How to develop milestones:

1. List of all tasks
2. Preliminary order
3. Go through and rapidly do a free write of each task (as a lesson)
4. Reorder as needed
5. Edit - add diagrams (How much info to give - be generous!)
6. Complete.

Task List:

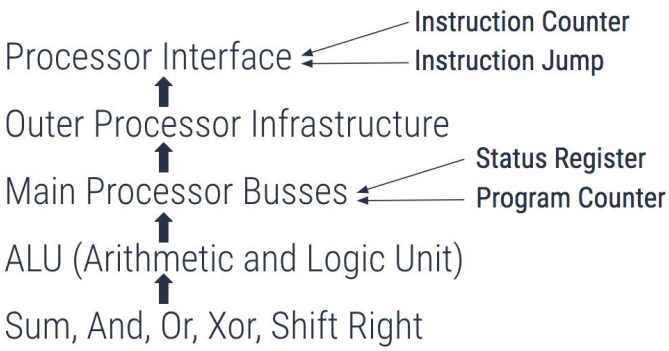
1. ~~Build ALU~~
    a. ~~Each part is a subcircuit~~
2. ~~Build Processor Bus~~
    a. ~~ALU, SP, ACC, X, Y~~
3. ~~Add Processor Status Register to processor Bus~~
4. ~~Build Processor Infrastructure~~
5. ~~Add PC to Infrastructure~~
6. Build Processor Interface
    a. Add Instruction PC
    b. Add the three ROM components
    c. Hook into Infrastructure
7. Develop more opcodes and programs!

Introduction

Welcome to the 6502 Subset Curriculum Program!

By following this set of instructions, you will build a subset of the 6502 processor in LogiSim!

Here is an Overview of how the processor is organized into subcircuits (arrows connect subcircuits to the circuit that builds upon it...



Conventions:

1. / is used as a convention for A into B (A/B)
2. Busses are 8 bit unless otherwise noted
3. Naming conventions are taken from Hanson's Block Diagram (See Last Page)
4. Almost everything is controlled using "Controlled Buffers" - essentially 8 bit bus transceivers
5. Sometimes the word "taken" is used to describe wiring one thing to another i.e input A is taken to register B, meaning input A is wired into register B

Remember to always take a look at Hanson's block diagram, it helps a lot with understanding! Note: Our version of the 6502 handles machine code (random control logic) differently than Hanson describes.

Other Resources:

6502 Reference: http://www.obelisk.me.uk/6502/reference.html
http://www.6502.org/tutorials/6502opcodes.html
http://www.masswerk.at/6502/6502_instruction_set.html

Advanced Resources: https://www.pagetable.com/?p=39
http://www.obelisk.me.uk/6502/addressing.html

Milestone 1

ALU and Its Components

If the processor was a being then the ALU would be the heart, this is the part of the processor that handles all of the math, which essentially is what a processor does at its core. This means that the first step in our journey to build the 6502 processor begins by building the ALU. Note that the 6502 uses 8 bit busses, unless otherwise noted.

Step 1: The Main Components

There are 6 main operations that the 6502 ALU handles.

1. SUM - Adds two numbers together (binary addition)
2. AND - Bitwise AND of the two inputs
3. OR - Bitwise OR of the two inputs
4. XOR - Bitwise XOR of the two inputs
5. SR A - Shifts the binary (of input A) one to the right
6. SR B - Shifts the binary (of input B) one to the right

LogiSim has many of these components built already, just add the inputs and output.

These main components (hmm, maybe we can make these into subcircuits?) make up the ALU, along with some infrastructure built upon it.

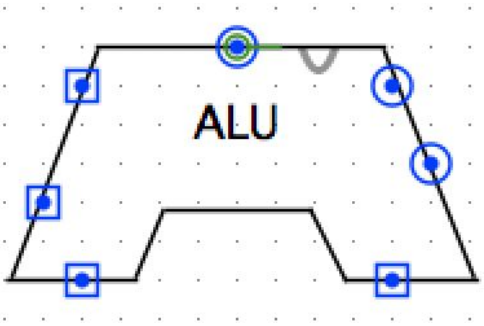
Step 2: The Infrastructure

In order to finish the ALU, one must add a way of selecting one of the functions from the rest (utilizing some control bits) and also output the carry out and overflow of the function that was used (if it applies).

Tasks For This Milestone:

1. Build the 6 main components of the ALU in seperate circuits.
2. Add these to your main ALU circuit
3. Hook these (main components) into something that can select one of them using some control bits
4. Add your inputs and outputs
5. Add circuit to detect overflow
6. Design the final ALU subcircuit.

Your final circuit should look something like this:



Expected Output

Control 1	Control 2	Control 3	Output
0	0	0	SUM
0	0	1	AND
0	1	0	OR
0	1	1	XOR
1	0	0	SRA
1	0	1	SRB
1	1	0	A
1	1	1	B

As another hint, here are the control lines (inputs) you will need to hook up to make this circuit work along with how many bits each input is…

A (8), B (8), Carry In (1), ALU Mux Control (3)

The outputs that need to be hoopked up are as follows…

Carry Out (1), Output (8), Overflow (1)

Milestone 2

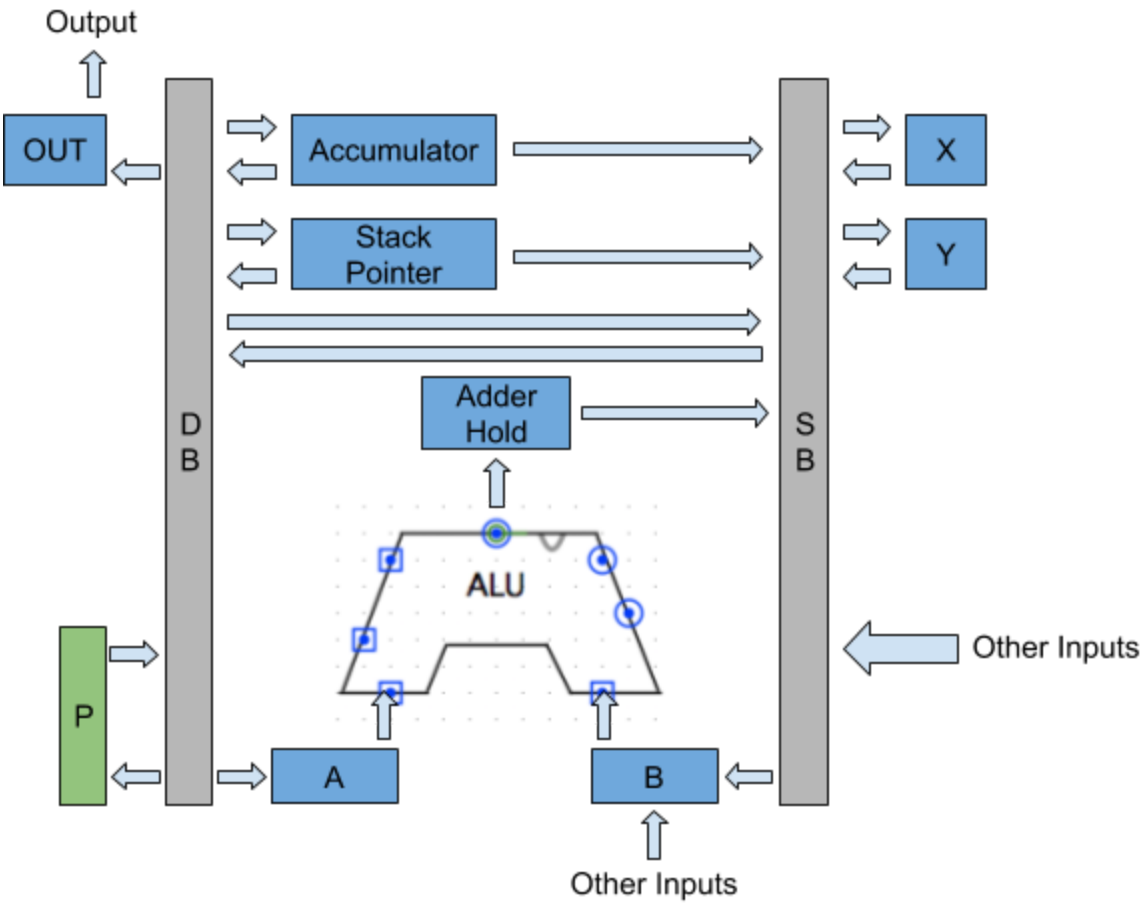
Processor Bus(ses)

The main processor busses are essentially the arteries of the processor (if we are following this being metaphor). In the 6502, there are two main busses (DB and SB) which are basically a collection of wires that carry data around different components. These busses connect the ALU, Stack Pointer, Accumulator, X Register, Y Register, and Processor Status Register. So let's get started!

Step 1: The Busses

The first step is to create the basic layout of this circuit. Take a look at Hanson's block diagram, you can see that the two main busses hook up to a variety of components, all of which we will need to build and connect.

Start by laying out the two busses and mark out space for each component, see the diagram below for a guide on how to do this.



Note that the diagram is an overview, and some things are left out that are defined elsewhere in this guide. Registers are depicted in blue, Subcircuits (aside from the ALU, which we already built) in green, and busses in grey. We will be making the processor status register (P) in the next milestone, so leave that out for now.

Step 2: Hooking it Up

Now you have the main layout, but you need to hook it all up, and more importantly, hook up the control lines that will control what the processor does. The following is a list of control lines you will need for this circuit. This is VERY important, as it will define EXACTLY what needs to be hooked up where. Pay attention to the Hanson Block diagram as you do this, as the same names for control lines are used in this list.

Note: Use tunnels in your circuit to organize it, don't drag inputs around!

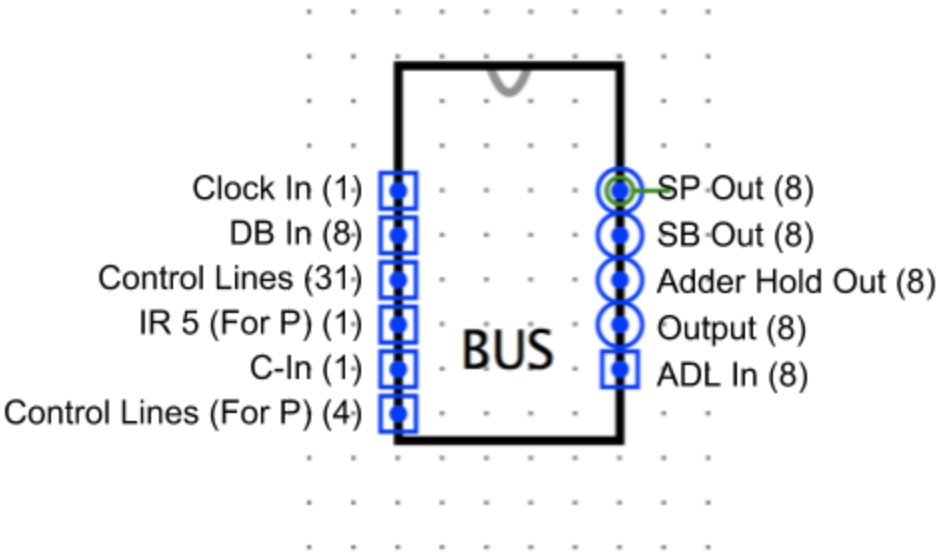
Accumulator/SB	SP/SB	SB/X
DB/Accumulator	SP/DB	X/SB
Accumulator/DB	DB/SP	
Add/SB	SB/DB	SB/Y
DB/A	DB/SB	Y/SB
SB/B	DB/OUT	

ADL/B - ADL has not yet been created, just take this from an arbitrary input (see below).

Once you have all of these control lines implemented, the guts of the circuit are done!

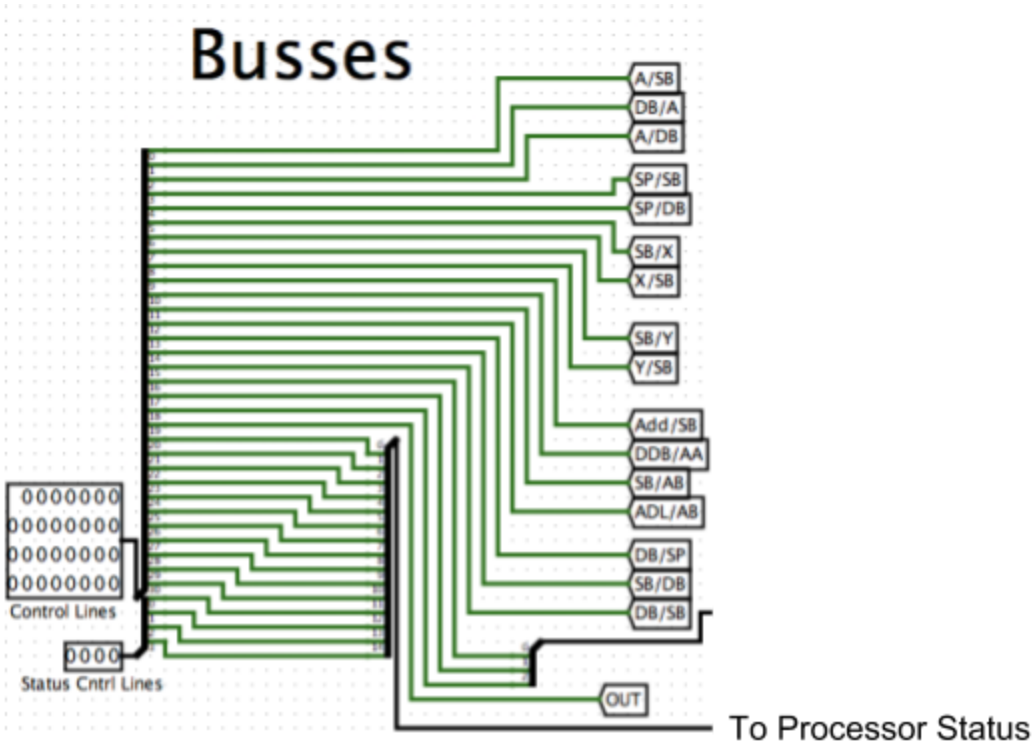
Step 3: Final Hookups

Your final circuit should look like this:



Finalize your wiring by making sure all of those outputs in inputs exist. Note that the number of bits for each output and each input is defined in parentheses. If the input says (For P), do not hook it to anything yet, as this is for the Processor Status Register.

For now and into the future, your control lines should be input on the left side of your circuit and tunneled where they need to go, as an example, here is this circuit's inputs…



Step 4: Testing!

Once everything is hooked up, place some numbers (using the hand tool) on the DB in and change the control lines to move things in and out of each register to make sure the data can transfer successfully around the circuit.

Most importantly, load the A and B and re-test your ALU, ensuring that the output is placed into adder hold successfully.

Now, you have finished the main processor busses circuitry! Head onto the next milestone to add the Processor Status Register (P).

One Final Note: The outputs of your ALU (Carry Out and Overflow) should not be hooked up right now, we will hook them up in the next milestone.

Milestone 3

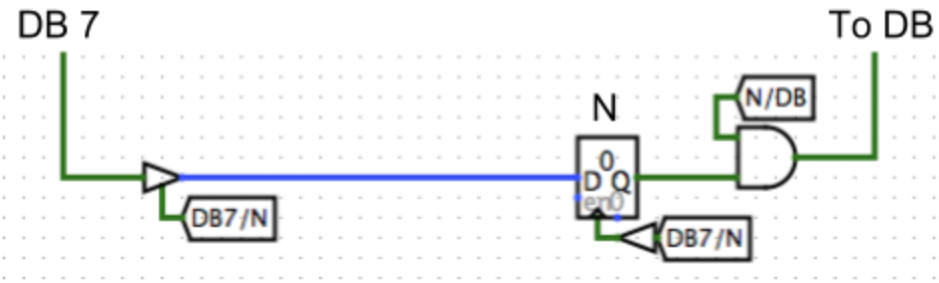
Processor Status Register (P)

The processor status register is a “register” in the 6502 processor that holds vital information about what numbers are in the ALU or have been processed by the ALU. The register originally contained 7 bits of information (one was unused) but our subset will only store 4 bits, and only output one of them at a time.

Note that this is a subcircuit that we will be placing inside the “Processor Busses” circuit.

Step 1: The Registers

This one “register” will be implemented as four separate one bit registers. Below the circuitry for N is given.



DB is inputted into this circuit, so taking Bit 7 is simply done with a splitter. The output of the and gate (or controlled buffer if you want to use it) is sent into and AND gate with the other outputs, which is output back to DB (controlled by a controlled buffer).

These four signals are as follows, along with the inputs for each.

Signal	Inputs (As Defined By Hanson)
Carry (C)	DB0 (DB bit 0), IR 5, ACR (Carry Out of ALU)
Zero (Z)	DB1 (DB bit 1), DBZ (DB Zero, 1 if DB is all 0s)
Overflow (V)	DB6 (DB bit 6), AVR (Overflow from ALU), 1/V (Logical 1), 0/V (Logical 0)
Negative (N)	DB7 (DB bit 7)

Each of these inputs should be on a controlled buffer (see example) so that we can control which input is stored and when we want to store it. For Registers with more than one input, OR the signals before sending them into the register update line.
Gate Delays: The buffer between DB7/N and the register update signal (see example) is there to make use of gate delays in LogiSim. If you ever find that a register is not updating properly, put an OR gate (with one input being ground) between the control line and the update signal, this may fix it.

Step 2: Hooking It Up

All of the inputs for each status signal are noted above, but we will need a few more to ensure that this subcircuit works properly. They are as follows:

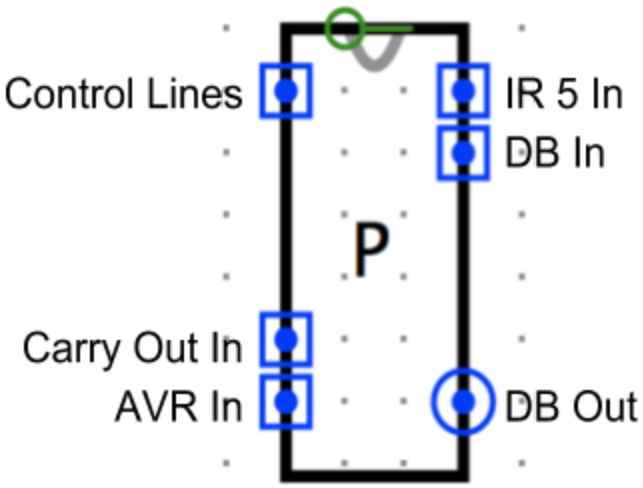
Inputs:

DB In: Many signals utilize bits from the DB, so we will need to input it.
IR 5: I already noted this, but I will reiterate that this is unused for now, but we will connect it later.
COUT and AVR: These are coming from the ALU to be used as defined in the table above. COUT is carry out and AVR is overflow.

Outputs:

C/DB, Z/DB, V/DB, N/DB: Control signals to output one of the four signals to the DB bus, this must be used in conjunction with P/DB.
P/DB: Signal to output the selected register to DB.
DB Out: The connection back to DB that P/DB outputs to.

Here is a final output diagram to show all of the inputs and outputs needed:



Step 3: Testing!

The most important thing to test is that the registers are updating properly. Manually add some more inputs (for testing) and ensure that for each control line, the value (test using 1) is successfully pushed to the register. If this is not working, implement more gate delays (noted above).

The only other thing to test in this circuit is that it outputs to DB properly, so test that as well.

Once the circuit is working, we can hook it up the the processor busses circuitry we made in Milestone 2.

Step 4: Implementation

The final step is to place this subcircuit into the processor busses circuit. Go to the processor busses circuit and drag this subcircuit in.

1. Make sure you have enough inputs in your processor busses circuit (There should be 35 in total), and drag a bus of the last 15 of them to the control lines input of your status register subcircuit. You'll have to keep in mind that order matters here, so however you ordered your control lines in P, you'll have to keep that order from now on.
2. Connect an arbitrary input up for IR 5 (we will connect this later).
3. Tunnel COUT and AVR from the ALU to their proper inputs.
4. Connect DB to both DB In and DB Out.

Then this circuit is connected and completed!

Milestone 4

Infrastructure

The processor infrastructure is essentially the way we will hook up the processor busses, or bus subcircuit to the interface that the programmer will deal with. The infrastructure provides space for the Input Data Latch (See Hanson's block diagram) which takes inputs from the interface, as well as the ADL and ADH address busses (ADL is Address Bus Low (Lower 8 Bits), and ADH is Address Bus High (Higher 8 bits)), which are essentially the busses that take information from DB and SB to the RAM and ROM chips.

Step 1: Hooking Up the Bus(ses)

The first step here is to drag in the Bus circuit, as the main role of the infrastructure is to hook up the inputs and outputs of the bus to the interface, as we stated before.

Drag in the Bus(ses) subcircuit that we created in Milestone 2 (and added to in Milestone 3). This will serve as the basis for this subcircuit.

There are two more "Address" busses that we will add in this circuit. They are called address busses because they carry the address we want to go to in RAM to the RAM.

Drag out a couple busses and implement the following control lines that take data from the busses subcircuit to the ADL and ADH busses.

1. SP/ADL
2. SB/ADH
3. Output (Take the "Output" output of the busses subcircuit and output it)

We will also need to passthrough a few inputs to the processor busses subcircuit (Take inputs into "Infrastructure" and pass them through to the busses subcircuit)

1. Clock
2. IR 5
3. C-IN
4. The control lines for the busses subcircuit (this should be taken in with the rest of the control lines for this circuit but combined using a splitter and then wired directly into the busses "control lines" and "control lines for P" inputs.

Then just wire ADL to "ADL In" of the Busses circuit (no control lines needed).

Step 2: Wiring the Inputs and Outputs

Yes, I know we just wired up many inputs and outputs but the ones we are about to do are separate from the “subcircuit design” and are the “meat and potatoes” of this circuit, they are what we are adding to interface with the Processor Interface.

There are two main parts of this, the Input data Latch, and the ABL and ABH outputs.

The Input Data Latch should be taken in as one 8 bit input. We then wire this input to three places, each controlled by a separate control line. These control lines are as follows (Note that IDL signifies the Input Data Latch):

1. IDL/DB - DB is the “DB In” of the Processor Busses subcircuit.
2. IDL/ADL
3. IDL/ADH

The main outputs of this circuit are ABH and ABL. There are only two more control lines in this circuit, which are ADH/ABH and ADL/ABL. These lines take the ADL and ADH busses and wire them to the ABH (Address Bus High) and ABL (Address Bus Low) outputs.

Step 3: Testing!

This is a relatively simple circuit, so it shouldn’t be too hard to test. The circuit doesn’t do anything outside of organizing inputs and outputs, so simply make sure that the controlled buffers are working and that the input data latch can send data successfully through to the busses subcircuit, and you should be good.

Try taking data from the input data latch and send it into the Accumulator in the processor busses subcircuit (this may be a bit difficult because you’ll need to manually control the control lines, but it is important as it will ensure the subcircuit is connected properly.

Milestone 5

The Program Counter (PC)

Before we can build the “Processor Interface”, we need to build yet another integral part of the 6502, the Program Counter. This beefed up “register” holds the position of the program in RAM, or more specifically, the address of the current instruction or next instruction, controlling the entire program. The Program Counter (PC) may seem confusing, but it is just a register with nine control lines, 3 inputs (+1 for the control lines), and 3 outputs. This subcircuit is simply for organization and readability of our circuitry.

Step 1: The Register

The first step in this circuit is to place a 16 bit register (Note: This can and should be implemented as two separate 8 bit registers), as the program counter has both the Program Counter Low (PCL) and Program Counter High (PCH).

Step 2: Inputs

There are three inputs (outside of control lines) in this circuit that need to be wired into the register we just placed. These inputs need to be wired into the PC using controlled buffers.

The inputs along with their control lines are as follows:

- ADL → ADL/PC
- ADH → ADH/PC
- SB → SB/PCL (PCL is Program Counter Low, the Lower 8 bits of the PC)

Step 3: Outputs

There are also three outputs…

The outputs along with their control lines are as follows:

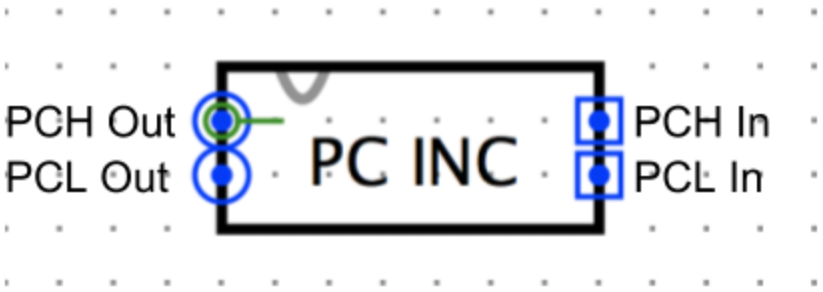
- ADL → PCL/ADL
- ADH → PCH/ADH (PCH is Program Counter High, the higher 8 bits of the PC)
- DB → PCL/DB, PCH/DB

Step 4: PC Incrementer

If you look at Hanson's block diagram, you can see that the PC has its own incrementor outside of the ALU. We will build this as a subcircuit and then hook it into the PC.

To build this subcircuit, take two inputs for PCL and PCH and send them into a 16 bit adder as input A. Input B on the adder should be 1 and carry in 0. Then split the output into PCL and PCH and output it, it's that simple!

Final PC Incrementer:



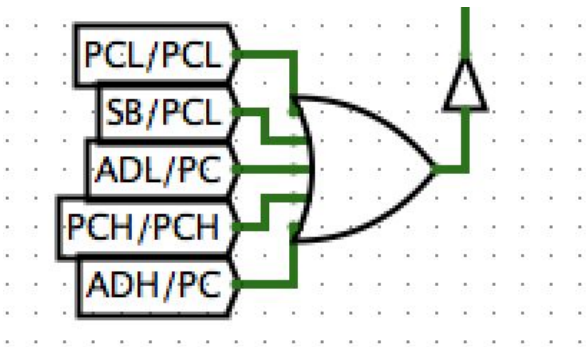
Drag this subcircuit into your PC circuit and hook up the PCL and PCH signals on both sides of the register (remember that a register takes input on its left and outputs to the right).

Each line needs a control line as well, so you will need to hook up two more controlled buffers here: PCL/PCL and PCH/PCH. These lines take the output of the Incrementor and control when we put them back into the PC.

Step 5: Updating the Register

If any control line is triggered (on the input side), we need to update the register. Wedo this by ORing the control lines, and then sending it into the update signal on the register. It also provides a gate delay, which is essential to how this works.

Here is a picture of the ORing:



Step 6: Placing it into the Infrastructure

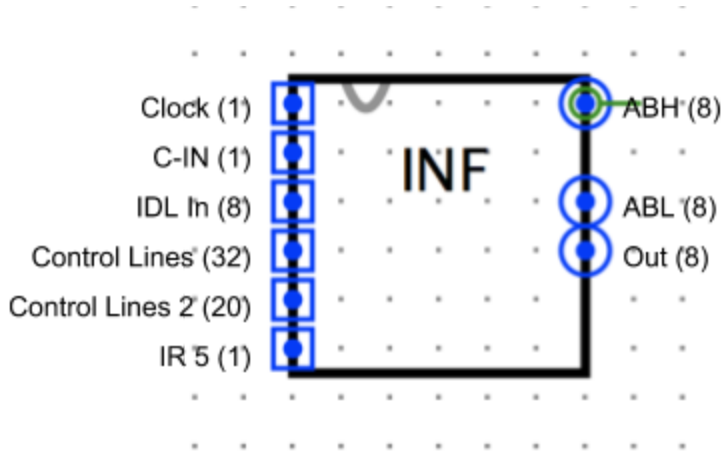
Drag this circuit into the Infrastructure circuit and connect the ADL, ADH, SB, and DB lines. These should all be available to you and connected to other things from the last milestone, so this step should be easy. Then connect the control lines and we are done.

Step 7: Testing!

Ensure that all the control lines are working and that the incrementor is adding one and placing it back into the register on the PCL/PCL or PCH/PCH signal. Also ensure that the register can be updated from the infrastructure, essentially test the connection between the infrastructure and the PC subcircuit.

Then we are done!

Here is a final circuit diagram of the Processor Infrastructure:



Milestone 6

The Processor Interface

Warning: This will be the largest milestone so far with many parts, so take it slow and read carefully.

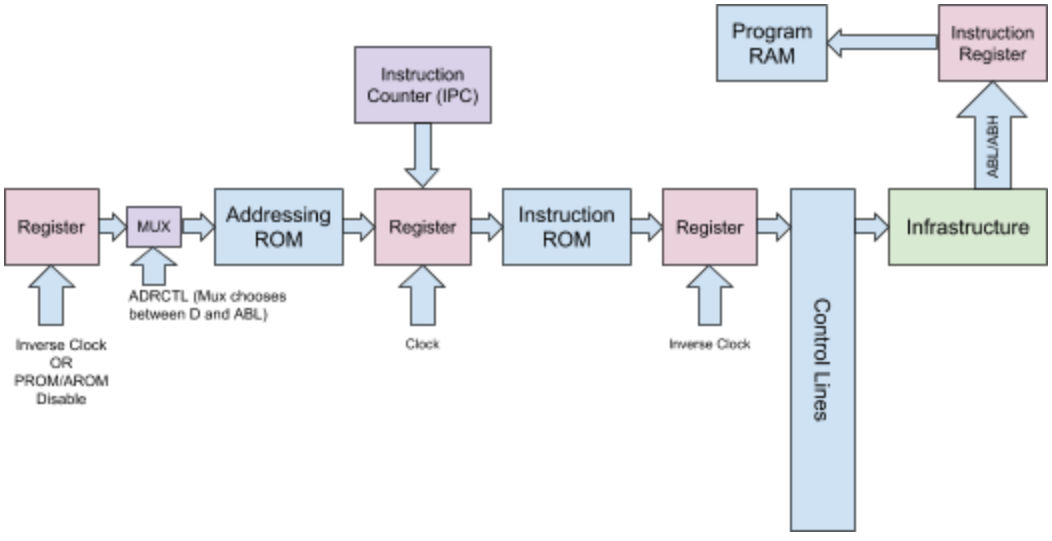
Welcome to Mission Control. The Processor Interface is the command center of the Processor. Before we begin building, we need to have a small lesson on how all of this actually works.

How does this thing even work?

Thus far, we have been building a lot of components using controlled buffers, abe believe it or not, this was purposeful. The way that Hanson describes the 6502 in his diagram is that the entire processor is controlled using “Control Lines” which are essentially signals coming from a central command center (defined by the programmer) to send other electrical signals through different circuitry to accomplish the task the programmer wants to accomplish.

The way that we are going to control these “control lines” (I know the word control is being overused) is through ROM (Read Only Memory) chips. The programmer can call locations in these ROM chips (through the use of opcodes) that in turn send the control signals to the processor. More extensively, the programmer programs an opcode into the RAM chip that then calls a location in the ROM chips. These ROM chips then take over control from the RAM and execute a series of instructions sending the control lines to each part of the processor, which change with each tick of the clock. We will get to programming the processor in the next milestone, so lets build the interface for now.

Here is a bird’s eye overview of this circuit:



Step 1: Placing the Interface

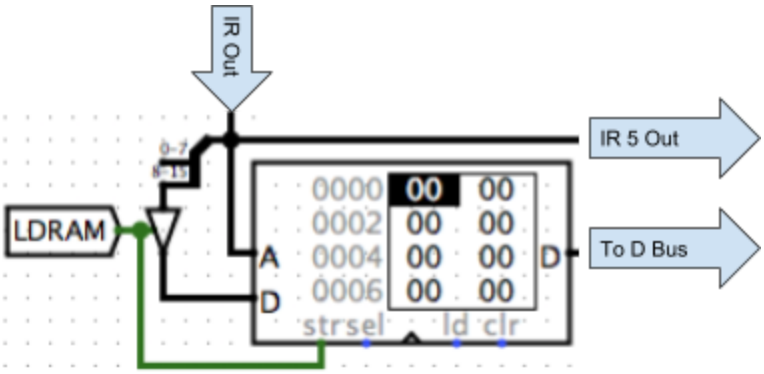
The first thing we need to do is place the infrastructure we have been building for the past couple milestones.

1. Drag in the processor infrastructure
2. Hook up the clock to the clock input as well as a tunnel to use inside this circuit.
3. Connect the output to two hex displays to display the 16 bits of output
4. Place a pull resistor (to zero) on the 16 bit ABL ABH data line.
5. Connect ABL and ABH to a 16 bit register known as the Instruction Register (IR). This should be controlled with PROMCNTRL (control line) and should update with each tick of the clock.

Step 2: The RAM

The RAM is where the programmer (or user of our processor if you will) programs 6502 opcodes which will be executed when the clock starts ticking. It is a bit difficult to hook up the RAM, so it will be given to you as follows…

Note: This is a 16 x 8 RAM



The D bus is the bus that carries the data from the “Data” line of the RAM to the ROM chips. IR 5 should finally be connected to the processor infrastructure as well.

Step 3: Addressing ROM

In order to use 6502 opcodes with their original hex values, we need to convert the opcodes coming from (programmed into) the RAM into the addresses of control lines in the instruction ROM. This is accomplished with the help of another ROM chip, which we will call the addressing ROM.

There are three main parts to this:

1. A register that can be disabled to allow the output of the RAM to be directed back into the input data latch, this is needed in order to allow both the triggering of new control lines or inputting data into the rest of the processor.
2. A multiplexer to choose between the D bus (data coming from RAM), or the ABL output, so that an already running program can trigger another opcode.
3. The actual register that has memory addresses of each set of control lines for each given 6502 instruction code (in hex).

Part 1

The register is a simple 16 bit register, the lower 8 bits hold the D bus and the upper 8 hold the ABL output. For the clock signal, input an OR of the inverse (NOT) clock and the inverse (NOT) of the PROM/AROM Disable control line.

Part 2

Then we need a multiplexer of the output of said register. This is a 1 bit controlled 2 input 8 bit mux. The control line for the multiplexer should be ADRCTL (Addressing ROM Control). A 1 should select ABL while 0 (the default - place a pull resistor to 0 on the control line) should select the D bus.

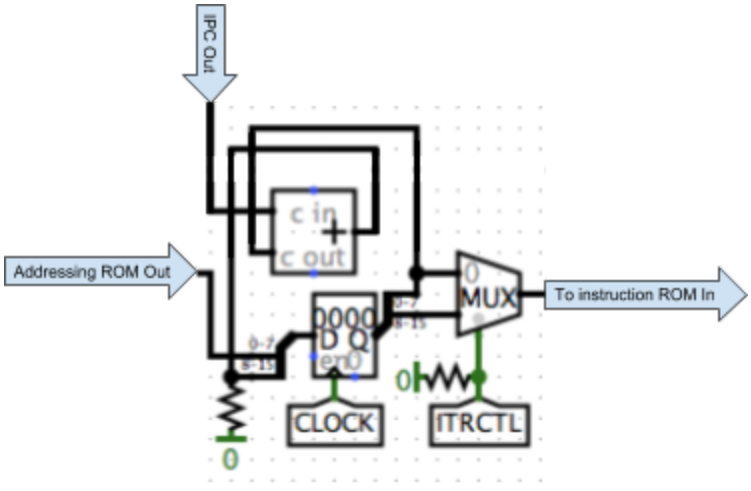
Part 3

This is a simple 8x8 ROM that takes the output of the multiplexer as A and outputs to the next register.

Step 4: Instruction ROM

The instruction ROM contains the actual control lines for each OPCODE. Once an opcode is triggered in the instruction ROM, control is handed off to it and the Instruction PC (which we will build next) takes over for the PC as the controller.

The way this is connected is given here (as it is not only a bit confusing but repeated from the last step):



The adder is there as an intermediary between the Addressing ROM out and the IPC when control is handed off, instead of starting at 0, IPC will start at the given instruction.

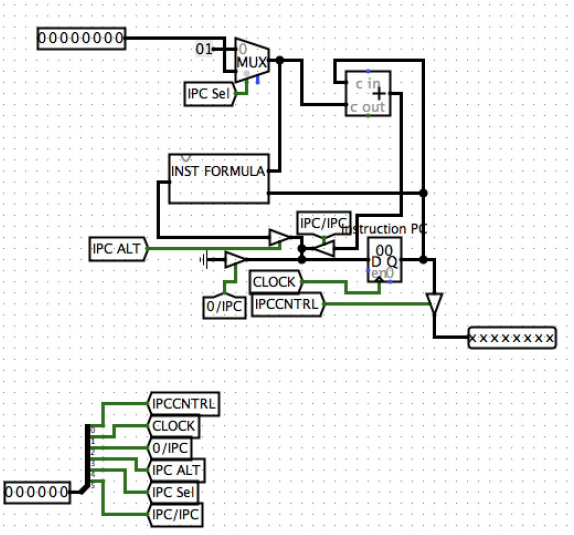
The output of the Mux should be sent to three ROM chips (as LogiSim has a 32 bit limit on a ROM chip and we have 75 control lines. There should be two 8x32 bit ROMs and 1 8x16 bit ROM.

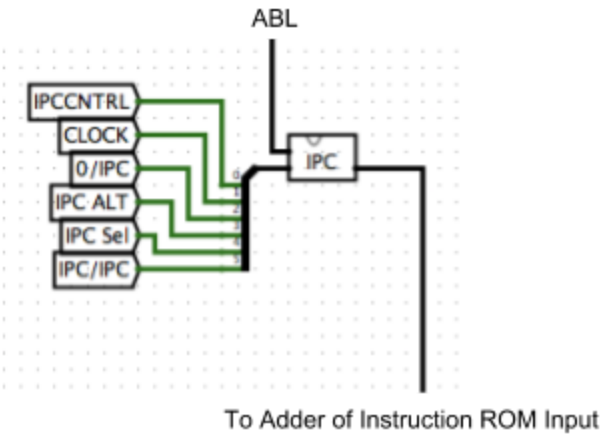
The output of each ROM should be sent into a register of the same size, all of which should be controlled by the inverse clock.

Step 5: IPC

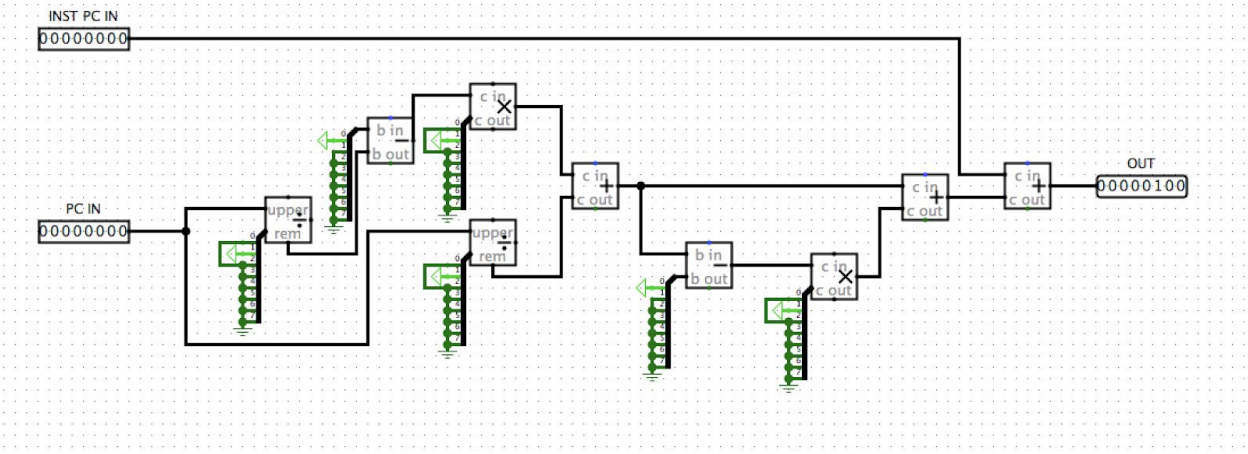
The Instruction PC is the program counter that takes control once an opcode is initiated. It should be built as a subcircuit.

This took a long time to develop and is very complex, the most complex part to the processor, and it is not integral to the 6502 itself, but how we have simplified the Random Control Logic, so this will be completely given to you.





Instruction Formula:



Step 6: Control Lines

Essentially the control lines are the output of the Instruction ROMs inputted into the control lines input of the Infrastructure. However you organized your inputs (and the order that you put them in) put them in the same order and wire them into the Infrastructure. The way that they should be ordered is as follows…

Unused	79	IPCCNTRL	54	0/V	29	SP/B	4
Unused	78	IPC sel	53	1/V	28	SP/B2	3
Unused	77	PROMCNTRL	52	AVR/V	27	A/B	2
Unused	76	N/DB	51	DB6/V	26	B/A	1
Unused	75	V/SB	50	DBZ/Z	25	A/B2	0
H/PROM	74	Z/DB	49	DBI/Z	24		
D/HRH	73	C/DB	48	ACR/C	23		
D/HRL	72	SB/PCL	47	IR5/C	22		
IPC ALT	71	IDL/ADH	46	DB0/C	21		
D/IDL	70	IDL/ADL	45	P/DB	20		
ADRCTL	69	IDL/DB	44	OUT	19		
ITRCTL	68	ADL/ABL	43	ALU Mux	18		
PROM/AROM Disable	67	ADH/ABH	42	ALU Mux	17		
LDRAM	66	SB/ADH	41	ALU Mux	16		
CIN	65	ADD/ADL	40	DB/SB	15		
IDL 7	64	SP/ADL	39	SB/DB	14		
IDL 6	63	PCH/ADH	38	DB/SP	13		
IDL 5	62	PCL/ADL	37	ADL/Add	12		
IDL 4	61	PCH/DB	36	B2/Add	11		
IDL 3	60	PCL/DB	35	B/Add	10		
IDL 2	59	PCH/PCH	34	Add/B2	9		
IDL 1	58	PCL/PCL	33	Y/B2	8		
IDL 0	57	ADH/PC	32	B2/Y	7		
0/IPC	56	ADL/PC	31	X/B2	6		
IPC/IPC	55	DB7/N	30	B2/X	5		

The only last thing is to OR the IDL input from the control lines with D controlled by a controlled buffer (the control line being D/IDL). Also, pull D/IDL to 0 using a pull resistor after the controlled buffer just before being placed into the OR gate.

Then the Processor Is Complete.

Copy and paste the values from the spreadsheet (given to you) for each ROM chip and start programing!

Milestone 7

More OPCODES!

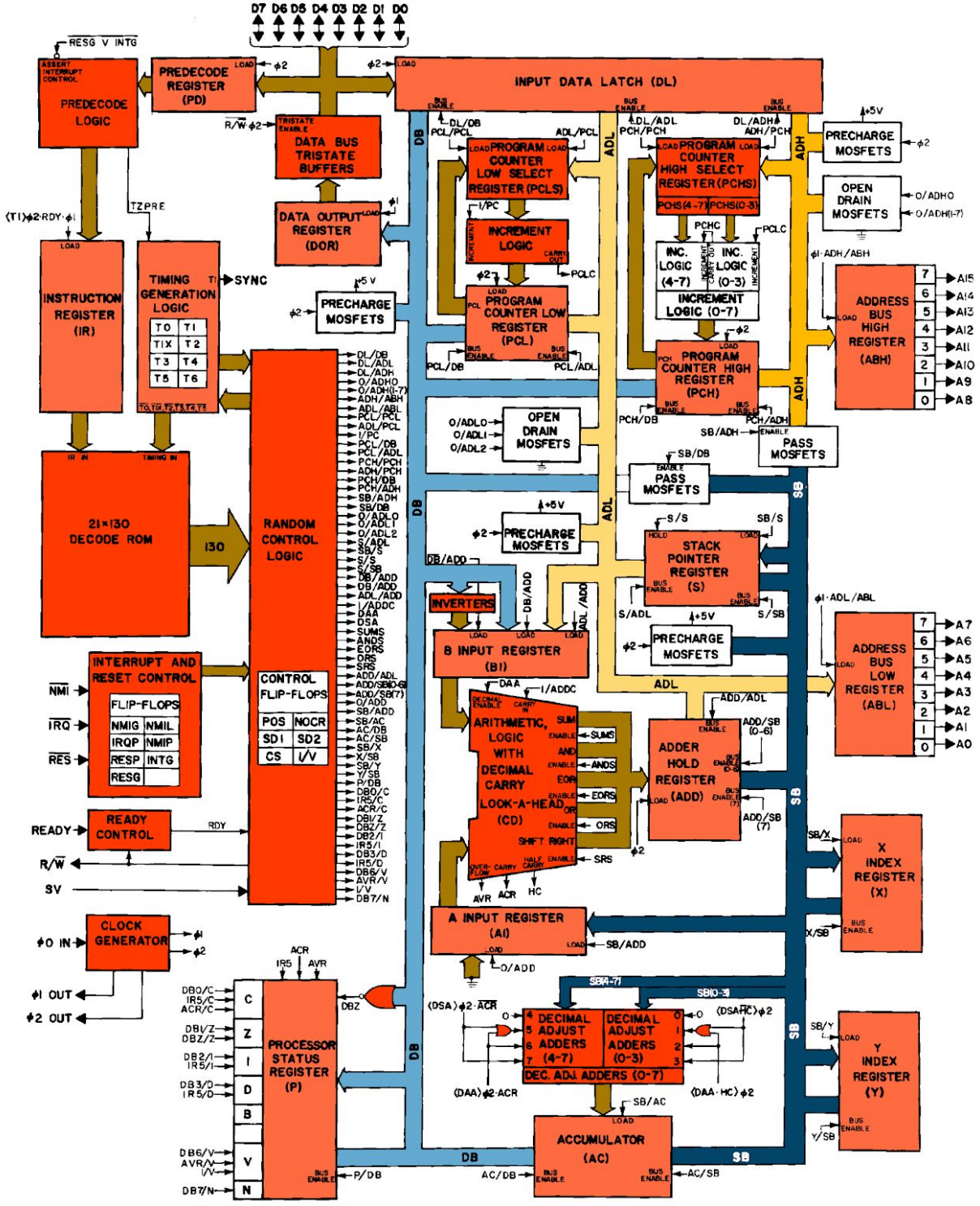
The processor is finished and some basic opcodes have been developed, but it is up to you to program more. In order to run complex programs, we need to program the control lines for each opcode.

The basic strategy here is to think about how to use what the processor has built in to run an opcode. Don’t worry about how many lines you have, just program each line (which is run with each tick of the clock). Good luck!

Remember to submit pull requests to the git repo with the opcodes you develop so that the programs can get more complex the further on people get. This is a group effort to iterate upon this curriculum!

Congratulations on building the 6502 Processor Subset!

Hanson's Block Diagram Colorized



Key

ABH **Address Bus Register (High)**, register that takes data from ADH and outputs it to RAM.

ABL **Address Bus Register (Low)**, register that takes data from ADL and outputs it to RAM.

ADH **Address Bus (High)**, the most significant (largest) 8 bits of the 16 bit address bus, (see also, ADL).

ADL **Address Bus (Low)**, the least significant (smallest) 8 bits of the 16 bit address bus, (see also, ADH).

ALU **Arithmetic Logic Unit**, a small circuit at the heart of a processor that performs mathematical operations.

IDL **Input Data Latch**, a register used to control data flow for DB, ADH, and ADL.

PC **Program Counter**, the register that controls the location of execution in RAM

PCL **Program Counter Low**, the Lower 8 Bits of PC

PCH **Program Counter High**, the Higher 8 bits of PC

IR **Instruction Register**, a 16 bit register that holds the data being sent to the RAM

D **D Bus**, the bus that holds the output (D) of the RAM in the processor interface.

IPC **Instruction Program Counter**, the program counter that is handed control at the start of each OPCODE execution.