CMPT 295

Unit – Microprocessor Design & Instruction Execution

Lecture 26 – Intro to Logic Design

Last Lecture

- ISA design
  - **MIPS**
  - Created our own **x295M**: “Memory only”
- ISA Evaluation
  - Examining the effect of the **von Neumann bottleneck** on the execution time of our program by counting **number of memory accesses**
    - The fewer memory accesses our program makes, the faster it executes, hence the “better” it is
- Improvements:
  - Decreasing effect of **von Neumann bottleneck** by reducing the **number of memory accesses**

Memory Traffic

And one way to achieve this is by introducing registers in the design of the ISA -> makes machine instructions shorter. Reducing the number of operands may also help (but not always).

Today's Menu

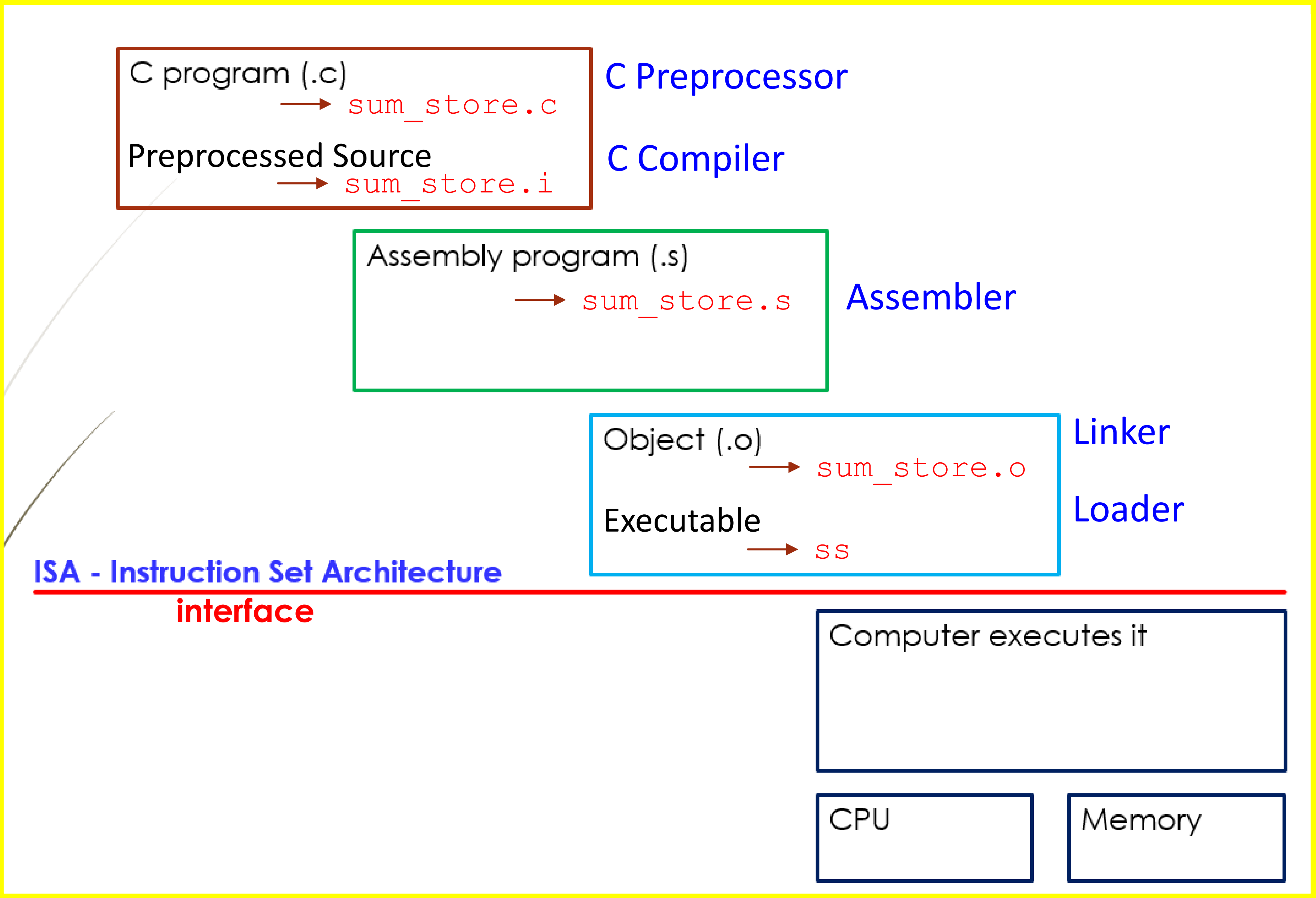
- Instruction Set Architecture (ISA)
  - Definition of ISA
- Instruction Set design
  - Design principles
  - Look at an example of an instruction set: MIPS
  - Create our own
  - ISA evaluation
- Implementation of a microprocessor (CPU) based on an ISA
  - Execution of machine instructions (datapath)
  - Intro to logic design + Combinational logic + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions + Hazards

The Big Picture

Now that we have had a look at a few instruction set architectures (ISA), i.e.,

- Specification of various models (memory model, computational model, operand model, etc …), and …
- Design of instruction set (or subset) (assembly instructions and their corresponding machine instructions, their formats, etc.) …

... let's step over this interface and explore how the microprocessor is constructed so it can execute these machine instructions



C program (.c) → sum_store.c

Preprocessed Source → sum_store.i

C Preprocessor

C Compiler

Assembly program (.s) → sum_store.s

Assembler

Object (.o) → sum_store.o

Executable → ss

Linker

Loader

ISA - Instruction Set Architecture

interface

Computer executes it

CPU

Memory

Datapath of a MIPS microprocessor

Sneak preview (how CPU hardware updates the PC in fetch-decode-execute loop):
(from our Lecture 25)

- Consider the memory address 0x00400000 i.e.,
  `0000 0000 0100 0000 0000 0000 0000`
  which holds the **MIPS** machine instruction
  `100011 11101 10001 0000000000000000`
- PC contains 0x00400000 i.e.,
  32 line bus with signals 0x00400000 input into ADDER with "4" as other input
- ADDER adds both inputs and produces 0x00400004 i.e., memory address of next machine instruction which holds the **MIPS** machine instruction
  `100011 11101 10010 0000000000000100`
  PC overwriting 0x00400000

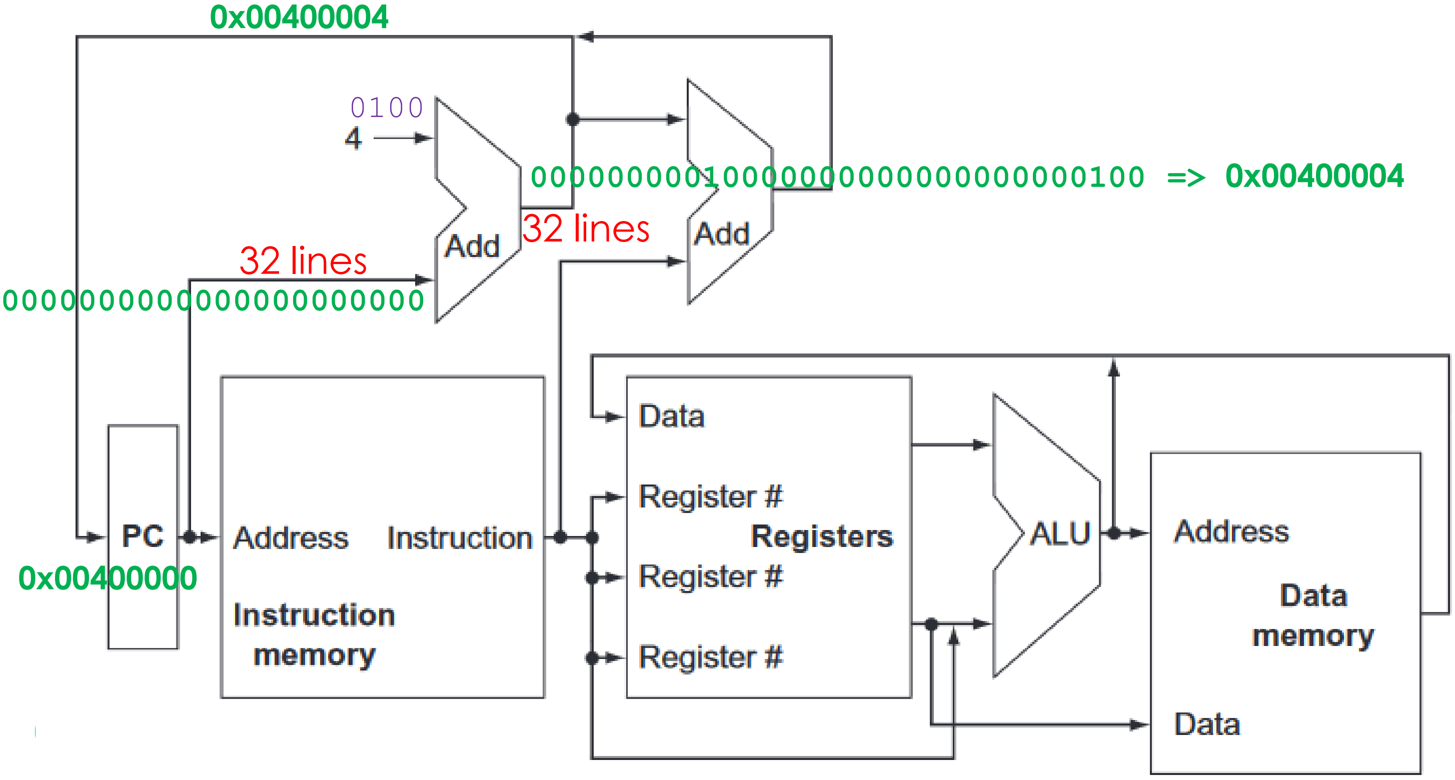


FIGURE 4.1 An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them. All instructions start by using the program counter to supply the instruction address to the instruction memory. After the instruction is fetched, the register operands used by an instruction are specified by fields of that instruction. Once the register operands have been fetched, they can be operated on to compute a memory address (for a load or store), to compute an arithmetic result (for an integer arithmetic-logical instruction), or a compare (for a branch). If the instruction is an arithmetic-logical instruction, the result from the ALU must be written to a register. If the operation is a load or store, the ALU result is used as an address to either store a value from the registers or load a value from memory into the registers. The result from the ALU or memory is written back into the register file. Branches require the use of the ALU output to determine the next instruction address, which comes either from the ALU (where the PC and branch offset are summed) or from an adder that increments the current PC by 4. The thick lines interconnecting the functional units represent buses, which consist of multiple signals. The arrows are used to guide the reader in knowing how information flows. Since signal lines may cross, we explicitly show when crossing lines are connected by the presence of a dot where the lines cross.

Digital circuits

- In order to understand how the microprocessor executes these machine instructions (series of 0's and 1's), we need to have a look at the components of a microprocessor and how they function:

  1. Combinational logic -> manipulate bits (compute functions on bits e.g., ADD)
  2. Memory elements -> store bits
  3. Clock signals -> regulate the update of memory elements

  and what affects the execution speed of these components such as propagation delay

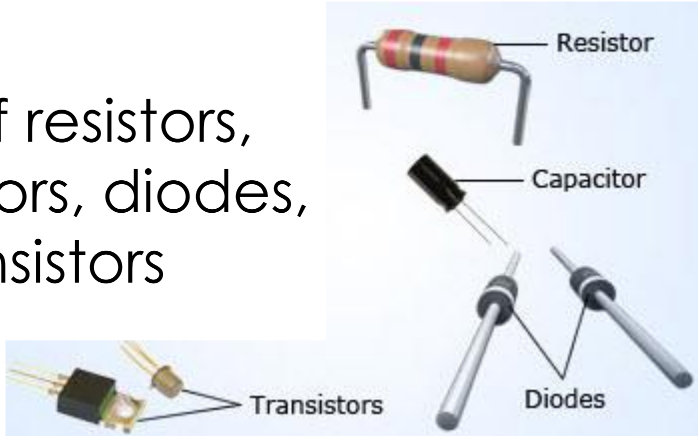
- So, we need to understand a few things about ***digital circuits***

Types of components found in a digital system such as a microprocessor

Microprocessor

Source: https://www.newegg.ca/Product/Product.aspx

- Made of resistors, capacitors, diodes, and transistors



Source: https://www.elprocus.com/semiconductor-devices-types-and-applications

- For example, 10-core Core i7 Broadwell-E (2016) from Intel contains 3,200,000,000 transistors

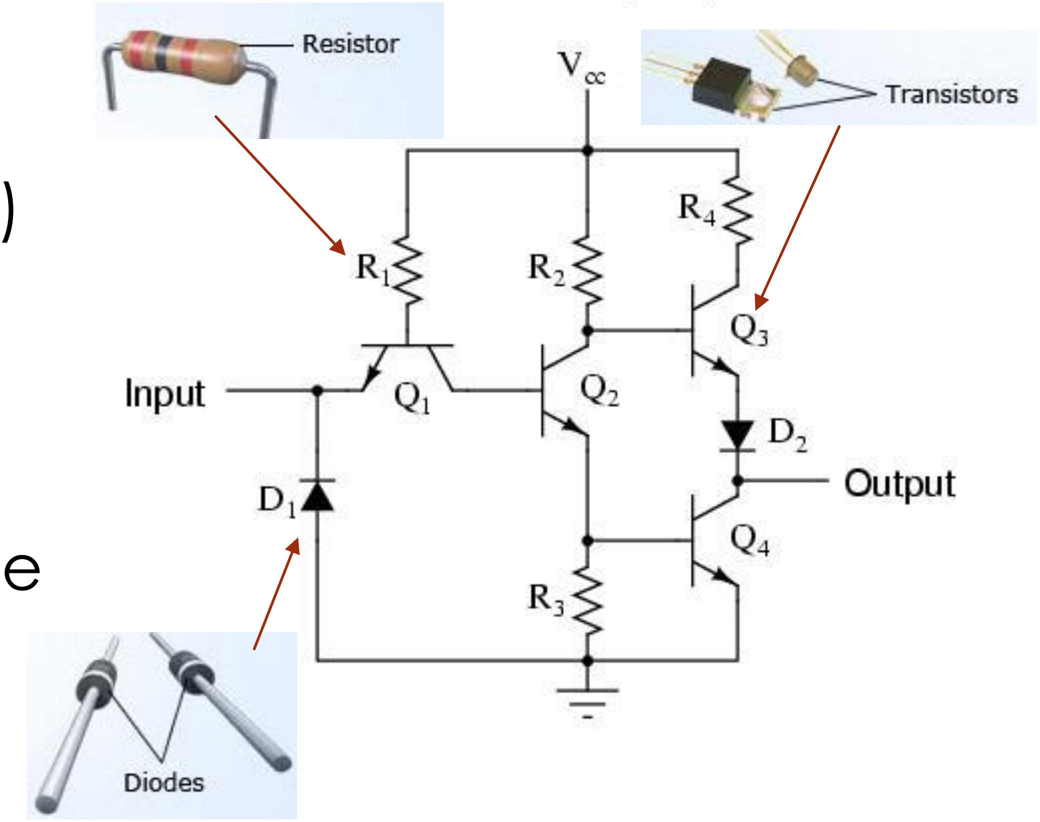
Source: https://en.wikipedia.org/wiki/Transistor_count

- Lots of incredibly small components

Logic gates

- Definition: A logic gate is an electronic device that can perform a Boolean function (AND, NAND, OR, XOR, NOT)
- On a chip, these can be made using transistors, resistors, and diodes
- Here is how a NOT logic gate is constructed:

Practical inverter (NOT) circuit



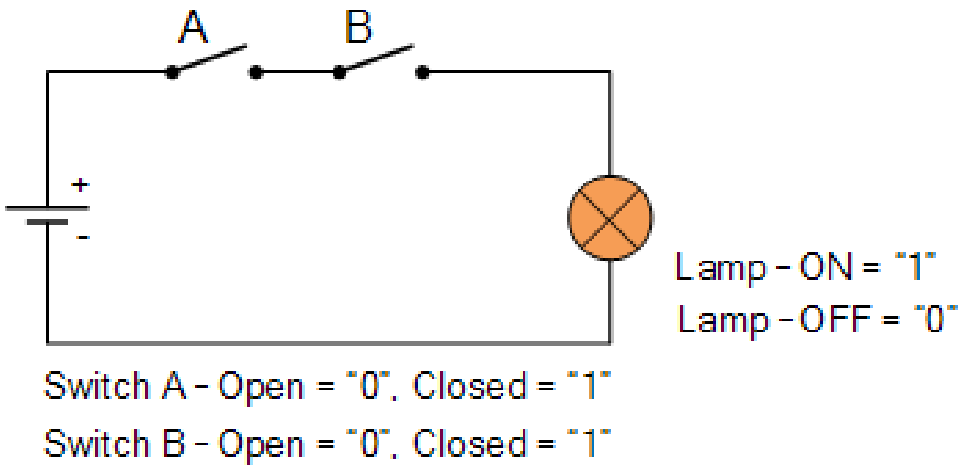
Source: http://www.cybermike.net/reference/liec_book/Digital/DIGI_3.html

Behaviour of logic gates

For the lamp to turn ON (for the current to go through the wire) …

- Here is a diagram representing the behaviour of an AND logic gate:

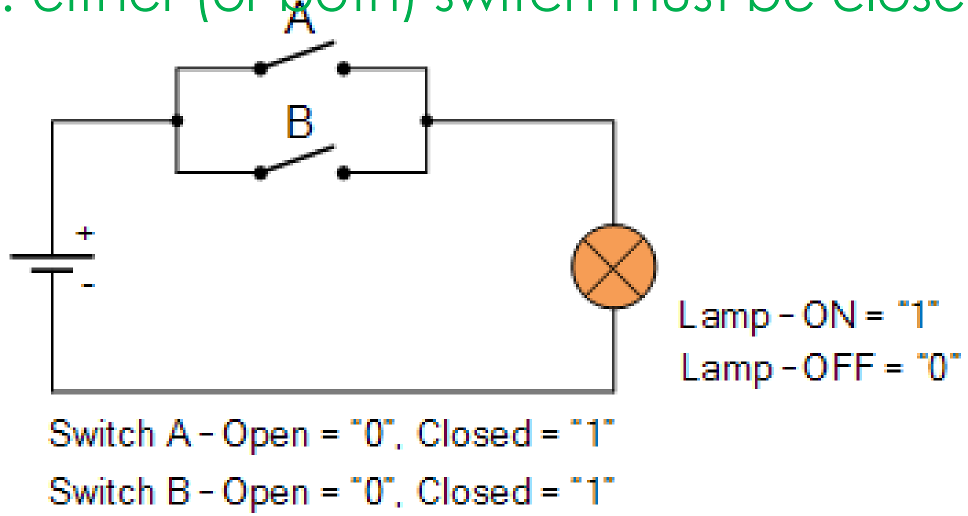
… both switches must be closed!



A
B
+
-
Lamp - ON = "1"
Lamp - OFF = "0"
Switch A - Open = "0", Closed = "1"
Switch B - Open = "0", Closed = "1"

- Here is a diagram representing the behaviour of an OR logic gate:

… either (or both) switch must be closed!



A
B
+
-
Lamp - ON = "1"
Lamp - OFF = "0"
Switch A - Open = "0", Closed = "1"
Switch B - Open = "0", Closed = "1"

https://www.electronics-tutorials.ws/boolean/bool_1.html

https://www.electronics-tutorials.ws/boolean/bool_2.html

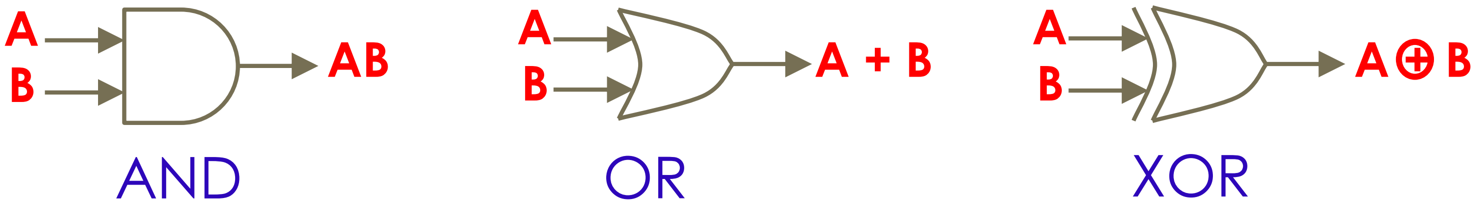
Abstracting using *black boxes*

- A ***black box*** is used to abstract the function of a device
  - The input and output of the device are visible/known
    - The idea is that we need to know these in order to use the device
  - The implementation of the devide (what is inside) is invisible/unknown, i.e., hidden
    - The idea is that we do not need to know how the device is implemented in order to use it

- Same thing is true for functions in software!

Abstracting logic gates

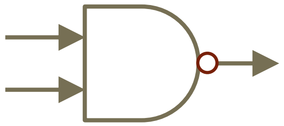
- Instead of drawing logic gates using their electronic components, we hide these components using a ***black box*** -> a symbol simplified representing a logic gate
- Symbols:



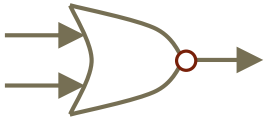
- Input: a signal i.e., 0 or 1 (abstraction of voltage levels) travels along the input wire/line
- Output: After a time delay (**propagation delay t_{pd}**), a signal, i.e., 0 or 1 travels along the output wire/line
- **Always active**
  - As soon as signal (0 or 1) travels along the input wires/lines, the logical gate produces a result, i.e., a signal (0 or 1) which then travels along the output wire/line

Abstracting logic gates – cont’d

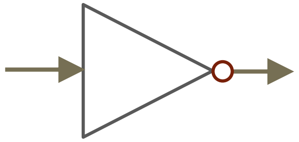
- Symbols:



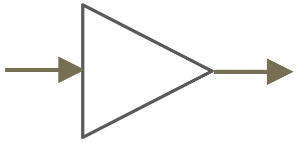
NAND



NOR



NOT
(inverter)



BUFFER

Propagation delay t_{pd}

- Definition: Longest time elapsed between the application of an input and the occurrence of the corresponding output

- t_{pd} often expressed in picosecond (10^{-12} seconds) to nanosecond (10^{-9} seconds)

Summary

- We have now started to explore how the microprocessor executes machine instructions (series of 0’s and 1’s)
  - More specifically, how its datapath can be constructed
- Microprocessor itself is …
  - Made of resistors, capacitors, diodes, and transistors
  - Billions of them, so understanding their behaviours (what they do) once they are linked together is **too onerous**
  - So we resort to abstraction (***black box***) in order to understand their functioning
    - Logic gates: perform a Boolean function
  - Hardware components (i.e., logic gates) have propagation delay
    - Signals (0’s and 1’s) take time to propagate through them

Next Lecture

- Instruction Set Architecture (ISA)
  - Definition of ISA
- Instruction Set design
  - Design principles
  - Look at an example of an instruction set: MIPS
  - Create our own
  - ISA evaluation
- Implementation of a microprocessor (CPU) based on an ISA
  - Execution of machine instructions (datapath)
  - Intro to logic design + Combinational logic + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions + Hazards