18-360
Intro to Computer-Aided Digital Design
ECE Department

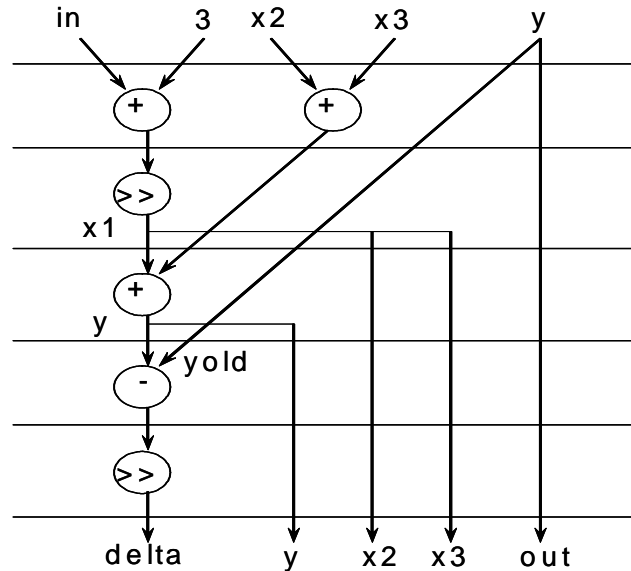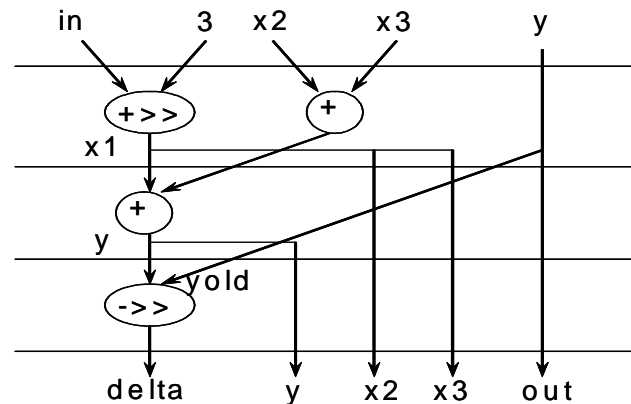**Carnegie Mellon**

Out: 2/19/04 Homework 3 Solutions

1. Do as soon as possible (15%)

a. The ASAP dataflow graph is shown below:

b. Assuming one ALU is available in the datapath, there are 5 states needed, if each operation takes one state in an FSM.

c. If shifting and addition/subtraction can be done in one clock cycle, the graph can be scheduled in 3 states, as shown below.

d. Physical events:

```
delta <= >> 1
setting y
out <= y;
setting x2 and x3
```
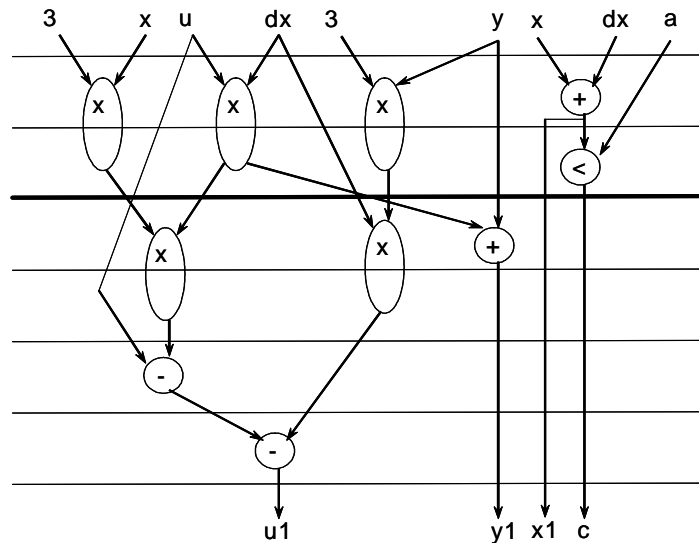
Some examples of the logical events:

```
setting of x1 (x1 = (in + 3) >> 1; seems like only used within)
setting yold (yold = y
calculating y - yold
calculating x1 + x2 + x3.
```

## 2. Slow multipliers   (20%)

a.

b. 8 registers (calculated as the maximum number of DISTINCT lines crossing any dotted line). See the bold line in the above figure. Here u*dx is only counted once.

c. We can use pipelined multipliers in the case when there is only one multiplier available, by serializing the multiplies. 9 registers are needed, as shown below (see the bold line). Note that two multipliers can't share any one clock cycle if they have data dependencies. .
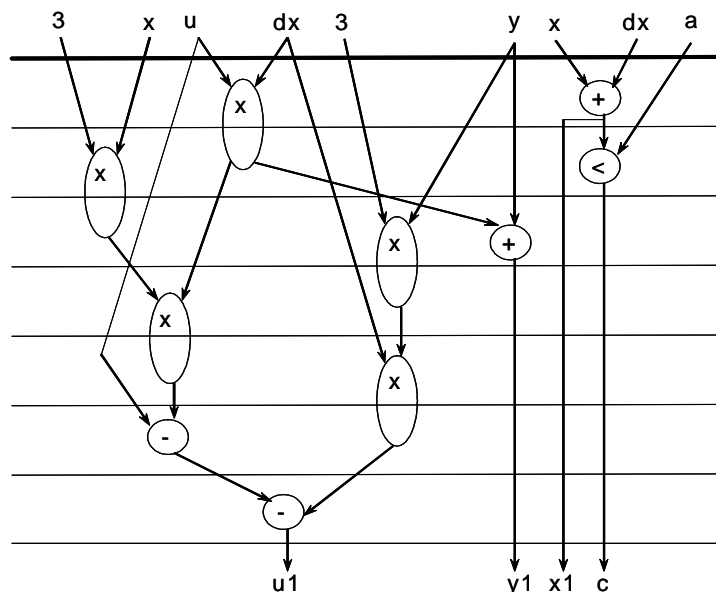
3. Does anyone know what time it is? (10%)

The outputs of these systems could be different, even if we clock them at the same rate and give them the same sequences of inputs. While simulating the cycle-accurate model, the combinational logic is assumed to take zero time. However, the gate-accurate model has delays between each gate. So, if InputA were to change a very small amount before the clock edge, the cycle-accurate module would always use the new value. The gate-accurate model could only use the value if it were able to propagate through its gates quickly enough to reach the register before the edge.

4. Not the p18240 Again … (20%)

Here is the code.

```
`define lda 10'b00_0001_0000
`define ldi 10'b00_0011_0000
`define sta 10'b00_0001_1000
`define add 10'b00_0011_1000
`define brn 10'b00_1011_0000

module p18360;

   reg [15:0] mem[0:999];
   reg [15:0] MAR, MDR, PC, IR, SP;
   reg [16:0] temp;
   reg [15:0] regs[0:7];
   reg        Z, N, C, V;
   reg [9:0]  cycle;
   reg        clock, reset;

   initial begin
      $readmemh("p4_test.o", mem);

      cycle = 0;
      PC = 16'h0000;
      N = 0;
      Z = 0;
      C = 0;
      V = 0;

      reset = 1;
      #2 reset = 0;
      #2 reset = 1;
      clock = 0;
   end

   always #10 clock = ~clock;

   always begin: main_loop
      MAR <= PC;
      @(posedge clock);
      PC <= PC + 1;
      MDR <= mem[MAR];
      @(posedge clock);
```

```verilog
        IR <= MDR;
  @(posedge clock);
        @(posedge clock);

      case(IR[15:6])
        `lda: begin
           MAR <= PC;
           @(posedge clock);
           PC <= PC + 1;
           MDR <= mem[MAR];
           @(posedge clock);
           MAR <= MDR;
           @(posedge clock);
           MDR <= mem[MAR];
           @(posedge clock);
           regs[IR[5:3]] <= MDR;
           N <= MDR[15];
           Z <= (MDR == 16'h0000) ? 1 : 0;
           @(posedge clock);
         end
       `ldi: begin
           MAR <= PC;
           @(posedge clock);
           PC <= PC + 1;
           MDR <= mem[MAR];
           @(posedge clock);
           regs[IR[5:3]] <= MDR;
           N <= MDR[15];
           Z <= (MDR == 16'h0000) ? 1 : 0;
           @(posedge clock);
         end

        `sta: begin
           MAR <= PC;
           @(posedge clock);
           PC <= PC + 1;
           MDR <= mem[MAR];
           @(posedge clock);
           MAR <= MDR;
           @(posedge clock);
           MDR <= regs[IR[2:0]];
           @(posedge clock);
           mem[MAR] <= MDR;
        N <= MDR[15];
           Z <= (MDR == 16'h0000) ? 1 : 0;
           @(posedge clock);
         end

        `add: begin
           temp = regs[IR[5:3]] + regs[IR[2:0]];
           N <= temp[15];
           regs[IR[5:3]] <= temp;
           Z <= (temp == 16'h0000) ? 1 : 0;
           C <= temp[16];
```

```
            @(posedge clock);
        end

        `brn: begin
            MAR <= PC;
            @(posedge clock);
            if (N) begin
                MDR <= mem[MAR];
                @(posedge clock);
                PC <= MDR;
                @(posedge clock);
        end
            else begin
                PC <= PC + 1;
                @(posedge clock);
            end
        end

    endcase
    end

    always @(posedge clock) begin
        if (cycle > 250) $finish;
        $display("cycle %d", cycle);
         $display("R0: %x, R1: %x, R2: %x, R3: %x mem[0x20]: %x", regs[0],
regs[1], regs[2], regs[3], mem[32]);
          $display("R4: %x R5: %x R6: %x R7: %x mem[0x21]: %x", regs[4],
regs[5], regs[6], regs[7], mem[33]);
        $display("MAR: %x MDR: %x IR: %x PC: %x N: %x", MAR, MDR, IR, PC, N);
        $display("=========================================");
        cycle = cycle + 1;
    end

endmodule // p18360
```

Here is the test file (p4_test.o).

```
0c00
0015
0c09
fffc
0c24
0020
0c2d
0001
0a25
0412
0020
0e13
0602
0021
0e05
0e0d
2c00
```

```
0009
0000
0001
0002
0003
0004
0000
0000
0000
0000
0000
0000
0000
```

The simulation result is too long to be included here. Please run "vlog p4.v" to see the result, or ask us for a copy.

5. State transition diagram needed   (15%)

Please also see Figure 7.3 right hand side on page 224 of the textbook (fourth edition) for a better view of the solution.