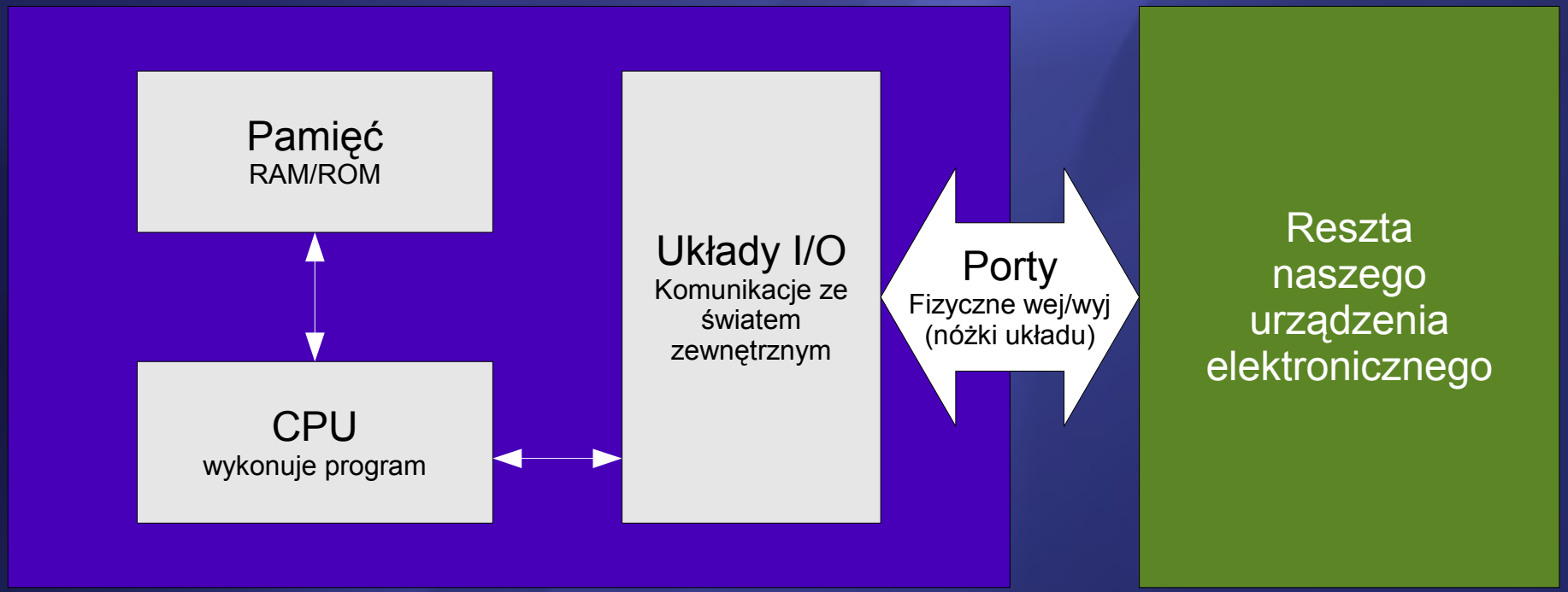
Programowanie mikrokontrolerów AVR

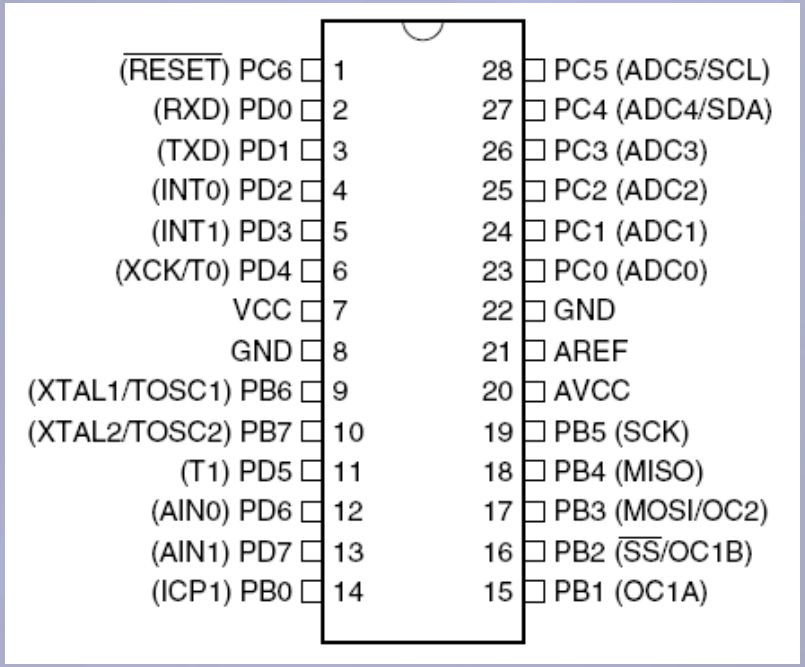
Czym jest mikrokontroler?

Mikrokontroler jest „małym komputerem” podłączanym do układów elektronicznych.



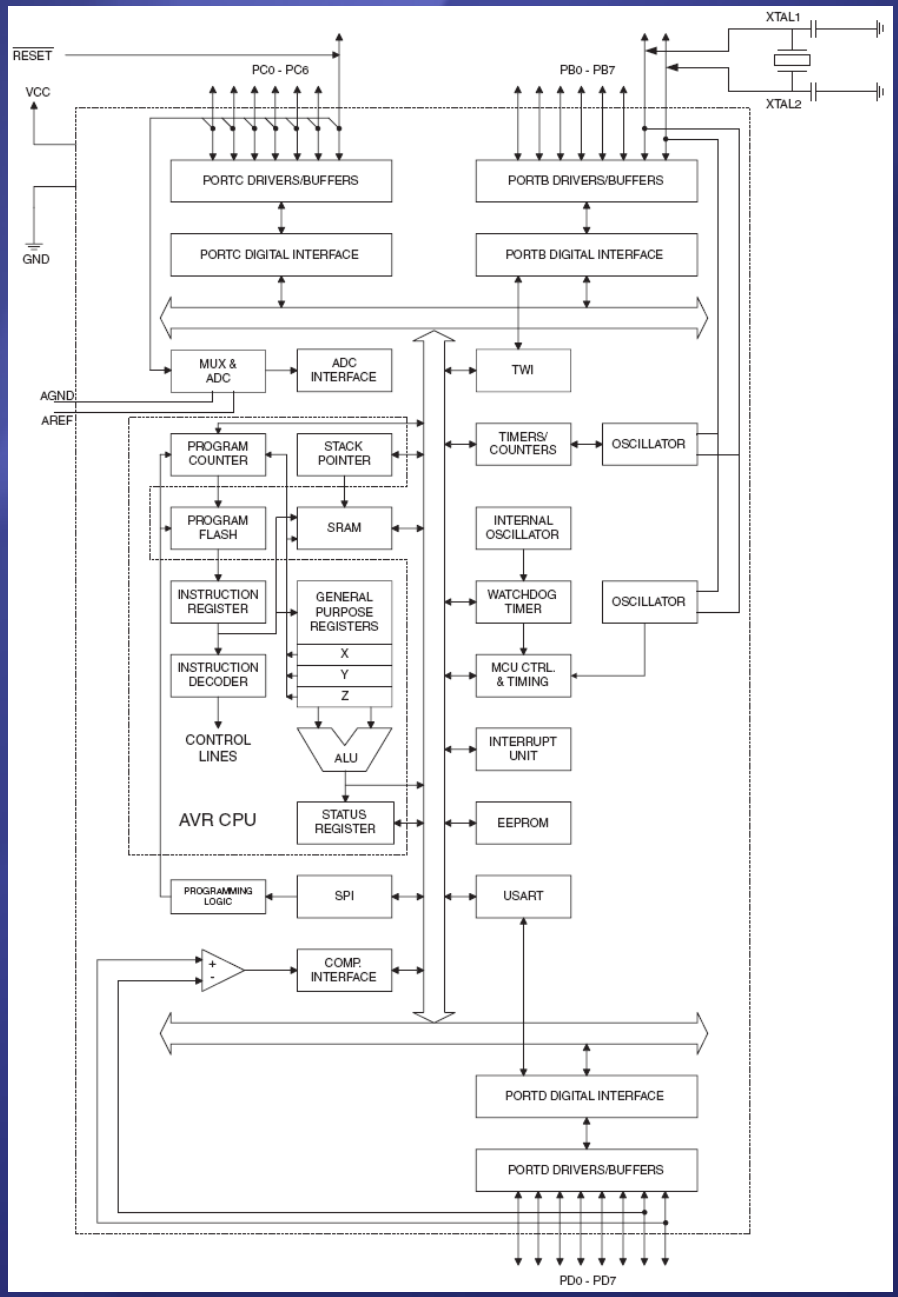
Opis mikrokontrolera AVR (ATmega8)

- ATmega8 posiada 22 porty (nóżki) ogólnego użytku
- Każdy może być sterowany bezpośrednio – podawanie stanu niskiego, wysokiego lub odczyt stanu.
- Każdy dodatkowo jest wykorzystywany w urządzeniach wejścia-wyjścia.



Opis mikrokontrolera AVR (ATmega8)

- **CPU**: 8-bitowy RISC
- **FLASH**: 8K, programowalna, nieulotna pamięć programu
- **SRAM**: 1K, zmienne, stos, itp.
- EEPROM: 0.5K, nieulotna
- Timer/licznik/PWM x3
- 10-bit ADC x6, Analog Comparator
- SPI, USART, TWI
- 23 programowalne cyfrowe linie IO
- 19 przerwań sprzętowych (w tym 2 zewnętrzne)
- ISP: proste programowanie
- Zegar: max 16 MHz - prawie 16 mln instrukcji/s
- Wewnętrzny zegar 1-8 MHz



Programowanie – co będzie potrzebne?

- Kompilator,
- Biblioteki dla konkretnego układu,
- Datasheet układu,
- Programator (sprzęt i software),
- Edytor plików źródłowych,

Może się też przydać:

- Symulator
- Debugger

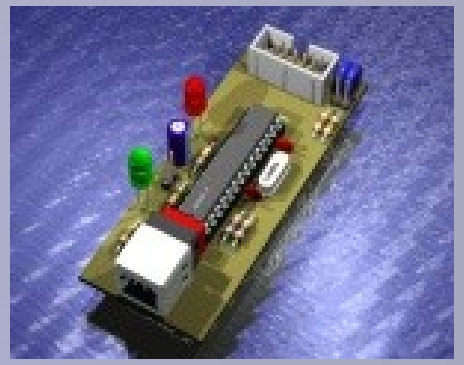
I oczywiście nasz mikrokontroler.

Opis narzędzi: Kompilator

- Będziemy używać darmowego i otwartego kompilatora GCC (wersja dla mikrokontrolerów AVR)
- Potrzebne biblioteki są już umieszczone w GCC
- Działa w systemach Linux, Windows i innych
- Interface użytkownika: linia poleceń
- Wersja dla Windows zawarta jest w pakiecie WinAVR: http://winavr.sourceforge.net/
- Precyzyjny opis kompilatora można znaleźć na: http://gcc.gnu.org/onlinedocs/gcc-4.2.3/gcc/ http://sourceware.org/binutils/docs-2.18/ (dla naprawdę dociekliwych)

Opis narzędzi: Programator

- Programator, który posiadacie to USBasp: http://www.fischl.de/usbasp/
- Komunikacja z min odbywa się przez port USB
- Oprogramowanie obsługujące programator to AVRDUDE: http://www.nongnu.org/avrdude/
- Programatora używa się z linii poleceń
- Działa pod systemem Linux i Windows
- Jest częścią pakietu WinAVR



Opis narzędzi: Edytor

- Może być dowolny edytor tekstu
- Najlepiej używać takich, które są przeznaczone do C, np.:
  - Kwrite / Kate (Linux)
  - Programmers Notepad (zawarty w WinAVR)
- Polecam bardziej zaawansowane narzędzia IDE:
  - AVR Studio: http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725
  - Eclipse CDT: http://www.eclipse.org/cdt/
  - Visual Studio Express: http://www.microsoft.com/express/
- Powyższe edytory można łatwo skonfigurować, aby współpracowały z naszymi narzędziami: kompilatorem i programatorem, dzięki czemu możemy stworzyć sobie przyjazne środowisko programistyczne. Opis takiej konfiguracji zostanie umieszczony w innych dokumentach.

Opis narzędzi: Symulator/Debugger

- AVR Studio 4
  - Wyprodukowane przez Atmel'a,
  - Pełne IDE po skonfigurowaniu kompilatora i programatora,
  - Symuluje wszystkie AVR'y,
  - Trudne lub czasami nie możliwe podłączenie wirtualnych urządzeń,
  - Do pobrania z: http://www.atmel.com/dyn/resources/prod_documents/aStudio4b589.exe
- VMLAB
  - Posiada kilka wirtualnych urządzeń, które łatwo można połączyć z symulowanym układem,
  - Wbudowany edytor i profiler,
  - Współpracuje z WinAVR,
  - Nie obsługuje wszystkich modeli,
  - Do pobrania z: http://www.amctools.com/download.htm
- Połączenie Eclipse, GDB i simulavr

Dokumentacja

- Wszystkie potrzebne informacje są w datasheet'ach do ściągnięcia z: http://www.atmel.com/dyn/products/param_table.asp?family_id=607
- Organizacja dokumentów
  - Możliwości, ogólny przegląd układu, konfiguracja pinów,
  - Rozdziały dotyczące każdego urządzenia,
  - Na końcu każdego rozdziału znajduje się opis rejestrów,

Krok po kroku: Program

```
#include <avr/io.h>

void wait (long n)
{
    volatile long i;
    for (i=0; i<n; i++);
}

void led(short on) {
    DDRB |= 1;
    if (on) {
        PORTB &= ~1;
    } else {
        PORTB |= 1;
    }
}

int main() {
    while (1) {
        led(1);
        wait(10000);
        led(0);
        wait(20000);
    }
    return 0;
}
```

Krok po kroku: Program

```
#include <avr/io.h>
```

Dodaje plik nagłówkowy zawierający deklaracje rejestrów IO. Plik ten zawiera wszystkie rejestry zawarte w Datasheet'cie konkretnego modelu. Czym jest rejestr IO? Z poziomu jezyka C jest to zmienna, która zapewnia komunikację z urządzeniami mikrokontrolera. Na schemacie ze slajdu 2 jest to połączenie CPU z urządzeniami, na slajdzie 4 jest to ta gruba szyna. Jak się ich używa, w dalszej części.

```
void wait (long n)
{
    volatile long i;
    for (i=0; i<n; i++);
}
```

Wykonuje pustą pętlę n razy, aby wstrzymać działanie programu na jakiś czas. Kompilator podczas optymalizacji może stwierdzić, że wykonywanie pustej pętli jest stratą czasu i ją usunąć, dlatego obok typu znajduje się słowo *volatile*, które mówi, że nie można optymalizować nic co operuje na zmiennej *i*.

Krok po kroku: Program

```
void led(short on) {
    DDRB |= 1;
    if (on) {
        PORTB &= ~1;
    } else {
        PORTB |= 1;
    }
}
```

`DDRB |= 1;`

Ustawia port PB0 jako port wyjściowy. Należy tutaj sięgnąć do datasheet'a, aby sprawdzić co oznacza rejestr DDRB. W rozdziale „I/O Ports” widzimy, że służy on do ustawienia kierunku. Wartość 1 ustawia port na wyjście, 0 na wejście.

Rejestry można traktować podobnie jak zwykłe zmienne. Utrudnieniem może być konieczność bitowego dostępu do nich, dlatego należy sobie przypomnieć operatory bitowe w C (~ & | ^ >> <<), np. operacja `|= 1` ustawia najmłodszy bit na 1, reszta zostaje nie zmieniona.

Aby zachować większą zgodność z datasheet'em można tą linie zapisać tak:

`DDRB |= (1 << DDB0);`

Co oznacza: ustaw jeden bit na pozycji DDB0 w rejestrze DDRB.

Krok po kroku: Program

```
PORTB &= ~1;
PORTB |= 1;
```

Jak można przeczytać w datasheet'cie, rejestr ten odpowiada za wyjście pinu. Wobec tego te operacje zmieniają wyjście PB0 na 1 (dioda się nie świeci) lub 0 (dioda zaczyna świecić).

Reszta programu powinna być zrozumiała.

Typy podstawowe w procesorach 8-bitowych:

		signed		unsigned	
	bits	**min**	**max**	**min**	**max**
char	8	-128	127	0	255
int, short	16	-32768	32767	0	65535
long	32	-2G	2G - 1	0	4G - 1
long long	64	-8E	8E - 1	0	16E - 1
float, double	32				

- Należy stosować małe typy danych, jeżeli jest to możliwe.
- Należy unikać stosowania typów zmiennoprzecinkowych
- Jeżeli się da, to należy wykorzystać typ stałoprzecinkowy np. oparty o signed long.

Krok po kroku: Konfiguracja kompilatora

- W jednym katalogu umieszczamy pliki źródłowe (w naszym przypadku main.c) oraz plik Makefile pochodzący z WinAVR (katalog sample).
- Makefile – podstawowe ustawienia:
  - model procesora: **MCU = atmega8**
  - częstotliwość taktowania (Hz): **F_CPU = 4000000**
  - lista plików źródłowych: **SRC = main.c**
  - stopień optymalizacji: **OPT = s**
- Makefile – ustawienia programatora:
  - sekcja „Programming Options” ok. linii 270
  - programator: **AVRDUDE_PROGRAMMER = usbasp**
  - port: **AVRDUDE_PORT = usb**

Krok po kroku: Kompilacja i programowanie

- W konsoli przechodzimy do naszego katalogu i wpisujemy „**make**”. W efekcie zostały utworzone:
  - main.hex – binarny kod programu, który zostanie załadowany do mikrokontrolera
  - main.elf – program wraz z informacjami dla debuggera i innymi
  - main.lss – skompilowany program w wersji czytelnej (assembler)
- Po kompilacji można wpisać w konsoli „**make program**”.
- Programator zaprogramuje skompilowany kod i natychmiast go uruchomi.
- Aby wyczyścić katalog, czyli usunąć wszystkie wygenerowane pliki lub wymusić ponowną rekompilację wszystkich plików, należy wpisać „**make clean**”.

Krok po kroku: Symulacja

- Symulujemy przy pomocy programu VMLAB
- Tworzymy nowy projekt w naszym katalogu, wybierając model mikrokontrolera oraz plik programu „main.hex”
- W edytorze projektu dodajemy nową linie, aby zobaczyć sygnał z portu PB0 na oscyloskopie (opis innych urządzeń w helpie):

```
.PLOT V(PB0)
```

- Chcemy zobaczyć to na diodzie D1:

```
R1 PB0 w1 500
D1 VDD w1
```

- Uruchamiamy symulacje
- Używamy debuggera, aby wykonywać program krok po kroku, podglądać zmienne, rejestry, itp.

Dokładniejsze opisy urządzeń mikrokontrolera, rejestrów, programów w innych dokumentach.