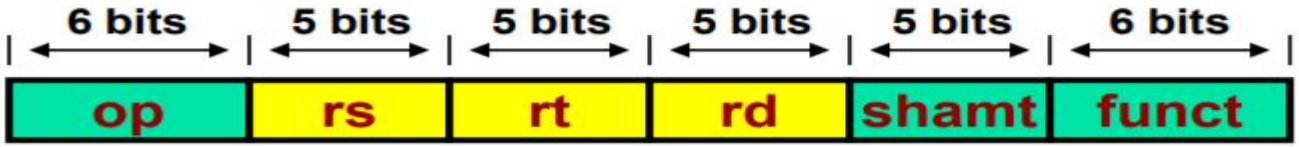
Lab 8: Pipeline Design

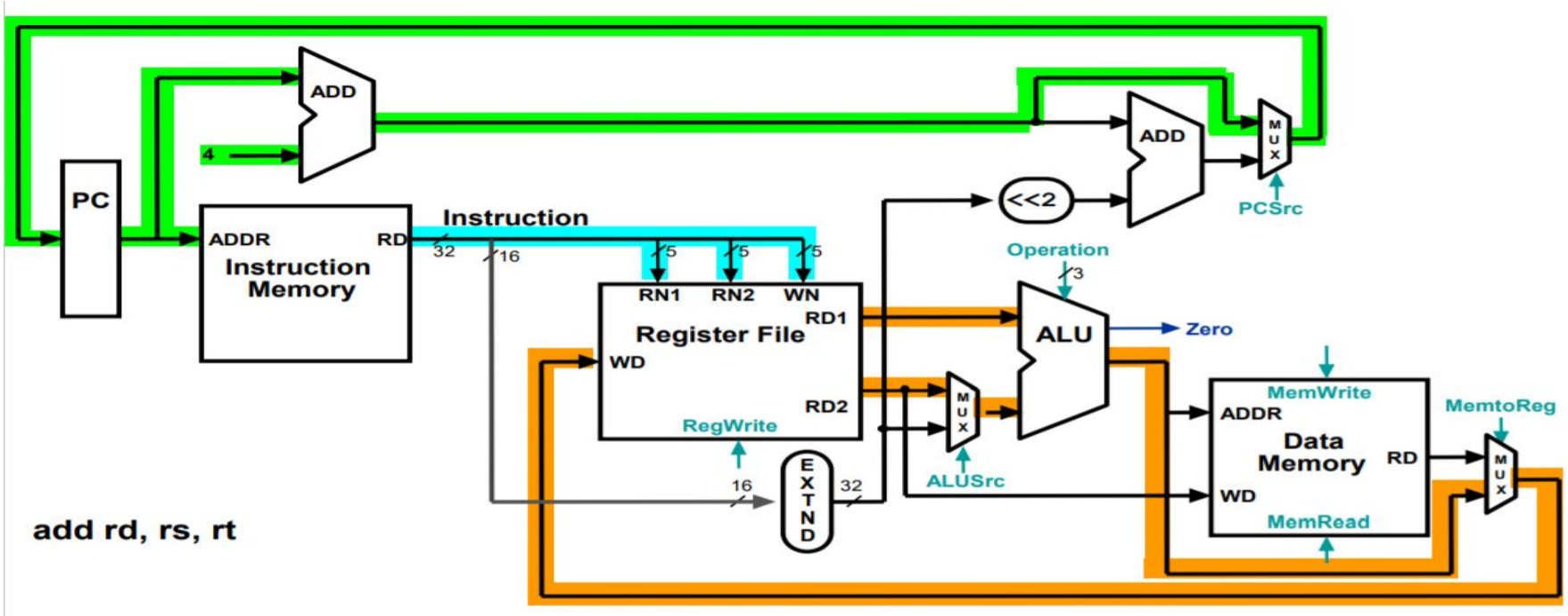
Learning outcomes:

- Modeling 3-stage pipeline design in Verilog
- Designing modules for each hardware component and some support modules occurring in the pipeline design.
- Implementation of the modules in Verilog
- Integrating these modules
- Writing test bench

What we have Done Till Now - Single Cycle DataPath



R-Format

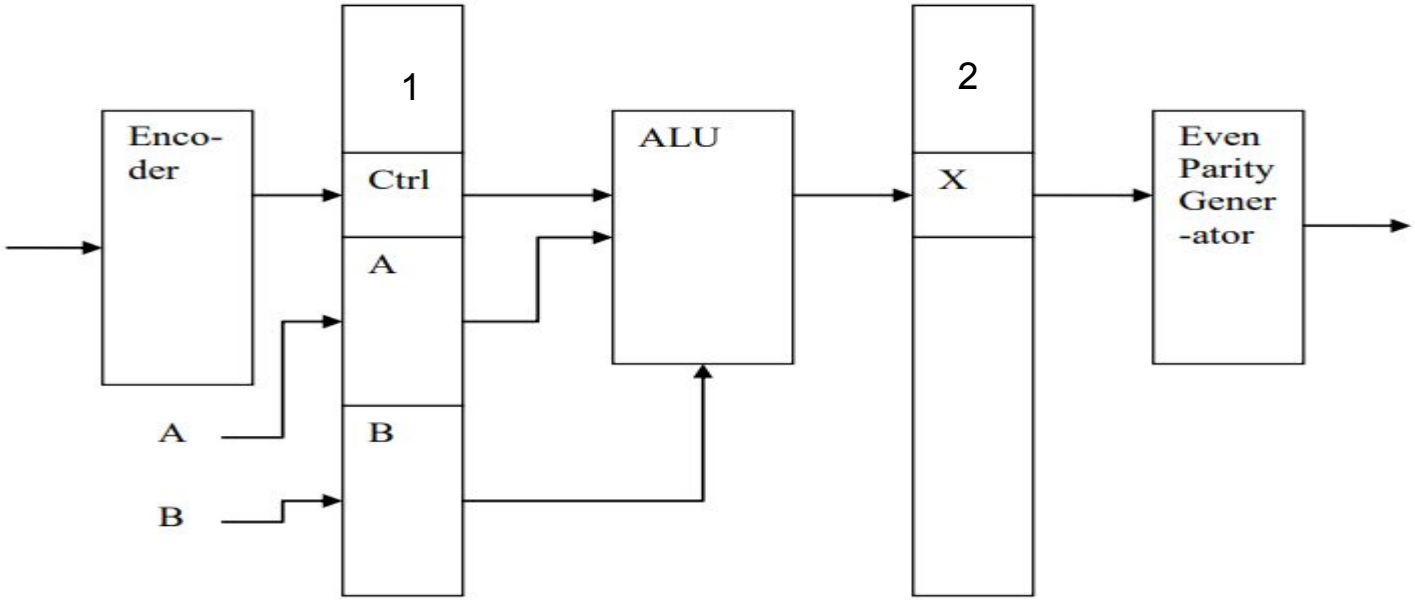


Modules we have already integrated

Instruction Memory	Lab-6
Register File	Lab-5
ALU	Lab-5
Control Unit and ALU control	Lab-5
Data Memory	Lab-6
Sign-Extender	Lab-6
Shifter	Lab-6
Adders (For Next PC address)	Lab-5
Multiplexers	Lab-5

In Lab-7, we learnt to implement control Logic.

Modelling Pipeline Design



There are 3 stages in this pipeline design: Fetch, execute and generate parity.
There are two pipeline registers present in the design.

Control Unit Design

We need 8-bit function code as input and then we will produce a 3-bit opcode as output.

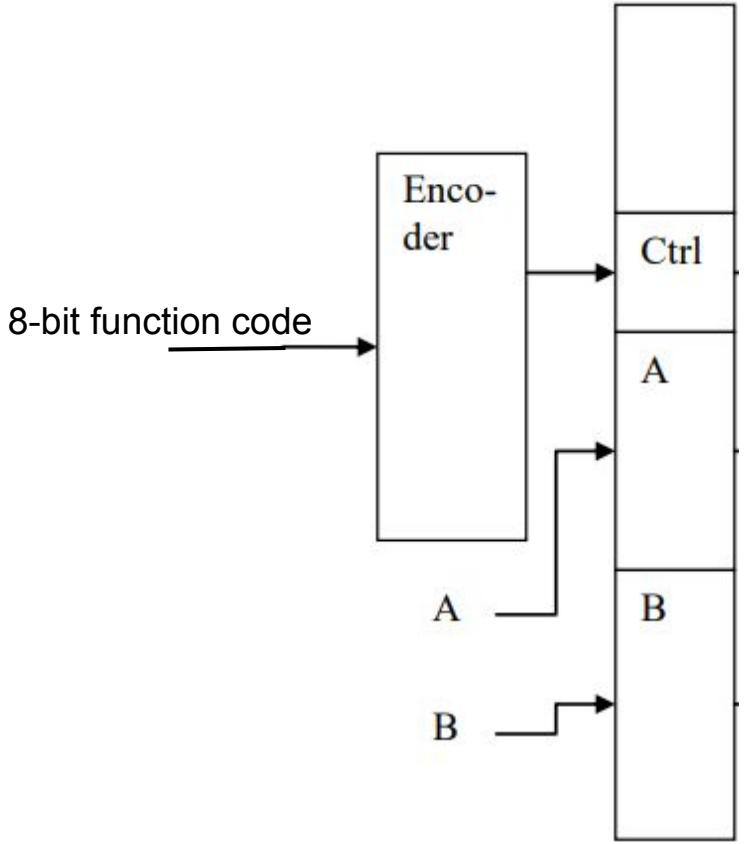
The 3-bit opcode enables us to do (2^3=) 8 operations

- ADD (3'b000)
- SUB (3'b001)
- XOR (3'b010)
- OR   (3'b011)
- AND (3'b100)
- NOR (3'b101)
- NAND (3'b110)
- XNOR (3'b111)

How will this work?

First step in the pipeline design is to FETCH.

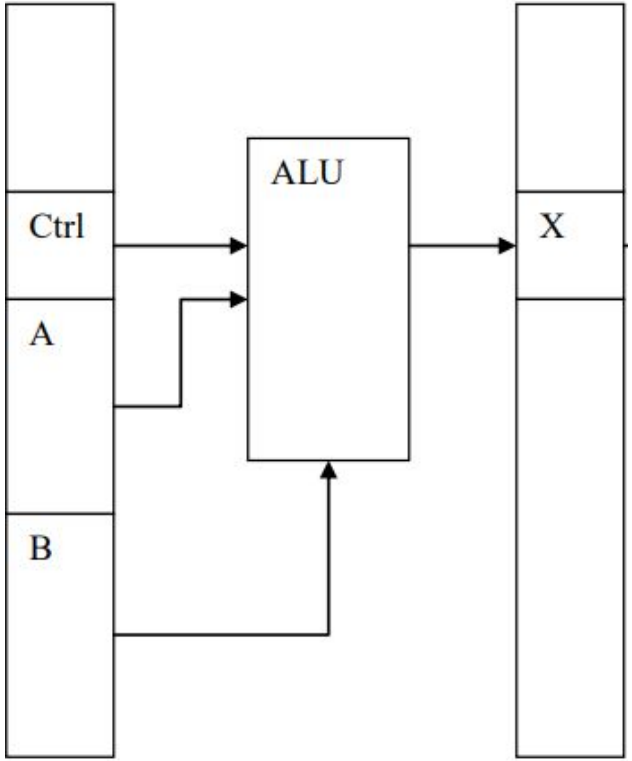
- First you will convert the ***8-bit function code to 3-bit op-code***.(use encoder)
- You will ***store operands A and B*** in the first pipeline register.



How will this work?

Second step in the pipeline design is to EXECUTE.

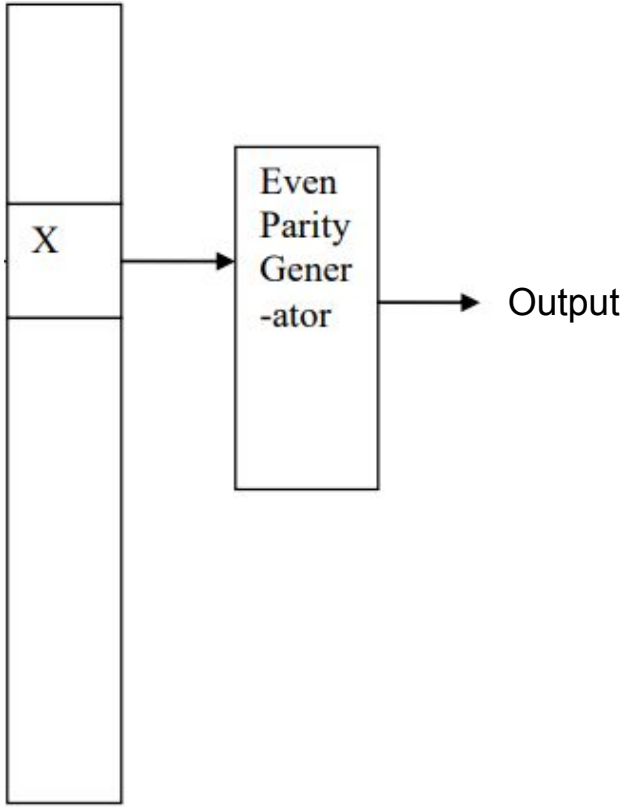
- You *read operands A and B* from the pipeline registers and pass as inputs to ALU.
- Use the 3-bit op-code to *perform the operation on ALU*.



How will this work?

Final step in the pipeline design is Generating Parity.

- *Read the output of ALU (X)* from second pipeline register.
- And *generate the even-parity*.



Components needed

- Encoder - *Behavioural modelling*
- ALU - *dataflow modelling*
- Even Parity Generator - *dataflow modelling*

Test-benches for the design.

Also, a module to integrate different modules into one.

Components needed

- Encoder - *Behavioural modelling*

Encoder takes in 8-bit function code and converts it into 3-bit op-code.

```
module encoder(funcode,opcode);
        input[7:0] funcode;
        output[2:0] opcode;

        reg[2:0] opcode;

        always@(funcode)
        begin
                case(funcode)
                        8'b00000001:opcode=3'b000;
                        8'b00000010:opcode=3'b001;
                        8'b00000100:opcode=3'b010;
                        8'b00001000:opcode=3'b011;
                        8'b00010000:opcode=3'b100;
                        8'b00100000:opcode=3'b101;
                        8'b01000000:opcode=3'b110;
                        8'b10000000:opcode=3'b111;
                endcase
        end
endmodule
```

- Even- Parity Generator - *Dataflow modelling*

```
module parityGen(aluOut,parity);
        input[3:0] aluOut;
        output parity;

        assign
parity=aluOut[0]^aluOut[1]^aluOut[2]^aluOut[3];
endmodule
```