Языки описания схем

(mk.cs.msu.ru → Лекционные курсы → Языки описания схем)

Блок 22

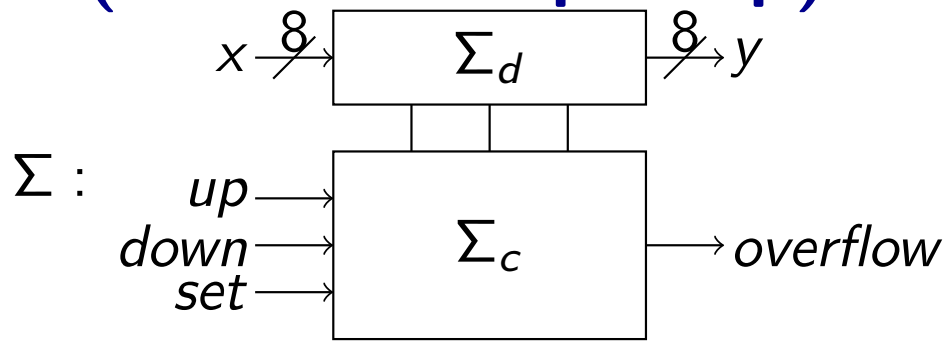
Как спроектировать
операционный автомат

Лектор:
Подымов Владислав Васильевич

E-mail:
valdus@yandex.ru

ВМК МГУ, 2020, осенний семестр

Напоминание (сквозной пример)



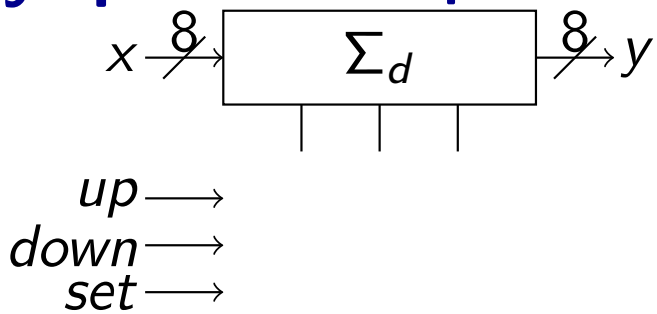
$y(1) = 0$

$overflow(1) = 0$

$$y(t+1) = \begin{cases} x(t), & \text{если } set(t) = 1; \\ y(t)+1, & \text{если } set(t) = 0 \text{ и } up(t) = 1; \\ y(t)-1, & \text{если } set(t) = up(t) = 0 \text{ и } down(t) = 1 \\ y(t) & \text{иначе} \end{cases}$$

$overflow(t+1) = 1 \quad \Leftrightarrow$
при переходе от $y(t)$ к $y(t+1)$ происходит арифметическое переполнение

Забываем про управляющий автомат



Забудем на время про

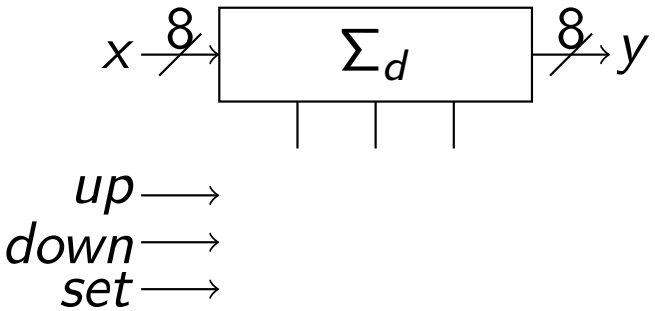
- подсхему Σ_c и
- **точную** зависимость значения $y(t+1)$
  от значений $up(t)$, $down(t)$, $set(t)$, $overflow(t)$, и

попробуем реализовать *операционный автомат* Σ_b,
отталкиваясь **только** от спектра значений $y(t+1)$

Для этого зададимся таким вопросом:

> **Что** в целом должна уметь делать схема
> (с данными x и, быть может, не только с ними),
> чтобы всегда можно было предоставить требуемые данные y?

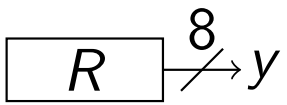
Проектируем память



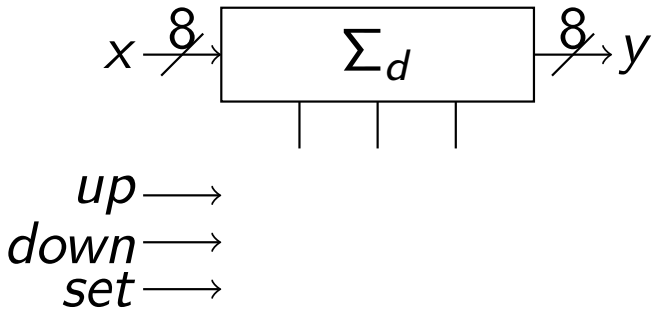
1. Определимся с основными ячейками памяти операционного автомата

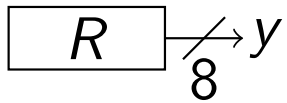
Данные y **не**однозначно определяются значениями на входах — значит, их потребуется извлекать из триггеров/регистров

Пойдём по простому пути:
y — **состояние параллельного регистра** R



Определяем вспомогательные подсхемы





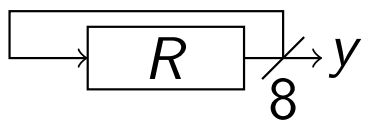
2. Для каждого способа преобразования данных (*независимо от остальных*) подумаем, что потребуется добавить в схему

Первый способ: $y(1) = 0$

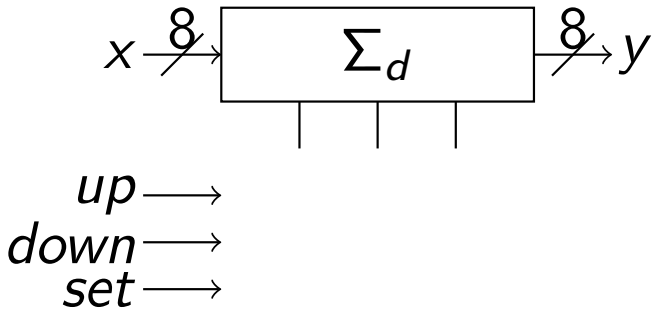
Значит, R — регистр со сбросом, и больше ничего не требуется

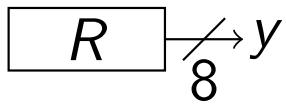
Второй способ: $y(t+1) = y(t)$

Достаточно направить выход R на вход:



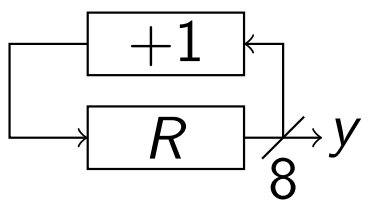
Определяем вспомогательные подсхемы



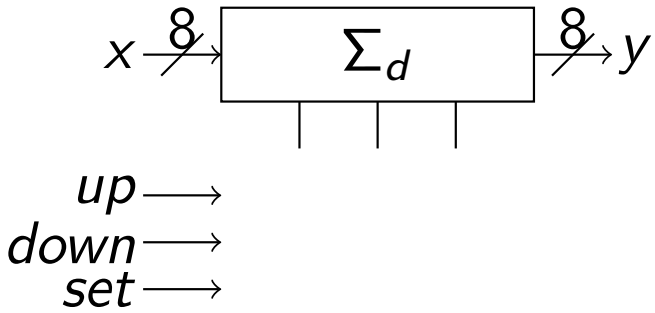


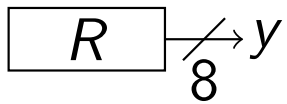
Третий способ: $y(t+1) = y(t) + 1$

Достаточно направить выход R на вход,
поставив посередине подходящую *комбинационную* схему:



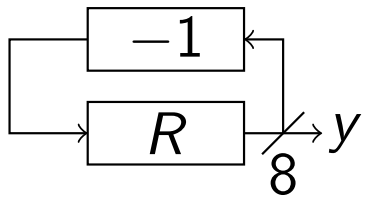
Определяем вспомогательные подсхемы





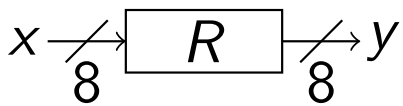
Четвёртый способ: $y(t+1) = y(t) - 1$

Здесь всё то же самое, только комбинационная схема другая:



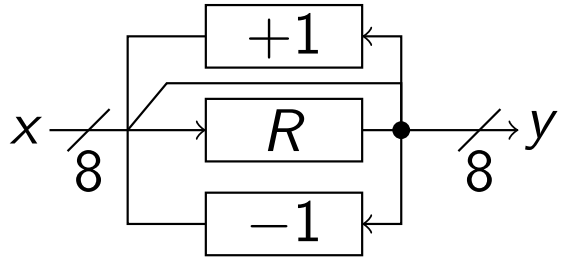
Пятый способ: $y(t+1) = x(t)$

Достаточно направить x на вход R:



Совмещаем вспомогательные подсхемы

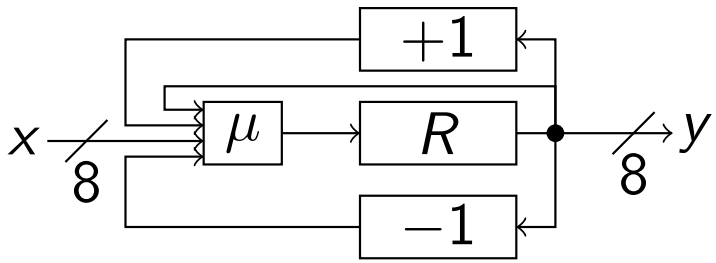
3. Совместим получившиеся схемы, “наложив” их друг на друга



Кажется, что-то пошло не так ☹

Возник **конфликт пересылки данных**: при разных обстоятельствах на вход R посылаются значения из разных точек схемы

Типовой способ разрешения такого конфликта: добавим *мультиплексор* (μ) в точку конфликта, и пусть **управляющий автомат** решает, какое значение должно быть на *управляющем входе* μ



Итог

Так обычно и разрабатывается операционный автомат:

1. Приблизительно расставляются регистры данных
2. Всё то, **что** операционный автомат должен уметь делать с данными, покрывается подсхемами
3. В места возникновения конфликтов пересылки данных вставляются мультиплексоры
   (*и в оба автомата добавляются соответствующие управляющие порты*)

Промежуточный итог для сквозного примера: