

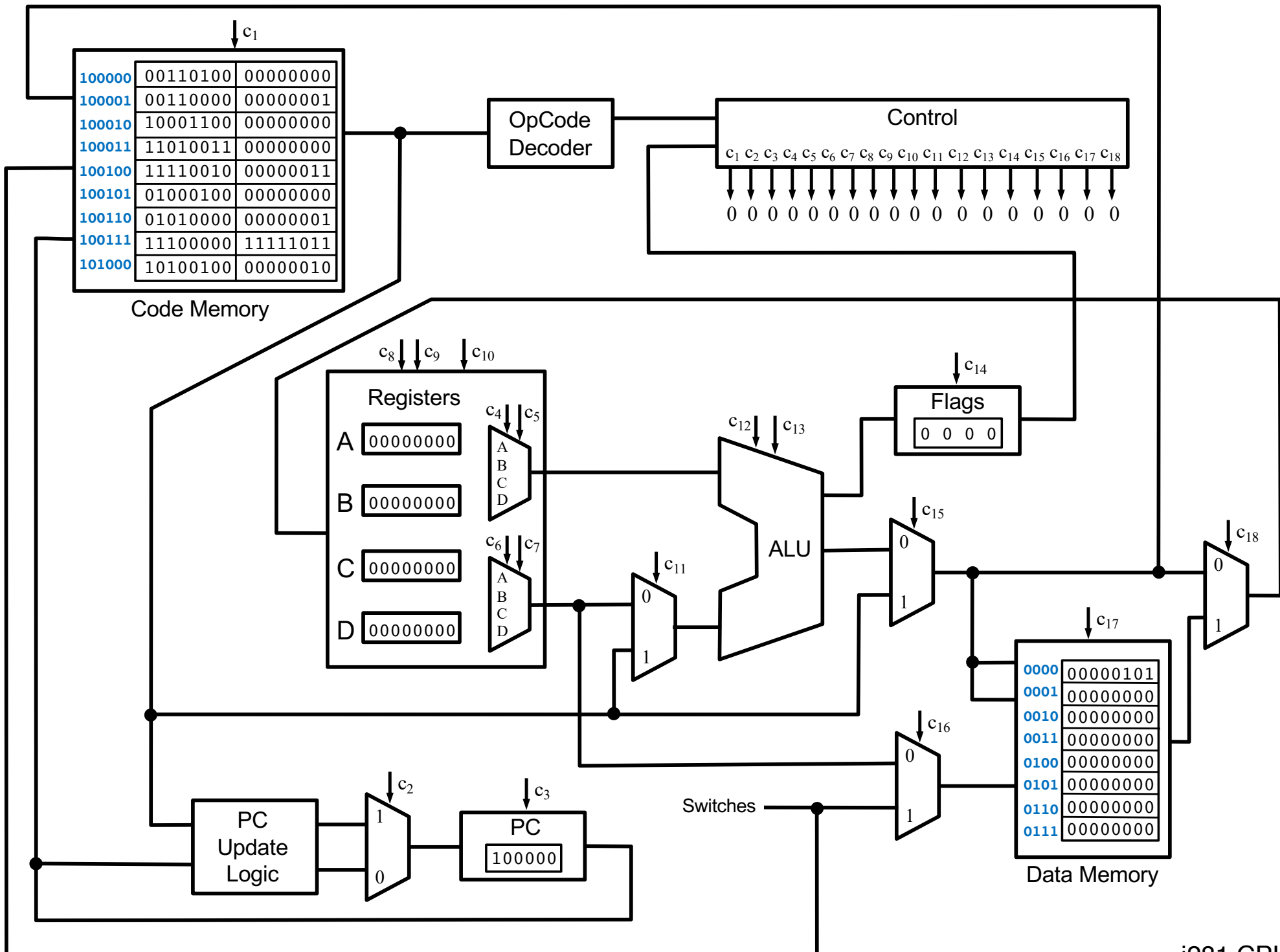
CprE 281: Digital Logic

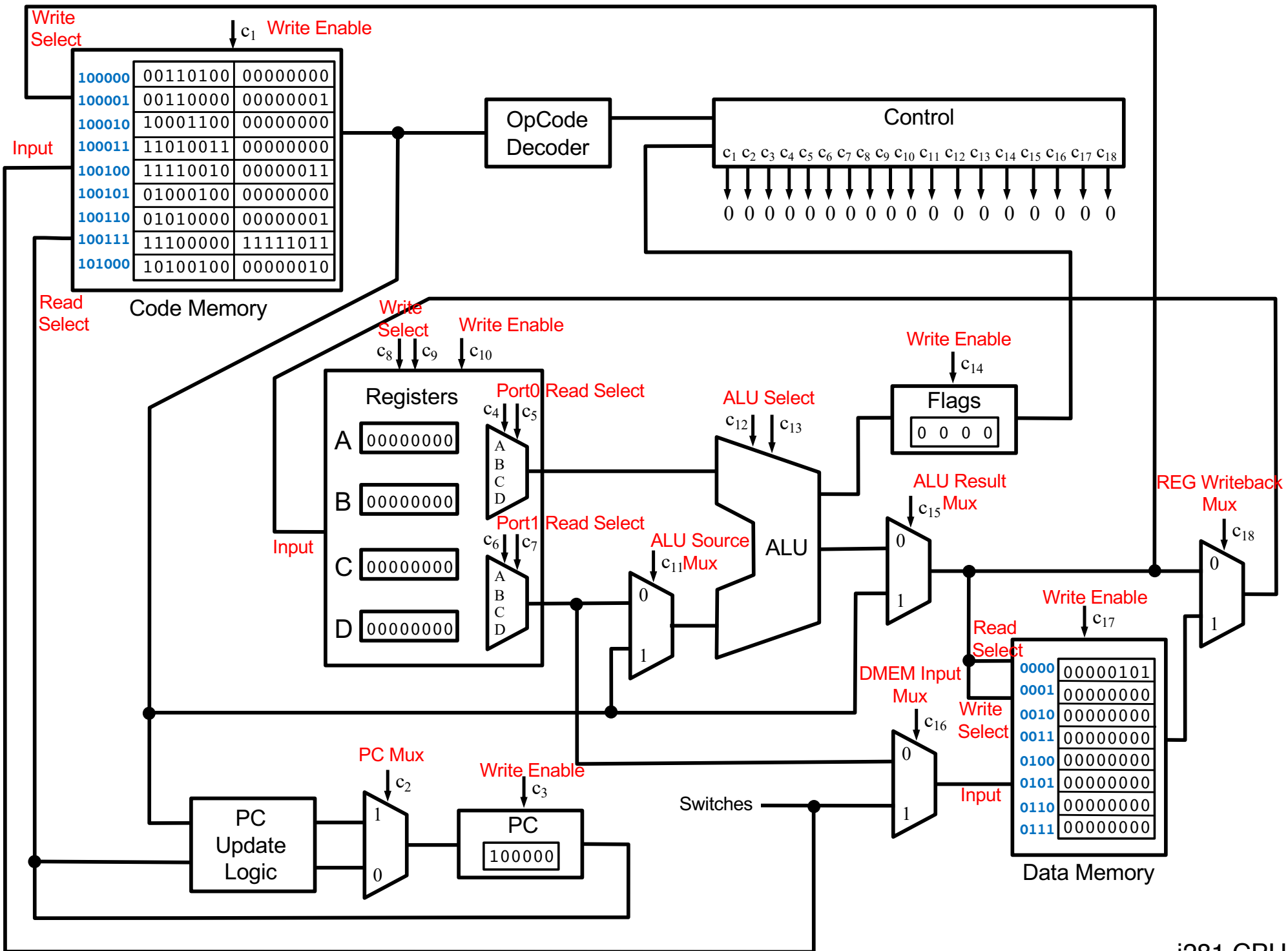
Instructor: Alexander Stoytchev

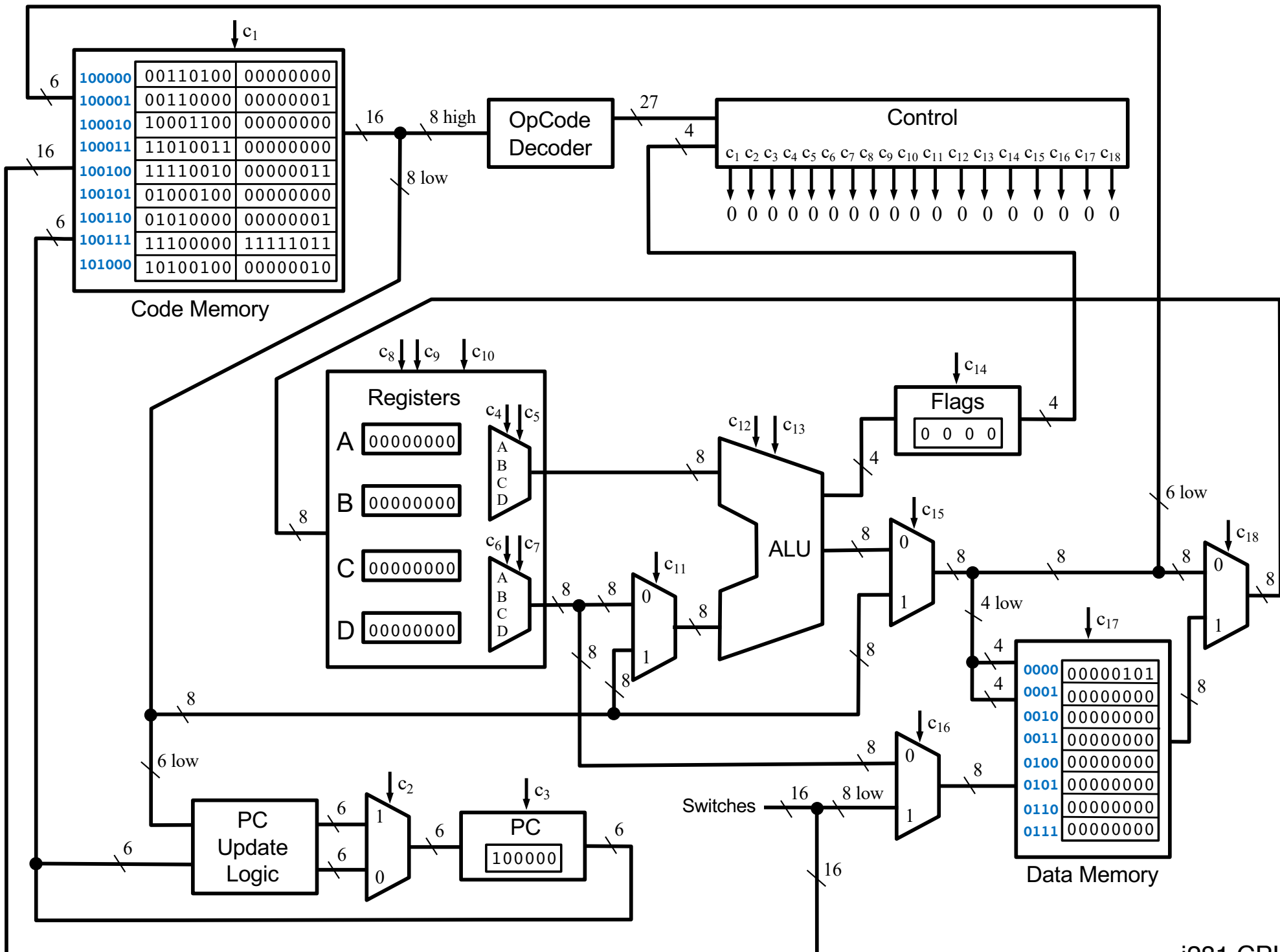
<http://www.ece.iastate.edu/~alexs/classes/>

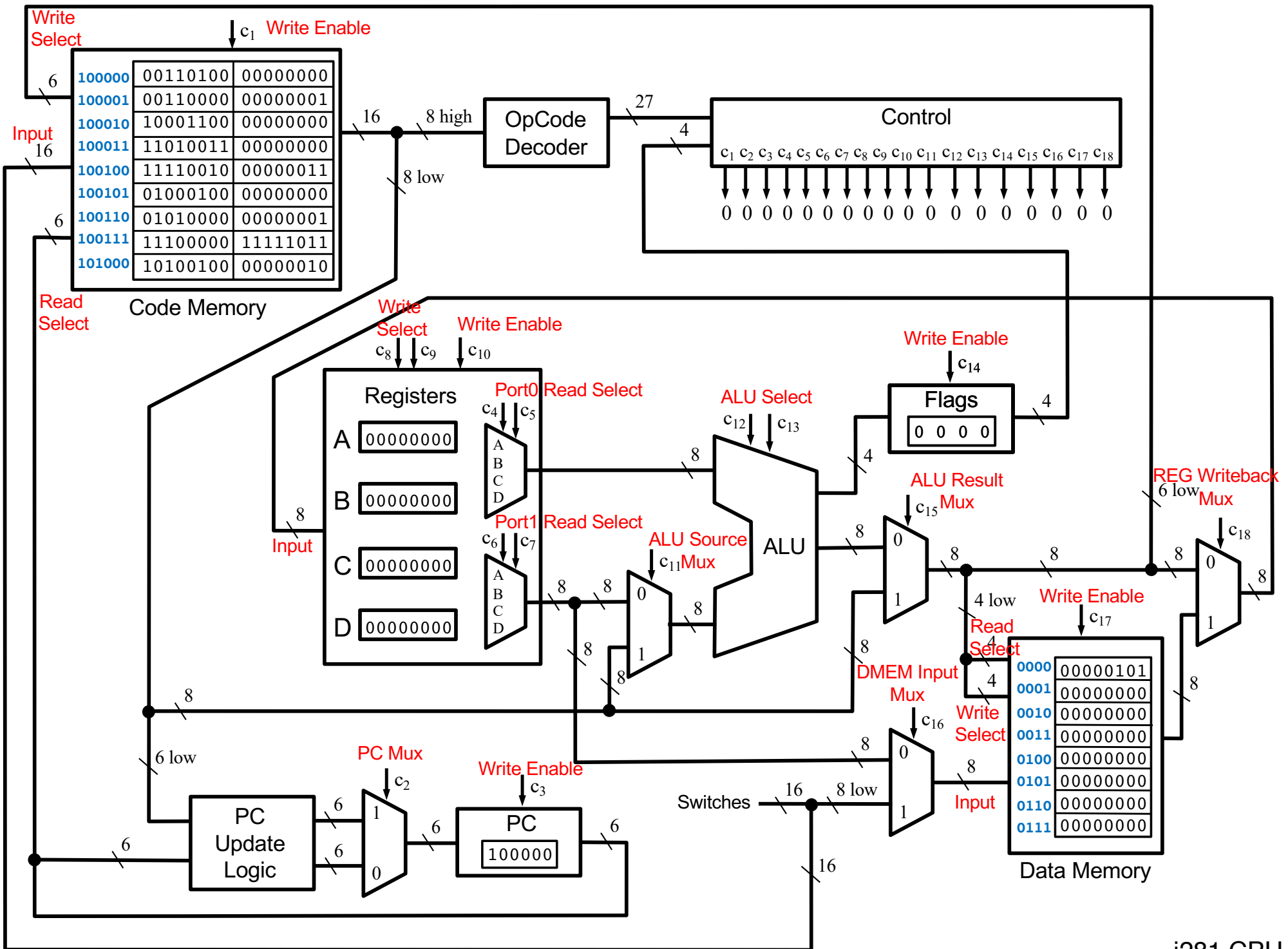
Arithmetic Logic Unit & Program Counter

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

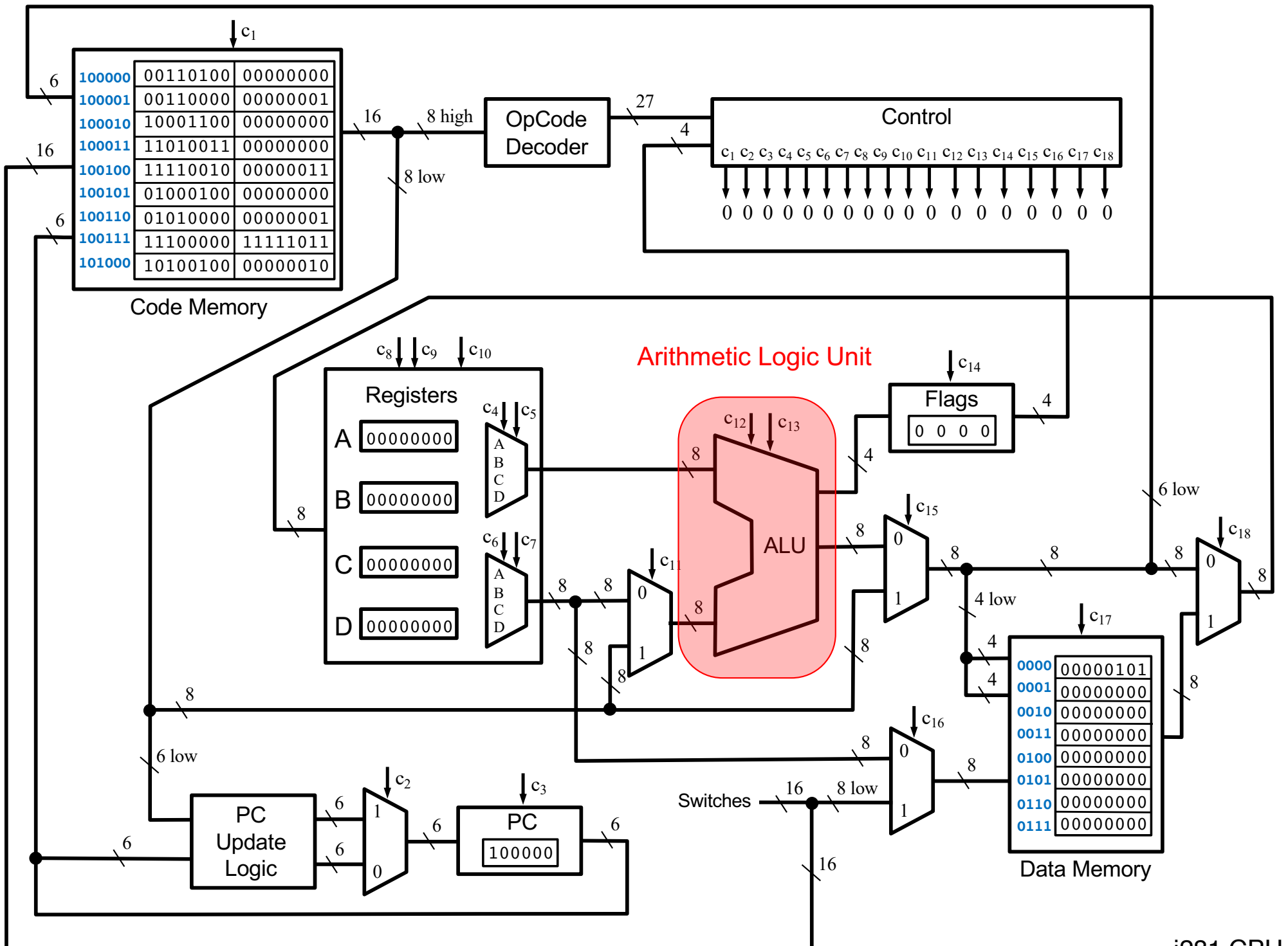


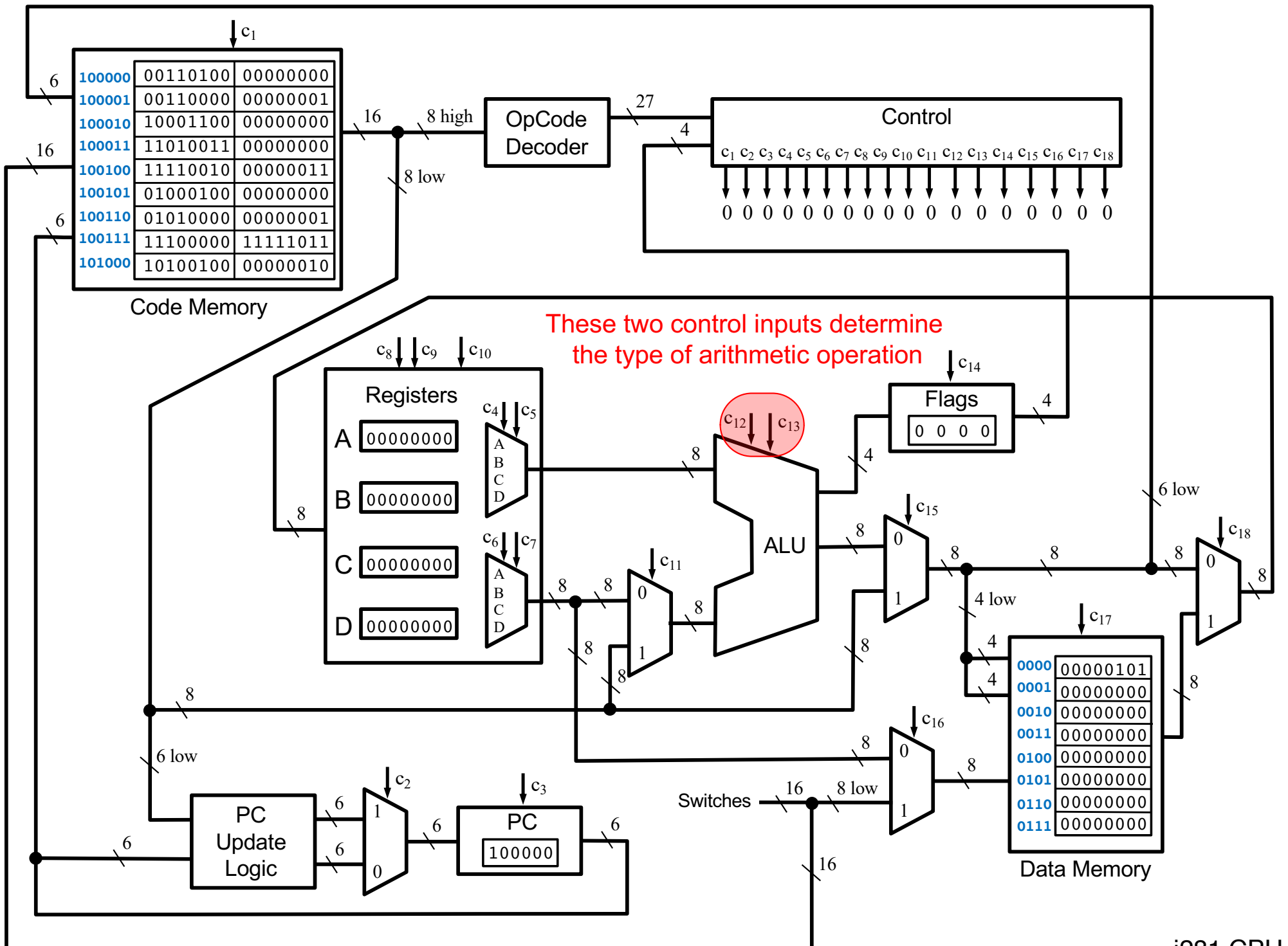






The Arithmetic Logic Unit (ALU)





This ALU Can Perform 4 Operations

ALU_SELECT1	ALU_SELECT0	Operation
0	0	SHIFTL
0	1	SHIFTR
1	0	ADD
1	1	SUB/CMP

This ALU Can Perform 4 Operations

Names of these
control lines

C₁₂

C₁₃

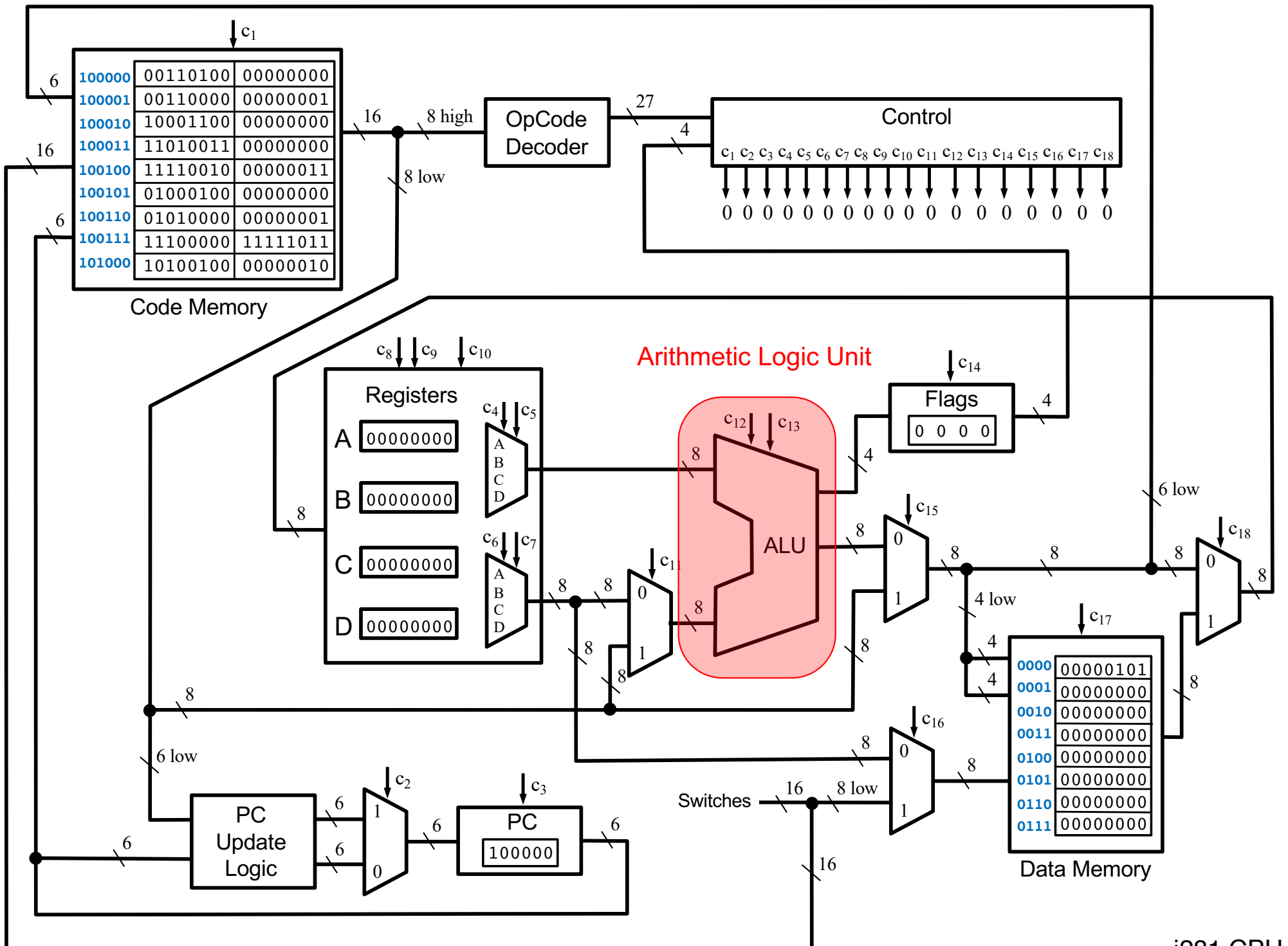
ALU_SELECT1	ALU_SELECT0	Operation
0	0	SHIFTL
0	1	SHIFTR
1	0	ADD
1	1	SUB/CMP

This ALU Can Perform 4 Operations

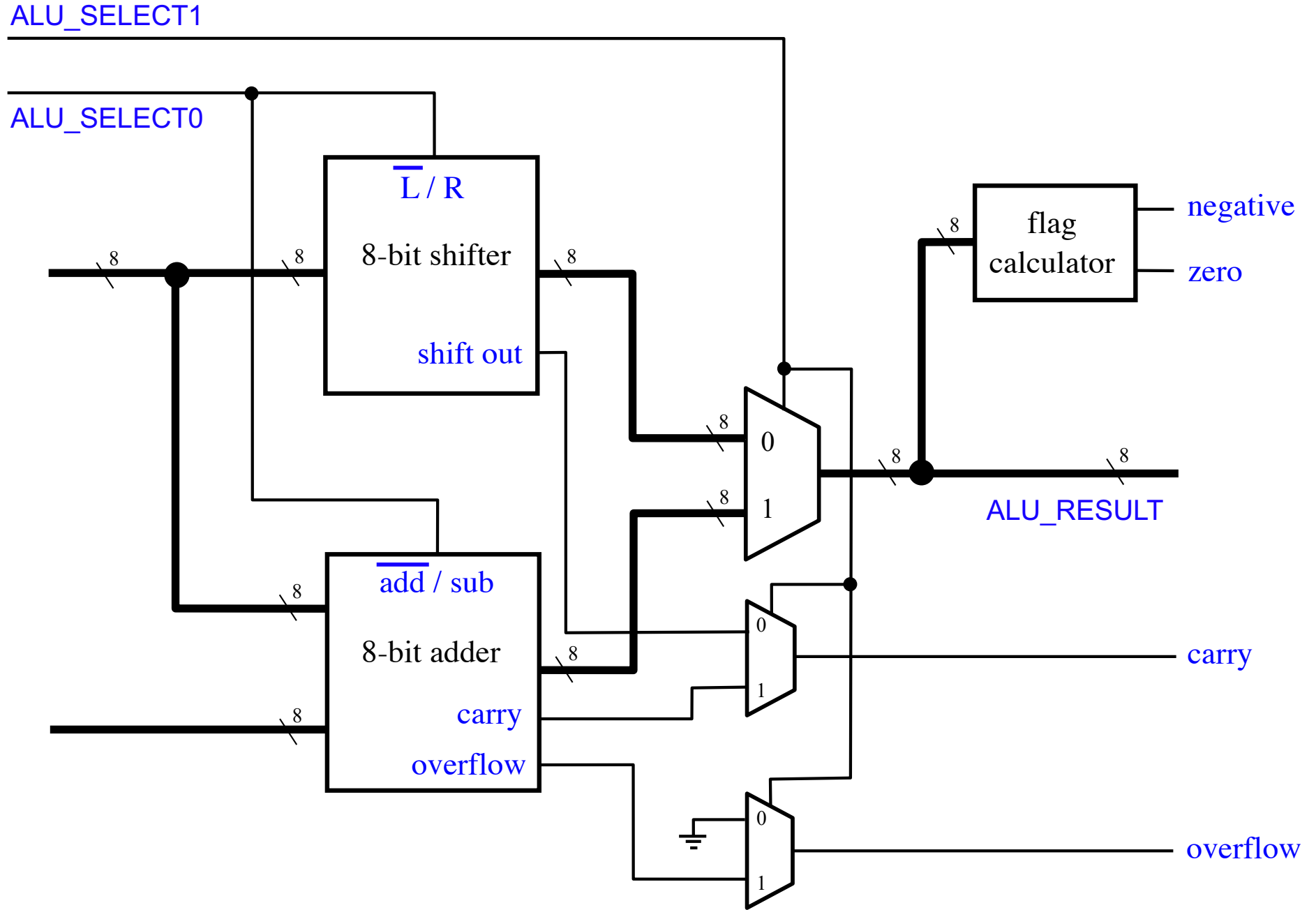
C_{12}	C_{13}	
ALU_SELECT1	ALU_SELECT0	Operation
0	0	SHIFTL
0	1	SHIFTR
1	0	ADD
1	1	SUB/CMP

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

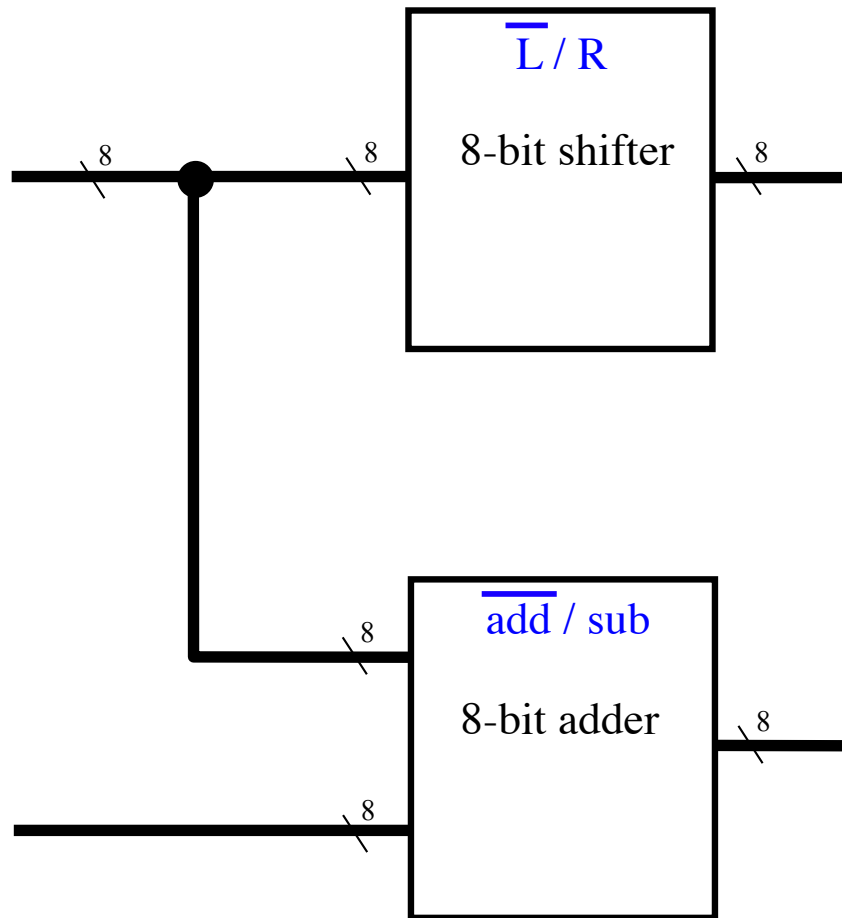
The difference is that CMP does not write back the result of the subtraction to the registers. Only the side effect through the flags remains.



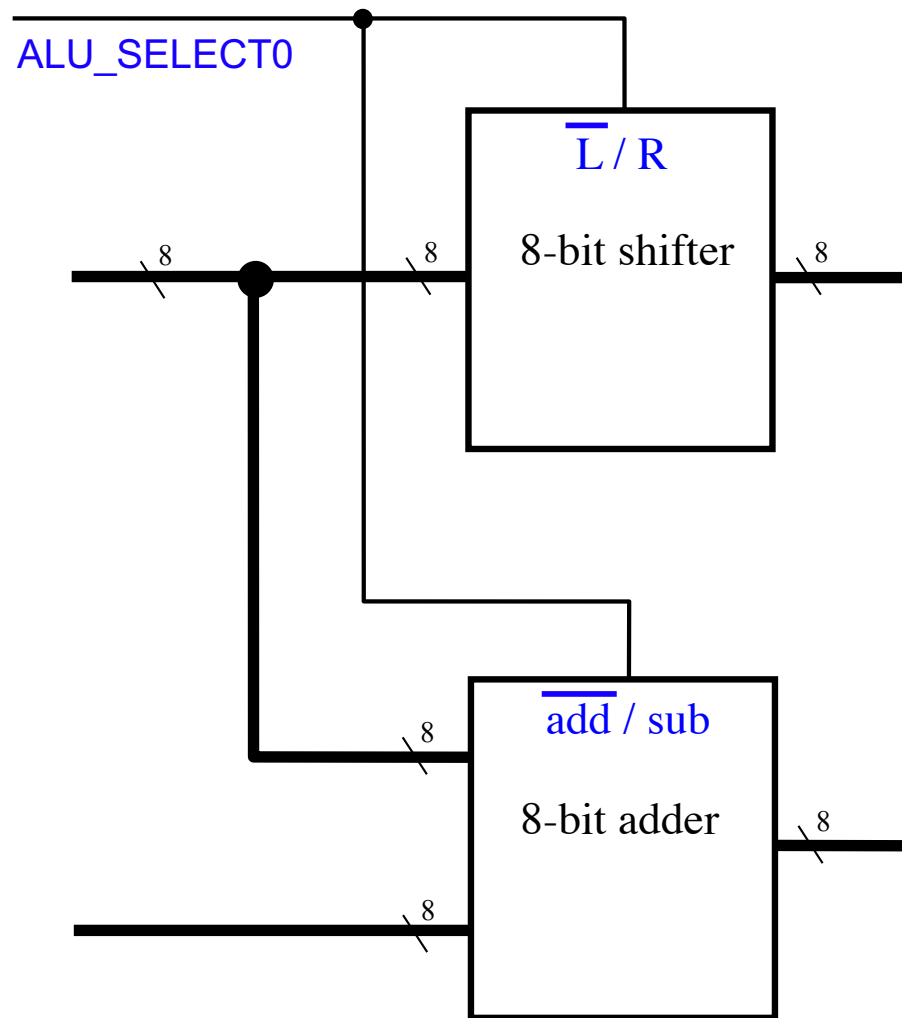
The ALU



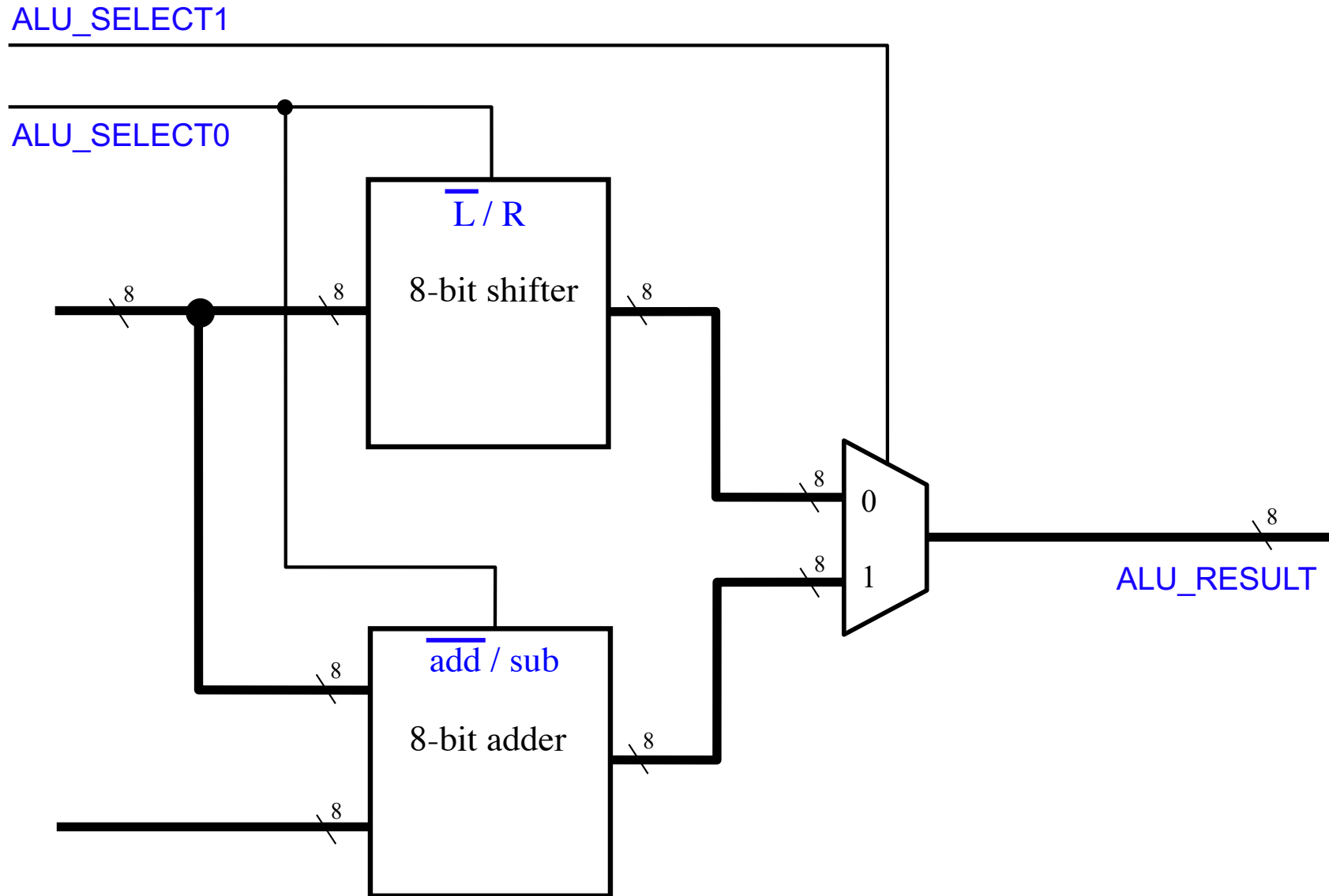
The ALU



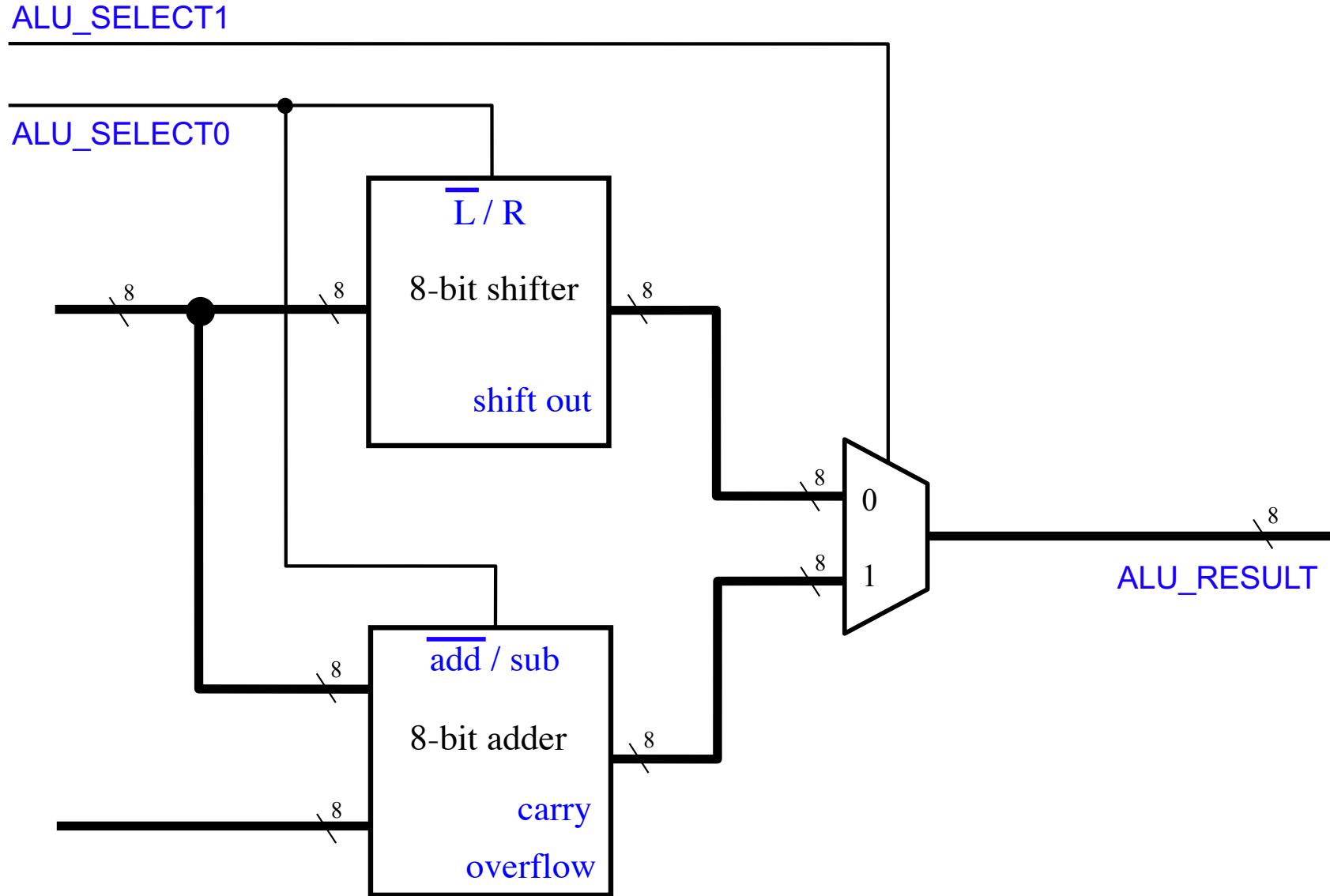
The ALU



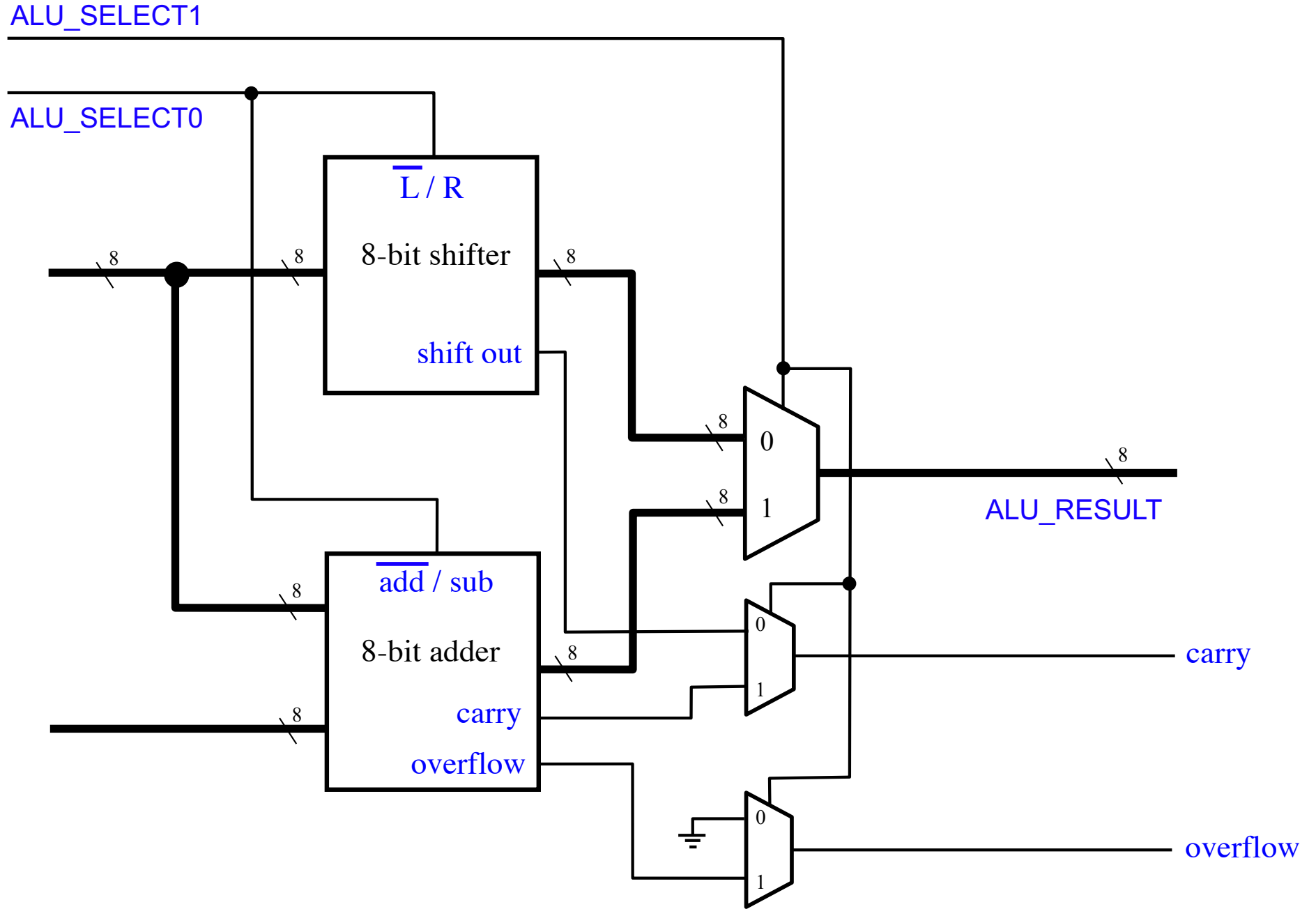
The ALU



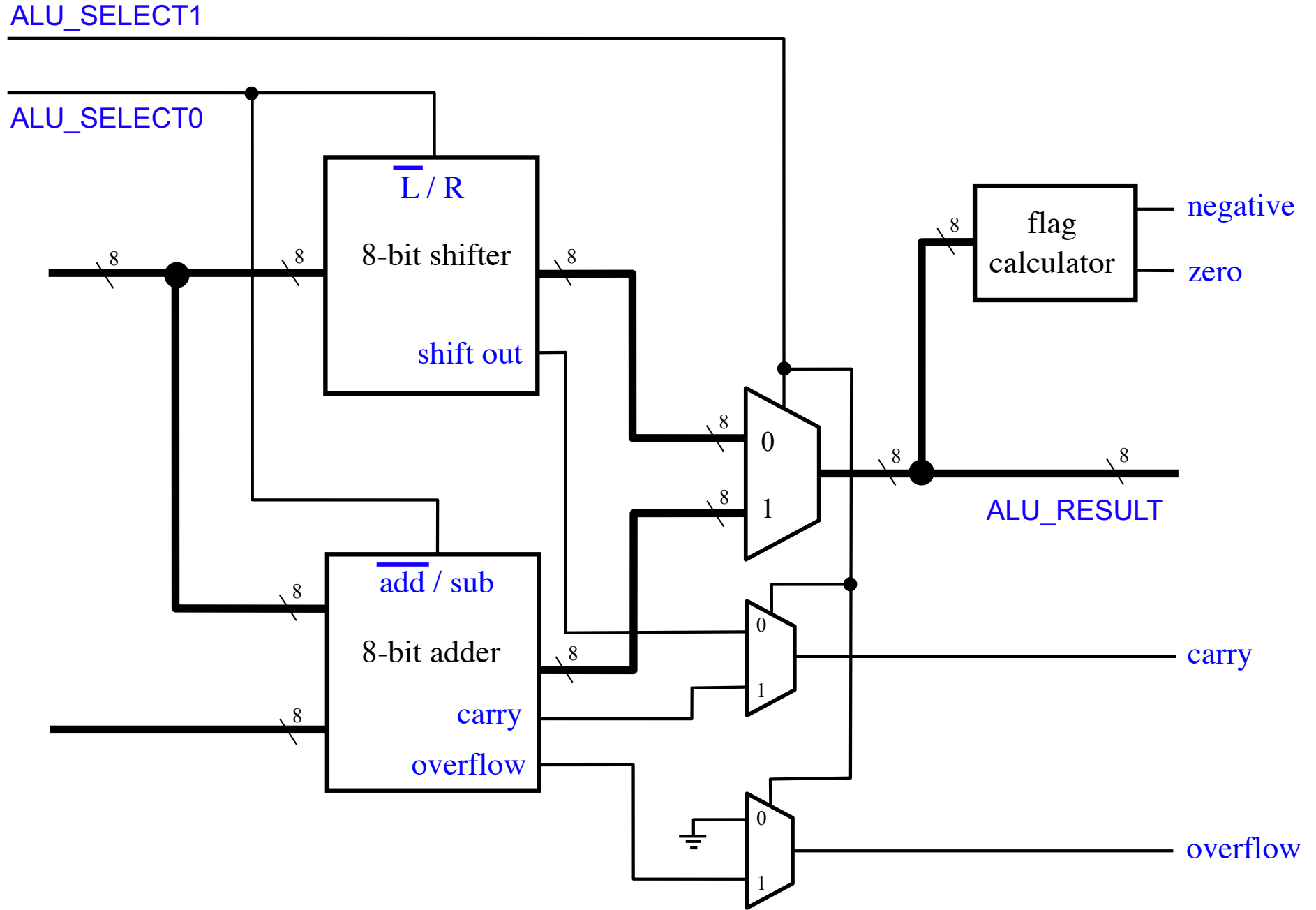
The ALU



The ALU



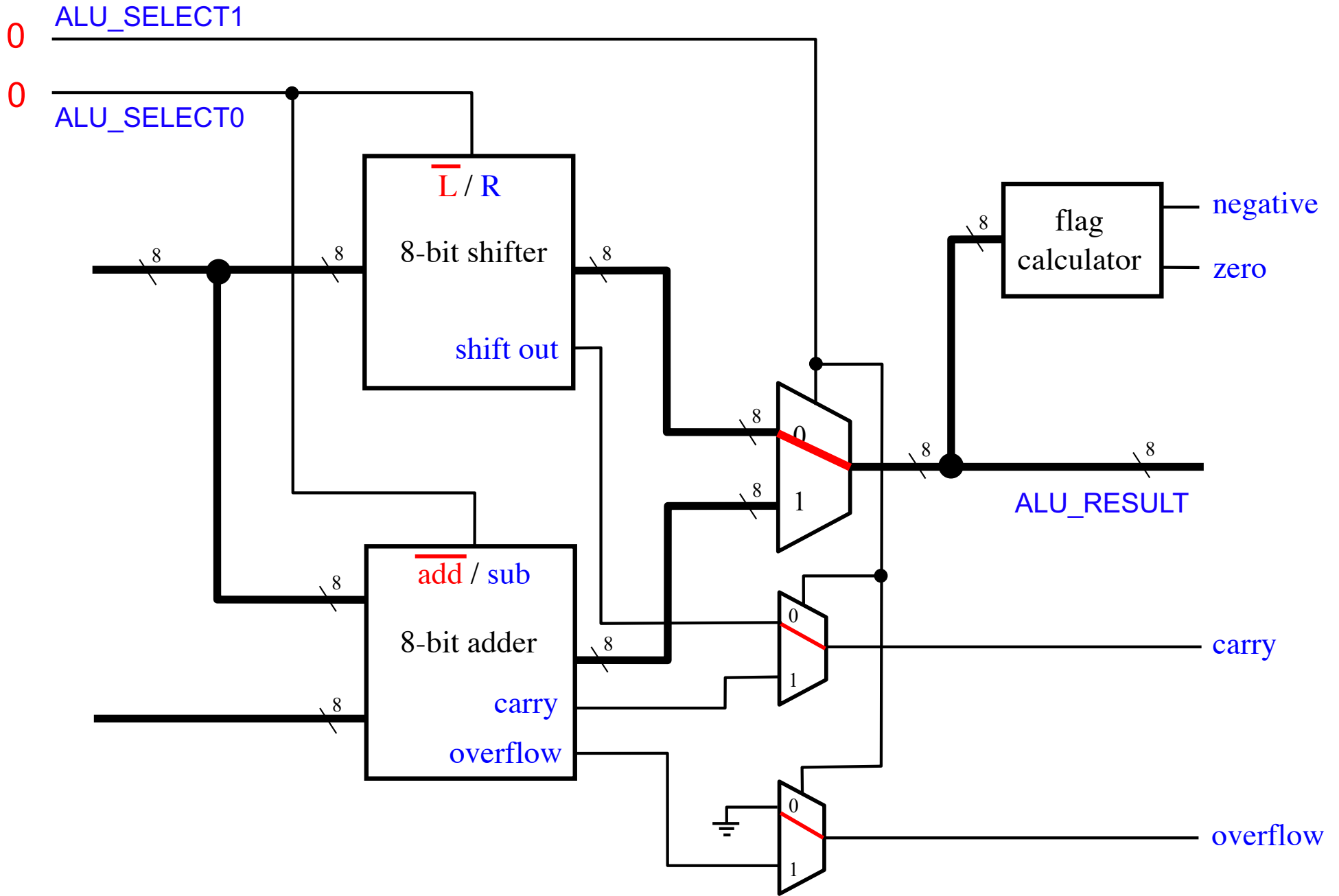
The ALU



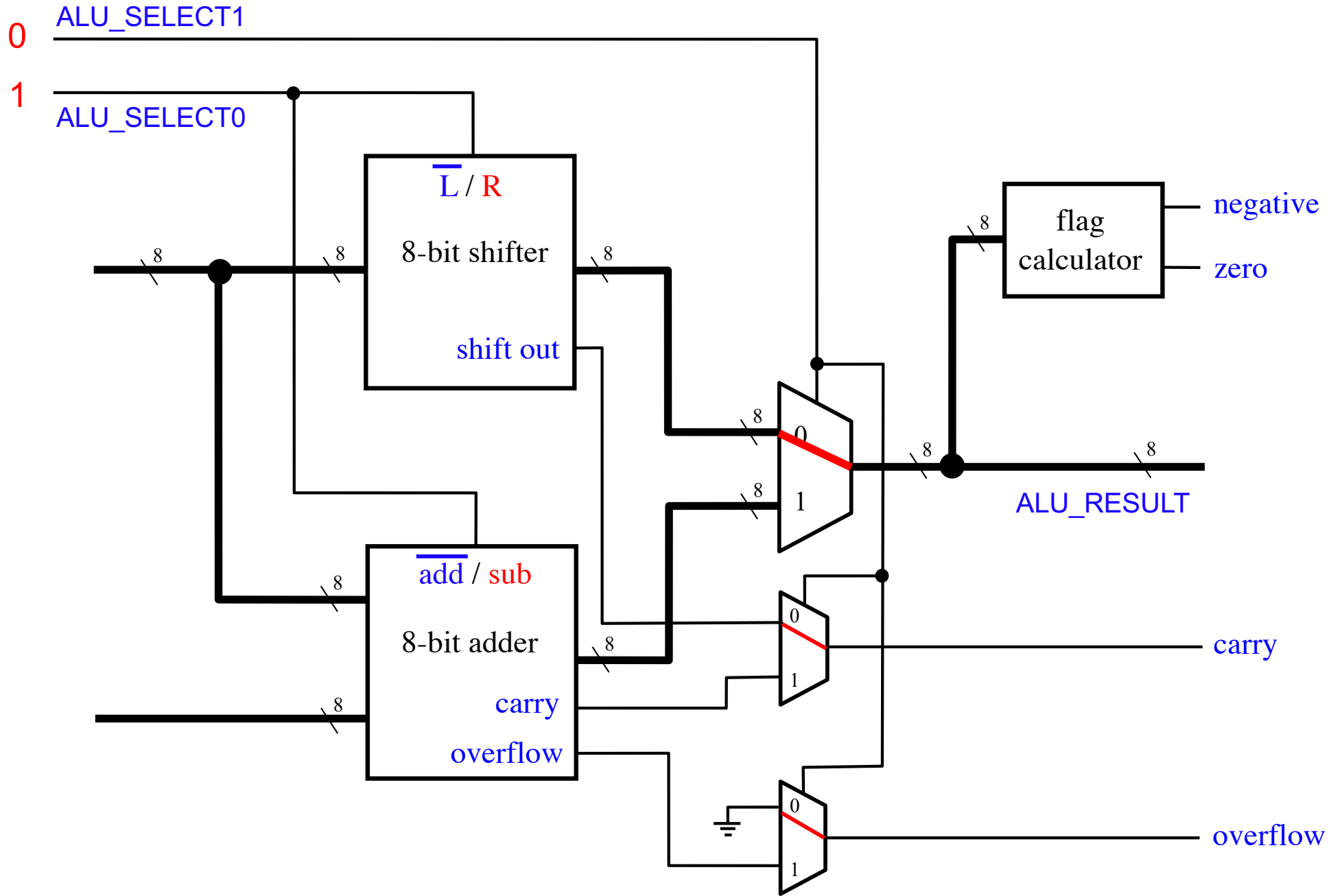
This ALU Can Perform 4 Operations

ALU_SELECT1	ALU_SELECT0	Operation
0	0	SHIFTL
0	1	SHIFTR
1	0	ADD
1	1	SUB/CMP

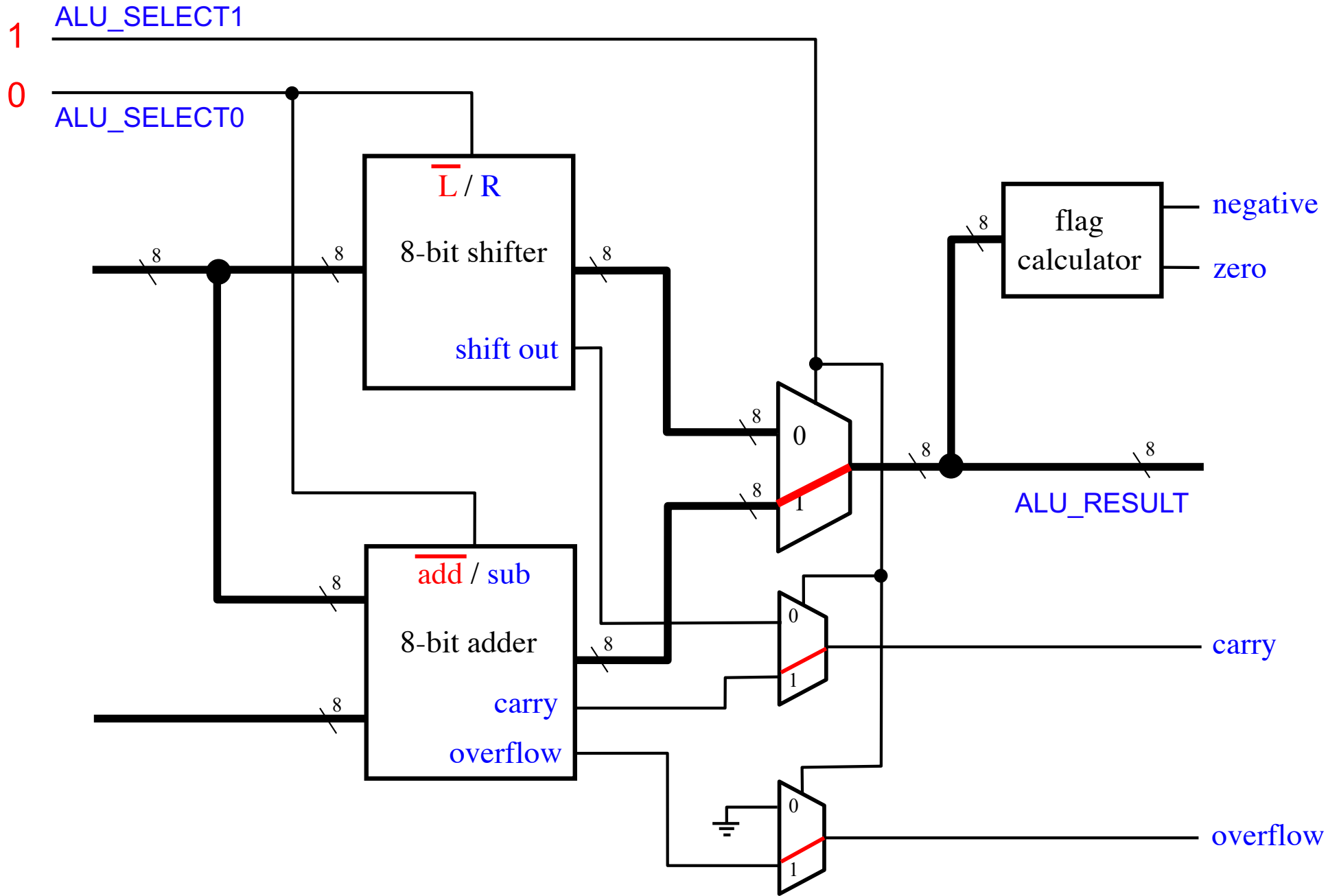
SHIFTL



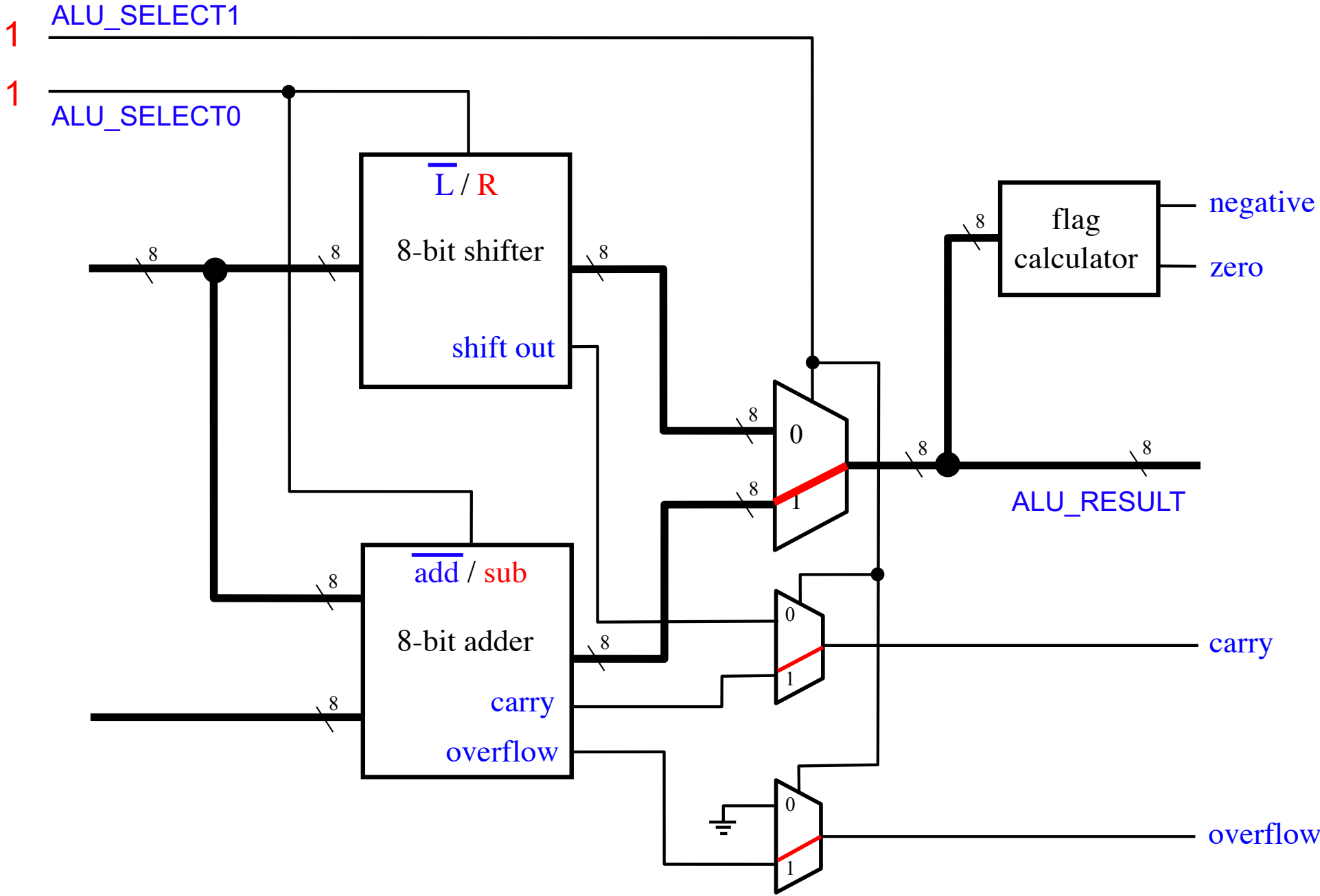
SHIFTR



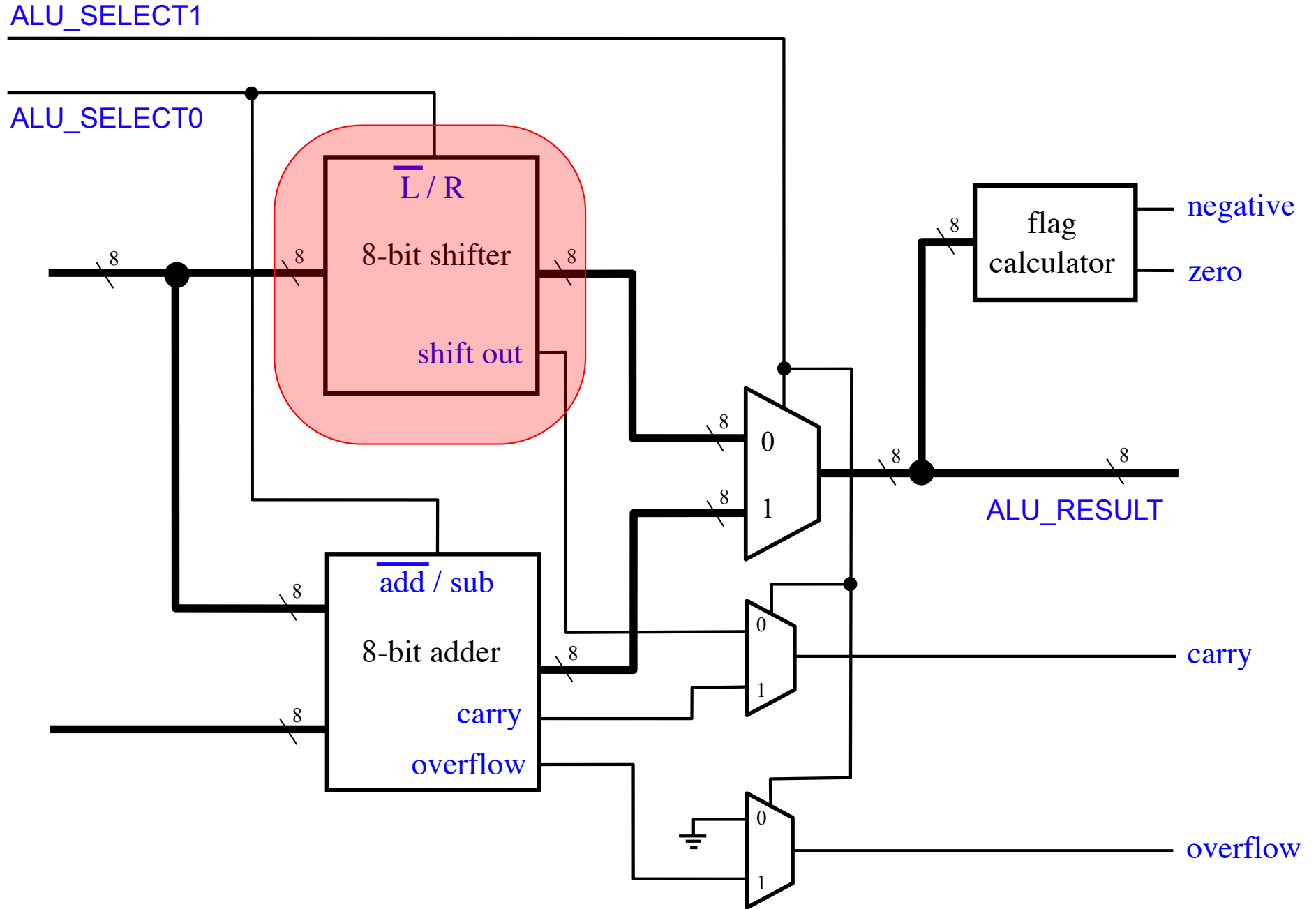
ADD



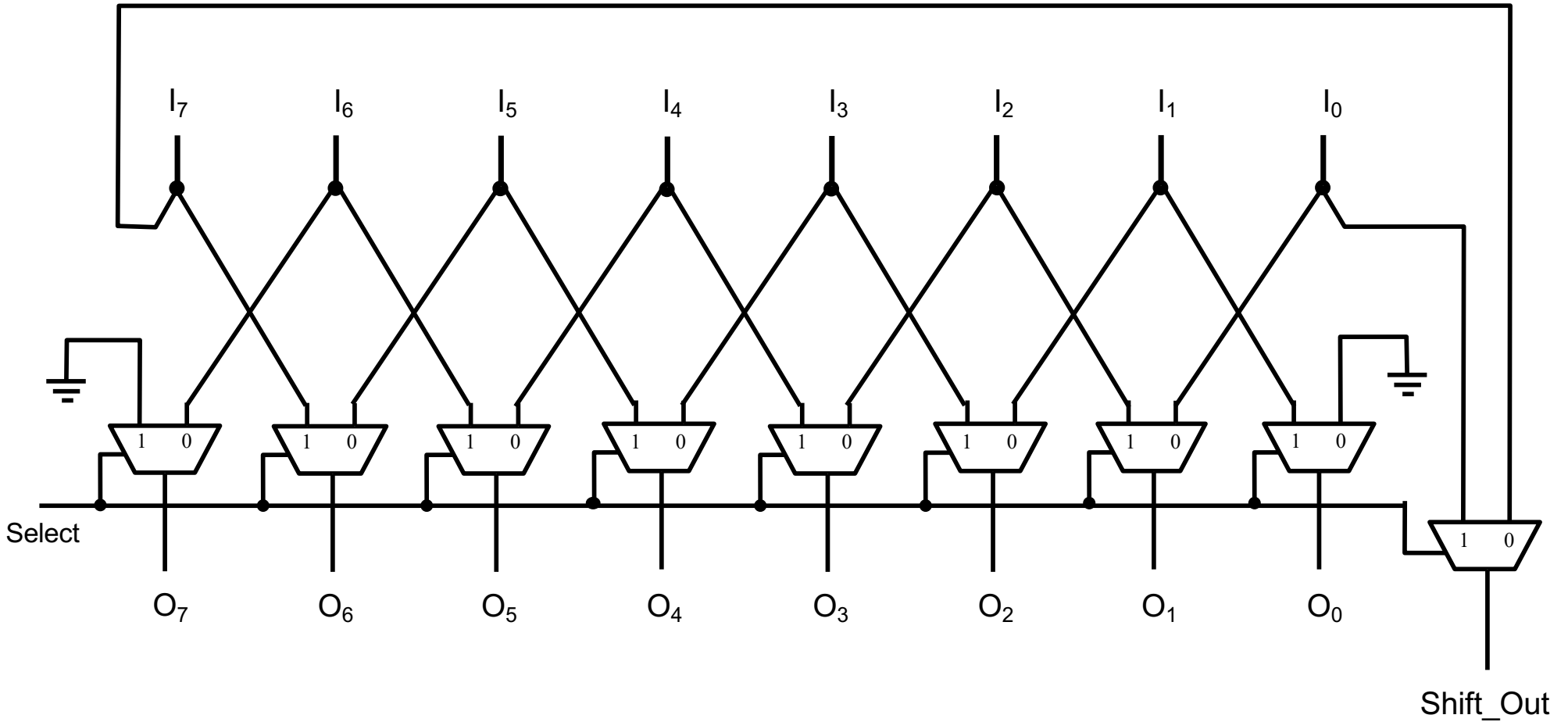
SUB / CMP

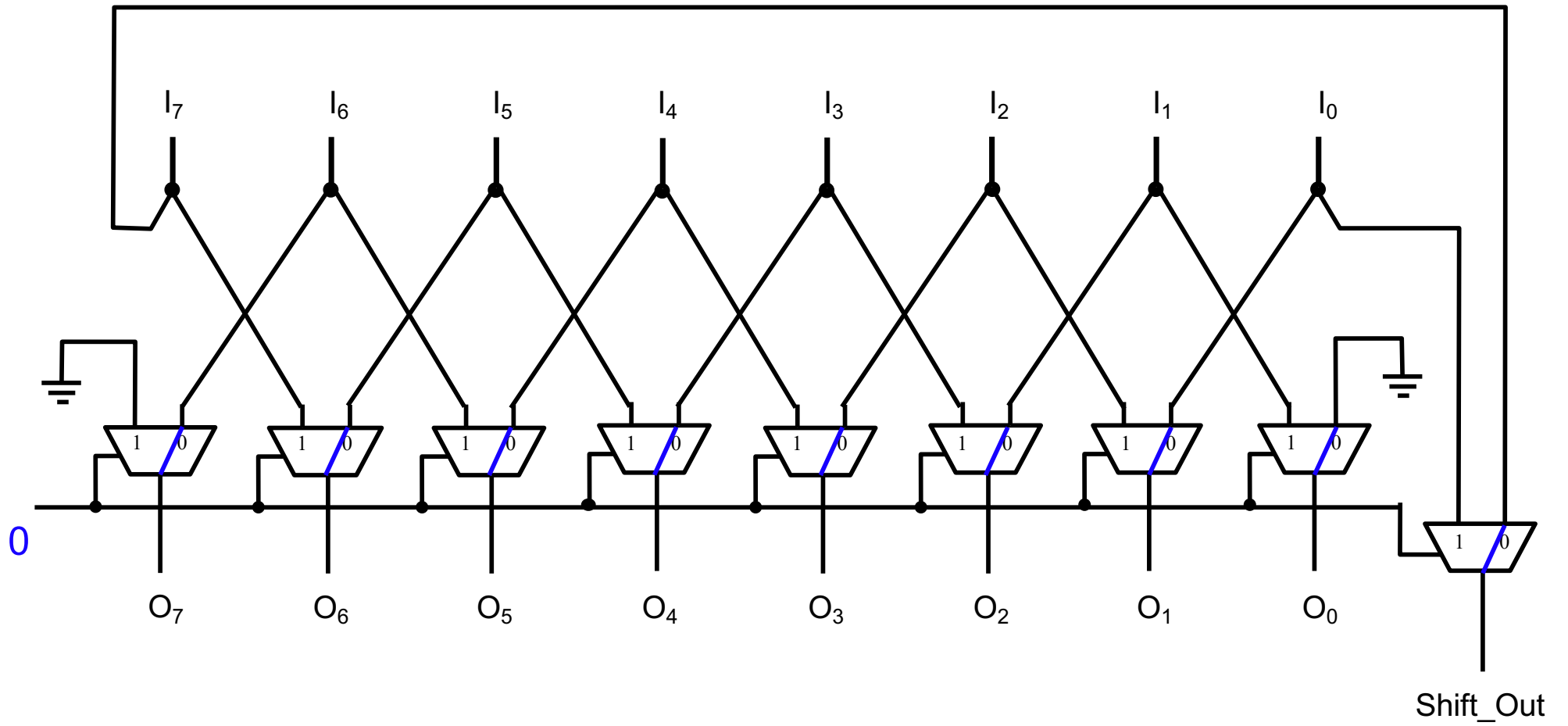


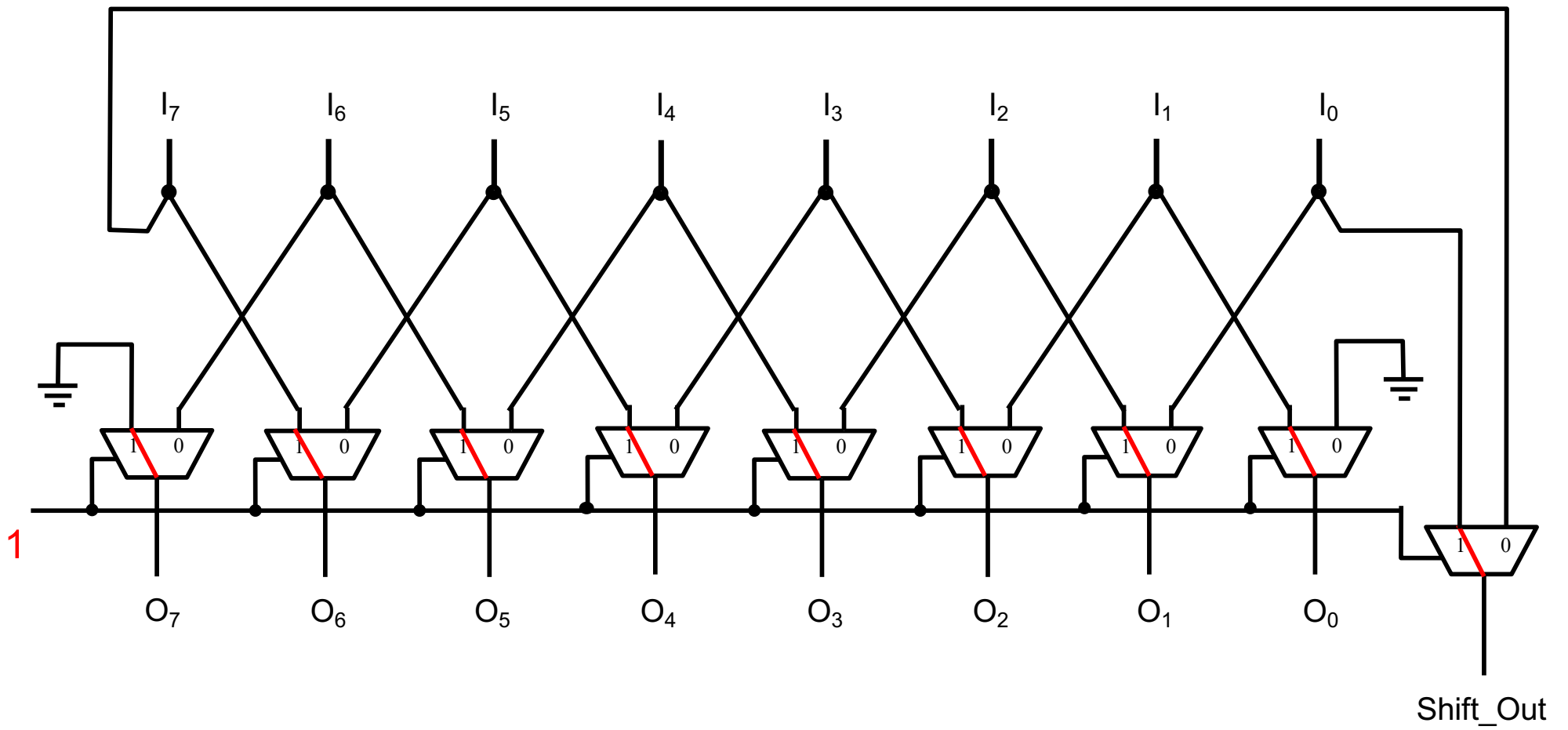
The Shifter Circuit



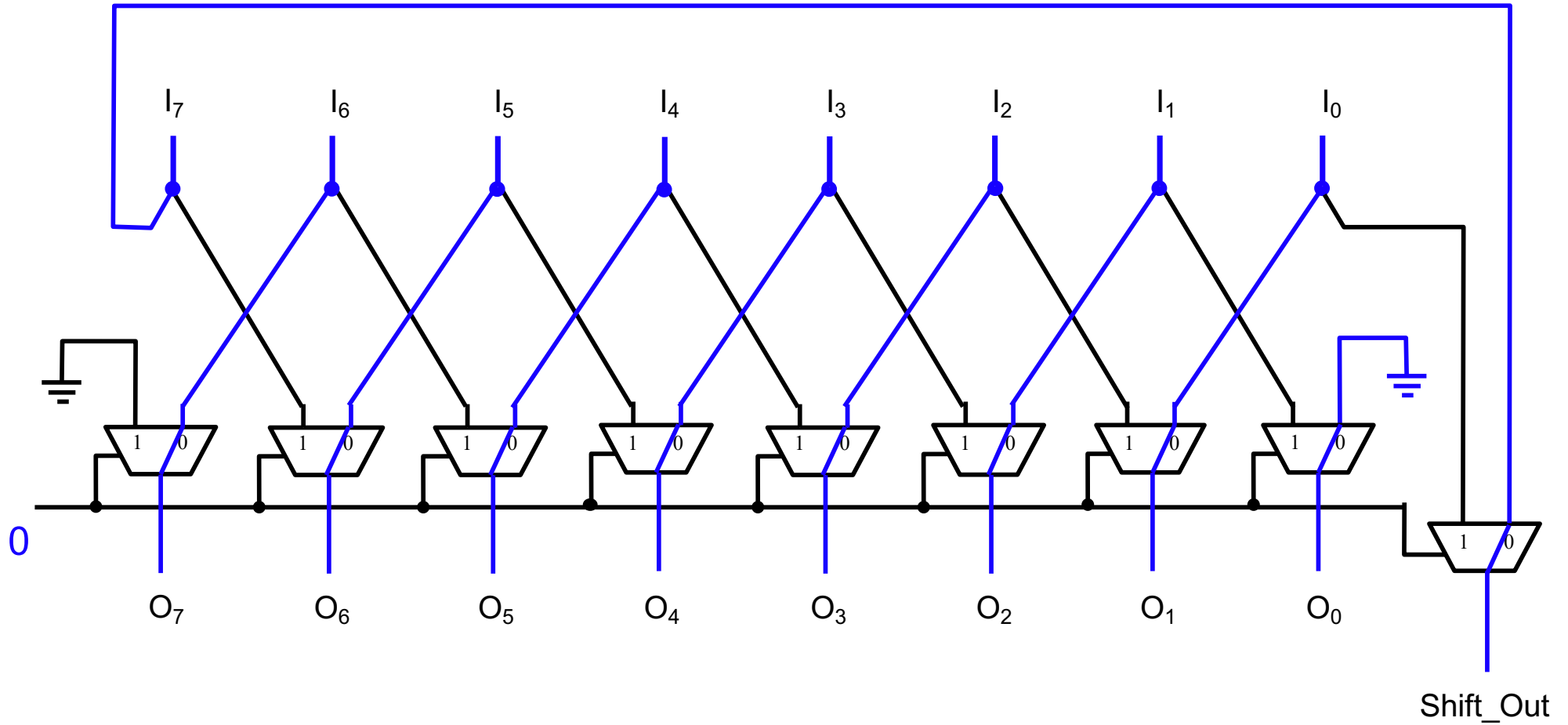
The Shifter Circuit



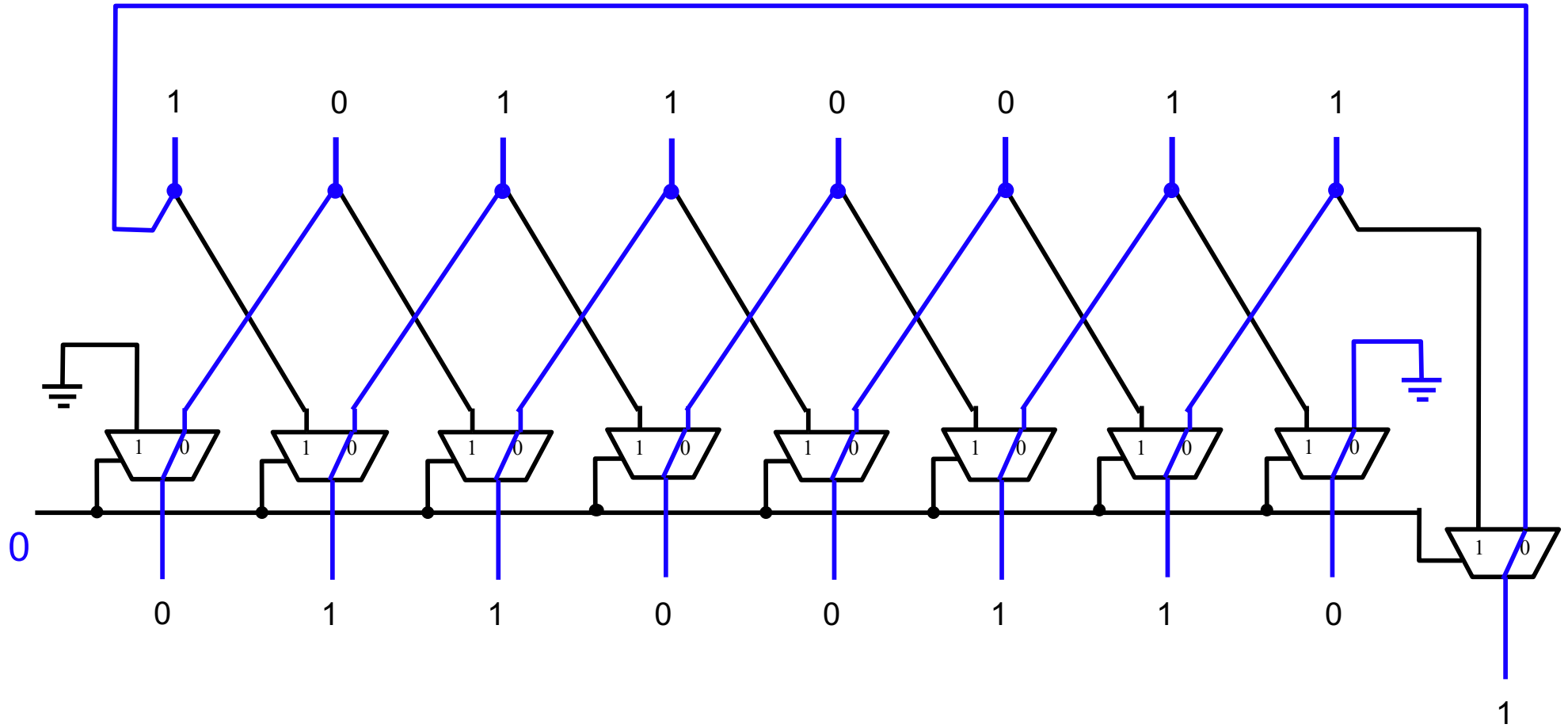




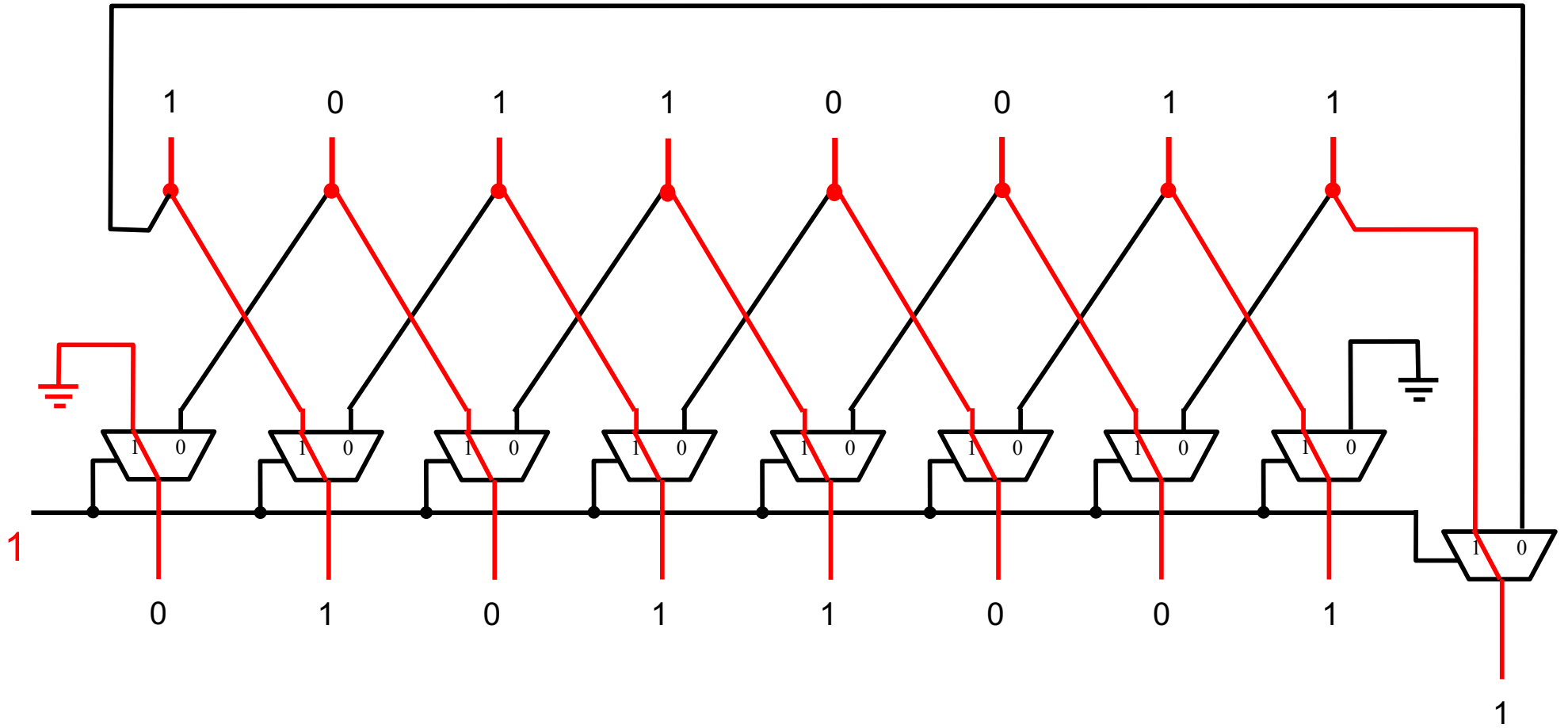
Shift Left



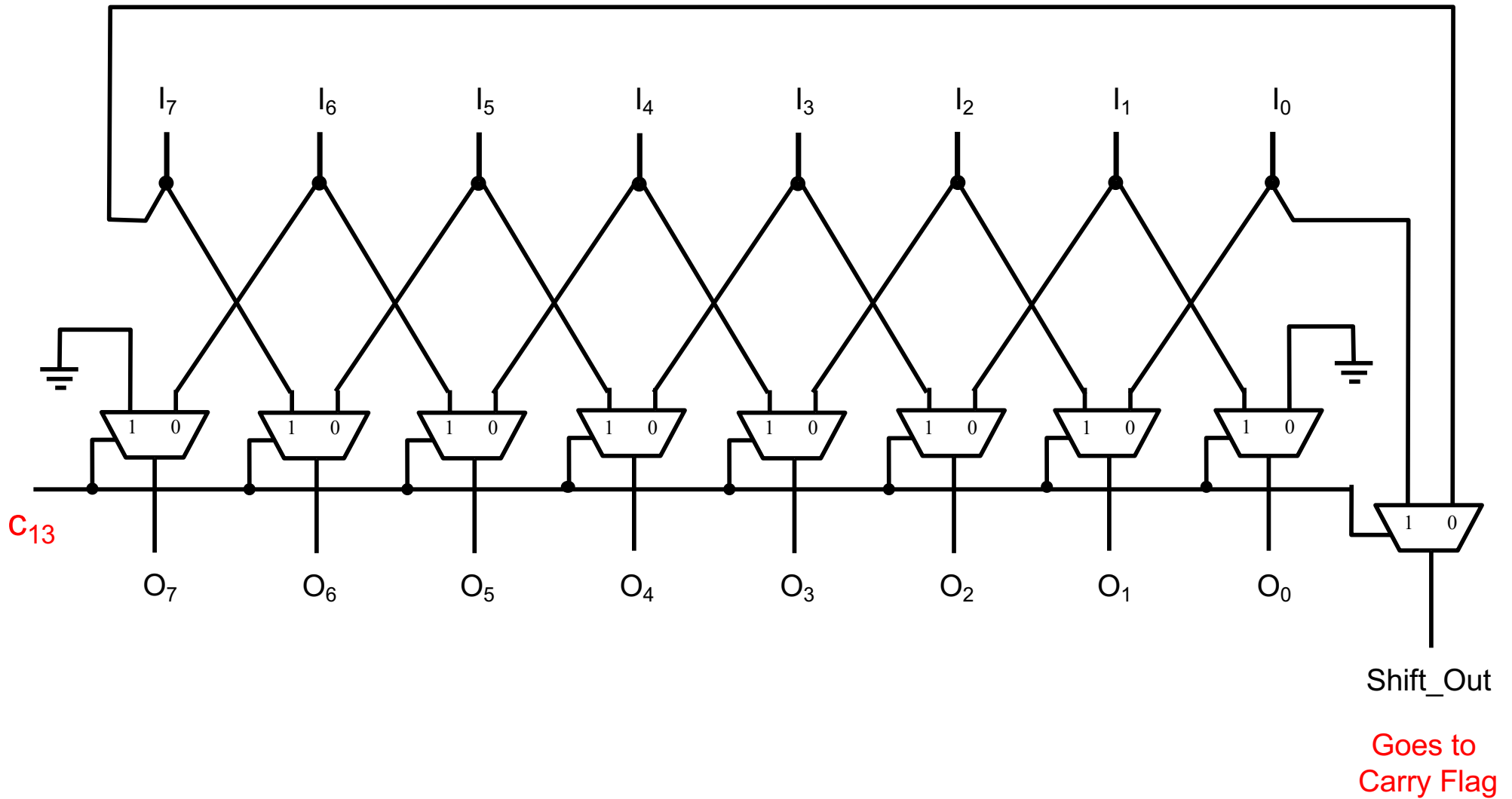
Shift Left



Shift Right



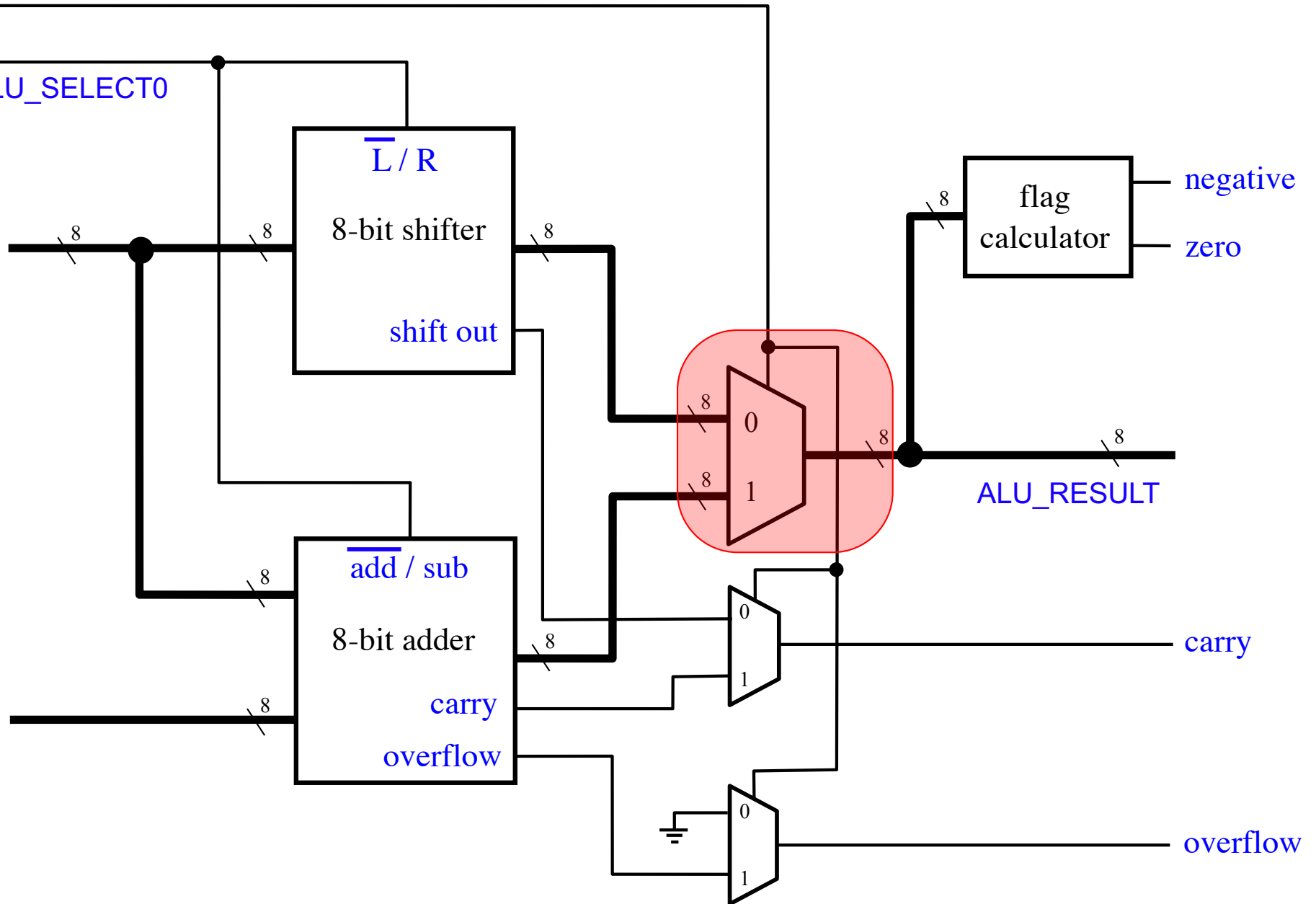
The ALU Shifter Circuit



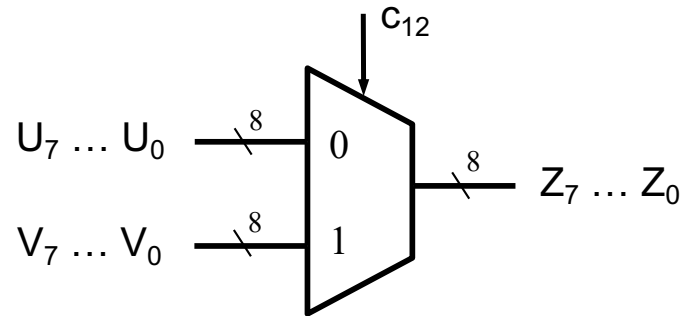
The Internal ALU Bus Multiplexer

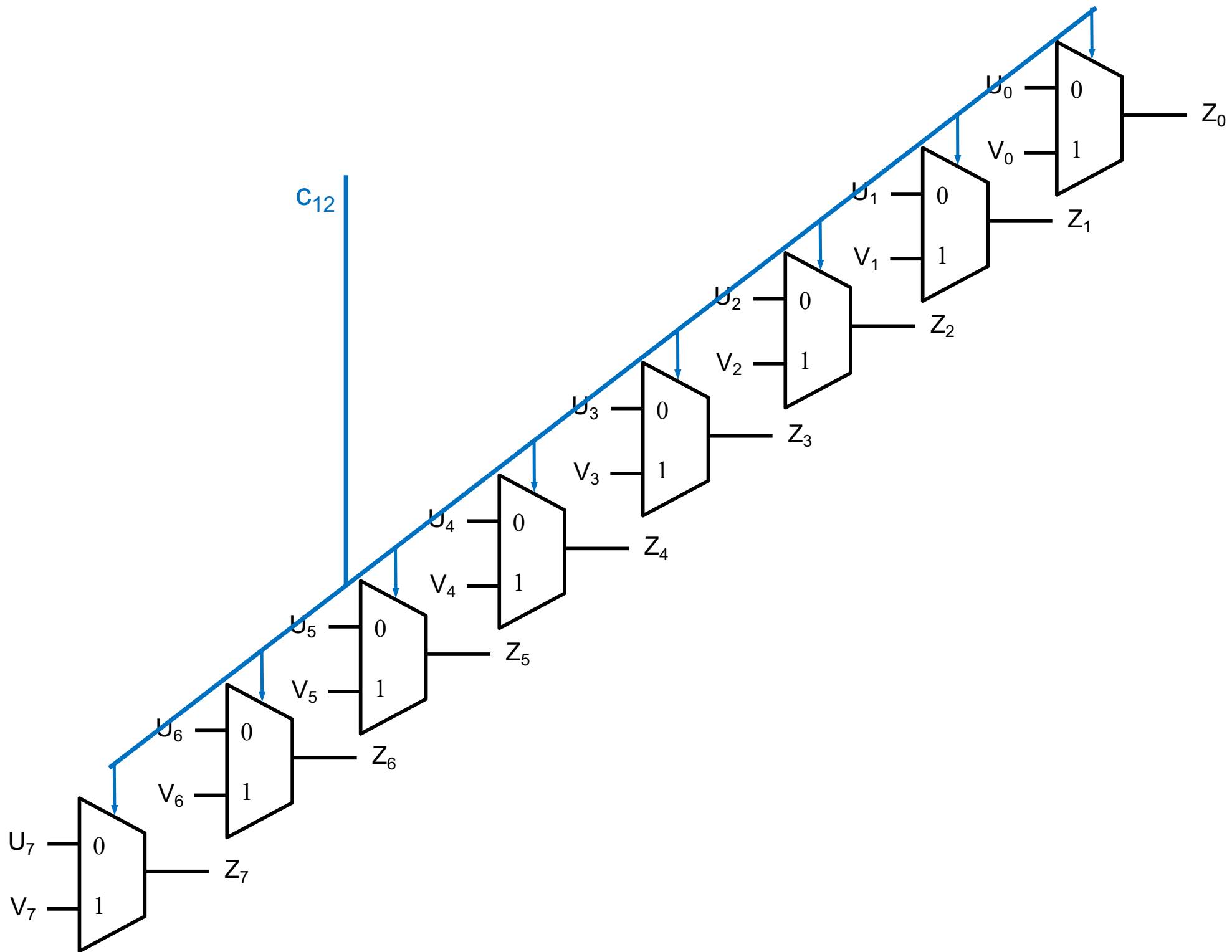
ALU_SELECT1

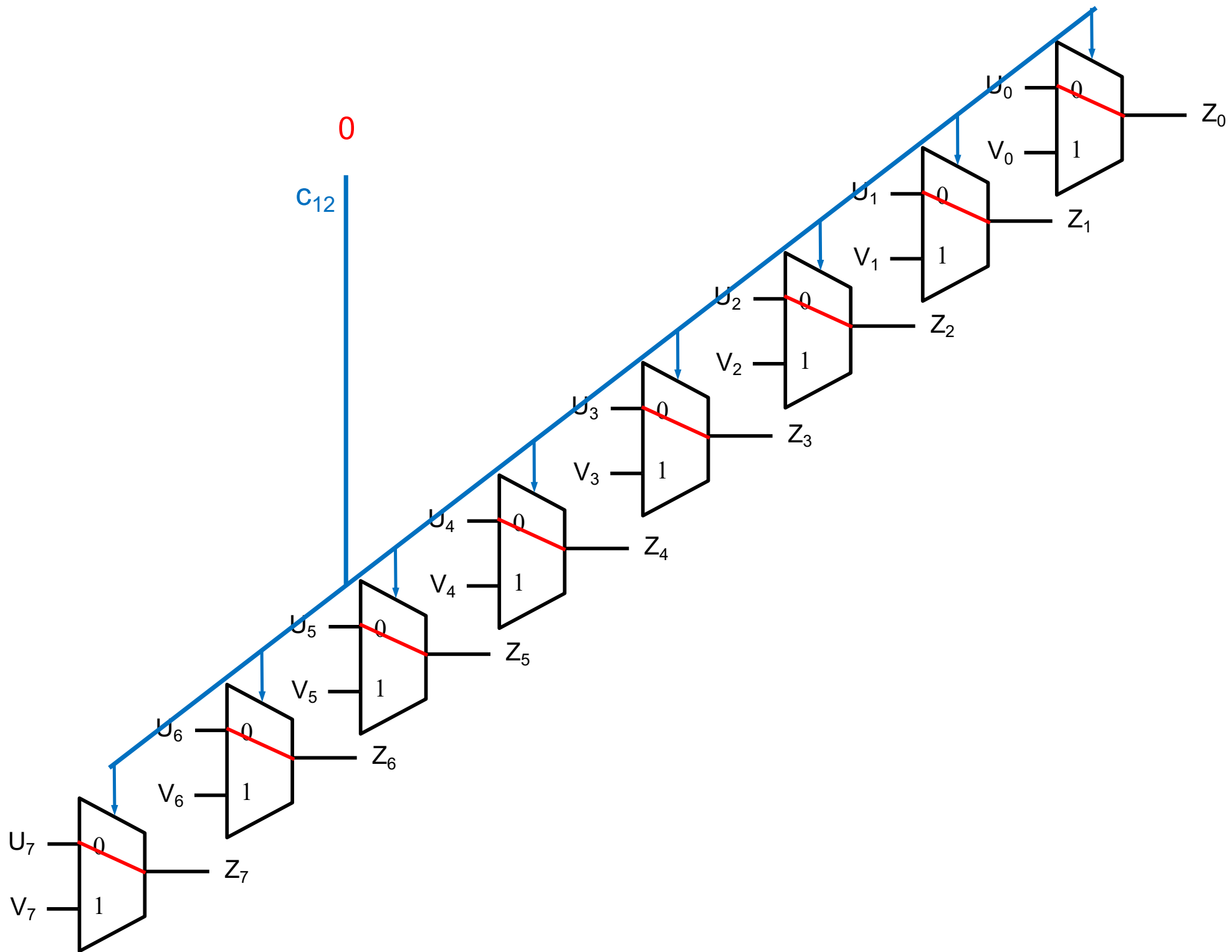
ALU_SELECT0

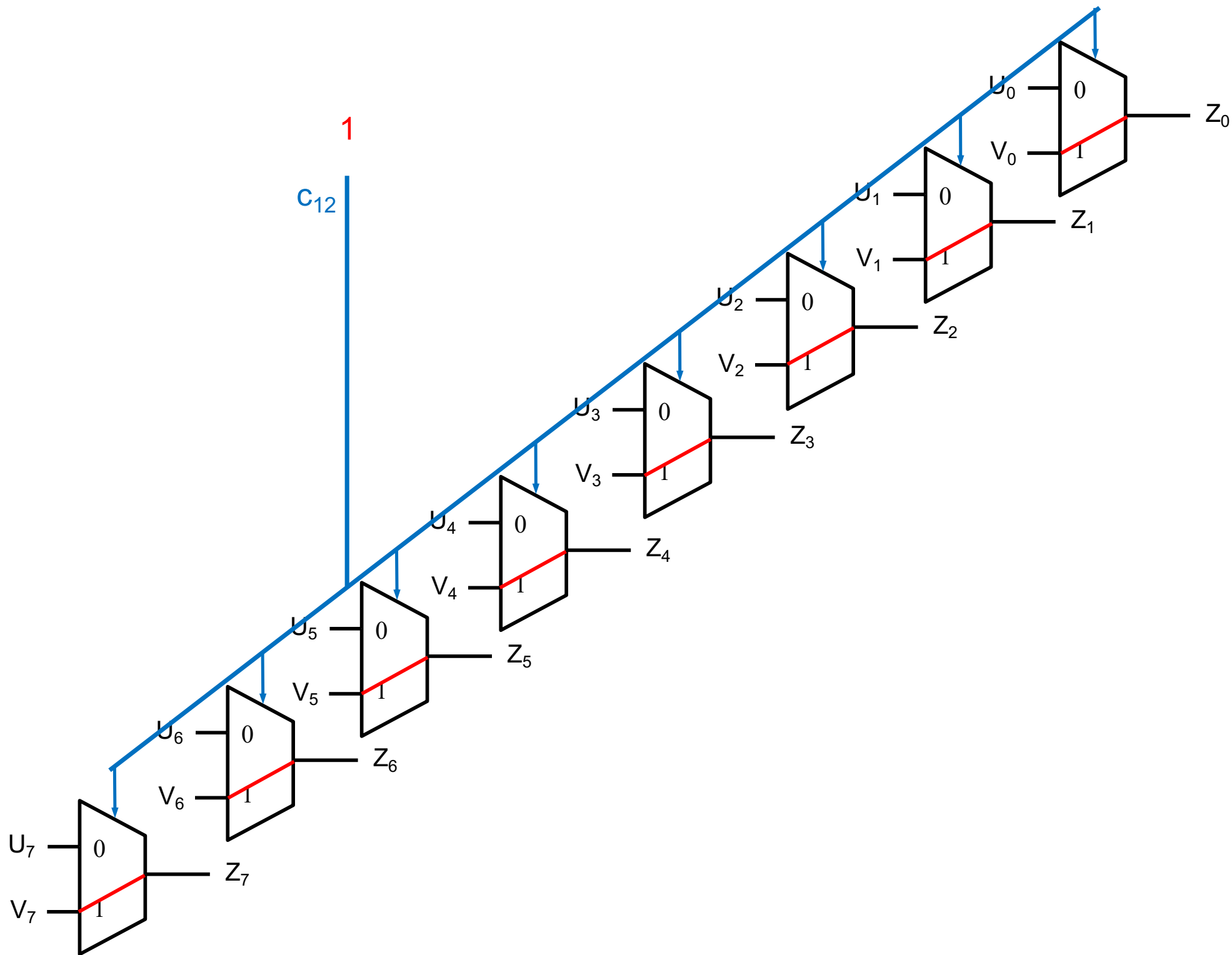


2-to-1 Bus Multiplexer (with 8-bit lines)

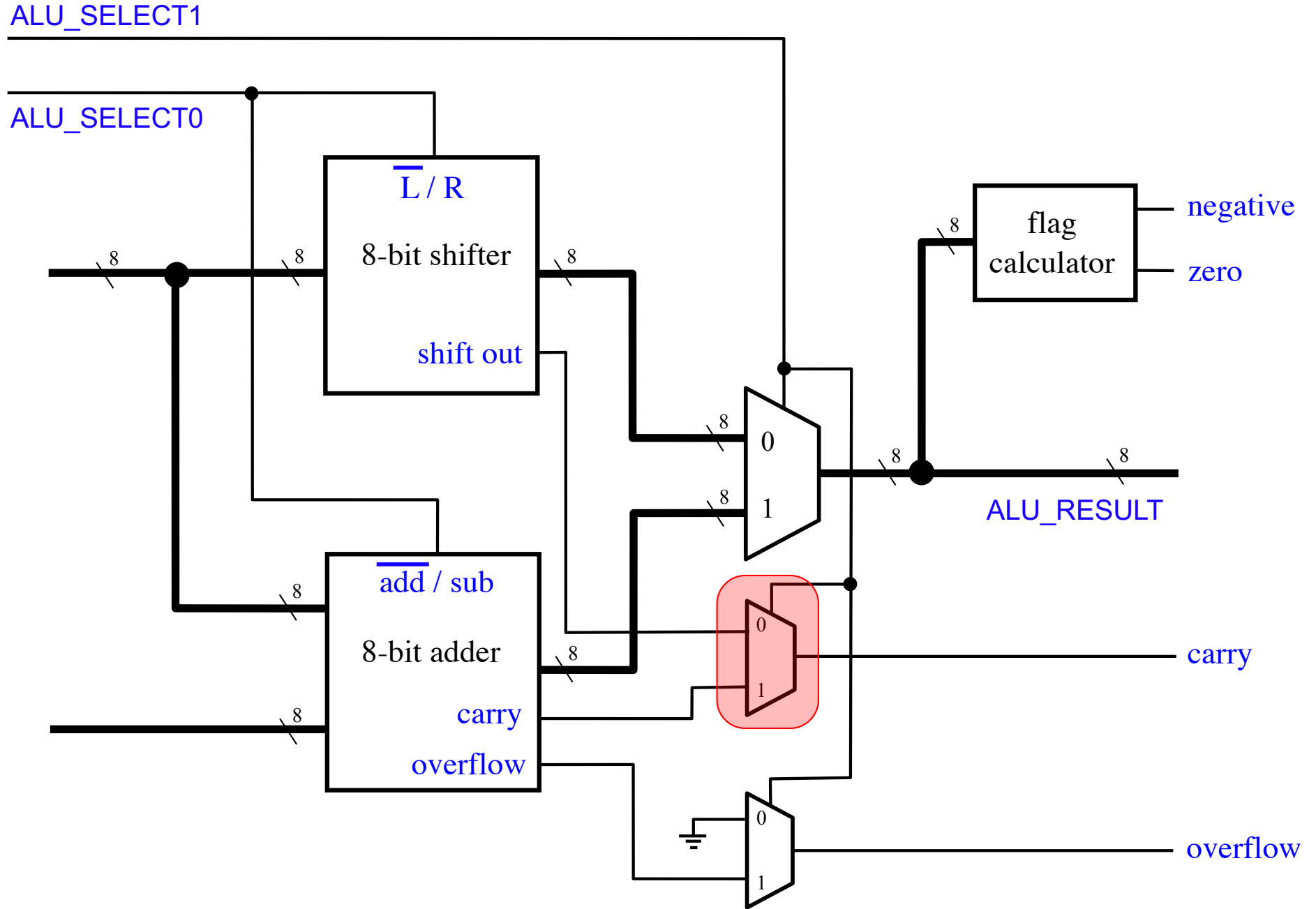




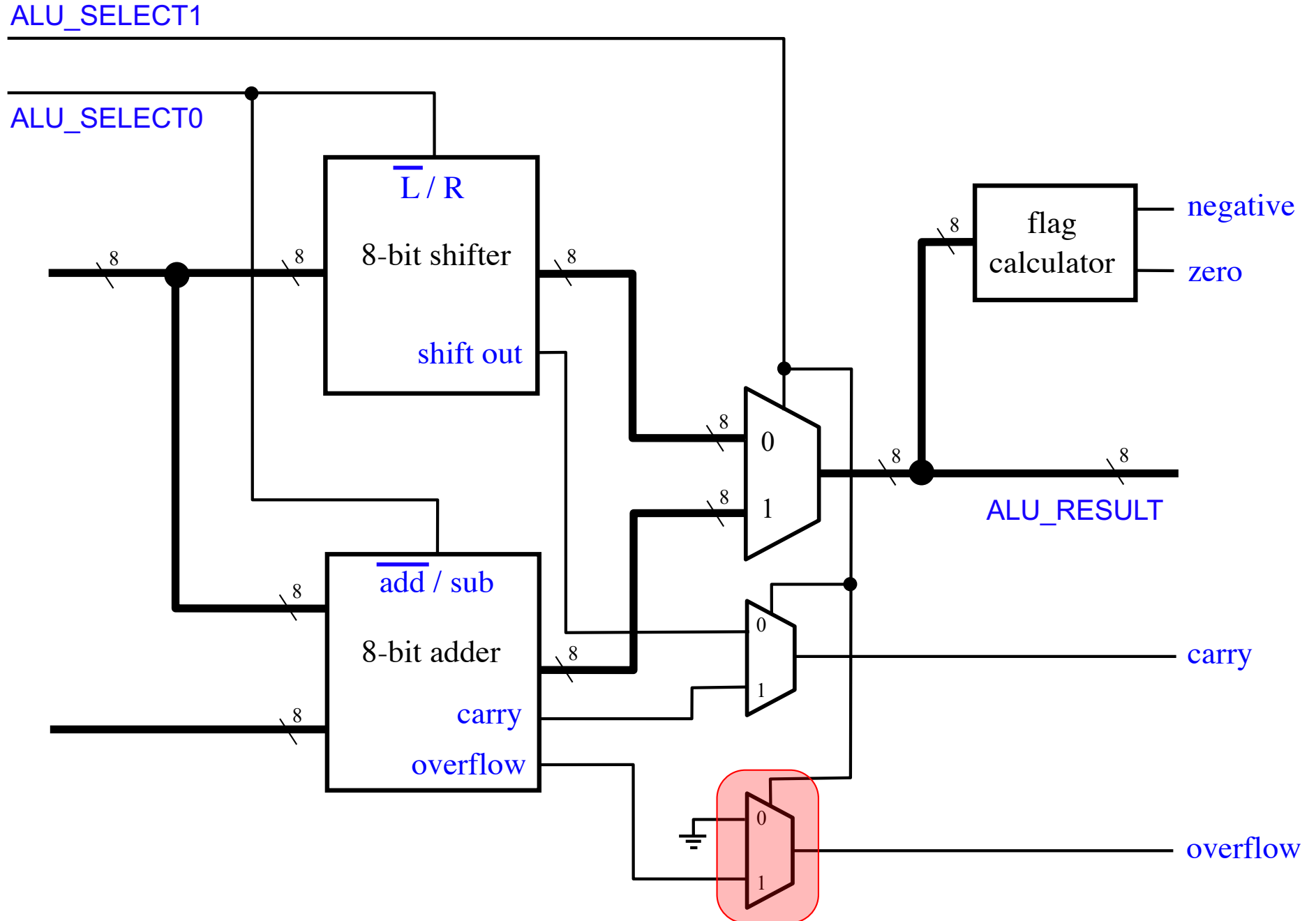




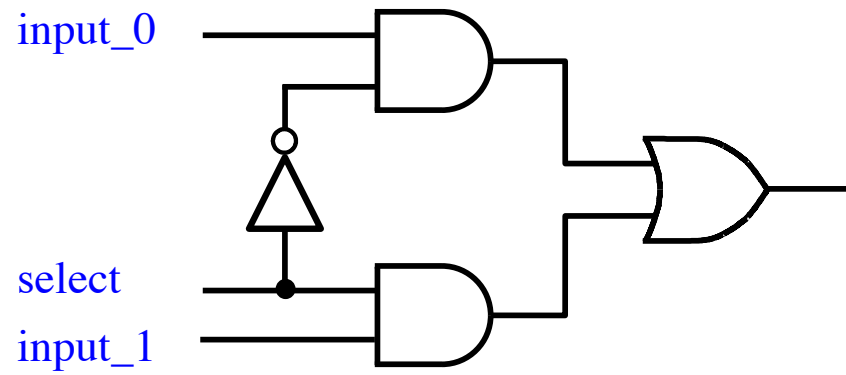
2-to-1 Multiplexer



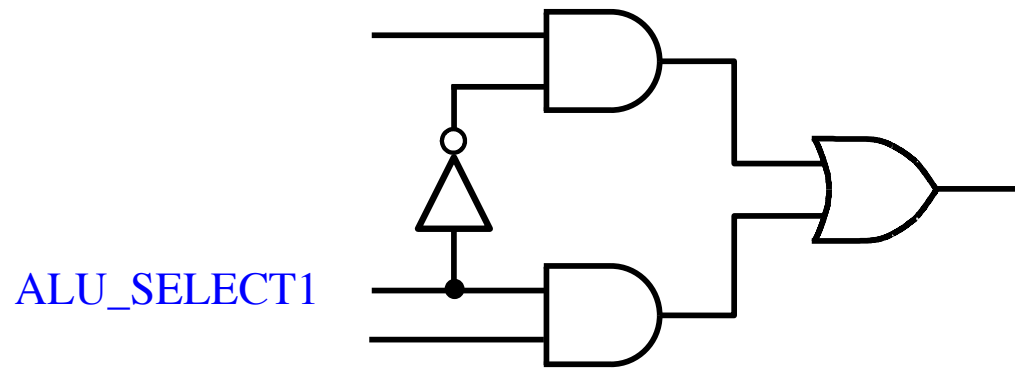
2-to-1 Multiplexer



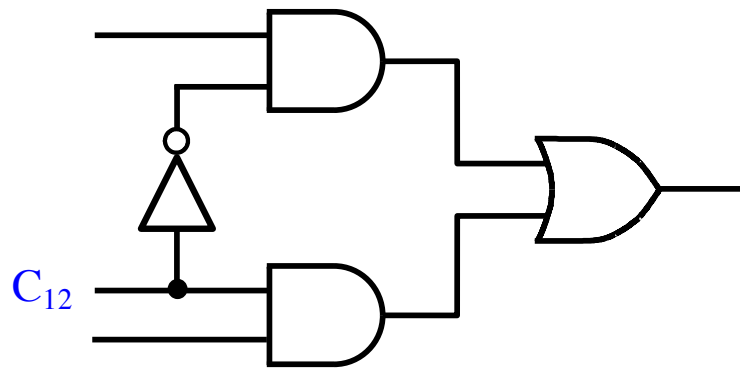
2-to-1 Multiplexer



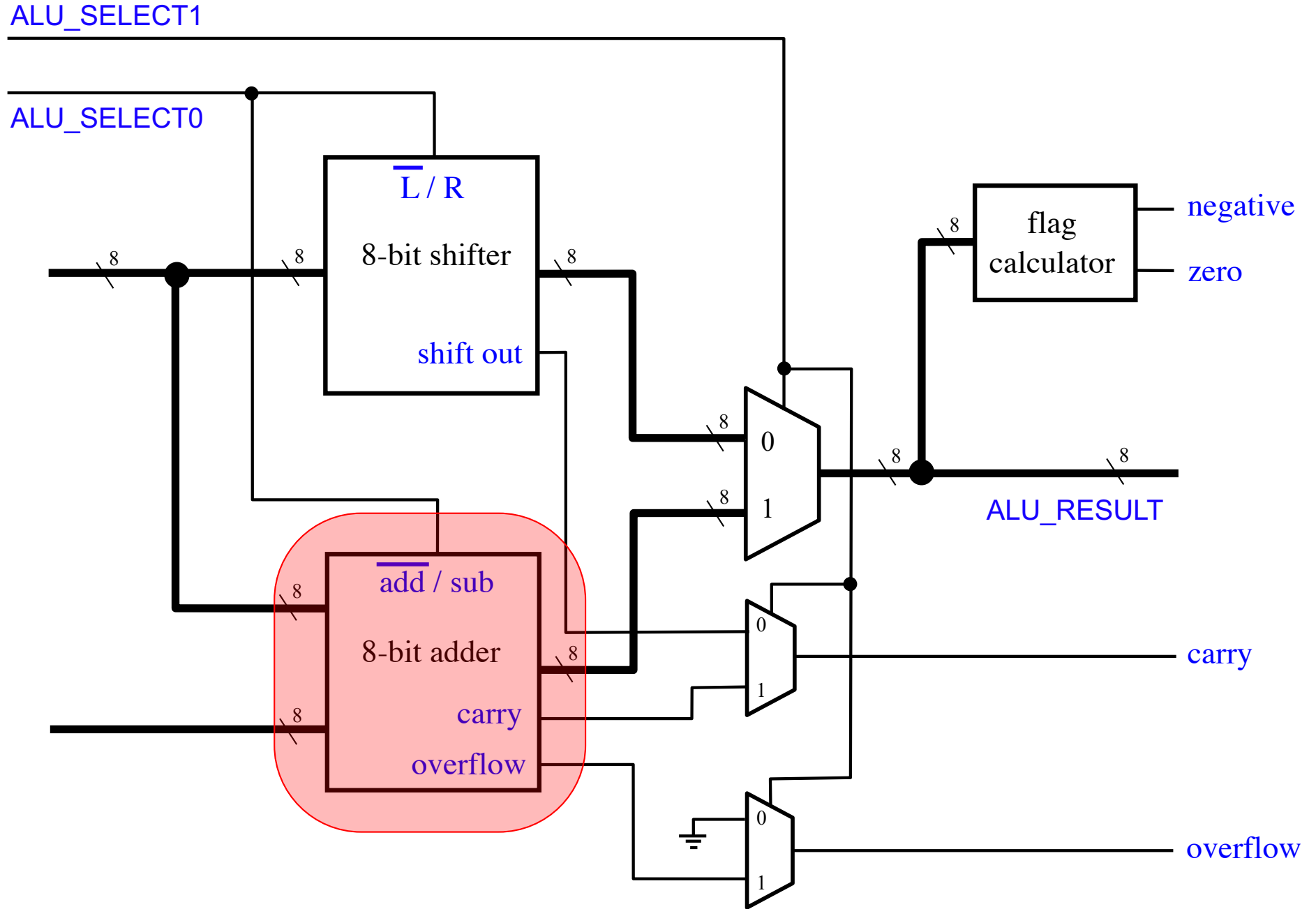
2-to-1 Multiplexer



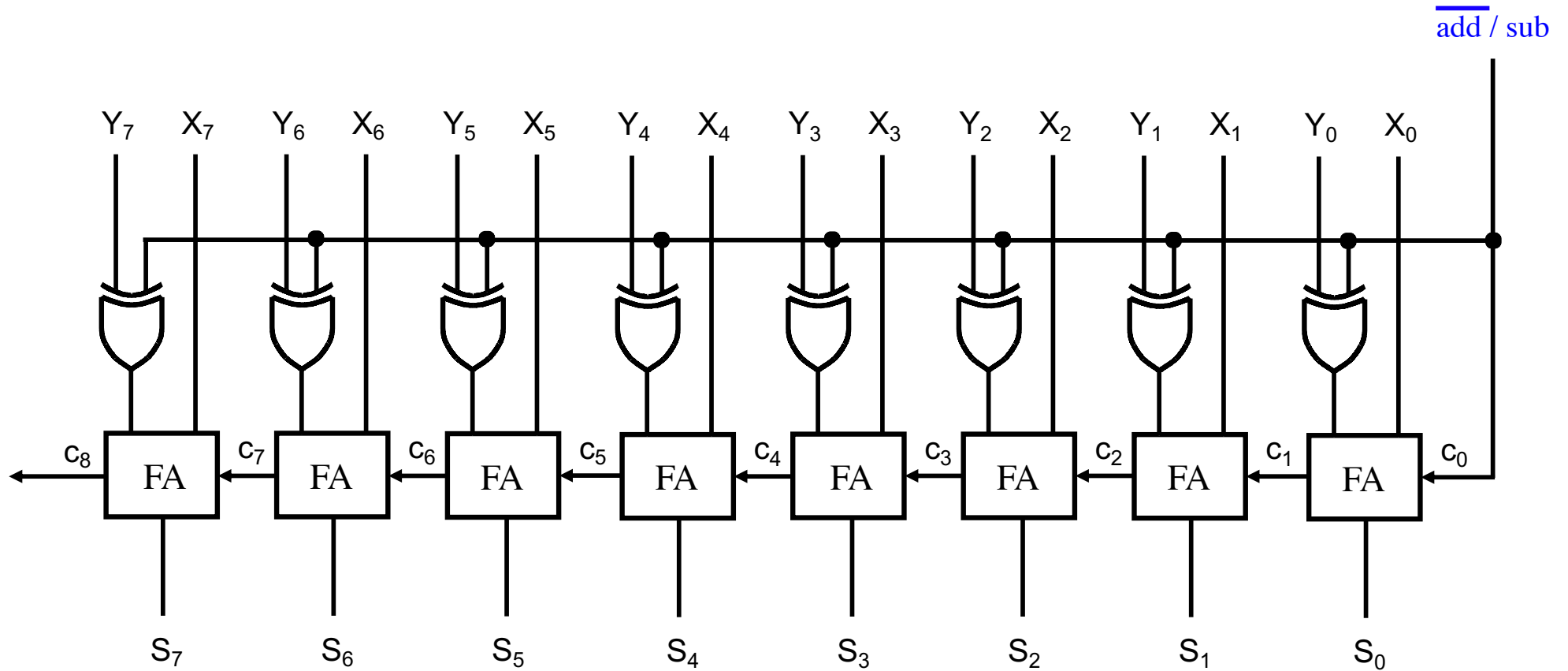
2-to-1 Multiplexer



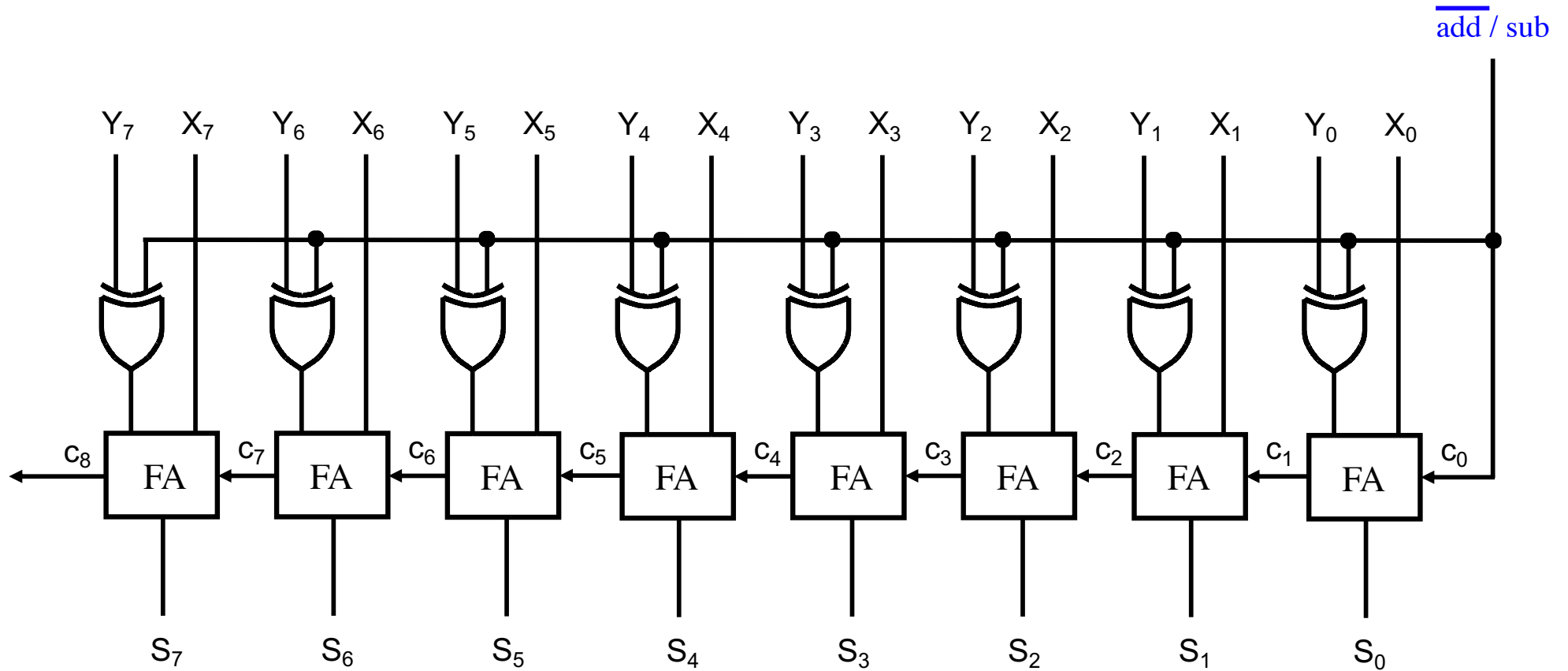
The Adder / Subtractor



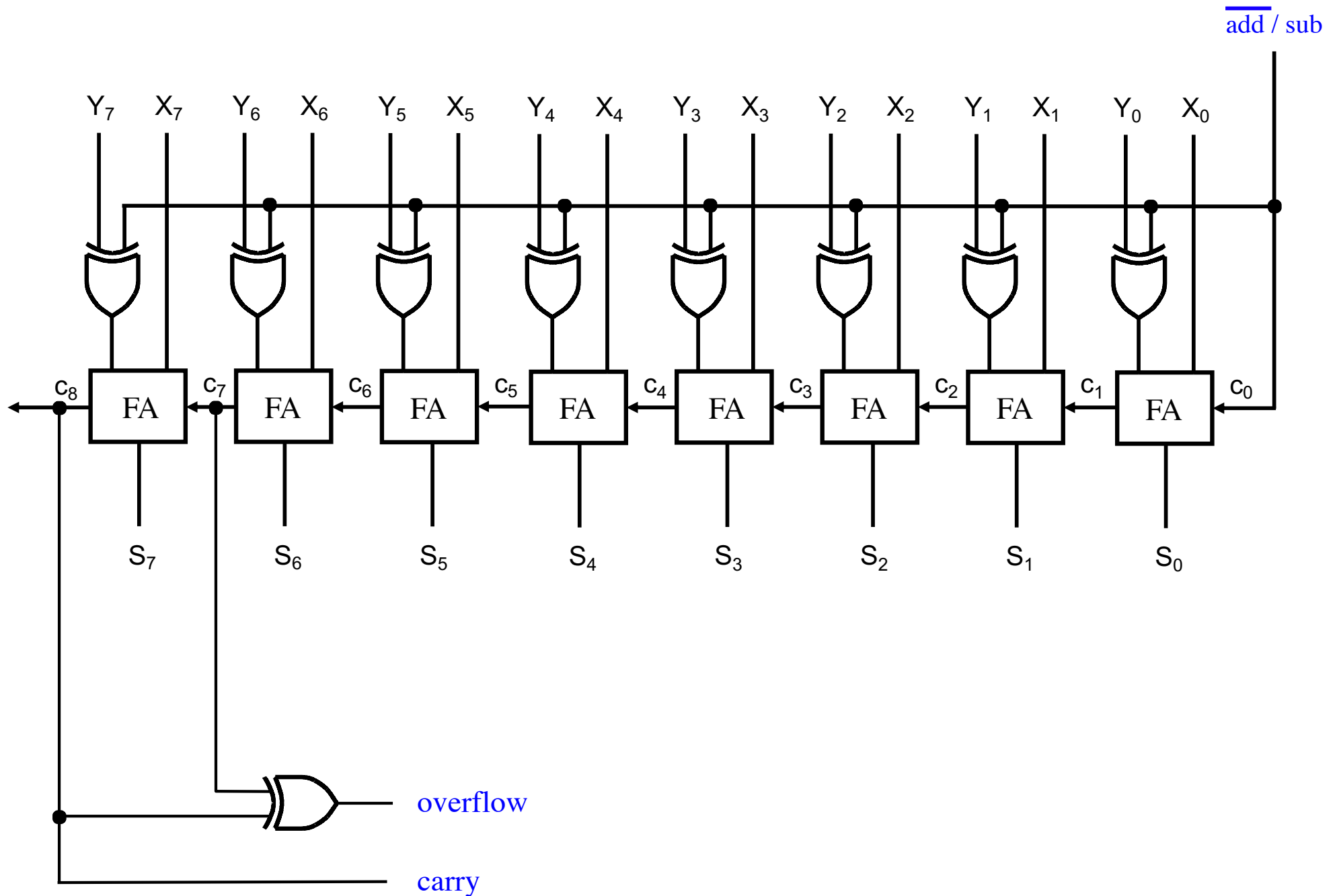
The Adder / Subtractor



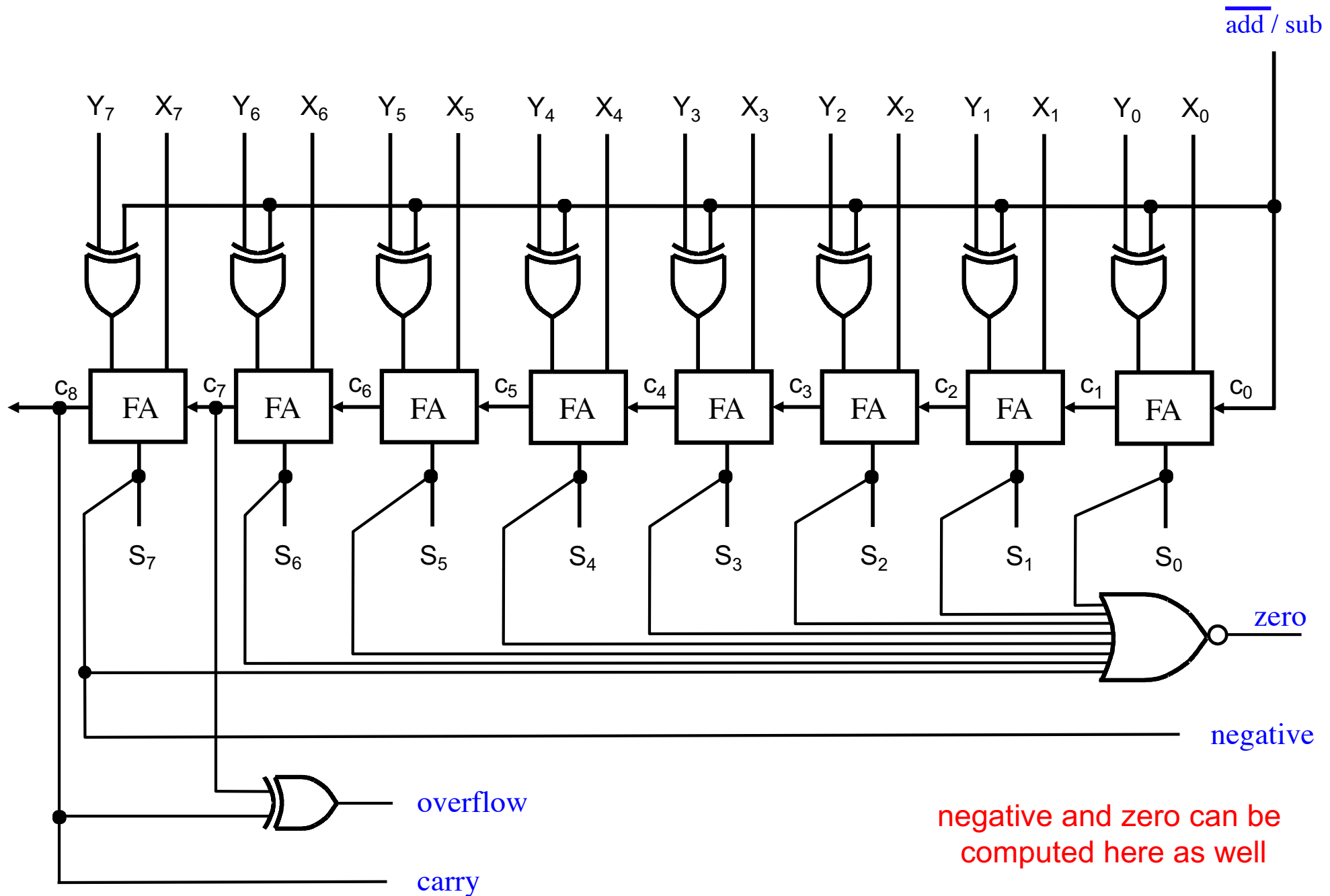
The Adder / Subtractor



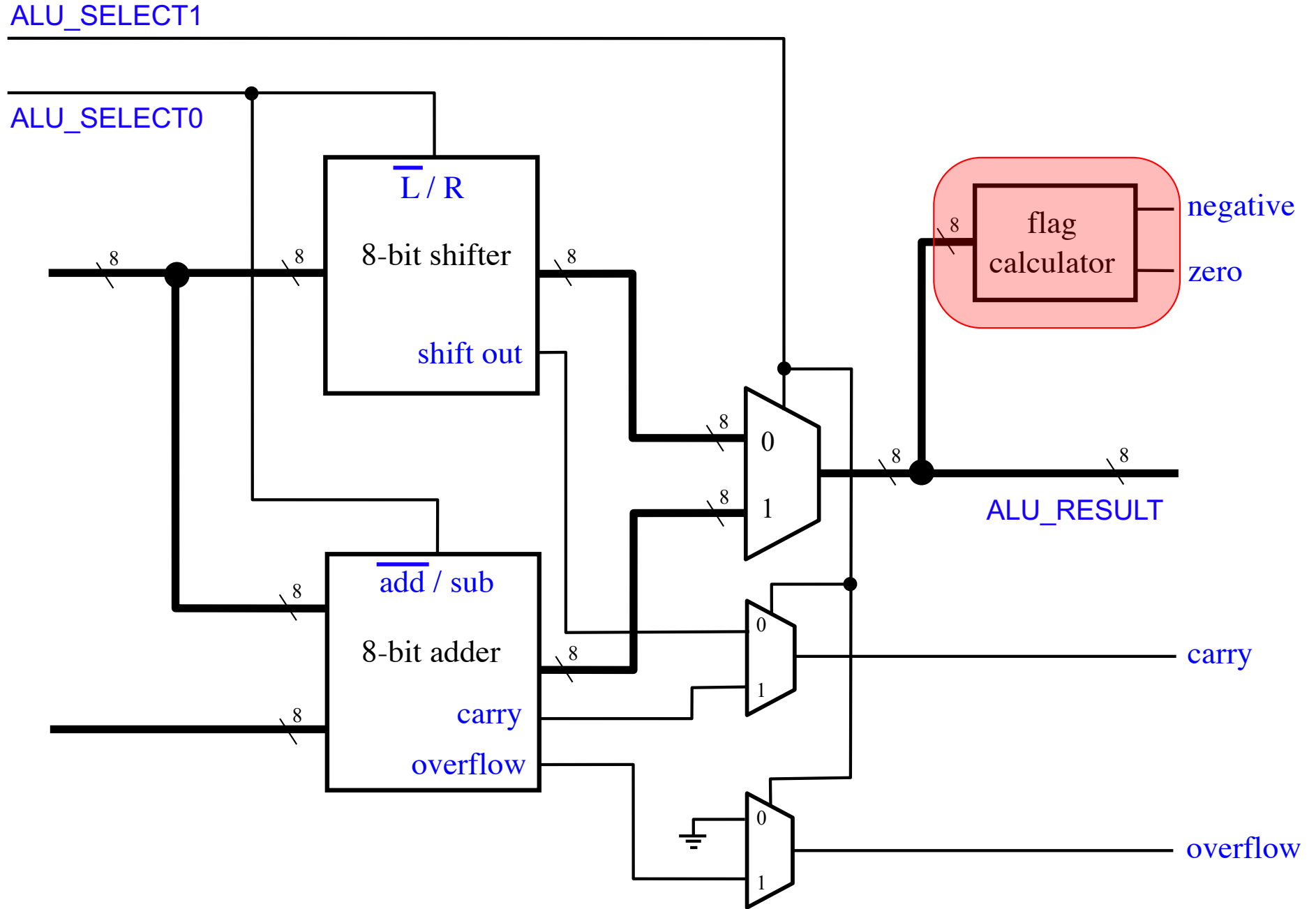
The Adder / Subtractor



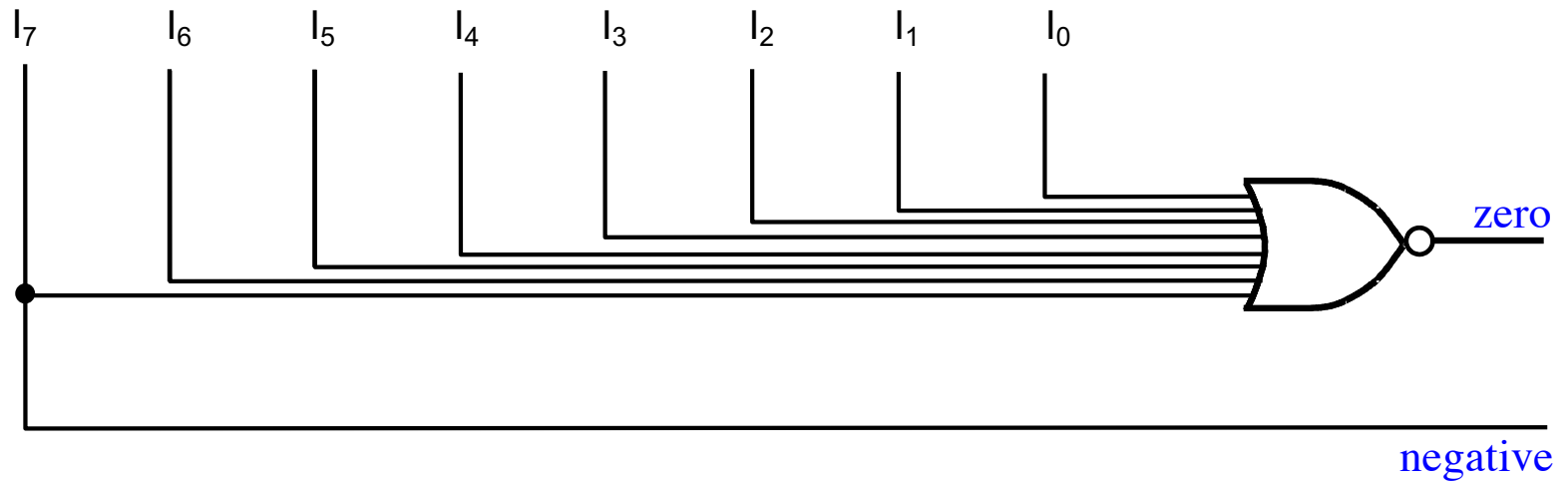
The Adder / Subtractor



The ALU Flag Calculator



The ALU Flag Calculator

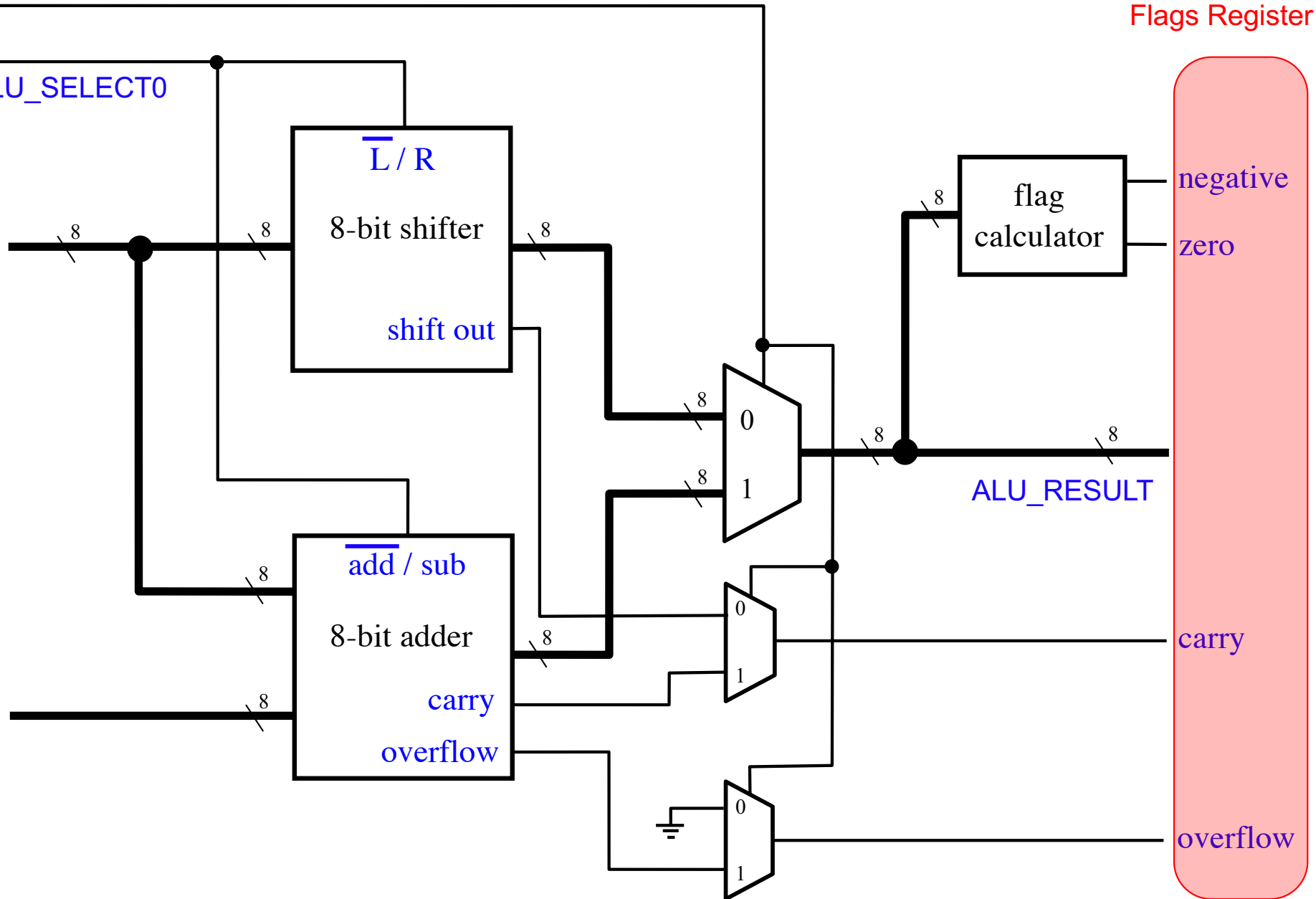


ALU Outputs to the Flags Register

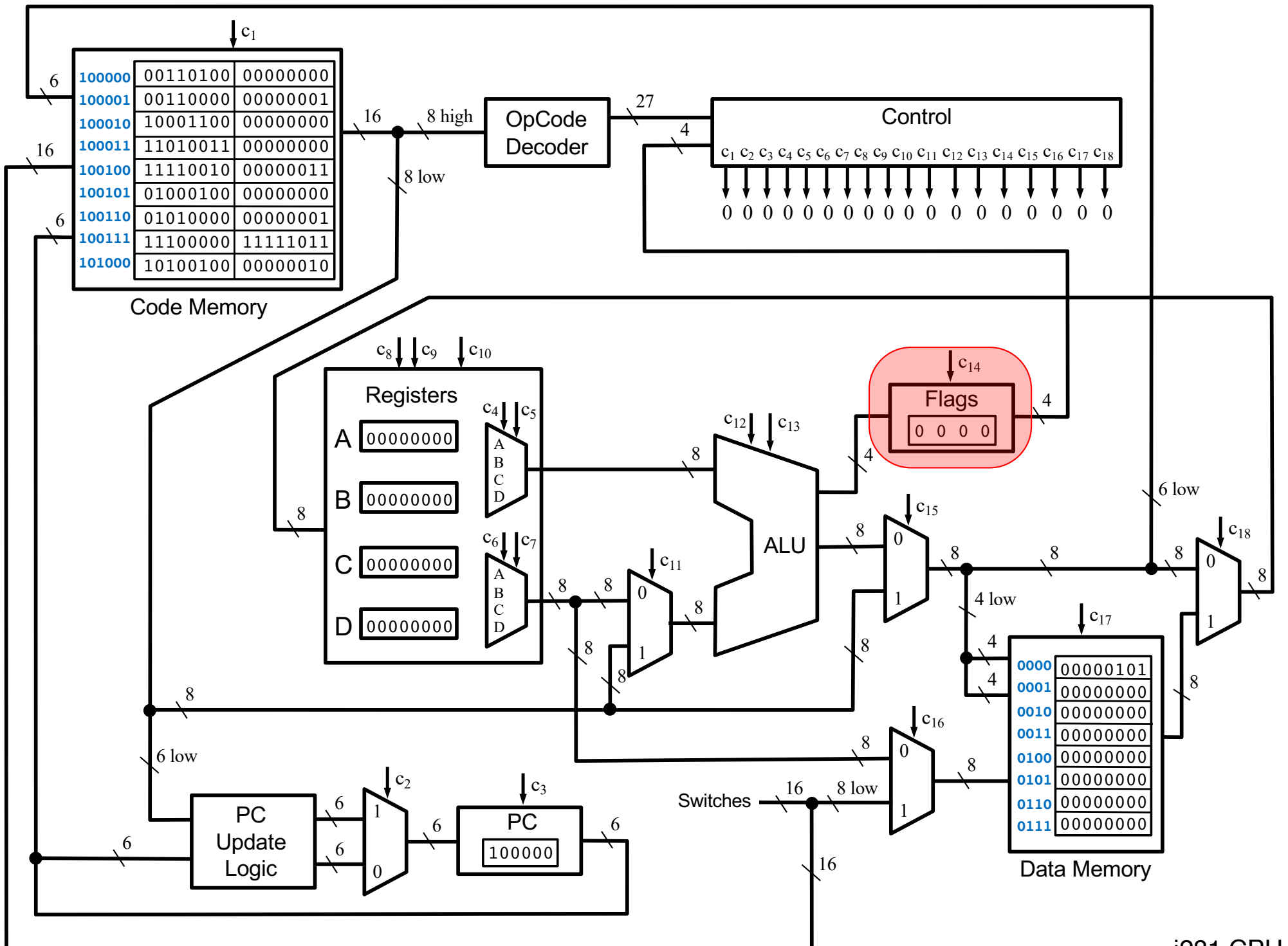
ALU_SELECT1

ALU_SELECT0

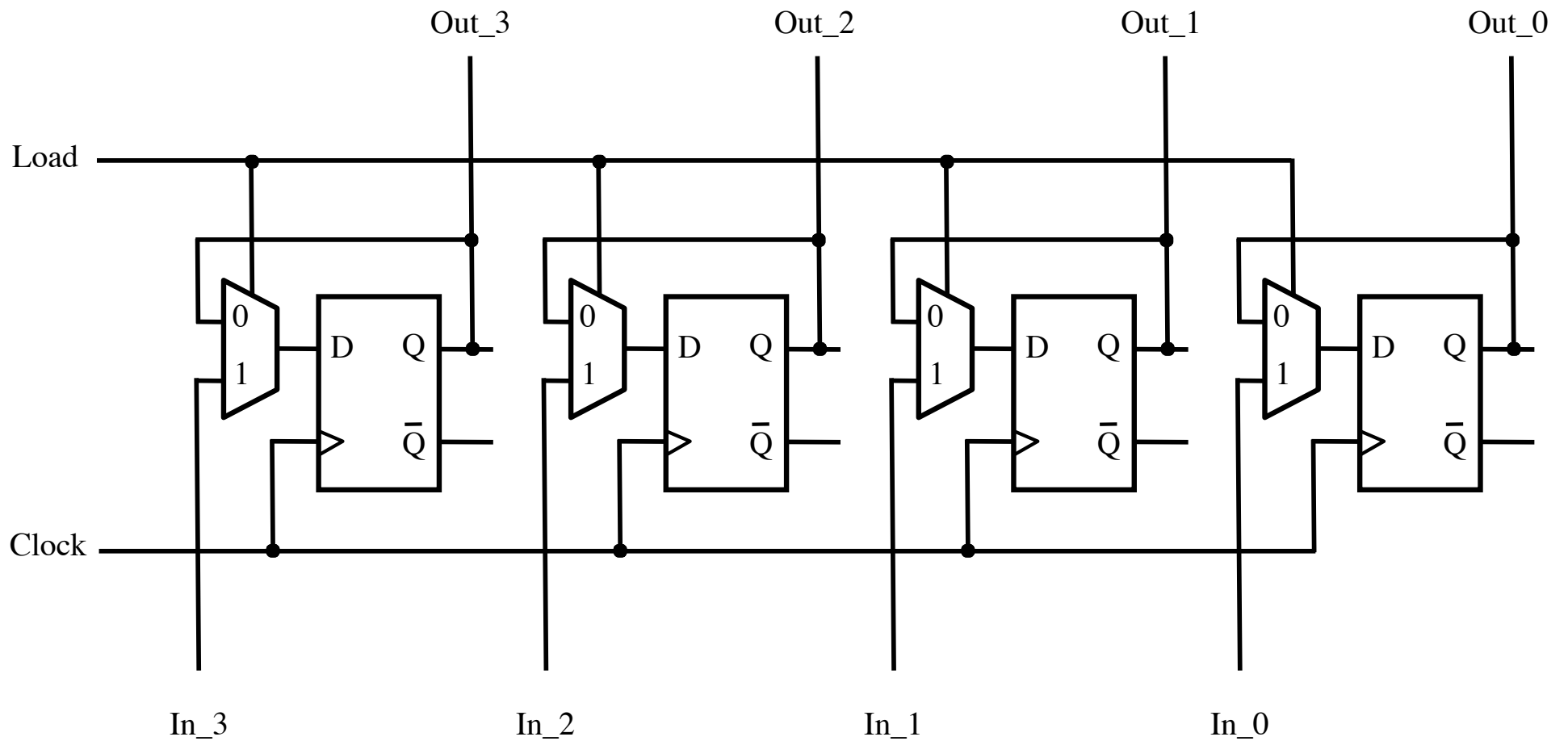
4 Outputs to the
Flags Register



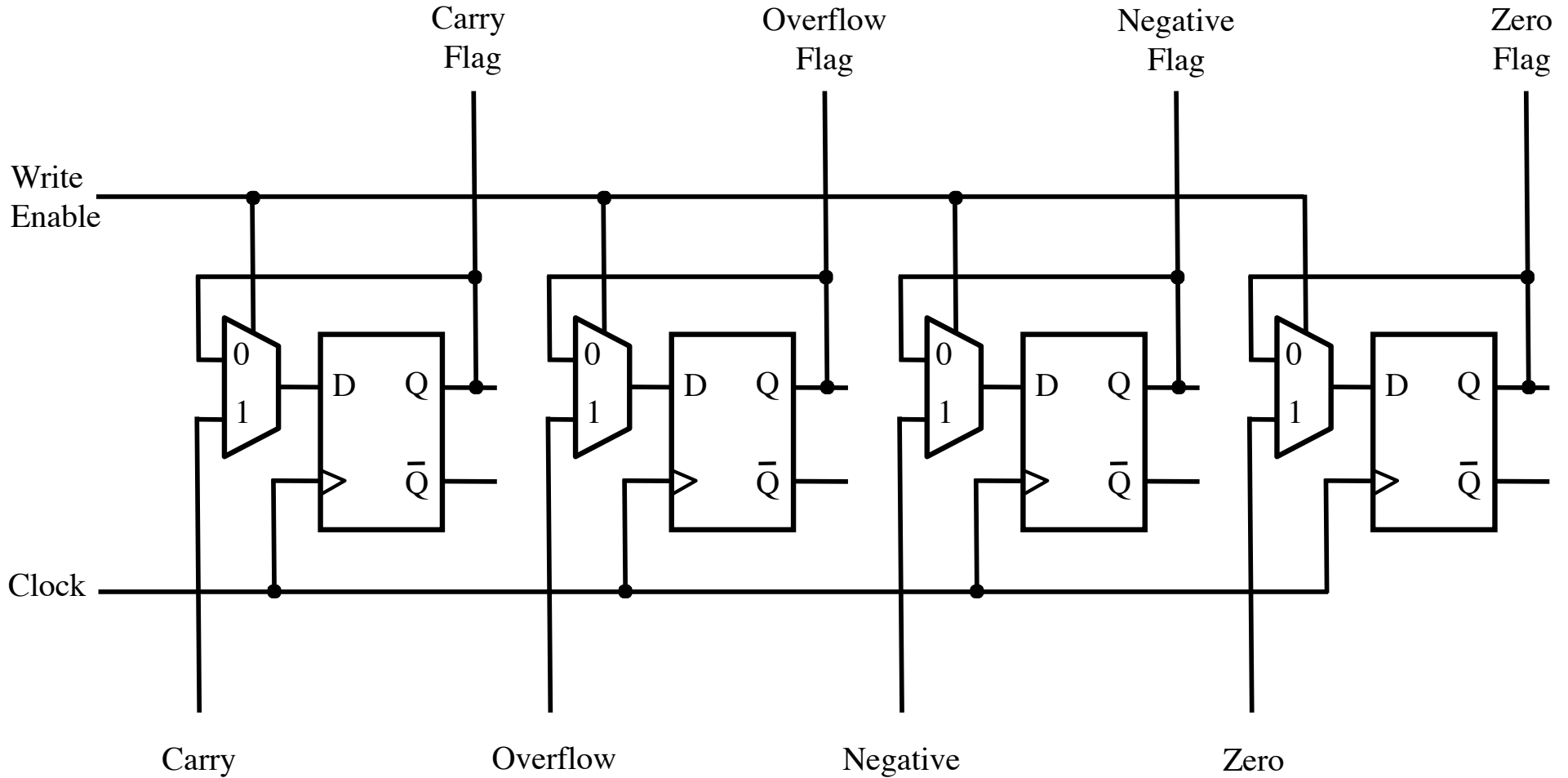
The Flags Register



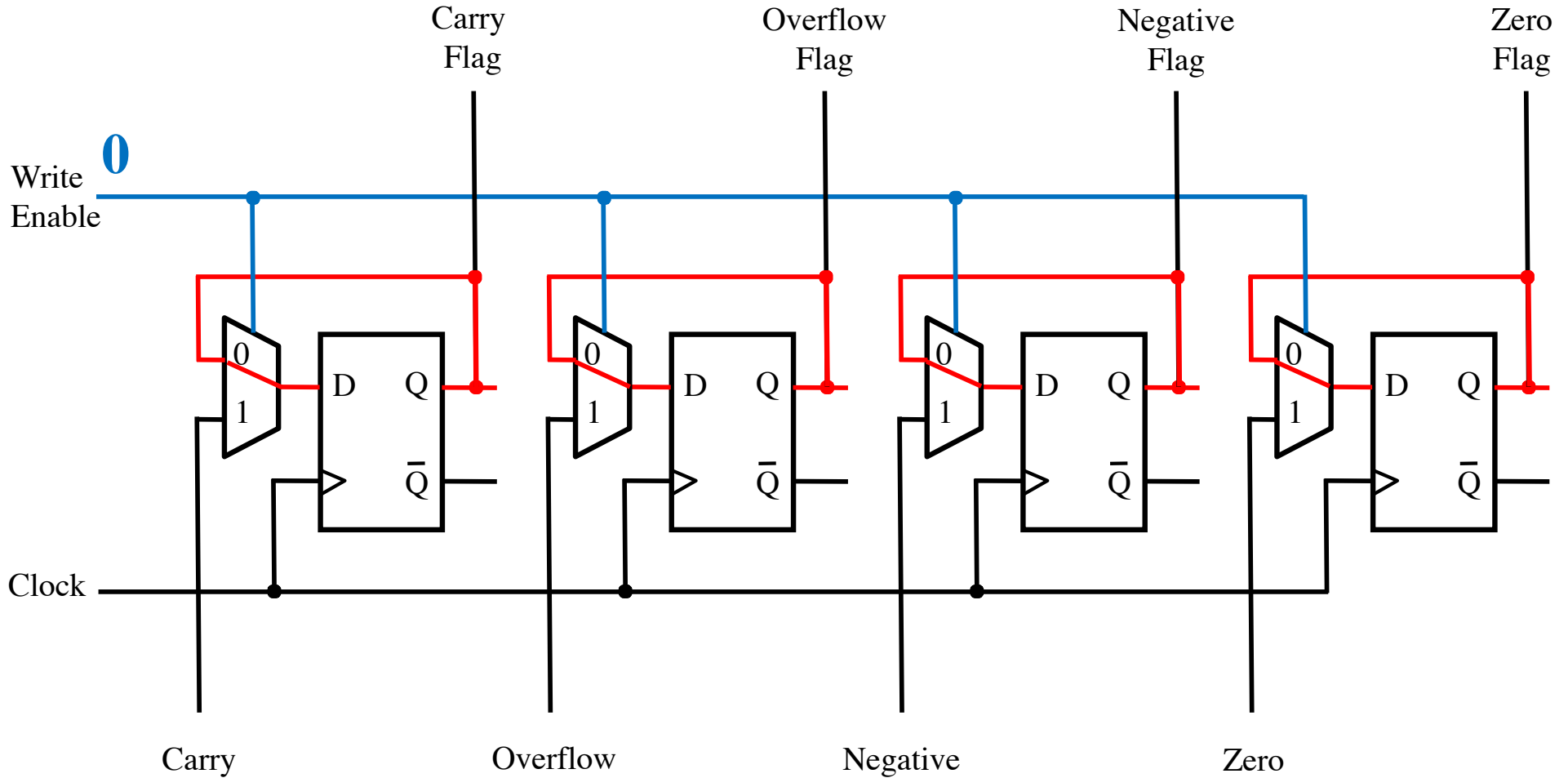
4-Bit Parallel-Access Register



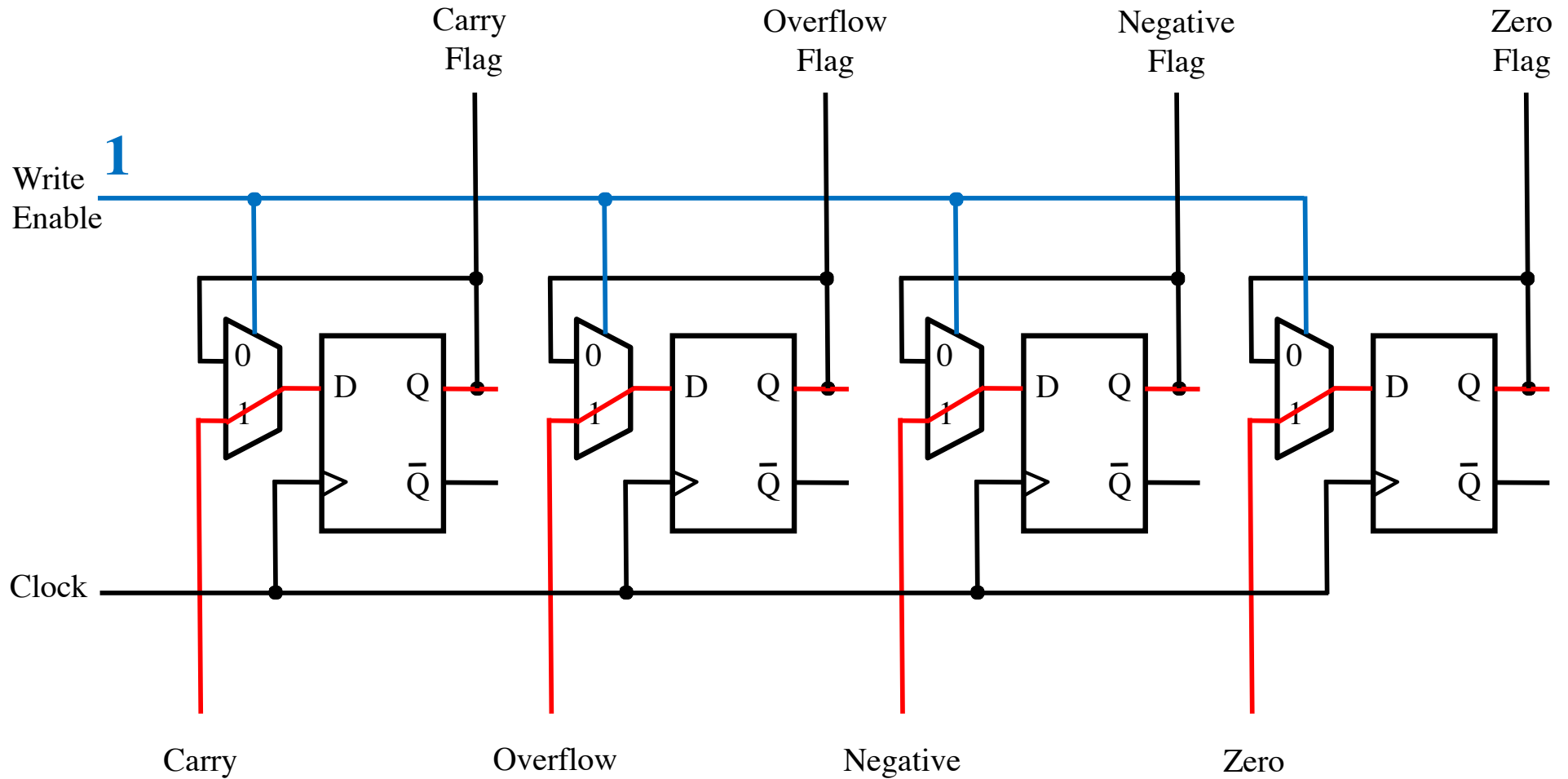
The Flags Register



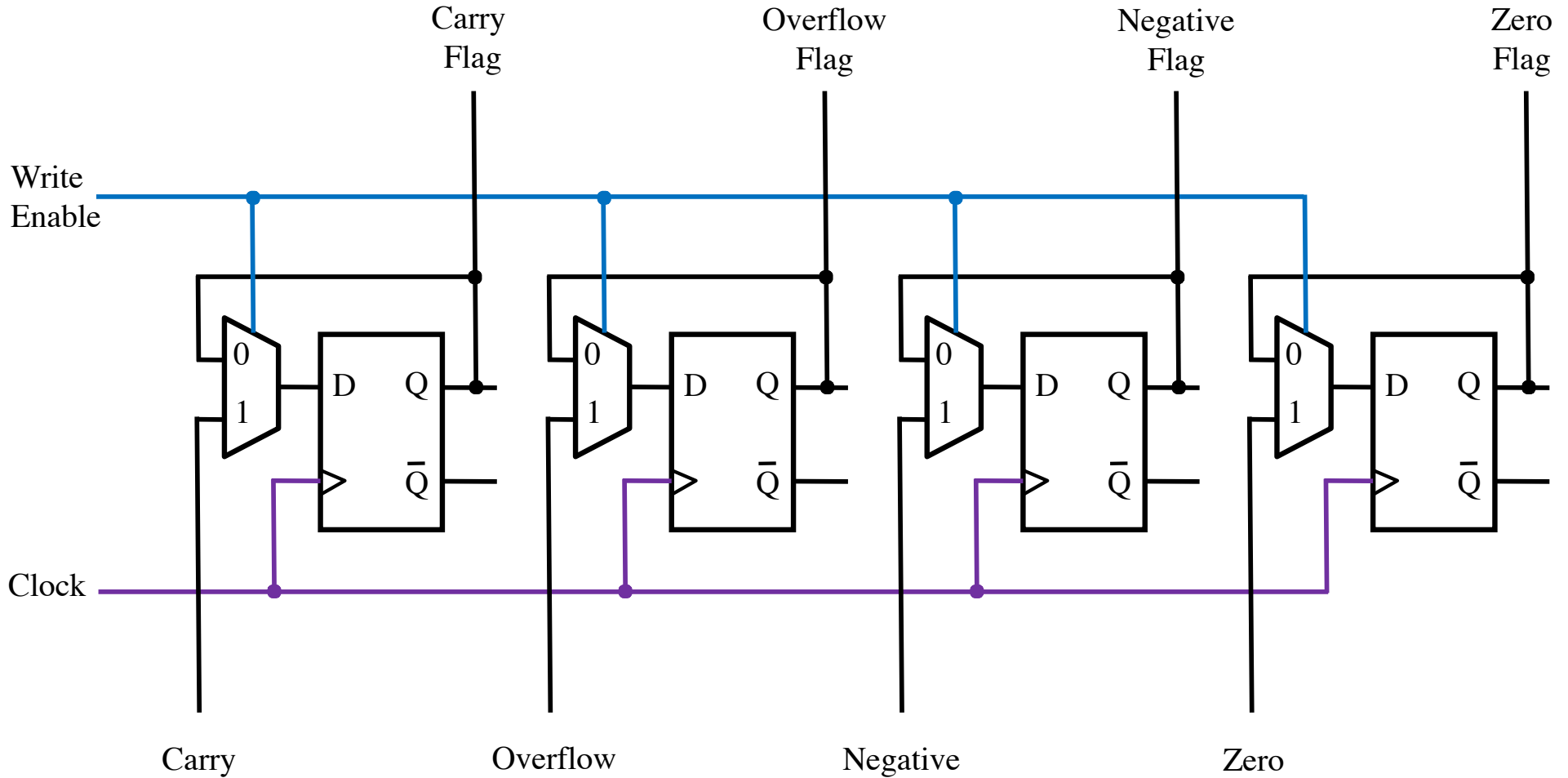
The Flags Register



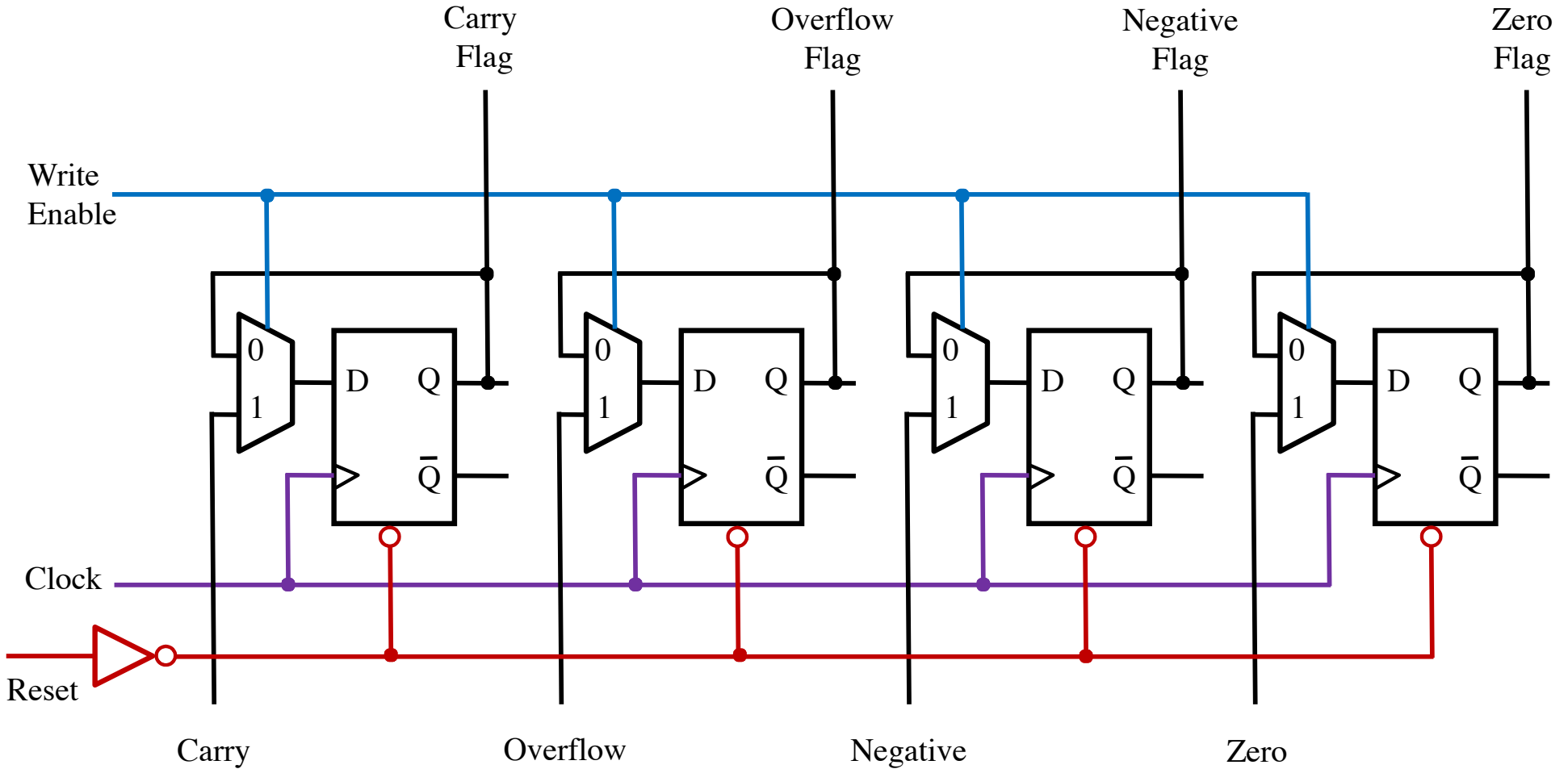
The Flags Register



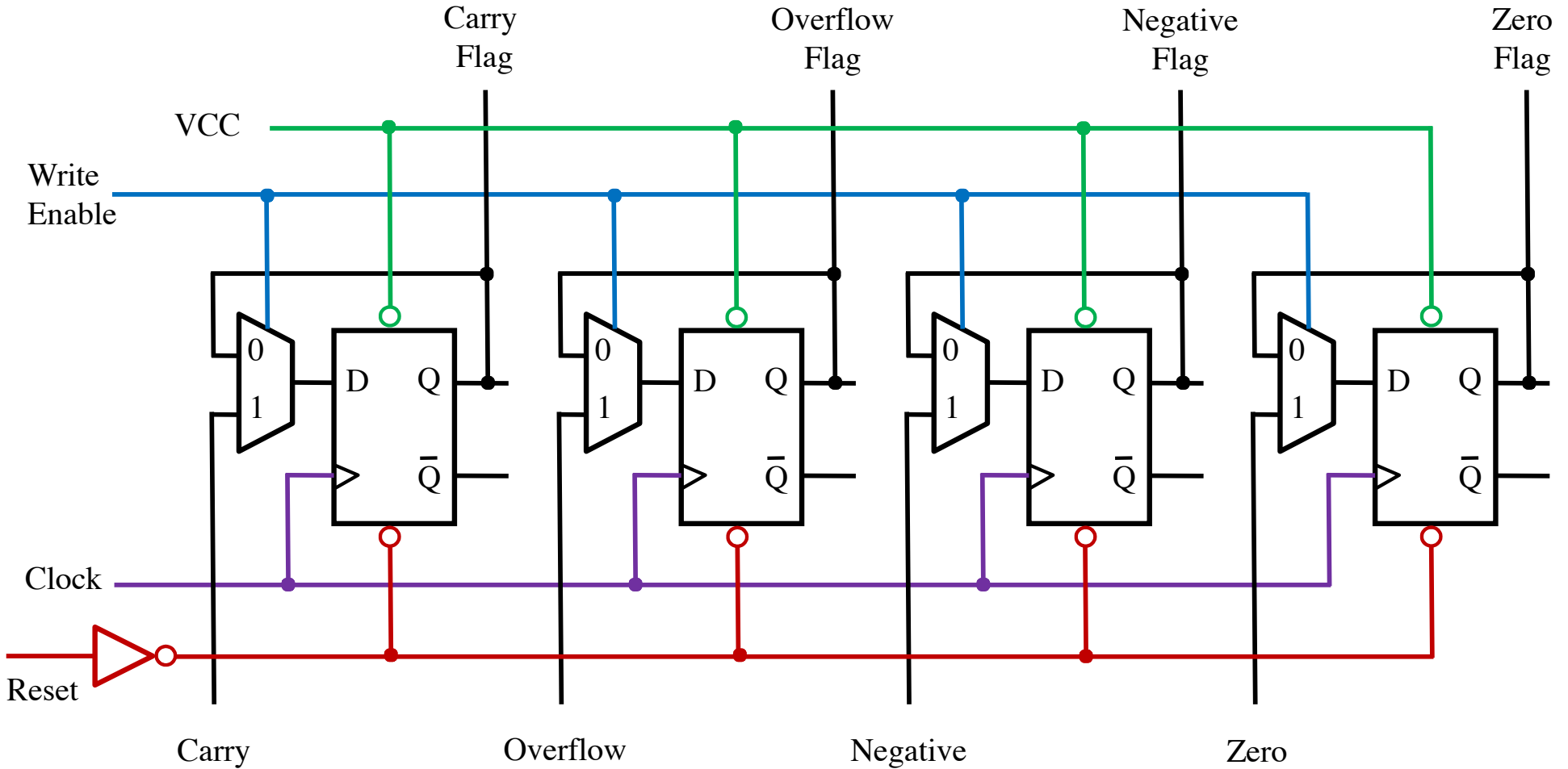
The Flags Register



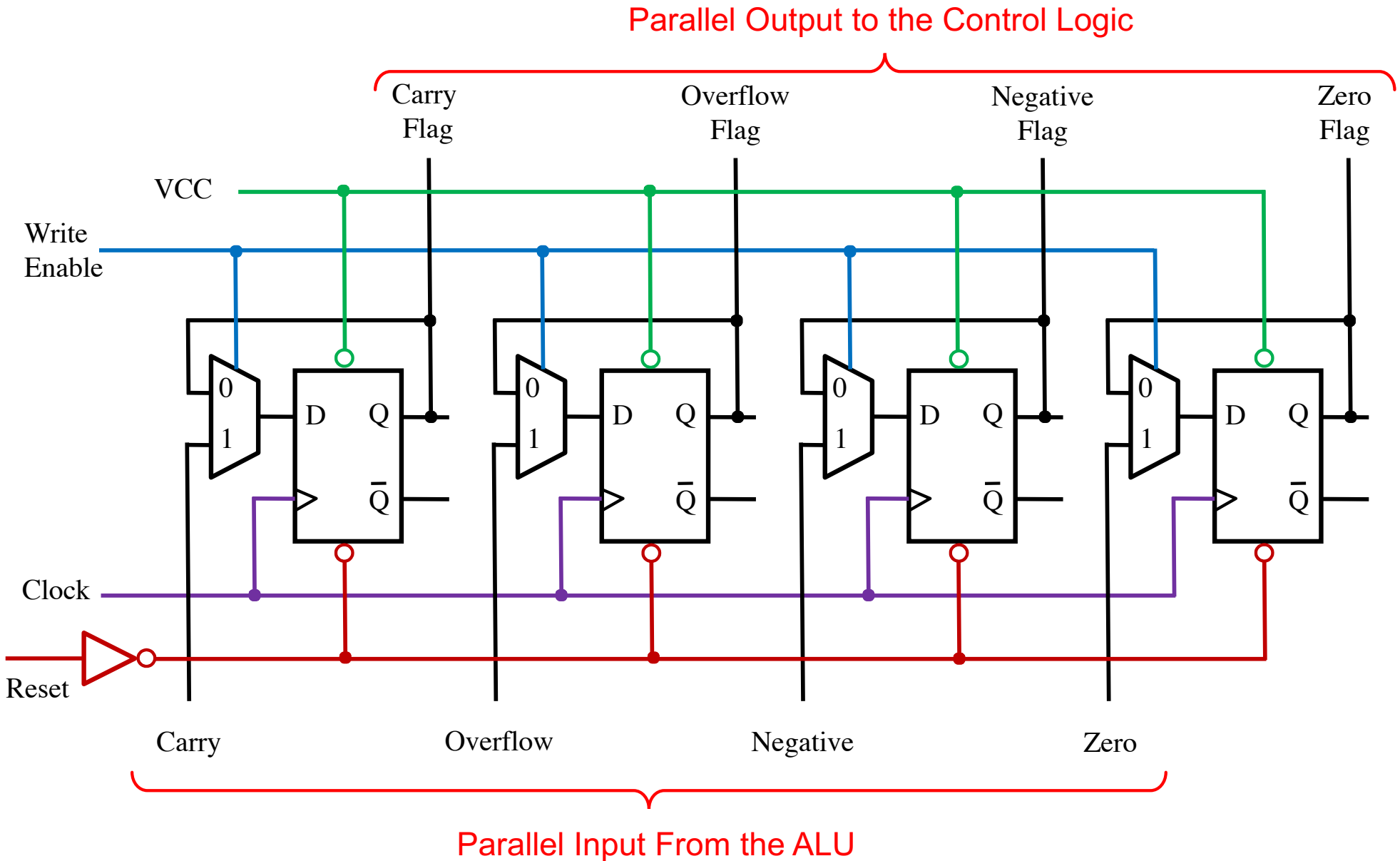
The Flags Register



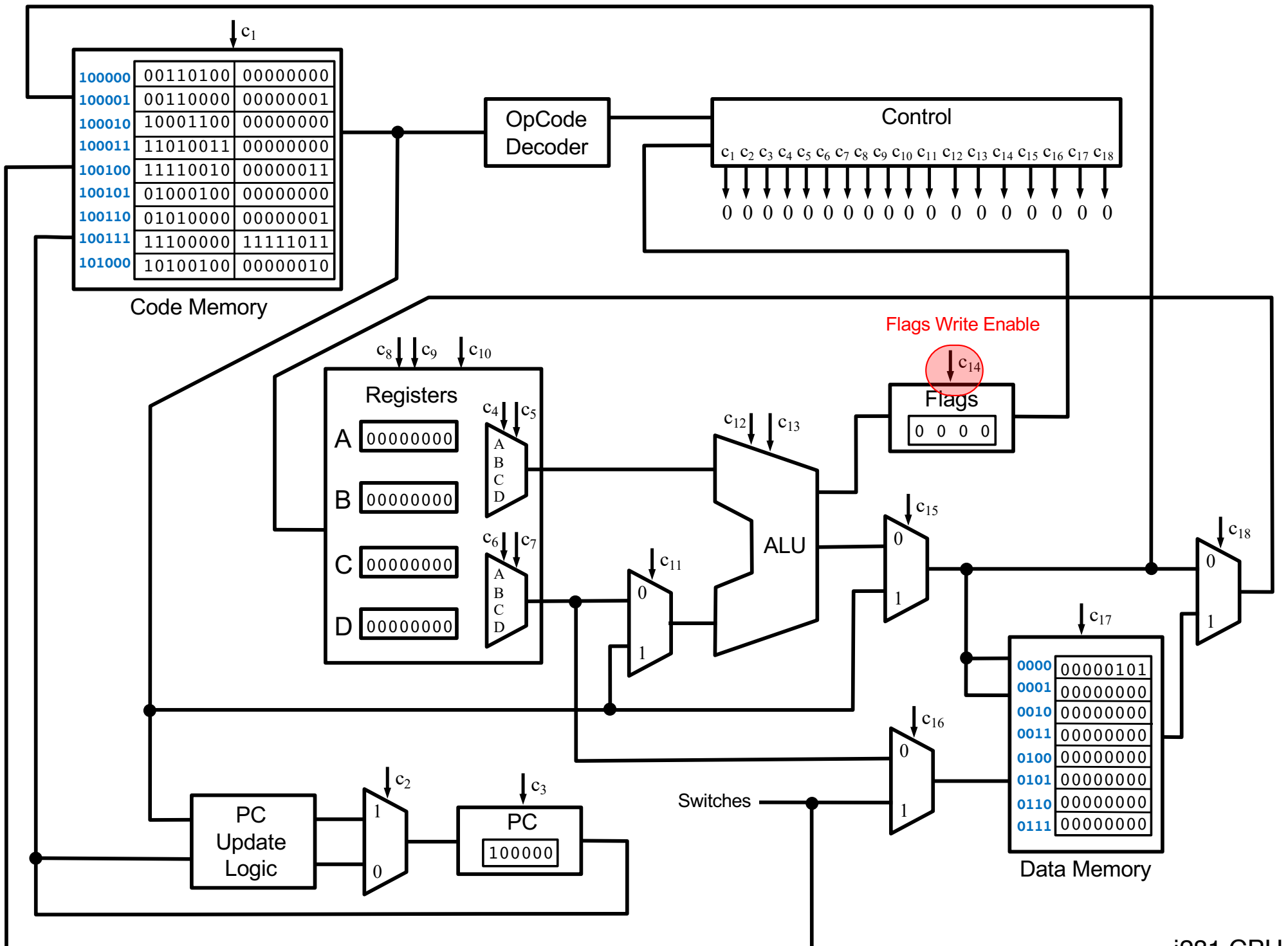
The Flags Register



The Flags Register



**Only Seven OPCODEs
Update the Flags Register**



	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
	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															

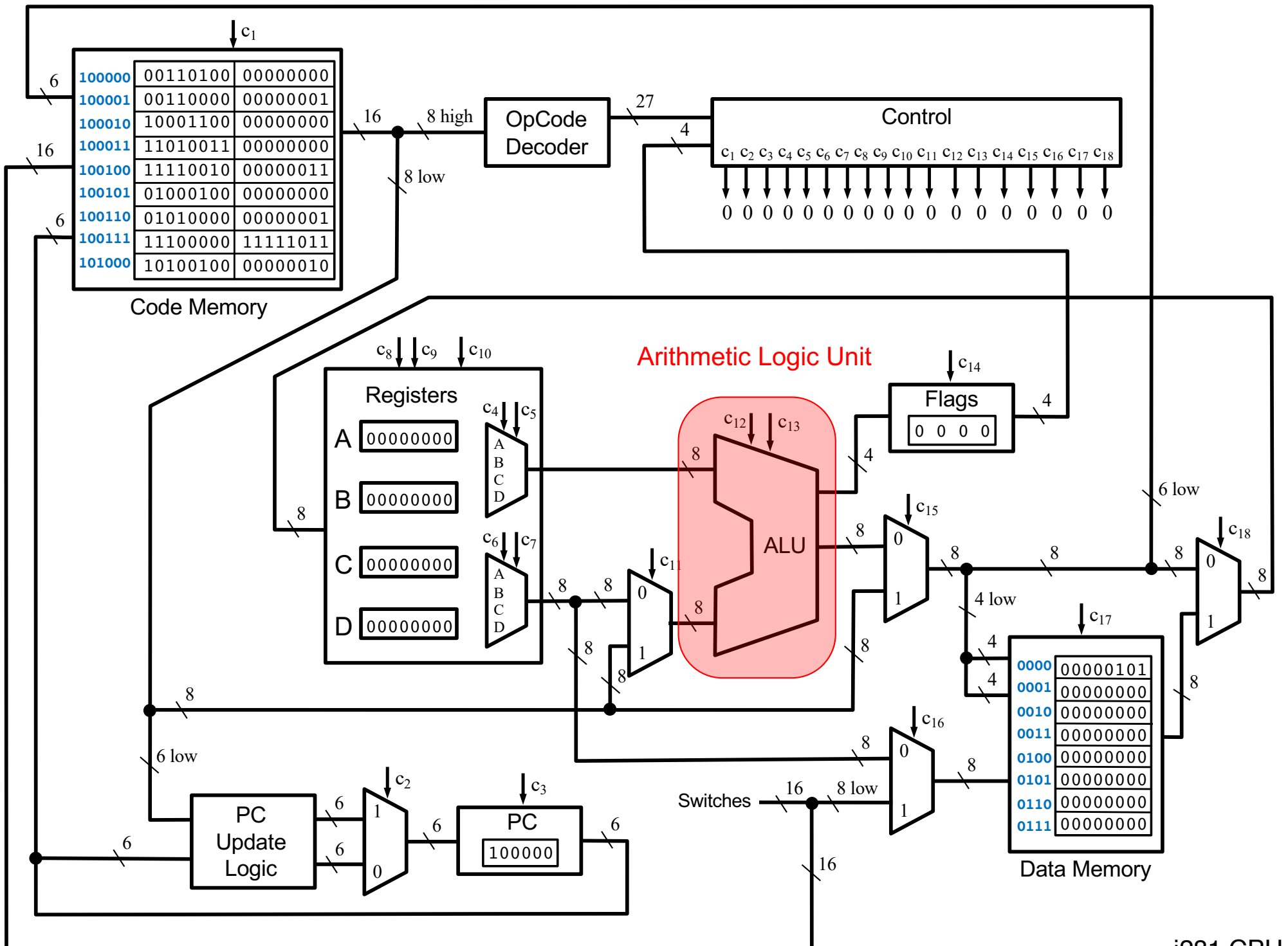
c₁₄ is set to 1 for opcodes that update the flags register. Only 7 opcodes do that.

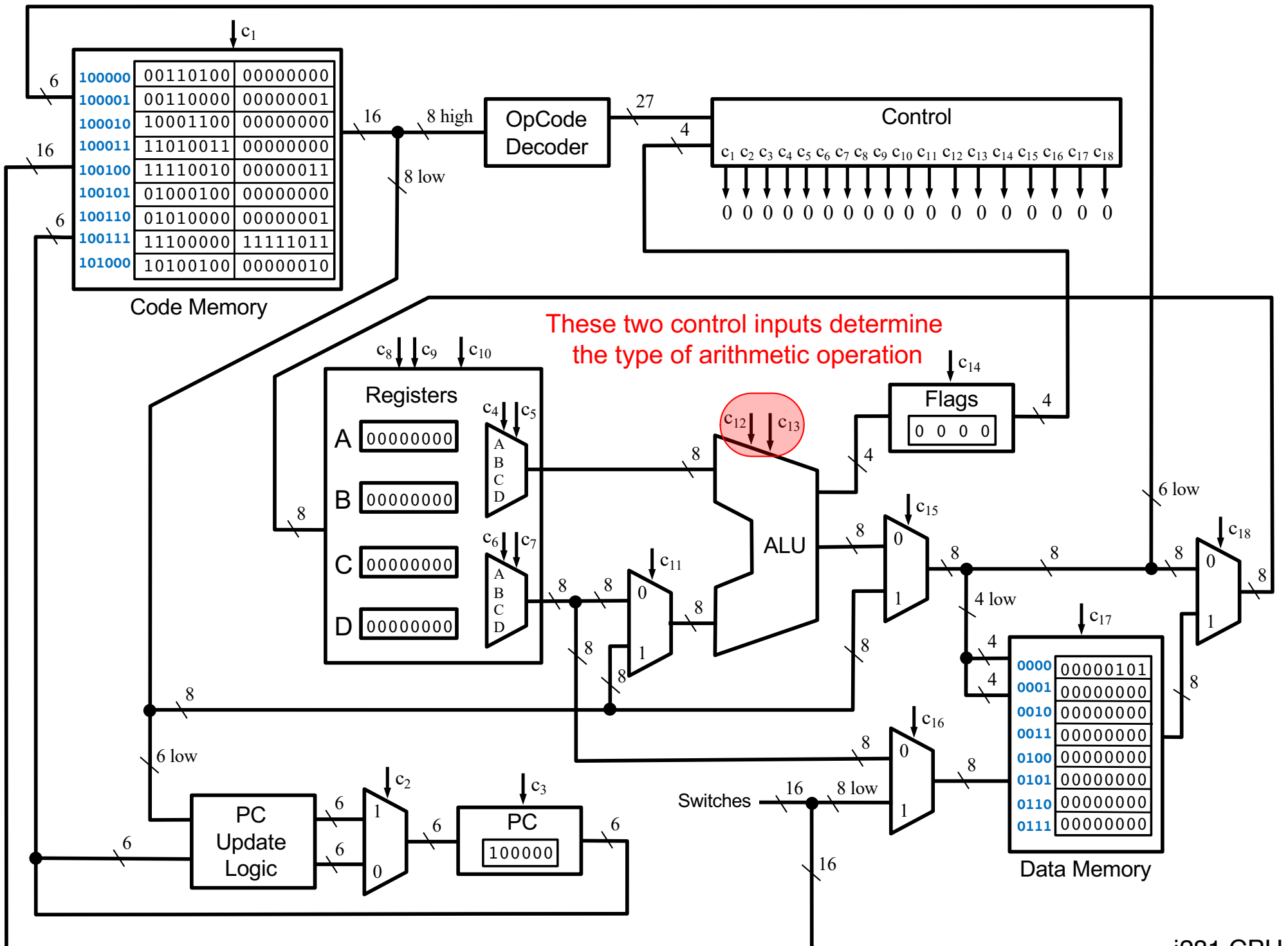
	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_EN	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	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															

c₁₄ is set to 1 for opcodes that update the flags register. Only 7 opcodes do that.

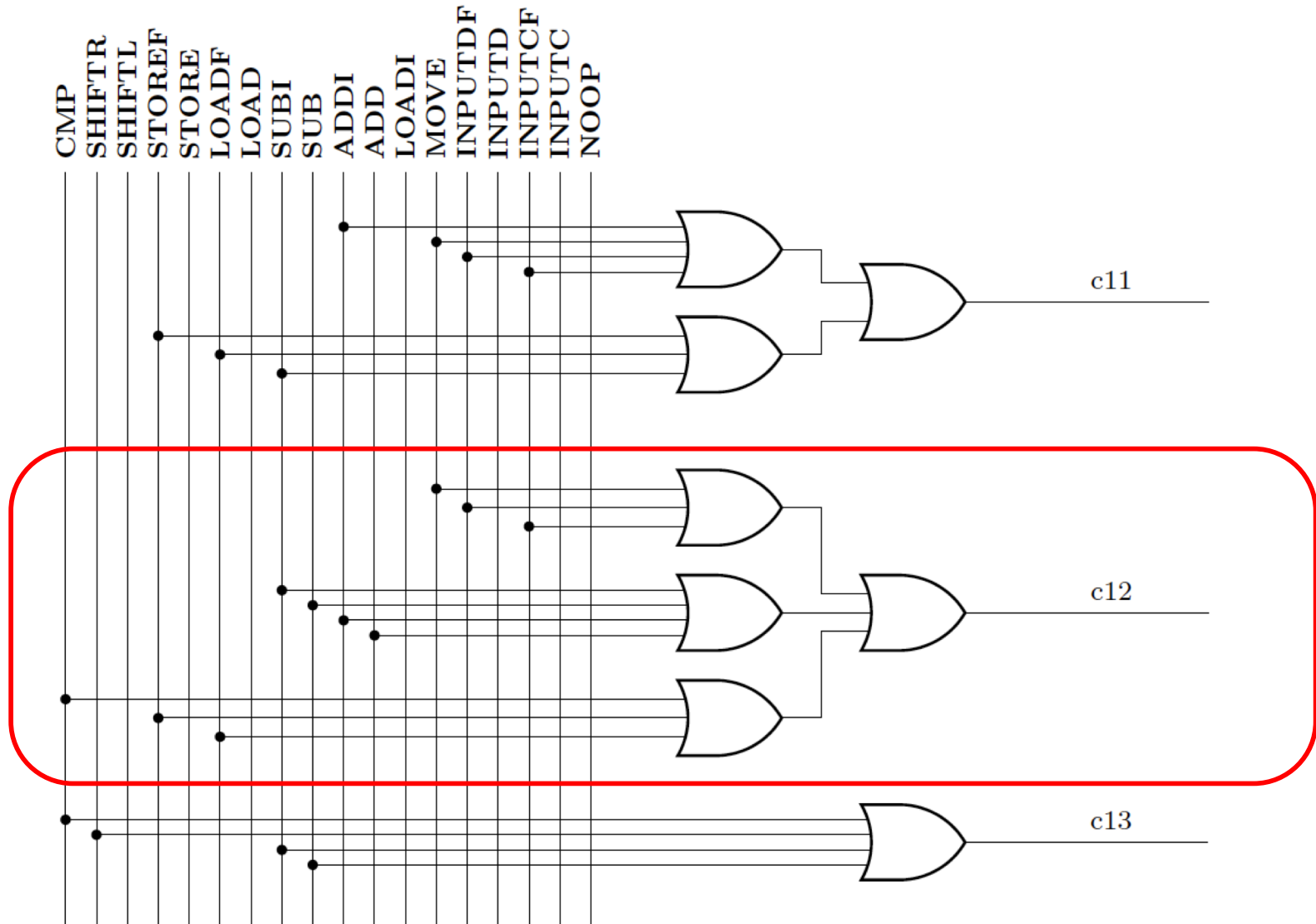
These 7 OPCODEs Update the Flags

NOOP	NO OPERATION
INPUTC	INPUT into Code memory
INPUTCF	INPUT into Code memory with offset
INPUTD	INPUT into Data memory
INPUTDF	INPUT into Data memory with offset
MOVE	MOVE the contents of one register into another
LOADI	LOAD Immediate value
LOADP	LOAD Pointer address
ADD	ADD two registers
ADDI	ADD an Immediate value to a register
SUB	SUBtract two registers
SUBI	SUBtract an Immediate value from a register
LOAD	LOAD from a data memory address into a register
LOADF	LOAD with an offset specified by another register
STORE	STORE a register into a data memory address
STOREF	STORE with an offset specified by another register
SHIFTL	SHIFT Left all bits in a register
SHIFTR	SHIFT Right all bits in a register
CMP	COMPare the values in two registers
JUMP	JUMP unconditionally to a specified address
BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal

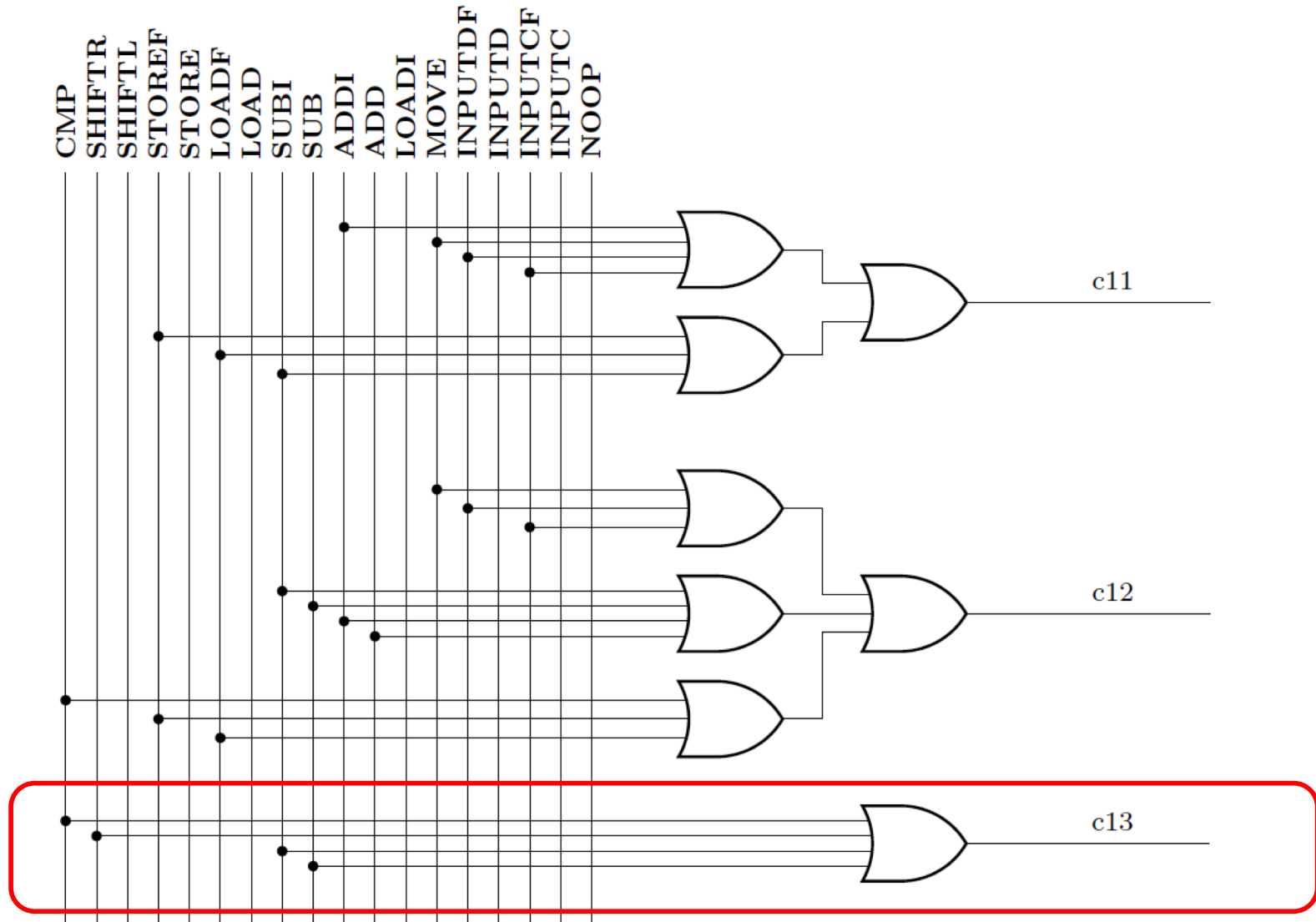




The Wiring Diagram for c_{12}



The Wiring Diagram for c_{13}



This ALU Can Perform 4 Operations

ALU_SELECT1	ALU_SELECT0	Operation
0	0	SHIFTL
0	1	SHIFTR
1	0	ADD
1	1	SUB/CMP

This ALU Can Perform 4 Operations

ALU_SELECT1	ALU_SELECT0	Operation
0	0	SHIFTL
0	1	SHIFTR
1	0	ADD
1	1	SUB/CMP

ADDI

SUBI

These two OpCodes
map to here as well

This ALU Can Perform 4 Operations

Names of these
control lines

C₁₂

C₁₃

ALU_SELECT1	ALU_SELECT0	Operation
0	0	SHIFTL
0	1	SHIFTR
1	0	ADD
1	1	SUB/CMP

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.

	c ₁	c ₂	c ₃	c ₄	c ₅	c ₆	c ₇	c ₈	c ₉	c ₁₀	c ₁₁	c ₁₂	c ₁₃	c ₁₄	c ₁₅	c ₁₆	c ₁₇	c ₁₈
	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															

c₁₂ and c₁₃ are set by these opcodes that need the ALU to compute something.

	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
	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		0	0	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															

SHIFTL also uses the ALU, but in this case the two control lines are both equal to zero.

	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
	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		0	0	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															

In fact, all of these other commands tell the ALU to shift left. The ALU is never idle!
But they ignore the result!

**The MOVE command
is implemented as addition with 0**

	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
	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															

The MOVE command copies the contents of one register into another. But it tells the ALU to do addition?! Why?

MOVE

Full name:

MOVE

Description:

Move (i.e., copy) the contents of the second register into the first register, overwriting the first register.

Assembly Example:

MOVE A, B

Instruction Layout:

0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

MOVE

Full name:

MOVE

Description:

Move (i.e., copy) the contents of the second register into the first register, overwriting the first register.

Assembly Example:

MOVE A, B

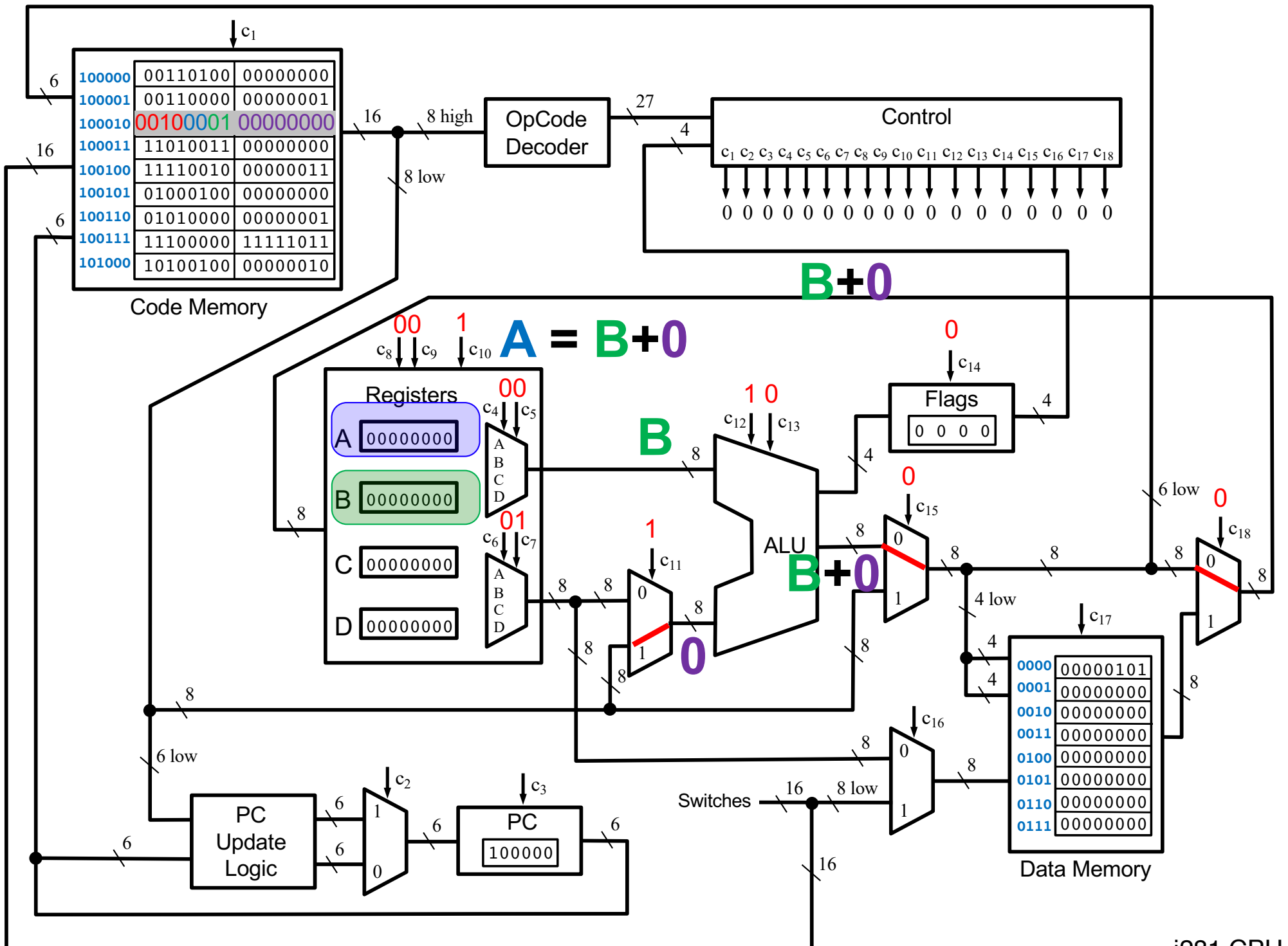
Instruction Layout:

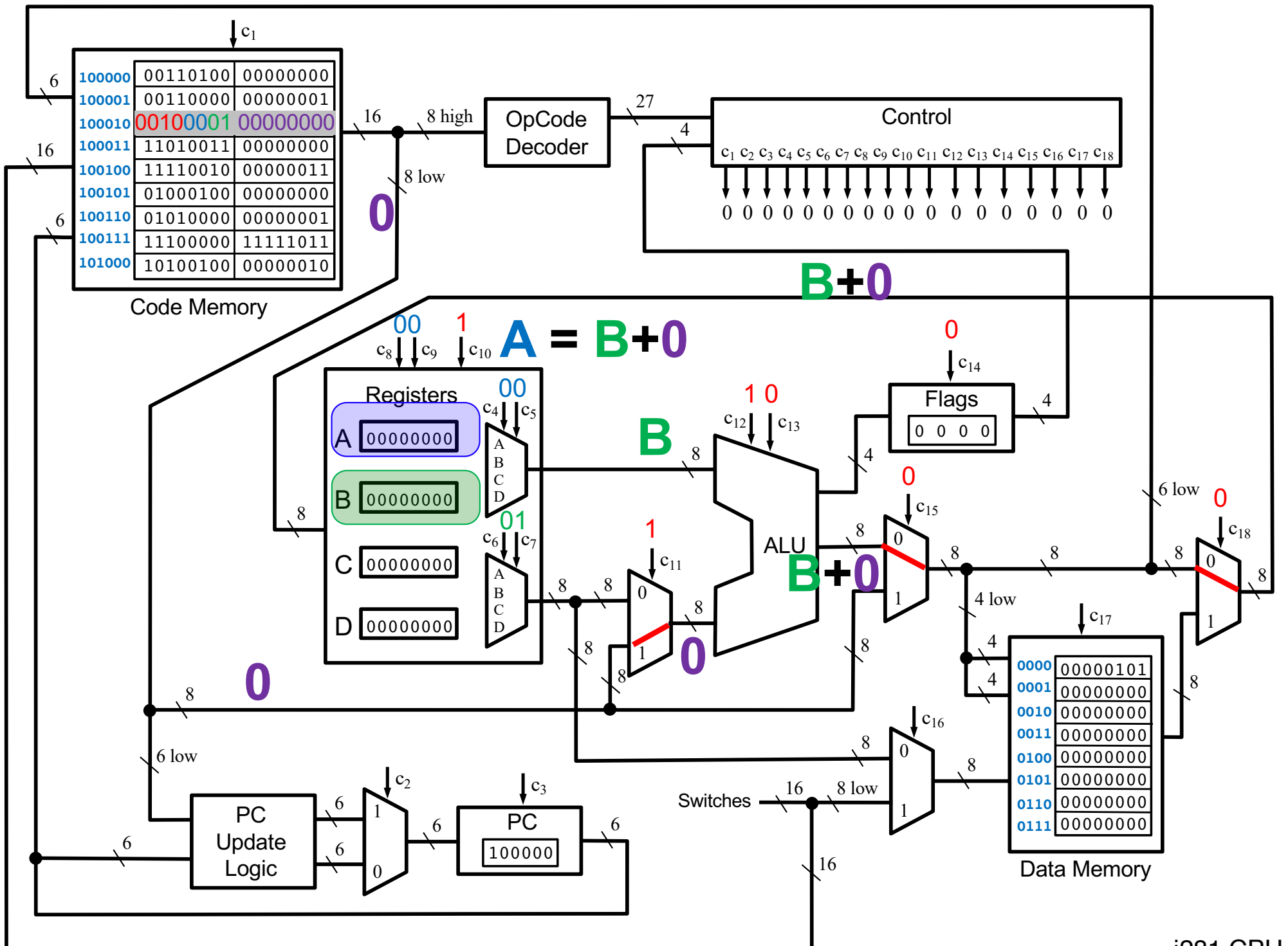
$$A = B + 0$$

0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

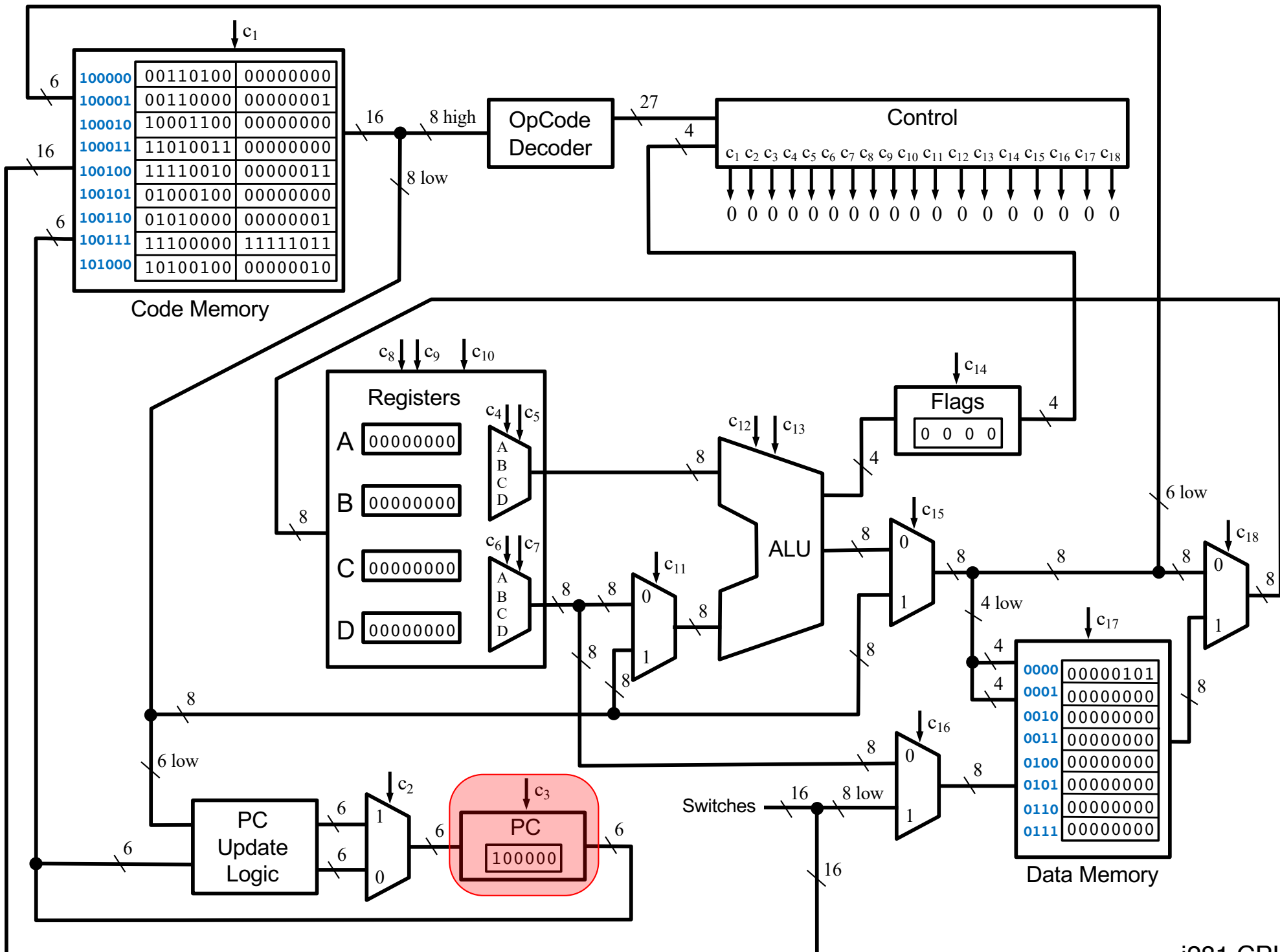
	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_EN	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	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															

The MOVE command copies the contents of one register into another. But it tells the ALU to do addition?! Why?

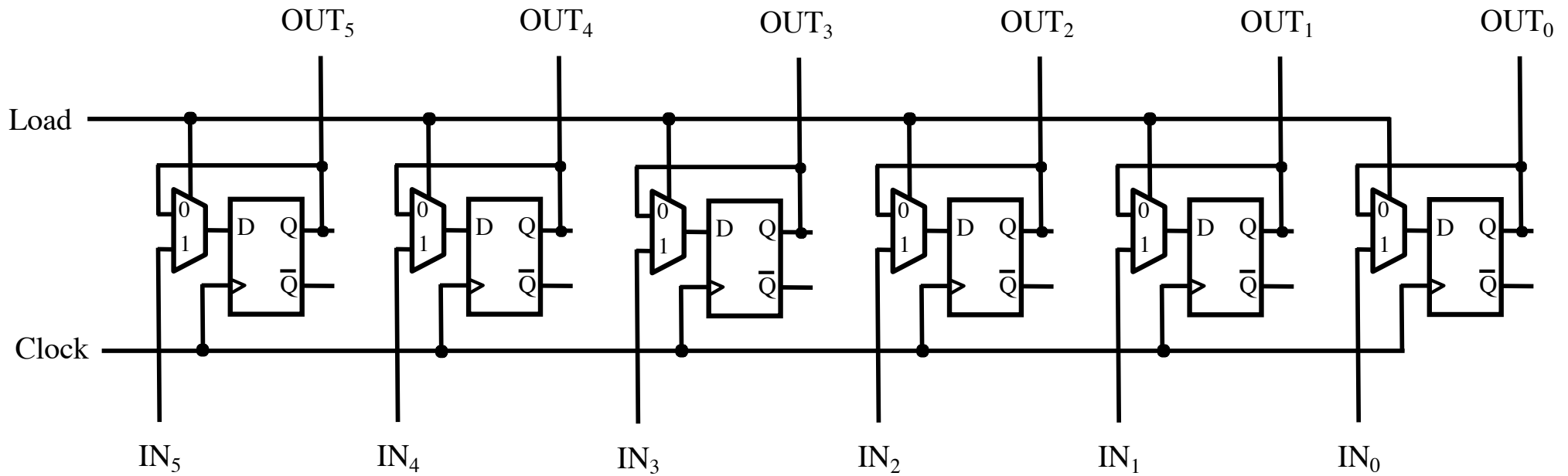




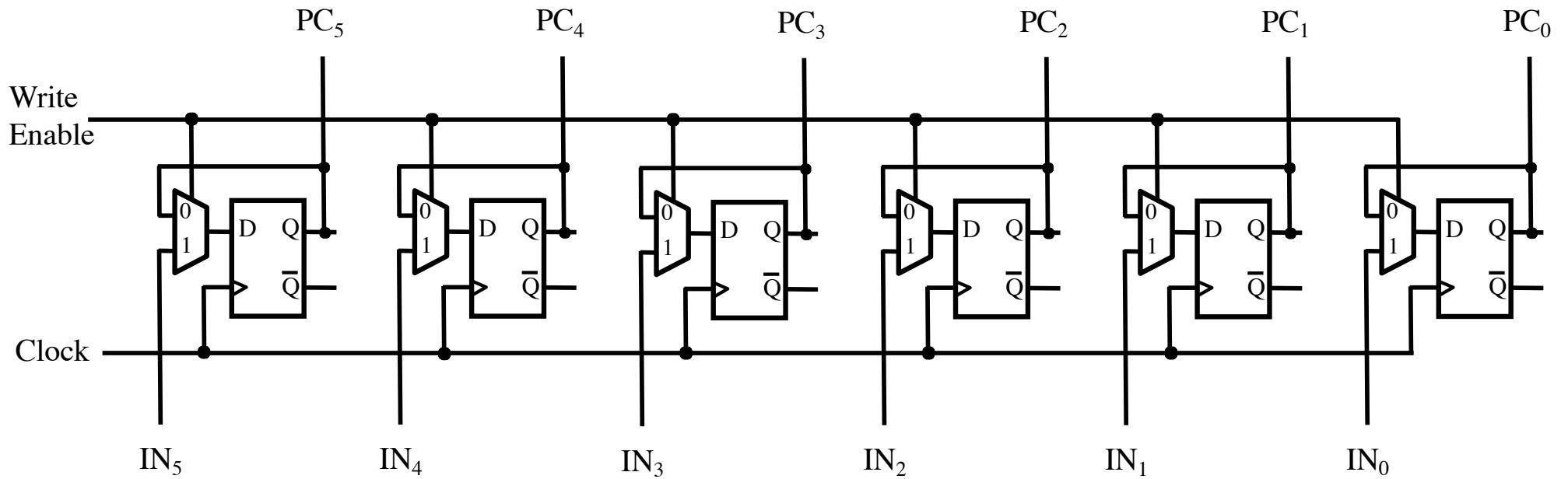
The Program Counter (PC)



6-Bit Parallel-Access Register

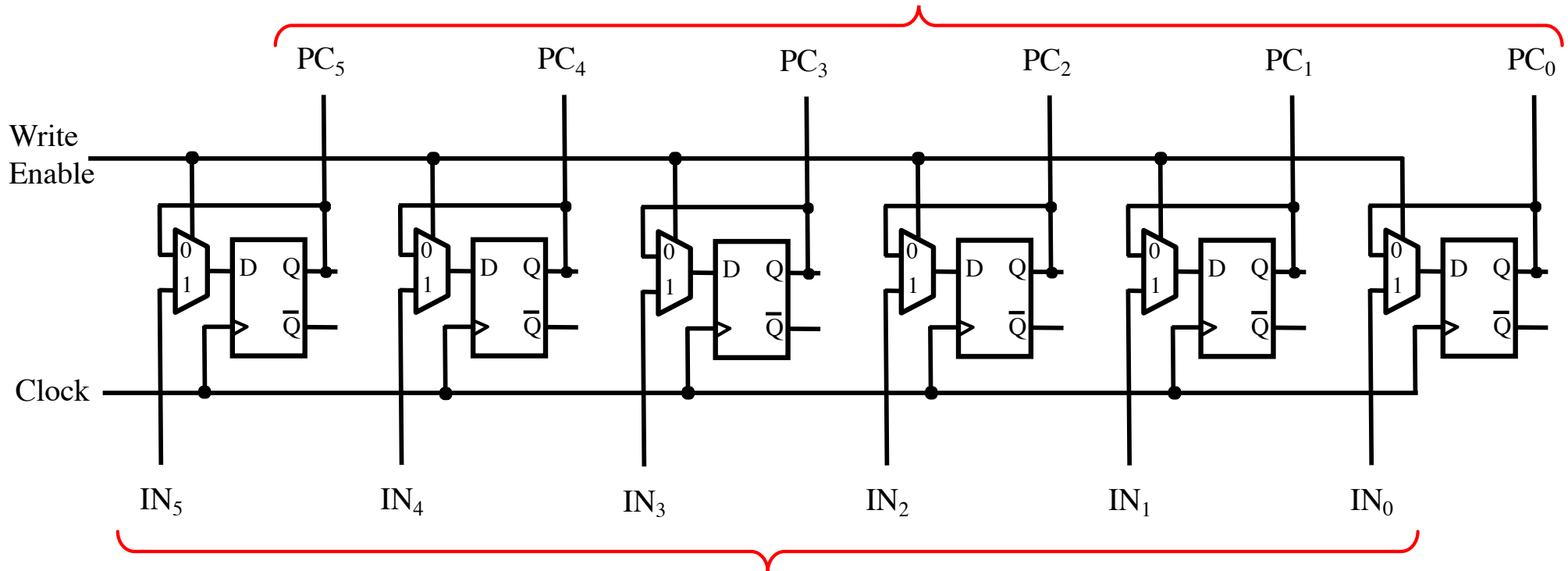


The Program Counter Register



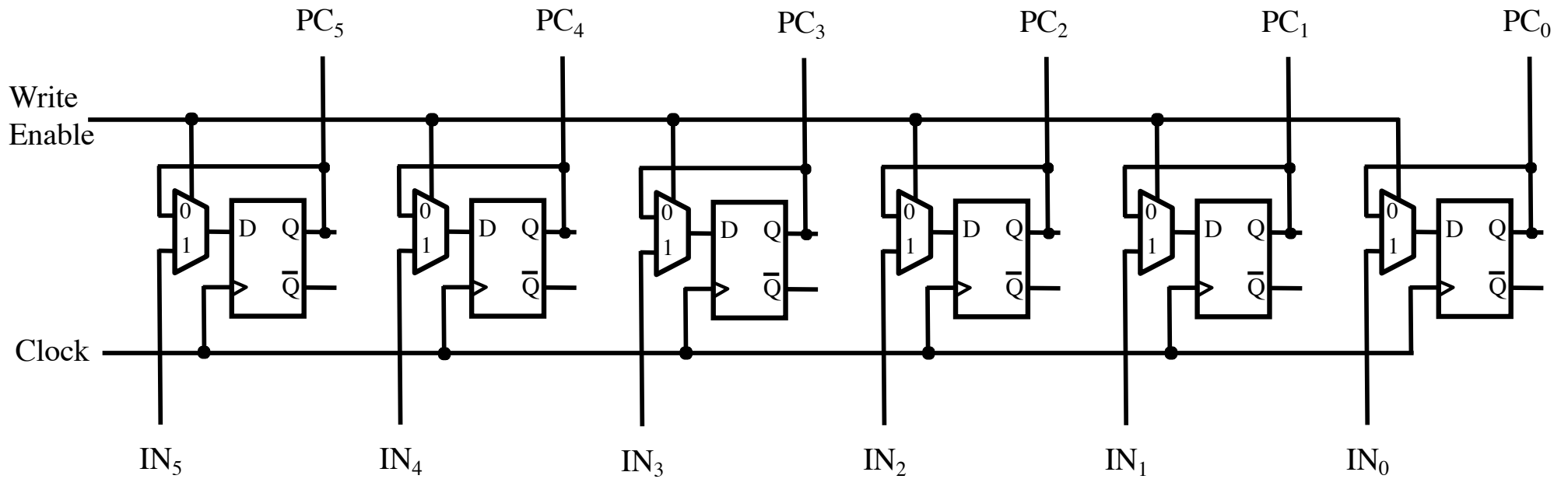
The Program Counter Register

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

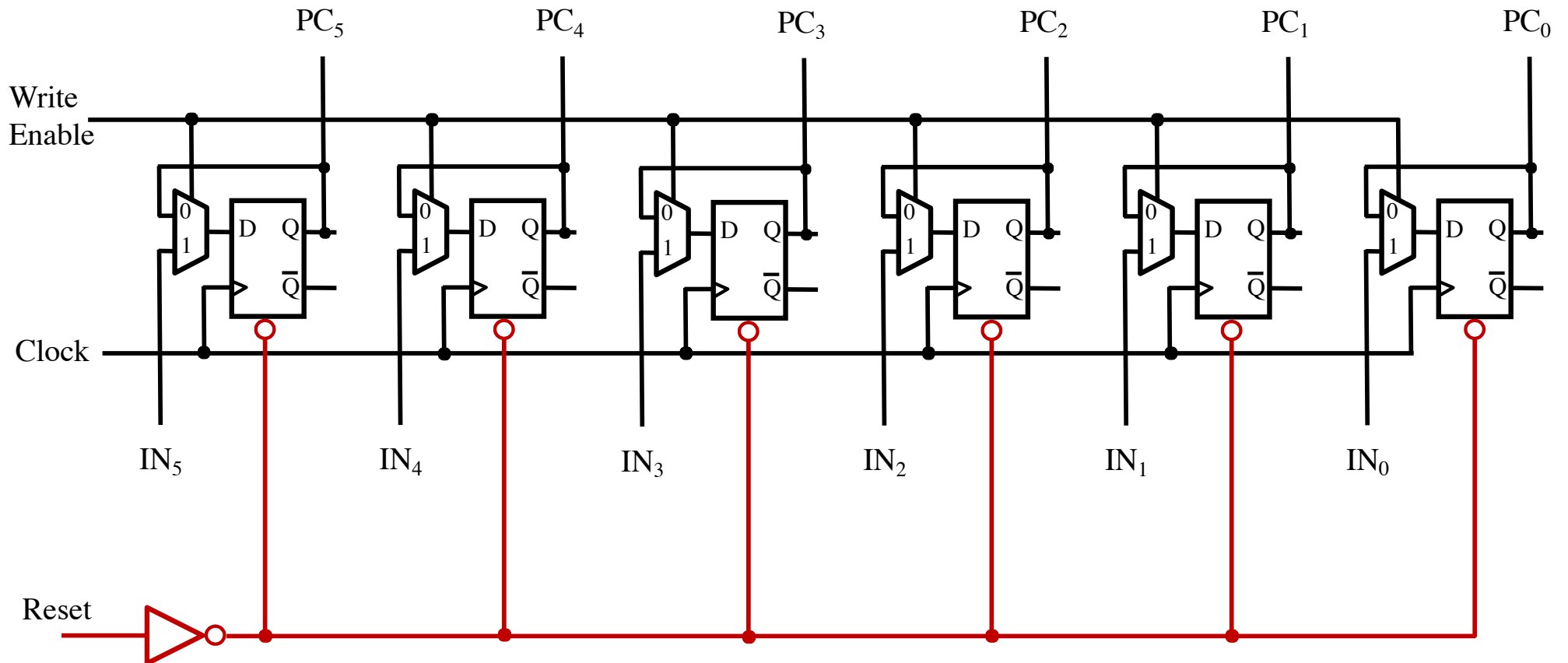


Parallel Input From the PC Update Logic

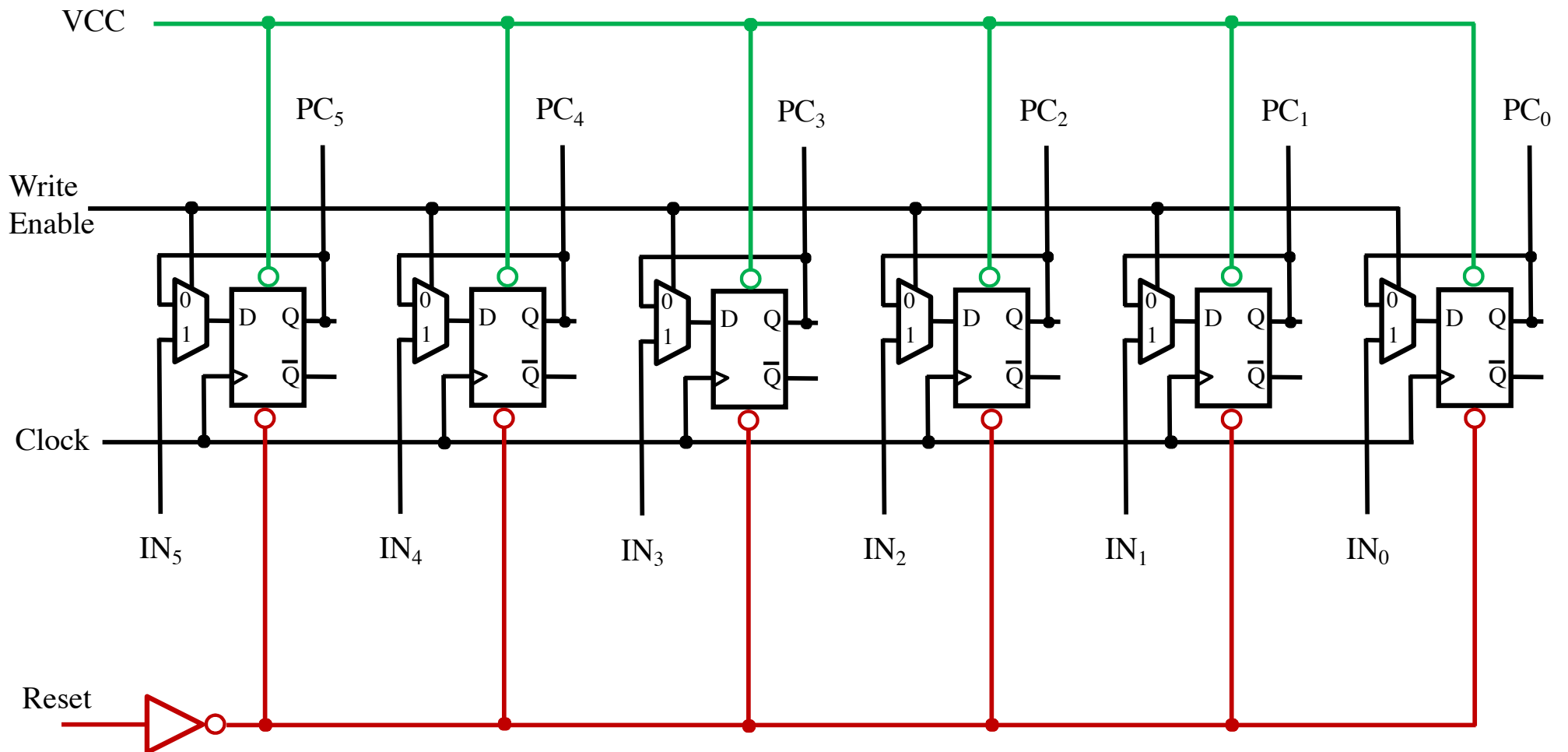
The Program Counter Register



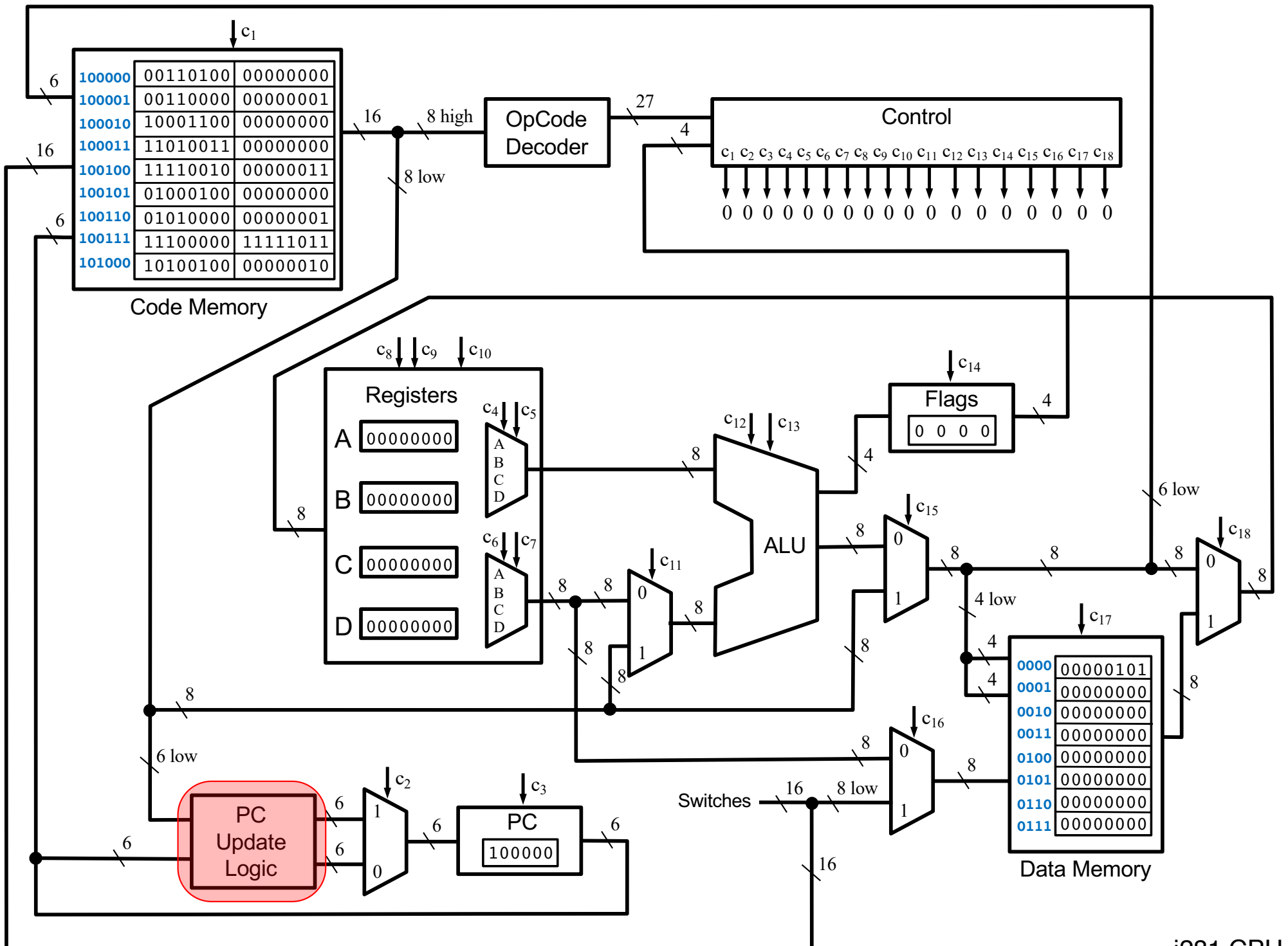
The Program Counter Register

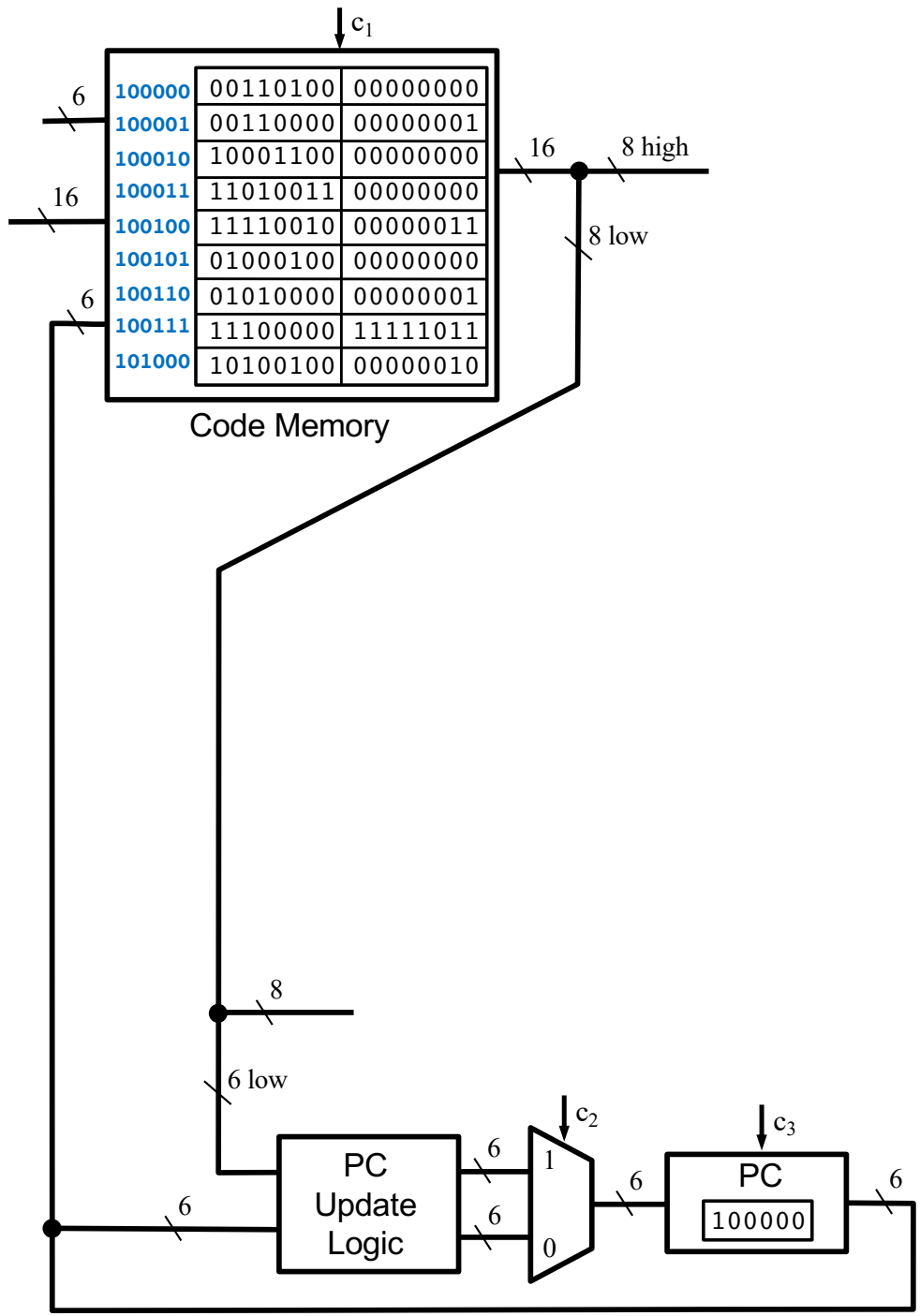


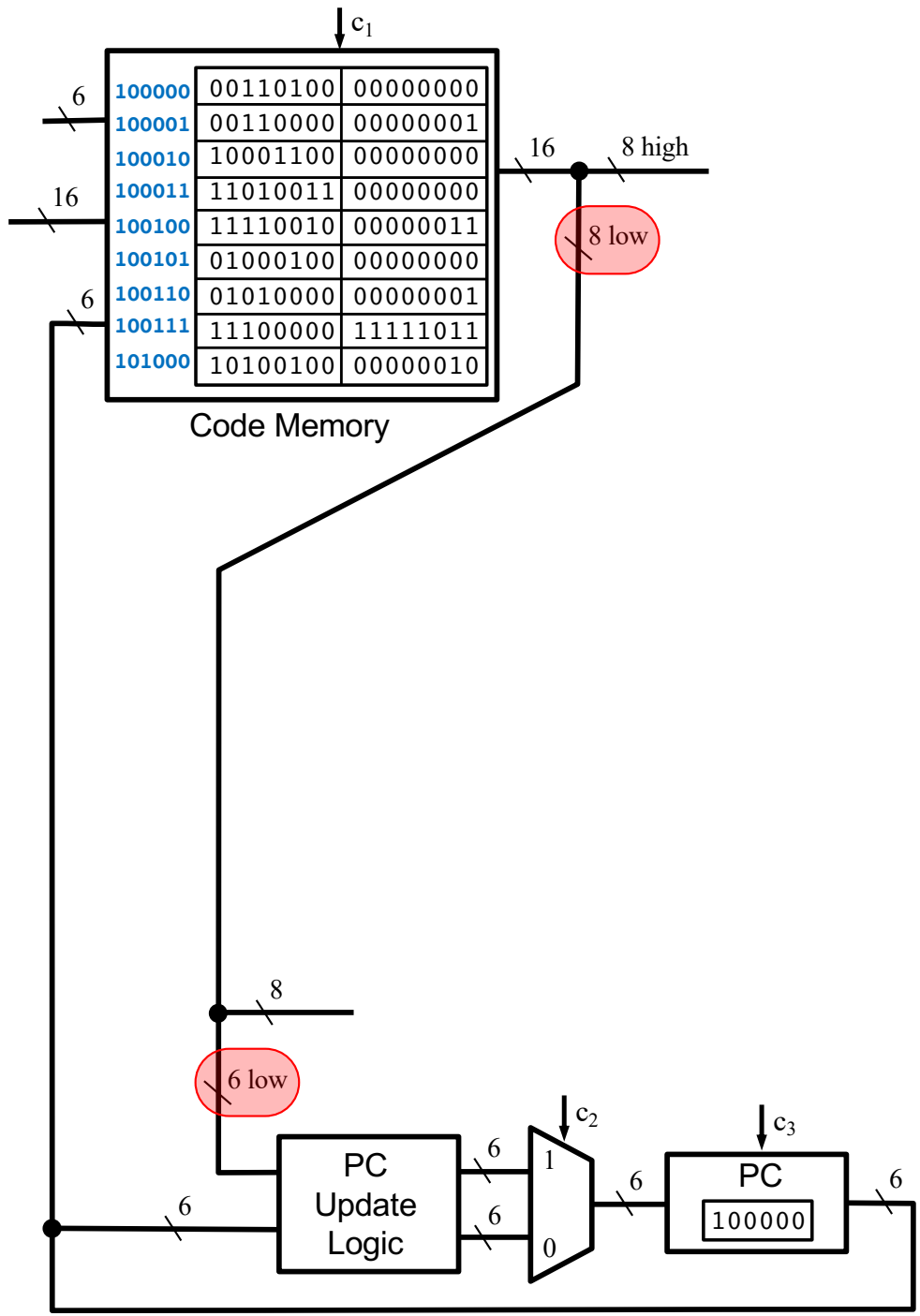
The Program Counter Register

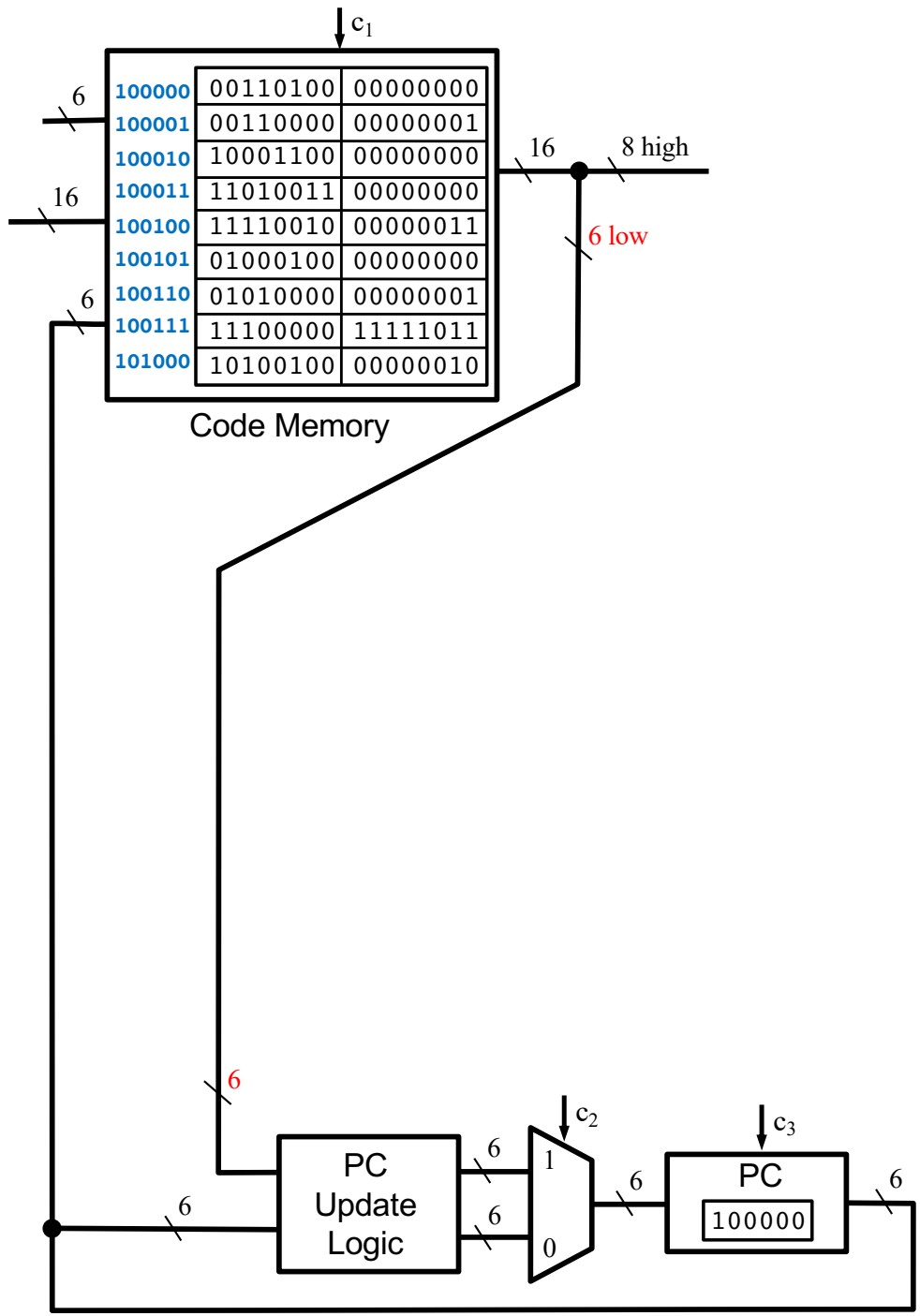


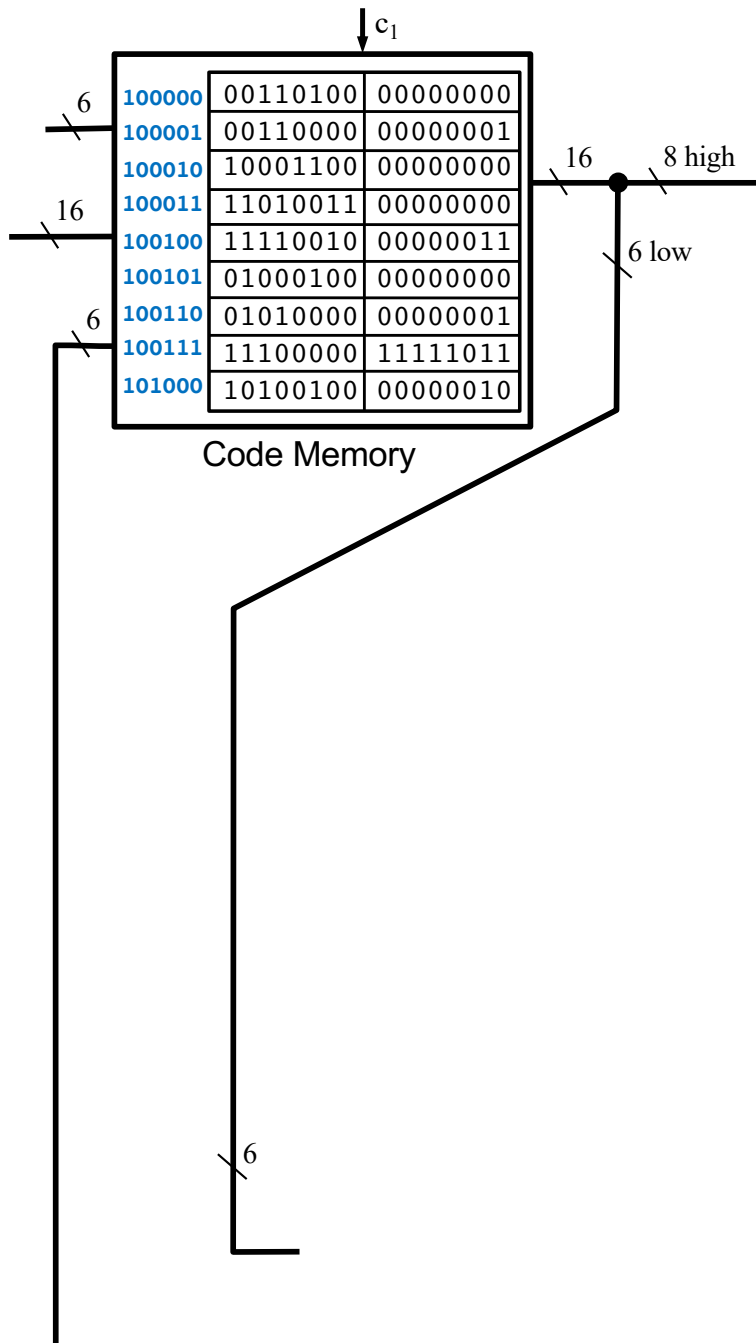
The Program Counter Update Logic

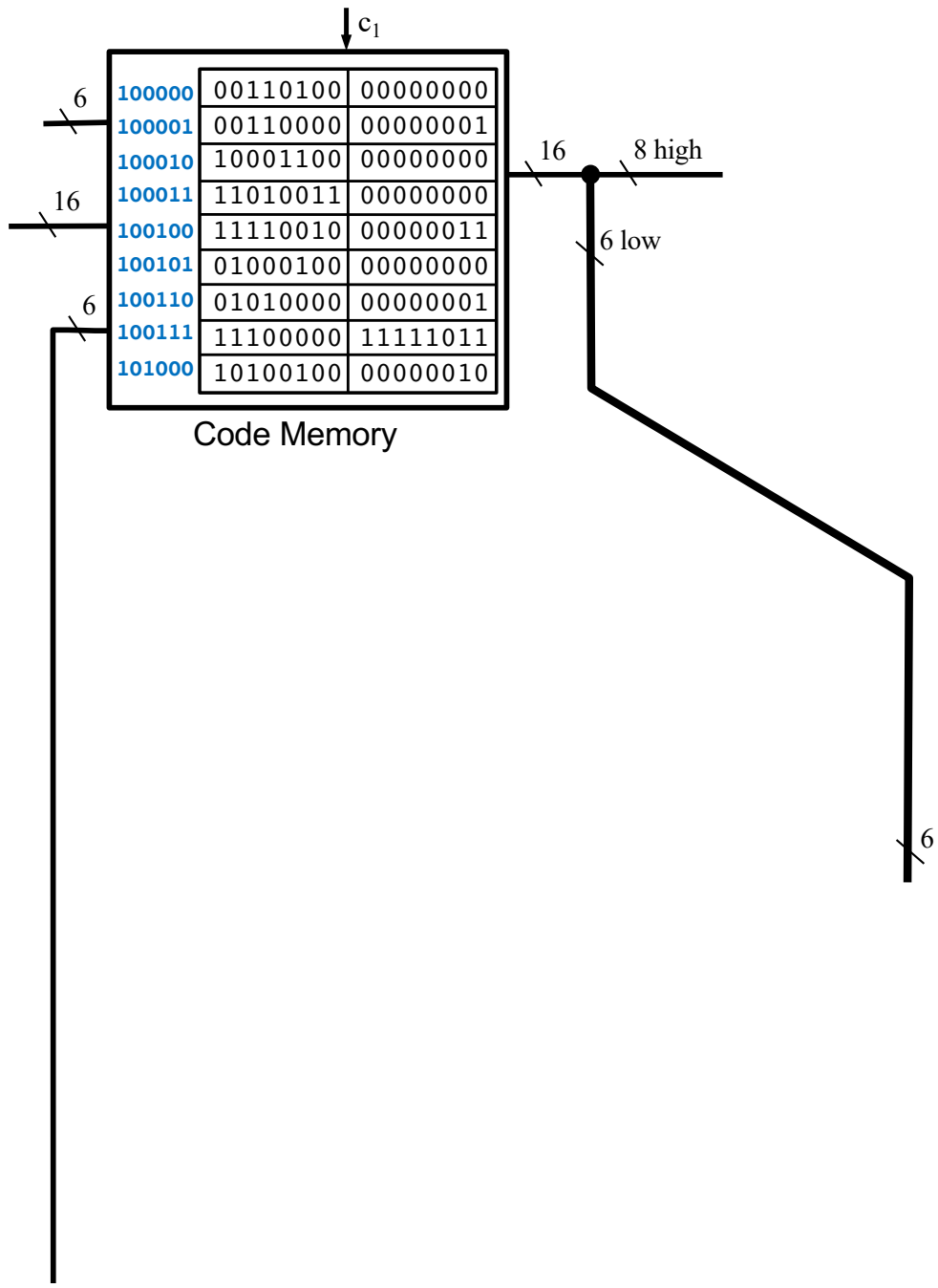


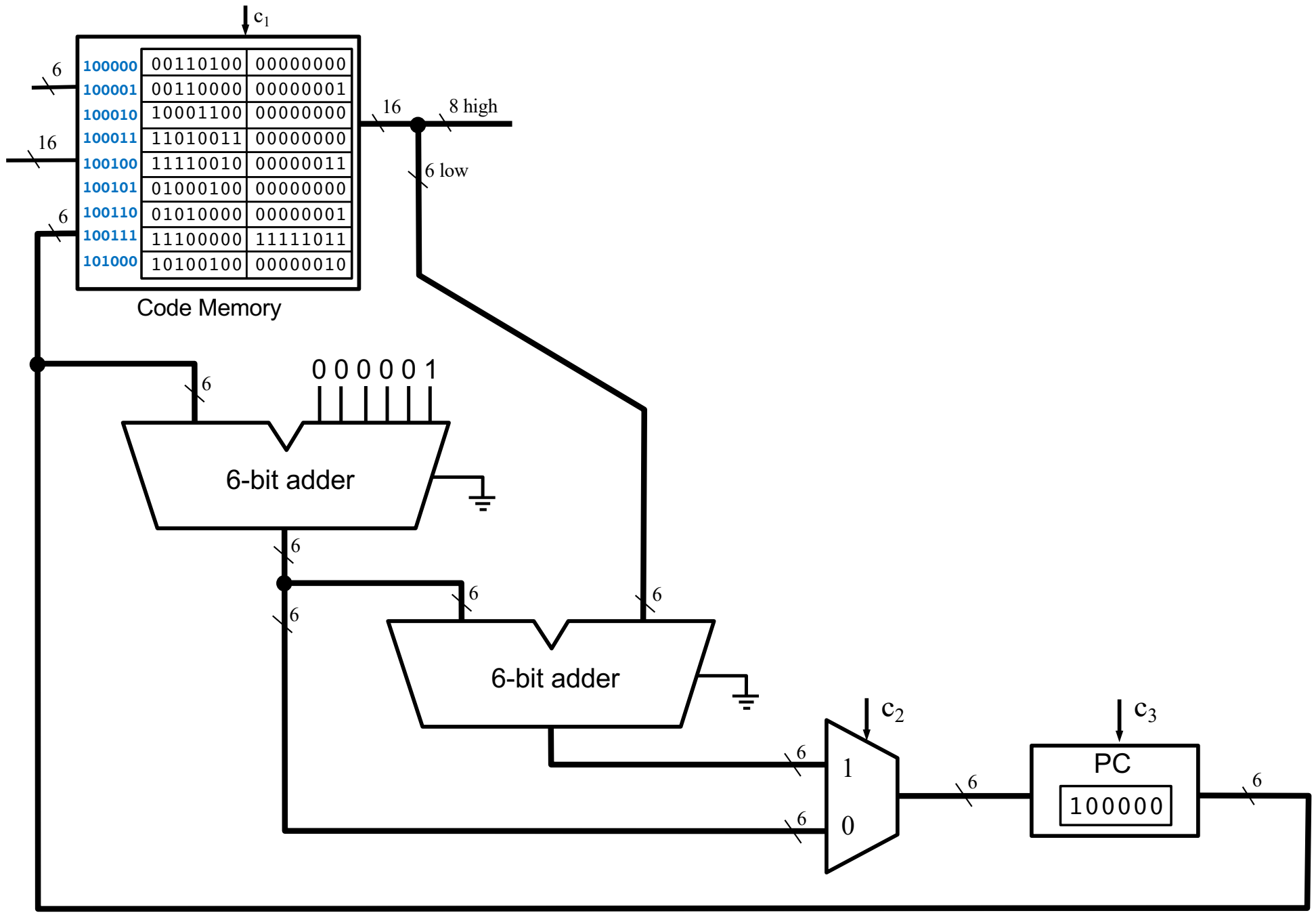


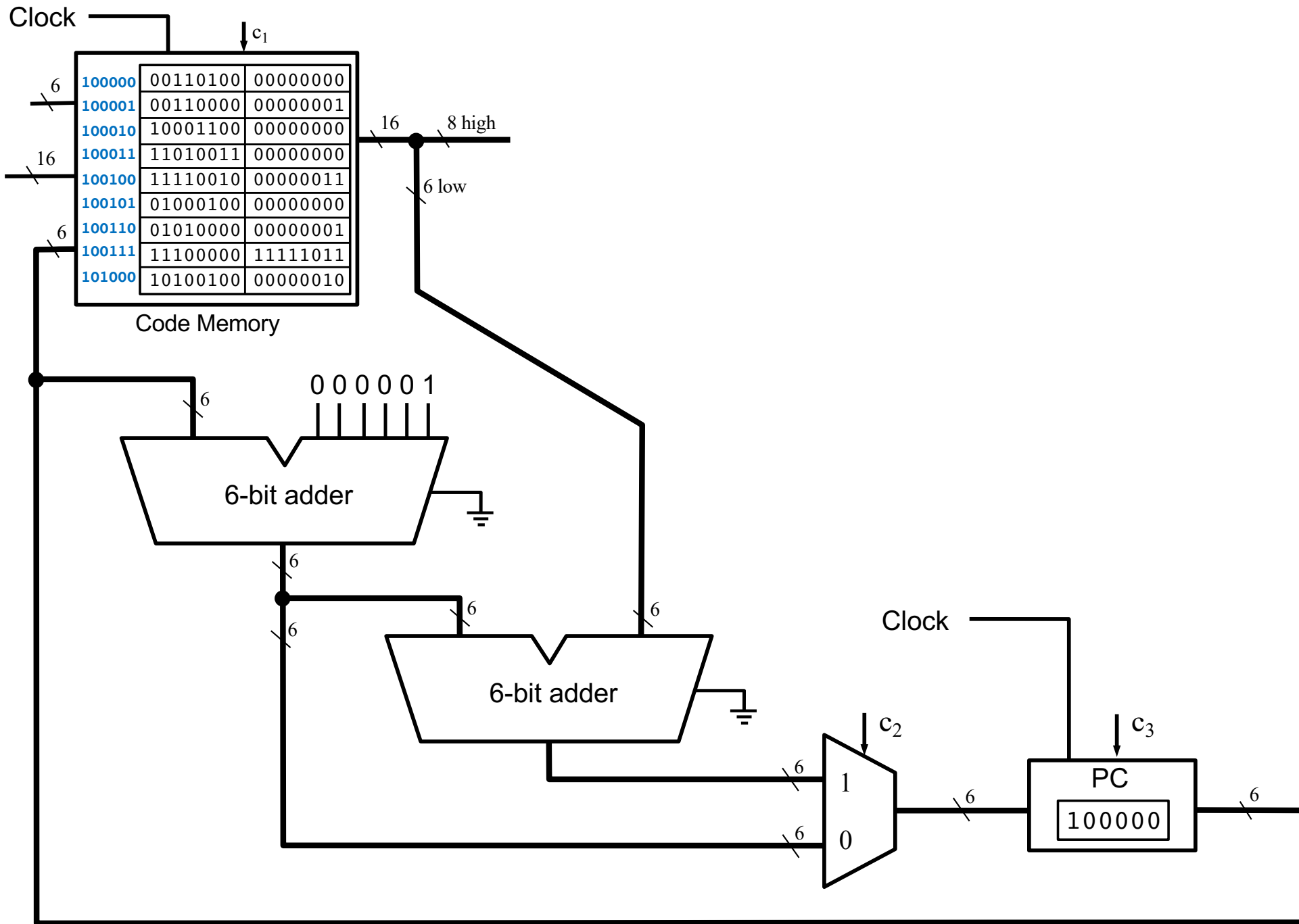


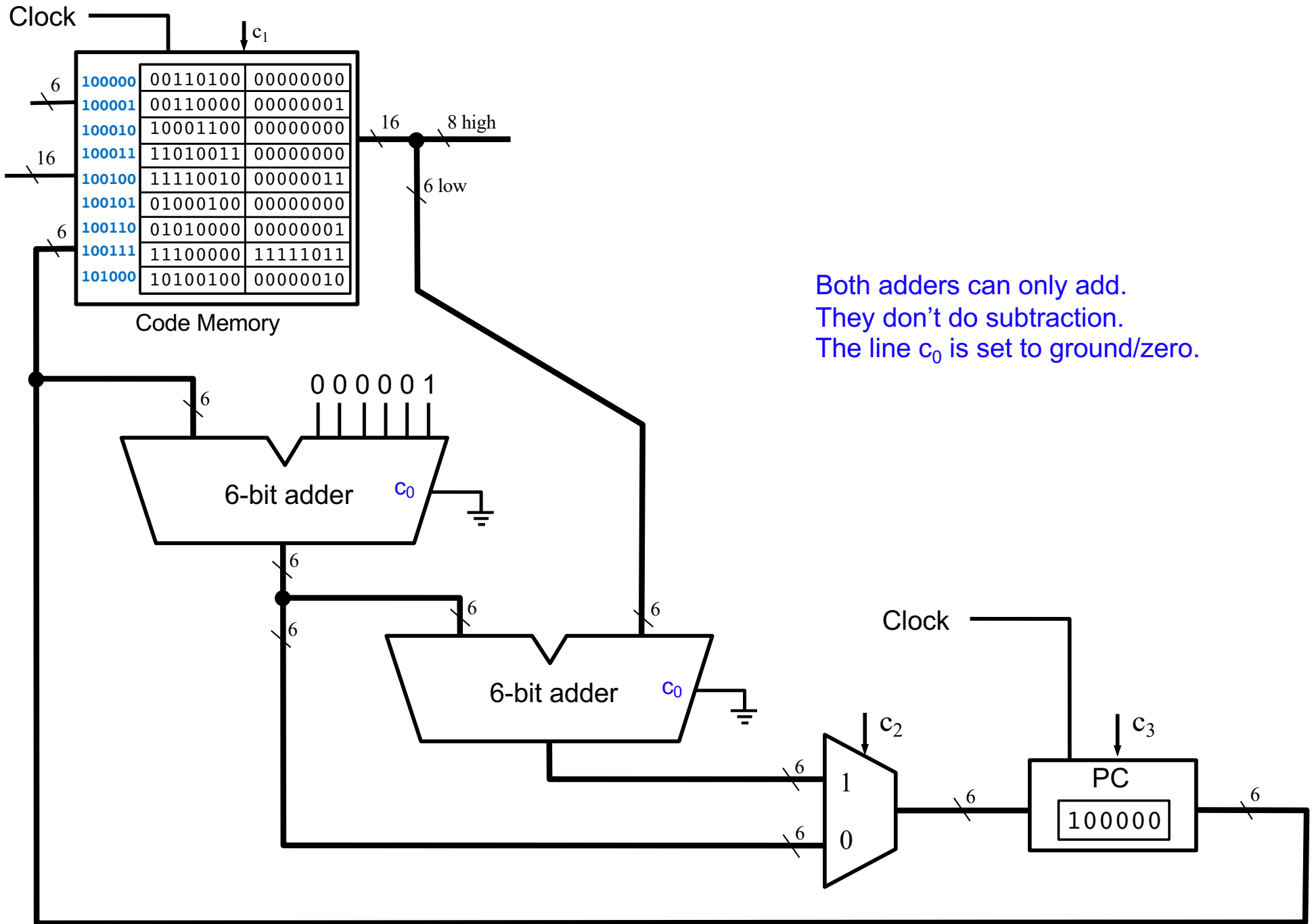




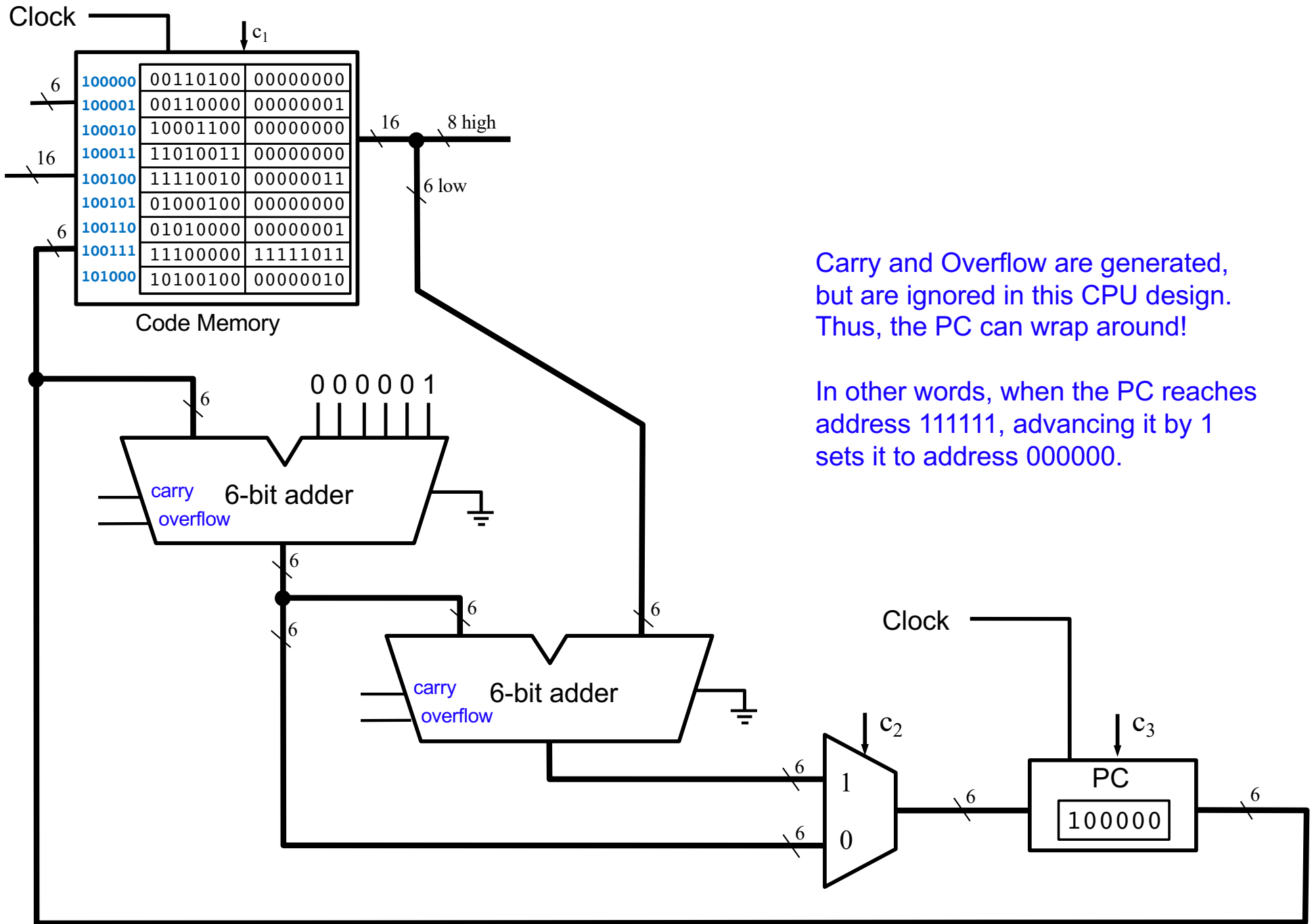






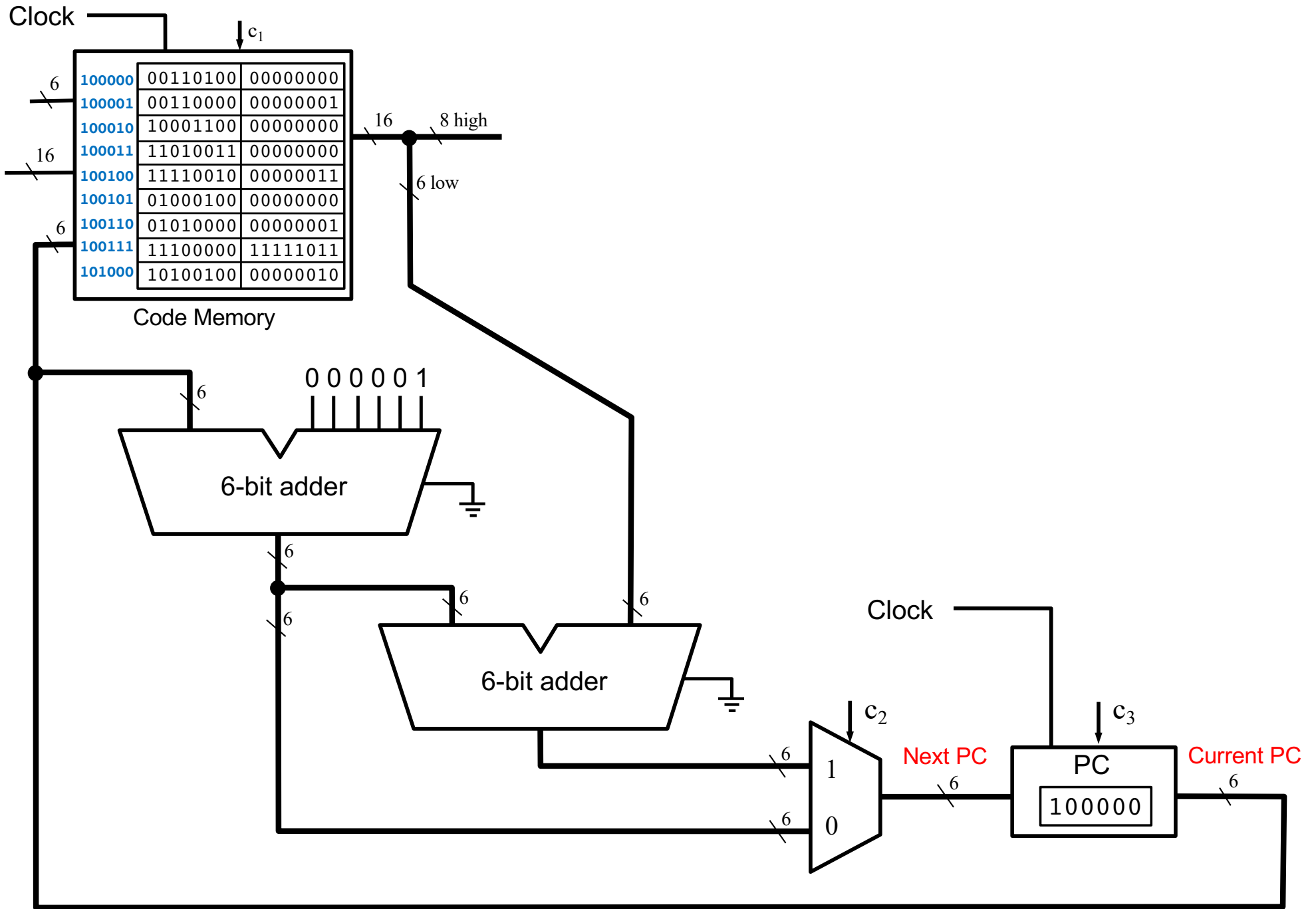


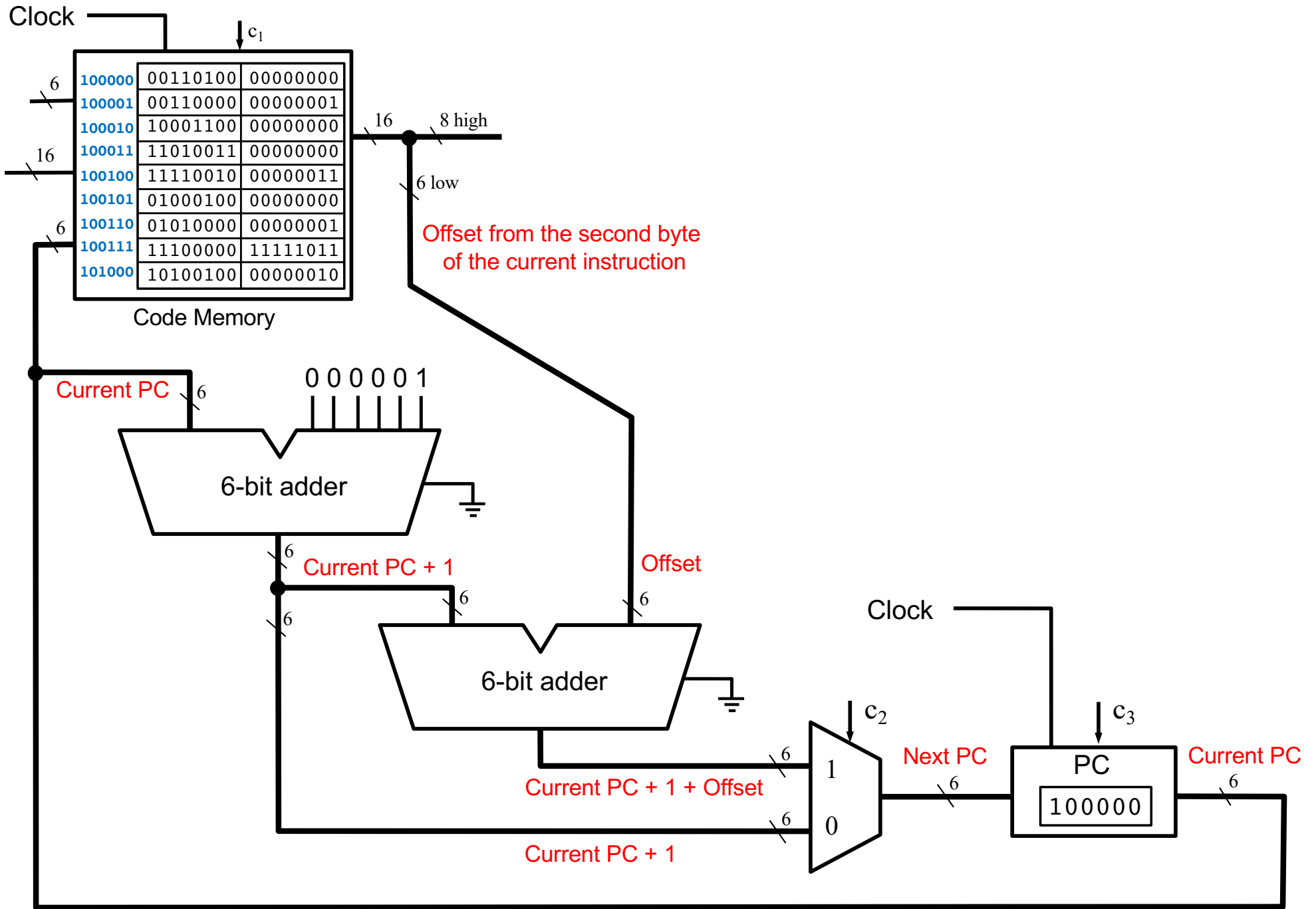
Both adders can only add.
They don't do subtraction.
The line c_0 is set to ground/zero.

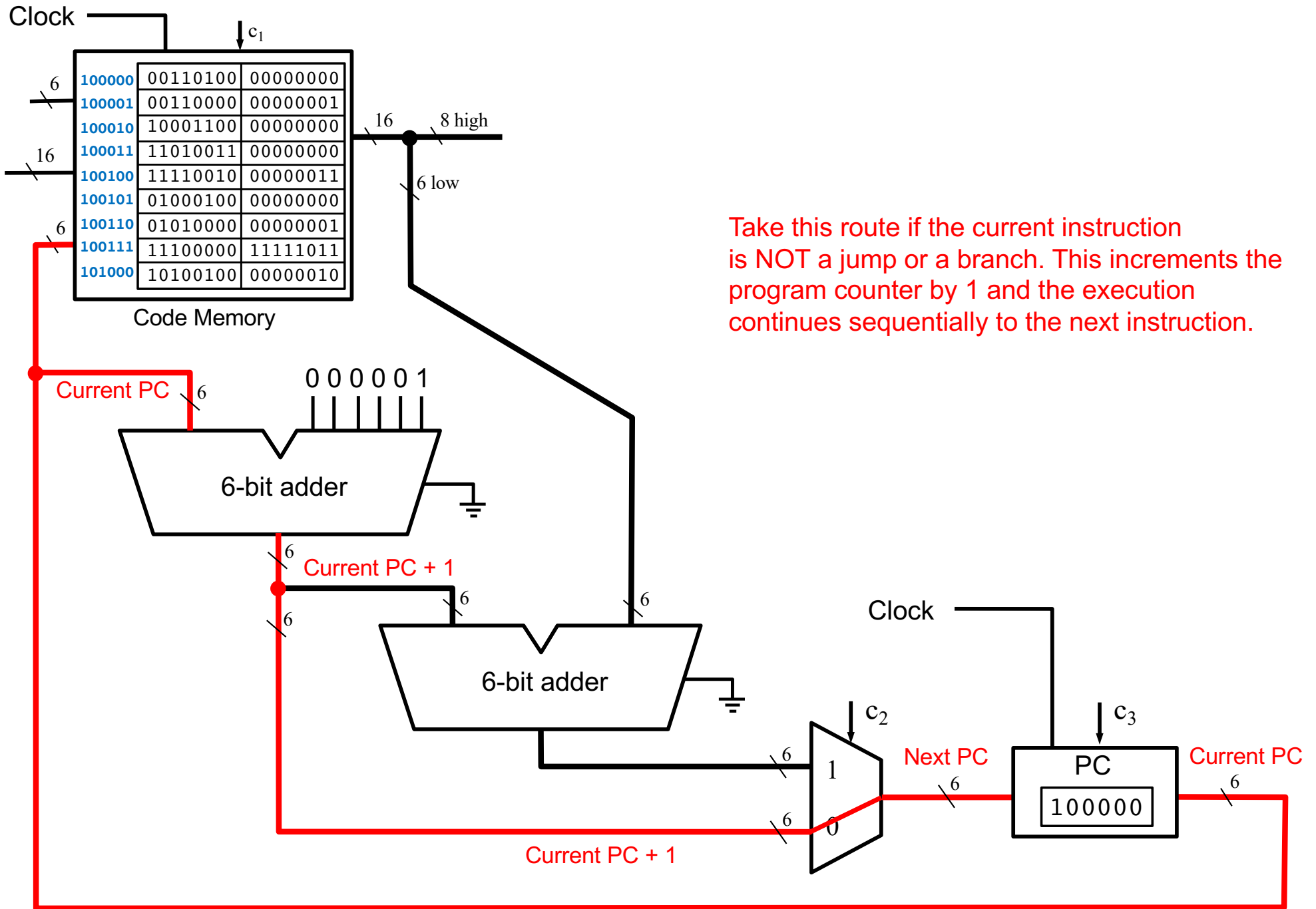


Carry and Overflow are generated, but are ignored in this CPU design. Thus, the PC can wrap around!

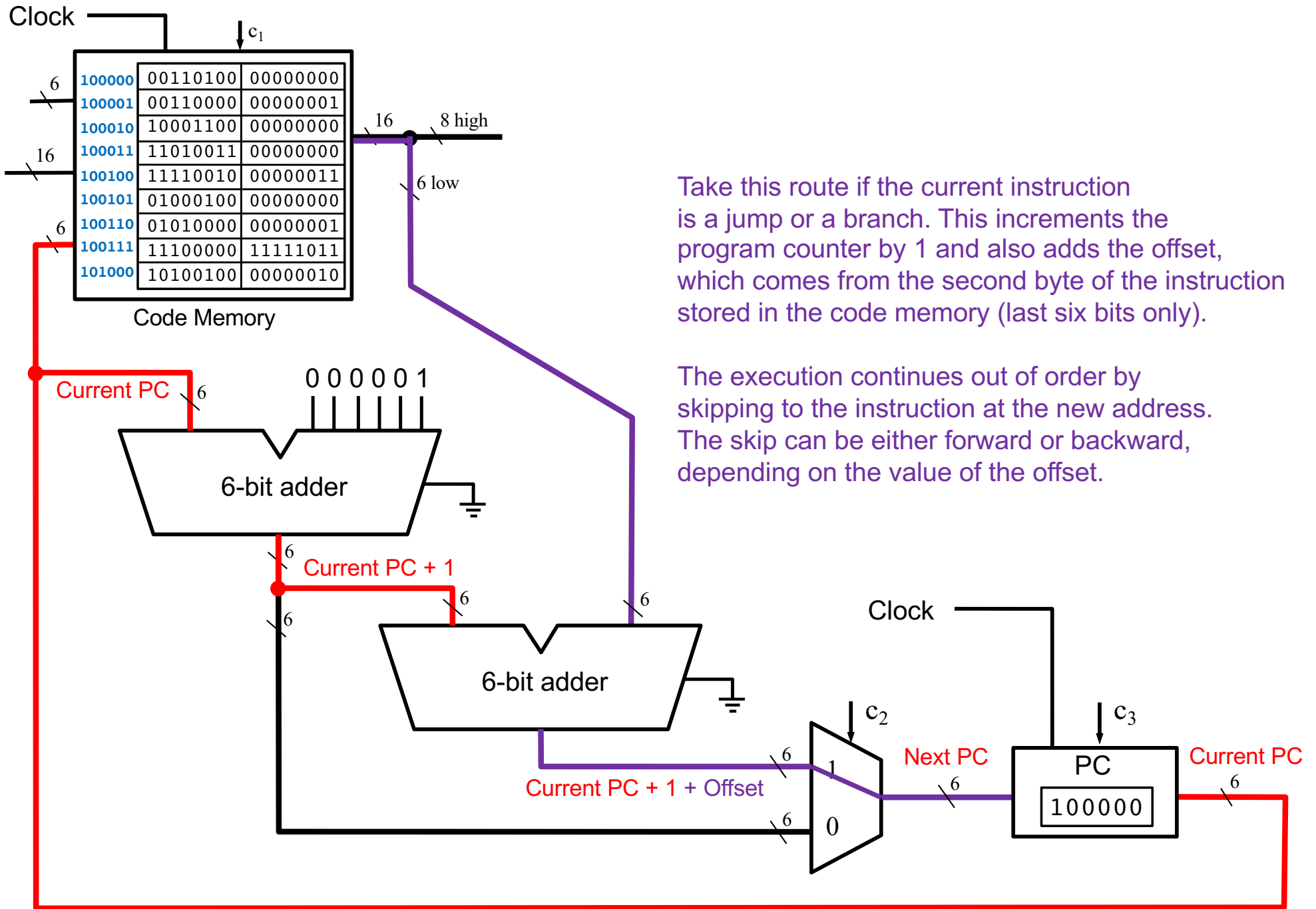
In other words, when the PC reaches address 11111, advancing it by 1 sets it to address 00000.







Take this route if the current instruction is NOT a jump or a branch. This increments the program counter by 1 and the execution continues sequentially to the next instruction.



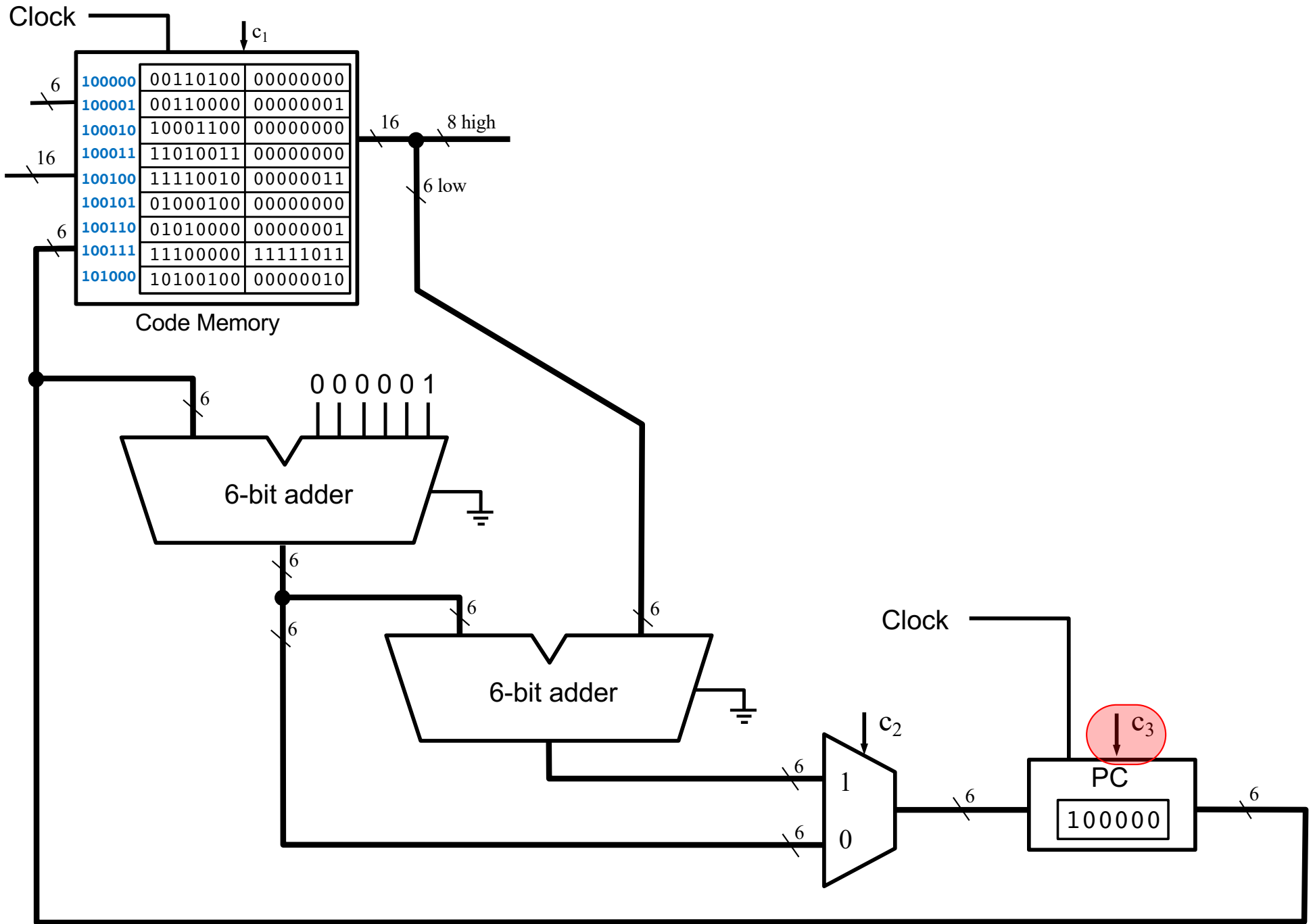
Take this route if the current instruction is a jump or a branch. This increments the program counter by 1 and also adds the offset, which comes from the second byte of the instruction stored in the code memory (last six bits only).

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.

Possible Offsets with +1 Correction

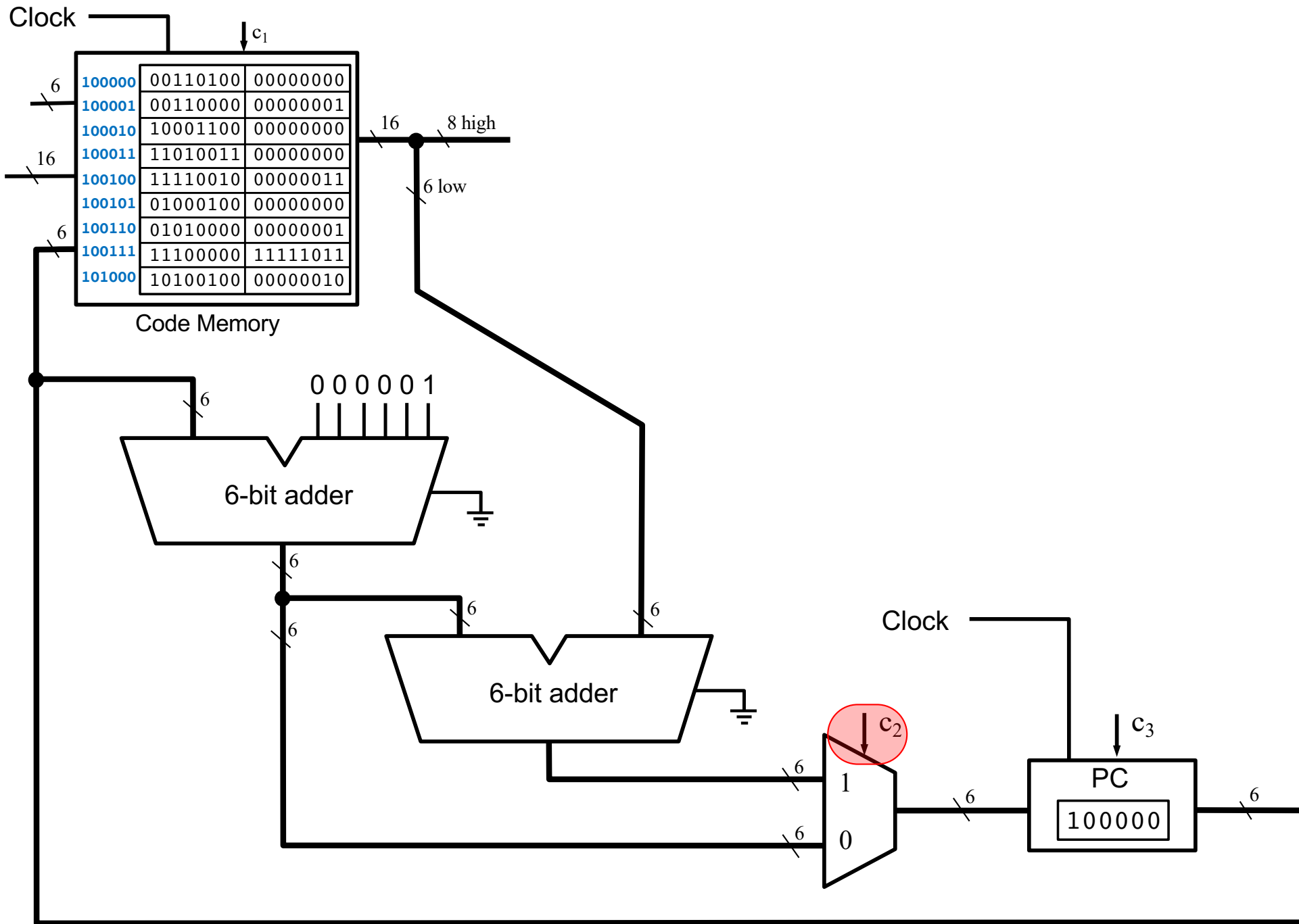
0	11111111
1	00000000
2	00000001
3	00000010
4	00000011
5	00000100
6	00000101
7	00000110
8	00000111
9	00001000
10	00001001
11	00001010
12	00001011
13	00001100
14	00001101
15	00001110
16	00001111
17	00010000
18	00010001
19	00010010
20	00010011
21	00010100
22	00010101
23	00010110
24	00010111
25	00011000
26	00011001
27	00011010
28	00011011
29	00011100
30	00011101
31	00011110
32	00011111

-1	11111110
-2	11111101
-3	11111100
-4	11111011
-5	11111010
-6	11111001
-7	11111000
-8	11110111
-9	11110110
-10	11110101
-11	11110100
-12	11110011
-13	11110010
-14	11110001
-15	11110000
-16	11101111
-17	11101110
-18	11101101
-19	11101100
-20	11101011
-21	11101010
-22	11101001
-23	11101000
-24	11100111
-25	11100110
-26	11100101
-27	11100100
-28	11100011
-29	11100010
-30	11100001
-31	11100000
-32	N/A



	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
	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															

c₃ is always 1, because this is a single-cycle processor, i.e., it executes one instruction per clock cycle.



	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
	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 ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇	C ₈	C ₉	C ₁₀	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈
	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															

C₂ 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)

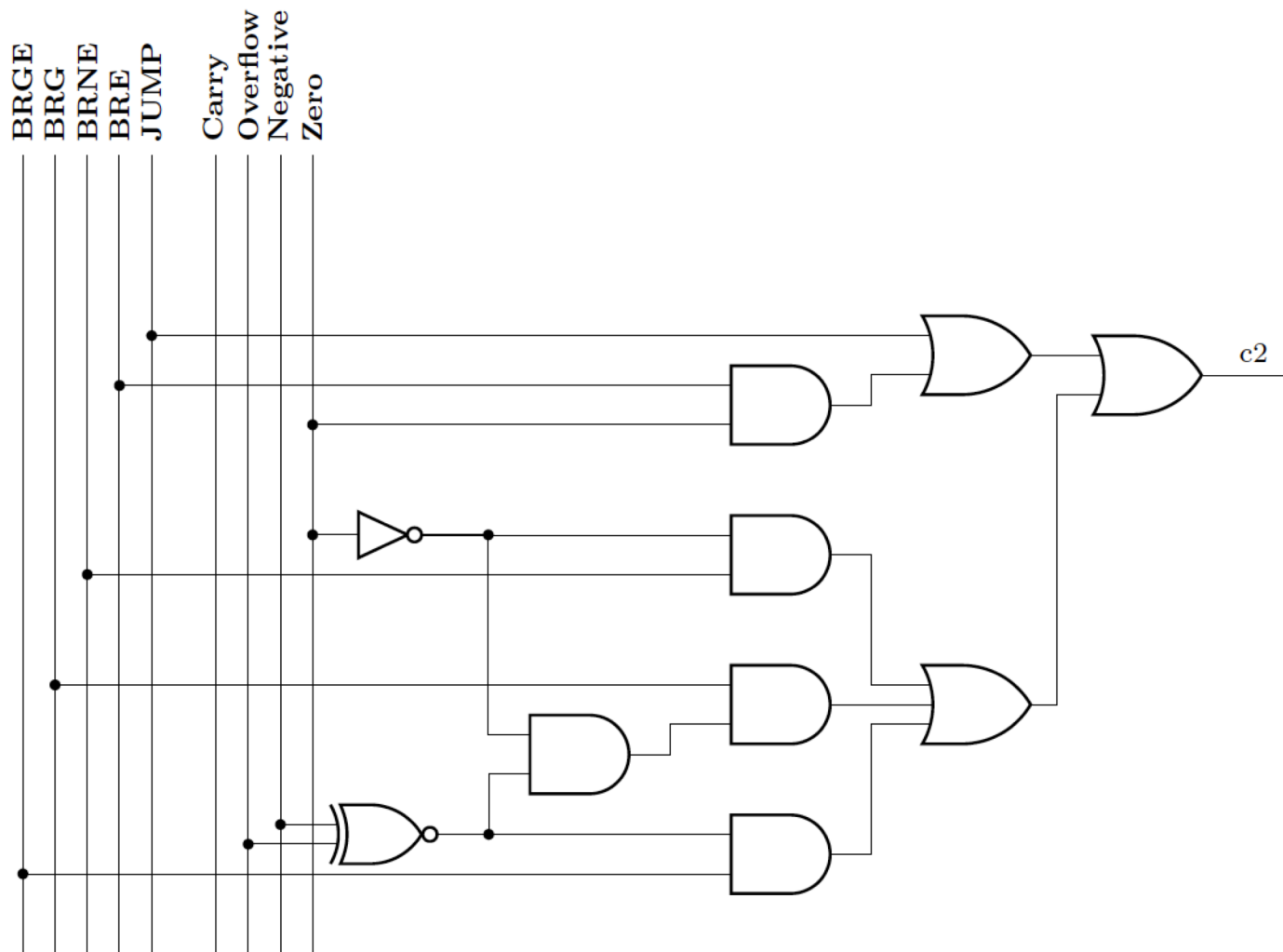
Comparison of Signed Numbers

Comparison of Signed Numbers

- **Equal** $ZF = 1$
- **Not equal** $ZF = 0$
- **Greater** $ZF = 0$ and $NF = OF$
- **Greater or Equal** $NF = OF$
- **Less** $NF \neq OF$
- **Less or Equal** $ZF = 1$ or $NF \neq OF$

Comparison of Signed Numbers

- Equal ZF
- Not equal \overline{ZF}
- Greater $\overline{ZF} \cdot XNOR(NF, OF)$
- Greater or Equal $XNOR(NF, OF)$
- Less $XOR(NF, OF)$
- Less or Equal $ZF + XOR(NF, OF)$

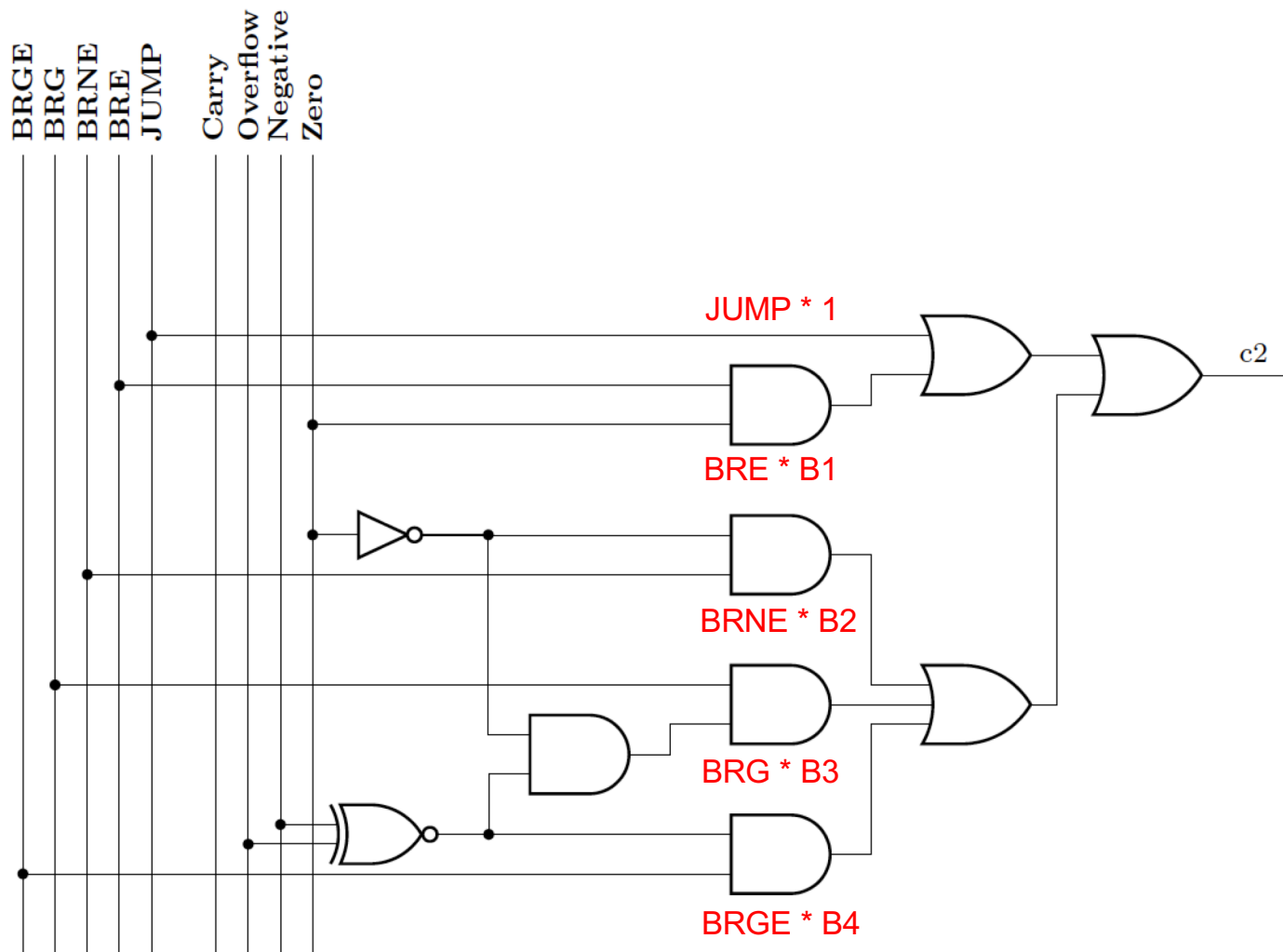


JUMP	1	1																		
BRE/BRZ	B1	1																		
BRNE/BRNZ	B2	1																		
BRG	B3	1																		
BRGE	B4	1																		

C₂ 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																		

C₂ is the OR of these five times the OPCODE

B1= ZF
 B2= $\sim ZF$
 B3= AND($\sim ZF$, XNOR(NF, OF))
 B4= XNOR(NF, OF)

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

Some Interesting Dualities

Equal = Not Equal

Greater = Less or Equal

Less = Greater or Equal

Comparison of Unsigned Numbers (not supported by this CPU)

Comparison of **Unsigned** Numbers

- Equal
- Not equal
- Greater
- Greater or equal
- Less
- Less or Equal

Comparison of **Unsigned** Numbers

- Equal
- Not equal
- Greater / Above
- Greater or Equal / Above or Equal
- Less / Below
- Less or Equal / Below or Equal

Comparison of **Unsigned** Numbers

- **Equal** $ZF = 1$
- **Not equal** $ZF = 0$
- **Greater** $ZF = 0$ and $CF = 1$
- **Greater or Equal** $CF = 1$
- **Less** $CF = 0$
- **Less or Equal** $ZF = 1$ or $CF = 0$

Comparison of **Unsigned** Numbers

- Equal ZF
- Not equal \overline{ZF}
- Greater $\overline{ZF} \cdot CF$
- Greater or Equal CF
- Less \overline{CF}
- Less or Equal $ZF + \overline{CF}$

Comparison of **Unsigned** Numbers

- Equal ZF
- Not equal \overline{ZF}
- Above $\overline{ZF} \cdot CF$
- Above or Equal CF
- Below \overline{CF}
- Below or Equal $ZF + \overline{CF}$

**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 ?

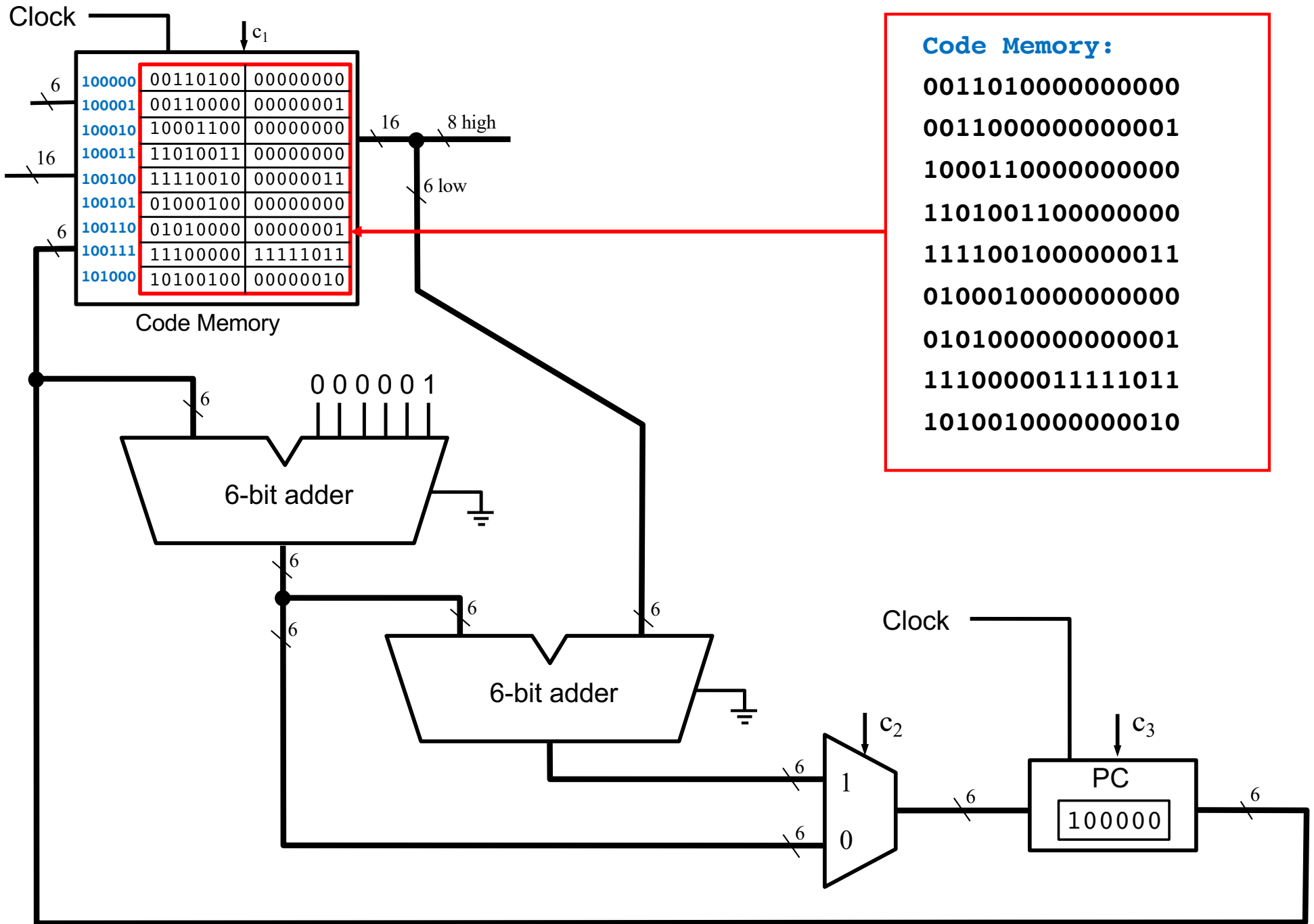
 BRG End ; exit if i>N

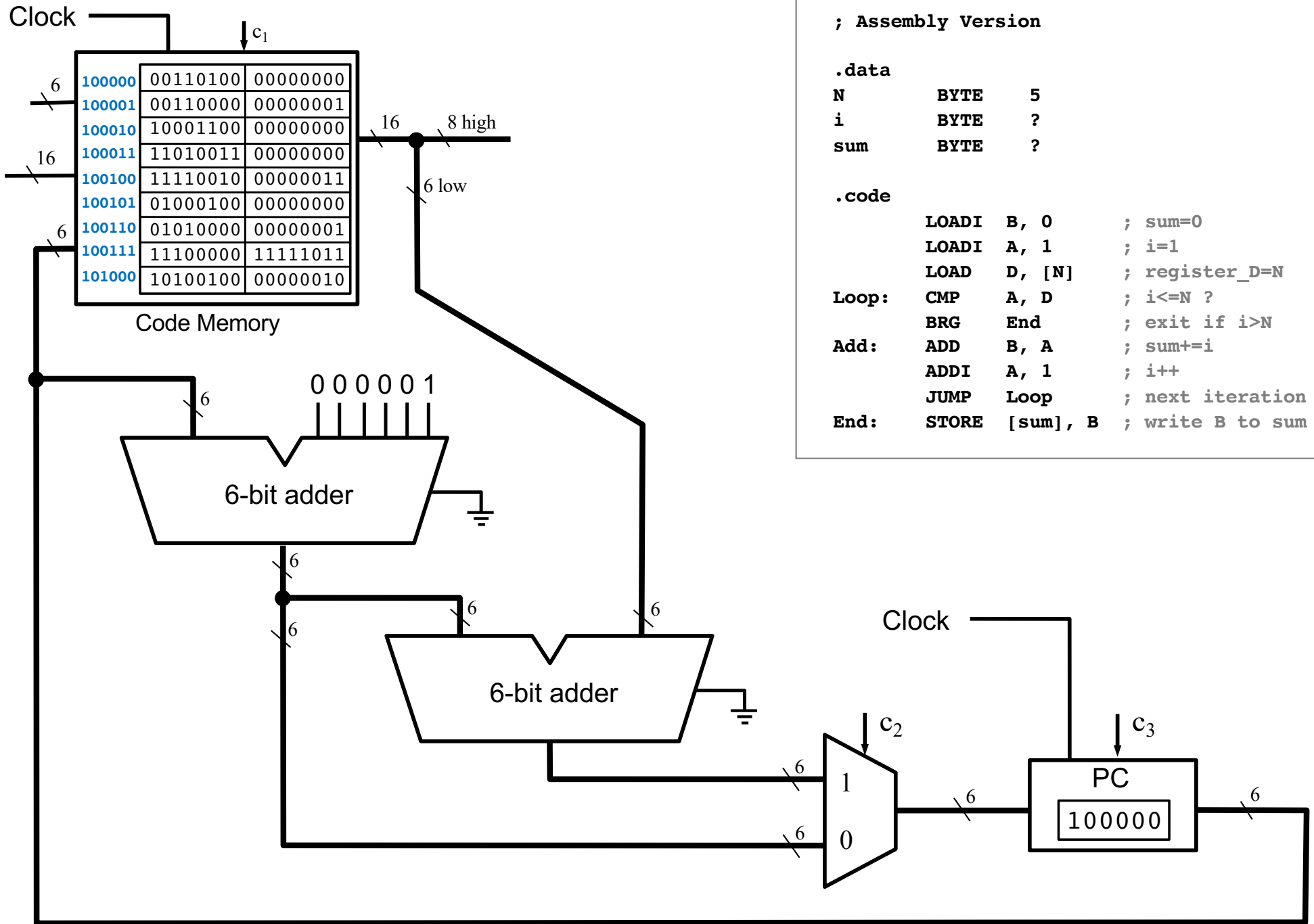
Add: ADD B, A ; sum+=i

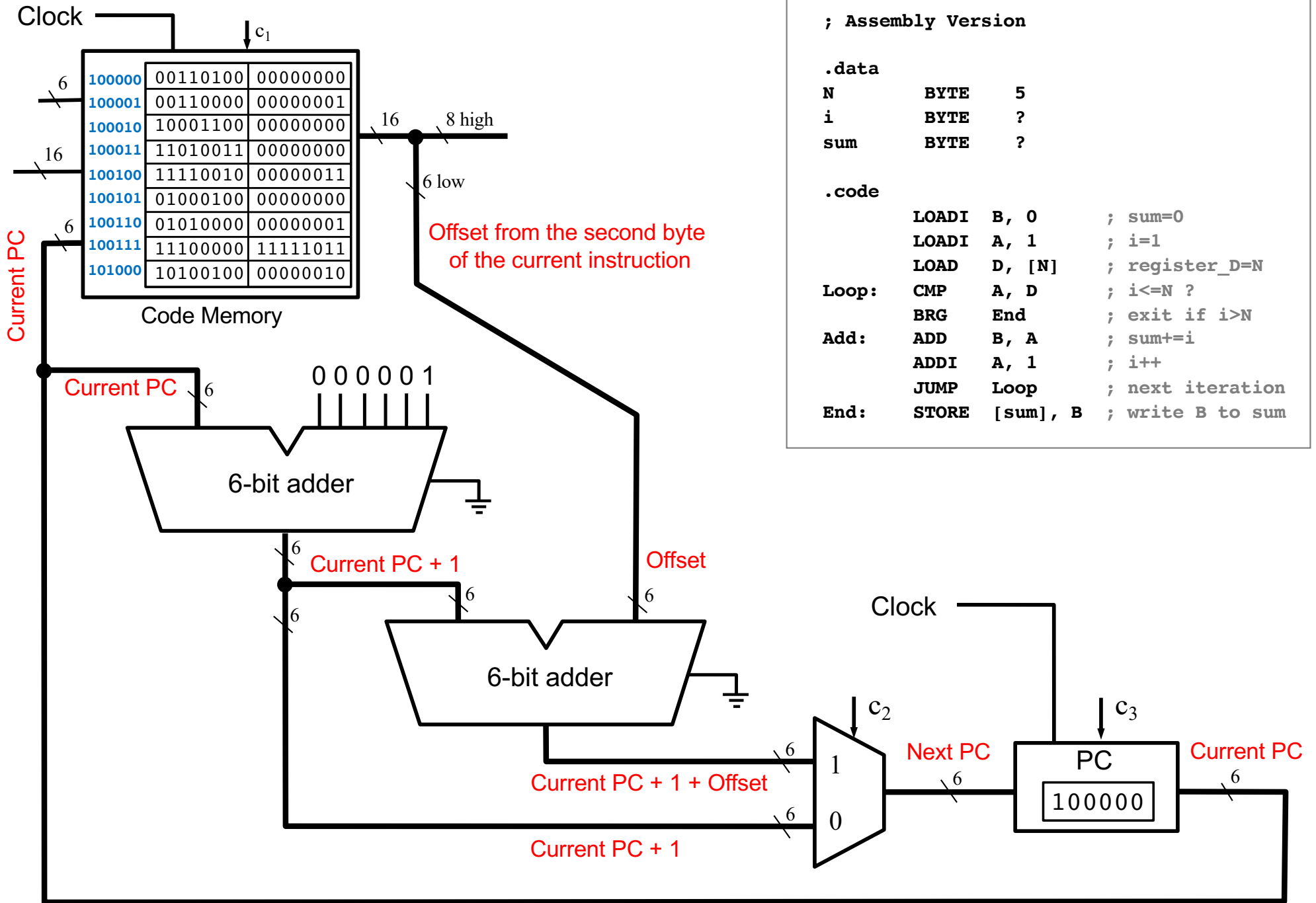
 ADDI A, 1 ; i++

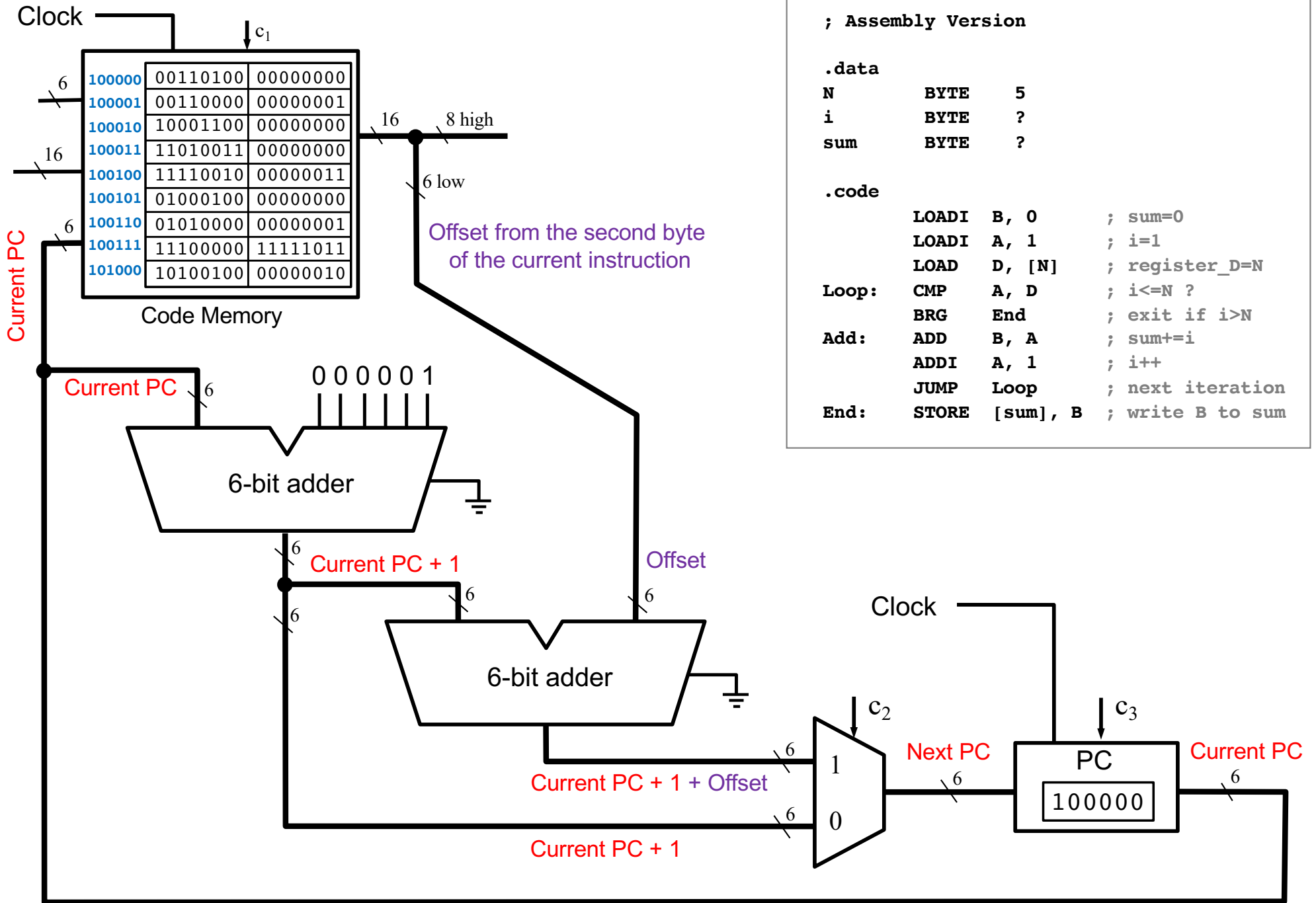
 JUMP Loop ; next iteration

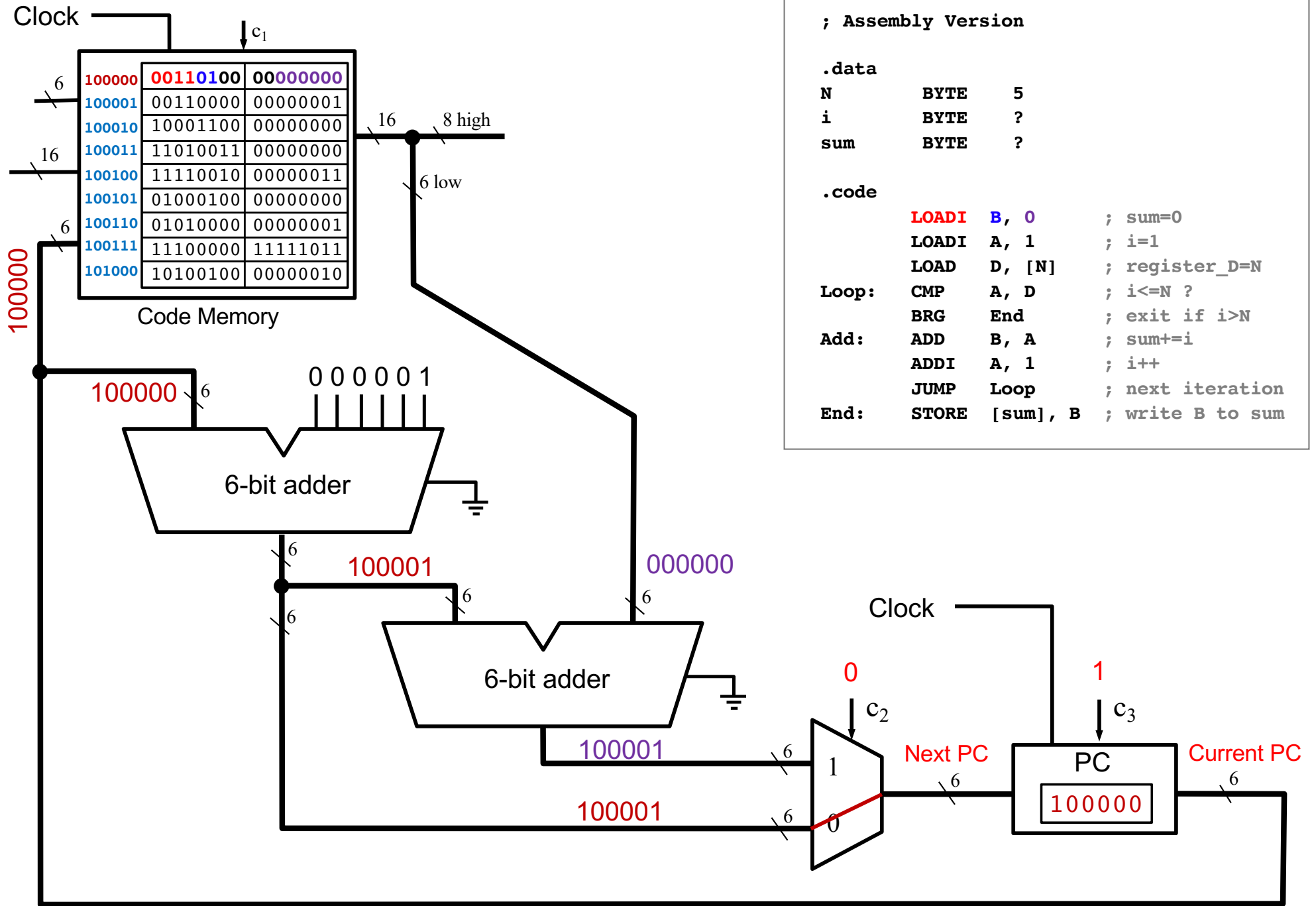
End: STORE [sum], B ; write B to sum

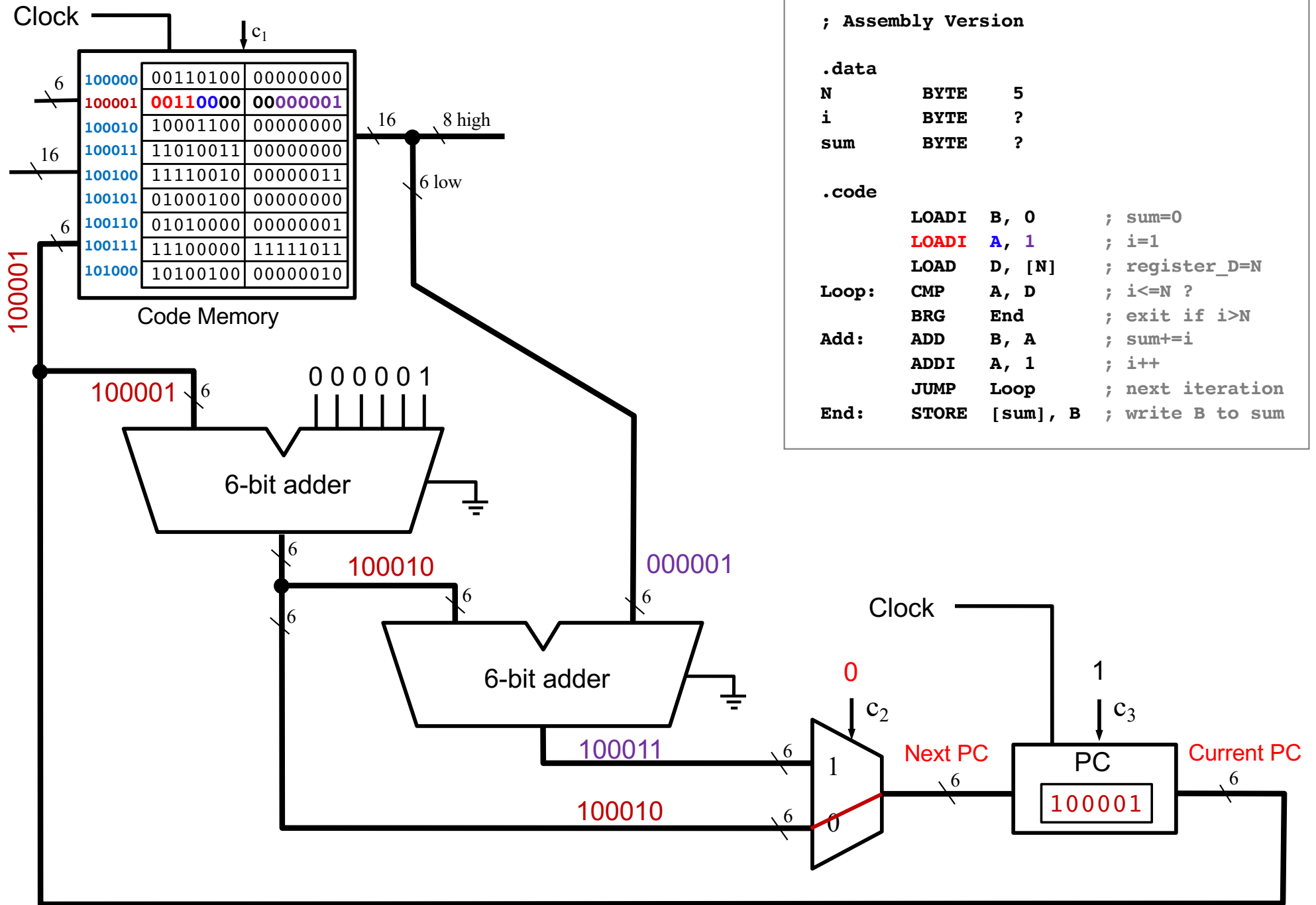


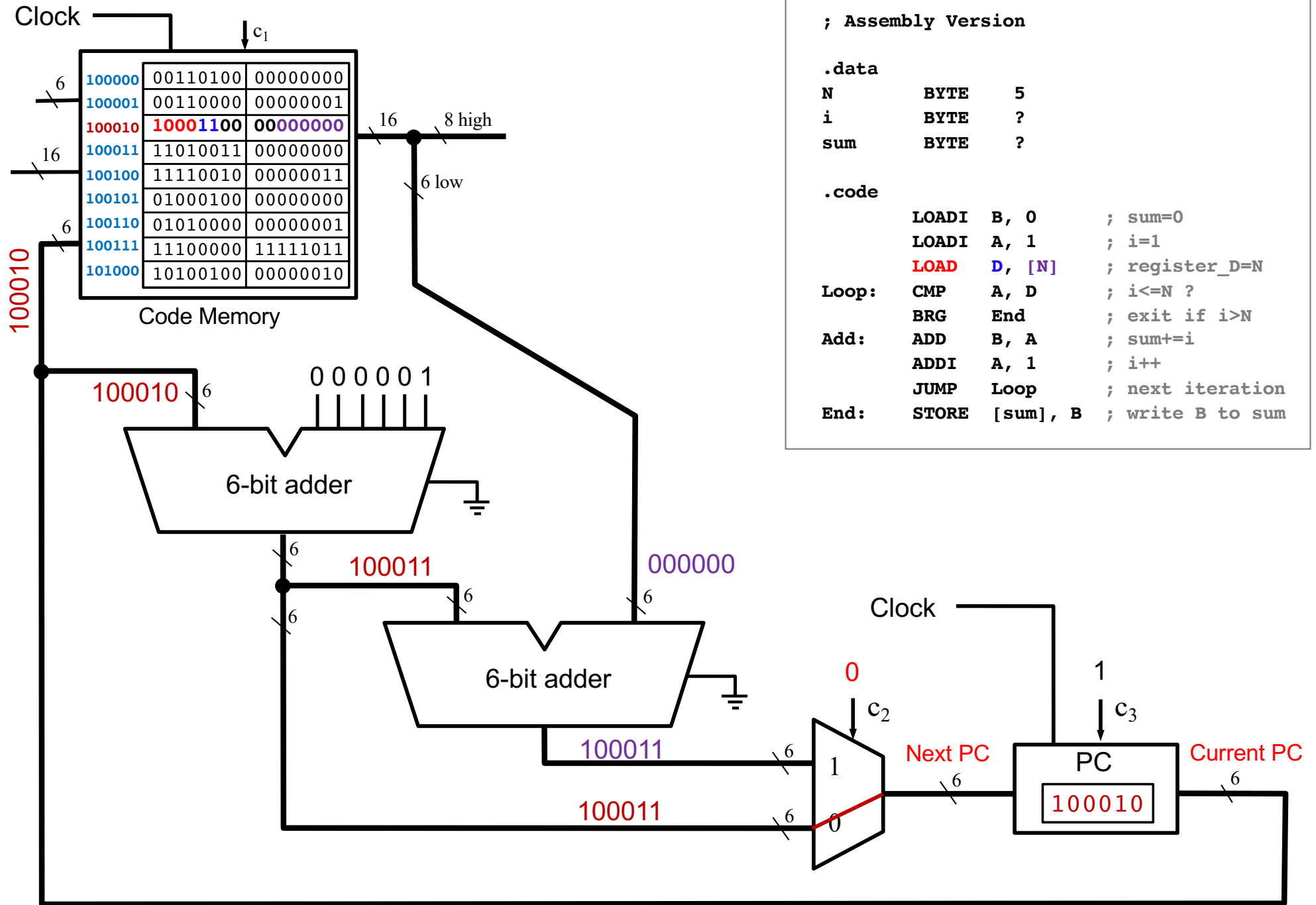


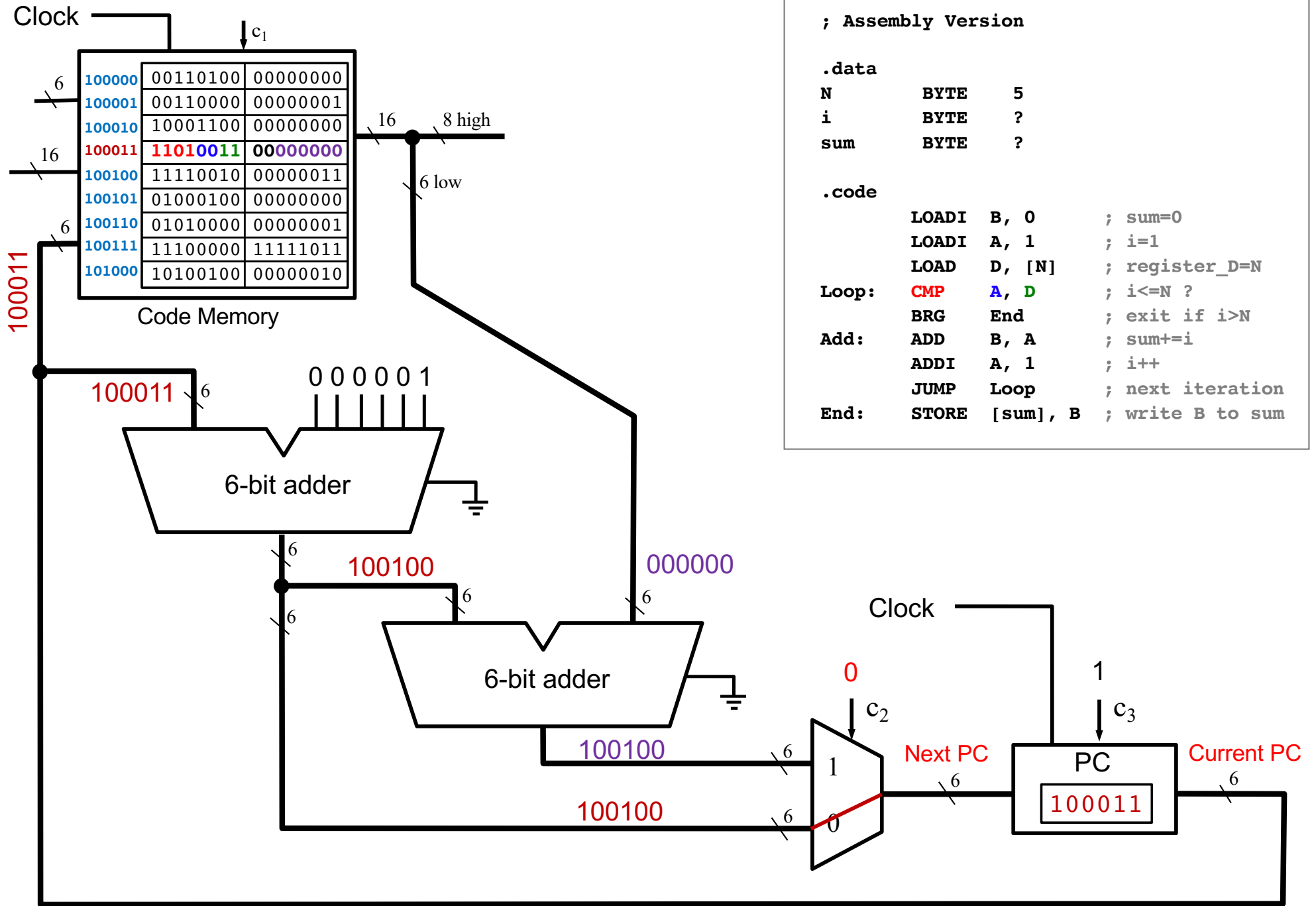


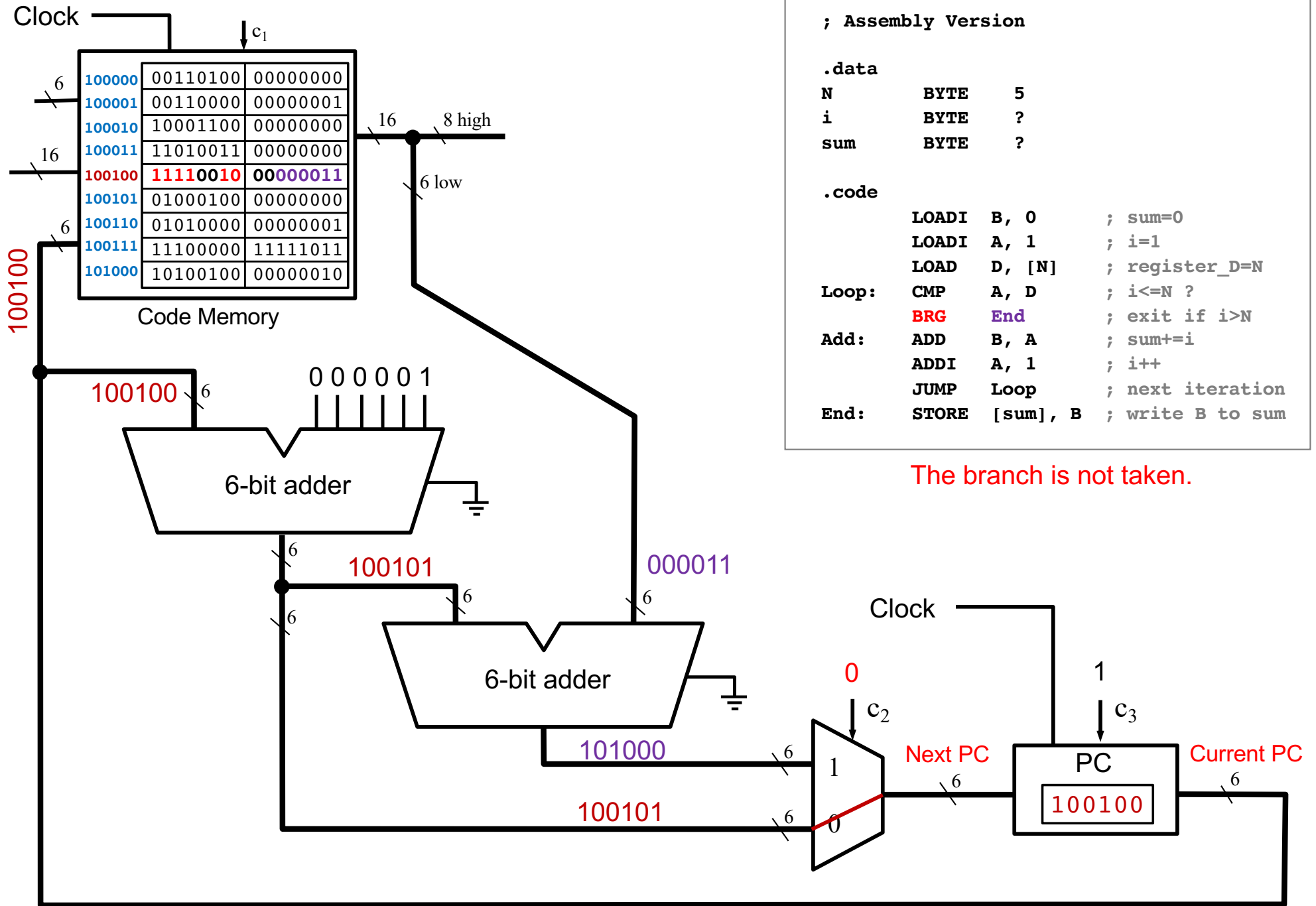


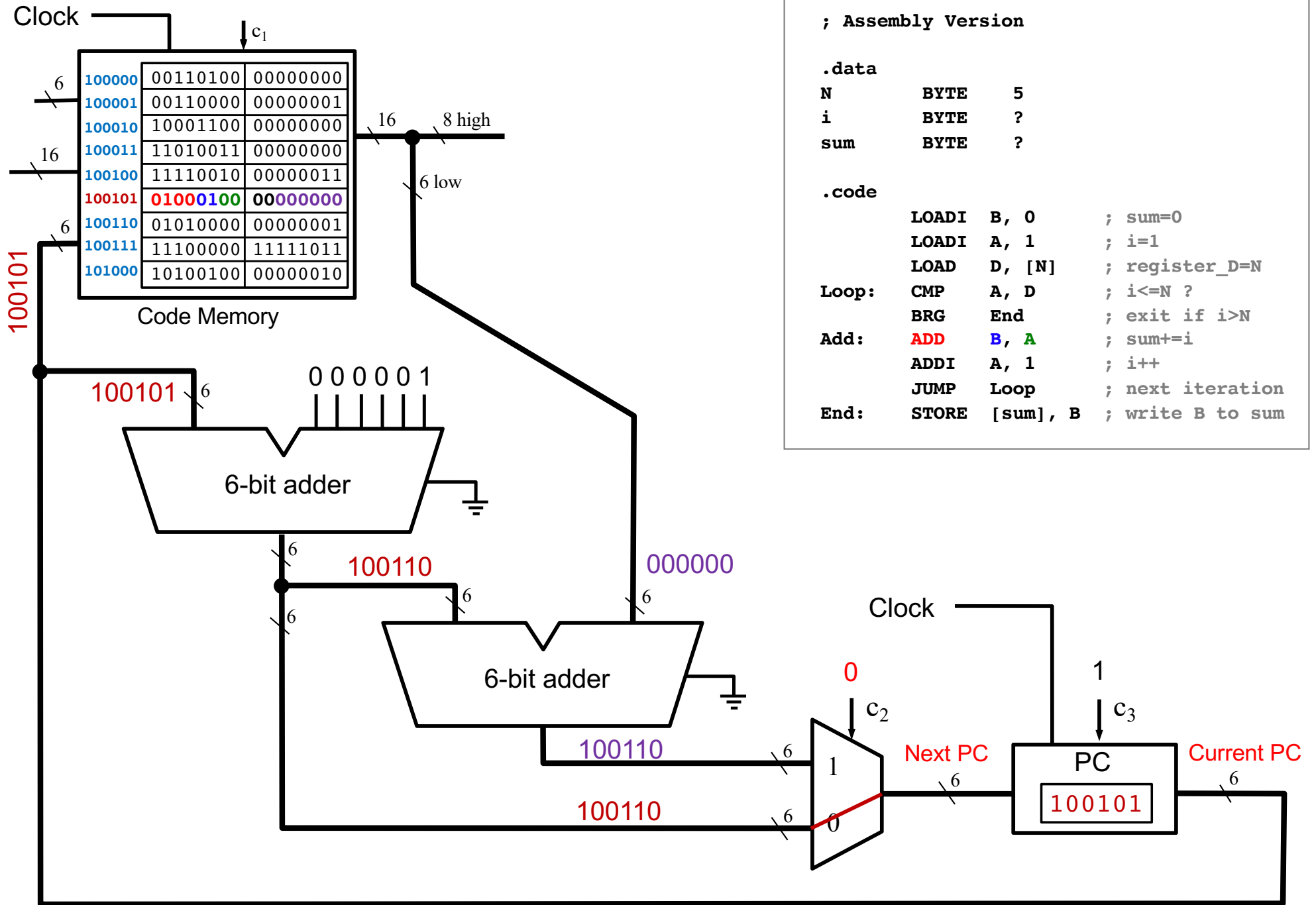


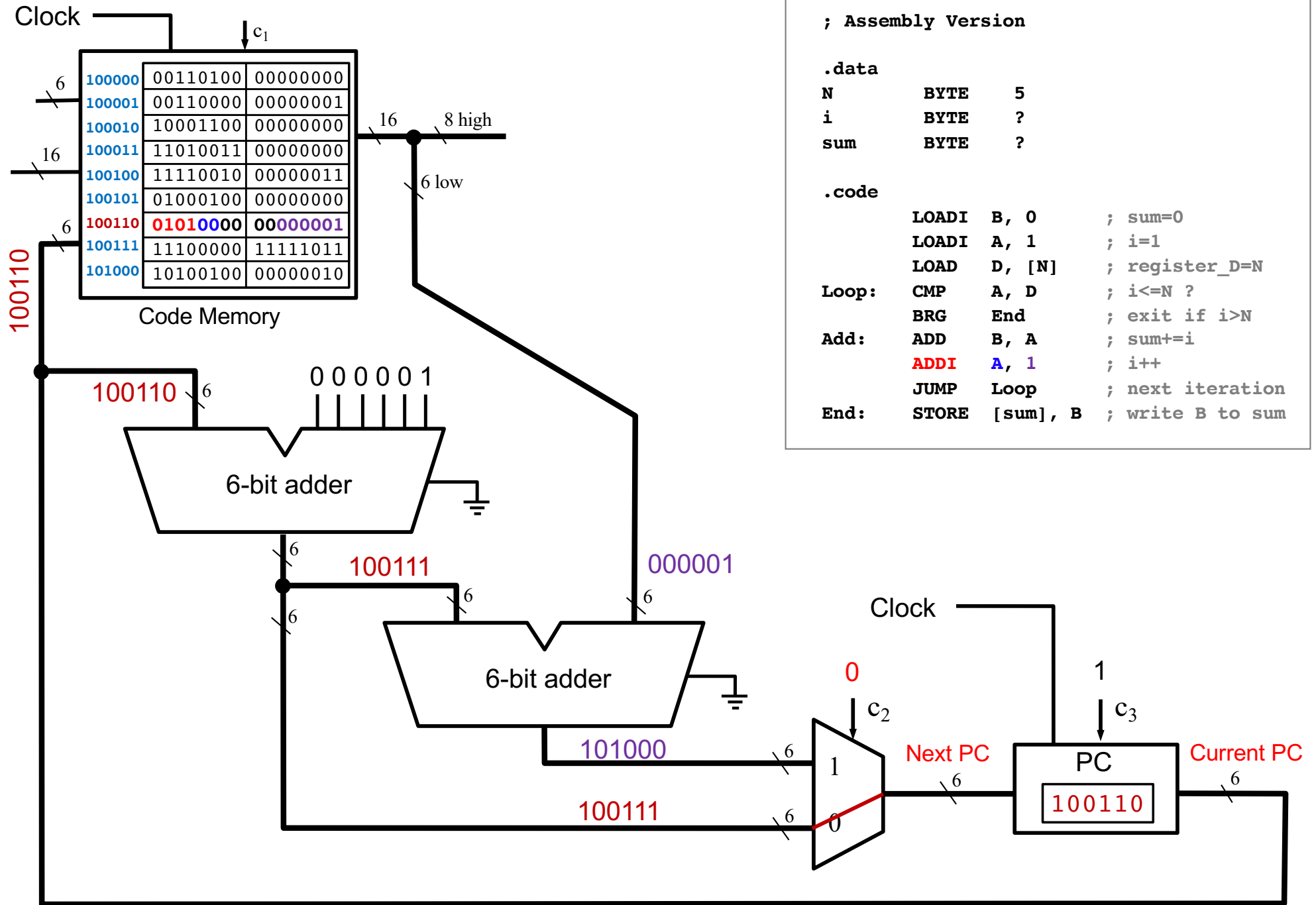


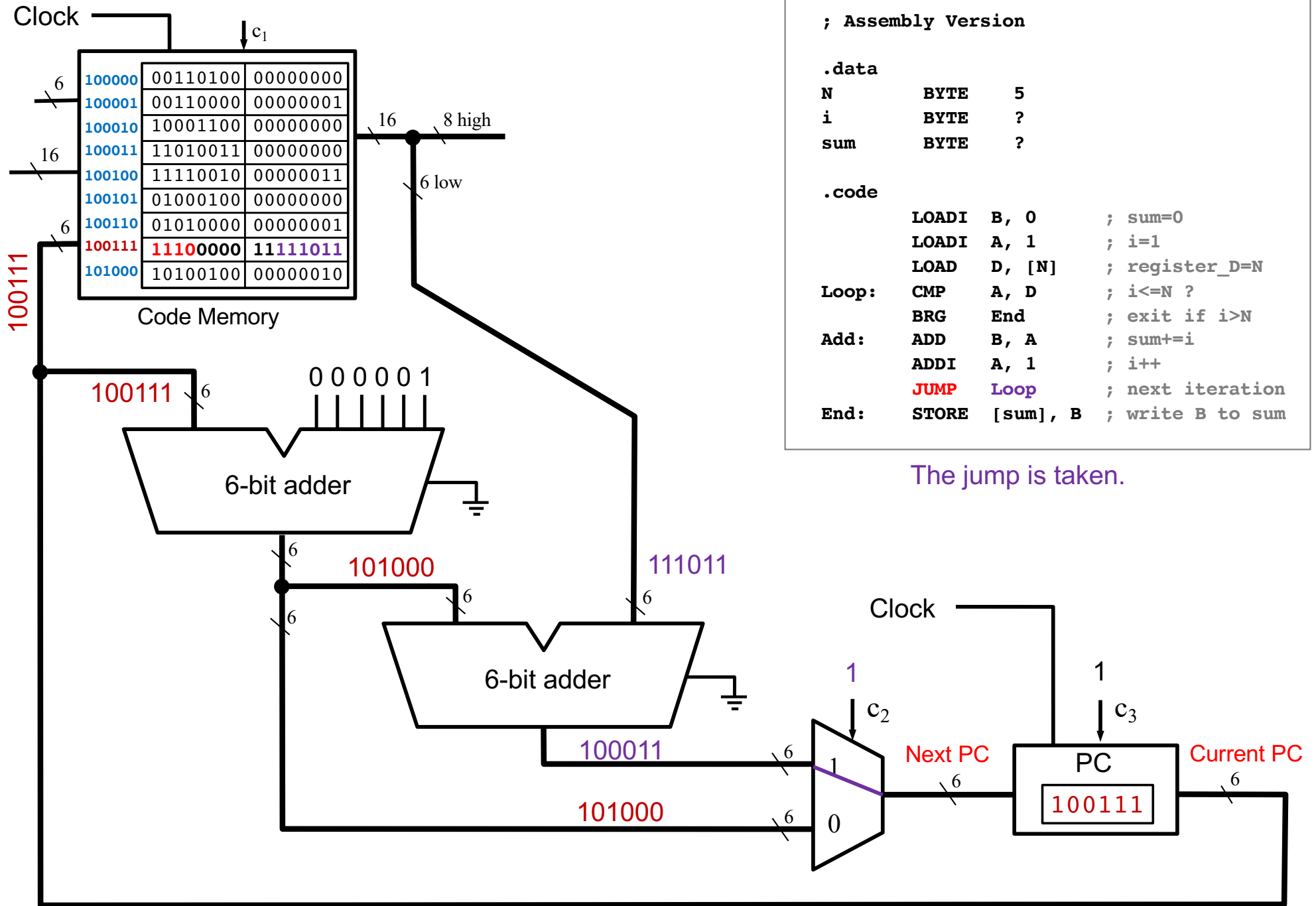


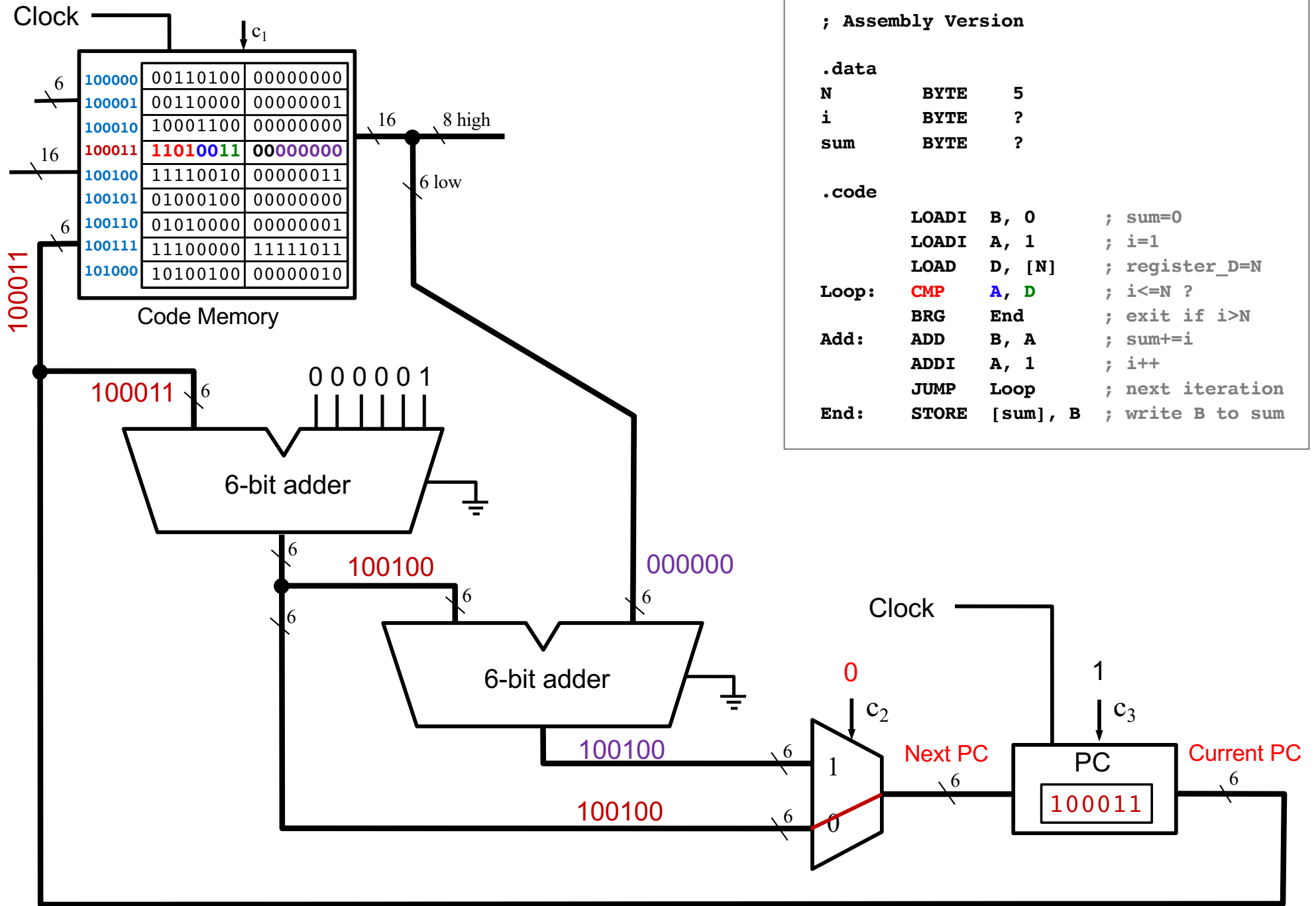


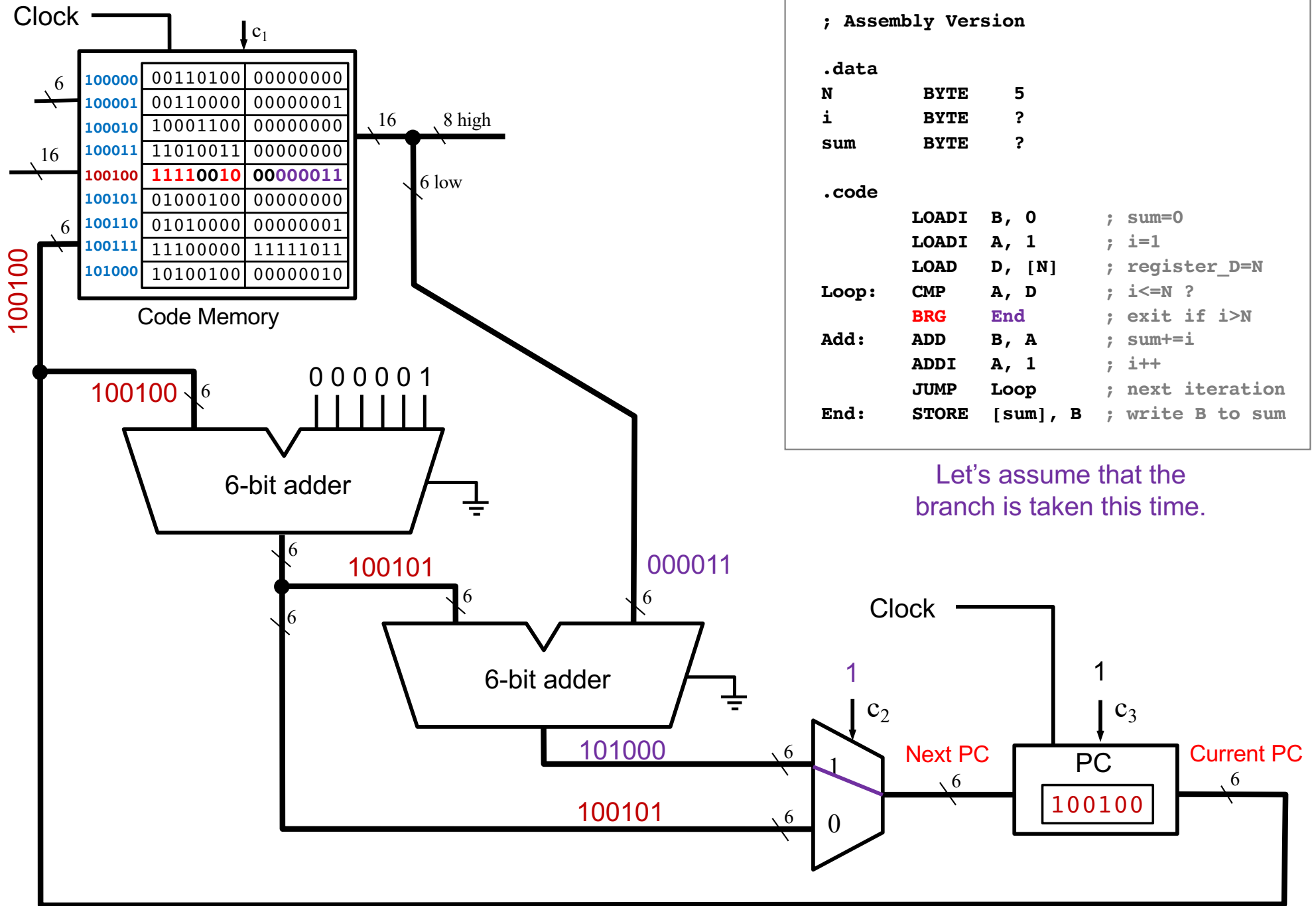


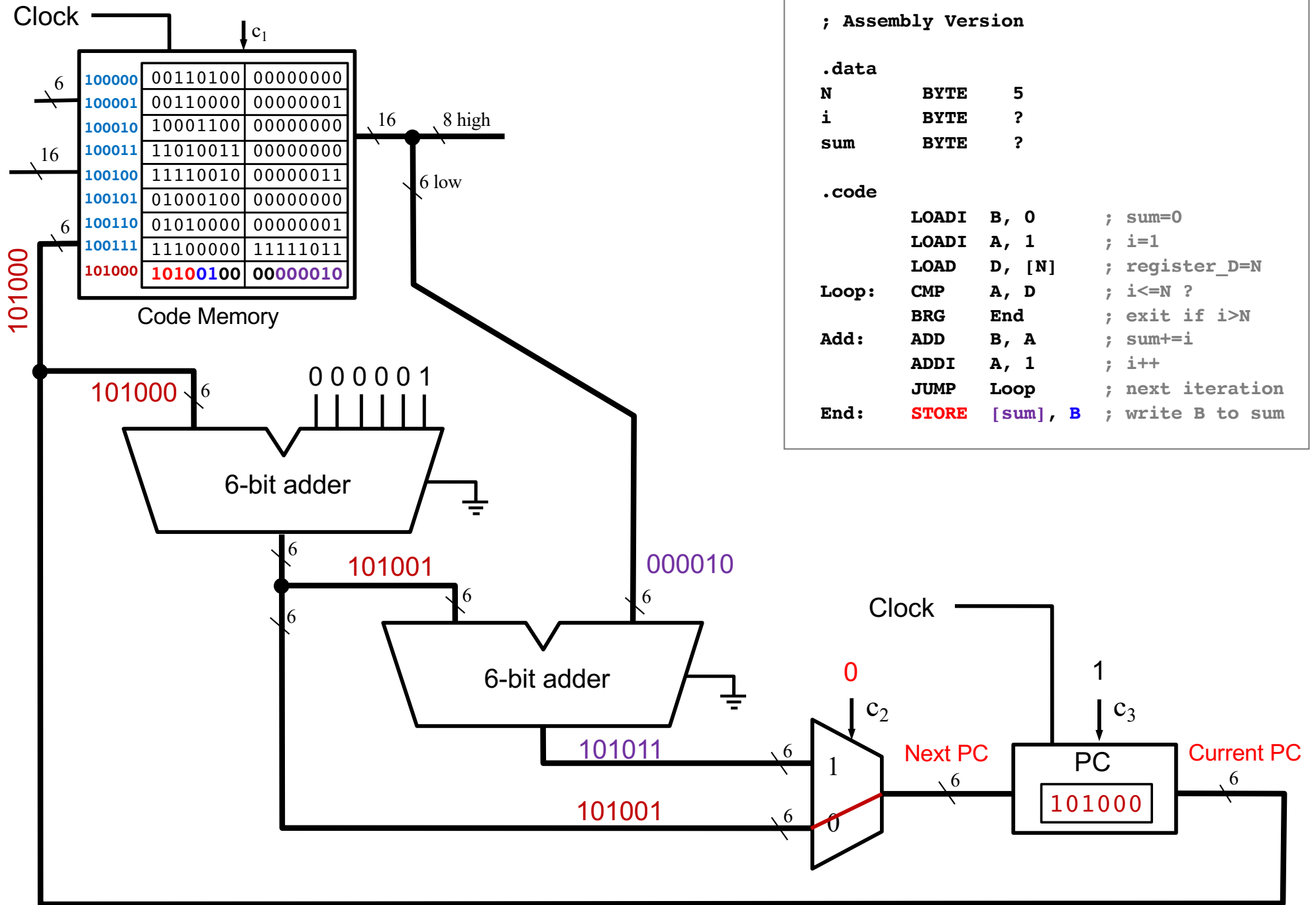


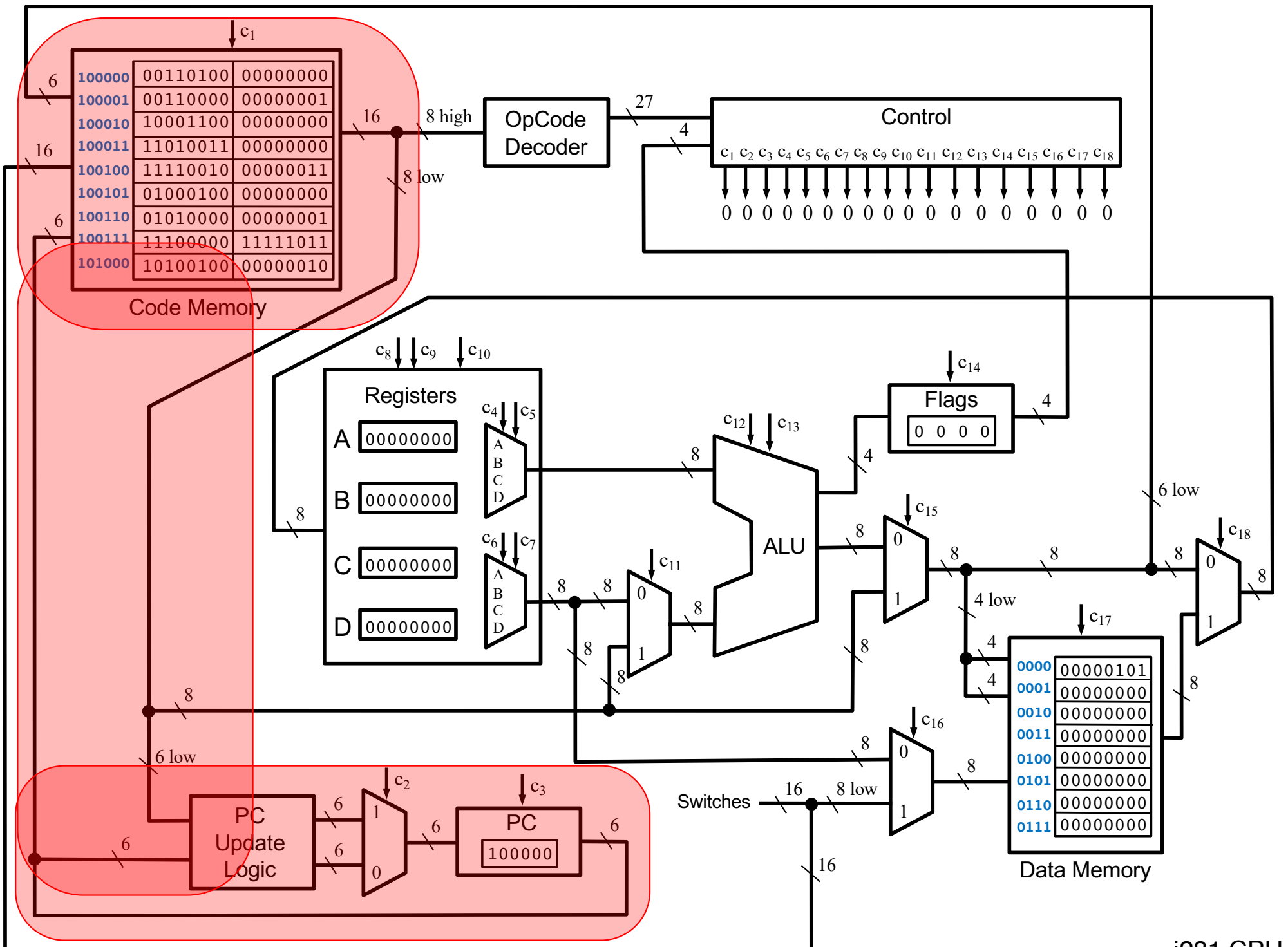


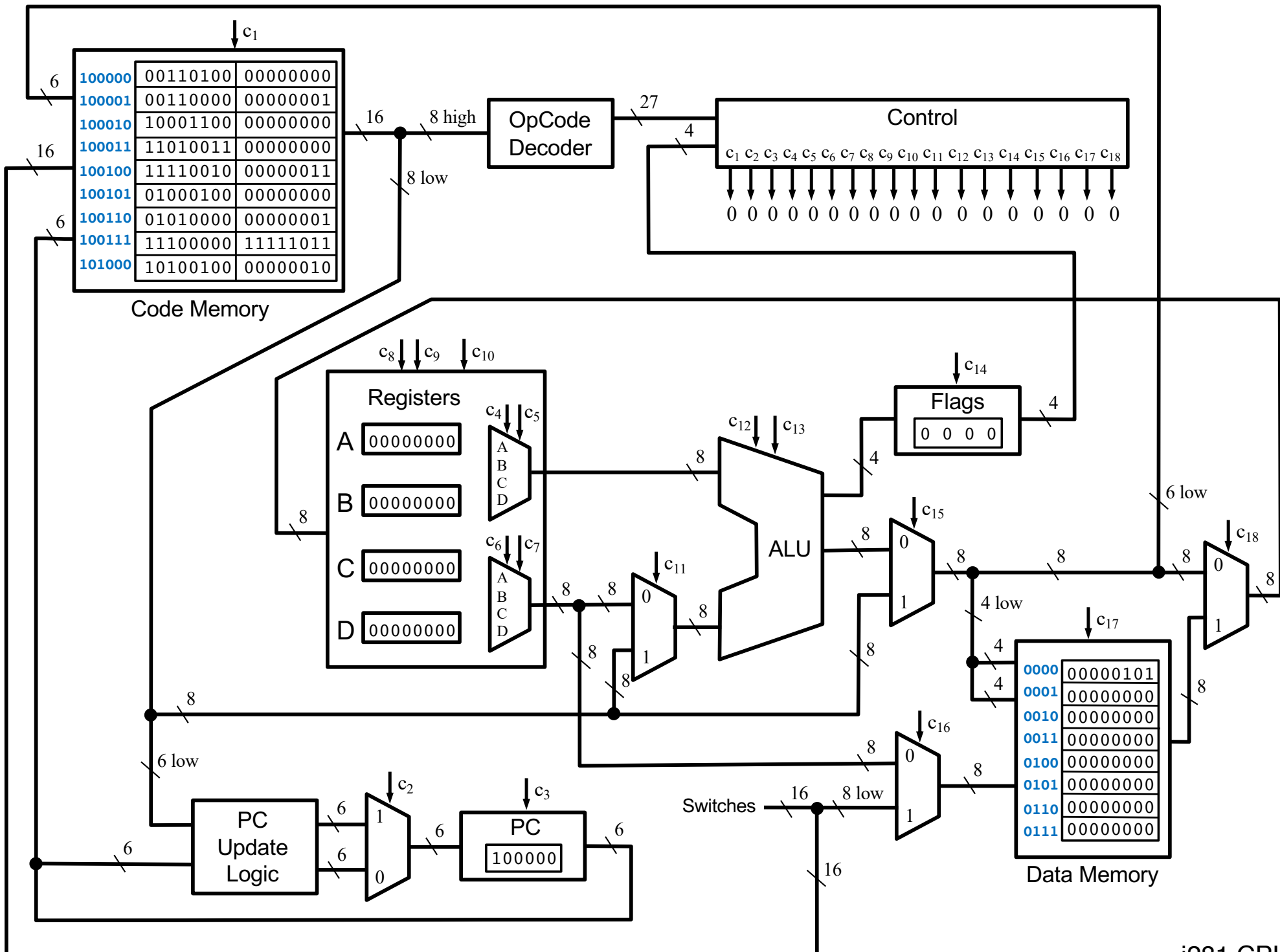


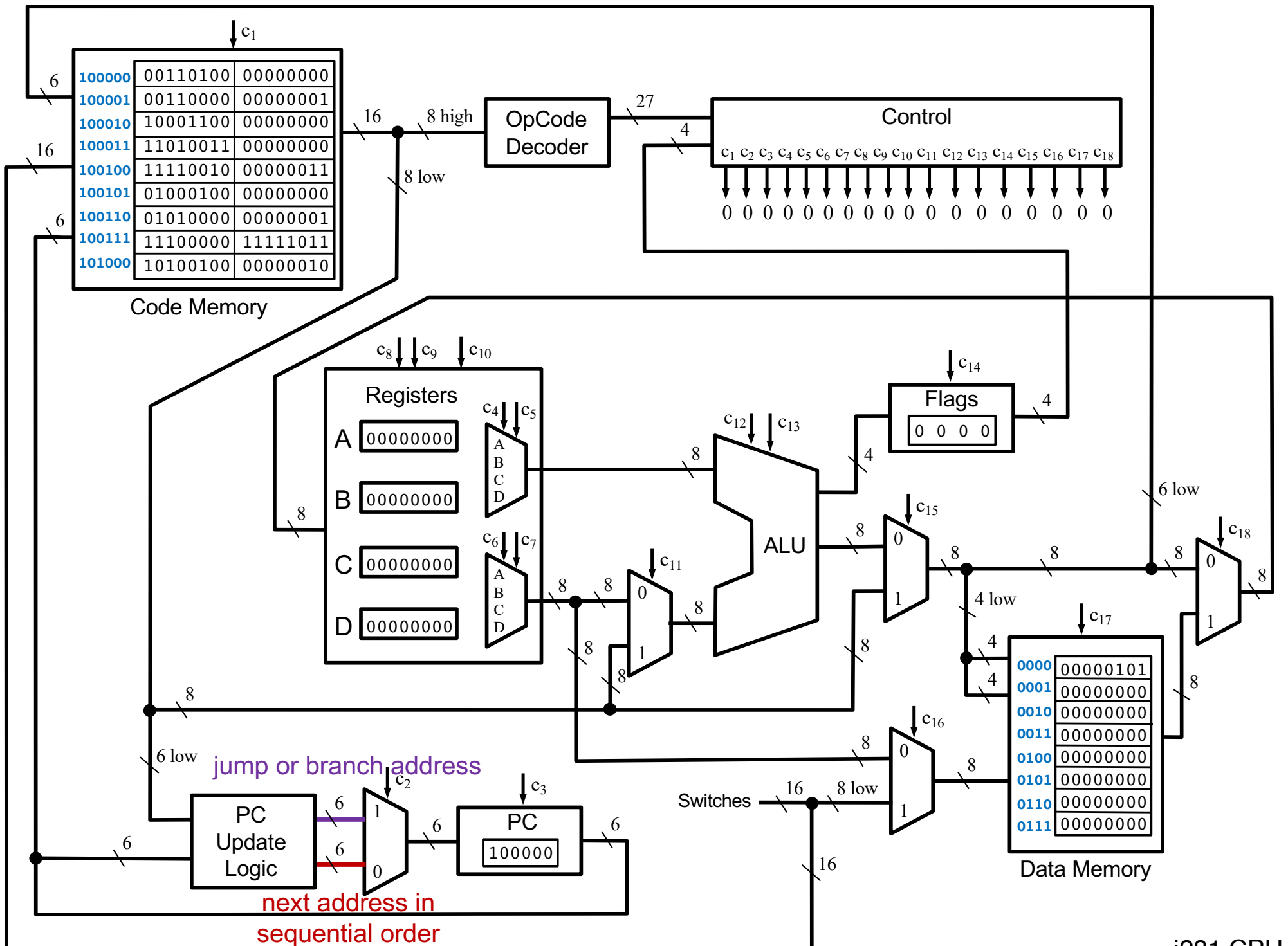












Questions?

THE END