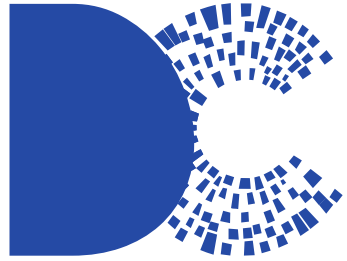
UNIVERSIDADE FEDERAL DE SERGIPE



DEPARTAMENTO DE COMPUTAÇÃO

Arquitetura AVR

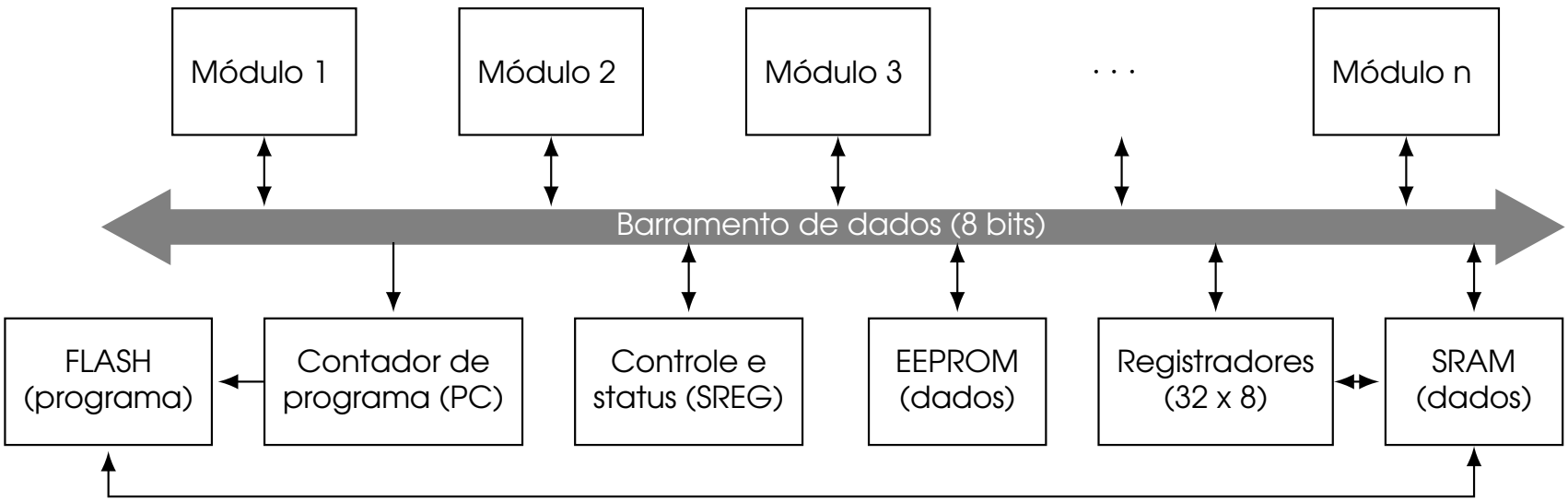
Sistemas Embarcados

Bruno Prado

Departamento de Computação / UFS

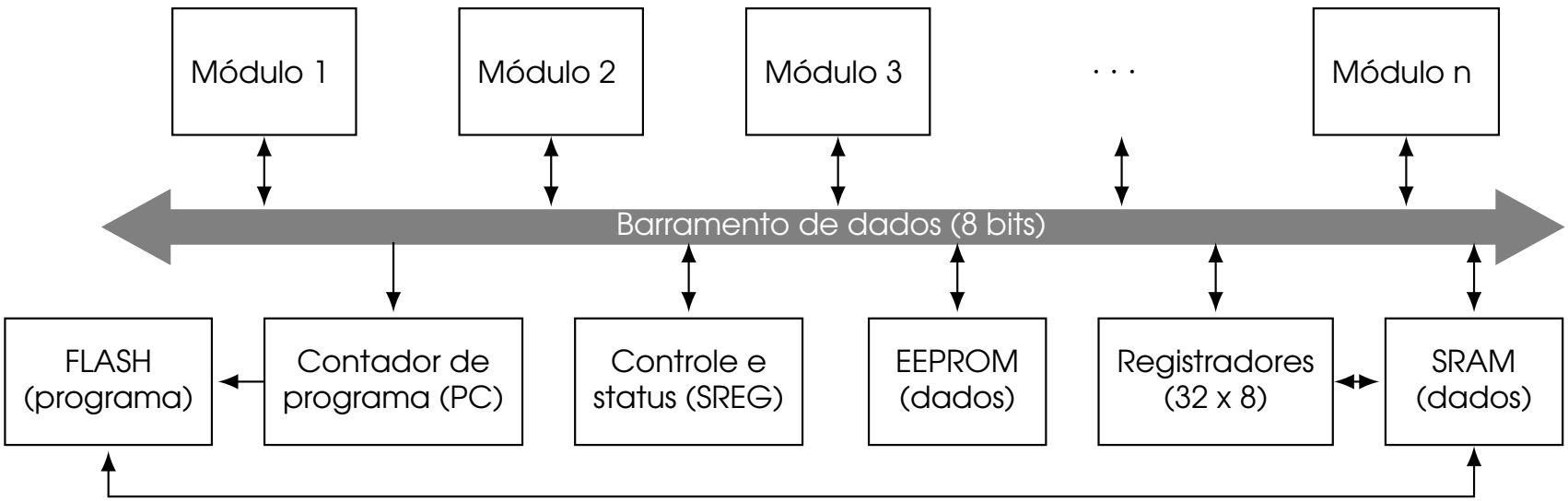
Introdução

- Visão geral
  - Criada em 1996 no Instituto de Ciências Norueguês pelos estudantes **A**lf-Egil Bogen e **V**egard Wollan
  - Arquitetura **R**ISC Harvard modificada de 8 bits



Introdução

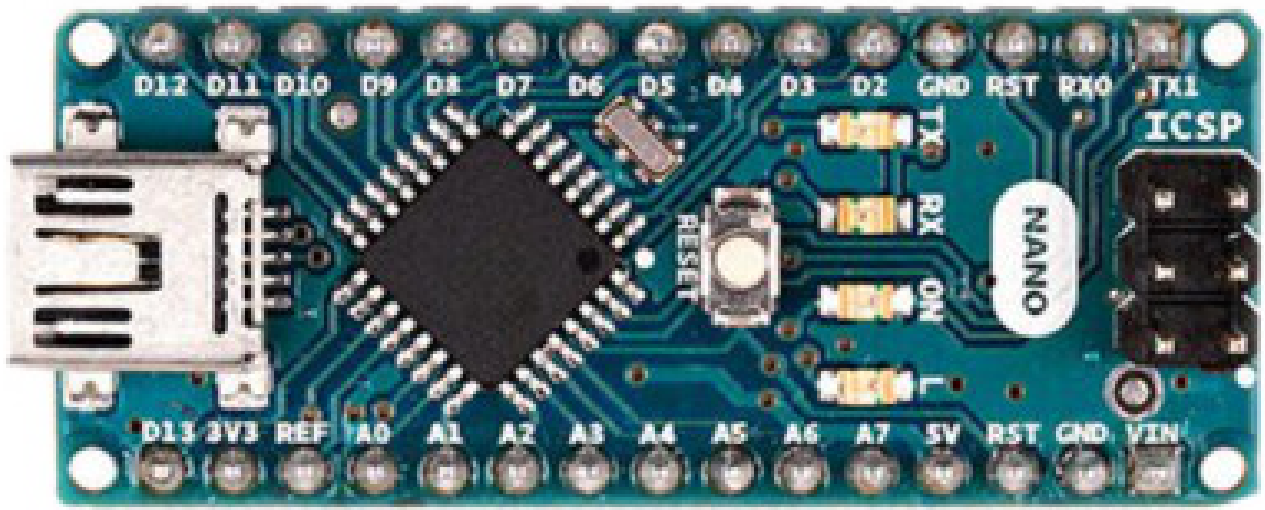
- Visão geral
  - Criada em 1996 no Instituto de Ciências Norueguês pelos estudantes **A**lf-Egil Bogen e **V**egard Wollan
  - Arquitetura **R**ISC Harvard modificada de 8 bits



Popularizada pelo projeto de hardware aberto Arduino

Introdução

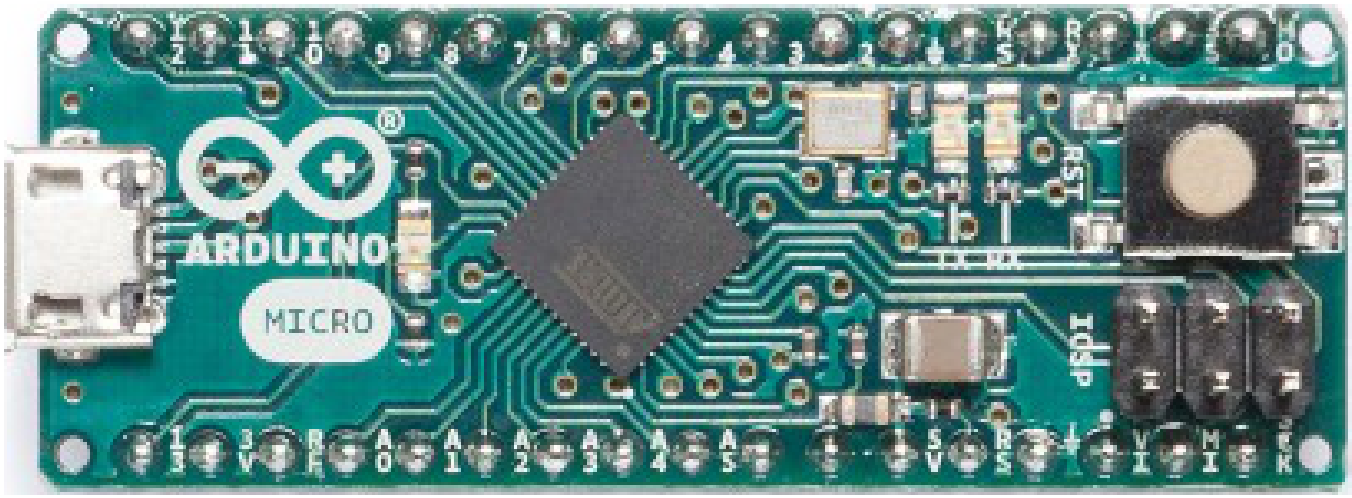
- Microcontrolador de 8 bits ATmega328P
  - Custo < US$ 2
  - Desempenho máximo de 16 MIPS @ 16 MHz
  - FLASH de 32 KB (código) e SRAM de 2 KB (dados)
  - Temperatura de operação entre -40° C e +125° C
  - Voltagem: 2,7 V até 5,5 V (40 uA até 14 mA)



Fonte: https://store.arduino.cc/usa/arduino-nano

Introdução

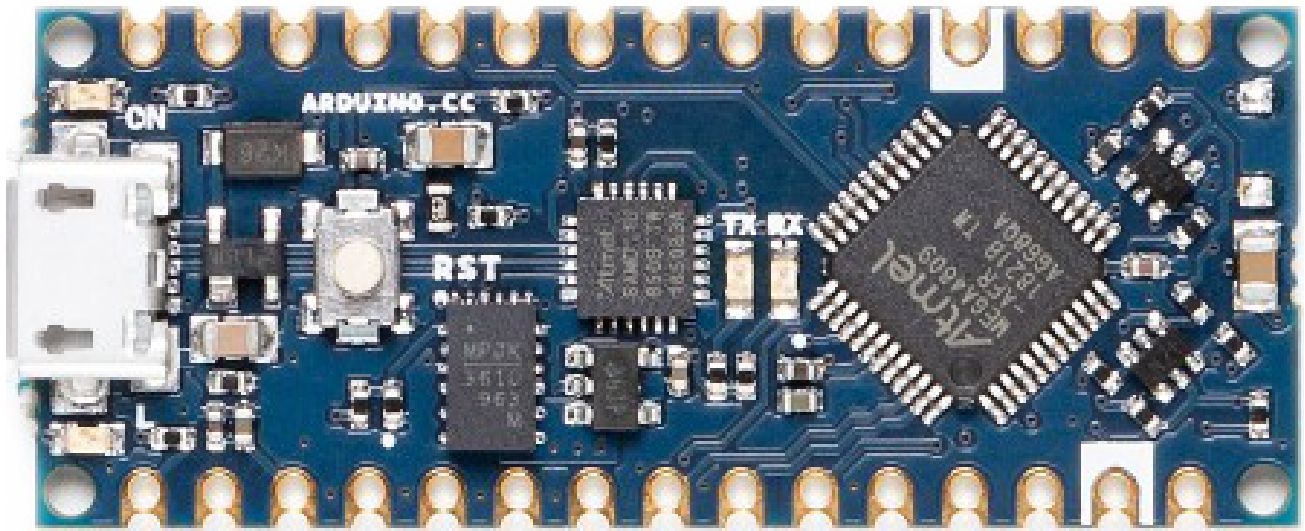
- Microcontrolador de 8 bits ATmega32U4
  - Custo < US$ 4
  - Desempenho máximo de 16 MIPS @ 16 MHz
  - FLASH de 32 KB (código) e SRAM de 2,5 KB (dados)
  - Temperatura de operação entre -40° C e +85° C
  - Voltagem: 2,7 V até 5,5 V (1 uA até 27 mA)



Fonte: https://store.arduino.cc/usa/arduino-micro

Introdução

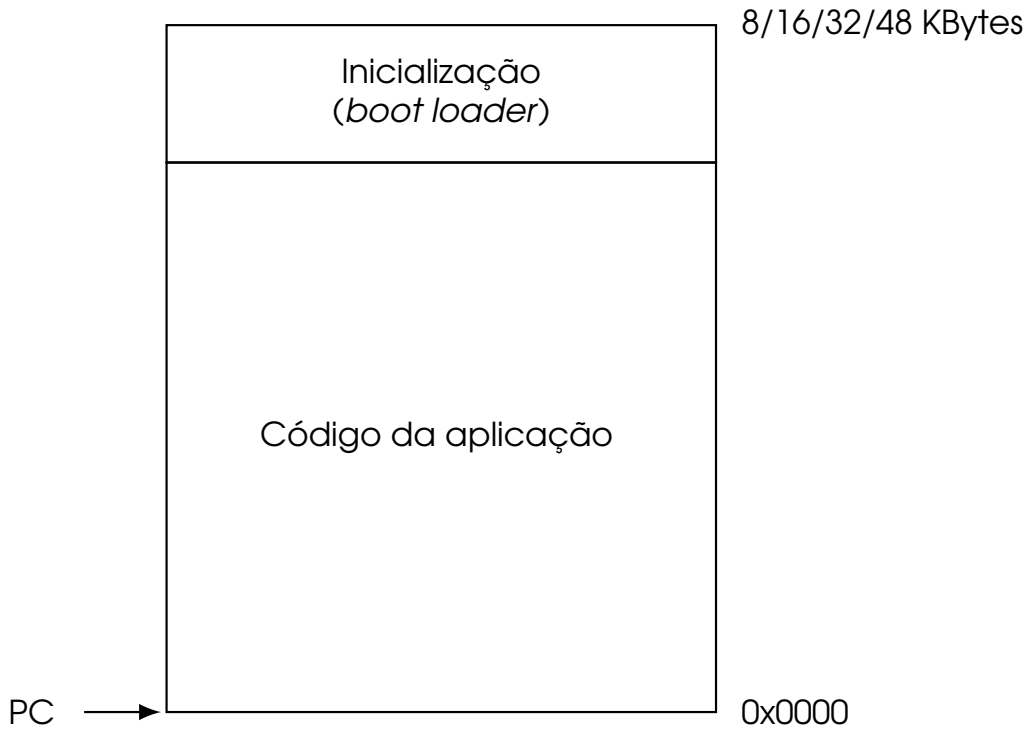
- Microcontrolador de 8 bits ATmega4809
  - Custo < US$ 1,5
  - Desempenho máximo de 20 MIPS @ 20 MHz
  - FLASH de 48 KB (código) e SRAM de 6 KB (dados)
  - Temperatura de operação entre -40° C e +105° C
  - Voltagem: 2,7 V até 5,5 V (0,1 uA até 8,5 mA)



Fonte: https://store.arduino.cc/usa/nano-every

Memória de programa

- Memória não volátil (FLASH)
  - As instruções possuem 16 ou 32 bits de tamanho que são indexadas pelo contador de programa (PC)
  - Permite pelo menos 10.000 até 100.000 ciclos de escrita/leitura de palavras de 16 bits (*endurance*)
  - Programação por interface SPI ou JTAG



Memória de programa

- Inicialização programada (*boot loader*)
  - Permite que o sistema seja capaz de atualizar a aplicação por conta própria, através de rotina de inicialização armazenada na memória de programa
  - Implementa mecanismos de proteção para definir que áreas da memória podem ser modificadas
  - Qualquer interface e protocolo disponíveis podem ser utilizados para transferência de dados

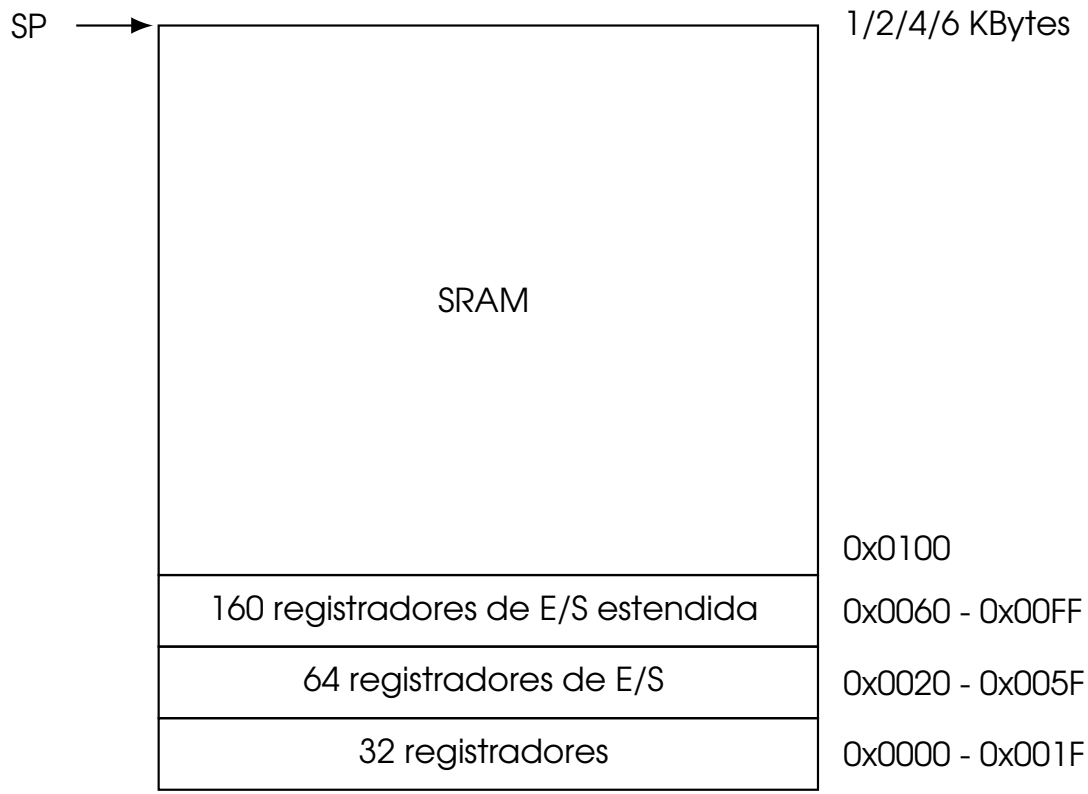
Memória de programa

- Inicialização programada (*boot loader*)
  - Permite que o sistema seja capaz de atualizar a aplicação por conta própria, através de rotina de inicialização armazenada na memória de programa
  - Implementa mecanismos de proteção para definir que áreas da memória podem ser modificadas
  - Qualquer interface e protocolo disponíveis podem ser utilizados para transferência de dados

Podem ser dedicados de 512 até 4096 bytes
para sua implementação

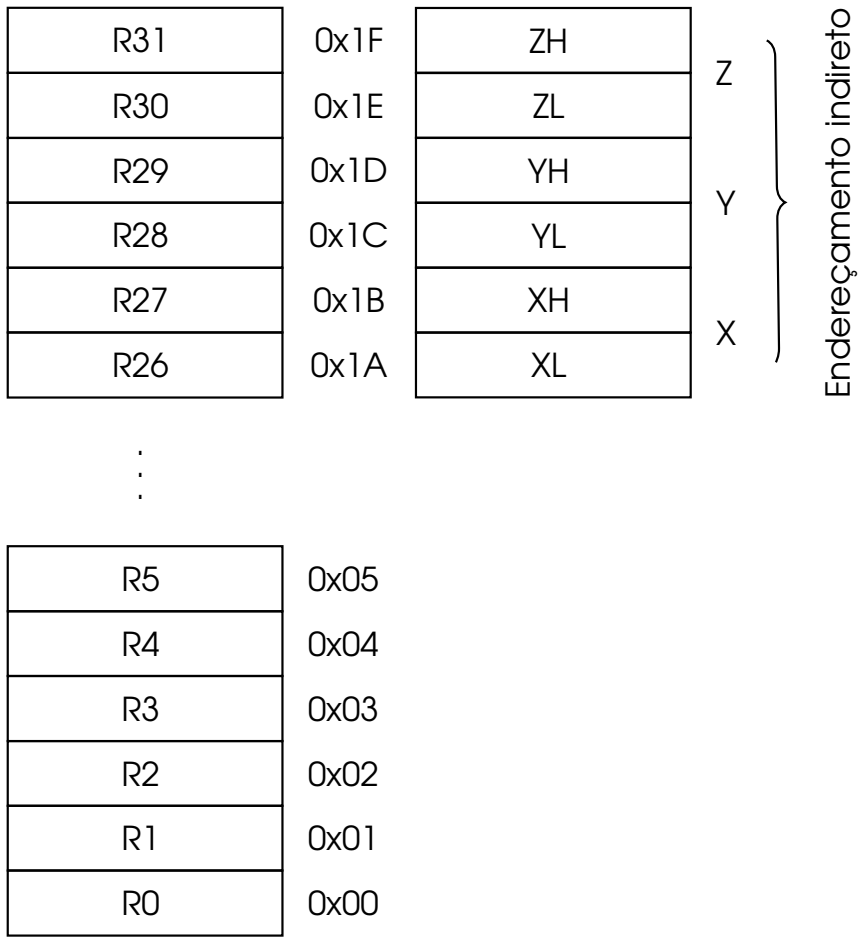
Memória de dados

- Memória volátil (registradores + SRAM)
  - Registradores mapeados e dados da aplicação



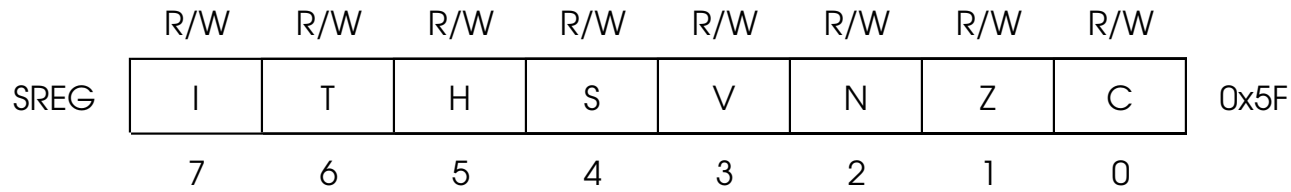
Memória de dados

- Registradores de propósito geral
  - 32 registradores de 8 bits



Memória de dados

- Registrador de controle e de status (SREG)



  - I: habilitação de interrupção global
  - T: armazenamento de bit de cópia
  - H: operação com *half carry*
  - S: bit de sinal
  - V: *overflow* complemento a 2
  - N: número negativo
  - Z: número zero
  - C: operação com *carry*

Memória de dados

- Memória não volátil (EEPROM)
  - Acessada por registradores de E/S, utiliza um espaço de memória de dados separado e garante pelo menos 100.000 ciclos de escrita/leitura de bytes

	7	6	5	4	3	2	1	0	
	R	R	R	R	R/W	R/W	R/W	R/W	
EEADH	-	-	-	-	EEAR11	EEAR10	EEAR9	EEAR8	0x42
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
EEADL	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	0x41
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
EEDR	EEDR7	EEDR6	EEDR5	EEDR4	EEDR3	EEDR2	EEDR1	EEDR0	0x40
	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
EECR	-	-	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	0x3F

Memória de dados

- Memória não volátil (EEPROM)
  - Escrever dado na memória

```
1  // Definições de E/S
2  #include <avr/io.h>
3  // Tipos inteiros de tamanho fixo
4  #include <stdint.h>
5  // Procedimento de escrita da EEPROM
6  void escrever(uint16_t address, uint8_t data) {
7      // Esperando por última escrita
8      while(EECR & (1 << EEPE));
9      // Ajustando o endereço e dado
10     EEAR = address;
11     EEDR = data;
12     // Habilitando modo de escrita
13     EECR = EECR | (1 << EEMPE);
14     // Iniciando escrita da EEPROM
15     EECR = EECR | (1 << EEPE);
16 }
```

Memória de dados

- Memória não volátil (EEPROM)
  - Ler dado da memória

```
1  // Definições de E/S
2  #include <avr/io.h>
3  // Tipos inteiros de tamanho fixo
4  #include <stdint.h>
5  // Função de leitura da EEPROM
6  uint8_t ler(uint16_t address) {
7      // Esperando por última escrita
8      while(EECR & (1 << EEPE));
9      // Ajustando o endereço
10     EEAR = address;
11     // Iniciando leitura da EEPROM
12     EECR = EECR | (1 << EERE);
13     // Retornando dado
14     return EEDR;
15 }
```

AVR Libc

- Biblioteca padrão de C para AVR
  - Padrão
    - Cadeias de caracteres (string.h)
    - Entrada e saída (stdio.h)
    - Funções e procedimentos genéricos (stdlib.h)
    - ...
  - AVR
    - Atraso de tempo (util/delay.h)
    - Definições de E/S (avr/io.h)
    - Gerenciamento de EEPROM (avr/eeprom.h)
    - ...

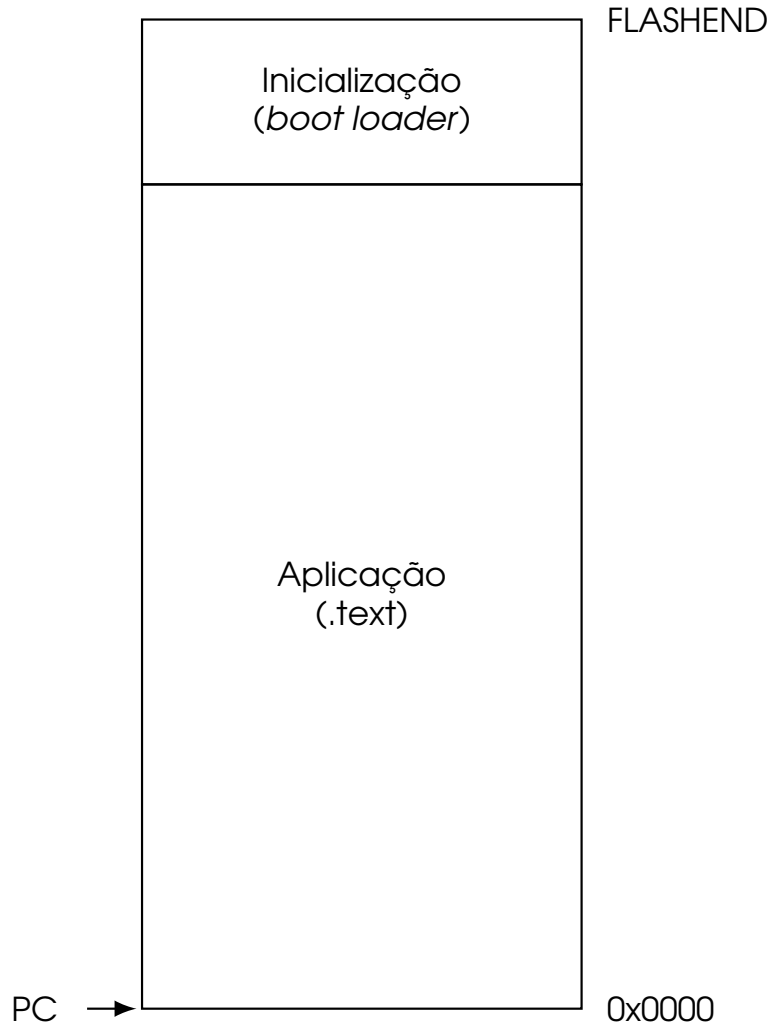
AVR Libc

- Biblioteca padrão de C para AVR
  - Padrão
    - Cadeias de caracteres (string.h)
    - Entrada e saída (stdio.h)
    - Funções e procedimentos genéricos (stdlib.h)
    - ...
  - AVR
    - Atraso de tempo (util/delay.h)
    - Definições de E/S (avr/io.h)
    - Gerenciamento de EEPROM (avr/eeprom.h)
    - ...

https://www.nongnu.org/avr-libc/user-manual/index.html

AVR Libc

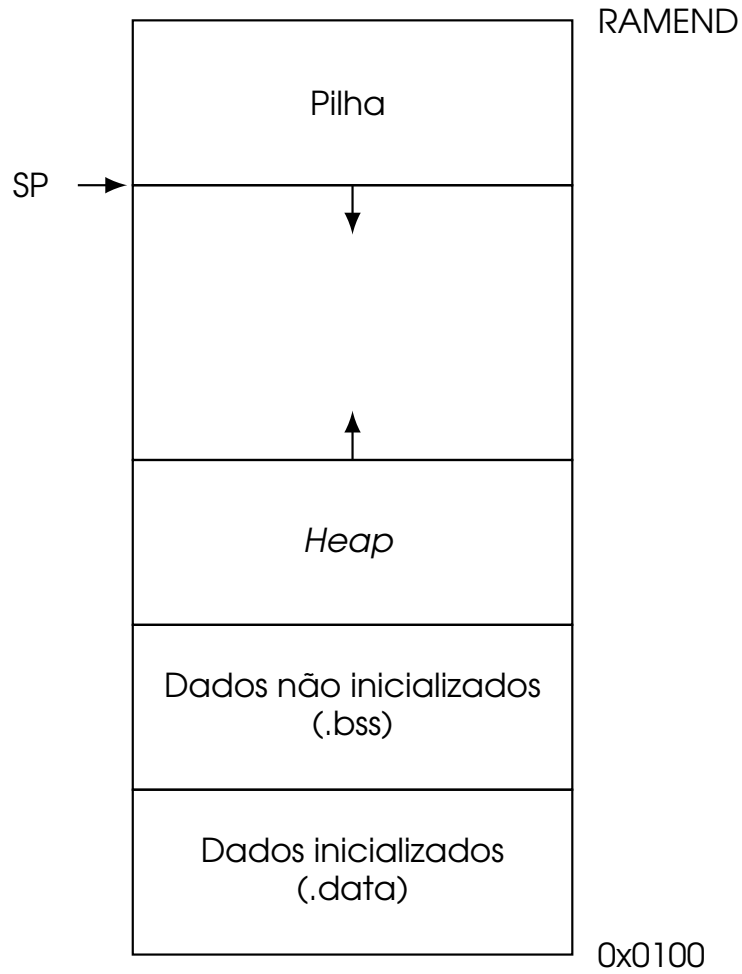
- Organização da memória de programa (FLASH)



Instruções do programa e dados constantes

AVR Libc

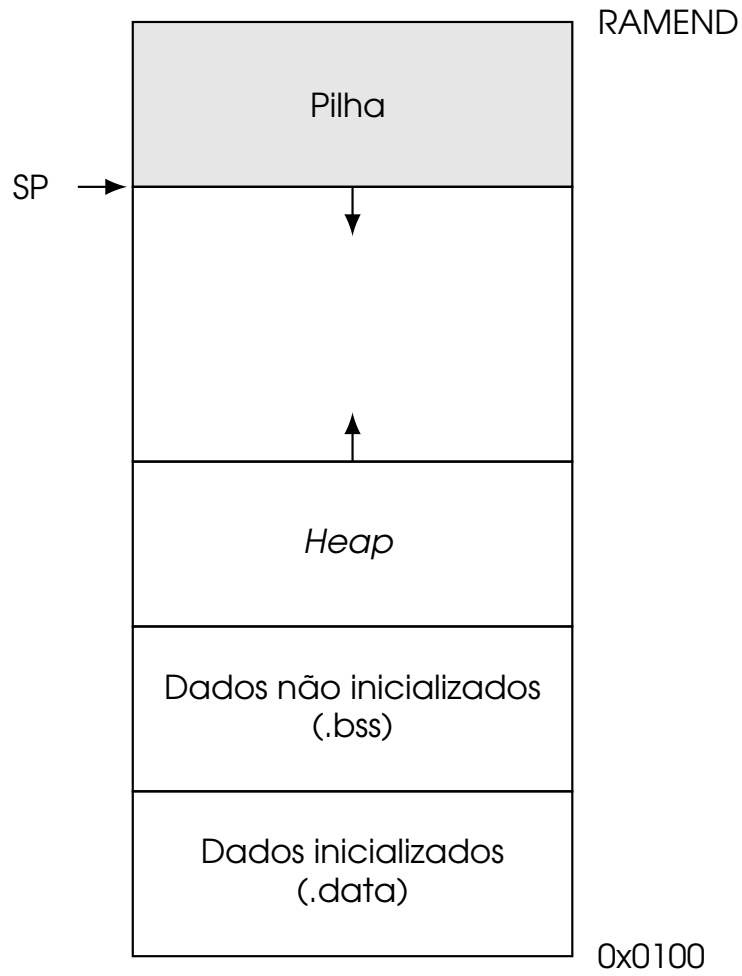
- Organização da memória de dados (SRAM)



Segmentos de memória

AVR Libc

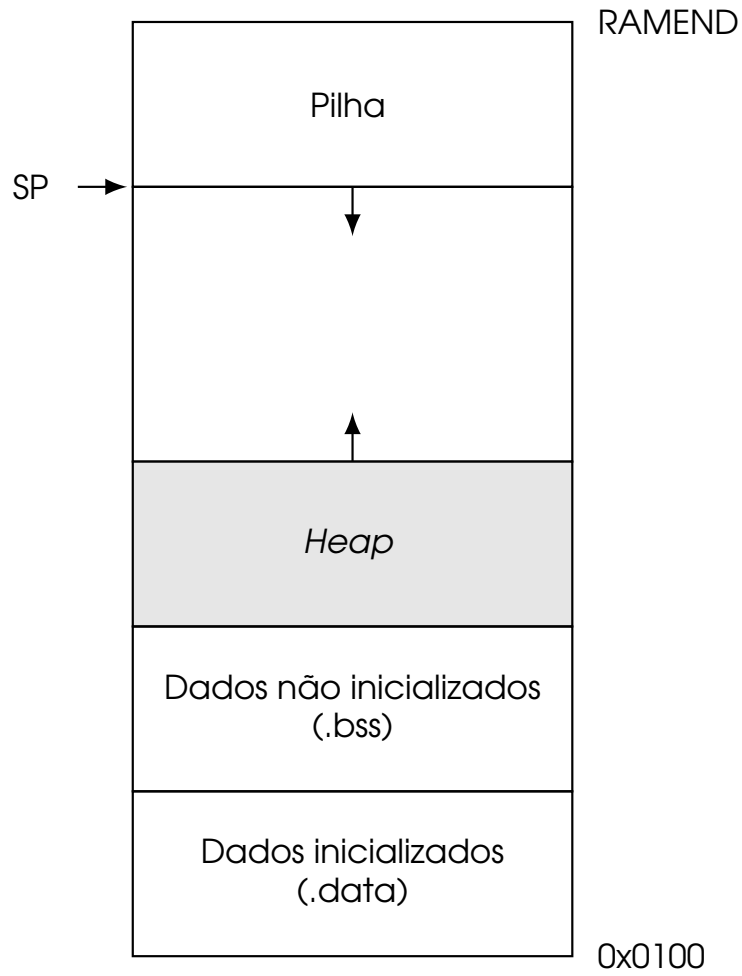
- Organização da memória de dados (SRAM)



Chamadas de funções e alocação estática

AVR Libc

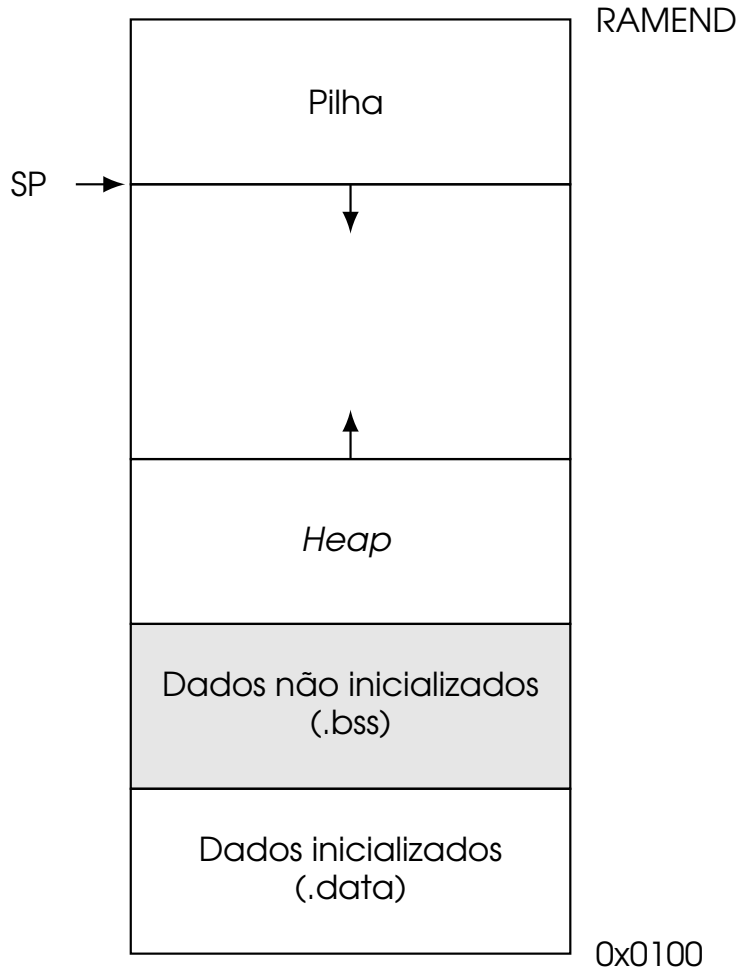
- Organização da memória de dados (SRAM)



Alocação dinâmica de memória (*malloc* e *free*)

AVR Libc

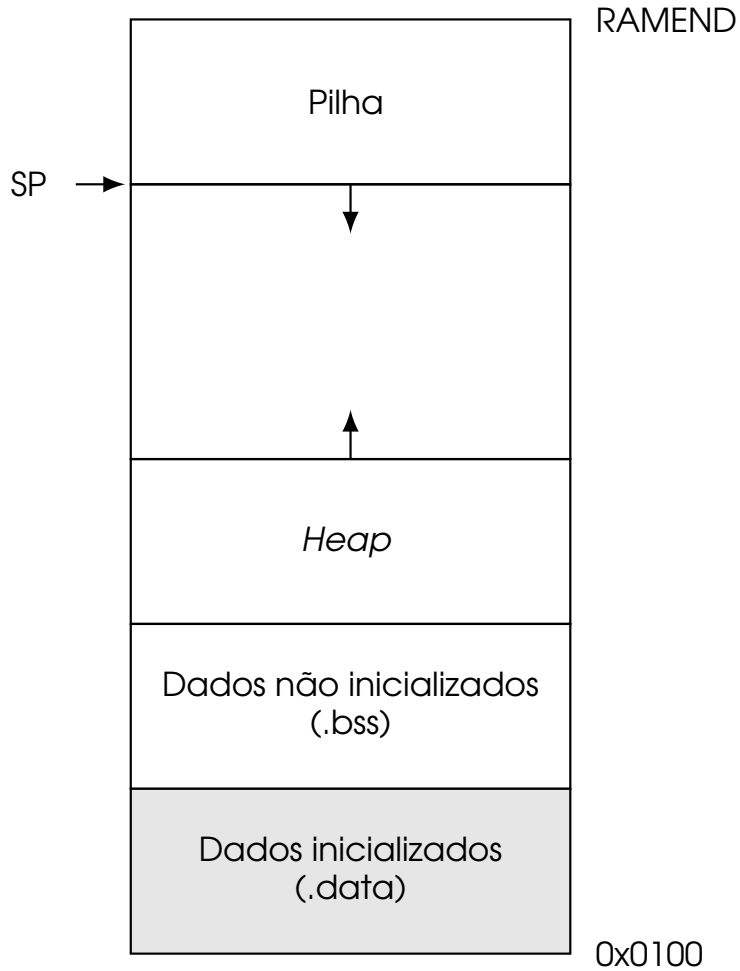
- Organização da memória de dados (SRAM)



Variáveis declaradas sem valor inicial

AVR Libc

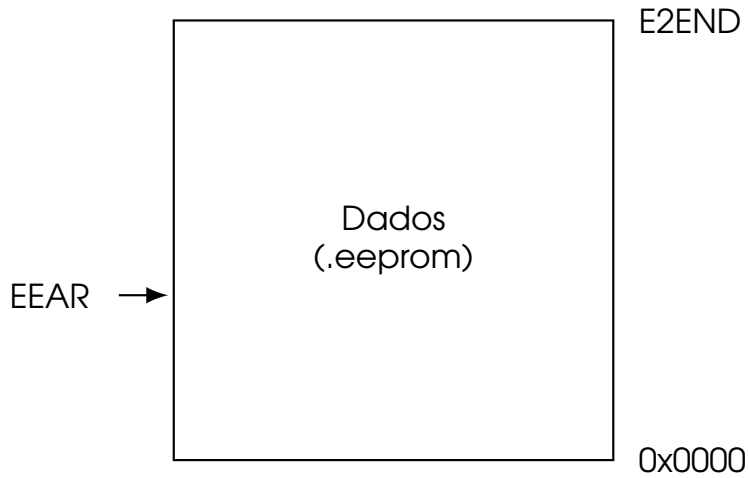
- Organização da memória de dados (SRAM)



Variáveis com valores inicializados

AVR Libc

- Organização da memória de dados (EEPROM)



Dados não voláteis da aplicação

AVR Libc

- Armazenando os dados na memória de programa

```
1 // Sequencia de Fibonacci (64 x 8 bytes = 512 bytes)
2 uint64_t fib[64] = { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
    55, 89, 144, 233, 377, 610, 987, 1597, 2584,
    4181, 6765, 10946, 17711, 28657, 46368, 75025,
    121393, 196418, 317811, 514229, 832040, 1346269,
    2178309, 3524578, 5702887, 9227465, 14930352,
    24157817, 39088169, 63245986, 102334155,
    165580141, 267914296, 433494437, 701408733,
    1134903170, 1836311903, 2971215073, 4807526976,
    7778742049, 12586269025, 20365011074,
    32951280099, 53316291173, 86267571272,
    139583862445, 225851433717, 365435296162,
    591286729879, 956722026041, 1548008755920,
    2504730781961, 4052739537881, 6557470319842 };
```

AVR Libc

- Armazenando os dados na memória de programa

```
1 // Sequencia de Fibonacci (64 x 8 bytes = 512 bytes)
2 uint64_t fib[64] = { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
    55, 89, 144, 233, 377, 610, 987, 1597, 2584,
    4181, 6765, 10946, 17711, 28657, 46368, 75025,
    121393, 196418, 317811, 514229, 832040, 1346269,
    2178309, 3524578, 5702887, 9227465, 14930352,
    24157817, 39088169, 63245986, 102334155,
    165580141, 267914296, 433494437, 701408733,
    1134903170, 1836311903, 2971215073, 4807526976,
    7778742049, 12586269025, 20365011074,
    32951280099, 53316291173, 86267571272,
    139583862445, 225851433717, 365435296162,
    591286729879, 956722026041, 1548008755920,
    2504730781961, 4052739537881, 6557470319842 };
```

Memória de dados é limitada (8,3% até 50%)

AVR Libc

- Armazenando os dados na memória de programa

```
1  // Biblioteca para armazenamento no espaço de programa
2  #include <avr/pgmspace.h>
3  // Sequencia de Fibonacci (64 x 8 bytes = 512 bytes)
4  uint64_t fib[64] PROGMEM = { 0, 1, 1, 2, 3, 5, 8, 13,
       21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
       2584, 4181, 6765, 10946, 17711, 28657, 46368,
       75025, 121393, 196418, 317811, 514229, 832040,
       1346269, 2178309, 3524578, 5702887, 9227465,
       14930352, 24157817, 39088169, 63245986,
       102334155, 165580141, 267914296, 433494437,
       701408733, 1134903170, 1836311903, 2971215073,
       4807526976, 7778742049, 12586269025, 20365011074,
       32951280099, 53316291173, 86267571272,
       139583862445, 225851433717, 365435296162,
       591286729879, 956722026041, 1548008755920,
       2504730781961, 4052739537881, 6557470319842 };
```

AVR Libc

- Armazenando os dados na memória de programa

```
1  // Biblioteca para armazenamento no espaço de programa
2  #include <avr/pgmspace.h>
3  // Sequencia de Fibonacci (64 x 8 bytes = 512 bytes)
4  uint64_t fib[64] PROGMEM = { 0, 1, 1, 2, 3, 5, 8, 13,
       21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
       2584, 4181, 6765, 10946, 17711, 28657, 46368,
       75025, 121393, 196418, 317811, 514229, 832040,
       1346269, 2178309, 3524578, 5702887, 9227465,
       14930352, 24157817, 39088169, 63245986,
       102334155, 165580141, 267914296, 433494437,
       701408733, 1134903170, 1836311903, 2971215073,
       4807526976, 7778742049, 12586269025, 20365011074,
       32951280099, 53316291173, 86267571272,
       139583862445, 225851433717, 365435296162,
       591286729879, 956722026041, 1548008755920,
       2504730781961, 4052739537881, 6557470319842 };
```

Alocação na memória de programa (.text)

AVR Libc

- Acessando os dados na memória de programa

```
1  // Biblioteca para armazenamento no espaço de programa
2  #include <avr/pgmspace.h>
3  // Tipos inteiros de tamanho fixo
4  #include <stdint.h>
5  // Função principal
6  int main() {
7      // Variável não inicializada de 64 bits
8      uint64_t data;
9      // Copiando fib[16] da FLASH para SRAM
10     memcpy_P(&data, &fib[16], 8);
11     // Laço infinito
12     while(1);
13 }
```

AVR Libc

- Linguagem de montagem em C (*inline assembly*)
  - Diretivas **asm** e **volatile**
  - Operadores de entrada e de saída
  - Registradores modificados (*clobbered*)

```
1  // asm(código : operandos de saída : operandos de
       entrada [: registradores modificados]);
2  asm("in␣%0,␣%1" : "=r"(value) :
       "I"(_SFR_IO_ADDR(PORTD)));
3  asm volatile(
4       "cli" "\n\t"
5       "ld␣r24,␣%a0" "\n\t"
6       "inc␣r24" "\n\t"
7       "st␣%a0,␣r24" "\n\t"
8       "sei" "\n\t"
9       : : "e"(ptr) : "r24"
10 );
```

AVR Libc

- Linguagem de montagem em C (*inline assembly*)
  - Diretivas **asm** e **volatile**
  - Operadores de entrada e de saída
  - Registradores modificados (*clobbered*)

```
1  // asm(código : operandos de saída : operandos de
       entrada [: registradores modificados]);
2  asm("in␣%0,␣%1" : "=r"(value) :
       "I"(_SFR_IO_ADDR(PORTD)));
3  asm volatile(
4       "cli" "\n\t"
5       "ld␣r24,␣%a0" "\n\t"
6       "inc␣r24" "\n\t"
7       "st␣%a0,␣r24" "\n\t"
8       "sei" "\n\t"
9       : : "e"(ptr) : "r24"
10 );
```

https://www.nongnu.org/avr-libc/user-manual/inline_asm.html

Desenvolvimento

- Ligando/desligando LED (*blink*)
  - Bit 5 da Porta B (pino 13 do Arduino Nano)

```
1  // Definições de E/S
2  #include <avr/io.h>
3  // Atraso de tempo
4  #include <util/delay.h>
5  // Função principal
6  int main() {
7      // Ajustando pino como saída
8      DDRB = DDRB | (1 << DDB5);
9      // Laço infinito
10     while(1) {
11         // Ligando o LED
12         PORTB = PORTB | (1 << PORTB5);
13         // Atraso de 1 segundo
14         _delay_ms(1000);
15         // Desligando o LED
16         PORTB = PORTB & ~(1 << PORTB5);
17         // Atraso de 1 segundo
18         _delay_ms(1000);
19     }
20 }
```

Desenvolvimento

- Processo de compilação e programação

Desenvolvimento

- Processo de compilação e programação

```
$ avr-gcc -Wall -Os -DF_CPU=16000000UL -mmcu=atmega328p blink.c -o bl
ink.elf
$
```

Desenvolvimento

- Processo de compilação e programação

```
$ avr-gcc -Wall -Os -DF_CPU=16000000UL -mmcu=atmega328p blink.c -o bl
ink.elf
$ avr-objcopy -O ihex -R .eeprom blink.elf blink.hex
$
```

Desenvolvimento

- Processo de compilação e programação

```
$ avr-gcc -Wall -Os -DF_CPU=16000000UL -mmcu=atmega328p blink.c -o bl
ink.elf
$ avr-objcopy -O ihex -R .eeprom blink.elf blink.hex
$ avrdude -b 57600 -c arduino -D -p atmega328p -P /dev/ttyUSB0 -U fla
sh:w:blink.hex
```

Desenvolvimento

- Processo de compilação e programação

```
$ avr-gcc -Wall -Os -DF_CPU=16000000UL -mmcu=atmega328p blink.c -o bl
ink.elf
$ avr-objcopy -O ihex -R .eeprom blink.elf blink.hex
$ avrdude -b 57600 -c arduino -D -p atmega328p -P /dev/ttyUSB0 -U fla
sh:w:blink.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0
.02s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading input file "blink.hex"
avrdude: input file blink.hex auto detected as Intel Hex
avrdude: writing flash (176 bytes):

Writing | ################################################## | 100% 0
.16s
```

Desenvolvimento

- Processo de compilação e programação

```
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0
.02s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading input file "blink.hex"
avrdude: input file blink.hex auto detected as Intel Hex
avrdude: writing flash (176 bytes):

Writing | ################################################## | 100% 0
.16s

avrdude: 176 bytes of flash written
avrdude: verifying flash memory against blink.hex:
avrdude: load data flash data from input file blink.hex:
avrdude: input file blink.hex auto detected as Intel Hex
avrdude: input file blink.hex contains 176 bytes
avrdude: reading on-chip flash data:
```

Desenvolvimento

- Processo de compilação e programação

```
avrdude: 176 bytes of flash written
avrdude: verifying flash memory against blink.hex:
avrdude: load data flash data from input file blink.hex:
avrdude: input file blink.hex auto detected as Intel Hex
avrdude: input file blink.hex contains 176 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0
.14s

avrdude: verifying ...
avrdude: 176 bytes of flash verified

avrdude: safemode: Fuses OK (E:00, H:00, L:00)

avrdude done.  Thank you.

$
```

Desenvolvimento

- Processo de compilação e programação

```
avrdude: 176 bytes of flash written
avrdude: verifying flash memory against blink.hex:
avrdude: load data flash data from input file blink.hex:
avrdude: input file blink.hex auto detected as Intel Hex
avrdude: input file blink.hex contains 176 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0
.14s

avrdude: verifying ...
avrdude: 176 bytes of flash verified

avrdude: safemode: Fuses OK (E:00, H:00, L:00)

avrdude done.  Thank you.

$ exit
```

Exercício

- Estude e reproduza os experimentos vistos nesta aula
  - Analise os manuais técnicos (*datasheets*) dos microcontroladores ATmega328/32u4/4809
  - Utilizando o *framework* Arduino, implemente e execute um exemplo equivalente ao *blink*
  - Faça um comparativo de utilização de memória dos binários gerados utilizando a ferramenta avr-size