Architecture: von Neumann; logic gates

COSC 208, Introduction to Computer Systems, 2021-10-01

Announcements

- First DEI reflection due Thursday at 11pm

Outline

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

Warm-up

Assume you are given the following code:

```
struct account {
    int number; // Account number
    int balance; // Current account balance
};
int deposit(struct account *acct, int amount);
int transfer(struct account *from, struct amount *to, int amount);
```

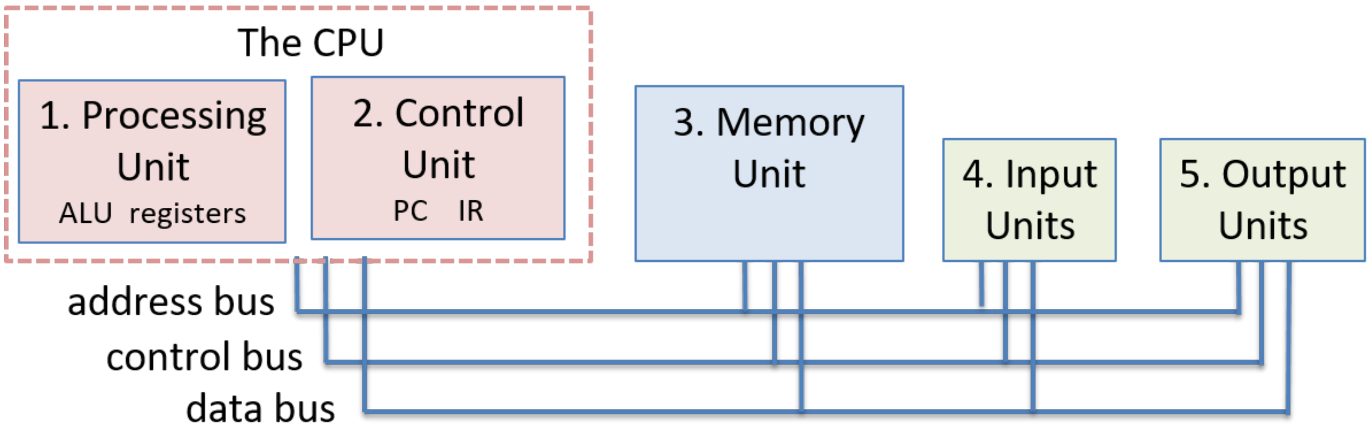
Q1: *Write the `deposit` function, which adds `amount` to the balance of `acct`. The function should return the amount deposited.*

```
int deposit(struct account *acct, int amount) {
    acct->balance += amount;
    return amount;
}
```

Q2: *Write the `transfer` function which moves `amount` from one account to another. The function should return the amount transferred if the transfer was successful or 0 otherwise.*

```
int transfer(struct account *from, struct amount *to, int amount) {
    if (from->balance < amount)
        return 0;
    from->balance -= amount;
    to->balance += amount;
    return amount;
}
```

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

Logic gates

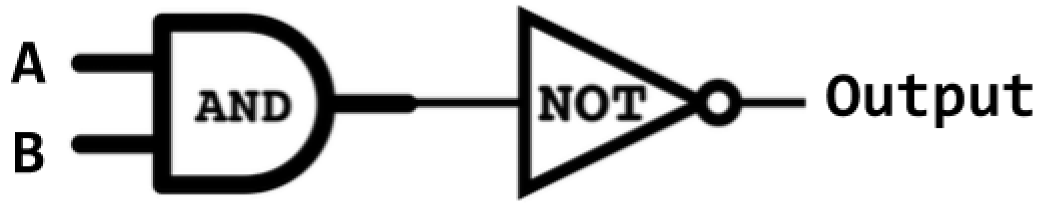
Q3: *Fill-in the truth tables for all six types of gates*

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

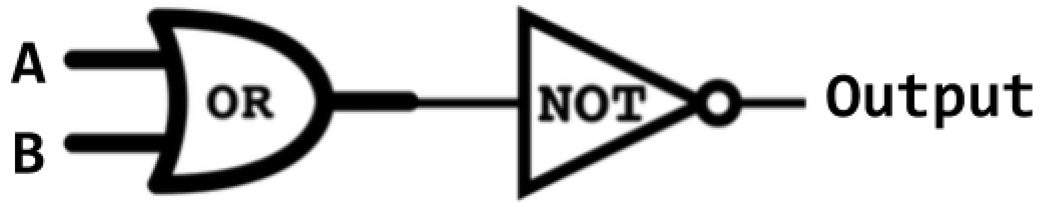
Building logic gates

- A chip is easier to build if it contains fewer types of gates

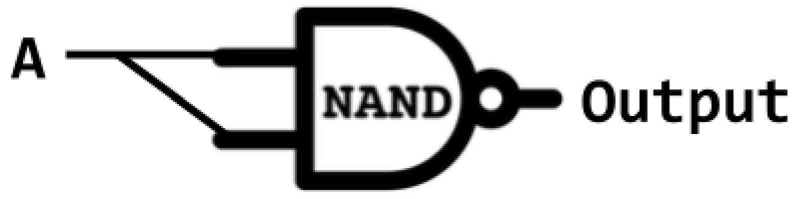
- Q4: *How do you use AND and NOT gates to create a NAND gate?*



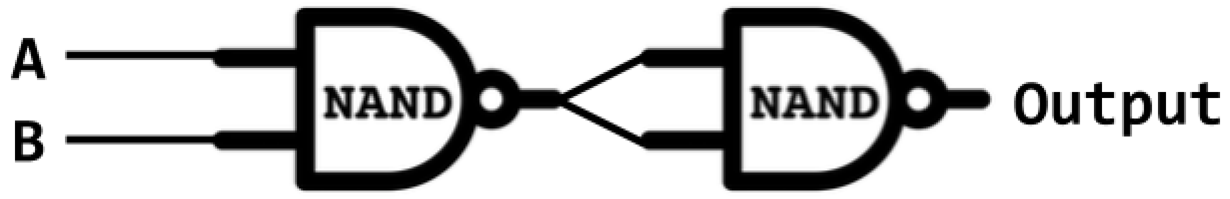
- Q5: *How do you use OR and NOT gates to create a NOR gate?*



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



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

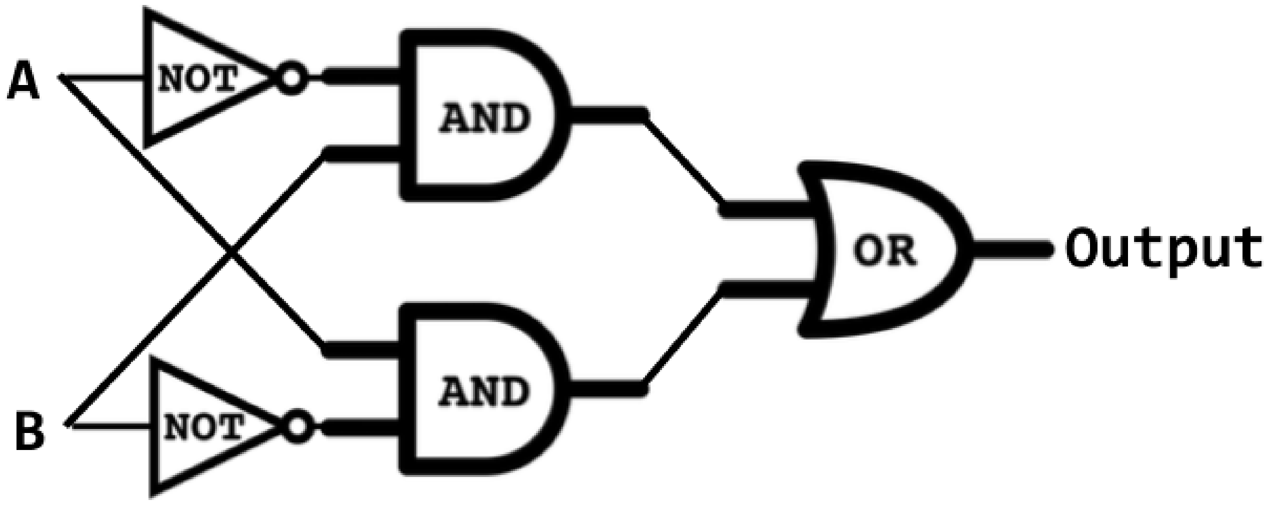


1-bit circuits

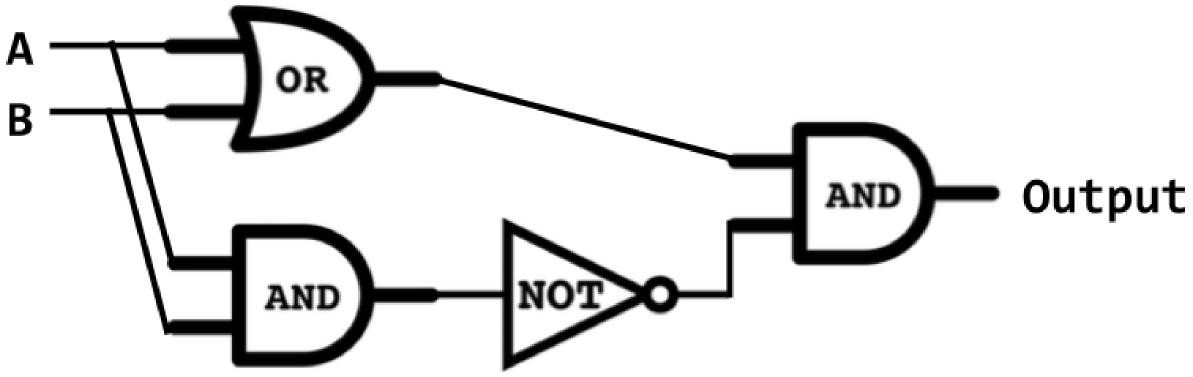
- Connect logic gates to perform a more complex operation
- *Design the truth table: e.g., A != B*

A	B	A != B
0	0	0
0	1	1
1	0	1
1	1	0

- For each row where the output value is 1:
  - Determine how to make each input 1 — e.g., either `A` or `NOT(A)`
  - Conjunct the two subexpressions — e.g., `NOT(A) AND B`
- Create the disjunction of the expressions for each row — e.g., `(NOT(A) AND B) OR (A AND NOT(B))`
- Create a circuit from left to right, starting with the inner-most subexpressions



- *Can we build a circuit that uses fewer gates?*
    - `A XOR B`
    - `(A OR B) AND (NOT (A AND B))`


A
B
OR
AND
NOT
AND
Output