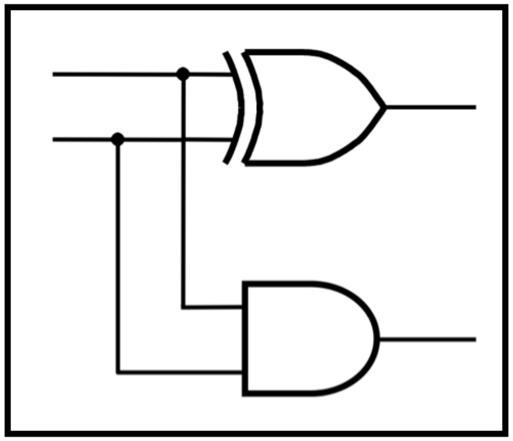


CprE 2810: Digital Logic

Instructor: Alexander Stoytchev

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

The Intersection Between Hardware and Software

CprE 2810: Digital Logic
Iowa State University, Ames, IA
Copyright © Alexander Stoytchev

Administrative Stuff

- The FINAL exam is scheduled for
- Wednesday Dec 18 @ 2:15 – 4:15 PM

Final Exam Format

- The exam will cover: Chapter 1 to Chapter 6, and Sections 7.1-7.2, register machines, and i281 CPU
- Emphasis will be on Chapter 5, 6, and 7
- The exam will be closed book but open notes.
- You can bring up to 5 pages of handwritten or typed notes.

Final Exam Format

- The exam will be out of 135 points
- You need 95 points to get an A on this exam
- It will be great if you can score more than 100 points.
  - but you can't roll over your extra points ☹

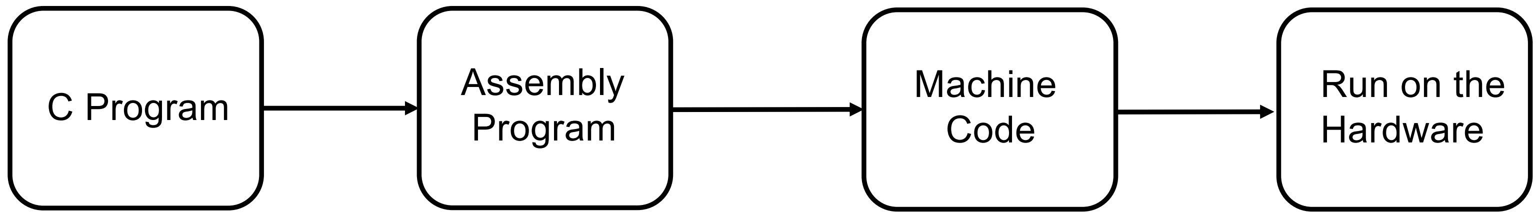
Topics for the Final Exam

- K-maps for 2, 3, and 4 variables
- Multiplexers (circuits and function)
- Synthesis of logic functions using multiplexers
- Shannon’s Expansion Theorem
- 1’s complement and 2’s complement representation
- Addition and subtraction of binary numbers
- Circuits for adding and subtracting
- Serial adder
- Latches (circuits, behavior, timing diagrams)
- Flip-Flops (circuits, behavior, timing diagrams)
- Counters (up, down, synchronous, asynchronous)
- Registers and Register Files

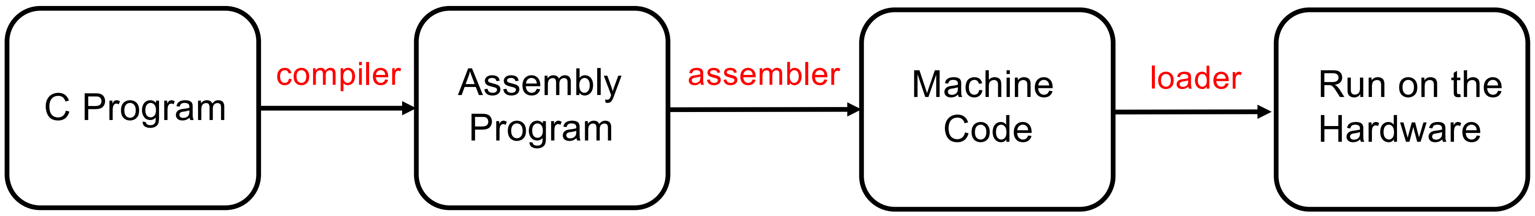
Topics for the Final Exam

- Synchronous Sequential Circuits
- FSMs
- Moore Machines
- Mealy Machines
- State diagrams, state tables, state-assigned tables
- State minimization
- Designing a counter
- Arbiter Circuits
- Reverse engineering a circuit
- ASM Charts
- Register Machines and programs for them
- ALU, PC, and control for a simple processor (i281 CPU)
- Assembly and machine language (i281 assembly)
- Something from Star Wars

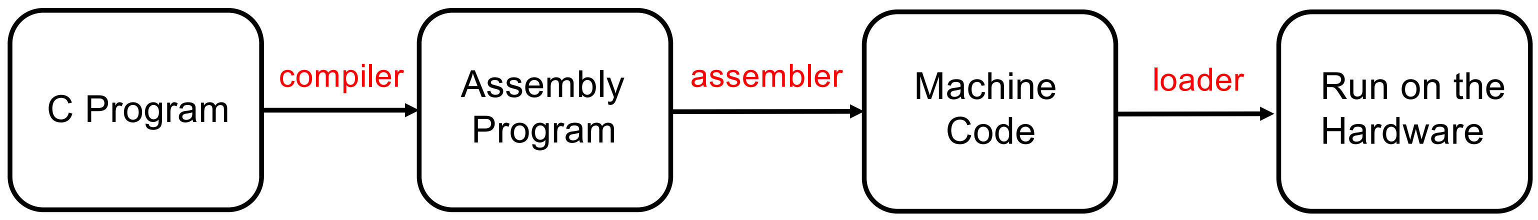
Writing and Running a Program



Writing and Running a Program

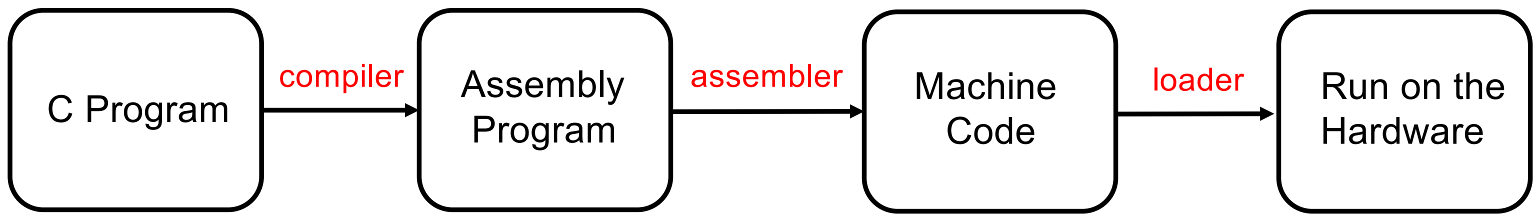


Writing and Running a Program



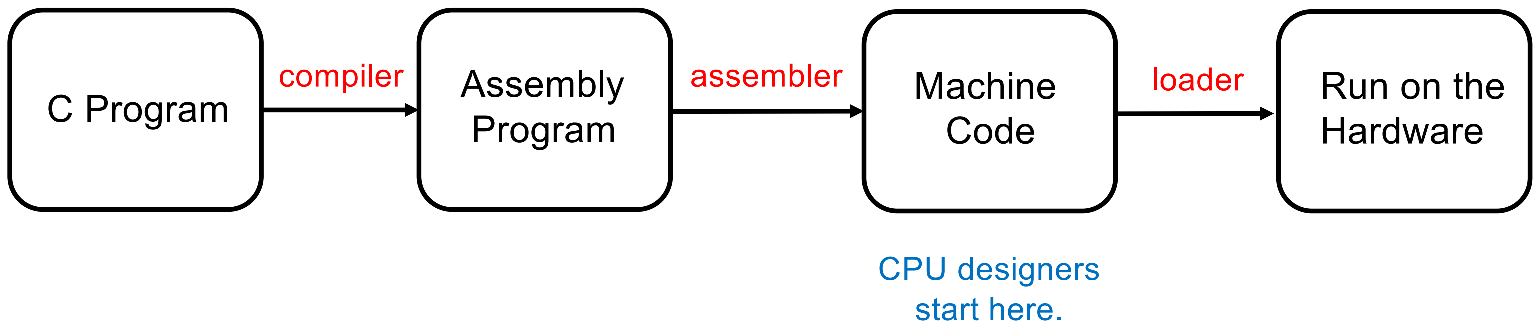
The programmer only writes this in a text editor.

Writing and Running a Program



Nerds skip the first step and start here.

Writing and Running a Program



i281 Example:
Add the numbers from 1 to 5

i281 Example:
Add the numbers from 1 to 5

C Language v.s. Assembly Language

C Version

```
// C Version
//
// Add the numbers from 1 to 5 using a for loop.

int main()
{
        int N=5;
        int i, sum;

        sum=0;
        for(i=1; i<=N; i++)
           sum+=i;

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

i281 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    ; update the memory for sum

; Register allocation:
; A: i
; B: sum
; C: <not used>
; D: N
```

i281 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     ; update the memory for sum

; Register allocation:
; A: i
; B: sum
; C: <not used>
; D: N
```

Add the numbers from 1 to 5

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





int main()
{
        int N=5;
        int i, sum;

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

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

```asm
; 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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

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

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

```asm
; 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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

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

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

i=1

```
; 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

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




int main()
{
        int N=5;
        int i, sum;

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

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

This has no analog in the C version, which is written in a high-level language.

```
; 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
```

Load the value of N into register D.

Add the numbers from 1 to 5

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

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

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

```asm
; 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

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




int main()
{
        int N=5;
        int i, sum;

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

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

i=2

```
; 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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

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

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

i=3

```
; 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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

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

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

              i=4
```

```
; 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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

        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

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



int main()
{
        int N=5;
        int i, sum;

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

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

i=5

```
; 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

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




int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

        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

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



int main()
{
        int N=5;
        int i, sum;

        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

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




int main()
{
        int N=5;
        int i, sum;

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

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

i=6

```
; 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

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




int main()
{
        int N=5;
        int i, sum;

        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
```

i281 Example:
Add the numbers from 1 to 5

Assembly Language v.s. Machine Language

i281 Assembly Code

```
.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     ; update the memory for sum
```

i281 Assembly Code

```
.data
N         BYTE    5
i         BYTE    ?
sum       BYTE    ?

.code
        LOADI  B, 0
        LOADI  A, 1
        LOAD   D, [N]
Loop:   CMP    A, D
        BRG    End
Add:    ADD    B, A
        ADDI   A, 1
        JUMP   Loop
End:    STORE  [sum], B
```

Mapping Assembly to Machine Code

```
.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code
         LOADI   B, 0
         LOADI   A, 1
         LOAD    D, [N]
Loop:    CMP     A, D
         BRG     End
Add:     ADD     B, A
         ADDI    A, 1
         JUMP    Loop
End:     STORE   [sum], B
```

Assembly Language

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011010000000000
0011000000000001
1000110000000000
1101001100000000
1111001000000011
0100010000000000
0101000000000001
1110000011111011
1010010000000010
```

Machine Language

Mapping Assembly to Machine Code

```
.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code
           LOADI   B, 0
           LOADI   A, 1
           LOAD    D, [N]
Loop:      CMP     A, D
           BRG     End
Add:       ADD     B, A
           ADDI    A, 1
           JUMP    Loop
End:       STORE   [sum], B
```

Assembly Language

```
Data Memory:
0000 0101
0000 0000
0000 0000

Code Memory:
0011 0100 0000 0000
0011 0000 0000 0001
1000 1100 0000 0000
1101 0011 0000 0000
1111 0010 0000 0011
0100 0100 0000 0000
0101 0000 0000 0001
1110 0000 1111 1011
1010 0100 0000 0010
```

Machine Language
in Binary

Mapping Assembly to Machine Code

```
.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code
           LOADI   B, 0
           LOADI   A, 1
           LOAD    D, [N]
Loop:      CMP     A, D
           BRG     End
Add:       ADD     B, A
           ADDI    A, 1
           JUMP    Loop
End:       STORE   [sum], B
```

Assembly Language

```
Data Memory:
 0     5
 0     0
 0     0

Code Memory:
 3     4     0     0
 3     0     0     1
 8     C     0     0
 D     3     0     0
 F     2     0     3
 4     4     0     0
 5     0     0     1
 E     0     F     B
 A     4     0     2
```

Machine Language
in Binary

Mapping Assembly to Machine Code

```
.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code
         LOADI   B, 0
         LOADI   A, 1
         LOAD    D, [N]
Loop:    CMP     A, D
         BRG     End
Add:     ADD     B, A
         ADDI    A, 1
         JUMP    Loop
End:     STORE   [sum], B
```

Assembly Language

```
Data Memory:
05
00
00

Code Memory:
34 00
30 01
8C 00
D3 00
F2 03
44 00
50 01
E0 FB
A4 02
```

Machine Language
in Hexadecimal

i281 Example:
Add the numbers from 1 to 5

Bit Mapping for OPCODEs

Mapping Assembly to Machine Code

```
.data
N           BYTE     5
i           BYTE     ?
sum         BYTE     ?

.code
            LOADI    B, 0
            LOADI    A, 1
            LOAD     D, [N]
Loop:       CMP      A, D
            BRG      End
Add:        ADD      B, A
            ADDI     A, 1
            JUMP     Loop
End:        STORE    [sum], B
```

Assembly Language

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011010000000000
0011000000000001
1000110000000000
1101001100000000
1111001000000011
0100010000000000
0101000000000001
1110000011111011
1010010000000010
```

Machine Language

Mapping Assembly to Machine Code

```
.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code
          LOADI   B, 0
          LOADI   A, 1
          LOAD    D, [N]
Loop:     CMP     A, D
          BRG     End
Add:      ADD     B, A
          ADDI    A, 1
          JUMP    Loop
End:      STORE   [sum], B
```

Assembly Language

```
Data Memory:
00000101
00000000
00000000

Code Memory:
00110100_00000000
00110000_00000001
10001100_00000000
11010011_00000000
11110010_00000011
01000100_00000000
01010000_00000001
11100000_11111011
10100100_00000010
```

Machine Language

Mapping Assembly to Machine Code

```
.data
N         BYTE    5
i         BYTE    ?
sum       BYTE    ?

.code
          LOADI   B, 0
          LOADI   A, 1
          LOAD    D, [N]
Loop:     CMP     A, D
          BRG     End
Add:      ADD     B, A
          ADDI    A, 1
          JUMP    Loop
End:      STORE   [sum], B
```

Assembly Language

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Machine Language

Mapping Assembly to Machine Code

```
.data                                       Data Memory:
N         BYTE    5                         00000101
i         BYTE    ?                         00000000
sum       BYTE    ?                         00000000

.code                                       Code Memory:
          LOADI   B, 0                      0011_01_00_00000000
          LOADI   A, 1                      0011_00_00_00000001
          LOAD    D, [N]                    1000_11_00_00000000
Loop:     CMP     A, D                      1101_00_11_00000000
          BRG     End                       1111_00_10_00000011
Add:      ADD     B, A                      0100_01_00_00000000
          ADDI    A, 1                      0101_00_00_00000001
          JUMP    Loop                      1110_00_00_11111011
End:      STORE   [sum], B                  1010_01_00_00000010
```

Mapping Assembly to Machine Code

```
.data                                      Data Memory:
N          BYTE    5                       00000101
i          BYTE    ?                       00000000
sum        BYTE    ?                       00000000

.code                                      Code Memory:
        LOADI   B, 0                       0011_01_00_00000000
        LOADI   A, 1                       0011_00_00_00000001
        LOAD    D, [N]                     1000_11_00_00000000
Loop:   CMP     A, D                       1101_00_11_00000000
        BRG     End                        1111_00_10_00000011
Add:    ADD     B, A                       0100_01_00_00000000
        ADDI    A, 1                       0101_00_00_00000001
        JUMP    Loop                       1110_00_00_11111011
End:    STORE   [sum], B                   1010_01_00_00000010
```

Mapping Assembly to Machine Code

```
.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code
        LOADI   B, 0
        LOADI   A, 1
        LOAD    D, [N]
Loop:   CMP     A, D
        BRG     End
Add:    ADD     B, A
        ADDI    A, 1
        JUMP    Loop
End:    STORE   [sum], B
```

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Mapping Assembly to Machine Code

```
.data                                  Data Memory:
N         BYTE    5                    00000101
i         BYTE    ?                    00000000
sum       BYTE    ?                    00000000

.code                                  Code Memory:
          LOADI   B, 0                 0011_01_00_00000000
          LOADI   A, 1                 0011_00_00_00000001
          LOAD    D, [N]               1000_11_00_00000000
Loop:     CMP     A, D                 1101_00_11_00000000
          BRG     End                  1111_00_10_00000011
Add:      ADD     B, A                 0100_01_00_00000000
          ADDI    A, 1                 0101_00_00_00000001
          JUMP    Loop                 1110_00_00_11111011
End:      STORE   [sum], B             1010_01_00_00000010
```

OPCODE Mapping

```
.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code
        LOADI   B, 0
        LOADI   A, 1
        LOAD    D, [N]
Loop:   CMP     A, D
        BRG     End
Add:    ADD     B, A
        ADDI    A, 1
        JUMP    Loop
End:    STORE   [sum], B
```

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

OPCODE Mapping

```
.data
N           BYTE    5
i           BYTE    ?
sum         BYTE    ?

.code
          LOADI   B, 0
          LOADI   A, 1
          LOAD    D, [N]
Loop:     CMP     A, D
          BRG     End
Add:      ADD     B, A
          ADDI    A, 1
          JUMP    Loop
End:      STORE   [sum], B
```

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Register Parameter Mapping

```
.data                                    Data Memory:
N          BYTE    5                     00000101
i          BYTE    ?                     00000000
sum        BYTE    ?                     00000000

.code                                    Code Memory:
        LOADI   B, 0                     0011_01_00_00000000
        LOADI   A, 1                     0011_00_00_00000001
        LOAD    D, [N]                   1000_11_00_00000000
Loop:   CMP     A, D                     1101_00_11_00000000
        BRG     End                      1111_00_10_00000011
Add:    ADD     B, A                     0100_01_00_00000000
        ADDI    A, 1                     0101_00_00_00000001
        JUMP    Loop                     1110_00_00_11111011
End:    STORE   [sum], B                 1010_01_00_00000010
```

Register Parameter Mapping

```
.data
N         BYTE    5
i         BYTE    ?
sum       BYTE    ?

.code
          LOADI   B, 0
          LOADI   A, 1
          LOAD    D, [N]
Loop:     CMP     A, D
          BRG     End
Add:      ADD     B, A
          ADDI    A, 1
          JUMP    Loop
End:      STORE   [sum], B
```

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Second Register Parameter Mapping

```
.data                                      Data Memory:
N        BYTE    5                         00000101
i        BYTE    ?                         00000000
sum      BYTE    ?                         00000000

.code                                      Code Memory:
         LOADI   B, 0                      0011_01_00_00000000
         LOADI   A, 1                      0011_00_00_00000001
         LOAD    D, [N]                    1000_11_00_00000000
Loop:    CMP     A, D                      1101_00_11_00000000
         BRG     End                       1111_00_10_00000011
Add:     ADD     B, A                      0100_01_00_00000000
         ADDI    A, 1                      0101_00_00_00000001
         JUMP    Loop                      1110_00_00_11111011
End:     STORE   [sum], B                  1010_01_00_00000010
```

Second Register Parameter Mapping

```
.data
N         BYTE    5
i         BYTE    ?
sum       BYTE    ?

.code
          LOADI   B, 0
          LOADI   A, 1
          LOAD    D, [N]
Loop:     CMP     A, D
          BRG     End
Add:      ADD     B, A
          ADDI    A, 1
          JUMP    Loop
End:      STORE   [sum], B
```

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Value / Address / Offset Mapping

```
.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code
        LOADI   B, 0
        LOADI   A, 1
        LOAD    D, [N]
Loop:   CMP     A, D
        BRG     End
Add:    ADD     B, A
        ADDI    A, 1
        JUMP    Loop
End:    STORE   [sum], B
```

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Value / Address / Offset Mapping

```
.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code
           LOADI   B, 0
           LOADI   A, 1
           LOAD    D, [N]
Loop:      CMP     A, D
           BRG     End
Add:       ADD     B, A
           ADDI    A, 1
           JUMP    Loop
End:       STORE   [sum], B
```

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

“Don’t care” bits …

```
.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code
        LOADI   B, 0
        LOADI   A, 1
        LOAD    D, [N]
Loop:   CMP     A, D
        BRG     End
Add:    ADD     B, A
        ADDI    A, 1
        JUMP    Loop
End:    STORE   [sum], B
```

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_dd_00000000
0011_00_dd_00000001
1000_11_dd_00000000
1101_00_11_dddddddd
1111_dd_10_00000011
0100_01_00_dddddddd
0101_00_dd_00000001
1110_dd_dd_11111011
1010_01_dd_00000010
```

… are mapped to 0 by the Assembler

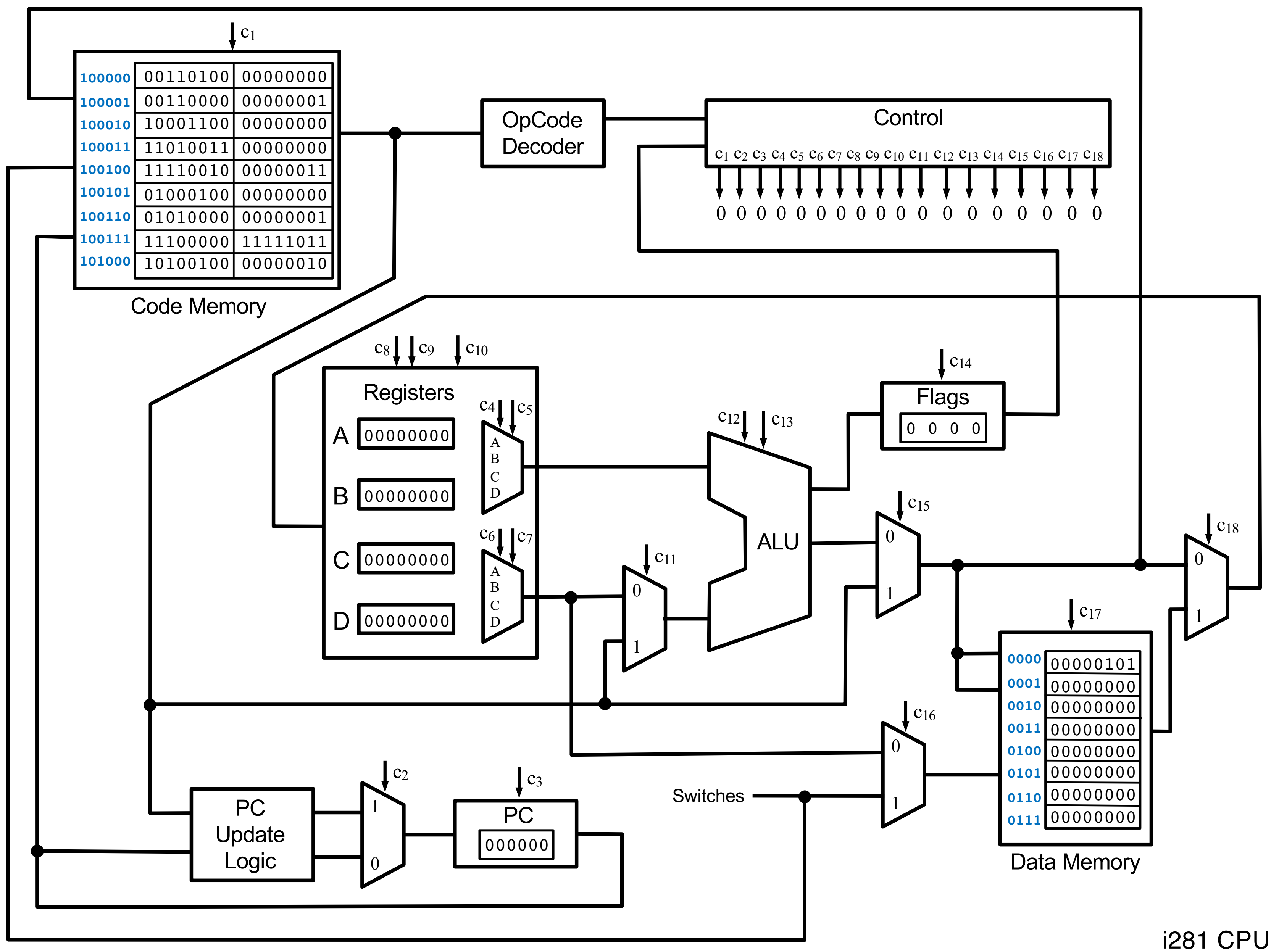
```
.data                                        Data Memory:
N          BYTE    5                         00000101
i          BYTE    ?                         00000000
sum        BYTE    ?                         00000000

.code                                        Code Memory:
           LOADI   B, 0                      0011_01_00_00000000
           LOADI   A, 1                      0011_00_00_00000001
           LOAD    D, [N]                    1000_11_00_00000000
Loop:      CMP     A, D                      1101_00_11_00000000
           BRG     End                       1111_00_10_00000011
Add:       ADD     B, A                      0100_01_00_00000000
           ADDI    A, 1                      0101_00_00_00000001
           JUMP    Loop                      1110_00_00_11111011
End:       STORE   [sum], B                  1010_01_00_00000010
```

Mapping Assembly to Machine Code

```
.data
N           BYTE     5
i           BYTE     ?
sum         BYTE     ?

.code
            LOADI    B, 0
            LOADI    A, 1
            LOAD     D, [N]
Loop:       CMP      A, D
            BRG      End
Add:        ADD      B, A
            ADDI     A, 1
            JUMP     Loop
End:        STORE    [sum], B
```

```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

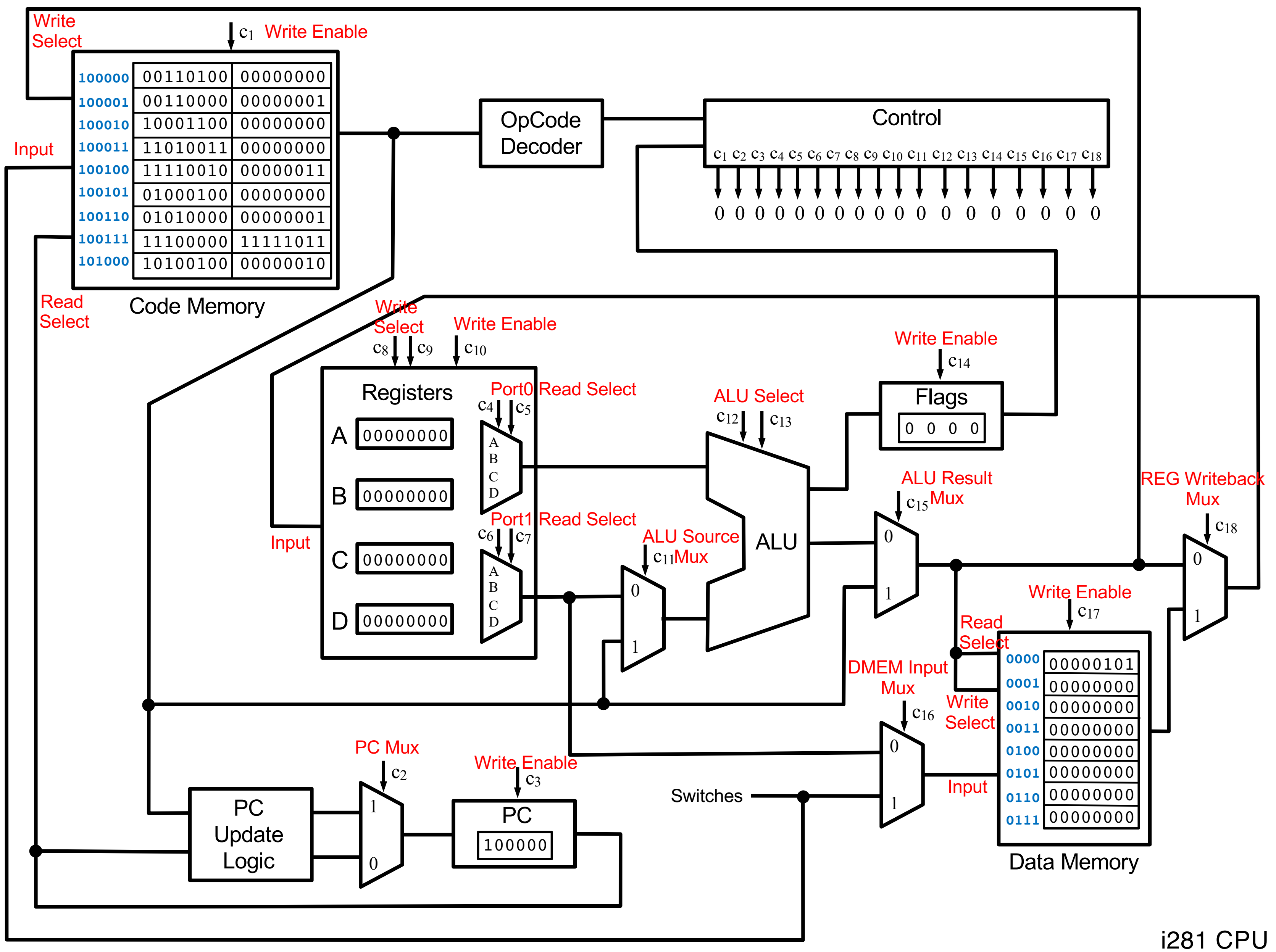
Loading the Program into Memory



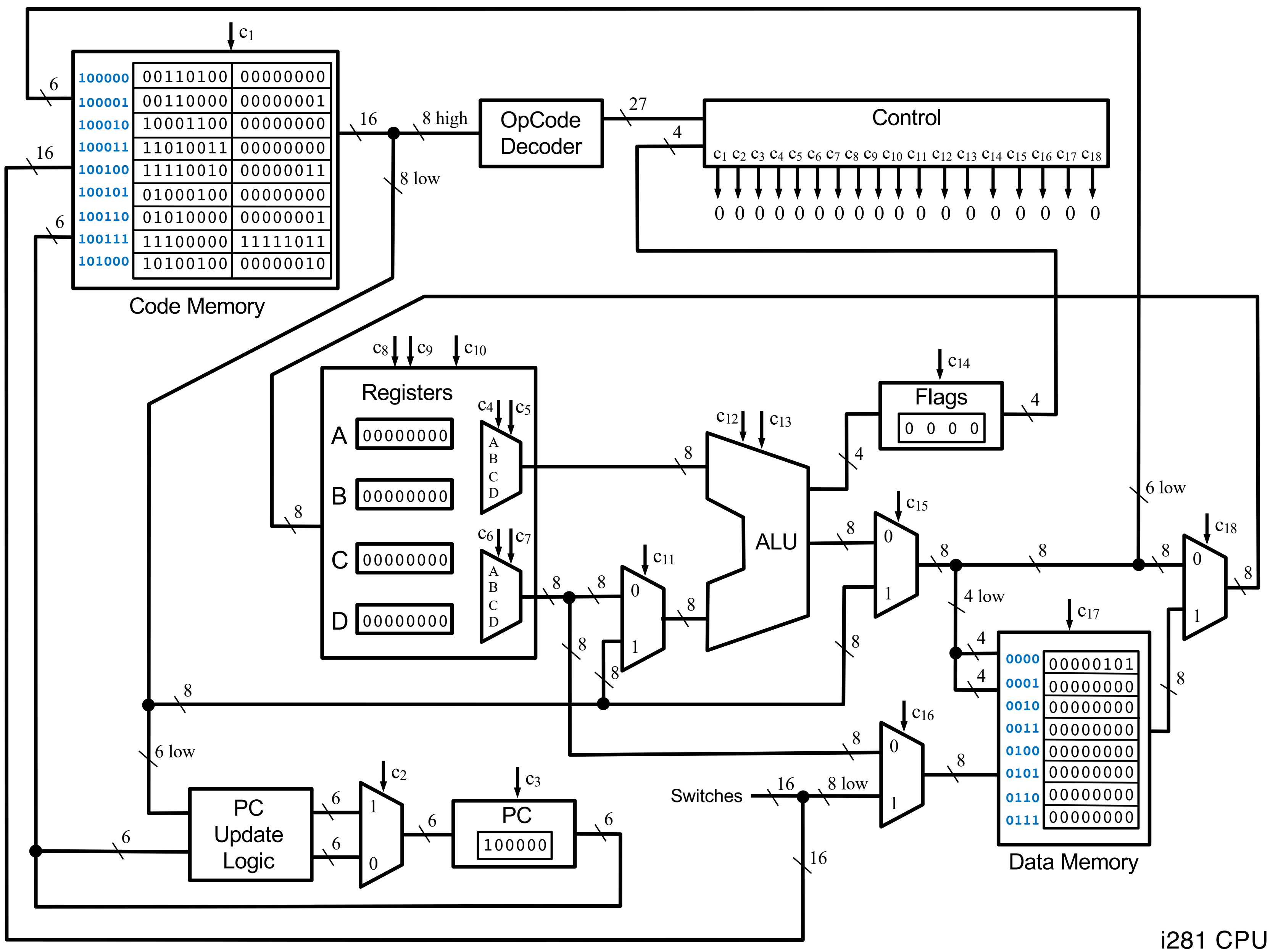
100000	00110100	00000000
100001	00110000	00000001
100010	10001100	00000000
100011	11010011	00000000
100100	11110010	00000011
100101	01000100	00000000
100110	01010000	00000001
100111	11100000	11111011
101000	10100100	00000010

0000	00000101
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000

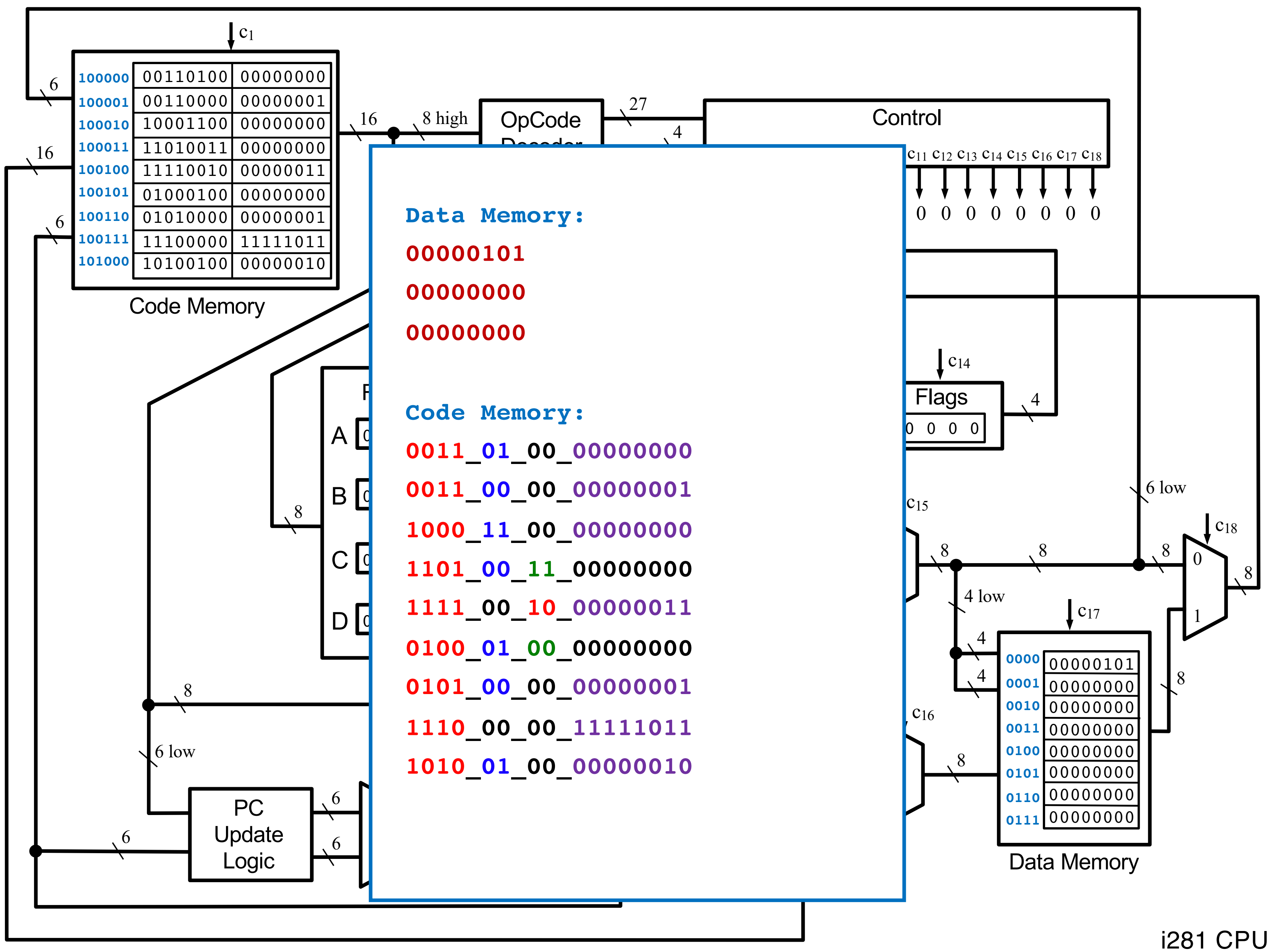
i281 CPU


Write Select
c1 Write Enable
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Input
Read Select
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Write Select
c8 c9
Write Enable
c10
Registers
A 00000000
B 00000000
C 00000000
D 00000000
Port0 Read Select
c4 c5
A B C D
Port1 Read Select
c6 c7
A B C D
Input
ALU Select
c12 c13
ALU
Write Enable
c14
Flags
0 0 0 0
ALU Result
c15 Mux
0 1
ALU Source
c11 Mux
0 1
REG Writeback Mux
c18
0 1
Write Enable
c17
Read Select
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
DMEM Input Mux
c16
0 1
Write Select
Input
Switches
PC Mux
c2
1 0
Write Enable
c3
PC
100000
PC Update Logic

i281 CPU

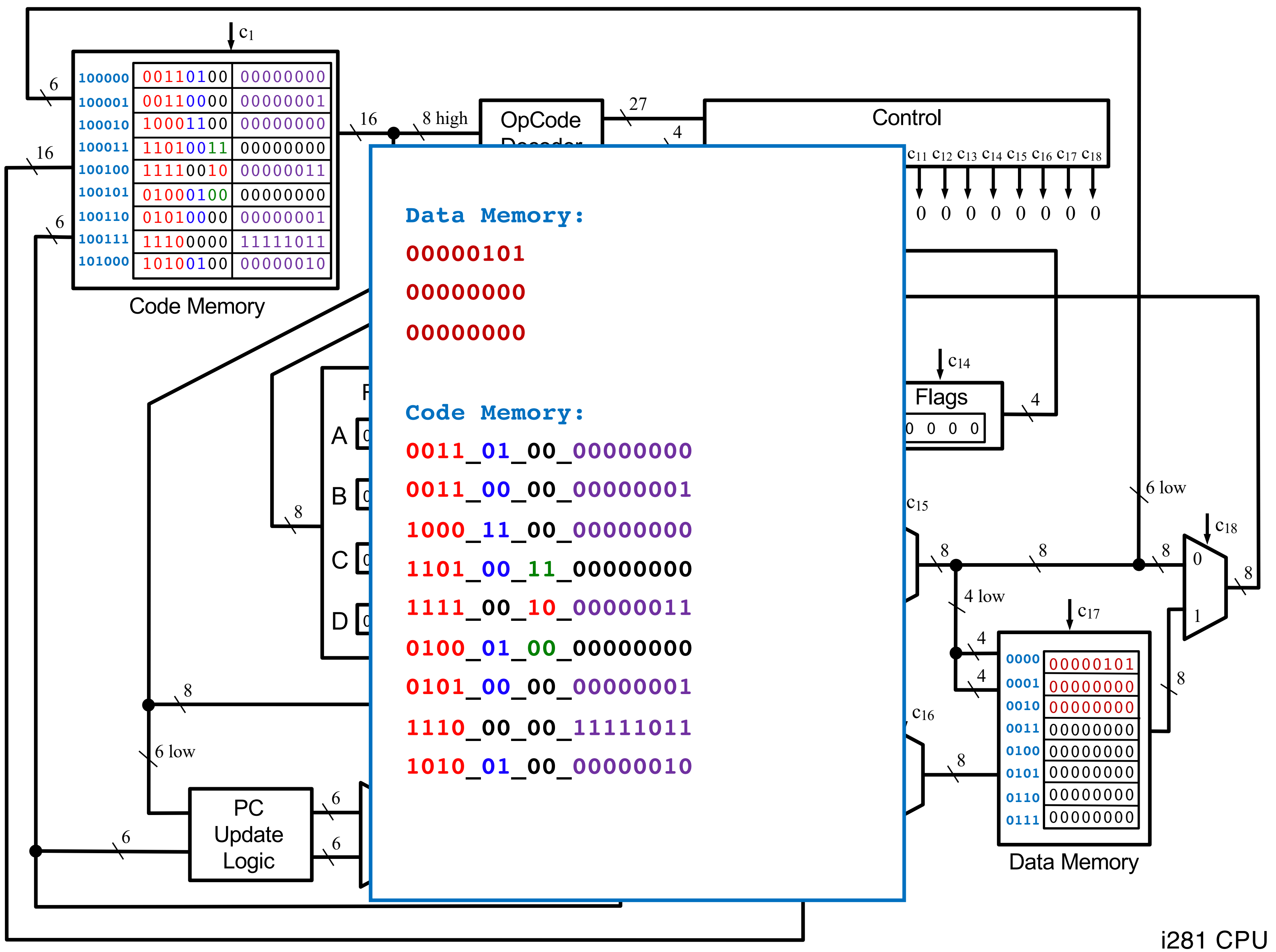

c1
Code Memory
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
16
8 high
8 low
OpCode Decoder
27
4
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Registers
c8 c9 c10
A 00000000
B 00000000
C 00000000
D 00000000
c4 c5
c6 c7
A B C D
c11
c12 c13
ALU
Flags
c14
0 0 0 0
c15
c16
c17
c18
6 low
4 low
Data Memory
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Switches
8 low
PC Update Logic
c2
c3
PC
100000
6 low
0
1
8
6
16
4

i281 CPU



```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

i281 CPU



Data Memory:

00000101

00000000

00000000

Code Memory:

0011_01_00_00000000

0011_00_00_00000001

1000_11_00_00000000

1101_00_11_00000000

1111_00_10_00000011

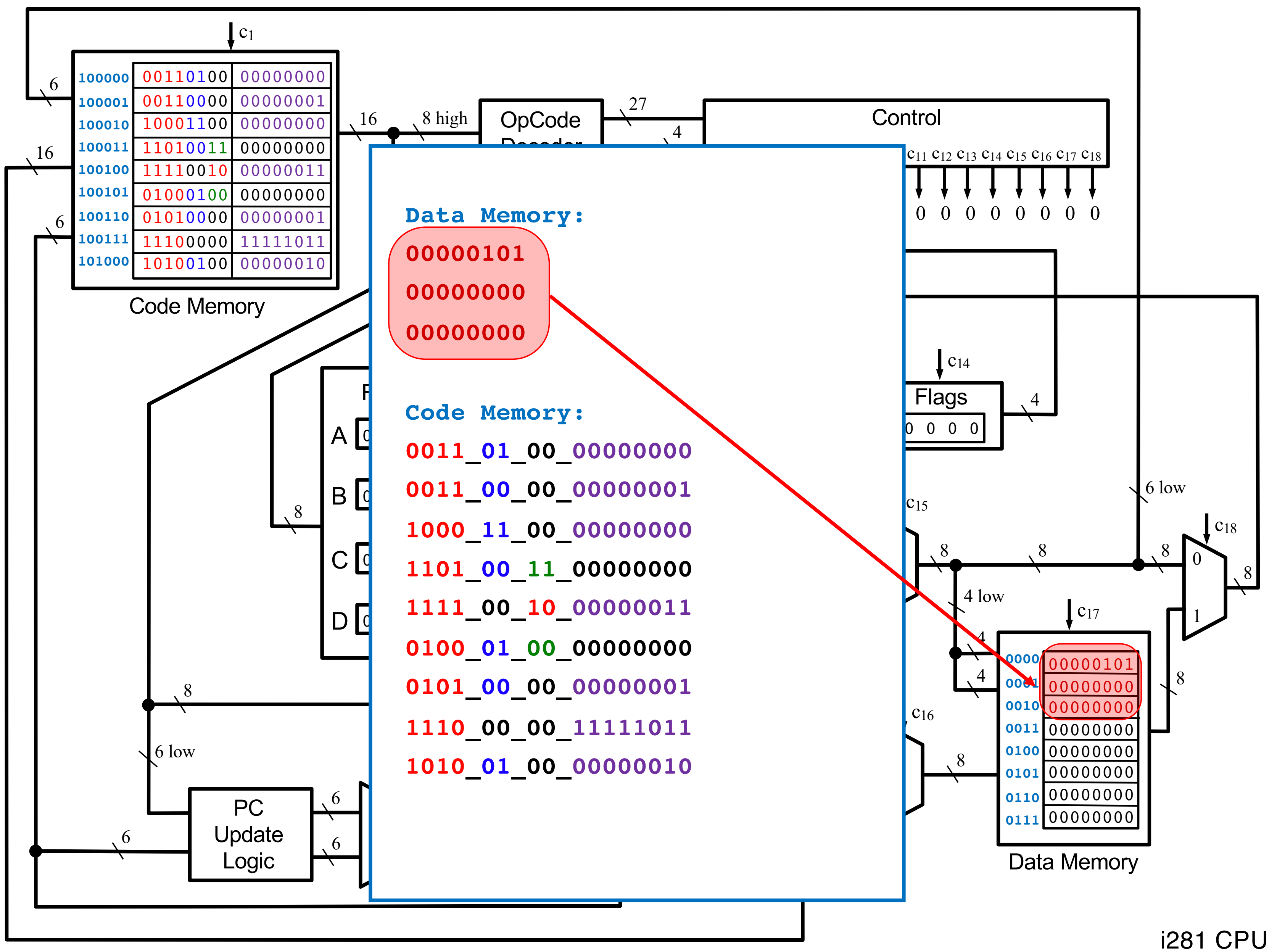
0100_01_00_00000000

0101_00_00_00000001

1110_00_00_11111011

1010_01_00_00000010

i281 CPU

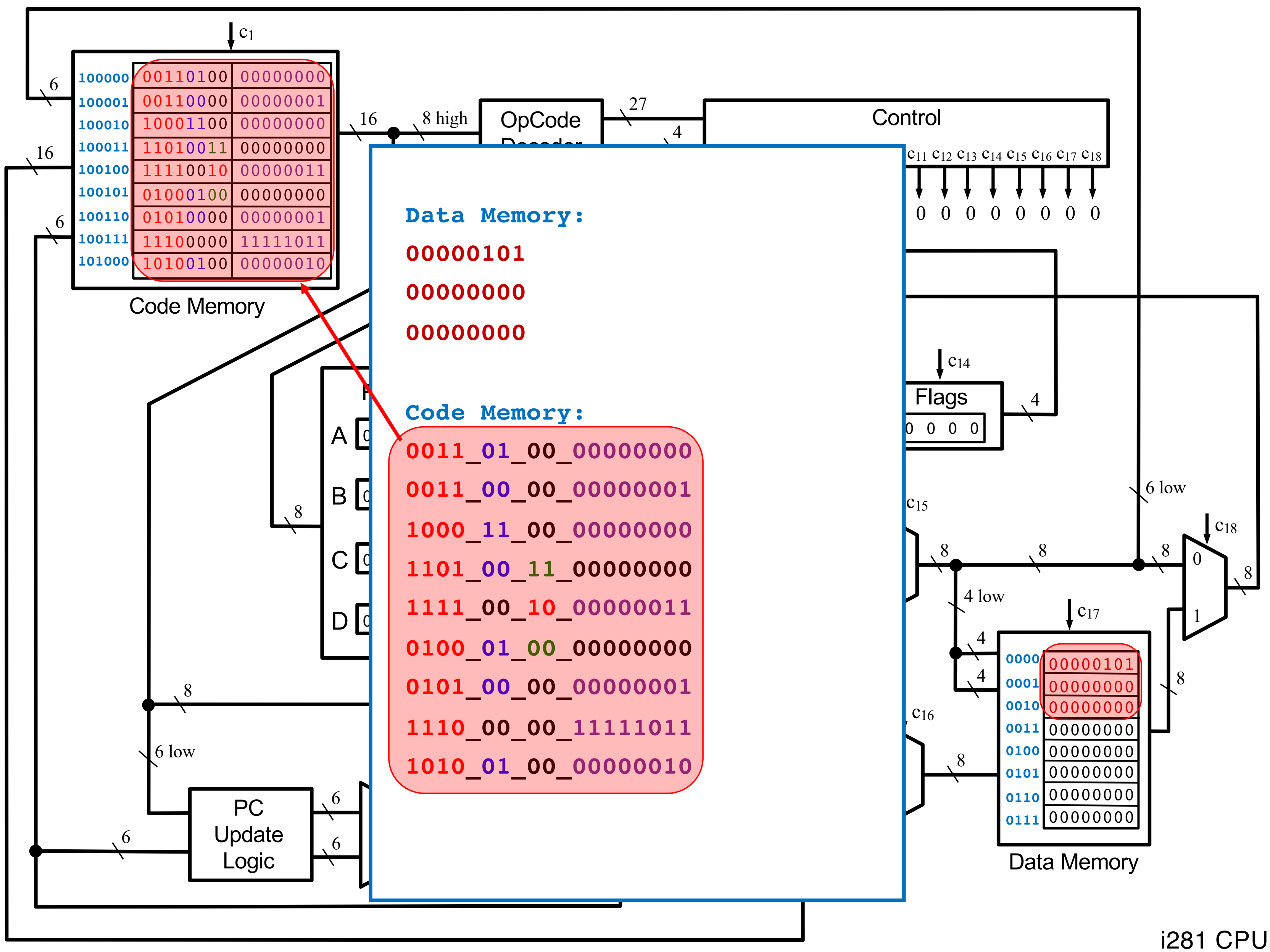


100000	00110100	00000000
100001	00110000	00000001
100010	10001100	00000000
100011	11010011	00000000
100100	11110010	00000011
100101	01000100	00000000
100110	01010000	00000001
100111	11100000	11111011
101000	10100100	00000010

0000	00000101
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000

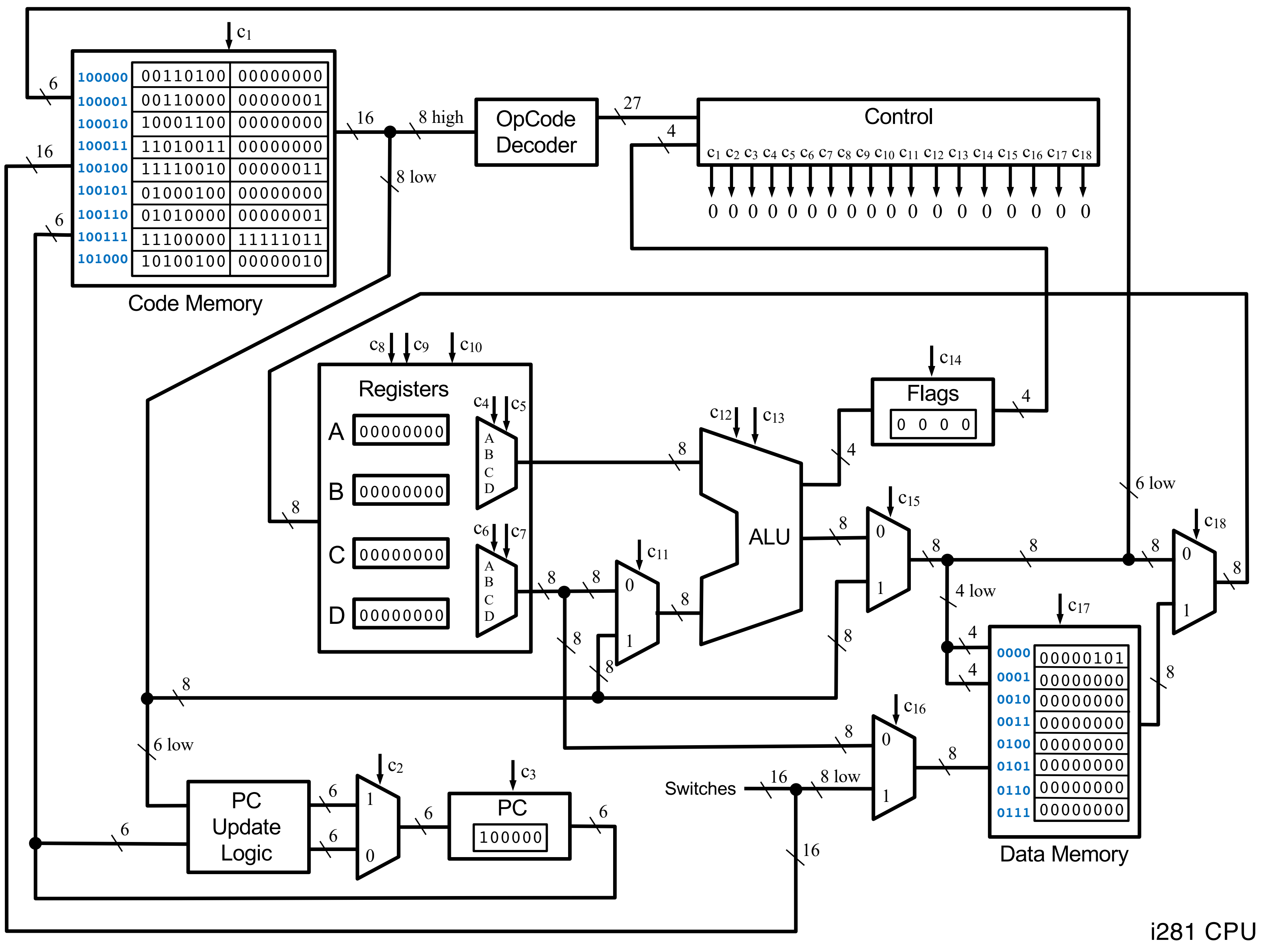
```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

i281 CPU



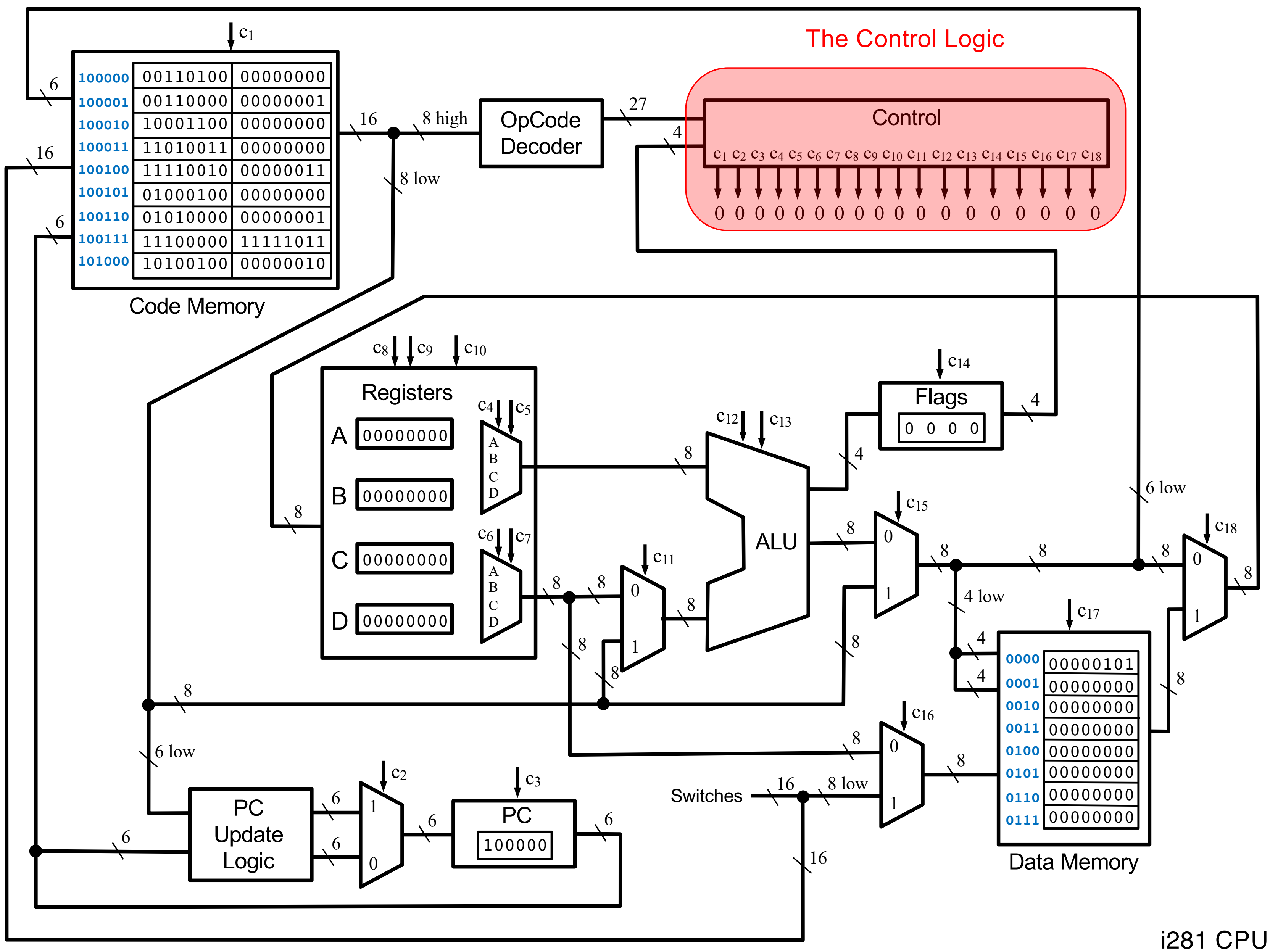
```
Data Memory:
00000101
00000000
00000000

Code Memory:
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

i281 CPU


c1
Code Memory
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
16
8 high
8 low
OpCode Decoder
27
4
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Registers
c8 c9 c10
c4 c5
c6 c7
A 00000000
B 00000000
C 00000000
D 00000000
A B C D
c11
0 1
c12 c13
ALU
c14
Flags
0 0 0 0
c15
c18
6 low
4 low
c17
Data Memory
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
c16
Switches
8 low
PC Update Logic
6 low
c2
c3
PC
100000

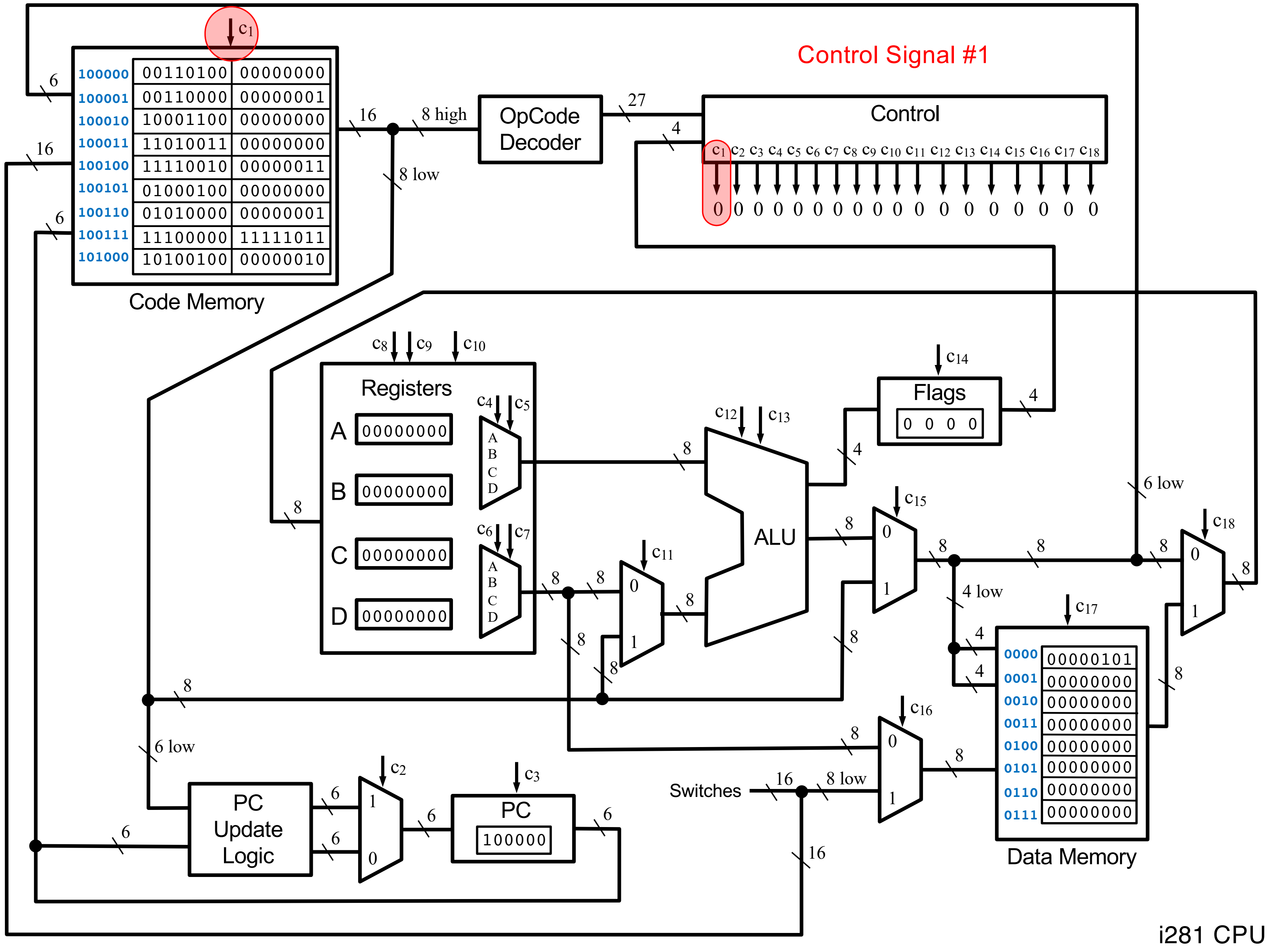
i281 CPU

The CPU Control Logic

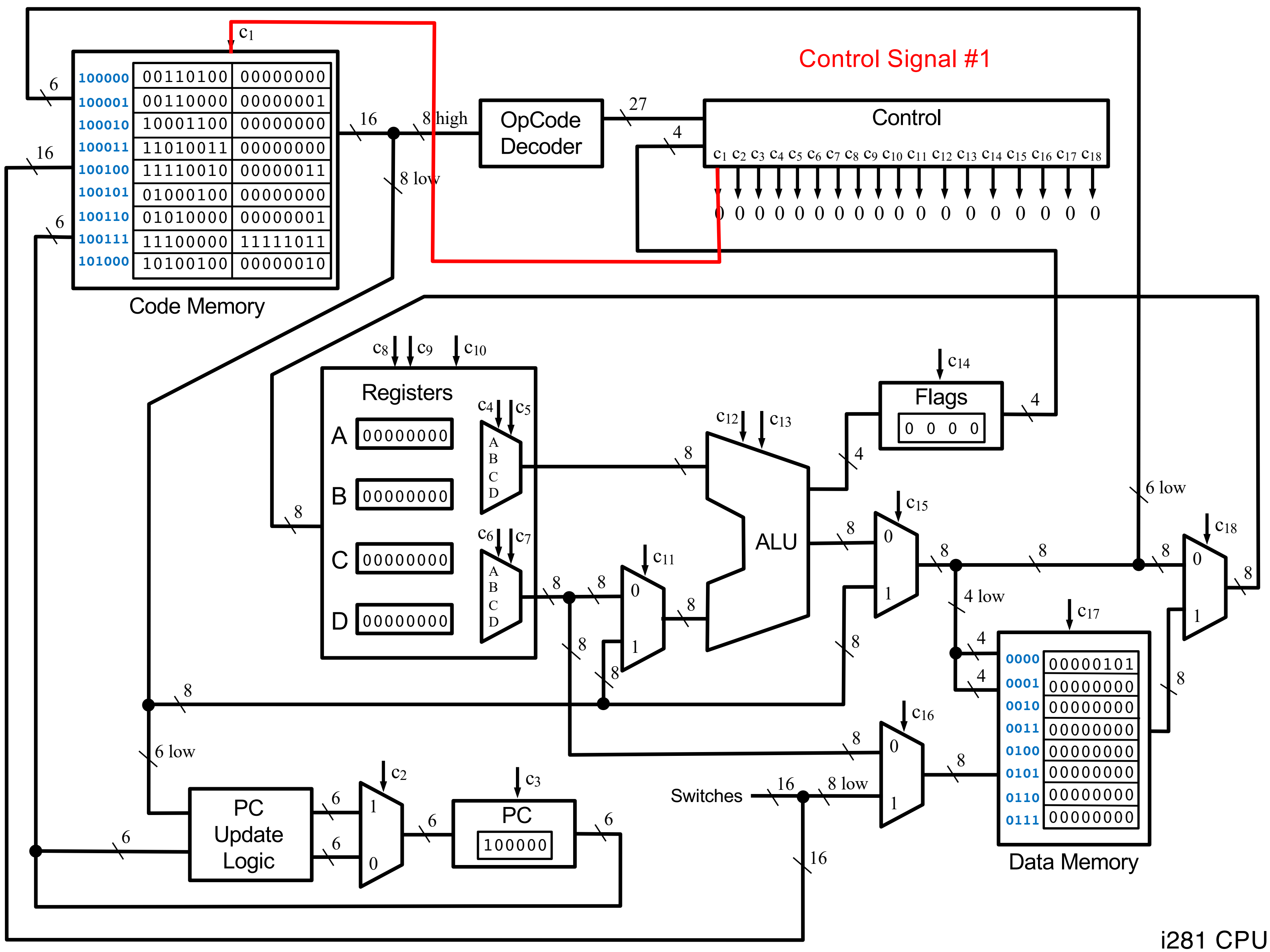

The Control Logic
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OpCode Decoder
27
4
16
8 high
8 low
6
Code Memory
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Registers
A 00000000
B 00000000
C 00000000
D 00000000
A B C D
ALU
Flags
0 0 0 0
6 low
4 low
8
Data Memory
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Switches
16
8 low
PC Update Logic
PC
100000
0
1

i281 CPU

Control Signal #1

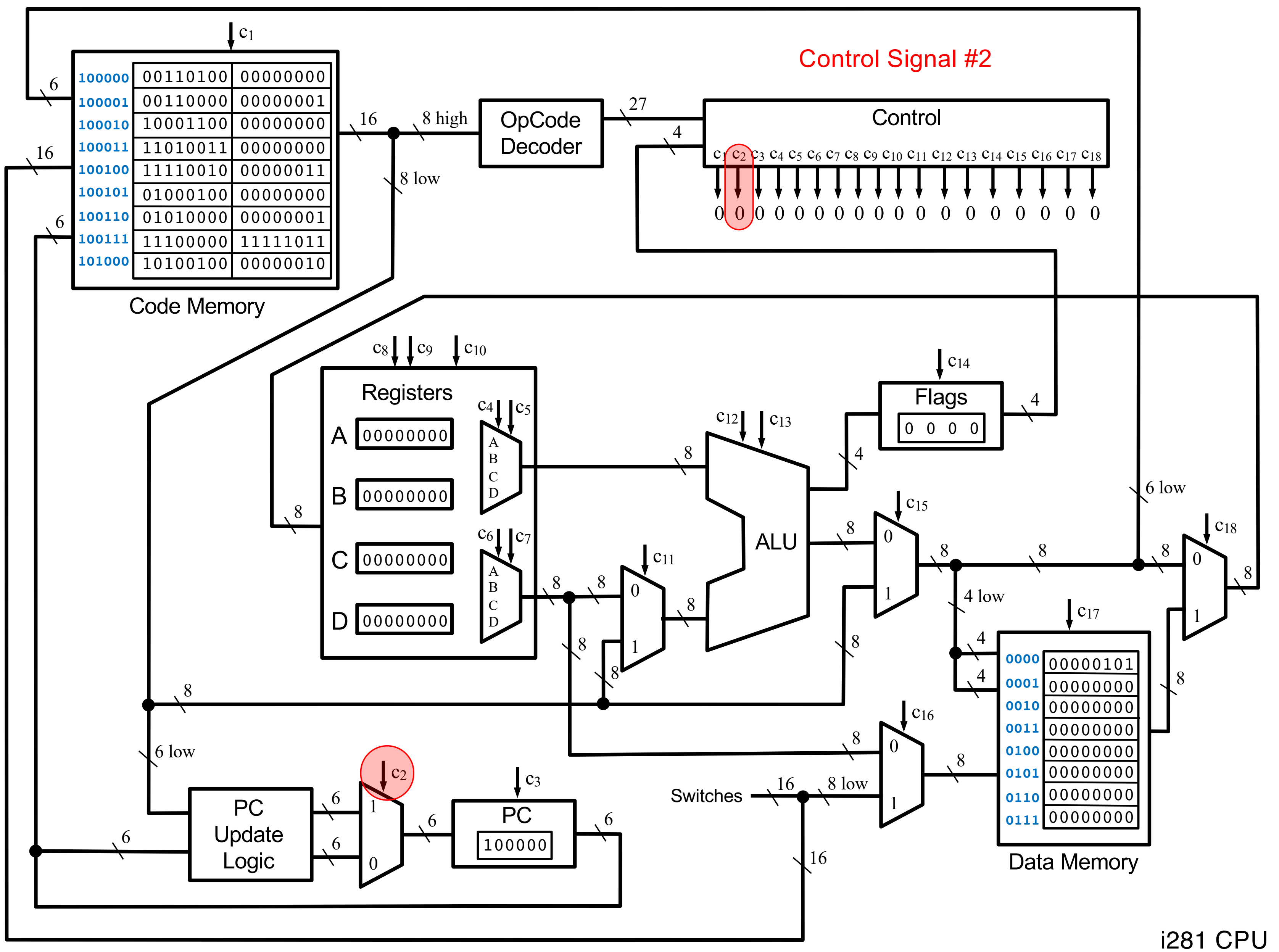


Control Signal #1



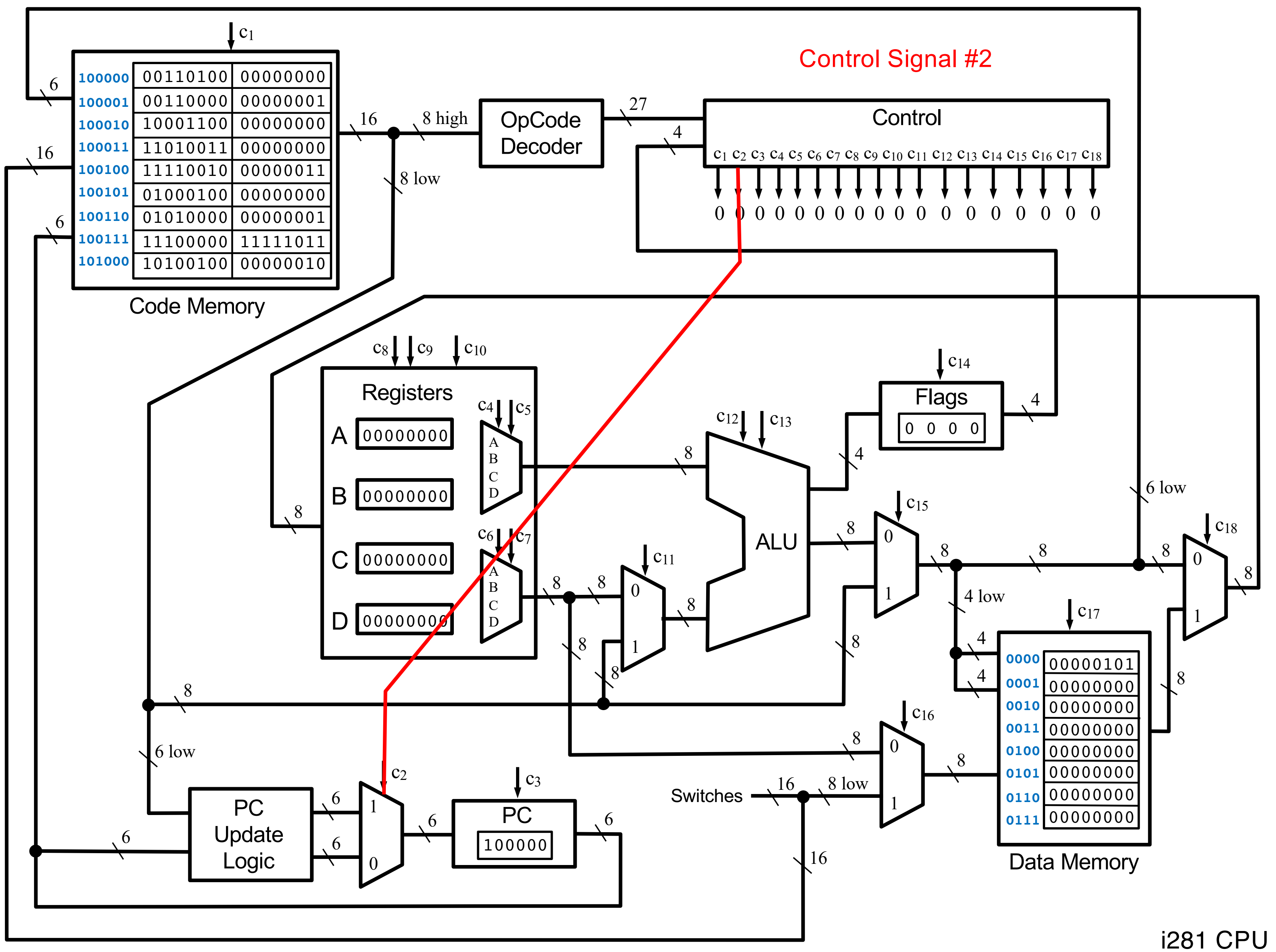
i281 CPU

Control Signal #2

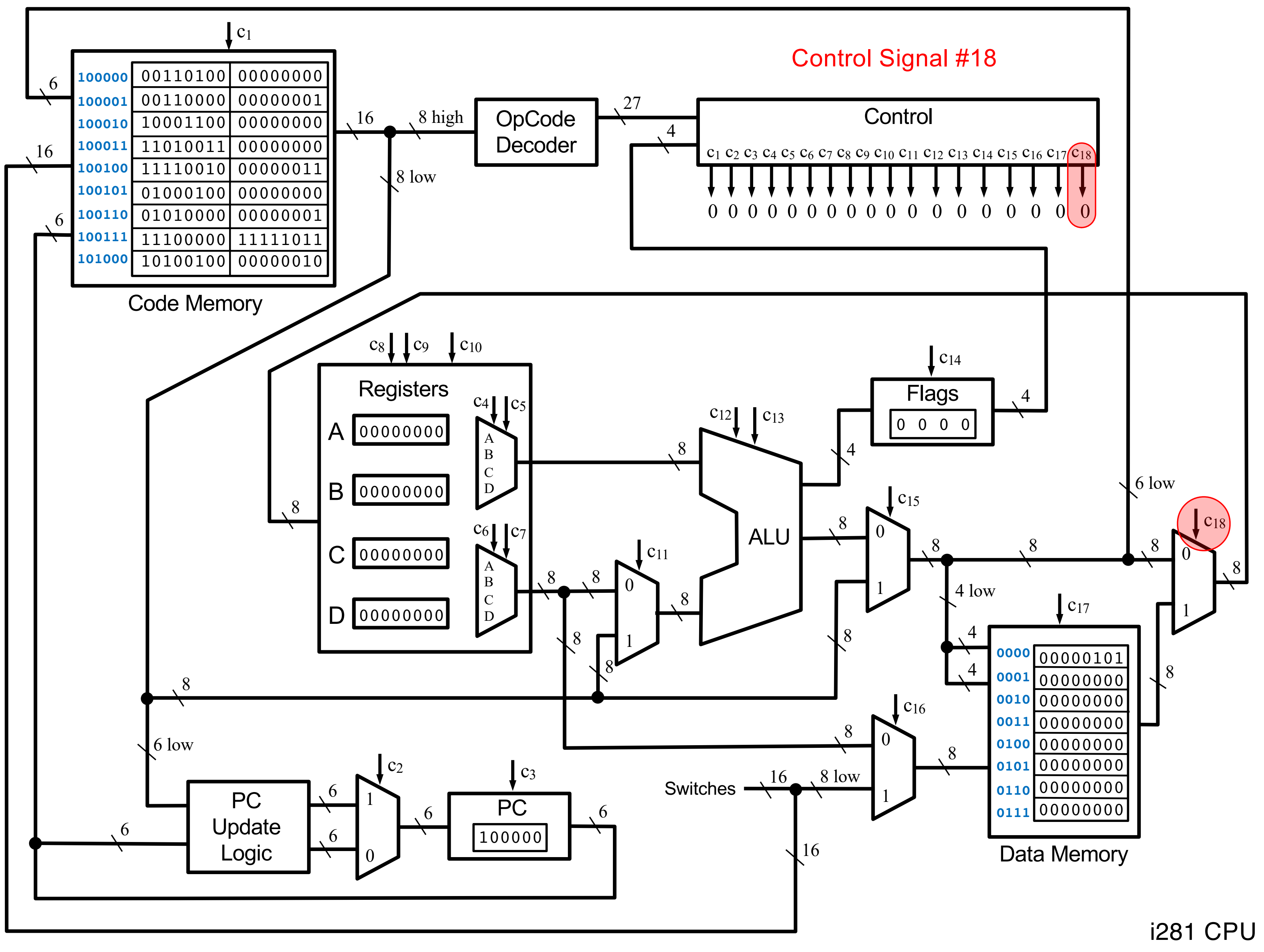


i281 CPU

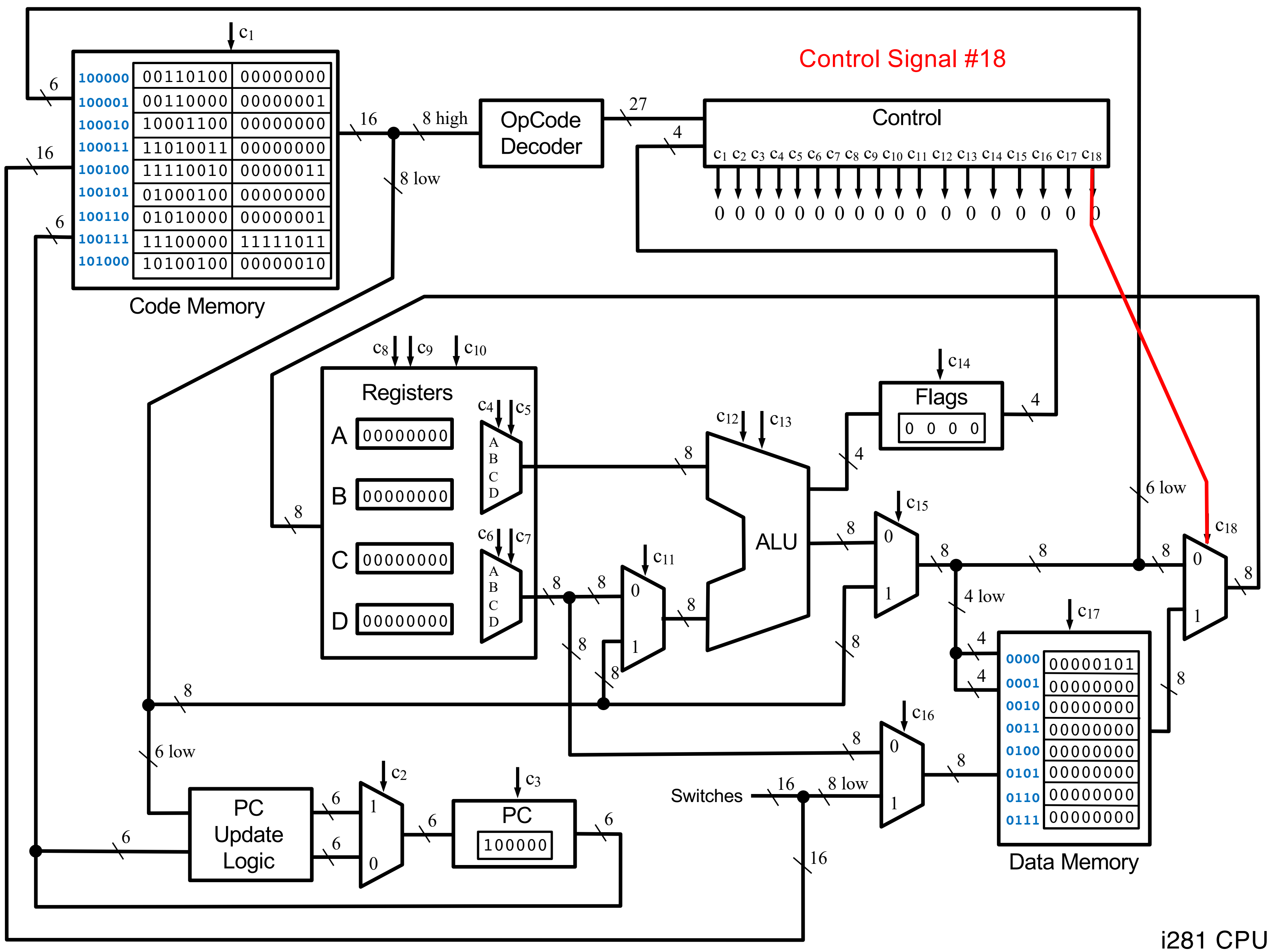
Control Signal #2



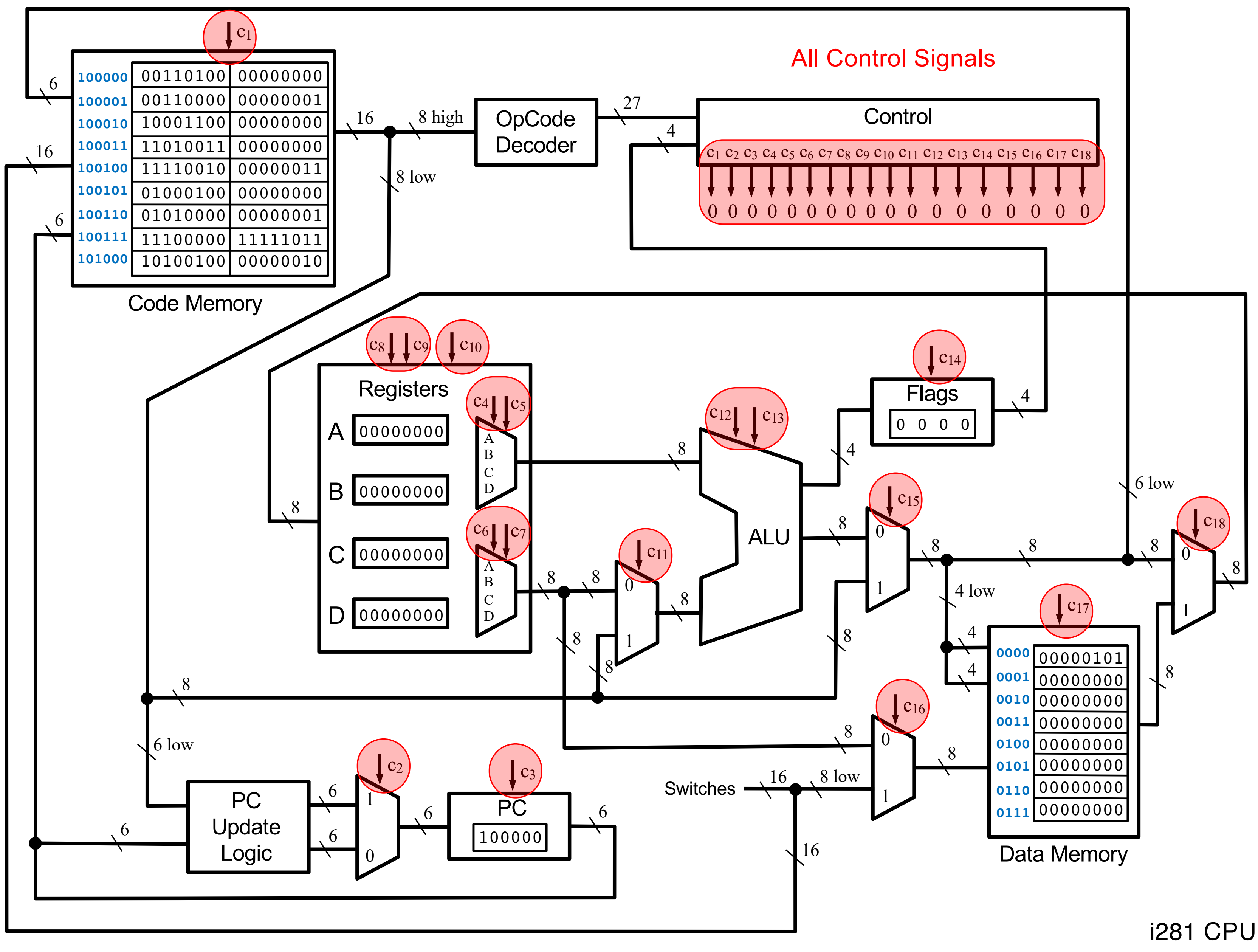
i281 CPU


Control Signal #18
Code Memory
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Registers
A 00000000
B 00000000
C 00000000
D 00000000
ALU
Flags
0 0 0 0
Data Memory
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
PC Update Logic
PC
100000
Switches
8 high
8 low
6 low
4 low
i281 CPU

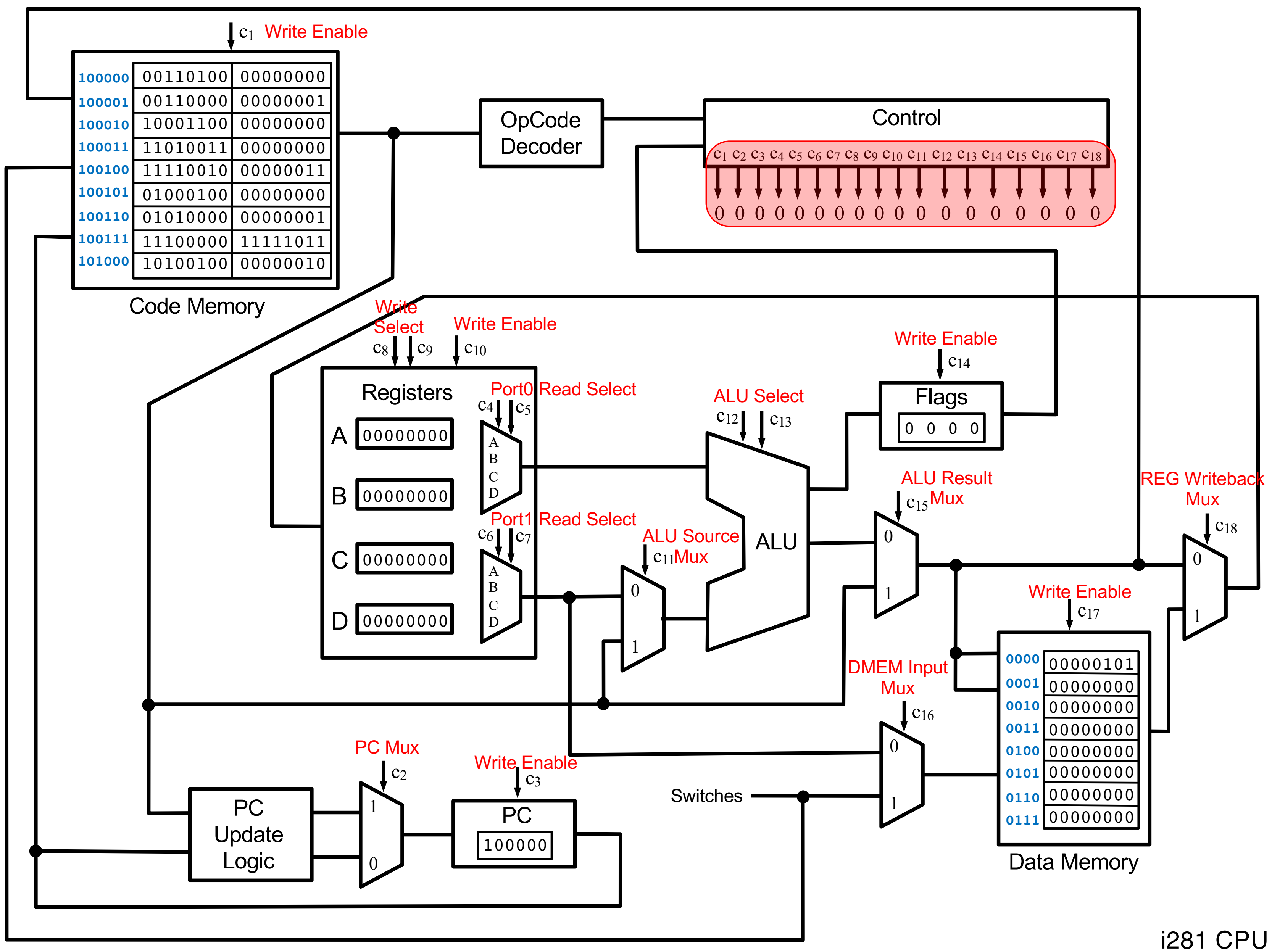
Control Signal #18



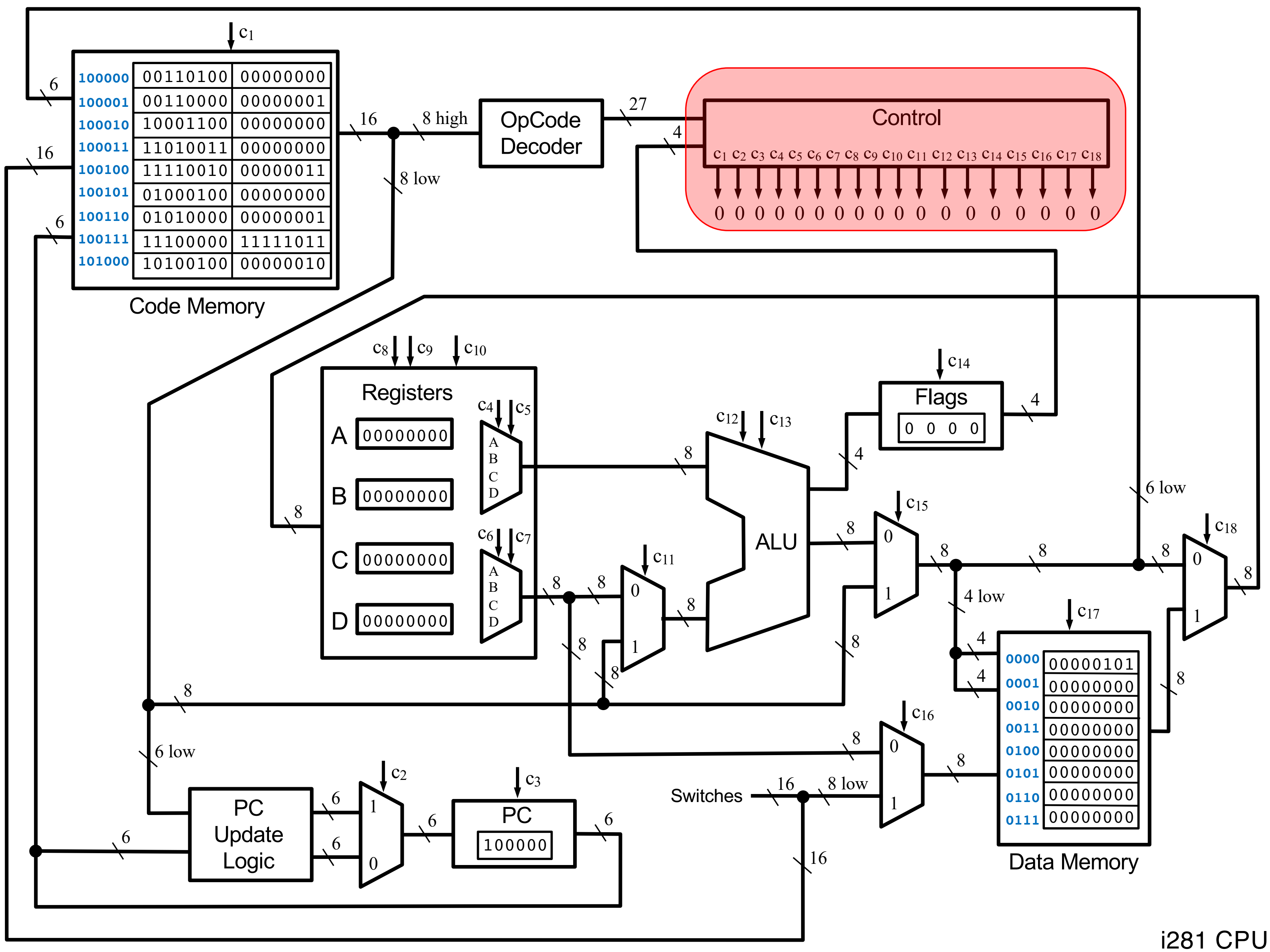
i281 CPU


All Control Signals
Code Memory
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Registers
A 00000000
B 00000000
C 00000000
D 00000000
ALU
Flags
0 0 0 0
Data Memory
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
PC Update Logic
PC
100000
Switches
8 high
8 low
6 low
4 low
27
16

i281 CPU


c1 Write Enable
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Write Select
c8 c9
Write Enable
c10
Registers
A 00000000
B 00000000
C 00000000
D 00000000
Port0 Read Select
c4 c5
A B C D
Port1 Read Select
c6 c7
A B C D
ALU Select
c12 c13
ALU
Write Enable
c14
Flags
0 0 0 0
ALU Result Mux
c15
0 1
ALU Source Mux
c11
0 1
REG Writeback Mux
c18
0 1
Write Enable
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
DMEM Input Mux
c16
0 1
Switches
PC Mux
c2
1 0
Write Enable
c3
PC
100000
PC Update Logic

i281 CPU


c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
16
8 high
8 low
OpCode Decoder
27
4
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
c8 c9 c10
Registers
A 00000000
B 00000000
C 00000000
D 00000000
c4 c5
c6 c7
A B C D
c11
c12 c13
ALU
c14
Flags
0 0 0 0
c15
6 low
4 low
c18
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c16
Switches
8 low
PC Update Logic
c2
c3
PC
100000
6 low

i281 CPU

The OPCODEs for this CPU

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

	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_ENABLE	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															

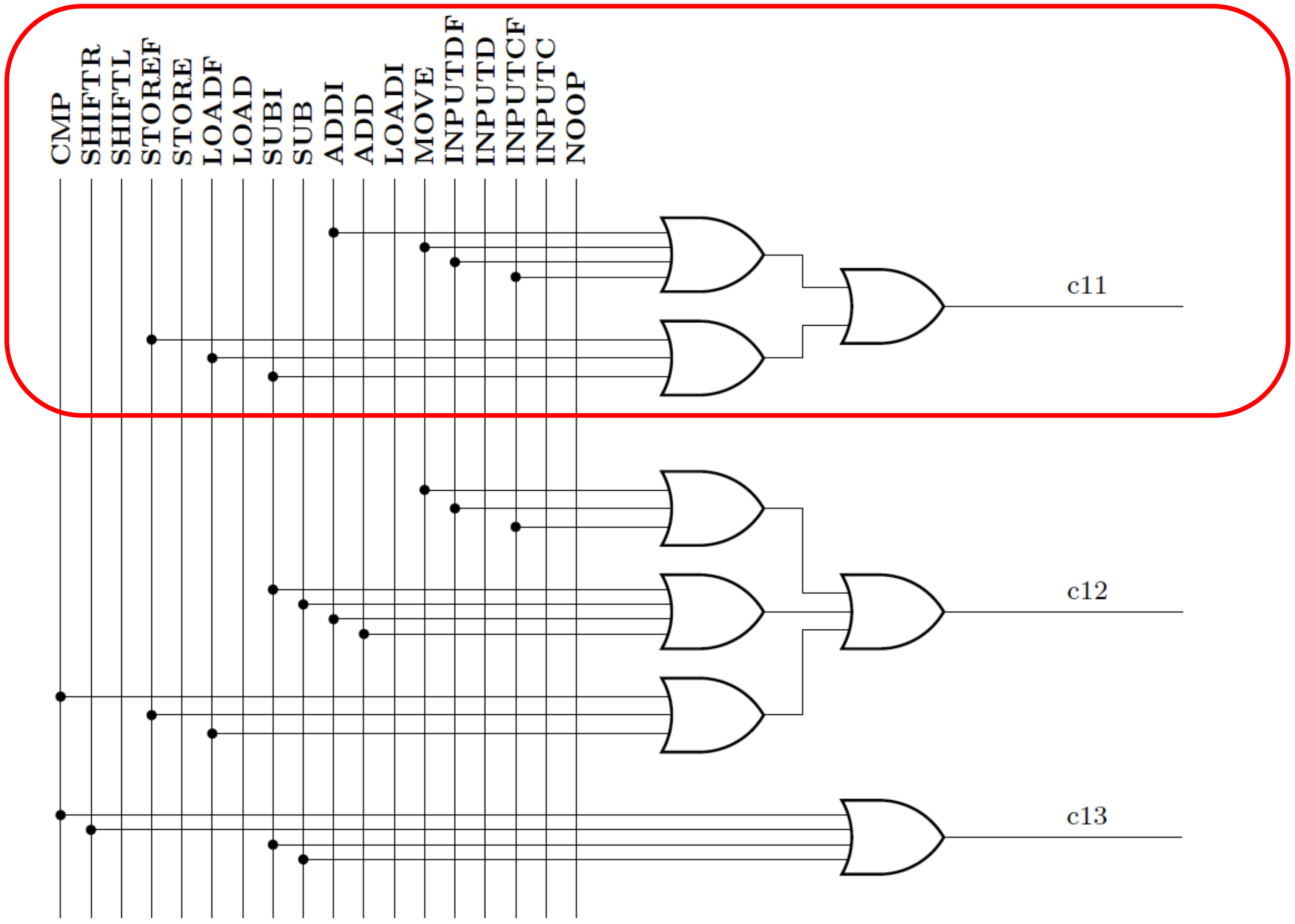
18 control lines

23 one-hot encoded OPCODEs

	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															

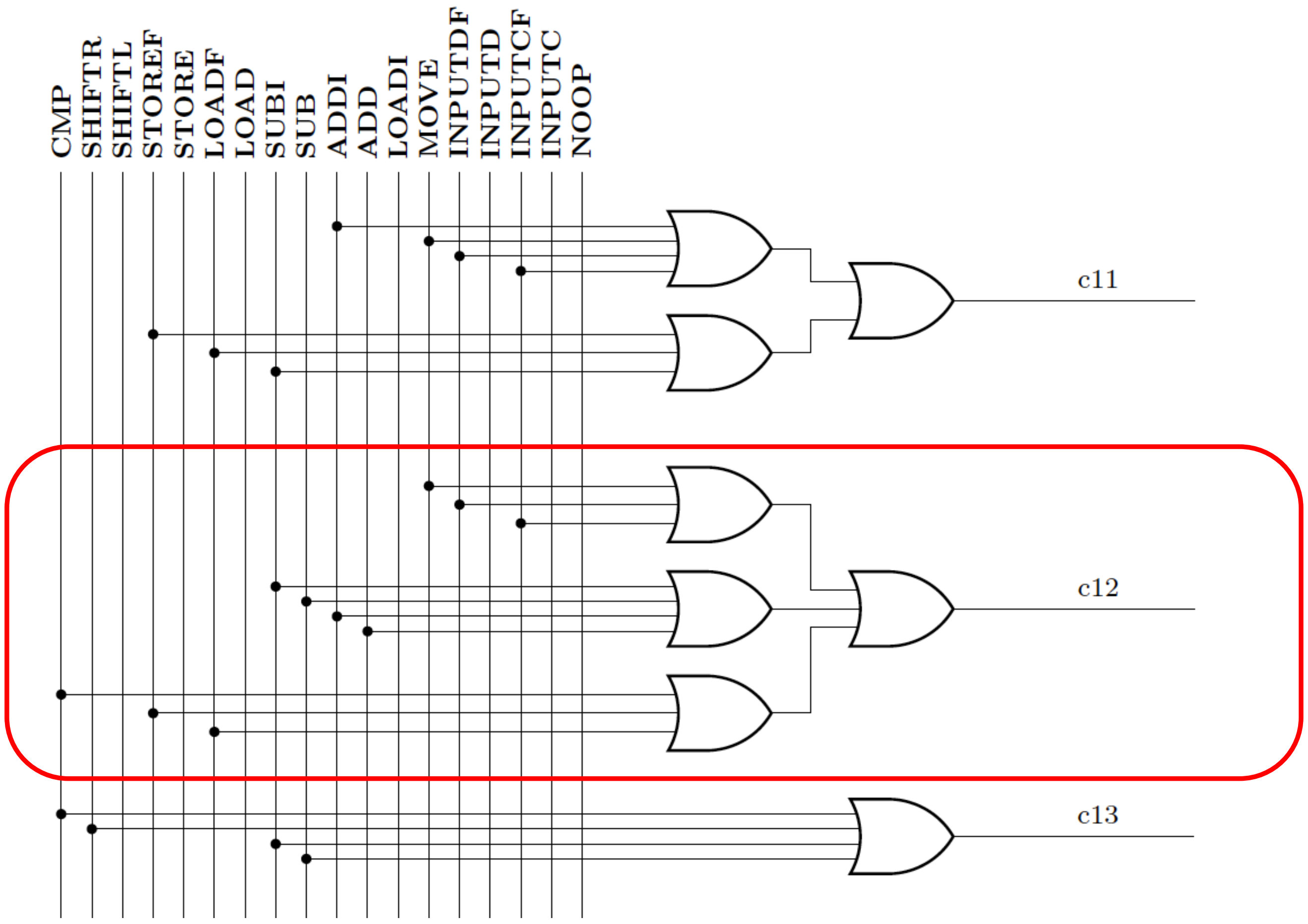
	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}
	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_EN	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	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 Wiring Diagram for c_{11}



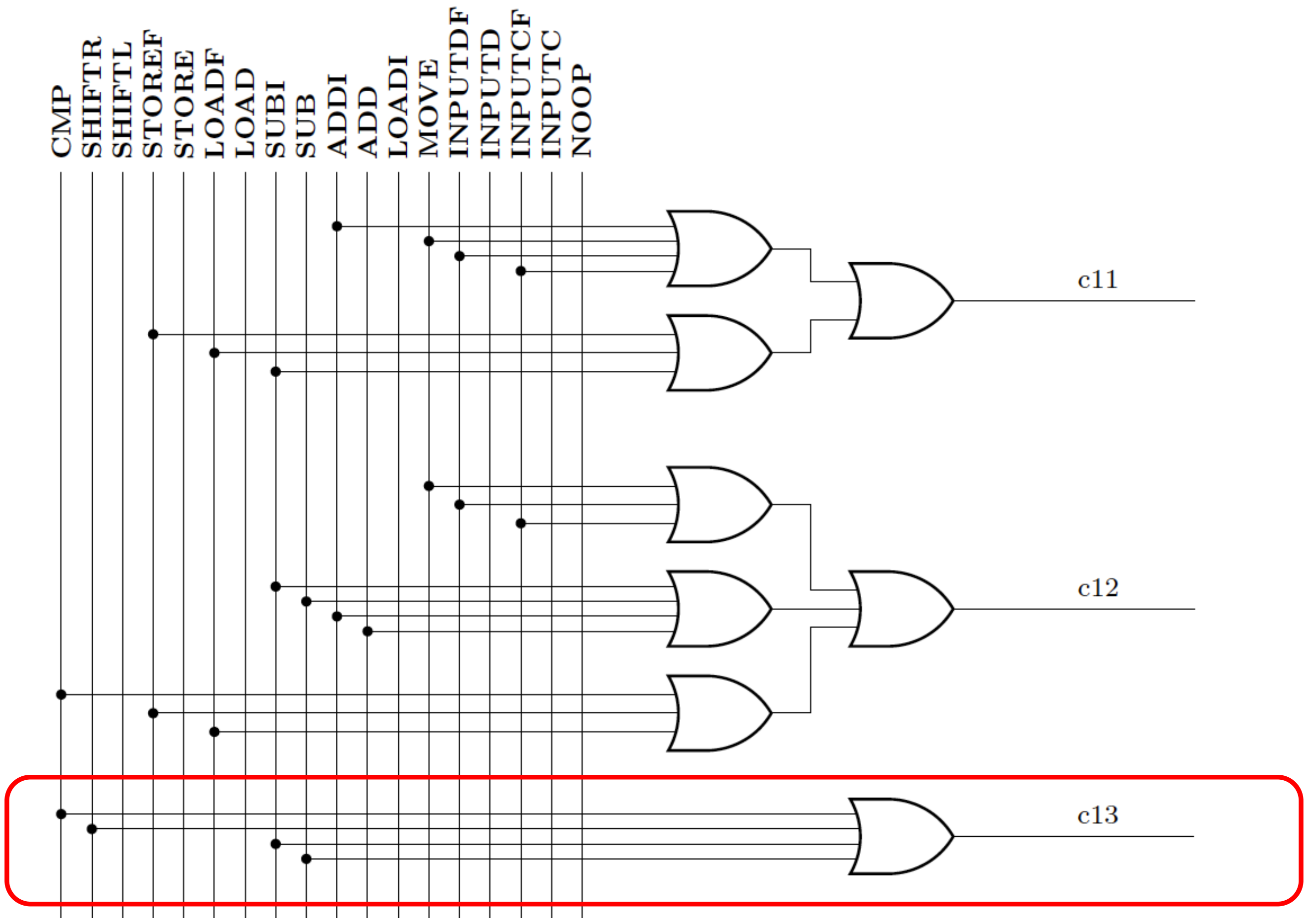
	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}
	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_EN	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	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 Wiring Diagram for c_{12}



	c_1 IMEM_WRITE_ENABLE	c_2 PROGRAM_COUNTER_MUX	c_3 PROGRAM_COUNTER_WRITE_EN	c_4 REGISTERS_PORT0_SELECT1	c_5 REGISTERS_PORT0_SELECT0	c_6 REGISTERS_PORT1_SELECT1	c_7 REGISTERS_PORT1_SELECT0	c_8 REGISTERS_WRITE_SELECT1	c_9 REGISTERS_WRITE_SELECT0	c_{10} REGISTERS_WRITE_ENABLE	c_{11} ALU_SOURCE_MUX	c_{12} ALU_SELECT1	c_{13} ALU_SELECT0	c_{14} FLAGS_WRITE_ENABLE	c_{15} ALU_RESUT_MUX	c_{16} DMEM_INPUT_MUX	c_{17} DMEM_WRITE_ENABLE	c_{18} 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 Wiring Diagram for c_{13}



	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															

LOADI affects only

C_3 , C_8 , C_9 , C_{10} , and C_{15}

All others are set 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				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

C_8 and C_9 depend on the instruction and the register that it uses.

	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															

C_8	C_9	Register
0	0	A
0	1	B
1	0	C
1	1	D

	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															

Simulation of the Program Execution

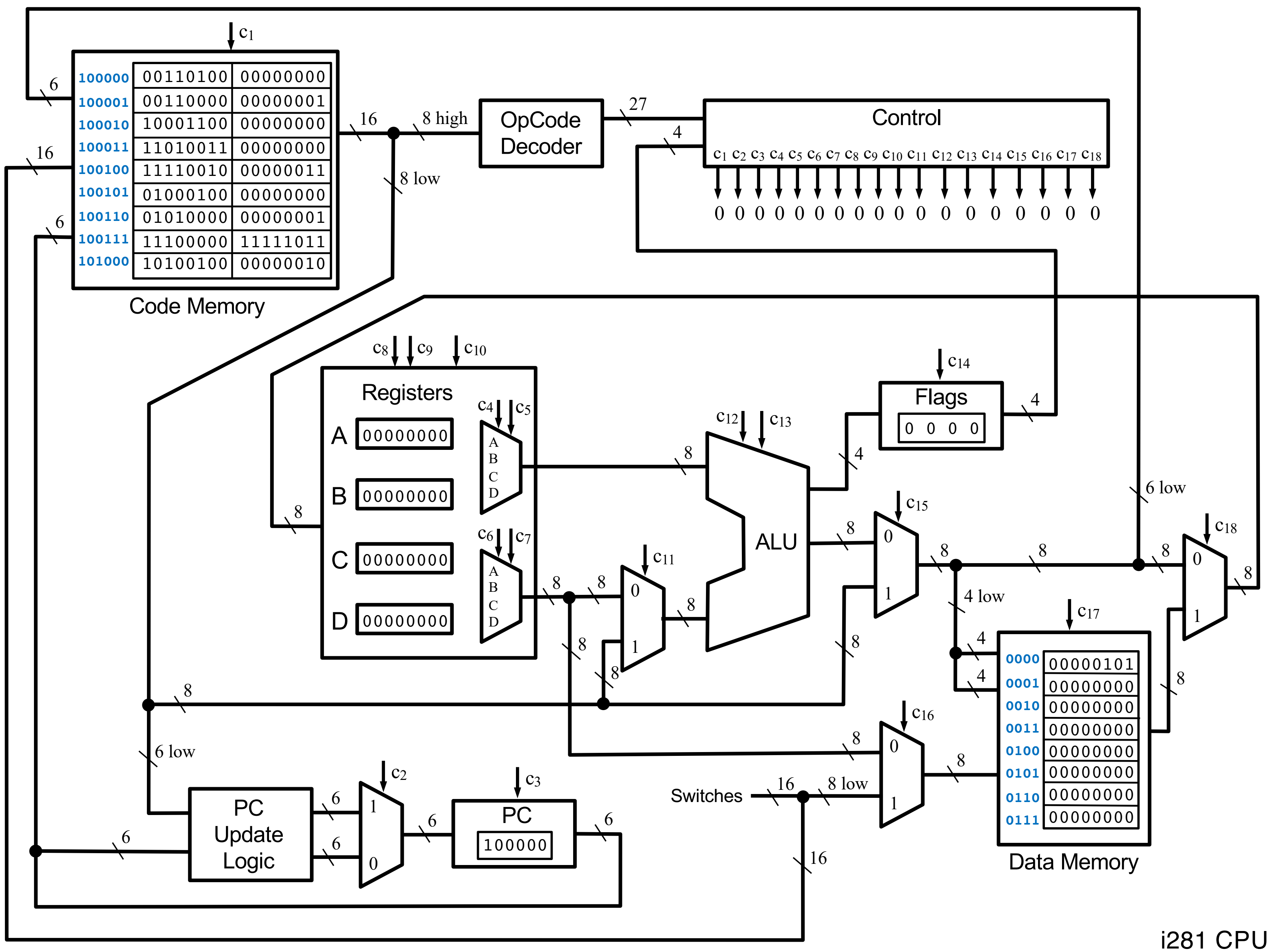
Add the numbers from 1 to 5

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





int main()
{
        int N=5;
        int i, sum;

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

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

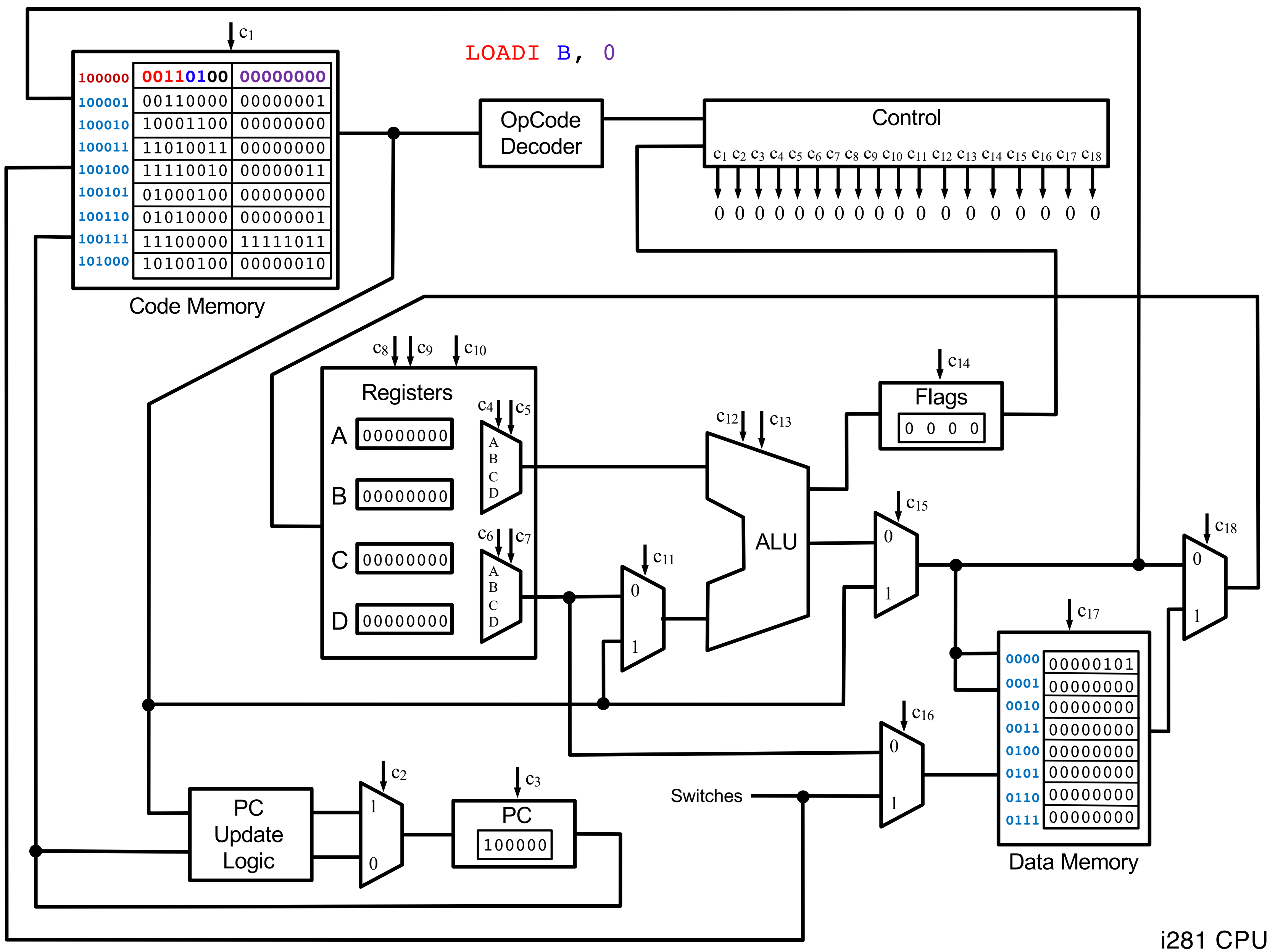
```
; Assembly Version

.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
```

Mapping Assembly to Machine Code

```
.data                                      Data Memory:
N         BYTE    5                        00000101
i         BYTE    ?                        00000000
sum       BYTE    ?                        00000000

.code                                      Code Memory:
          LOADI   B, 0                     0011_01_00_00000000
          LOADI   A, 1                     0011_00_00_00000001
          LOAD    D, [N]                   1000_11_00_00000000
Loop:     CMP     A, D                     1101_00_11_00000000
          BRG     End                      1111_00_10_00000011
Add:      ADD     B, A                     0100_01_00_00000000
          ADDI    A, 1                     0101_00_00_00000001
          JUMP    Loop                     1110_00_00_11111011
End:      STORE   [sum], B                 1010_01_00_00000010
```

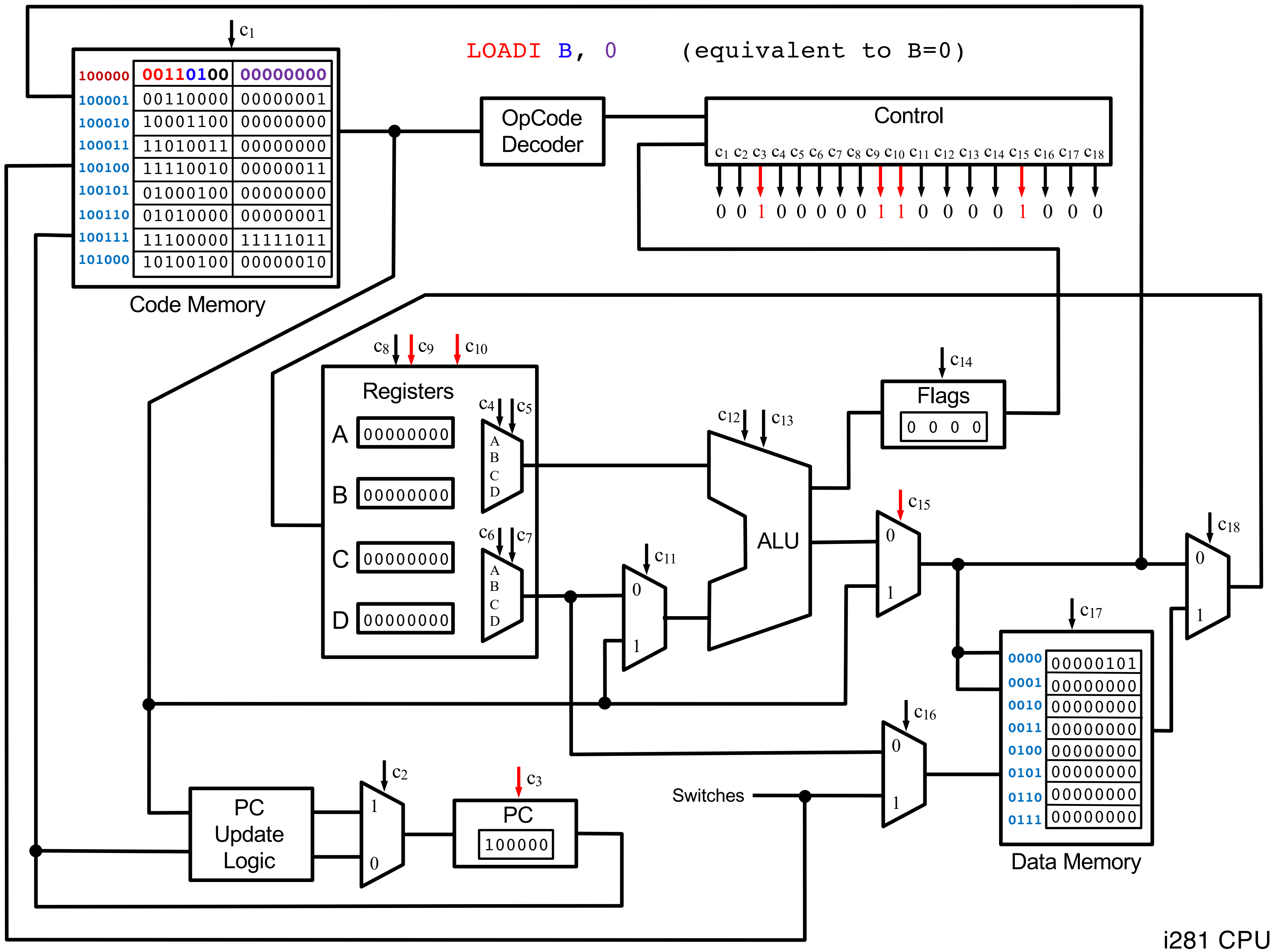

Code Memory
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
OpCode Decoder
8 high
8 low
27
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Registers
c8 c9 c10
A 00000000
B 00000000
C 00000000
D 00000000
c4 c5 c6 c7
ALU
c12 c13
Flags
c14
0 0 0 0
c11
c15
c16
c17
c18
6 low
4 low
Data Memory
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Switches
8 low
PC Update Logic
c2
PC
c3
100000

i281 CPU


LOADI B, 0
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
c8 c9 c10
Registers
A 00000000
B 00000000
C 00000000
D 00000000
c4 c5 c6 c7
A B C D
A B C D
c11
0 1
c12 c13
ALU
c14
Flags
0 0 0 0
c15
0 1
c18
0 1
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c16
0 1
Switches
PC Update Logic
c2
1 0
c3
PC
100000

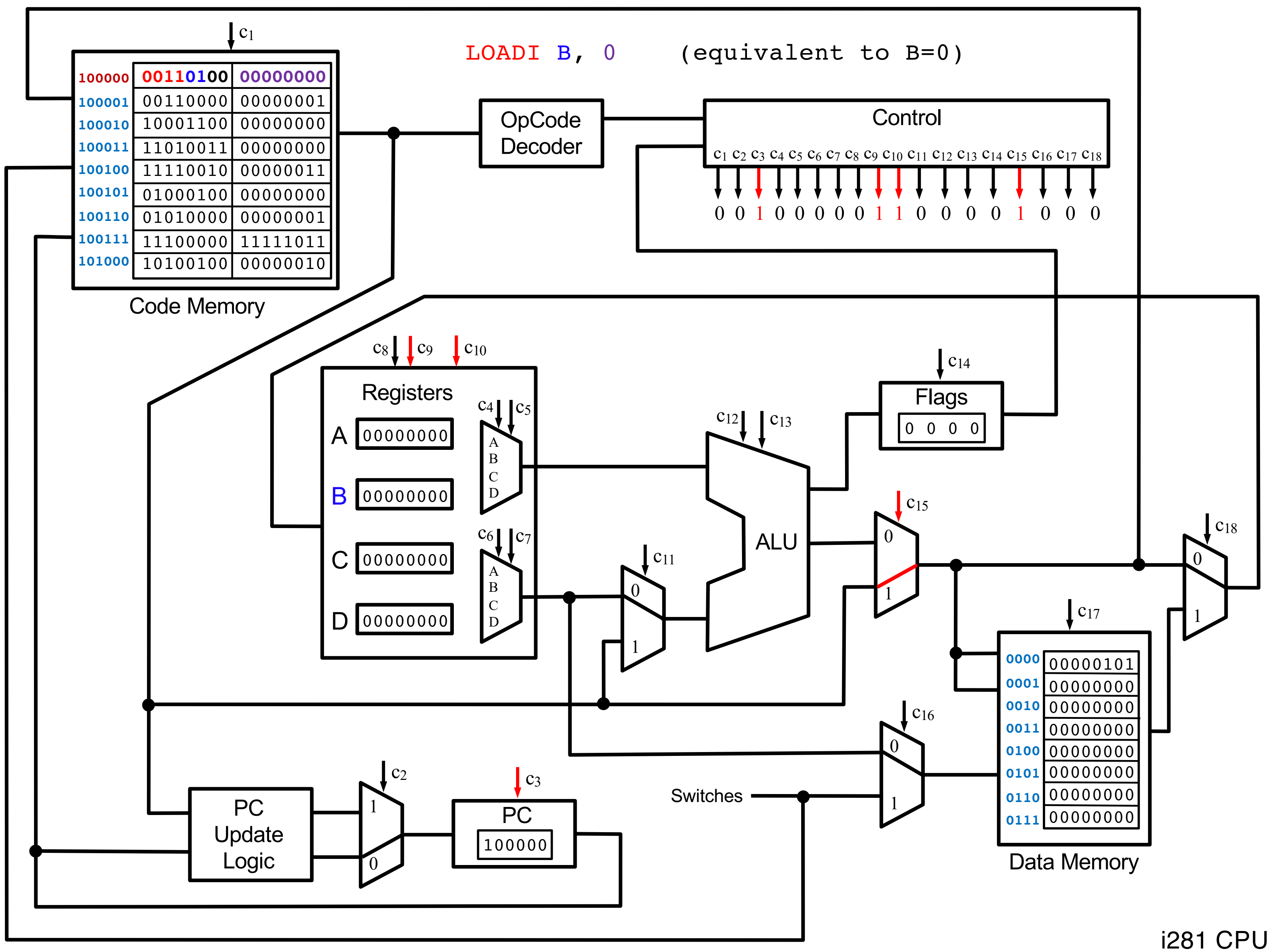
i281 CPU

LOADI B, 0   (equivalent to B=0)



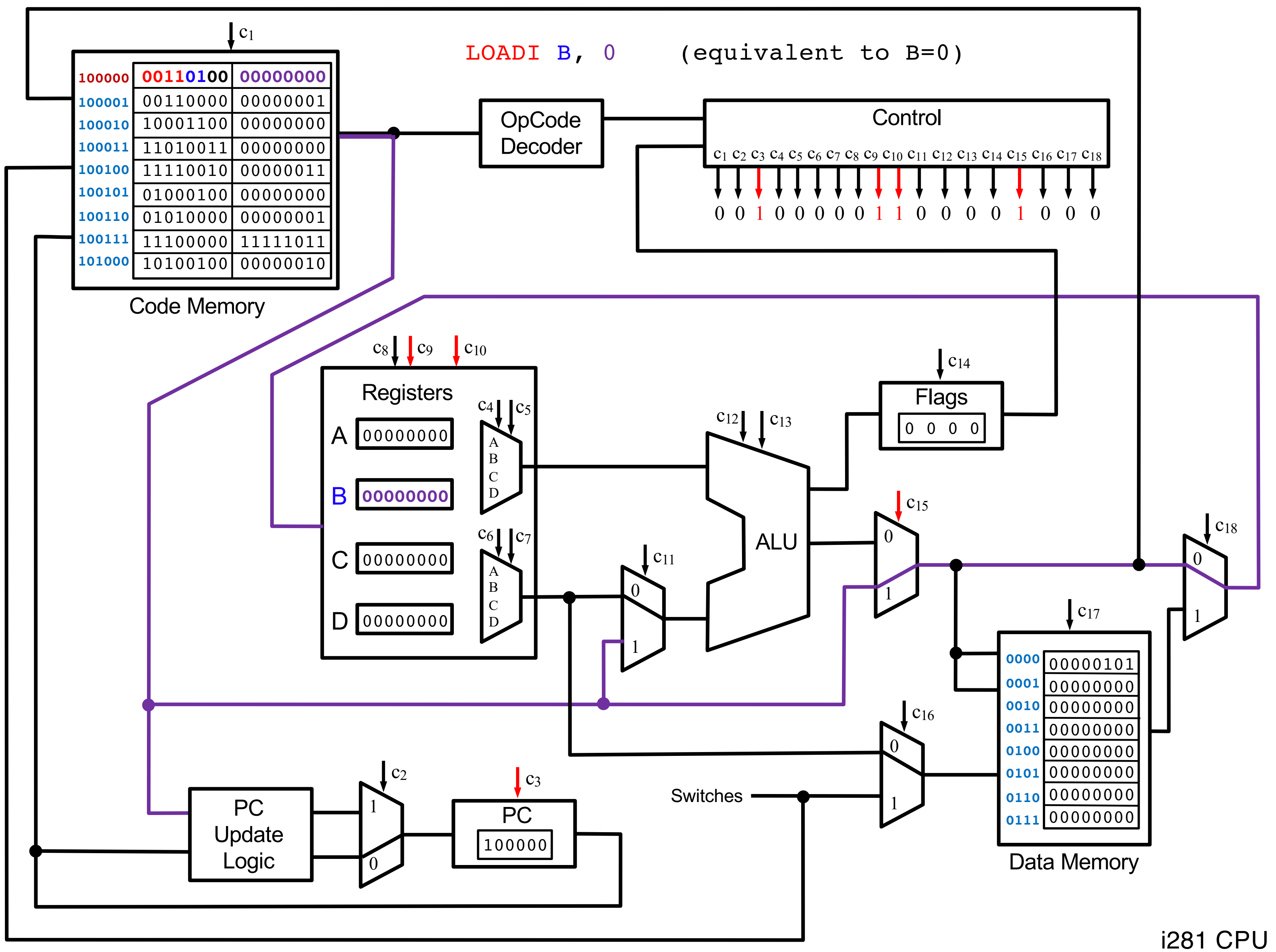
i281 CPU

LOADI B, 0   (equivalent to B=0)



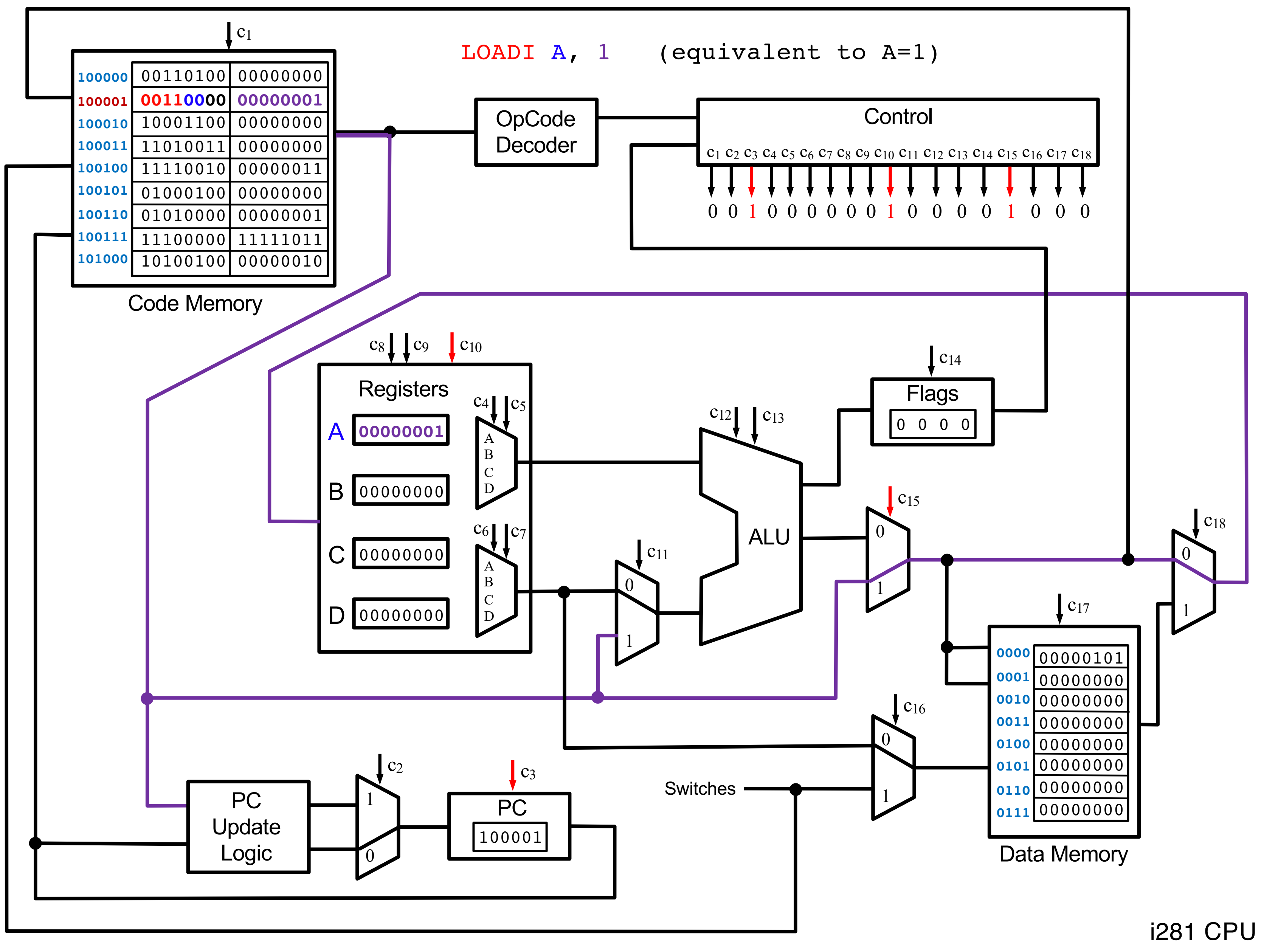
i281 CPU

LOADI B, 0 (equivalent to B=0)

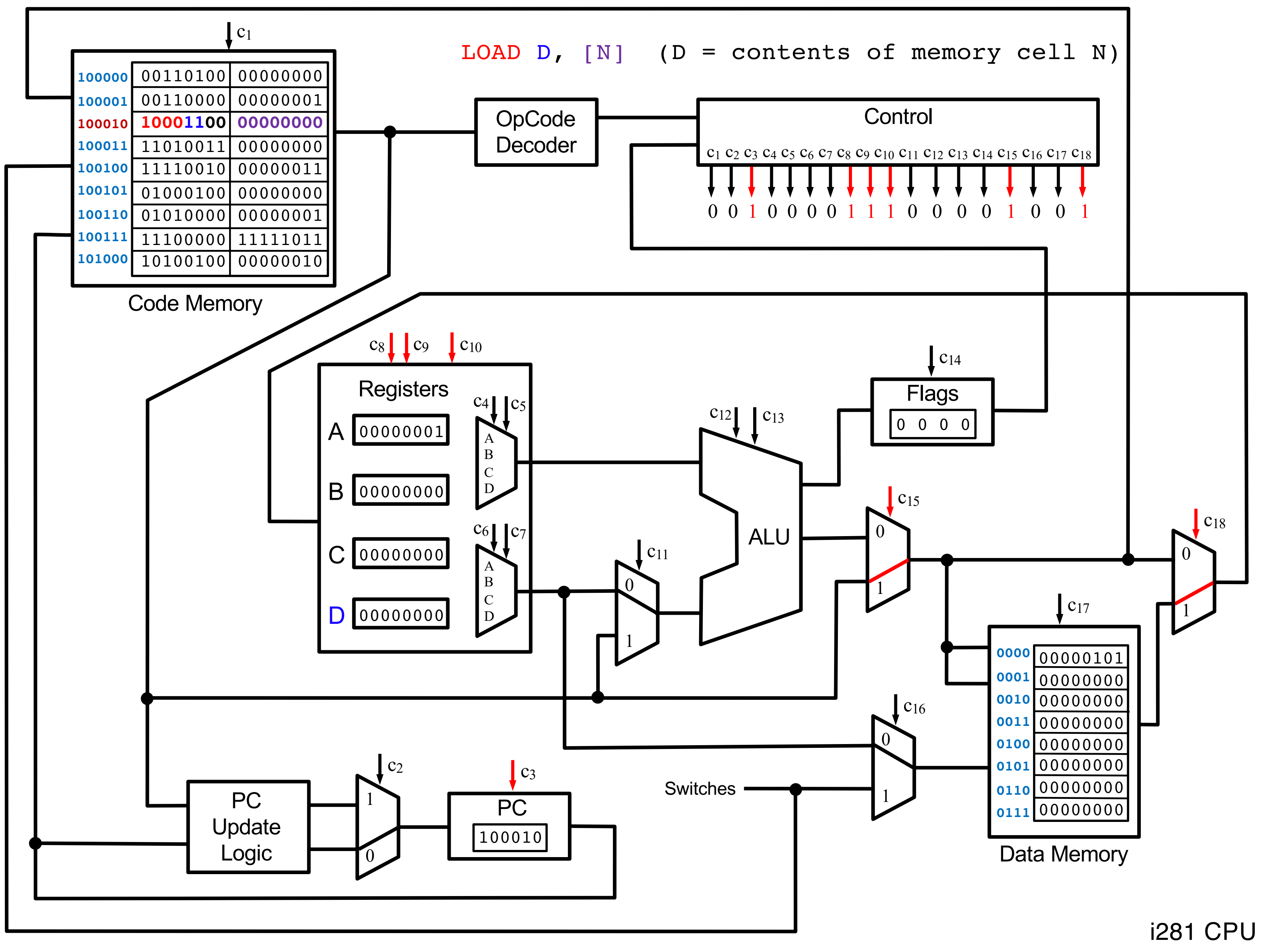


i281 CPU

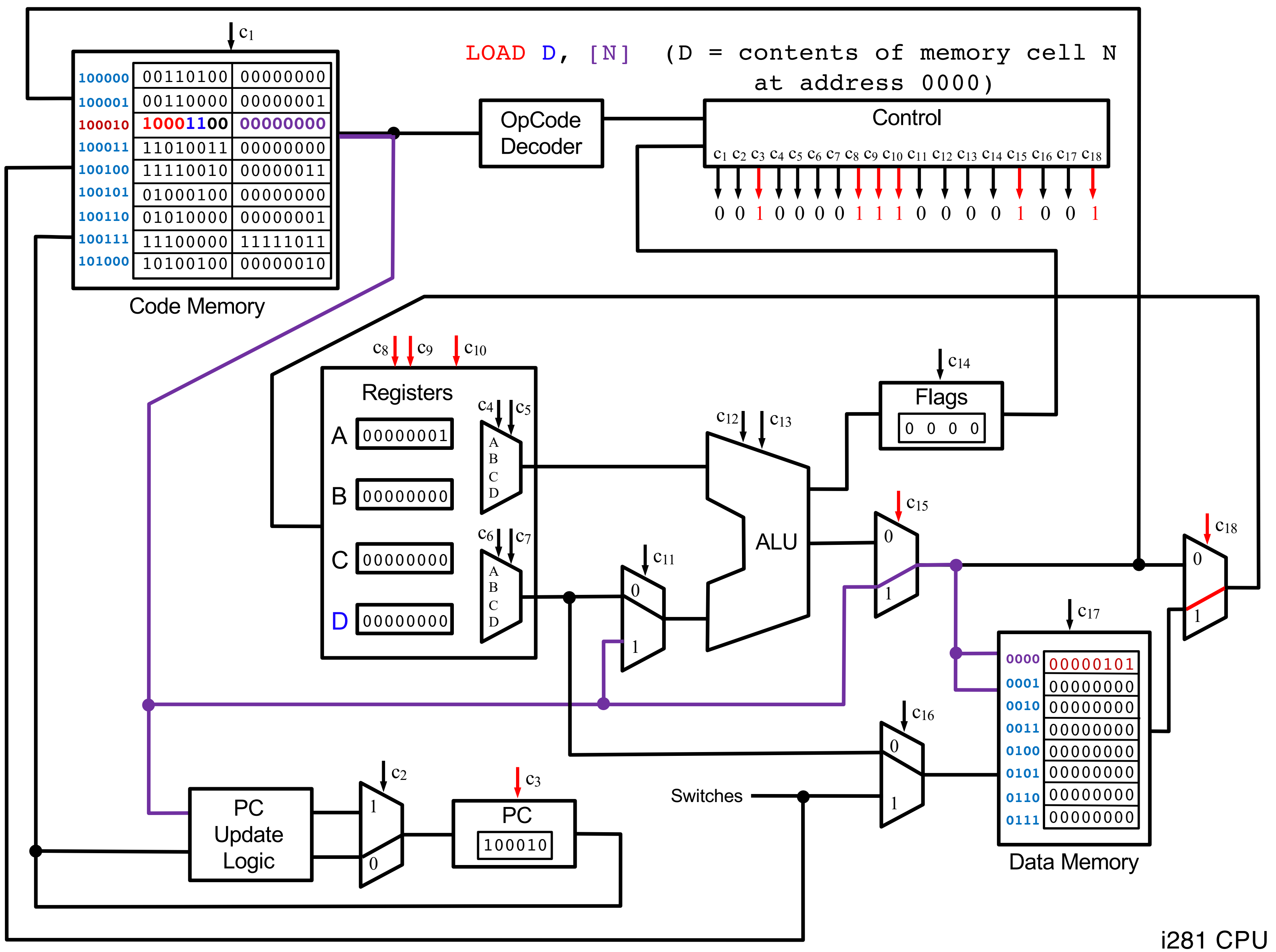
```
LOADI A, 1    (equivalent to A=1)
```



LOAD D, [N] (D = contents of memory cell N)

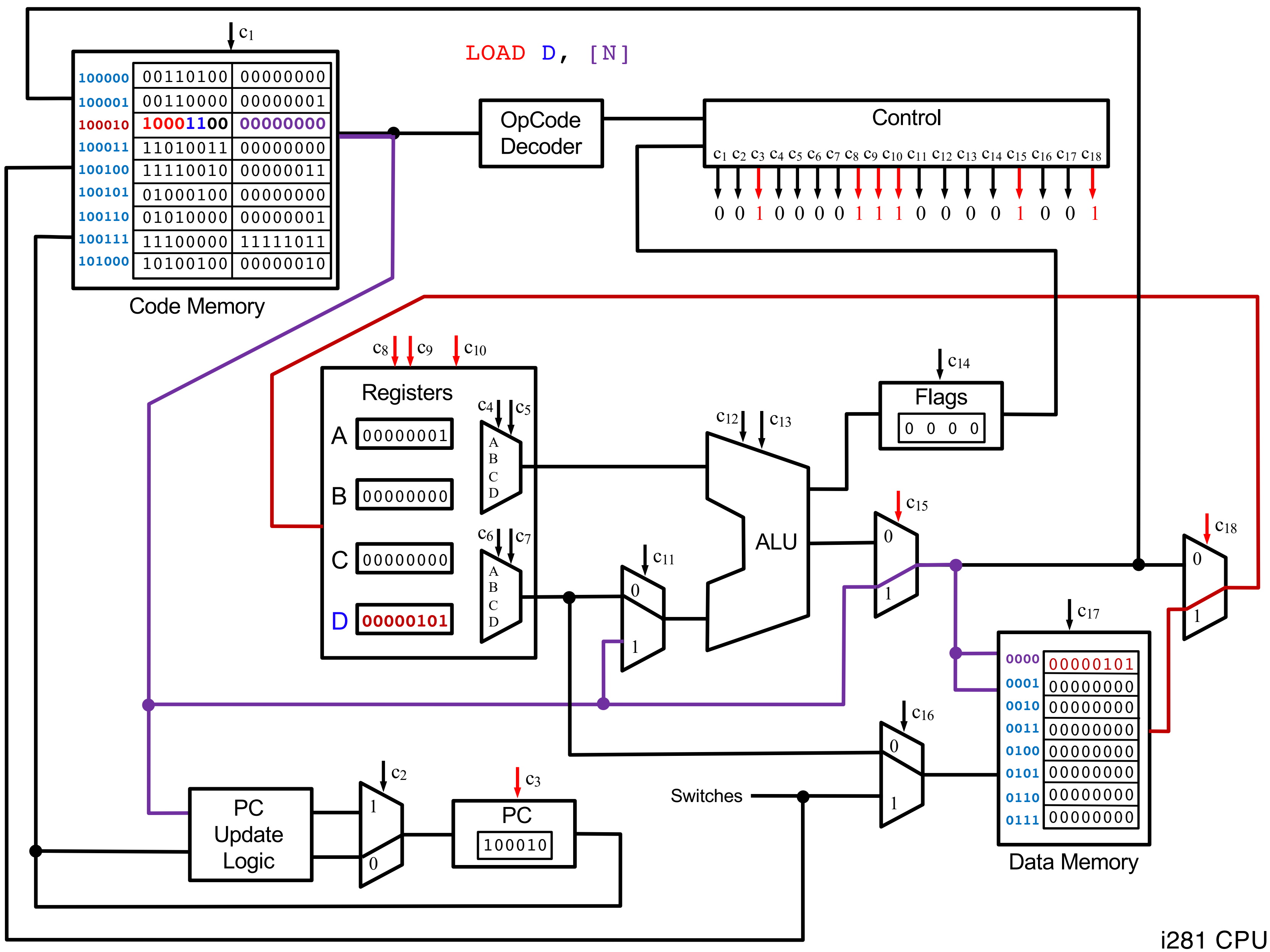


LOAD D, [N] (D = contents of memory cell N at address 0000)



i281 CPU

LOAD D, [N]

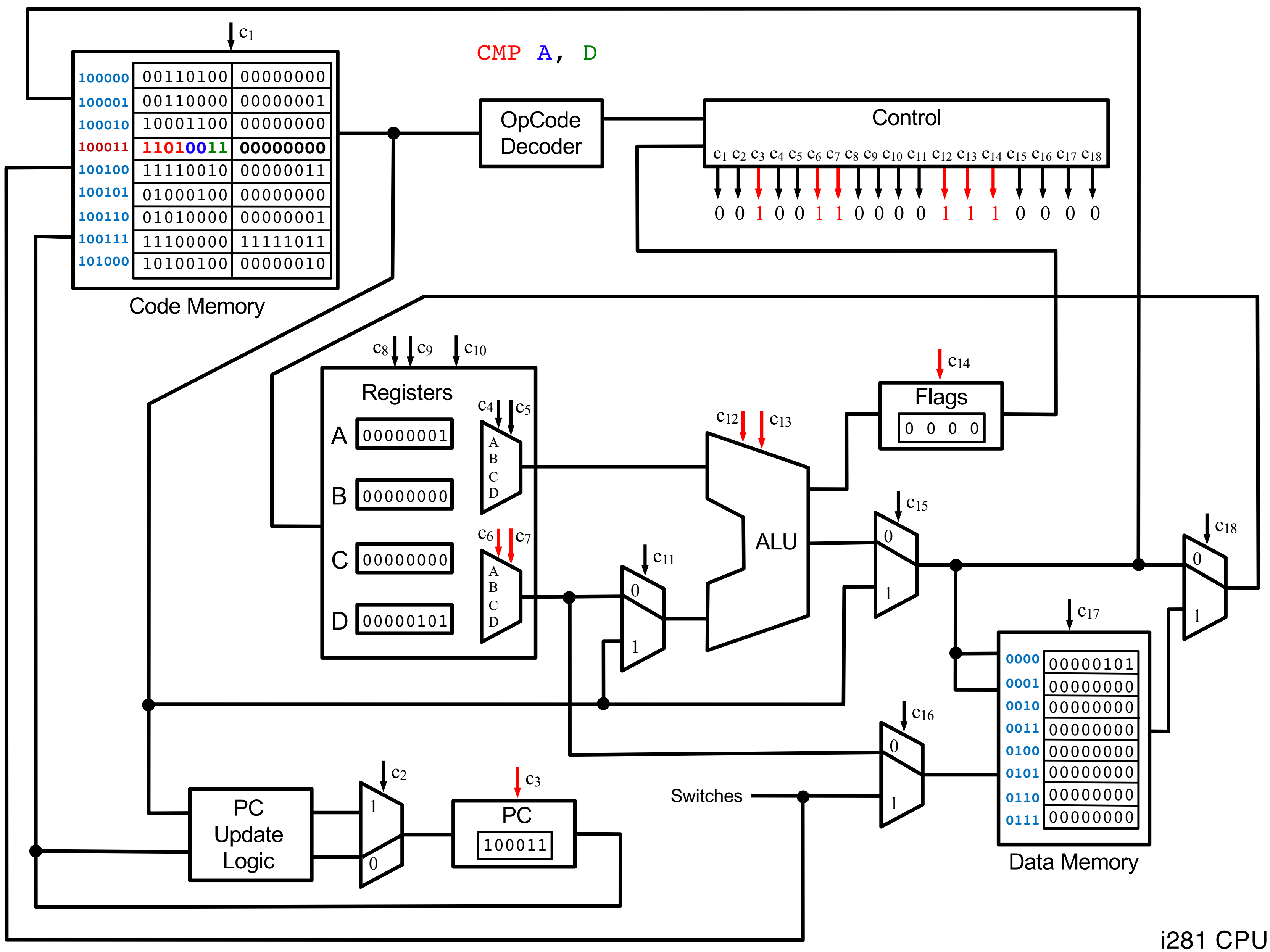


100000	00110100	00000000
100001	00110000	00000001
100010	10001100	00000000
100011	11010011	00000000
100100	11110010	00000011
100101	01000100	00000000
100110	01010000	00000001
100111	11100000	11111011
101000	10100100	00000010

0000	00000101
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000

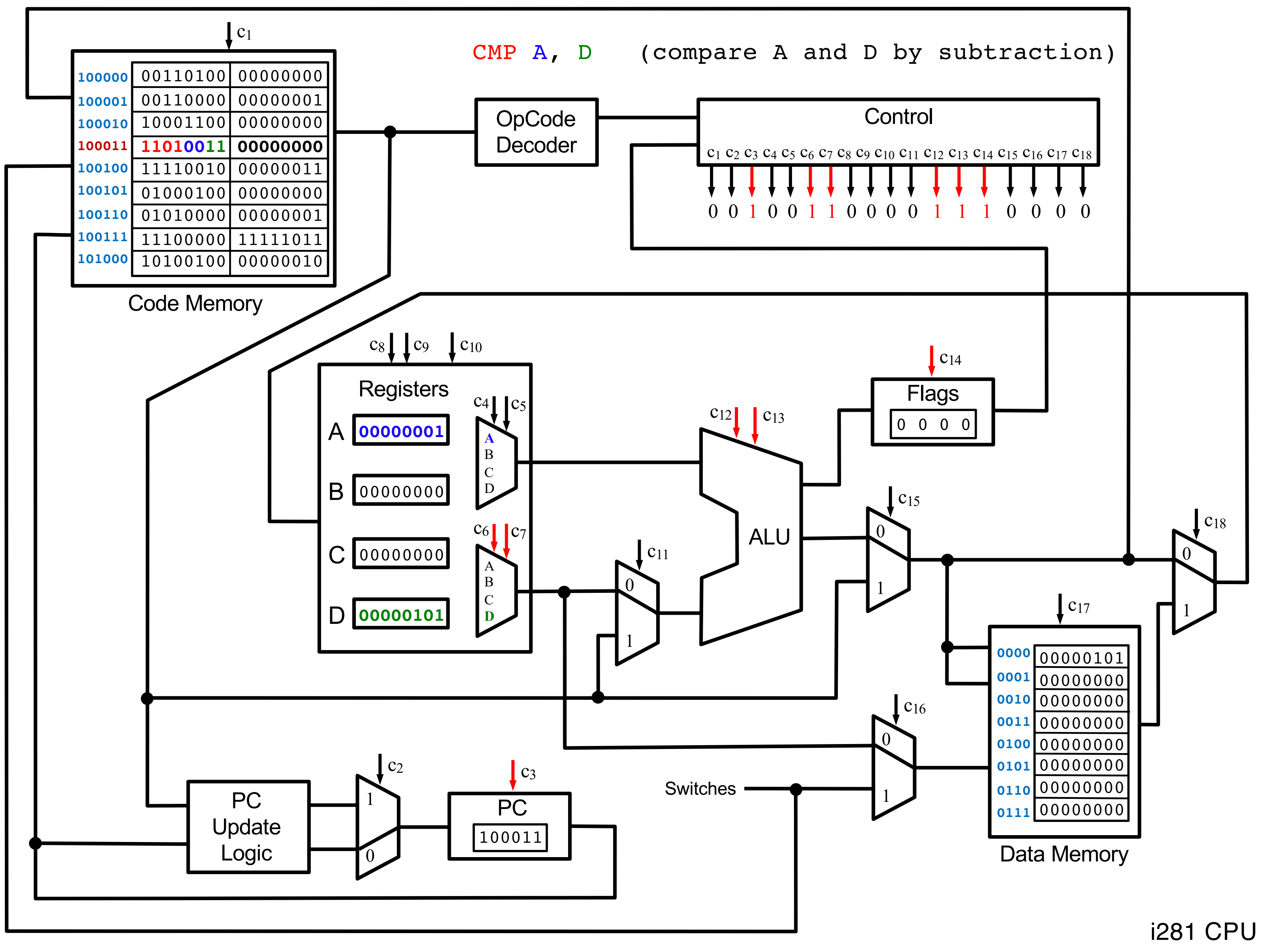
i281 CPU

CMP A, D

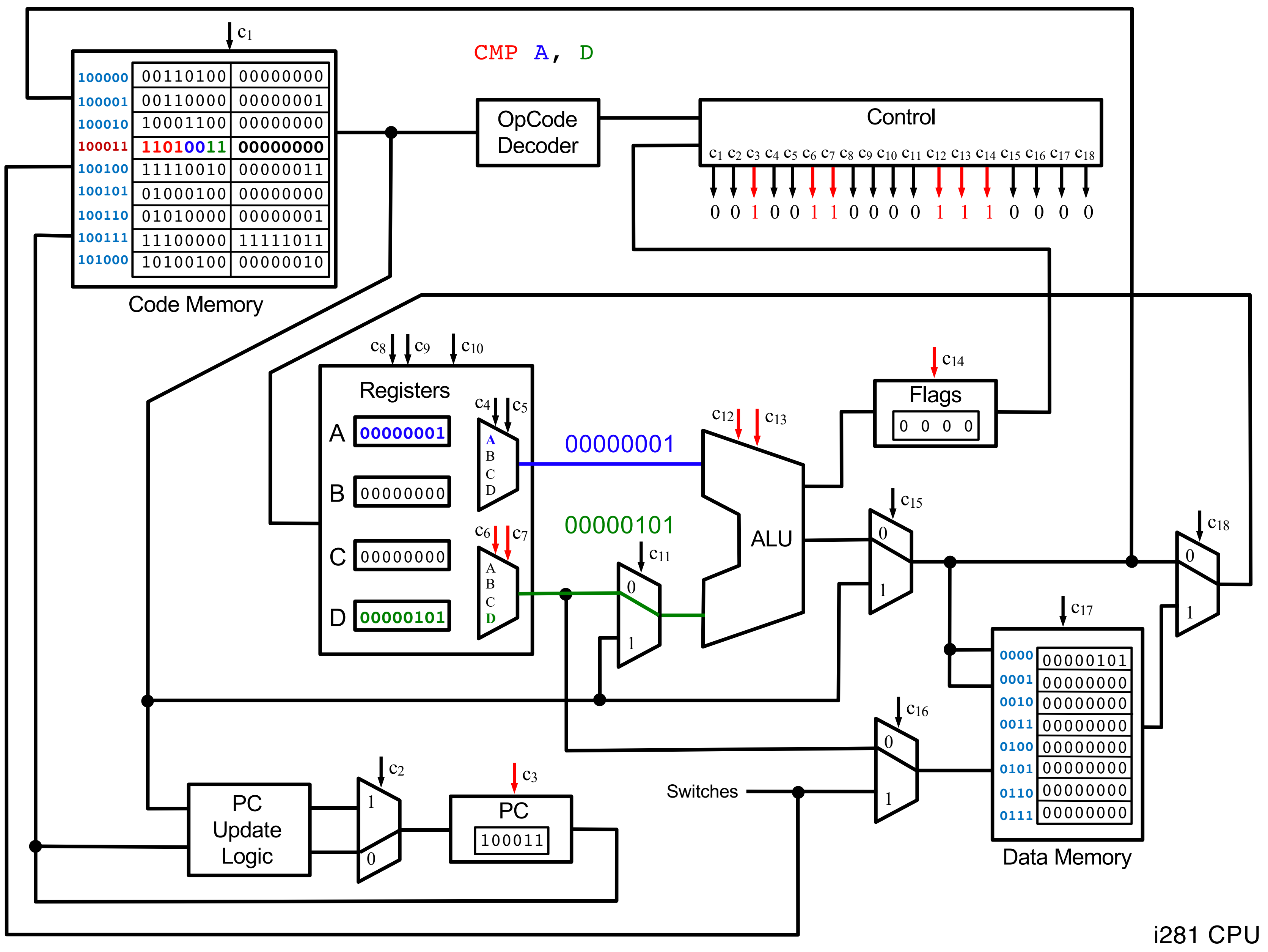


i281 CPU

CMP A, D   (compare A and D by subtraction)

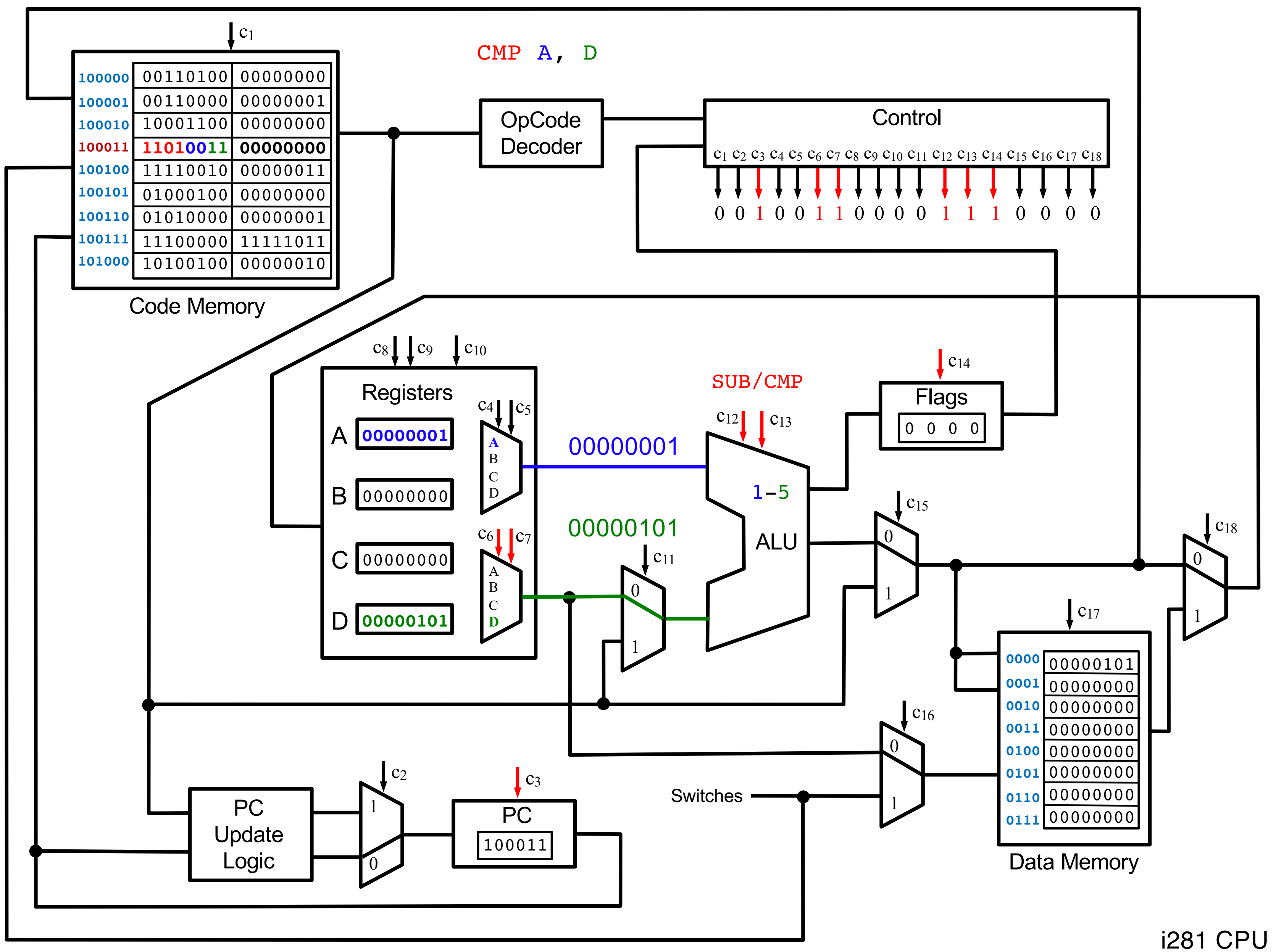


i281 CPU



100000	00110100	00000000
100001	00110000	00000001
100010	10001100	00000000
100011	11010011	00000000
100100	11110010	00000011
100101	01000100	00000000
100110	01010000	00000001
100111	11100000	11111011
101000	10100100	00000010

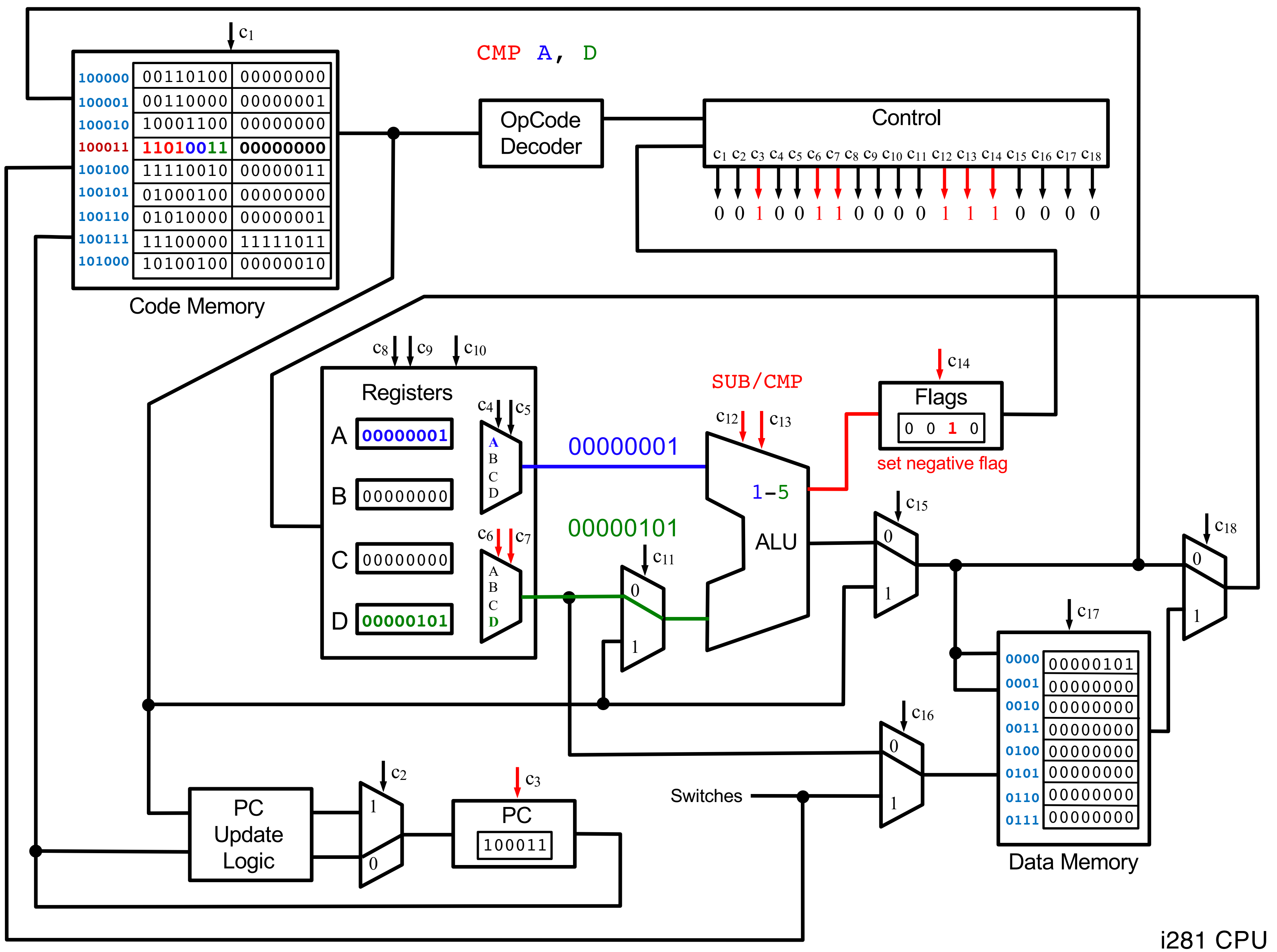
0000	00000101
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000



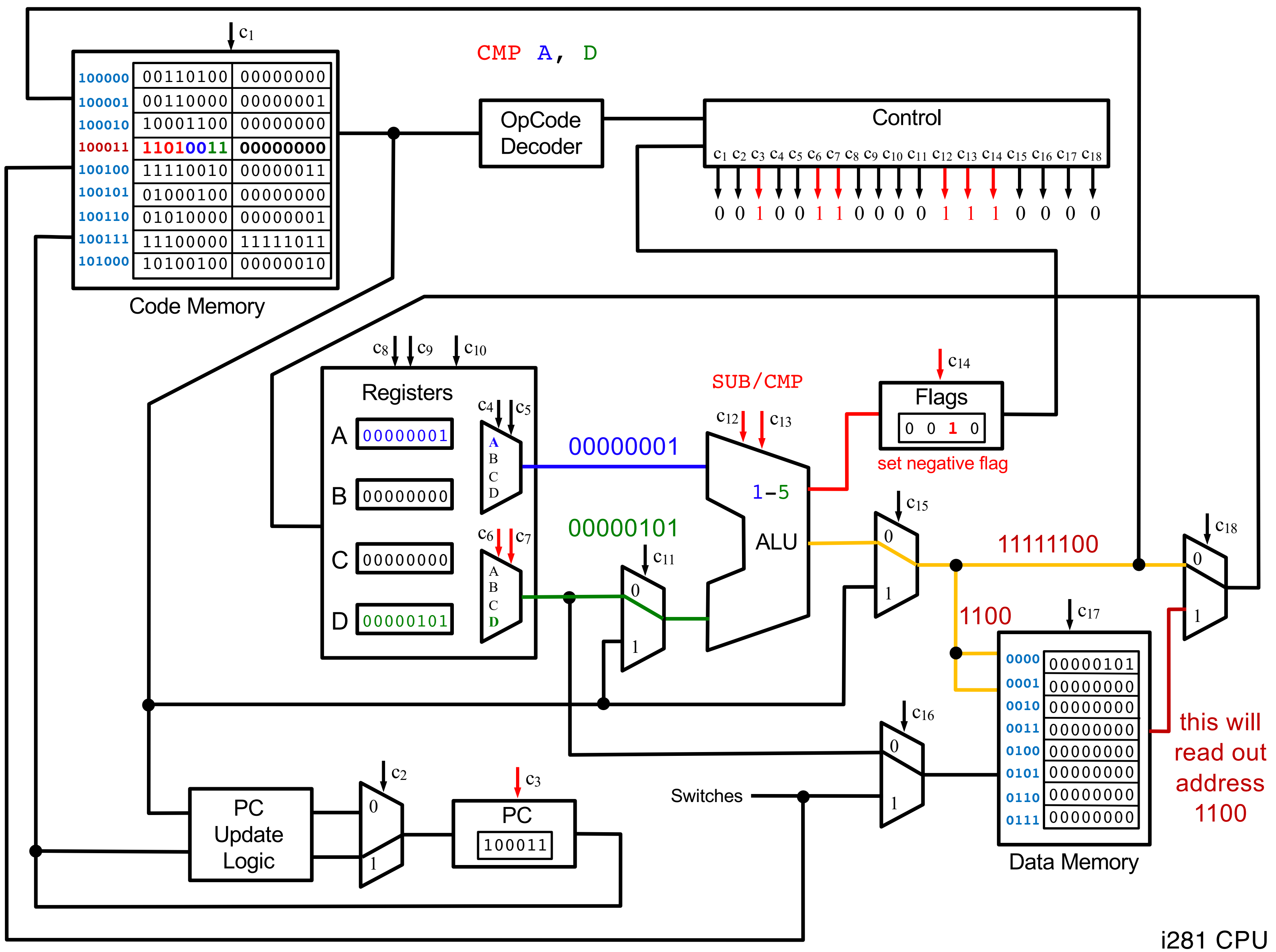
100000	00110100	00000000
100001	00110000	00000001
100010	10001100	00000000
100011	11010011	00000000
100100	11110010	00000011
100101	01000100	00000000
100110	01010000	00000001
100111	11100000	11111011
101000	10100100	00000010

0000	00000101
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000

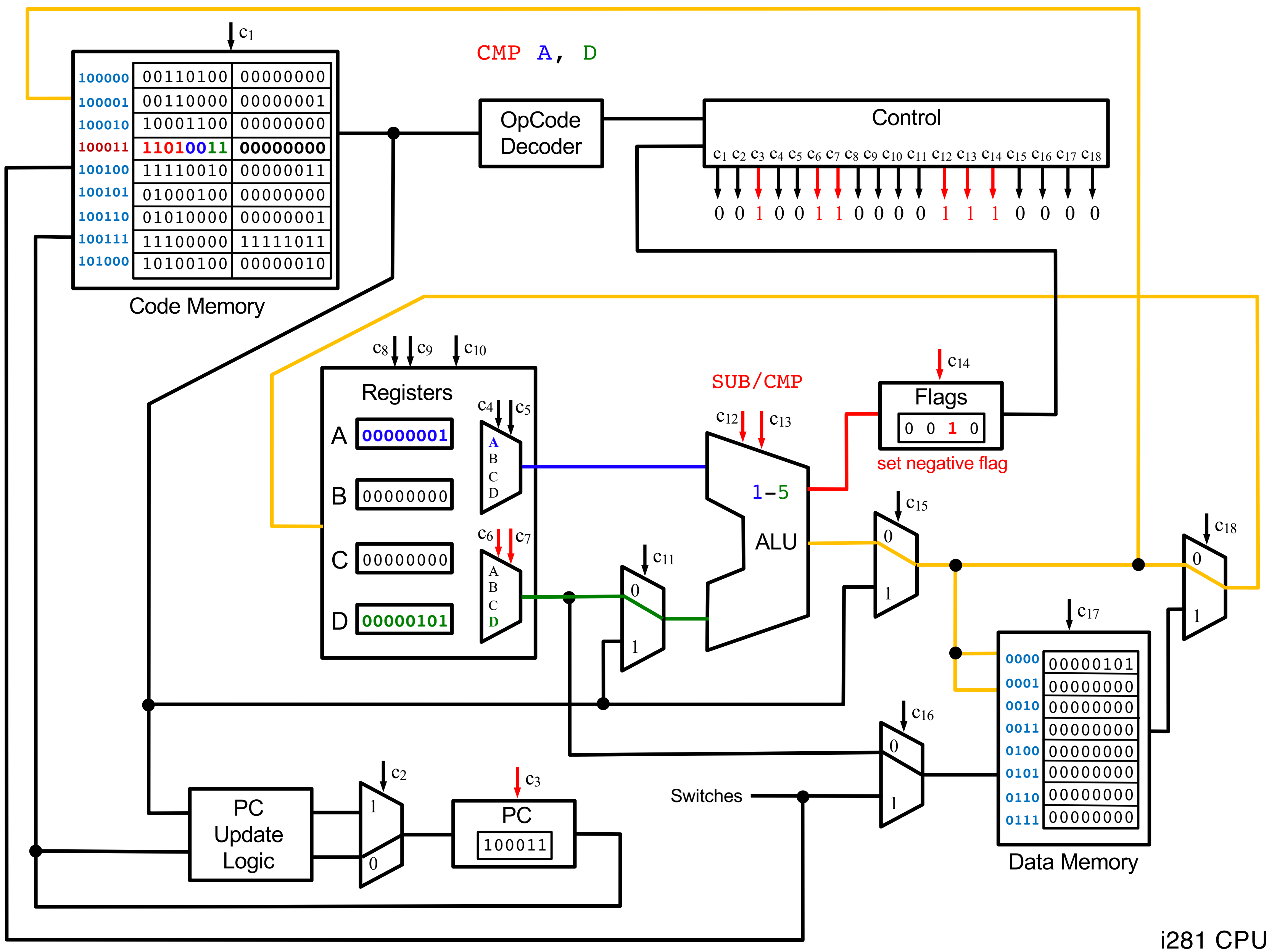
i281 CPU


CMP A, D
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000101
c4 c5
A B C D
c6 c7
A B C D
00000001
00000101
c11
0 1
SUB/CMP
c12 c13
1−5
ALU
c14
Flags
0 0 1 0
set negative flag
c15
0 1
c18
0 1
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c16
0 1
Switches
c2
1 0
c3
PC
100011
PC Update Logic

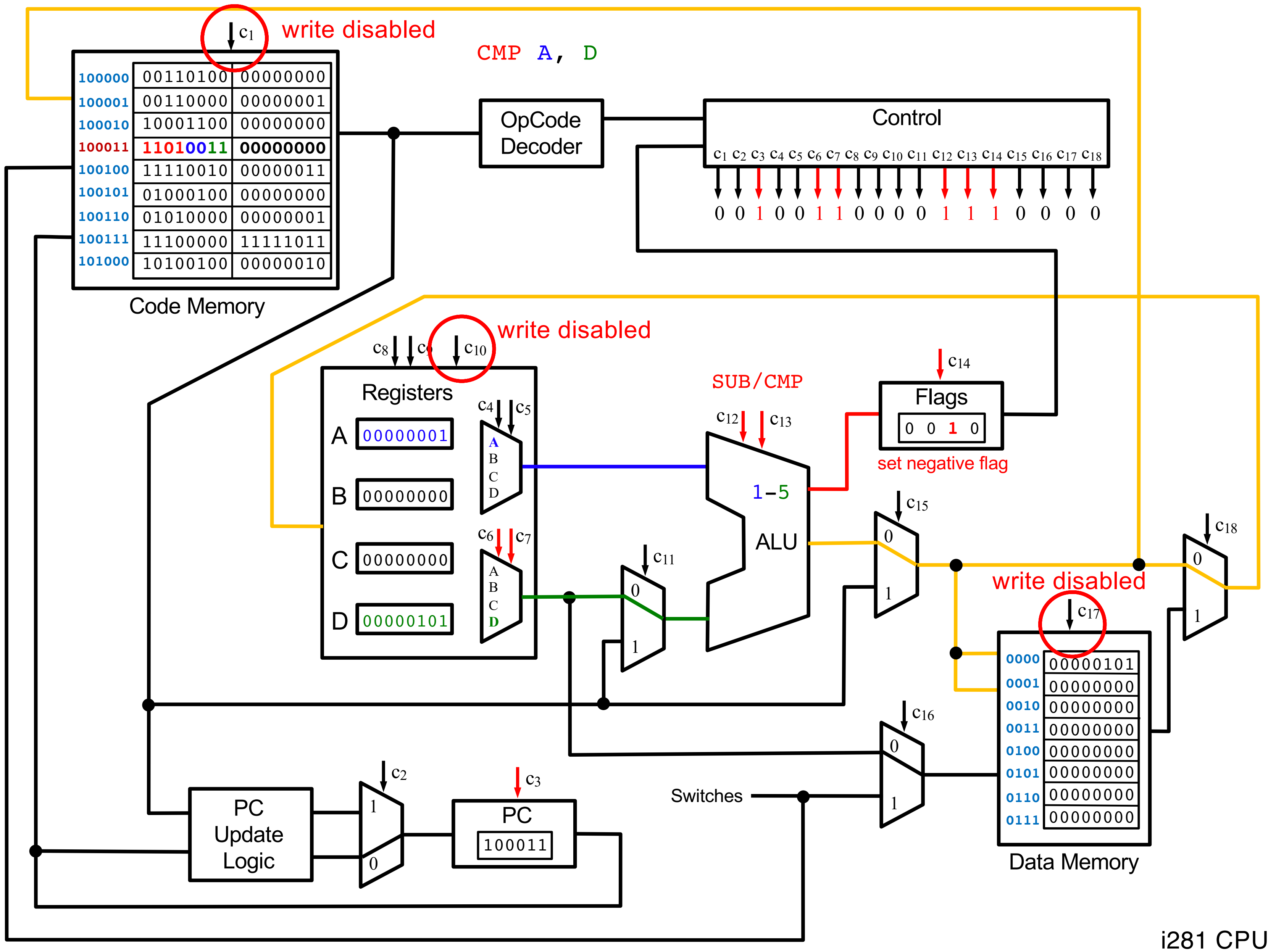
i281 CPU

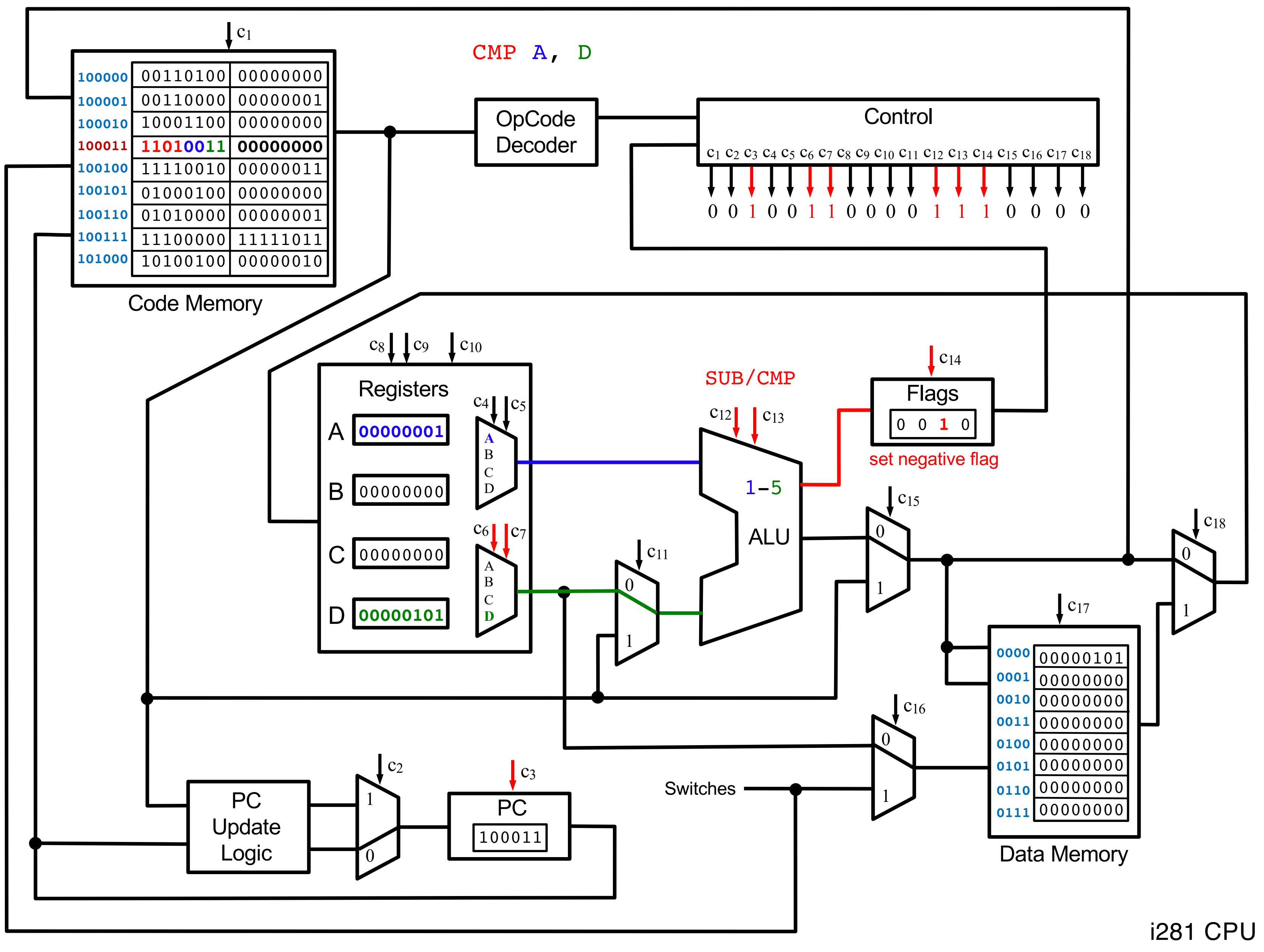

CMP A, D
c1
Code Memory
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000101
c4 c5
A B C D
c6 c7
A B C D
00000001
00000101
c11
0
1
SUB/CMP
c12 c13
1−5
ALU
c14
Flags
0 0 1 0
set negative flag
c15
0
1
11111100
1100
c18
0
1
c17
Data Memory
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
this will read out address 1100
c16
0
1
Switches
PC Update Logic
c2
0
1
c3
PC
100011

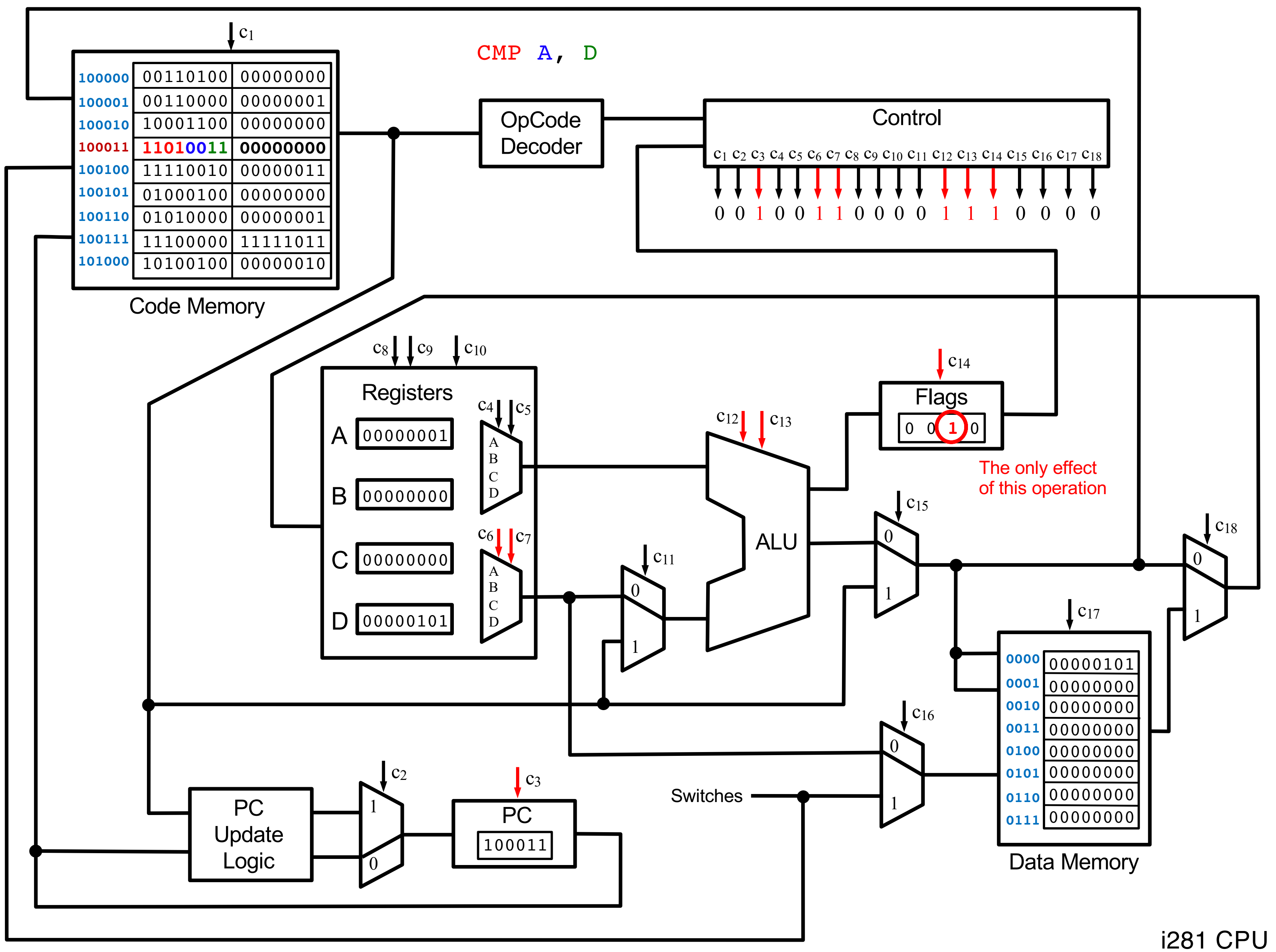
i281 CPU


CMP A, D
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000101
c4 c5
A B C D
c6 c7
A B C D
SUB/CMP
c12 c13
1-5
ALU
c14
Flags
0 0 1 0
set negative flag
c11
0 1
c15
0 1
c18
0 1
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c16
0 1
Switches
PC Update Logic
c2
1 0
c3
PC
100011

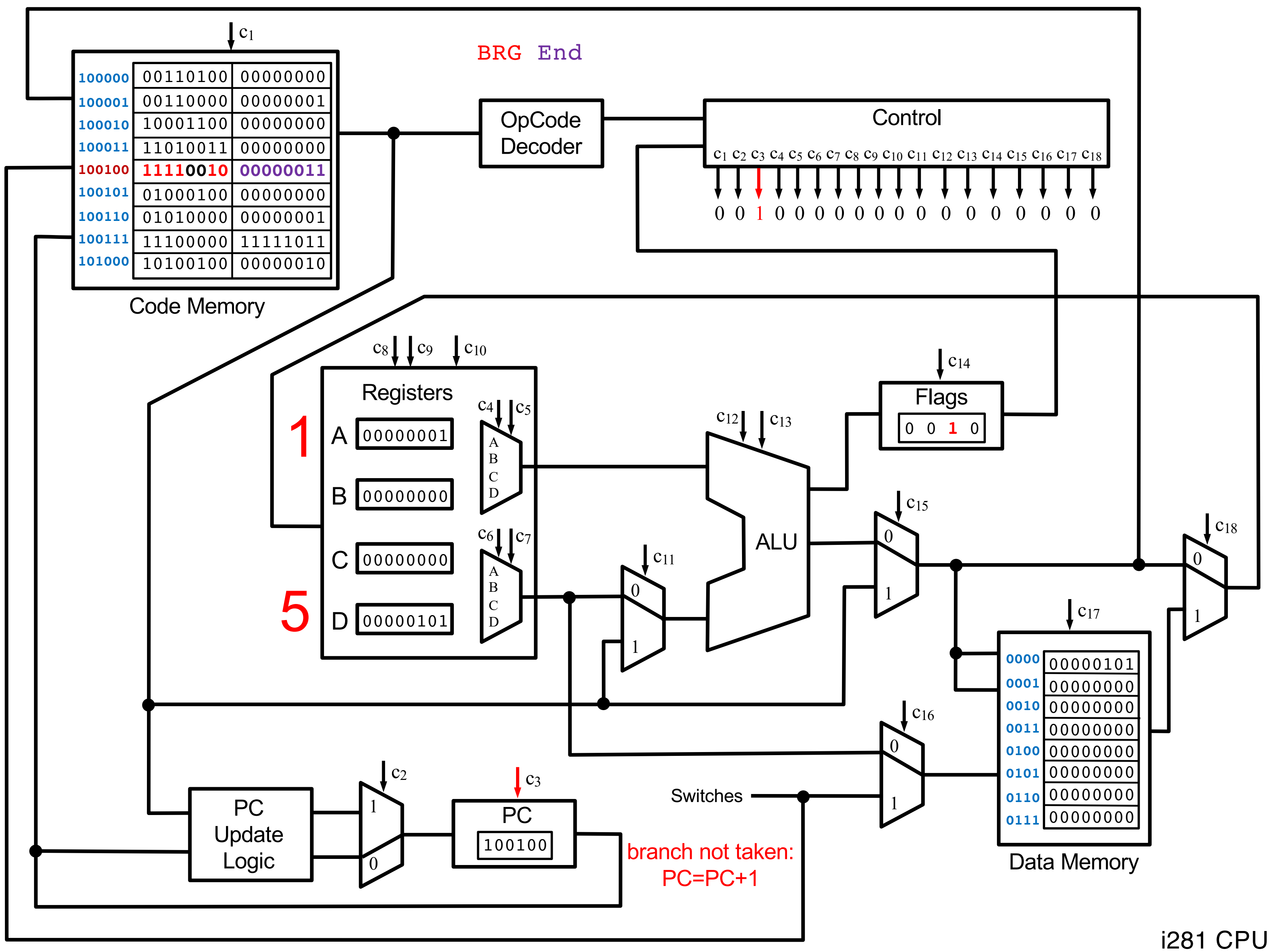
i281 CPU


write disabled
CMP A, D
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0
write disabled
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000101
c4 c5
c6 c7
SUB/CMP
c12 c13
1–5
ALU
c14
Flags
0 0 1 0
set negative flag
c11
c15
c18
write disabled
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c16
Switches
PC Update Logic
c2
c3
PC
100011
i281 CPU

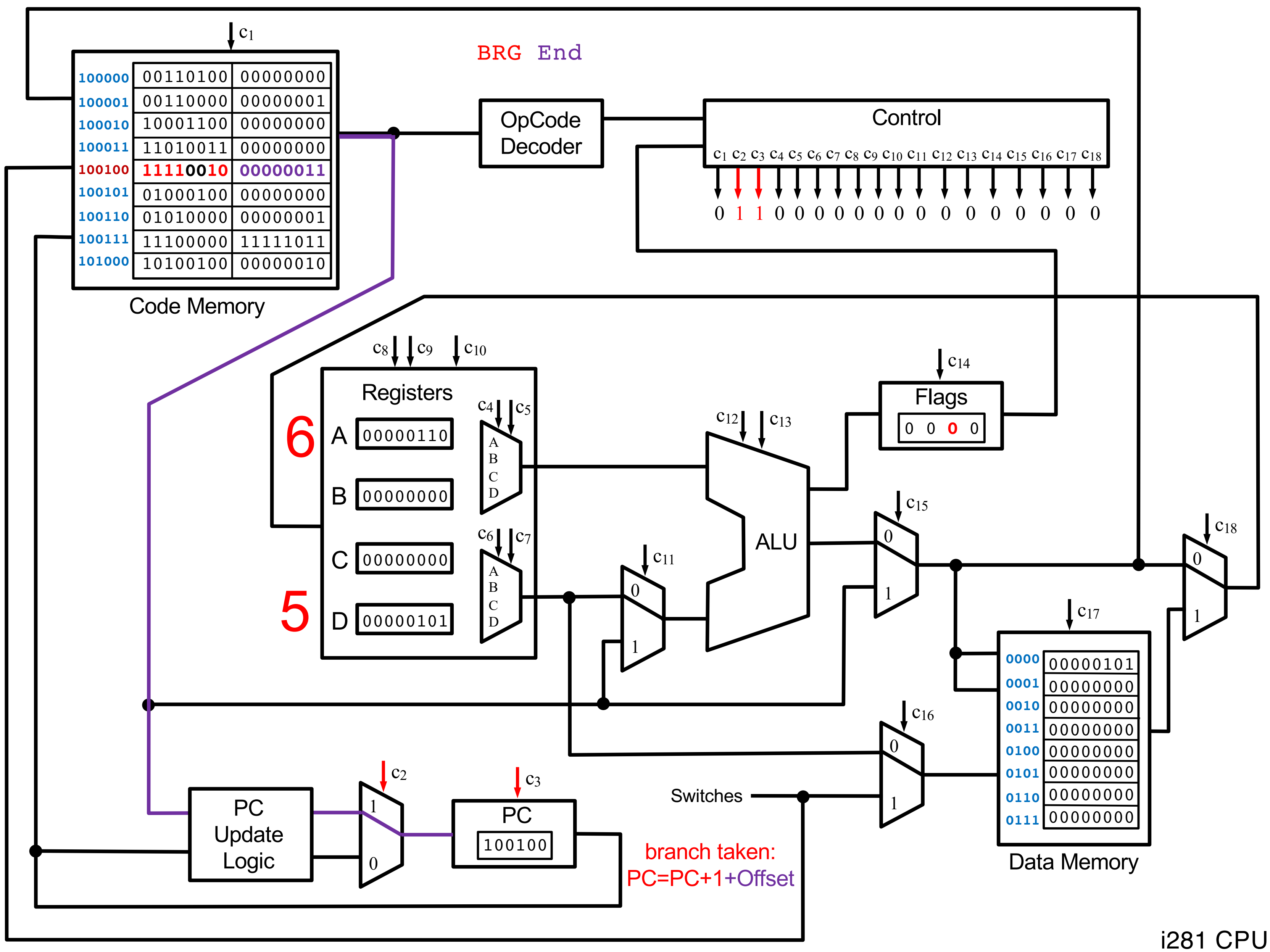

CMP A, D
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000101
c4 c5
A B C D
c6 c7
A B C D
SUB/CMP
c12 c13
1-5
ALU
c11
0
1
c14
Flags
0 0 1 0
set negative flag
c15
0
1
c18
0
1
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c16
0
1
Switches
PC Update Logic
c2
1
0
c3
PC
100011
i281 CPU


CMP A, D
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000101
c4 c5
c6 c7
c12 c13
ALU
c14
Flags
0 0 1 0
The only effect of this operation
c11
c15
c18
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c16
Switches
c2
c3
PC
100011
PC Update Logic

i281 CPU


BRG End
Code Memory
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Registers
1
A 00000001
B 00000000
C 00000000
5
D 00000101
ALU
Flags
0 0 1 0
Data Memory
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Switches
PC Update Logic
PC
100100
branch not taken: PC=PC+1

i281 CPU



Code Memory

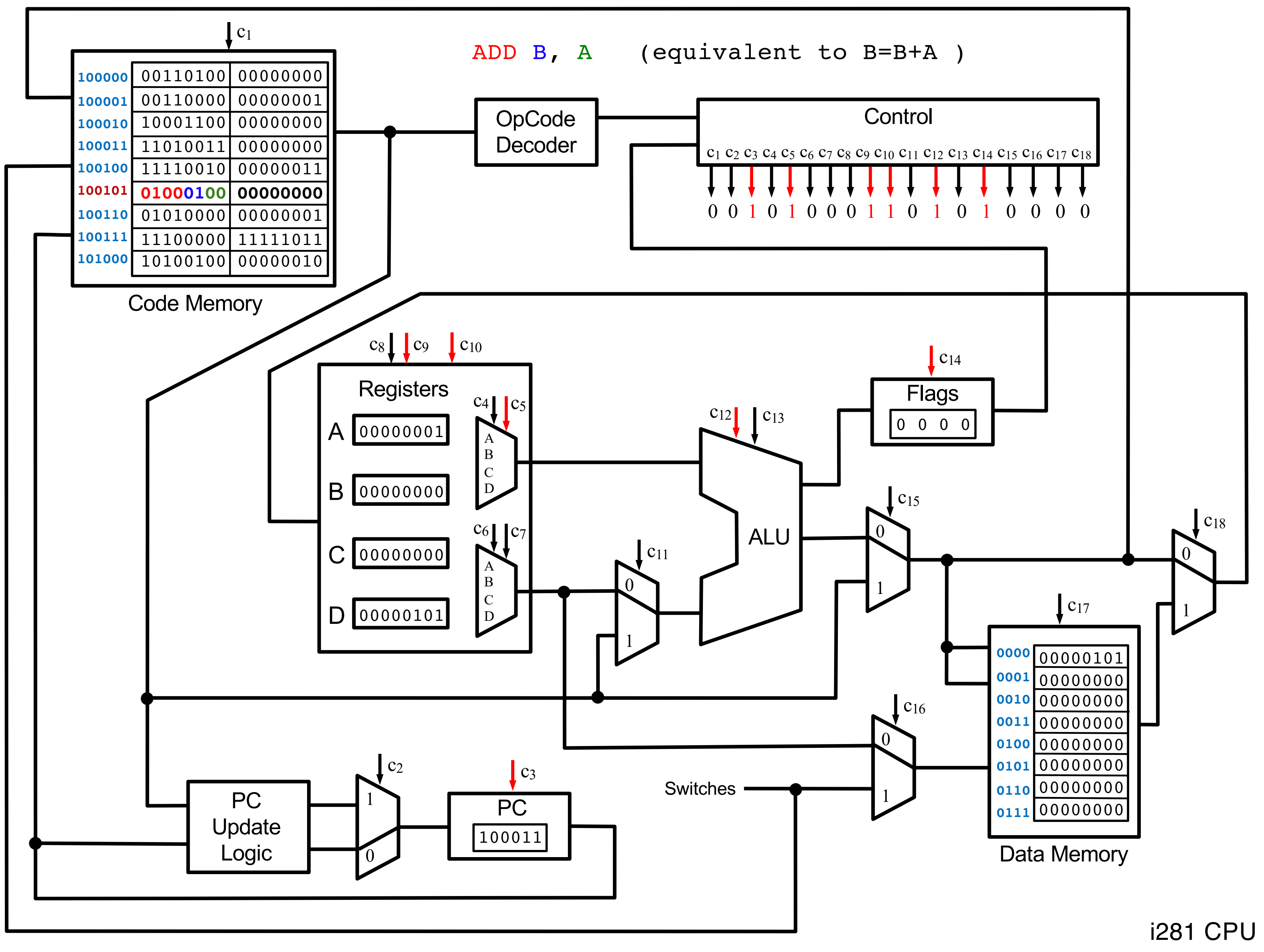
100000	00110100	00000000
100001	00110000	00000001
100010	10001100	00000000
100011	11010011	00000000
100100	11110010	00000011
100101	01000100	00000000
100110	01010000	00000001
100111	11100000	11111011
101000	10100100	00000010

Data Memory

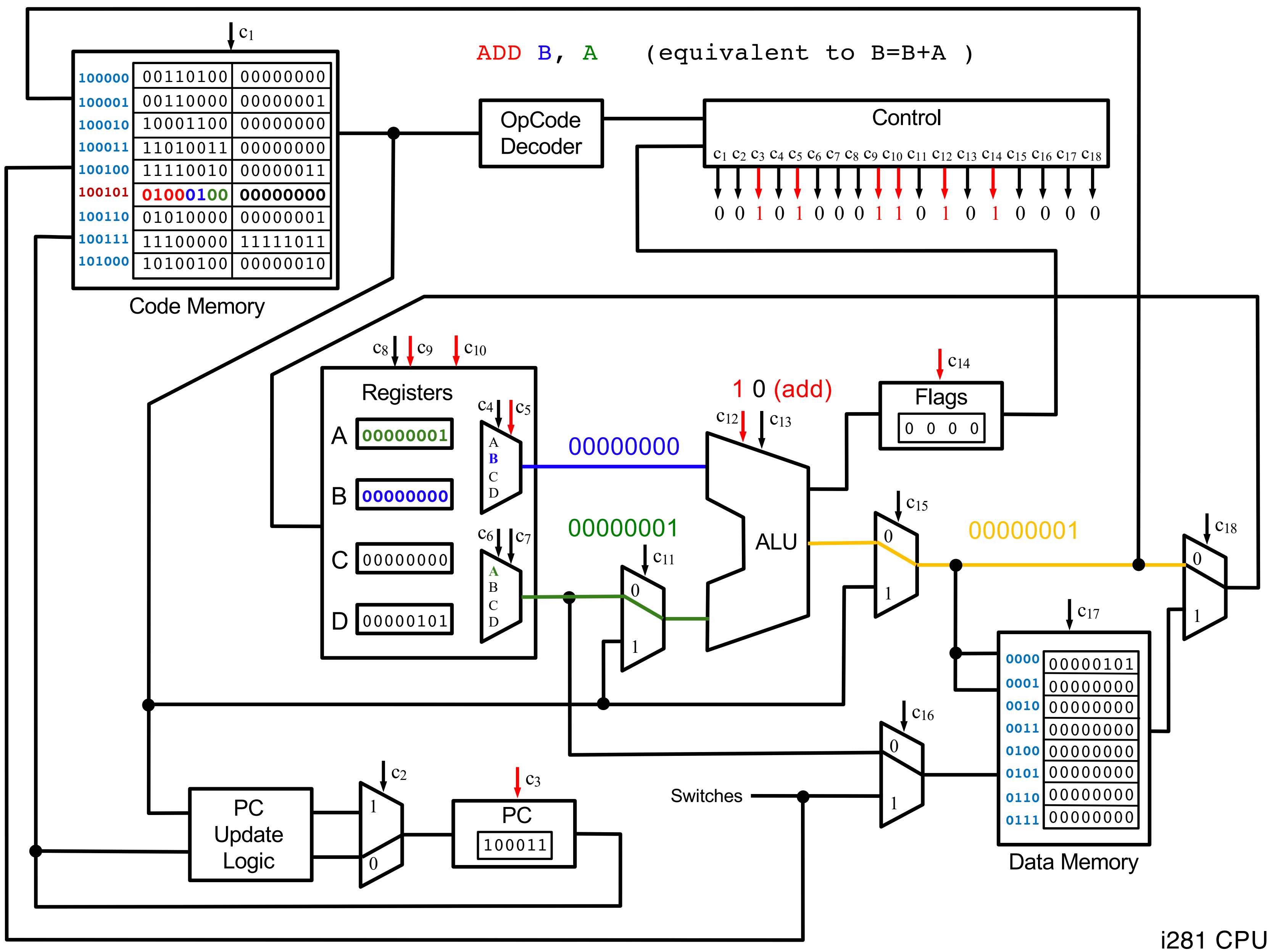
0000	00000101
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000

i281 CPU

ADD B, A   (equivalent to B=B+A)

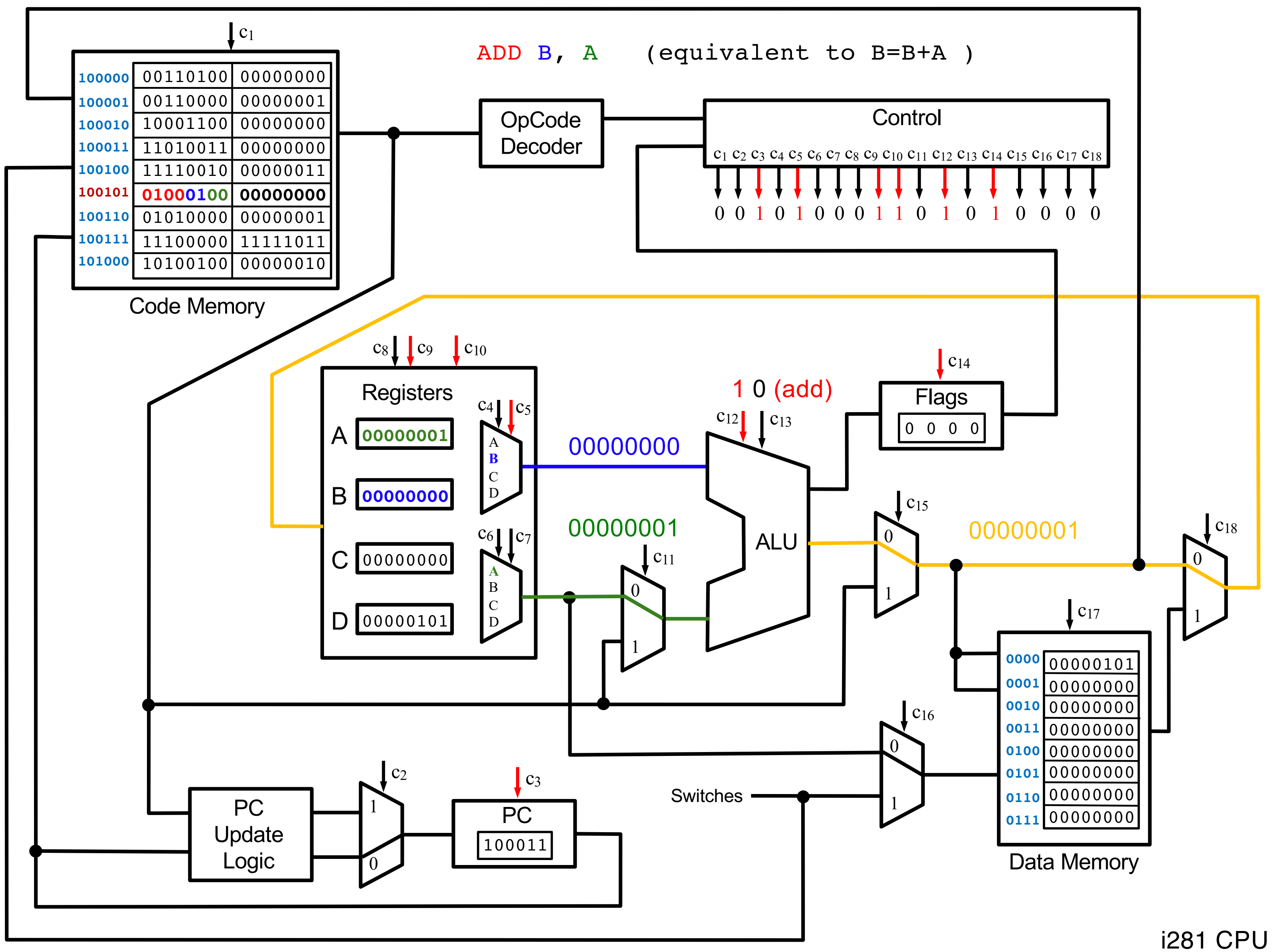


ADD B, A (equivalent to B=B+A)

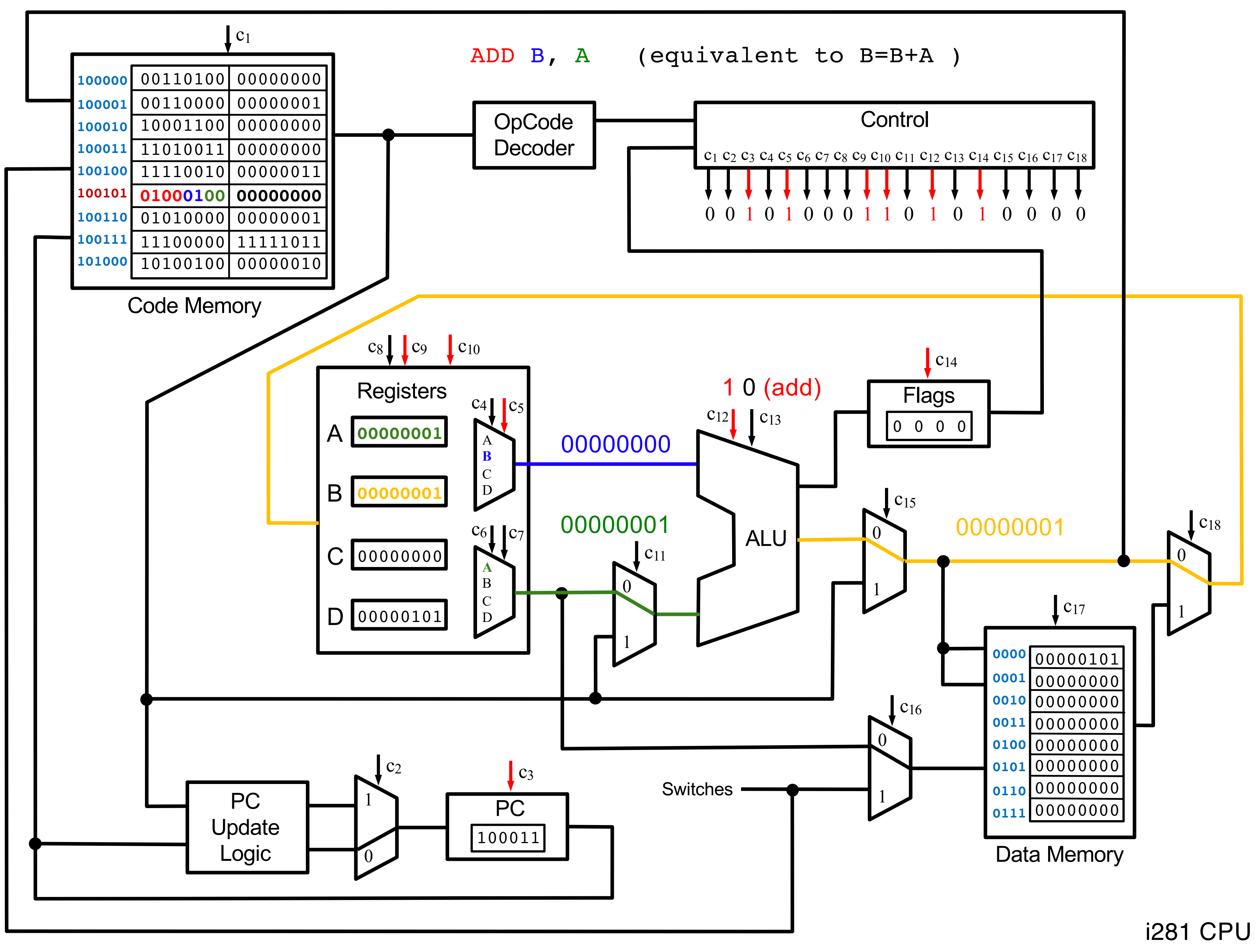


i281 CPU

ADD B, A (equivalent to B=B+A)



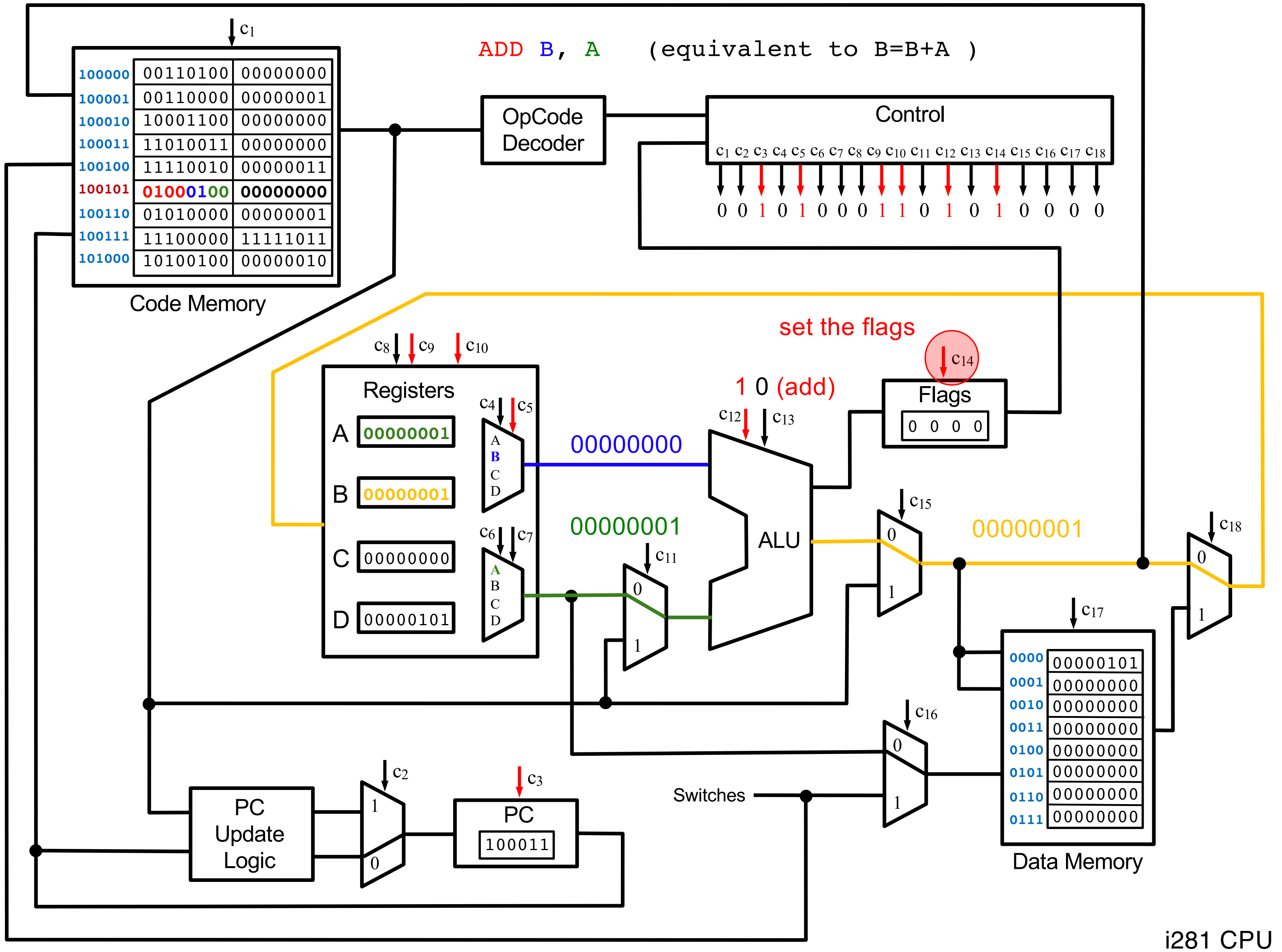
i281 CPU



ADD B, A (equivalent to B=B+A)
Code Memory
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 1 0 0 0 1 1 0 1 0 1 0 0 0 0
Registers
A 00000001
B 00000001
C 00000000
D 00000101
00000000
00000001
1 0 (add)
ALU
Flags
0 0 0 0
00000001
Data Memory
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Switches
PC Update Logic
PC
100011

i281 CPU

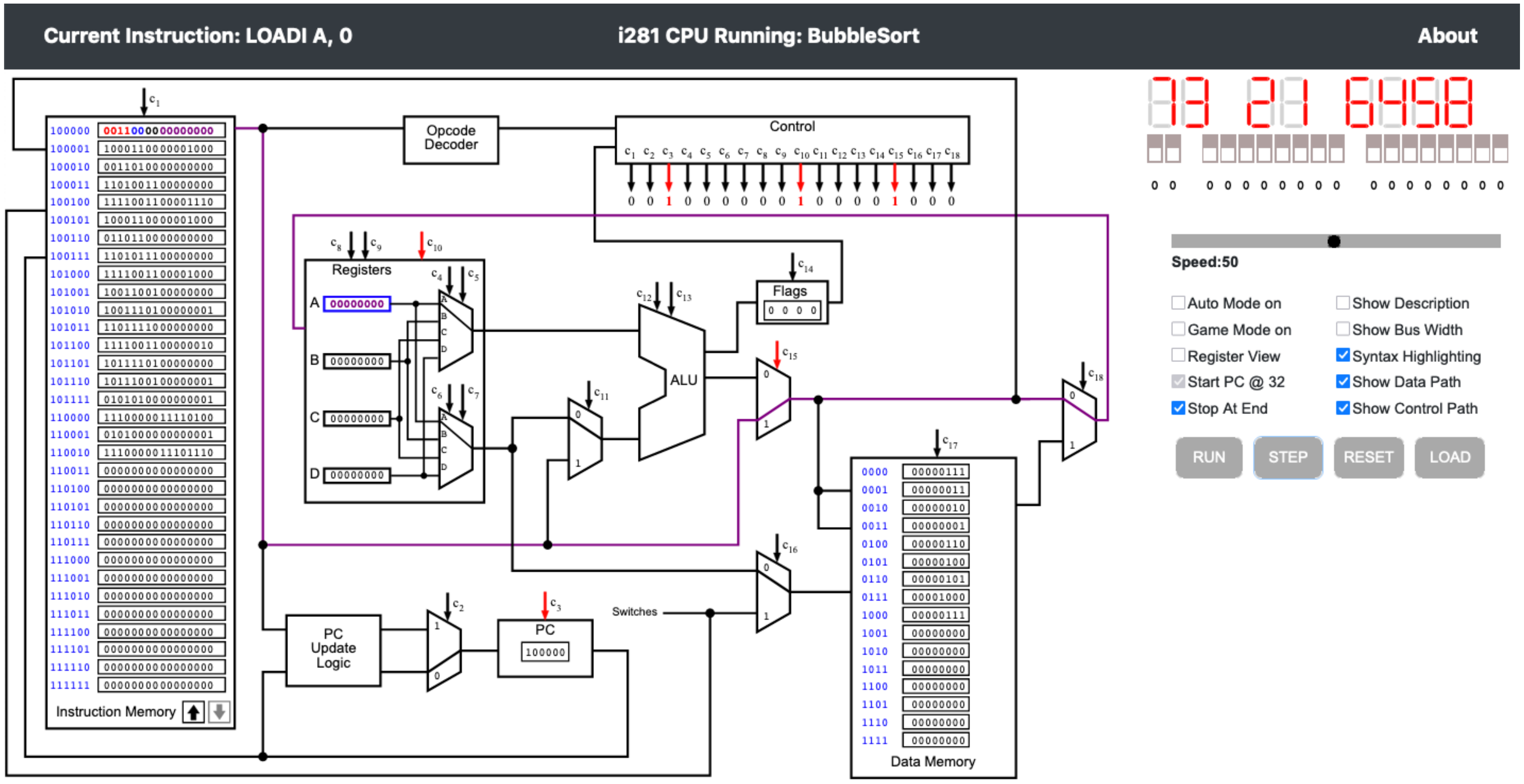
ADD B, A (equivalent to B=B+A)



i281 CPU

For more examples try the i281 simulator

i281 Simulator



Current Instruction: LOADI A, 0

i281 CPU Running: BubbleSort

About

```
100000  0011000000000000
100001  1000110000001000
100010  0011010000000000
100011  1101001100000000
100100  1111001100001110
100101  1000110000001000
100110  0110110000000000
100111  1101011100000000
101000  1111001100001000
101001  1001100100000000
101010  1001110100000001
101011  1101111100000000
101100  1111001100000010
101101  1011110100000000
101110  1011100100000001
101111  0101010000000001
110000  1110000011110100
110001  0101000000000001
110010  1110000011101110
110011  0000000000000000
110100  0000000000000000
110101  0000000000000000
110110  0000000000000000
110111  0000000000000000
111000  0000000000000000
111001  0000000000000000
111010  0000000000000000
111011  0000000000000000
111100  0000000000000000
111101  0000000000000000
111110  0000000000000000
111111  0000000000000000
Instruction Memory
```

```
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0  0  1  0  0  0  0  0  0  1   0   0   0   0   1   0   0   0
```

```
Registers
A  00000000
B  00000000
C  00000000
D  00000000

Flags  0 0 0 0

PC  100000
```

```
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
```

Speed:50

- [ ] Auto Mode on
- [ ] Game Mode on
- [ ] Register View
- [x] Start PC @ 32
- [x] Stop At End
- [ ] Show Description
- [ ] Show Bus Width
- [x] Syntax Highlighting
- [x] Show Data Path
- [x] Show Control Path

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

Questions?

THE END