CMSC 11:
Introduction to Computer Science

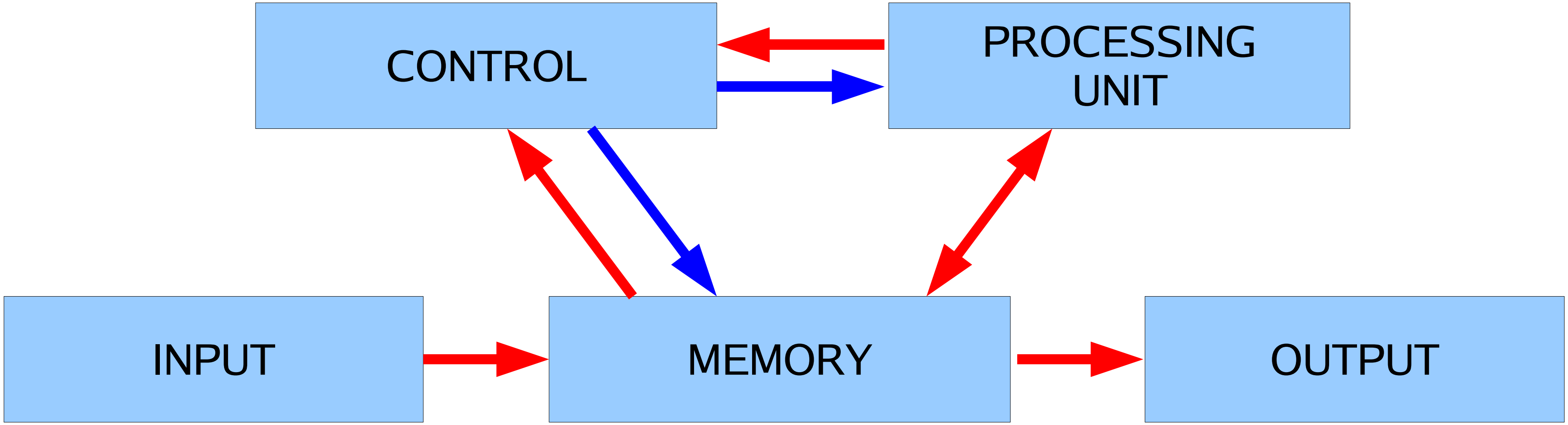
Jaderick P. Pabico <jppabico@uplb.edu.ph>
Institute of Computer Science, CAS, UPLB

Review

- RAM – volatile
- ROM - unerasable
- Program – computer's control
- Software – hardest thing about computer
- Turing Machine – logic brought to life
- Algorithm – well defined step-by-step procedure

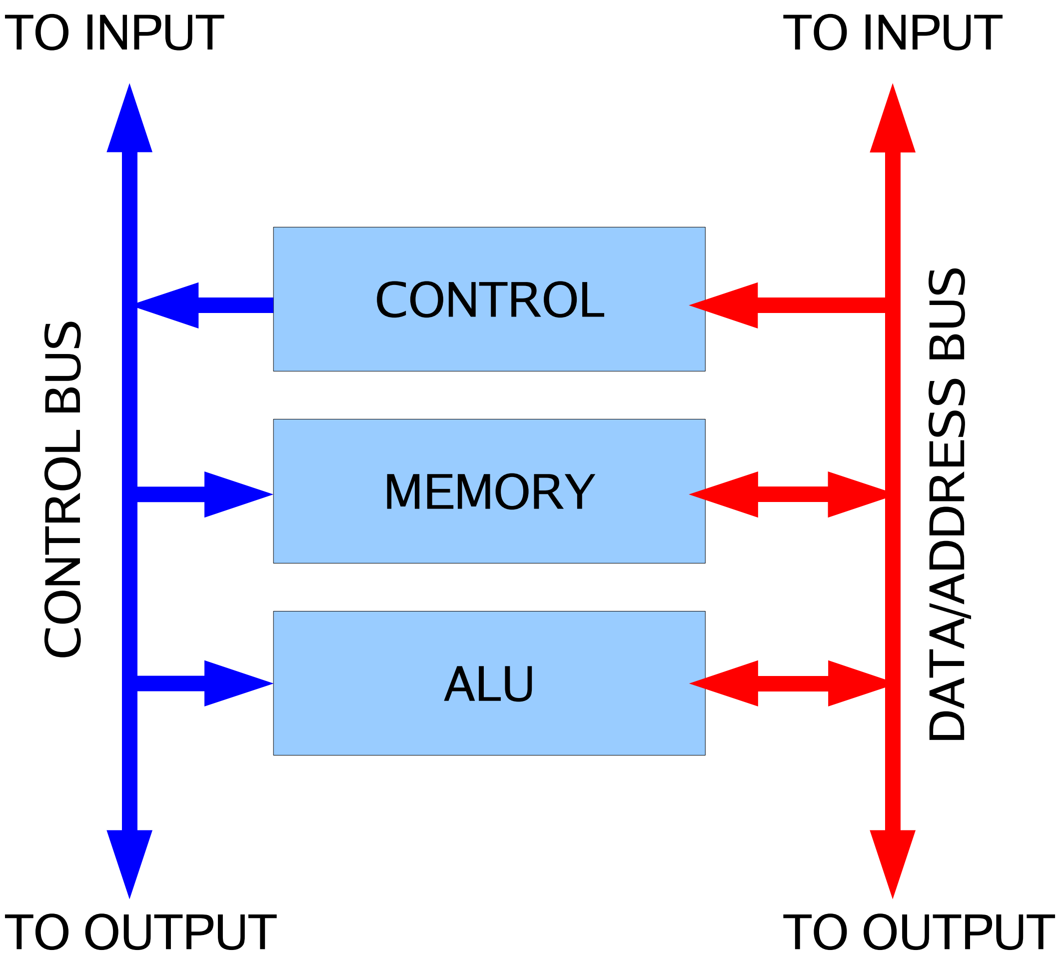
CONTROL

- Getting everything under control
- Along with I/O, Memory, and the ALU, control is the computer's final, critical ingredient
- Our old schematic diagram shows the flow of control and information.



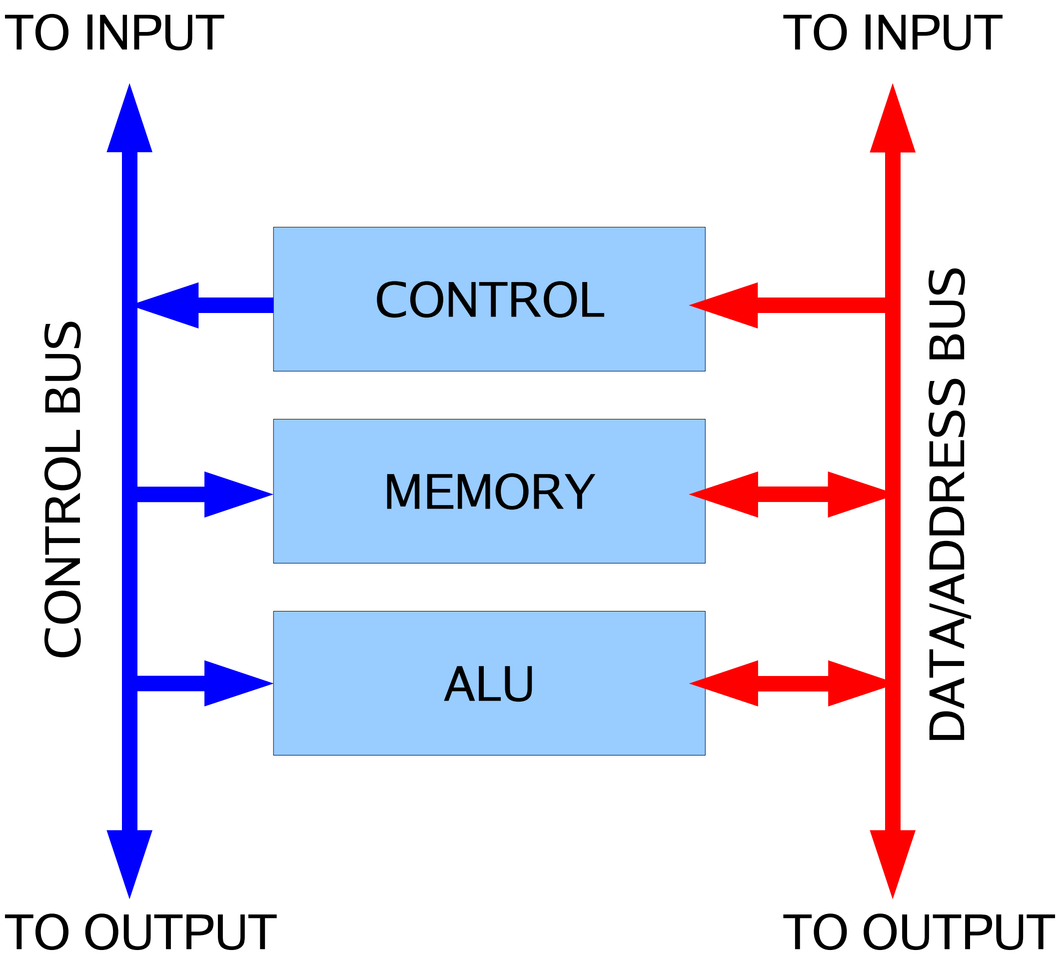
CONTROL

- Let's redraw the diagram in a way that better reflects a genuine computer design: **The Bus Architecture**



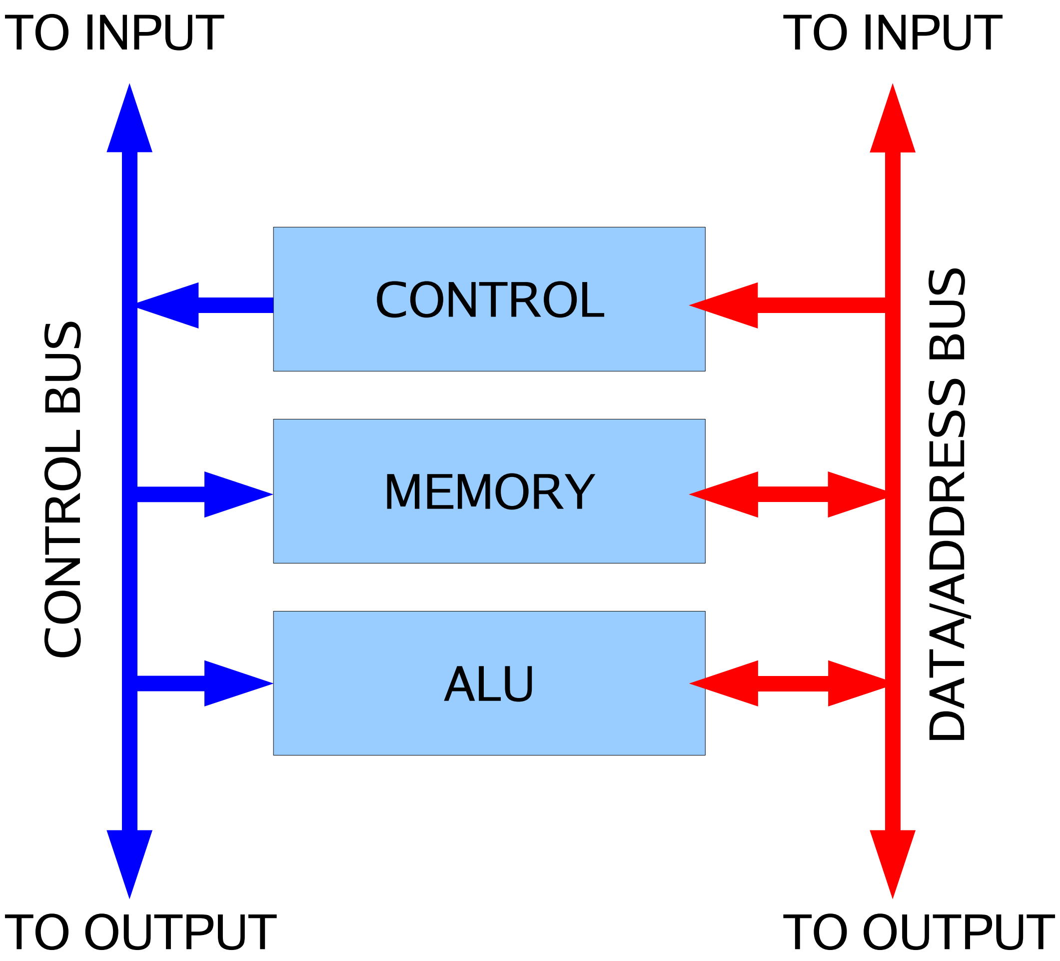
CONTROL

- The vertical arrows, representing electrical pathways a byte or more wide, are the **buses**.



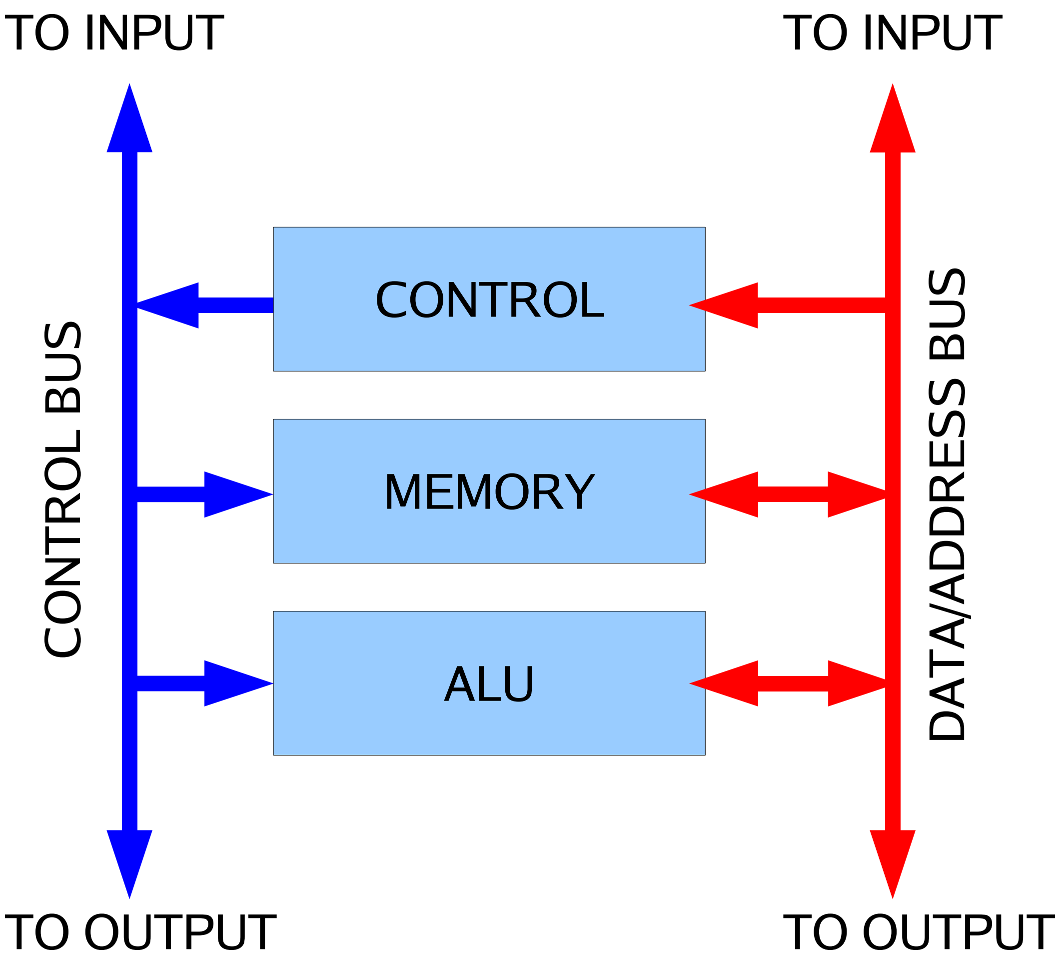
CONTROL

- According to signals passed along the control bus, addresses and data, get on and off the data/address bus...



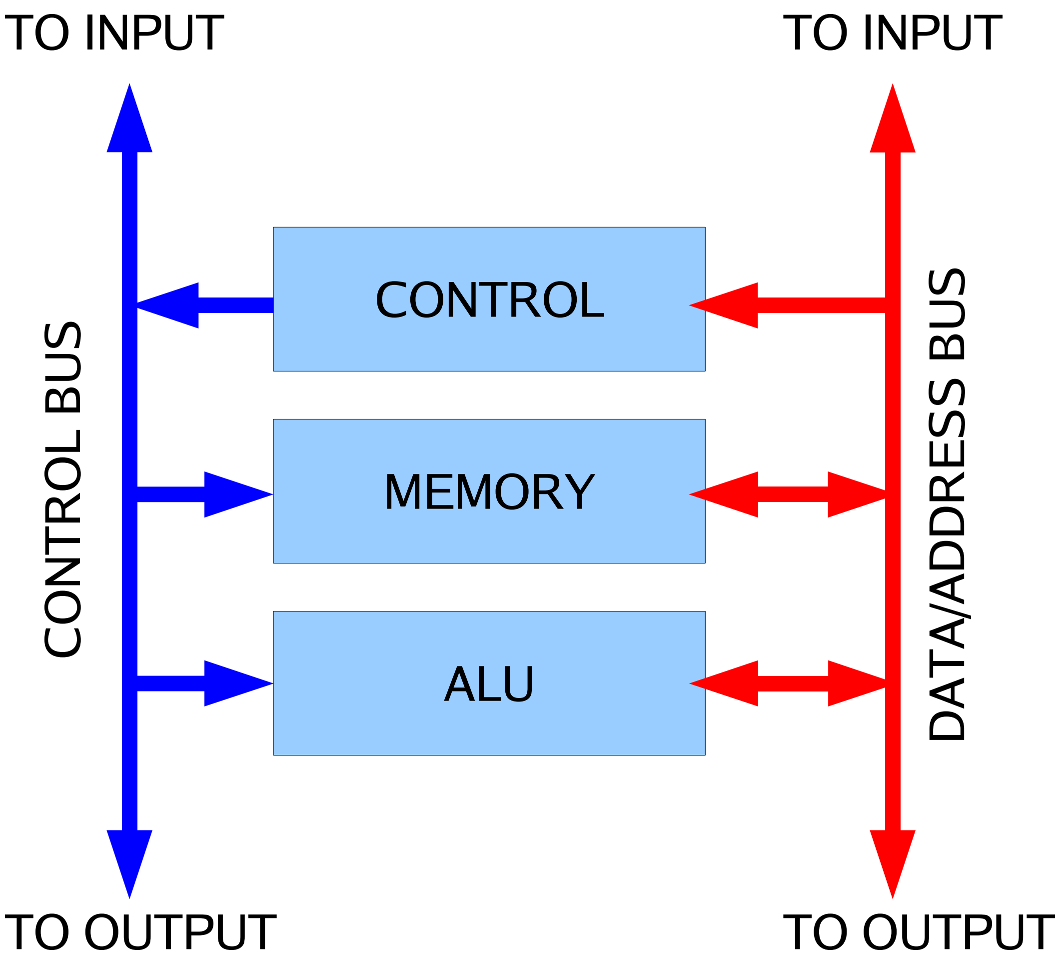
CONTROL

- … with the only prerequisite that only one “passenger” can ride the bus at a time.



CONTROL

- Notice that all the arrows of the control bus point away from the control section.



CONTROL

- How are we to imagine this control, from which all blue arrows point away?

As a megalomaniacal robot that can't keep its electronic fingers out of anything?

I-MUST-MAINTAIN-CONTROL-AT-ALL-COST

CONTROL

- How are we to imagine this control, from which all blue arrows point away?

As a wise ruler who judiciously chooses the time for every act?

Go ye... and multiply!

CONTROL

- How are we to imagine this control, from which all blue arrows point away?

As a relentless tyrant who wields a whip hand over revellious glitches?

Well, at least the buses run on time

CONTROL

- Like anyone else, control reveals its character by its behavior.
- SO let's follow what happens in this oversimplified diagram

CONTROL

- This is a minimal collection of equipment
- A typical computer has more registers and counters
- But all computers have the ones shown here.

Here's what they're for:

- Program counter ticks off the instructions one by one

One-two, one-two...

Here's what they're for:

- Instruction register holds an encoded version of the instruction being formed

Boil spaghetti ten minutes

Here's what they're for:

- Address register holds the address of whatever is to enter or leave memory

Get me Byte
#0101!

Here's what they're for:

- Accumulator, the ALU's main register, keeps a running total of all ALU operations

Couldn't solve it without it!

Here's what they're for:

- B Register is an auxillary register to hold numbers on their way to ALU.

Like a motel that rents room by the microsecond

Here's what they're for:

- C Register holds data on their way to output.

Is their control in
the outside
world?

Here's what they're for:

- Control spends most of its time just moving the contents of these registers around.
- To see how control works, let's follow what happens when the computer ADDS TWO NUMBERS
- This will be our very first program
- Like everything about computers, programs can be described at various levels.
- We begin with...

Assembly Language

- Specifies the computer's actual moves
- BUT omits the fine details.
- AT this level, here is how to add two numbers:

STEP 0:
LOAD THE FIRST NUMBER
INTO THE ACCUMULATOR

Assembly Language

STEP 0:
LOAD THE FIRST NUMBER
INTO THE ACCUMULATOR

STEP 1:
ADD THE SECOND NUMBER,
HOLDING THE SUM IN THE ACCUMULATOR

Assembly Language

STEP 0:
LOAD THE FIRST NUMBER
INTO THE ACCUMULATOR

STEP 1:
ADD THE SECOND NUMBER,
HOLDING THE SUM IN THE ACCUMULATOR

STEP 2:
OUTPUT THE CONTENTS
OF THE ACCUMULATOR

Assembly Language

STEP 0:
LOAD THE FIRST NUMBER
INTO THE ACCUMULATOR

STEP 1:
ADD THE SECOND NUMBER,
HOLDING THE SUM IN THE ACCUMULATOR

STEP 2:
OUTPUT THE CONTENTS
OF THE ACCUMULATOR

STEP 3:
HALT!

Assembly Language

- To express this in proper assembly language,
- We must specify the precise location in memory of the two numbers to be added
- And condense the wordy statements into mnemonic abbreviations
- Suppose for example, that the numbers are stored at addresses **1E** and **1F** (hexadecimal).

Assembly Language

- LDA 1E
- ADD 1F
- OUT
- HALT

Load accumulator
with contents of 1E

Add contents of 1F

Output contents
of accumulator

Assembly Language

- In general, assembly-language statements have two parts:

The OPERATOR which describes the step to be performed

The OPERAND which gives the address on which the operator acts

LDA 1E

Assembly Language

- Note, however, that some operators don't need an explicit operand
- OUT, for example, is understood to apply to the accumulator

OPERATOR

NO OPERAND required

OUT