



# **CprE 281: Digital Logic**

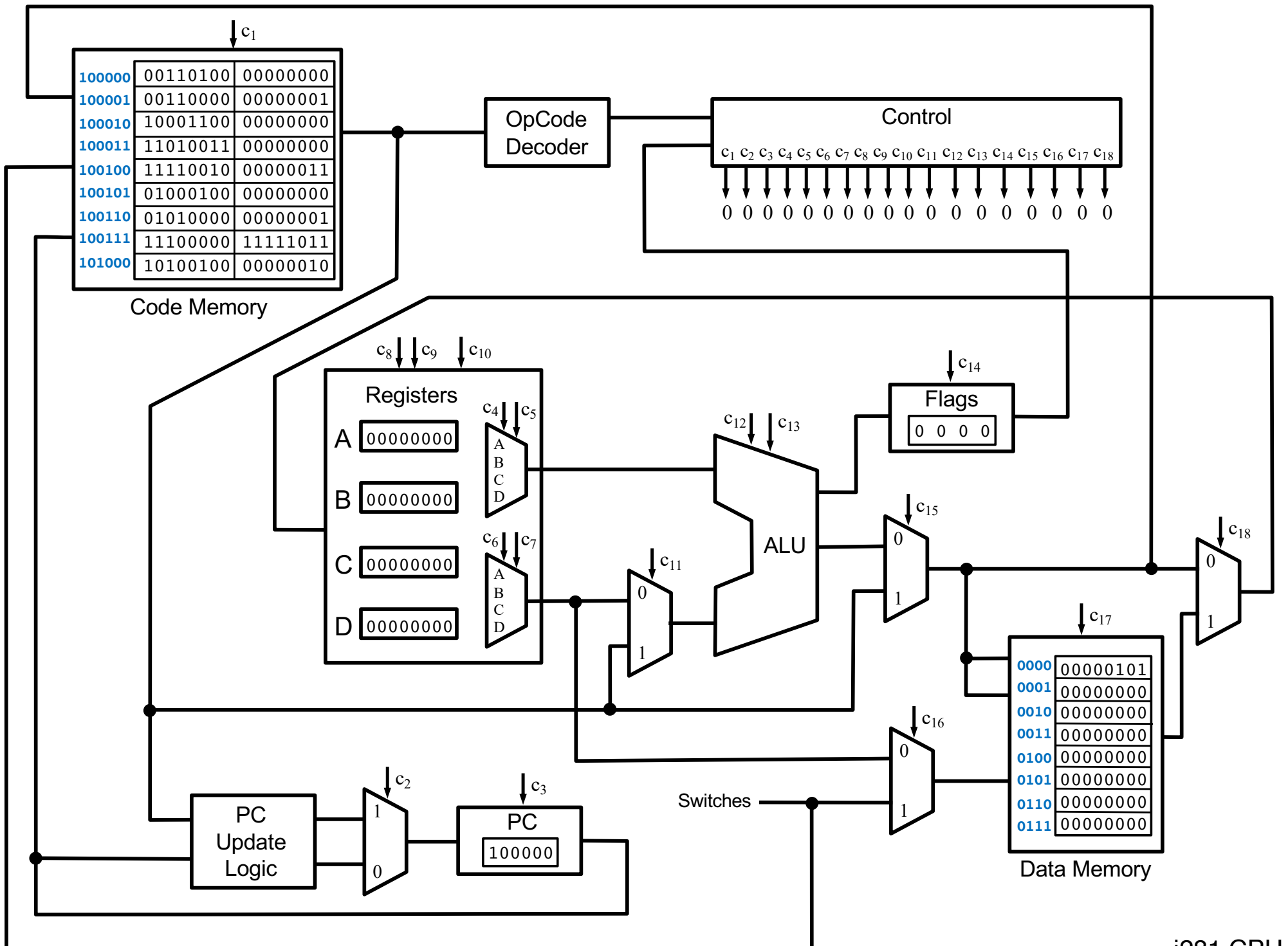
**Instructor: Alexander Stoytchev**

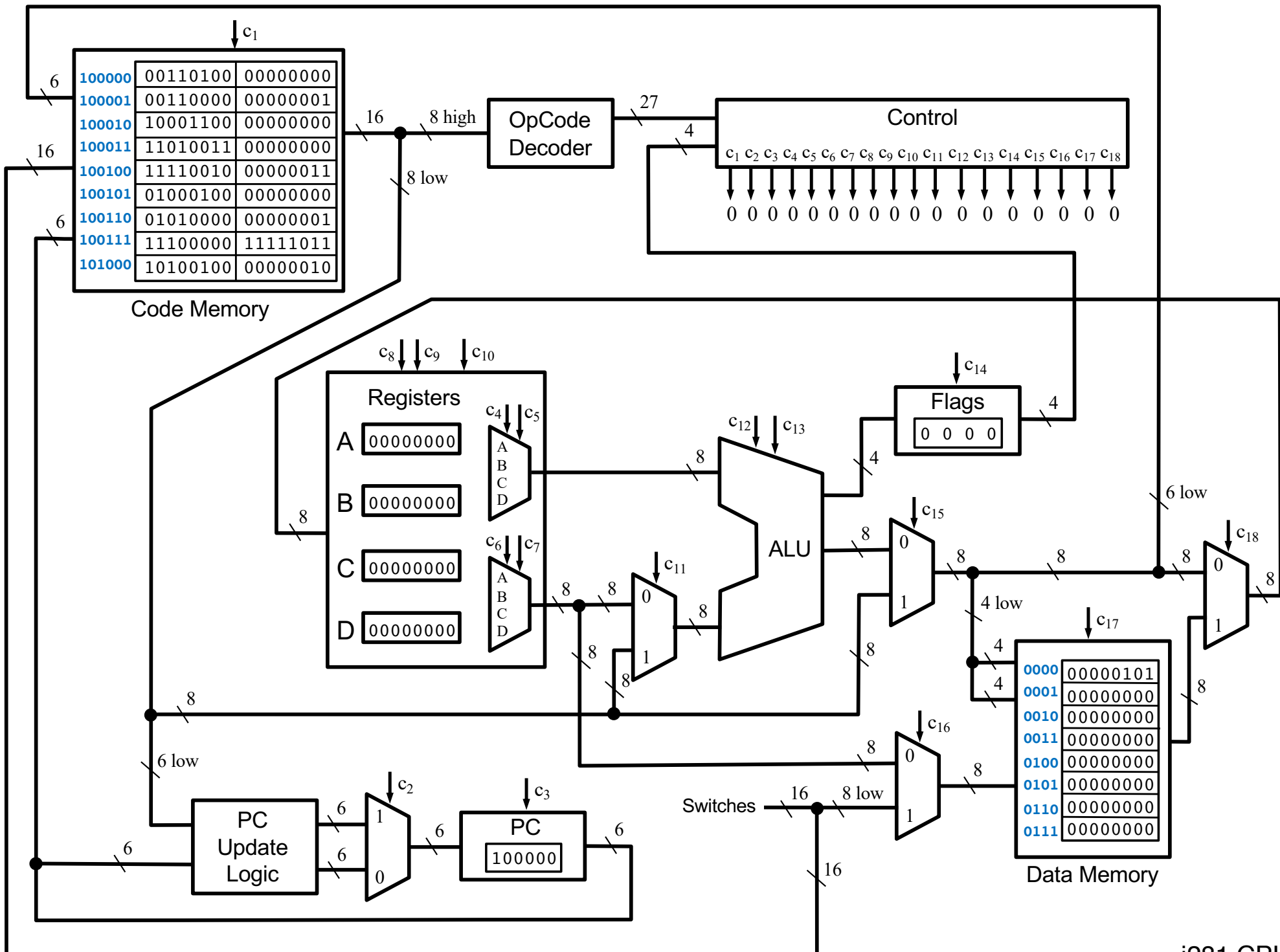
**<http://www.ece.iastate.edu/~alexs/classes/>**

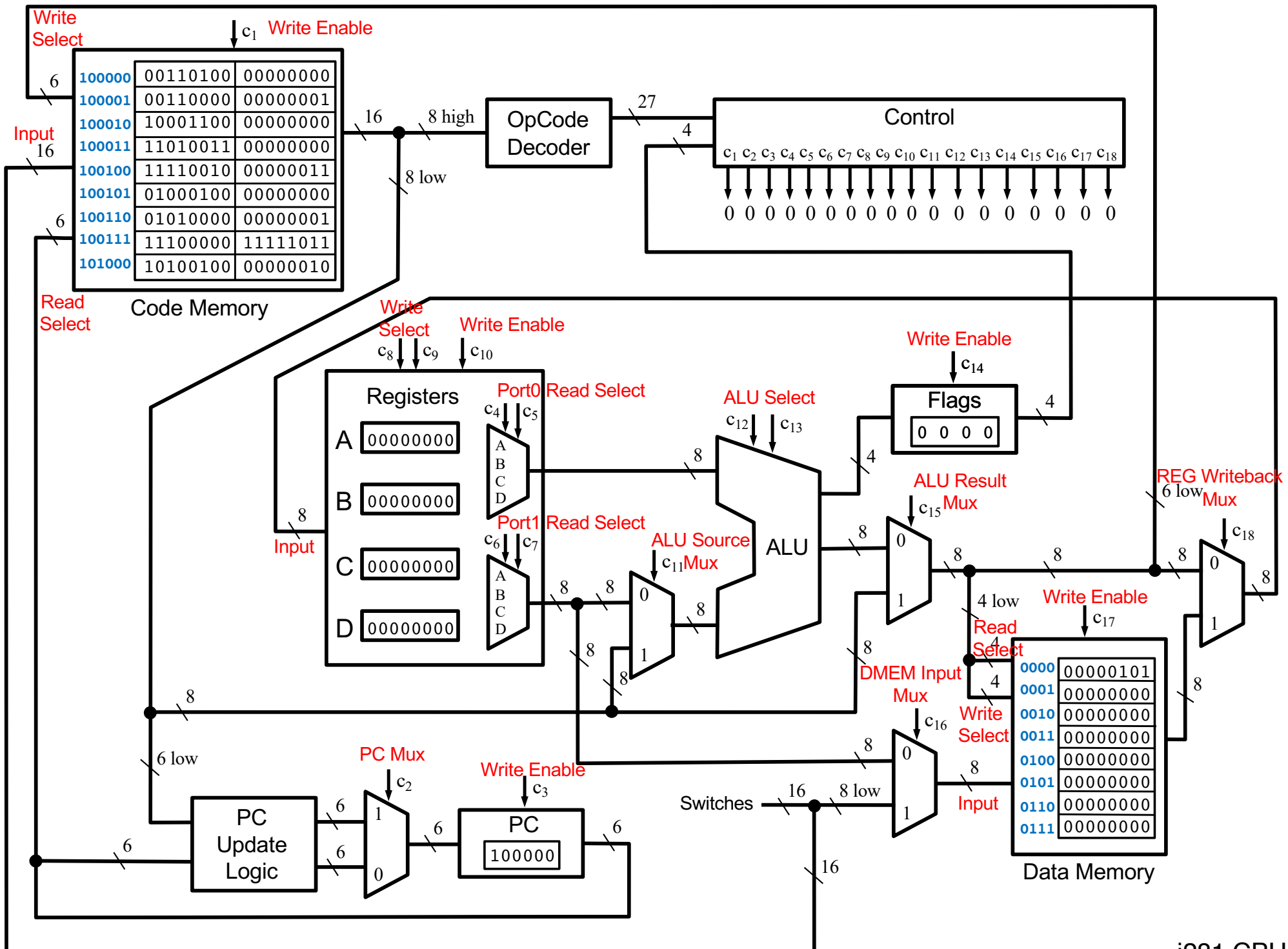
# **i281 CPU Architecture**

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









# Drawing Convention

- **Inputs enter from the left (for each box)**
- **Outputs exit from the right (for each box)**
- **Control lines are vertical arrows (come from the top)**

# i281 CPU

- **The CPU was designed specifically for this class 😊**

# **i281 CPU**

- **The CPU was designed specifically for this class 😊**
- **It was designed by:**  
**Kyung-Tae Kim and Alexander Stoytchev**

# i281 CPU Web Links

**Download link for the software and hardware that comes with Bubble Sort preinstalled:**

[https://www.ece.iastate.edu/~alexs/classes/2022\\_Fall\\_281/i281\\_CPU/](https://www.ece.iastate.edu/~alexs/classes/2022_Fall_281/i281_CPU/)

**Download link for the hardware version that comes with PONG preinstalled:**

[https://www.ece.iastate.edu/~alexs/classes/2022\\_Fall\\_281/labs/EC\\_lab/](https://www.ece.iastate.edu/~alexs/classes/2022_Fall_281/labs/EC_lab/)

**Web link for the i281 Simulator:**

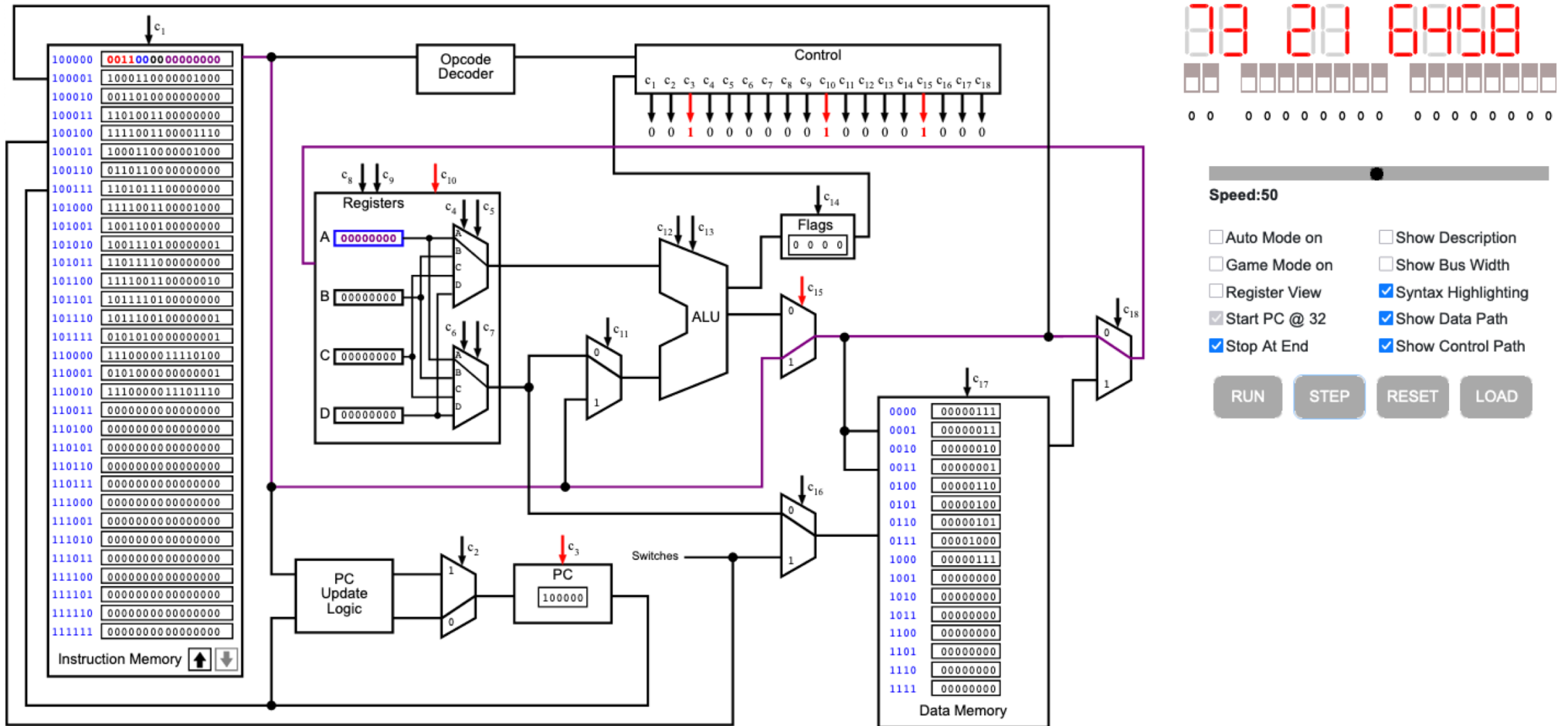
<https://www.ece.iastate.edu/~alexs/classes/i281/index.html>

# i281 Simulator

Current Instruction: **LOADI A, 0**

i281 CPU Running: **BubbleSort**

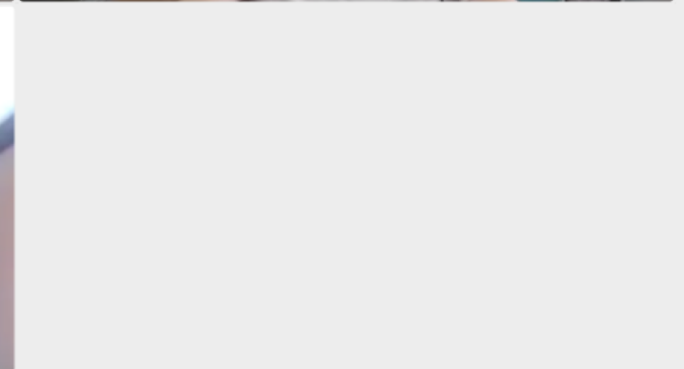
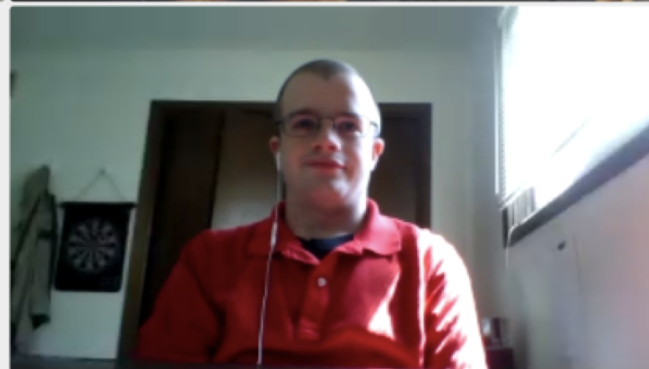
About



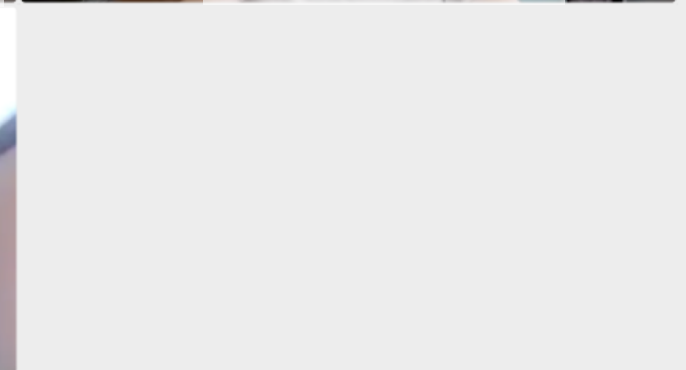
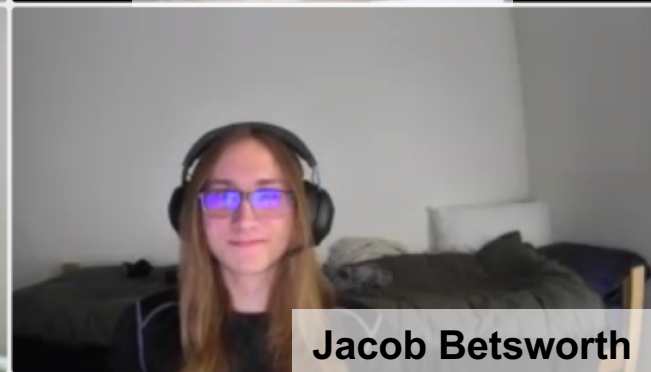
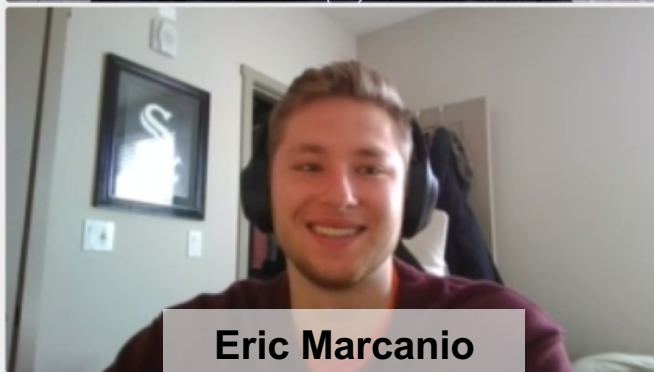
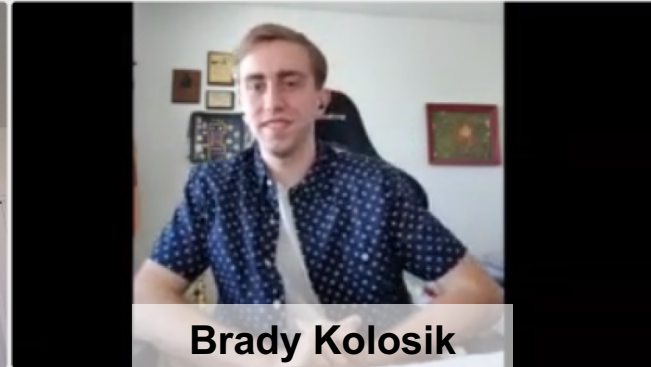
To try the simulator, go to the class web page and follow the link.



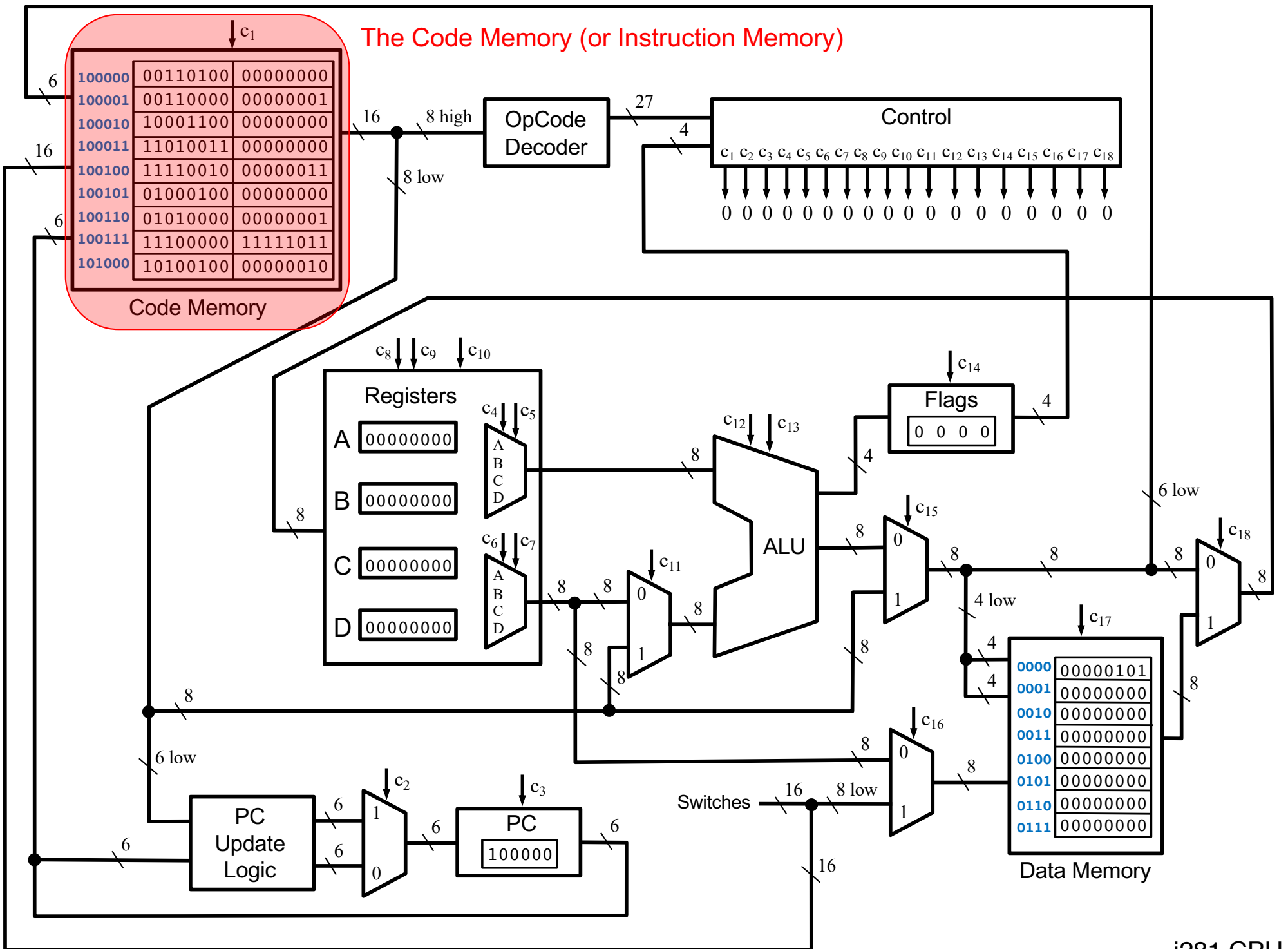
# The i281 Simulator Was Implemented by a Senior Design Team (Fall 2020/Spring 2021)

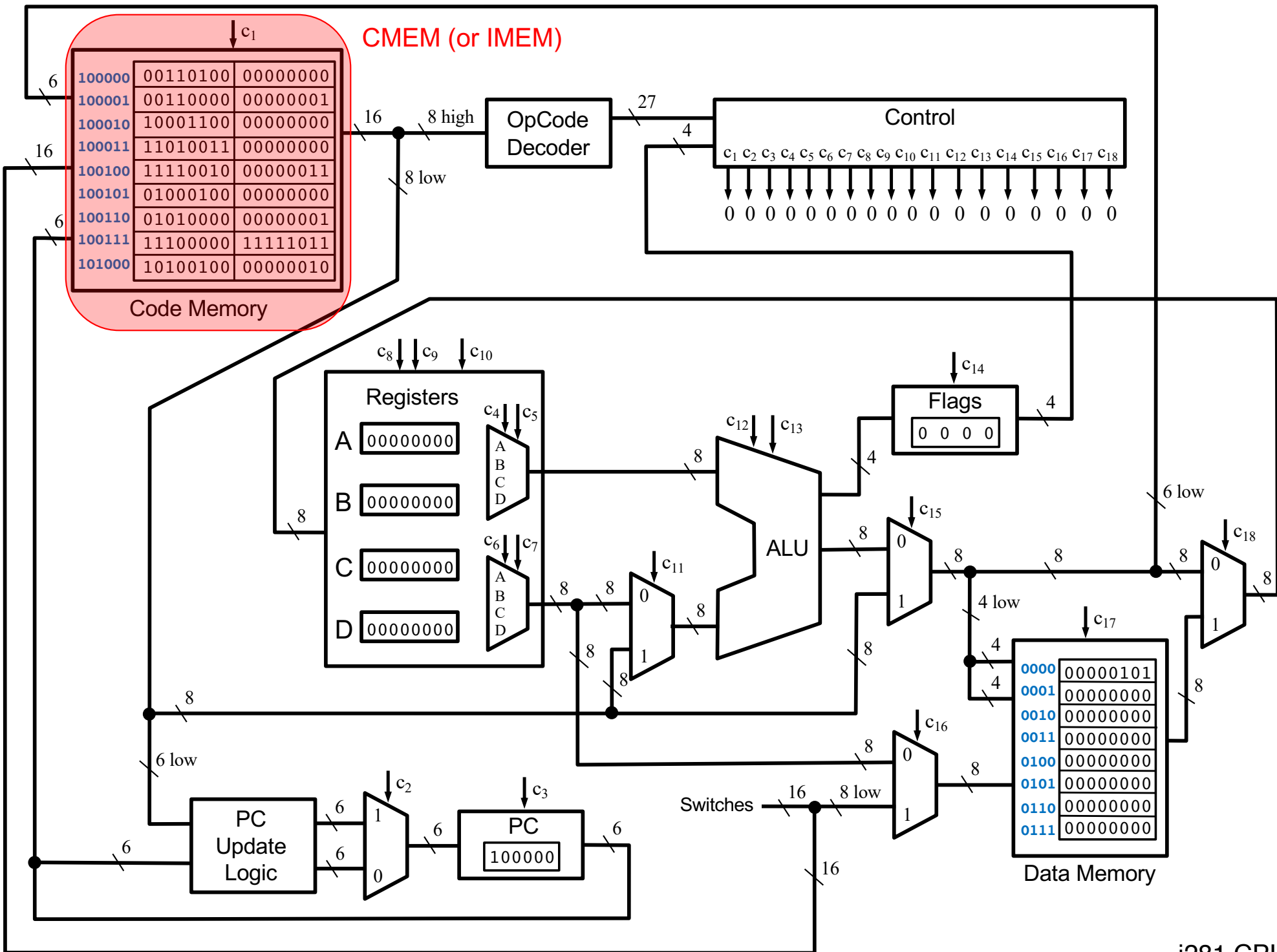


# The i281 Simulator Was Implemented by a Senior Design Team (Fall 2020/Spring 2021)

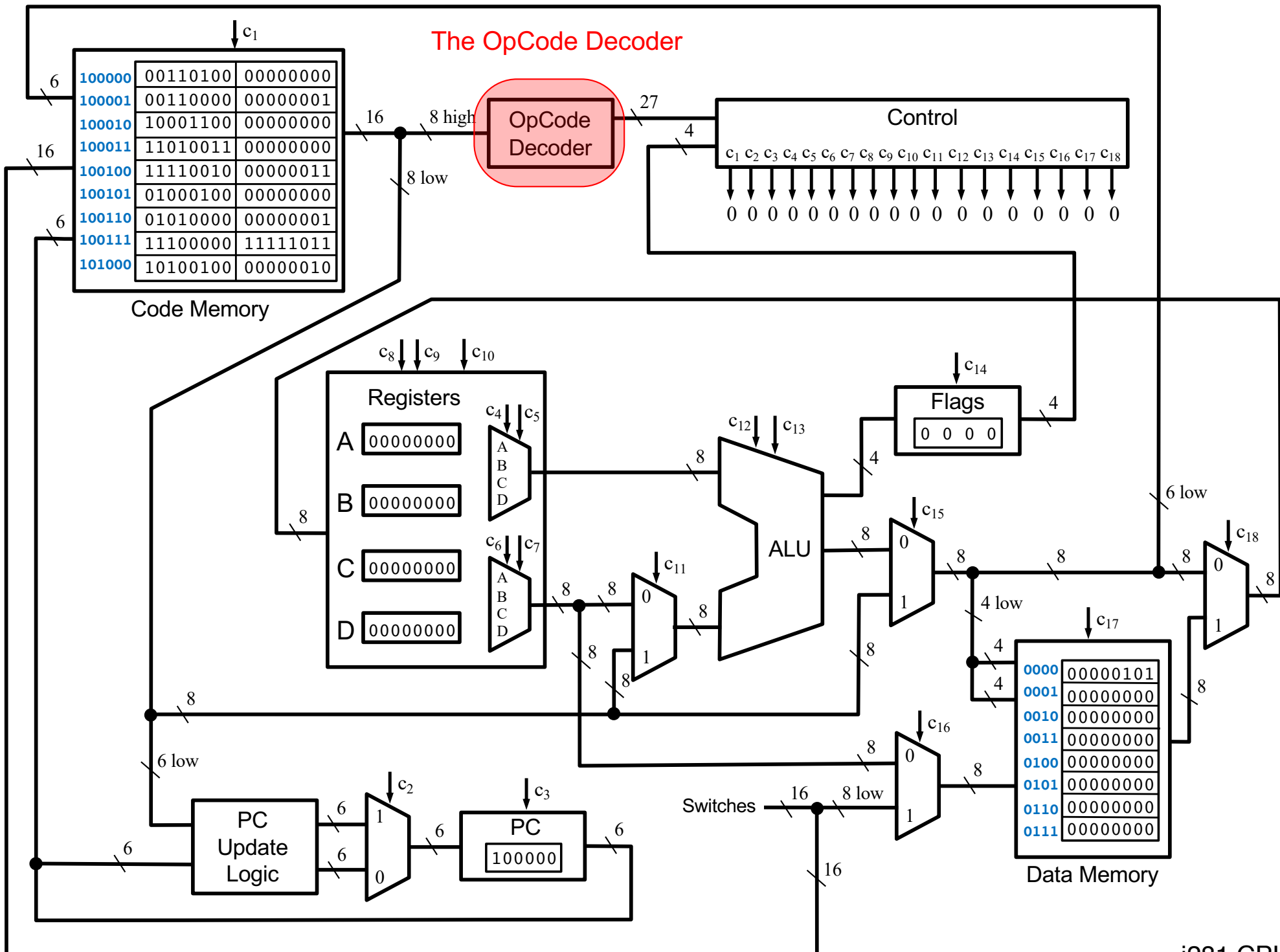


# **The CPU Components**

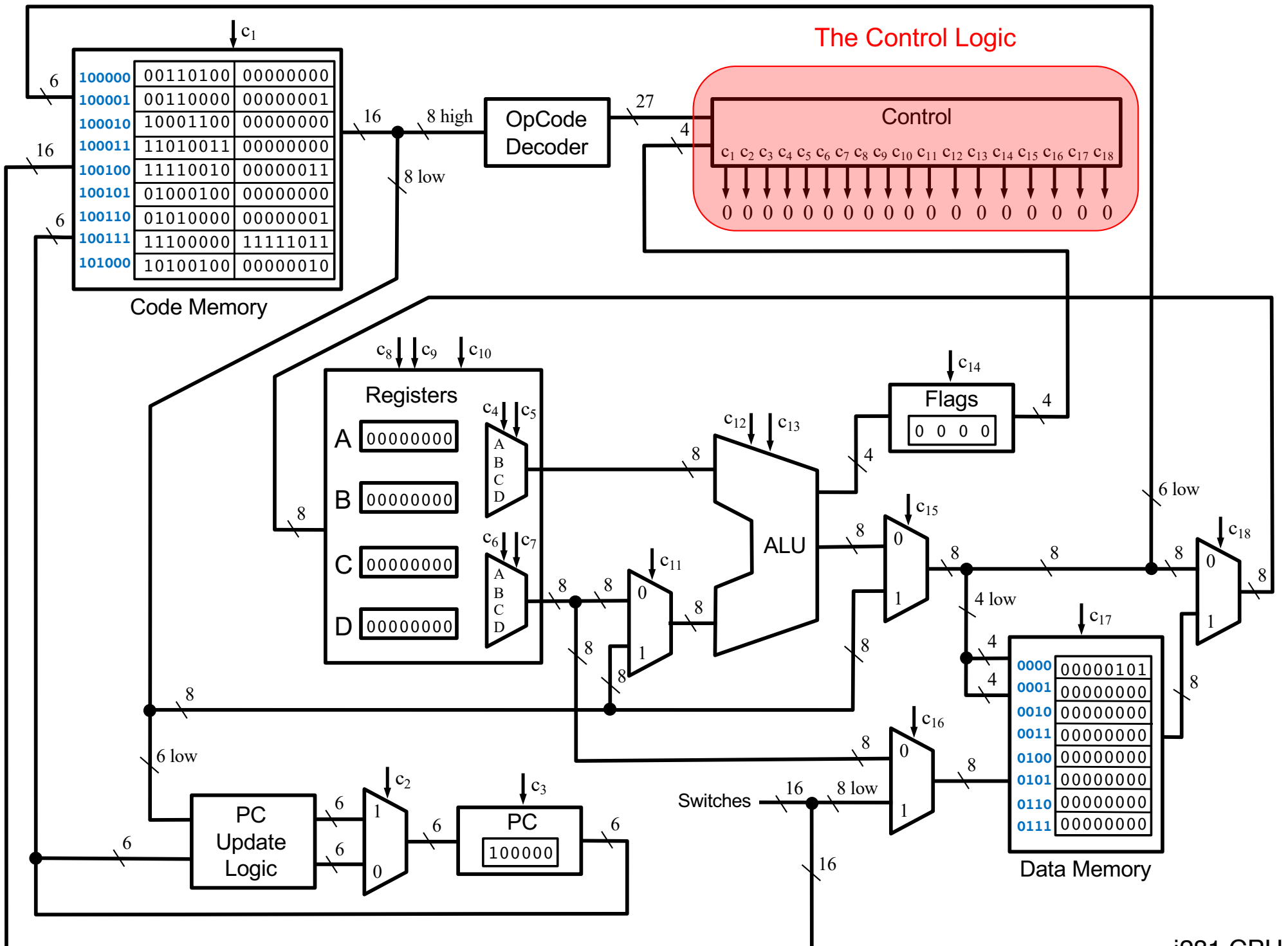


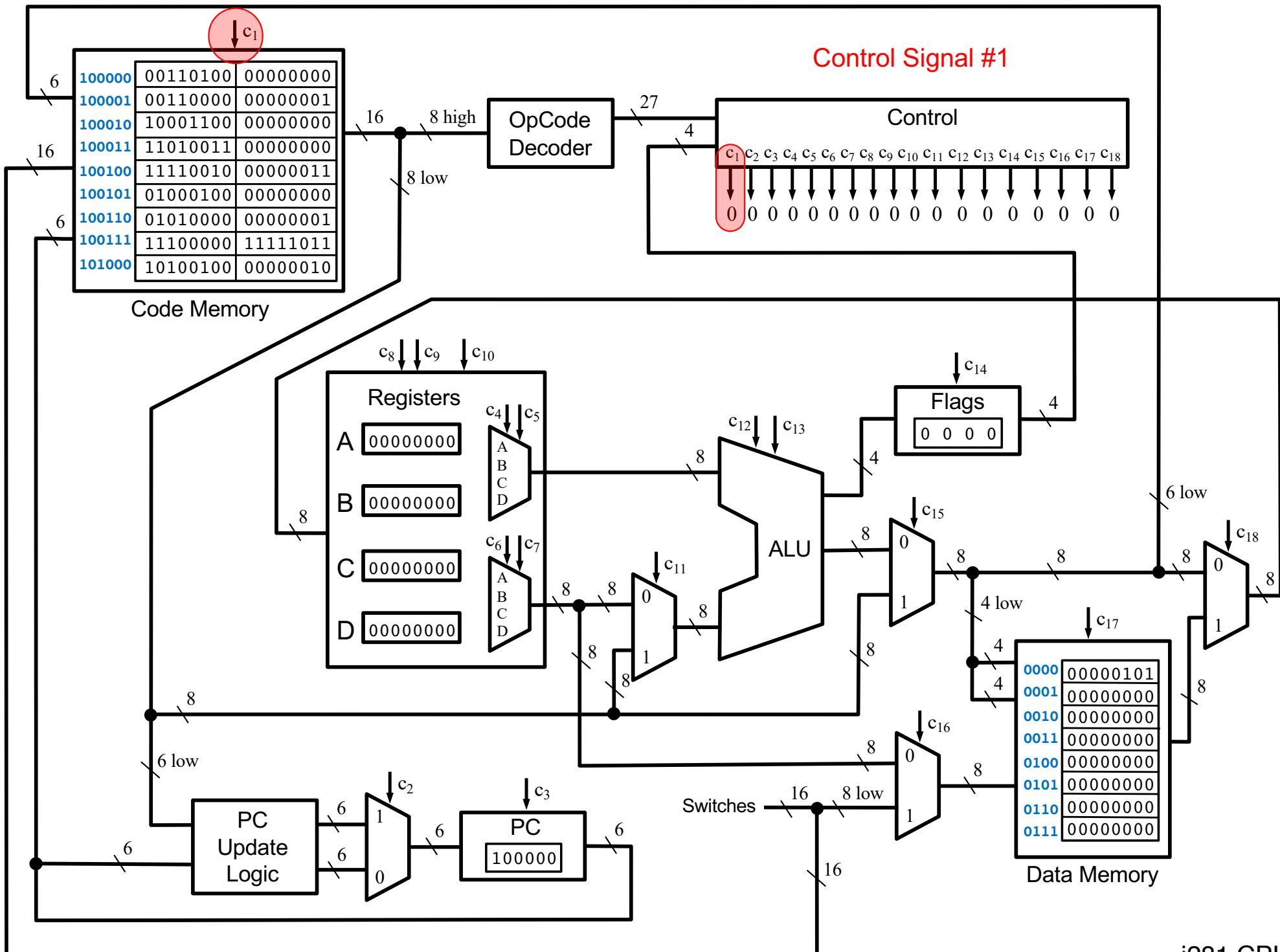


## The OpCode Decoder

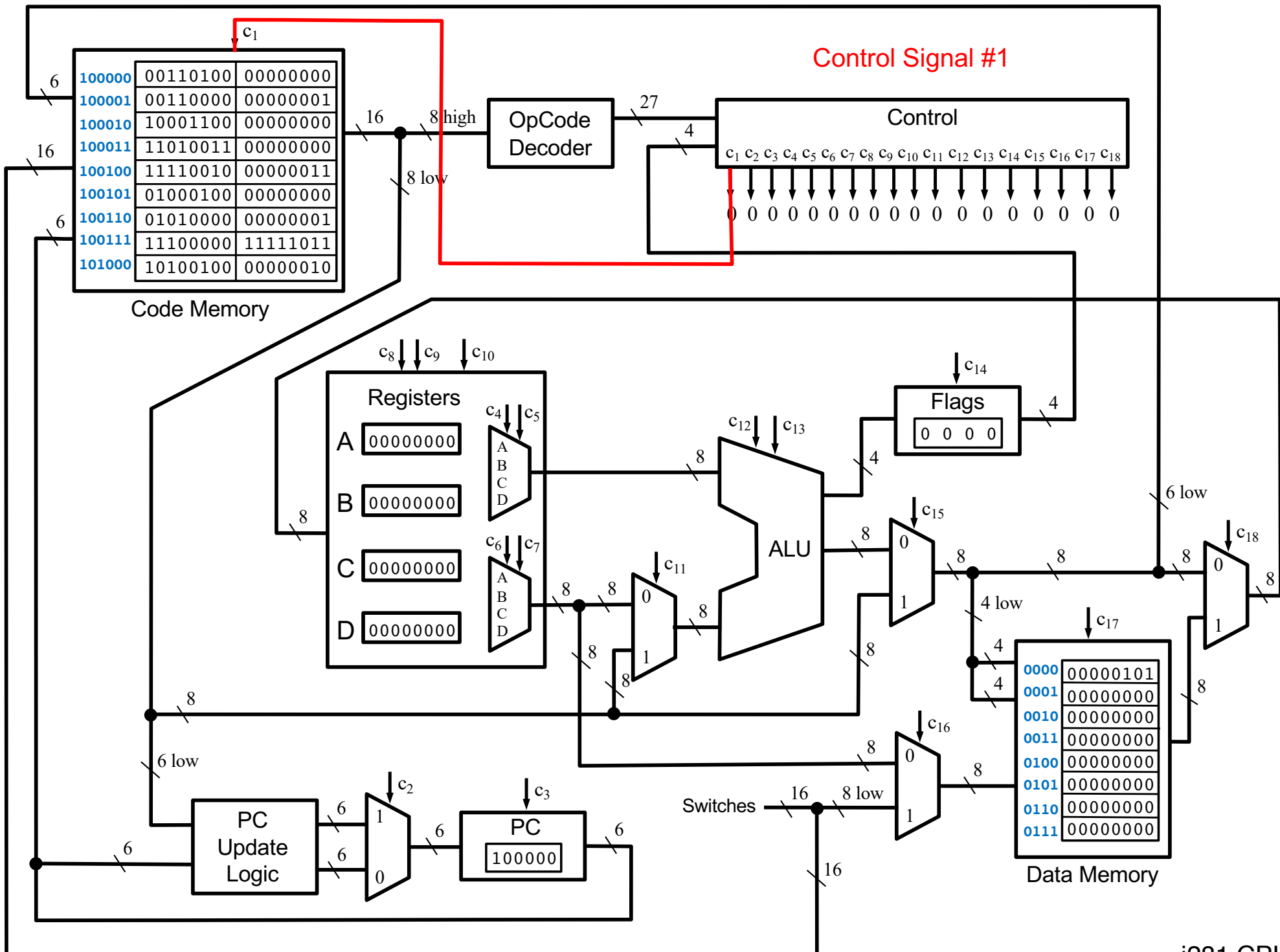


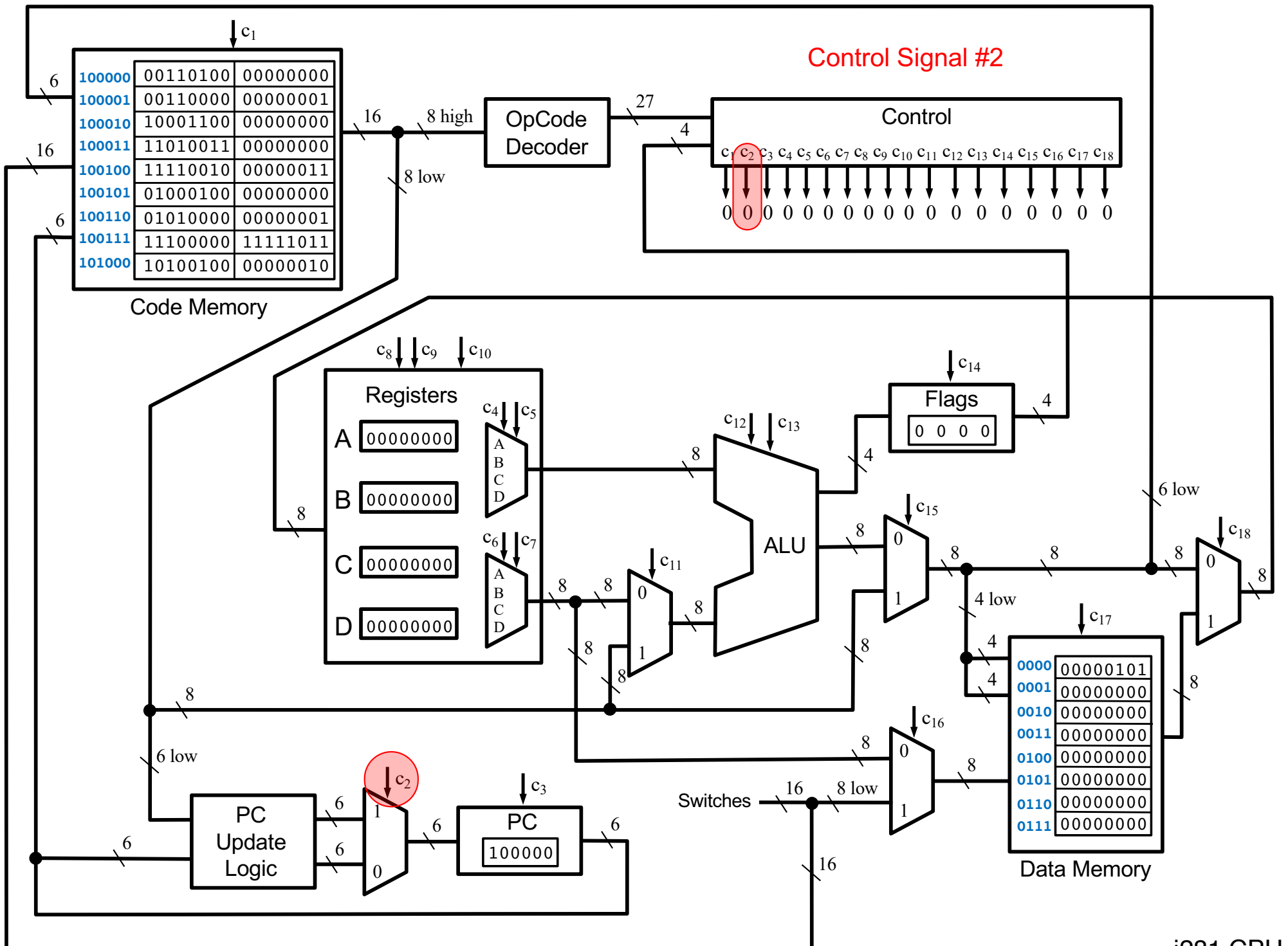


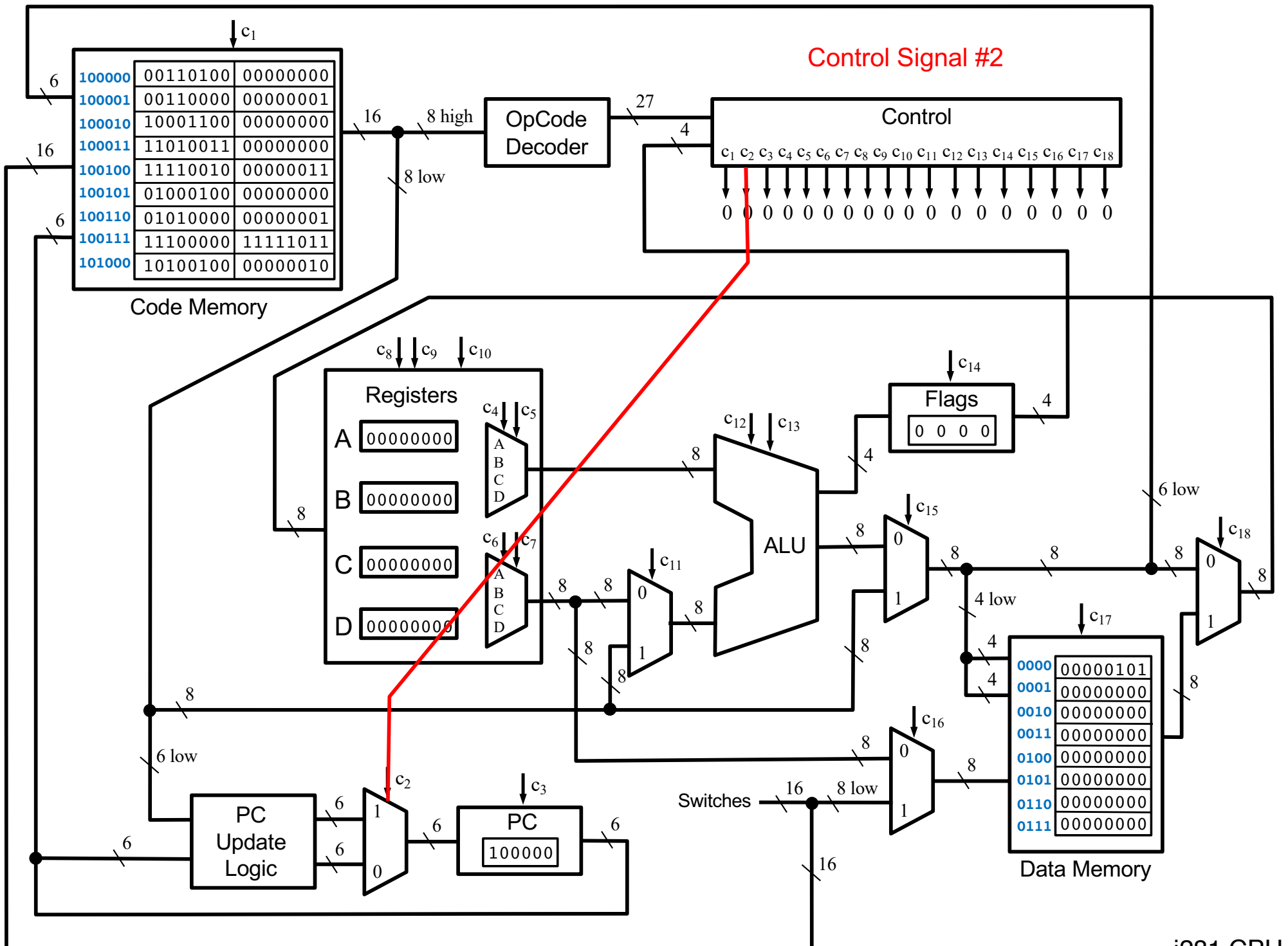


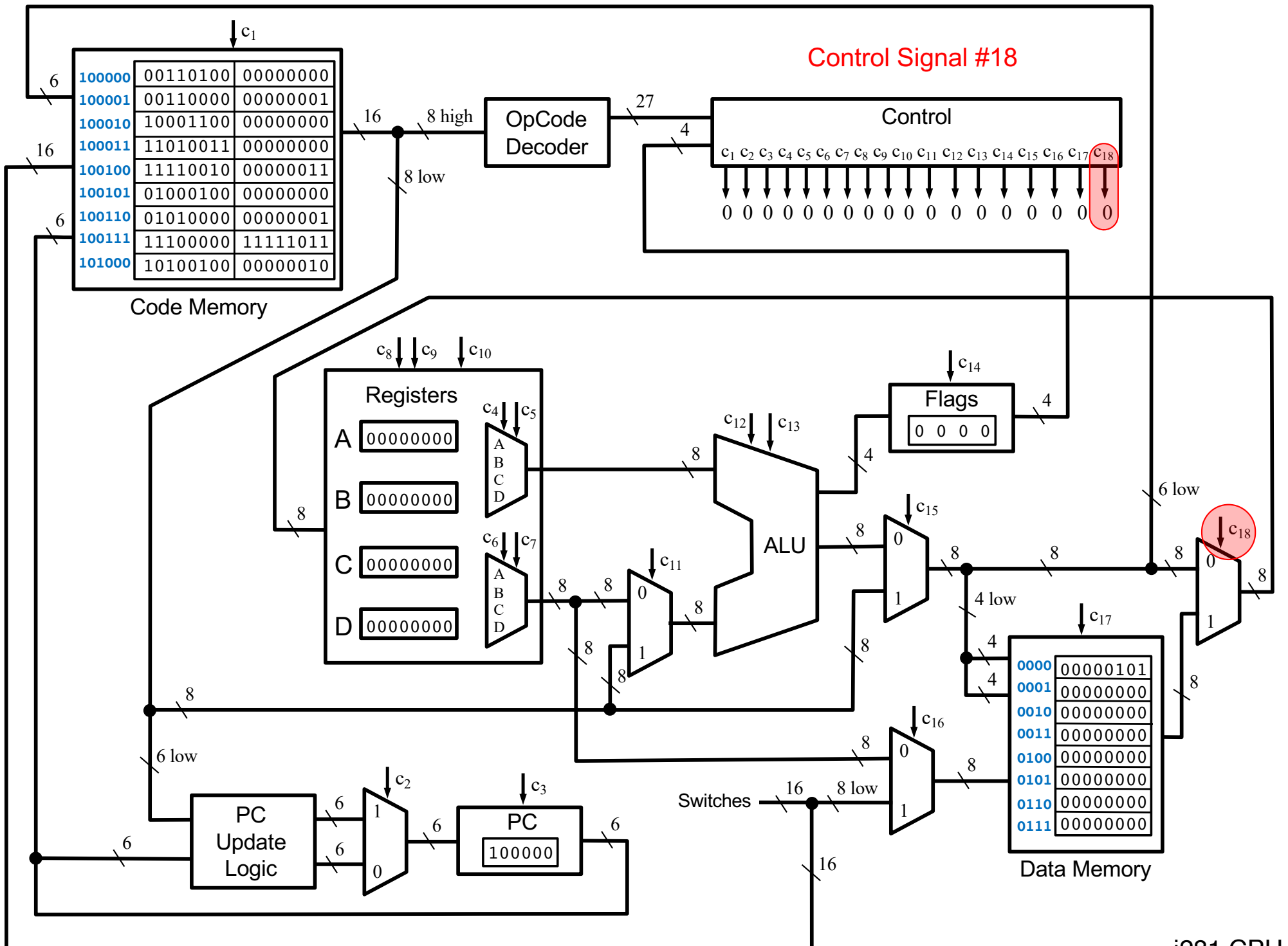


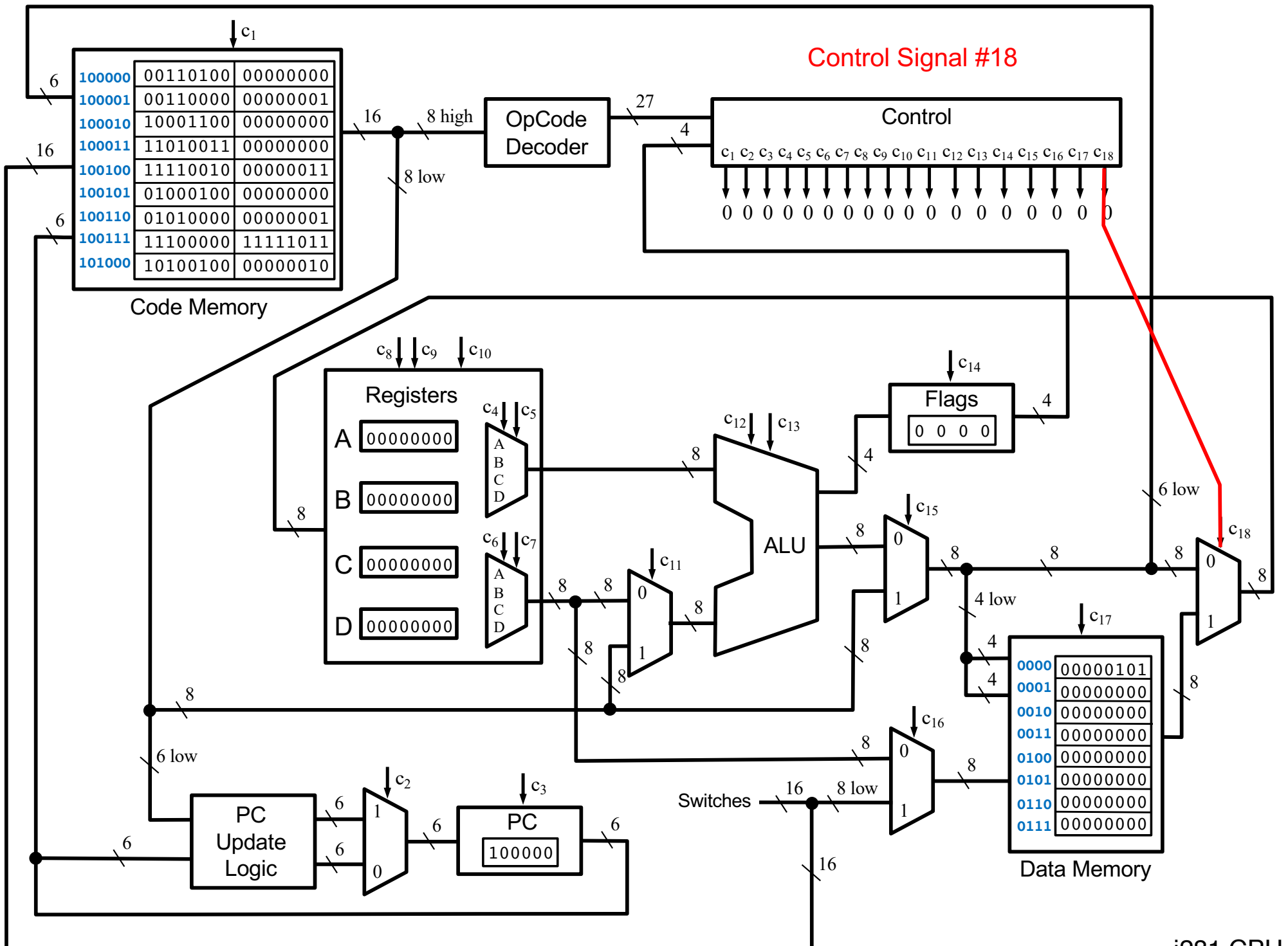


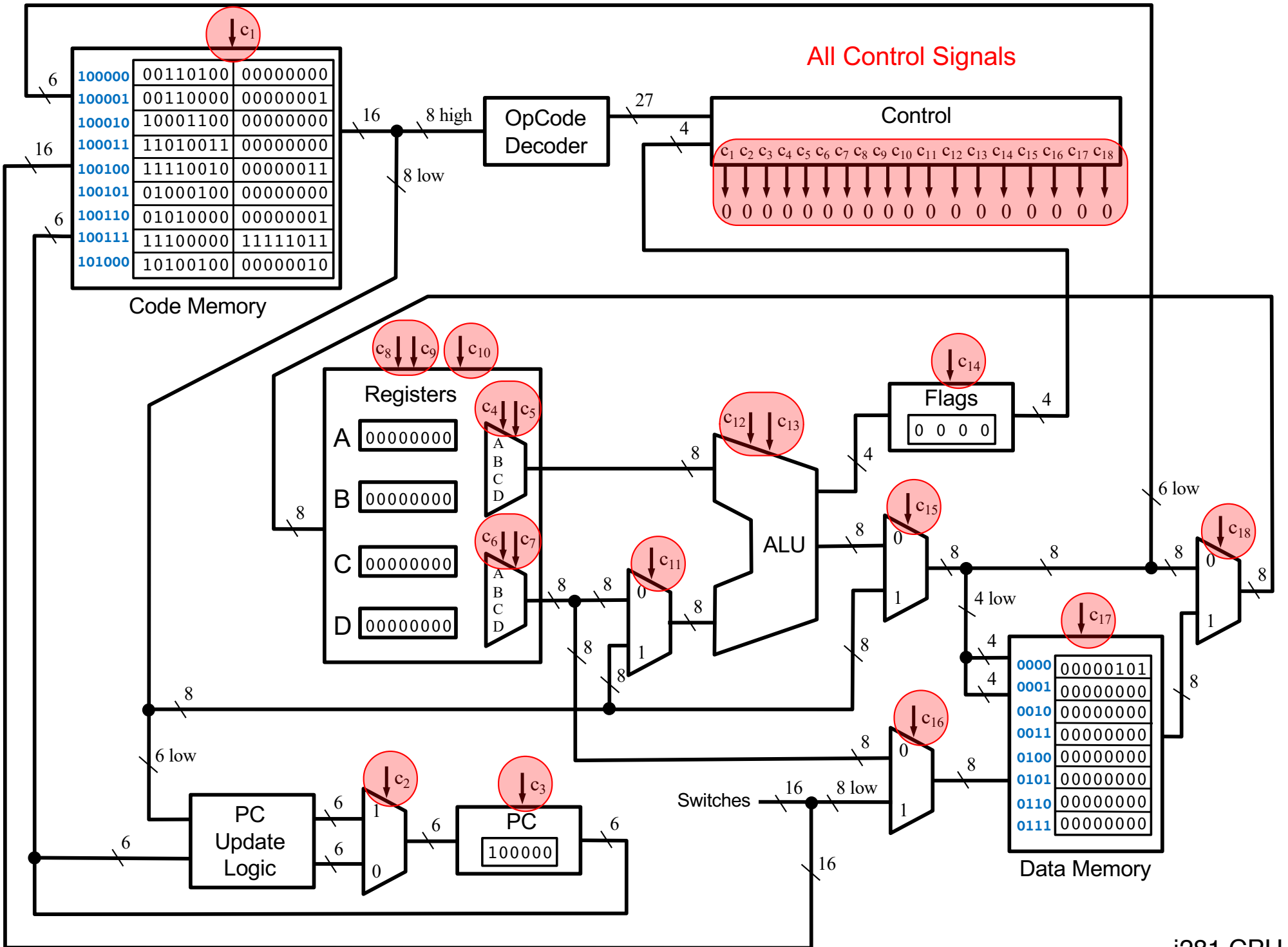


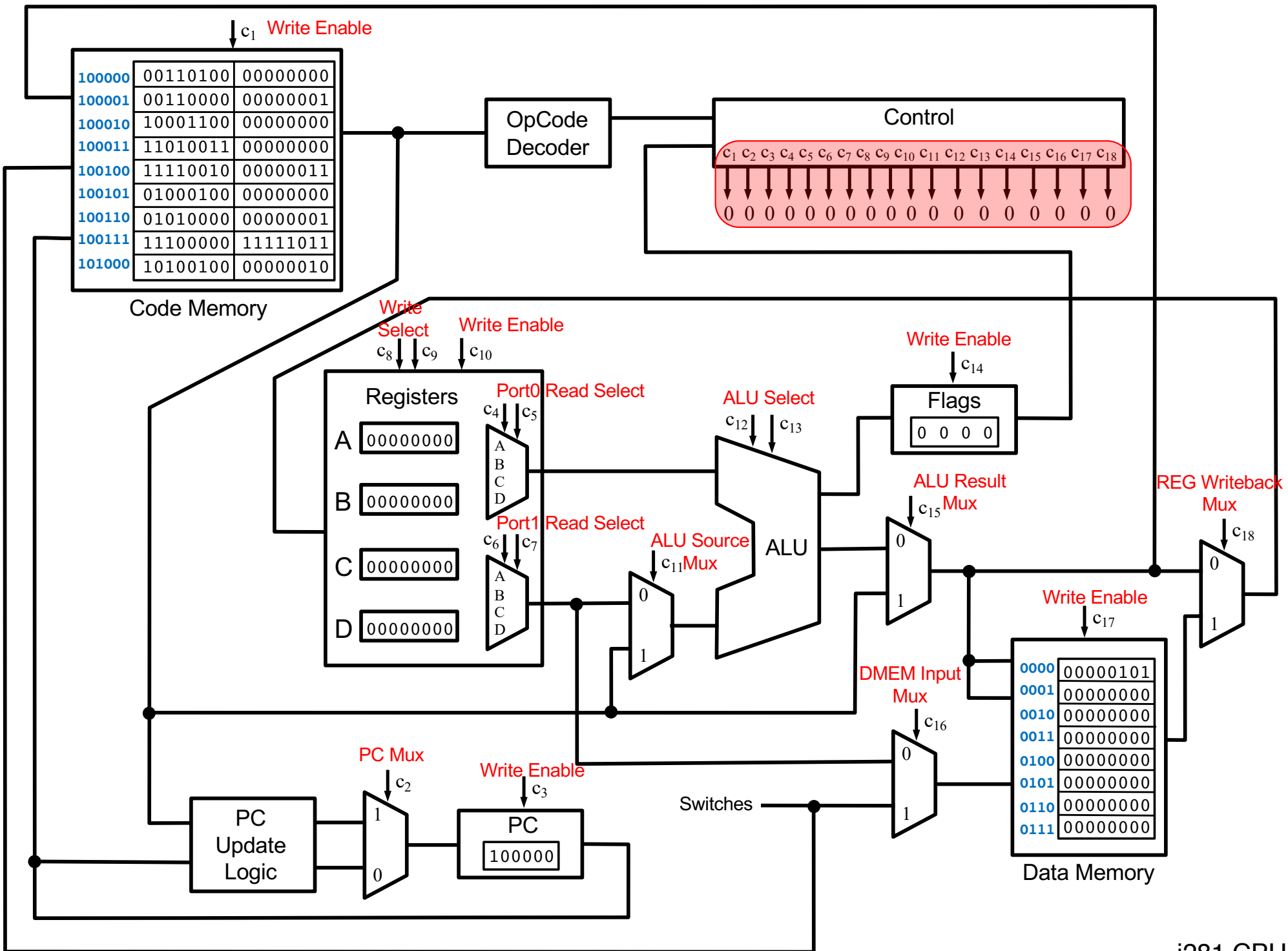




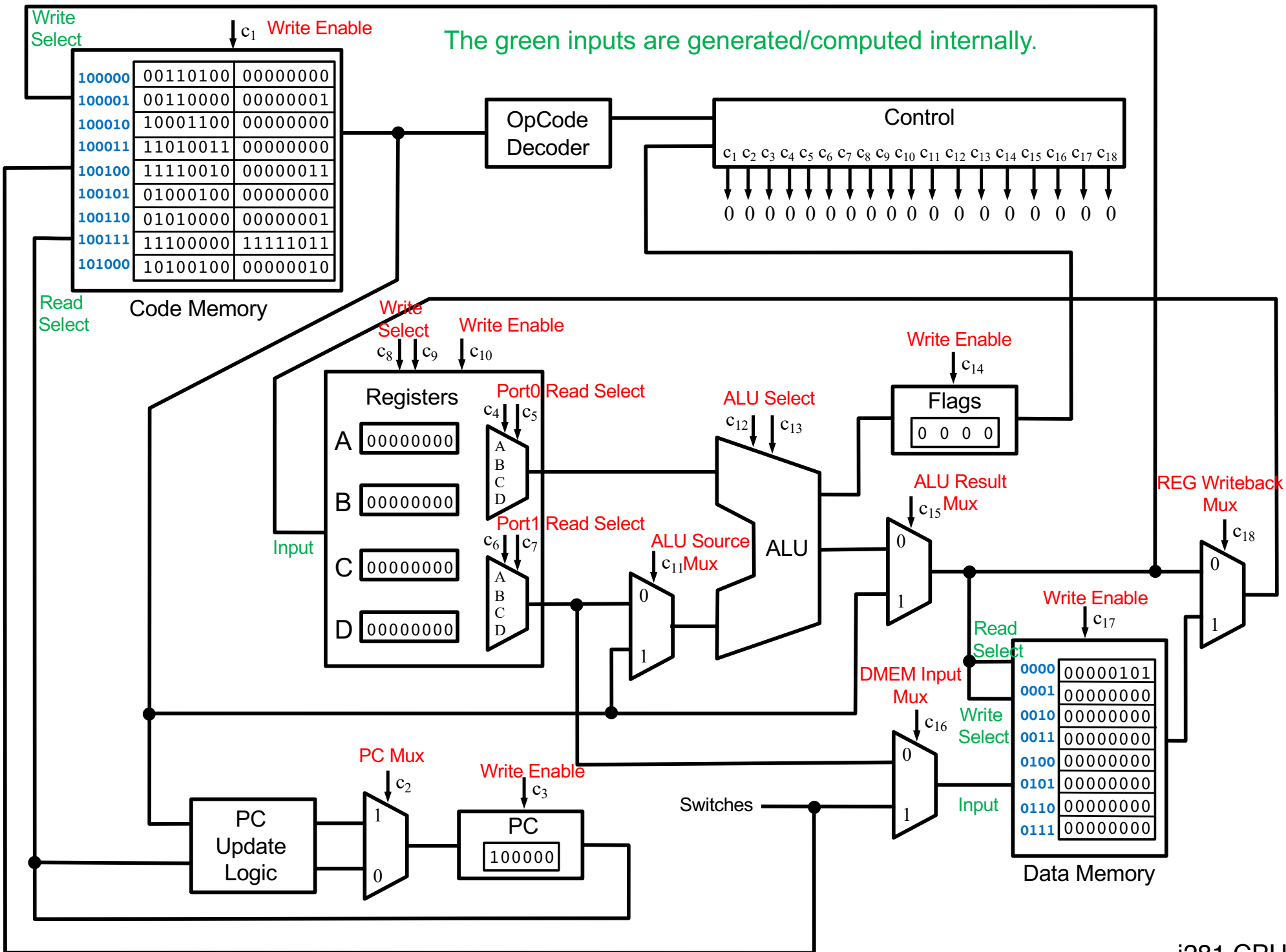






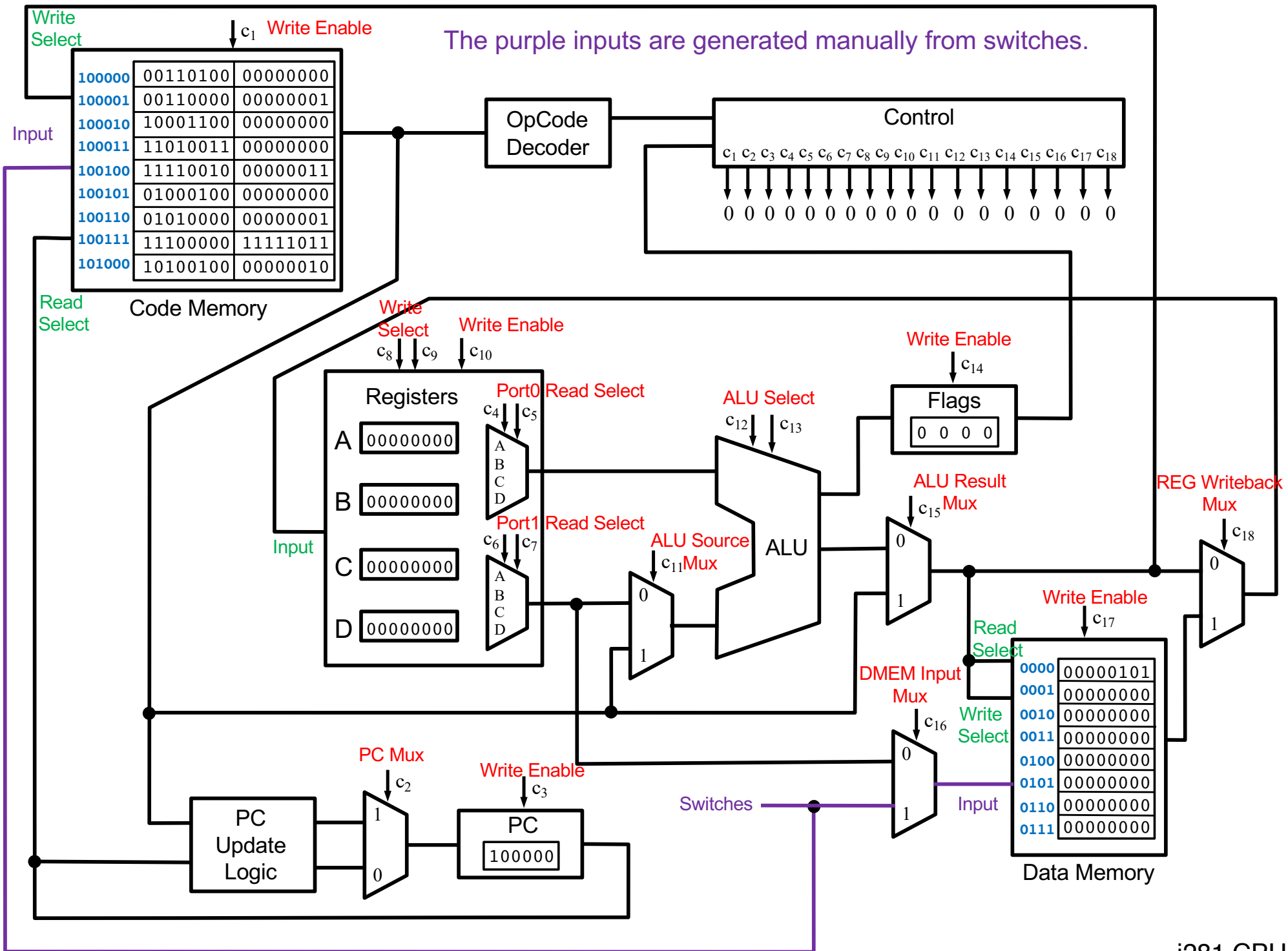


The green inputs are generated/computed internally.

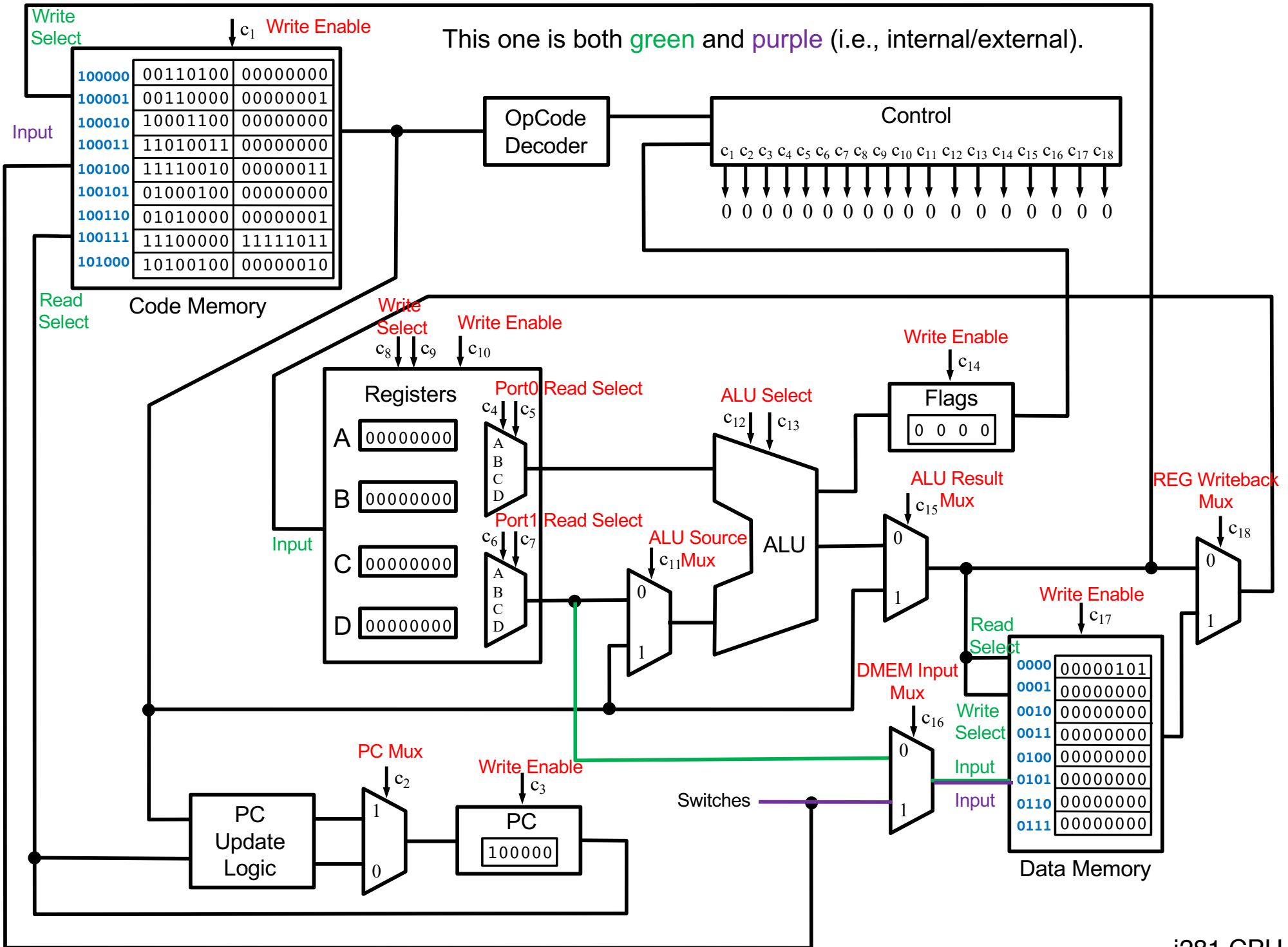


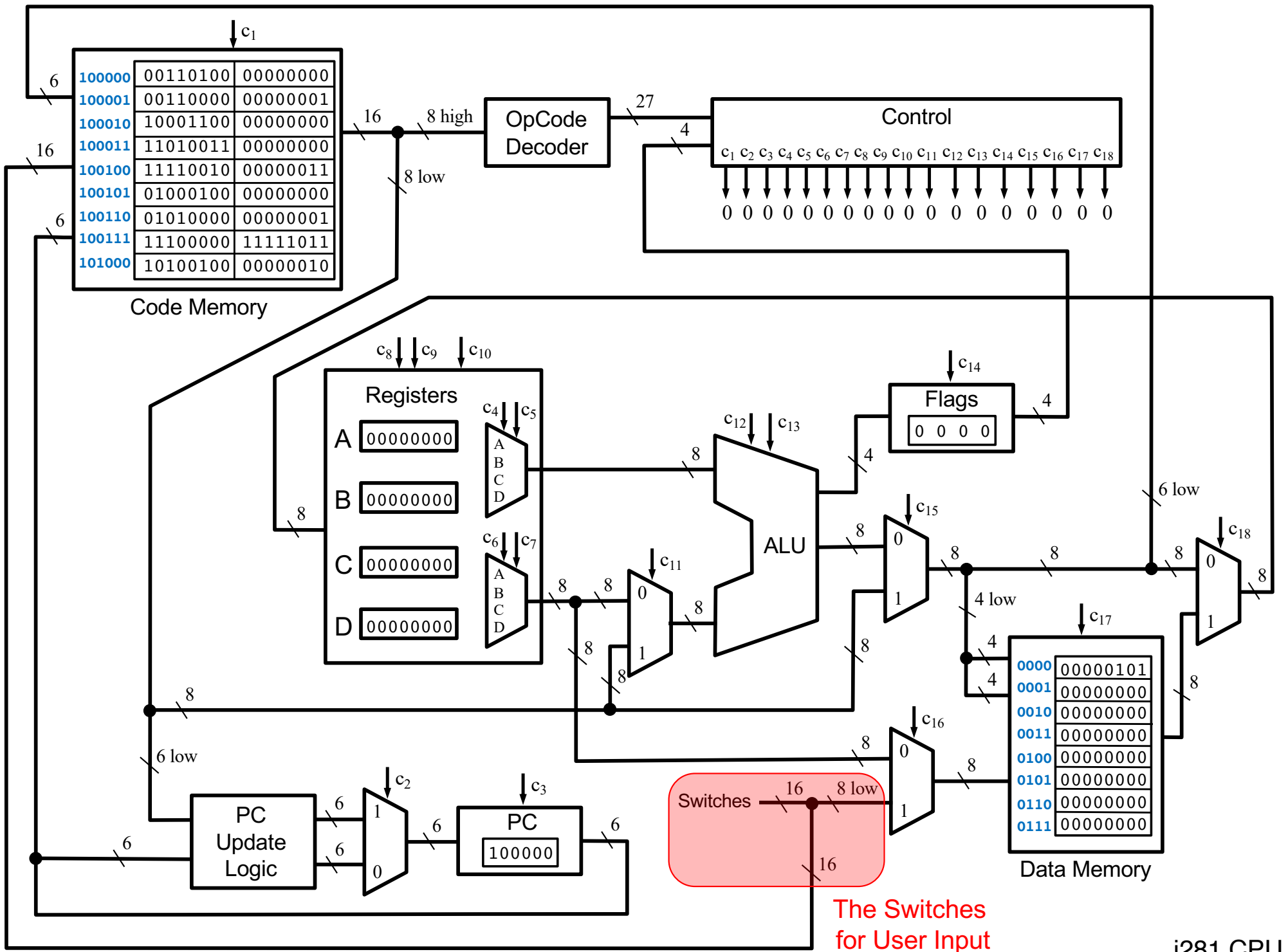


The purple inputs are generated manually from switches.

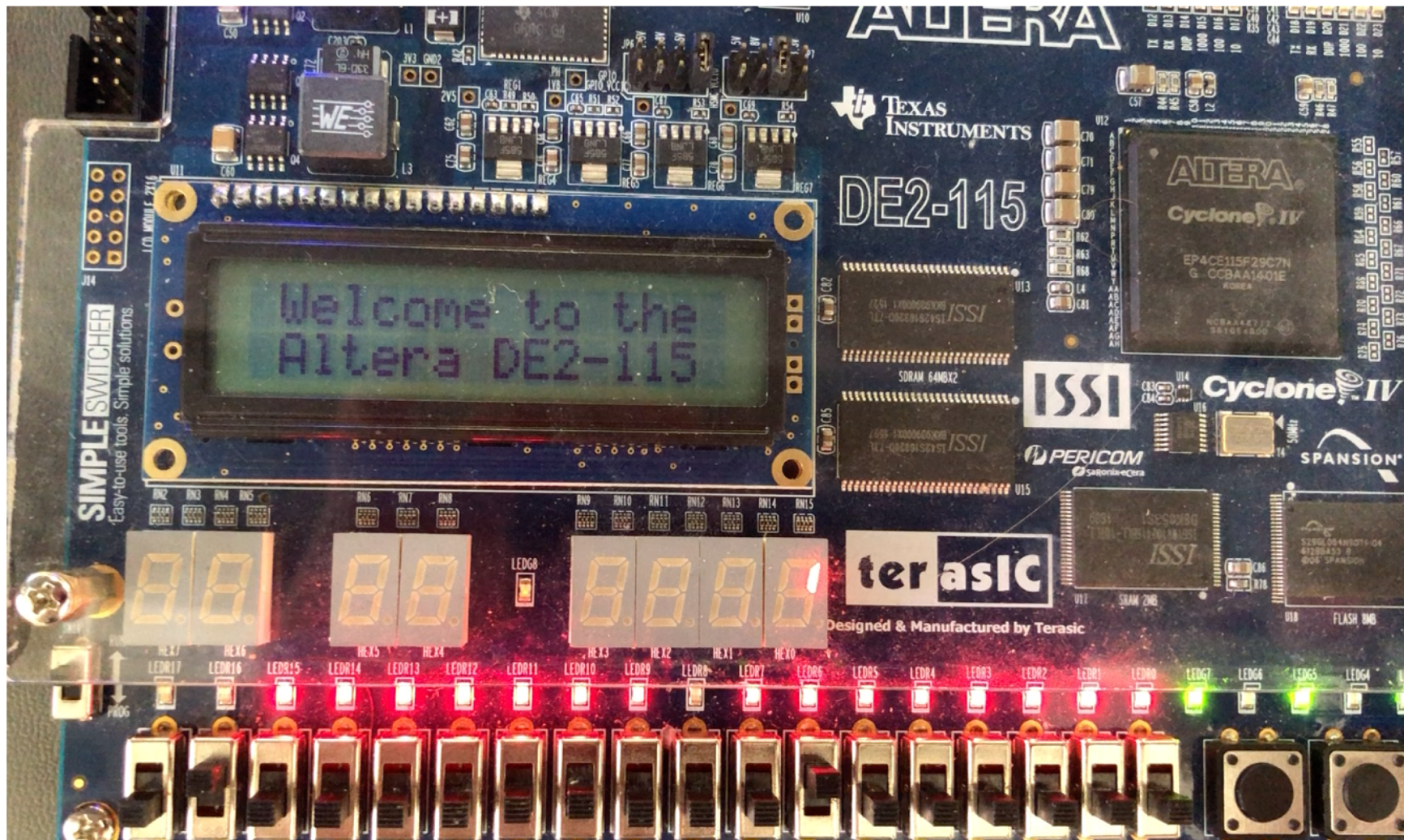


This one is both green and purple (i.e., internal/external).





The Switches  
for User Input



**SIMPLE SWITCHER**  
Easy-to-use tools. Simple solutions.

Welcome to the  
Altera DE2-115

HEX7  
LED17

HEX6  
LED16

HEX5  
LED15

**terasic**  
Designed & Manufactured by Terasic

**ALTERA**  
Cyclone IV  
EP4CE115F29C7H  
G-CBAA14-01E  
K06A

DE2-115

ISSI  
7251-17000605X08  
1L1-08091829S17

SDRAM 64MBX2

ISSI

PERICOM  
Satornia-eCora

Cyclone IV

SPANSION

ISSI  
7251-17000605X08  
1L1-08091829S17

SDRAM 2MB

ISSI  
7251-17000605X08  
1L1-08091829S17

FLASH 1MB

HEX17  
LED17

HEX16  
LED16

HEX15  
LED15

HEX14  
LED14

HEX13  
LED13

HEX12  
LED12

HEX11  
LED11

HEX10  
LED10

HEX9  
LED9

HEX8  
LED8

HEX7  
LED7

HEX6  
LED6

HEX5  
LED5

HEX4  
LED4

HEX3  
LED3

HEX2  
LED2

HEX1  
LED1

LED0

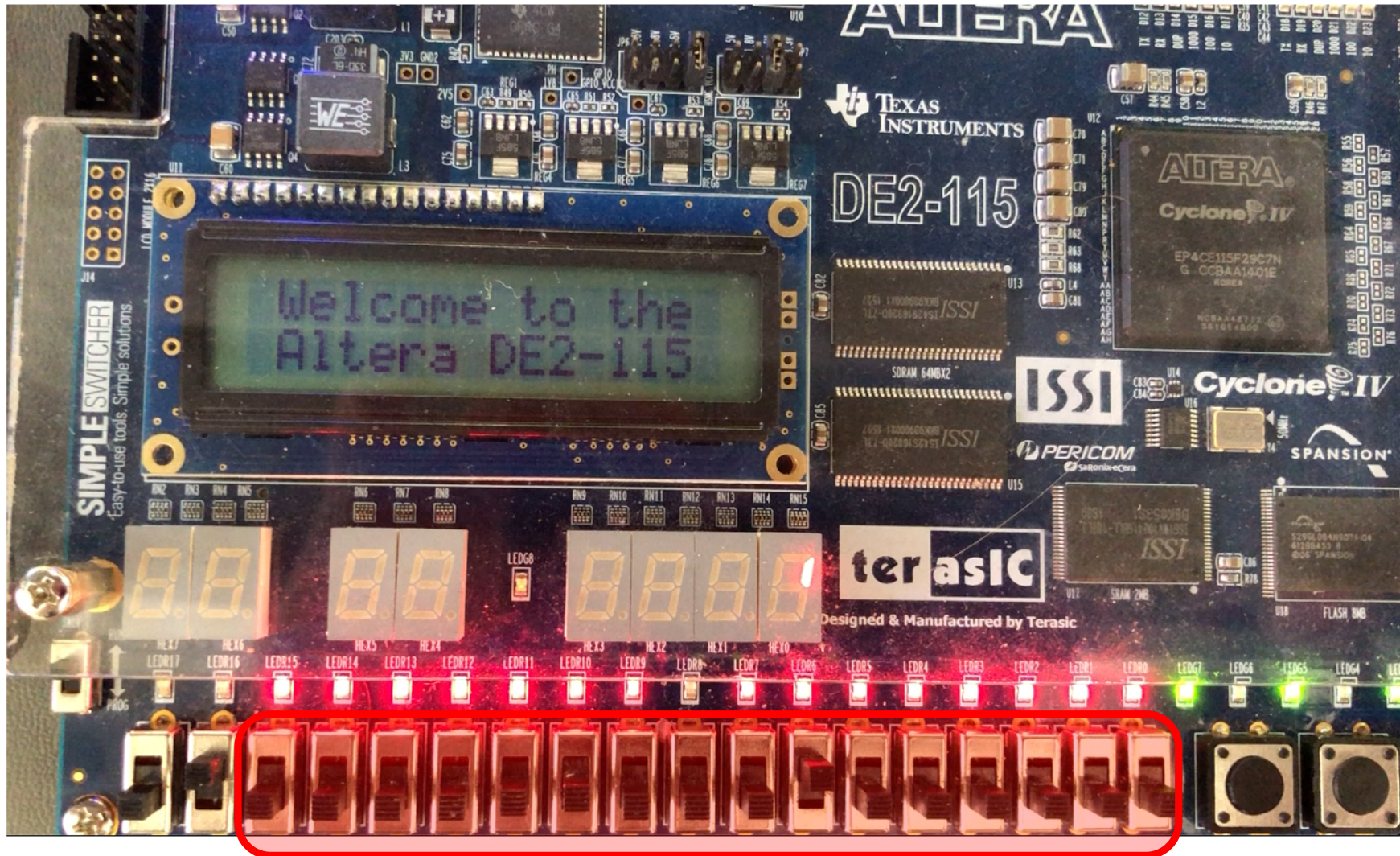
LED7

LED6

LED5

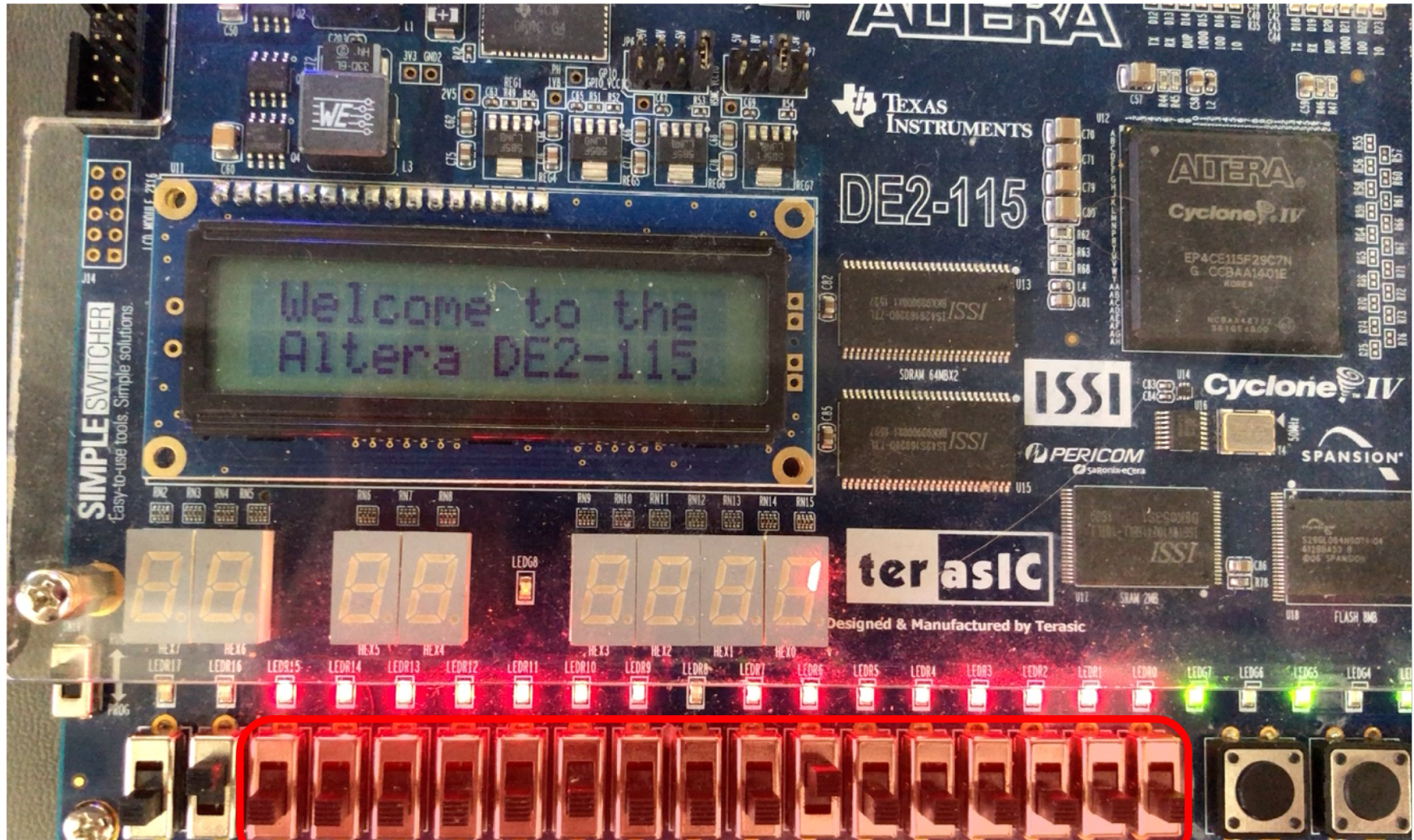
LED4





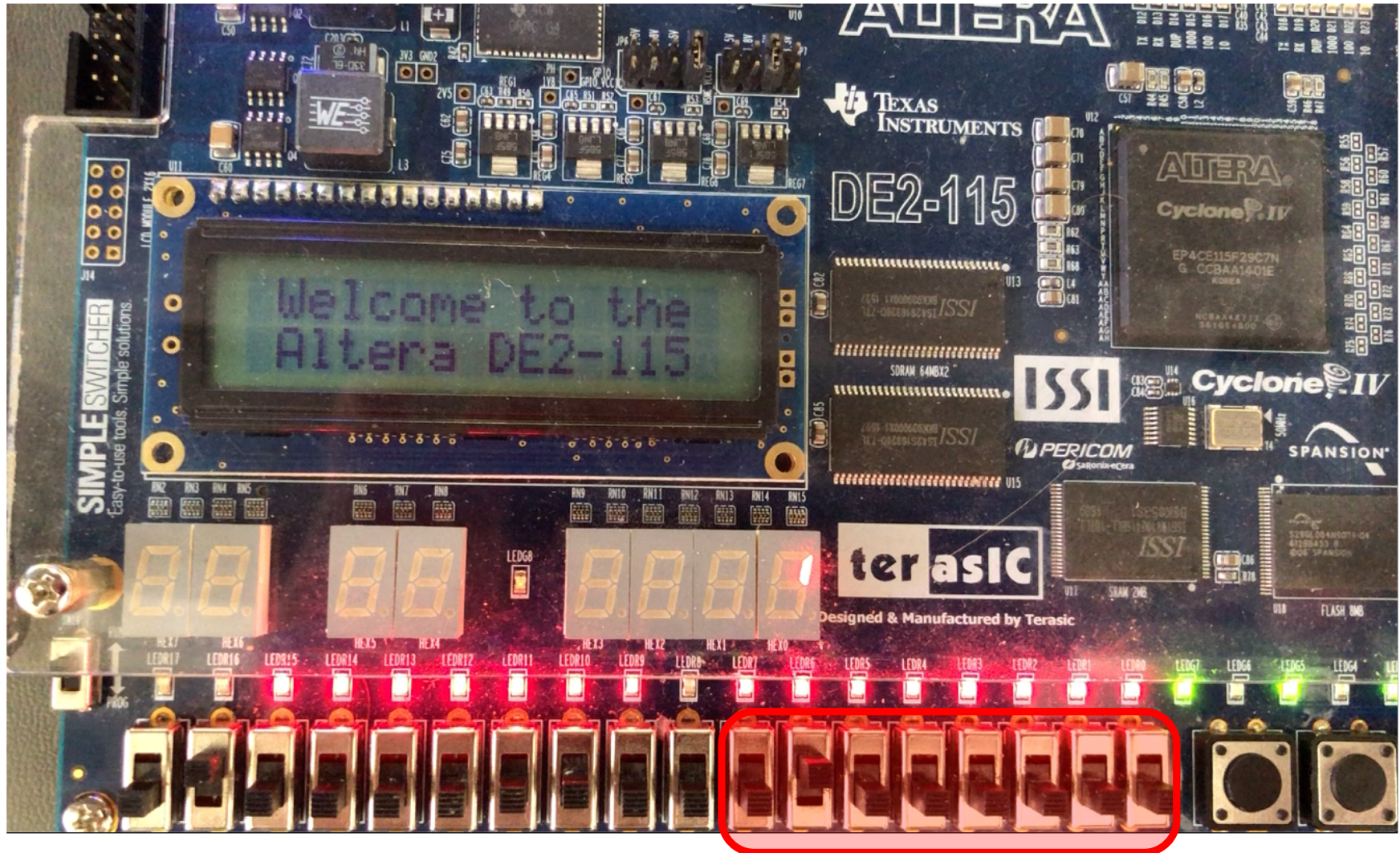
These 16 switches are used for input into the Code Memory.





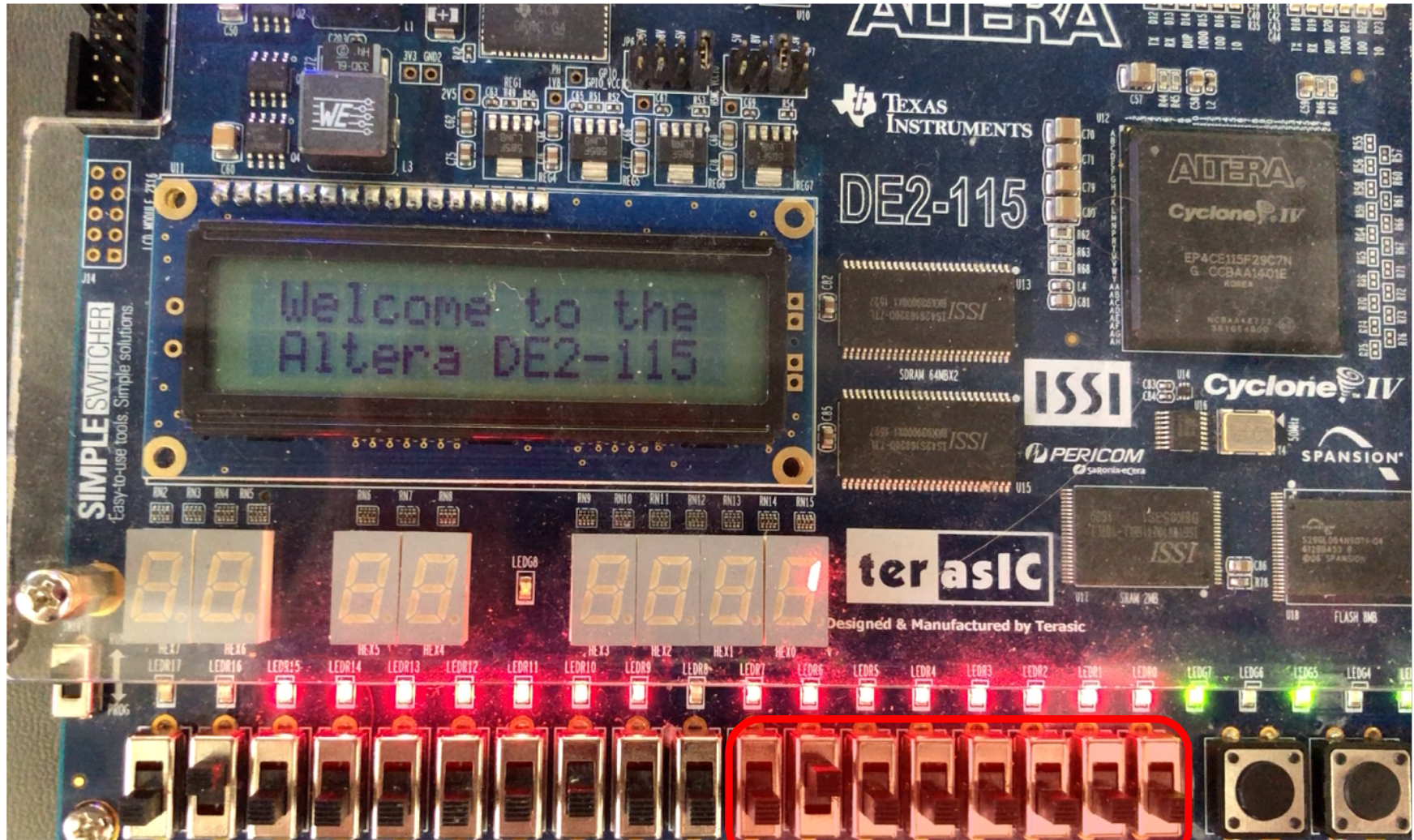
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0





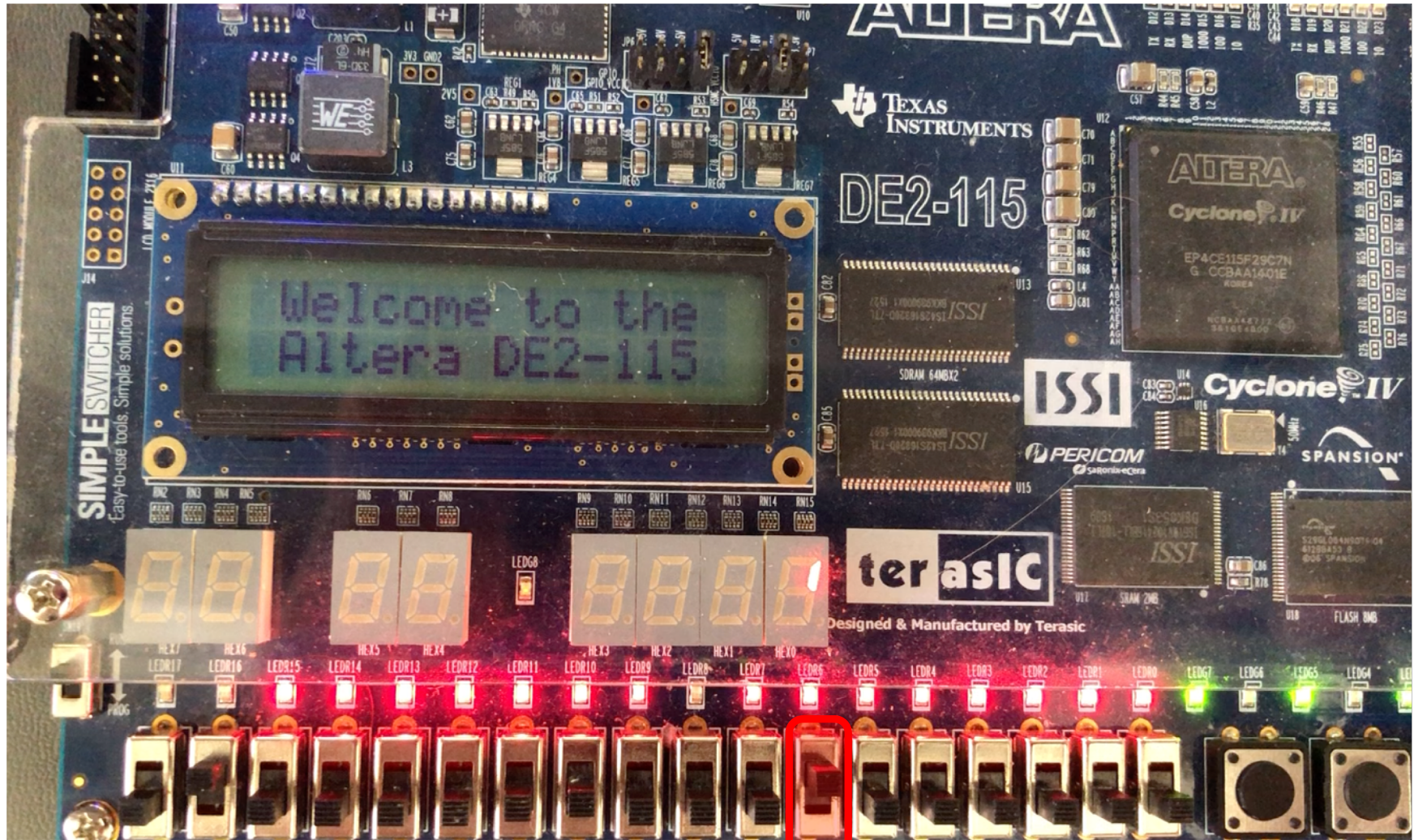
These 8 switches are used for input into the Data Memory.





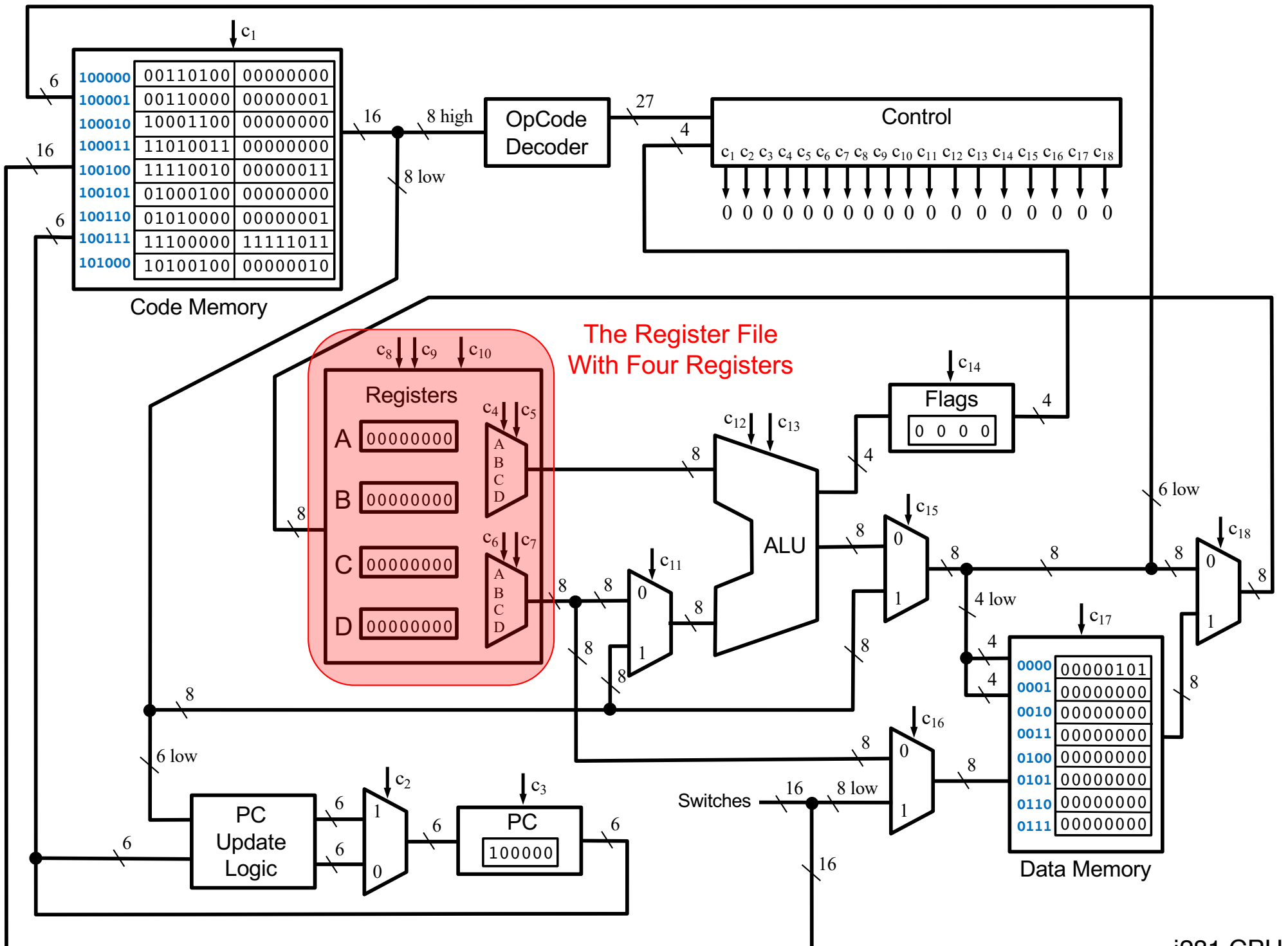
0 1 0 0 0 0 0 0

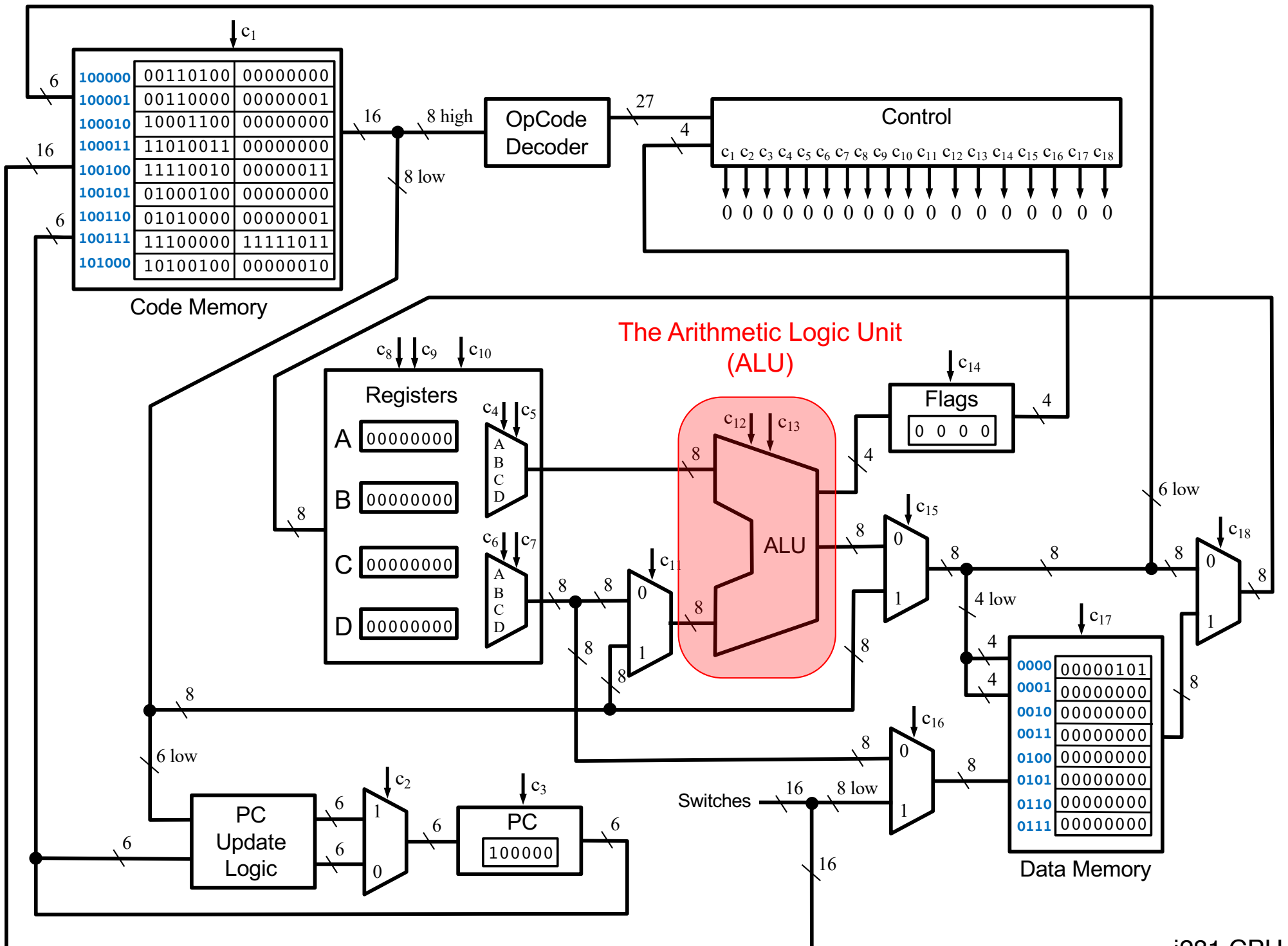


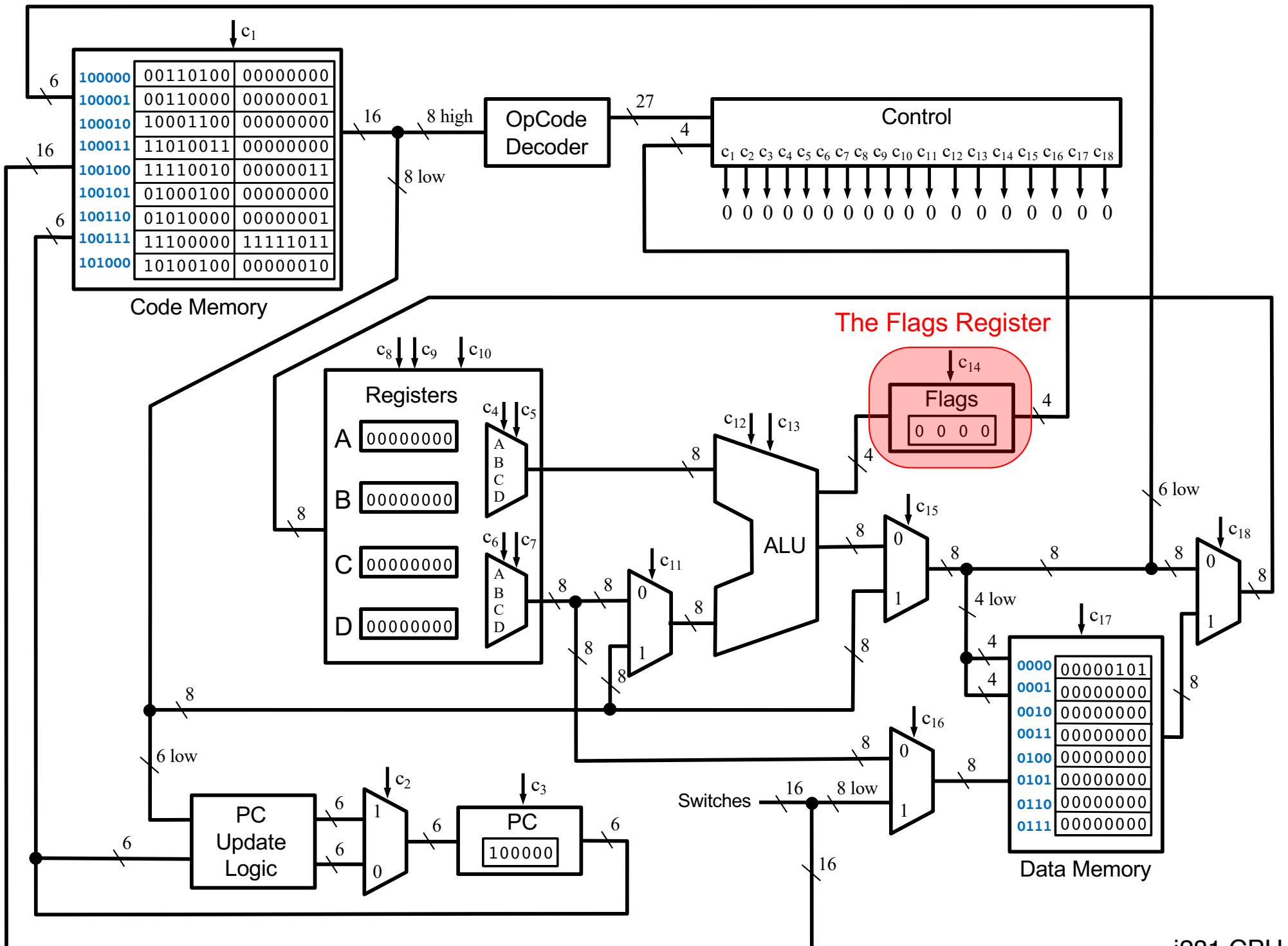


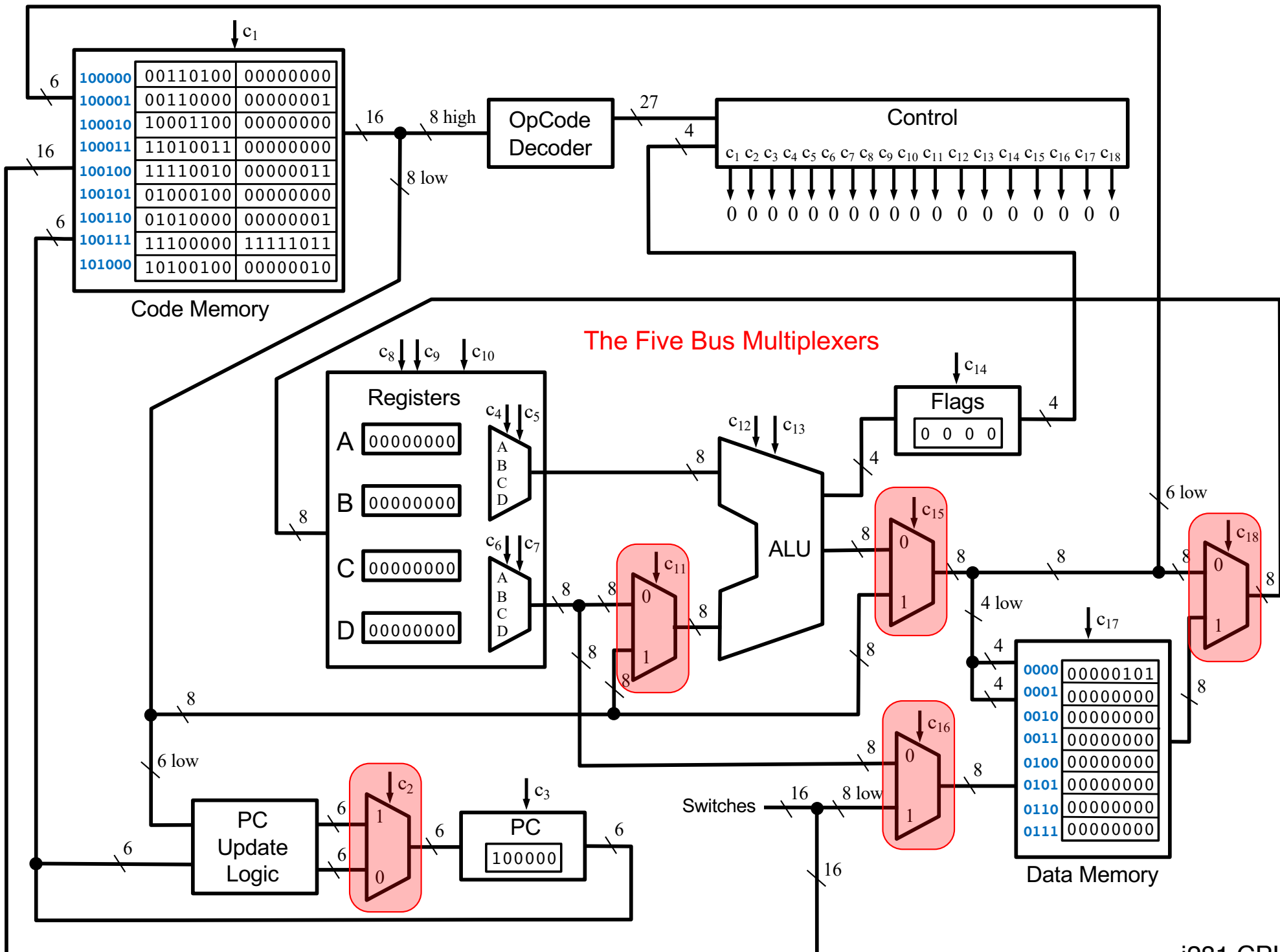
0 1 0 0 0 0 0 0

This switch controls the paddle in PONG

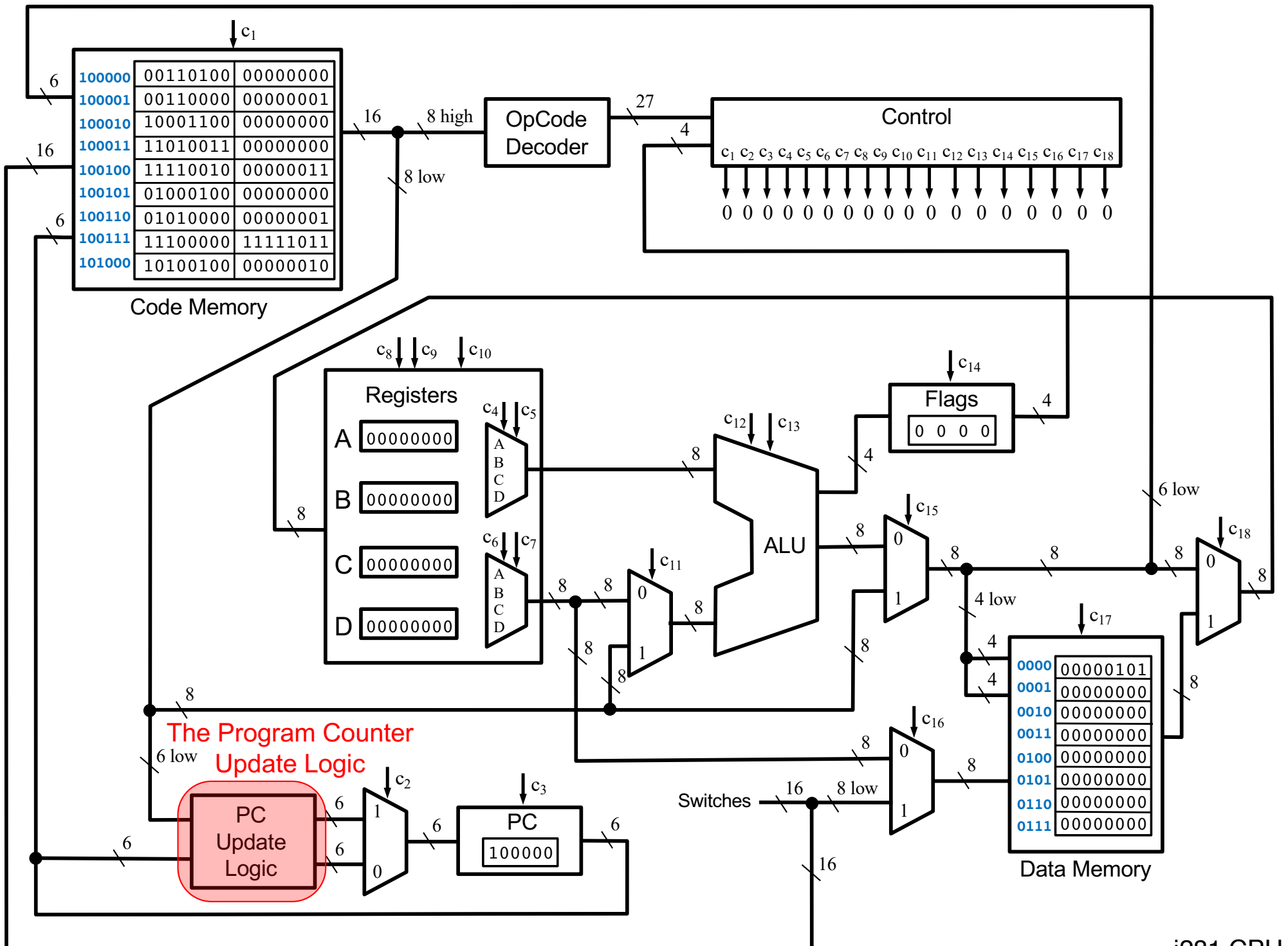


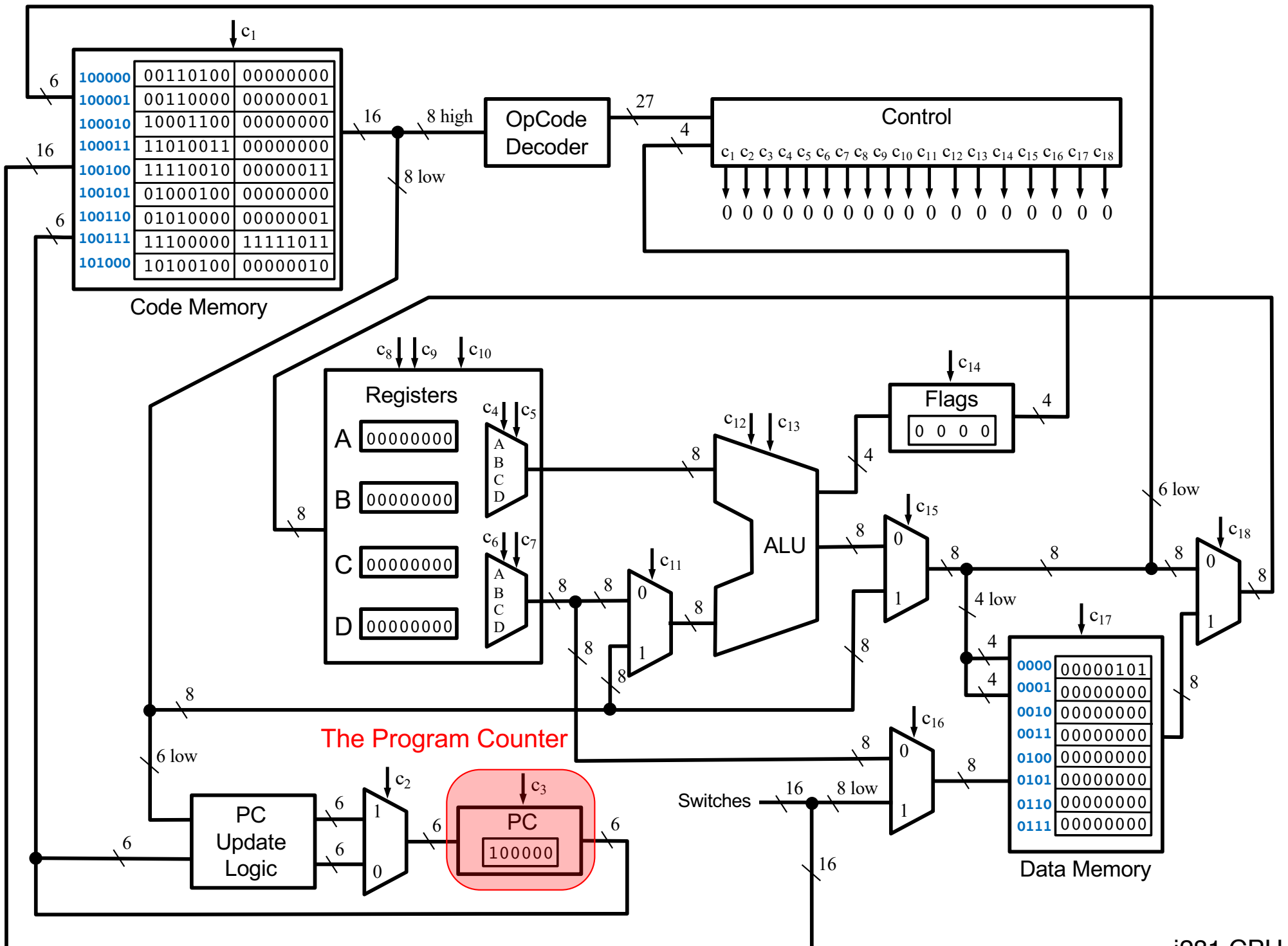


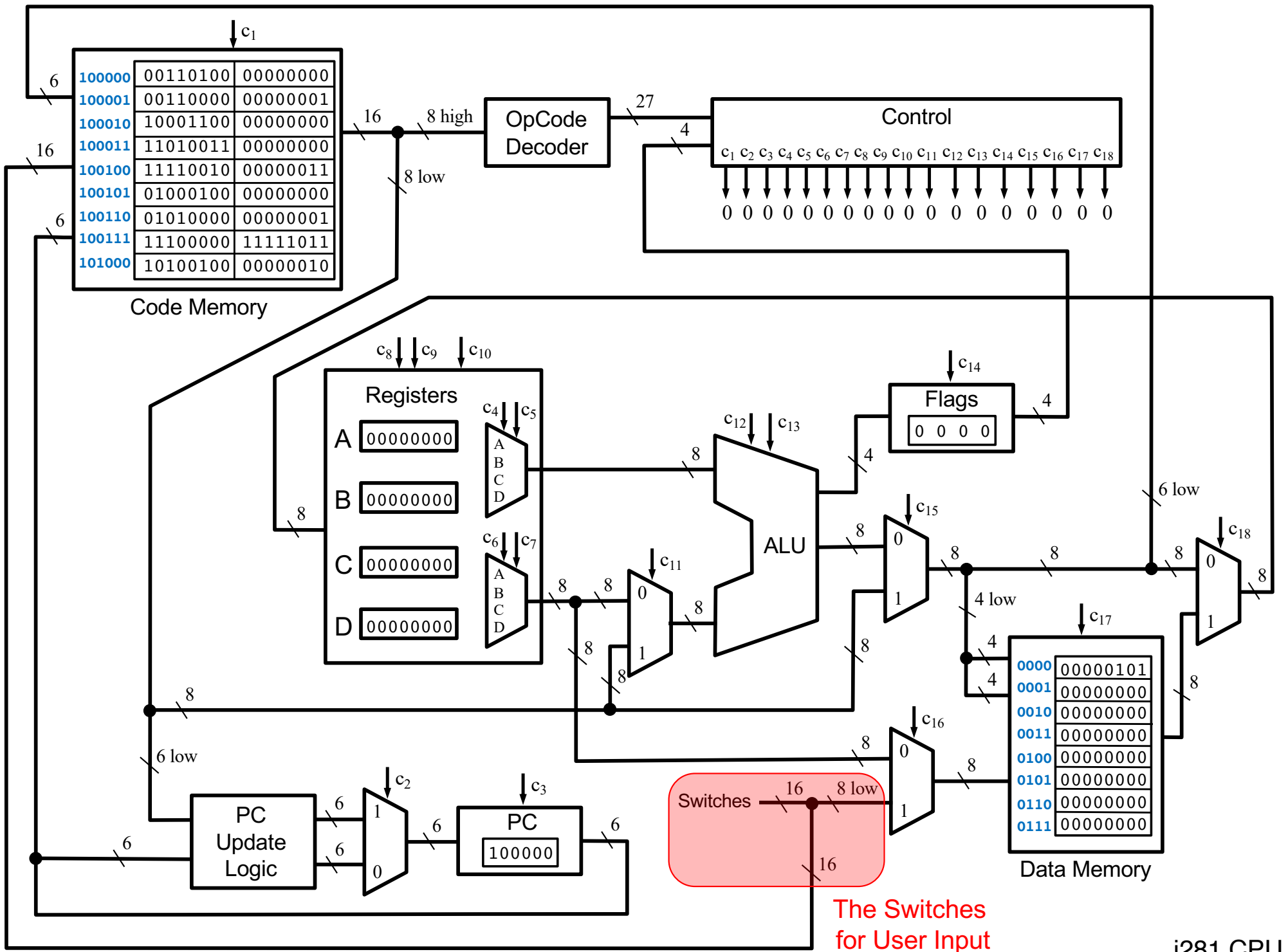




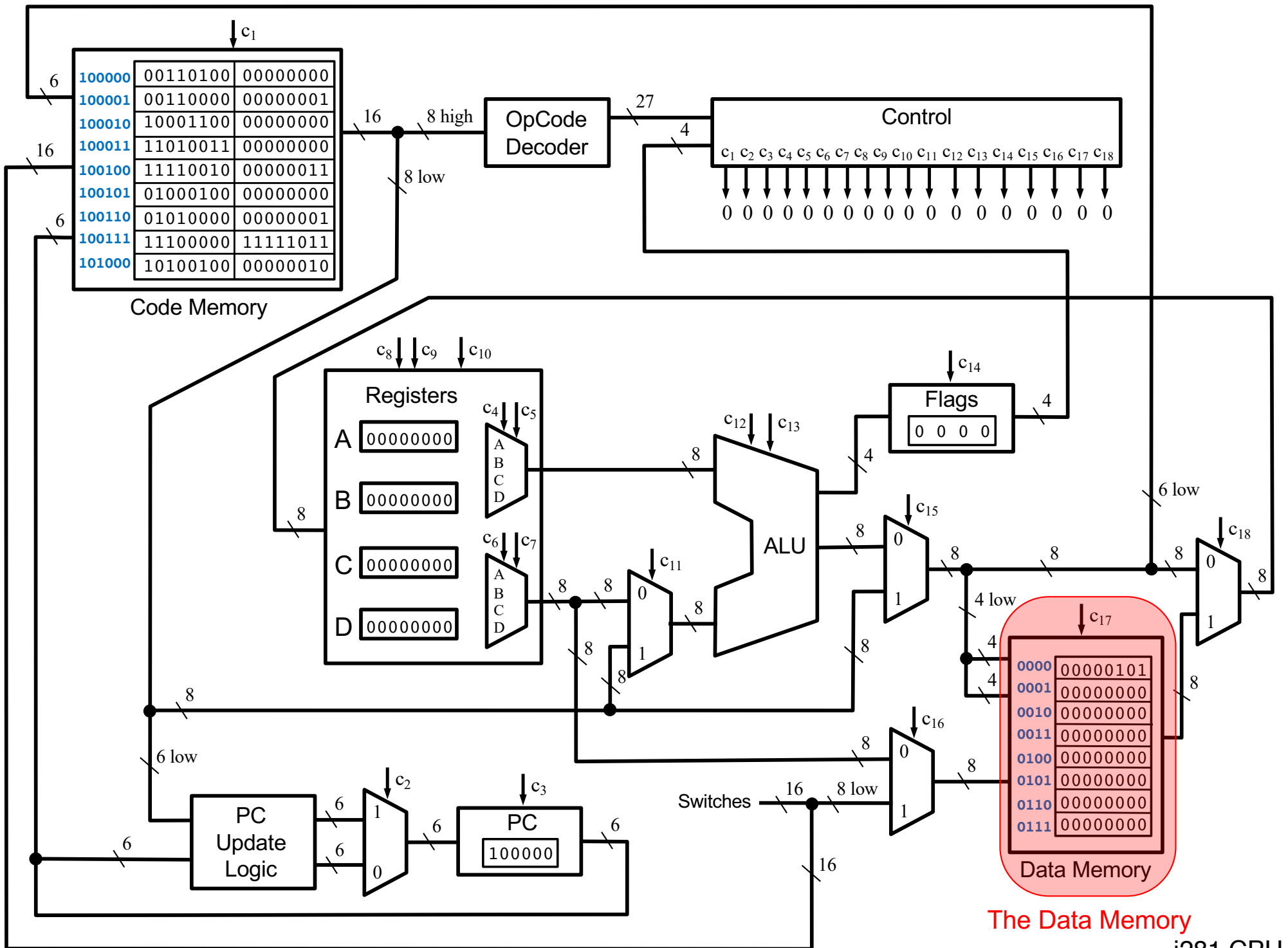




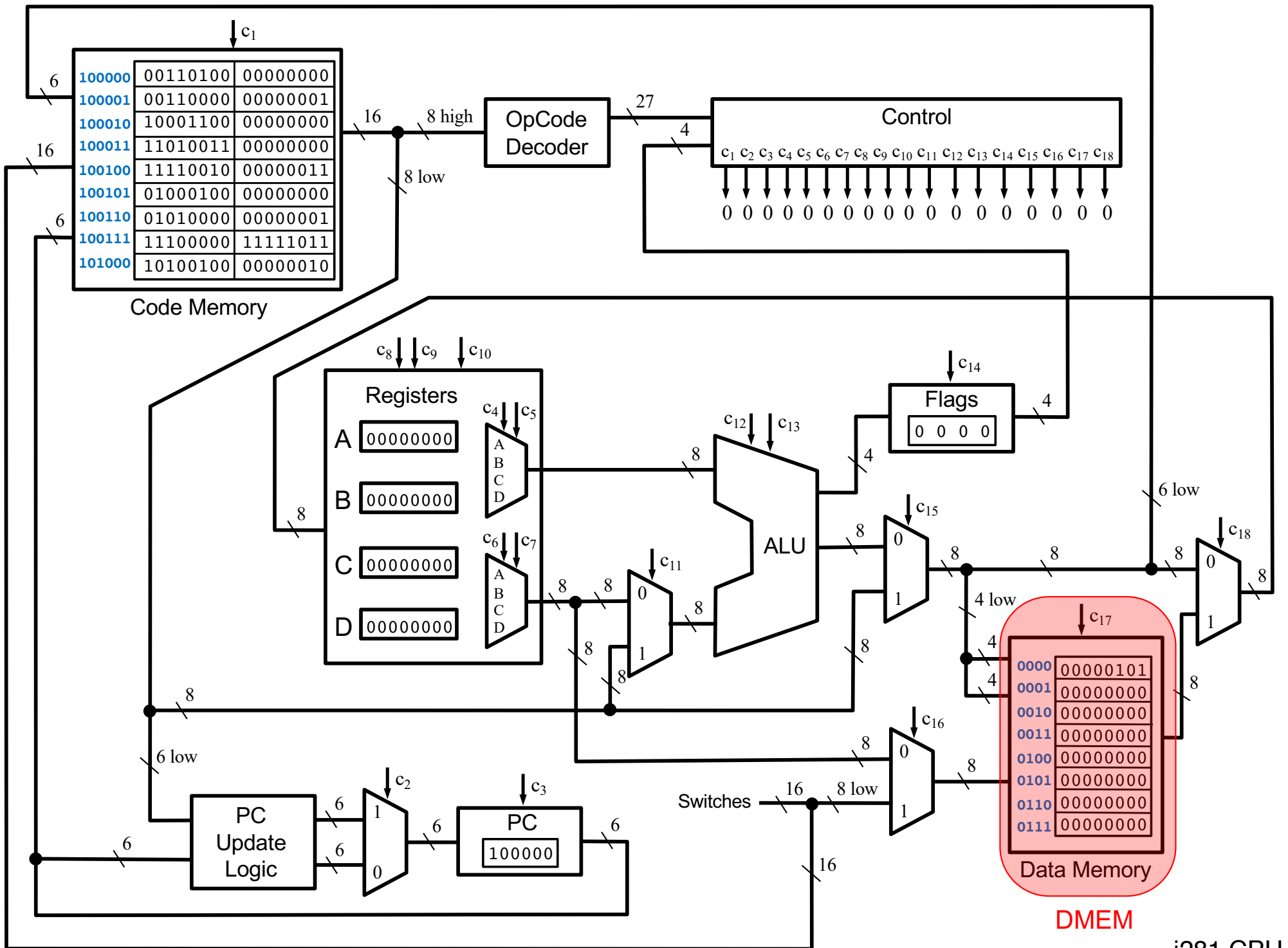








The Data Memory

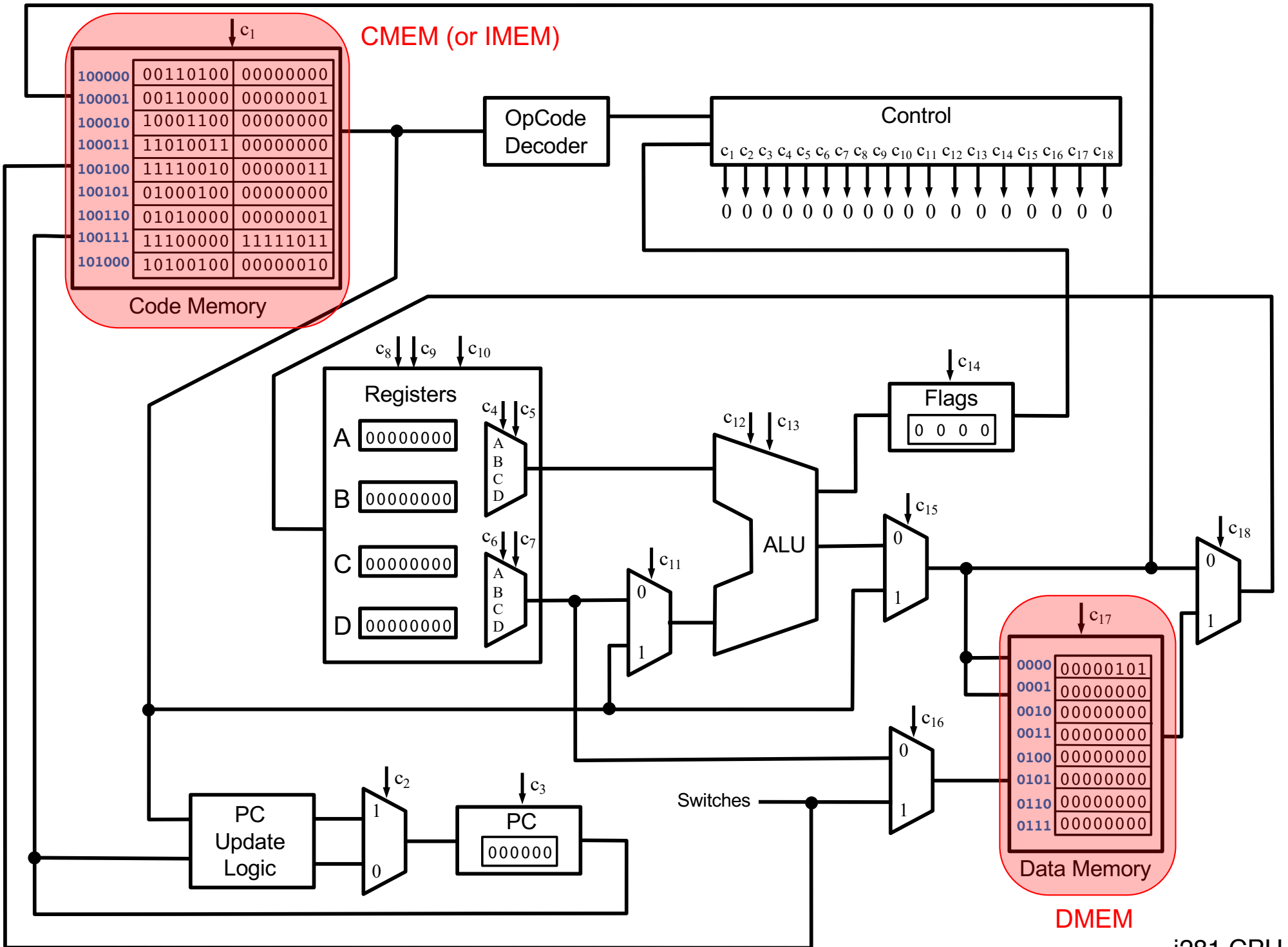


i281 CPU

# **The Memory Layout**

# Memory Layout

- **The i281 CPU uses two different memories**



# Memory Layout

- **The i281 CPU uses two different memories**
- **Data Memory**
  - 16 x 8 bits
- **Code Memory**
  - 64 x 16 bits

# Memory Layout

- The i281 CPU uses two different memories
- Data Memory
  - 16 x 8 bits
- Code Memory
  - 64 x 16 bits
- Note that they have different number of bits

# Memory Layout

- The i281 CPU uses two different memories
- Data Memory
  - 16 x 8 bits ( only 16 bytes! )
- Code Memory
  - 64 x 16 bits ( only 128 bytes! )
- This is a combined total of 144 bytes!



# Memory Layout

- The i281 CPU uses two different memories
- Data Memory
  - 16 x 8 bits ( only 16 bytes! )
- Code Memory
  - 64 x 16 bits ( only 128 bytes! )
- This is a combined total of 144 bytes!
- Which is enough to represent 48 pixels on this slide!

# Data Memory Layout

- **Organized as one contiguous block**
- **Data Memory**
  - **Random access**
  - **Read/write memory**
  - **16 x 8 bits**
  - **Only 16 bytes!**

# Data Memory Layout

- **Organized as one contiguous block**
- **Data Memory**
  - **Random access**
  - **Read/write memory**
  - **16 x 8 bits**
  - **Only 16 bytes!**
- **Implemented as a register file with 16 registers, each of which is 8-bits wide.**
- **The register file has one read port and one write port. It also has a write enable input.**

# Video Card

- **Memory-mapped video memory**
- **The first 8 bytes of the data memory are connected to the 7-segment displays on the Altera board**
- **Writing to these memory cells automatically lights up the displays, which use only the 4 least significant bits**
- **In video game mode each LED is controlled separately using a different set of 7-segment decoders & all 8 bits**
- **The contents of the second 8 bytes of the data memory cannot be visualized, but programs can still use them**

# Data Memory Contents for the Bubble Sort Program

```
00000111
00000011
00000010
00000001
00000110
00000100
00000101
00001000
00000111
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

# Data Memory Contents for the Bubble Sort Program

Address	Data
0000	00000111
0001	00000011
0010	00000010
0011	00000001
0100	00000110
0101	00000100
0110	00000101
0111	00001000
1000	00000111
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

# Data Memory Contents for the Bubble Sort Program

Address	Data	Comment
0000	00000111	//array[0]
0001	00000011	//array[1]
0010	00000010	//array[2]
0011	00000001	//array[3]
0100	00000110	//array[4]
0101	00000100	//array[5]
0110	00000101	//array[6]
0111	00001000	//array[7]
1000	00000111	//last
1001	00000000	//temp
1010	00000000	
1011	00000000	
1100	00000000	
1101	00000000	
1110	00000000	
1111	00000000	

# Memory-Mapped Video Memory

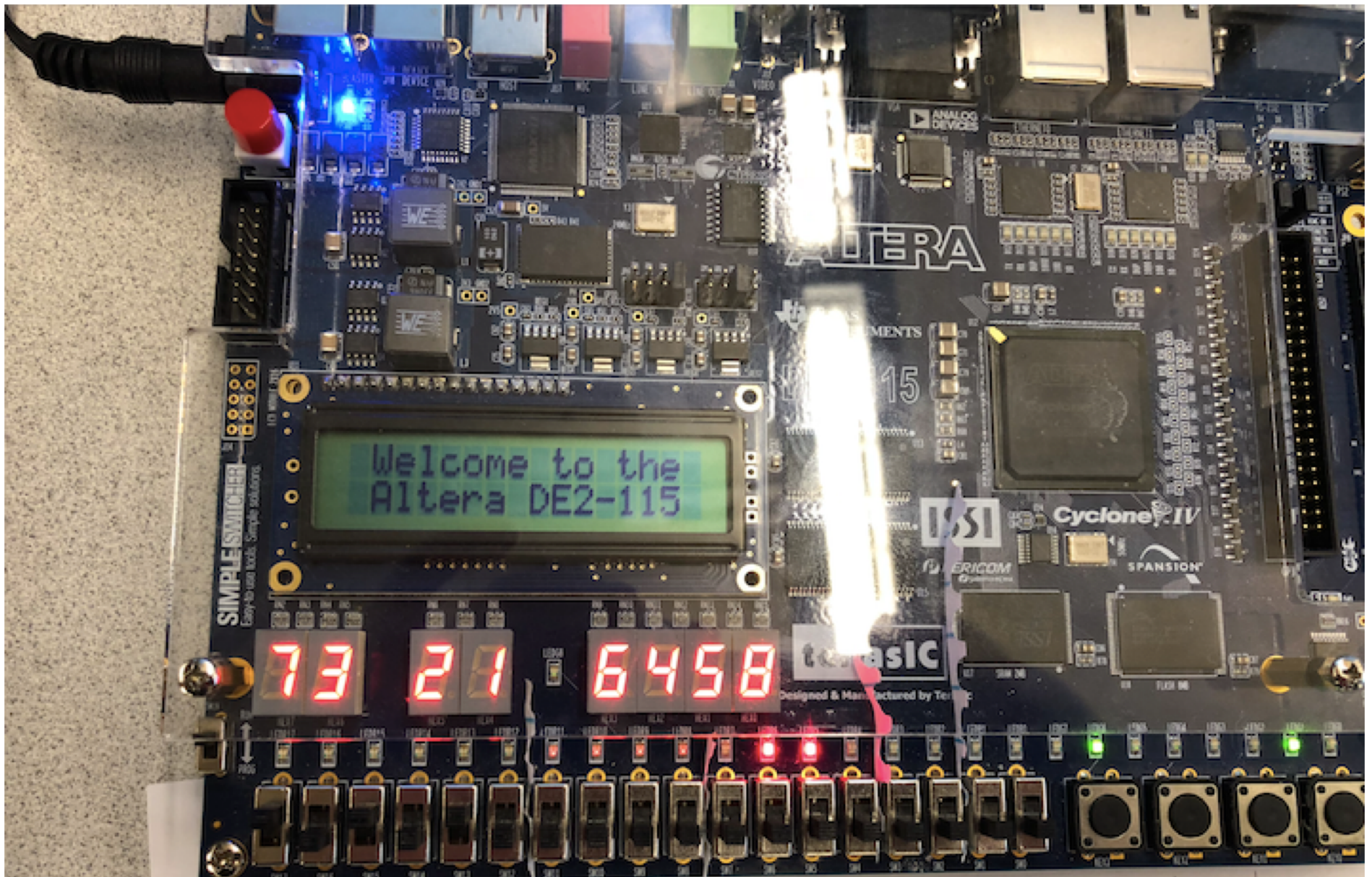
Address	Data
0000	00000111
0001	00000011
0010	00000010
0011	00000001
0100	00000110
0101	00000100
0110	00000101
0111	00001000
1000	00000111
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000



# Memory-Mapped Video Memory

Address	Data
0000	00000111 → 7
0001	00000011 → 3
0010	00000010 → 2
0011	00000001 → 1
0100	00000110 → 6
0101	00000100 → 4
0110	00000101 → 5
0111	00001000 → 8
1000	00000111
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

# Memory-Mapped Video Memory



# Memory-Mapped Video Memory

Address	Data
0000	00000111 → 3
0001	00000011 → 3
0010	00000010 → 2
0011	00000001 → 1
0100	00000110 → 6
0101	00000100 → 4
0110	01100101 → 5
0111	00001000 → 8
1000	00000111
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

Changing these bits will not affect what is displayed, but it will affect the program.

# Memory-Mapped Video Memory

Address	Data
0000	0 <b>1</b> 000111
0001	000 <b>1</b> 0011
0010	<b>1</b> 0000010
0011	0 <b>1</b> 000001
0100	000 <b>1</b> 0110
0101	<b>1</b> 0000100
0110	0 <b>11</b> 00101
0111	0 <b>1</b> 001000
1000	00000111
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

Changing these bits will not affect what is displayed, but it will affect the program.

# Memory-Mapped Video Memory

Address	Data	
0000	00000111	→ 1
0001	00000011	→ 3
0010	00000010	→ 2
0011	00000001	→ 1
0100	00000110	→ 3
0101	00000100	→ 4
0110	00000101	→ 5
0111	00001000	→ 8
1000	00000111	
1001	00000000	
1010	00000000	
1011	00000000	
1100	00000000	
1101	00000000	
1110	00000000	
1111	00000000	

This memory cell is used by the program, but it cannot be visualized on the 7-segment displays.

# Memory-Mapped Video Memory

Address	Data	
0000	00000111	→ 1
0001	00000011	→ 3
0010	00000010	→ 2
0011	00000001	→ 1
0100	00000110	→ 3
0101	00000100	→ 4
0110	00000101	→ 5
0111	00001000	→ 8
1000	00000111	
1001	00000000	
1010	00000000	
1011	00000000	
1100	00000000	
1101	00000000	
1110	00000000	
1111	00000000	









All of these cannot be visualized on the 7-segment displays.

# Memory-Mapped Video Memory in Video Game Mode









Address	Data
0000	00000000
0001	00000000
0010	00000000
0011	00000000
0100	01111001
0101	01010100
0110	01011110
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

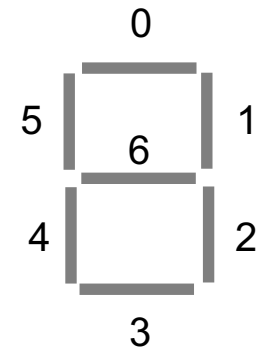


# Memory-Mapped Video Memory in Video Game Mode

Address	Data
0000	00000000 → 
0001	00000000 → 
0010	00000000 → 
0011	00000000 → 
0100	01111001 → 
0101	01010100 → 
0110	01011110 → 
0111	00000000 → 
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

# Memory-Mapped Video Memory in Video Game Mode

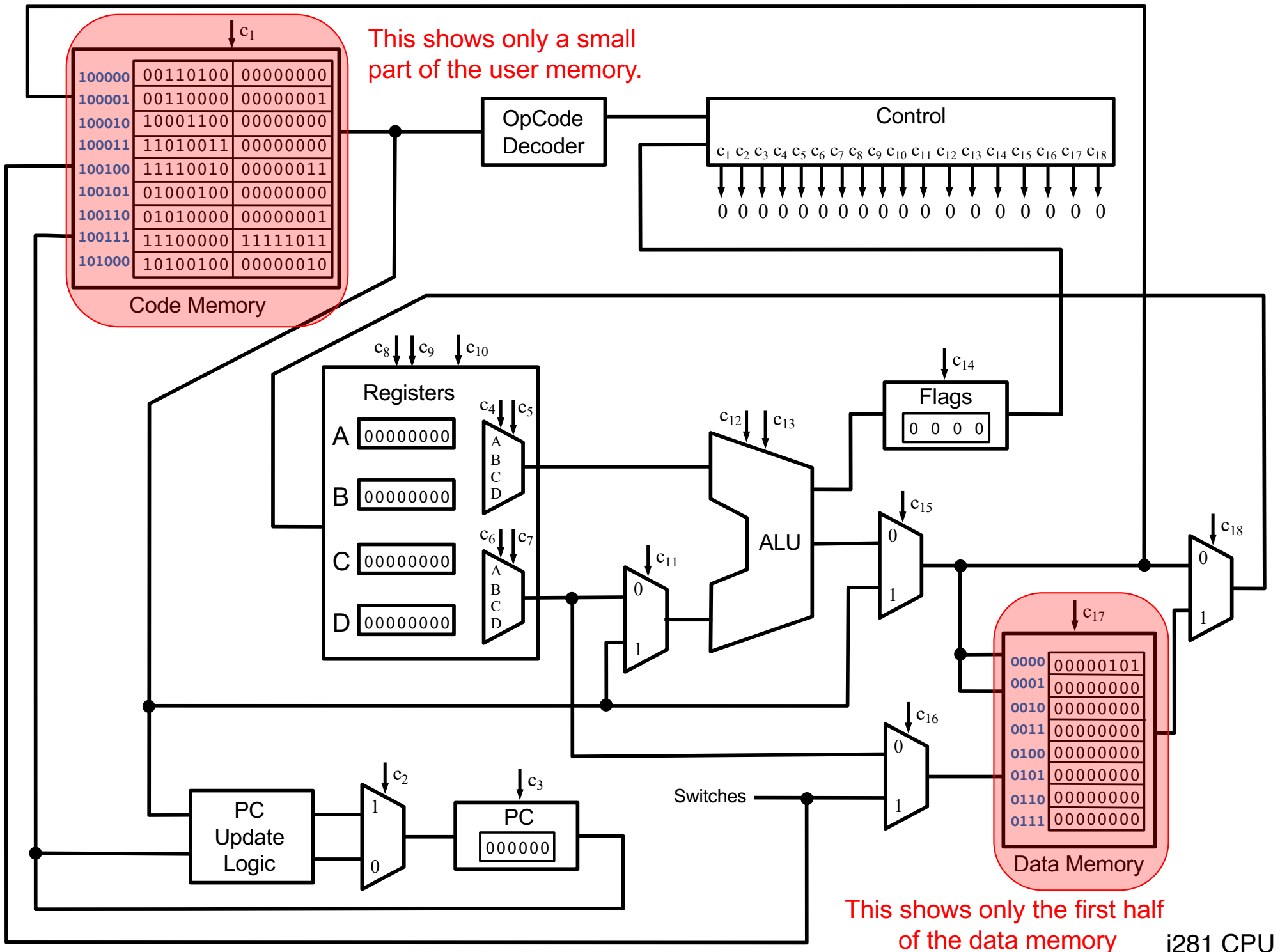
Address	Data
0000	00000000 → 
0001	00000000 → 
0010	00000000 → 
0011	00000000 → 
0100	01111001 → 
0101	01010100 → 
0110	01011110 → 
0111	00000000 → 
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

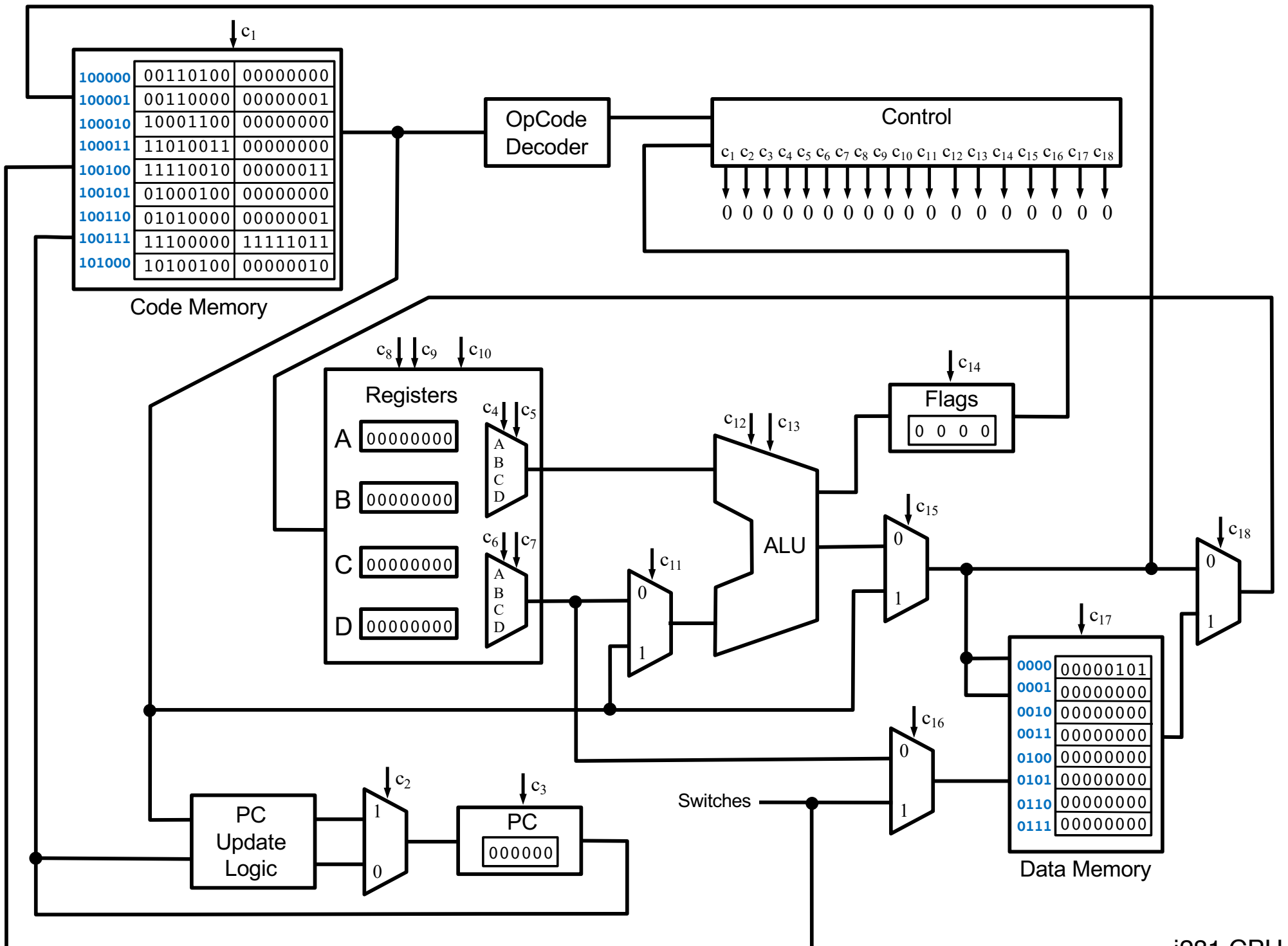


In this case the last 7 bits are used and each controls one of the 7 LEDs.

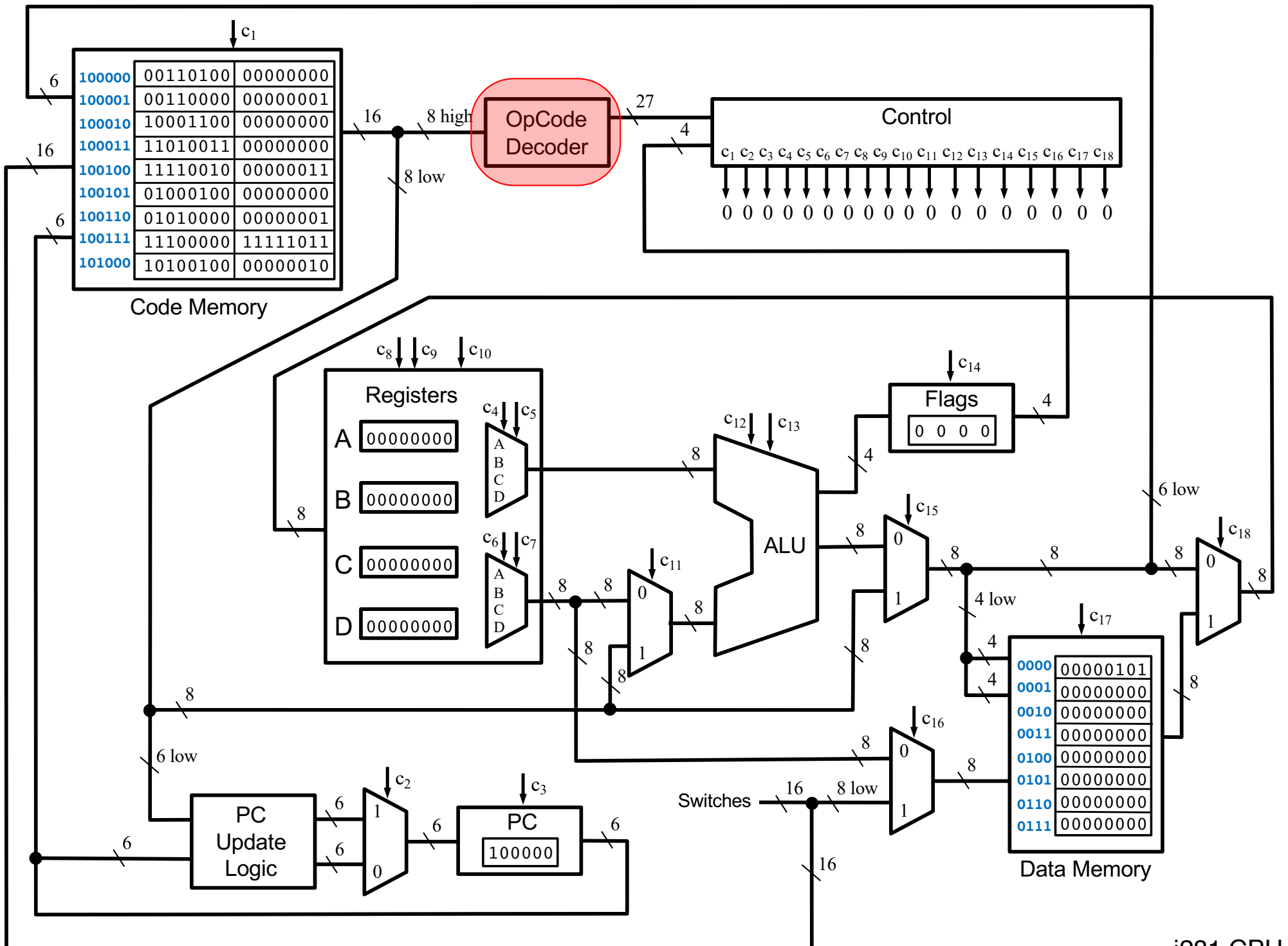
# Code Memory Layout

- **Split into two parts**
- **BIOS Code Memory (addresses 0 to 31)**
  - Read only memory
  - 32 x 16 bits
- **User Code Memory (addresses 32 to 63)**
  - Read only memory (in User mode)
  - Read/Write memory (in BIOS mode)
  - 32 x 16 bits





# **The OPPOSITE Decoder**





# The i281 Assembly Instructions

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 i281 Assembly Instructions

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

# There are only 26 OPCODEs

NOOP

INPUTC

INPUTCF

INPUTD

INPUTDF

MOVE

LOADI

LOADP

ADD

ADDI

SUB

SUBI

LOAD

LOADF

STORE

STOREF

SHIFTL

SHIFTR

CMP

JUMP

BRE

BRZ

BRNE

BRNZ

BRG

BRGE

# There are only 26 OPCODEs

NOOP	ADD	SHIFTL
INPUTC	ADDI	SHIFTR
INPUTCF	SUB	CMP
INPUTD	SUBI	JUMP
INPUTDF	LOAD	BRE
MOVE	LOADF	BRZ
LOADI	STORE	BRNE
LOADP	STOREF	BRNZ
		BRG
		BRGE

All of these are available in the assembly language for this processor.  
However, three pairs are aliased at the machine language level.

# There are only <sup>25</sup>~~26~~ OPCODEs

NOOP

INPUTC

INPUTCF

INPUTD

INPUTDF

MOVE

LOADI

LOADP

these two  
are aliased

ADD

ADDI

SUB

SUBI

LOAD

LOADF

STORE

STOREF

SHIFTL

SHIFTR

CMP

JUMP

BRE

BRZ

BRNE

BRNZ

BRG

BRGE

They have a different meaning in the assembly language, but the assembler maps them to the same machine language OPCODE.

# There are only <sup>25</sup>~~26~~ OPCODEs

NOOP

INPUTC

INPUTCF

INPUTD

INPUTDF

MOVE

LOADI/LOADP

ADD

ADDI

SUB

SUBI

LOAD

LOADF

STORE

STOREF

SHIFTL

SHIFTR

CMP

JUMP

BRE

BRZ

BRNE

BRNZ

BRG

BRGE

# There are only <sup>24</sup>~~26~~ OPCODES

NOOP  
INPUTC  
INPUTCF  
INPUTD  
INPUTDF  
MOVE  
LOADI/LOADP

ADD  
ADDI  
SUB  
SUBI  
LOAD  
LOADF  
STORE  
STOREF

SHIFTL  
SHIFTR  
CMP  
JUMP  
BRE  
BRZ  
BRNE  
BRNZ  
BRG  
BRGE

these two  
are aliased



# There are only <sup>23</sup>~~26~~ OPCODES

NOOP  
INPUTC  
INPUTCF  
INPUTD  
INPUTDF  
MOVE  
LOADI/LOADP

ADD  
ADDI  
SUB  
SUBI  
LOAD  
LOADF  
STORE  
STOREF

SHIFTL  
SHIFTR  
CMP  
JUMP  
BRE  
BRZ  
BRNE  
BRNZ  
BRG  
BRGE

these two  
are aliased

these two  
are aliased

# There are only 23 OPCODEs

NOOP

INPUTC

INPUTCF

INPUTD

INPUTDF

MOVE

LOADI/LOADP

ADD

ADDI

SUB

SUBI

LOAD

LOADF

STORE

STOREF

SHIFTL

SHIFTR

CMP

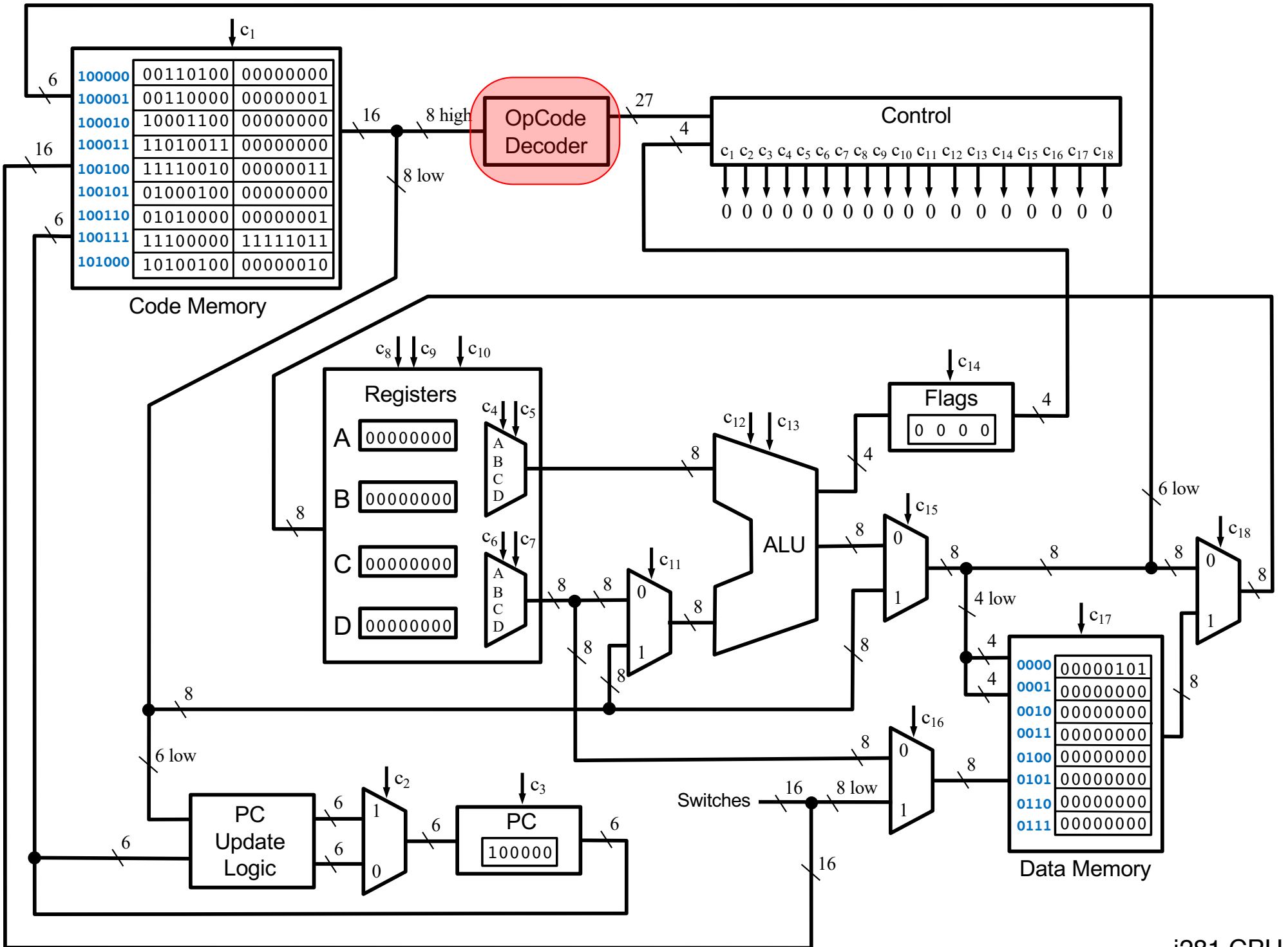
JUMP

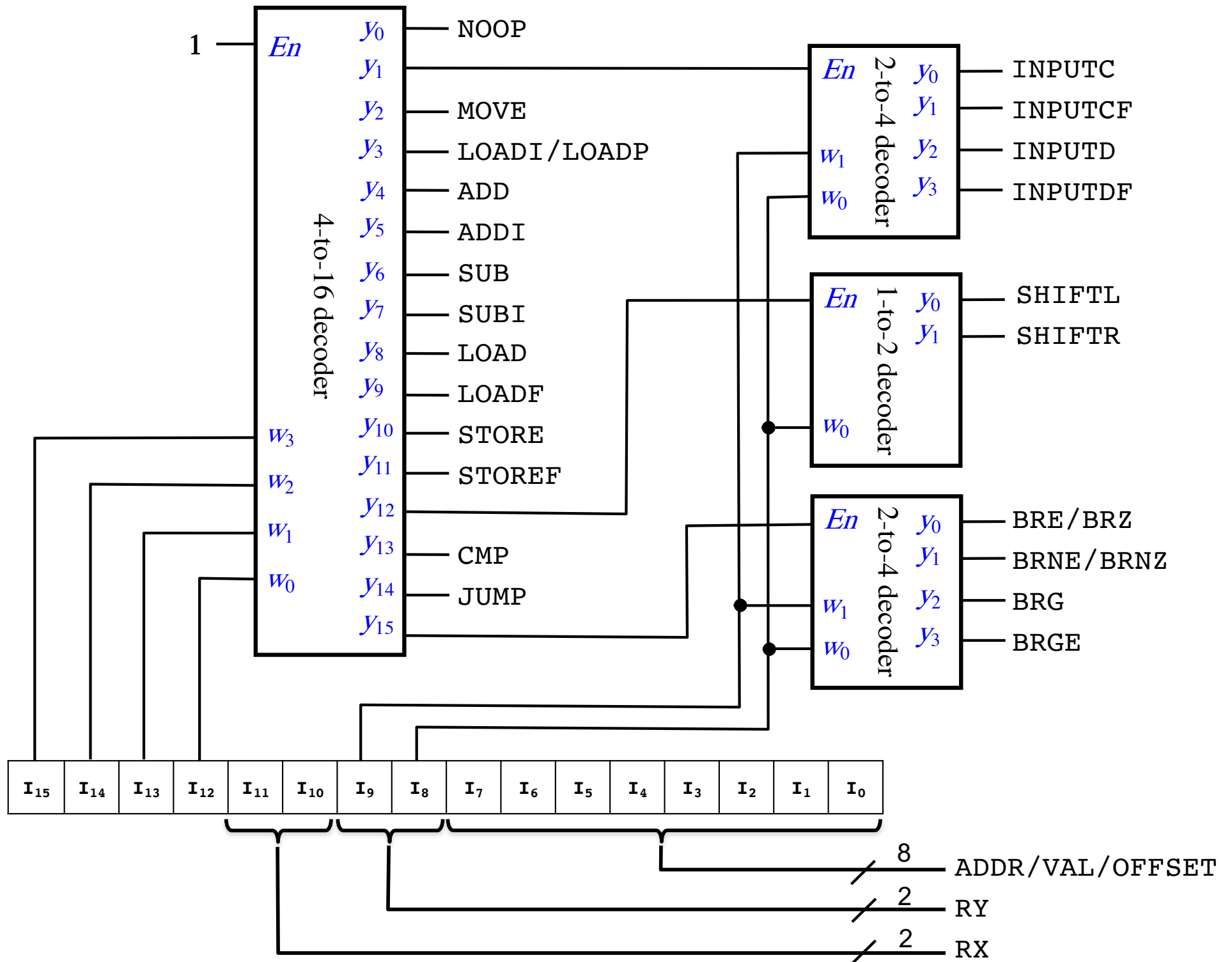
BRE/BRZ

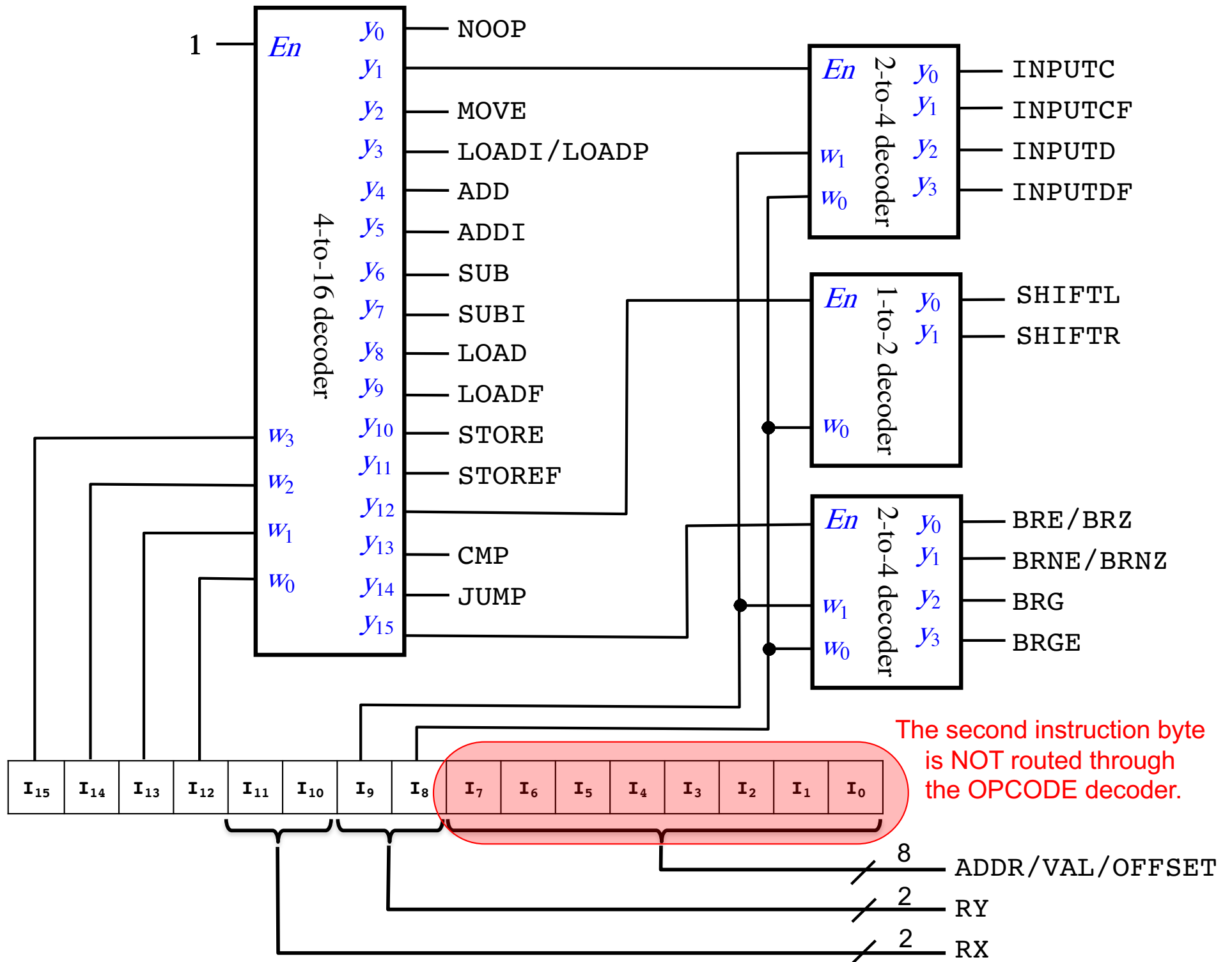
BRNE/BRNZ

BRG

BRGE

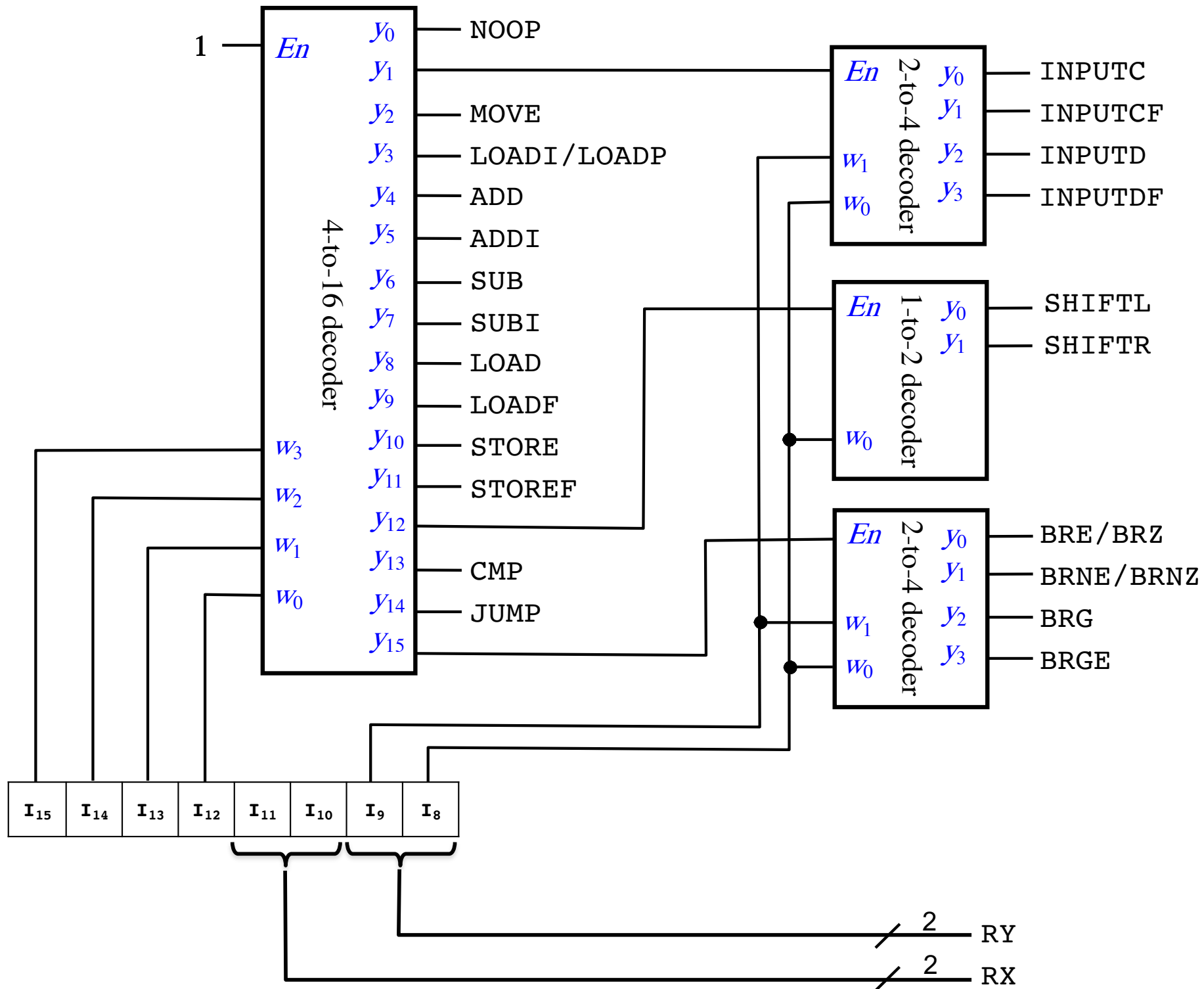


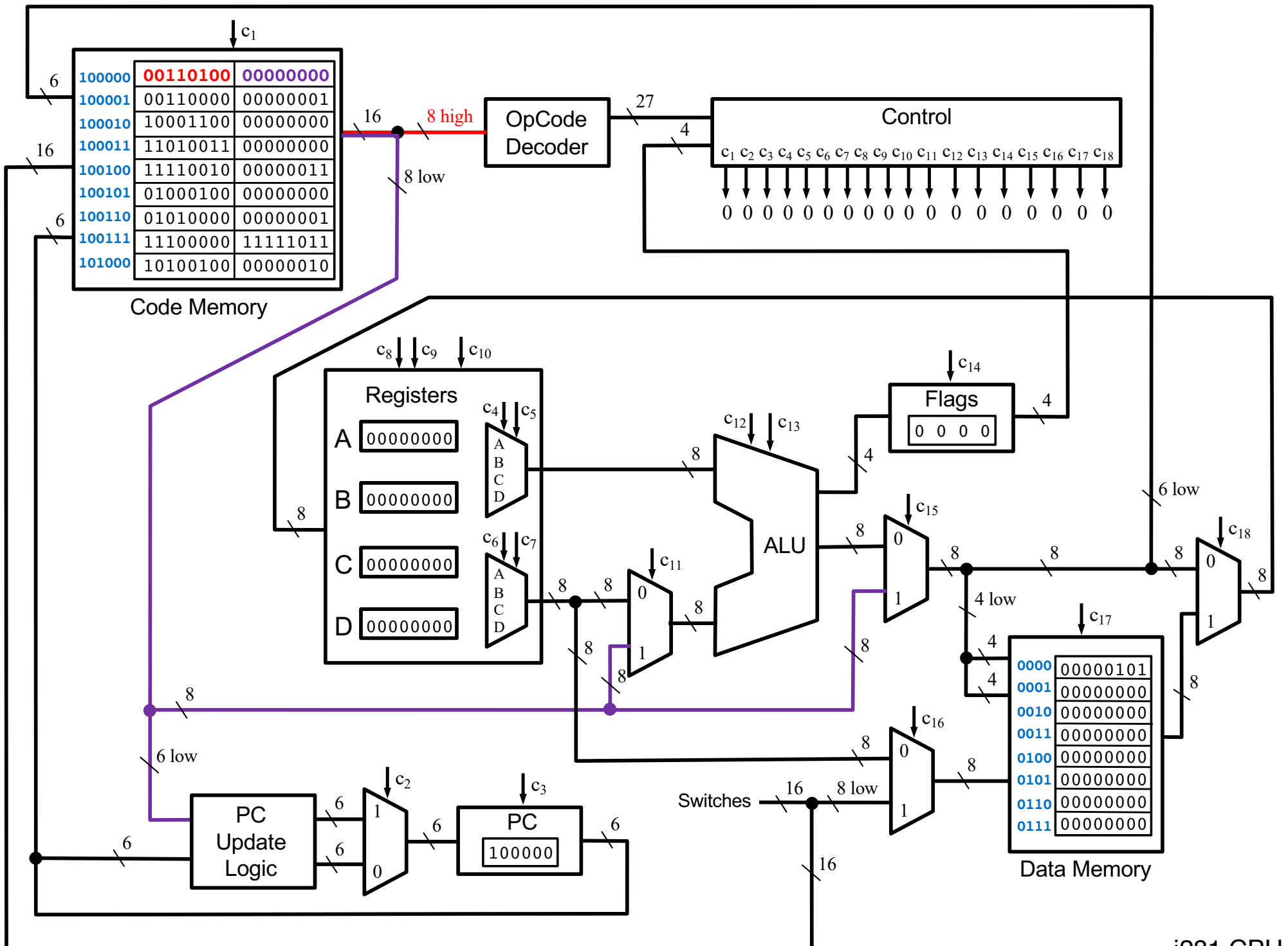




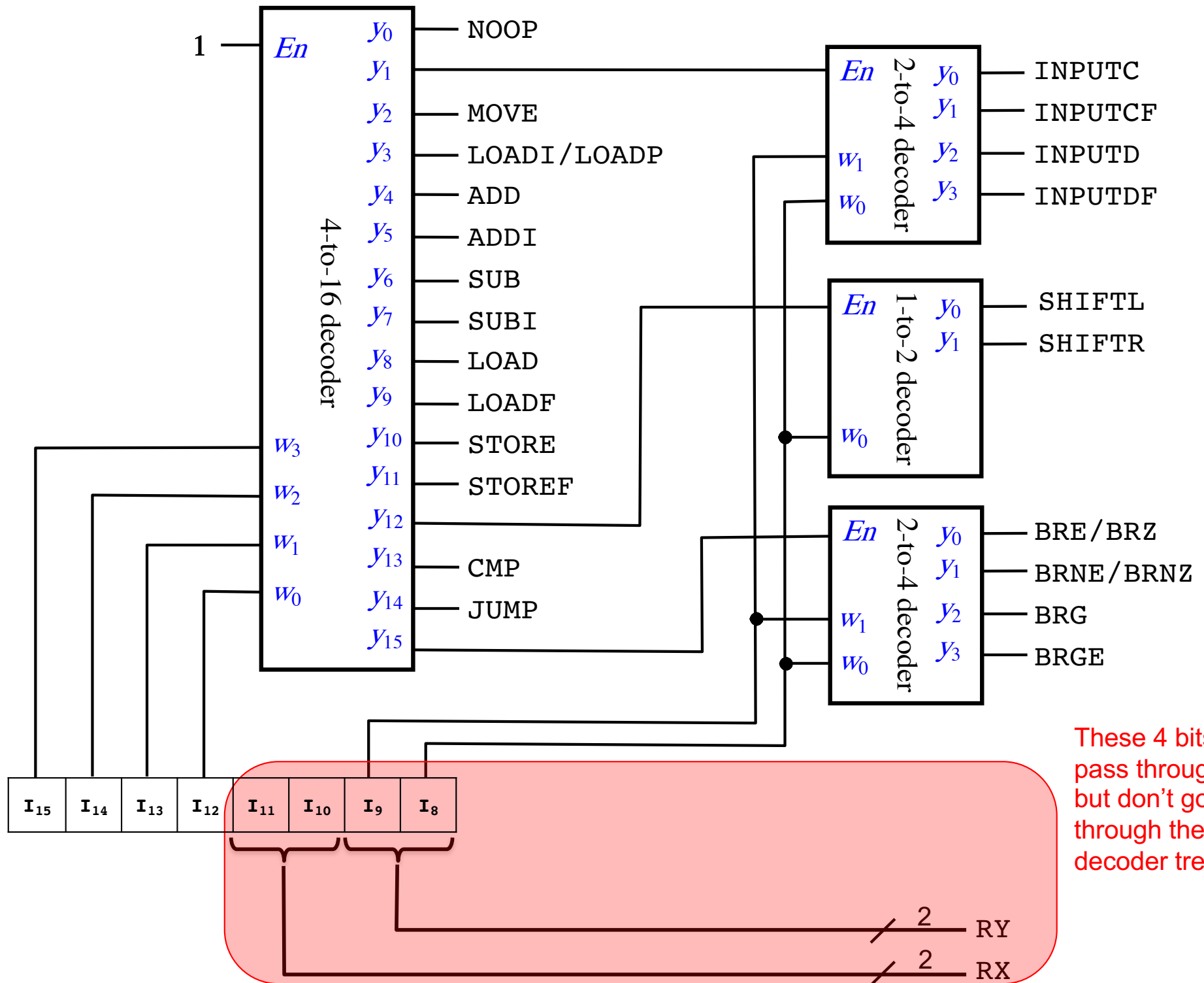
The second instruction byte is NOT routed through the OPCODE decoder.

8 ADDR/VAL/OFFSET  
 2 RY  
 2 RX

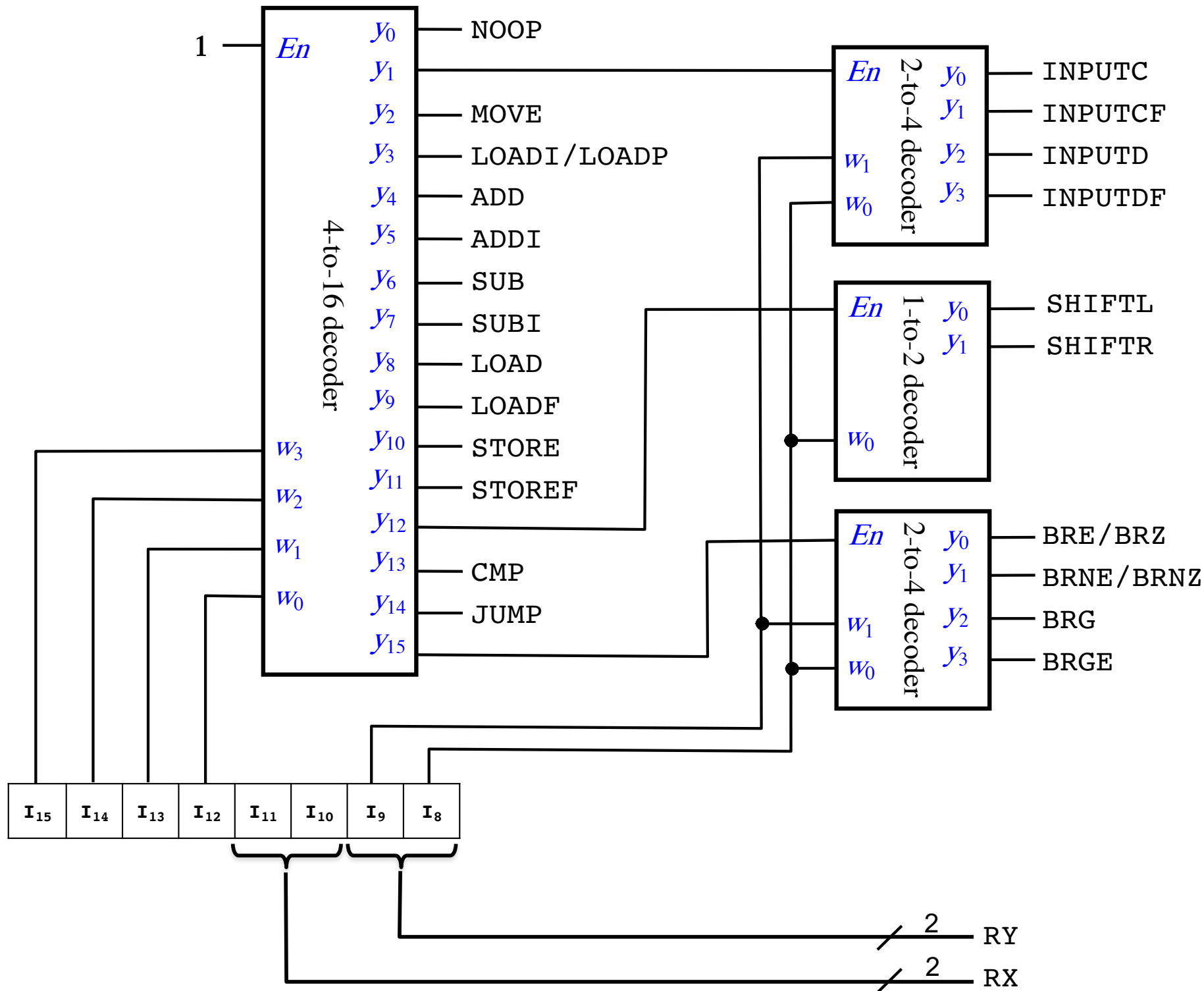


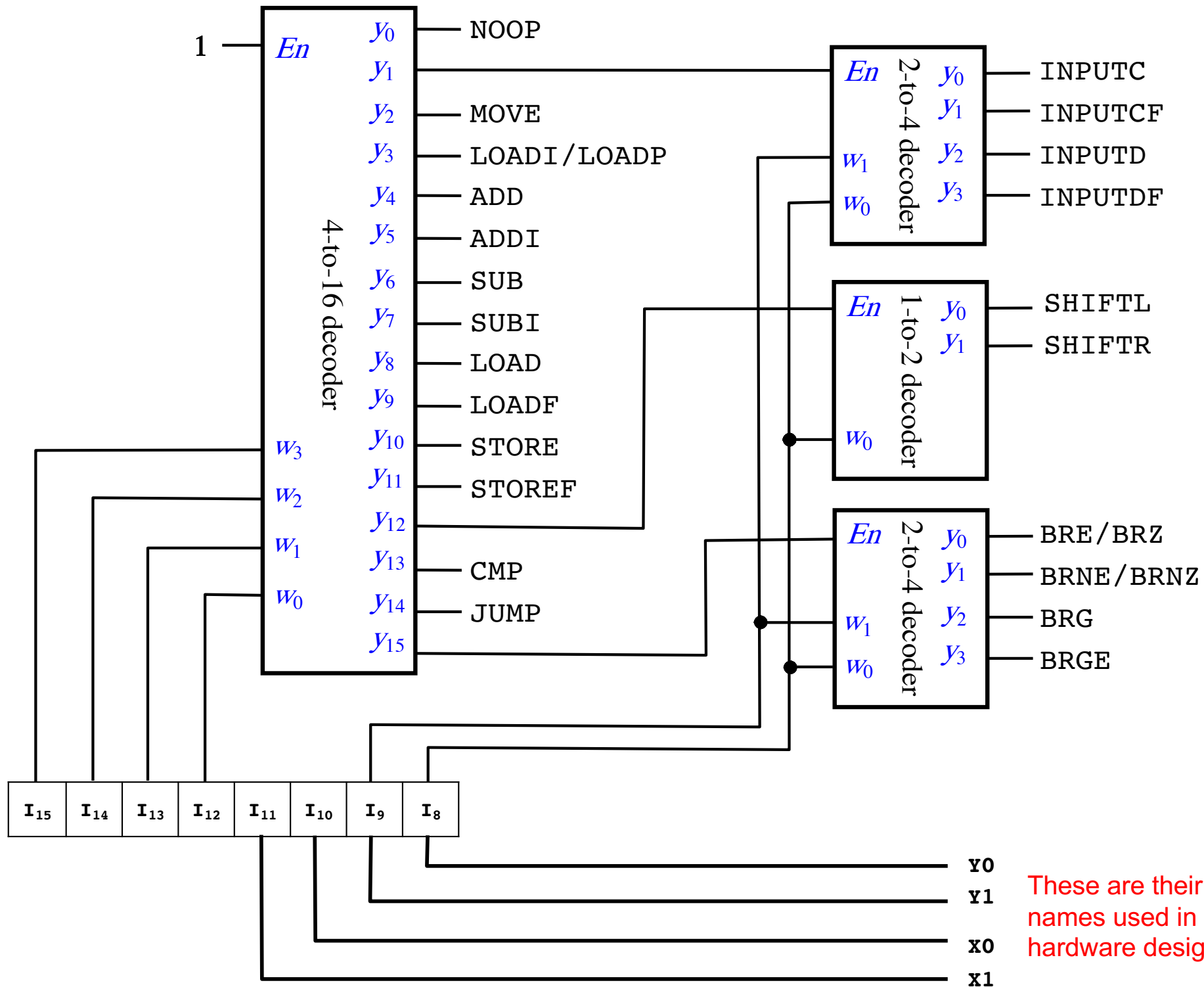






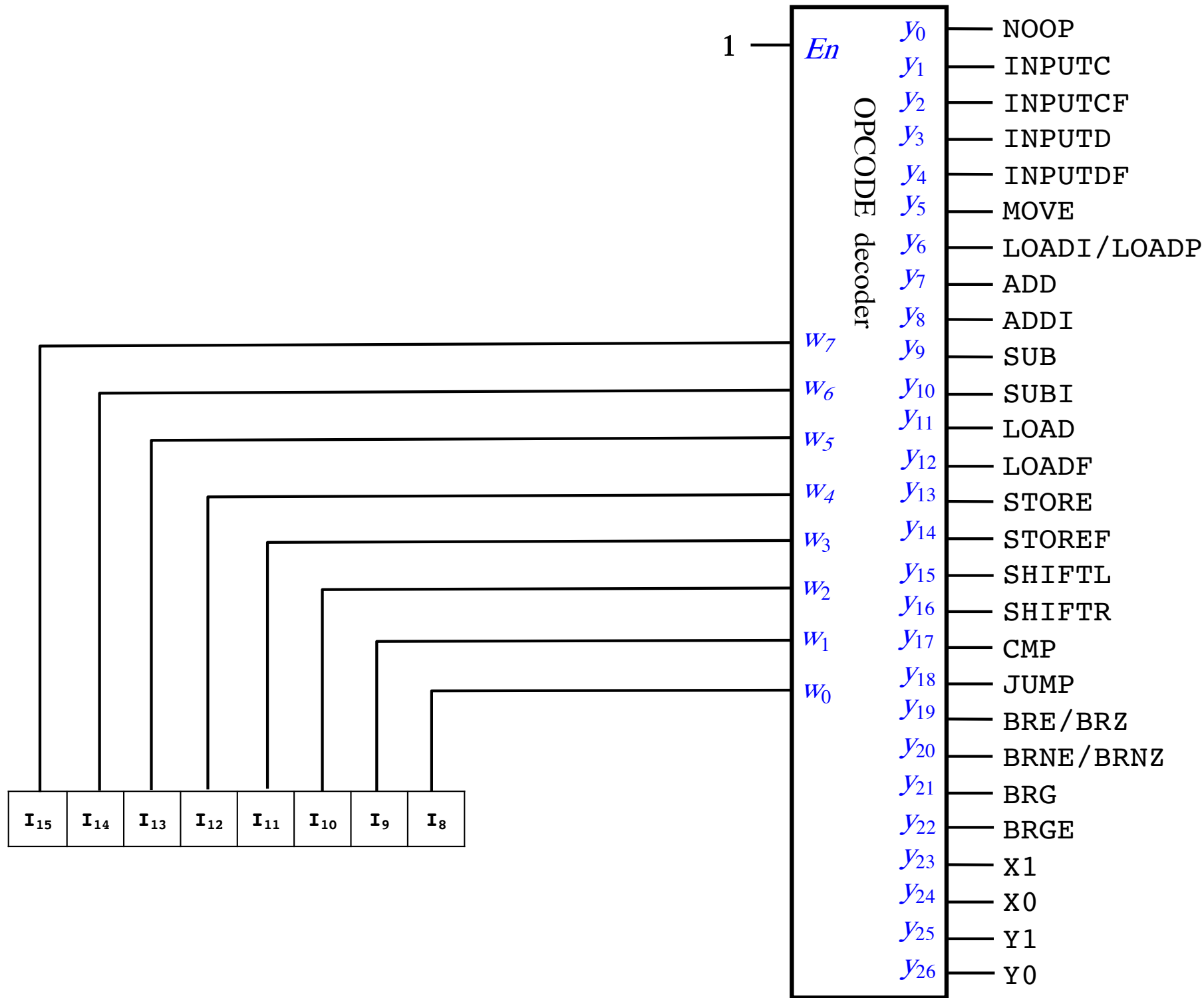
These 4 bits pass through, but don't go through the decoder tree.

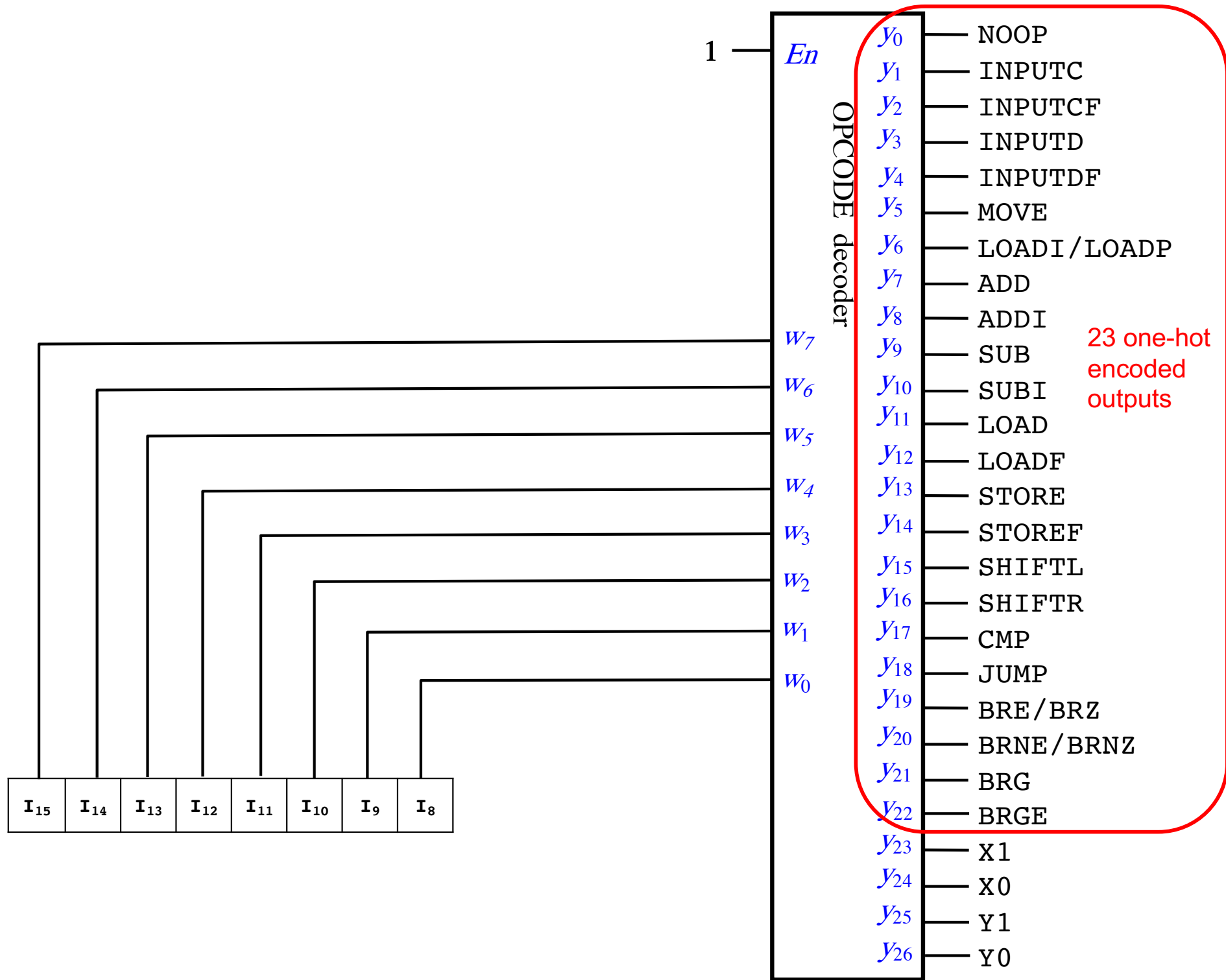


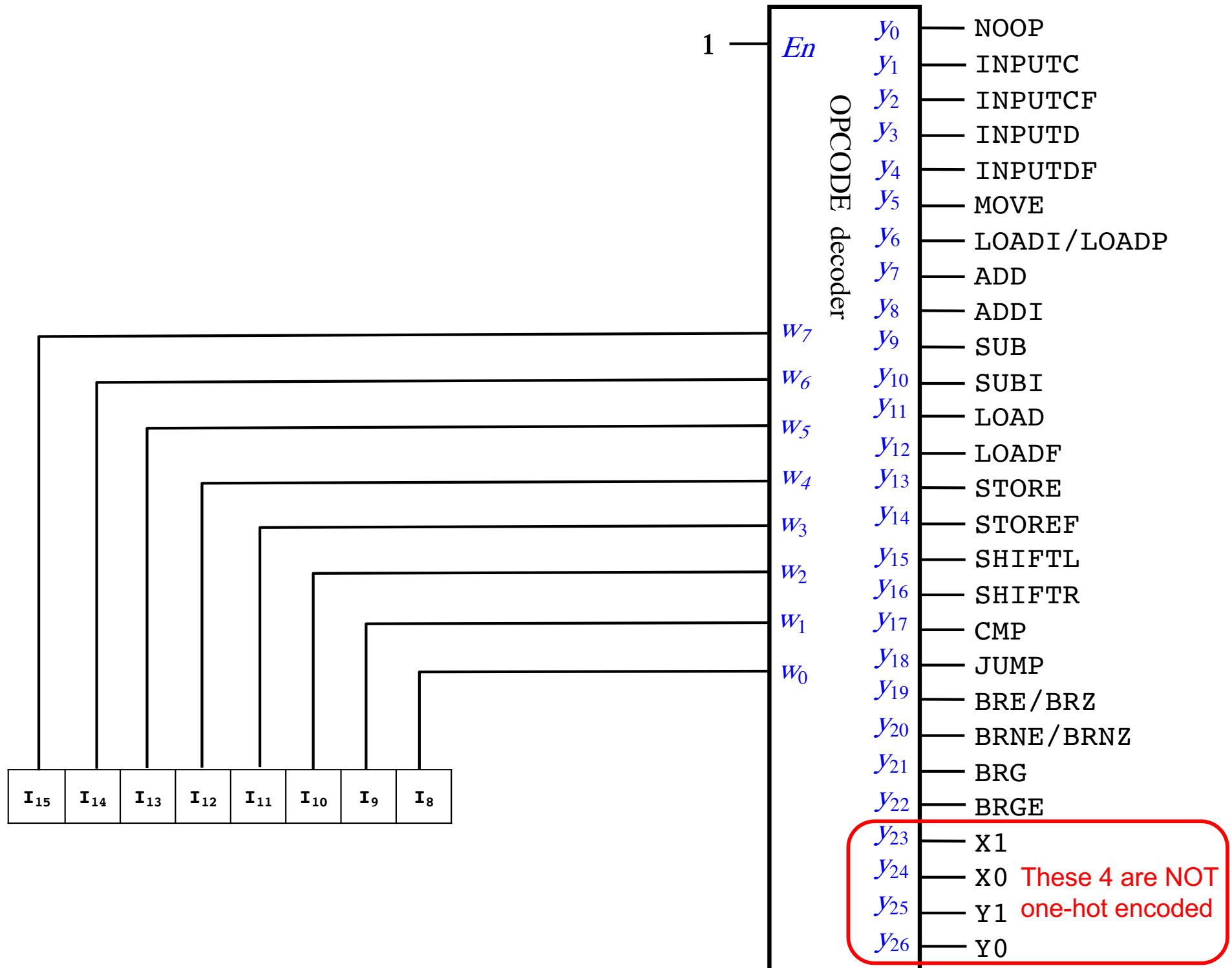


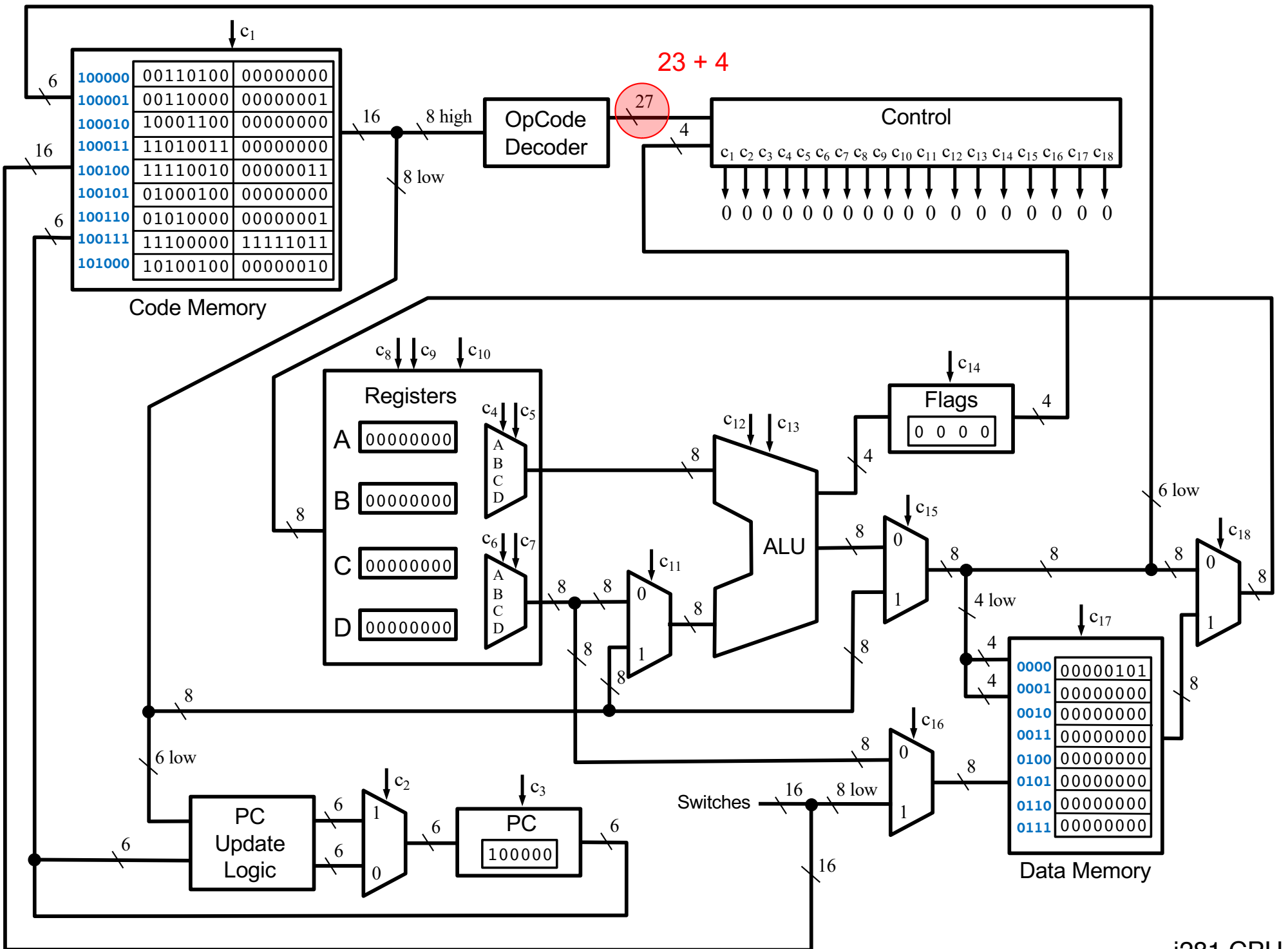
These are their names used in the hardware design.

<b>I<sub>15</sub></b>	<b>I<sub>14</sub></b>	<b>I<sub>13</sub></b>	<b>I<sub>12</sub></b>	<b>I<sub>11</sub></b>	<b>I<sub>10</sub></b>	<b>I<sub>9</sub></b>	<b>I<sub>8</sub></b>
-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	----------------------	----------------------



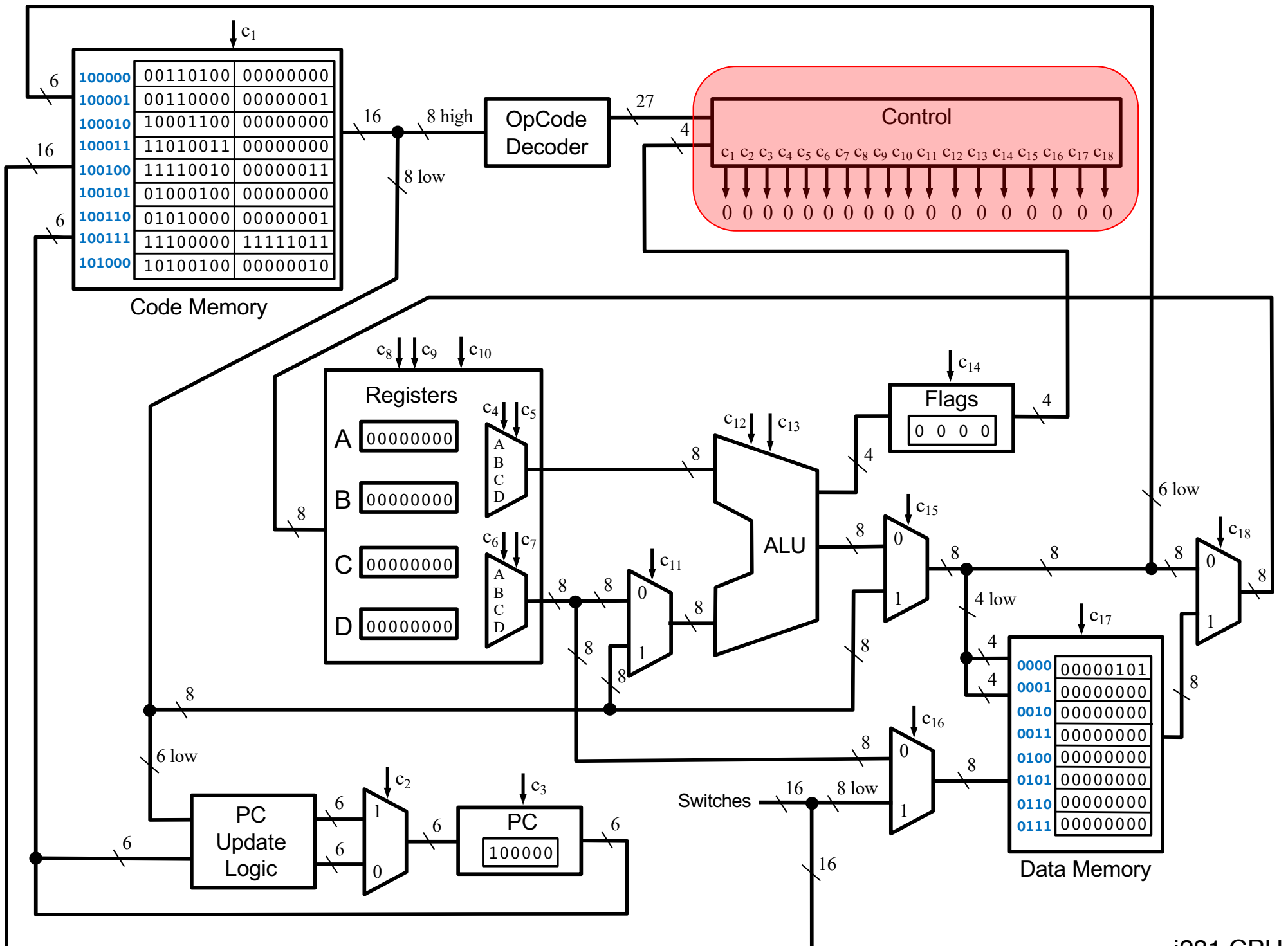








# **The Control Logic**













	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															

Taken from these bits of the instruction

I <sub>15</sub>	I <sub>14</sub>	I <sub>13</sub>	I <sub>12</sub>	I <sub>11</sub>	I <sub>10</sub>	I <sub>9</sub>	I <sub>8</sub>	I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>
						Y <sub>1</sub>	Y <sub>0</sub>								



	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															

Taken from these bits of the instruction

I <sub>15</sub>	I <sub>14</sub>	I <sub>13</sub>	I <sub>12</sub>	I <sub>11</sub>	I <sub>10</sub>	I <sub>9</sub>	I <sub>8</sub>	I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>
						Y <sub>1</sub>	Y <sub>0</sub>								



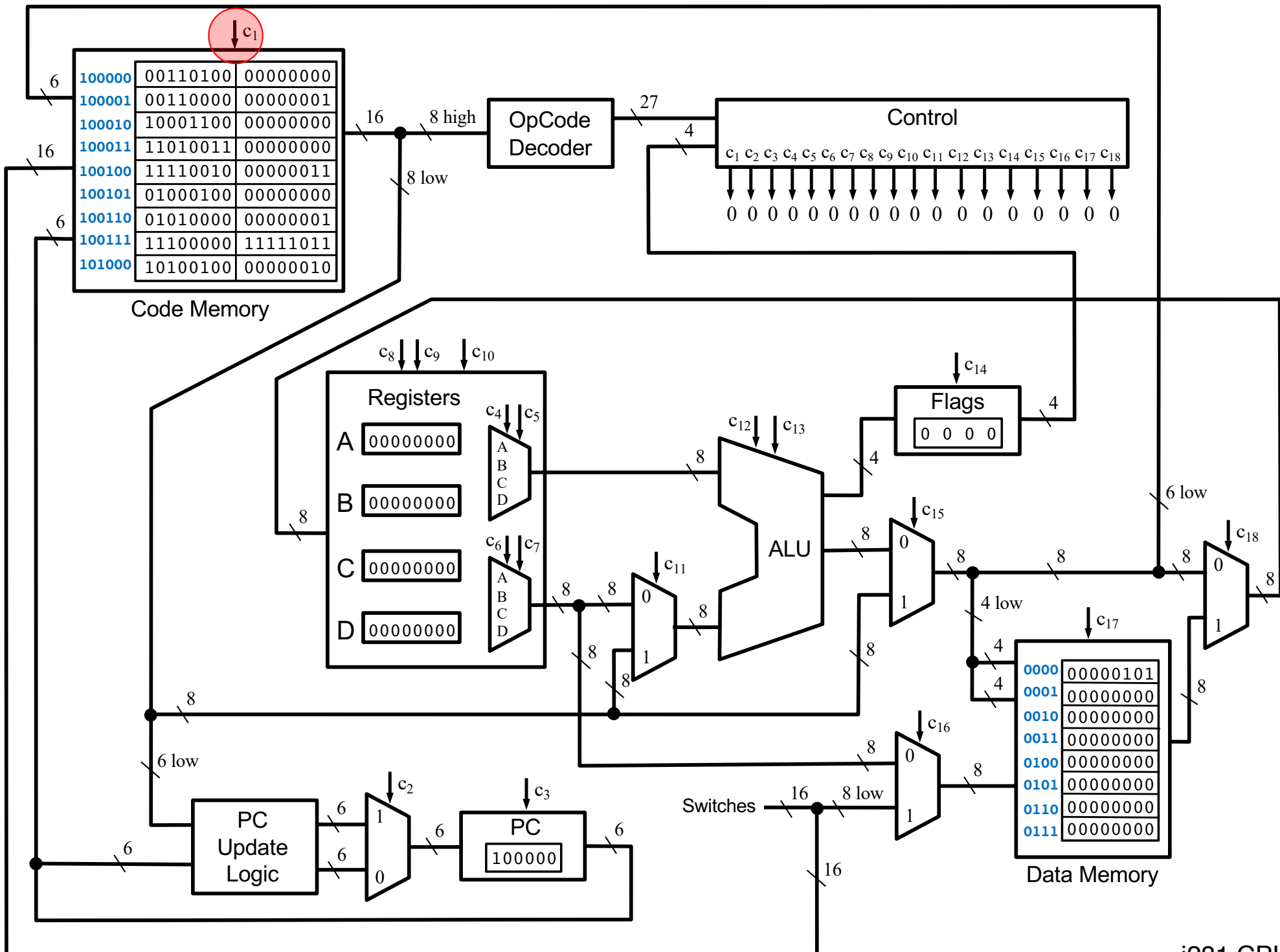
	C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>	C <sub>5</sub>	C <sub>6</sub>	C <sub>7</sub>	C <sub>8</sub>	C <sub>9</sub>	C <sub>10</sub>	C <sub>11</sub>	C <sub>12</sub>	C <sub>13</sub>	C <sub>14</sub>	C <sub>15</sub>	C <sub>16</sub>	C <sub>17</sub>	C <sub>18</sub>
	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	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															

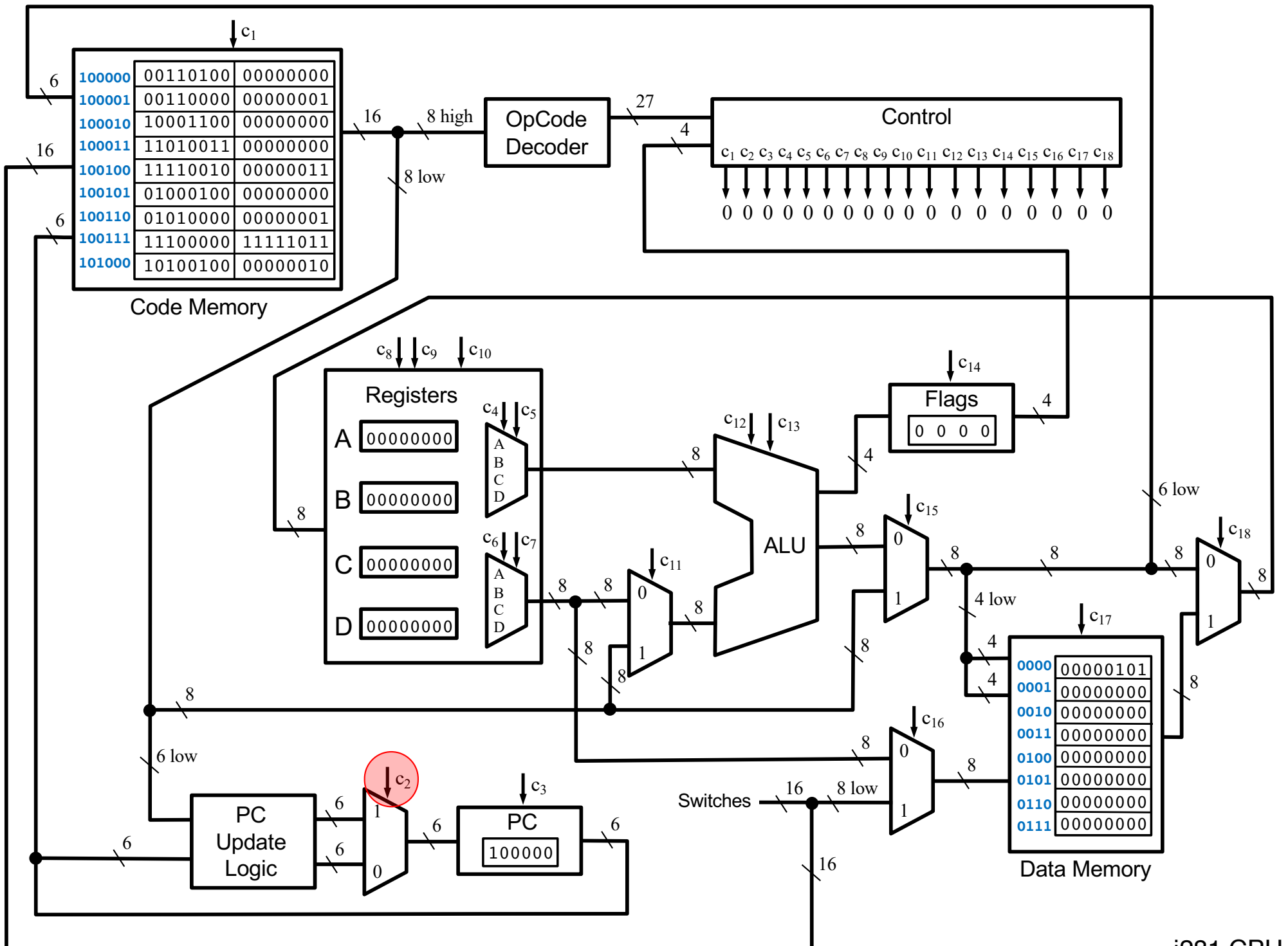
computed using  
the flags register

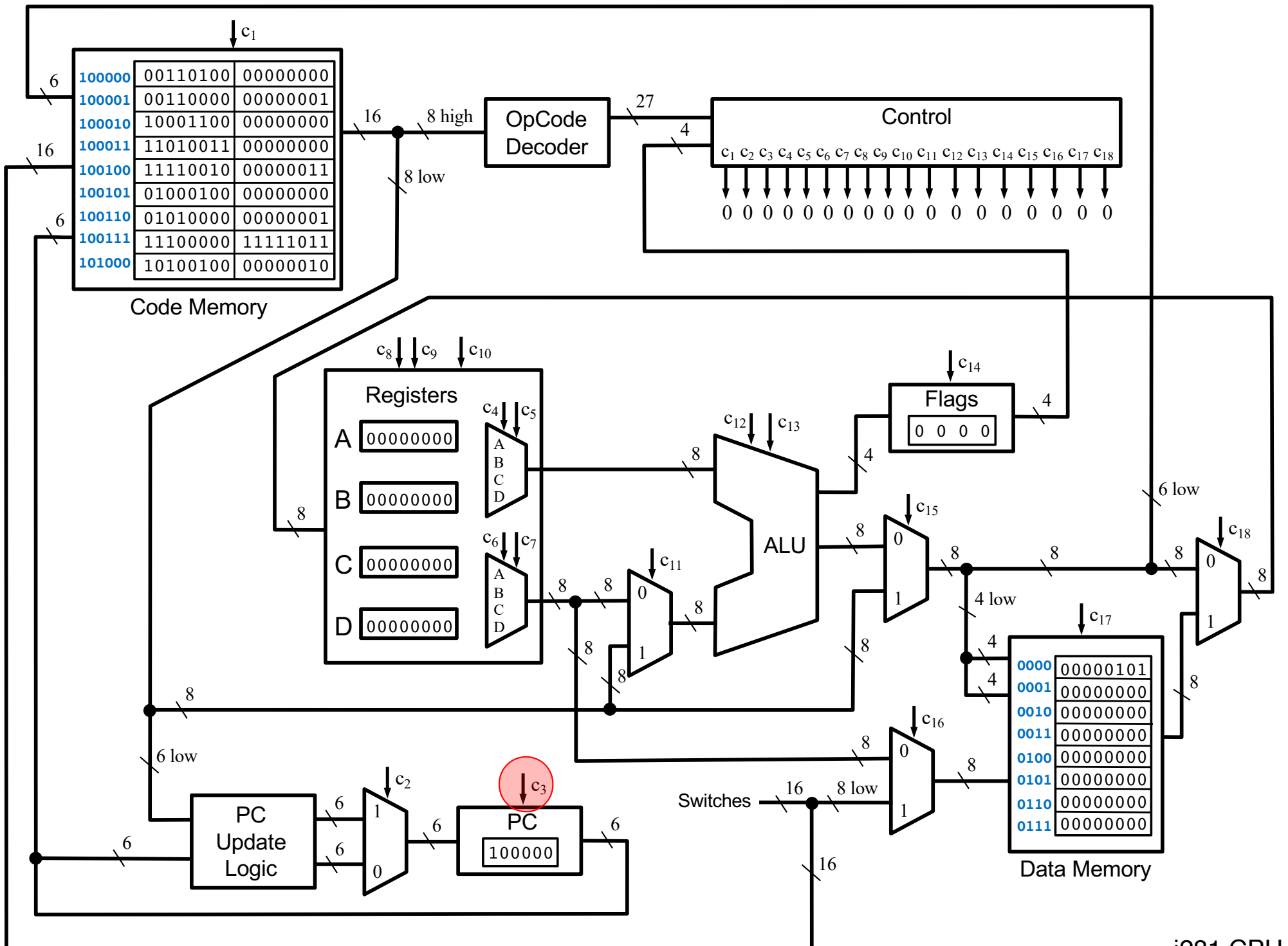
- B1= ZF
- B2= ~ZF
- B3= AND (~ZF, XNOR(NF, OF))
- B4= XNOR(NF, OF)

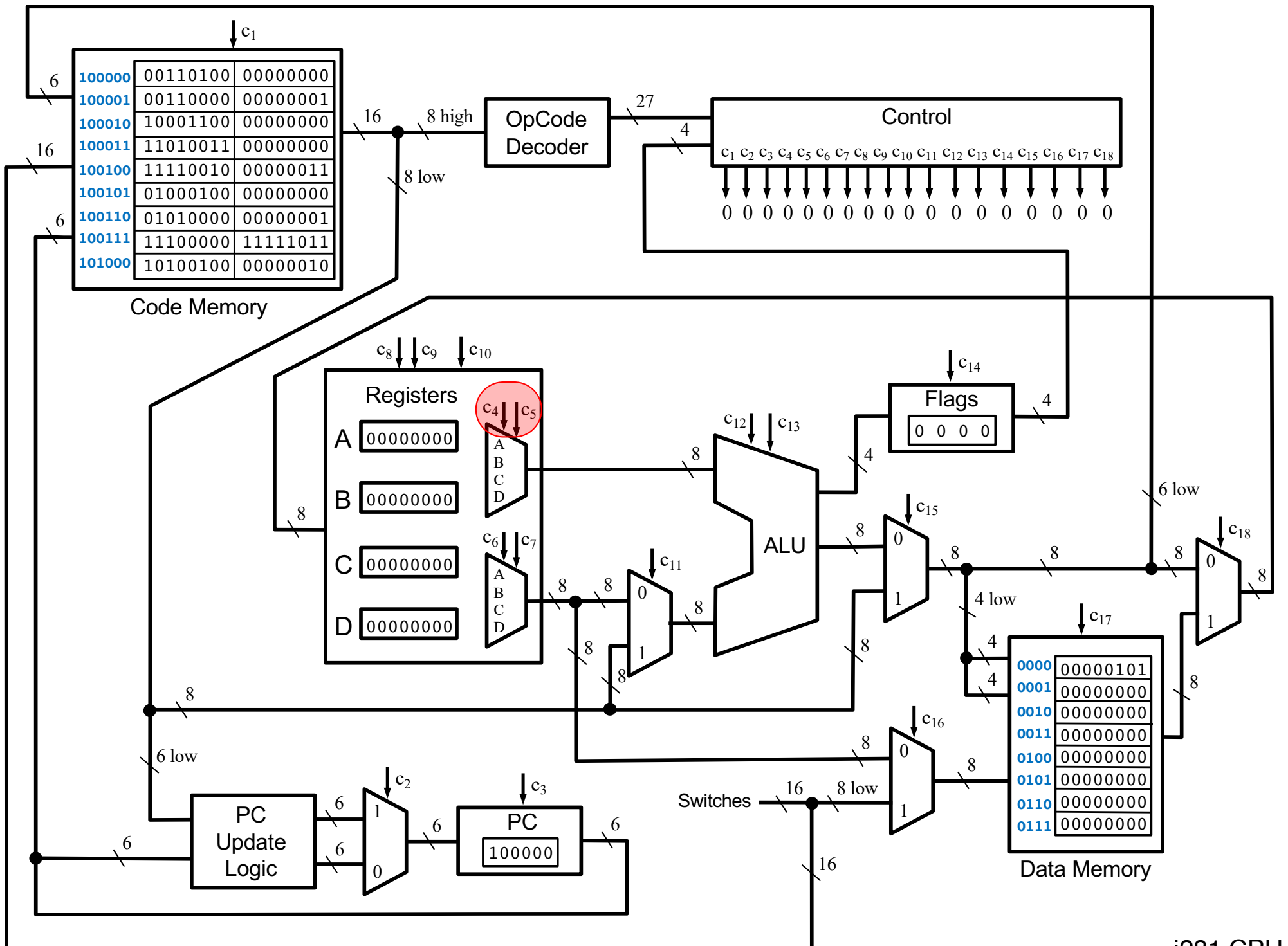
- Zero Flag (ZF)
- Negative Flag (NF)
- Overflow Flag (OF)

# **The Control Signals**

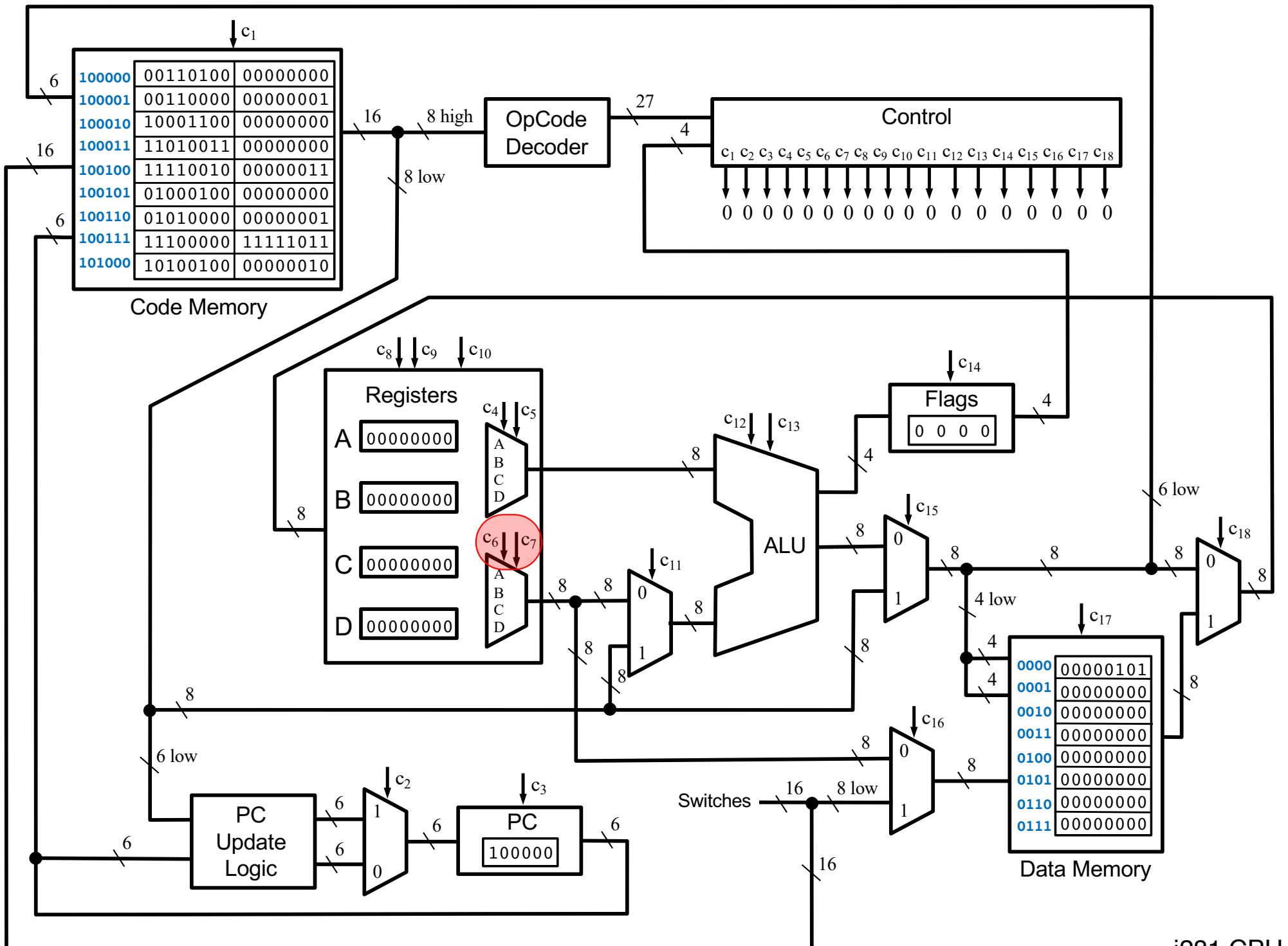


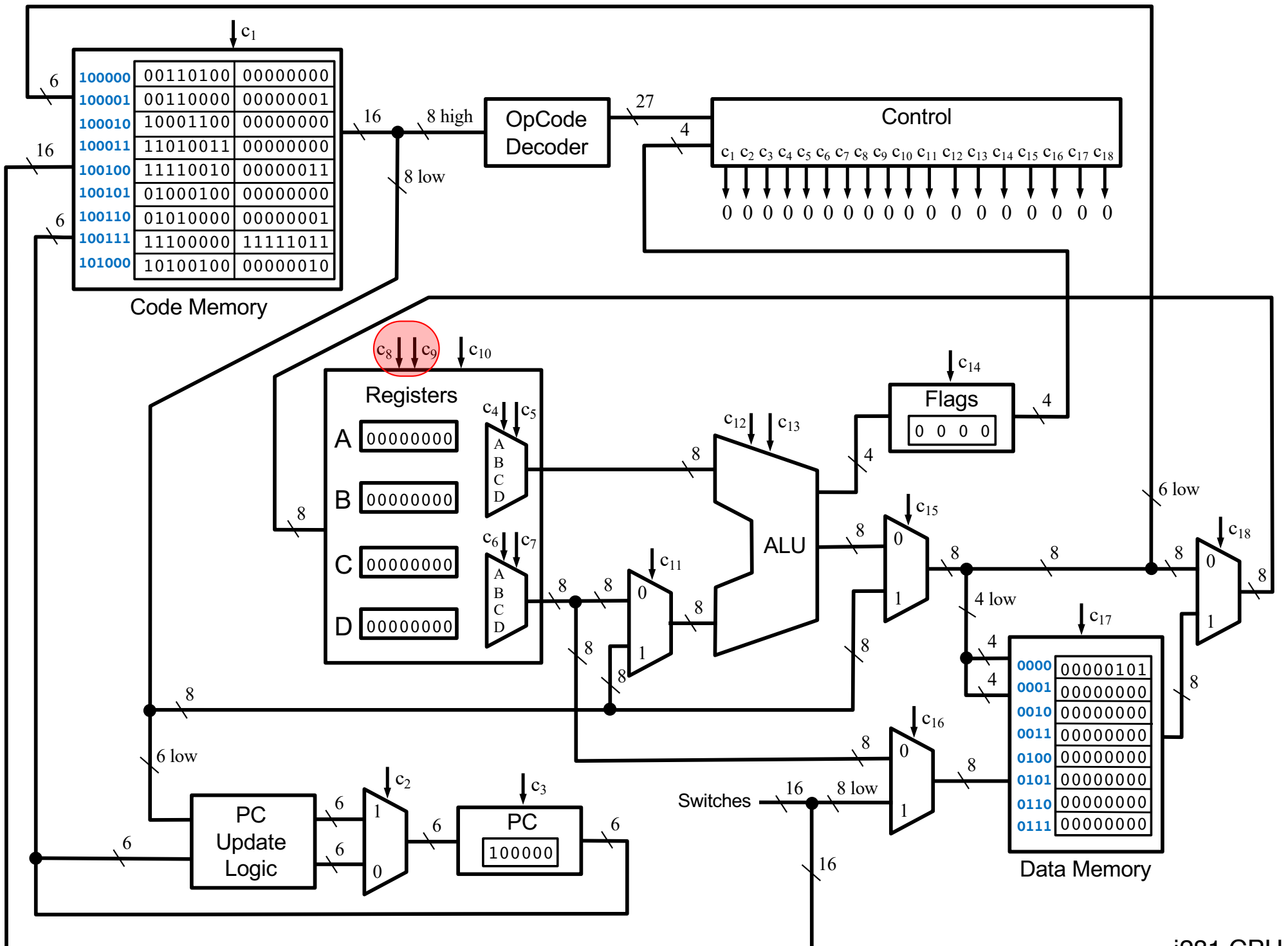


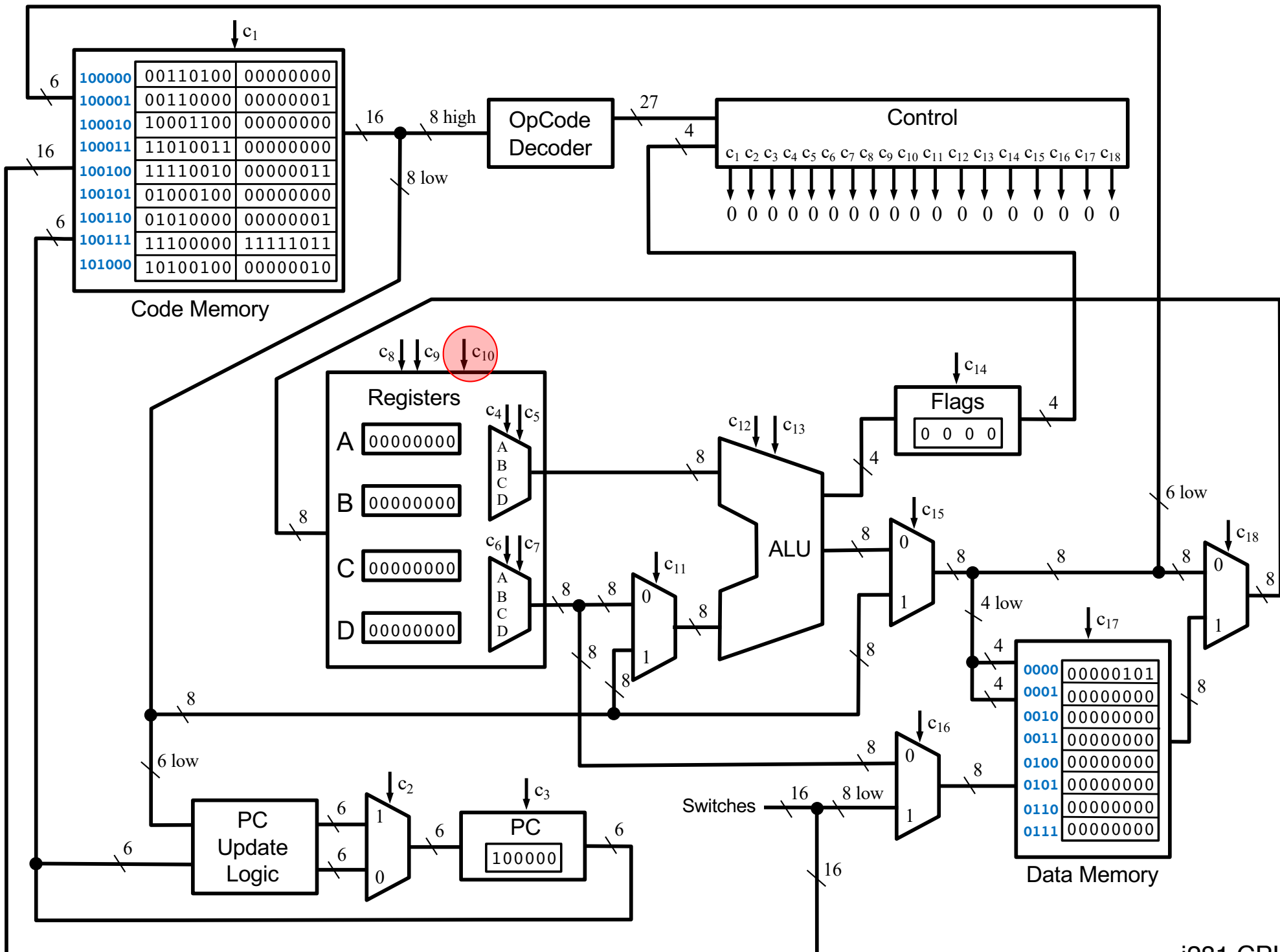


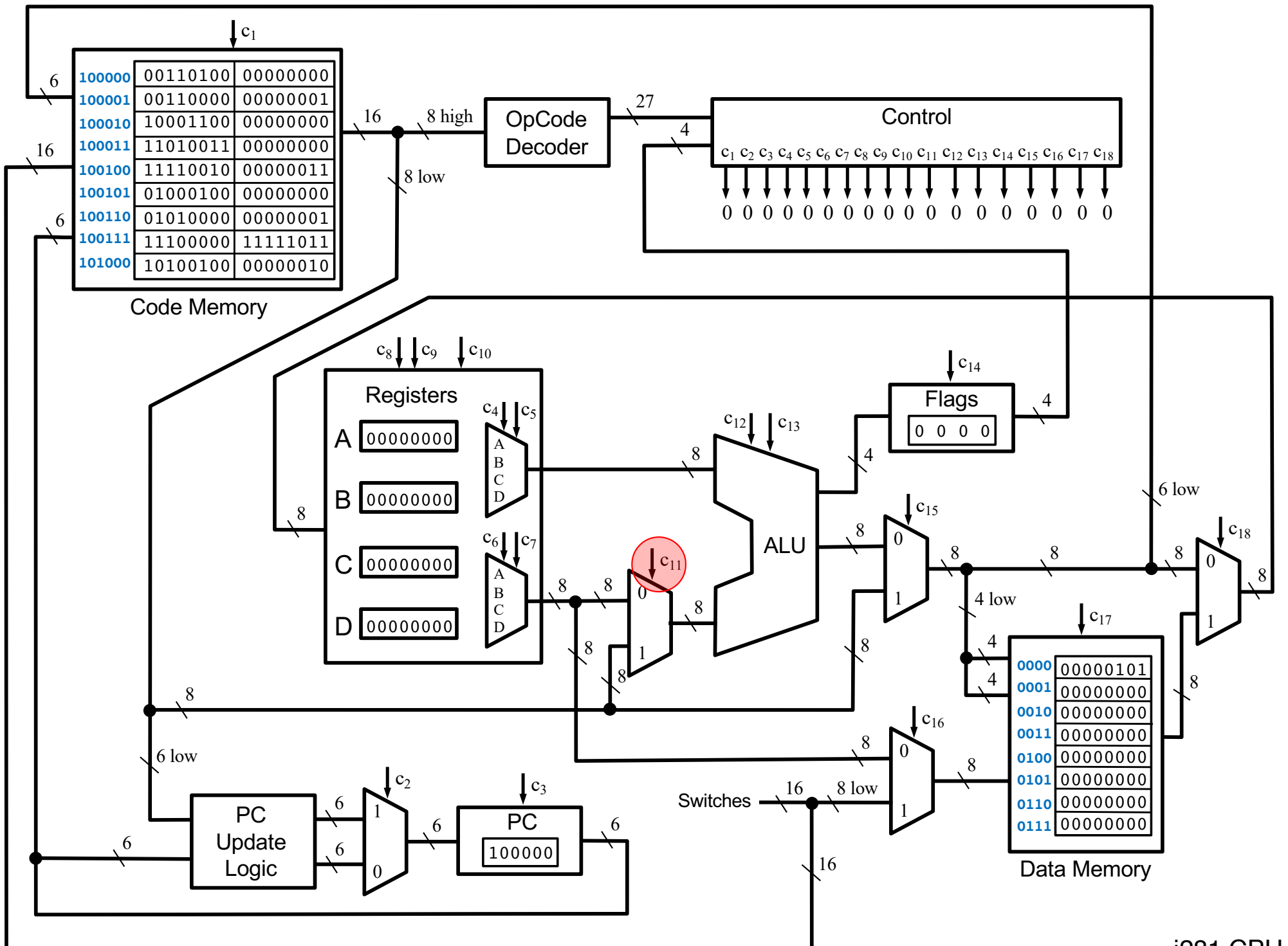


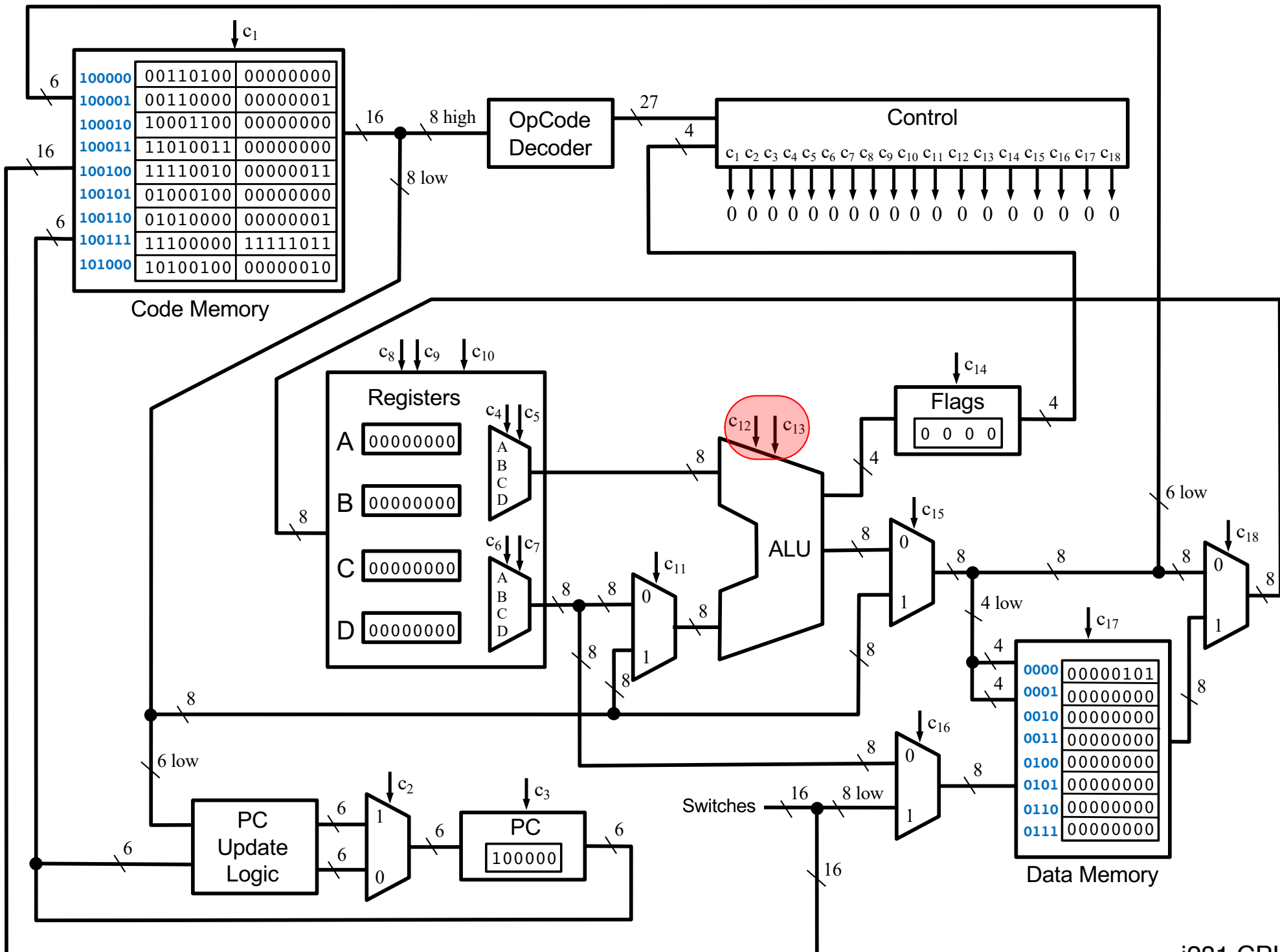


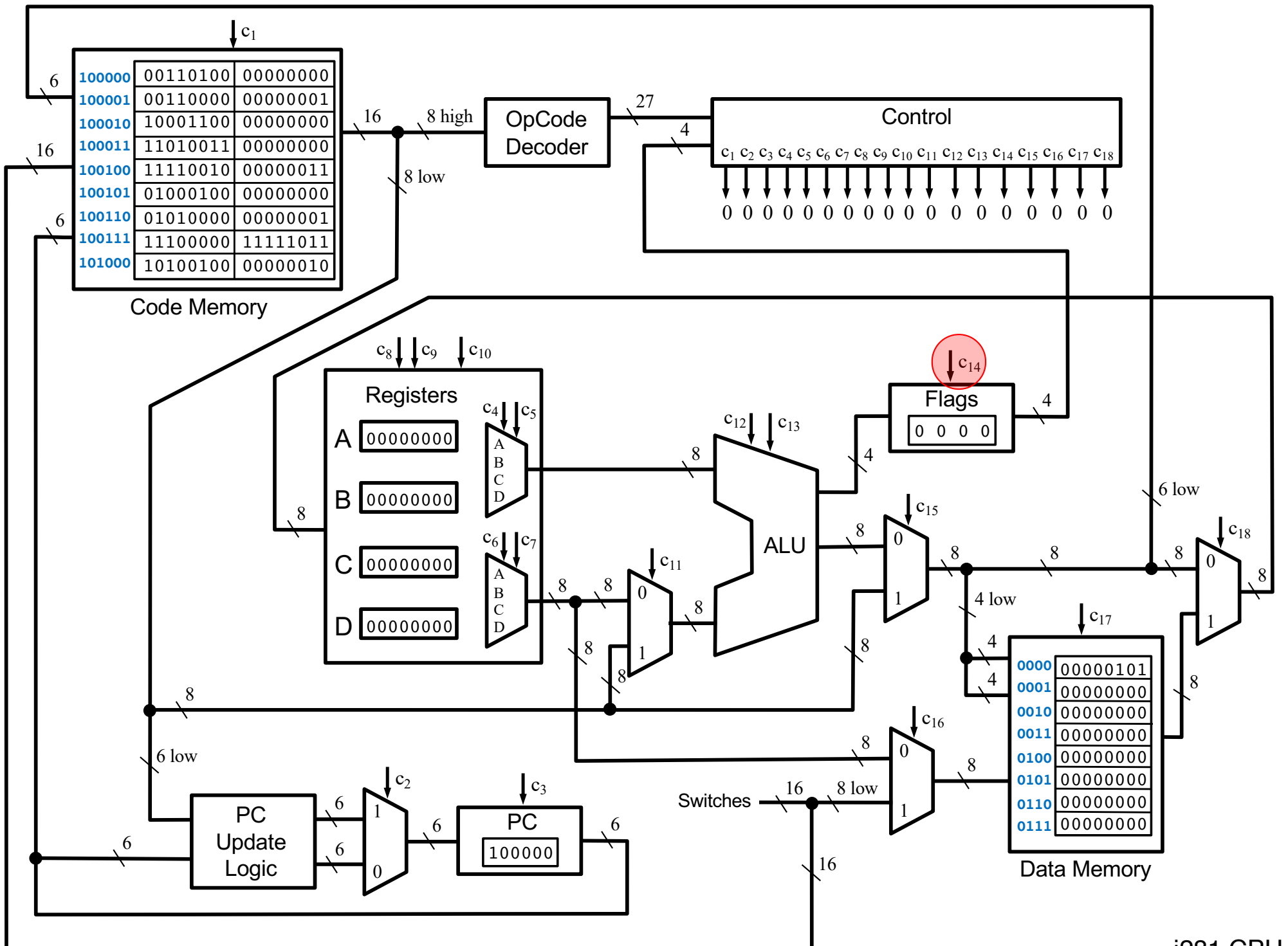


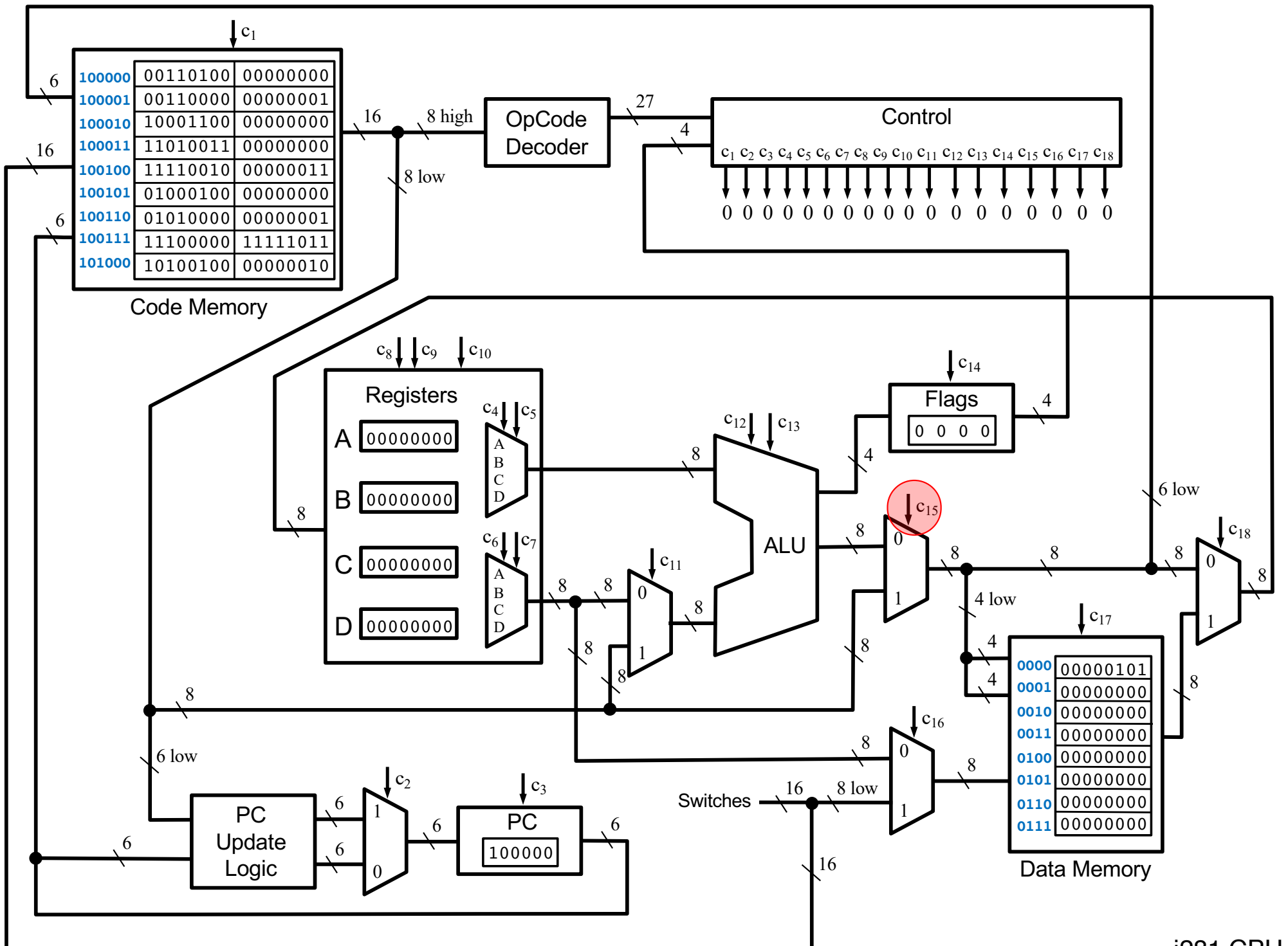


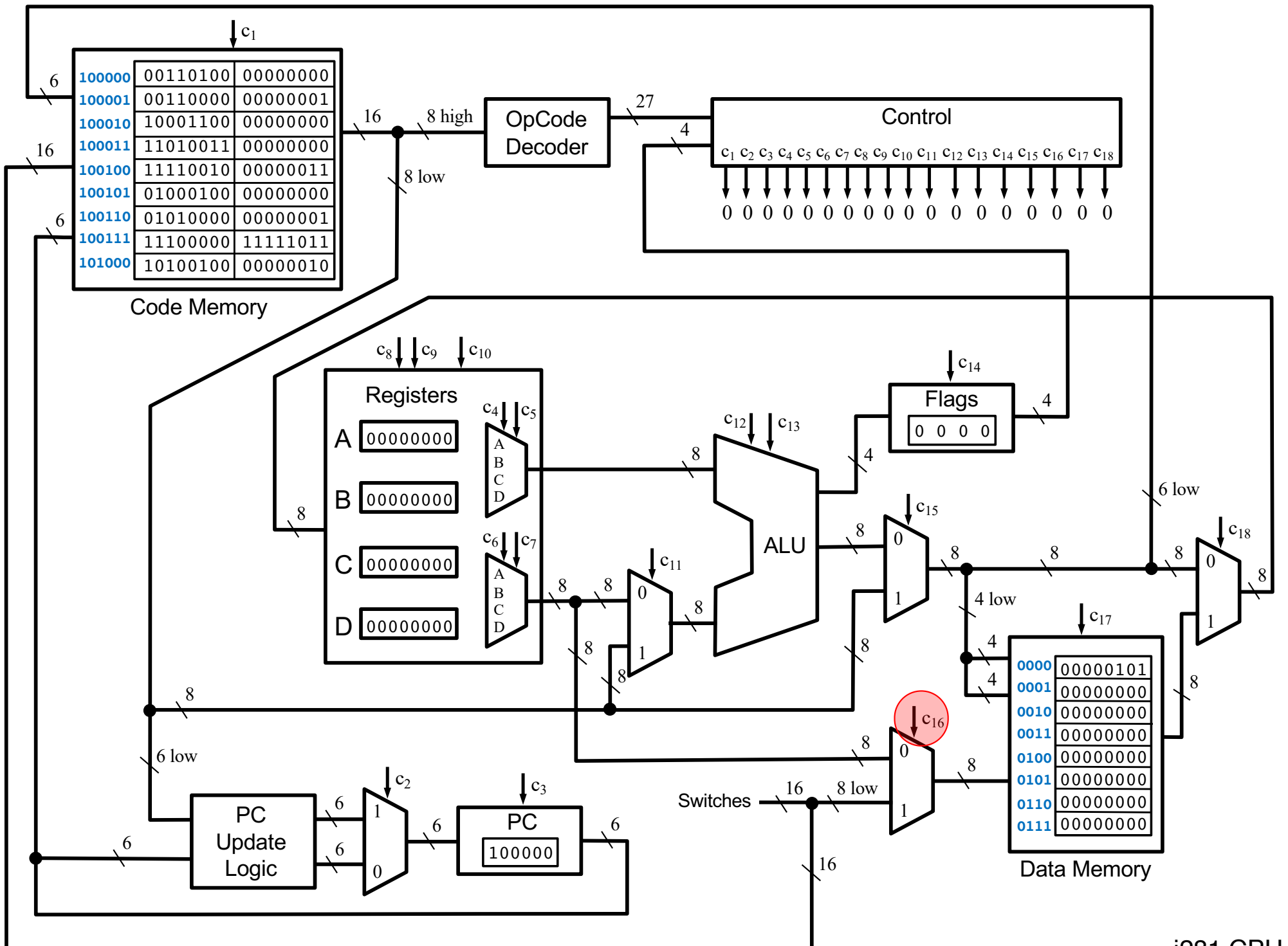




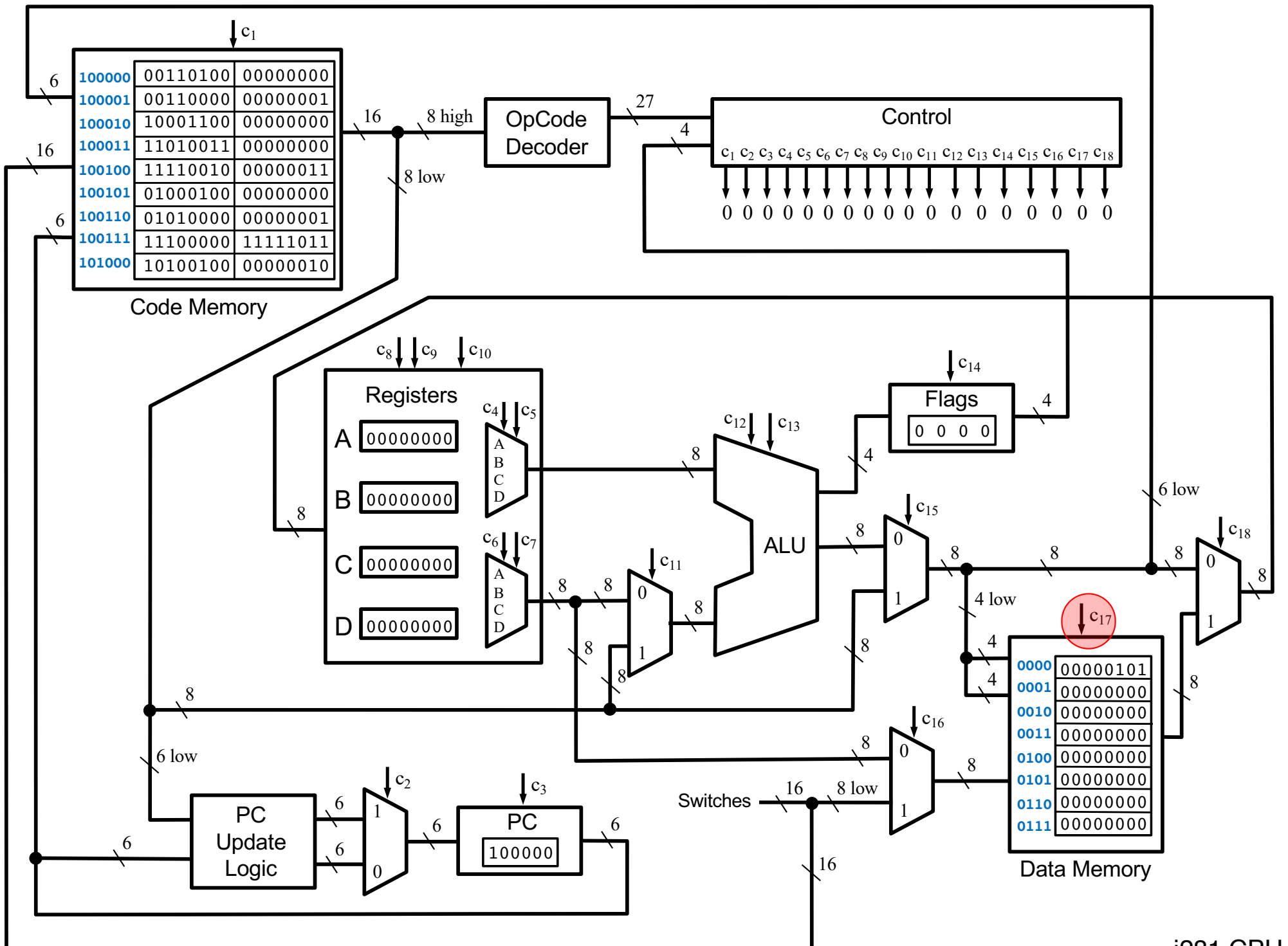


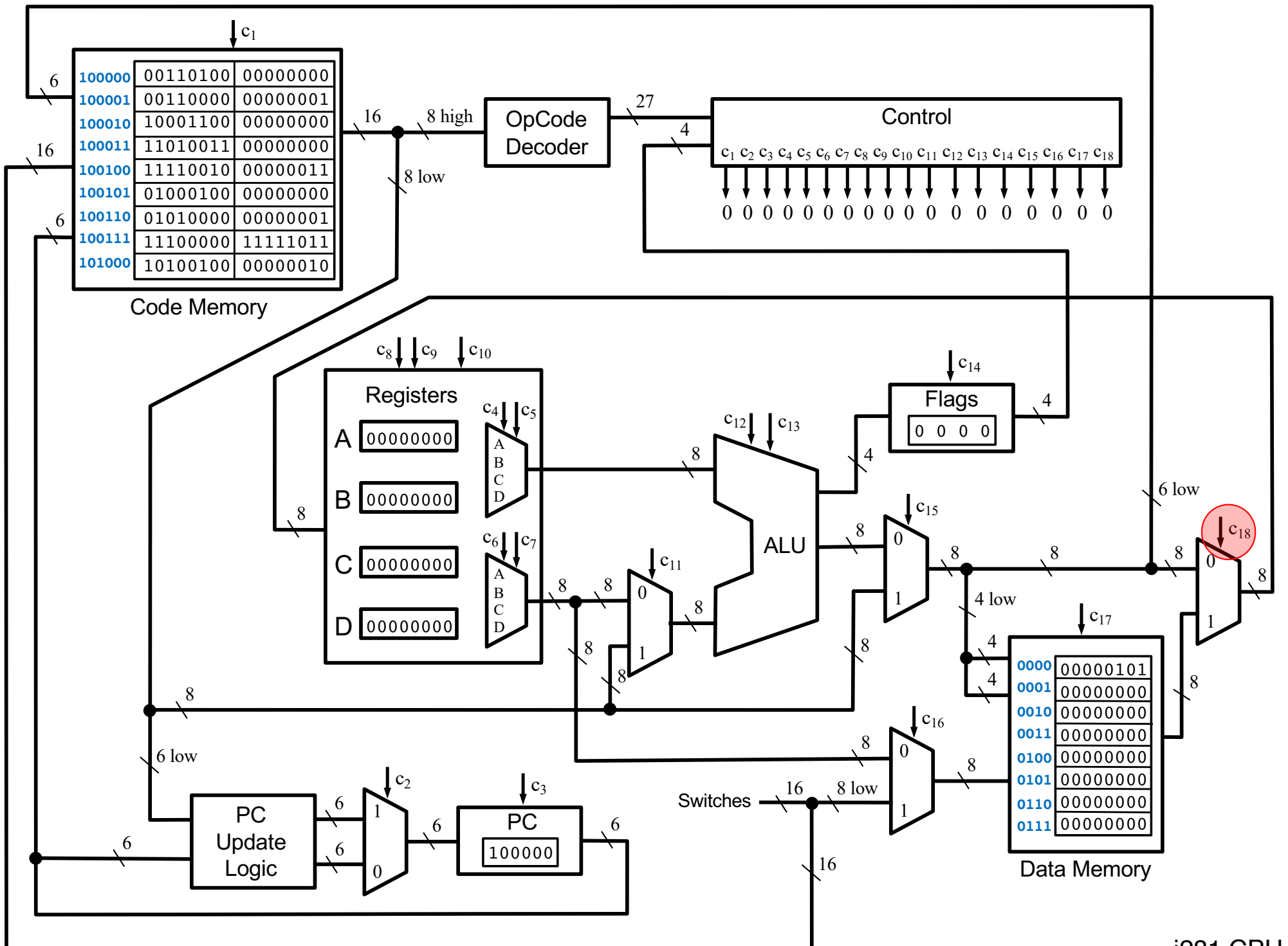


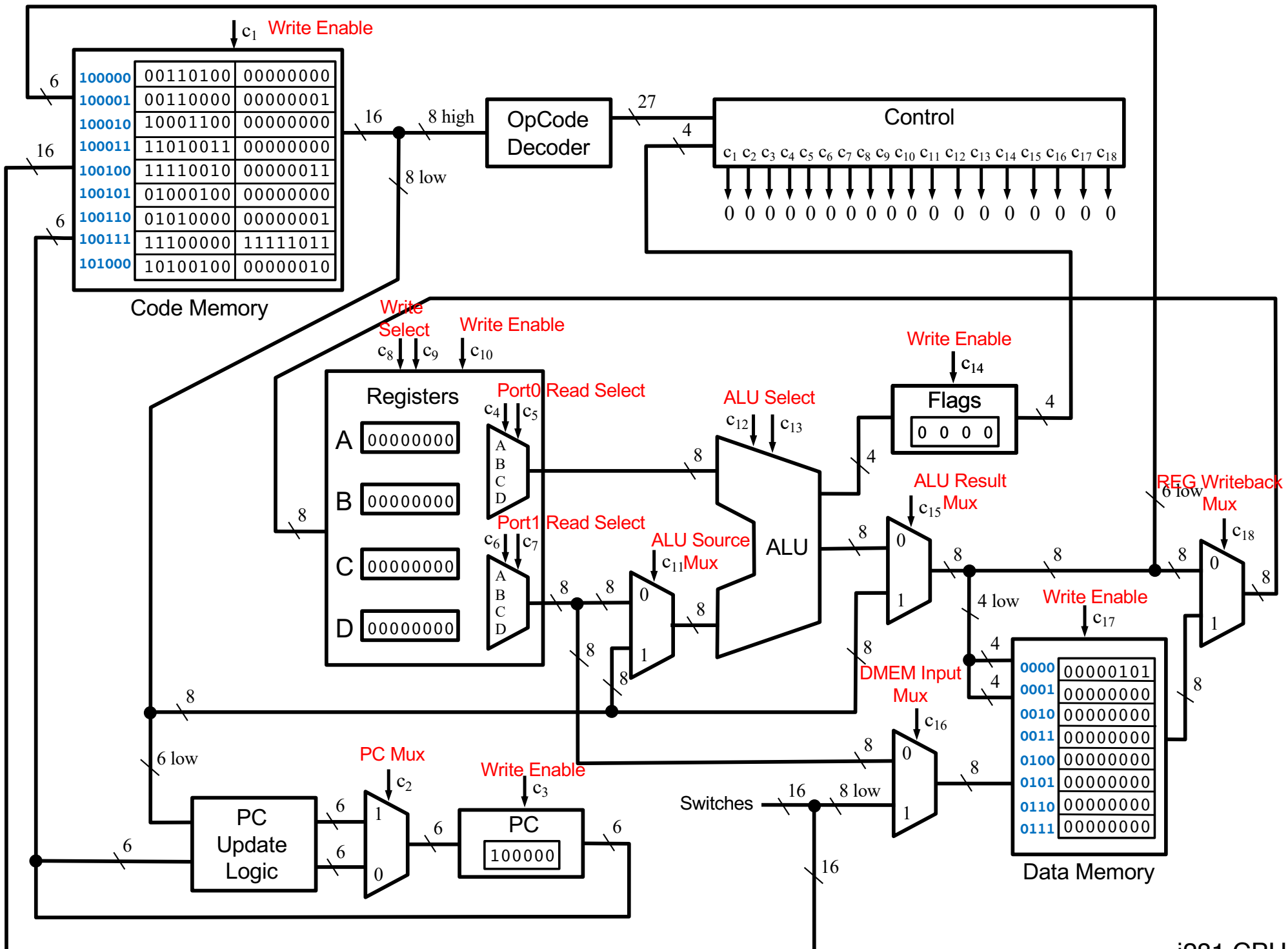


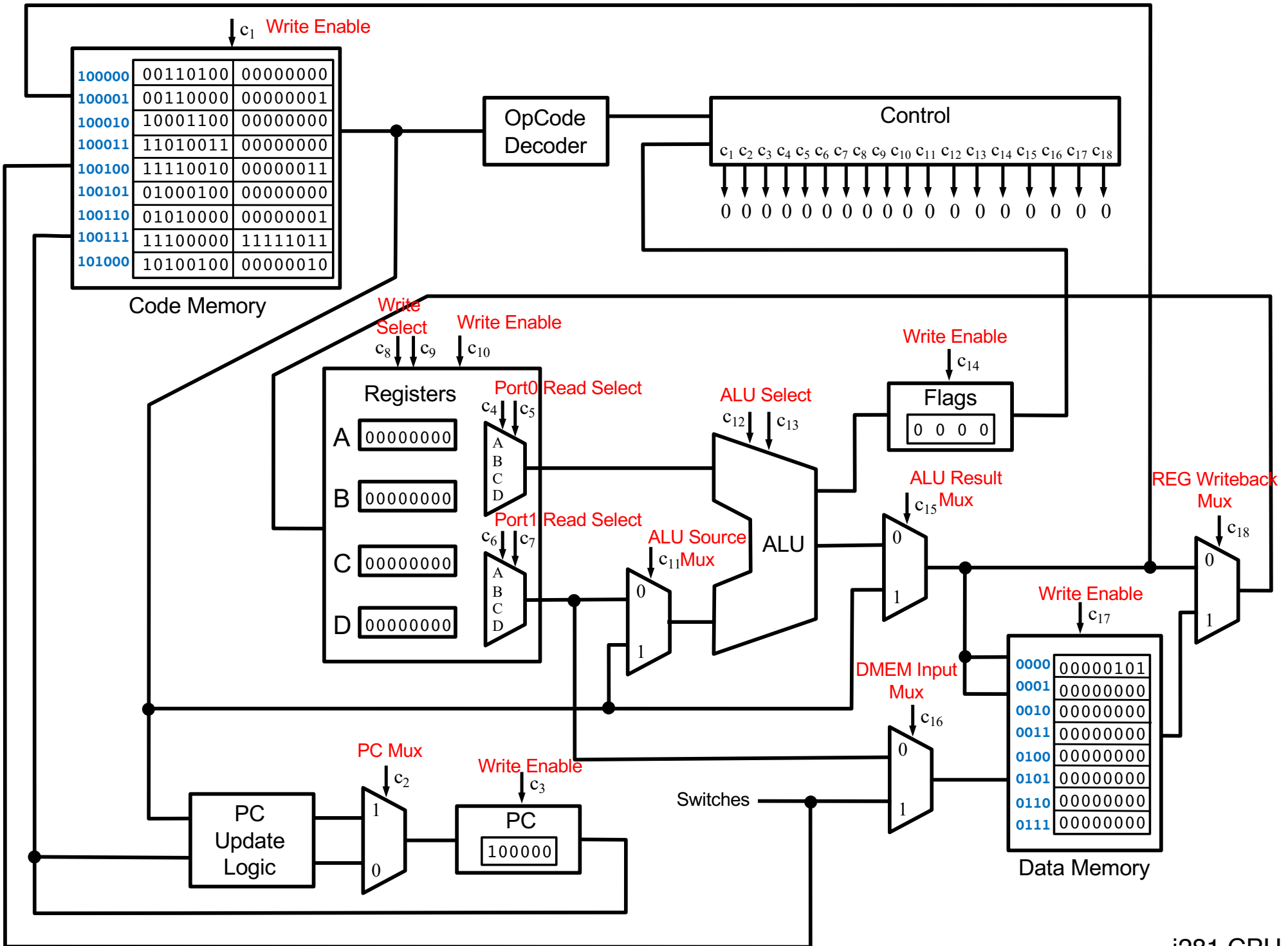




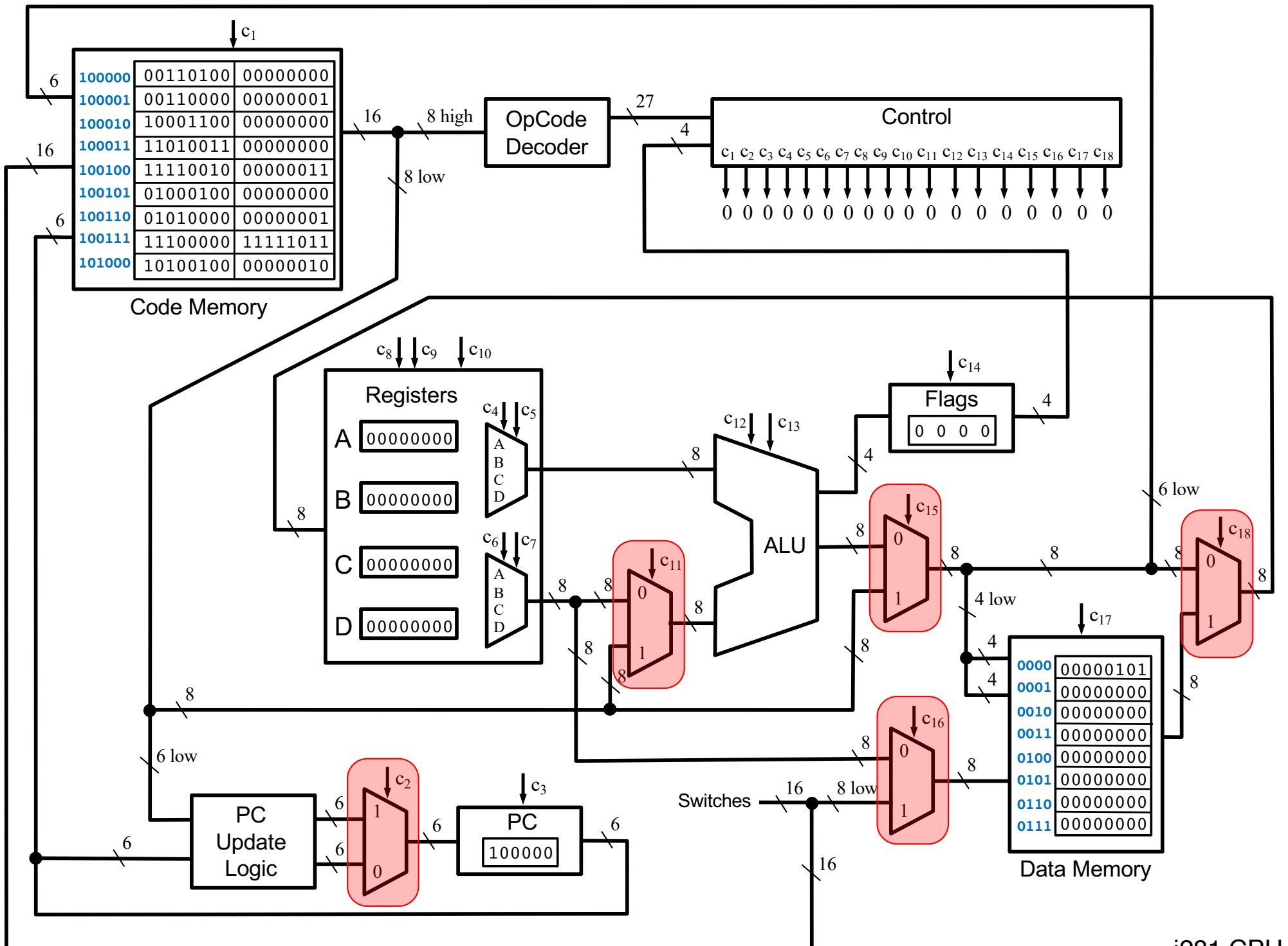




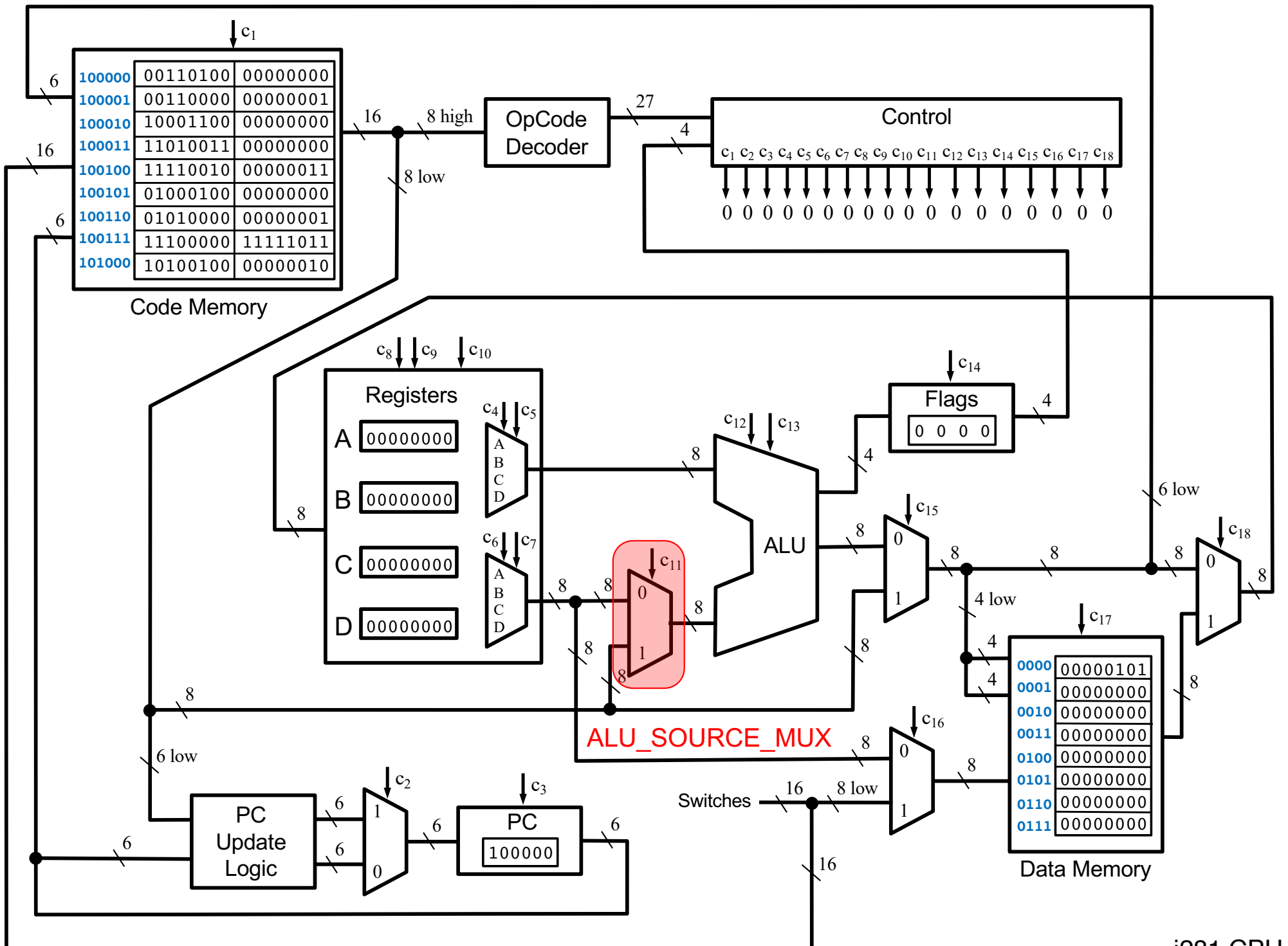




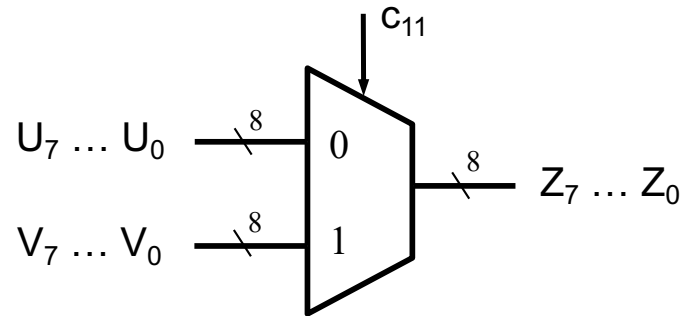
# **The Five Bus Multiplexers**



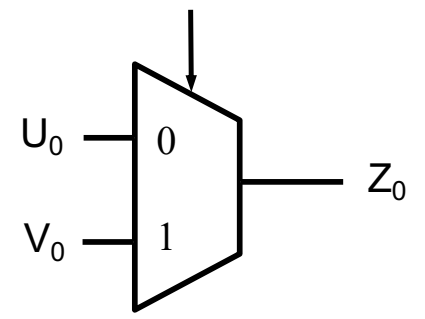
i281 CPU

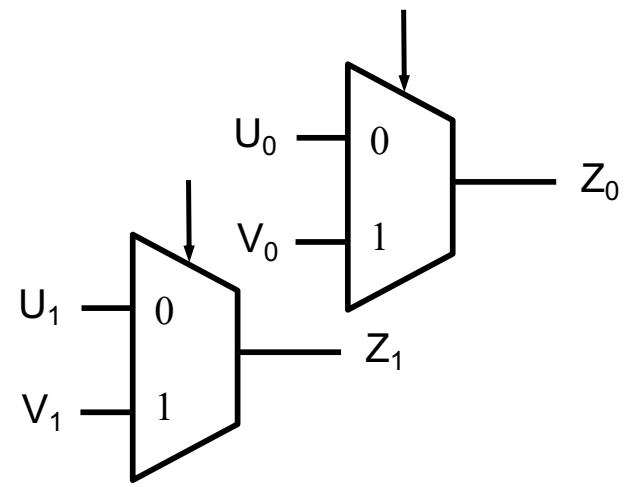


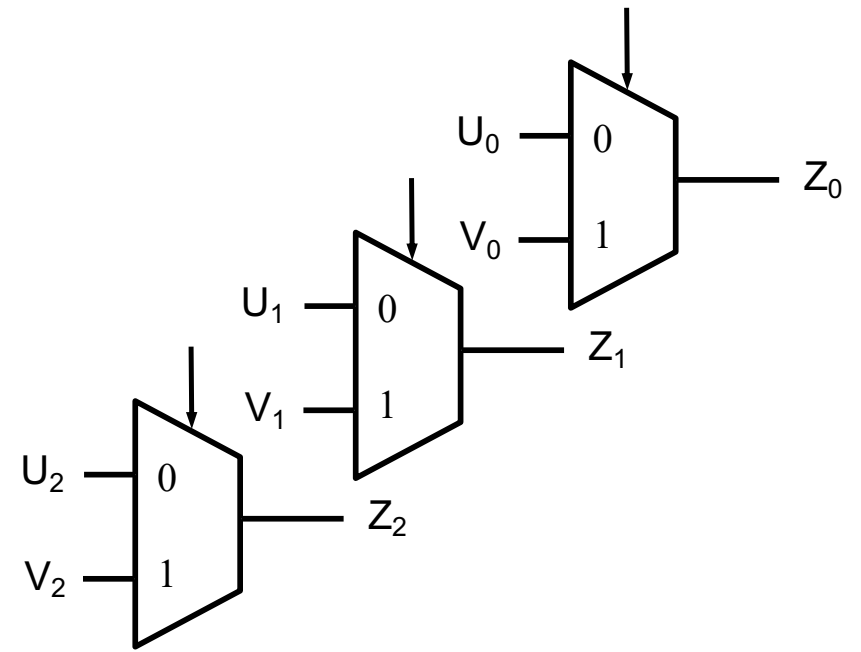
# 2-to-1 Bus Multiplexer (with 8-bit lines)

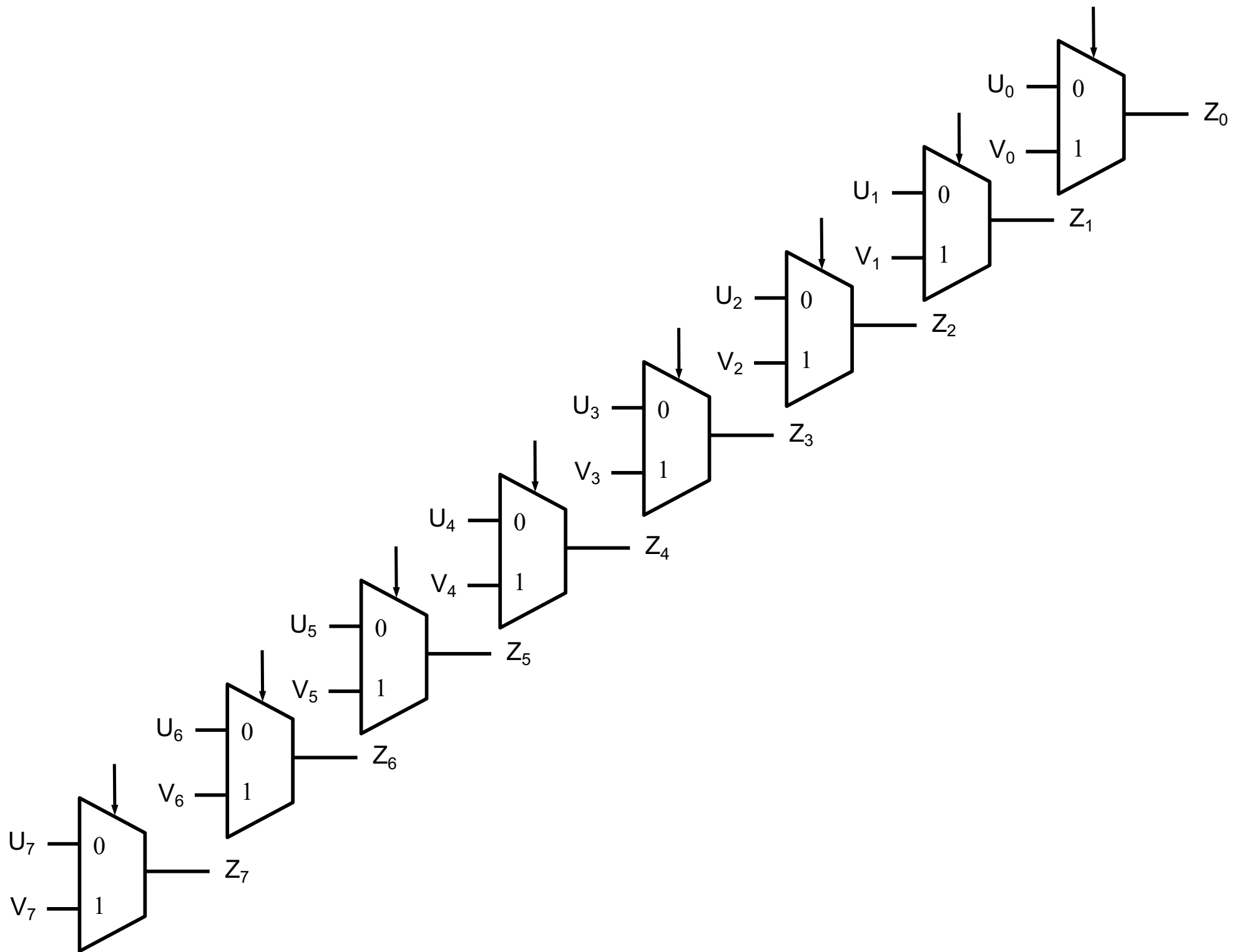


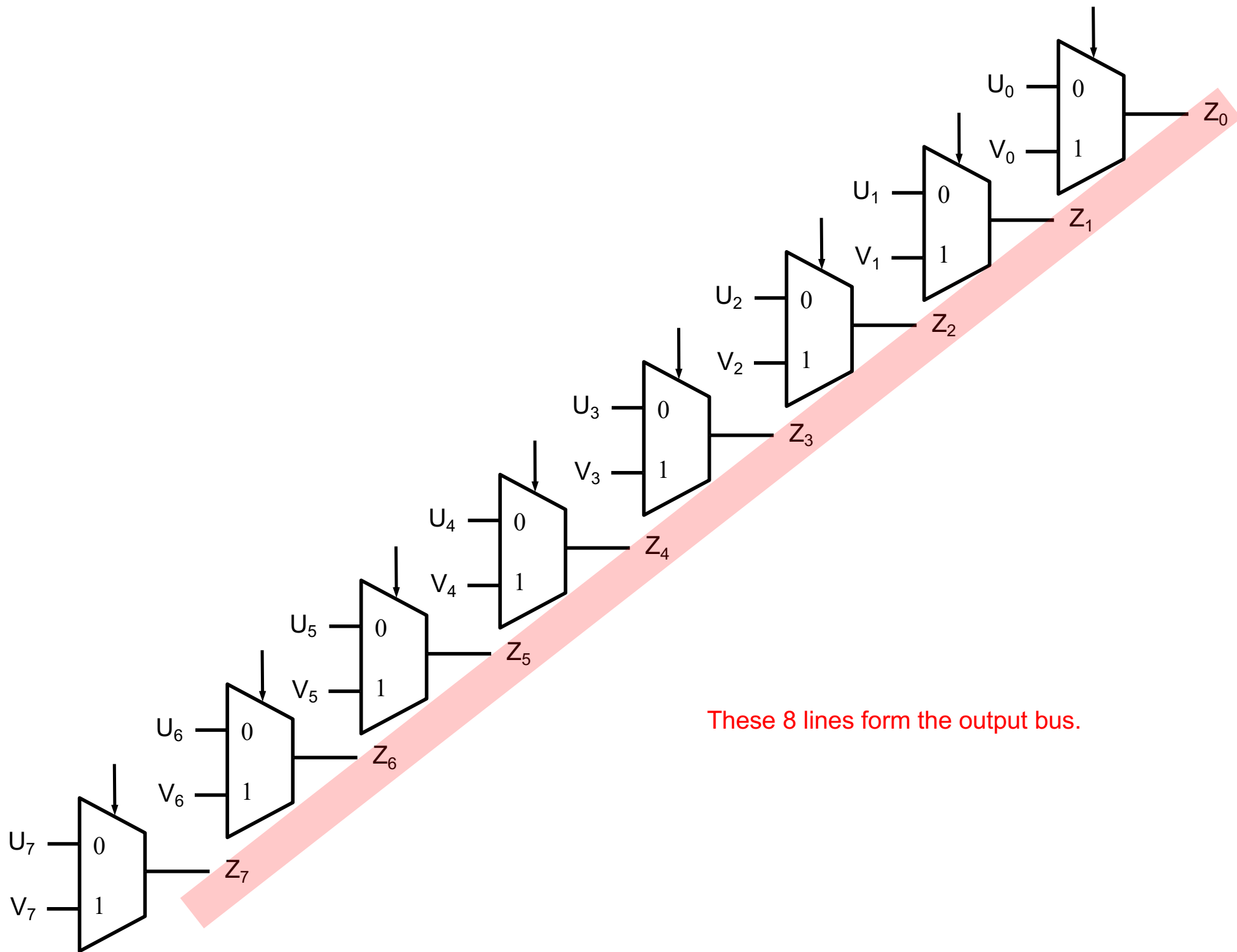


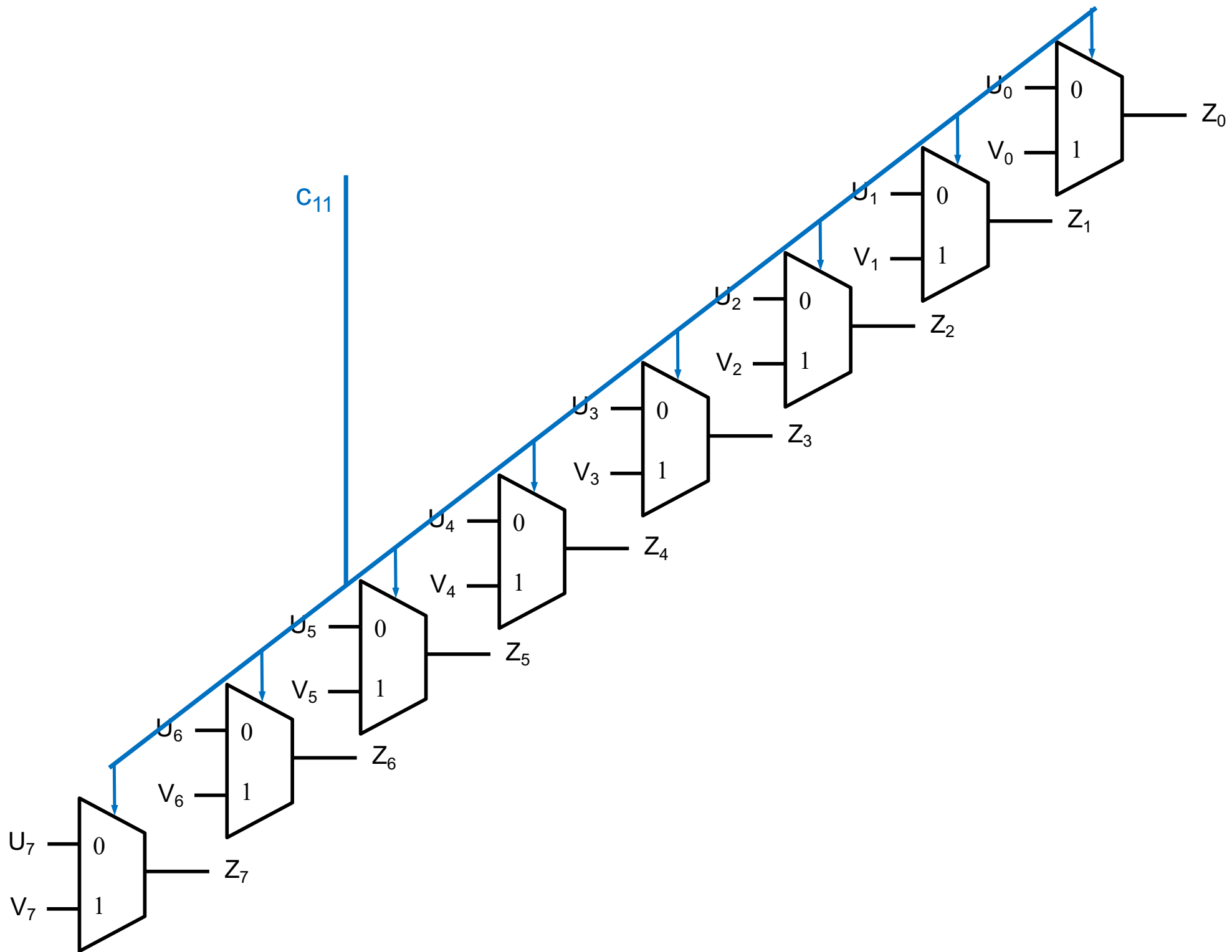


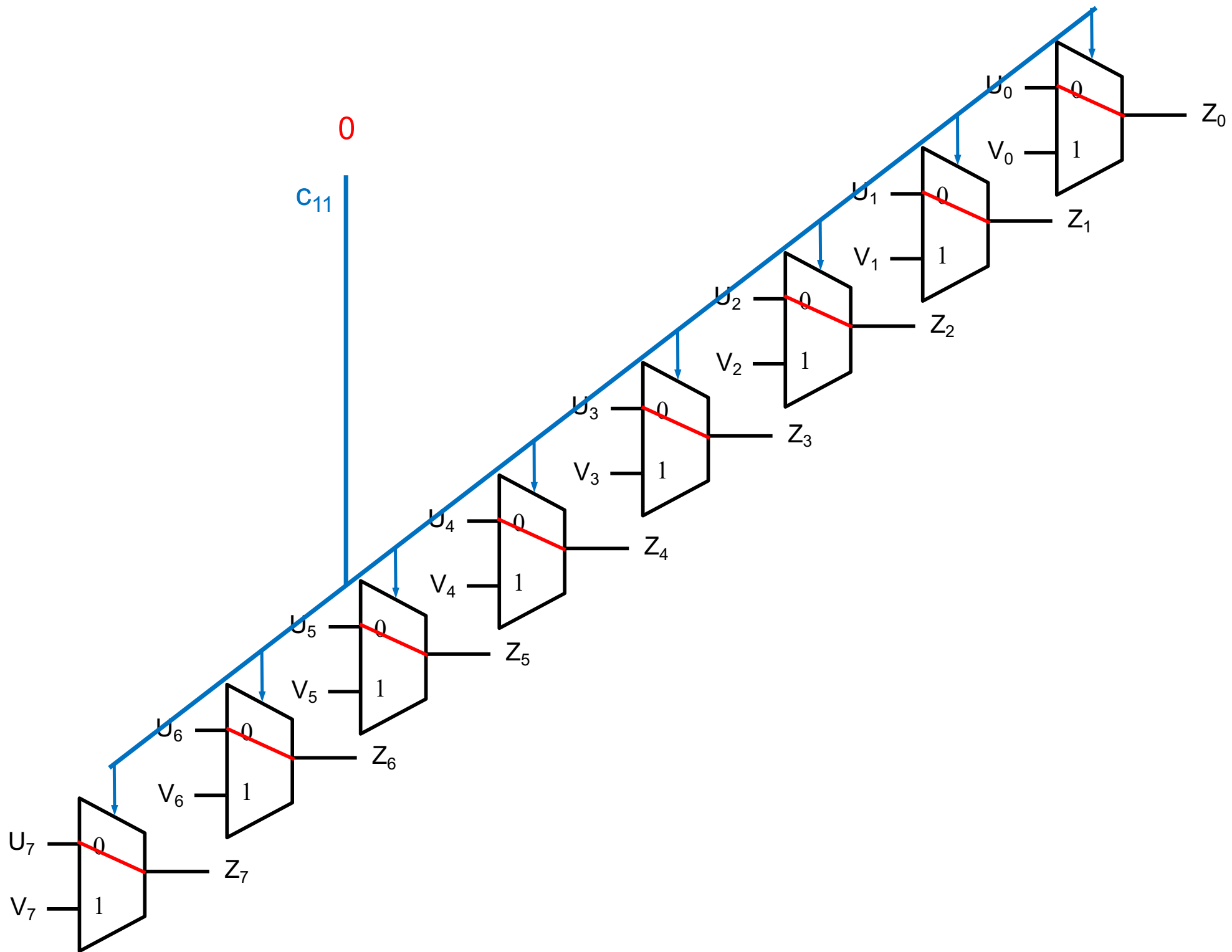


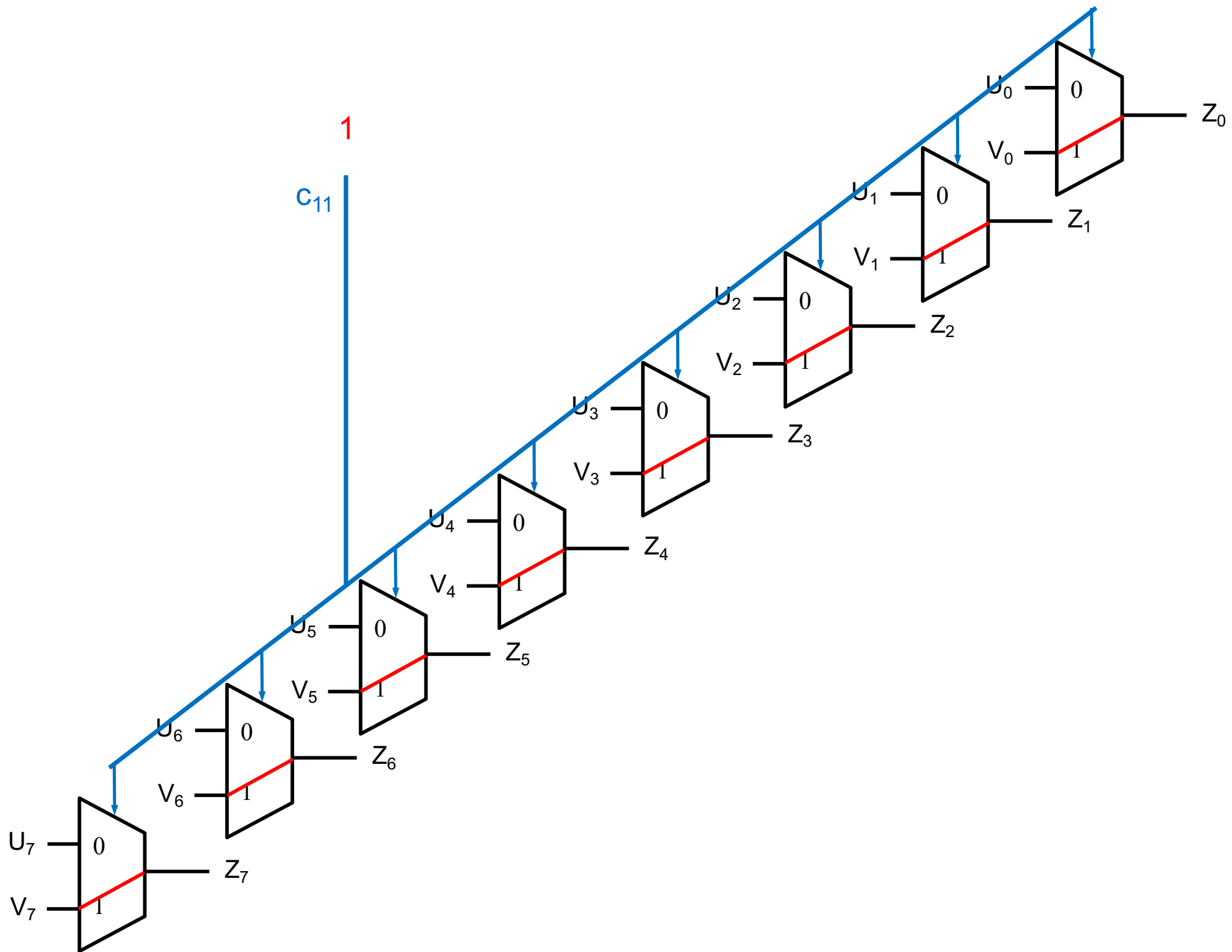




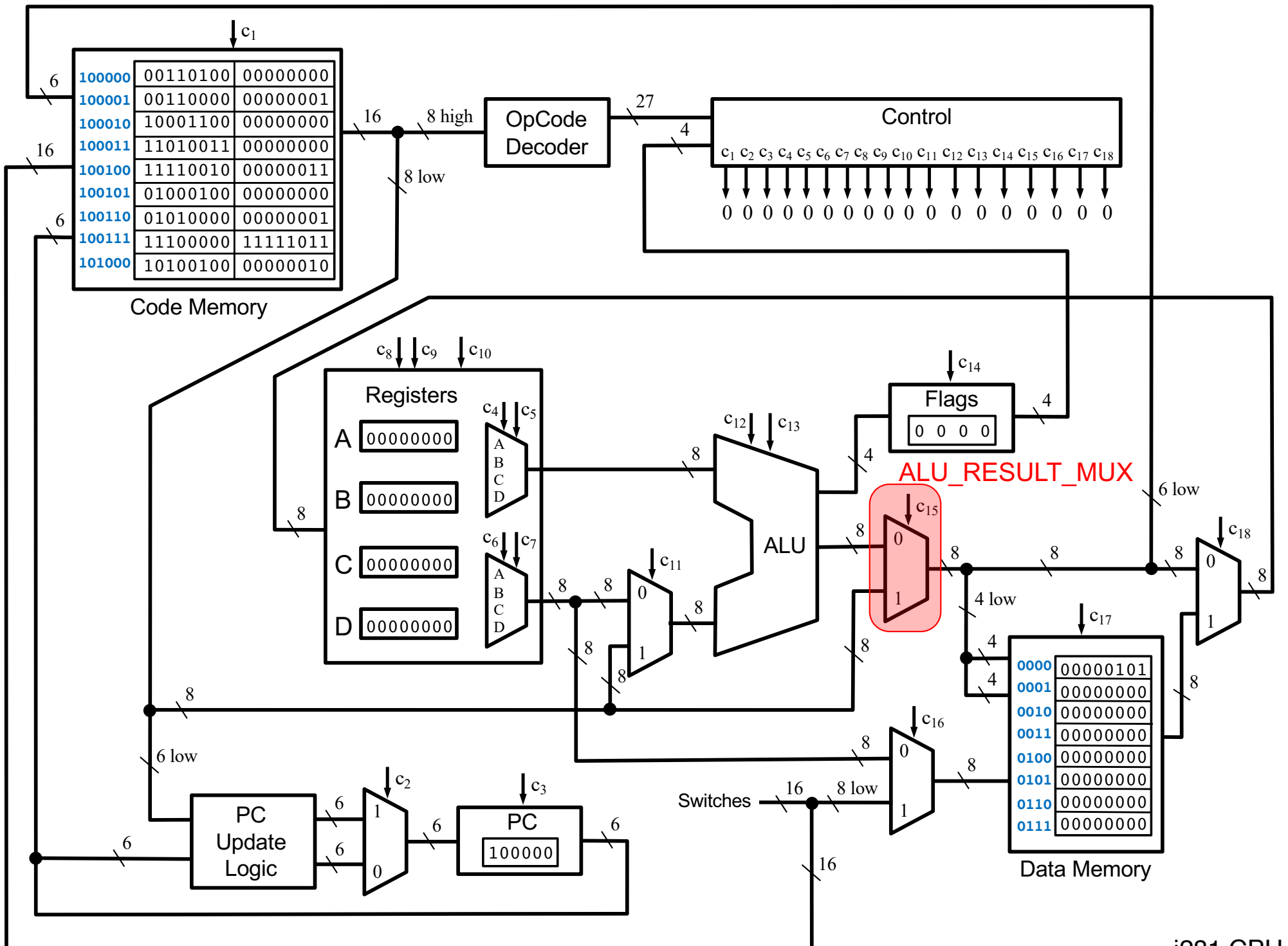


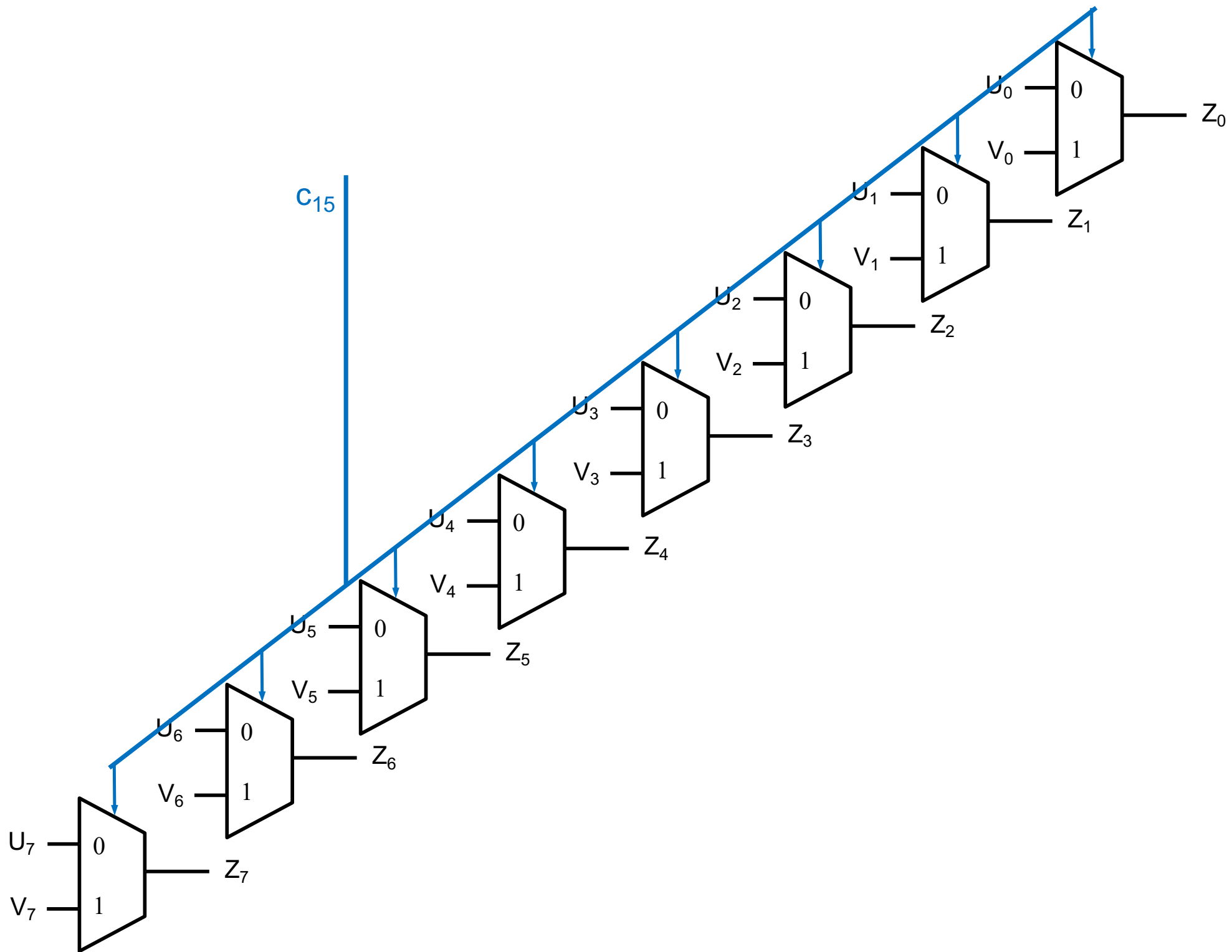


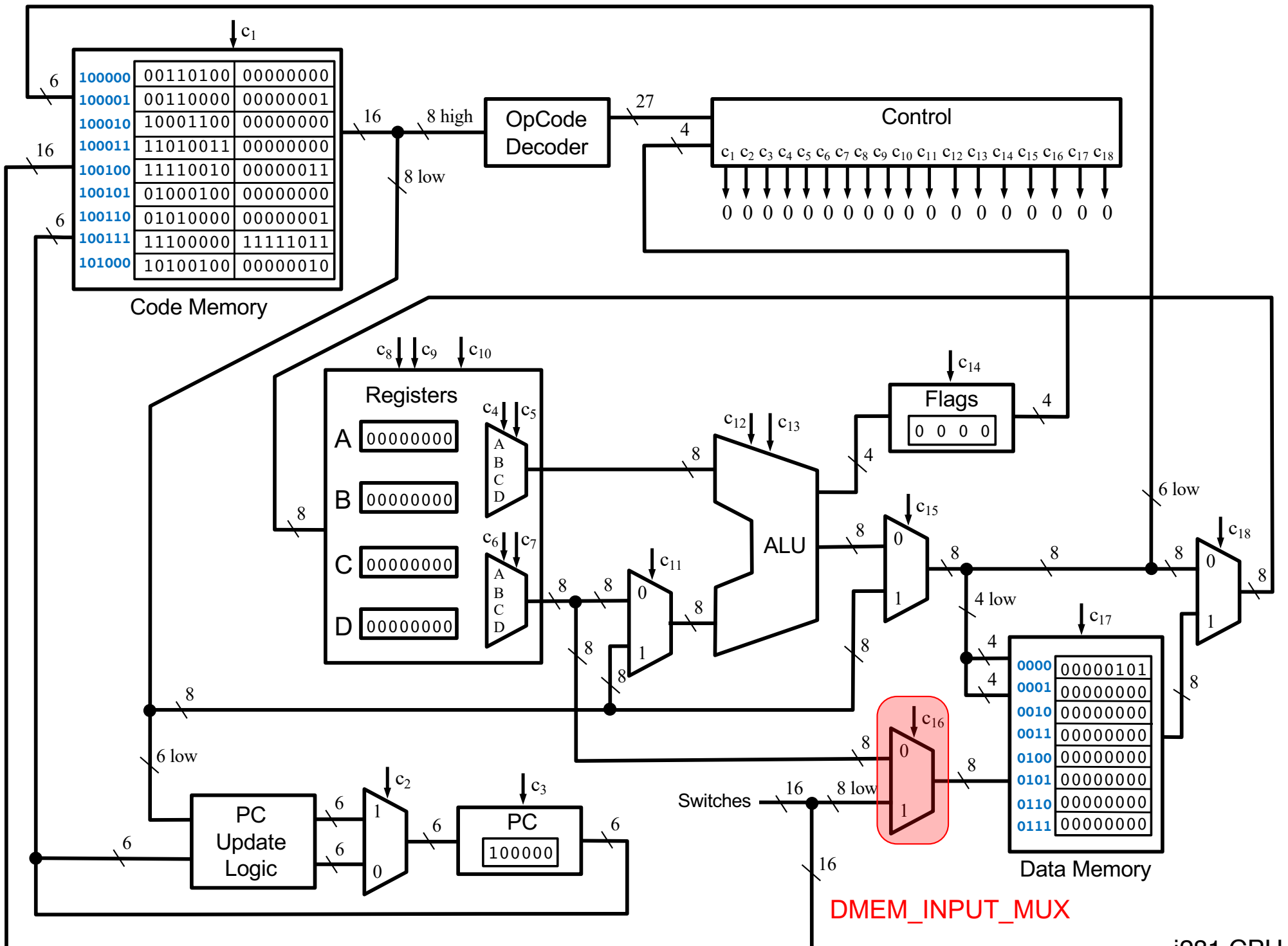


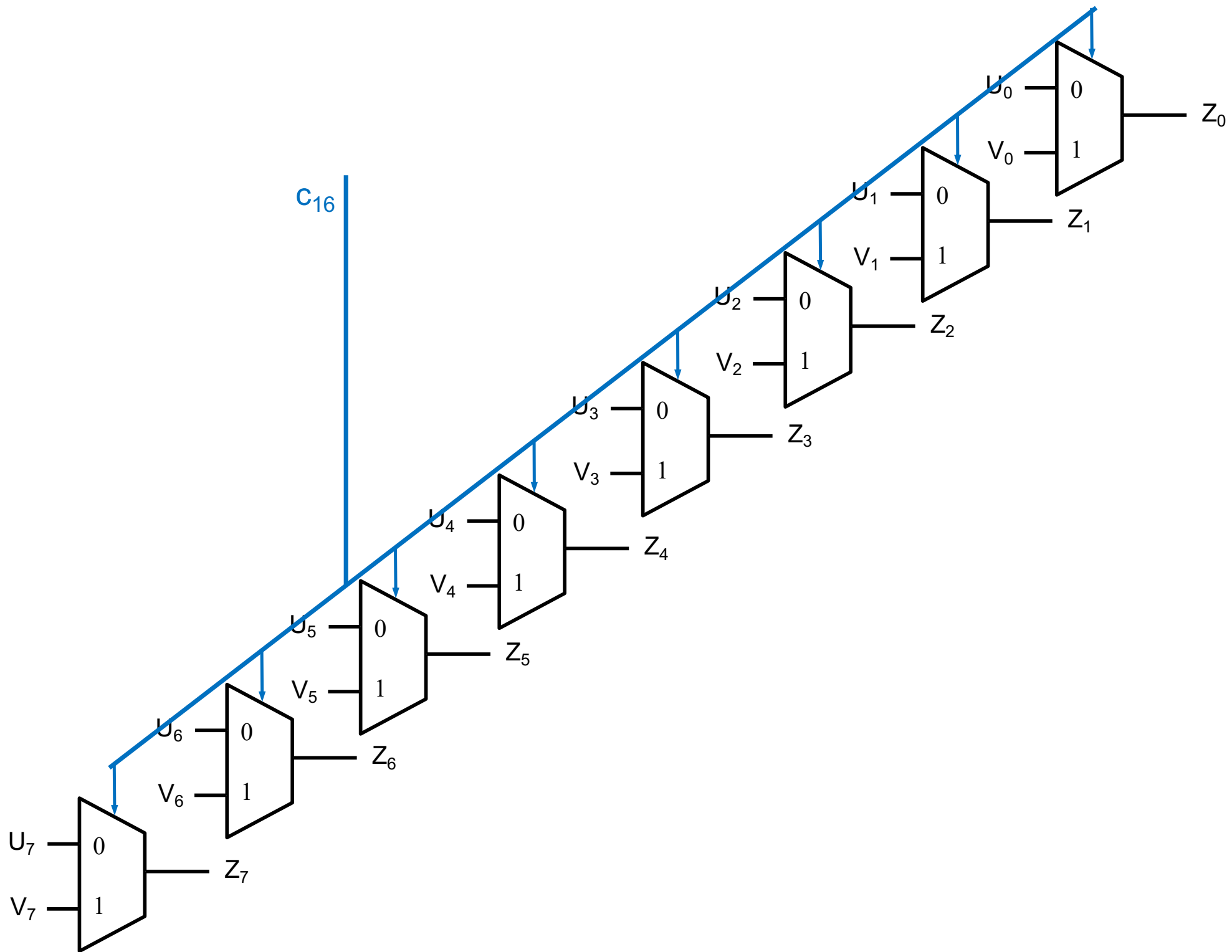


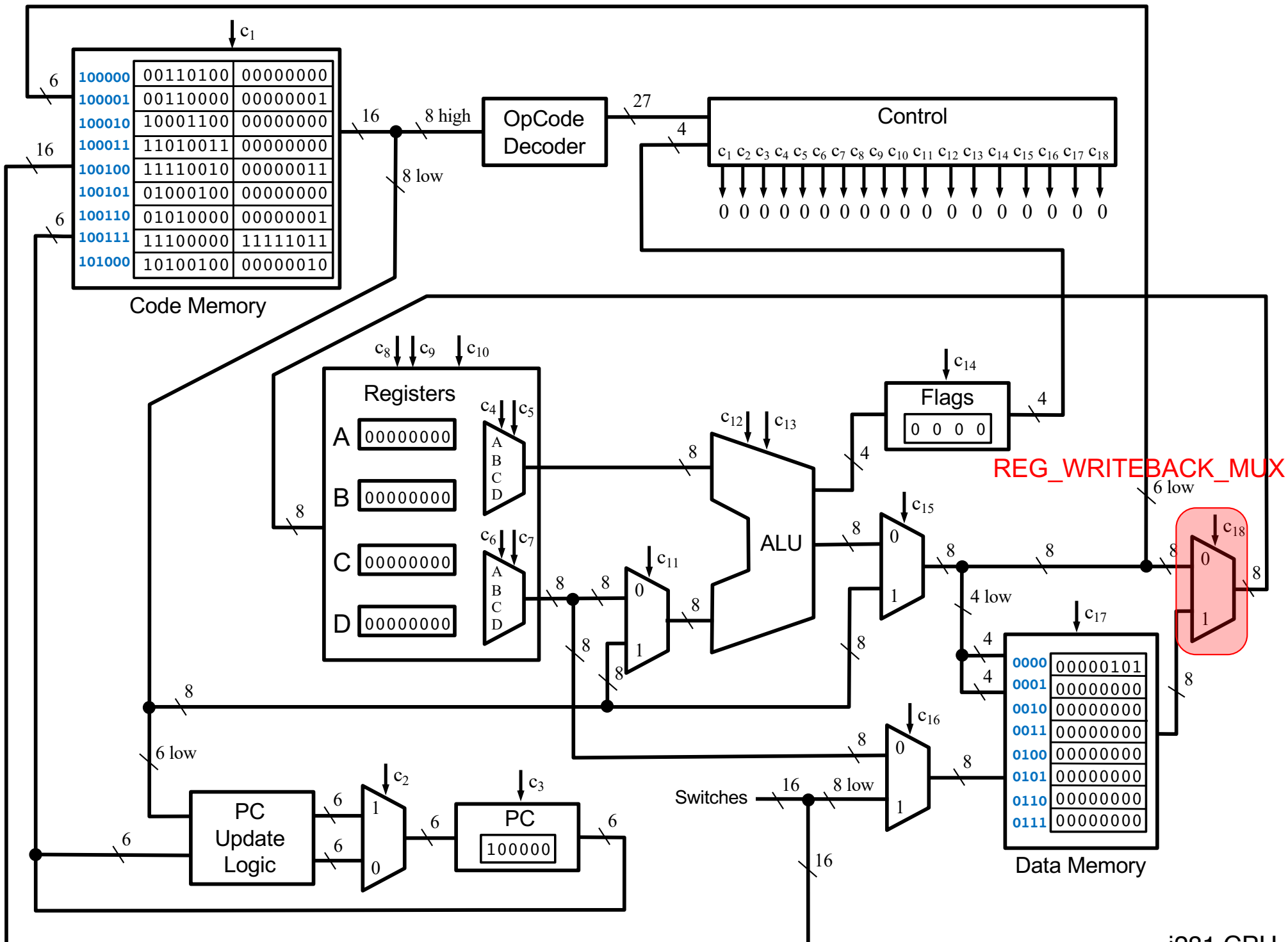


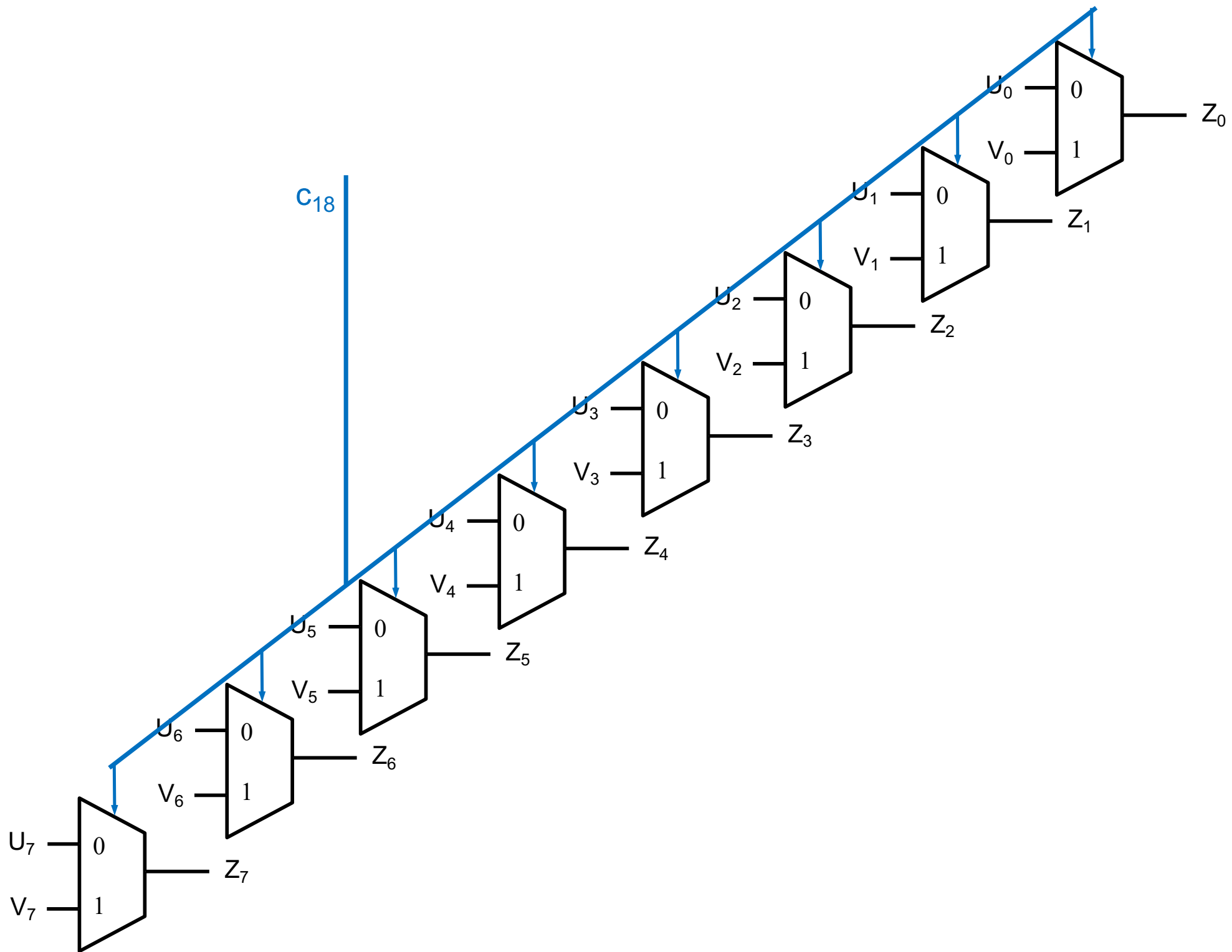


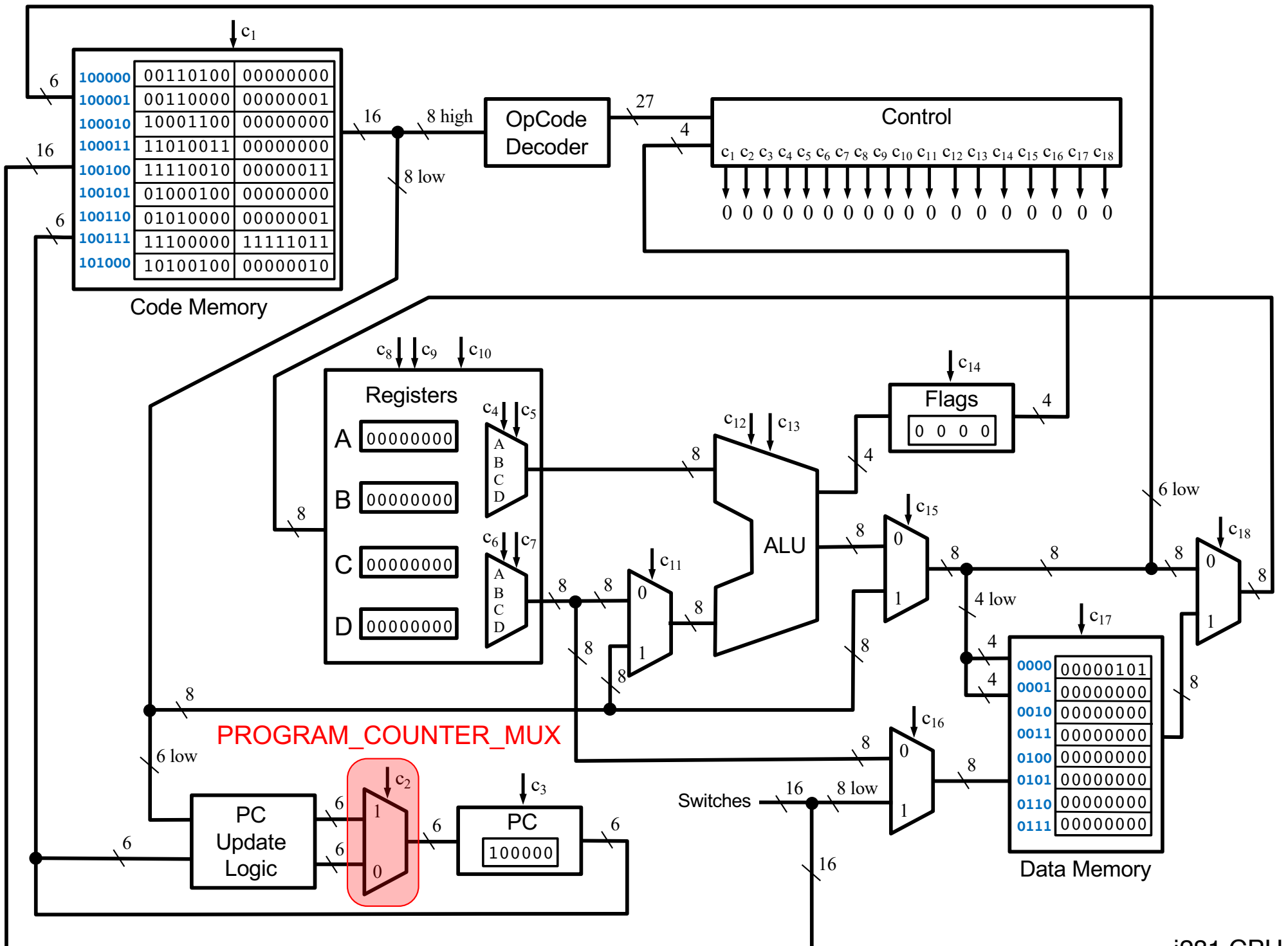


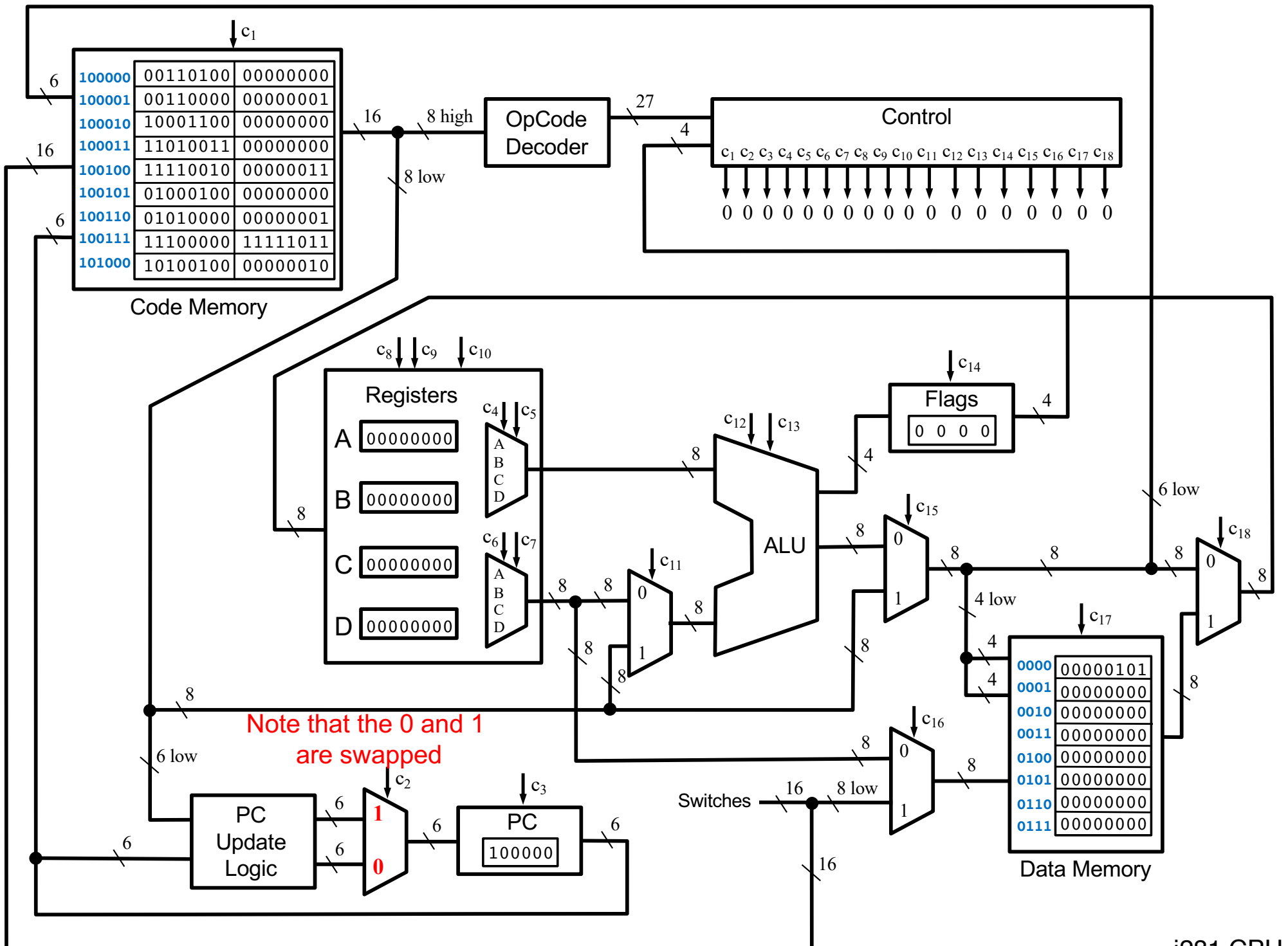






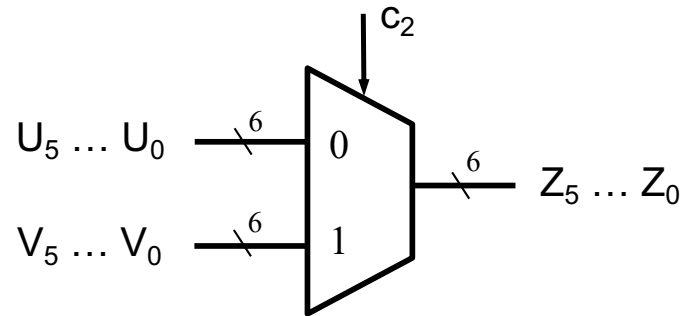




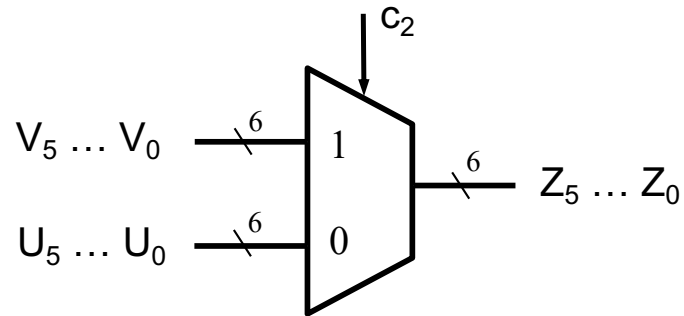




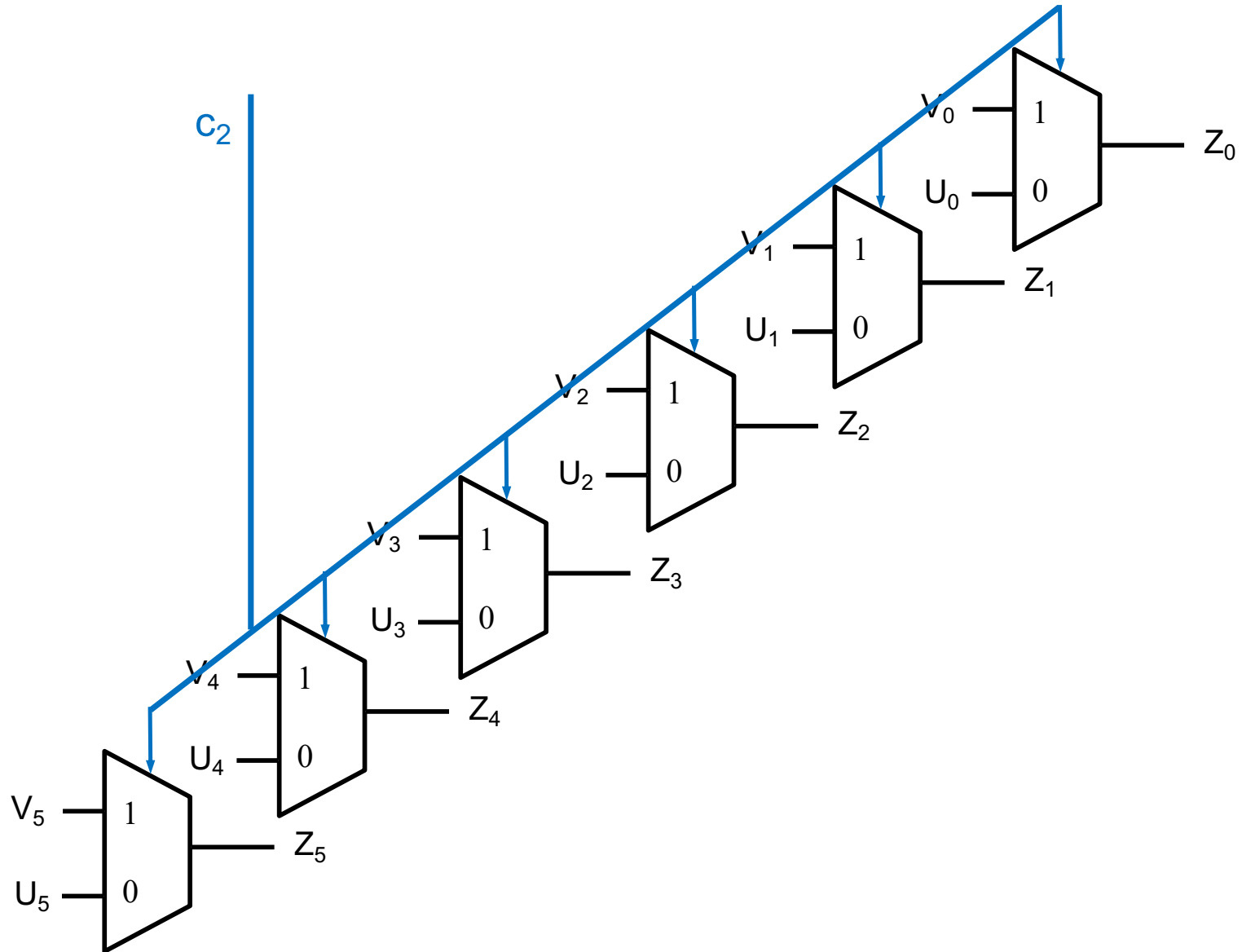
# 2-to-1 Bus Multiplexer (with **6-bit** lines)

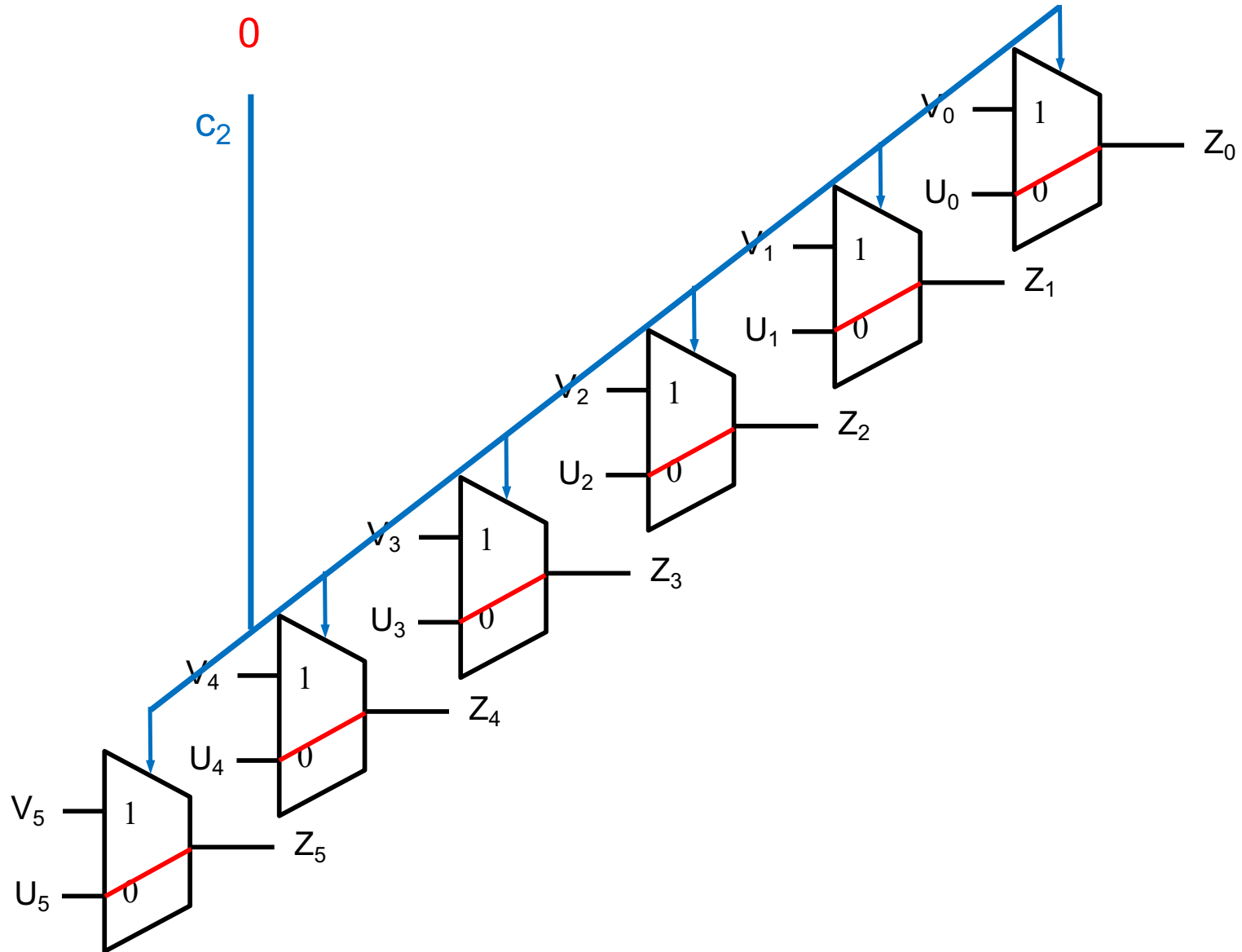


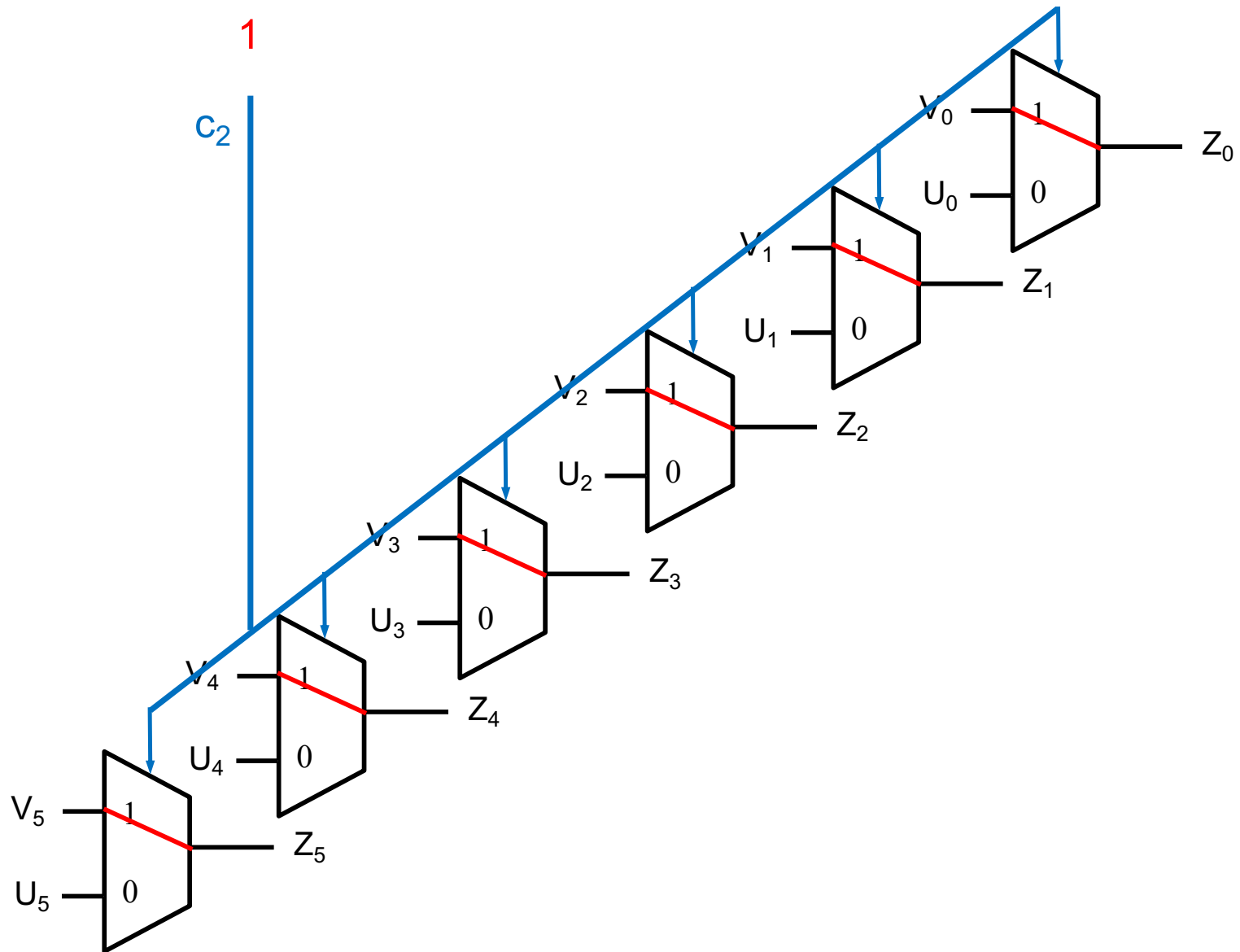
# 2-to-1 Bus Multiplexer (with **6-bit** lines)



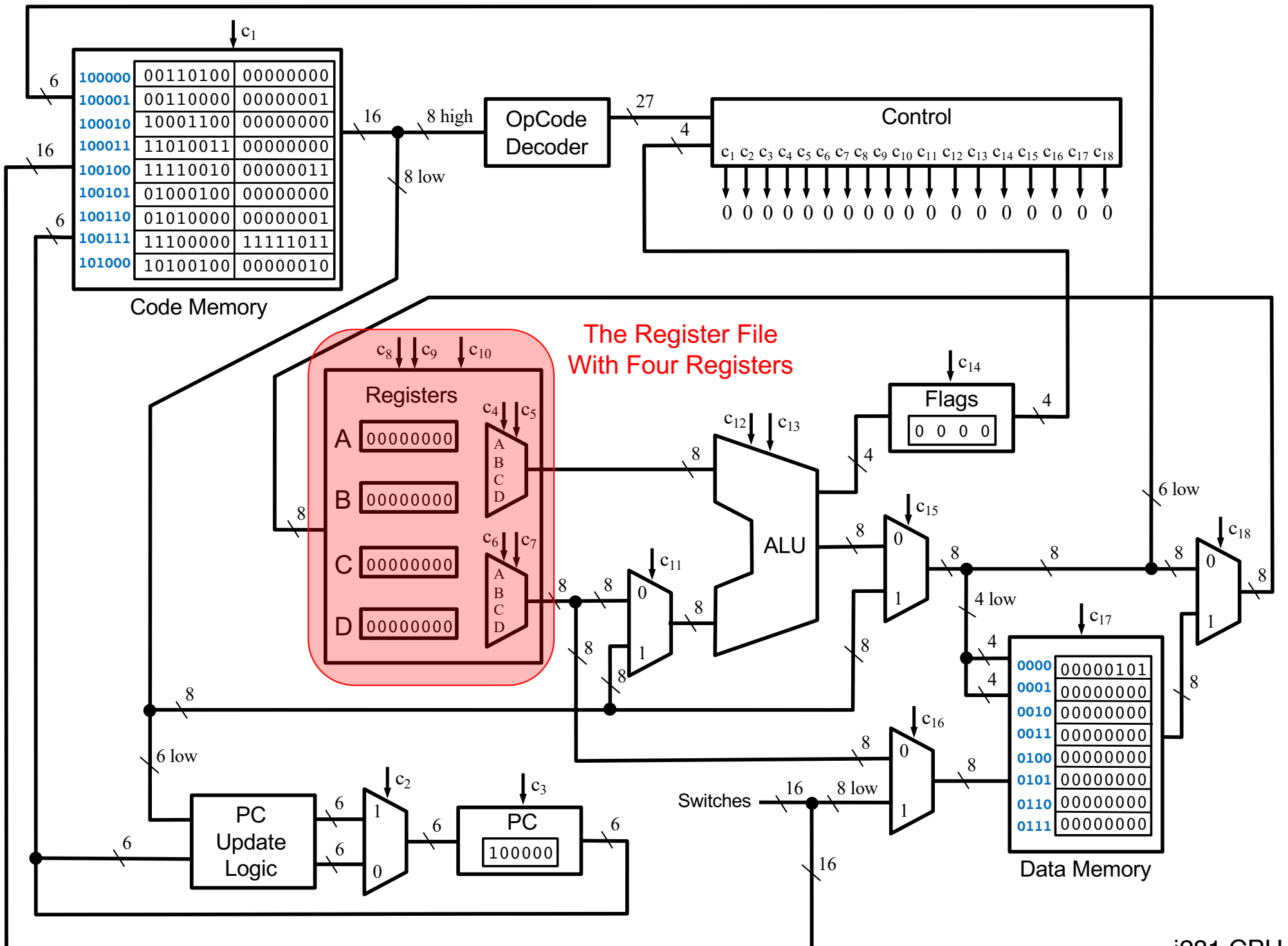
Note that the 0 and the 1 are swapped  
(in order to simplify the large diagram).

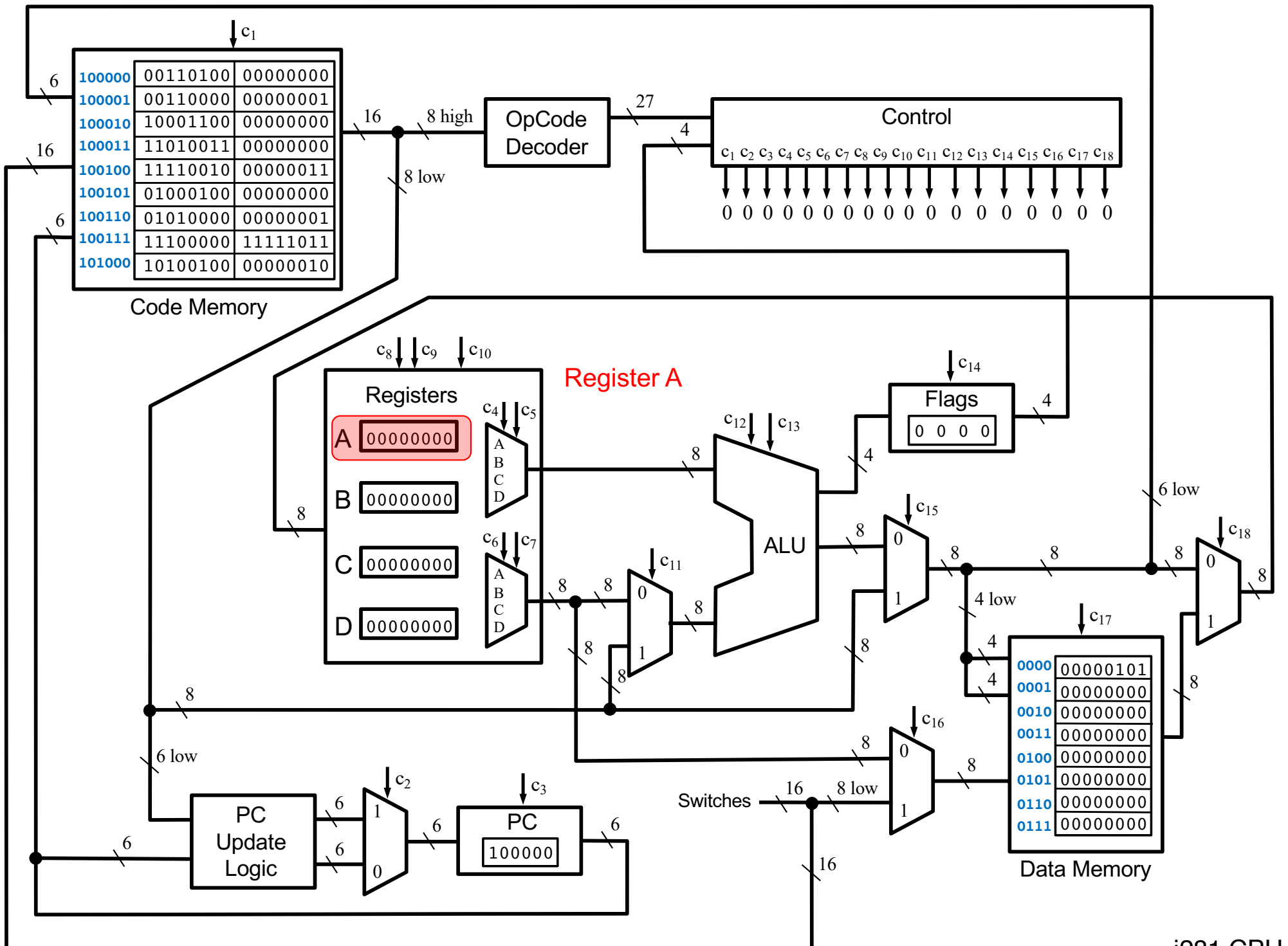




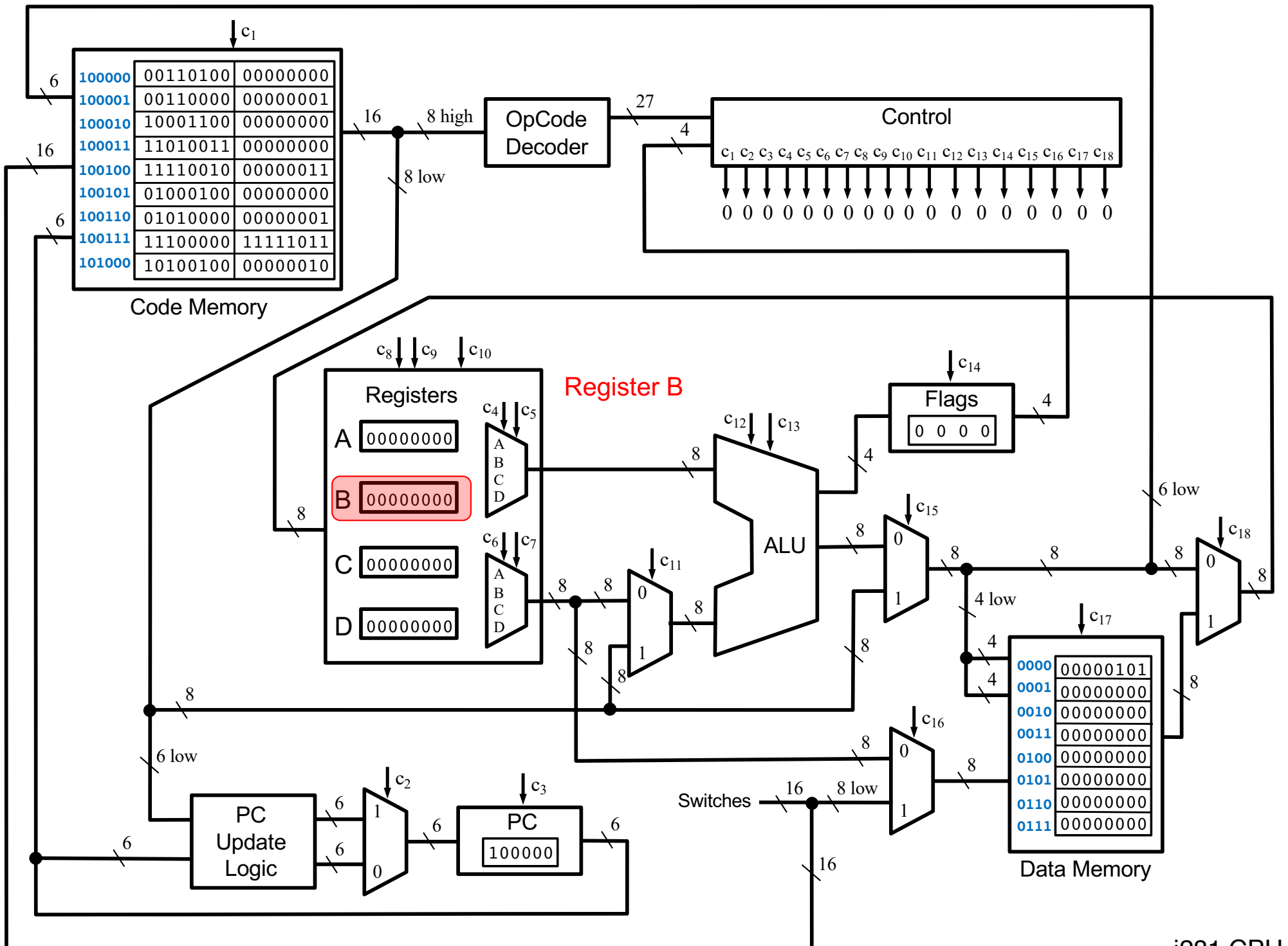


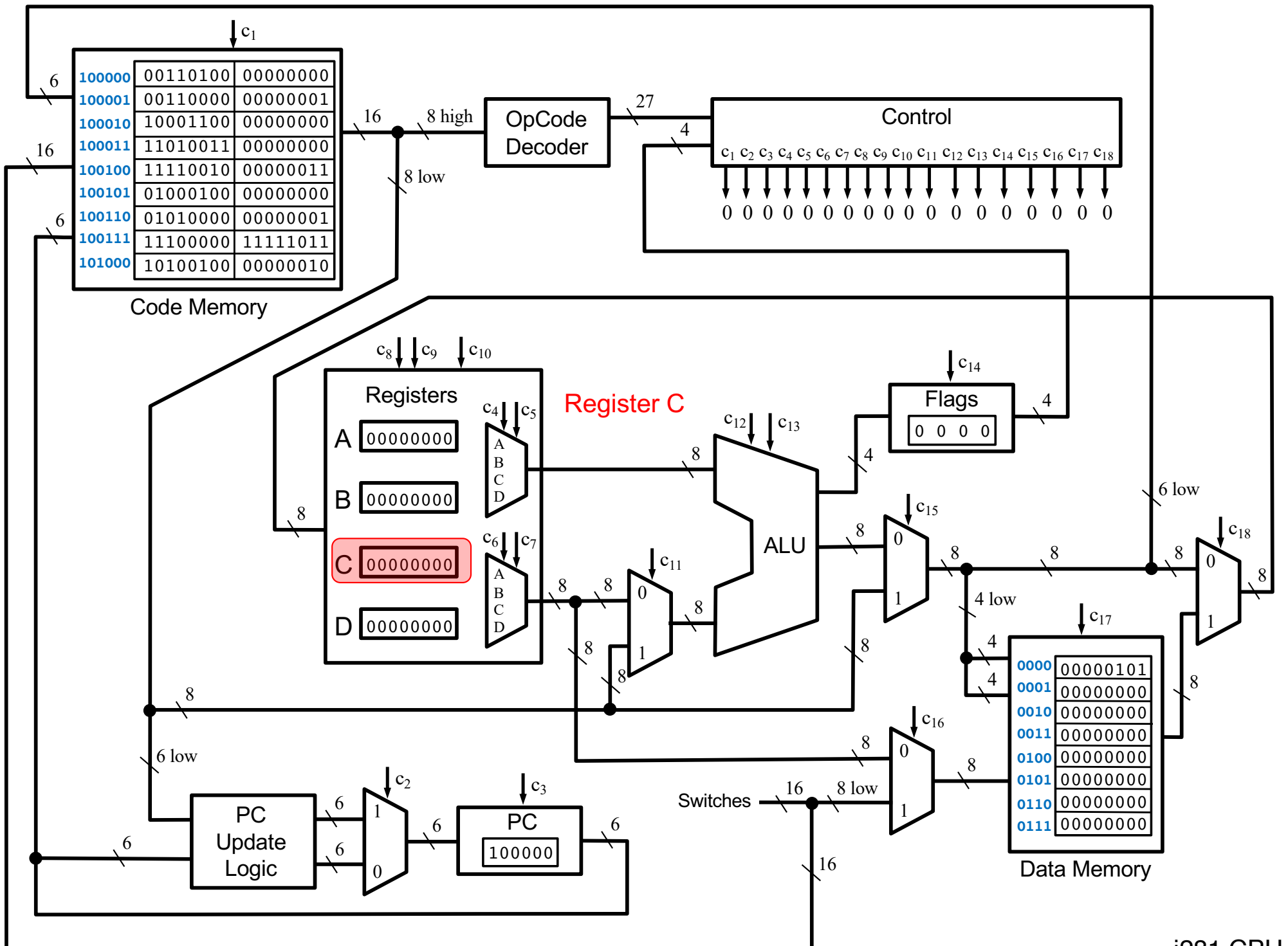
# **The Four Registers**

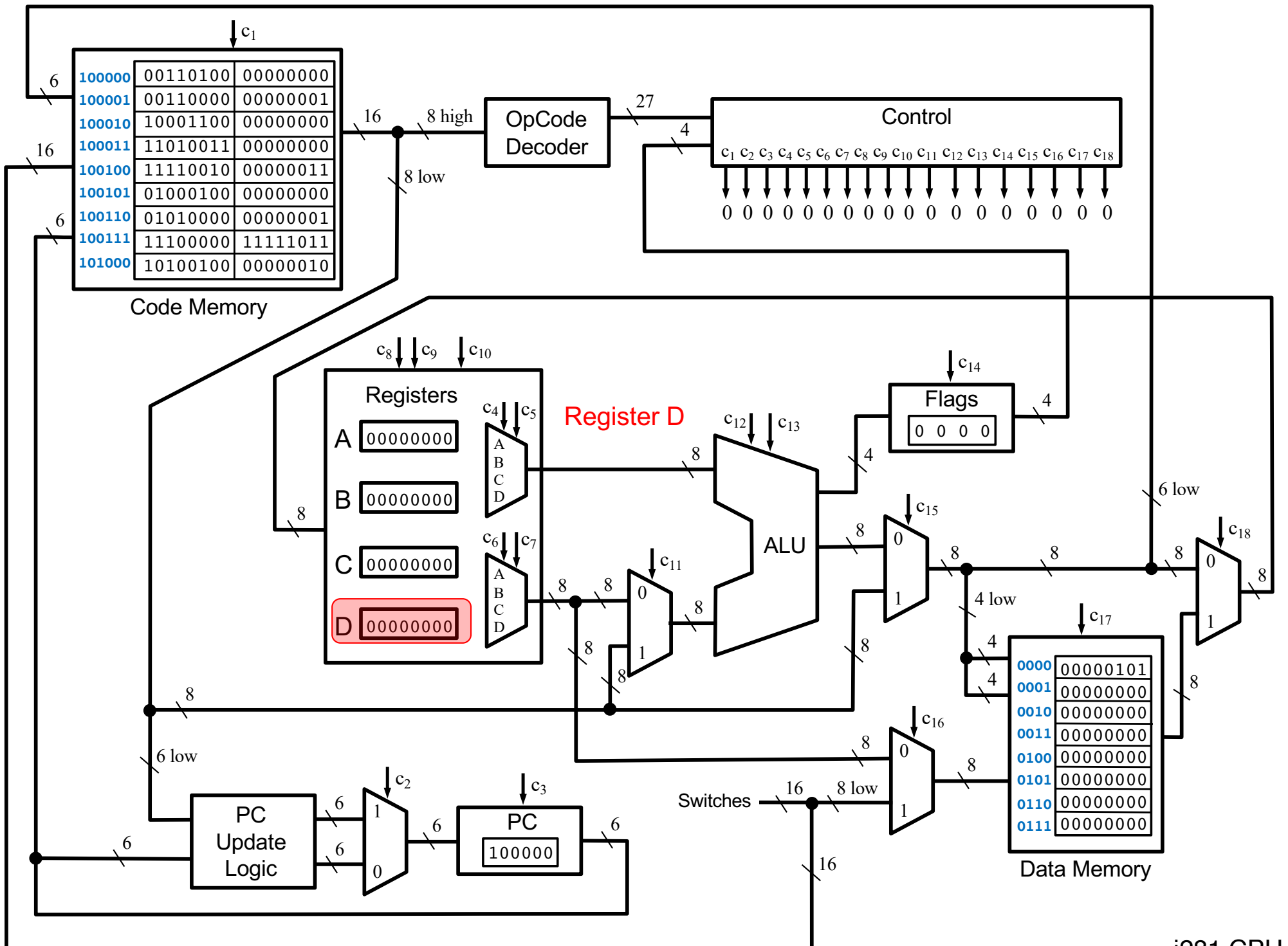




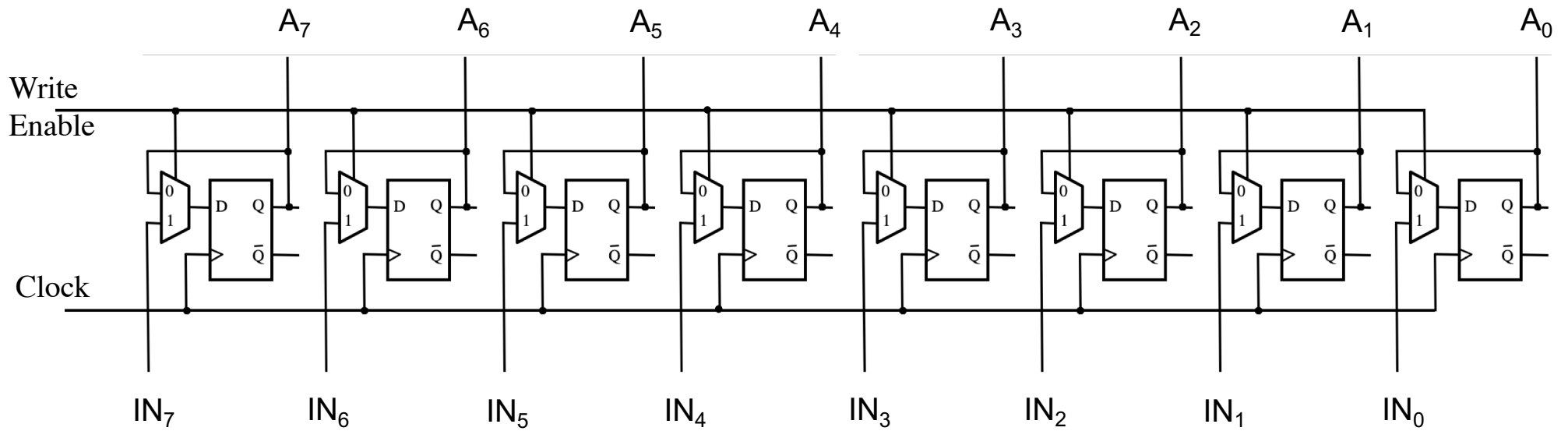






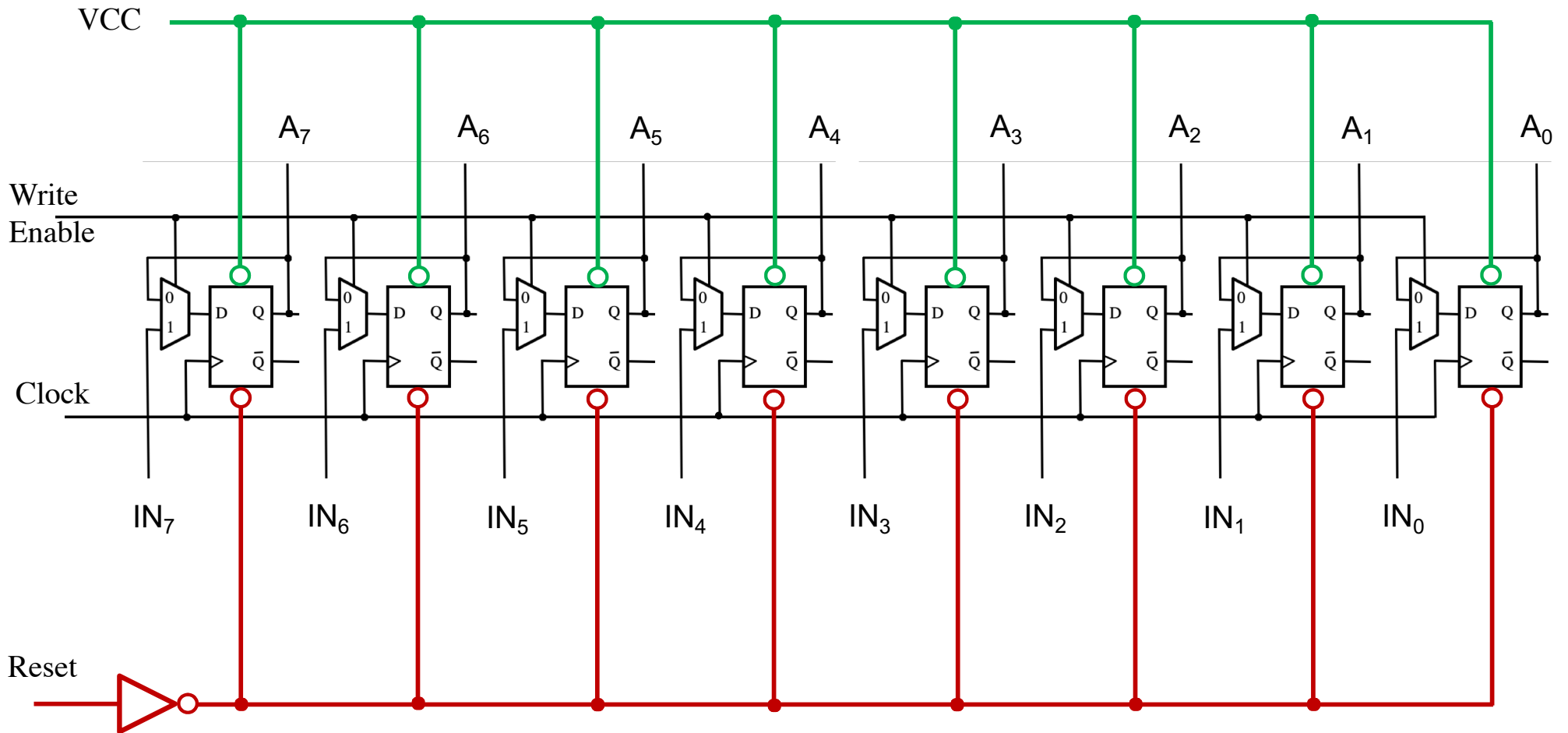


# Register A

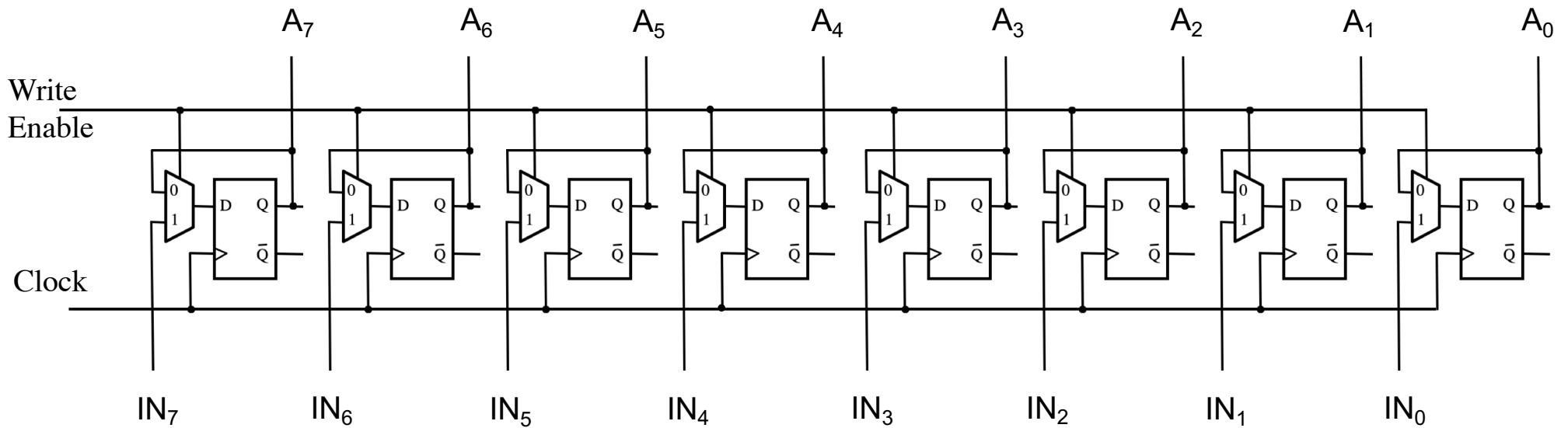


8-Bit Parallel-Access Register

# Register A

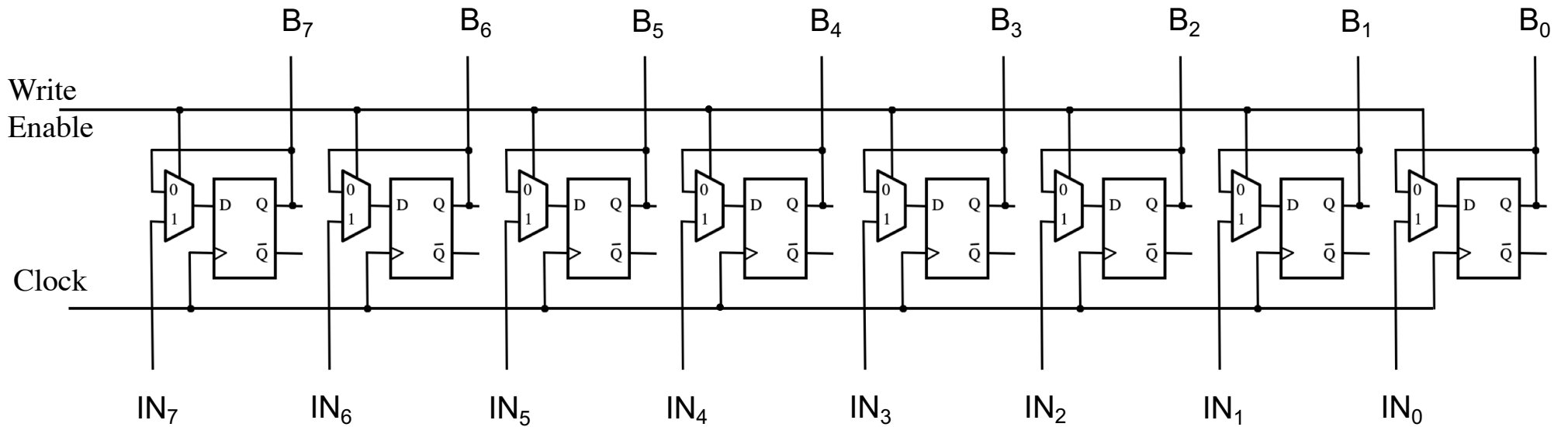


# Register A



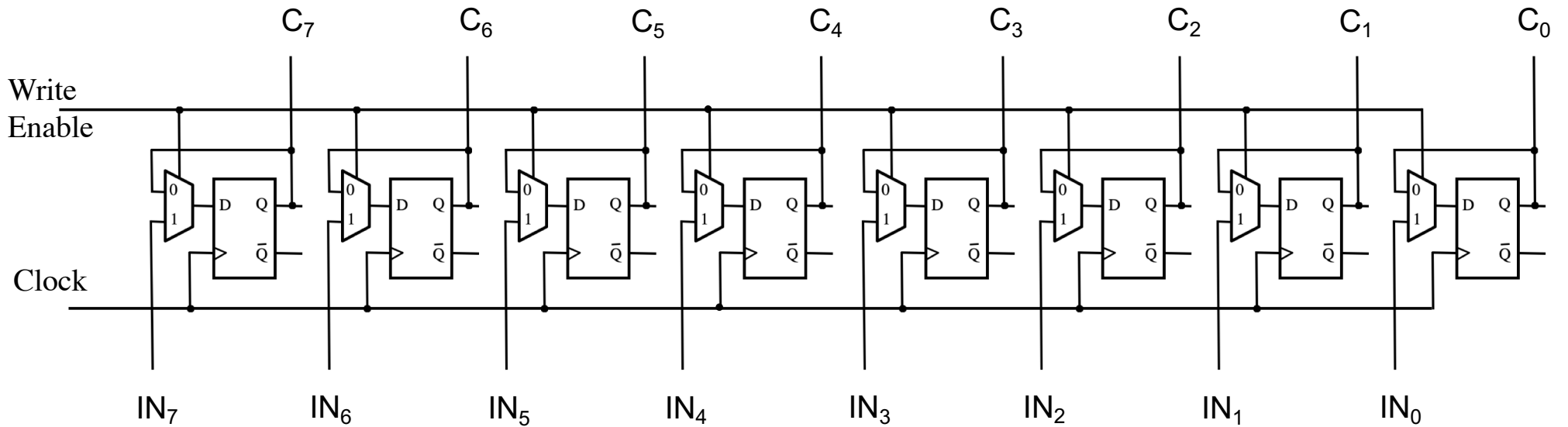
8-Bit Parallel-Access Register

# Register B



8-Bit Parallel-Access Register

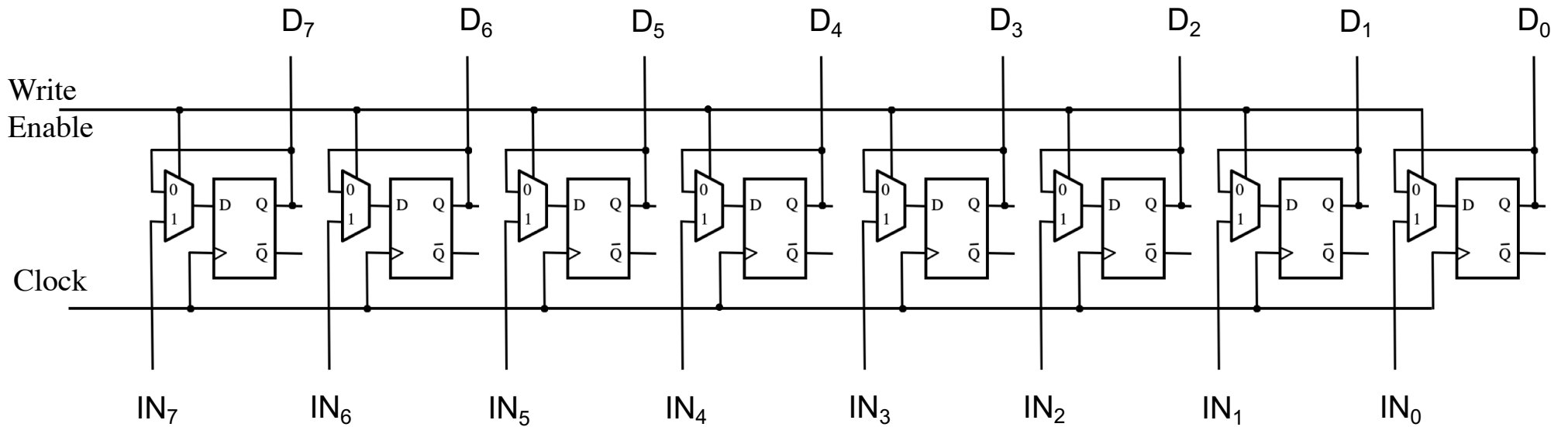
# Register C



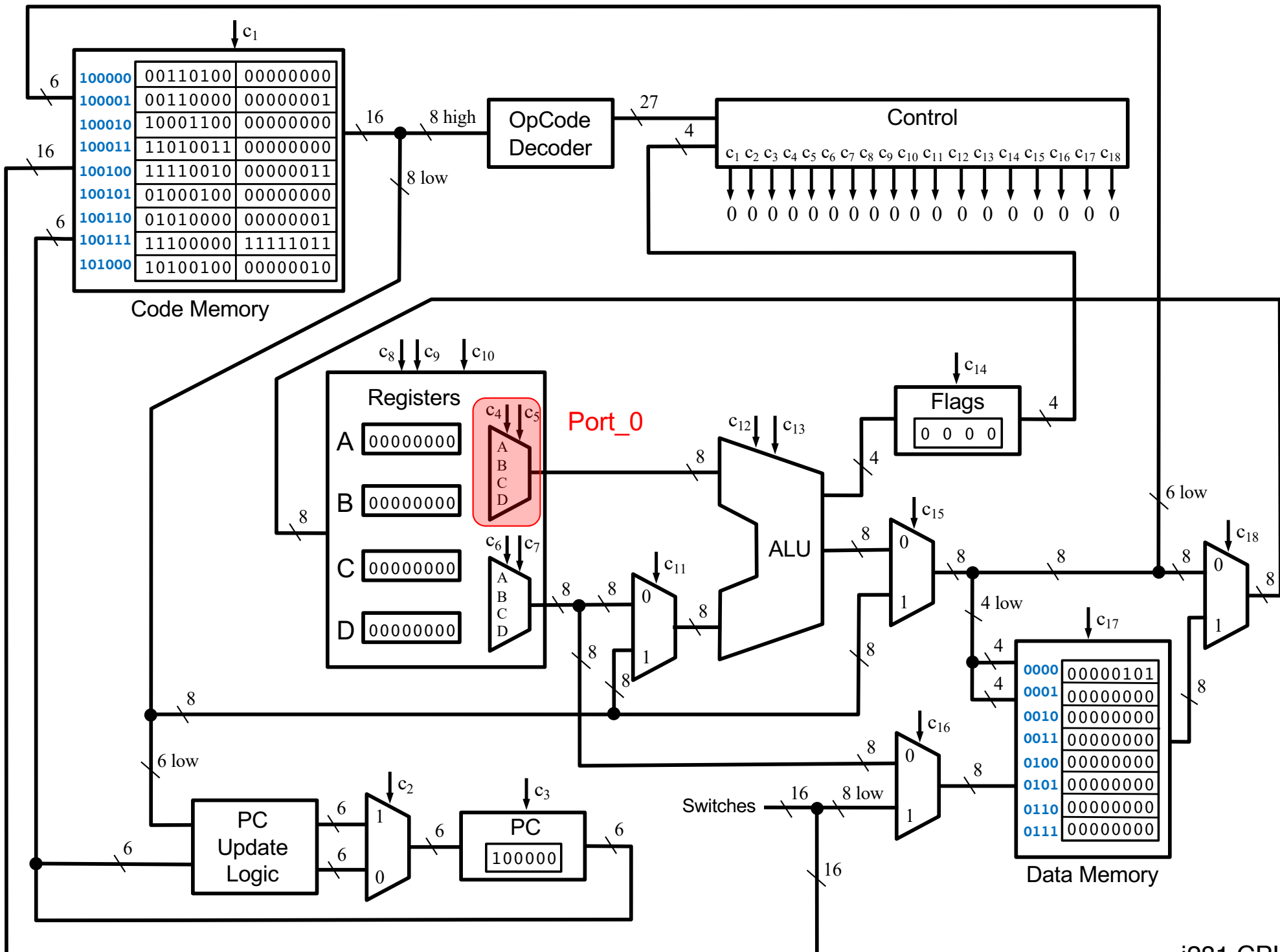
8-Bit Parallel-Access Register



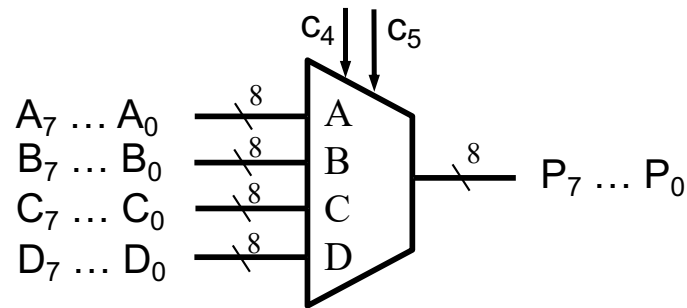
# Register D

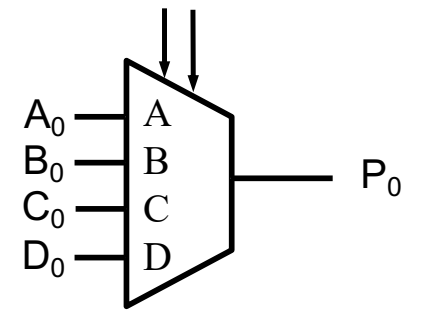


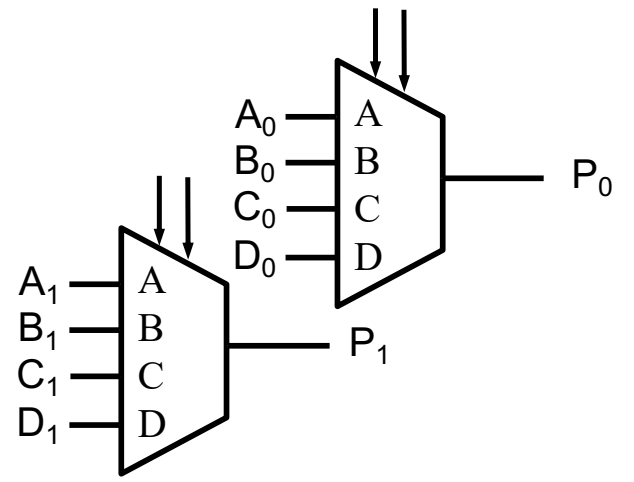
8-Bit Parallel-Access Register

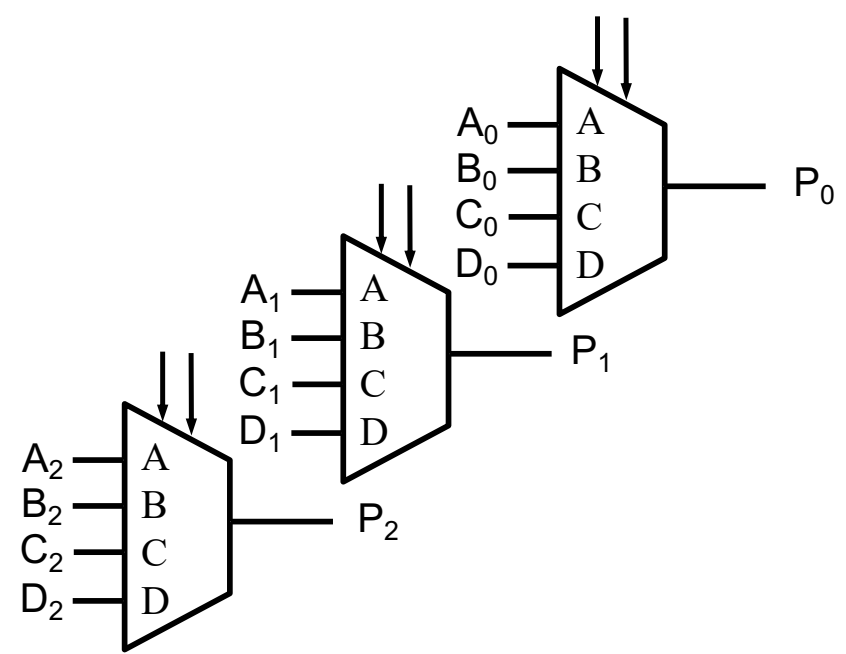


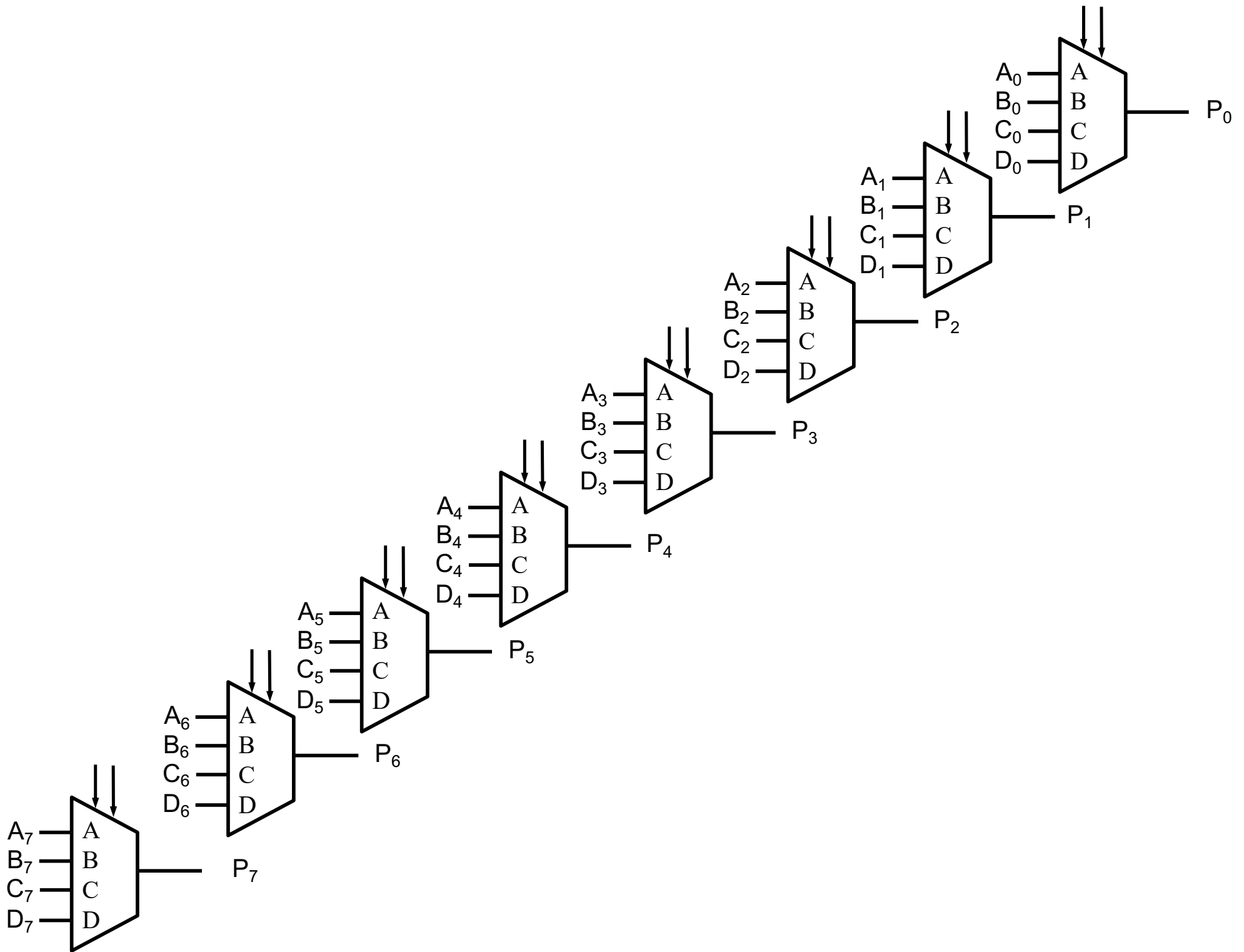
# 4-to-1 Bus Multiplexer (with 8-bit lines)

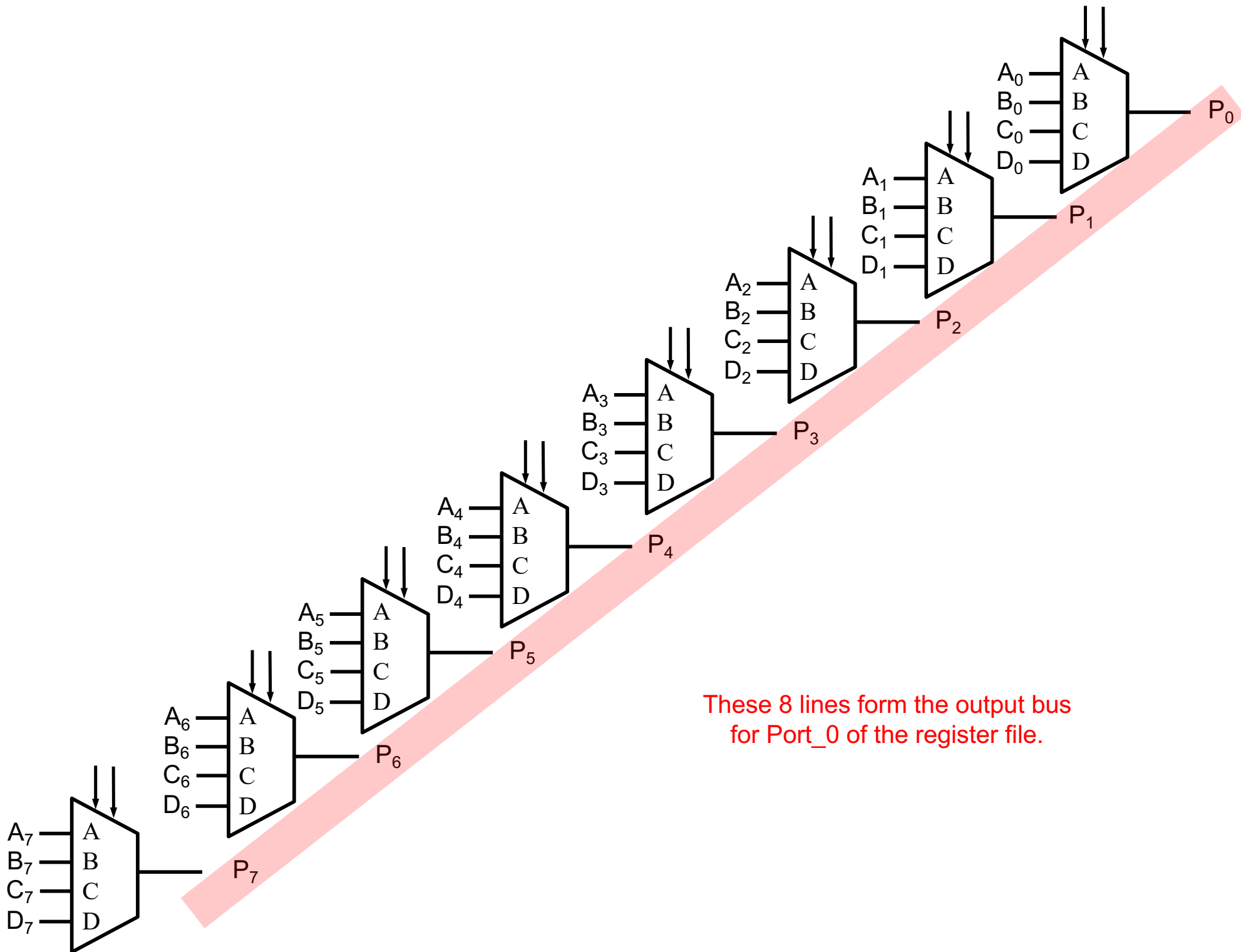






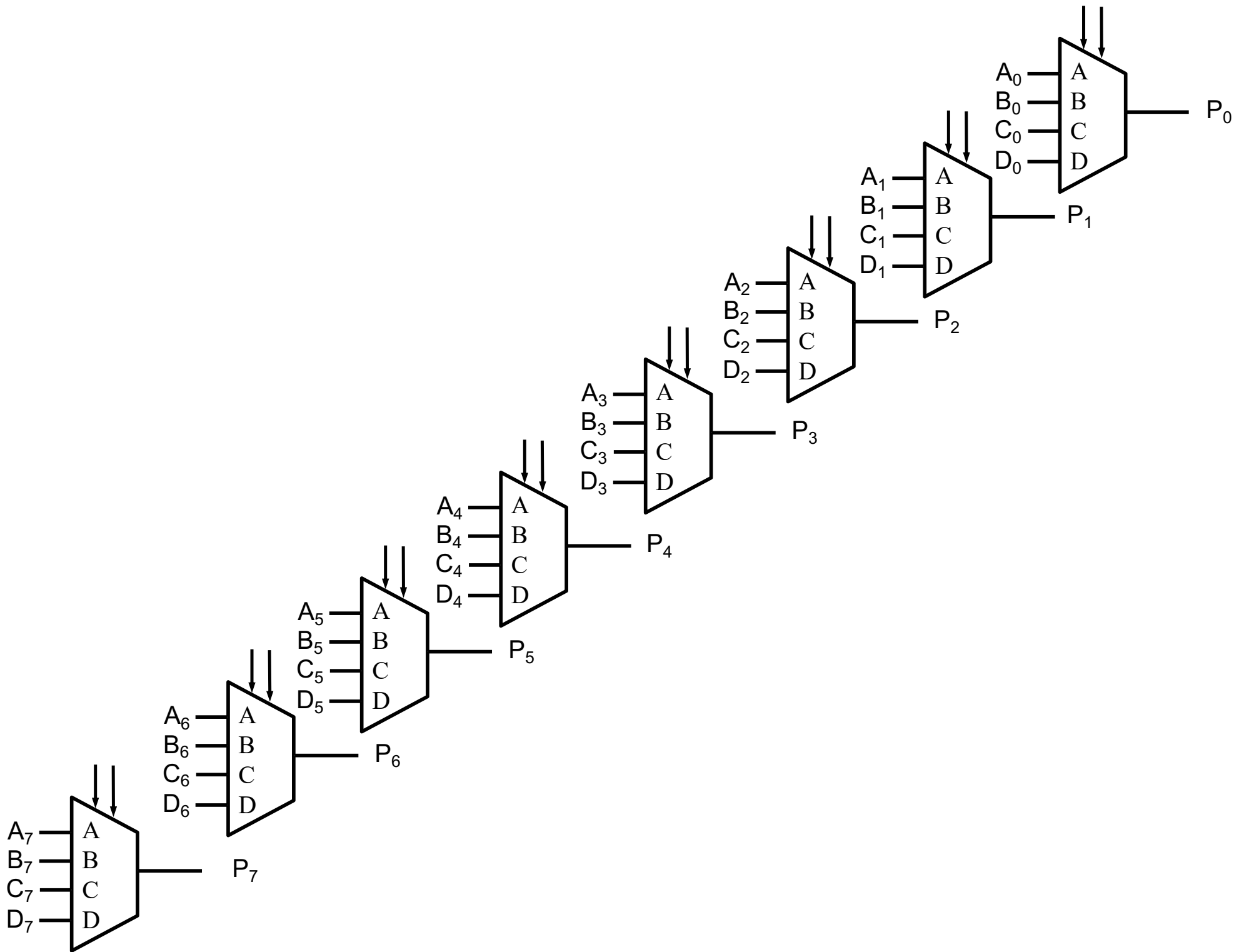


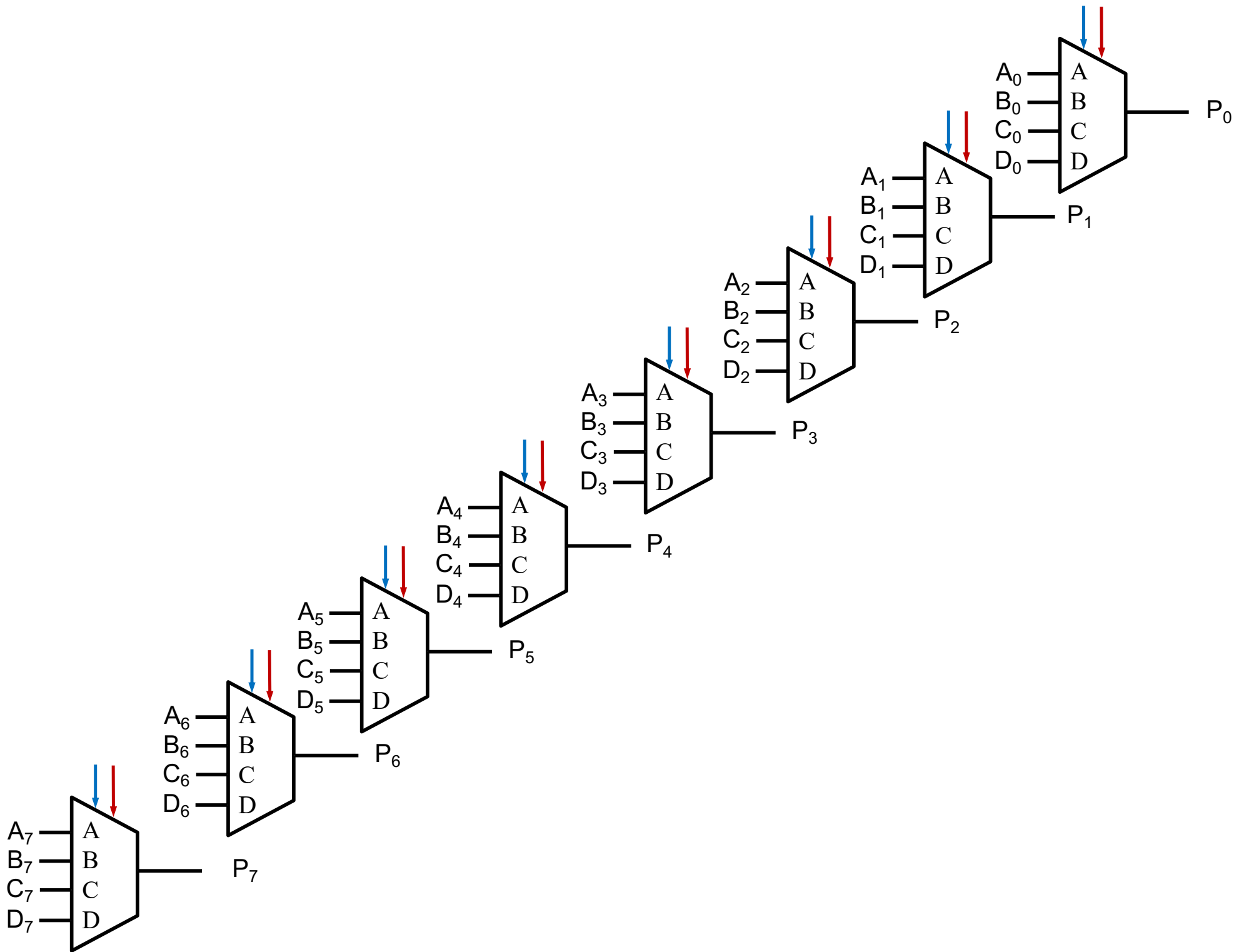


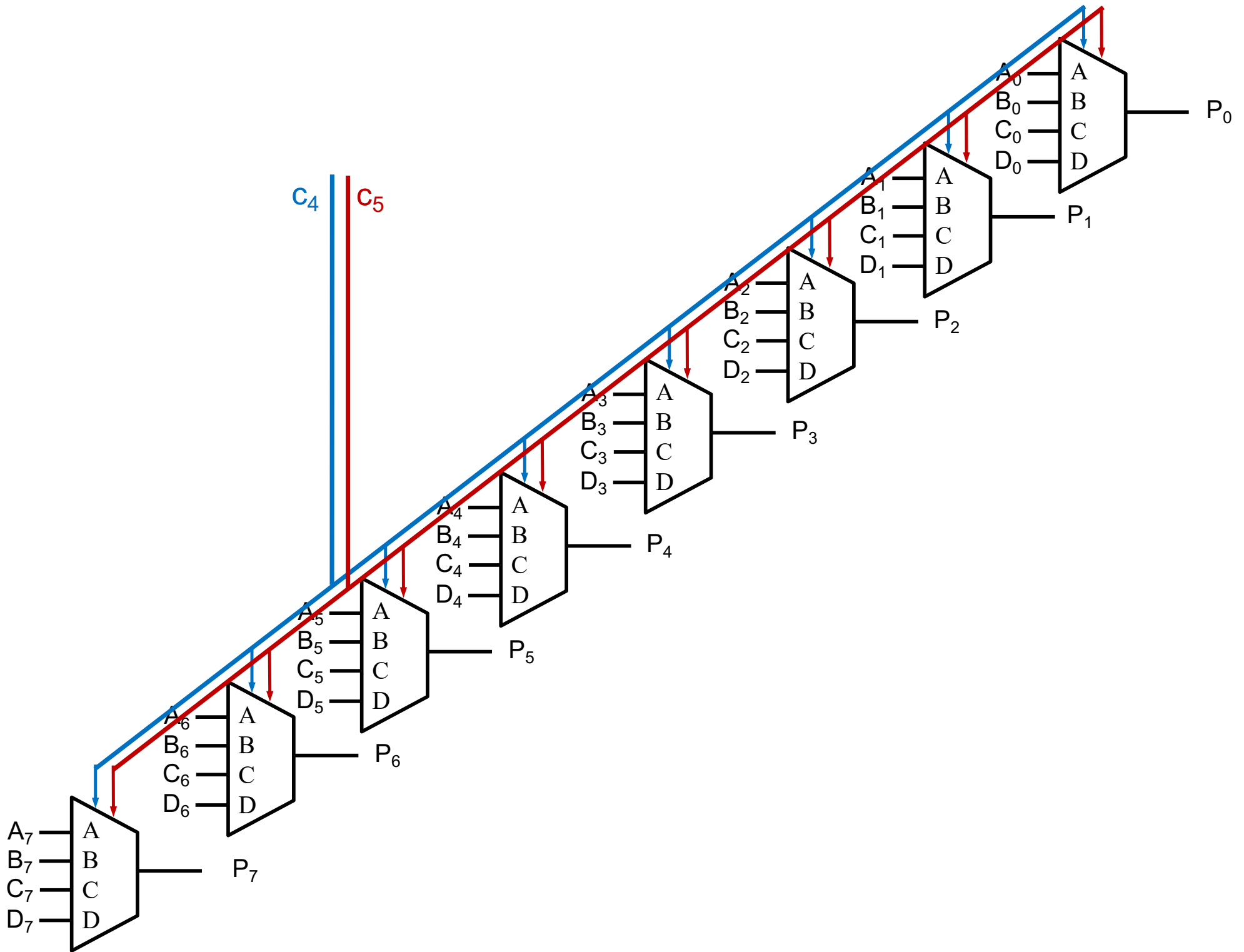


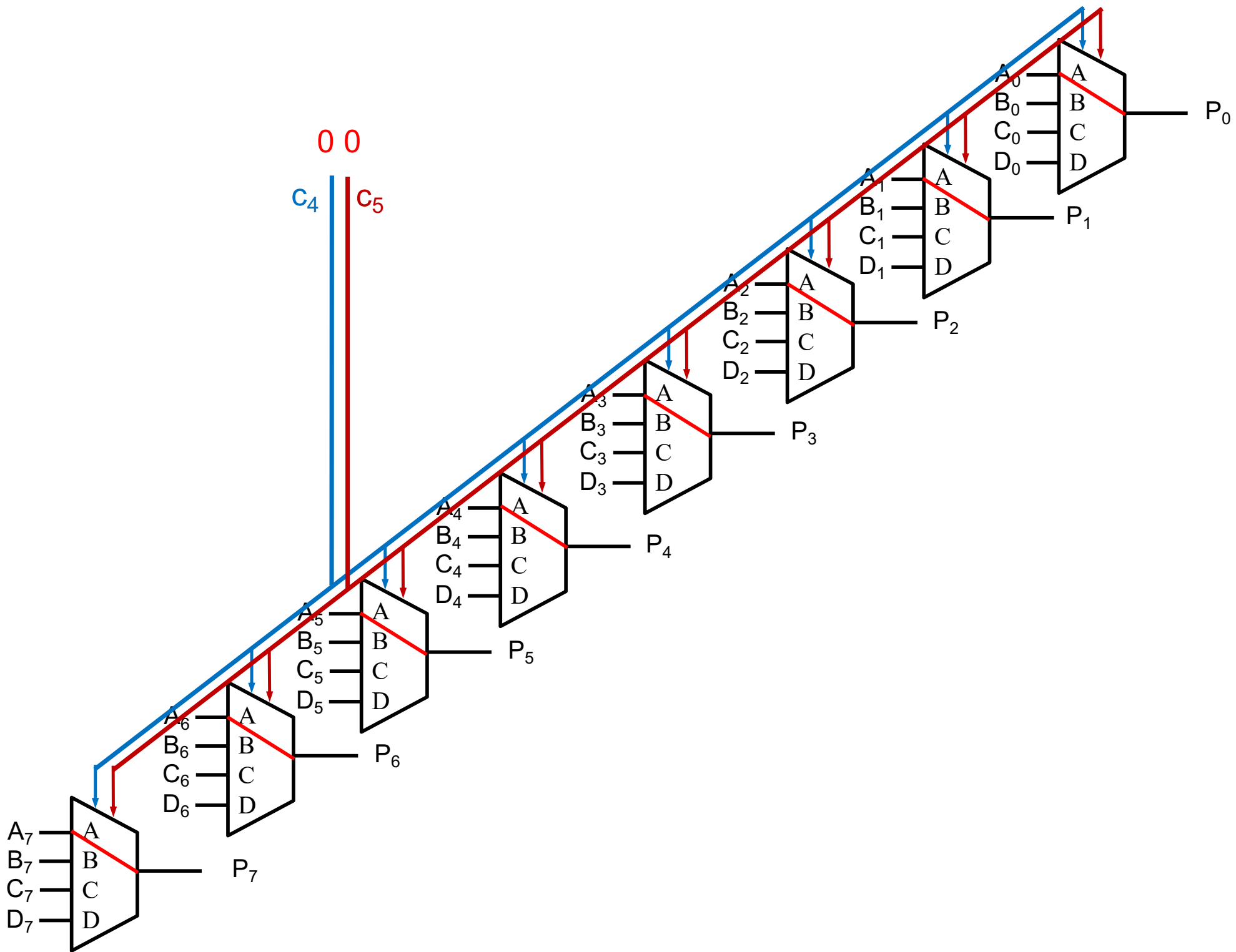
These 8 lines form the output bus for Port\_0 of the register file.



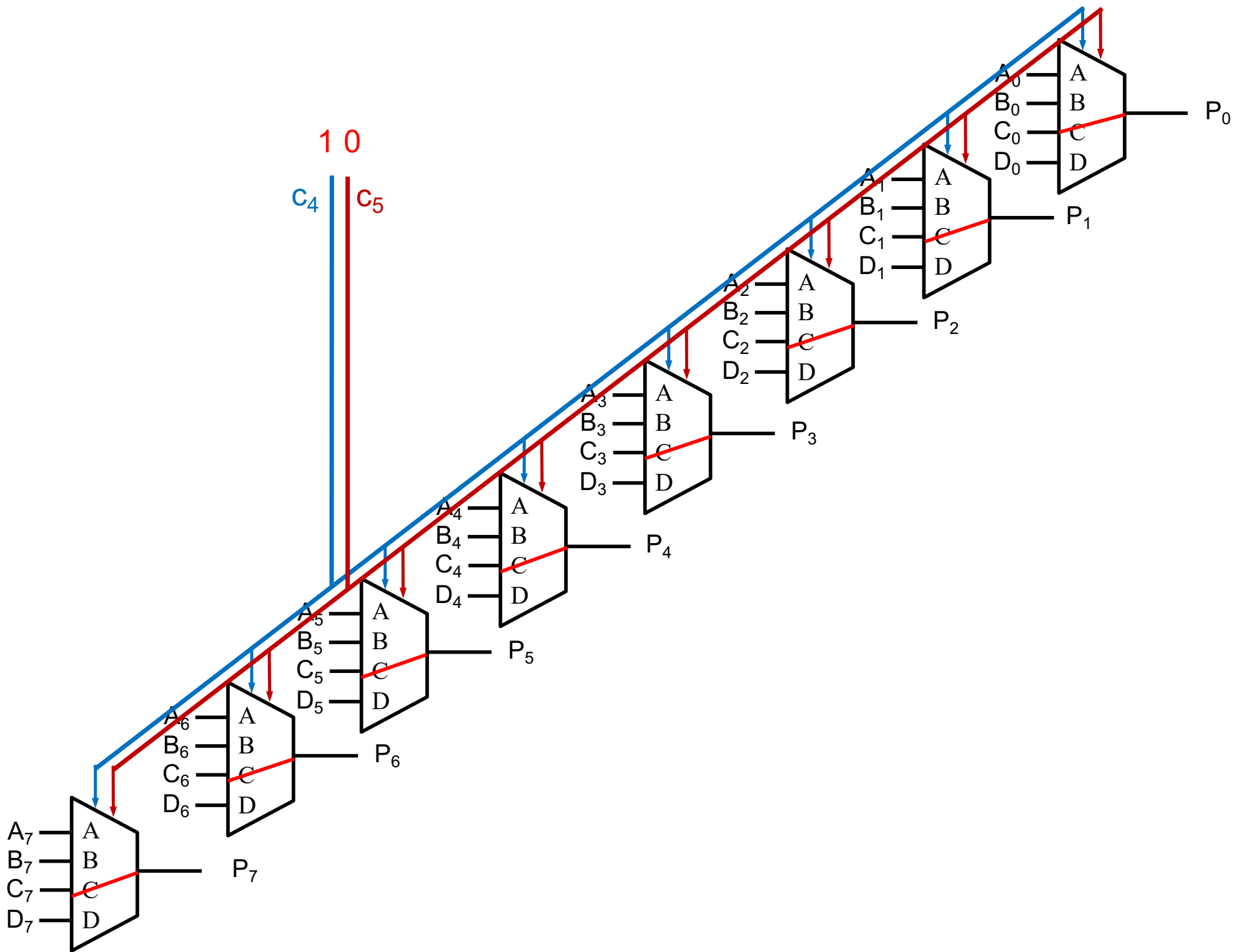


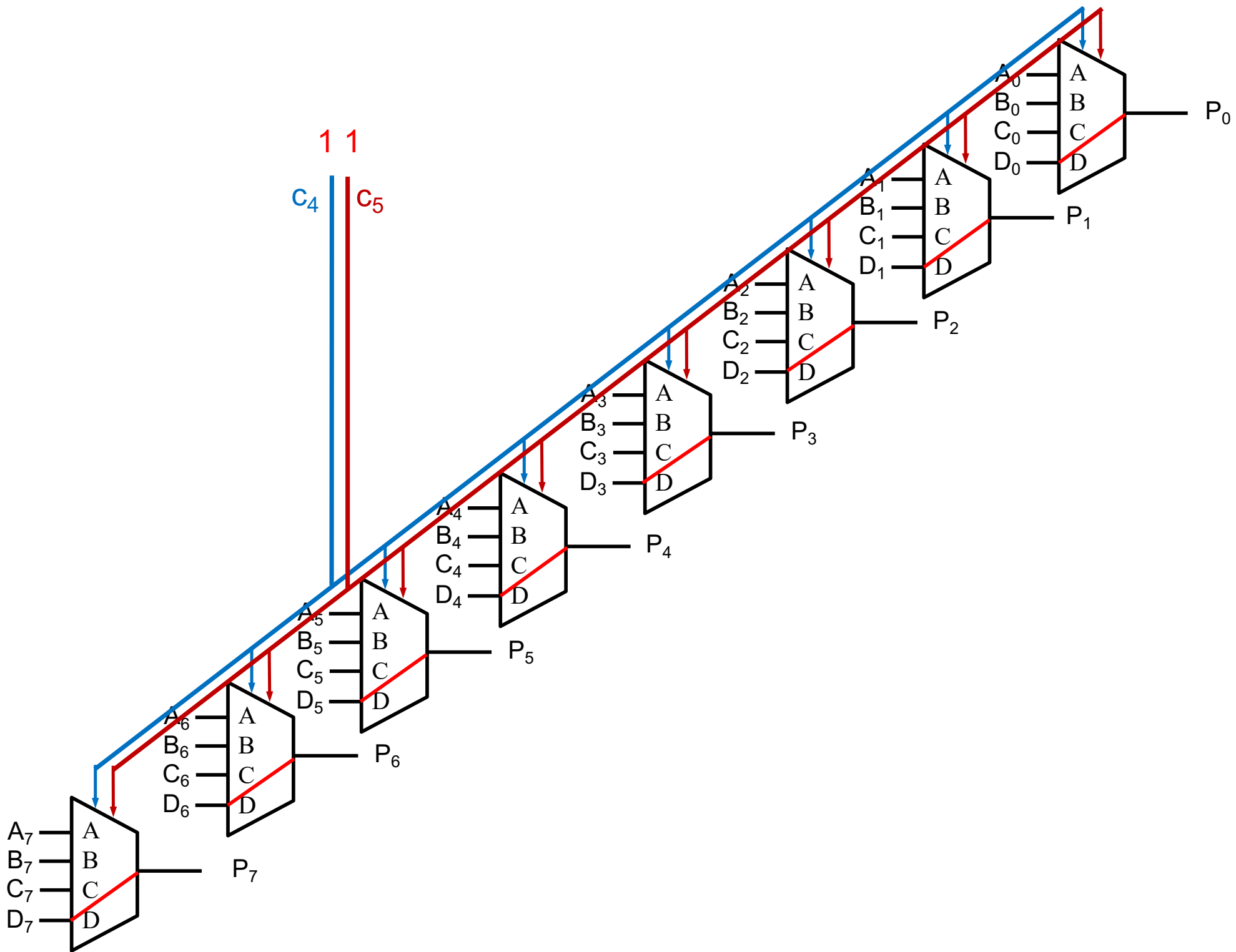


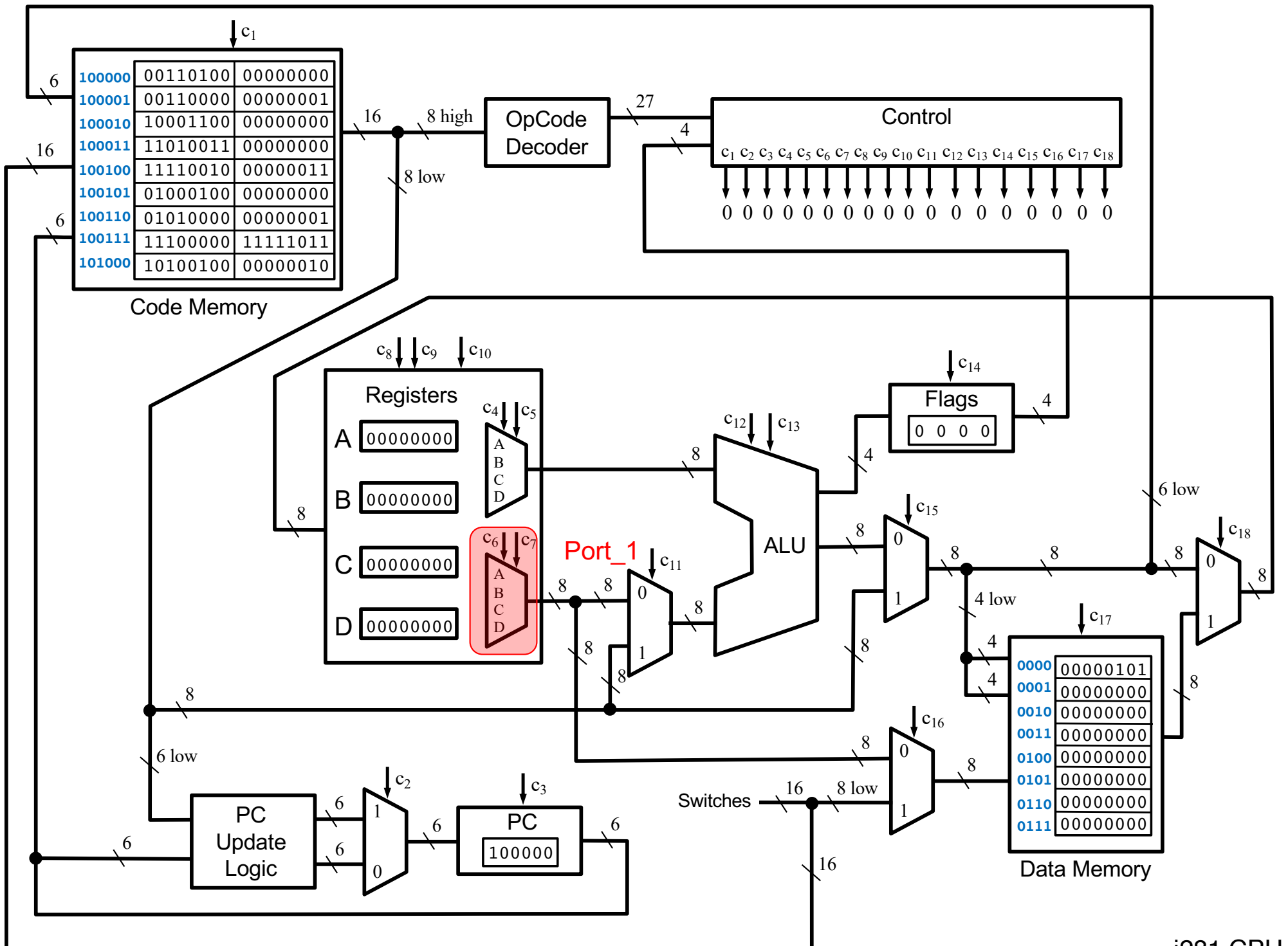






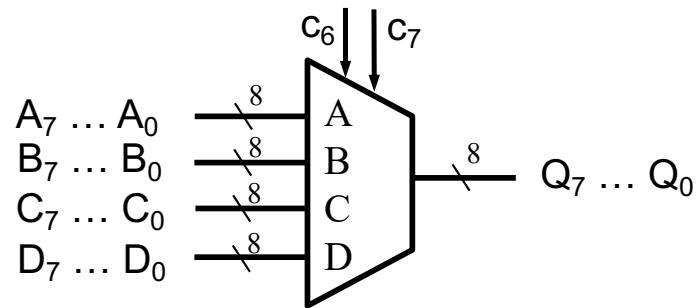


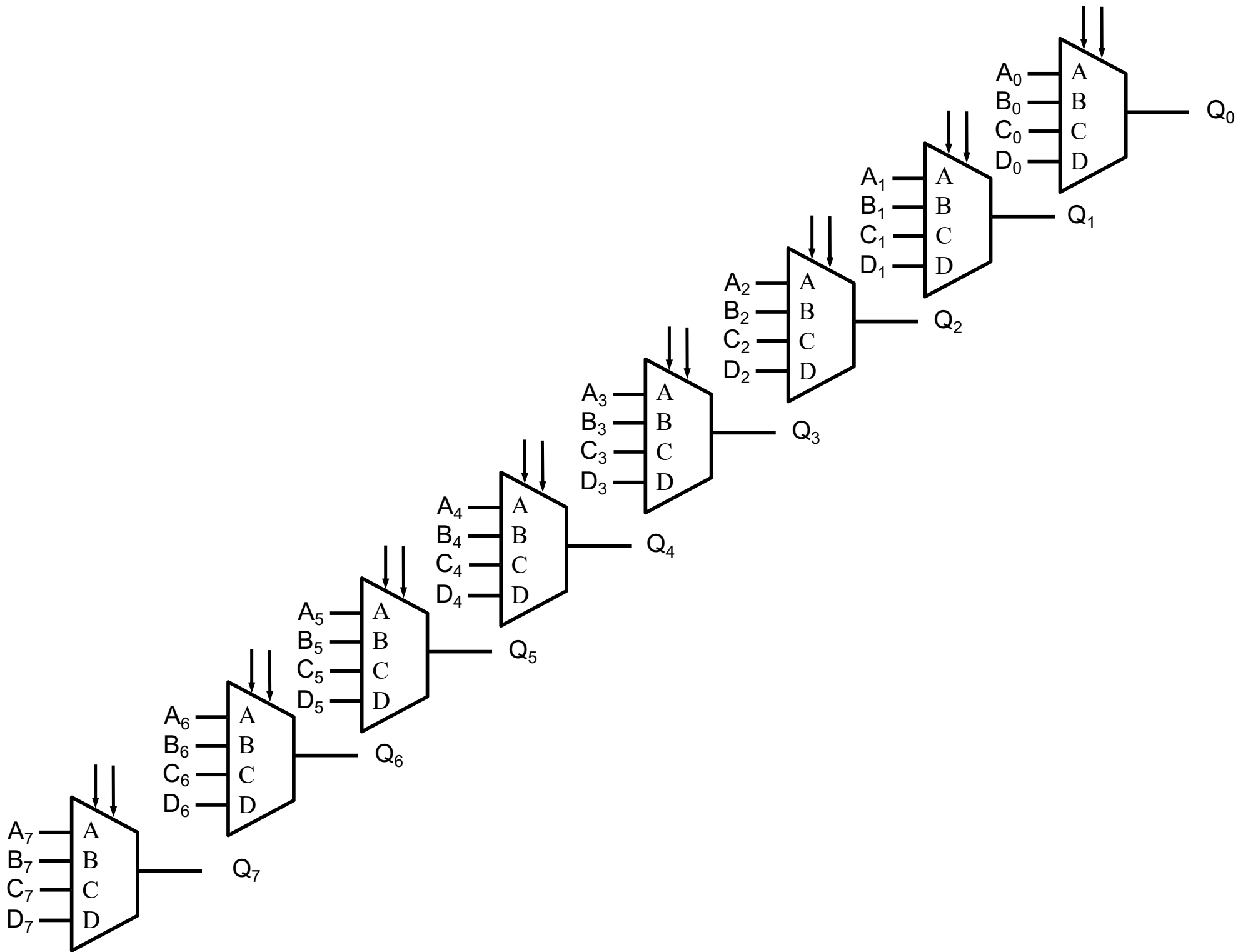


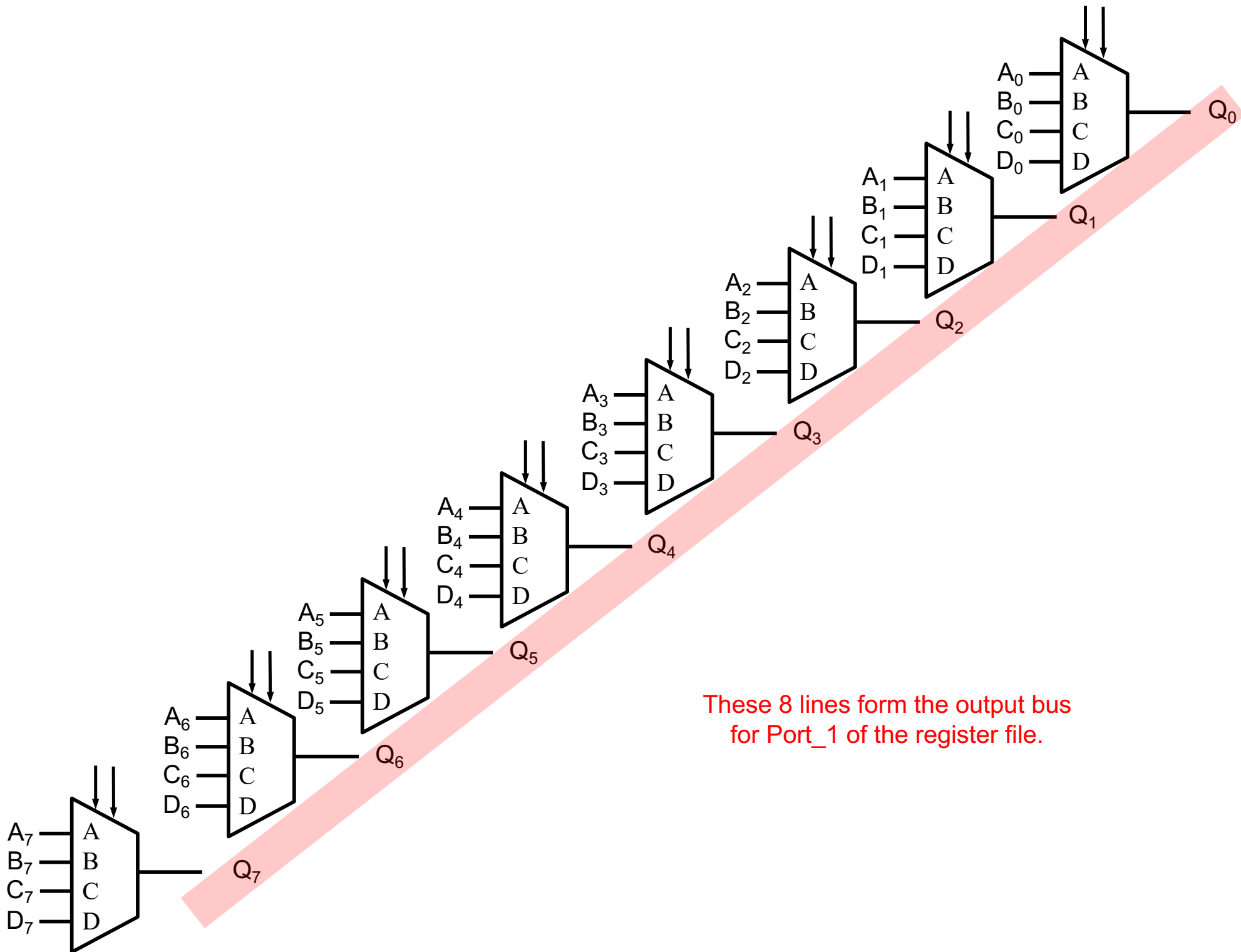


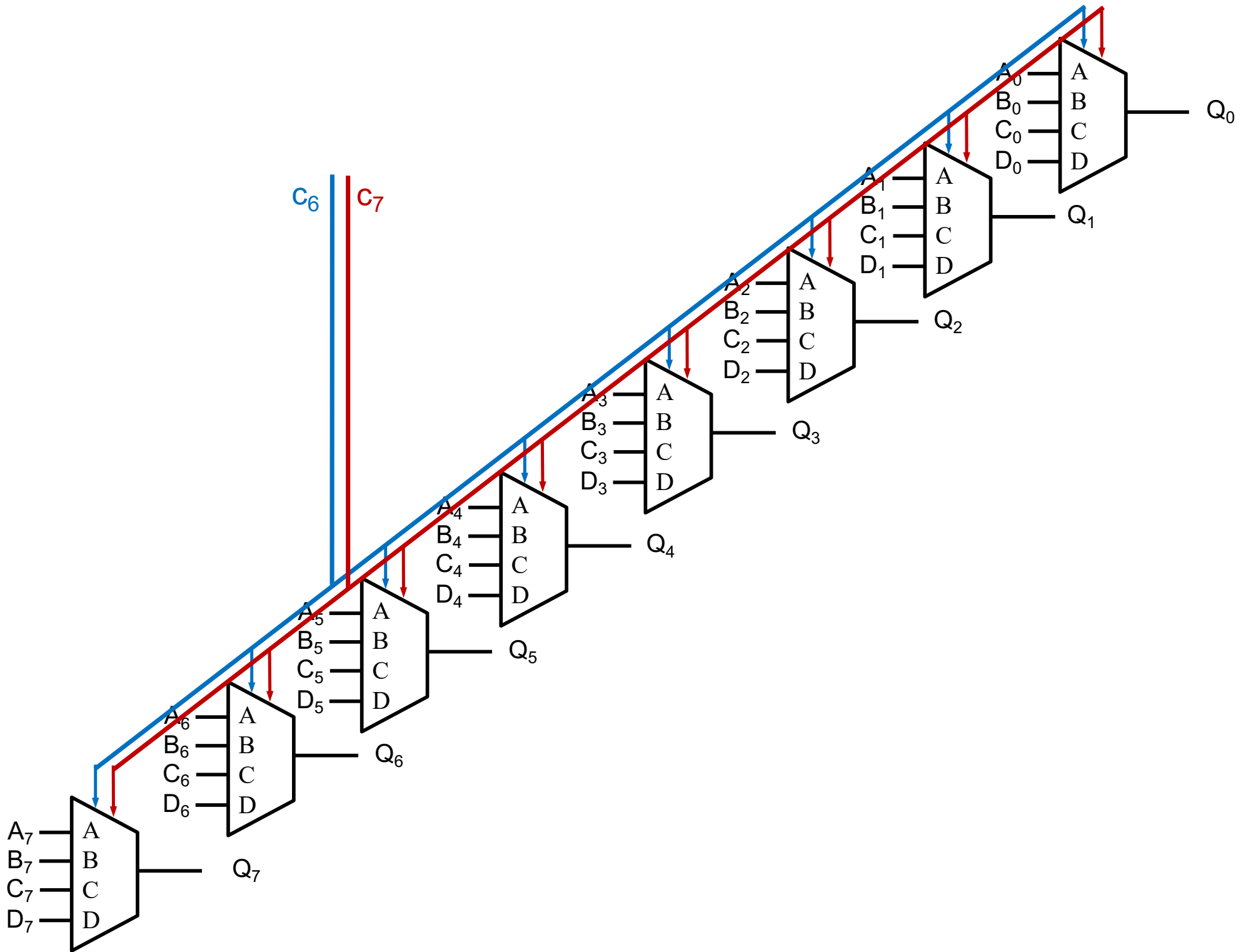


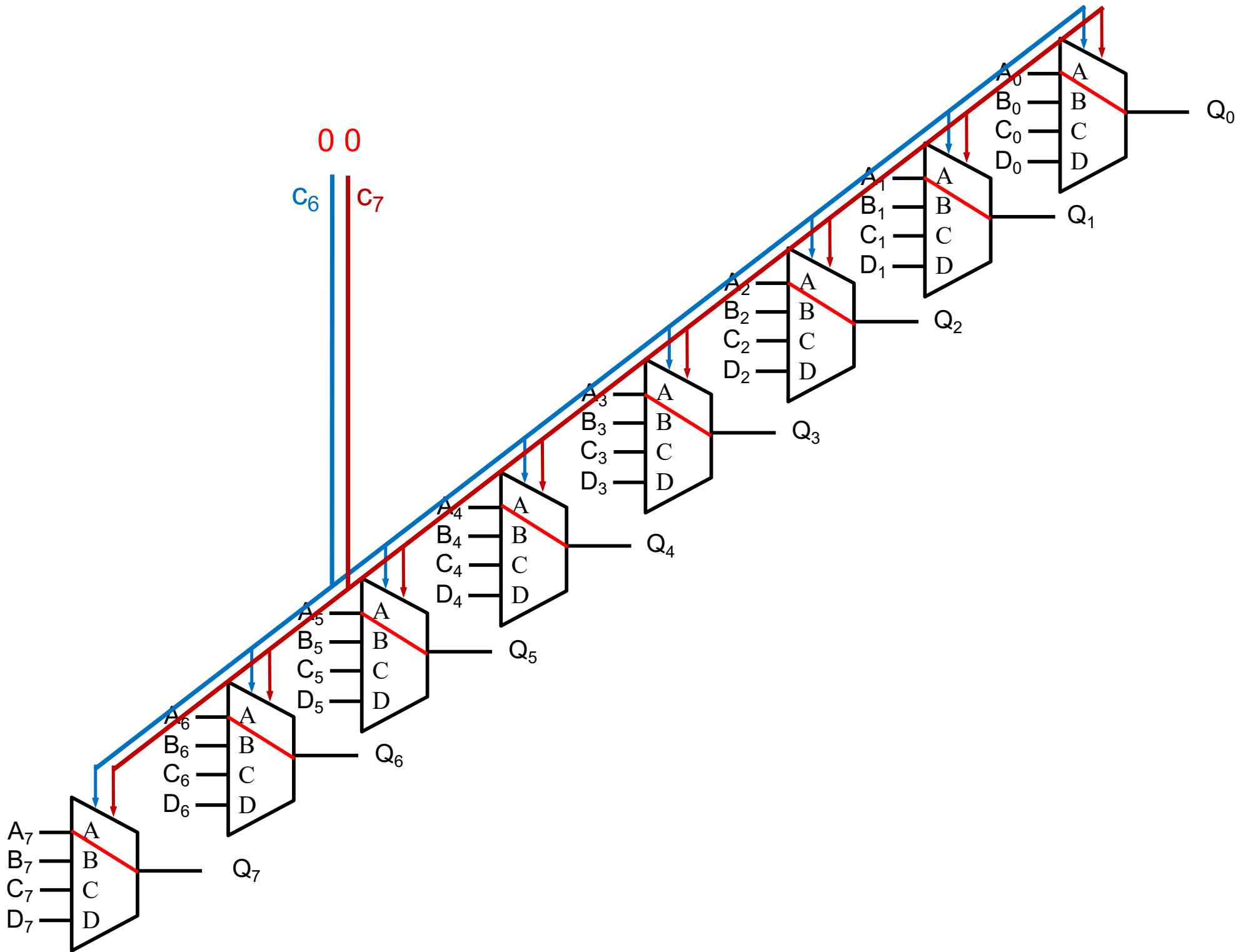
# 4-to-1 Bus Multiplexer (with 8-bit lines)

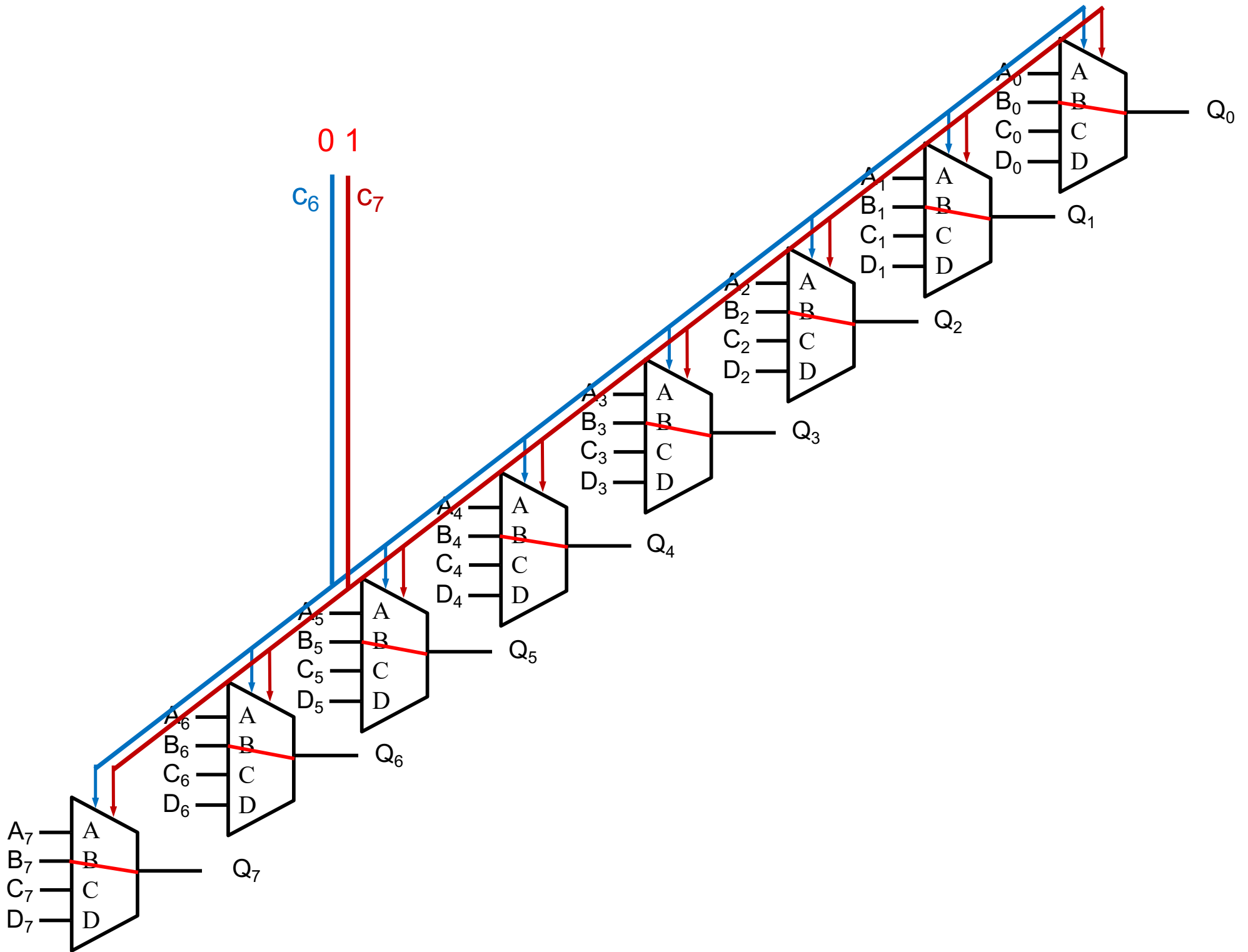


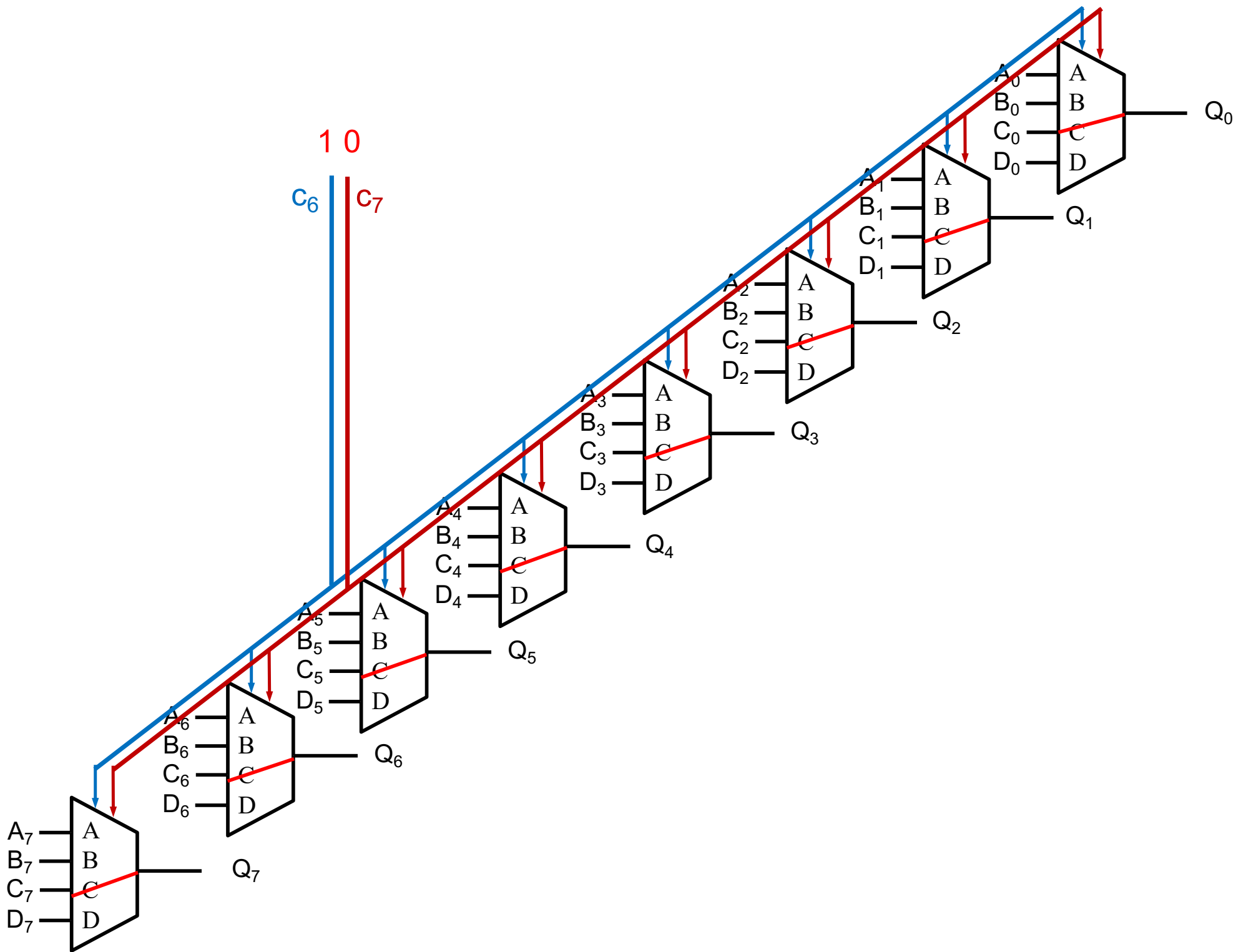


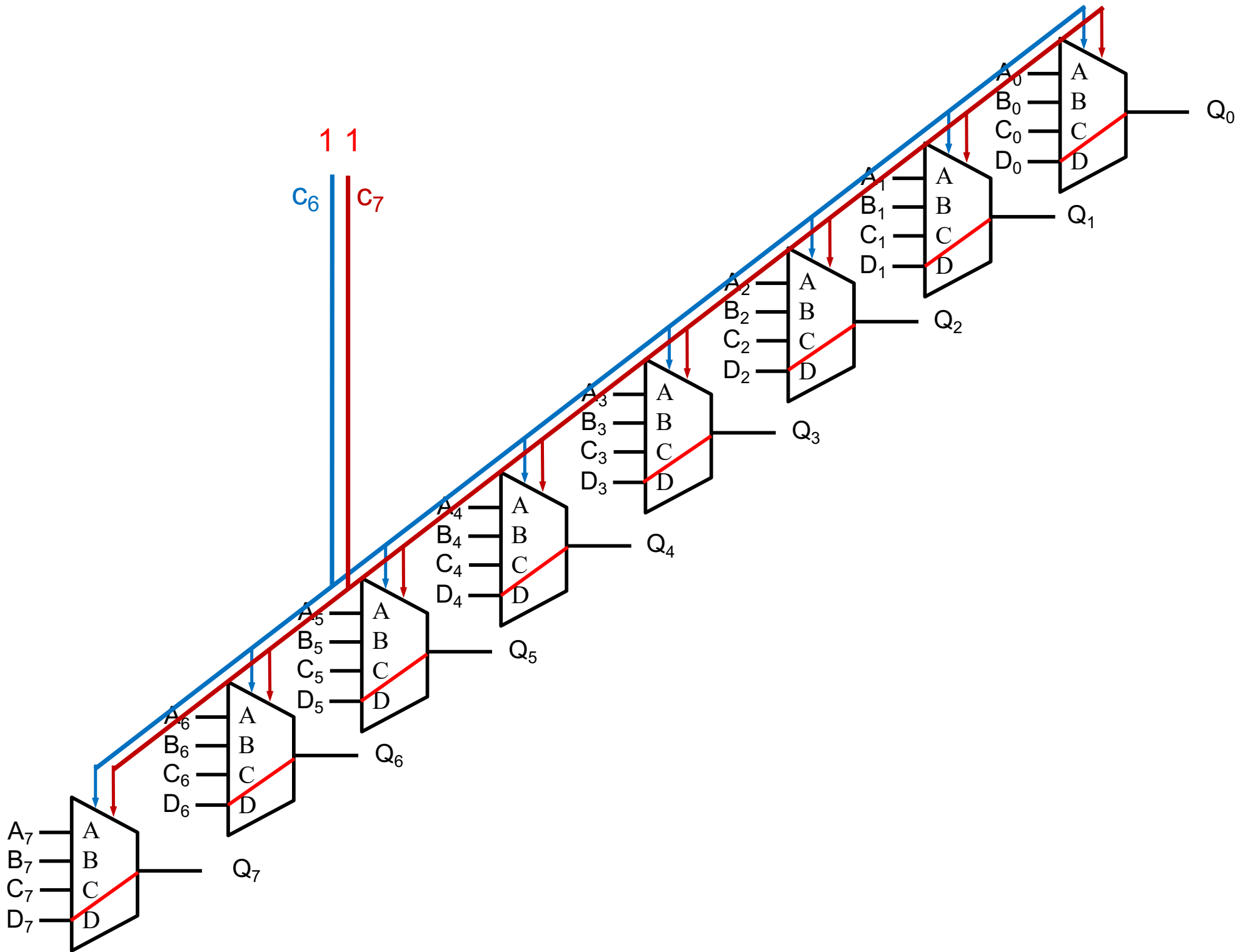




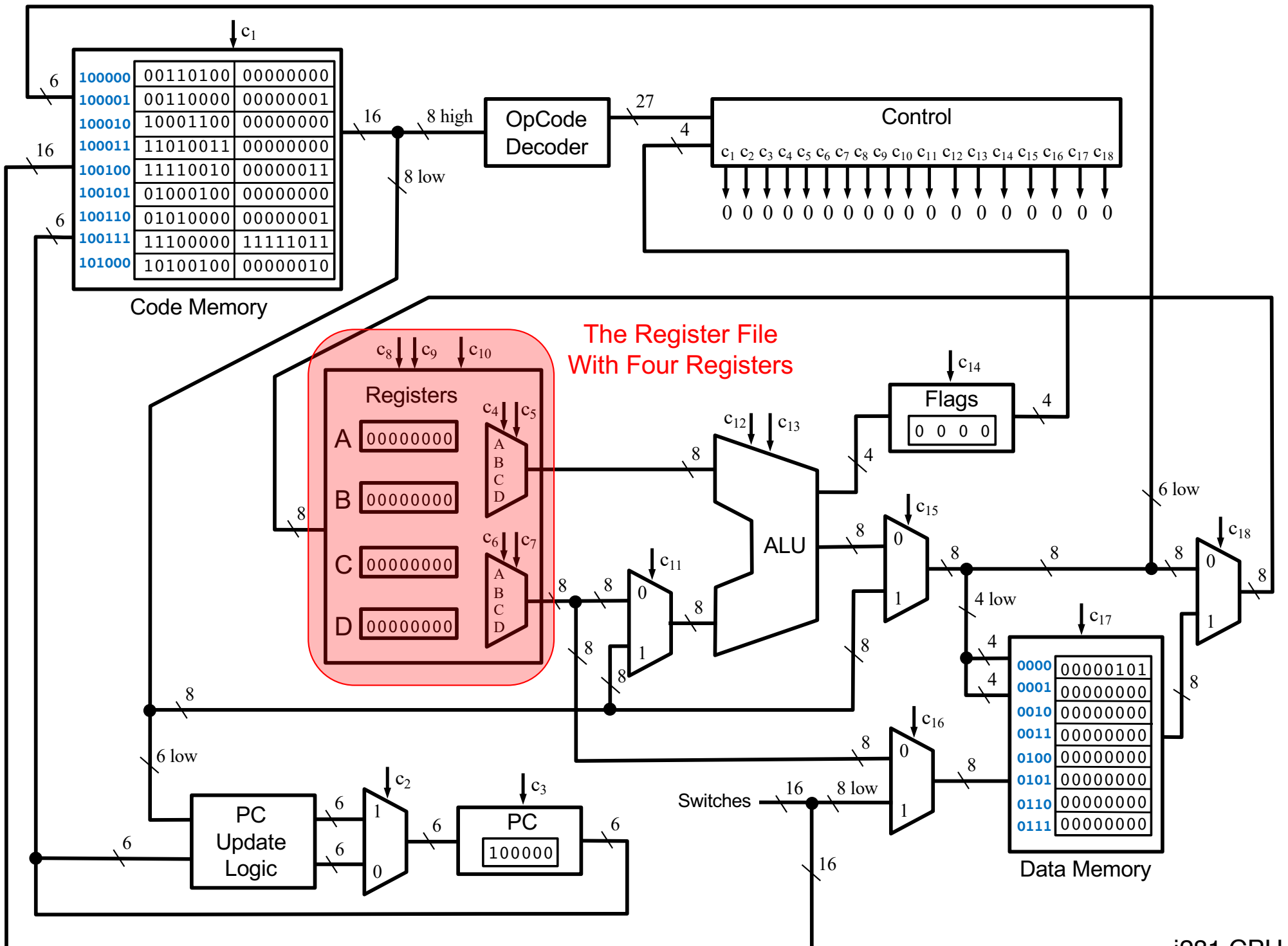




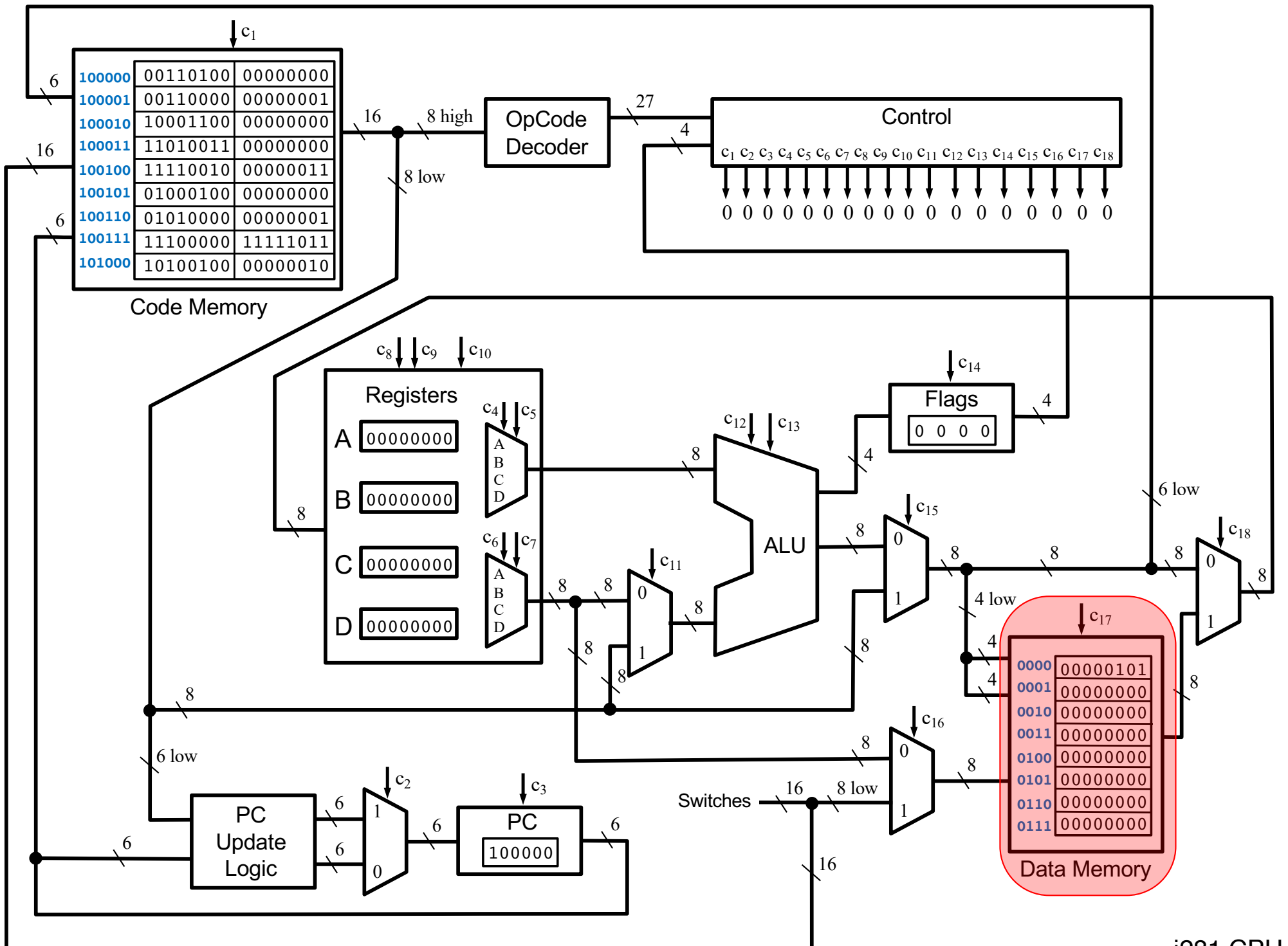




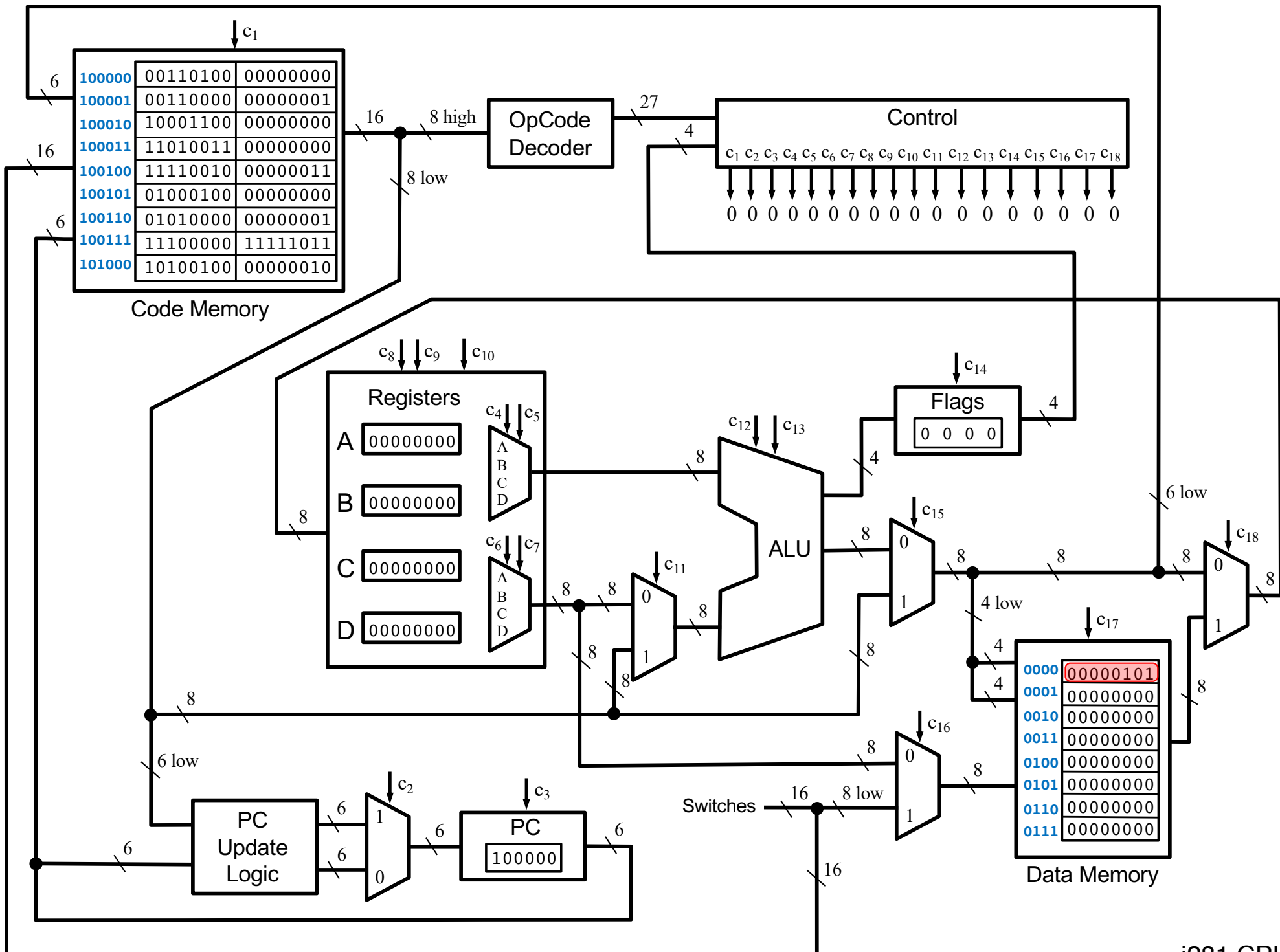




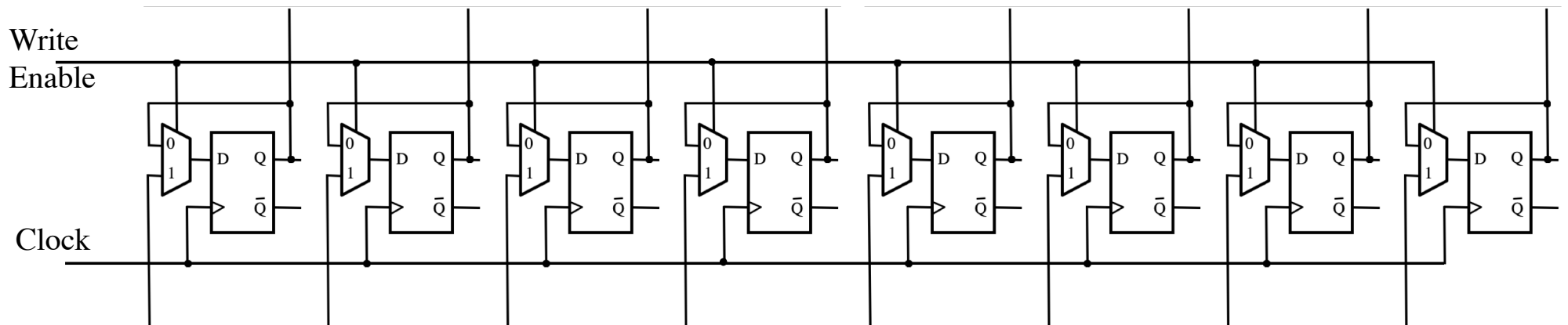
# **The Data Memory**



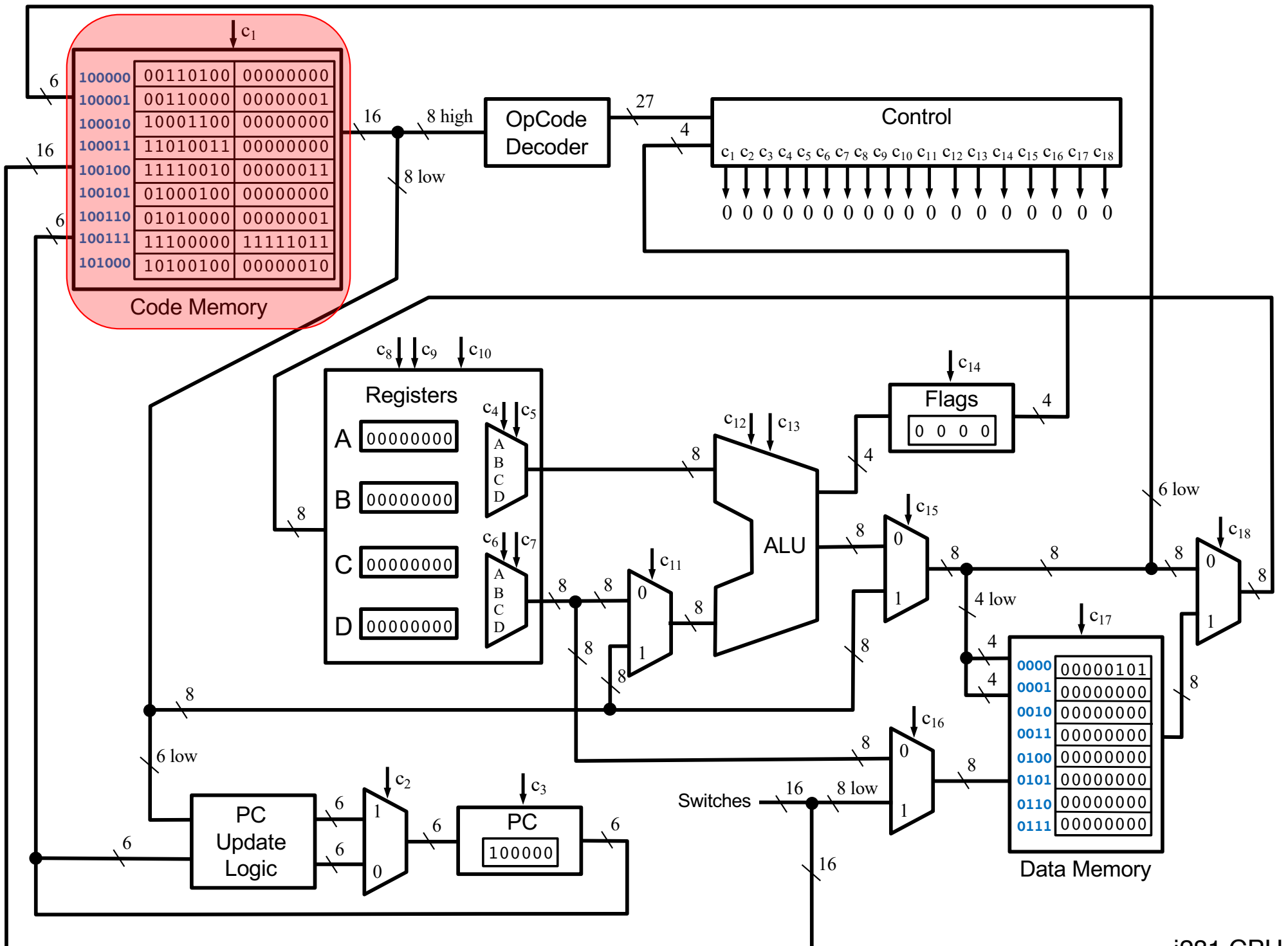
i281 CPU

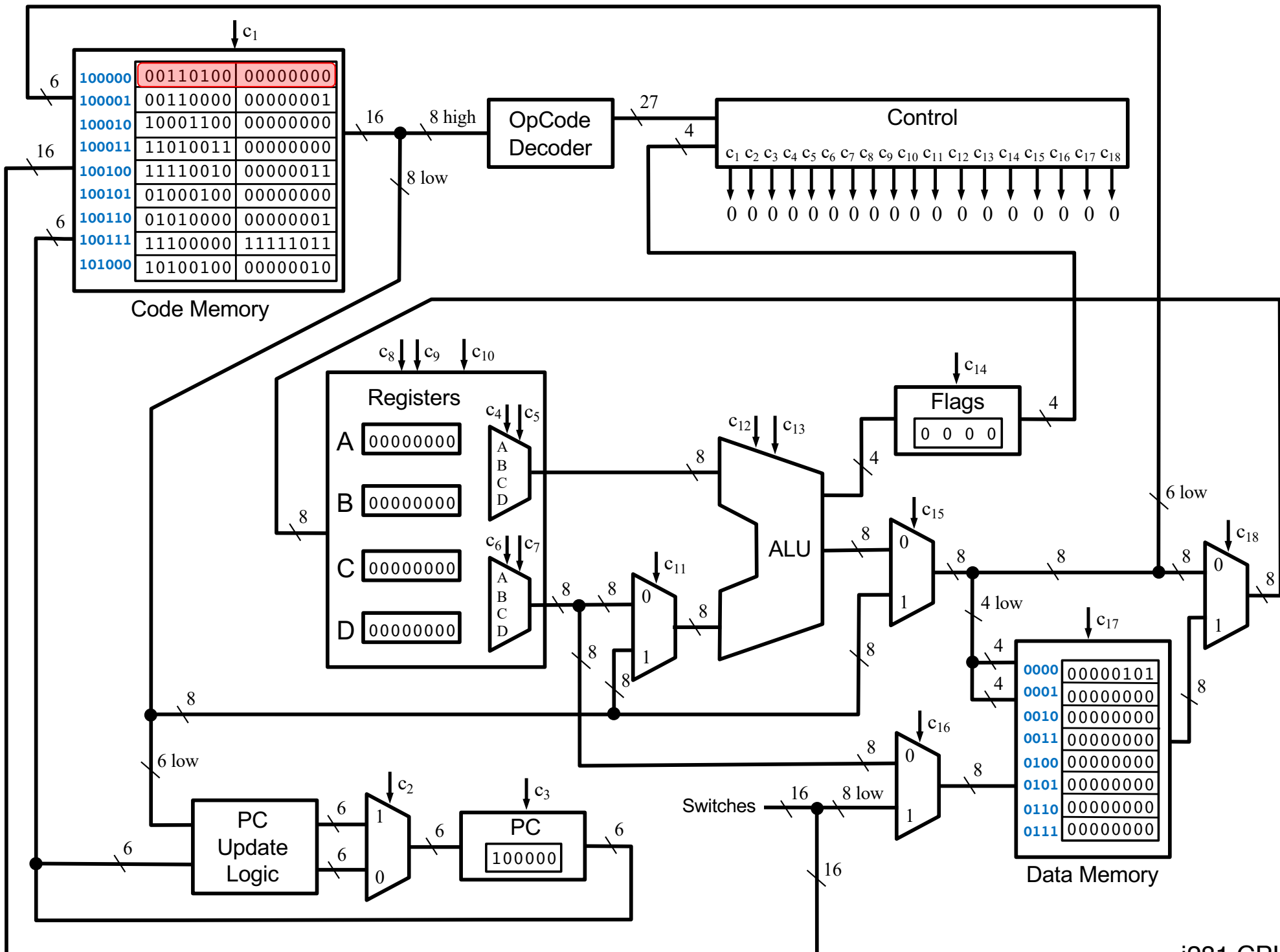


# 8-Bit Parallel-Access Register



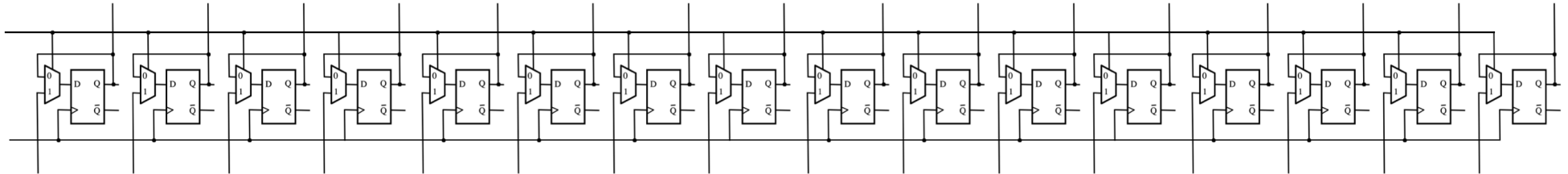
# **The Code Memory**



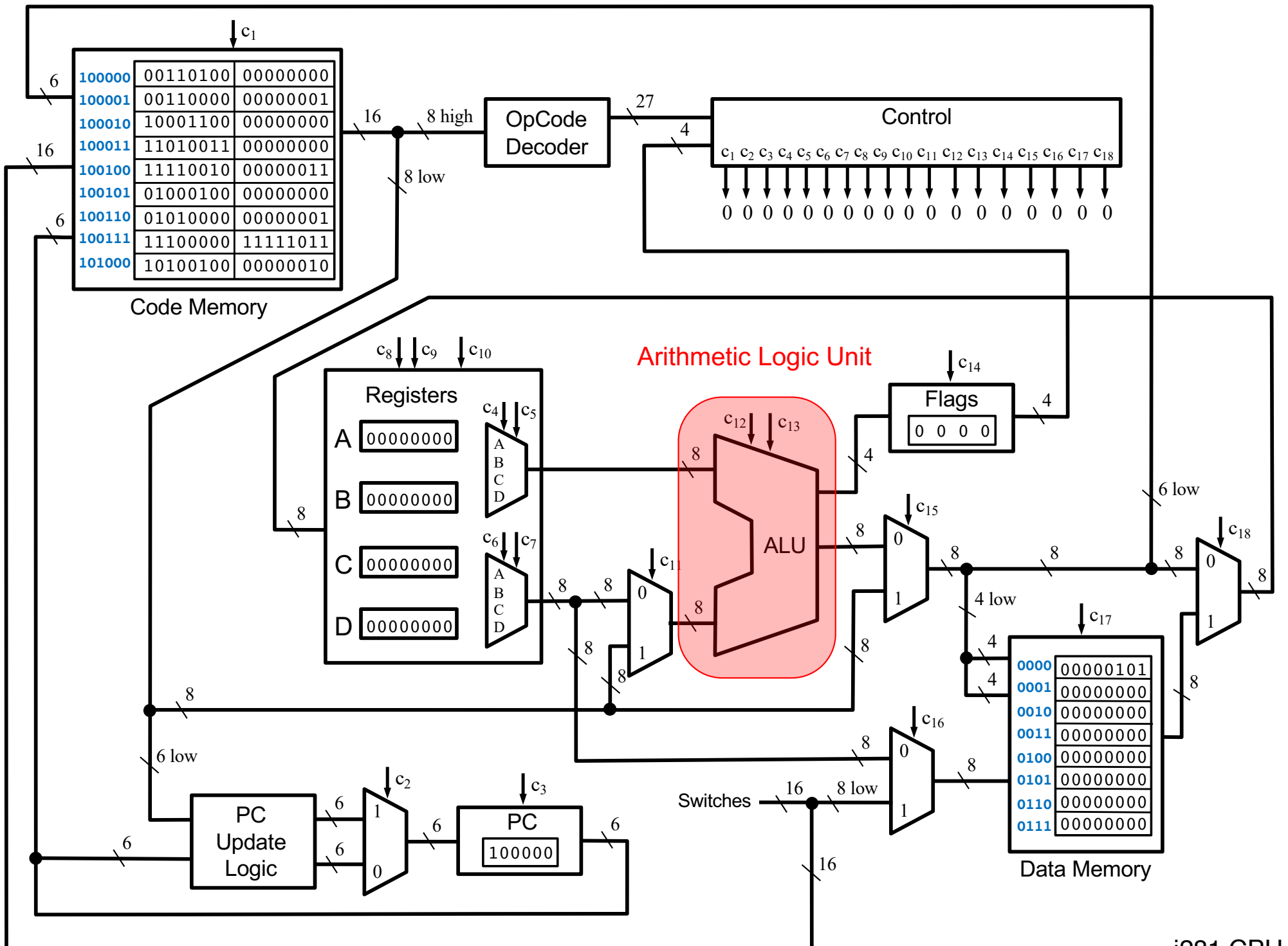


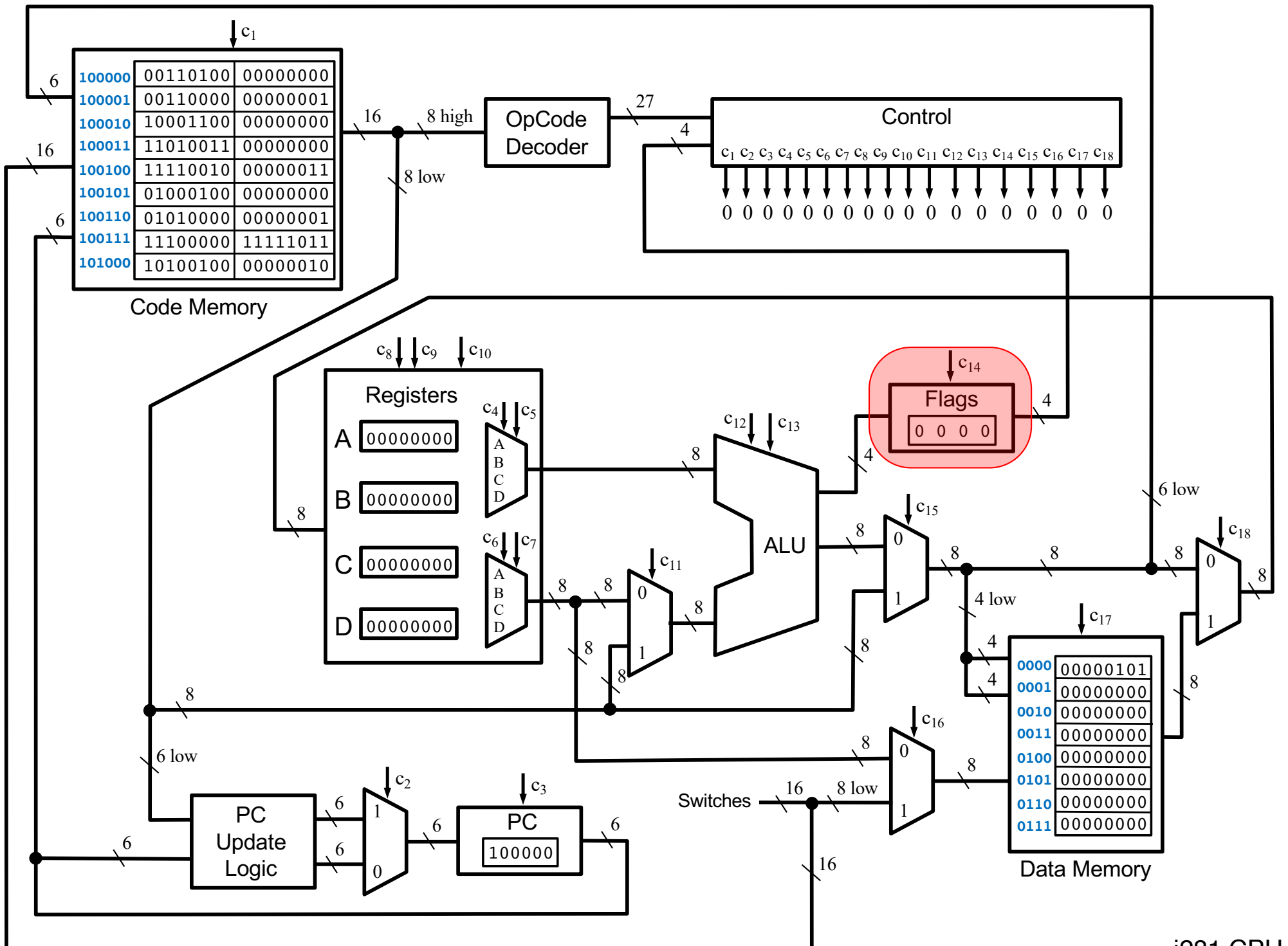


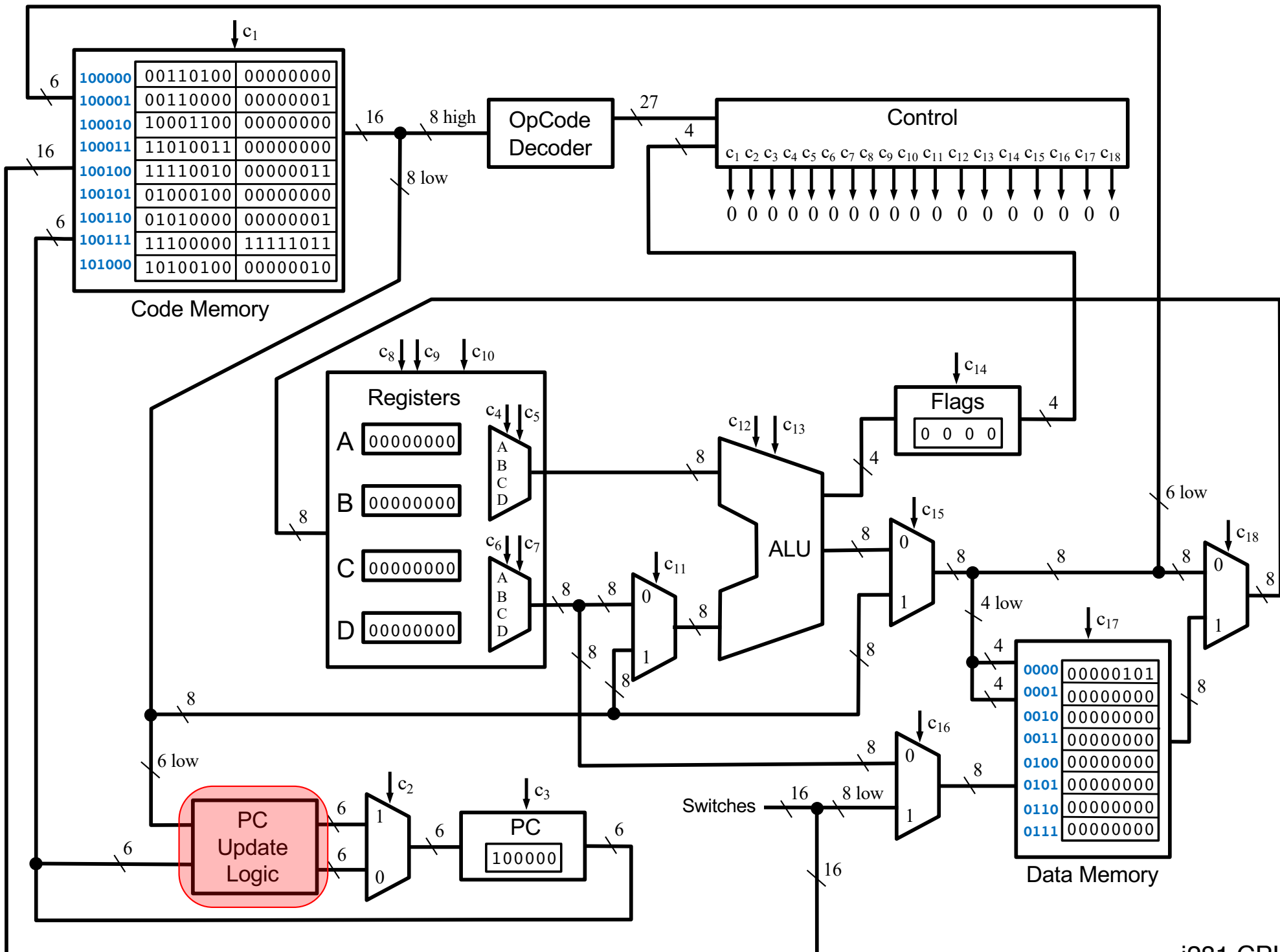
# 16-Bit Parallel-Access Register

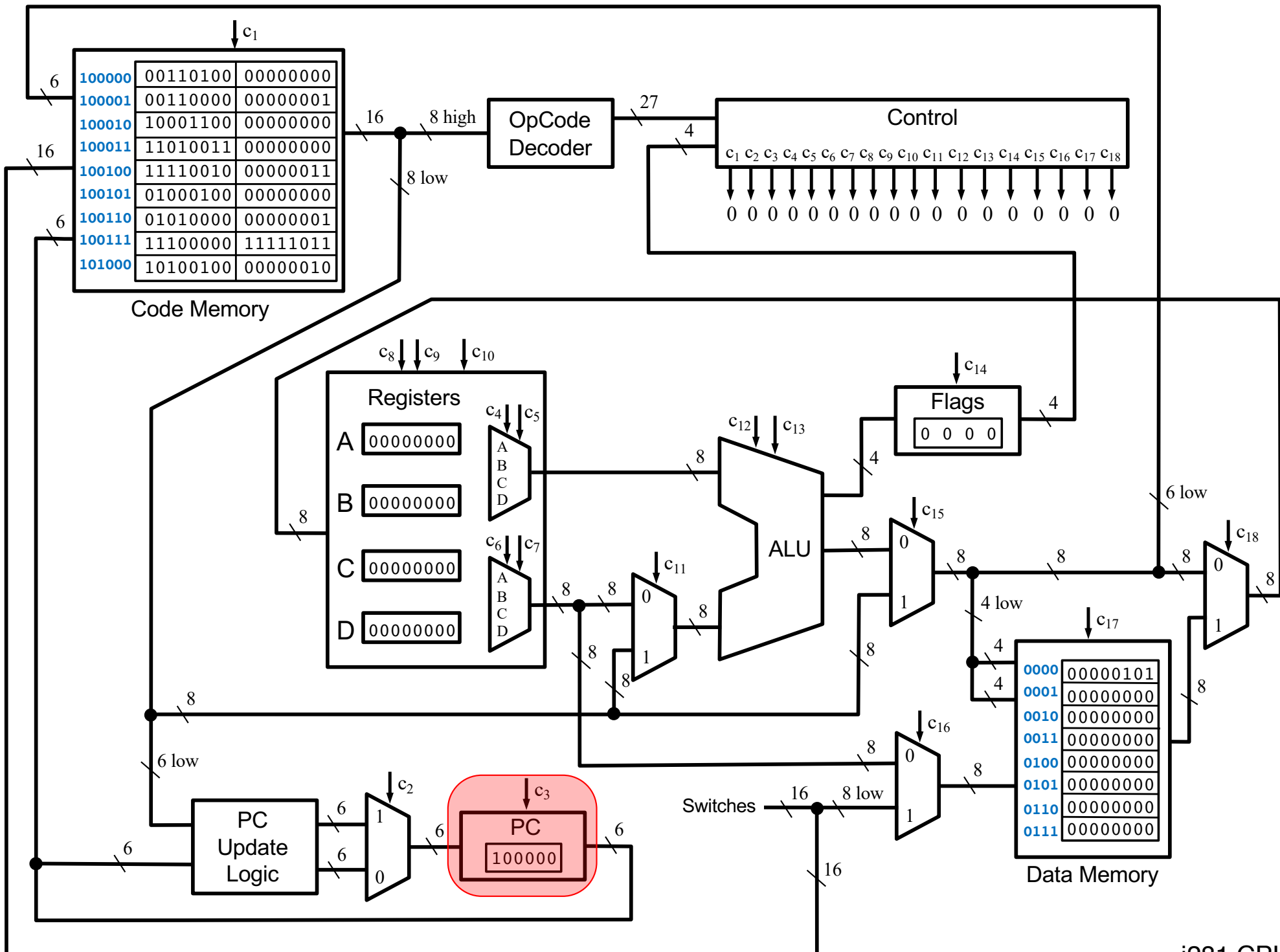


**To Be Covered  
Next Time**







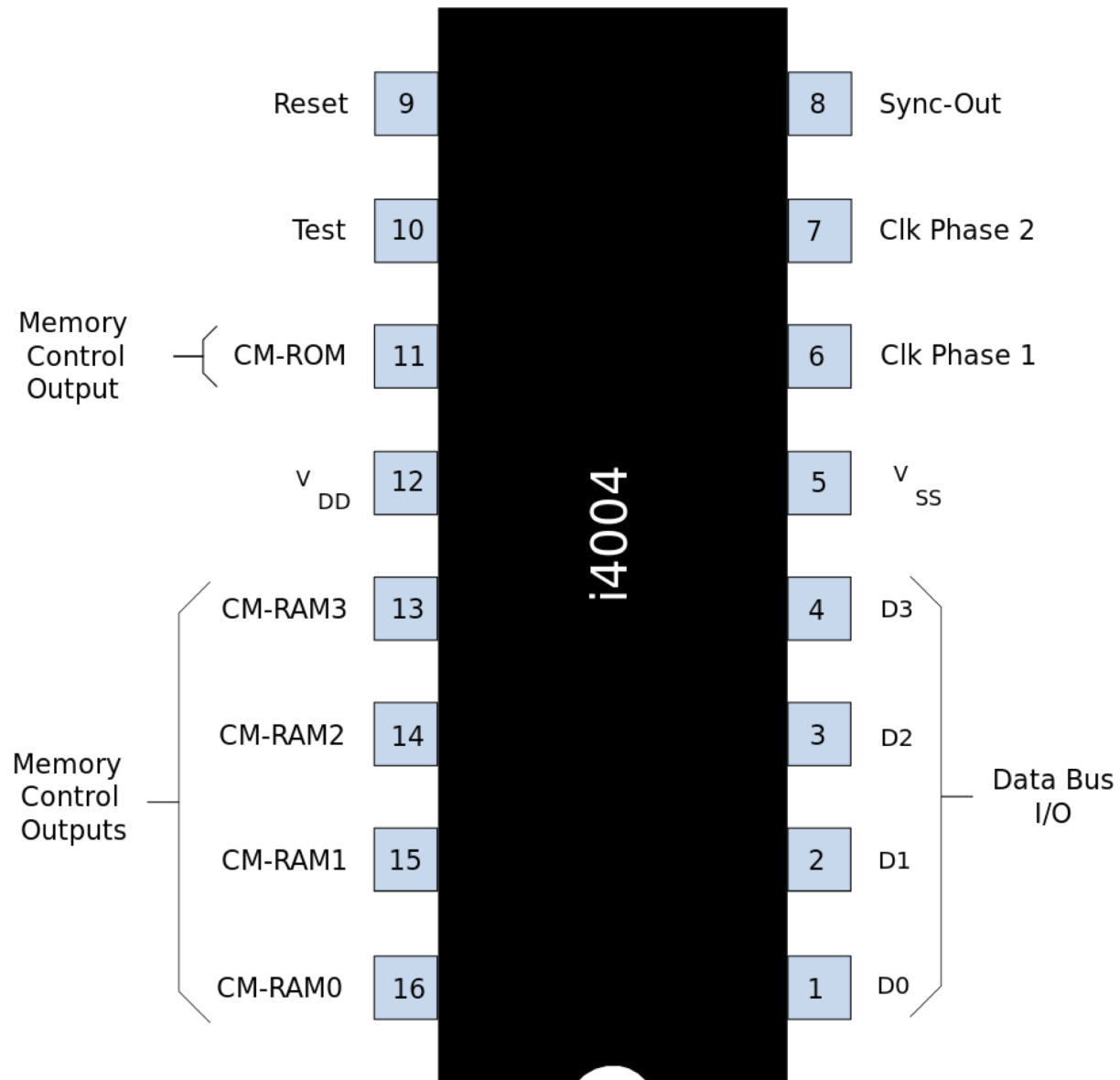


# **Some Additional Topics**

# **Examples of Some Famous Microprocessors**



# Intel's 4004 Chip

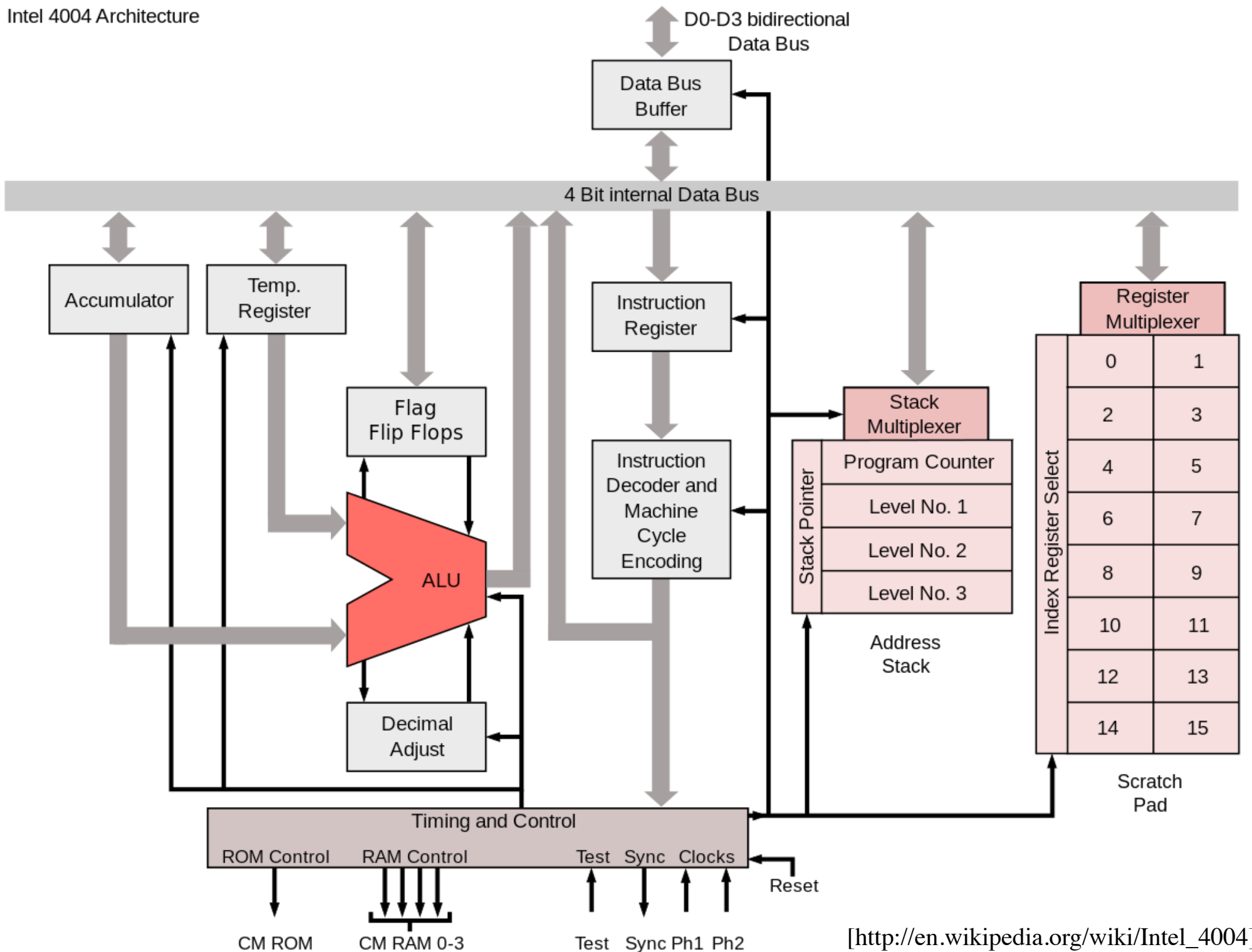


# Technical specifications

- **Maximum clock speed was 740 kHz**
- **Instruction cycle time: 10.8  $\mu$ s  
(8 clock cycles / instruction cycle)**
- **Instruction execution time 1 or 2 instruction cycles  
(10.8 or 21.6  $\mu$ s), 46300 to 92600 instructions per  
second**
- **Built using 2,300 transistors**

# Technical specifications

- **Separate program and data storage.**
- **The 4004, with its need to keep pin count down, used a single multiplexed 4-bit bus for transferring:**
  - **12-bit addresses**
  - **8-bit instructions**
  - **4-bit data words**
- **Instruction set contained 46 instructions (of which 41 were 8 bits wide and 5 were 16 bits wide)**
- **Register set contained 16 registers of 4 bits each**
- **Internal subroutine stack, 3 levels deep.**



## Intel 4004 registers

1 1 0 0 0 0 7 0 0 5 0 0 3 0 2 0 1 0 0 (bit position)

### Main registers

		<b>A</b>	<b>Accumulator</b>
	<b>R0</b>	<b>R1</b>	
	<b>R2</b>	<b>R3</b>	
	<b>R4</b>	<b>R5</b>	
	<b>R6</b>	<b>R7</b>	
	<b>R8</b>	<b>R9</b>	
	<b>R10</b>	<b>R11</b>	
	<b>R12</b>	<b>R13</b>	
	<b>R14</b>	<b>R15</b>	

### Program counter

<b>PC</b>	<b>Program Counter</b>
-----------	------------------------

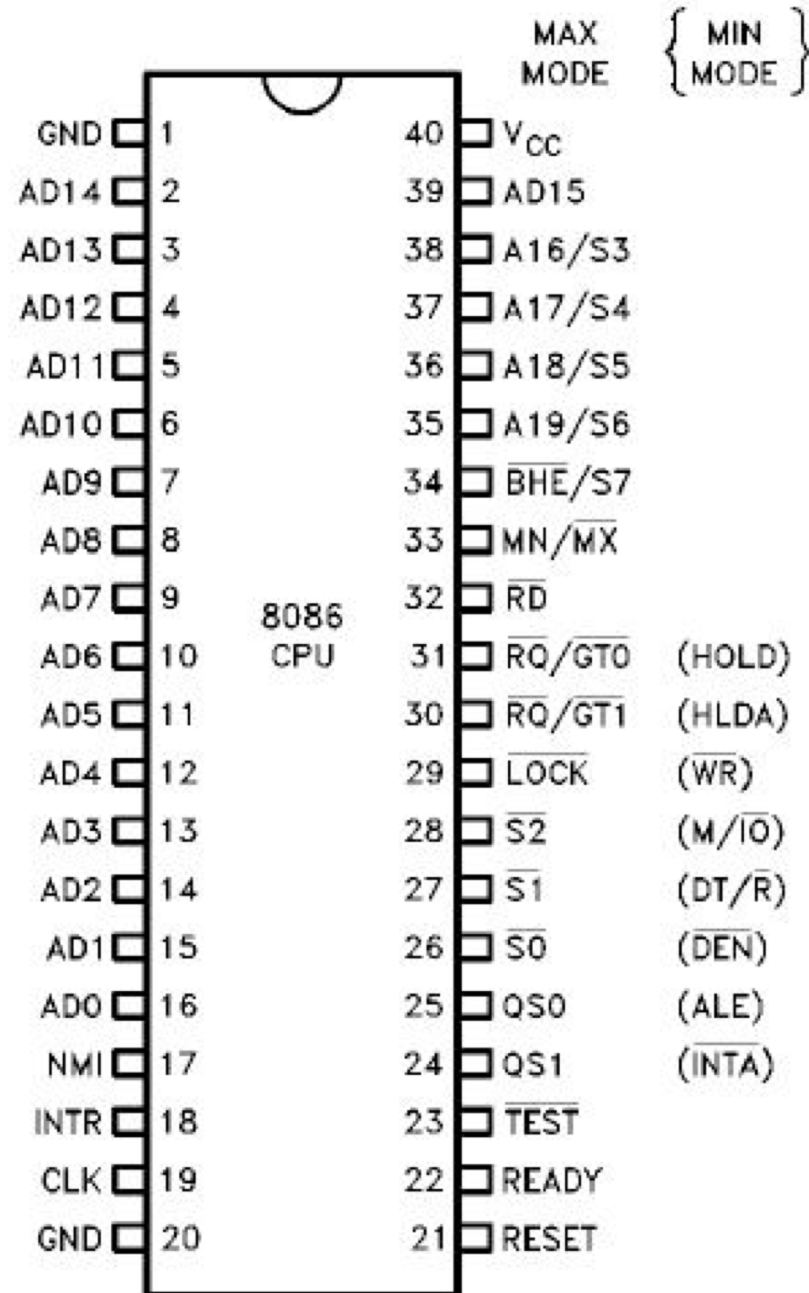
### Push-down address call stack

<b>PC1</b>	<b>Call level 1</b>
<b>PC2</b>	<b>Call level 2</b>
<b>PC3</b>	<b>Call level 3</b>

### Status register

	<b>C</b>	<b>P</b>	<b>Z</b>	<b>S</b>	<b>Flags</b>
--	----------	----------	----------	----------	--------------

# Intel's 8086 Chip



## Intel 8086 registers

1<sub>9</sub> 1<sub>8</sub> 1<sub>7</sub> 1<sub>6</sub> 1<sub>5</sub> 1<sub>4</sub> 1<sub>3</sub> 1<sub>2</sub> 1<sub>1</sub> 1<sub>0</sub> 0<sub>9</sub> 0<sub>8</sub> 0<sub>7</sub> 0<sub>6</sub> 0<sub>5</sub> 0<sub>4</sub> 0<sub>3</sub> 0<sub>2</sub> 0<sub>1</sub> 0<sub>0</sub> (bit position)

### Main registers

	AH	AL	<b>AX</b> (primary accumulator)
	BH	BL	<b>BX</b> (base, accumulator)
	CH	CL	<b>CX</b> (counter, accumulator)
	DH	DL	<b>DX</b> (accumulator, other functions)

### Index registers

0 0 0 0	SI	<b>Source Index</b>
0 0 0 0	DI	<b>Destination Index</b>
0 0 0 0	BP	<b>Base Pointer</b>
0 0 0 0	SP	<b>Stack Pointer</b>

### Program counter

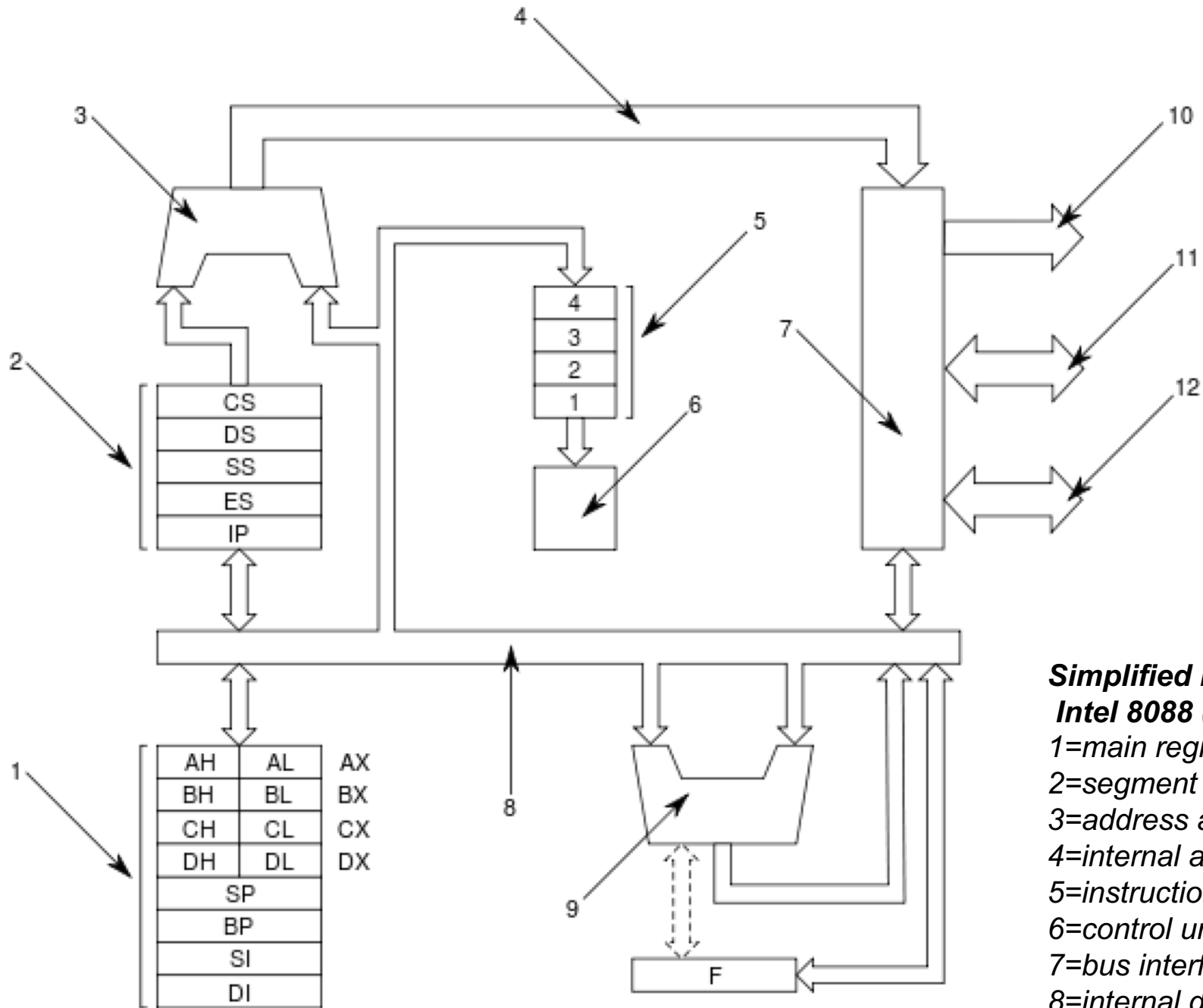
0 0 0 0	IP	<b>Instruction Pointer</b>
---------	----	----------------------------

### Segment registers

CS	0 0 0 0	<b>Code Segment</b>
DS	0 0 0 0	<b>Data Segment</b>
ES	0 0 0 0	<b>ExtraSegment</b>
SS	0 0 0 0	<b>Stack Segment</b>

### Status register

- - - - O D I T S Z - A - P - C	Flags
---------------------------------	-------



**Simplified block diagram of Intel 8088 (a variant of 8086);**

- 1=main registers;
- 2=segment registers and IP;
- 3=address adder;
- 4=internal address bus;
- 5=instruction queue;
- 6=control unit (very simplified!);
- 7=bus interface;
- 8=internal databus;
- 9=ALU;
- 10/11/12=external address/  
data/control bus.



**Questions?**

**THE END**