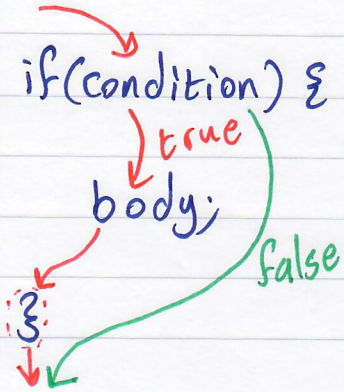
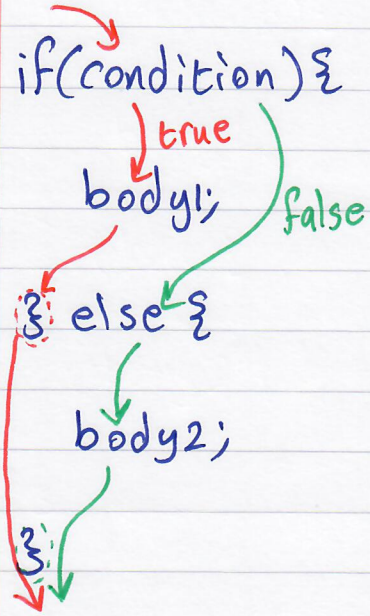
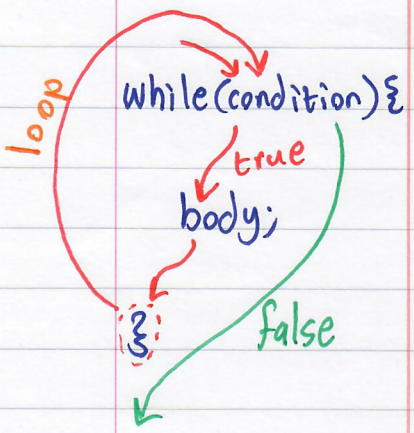
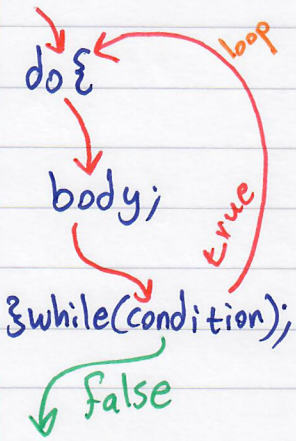
Lecture 6

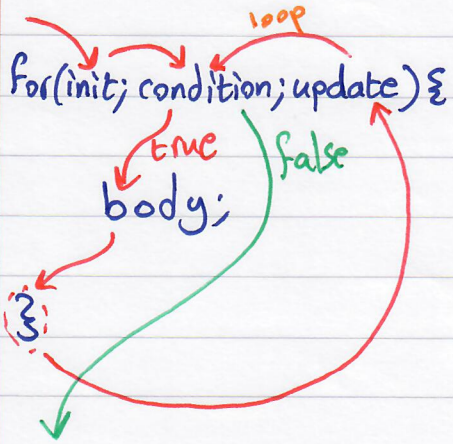
Control Statements







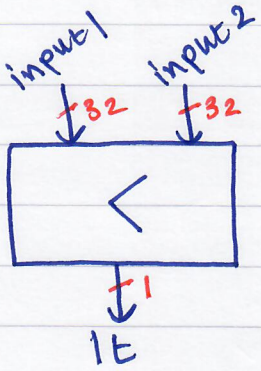




Ali Nawab

ComparatorLT (behavioral)

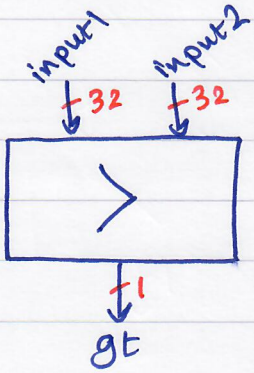
Less Than



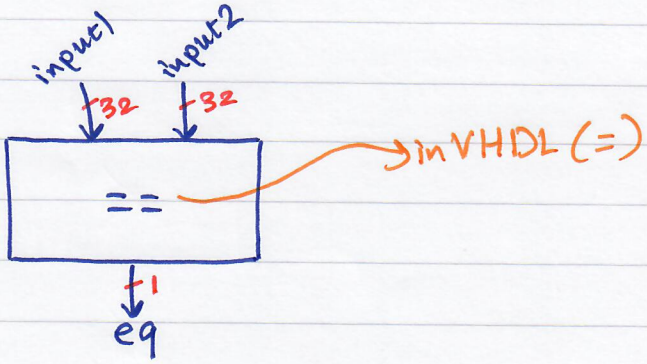

```
library ieee;
use ieee.std_logic_1164.all;

entity ComparatorLT is
port(
input1, input2 : in integer;
lt : out std_logic
);
end entity;

architecture behavior of comparatorLT is
begin

    lt <= '1' when (input1 < input2) else '0';

end architecture;
```

Comparator GT (behavioral)

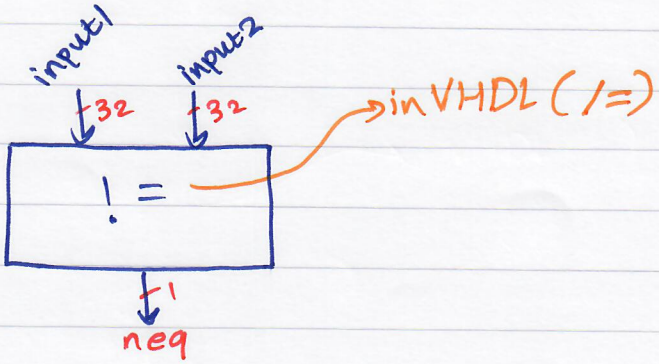
Greater Than



Comparator EQ (behavioral)



ComparatorNEQ (behavioral)

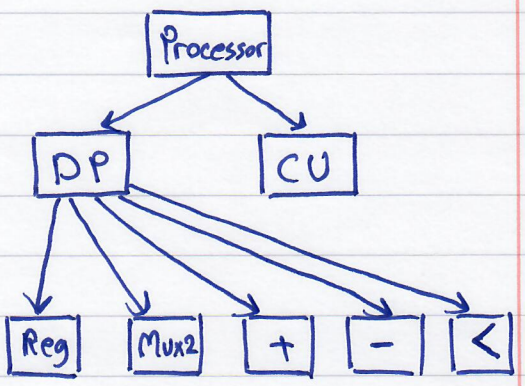


Design a Custom SPP from the following code:

```
x = a+b;
y = a-b;
if(x<y){
  z = x;
} else {
  z = y;
}
```

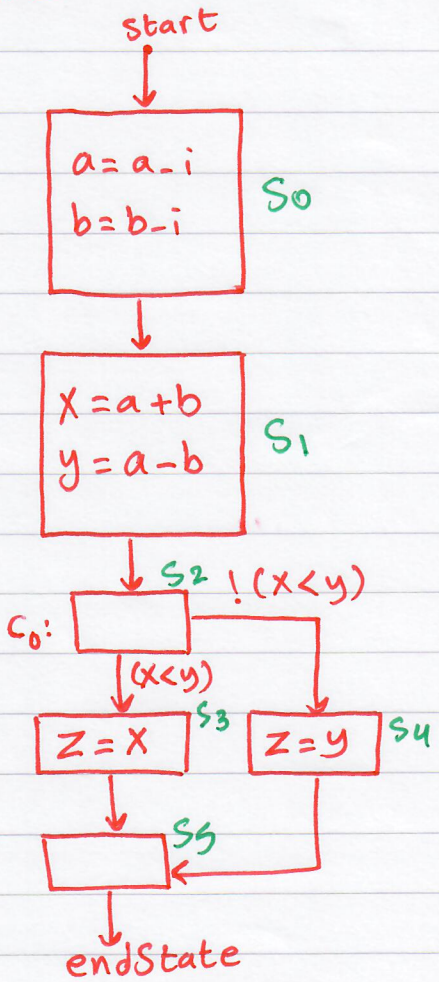
The inputs are a and b
The output is z

[1] Hierarchy

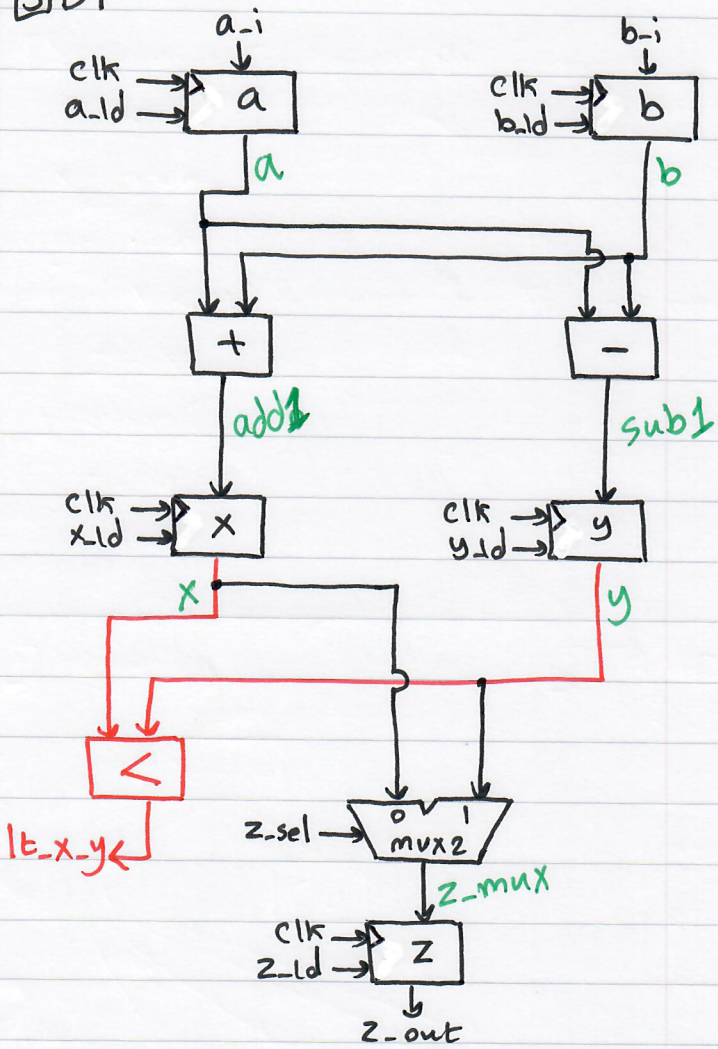


Always Fill In The Details

[2] FSMD

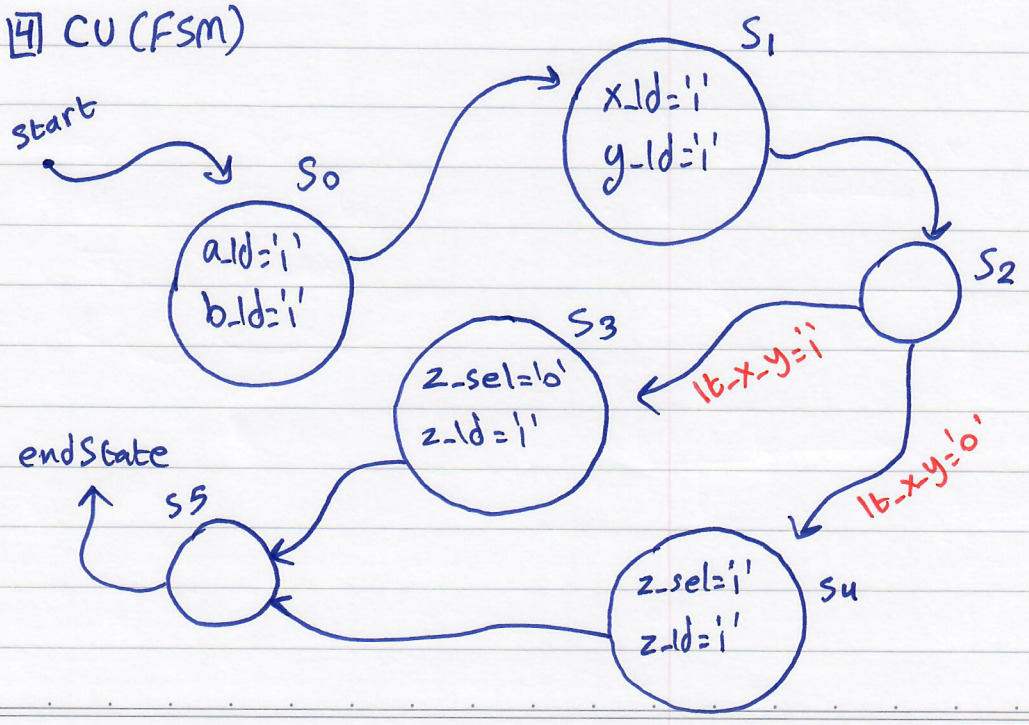


3 DP

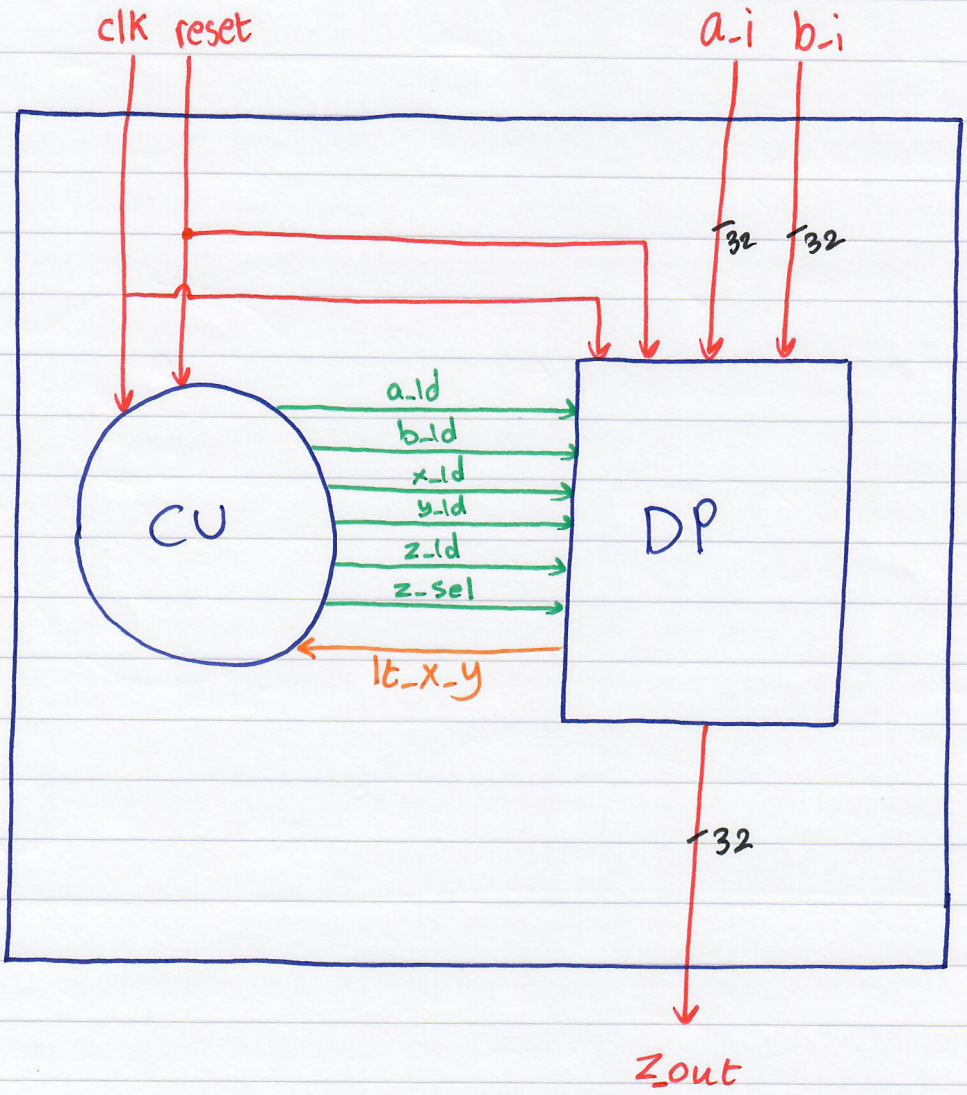


Always Fill In The Details

4 CU (FSM)



5 Processor

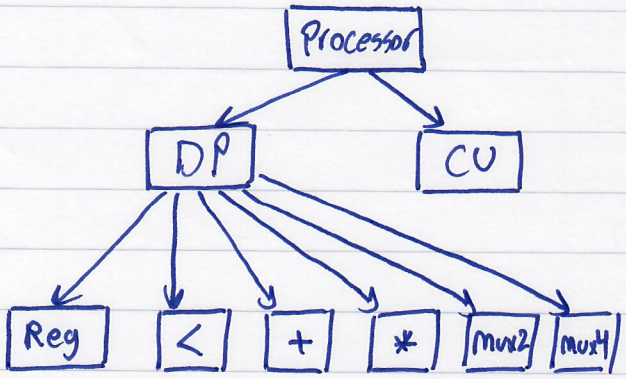


Design a custom SPP for the following code:

```
int z=0;
int x=0;
while(x<20){
    z=z+x;
    x++;
}
z=z*n
```

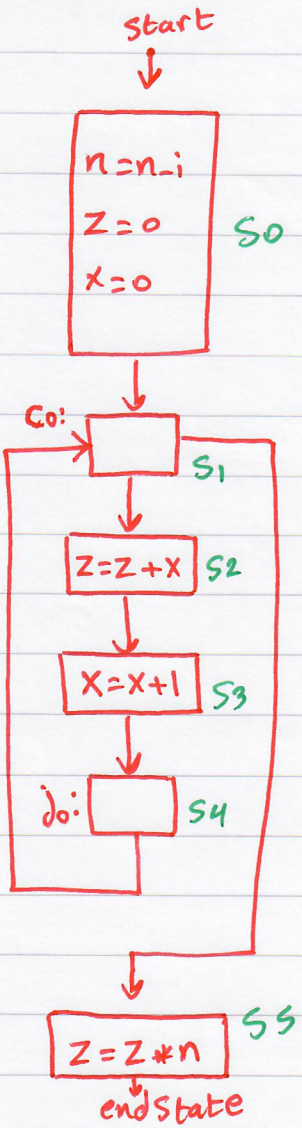
The input is n
The output is z

[1] Hierarchy

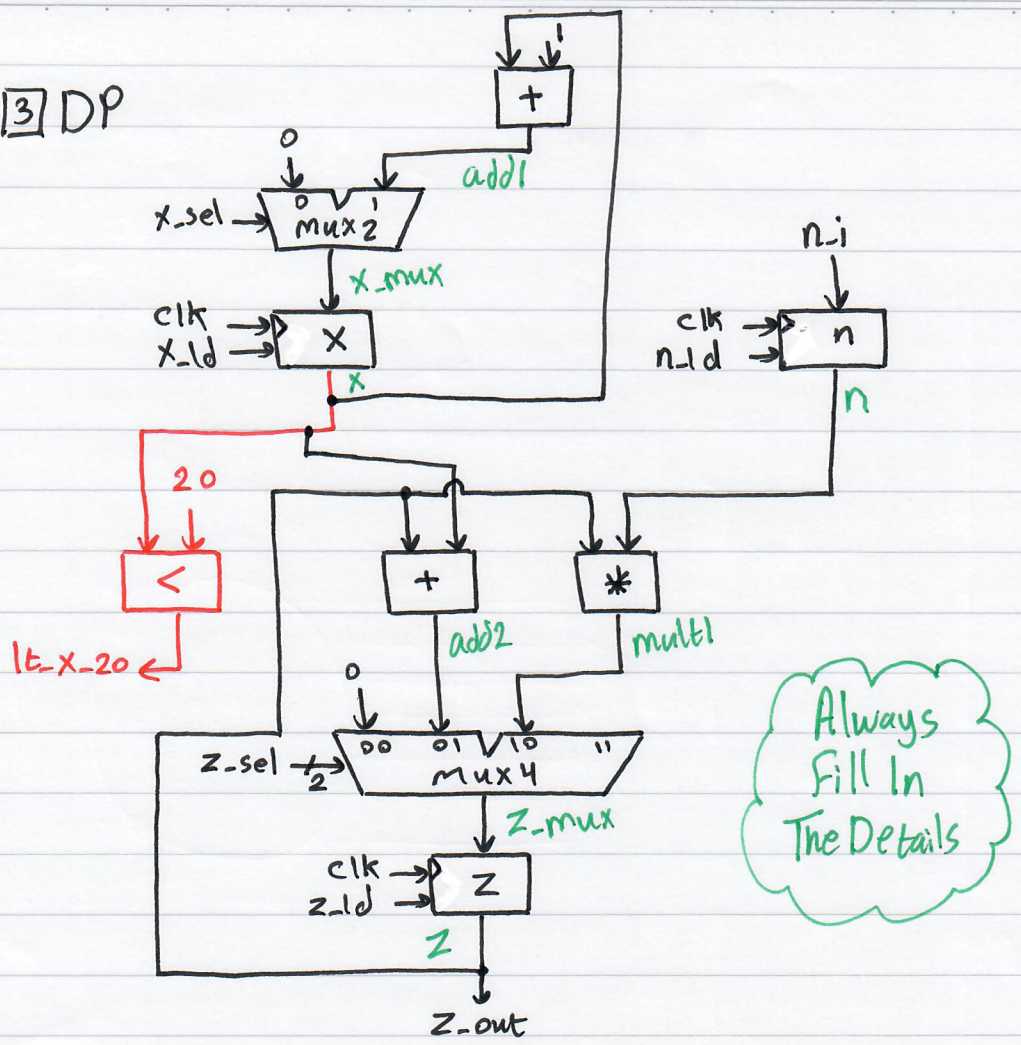


Always Fill In The Details

[2] FSMD

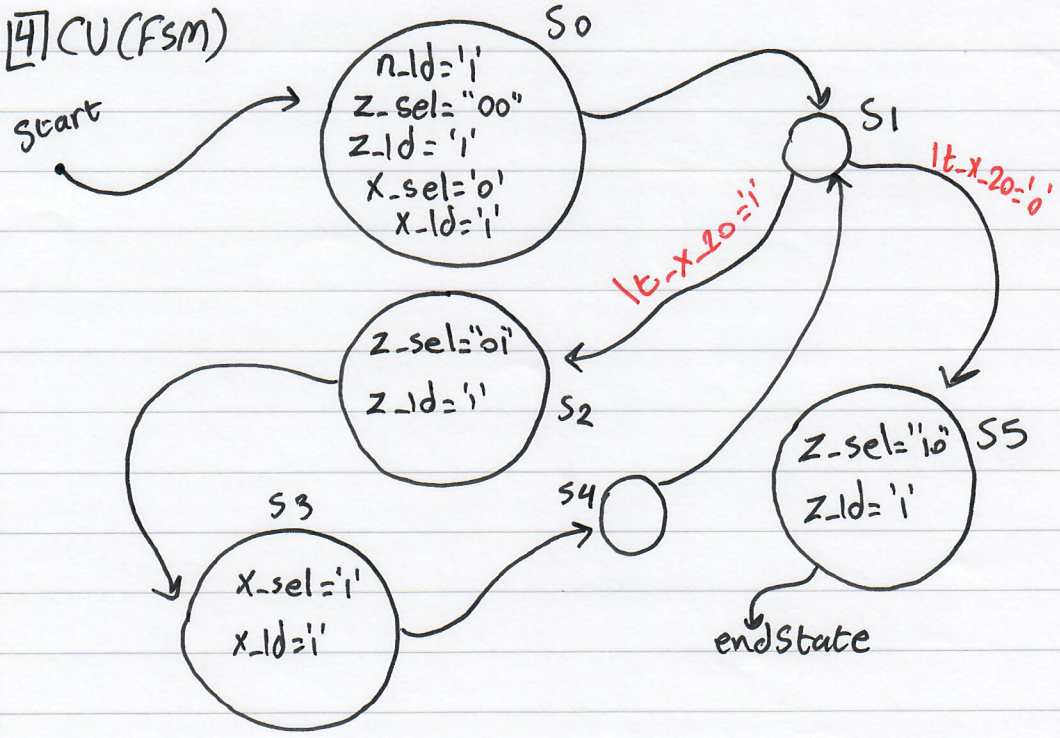


3 DP

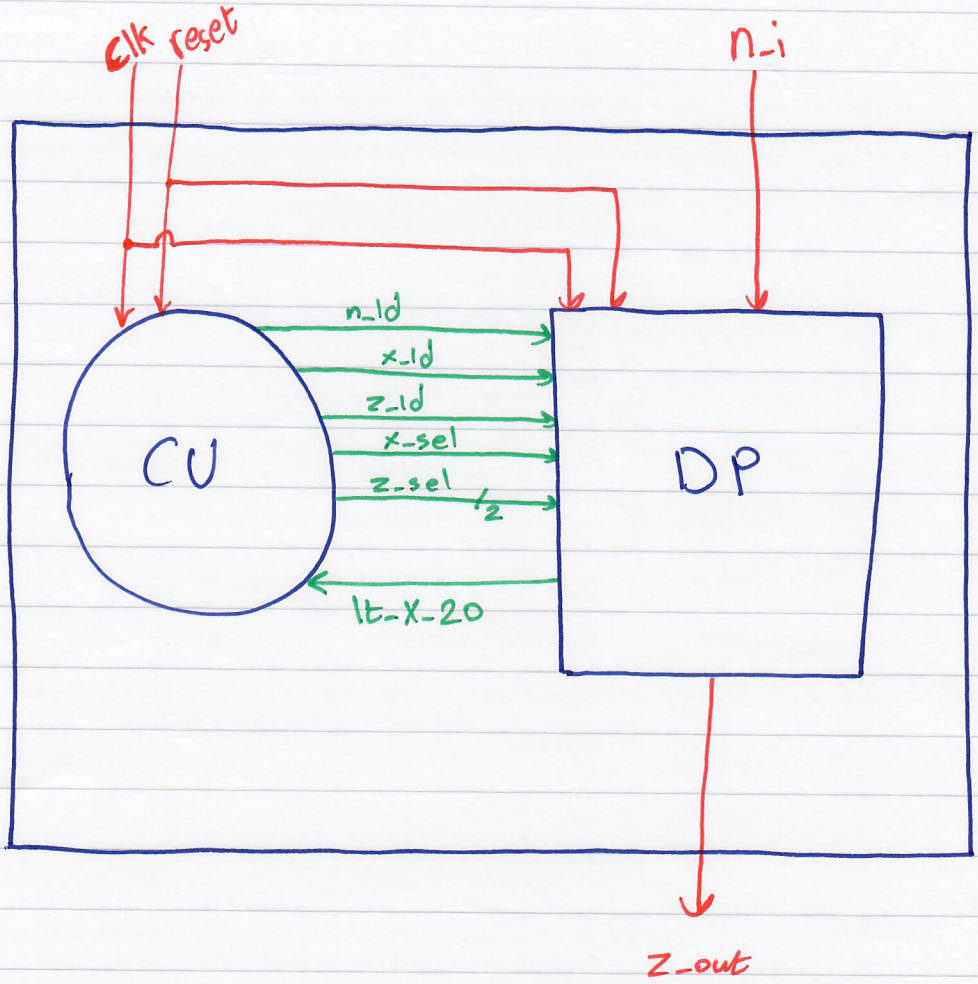


Always Fill In The Details

4 CU (FSM)



15 Processor



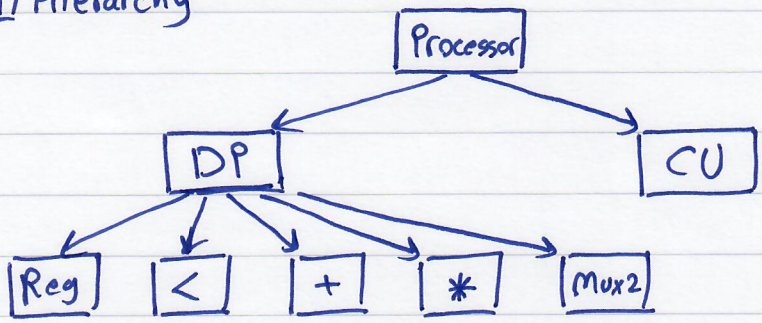
Design a custom SPP for the following Java method:

```
public int factorial (int n){
 int f=1;
 for(int i=1; i<n+1; i++){
     f=f*i;
 }
 return f;
}
```

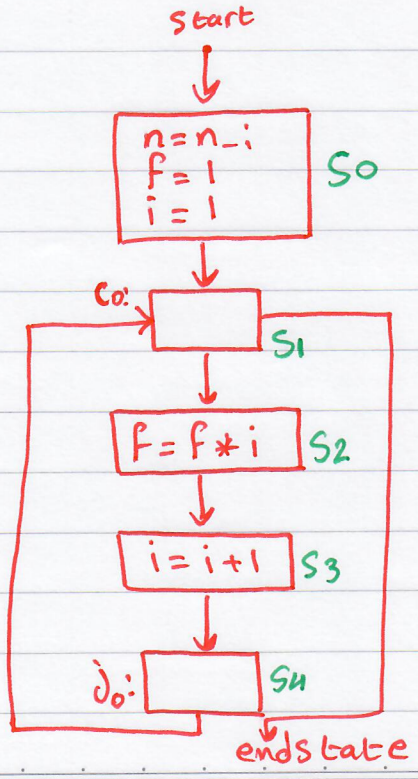
input → n

output → f

1 Hierarchy

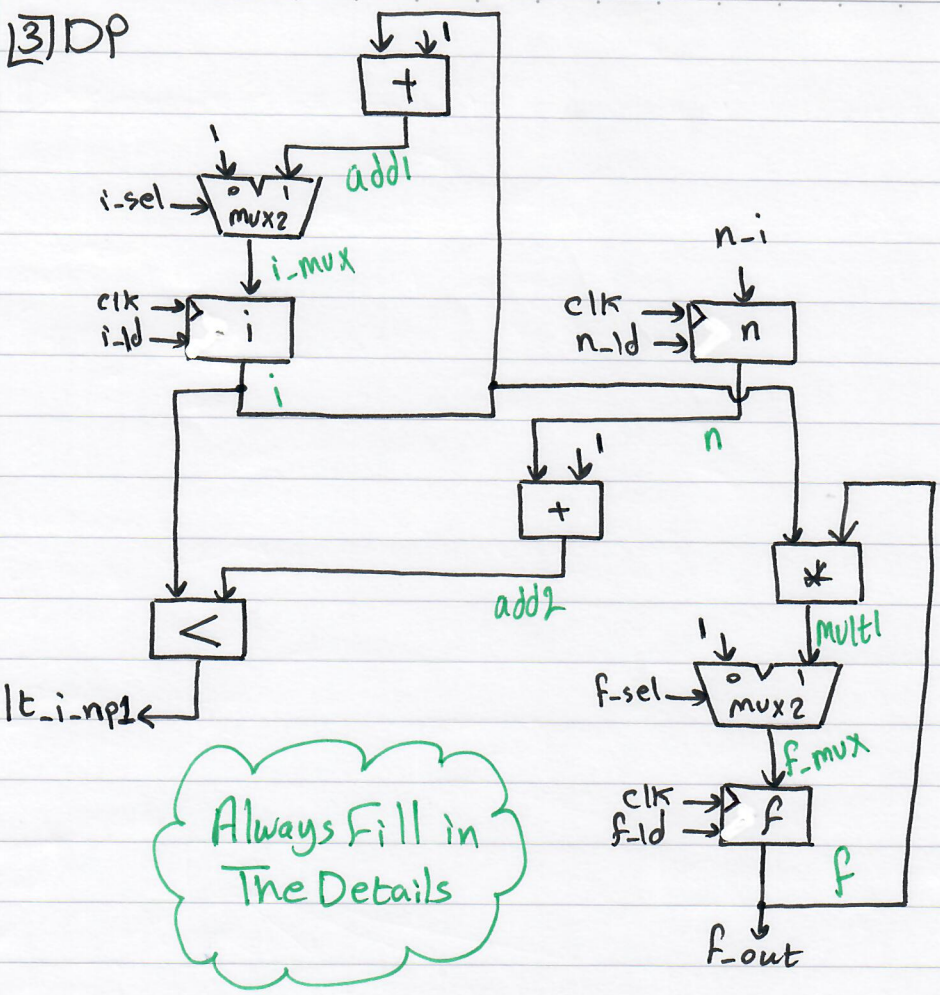


2 FSMD

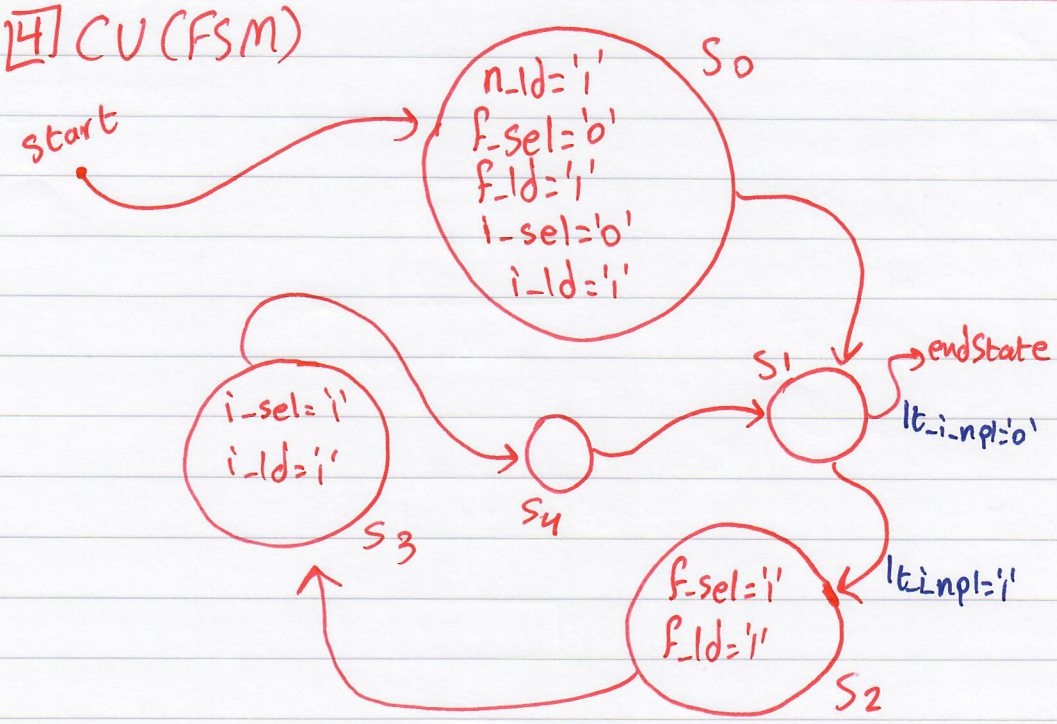


Always Fill In The Details

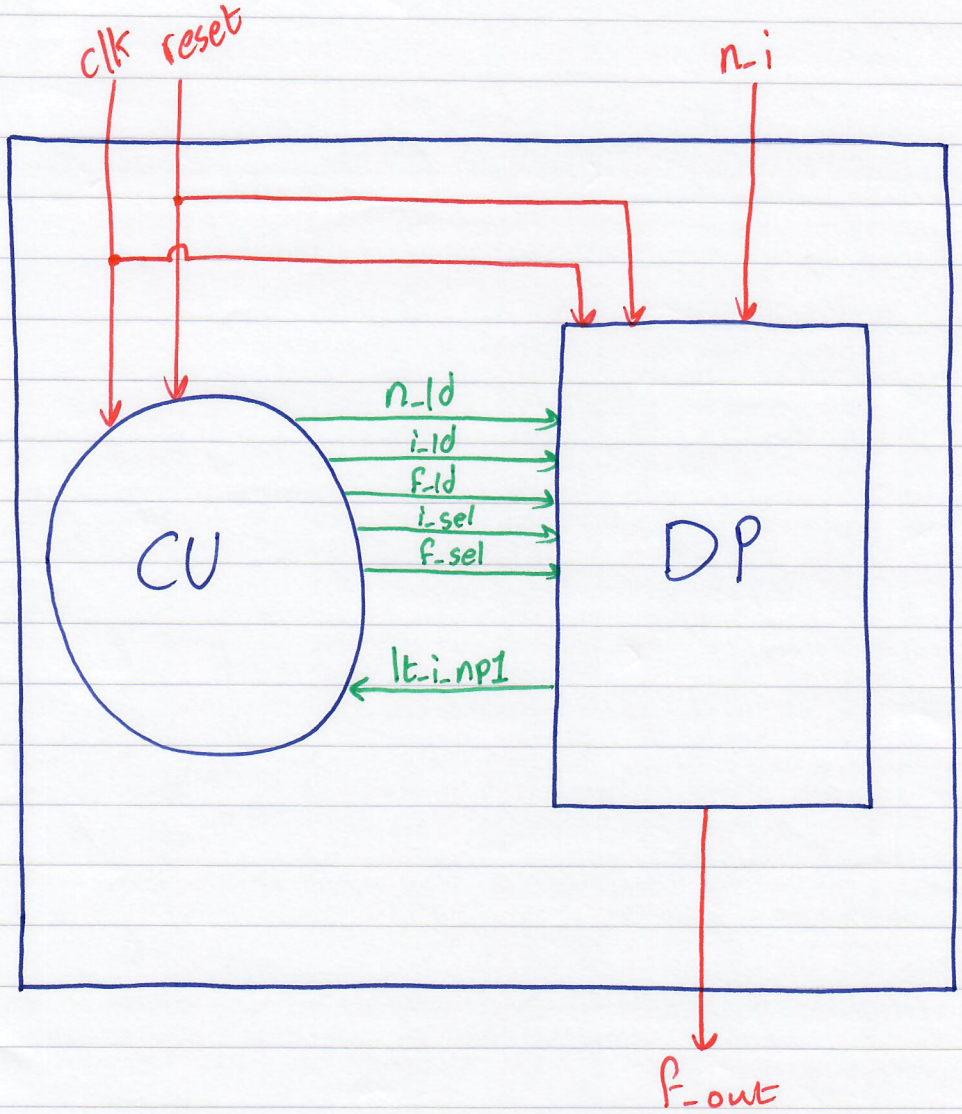
[3] DP


+
1
add1
i_sel
mux2
i_mux
clk
i_ld
i
n_i
clk
n_ld
n
+
add2
*
mult
f_sel
mux2
f_mux
clk
f_ld
f
f_out
<
lt_i_np1
Always Fill in The Details

[4] CU (FSM)


start
n_ld='1'
f_sel='0'
f_ld='1'
i_sel='0'
i_ld='1'
S0
S1
endState
lt_i_np1='0'
i_sel='1'
i_ld='1'
S3
S4
f_sel='1'
f_ld='1'
lt_i_np1='1'
S2

[5] Processor (factorial)



Try implementing the previous design in VHDL.