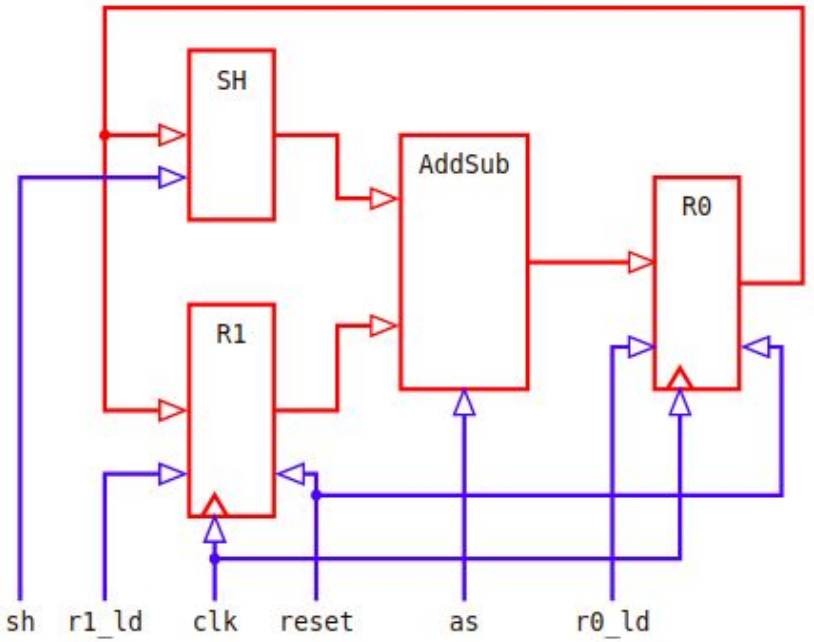
Design Of A Mini Data Path Part 1

Instructor: Dr. Vinicius Prado da Fonseca (vpradodafons@online.mun.ca)

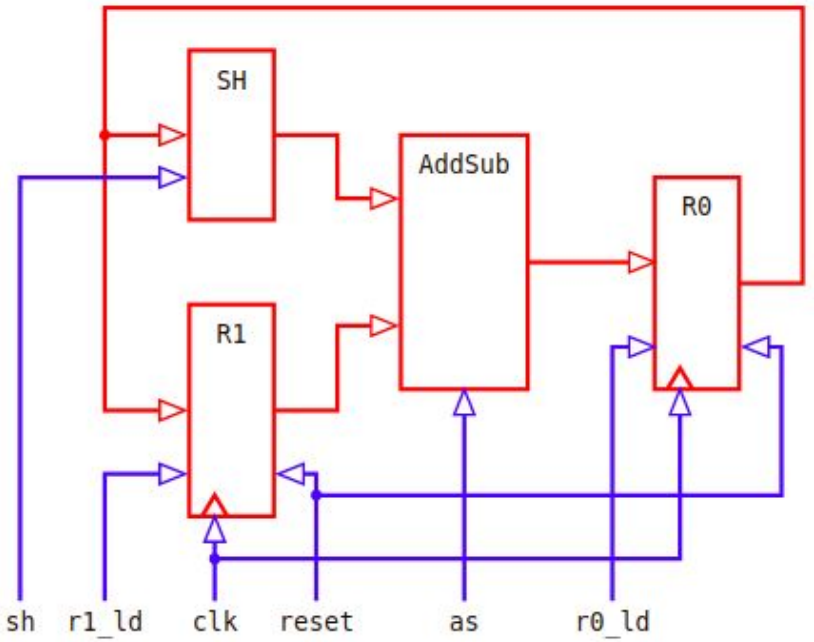
Data Paths and Control

- A data path consists of registers, adders, subtractors, shifters, and any circuit that modifies a data value
- The design of most computers consist of a data path and control logic.
- A finite state machine (FSM) is used to control the operations of the data path
    - Part 2



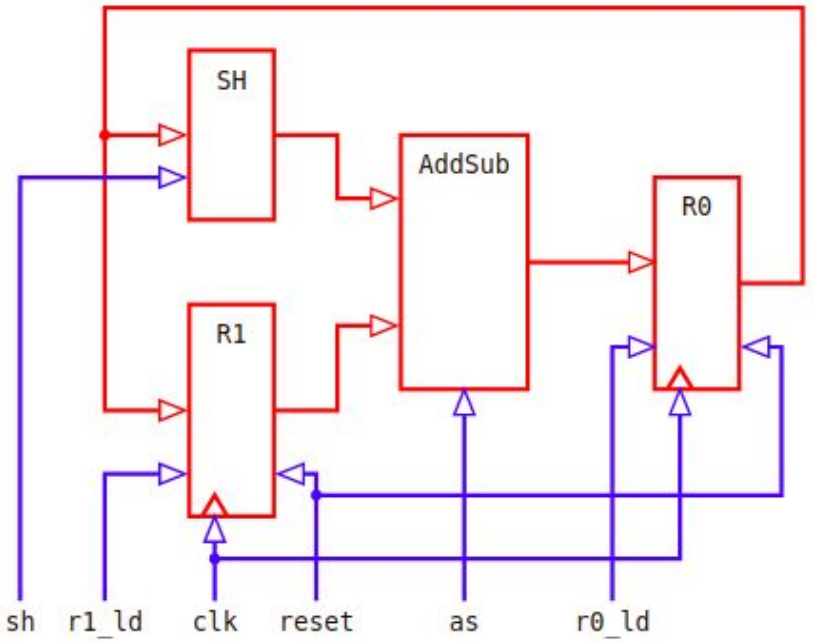
Data Paths and Control

- The blue lines are part of the control logic
- The red lines are part of the data path



Data Paths and Control

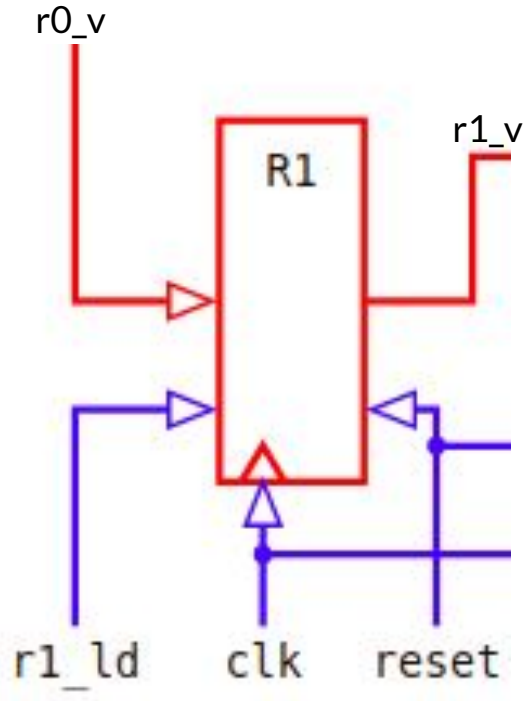
- Data path contains:
  - Add/subtract module
  - Two 16-bit registers (R1 and R0)
  - Combinational logic block (SH) to generate:
    - 1
    - -1
    - multiply by 2
    - pass through R0
- These modules can be used to generate integer sequences



Load and Reset Register

- On the positive edge:
  - If reset is 1: register is set to all zeros
  - If the ld input is 1: register is loaded from the data inputs.
  - If reset and ld are both 0: the register is not changed.
- R1 and R0 works the same way
- 16-bit type definition:

```
typedef logic [15:0] reg16_t;
```



Load and Reset Register

- On the positive edge:
  - If reset is 1: register is set to all zeros
  - If the ld input is 1: register is loaded from the data inputs.
  - If reset and ld are both 0: the register is not changed.
- R1 and R0 works the same way
- 16-bit type definition:

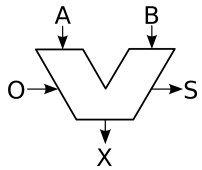
```
typedef logic [15:0] reg16_t;
```

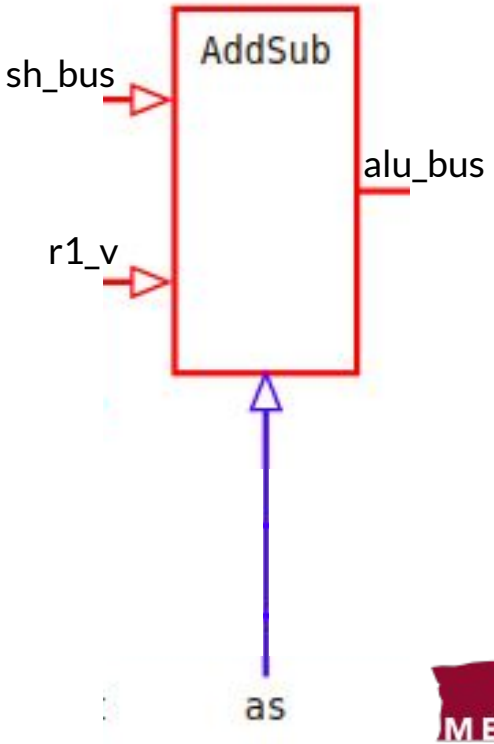
ld_reg.v

```
module ld_reg(output reg16_t q,

              input reg16_t d,

              input logic clk, reset, ld);

    always_ff @(posedge clk) begin

        if ( reset ) begin q <= 0; end

        else begin if ( ld ) q <= d; end

    end

endmodule
```

Add/Subtract Module

- Module “as_alu”
  - output: out
  - input: a, b, ctl
- The out data is either:
  - a + b
  - a - b
  - a
  - b
- Depends on the control input
- ALU will be more complex
- ALU symbol
  - O: operation
  - S: flags (CNZV)
  - A, B: Operands
  - X: Result





Add/Subtract Module

- The out data is either:
  - a + b // AS_ADD
  - a - b // AS_SUB
  - a // AS_a
  - b // AS_b
- Depending on the control input
- as_ctl_t is an enumeration of all operations

```
typedef enum logic [1:0] { AS_ADD, AS_SUB, AS_A, AS_B } as_ctl_t;
```

Add/Subtract Module

- The out data is either:
  - a + b // AS_ADD
  - a - b // AS_SUB
  - a // AS_a
  - b // AS_b
- Depending on the control input
- as_ctl_t is an enumeration of all operations

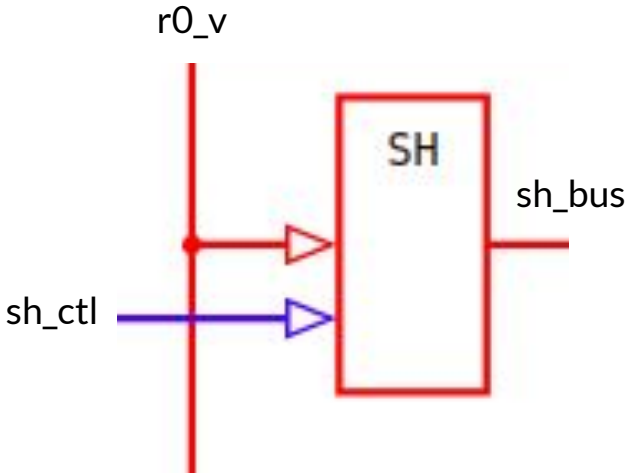
```
typedef enum logic [1:0] {
     AS_ADD,
     AS_SUB,
     AS_A,
     AS_B } as_ctl_t;
```

as_alu.v

```
always_comb begin

      case( ctl )

            AS_ADD : out = a + b;

            AS_SUB : out = a - b;

            AS_A : out = a;

            AS_B : out = b;

      endcase

end
```

Shifter Module

- Operand “a” for the Add/Sub
- Recover the value of R0
- The shifter module
  a. passes its input to the output
  b. shifts the input one position to the left (r0 * 2)
  c. provides a constant 1
  d. provides a constant all ones
- All ones is -1 (two's complement)



Shifter Module

- The shifter module
  - passes its input to the output
  - shifts the input one position to the left (r0 * 2)
  - provides a constant 1
  - provides a constant all ones
- In two's complement, all ones is -1.
- shift_ctl_t is the enumeration for the operations of the shifter module.

```
typedef enum logic [1:0] {SH_PASS,SH_LEFT,SH_ONE,SH_ONES} shift_ctl_t;
```

Shifter Module

- The shifter module
  - passes its input to the output
  - shifts the input one position to the left (r0 * 2)
  - provides a constant 1
  - provides a constant all ones
- In two's complement, all ones is -1.
- shift_ctl_t is the enumeration for the operations of the shifter module.

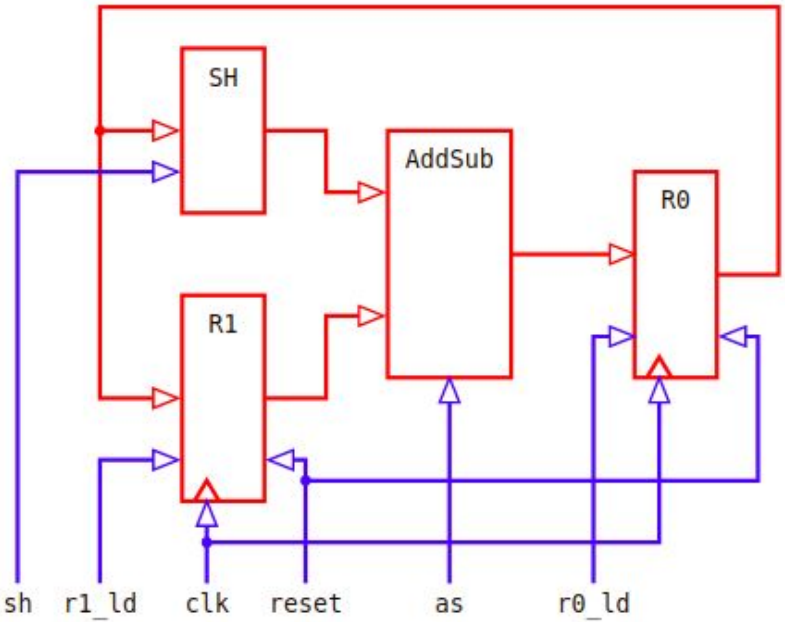
```
typedef enum logic [1:0] {SH_PASS, SH_LEFT, SH_ONE, SH_ONES} shift_ctl_t;
```

shifter.v

```
always_comb begin

      case( ctl )

            SH_PASS: out = in;

            SH_LEFT: out = in << 1;

            SH_ONE: out = 1;

            SH_ONES: out = 16'hffff;

      endcase

end
```

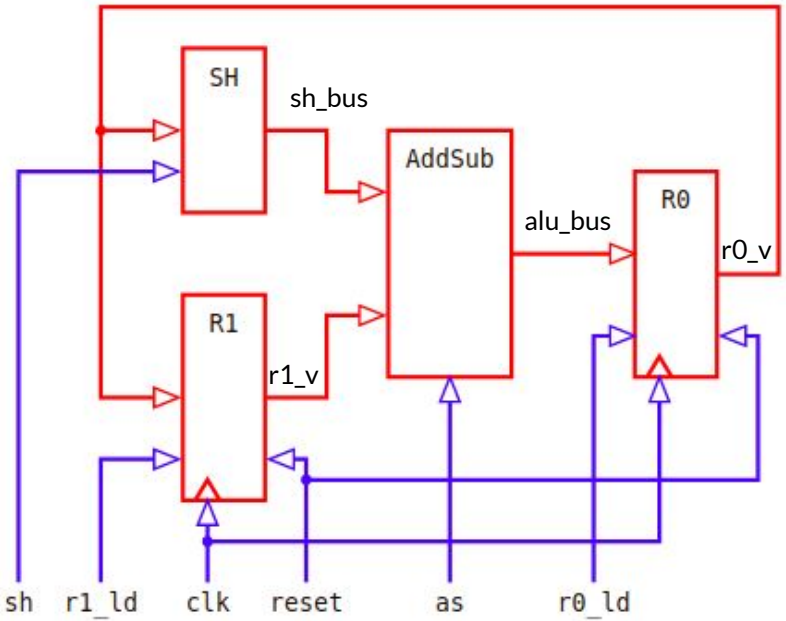
Data Paths and Control in Verilog

- Putting everything together
- Sequence generator module (seq_gen_dp.v)
- output
  - reg16_t out
- input
  - clk
  - reset
  - as_ctl
  - shift_ctl
  - r1_ld
  - r0_ld



Data Paths and Control in Verilog

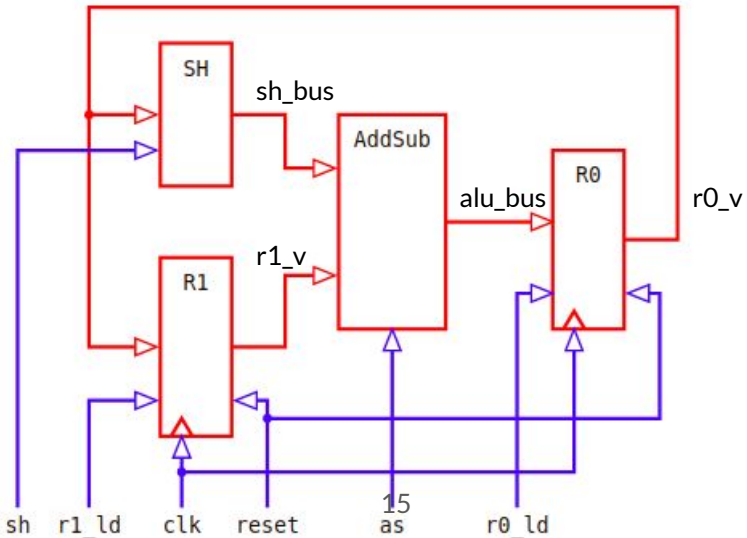
- Putting everything together
- Sequence generator module (seq_gen_dp.v)
- Busses:
  - alu_bus
  - sh_bus
  - r0_v
  - r1_v



Data Paths and Control in Verilog

- Putting everything together
- Sequence generator module (seq_gen_dp.v)

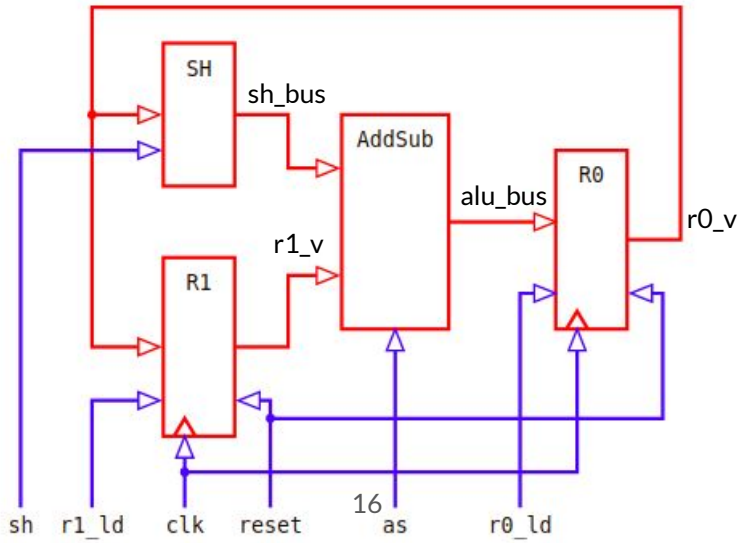
```
shifter SH( .out(sh_bus), .in(r0_v), .ctl(shift_ctl) );
```



Data Paths and Control in Verilog

- Putting everything together
- Sequence generator module (seq_gen_dp.v)

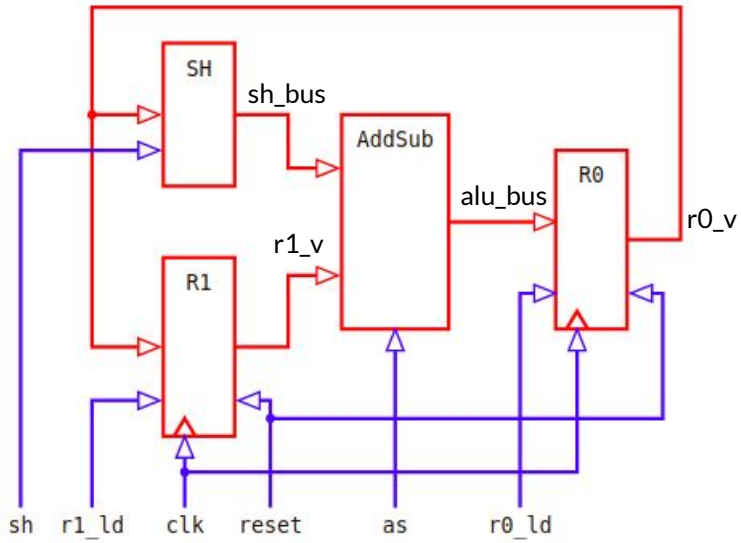
```
ld_reg R1(.q(r1_v), .d(r0_v), .clk(clk), .reset(reset), .ld(r1_ld));
```



Data Paths and Control in Verilog

- Putting everything together
- Sequence generator module (seq_gen_dp.v)

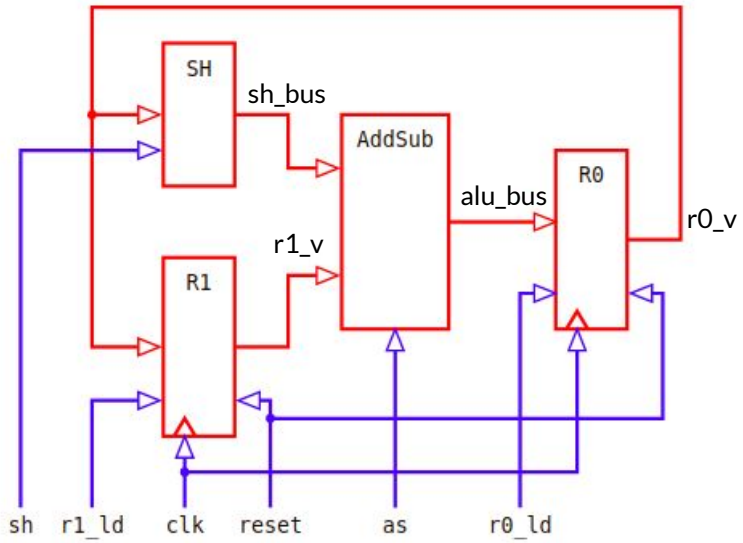
```
as_alu ALU( .out(alu_bus), .a(sh_bus), .b(r1_v), .ctl(as_ctl) );
```



Data Paths and Control in Verilog

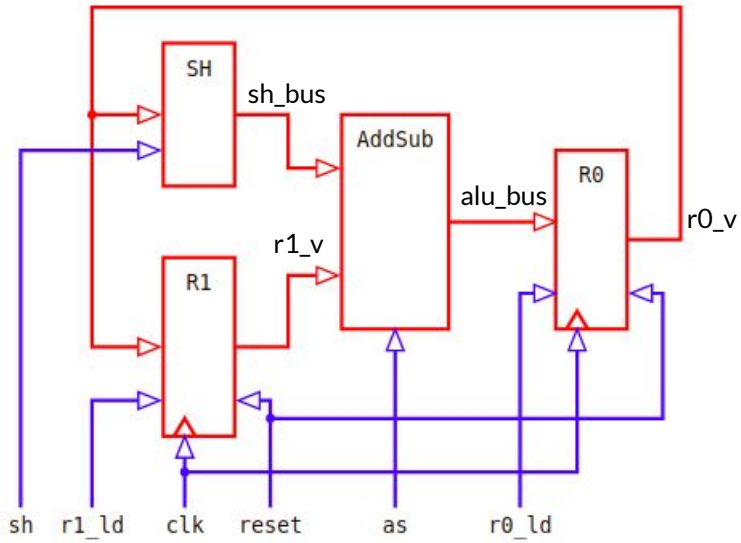
- Putting everything together
- Sequence generator module (seq_gen_dp.v)

```
ld_reg R0( .q(r0_v), .d(alu_bus), .clk(clk), .reset(reset), .ld(r0_ld) );
```



Data Paths and Control in Verilog

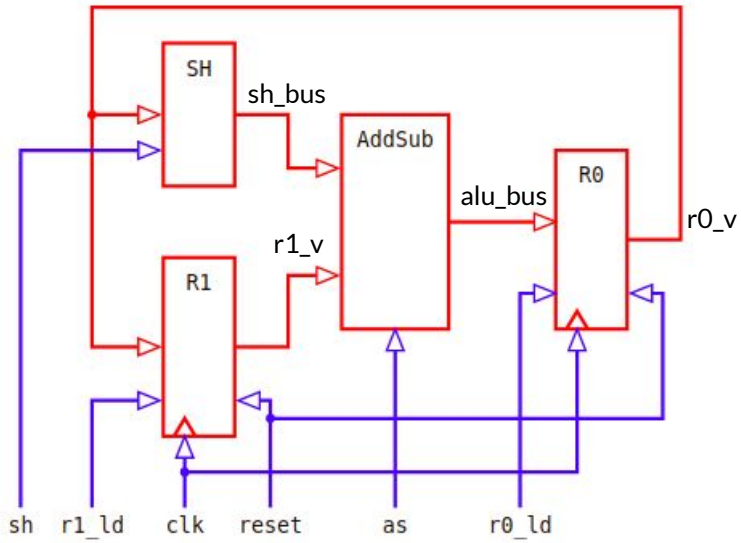
- Putting everything together
- Sequence generator module (seq_gen_dp.v)
- iverilog -g2012 tb_rtl.v
  - R0<=1
  - R1<=R0



Data Paths and Control in Verilog

- Putting everything together
- Sequence generator module (seq_gen_dp.v)

```
assign out = r0_v;
```



End of part 1

- Design of a mini data path
- Next part:
  - Test the mini datapath
  - Register Transfer Level (RTL) Design