Fundusze Europejskie
Wiedza Edukacja Rozwój

Rzeczpospolita Polska

Unia Europejska
Europejski Fundusz Społeczny

Politechnika Śląska jako Centrum Nowoczesnego Kształcenia
opartego o badania i innowacje

POWR.03.05.00-IP.08-00-PZ1/17

Projekt współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego

# Microprocessor and Embedded Systems

**Faculty of Automatic Control, Electronics and Computer Science, Informatics, Bachelor Degree**

# Lecture 10

# Microprocessor systems development & testing

Bartłomiej Zieliński, PhD, DSc

# Development & testing

Program:

- Problems during µp systems development
  - Hardware problems
  - Software problems
  - I/O problems
- µp systems analysis
  - Asynchronous analysis
  - Synchronous analysis
- Development tools
  - Debuggers, simulators
  - Logic state analyzers
  - In-circuit emulators

# Development & testing

- µp systems
  - µp's are VLSI
    - No internal signals available

      →Observe bus signals only
  - µp works according to a program
    - Results of both hardware & software

      →Data collected from all buses concurrently

- µp systems testing & development
  - New methods & tools necessary

# Development & testing

- Problems during µp systems development
  - Hardware problems
    - „cold solder”, bad contacts
      → Variable, weak or no electric contact
    - IC damage or failure
    - Bad IC installation on PCB
    - Bad contact with IC socket
    - Bad PCB project
      → crosstalk, interference
    - High resistance of Vcc and Gnd paths
      → improper logic signals voltage levels
      → interference margin decreased

# Development & testing

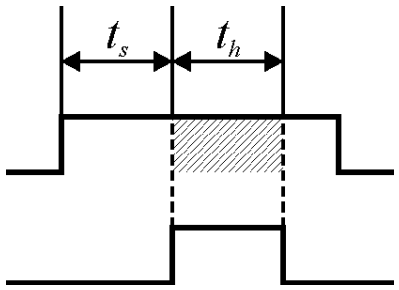
- Problems during µp systems development
  - Software problems
    - Hardware OK, software OK, together work bad
      - Single step operation
    - Loops – jump to a random place
    - „data execution”
    - Sometimes works, sometimes not
      - *„if something worked and stopped, something must have changed*” (Alex Ragen: „A lexicon of C” (?))
  - I/O problems
    - Wrong transmission rate
    - Wrong data format
    - Transmission lines interference

# Development & testing

- Synchronous analysis
  - Machine cycles storage according to µp clock
    - Triggered by a given word
      - Only a part of a program is observed/debugged
      - Some words preceding trigger must also be stored
        - → *How the µp came to this program fragment?*
          - Jump / next instruction / malfunction
    - Counting a trigger
      - When circuit works or not
      - Analysis starts with $n^{th}$ pass

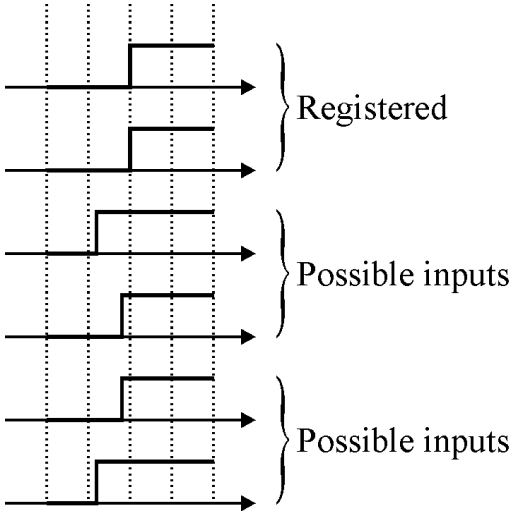
# Development & testing

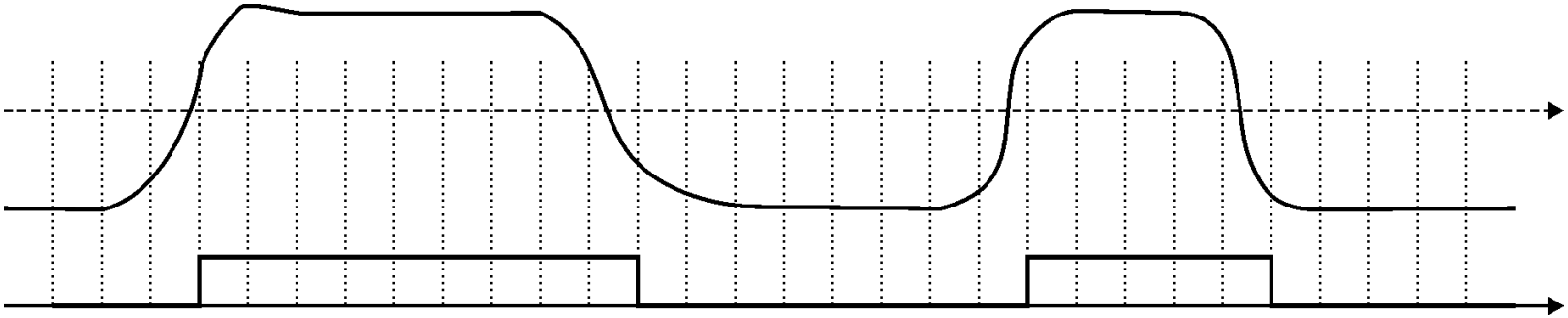
- Synchronous analysis
  - Machine cycles storage according to µp clock
    - Trigger by a „logic delay”
      - Trigger word can’t be precisely determined
        - » Trigger found („last known as good”)
        - » Count *n* cycles
        - » Start to register
    - Dynamic parameters not always taken into account
      - „time set”, „time hold”



# Development & testing

- Asynchronous analysis
  - Probing at moments determined by analyser clock
    - Very high frequency
    - Probing all channels concurrently
  - Results may be ambigous
    - $f \neq \infty$
    - Internal analyser delays



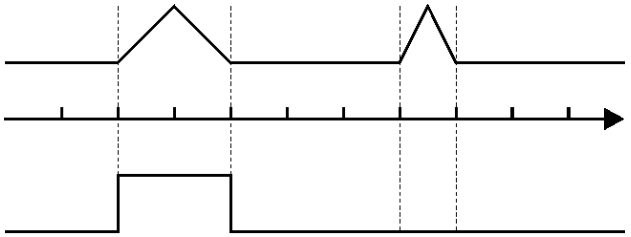



# Development & testing

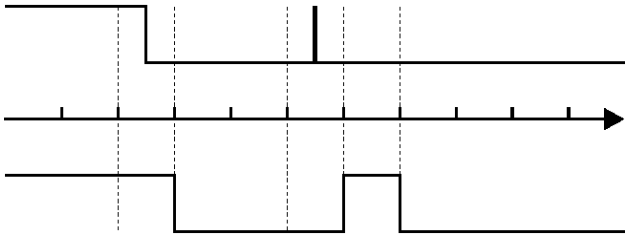
- Asynchronous analysis
  - Glitch detection
    - What is a glitch?
      - Very short pulse
      - Result of: crosstalk, interference, bad Vcc/Gnd
      - Cause of: false Int, count errors, data exchange errors…
    - How to detect a glitch?
      - Very high frequency (e.g. 200 MHz → 5 ns glitch)
        - » (Glitch < 5 ns not very relevant for µp)
      - Latch flip-flops
        - » Was there a single glitch, or a sequence?
      - Separate signal and glitch registration

# Development & testing

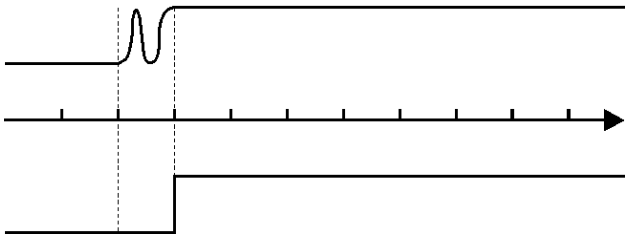
- Asynchronous analysis
  - Glitch detection
    - Probing method



    - With latch registers

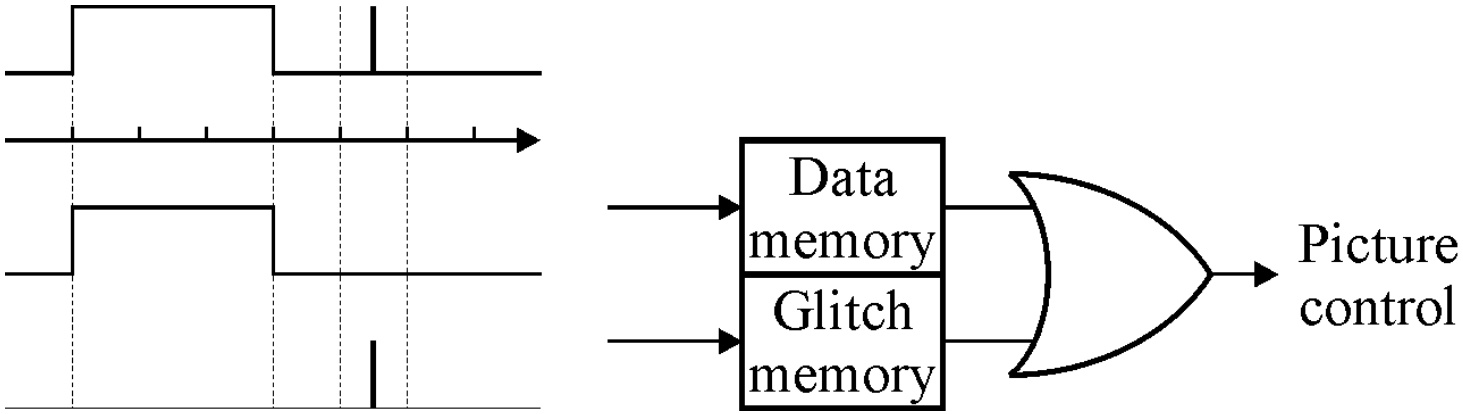


    - Glitch might be omitted by analyser input



# Development & testing

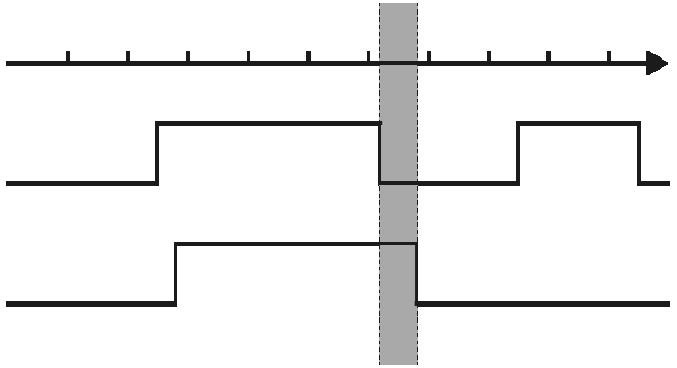
- Asynchronous analysis
  - Glitch detection
    - Separate data and glitch recording



# Development & testing

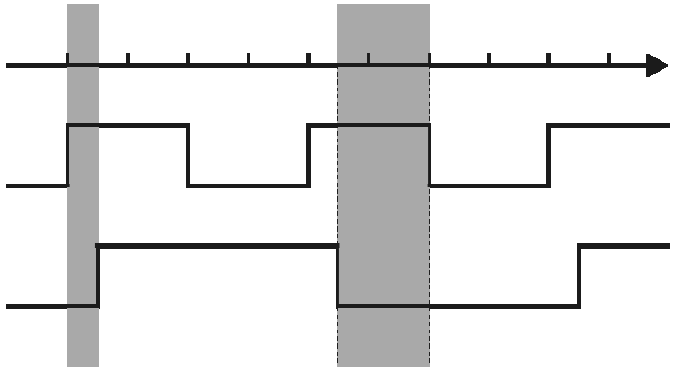
- Asynchronous analysis
  - Trigger condition ambiguity in probed signals
    - Missed trigger condition

      (between probes)



    - False trigger conditions

      (e.g., because of different delays)



# Development & testing

- Asynchronous analysis
  - Trigger condition ambiguity in probed signals
    - Time dependencies with regard to clk
      - Delay = circuit propagation delay, and trigger pattern comparison only after that
    - Recording with some delay after a trigger
      - Interesting consequence of a dull („boring”) event
    - Glitch-triggered recording
      1. Recording a single word
      2. No glitch → goto 1
      3. Glitch found → record longer

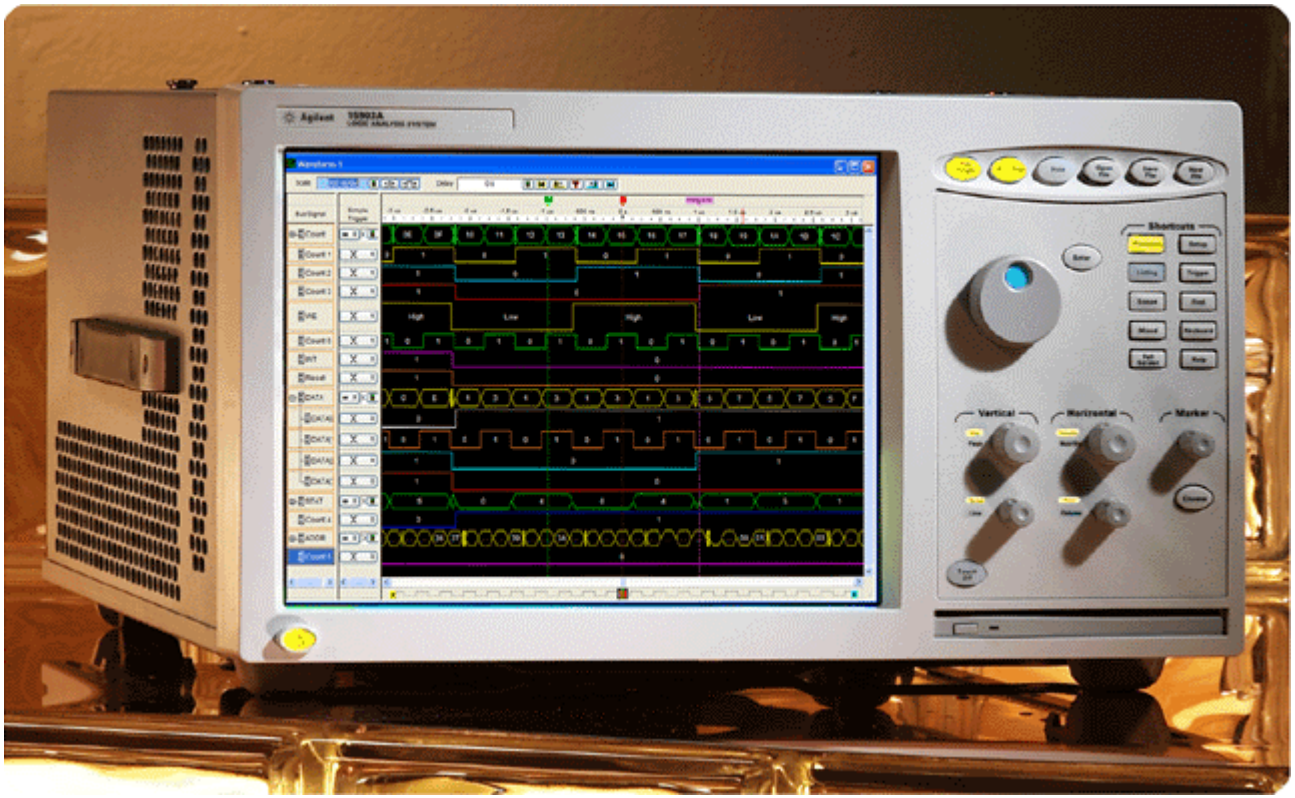
      → where/when was a glitch, and what conseqences it caused

# Development & testing

- Analysis – common problems
  - Improper power, circuit assembly error
  - Wrong voltage levels (e.g. TTL vs CMOS)
  - Bad connections
    - Analyser shows something different than the µp can see
  - Time set/time hold
  - etc.

# Development & testing

- Logic analyser – example



https://vishnuanirudh.files.wordpress.com/2012/07/logic-analyzer.gif

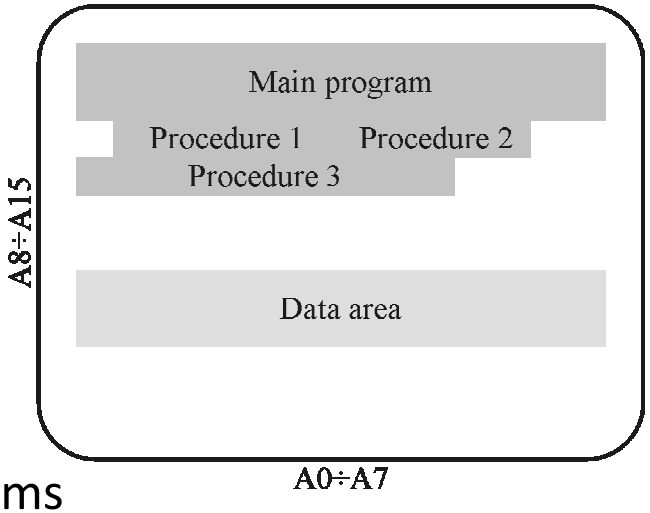
# Development & testing

- Logic analyser – example


https://eleshop.eu/sla1016.html

# Development & testing

- Logic analyser – functions
  - Like oscilloscope, but:
    - Many inputs (e.g., > 100) concurrently observed
    - Digital signals
    - Pattern-triggered analysis
  - Output:
    - Series of „0” and „1”
    - Time diagrams for many signals
    - Memory access map
      - Useful for debugging µp-based systems
    - Disassembled program

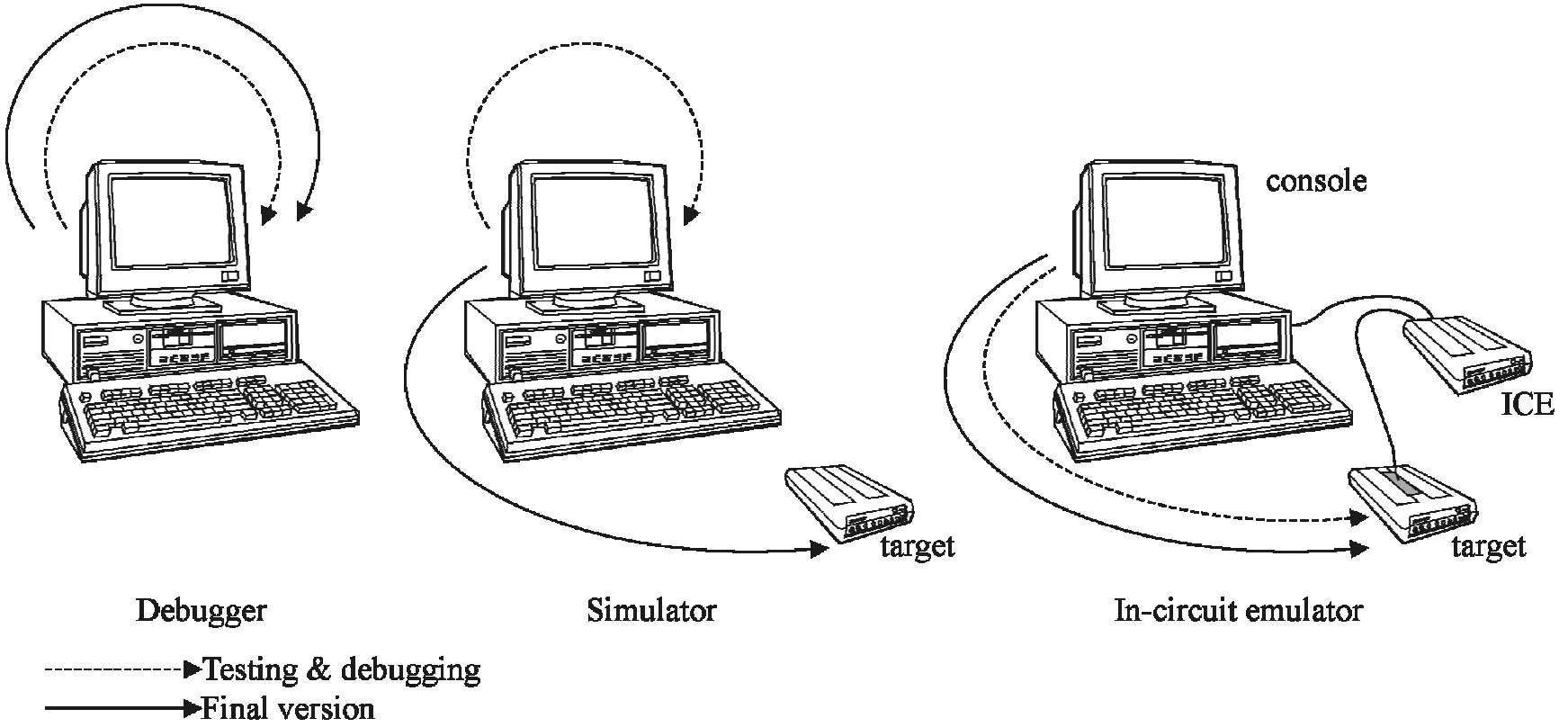


# Development & testing

- Logic analyser – summary
  - Good for finding both hardware & software errors
  - „Observe-only” operation
    - Passive recording circuit behaviour
    - No active influence on circuit
    - We can find errors/mistakes, but we can’t fix them
  - Other tools could be useful

# Development & testing

- Debugger / simulator / in-circuit emulator

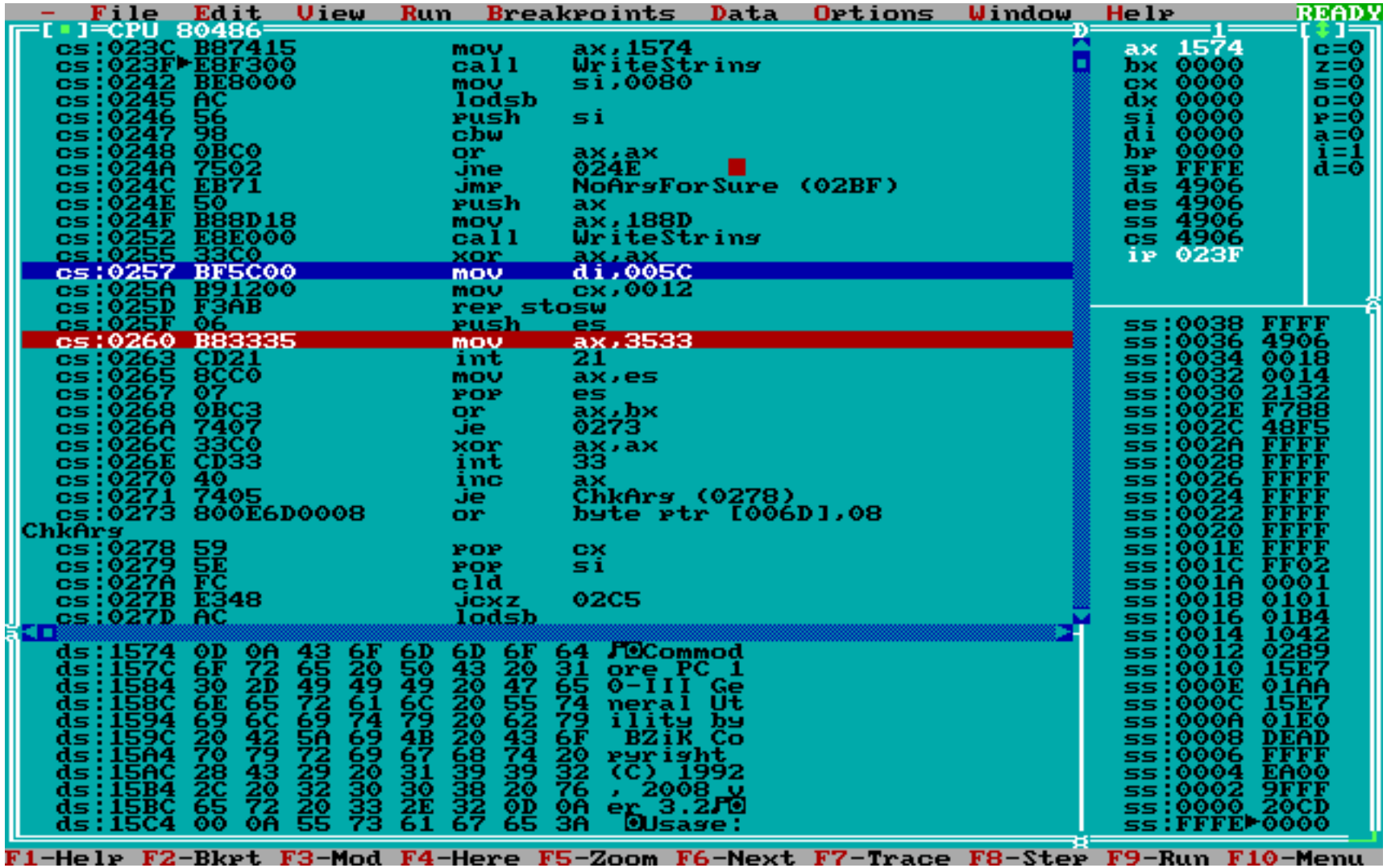


# Development & testing

- Debugger / simulator
  - Continuous/single step program execution
    - Some simulators can perform „step back”
  - Program displayed as assembler code
    - Symbolic names possible
    - High-level language debugging possible
      - Requires compiler/linker information for the debugger
  - µp registers content display/modification
  - Memory content display/modification

# Development & testing

- Turbo Debugger 2.0 example

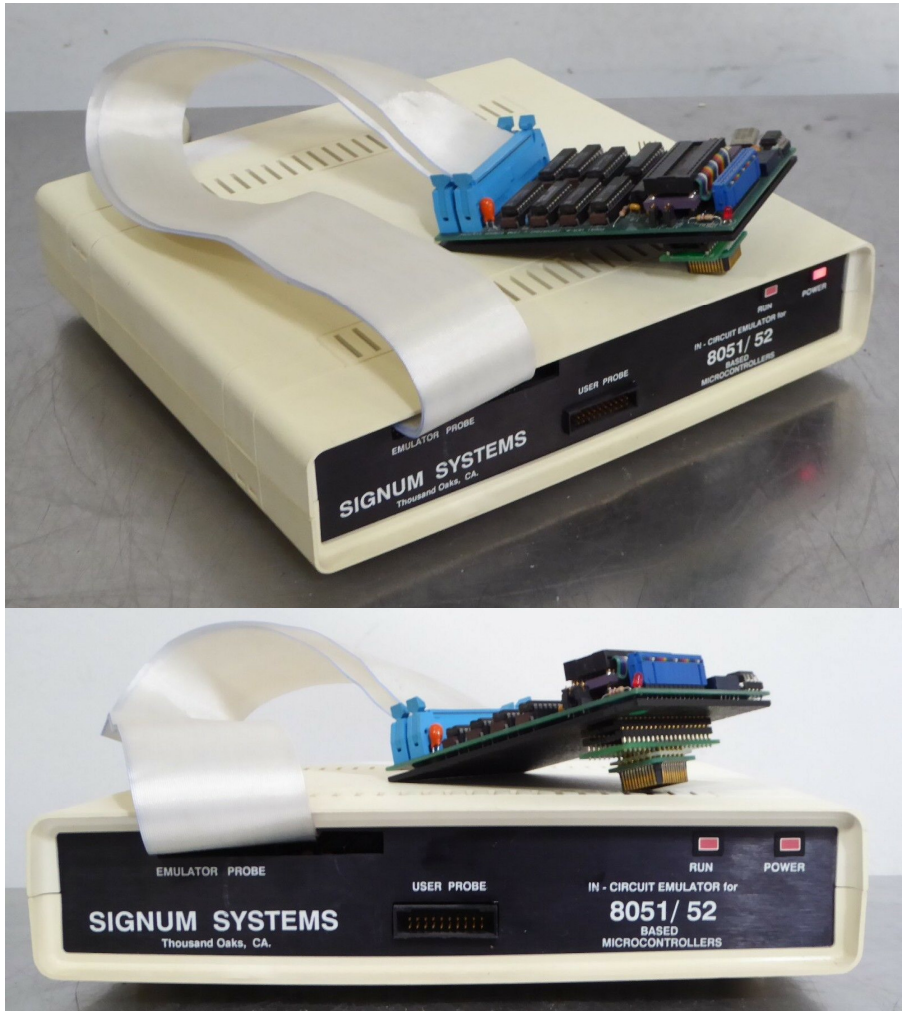


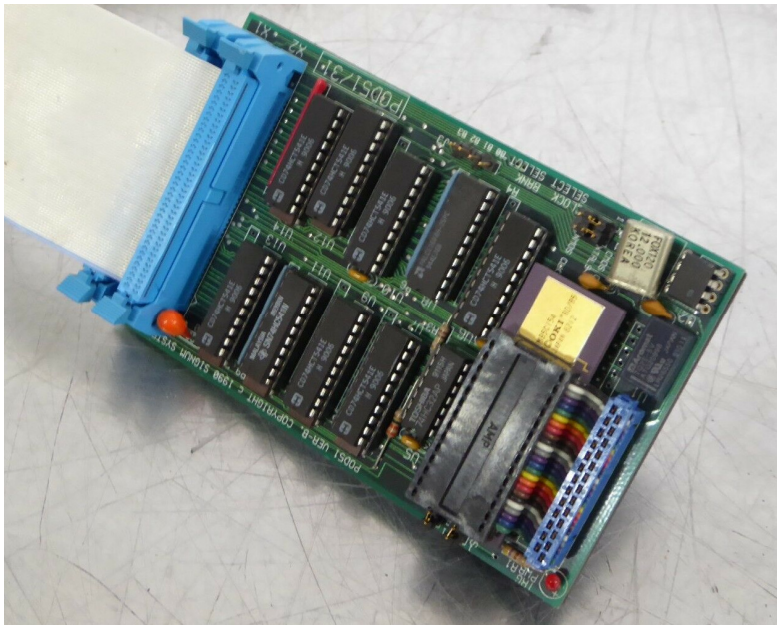
# Development & testing

- Simulator vs debugger
  - Simulator works in a „virtual” environment
    - Commands → procedures
    - Registers → variables
    - Target system memories → arrays, tables
    - Target system I/O → records + procedures
    - Address ranges for Mem & IO must be defined
  - Real-time simulation requires:
    - Detailed knowledge of µp cycles (command cycles, machine cycles, clock cycles)
    - Significantly higher computing power (e.g., Pentium II @ 300 MHz can simulate 8051 @ 12 MHz)

# Development & testing

- In-circuit emulator – example
  - Signum Systems ICE-51



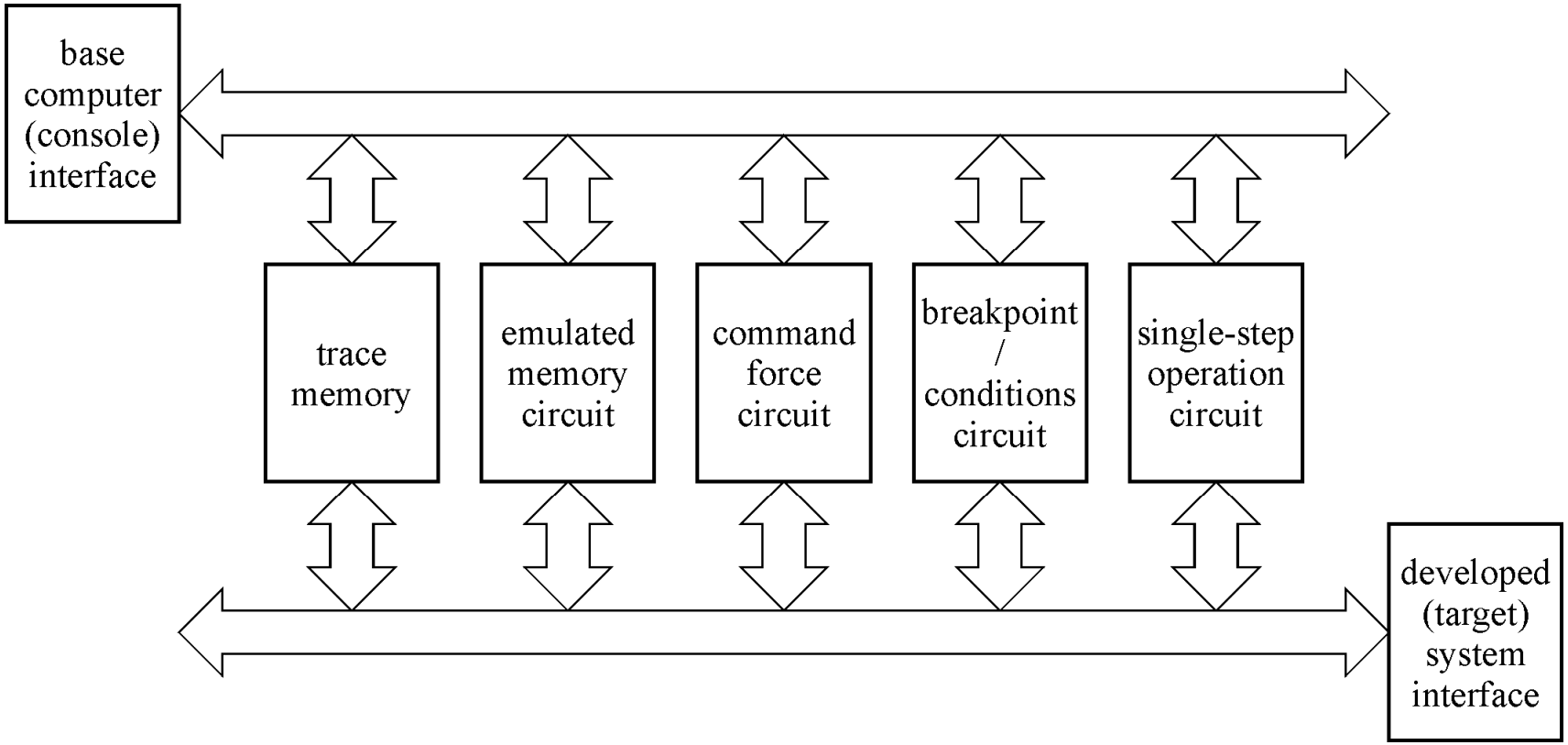



# Development & testing

- In-circuit emulator
  - User can „see” the target circuit exactly as the µp does
    - Memory types and addresses
    - I/O interfaces
    - External environment
  - All imperfections can be found
    - Logic design errors (i.e., wrong address decoding)
    - PCB design/production error
    - Circuit assembly mistakes
    - Hardware/software integration problems
    - Pure software problems

# Development & testing

- In-circuit emulator – example structure



# Development & testing

- In-circuit emulator – example structure
  - Single-step circuit
    - Modes
      - Continuous mode
      - Stop at the nearest command
        - » For software testing/debugging
      - Stop at the nearest machine cycle
        - » For hardware testing/debugging
    - How to implement?
      - Classical µp (e.g., Z80, 8086, etc.) – machine cycle extension input (e.g., $\overline{\text{Wait}}$, Ready etc.)
      - Single chip / µc – must have the support built-in (e.g., 8048 has $\overline{\text{SS}}$ input, 8051 – no built-in support)

# Development & testing

- In-circuit emulator – example structure
  - Breakpoint circuit
    - Simple
      - Address and/or data equal to a given value
      - Specified machine cycle occured
    - (not so) simple
      - Address and/or data matches the pattern
        - » Only selected bits are compared
      - Conjunction of simple conditions

# Development & testing

- In-circuit emulator – example structure
  - Breakpoint circuit
    - Complex (e.g., Signum Systems ICE-51)
      - Events
        - » Selected machine cycle (+adress, data pattern match)
        - » User signals
        - » Event count
        - » Event sequence
        - » Trace memory full
      - Effects
        - » Breakpoint
        - » Event counter start/stop
        - » Trace end

# Development & testing

- In-circuit emulator – example structure
  - Command force circuit
    - To make the µp perform a command/procedure that is absent from the program memory
    - *… but why do we need it?*
      - To quickly fix a bug without need to recompile the program
      - To perform a diagnostic procedure
        - » E.g., store register content outside of the µp after breakpoint occured
        - » Update µp state window
      - To quickly test some actions before they are implemented in software/in system

# Development & testing

- In-circuit emulator – example structure
  - Emulated memory circuit
    - Replaces/completes system RAM/ROM
    - Contains RAM only
      - Higher priority than system memory
        - » „hides”/works instead of the system memory
      - Content can be modified easily
        - » Especially when system memory is ROM/PROM/EEPROM
      - Address ranges user-definable
    - *What about I/O interfaces?*

# Development & testing

- In-circuit emulator – example structure
  - Trace memory circuit
    - Stores the execution trace
      - All the machine cycles found on the µp pins
    - User signals can also be traced
    - Presented as a program flow with decoded commands, data, I/O signals, etc.
      - Like logic analysers (synchronous analysis)

# Development & testing

- JTAG – modern methods
  - *Joint Test Action Group* (1985)
    - IEEE Std. 1149 (1990)
  - Early applications
    - device, board & system testing & diagnosis
  - Modern applications
    - Access of IC sub-blocks
    - Debugging of embedded systems
    - Firmware programming
    - Boundary scan

# Development & testing

- JTAG – interface
  - Another built-in interface in a µp/µc
  - „full” JTAG
    - TDI (*Test Data In*)
    - TDO (*Test Data Out*)
    - TCK (*Test Clock*)
    - TMS (*Test Mode Select*)
    - TRST (*Test Reset*, optional)
  - „compact” JTAG – cJTAG
    - TMSC (*Test Serial Data*)
    - TCKC (*Test Clock*)