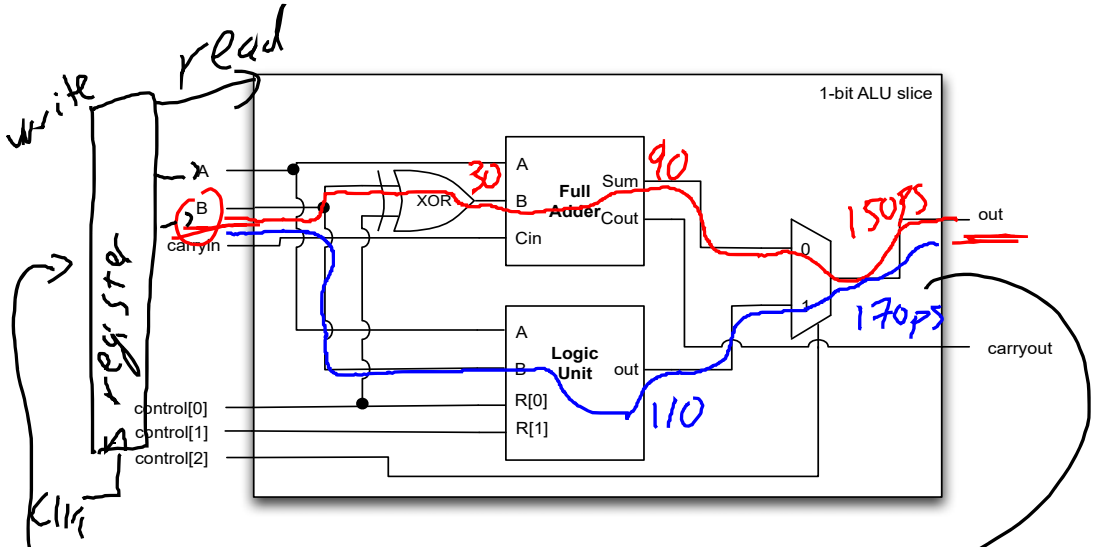
Computing components from ALU1



What is the worst case propagation delay from B to out? (consider both arithmetic and logic operations)

A: 120ps
B: 150ps
C: 160ps
D: 170ps
E: 190ps

XOR gate

In	Out	Delay
A,B	out	30ps

Full Adder

In	Out	Delay
A,B	Sum	60ps
Cin	Sum	30ps
A,B	Cout	90ps
Cin	Cout	60ps

Logic Unit

In	Out	Delay
A,B	out	110ps
R	out	10ps

2-to-1 Multiplexor

In	Out	Delay
A,B,C	out	60ps
R S	out	80ps

2 pages for handout

More Finite State Machines: Multiplication as an Example

Lab 4

Exam 1

Learning Objectives

- Design a Finite State Machine that controls a datapath
- Use Register Transfer Language to describe how state is modified each clock cycle

Step 1: Understand your problem

We can perform binary multiplication using the same algorithm as decimal multiplication

```
    1101 (13)
  x 1010 (10)
  ------------
    0000
   1101
  0000
 1101
 ------------
 10000010 (130)
```

a) Add 0000

b) Add 1101

c) Add 1010

Multiplying two unsigned N-bit integers produces at most a 2N-bit integer

```
   1111 (15)
x  1111 (15)
-----------
   1111
  1111
 1111
+1111
--------------
11100001 (225)
```

Multiplication can be described as a series of shift and add operations

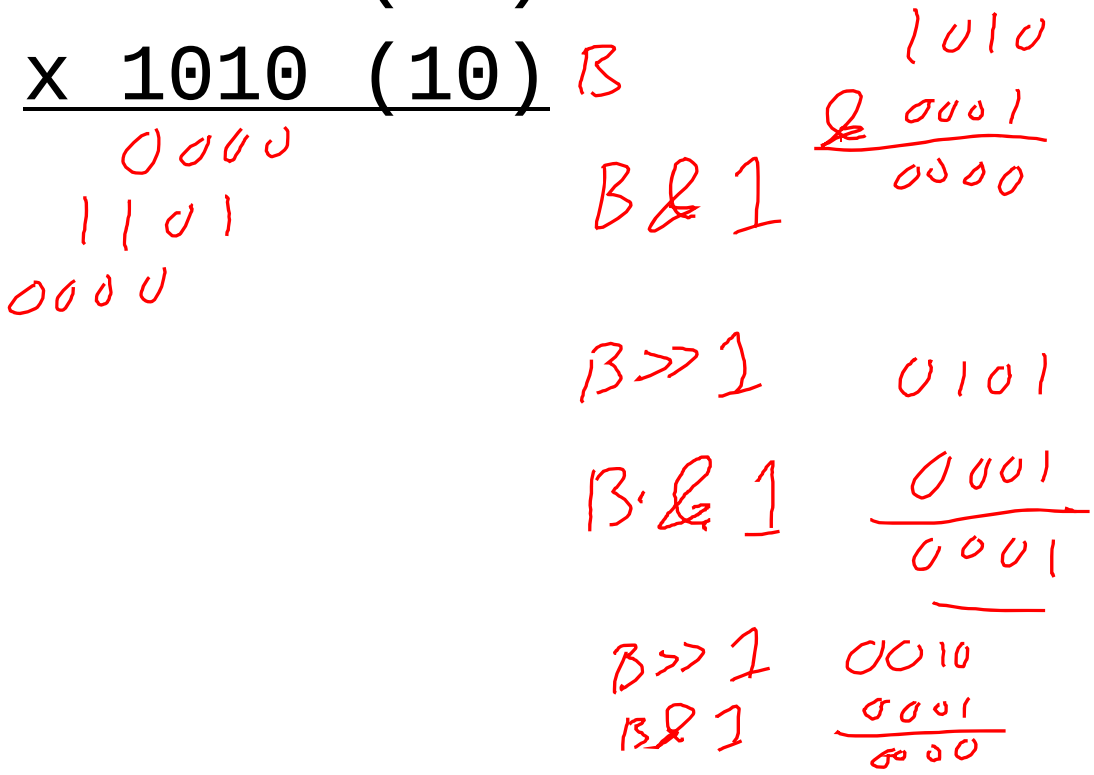
```
  00001101 (13)   A
     x 1010 (10)
  ---------
     0000
  00011010          A << 1
     0000
+ 01101000          A << 3
  ---------
  10000010 (130)   P = (A << 1) + (A << 3)
```

We determine whether to add each iteration by one-bit from the multiplicand

```
    1101 (13)
x  1010 (10)
-------------
    0000
   1101
  0000
+1101
-------------
10000010 (130)
```

We can use bitshifting and bitmasks to extract one bit from the multiplicand for control

```
   1101 (13)
x 1010 (10)
```



You try another example

```
  1001 (9)
x 0011 (3)
```

You try another example

```
   1001 (9)
x  0011 (3)
-------------
   1001
  1001
  0000
+0000
-------------
00011011 (27)
```

0 0 0 1
0 0 0 0

Write some C code that can perform the shift-add multiply algorithm

```
   1101 (13) A
x  1010 (10) B
-------------
   0000
  1101
 0000
+1101
-------------
10000010 (130) P
```

```
int shift_add_multiply(unsigned short X,
unsigned short Y) {
    unsigned int A = (unsigned int) X;
    unsigned short B = Y;
    unsigned int P = 0;

    return P;
}
```

```
int shift_add_multiply(unsigned short X, unsigned short Y) {
   unsigned int A = (unsigned int) X;
   unsigned short B = Y;
   unsigned int P = 0;

   while (B != 0) {
      if ((B & 1) == 1) {
         P = P + A;
      }
      B = B >> 1;
      A = A << 1;
   }
   return P;
}
```

0011

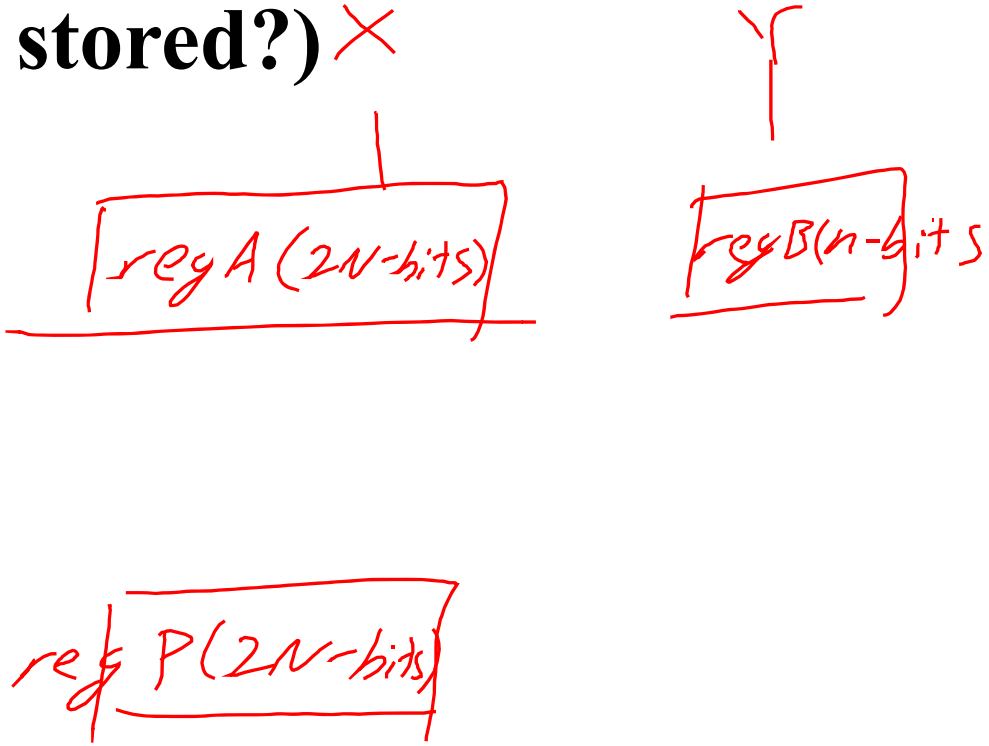
0001

0000

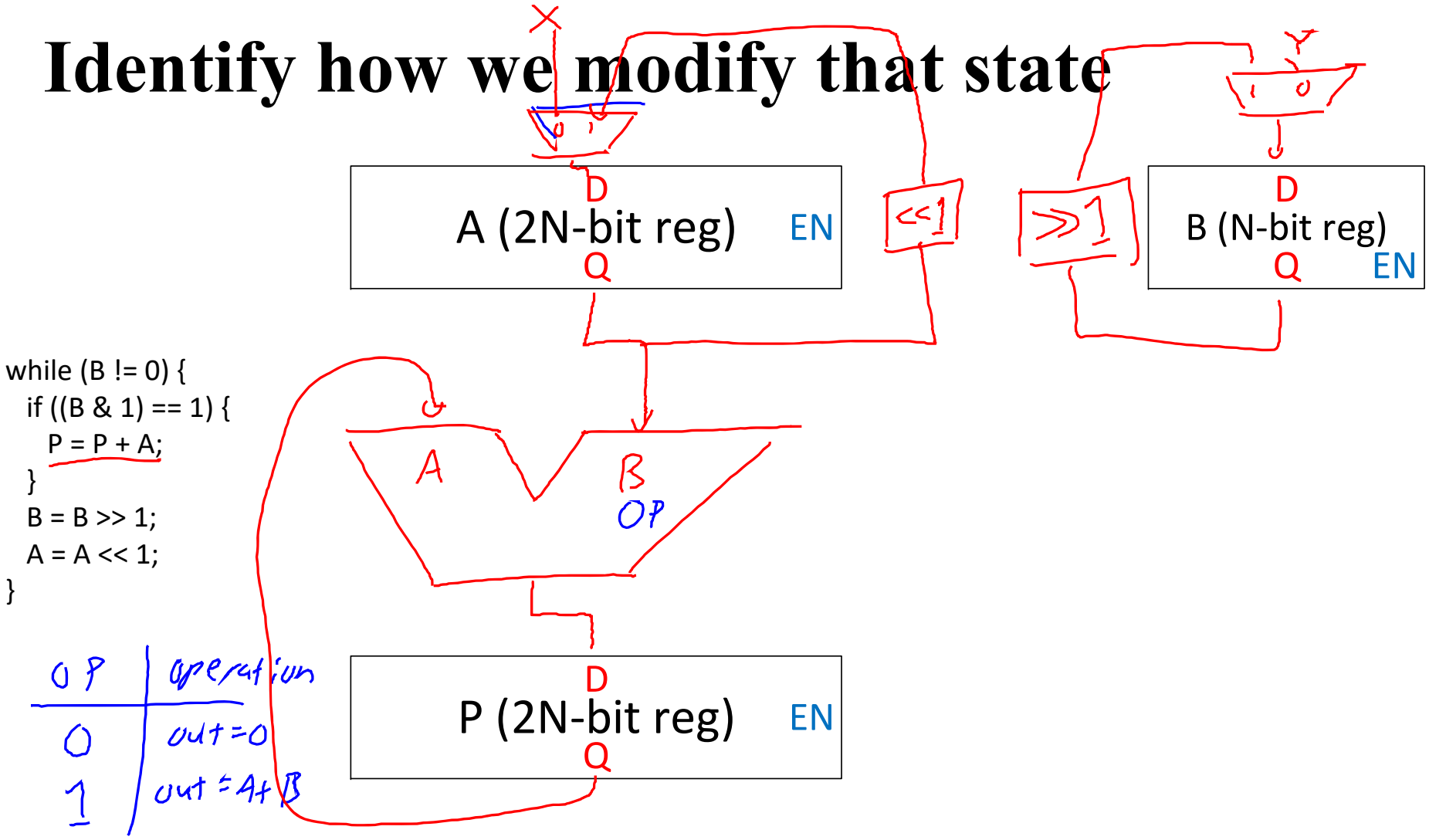
Step 2: Understand the datapath

Identify needed state components from variables (what state is stored?)

```
int shift_add_multiply(…) {
   unsigned int A = (unsigned int) X;
   unsigned short B = Y;
   unsigned int P = 0;

   return P;
}
```



Identify how we modify that state

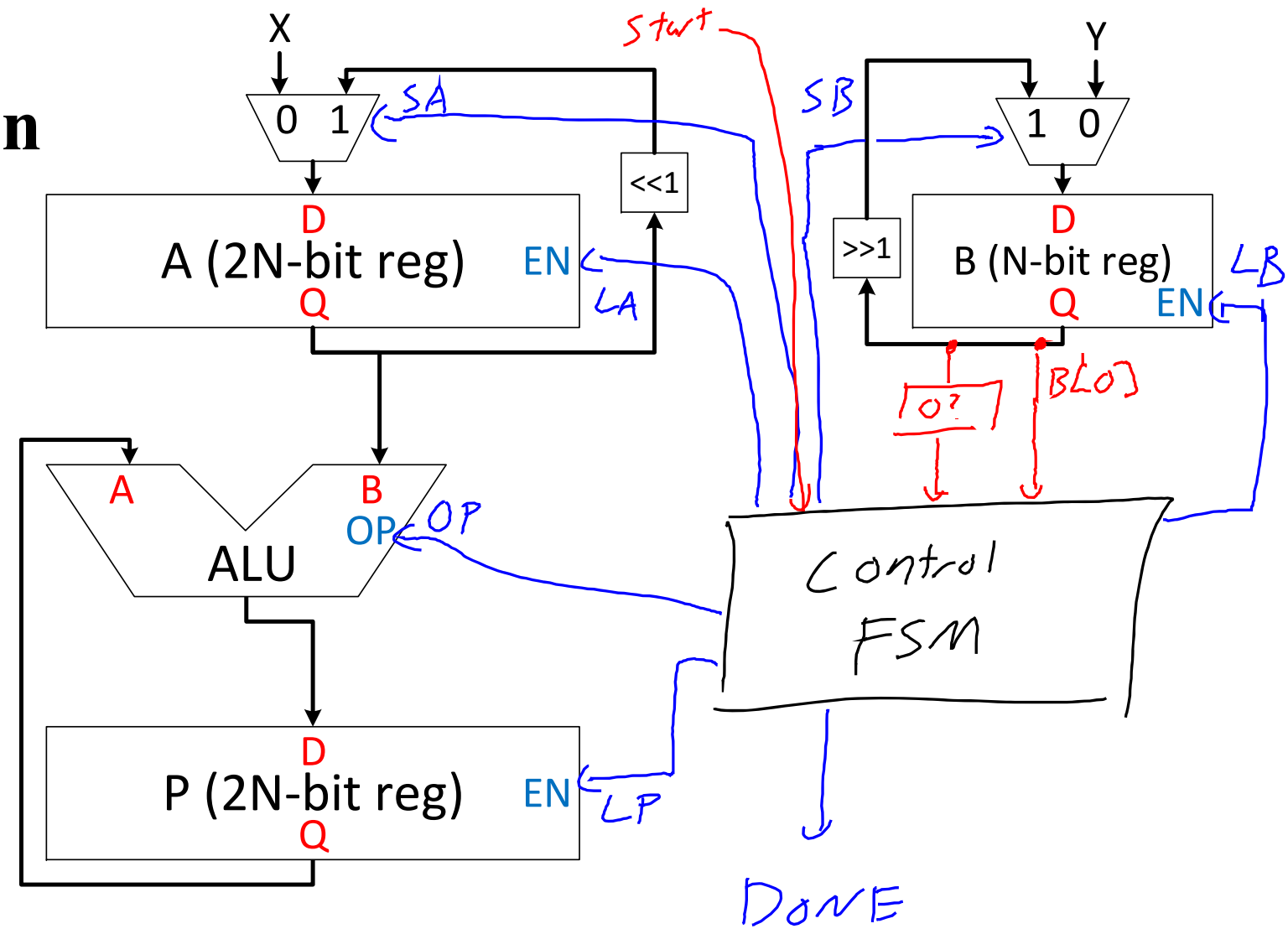



```
while (B != 0) {
  if ((B & 1) == 1) {
    P = P + A;
  }
  B = B >> 1;
  A = A << 1;
}
```

OP	operation
0	out = 0
1	out = A + B

Define specification for control FSM

```
while (B != 0){
  if ((B & 1) == 1){
    P = P + A;
  }
  B = B >> 1;
  A = A << 1;
}
```

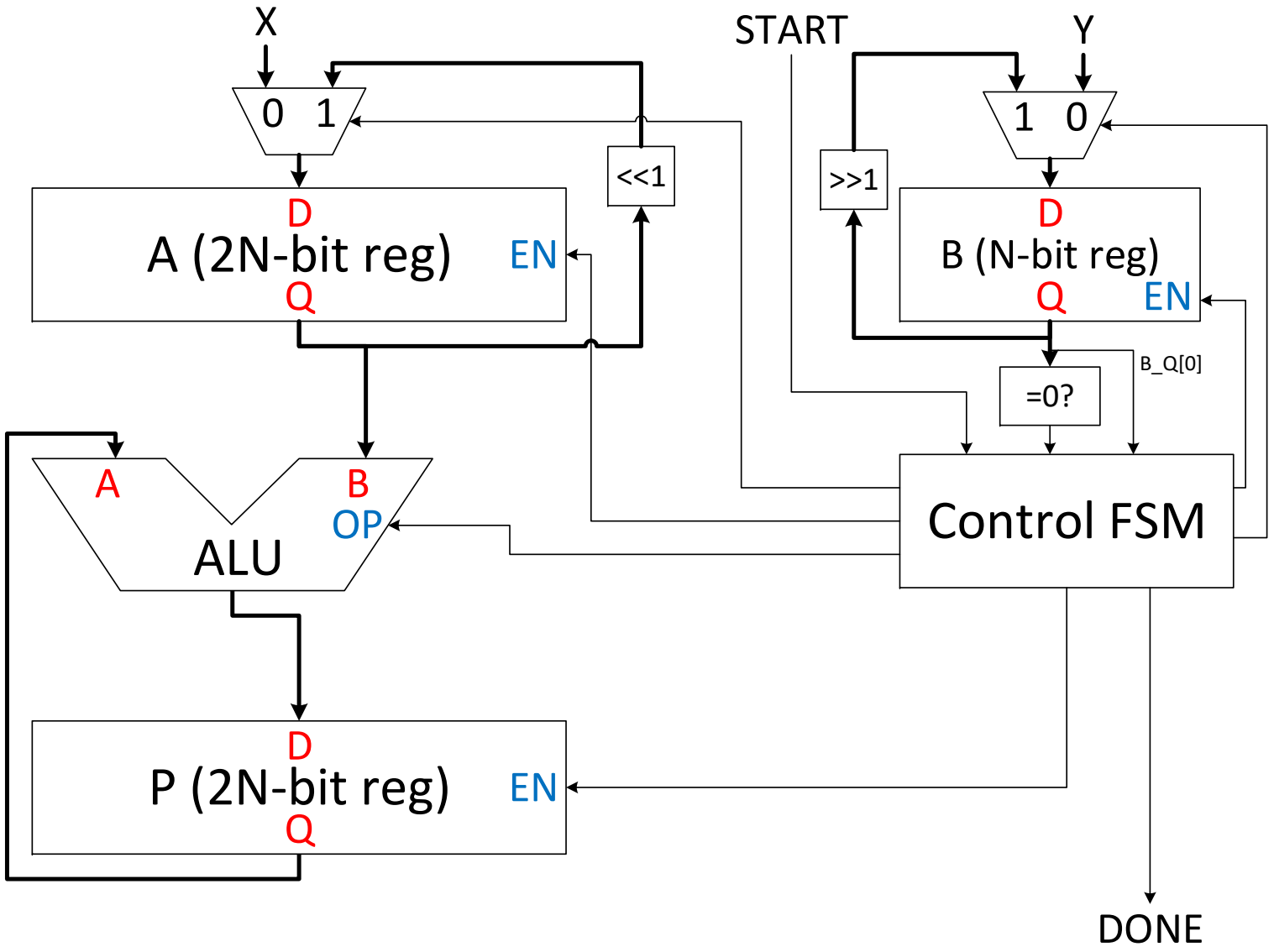


Control FSM has 3 inputs and 7 outputs

- B is zero (Z)
- B_Q[0] is 1 (B[0])
- START

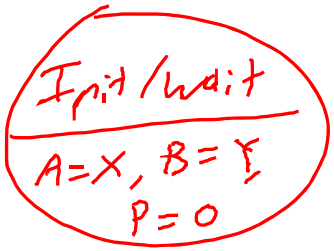
- Load A (LA)
- Shift A (SA)
- Load B (LB)
- Shift B (SB)
- Add when 1/Zero when 0 (OP)
- Load P (LP)
- DONE

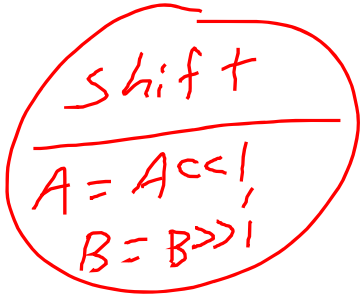
Control and Datapath

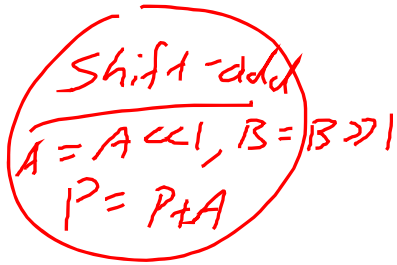


Step 3: Design your FSM (don’t forget timing!)

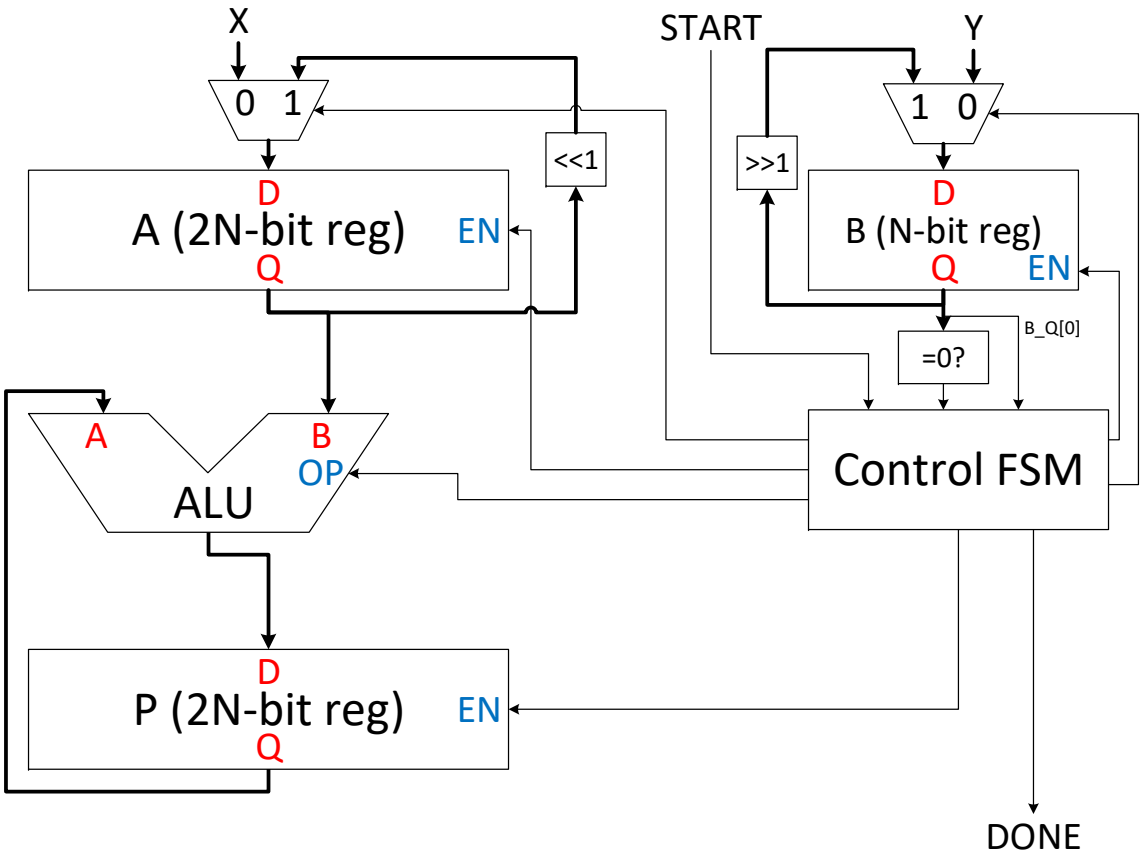
```
int shift_add_multiply(…) {
  unsigned int A = (unsigned int) X;
  unsigned short B = Y;
  unsigned int P = 0;

  while (B != 0) {
    if ((B & 1) == 1) {
      P = P + A;
    }
    B = B >> 1;
    A = A << 1;
  }
  return P;
}
```



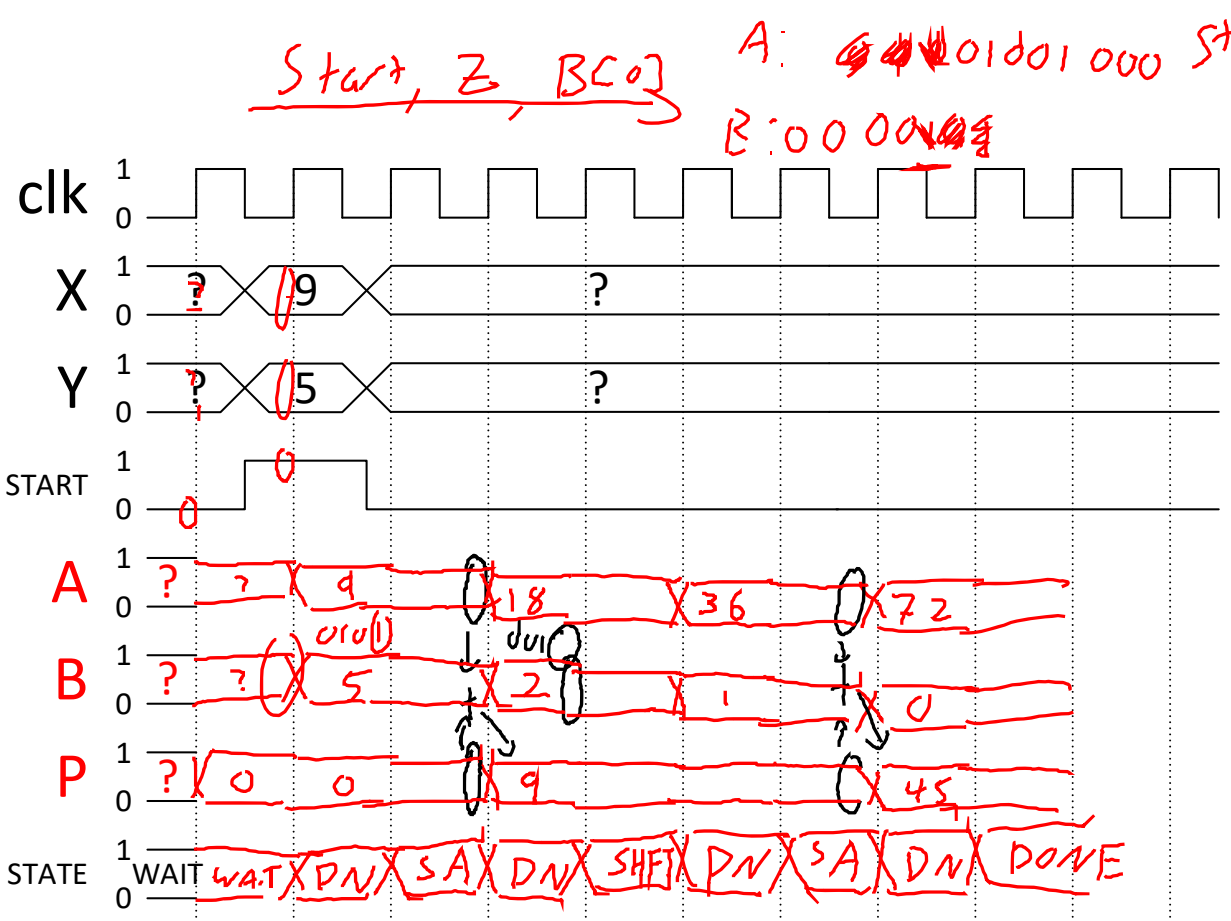


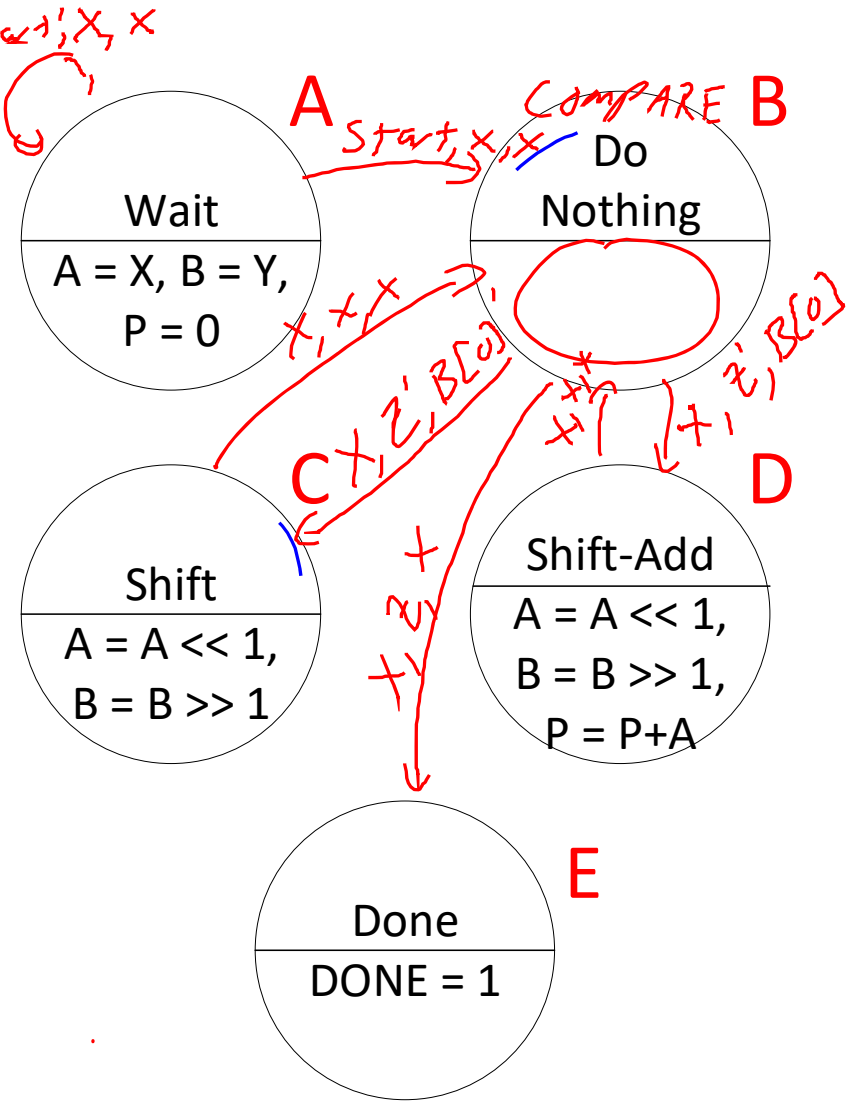


Register transfers show source state elements, operations, and destination state elements

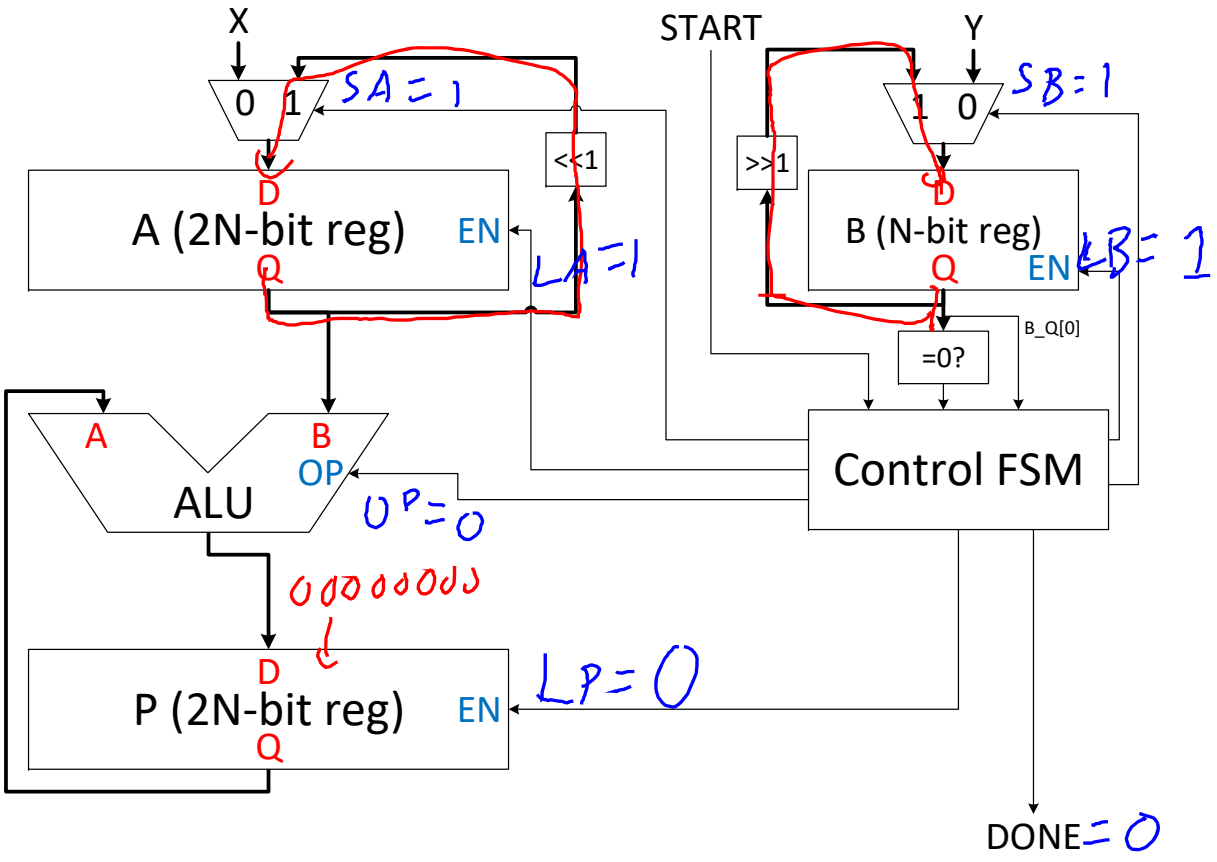


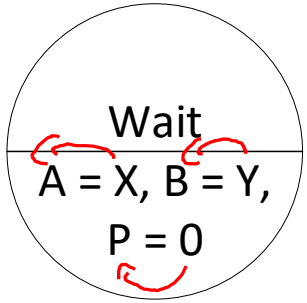

```
A = X
B = Y
P = 0
A = A << 1
B = B >> 1
P = P + A
```

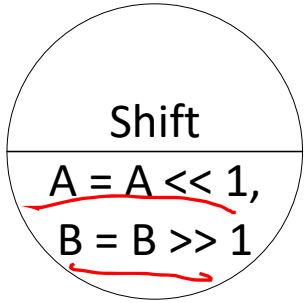

clk
X
Y
START
A
B
P
STATE
WAIT

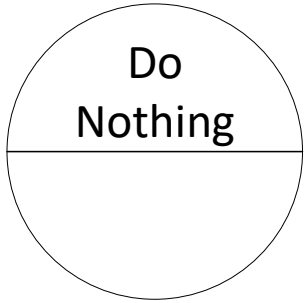

A
Wait
A = X, B = Y,
P = 0
B
Do
Nothing
C
Shift
A = A << 1,
B = B >> 1
D
Shift-Add
A = A << 1,
B = B >> 1,
P = P+A
E
Done
DONE = 1

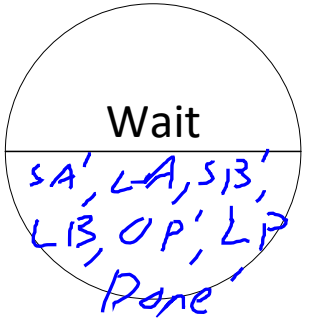
Translate register transfers into control signals

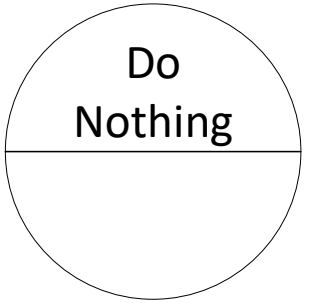


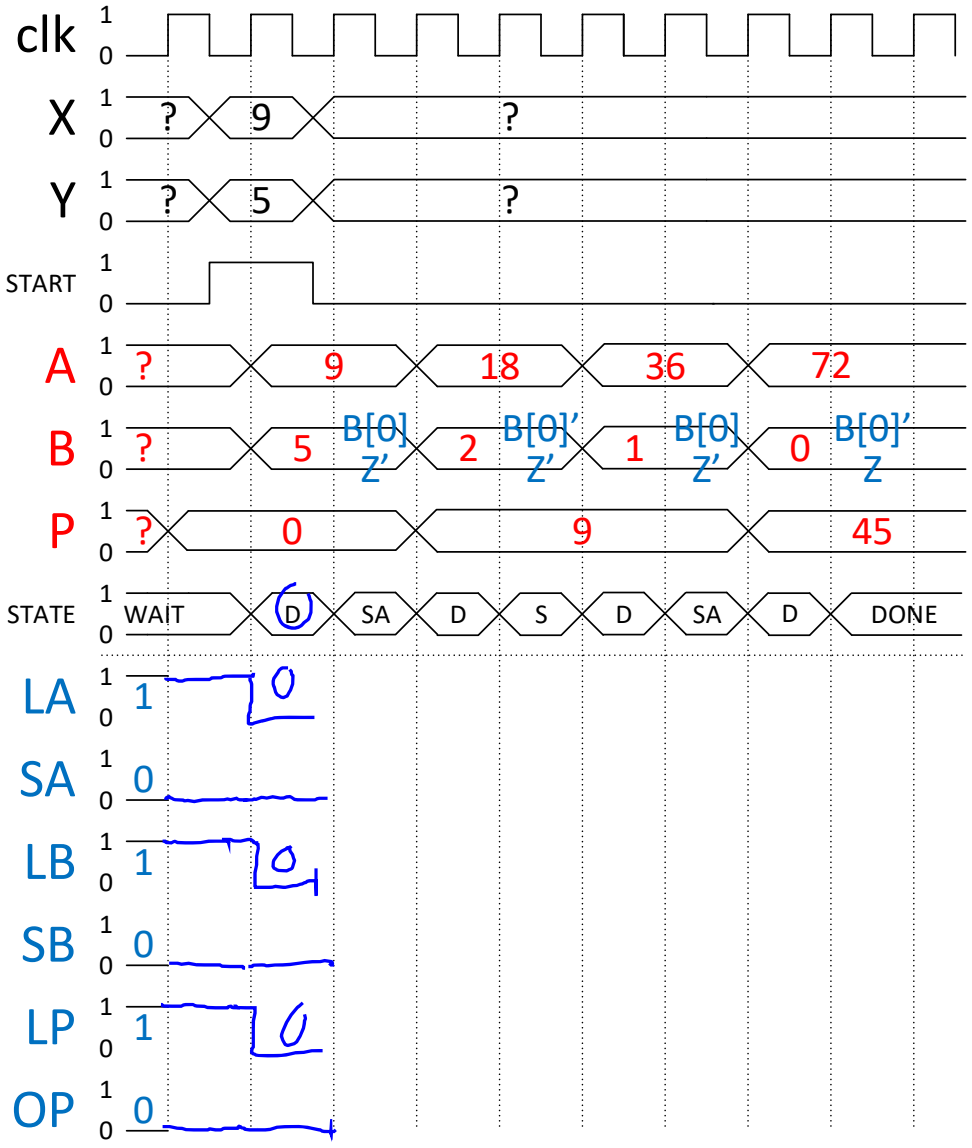


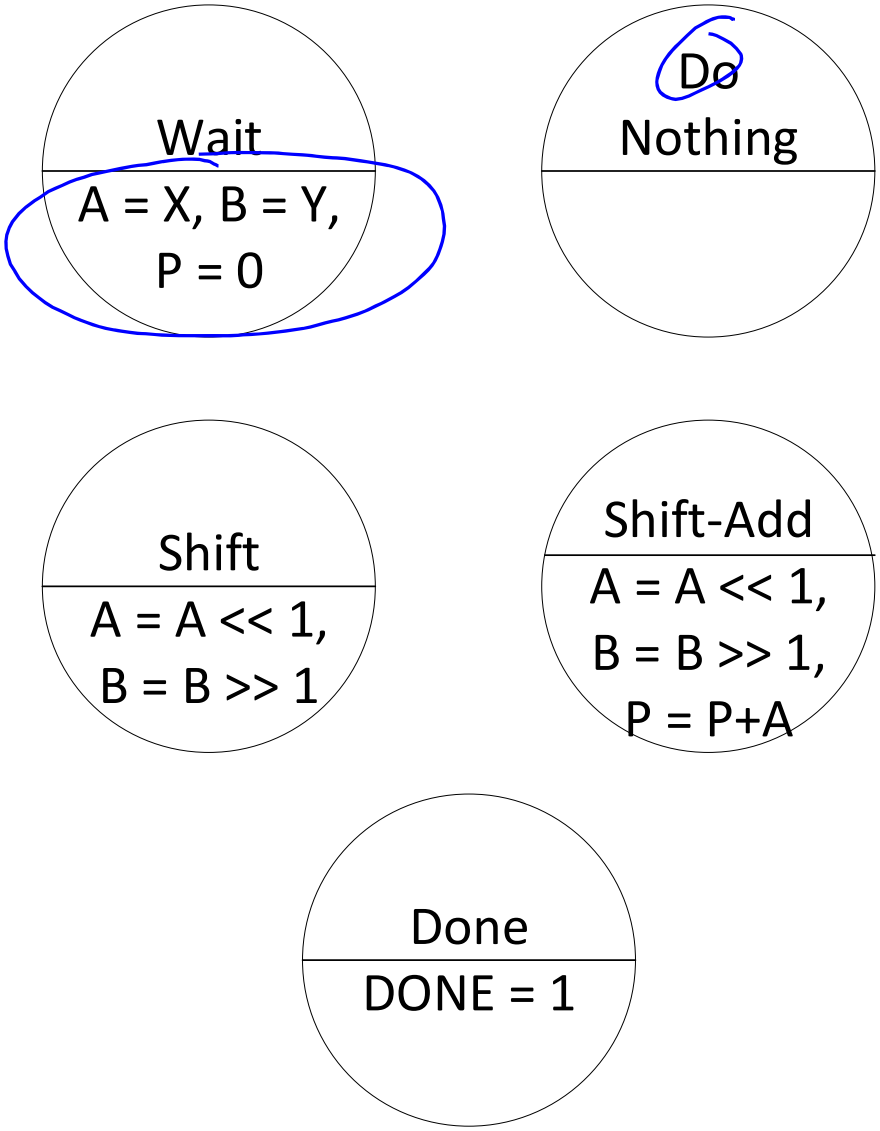









clk
X
Y
START
A
B
P
STATE
LA
SA
LB
SB
LP
OP
WAIT
D
SA
D
S
D
SA
D
DONE


Wait
A = X, B = Y,
P = 0
Do
Nothing
Shift
A = A << 1,
B = B >> 1
Shift-Add
A = A << 1,
B = B >> 1,
P = P+A
Done
DONE = 1