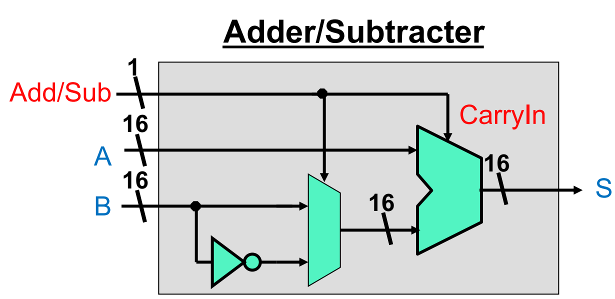
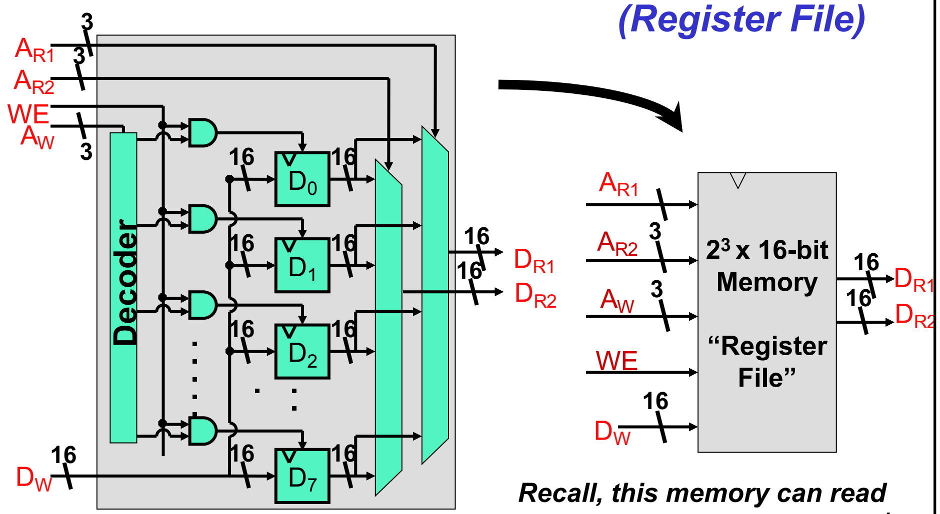
EXAMPLE: BASIC CPU DATAPATH & CONTROL

16-bit Adder/Subtracter *(simplified ALU)*



- A Basic Arithmetic Logic Unit (ALU) allows us to do two operations + / - on 16 bit values.
- Inputs: A, B, Add/Sub
- Outputs: S
- *Data* inputs: A & B, "*Control*" inputs: Add/Sub

2^3 by 16-bit Memory: Two Read Ports, One Write *(Register File)*



Recall, this memory can read 2 addresses and write to a 3rd on the positive edge of CLK

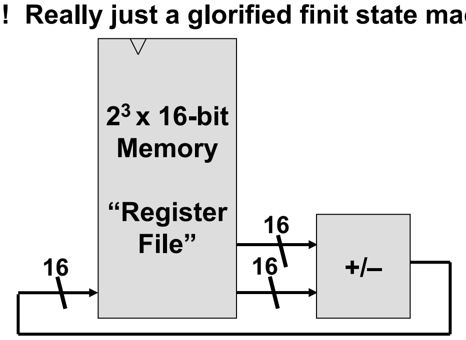
Address Space, k= 2^3 = 8, or 8 drawers
Addressability, m=16 bits, or 2 bytes in each drawer

3

Simple Processor: Datapath

Putting Register File together with ALU:

- **Our first CPU! Really just a glorified finit state machine!**

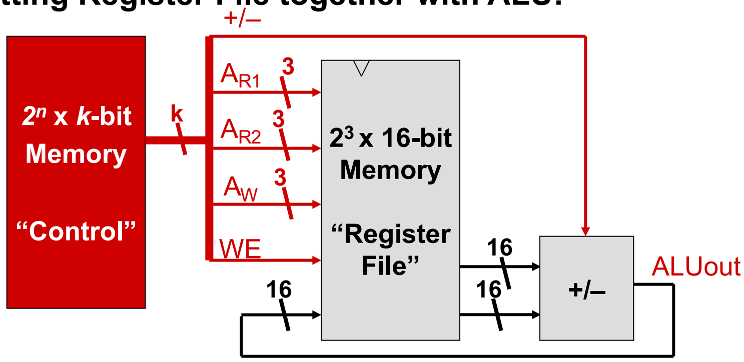


- This is the "datapath"
  - Literally, the path through which data values travel
- What controls the flow of data through this datapath?

4

Simple Processor: Datapath w/Control

Putting Register File together with ALU:



These are the "control" signals *(The lines in red)*

- **The signals needed to control the flow of data along the datapath**

Notice, we added a second "Memory"

This memory will hold values for the control signals

i.e.: AR1, AR2, AW, WE, +/-

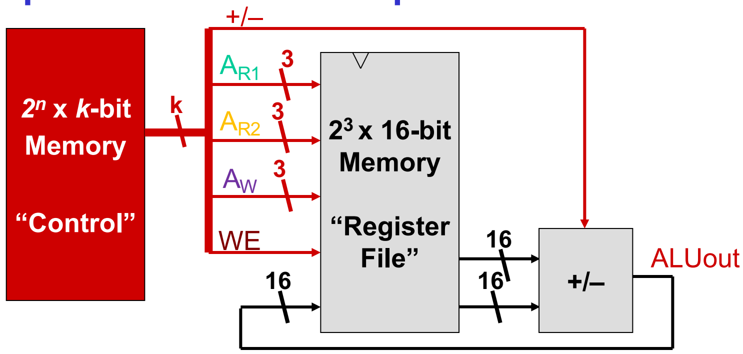
5

A Useful Analogy

- The datapath corresponds to the *tracks* in a railway
  - pathways that allow you to move information around the CPU
- The control signals control the *switches* that connect tracks
  - Signals that setup the pathways so data can flow through CPU

6

Simple Processor: Example: Add two #s



Our register file has 2^3=8 DFF Registers within it:

- Let's say we wish to add the contents of Regfiles: Reg1 to Reg2
- Then store the result in Reg3. *(akin to a program: a=b+c in Java)*

How would we set the "control" lines?

+/- = 0 (to indicate an add)

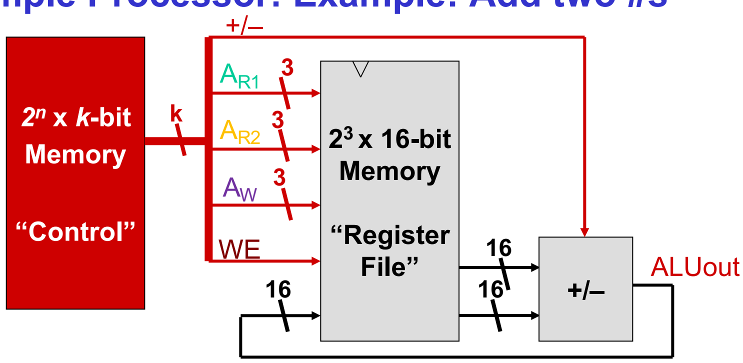
A_{R1} = 001, A_{R2} = 010, A_W = 011, WE = 1 (to store result)

The settings for the control lines, come out of the control memory:

k is then 11 bits wide (in this case) and equals: 0 001 010 011 1

7

Simple Processor: Example: Add two #s



To ADD contents of R1+R2, control memory must contain:

+/–	A_{R1}	A_{R2}	A_W	WE
0	001 (R1)	010 (R2)	011 (R3)	1

If "row0" of the control memory had these 11-bits in it

Our ALU would perform an ADD of R1+R2 and write the results to R3

8

Simple Processor: Limitation

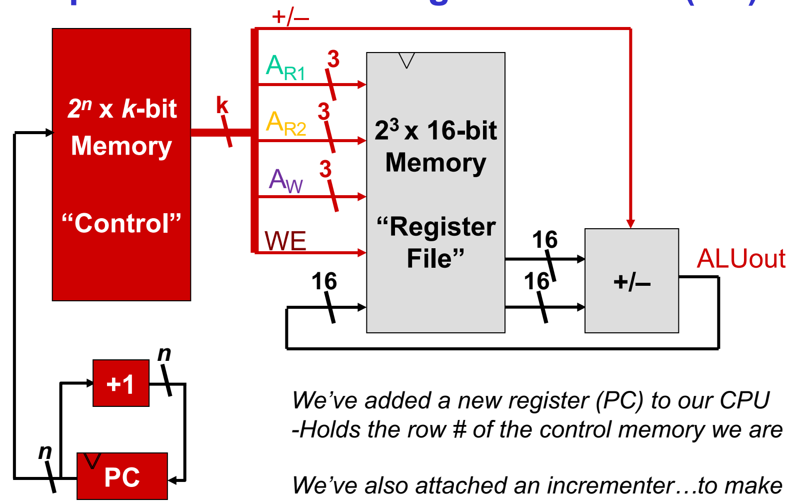
- Register File
  - Contains DATA our processor operates upon
- Control Memory
  - Holds control signals for our processor
  - In essence – holds the “program” we want our CPU to execute
- Limitation in our *Simple Processor Model…*
  - If we had a program with more than 1 instruction: a=b+c…
    - o How could we advance to the next row of control Memory?
      - – We need a device to tell us what ‘row’ of our program we are on
- In the next slide, we’ll try to fix this limitation…

9

BASIC CPU W/ PROGRAM COUNTER

10

Simple Processor w/Program Counter (PC)



Program Counter
(points to current row in control memory)

We've added a new register (PC) to our CPU
-Holds the row # of the control memory we are on

We've also attached an incrementer…to make a counter
-Advances us 1 row of control memory at each clock pulse
-PC=PC+1 (just means go to next line in control mem)

11

Simple Processor w/PC - Limitations

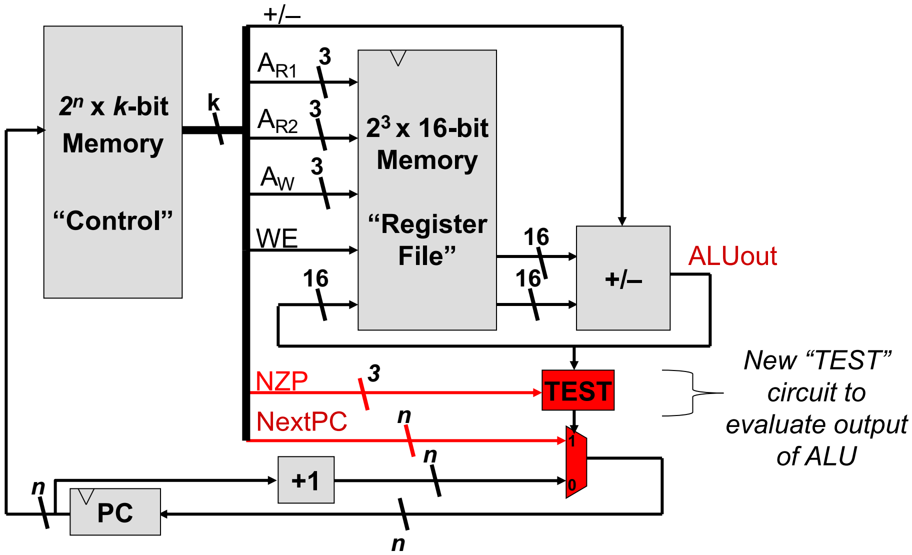
- Register File
  - Contains DATA our processor operates upon
- Control Memory
  - Holds control signals for our processor
  - In essence – holds the "program" we want our CPU to execute
- PC
  - Holds "state" of our system
  - Essentially tells us what row of control memory to lookup
- Limitation in our *Simple Processor w/PC Model…*
  - This system is great if our programs run one line after another
  - But what if we don't want to execute the program in order?
    - Can we add hardware to enable IF/THEN capability?
    - Want hardware that allows us to jump around in our program

12

ENHANCED CPU IF/THEN/WHILE

13

Enhanced Processor /w PC & Tester Circuit

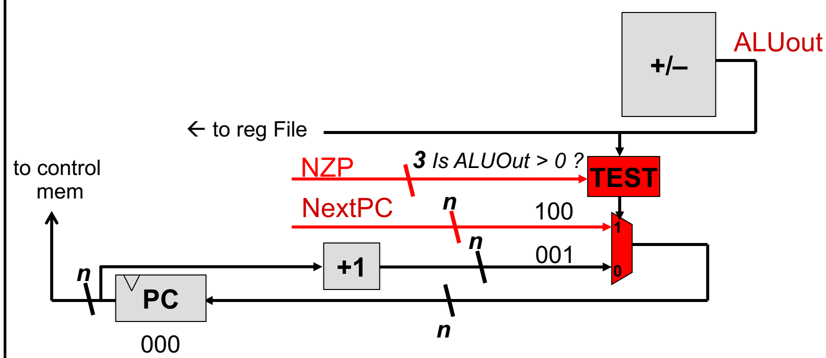


Also, two new control lines: NZP and NextPC, controls TEST box

14

What does the TEST circuit do?

- Ultimately…control the next value for the Program Counter
  - Tests output of ALU, for some condition: NZP
  - If condition is TRUE, PC=NextPC (jump to another line like: 100)
  - If condition is FALSE, PC=PC+1 (go to next line: 000+001=001)



15

Why do we want a TEST circuit?

- Gives CPU ability to make decisions at runtime
  - Skip over instructions in control memory
  - Loop / repeat instructions in control memory
  - Examples:

```
if (a>b) {
    // do something
} else {
    // otherwise skip here
}
```

```
while (a>b) {
    // repeat these lines
}
```

  - Values of variables (a,b) may not be known until program is running

16

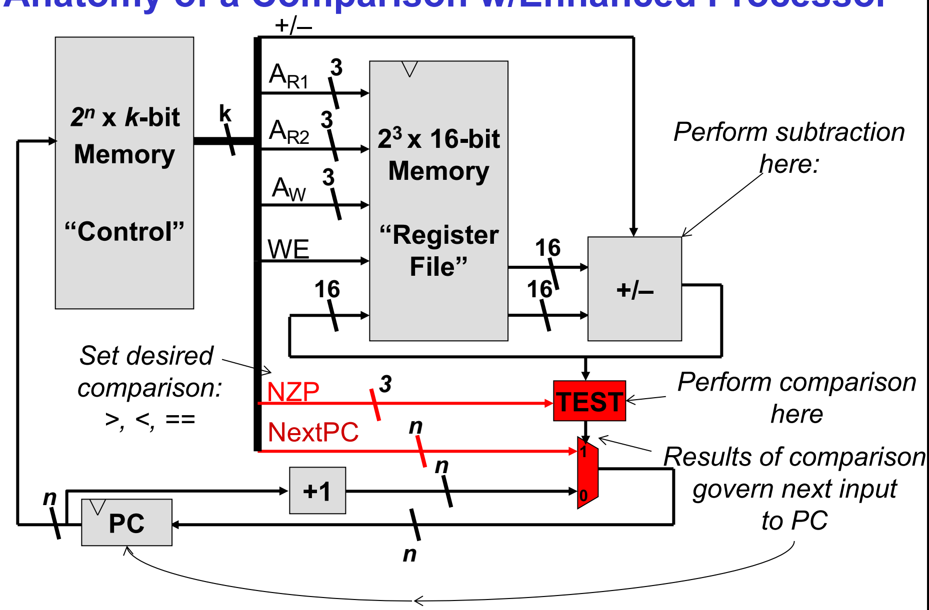
How do we Implement Comparisons?

```
while (a>b) {
    // repeat these lines
}
```

- We use <u>ALU / Tester circuit / MUX</u> to perform comparison…example:
- Step 1: perform subtraction: a-b *(ALU can do this)*
- Step 2: judge output of ALU: *(Tester does this)*
    - If a-b < 0, then *a* is smaller than *b*
    - If a-b = 0, then *a* is equal to *b*
    - If a-b > 0, then *a* is greater than *b*
    - Basic question for Tester: *did subtraction produce a…*
        - Negative, Zero, or Positive number (NZP)
- Step 3: Jump to a line of program based on result of comparison
    - If a>b, then PC=PC+1 *(MUX does this)*
    - If a<=b, then PC=some new value

17

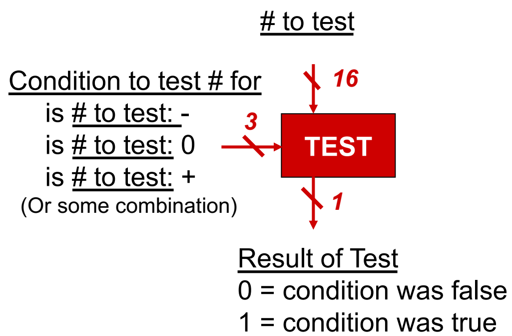
Anatomy of a Comparison w/Enhanced Processor



18

Let’s Define Behavior of Our “TEST” Circuit

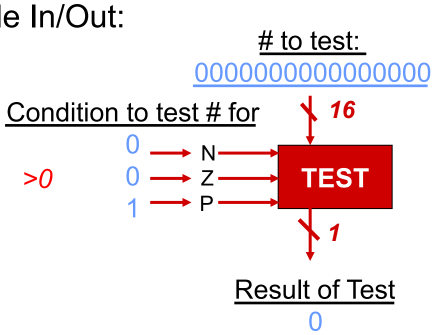
- Want a circuit that can determine if output of ALU is: +, - or 0
- *(OR MORE than 1 of those conditions: e.g.: >=0)*
- Inputs/Outputs/Behavior:
-



19

Example of “TEST” Circuit in operation

- Let’s test a 16-bit # to see if it is POSTIVE
- Let’s say ALU outputs: 0000000000000000
- Example In/Out:
-



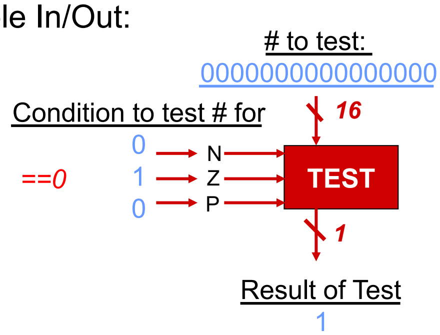
Output of 0 indicates, condition was false,
The # to test was not positive

20

Example of "TEST" Circuit in operation

- Let's test a 16-bit # to see if it is ZERO
- Let's say ALU outputs: 0000000000000000

- Example In/Out:
-



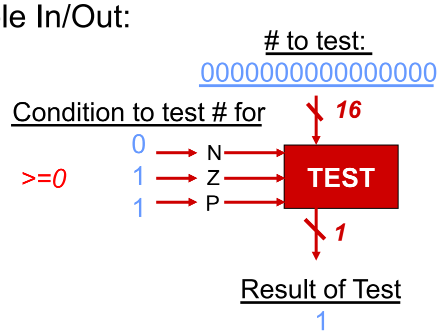
Output of 1 indicates, condition was true,
The # to test was indeed equal to 0

21

Example of "TEST" Circuit in operation

- Let's test a 16-bit # to see if it is POSITIVE (OR) ZERO
- Let's say ALU outputs: 0000000000000000

- Example In/Out:
-



Output of 1 indicates, condition was true,
The # to test was POSITIVE (OR) ZERO

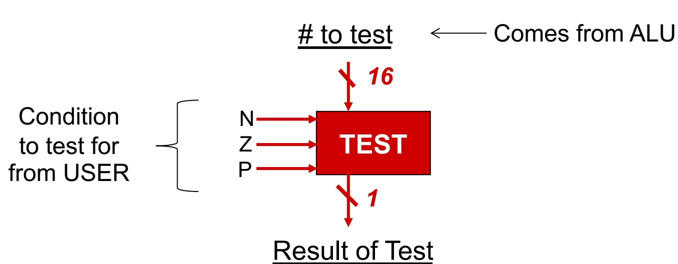
22

ENHANCED CPU
INSIDE NZP TESTER

23

Implementing the NZP TEST Component:

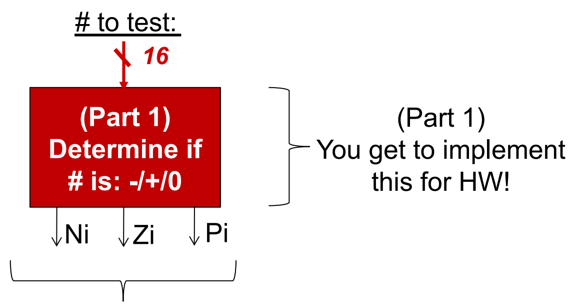
- 4 inputs:
  - # to test ← (1) 16-bit input
  - N, Z, P ← (3) 1-bit inputs Condition to test for from USER
- 1 output:
  - Result of Test ← 1-bit output 0=Condition from user is FALSE, 1 if TRUE
- Two internal parts to the TEST component
  - *1) Determine if incoming # is Negative/Zero/Positive*
  - *2) Compare incoming condition from user to output of part 1)*



24

Implementing the NZP TEST Component (Part 1):

- Two internal parts to the TEST component
  - *1) Determine if incoming # is Negative/Zero/Positive*

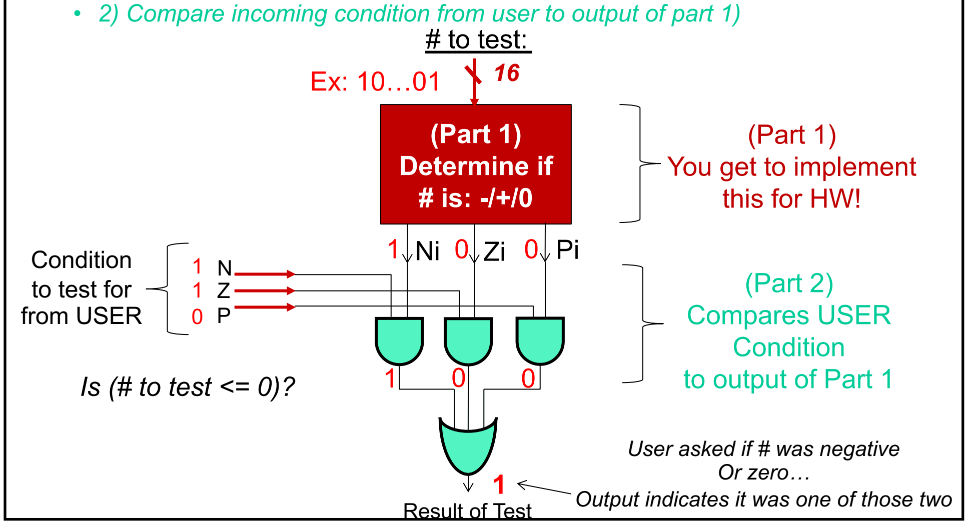


Output of PART 1:
If # is negative, Ni=1
If # is zero, Zi=1
If # is positive, Pi=1
Only 1 line can be high at a time

25

Implementing the NZP TEST Component (Part 2):

- Two internal parts to the TEST component
  - *1) Determine if incoming # is Negative/Zero/Positive*
  - *2) Compare incoming condition from user to output of part 1)*



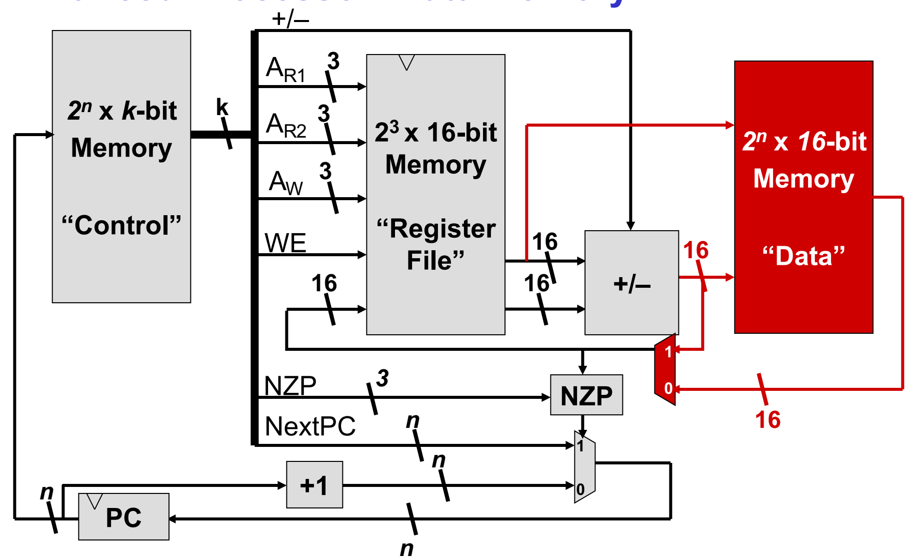
26

ENHANCED CPU
& THE VON NEUMANN MODEL

What Else Can Our CPU Do?

- Not much…
- We still need memory
  - 8 16-bit *words* is not enough to do anything interesting
  - Storage space for large data structures

Enhanced Processor: Data Memory



With "Data Memory", we can now load register file with data!

Project: Processor Design

- Design a 'mini' processor
  - Datapath and Control path
- You are given a set (subset of a real set) of instructions
  - Design ALU, Memory
  - Design datapath
  - Design control path and 'microprogram' or FSM to control the executions
- Test your logic component using Cedar Logic
  - Provide a final paper design/schematic