Lecture 7
System architecture
Input-output

Computing platforms, semester 2

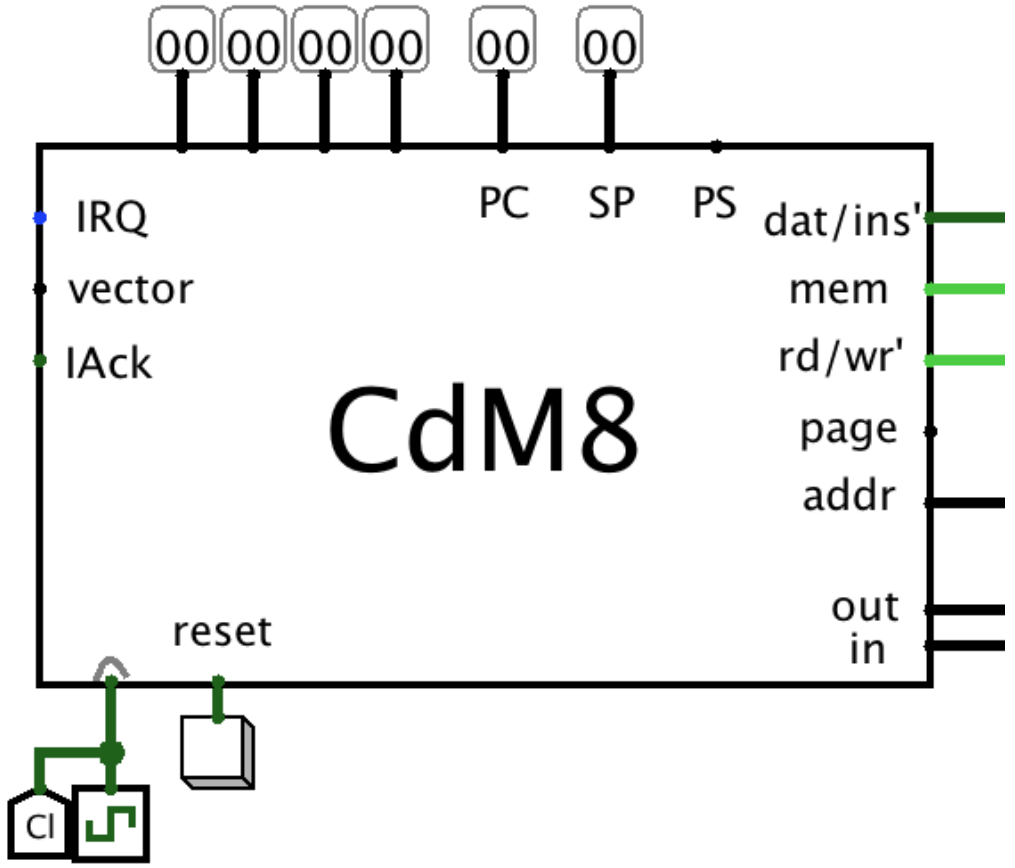
Novosibirsk State University
University of Hertfordshire

D. Irtegov, A.Shafarenko

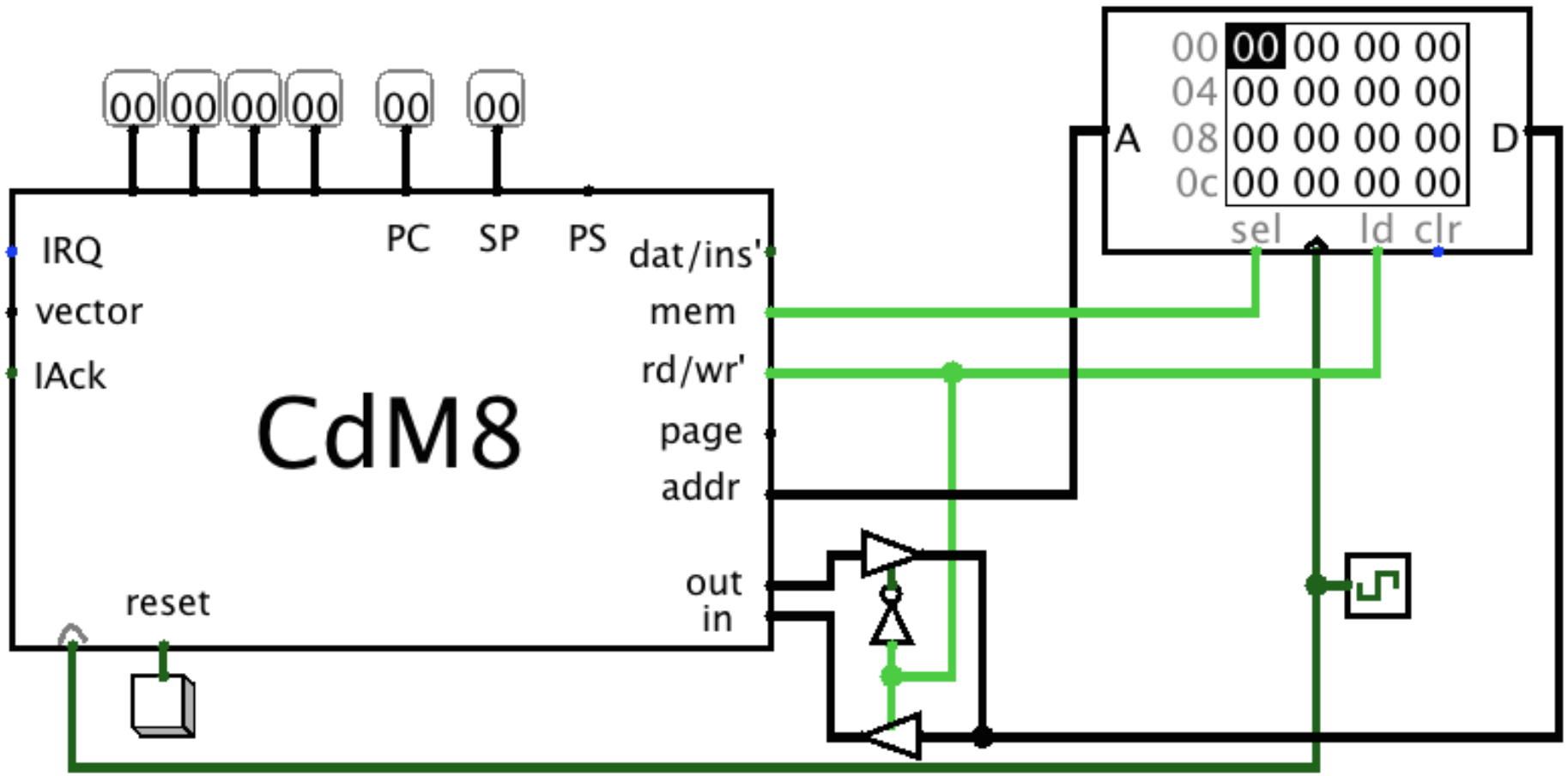
2019

CdM-8 as a chip

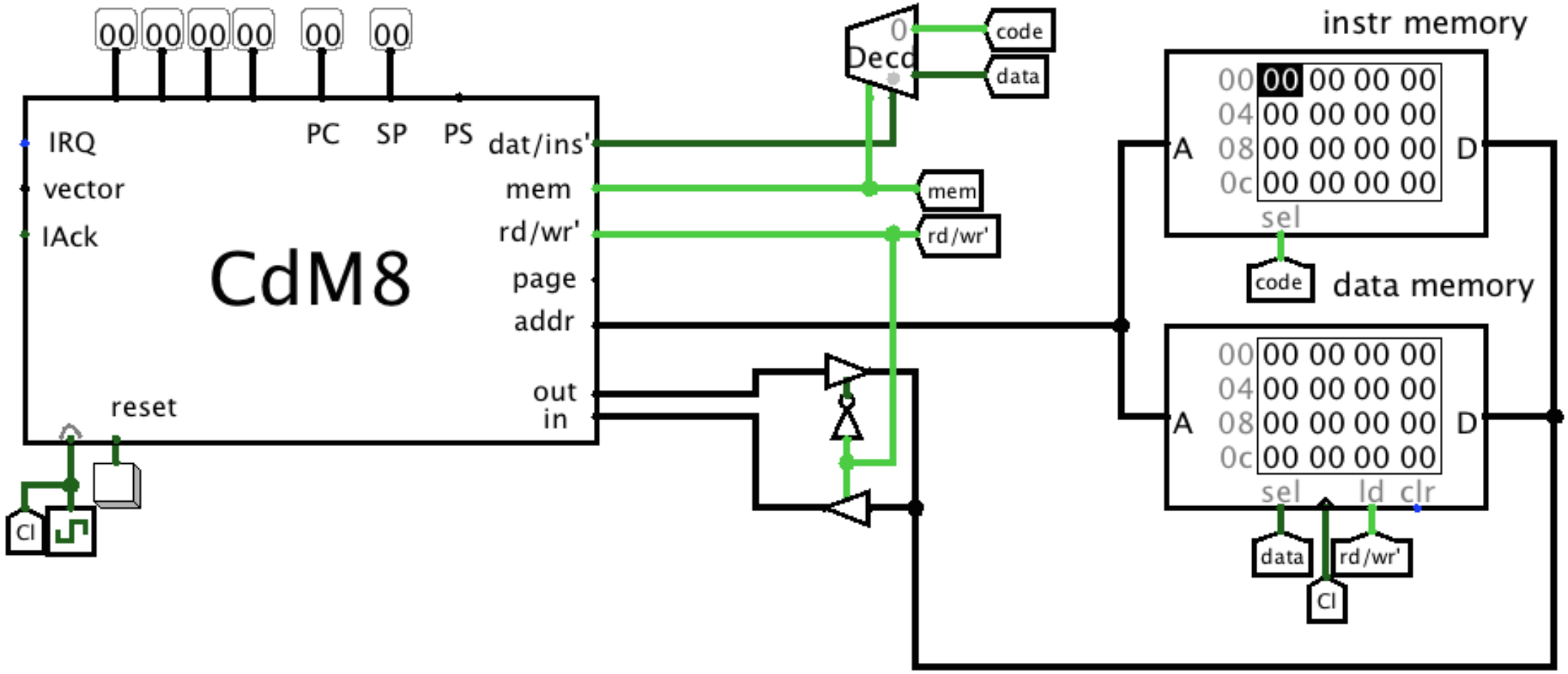
Unmarked outputs on the top: register monitors
IRQ, vector and IAck we will discuss later
In and out buses are memory bus

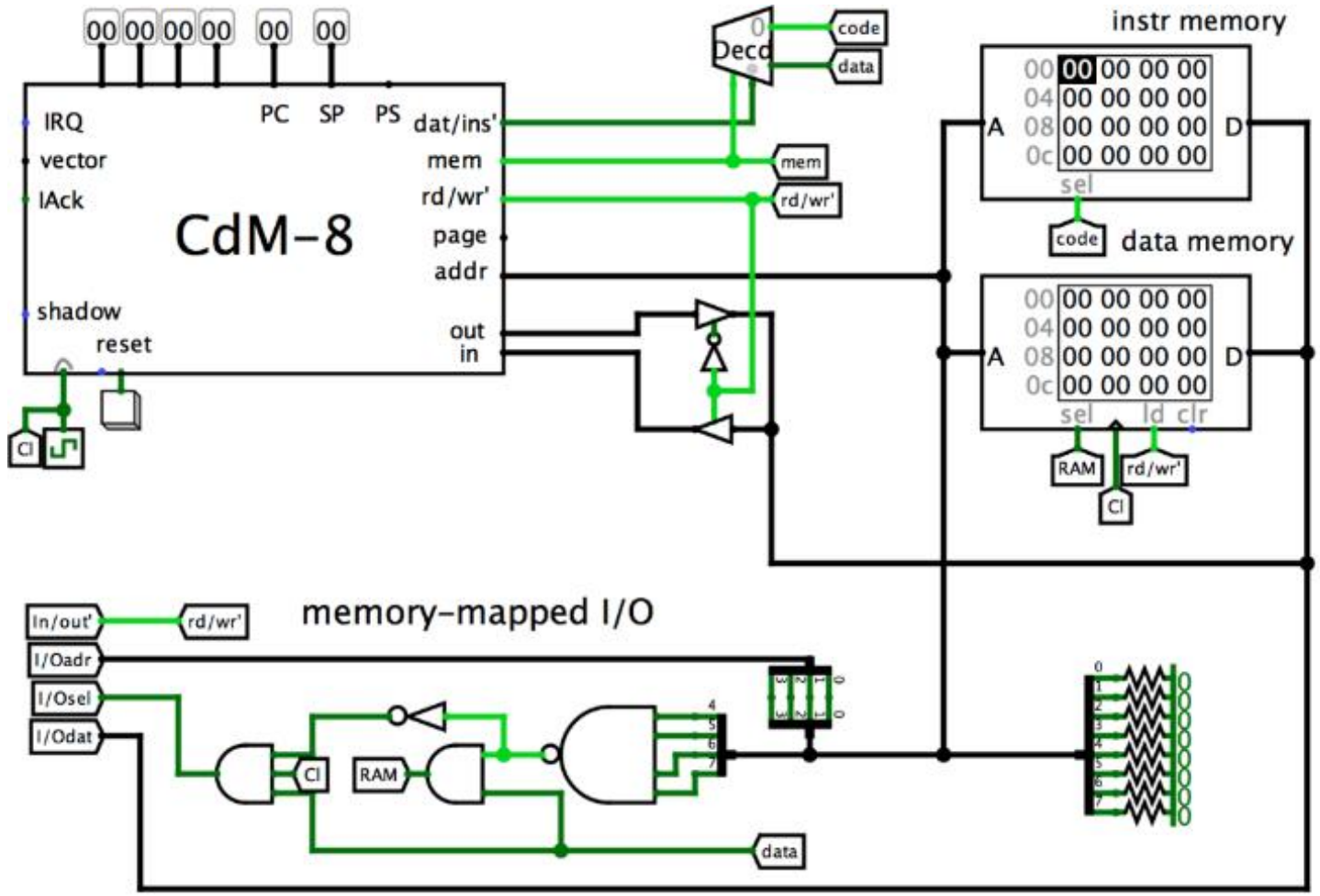


CdM-8 in von Neumann (Manchester) setup



CdM-8 in Harvard setup

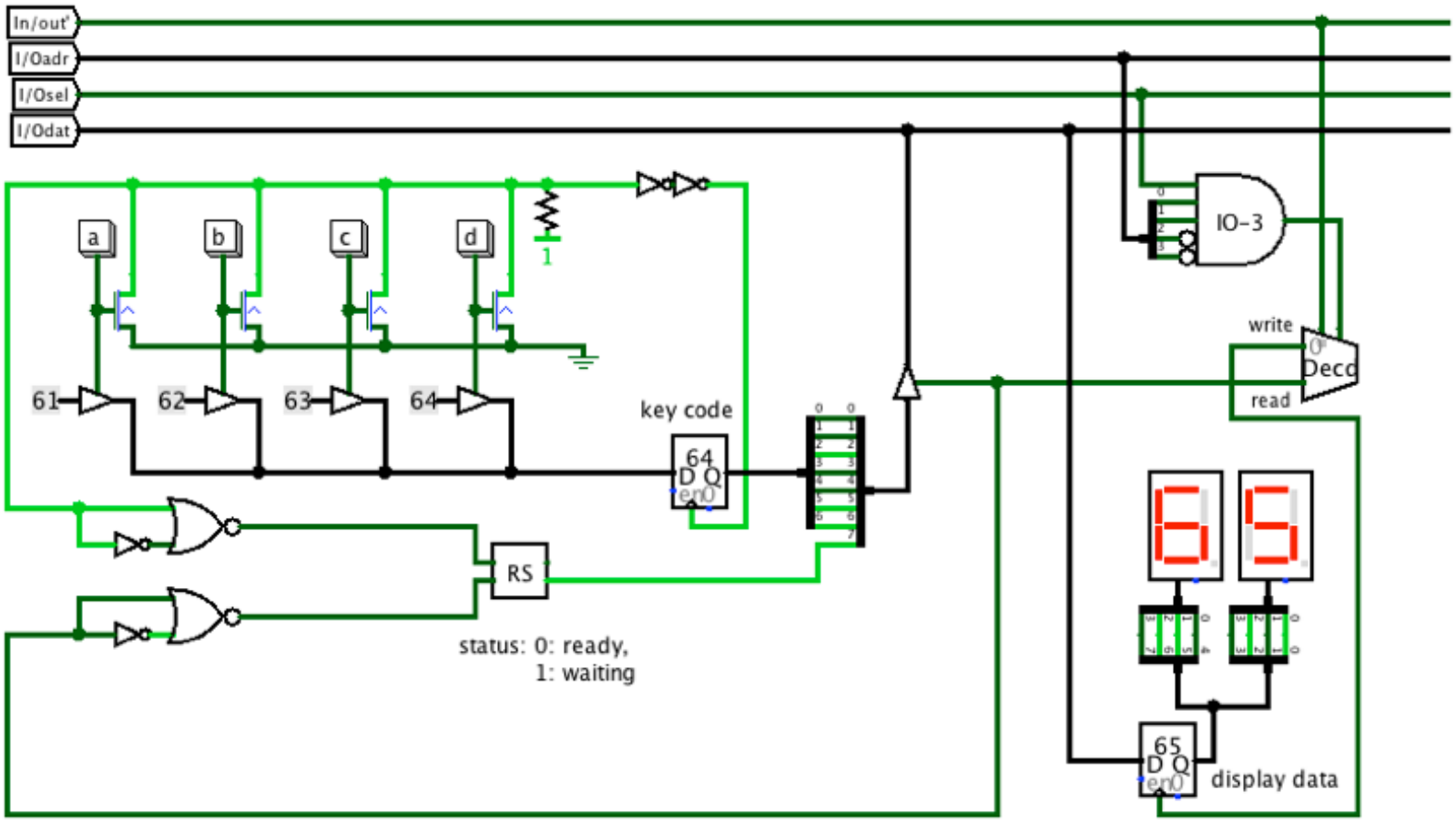



00 00 00 00 00 00
IRQ
vector
IAck
shadow
reset
PC
SP
PS
dat/ins'
mem
rd/wr'
page
addr
out
in
CdM-8
Decd
code
data
mem
rd/wr'
instr memory
A
D
sel
code
data memory
RAM
rd/wr'
ld
clr
In/out'
I/Oadr
I/Osel
I/Odat
memory-mapped I/O
Cl
RAM
data

I/O as a memory cell

- Sequential logical device is
  - a set of registers
  - some combinatory logic
- If we make some registers available to CPU, we can
  - Read [parts of] state of the device
  - It is up to device designers to make this part of the state usable
  - Change state of the device (issue commands and transfer data)

I/O device: [part of] keyboard and a display



Reading data from the keyboard

- Uppermost (7^{th}) bit of data latch is reset to 0 on key press
- And to 1 on data read
- Thus, CPU can poll (repeatedly read) the data until bit 7 is 0
- While polling, CPU cannot do anything else
  - Also, polling leads to high CPU load and power consumption
- If CPU does something else (timed polling),
  it cannot react to keyboard immediately
- In multithreaded programming, polling is considered harmful
- But on low level I/O you sometimes have no other choice

Code sample

```
        asect 0xf3
IOReg:  # Gives the address 0xf3 the symbolic name IOReg
        asect 0xf0
stack:  # Gives the address 0xf0 the symbolic name stack
        asect 0x00
start:
        setsp stack     # sets the initial value of SP to 0xf0

        ldi  r0, IOReg  # load the address of the keyboard and display in r0

readkbd:
        do      # begin the keyboard read loop
          ld r0,r1      #   load r1 from data memory, which includes
                #   the I/O address space
                #   now bits 0..6 of r1 encode the last char typed,
                #   and bit 7 indicates the keyboard status
          tst r1        #   flag N is taken from bit 7 of the register
        until pl        # drop out of the loop when the N flag is 0
                # and now r1 contains the ASCII code of the last char
        st r0,r1        # display the hex of the ASCII code

        br readkbd      # go back to the start of the keyboard read loop
        end
```

Присваивает адресу 0xf3 символическое имя Org

Interrupts (alternative to polling)

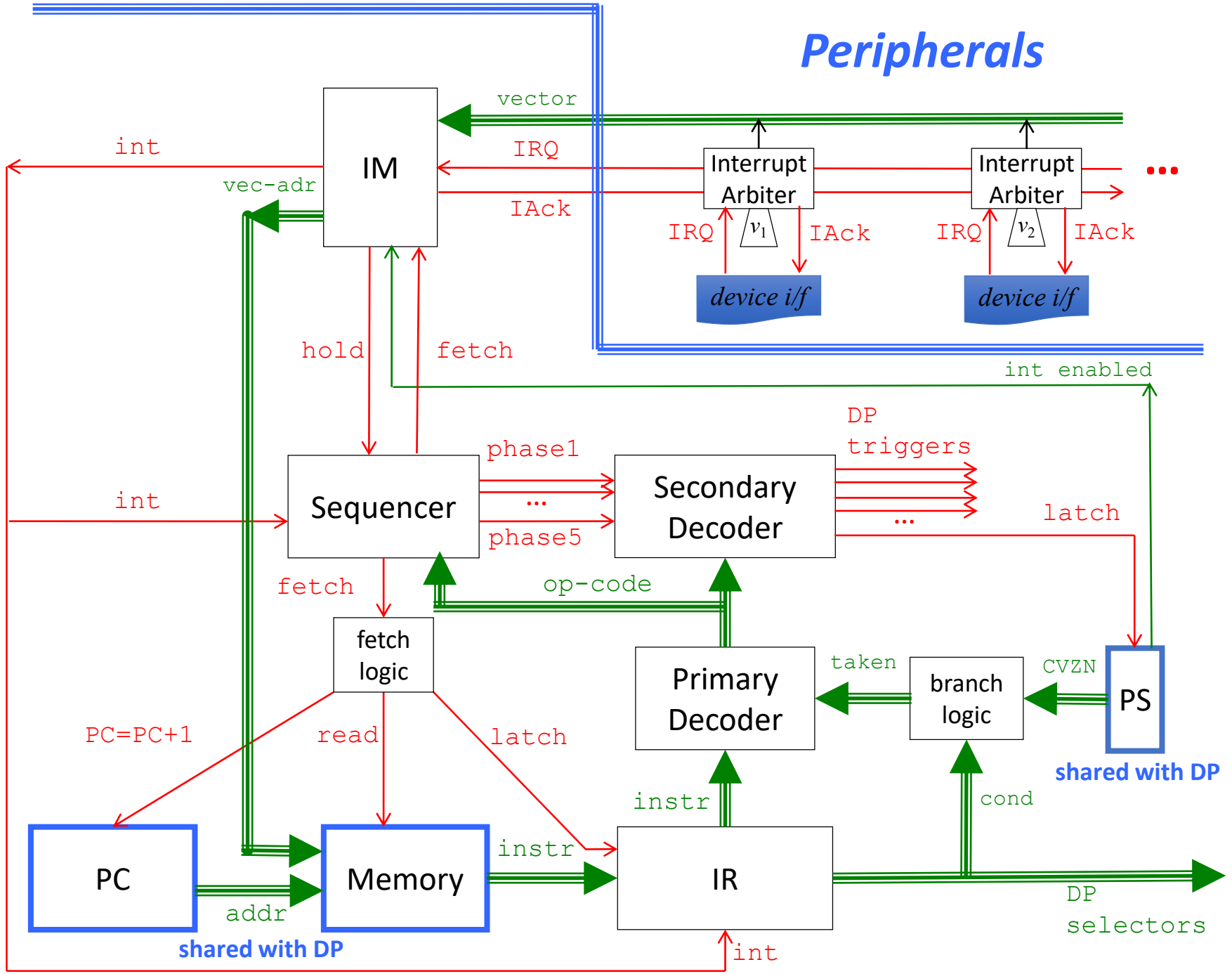
- Прерывания — это аппаратная служба, которая позволяет
  - Сигнализация событий от устройства I/O
  - CPU – Обработка событий
- Требуются изменения во внутренней архитектуре процессора
- CPU должен опрашивать сигнал IRQ в каждом цикле выборки.
- Если сигнал присутствует, вместо выборки следующей инструкции CPU запускает последовательность обработки прерывания.
- Для обработки прерывания CPU сохраняет часть своего состояния (регистры PC и PS) и вызывает обработчик (процедура обслуживания прерывания, ISR).

Changes to CdM-8 CPU

- **Interrupt Master (IM)** , который распознает запросы на прерывание, решает, должно ли прерывание быть предоставлено, и извлекает вектор прерывания из устройства
- **Interruptable Sequencer** фиксирующий команду виртуального прерывания (VII) вместо обычной выборки на основе PC.
- **Extension of the Secondary Decoder** to accommodate the execution of the VII, (Расширение вторичного декодера для обеспечения выполнения VII)
- **Extension of the Instruction Set** with the rti instruction (return from interrupt): the last instruction that an ISR must execute. (Расширение набора команд инструкцией rti (возврат из прерывания): последняя инструкция, которую должен выполнить ISR)
- **Interrupt Arbiter** handles the situation when several devices request interrupts at overlapping periods (Арбитр прерываний обрабатывает ситуацию, когда несколько устройств запрашивают прерывания с перекрывающимися периодами)

Why handle interrupts only on Fetch stage?

- Если мы обрабатываем прерывания только между инструкциями, мы можем сохранять регистры, необходимые для ISR программно (и не сохранять ненужные)
- Если мы обрабатываем прерывания внутри инструкций, мы должны сохранить все состояние процессора, включая регистры, которые недоступны программно (IR, RR, RX, состояние секвенсора)
- В промышленно используемом процессоре это нарушило бы совместимость между процессорами с одинаковой ISA, но разной архитектурой RTL (например, AMD и Intel)


Peripherals
vector
int
IRQ
vec-adr
IM
IAck
Interrupt Arbiter
Interrupt Arbiter
IRQ
v_1
IAck
IRQ
v_2
IAck
device i/f
device i/f
hold
fetch
int enabled
DP triggers
phase1
int
Sequencer
Secondary Decoder
latch
phase5
fetch
op-code
fetch logic
Primary Decoder
taken
branch logic
CVZN
PS
shared with DP
PC=PC+1
read
latch
instr
cond
PC
Memory
instr
IR
DP selectors
addr
shared with DP
int

Interrupts from software point of view

- Каждое устройство, поддерживающее прерывание, имеет уникальный номер в диапазоне от 0 до 7
- Каждое возможное значение номеров устройств выбирает пару байтов, называемую вектором прерывания
- По умолчанию векторы прерываний отображаются в верхние 16 байт памяти
- В манчестерской архитектуре это те же байты, что используются для ввода-вывода с отображением в память, поэтому вы не можете использовать все 7 прерываний и все 16 адресов регистров
- В архитектуре Гарварда ввод-вывод сопоставляется с памятью данных, а векторы - с памятью программ

But what happens when interrupt occurs?

- Устройство устанавливает запрос IRQ на входную линию CPU
- Когда CPU завершает выполнение каждой инструкции, он опрашивает строку запроса IRQ.
- Если прерывания разрешены (мы обсудим это позже), он получает номер устройства.
- Затем вместо инструкции в mem[PC] выполняется инструкция VII (ioi)
  - В каком-то смысле ioi — «обычная» инструкция: у нее есть код операции, ее можно вставить в машинный код и выполнить как любую другую команду.
  - Это называется «программное прерывание».
- Но во время прерывания в mem[PC] нет инструкции ioi.
- Но процессор ведет себя так, как будто он получил эту инструкцию

loi instruction

- **Phase 1** decrement SP for stack push
- **Phase 2** store PC on stack; decrement SP for stack push
- **Phase 3** store PS on stack
- **Phase 4** fetch new PC value from vector’s first cell (0xf0 + 2R)
- **Phase 5** fetch new PS value from vector’s second cell (0xf1 + 2R)
- It is similar to **jsr**, but two registers are saved (PC and PS)
- You need to use **rti** instruction to return from **ioi** routine
- И цель вызова зависит от оборудования (номер устройства R)
- Таким образом, вы можете написать конкретную процедуру обработчика для каждого устройства.

What you can do in interrupt handler?

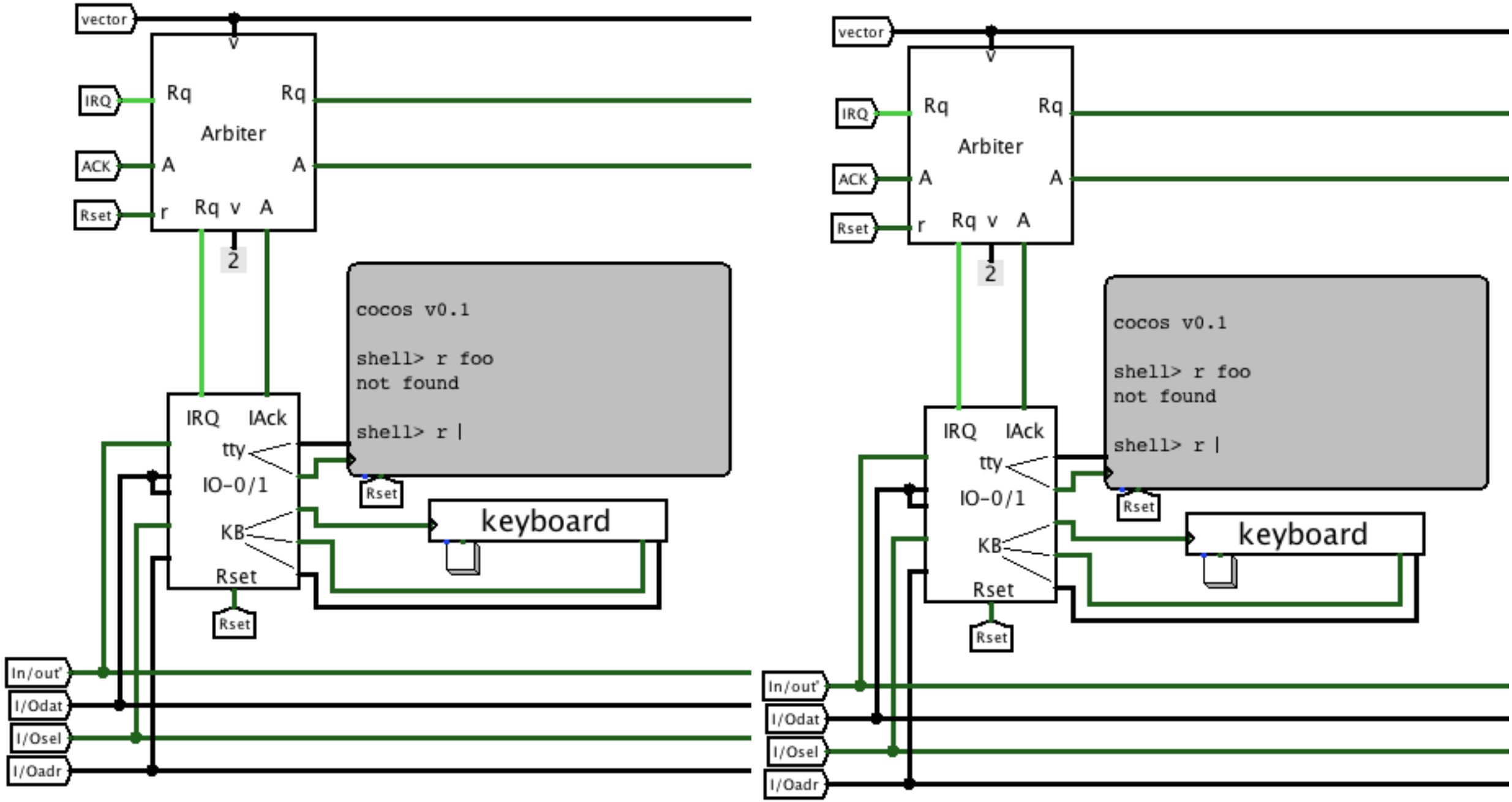
- Как правило, прерывание сигнализирует о том, что у устройства есть какие-то данные для
- Таким образом, вы должны получить данные
- Некоторые устройства требуют дальнейших инструкций, что делать дальше
- Например, когда вы читаете данные с диска, вы должны указать диску, какой сектор читать или писать дальше (или ничего не говорить, и диск будет простаивать)
- Затем вы должны установить некоторые флаги, чтобы основная программа знала, что данные готовы.
- Затем вы должны вернуться в основную программу (выполнить инструкцию rti)
- Или вы можете сделать что-то еще(мы обсудим это в курсе «Операционные системы»)

Why interrupts are bad?

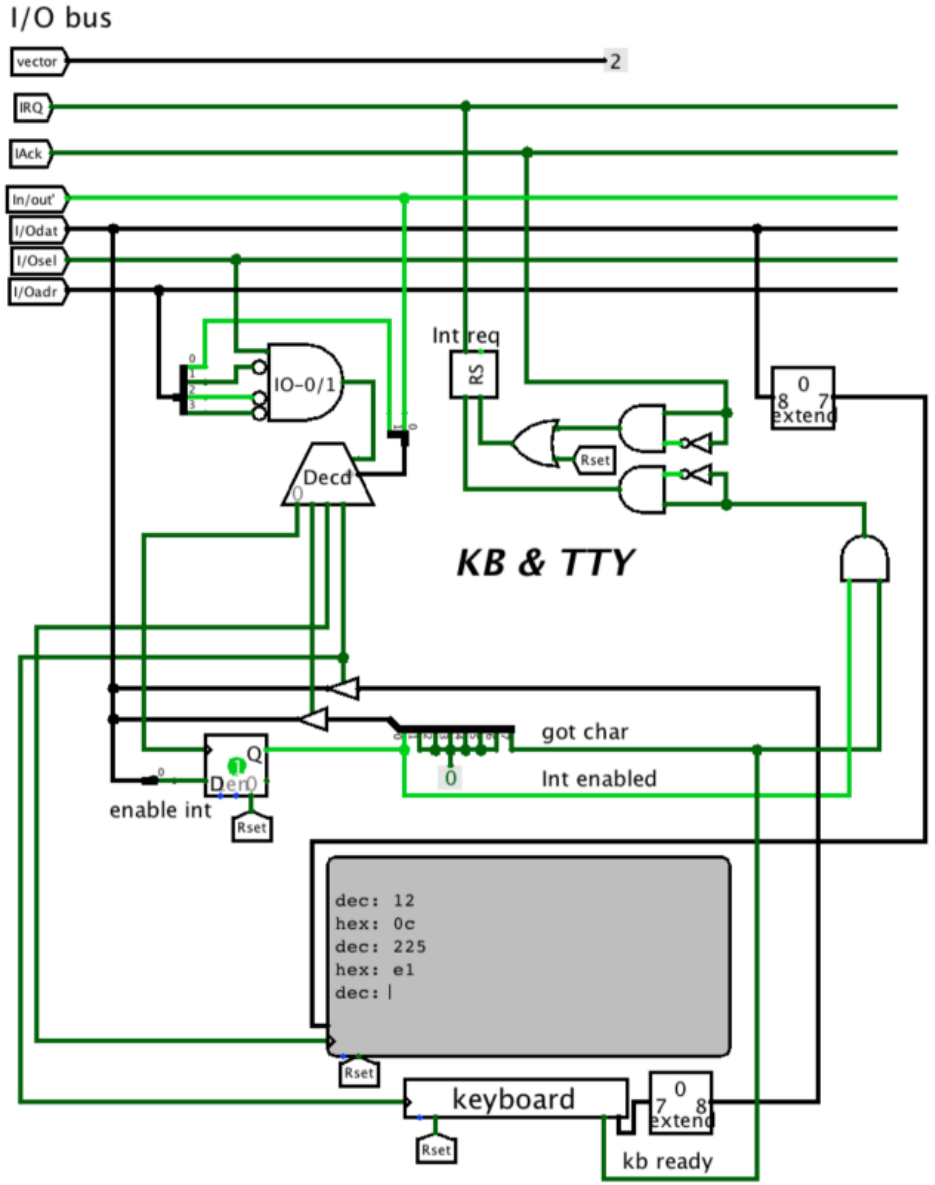
- Асинхронность
- Они могут возникнуть в любой момент выполнения программы
- Очень легко написать обработчик, который может сломать основную программу (повредить ее данные).
- И очень трудно уловить это состояние путем тестирования
- Итак, есть механизм отключения прерываний (флаг в регистре PS)
- Обработка прерываний — простейшая (и исторически первая) форма параллельного программирования, а параллельное программирование имеет много подводных камней.
- большинство из этих ловушек трудно избежать
- В нашей учебной программе будут курсы по параллелизму и параллельному программированию.

Why interrupts are good?

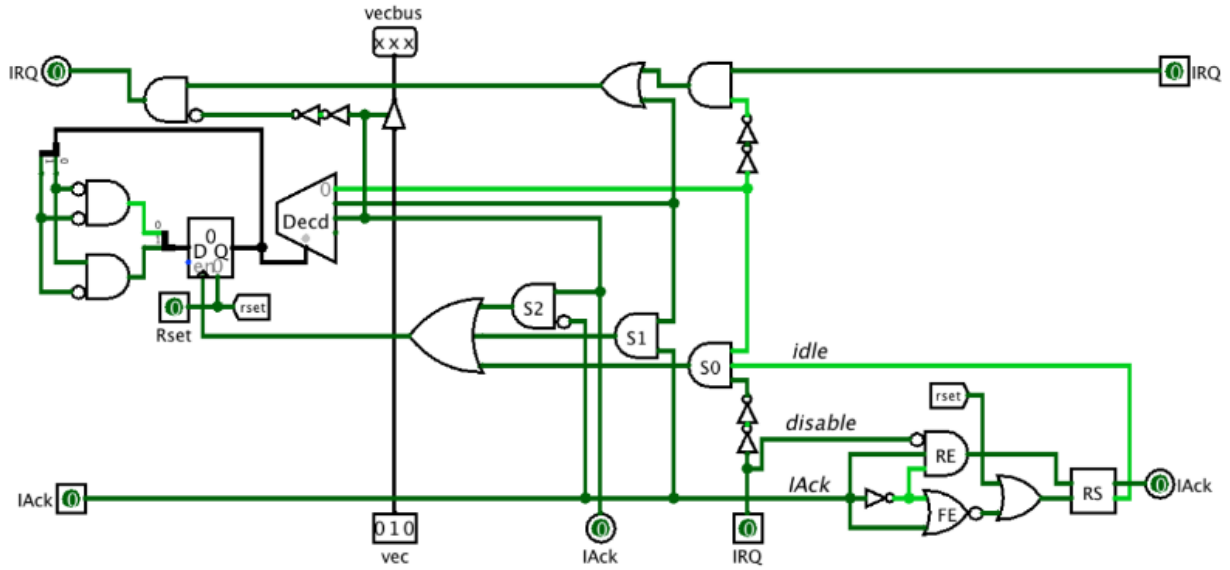
- Вы можете обрабатывать несколько источников событий одновременно
- Вам не нужно переписывать свою программу, чтобы добавить другой источник событий.
- Вы можете сделать что-то полезное в ожидании события
- Операционные системы используют прерывания для реализации многопоточности и многозадачности.


vector
IRQ
ACK
Rset
Rq
Rq
Arbiter
A
A
r
Rq v A
2
IRQ
IAck
tty
IO-0/1
KB
Rset
Rset
Rset
cocos v0.1
shell> r foo
not found
shell> r |
keyboard
In/out'
I/Odat
I/Osel
I/Oadr

мастер прерываний



Interrupt Arbiter



Terminal device driver

ISR читает символ с клавиатуры
И вставляет его в кольцевой буфер
Основная программа может читать данные из буфера, когда он не занят другими задачами
Выходные данные немедленно выводятся на экран
Код находится в томе.pdf