NATIONAL RESEARCH UNIVERSITY

Computer Architecture and Operating Systems
Lecture 12: Memory-Mapped I/O (MMIO)

Andrei Tatarnikov

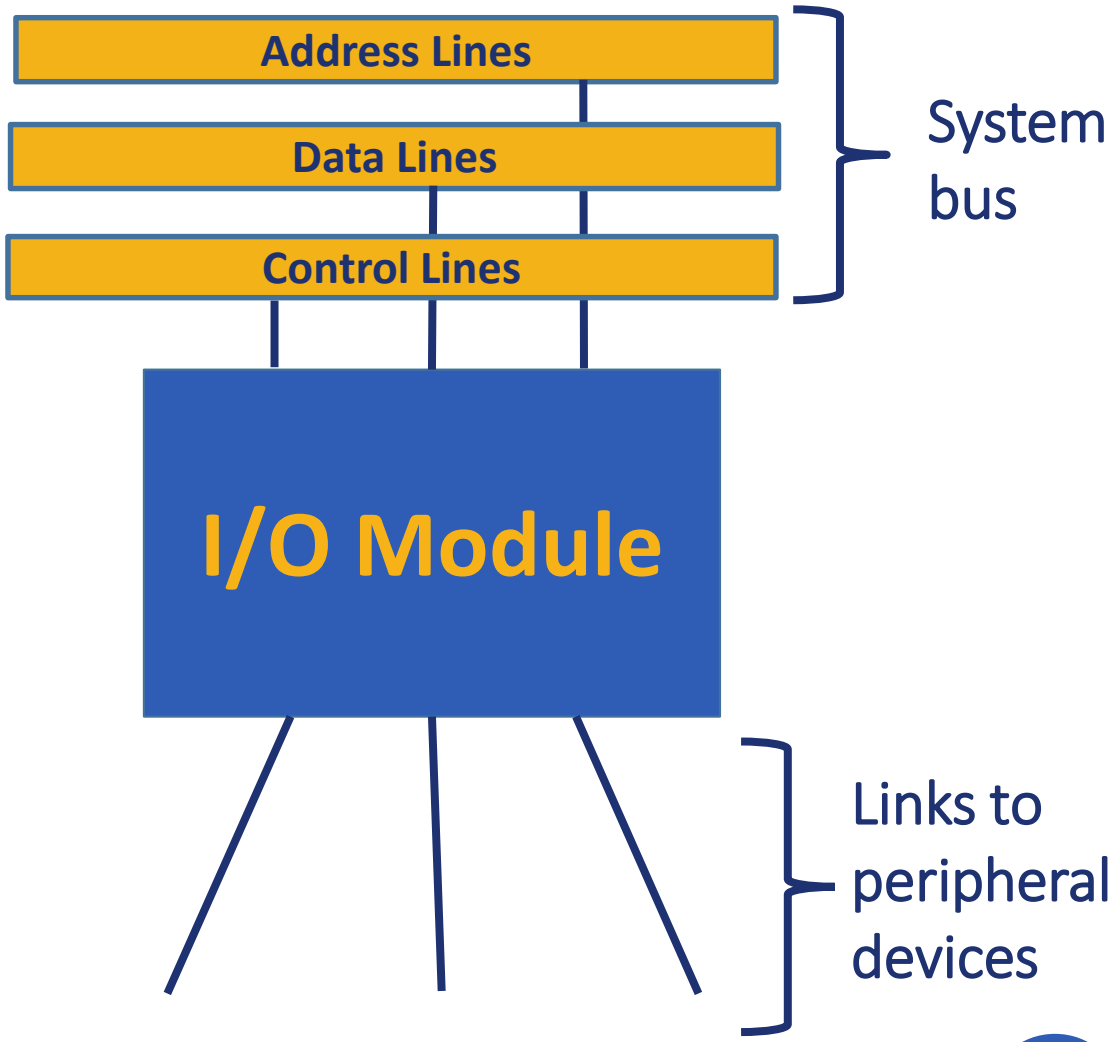
atatarnikov@hse.ru
@andrewt0301

I/O Devices

- Human readable
  - Suitable for communicating with users
  - Video displays, printers
- Machine readable
  - Suitable for communicating with equipment
  - Magnetic disks, SSDs, sensors
- Communication
  - Suitable for communicating with remote devices such as a terminal or another computer
  - Network interface card

I/O Module

- Attach to the processor by a link to an I/O module
  - The link is used to exchange control, status, and data between the I/O module and the external device
- Peripheral device
  - An external device connected to an I/O module



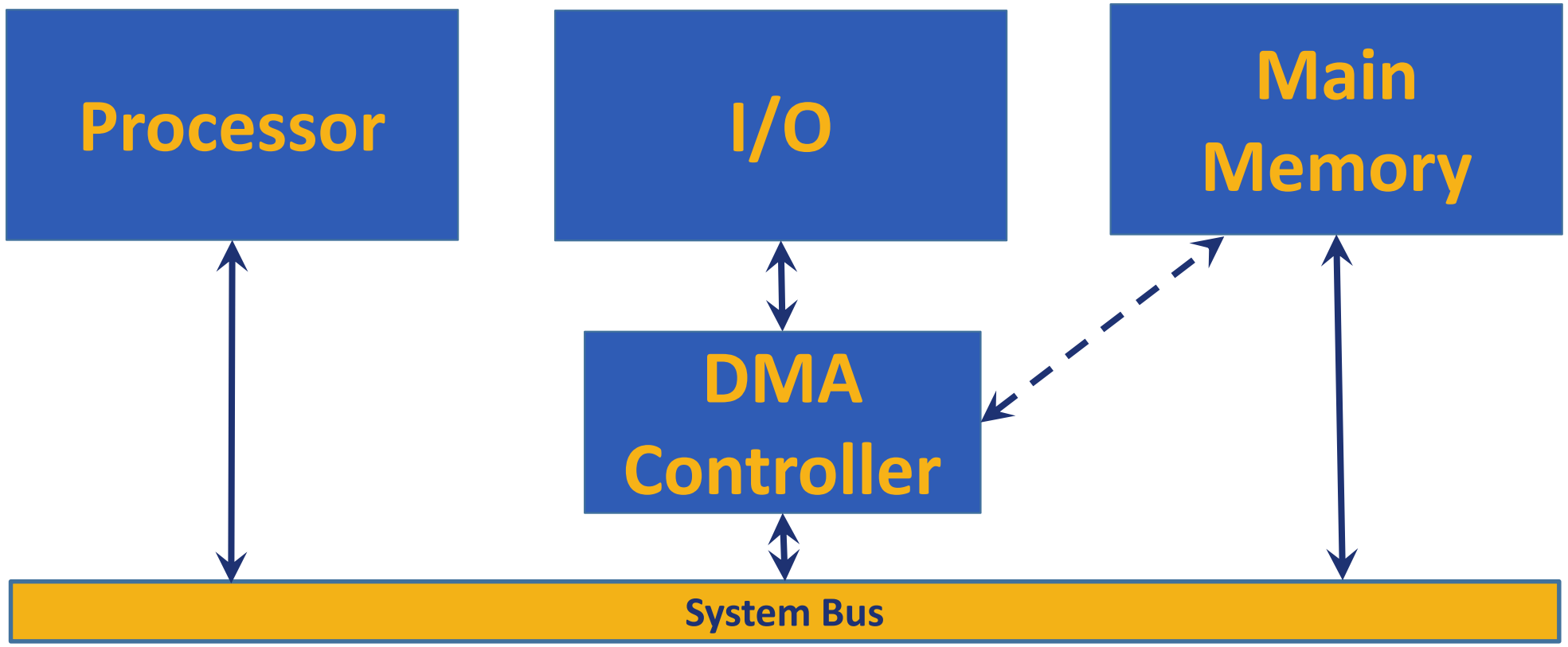
Signals

- **Control signals** determine the function that the device will perform
- **Data** are a set of bits to be sent to or received from the I/O module
- **Status signals** indicate the state of the device

Three Techniques for I/O Operations

- Programmed I/O
  - Data are exchanged between the processor and the I/O module
  - Processor executes a program that gives it direct control of the I/O operation
  - When the processor issues a command it must wait until the I/O operation is complete
  - If the processor is faster than the I/O module this is wasteful of processor time
- Interrupt-driven I/O
  - Processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work
- Direct memory access (DMA)
  - The I/O module and main memory exchange data directly without processor involvement

Direct Memory Access (DMA)



Memory-Mapped I/O (MMIO)

- Processor accesses I/O devices just like memory (like keyboards, monitors, printers)
- Each I/O device assigned one or more address
- When that address is detected, data read/written to I/O device instead of memory
- A portion of the address space dedicated to I/O devices

Key Ideas

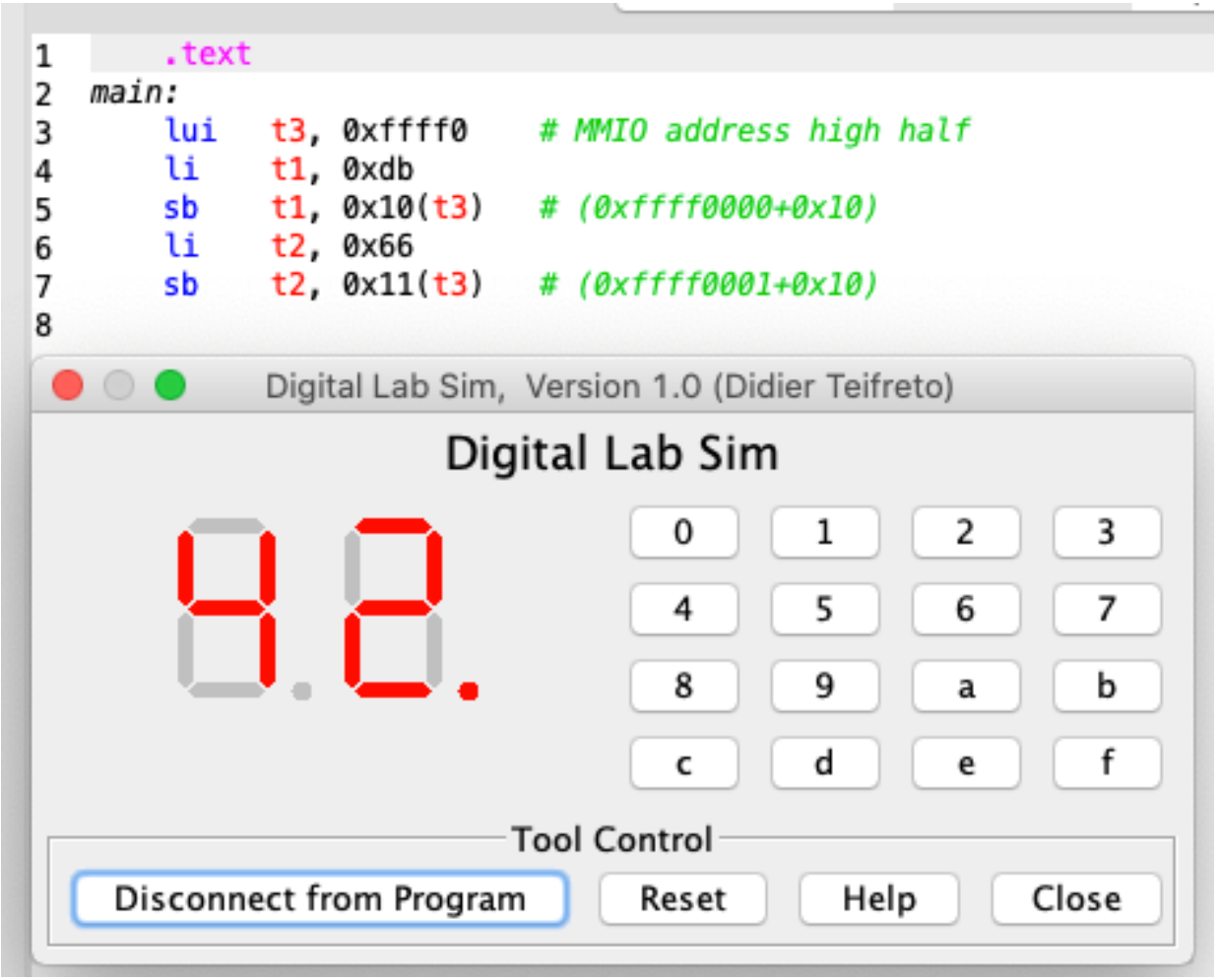
- **Memory-Mapped I/O** is an I/O scheme in which portions of the address space are assigned to I/O devices, and reads and writes to those addresses are interpreted as commands to the I/O device
- **Direct Memory Access (DMA)** is a mechanism that provides a device controller with the ability to transfer data directly to or from the memory without involving the processor

Key Ideas

- **Interrupt-Driven I/O** is an I/O scheme that employs interrupts to indicate to the processor that an I/O device needs attention
- **Polling** is the process of periodically checking the status of an I/O device to determine the need to service the device
- **Device Driver** is a program that controls an I/O device that is attached to the computer

Example: RARS Digital Lab Sim

```
1       .text
2   main:
3       lui     t3, 0xffff0     # MMIO address high half
4       li      t1, 0xdb
5       sb      t1, 0x10(t3)    # (0xffff0000+0x10)
6       li      t2, 0x66
7       sb      t2, 0x11(t3)    # (0xffff0001+0x10)
8
```



Seven segment display

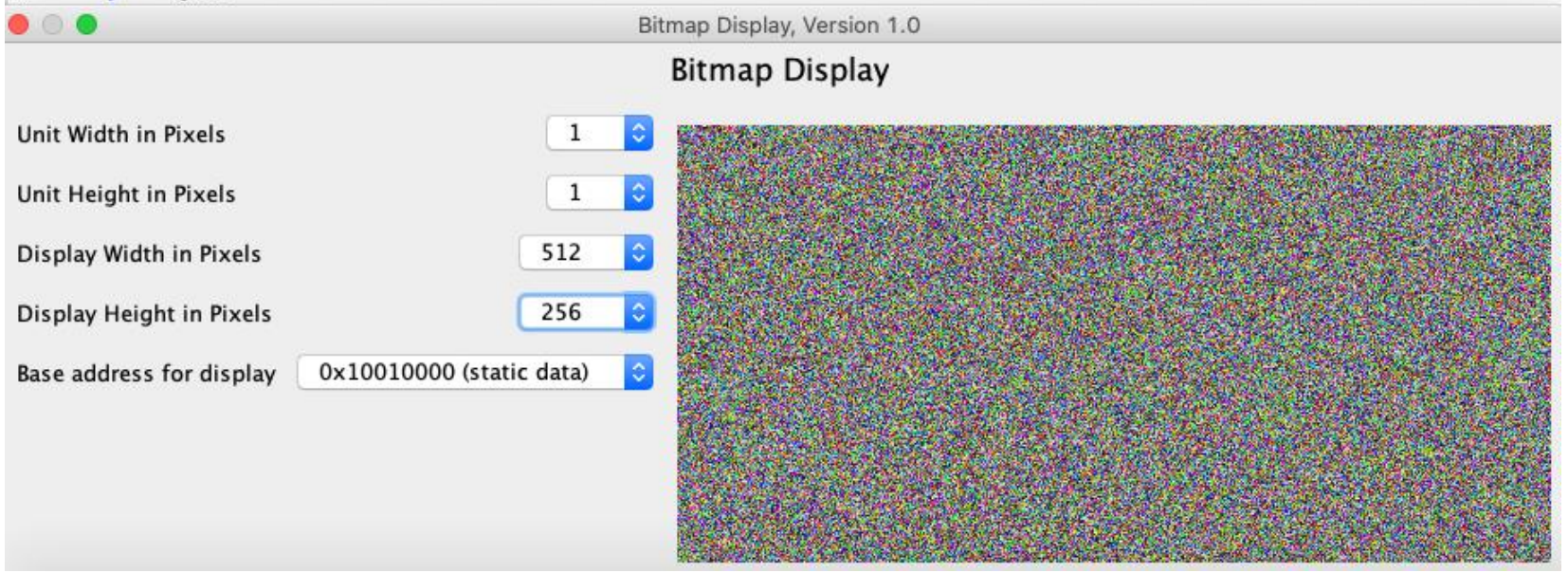
- Byte value at address **0xffff0010**: command right segment display
- Byte value at address **0xffff0011**: command left segment display

Hexadecimal keyboard

- Byte value at address **0xffff0012**: command row number of hexadecimal keyboard (bit 0 to 3) and enable keyboard interrupt (bit 7)
- Byte value at address **0xffff0014**: receive row and column of the key pressed, 0 if not key pressed

Example: RARS Bitmap Display

```
1       .eqv   ALLSIZE 0x20000     # videomemory size (in words)
2       .eqv   BASE    0x10010000  # MMIO base
3       .text
4       li s0, BASE
5   again:
6       mv   a0, zero
7       li   a1, ALLSIZE           # Max 512*Y+X + 1
8       li   a7, 42
9       ecall                      # random 512*Y+X
10
11      slli t2, a0, 2             # make an address by multiplying to 4
12      add  t2, s0, t2            # add addres to base
13
14      mv   a0, zero
15      li   a1, 0x1000000         # MAX RGB value + 1
16      li   a7, 42
17      ecall                      # random color
18
19      sw   a0, 0(t2)
20      j    again
```



Any Questions?

```
            .text
__start:    addi t1, zero, 0x18
            addi t2, zero, 0x21
cycle:      beq t1, t2, done
            slt t0, t1, t2
            bne t0, zero, if_less
            nop
            sub t1, t1, t2
            j cycle
            nop
if_less:    sub t2, t2, t1
            j cycle
done:       add t3, t1, zero
```