



**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

**B**artłomiej Zieliński, PhD, DSc

# Development & testing

---

## Program:

- Problems during  $\mu\text{p}$  systems development
  - Hardware problems
  - Software problems
  - I/O problems
- $\mu\text{p}$  systems analysis
  - Asynchronous analysis
  - Synchronous analysis
- Development tools
  - Debuggers, simulators
  - Logic state analyzers
  - In-circuit emulators

# Development & testing

---

- $\mu$ p systems
  - $\mu$ p's are VLSI
    - No internal signals available
      - Observe bus signals only
  - $\mu$ p works according to a program
    - Results of both hardware & software
      - Data collected from all buses concurrently
- $\mu$ p systems testing & development
  - New methods & tools necessary

# Development & testing

---

- Problems during  $\mu\text{p}$  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  $\mu\text{p}$  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

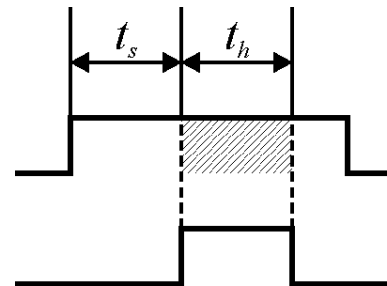
---

- Synchronous analysis
  - Machine cycles storage according to  $\mu$ p 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  $\mu$ p came to this program fragment?*
          - Jump / next instruction / malfunction
  - Counting a trigger
    - When circuit works or not
    - Analysis starts with  $n^{\text{th}}$  pass

# Development & testing

---

- Synchronous analysis
  - Machine cycles storage according to  $\mu\text{p}$  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

---

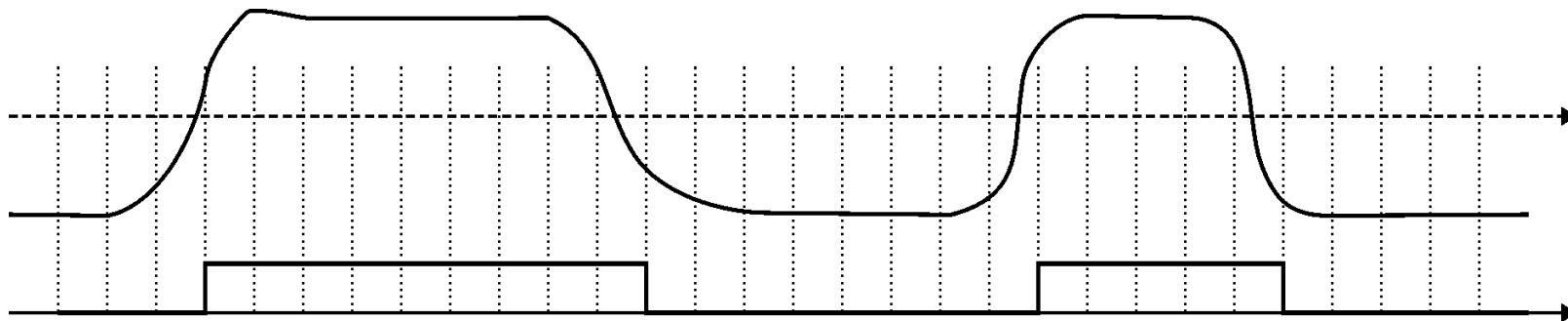
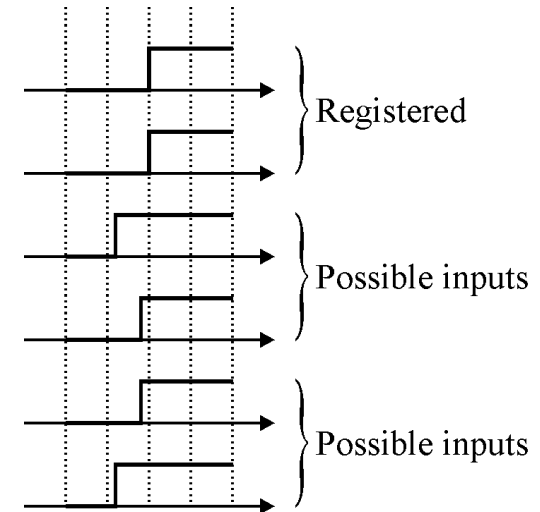
- Asynchronous analysis

- Probing at moments determined by analyser clock

- Very high frequency
    - Probing all channels concurrently

- Results may be ambiguous

- $f \neq \infty$
    - 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  $\mu\text{p}$ )
      - Latch flip-flops
        - » Was there a single glitch, or a sequence?
      - Separate signal and glitch registration

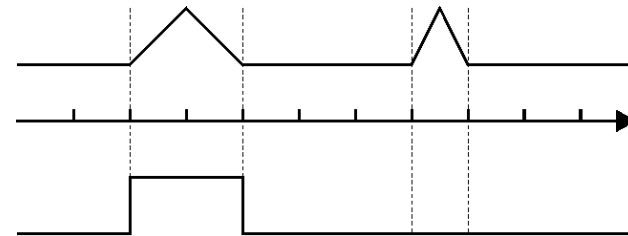
# Development & testing

---

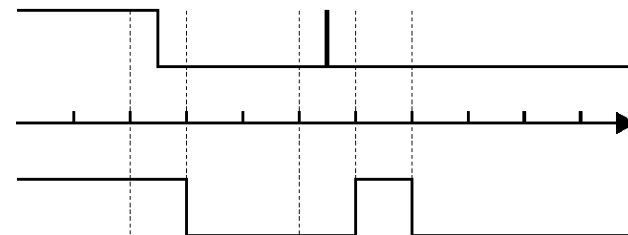
- Asynchronous analysis

- Glitch detection

- Probing method



- With latch registers



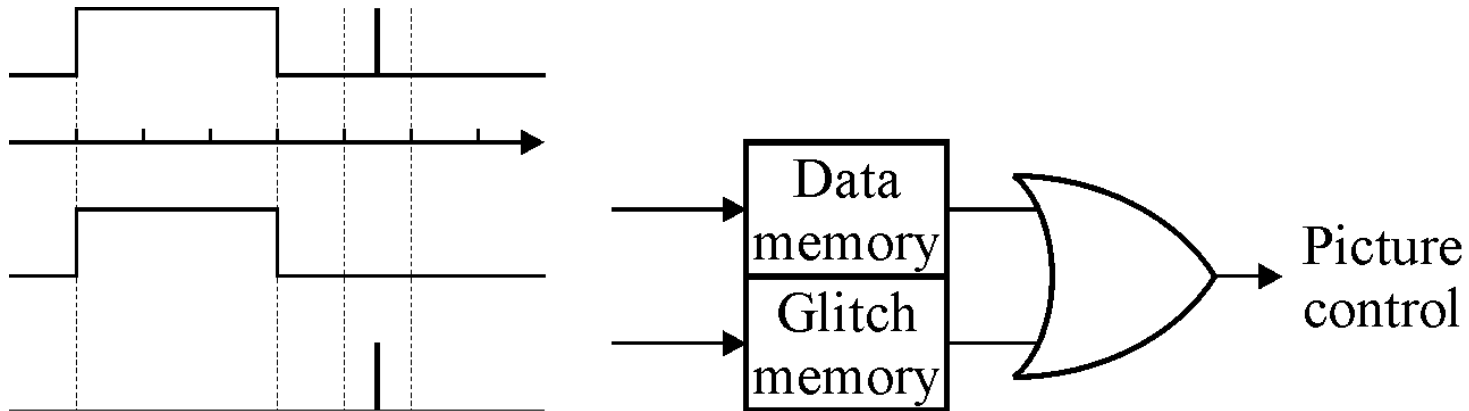
- Glitch might be omitted by analyser input



# Development & testing

---

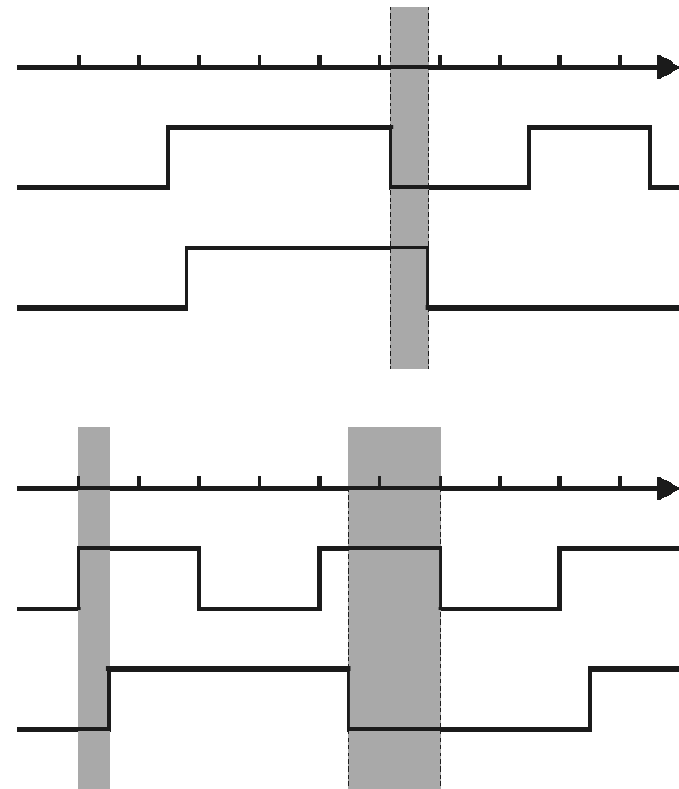
- Asynchronous analysis
  - Glitch detection
    - Separate data and glitch recording



# 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 consequences it caused

# Development & testing

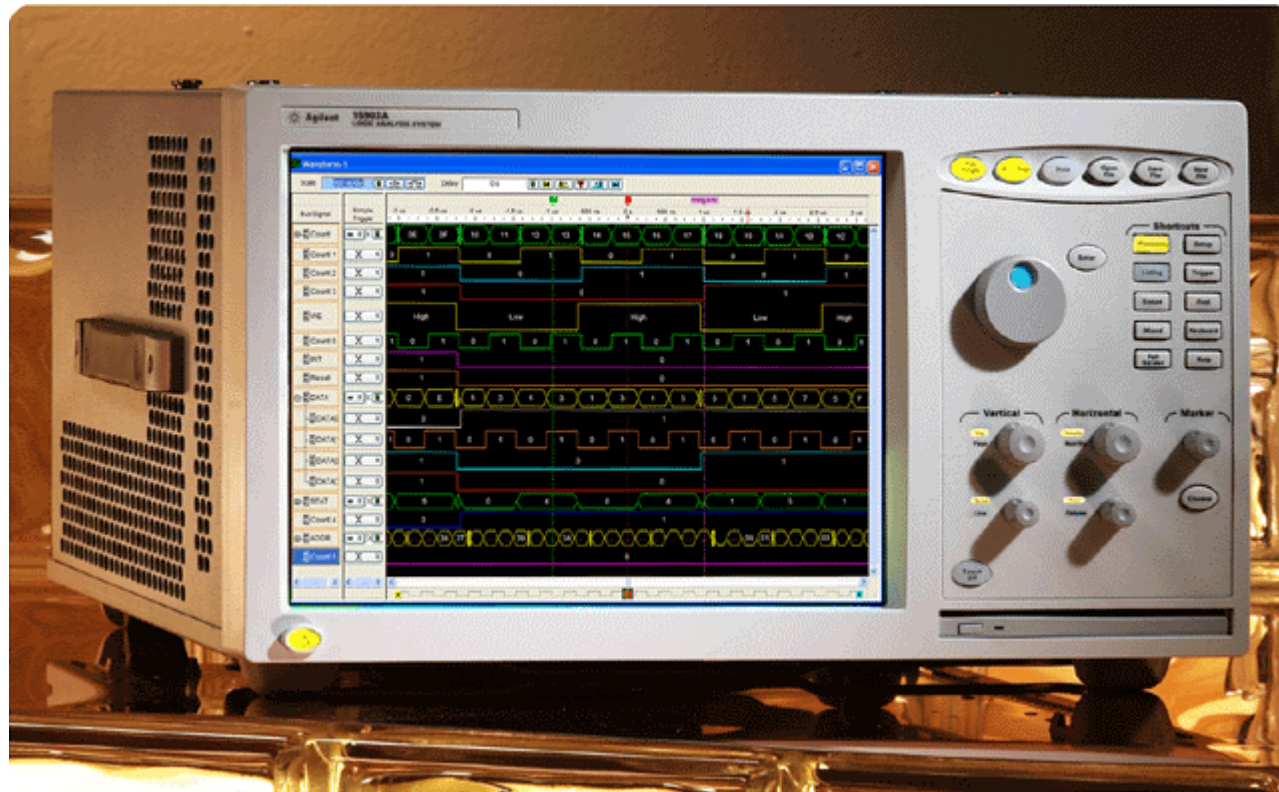
---

- Analysis – common problems
  - Improper power, circuit assembly error
  - Wrong voltage levels (e.g. TTL vs CMOS)
  - Bad connections
    - Analyser shows something different than the  $\mu\text{p}$  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

---

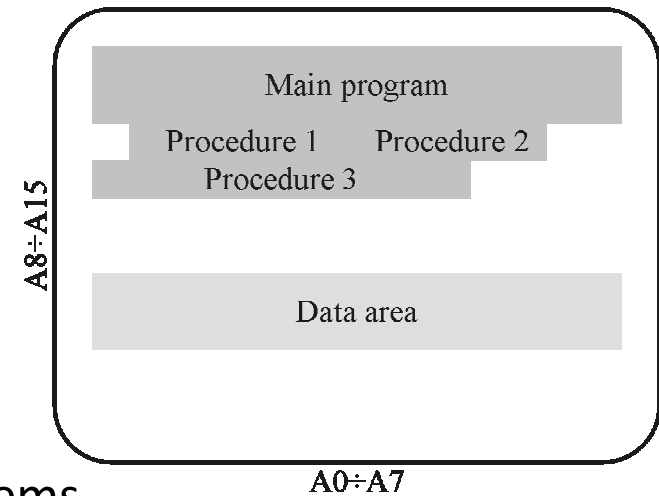
- Logic analyser – functions

- 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  $\mu$ p-based systems
    - Disassembled program



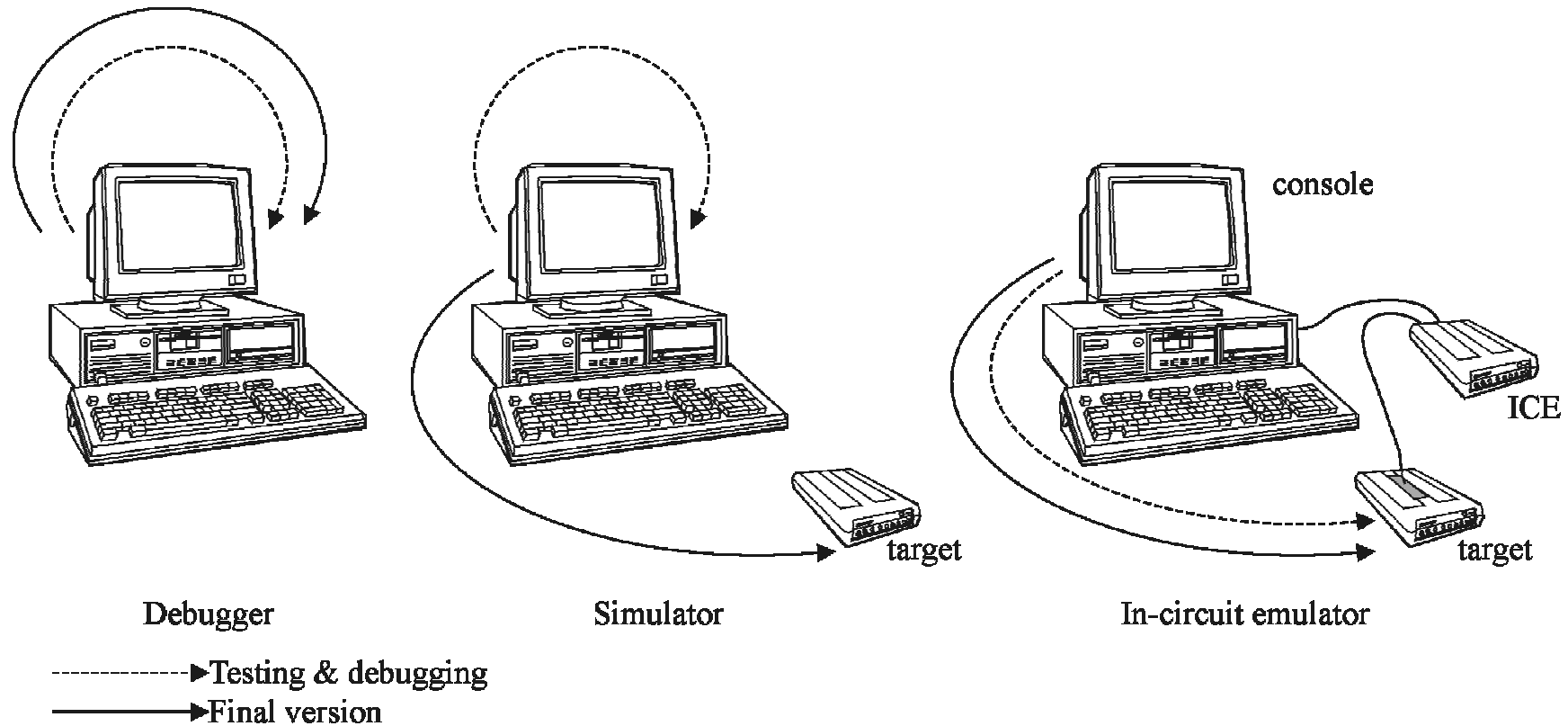
# Development & testing

---

- 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
  - $\mu$ p registers content display/modification
  - Memory content display/modification

# Development & testing

- Turbo Debugger 2.0 example

The screenshot displays the Turbo Debugger 2.0 interface. The main window shows assembly code for the CS segment, with the instruction at address 0257 (BF5C00) highlighted in red. The instruction is `mov di, 005C`. The registers window on the right shows the current state of the registers, with IP at 023F. The memory window at the bottom shows the contents of memory starting at address 1574, displaying a list of names and their corresponding memory addresses.

```
- File Edit View Run Breakpoints Data Options Window Help
CPU 80486
cs:023C B87415 mov ax,1574
cs:023F E8F300 call WriteStrins
cs:0242 BE8000 mov si,0080
cs:0245 AC lodsb
cs:0246 56 push si
cs:0247 98 cbw
cs:0248 0BC0 or ax,ax
cs:024A 7502 jne 024E
cs:024C EB71 jmp NoArsForSure (02BF)
cs:024E 50 push ax
cs:024F B88D18 mov ax,188D
cs:0252 E8E000 call WriteStrins
cs:0255 33C0 xor ax,ax
cs:0257 BF5C00 mov di,005C
cs:025A B91200 mov cx,0012
cs:025D F3AB rep stosw
cs:025F 06 push es
cs:0260 B83335 mov ax,3533
cs:0263 CD21 int 21
cs:0265 8CC0 mov ax,es
cs:0267 07 pop es
cs:0268 0BC3 or ax,bx
cs:026A 7407 je 0273
cs:026C 33C0 xor ax,ax
cs:026E CD33 int 33
cs:0270 40 inc ax
cs:0271 7405 je ChkArs (0278)
cs:0273 800E6D0008 or byte ptr [006D],08
ChkArs
cs:0278 59 pop cx
cs:0279 5E pop si
cs:027A FC cld
cs:027B E348 jcxz 02C5
cs:027D AC lodsb
ds:1574 0D 0A 43 6F 6D 6D 6F 64 J@Commod
ds:157C 6F 72 65 20 50 43 20 31 ore PC 1
ds:1584 30 2D 49 49 49 20 47 65 0-III Ge
ds:158C 6E 65 72 61 6C 20 55 74 neral Ut
ds:1594 69 6C 74 79 20 62 79 ility by
ds:159C 20 42 5A 69 4B 20 43 6F BZiK Co
ds:15A4 70 79 72 69 67 68 74 20 ryright
ds:15AC 28 43 29 20 31 39 39 32 (C) 1992
ds:15B4 2C 20 30 30 30 30 76 , 2008 J@
ds:15BC 65 72 33 2E 32 0D 0A er 3.2J@
ds:15C4 00 0A 55 73 61 67 65 3A @Usase:
ax 1574
bx 0000
cx 0000
dx 0000
si 0000
di 0000
bp 0000
sp FFFE
ds 4906
es 4906
ss 4906
ip 023F
c=0
z=0
s=0
o=0
v=0
a=0
i=1
d=0
ss:0038 FFFF
ss:0036 4906
ss:0034 0018
ss:0032 0014
ss:0030 2132
ss:002E F788
ss:002C 48F5
ss:002A FFFF
ss:0028 FFFF
ss:0026 FFFF
ss:0024 FFFF
ss:0022 FFFF
ss:0020 FFFF
ss:001E FFFF
ss:001C FF02
ss:001A 0001
ss:0018 0101
ss:0016 01B4
ss:0014 1042
ss:0012 0289
ss:0010 15E7
ss:000E 01AA
ss:000C 15E7
ss:000A 01E0
ss:0008 DEAD
ss:0006 FFFF
ss:0004 EA00
ss:0002 9FFF
ss:0000 20CD
ss:FFFE 0000
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

# Development & testing

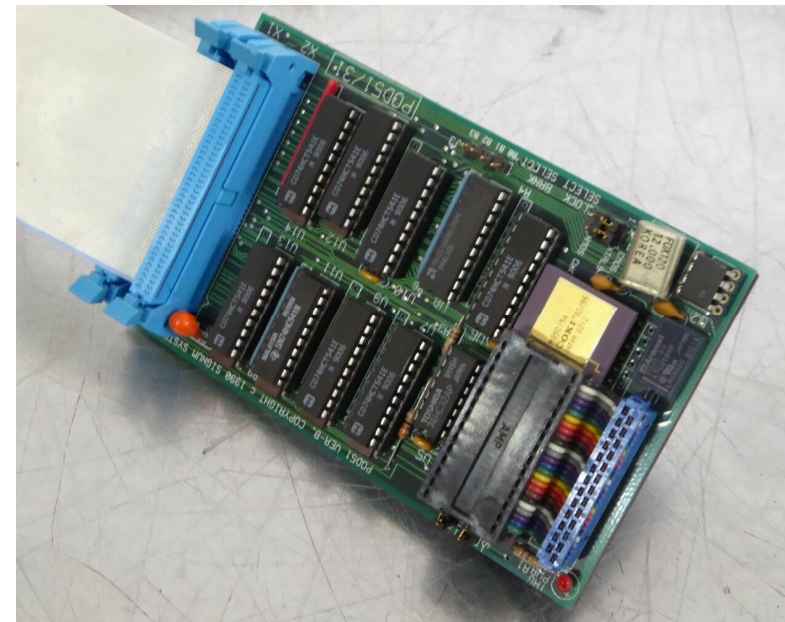
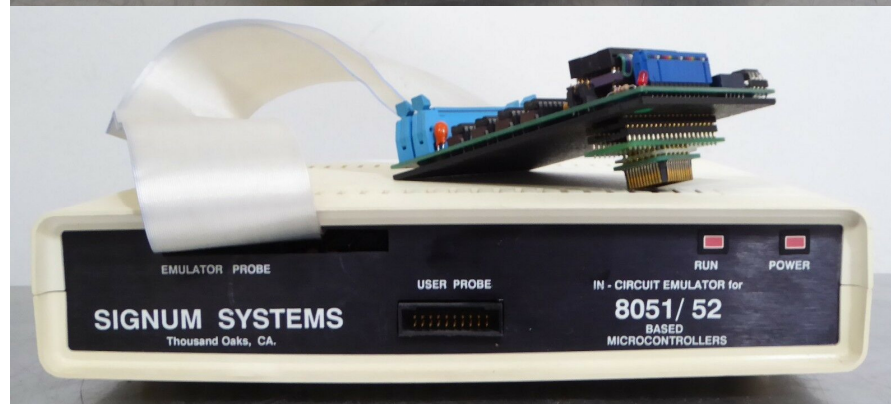
---

- 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  $\mu$ p 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





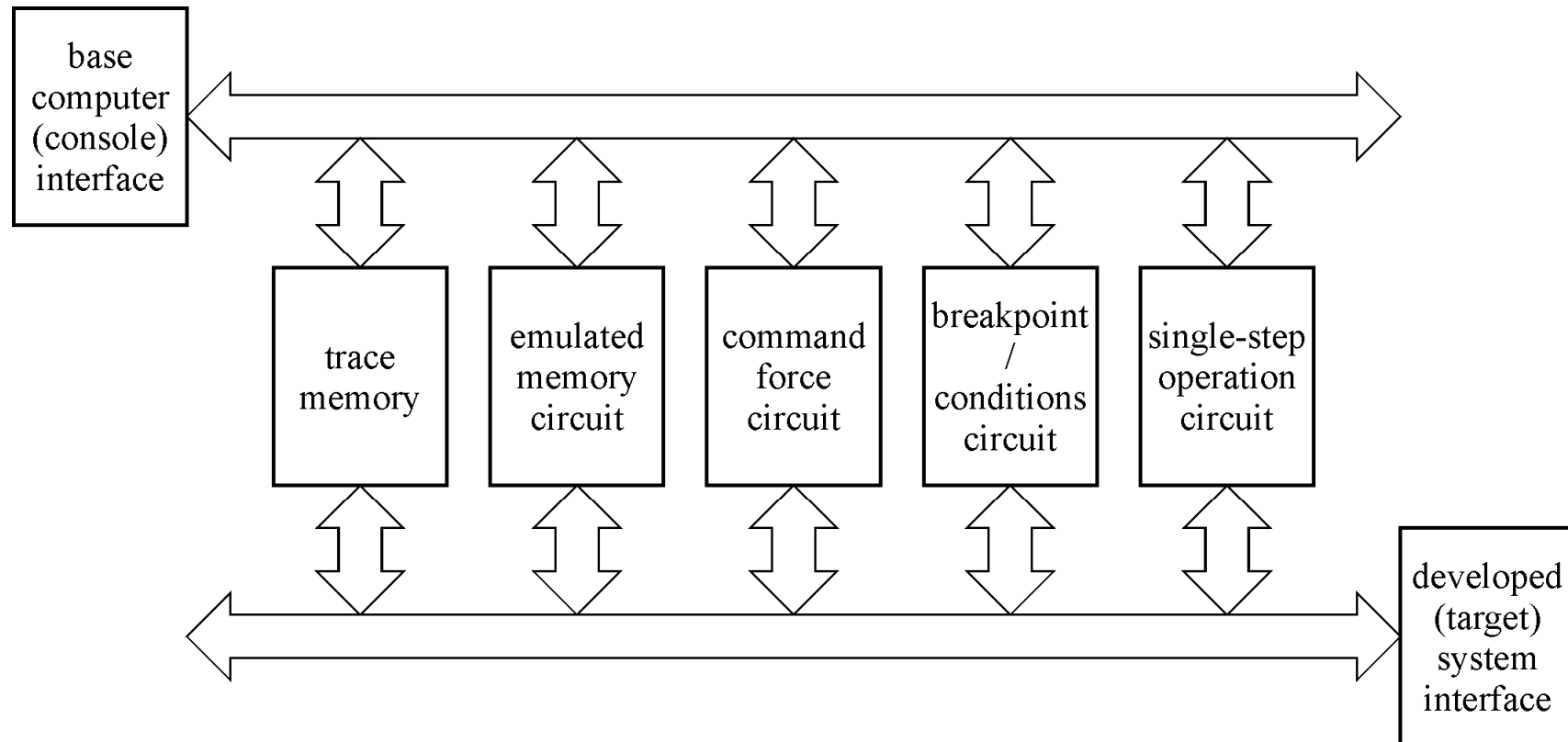
# Development & testing

---

- In-circuit emulator
  - User can „see” the target circuit exactly as the  $\mu\text{p}$  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  $\mu$ p (e.g., Z80, 8086, etc.) – machine cycle extension input (e.g.,  $\overline{\text{Wait}}$ , Ready etc.)
      - Single chip /  $\mu$ c – 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 occurred
    - (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 (+address, 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  $\mu\text{p}$  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  $\mu\text{p}$  after breakpoint occurred
        - » Update  $\mu\text{p}$  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  $\mu\text{p}$  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  $\mu\text{p}/\mu\text{c}$
  - „full” JTAG
    - TDI (*Test Data In*)
    - TDO (*Test Data Out*)
    - TCK (*Test Clock*)
    - TMS (*Test Mode Select*)
    - TRST (*Test Reset*, optional)
  - „compact” JTAG – cJTAG
    - TMSD (*Test Serial Data*)
    - TCKC (*Test Clock*)