Program Counter

Let's make it count this time

Akilesh Praveen — CMSC398E

UMD

April 22, 2020

Agenda

1. Announcements
   - Projects 5, 6, and 7
2. Intro + Background
   - What we need
3. Building a Program Counter
   - Brainstorm
   - Design Choice- Latches/Flip Flops
   - Design Choice- Adder
   - Bringing it All Together
   - Aside: Ripple Counters
4. Overview + Nuances
5. Demo

Announcements

Projects 5, 6, 7

- Projects 5, 6, and 7 are now released on Piazza
- Relevant instructional material is/will be linked
- They can be done in **any order**, but I would suggest doing them in order (5, then 6, then 7)
- We already did a lecture on Project 5, today we'll be talking about **Project 6**

Intro

Intro

- We've built the ALU; the brains of the operation
- Now we need a few more things to take this from just a calculator circuit to an actual computer
    - Ways to **store** programs
    - Ways to **interpret** those programs
    - Ways to **execute** those programs
    - Ways to **store data** for those programs while they're executing
- We're going to use the digital logic circuit theory to build circuits to address all of these! (Projects 5, 6, and 7)

Intro

- Ways to **store** programs - **ROM** *(Project 5)*
- Ways to **interpret** those programs - **389E Assembly** *(Project 5)*
- Ways to **execute** those programs - **Program Counter** *(Project 6)*
- Ways to **store data** for those programs while they're executing - **RAM** *(Project 7)*
- Today, we'll be talking about ways to execute these programs, using a **Program Counter**.

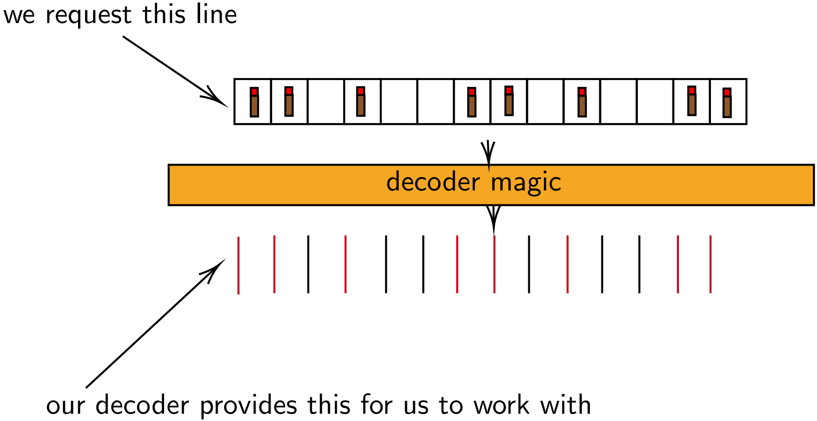
Background

The Story So far

- So far, let's take a look at the system we've got

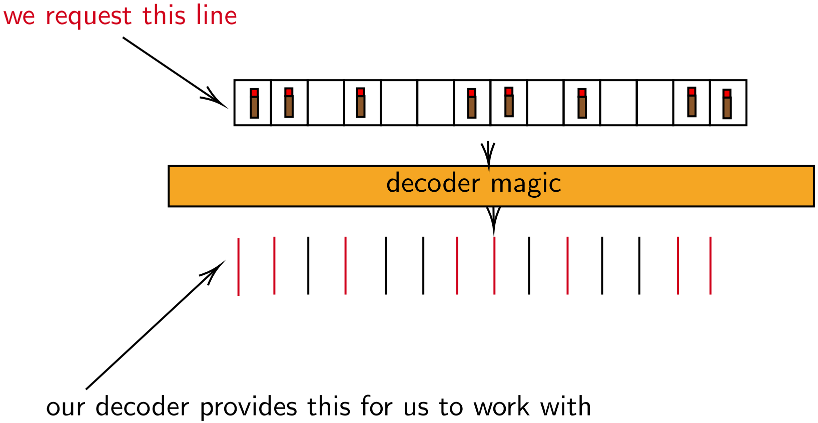
The Story So far

- So far, let's take a look at the system we've got
- I've taken the liberty of black boxing the components



The Story So Far

- Here's what we care about right now.



Requirements

- So we know that we need a way to **request lines**.
- We also know that such a request has to be input in **binary**.
- *If you don't know this already, make sure you know why.*

Sequentiality

- Ok, let's think about the **order** in which we request them
- After all, after we request one line, we're going to need a way to request the next.
    - Then the next after that, then the next after that, etc.
- This is where the idea of sequential logic comes in.

Sequentiality

- And, we aren't just thinking in terms of moving one by one.
- After all, is 1 the only amount we're ever going to be incrementing by?
- What about statements like `BRANCHEQ` in our 389E Assembly? How will we handle that?

Requirements

- Let's not get too confused with it all right now, we're just outlining specifications
- We've pondered enough, let's quantify our goals
  - What we've **got so far**
  - What we **need**

So What've We Got?

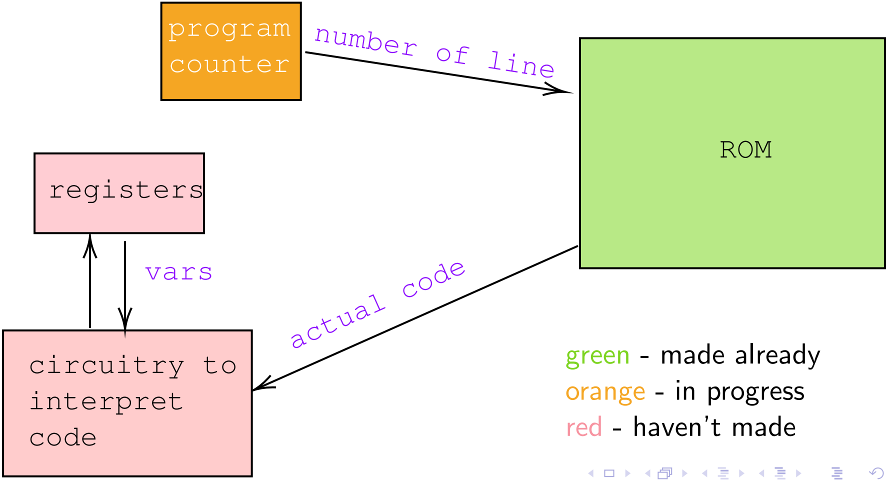
- Excellent ROM Architecture that takes in a **binary number** as a signal and produces the corresponding **line of code**

What Do We Need?

- We need a circuit that produces **line numbers** in **binary**.
- It needs to work **sequentially** (Produce a signal for 1, then 2, then 3, etc.)
- Additionally, it needs to be able to handle `BRANCHEQ` as well.
  - That is, it needs to be able to **increment and decrement by arbitrary quantities**, i.e. **jump** from one line to another.
  - For example, if we needed to 'jump' from line 2 to 5, this could be handled by incrementing our counter by 3.

In Relation To..

- Here's where our new circuit will belong among our other planned components. Again, we're working on the orange one.



green - made already
orange - in progress
red - haven't made

Building the Program Counter

Realizing our Plans

- Now we know exactly what to build, so let's brainstorm how we'd put something like this together
- Similar to last week, it's important to note that we now **have all the conceptual knowledge we need** to get this done
- The question then becomes:
  - How can we leverage the circuits we already know how to make in order to put together a counting circuit?

Realizing Our Plans

- Let's think of all the components we've learned about so far.

latches/flip flops

logic gates

demultiplexer

multiplexer

adder

encoder

decoder

multiplier

Realizing Our Plans

- Now, understand that we need to build a circuit that **stores** a number, then repeatedly **adds** 1 to it. (It also needs to be able to add and subtract arbitrary values to that stored number)
- *Do any of these look like they'd be useful in that case?*

latches/flip flops

logic gates

demultiplexer

multiplexer

adder

encoder

decoder

multiplier

Realizing Our Plans

- Now, understand that we need to build a circuit that **stores** a number, then repeatedly **adds** 1 to it. (It also needs to be able to add and subtract arbitrary values to that stored number)
- *Do any of these look like they'd be useful in that case?*

latches/flip flops

logic gates

demultiplexer

multiplexer

adder

encoder

decoder

multiplier

Realizing Our Plans

- It turns out, in order to implement our program counter, all we really need is a clever combination of an **adder** and **latches/flip-flops** (in other words, memory).

Why Latches?

- First and foremost, we need to keep track of what line we want to be pulling from
- At the end of the day, we need to **store** a value of the current line we're at, so we can feed this value into the ROM (see previous slides)
- If you'll think back to the Memory lecture, you'll know that there's an obvious choice for us in terms of storing memory in our circuits

Why Latches?

- You may be tempted to recall that latches can only store a single bit's worth of state- 1 or 0
- However, remember that our ROM needs input in binary
- Thanks to the ROM's binary input requirement, our solution can be just as simple as storing 3 latches worth of data!
- That way, when it's needed, we can just send the values of these 3 latches down to the ROM in order to request a line

Why an Adder?

- Ok, we've figured out how we're **storing** a number, now how would we increment it?
- Adders are an excellent choice for one reason- and to explore this reason, answer the simple question:
    - **Q:** What do we intend to do regarding the ROM at the start of every cycle?

Why an Adder?

- Adders are an excellent choice for one reason- and to explore this reason, answer the simple question:
  - **Q:** What do we intend to do regarding the ROM at the start of every cycle?
  - **A:** We intend to **add 1** to the program counter.

Why an Adder?

- Adders are an excellent choice for one reason- and to explore this reason, answer the simple question:
  - **Q:** What do we intend to do regarding the ROM at the start of every cycle?
  - **A:** We intend to **add 1** to the program counter.
- You might be tempted to ask, aren't there simpler ways of adding 1 to our number stored in memory?

Why an Adder?

- Adders are an excellent choice for one reason- and to explore this reason, answer the simple question:
    - **Q:** What do we intend to do regarding the ROM at the start of every cycle?
    - **A:** We intend to **add 1** to the program counter.
- You might be tempted to ask, aren't there simpler ways of adding 1 to our number stored in memory?
    - The answer: certainly! But remember, we don't **just** want to be able to add 1

Why an Adder?

BRANCHEQ

- The above call is known as a Branch statement
- Let's examine its core functionality
- All it really is asking us to do is **increment** or **decrement** the **program counter** by an arbitrary amount
- In order to support this functionality, we're going to leverage an adder (with a built in subtract flag!)

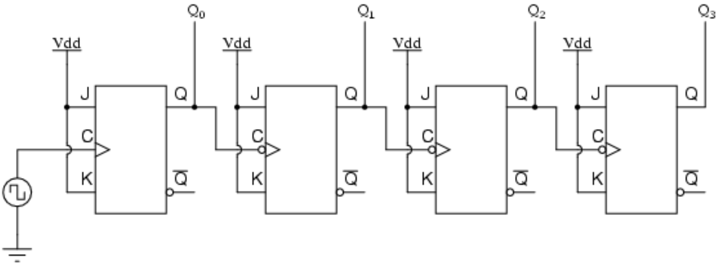
Bringing It All Together

- Now all we need to do is combine the two, in a meaningful way
- Let's use the Latch setup to store our PC variable, and let's hook up an adder to it in order to allow us to add and subtract arbitrary amounts

Ripple Counters

- Here's another way to create a counter, based on only JK-latches
- **https://www.falstad.com/circuit/e-counter.html**
- By 'chaining' these latches together, it allows us to pulse them separately and increment them one by one

A four-bit "up" counter



-

Overview + Nuances

Key Takeaways

- The Program Counter is how we tell which line we're at, and we need to output this line in binary (for the ROM)
    - Fun fact: there is literally a register in many Assembly implementations that stores this value (usually called PC)
- We're building this using **latches** and an **adder**
- Now, we'll be able to
    - Store the value of the line we're at
    - Increment this value by 1 every cycle
    - Increment or decrement by arbitrary values for `BRANCH` statements (Jump statements)

Demo

Building this in Minecraft

- We're essentially looking at an adder, retrofitted with latches and locking mechanisms to facilitate storage of a value
- Now, let's take a look at a demo in MC that I've color coded- this should give you a better understanding of what we're looking to build
- Keep in mind that in our Minecraft implementation, the latches and adder are a little entangled, but we're still looking to accomplish the same basic tasks