

Systemy Mikroprocesorowe

Piotr M. Szczypiński

Wykładowca

Piotr M. Szczypiński, Dr inż.

Instytut Elektroniki Politechniki Łódzkiej

Wólczańska 223, 90-924 Łódź

Pokój: 205

http: <http://www.eletel.p.lodz.pl/~pms/>

email: pms@p.lodz.pl

tel. +4842 631 2638

Zakres materiału

- Tło historyczne
- Wprowadzenie do obsługi AVR IDE
WinAVR, AVR Studio, MegaLoad
- Przypomnienie podstaw układów cyfrowych
Technologia, bramki, wyjścia trójstanowe, przerzutniki...
- Mikroprocesor
Budowa mikroprocesora, szyny, kod maszynowy, asembler...
- Pamięci
RAM i ROM, Dynamiczne i statyczne, dekodowanie adresu
- Mikroprocesor-pamięć-układy we/wy
Zasady transmisji danych między układami
- Reprezentacja danych w systemie binarnym
Liczby całkowite, zmiennoprzecinkowe, znaki alfanumeryczne, grafika
- Programowanie
Algorytm, języki programowania, kompilacja, konsolidacja, debugging
- Programowalne układy scalone
- Przykłady układów mikroprocesorowych

Źródła informacji

- This lecture: http://www.eletel.p.lodz.pl/~pms/dyda_pl.html
- H. Feichtinger, Mikrokomputery – poradnik, WKŁ
- P. Misiurewicz, Podstawy techniki mikroprocesorowej, WNT
- R. Baranowski, Mikrokontrolery AVR ATmega w praktyce, BTC
- J. Doliński, Mikrokontrolery AVR w praktyce, BTC
- T. Łuba, K. Jasiński, B. Zbierchowski,
Specjalizowane układy cyfrowe w strukturach PLD i FPGA, WKŁ
- Standardowe cyfrowe układy scalone
 - <http://www.standardics.philips.com/products/>
- AT Mega:
 - <http://www.atmel.com/>
 - <http://www.avrfreaks.net/>
 - <http://sourceforge.net/projects/winavr/>
 - <http://www.microsyt.com/megaload/megaload.html>
- Układy programowalne
 - <http://www.altera.com/html/literature/>
 - <http://www.latticesemi.com/products/>
 - <http://www.xilinx.com/partinfo/databook.htm>

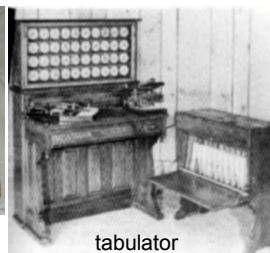
Tło historyczne

Kamienie milowe rozwoju technik obliczeniowych

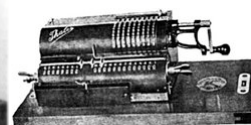
- Liczydło - wynalezione w Babilonie około 5000 lat temu
- Arytmometr mechaniczny - wynaleziony niezależnie przez Wilhema Schnickarda w 1623 i Blaisea Pascala w 1652
- System binarny - podstawy teoretyczne opracowane przez Gottfrieda Wilhelma Leibniza w 1674
- Tabulator - urządzenie do zapisu danych na kartach perforowanych wynalezione przez Hermana Holleritha w 1880, wykorzystane podczas spisu ludności w USA
- Komputer Alana Turinga - abstrakcyjny model komputera
- Komputer elektromechaniczny (przełącznikowy) - 1935, Conrad Zuse



Liczydło rzymskie i rosyjskie



tabulator



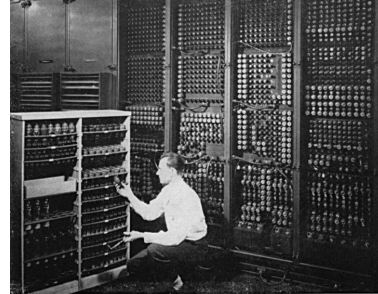
arytmometr

Źródło ilustracji: Wikipedia

Generacje komputerów

I generacja komputerów

- Eniac - komputer składający się z 1800 lamp elektronowych, zbudowany w 1946 przez na potrzeby Armii USA (zajmował 150m³, zużywał 50kW energii, wykonywał około 10000 prostych operacji arytmetycznych na sekundę)



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Źródło ilustracji: Wikipedia

II generacja komputerów

- W 1956 w MIT zaprojektowano i wykonano pierwszy komputer tranzystorowy

III generacja komputerów

- W końcu lat 60. opracowano pierwsze układy scalone zawierające nieliczne bramki logiczne. Komputery zbudowane z takich układów zaliczane są do trzeciej generacji.

IV generacja komputerów

- W budowie komputerów wykorzystano układy scalone wielkiej skali integracji (Very Large Scale of Integration - VLSI).

Pamięci

Mechaniczne - np. karty perforowane

Elektromechaniczne - przekaźnikowe

Ferrytowe

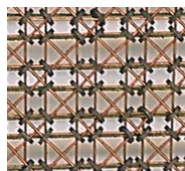
Na rurach rtęciowych

Drutowe

Bębnowe

...

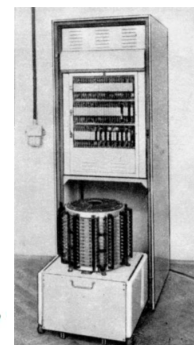
Półprzewodnikowe



pamięć ferrytowa



pamięć półprzewodnikowa



pamięć bębnowa

Źródło ilustracji: Wikipedia

Mikro...

Procesor

układ elektroniczny wykonujący obliczenia według modyfikowalnego programu

Mikroprocesor

procesor upakowany w pojedynczym układzie scalonym

Mikrokomputer

komputer zawierający mikroprocesor



Źródło ilustracji: Wikipedia

Mikroprocesory

Mikroprocesor

- 4-bitowy - 1970, TMS1000 firmy Texas Instruments
- 8-bitowy - Intel 8008 w 1972, Intel 8080 w 1974, Motorola 6800 ...
- 16-bitowy... 32-bitowy... 64-bitowy...
- ...

Komputer osobisty

Komputer osobisty - Personal computer

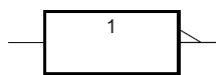
Stosunkowo tani (<\$1000),
zaprojektowany do użytku
domowego lub biurowego

- Mark 8 (do samodzielnego montażu)
1974,
- Altair (do samodzielnego montażu),
400\$, Intel 8080, 1975,
- Apple, 900\$, procesor Motorola
6502, 1976,
- IBM PC, 1981,
- Apple, MacIntosh, 1983,
- ...

Podstawowe układy logiczne przypomnienie

Bramki logiczne

Inwerter



	We2		
	0	1	?
We1	0	0	1
	1	1	1

Bufor Inwerter



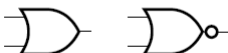
	We2		
	0	1	
We1	0	0	0
	1	0	1

AND NAND

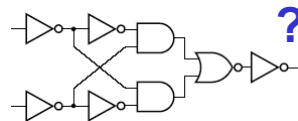


	We2		
	0	1	?
We1	0	0	1
	1	1	0

OR NOR



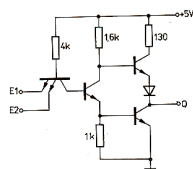
EXOR



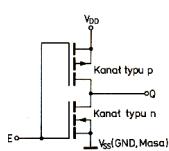
Zadanie: Połącz tablice i układ po prawej stronie z odpowiadającymi im bramkami po lewej

Bramki logiczne

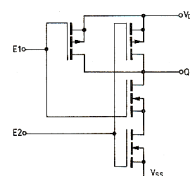
Technologia



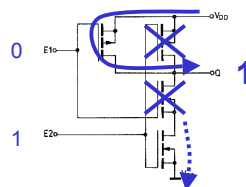
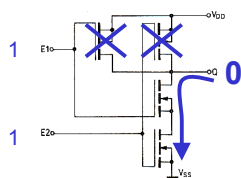
TTL - NAND



CMOS - Inwerter

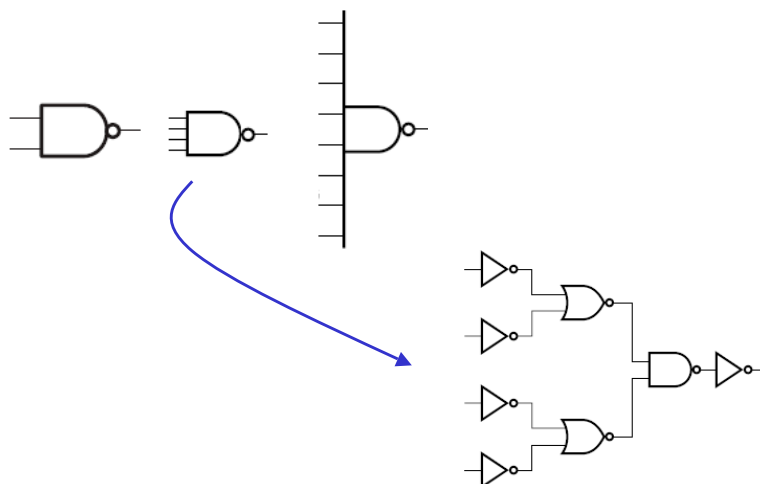


CMOS - NAND



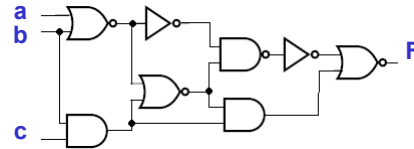
Bramki logiczne

Wielowejściowe bramki logiczne



Funkcje logiczne

Układ



Funkcja logiczna

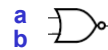
$$F = \overline{\overline{\overline{((b \cdot c) \cdot \overline{a+b} + (b \cdot c))} + a+b \cdot \overline{(a+b + (b \cdot c))}}}}$$

$$F = \sim(((b \& c) \& \sim(\sim(a|b))|(b \& c))) | \sim\sim(\sim\sim(a|b) \& \sim(a|b)|(b \& c)))$$

Tablica Karnaugh

		c	0	1
a	b			
0	0	1	1	
0	1	0	0	
1	1	0	0	
1	0	0	0	

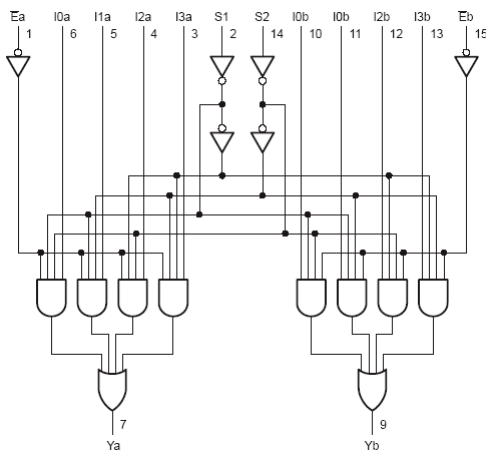
Układ i funkcja po uproszczeniu



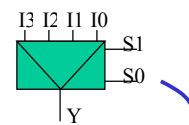
$$F = \overline{a+b}$$

$$F = \sim(a|b)$$

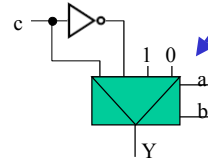
Multipleksery



Dwa 4-wejściowe multipleksery
z 2. wspólnymi wejściami adresującymi
74f153 - <http://www.standardics.philips.com/products/>

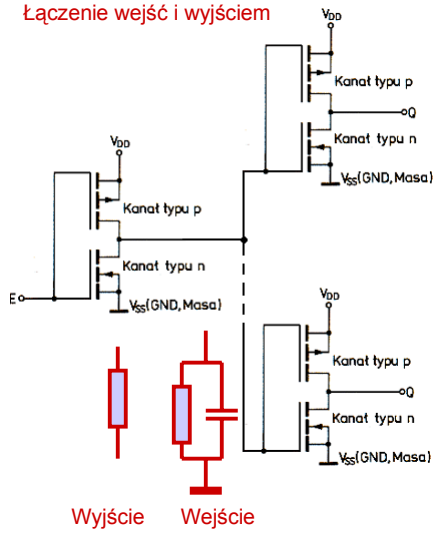


		c	0	1
a	b			
0	0	0	0	
0	1	1	1	
1	1	0	1	
1	0	1	0	



Bramki logiczne

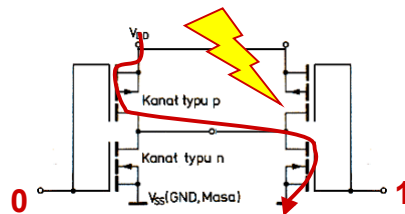
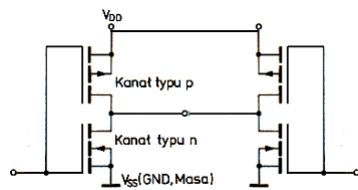
Łączenie wejść i wyjściem



- Sprawdź na stronie producenta parametry typowych cyfrowych układów scalonych CMOS:
 - częstotliwości pracy
 - opóźnienia sygnału
 - impedancję wejściową bramki
 - impedancję wyjściową zwykłej bramki
 - impedancję wyjścia bramki trójstanowej w czasie gdy jest ona odłączona

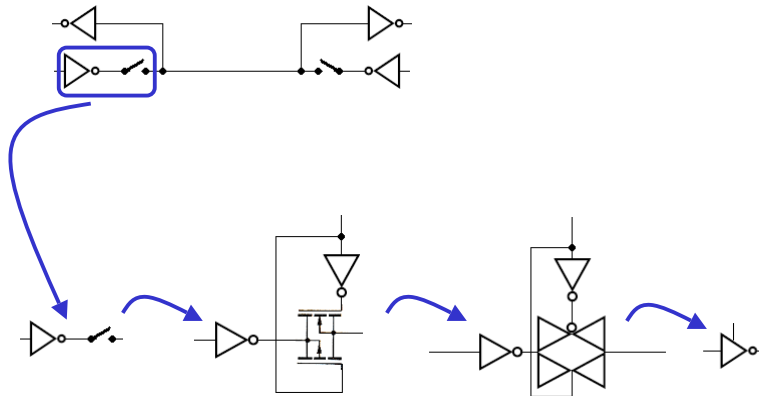
Bramki logiczne

Łączenie wyjść bramek

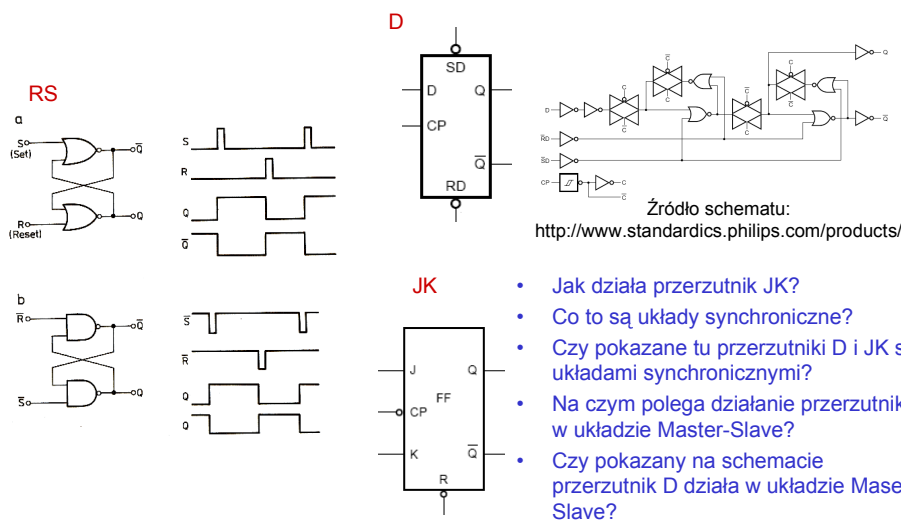


Bramki logiczne

Wyjścia trójstanowe



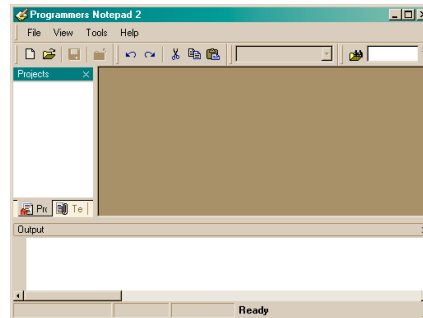
Przerzutniki



- Jak działa przerzutnik JK?
- Co to są układy synchroniczne?
- Czy pokazane tu przerzutniki D i JK są układami synchronicznymi?
- Na czym polega działanie przerzutnika w układzie Master-Slave?
- Czy pokazany na schemacie przerzutnik D działa w układzie Maser-Slave?

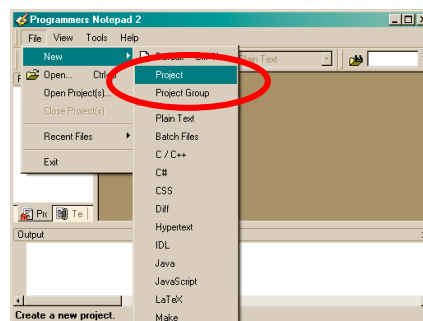
WinAVR: Nowy projekt

1. Uruchom *Programmers Notepad (PN)*
2. Wybierz *File -> NewProject*
3. Utwórz nowy folder dla projektu
4. Utwórz nowy plik projektu
5. Utwórz nowy plik w języku C
6. Zapisz pliki do foldera projektu
7. Uruchom *MFile*
8. Ustaw podstawową nazwę pliku
9. Ustaw jednostkę *ATMega 128*
10. Wyłącz optymalizację (ustawić na 0)
11. Ustaw typ danych wyjściowych na *Extended COFF*
11. Zapisz *makefile* do foldera projektu
12. Dodaj pliki do projektu (w *PN*)



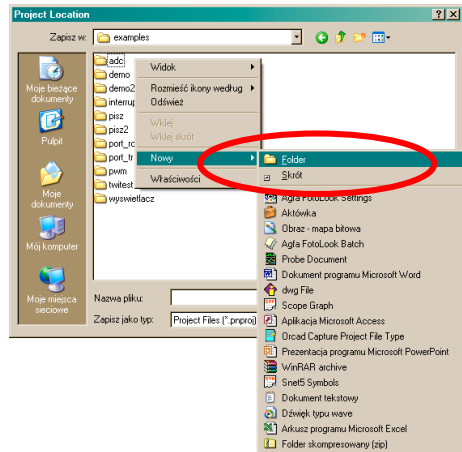
WinAVR: Nowy projekt

1. Uruchom *Programmers Notepad (PN)*
2. Wybierz *File->NewProject*
3. Utwórz nowy folder dla projektu
4. Utwórz nowy plik projektu
5. Utwórz nowy plik w języku C
6. Zapisz pliki do foldera projektu
7. Uruchom *MFile*
8. Ustaw podstawową nazwę pliku
9. Ustaw jednostkę *ATMega 128*
10. Wyłącz optymalizację (ustawić na 0)
11. Ustaw typ danych wyjściowych na *Extended COFF*
11. Zapisz *makefile* do foldera projektu
12. Dodaj pliki do projektu (w *PN*)



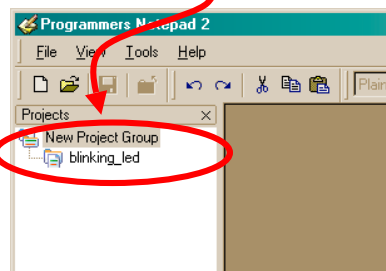
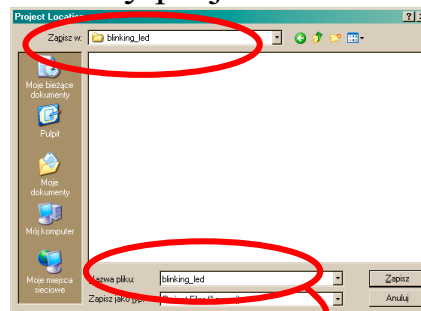
WinAVR: Nowy projekt

1. Uruchom *Programmers Notepad (PN)*
2. Wybierz *File->NewProject*
3. Utwórz nowy folder dla projektu
4. Utwórz nowy plik projektu
5. Utwórz nowy plik w języku C
6. Zapisz pliki do foldera projektu
7. Uruchom *MFile*
8. Ustaw podstawową nazwę pliku
9. Ustaw jednostkę *ATMega 128*
10. Wyłącz optymalizację (ustawić na 0)
11. Ustaw typ danych wyjściowych na *Extended COFF*
11. Zapisz *makefile* do foldera projektu
12. Dodaj pliki do projektu (w *PN*)



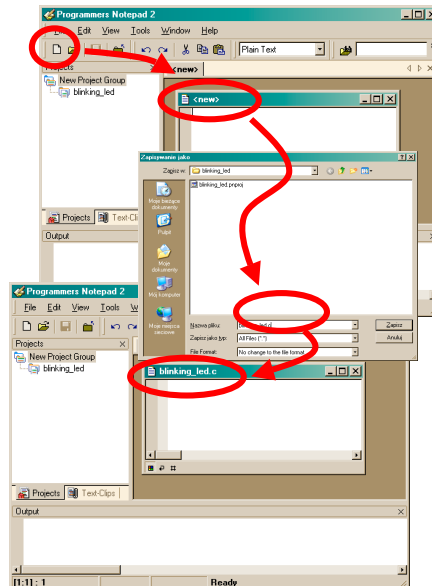
WinAVR: Nowy projekt

1. Uruchom *Programmers Notepad (PN)*
2. Wybierz *File->NewProject*
3. Utwórz nowy folder dla projektu
4. Utwórz nowy plik projektu
5. Utwórz nowy plik w języku C
6. Zapisz pliki do foldera projektu
7. Uruchom *MFile*
8. Ustaw podstawową nazwę pliku
9. Ustaw jednostkę *ATMega 128*
10. Wyłącz optymalizację (ustawić na 0)
11. Ustaw typ danych wyjściowych na *Extended COFF*
11. Zapisz *makefile* do foldera projektu
12. Dodaj pliki do projektu (w *PN*)



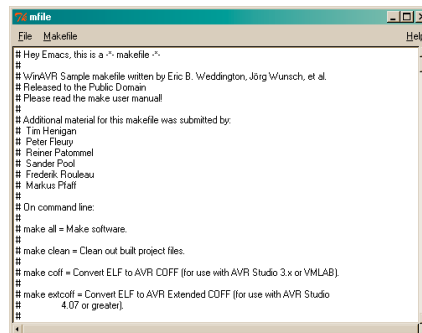
WinAVR: Nowy projekt

1. Uruchom *Programmers Notepad (PN)*
2. Wybierz *File->NewProject*
3. Utwórz nowy folder dla projektu
4. Utwórz nowy plik projektu
5. Utwórz nowy plik w języku C
6. Zapisz pliki do foldera projektu
7. Uruchom *MFile*
8. Ustaw podstawową nazwę pliku
9. Ustaw jednostkę *ATMega 128*
10. Wyłącz optymalizację (ustawić na 0)
11. Ustaw typ danych wyjściowych na *Extended COFF*
11. Zapisz *makefile* do foldera projektu
12. Dodaj pliki do projektu (w *PN*)



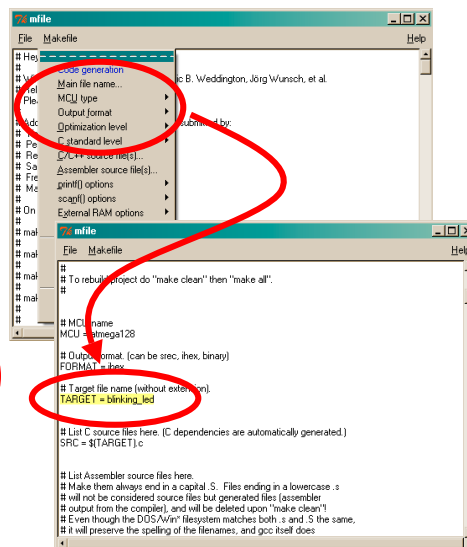
WinAVR: Nowy projekt

1. Uruchom *Programmers Notepad (PN)*
2. Wybierz *File->NewProject*
3. Utwórz nowy folder dla projektu
4. Utwórz nowy plik projektu
5. Utwórz nowy plik w języku C
6. Zapisz pliki do foldera projektu
7. Uruchom *MFile*
8. Ustaw podstawową nazwę pliku
9. Ustaw jednostkę *ATMega 128*
10. Wyłącz optymalizację (ustawić na 0)
11. Ustaw typ danych wyjściowych na *Extended COFF*
11. Zapisz *makefile* do foldera projektu
12. Dodaj pliki do projektu (w *PN*)



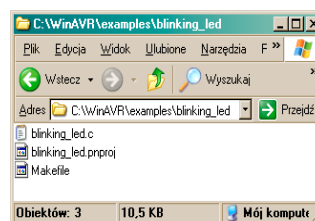
WinAVR: Nowy projekt

1. Uruchom *Programmers Notepad (PN)*
2. Wybierz *File->NewProject*
3. Utwórz nowy folder dla projektu
4. Utwórz nowy plik projektu
5. Utwórz nowy plik w języku C
6. Zapisz pliki do foldera projektu
7. Uruchom *MFile*
8. Ustaw podstawową nazwę pliku
9. Ustaw jednostkę *ATMega 128*
10. Wyłącz optymalizację (ustawić na 0)
11. Ustaw typ danych wyjściowych na *Extended COFF*
11. Zapisz *makefile* do foldera projektu
12. Dodaj pliki do projektu (w *PN*)



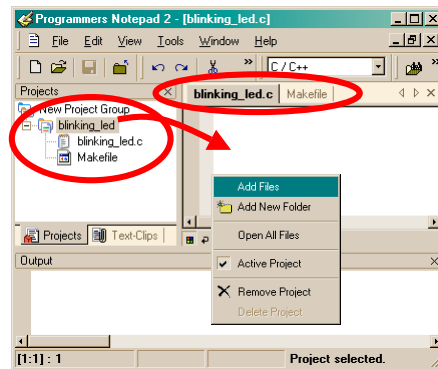
WinAVR: Nowy projekt

1. Uruchom *Programmers Notepad (PN)*
2. Wybierz *File->NewProject*
3. Utwórz nowy folder dla projektu
4. Utwórz nowy plik projektu
5. Utwórz nowy plik w języku C
6. Zapisz pliki do foldera projektu
7. Uruchom *MFile*
8. Ustaw podstawową nazwę pliku
9. Ustaw jednostkę *ATMega 128*
10. Wyłącz optymalizację (ustawić na 0)
11. Ustaw typ danych wyjściowych na *Extended COFF*
11. Zapisz *makefile* do foldera projektu
12. Dodaj pliki do projektu (w *PN*)

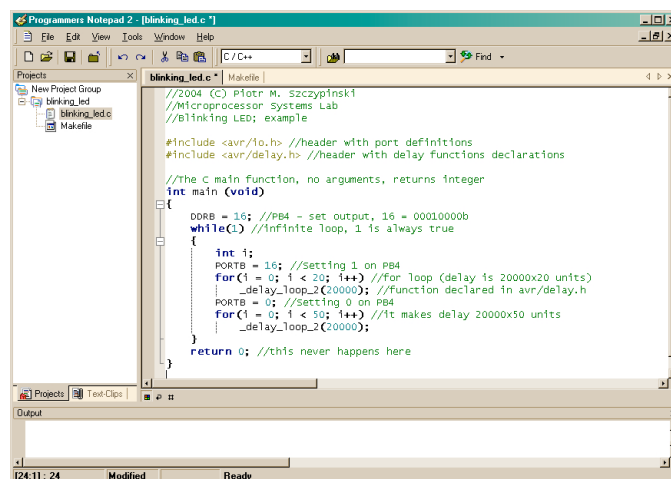


WinAVR: Nowy projekt

1. Uruchom *Programmers Notepad (PN)*
2. Wybierz *File->NewProject*
3. Utwórz nowy folder dla projektu
4. Utwórz nowy plik projektu
5. Utwórz nowy plik w języku C
6. Zapisz pliki do foldera projektu
7. Uruchom *MFile*
8. Ustaw podstawową nazwę pliku
9. Ustaw jednostkę *ATMega 128*
10. Wyłącz optymalizację (ustawić na 0)
11. Ustaw typ danych wyjściowych na *Extended COFF*
11. Zapisz *makefile* do foldera projektu
12. Dodaj pliki do projektu (w *PN*)



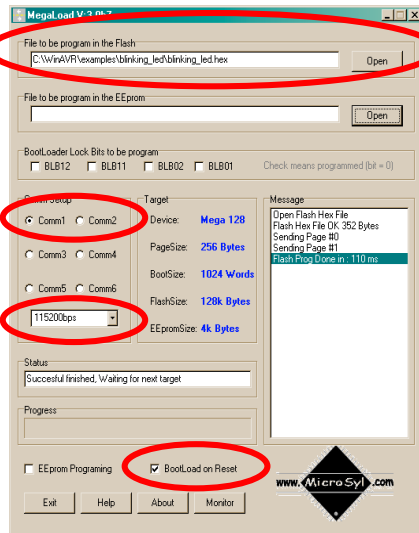
WinAVR: Edycja pliku programu

The screenshot shows the 'Programmers Notepad 2' interface with the source code for a blinking LED program. The code is as follows:

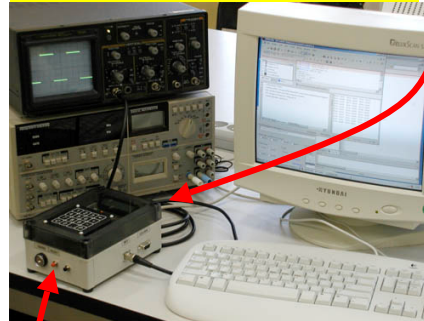
```
//2004 (C) Piotr M. Szczygiński  
//Microprocessor Systems Lab  
//blinking LED; example  
  
#include <avr/io.h> //header with port definitions  
#include <avr/delay.h> //header with delay functions declarations  
  
//The C main Function, no arguments, returns integer  
int main (void)  
{  
    DDRA = 0x00; //PB4 - set output, 16 = 00010000b  
    while(1) //infinite loop, 1 is always true  
    {  
        int i;  
        PORTB = 16; //Setting 1 on PB4  
        for(i = 0; i < 20; i++) //for loop (delay is 20000x20 units)  
        {  
            _delay_loop_2(20000); //function declared in avr/delay.h  
            PORTB = 0; //Setting 0 on PB4  
            for(i = 0; i < 50; i++) //it makes delay 20000x50 units  
            {  
                _delay_loop_2(20000);  
            }  
        }  
        return 0; //this never happens here  
    }  
}
```

Nie zapomnij zapisać zmienionych plików!

MegaLoad: Ładowanie programu



Pamiętaj by połączyć Zestaw ATmega (RS1 connector) z komputerem PC (COM1 or COM2)!



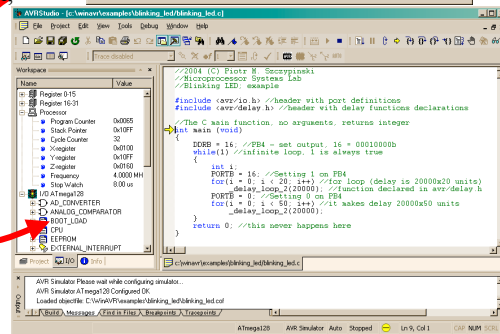
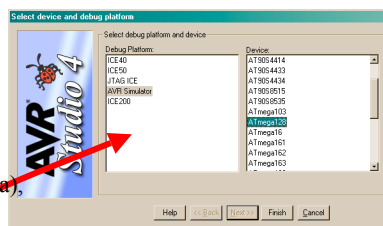
Wciśnij RESET by załadować program

WinAVR → AVR Studio

AVR Studio jest środowiskiem programistycznym (IDE) do uruchamiania i testowania napisanych programów.

Aby załadować projekt utworzony w WinAVR do AVR Studio należy:

1. Uruchomić AVR Studio (wer. 4.07 lub późniejsza),
2. Zamknąć okno *Welcome* jeśli się pojawi,
3. Wybrać *File->Open file...* i załadować plik z rozszerzeniem *cof*,
4. Wybrać układ do symulacji: *AVR Simulator* i *ATmega 128*
5. Po załadowaniu projektu *AVR Studio* jest gotowe do pracy (*debugowania*)



AVR Studio: Co należy wiedzieć

Znajdź w dokumentacji AVR Studio:

1. Do czego służą narzędzia: *watch view*, *register window*, *memory window* i *disassembler*?



2. Jak uruchomić program, zatrzymać jego wykonywanie i zresetować program?



3. Jaka jest różnica między *trace into*, *step over* a *step out*?

4. Co to jest pułapka (*breakpoint*)?

**Metody reprezentacji
danych w systemie binarnym**

System binarny

Byte (bajt) jest podstawową jednostką pojemności pamięci komputerowej. W przypadku nowoczesnych komputerów przyjmuje się, że bajt składa się z 8. bitów (8. cyfr w systemie dwójkowym).

Multiples of bytes as defined by IEC 60027-2					
SI prefix			Binary prefixes		
Name	Symbol	Multiple	Name	Symbol	Multiple
kilobyte	kB	10^3 (or 2^{10})	kibibyte	KiB	2^{10}
megabyte	MB	10^6 (or 2^{20})	mebibyte	MiB	2^{20}
gigabyte	GB	10^9 (or 2^{30})	gibibyte	GiB	2^{30}
terabyte	TB	10^{12} (or 2^{40})	tebibyte	TiB	2^{40}
petabyte	PB	10^{15} (or 2^{50})	pebibyte	PiB	2^{50}
exabyte	EB	10^{18} (or 2^{60})	exbibyte	EiB	2^{60}
zettabyte	ZB	10^{21} (or 2^{70})			
yottabyte	YB	10^{24} (or 2^{80})			

(Wikipedia)

Word (słowo) jest określeniem ilości bitów, które w sposób naturalny mogą być jednocześnie przetwarzane przez określony system komputerowy. Liczba bitów składających się na słowo zależy od architektury określonego systemu komputerowego.

System binarny

Binarny system liczbowy przedstawia wartości liczb za pomocą dwóch symboli (cyfr), zazwyczaj 0 i 1.

Zamiana na system dziesiętny:

X	...	1	0	0	1	1	0
2^n	...	2^5	2^4	2^3	2^2	2^1	2^0
		32	16	8	4	2	1

$$1 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 38$$

4-bit numbers	
Binary	Decimal
1111	15
1110	14
1101	13
1100	12
1011	11
1010	10
1001	9
1000	8
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0

Zamiana na system szesnastkowy:

4-bit numbers	
Binary	Hexadecimal
1111	0xF
1110	0xE
1101	0xD
1100	0xC
1011	0xB
1010	0xA
1001	0x9
1000	0x8
0111	0x7
0110	0x6
0101	0x5
0100	0x4
0011	0x3
0010	0x2
0001	0x1
0000	0x0

Binarnie: 1001 1101 0110 0001 1110
Szesnastkowo: 9 D 6 1 E

System binarny

Uzupełnij:

Wartość 8. bitowa	
Binarna	Dziesiętna
11111111	255
11111110	254
...	...
11101000	
11100111	
...	...
10000001	129
10000000	128
01111111	127
01111110	126
...	...
00100111	
00100110	
00100101	
...	...
00000011	3
00000010	2
00000001	1
00000000	0

Zamień liczby w systemie binarnym na odpowiadające im liczby w systemach dziesiętnym i szesnastkowym

110100100100
111111111111
000100000000
000000010000
000011111111

Zamień na system binarny:

Dziesiętnie: 128, 1276, 2048, 17
Szesnastkowo: 0xFF, 0x1111, 0x81

Endianness - kolejność bajtów

W sytuacjach kiedy dane zapisywane są przy użyciu wielu (przynajmniej dwóch) bajtów nie istnieje jeden określony sposób uporządkowania tych bajtów w pamięci lub w czasie transmisji. Musi być użyta jedna z wielu konwencji ustalającej **kolejność bajtów** (ang. "byte order" lub "endianness"). (Wikipedia)

Przykład:

00010111 01101100 = Dziesiętnie 5996

Adres	Dana
n+2	...
n+1	01101100
n	00010111
n-1	...

Adres	Dana
n+2	...
n+1	00010111
n	01101100
n-1	...

Big-endian - najbardziej znaczący bajt (**MSB**) danych zapisywany jest w pamięci jako pierwszy (pod niższym adresem). Następny bajt według znaczenia zapisywany jest pod następnym adresem pamięci, itd.

Little-endian - najmniej znaczący bajt (**LSB**) danych zapisywany jest w pamięci jako pierwszy (pod niższym adresem).

Liczby ze znakiem

Metoda uzupełnień do dwóch (U2)

Policzmy w dół (liczby 9. bitowe):

1 0000 0011 = dec. 259
 1 0000 0010 = dec. 258
 1 0000 0001 = dec. 257
 1 0000 0000 = dec. 256
 0 1111 1111 = dec. 255
 0 1111 1110 = dec. 254
 0 1111 1101 = dec. 253

Usuńmy 9. ty, najbardziej znaczący bit:

~~1~~ 0 000 0011 = dec. 259 - 256 = 3
~~1~~ 0 000 0010 = dec. 258 - 256 = 2
~~1~~ 0 000 0001 = dec. 257 - 256 = 1
~~1~~ 0 000 0000 = dec. 256 - 256 = 0
~~0~~ 1 111 1111 = dec. 255 - 256 = -1
~~0~~ 1 111 1110 = dec. 254 - 256 = -2
~~0~~ 1 111 1101 = dec. 253 - 256 = -3

Znak



Liczby ze znakiem

Metoda uzupełnień do dwóch (U2)

Uzupełnij:

Binarnie U2	Dziesiętnie
01111111	127
01111110	126
...	...
00100111	
00100110	
00100101	
...	...
00000011	3
00000010	2
00000001	1
00000000	0
11111111	-1
11111110	-2
...	...
11101000	
11100111	
...	...
10000001	-127
10000000	-128

Zmiana znaku w U2

Algorytm

1. Oblicz negację bitową wszystkich bitów (~)
2. Dodaj jeden (+1)

Przykład:

Dziesiętnie 12	00001100
Negacja bitowa	11110011
+1	11110100
Dziesiętnie -12	
Negacja bitowa	00001011
+1	00001100
Dziesiętnie 12	

System Binarny

Liczby stałoprzecinkowe

Liczba całkowita:

X	...	0	0	1	0	0	1	1	0
2^n	...	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
		128	64	32	16	8	4	2	1

= Dziesiętnie 38

Dodajmy przecinek gdzieś pomiędzy bitami:

X	...	0	0	1	0	0	1	1	0
2^k	...	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
		16	8	4	2	1	1/2	1/4	1/8

= Dziesiętnie $4 \frac{3}{4}$

Cz. całkowita

Cz. ułamkowa

$$38 \cdot 2^{-3} = 38 / 8 = 4 \frac{3}{4}$$

System Binarny

Liczby zmiennoprzecinkowe

Liczba zmiennoprzecinkowa jest binarną reprezentacją liczby wymiernej, używaną do reprezentowania przybliżonej wartości liczb rzeczywistych. Liczba taka składa się z grup bitów mantysy (części ułamkowej), cechy (wykładnika o podstawie 2) oraz bitu znaku.

System Binarny

Liczby zmiennoprzecinkowe

Pojedyncza precyzja (32-bit) *IEEE 754 Std.*

S	E		M		
sign	exponent (bias +127)		fraction (mantissa)		
31	30	23	22		0

$$\text{Wartość} = (-1)^S 2^{E-127} (1.M)$$

Wartość E zamieniona jest tu na dziesiętną bez znaku (dla $0 < E < 255$)

Część ułamkowa

Wyjątki:

S=0, E=255, M=0	$+\infty$
S=1, E=255, M=0	$-\infty$
E=255	NaN
S=0, E=0, M=0	0
E=0	denormalised no. = $=(-1)^S 2^{-126} (0.M)$

System Binarny

Liczby zmiennoprzecinkowe

S	E	M
sign	exponent (bias +127)	fraction (mantissa)
31	30	23 22 0

Wartość = $(-1)^S 2^{E-127} (1.M)$

Przykład 1:

Zamień **1 10000010 110100000...0** na liczbę dziesiętną

$$S = 1 \quad E = 10000010_2 = 130 \quad 1.M = 1.1101_2 = 1 \frac{13}{16}$$

$$FP = (-1)^1 2^{130-127} 1.1101_2 = -2^3 1.1101_2 = -1110.1_2 = -14 \frac{1}{2}$$

Przykład 2:

Zapisz π jako liczbę binarną, zmiennoprzecinkową

$$\begin{aligned} \pi &= 3.14159265358979 = 3 + 0.14159265358979 \times 2^{23} / 2^{23} = \\ &\approx 3 + 1187765 / 2^{23} = 3 + 0.100100001111110110101_2 = \\ &= 11.100100001111110110101_2 = \\ &= 2^{128-127} 1.1100100001111110110101_2 \end{aligned}$$

0 10000000 11001000011111101101010

System Binarny

Liczby zmiennoprzecinkowe

Podwójna precyzja (64.bity) *IEEE 754 Std.* :

S	E	M
sign	exponent (bias +1023)	fraction (mantissa)
63	62	52 51 0

Wartość = $(-1)^S 2^{E-1023} (1.M)$

System Binarny

Binary-coded decimals (BCD)

W **BCD** (SBCD 8421) każda cyfra dziesiętna kodowana jest binarnym kodem składającym się z 4. bitów. Zazwyczaj bit po lewej ma wagę 8, następny 4, 2, i 1. Jedynie kombinacje bitów dające w wyniku wartość od 0 do 9 są dopuszczalne.

Digit	SBCD 8421	Excess-3	BCD 2421	BCD 84-2-1	IBM 702 IBM 705 IBM 7080 IBM 1401 8421
0	0000	0011	0000	0000	1010
1	0001	0100	0001	0111	0001
2	0010	0101	0010	0110	0010
3	0011	0110	0011	0101	0011
4	0100	0111	0100	0100	0100
5	0101	1000	1011	1011	0101
6	0110	1001	1100	1010	0110
7	0111	1010	1101	1001	0111
8	1000	1011	1110	1000	1000
9	1001	1100	1111	1111	1001

(Wikipedia)

System Binarny

Binary-coded decimals (BCD)

Przykład 1:

BCD 1001 0010 0101 0110
 Dziesiętnie 9 3 5 6

Digit	SBCD 8421
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Przykład 2:

BCD 1011 0010 1101 0110
 Dziesiętnie Invalid 3 Invalid 6

System Binarny

Znaki alfanumeryczne i teksty

ASCII (American Standard Code for Information Interchange) jest liczbową reprezentacją znaków alfanumerycznych (t.j.: 'a', '5' lub '#') lub kodami sterującymi dla urządzeń drukujących i wyświetlających tekst. ASCII opublikowano w 1963 (zdefiniowano kody dużych liter) i w 1967.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENO (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com

System Binarny

Znaki alfanumeryczne i teksty

Przykład:

10001110 1101111 01101111 01100100 00001101 00001010

G o o d [CR] [LF]

1100111 01110101 01111001 01110011 00000000

g u y s [NULL]

Koniec łańcucha tekstu w C

Przejsięcie do nowej linii

System Binarny

Znaki alfanumeryczne i teksty

ASCII (**kod 7. bitowy**) jest bardzo prosty i nie pozwala na kodowanie znaków specyficznych dla języków innych niż angielski. Istnieją rozszerzone "standardy ASCII" (**kod 8. bitowy**) opracowane dla innych alfabetów, lecz są wzajemnie niezgodne. Unicode (**zazwyczaj kodowany 16. bitami**) to komputerowy zestaw znaków mający w zamierzeniu obejmować wszystkie pisma używane na świecie.

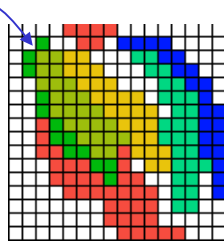
Dodatkowe informacje:

- <http://www.lookuptables.com/>
- <http://www.unicode.org/>

Obrazy cyfrowe

Obrazy Rastrowe

Obrazy **rastrowe** to struktury danych binarnych reprezentujące sieć prostokątną punktów obrazu (pikseli). Każdy z takich punktów ma indywidualnie przypisaną daną określającą jego kolor lub poziom jasności.



Nagłówek

```
42 4D 36 05 00 00 00 00 00 36 04 00 00 28 00
00 00 10 00 00 00 10 00 00 00 01 00 08 00 00 00
00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 80 00 00 80
00 00 80 80 00 80 00 00 80 00 80 00 80 80 80
```

Dane obrazowe

```
FF FF FF FF 4F 4F 4F 4F FF FF FF FF FF FF FF
FF FF 31 FF FF 4F 4F FF FC FC FC FC FF FF FF FF
FF 30 35 35 37 37 FF FF FF BA BA FC FC FF FF FF
FF 30 35 35 37 37 37 FF FF FF BA BA FC FC FF FF
FF FF 35 35 35 37 37 37 FF FF FF BA BA FC FC FF
FF FF 31 35 35 35 37 37 37 FF BA BA BA FC FC FF
FF FF 30 35 35 35 35 37 37 37 FF BA BA FC FC FF
FF FF 4F 30 35 35 35 35 37 37 BA BA BA FC FC FF
FF FF 4F 30 30 35 35 35 37 37 BA BA BA FC FC FF
FF FF 4F 4F 4F 30 30 4F FF 37 37 BA BA BA FC
FF FF FF 4F 4F 4F 30 30 4F FF 37 37 BA BA BA FC
FF FF FF 4F 4F 4F 30 4F 4F FF FF BA BA FF FC
FF FF FF FF 4F 4F 4F 4F 4F 4F FF BA BA FF FC
FF FF FF FF FF FF 4F 4F 4F 4F FF BA BA FF FC
FF FF FF FF FF FF 4F 4F 4F 4F FF FF FF FF FF
FF FF FF FF FF FF 4F 4F 4F 4F FF FF FF FF FF
```

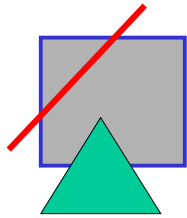
Paleta barw

```
40 00 40
40 60 00
40 40 80
00 40 40
A0 00 40
40 C0 00
40 40 E0
...
...
...
```

Obrazy cyfrowe

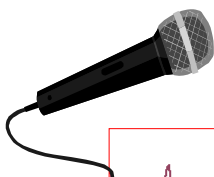
Obrazy Wektorowe

Obrazy wektorowe to dane opisujące obraz za pomocą tzw. prymitywów czyli podstawowych kształtów, takich jak punkty, linie, krzywe, wielokąty, itp. Kształt, kolor i położenie każdego z prymitywów kodowane są za pomocą kodów liczbowych.



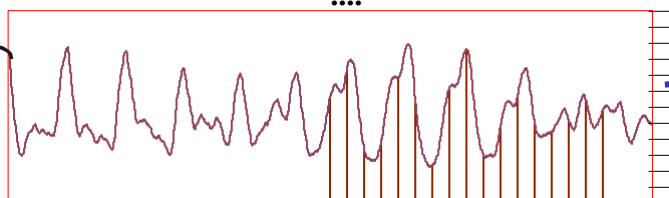
SetPixelV(kontekst, x, y, RGB(r, g, b));
MoveToEx(kontekst, x, y, NULL);
LineTo(kontekst, x, y);
Rectangle(kontekst, x1, y1, x2, y2);

Dźwięk



.... 10000010 11010000

....



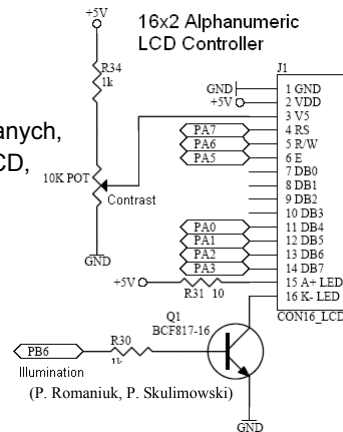
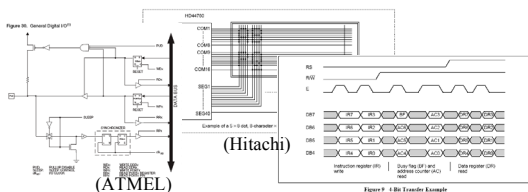
Próbkowanie i cyfryzacja dźwięku

Programowanie układów wejścia/wyjścia na przykładzie laboratoryjnego systemu z mikrokontrolerem ATMega128

Laboratoryjny System ATMega 128 Obsługa wyświetlacza

Wykład prowadzony jest z wykorzystaniem dokumentacji mikrokontrolera ATMega 128 firmy ATMEL® oraz dokumentacji sterownika wyświetlacza ciekłokrystalicznego (LCD) HD44780U firmy Hitachi®.

1. Schemat wewnętrzny portów ATMega 128,
2. Rejestry sterowania portami (PORTx, DDRx),
3. Funkcje wyprowadzeń sterownika wyświetlacza,
4. Konfiguracja sterownika z 4. i 8. bitową szyną danych,
5. Przesyłanie kodów konfigurujących sterownik LCD,
6. Wysyłanie znaków alfanumerycznych.

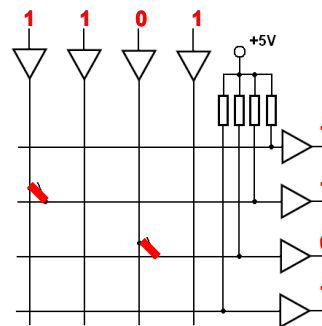
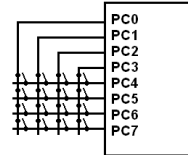
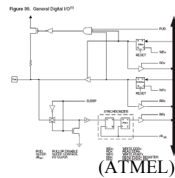


Laboratoryjny System ATmega 128

Klawiatura 4x4

Wykład prowadzony jest z wykorzystaniem dokumentacji mikrokontrolera ATmega 128 firmy ATMEL®.

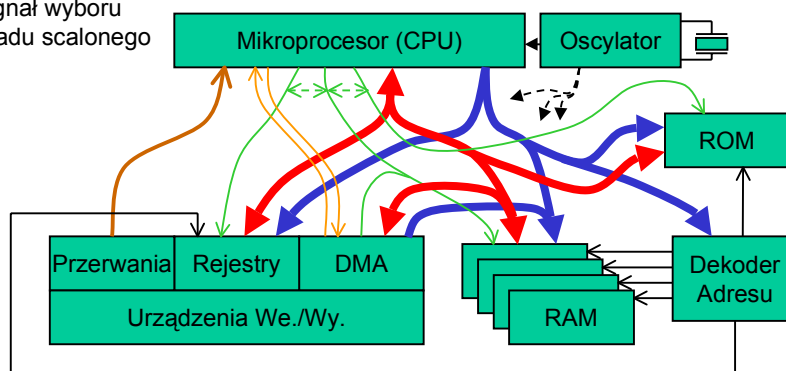
1. Schemat układu klawiatury,
2. Metody identyfikacji wciśniętego klawisza,
3. Schemat wewnętrzny portów ATmega 128,
4. Rejestry sterowania portami (PINx, SFIOR bit PUD).



Architektura systemu komputerowego

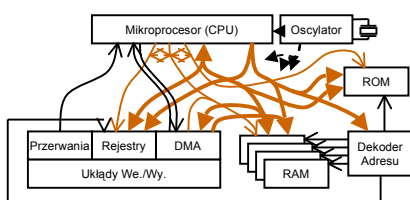
Architektura systemu komputerowego

- Sygnał żądania przerwania
- ↔ Sygnały DMA (bezpośredniego dostępu do pamięci)
- Sygnały zapisu i odczytu
- ↔ Magistrała adresowa
- ↔ Magistrała danych
- Sygnał wyboru układu scalonego



Architektura systemu komputerowego

Magistrała danych, adresowa i sterująca



Magistrała lub **szyna** w systemie komputerowym to system połączeń służący do przesyłania danych lub zasilania pomiędzy układami wewnętrznymi komputera. Magistrała (np. danych lub adresowa) to grupy równoległych połączeń (przewodów) łączących podzespoły systemu komputerowego, z których każdy służy do przesyłania sygnału logicznego.

Magistrała adresowa służy do przesyłania adresu czyli wartości binarnej określającej gdzie ma zostać zapisana dana lub skąd ma zostać pobrana dana.

Magistrała danych służy do przesyłania danej, wartości binarnej w czasie zapisu lub odczytu.

Magistrała sterująca jest wykorzystywana w systemie komputerowym do komunikacji pomiędzy układami tego systemu.

Gdzie

Co

Jak

Przesłanie danej (np. instrukcji programu) pomiędzy pamięcią a mikroprocesorem wymaga podania przez procesor miejsca w pamięci (magistrała adresowa) gdzie znajduje się dana, wysłania do pamięci sygnału żądania odczytu (magistrała sterująca) i odczytu wartości danej poprzez magistralę danych.

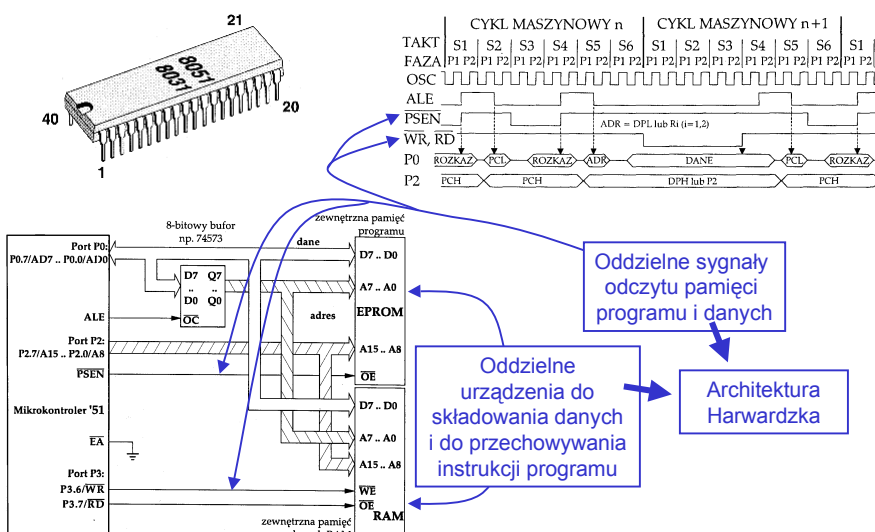
Architektura systemu komputerowego

Architektura von Neumanna oznacza taki system komputerowy, w którym zarówno program wykonywany przez system jak i dane przetwarzane przez ten system zapisane są w tym samym urządzeniu do składowania danych. Określenie architektura von Neumanna używane jest też w znaczeniu rozdzielania układów odpowiedzialnych za składowanie danych i za ich przetwarzanie.

Architektura Harwardzka oznacza taki system komputerowy w którym stosowane są oddzielne urządzenia do przechowywania instrukcji programu i oddzielne dla przetwarzanych danych. Określenie to pochodzi od komputera przekaźnikowego Harvard Mark I, w którym instrukcje programu zapisane były na taśmie perforowanej a dane w pamięci wykorzystującej przekaźniki.

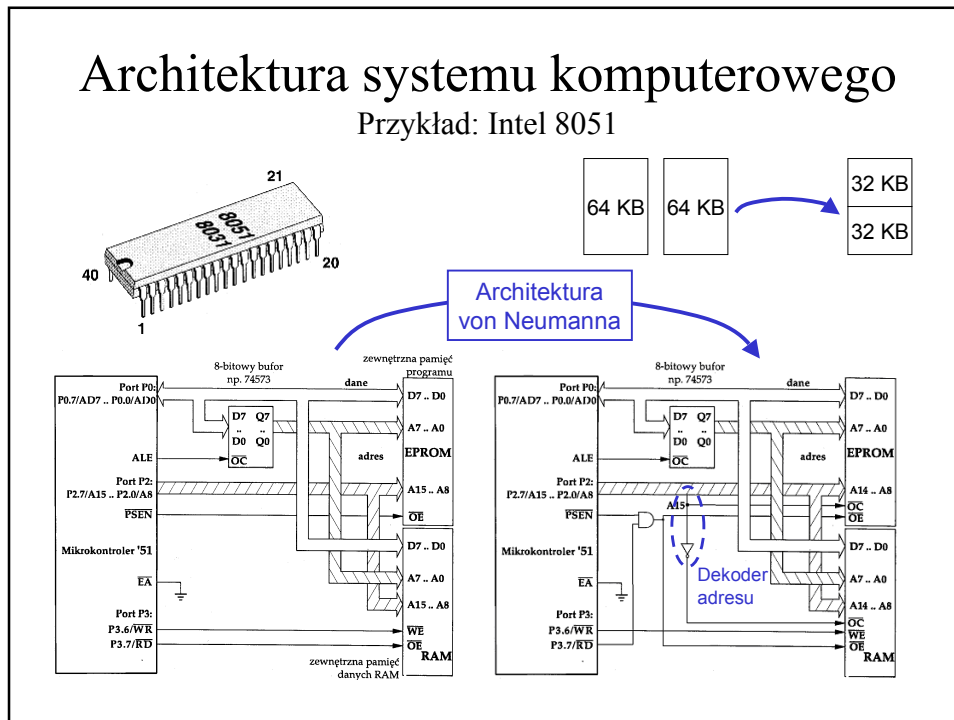
Architektura systemu komputerowego

Przykład: Intel 8051



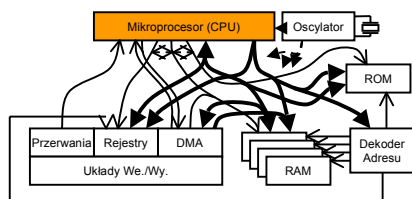
Architektura systemu komputerowego

Przykład: Intel 8051

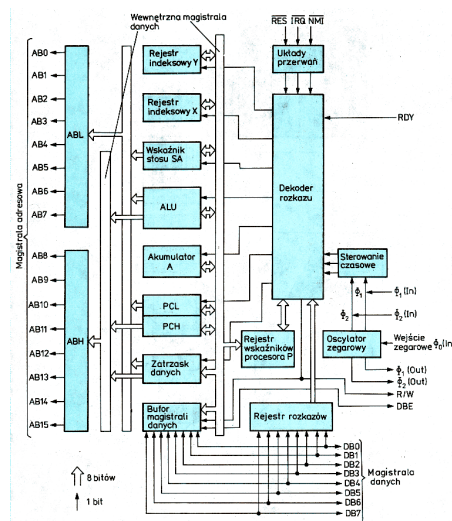


Architektura systemu komputerowego

Mikroprocesor (CPU)



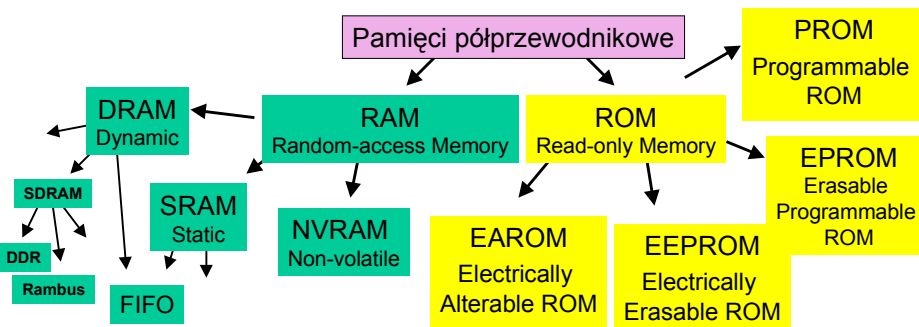
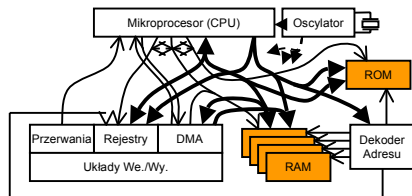
Mikroprocesor (CPU - Central Processing Unit) to część systemu komputerowego, która odczytuje, interpretuje i wykonuje instrukcje.



Motorola 6800 (H. Feichtinger, Mikrokomputery – poradnik)

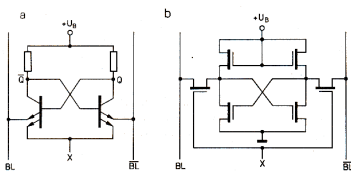
Architektura systemu komputerowego

Pamięć

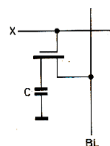


Architektura systemu komputerowego

Pamięć statyczna i dynamiczna RAM



TTL
Komórka pamięci statycznej



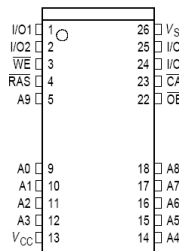
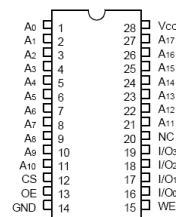
Komórka pamięci dynamicznej

Pamięć statyczna RAM zachowuje swoją zawartość tak długo jak dołączone jest do niej zasilanie.

- Szybki dostęp,
- Małe pojemności.

Pamięć dynamiczna RAM wymaga odświeżania.

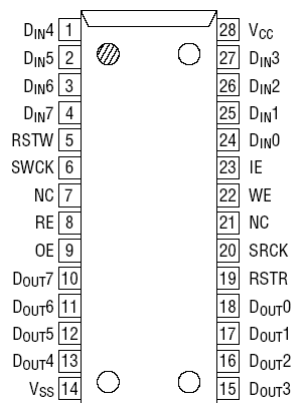
- Wydłużony czas dostępu,
- Duże pojemności.



Architektura systemu komputerowego FIFO

Pamięć FIFO (First In, First Out - pierwszy wchodzi, pierwszy wychodzi) jest pamięcią, w której dane organizowane są w formie kolejki. Dane muszą być odczytane z pamięci w tej samej kolejności w jakiej zostały do tej pamięci wpisane. Pamięci FIFO wykorzystywane są w układach do buforowania danych.

Pamięć LIFO (Last In, First Out - ostatni wchodzi, pierwszy wychodzi)



MSM518221A (OKI Semiconductors)

Architektura systemu komputerowego Dynamiczne RAM: SDRAM, DDR RAM

Synchronous Dynamic (SD) RAM (Dynamiczna pamięć synchroniczna RAM) jest udoskonaloną pamięcią typu DRAM. Dostęp do pamięci DRAM jest asynchroniczny. Pamięć DRAM natychmiast reaguje na sygnały odczytu. SDRAM jest pamięcią synchroniczną, która odpowiada na sygnały odczytu lub zapisu dopiero po pojawieniu się odpowiedniego sygnału taktującego (zegarowego). Sygnał zegarowy steruje wewnętrznym układem kolejującym żądania odczytu lub zapisu danych.

Double data rate (DDR) SDRAM jest nowszym opracowaniem pamięci synchronicznych SDRAM. Pamięci te wykorzystywane są w komputerach PC od roku 2000. W pamięciach tych również wykorzystywany jest sygnał zegarowy. O ile jednak zwykle pamięci SDRAM reagują na opadające zbocze sygnału zegarowego o tyle pamięci DDR SDRAM wykorzystują oba zbocza, narastające jak i opadające.

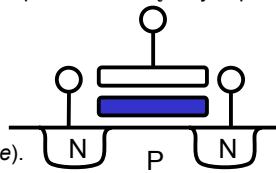
Pamięci **Direct Rambus DRAM (DRDRAM)**, czasami określane również jako **RDRAM**, działają w podobny sposób jak DDR SDRAM, wykorzystują jednak specyficzną metodę sterowania opracowaną przez firmę Rambus, umożliwiającą zastosowanie sygnałów zegarowych o większej częstotliwości.

Architektura systemu komputerowego

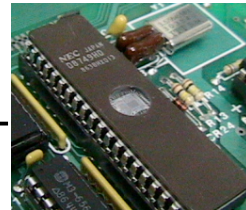
PROM, EPROM, EEPROM

PROMs (Programmable Read-Only Memory) są pamięciami ROM, które mogą być programowane jednorazowo za pomocą specjalnego urządzenia, programatora. Proces programowania polega na trwałym przepaleniu wewnętrznych połączeń wewnątrz układu scalonego pamięci.

EPROMs (Erasable Programmable Read-Only Memory) są pamięciami, w których jako komórkę pamięci wykorzystuje się tranzystor MESHFET z tzw. bramką pływającą (*floating gate*). Poziom logiczny zapisywany jest za pomocą ładunku zgromadzonego na bramce pływającej. Pamięć może być kasowana promieniowaniem ultrafioletowym, a następnie ponownie programowana.



MESHFET z pływającą bramką

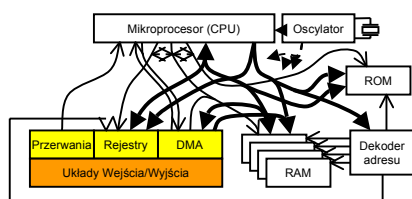


EAROMs (Electrically Alterable Read-Only Memory), w pamięci tej elektrycznie mogą być modyfikowane poszczególne bity zapisanej w nich informacji. Jednak proces modyfikowania danych powinien być przeprowadzany stosunkowo rzadko.

EEPROM np. Flash memory (Electrically Erasable Read-Only Memory) to pamięć, w której zapis danych przeprowadzany jest podobnie jak w EPROM. Cała pamięć (lub jej bloki) może zostać skasowana elektrycznie.

Architektura systemu komputerowego

Układy wejścia/wyjścia



Układy wejścia/wyjścia (I/O devices) to te układy systemu komputerowego, które umożliwiają komunikację systemu z człowiekiem (np. za pomocą monitora, klawiatury czy myszy) lub innym systemem komputerowym (np. za pomocą karty sieciowej czy modemu).

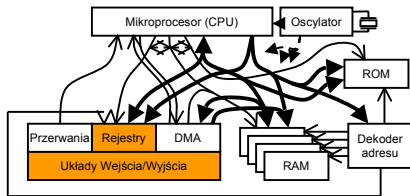
W architekturze systemu komputerowego mikroprocesor oraz pamięć traktowane są jako serce komputera. Pozostałe układy systemu komputerowego traktowane są jako urządzenie wejścia/wyjścia.

Metody komunikacji z układami wejścia/wyjścia:

- Rejestry urządzeń wejścia/wyjścia
 - *Memory-mapped I/O (MMIO)* - rejestry w przestrzeni adresowej pamięci
 - *Port-mapped I/O (PMIO)* - rejestry w specjalnej przestrzeni tzw. portów
- *Direct memory access (DMA)* - bezpośredni dostęp do pamięci
- *Interrupts* - system przerwania

Architektura systemu komputerowego

Układy wejścia/wyjścia: Rejestry

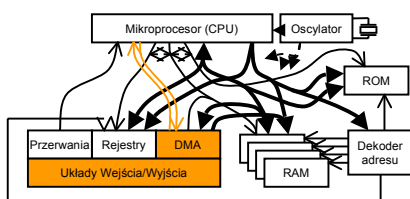


MMIO - część przestrzeni adresowej pamięci zostaje wydzielona na potrzeby rejestrów urządzeń we/wy. Do zapisu/odczytu danych do/z rejestrów wykorzystywane są te same szyny i te same rozkazy procesora co w przypadku pamięci.

PMIO - przestrzeń adresowa rejestrów urządzeń we/wy (portów) i pamięci są rozdzielone. Oddzielne są sygnały zapisu/odczytu danych (szyna sterująca) w przypadku pamięci i rejestrów. Czasami rozdzielone są również szyny adresowa i danych. Inne są też rozkazy procesora dotyczące przesyłania danych między procesorem a pamięcią i inne dla przesyłania danych między procesorem a rejestrami urządzeń we/wy.

Architektura systemu komputerowego

Układy wejścia/wyjścia: DMA



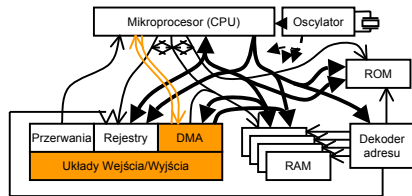
Direct memory access (DMA) - bezpośredni dostęp do pamięci to technika przesyłania danych w systemie komputerowym, w której pewne urządzenia wejścia/wyjścia zapisują dane do pamięci lub odczytują dane z pamięci bez pośrednictwa mikroprocesora.

Sygnały sterujące DMA:

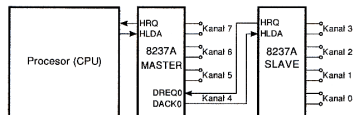
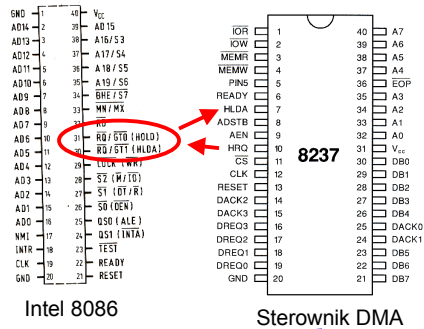
- żądanie dostępu do pamięci
- zewolenie na dostęp do pamięci

Architektura systemu komputerowego

Układy wejścia/wyjścia: DMA



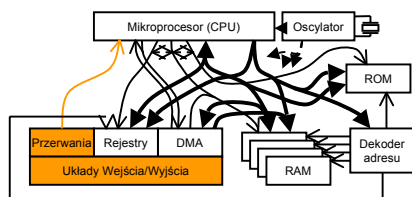
Przykład: DMA w IBM PC



Sterownik DMA w konfiguracji kaskadowej (Master-Slave)

Architektura systemu komputerowego

Układy wejścia/wyjścia: Przerwania

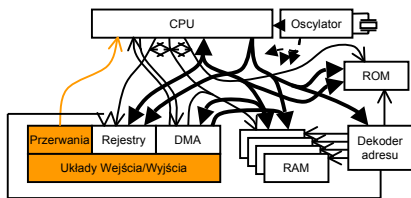


Przerwania: systemy komputerowe zazwyczaj pozwalają na niezwłoczne wykonanie określonych fragmentów programu w wyniku zaistnienia zdarzenia związanego z działaniem układów wejścia/wyjścia. Zdarzenie takie sygnalizowane jest mikroprocesorowi przez urządzenie we./wy. poprzez sygnał żądania przerwania (**interrupt requests IRQ**). Procesor zawieszając wówczas wykonywanie aktualnie realizowanego programu i wykonuje inny fragment programu, tzw. procedurę obsługi przerwania.

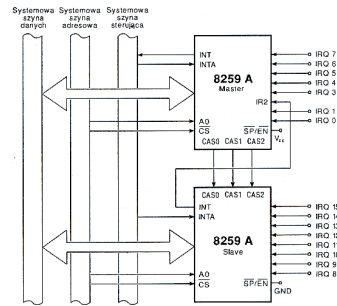
Procesory mają zazwyczaj programowy mechanizm umożliwiający zablokowanie reagowania na żądanie przerwania (np. poprzez odpowiednie wpisy do rejestrów urządzenia generującego przerwania lub rejestrów wewnętrznych samego procesora, poprzez które blokowane są przerwania). Przerwania, które mogą być blokowane nazywa się **maskowalnymi**. Przerwania, których nie można wyłączyć programowo nazywają się **niemaskowalnymi**.

Architektura systemu komputerowego

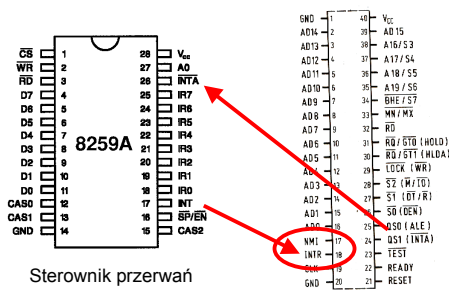
Układy wejścia/wyjścia: Przerwania



Przykład: System przerwań w IBM PC



Sterownik przerwań w konfiguracji kaskadowej (Master-Slave)



Sterownik przerwań

Intel 8086

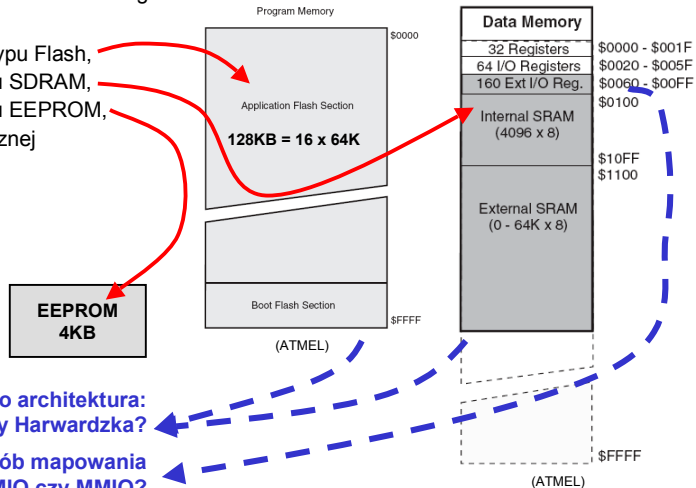
Pamięci ATmega 128

ATMega 128

Pamięci

Wykład prowadzony jest z wykorzystaniem dokumentacji mikrokontrolera ATMega 128 firmy ATMEL®.

1. Pamięć Programu typu Flash,
2. Pamięć danych typu SDRAM,
3. Pamięć danych typu EEPROM,
4. Dołączanie zewnętrznej pamięci danych.



Jaka to architektura: von Neumanna czy Harvardzka?

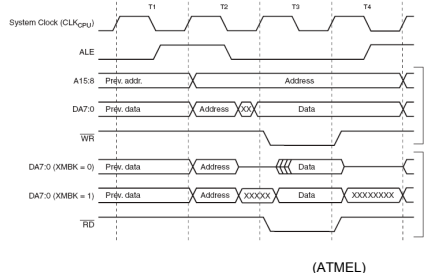
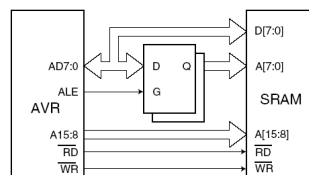
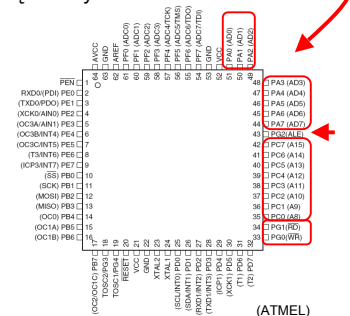
Jaki jest sposób mapowania rejestrów we/wy: PMIO czy MMIO?

ATMega 128

Pamięci

Wykład prowadzony jest z wykorzystaniem dokumentacji mikrokontrolera ATMega 128 firmy ATMEL®.

1. Pamięć Programu typu Flash,
2. Pamięć danych typu SDRAM,
3. Pamięć danych typu EEPROM,
4. Dołączanie zewnętrznej pamięci danych.



System przerwań ATmega 128

ATmega 128 System przerwań

Wykład prowadzony jest z wykorzystaniem dokumentacji mikrokontrolera ATmega 128 firmy ATMEL®.

1. Tablica wektorów przerwań,
2. Położenie tablicy,
3. Programownie obsługi przerwań.

Tablica 23. Reset and Interrupt Vectors

Vector No.	Program Address??	Source	Interrupt Definition
1	\$0000??	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	INT3	External Interrupt Request 3
6	\$000A	INT4	External Interrupt Request 4
7	\$000C	INT5	External Interrupt Request 5
8	\$000E	INT6	External Interrupt Request 6
9	\$0010	INT7	External Interrupt Request 7
10	\$0012	TIMER2 COMP	TimerCounter2 Compare Match
11	\$0014	TIMER2 OVF	TimerCounter2 Overflow
12	\$0016	TIMER1 CAPT	TimerCounter1 Capture Event
13	\$0018	TIMER1 COMP A	TimerCounter1 Compare Match A
14	\$001A	TIMER1 COMP B	TimerCounter1 Compare Match B
15	\$001C	TIMER1 OVF	TimerCounter1 Overflow
16	\$001E	TIMER0 COMP	TimerCounter0 Compare Match
17	\$0020	TIMER0 OVF	TimerCounter0 Overflow
18	\$0022	SPI STC	SPI Serial Transfer Complete
19	\$0024	USART0, RX	USART0, Rx Complete
20	\$0028	USART0, UDRE	USART0 Data Register Empty
21	\$002B	USART0, TX	USART0, Tx Complete
22	\$002A	ADC	ADC Conversion Complete
23	\$002C	EE READY	EEPROM Ready
24	\$002E	ANALOG COMP	Analog Comparator
25	\$0030??	TIMER0 COMP B	TimerCounter0 Compare Match B

(ATMEL)

Zmienne globalne

Procedura przerwania

Konfigurowanie urządzenia we./wy. wysyłającego żądanie przerwania

```
//2005 (C) Piotr M. Szczypiński
//Interrupt handling and PWM example
//Microprocessor Systems Lab

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

//Variables store the current Pulse Width (PW)
int pwm_PB5 = 0;
int pwm_PB6 = 0;

SIGNAL (SIG_OVERFLOW1)
{
    //Modifying the PW for PB5 (Red/Green Led)
    pwm_PB5++;
    if(pwm_PB5>=1024) pwm_PB5 = -1023;
    OCR1A = (pwm_PB5<0 ? -pwm_PB5 : pwm_PB5);

    //Modifying the PW for PB6 (Display illumination)
    pwm_PB6++;
    if(pwm_PB6>=16384) pwm_PB6 = -16383;
    OCR1B = (pwm_PB6<0 ? (-pwm_PB6)/16 : pwm_PB6/16);
}

int main(void)
{
    TCCR1A = 128|32|2|1; // 10.bit PWM
    TCCR1B = 1; // Prescaler 1-5
    OCR1A = pwm_PB5;
    OCR1B = pwm_PB6/16;
    DDRB = 64|32; //PB6, PB5 outputs
    TIMSK = 4; //Interrupt mask
    sei(); //Enables interrupts
    while(1); //Infinite loop
    return 0;
}
```

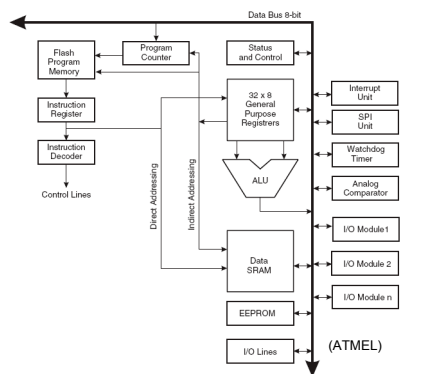
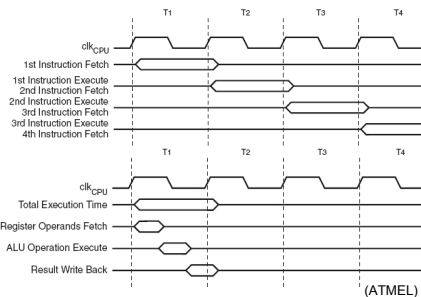
Jednostka centralna ATmega 128

ATmega 128

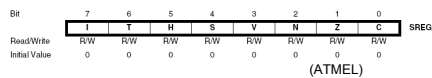
Jednostka centralna - procesor

Wykład prowadzony jest z wykorzystaniem dokumentacji mikrokontrolera ATmega 128 firmy ATMEL®.

1. Jednostka arytmetyczno-logiczna,
2. Rejestry ogólnego przeznaczenia,
3. Rejestry indeksujące,
4. Rejestr (Flagowy) Statusowy,
5. Stos i wskaźnik stosu,



The AVR status Register – SREG – is defined as:



Procesory

Procesory

RISC, CISC, VLIW...

Complex Instruction Set Computer (CISC) - komputer o złożonej liście instrukcji to taki, w którym każda z instrukcji wymaga od procesora wykonania sekwencji kilku operacji niższego poziomu, np. odczytu danej z pamięci, operacji arytmetycznej, zapisu danej wynikowej. Cechami CISC są:

- Czas wykonania instrukcji jest różny i zależy od jej złożoności,
- Kody instrukcji mają różną długość w bitach,
- Program składa się ze stosunkowo niewielkiej liczby instrukcji.

Reduced Instruction Set Computer (RISC) - komputer o ograniczonej liście instrukcji to taki, w którym lista instrukcji procesora jest stosunkowo niewielka. W większości instrukcje powodują wykonanie stosunkowo prostej operacji. Procesor jest tak zaprojektowany by każda instrukcja została wykonana w jak najkrótszym czasie. Cechami RISC są:

- Większość instrukcji wykonywana jest w jednym cyklu zegarowym,
- Kody większości instrukcji mają równą długość w bitach,
- Program składa się zazwyczaj ze znacznej liczby instrukcji.

Procesory

RISC, CISC, VLIW...

Very Long Instruction Word (VLIW) - bardzo długie słowo instrukcji to określenie charakteryzujące mikroprocesory, których pojedyncze instrukcje wymagają wykonania kilku prostych operacji, przy czym operacje te wykonywane są równolegle (w tym samym czasie). is a design of a microprocessor that packs many simple RISC-like instructions into a much longer internal instruction word format.

Explicitly Parallel Instruction Computing (EPIC) to określenie wprowadzone przez firmę Intel równoważne VLIW.

Procesory

Pipeline - potokowość

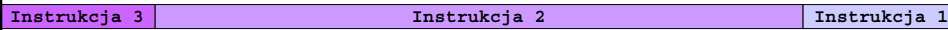
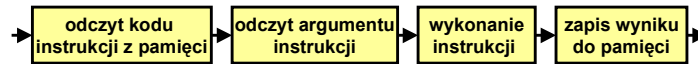
Instruction pipeline - przetwarzanie potokowe instrukcji to technologia wykorzystywana do zwiększenia efektywności działania mikroprocesora. Przetwarzanie potokowe zwiększa szybkość wykonania programu choć wprowadza niewielkie opóźnienia w wykonywaniu poszczególnych instrukcji.

Wykonanie pojedynczej instrukcji wymaga zazwyczaj kilku kroków obsługiwanych często przez niezależne moduły mikroprocesora. System bez przetwarzania potokowego wykonuje jedną instrukcję w czasie i dopiero po jej kompletnym wykonaniu przechodzi do wykonywania następnej. W systemie z przetwarzaniem potokowym wykonywanie następnej instrukcji rozpoczyna się w czasie wykonywania kolejnych kroków instrukcji poprzedniej.

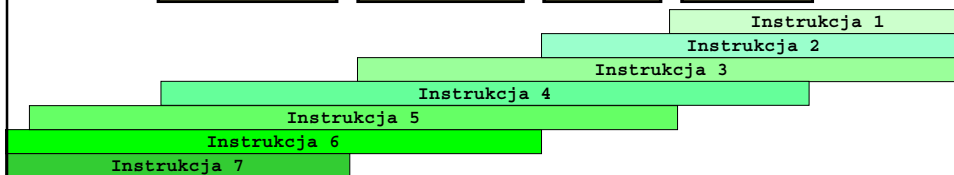
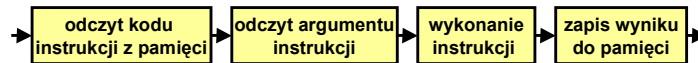
Procesory

Pipeline - potokowość

Bez przetwarzania potokowego



Z przetwarzaniem potokowym



Procesory

Skalarny i superskalarny

Określenie **skalarny** oznacza procesor, który wykonuje jedną instrukcję w pojedynczym cyklu zegarowym w przeciwieństwie do systemów, które do wykonania instrukcji wymagają kilku takich cykli.

Procesor superskalarny to taki, w którym zaprojektowano mechanizm wykonywania kilku instrukcji równoległe w jednym cyklu zegarowym.

Procesory

SIMD

Single Instruction, Multiple Data (SIMD) - jedna instrukcja, wiele danych - to określenie jest związane z systemami komputerowymi, w których pojedyncze instrukcje powodują przetworzenie zbioru różnych danych (wektora danych lub macierzy danych) w sposób równoległy (w tym samym czasie). Mikroprocesor, który umożliwia takie przetwarzanie danych nazywany jest zazwyczaj wektorowym lub macierzowym (*vector processor, array processor*).

Przykład: technologia SIMD w Intel Pentium II i III:

- Multi-Media eXtensions (MMX) to dodatkowy zbiór instrukcji umożliwiających równoległe przetwarzanie danych całkowitych, szczególnie przydatne w zastosowaniach multimedialnych.
- Streaming SIMD Extensions (SSE) to zbiór instrukcji umożliwiających równoległe przetwarzanie danych zmiennoprzecinkowych pojedynczej precyzji.

Programowanie w assemblerze

Asembler

Opkod i Mnemonik

System kodów (binarnych) bezpośrednio zrozumiałym dla mikroprocesora, wykonywanym przez procesor, jest nazywany **kodem maszynowym** mikroprocesora.

Opkod (*opcode = operation code*) to część kodu maszynowego instrukcji określająca jaka operacja ma być wykonana przez mikroprocesor.

Język asemblera (*assembly*) to symboliczny zapis odpowiadający kodowi maszynowemu, który jest bardziej zrozumiały dla człowieka. Kod maszynowy (kodu binarne zero-jedynkowe) staje się bardziej zrozumiały po zamianie na nazwy literowe nazywane mnemonikami.

Mnemonik (*mnemonic*) to kod literowy składający się zazwyczaj z 2 do 5 liter, który reprezentuje opkod. Za mnemonikiem znajduje się zazwyczaj odpowiednia lista argumentów, operandów instrukcji.

Asembler

Opkod i Mnemonik

AVR ATmega 128

Intel Pentium Processor

Disassembler

Address	Opcode	Mnemonic	Argument(s)
14	8189	LDD	R24, Y+1
+00000072:	3085	CPI	R24, 0x05
+00000073:	F409	BRNE	+0x01
+00000074:	C008	RJMP	+0x0008
17:	E380	LDI	R24, 0x30
+00000076:	E795	LDI	R25, 0x75
+00000077:	940E0090	CALL	0x00000090
18:		counter++:	
+00000078:	8189	LDD	R24, Y+1
+00000079:	5F8F	SUBI	R24, 0xFF
+0000007A:	8389	STD	Y+1, R24
+0000007B:	CFF4	RJMP	-0x000C

(AVRStudio disassembler window)

CISC czy RISC?

(Borland C++ Builder CPU window)

Asembler

Język i Kompilator

Asembler w języku polskim używany jest w dwóch znaczeniach, jako język programowania niskiego poziomu (*ang. assembly*) oraz jako program tłumaczący ten język na kod maszynowy - kompilator tego języka (*ang. assembler*).

Każdy rodzaj mikroprocesora ma swój własny, specyficzny kod maszynowy. Różnią się one rodzajem instrukcji, operacji jakie mogą być wykonane, rejestrami, typami danych, itp. Stąd, również języki asemblera są specyficzne dla różnych typów mikroprocesorów.

Tłumaczenie asemblera języka wykonywane jest przez asembler kompilator. Przy czym instrukcja kodu maszynowego jednoznacznie odpowiada jednej instrukcji asemblera (mnemonikowi). Dlatego w przypadku asemblera możliwe jest tłumaczenie w drugą stronę, kodu maszynowego na język asemblera. Proces ten nazywany jest disasemblacją.

Kompilator asemblera jednego typu mikroprocesora uruchamiany w systemie komputerowym z innym typem mikroprocesora nazywany jest krosassemblerem (*crossassembler*).

Asembler

Grupy instrukcji

- Działania arytmetyczne i logiczne
- Rozgałęziania programu (skoki)
- Przesyłanie danych
- Działania na bitach
- Instrukcje sterujące

(ATMEL)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADW	Rd,K	Add Immediate to Word	Rd ← Rd + Rd + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBW	Rd,K	Subtract Immediate from Word	Rd ← Rd - Rd - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd & Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd & K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← ~Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← -Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd & (~K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd & Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← 0	Z,N,V	1
SR	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2

Asembler

Grupy instrukcji

- Działania arytmetyczne i logiczne
- Rozgałęzienia programu (skoki)
- Przesyłanie danych
- Działania na bitach
- Instrukcje sterujące

(ATMEL)

BRANCH INSTRUCTIONS						
RJMP	k	Relative Jump		PC ← PC + k + 1	None	2
JMP	k	Direct Jump		PC ← Z	None	2
JMP	k	Direct Jump		PC ← k	None	3
RCALL	k	Relative Subroutine Call		PC ← PC + k + 1	None	3
ICALL	k	Indirect Call to Z		PC ← Z	None	3
CALL	k	Direct Subroutine Call		PC ← k	None	4
RET		Subroutine Return		PC ← STACK	None	4
RETI		Interrupt Return		PC ← STACK	I	4
CPSE	Rd, Rr	Compare, Skip if Equal		If (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd, Rr	Compare		Rd ← Rr	Z, N, V, C, H	1
CPC	Rd, Rr	Compare with Carry		Rd ← Rr - C	Z, N, V, C, H	1
CPI	Rd, K	Compare Register with Immediate		Rd ← K	Z, N, V, C, H	1
SBRC	Rr, b	Skip if Bit in Register Cleared		If (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set		If (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBC	P, b	Skip if Bit in I/O Register Cleared		If (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set		If (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set		If (SREG(s) = 1) then PC ← PC + k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared		If (SREG(s) = 0) then PC ← PC + k + 1	None	1 / 2
BREQ	k	Branch if Equal		If (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal		If (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set		If (C = 1) then PC ← PC + k + 1	None	1 / 2
BROC	k	Branch if Carry Cleared		If (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher		If (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower		If (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus		If (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus		If (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed		If (N & V = 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed		If (N & V = 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set		If (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared		If (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set		If (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared		If (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set		If (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared		If (V = 0) then PC ← PC + k + 1	None	1 / 2
BRIF	k	Branch if Interrupt Enabled		If (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled		If (I = 0) then PC ← PC + k + 1	None	1 / 2

Asembler

Grupy instrukcji

- Działania arytmetyczne i logiczne
- Rozgałęzienia programu (skoki)
- Przesyłanie danych
- Działania na bitach
- Instrukcje sterujące

(ATMEL)

DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers			Rd ← Rr
MOVW	Rd, Rr	Copy Register Word			Rd+1:Rd ← Rr+1:Rr
LDI	Rd, K	Load Immediate			Rd ← K
LD	Rd, X	Load Indirect			Rd ← (X)
LD	Rd, X+	Load Indirect and Post-Inc.			Rd ← (X), X ← X + 1
LD	Rd, -X	Load Indirect and Pre-Dec.			X ← X - 1, Rd ← (X)
LD	Rd, Y	Load Indirect			Rd ← (Y)
LD	Rd, Y+	Load Indirect and Post-Inc.			Rd ← (Y), Y ← Y + 1
LD	Rd, -Y	Load Indirect and Pre-Dec.			Y ← Y - 1, Rd ← (Y)
LDD	Rd, Y+q	Load Indirect with Displacement			Rd ← (Y + q)
LD	Rd, Z	Load Indirect			Rd ← (Z)
LD	Rd, Z+	Load Indirect and Post-Inc.			Rd ← (Z), Z ← Z + 1
LD	Rd, -Z	Load Indirect and Pre-Dec.			Z ← Z - 1, Rd ← (Z)
LDD	Rd, Z+q	Load Indirect with Displacement			Rd ← (Z + q)
LDS	Rd, k	Load Direct from SRAM			Rd ← (k)
ST	X, Rr	Store Indirect			(X) ← Rr
ST	X+, Rr	Store Indirect and Post-Inc.			(X) ← Rr, X ← X + 1
ST	-X, Rr	Store Indirect and Pre-Dec.			X ← X - 1, (X) ← Rr
ST	Y, Rr	Store Indirect			(Y) ← Rr
ST	Y+, Rr	Store Indirect and Post-Inc.			(Y) ← Rr, Y ← Y + 1
ST	-Y, Rr	Store Indirect and Pre-Dec.			Y ← Y - 1, (Y) ← Rr
STD	Z+q, Rr	Store Indirect with Displacement			(Z + q) ← Rr
ST	Z, Rr	Store Indirect			(Z) ← Rr
ST	Z+, Rr	Store Indirect and Post-Inc.			(Z) ← Rr, Z ← Z + 1
ST	-Z, Rr	Store Indirect and Pre-Dec.			Z ← Z - 1, (Z) ← Rr
STD	Z+q, Rr	Store Indirect with Displacement			(Z + q) ← Rr
STS	k, Rr	Store Direct to SRAM			(k) ← Rr
LPM		Load Program Memory			R0 ← (Z)
LPM	Rd, Z	Load Program Memory			Rd ← (Z)
LPM	Rd, Z+	Load Program Memory and Post-Inc.			Rd ← (Z), Z ← Z + 1
ELPM		Extended Load Program Memory			R0 ← (RAMPZ:Z)
ELPM	Rd, Z	Extended Load Program Memory			Rd ← (RAMPZ:Z)
ELPM	Rd, Z+	Extended Load Program Memory and Post-Inc.			Rd ← (RAMPZ:Z), RAMPZZ ← RAMPZZ + 1
SFM		Store Program Memory			(Z) ← R1:R0
IN	Rd, P	In Port			Rd ← P
OUT	P, Rr	Out Port			P ← Rr
PUSH	Rr	Push Register on Stack			STACK ← Rr
POP	Rd	Pop Register from Stack			Rd ← STACK

Asembler

Grupy instrukcji

- Działania arytmetyczne i logiczne
- Rozgałęziania programu (skoki)
- Przesyłanie danych
- Działania na bitach
- Instrukcje sterujące

(ATMEL)

BIT AND BIT-TEST INSTRUCTIONS					
SBI	P, b	Set Bit in I/O Register	$I/O(P,b) \leftarrow 1$	None	2
CBI	P, b	Clear Bit in I/O Register	$I/O(P,b) \leftarrow 0$	None	2
LSL	Rd	Logical Shift Left	$Rd(n+1) \leftarrow Rd(n), Rd(0) \leftarrow 0$	Z, C, N, V	1
LSR	Rd	Logical Shift Right	$Rd(n) \leftarrow Rd(n+1), Rd(7) \leftarrow 0$	Z, C, N, V	1
ROL	Rd	Rotate Left Through Carry	$Rd(0) \leftarrow C, Rd(n+1) \leftarrow Rd(0), C \leftarrow Rd(7)$	Z, C, N, V	1
ROR	Rd	Rotate Right Through Carry	$Rd(7) \leftarrow C, Rd(n) \leftarrow Rd(n+1), C \leftarrow Rd(0)$	Z, C, N, V	1
ASR	Rd	Arithmetic Shift Right	$Rd(n) \leftarrow Rd(n+1), n=0..6$	Z, C, N, V	1
SWAP	Rd	Swap Nibbles	$Rd(3..0) \leftrightarrow Rd(7..4), Rd(7..4) \leftrightarrow Rd(3..0)$	None	1
SSET	s	Flag Set	$SREG(s) \leftarrow 1$	SREG(s)	1
BCLR	s	Flag Clear	$SREG(s) \leftarrow 0$	SREG(s)	1
BST	Rv, b	Bit Store from Register to T	$T \leftarrow Rv(b)$	T	1
BLD	Rd, b	Bit load from T to Register	$Rv(b) \leftarrow T$	None	1
SEC		Set Carry	$C \leftarrow 1$	C	1
CLC		Clear Carry	$C \leftarrow 0$	C	1
SEN		Set Negative Flag	$N \leftarrow 1$	N	1
CLN		Clear Negative Flag	$N \leftarrow 0$	N	1
SEZ		Set Zero Flag	$Z \leftarrow 1$	Z	1
CLZ		Clear Zero Flag	$Z \leftarrow 0$	Z	1
SEI		Global Interrupt Enable	$I \leftarrow 1$	I	1
CLI		Global Interrupt Disable	$I \leftarrow 0$	I	1
SES		Set Signed Test Flag	$S \leftarrow 1$	S	1
CLS		Clear Signed Test Flag	$S \leftarrow 0$	S	1
SEV		Set Two's Complement Overflow	$V \leftarrow 1$	V	1
CLV		Clear Two's Complement Overflow	$V \leftarrow 0$	V	1
SET		Set T in SREG	$T \leftarrow 1$	T	1
CLT		Clear T in SREG	$T \leftarrow 0$	T	1
SEH		Set Half Carry Flag in SREG	$H \leftarrow 1$	H	1
CLH		Clear Half Carry Flag in SREG	$H \leftarrow 0$	H	1

Asembler

Grupy instrukcji

- Działania arytmetyczne i logiczne
- Rozgałęziania programu (skoki)
- Przesyłanie danych
- Działania na bitach
- Instrukcje sterujące

(ATMEL)

MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

Asembler

Tryby adresowania

Tryby adresowania to określenie związane ze sposobem pobierania i zapisywania danych (operandów) niezbędnych do wykonania instrukcji kodu maszynowego. Tryby adresowania określają sposób obliczania adresu danej w pamięci lub przestrzeni rejestrów urządzeń wejścia/wyjścia lub określenia rejestru wewnętrznego mikroprocesora, który tę daną zawiera.

Przykłady trybów adresowania:

Natychmiastowy - wartość operandu podana w kodzie maszynowym instrukcji

Bezpośredni - w kodzie podany jest adres operandu w pamięci

Pośredni - adres operandu znajduje się w pamięci pod podanym adresem

Rejestrowy - operand znajduje się w rejestrze wewnętrznym procesora

Rejestrowy pośredni - adres operandu znajduje się w rejestrze procesora

Indeksowy bezpośredni - adres operandu jest sumą zawartości rejestru i wartości podanej w kodzie

Autoinkrementowany - jak wyżej z tym, że zawartość tego rejestru jest zwiększona o jeden po wykonaniu instrukcji

Bazowy - adres danej jest sumą zawartości rejestru i wartości podanej w kodzie

Absolutny - w kodzie podany jest adres w pamięci programu

Względny - w kodzie podane jest przesunięcie względem adresu instrukcji

[Jakie są tryby adresowania ATMEGA 128?](#)

Asembler

Format linii kodu w asemblerze ATMEGA 128

Linia kodu źródłowego asemblera:

```
[etykieta:] dyrektywa [operand(y)] [Komentarz]
```

```
[etykieta:] mnemonik [operand(y)] [Komentarz]
```

```
Komentarz
```

```
Pusta linia
```

```
etykieta:
```

przy czym **komentarz** to tekst rozpoczynający się od średnika:

```
; [Tekst]
```

(AVRStudio Tools User Guide)

Asembler

Etykieta, stała, zmienna w asemblerze ATmega 128

Etykieta to identyfikator złożony z liter i cyfr rozpoczynający się od litery. Zazwyczaj etykieta określa konkretne miejsce w pamięci czyli adres.

```
procedura:      sts portb, r17
                dec r17
```

Stała w programie definiowana jest za pomocą dyrektywy EQU. Wartości stałych nie mogą być zmieniane w czasie wykonywania programu.

```
.EQU tab_size = 4
.EQU porta = 0x25
```

Zmienna definiowana jest z wykorzystaniem etykiety. Etykieta wskazuje miejsce w pamięci danych a wartość znajdująca się w tym miejscu może być modyfikowana w czasie wykonywania programu.

```
var1:          .BYTE 1
table:         .BYTE tab_size
```

Asembler

Procedury w asemblerze ATmega 128

```
    ; main program
    ...
    rcall procedure_name
    ...
procedure_name:
    ...
    ; procedure body here
    call other_procedure_name
    ...
    ret
other_procedure_name:
    ...
    ; procedure body here
    ...
    ret
```

Co to jest stos? Jak instrukcje `call` i `ret` wykorzystują stos?

Asembler

Makrodefinicja w asemblerze ATmega 128

```
.MACRO SUBI16 ; Start macro definition
    subi @1,low(@0) ; Subtract low byte
    sbci @2,high(@0) ; Subtract high byte
.ENDMACRO ; End macro definition

.CSEG ; Start code segment
SUBI16 0x1234,r16,r17 ; Sub.0x1234 from r17:r16
```

(AVRStudio Tools User Guide)

Asembler

Procedura a makrodefinicja

Procedura:

- wolniej działa
potrzebny jest dodatkowy czas na skoki będące wynikiem instrukcji `call` i `ret`
- mniejszy kod programu
procedura znajduje się tylko w jednym określonym miejscu kodu programu

Makrodefinicja:

- szybciej działa
kod jest wklejony w całości tam gdzie jest potrzebny, brak dodatkowych skoków
- większy kod programu
kod makrodefinicji jest powielany

Asembler

Dyrektywy

Dyrektywy wykorzystywane są do sterowania procesem kompilacji asemblera na kod maszynowy. Dyrektywy nie są tłumaczone przez na kod maszynowy. Są one między innymi wykorzystywane do określania położenia programu w pamięci, definiowania makrodefinicji, inicjalizacji pamięci, itp.

(AVRStudio Tools User Guide)

Dyrektywa Opis

BYTE	Reserve byte to a variable
CSEG	Code Segment
CSEGSIZE	Program memory size
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data Segment
DW	Define Constant word(s)
ENDM	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn Macro expansion in list file on
MACRO	Begin macro
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

Asembler

Wyrażenia

Wyrażenia w asemblerze mogą zawierać operandy, operatory i funkcje. Wyrażenia te wykorzystywane są w procesie kompilacji i nie są tłumaczone na kod maszynowy

(AVRStudio Tools User Guide)

Funkcje

LOW(expression) returns the low byte of an expression
HIGH(expression) returns the second byte of an expression
BYTE2(expression) is the same function as HIGH
BYTE3(expression) returns the third byte of an expression
BYTE4(expression) returns the fourth byte of an expression
LWRD(expression) returns bits 0-15 of an expression
HWRD(expression) returns bits 16-31 of an expression
PAGE(expression) returns bits 16-21 of an expression
EXP2(expression) returns 2 to the power of expression
LOG2(expression) returns the integer part of log2(expression)

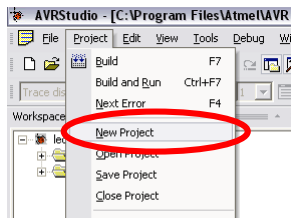
Operatory

Symbol	Opis
!	Logical Not
~	Bitwise Not
-	Unary Minus
*	Multiplication
/	Division
+	Addition
-	Subtraction
<<	Shift left
>>	Shift right
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
!=	Not equal
&	Bitwise And
^	Bitwise Xor
	Bitwise Or
&&	Logical And
	Logical Or

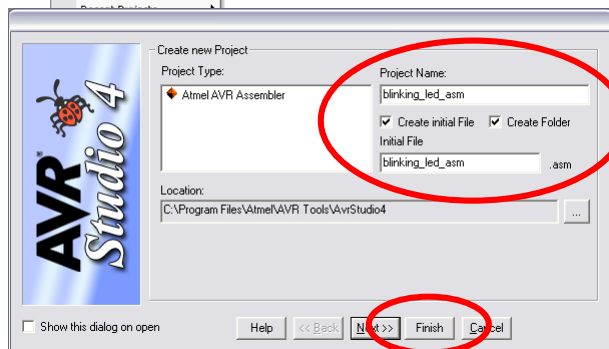
Czy operator '+' jest zamieniany przez asembler na instrukcję add?

Asembler

Projekt asemblerowy w AVR Studio



1. Uruchom AVR Studio
2. Wybierz *Project->NewProject*
3. Zaznacz *Create initial File* oraz *Create Folder*
4. Wypełnij pola *Project Name* i *Initial File*
5. Wciśnij *Finish*
6. Napisz program w asemblerze
7. Skompiluj projekt do kodu maszynowego



Asembler

Projekt asemblerowy w AVR Studio

Przykład:

```

;:2005(C) Piotr M. Szczypinski
;:Microprocessor Systems Lab
;:Assembly programming example
;:Blinking Led in Assembly

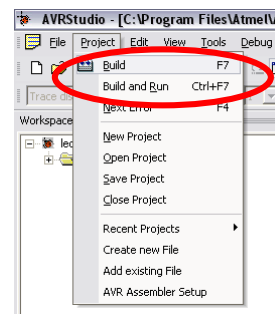
.equ  DDRB  = $37
.equ  PORTB = $38
.equ  SPH   = $3E
.equ  SPL   = $3D
.equ  RAM_TOP = $10FF
.equ  DELAY = $20

;:Everything starts here
START:
;:Stack initialization
LDI R16, HIGH(RAM_TOP)
OUT SPH, R16
LDI R16, LOW(RAM_TOP)
OUT SPL, R16

;:Main program
LDI R18, 16
LDI R17, 0
STS DDRB, R18
LOOP:
STS PORTB, R17
CALL DELAY_PROC
STS PORTB, R18
CALL DELAY_PROC
RJMP LOOP ;:Loops forever

;:Delay procedure
DELAY_PROC:
LDI R20, DELAY
DELAY_LOOP1:
LDI R19, $FF
DELAY_LOOP2:
LDI R16, $FF
DELAY_LOOP3:
NOP
NOP
NOP
DEC R16
BRNE DELAY_LOOP3
DEC R19
BRNE DELAY_LOOP2
DEC R20
BRNE DELAY_LOOP1
RET
    
```

1. Uruchom AVR Studio
2. Wybierz *Project->NewProject*
3. Zaznacz *Create initial File* oraz *Create Folder*
4. Wypełnij pola *Project Name* i *Initial File*
5. Wciśnij *Finish*
6. Napisz program w asemblerze
7. Skompiluj projekt do kodu maszynowego



Asembler

Łączenie asemblera z C w WinAvr

W języku C słowo **asm** umożliwia dołączenie pojedynczych linii w języku asemblera w kodzie źródłowym w języku C.

W kompilatorze GCC robi się to tak:

```
asm("mnemonik [operand(y)]");
```

Przykład:

```
//2005 (C) Piotr M. Szczypinski
//Microprocessor Systems Lab
//assembly in C source code

#include <avr/io.h>
#include <avr/delay.h>
int main (void)
{
  // Equivalent of DDRB = 16;
  asm ("LDI R24, 16"); // Loads 16 into R24 register
  asm ("STS 0x37, R24"); // Sets DDRB (address 0x37) register

  while(1)
  {
    int i;
    PORTB = 16;
    for(i = 0; i < 20; i++) _delay_loop_2(20000);
    PORTB = 0;
    for(i = 0; i < 50; i++) _delay_loop_2(20000);
  }
  return 0;
}
```

Asembler

Łączenie asemblera z C w WinAvr

W projekcie WinAvr w kodach źródłowych języka C można wykorzystywać całe procedury napisane w asemblerze. Kody źródłowe języka C oraz asemblera muszą być umieszczone w osobnych plikach (z rozszerzeniem c i asm).

1. Utwórz nowy plik źródłowy z rozszerzeniem asm i dodaj go do projektu,
2. Dodany plik poddaj edycji, zdefiniuj w nim procedurę w asemblerze, która ma być wywoływana z kodu w języku C,
3. Dodaj utworzony plik do listy plików asemblerowych w pliku Makefile
4. Zadeklaruj odpowiednio napisaną procedurę w kodzie źródłowym języka C,
5. Uzupełnij kod w C o wywołanie procedury asemblerowej

Asembler

Łączenie asemblera z C w WinAver

Przykład:

The screenshot shows three windows in the WinAver IDE:

- blinking_led.c**: A C source file with the following code:

```
//2005 (C) Piotr M. Szczypinski
//Microprocessor Systems Lab
//calling a asseblly procedure example

#include <avr/io.h>
#include <avr/delay.h>
extern void SET_DDRB_16(void);

int main (void)
{
    SET_DDRB_16();
    while(1)
    {
        int i;
        PORTB = 16; //set pin 16
        for(i = 0; i < 10; i++)
            _delay_loop_1(100);
        PORTB = 0; //set pin 16
        for(i = 0; i < 10; i++)
            _delay_loop_1(100);
    }
    return 0; //this n
```
- set_ddrb.asm**: An assembly source file with the following code:

```
.equ DDRB    = $37

.global SET_DDRB_16
SET_DDRB_16:
    LDI R24, 16
    STS DDRB, R24
    RET
```
- Makefile**: A Makefile with the following content:

```
# List C source files here. (C dependencies are automatically generated.)
SRC = $(TARGET).c

# List Assembler source files here.
# Make them always end in a capital .S. Files ending in a lowercase .s
# will not be considered source files but generated files (assembler
# output from the compiler), and will be deleted upon "make clean"!
# Even though the DOS/win* filesystem matches both .s and .S the same,
# it will preserve the spelling of the filenames, and gcc itself does
# care about how the name is spelled on its command-line.
ASRC = set_ddrb.asm
```

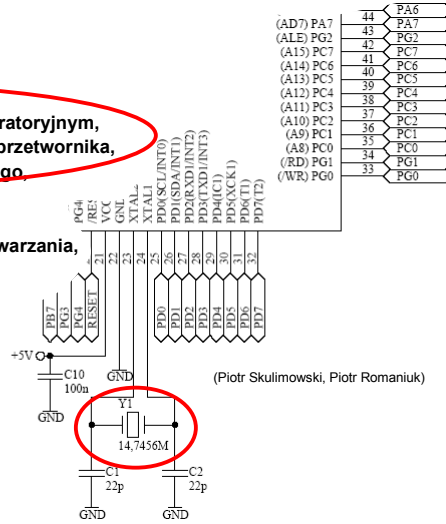
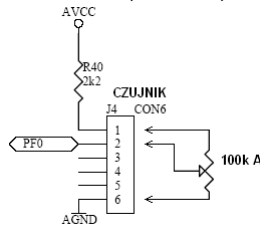
Red arrows with numbers 1 through 5 point to specific elements: 1 points to the assembly file window, 2 points to the assembly code, 3 points to the Makefile window, 4 points to the `extern void SET_DDRB_16(void);` line in the C file, and 5 points to the `SET_DDRB_16();` call in the C file.

Przetwornik
Analogowo-Cyfrowy
ATMega 128

ATMega 128 ADC

Wykład prowadzony jest z wykorzystaniem dokumentacji mikrokontrolera ATMega 128 firmy ATMEL®.

1. Połączenie wejścia analogowego w zestawie laboratoryjnym,
2. Częstotliwość zegara układu ma wpływ na pracę przetwornika,
3. **Charakterystyki przetwornika analogowo-cyfrowego,**
4. Kanaly analogowe i wybór wzmocnienia,
5. Wybór napięcia odniesienia (referencyjnego),
6. Tryby pojedynczej konwersji i swobodnego przetwarzania,
7. Odczyt wyniku działania przetwornika,
8. Rejestry przetwornika: ADMUX, ADCSRA, ADC

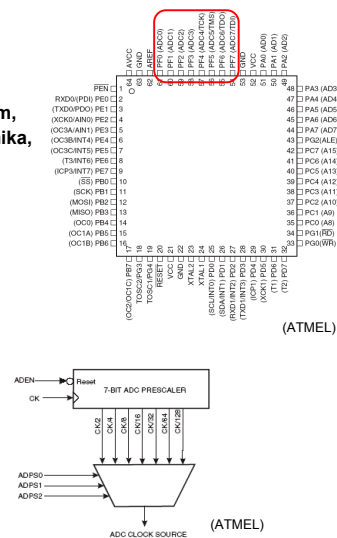


(Piotr Skulimowski, Piotr Romaniuk)

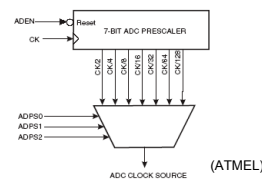
ATMega 128 ADC

Wykład prowadzony jest z wykorzystaniem dokumentacji mikrokontrolera ATMega 128 firmy ATMEL®.

1. Połączenie wejścia analogowego w zestawie laboratoryjnym,
2. Częstotliwość zegara układu ma wpływ na pracę przetwornika,
3. **Charakterystyki przetwornika analogowo-cyfrowego,**
4. **Kanaly analogowe i wybór wzmocnienia,**
5. Wybór napięcia odniesienia (referencyjnego),
6. Tryby pojedynczej konwersji i swobodnego przetwarzania,
7. Odczyt wyniku działania przetwornika,
8. Rejestry przetwornika: ADMUX, ADCSRA, ADC



(ATMEL)



(ATMEL)

ATMega 128

ADC

Wykład prowadzony jest z wykorzystaniem dokumentacji mikrokontrolera ATMega 128 firmy ATMEL®.

1. Połączenie wejścia analogowego w zestawie laboratoryjnym,
2. Częstotliwość zegara układu ma wpływ na pracę przetwornika,
3. Charakterystyki przetwornika analogowo-cyfrowego,
4. Kanaly analogowe i wybór wzmocnienia,
7. Wybór napięcia odniesienia (referencyjnego),
5. Tryby pojedynczej konwersji i swobodnego przetwarzania,
6. Prosta a czas i dokładność przetwarzania,
7. Odczyt wyniku działania przetwornika,
8. Rejestry przetwornika: ADMUX, ADCSRA, ADC^{ADLAR = 0}

ADC Multiplexer Selection Register – ADMUX

Bit	7	6	5	4	3	2	1	0	ADMUX
	REFS	REFS	ADLAR	ADIF	ADSC	ADIF	ADIF	ADIF	
Read/Write	RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value	0	0	0	0	0	0	0	0	

ADC Control and Status Register A – ADCSRA

Bit	7	6	5	4	3	2	1	0	ADCSRA
	ADEN	ADSC	ADIF	ADIF	ADIF	ADIF	ADIF	ADIF	
Read/Write	RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value	0	0	0	0	0	0	0	0	

(ATMEL)

The ADC Data Register – AD_CONVERTER

Bit	15	14	13	12	11	10	9	8	ADCH
	ADCF	ADCF	ADCF	ADCF	ADCF	ADCF	ADCF	ADCF	
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

ADLAR = 1:

Bit	15	14	13	12	11	10	9	8	ADCH
	ADCF	ADCF	ADCF	ADCF	ADCF	ADCF	ADCF	ADCF	
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

(ATMEL)