# Architecture: von Neumann

*COSC 208, Introduction to Computer Systems, 2022-03-08*

## Outline

- Warm-up
- von Neumann Architecture
- Hardware building blocks

## Warm-up

- Assume you are given the following code:

```
struct account {
    int number; // Account number
    int balance; // Current account balance
};
struct account *open_account(int starting);
int close_account(struct account *acct);
```

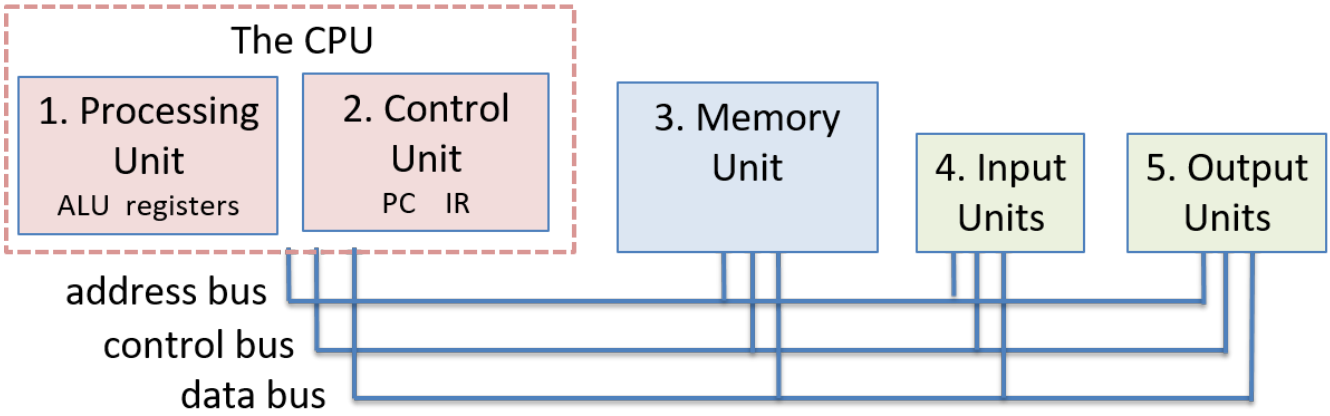
- Q1: *Write the `open_account` function, which creates a new account with a random account number and the specified `starting` balance.*

```
struct account *open_account(int starting) {
    struct account *new = malloc(sizeof(struct account));
    new->number = rand();
    new->balance = starting;
    return new;
}
```

- Q2: *Write the `close_account` function, which eliminates the account `acct` and returns the remaining balance.*

```
int close_account(struct account *acct) {
    int remain = acct->balance;
    free(acct);
    return remain;
}
```

## von Neumann Architecture




- *Where are instructions stored prior to execution?* — memory unit
- *Where are instructions stored during execution?* — instruction register
- *Where is data stored when it is not in use?* — memory unit
- *Where is data stored when it is being operated on?* — (general purpose) registers
- Notice: instructions and data are both stored in the memory unit, but there are different registers for instructions and data in the CPU
- Fetch-Decode-Execute-Store cycle
    - *What happens in the fetch stage?* — The control unit loads the next instruction from memory, based on the program counter, into the instruction register
    - *What happens in the decode stage?* — break instruction into operation and operands; load operands from memory into registers, if necessary
    - *What happens in the execute stage?* — The ALU performs the operation on the operands
    - *What happens in the store stage?* — The control unit stores the result in memory
- *How can we make this cycle faster?*
    - Pipelining
    - Parallelism
    - Faster bus
    - Faster ALU/control unit
    - Faster memory
    - Use separate memory units for storing instructions and data and separate buses for loading/storing instructions and data; known as the Harvard Architecture, which addresses the von Neumann bottleneck

## Hardware building blocks

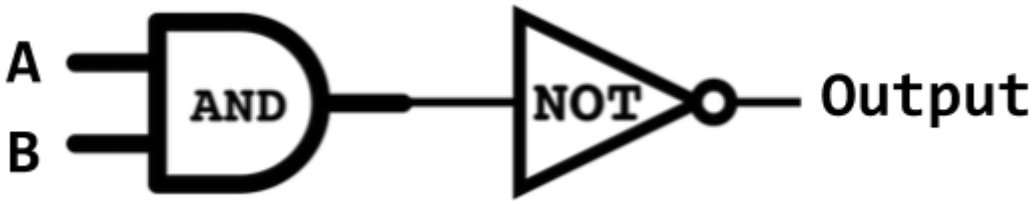
- Transistors — switches that control electrical flow; output state depends on current state plus input state
- Logic gates — created from transistors; implement boolean operations (AND, NO, NOT, etc.)
- Circuit — created from logic gates
- Processing, control, and units — created from circuits

*Fill-in the truth tables for all six types of gates*

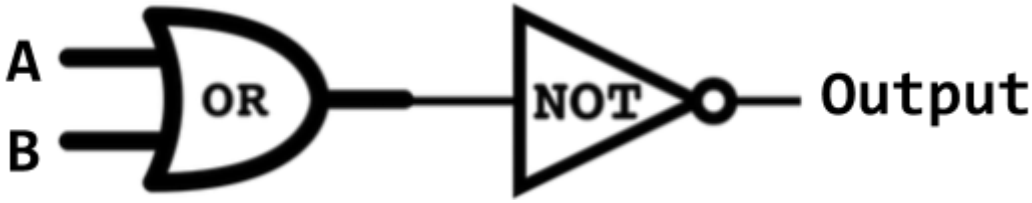
| A | B | A AND B | A OR B | NOT A | A NAND B | A NOR B | A XOR B |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

## Building logic gates

- A chip is easier to build if it contains fewer types of gates
- Q4: *How do you use AND and NOT gates to create a NAND gate?*




- Q5: *How do you use OR and NOT gates to create a NOR gate?*




- Q6: *How do you use NAND gates to create a NOT gate?*


- Q7: *How do you use NAND gates to create an AND gate?*