Frankie

A Frankenstein’s Monster of Architectures

The Accumulators

- Started out with one accumulator and a stack
- Realized that it was hard to keep track of multiple variables
- Fixed by adding second accumulator
- Named Mary and Shelley after author of Frankenstein

Why Shelley?

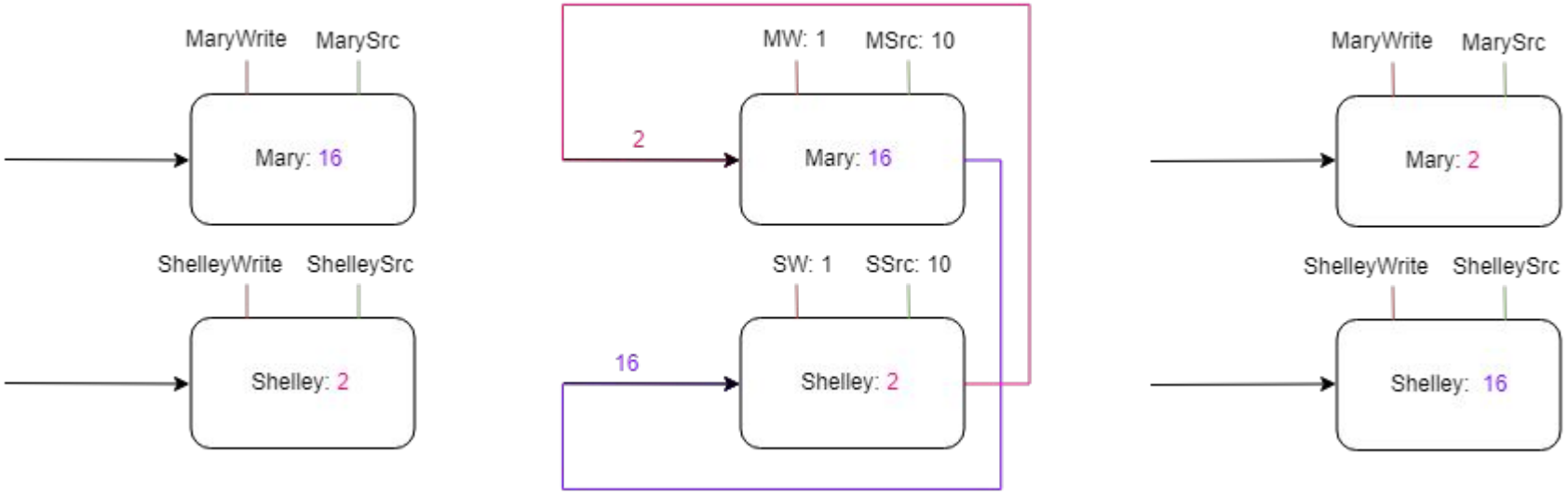
- Try this:

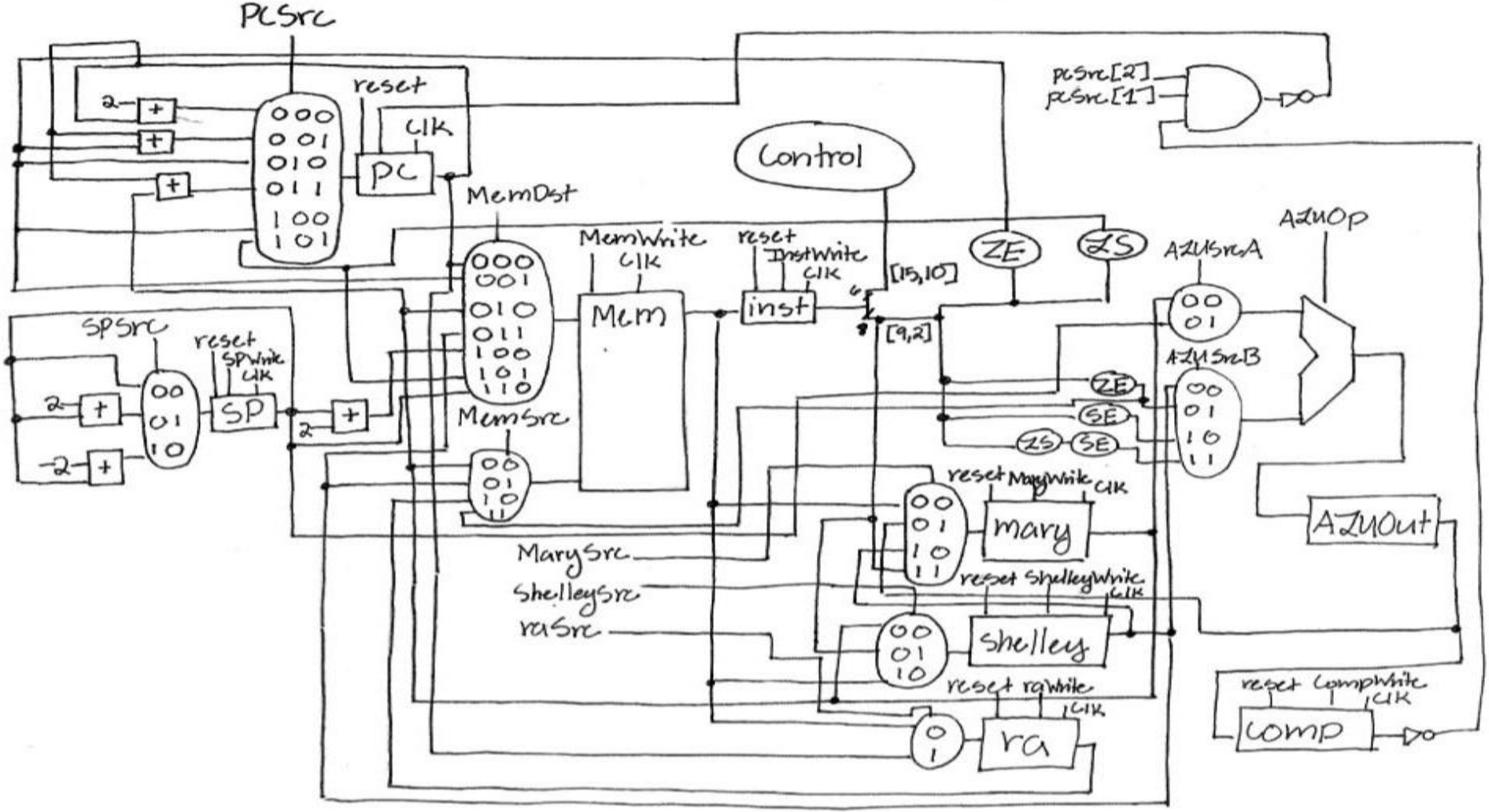
```
for(int i = 0; i < 10; i++)

    sum += i;
```

- sum goes in Mary
- Where do you keep i? On the stack? Should we add right off the stack, then?
  - This only fixes the problem if the thing you want from the stack doesn't get buried
    - you can't pop without losing Mary's contents

Two Accumulators: (Swap)




PCSrc
reset
clk
pc
Control
MemDst
MemWrite
clk
reset
InstWrite
clk
Mem
inst
[15,10]
[9,2]
ZE
ZS
ALUSrcA
ALUOp
SPSrc
reset
SPWrite
clk
SP
MemSrc
ALUSrcB
ZE
SE
ZS
SE
reset MaryWrite clk
mary
ALUOut
MarySrc
ShelleySrc
raSrc
reset ShelleyWrite clk
shelley
reset raWrite clk
ra
reset CompWrite clk
comp

Instruction Set

- Instructions are grouped together into common operation groups
  - Arithmetic/Logical
  - Stack
  - Jump
  - Compare
  - Memory-based
  - Backup

flag	op code	immediate	unused
1	5	8	2

Flagbit

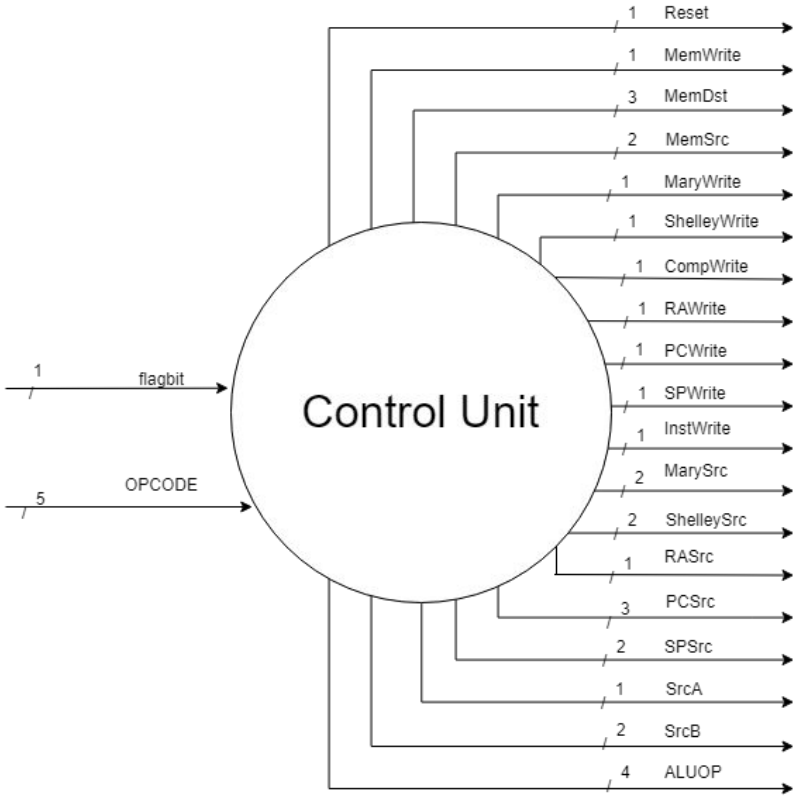
- The flagbit provides an alternate form of most instructions
  - Solution to duplication of instructions w/ two accumulators
- Indicated in assembly code using the ‘@’ symbol
- Flagbit 0: Operation is performed on Mary and an immediate
- Flagbit 1: Operation is performed on Mary and Shelley
- Example: aadd instruction
  - aadd 5 //adds 5 to the value in Mary
  - aadd@ //adds the value in Shelley to the value in Mary

Assembler

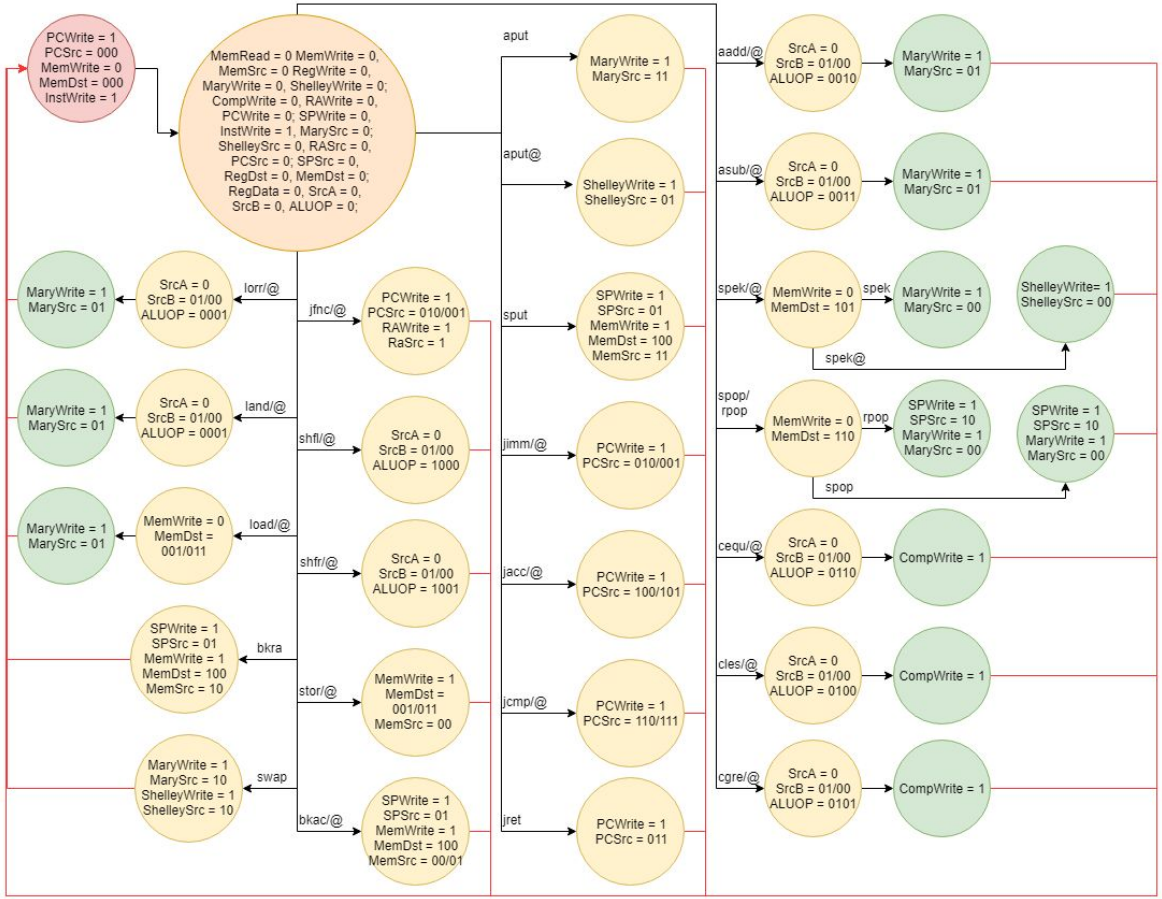
- Written in python
- Supports the entirety of our instruction set
- Auto-injects our kernel into the memory space for interrupts
- Can output debug data during assembling step to ease debugging

Control Unit

- For regular functionality...
  - Controls everything except I/O
  - 2 inputs (OPCODE, flagbit)
  - 19 outputs
- States:
  - 4 general states:
    - Fetch-- fetches instruction
    - Decode-- what OPCODE, flagbit?
    - 3-- determined by OPCODE and flagbit
    - 4-- determined by OPCODE and flagbit



Finite State Machine:

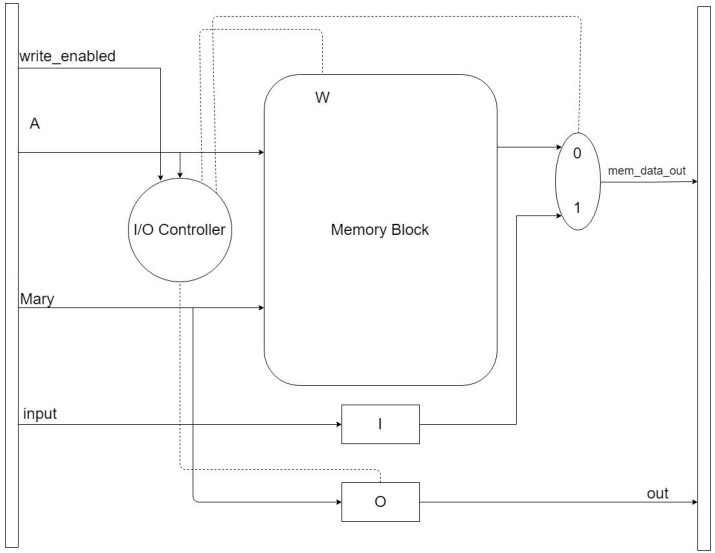


Memory

- We didn’t use the Xilinx block memory generator
- Didn’t have to use .coe files; used .txt files instead
- Verilog provides a command to read binary data (instructions) from files
    - $readmemb(“file.txt”, memory);

Memory-Mapped I/O:

- Controlled via a “sub-control unit” inside our memory block
- Uses memory register 255
  - `stor 255 //store value from Mary into output register`
  - `load 255 //load value from input register into Mary`



Interrupts

- Frankie provides support for input via interrupts
- Interrupt is triggered when the value in the input register changes
- Control unit handles the interrupt by jumping to the kernel
- Program returns from kernel upon reaching address 255 in memory

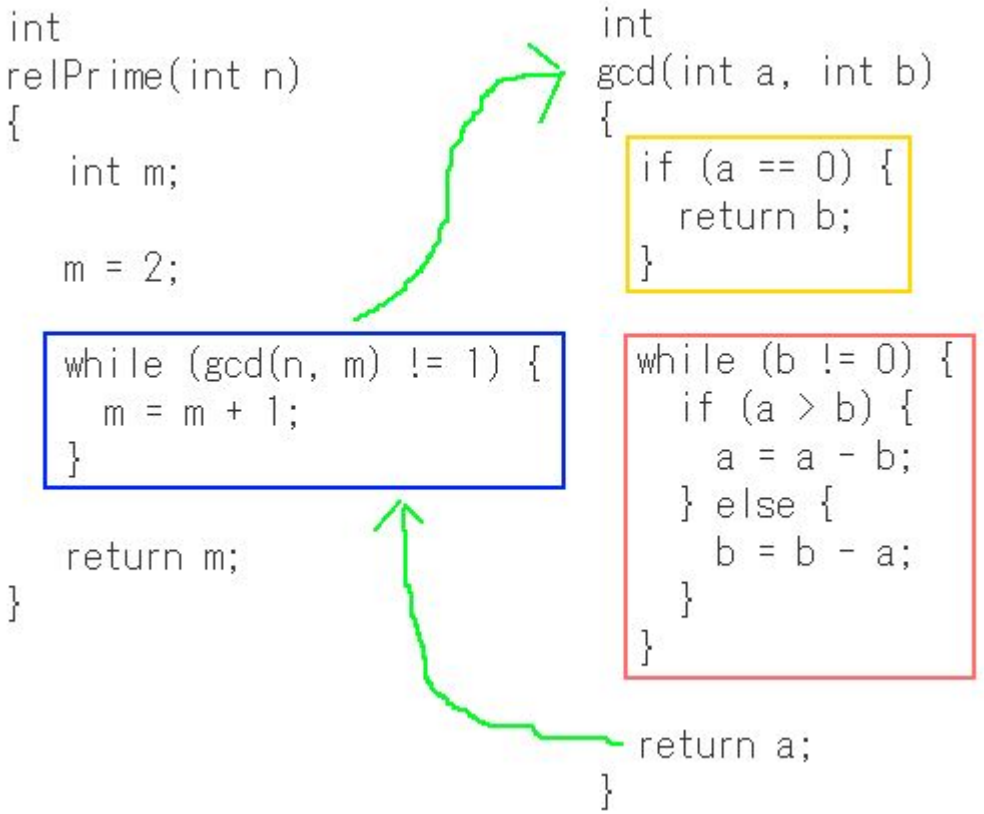
OUR KERNEL

load 255

Euclid’s

```
prep:
    aput@ 2 //int m; m = 2;
    bkac@ //back up shelley's initial value
relprime_loop:
    jfnc 6 //gcd
    cequ 1 //if (gcd(n, m) == 1
    jcmp@ 23 //relprime_end
    spop //restore shelley's initial value
    aadd 1 //m=m+1
    bkac //back up shelley
    swap //put a in front (prep args)
gcd:
    cequ 0 //if (a == 0)
    jimm@ 2
    swap //prep b for return
    jret //return b;
    jcmp@ -3
    bkac //back up mary
gcd_loop:
    jcmp@ 10 //break
    cgre@ //if a > b
    jcmp@ 5 //jump past a=a-b to b=b-a (gcd_2)
    swap
    asub@ //else b=b-a
    cequ 0 //if b == 0
    swap
    jimm@ -8 //gcd_loop
gcd_2
    asub@ //a=a-b
    cequ 0 //if b == 0
    jimm@ -11 //gcd_loop
gcd_end:
    spop@ //pop backup of mary back into mary
    jret //return a
relprime_end:
    spop //put m into mary to prep for return


kernel:
    load 255 //load input into mary
```



Performance Data

- Memory space: 62 bytes
- Instruction count: 61286
- Cycle count: 214466
- CPI: 3.4994
- Cycle time: 9.247 ns = 108.143 MHz
- Runtime for Euclid’s, n=5040:

2.2634 ms

Future Modifications

- Exception handling with interrupts
- Able to handle larger programs
- More memory
- FPGA board
- Even faster!