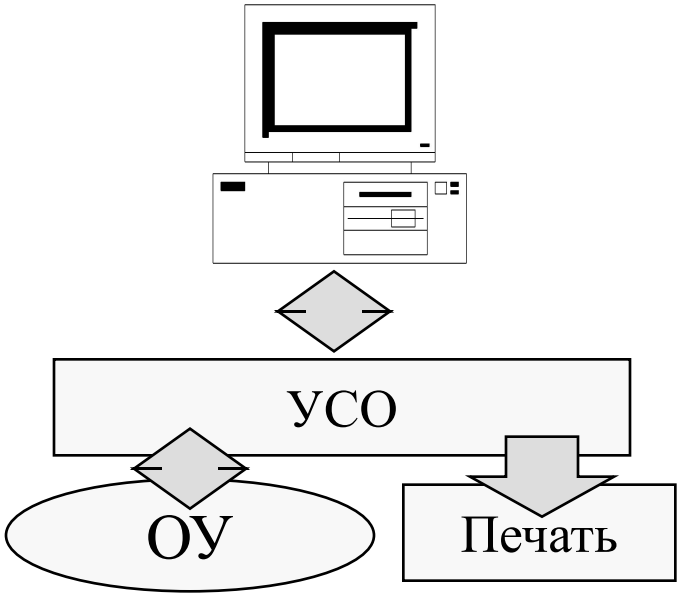
4. Монопольные (Standalone) приложения

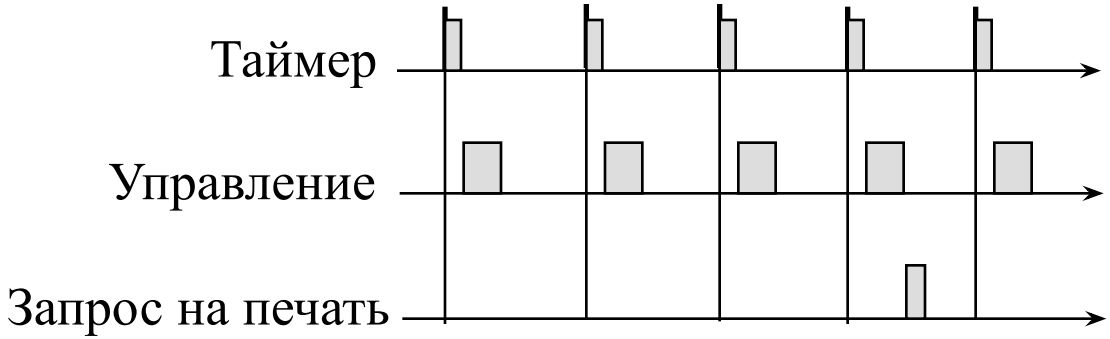
Standalone приложение:

- монопольно использует ресурсы целевой платформы;
- поддерживается платформо-ориентированной (native) библиотекой драйверов устройств (драйвер осуществляет управление устройством, реализует процедуры ввода/вывода);

- может включать в себя самостоятельные процедуры обработки прерываний и действия по вводу/выводу;
- сильно зависит от платформы

Пример





Управление:

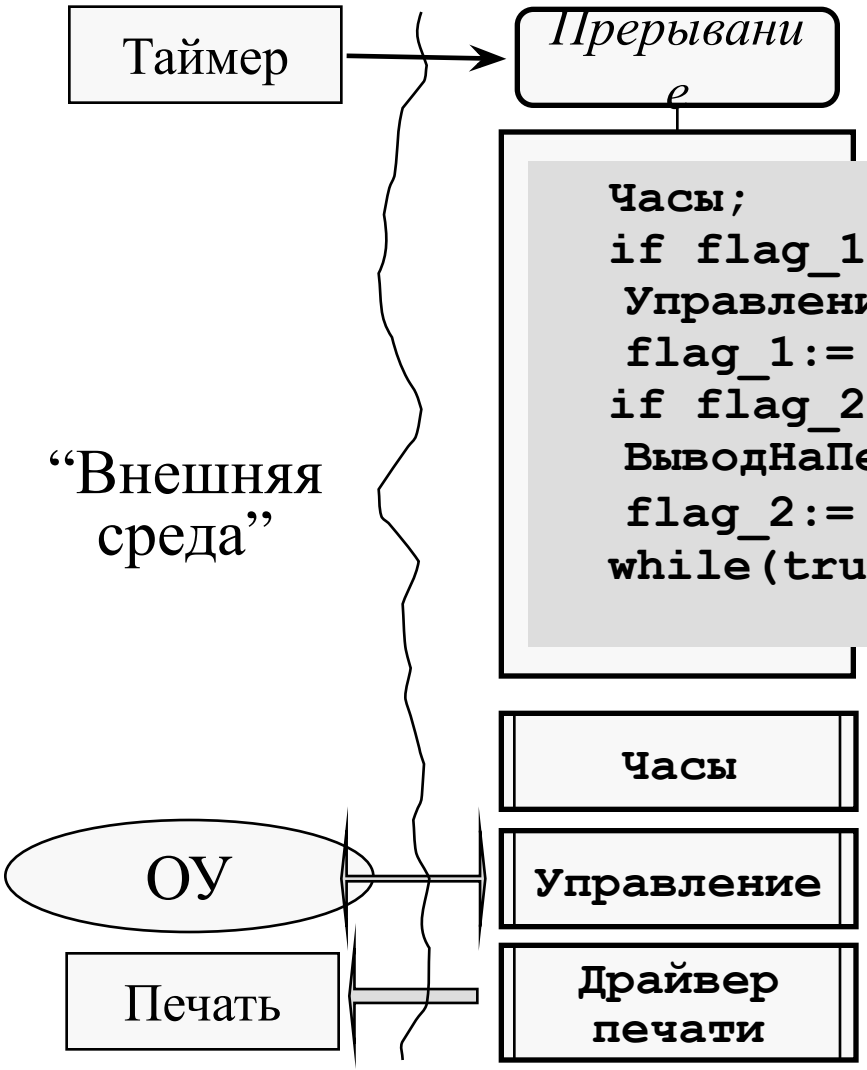
- Период = 1 s
- WCET < 0.3 ms
- Функции
  - Измерение
  - Принятие решения
  - Вывод воздействия

Запрос на печать

- Период = 180 s
- WCET < 0,05 ms
- Функции
  - Запуск печати:

```
ВыводНаПечать(Данное: X)
```

Пример (2) Возможная реализация



Таймер:
Запрос на прерывание каждую секунду

Часы:
реализует два счетчика, значения которых увеличиваются на 1 при каждом обращении; когда

```
счетчик_1 = 1   flag_1:= true
счетчик_2 = 180 flag_2:= true
```

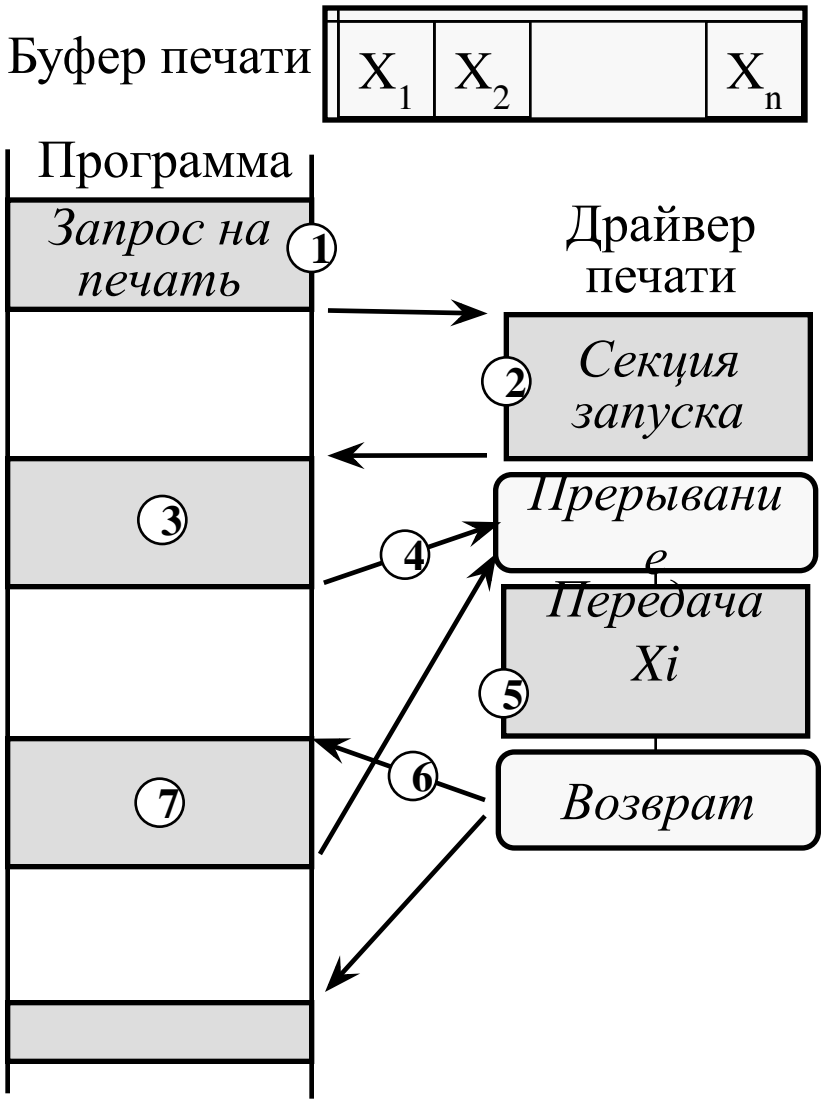
Драйвер печати:
Библиотечная программа

Пример (3) Возможная реализация

Предлагаемая реализация некорректна - вывод на печать время от времени “зависает”

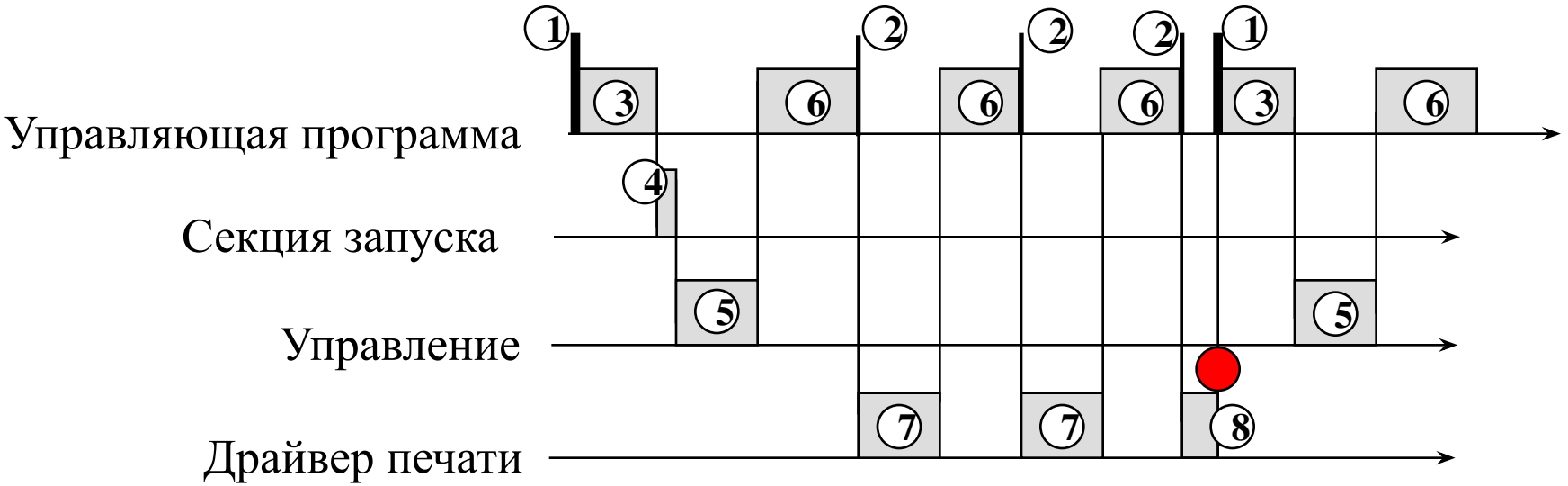
Необходимо более детальное рассмотрение работы системы

Пример (4) Работа печати



1. Запрос: **`ВыводНаПечать(Данное: X)`**
2. Секция запуска считывает X в буфер вывода, пересылает порцию X_1 на устройство и возвращает управление
3. Программа и устройство работают параллельно
4. Устройство закончило печать X_1, и формирует запрос на прерывание
5. Обработчик прерывания драйвера пересылает X_2 на устройство
6. Возврат в точку прерывания
7. Программа и устройство работают параллельно
8. И так далее для X_2 X_3 X_4 ...

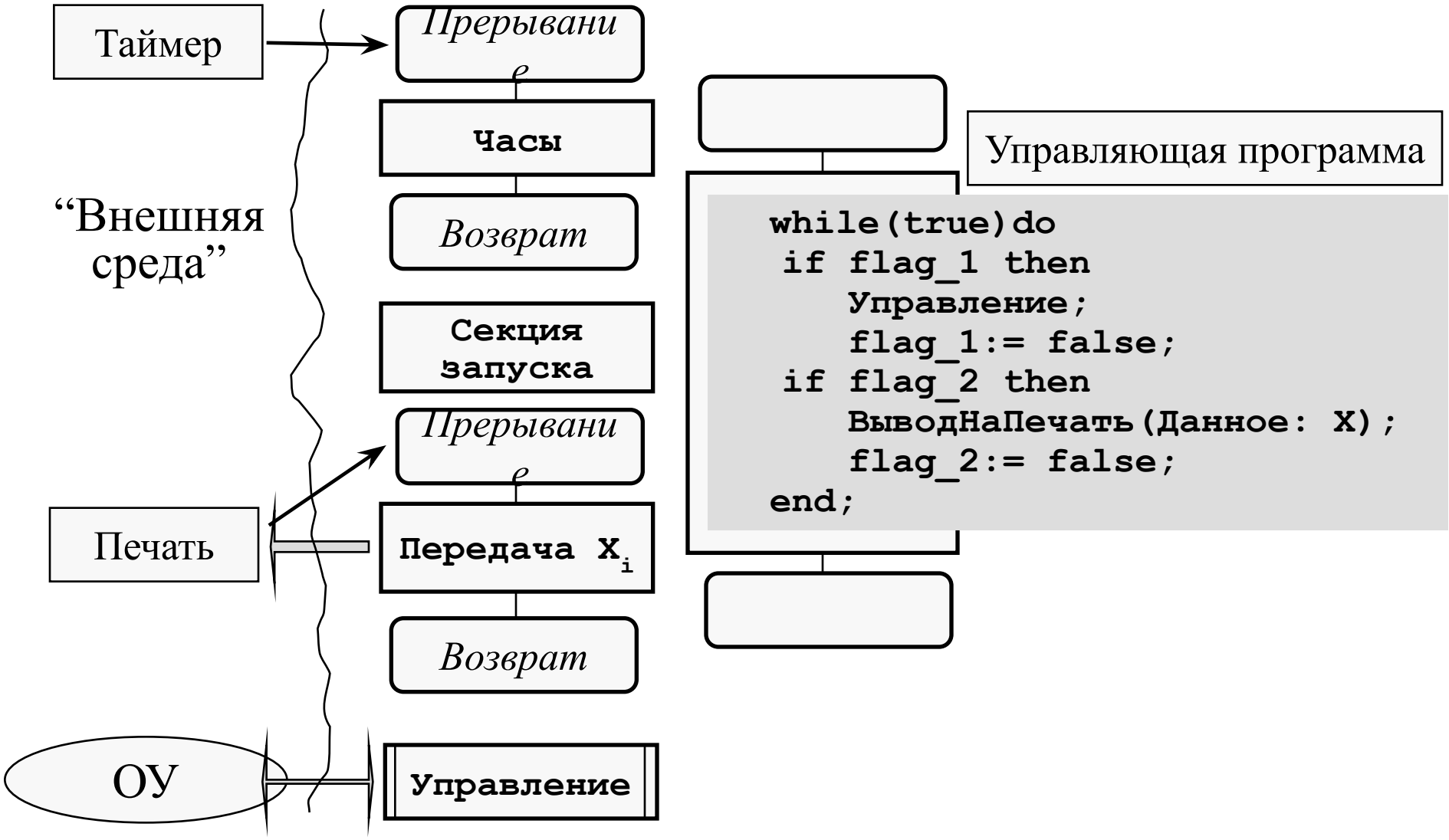
Пример (5) Работа печати



1. Прерывания от таймера
2. Прерывания печати
3. Часы
4. Секция запуска драйвера печати
5. Управление
6. Занятое ожидание while(true) do end
7. Обработка прерывания печати
8. Драйвер прерывается таймером (!)

Некорректно реализован возврат из процедуры обработки прерывания таймера (Управляющая программа). Вследствие этого зависает драйвер печати

Пример (5) Корректная реализация



Проблемы

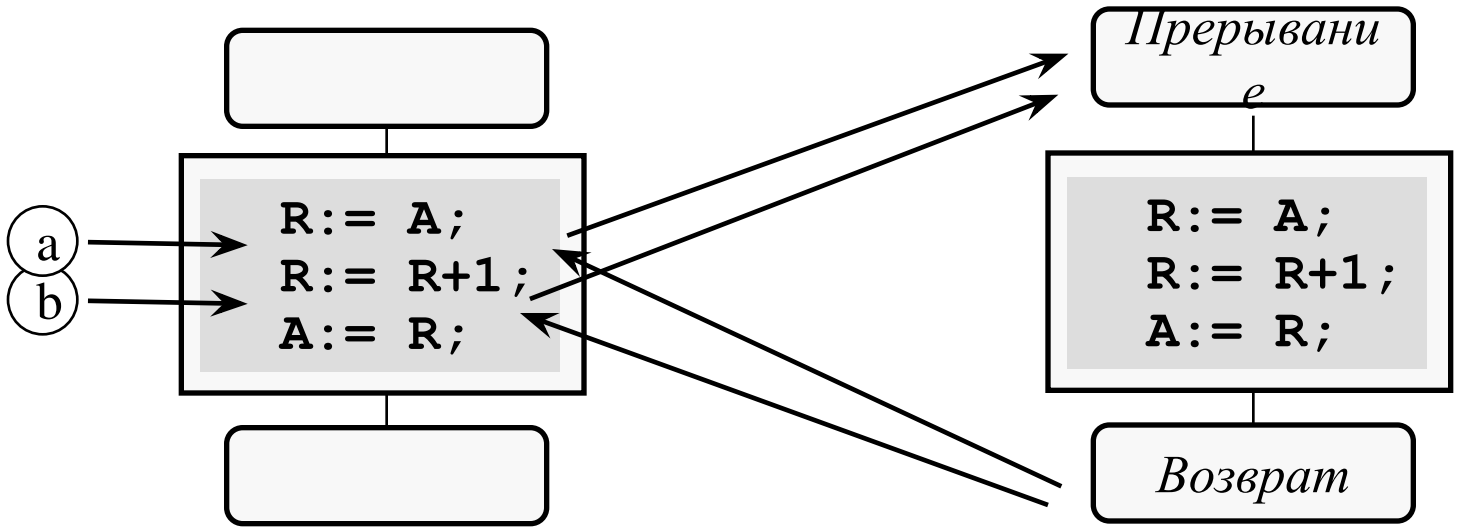
Критическая секция

КС - участок программы, выполнение которого не должено прерываться

Синхронизация

согласование во времени выполнения заданных участков программы

Критическая секция



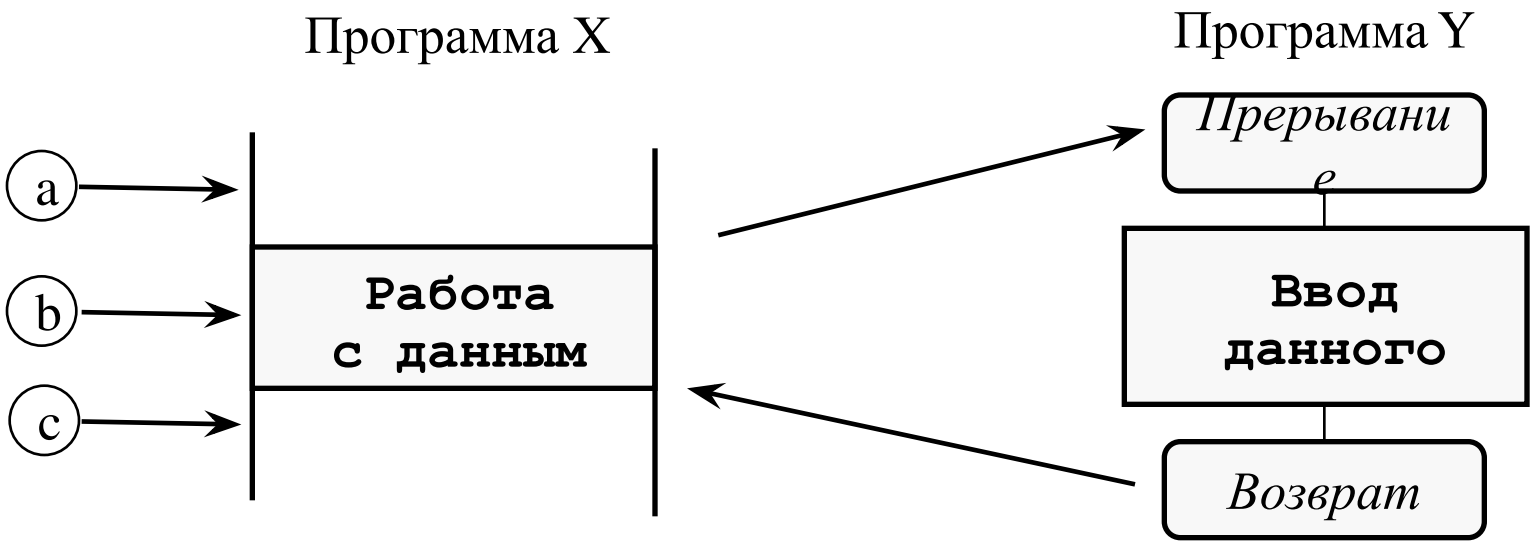
a) A = 5; R = 5; ***R = 5; R = 6; A = 6;*** R = 7; A = 7;

b) A = 5; R = 5; R = 6; ***R = 5; R = 6; A = 6;*** A = 6;

Возможное средство:

маскировать прерывания (CLI …… STI). Однако, КС не должна быть длинной – увеличение latancy (В предыдущем примере недопустимо целиком маскировать обработку прерывания печати)

Синхронизация



Возможное средство: механизм флагов

Идея:

Программа X:

```
flag := false;
repeat until flag;
Работа с данным;
flag := false;
```

Программа Y:

```
Ввод данного;
flag := true;
```

Механизм флагов - реализация

Флаги реализуются на системном (аппаратном) уровне и поддерживаются неделимыми примитивами доступа

```
class Flag {
 boolean flag;

 void setFlag(boolean value){
    установить_маску;
    flag := value;
    сбросить_маску;
 }

 boolean getFlag(){
    return flag;
 }
}
```

Особенности Standalone приложений

Разработка Standalone приложений требует:

- написания последовательно выполняющихся процедур, обрабатывающих события от параллельно развивающихся физических процессов и отслеживать возможные конфликты, возникающие в процессе их выполнения
- написания собственных обработчиков прерываний
- решений вопросов синхронизации и обеспечения неделимости
- программирования ввода/вывода