



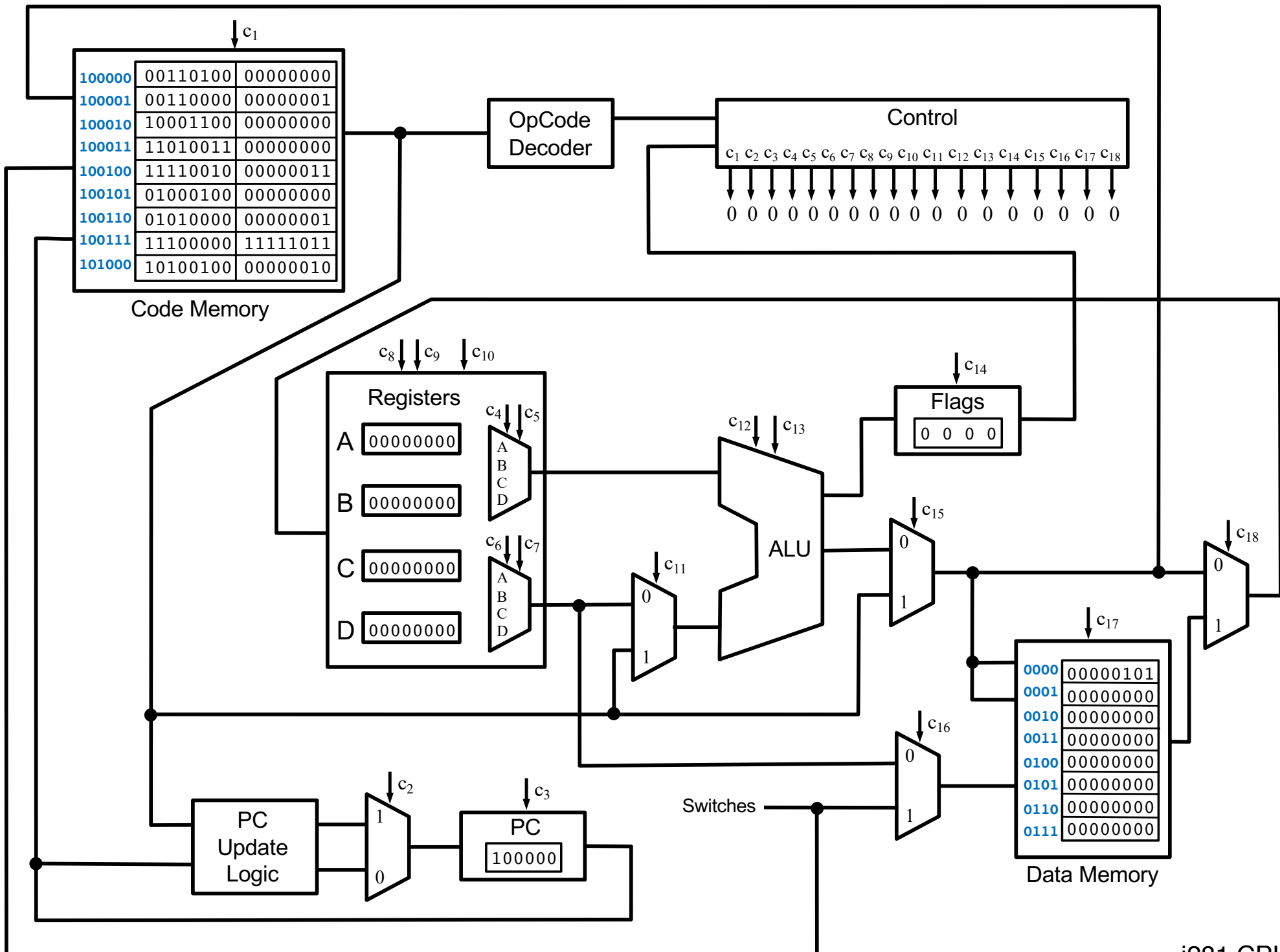
# **CprE 281: Digital Logic**

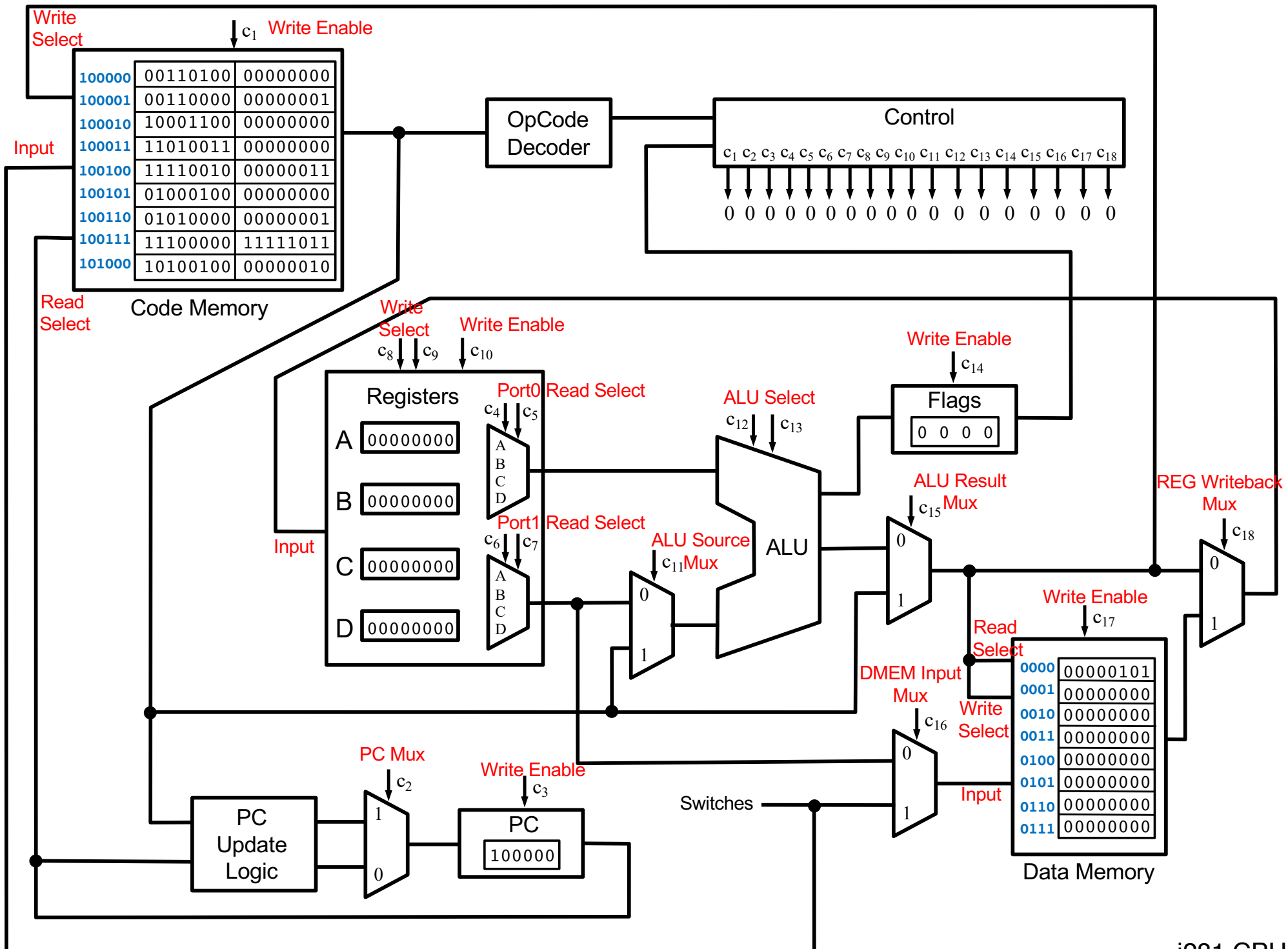
**Instructor: Alexander Stoytchev**

**<http://www.ece.iastate.edu/~alexs/classes/>**

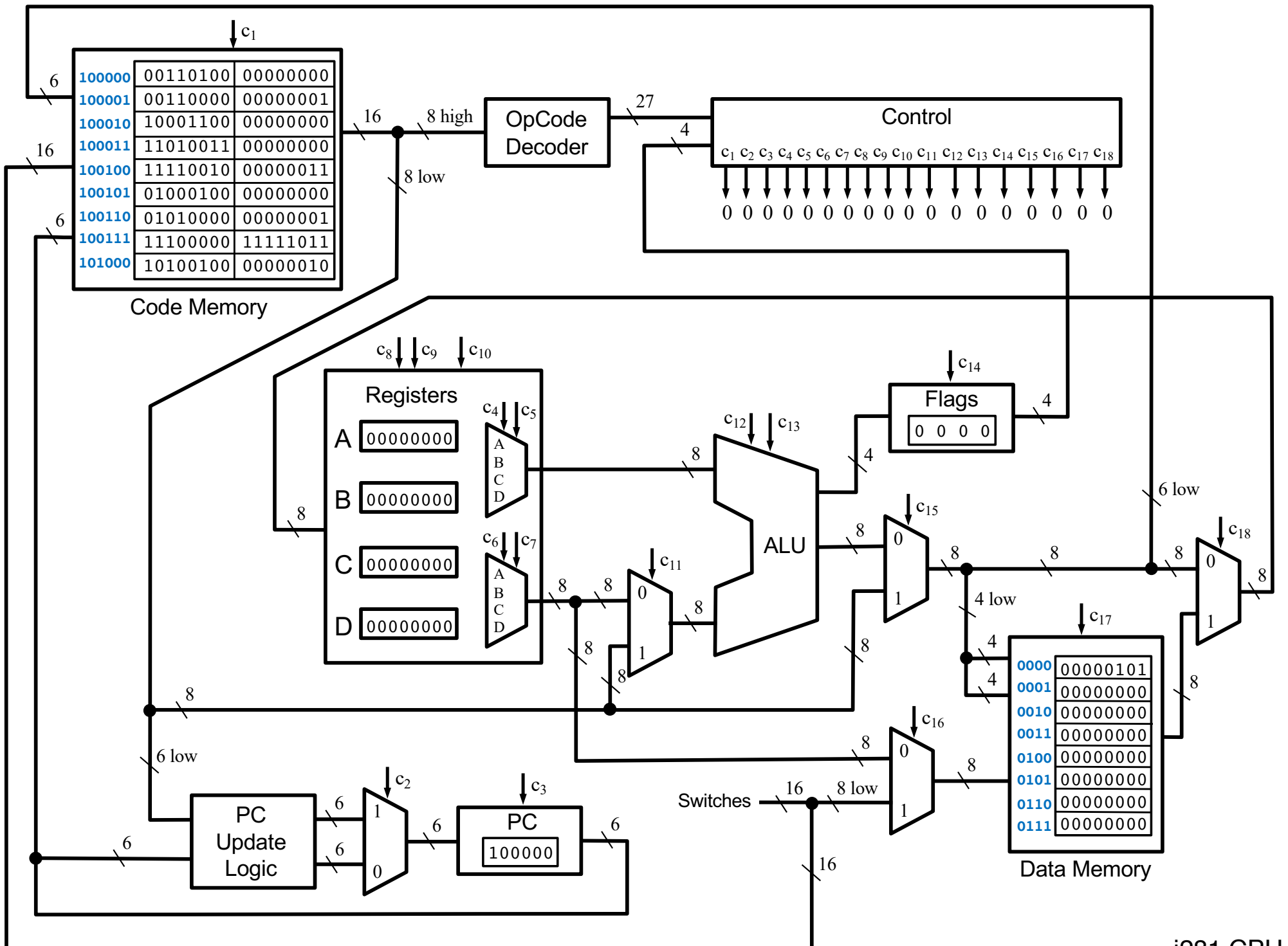
# **Arithmetic Logic Unit & Program Counter**

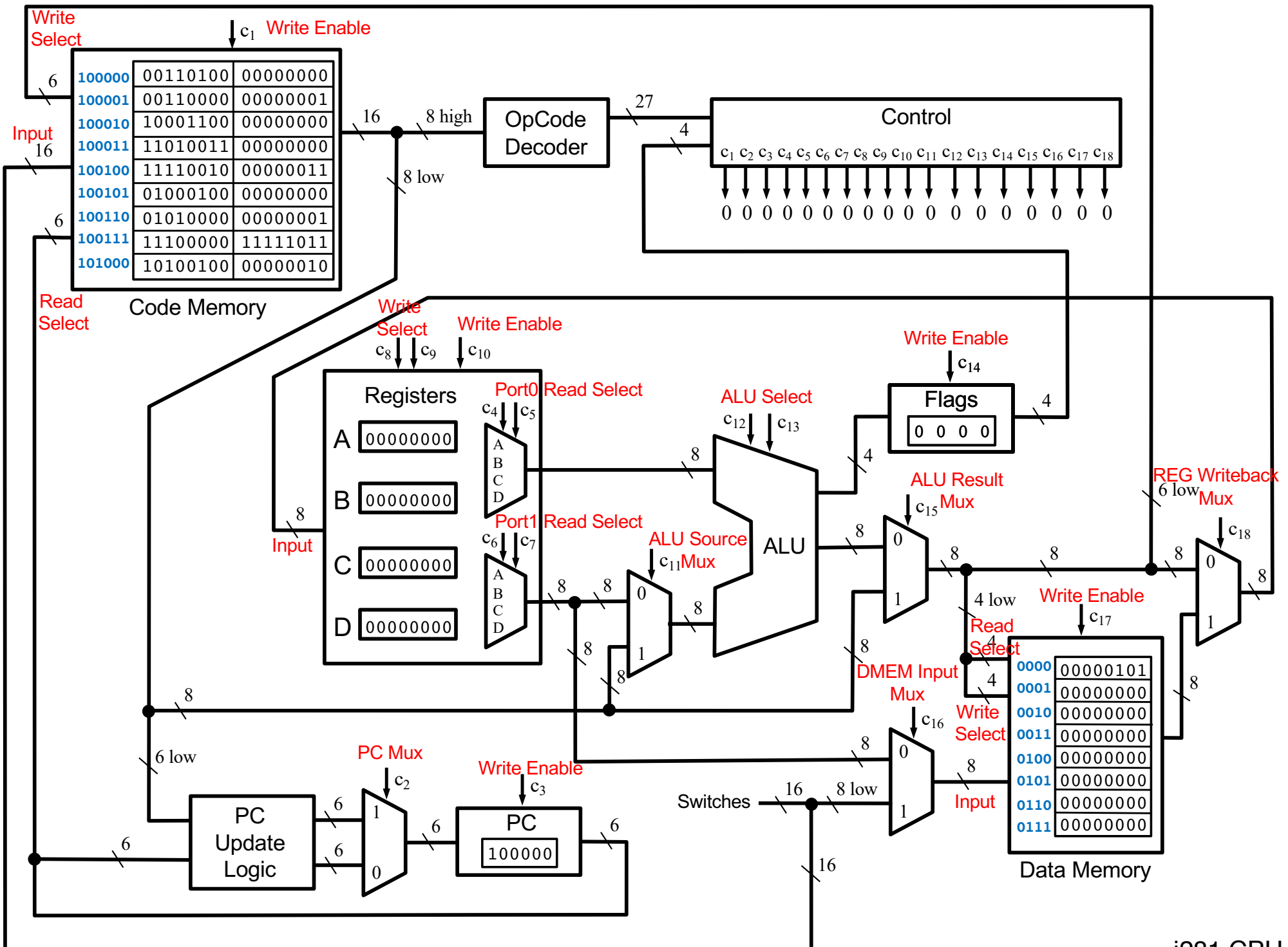
*CprE 281: Digital Logic  
Iowa State University, Ames, IA  
Copyright © Alexander Stoytchev*



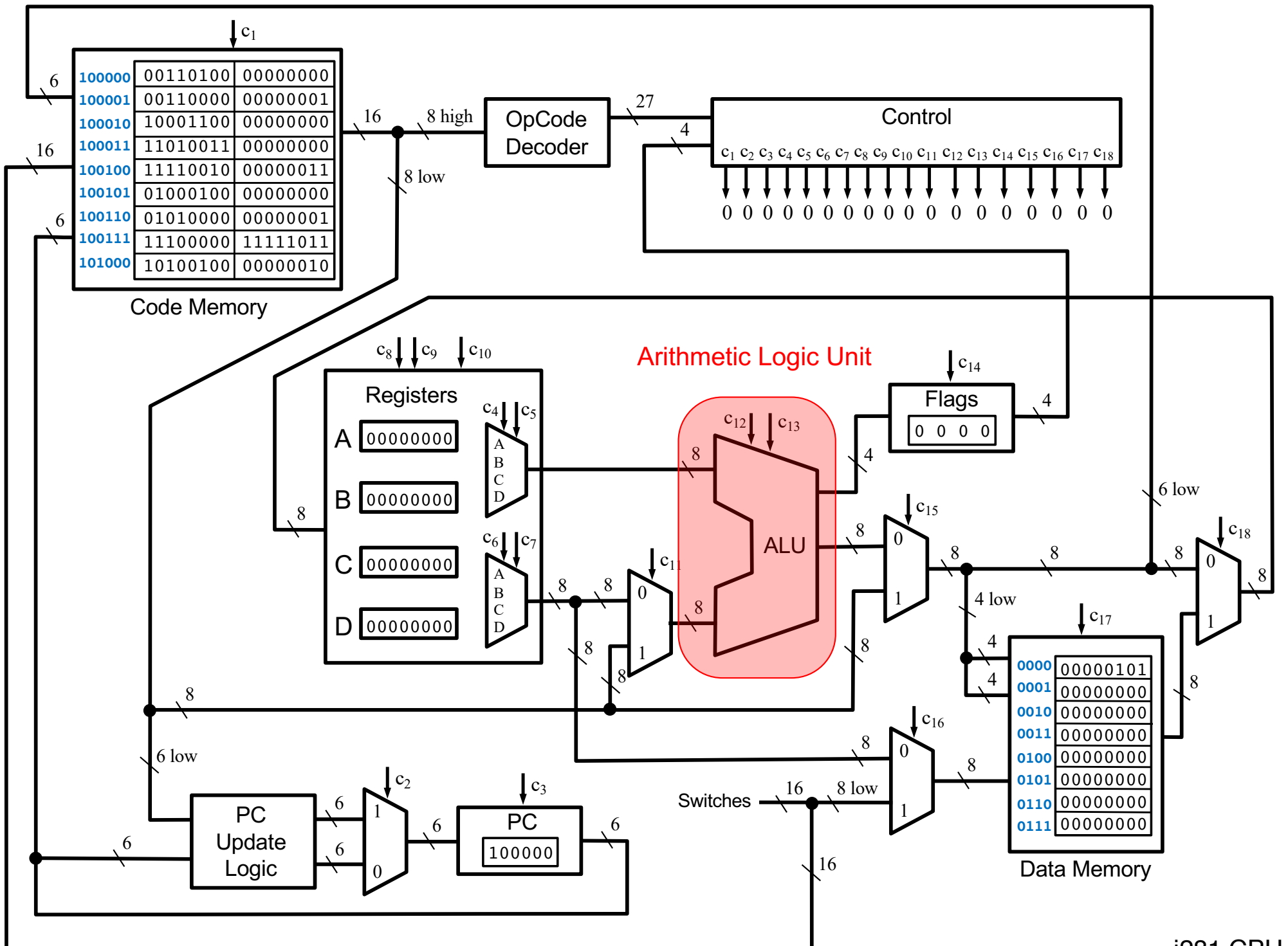


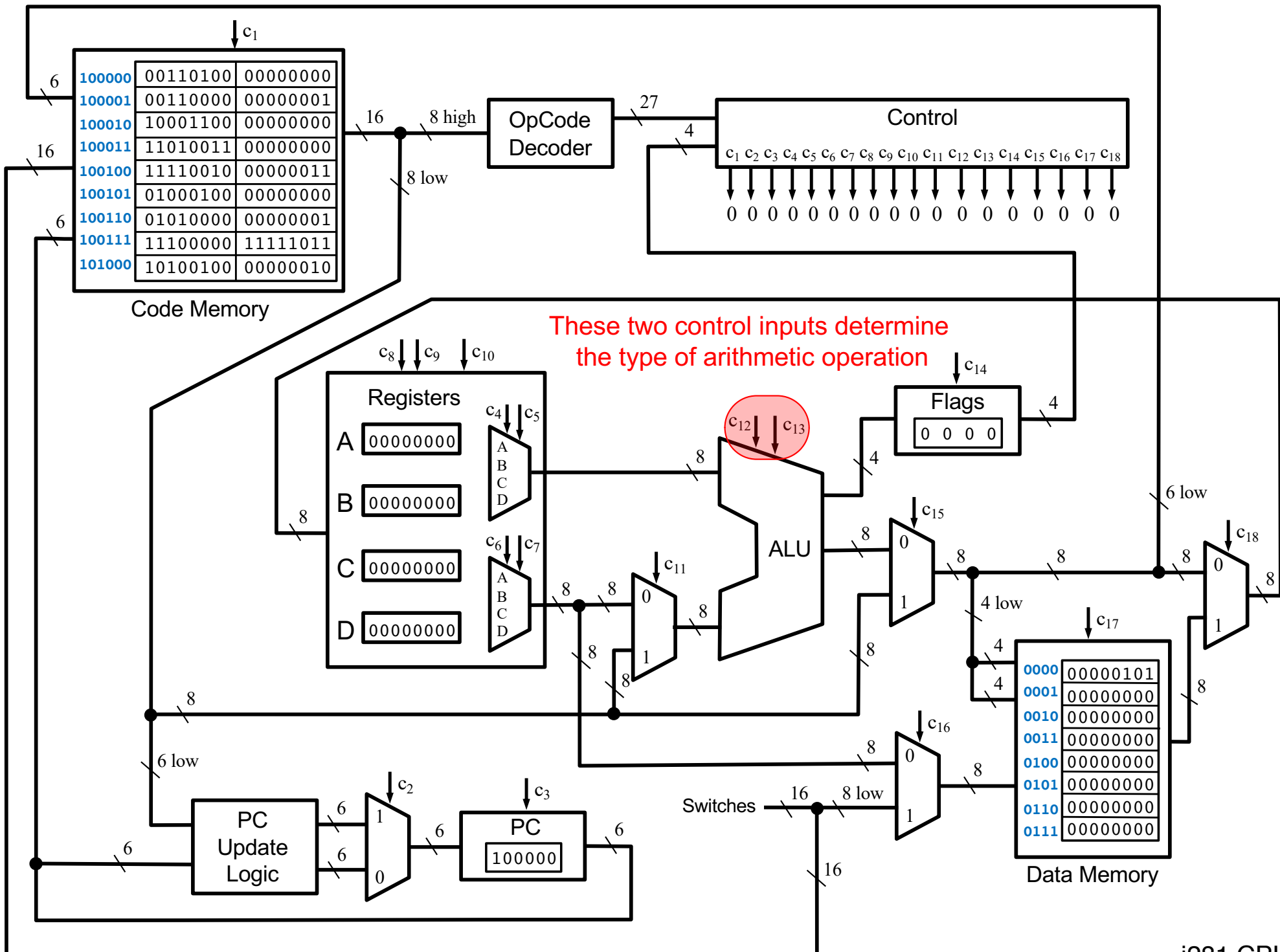






# **The Arithmetic Logic Unit (ALU)**





# This ALU Can Perform 4 Operations

<b>ALU_SELECT1</b>	<b>ALU_SELECT0</b>	<b>Operation</b>
<b>0</b>	<b>0</b>	<b>SHIFTL</b>
<b>0</b>	<b>1</b>	<b>SHIFTR</b>
<b>1</b>	<b>0</b>	<b>ADD</b>
<b>1</b>	<b>1</b>	<b>SUB/CMP</b>

# This ALU Can Perform 4 Operations

Names of these  
control lines

**C<sub>12</sub>**

**C<sub>13</sub>**

<b>ALU_SELECT1</b>	<b>ALU_SELECT0</b>	<b>Operation</b>
<b>0</b>	<b>0</b>	<b>SHIFTL</b>
<b>0</b>	<b>1</b>	<b>SHIFTR</b>
<b>1</b>	<b>0</b>	<b>ADD</b>
<b>1</b>	<b>1</b>	<b>SUB/CMP</b>

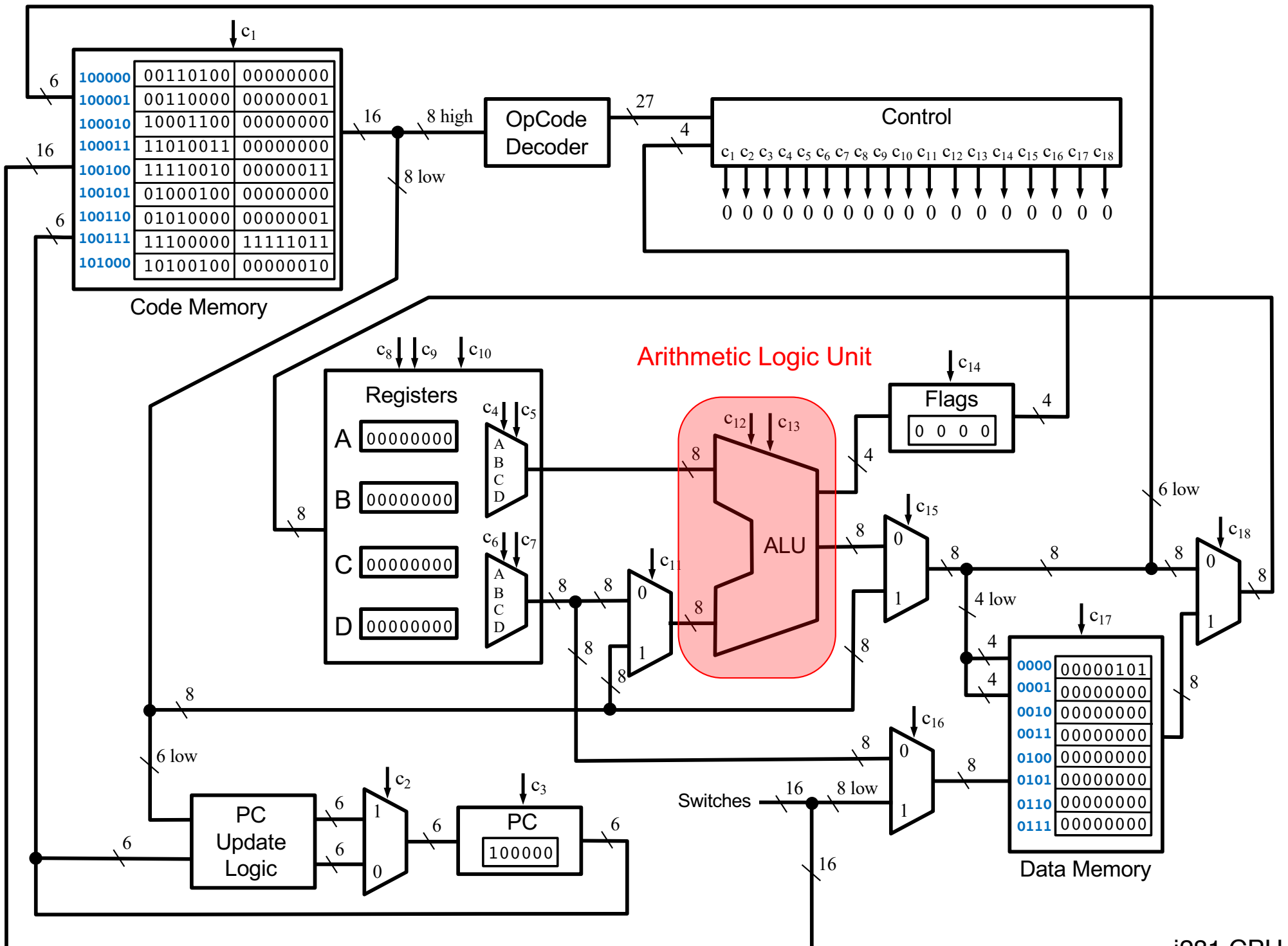
# This ALU Can Perform 4 Operations

$C_{12}$	$C_{13}$	
ALU_SELECT1	ALU_SELECT0	Operation
0	0	SHIFTL
0	1	SHIFTR
1	0	ADD
1	1	SUB / CMP

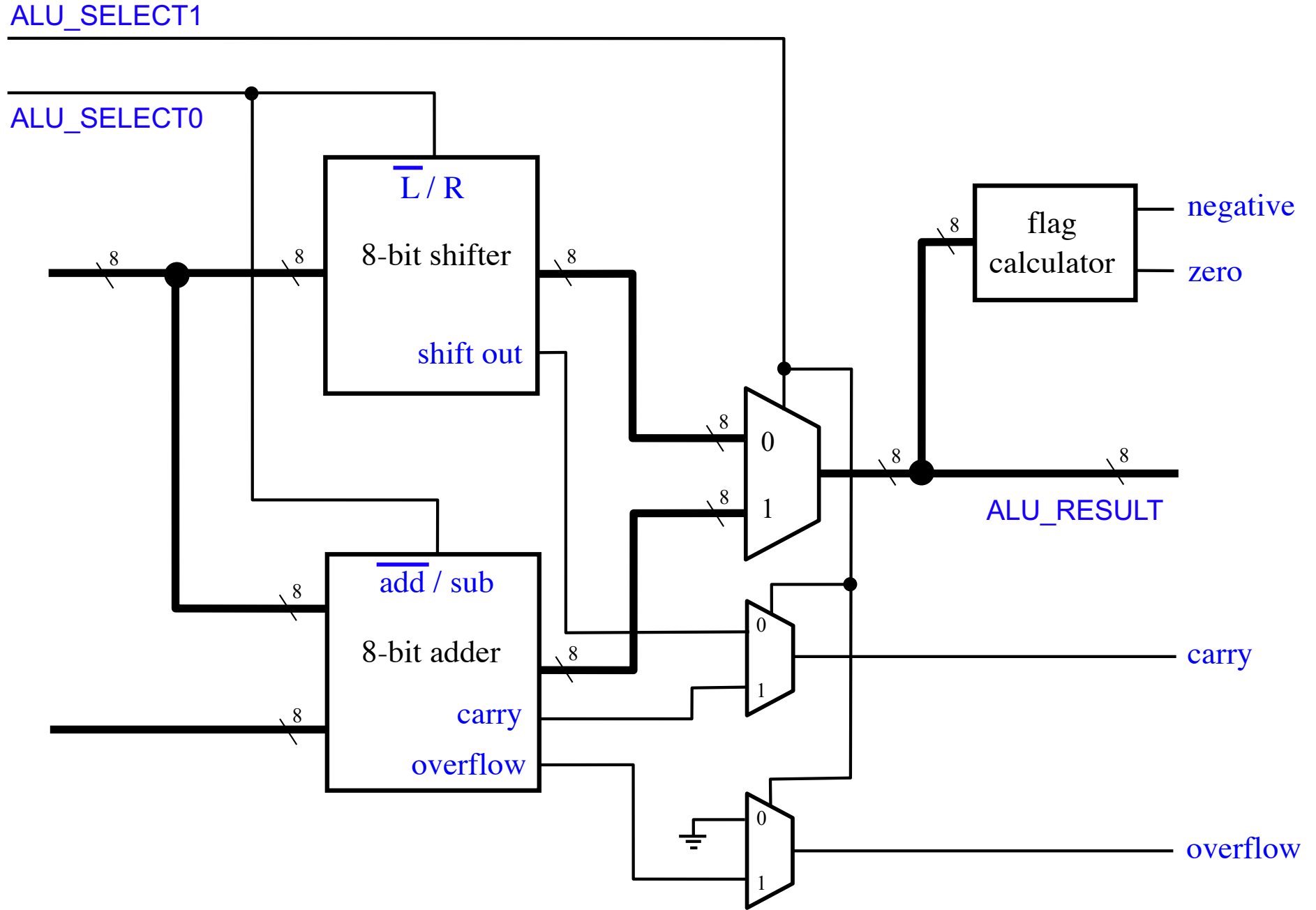
Both SUB and CMP are implemented as subtraction. They both set the flags.

The difference is that CMP does not write back the result of the subtraction to the registers. Only the side effect through the flags remains.

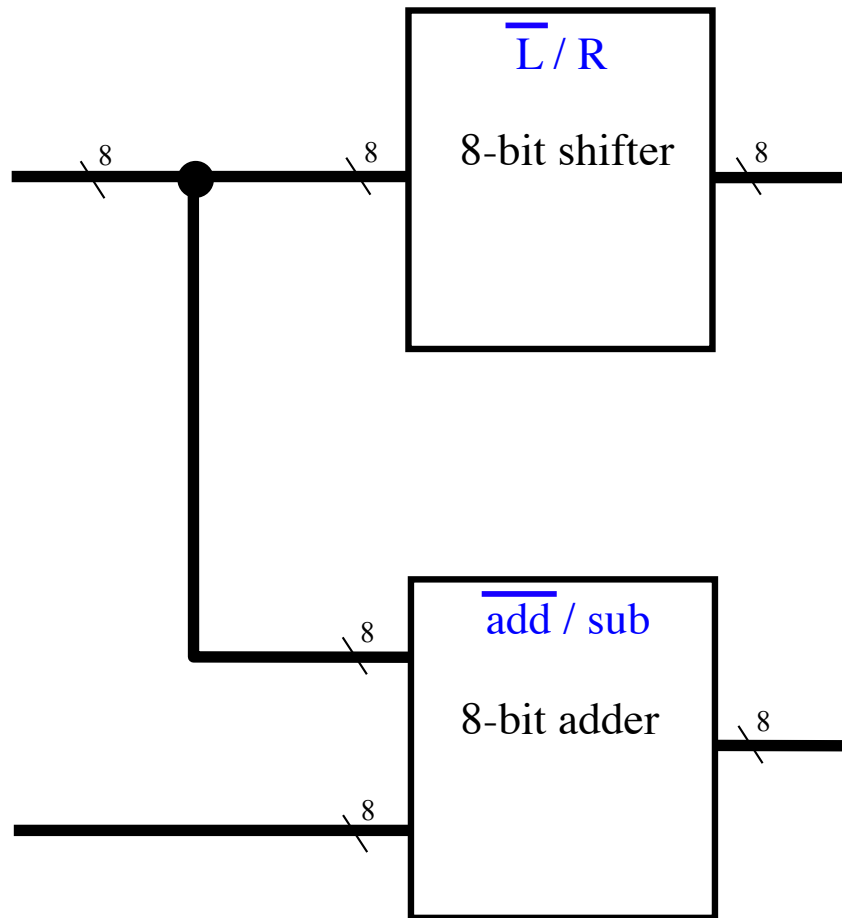




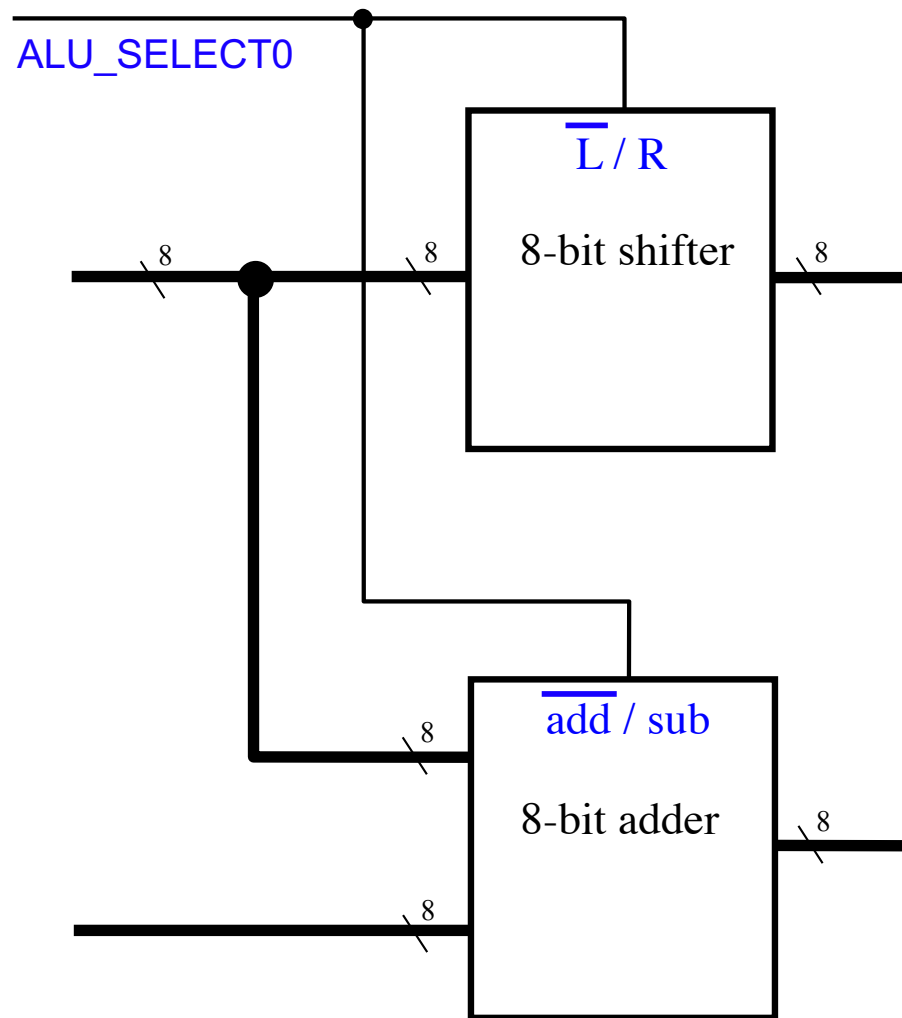
# The ALU



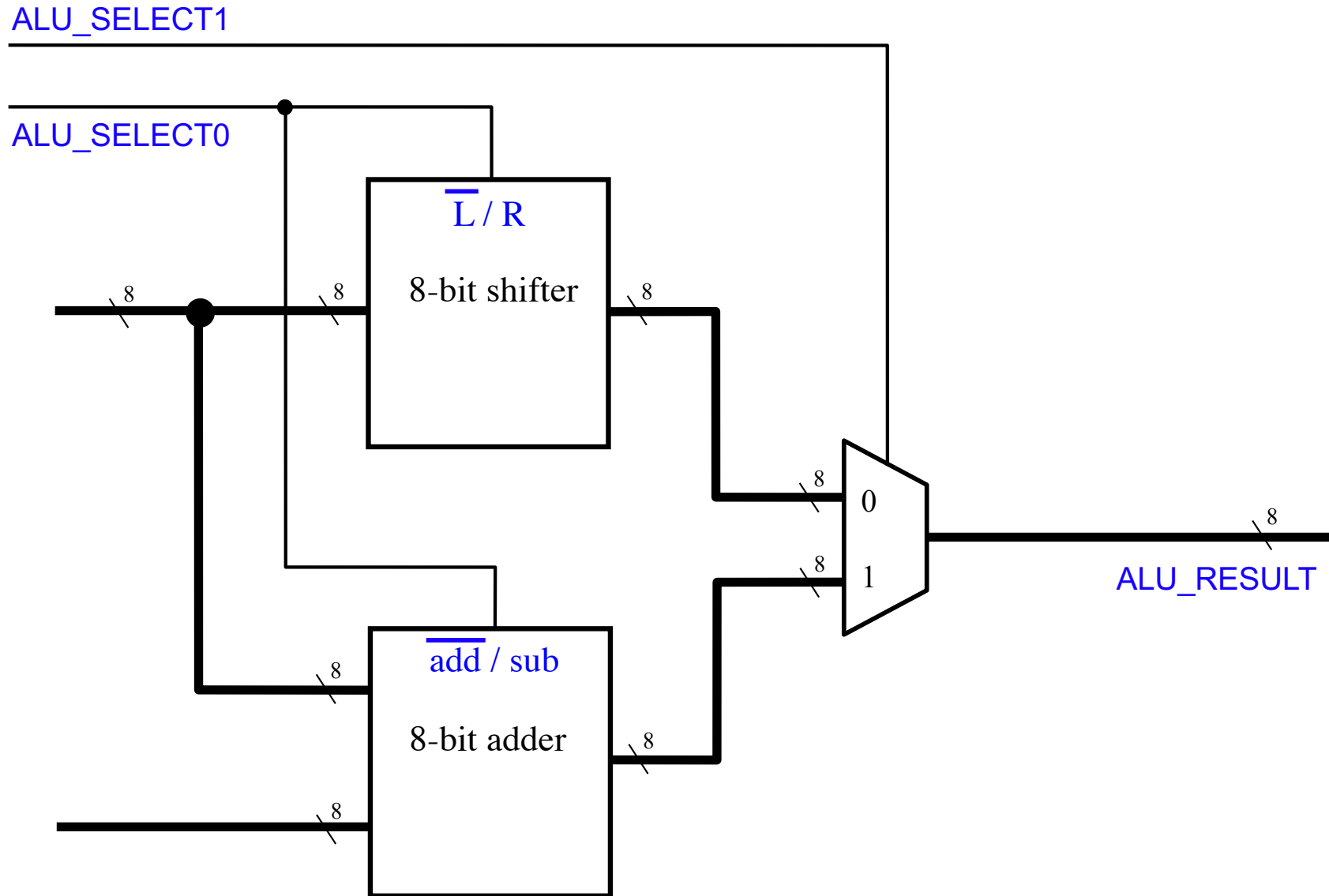
# The ALU



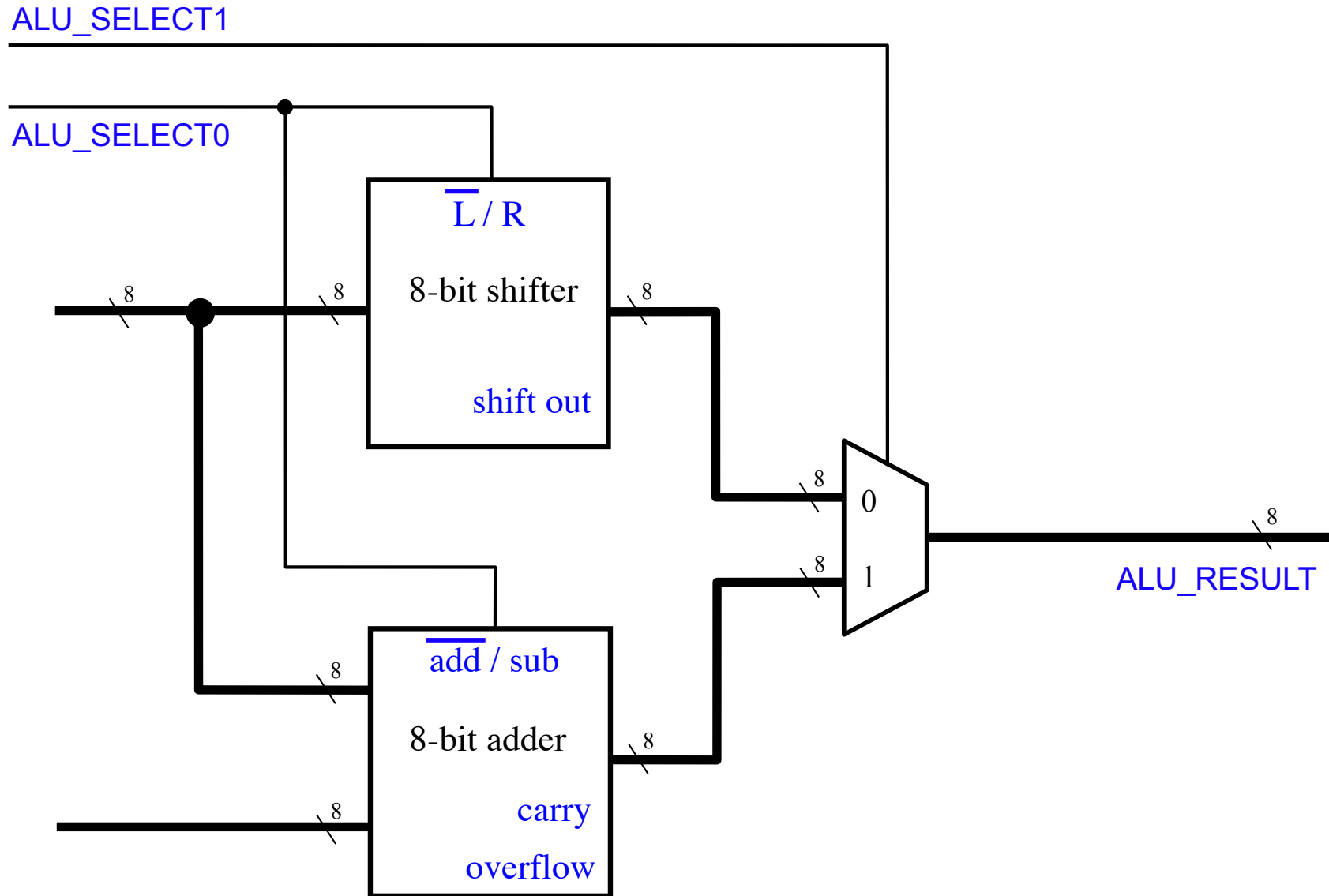
# The ALU



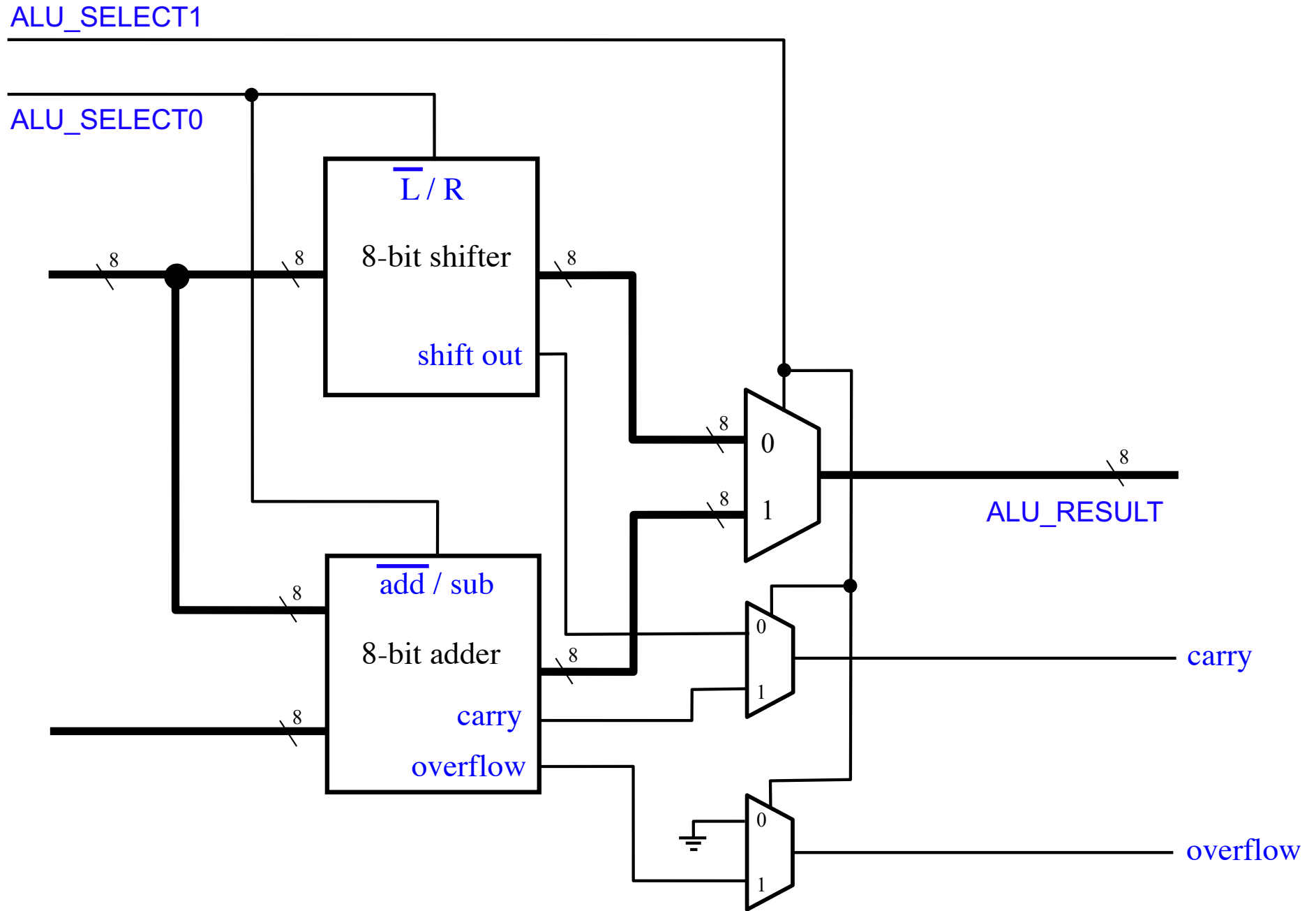
# The ALU



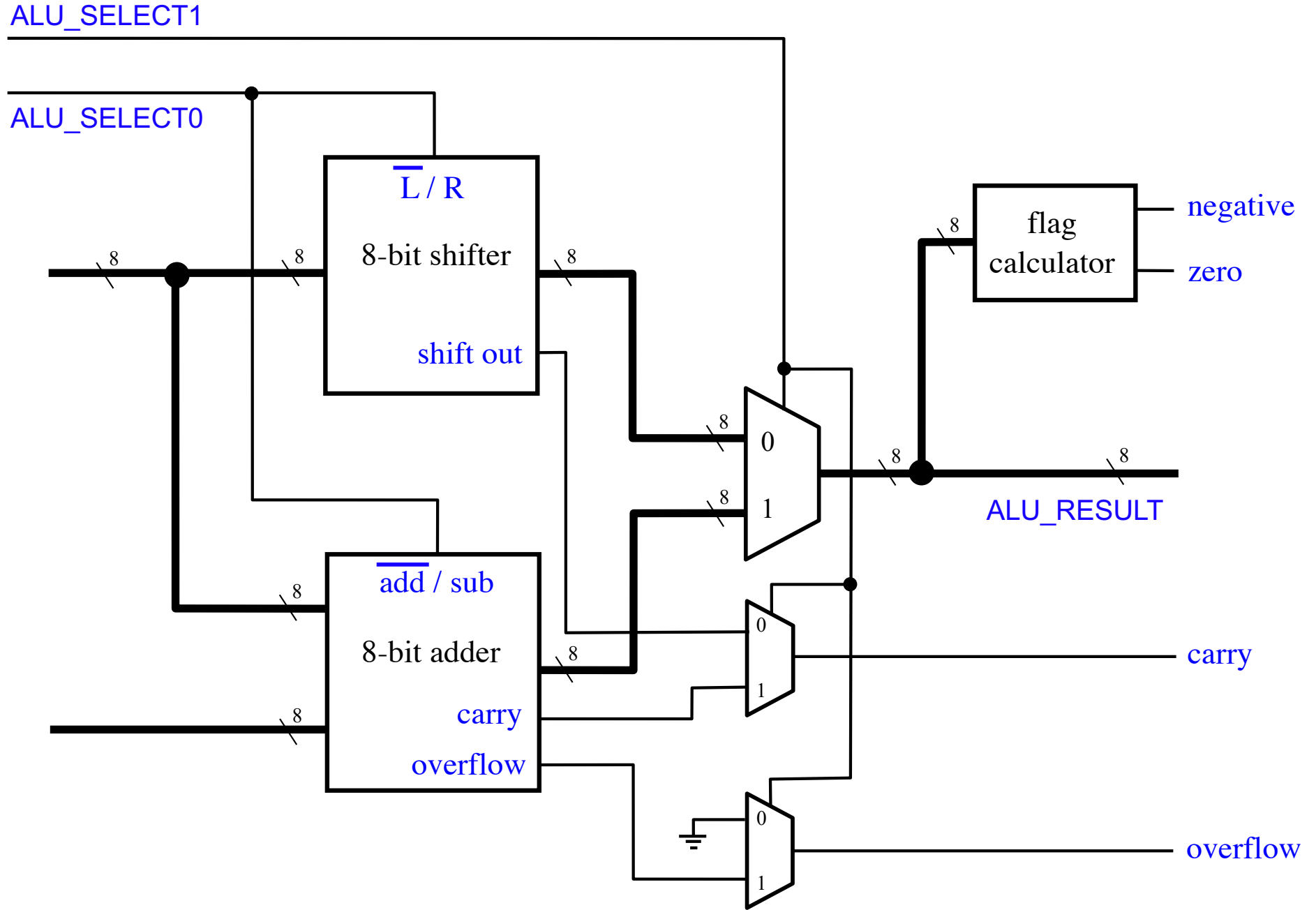
# The ALU



# The ALU



# The ALU

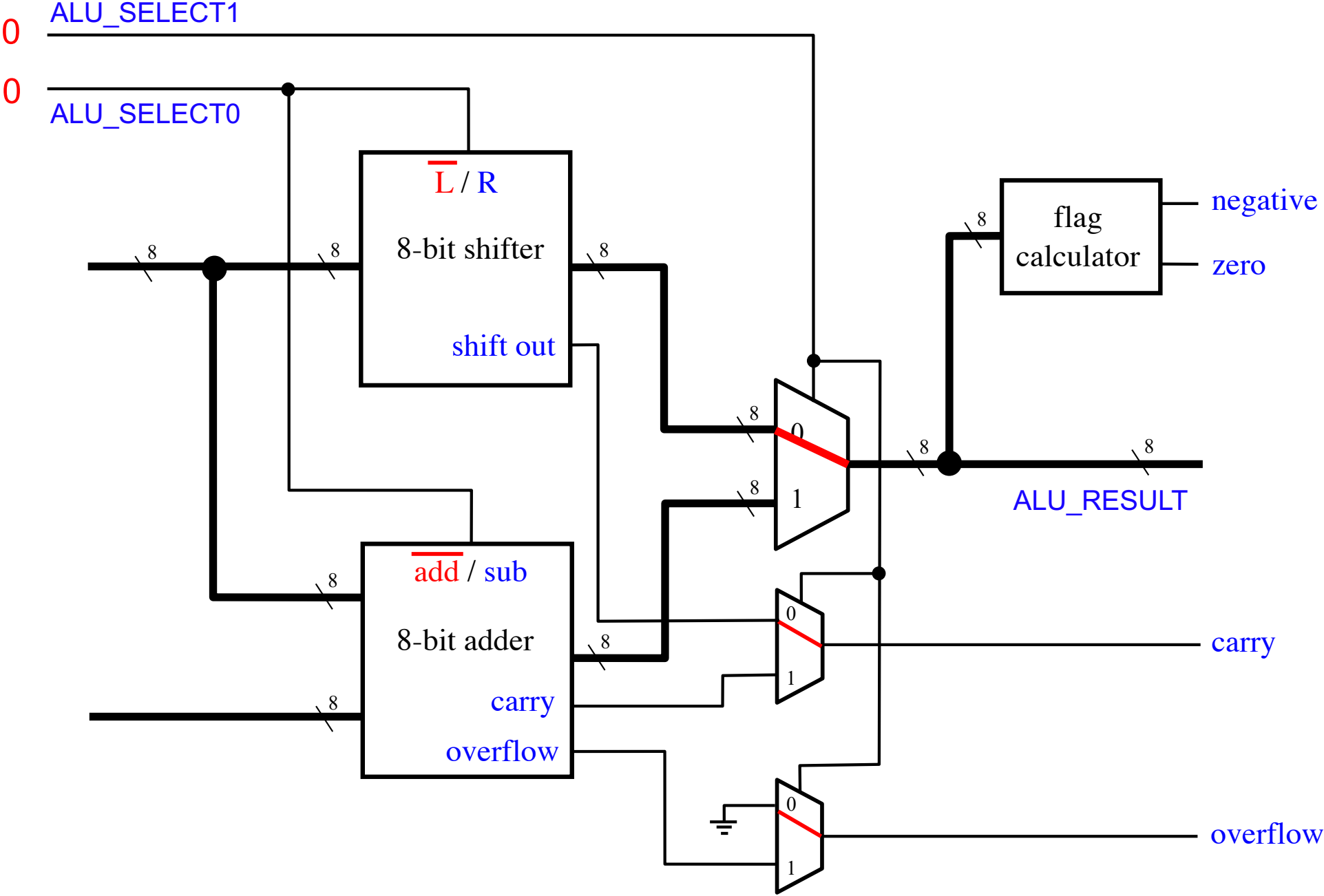




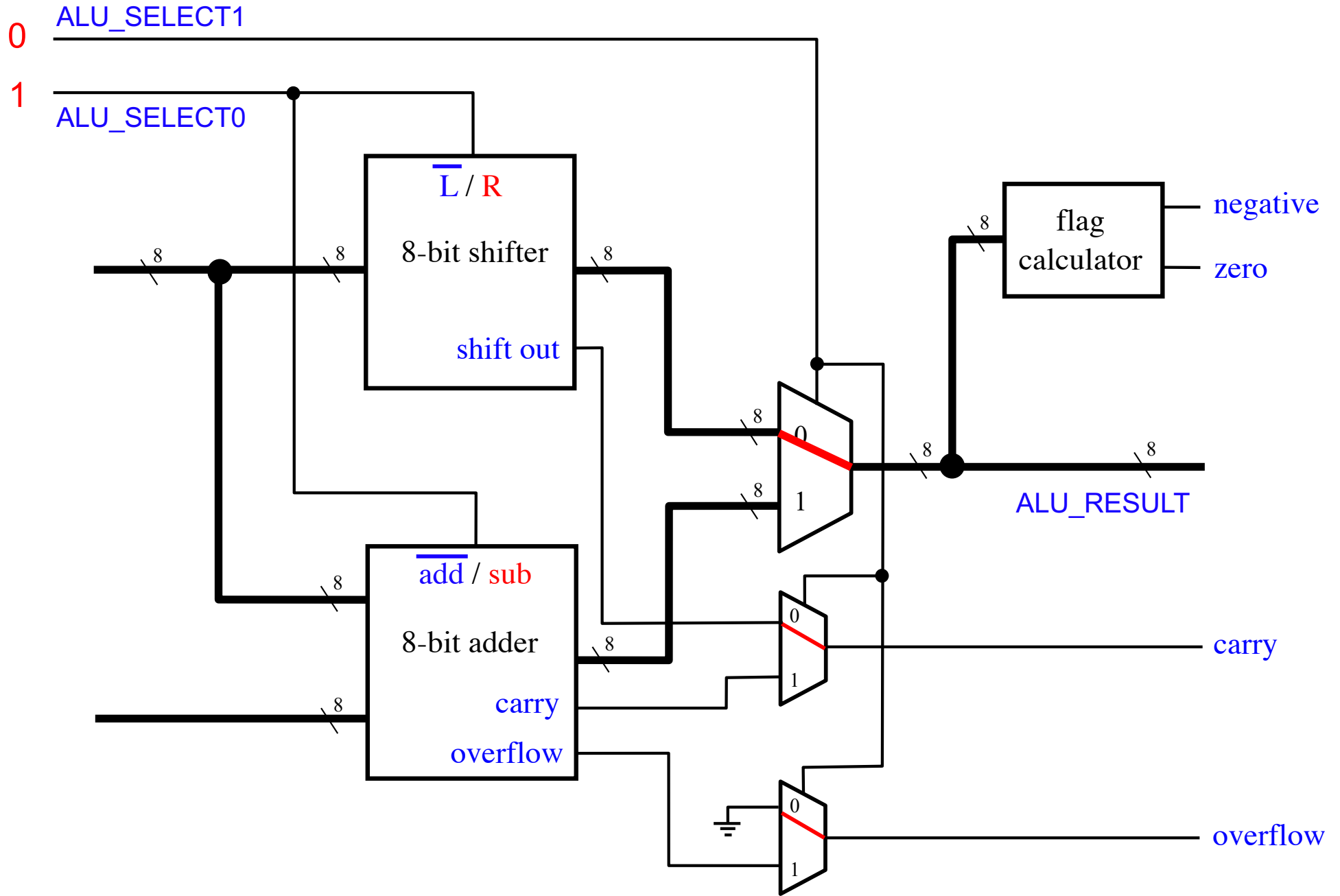
# This ALU Can Perform 4 Operations

<b>ALU_SELECT1</b>	<b>ALU_SELECT0</b>	<b>Operation</b>
<b>0</b>	<b>0</b>	<b>SHIFTL</b>
<b>0</b>	<b>1</b>	<b>SHIFTR</b>
<b>1</b>	<b>0</b>	<b>ADD</b>
<b>1</b>	<b>1</b>	<b>SUB/CMP</b>

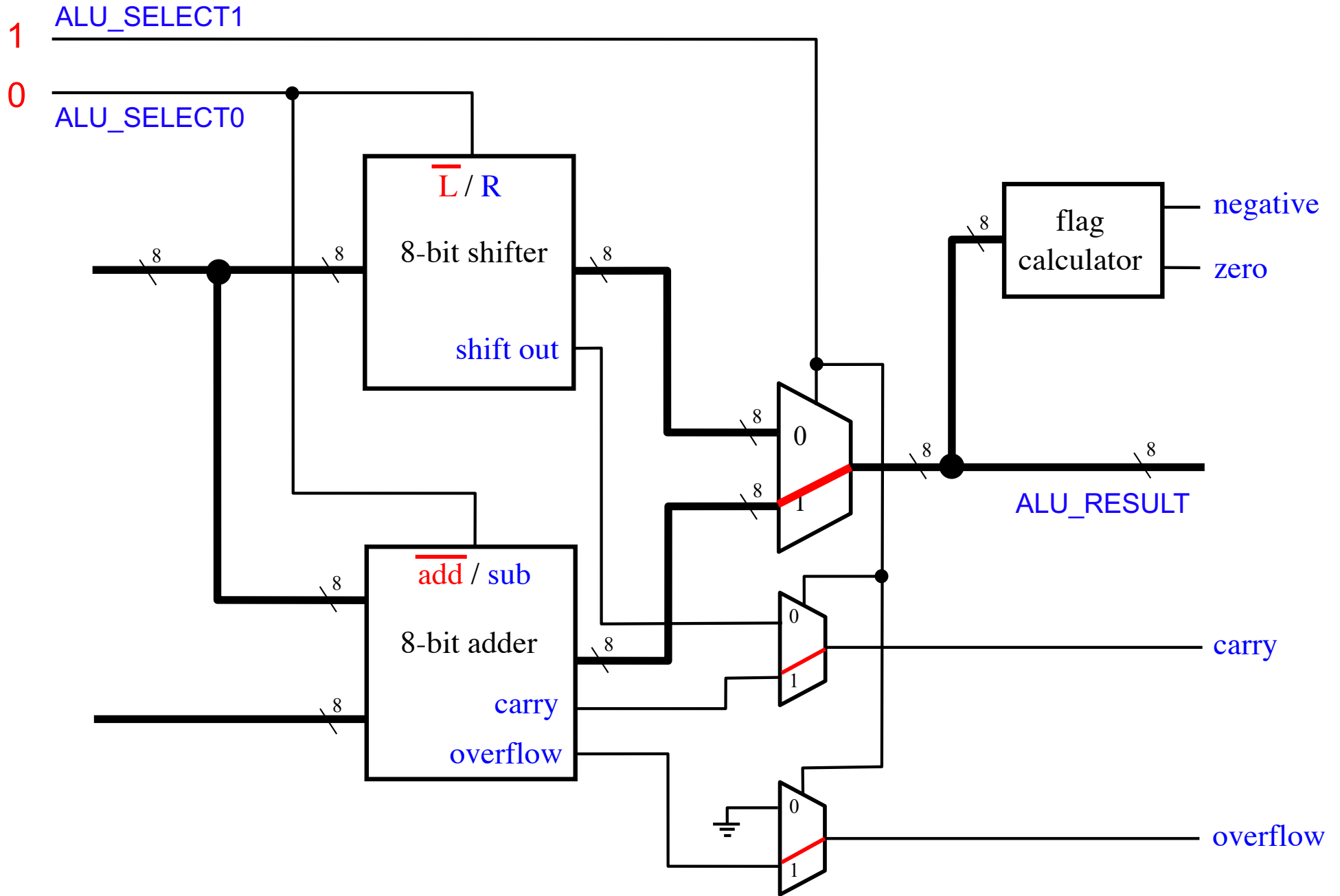
# SHIFTL



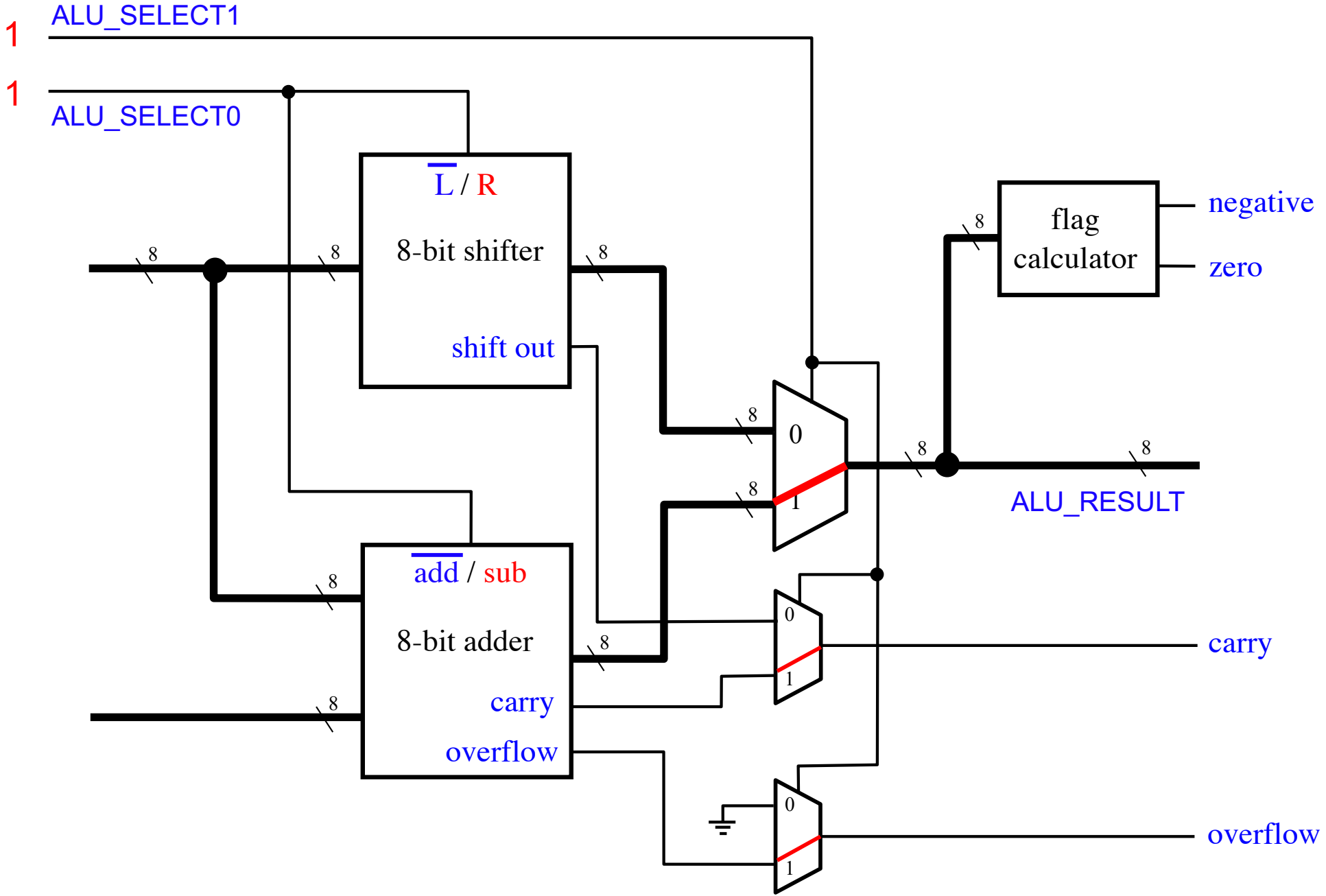
# SHIFTR



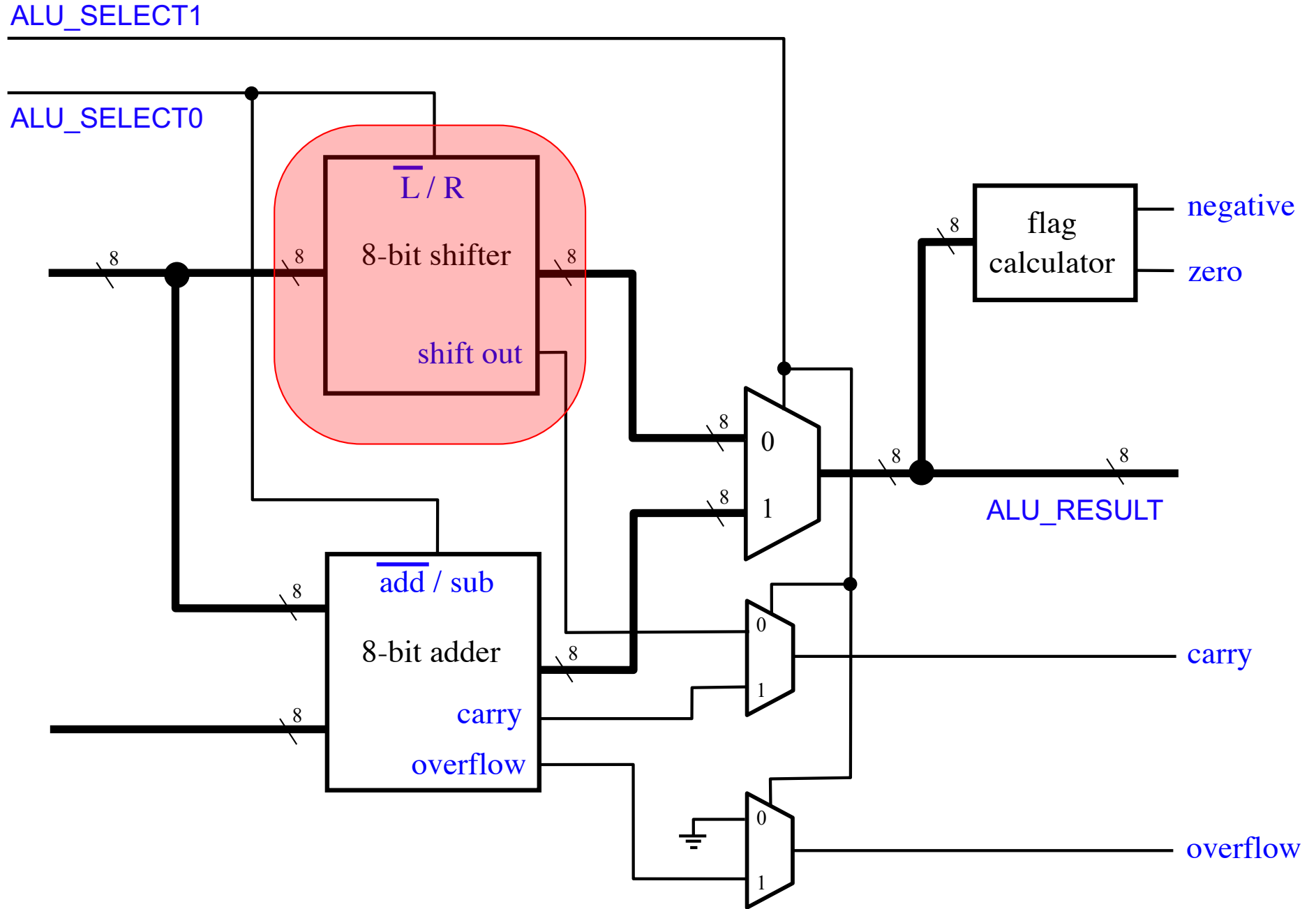
# ADD



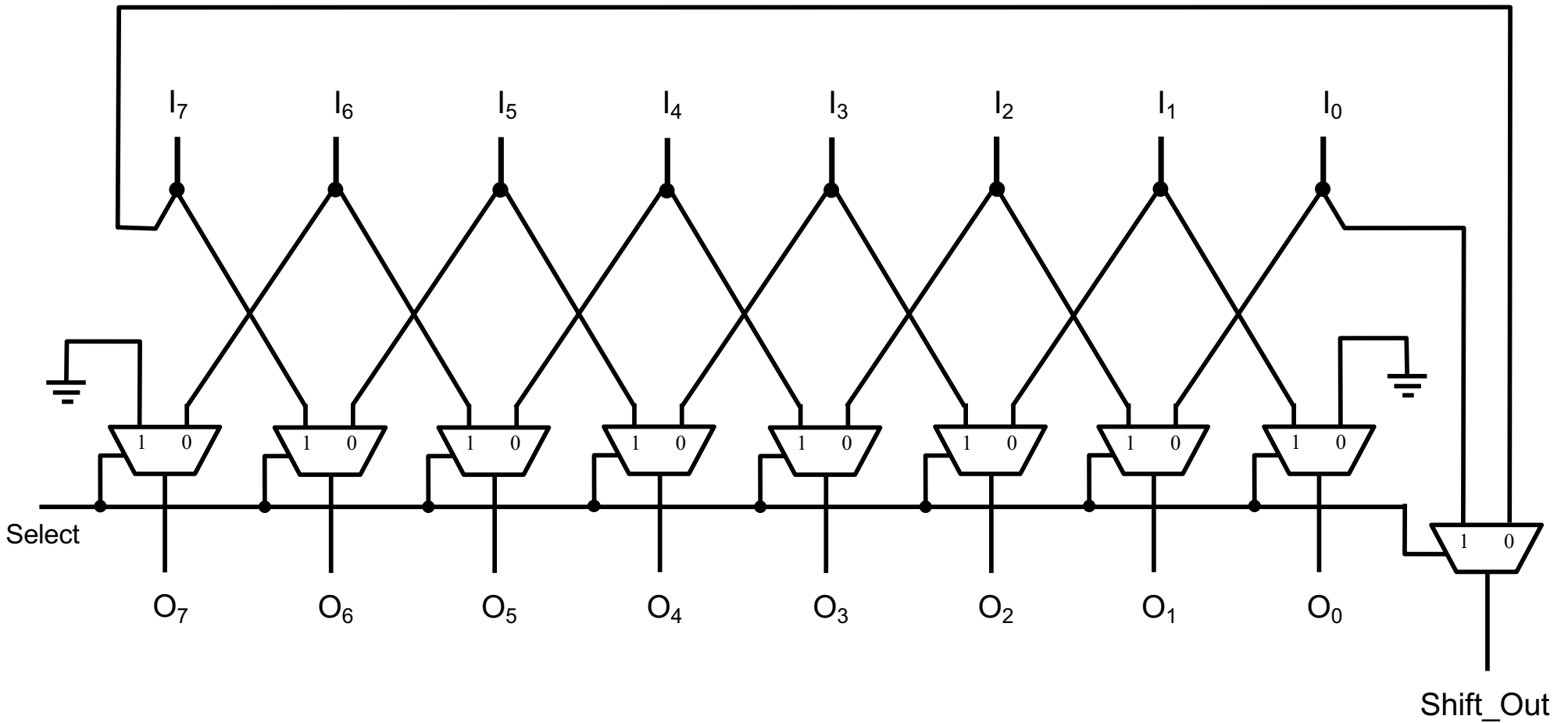
# SUB / CMP

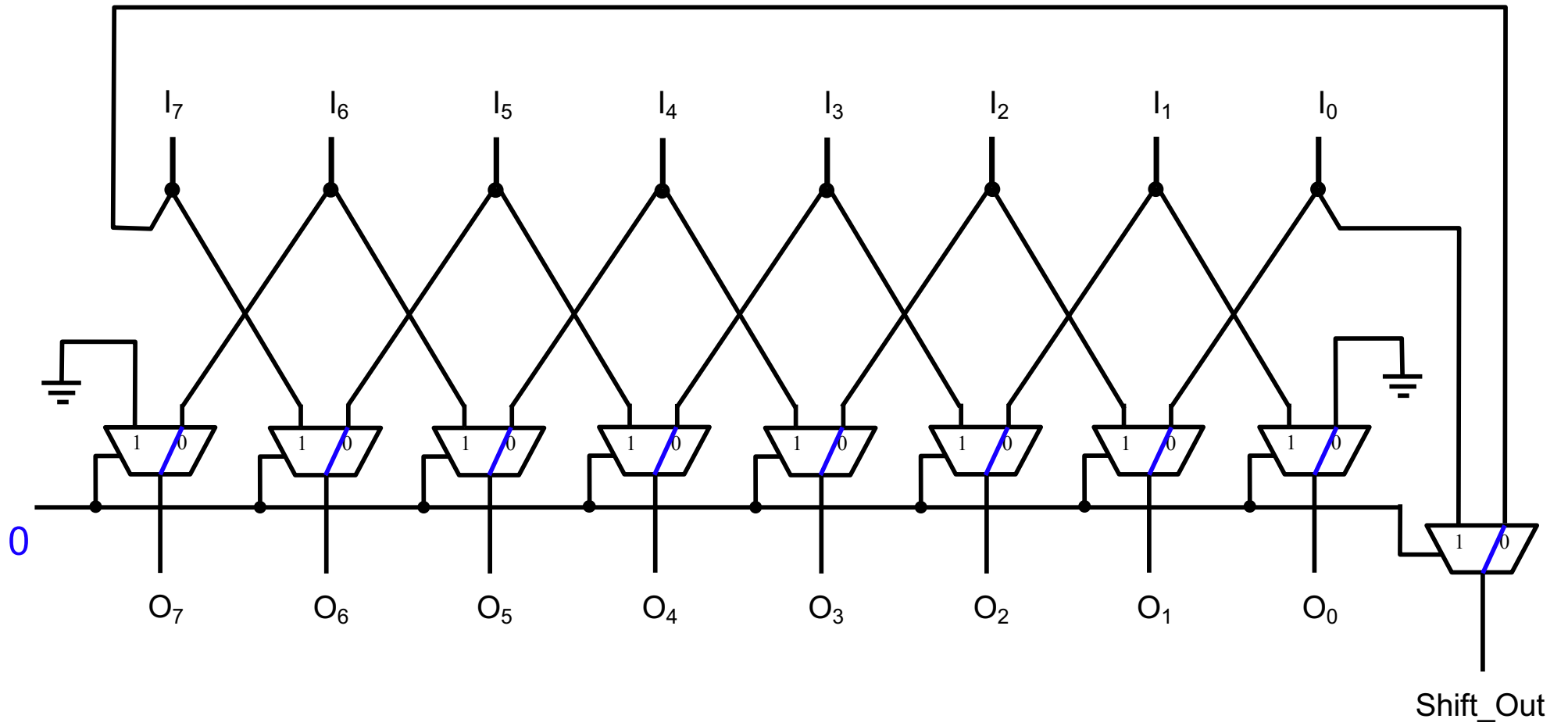


# The Shifter Circuit

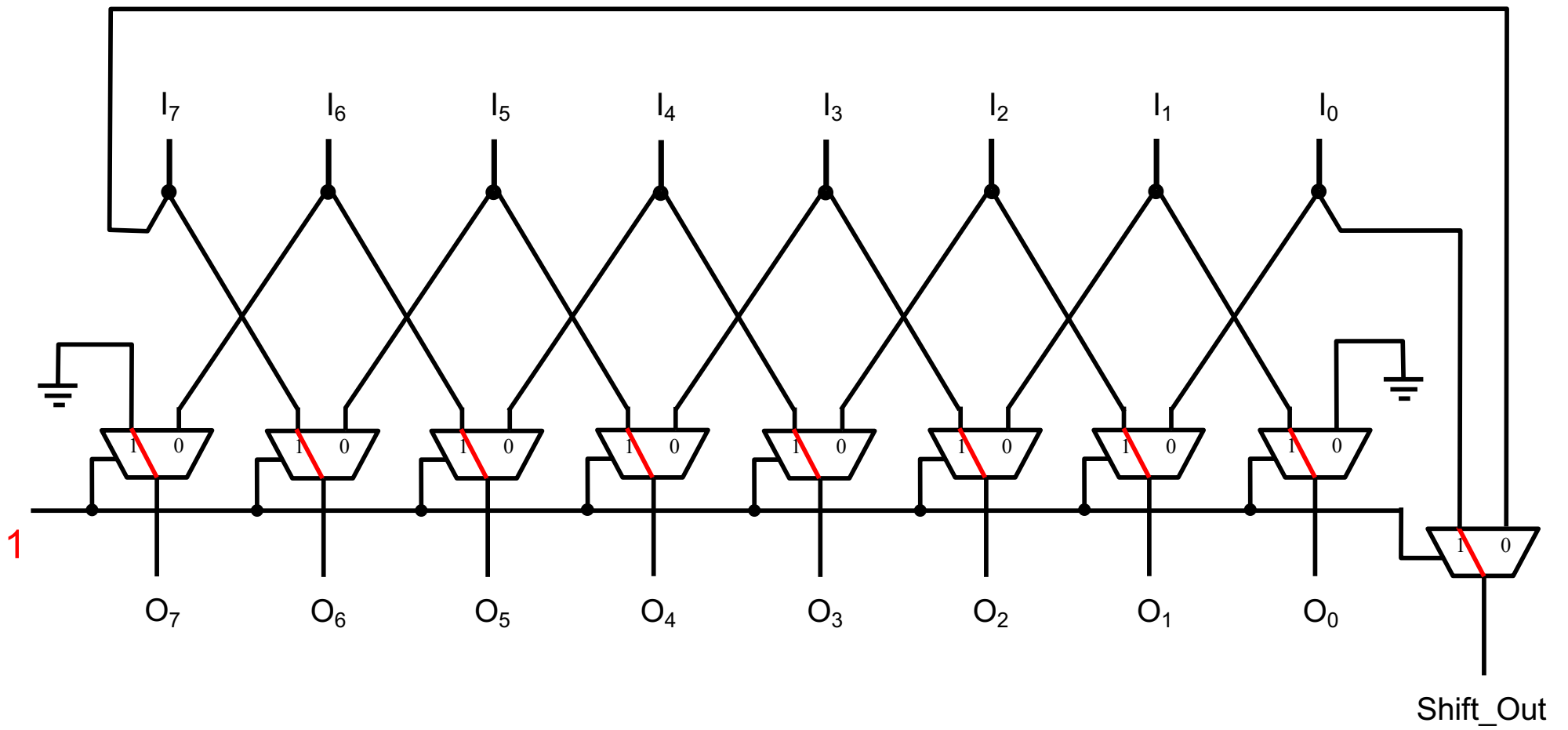


# The Shifter Circuit

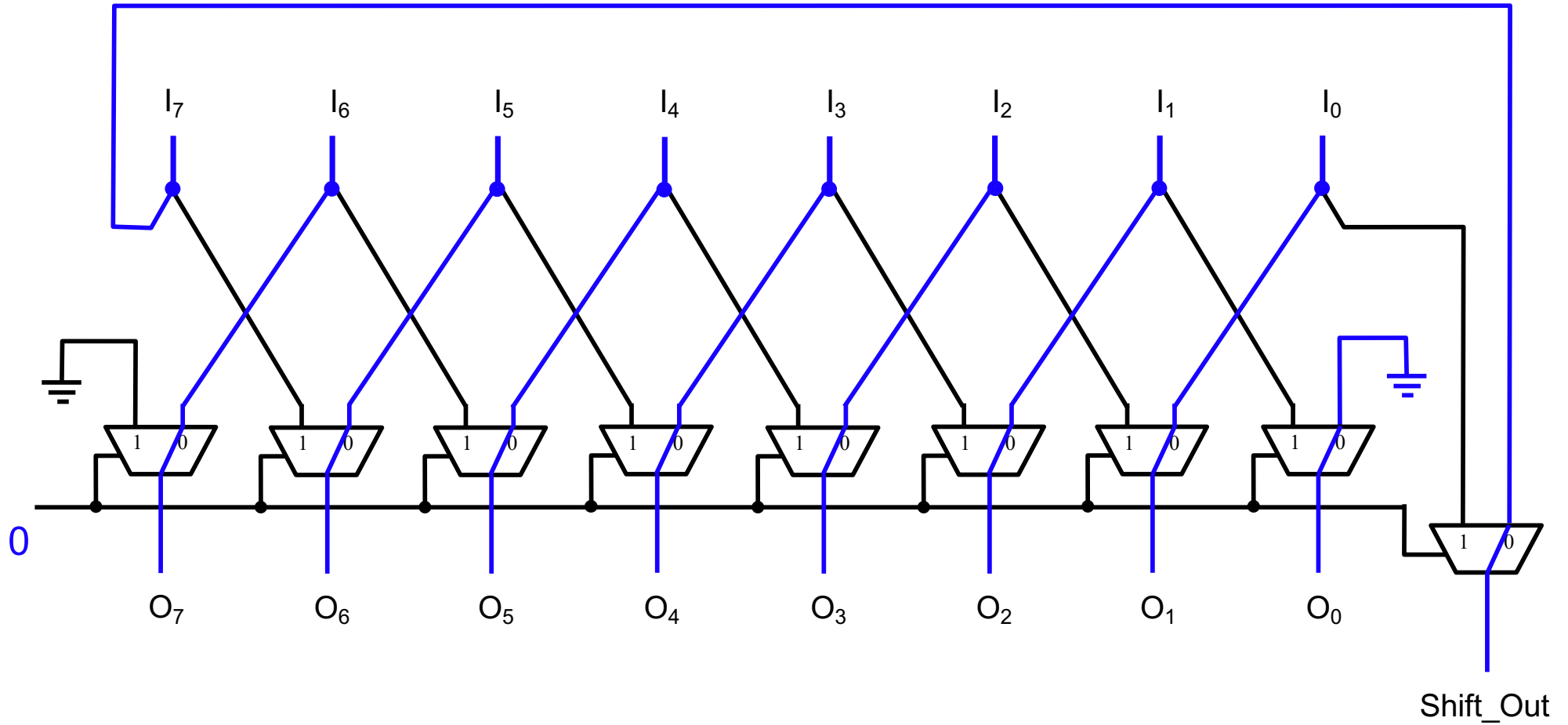




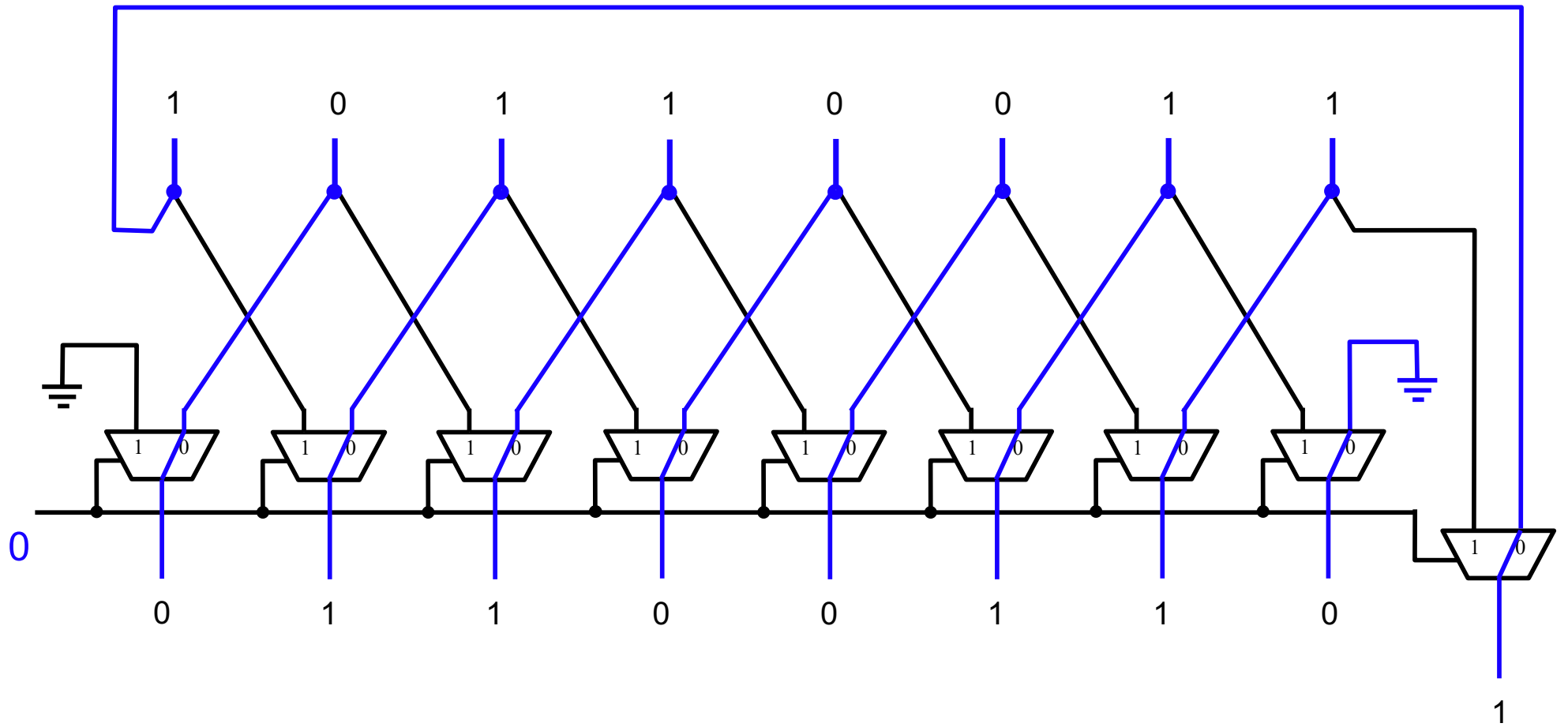




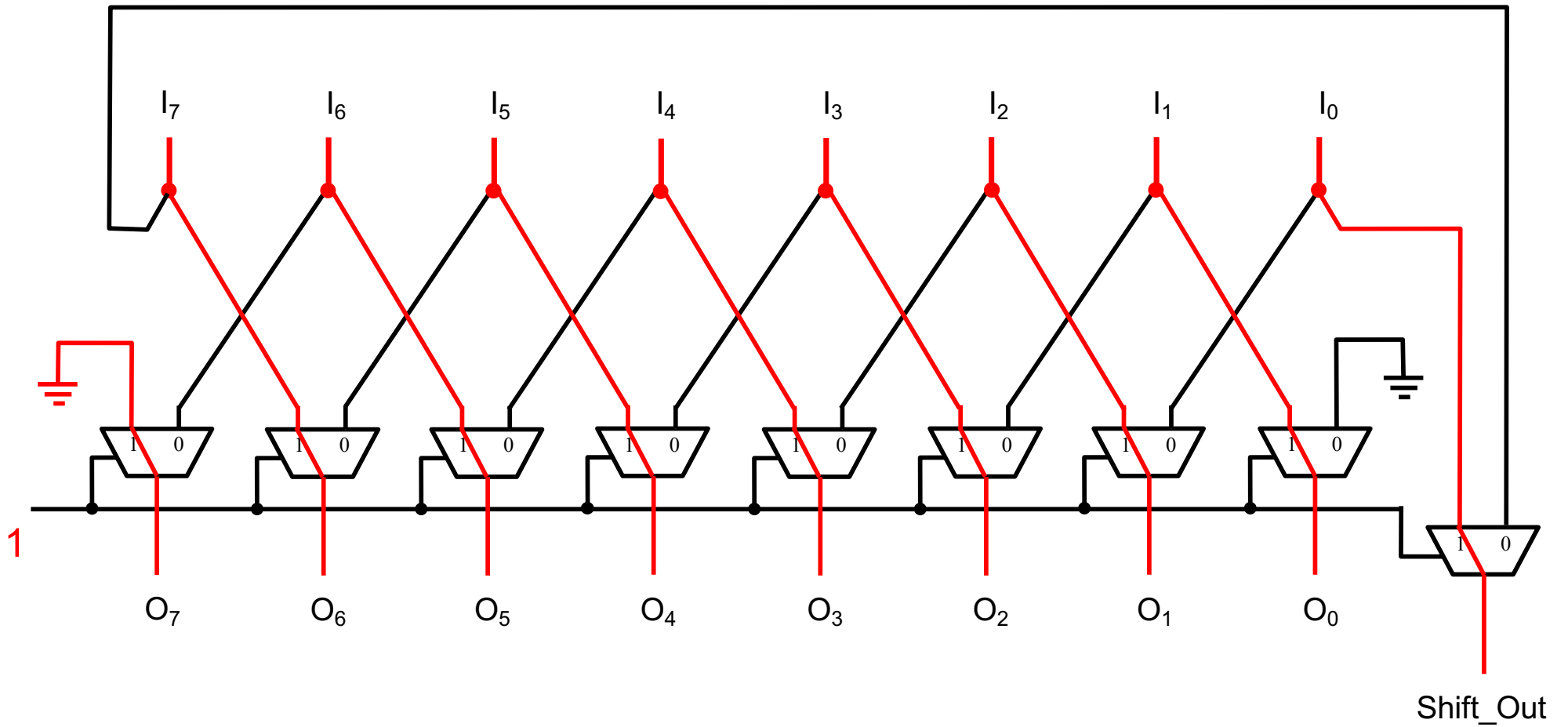
# Shift Left



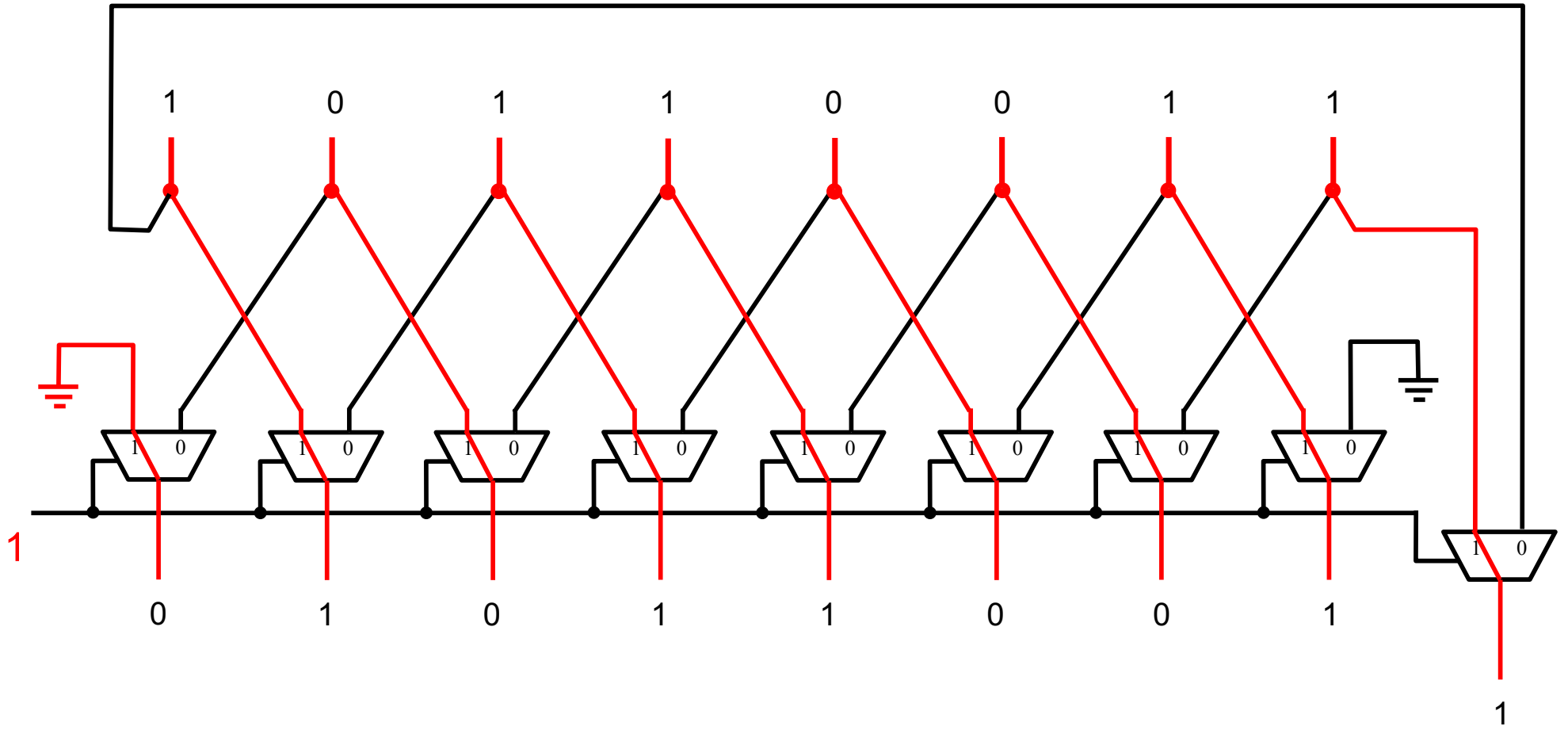
# Shift Left



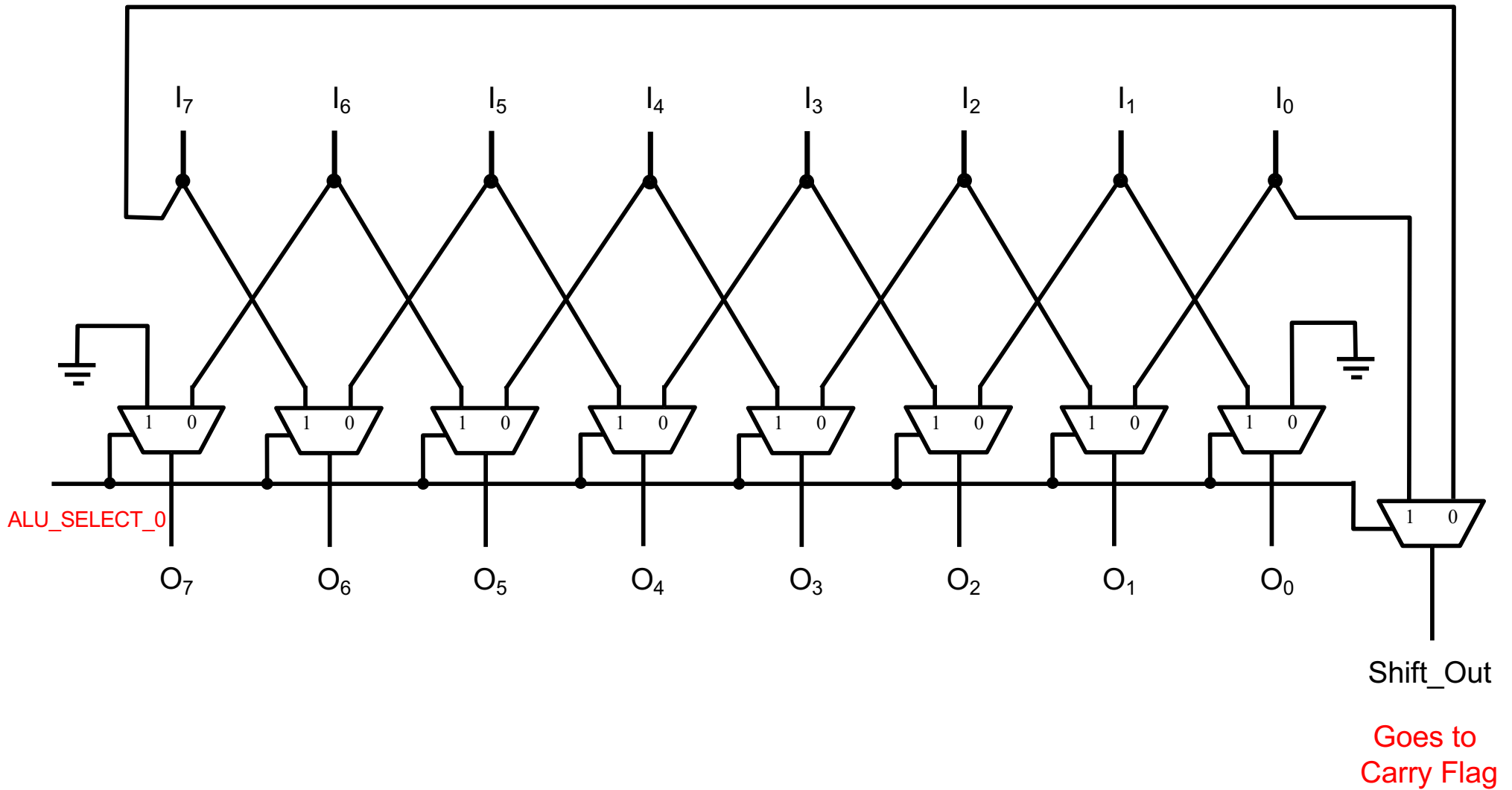
# Shift Right



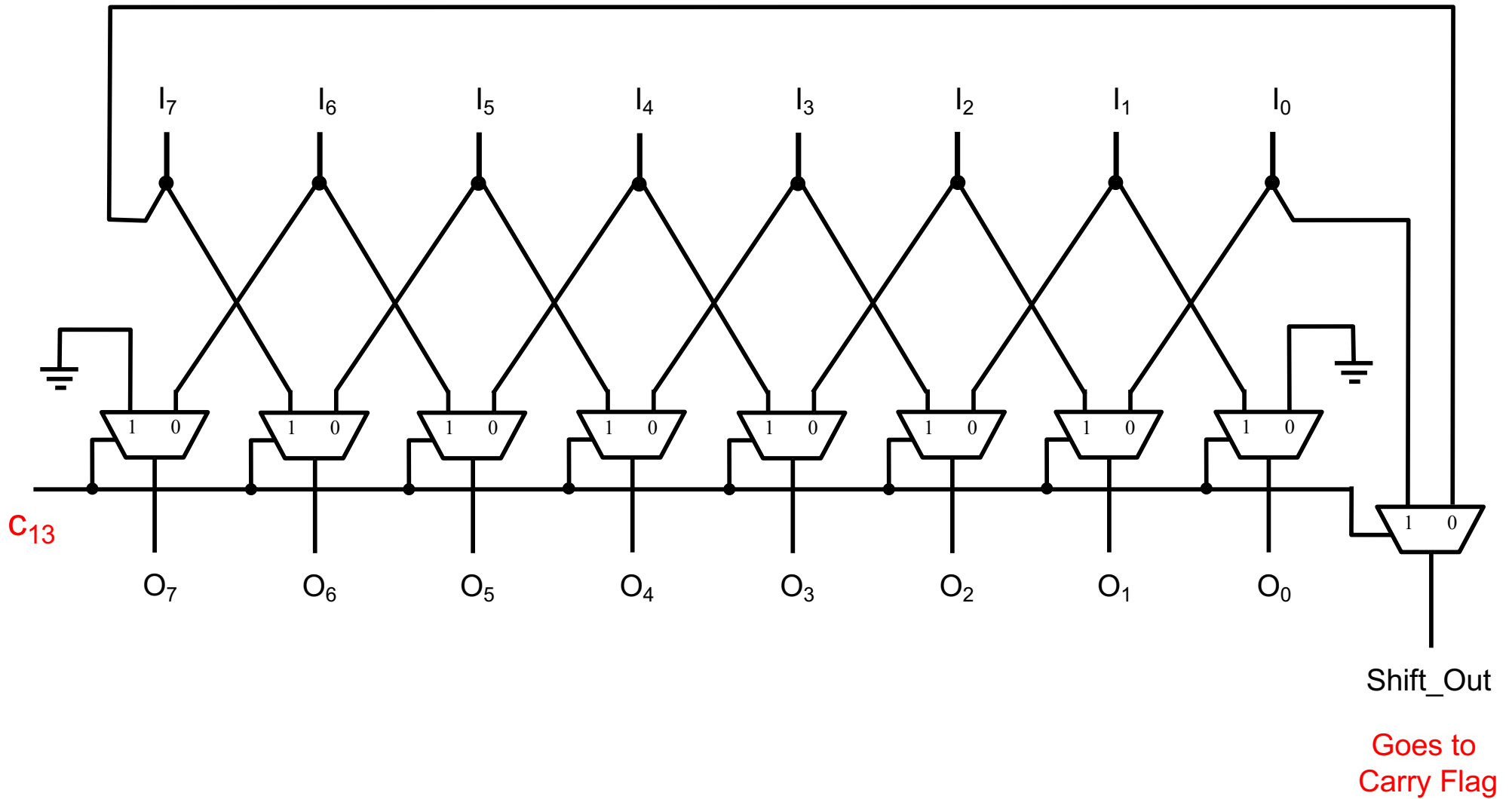
# Shift Right



# The ALU Shifter Circuit



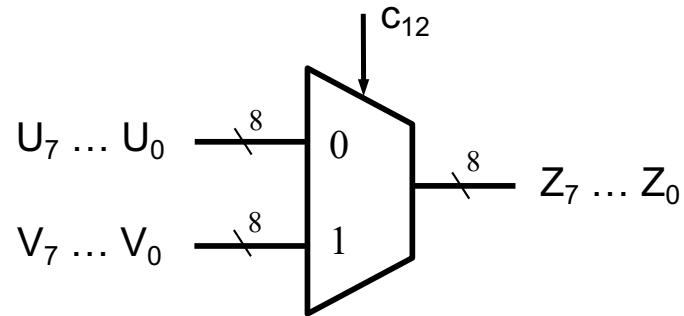
# The ALU Shifter Circuit

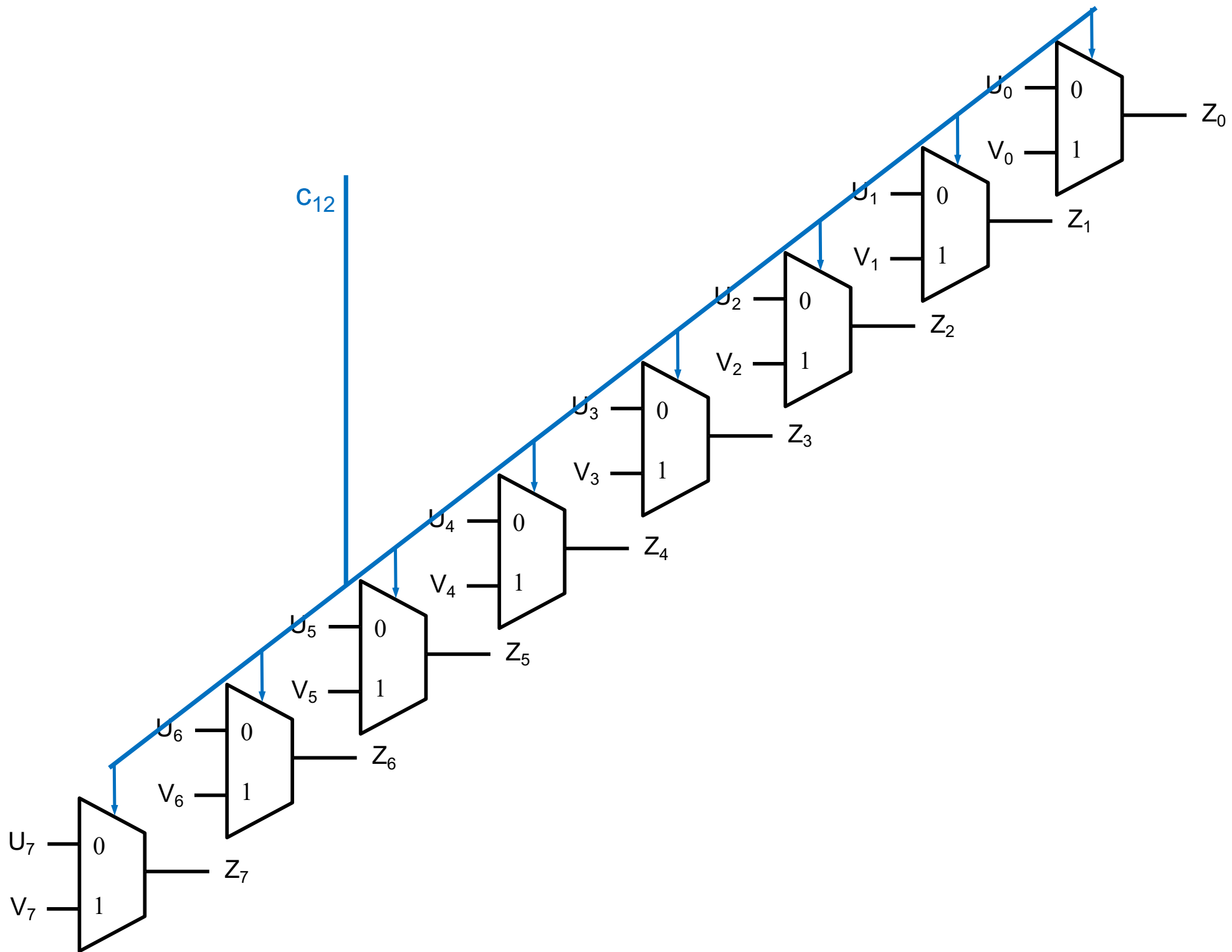


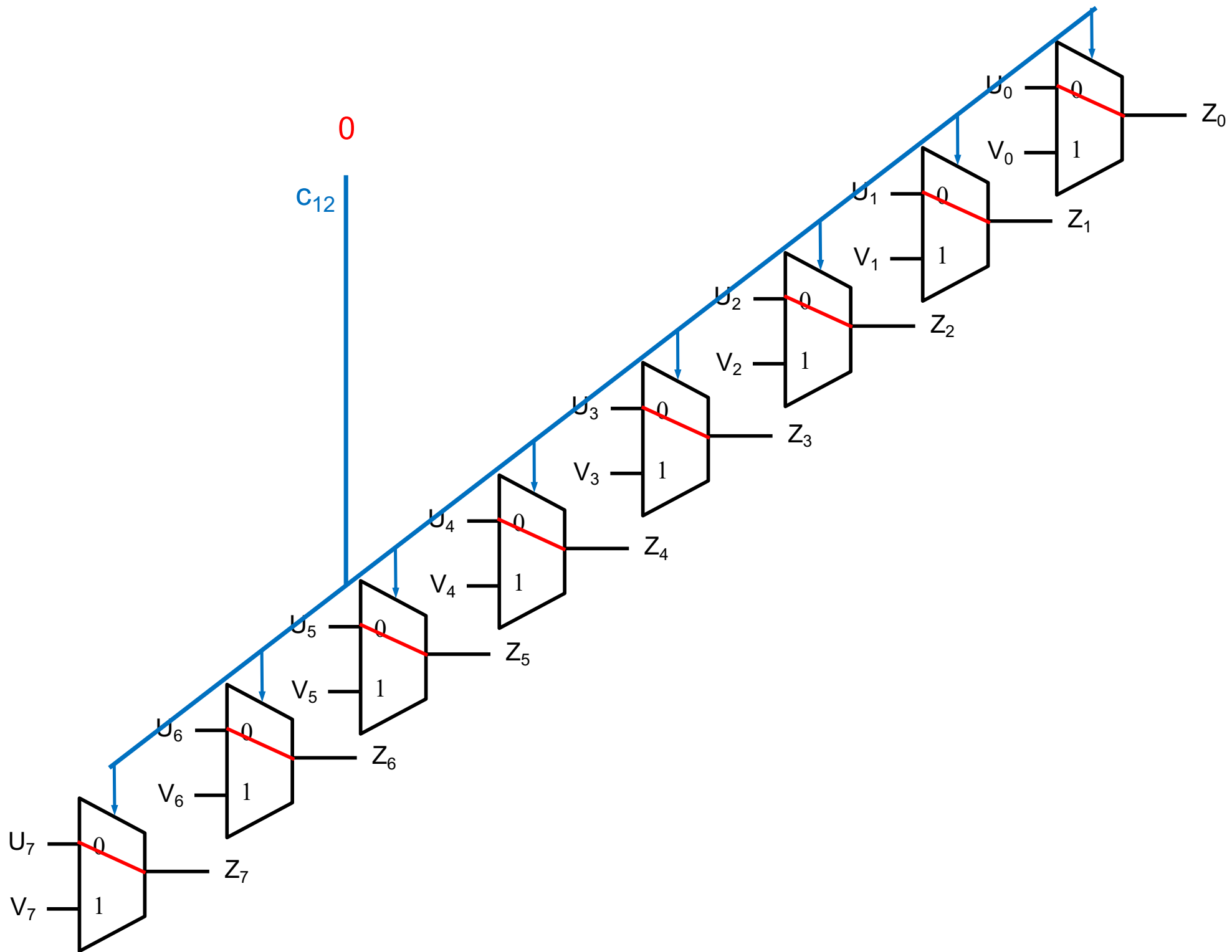


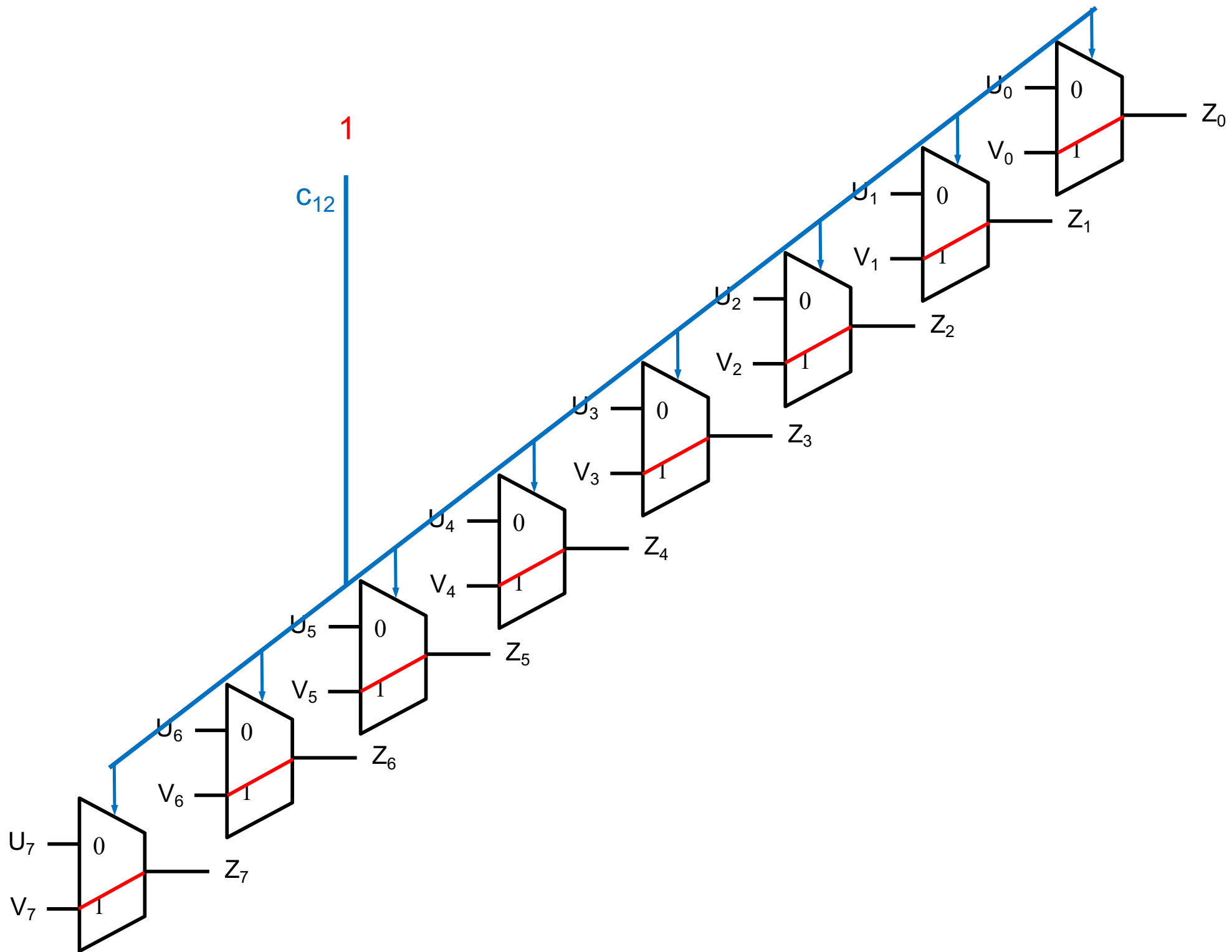


# 2-to-1 Bus Multiplexer (with 8-bit lines)

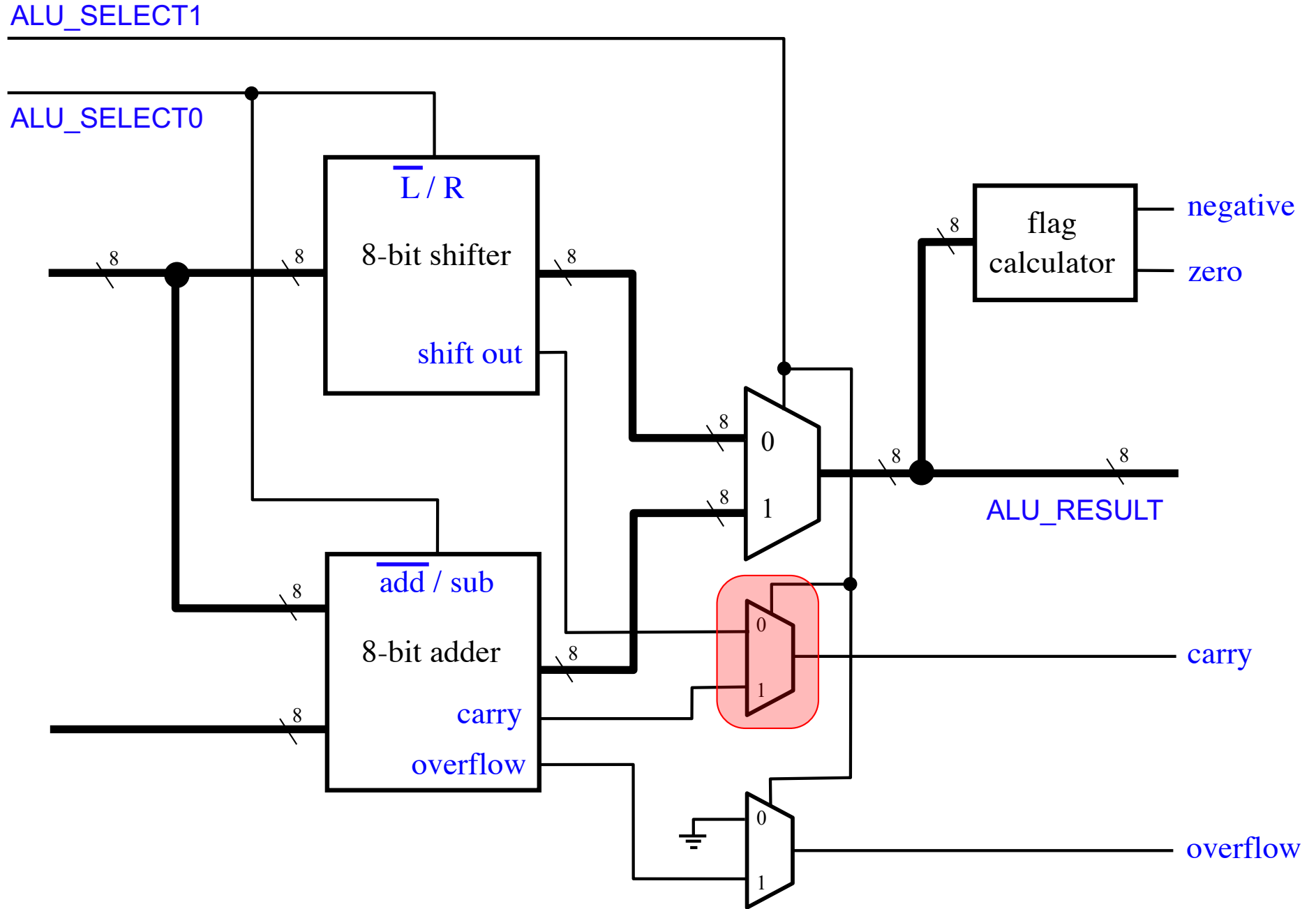




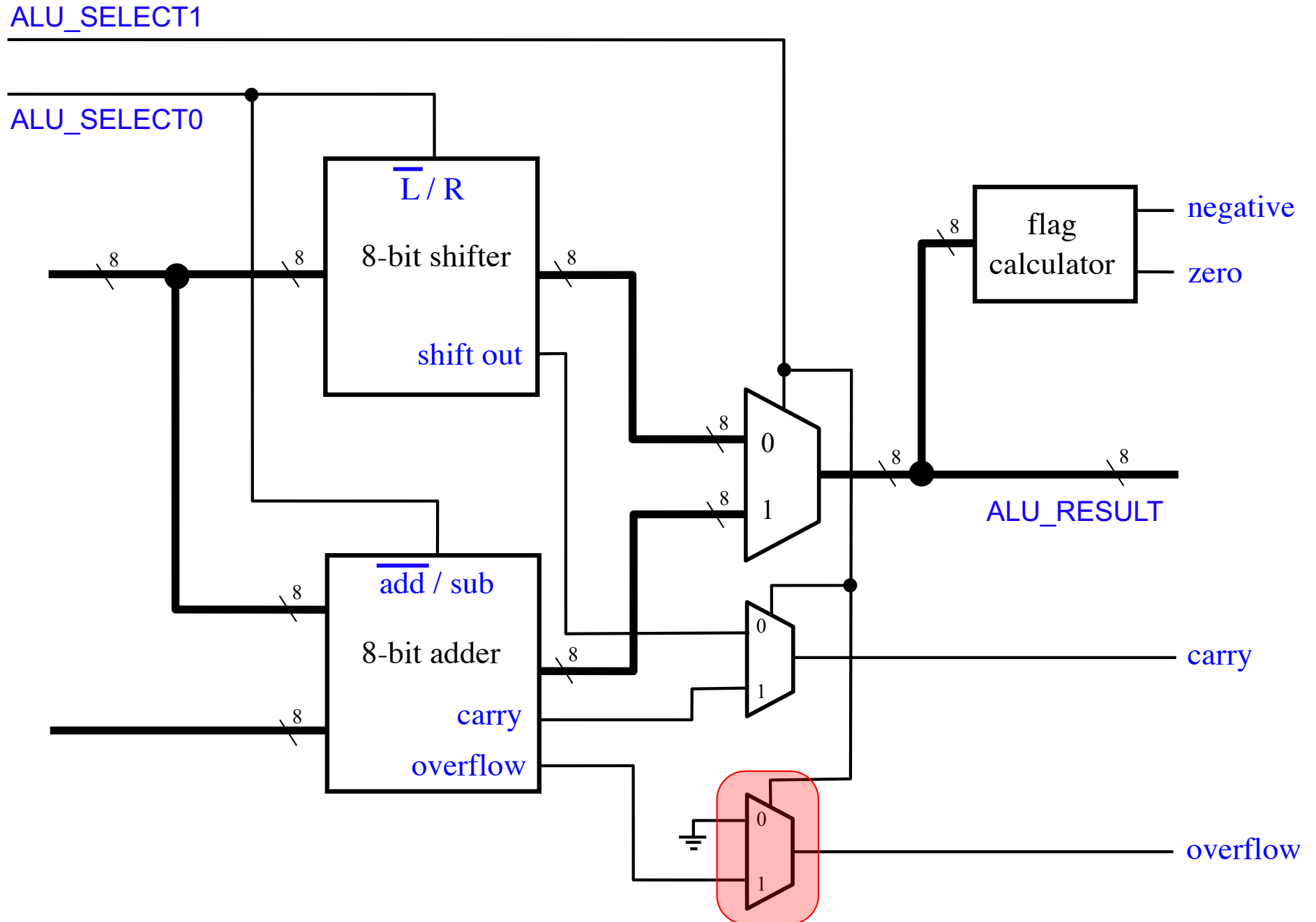




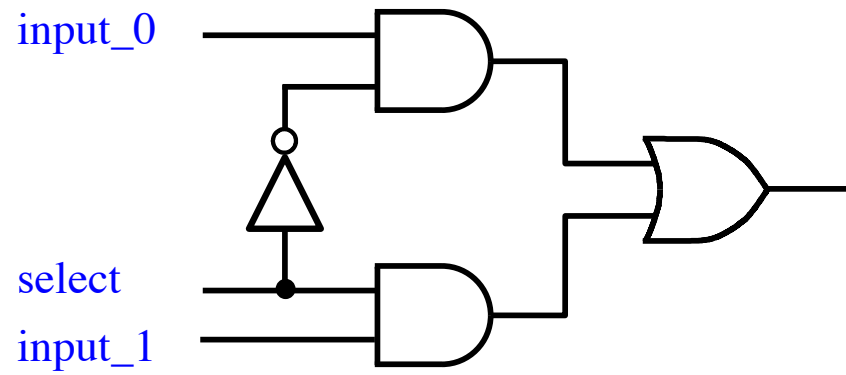
# 2-to-1 Multiplexer



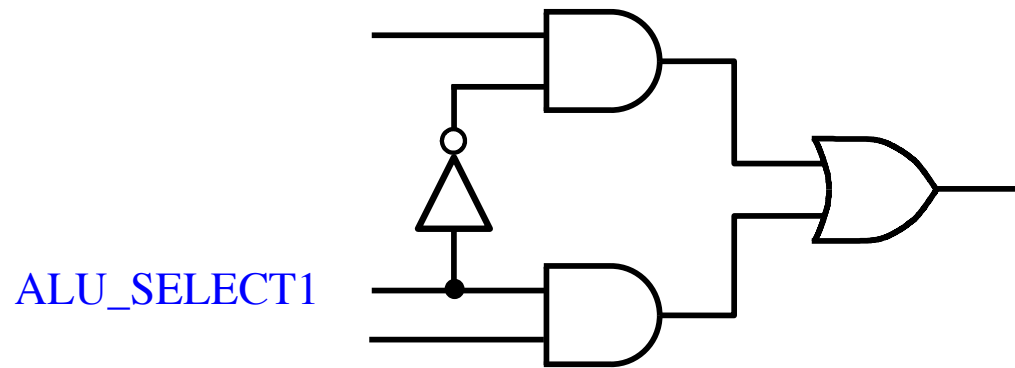
# 2-to-1 Multiplexer



# 2-to-1 Multiplexer

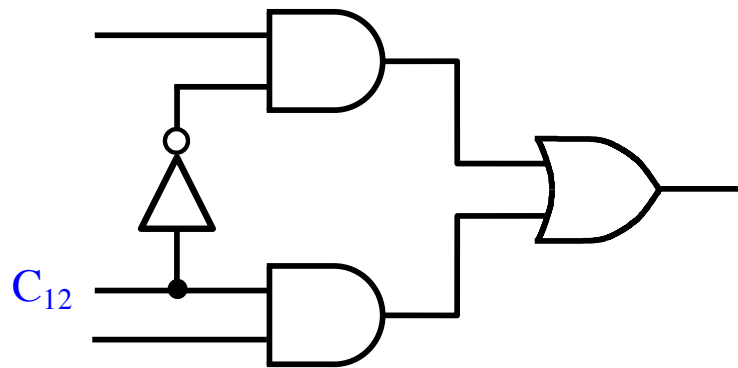


# 2-to-1 Multiplexer

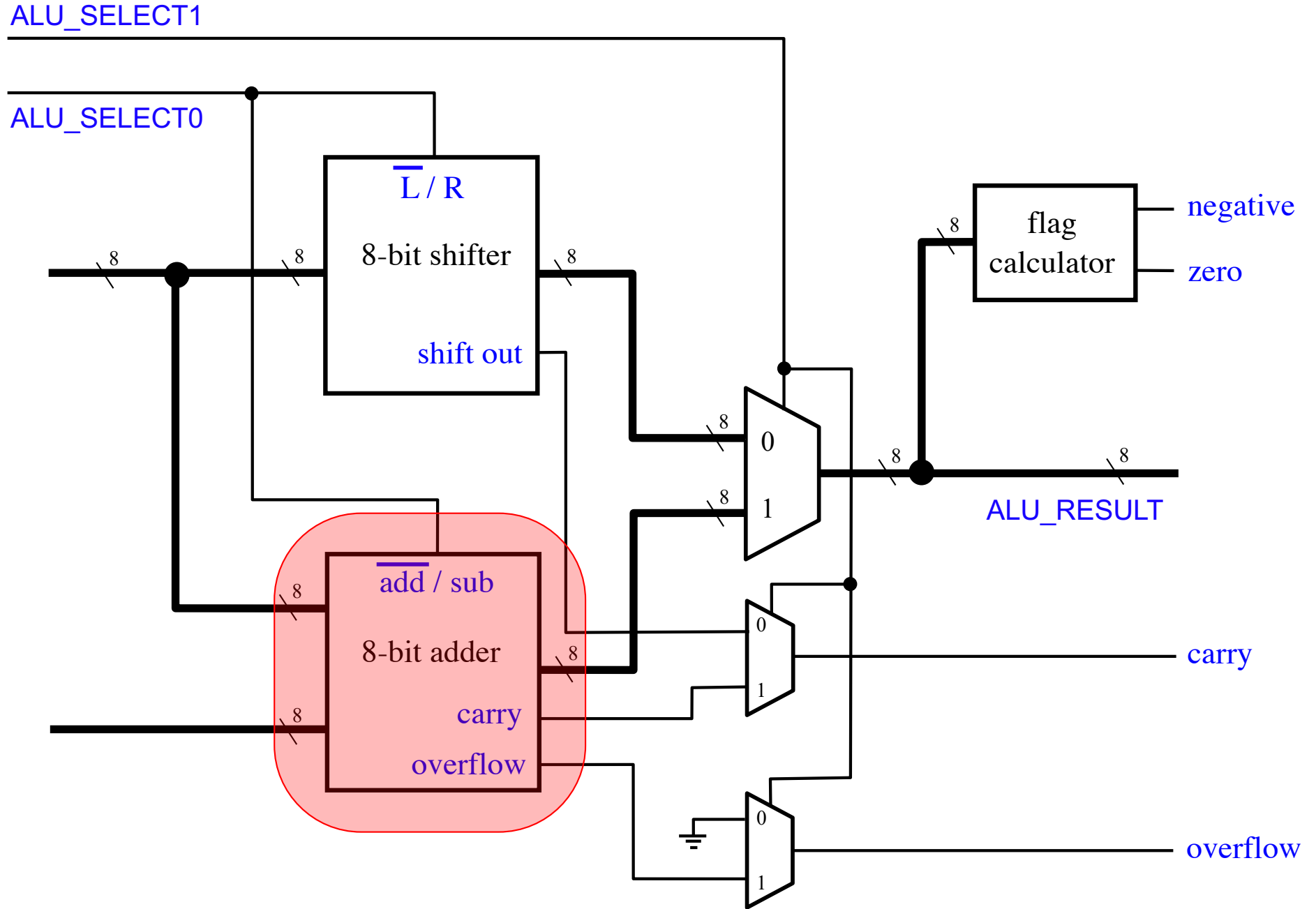




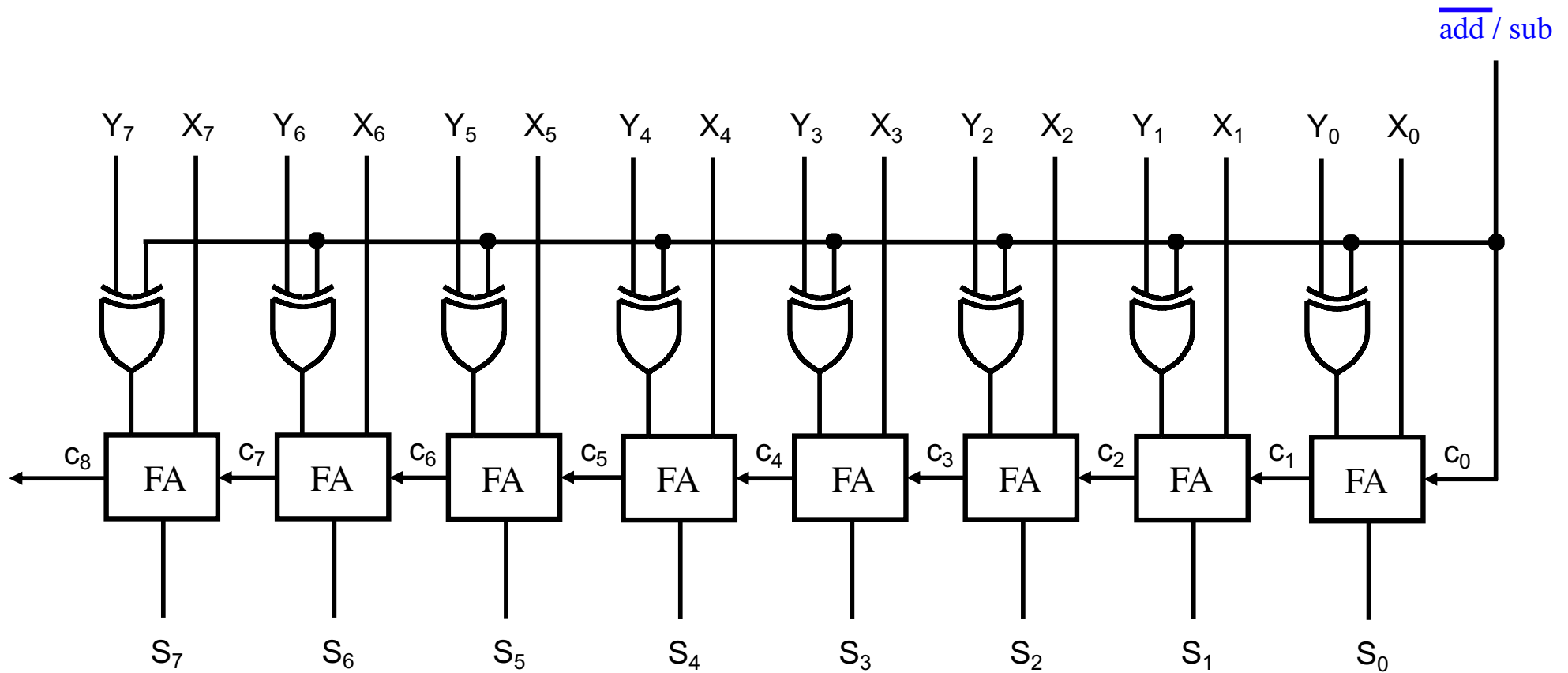
# 2-to-1 Multiplexer



# The Adder / Subtractor



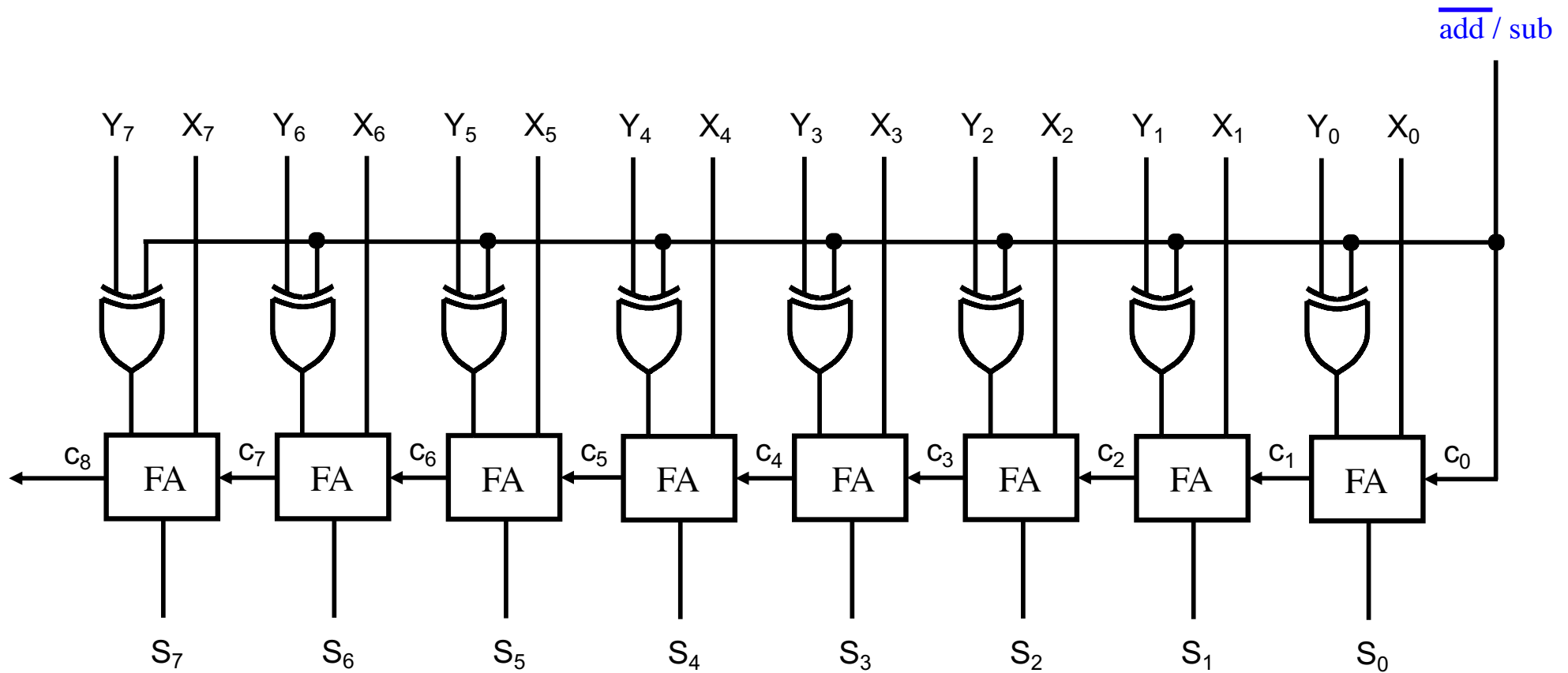
# The Adder / Subtractor





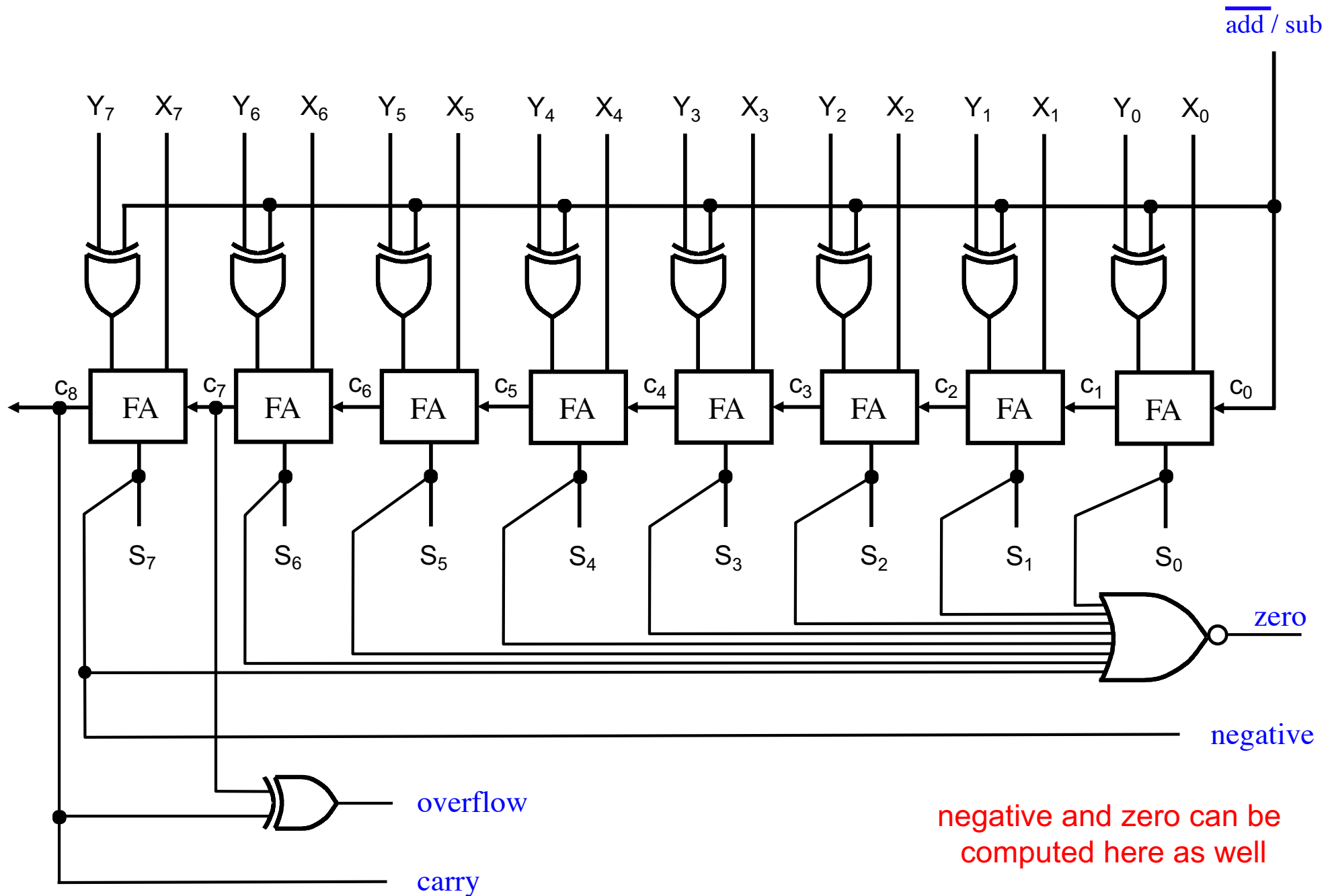


# The Adder / Subtractor



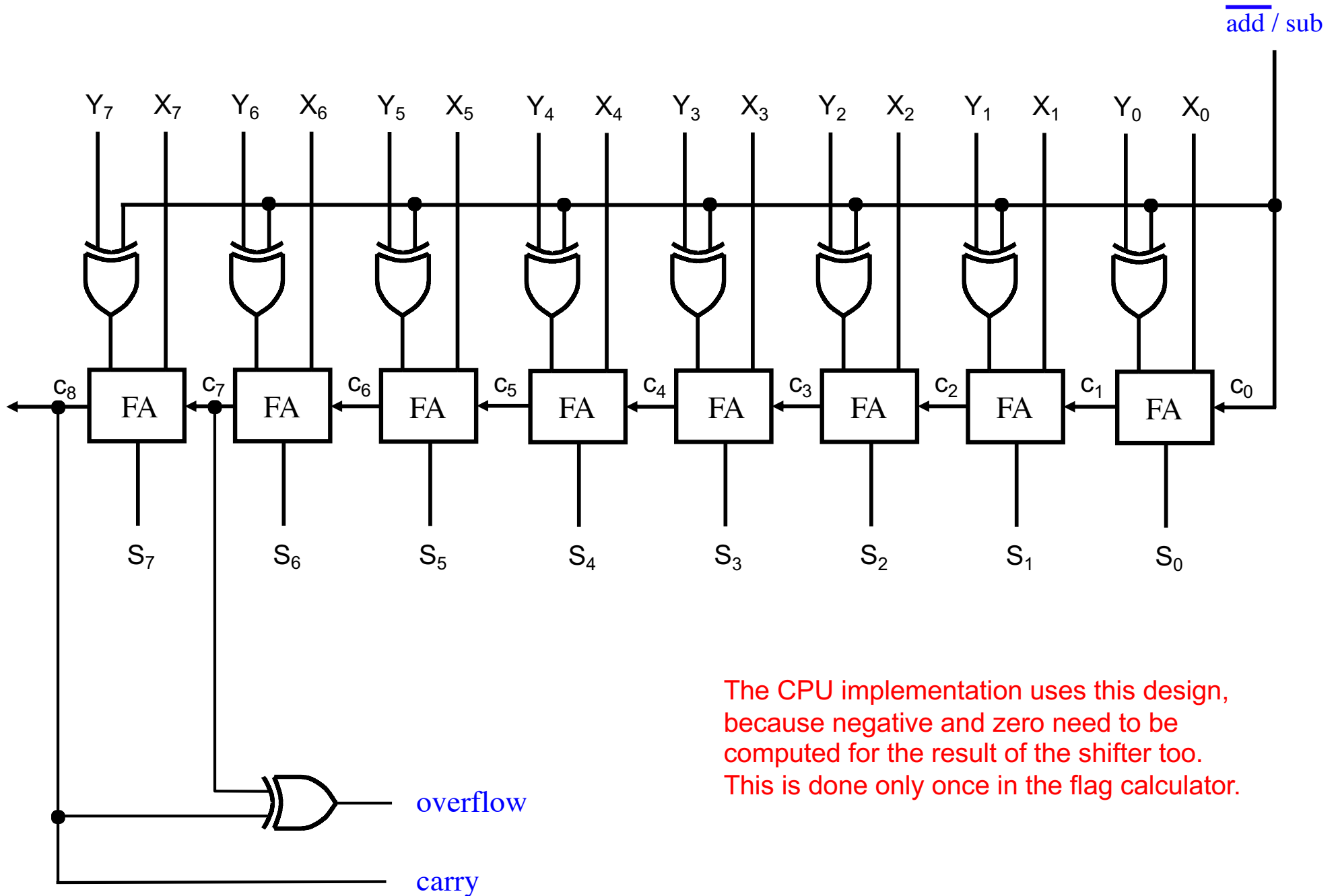


# The Adder / Subtractor

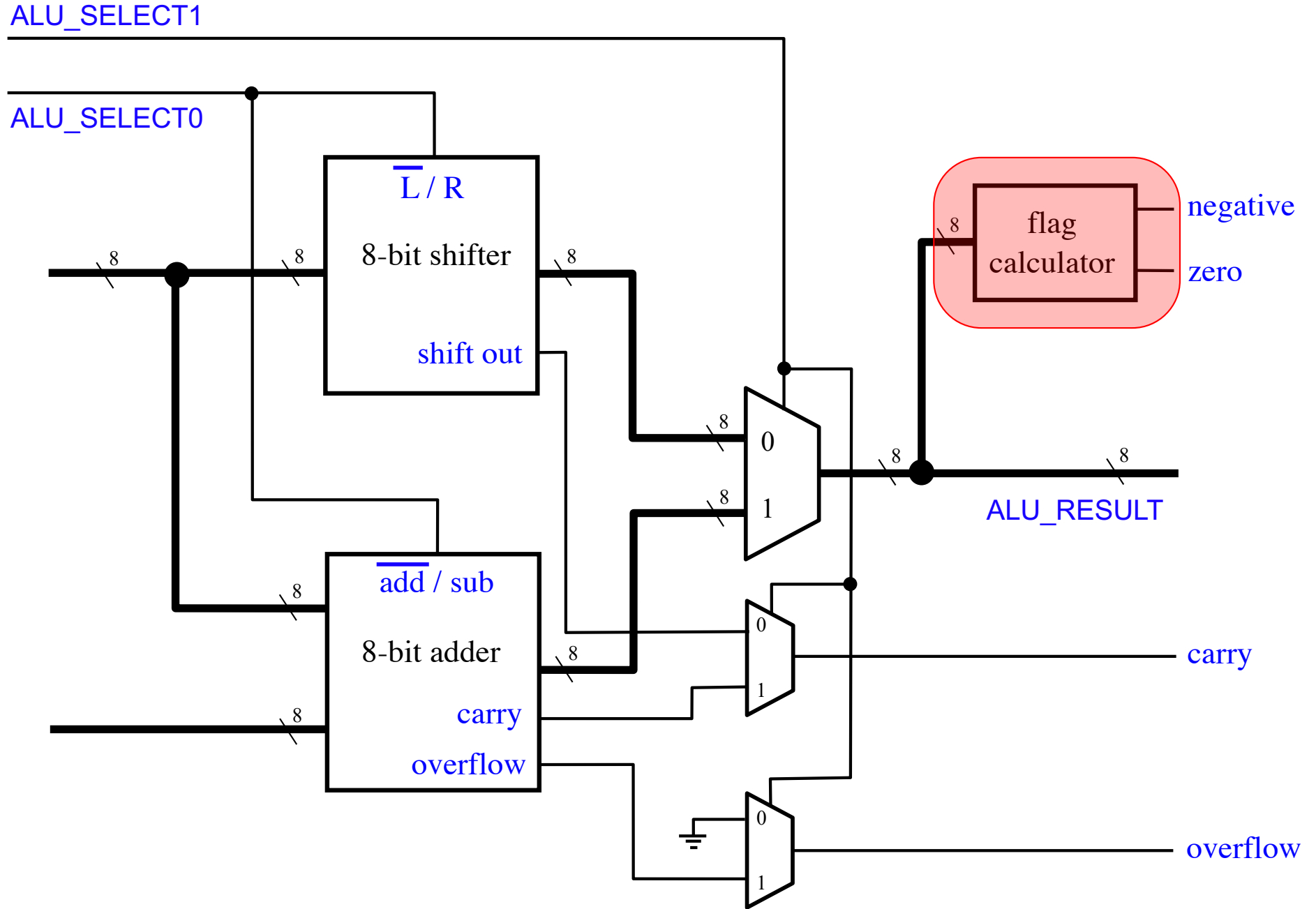




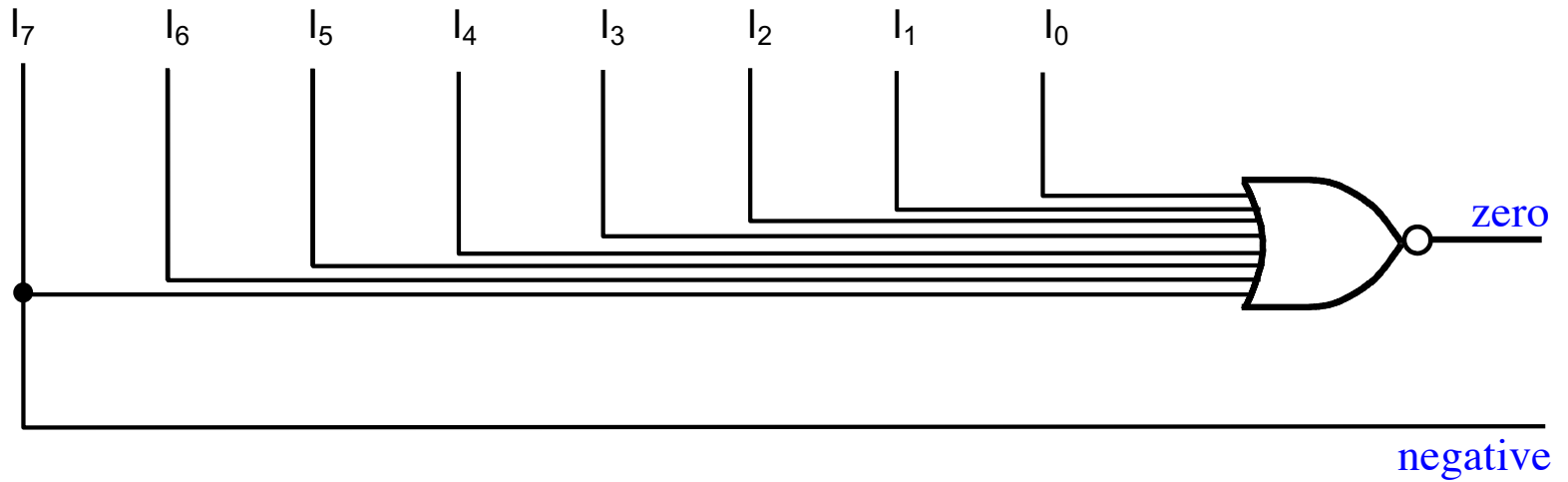
# The Adder / Subtractor



# The ALU Flag Calculator



# The ALU Flag Calculator

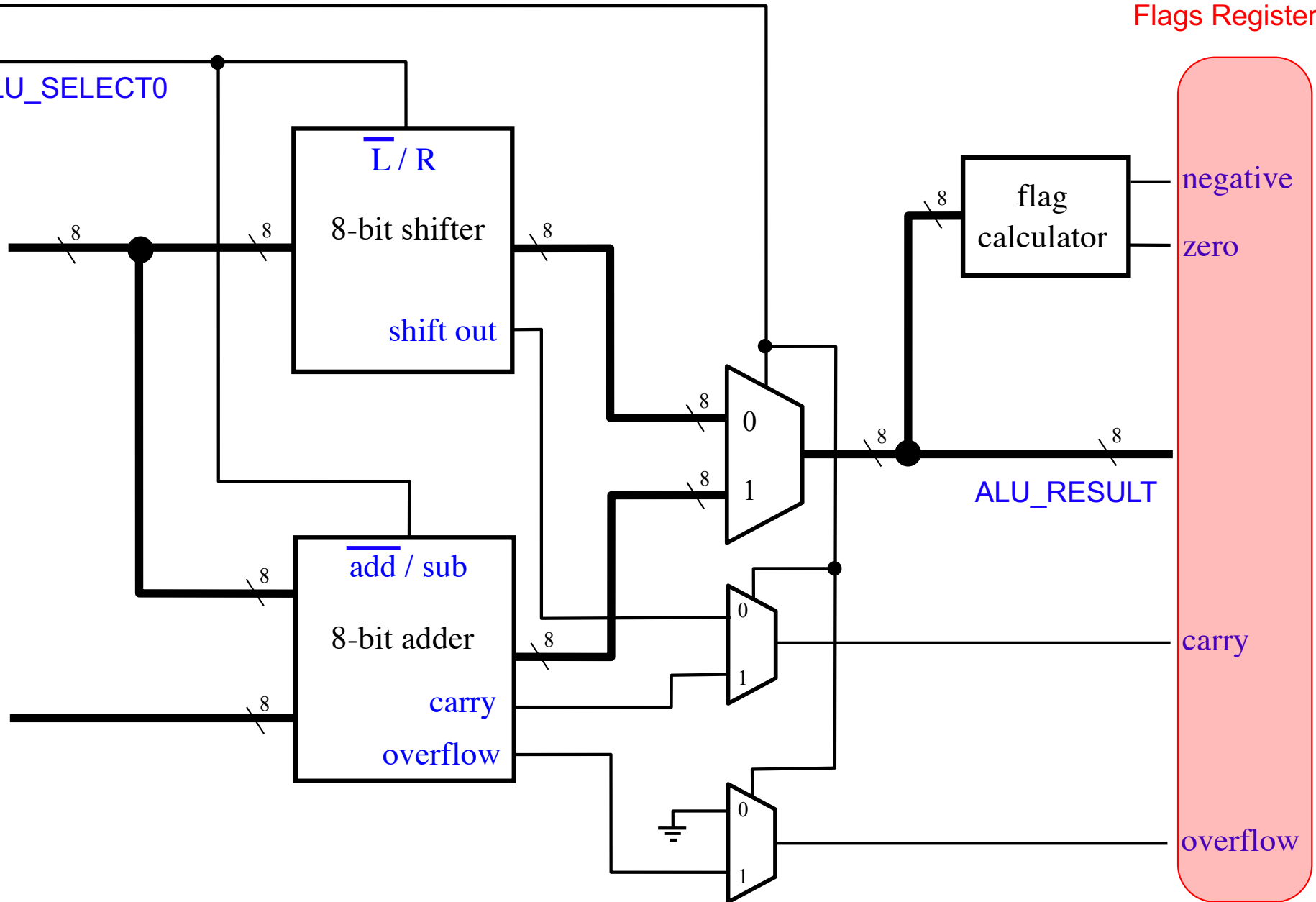


# ALU Outputs to the Flags Register

ALU\_SELECT1

ALU\_SELECT0

4 Outputs to the  
Flags Register



negative

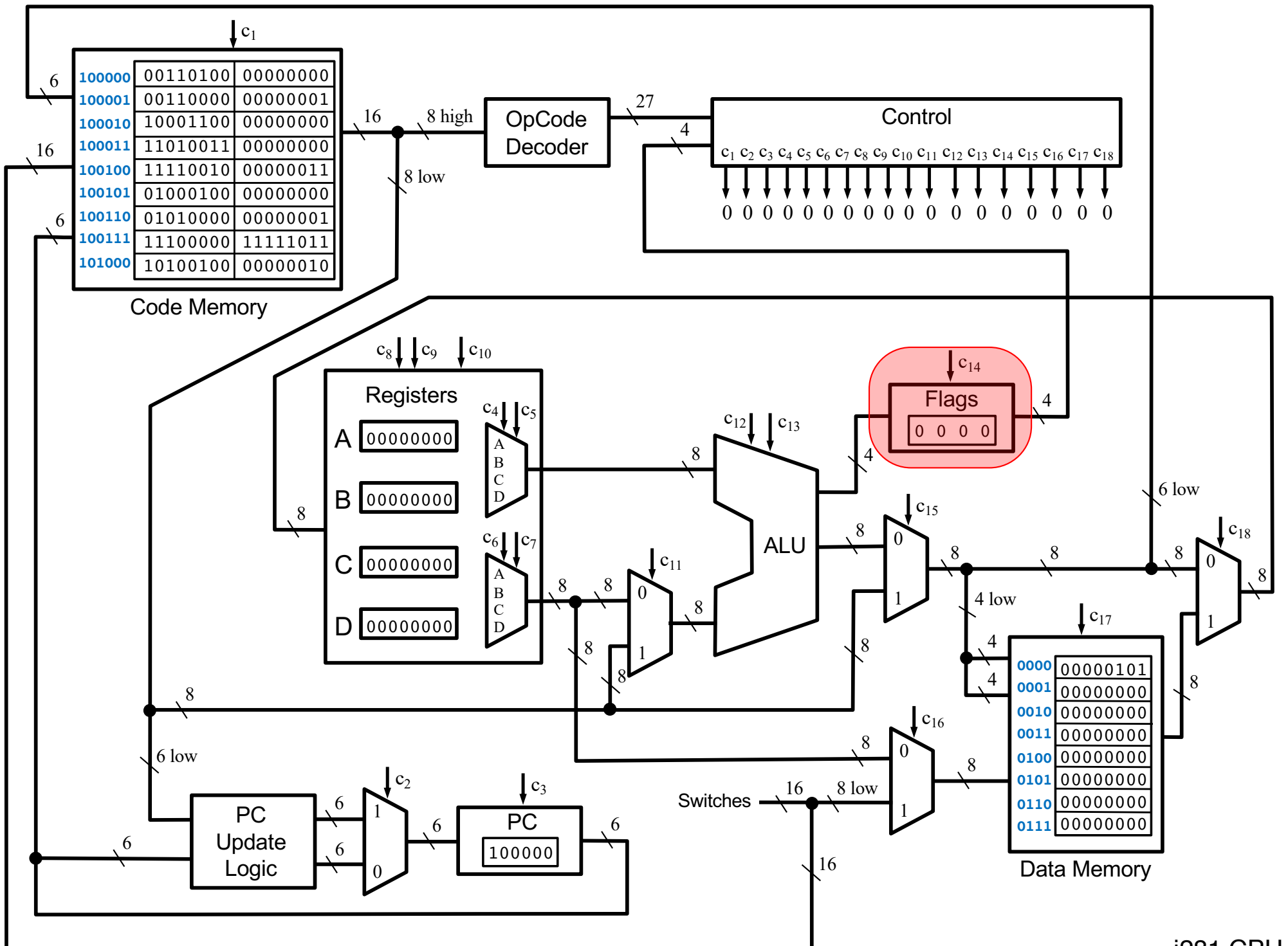
zero

ALU\_RESULT

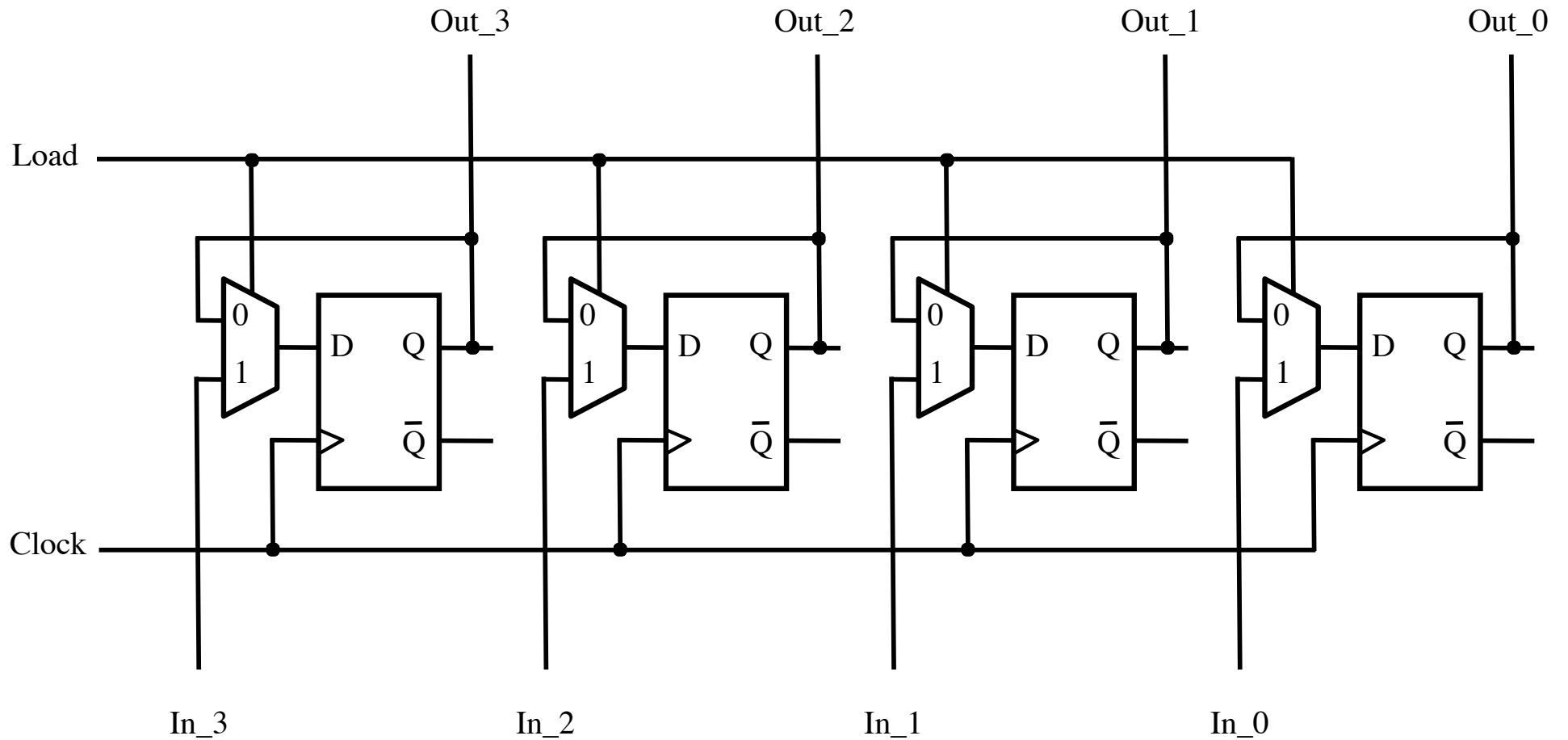
carry

overflow

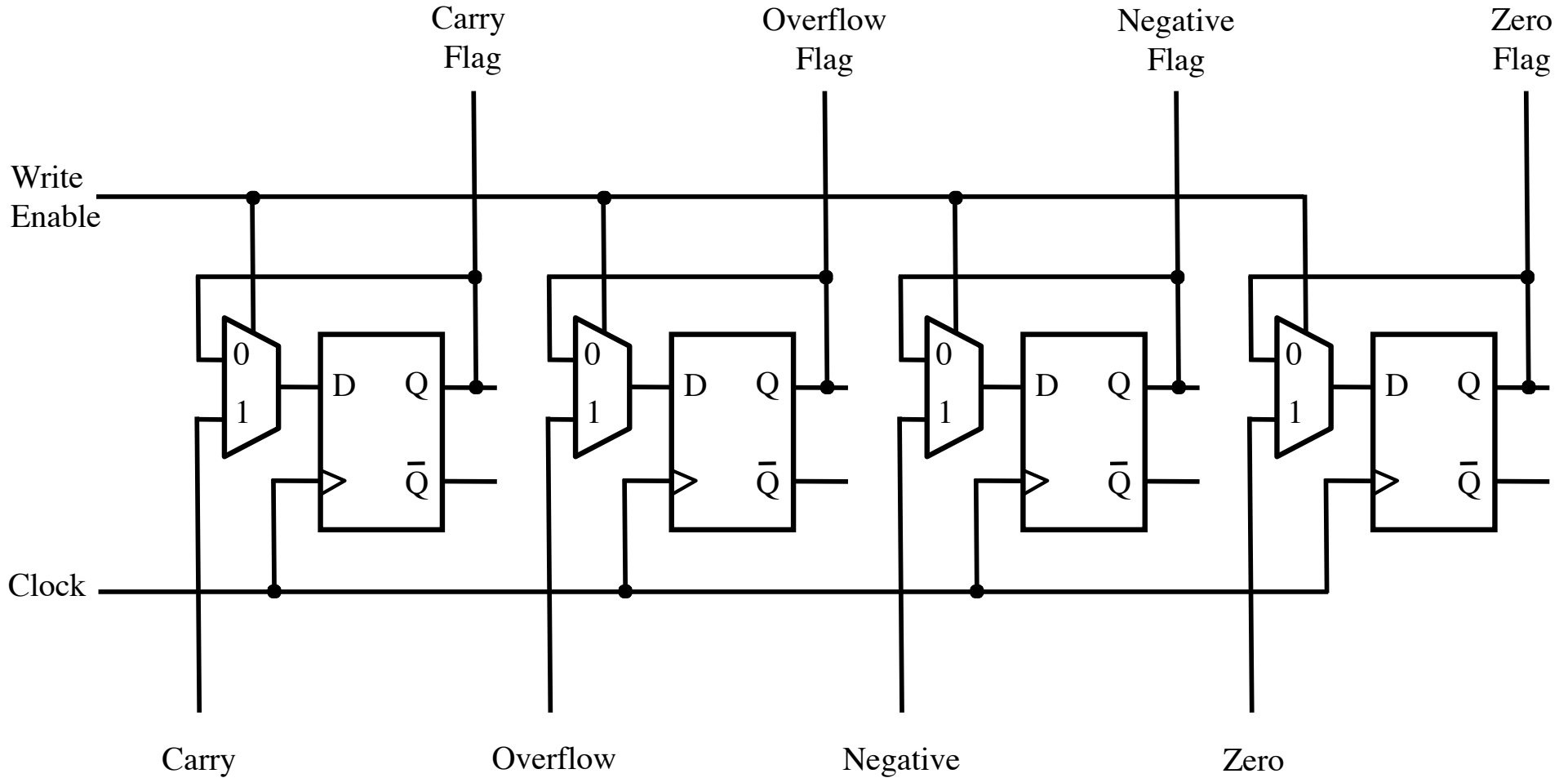
# **The Flags Register**



# 4-Bit Parallel-Access Register

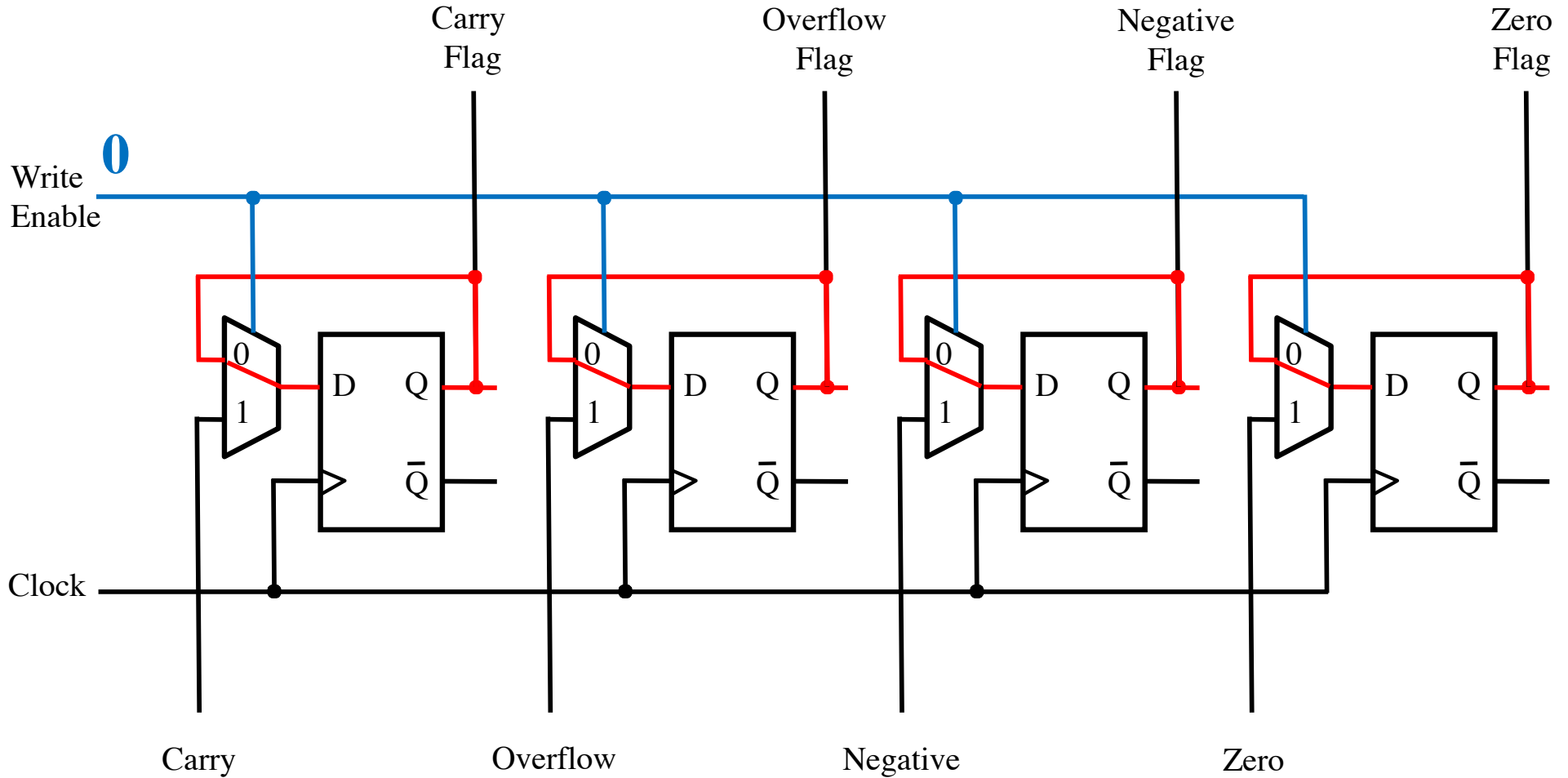


# The Flags Register

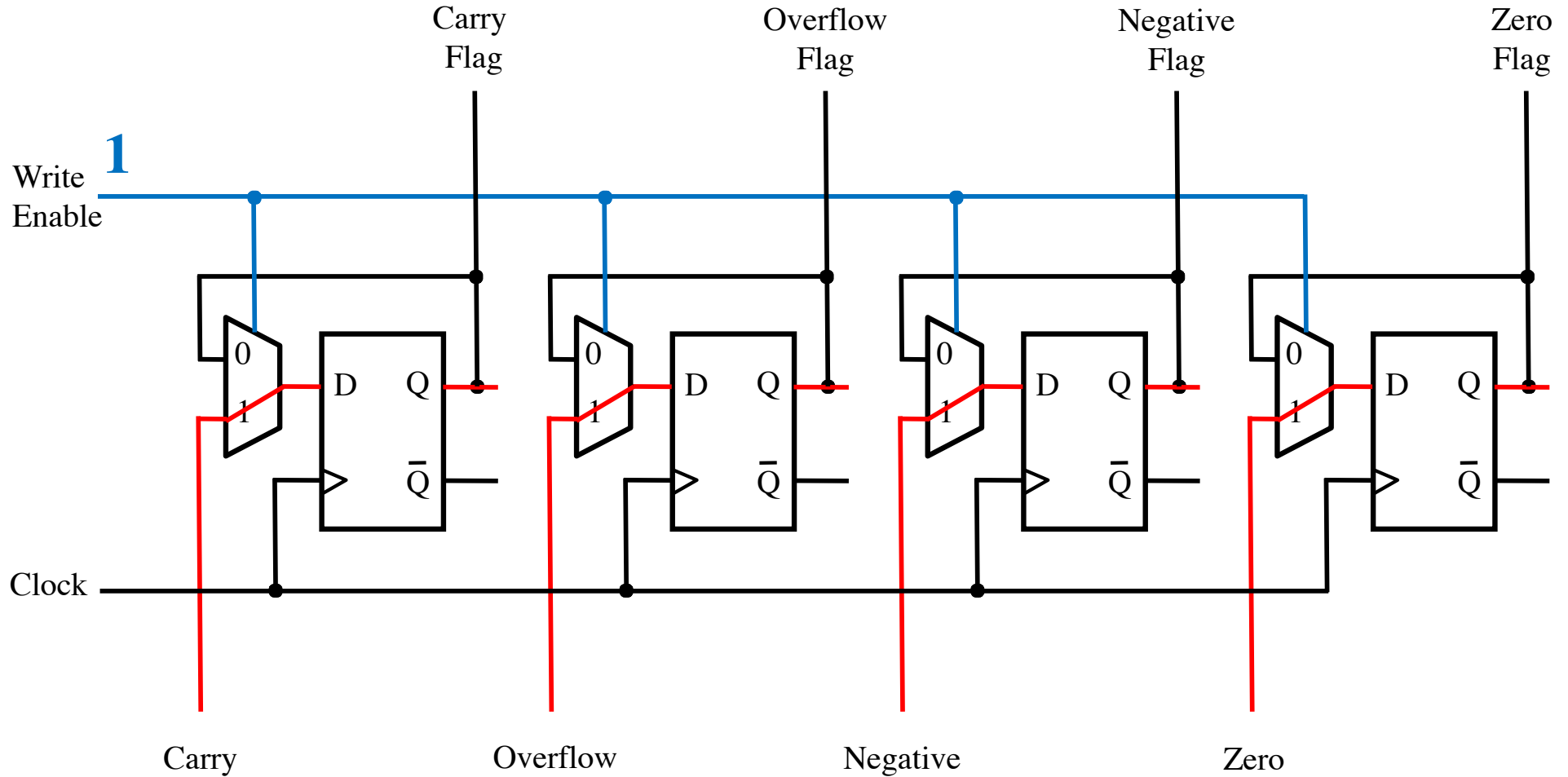




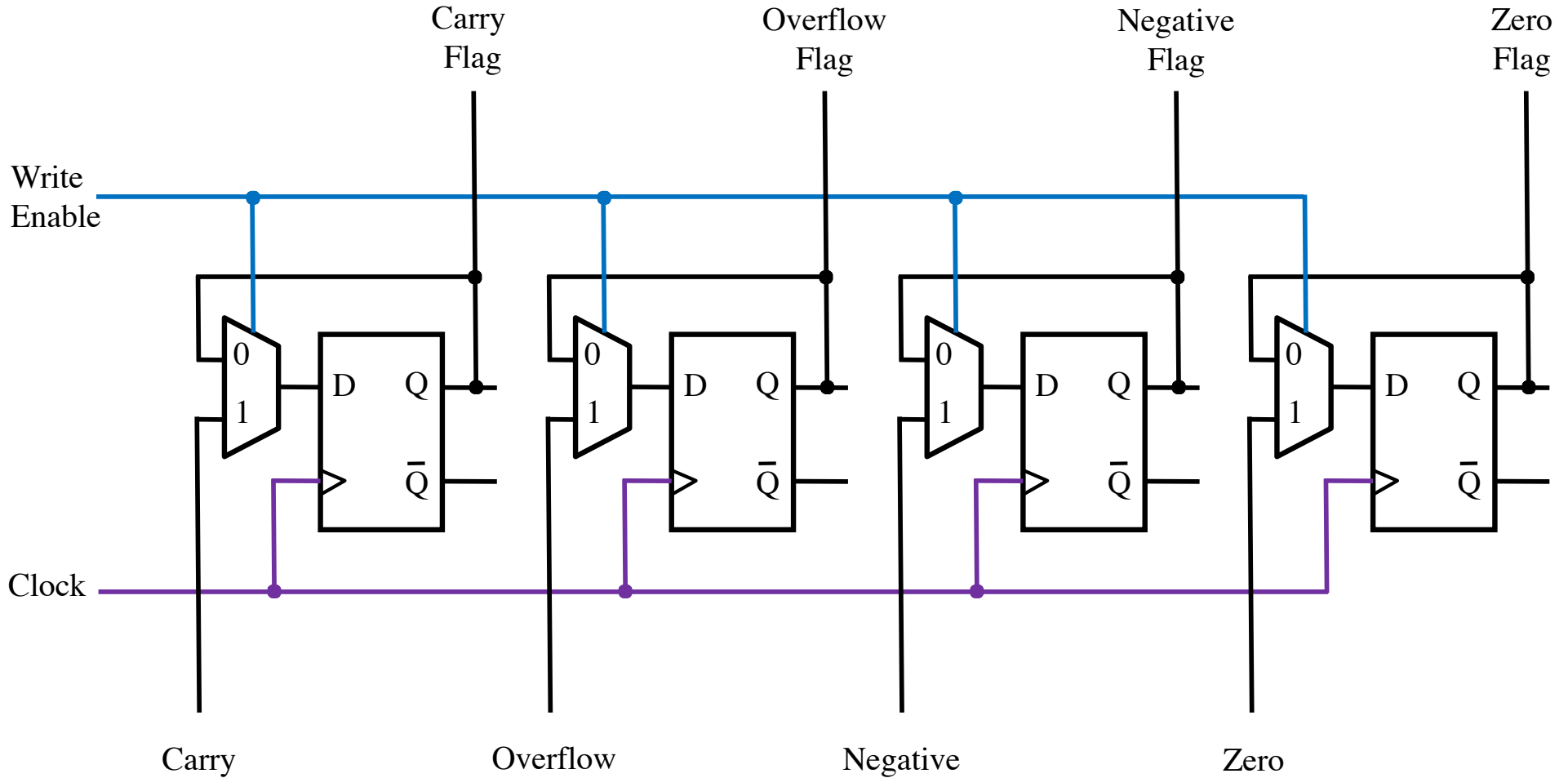
# The Flags Register



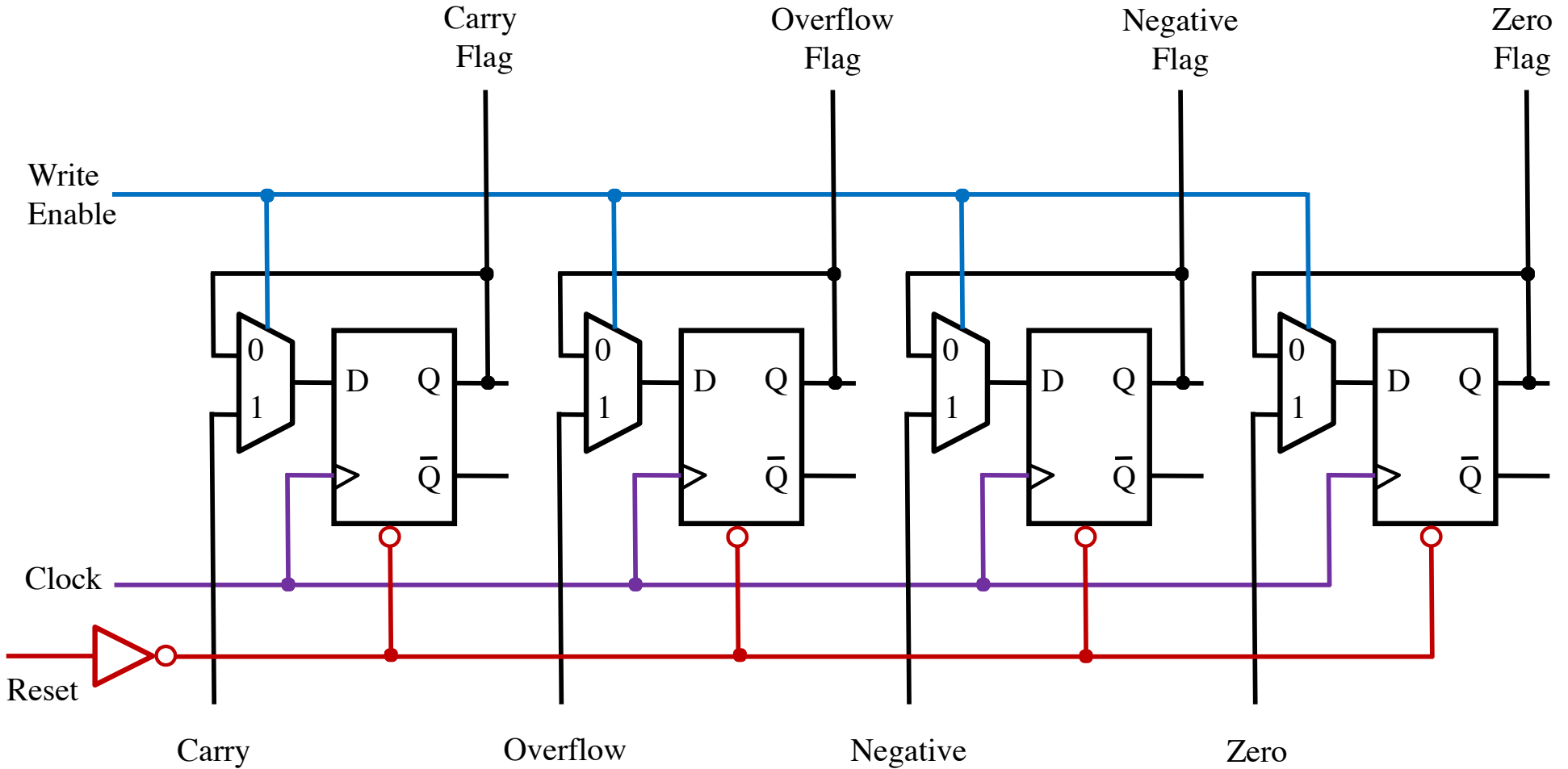
# The Flags Register



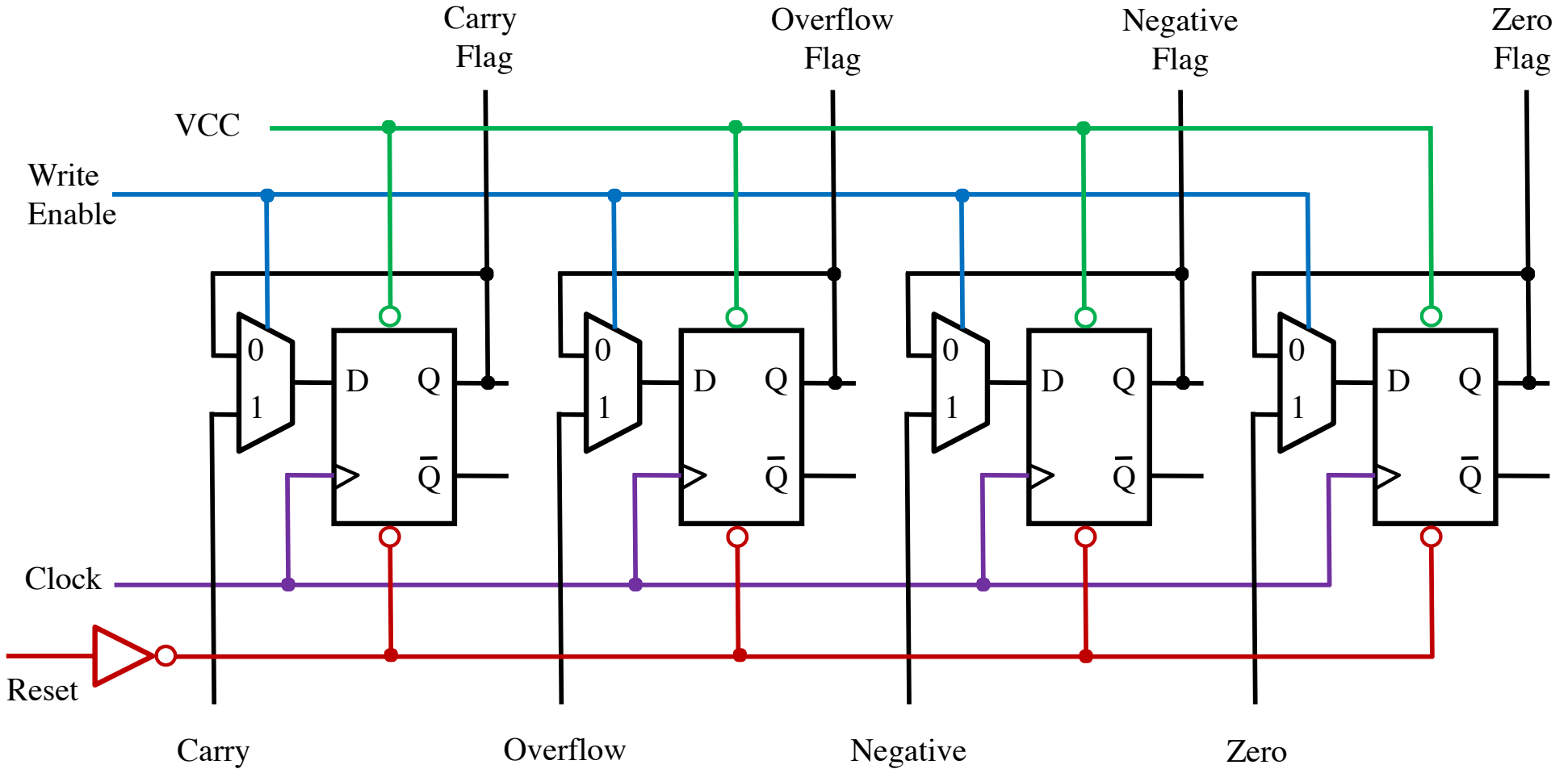
# The Flags Register



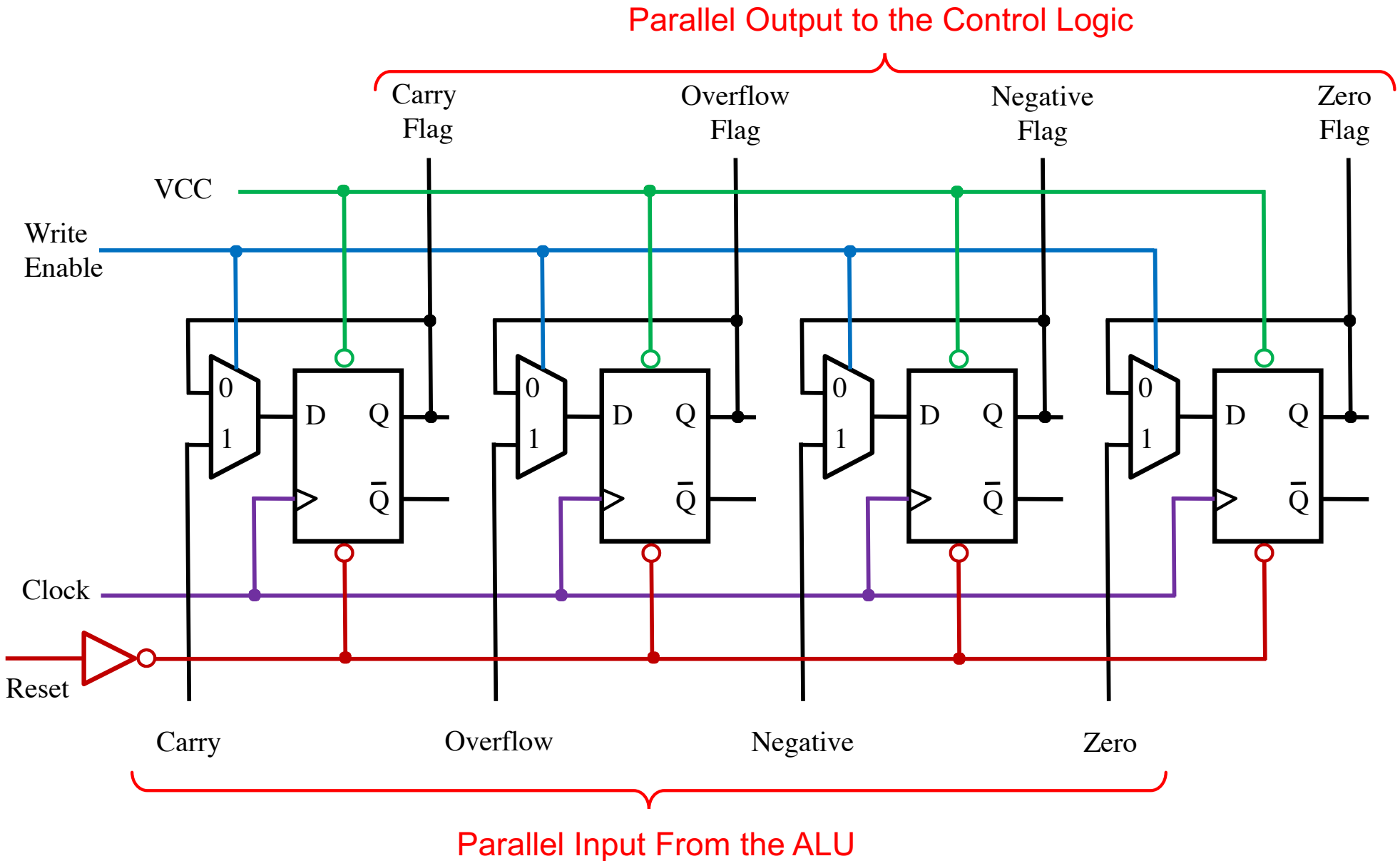
# The Flags Register



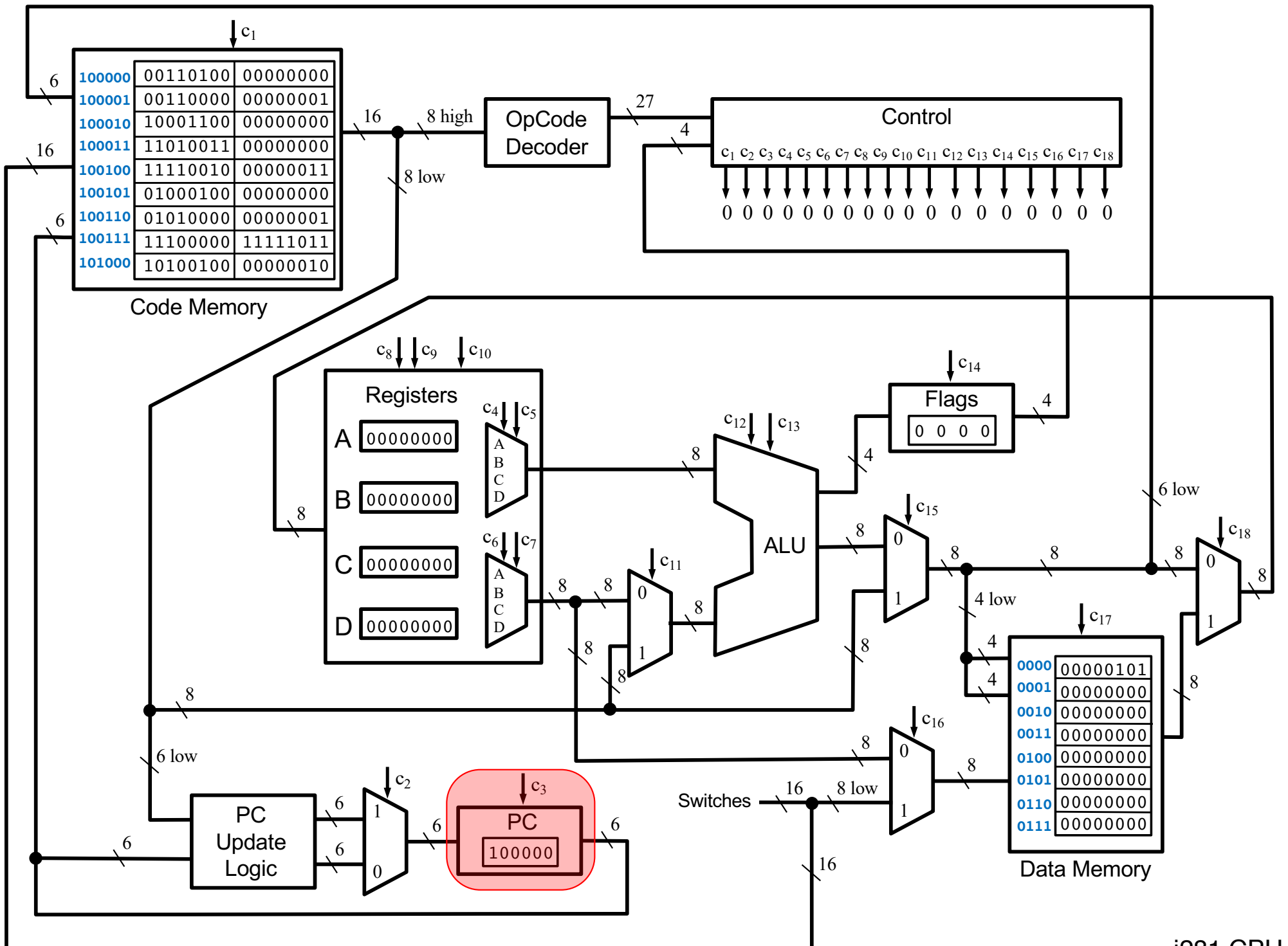
# The Flags Register



# The Flags Register

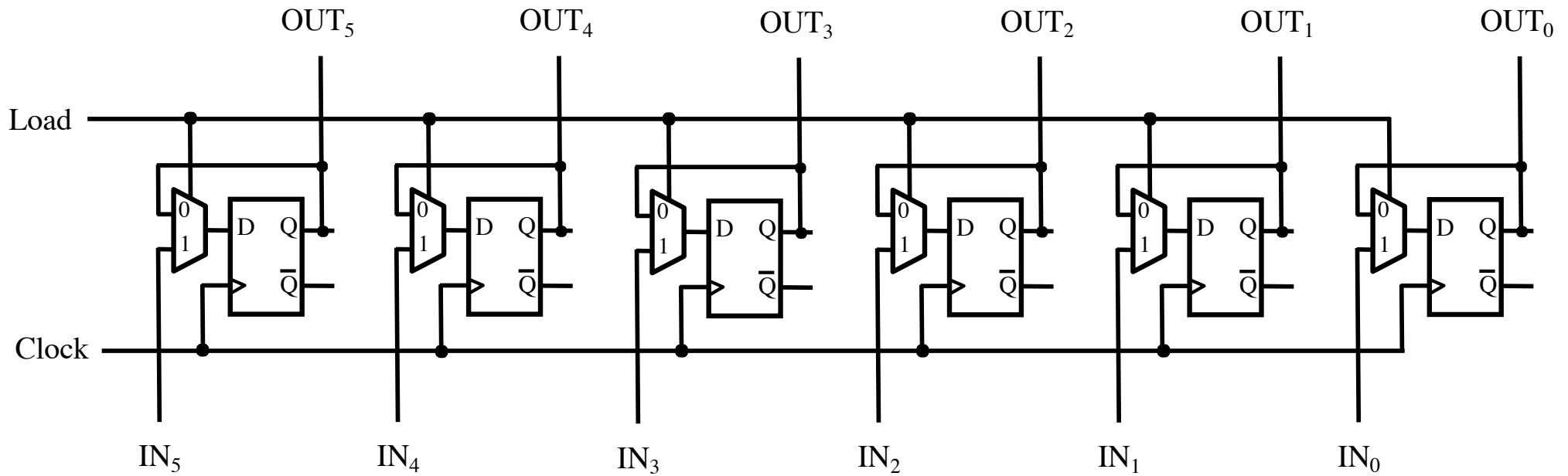


# **The Program Counter (PC)**

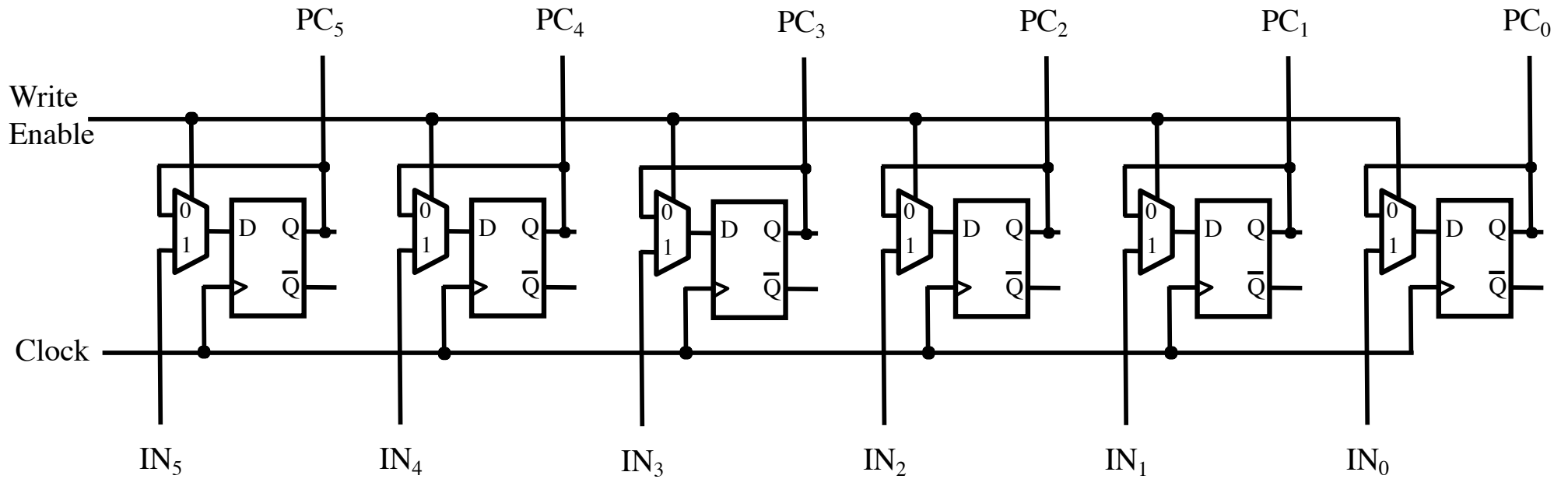




# 6-Bit Parallel-Access Register

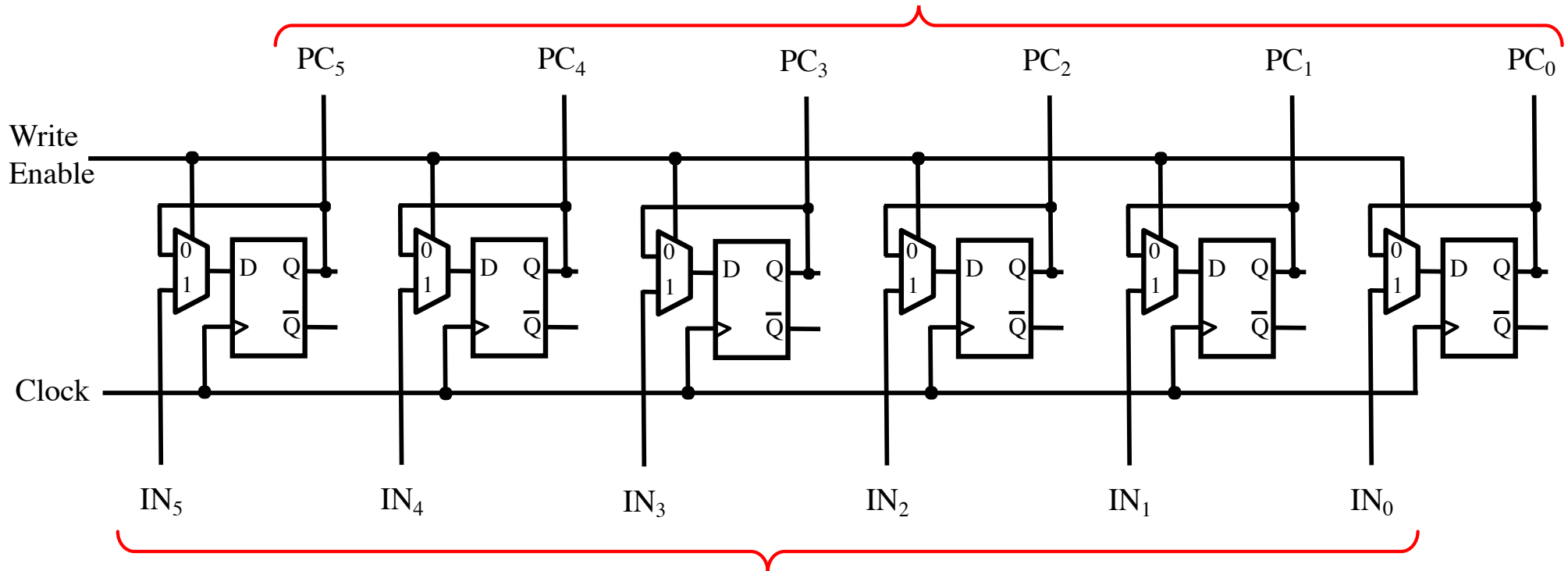


# The Program Counter Register



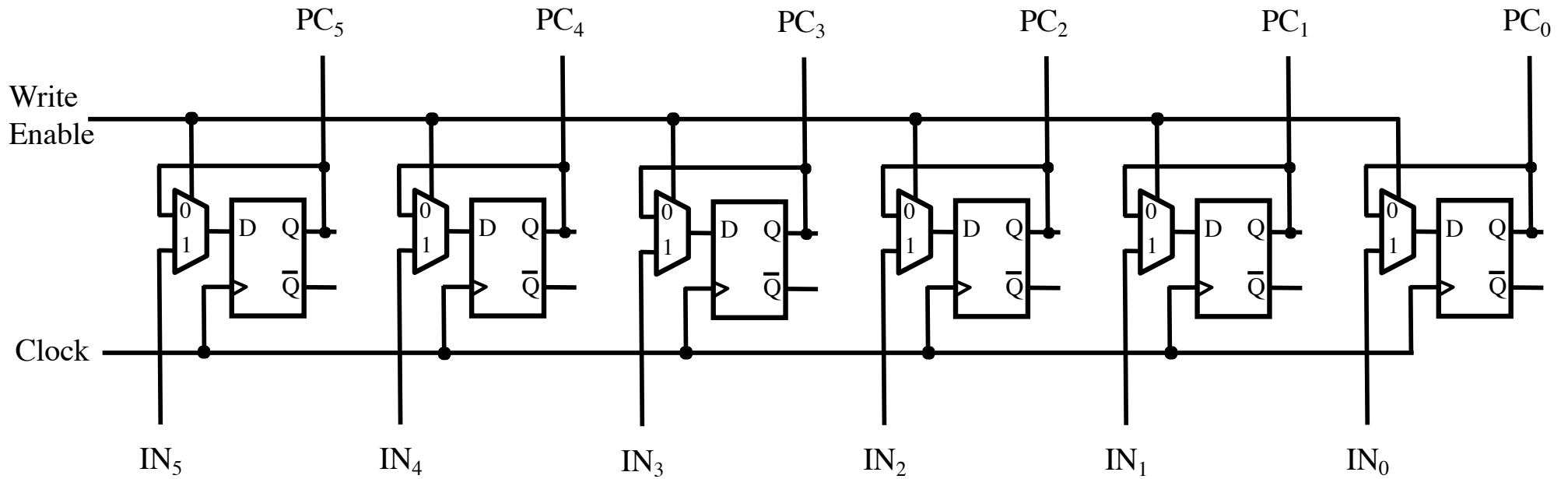
# The Program Counter Register

Parallel Output to the 6 Read Select Lines of the Code Memory

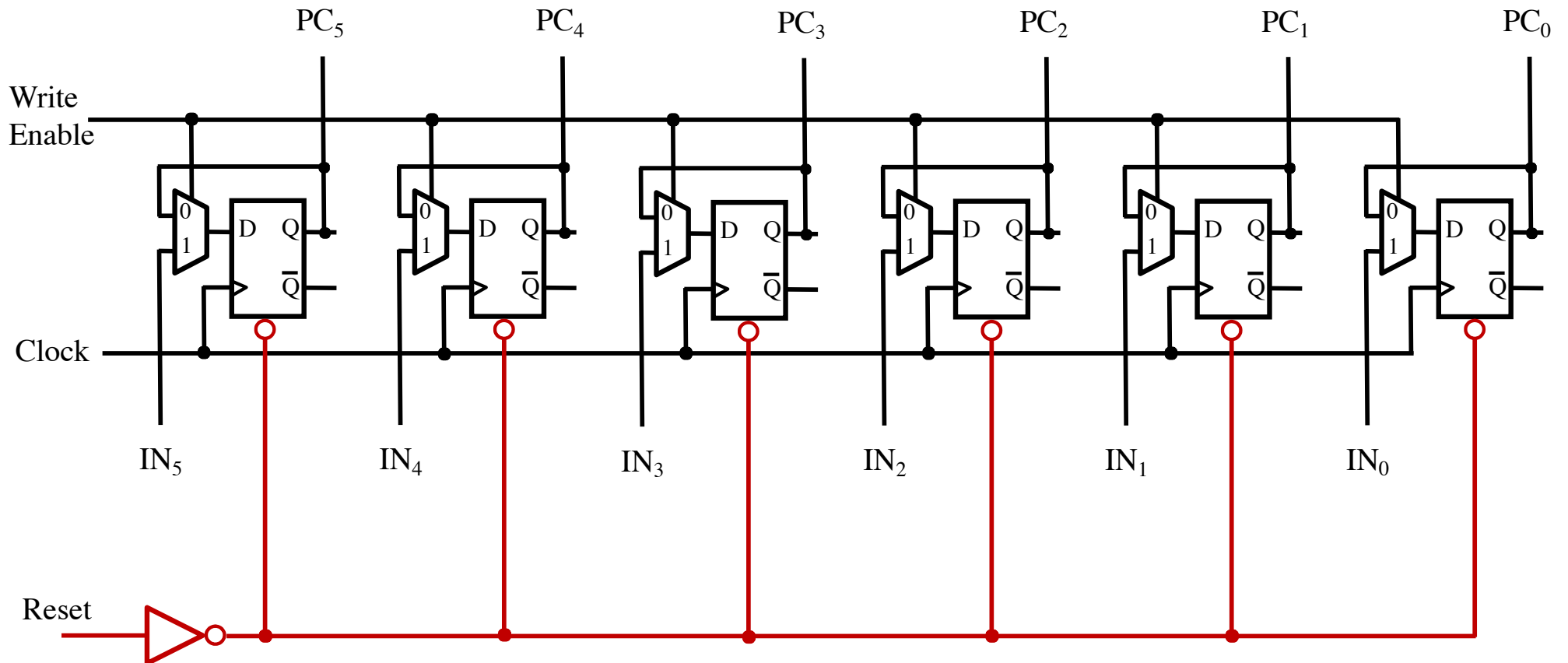


Parallel Input From the PC Update Logic

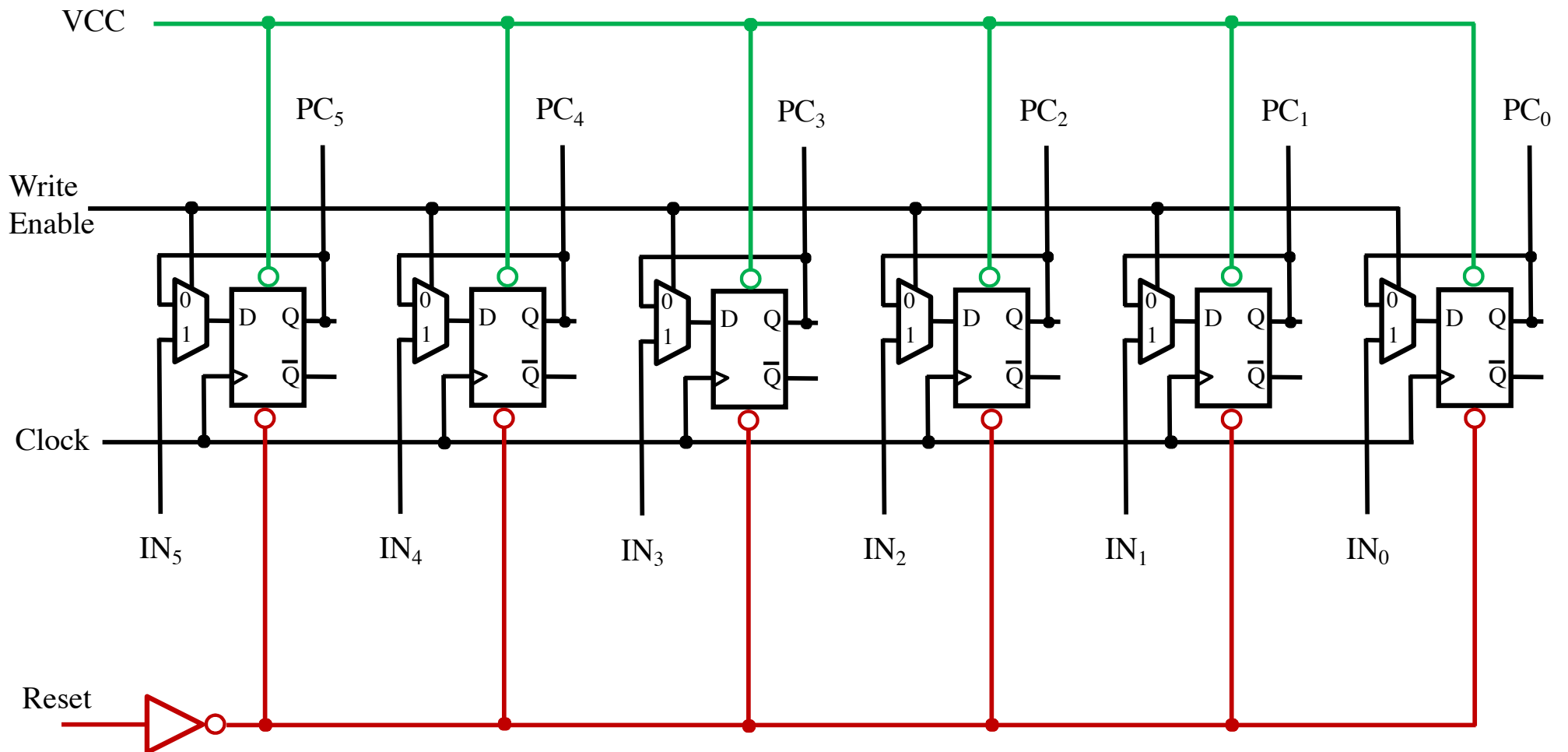
# The Program Counter Register



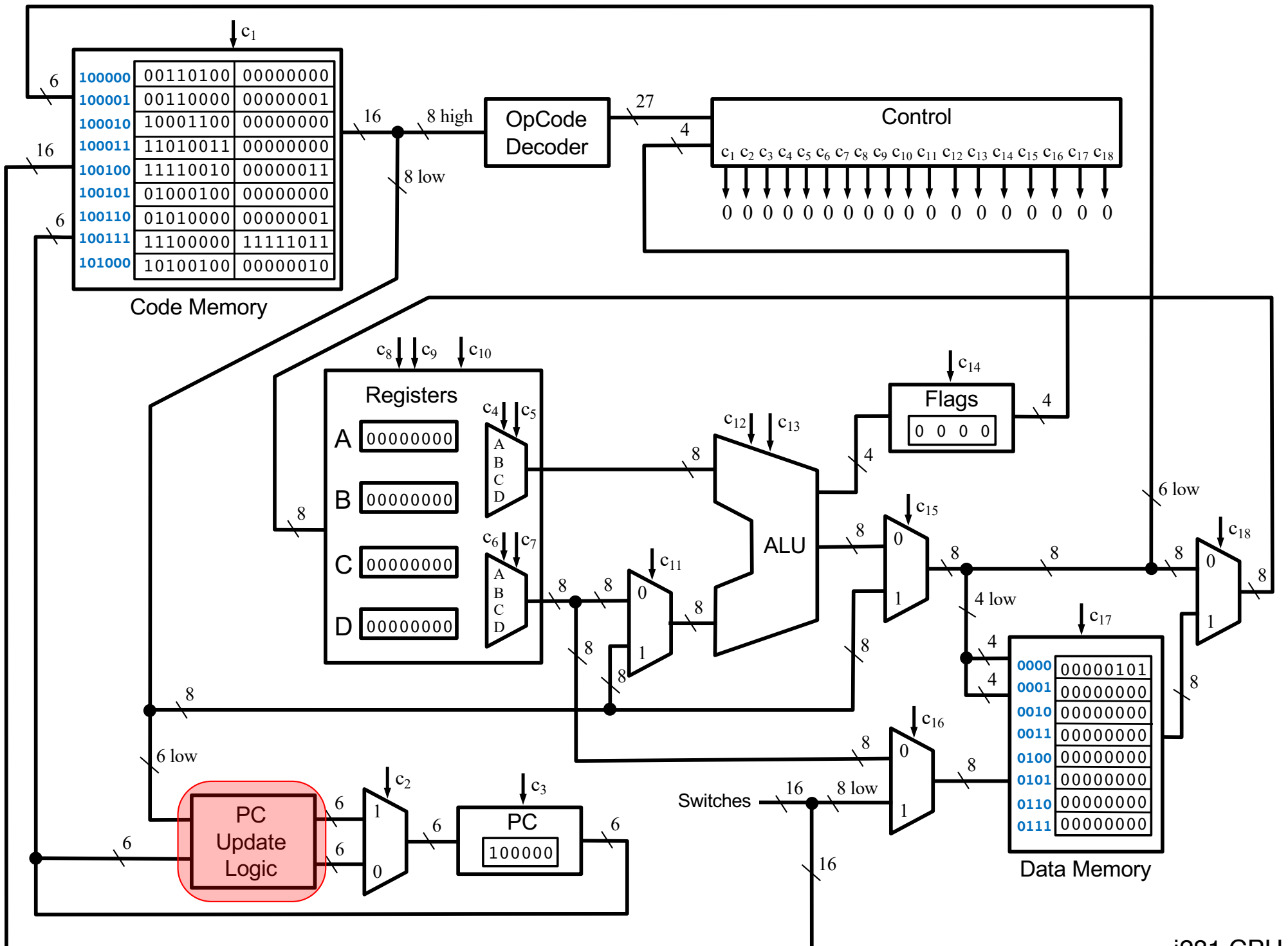
# The Program Counter Register



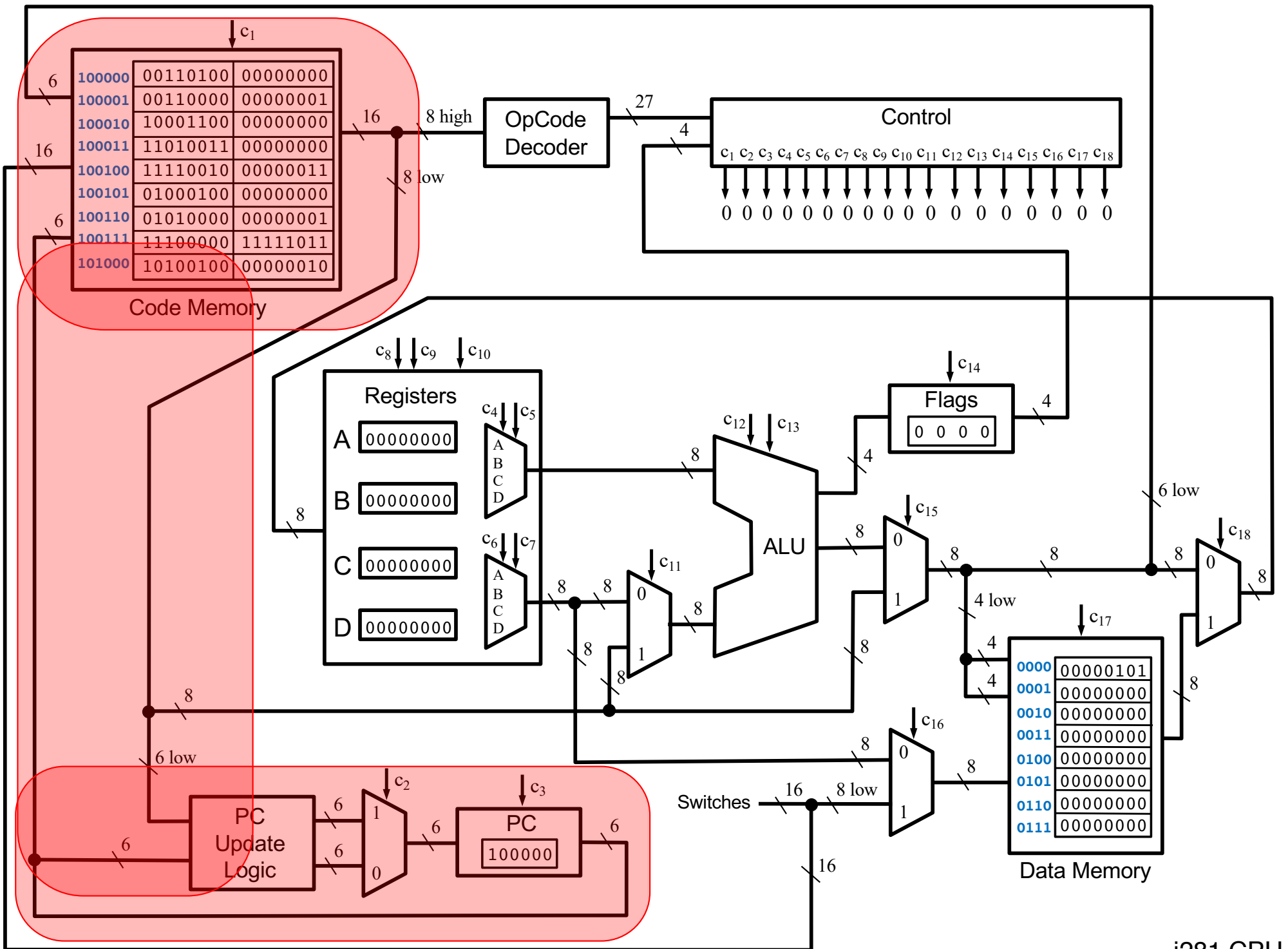
# The Program Counter Register

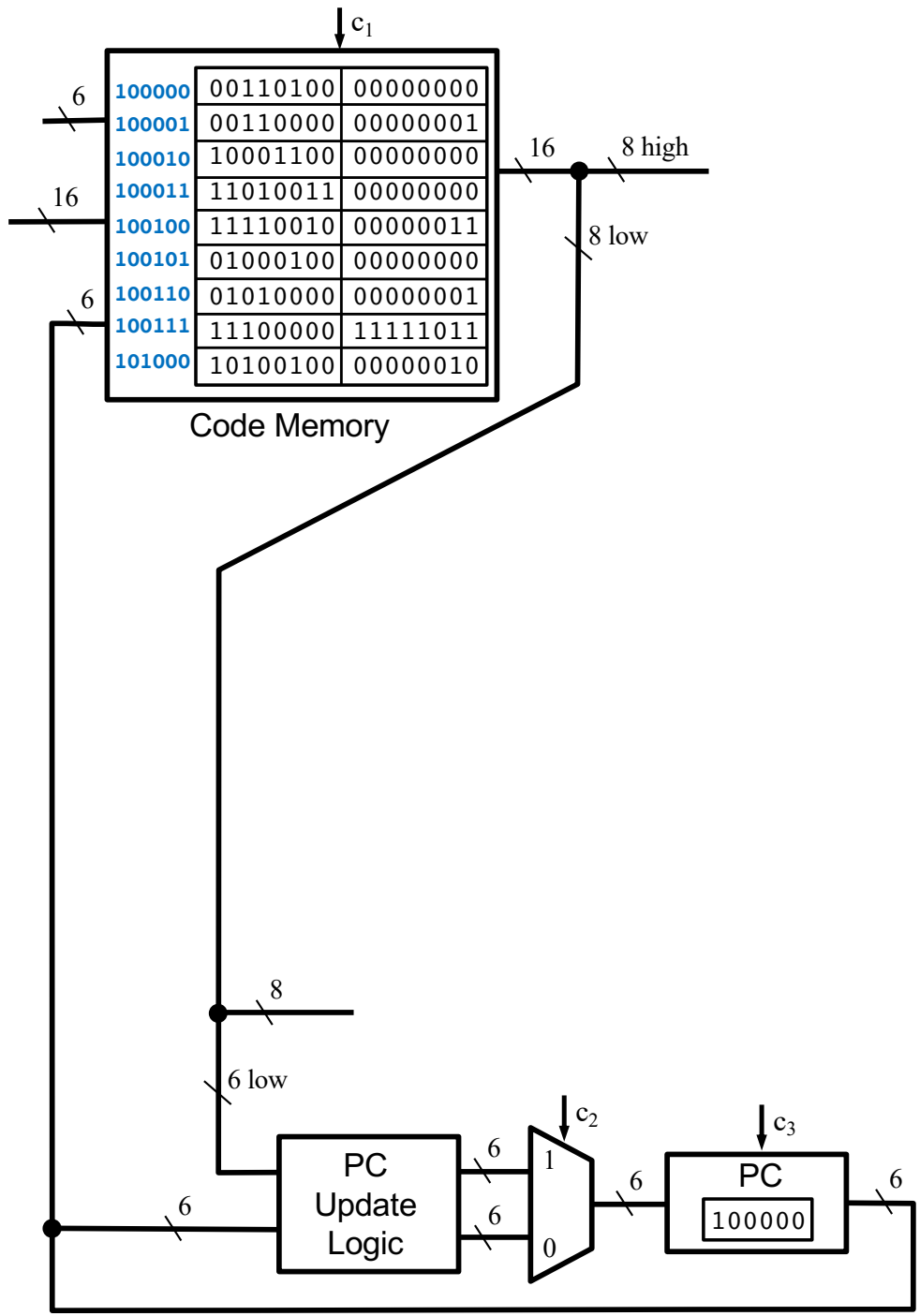


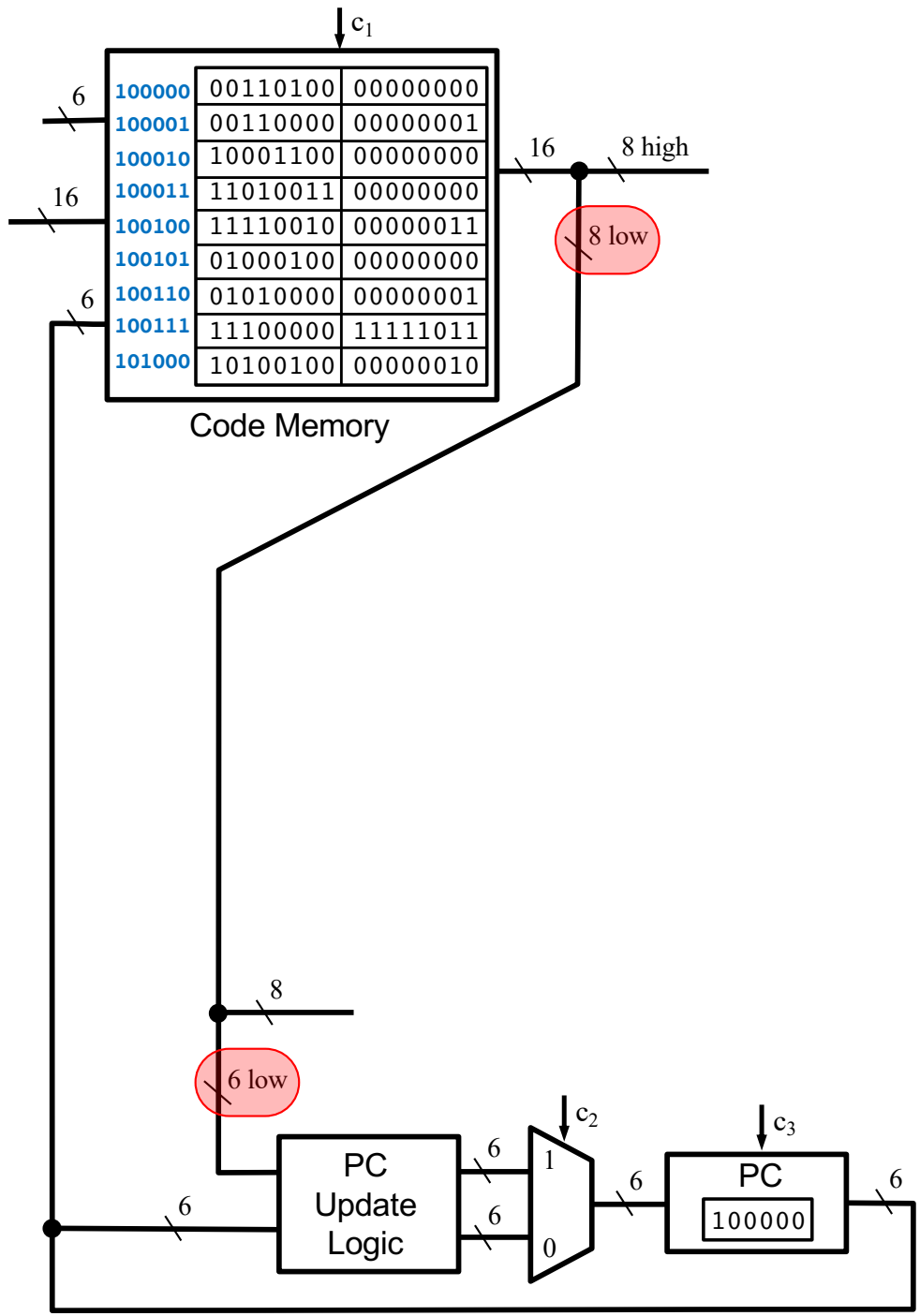
# **The Program Counter Update Logic**

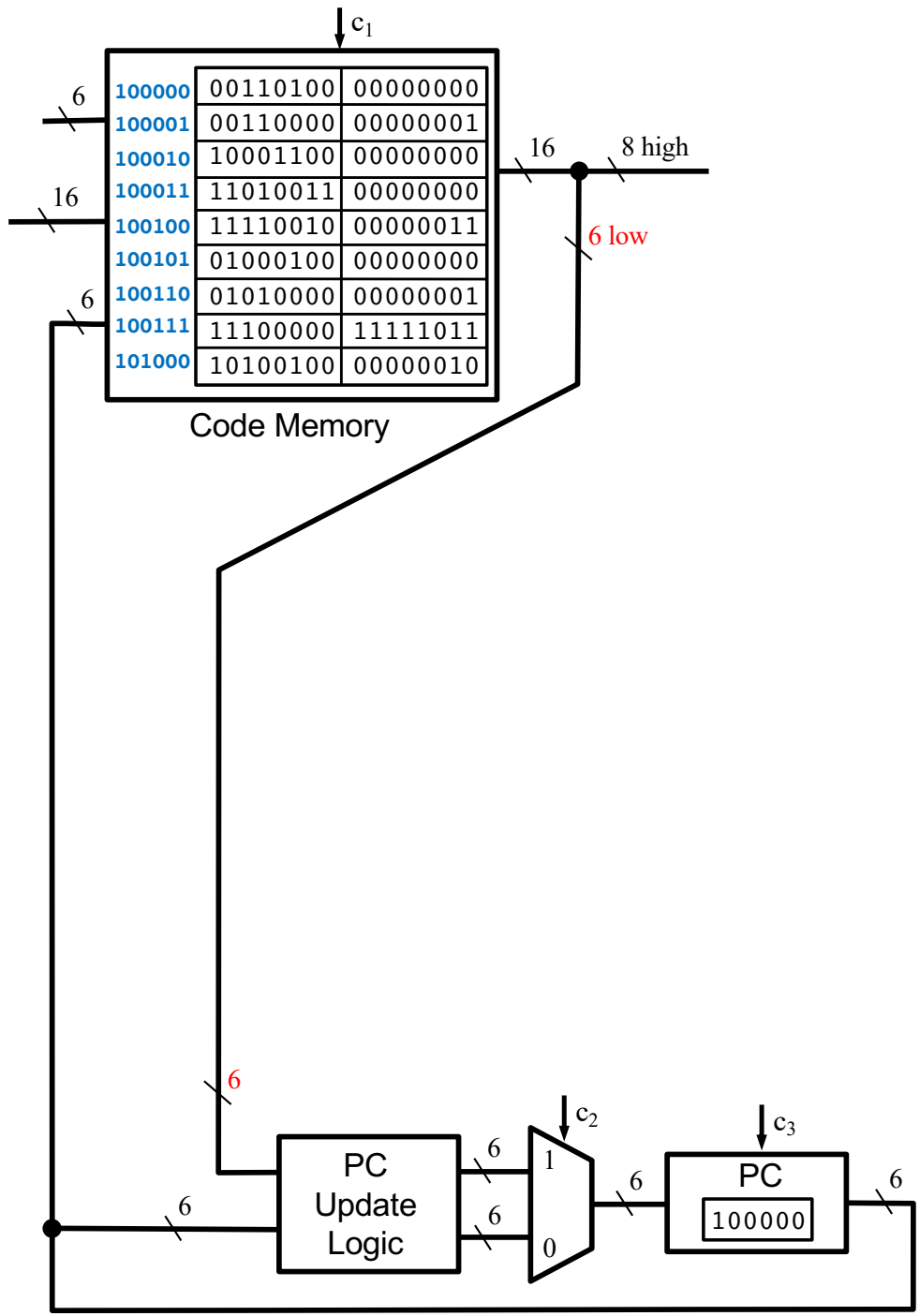


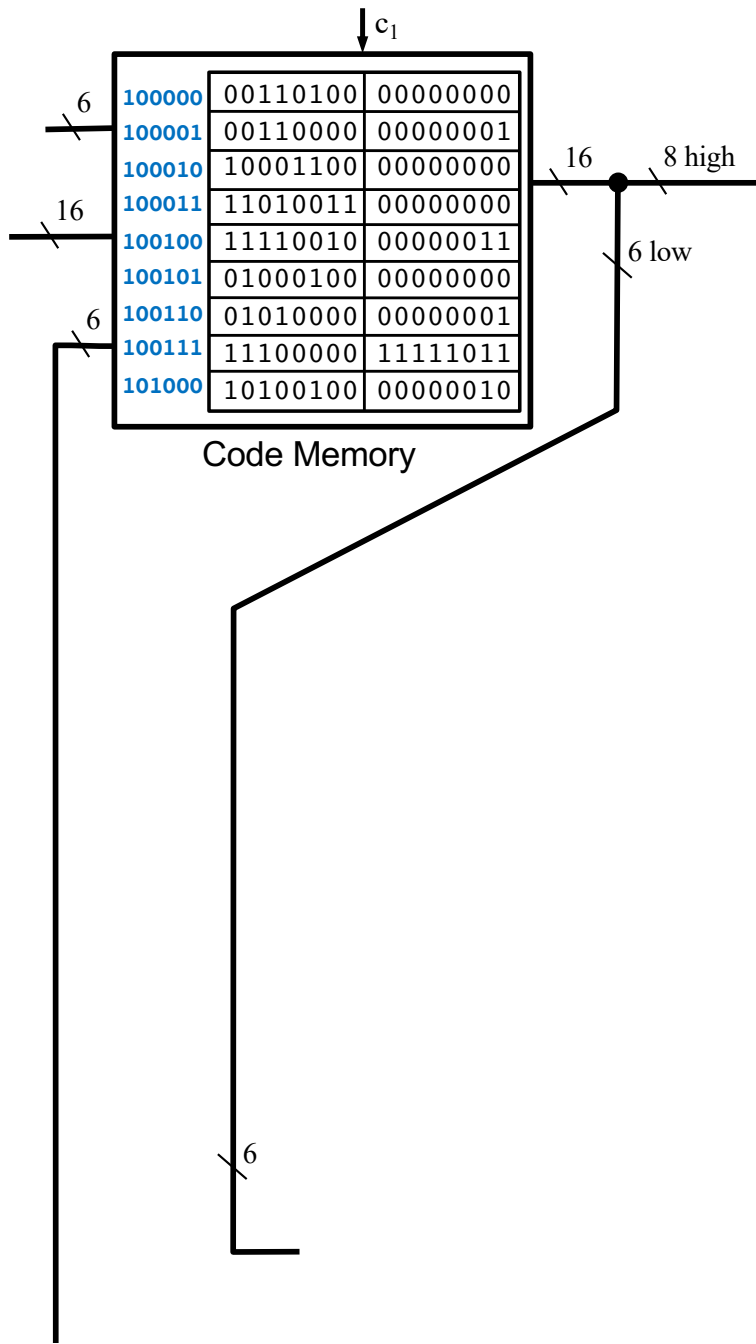


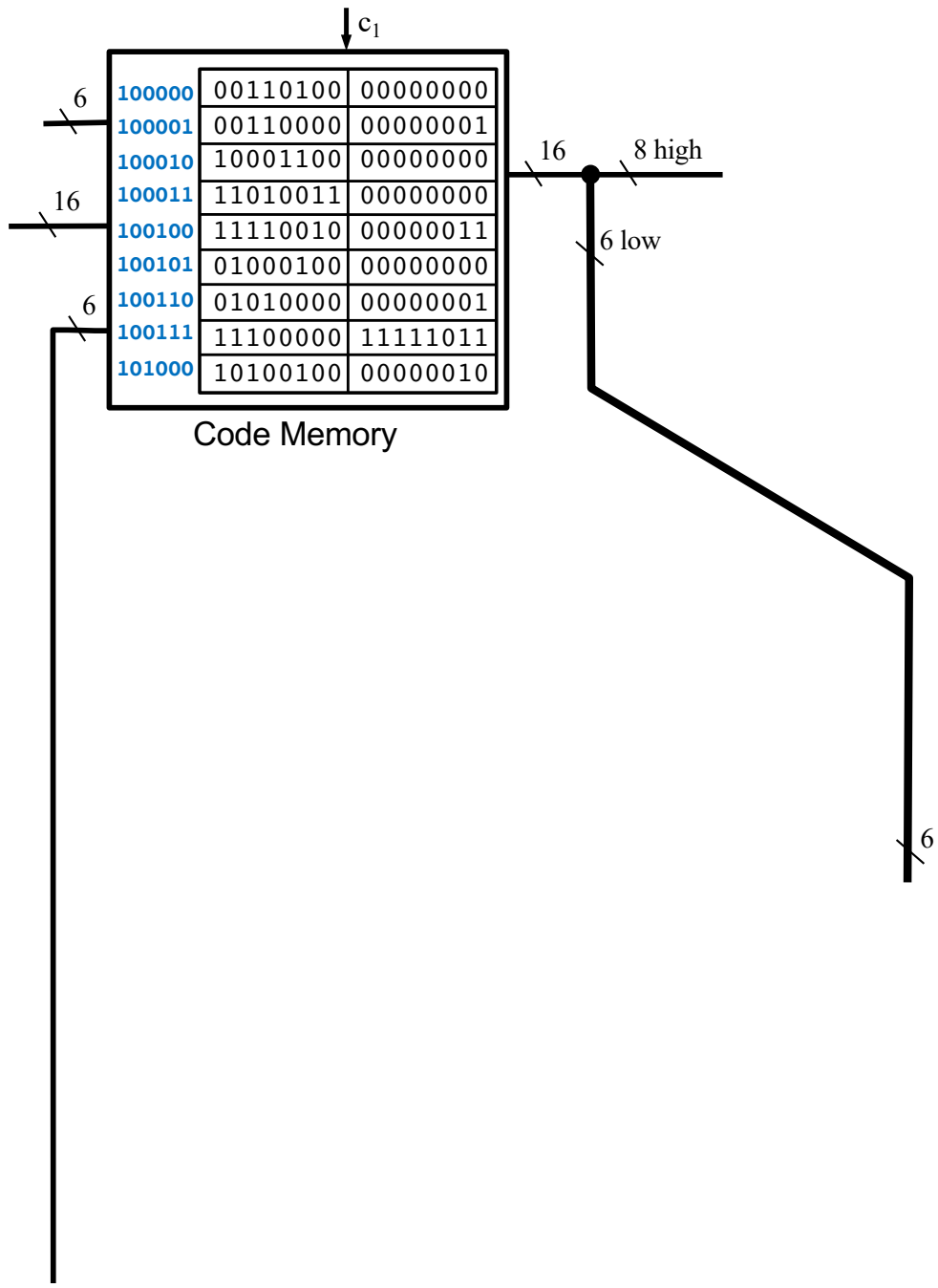


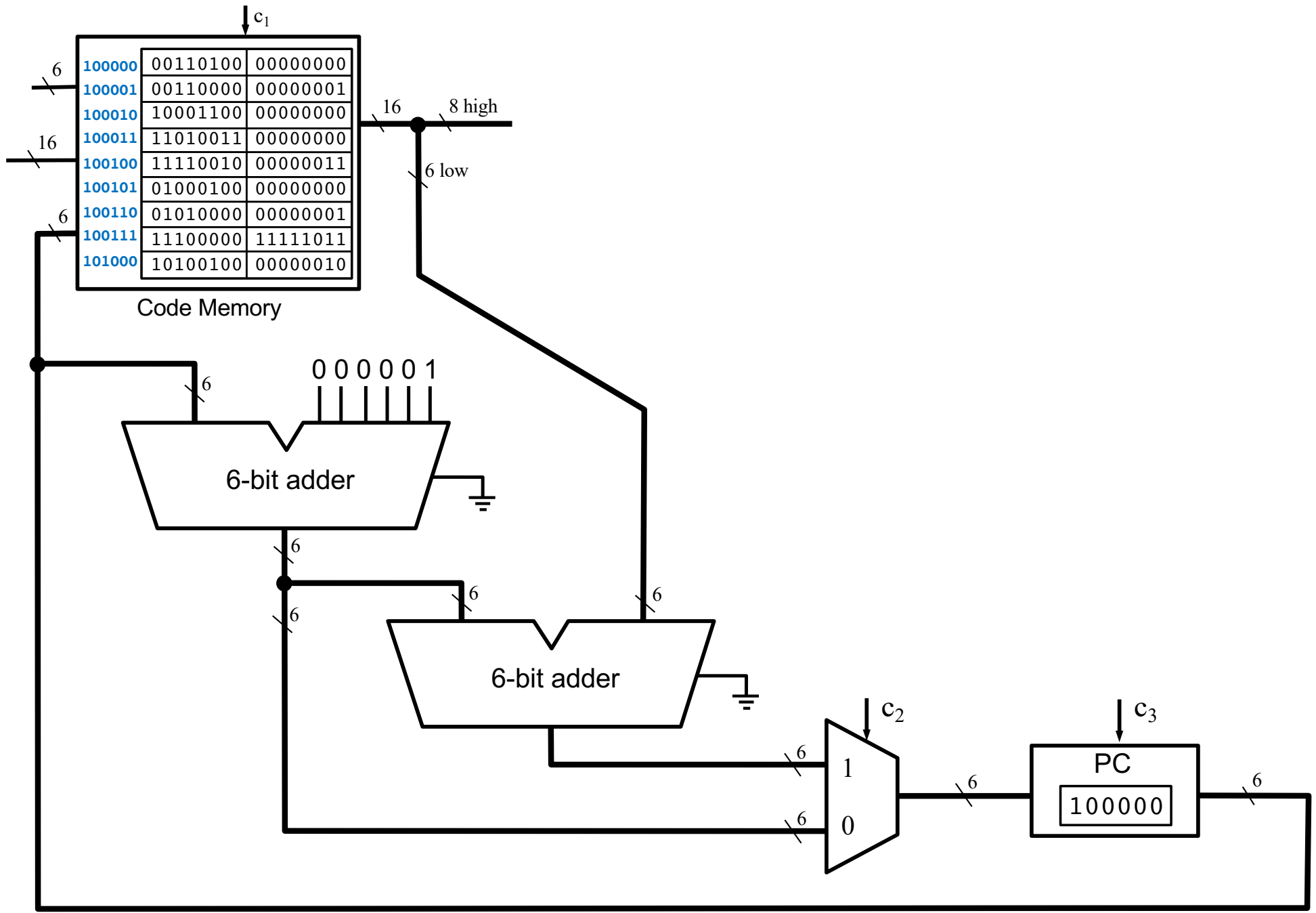


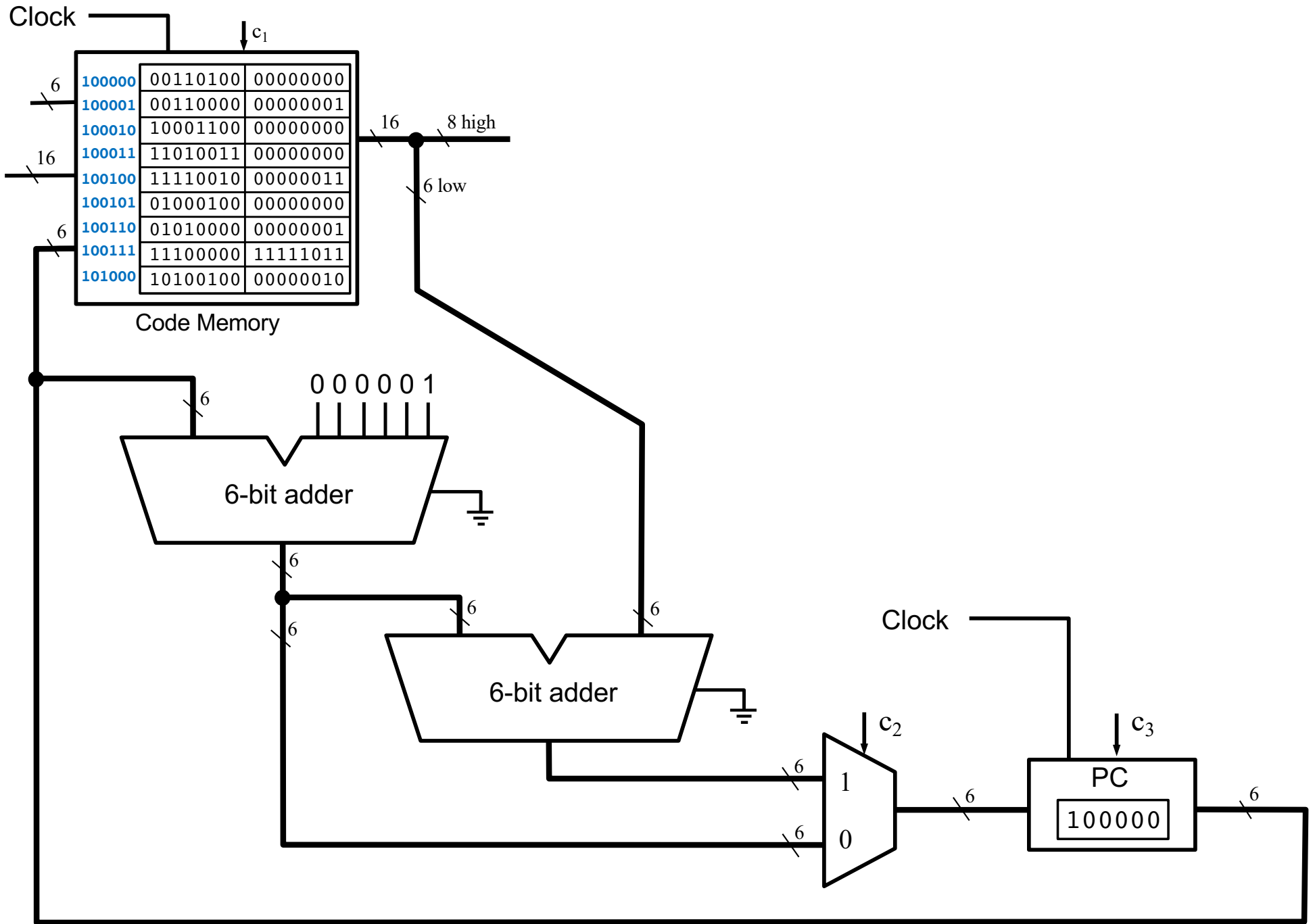




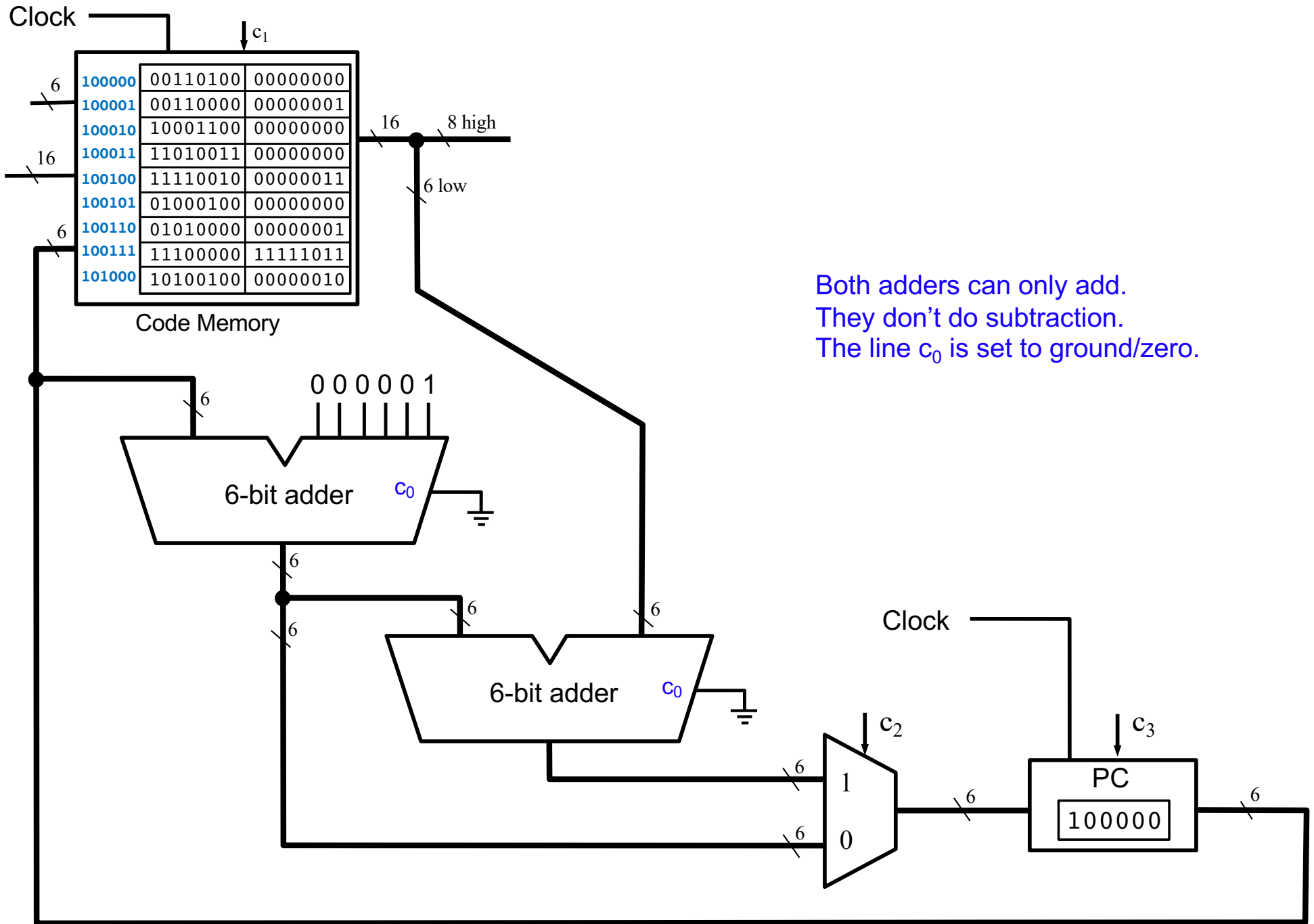






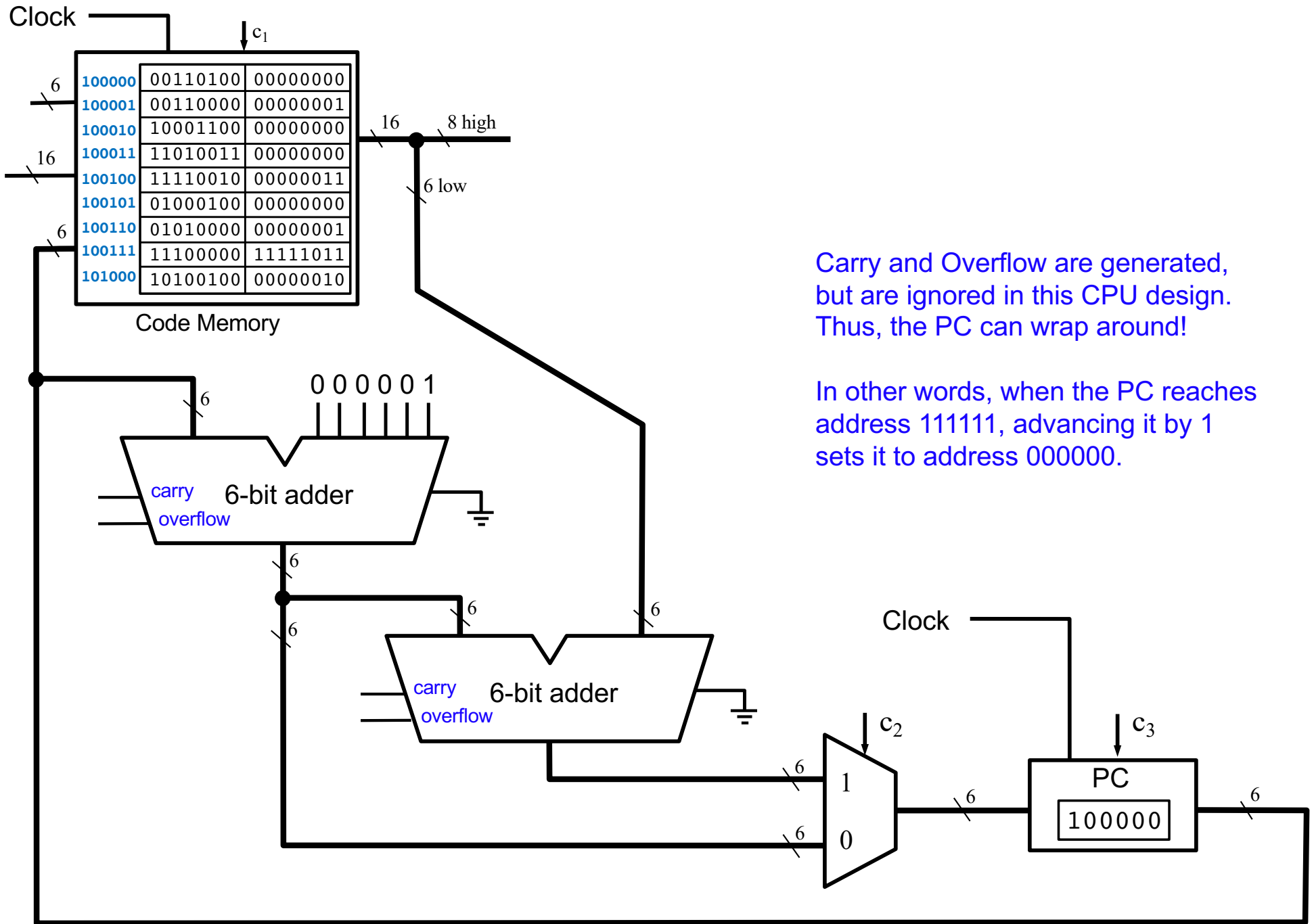






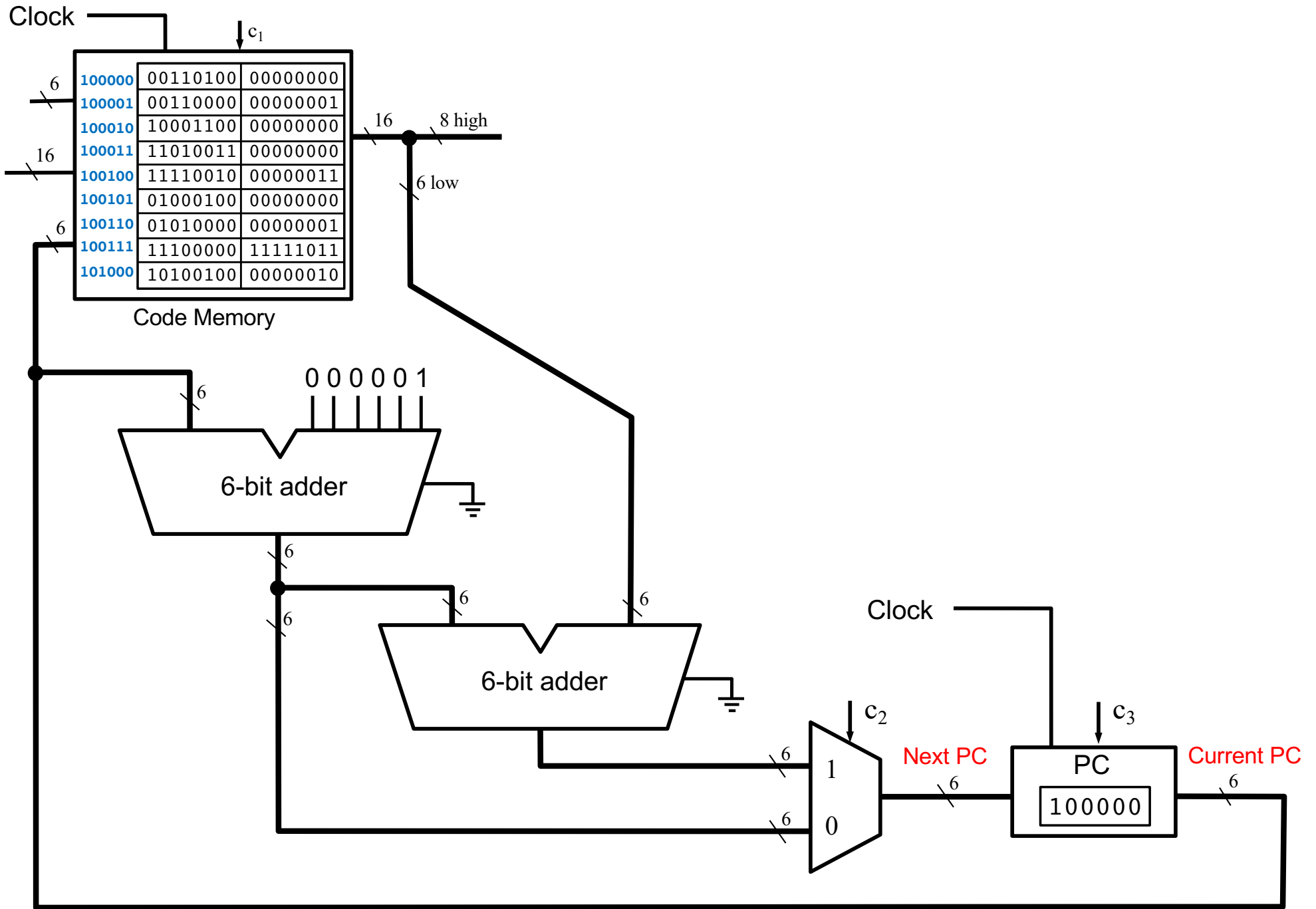
Both adders can only add.  
They don't do subtraction.  
The line  $c_0$  is set to ground/zero.

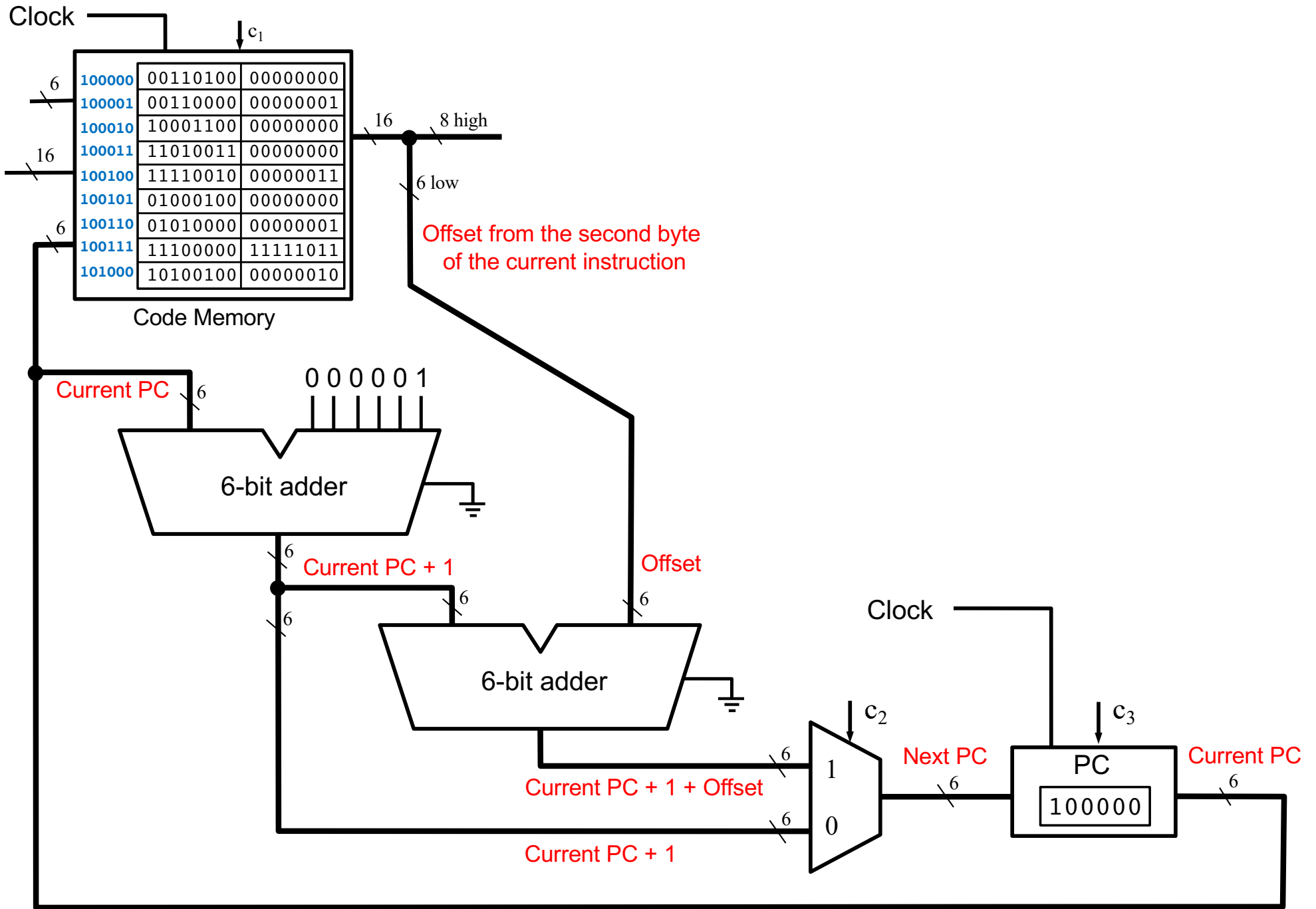


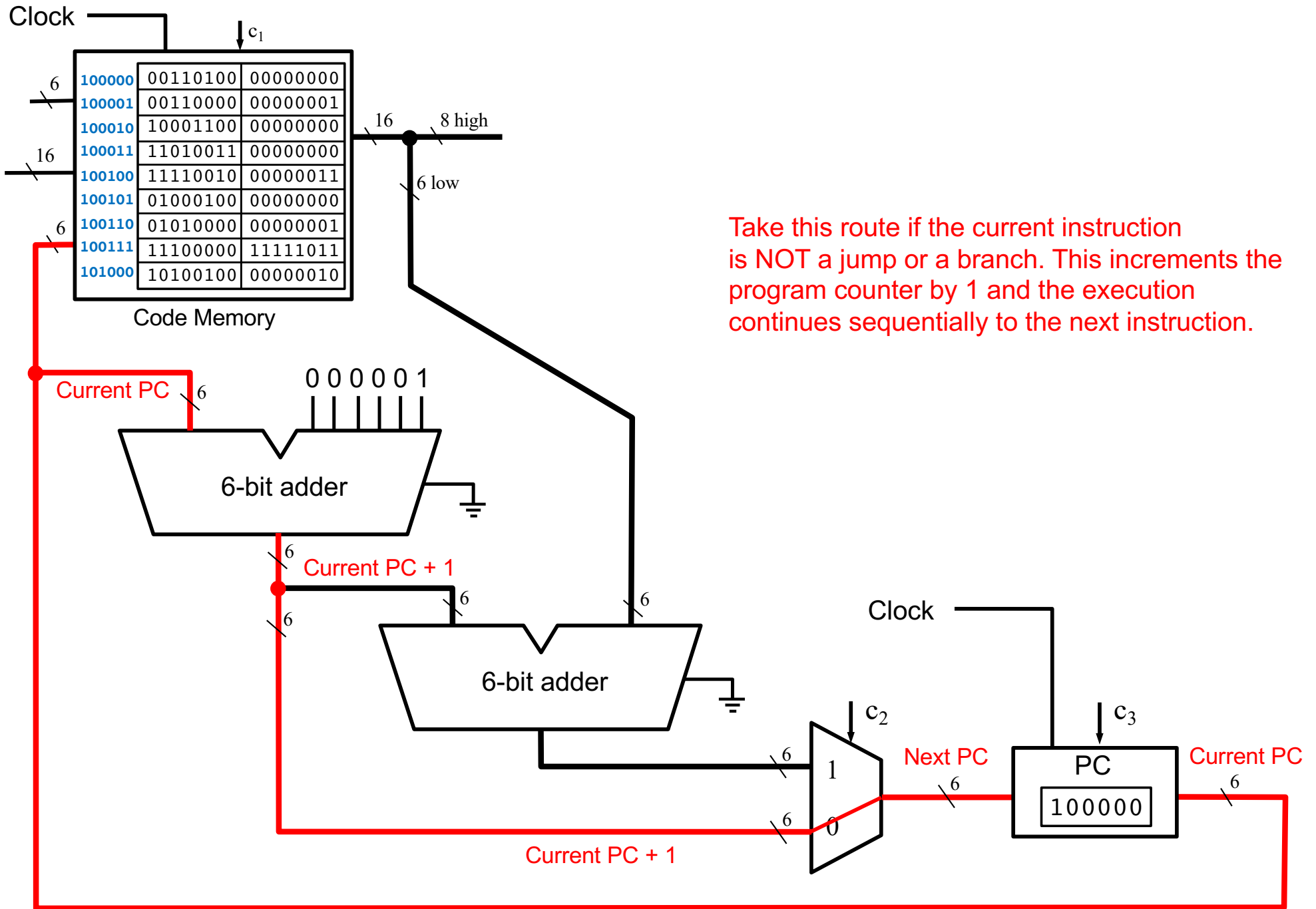


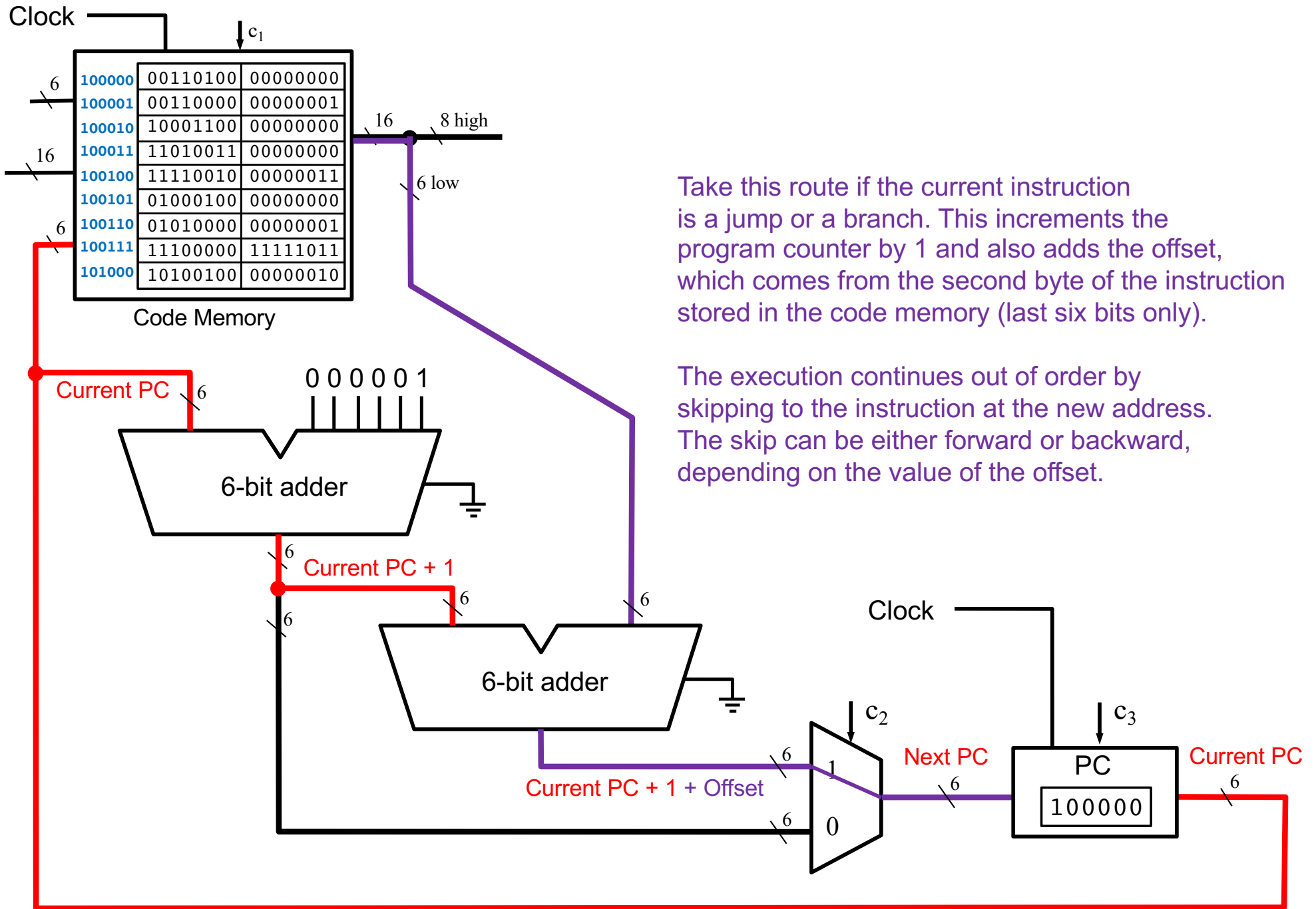
Carry and Overflow are generated, but are ignored in this CPU design. Thus, the PC can wrap around!

In other words, when the PC reaches address 11111, advancing it by 1 sets it to address 00000.









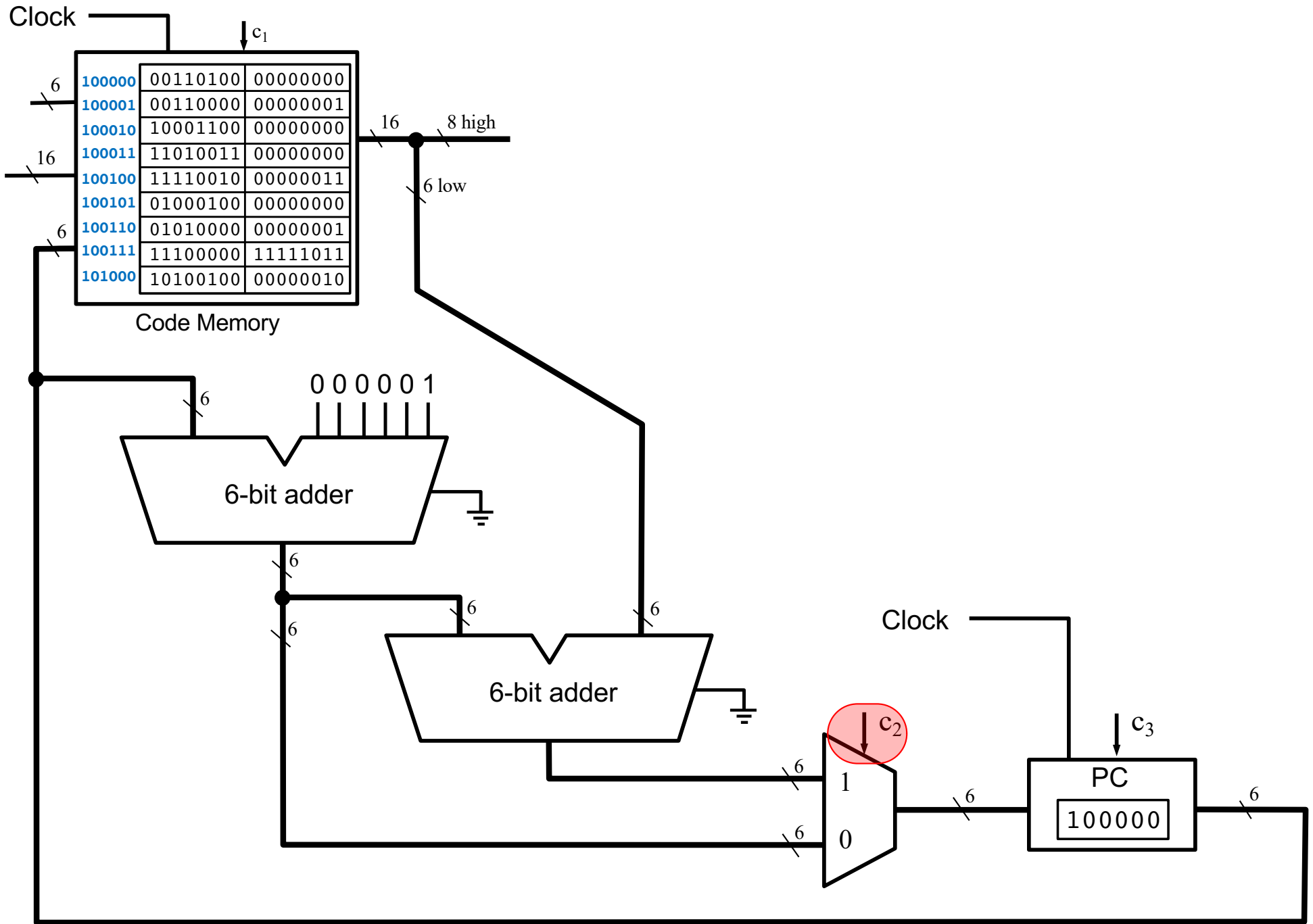
Take this route if the current instruction is a jump or a branch. This increments the program counter by 1 and also adds the offset, which comes from the second byte of the instruction stored in the code memory (last six bits only).

The execution continues out of order by skipping to the instruction at the new address. The skip can be either forward or backward, depending on the value of the offset.

# Offsets with +1 Correction

0	11111111	-1	11111110
1	00000000	-2	11111101
2	00000001	-3	11111100
3	00000010	-4	11111011
4	00000011	-5	11111010
5	00000100	-6	11111001
6	00000101	-7	11111000
7	00000110	-8	11110111
8	00000111	-9	11110110
9	00001000	-10	11110101
10	00001001	-11	11110100
11	00001010	-12	11110011
12	00001011	-13	11110010
13	00001100	-14	11110001
14	00001101	-15	11110000
15	00001110	-16	11101111
16	00001111	-17	11101110
17	00010000	-18	11101101
18	00010001	-19	11101100
19	00010010	-20	11101011
20	00010011	-21	11101010
21	00010100	-22	11101001
22	00010101	-23	11101000
23	00010110	-24	11100111
24	00010111	-25	11100110
25	00011000	-26	11100101
26	00011001	-27	11100100
27	00011010	-28	11100011
28	00011011	-29	11100010
29	00011100	-30	11100001
30	00011101	-31	11100000
31	00011110	-32	N/A
N/A	00011111		





	C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>	C <sub>5</sub>	C <sub>6</sub>	C <sub>7</sub>	C <sub>8</sub>	C <sub>9</sub>	C <sub>10</sub>	C <sub>11</sub>	C <sub>12</sub>	C <sub>13</sub>	C <sub>14</sub>	C <sub>15</sub>	C <sub>16</sub>	C <sub>17</sub>	C <sub>18</sub>
	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_EN	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	ALU_SOURCE_MUX	ALU_SELECT1	ALU_SELECT0	FLAGS_WRITE_ENABLE	ALU_RESUT_MUX	DMEM_INPUT_MUX	DMEM_WRITE_ENABLE	REG_WRITEBACK_MUX
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1												1	1	1	
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADP			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

computed using  
the flags register

B1= ZF  
 B2= ~ZF  
 B3= AND (~ZF, XNOR(NF, OF))  
 B4= XNOR(NF, OF)

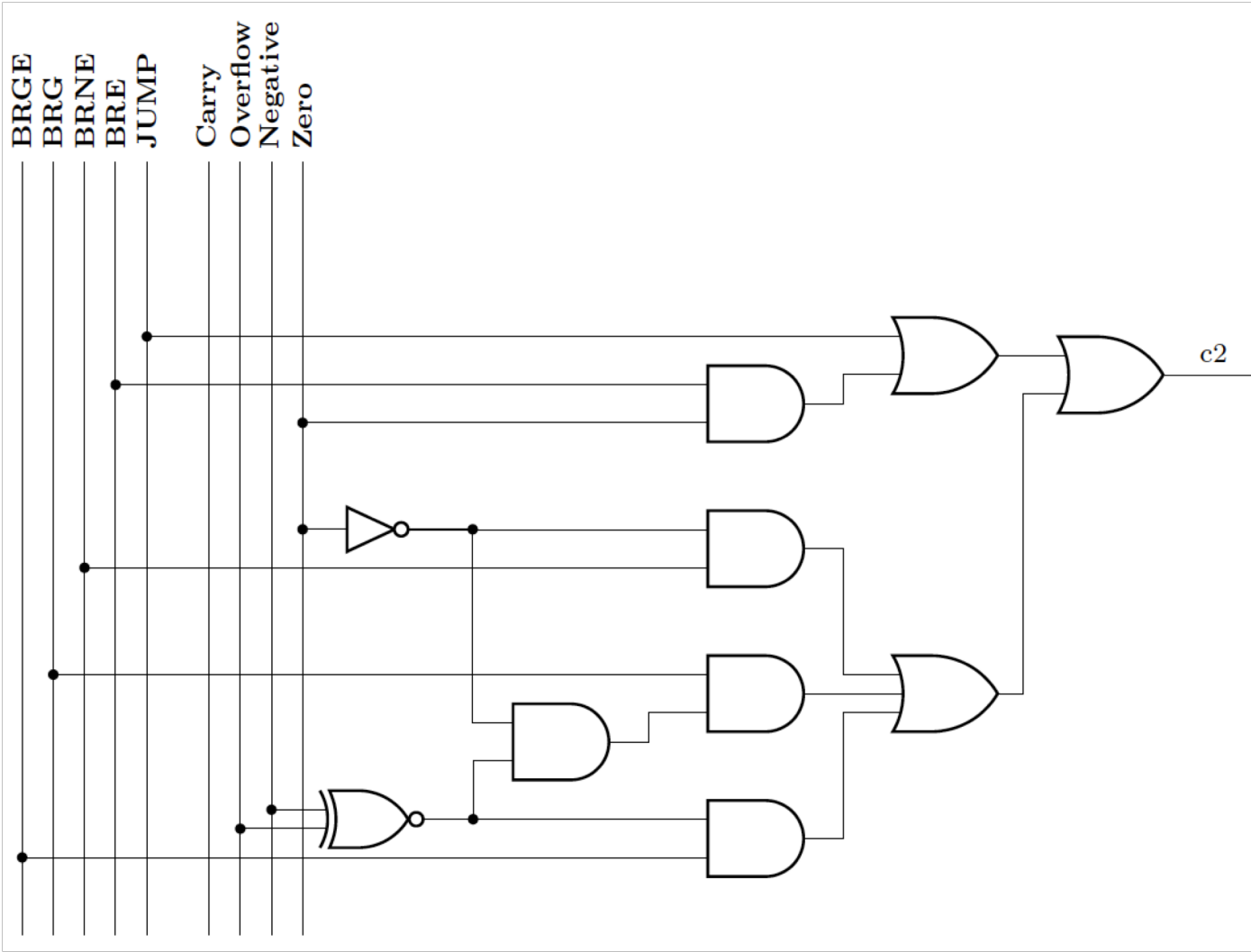
Zero Flag (ZF)  
 Negative Flag (NF)  
 Overflow Flag (OF)

	C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>	C <sub>5</sub>	C <sub>6</sub>	C <sub>7</sub>	C <sub>8</sub>	C <sub>9</sub>	C <sub>10</sub>	C <sub>11</sub>	C <sub>12</sub>	C <sub>13</sub>	C <sub>14</sub>	C <sub>15</sub>	C <sub>16</sub>	C <sub>17</sub>	C <sub>18</sub>
	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_EN	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	ALU_SOURCE_MUX	ALU_SELECT1	ALU_SELECT0	FLAGS_WRITE_ENABLE	ALU_RESUT_MUX	DMEM_INPUT_MUX	DMEM_WRITE_ENABLE	REG_WRITEBACK_MUX
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1											1	1	1		
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADP			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

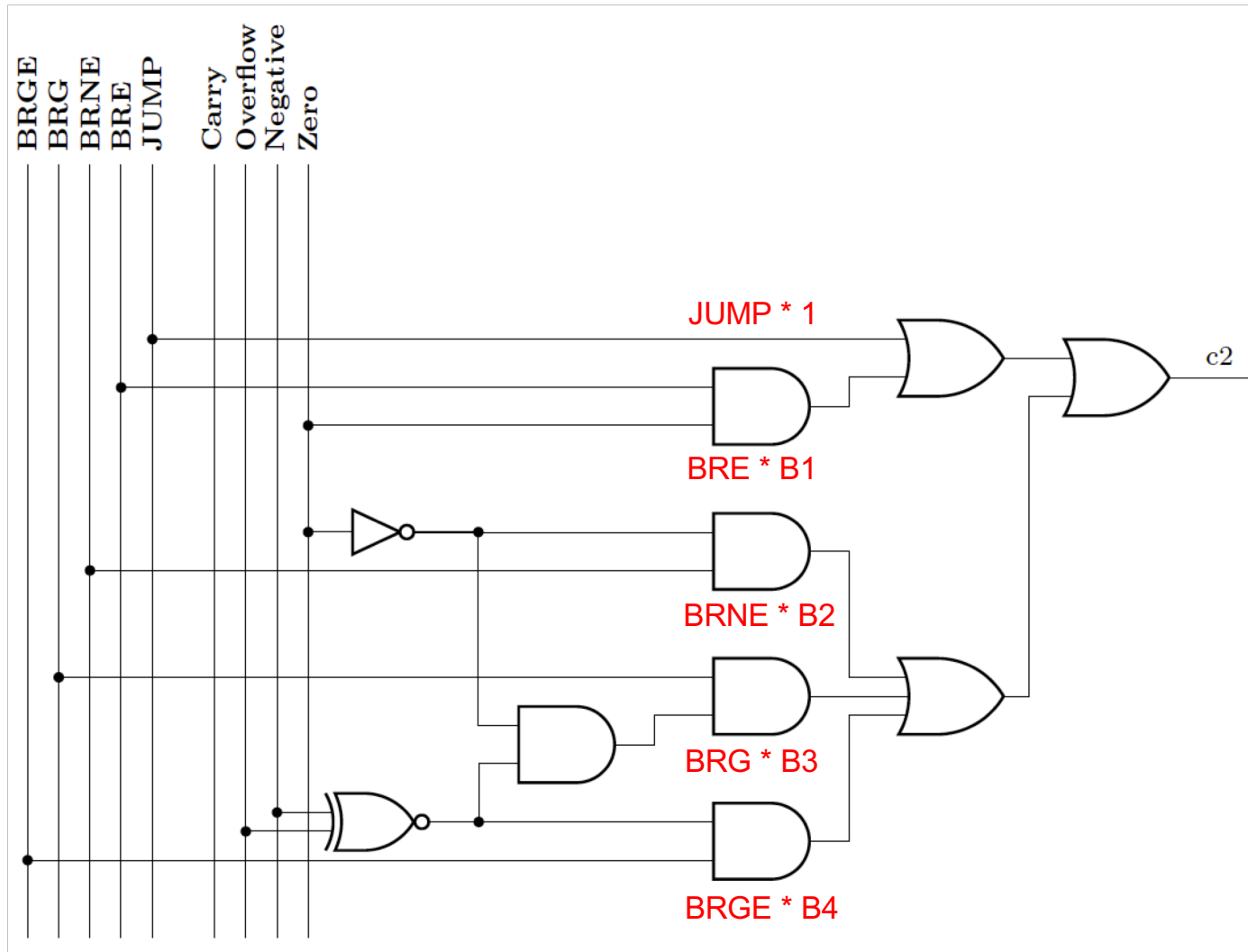
C<sub>2</sub> is the OR of these five times the OPCODE

B1= ZF  
 B2= ~ZF  
 B3= AND (~ZF, XNOR(NF, OF))  
 B4= XNOR(NF, OF)

Zero Flag (ZF)  
 Negative Flag (NF)  
 Overflow Flag (OF)





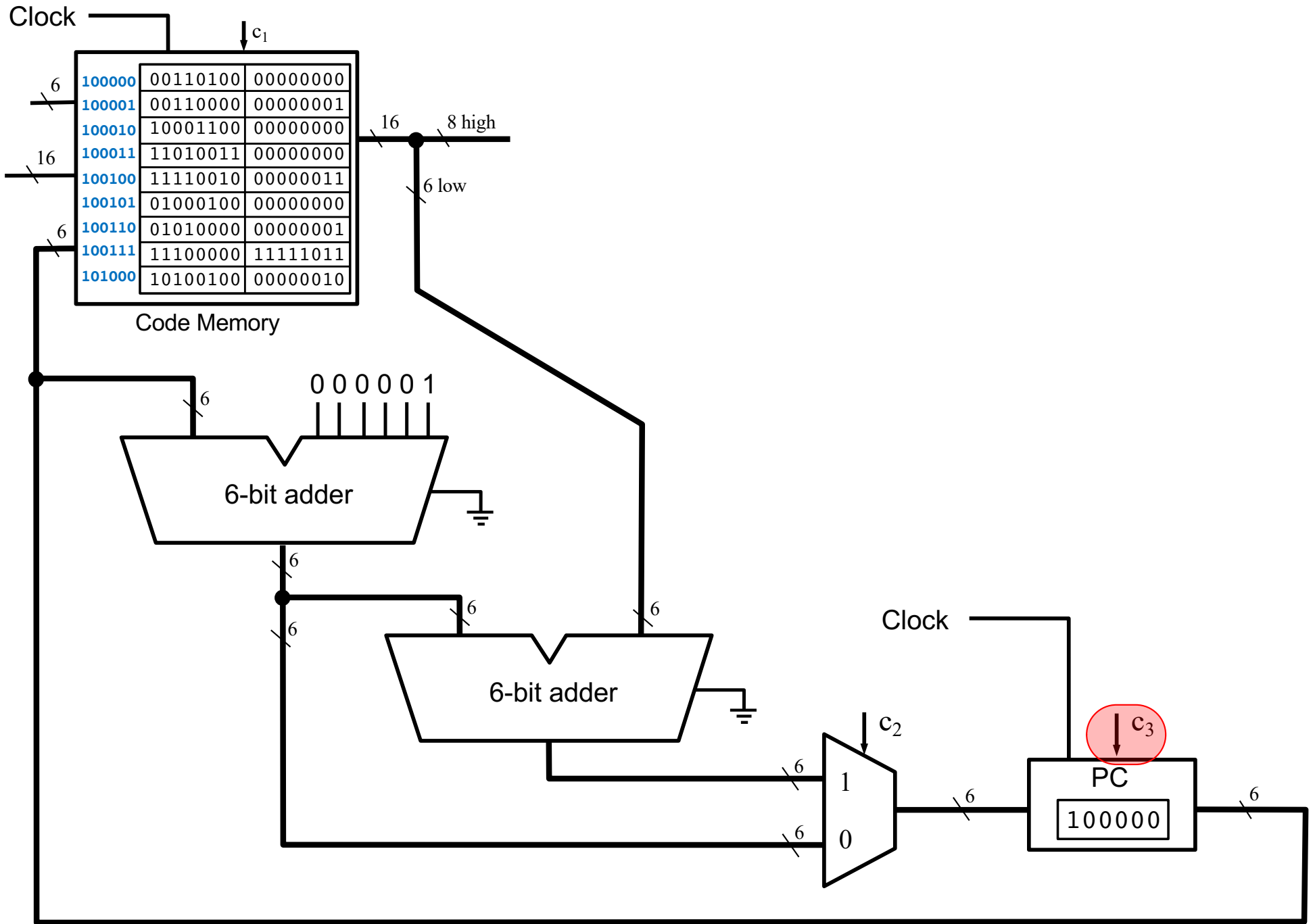


JUMP	1	1																		
BRE/BRZ	B1	1																		
BRNE/BRNZ	B2	1																		
BRG	B3	1																		
BRGE	B4	1																		

**C<sub>2</sub> is the OR of these five times the OPCODE**

B1= ZF  
 B2= ~ZF  
 B3= AND (~ZF, XNOR (NF, OF))  
 B4= XNOR (NF, OF)

Zero Flag (ZF)  
 Negative Flag (NF)  
 Overflow Flag (OF)



	C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>	C <sub>5</sub>	C <sub>6</sub>	C <sub>7</sub>	C <sub>8</sub>	C <sub>9</sub>	C <sub>10</sub>	C <sub>11</sub>	C <sub>12</sub>	C <sub>13</sub>	C <sub>14</sub>	C <sub>15</sub>	C <sub>16</sub>	C <sub>17</sub>	C <sub>18</sub>
	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_EN	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	ALU_SOURCE_MUX	ALU_SELECT1	ALU_SELECT0	FLAGS_WRITE_ENABLE	ALU_RESUT_MUX	DMEM_INPUT_MUX	DMEM_WRITE_ENABLE	REG_WRITEBACK_MUX
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1												1	1	1	
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADP			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

c<sub>3</sub> is always 1, because this is a single-cycle processor, i.e., it executes one instruction per clock cycle.



**Simulation of the For Loop program  
that adds the numbers from 1 to 5**

# Add the numbers from 1 to 5

```
// C Version
// using a for loop

int N=5;
int i;
int sum;

int main()
{
    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]     ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

# Add the numbers from 1 to 5

## Machine Code Version

### Data Memory:

00000101

00000000

00000000

### Code Memory:

0011010000000000

0011000000000001

1000110000000000

1101001100000000

1111001000000011

0100010000000000

0101000000000001

1110000011111011

1010010000000010

## ; Assembly Version

### .data

N            BYTE     5

i            BYTE     ?

sum          BYTE     ?

### .code

          LOADI    B, 0            ; sum=0

          LOADI    A, 1            ; i=1

          LOAD     D, [N]          ; register\_D=N

Loop:     CMP      A, D            ; i<=N ?

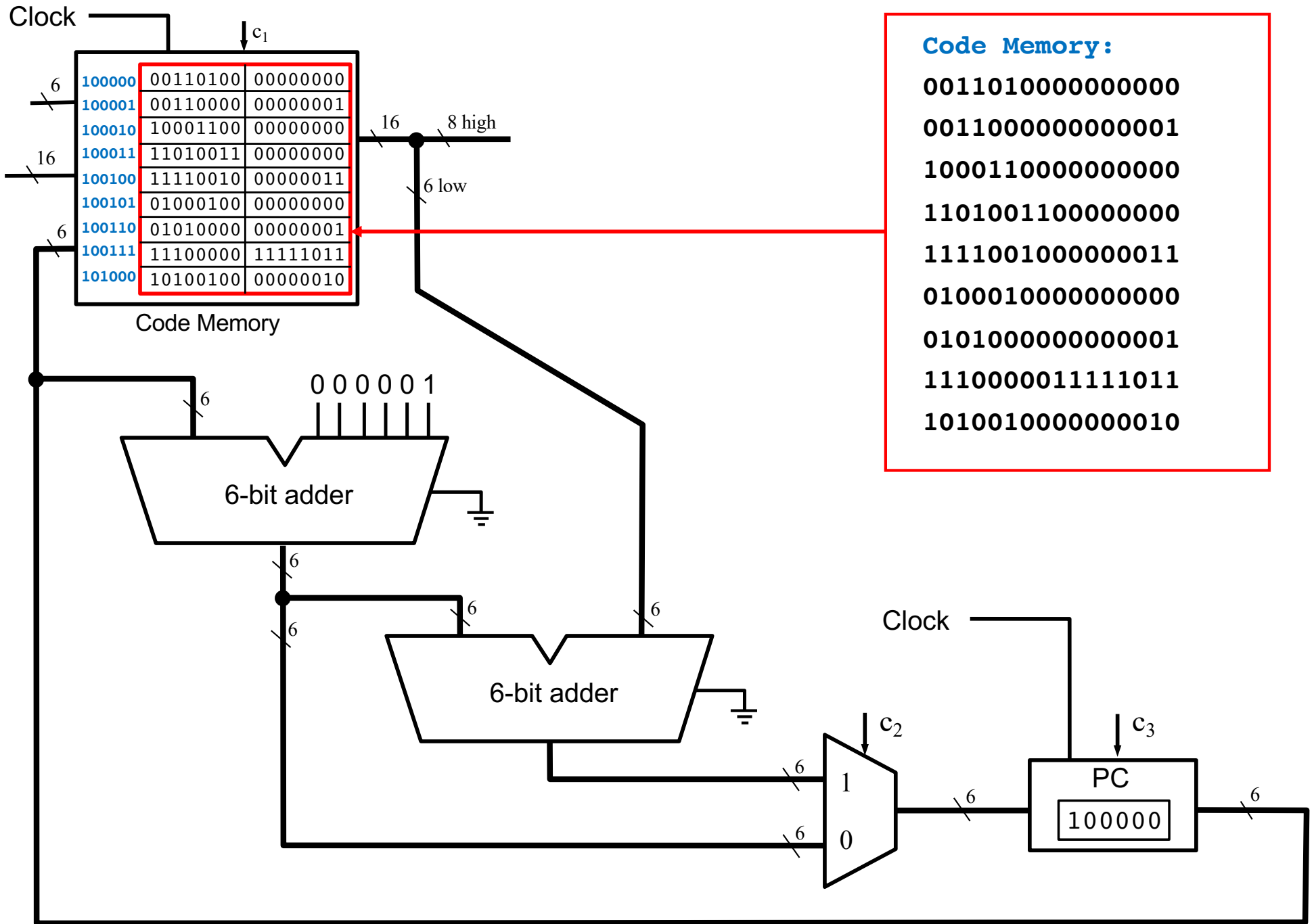
          BRG      End            ; exit if i>N

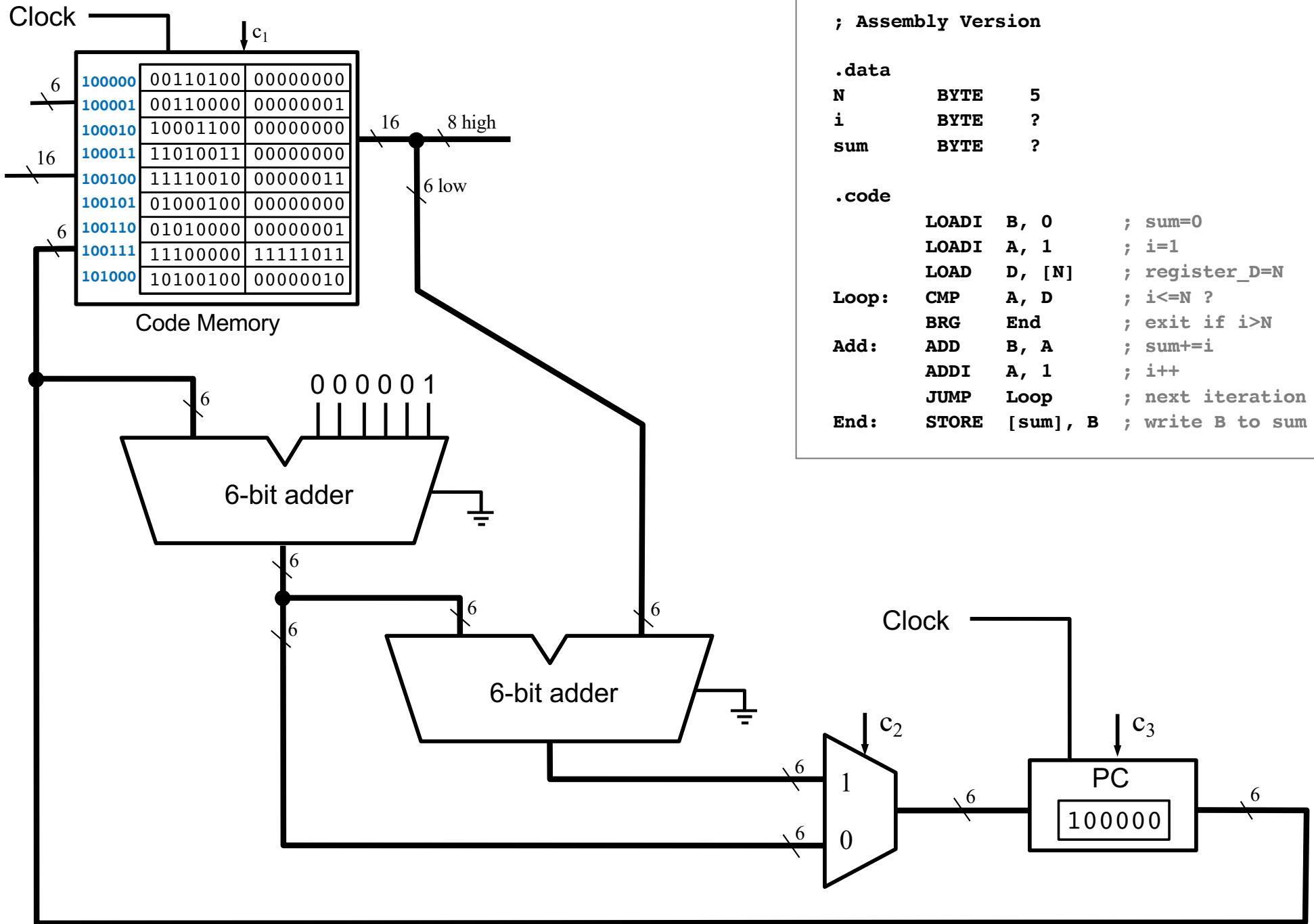
Add:     ADD      B, A            ; sum+=i

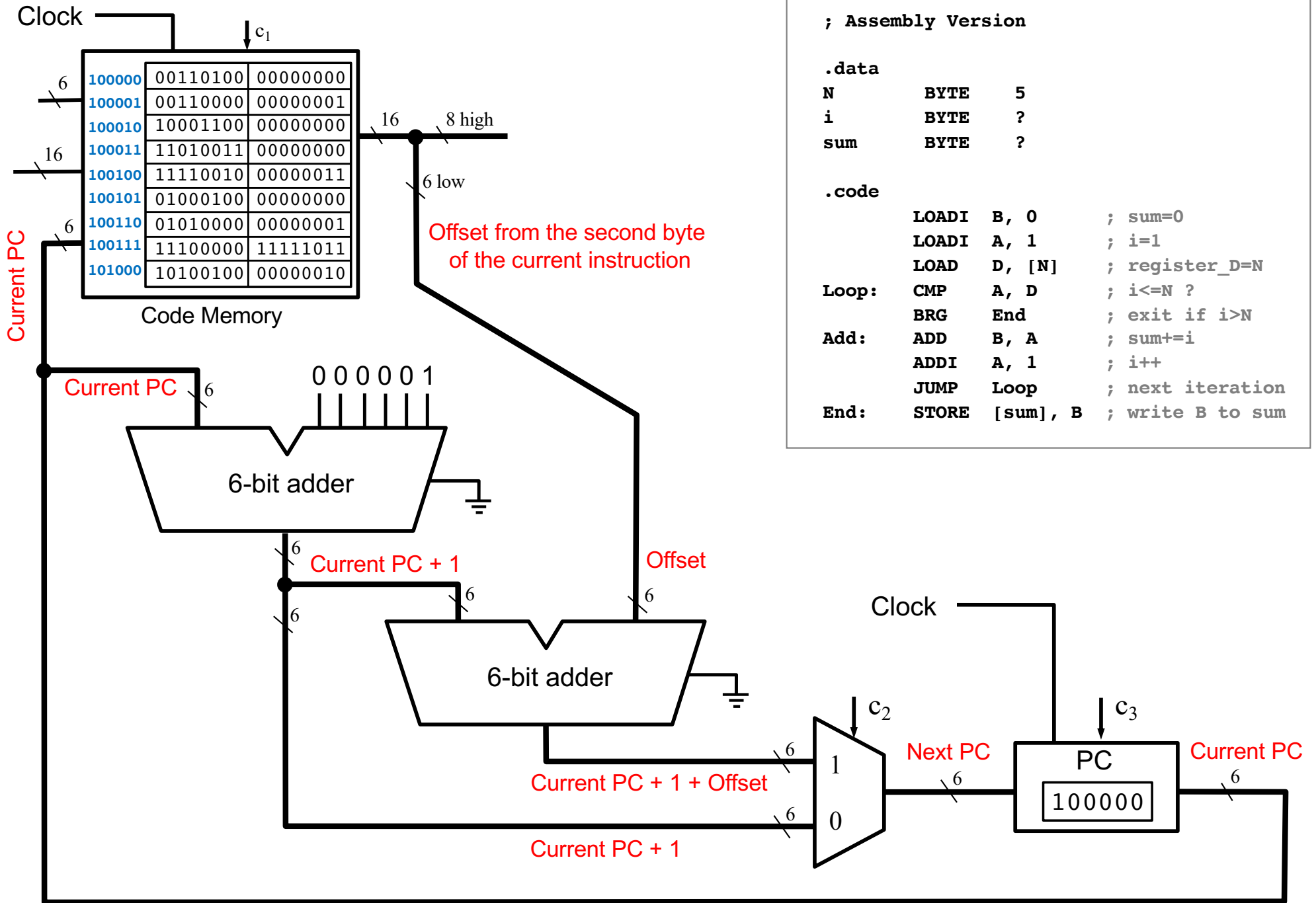
          ADDI     A, 1            ; i++

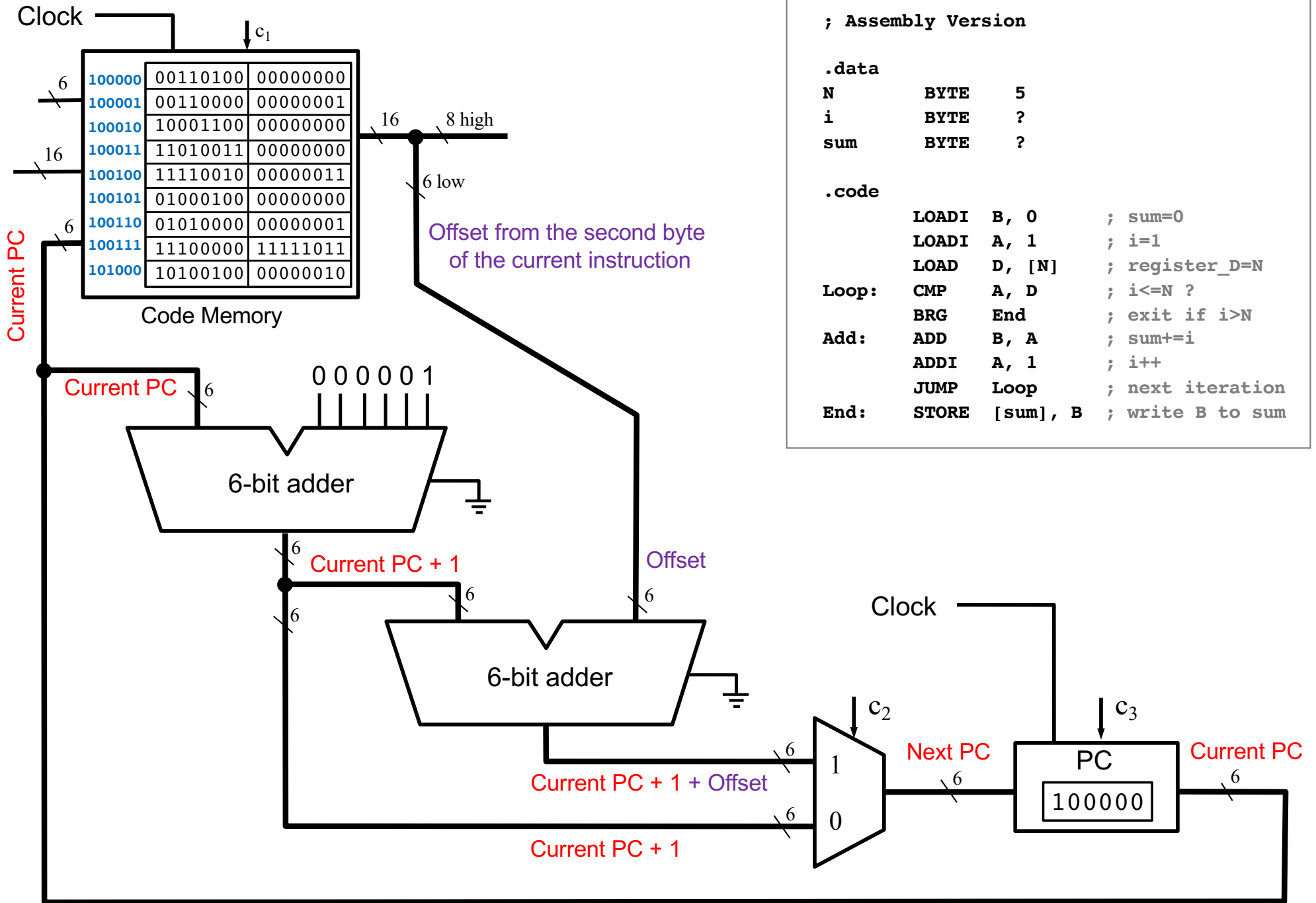
          JUMP     Loop           ; next iteration

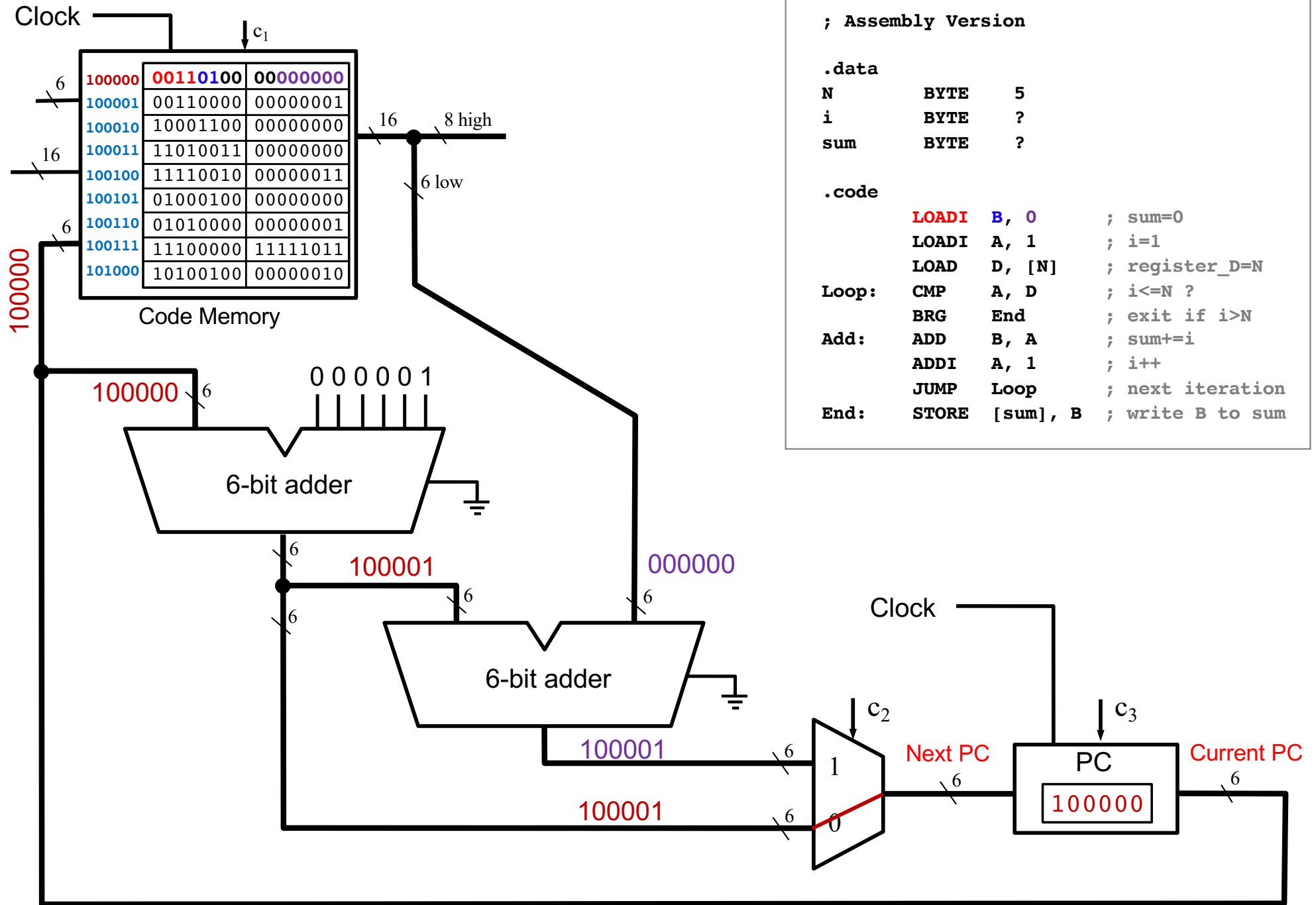
End:      STORE    [sum], B       ; write B to sum



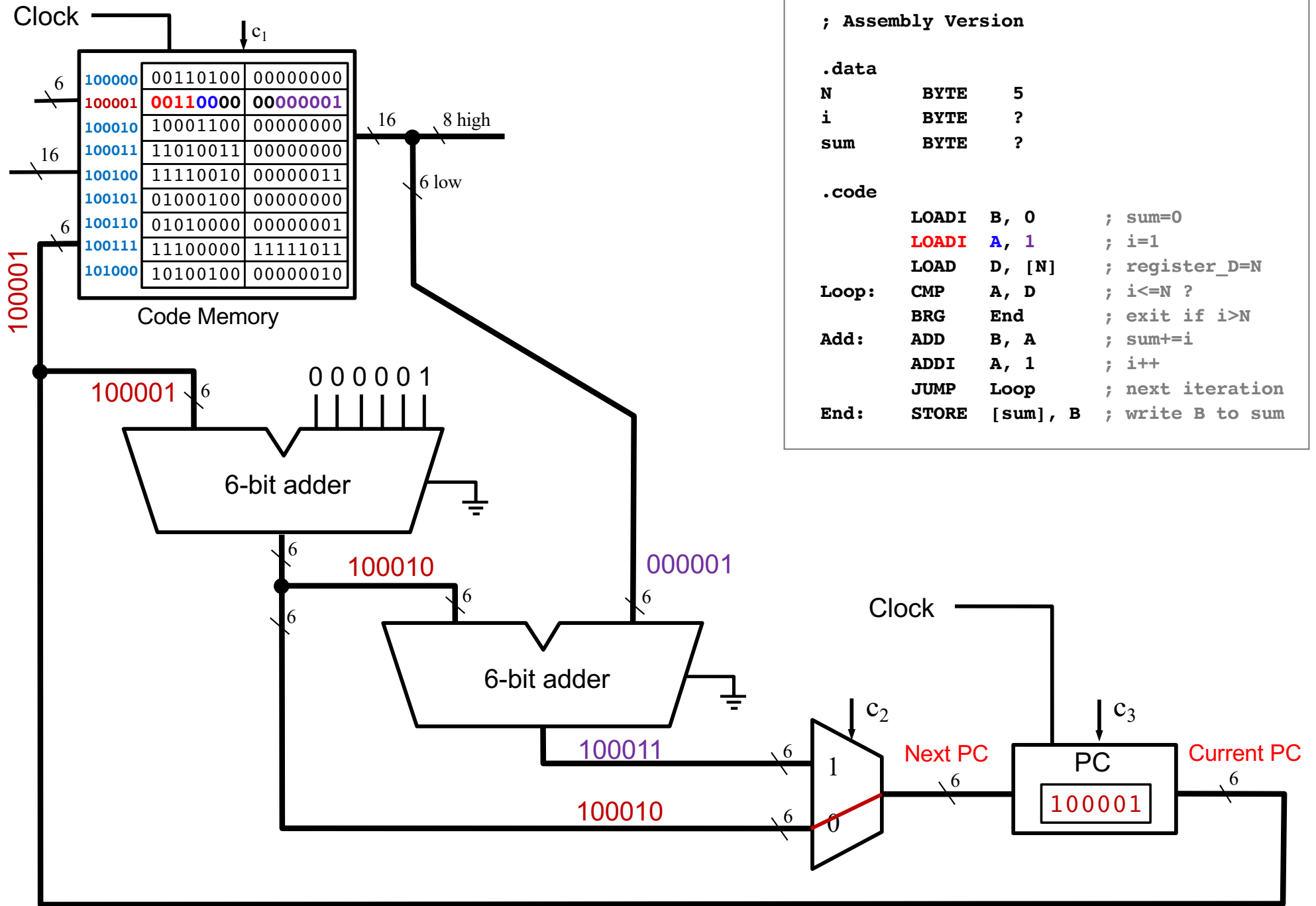


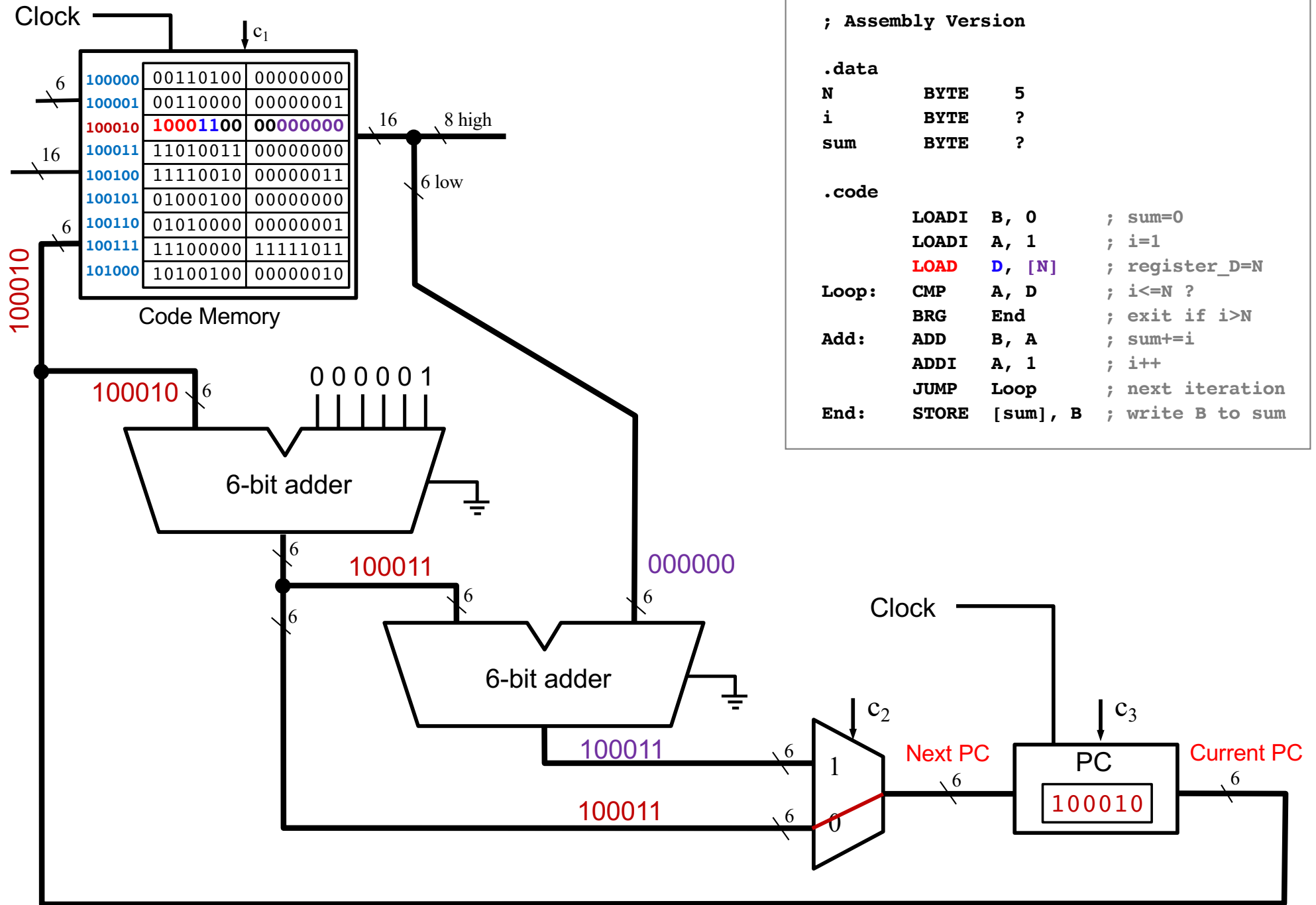


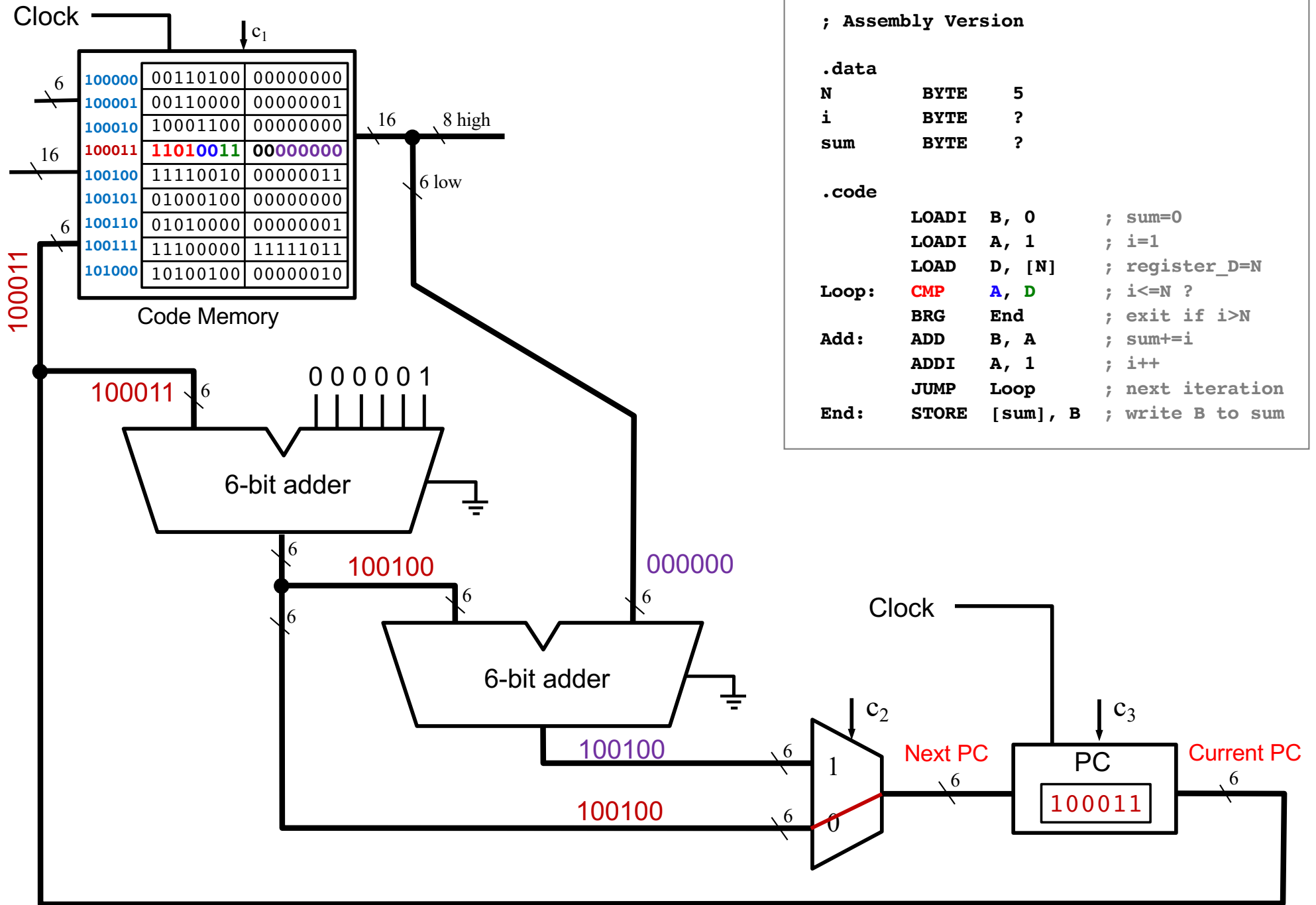


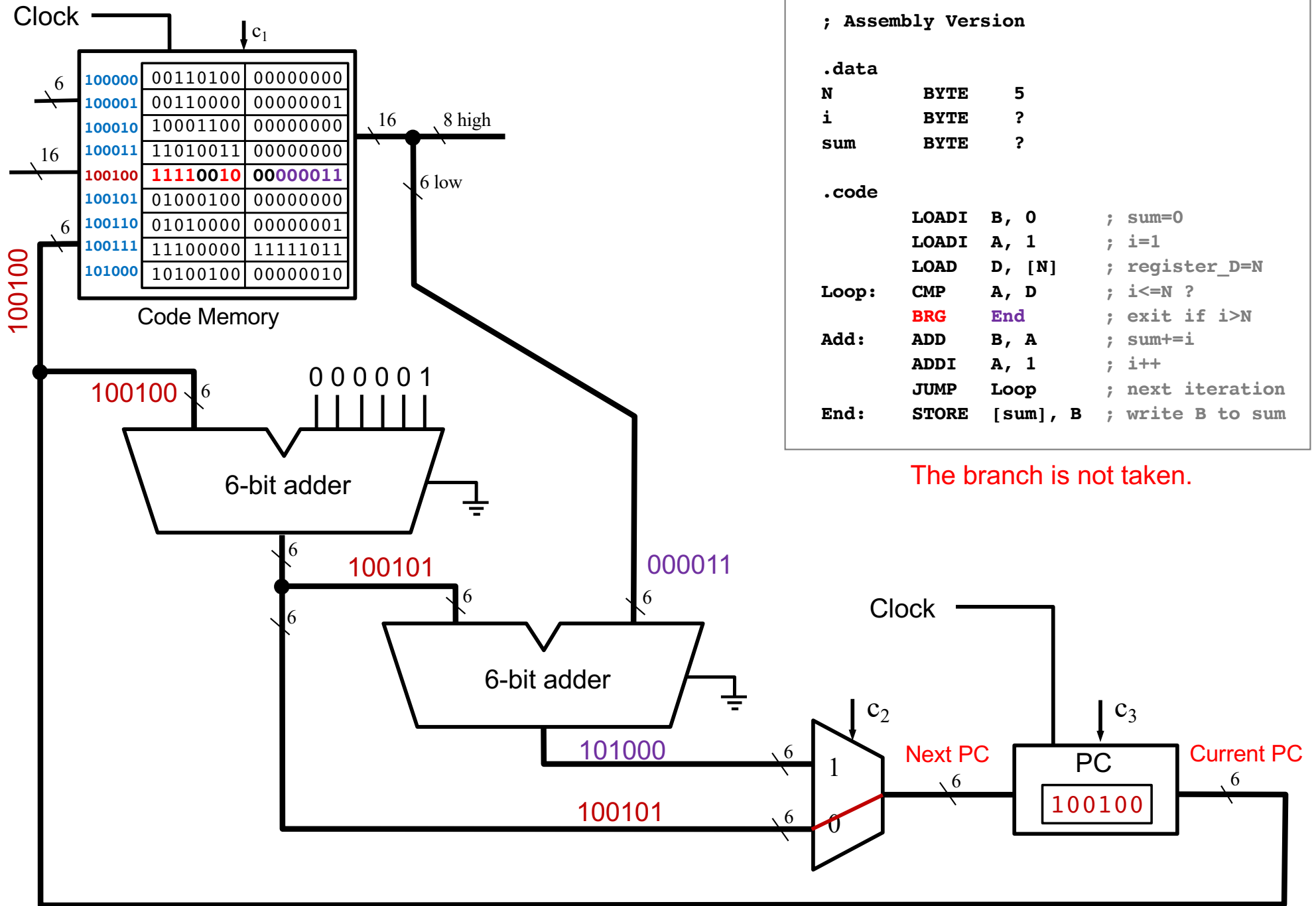


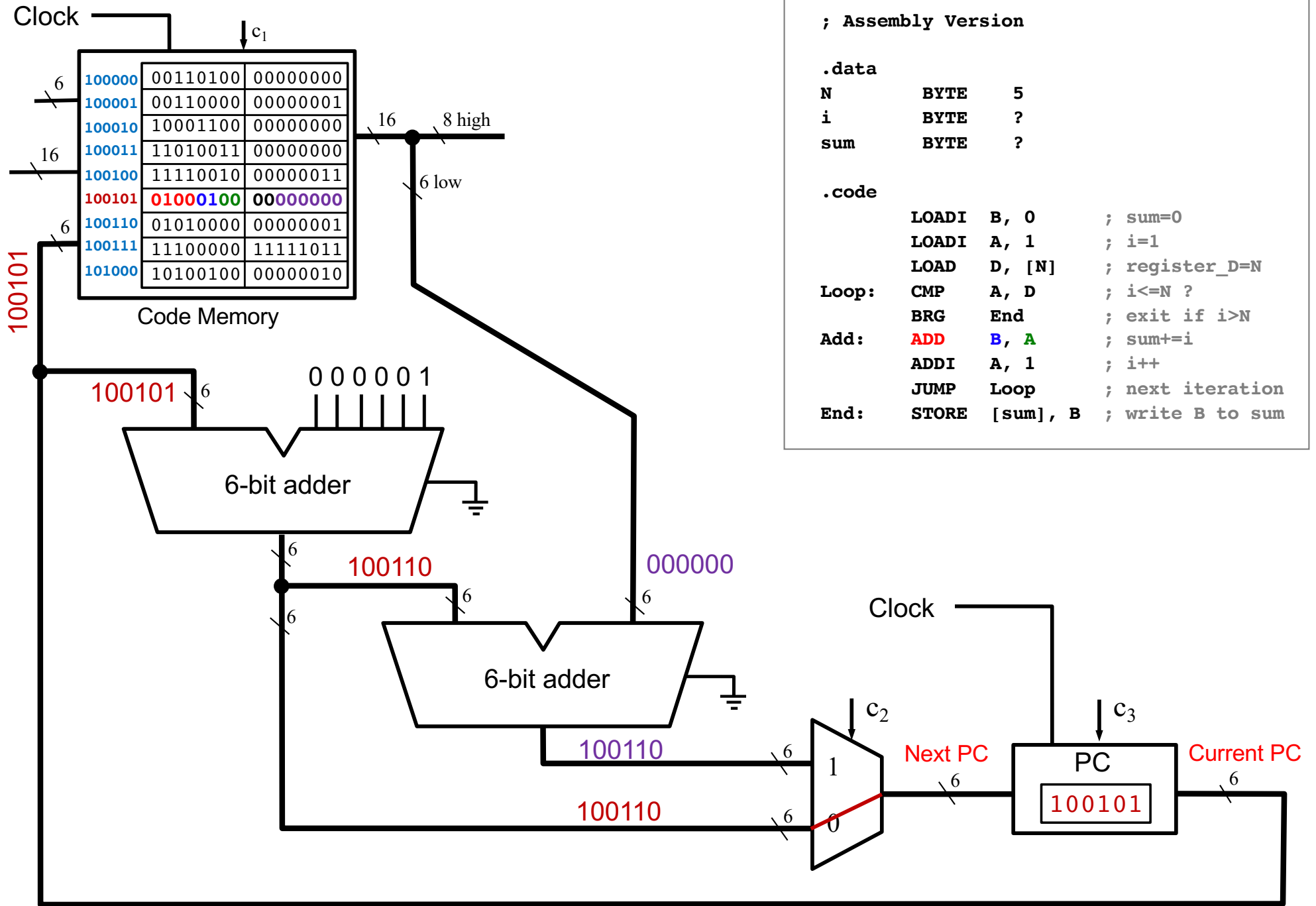


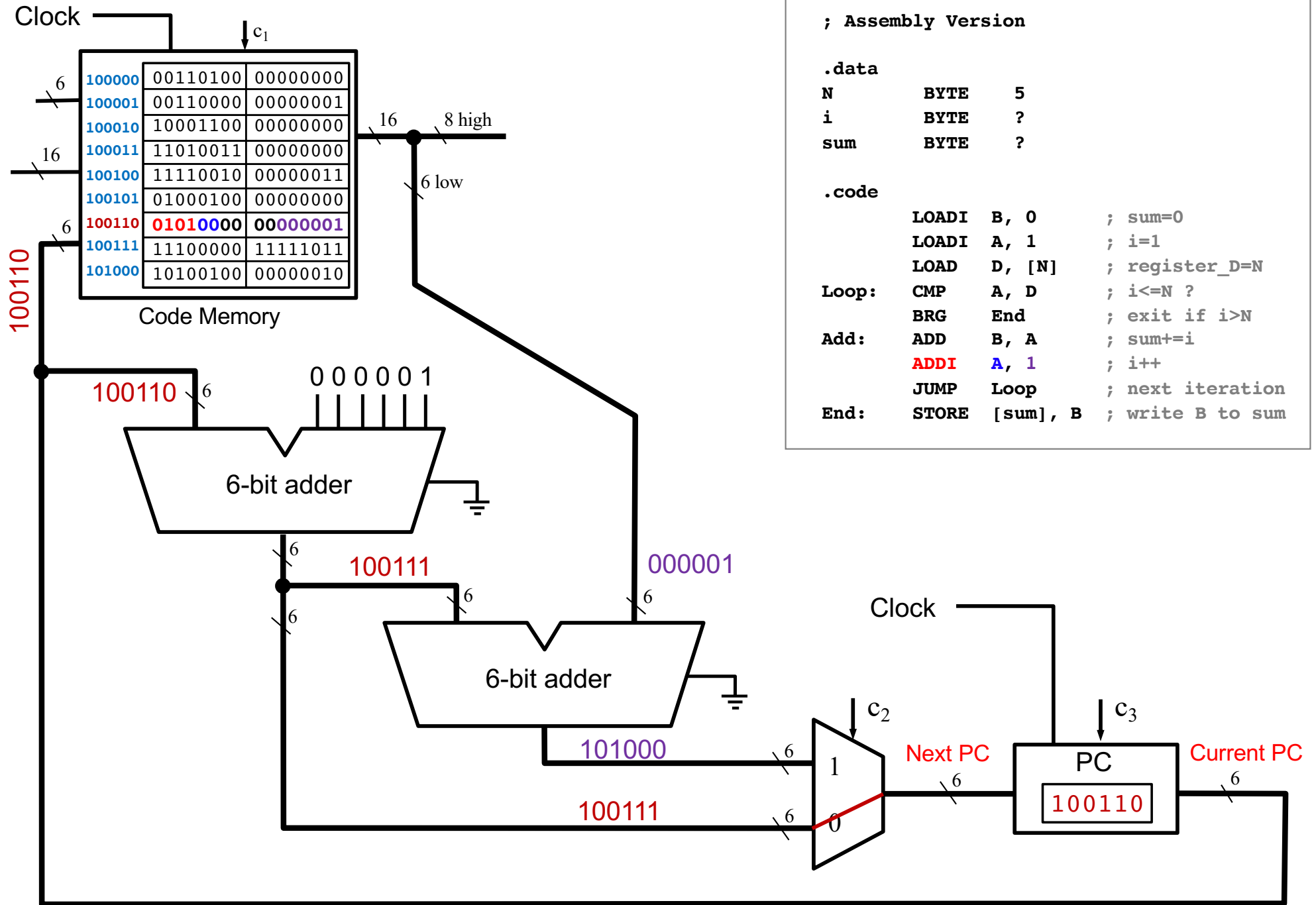


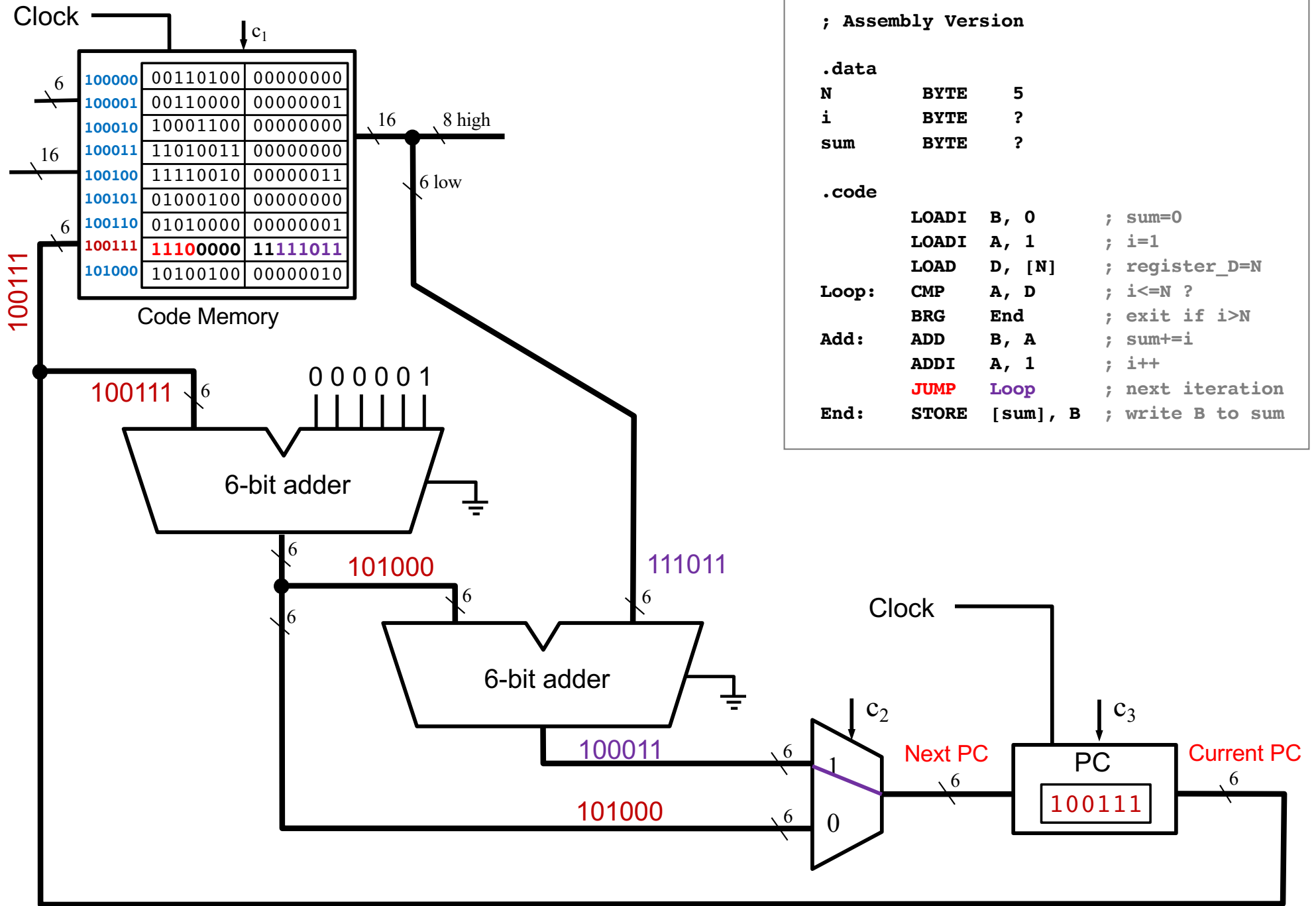


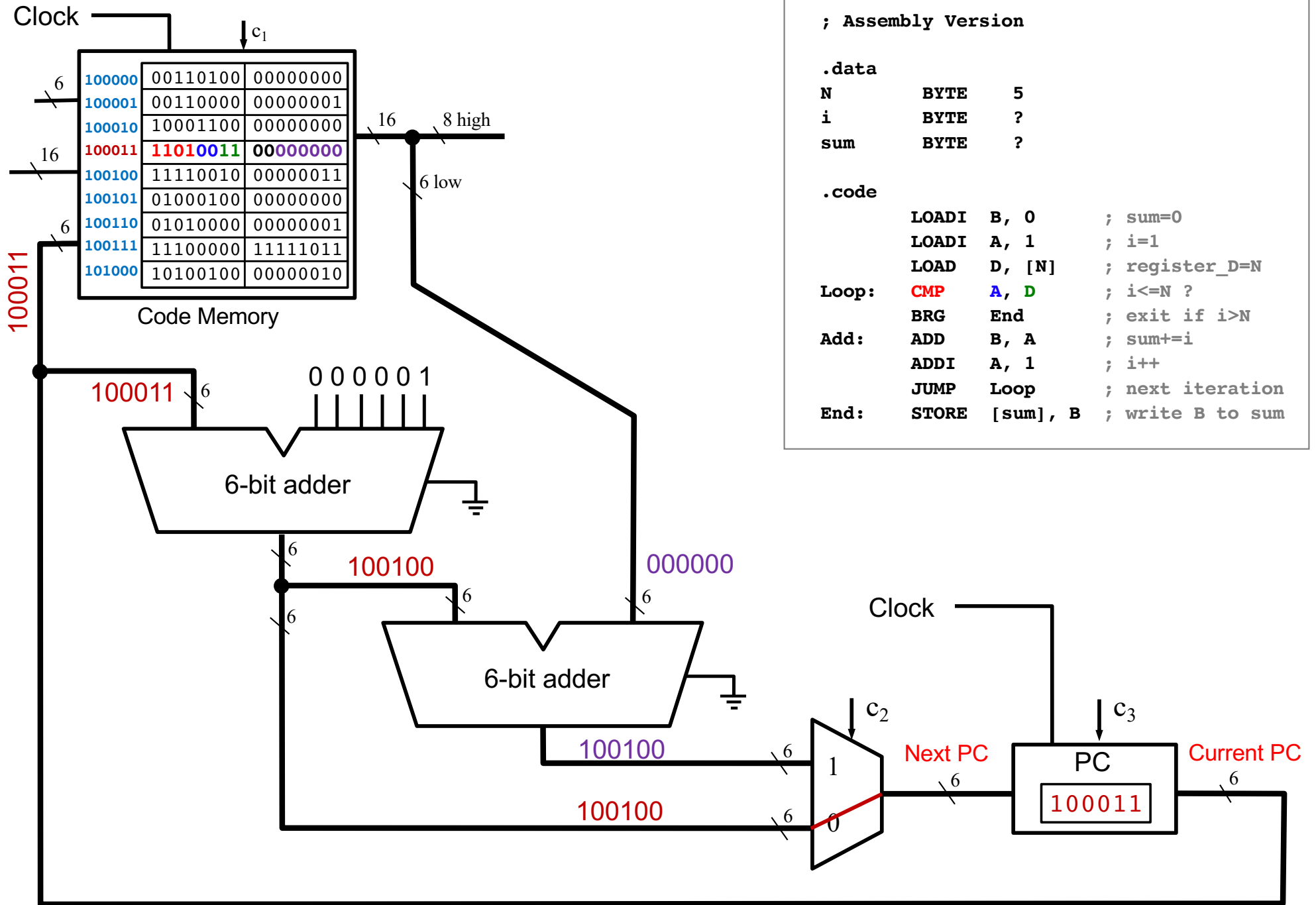




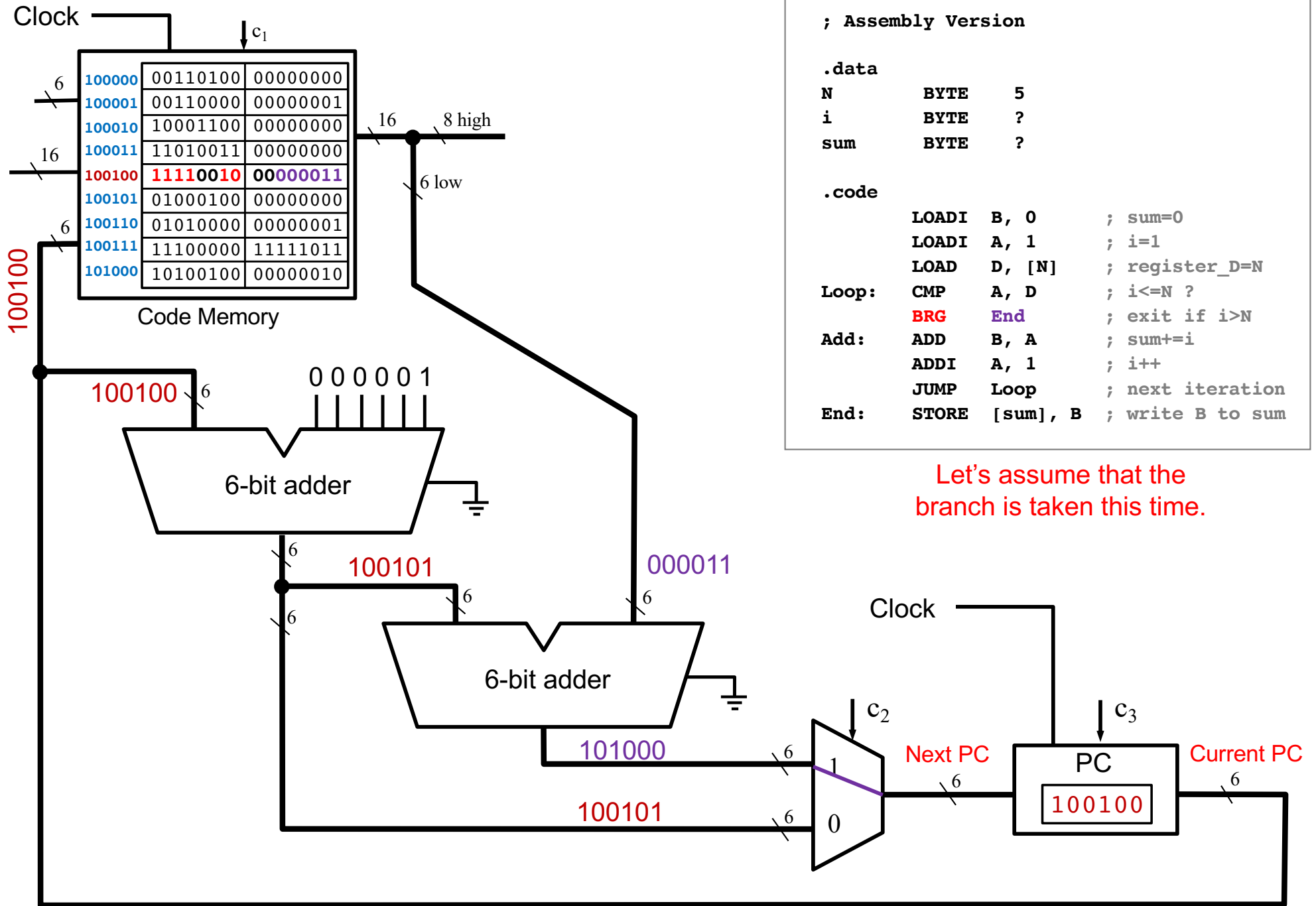


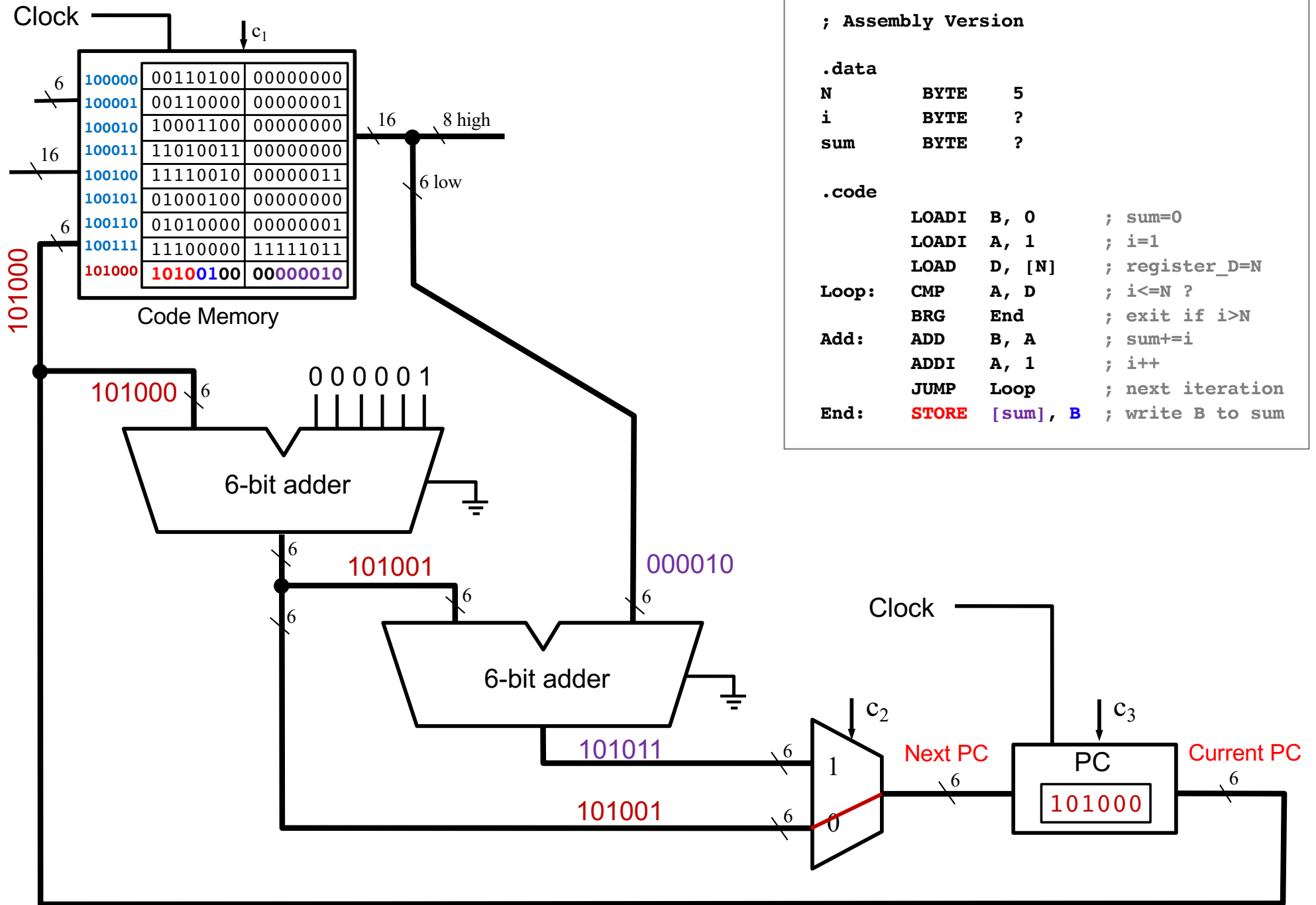


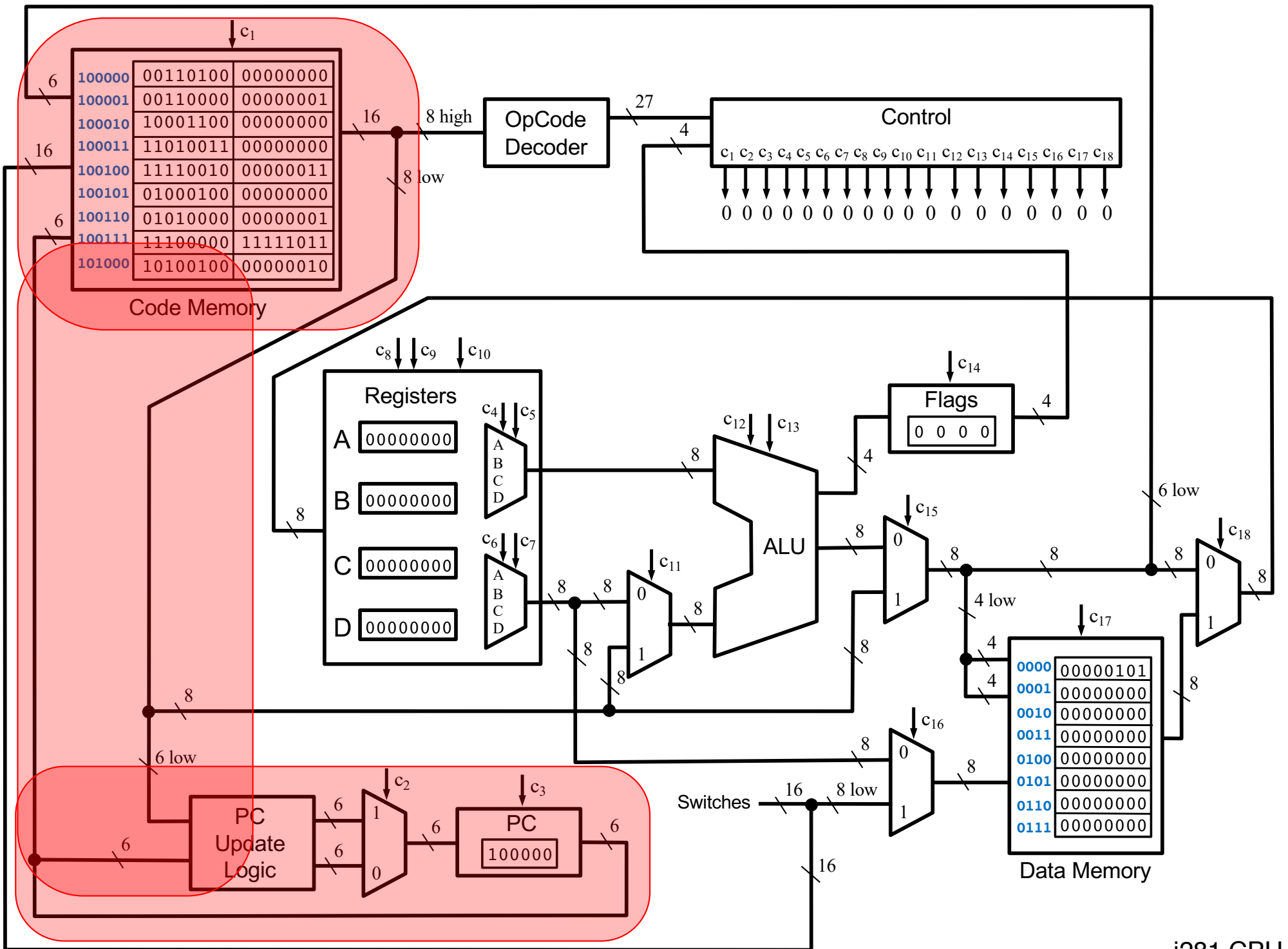


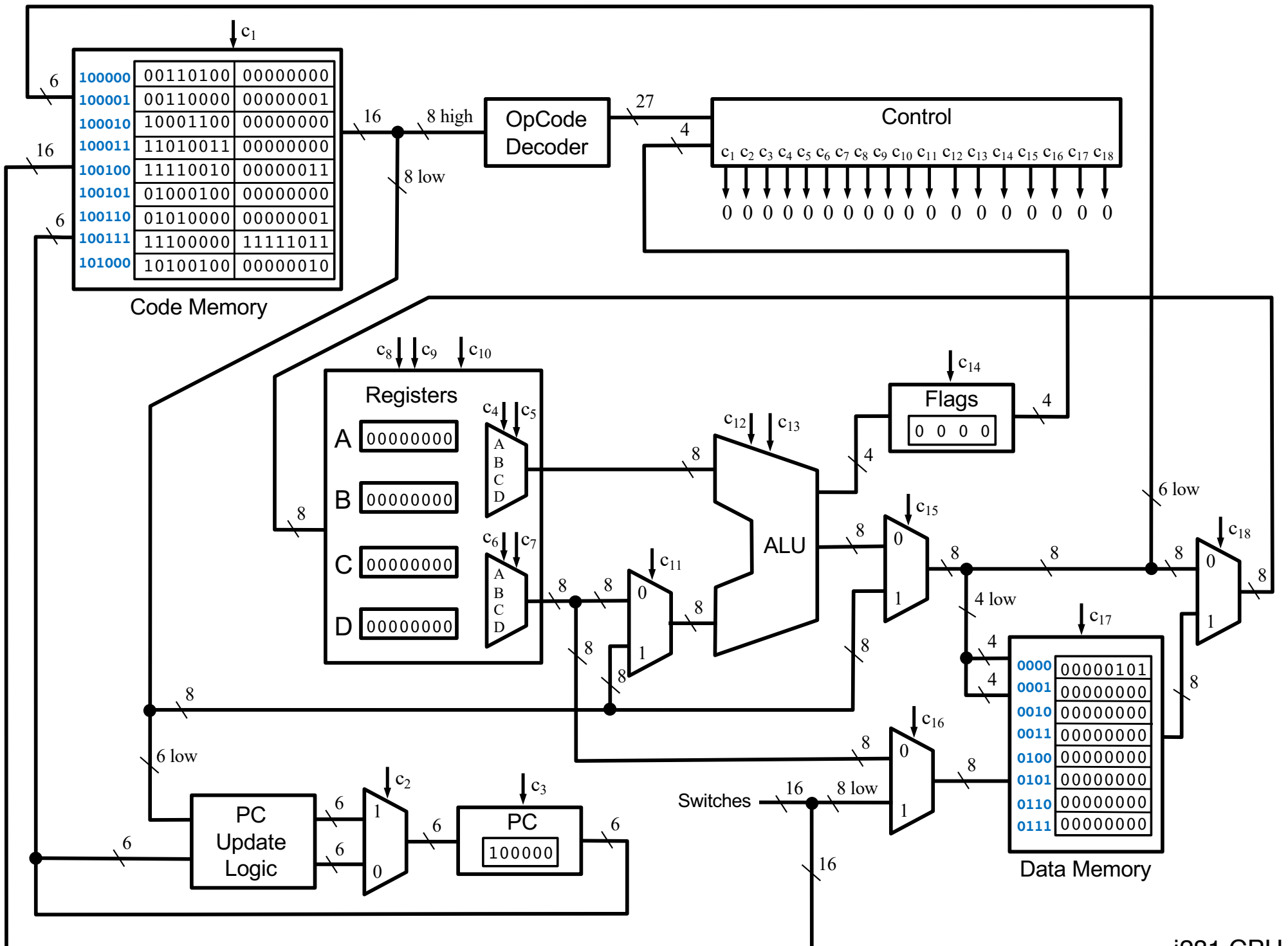


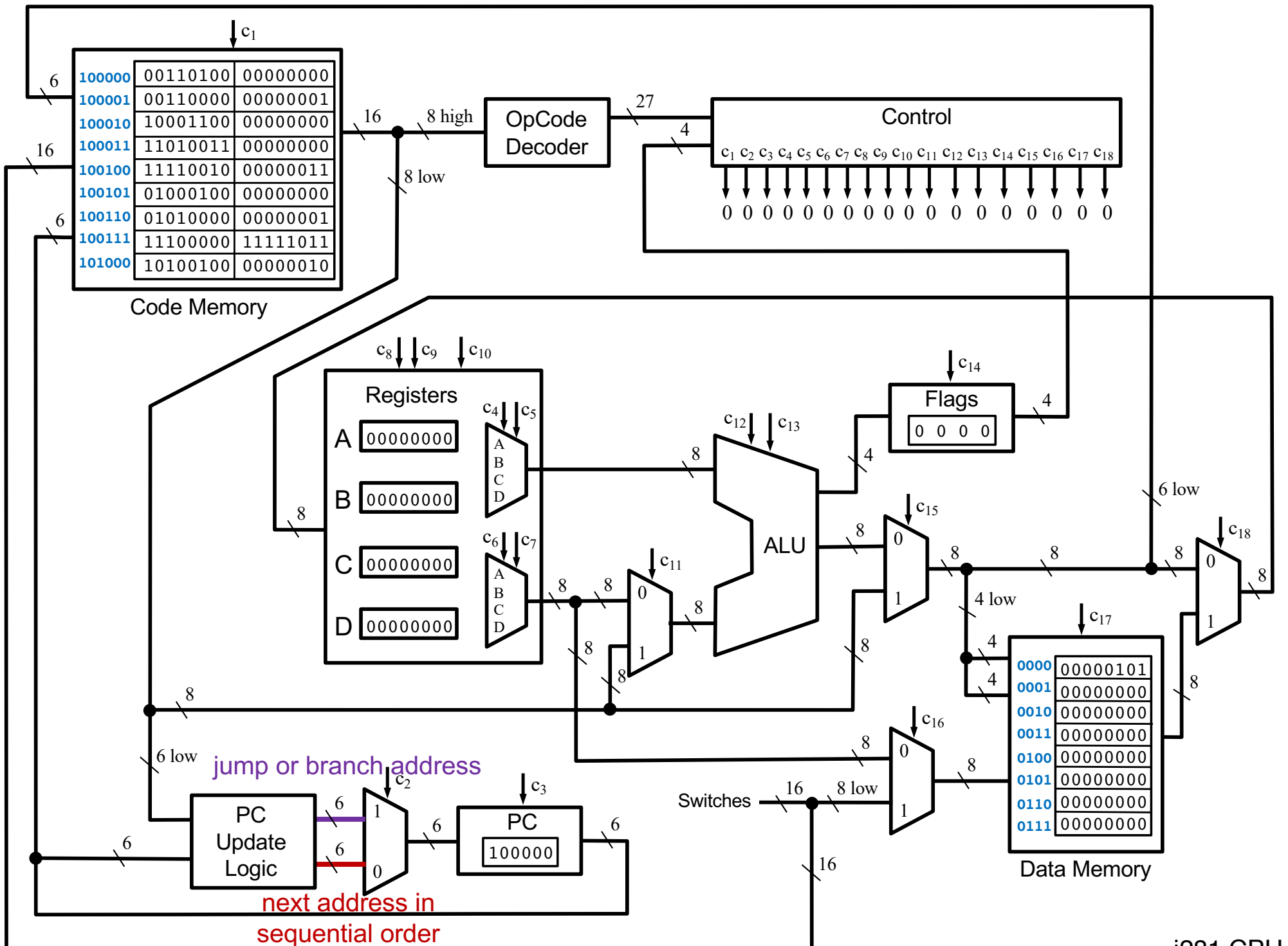












**Questions?**

**THE END**