Programowanie mikrokontrolerów - wykład 10

Wstęp - kilka otwartych pytań

- Czy istnieje uniwersalizm w technice ?
- Czy istnieją uniwersalne języki programowania?
- Czy można oddzielić język programowania od warstwy sprzętowej ?

Paradygmaty programowania

- *Programowanie imperatywne* - sekwencję poleceń zmieniających krok po kroku stan maszyny, aż do uzyskania oczekiwanego wyniku (stan będący funkcją czasu) - związany ściśle z budową sprzętu komputerowego o architekturze von Neumanna (Asemblery, Fortran, Cobol, Pascal, C),
- *Programowanie obiektowe* - program to zbiór porozumiewających się ze sobą obiektów,
- *Programowanie funkcyjne* - składamy i obliczamy funkcje, w sensie podobnym do funkcji znanych z matematyki. Nie ma stanu maszyny — nie ma zmiennych mogących zmieniać wartość. Nie ma zatem „samodzielnie biegnącego” czasu, a jedynie zależności między danymi (Lisp, Scheme, Ocaml),
- *Programowanie w logice* - opisujemy, co wiemy i co chcemy uzyskać (języki funkcyjne i logiczne nazywa się łącznie językami deklaratywnymi).
- inne - np. programowanie współbieżne, Programowanie sterowane zdarzeniami, programowanie strukturalne.

Interpreter i kompilator

- *Interpretr* **-** analizuje kod źródłowy programu, a następnie wykonuje przeanalizowane fragmenty,
- *Kompilator* **-** konwertuje kod źródłowy do *kodu maszynowego*. Kod maszynowy umieszczany jest w pamięci programu i wykonywany.

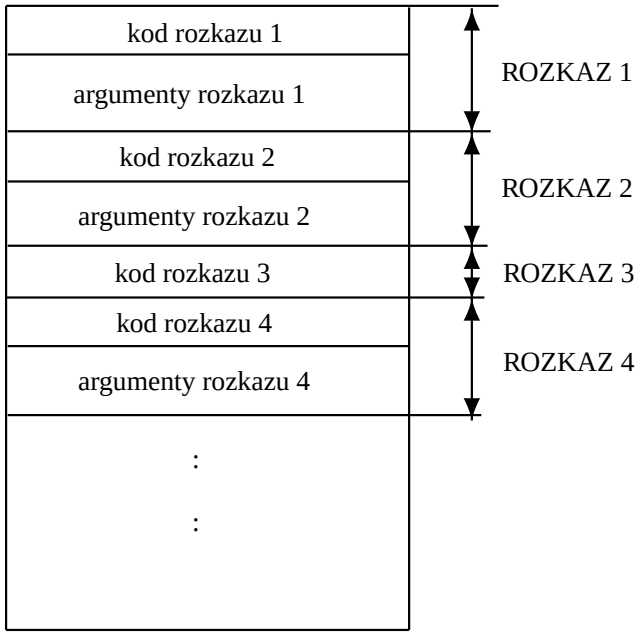
Kompilacja

Kompilacja to proces automatycznego tłumaczenia kodu napisanego w języku programowania na *kod maszynowy*. Dane wejściowe najczęściej nazywa się *kodem źródłowym*.

- Nazwa kompilacja na co dzień jest używana w kontekście tłumaczenia z języka wyższego poziomu na język niższego poziomu.
- Po kompilacji *kod maszynowy* zapisywany jest w plikach o rozszerzeniu *hex* albo *bin*,
- Postać binarne jest nieczytelna trudna do bezpośredniej analizy,
- Kompilacja może być częścią większego "procesu tłumaczenia", tworzony w jej trakcie kod wynikowy (object code) jest przekazywany do innych programów (linkera).

Kod maszynowy

Kod maszynowy - to postać programu komputerowego (wykonywalna, binarna) przeznaczona do bezpośredniego wykonania przez procesor.



- Kod maszynowy jest sekwencją rozkazów,
- Rozkaz składa się z *kodu rozkazu* i *argumentu rozkazu*.
- W architekturze CISC rozkazy mogą być różnych rozmiarów, a ich czas wykonywania może trwać różną liczbę cykli maszynowych.

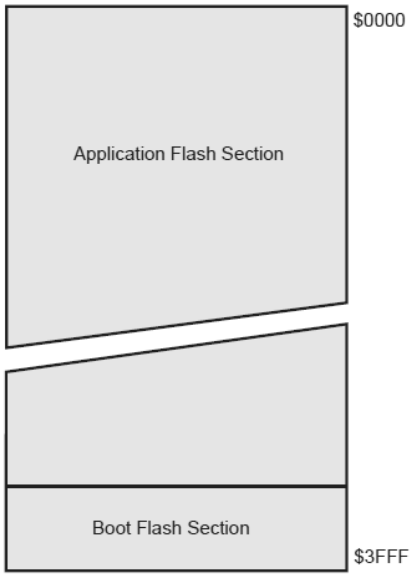
Programowanie - umieszczenie kodu maszynowego w pamięci programu

- *Architektura von Neumanna* - kod maszynowy umieszczany jest w pamięci ROM we wspólnej przestrzeni adresowej z pamięcią RAM,
- *Architektura harwardzka* - w μC o tej architekturze kod maszynowy umieszczany jest w osobnej przestrzeni adresowej niż pamięć RAM.

Metody umieszczenie *kodu maszynowego* **w pamięci programu**

- *High voltage Programming* czyli sposób programowania wprowadzony ponad 15 lat temu do programowania pamięci EPROM za pomocą sygnałów 12V - wymaga programatora,
- *ISP (In-System Programmable)* które nie wymaga wyjmowania pamięci programu z systemu w którym pracuje,
- *Bootloader* - po resecie μC uruchamiany jest program znajdujący się w sekcji Bootloadera, łączy się on z komputerem nadrzędnym, pobiera kod programu i umieszcza go w przeznaczonej do tego obszarze pamięci ROM,
- inne, np. w komputerach z systemem operacyjnym za umieszczenie kodu programu odpowiada *loader*.

Programowanie μC Atmel AtMega32



W przypadku μC AtMega32 możliwe są następujące sposoby programowania.

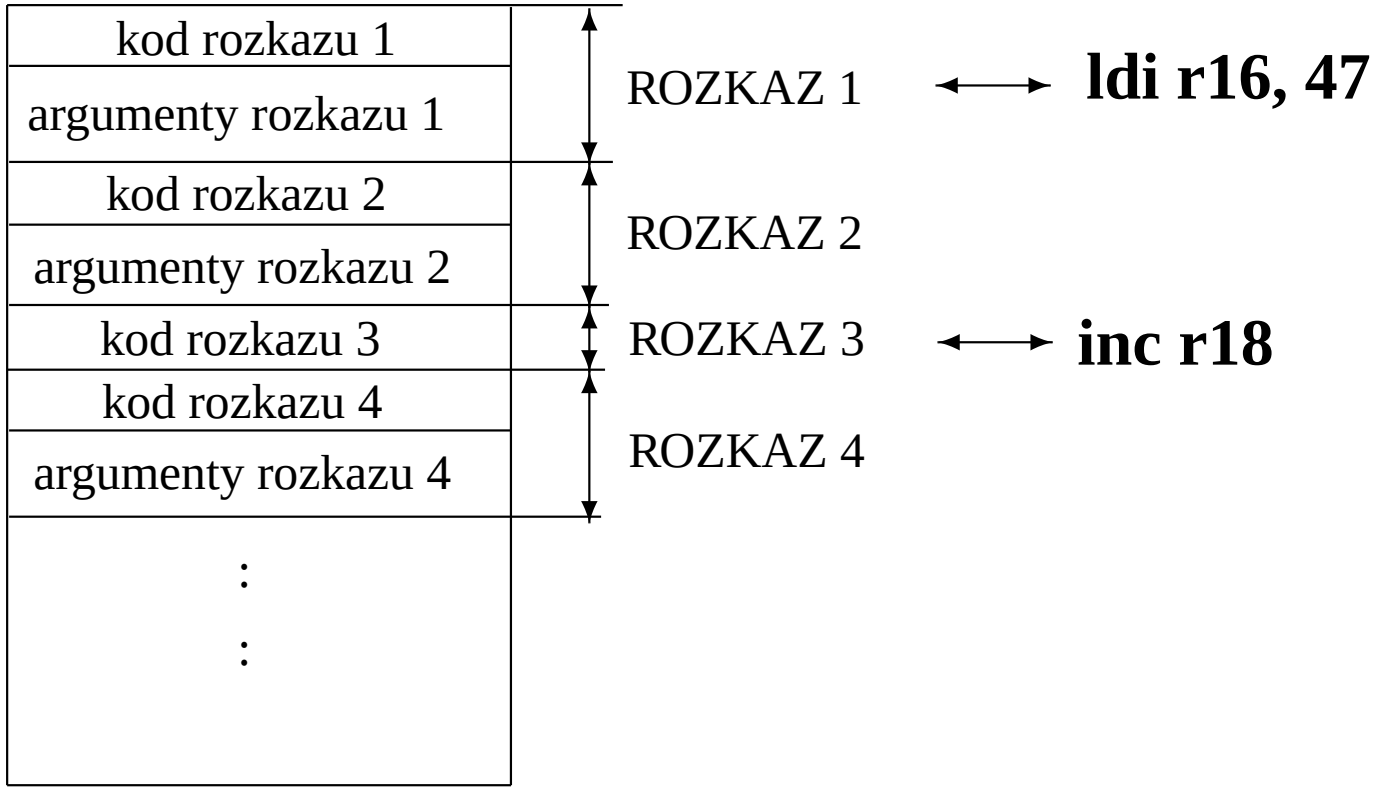
- Z wykorzystaniem *interfejsu ISP/SPI* (ang. In-System Programming/ Serial Peripheral Interface) - poprzez ten interfejs program umieszczany jest w pamięci programu. Wymaga programatora.
- Z wykorzystaniem *protokołu JTAG* (ang. Joint Test Action Group),
- *Bootloader* - po resecie μC uruchamiany jest program znajdujący się w sekcji Bootloadera, który łączy się z komputerem nadrzędnym, pobiera kod programu i umieszcza go w sekcja aplikacji. Nie wymaga programatora.

Programowanie poprzez ISP i JTAG

- In-System Programming lub ISP – umożliwiające zaprogramowanie układu bez demontażu,
- Możliwość połączenia programowania i testowania w jednej fazie produkcyjnej,
- Układy scalone wyposażone w ISP mają wewnętrzne obwody, generujące napięcia, niezbędne do zaprogramowania wbudowanej pamięci, a także interfejs szeregowy, umożliwiający komunikację z programatorem.
- Do komunikacji większość układów wykorzystuje protokół JTAG, choć są w tym celu wykorzystywane także inne protokoły, np. SPI,
- JTAG (ang. Joint Test Action Group) to nazwa standardu IEEE 1149.1 definiującego protokół używany do testowania połączeń na płytkach drukowanych,
- JTAG stosowany jest także do uruchamiania i programowania układów programowalnych i systemów mikroprocesorowych.

Asembler

Asembler jest językiem niskopoziomowym, gdzie każdej instrukcji procesora odpowiada słowne polecenie (mnemonik) wraz z operandami.



- ROZKAZOWI 1 odpowiada mnemonik *ldi r16, 47*
- ROZKAZOWI 3 odpowiada mnemonik *inc r18*

Podstawowe elementy języka Asembler

- Etykieta,
- Dyrektywy,
- Makra,
- Inne.

Etykiety asemblera

Zadaniem etykiet jest symboliczne określenie miejsca w programie.

- Etykiety składają się z nazwy zakończonej dwukropkiem, np.
  etykieta:
  petla:
- Podczas kompilacji symboliczne nazwy zamieniane są konkretnymi wartościami adresów,
- Typowym przykładem wykorzystania etykiety poprzedzającej określoną instrukcję języka jest wykonanie skoku za pomocą instrukcji, która ma postać: *goto etykieta* lub *jmp etykieta*.

Dyrektywy asemblera

Polecenie dla kompilatora, wykonania określonej czynności, lub ustalające sposób kompilacji.

- .*DEF nazwa=Rxx* Przypisanie rejestrowi *Rxx* nazwy *nazwa*.
- .*EQU etykieta=wyr* Dyrektywa przypisuje etykiecie wartość określoną wyrażeniem. Wartość przypisana etykiecie nie może być zmieniona
- .*SET etykieta=wyr* Dyrektywa przypisuje etykiecie wartość określoną wyrażeniem.
- .*ORG wyrażenie* **-** Ustawia licznik lokacji pamięci ROM na wartość określoną wyrażeniem.

Makroasembler, makra

- *Makro* - ciąg instrukcji, nie stanowiący samodzielnego programu, przeznaczony do wielokrotnego wykorzystania w kodzie programu,
- Makra można definiować za pomocą dyrektywy *MACRO* oznacza początek definicji makroinstrukcji. Dyrektywa wymaga argumentu w postaci nazwy makroinstrukcji.
- *Makroasembler* jest to asembler posiadający obsługę makr w czasie prekompilacji. Posiada on wszystkie funkcje właściwe asemblerom dając równocześnie możliwość stosowania makropoleceń.

Stos i rejestr *Wskaźnik stosu*

- Miejsce na stosie (adres stosu) wskazuje wskaźnik stosu,
- Wskaźnik stosu składa się z dwóch rejestrów Sph i Spl zawierających odpowiednio starszy i młodszy bajt adresu stosu.
- Jeśli chcemy korzystać ze stosu musimy go najpierw zainicjalizować - wpisać adres do rejesrtów sp np.:

```
ldi R17, 0x08
ldi R16, 0x5f
Out Sph , R17
Out Spl , R16
```

Licznik rozkazów

Licznik rozkazów (PC) - rejestr procesora zawierający adres aktualnie wykonywanej lub następnej w kolejności instrukcji kodu maszynowego.

- Licznik rozkazu nie może być modyfikowany poprzez bezpośrednie wpisanie wartości
- W przerwaniach, instrukcjach typu *call* zawartość rejestru PC kładziona jest na stos

Tryb adresowania μC AtMega32

- natychmiastowe,
- bezpośrednie,
- pośrednie - z przemieszczeniem, pośrednie z preinkrementacją, pośrednie z postinkremetacją,
- Do adresowań pośrednich wykorzystuje się rejestry od R26 do R31 (X, Y, Z).

Instrukcje asemblera μC AtMega32

- Instrukcje arytmetyczne i logiczne,
- Instrukcje skoków,
- Instrukcje przesyłania danych,
- Instrukcje operacji bitowych.

Dokładny opis w dokumentacji.

Biblioteka wspomagająca assembler Baskom-AVR

- I2C, Extended I2C,
- MCSBYT, MCSBYTEINT - wspomagają konwersję typów ze stringa,
- TCPIP,
- Floating Point,
- LCD,
- CF Card,
- SPI,
- Data i czas,
- inne.

Nieulotna pamięć danych EEPROM

- ATmega32 zawiera 1024 bajty nieulotnej pamięci danych EEPROM. Przestrzeń adresowa jest liniowa od 0 do 1023.
- Przestrzeń ta jest zorganizowana jako osobna przestrzeń danych z zapisem i odczytem bajtowym,
- Dostęp do pamięci w specjalny sposób poprzez rejestry EEARH EEARL (rejestry adresu), EEDR (rejestr danych), EECR - rejestr kontrolny

Odczyt i zapis z pamięci danych EEPROM

```
EEPROM_write:
  ; Wait for completion of previous write
  sbic EECR,EEWE
  rjmp EEPROM_write
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  out  EEARL, r17
  ; Write data (r16) to data register
  out  EEDR,r16
  ; Write logical one to EEMWE
  sbi  EECR,EEMWE
  ; Start eeprom write by setting EEWE
  sbi  EECR,EEWE
```

```
EEPROM_read:
  ; Wait for completion of previous write
  sbic EECR,EEWE
  rjmp EEPROM_read
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  out  EEARL, r17
  ; Start eeprom read by writing EERE
  sbi  EECR,EERE
  ; Read data from data register
  in   r16,EEDR
```

Języki wysokiego rzędu

- W językach tych pojedynczej instrukcji zazwyczaj odpowiada wiele instrukcji procesora,
- W wyniku kompilacji otrzymany kod jest daleki od najbardziej efektywnego,
- zaletą jest łatwość w programowaniu.

Łączenie języków programowania - wstawki asemblerowe

- W BASCOM-AVR używamy *$ASM* i kończymy *$END ASM*,
- nie wszystkie dyrektywy i makra są dostępne w BASCOM-AVR.

Zadania na ćwiczenia

1. Napisz program, który posługując się językiem asembler, wpisze zawartość rejestru *r16* do zmiennej typu integer. Zmienną tą wyślij na konsole, używając funkcji Bascom-AVR.
2. Napisz program, który posługując się językiem asembler, zapisuje do obszaru nieulotnej pamięci danych 1 bajt pod adres wskazany przez osobę prowadzącą[a].
3. Napisz program, który posługując się językiem asembler, liczy liczbę resetów. Licznik ma liczyć resety nawet po odłączeniu zasilania. Wynik (liczbę resetów) wyślij na konsole, używając funkcji Bascom-AVR.

[a] adres powyżej 255