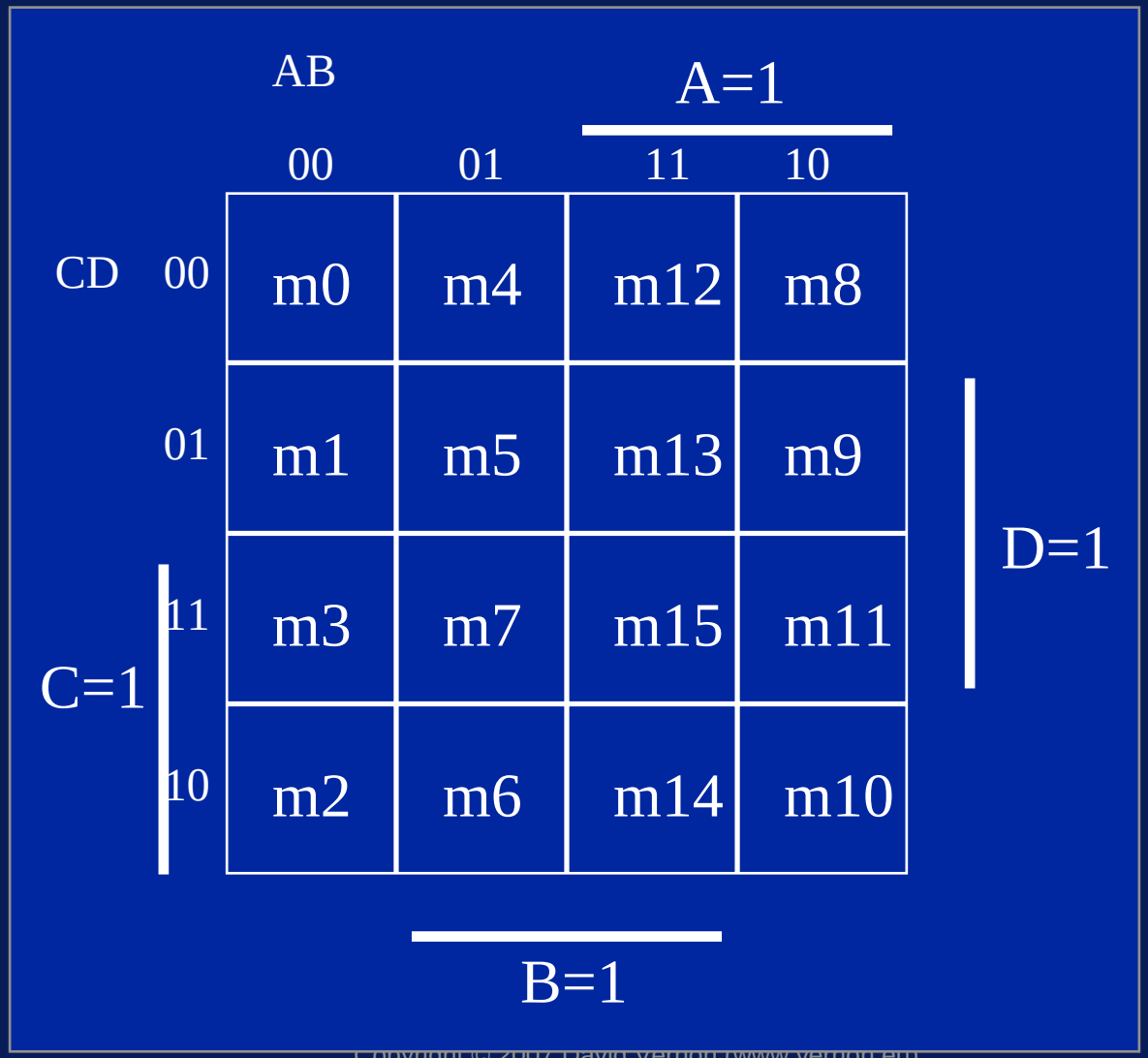
Simplification using Minterms and Karnaugh Maps

- Identify the least number of variables that are required to unambiguously label that group
- The simplified expression is then the logical OR of all the terms that are needed to identify each (largest as possible) group of ‘1’s
- Note: groups may overlap and this sometimes helps when identifying large groups

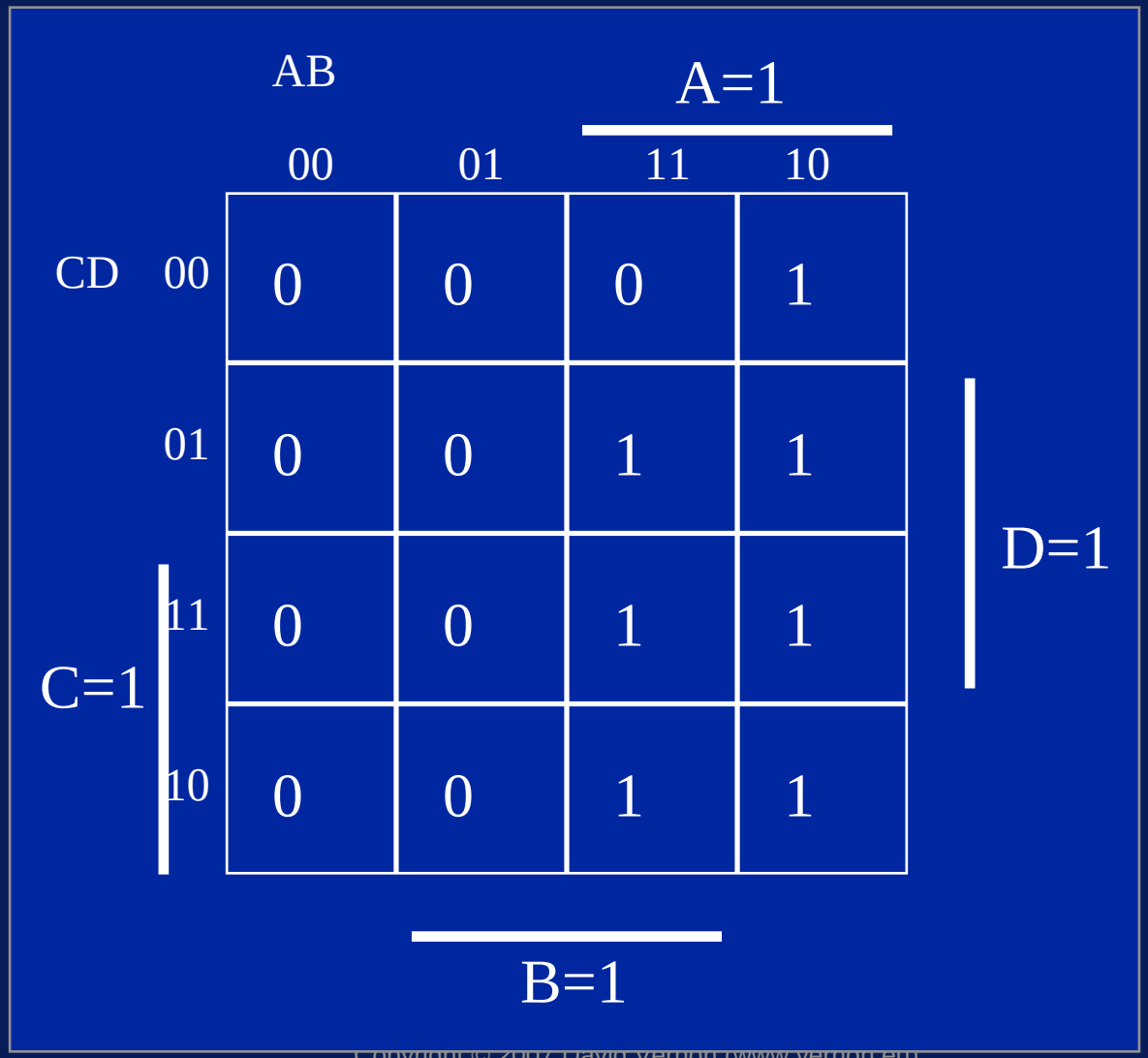
Simplification using Minterms and Karnaugh Maps

Row	A	B	C	D	f1	f2	Minterm
0	0	0	0	0	0	1	m0
1	0	0	0	1	0	1	m1
2	0	0	1	0	0	1	m2
3	0	0	1	1	0	1	m3
4	0	1	0	0	0	1	m4
5	0	1	0	1	0	1	m5
6	0	1	1	0	0	1	m6
7	0	1	1	1	0	1	m7
8	1	0	0	0	1	0	m8
9	1	0	0	1	1	0	m9
10	1	0	1	0	1	0	m10
11	1	0	1	1	1	0	m11
12	1	1	0	0	0	1	m12
13	1	1	0	1	1	0	m13
14	1	1	1	0	1	0	m14
15	1	1	1	1	1	0	m15

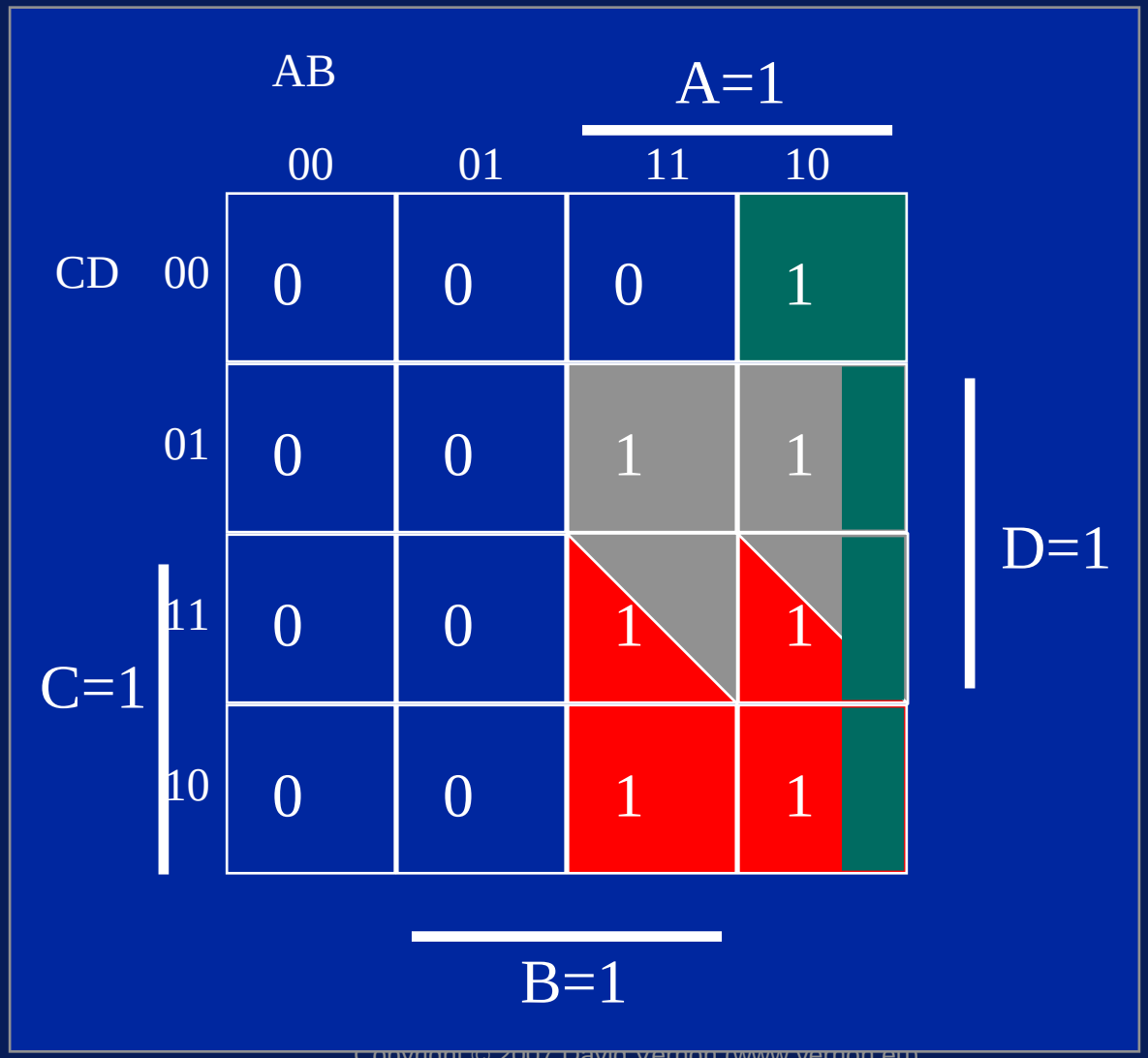
Simplification using Minterms and Karnaugh Maps



f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ +A. (B + C + D) . (B . $\underline{C}$. $\underline{D}$) Karnaugh Maps



f1 = A . B . C . D +A. (B + C + D) . (B . C . D) Karnaugh Maps

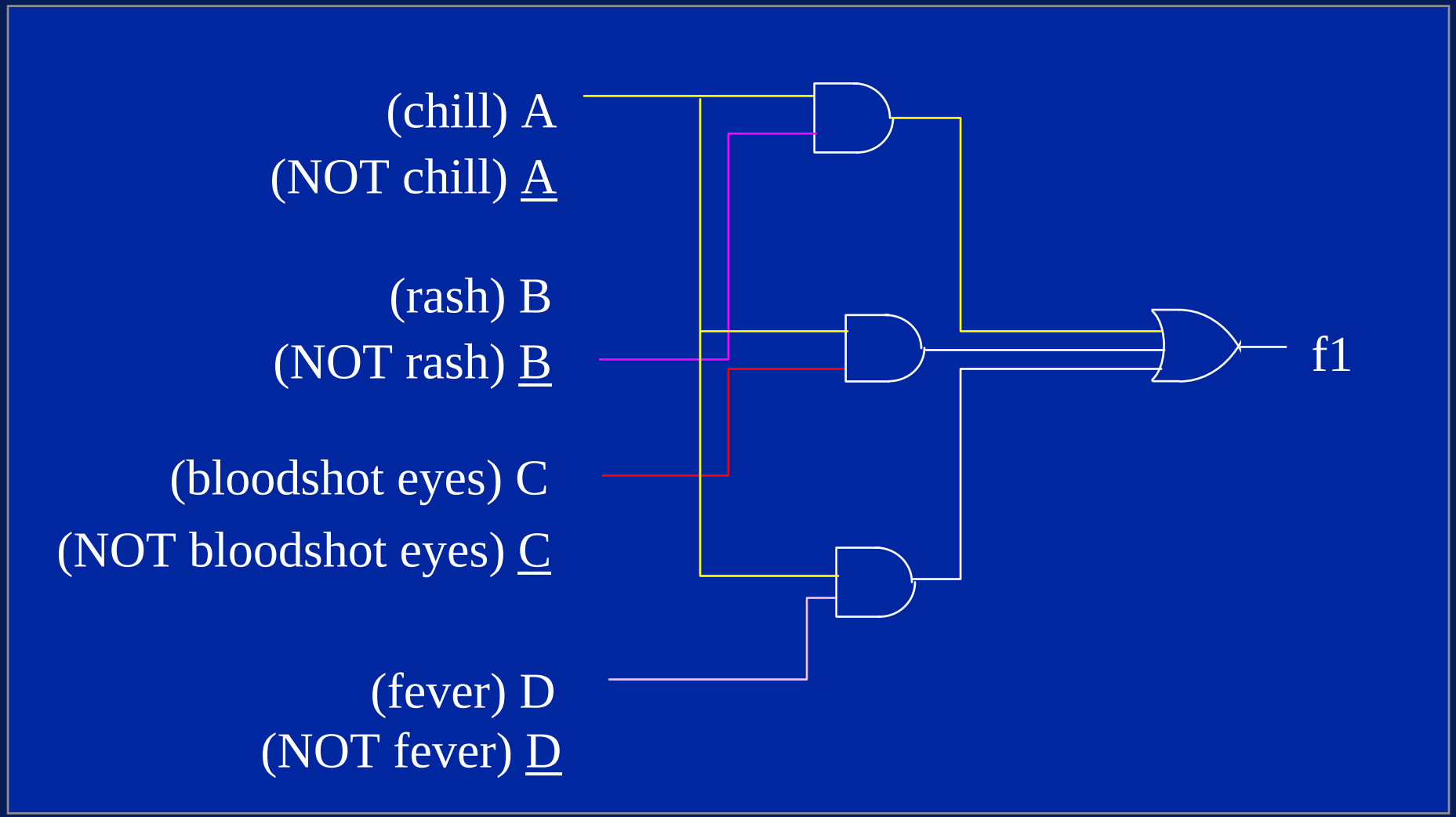


f1 =
A.B
+ A.C
+ A.D

Karnaugh Maps

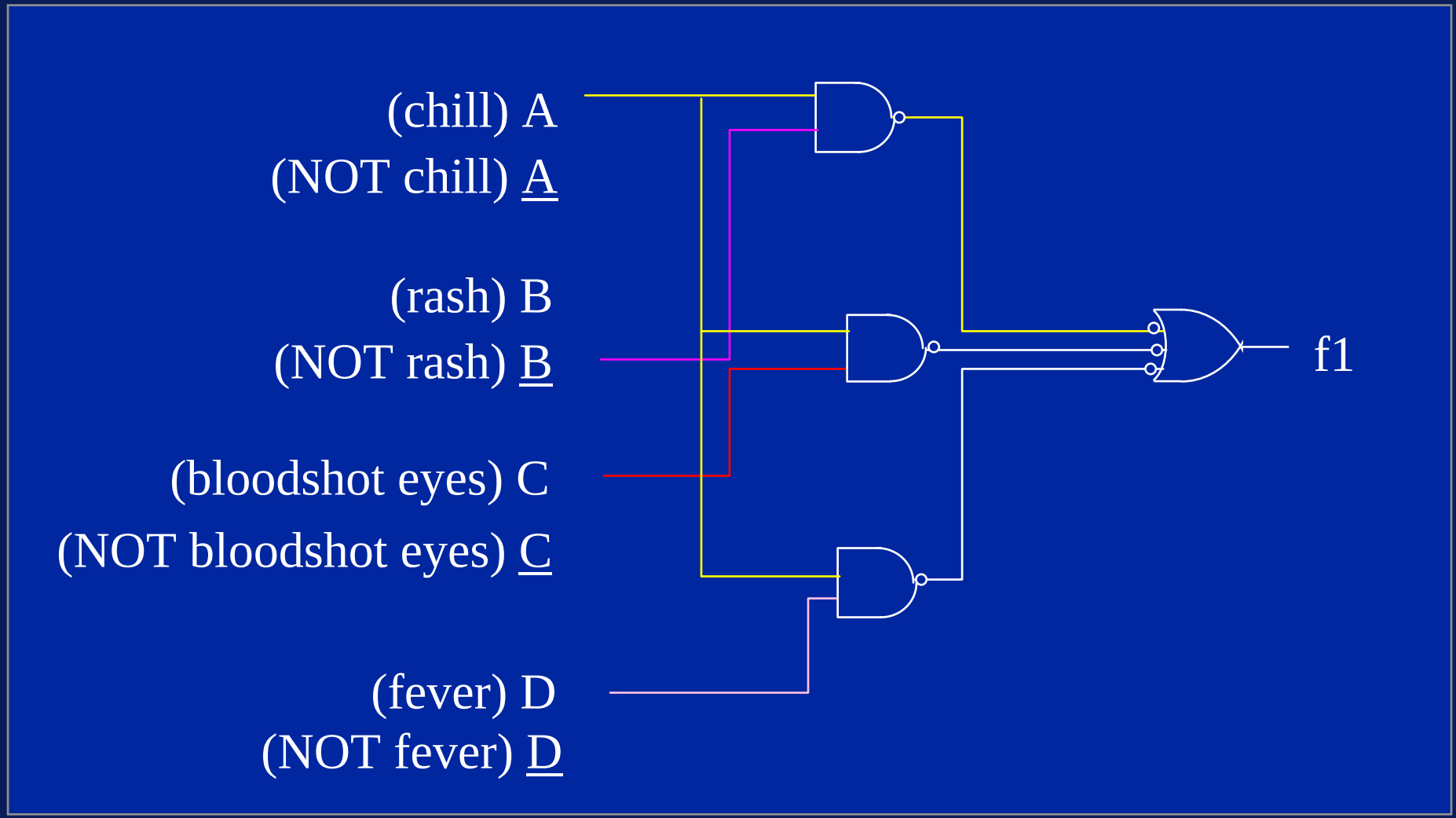
- Note: This simplification is better than we managed with our ‘hand’ simplification earlier!!

f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ + A.C + A.D Corresponding Digital Circuit



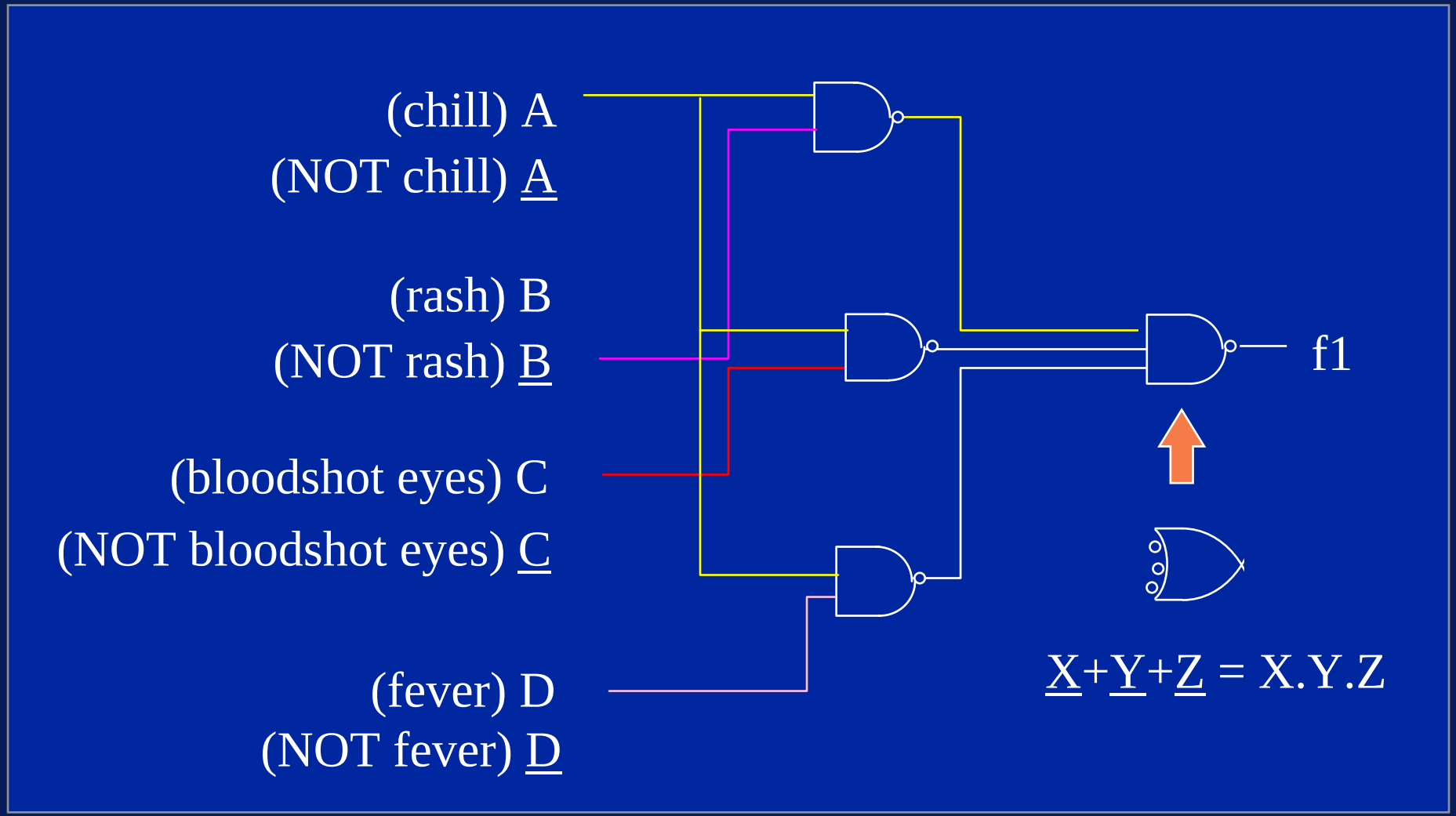
f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ + A.C + A.D

Corresponding NAND Digital Circuit



f1 = A . $\underline{B}$. $\underline{C}$. $\underline{D}$ + A.C + A.D

Corresponding NAND Digital Circuit



Exercise

- Use a truth table, minterms, and a Karnaugh map to simplify the following expression

$$f2 = \overline{A} \,.\, (B + C + D) + A \,.\, B \,.\, \overline{C} \,.\, \overline{D}$$

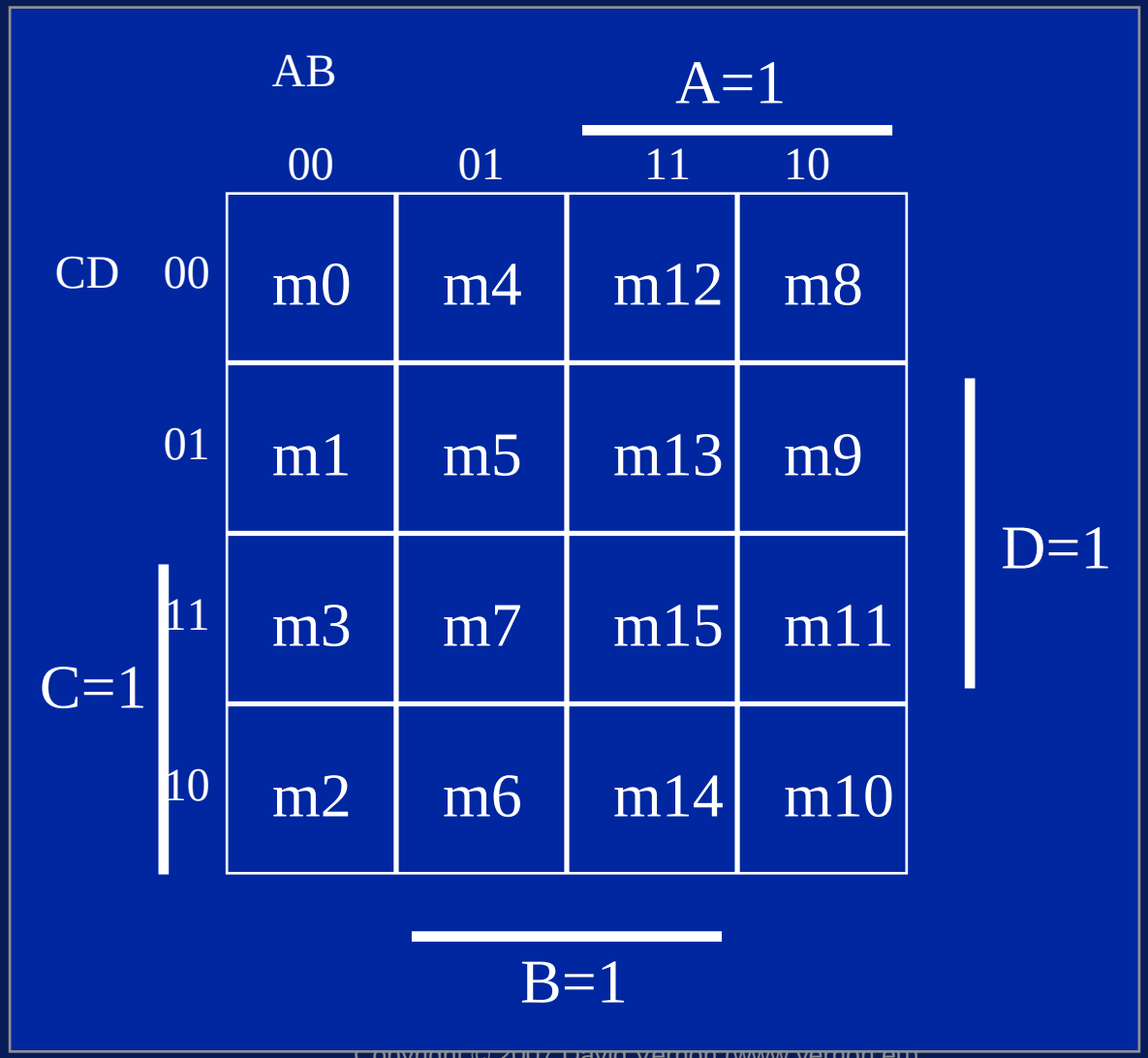
f2 = $\overline{A}$. (B + C + D) + A . B . $\overline{C}$. $\overline{D}$

Karnaugh Maps

Row	A	B	C	D	f1	f2	Minterm
0	0	0	0	0	0	0	m0
1	0	0	0	1	0	1	m1
2	0	0	1	0	0	1	m2
3	0	0	1	1	0	1	m3
4	0	1	0	0	0	1	m4
5	0	1	0	1	0	1	m5
6	0	1	1	0	0	1	m6
7	0	1	1	1	0	1	m7
8	1	0	0	0	1	0	m8
9	1	0	0	1	1	0	m9
10	1	0	1	0	1	0	m10
11	1	0	1	1	1	0	m11
12	1	1	0	0	0	1	m12
13	1	1	0	1	1	0	m13
14	1	1	1	0	1	0	m14
15	1	1	1	1	1	0	m15

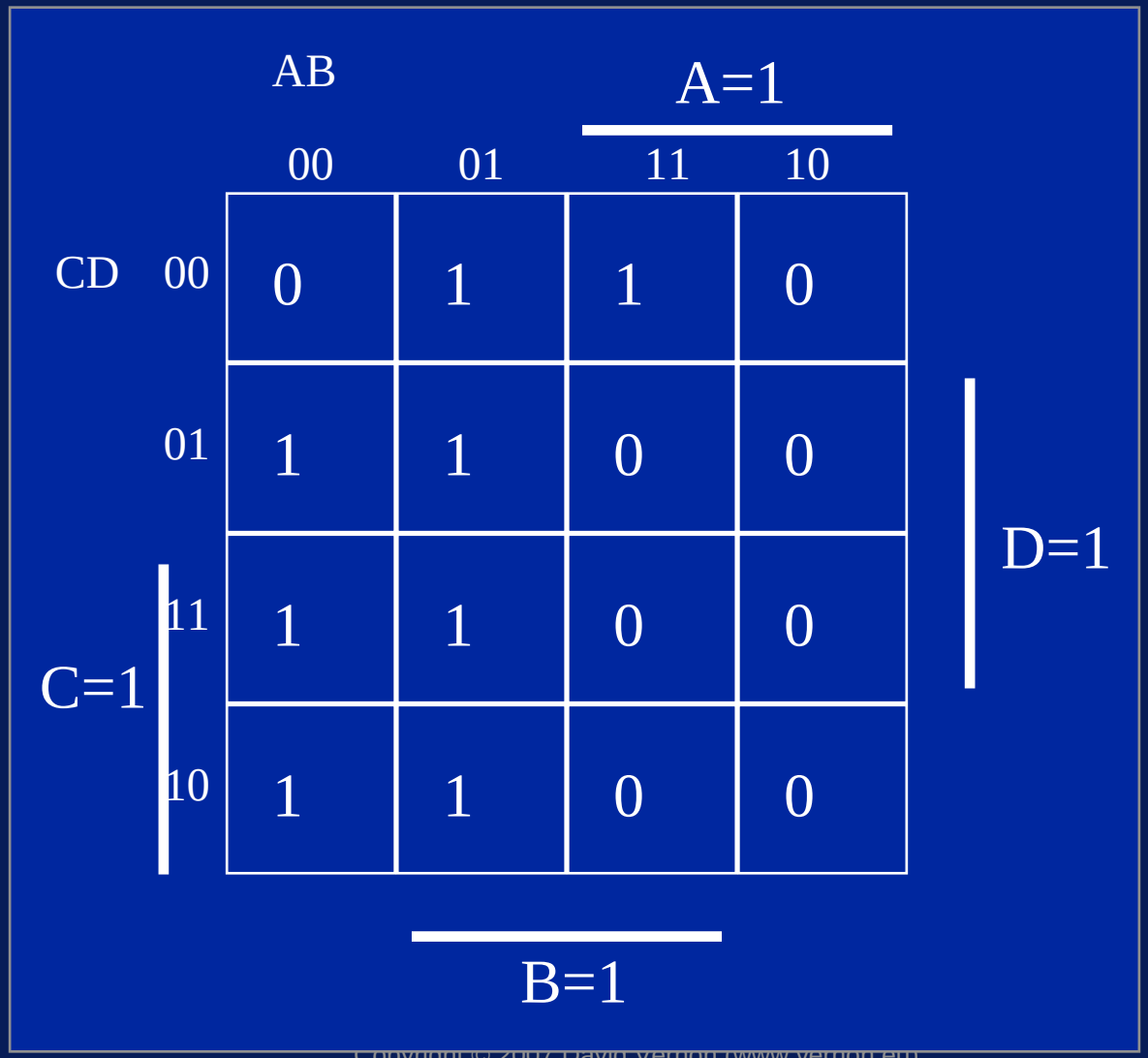
f2 = $\underline{A}$. (B + C + D) + A . B . $\underline{C}$. $\underline{D}$

Karnaugh Maps



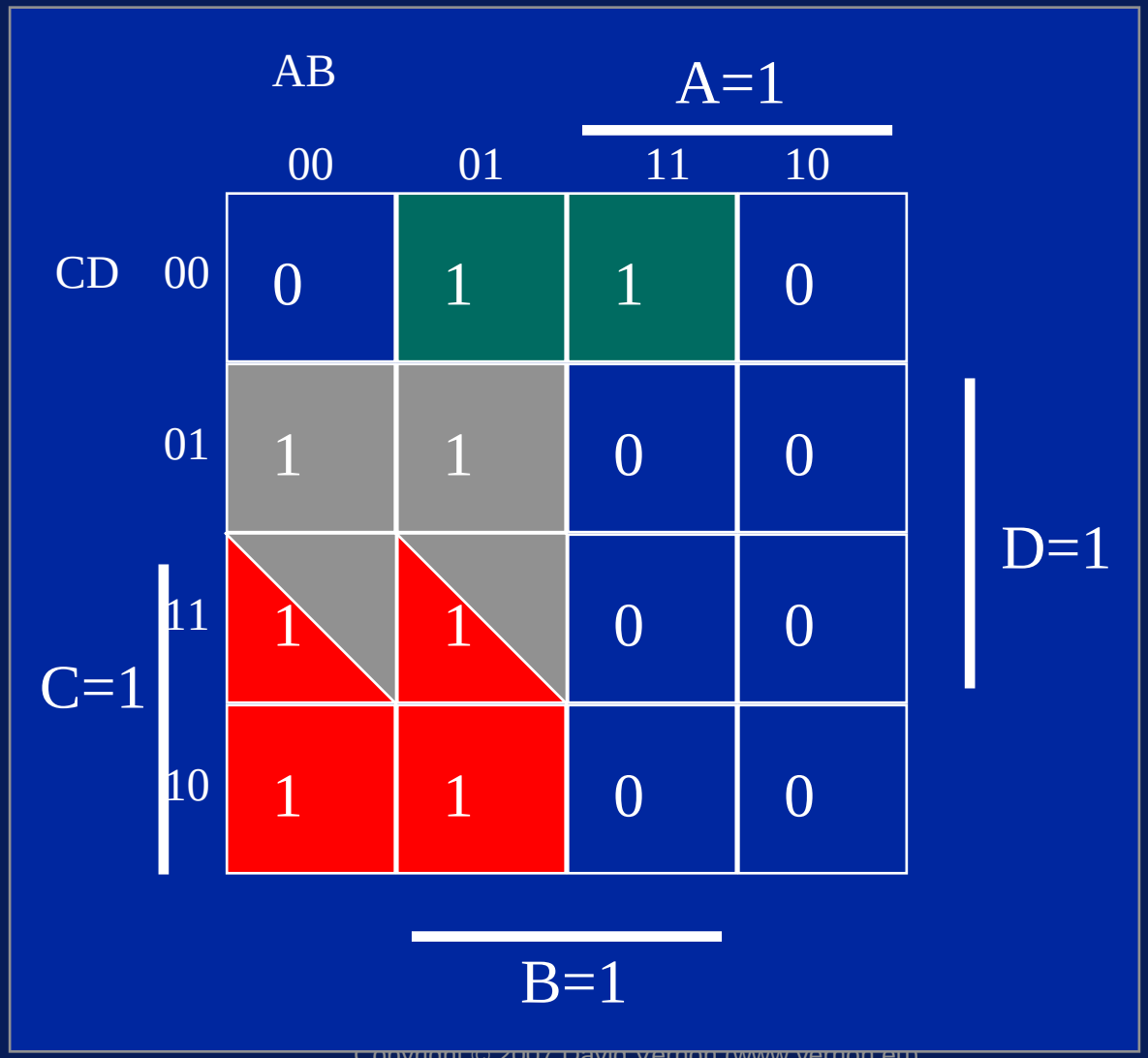
f2 = $\underline{A}$. (B + C + D) + A . B . $\underline{C}$. $\underline{D}$

Karnaugh Maps

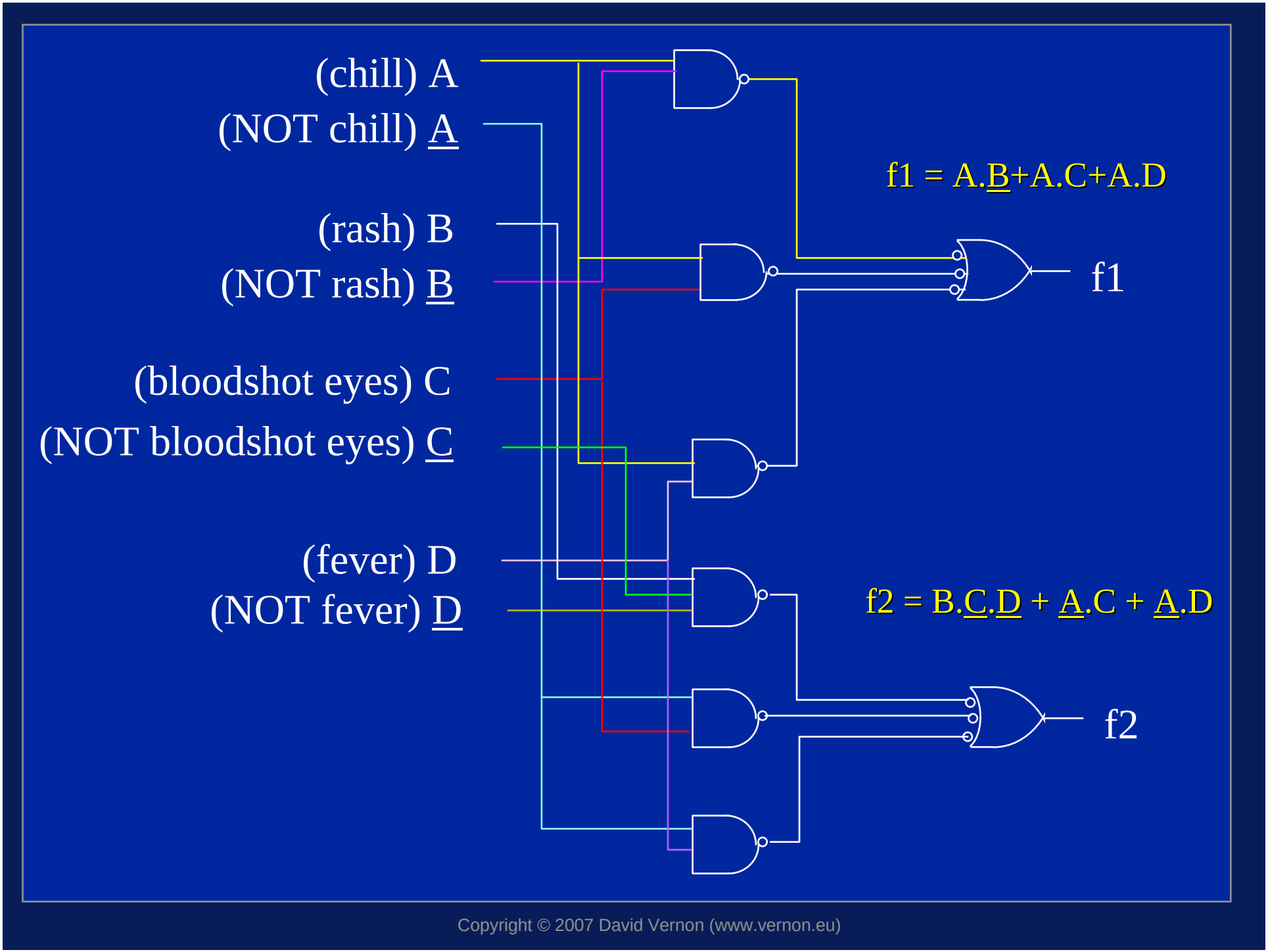


f2 = $\underline{A}$. (B + C + D) + A . B . $\underline{C}$. $\underline{D}$

Karnaugh Maps



f2 =
B.$\underline{C}$.$\underline{D}$
+ $\underline{A}$.C
+ $\underline{A}$.D



Simplification of Expressions

- Sometimes, a minterm never occurs in a system, i.e., the condition given by that minterm never arises
- In this instance, we can use either 1 or 0 in the Karnaugh map when simplifying expressions
- In fact, we use the convention that such conditions are ‘don’t care’conditions and are signified by X rather than 0/1
- We use the value 0 or 1 depending on which leads to the simplest expression

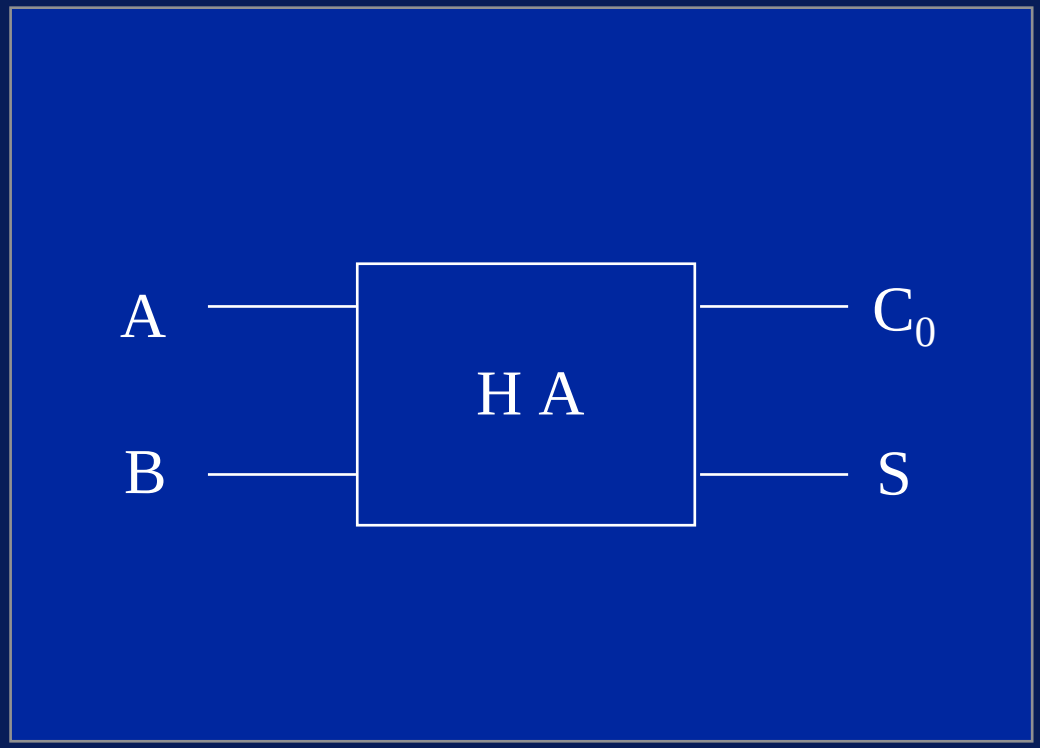
BINARY ARITHMETIC

Binary Addition

Half Adder

- A digital adder will add just two binary numbers
- When two binary digits (bits) A and B are added, two results are required:
  - the sum S
  - the ‘carry’ C_0 to the next place
- The circuit to do this is called a Half Adder (HA)

Half Adder



Half Adder

Truth Table for addition of two binary digits

A	B	S	C_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half-Adder

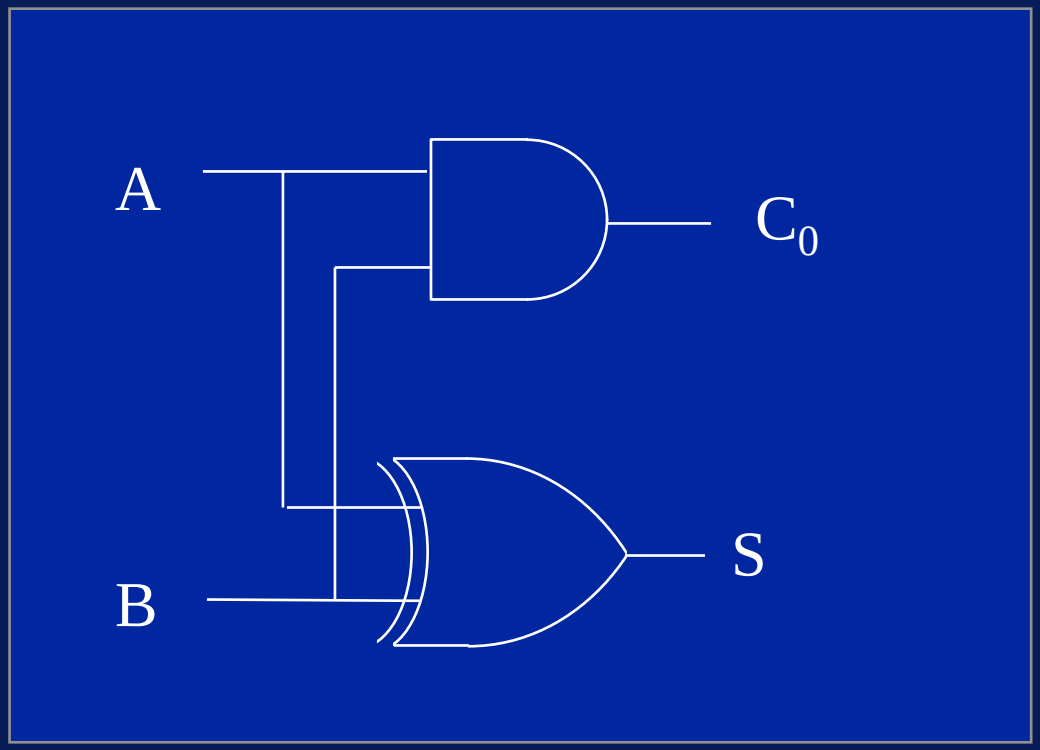
- From the truth table we see that:

$S = \underline{A} \cdot B + A \cdot \underline{B}$

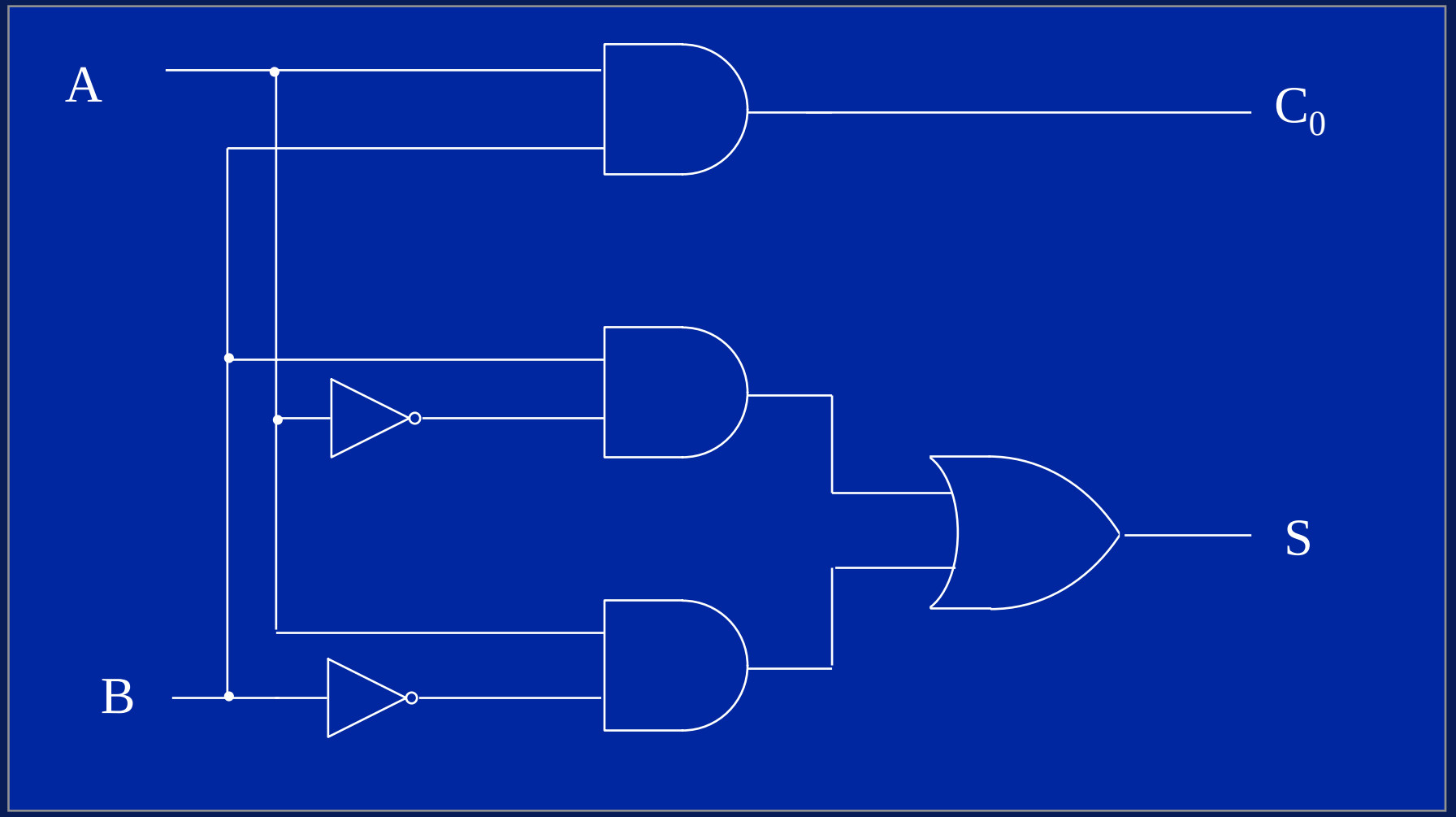
$= A \text{ XOR } B$

$C_0 = A \cdot B$

Half Adder



Half Adder



Half Adder

- Note that the S output is separated from the inputs by 3 levels of gates
  - referred to as logical depth of 3
- While the C0 output is separated by just 1 level
  - logical depth of 1
- Because of propagation delays, this means that the carry out will be produced before the sum
  - This may cause problems in some circumstances

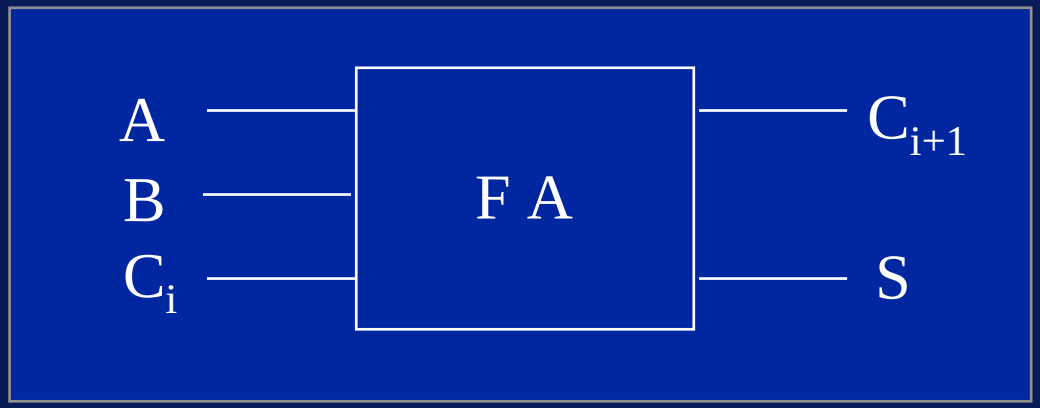
Full Adder

- In order to add together multiple-digit numbers, we need a slightly more complicated circuit
  - Need to add the two digits and the carry out from the ‘previous’ or less significant digit
  - Only in the addition of the right-most digits can we ignore this carry

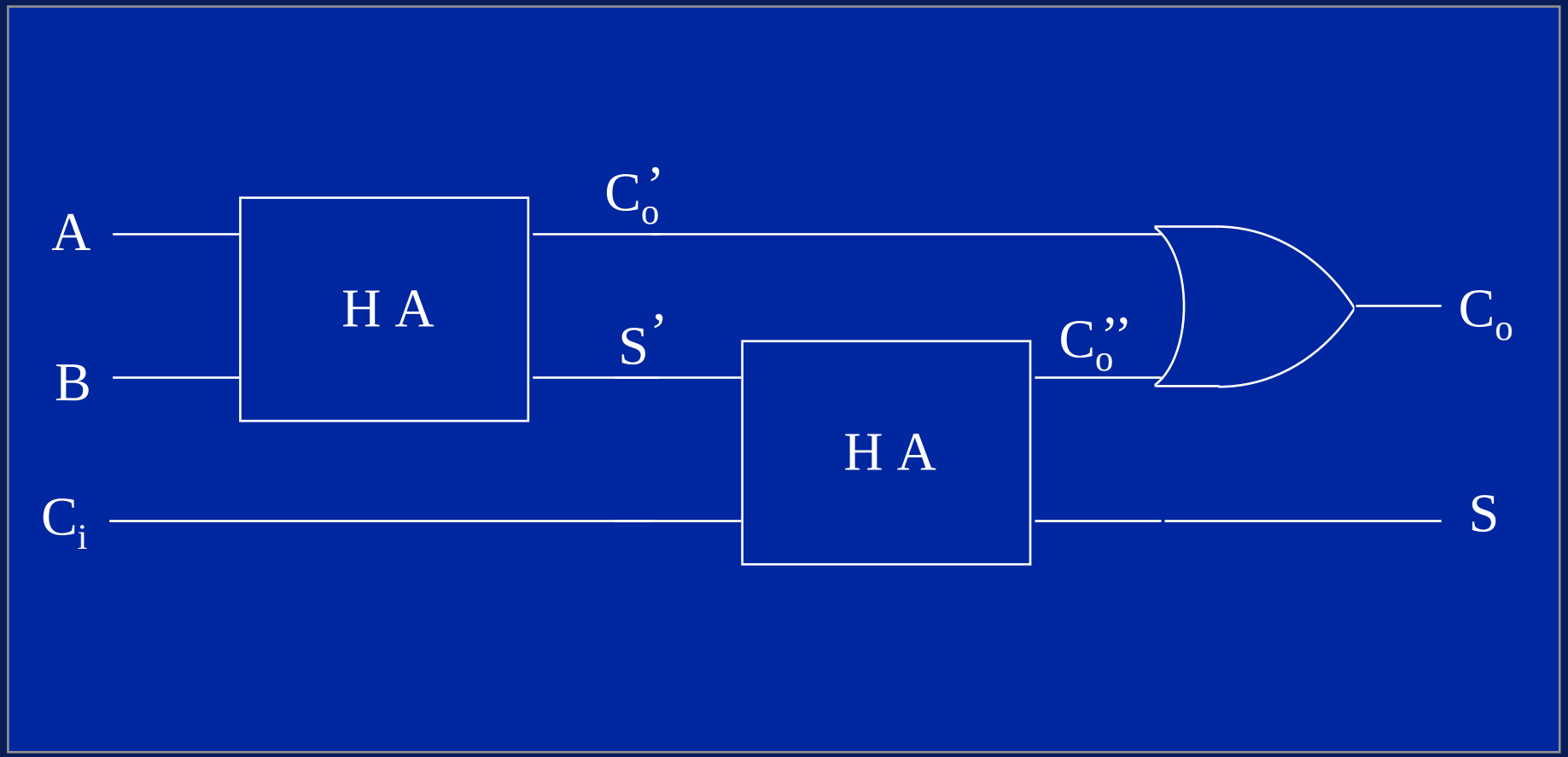
Full Adder

Full Adder

- The circuit to add three binary digits (two operands and a carry bit) is called a Full Adder (FA)
- It can be implemented using two half adders



Full Adder



Full Adder

- Addition is carried out in two stages
  - add bits A and B to produce
    - » partial sum S’
    - » and (the first) intermediate output carry C_o’
  - add partial sum S’ and input carry C_i from previous stage to produce
    - » final sum
    - » and (the second) intermediate output carry C_o’’
  - We then need to combine the intermediate carry bits (they don’t have to be added)

Full Adder

A	B	Ci	S’	C_o’	C_o’’	C_o	S	A ⊕ B	A ⊕ B ⊕ C_i
0	0	0							
0	0	1							
0	1	0							
0	1	1							
1	0	0							
1	0	1							
1	1	0							
1	1	1							

Full Adder

A	B	Ci	S’	C_o’	C_o’’	C_o	S	A ⊕ B	A ⊕ B ⊕ C_i
0	0	0	0	0					
0	0	1	0	0					
0	1	0	1	0					
0	1	1	1	0					
1	0	0	1	0					
1	0	1	1	0					
1	1	0	0	1					
1	1	1	0	1					

Full Adder

A	B	Ci	S’	C_o’	C_o’’	C_o	S	$A \oplus B$	$A \oplus B \oplus C_i$
0	0	0	0	0	0		0		
0	0	1	0	0	0		1		
0	1	0	1	0	0		1		
0	1	1	1	0	1		0		
1	0	0	1	0	0		1		
1	0	1	1	0	1		0		
1	1	0	0	1	0		0		
1	1	1	0	1	0		1		

Full Adder

A	B	Ci	S’	C_o’	C_o’’	C_o	S	$A \oplus B$	$A \oplus B \oplus C_i$
0	0	0	0	0	0	0	0		
0	0	1	0	0	0	0	1		
0	1	0	1	0	0	0	1		
0	1	1	1	0	1	1	0		
1	0	0	1	0	0	0	1		
1	0	1	1	0	1	1	0		
1	1	0	0	1	0	1	0		
1	1	1	0	1	0	1	1		

Full Adder

A	B	Ci	S’	C_o’	C_o’’	C_o	S	$A \oplus B$	$A \oplus B \oplus C_i$
0	0	0	0	0	0	0	0	0	
0	0	1	0	0	0	0	1	0	
0	1	0	1	0	0	0	1	1	
0	1	1	1	0	1	1	0	1	
1	0	0	1	0	0	0	1	1	
1	0	1	1	0	1	1	0	1	
1	1	0	0	1	0	1	0	0	
1	1	1	0	1	0	1	1	0	

Full Adder

A	B	Ci	S’	C_o’	C_o’’	C_o	S	$A \oplus B$	$A \oplus B \oplus C_i$
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	1
0	1	0	1	0	0	0	1	1	1
0	1	1	1	0	1	1	0	1	0
1	0	0	1	0	0	0	1	1	1
1	0	1	1	0	1	1	0	1	0
1	1	0	0	1	0	1	0	0	0
1	1	1	0	1	0	1	1	0	1

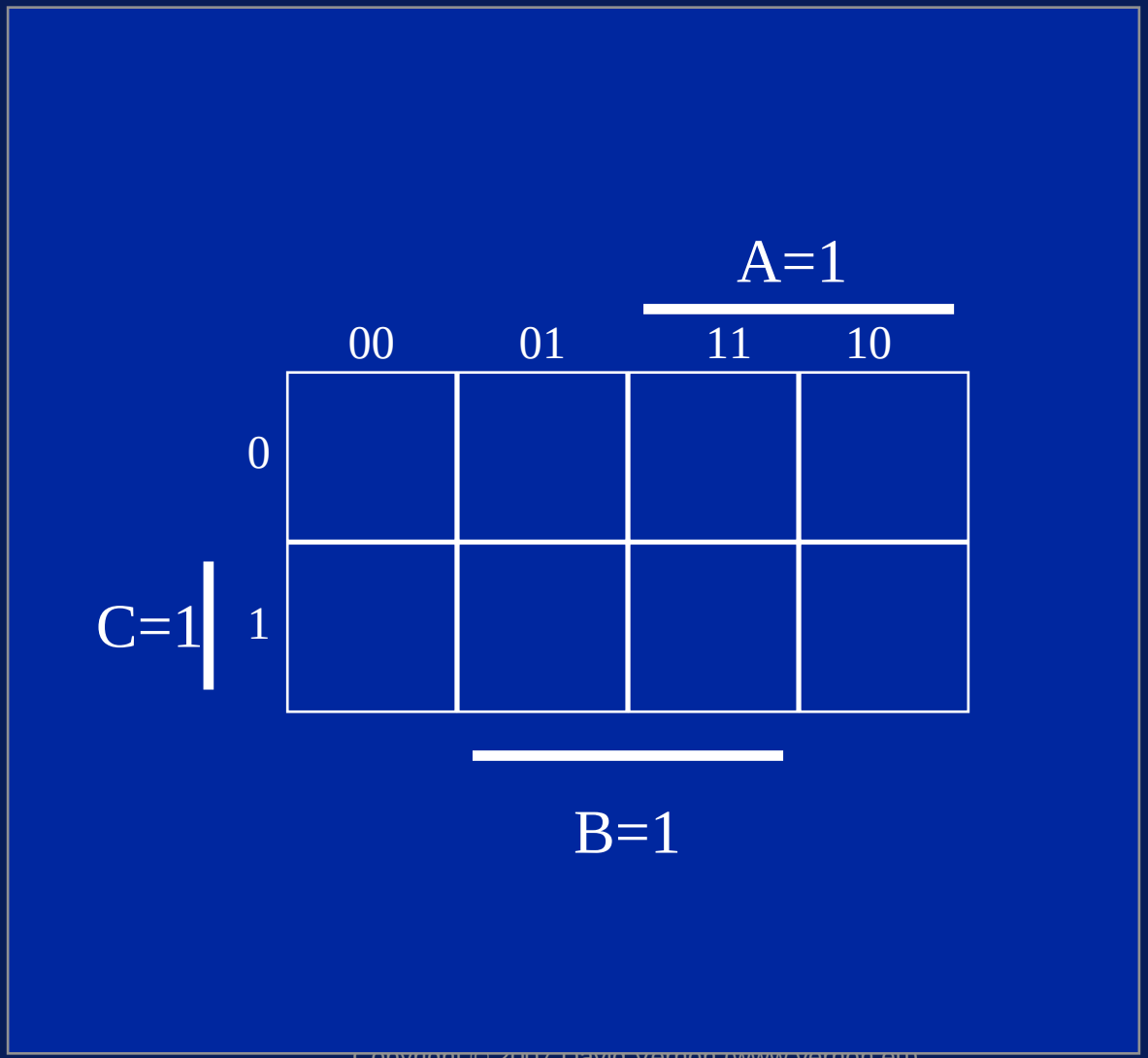
Full Adder

- A few observations
- The truth table demonstrates why $C_o = C_o' + C_o''$
- It's clear also that $S = A \oplus B \oplus C_i$

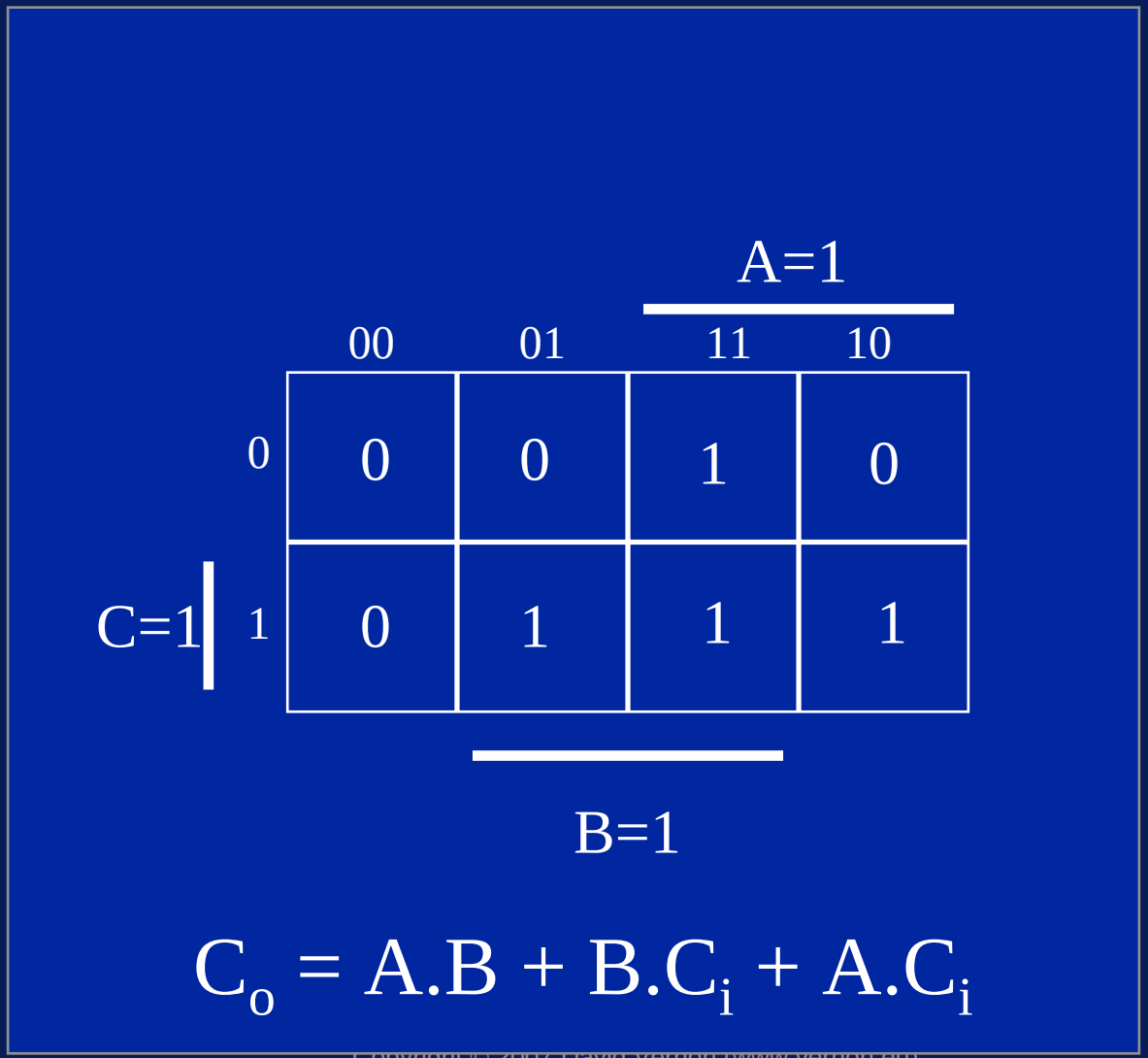
- Also, we could obtain a simplified expression for C_o from a Karnaugh Map

$C_o = A.B + B.C_i + A.C_i$

3-Variable Karnaugh Map



3-Variable Karnaugh Map



$$C_o = A.B + B.C_i + A.C_i$$

Full Adder

- So, instead of implementing a full adder as two half adders, we could implement it directly from the gating:

$$S = A \oplus B \oplus C_i$$
$$C_o = A.B + B.C_i + A.C_i$$

Full Adder

- Irrespective of the implementation of a full adder, we can combine them to add multiple digit binary numbers

4-Bit Binary Adder