Основы проектирования аппаратных ускорителей систем искусственного интеллекта

Лекция 5 – конечные автоматы

План лекции

- Конечный автомат.
- Проектирование конечного автомата.
- Понятие об операционном и управляющем автоматах.
- Построение указанных автоматов по описанию алгоритма.
- Аппаратная реализация простых структур данных.

Конечный автомат

- Пусть A, B, Q – конечные алфавиты
- Функция состояний:

$$\varphi: A \times Q \to Q$$

- Функция переходов:

$$\psi: A \times Q \to B$$

- Начальное состояние: $q_0 \in Q$
- Конечный автомат:

$$Aut = \{A, B, Q, \varphi, \psi, q_0\}$$

Потоковый двоичный сумматор

- $A = \{00,01,10,11\}$
- $B = Q = \{0,1\}$

Q	A	φ	ψ
0	00	0	0
0	01	0	1
0	10	0	1
0	11	1	0
1	00	0	1
1	01	1	0
1	10	1	0
1	11	1	1

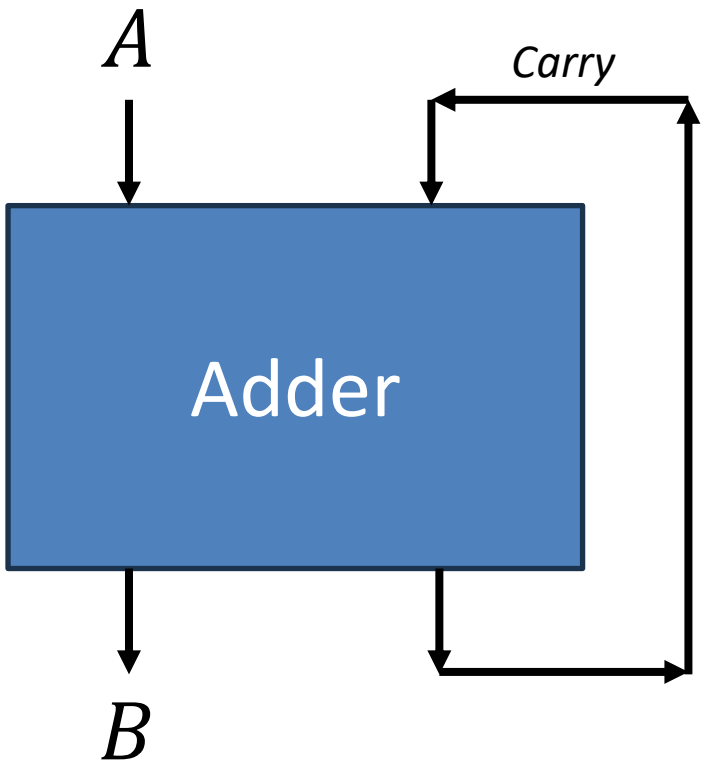
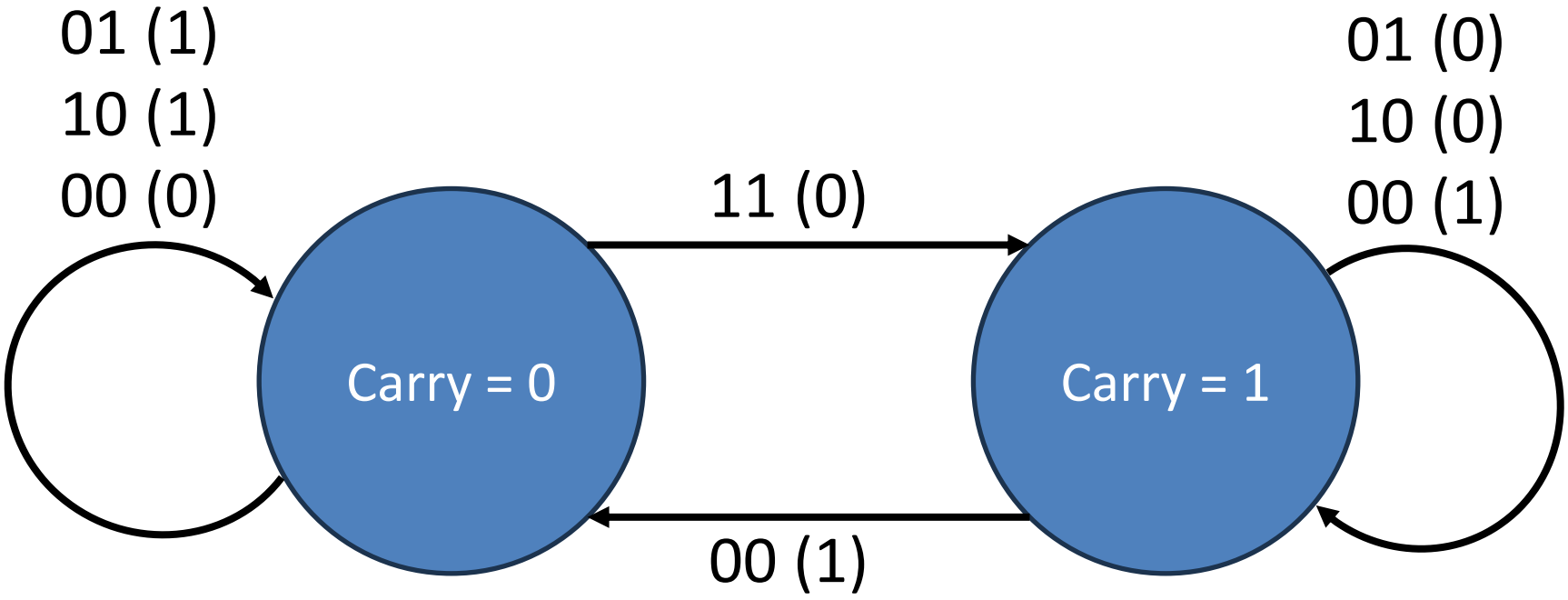


Диаграмма Мура

- Конечный автомат представим в виде графа специального вида:

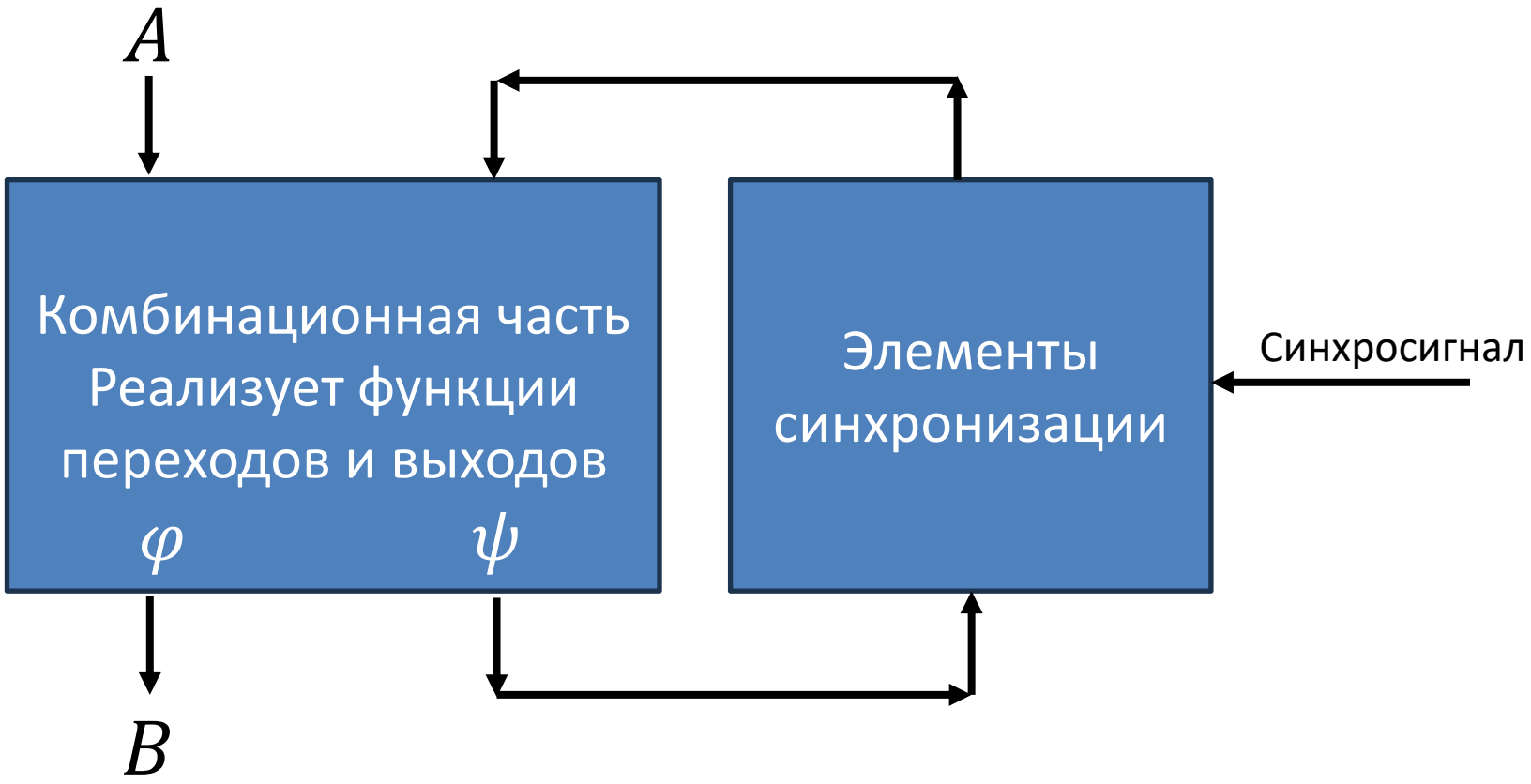


Кодирование состояний

- Каждое состояние кодируется при помощи двоичной последовательности.
- Можно использовать различные кодировки.
- Для $|Q| = k$ при равномерном кодировании потребуется не менее $\lceil \log_2 k \rceil$ кодов

$$q_1 = 00, q_2 = 01, q_3 = 10$$

Реализация конечного автомата



Реализация конечного автомата

- Реализация комбинационной части:

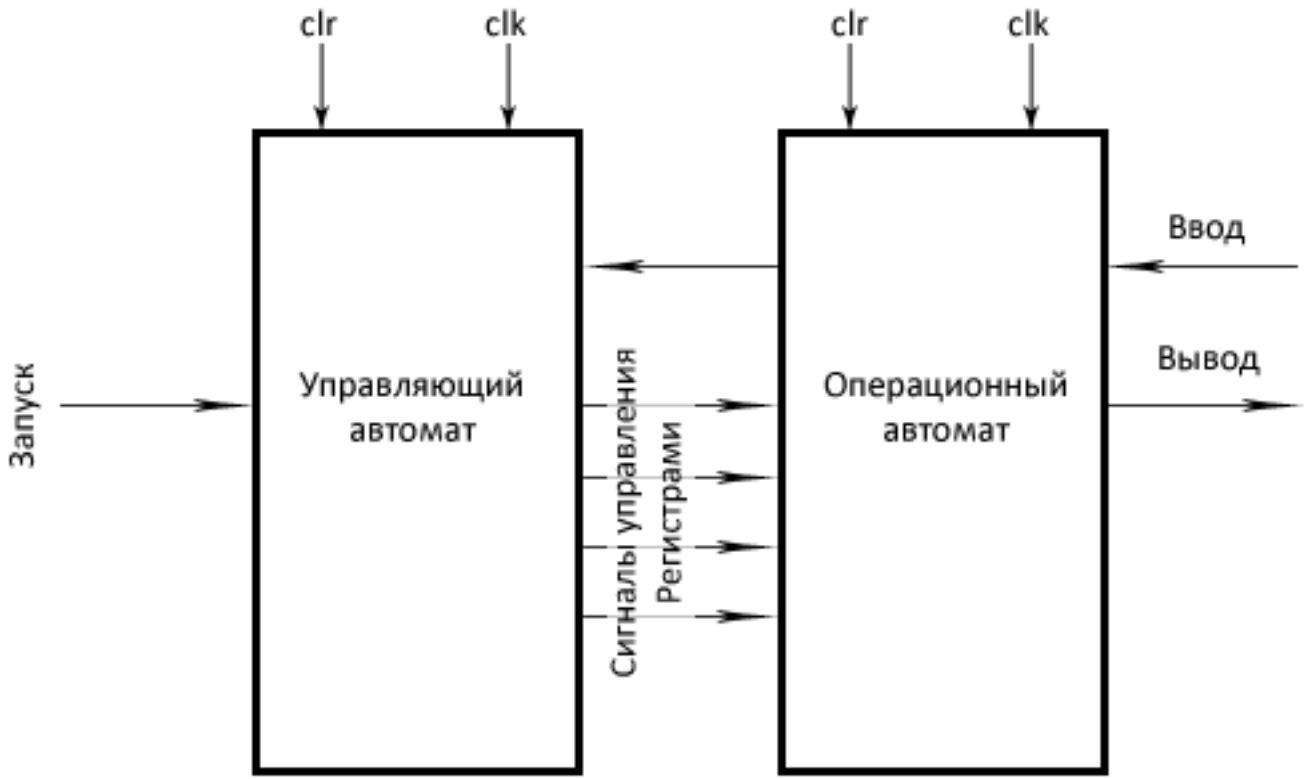
```
always @ (c_state, in)
        out = c_state ^ in[0] ^ in[1];
        case (c_state)
                1'b0: if(in == 2'b11)
                        n_state = 1'b1;
                else
                        n_state = 1'b0;
                `1'b1: if(in == 2'b00)
                        n_state = 1'b0;
                else
                        n_state = 1'b1;
        endcase
```

Реализация конечного автомата

- Реализация элементов синхронизации

```
always @ (posedge clock, negedge reset)
        if (~reset)
                c_state <= 1'b0;
        else
                c_state <= n_state;
```

Операционный и управляющий автоматы



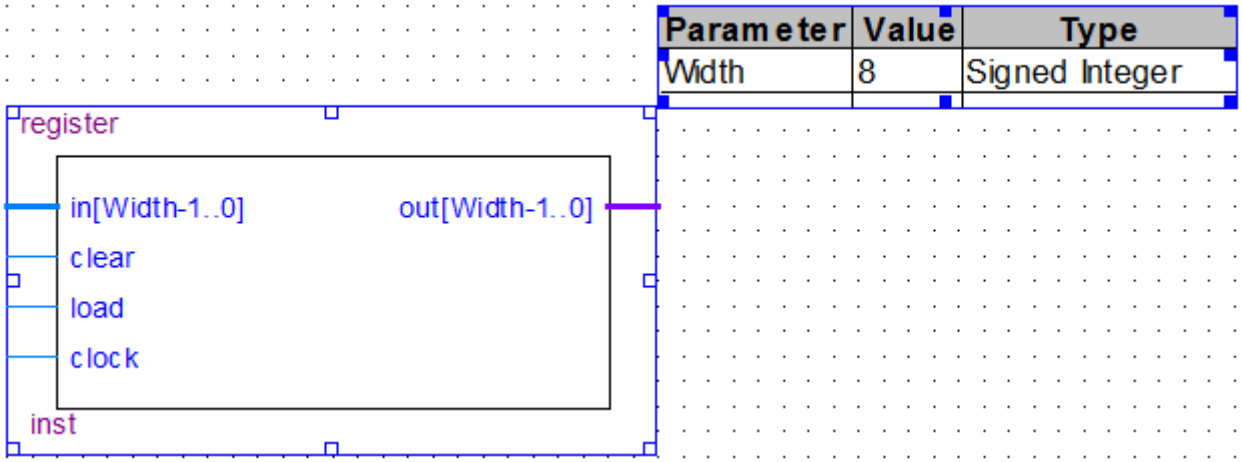
Пример алгоритма

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```

Регистры

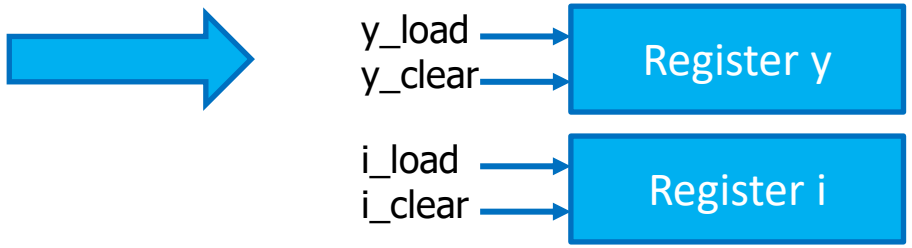
```
module register
        #(parameter              Width = 8)
        (output reg      [Width-1:0] out,
        input            [Width-1:0] in,
        input                        clear, load, clock);

        always @(posedge clock)
                if (~clear)
                        out <= 0;
                else if (~load)
                        out <= in;
endmodule
```



Определение регистров

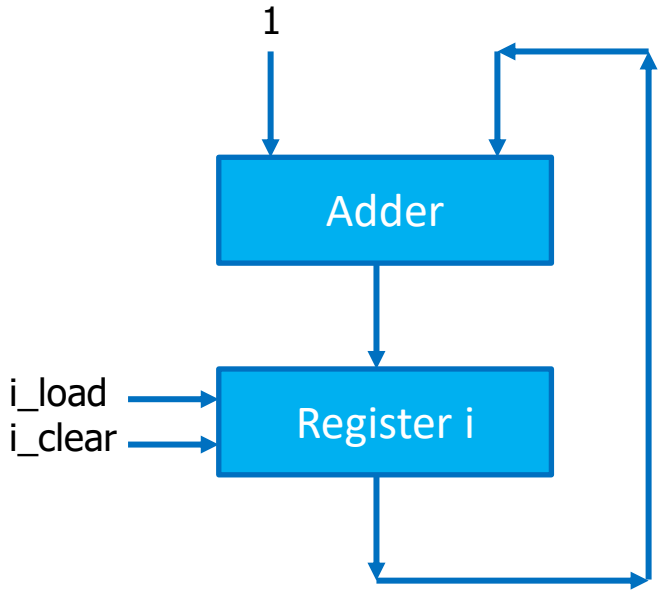
```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```



- Каждая переменная порождает свой регистр
- Возможно, что выбранная реализация требует дополнительные (служебные) регистры
- x_load = 0 – загрузка нового значения в регистр
- x_clear = 0 – сброс регистра в значение «0»

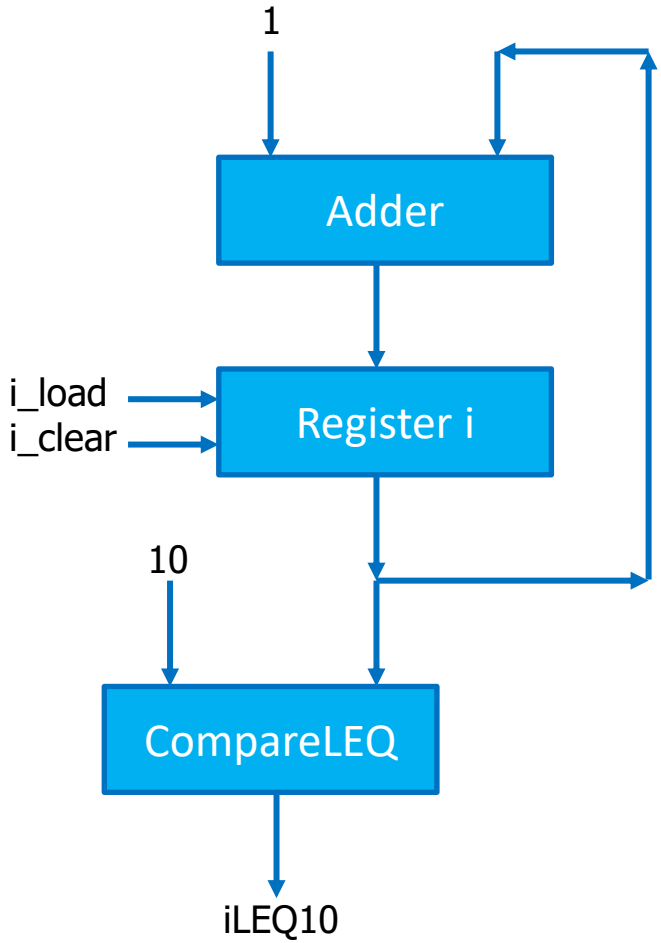
Построение операционного автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```



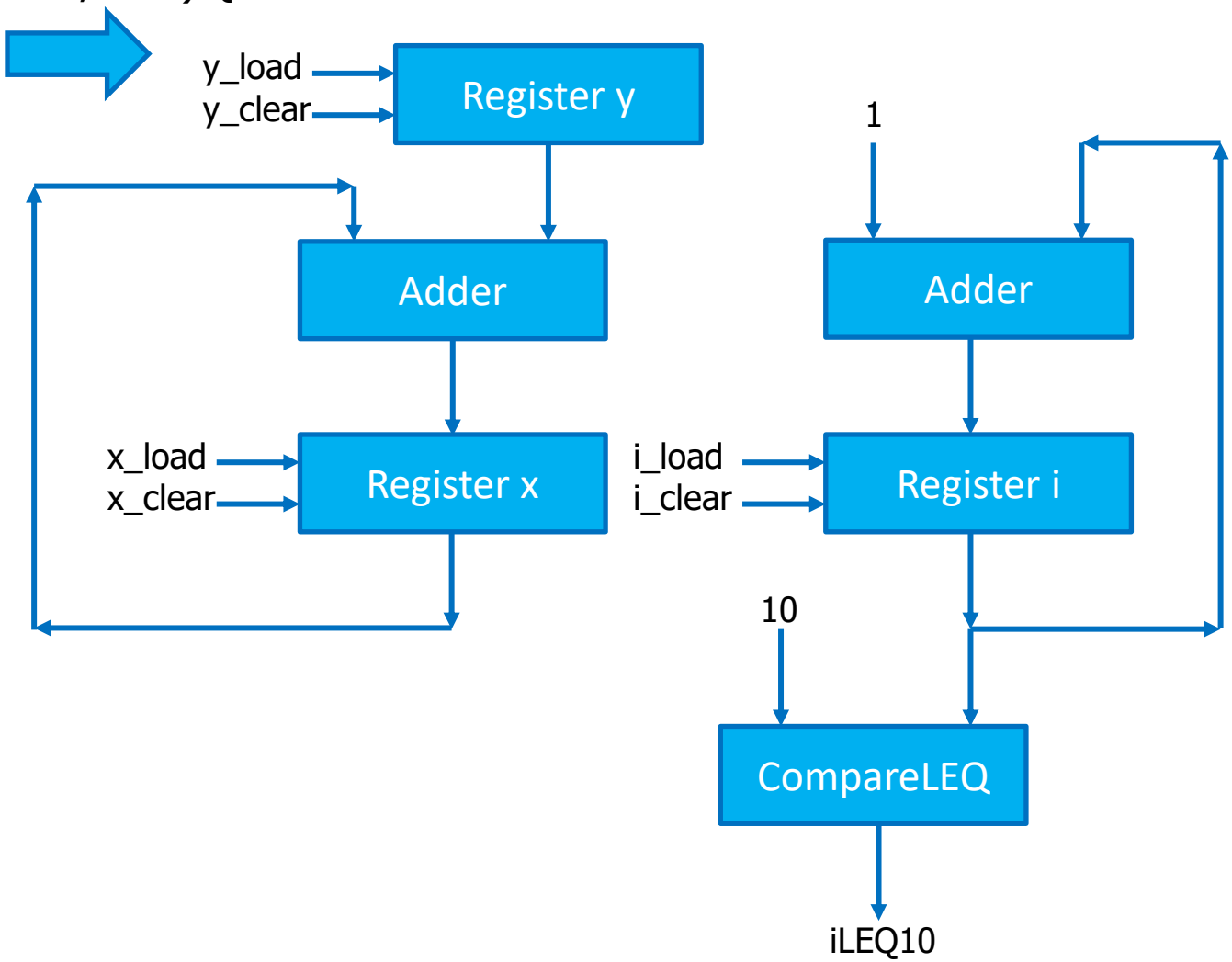
Построение операционного автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```



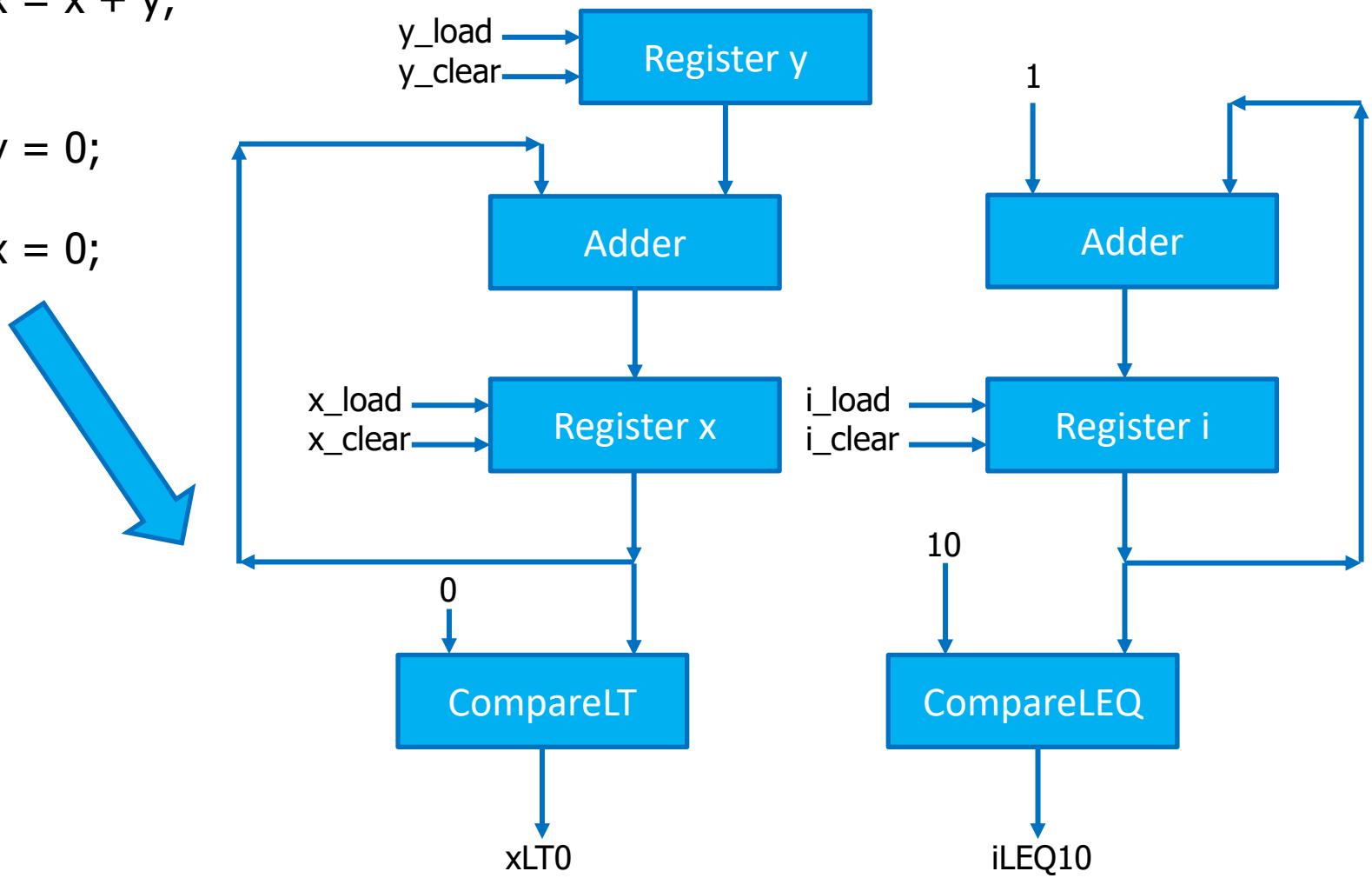
Построение операционного автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```



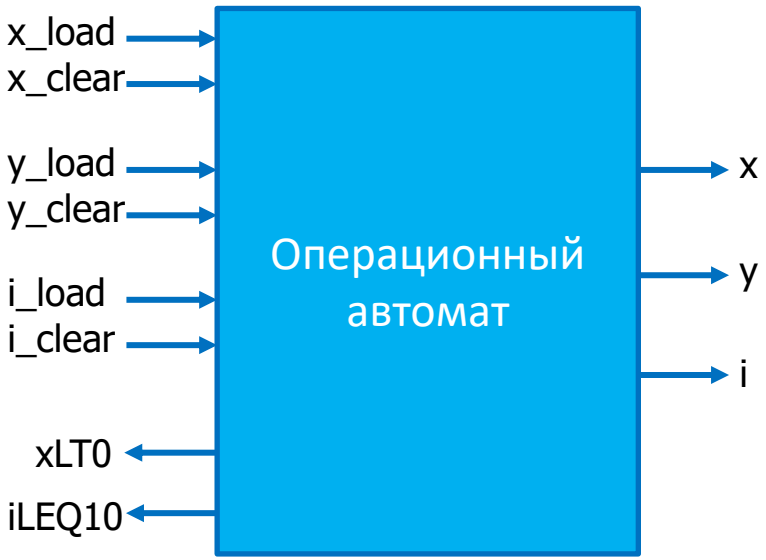
Построение операционного автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```



Построение операционного автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```



Построение управляющего автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```

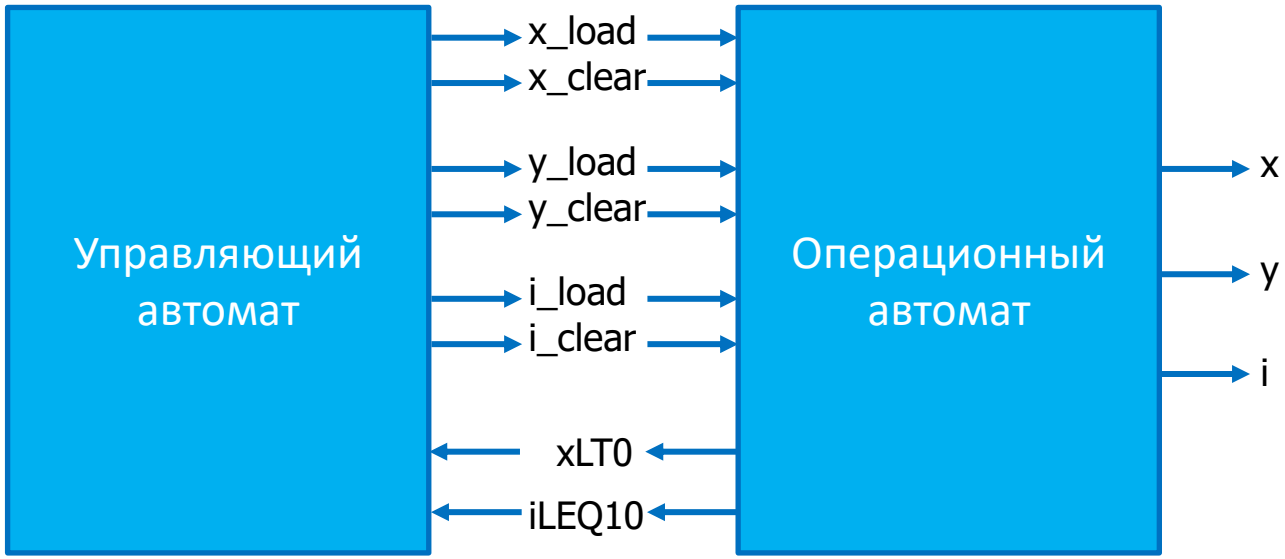


Диаграмма Мура управляющего автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```

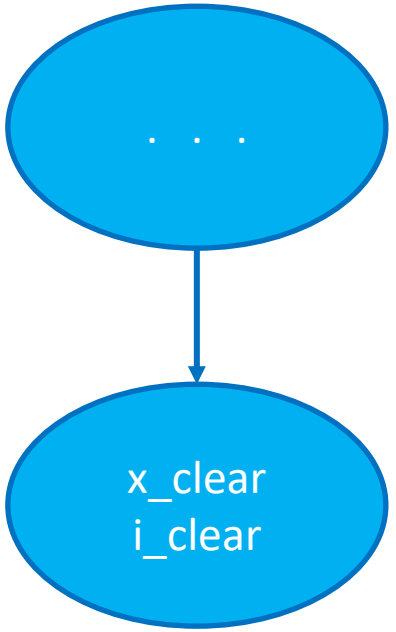


Диаграмма Мура управляющего автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```

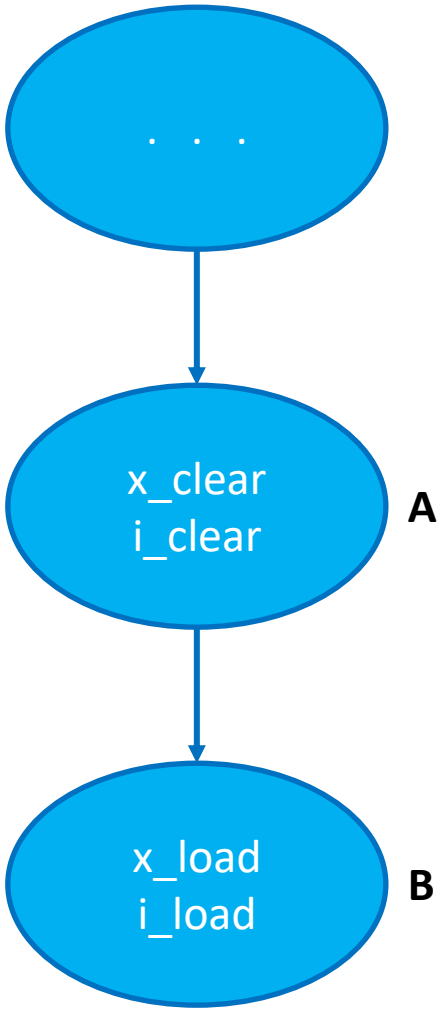


Диаграмма Мура управляющего автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```

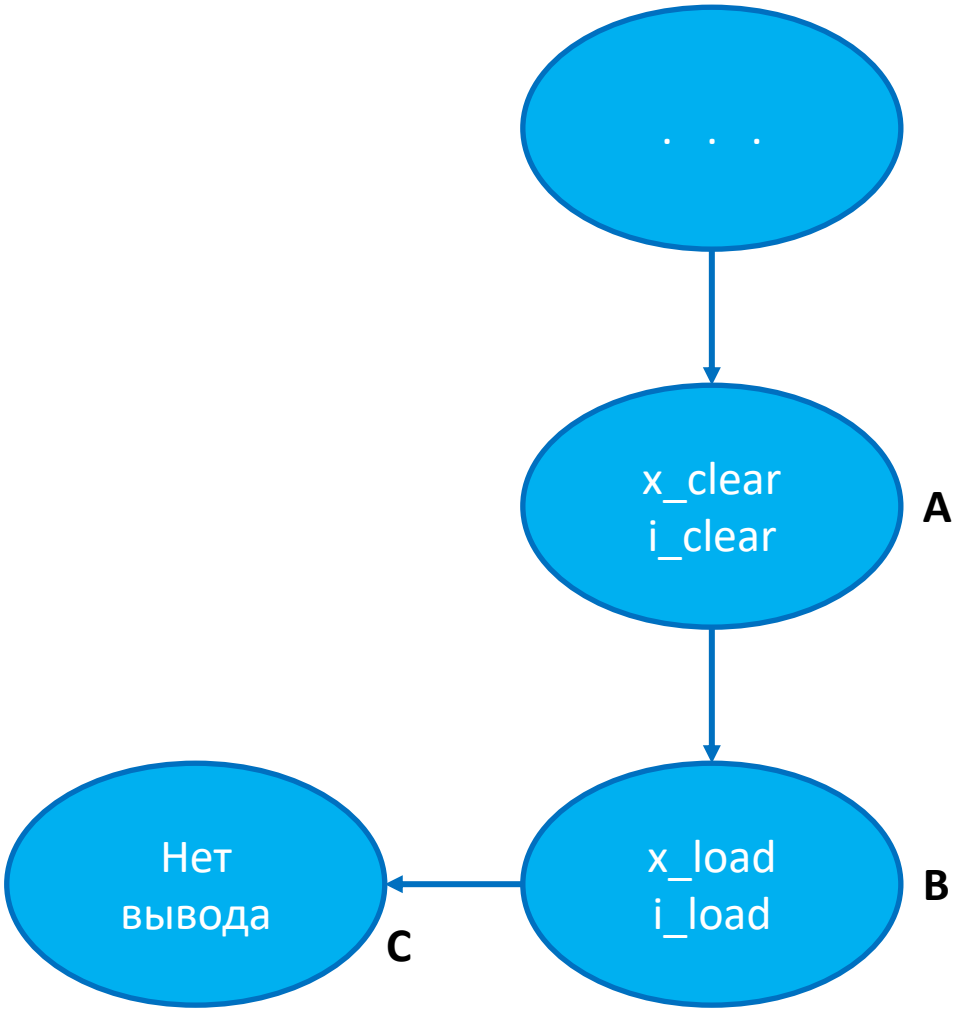


Диаграмма Мура управляющего автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```

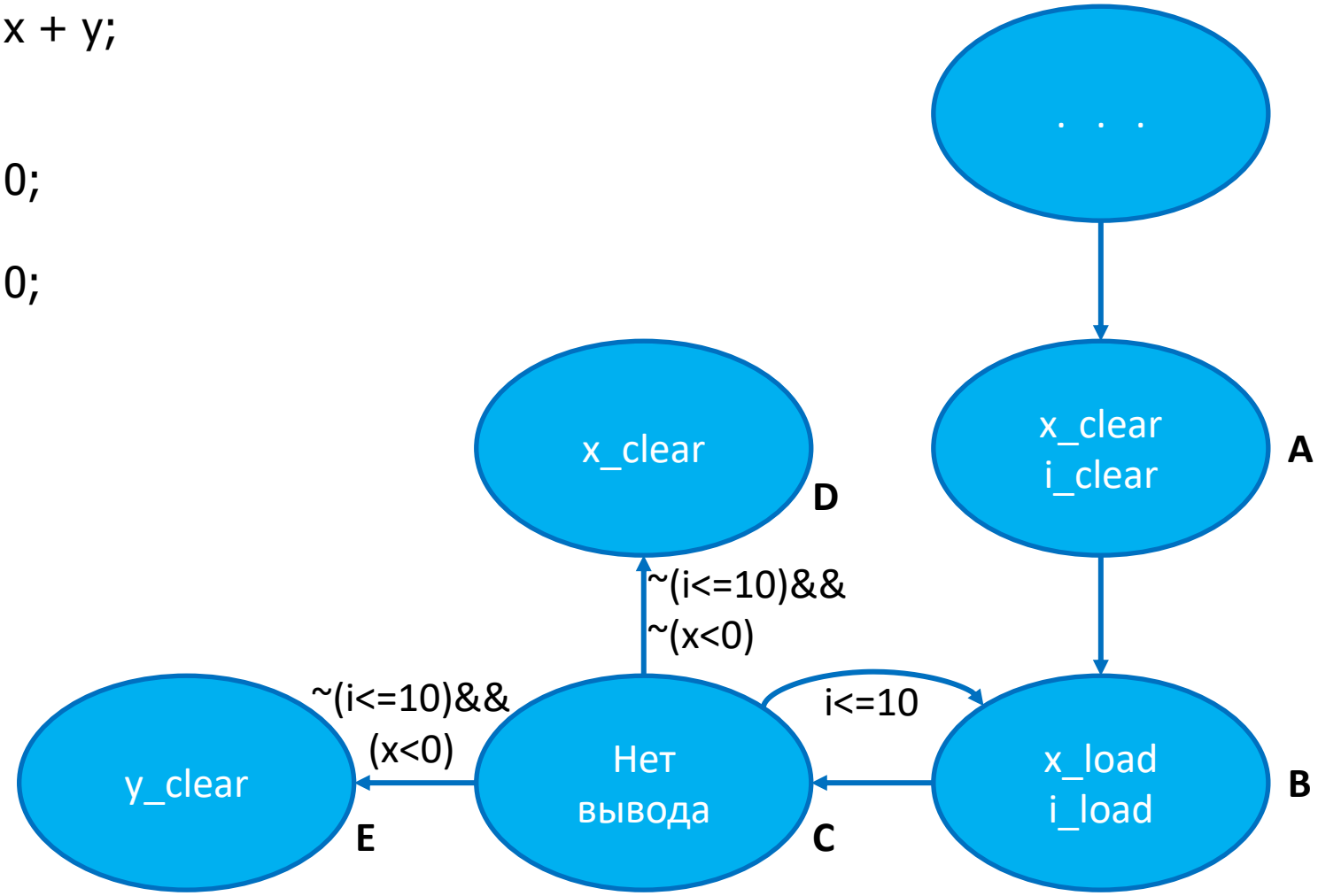
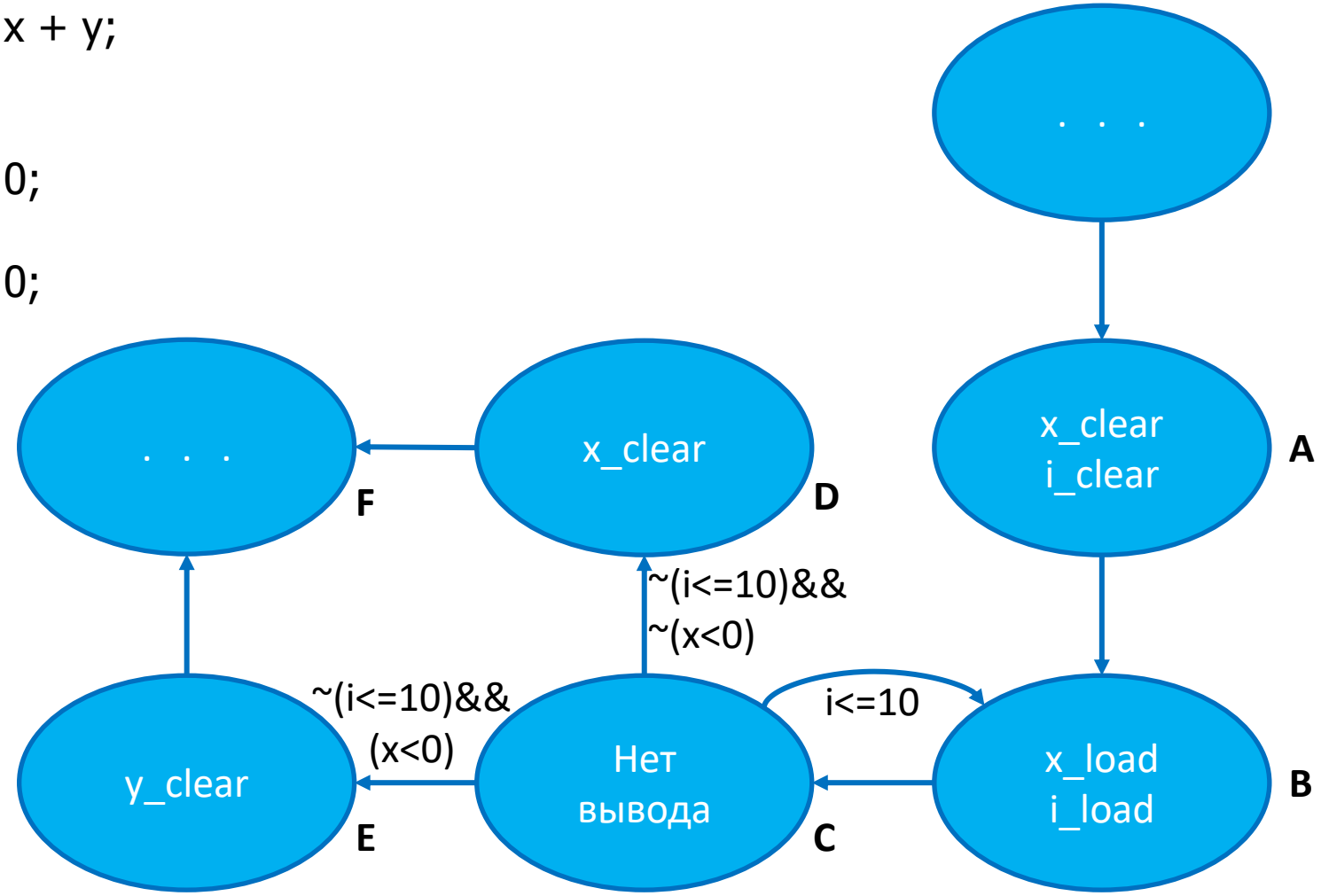


Диаграмма Мура управляющего автомата

```
for(x = 0, i = 0; i <= 10; i++) {
        x = x + y;
}
if(x<0) {
        y = 0;
} else {
        x = 0;
}
```



Построение управляющего автомата

- Кодирование состояний

A: `define A 3'b000

B: `define B 3'b001

C: `define C 3'b010

D: `define D 3'b011

E: `define E 3'b100

F: `define F 3'b101

Построение управляющего автомата

- Реализация комбинационной части

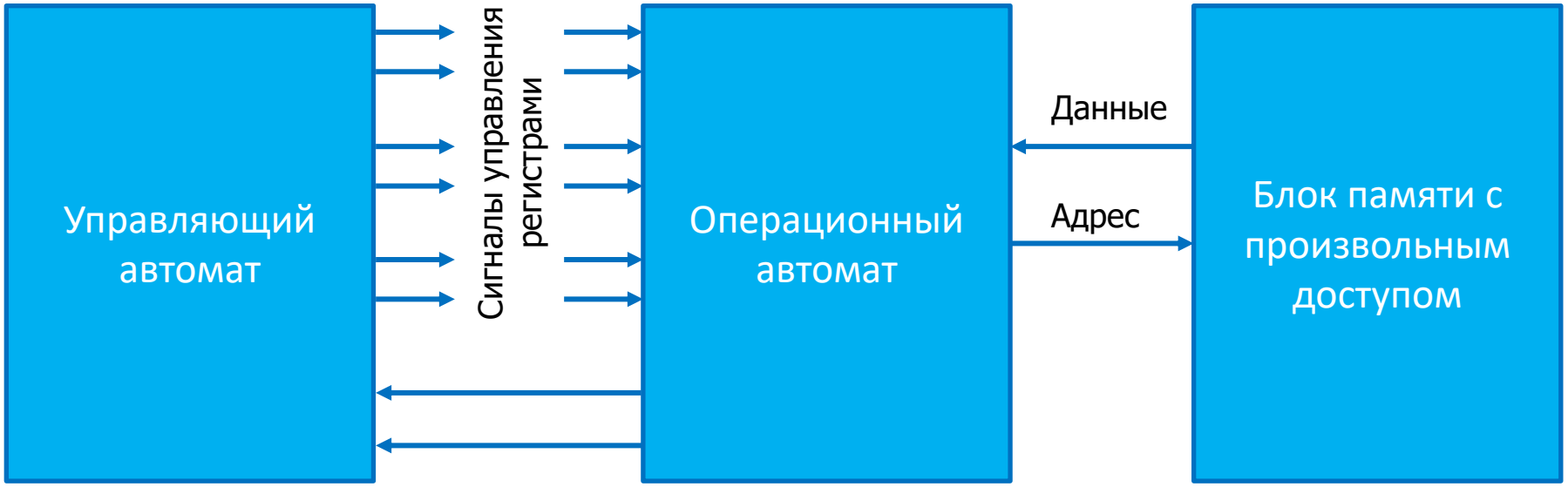
```
always @ (c_state, xLT0, iLEQ10)
        case (c_state)
                `A:                     `A:
                                                begin
                        …                               i_load = 1;
                `B:                                     i_clear = 0;
                        …                               x_load = 1;
                `C:                                     x_clear = 0;
                        …                               y_load = 1;
                `D:                                     y_clear = 1;
                        …                               n_state = `B;
                `E:                             end
                        …
                `F:
                        …
                default:
        endcase
```

Построение управляющего автомата

- Реализация элементов синхронизации

```
always @ (posedge clock, negedge reset)
        if (~reset)
                c_state <= `A;
        else
                n_state <= c_state;
```

Использование элементов памяти



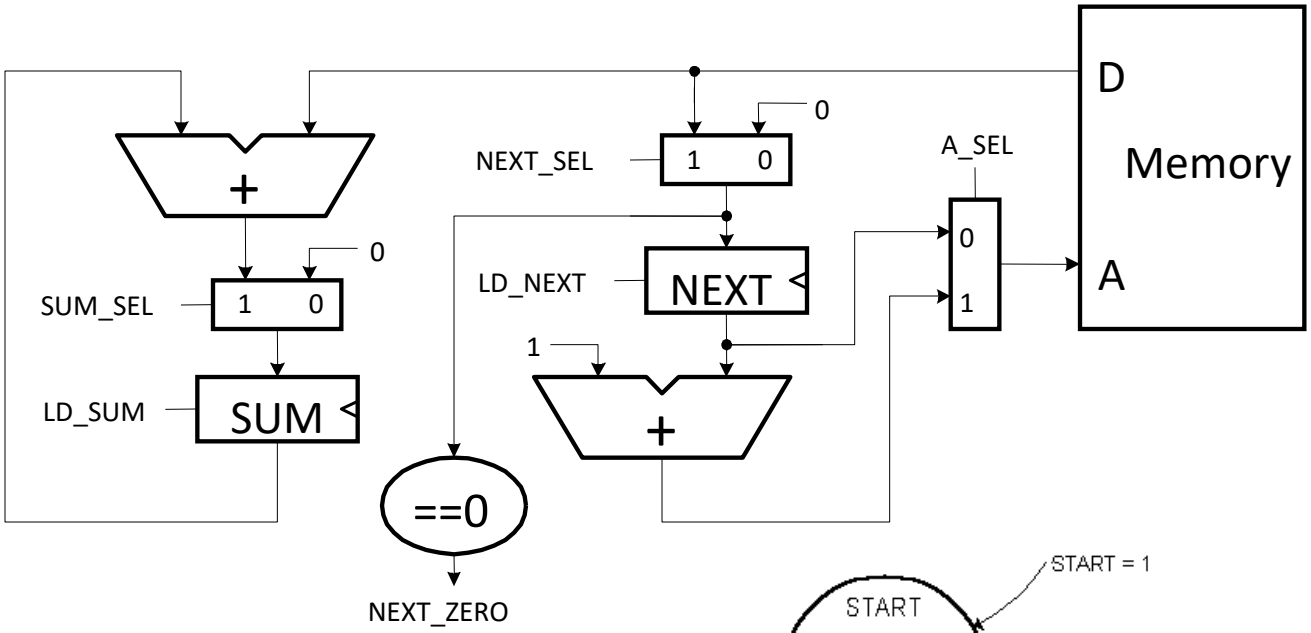
Файл инициализации памяти (.MIF)

```
DEPTH= 32;
WIDTH= 4;
ADDRESS_RADIX= HEX;
DATA_RADIX= BIN;
CONTENT
        BEGIN
                0 : 0000;
                1 : 0001;
                2 : 0010;
                3 : 0011;
                …
                1E : 1110;
                1F : 1111;
        END;
```

Связанный список

```
If (START==1) NEXT←0, SUM←0;
        repeat  {
                SUM←SUM + Memory[NEXT+1];
                NEXT←Memory[NEXT];
        } until (NEXT==0);
        R←SUM, DONE←1;
```

Аппаратная реализация связанного списка




```
If (START==1) NEXT←0, SUM←0;
   repeat  {
      SUM←SUM + Memory[NEXT+1];
      NEXT←Memory[NEXT];
   } until (NEXT==0);
R←SUM, DONE←1;
```