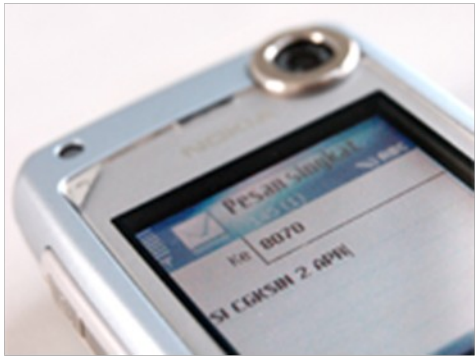




Computer Architectures

1. A card and code based door-lock

Horváth Gábor
associate professor
BUTE Department of Networked Systems and Services
ghorvath@hit.bme.hu

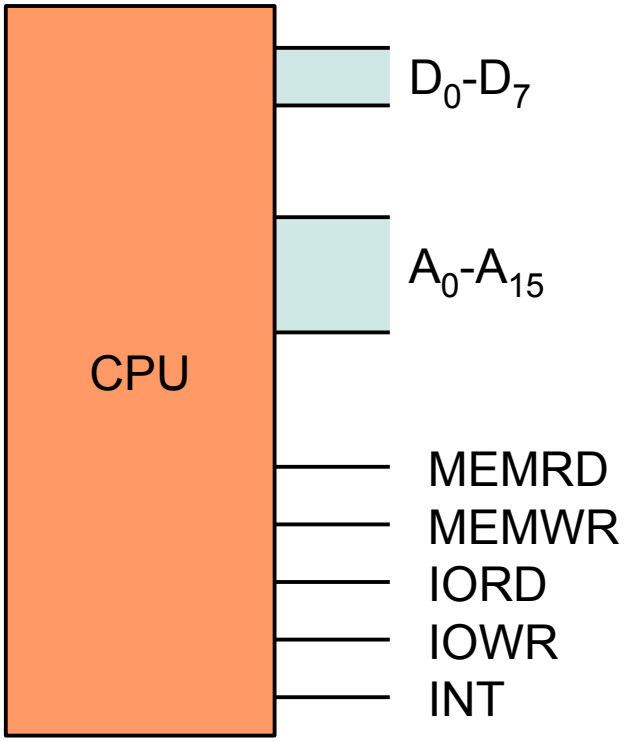
2017. február 16.
Budapest

Outline

- Designing the hardware
  - The building blocks of the system
  - Adding memory to the CPU
  - Adding peripherals to the CPU
- Desigining the software
  - State-machine model of the system
  - Flow-charts of the algorithms
  - Implementation

THE HARDWARE

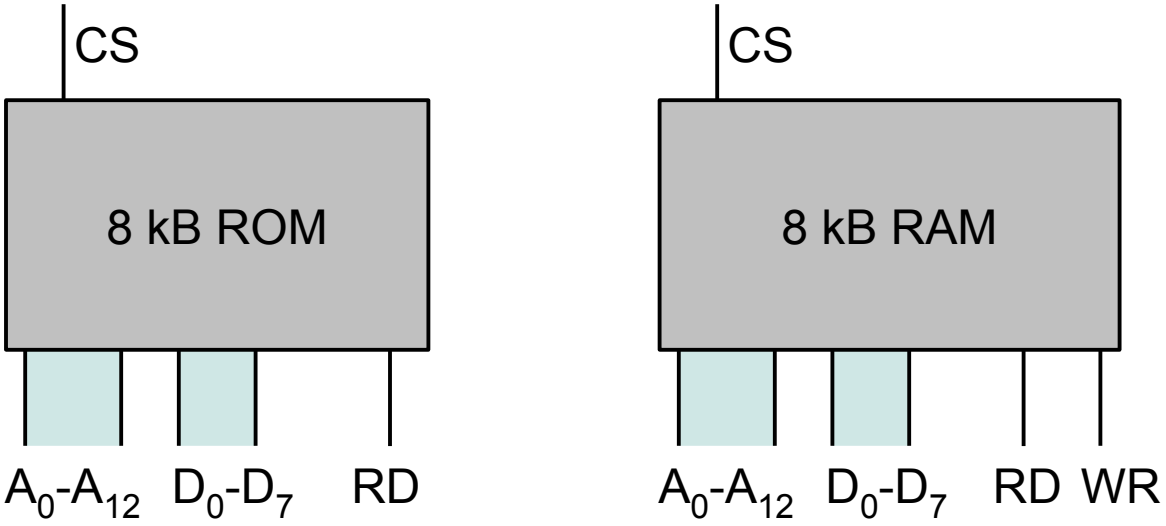
Building blocks of the system



- A 8-bit CPU
  - Data bus: 8 bit wide
  - Address bus: 16 bit wide
  - Multiplexed memory and I/O busses
  - RISC instruction set
  - 3-operand instruction format
  - Starting address: **`0000h`**
  - Interrupt subroutine address: **`1000h`**

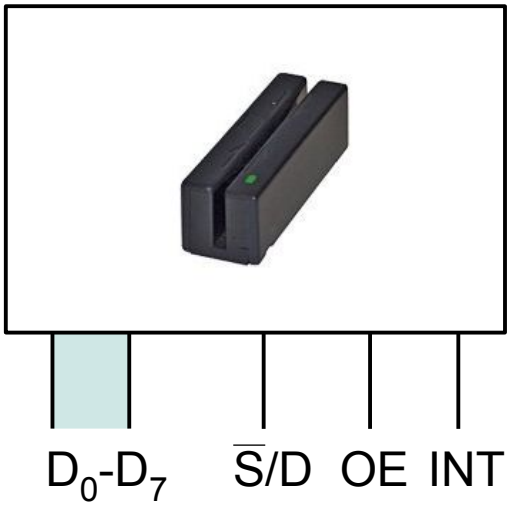
Building blocks of the system

- Memory
  - ROM: to store **instructions** and program constants
  - RAM: to store **program variables** and **stack**
- Let us use 8kB ROM and 8kB RAM
  - 8 bit data bus
  - 13 bit address bus



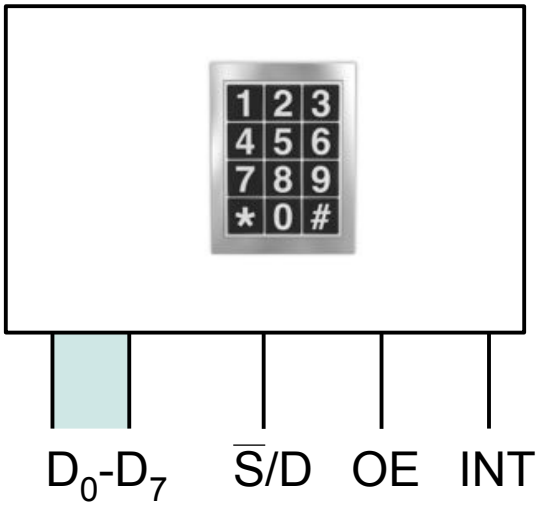
Building blocks of the system

- Card reader
  - OE: output enable
  - It has a 8 bit output D:
    - If $\overline{S}$/D = 0: it gives back an 1 if a new card has arrived
    - If $\overline{S}$/D = 1: it gives back the code (ID) of the last card
  - INT: generates an interrupt if a new card is recognized



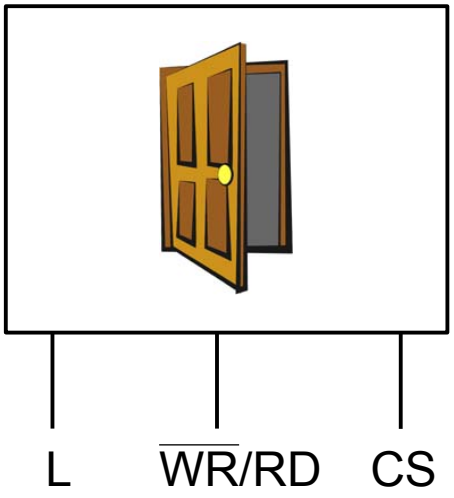
Building blocks of the system

- Keypad
  - OE: output enable
  - It has a 8 bit output D:
    - If $\overline{S}$/D = 0: it gives back an 1 if a button has been pressed
    - If $\overline{S}$/D = 1: it gives back the code of the last key
  - INT: generates an interrupt upon each key press



Building blocks of the system

- Door lock
  - CS: chip select
  - $\overline{WR}$/RD
  - If $\overline{WR}$/RD = 0: sets L according to the state of the door
  - If $\overline{WR}$/RD = 1: opens the door. The door locks again after a timeout automatically.

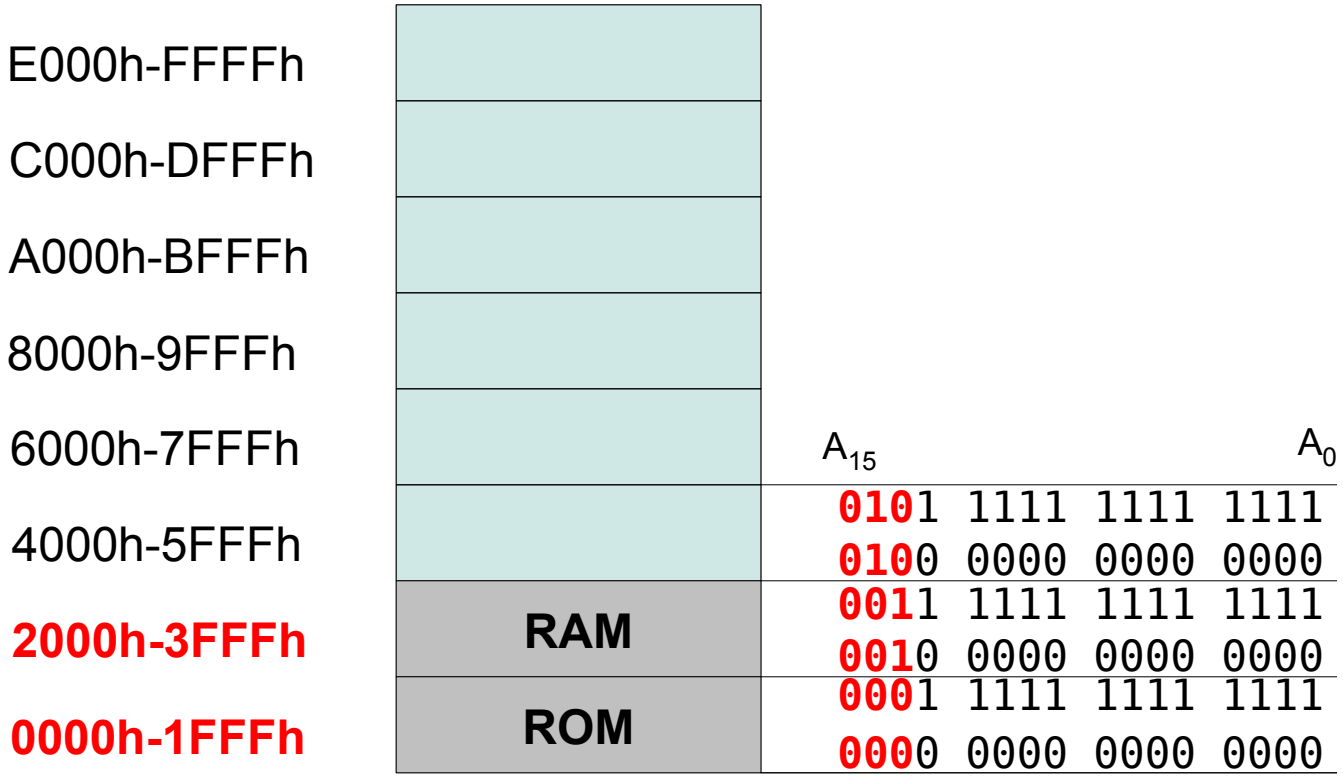


Building blocks of the system

- LED lights
  - Indicate the status of the door (red/green)
- Additional elements:
  - Decoder – for interfacing the memory
  - Comparators – for detecting the addresses of peripherals
  - D flip-flop – for driving the LED lights

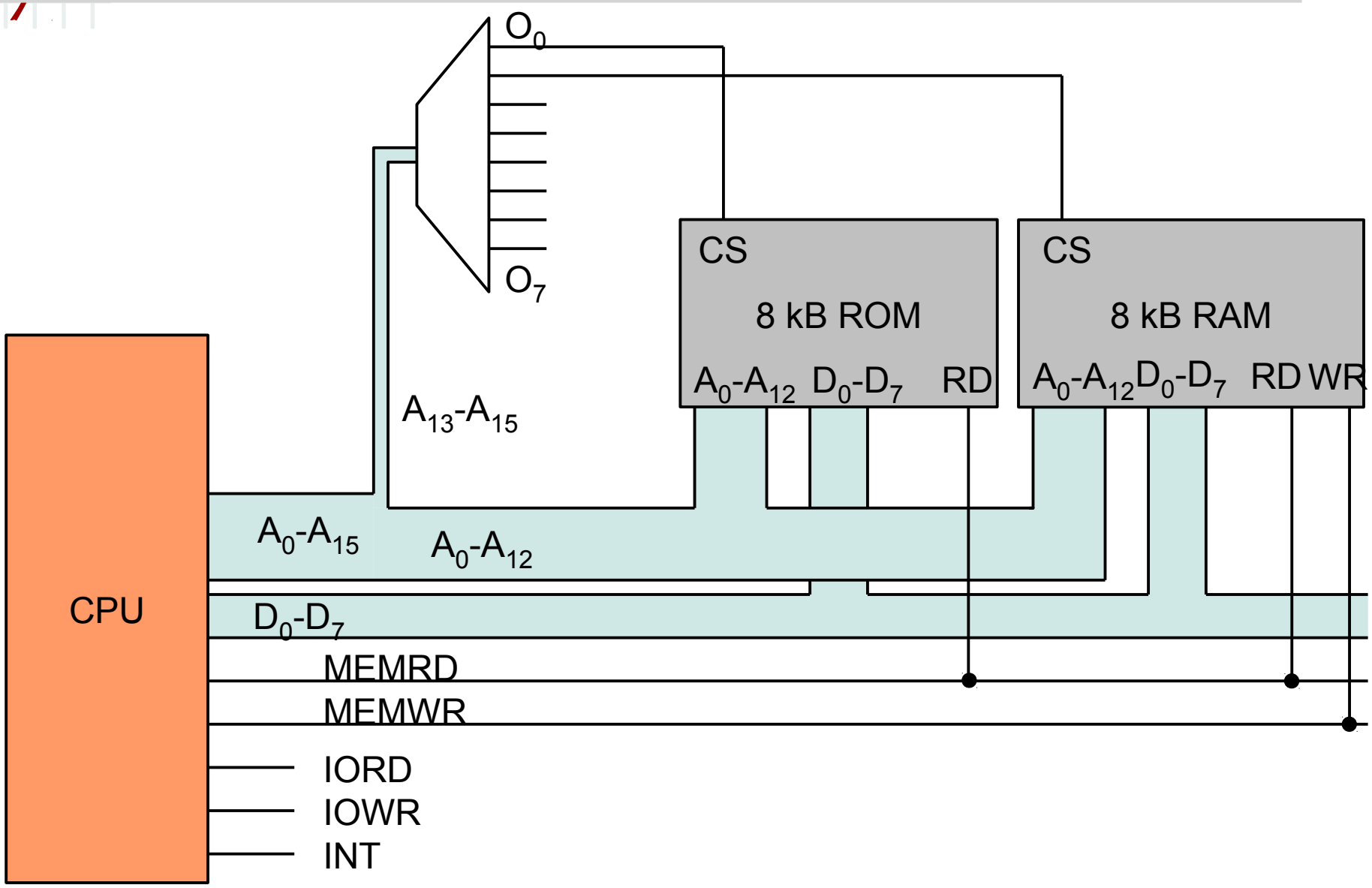
Adding memory to the CPU

- We have 8 kB ROM + 8 kB RAM
- Memory map:

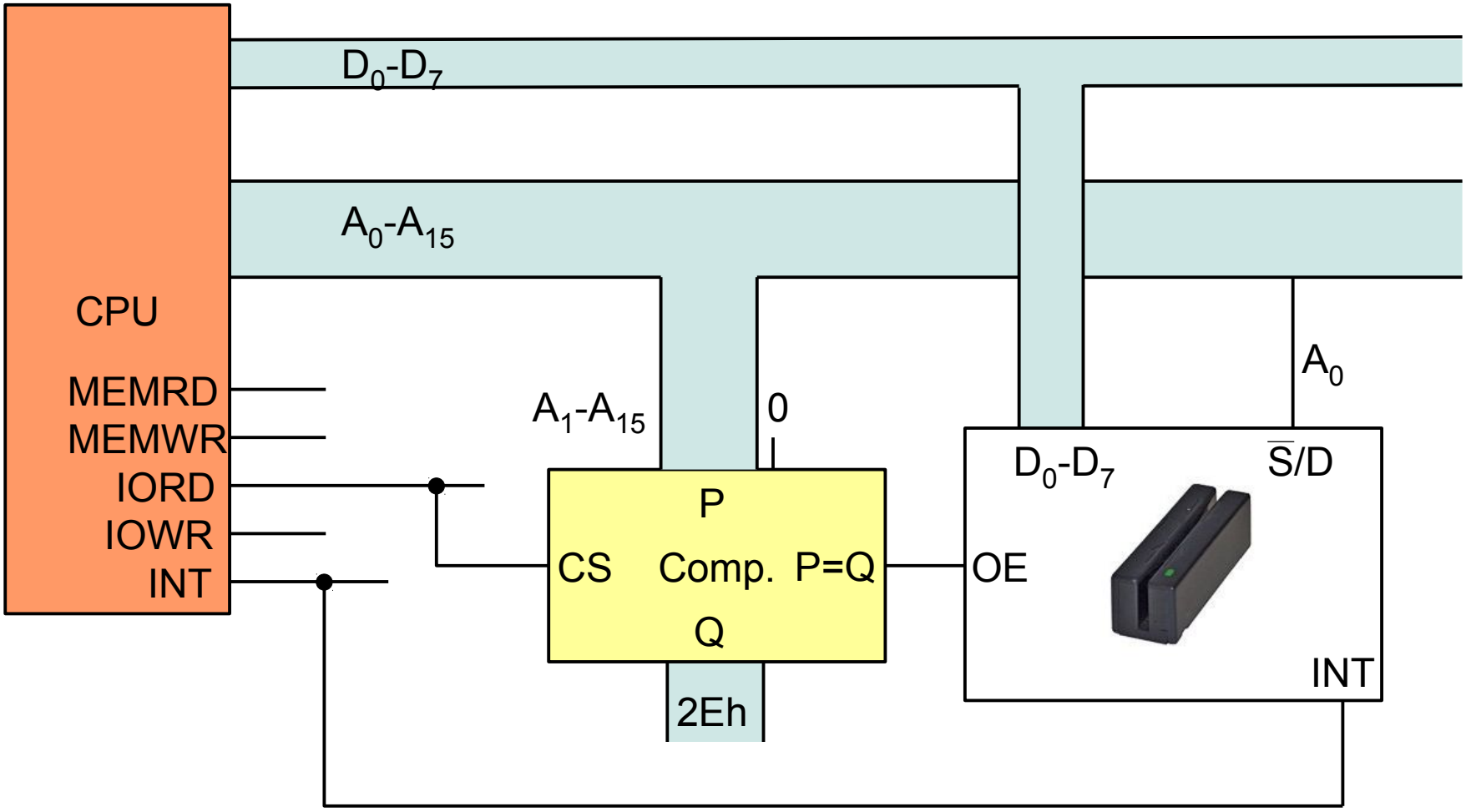


- A_{13}-A_{15} determine which module to use
- A_0-A_{12} determine the byte position in the selected module

Adding memory to the CPU

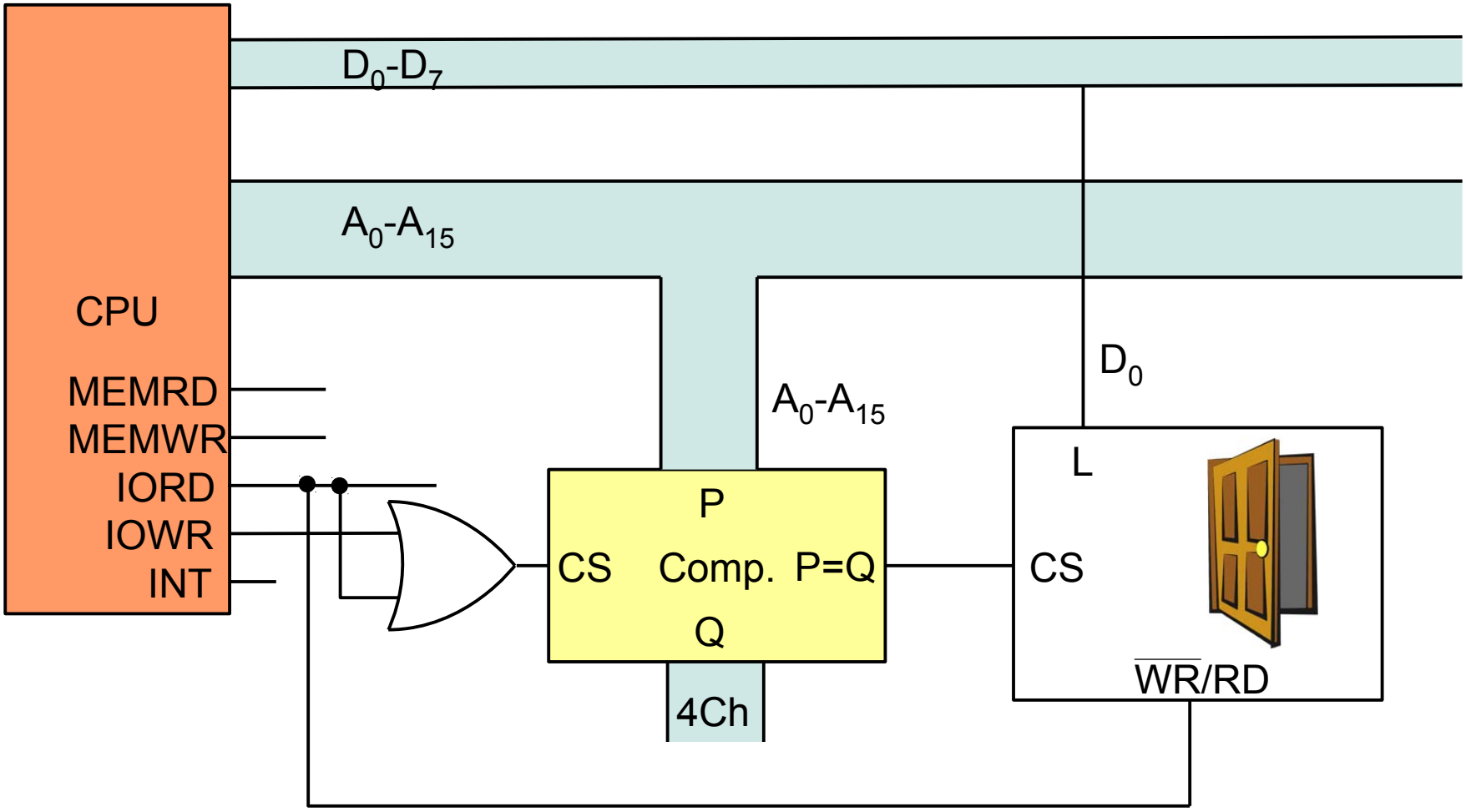


Adding the card reader

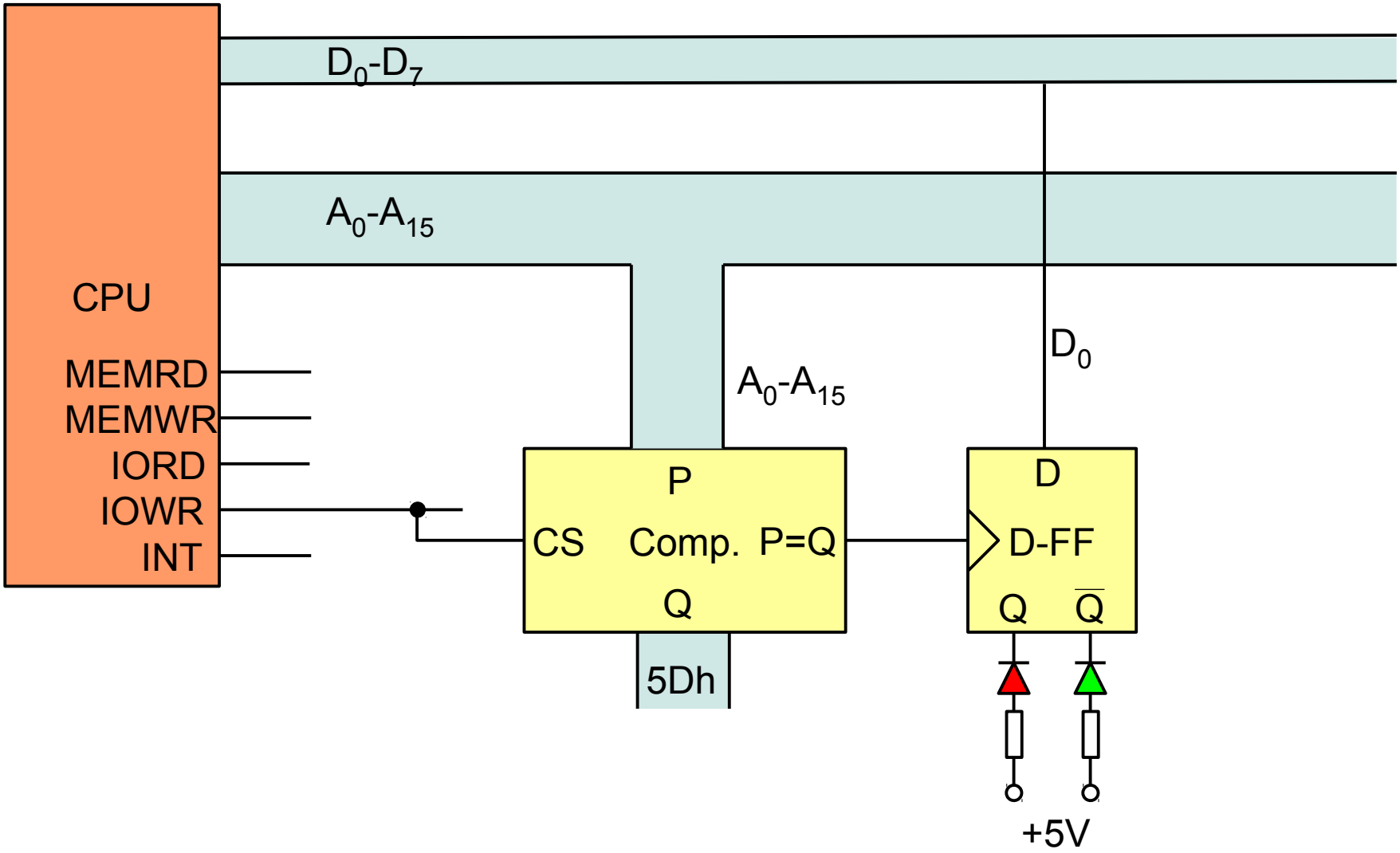


- Adding the keypad: the same, with base address 3Eh

Adding the door lock



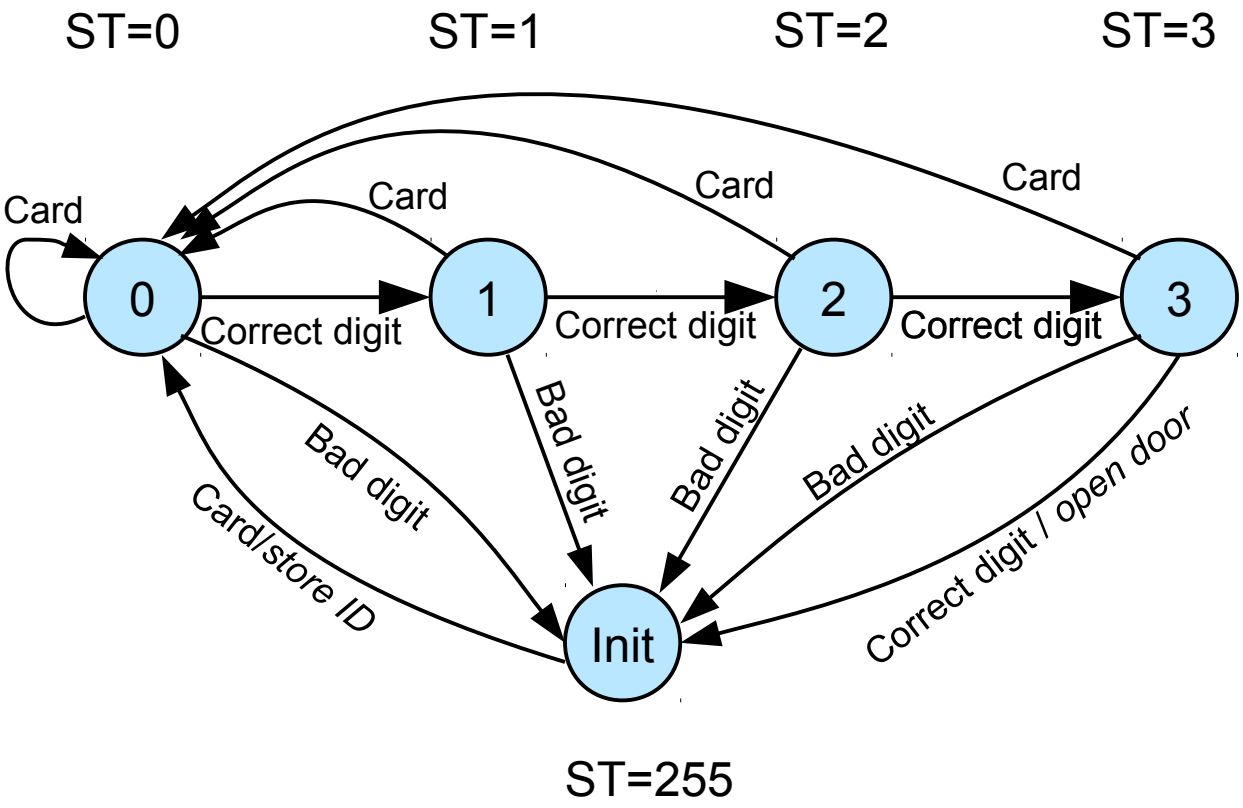
Adding the LED lights



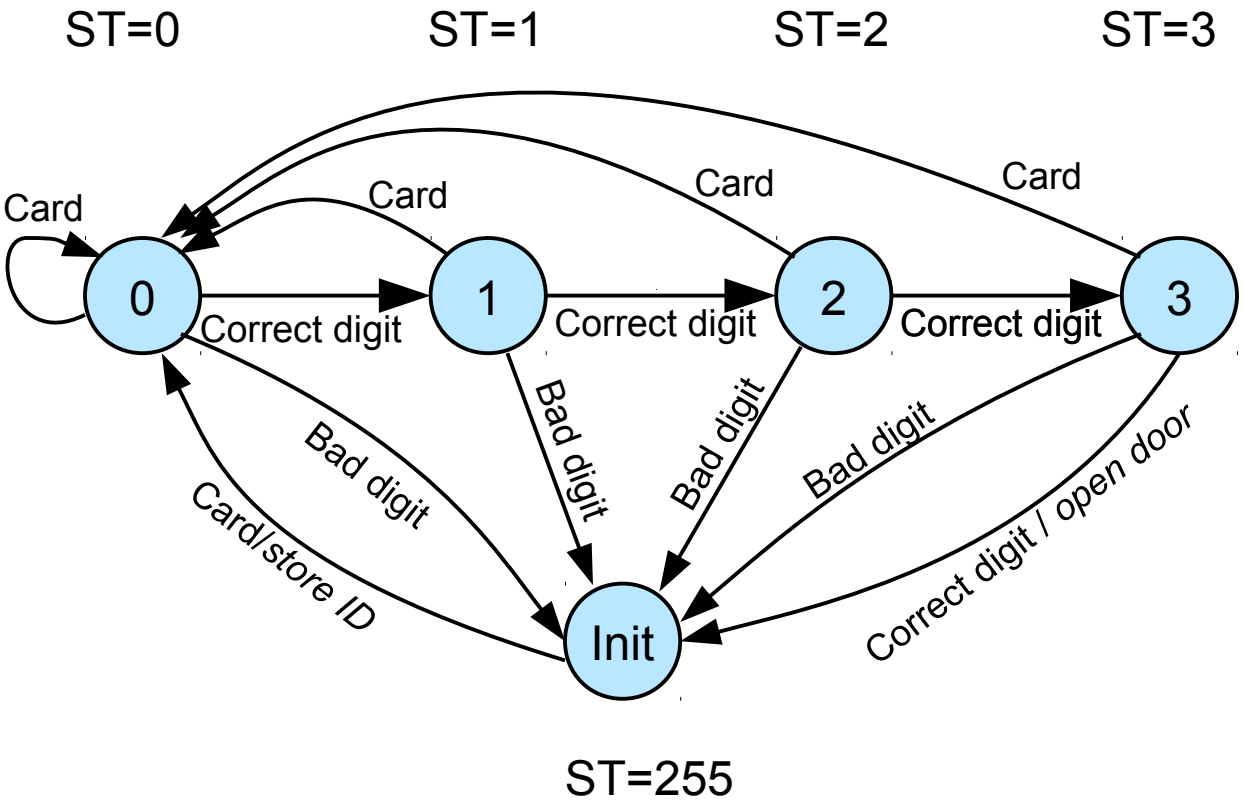
THE SOFTWARE

State-machine of the system

- ST (state) = number of correct code digits typed so far on the keypad



Transitions of the state-machine



- Card event: ST=0, and store the card ID in a local variable
- Correct digit: ST=ST+1, if ST is not 255
- Bad digit: ST=255

Data structures

- The array of 4-digit key codes for each card ID
  - The card ID is 8 bit wide → 256 different IDs are allowed
  - An array is used
    - with 256 entries, one for each card ID
    - each entry is the correct code (4-byte long) of the given card
  - Total memory consumption: 256*4=1024 bytes (=**400h**)
  - The ***i***th code digit of card ***j*** is located at

    array start address + ***j****4 + ***i***

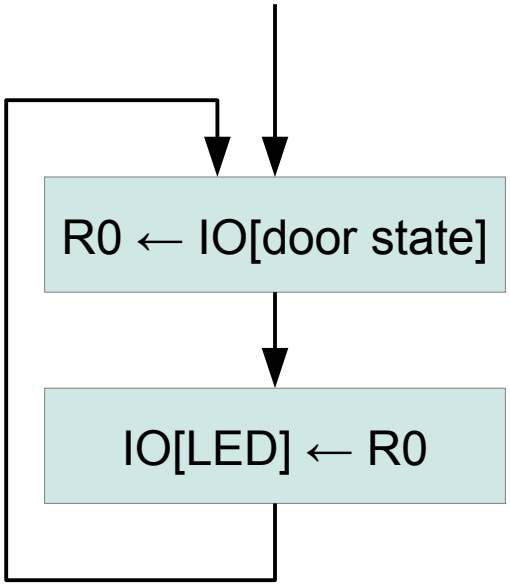
- Local variables:
  - 1-byte integer variable ST
  - 1-byte integer variable CARDID

Algorithms

- The program has two parts:
  - Main program
  - Interrupt service routine
- The purpose of the main program:
  - Monitors the state of the door
  - Adjusts the LEDs accordingly
- The purpose of the interrupt service routine
  - Handles card events and key presses
  - Opens the door, if the correct code is given

The main program

- In an infinite loop:
  - Ask door for the state
  - Set LEDs accordingly

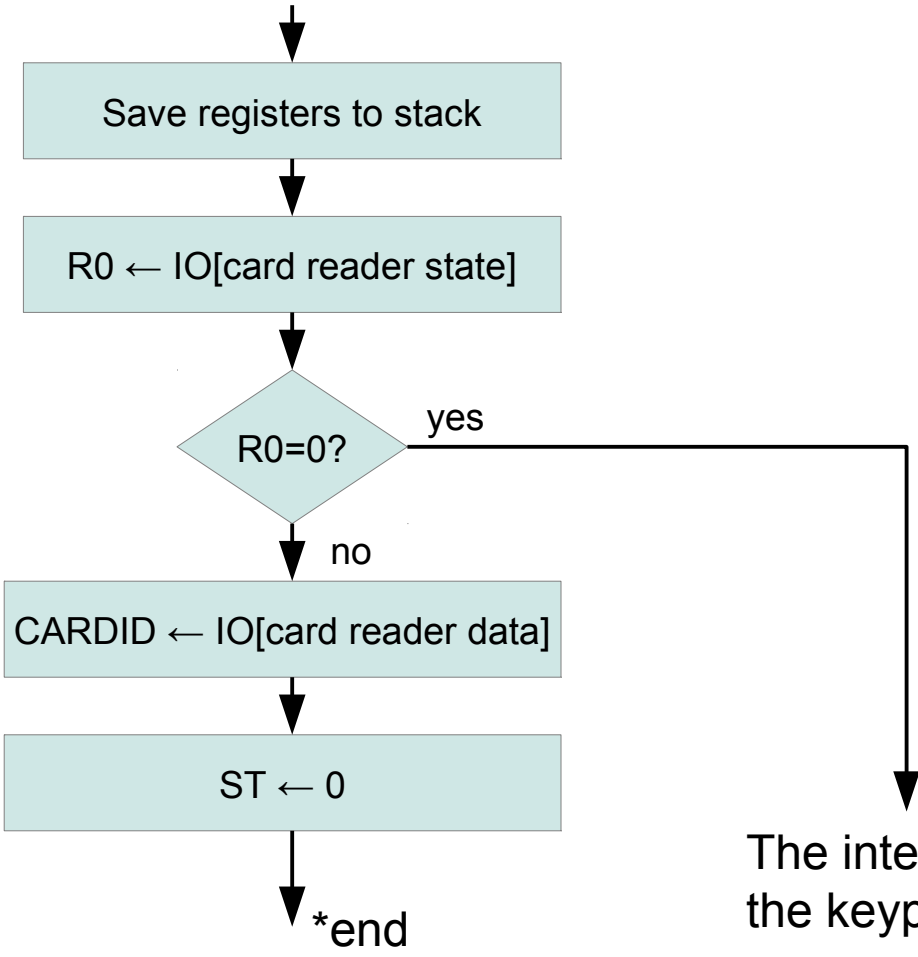


The interrupt service routine

- First we have to find out the interrupt source (polling!)
- Then apply the transition on the state machine

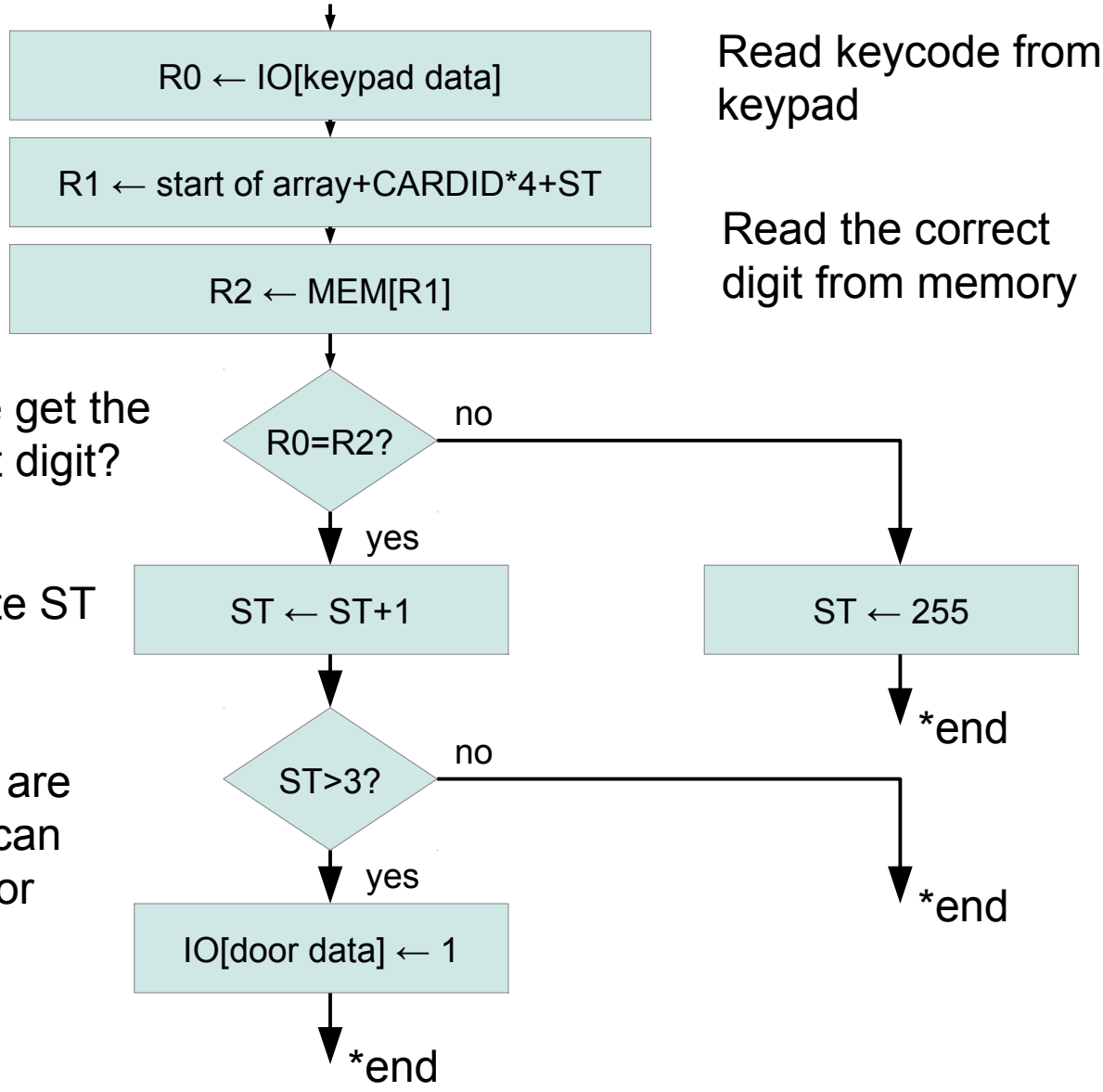
Check if the interrupt arrived from the card reader

Store ID, set the state machine to ST=0 and go to the end of interrupt routine



The interrupt service routine

The interrupt is from the keypad...



Read keycode from keypad

Read the correct digit from memory

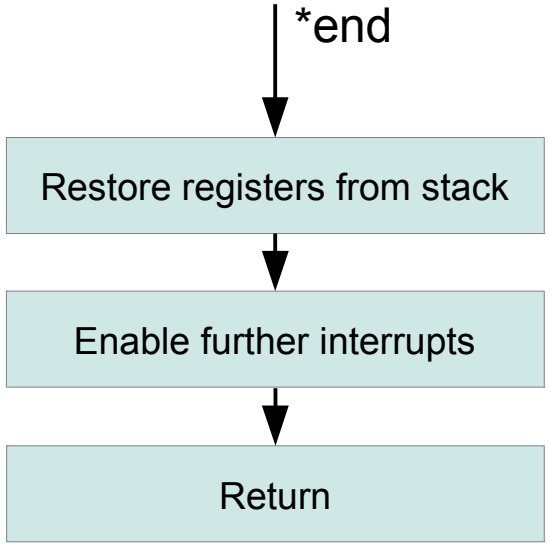
Did we get the correct digit?

If yes, update ST

If all 4 digits are correct, we can open the door

The interrupt service routine

The end of interrupt service routine:



Placement decisions

- **Content of the RAM**
  - 1-byte integer variable ST: **2000h** (first byte of the RAM)
  - 1-byte integer variable CARDID: **2001h** (second byte of the RAM)
  - Initial stack pointer: **3FFFh**, since it grows downwards
- **Content of the ROM**
  - The program, starting at **0000h**
  - The interrupt service routine, starting at **1000h**
  - The array of correct codes for each card
    - Size: **400h**
    - Can be placed anywhere, where space is available
    - Let us put it to **500h**
      - The main program is small, it does not go beyond **500h**
      - The end of the array is **900h**, which is bellow the interrupt routine

Assembly implementation

- Implementation of the main program:

```
        ORG 0000h          start code at this address
        SP ← 3FFFh         set stack pointer
        EI                 enable interrupts
label:  R0 ← IO[4Ch]       read door status
        IO[5Dh] ← R0       update LEDs
        JUMP label         jump back and do it again
```

Assembly implementation

- Array of 4-byte codes for each card

```
ORG 500h
codes: DB 1, 3, 4, 7
       DB 5, 7, 2, 9
       DB 8, 2, 0, 8
       DB 3, 1, 8, 9
       ...
```

Assembly implementation

- Implementation of the interrupt service routine:

```
ORG 1000h                    CPU jumps at this address on interrupt
PUSH R0                      save R0/R1/R2 register to stack
PUSH R1
PUSH R2
R0 ← IO[2Eh]                 read card reader state
JUMP keycode IF R0==0        jump if interrupt source is the keypad
R0 ← IO[2Fh]                 read card ID
MEM[2001h] ← R0              save to variable CARDID
MEM[2000h] ← 0               ST=0
JUMP end                     jump to the end of interrupt routine
```

Assembly implementation

- Implementation of the interrupt service routine:

```
keycode: R0 ← IO[3Fh]          R0=new digit
         R1 ← MEM[2000h]       R1=ST
         R2 ← MEM[2001h]       R2=CARDID
         R2 ← R2 * 4           R2=4*CARDID
         R2 ← R1 + R2          R2=4*CARDID+ST
         R2 ← MEM[R2+codes]    R2=the correct digit
         JUMP match IF R2==R0  Jump if new digit is correct
         MEM[2000h] ← 255      If not correct, go the init state
         JUMP end              ...and go to the end of interrupt
match:   R1 ← R1+1             If correct, increment ST
         MEM[2000h] ← R1       ... and save ST to memory
         JUMP end IF R1<4      If this is not the last digit, go to the end
         IO[4Ch] ← 1           If last digit, open the door
         JUMP end              Go to the end of interrupt
```

Assembly implementation

- Implementation of the interrupt service routine:

```
end: POP R2          restore registers in a reverse order
     POP R1
     POP R0
     EI              enable further interrupts
     RET             return from interrupt
```

Variations

- Can we solve the problem without RAM?
  - Yes.
  - But we lose the stack
    - No more interrupts!!!
    - Main program continuously polls:
      - The state of the door (to update LEDs)
      - The state of the card reader
      - The state of the keypad
  - We can not put ST and CARDID into the RAM
    - We can store them in registers permanently