Components of Lab 1 report:

0. Team.
List the names of all members of your team, but only one copy of the report should be submitted.

Christopher Yeh

1. Introduction.
This should include the name of your architecture, overall philosophy, specific goals strived for and achieved.

PNP, Powerful Node Pipelined Architecture. Overall philosophy is to complete encoding operations with as few bit instructions as possible for efficiency.

2. Instruction formats.
List all formats and an example of each. (ARM has R, I, and B type instructions, for example.)

Only one format, a 3 bit op code, and 6 bits for optional three registers, r0-r4, which are two bits each

3. Operations.
List all instructions supported and their opcodes/formats.

Instruction Name	**Op-code**	**Description**
ldr	000	Load register
str	001	Store register
bne	010	Branch not equal
cmp	011	Compare
eor	100	Logical exclusive OR
lsl	101	Logical shift left
lsr	110	Logical shift right
add	111	Add

4. Internal operands.
How many registers are supported? Is there anything special about any of the registers, or all of them general purpose?

Up to three registers are supported per instruction. All of them are general purpose.

5. Control flow (branches).
What types of branches are supported? How are the target addresses calculated? What is the maximum branch distance supported?

Branch if not equal, calculated with a look up table. There will be limitations on distance with a look up table.

6. Addressing modes.
What memory addressing modes are supported, e.g. direct, indirect? How are addresses calculated? Give examples.

Direct, calculated by supplying memory address or register

ldr r1, r2
ldr r1, 0x80

7. Can you classify your machine in any of the classical ways (e.g., stack machine, accumulator, register, load-store)? If so, which? If not, devise a name for your class of machine.

Register

8. Give an example of an "assembly language" instruction in your machine, then translate it into machine code.

add r0, r2, r3

111001011

For 9-11, give assembly instructions. Make sure your assembly format is either very obvious or well described, and that the code is well commented. If you also want to include machine code, the effort will not be wasted, since you will need it later. We shall not correct/grade the machine code. State any assumptions you make. If you need initial nonzero values in registers or memory, you need to put them there. (Exception -- the test bench will load the incoming operands for you.)

9. Program 1 (forward error correction block coder/transmitter): Given a series of fifteen 11-bit message blocks in data mem[0:29], generate the corresponding 15-bit (essentially 2-byte) encoded versions and store these in data mem[30:59]. Input and output formats are as follows:

```
input MSW = 0   0   0   0   0   b11 b10 b9
      LSW =  b8 b7 b6 b5 b4 b3   b2   b1, where bx denotes a data bit

output MSW =  b11 b10 b9 b8 b7 b6 b5 p8
        LSW =  b4   b3  b2   p4 b1 p2 p1 p0, where px denotes a parity bit
```

ModelSim ignores _, so I use them to parse the vectors for clarity.

Example, to clarify "endianness": binary data value = 10101010101
mem[1] = 00000101 -- 5 bits zero pad followed by b11:b9 = 00000_101
mem[0] = 01010101 -- lower 8 data bits b8:b1

You would generate and store:
mem[31] = 10101010 -- b11:b5, p8 = 1010101_0
mem[30] = 01011010-- b4:b2, p4, b1, p2, p1, p0 = 010_1_1_01_0

p8 = ^(b11:b5) = 0;
p4 = ^(b11:b8,b4,b3,b2) = 1;
p2 = ^(b11,b10,b7,b6,b4,b3,b1) = 0;
p1 = ^(b11,b9,b7,b5,b4,b2,b1) = 1;
p0 = ^(b11:1,p8,p4,p2,p1)

Pseudocode:

```
// Assume b1:b11 are decided via shift instructions.
Procedure encode(b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, mem[0:59]):
    P8 = b11 ^ b10 ^ b9 ^ b8 ^ b7 ^ b6 ^ b5
    P4 = b11 ^ b10 ^ b9 ^ b8 ^ b4 ^ b3 ^ b2
    P2 = b11 ^ b10 ^ b7 ^ b6 ^ b4 ^ b3 ^ b1
    P1 = b11 ^ b9 ^ b7 ^ b5 ^ b4 ^ b2 ^ b1
    P0 = b11 ^ b10 ^ b9 ^ b8 ^ b7 ^ b6 ^ b5 ^ b4 ^ b3 ^ b2 ^ b1 ^ p8 ^ p4 ^ p2 ^ p1
    mem[30] = b4 + b3 + b2  + P4 + b1 + P2 + P1 + P0 // Concatenation via shift assignments
    mem[31] = b11 + b10 + b9 + b8 + b7 + b6 + b5 + P8
```