Computer Architectures

1. A card and code based door-lock

Prepared by: **Gábor Horváth**, ghorvath@hit.bme.hu
Presented by: **Gábor Lencse**, lencse@hit.bme.hu

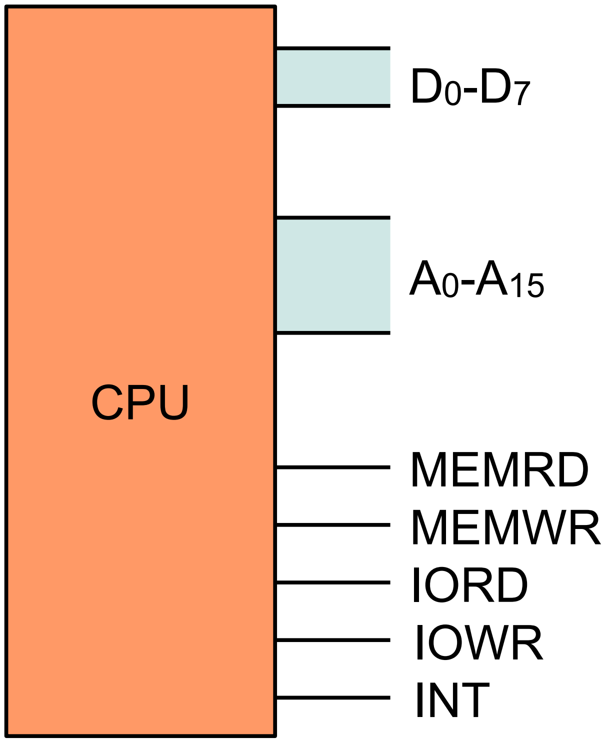
2023. márc. 10.
Budapest

Outline

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

THE HARDWARE

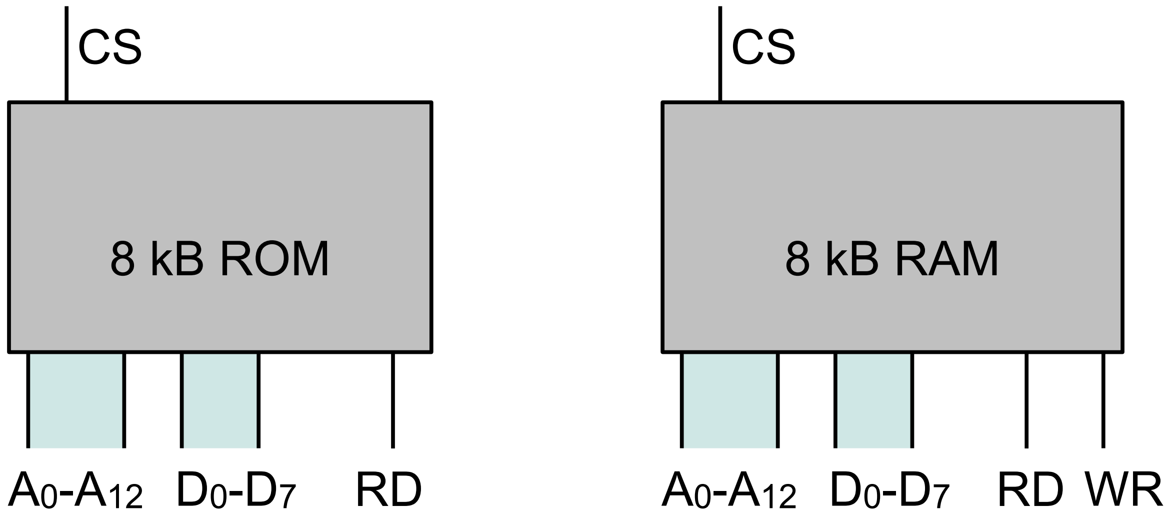
Building blocks of the system



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

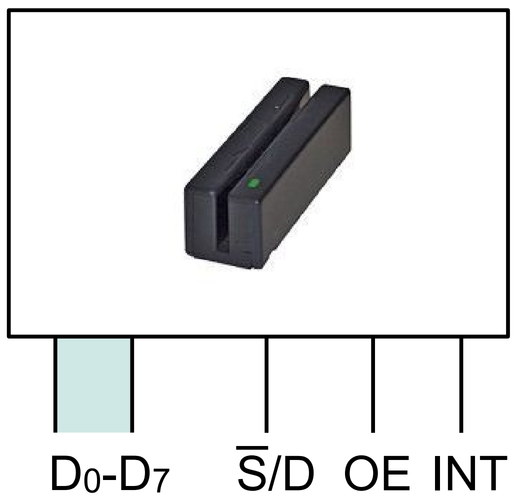
Building blocks of the system

- 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



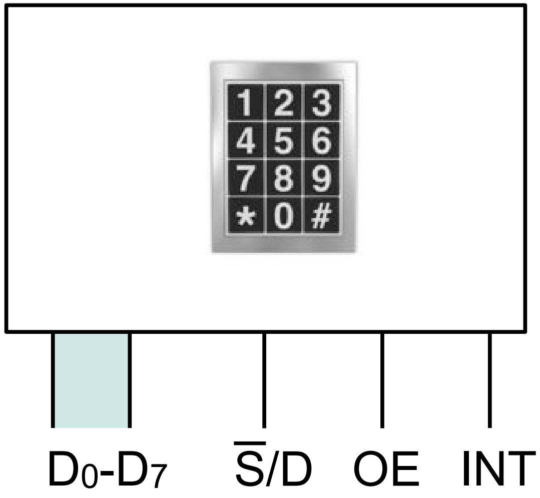
Building blocks of the system

- Card reader
  - OE: output enable
  - It has an 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



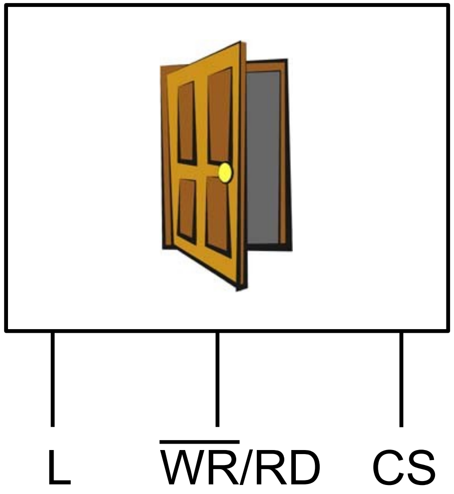
Building blocks of the system

- 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 = 1: sets L according to the state of the door
    - (open=1, locked=0)
  - If $\overline{WR}$/RD = 0: opens the door. The door automatically locks again after a timeout.



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

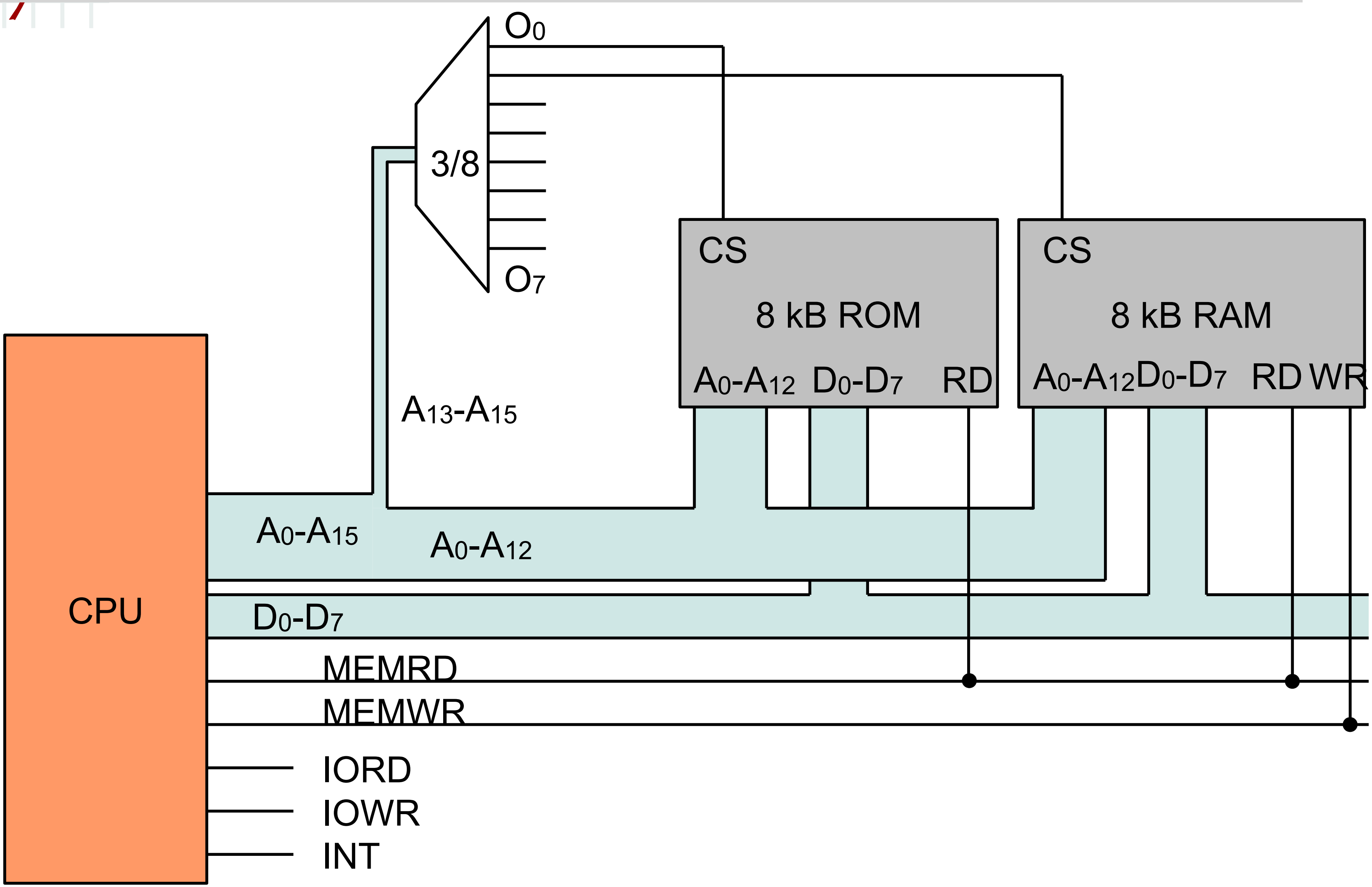
Adding memory to the CPU

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

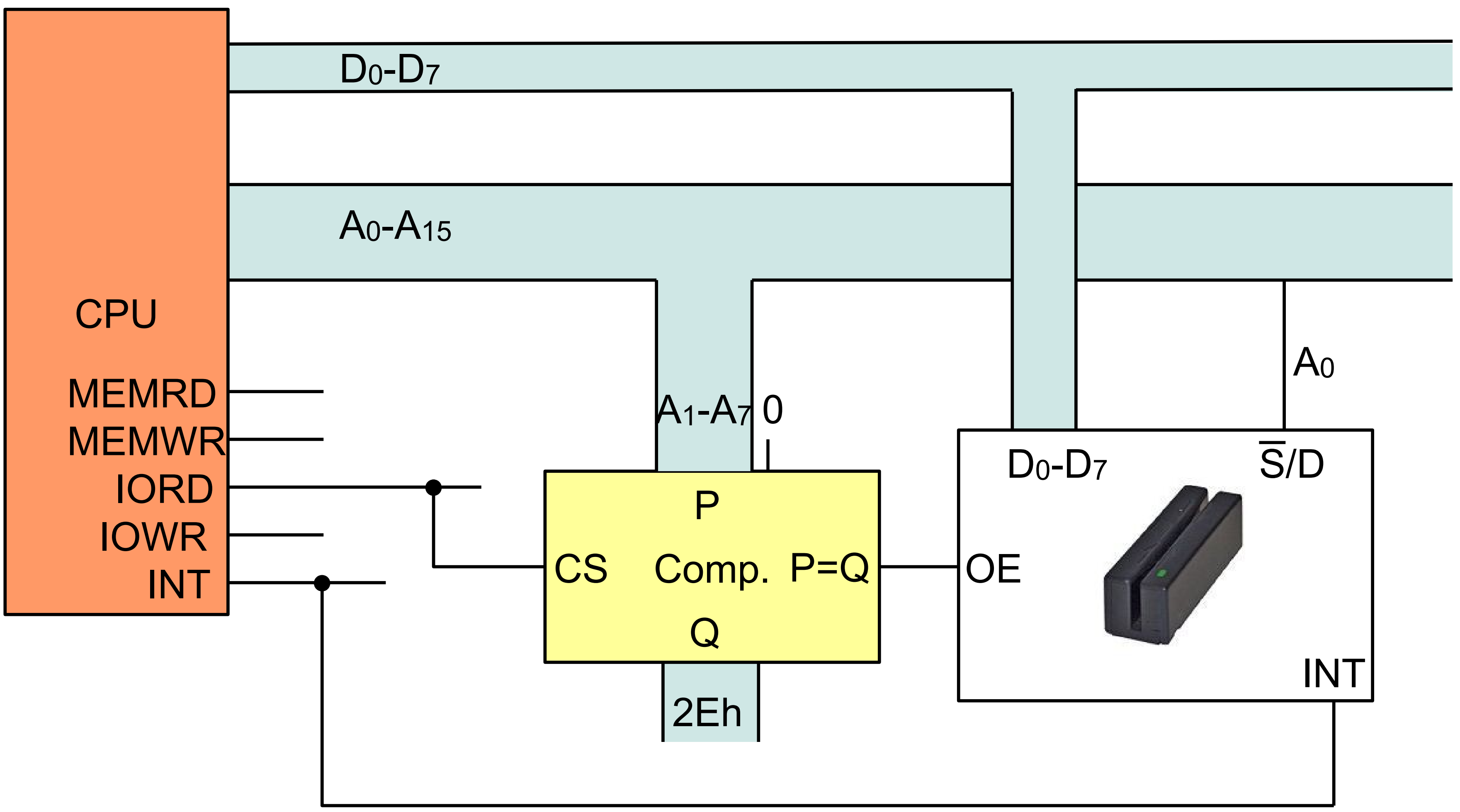
Address range	Module	A_{15} … A_0
E000h-FFFFh		
C000h-DFFFh		
A000h-BFFFh		
8000h-9FFFh		
6000h-7FFFh		
4000h-5FFFh		0101 1111 1111 1111 0100 0000 0000 0000
2000h-3FFFh	**RAM**	0011 1111 1111 1111 0010 0000 0000 0000
0000h-1FFFh	**ROM**	0001 1111 1111 1111 0000 0000 0000 0000

- 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

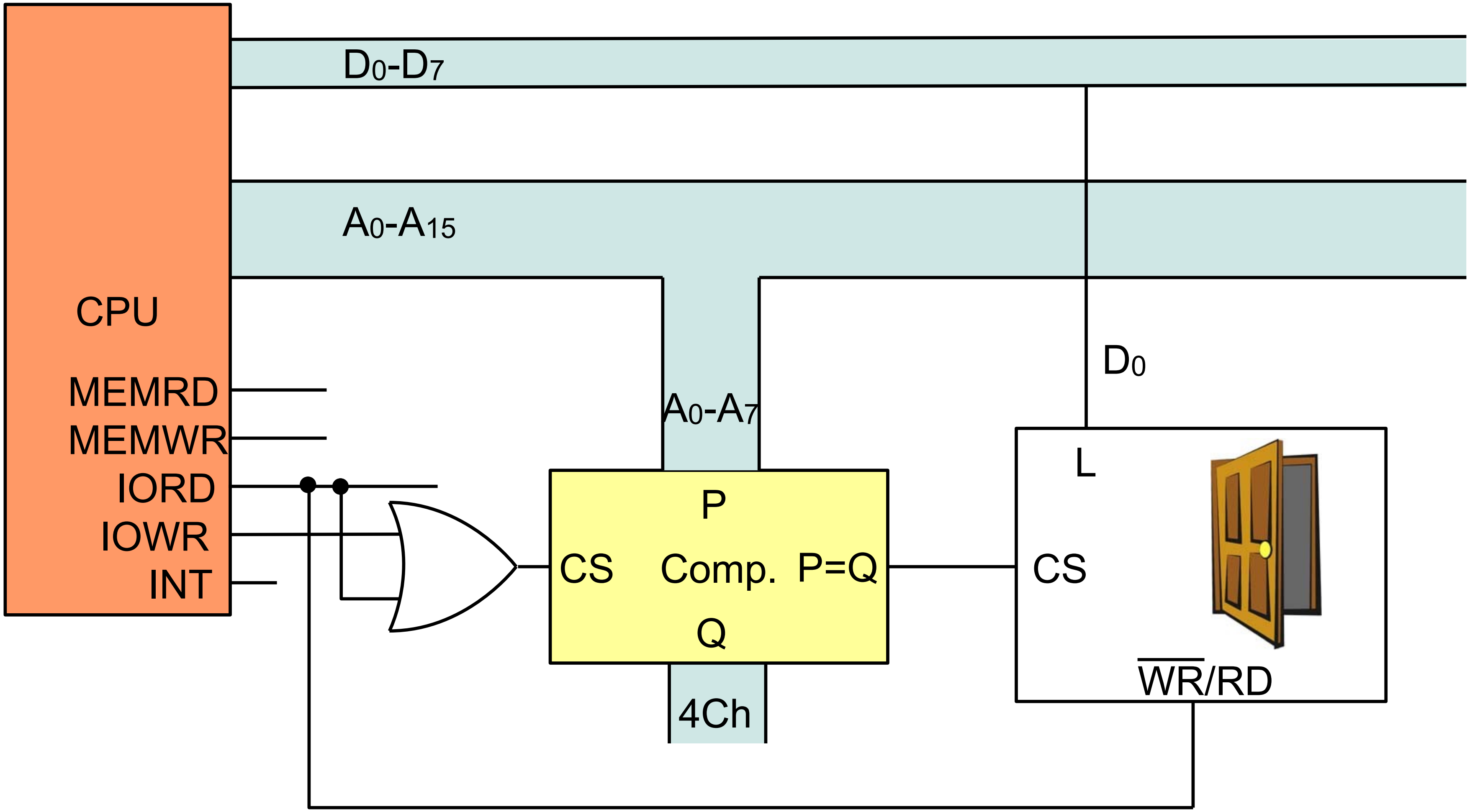


Adding the card reader

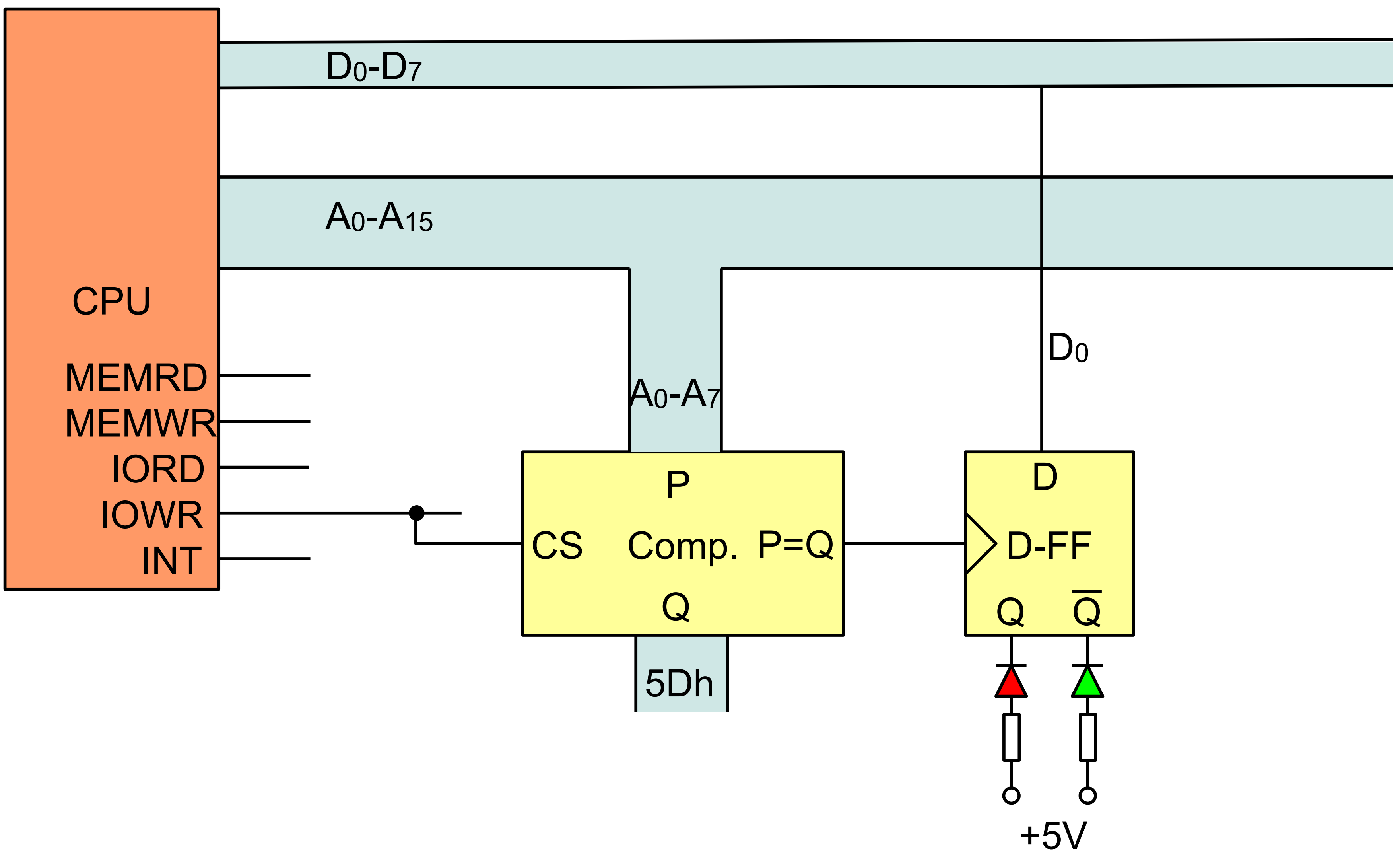


- Adding the keypad: the same, with base address 3Eh

Adding the door lock



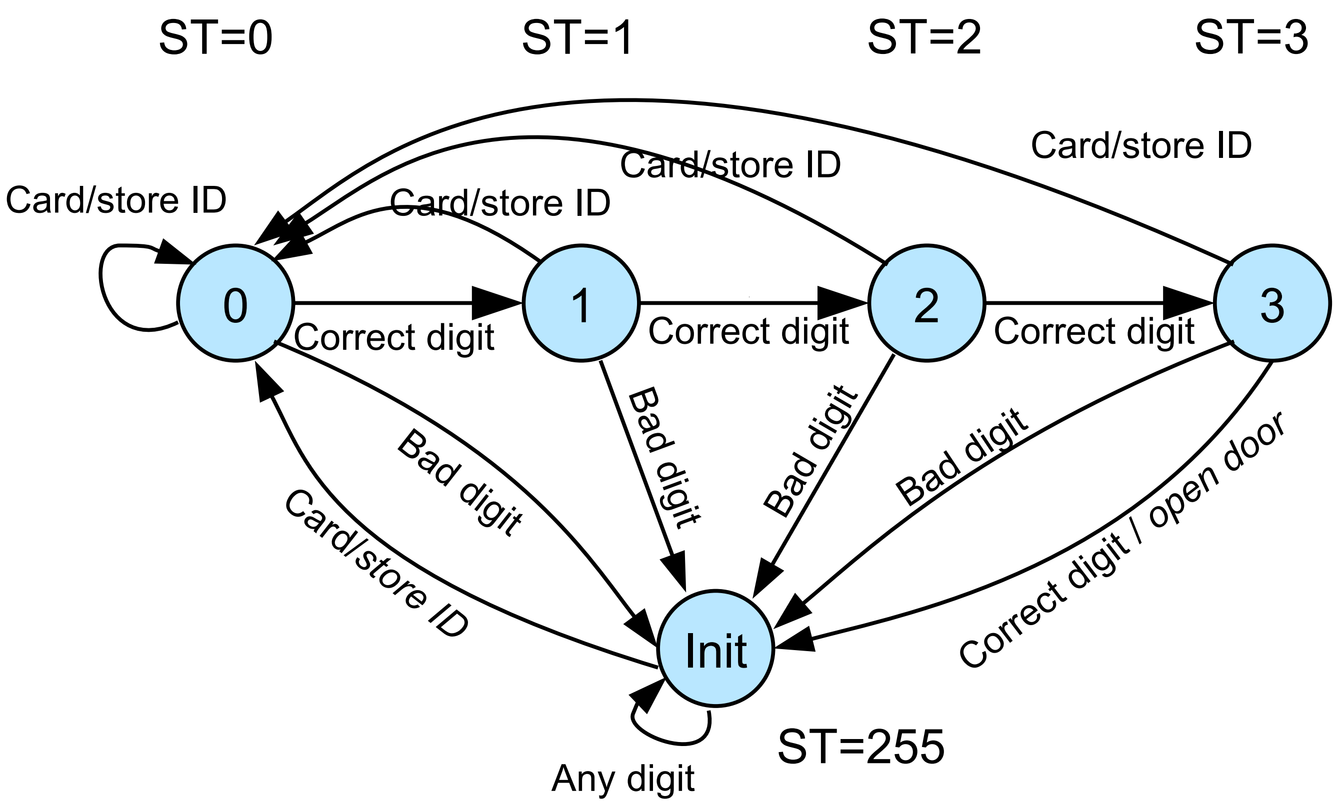
Adding the LED lights



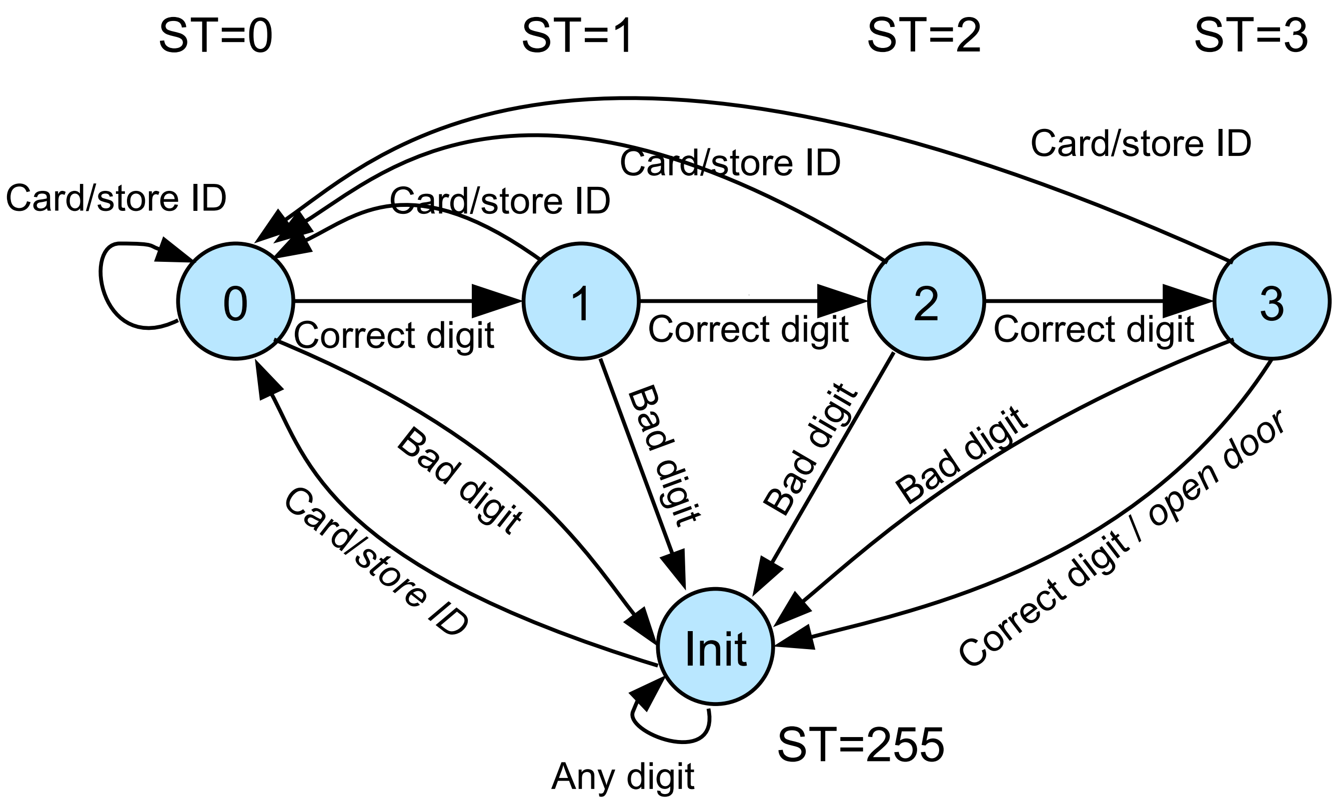
THE SOFTWARE

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: (if ST!=255 then) ST=ST+1,
  - if now ST==4 then: ST=255 and open door
- 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 possible
  - 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***   (*j* in {0..255}, *i* in {0..3})

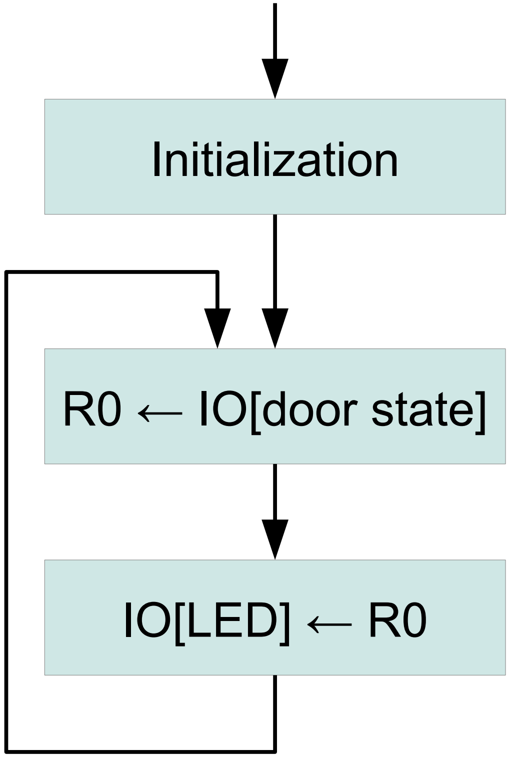
- Local variables:
  - 1-byte integer variable ST
  - 1-byte integer variable CARDID

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

- Initialization:
  - Set stack pointer
  - Set the value of ST (state variable)
  - Enable interrupts
- In an infinite loop:
  - Ask the door for the state
  - Set the LED-s accordingly

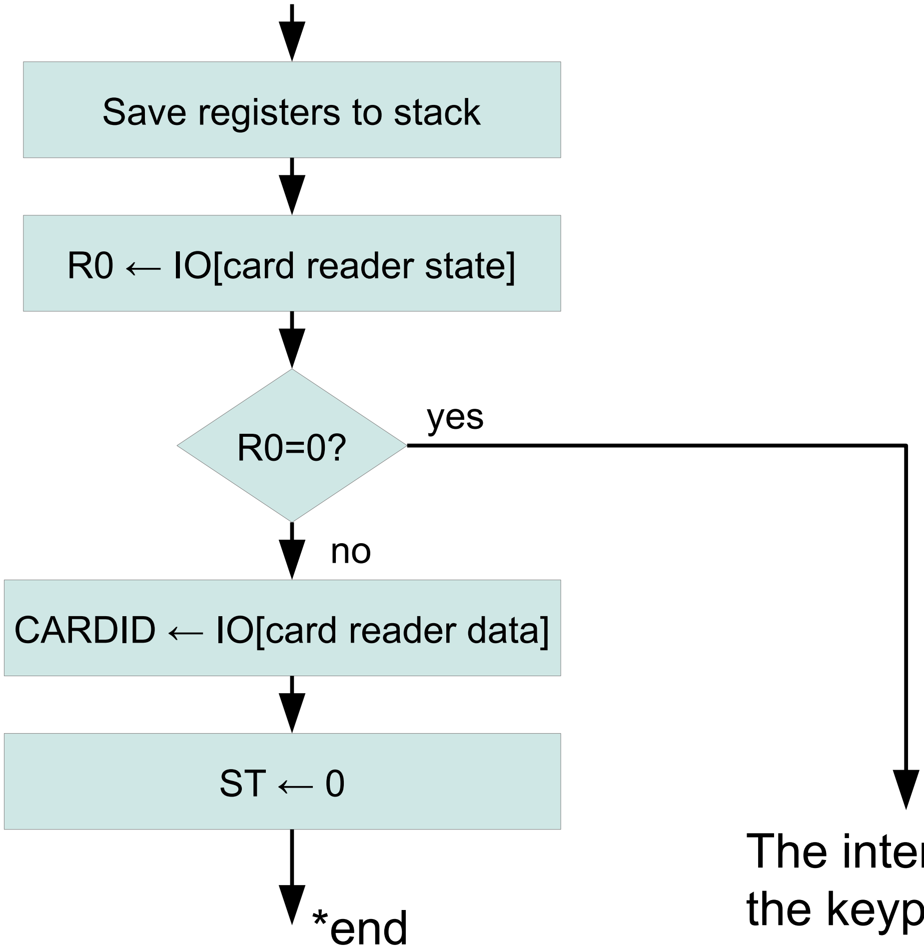


The interrupt service routine

- First, we have to find out the interrupt source (polling!)
- Then, we 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 is from the keypad...

The interrupt service routine

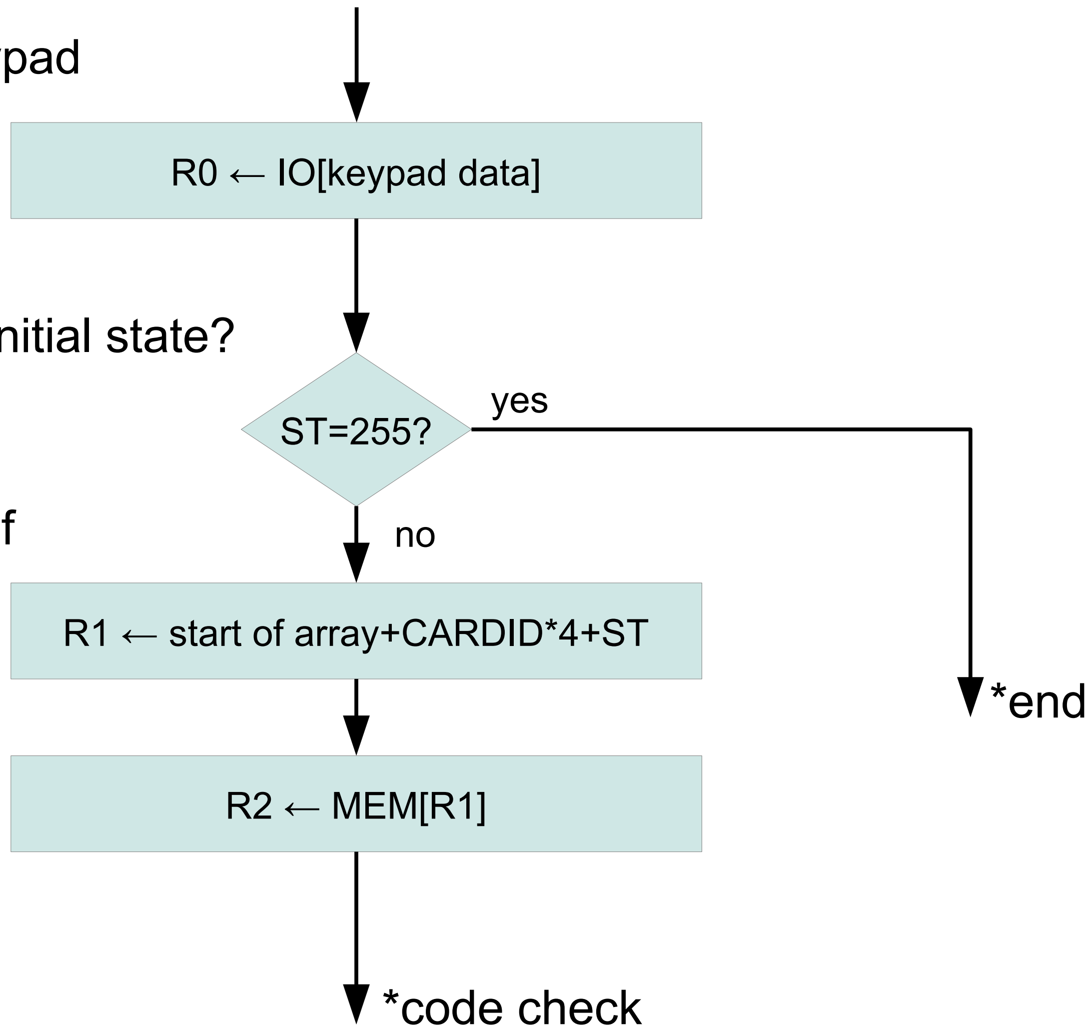
The interrupt is from the keypad...

Read keycode from keypad

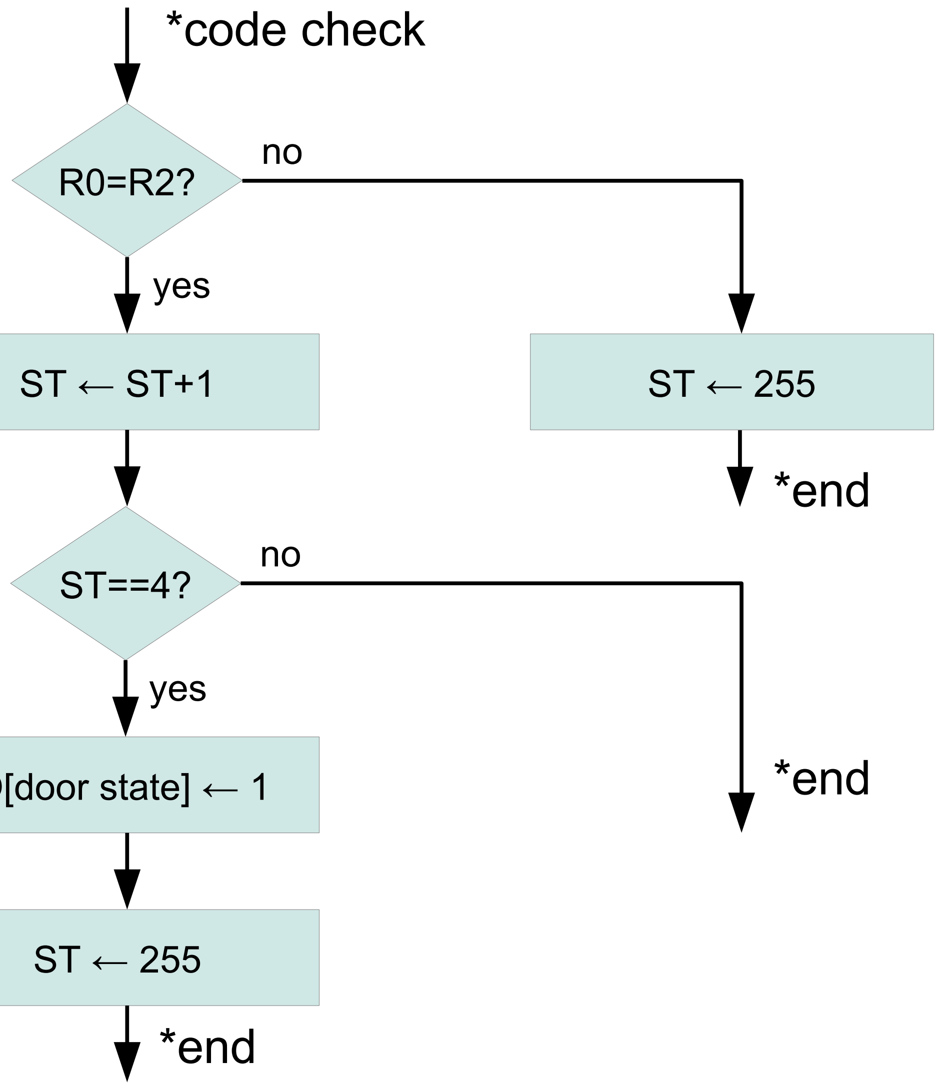
Are we in the initial state?

Calculate the address of
the correct digit

Read the correct digit
from the memory



The interrupt service routine



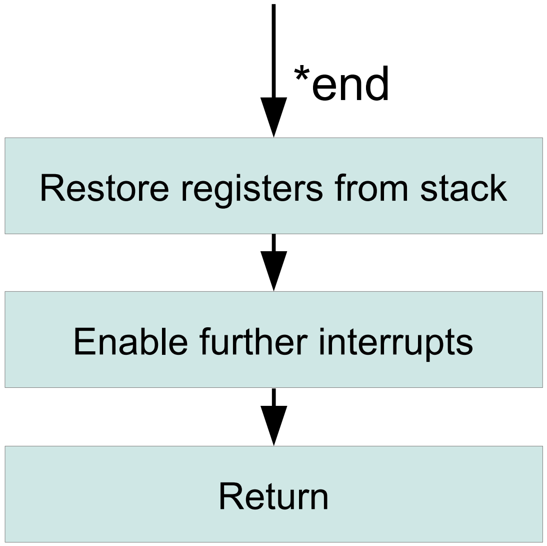
Did we get the correct digit?

If yes, update ST

If all 4 digits are correct, then 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`**, the highest address of the RAM, 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 **`8FFh`**, which is below the interrupt routine

Assembly implementation

- Implementation of the main program:

```
            ORG 0000h               start code at this address
            SP ← 3FFFh              set stack pointer
            MEM[2000h] ← 255        ST=255 (Initial state)
            EI                      enable interrupts
label:      R0 ← IO[4Ch]            read door status
            IO[5Dh] ← R0            update LEDs
            JUMP label              jump back to reading
```

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 IT source is not the reader
`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
         JUMP end IF R1==255             we are not expecting a digit
         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                state machine: ST=225 (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
         MEM[2000h] ← 255                state machine: ST=225 (init. state)
         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