Instruction Syntax:

1. load `ld  [dest_reg]`
    1. Always loads from the address at r1 into `dest_reg`.
2. store `str [source_reg] [dest_addr]`
    1. Always stores into the the address at r1 from `dest_reg`.
3. logical shift left `lsl [dest_reg] [src_reg]`
    1. Shifts in 0's to destination
    2. `dest_reg` can be any reg
    3. `src_reg` can be (half)
    4. Shift amount is always grabbed from `r8`
4. logical shift right `lsr [dest_reg] [src_reg] shamt`
    1. Shifts in 0's to destination
    2. `dest_reg` can be any reg
    3. `src_reg` can be (half)
    4. Shift amount is always grabbed from `r8`
5. bitwise or `or  [dest_reg] [other_reg]`
    1. Bitwise or `dest_reg` w/ `other_reg` and store in `dest_reg`
    2. `dest_reg` can be any reg
    3. `other_reg` can be `r[4-7]`
    4. TODO: Make sure all `or` instructions only use `r[4-7]` in program2
6. bitwise xor `xor [dest_reg]`
    1. Bitwise xor `dest_reg` and the reg number store in `r8` and store in `dest_reg`
    2. TODO: change all xor instructions to mov + xor
7. reduction xor `rxr [dest_reg]`
8. add immediate `add [dest_reg] immi2`
    1. TODO: change all adds to have <3 as immi. Can use mov and shift to get bigger numbers
9. move immediate `mov immi5`
    2. Always moves into `r8`
10. jump if equal `je  [1 or 2 or 3]`
    1. jumps to the address stored into one of the 3 dedicated saved PC reg
    2. TODO: double check all jumps to make sure the are correct je/jne
11. jump not equal `jne [1 or 2 or 3]`
    1. jumps to the address stored into one of the 3 dedicated saved PC reg
    2. TODO: double check all jumps to make sure the are correct je/jne
12. store PC `spc [1 or 2 or 3] offset`
    1. if `offset == 0` stores current address into one of 3 dedicated saved PC reg
    2. otherwise, check `r8` for the relative offset amount
    3. TODO: fix all offsets that are >0 to use `r8`
13. Lookup Table `lut [input_reg] [1 or 2]`
14. Counter clear `ctc [a | b | c | b_flag]`
15. Counter increment `cti [a | b | c | b_flag]`

16. Counter store `cts [a | b | c | b_flag]`
17. B_flag condition set `cbf` // sets the zero/
18. single substring `sbs [dest_reg]`
    1. stores 5-bit subtring from `r5` into top 5 bits of `dest_reg` based on value of `r3` to determine which 5-bit substring to grab
19. double substring `dbs [dest_reg]`
    1. stores 5-bit subtring that spans across `r5` and `r6` into top 5 bits of `dest_reg` based on value of `r3` to determine which 5-bit substring to grab
20. Copy reg `cpy [dest_reg]`
    1. Copies value of `r8` into `[dest_reg]`

Instruction Formats

1. ld `[4'b opcode | 3'b reg encode]` -> 7 bits
2. str `[4'b opcode | 3'b reg encode]` -> 7 bits
3. lsl `[4'b opcode | 3'b reg encode | 2'b reg encode]` -> 9 bits
4. lsr `[4'b opcode | 3'b reg encode | 2'b reg encode]` -> 9 bits
5. or `[4'b opcode | 3'b reg encode | 2'b reg encode]` -> 9 bits
6. xor `[4'b opcode | 3'b reg encode ]` -> 7 bits
7. rxr `[4'b opcode | 3'b reg encode]` -> 7 bits
8. add `[4'b opcode | 3'b reg encode | immi2]` -> 9 bits
9. mov `[4'b opcode | immi5]` -> 9 bits
10. je `[4'b opcode | 3'b choice]` -> 7 bits
11. jne `[4'b opcode | 3'b choice]` -> 7 bits
12. spc `[4'b opcode | 2'b reg choice | offset1]` -> 7 bits
13. lut `[4'b opcode | 3'b reg encode | 1'b choice]` -> 8 bits
14. ctc `[4'b opcode | 2'b ctr op | 2'b reg choice]` -> 8 bits
15. cti `[4'b opcode | 2'b ctr op | 2'b reg choice]` -> 8 bits
16. cts `[4'b opcode | 2'b ctr op | 2'b reg choice]` -> 8 bits
17. cbf `[4'b opcode | 2'b ctr op]` -> 6 bits
18. sbs `[4'b opcode | 3'b reg encode]` -> 7 bits
19. dbs `[4'b opcode | 3'b reg encode]` -> 7 bits
20. cpy `[4'b opcode | 3'b reg encode]` -> 7 bits

THINGS TO KEEP TRACK OF:

- r1 is always a loop counter + what we use to address memory
- load shift amount into register 8 before actual shift
- can use shift to copy values between registers other than 8, always use cpy to copy out of reg 8
- ors only use registers 4, 5, 6, 7 -> rewrite program 2 to overwrite given LSW/MSW with our LSW/MSW
- when xor'ing, use reg 8 to mark which other reg to use as a comparison
- can add small amounts, otherwise use shifts to create large numbers
- specify 0 offset for non-if statement spc moves, otherwise load offset amount in reg 8

cto (count op) 4 bit opcode, 3 bit more-specific op choice, 2 bits remaining to pick what to operate on equal flag for jump ops

Registers needed:

Program 1:

1. i `r1`
2. LSW `r2`
3. MSW `r3`
4. tLSW `r4`
5. rLSW `r5`
6. tmp
7. tmp2
8. parity `r6`
9. saved PC

Program 2:

1. loop counter (i)
2. LSW
3. MSW
4. tmp for calculating parity bits
5. tmp2
6. parity
7. new parity reg for comparison(nParity)
8. saved PC
9. mflip
10. lflip

Program 3:

1. loop counter (i)
2. loop counter (j)
3. LSW
4. MSW
5. act
6. bct
7. cct
8. bflag
9. saved PC1
10. saved PC2