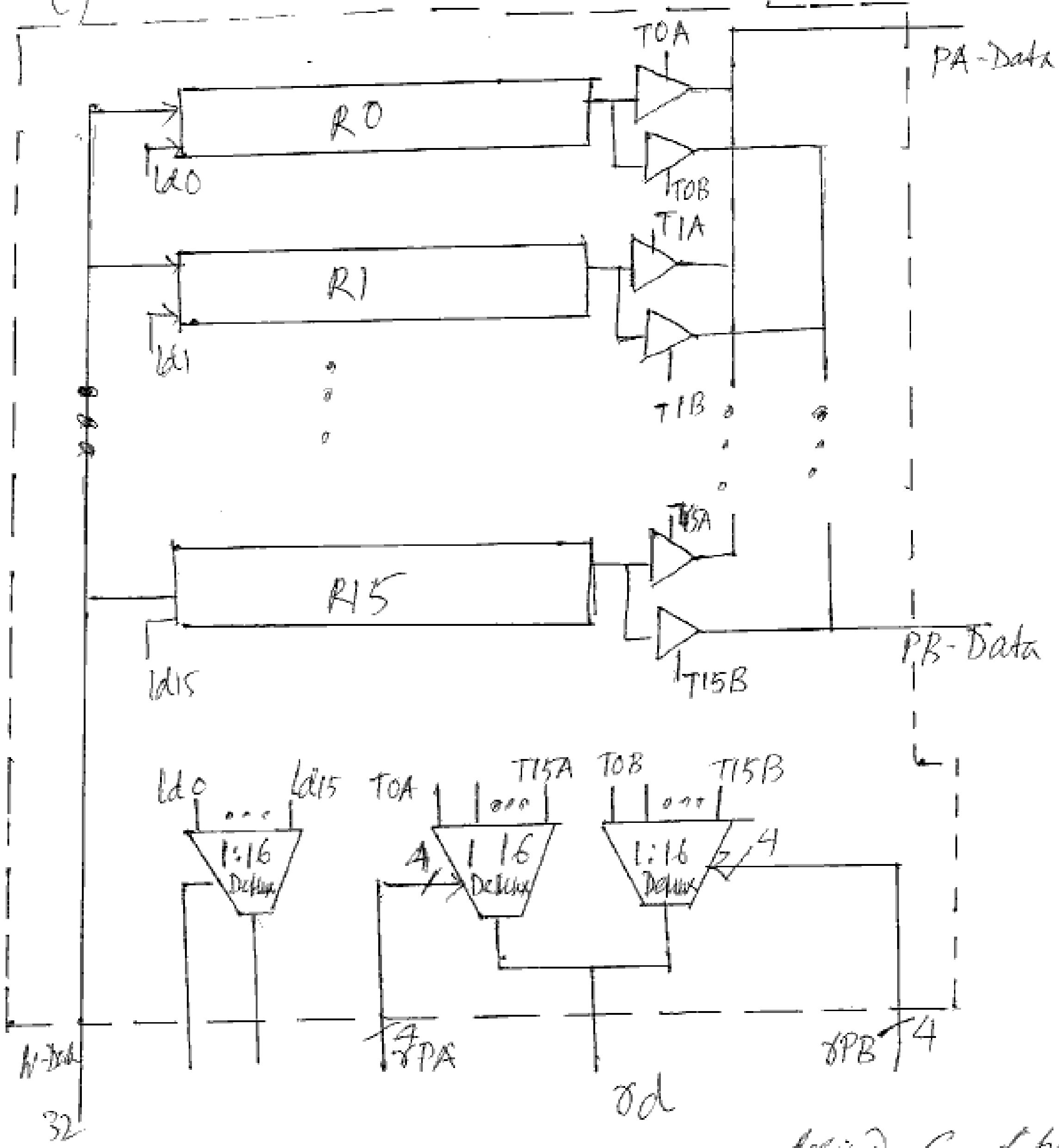
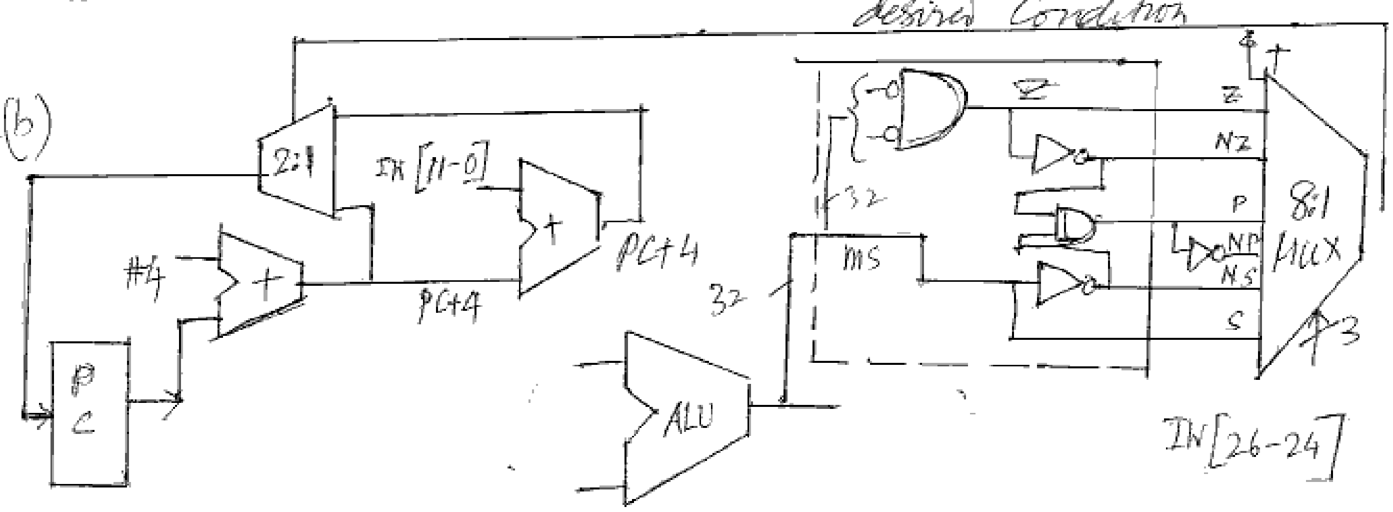
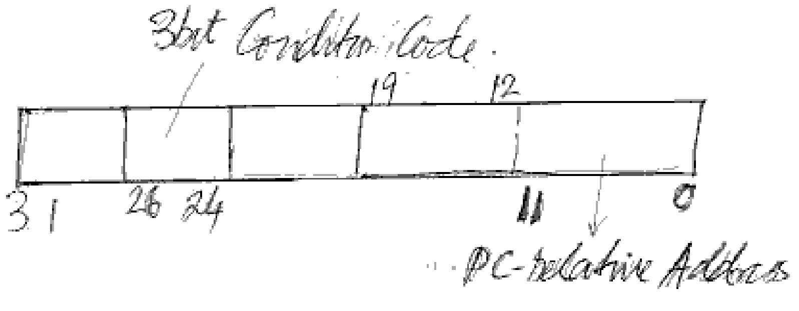
1(a)



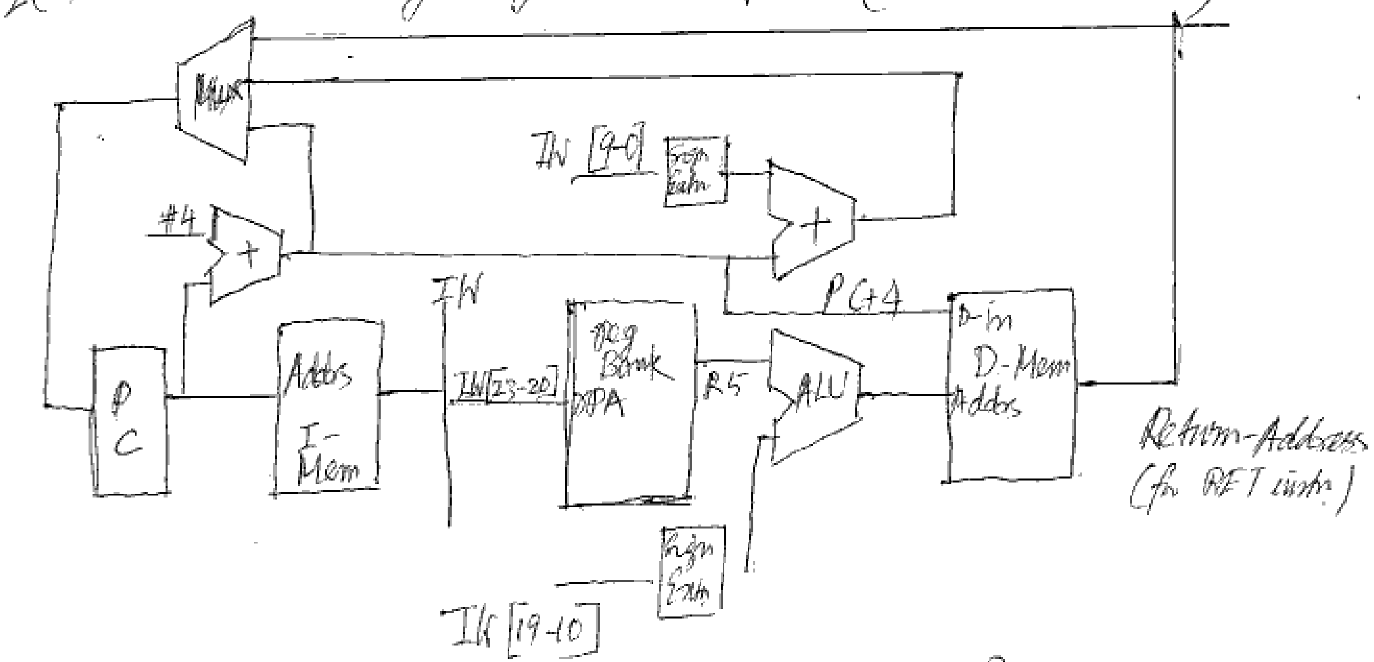
1(b)



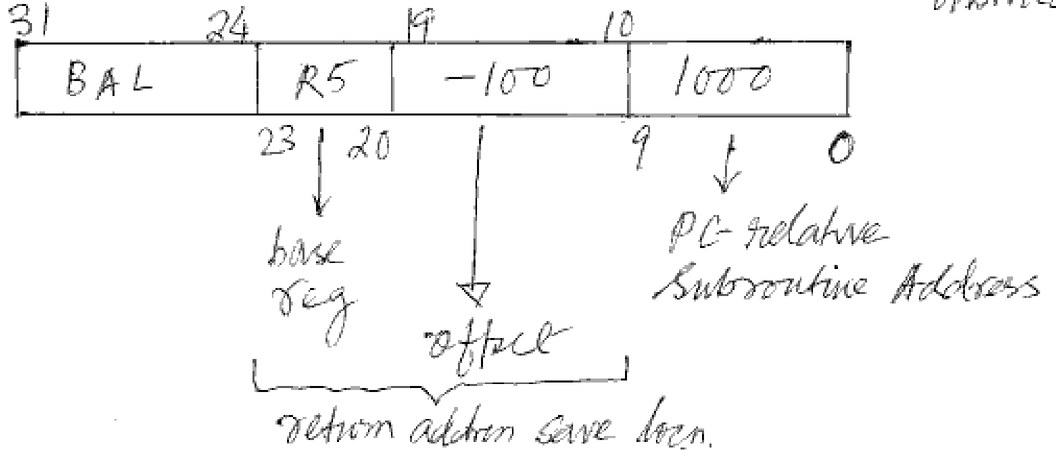


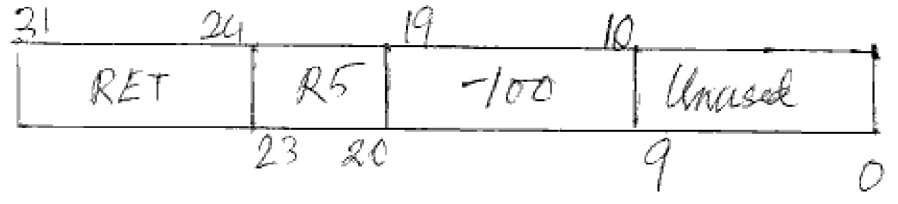
Let the instruct

2(a) Single Cycle Datapath (for BAL- & RET-)



Single cycle DP for BAL & RET instructions





BAL- μ-ops

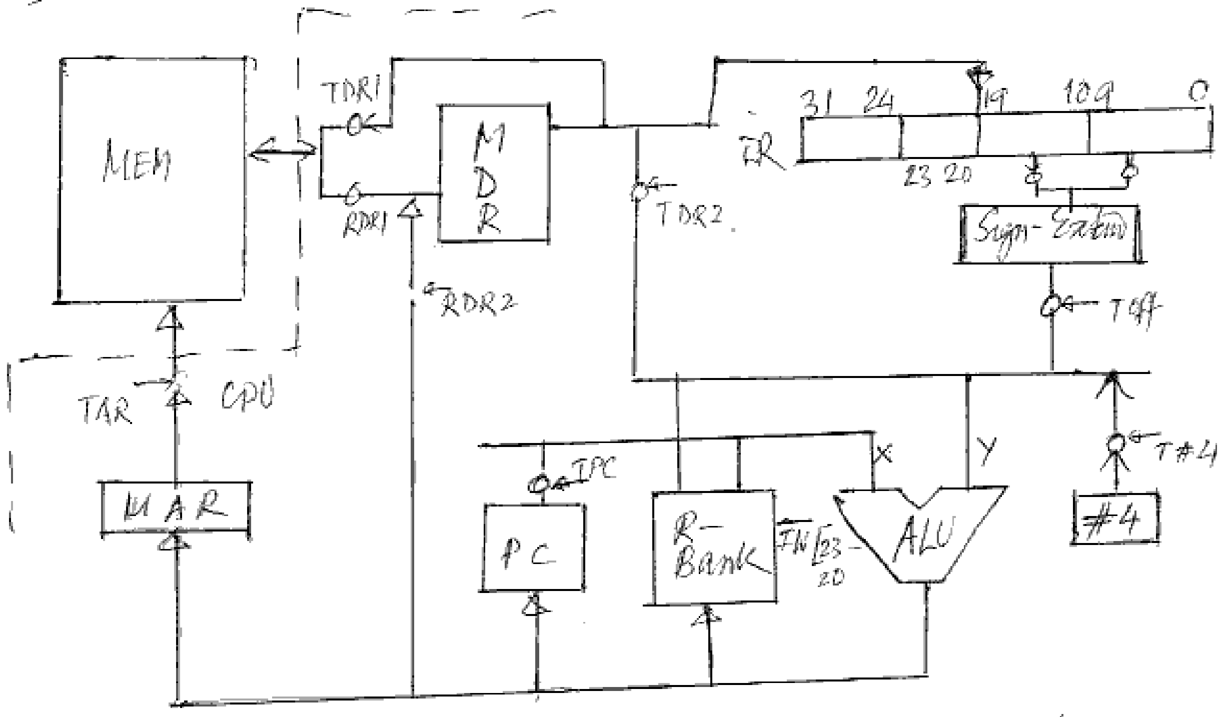
$M[\text{regBank}[IW[23-20]] + \text{SignExtend}(IW[19-10])] \leftarrow PC+4$

$\parallel\ PC \leftarrow PC+4+\text{SignExtend}(IW[9-0])$

RET-μ-ops:

$PC \leftarrow M[\text{regBank}[IW[23-20]]]$

BAL, RET Instruction execution in Multi-cycle DP :



BAL- instruction execution cycle — ; delimits ck-boundaries
// assumed PC⇐PC+4 takes place in the fetch cycle

MAR ⇐ R[IW [23-20]] + Sign-Extend(IW[19-10]) ;

MDR ⇐ PC ; // through ALU "transX" function

M [MAR] ⇐ MDR ;

PC ⇐ PC + Sign-Extend (IW [9-0]) ;

RET Instruction execution cycle

MAR ⇐ R [IW[23-20]] + Sign Extend (IW[19-10])

MDR ⇐ M [MAR] ;

PC ⇐ MDR ;

8-bit Verilog Code for Booth’s Multiplier

```
module multiplier(prod, busy, mc, mp, clk, start);
output [15:0] prod;
output busy;
input [7:0] mc, mp;
input clk, start;
reg [7:0] A, Q, M;
reg Q_1
;reg [3:0] count;
wire [7:0] sum, difference;

always @(posedge clk)
begin
        if (start)
        begin
                A <= 8'b0;
                M <= mc;
                Q <= mp;
                Q_1 <= 1'b0;
                count <= 4'b0;
        end

        else
         begin case ({Q[0], Q_1})
                2'b0_1 : {A, Q, Q_1} <= {sum[7], sum, Q};
                2'b1_0 : {A, Q, Q_1} <= {difference[7], difference, Q};
                default: {A, Q, Q_1} <= {A[7], A, Q};
        endcase

        count <= count + 1'b1;
        end
end

alu adder (sum, A, M, 1'b0);
alu subtracter (difference, A, ~M, 1'b1);
assign prod = {A, Q};
assign busy = (count < 8);

endmodule
```

```
// The following is an alu.
//It is an adder, but capable of subtraction:
//Recall that subtraction means adding the two's complement—
//a - b = a + (-b) = a + (inverted b + 1)
//The 1 will be coming in as cin (carry-in)

module alu(out, a, b, cin);

output [7:0] out;
input [7:0] a;
input [7:0] b;
input cin;

        assign out = a + b + cin;

endmodule
```

--
TEST BENCH
--

```
`timescale 1ns/1ps
module booth_multi_tb;
reg [7:0] mp,mc;
reg clk, start;
wire [15:0] prod; wire busy;

multiplier u1 (.prod(prod), .busy(busy), .mc(mc), .mp(mp), .clk(clk), .start(start));

initial
begin
clk=1'b1;
forever #50 clk=~clk;
end

initial
begin
start=1'b1;
mc=8'b1010_1010;
```

```
mp=8'b0010_0010;


$stop;
end
endmodule
```