My design is named PNP, Powerful Node Pipelined Architecture. Overall philosophy is to complete encoding operations with as few bit instructions as possible for efficiency. It only has one format

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

The operations planned were as follows:

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

With three registers supported per instruction, general purpose. I figured for program 1 we would just need to do a bunch of XORs and stores, which is what my following pseudocode did:

```
// 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
```

In hindsight it probably would have been better to have more space for registers and using beq instead of bne as the TA suggested.

I also decided to use direct addressing.

Unfortunately I could not get program 1 to run and backtraced, but everything compiles in quartus.