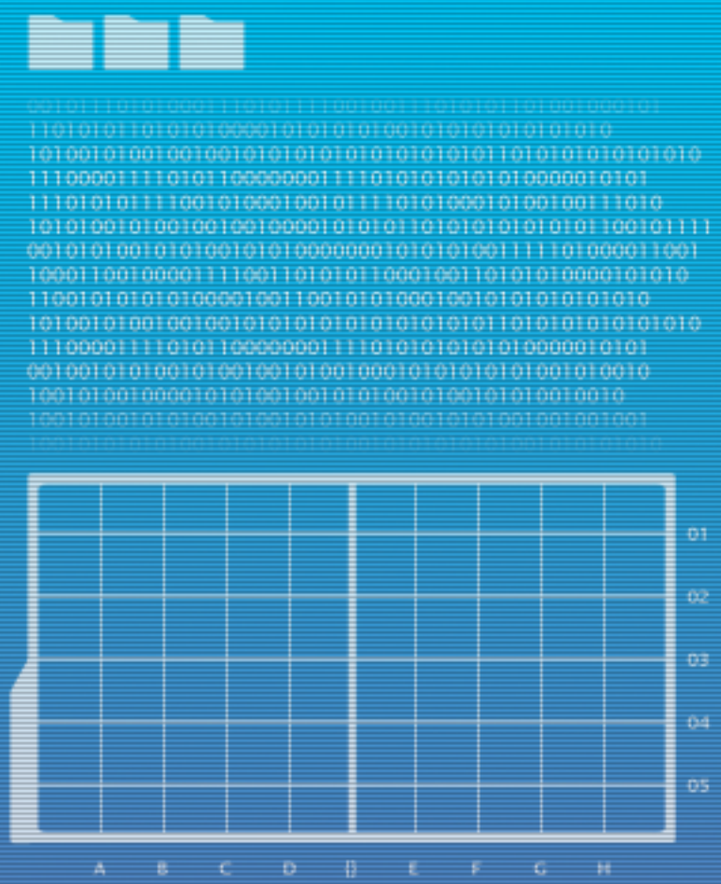
DEPARTMENT OF
INFORMATION SYSTEMS
AND COMPUTER SCIENCE



Datapaths

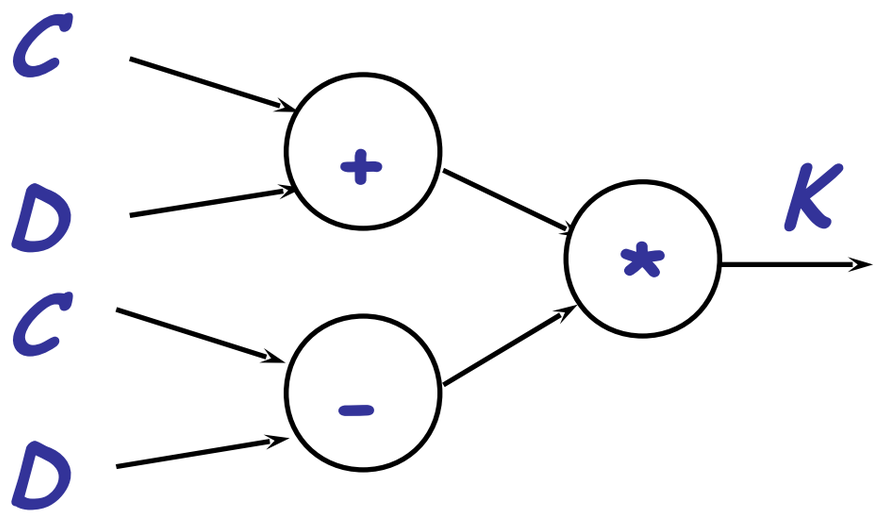
Because Having Infinite Hardware is (Currently) Impossible

Computer = Programmable Finite State Machine

Given an expression:

$$K = (C + D) * (C - D)$$

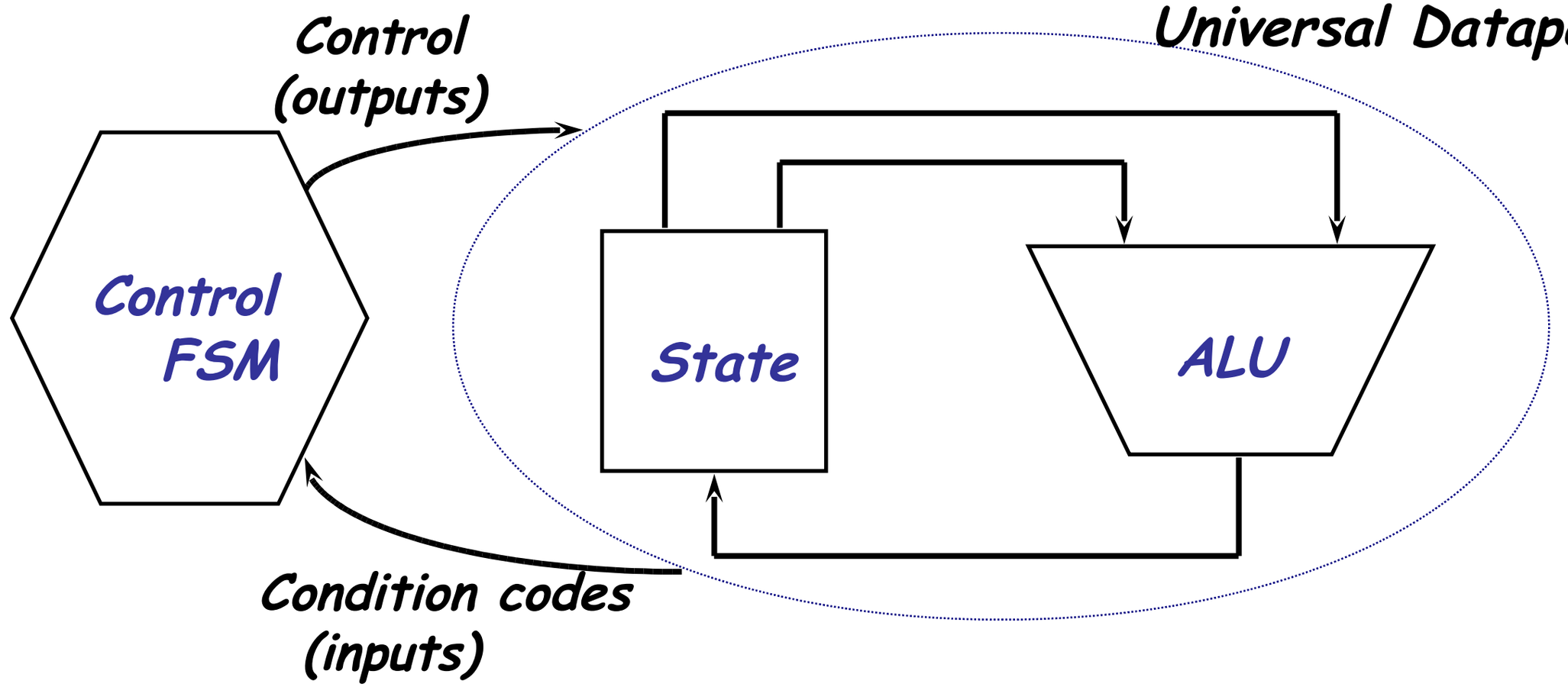
Can build a circuit for that expression:



Circuit size proportional to size of expression;
And this time, it's not just addition involved,
so the sequential adder from last time can't be used.

Abstraction: Control/Datapath

Compute the expression using a *sequence* of commands to a single function unit called the *datapath*.



We still want some kind of sequence + loop so that the size of the datapath (might be) constant no matter what expression should be computed!

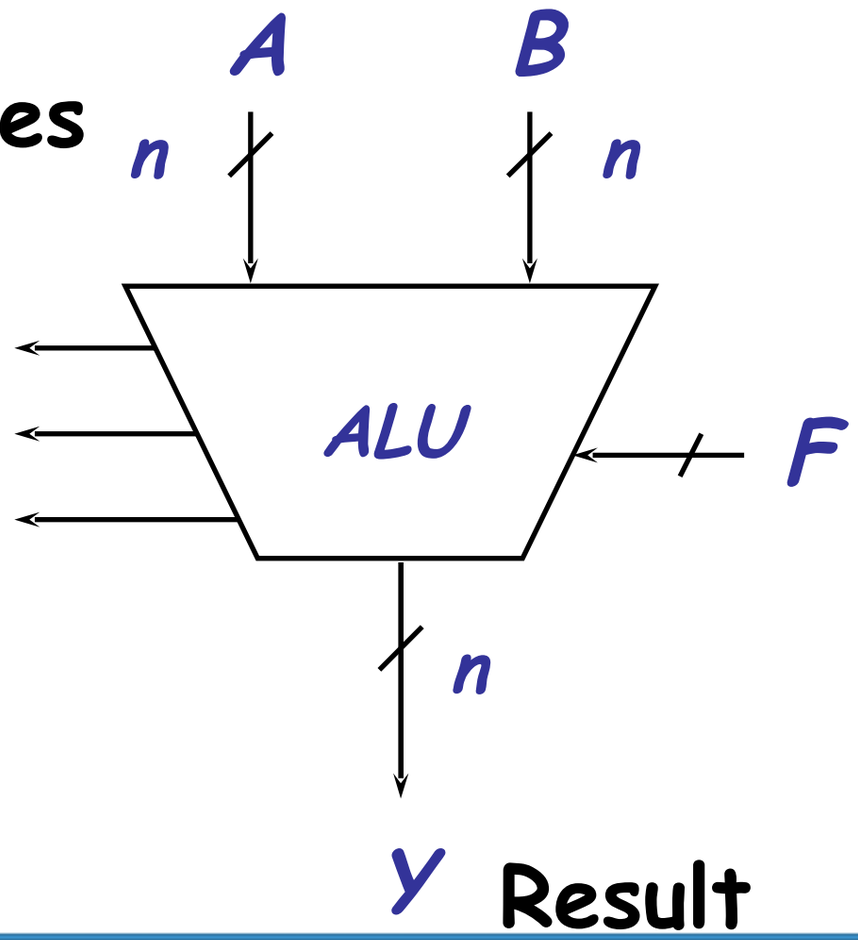
Arithmetic/Logic Unit (ALU)

Purely combinational logic, definitely not a ROM.

Note: this is NOT the same as the ALU in lab!

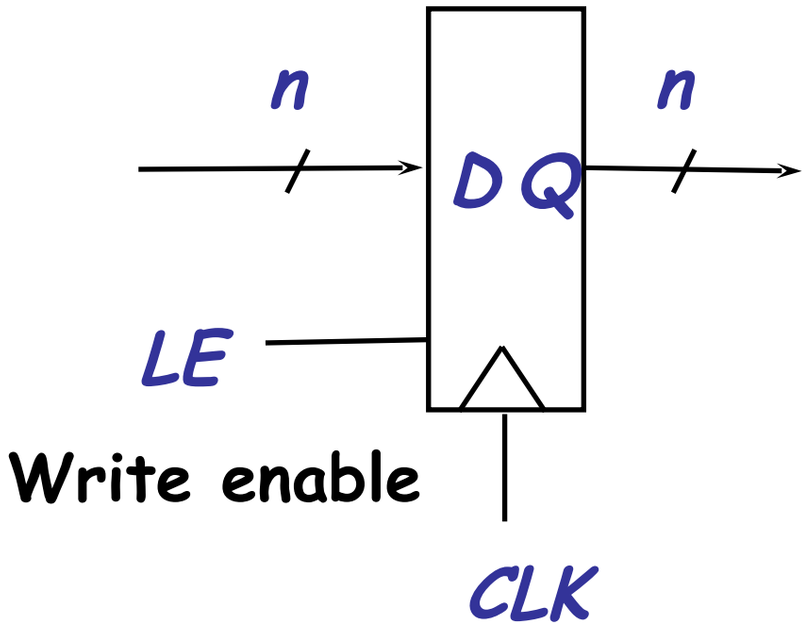
Condition codes (aka "flags")

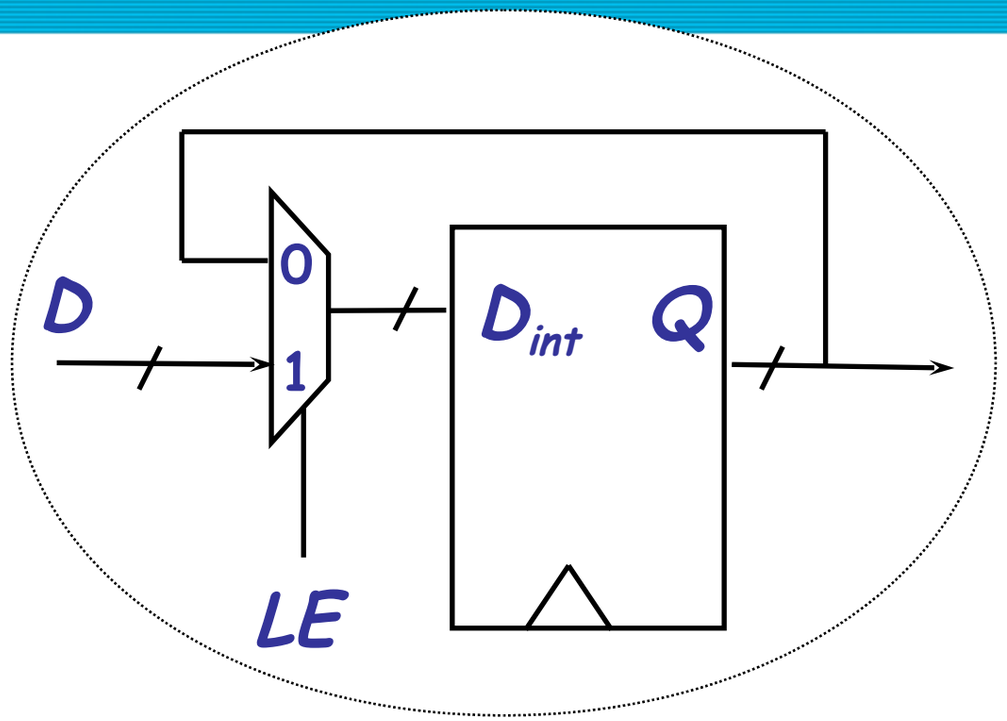
- **V**: overflow
- **N**: Y < 0
- **Z**: Y = 0



F	Y
000	A + B
001	A - B
010	A - 1
011	A and B
100	A or B
101	A * B
110	A
111	B

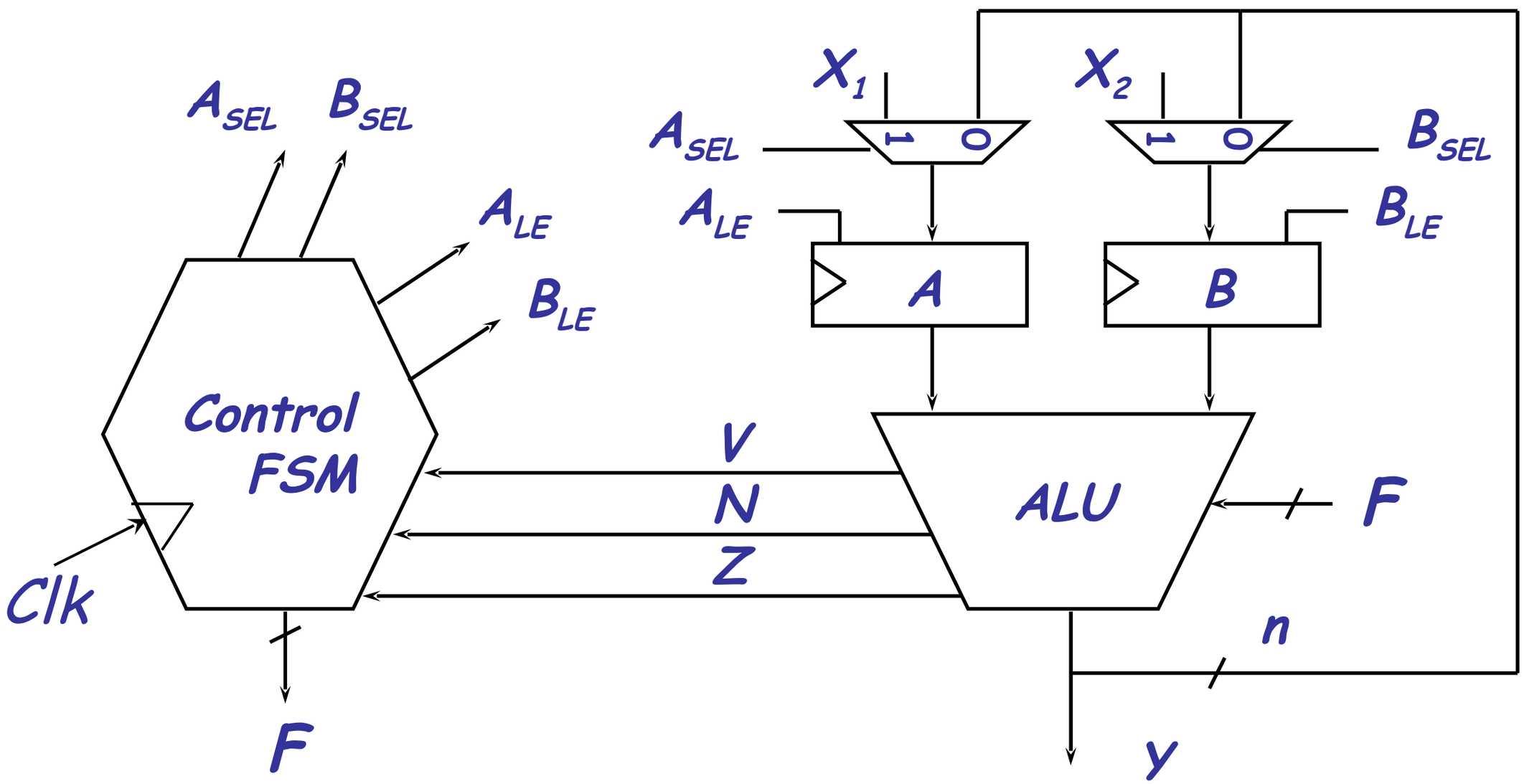
Register w/ Enable





- Only samples input on clock edge when LE = 1.
- LE input = EN input (remember our counter FSM?)
- Usually implemented by adding a mux as shown above.
  - Can also be done by ANDing CLK, but harder to analyze because of the added clock skew/delay.
- D and LE must obey (equivalent) setup time constraints.
  - Delayed input! (remember blackboxing FFs?)

Our First Datapath



How can we compute K = (C + D) * D?

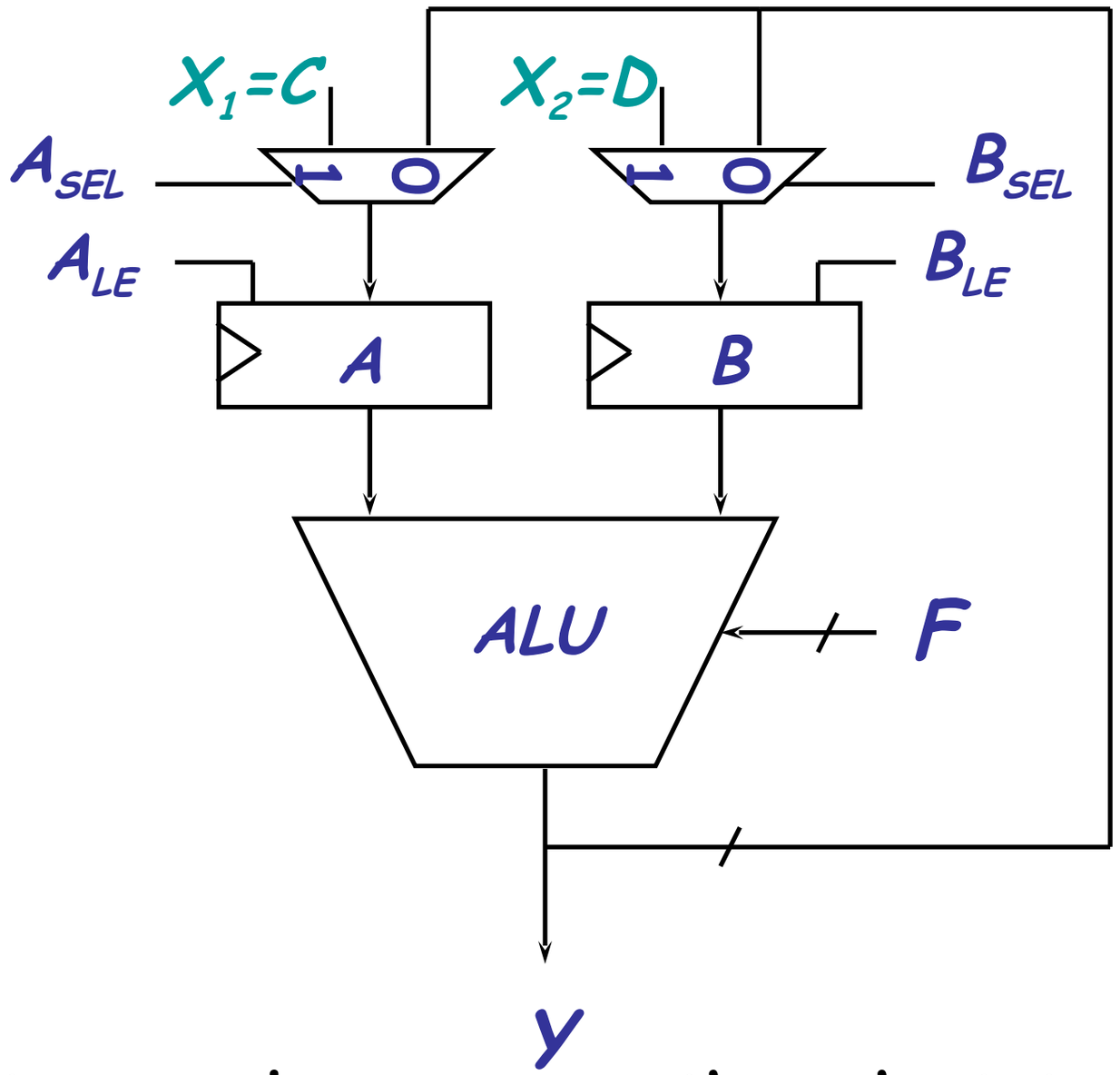
Control FSM for K = (C + D) * D

Assume Inputs appear at X_1 , X_2 in first clock cycle alone

Control FSM

current state → outputs

	A_{SEL}	B_{SEL}	A_{LE}	B_{LE}	F
S_1					
S_2					
S_3					



Note: this control FSM has no inputs, merely sequences through states

Control FSM for K = (C + D) * D

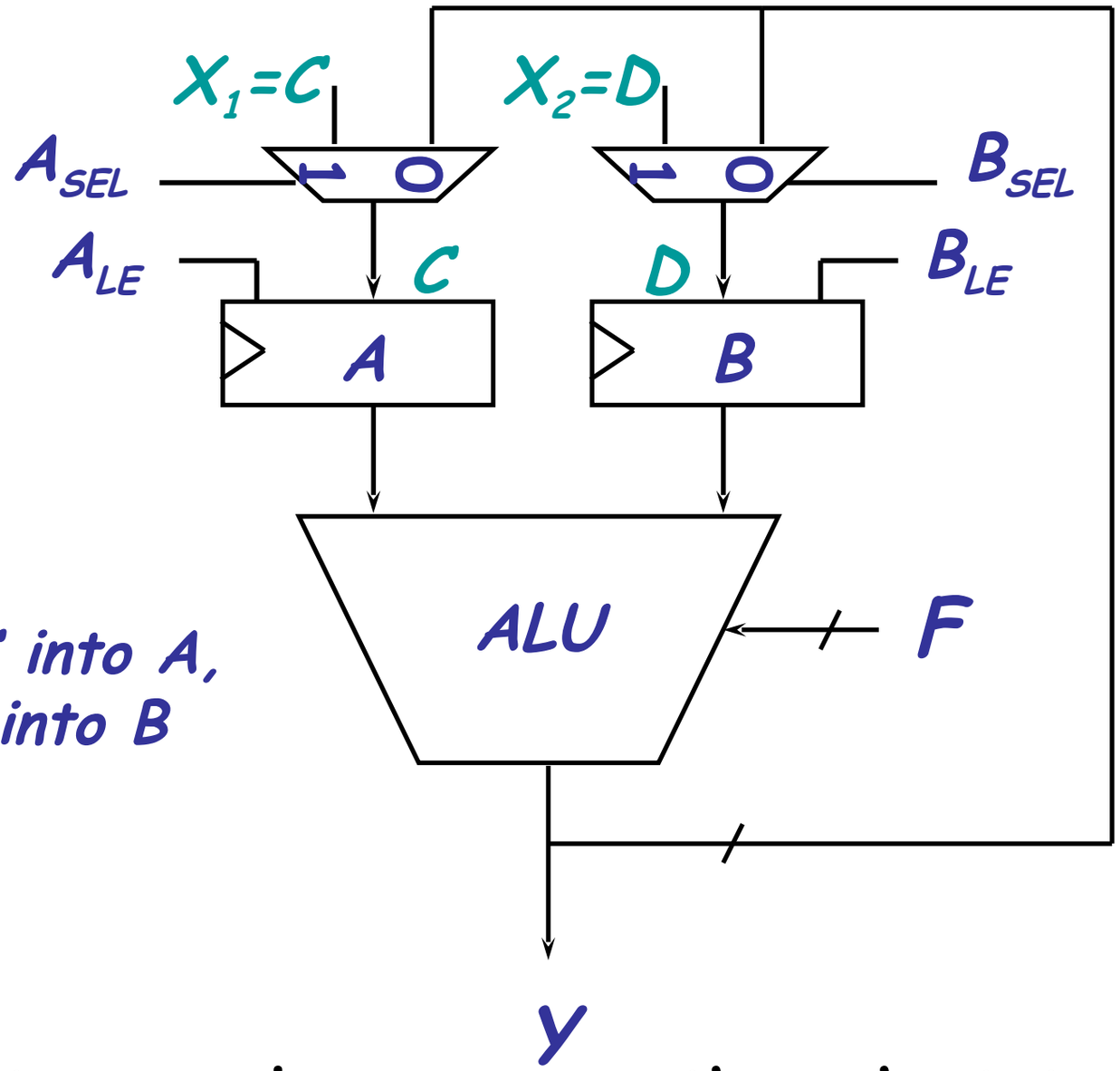
Step 1: Load C and D into A and B by using A_{LE} and B_{LE}

Control FSM

current state → outputs

	A_{SEL}	B_{SEL}	A_{LE}	B_{LE}	F
s_1	1	1	1	1	–
s_2					
s_3					

Load C into A, and D into B



Note: this control FSM has no inputs, merely sequences through states

Control FSM for K = (C + D) * D

Step 2: Add contents of A and B and put into A

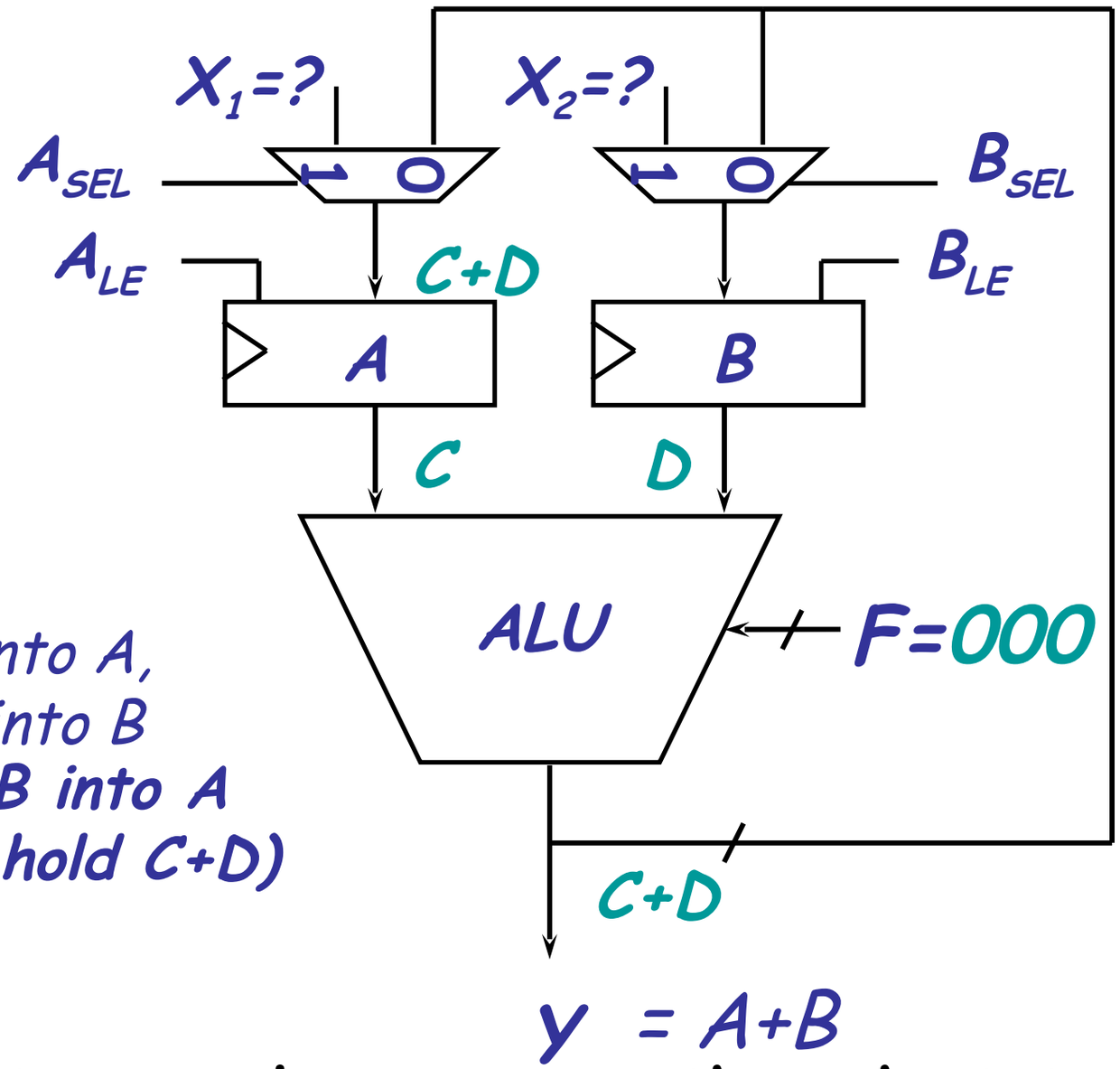
Control FSM

current state → outputs

	A_{SEL}	B_{SEL}	A_{LE}	B_{LE}	F
S_1	1	1	1	1	–
S_2	0	-	1	0	A+B
S_3					

Put C into A, and D into B

Put A+B into A (A will hold C+D)



Note: this control FSM has no inputs, merely sequences through states

Control FSM for K = (C + D) * D

Step 3: Multiply contents of A and B and put into A

Control FSM

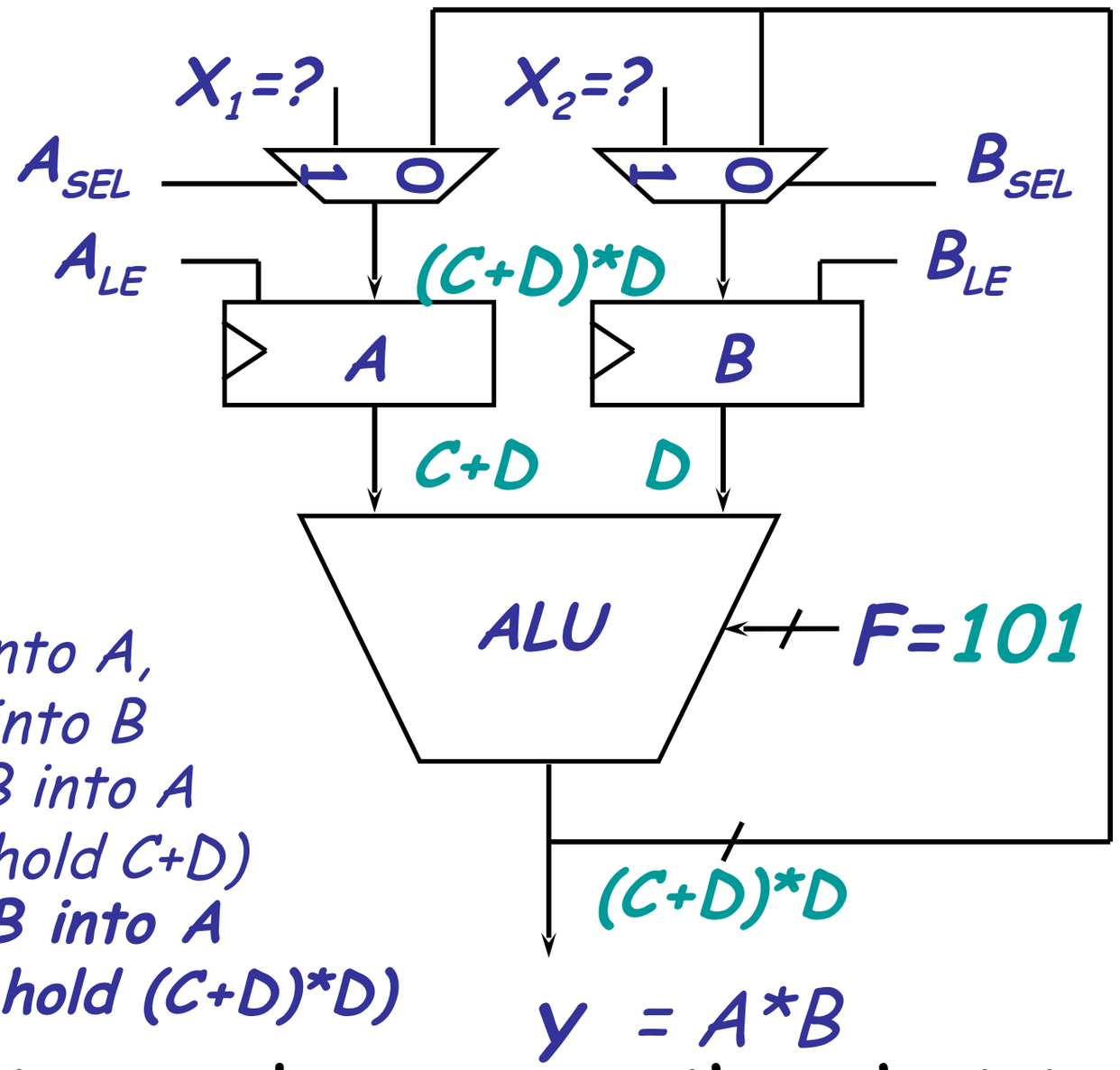
current state → outputs

	A_{SEL}	B_{SEL}	A_{LE}	B_{LE}	F
S_1	1	1	1	1	–
S_2	0	-	1	0	A+B
S_3	0	-	1	0	A*B

Put C into A, and D into B

Put A+B into A (A will hold C+D)

Put A*B into A (A will hold (C+D)*D)



Note: this control FSM has no inputs, merely sequences through states

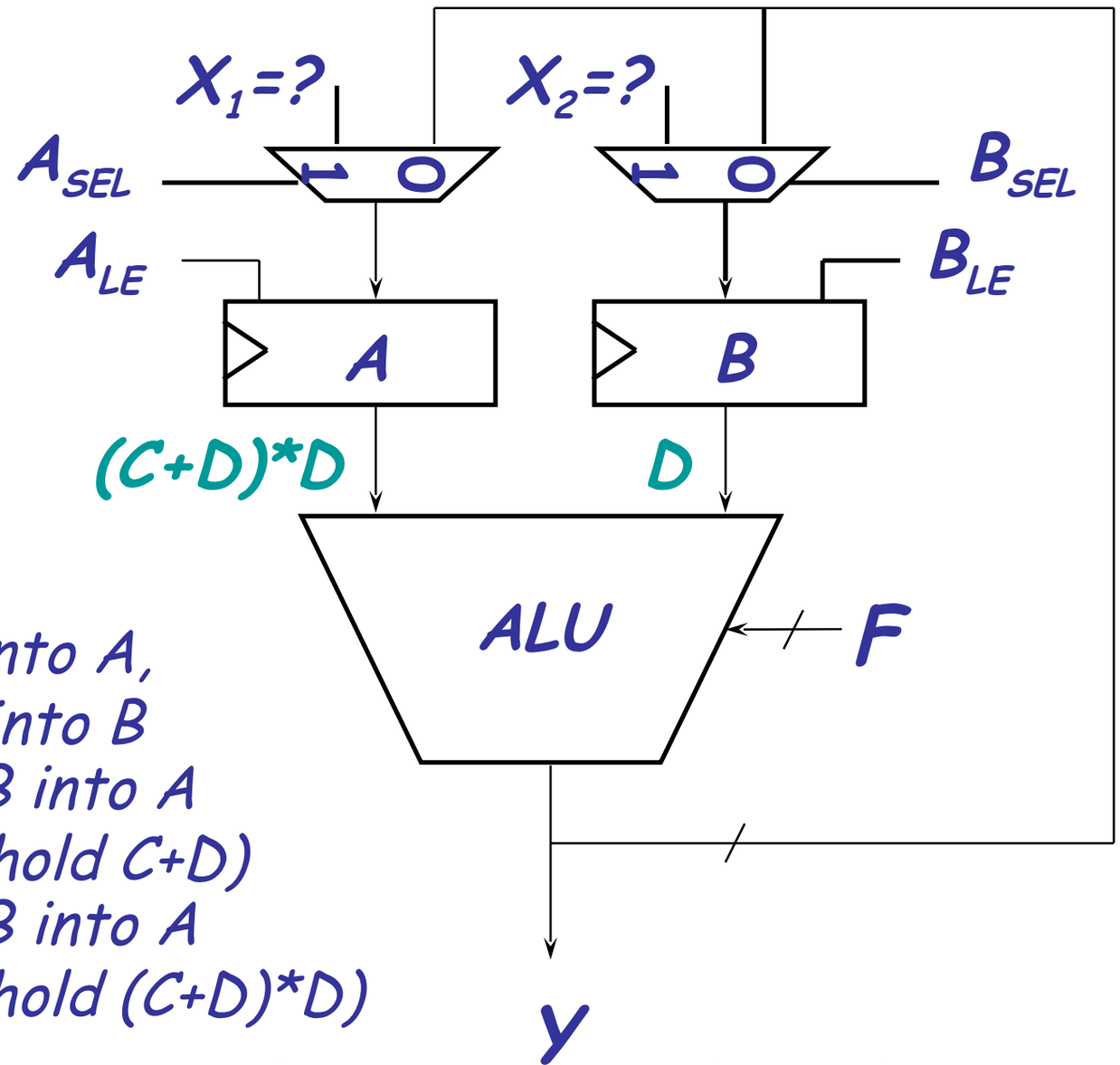
Control FSM for K = (C + D) * D

After clock edge at end of Step 3: A now holds (C+D)*D

Control FSM

current state → outputs

	A_{SEL}	B_{SEL}	A_{LE}	B_{LE}	F	
s_1	1	1	1	1	–	Put C into A, and D into B
s_2	0	-	1	0	A+B	Put A+B into A (A will hold C+D)
s_3	0	-	1	0	A*B	Put A*B into A (A will hold (C+D)*D)



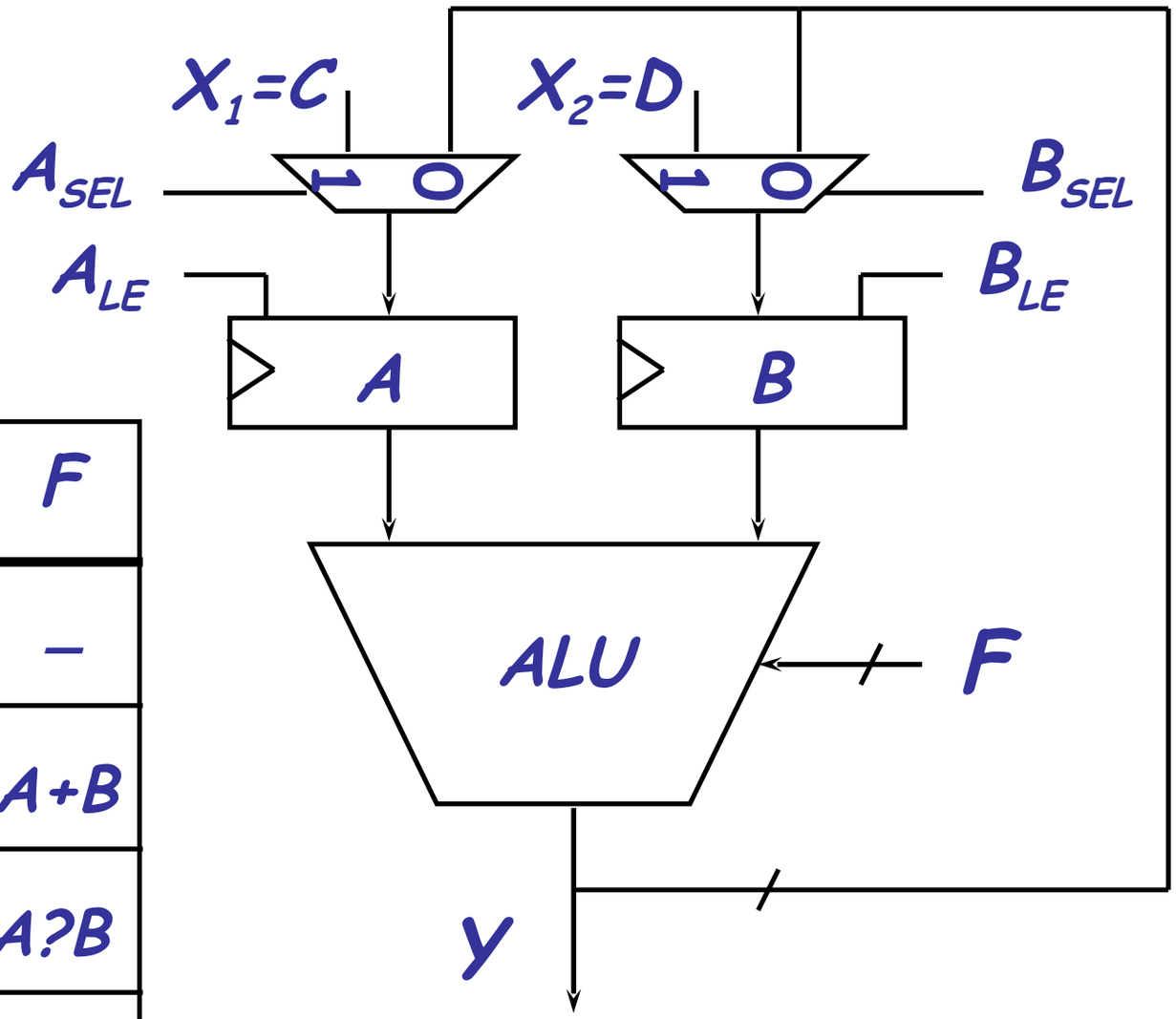
Note: this control FSM has no inputs, merely sequences through states

Control FSM for K = (C + D) * (C - D)

Control FSM

current state → outputs

	A_{SEL}	B_{SEL}	A_{LE}	B_{LE}	F
s_1	1	1	1	1	–
s_2	0	-	1	0	A+B
s_3	0	0	?	?	A?B



Where do we store C, D, C+D, C-D?

Storage Requirements

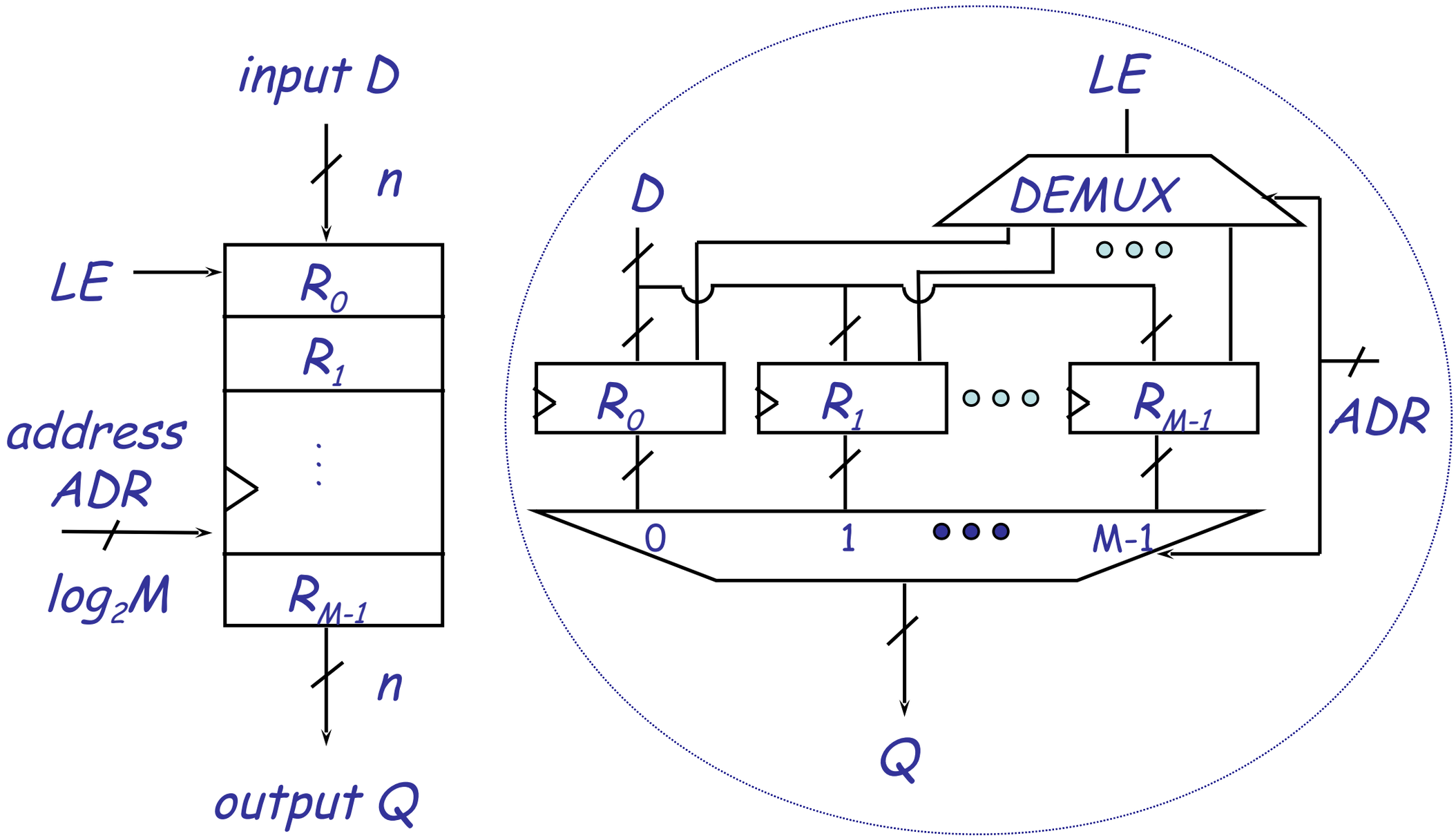
- How does Java deal with expressions like these?
  - One operation at a time.
  - Needs temporary storage!
- Closer look at expression:

$$K = (C + D) * (C - D)$$

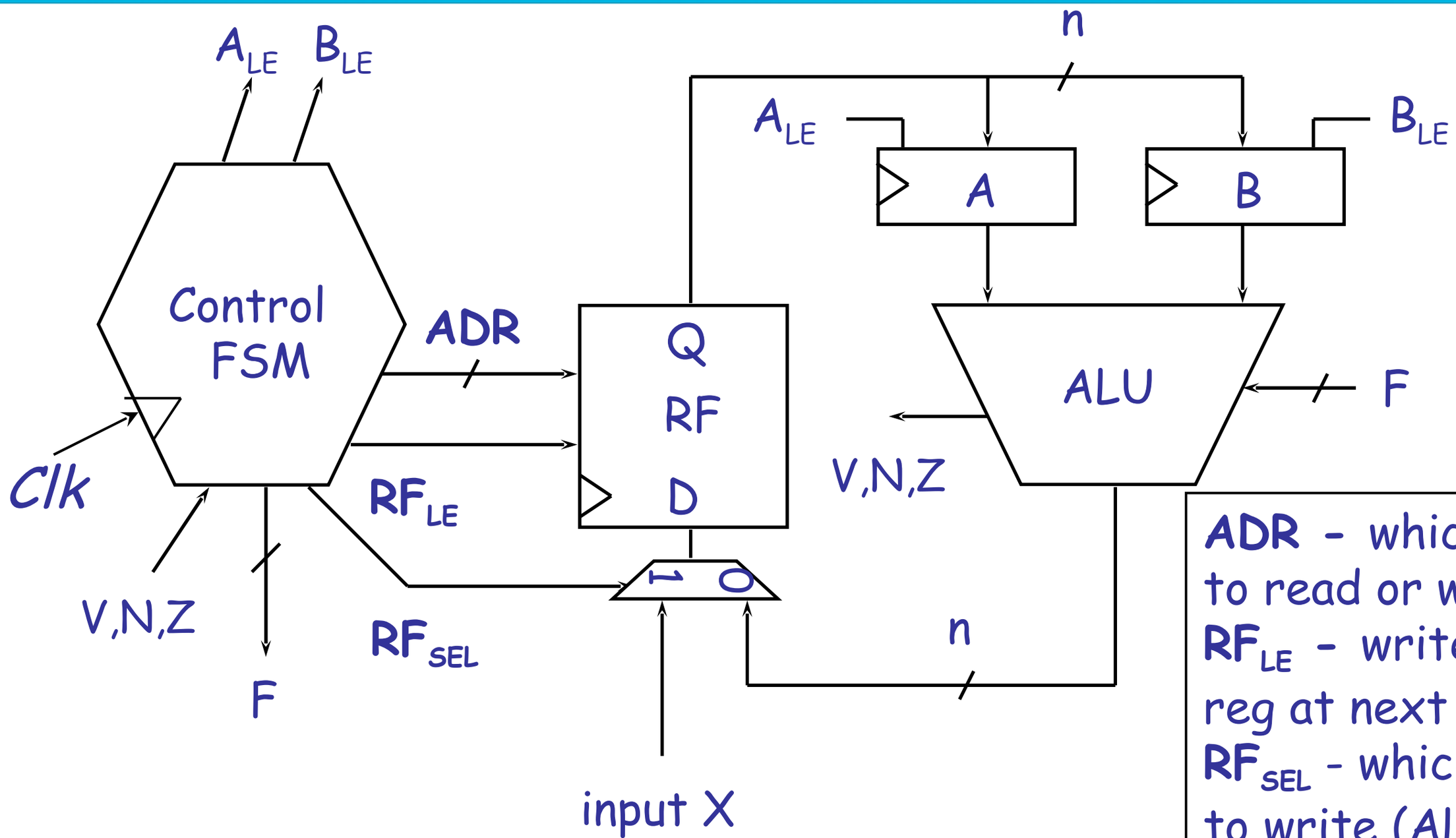
T1 *T2*

- Need to store *C*, *D*, *T1*, *T2* and possibly **K**.
  - **Maybe not at the same time, but we need to make sure that data we may need later is not lost or overwritten ("clobbered registers").**
- ***Need more than two registers!***

Register File (RF)



Our Second Datapath



ADR - which reg to read or write
RF_{LE} - write to reg at next edge
RF_{SEL} - which data to write (ALU or input X)

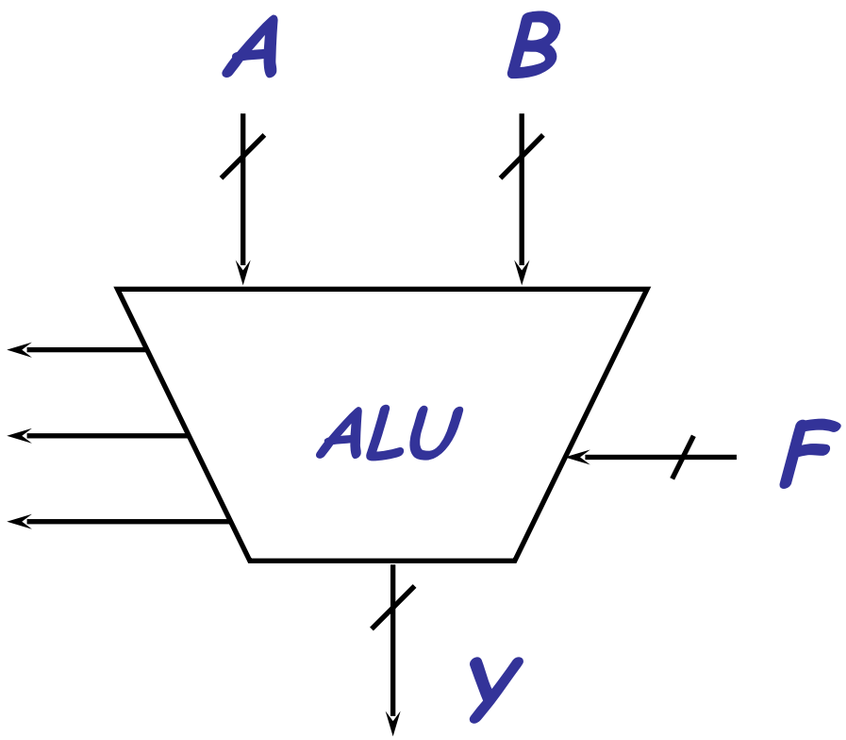
Control FSM for K = (C + D) * (C - D)

Assume input X holds C in cycle 1, and D in cycle 2

	RF_{SEL}	RF_{LE}	ADR	A_{LE}	B_{LE}	F	
s_1	1	1	000	0	0	–	Load input C (from X) in R_0
s_2	1	1	001	0	0	–	Load input D (from X) in R_1
s_3	-	0	000	1	0	–	Load R_0 value in A
s_4	-	0	001	0	1	–	Load R_1 value in B
s_5							**Put C+D in R_0**
s_6							**Put C-D in R_1**
s_7	-	0	000	1	0	–	Load R_0 value in A
s_8	-	0	001	0	1	–	Load R_1 value in B
s_9	0	1	000	0	0	A*B	Put (C+D)*(C-D) in R_0

What About Conditionals?

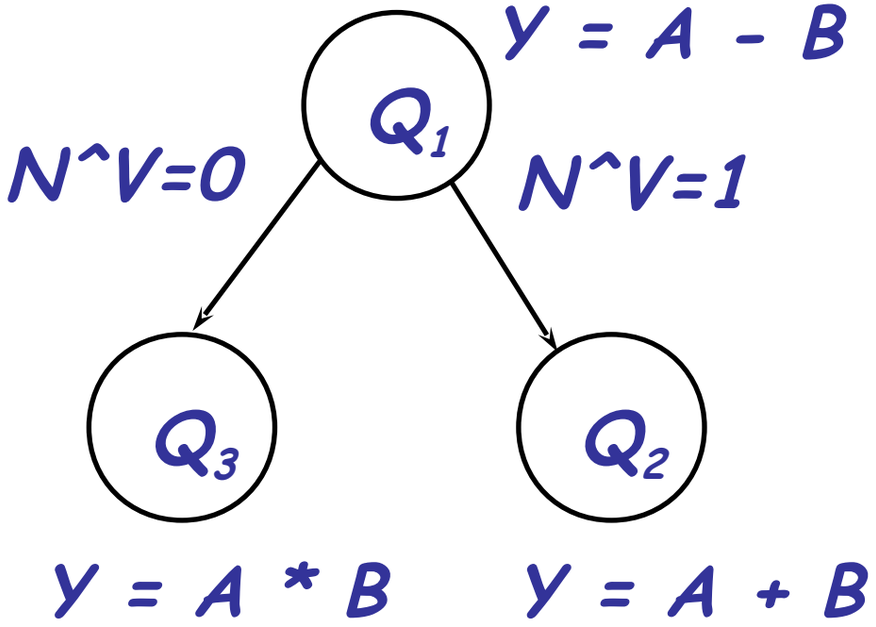
FSM for Control component is no longer only a sequence and now requires an input!



Condition codes

overflow V
Y < 0 N
Y = 0 Z

Control FSM



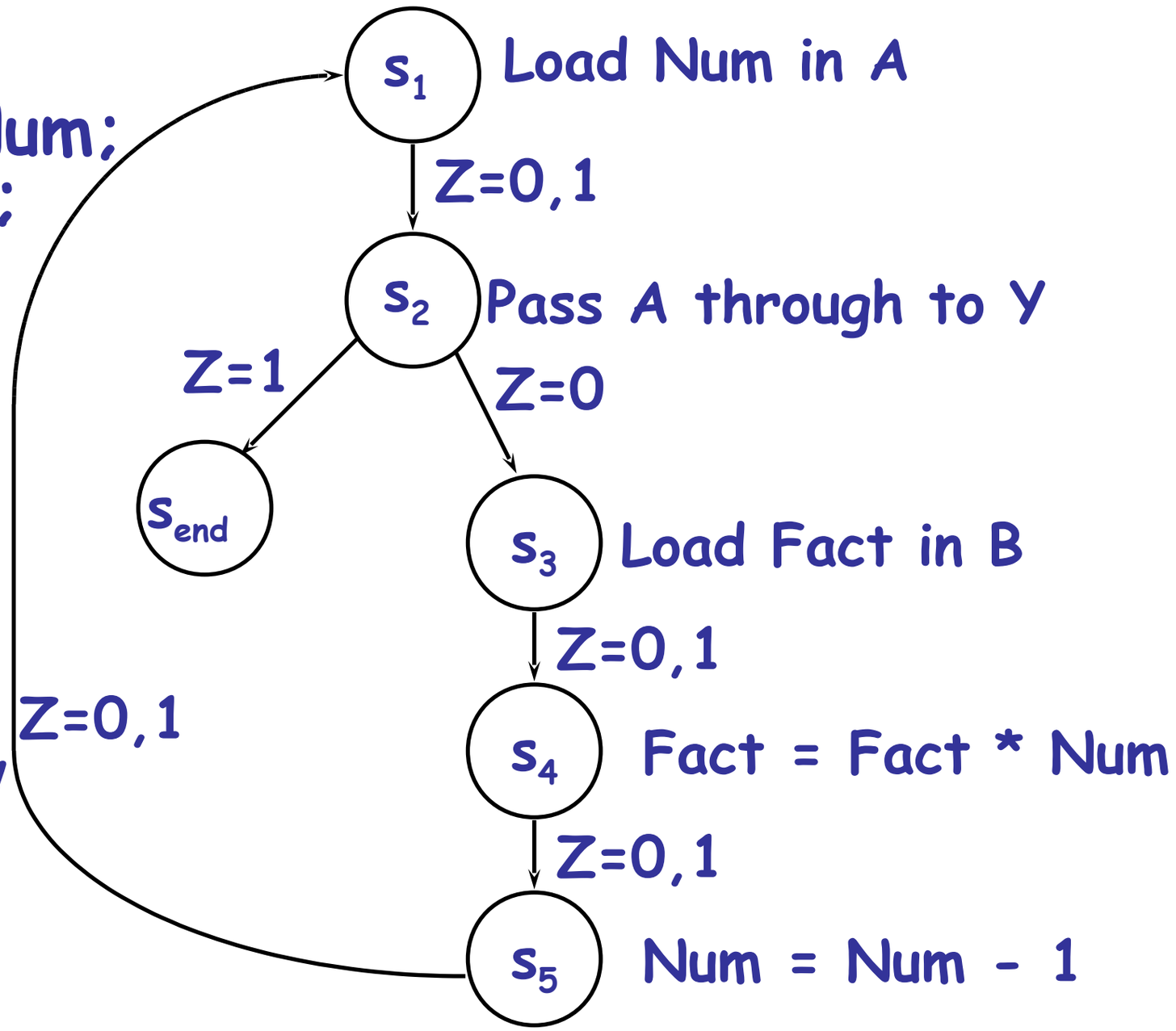
if (A < B) then Y = A + B
*else Y = A * B*

Computing Factorial

```
Fact = 1;
while ( Num > 0 ) {
      Fact = Fact * Num;
      Num = Num - 1;
}
// assume Num is
// positive at start so
// we just check for
// Num == 0 to break
```

Note that

Z=1 if Y==0 where Y is the *end result* of a state's *current* operation



Computing Factorial

```
Fact = 1;
while ( Num > 0 ) {
      Fact = Fact * Num;
      Num = Num - 1;
}
```

R_0 corresponds to variable Num

R_1 corresponds to variable Fact

assumed to be 1 initially

cur state	in put	next state	outputs					
CS	Z	NS	RF_{SEL}	RF_{LE}	ADR	A_{LE}	B_{LE}	F
s_1	*	s_2	-	0	000	1	0	–
s_2	1	s_{end}	-	0	–	0	0	A
s_2	0	s_3	-	0	–	0	0	A
s_3	*	s_4	-	0	001	0	1	–
s_4	*	s_5	0	1	001	0	0	A*B
s_5	*	s_1	0	1	000	0	0	A-1

next state based on Z

Load Num in A

if Num==0 exit

if Num > 0 continue

Load Fact in B

Fact = Fact * Num

Num = Num - 1;goto s_1

Computing Factorial

Note: This is a Moore Machine!

This is necessary in order to make F and Y stable.
(i.e., if F depends on Z, we get a cycle!)

cur state	in put	next state	outputs					
CS	Z	NS	RF_{SEL}	RF_{LE}	ADR	A_{LE}	B_{LE}	F
s_1	*	s_2	-	0	000	1	0	–
s_2	1	s_{end}	-	0	–	0	0	A
s_2	0	s_3	-	0	–	0	0	A
s_3	*	s_4	-	0	001	0	1	–
s_4	*	s_5	0	1	001	0	0	A*B
s_5	*	s_1	0	1	000	0	0	A-1

next state based on Z

Load Num in A

if Num==0 exit

if Num > 0 continue

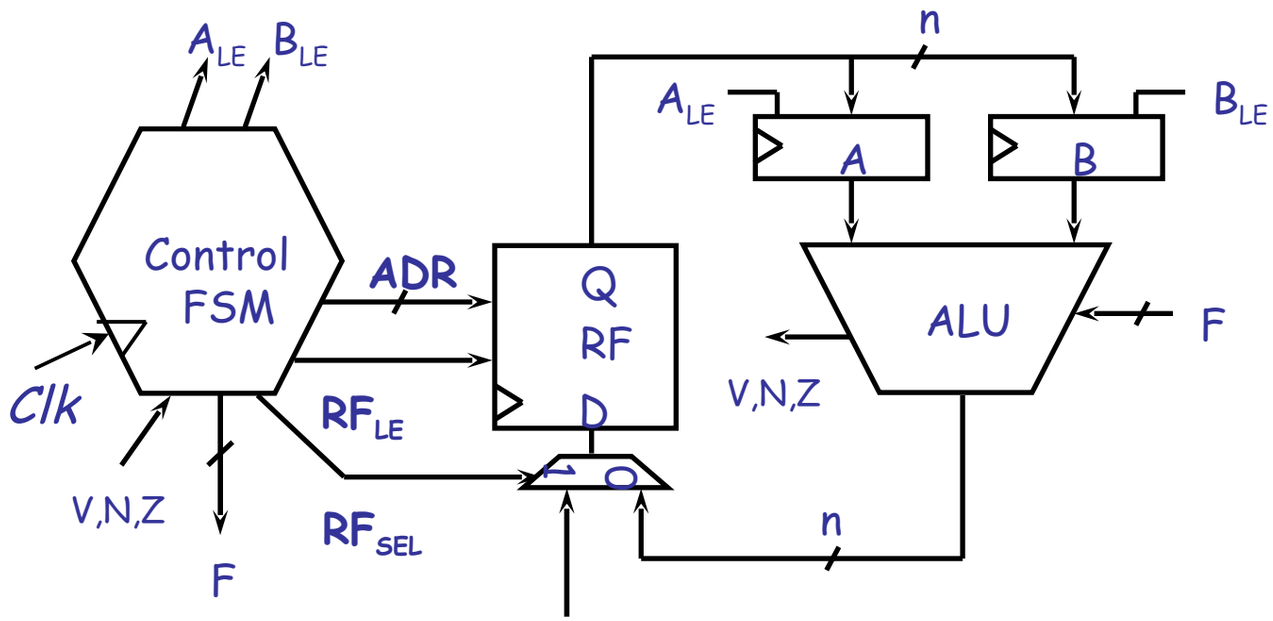
Load Fact in B

Fact = Fact * Num

Num = Num - 1;goto s_1

In general, use Moore machines

- **Remember, no cycles!**
- **Example: Put max(i,j) in R2, where i and j are stored in R0 and R1 initially.**



cur state | in put | next state | outputs

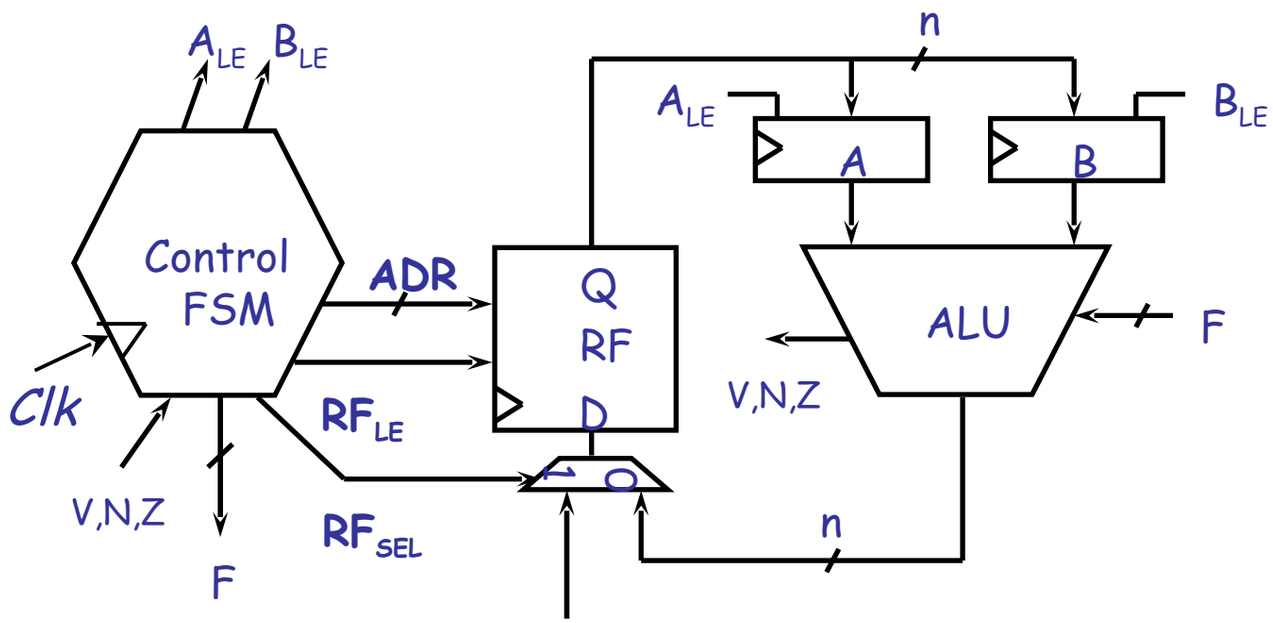
CS	N	NS	RF_{SEL}	RF_{LE}	ADR	A_{LE}	B_{LE}	F	
s_1	*	s_2	-	0	000	1	0	–	Load i into A
s_2	*	s_3	-	0	001	0	1	–	Load j into B
s_3	0	s_4							**if (A>=B) goto s4**
s_3	1	s_5							**if (A<B) goto s5**
s_4	*	s_{end}	0	1	010	0	0	A	Put A into R2; exit
s_5	*	s_{end}	0	1	010	0	0	B	Put B into R2; exit

next state based on N

F based on state

Sometimes, Mealy is OK

- **Only if it has no cycles!**
- **Less steps/states, but possibly longer t_{clk}**
- **<u>Example:</u> Put abs(i) in R0, where i is in R0 initially; assume OK to clobber/overwrite regs.**



cur state | in put | next state | outputs

CS	N	NS	RF_{SEL}	RF_{LE}	ADR	A_{LE}	B_{LE}	F
s_1	*	s_2	-	0	000	1	1	–
s_2	*	s_3	0	1	001	0	0	A-B
s_3	*	s_4	-	0	001	1	0	–
s_4	0	s_{end}						
s_4	1	s_{end}						

RF_{LE} based on N

Load i into A and B

Load zero into R1

Load zero into A

if (-i>=0) put -i in R0

if (-i<0) don't write

Possibly diff. output for same state?

You have to compute -i (0-i) regardless of N. Why?

Observe...

- Same datapath could perform many different tasks (e.g., **K = (C + D) * (C - D)**, **Factorial**)
  - Achieved by manipulating the control FSM.
- Do we want to write control FSMs for each task we want the datapath to perform?
  - Only if you like pain (in the form of 1's and 0's).
- Do we have a choice? Yes and no.
  - No: We still have to write a control FSM.
  - Yes: We can write it in a MUCH better format.
    - Then let something else translate it for us.

Control FSMs Are Tedious and Boring!

- Register-Transfer Language (RTL) or machine language is an alternative to filling in 1's and 0's in FSM.

Destination Address ⟵ Source Value (say "gets")

<*Address*> denotes value at *Address* (say "contents of")

Example:

$$R_2 \longleftarrow \langle R_0 \rangle + \langle R_1 \rangle$$

R_2 *gets contents of* R_0 **plus** *contents of* R_1

Which would you rather write?

$$K = (C + D) * (C - D)$$

	RF_{SEL}	RF_{LE}	ADR	A_{LE}	B_{LE}	F	
s_1	1	1	000	0	0	–	$R_0 \leftarrow C$
s_2	1	1	001	0	0	–	$R_1 \leftarrow D$
s_3	-	0	000	1	0	–	$A \leftarrow \langle R_0 \rangle$
s_4	-	0	001	0	1	–	$B \leftarrow \langle R_1 \rangle$
s_5	0	1	000	0	0	A+B	$R_0 \leftarrow \langle A \rangle + \langle B \rangle$
s_6	0	1	001	0	0	A-B	$R_1 \leftarrow \langle A \rangle - \langle B \rangle$
s_7	-	0	000	1	0	–	$A \leftarrow \langle R_0 \rangle$
s_8	-	0	001	0	1	–	$B \leftarrow \langle R_1 \rangle$
s_9	0	1	000	0	0	A*B	$R_0 \leftarrow \langle A \rangle * \langle B \rangle$

RTL or Machine Language Requirements

- Need abstract description of machine.
  - Set of registers: R_0, R_1, …
  - Set of operations: +, *, or …
  - Coded sequence of instructions:
    $R_1 \leftarrow \langle R_2 \rangle + \langle R_3 \rangle$
- Need precise understanding of how instructions change machine state.
  - RTL expressions that cannot be implemented in a single cycle will have to be broken down into expressions that can.
- Need to define instructions that affect control flow (conditionals).

More RTL Examples

Factorial(Num)

```
s1: A ← <R0>                    // Load Num in A
s2: if (<A>==0) goto s_end      // if Num==0 exit
        else goto s3            // else goto s3
s3: B ← <R1>                    // Load Fact in B
s4: R1 ← <A>*<B>                // Fact = Fact * Num
s5: R0 ← <A>-1; goto s1         // Num=Num-1; goto s_1
```

Max(i,j)

```
s1: A ← <R0>                    // Load i into A
s2: B ← <R1>                    // Load j into B
s3: if ((<A>-<B>)>=0 goto s4    // if (i >=j) goto s4;
    else goto s5                // else goto s5;
s4: R2 ← <A>; goto s_end        // R2 gets i; exit
s5: R2 ← <B>; goto s_end        // R2 gets j; exit
```

Note: Higher-level comments are still useful.