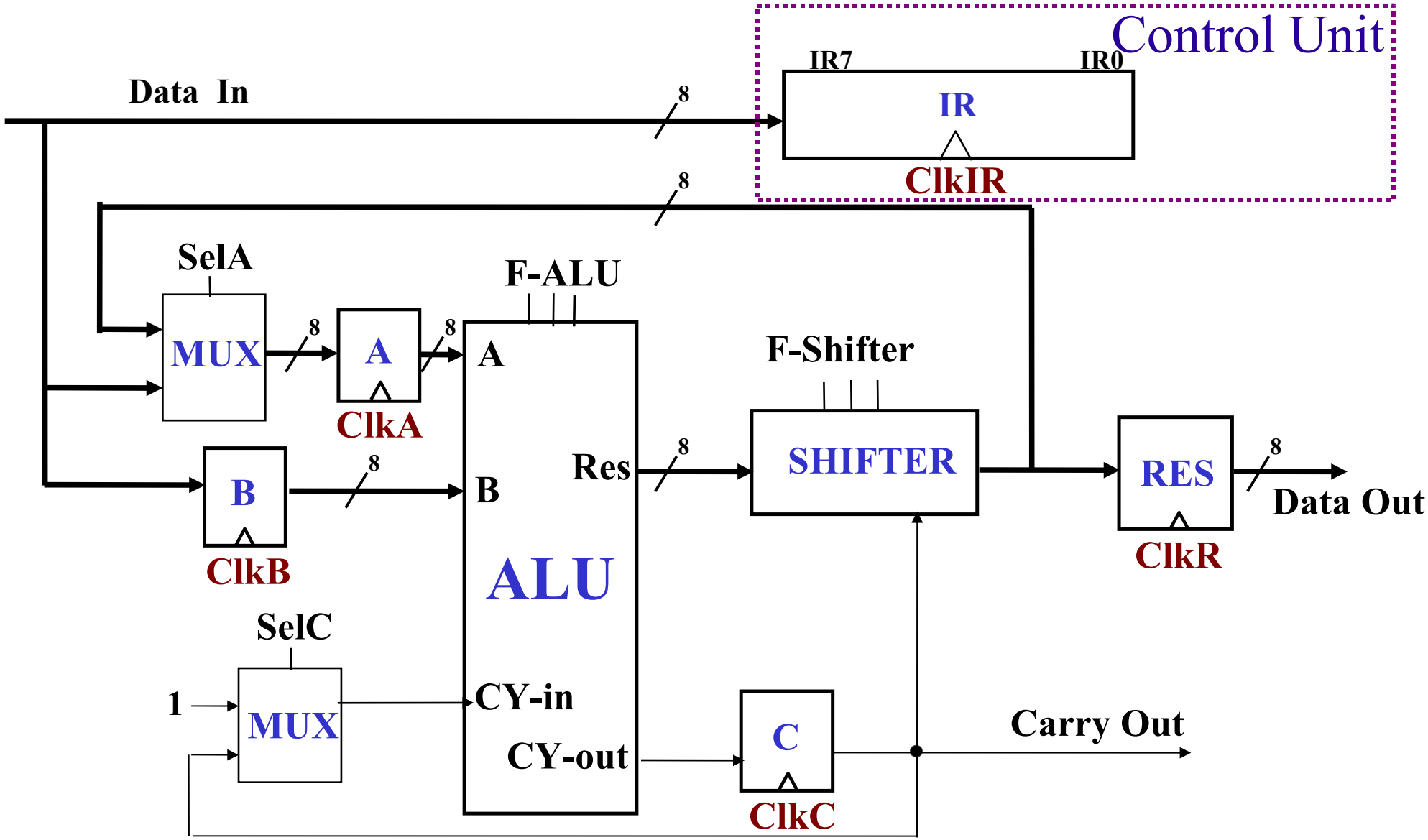
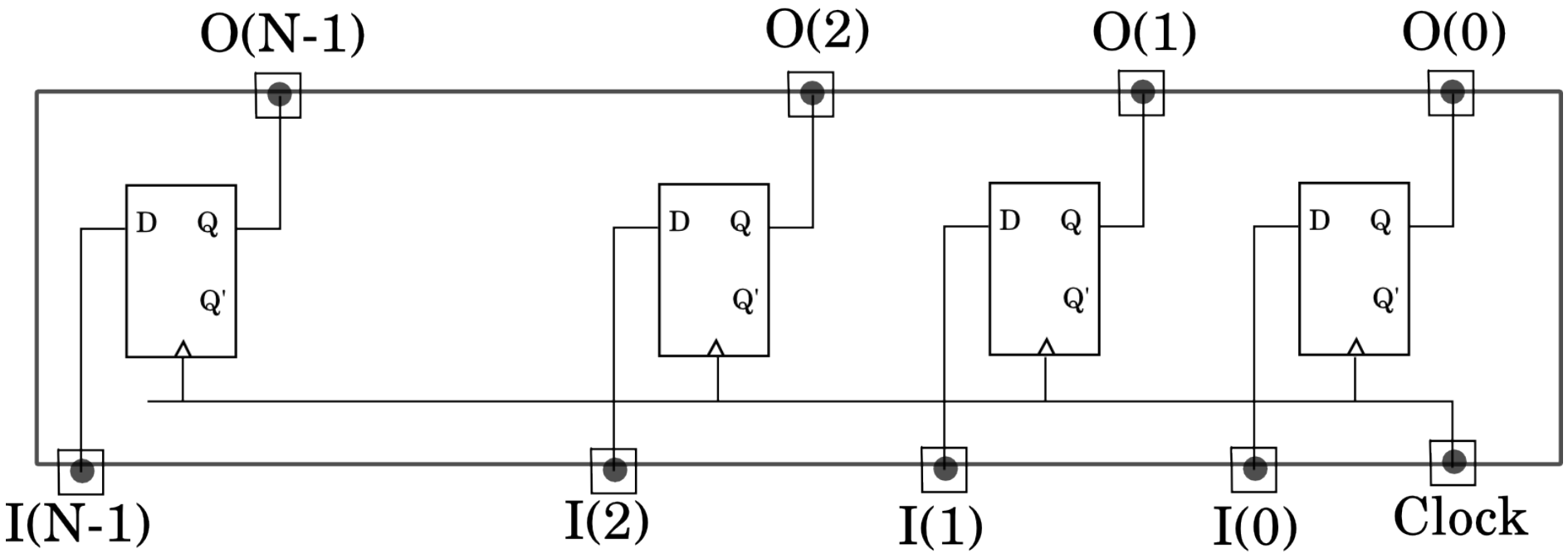
Lecture 15:

A Manual Processor (part 2)

The Basic Processor Architecture from last lecture was:

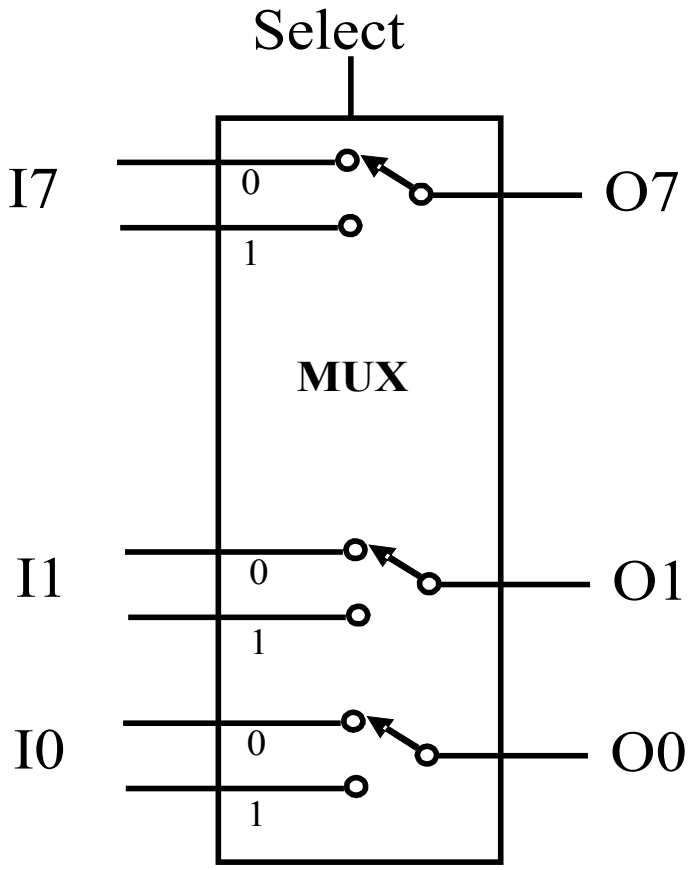


Registers A B Res and IR are all of the form:



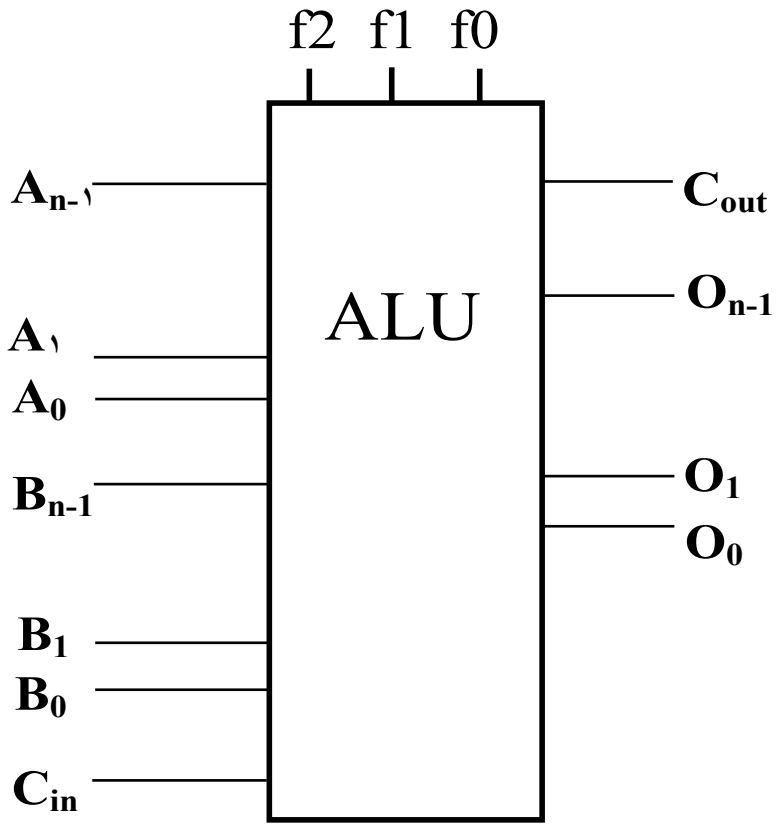
Register C is a single D-Q flip-flop

Multiplexer to select A has the form:



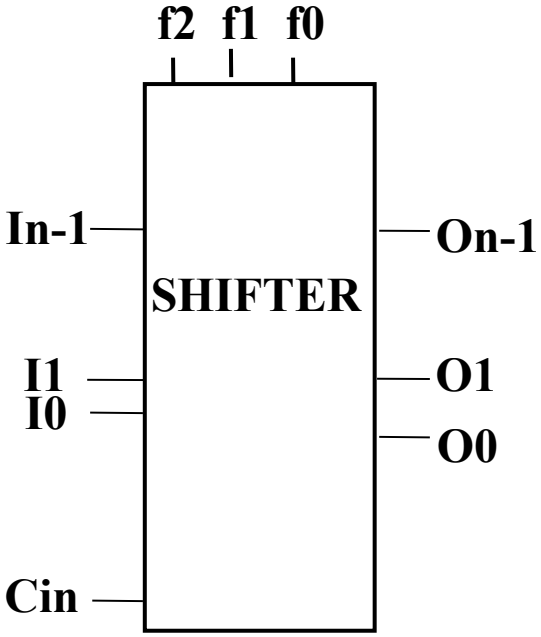
The multiplexer to select C is a single bit version of this

The Arithmetic and Logic Unit is a combinatorial logic circuit



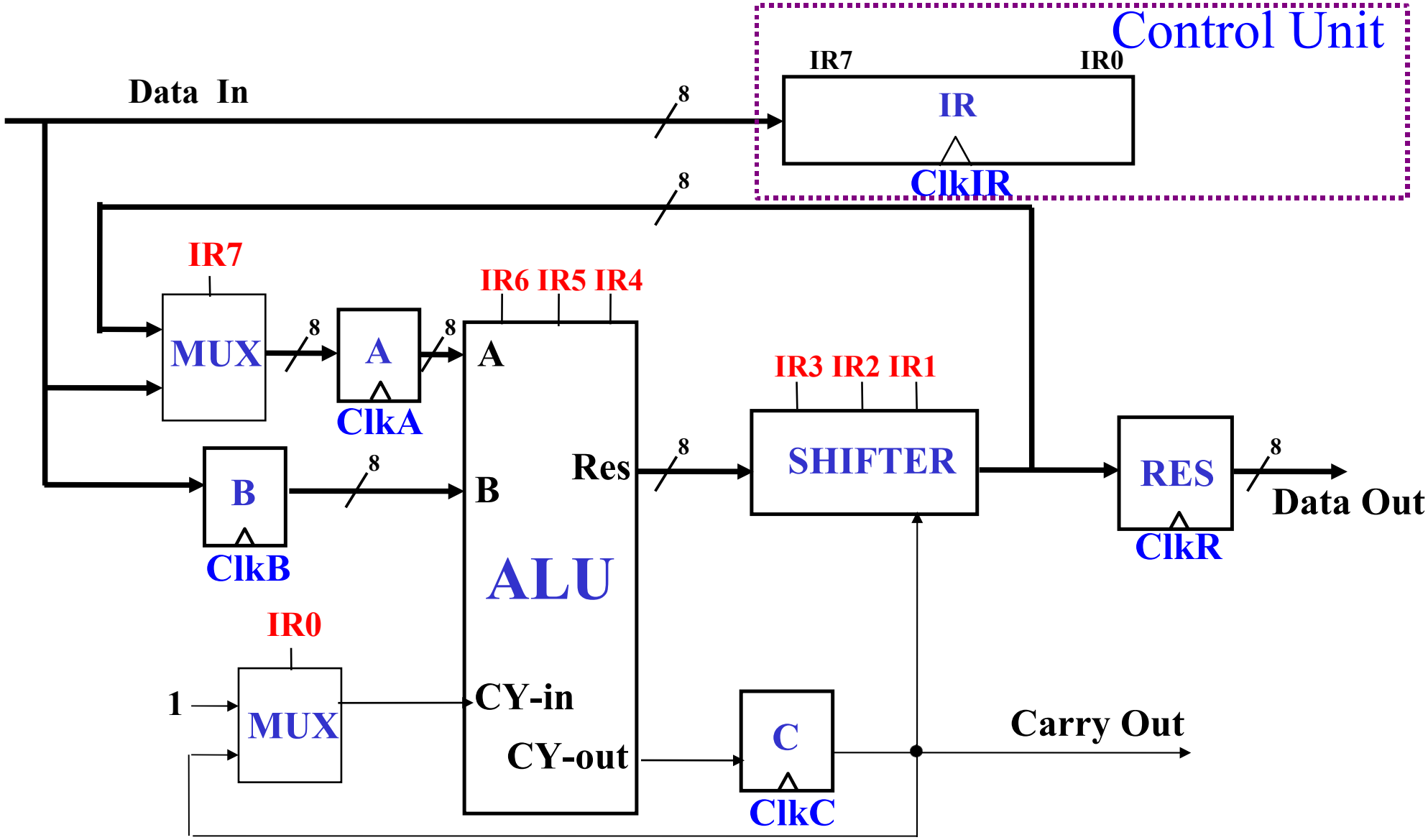
Selection Bits/Results							
000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
000 **zero**	**B - A** **minus**	**A - B** **minus**	**A + B** **plus**	**A ⊕ B** **xor**	**A or B** **or**	**A • B** **and**	**1111** **-1**

The Shifter is another combinatorial logic circuit implemented by multiplexers:

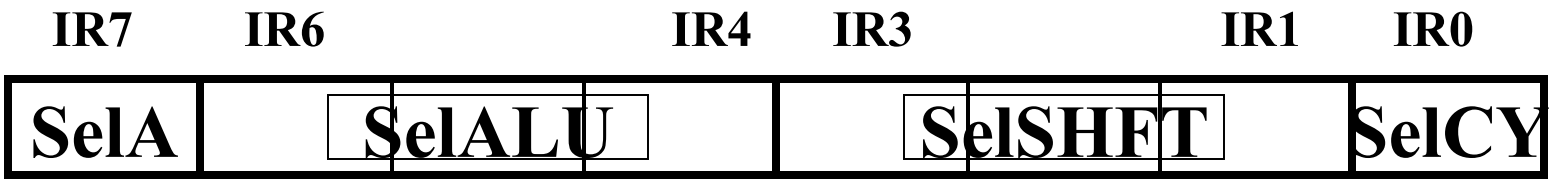


F[2]	F[1]	F[0]	Shift	Carry	Function
0	0	0		Input[0]	unchanged
0	0	1	left	0	arithmetic left shift
0	1	0	right	Input[0]	rotate right
0	1	1	right	0	logical right shift
1	0	0		Input[7]	unchanged
1	0	1	left	Cin	left shift with carry
1	1	0	right	Input[7]	arithmetic right shift
1	1	1	right	Cin	shift right with carry

Now for the control unit:



Instruction Format



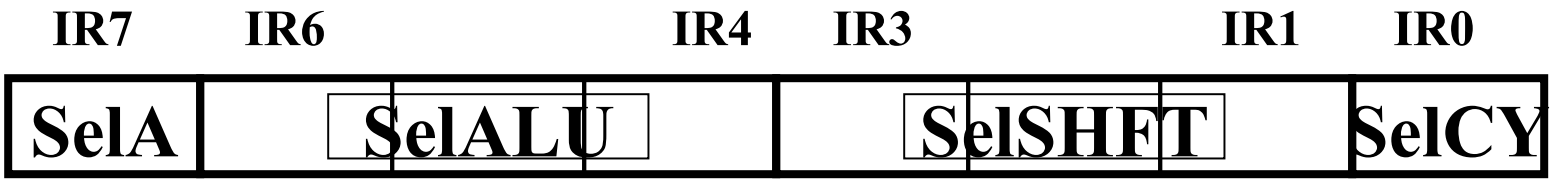
Instruction Register IR

The multiplexers are controlled by bits 0 and 7 of the instruction

SELA 0-1 Selects the input to the A register
0 selects shifter output, 1 selects external Data Input.

SELCY 0-1 Selects the input to the CARRY-in of the ALU:
0 selects logic 1, 1 selects C-out

Instruction Format (Continued)



Instruction Register
IR

SELALU bits 4, 5 and 6 selects the ALU output:

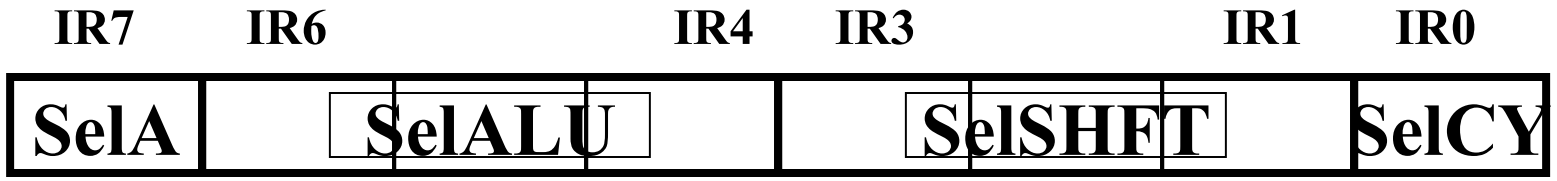
000 ZERO 001 B-mi-A

010 A-mi-B 011 A-pl-B

100 A-xor-B 101 A-or-B

110 A-and-B 111 MINUS ONE

Instruction Format (Continued)



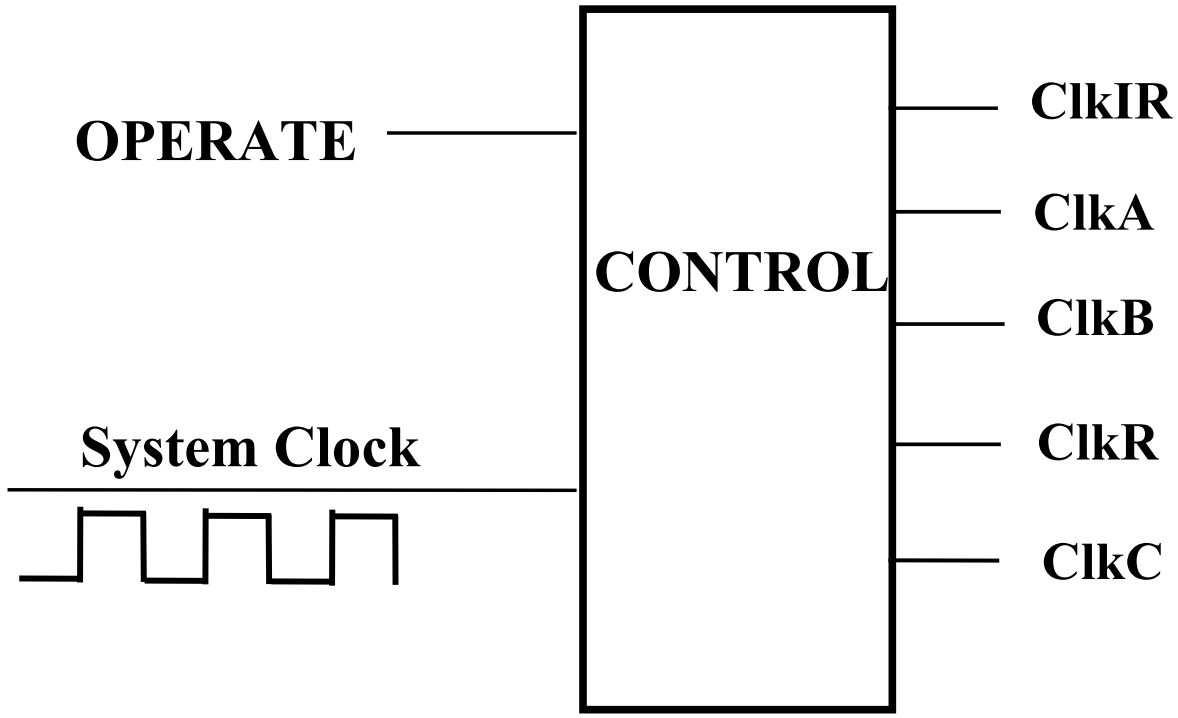
Instruction Register IR

SELSHIFT Selects the shift function:

000: Unchanged	001: Arithmetic Left Shift
010: Rotate Right	011: Logical right shift
100: Unchanged	101: Left shift with carry
110: Arithmetic right shift	111: Right shift with carry

Designing the control unit

At each clock pulse the controller provides one or more register with a clock pulse taking the processor through a fixed cycle of states

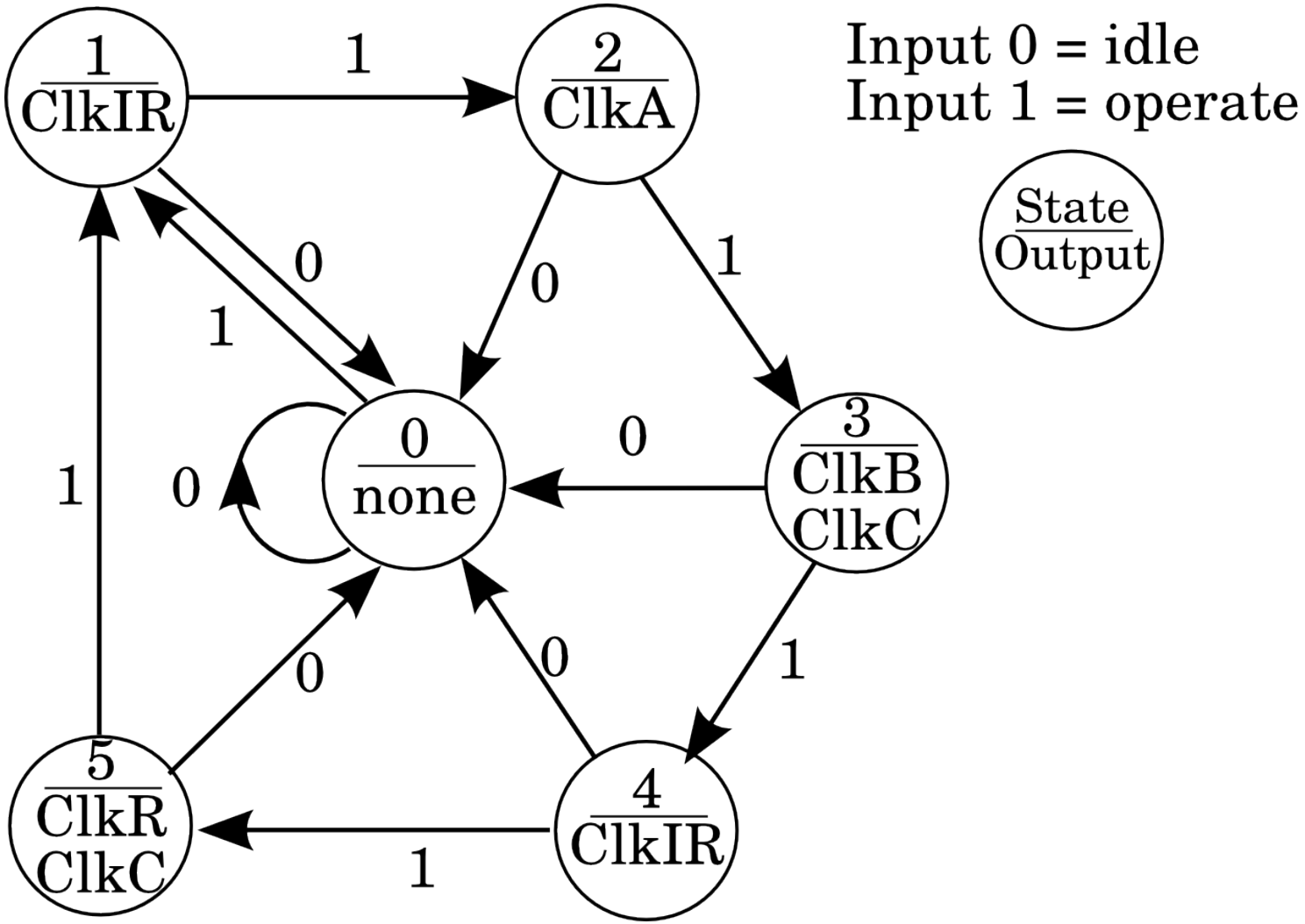


Execution Cycle of the Manual Processor

1. **Load the "Data In" lines into the IR register.**
2. **Load the A register**
3. **Load the B and the C registers**
4. **Load the "Data In" lines into the IR register**
5. **Load the RES and the C registers**

.... Go to step 1 of the next instruction ....

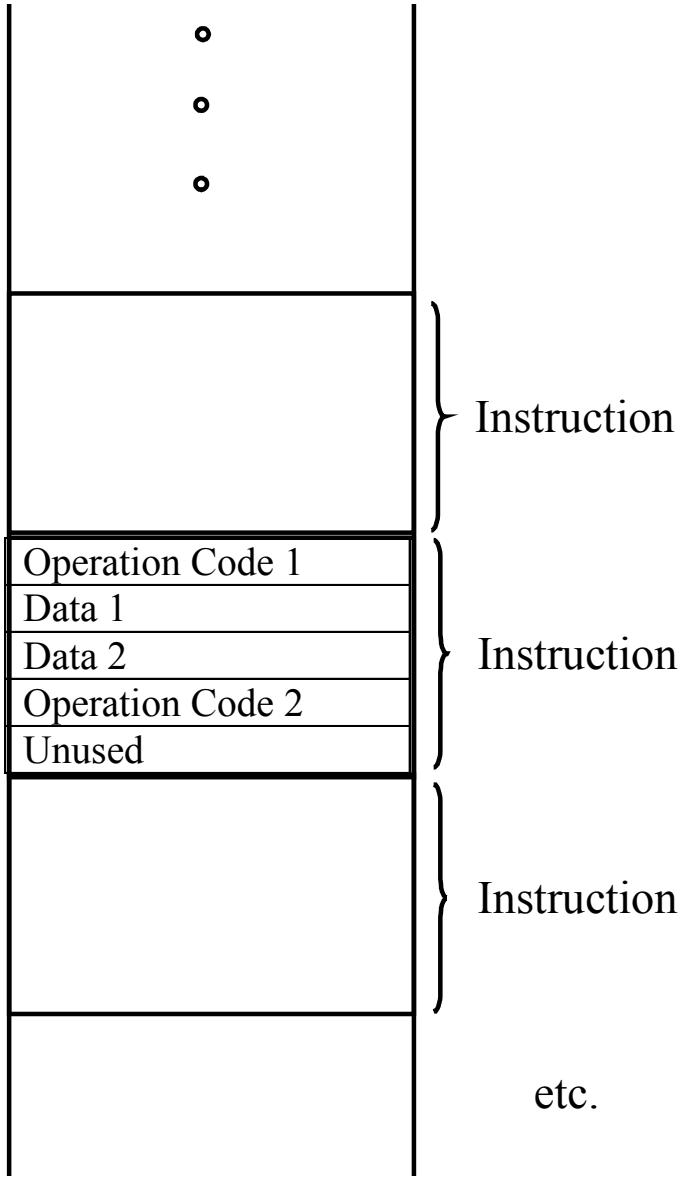
The State Transition Diagram



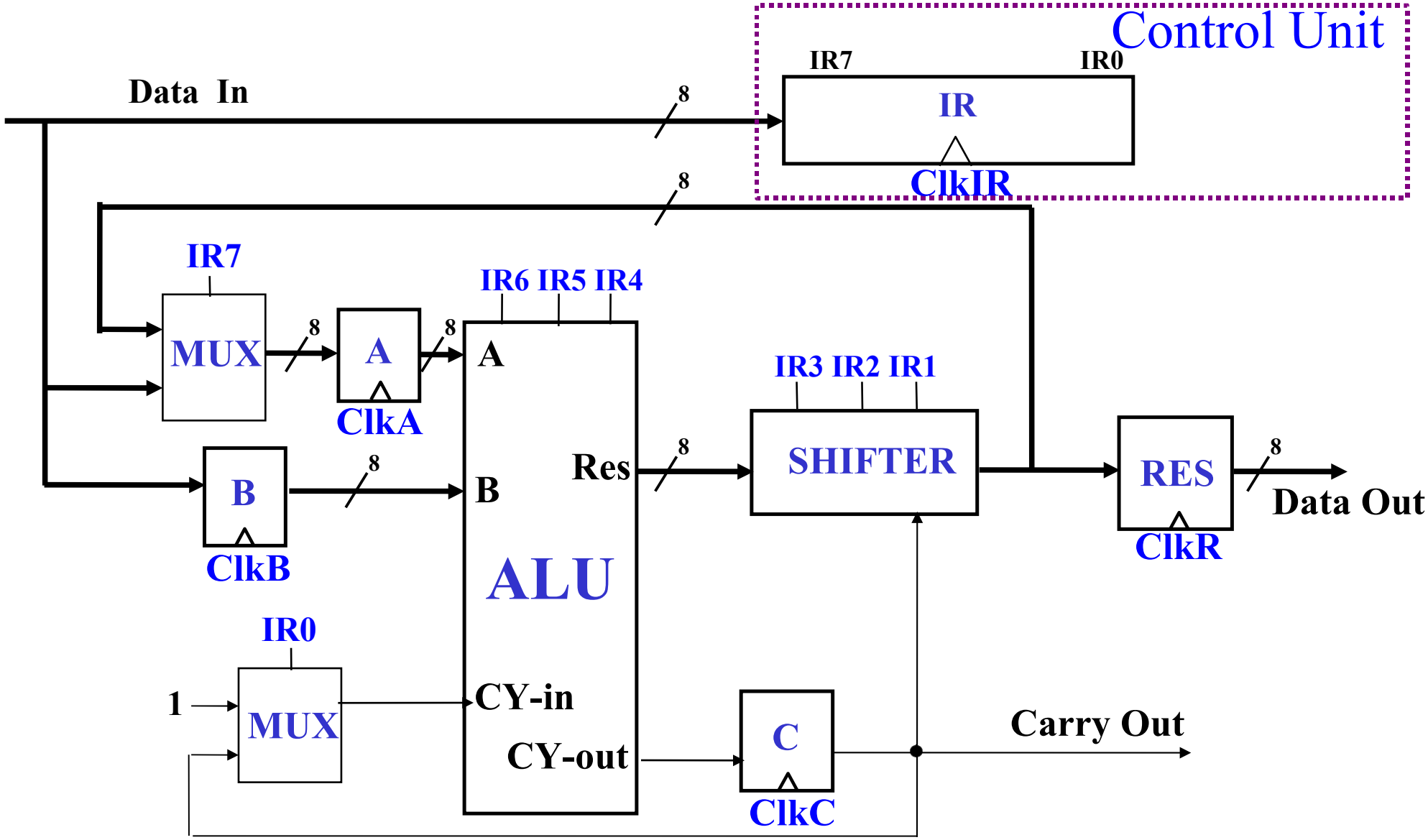
The Program

The execution cycle implicitly defines the format of the program.

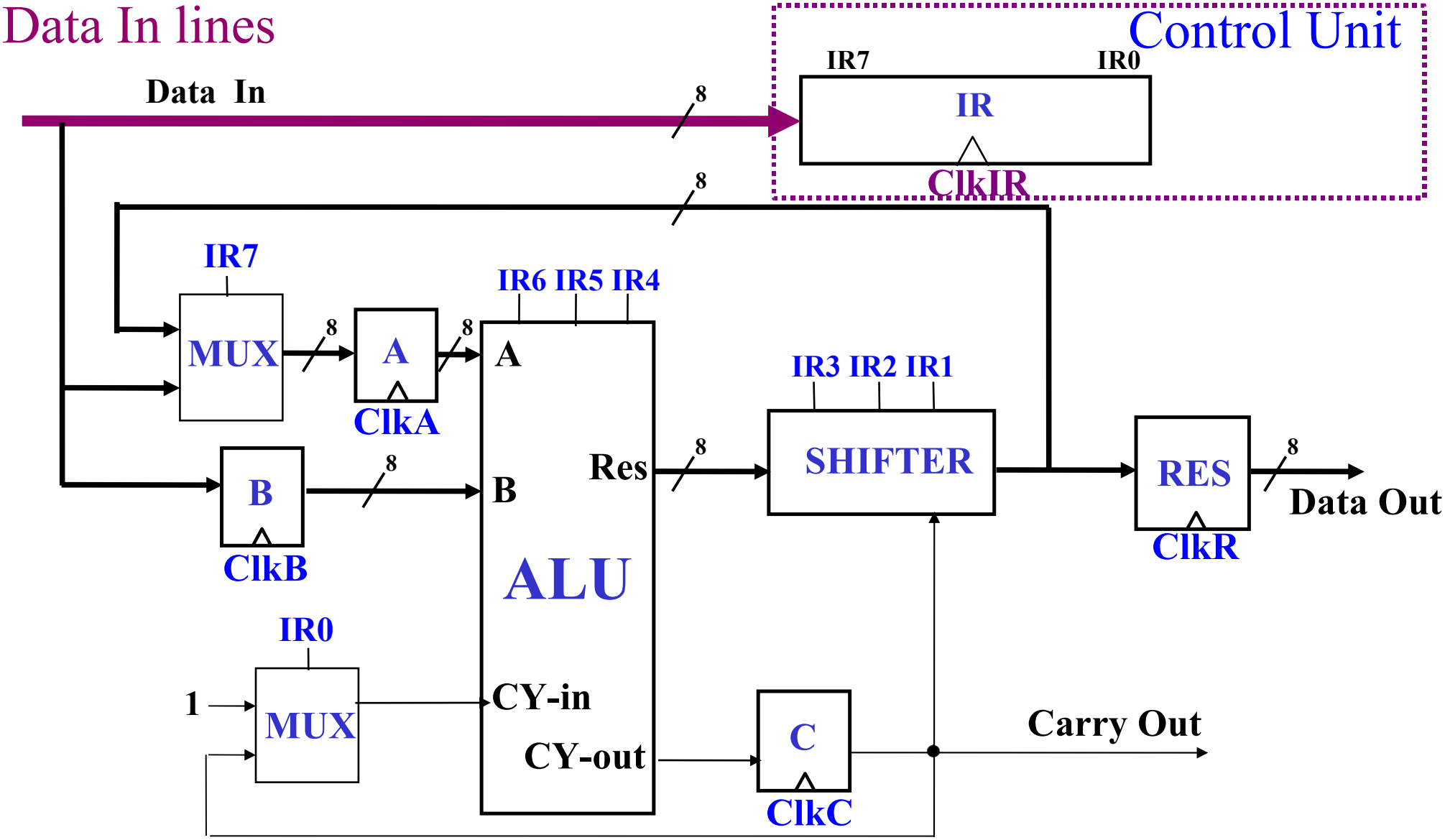
At each step a new item arrives on the data in lines synchronised with the system clock.



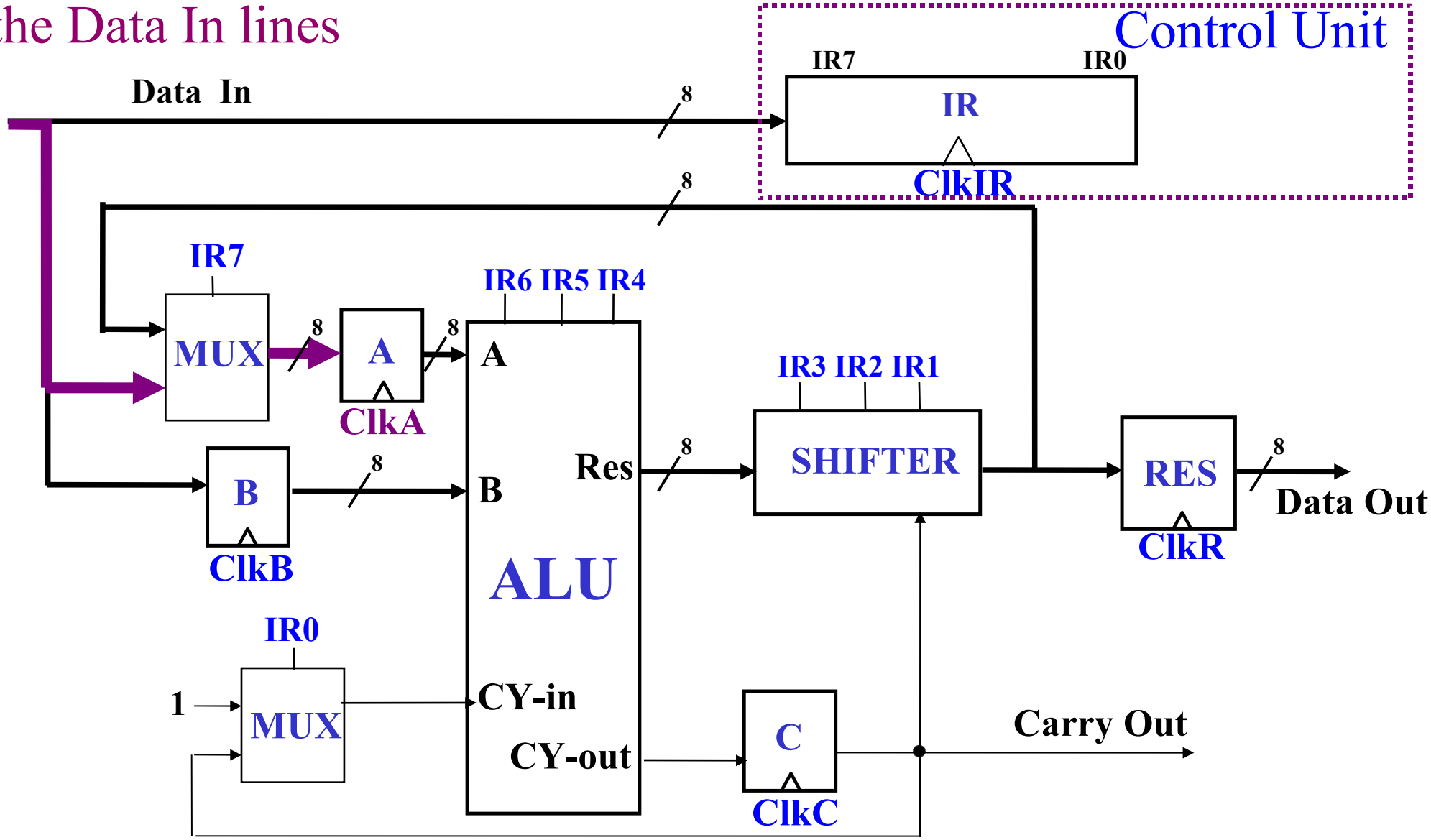
The processing cycle



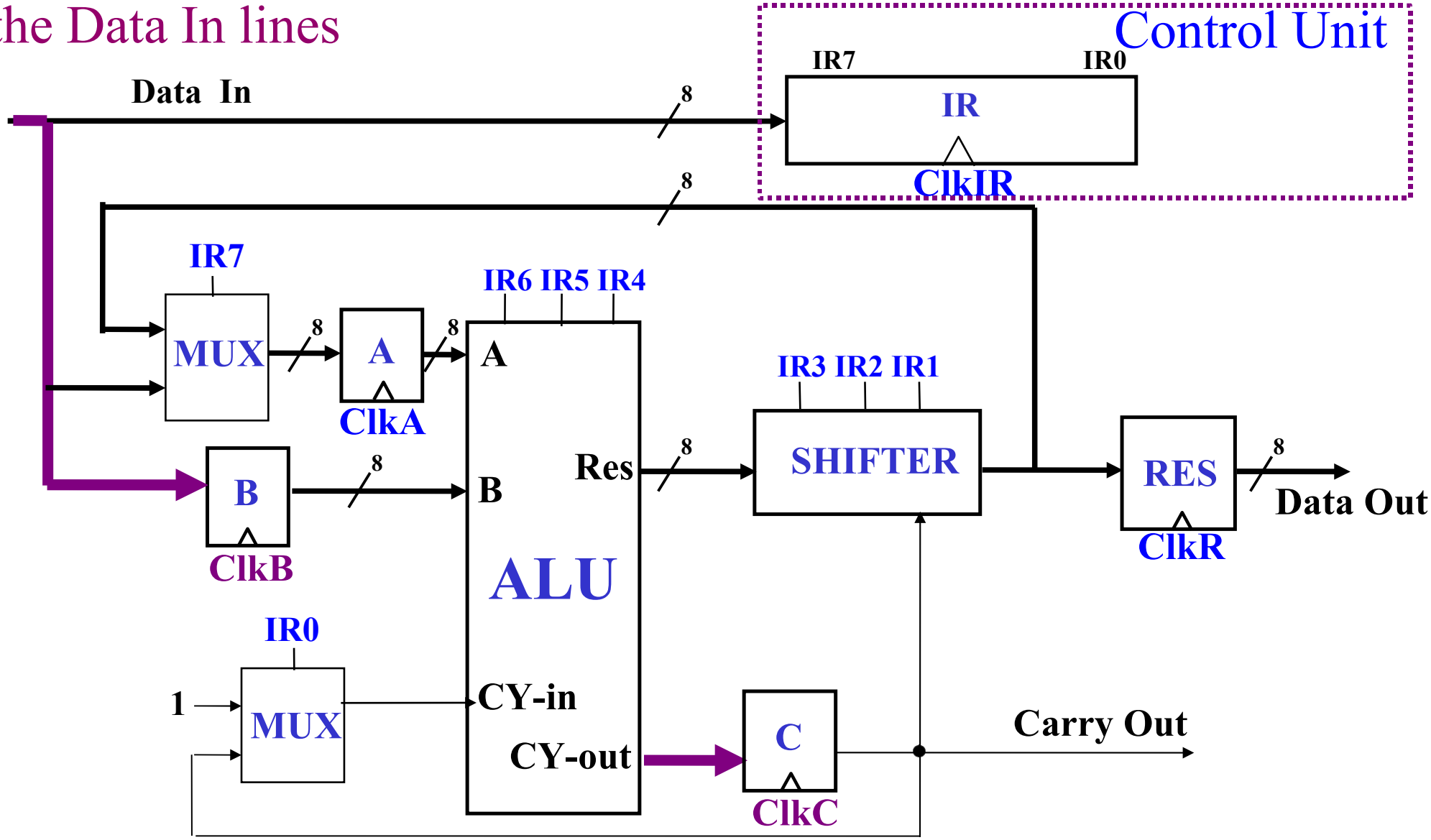
Step 1: IR receives a clock pulse. It is loaded from the Data In lines



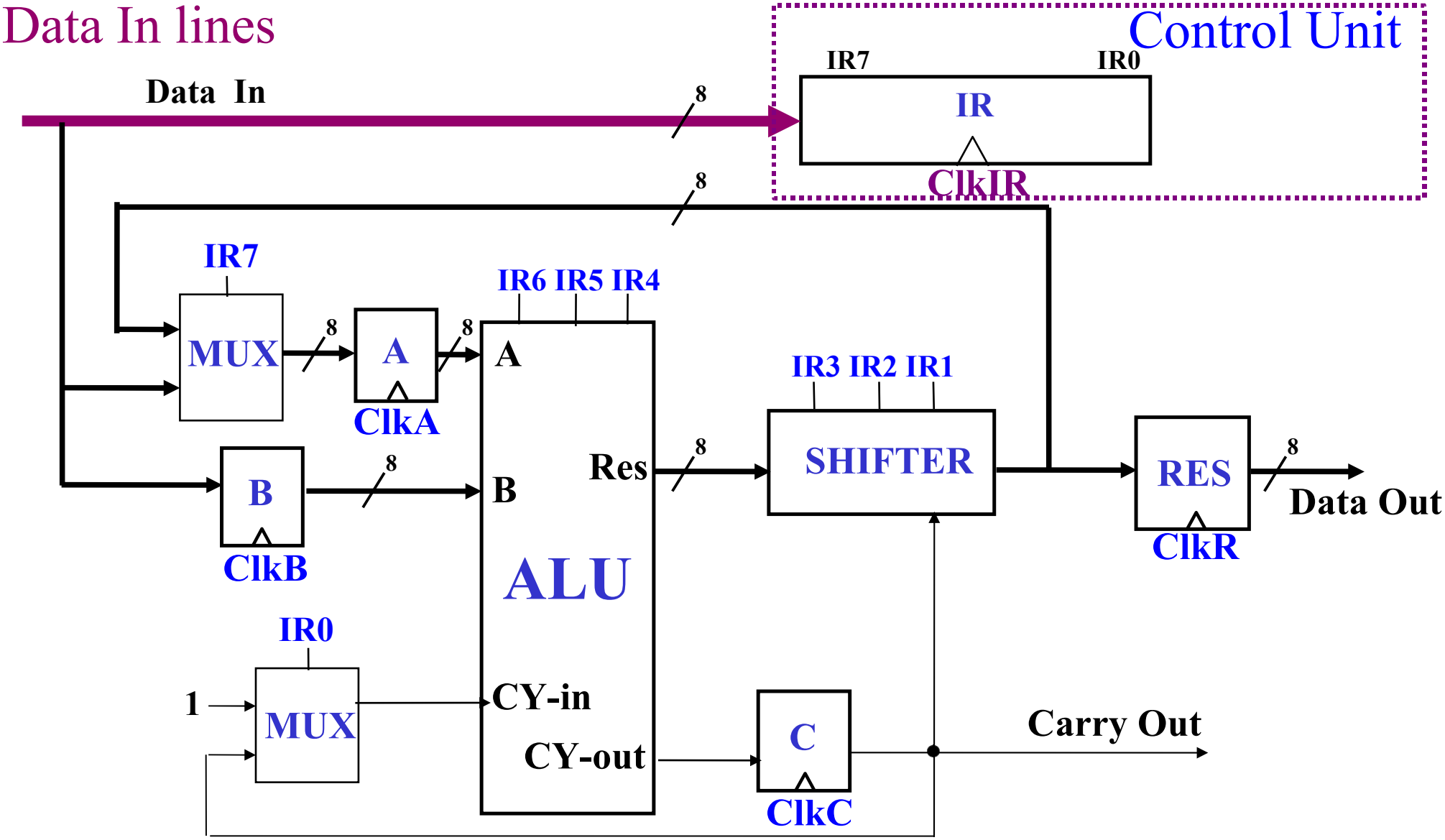
Step 2: A receives a clock pulse. It might be loaded from the Data In lines



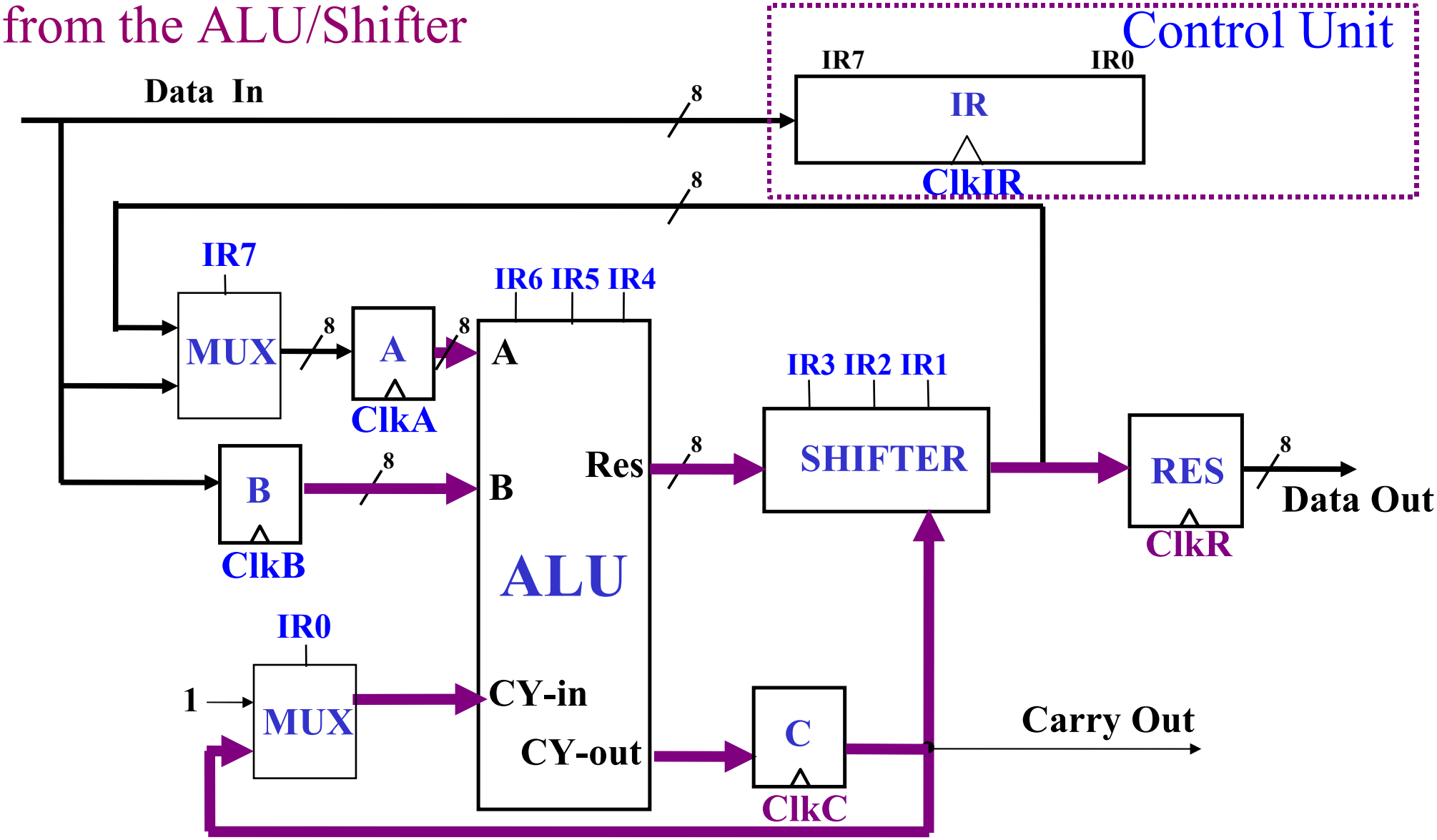
Step 3: B and C receive a clock pulse. B is loaded from the Data In lines



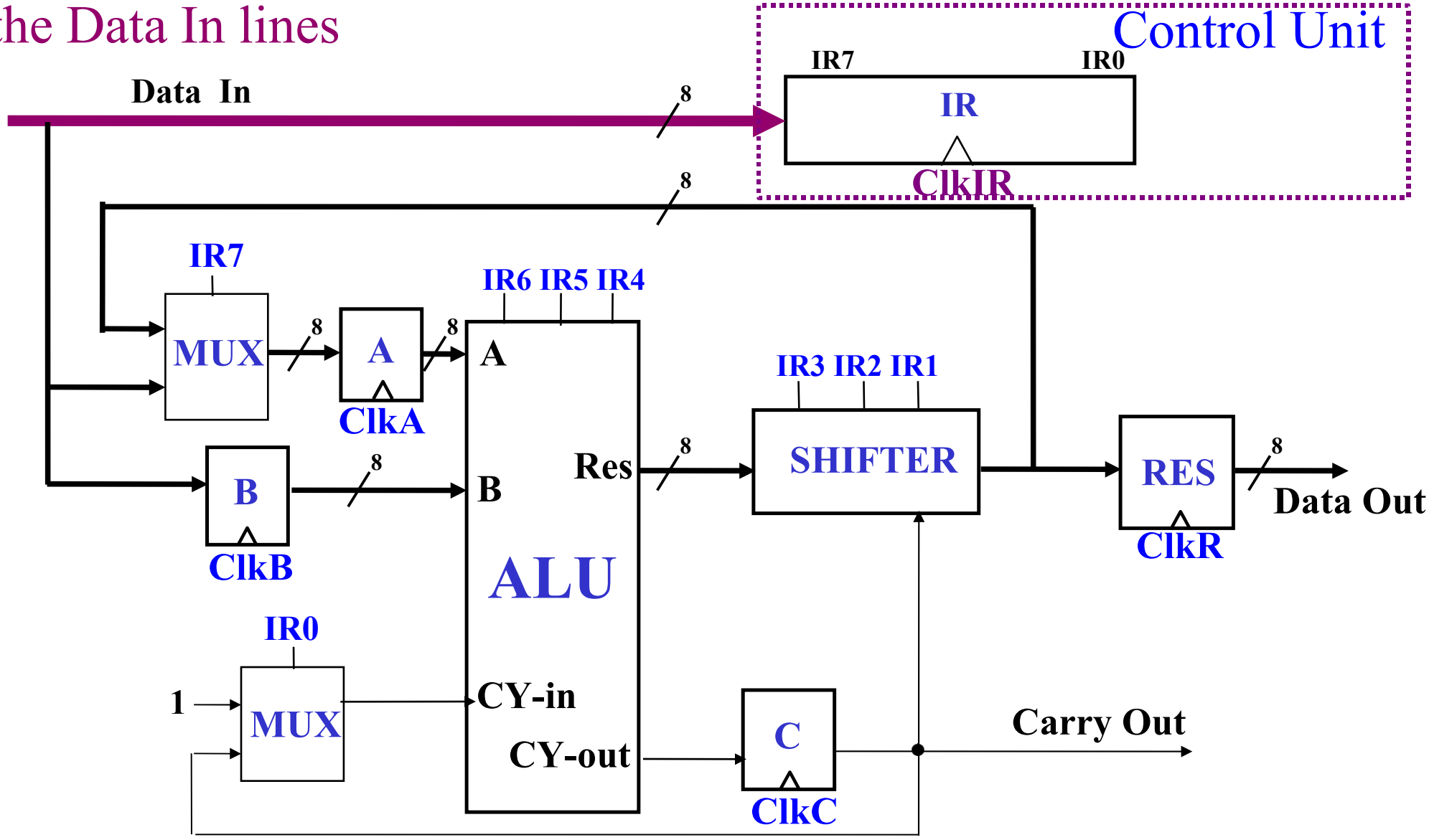
Step 4: IR receives a clock pulse. It is loaded from the Data In lines



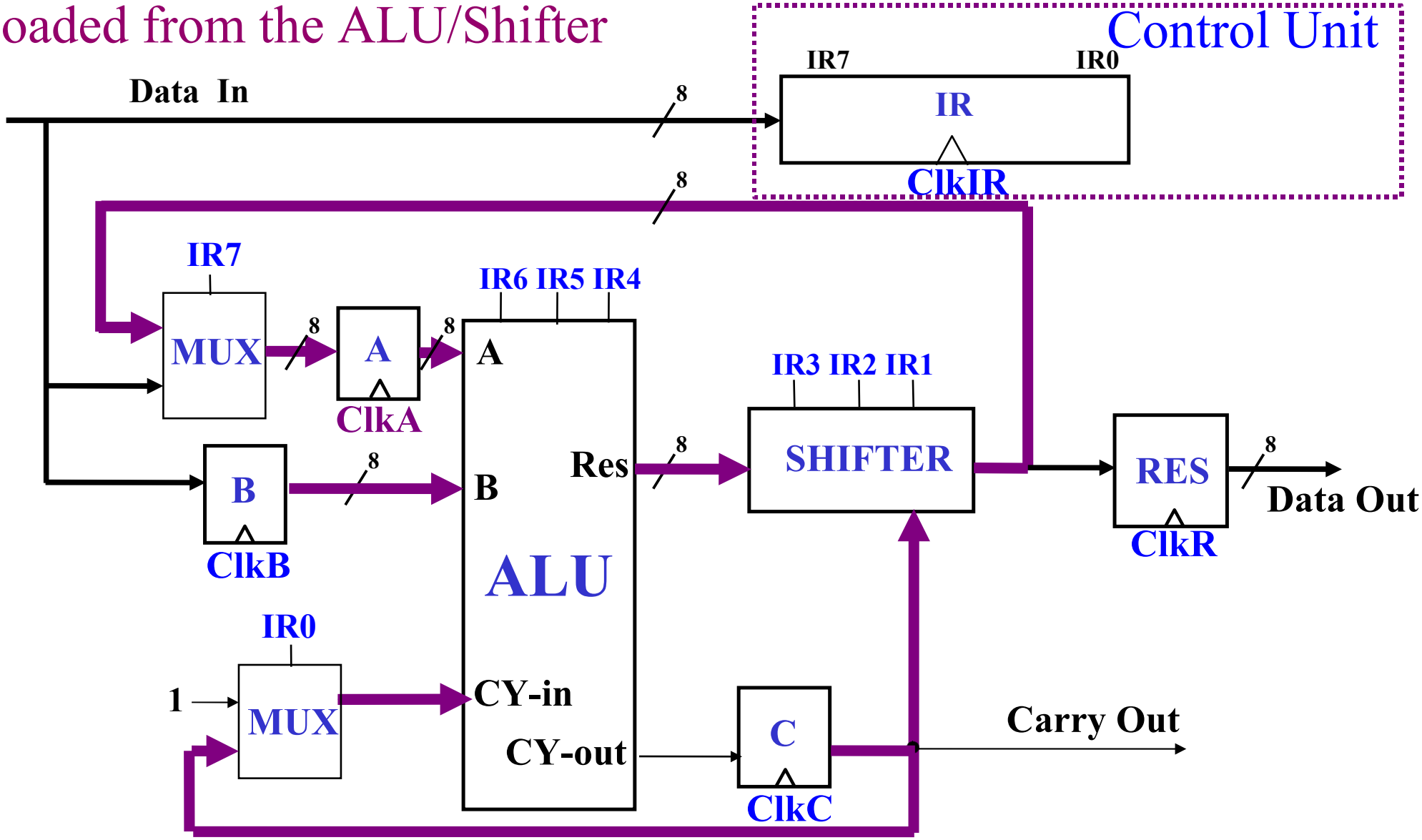
Step 5: Res and C receive a clock pulse. They are loaded from the ALU/Shifter



Step 1 (again): IR receives a clock pulse. It is loaded from the Data In lines



Step 2 (again): A receives a clock pulse. It might be loaded from the ALU/Shifter



The cycle continues

The cycle continues in the same way while the processor is being operated, taking data and instructions from the Data In lines and putting the results on the Data Out lines.

The design of the control unit which implements this cycle follows our (now very familiar) methodology.

Control Signals (output logic)

We can choose a state assignment that makes the output logic simple

Flip-Flop Outputs	State	Required Clock Output
000	0	none
001	1	ClkIR
100	2	ClkA
010	3	ClkB, ClkC
101	4	ClkIR
110	5	ClkC, ClkRES

Q1 Q0

Q2	00	01	11	10
0	0	0	X	0
1	1	0	X	0

ClkA = Q2 Q1' Q0'

Q1 Q0

Q2	00	01	11	10
0	0	0	X	1
1	0	0	X	0

ClkB = Q2'Q1

Q1 Q0

Q2	00	01	11	10
0	0	0	X	1
1	0	0	X	1

ClkC= Q1

Q1 Q0

Q2	00	01	11	10
0	0	1	X	0
1	0	1	X	0

ClkIR= Q0

Q1 Q0

Q2	00	01	11	10
0	0	0	X	0
1	0	0	X	1

ClkR= Q2Q1

The State Transition Table

Operate	Current State	Flip-Flops	Next State	D_2	D_1	D_0
0	0	000	0	0	0	0
0	1	001	0	0	0	0
0	2	100	0	0	0	0
0	3	010	0	0	0	0
0	4	101	0	0	0	0
0	5	110	0	0	0	0
0	6	011	?	×	×	×
0	7	111	?	×	×	×
1	0	000	1	0	0	1
1	1	001	2	1	0	0
1	2	100	3	0	1	0
1	3	010	4	1	0	1
1	4	101	5	1	1	0
1	5	110	1	0	0	1
1	6	011	?	×	×	×
1	7	111	?	×	×	×

Karnaugh Maps of the Controller

I Q2 \ Q1 Q0	00	01	11	10
00	0	0	X	0
01	0	0	X	0
11	0	1	X	0
10	0	1	X	1

D2 = I (Q0 + Q2'Q1)

I Q2 \ Q1 Q0	00	01	11	10
00	0	0	X	0
01	0	0	X	0
11	1	1	X	0
10	0	0	X	0

D1 = I Q2 Q1'

I Q2 \ Q1 Q0	00	01	11	10
00	0	0	X	0
01	0	0	X	0
11	0	0	X	1
10	1	0	X	1

D0 = I(Q1 + Q2'Q0')

Checking the Unused States

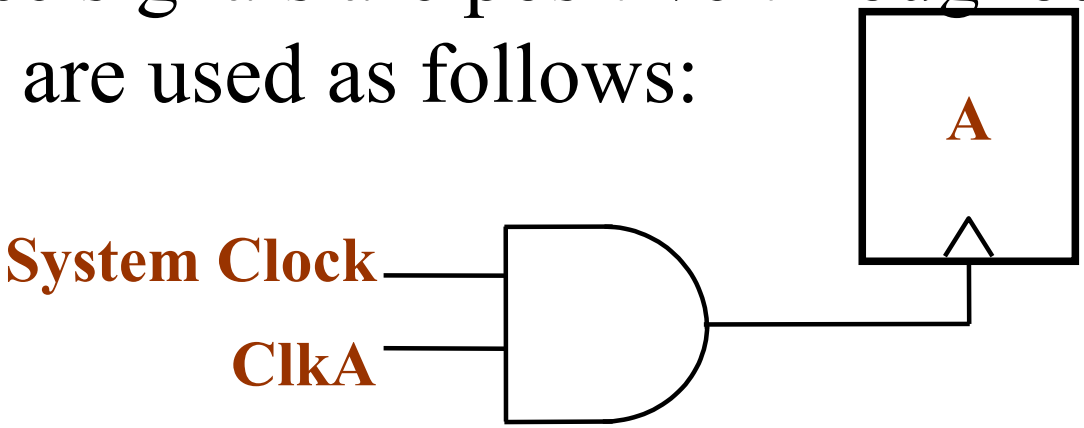
Operate	Current State	Flip-Flops	Next State	D_2	D_1	D_0
0	6	011	0	0	0	0
0	7	111	0	0	0	0
1	6	011	4	1	0	1
1	7	111	4	1	0	1

Thus, if the **OPERATE/IDLE** signal is at logical **0** the system drops into the **IDLE** state immediately and the processor is ready to start working properly.

Gating the Registers

The controller generates outputs for ClkA, ClkB, ClkC, ClkIR and ClkRes.

These signals are positive throughout the state, they are used as follows:



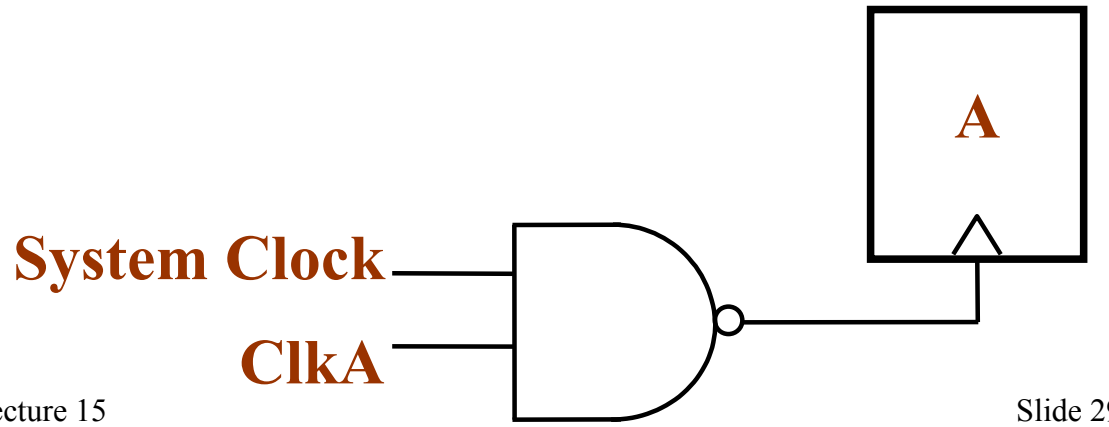
Everything happens on the falling edge of the clock.

Gating the Registers

A safer design has the controller changing on the negative edge and the registers being loaded on the positive edge of the clock.

This ensures that, when the input is taken from the RAM it can be properly established before a register is loaded.

A NAND gate does the trick



Program Execution: Compute (A+B)/2

0. Set OPERATE/IDLE signal to 0 & apply two clocks.

The processor is in the IDLE state

1. Set OPERATE/IDLE to 1 Set input data to 10000000 & apply one clock pulse.

Input is loaded in the IR register. ALU is set to 00000000, shifter = unchanged.

2. Set number A on the Data In lines & apply one clock pulse

Because IR7=1 the Data In lines are loaded onto register A.

Program Execution: (A+B)/2

3. **Set number B on the Data In lines and apply one clock pulse**

B is loaded from the Data In lines and C is loaded from the ALU carry which is 0

4. **Set the Data In lines to to 00110111 and apply one clock pulse.**

The IR register is loaded from the Data In line. The ALU is set to A plus B and the shifter to logical shift right.

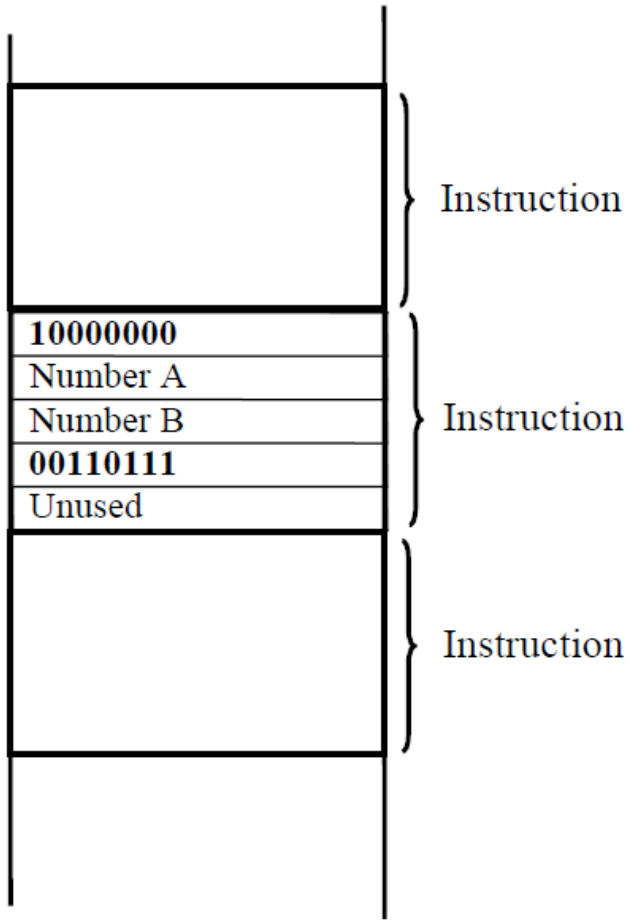
5. **Apply one clock pulse.**

The result (A+B)/2 is clocked into the RES register, the C bit indicates overflow

The (A+B)/2 instruction in a program

The mean instruction could be used as part of a longer program.

The mean of the two arguments goes back to memory via the data out lines



At last:

We have succeeded in executing one program instruction!

We'll take a look at getting the instructions in and the results out next time.