CS 101: Computer Programming and Utilization

July-Nov 2017

Umesh Bellur
(cs101@cse.iitb.ac.in)

Lecture 2: How Computers Work (A very high level view)

What's a Computer?

Very simply:

A computer is **<u>a large* electric circuit</u>** with parts that

- reads numbers from external world (Input)
- Stores numbers (Storage)
- Performs arithmetic on numbers (Processing)
- Sends numbers to external world (Output)

*How large can large be?

•Physically small: ~1000 mm^2

•Logically large: ~10^9 transistors

How To Use A Computer To Solve Real-Life Problems?

1. Express the problem as **calculations** on numbers

2. Think of a solution in terms of the computations need to be performed, possibly repeatedly and conditionally

   = program

3. Feed the
   1. the data in term of number, AND
   2. The program to the computer (also numbers)

4. Get the output on a screen or elsewhere (?)

Outline

- **Expressing real life problems as numerical problems**
  - Picture processing
  - Predicting the weather
  - Understanding and Processing language

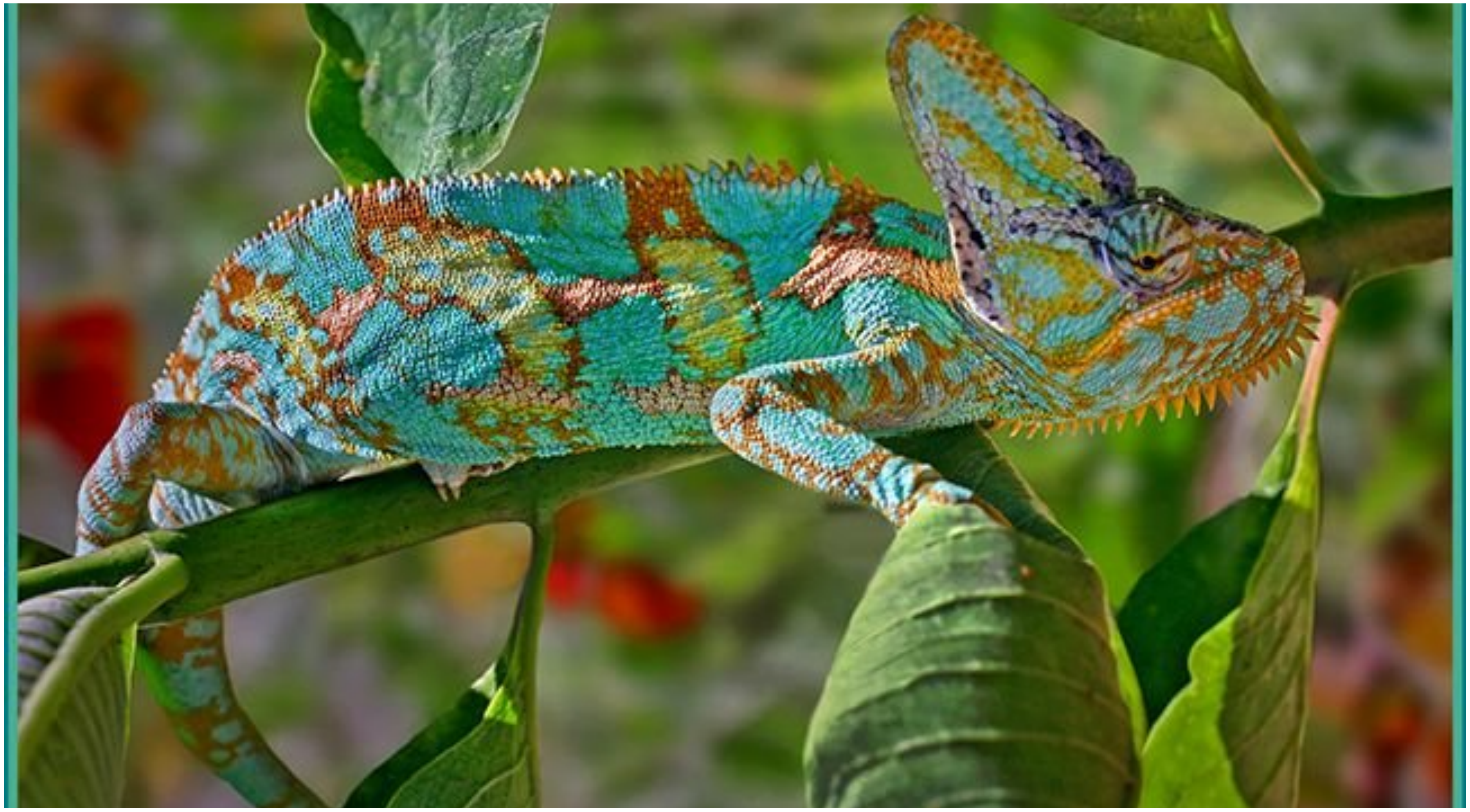
- High level **design** of a computer – how does it actually work?
  - Digital circuits
  - Parts of a computer
  - Stored program, compilation

Outline

- **Expressing real life problems as numerical problems**
  - Picture processing
  - Predicting the weather
  - Understanding and Processing language

- High level design of a computer
  - Digital circuits
  - Parts of a computer
  - Stored program, compilation

“What is in this picture?”



Can we ask this question to a computer and get an answer?

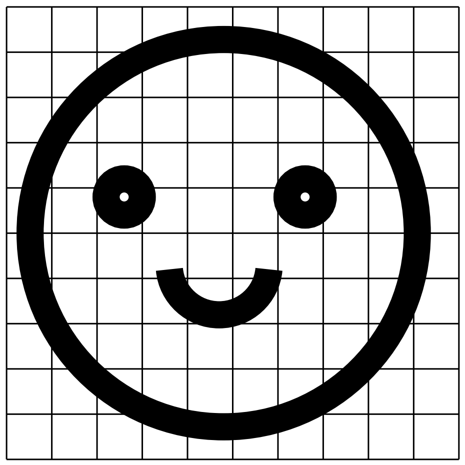
Start small

- Black and white picture representation and comprehension

How to represent black and white pictures?

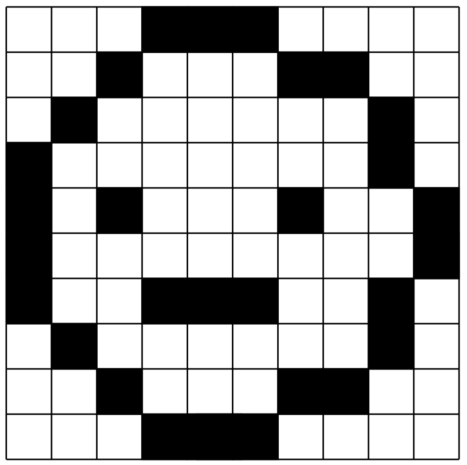
- Suppose the picture is 10cm x 10cm
- Divide it into 0.1 mm x 0.1 mm squares
- The number of squares (or pixels) is 1000 x 1000
- If a square is mostly white, represent it by 0
- If a square is mostly black, represent it by 1
- Thus, our picture = 1 million numbers

Picture Representation


(a)

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

(b)


(c)

Can I ask – does this picture have a vertical line?

- Input:

  A sequence of 1 million numbers (0 or 1) representing a 10cm x 10cm black & white picture
- What property does the sequence need to have if it is to contain a vertical line somewhere?
- All 0s, except for 1s at positions

  i, i+1000, i+2000, i+3000, i+4000,…

  for some *i*

A question about a picture *converted into* ***a question about numbers!***

Remarks

- Better representation if picture divided into more cells.
  - The finer grained this is, the better the "**resolution**".
- Pictures with different "gray levels": use numbers 0, 0.1, …, 1.0 to represent level of darkness rather than just 0, 1.
- Pictures with colors: picture = 3 sequences
  - sequence for red component,
  - sequence for blue component,
  - sequence for green component
- Apply the mathematical function of R,G and B to get the actual color.

More *Image Recognition* problems

- Is the picture largely red in color?
- Is there a square in the picture?
- Are two pictures similar?
- ...

Getting back to: Does the picture contain a chameleon?

- This question will need to be expressed as:

- Does the sequence of numbers representing the picture contain a subsequence satisfying certain properties? MUCH HARDER PROBLEM!

- Which properties?
  - Enormous ingenuity needed to specify.
  - Very difficult problem
  - Main concern of a subject called Computer Vision

Second Example: Weather Prediction

- Divide the surface of the earth into small regions (like pixels)

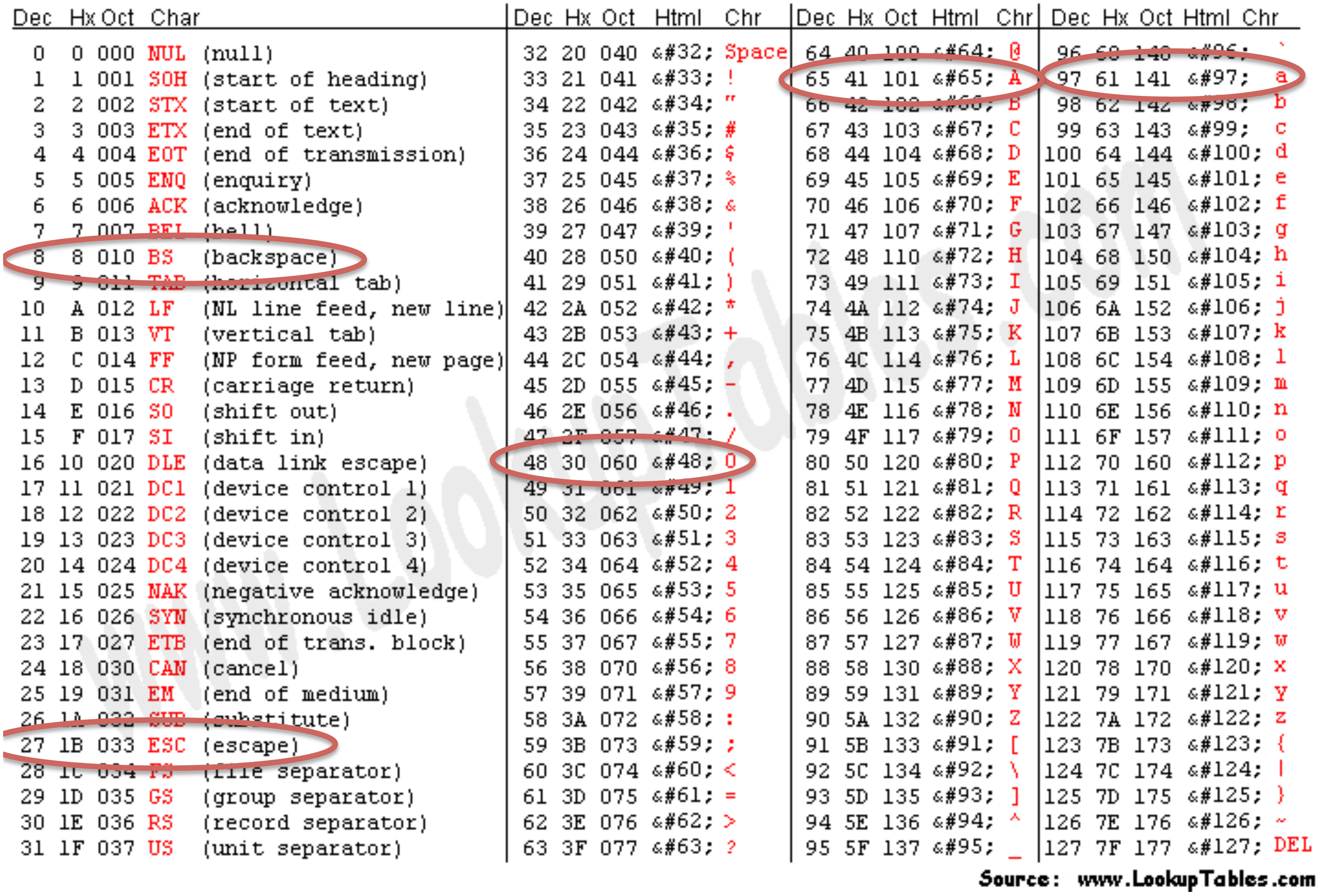
- For each region *i*, and each time *t*, let $p_{i,t}$, $c_{i,t}$, $h_{i,t}$ represent pressure, temperature, humidity

- Laws of physics will tell us what relationships $p_{i,t}$, $c_{\mathrm{i,t}}$, $h_{i,t}$ must satisfy across *(i,t)* for certain types of weather.

- We can measure Pressure, Humidity and Temperature values for past times, and predict for the future
  - The details are complicated.

Example 3: Representing a language using Numbers

- Define a numeric code for representing letters
- ASCII (American Standard Code for Information Interchange) is the commonly used code
  - Letter 'a' = 97 in ASCII, 'b' = 98, …
  - Uppercase letters, symbols, digits also have codes
  - Code also for space character
- Words = sequences of ASCII codes of letters in the word
  - For Example: **'computer' = 99, 111,109,112,117,116,101,114**



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Representing A Language Using Numbers

- Sentences/paragraphs = larger sequences
- Does the word “computer” occur in a paragraph?
- Does a certain sequence of numbers occur inside another sequence of numbers?

Summary

- Questions about pictures, weather, documents can be converted to questions about properties of number sequences

- Finding answers requires solving interesting math problems

- Games: How will you represent Tic-Tac-Toe as a question on numbers?
- How will you represent Chess playing as a question on numbers?

Outline

Examples of expressing real life problems as numerical problems

- Picture processing
- Predicting the weather
- Understanding and processing language

<u>High level design of a computer</u>

- Digital circuits
- Parts of a computer
- Stored program, compilation

The Hardware
(A very high level glimpse)

- How do we store numbers in hardware? Why Binary systems?
- How is an instruction expressed in hardware?
- How is it executed?
- How do we output the numbers?

Where it all began….

- ***Alan Turing's insight**** – given any computer algorithm, we can construct a Turing machine that simulates the algorithm's logic

  => We need a machine to manipulate symbols and a definitive method (mathematical model) of doing so.

- Question is: What are these symbols?

Choices of Symbols

1. ✗ Real numbers – Infinite precision and too many to represent (no compact representation)
2. ✗ Analog signals – arbitrary and finite precision. Errors accumulate with number of computations.
3. ✓ Digital signals – a finite set of discretized symbols.
   1. Unary – just one symbol. Complexity of ✗ manipulation AND representation . O(n) is unacceptable.

Next symbol set

- **<u>Decimal, base 10</u>**, the ten symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Crucial idea: ***Weighted positional notation***
- N written as

$$d_i x10^i + \ldots + d_1 x10^1 + d_0 x10^0 + d_{-1} x10^{-1} + \ldots + d$$

10^i-place 10s-place 1s-place Fractional part

Most significant digit Least significant di

- Now N written in O(log N) symbols; MUCH shorter
- Addition can be done in constant time

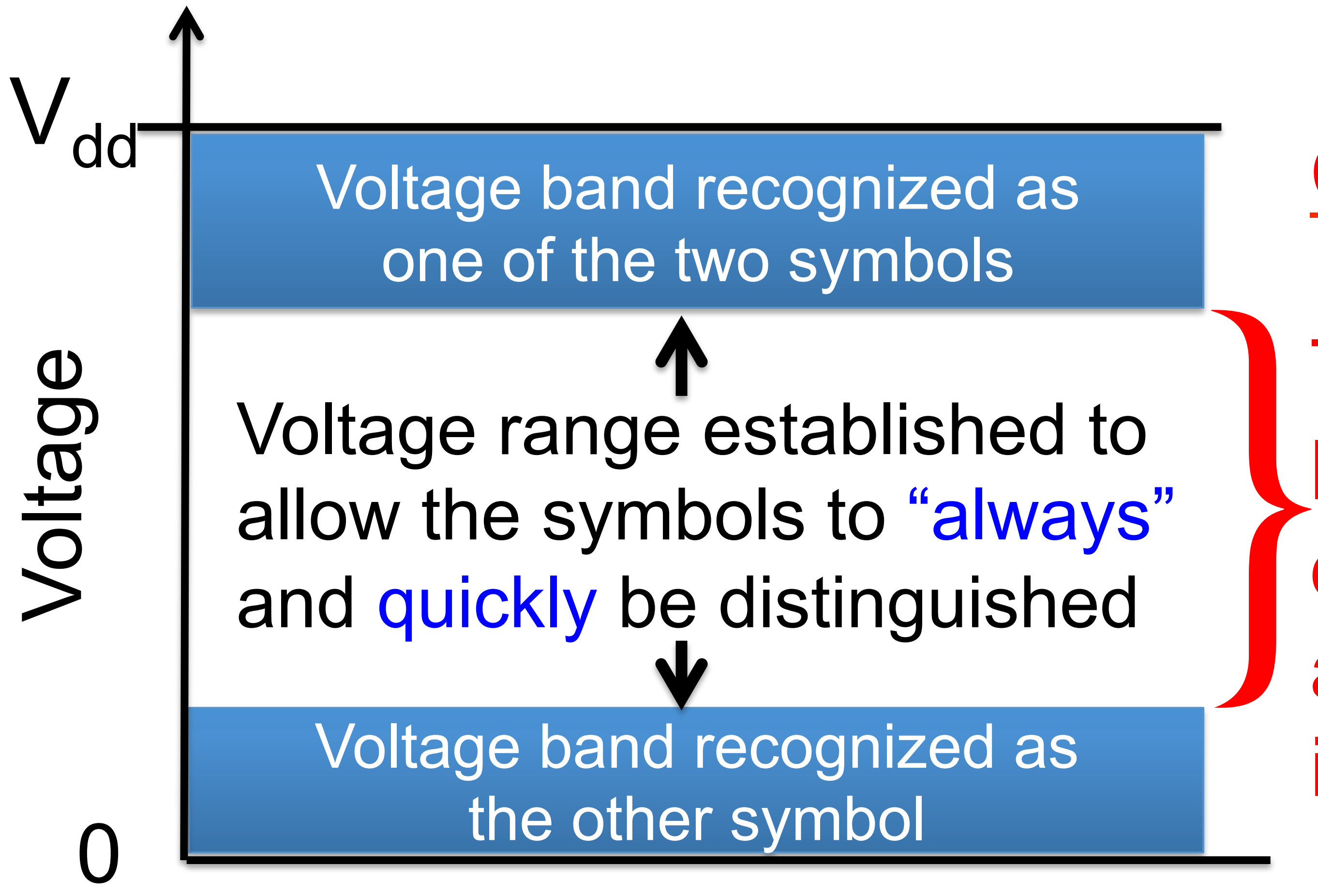
Representing the symbols

- Electrically, we would need ***ten distinct voltage levels*** and allowance for imprecision of the electrical parts
- Typical electrical part has ±5% variance from nominal specification
- Also we want FAST and settling in on one of ten possibilities is not going to be as fast as settling in on fewer possibilities

And so …..

- We use a ***two-symbol*** representation, aka Binary
  - A logarithmic representation scheme to avoid the time complexity of unary
  - Electrical circuit need only move between and settle into one of two different voltage levels: FASTEST choice
- Distinguishing symbols requires that the voltages be sufficiently different and circuit imprecision requires acceptance of a range of voltages for each symbol
  - Tolerant of imprecision

Electric binary symbol representation


V_{dd}
Voltage
0
Voltage band recognized as one of the two symbols
Voltage range established to allow the symbols to "always" and quickly be distinguished
Voltage band recognized as the other symbol

Crucial idea

This "gap" is how analog devices form a circuit that is "digital"

Digital Circuits

- Building blocks of computers
- Circuits have wires that carry current, and are at a certain electric potential.
- Digital circuits: interpret electrical potential/voltage as numbers.
- Simplest convention
  - Voltage above 3.5 volt = number 1
  - Voltage between 0 and 0.5 volt = number 0
  - Circuit designed so that voltage will never be between 0.5 and 3.5 volt, hence no ambiguity.

Two Symbol System

Denote statement truth values using 2 symbols, typically {T,F} or {0,1}

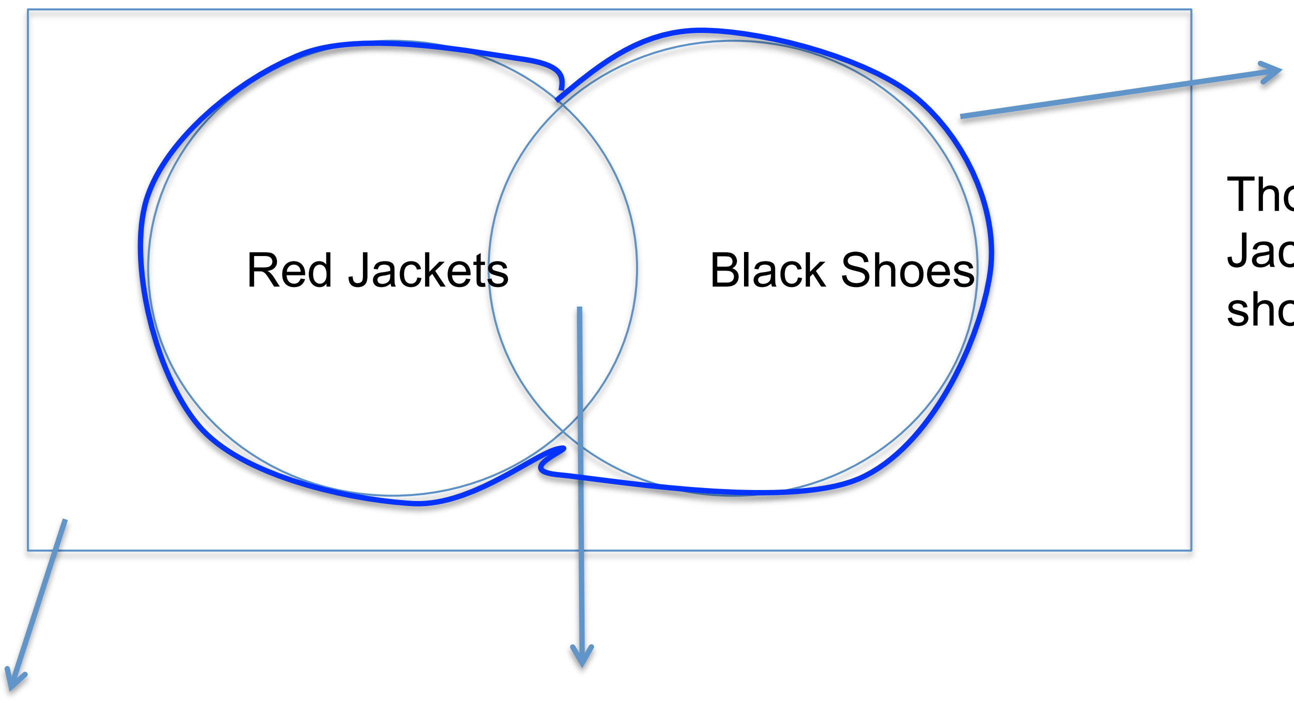
Example statements

Statement A: “10 < 7”

Statement B: “Today is Friday”

Each statement can TRUE (1) or FALSE (0).

Combining Statements



How about this?

Those with Red Jackets **OR** black shoes **OR** Both

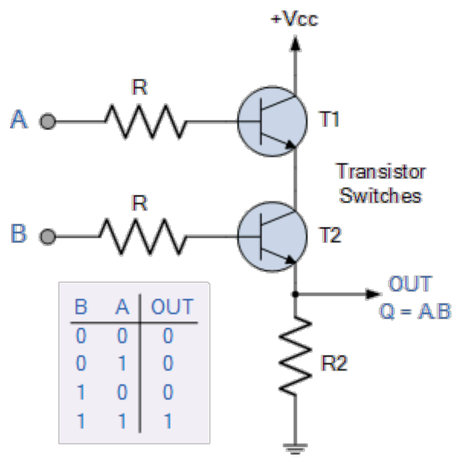
Those **NOT** wearing either Red Jackets or Black Shoes

What does this represent?

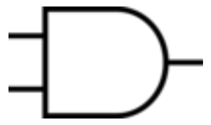
Those with Red Jackets **AND** black shoes

The simplest building blocks - Gates

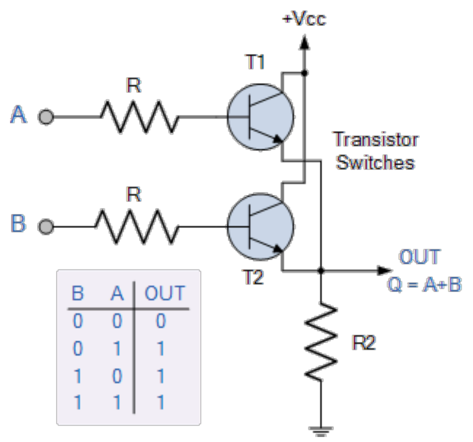
Circuits that realizes the boolean AND, OR and Complement operations



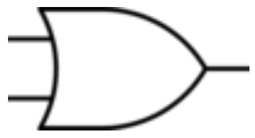
B	A	OUT
0	0	0
0	1	0
1	0	0
1	1	1



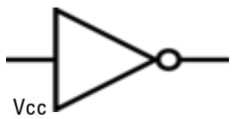
AND



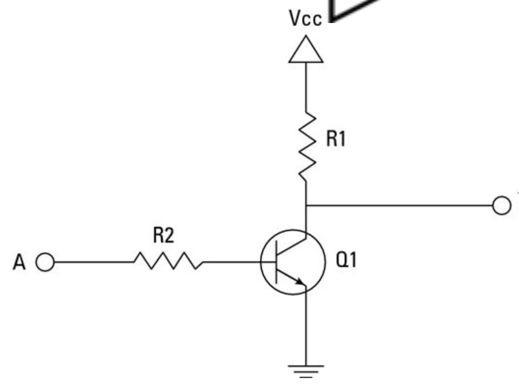
B	A	OUT
0	0	0
0	1	1
1	0	1
1	1	1



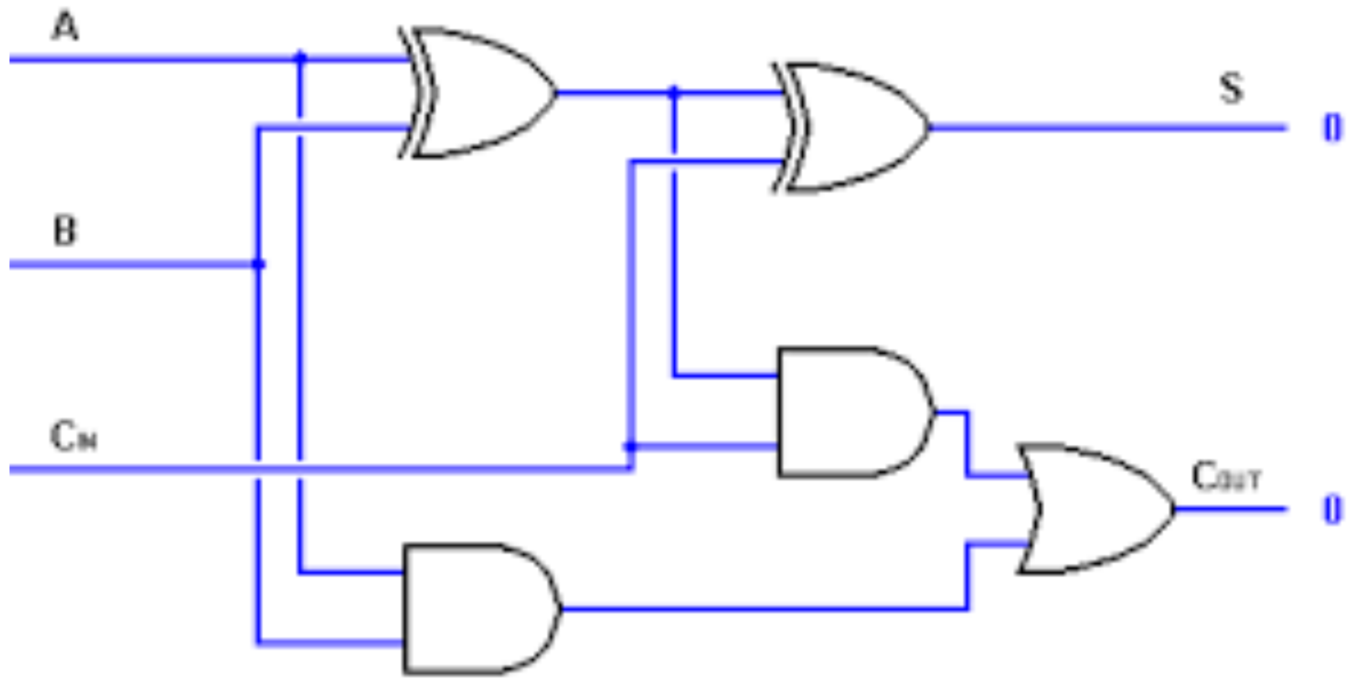
OR



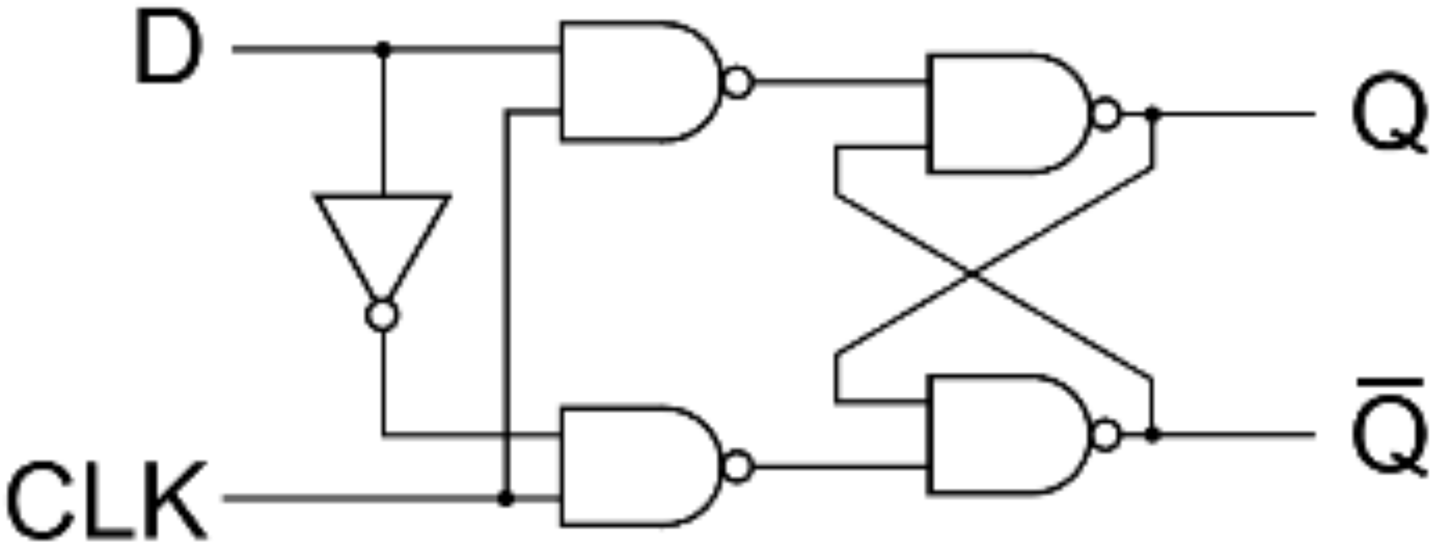
NOT



More complex blocks – Adders



Memories – Persistence Circuits



Representing Numbers

- How to represent numbers using this capability?
- Key idea : Binary number system
  - Represent all numbers using only 1's and 0's
  - Also called "Bits": "Binary digits"
- Details on conversion in next lecture
  - For now assume that all decimal numbers can be converted into binary numbers...i.e. into a sequence of 1' s and 0's

Representing Numbers on circuits

Key idea:
Store each bit of the number on a separate wire and element (transistor)

Example: **25 Decimal = 11001 binary**

- Use a **5 part circuit**.
- Store high charge on 1st, 2nd, 5th, and low charge on 3rd, 4th

To ***transmit*** 25 from one part of the computer to another

- Use 5 wires and raise the wires to appropriate voltages at one end.
- Sense the voltages at the other end

Bits, bytes, half-words, words

Bit = 1 Transistor/wire

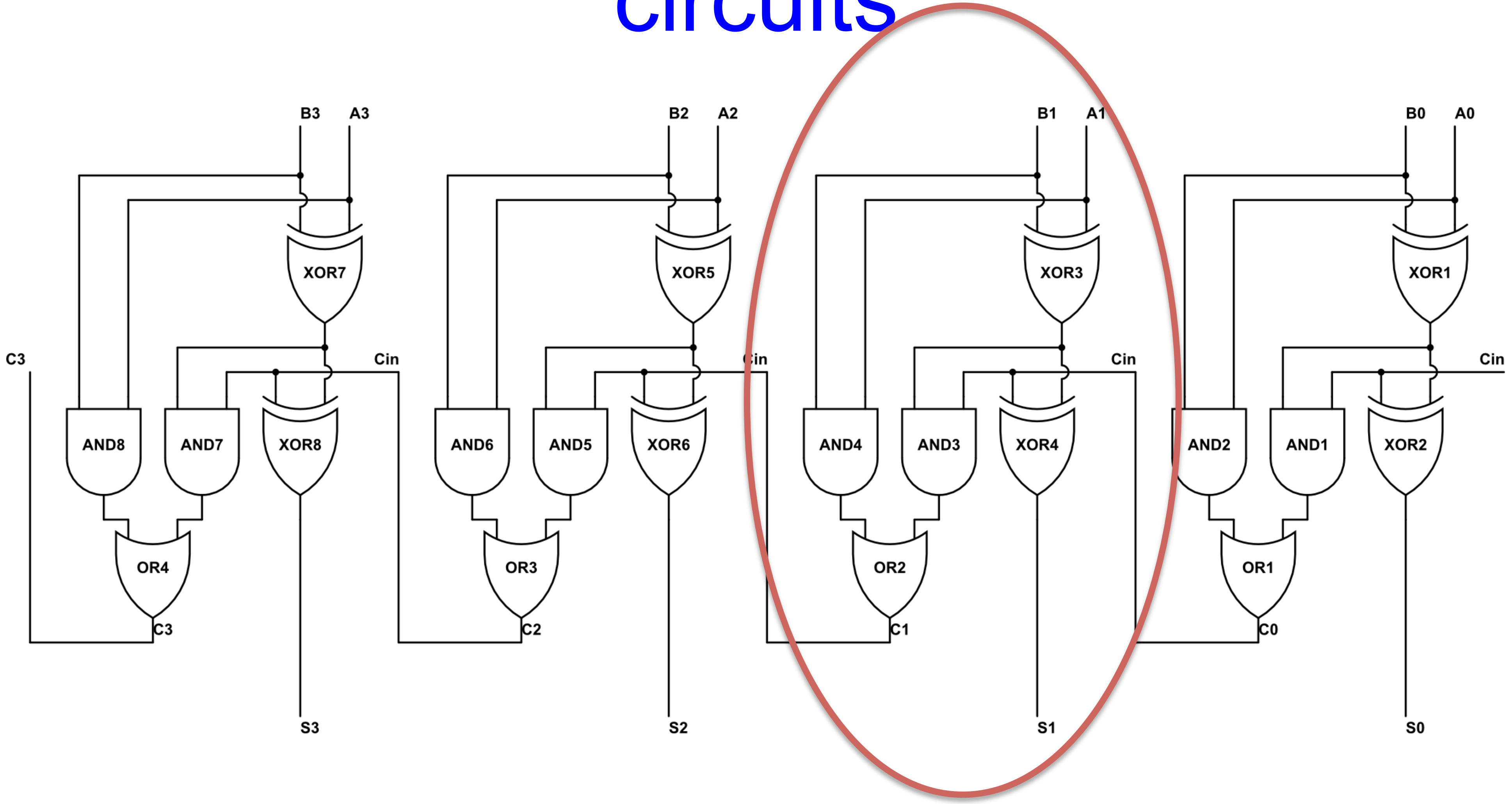
byte = 8 bits

half-word = 16 bits

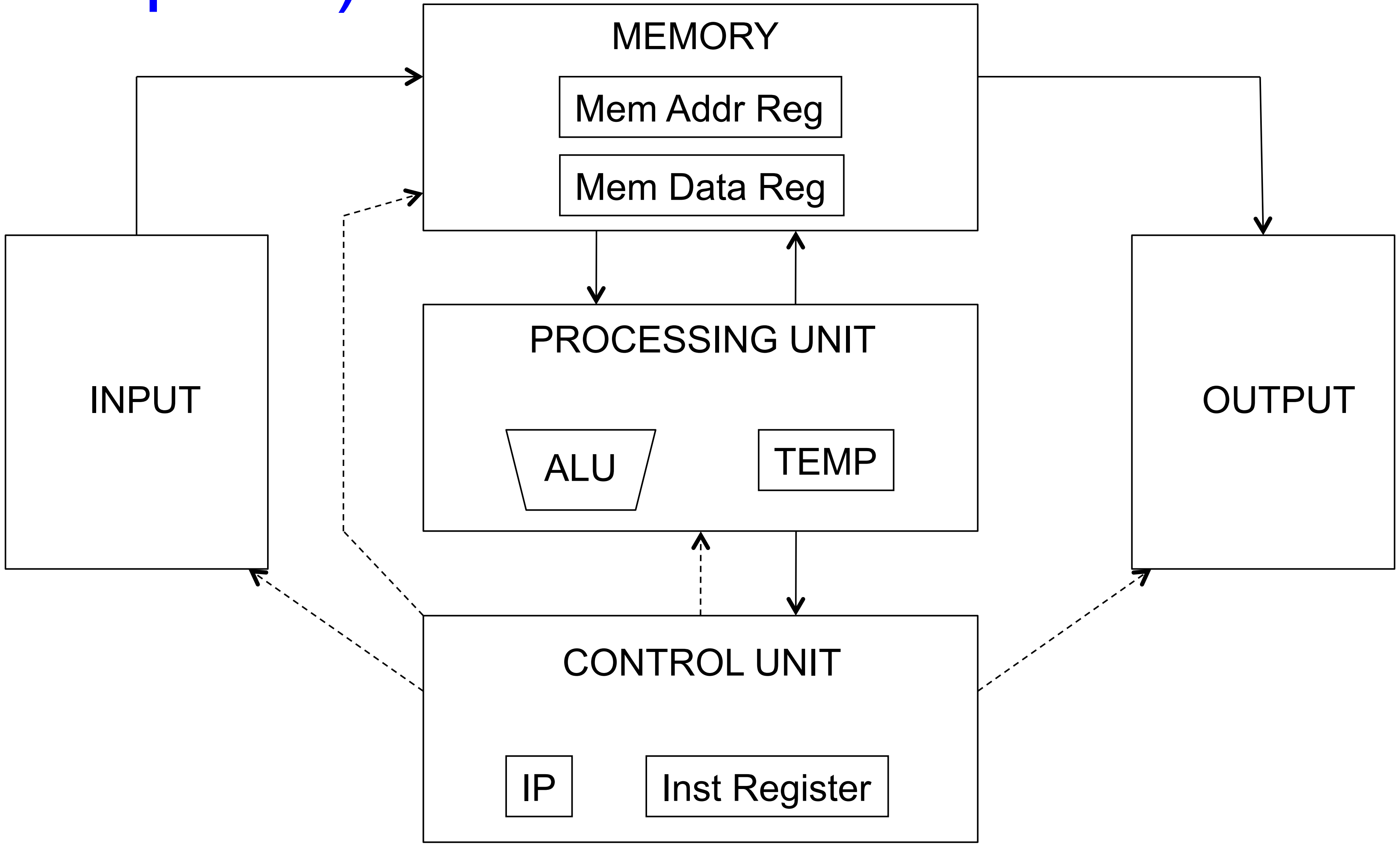
word = 32 bits

double word = 64 bits

Manipulating numbers on circuits



The Von-Neumann Model (of a Computer)



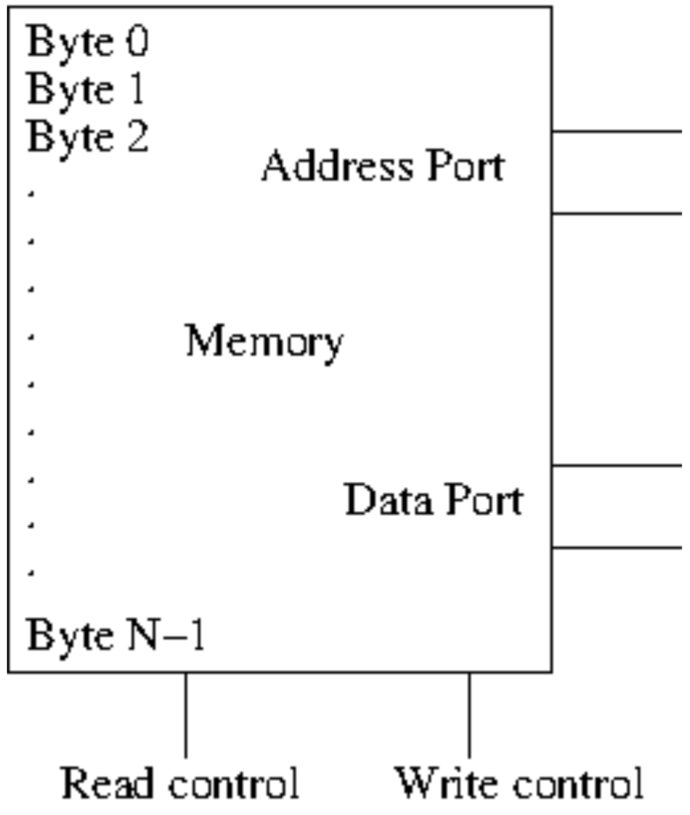
Memory

- Organized into bytes (groups of eight ***D latches***)
- Memories of present day computers contain few Gigabytes, Giga=2^{30}
- Each byte in the memory is assigned a distinct number, or an address.
- In a memory with N bytes, the addresses go from 0 to N-1

0	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**
1								
2								
3								
4								
5	1	0	1	1	1	1	0	1
6								
7								
8								
9								

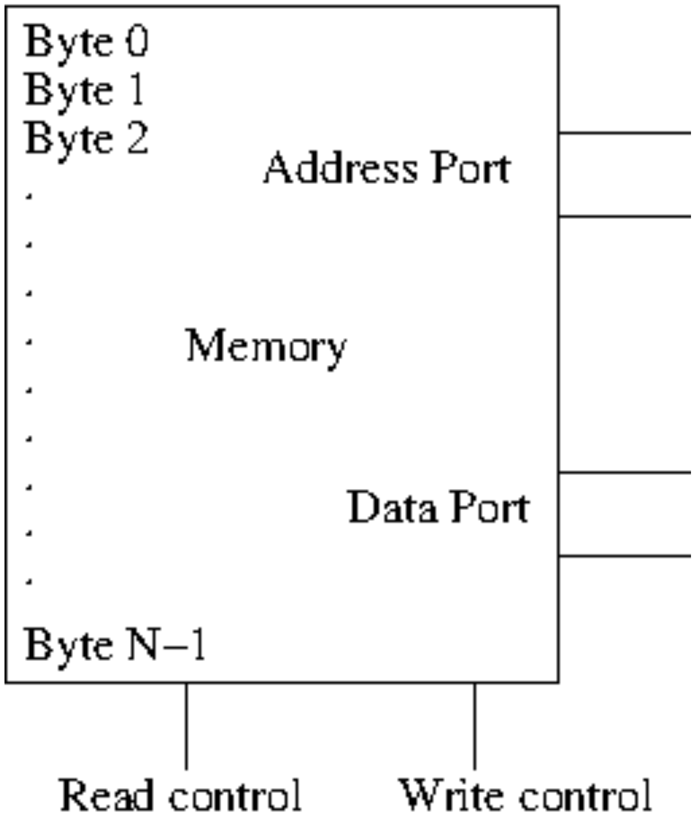
Memory Addressing

- Memory has 3 ports: address port, data port, control port.
- **Address port** consists of log N wires. (N = number of bytes in memory)
  - You can place numbers 0..N-1 on address port.
- **Data port** will have w wires, where w is a multiple of 8. Say w=8m.
- **Control Port** may be just 1 wire.
  - Wire = 0: Memory to perform read operation.
  - Wire = 1: Memory to perform write operation.

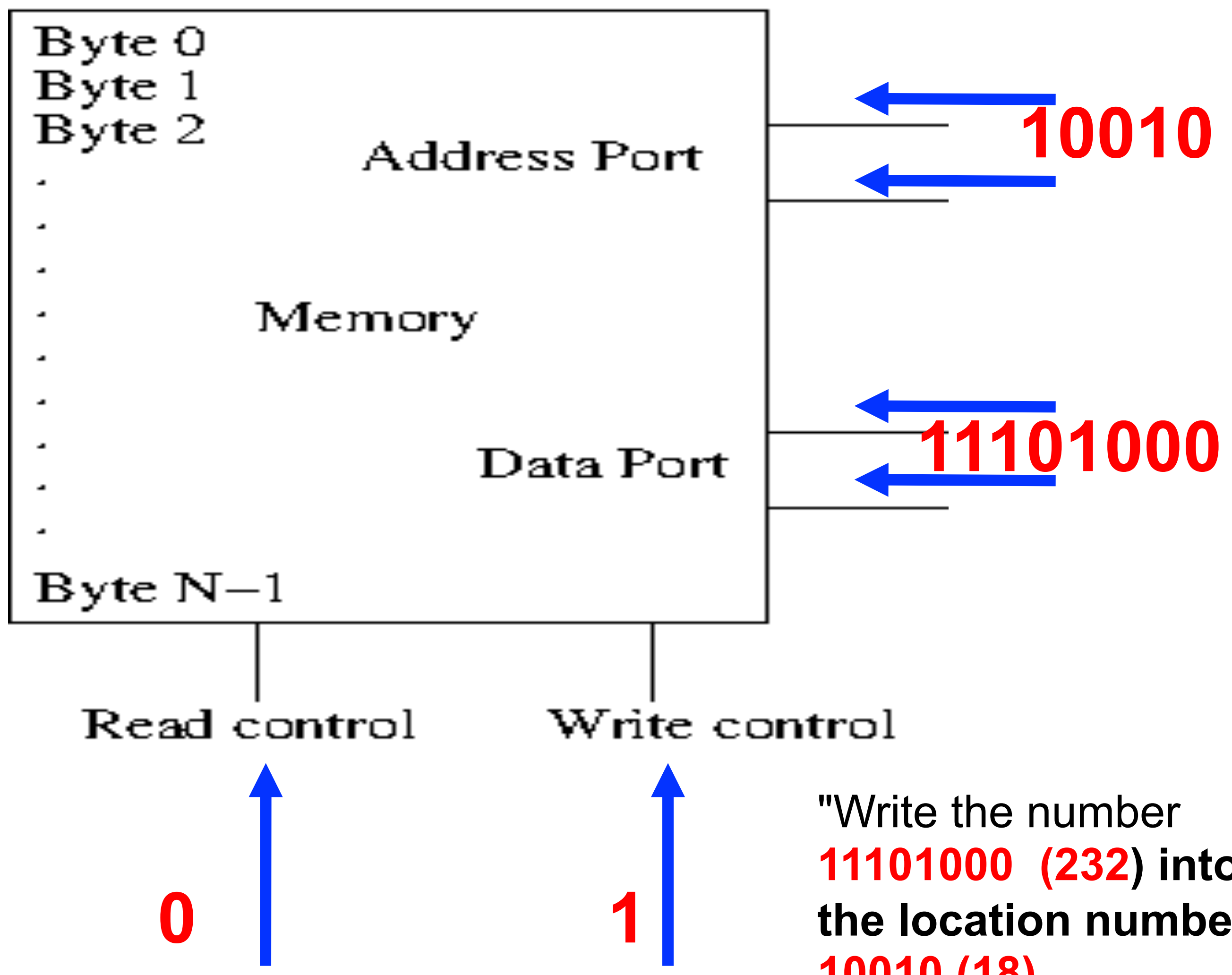


Write Operation

- Control Port must be set to 1.
- If **A** is placed on the address port, and **D** on data port, then **D** will be stored in the **m** bytes starting at byte **A**.
- (Remember that the data port had 8m wires, and so m bytes are available on the data port)
- Yes, it is possible to build circuits that can do this!



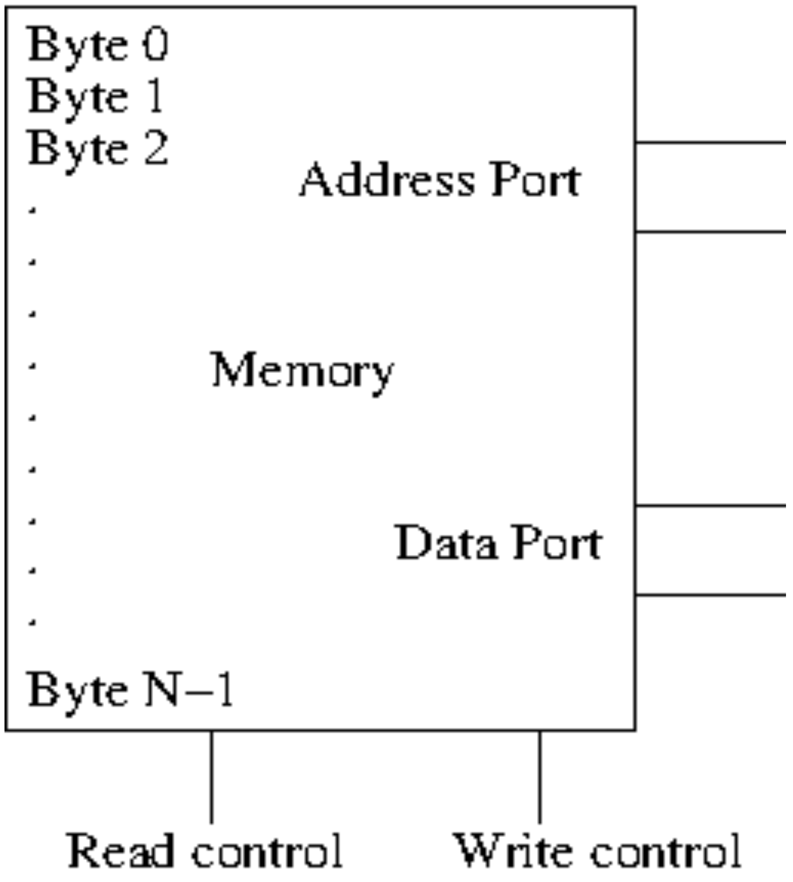
Write Operation



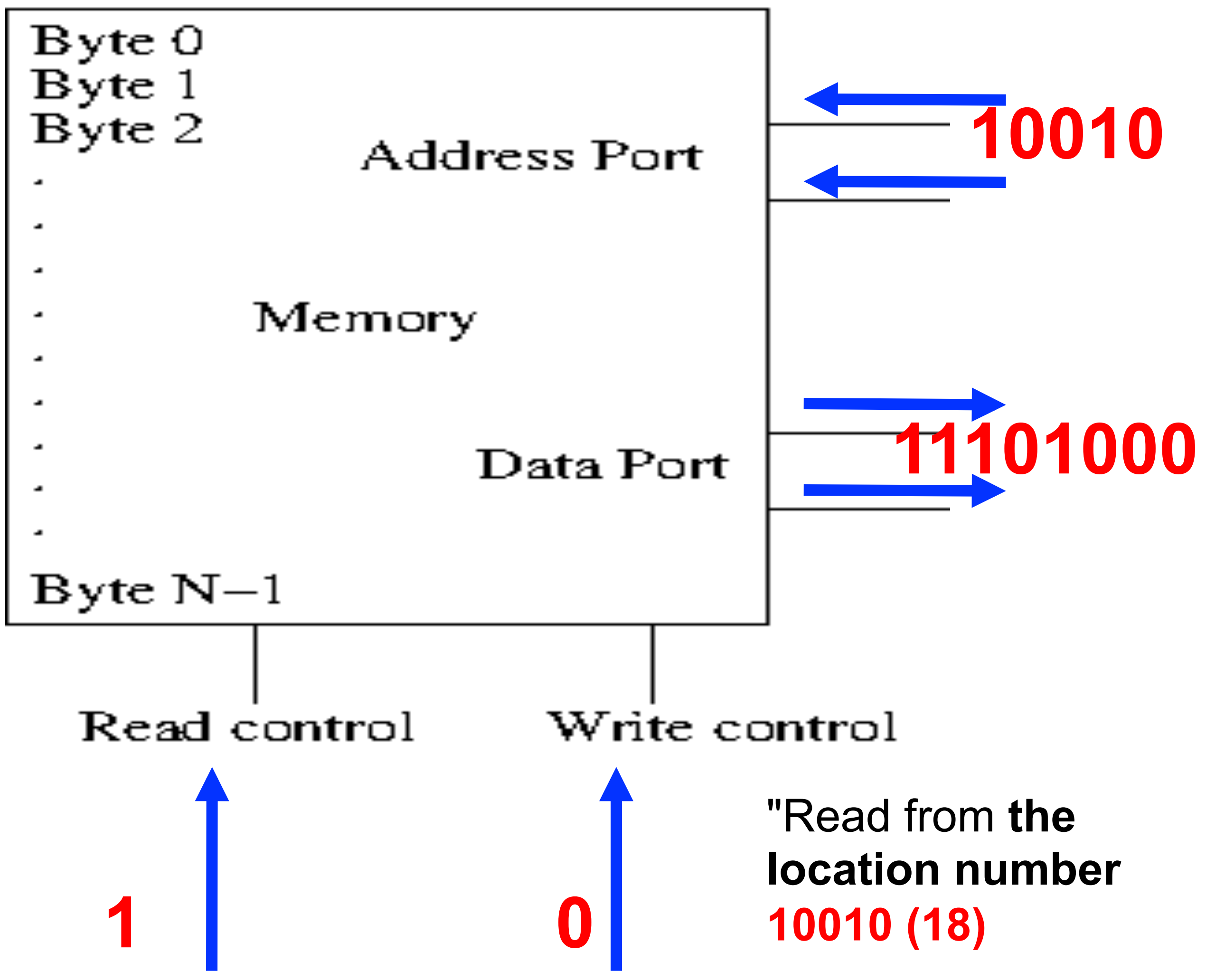
"Write the number **11101000 (232) into the location number 10010 (18)**

Read Operation

- Control Port must be set to 0.
- If A is placed on the address port, the m bytes starting at byte A will appear on the data port
- (Data port has 8m wires, and so m bytes will fit on the data port)

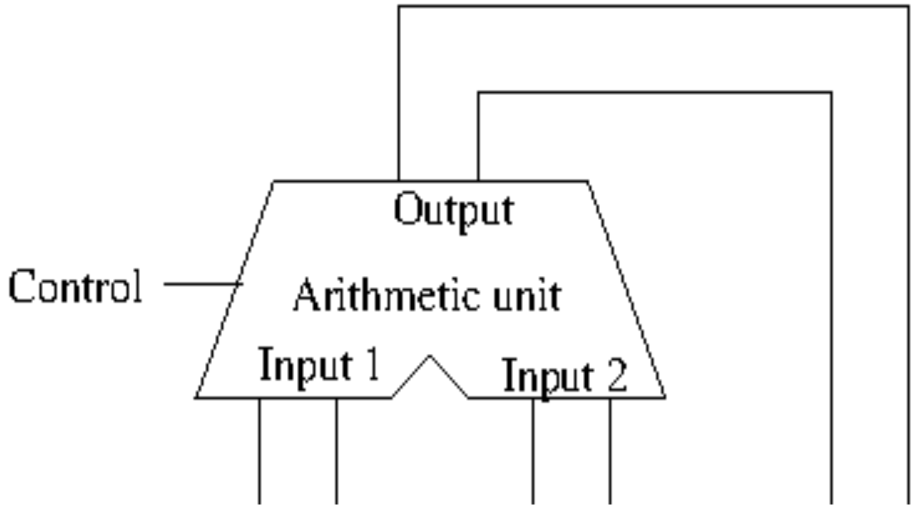


Read Operation



"Read from **the location number** 10010 (18)

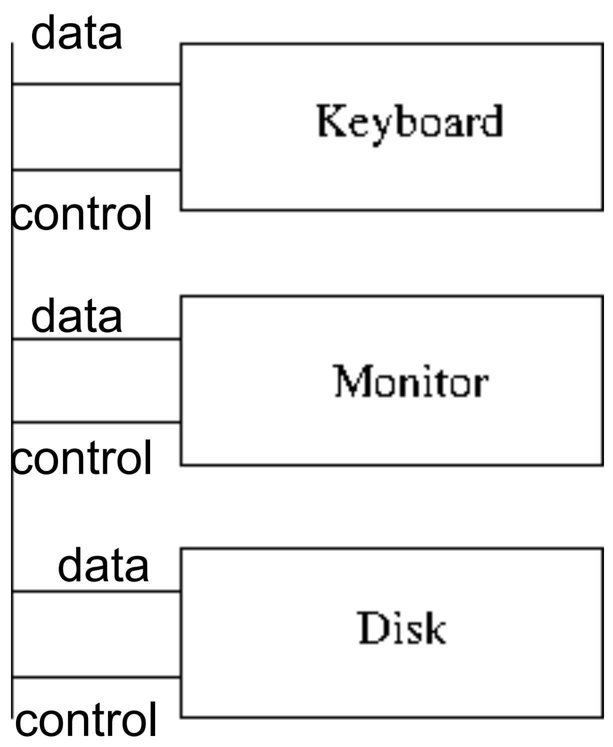
Arithmetic Logical Unit (ALU)



- Ports: Input1, Input2, Output, Control
- Typically Input1, Input2, Output will consist of w wires, w = size of memory data port
- Control = several wires. Number appearing on the control wires will say what operation should be performed.
- 1 cycle after values are placed on Control, the Output will take the commanded value: sum, product, …

Peripherals: keyboard, screen, disk…

- Also have control port and data port like organization.
- Depending upon value placed on control port, the peripheral decides what to do with the value on the data port, or itself places values on the data port.



Control Unit

- Tells other parts of the computer what to do.
  - Sends control signals (0 or 1) on control wires of each unit

- The control unit decides what to tell other units by reading a "machine language program" from the memory of the computer.

Machine language progam

- Program = Sequence of instructions
- Instruction = sequence of numbers
  - First number is OPERATION CODE (OPCODE). This is the code that tells the Control Unit what OPERATION should do.
  - Subsequent numbers are OPERANDS. These are "arguments" to the operation.

OPCODE	**OPERAND1**	**OPERAND2**	**OPERAND3**

Example – Op Code 57

57	**100**	**200**	**300**

- Interpret the 3 numbers following 57 as addresses.
- Read the words at the first two addresses and send them to the Arithmetic unit.
- Command the arithmetic unit to perform ***multiplication*** by sending appropriate number on its control wires.
- Store the result from the arithmetic unit into the ***word*** at the third address

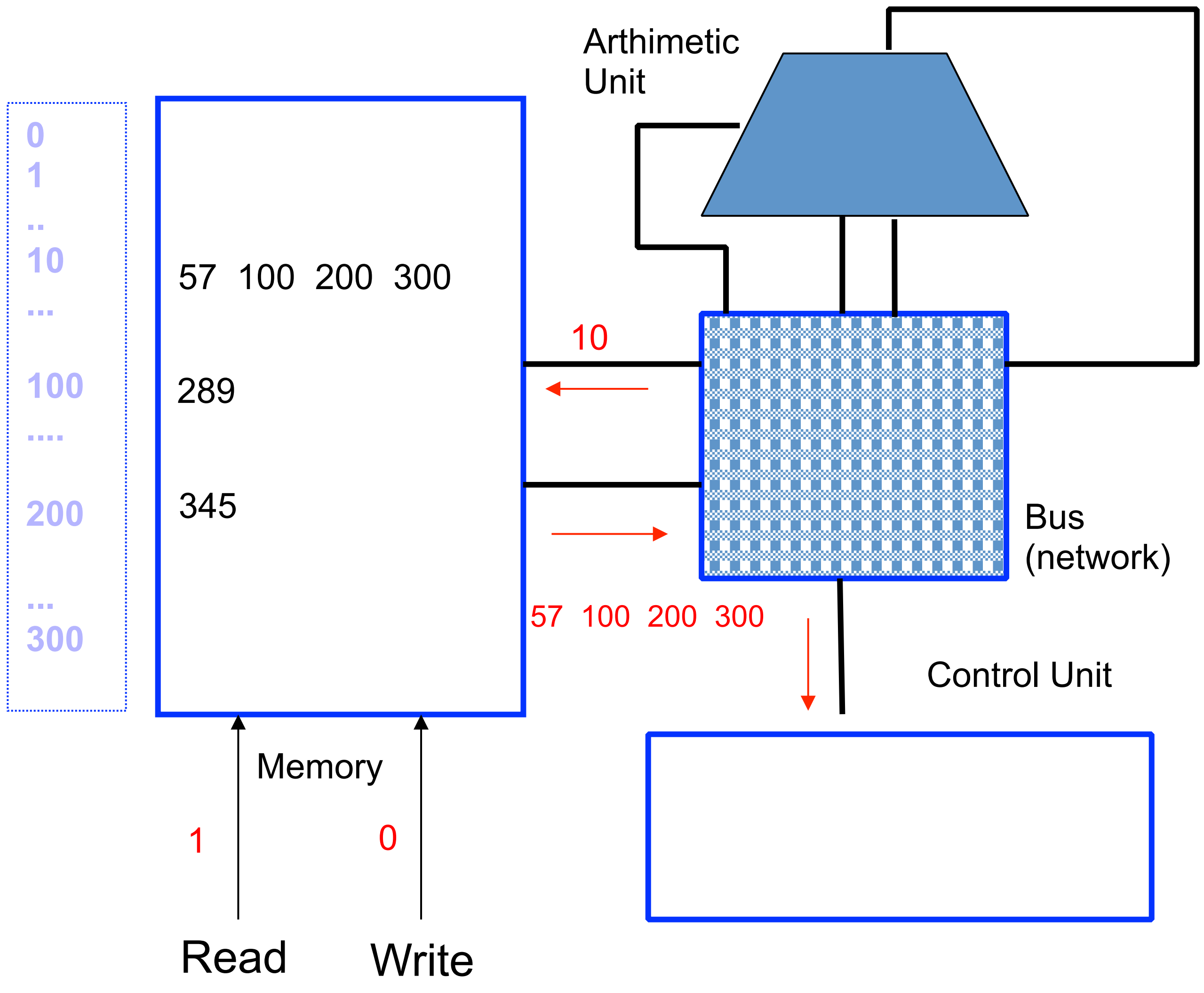
Machine language progam

- Program = Sequence of instructions
- Instruction = sequence of numbers, first of which is an "operation code" which denotes what operation to perform
- Example: operation code 57 might mean:
  - Interpret the 3 numbers following 57 as addresses.
  - Read the words at the first two addresses and send them to the Arithmetic unit.
  - Command the arithmetic unit to perform multiplication by sending appropriate number on its control wires.
  - Store the result from the arithmetic unit into the word at the third address
- The sequence 57, 100, 200, 300 is an instruction that would cause the product of the numbers stored in addresses 100, 200 to be stored in the address 300.
- The operation codes are defined by the computer designer.
  - She will assign a different code for each operation that she would like the computer to perform.
  - Example: 58 might mean the same thing as above, except that the numbers would be added.

Control unit operation

- Control unit must be told where the machine language program is in memory.
- The control unit then fetches the instructions constituting the program, interprets the codes, and performs the required operation.
- After one instruction is fetched and executed, it fetches the next instruction and repeats the process.

Putting it together

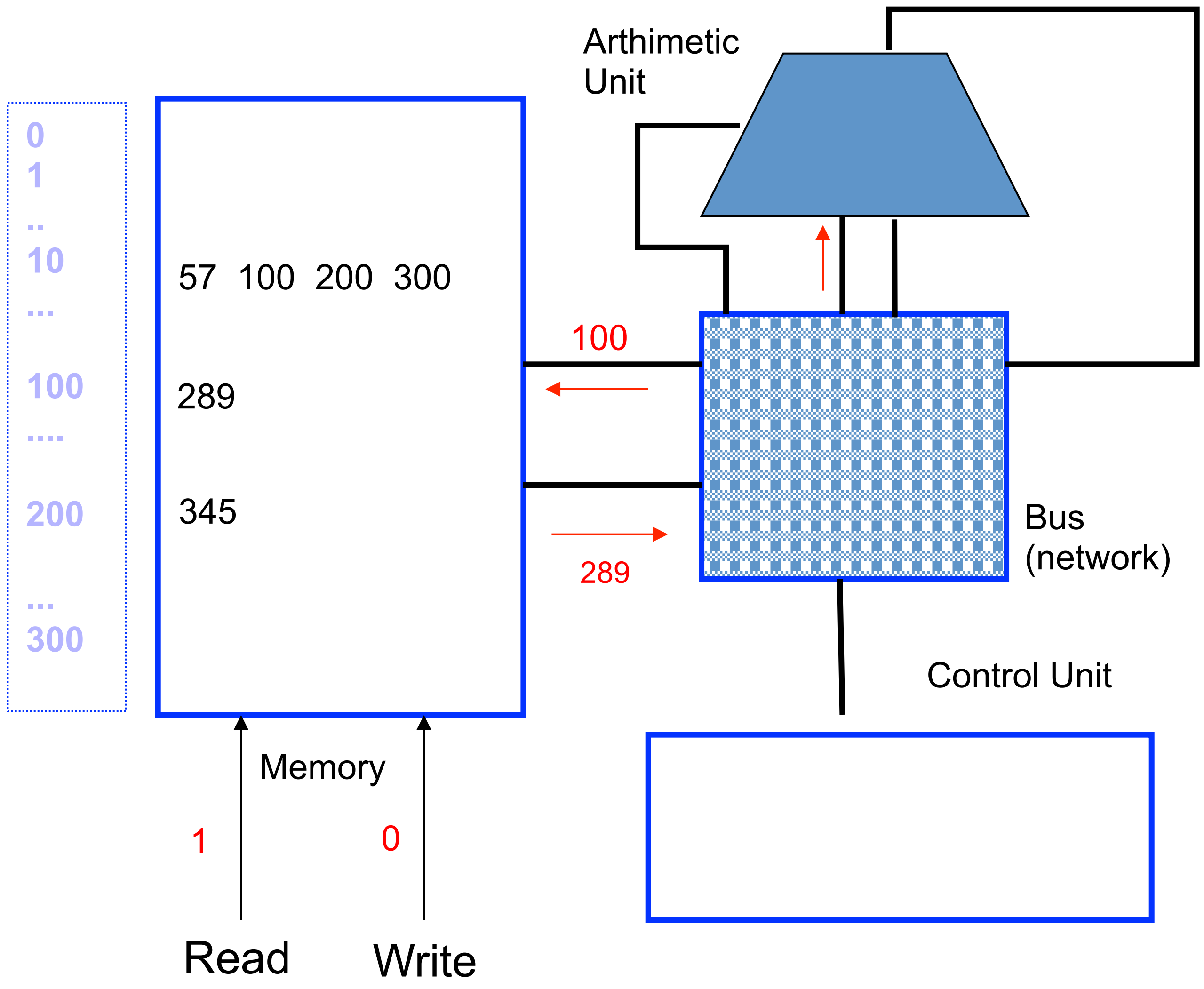


1. Control unit is told the address of the instruction to fetch (e.g. instruction is at location 10)

2. A read operation is performed on memory location 10

3. instruction 57 100 200 300 is now "loaded" into the control unit

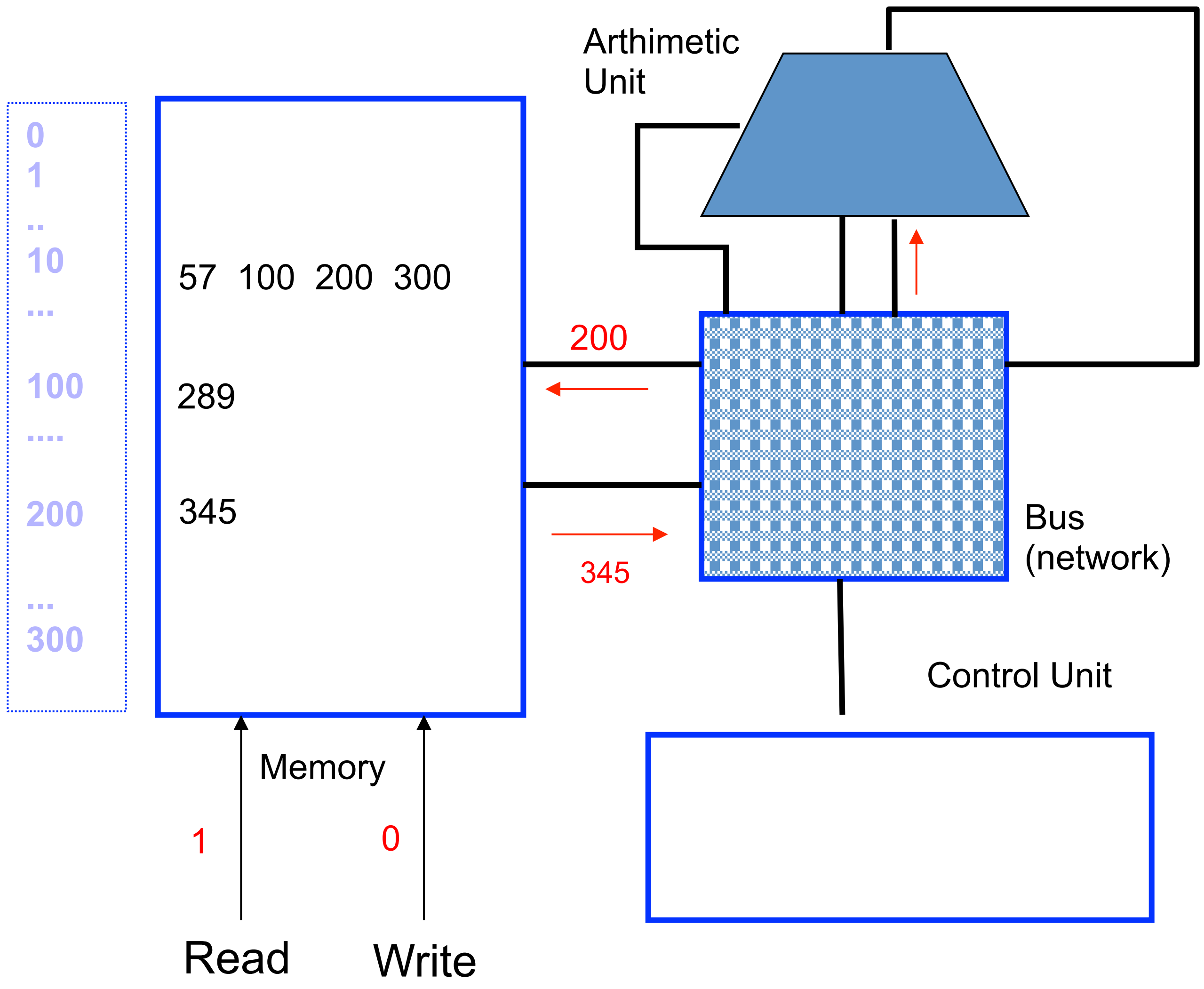
Putting it together



1. Now control unit performs a read operation of address 100

2. The number 289 is sent to input 1 of arithmetic unit

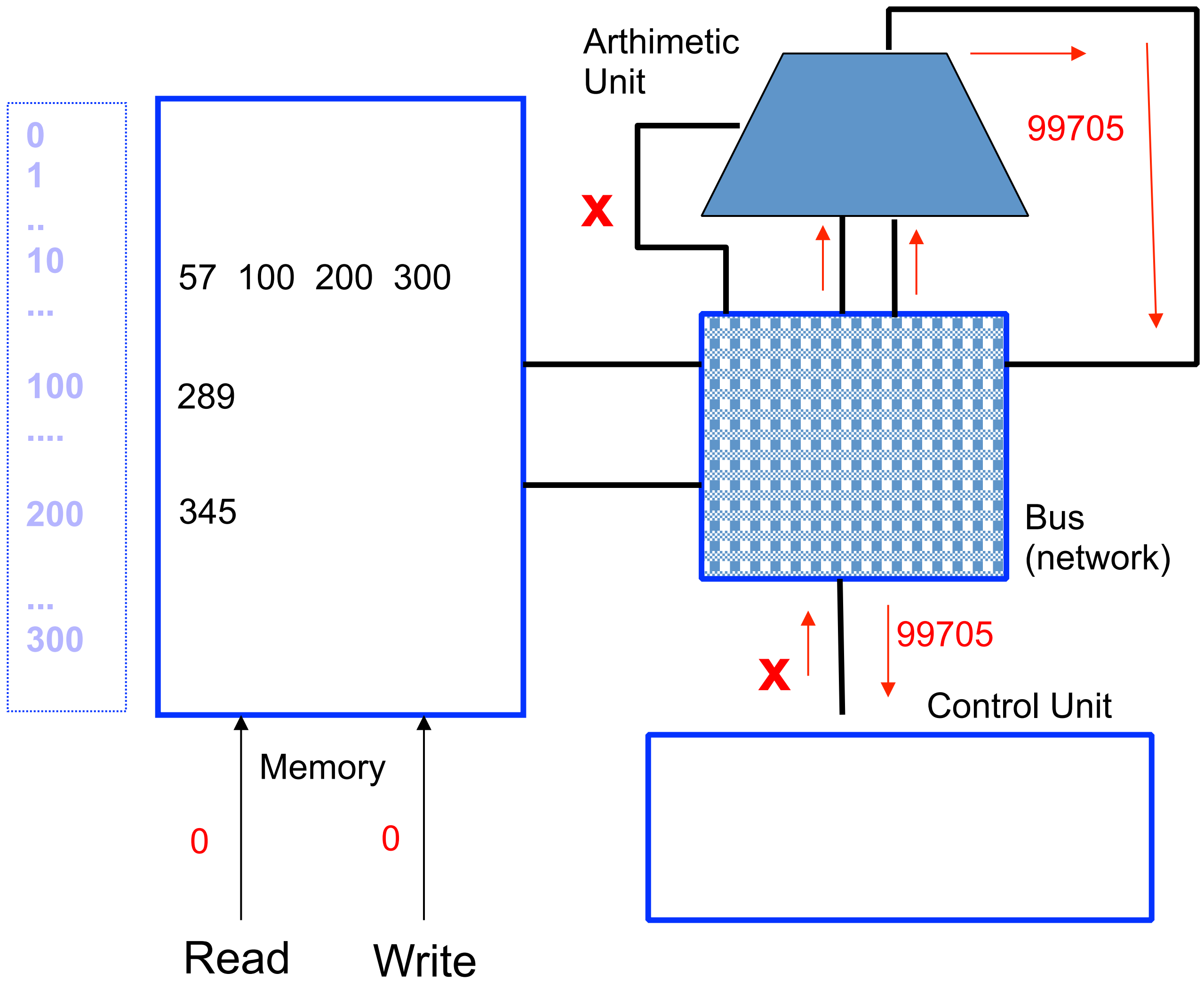
Putting it together



1. Now control unit performs a read operation of address 200

2. The number 345 is sent to input 2 of arithmetic unit

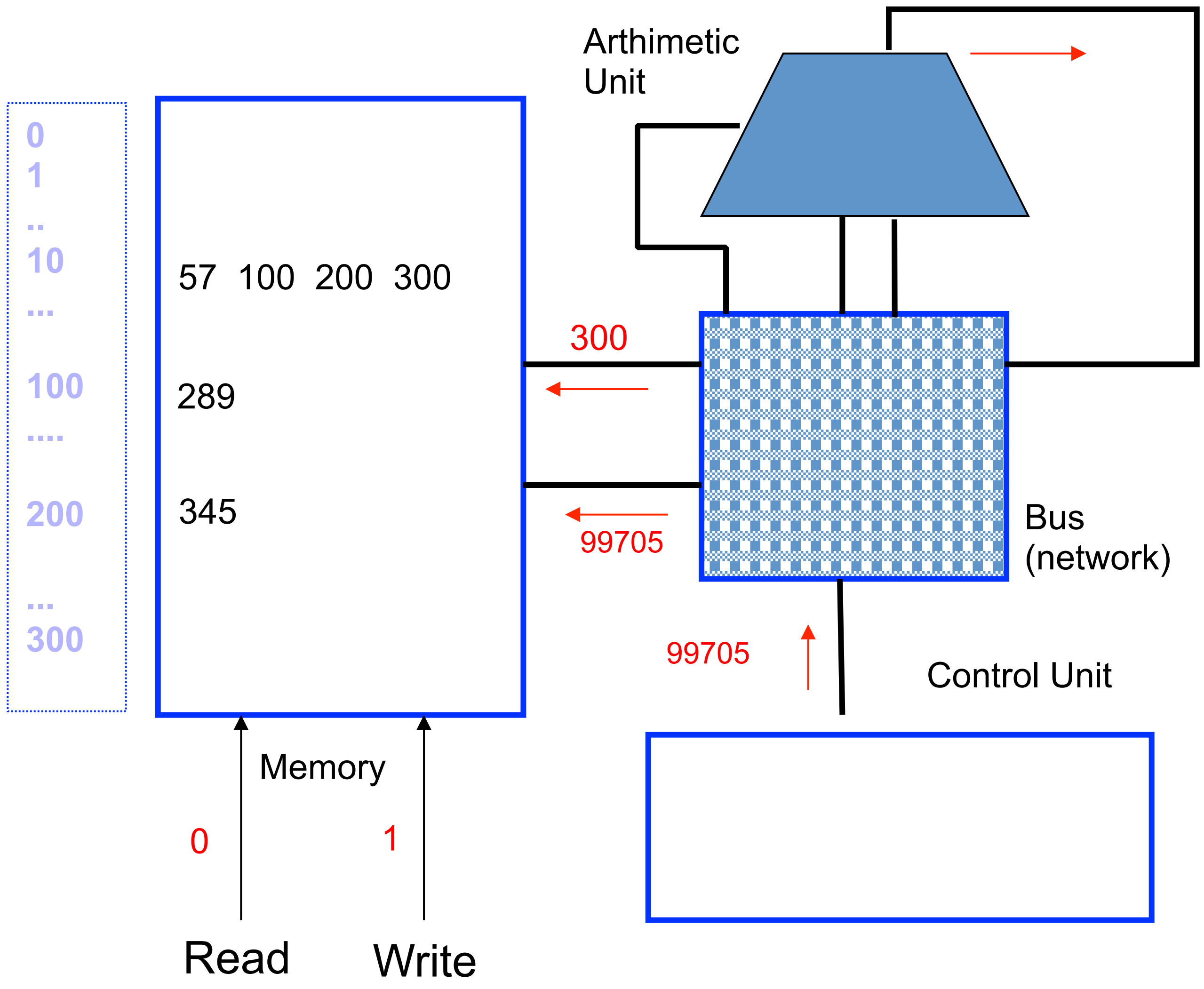
Putting it together



1. Now control unit instructs the arithmetic unit to perform a multiply operation (the "control" wires are set to the operation code of "multiply")

2. Control unit stores the product (289 x 345 = 99705) temporarily inside its own unit

Putting it together

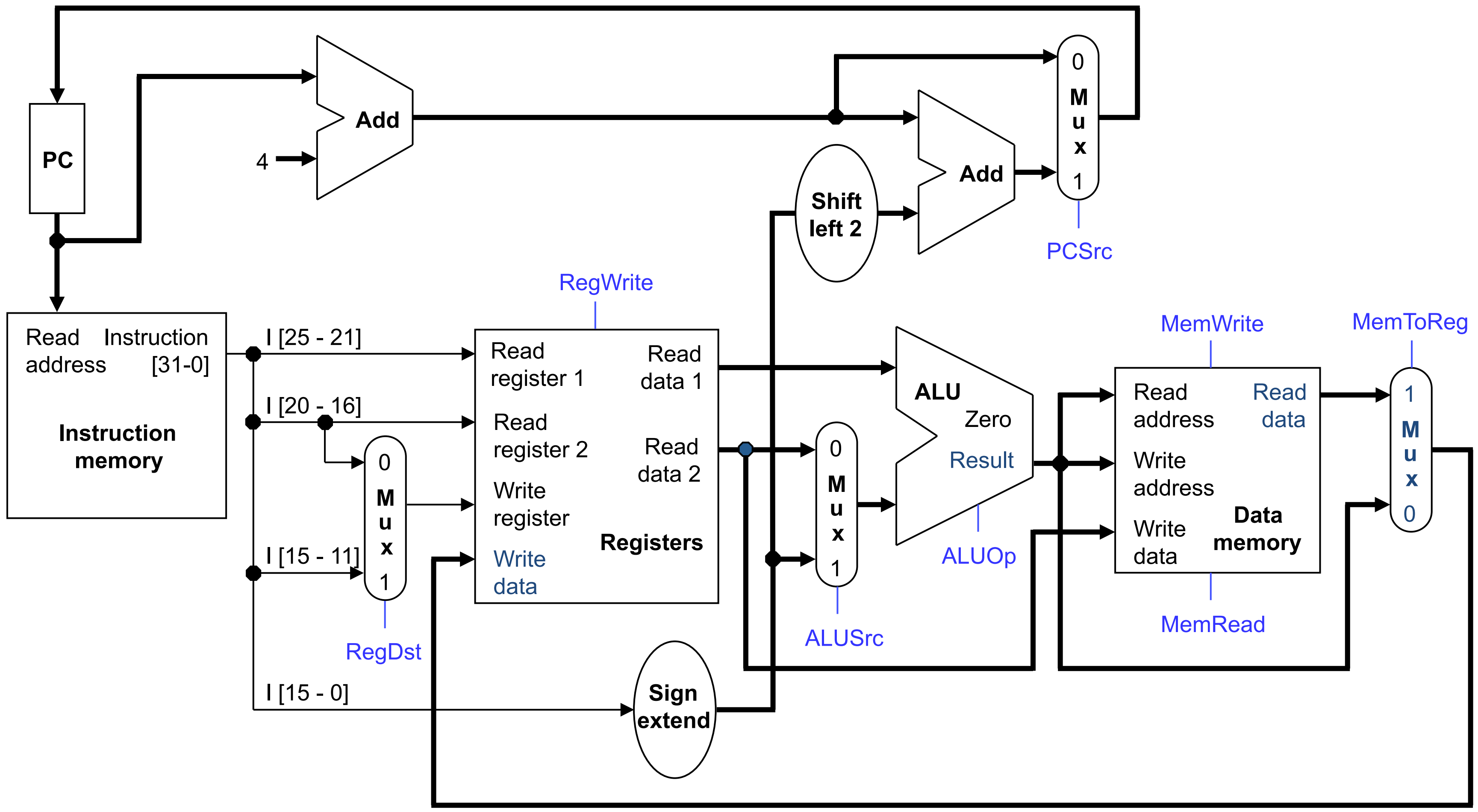


1. Now control unit performs a write operation on address 300

2. The number 99705 is sent on the data port of memory and 300 is sent on the address port of the memory

Instruction execution is COMPLETE

A Real Processor



A single-cycle MIPS processor

Machine language ***program*** example

Example:

57, 100, 100, 100

57, 100, 100, 100

This contains two instructions.

Both instructions cause the word at address 100 to be multiplied by itself and the result stored back in address 100.

After executing the first instruction, address 100 would contain the square of the number that was present before.

The second operation would repeat the squaring operation.

Thus this is a machine language program to compute the fourth power of a number.

More complex instructions

Example 1: ***operation code 59*** might mean: “Shut down the computer”

Example 2: **operation code 60** might mean:

- Interpret the next number in the program as an address
- Instead of next executing the instruction following the current one, execute the instruction starting at this address.

Example 3: **operation code 61** might mean:

- Interpret the next number in the program as an address.
- If the last result produced by the arithmetic unit was 0, then execute the instruction starting at this address.
- If the last result produced by the arithmetic unit was not 0, then execute the instruction following the current one.
- Analogous to the repeat statement of chapter 1.
- Using such instructions, we will be able to perform an operation 100s of times without making our machine language very long.

Machine language programs and C++ programs

On early computers, you would have to write machine language programs.

- Find out what operation you want.
- Look up the manual and find its code.
- Enter the code into the memory of the computer.
- Repeat.
- Process is laborious, error-prone.

Modern computers also need machine language programs.

- You write C++ program.
- A prewritten program, “compiler”, translates your C++ program to a machine language program.
- Another program, “loader”, will load it into the memory and start its execution.

Concluding Remarks

- Key idea 1: use numerical codes to represent non numerical entities
  - letters and other symbols: ASCII code
  - operations to perform on the computer: Operation codes
- Key idea 2: Current/charge/voltage values in the computer circuits represent bits (0 or 1).

Concluding Remarks

- Memory is organized into bytes. Each byte has an address.
- What the computer does is determined by a machine language program that must be stored in the memory.
- Computer users do not need to themselves write a machine language program, but can write a C++ program which is then compiled by a compiler.