

Assembler

2



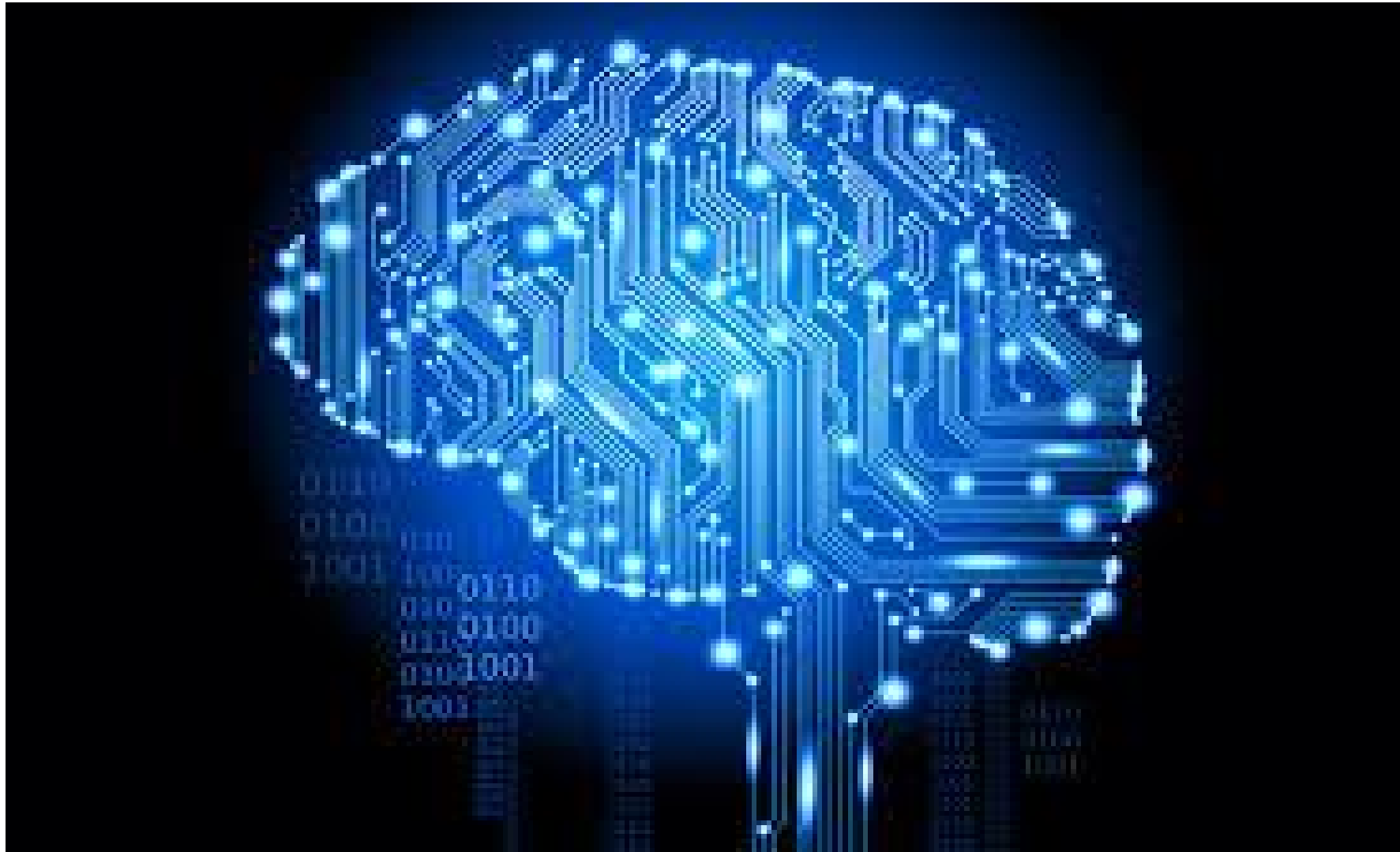
Kiedy asembler

- 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), jądra systemów (kernel)
 - Gdy piszemy oprogramowanie o podstawowym znaczeniu dla działania systemu (BIOS)



WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Architektura komputera





Architektura komputera



*Pamiętajmy, że za 20-30 lat nasze
dzisiejsze komputery będą wyglądały
równie śmiesznie*

First ever computer ENIAC, 1946



Architektura komputera

*Nie jesteśmy dzisiaj w ogóle sobie
wyobrazić jak będą wyglądały.....*



SPL



Architektura von Neumana

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

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.



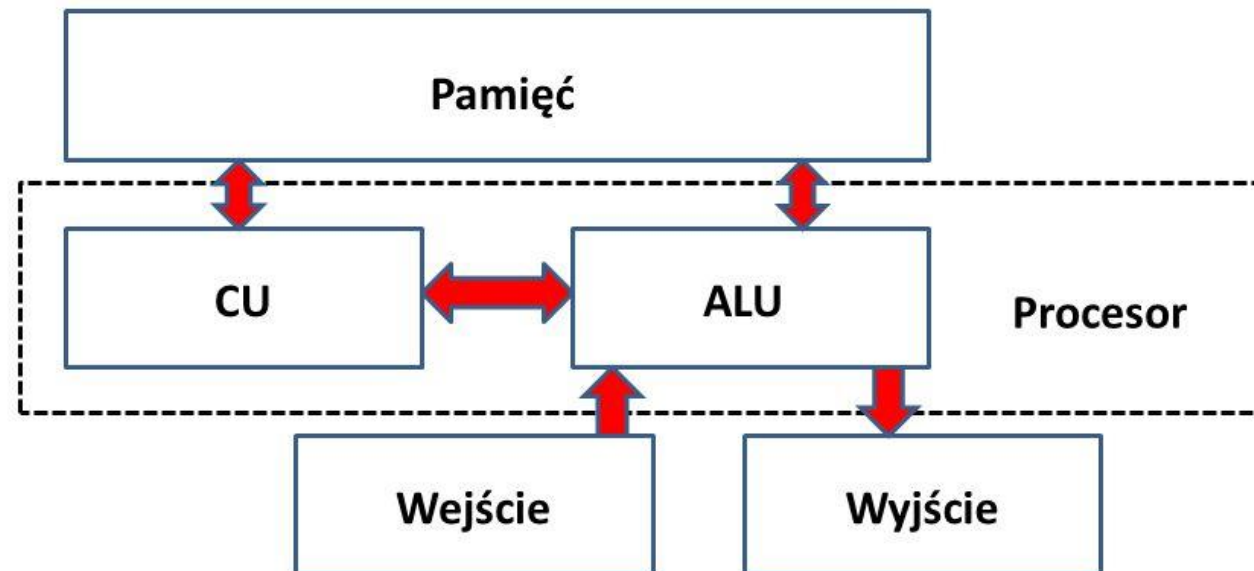
John von Neuman



Architektura von Neumanna

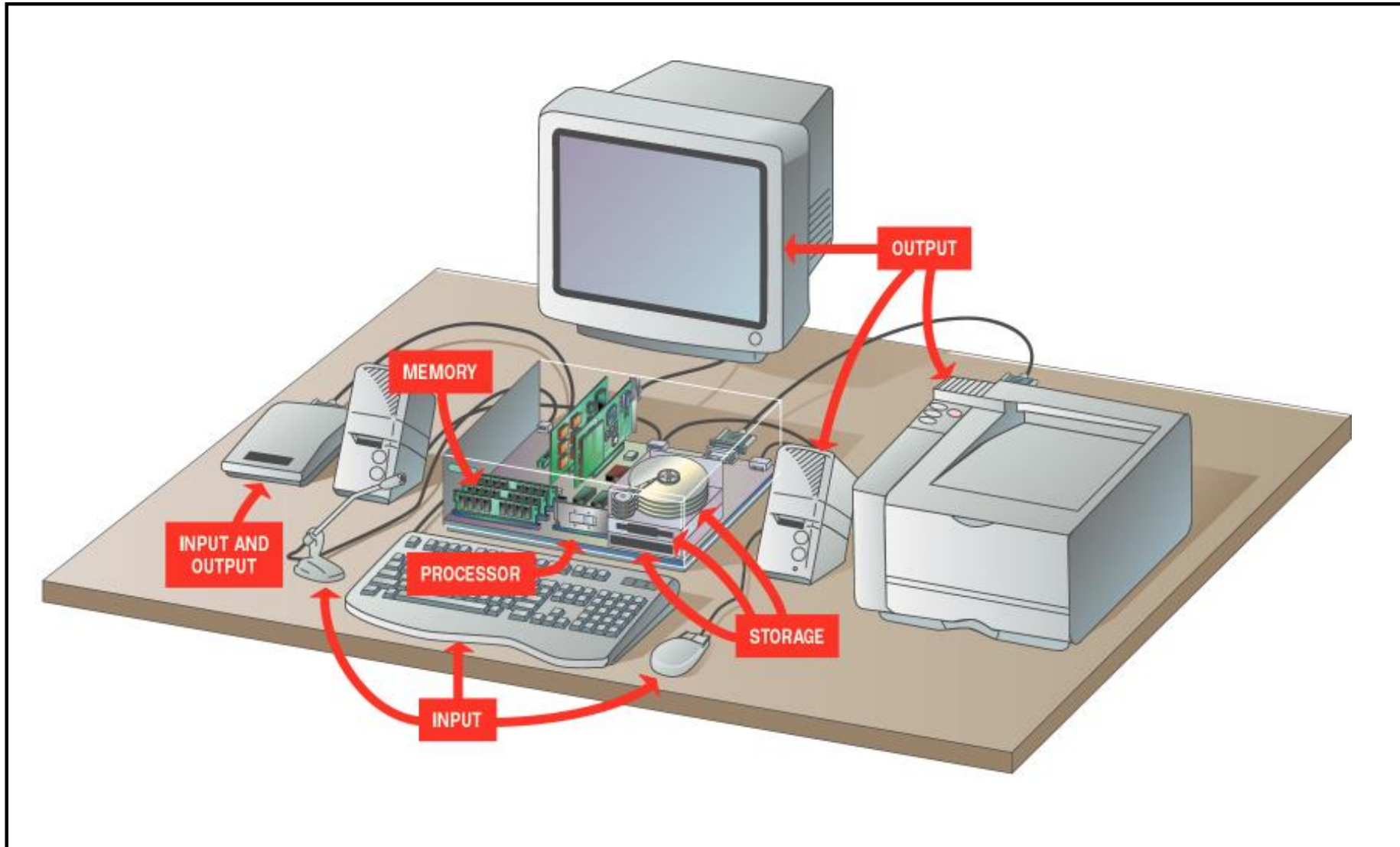
Komputer o architekturze von Neumanna składa się z trzech bloków:

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





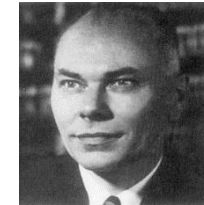
Komputer



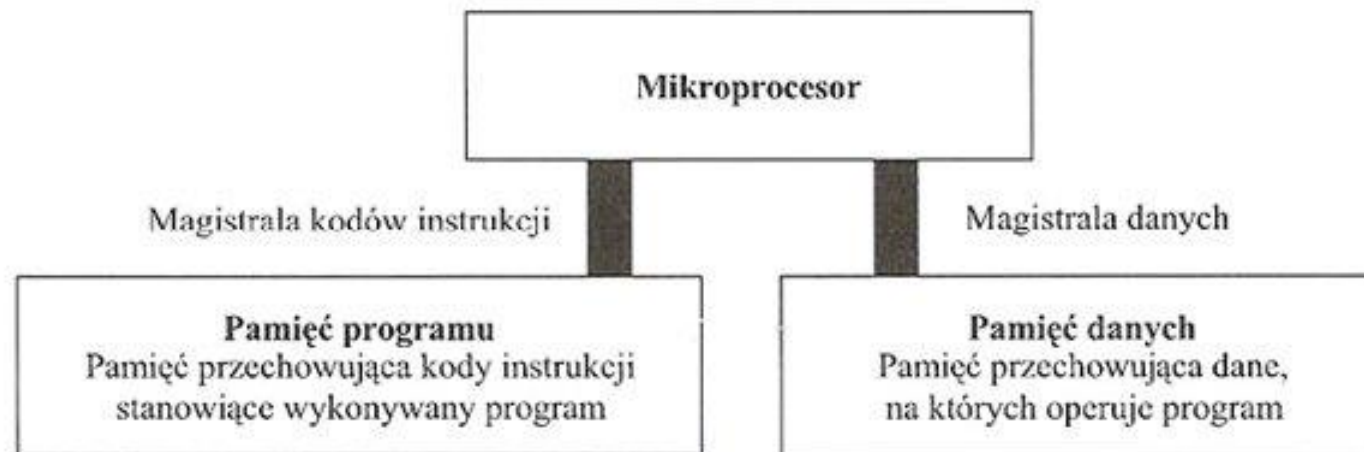


Architektura harwardzka

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.



Howard Hathaway
Aiken

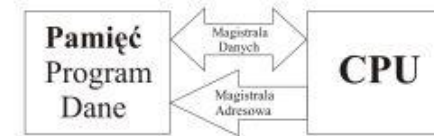




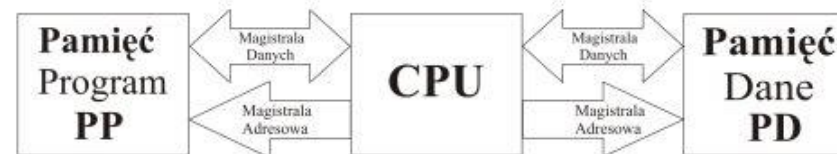
Architektura superharwardzka

Koncepcja nowej architektury

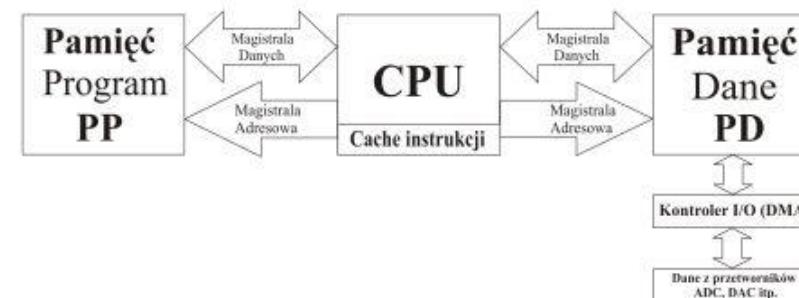
- **Architektura von Neumanna.**
(optym dla prostych operacji z danymi)



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

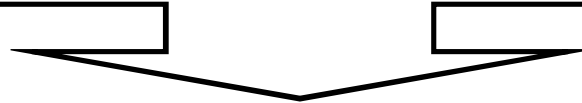


- **Architektura superharwardzka**
(poza podwójną pamięcią *Cache instrukcji* dla danych typu instrukcja-adres i DMA z pominięciem rejestrów)





Rodzaje Procesorów



CISC

Complex Instruction Set Computer

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



Rodzaje Procesorów

CISC

RISC

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



Rodzaje Procesorów



CISC

RISC

DSP

Digital Signal Processor

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



Rodzaje Procesorów

```
graph TD; A[Rodzaje Procesorów] --> B[CISC]; A --> C[RISC]; D[DSP]; E[HYBRIDE]; F[Hybride: procesory będące kombinacją CISC, RISC i DSP.];
```

CISC

RISC

DSP

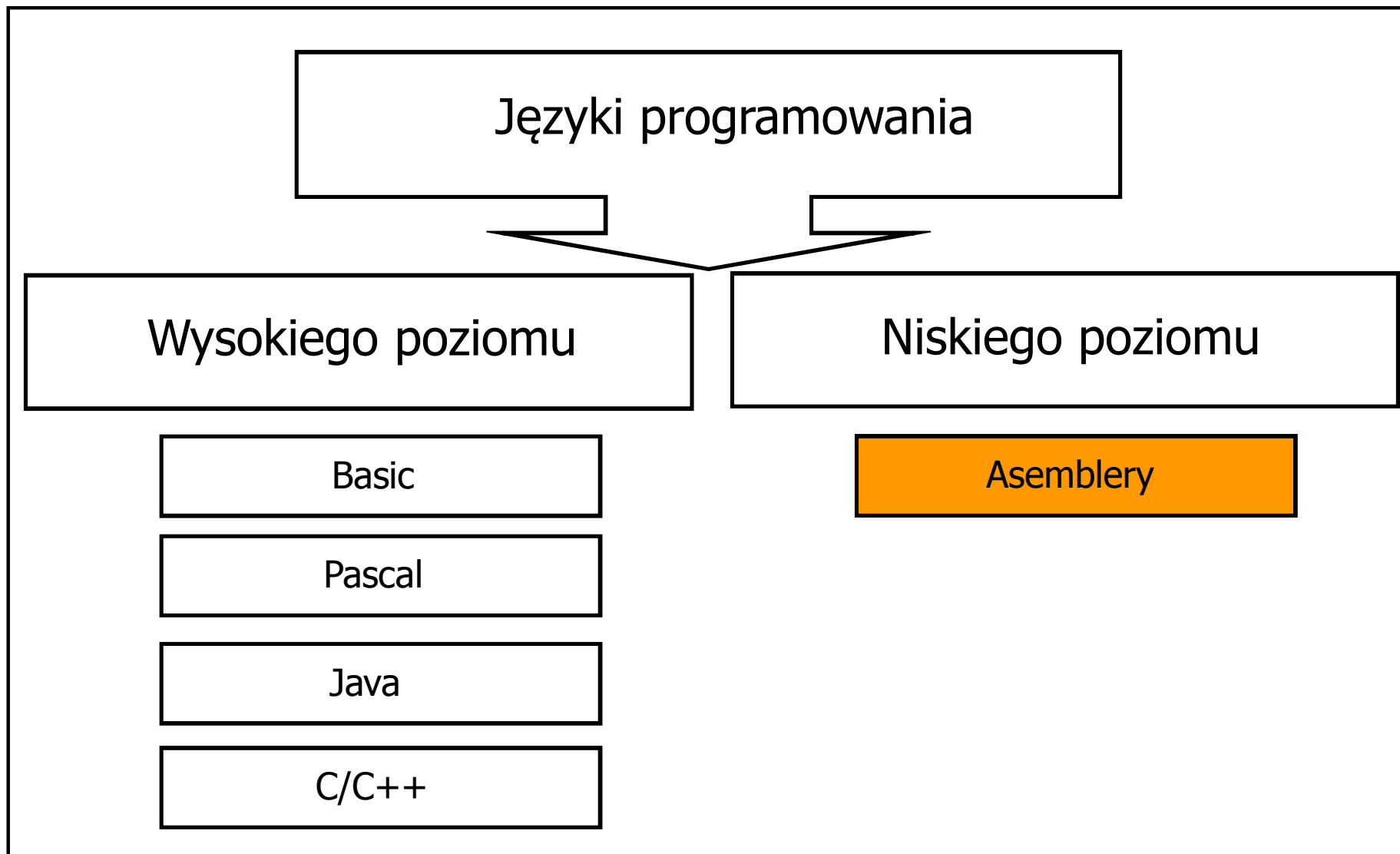
HYBRIDE

Hybride

procesory będące kombinacją CISC, RISC i DSP.

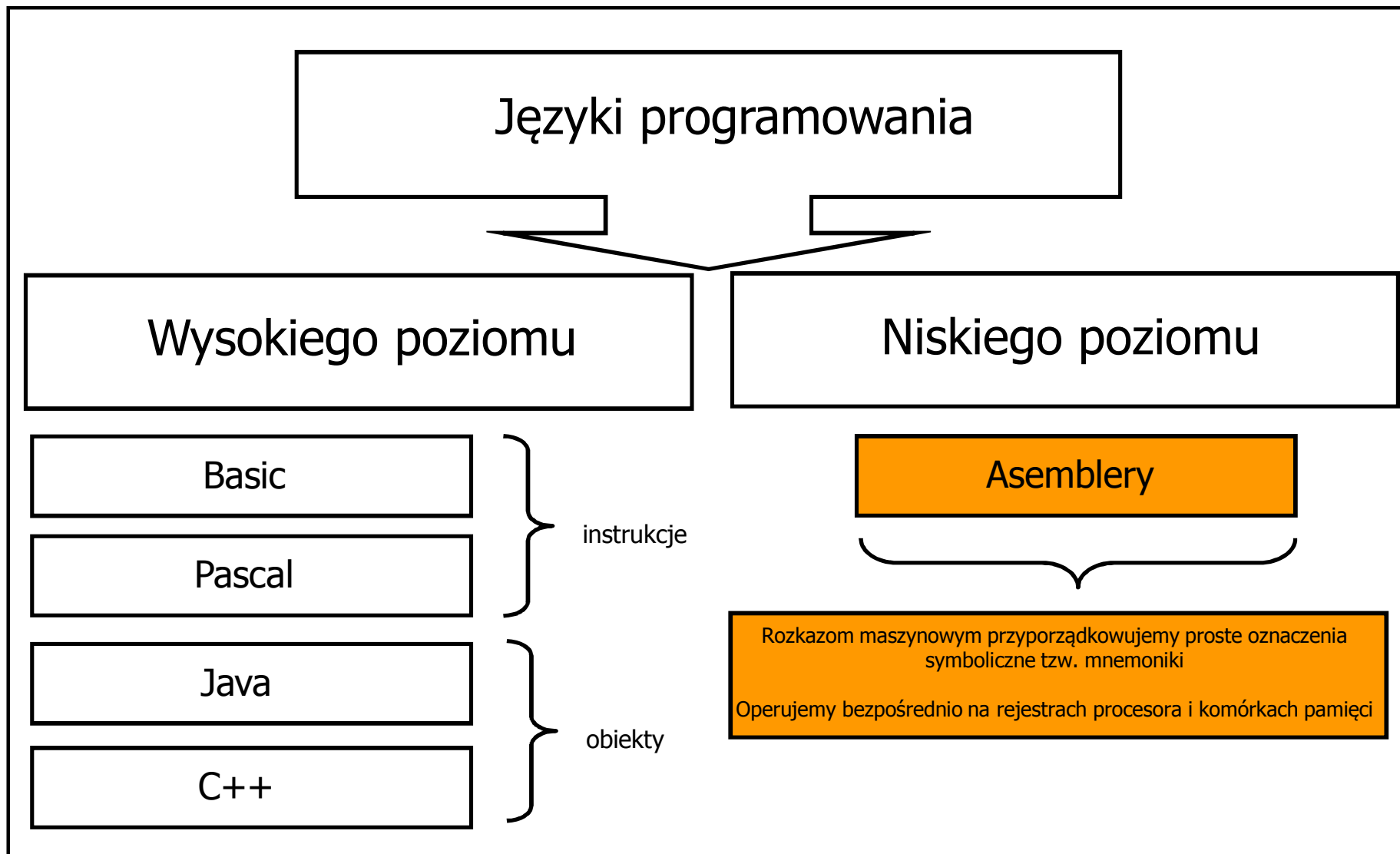


Języki Programowania



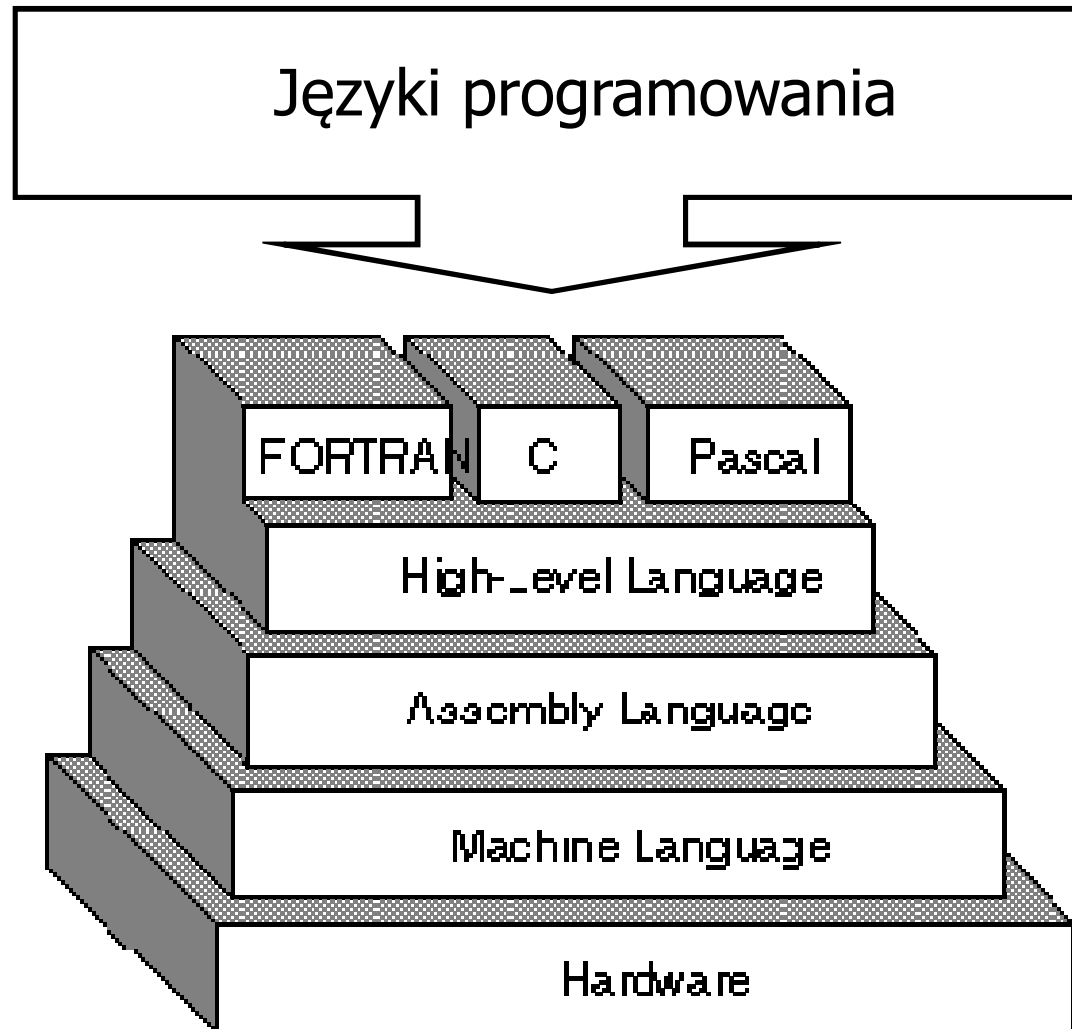


Języki Programowania





Języki Programowania





Cechy języka niskiego poziomu

Asemblery

Rozkazom maszynowym przyporządkujemy 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

Generacje języków niskiego poziomu

Pierwsza generacja
Język wewnętrzny (maszynowy)

Druga generacja
Język symboliczny (assembler)



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 (assembler)

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



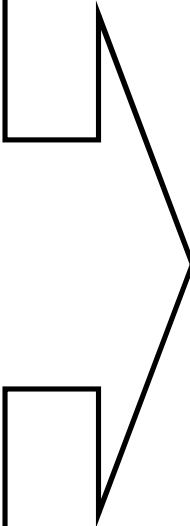
Języki programowania niskiego poziomu

rozkazy

j
ę
z
y
k

m
a
s
z
y
n
o
w
y

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



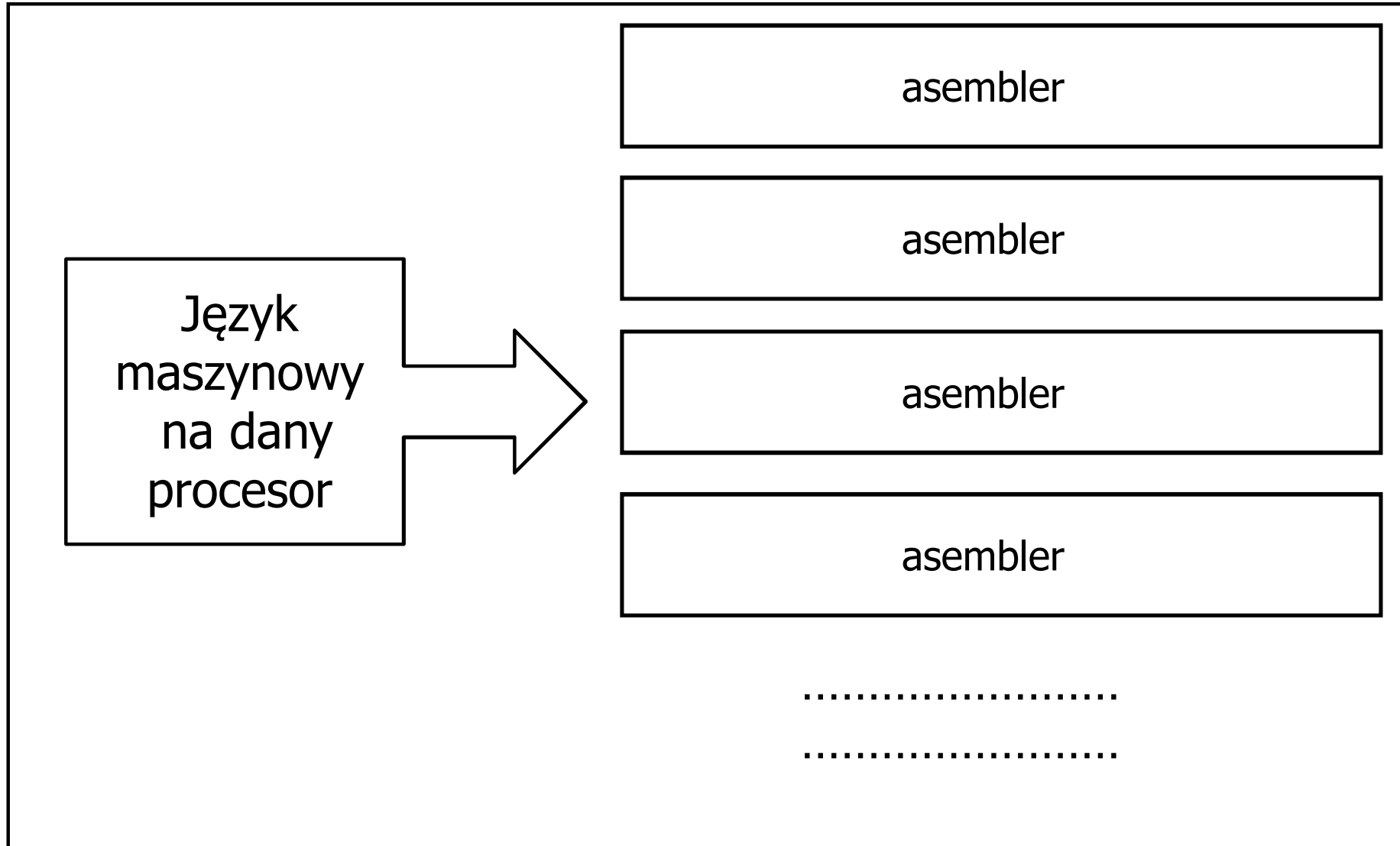
mnemoniki

a
s
e
m
b
l
e
r

```
.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,00h  
int 21h  
Witaj DB 'Witaj swiecie',0dh,0ah,'$'  
END start
```



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
```

j
ę
z
y
k

m
a
s
z
y
n
o
w
y

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



Wybór Procesora – rodzina mikrokontrolerów AVR firmy ATMEL



Atmel

**Rdzeń rodziny AVR
oparty jest na
architekturze RISC**

ATMEL
AVR

nazwa firmy produkującej rodzinę mikrokontrolerów AVR
Advanced Virtual RISC

ATtiny –

układy z rdzeniem AVR w mniejszych obudowach z małą liczbą peryferii przeznaczone do prostych zadań

ATmega –

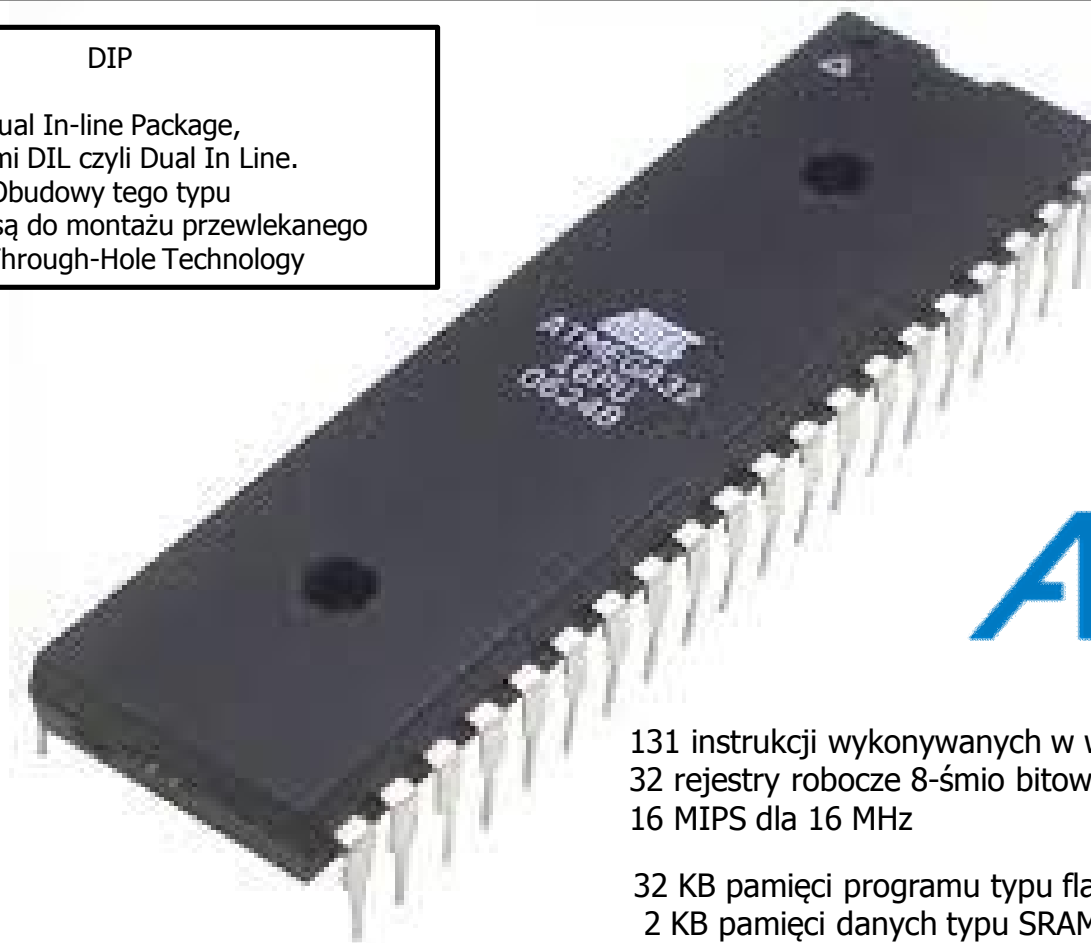
układy z rdzeniem AVR charakteryzujące się bogatym wyposażeniem takim jak kilka linii we/wy, przetworniki A/C, C/A, timery



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

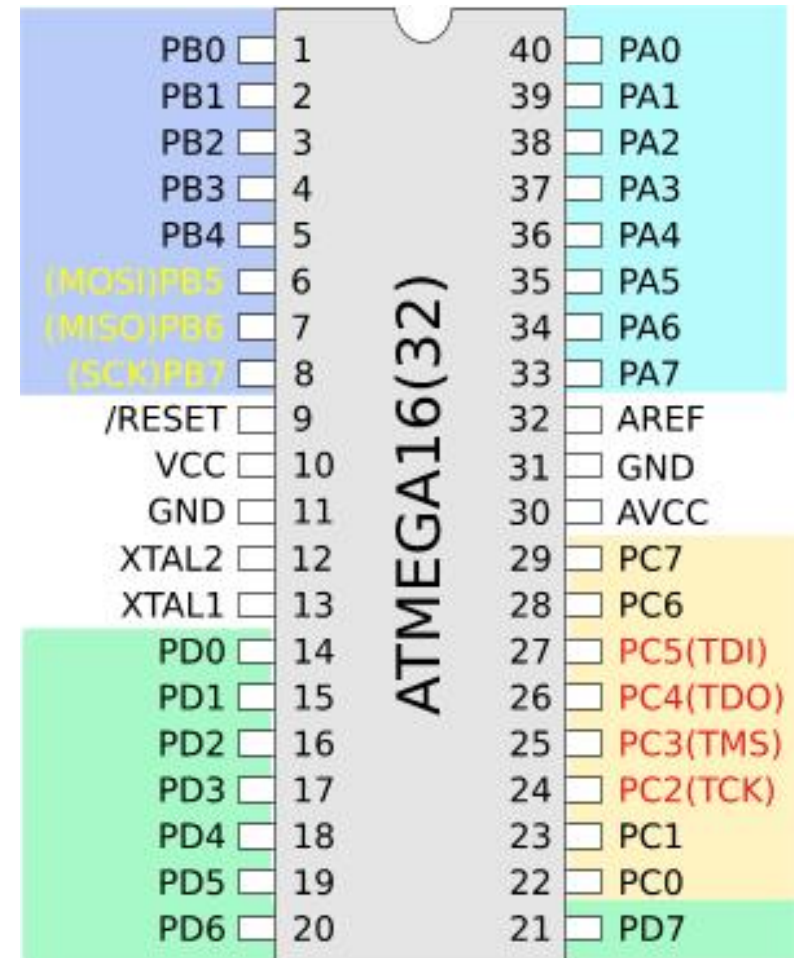
32 KB pamięci programu typu flash - programowanie metodą ISP
2 KB pamięci danych typu SRAM
1 KB EEPROM

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



Mikrokontroler ATMEGA 32

(XCK/T0) PB0	1	40	PA0 (ADC0)
(T1) PB1	2	39	PA1 (ADC1)
(INT2/AIN0) PB2	3	38	PA2 (ADC2)
(OC0/AIN1) PB3	4	37	PA3 (ADC3)
(SS) PB4	5	36	PA4 (ADC4)
(MOSI) PB5	6	35	PA5 (ADC5)
(MISO) PB6	7	34	PA6 (ADC6)
(SCK) PB7	8	33	PA7 (ADC7)
RESET	9	32	AREF
VCC	10	31	GND
GND	11	30	AVCC
XTAL2	12	29	PC7 (TOSC2)
XTAL1	13	28	PC6 (TOSC1)
(RXD) PD0	14	27	PC5 (TDI)
(TXD) PD1	15	26	PC4 (TDO)
(INT0) PD2	16	25	PC3 (TMS)
(INT1) PD3	17	24	PC2 (TCK)
(OC1B) PD4	18	23	PC1 (SDA)
(OC1A) PD5	19	22	PC0 (SCL)
(ICP1) PD6	20	21	PD7 (OC2)





**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 × 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 16 MIPS Throughput at 16MHz
 - On-chip 2-cycle Multiplier

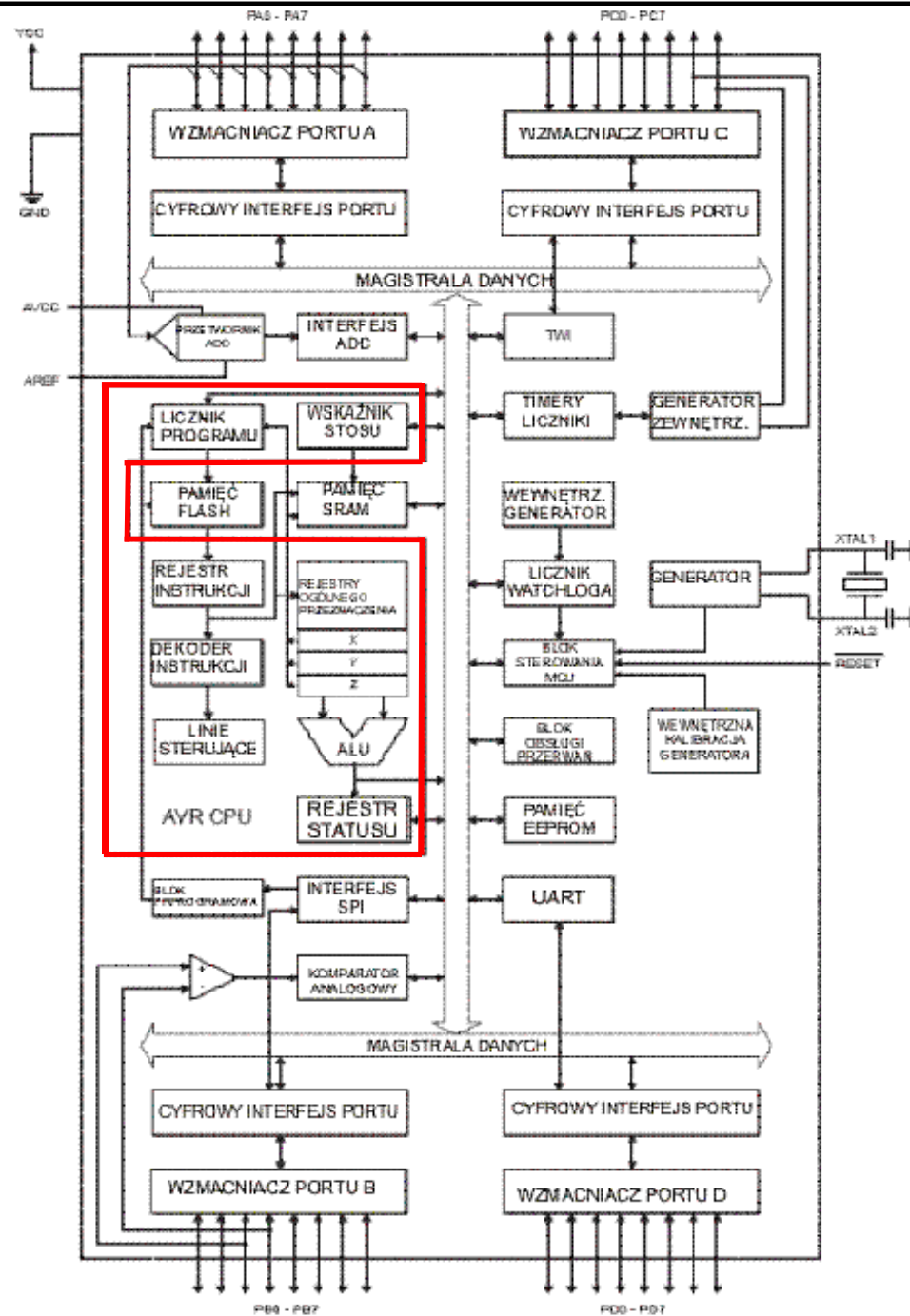
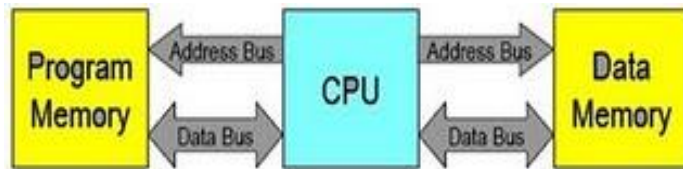
- **High Endurance Non-volatile Memory segments**
 - 32Kbytes of In-System Self-programmable Flash program memory
 - 1024Bytes EEPROM
 - 2Kbytes Internal SRAM
 - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
 - Data retention: 20 years at 85°C/100 years at 25°C⁽¹⁾
 - Optional Boot Code Section with 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



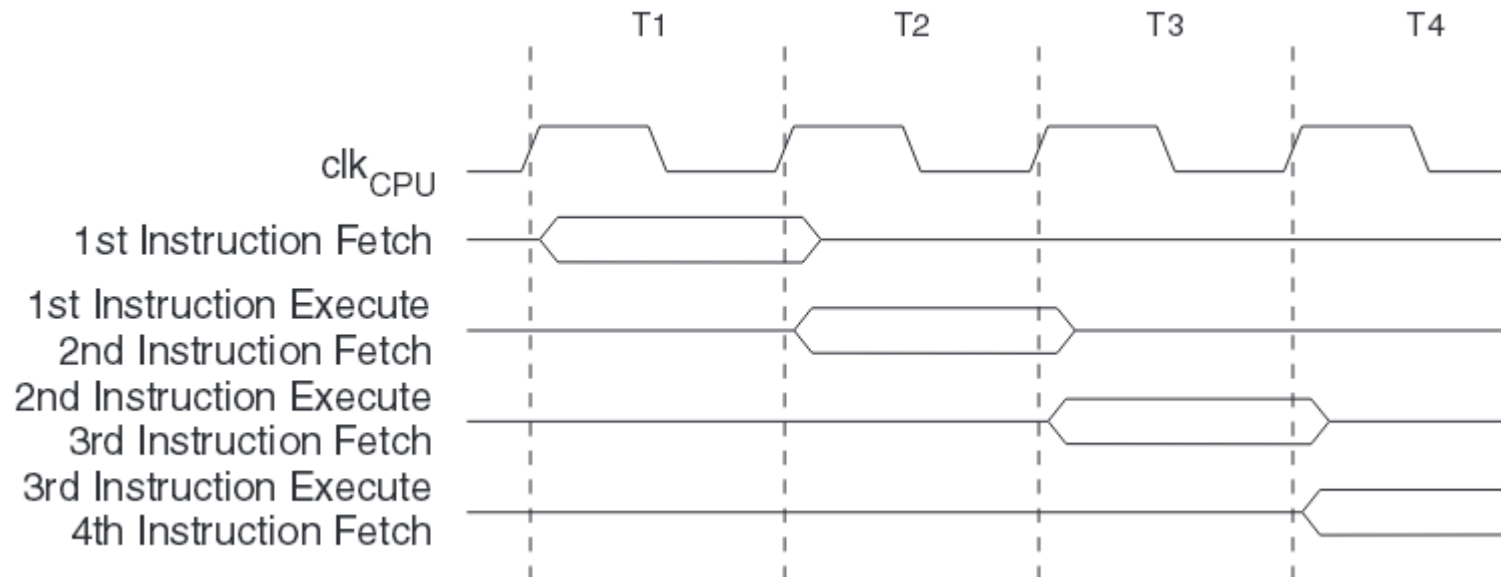
Mikrokontroler ATMEGA 32

Rdzeń mikrokontrolera AVR





Przetwarzanie potokowe (pipelining)

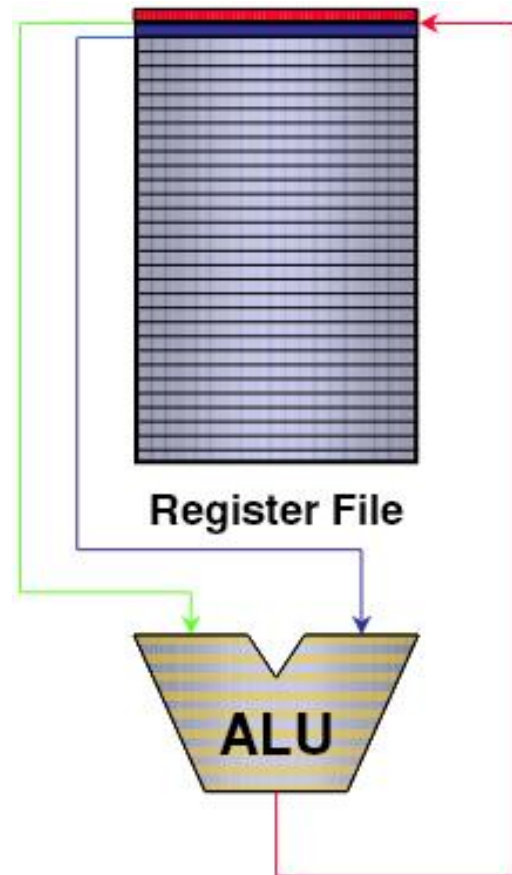


W AVR zastosowano przetwarzanie potokowe które polega na jednoczesnym wykonywaniu danej instrukcji i pobieraniu instrukcji następnej.

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



ALU- Arithmetic Logic Unit

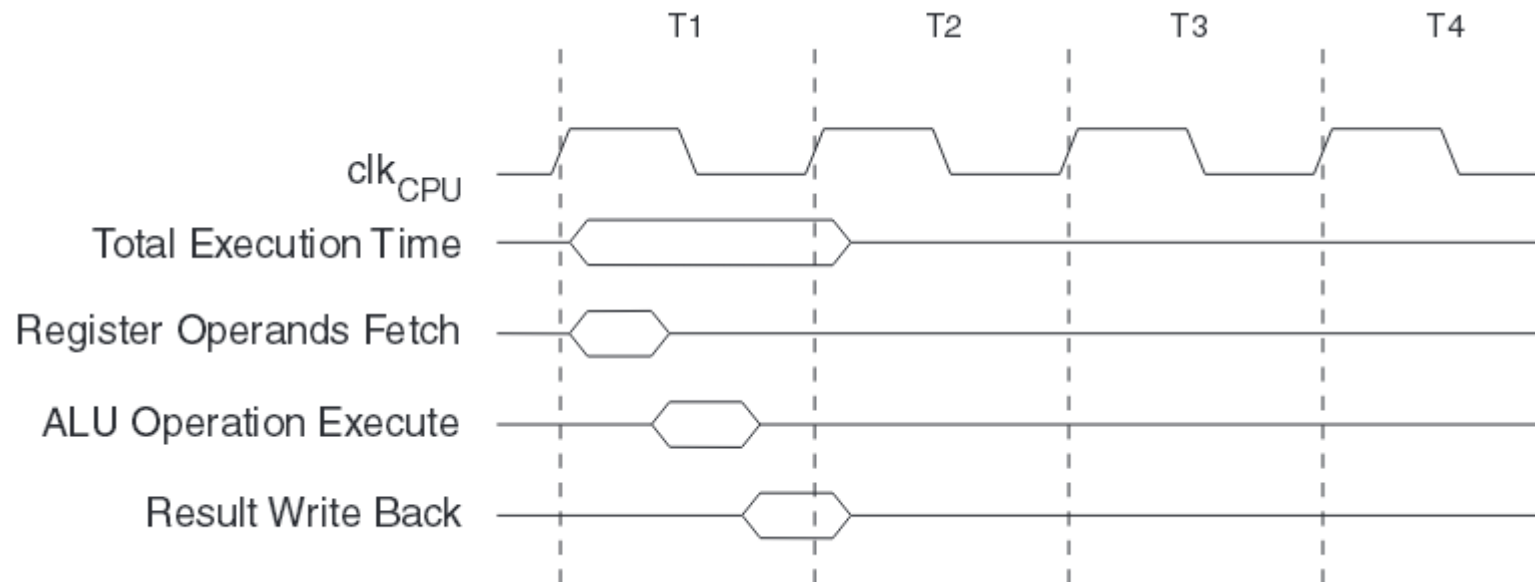
Jest bezpośrednio połączona z 32 ośmiobitowymi rejestrami

Wykonuje operacje w jednym cyklu zegarowym:

- " Arytmetyczne
- " Logiczne
- " Funkcje bitowe



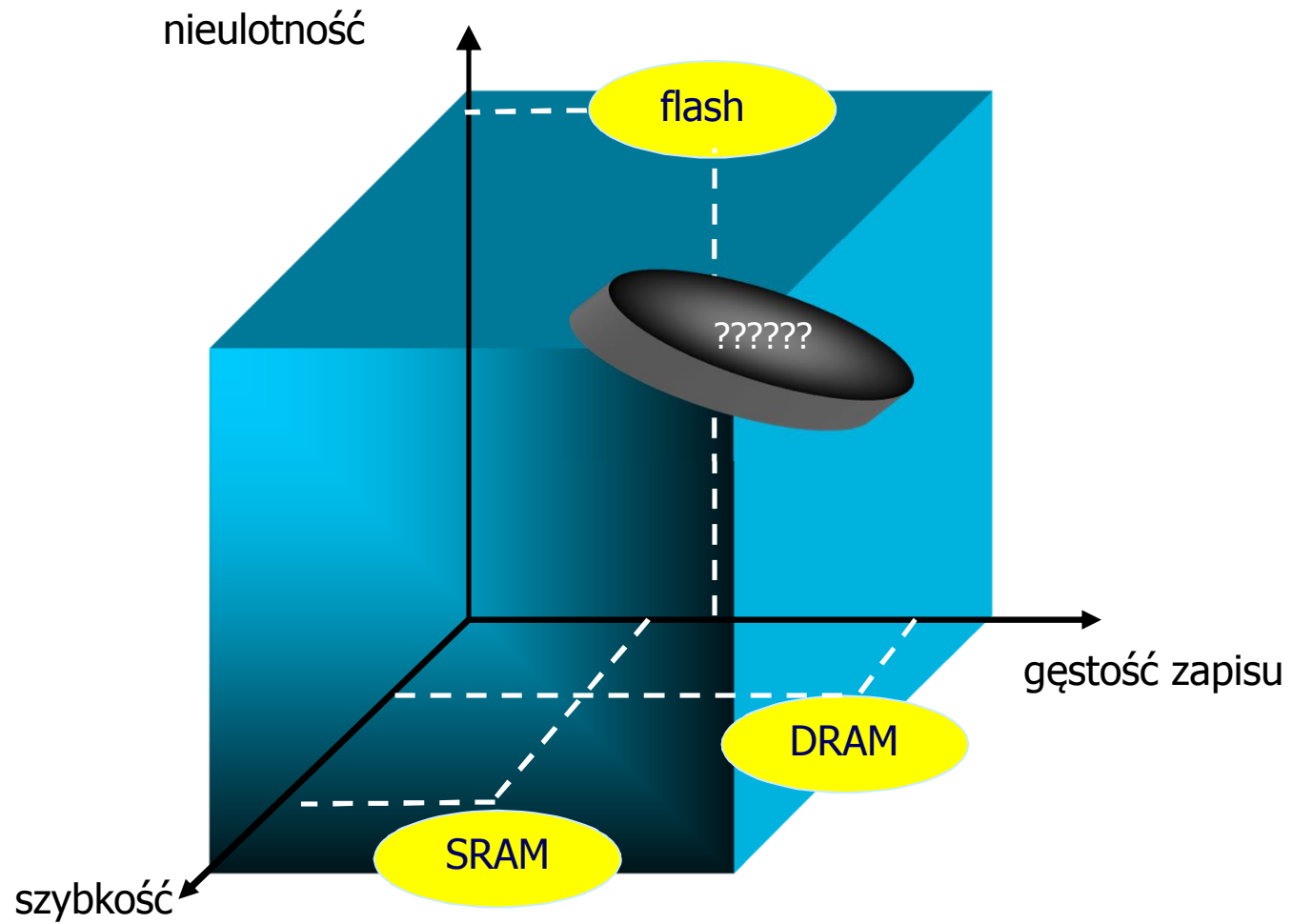
Pojedynczy cykl pracy ALU



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



Pamięć SRAM/DRAM/Flash





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 pamięć 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 pamięć typu **Flash**
32 Kbytes

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

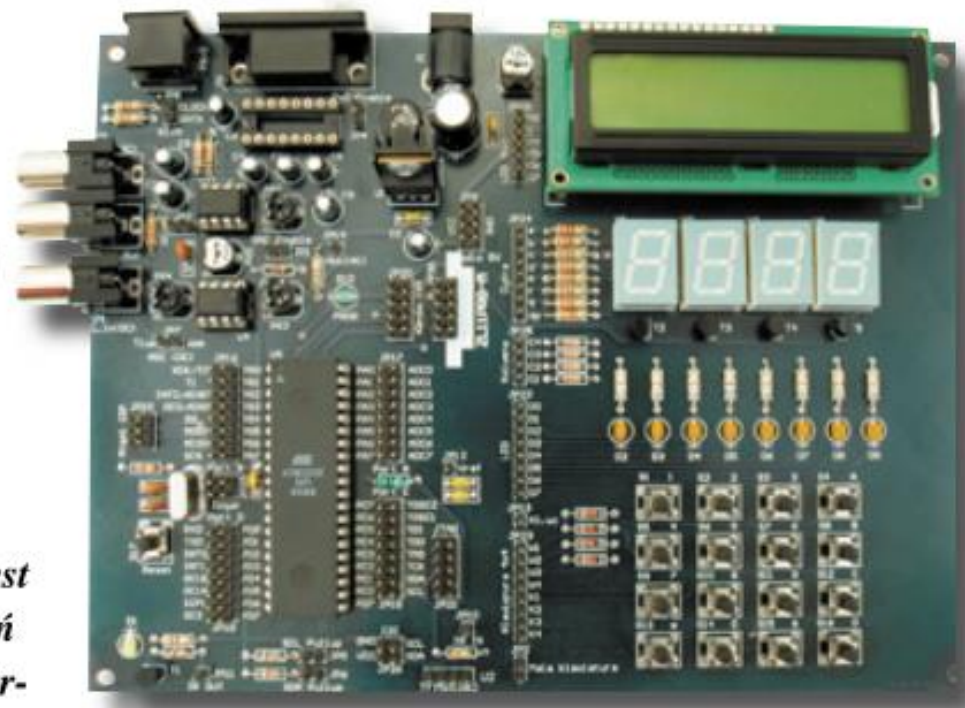




ZL3AVR

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.





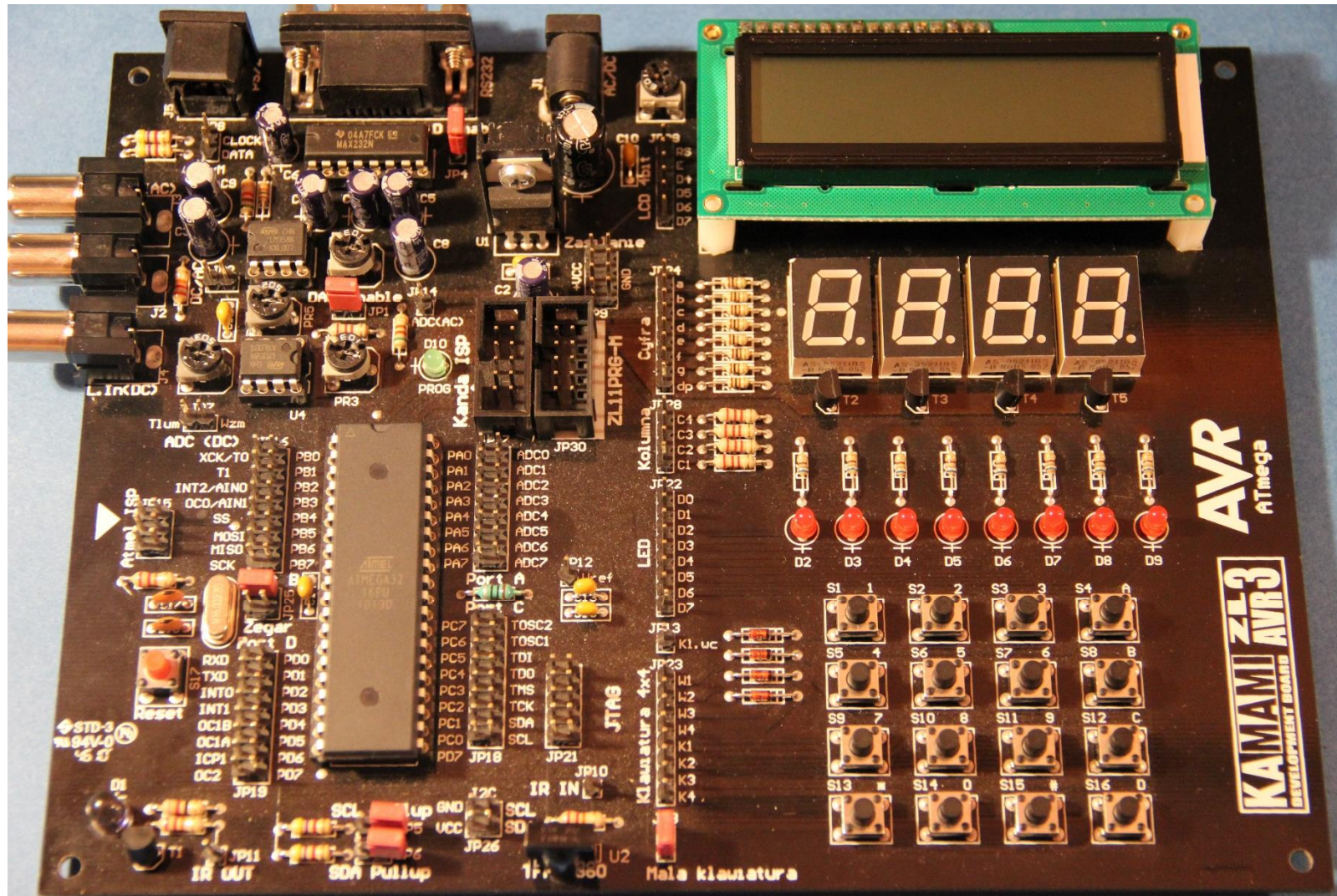
Zawartość zestawu ZL3AVR

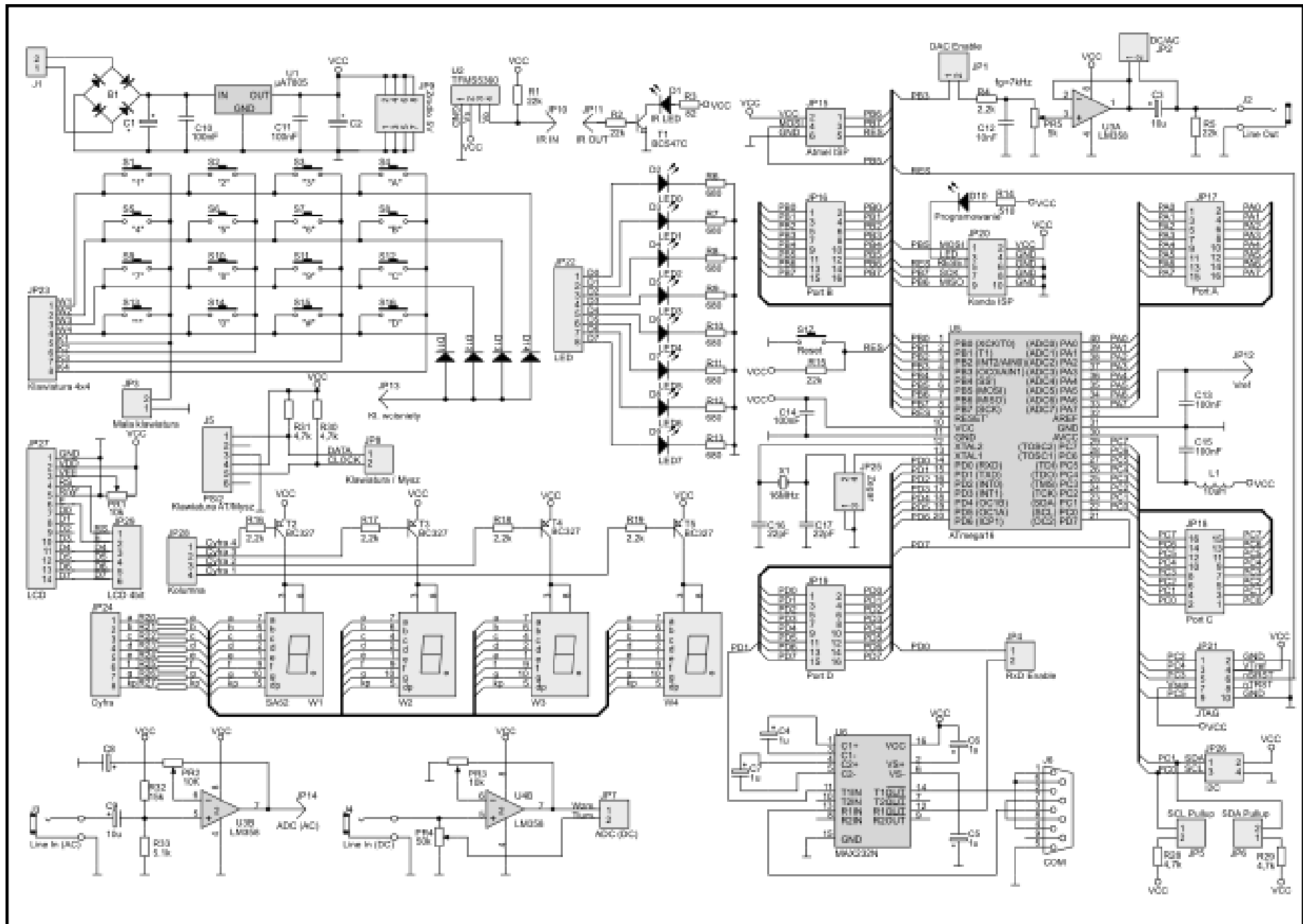
mikrokontroler ATmega32 (DIP40)
rezonator kwarcowy o częstotliwości nominalnej 16 MHz
16-przyciskowa klawiatura (4x4)
pole sygnalizacyjne na 8 diodach LED
4 wyświetlacze 7-segmentowe LED (sterowanie multipleksowe)
alfanumeryczny wyświetlacz LCD o organizacji 2x16 znaków
wyprowadzenie linii interfejsu 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

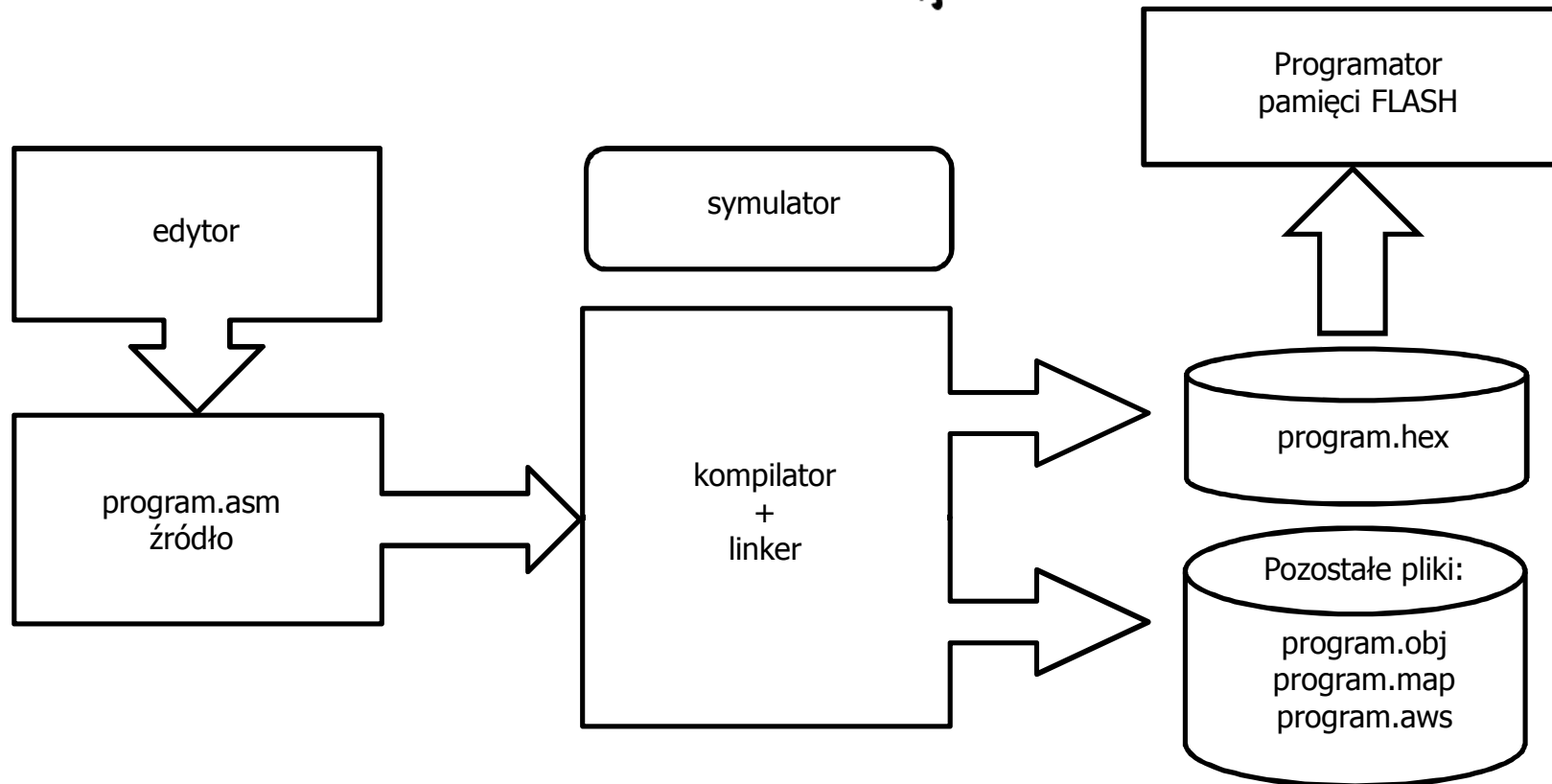
**Będziemy używali pakietu AVR Studio ATMEL
Dostępne są wyższe wersje**





Będziemy używali pakietu AVR Studio ATMEL

IDE – Integrated Development Environment





The screenshot displays the AVR Studio IDE interface. The main window shows assembly code for a test program. The Processor window on the left lists various registers and their values. The I/O View window on the right shows the state of various hardware modules. The Memory window at the bottom shows the memory dump.

Będziemy używali pakietu AVR Studio ATMEL

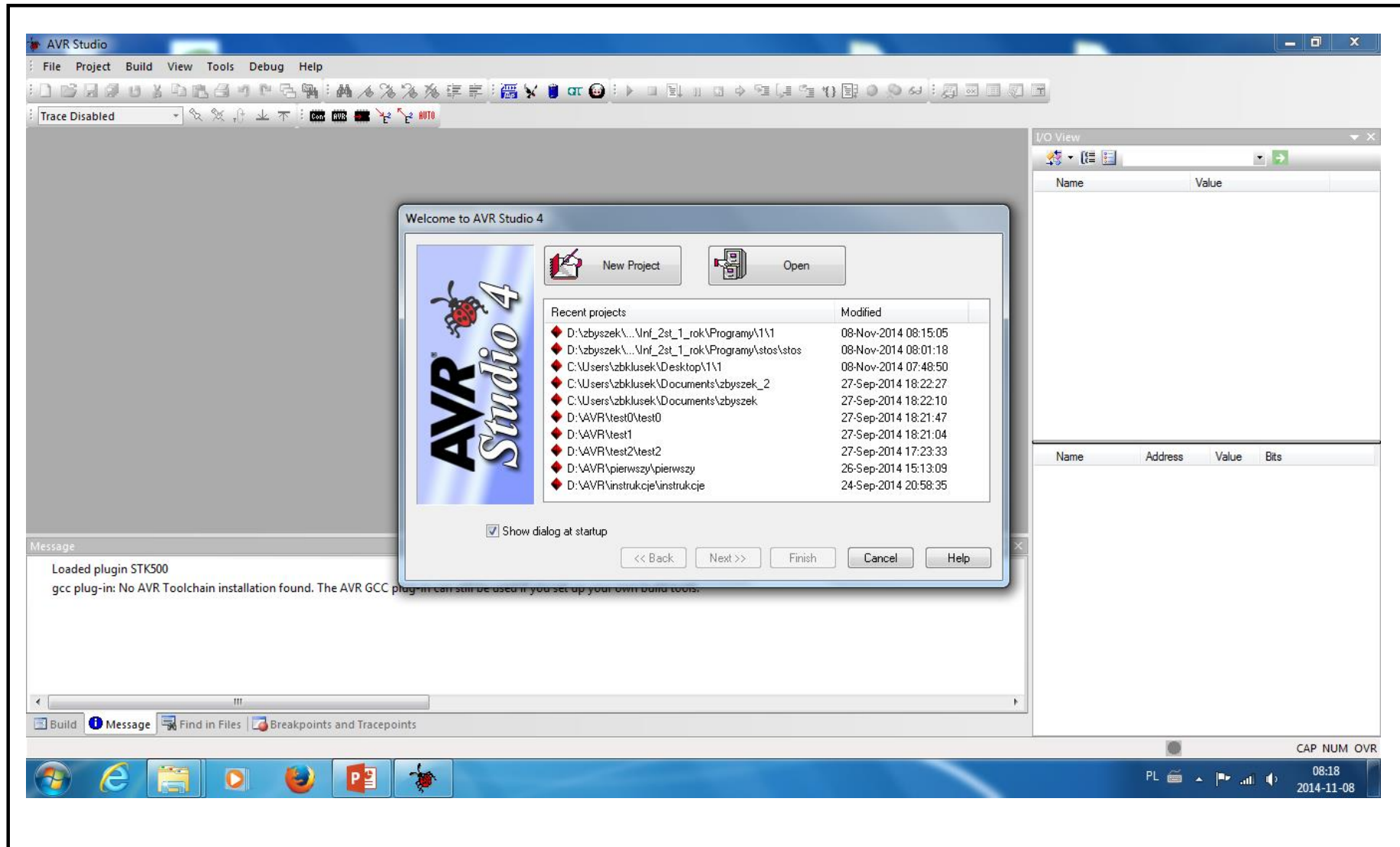
Name	Value
Program Counter	0x000000
Stack Pointer	0x0000
X pointer	0x0000
Y pointer	0x0000
Z pointer	0x0000
Cycle Counter	0
Frequency	1.0000 MHz
Stop Watch	0.00 us
SREG	0x00

Name	Value
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00

Name	Address	Value	Bits
MCUCR	0x35 (0x55)	0x00	00000000
MCUCSR	0x34 (0x54)	0x01	00000001
JTD		0x01	00000000
JTRF		0x01	00000000
WDRF		0x01	00000000
BORF		0x01	00000000
EXTRF		0x01	00000000
PORF		0x01	00000000
OSCCAL	0x31 (0x51)	0x00	00000000
SFIOR	0x30 (0x50)	0x00	00000000

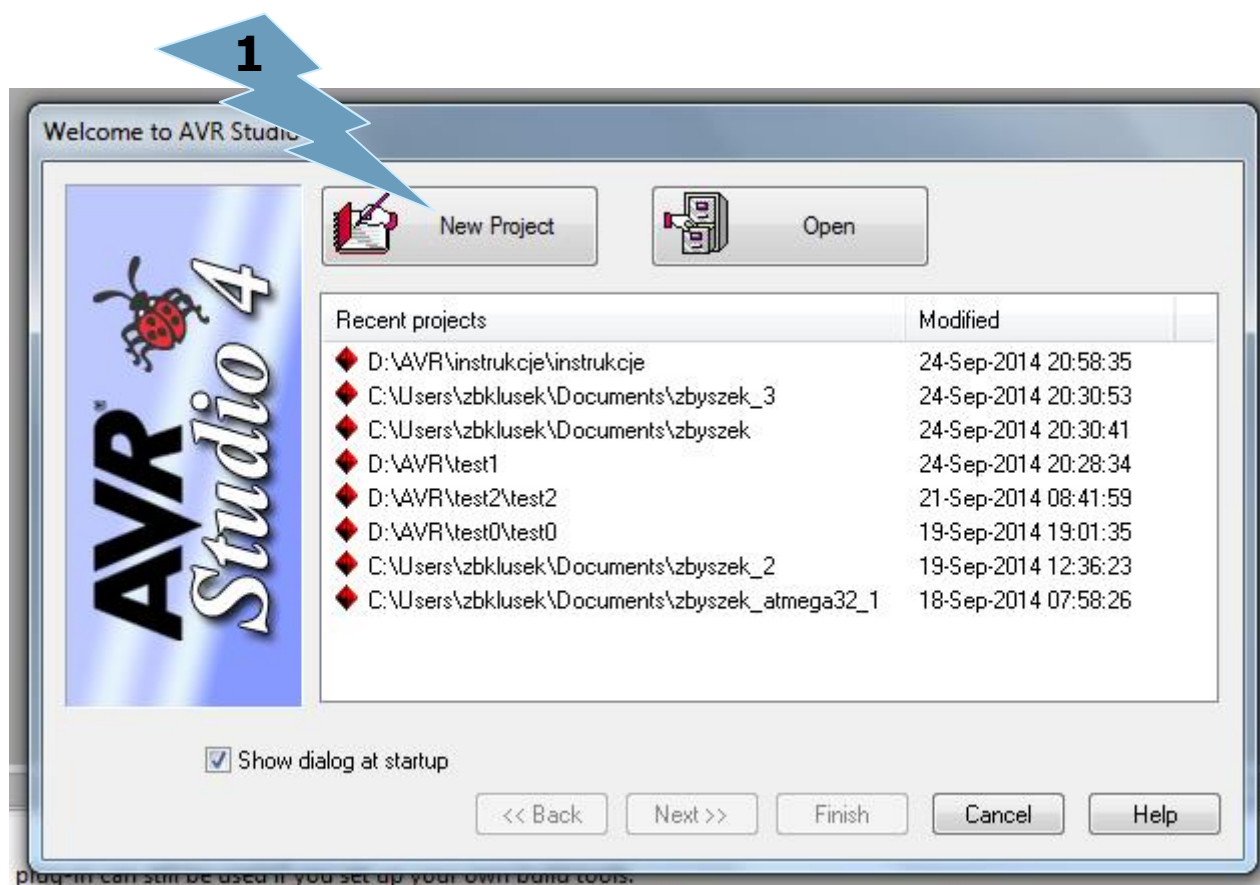


Pierwszy program – środowisko AVR Studio





Pierwszy program – środowisko AVR Studio



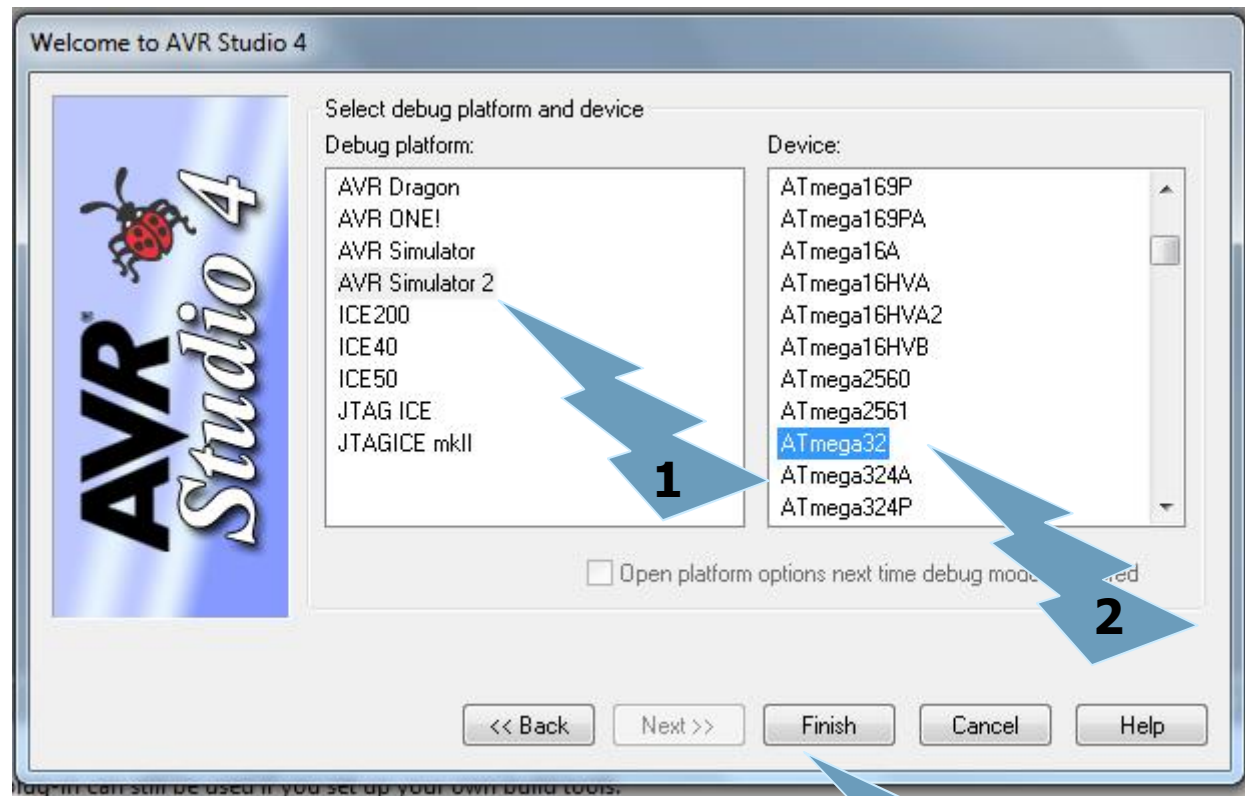


Pierwszy program – środowisko AVR Studio





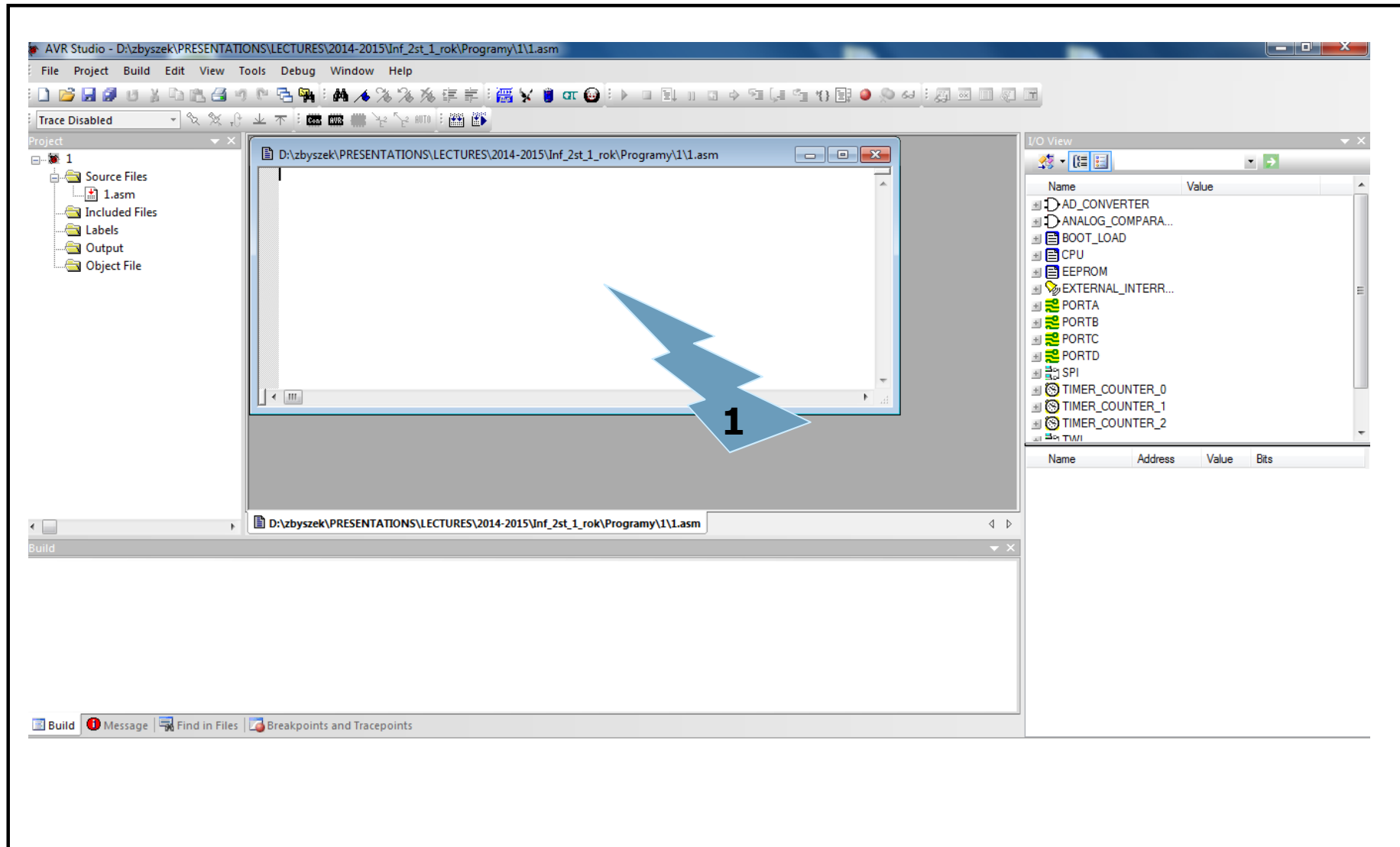
Pierwszy program – środowisko AVR Studio





WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Pierwszy program – środowisko AVR Studio





WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódźki

Pierwszy program – środowisko AVR Studio

The screenshot displays the AVR Studio IDE interface. The main window shows assembly code for a program. The code includes a header file, sets the origin, and initializes stack pointers. It then pushes a value to the stack and exits.

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

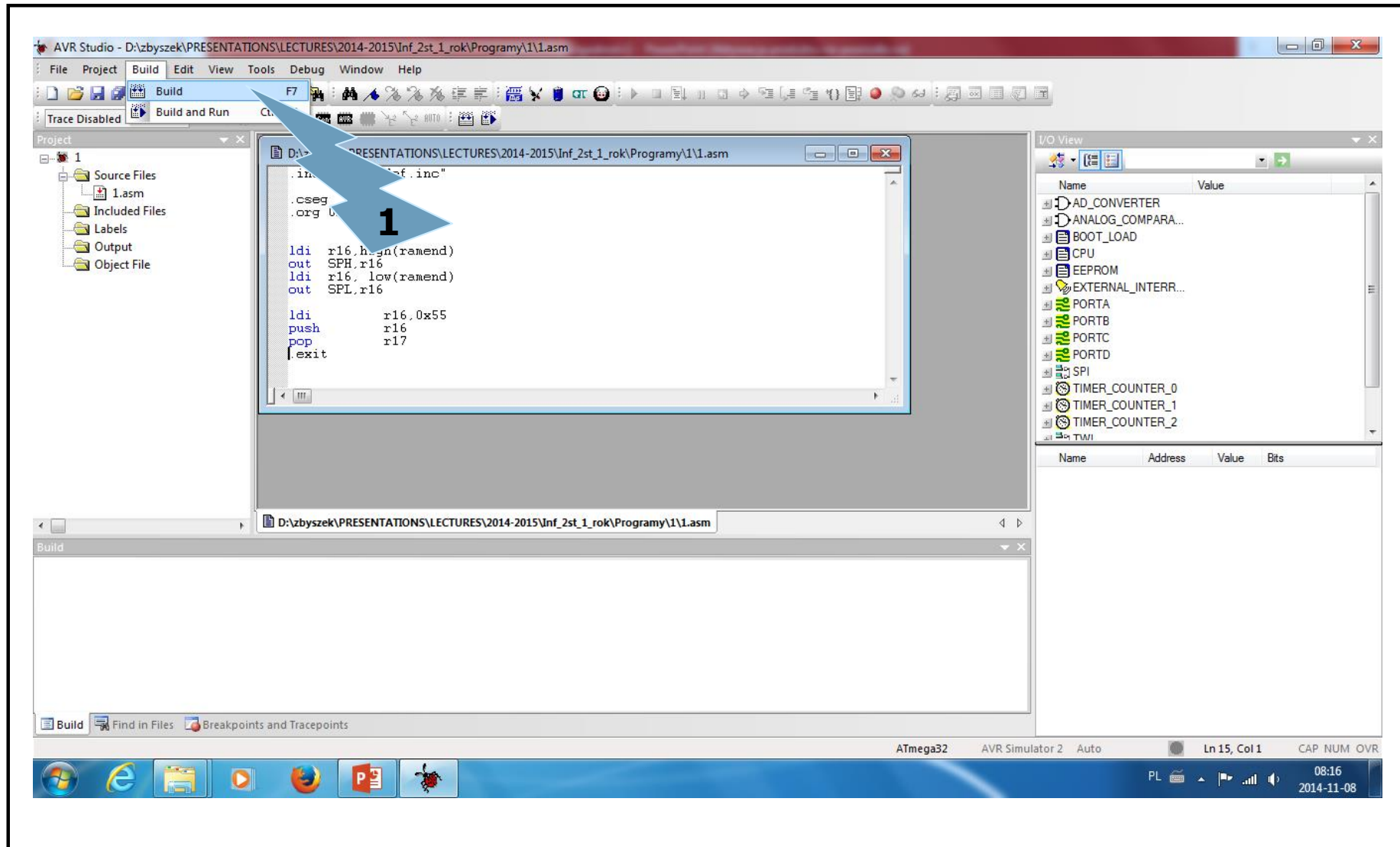
ldi r16, high(ramend)
out SPH, r16
ldi r16, low(ramend)
out SPL, r16

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

The interface also shows a Project Explorer on the left, a Build window at the bottom, and an I/O View on the right listing hardware components like AD_CONVERTER, CPU, and various ports.



Pierwszy program – środowisko AVR Studio





Pierwszy program – środowisko AVR Studio

The screenshot displays the AVR Studio interface. The main window shows assembly code for a program. The build window at the bottom shows the compilation results, indicating that the assembly is complete with no errors or warnings. The I/O View window on the right lists various hardware components.

```
.include "m32def.inc"

.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
```

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

I/O View:

Name	Value
AD_CONVERTER	
ANALOG_COMPARA...	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERR...	
PORTA	
PORTB	
PORTC	
PORTD	
SPI	
TIMER_COUNTER_0	
TIMER_COUNTER_1	
TIMER_COUNTER_2	
TWI	

Build window:

Name	Address	Value	Bits
------	---------	-------	------

W oknie wyświetlają się informacje o wynikach kompilacji

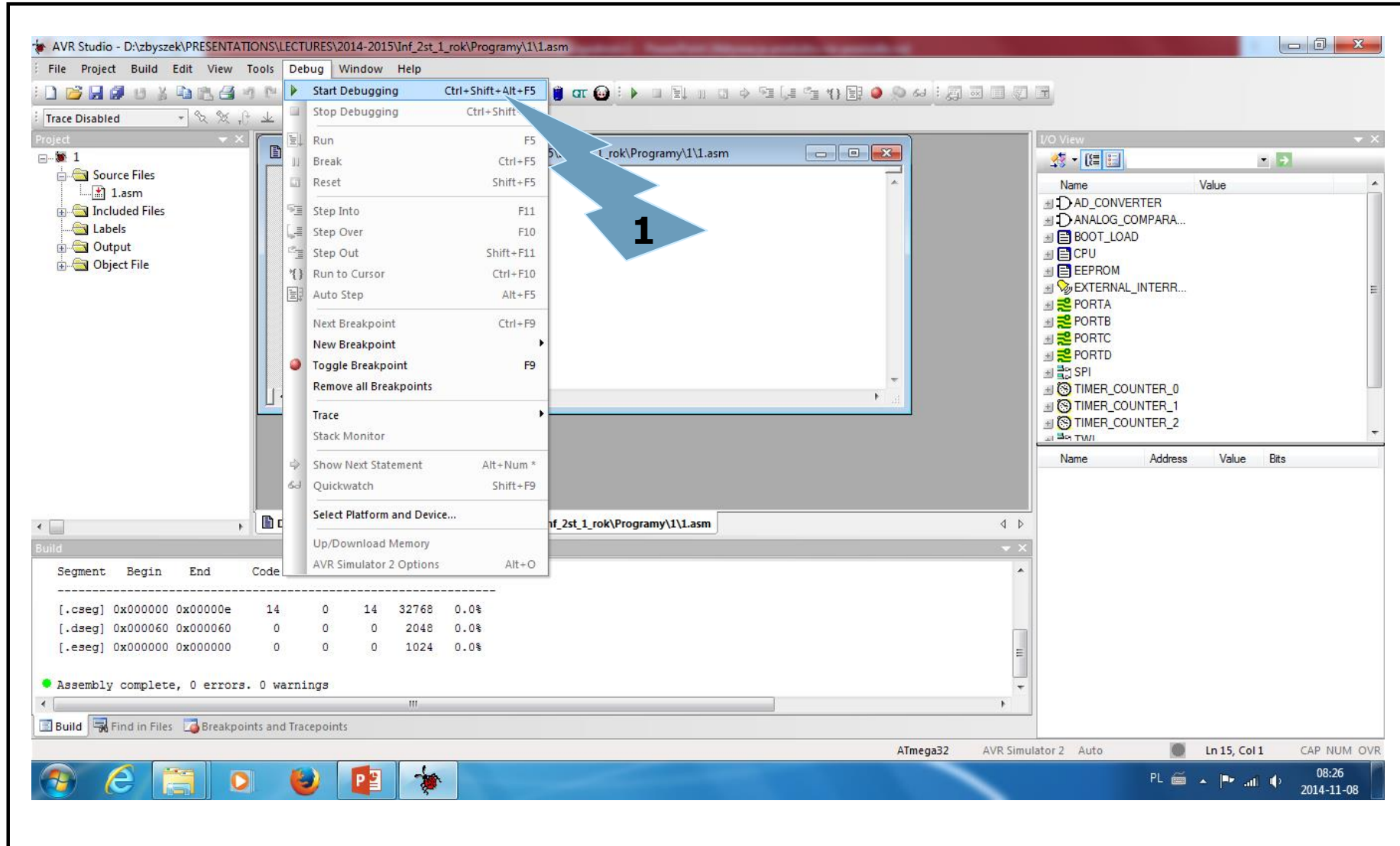


Debugowanie programu

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



Pierwszy program – środowisko AVR Studio





Pierwszy program – środowisko AVR Studio

The screenshot displays the AVR Studio IDE interface. The main window shows assembly code for an AVR microcontroller. The code includes a header file, sets the segment and origin, and performs stack pointer initialization. The processor status window on the left shows various registers and system parameters. The I/O View window on the right lists hardware components like the AD_CONVERTER, ANALOG_COMPARATOR, CPU, EEPROM, and various ports and timers.

Processor

Name	Value
Program Counter	0x000000
Stack Pointer	0x0000
X pointer	0x0000
Y pointer	0x0000
Z pointer	0x0000
Cycle Counter	0
Frequency	1.0000 MHz
Stop Watch	0.00 us
SREG	00000000
Registers	

```
.include "m32def.inc"
.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
```

I/O View

Name	Value
AD_CONVERTER	
ANALOG_COMPARATOR	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
PORTA	
PORTB	
PORTC	
PORTD	
SPI	
TIMER_COUNTER_0	
TIMER_COUNTER_1	
TIMER_COUNTER_2	
TWI	

ATmega32 AVR Simulator 2 Auto Stopped Ln 7, Col 1 CAP NUM OVR
08:31 2014-11-08



Pierwszy program – środowisko AVR Studio

The screenshot shows the AVR Studio IDE with the following components:

- Processor Window:** Shows the state of the AVR processor. Key values include:

Name	Value
Program Counter	0x000006
Stack Pointer	0x085E
X pointer	0x0000
Y pointer	0x0000
Z pointer	0x0000
Cycle Counter	7
Frequency	1.0000 MHz
Stop Watch	7.00 us
SREG	0x00
- Code Editor:** Contains the following assembly code:

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

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

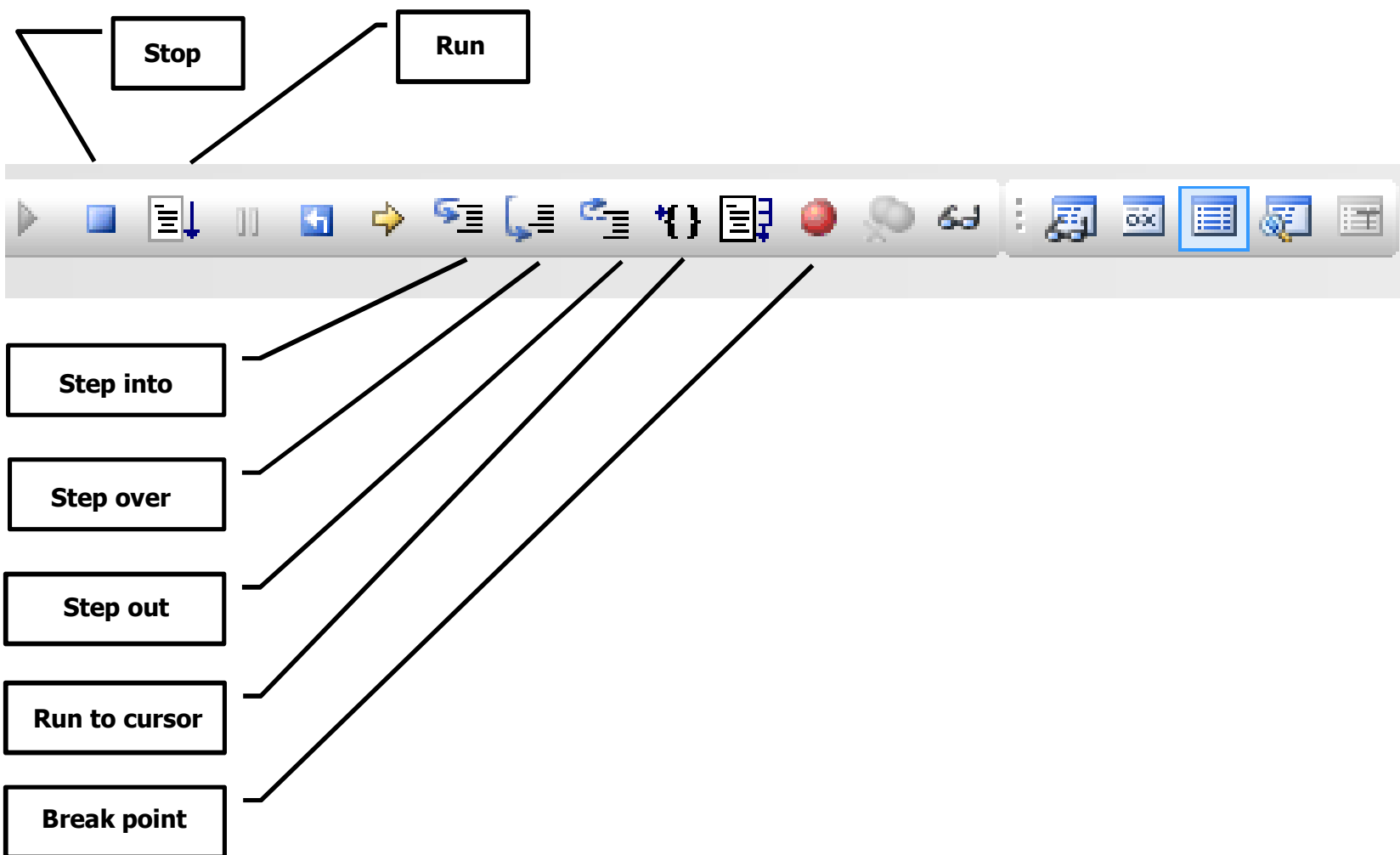
ldi r16, 0x55
push r16
pop r17
.exit
```
- I/O View:** Shows the state of various hardware modules. The SP register is highlighted with a value of 0x085E.

Name	Address	Value	Bits
MCUCR	0x35 (0x55)	0x00	00000000
MCUCSR	0x34 (0x54)	0x01	00000000
OSCCAL	0x31 (0x51)	0x00	00000000
SFIOR	0x30 (0x50)	0x00	00000000
SP	0x3D (0x5D)	0x085E	00000000
SREG	0x3F (0x5F)	0x00	00000000
- Memory Window:** Shows the program memory layout. The address 0x0808 is highlighted, showing the value 0x55.

Address	Value
0x0800	00 00 00 00
0x0806	00 00 00 00
0x0808	00 00 00 00
0x080C	00 00 00 00
0x0812	00 00 00 00
0x0818	00 00 00 00
0x081E	00 00



Pierwszy program – środowisko AVR Studio



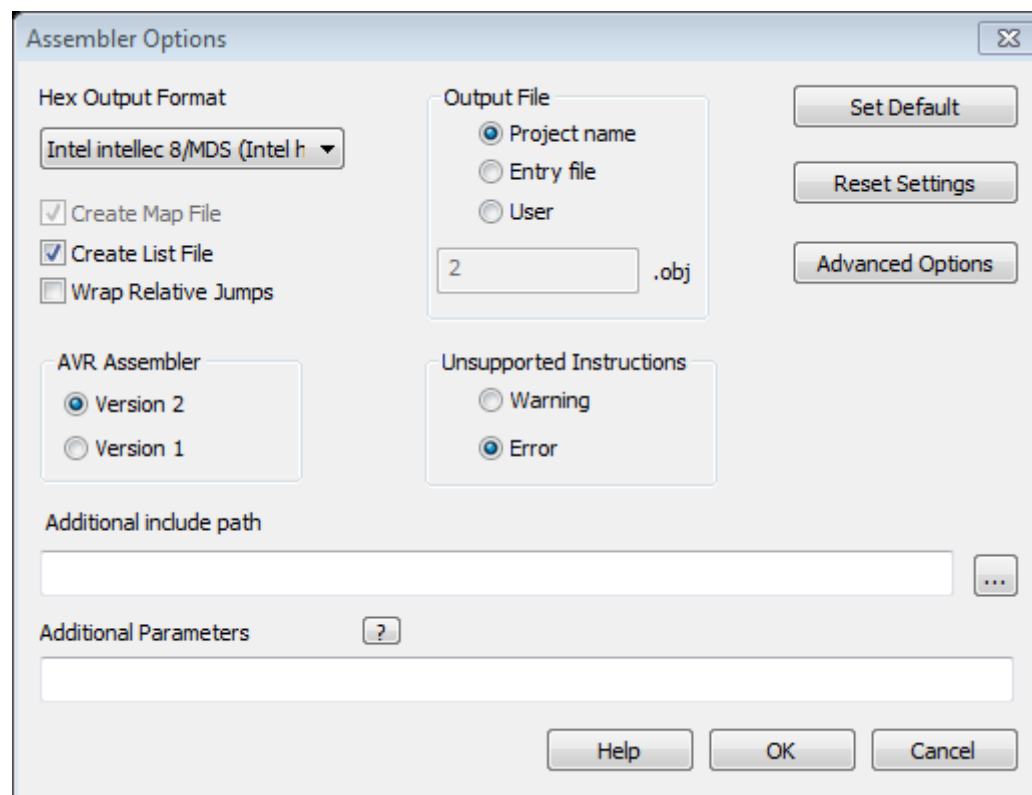


Listing programu

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



Struktura pliku Ist





Struktura pliku Ist

```
; 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 OCFR = 0x30
.equ SFOR = 0x2f
.equ TCCR1A = 0x2e
.equ TCCR1B = 0x2d
.equ TCNT1L = 0x2c
.equ TCNT1H = 0x2b
.equ OCR1AL = 0x2a
.equ OCR1AH = 0x29
.equ OCR1BL = 0x28
.equ OCR1BH = 0x27
.equ ICR1L = 0x26
.equ ICR1H = 0x25
.equ TCCR2 = 0x24
.equ TCNT2 = 0x23
.equ OCR2 = 0x22
.equ ASSR = 0x21
.equ WDTCSR = 0x20
.equ UBRRH = 0x1e
.equ UCSRC = 0x1d
.equ EEARL = 0x1c
.equ EEADR = 0x1b
.equ EECR = 0x1a
.equ PORTA = 0x19
.equ DDRA = 0x18
.equ PINA = 0x17
.equ PORTB = 0x16
.equ DDRB = 0x15
.equ PINB = 0x14
.equ PORTC = 0x13
.equ DDRC = 0x12
.equ PINC = 0x11
.equ PORTD = 0x10
.equ DDRD = 0x0f
.equ PIND = 0x0e
.equ SPDR = 0x0d
.equ SPCR = 0x0c
.equ UDR = 0x0b
.equ UCSRA = 0x0a
.equ UCSRB = 0x09
.equ UBRRL = 0x08
.equ ACSR = 0x07
.equ ADMUX = 0x06
.equ ADCSRA = 0x05
.equ ADCH = 0x04
.equ ADCL = 0x03
.equ TWDR = 0x02
.equ TWAR = 0x01
.equ TWSR = 0x00
.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 EWE = 1 ; EEPROM Write Enable
.equ EEMWE = 2 ; EEPROM Master Write Enable
.equ EERIE = 3 ; EEPROM Ready Interrupt Enable

; ***** WATCHDOG *****
; WDTCSR - 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 = WDDE ; For compatibility
```



Struktura pliku Ist

```
; **** 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  OV2addr              = 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  OV1addr              = 0x0012      ; Timer/Counter1 Overflow
.equ  OC0addr              = 0x0014      ; Timer/Counter0 Compare Match
.equ  OV0addr              = 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 Ist

```
#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 Ist

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	sbcj : 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



Dyrektywa list/nolist

1. Struktura pliku list
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 .adw : 0 .and : 0
.andi : 0 .asr : 0 .bclr : 0 .bld : 0 .brbc : 0 .brbs : 0
.brcc : 0 .brcs : 0 .break : 0 .brq : 0 .brge : 0 .brhc : 0
.brhs : 0 .brid : 0 .brle : 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
.fmulsc : 0 .jcall : 0 .jmp : 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

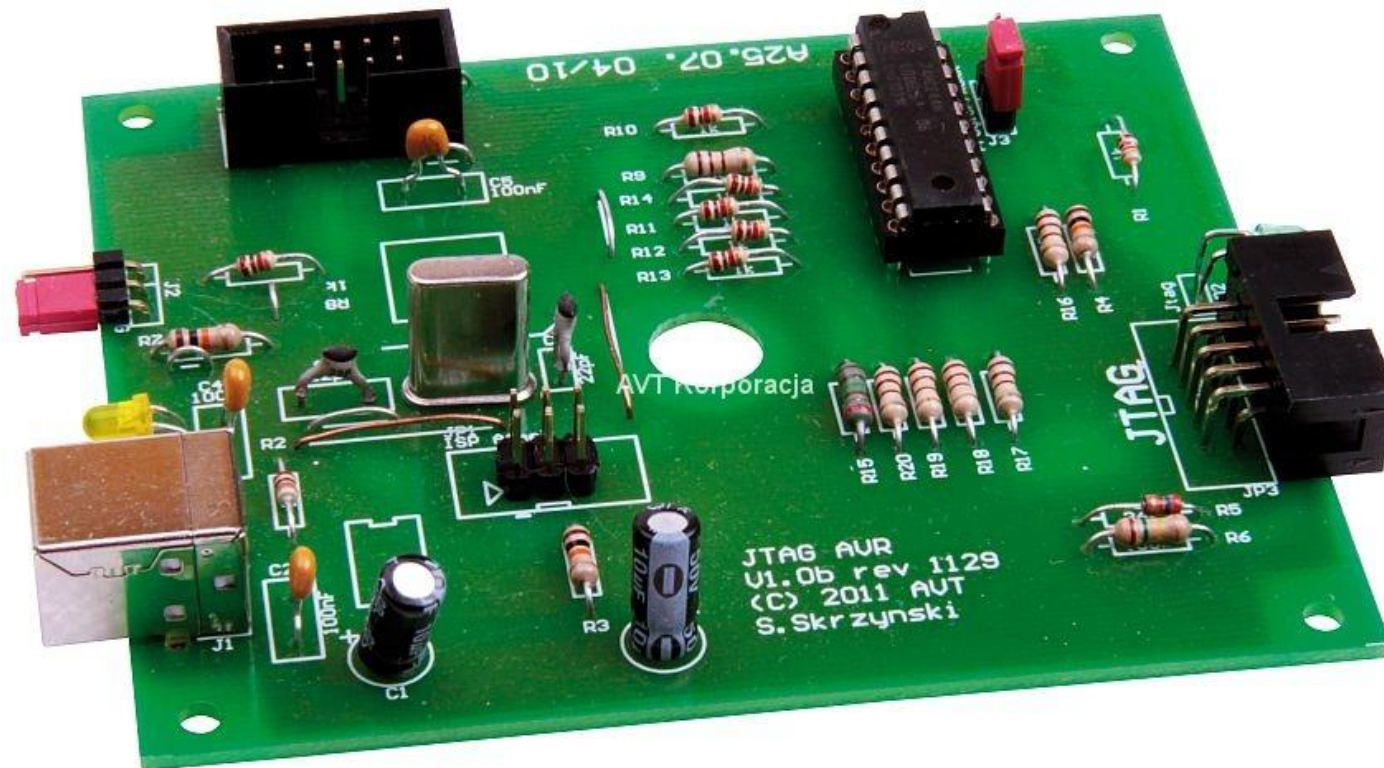
ICE
In Circuit Emulation

OCD
On-Chip Debug

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

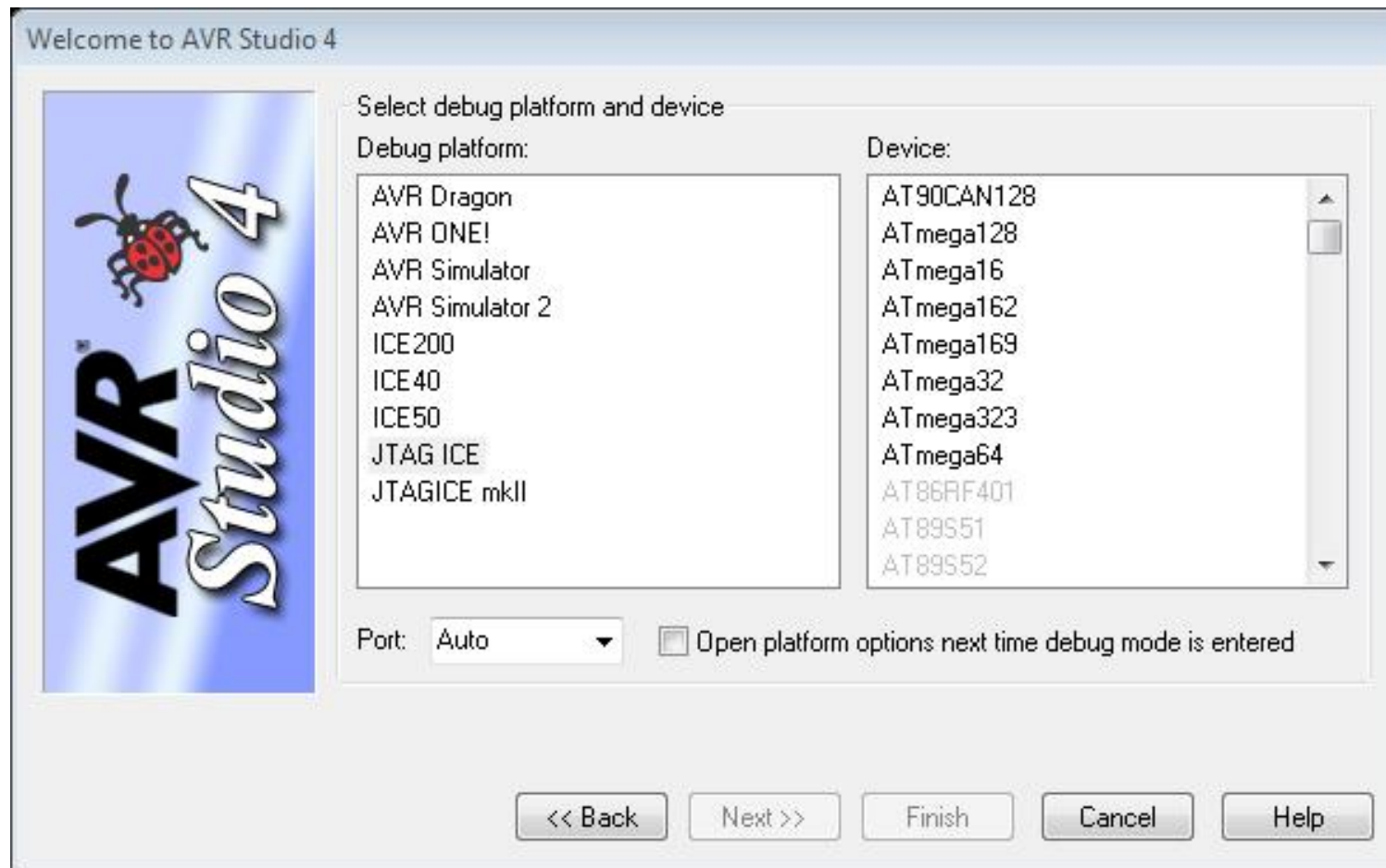


AVT5322





Emulatory





WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Emulacja sprzętowa, Debager działa tak jak symulator AVR Simulator2

The screenshot displays the AVR Studio 2.0 development environment. The main window shows the assembly code for a program:

```
.include "m32def.inc"
.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
```

The Processor window on the left shows the state of the AVR processor:

Name	Value
Program Counter	0x000006
Stack Pointer	0x085E
X pointer	0x0000
Y pointer	0x0000
Z pointer	0x0000
Cycle Counter	7
Frequency	1.0000 MHz
Stop Watch	7.00 us
SREG	00000000

The Registers window shows the state of the registers:

Register	Value
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x55
R17	0x00
R18	0x00
R19	0x00
R20	0x00

The Memory window shows the memory contents:

Address	Value
0x0000	08 E0 0E BF 0F E5 0D BF 05 E5 0F 93 1F 91 FF FF
0x0001	0D BF 05 E5 0F 93 FF FF FF FF FF FF FF FF FF
0x0002	1F 91 FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0003	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0004	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0005	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0006	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0007	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0008	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0009	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x000A	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x000B	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x000C	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x000D	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x000E	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x000F	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0010	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0011	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0012	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0013	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0014	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0015	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0016	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0017	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0018	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0019	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x001A	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x001B	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x001C	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x001D	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x001E	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x001F	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

The I/O View window shows the state of the I/O devices:

Name	Value
AD_CONVERTER	
ANALOG_COMPARA...	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERR...	
PORTA	
PORTB	
PORTC	
PORTD	
SPI	
TIMER_COUNTER_0	
TIMER_COUNTER_1	
TIMER_COUNTER_2	
TMU	

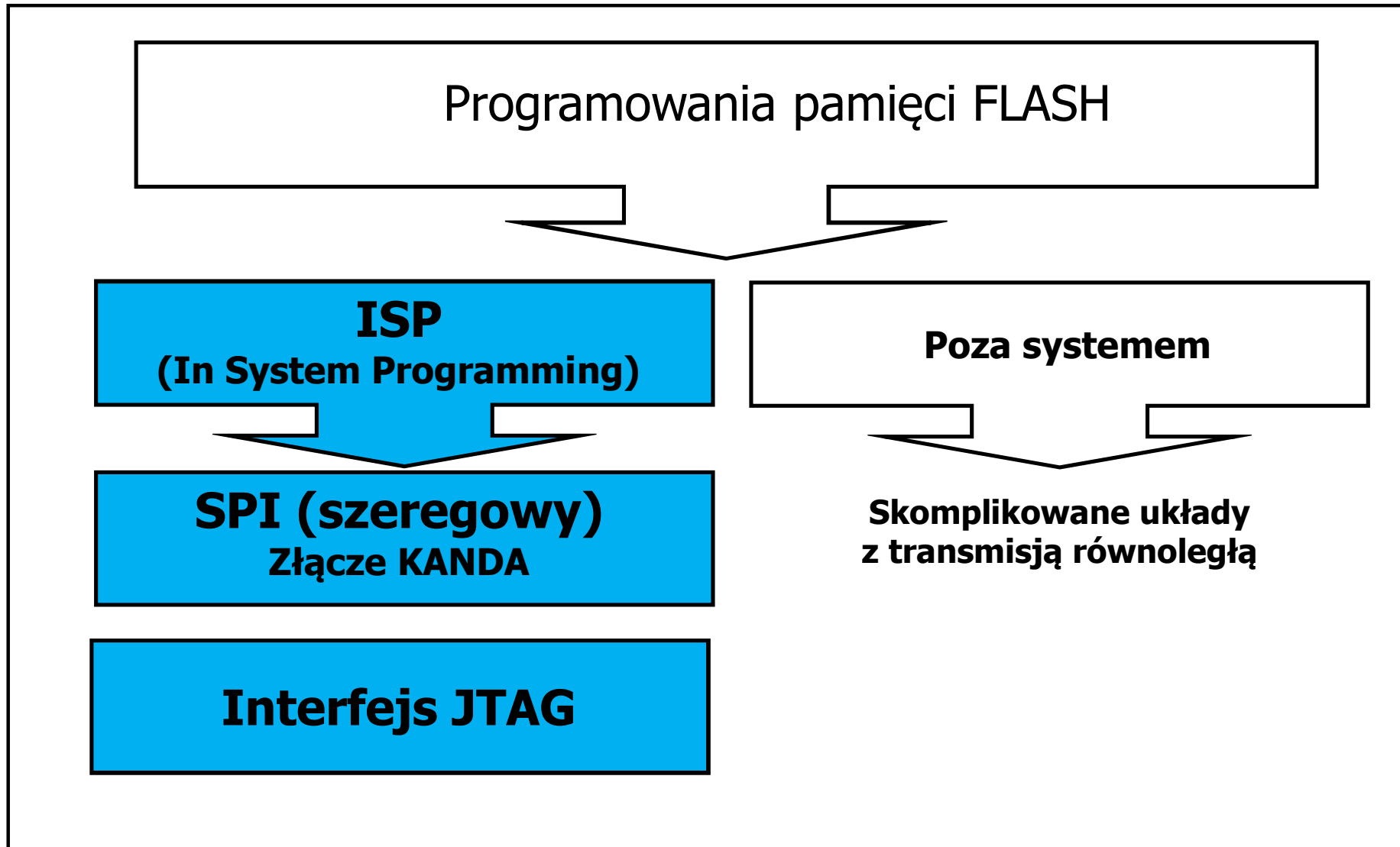
The Memory window shows the memory contents:

Address	Value
0x0000	00 00 00 00
0x0001	00 00 00 00
0x0002	00 00 00 00
0x0003	00 00 00 00
0x0004	00 00 00 00
0x0005	00 00 00 00
0x0006	00 00 00 00
0x0007	00 00 00 00
0x0008	00 00 00 00
0x0009	00 00 00 00
0x000A	00 00 00 00
0x000B	00 00 00 00
0x000C	00 00 00 00
0x000D	00 00 00 00
0x000E	00 00 00 00
0x000F	00 00 00 00
0x0010	00 00 00 00
0x0011	00 00 00 00
0x0012	00 00 00 00
0x0013	00 00 00 00
0x0014	00 00 00 00
0x0015	00 00 00 00
0x0016	00 00 00 00
0x0017	00 00 00 00
0x0018	00 00 00 00
0x0019	00 00 00 00
0x001A	00 00 00 00
0x001B	00 00 00 00
0x001C	00 00 00 00
0x001D	00 00 00 00
0x001E	00 00 00 00
0x001F	00 00 00 00

The status bar at the bottom shows the current state: ATmega32, AVR Simulator 2, Auto, Stopped, Ln 14, Col 1, CAP NUM OVR, 08:38, 2014-11-08.



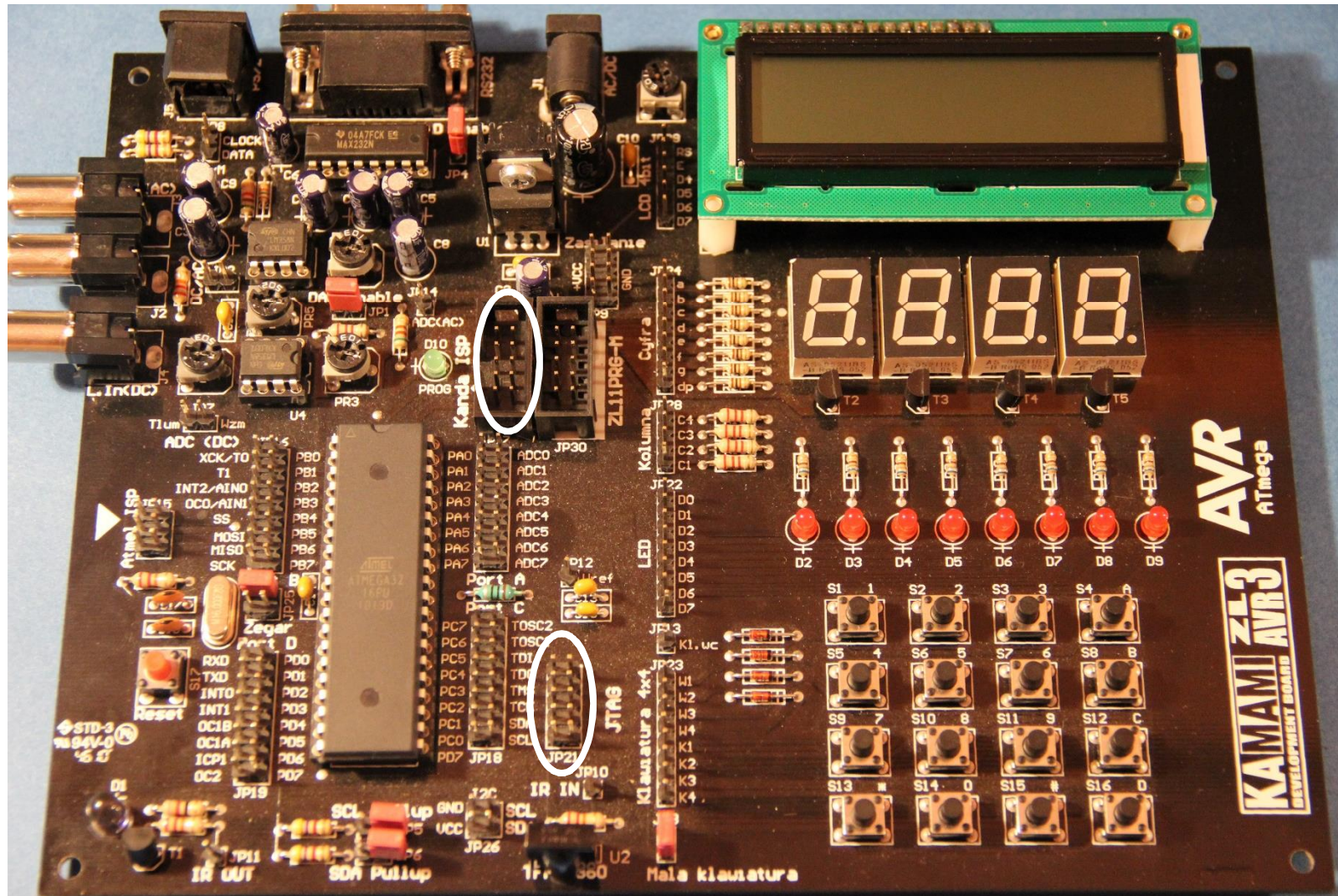
Interfejsy programowania i uruchomieniowe





WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

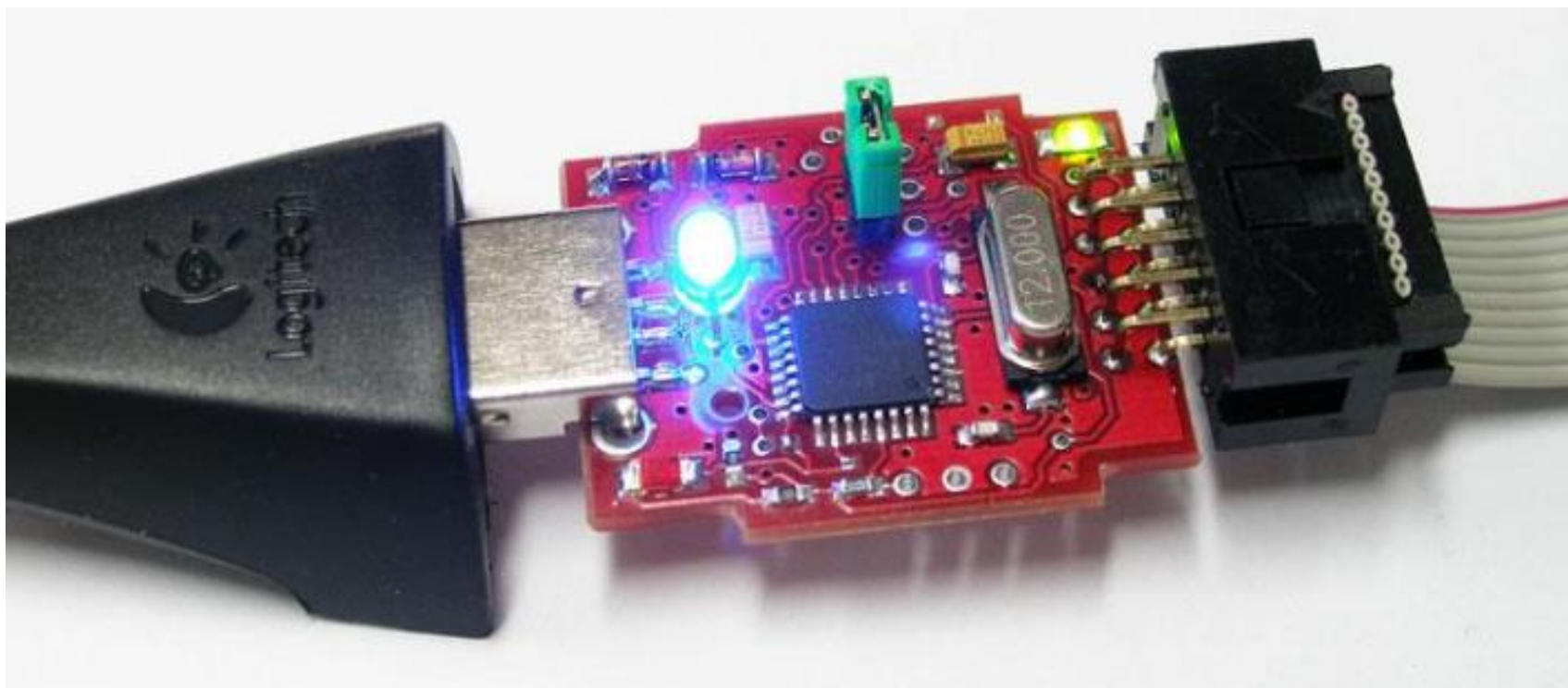
Zestaw ZL3AVR



Wydział Fizyki i Informatyki Stosowanej UŁ



SPI (szeregowy) Złącze KANDA





WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

Programator USBASP

mkAVR Calculator 1.1.0 build 67 : Witold Kozłowski

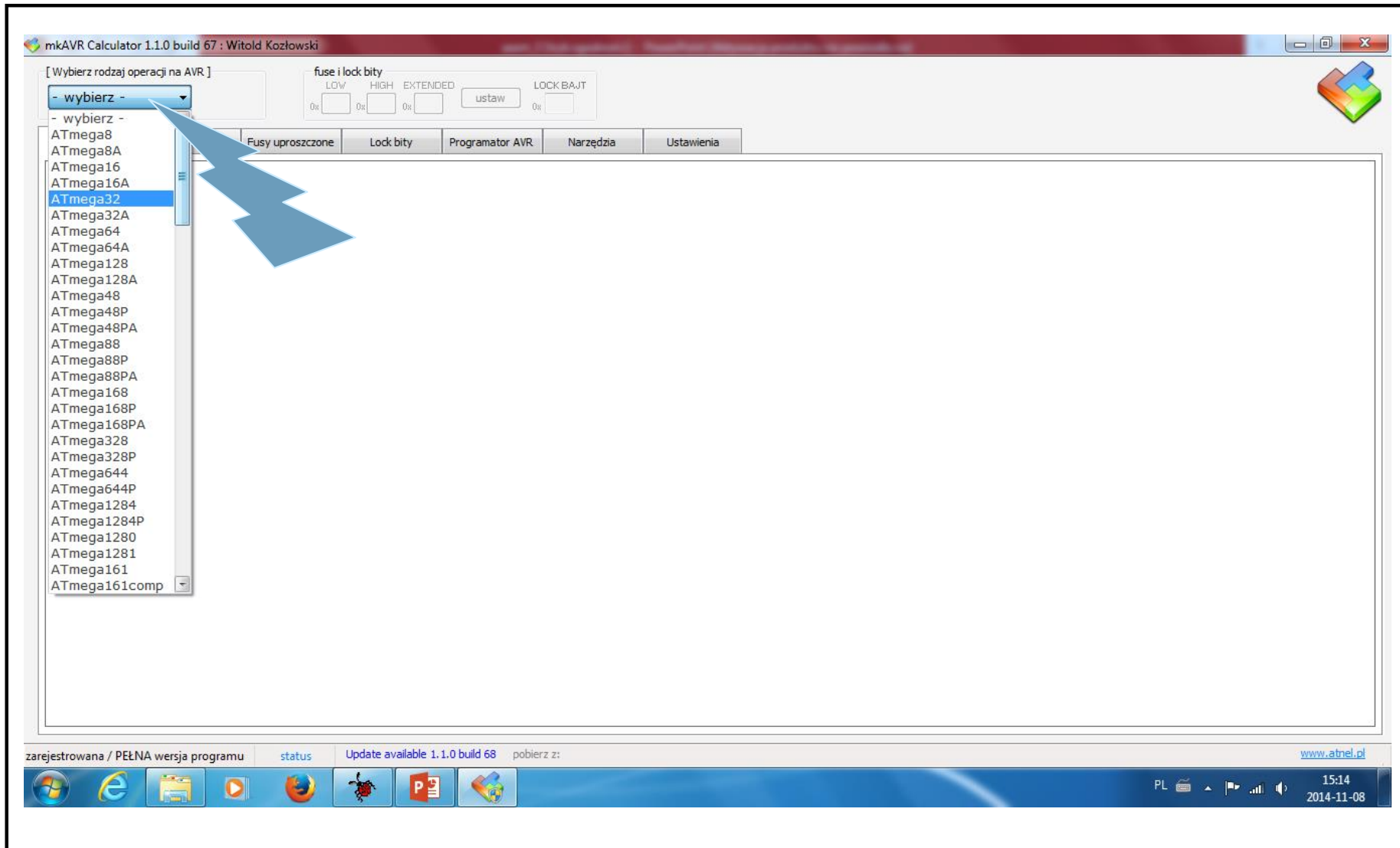
[Wybierz rodzaj operacji na AVR]
- wybierz -

fuse i lock bity
LOW HIGH EXTENDED LOCK BAJT
0x 0x 0x 0x
ustaw

Fusy właściwości | Fusy manualnie | Fusy uproszczone | Lock bity | Programator AVR | Narzędzia | Ustawienia

zarejestrowana / PEŁNA wersja programu status Update available 1.1.0 build 68 pobierz z: www.atmel.pl

15:12
2014-11-08





Programator USBASP

mkAVR Calculator 1.1.0 build 67 : Witold Kozłowski [ATmega32]

[Wybierz rodzaj operacji na AVR]
Wybierz rodzaj operacji na AVR: ATmega32
ustawienia fabryczne

FLASH: 32 768
EEPROM: 1 024
INT RAM: 2 048
EXT RAM: 0

fuse i lock bity
LOW HIGH EXTENDED
0x E1 0x 99 0x
ustaw
LOCK BAJT
0x 3F

Fusy właściwości | Fusy manualnie | Fusy uproszczone | Lock bity | Programator AVR | Narzędzia | Ustawienia

Brown-out detection level at VCC=2.7 V; [BODLEVEL=1]

Brown-out detection enabled; [BODEN=0]

Int. RC Osc. 1 MHz; Start-up time: 6 CK + 64 ms; [CKSEL=0001 SUT=10]; default value

On-Chip Debug Enabled; [OCDEN=0]

JTAG Interface Enabled; [JTAGEN=0]

Serial program downloading (SPI) enabled; [SPIEN=0]

Preserve EEPROM memory through the Chip Erase cycle; [EESAVE=0]

Boot Flash section size=2048 words Boot start address=\$3800; [BOOTSZ=00]; default value

Boot Reset vector Enabled (default address=\$0000); [BOOTRST=0]

CKOPT fuse (operation dependent of CKSEL fuses); [CKOPT=0]

zarejestrowana / PEŁNA wersja programu | status | Update available 1.1.0 build 68 | pobierz z: | www.atmel.pl

15:15
2014-11-08



mkAVR Calculator 1.1.0 build 67 : Witold Kozłowski [ATmega32]

[Wybierz rodzaj operacji na AVR]
ATmega32

FLASH: 32 768 fuse i lock bity
EEPROM: 1 024 LOW HIGH EXTENDED
INT RAM: 2 048 0x: E1 0x: 99 0x:
EXT RAM: 0 LOCK BAJT 0x: 3F

Fusy właściwości | **Fusy manualnie** | Fusy uproszczone | Lock bity | Programator AVR | Narzędzia | Ustawienia

Bit	Byte LOW	Byte HIGH	Byte EXTENDED
7	<input type="checkbox"/> BODLEVEL Brown out detector trigger level	<input type="checkbox"/> OCDEN Enable OCD, On-Chip Debug Enabled	
6	<input type="checkbox"/> BODEN Brown out detector enable	<input checked="" type="checkbox"/> JTAGEN Enable JTAG, JTAG Interface Enabled	
5	<input type="checkbox"/> SUT1 Select start-up time	<input checked="" type="checkbox"/> SPIEN Enable Serial programming and Data Downloading, Serial program	
4	<input checked="" type="checkbox"/> SUT0 Select start-up time	<input type="checkbox"/> CKOPT Oscillator options	
3	<input checked="" type="checkbox"/> CKSEL3 Select Clock Source	<input type="checkbox"/> EESAVE EEPROM memory is preserved through chip erase, Preserve	
2	<input checked="" type="checkbox"/> CKSEL2 Select Clock Source	<input checked="" type="checkbox"/> BOOTSZ1 Select Boot Size	
1	<input checked="" type="checkbox"/> CKSEL1 Select Clock Source	<input checked="" type="checkbox"/> BOOTSZ0 Select Boot Size	
0	<input type="checkbox"/> CKSEL0 Select Clock Source	<input type="checkbox"/> BOOTRST Select Reset Vector, Boot Reset vector Enabled	

status Update available 1.1.0 build 68 pobierz z: www.atmel.pl

PL 15:17 2014-11-08





Programator USBASP

mkAVR Calculator 1.1.0 build 67 : Witold Kozłowski [ATmega32]

[Wybierz rodzaj operacji na AVR]
ATmega32 ustawienia fabryczne

FLASH: 32 768 fuse i lock bity
EEPROM: 1 024 LOW HIGH EXTENDED LOCK BAJT
INT RAM: 2 048 0x: E1 0x: 99 0x: ustaw 0x: 3F
EXT RAM: 0

Fusy właściwości Fusy manualnie Fusy uproszczone Lock bity Programator AVR Narzędzia Ustawienia

Mode 1: No memory lock features enabled
Application Protection Mode 1: No lock on SPM and LPM in Application Section
Boot Loader Protection Mode 1: No lock on SPM and LPM in Boot Loader Section

status Update available 1.1.0 build 68 pobierz z: www.atmel.pl

PL 15:18
2014-11-08



Programator USBASP

The screenshot shows the mkAVR Calculator 1.1.0 build 67 interface. The window title is "mkAVR Calculator 1.1.0 build 67 : Witold Kozłowski [ATmega32]". The main area is divided into several sections:

- [Wybierz rodzaj operacji na AVR]**: Shows "ATmega32" selected in a dropdown. Memory statistics: FLASH: 32 768, EEPROM: 1 024, INT RAM: 2 048, EXT RAM: 0. Fuse and lock bits are set to E1, 99, and 3F.
- [Ustawienia AVRDUDE]**: Shows "usbasp" as the programmer and "usb" as the port. The SCK speed is set to "4.0 - 8.0 -> 187.5 kHz". A "Sprawdź podłączony AVR" button is highlighted with a blue arrow.
- [Operacja AVR]**: Shows "ODCZYT" (Read) selected. Other options include "ZAPIS" (Write) and "WERYFIKACJA" (Verify). There are checkboxes for "FLASH", "Fuse bity", and "Lock bity".
- [Linia poleceń AVRDUDE]**: Shows the command line: `avrdude -p atmega32 -c usbasp -P usb -B 8`. A green "ODCZYT z AVR" button is visible above the command line.

The Windows taskbar at the bottom shows the system tray with the date "2014-11-08" and time "15:19".



Programator USBASP

The screenshot shows the mkAVR Calculator 1.1.0 build 67 interface. The window title is "mkAVR Calculator 1.1.0 build 67 : Witold Kozłowski [ATmega32]". The main area is titled "[Ustawienia AVRDUDE]".

At the top, there are memory and fuse settings: FLASH: 32 768, EEPROM: 1 024, INT RAM: 2 048, EXT RAM: 0. Fuse settings include LOW (E4), HIGH (99), and LOCK BAJT (3F).

The "Szybki wybór programatora" section has "USBASP" selected. The "Programator" dropdown is set to "usbasp", and the "Port" is "usb". The "Słów SCK" dropdown is set to "brak". A yellow button "Sprawdź podłączony AVR" is visible. To its right, a box displays "Sygnatura AVR: 1E9502" and "Nazwa AVR: ATmega32", with "ATmega32" circled in black.

The "[Operacja AVR]" section has "ODCZYT" selected. The "rodzaj pamięci" section has "FLASH" selected. There are buttons for "Otwórz profil" and "Zapisz profil".

The "[Linia poleceń AVRDUDE]" section shows the command: `avrdude -p atmega32 -c usbasp -P usb`. A green "ODCZYT z AVR" button is present. At the bottom right of this section are "WYKONAJ" and "reset" buttons.

The bottom status bar shows "zarejestrowana / PE&NA wersja programu", "status", "Update available 1.1.0 build 68", and "pobierz z: www.atmel.pl". The Windows taskbar at the very bottom shows the time as 15:20 on 2014-11-08.



Programator USBASP

mkAVR Calculator 1.1.0 build 67 : Witold Kozłowski [ATmega32]

[Wybierz rodzaj operacji na AVR]
ATmega32

FLASH: 32 768
EEPROM: 1 024
INT RAM: 2 048
EXT RAM: 0

fuse i lock bity
LOW HIGH EXTENDED
0x E4 0x 99 0x
LOCK BAJT
0x 3F

Fusy właściwości | Fusy manualnie | Fusy uproszczone | Lock bity | Programator AVR | Narzędzia | Ustawienia

[Ustawienia AVRDUDE]
Szybki wybór programatora: USBASP (wybrany), ATB-FT232R, stk500v2, własny wybór
Programator: usbasp | Port: usb | Slow SCK: brak | Auto SCK speed:

[Operacja AVR]
ODCZYT | ZAPIS (wybrany) | WERYFIKACJA
rodzaj pamięci: FLASH (wybrany), EEPROM, Fuse bity, Lock bity

Opcje dodatkowe: -D Wylącz auto kasowanie (0%), -e wykonaj kasowanie AVR, -n nie zapisuj do AVR

[Linia poleceń AVRDUDE]
additional options (speed) | ZAPIS do AVR

```
avrdude -p atmega32 -c usbasp -P usb -V -U flash:w:"D:\zbyszek\PRESENTATIONS\LECTURES\2014-2015\Inf_2st_1_rok\Programy\4\4.hex":i
```

WYKONAJ | reset

zarejestrowana / PEŁNA wersja programu | status | Update available 1.1.0 build 68 | pobierz z: | www.atmel.pl

15:21
2014-11-08