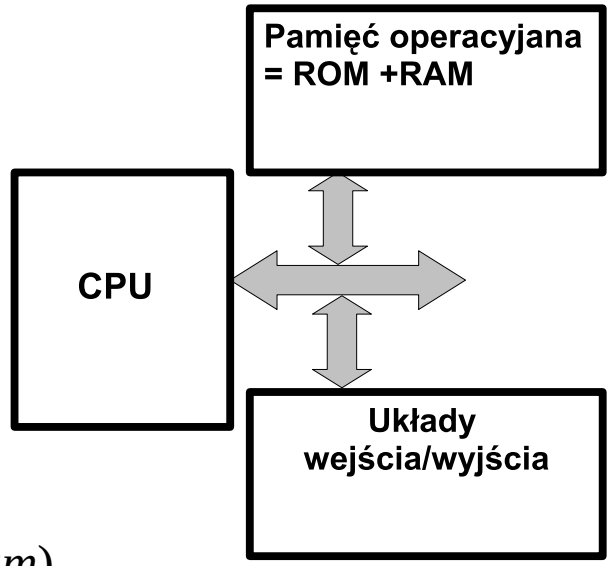
Elementy składowe μC - przypomnienie



Elementy składowe μC:

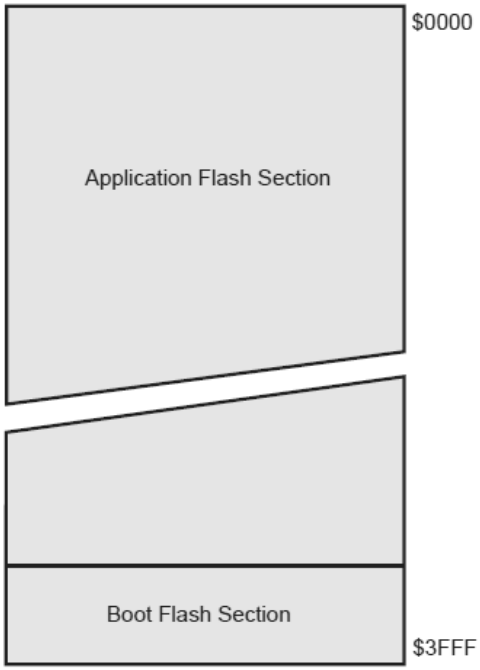
- procesor z ALU
- pamięć komputera (zawierająca *dane* i *program*)
- urządzenia wejścia/wyjścia

Obraz pamięci ROM i RAM

μC AVR ATMega32 posiada architekturę Harwardzką - osobna przestrzeń adresowa (szyny danych i adresowe) dla pamięci ROM i pamięci RAM

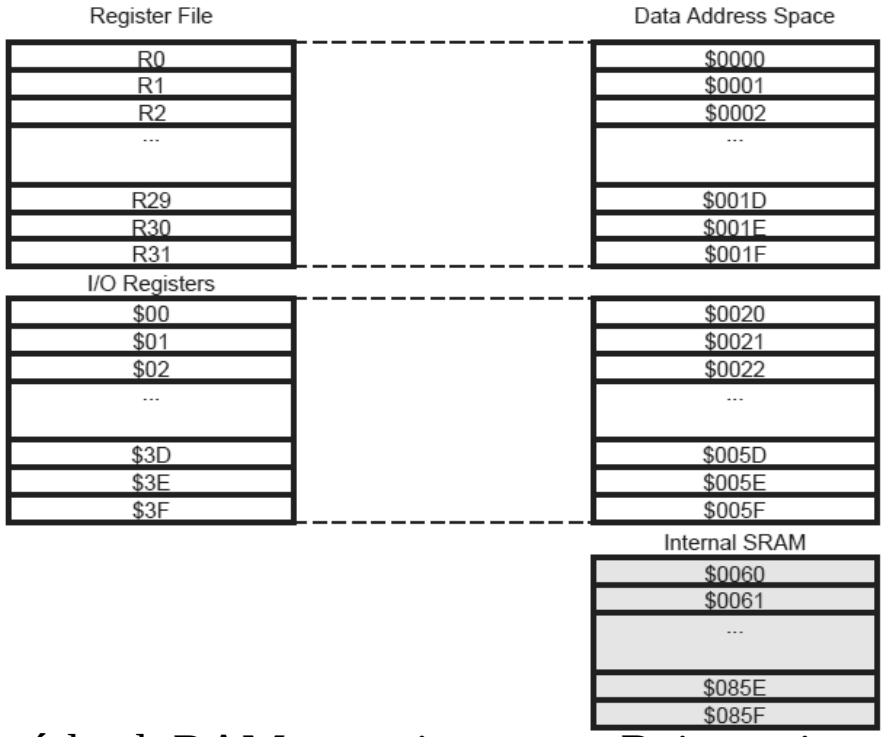
- *Pamięć ROM* - 16kB pamięci Flesh (programu) zorganizowanej po 16 bitów. Szacowana żywotność pamięci Flesh - kilka tysięcy cykli zapisu i odczytu. Do adresowania pamięci stosuje się 14 bitowy licznik programu - rejestr PC.
- *Pamięć RAM* - 32kB pamięci zorganizowanej po 8 bitów.

Obraz pamięci programu



- *bootloader* - istnieje możliwość uruchomienia programu z bootloadera po resecie,
- *bootloader* umożliwia programowanie pamięci programu bez użycia programatora (tryby SPI, JTAG, albo równoległego),
- "właściwy" program umieszczany jest na początku przestrzeni programu.

Pamięć danych



- W pierwszych 96 komórkach RAM są umieszczone Rejestry i pamięć I/O,
- Od 97 do 2048 jest wewnętrzna pamięć SRAM,
- 32 rejestry ogólnego przeznaczenia, 64 rejestry I/O.
- Jest pięć sposobów (trybów) adresowania danych: *bezpośrednie*, *pośrednie z przemieszczeniem*, *pośrednie*, *pośrednie z preinkrementacją*, *pośrednie z postinkremetacją*,
- Do adresowań pośrednich wykorzystuje się rejestry od R26 do R31.

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
```

Kod maszynowy

Kod maszynowy to postać programu komputerowego (zwana postacią wykonywalną lub binarną) przeznaczona do bezpośredniego lub prawie bezpośredniego wykonania przez procesor.

- Jest ona dopasowana do konkretnego typu procesora i wyrażona w postaci rozumianych przez niego kodów rozkazów i ich argumentów,
- Po kompilacji kod maszynowy zapisywany jest w plikach o rozszerzeniu *hex* albo *bin*,
- Bardzo trudna i nieczytelna do bezpośredniej analizy.

Asembler

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

- ldi r16, 15
- inc r16

Podstawowe elementy języka Asembler

- Etykieta
- dyrektywy
- makra
- podoprogramy

Dyrektywy asemblera

- .*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.

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

Kompilacja

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

- Kompilacja może być częścią większego "procesu tłumaczenia", tworzony w jej trakcie kod wynikowy (object code) jest przekazywany do innych programów (linkera),
- Nazwa kompilacja na co dzień jest używana w kontekście tłumaczenia z języka wyższego poziomu na język niższego poziomu.
- Tłumaczenie w odwrotnym kierunku określa się terminem dekompilacji.

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.

Tryb adresowania

- bezpośrednie,
- pośrednie z przemieszczeniem,
- pośrednie,
- pośrednie z preinkrementacją,
- pośrednie z postinkremetacją.

Sposoby programowania mikrokontrolerów

Mikrokontrolery można programować na trzy sposoby:

1. *High voltage Programming* czyli sposób programowania wprowadzony ponad 15lat temu do programowania pamięci EPROM za pomocą sygnałów 12V.
2. *Emulacja pamięci ROM*
3. *ISP (In-System Programmable)* które nie wymaga wyjmowania układu z systemu w którym pracuje.
4. wykorzystuje *Bootloader*, czyli "kawałek" kodu który programuje nasz mikrokontroler np. przez port szeregowy, Bluetooth czy USB. Nie wymaga posiadania programatora poza pierwszym programowaniem (jakoś trzeba wgrać Bootloader).

JTAG

- JTAG (ang. Joint Test Action Group) to nazwa standardu IEEE 1149.1 definiującego protokół używany do testowania połączeń na płytkach drukowanych,
- Stosowany także do uruchamiania i programowania układów programowalnych i systemów mikroprocesorowych,
- Jednym z najważniejszych założeń standardu JTAG jest możliwość programowania układu bez zewnętrznego programatora (ang. In-System Programming, w skrócie ISP).

Programowanie poprzez ISP

- 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, z typowego napięcia zasilającego, 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.

Zadania na ćwiczenia

Wszystkie zadania należy wykonać w środowisku BASCOM-AVR

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.
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.