Circuits for Computing

Michael L. Littman

CS 22 2020

March 6, 2020
Overview

Feed forward circuits

Machine language

Circuit for Mini Machine Language
What we can compute

So far:

- **IfThenElse5**: Given two 5-bit numbers and a control bit, return either the first or second.
- **Equal5**: Given two 5-bit numbers, return whether they are the same.
- **Add5**: Given two 5-bit numbers, produce their 5-bit sum.
- **Neg5**: Given a 5-bit number, produce its negation.
Composing computations

With these computational units, we can build circuits that solve other problems as well. Example:
Arguments for/against special purpose circuits

For:

- Can make lots of stuff.
- Can be VERY fast when built.

Against:

- Cannot be repurposed.
- Have to build for the DEEPEST possible calculation.
Remainder: Pseudocode

How might we compute rem($a, b$)? That’s the remainder when we divide $a$ by $b$.

while $a > b$:
    $a = a - b$

Not the fastest, but should work. We can make circuits for subtraction, but “while”?

Here’s what we’re going to do. We’ll invent a computer language that’s circuit friendly, write our algorithm in that language, then, design a circuit for interpreting the language.
Mini Machine Language

Each line of code has a number, executed in order. There’s an array of memory. There’s just one variable called the “accumulator”.

- 000 ADD($m$) add the contents of memory location $m$ to accumulator
- 001 STA($m$) store the value in the accumulator into memory location $m$
- 010 SET($v$) set accumulator to value $v$
- 011 BRP($l$) transfer control to line $l$ (“branch”) if value in accumulator is positive (or zero)
- 100 NEG negate accumulator
- 101 HLT halt, program complete
- 110 NOP do nothing (no operation), continue execution
- 111 NOP do nothing (no operation), continue execution
Remainder in mini machine language

Commands: ADD, STA, SET, BRP, NEG, HLT, NOP.

Why these? Close to what we know how to build in circuitry. Sufficient to express any (!) computation, given enough time and memory.

0: SET(0)
1: ADD(14)
2: BRP(4)
3: NEG
4: STA(14)
5: HLT

What does it do? Loads the value of memory location 14 into the accumulator. If it’s positive, it skips over negating the accumulator. It stores the value in the accumulator in memory location 14, and halts. Absolute value!
Remainder in mini machine language

Here’s the remainder algorithm with a loop. Memory location 1 is $a$ and memory location 2 is $b$. (Almost. Doesn’t handle 0 remainder.)

0: SET(0)
1: ADD(2)
2: NEG
3: ADD(1)
4: BRP(6)
5: HLT
6: STA(1)
7: SET(1)
8: BRP(0)
Von Neumann Architecture

▶ memory: RAM = Random access memory. Basically, an array of stored numbers.
▶ commands: Program is list of stored numbers.
▶ arguments: Arguments associated with commands are also stored as a list.
▶ accumulator: Stores one number.
▶ program counter: Indicates which command in the list of commands should be executed next.
▶ Each “cycle”, command fetched from list of commands.
▶ Memory, accumulator, and program counter are updated based on the command.
Lookup

LookUp5 takes a 5-bit address, \(2^5 = 32\) 5-bit numbers and returns the 5-bit number at position “addr” of the list.
5Orx

LookUp5 combines 32 5-bit numbers into one. 5Orx takes \( \times \) 5-bit numbers and ors them together bitwise.
Bottom out

At bottom level, two 5-bit numbers are ored bitwise.
Accumulator update
Program counter update

Circuit for Mini Machine Language

Circuits for Computing

UpdatePC

cmd
acc
param
PC

00101 (HLT)
Equal5

00011 (BRP)
Equal5

IfThenElse5

IfThenElse5

00001
Add5

PC
Memory update

Circuit for Mini Machine Language

UpdateMemory

00001
Equal5
(STA)

00000
Equal5

IfThenElse5

cmd

addr

acc

memory

...
Cycle

memory
commands
arguments
acc
PC

UpdateMemory
UpdateAccumulator
LookUp5
LookUp5
UpdatePC
memory
acc
PC