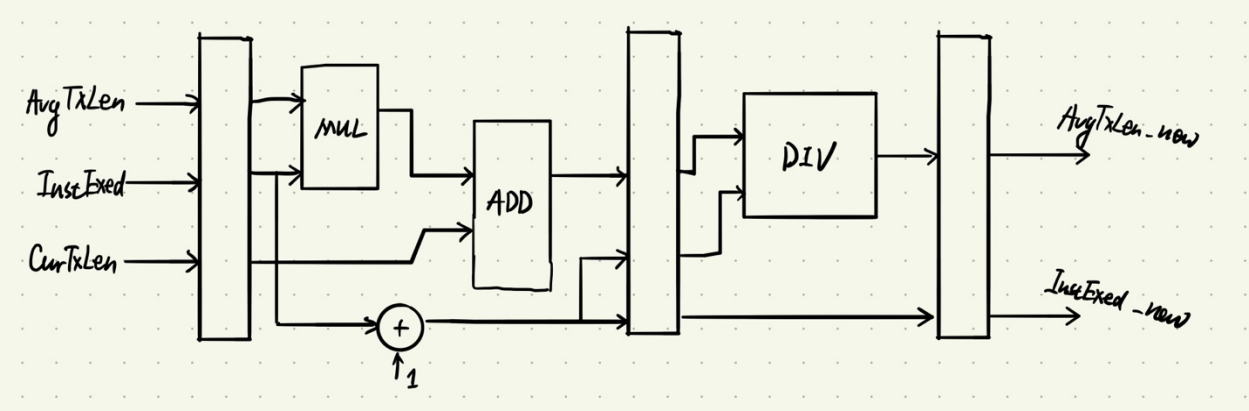
Question 2:

We can just combine the first two stages (MUL and ADD) stages because the critical path is from stage_2_reg to stage_3_reg through the divider. I expect the total delay of 8-bit multiplier and 16-bit adder is less than the delay of divider. If so, we can reduce my design from 3-stage to 2-stage which minimize sequential overhead and increase the throughput.

The schematic of my design:



Then I modeled this new design and finished synthesis. The fastest that this design can run is also 200 MHz (same as original design). As shown in timing report, the critical path is still through the divider.

```
Startpoint: stage_1_reg[9]
            (rising edge-triggered flip-flop clocked by clk)
Endpoint: stage_2_reg[8]
          (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max
```

That means total delay of adder and multiplier is still less than that of divider, so we can further fasten this design by placing some combinational logic of divider into the previous stage (register rebalancing) to further minimize sequential overhead. As shown in area report, the total area decreases (compared to 6355 of original design) because I save a stage register and related circuits.

```
Combinational area:               5216.738701
Buf/Inv area:                     1494.251166
Noncombinational area:             743.371216
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                  5960.109916
Total area:                 undefined
1
```

Appendix:

Source code

```
module TM_ALU (clk, reset,
AvgTxLen, // 8-bit input
InstExed, // 8-bit input
CurTxLen, // 8-bit input
AvgTxLen_new, // 8-bit output
InstExed_new // 8-bit output
);
    input clk, reset;
    input [7:0] AvgTxLen, CurTxLen;
    input [7:0] InstExed;

    output [7:0] AvgTxLen_new;
    output [7:0] InstExed_new;

    // Stage 1;
    reg [23:0] stage_0;
    reg [31:0] stage_1;
    wire [15:0] product;
    assign product = stage_0[23:16] * stage_0[15:8];

    always @(posedge clk, posedge reset)
    begin
        if(reset) {stage_0, stage_1} <= 0;
        else
        begin
            stage_0 <= {AvgTxLen, InstExed, CurTxLen};
            stage_1[31:16] <= product + stage_0[7:0];
            stage_1[15:8] <= stage_0[15:8] + 1;
            stage_1[7:0] <= stage_0[15:8] + 1;
        end
    end

    // Stage 2;
    reg [15:0] stage_2;
    always @(posedge clk, posedge reset)
    begin
        if(reset) stage_2 <= 0;
        else
        begin
            stage_2[15:8] <= stage_1[31:16] / stage_1[15:8];
            stage_2[7:0] <= stage_1[7:0];
        end
    end

    assign AvgTxLen_new = stage_2[15:8];
```

```
    assign InstExed_new = stage_2[7:0];

endmodule
```