
Asembler
2

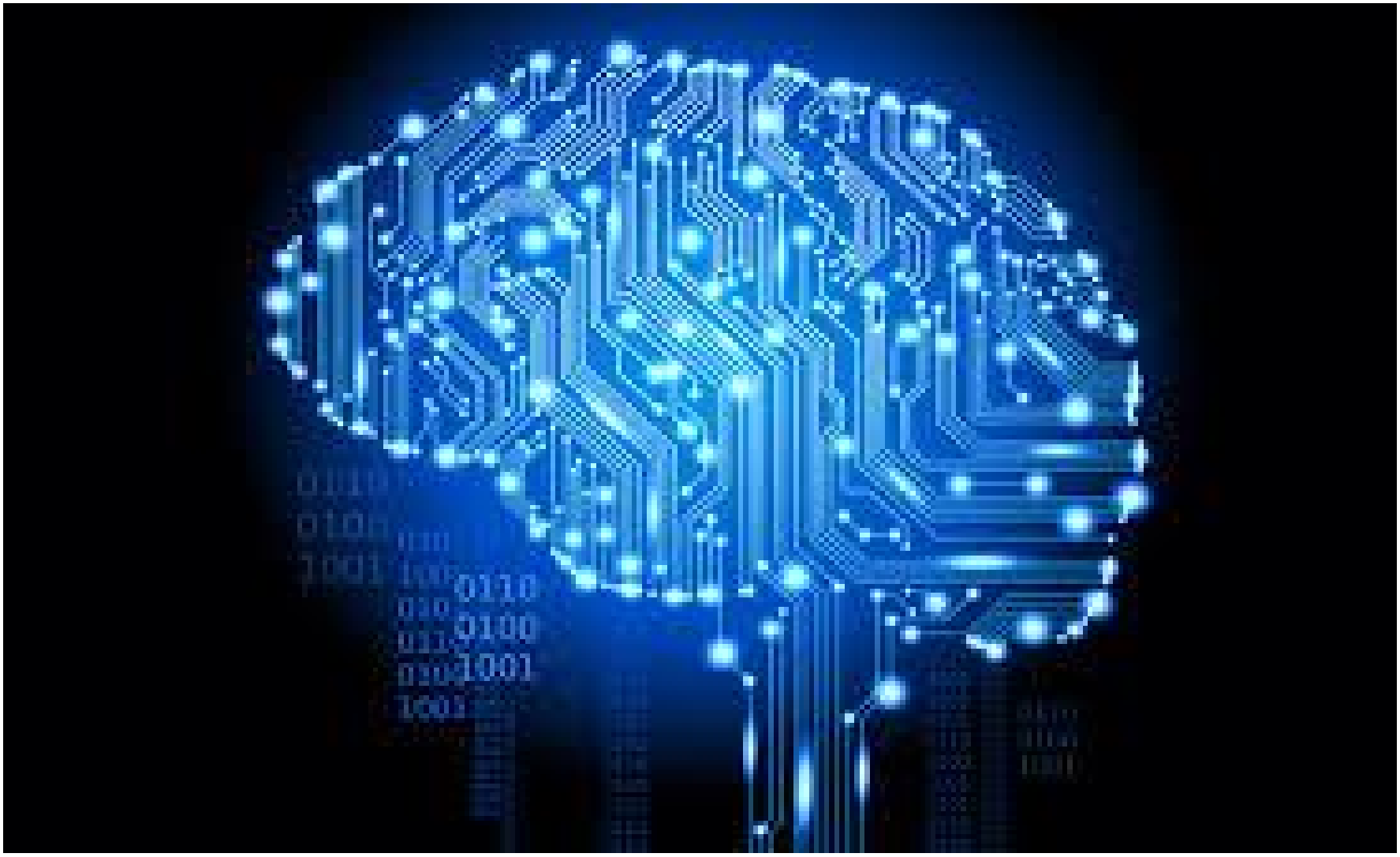
Kiedy asembler

- Gdy potrzebny jest kod działający samodzielnie tzn. bez systemu operacyjnego
- Gdy potrzebny jest kod o krytycznej wydajności
- Gdy kod bezpośrednio współpracuje ze sprzętem
- Gdy kod korzysta z rozkazów niedostępny z poziomu języków wysokiego poziomu
- Gdy zależy nam na ekstremalnie małym rozmiarze kodu
- Gdy nie dysponujemy kompilatorem języka wysokiego poziomu na nowym typie procesora

- Gdy piszemy programy ładujące system operacyjny (bootloader), jadra systemów (kernel)
- Gdy piszemy oprogramowanie o podstawowym znaczeniu dla działania systemu (BIOS)

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Architektura komputera



Architektura komputera

First ever computer ENIAC, 1946

Architektura komputera

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Architektura von Neumana

Architektura von Neumanna – pierwszy rodzaj architektury komputera, opracowanej przez Johna von Neumana, Johna Mauchly'ego oraz Johna Eckerta w 1945 roku.

John von Neuman

Cechą charakterystyczną tej architektury jest to, że dane przechowywane są wspólnie z instrukcjami, co sprawia, że są kodowane w ten sam sposób.

W architekturze tej komputer składa się z czterech głównych komponentów:

•Pamięci komputera przechowującej dane programu oraz instrukcje programu; każda komórka pamięci ma unikatowy identyfikator nazywany jej adresem

•jednostki sterującej odpowiedzialnej za pobieranie danych i instrukcji z pamięci oraz ich sekwencyjne przetwarzanie

•Jednostki arytmetyczno-logicznej odpowiedzialnej za wykonywanie podstawowych operacji arytmetycznych.

•Urządzeń wejścia/wyjścia służących do interakcji z operatorem

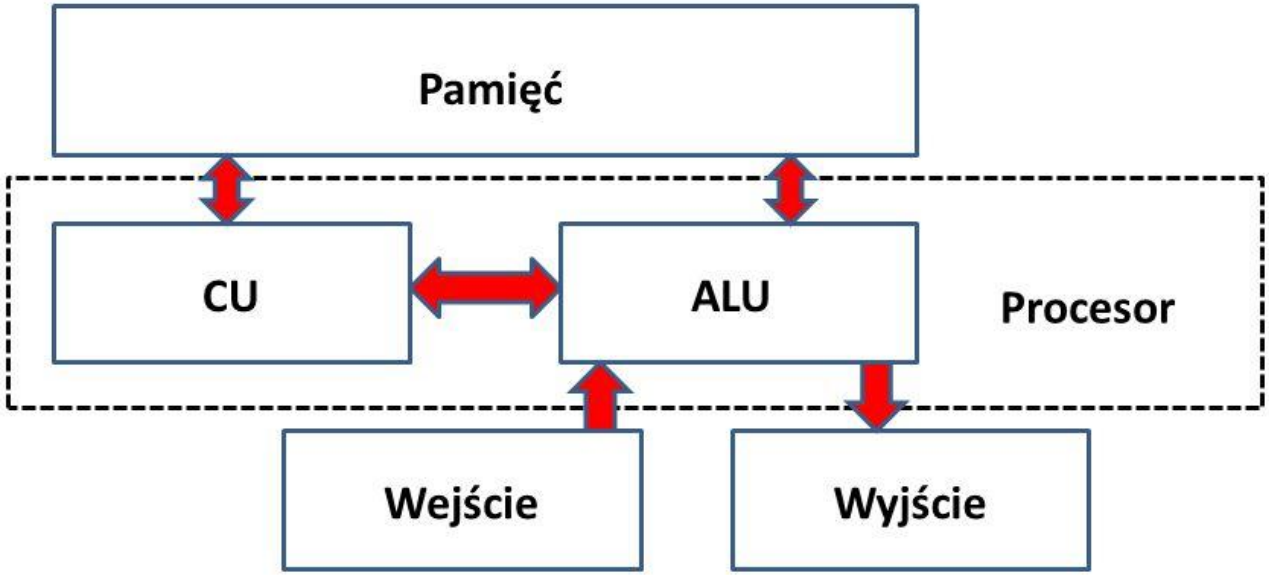
Jednostka sterująca wraz z jednostką arytmetyczno-logiczną tworzą procesor.

Architektura von Neumana

Architektura von Neumanna

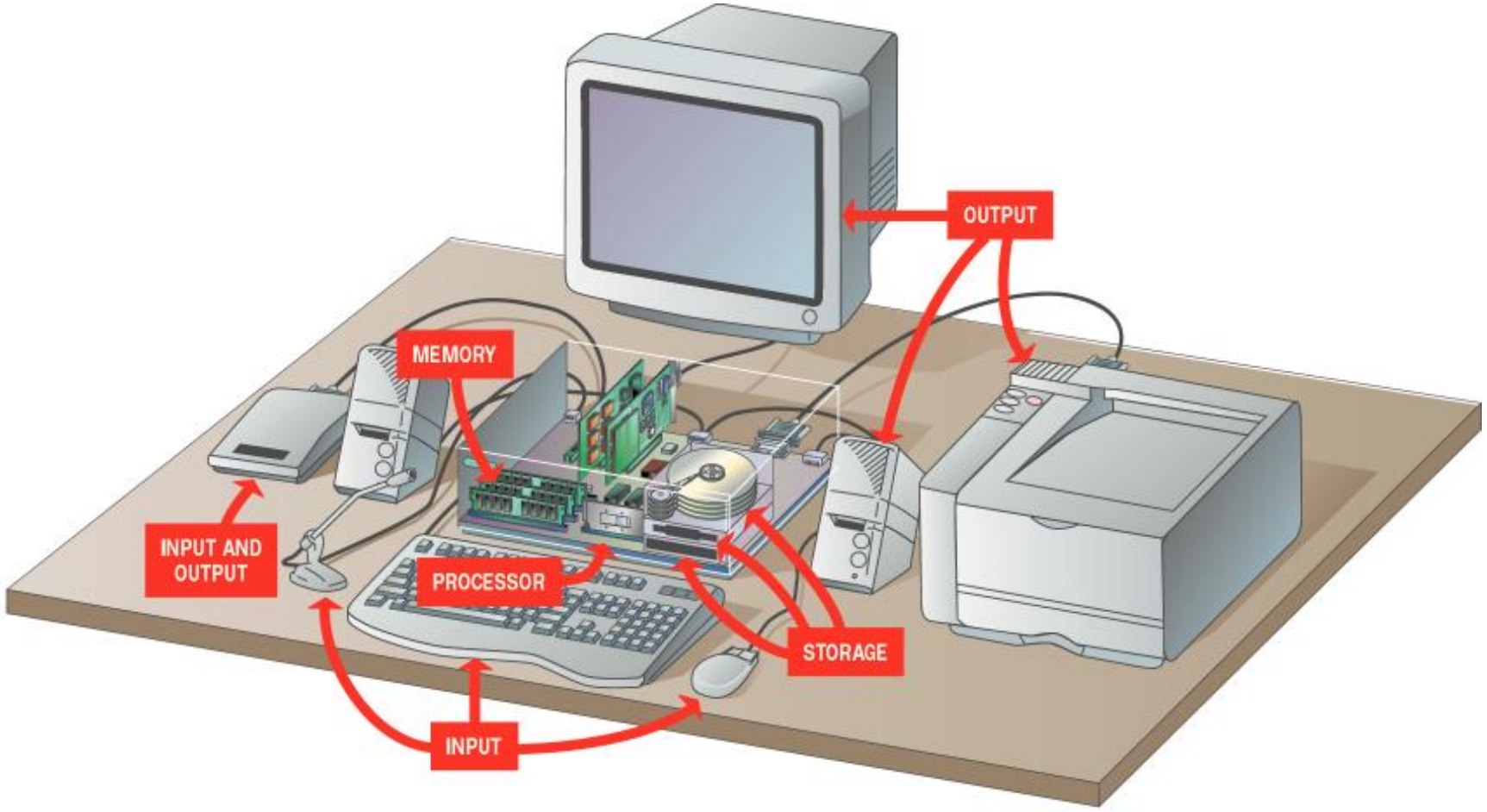
Komputer o architekturze von Neumanna składa się z trzech bloków:

- procesora (jednostki ALU i CU)
- pamięci
- urządzeń wejścia/wyjścia



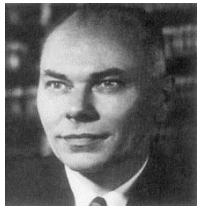
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Komputer



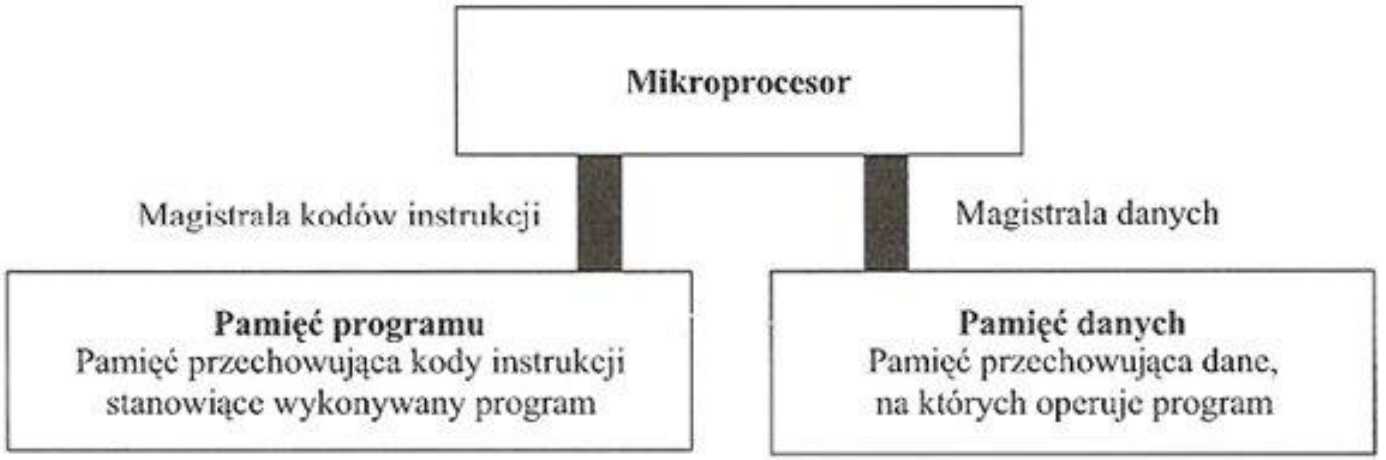
Architektura harwardzka

Architektura harwardzka



Howard Hathaway Aiken

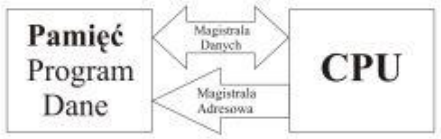
System o architekturze harwardzkiej posiada dwie pamięci. Jedna jest przeznaczona na rozkazy, natomiast druga na dane. Są one połączone z procesorem osobnymi magistralami. Dane z pamięci danych i pamięci programu mogą być odczytywane jednocześnie. Dzięki temu systemy o tej architekturze są szybsze od systemów o architekturze von Neumanna.



Architektura superharwardzka

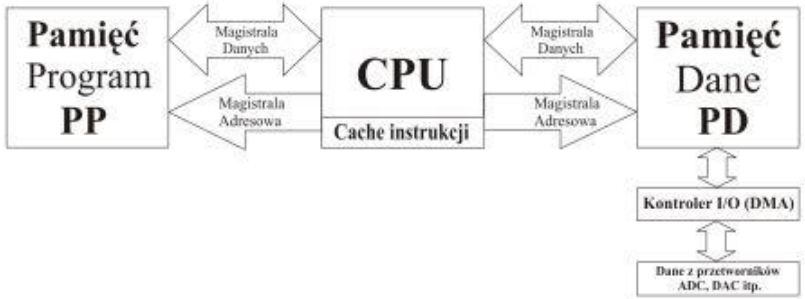
Koncepcja nowej architektury

- **Architektura vin Neumanna.**
(optym dla prostych operacji z danymi)

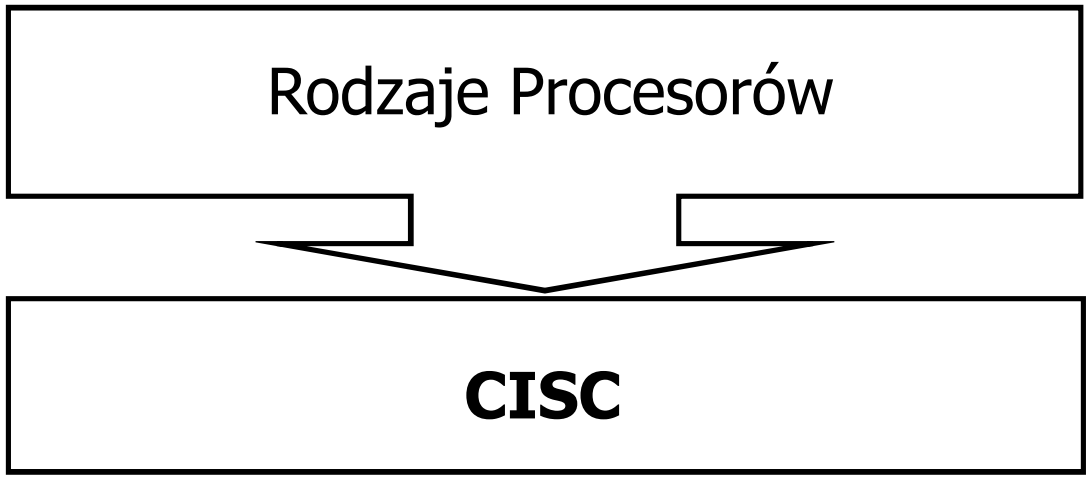


- **Architektura harwardzka** (możliwy jednoczesny dostęp do obu pamięci)

- **Architektura superharwardzka**
(poza podwójną pamięcią *Cache instrukcji* dla danych typu instrukcja-adres i DMA z pominięciem rejestrów)



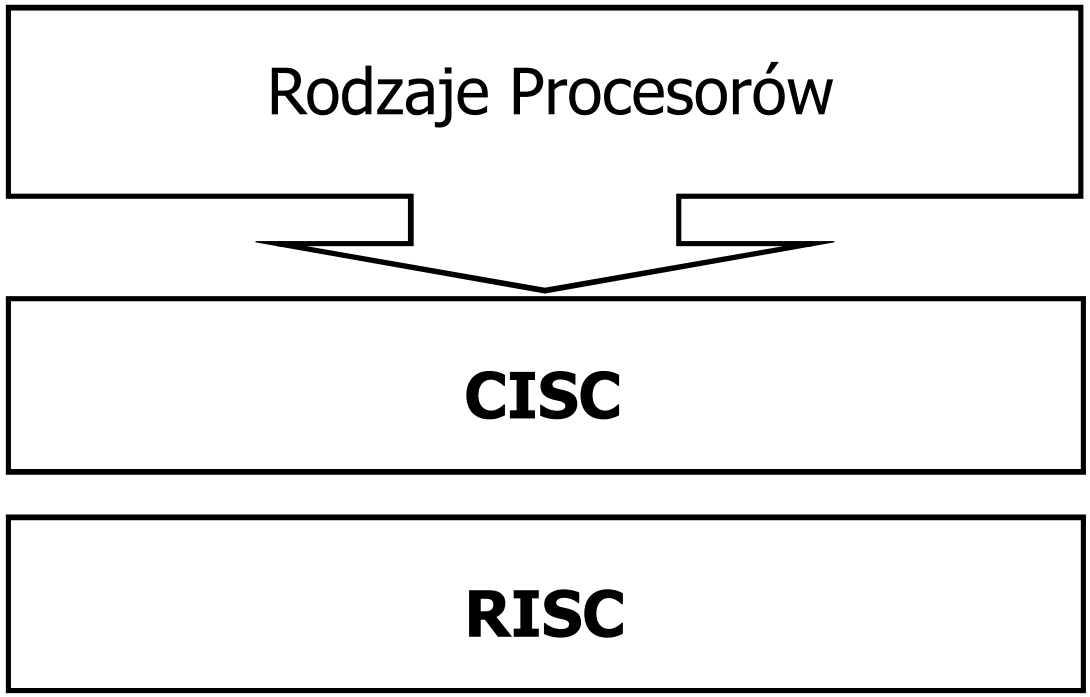
Architektury Procesorów



Complex Instruction Set Computer

Procesory charakteryzujące się dużą ilością rozbudowanych rozkazów np. ładowanie z pamięci, operacje arytmetyczno-logiczne itp. Współczesne mikrokomputery wyposażone są głownie w procesory typu CISC.

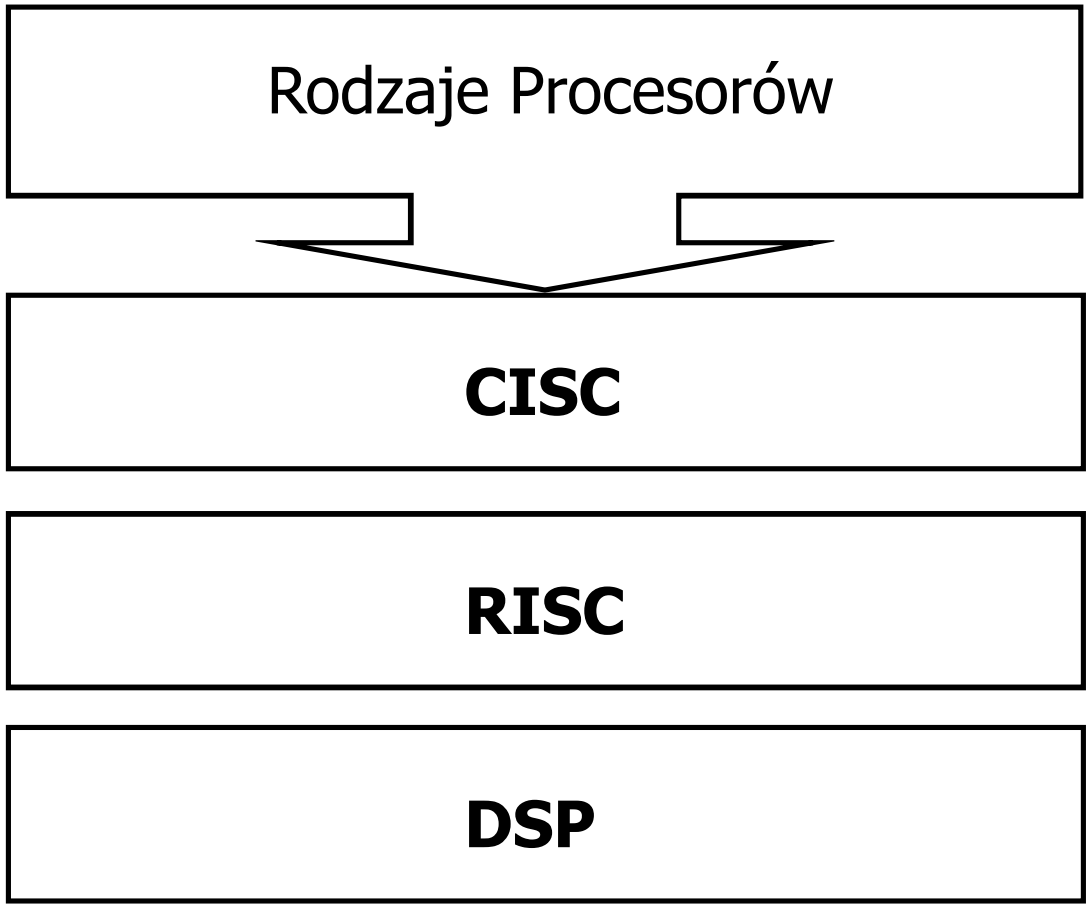
Architektury Procesorów



Reduced Instruction Set Computer

procesory o zmniejszonej liczbie instrukcji. Zapewniają dużo większą szybkość i efektywność działania kosztem trudniejszego programowania. Rozdzielenie pamięci danych od pamięci programu.
Architektura harwardzka

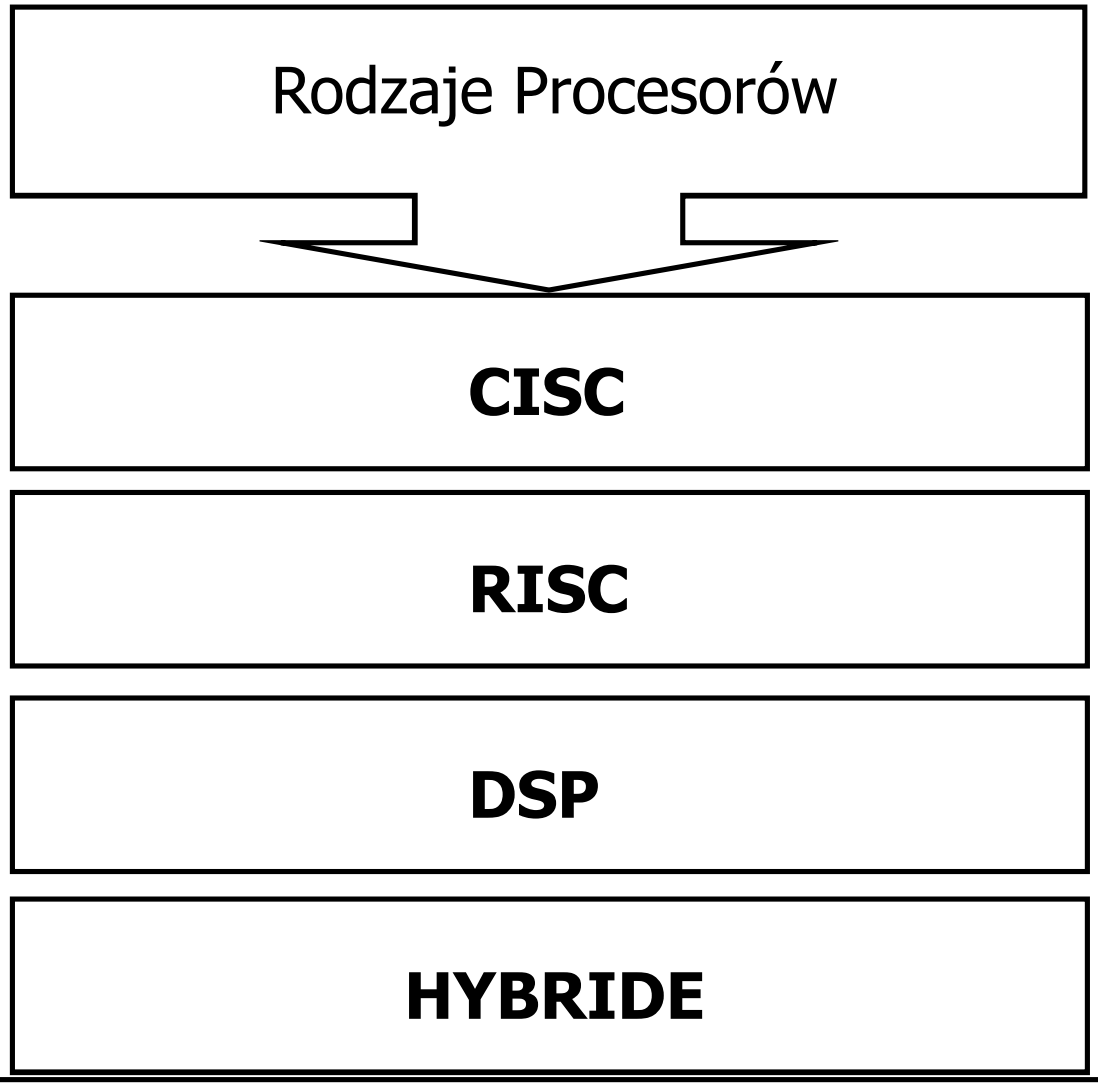
Architektury Procesorów



Digital Signal Processor

procesory sygnałowe zaprojektowane do przetwarzania sygnałów. Znalazły zastosowanie w modemach, kamerach cyfrowych, kartach graficznych, urządzeniach muzycznych.

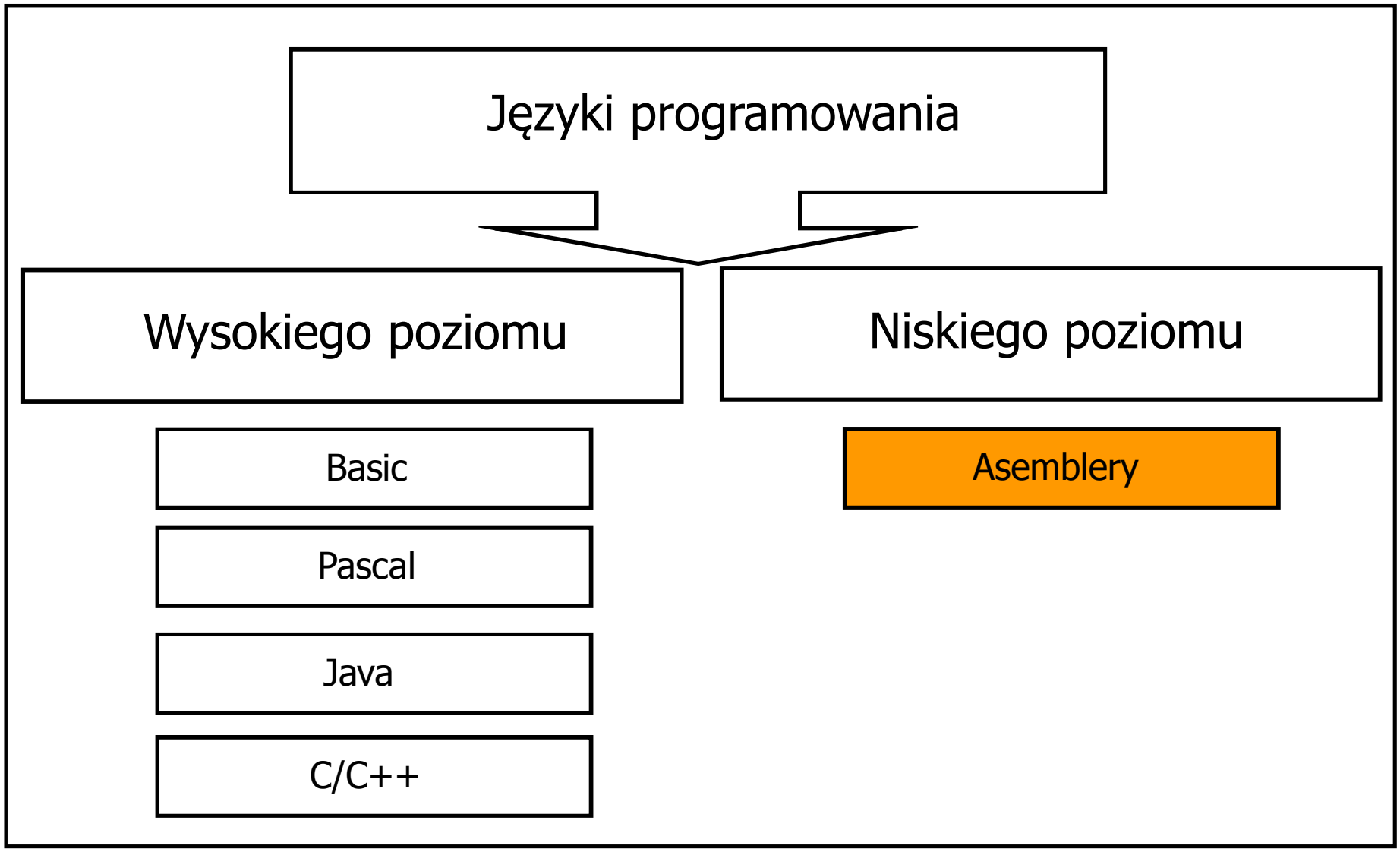
Architektury Procesorów



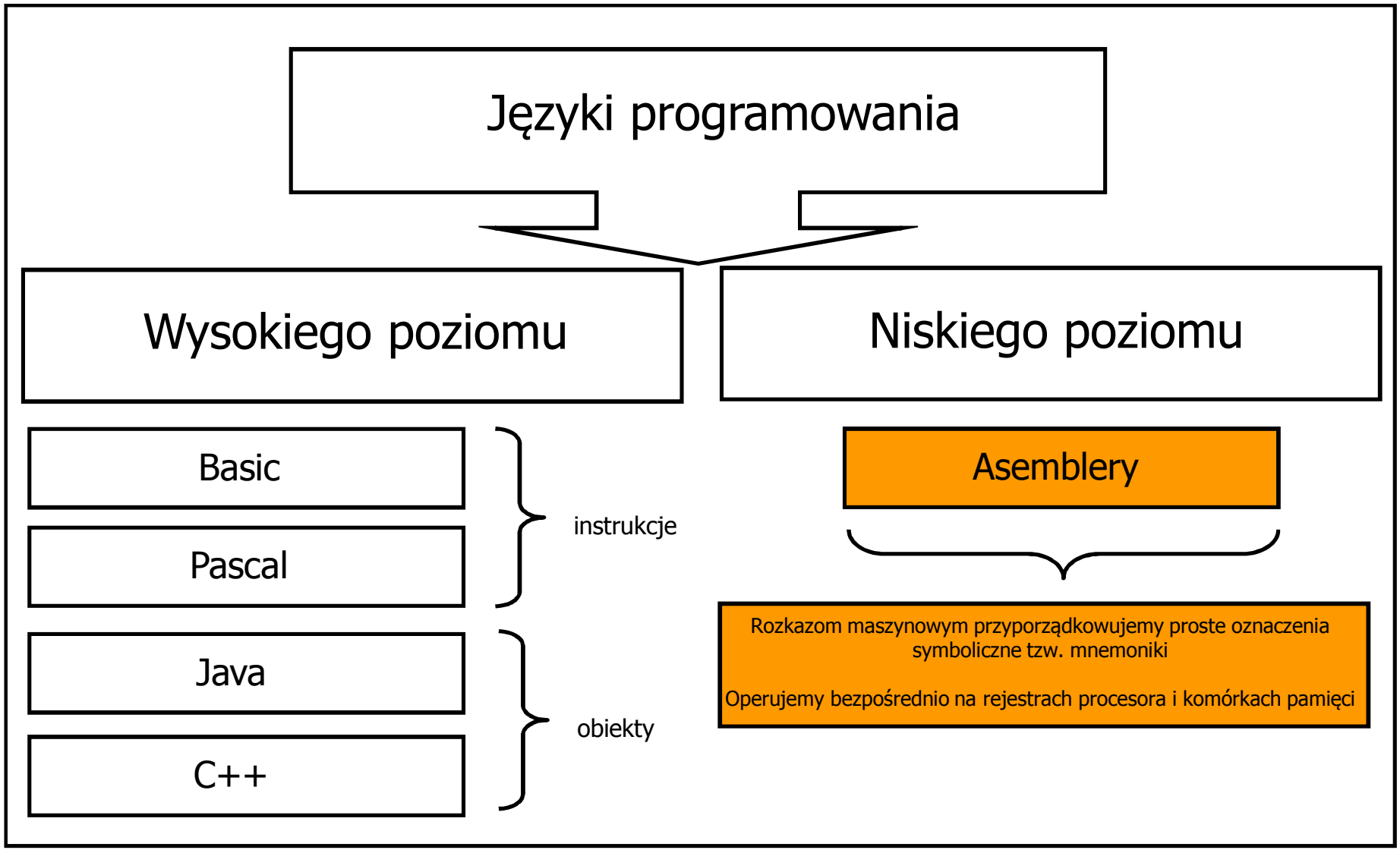
Hybride

procesory będące kombinacją CISC, RISC i DSP.

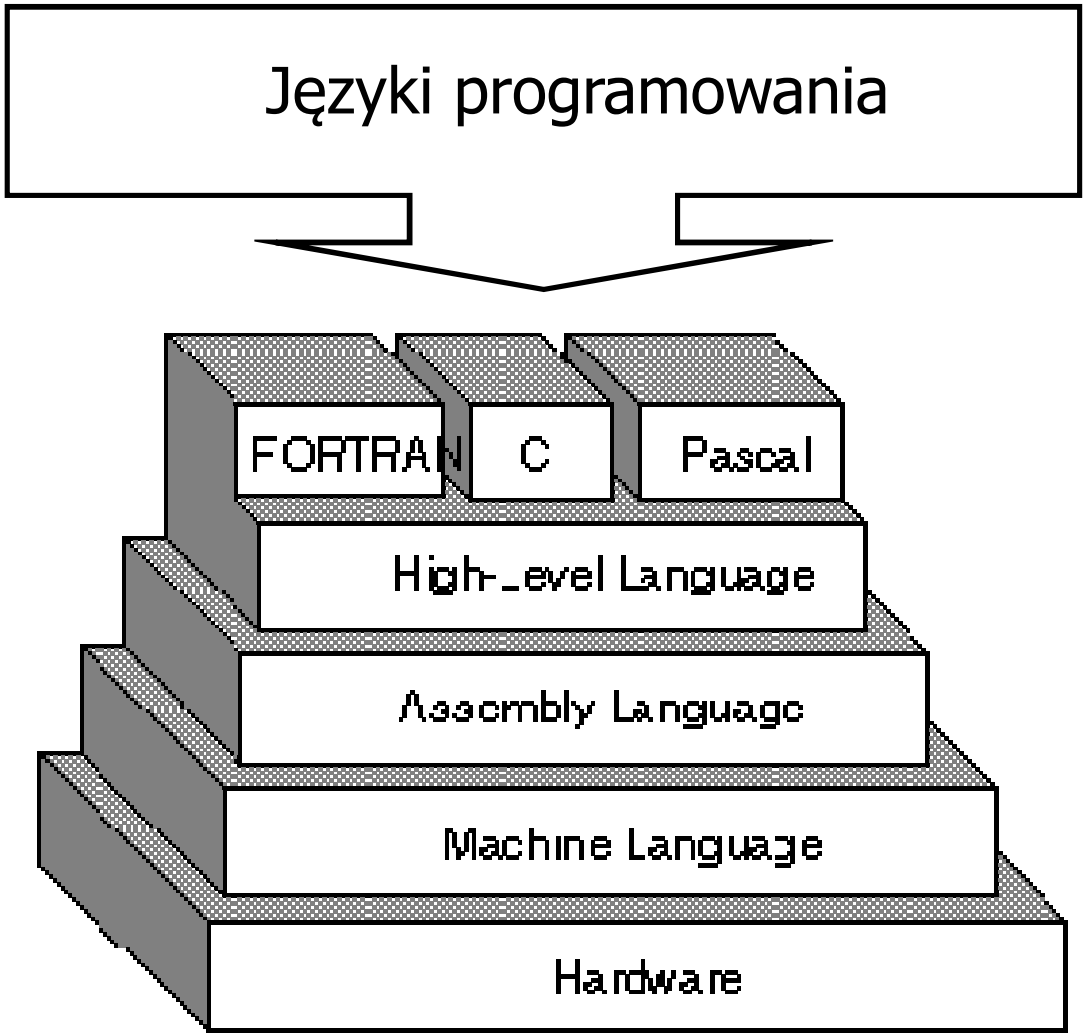
Języki Programowania



Języki Programowania



Języki Programowania



Języki Programowania

Cechy języka niskiego poziomu

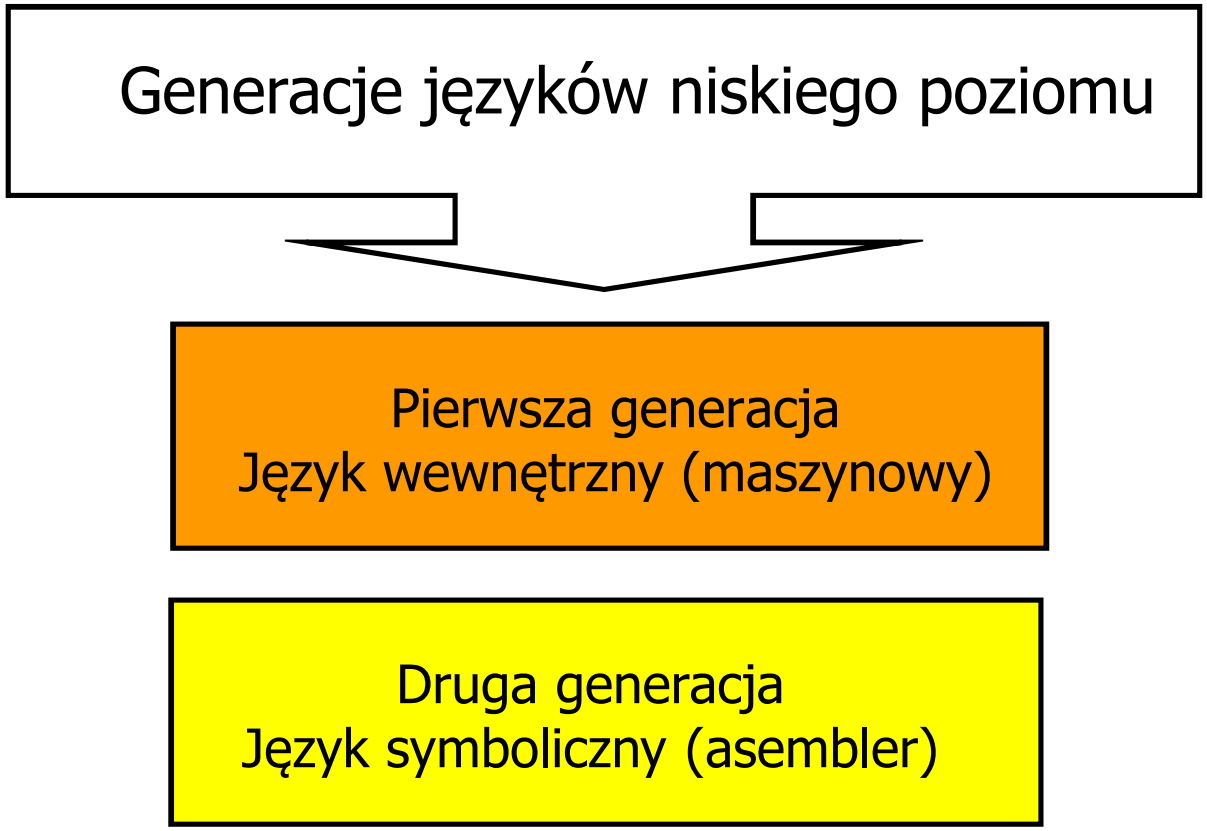
Asemblery

Rozkazom maszynowym przyporządkowujemy proste oznaczenia symboliczne tzw. mnemoniki

Operujemy bezpośrednio na rejestrach procesora i komórkach pamięci

1. Wymaga znajomości architektury procesora
2. Umożliwia dostęp do rejestrów procesora, portów I/O, pamięci systemu
3. Pozwala na optymalizację kodu z punktu widzenia szybkości wykonania programu i jego objętości
4. Doświadczenie w programowaniu i elektronice
5. Czasochłonność, błędy

Języki programowania niskiego poziomu



Języki programowania niskiego poziomu

Język wewnętrzny (maszynowy)

To zbiór operacji wykonywanych przez procesor.
Operacje te wymuszamy za pomocą rozkazów.
Rozkazy są kodowane za pomocą liczb
najczęściej w formacie binarnym lub heksadecymalnym

Ciąg rozkazów stanowi tzw. **kod maszynowy**

Języki programowania niskiego poziomu

Język wewnętrzny (maszynowy)

Podstawowa wada – jest trudno zrozumiały

BA 12 01 B4 09 B9 0A 00 CD 21 E2 FC B4 4C B0 00
CD 21 57 69 74 61 6A 20 73 77 69 65 63 69 65 0D
0A 24

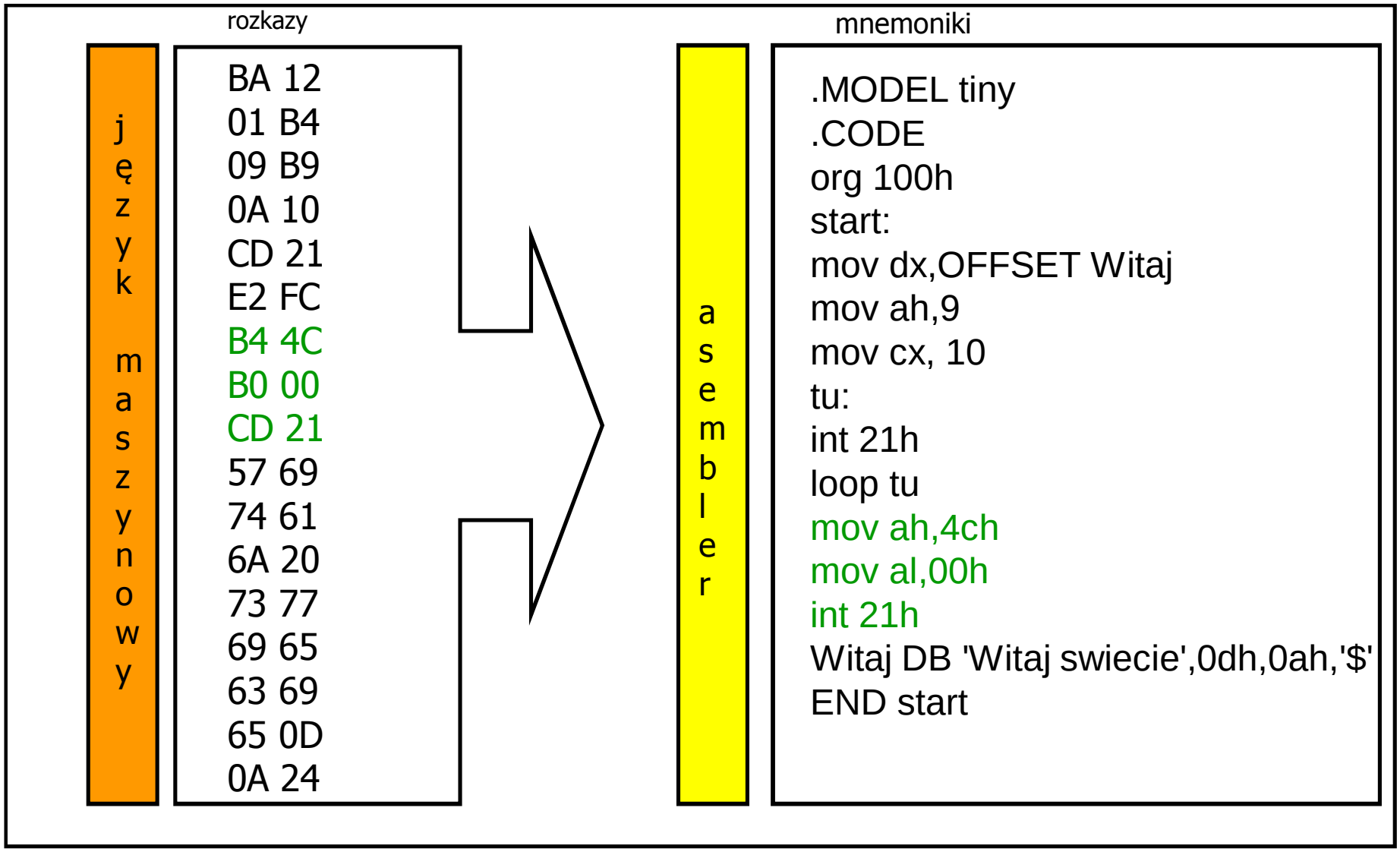
Języki programowania niskiego poziomu

Język symboliczny (asembler)

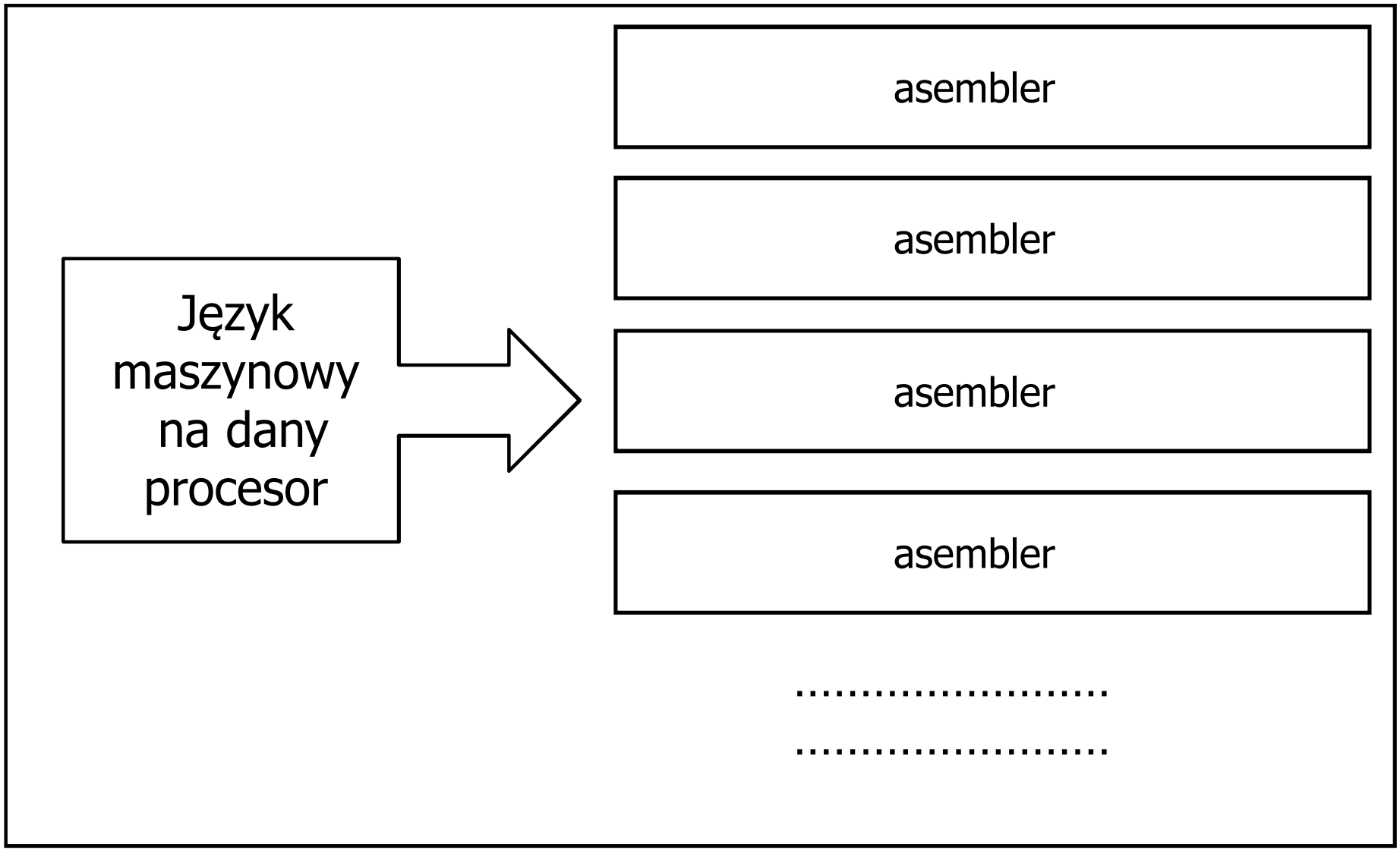
Asemblery powstały na bazie języka maszynowego poprzez zastąpienie każdego rozkazu nazwami symbolicznymi tzw. mnemonikami

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Języki programowania niskiego poziomu



Języki programowania niskiego poziomu



Języki programowania

BASIC 21 kB

```
10 FOR i = 0 TO 20 STEP 1
20 PRINT "witaj swiecie"
30 NEXT i
```

C 7 kB

```
#include <stdio.h>
int main(argc, argv)
int argc;
char ** argv;
{
int i;
for(i = 0; i < 20; i++)
printf("witaj swiecie\n");
getchar();
return 0;
}
```

C++ ok. 7 kB

```
#include <iostream.h>
int main()
{
for(int i = 0; i < 20; i++)
cout << "witaj swiecie" << endl;
char c;
cin >> c;
return 0;
}
```

Języki programowania

ASEMBLER 34 B

```
.MODEL tiny
.CODE
org 100h
start:
mov dx,OFFSET Witaj
mov ah,9
mov cx, 10
tu:
int 21h
loop tu
mov ah,4ch
mov al,0
int 21h
Witaj DB 'Witaj swiecie',0dh,0ah,'$'
END start
```

język maszynowy

```
BA 12
01 B4
09 B9
0A 10
CD 21
E2 FC
B4 4C
B0 00
CD 21
57 69
74 61
6A 20
73 77
69 65
63 69
65 0D
0A 24
```

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Wybór Procesora – rodzina mikrokontrolerów AVR firmy ATMEL

Atmel

Rdzeń rodziny AVR oparty jest na architekturze RISC

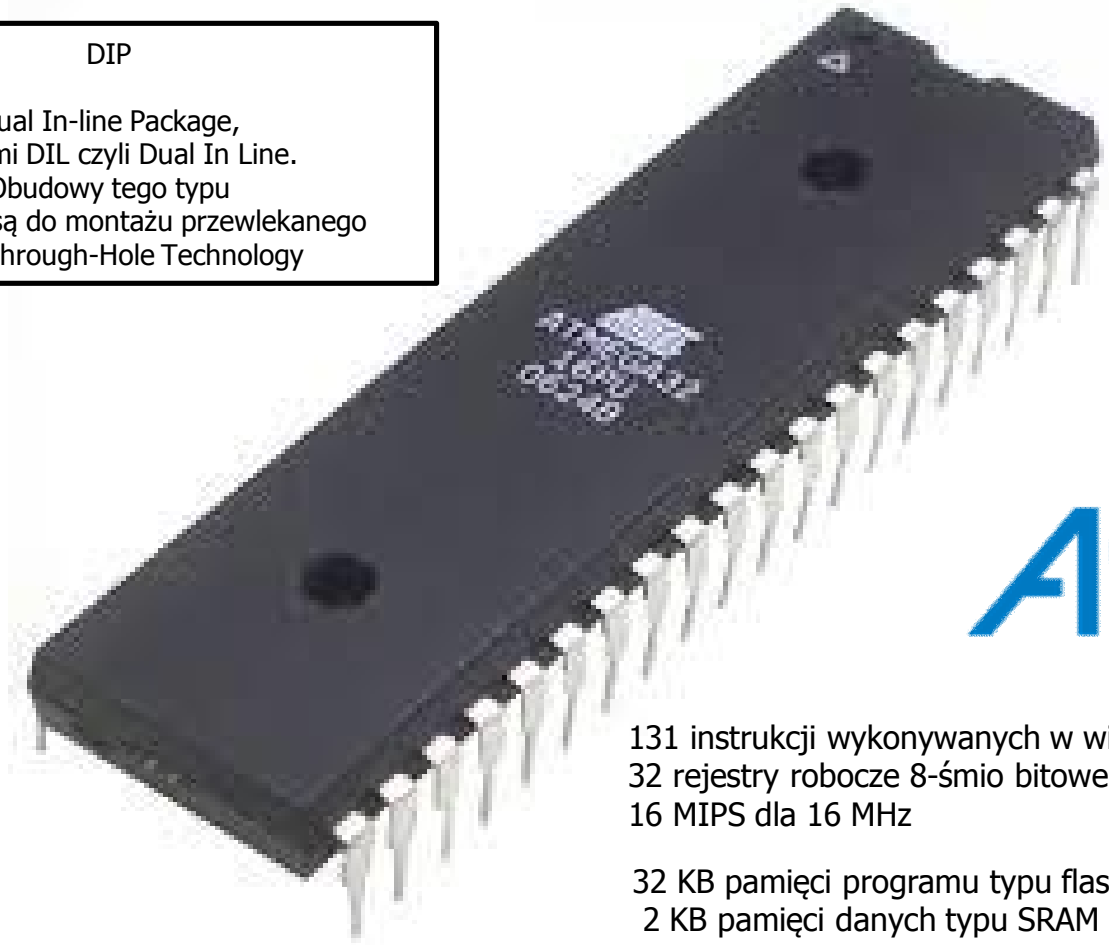
ATMEL		nazwa firmy produkującej rodzinę mikrokontrolerów AVR
AVR		Advanced Virtual RISC
ATtiny	–	układy z rdzeniem AVR w mniejszych obudowach z małą liczbą peryferii przeznaczone do prostych zadań
ATmega	–	układy z rdzeniem AVR charakteryzujące się bogatym wyposażeniem takim jak kilka linii we/wy, przetworniki A/C, C/A, timery

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Mikrokontroler ATMEGA 32

DIP

Dual In-line Package,
czasami DIL czyli Dual In Line.
Obudowy tego typu
stosowane są do montażu przewlekanego
THT Through-Hole Technology



Atmel

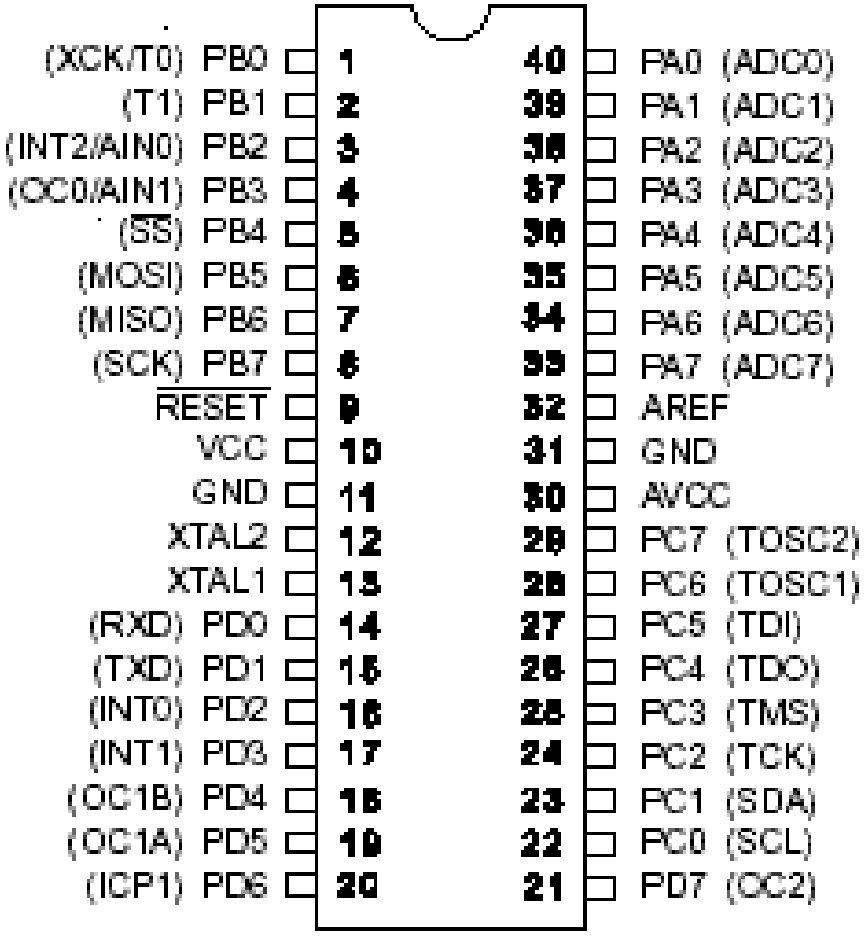
131 instrukcji wykonywanych w większości w jednym cyklu zegara
32 rejestry robocze 8-śmio bitowe
16 MIPS dla 16 MHz

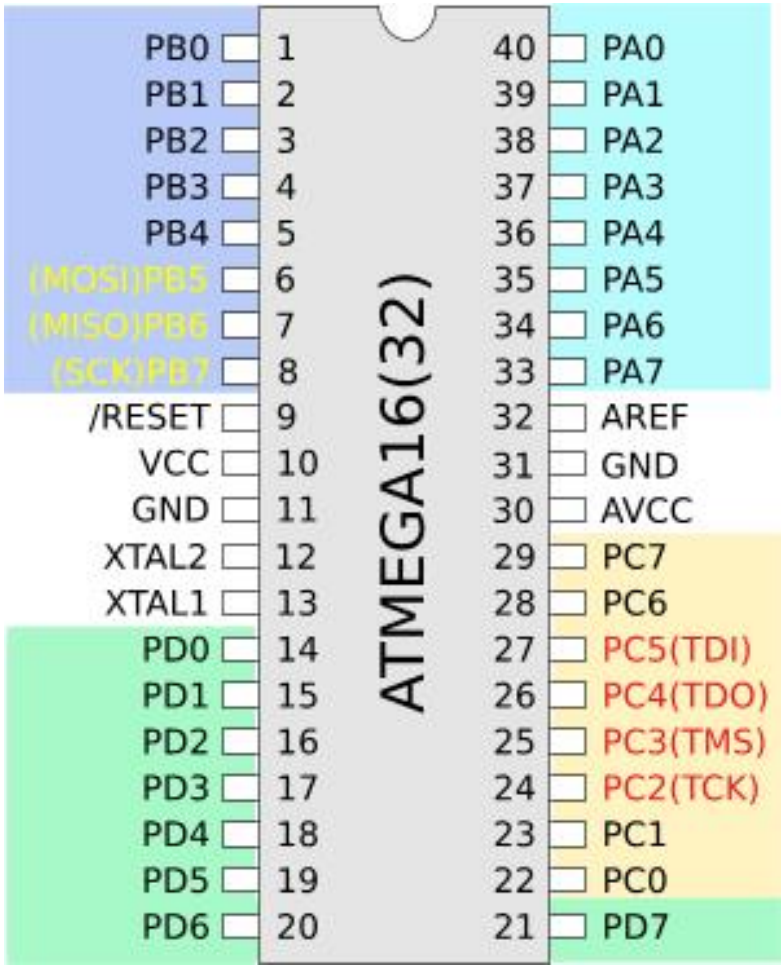
32 KB pamięci programu typu flash - programowanie metodą ISP
2 KB pamięci danych typu SRAM
1 KB EEPROM

Flash 10 000 write/erase
EEPROM 100 000 write/erase

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Mikrokontroler ATMEGA 32





WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

8-bit AVR® Microcontroller with 32KBytes In-System Programmable Flash

ATmega32
ATmega32L

- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-chip 2-cycle Multiplier

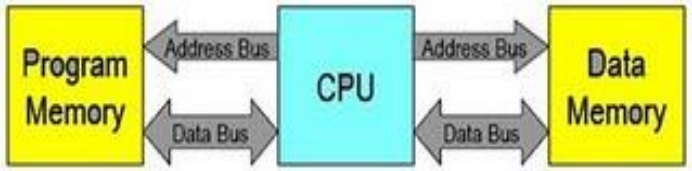
- High Endurance Non-volatile Memory segments
  - 32Kbytes of In-System Self-programmable Flash program memory
  - 1024Bytes EEPROM
  - 2Kbytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security

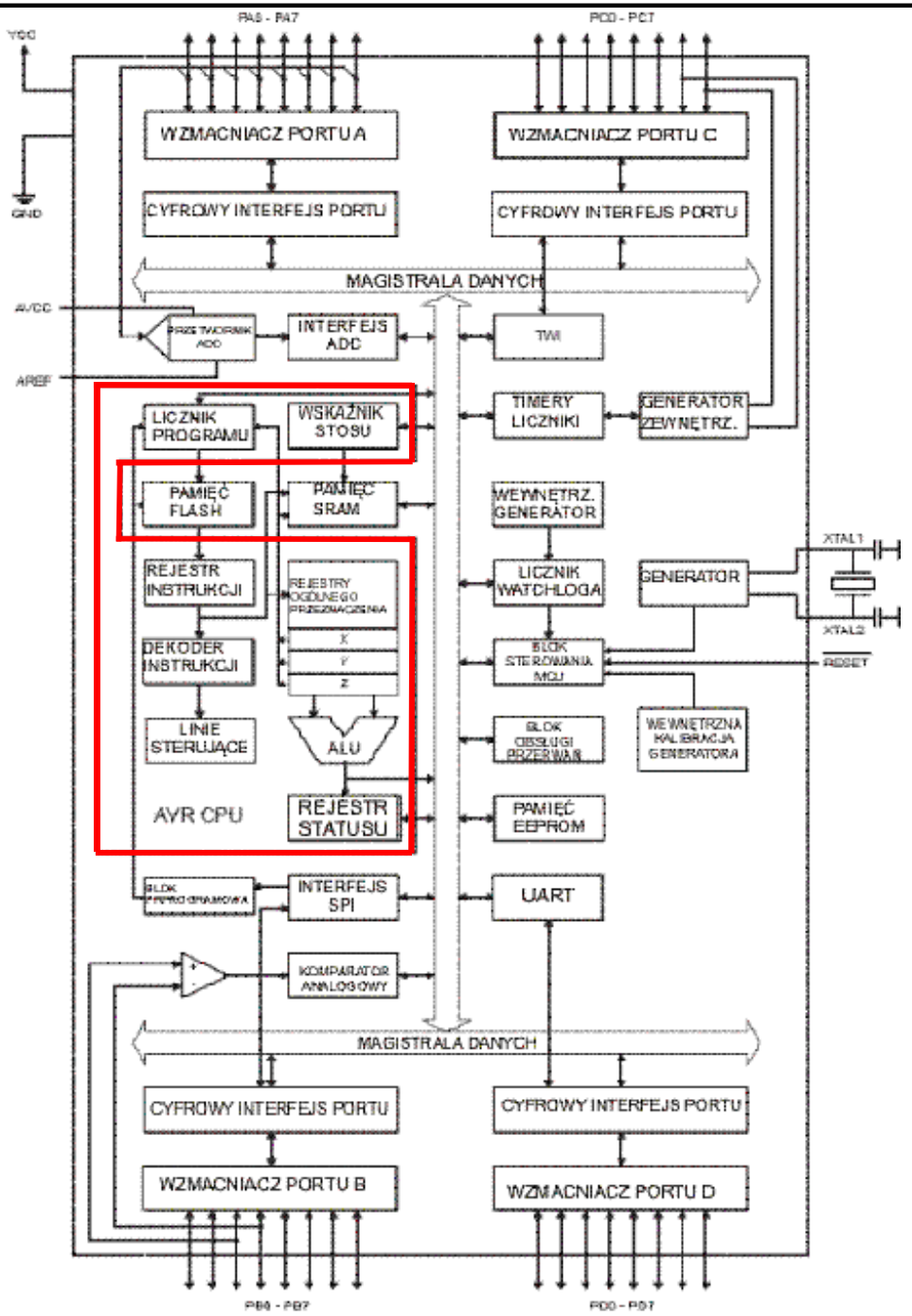
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    - 8 Single-ended Channels
    - 7 Differential Channels in TQFP Package Only
    - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

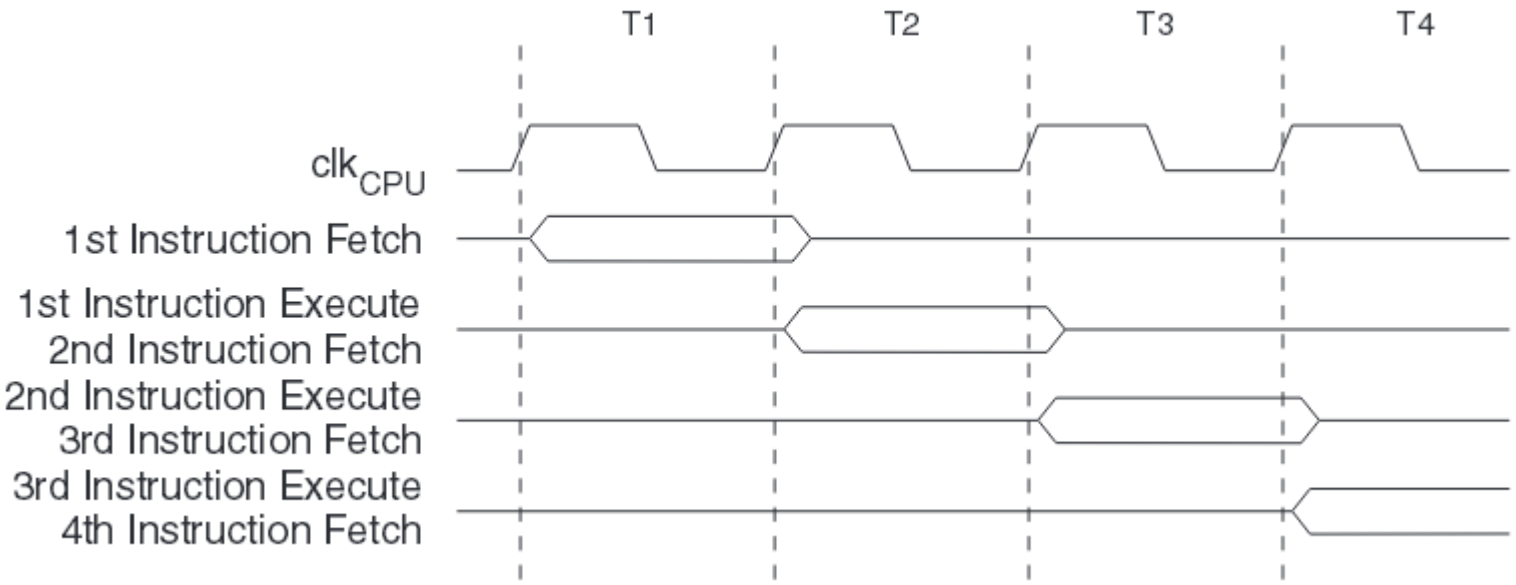
Mikrokontroler ATMEGA 32

Rdzeń mikrokontrolera AVR





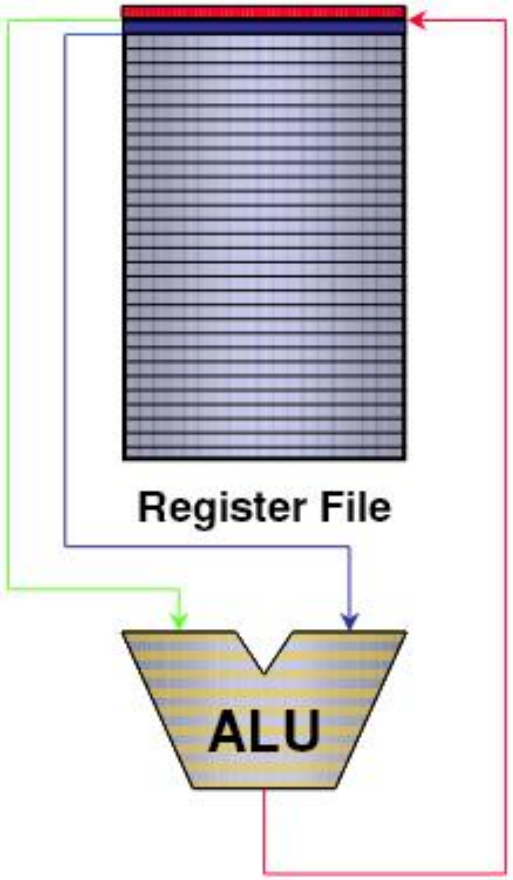
Przetwarzanie potokowe (pipelining)



W AVR zastosowano przetwarzanie potokowe które polega na jednoczesnym wykonywaniu danej instrukcji i pobieraniu instrukcji następnej.
Ma to miejsce w jednym cyklu zegarowym.
AVR są układami 8 bitowymi jednak ich słowo rozkazowe ma 16 bitów.

Dla zegara 1MHz można osiągnąć 1 MIPS

Jednostka arytmetyczno-logiczna



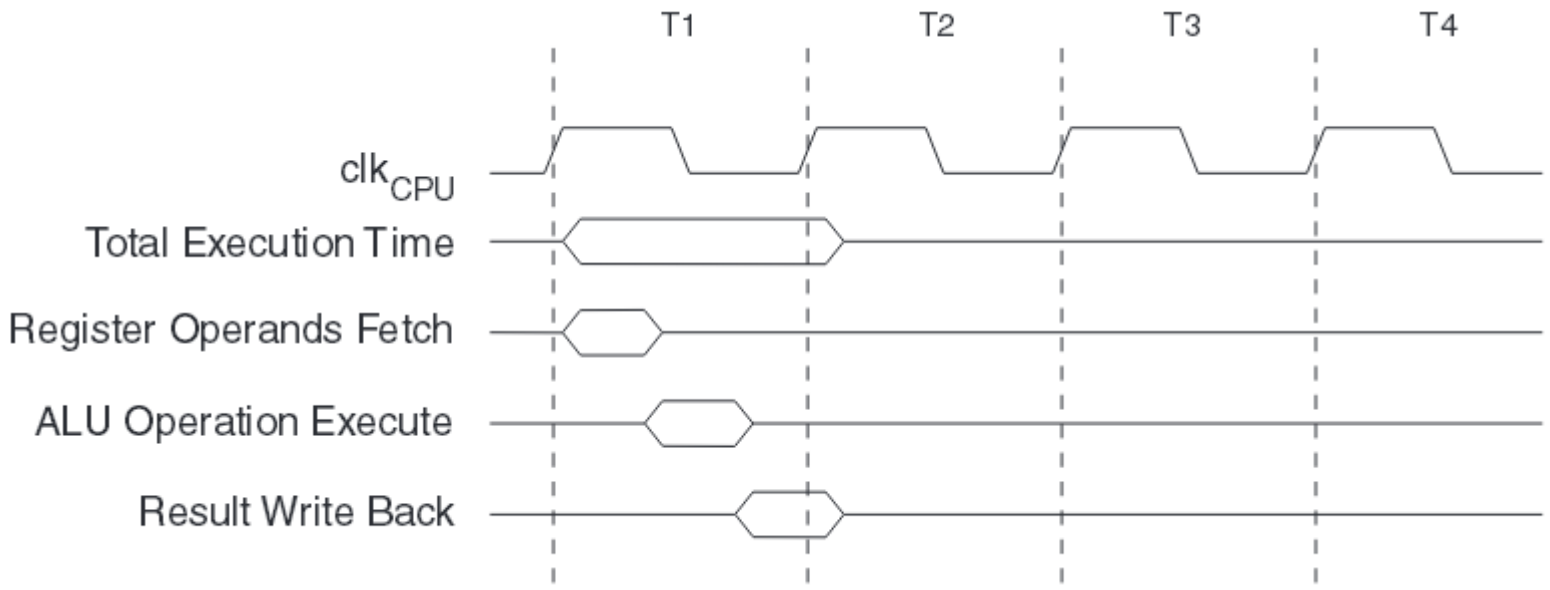
ALU- Arithmetic Logic Unit

Jest bezpośrednio połączona z 32 ośmiobitowymi rejestrami

Wykonuje operacje w jednym cyklu zegarowym:

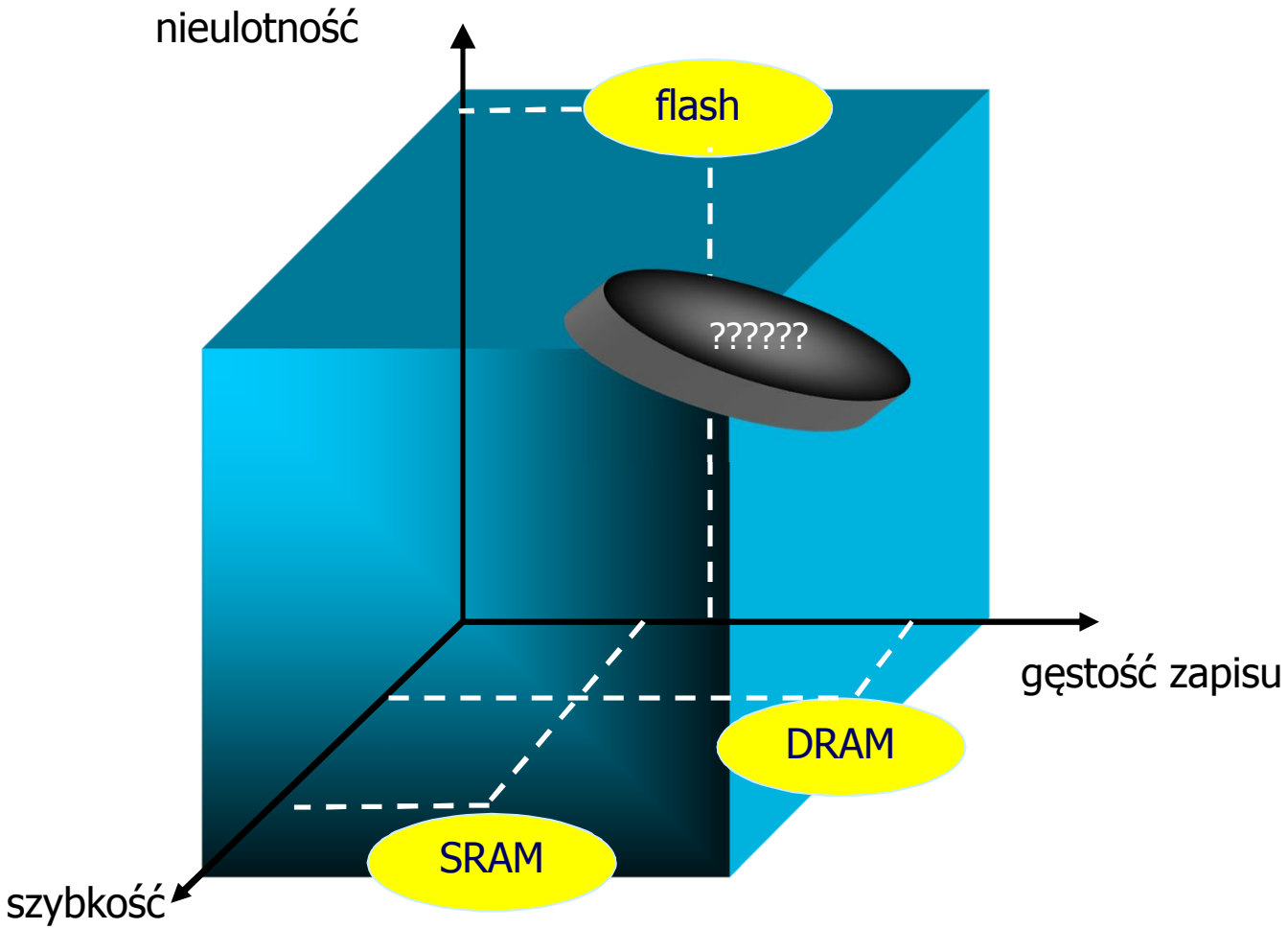
- Arytmetyczne
- Logiczne
- Funkcje bitowe

Pojedynczy cykl pracy ALU



W pojedynczym cyklu zegara ALU wykonujemy operację na 2 rejestrach (źródło, przeznaczenie) i wynik przesyłamy do rejestru przeznaczenia

Pamięć SRAM/DRAM/Flash



WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Rodzaje Pamięci

Układy AVR mają architekturę harwardzką tzn. rozdzielono w nich obszary adresowe pamięci programu i pamięci danych

Pamięć Danych SRAM

Obszar pamięci danych w układach AVR połączono z rejestrami roboczymi i rejestrami funkcyjnymi (przestrzeń we/wy) tworząc wspólna przestrzeń adresową

Jest to pomięć typu SRAM
Static Random Acces Memory

2 Kbytes

Pamięć Programu Flash ROM

Obszar pamięci programu w układach AVR może być traktowany jako jednolita przestrzeń adresowa pod warunkiem, że nie używamy *boot loadera*.

Jest to pomięć typu Flash
32 Kbytes

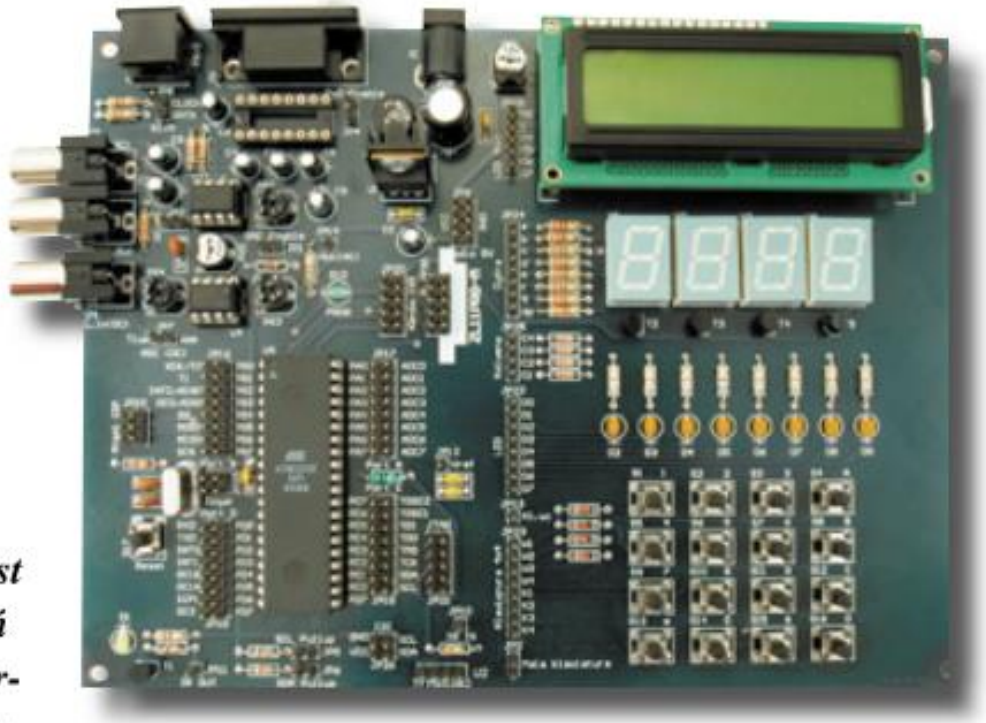
Programowanie pamięci Flash w systemie ISP (*In System Programming*)

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Zestaw dydaktyczny ZL3AVR

ZL3AVR

Zestaw startowy dla mikrokontrolerów AVR – ATmega



Zestaw startowy ZL3AVR opracowano z myślą o Czytelnikach książki „Mikrokontrolery AVR ATmega w praktyce". Jest on sprzętową bazą dla wszystkich ćwiczeń opisanych w tej książce, ale dzięki uniwersalnej budowie może być stosowany także podczas dowolnych innych prac ewaluacyjnych.

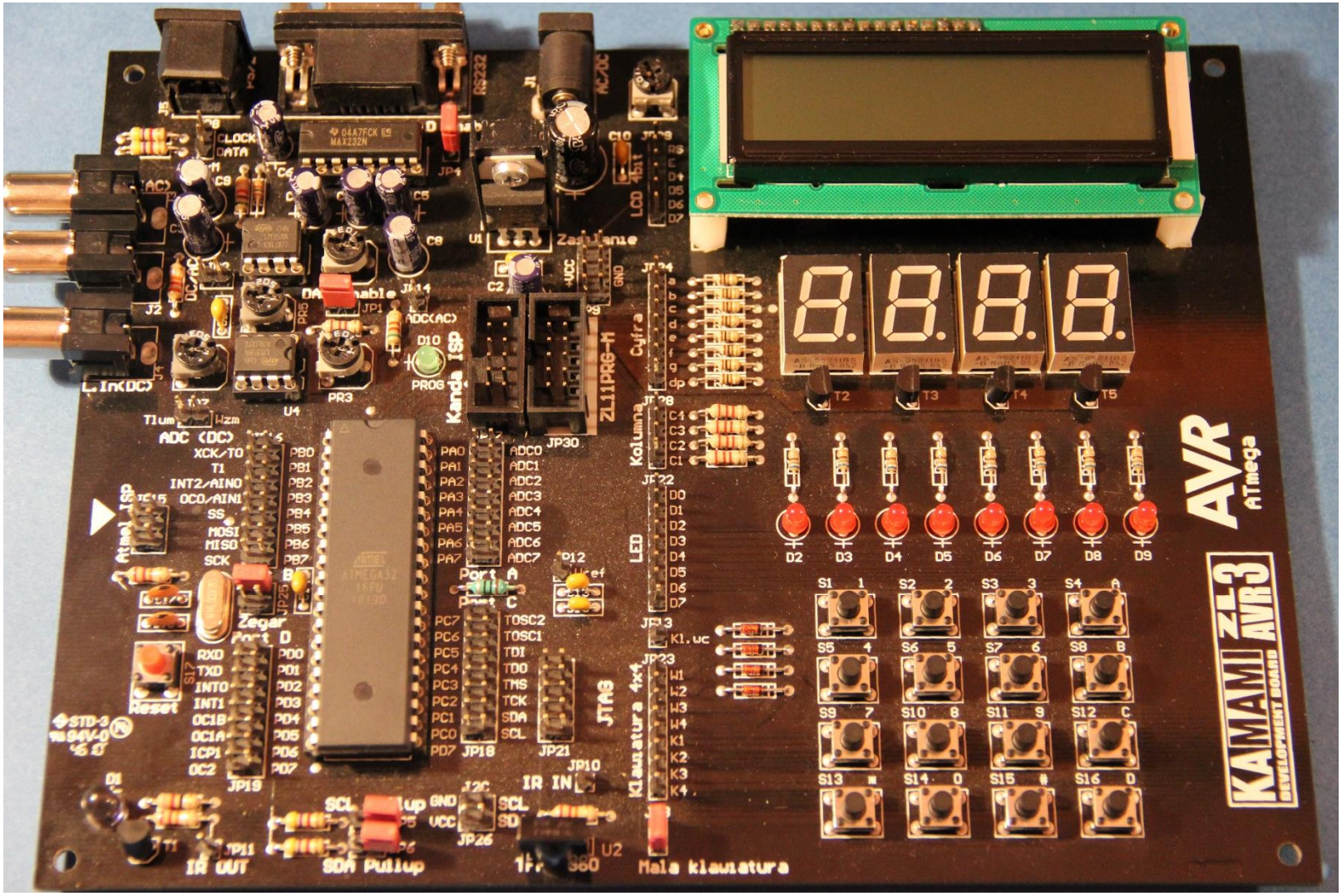
Zestaw ZL3AVR

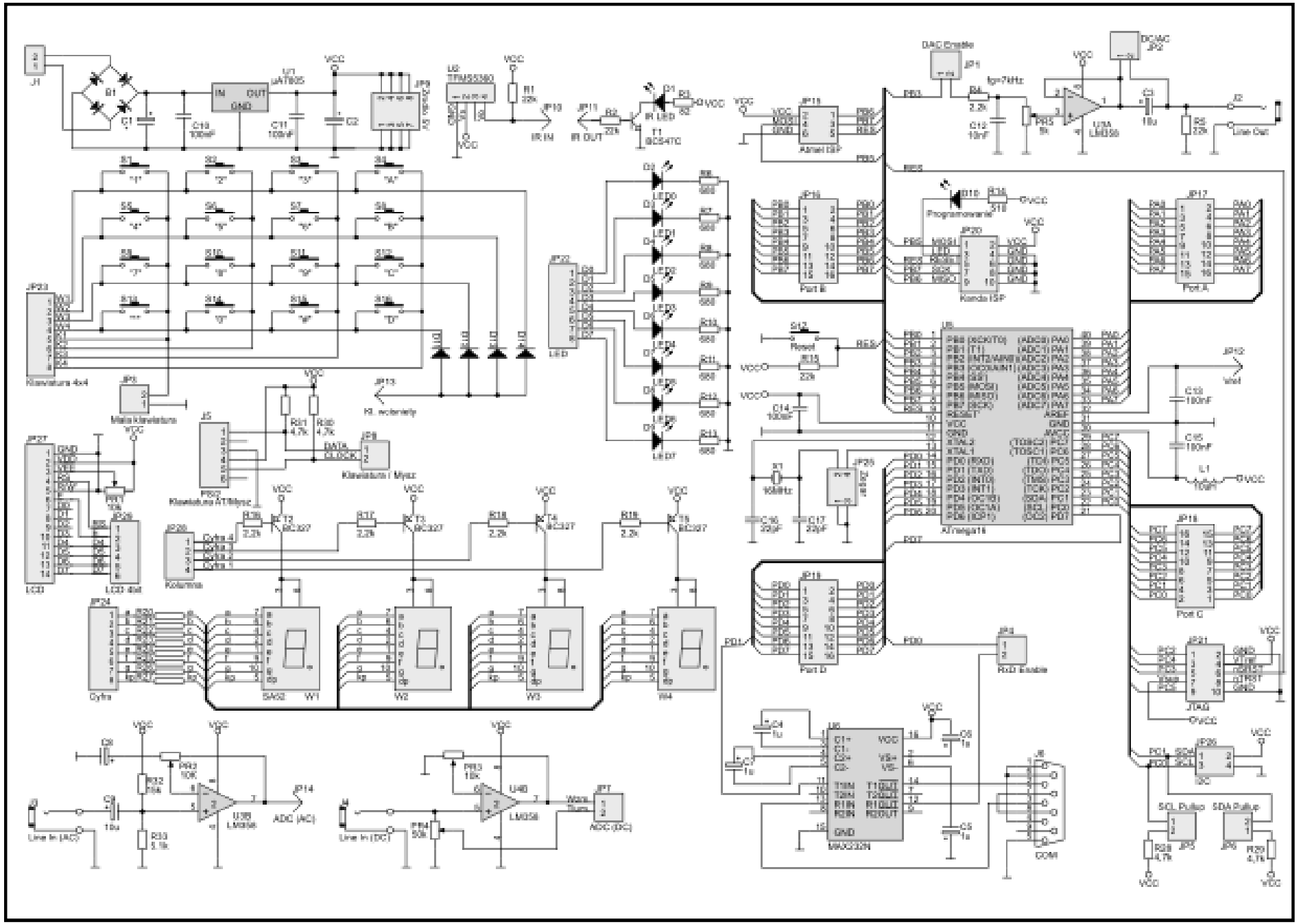
Zawartość zestawu ZL3AVR

mikrokontroler ATmega32 (DIP40)
rezonator kwarcowy o częstotliwości nominalnej 16 MHz
16-przyciskowa klawiatura (4x4)
pole sygnalizacyjne na 8 diodach LED
4 wyświetlacze 7-segmentowe LED (sterowanie multipleksowe)
alfanumeryczny wyświetlacz LCD o organizacji 2x16 znaków
wyprowadzenie linii interfejsu I2C
interfejs RS232
6-stykowe złącze PS/2
nadajnik i odbiornik podczerwieni
wejściowy tor analogowy napięć zmiennych (AC)
wejściowy tor analogowy napięć stałych (DC)
wyjściowy tor analogowy AC/DC
złącze programowania ISP (np. ZL2PRG)
złącze programowania ISP (np. ZL11PRG-M)
złącze JTAG
złącza z wyprowadzonymi liniami we/wy mikrokontrolera
zasilanie 9 VDC/500 mA

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Zestaw ZL3AVR





WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Środowisko projektowe programistyczne

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

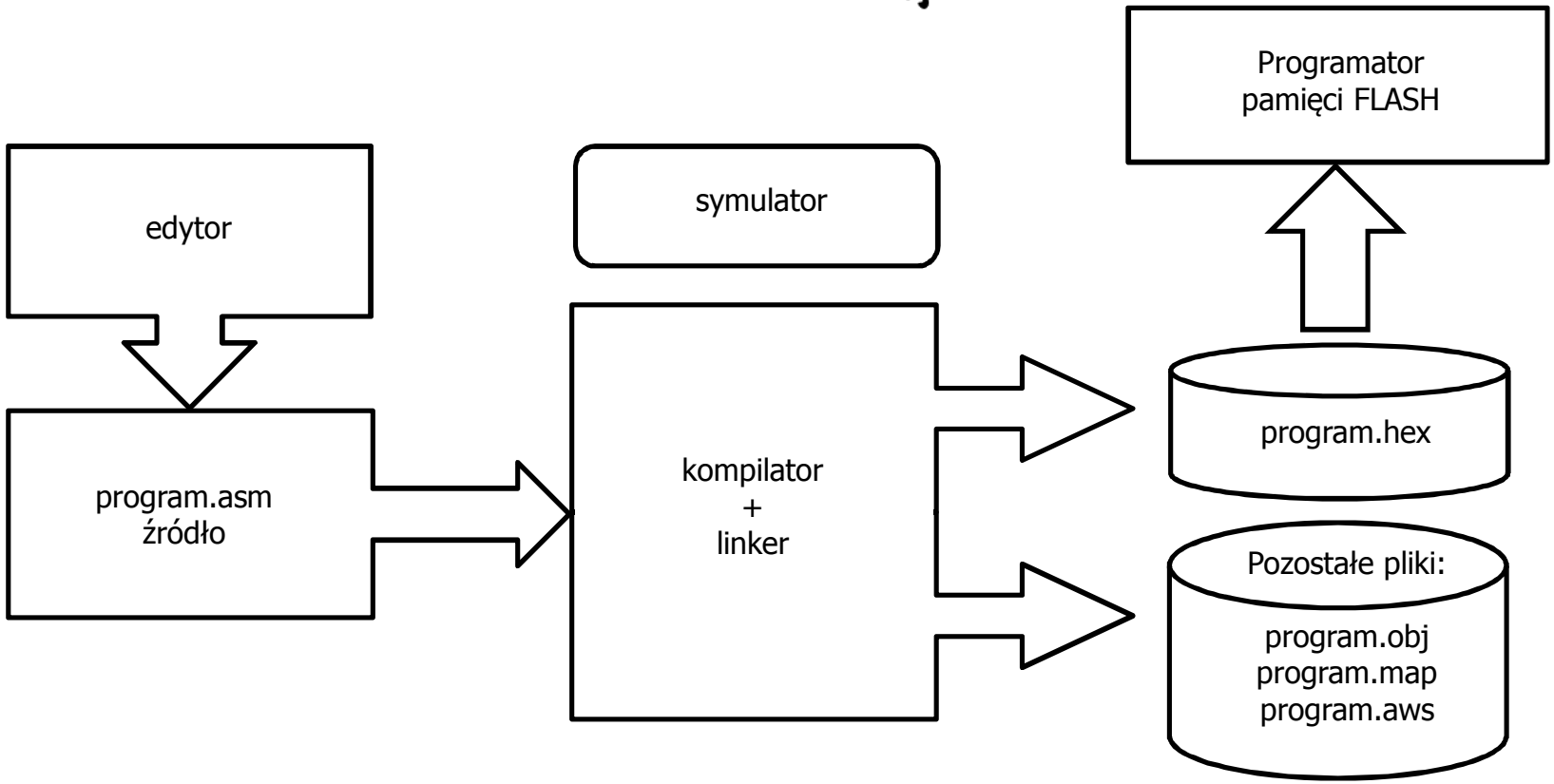
Środowisko projektowe

Będziemy używali pakietu AVR Studio ATMEL
Dostępne są wyższe wersje

Środowisko projektowe

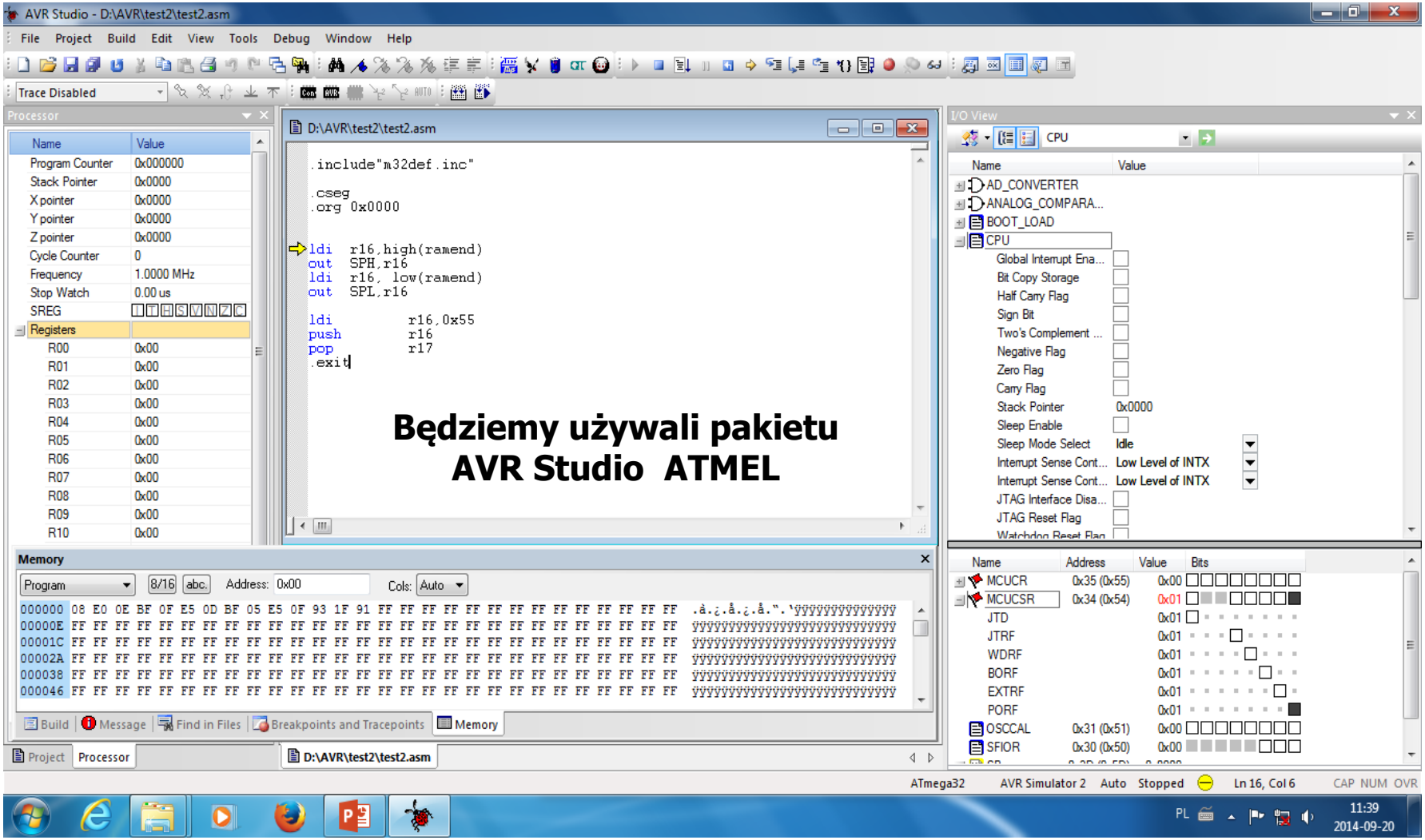
Będziemy używali pakietu AVR Studio ATMEL

IDE – Integrated Development Environment



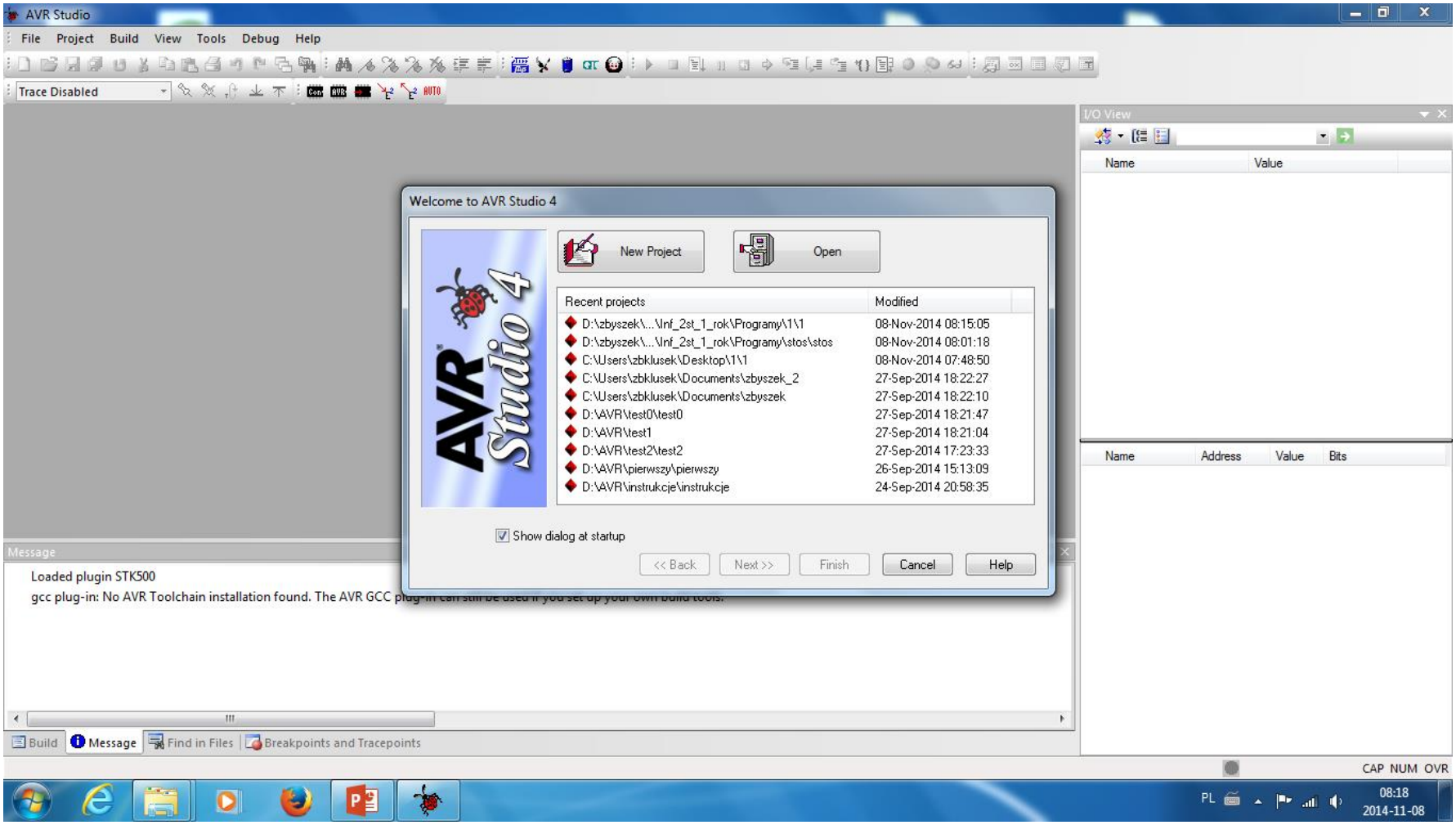
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Środowisko projektowe



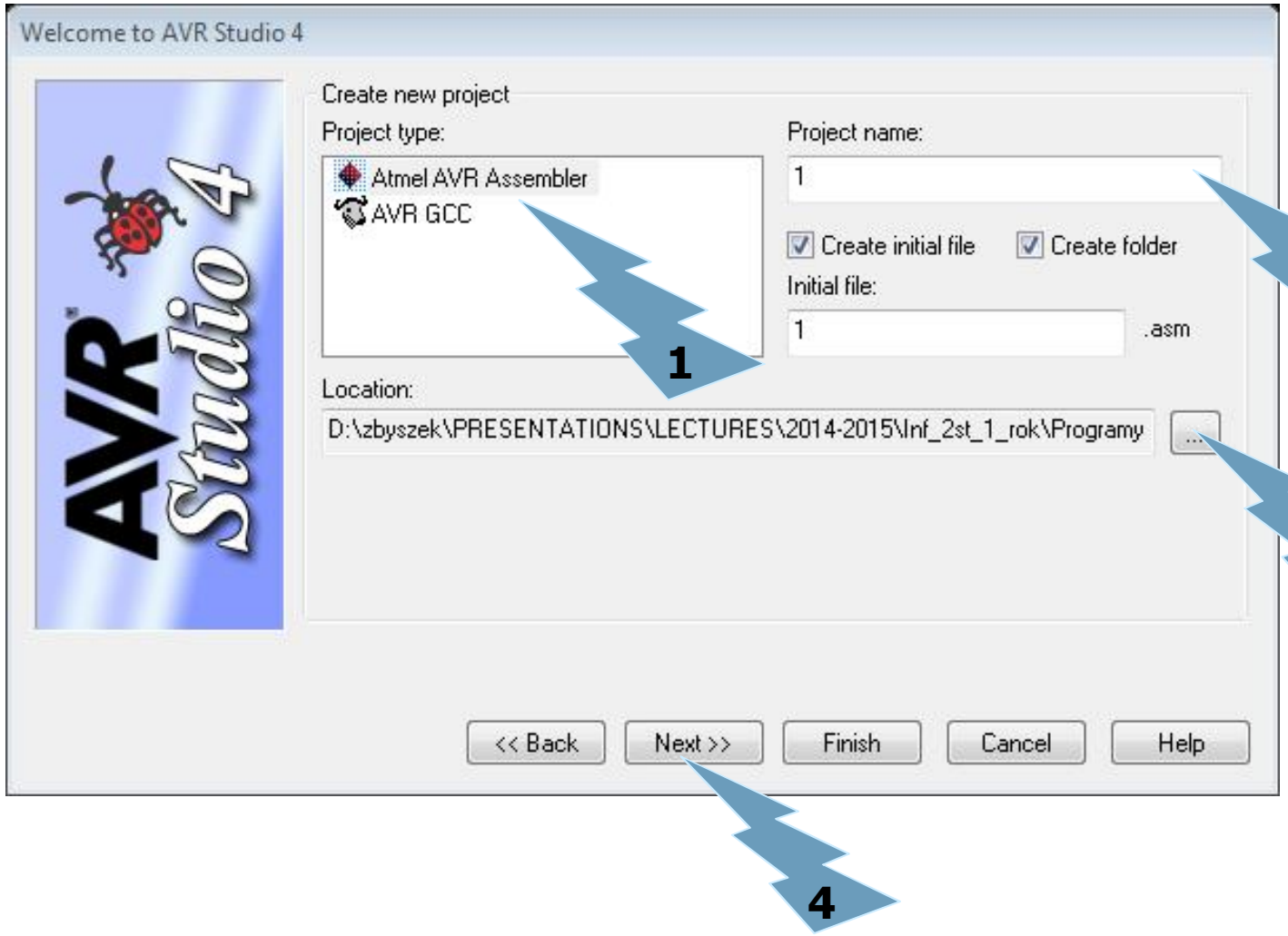
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Pierwszy program – środowisko AVR Studio

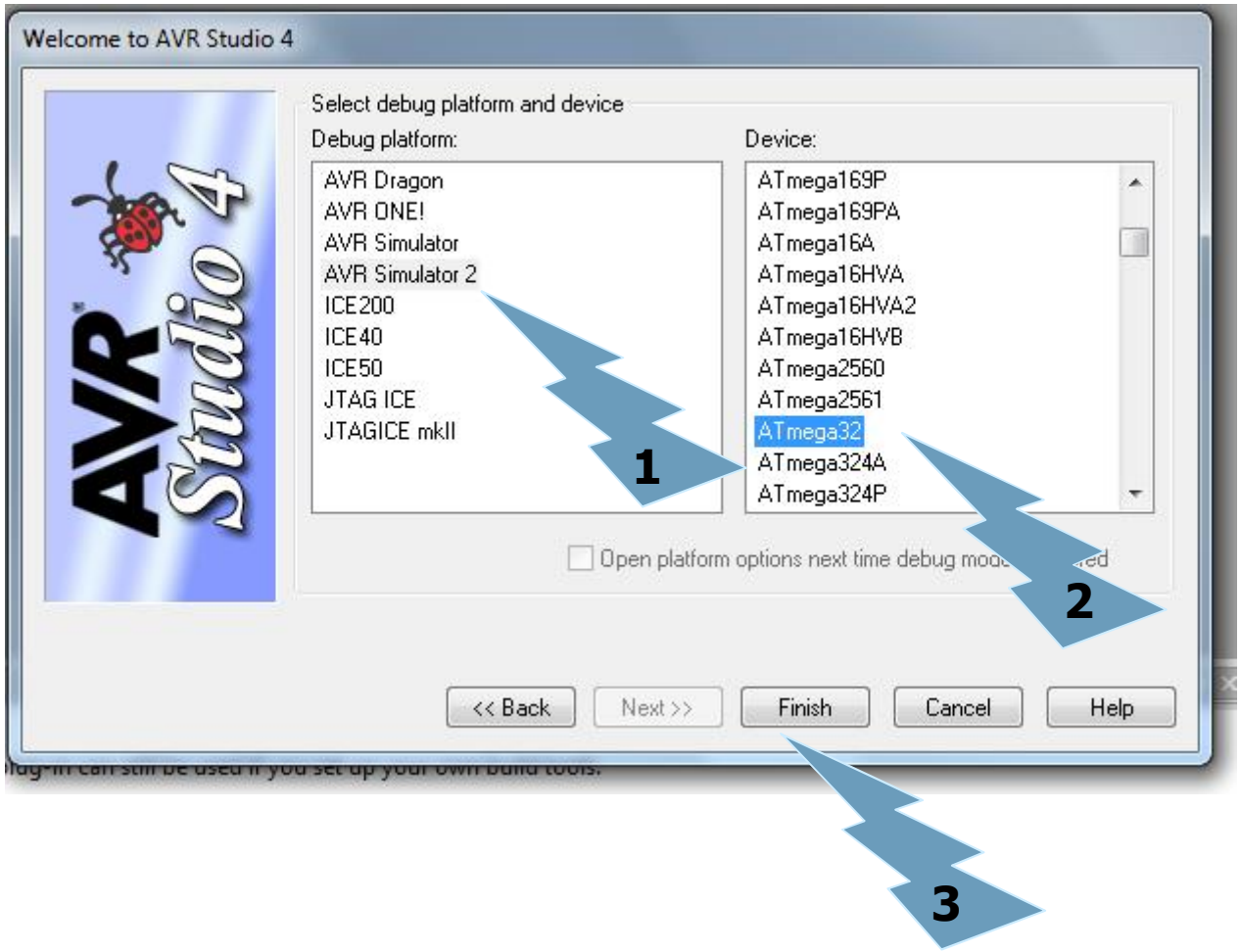


Pierwszy program – środowisko AVR Studio

Pierwszy program – środowisko AVR Studio

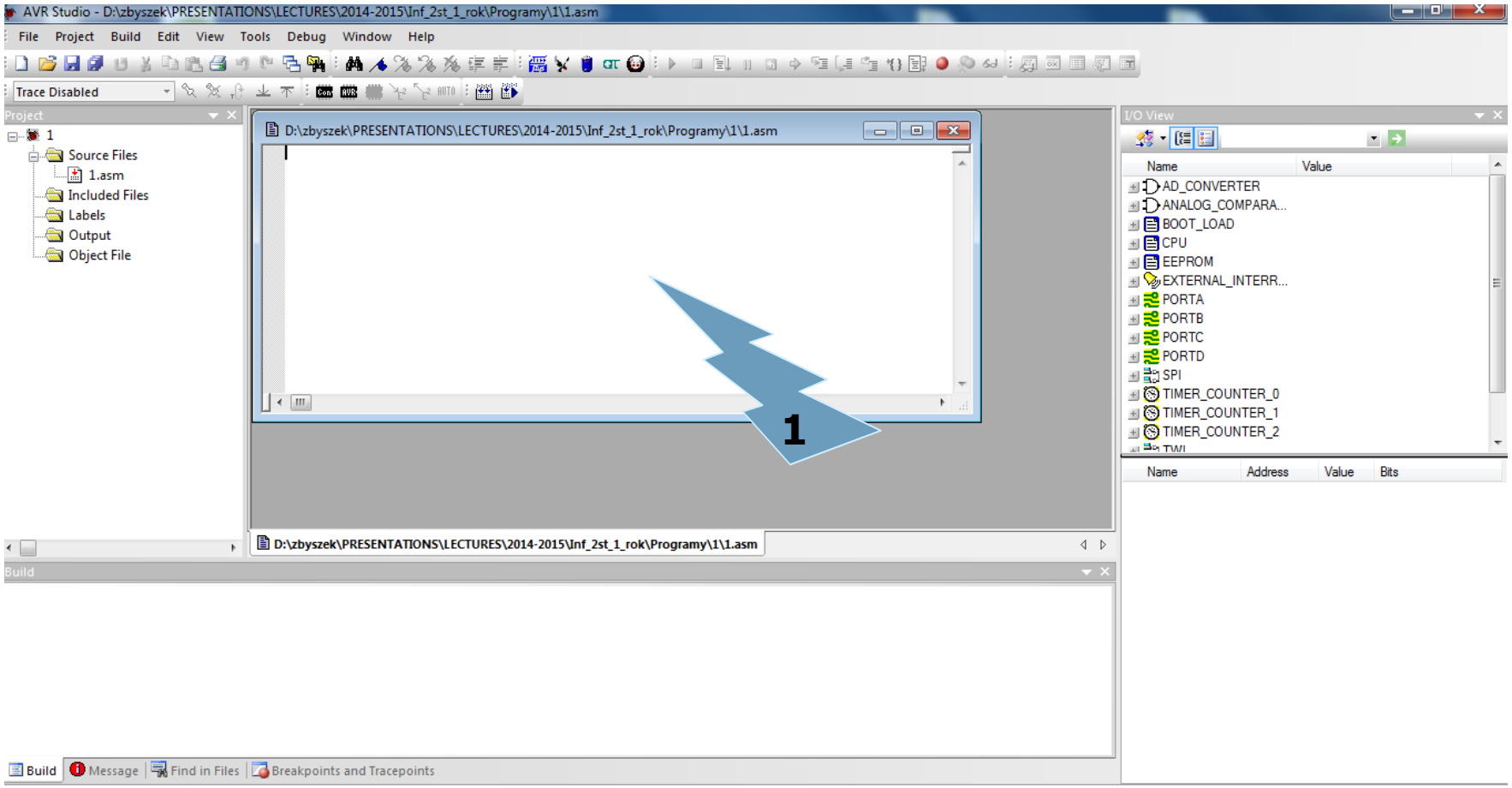


Pierwszy program – środowisko AVR Studio



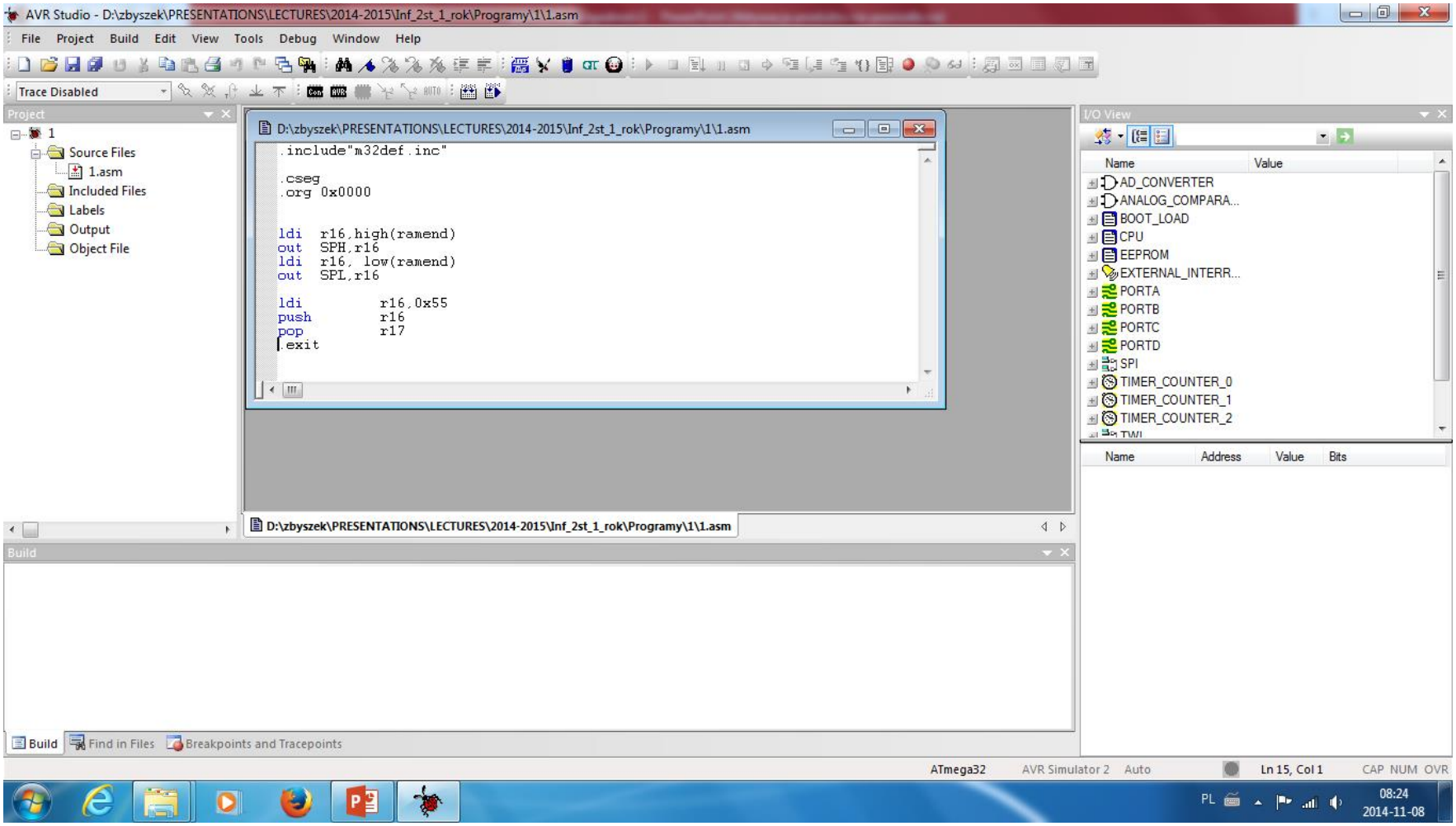
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Pierwszy program – środowisko AVR Studio



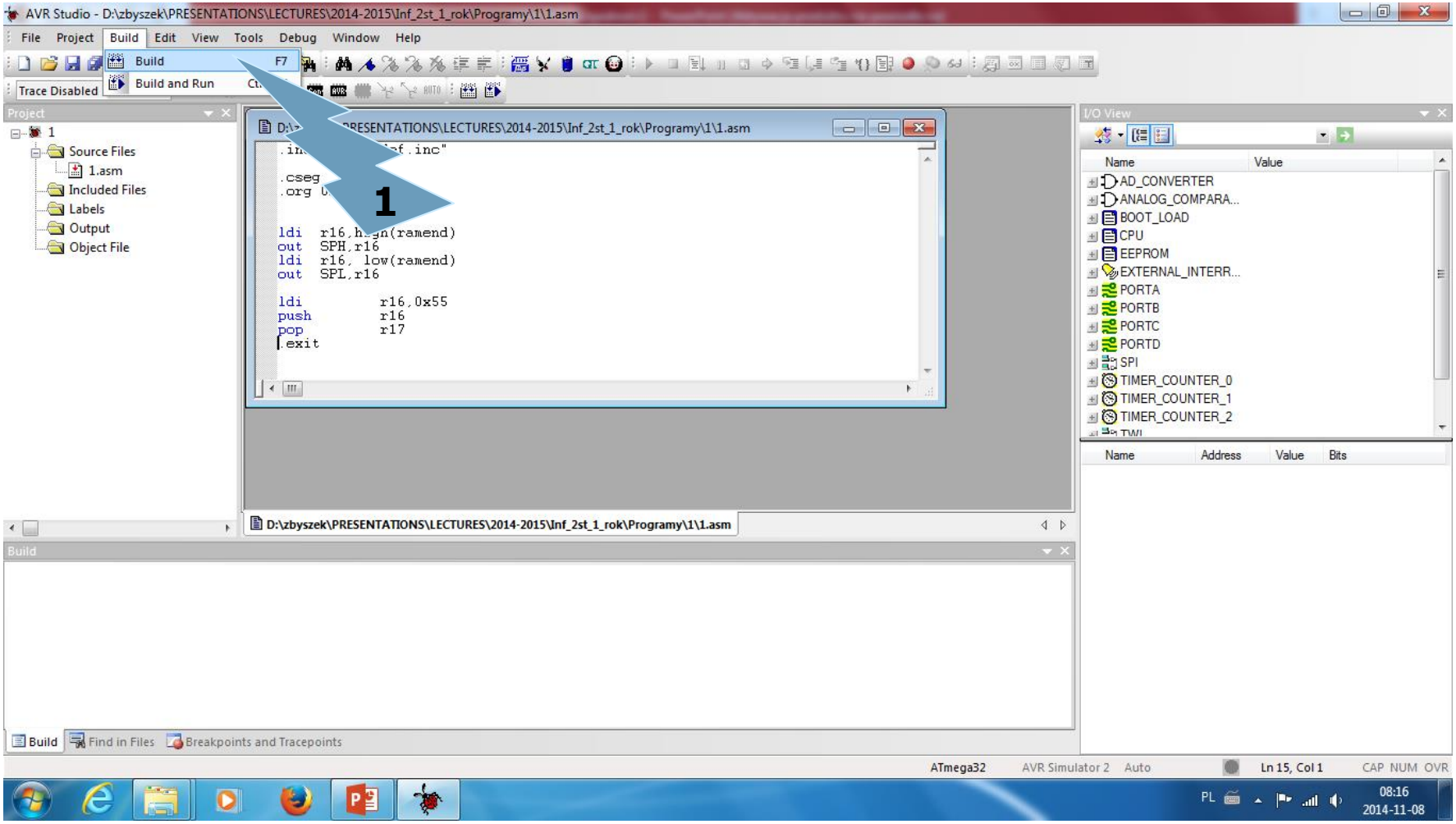
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Pierwszy program – środowisko AVR Studio



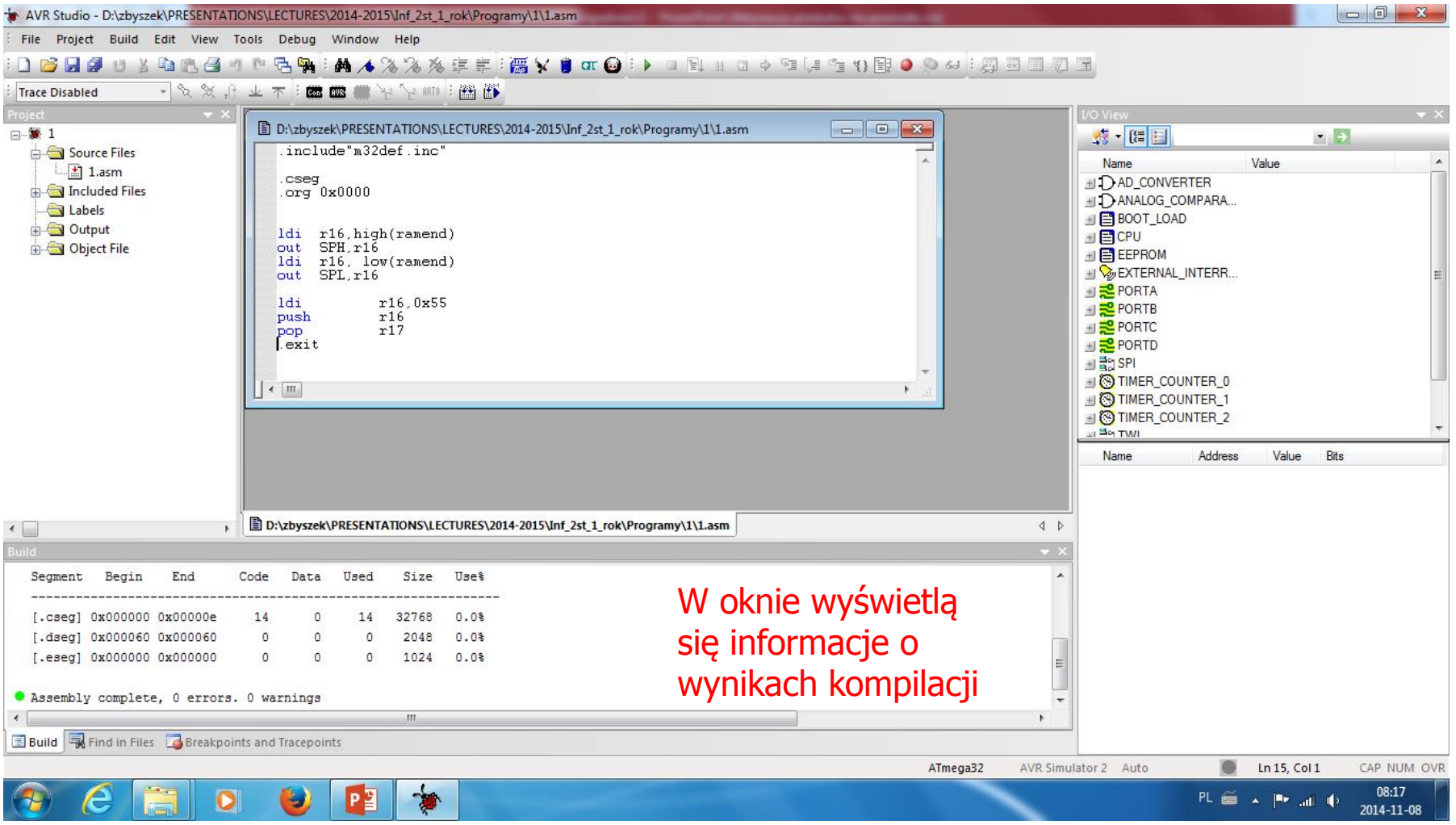
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Pierwszy program – środowisko AVR Studio



WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Pierwszy program – środowisko AVR Studio



W oknie wyświetlą się informacje o wynikach kompilacji

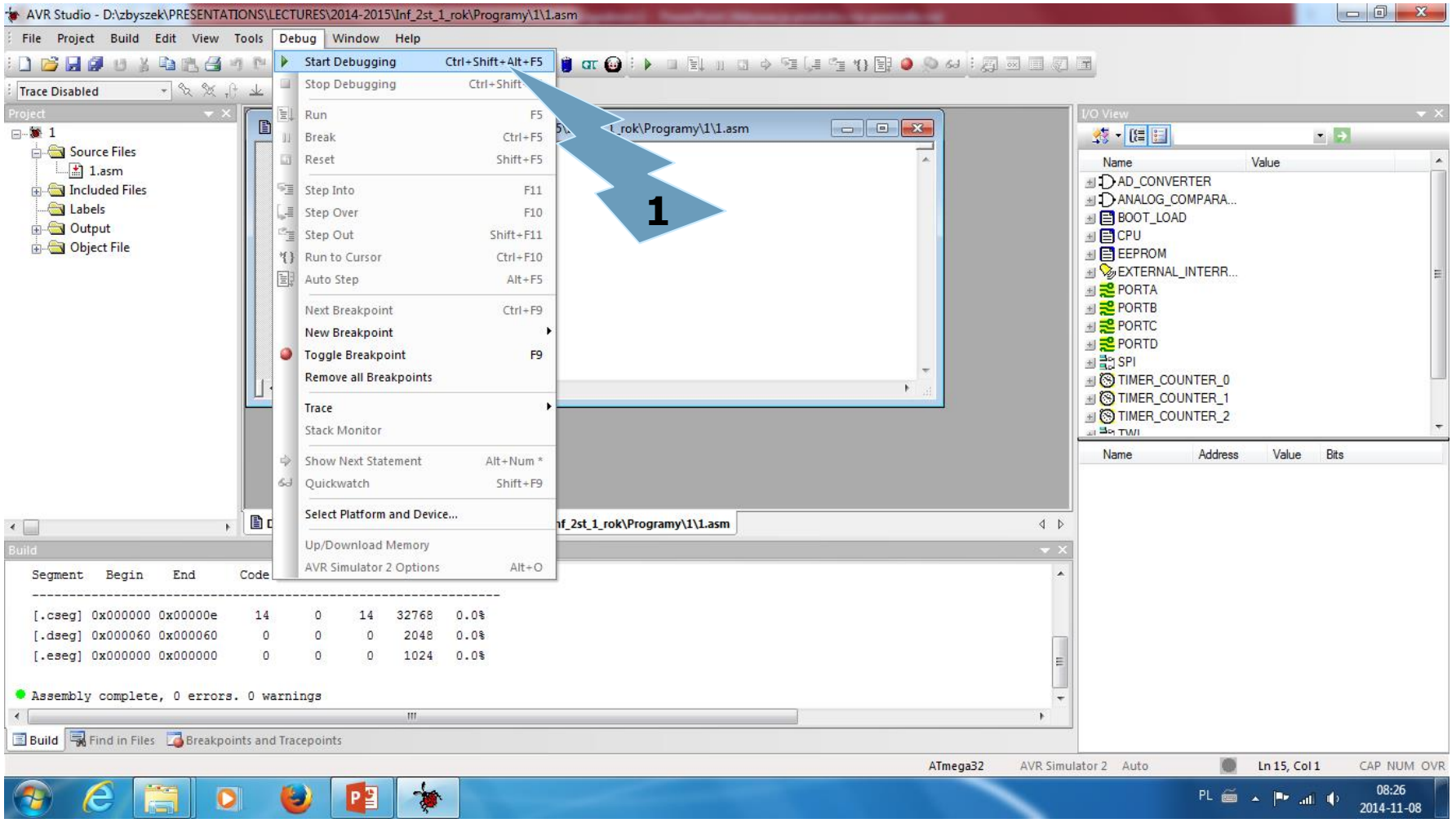
Pierwszy program – środowisko AVR Studio

Debagowanie programu

1. **Praca krokowa**
2. **Śledzenie zawartości rejestrów**
3. **Śledzenie zawartości CPU (w szczególności zawartości SP)**
4. **Śledzenie zawartości pamięci danych (DATA)**

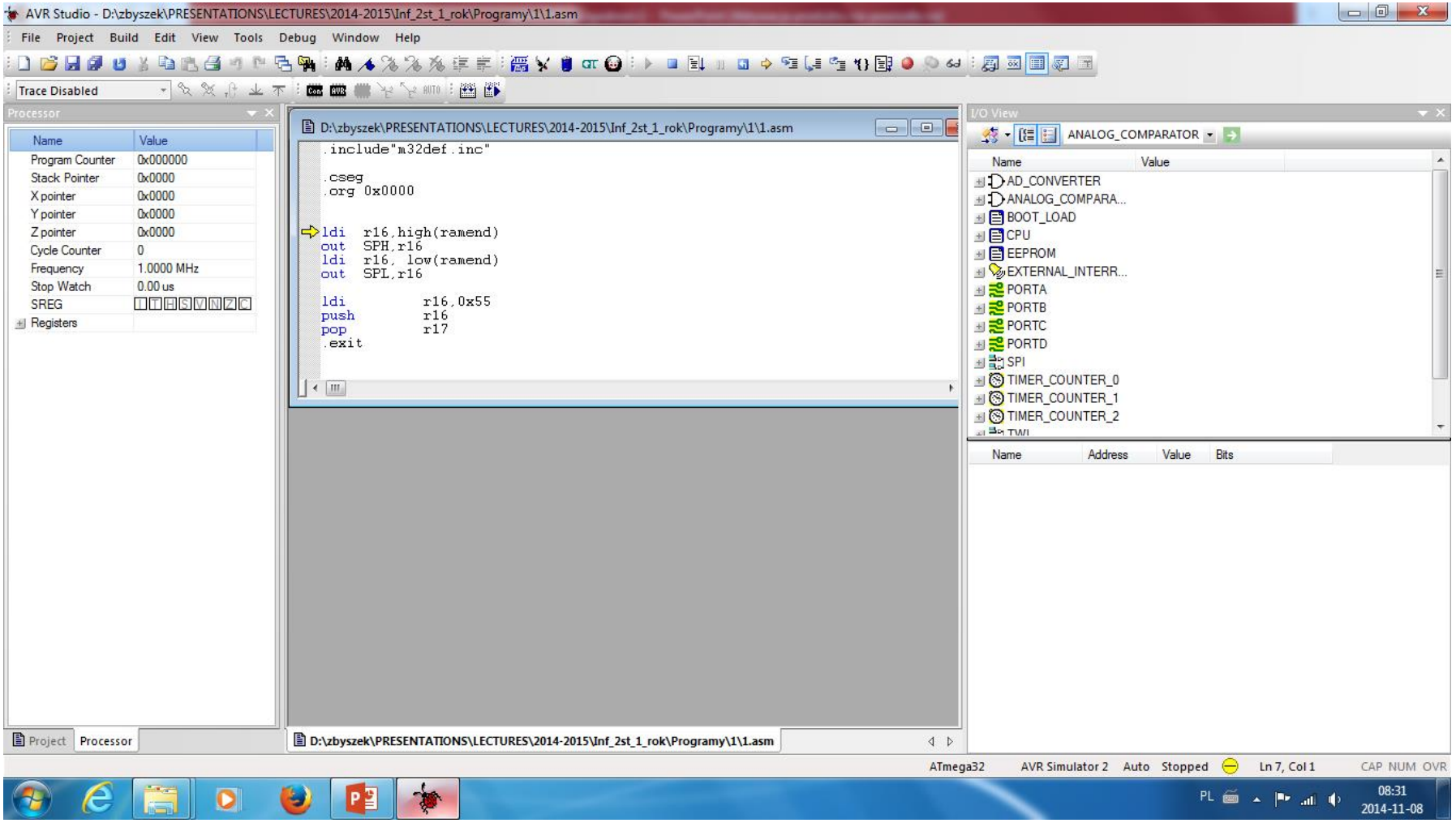
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Pierwszy program – środowisko AVR Studio



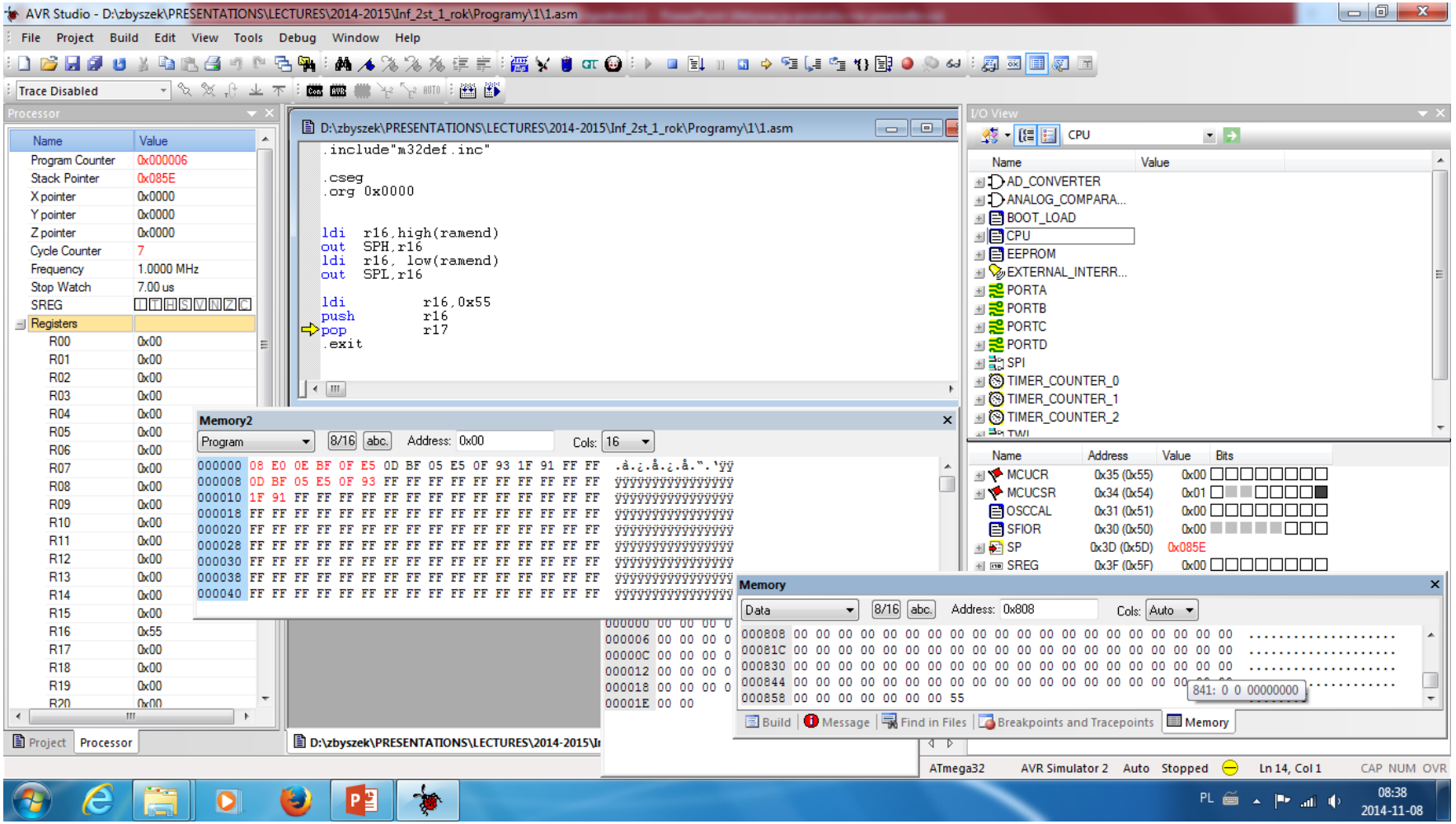
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Pierwszy program – środowisko AVR Studio



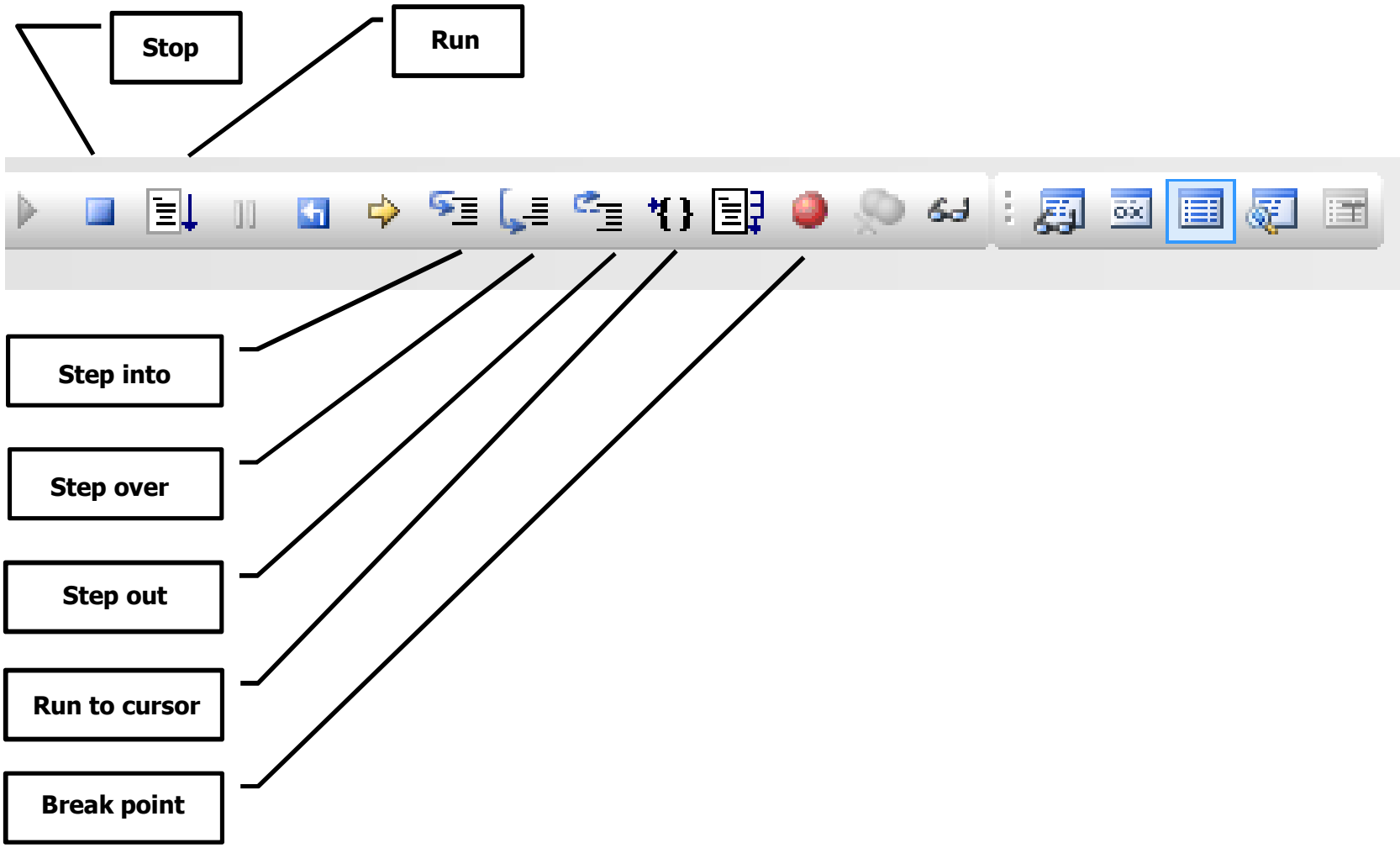
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Pierwszy program – środowisko AVR Studio



WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Pierwszy program – środowisko AVR Studio

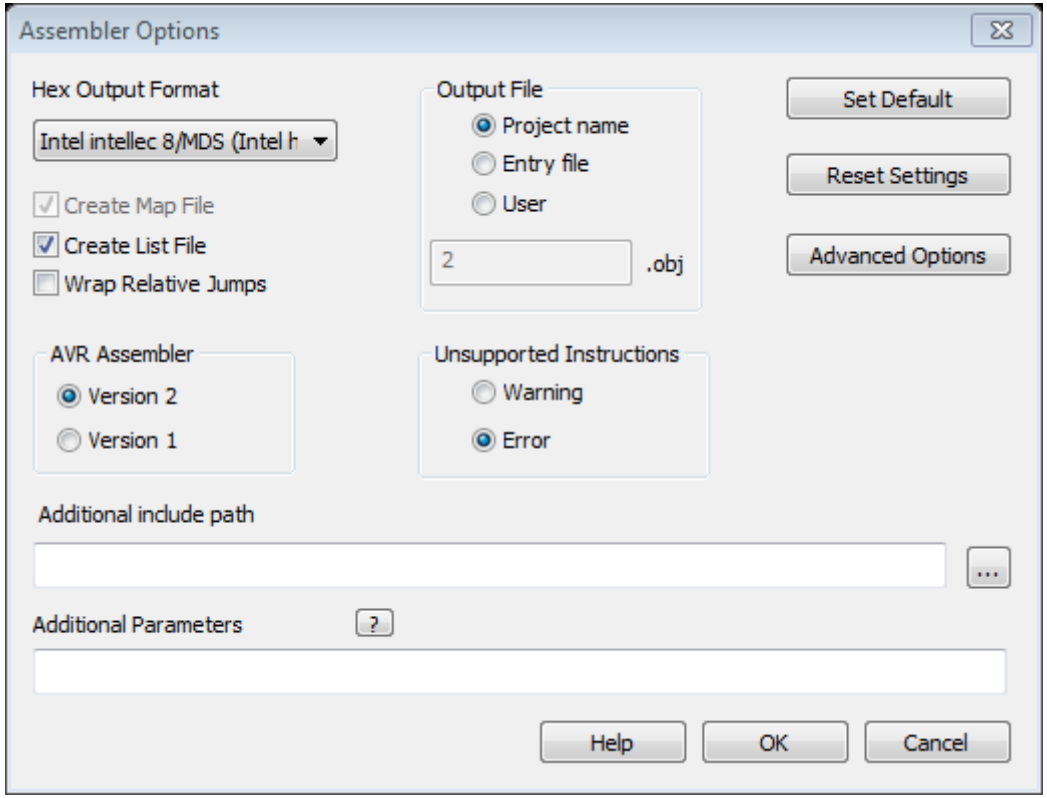


Struktura pliku lst

Listing programu

1. **Struktura pliku lst**
2. **Po co nam listing programu**

Struktura pliku lst



WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Struktura pliku lst

```
; Definitions marked "MEMORY MAPPED"are extended I/O ports
; and cannot be used with IN/OUT instructions
.equ    SREG    = 0x3f
.equ    SPL     = 0x3d
.equ    SPH     = 0x3e
.equ    OCR0    = 0x3c
.equ    GICR    = 0x3b
.equ    GIFR    = 0x3a
.equ    TIMSK   = 0x39
.equ    TIFR    = 0x38
.equ    SPMCR   = 0x37
.equ    TWCR    = 0x36
.equ    MCUCR   = 0x35
.equ    MCUCSR  = 0x34
.equ    TCCR0   = 0x33
.equ    TCNT0   = 0x32
.equ    OSCCAL  = 0x31
.equ    OCDR    = 0x31
.equ    SFIOR   = 0x30
.equ    TCCR1A  = 0x2f
.equ    TCCR1B  = 0x2e
.equ    TCNT1L  = 0x2c
.equ    TCNT1H  = 0x2d
.equ    OCR1AL  = 0x2a
.equ    OCR1AH  = 0x2b
.equ    OCR1BL  = 0x28
.equ    OCR1BH  = 0x29
.equ    ICR1L   = 0x26
.equ    ICR1H   = 0x27
.equ    TCCR2   = 0x25
.equ    TCNT2   = 0x24
.equ    OCR2    = 0x23
.equ    ASSR    = 0x22
.equ    WDTCR   = 0x21
.equ    UBRRH   = 0x20
.equ    UCSRC   = 0x20
.equ    EEARL   = 0x1e
.equ    EEARH   = 0x1f
.equ    EEDR    = 0x1d
.equ    EECR    = 0x1c
.equ    PORTA   = 0x1b
.equ    DDRA    = 0x1a
.equ    PINA    = 0x19
.equ    PORTB   = 0x18
.equ    DDRB    = 0x17
.equ    PINB    = 0x16
.equ    PORTC   = 0x15
.equ    DDRC    = 0x14
.equ    PINC    = 0x13
.equ    PORTD   = 0x12
.equ    DDRD    = 0x11
.equ    PIND    = 0x10
.equ    SPDR    = 0x0f
.equ    SPSR    = 0x0e
.equ    SPCR    = 0x0d
.equ    UDR     = 0x0c
.equ    UCSRA   = 0x0b
.equ    UCSRB   = 0x0a
.equ    UBRRL   = 0x09
.equ    ACSR    = 0x08
.equ    ADMUX   = 0x07
.equ    ADCSRA  = 0x06
.equ    ADCH    = 0x05
.equ    ADCL    = 0x04
.equ    TWDR    = 0x03
.equ    TWAR    = 0x02
.equ    TWSR    = 0x01
.equ    TWBR    = 0x00
```

```
; ***** BIT DEFINITIONS ************************************************

        ; ***** EEPROM ************************
        ; EEDR - EEPROM Data Register
        .equ    EEDR0           = 0             ; EEPROM Data Register bit 0
        .equ    EEDR1           = 1             ; EEPROM Data Register bit 1
        .equ    EEDR2           = 2             ; EEPROM Data Register bit 2
        .equ    EEDR3           = 3             ; EEPROM Data Register bit 3
        .equ    EEDR4           = 4             ; EEPROM Data Register bit 4
        .equ    EEDR5           = 5             ; EEPROM Data Register bit 5
        .equ    EEDR6           = 6             ; EEPROM Data Register bit 6
        .equ    EEDR7           = 7             ; EEPROM Data Register bit 7

        ; EECR - EEPROM Control Register
        .equ    EERE            = 0             ; EEPROM Read Enable
        .equ    EEWE            = 1             ; EEPROM Write Enable
        .equ    EEMWE           = 2             ; EEPROM Master Write Enable
        .equ    EERIE           = 3             ; EEPROM Ready Interrupt Enable


        ; ***** WATCHDOG **********************
        ; WDTCR - Watchdog Timer Control Register
        .equ    WDP0            = 0             ; Watch Dog Timer Prescaler bit 0
        .equ    WDP1            = 1             ; Watch Dog Timer Prescaler bit 1
        .equ    WDP2            = 2             ; Watch Dog Timer Prescaler bit 2
        .equ    WDE             = 3             ; Watch Dog Enable
        .equ    WDTOE           = 4             ; RW
        .equ    WDDE            = WDTOE         ; For compatibility
```

Struktura pliku lst

```
 ; ***** BOOTLOADER DECLARATIONS ***********************************************
            .equ    NRWW_START_ADDR                 = 0x3800
            .equ    NRWW_STOP_ADDR                  = 0x3fff
            .equ    RWW_START_ADDR                  = 0x0
            .equ    RWW_STOP_ADDR   = 0x37ff
            .equ    PAGESIZE        = 64
            .equ    FIRSTBOOTSTART  = 0x3f00
            .equ    SECONDBOOTSTART                 = 0x3e00
            .equ    THIRDBOOTSTART  = 0x3c00
            .equ    FOURTHBOOTSTART                 = 0x3800
            .equ    SMALLBOOTSTART  = FIRSTBOOTSTART
            .equ    LARGEBOOTSTART  = FOURTHBOOTSTART
; ***** INTERRUPT VECTORS ************************************************
            .equ    INT0addr        = 0x0002        ; External Interrupt Request 0
            .equ    INT1addr        = 0x0004        ; External Interrupt Request 1
            .equ    INT2addr        = 0x0006        ; External Interrupt Request 2
            .equ    OC2addr         = 0x0008        ; Timer/Counter2 Compare Match
            .equ    OVF2addr        = 0x000a        ; Timer/Counter2 Overflow
            .equ    ICP1addr        = 0x000c        ; Timer/Counter1 Capture Event
            .equ    OC1Aaddr        = 0x000e        ; Timer/Counter1 Compare Match A
            .equ    OC1Baddr        = 0x0010        ; Timer/Counter1 Compare Match B
            .equ    OVF1addr        = 0x0012        ; Timer/Counter1 Overflow
            .equ    OC0addr         = 0x0014        ; Timer/Counter0 Compare Match
            .equ    OVF0addr        = 0x0016        ; Timer/Counter0 Overflow
            .equ    SPIaddr         = 0x0018        ; Serial Transfer Complete
            .equ    URXCaddr        = 0x001a        ; USART, Rx Complete
            .equ    UDREaddr        = 0x001c        ; USART Data Register Empty
            .equ    UTXCaddr        = 0x001e        ; USART, Tx Complete
            .equ    ADCCaddr        = 0x0020        ; ADC Conversion Complete
            .equ    ERDYaddr        = 0x0022        ; EEPROM Ready
            .equ    ACIaddr         = 0x0024        ; Analog Comparator
            .equ    TWIaddr         = 0x0026        ; 2-wire Serial Interface
            .equ    SPMRaddr        = 0x0028        ; Store Program Memory Ready

            .equ    INT_VECTORS_SIZE                = 42            ; size in words
```

Struktura pliku lst

```
#endif  /* _M32DEF_INC_ */

.cseg
.org 0x0000


000000 e008     ldi  r16,high(ramend)
000001 bf0e     out  SPH,r16
000002 e50f     ldi  r16, low(ramend)
000003 bf0d     out  SPL,r16

000004 e505     ldi                     r16,0x55
000005 930f     push                    r16
000006 911f     pop                     r17


RESOURCE USE INFORMATION
------------------------

Notice:
The register and instruction counts are symbol table hit counts,
and hence implicitly used resources are not counted, eg, the
'lpm' instruction without operands implicitly uses r0 and z,
none of which are counted.

x,y,z are separate entities in the symbol table and are
counted separately from r26..r31 here.

.dseg memory usage only counts static data declared with .byte

ATmega32 register use summary:
r0 :  0 r1 :  0 r2 :  0 r3 :  0 r4 :  0 r5 :  0 r6 :  0 r7 :  0
r8 :  0 r9 :  0 r10:  0 r11:  0 r12:  0 r13:  0 r14:  0 r15:  0
r16:  6 r17:  1 r18:  0 r19:  0 r20:  0 r21:  0 r22:  0 r23:  0
r24:  0 r25:  0 r26:  0 r27:  0 r28:  0 r29:  0 r30:  0 r31:  0
x :  0 y :  0 z :  0
Registers used: 2 out of 35 (5.7%)
```

Struktura pliku lst

ATmega32 instruction use summary:

lds : 0	sts : 0	adc : 0	add : 0	adiw : 0	and : 0
andi : 0	asr : 0	bclr : 0	bld : 0	brbc : 0	brbs : 0
brcc : 0	brcs : 0	break : 0	breq : 0	brge : 0	brhc : 0
brhs : 0	brid : 0	brie : 0	brlo : 0	brlt : 0	brmi : 0
brne : 0	brpl : 0	brsh : 0	brtc : 0	brts : 0	brvc : 0
brvs : 0	bset : 0	bst : 0	call : 0	cbi : 0	cbr : 0
clc : 0	clh : 0	cli : 0	cln : 0	clr : 0	cls : 0
clt : 0	clv : 0	clz : 0	com : 0	cp : 0	cpc : 0
cpi : 0	cpse : 0	dec : 0	eor : 0	fmul : 0	fmuls : 0
fmulsu: 0	icall : 0	ijmp : 0	in : 0	inc : 0	jmp : 0
ld : 0	ldd : 0	ldi : 3	lds : 0	lpm : 0	lsl : 0
lsr : 0	mov : 0	movw : 0	mul : 0	muls : 0	mulsu : 0
neg : 0	nop : 0	or : 0	ori : 0	out : 2	pop : 1
push : 1	rcall : 0	ret : 0	reti : 0	rjmp : 0	rol : 0
ror : 0	sbc : 0	sbci : 0	sbi : 0	sbic : 0	sbis : 0
sbiw : 0	sbr : 0	sbrc : 0	sbrs : 0	sec : 0	seh : 0
sei : 0	sen : 0	ser : 0	ses : 0	set : 0	sev : 0
sez : 0	sleep : 0	spm : 0	st : 0	std : 0	sts : 0
sub : 0	subi : 0	swap : 0	tst : 0	wdr : 0	

Instructions used: 4 out of 113 (3.5%)

ATmega32 memory use summary [bytes]:

Segment	Begin	End	Code	Data	Used	Size	Use%
[.cseg]	0x000000	0x00000e	14	0	14	32768	0.0%
[.dseg]	0x000060	0x000060	0	0	0	2048	0.0%
[.eseg]	0x000000	0x000000	0	0	0	1024	0.0%

Assembly complete, 0 errors, 0 warnings

Sterowanie listingiem

Dyrektywa list/nolist

1. **Struktura pliku lst**
2. **Po co nam listing programu**

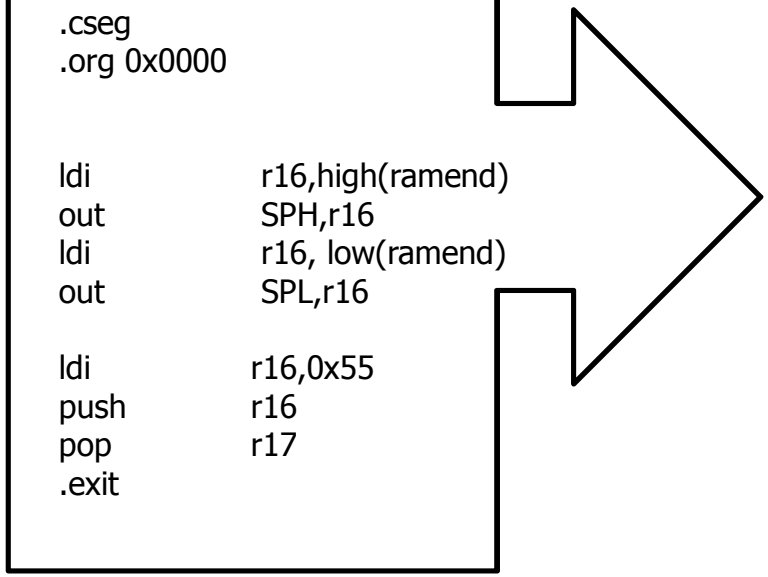
Dyrektywa .listmac

Sterowanie listingiem

```
.nolist
.include"m32def.inc"
.list
.cseg
.org 0x0000


ldi             r16,high(ramend)
out             SPH,r16
ldi             r16, low(ramend)
out             SPL,r16

ldi             r16,0x55
push            r16
pop             r17
.exit
```



```
AVRASM ver. 2.1.42  D:\zbyszek\PRESENTATIONS\LECTURES\2014-
2015\Inf_2st_1_rok\Programy\2\2.asm Sat Nov 08 09:24:48 2014

D:\zbyszek\PRESENTATIONS\LECTURES\2014-2015\Inf_2st_1_rok\Programy\2\2.asm(2): Including
file 'C:\Program Files (x86)\Atmel\AVR Tools\AvrAssembler2\Appnotes\m32def.inc'

                .list
                .cseg
                .org 0x0000


000000 e008     ldi             r16,high(ramend)
000001 bf0e     out             SPH,r16
000002 e50f     ldi             r16, low(ramend)
000003 bf0d     out             SPL,r16

000004 e505     ldi             r16,0x55
000005 930f     push            r16
000006 911f     pop             r17


RESOURCE USE INFORMATION
------------------------
Notice:
The register and instruction counts are symbol table hit counts,and hence implicitly used resources are
not counted, eg, the 'lpm' instruction without operands implicitly uses r0 and z,
none of which are counted.

x,y,z are separate entities in the symbol table and are
counted separately from r26..r31 here.

.dseg memory usage only counts static data declared with .byte

ATmega32 register use summary:
r0 :  0 r1 :  0 r2 :  0 r3 :  0 r4 :  0 r5 :  0 r6 :  0 r7 :  0
r8 :  0 r9 :  0 r10:  0 r11:  0 r12:  0 r13:  0 r14:  0 r15:  0
r16:  6 r17:  1 r18:  0 r19:  0 r20:  0 r21:  0 r22:  0 r23:  0
r24:  0 r25:  0 r26:  0 r27:  0 r28:  0 r29:  0 r30:  0 r31:  0
x :  0 y :  0 z :  0
Registers used: 2 out of 35 (5.7%)

ATmega32 instruction use summary:
.lds :  0 .sts :  0 adc :  0 add :  0 adiw :  0 and :  0
andi :  0 asr :  0 bclr :  0 bld :  0 brbc :  0 brbs :  0
brcc :  0 brcs :  0 break :  0 breq :  0 brge :  0 brhc :  0
brhs :  0 brid :  0 brie :  0 brlo :  0 brlt :  0 brmi :  0
brne :  0 brpl :  0 brsh :  0 brtc :  0 brts :  0 brvc :  0
brvs :  0 bset :  0 bst :  0 call :  0 cbi :  0 cbr :  0
clc :  0 clh :  0 cli :  0 cln :  0 clr :  0 cls :  0
clt :  0 clv :  0 clz :  0 com :  0 cp :  0 cpc :  0
cpi :  0 cpse :  0 dec :  0 eor :  0 fmul :  0 fmuls :  0
fmulsu:  0 icall :  0 ijmp :  0 in :  0 inc :  0 jmp :  0
ld :  0 ldd :  0 ldi :  3 lds :  0 lpm :  0 lsl :  0
lsr :  0 mov :  0 movw :  0 mul :  0 muls :  0 mulsu :  0
neg :  0 nop :  0 or :  0 ori :  0 out :  2 pop :  1
push :  1 rcall :  0 ret :  0 reti :  0 rjmp :  0 rol :  0
ror :  0 sbc :  0 sbci :  0 sbi :  0 sbic :  0 sbis :  0
sbiw :  0 sbr :  0 sbrc :  0 sbrs :  0 sec :  0 seh :  0
sei :  0 sen :  0 ser :  0 ses :  0 set :  0 sev :  0
sez :  0 sleep :  0 spm :  0 st :  0 std :  0 sts :  0
sub :  0 subi :  0 swap :  0 tst :  0 wdr :  0
Instructions used: 4 out of 113 (3.5%)

ATmega32 memory use summary [bytes]:
Segment  Begin   End     Code  Data  Used   Size  Use%
---------------------------------------------------------------
[.cseg] 0x000000 0x00000e    14     0    14  32768  0.0%
[.dseg] 0x000060 0x000060     0     0     0   2048  0.0%
[.eseg] 0x000000 0x000000     0     0     0   1024  0.0%

Assembly complete, 0 errors, 0 warnings
```

Emulatory

Emulacja

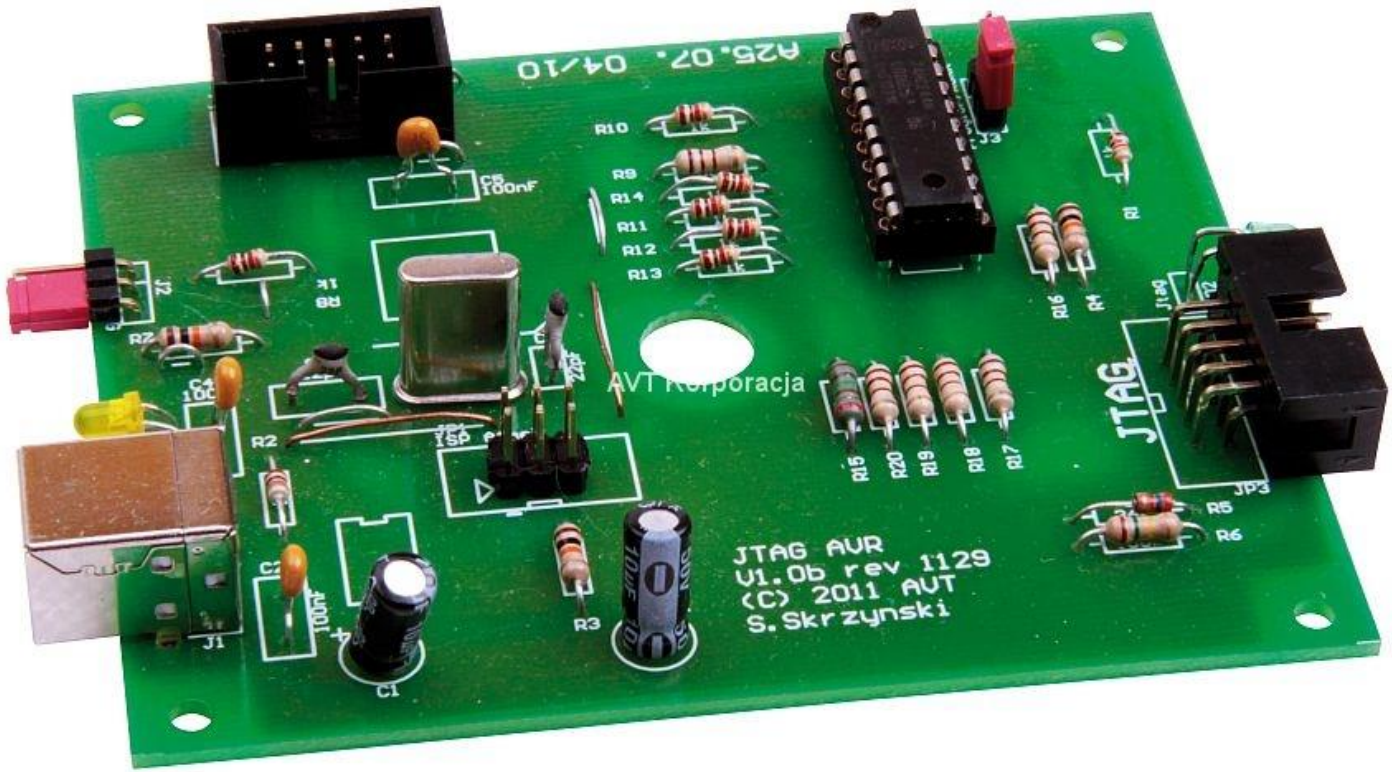
ICE
In Circuit Emulation

OCD
On-Chip Debug

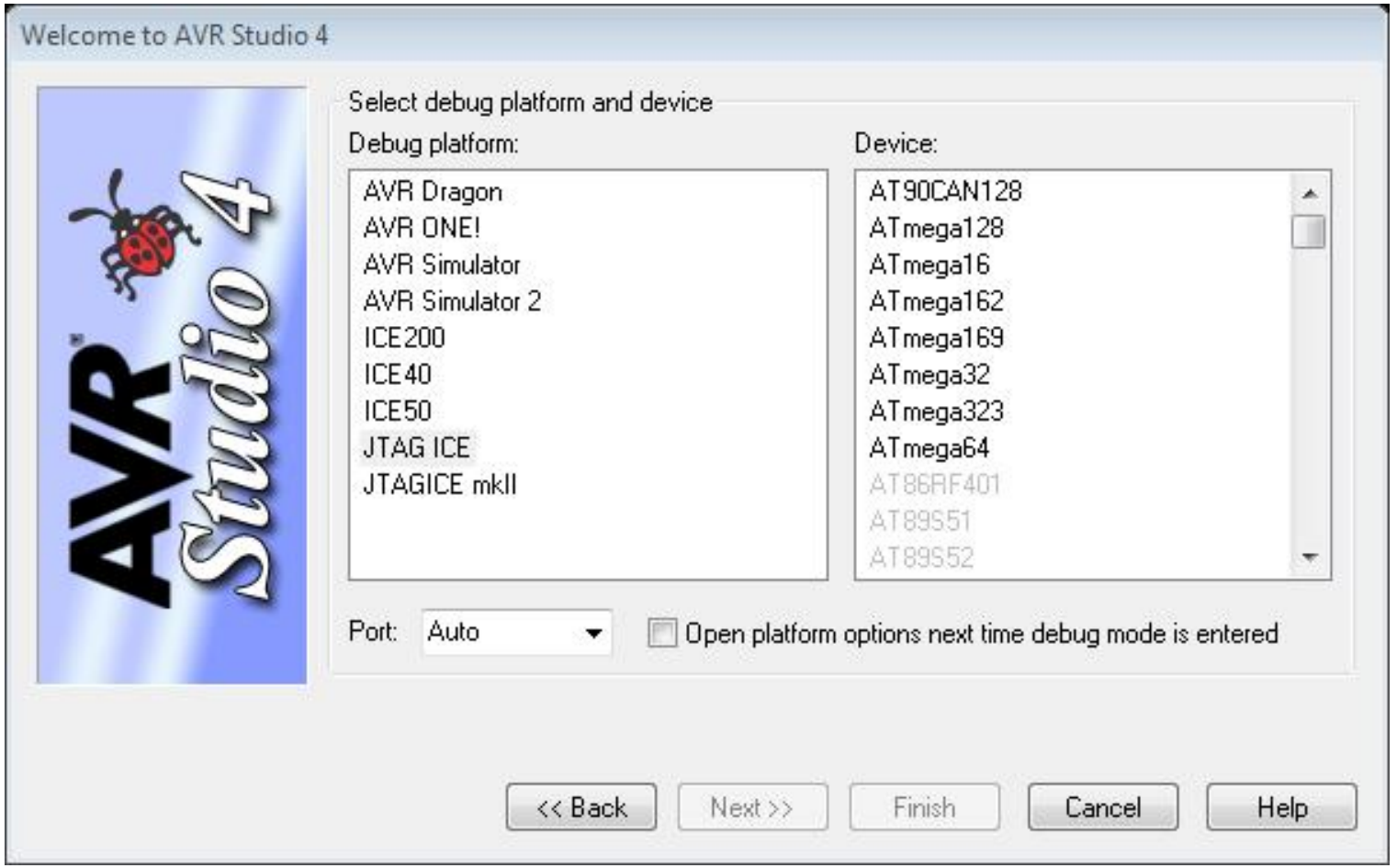
Rezerwacja linii dla złączą JTAG
TDO – PC4
TDI – PC5
TCK – PC2
TMS - PC3

Emulatory

AVT5322

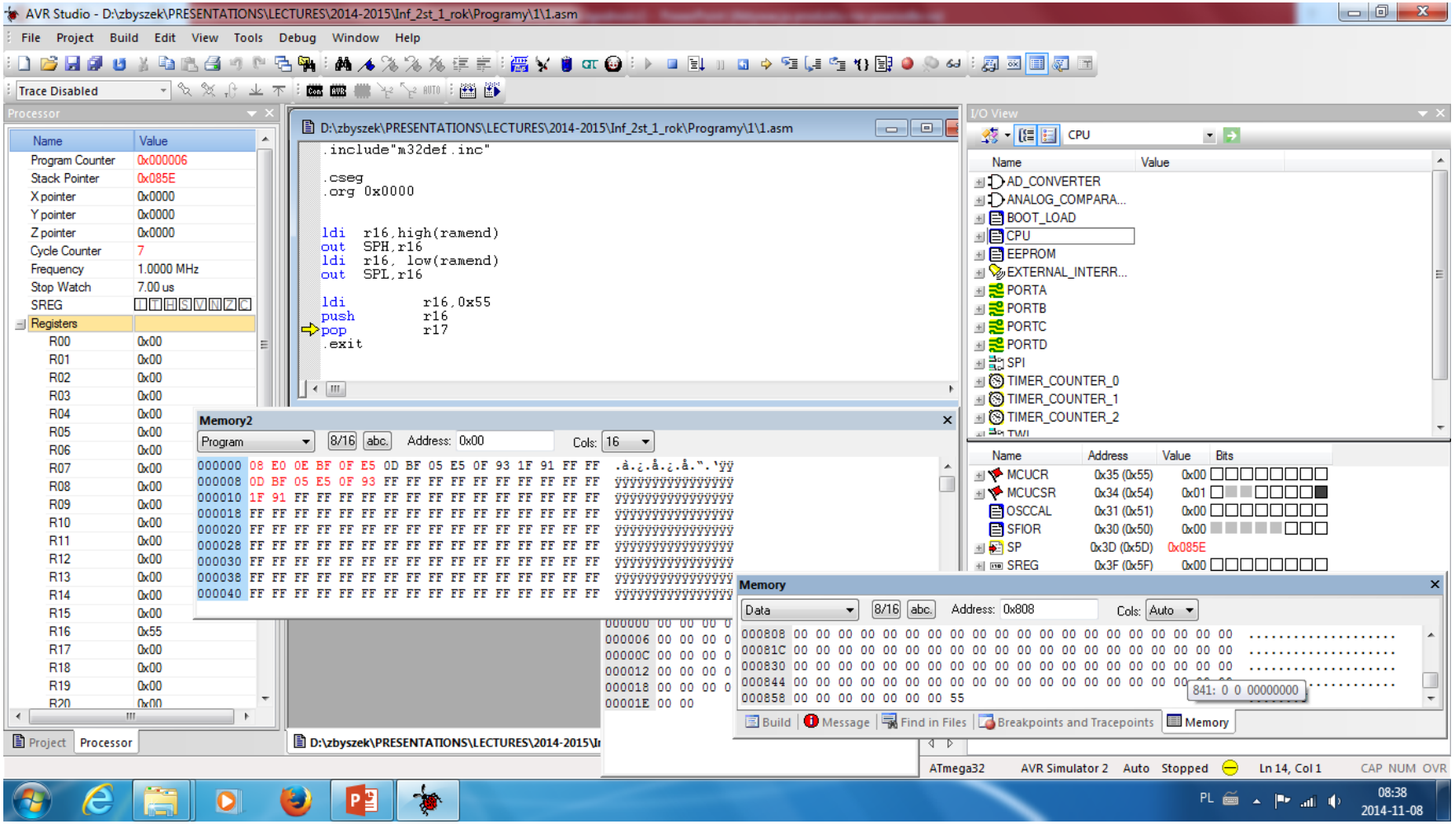


Emulatory

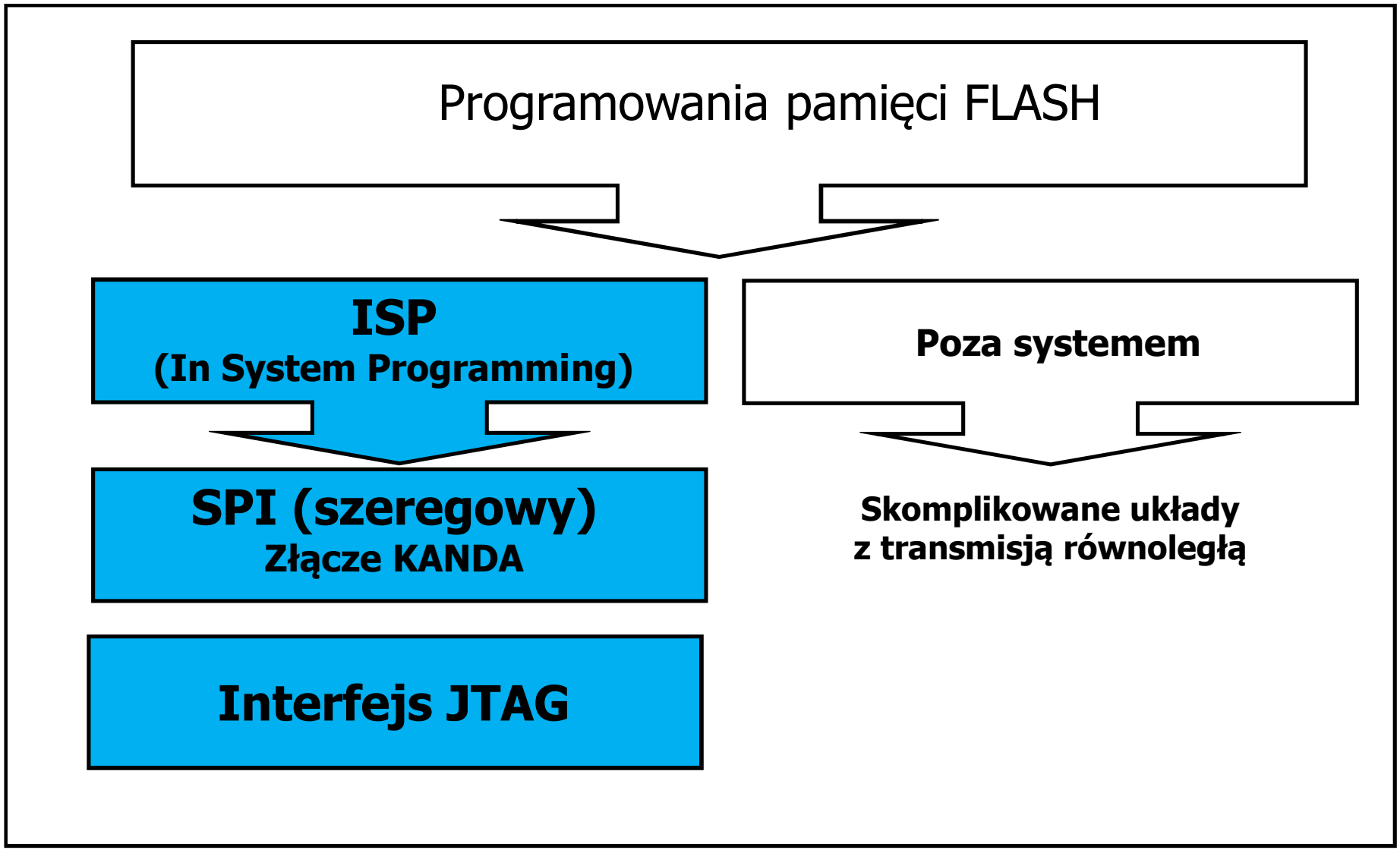

Welcome to AVR Studio 4
AVR Studio 4
Select debug platform and device
Debug platform:
AVR Dragon
AVR ONE!
AVR Simulator
AVR Simulator 2
ICE200
ICE40
ICE50
JTAG ICE
JTAGICE mkII
Device:
AT90CAN128
ATmega128
ATmega16
ATmega162
ATmega169
ATmega32
ATmega323
ATmega64
AT86RF401
AT89S51
AT89S52
Port:
Auto
Open platform options next time debug mode is entered
<< Back
Next >>
Finish
Cancel
Help

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Emulacja sprzętowa, Debager działa tak jak symulator AVR Simulator2

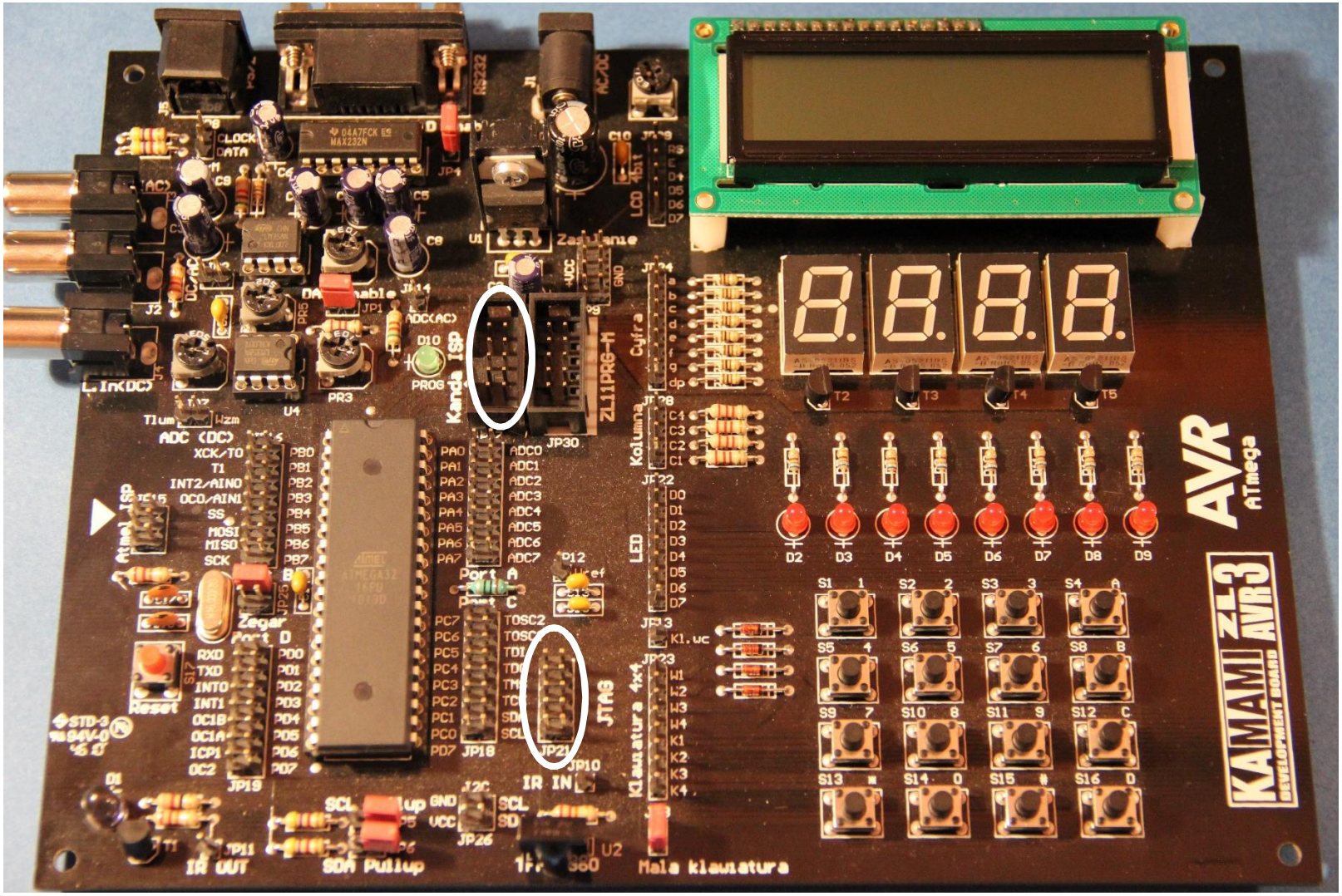


Interfejsy programowania i uruchomieniowe



WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Zestaw ZL3AVR



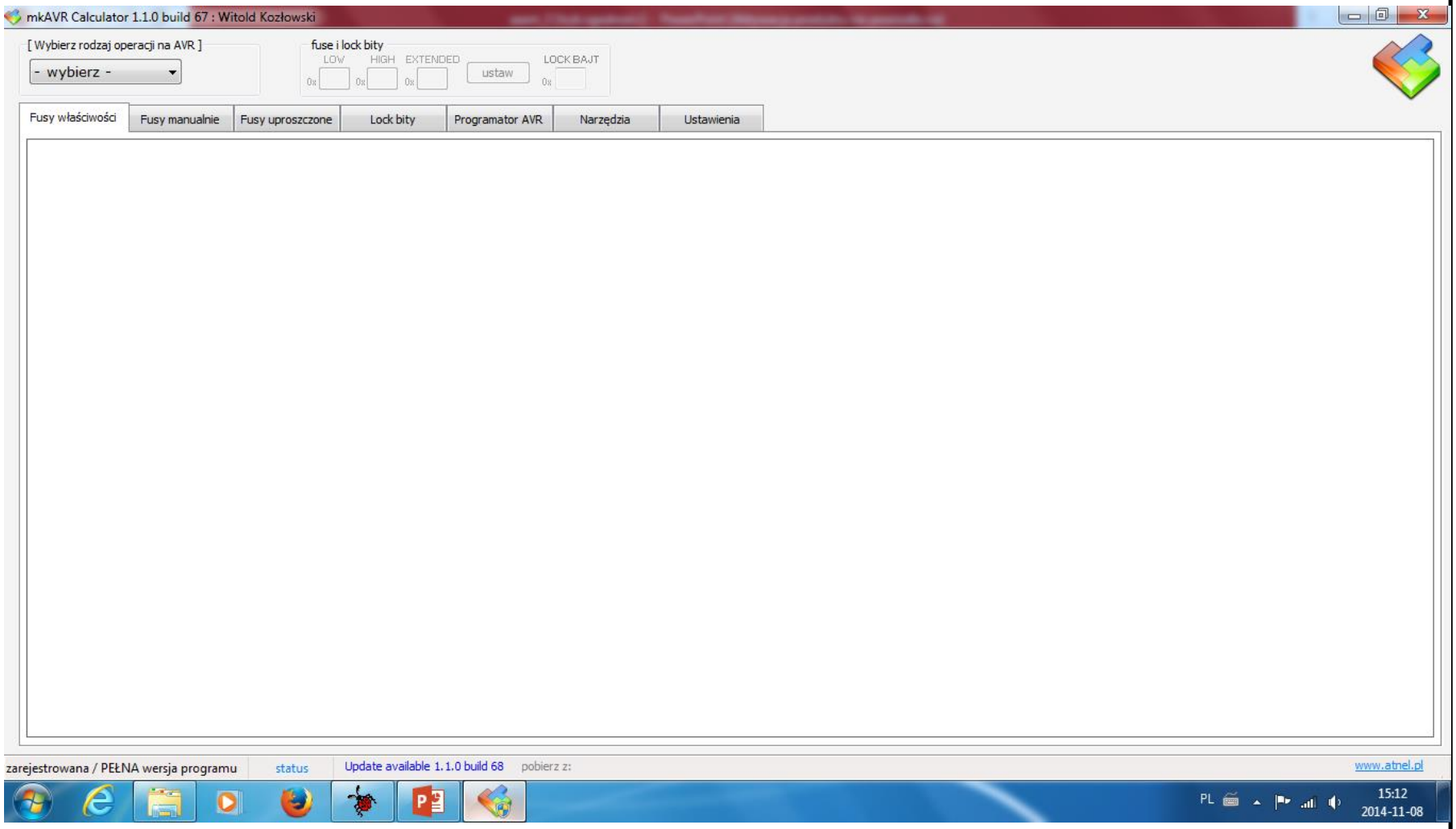
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP

SPI (szeregowy)
Złącze KANDA

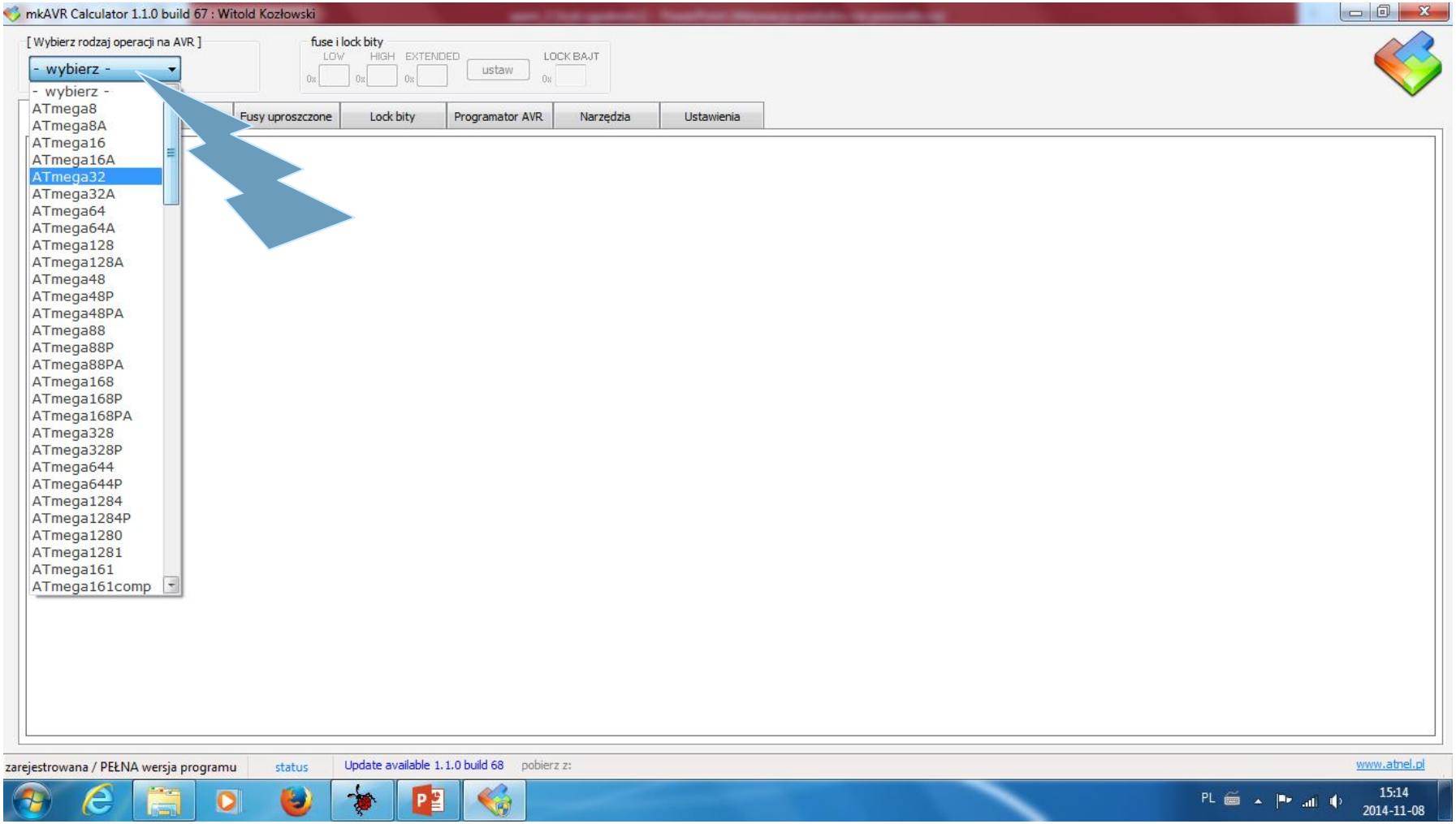
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP



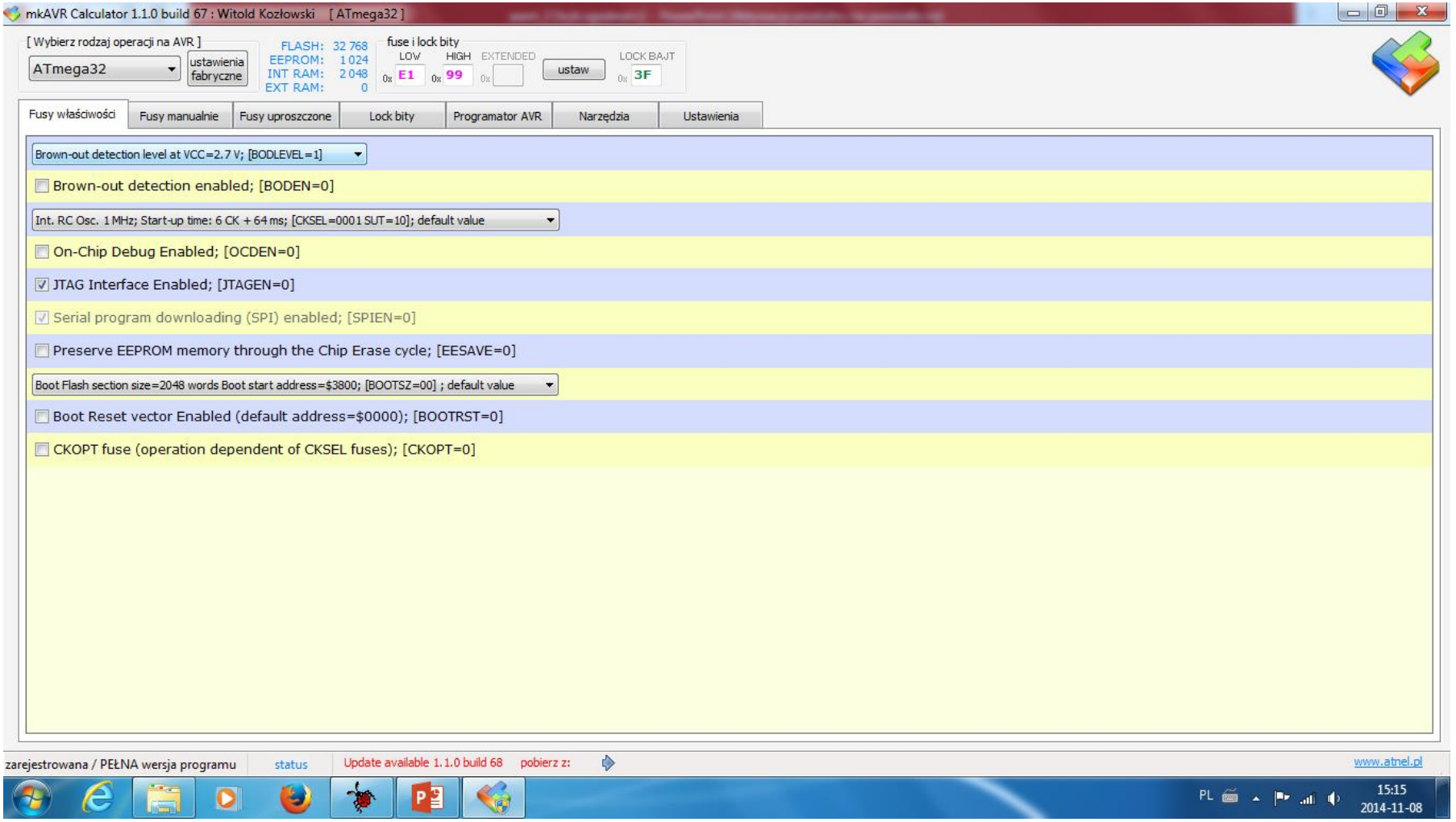
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP



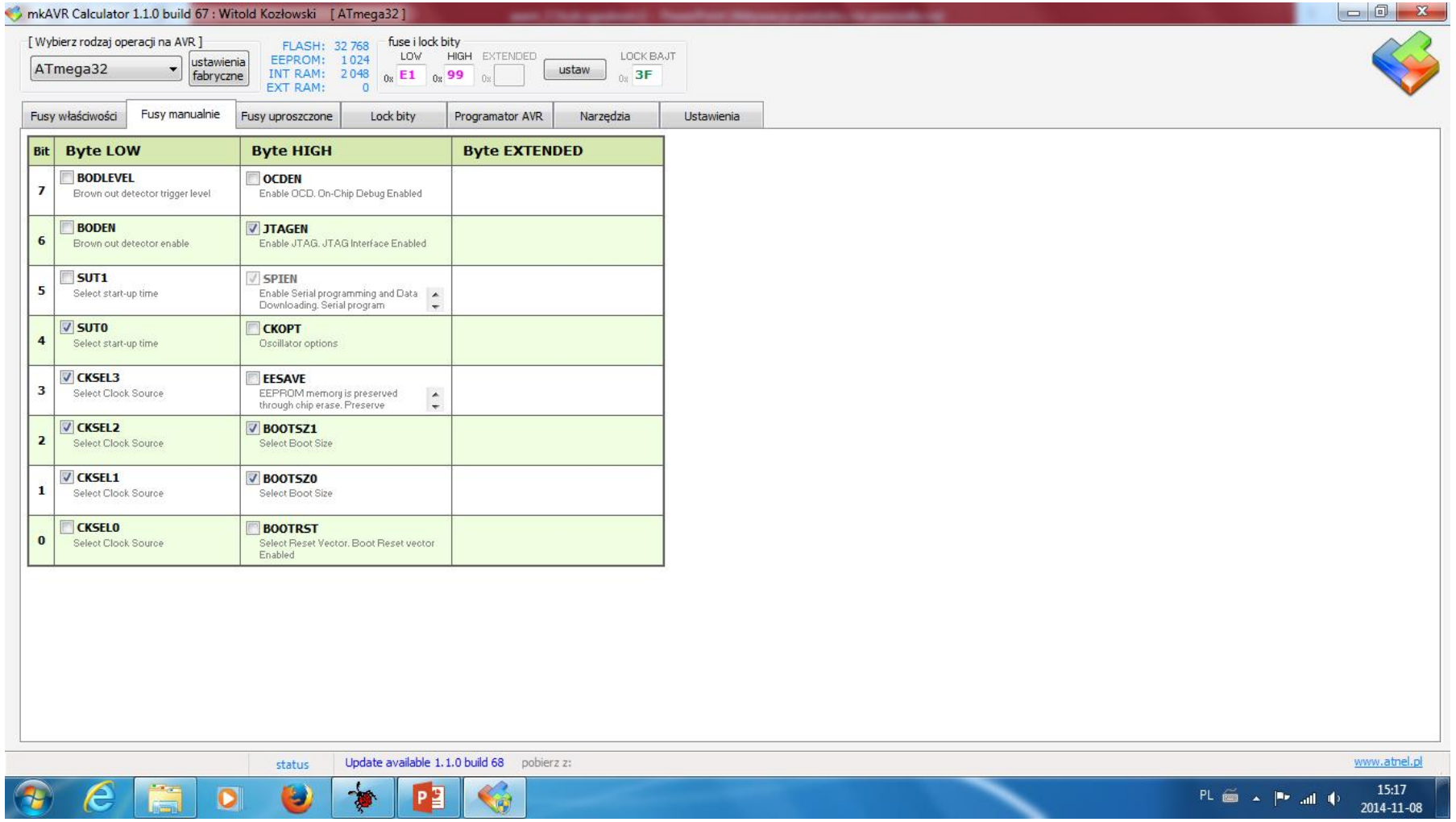
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP



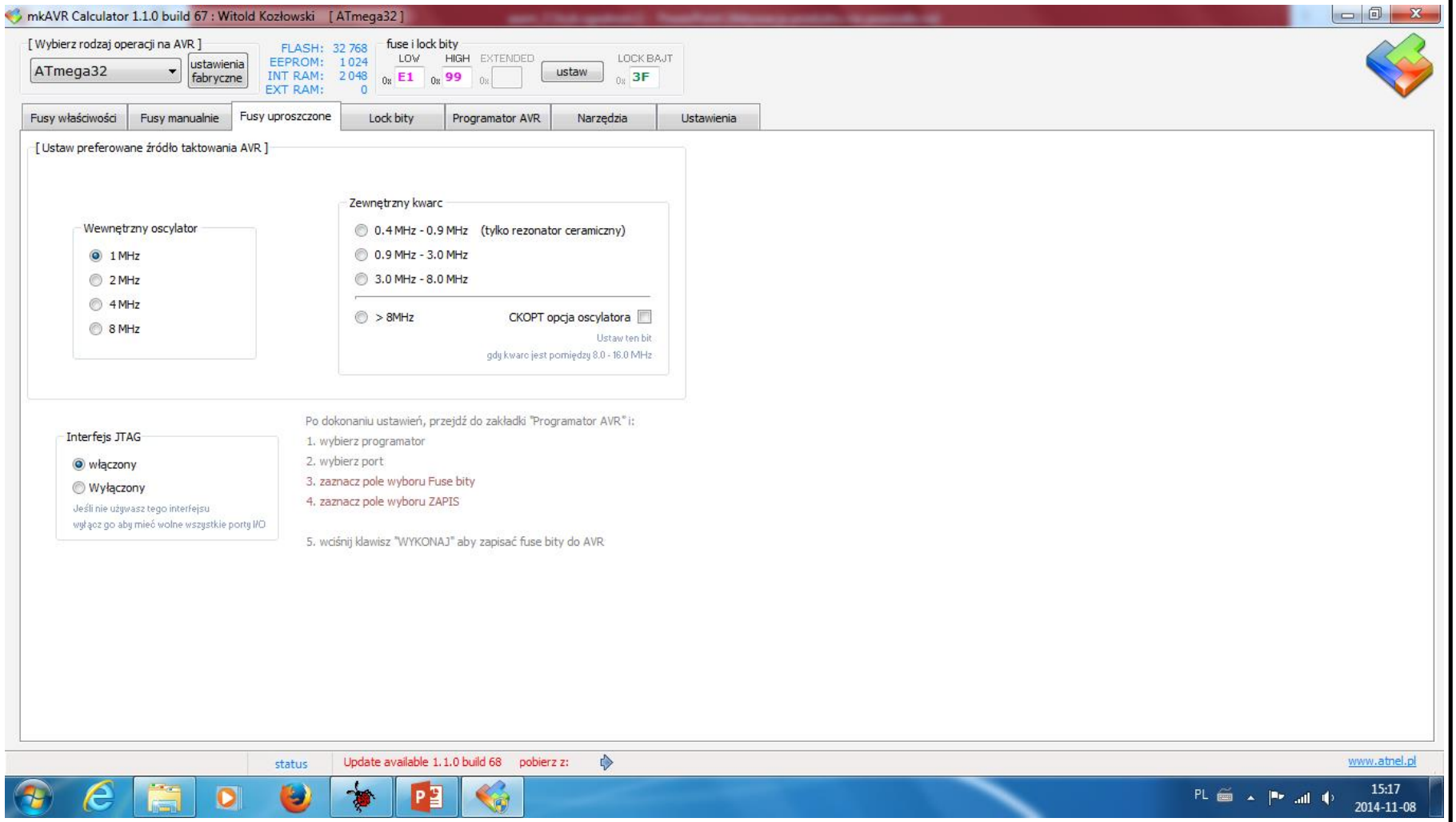
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP



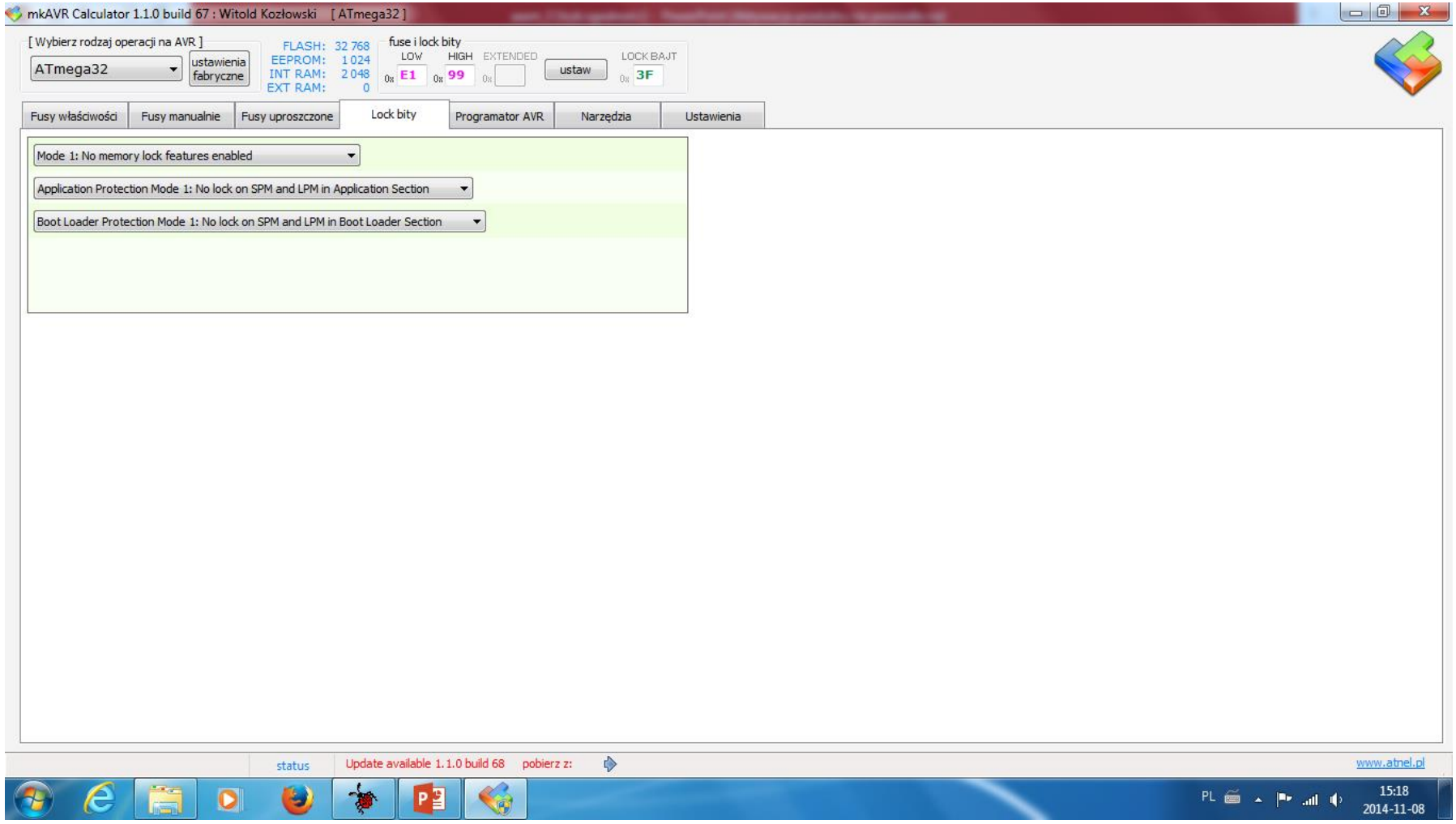
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP



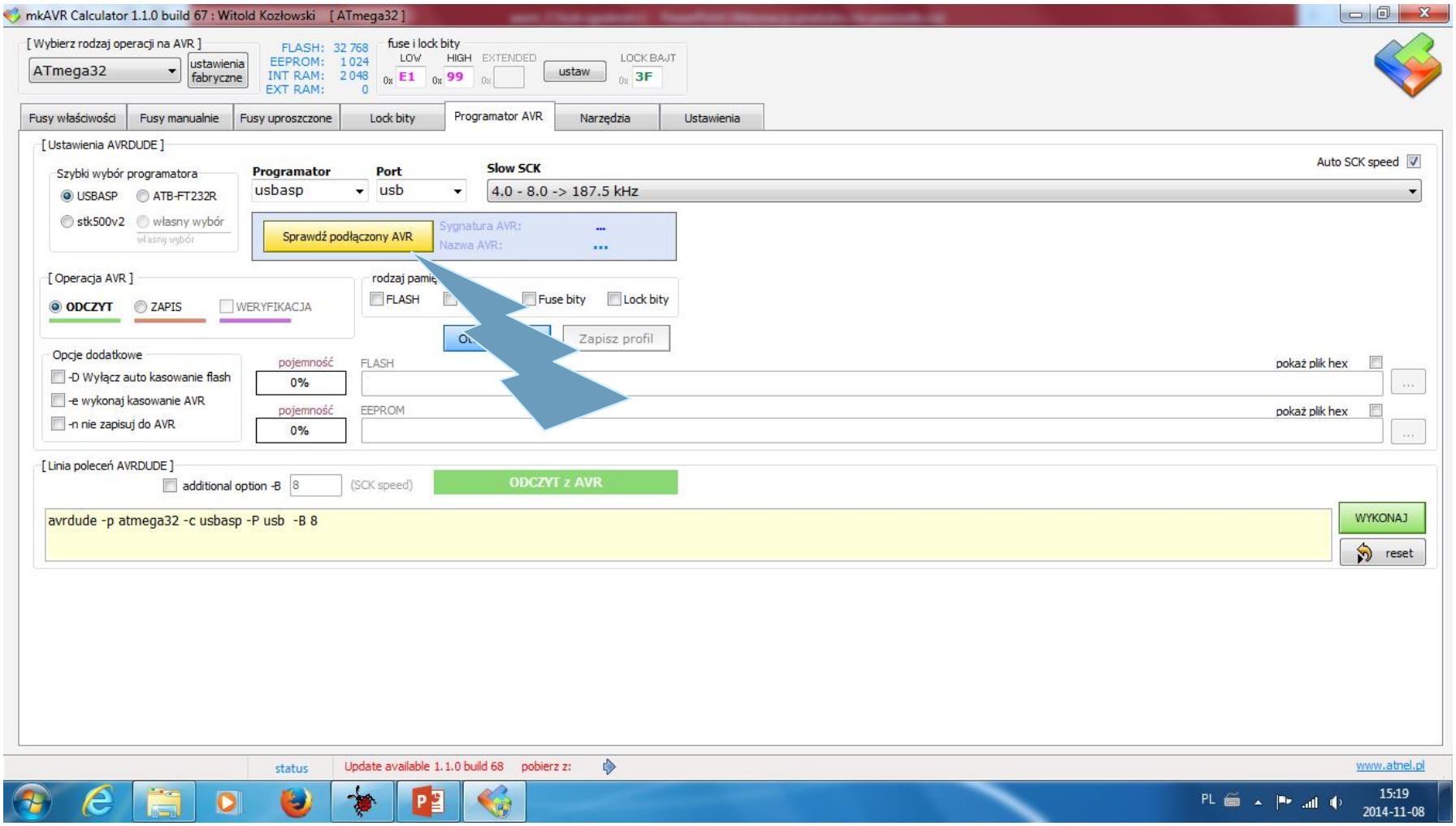
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP



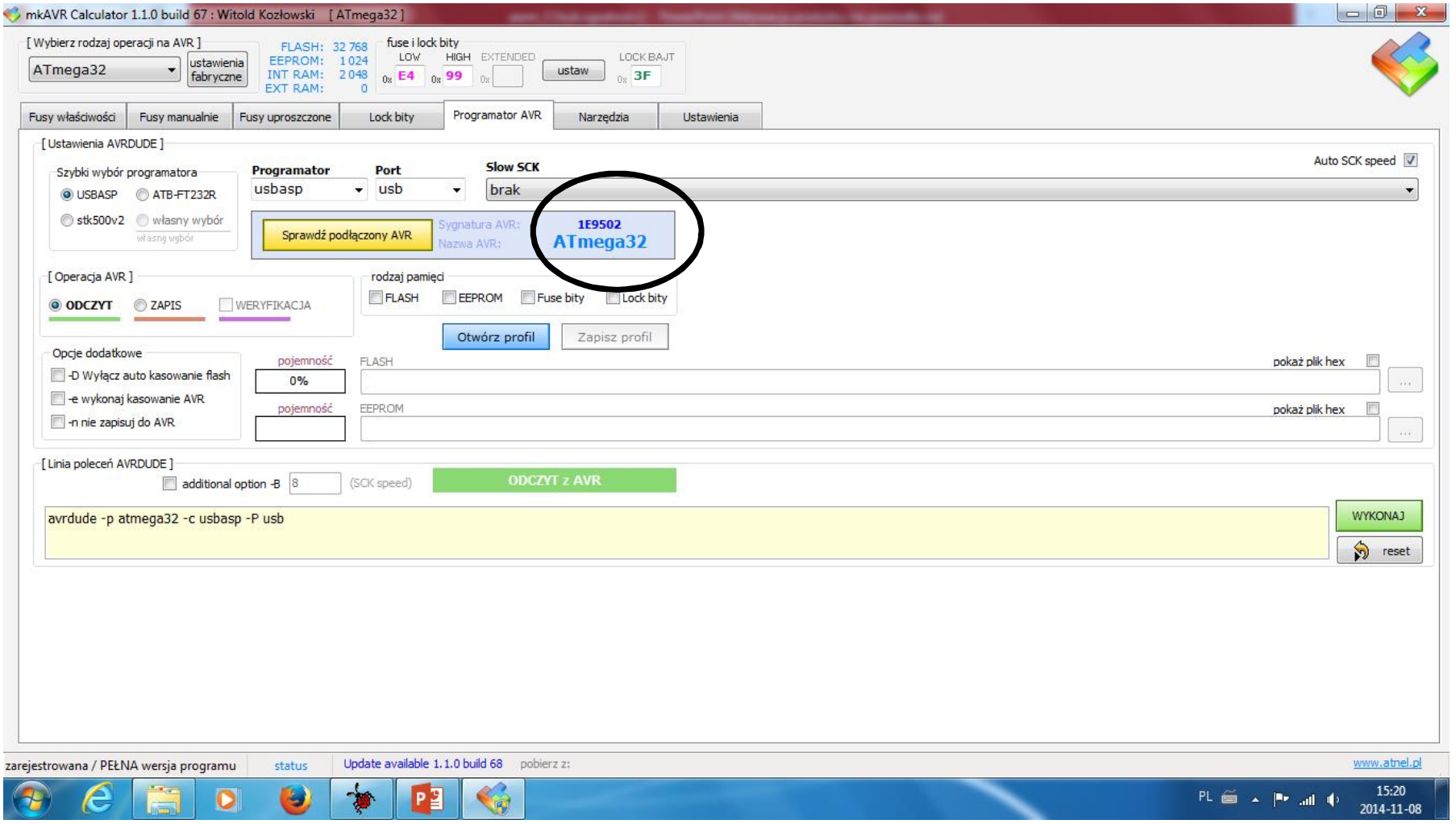
WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP



WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP



WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP