Hardware Implementation

Lecture B17

Lecture notes section B17

Last time

- Analysis of translation
- Writing efficient C

In this lecture

- Gates
- Combinatorial logic
  - adder
  - ALU
- Sequential logic
  - flip-flop
  - memory
- CPU
  - fetch-execute cycle

How to make a computer

- Computers are electronic equipment
  - contain many connected electronic components
    - memory
    - Central Processing Unit (CPU)
    - specialized devices (network, video, etc.)
  - components largely made of circuits containing wires and gates

Gates

- Tiny electronic switches
  - made of transistors
  - implement Boolean logic
    - 0/1 ⇔ low/high voltage
    - digital
  - for given inputs, produce known output
    - according to truth table
  - represented in circuit diagrams by symbols

Gates

NOT

In	Out
0	1
1	0

AND

In 1	In 2	Out
0	0	0
0	1	0
1	0	0
1	1	1

OR

In 1	In 2	Out
0	0	0
0	1	1
1	0	1
1	1	1

XOR

In 1	In 2	Out
0	0	0
0	1	1
1	0	1
1	1	0

Combinatorial logic

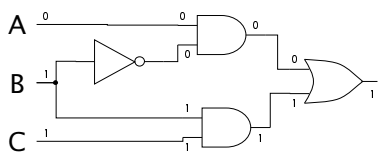
- What about more complex truth tables?
- Made by combining many gates together
  - wires connect inputs and outputs
  - each wire can carry only one bit, 0 or 1

Combinatorial logic

- Circuits made from collections of gates
  - outputs depend only on inputs
    - not on prior state
  - characterized by truth table
  - comparable to Boolean expression

Combinatorial logic

X = (A & ~B) | (B & C)



A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

All three of these are equivalent; each can be converted to the other two.

Combinatorial logic

- Many parts of a computer are constructed of combinatorial logic
  - adder circuit
    - performs addition
  - arithmetic logic unit (ALU)
    - performs many kinds of arithmetic
    - contains adder circuit
  - multiplexer and decoder
    - direct bit traffic between components

Adder

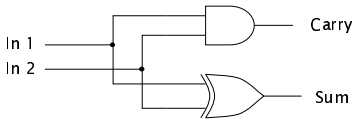
- A combinatorial circuit that adds binary values
  - according to this truth table

In 1	In 2	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum is equivalent to XOR

Carry is equivalent to AND

Adder



This circuit can add two binary digits and is called a half-adder

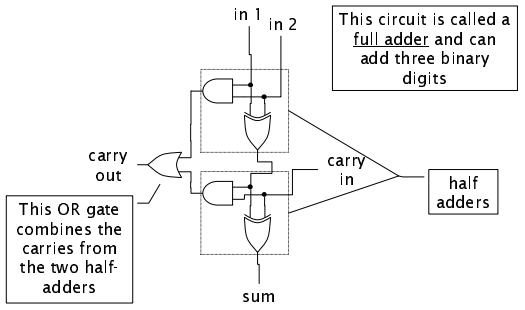
Adder

Why "half-adder". . . ?

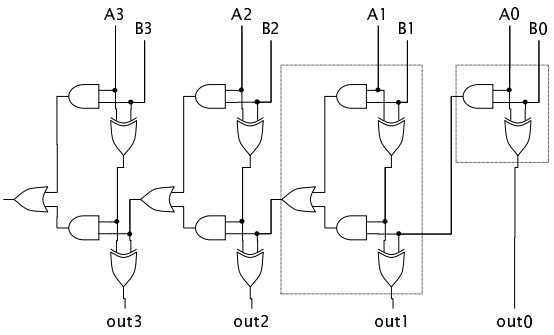
```
    1
  1 5 3
+ 4 3 8
-------
  9 1
```

. . . because to add multi-digit numbers each column requires two additions

Full adder

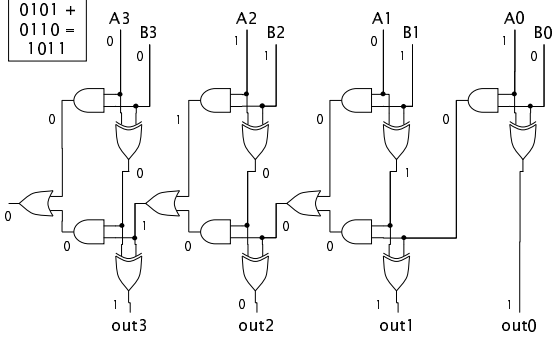


Ripple-carry adder



Ripple-carry adder

0101 + 0110 = 1011

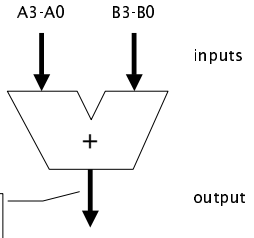


Ripple-carry adder

- So named because gate results (including carries) propagate ("ripple") from LSB to MSB
  - right to left
  - corresponds to how humans add numbers with pen and paper
- More sophisticated, faster, adder circuits exist
  - see CSE3304

Adder

Adders (and other arithmetic circuits) are usually drawn like this in block diagrams

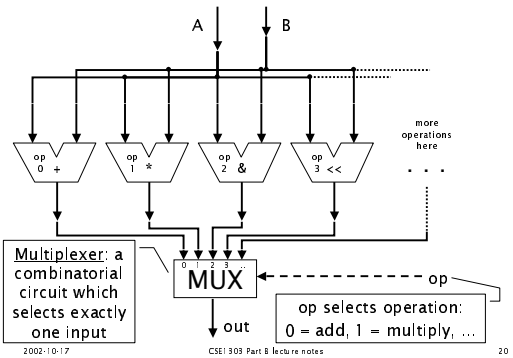


collections of parallel, related wires like this are known as buses; they carry multi-bit values between components

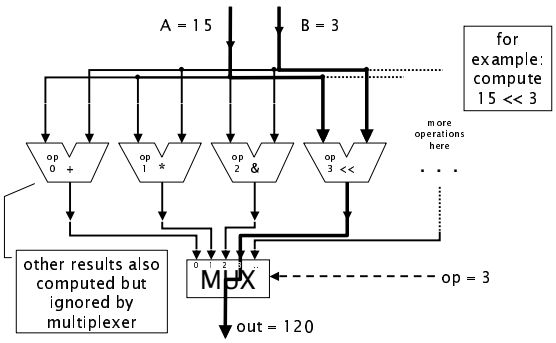
Arithmetic

- Computers need to do more than just addition
  - arithmetic: + - * / %
  - logic: & | ~ << >>
- Need a circuit that can select operation to perform

Arithmetic Logic Unit (ALU)



Arithmetic Logic Unit (ALU)



Memory

- Computers need memory for storage
- Different kinds of memory distinguished by speed, size, cost and proximity to CPU
  - main memory
    - slowish, huge, cheap, far from CPU
    - typical size 10^9 bits
  - cache
    - fast, medium-sized, expensive, near to CPU
    - typical size 10^6 bits
  - registers
    - extremely fast, tiny, very expensive, located on CPU
    - in MIPS 32x32 = 1024 bits

Memory

- The smallest piece of memory is a single binary digit (bit)
  - can hold 0 or 1 only
- A one-bit memory is called a flip-flop or latch
  - because its value can flip and flop between 0 and 1
  - because it can latch onto a value and store it

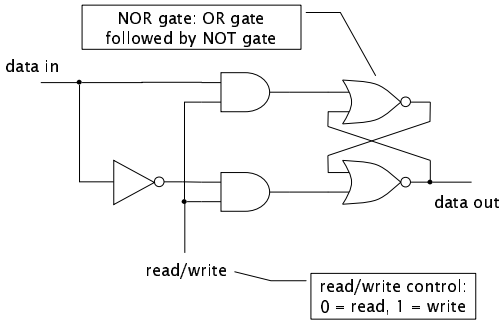
Flip-flop

- Flip-flop needs two operation modes
  - write: store (memorize) a value
  - read: load (recall) a previously stored value
- Also need
  - data in
    - for telling flip-flop what value to store (0 or 1)
    - used only when writing
  - data out
    - for finding out what value flip-flop currently contains (0 or 1)
    - used only when reading

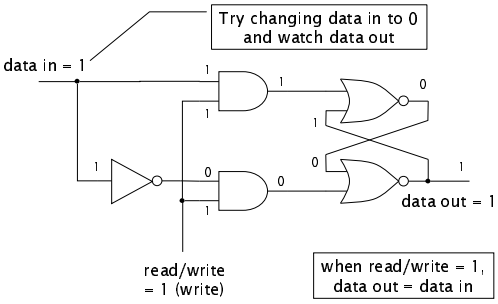
Flip-flop

- Flip-flop can be implemented with gates
- Not combinatorial logic
  - because current output may depend on previous state
- Example of sequential logic
  - current output depends on inputs and prior output

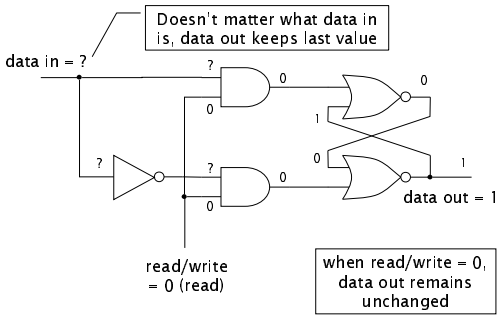
Flip-flop



Flip-flop: writing

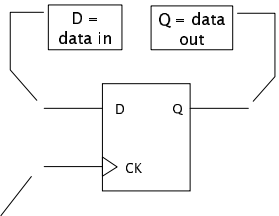


Flip-flop: reading



Flip-flop

Flip-flops are often drawn like this in block diagrams

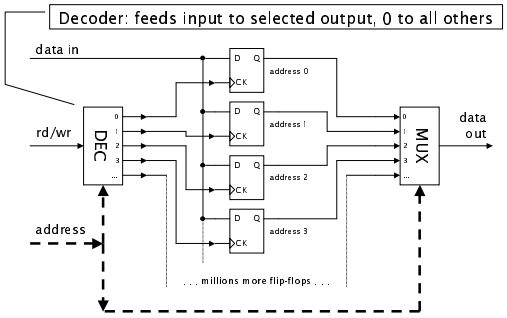


CK is read/write ("clock" because this input is often connected to computer's processor clock)

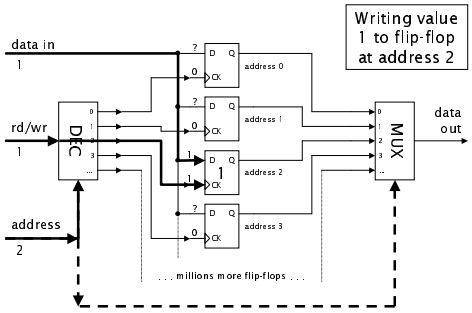
Memory

- Memory can store many bits independently
  - many flip-flops
- Need to identify which bit (flip-flop) to read or write
- Give each flip-flop a unique number (address)

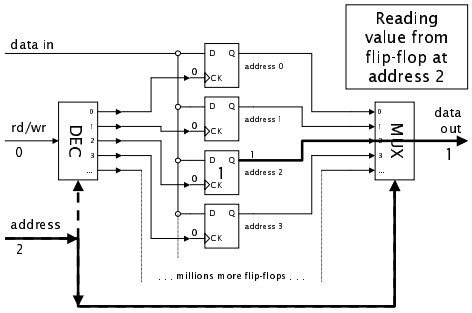
Memory



Memory: writing



Memory: reading



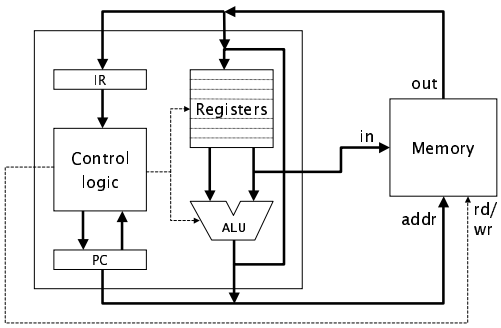
Memory

- Memory usually operates in terms of bytes (8 bits), not single bits
- Repeat memory circuit eight times
  - connect each memory circuit to one of the eight lanes of the data bus
  - reads and writes occur in parallel for each bit in byte

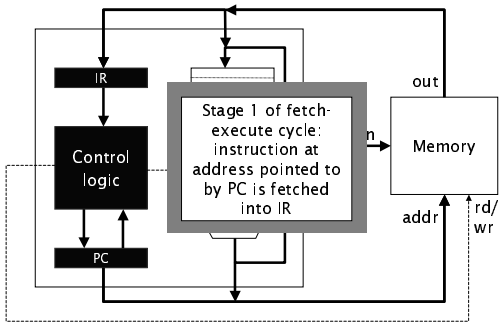
Central Processing Unit (CPU)

- Coordinates all computer's components according to program being run
- Contains
  - registers
  - ALU
  - program counter (PC)
    - address of current instruction
  - instruction register (IR)
    - copy of current instruction
  - control logic
- Runs programs using fetch-execute cycle

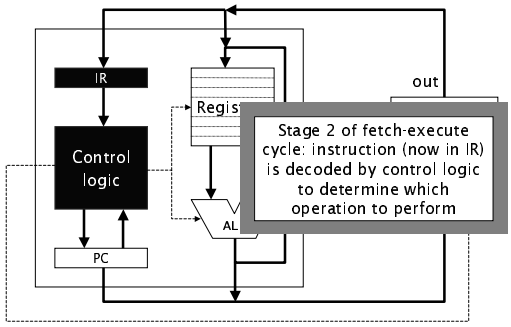
CPU



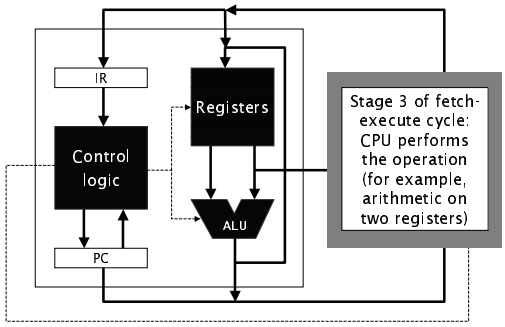
Fetch-execute cycle: fetch



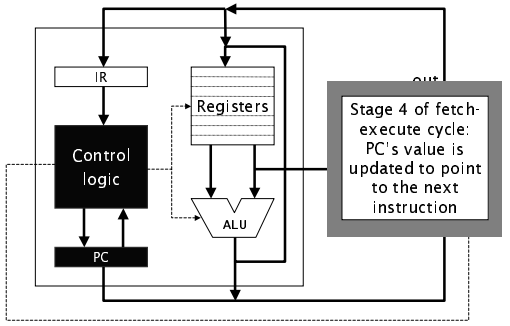
Fetch-execute cycle: decode



Fetch-execute cycle: execute



Fetch-execute cycle: update PC



Covered in this lecture

- Gates
- Combinatorial logic
  - adder
  - ALU
- Sequential logic
  - flip-flop
  - memory
- CPU
  - fetch-execute cycle

Going further

- Digital logic
  - CSE1101/CSE1308/CSE2306
- Implementing computers
  - CSE1102

Next time

- Revision

Copyright