Projeto de Sistemas Embarcados Microcontrolados
Departamento de Engenharia Elétrica
Universidade de Brasília

Desenvolvimento com o ATmega8

Prof. Geovany A. Borges
gaborges@ene.unb.br

Linha AVR de microcontroladores ATMEL

- Microcontroladores RISC de 8 bits, arquitetura Harvard
- Alto desempenho:
  - Relógio: DC a até 20MHz (atualmente)
  - Uma instrução por ciclo de relógio (1 MIPS por MHz), exceto salto, retorno de interrupção,...
  - 32 registros de propósito geral: R0 a R31
  - 130 instruções assembly
- Baixo custo
- Baixo consumo
  - Dispositivos de 1,8V a 5V
  - Até seis modos de operação em baixo consumo

Linha AVR de microcontroladores ATMEL

- Arquitetura escalável: reusabilidade de código mesmo para dispositivos mais avançados
- Desenvolvimento no sistema:
  - In-System Programming (ISP)
  - On-Chip Debugging (OCD)
- Proteção de código e dados
- Famílias:
  - ATTiny: pouca memória flash (max. 2KB)
  - ATMega: mais memória flash (max. 256KB)
  - AVR32: microcontrolador/DSP de 32 bits (roda Linux!)

Linha AVR de microcontroladores ATMEL

- Suporte:
  - Notas de aplicação (http://www.atmel.com)
  - Comunidade AVRFreaks (http://www.avrfreaks.net/)

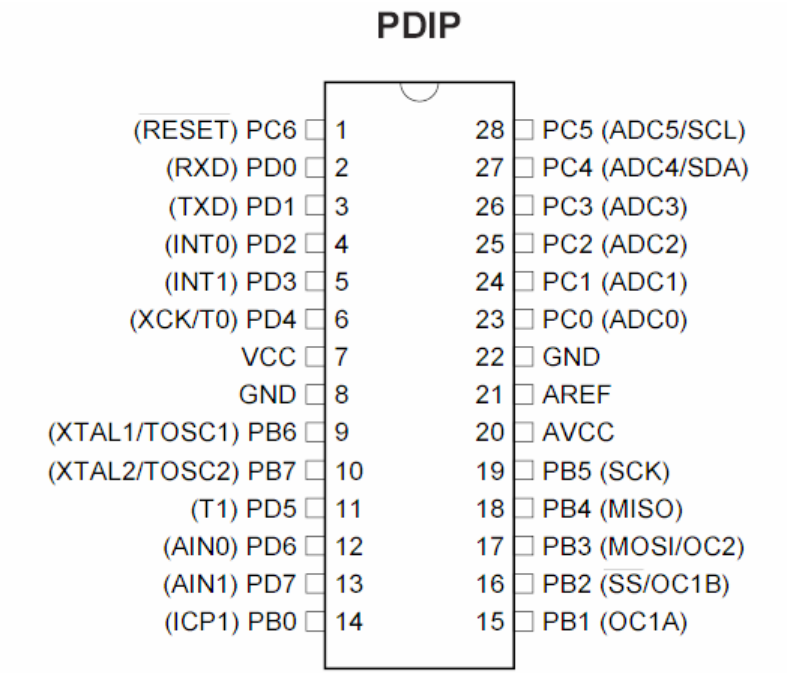
- Ferramentas de código livre:
  - AVR-GCC toolchain
  - avrlib (http://members.home.nl/jmnieuwkamp1/AVRlib/)
  - FreeRTOS (http://www.freertos.org/)

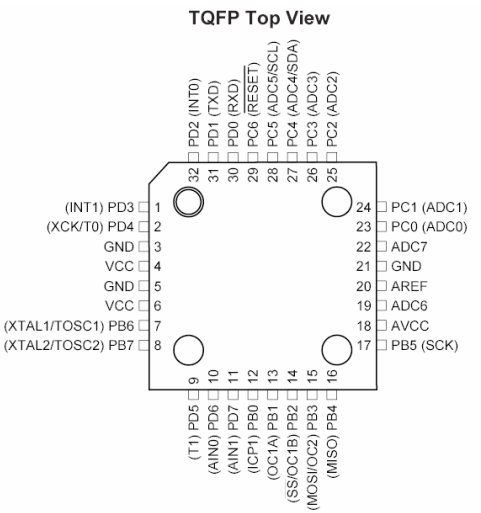
Microcontrolador ATmega8

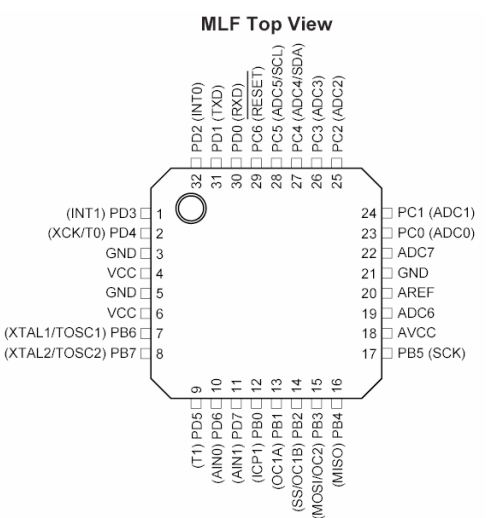
- Características:
  - Memória:
    - 8 KB de memória Flash ISP (10.000 ciclos de escrita/apagamento)
    - 512 B de EEPROM (100.000 ciclos de escrita/apagamento)
    - 1 KB de RAM interna
  - Oscilador RC Interno
  - Várias fontes de interrupção externas e internas
  - Cinco modos de baixo consumo
  - Cinco fontes de relógio do sistema
  - 4,5 a 5V de operação, até 16MHz de relógio

Microcontrolador ATmega8

- Características:
  - Encapsulamentos:





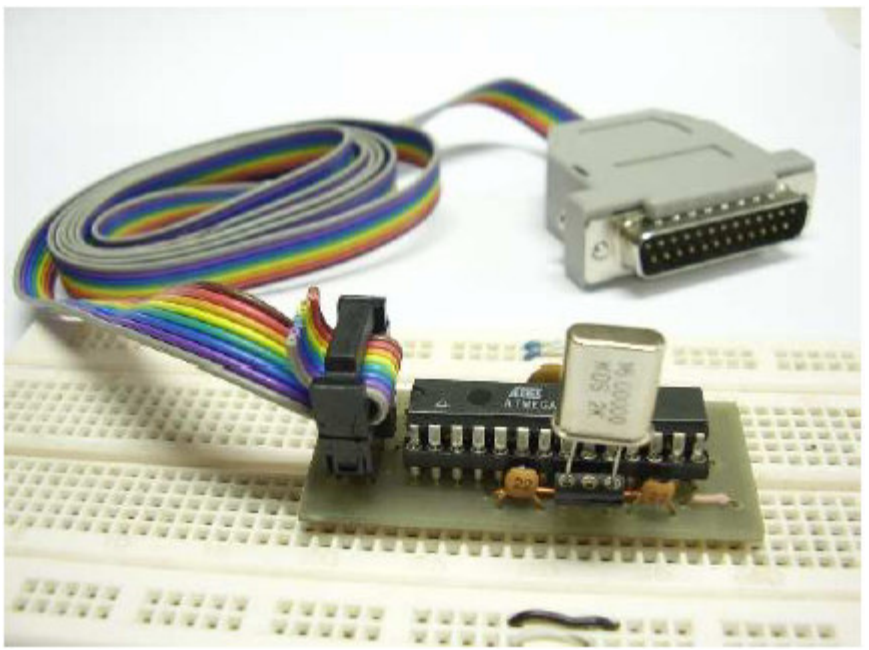


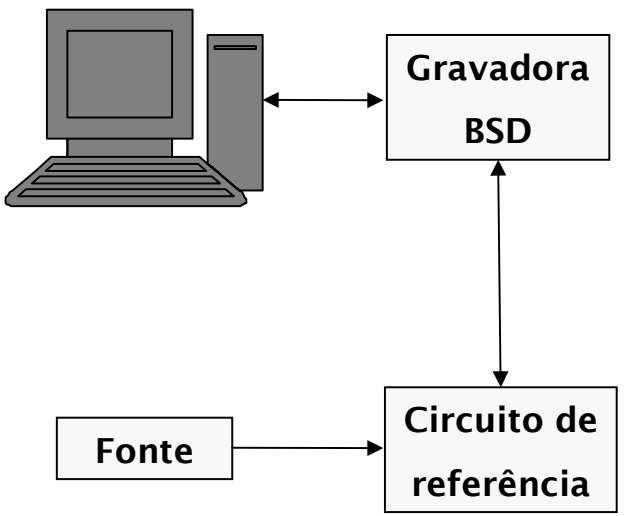
Microcontrolador ATmega8

- Características:
  - Periféricos:
    - Dois contadores/temporizadores de 8 bits
    - Um contador/temporizador de 16 bits
    - RTC com oscilador próprio
    - Geração por hardware de até três canais PWM
    - Conversor A/D de 6 canais: 10-bits (4 canais) e 8-bits (2 canais)
    - Two-wire Serial Interface (TWI)
    - Interface serial USART
    - Interface serial SPI
    - Cão-de-guarda com oscilador próprio
    - Comparador analógico interno

Material de desenvolvimento

- Circuito de referência e gravadora BSD:
  - Para inserção em protoboard com desenvolvimento em microcomputador PC com Windows98/2000/XP





Material de desenvolvimento

- Manual do dispositivo e notas técnicas
- Distribuição WinAVR
  - Toolchain avr-gcc (GCC significa GNU Compiler Collection):
    - Compiladores C/C++
    - Montador assembly (*assembler*)
    - Depurador
    - Linker
    - Binutils
  - Ambiente de desenvolvimento: Programmer's Notepad
  - Gerenciadores da gravadora: avrdude e avrdude-gui
  - Documentação

Material de desenvolvimento

- Nota técnica
  - “Desenvolvimento com microcontroladores Atmel AVR”
- CD de iniciação ao AVR:
  - preparado com ferramentas de hardware/software, documentação e exemplos de programas.

Acessar o site:

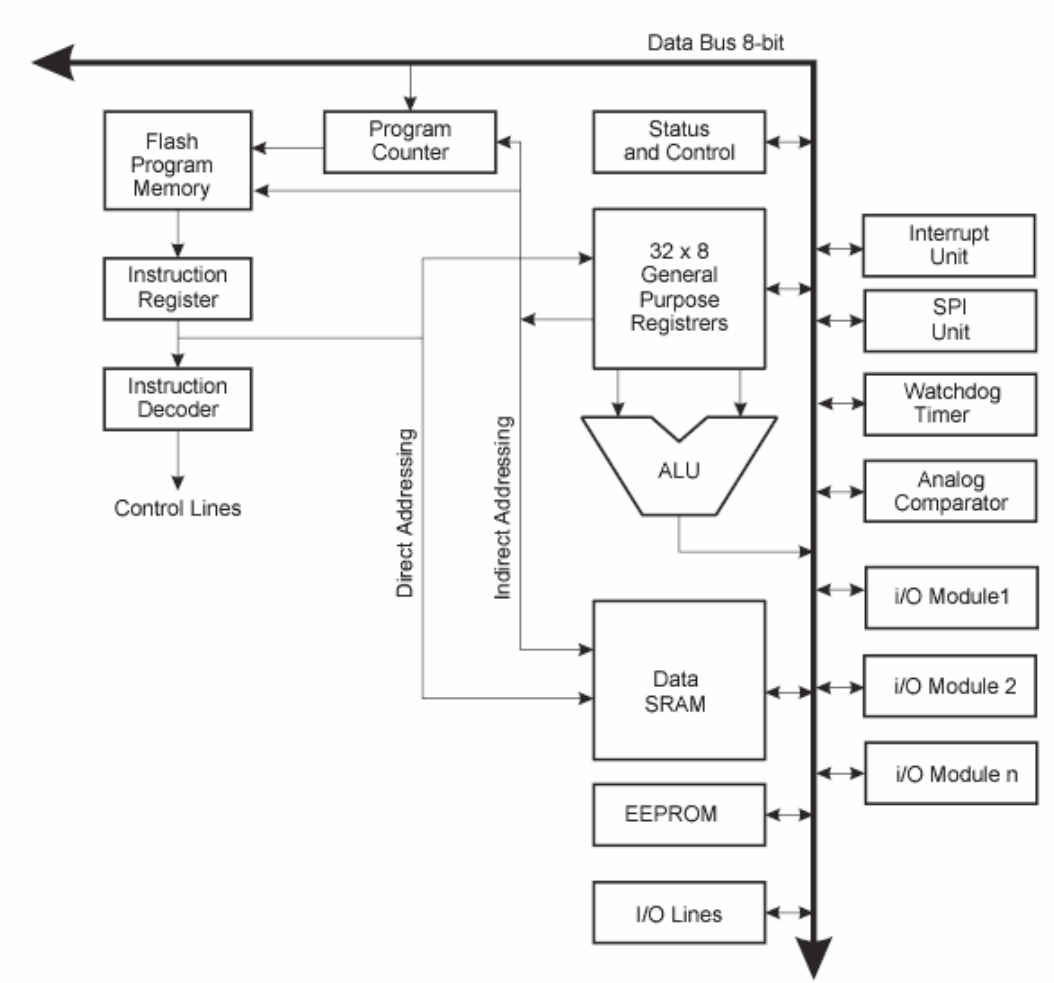
`http://www.ene.unb.br/~gaborges/recursos/embarcados/index.htm`

Material de desenvolvimento

- Iniciação rápida:
  - Instalar o WinAVR (Ferramentas - software\WinAVR). Se houver uma versão anterior, desinstale-a.
  - Construir a gravadora BSD (Ferramentas - hardware\BSD prog simples).
  - Configurar na BIOS do microcomputador a porta paralela em modo Standard (SPP).
  - Construir o circuito de referência (Ferramentas - hardware\Circuito de referência ATMega8)
  - Usar o Programmer's Notepad para desenvolver o projeto
  - Se necessário, usar avrdude.exe (WinAVR\bin) para alterar a configuração do microcontrolador
  - Usar avrdude-gui.exe (WinAVR\bin) para programar o microcontrolador

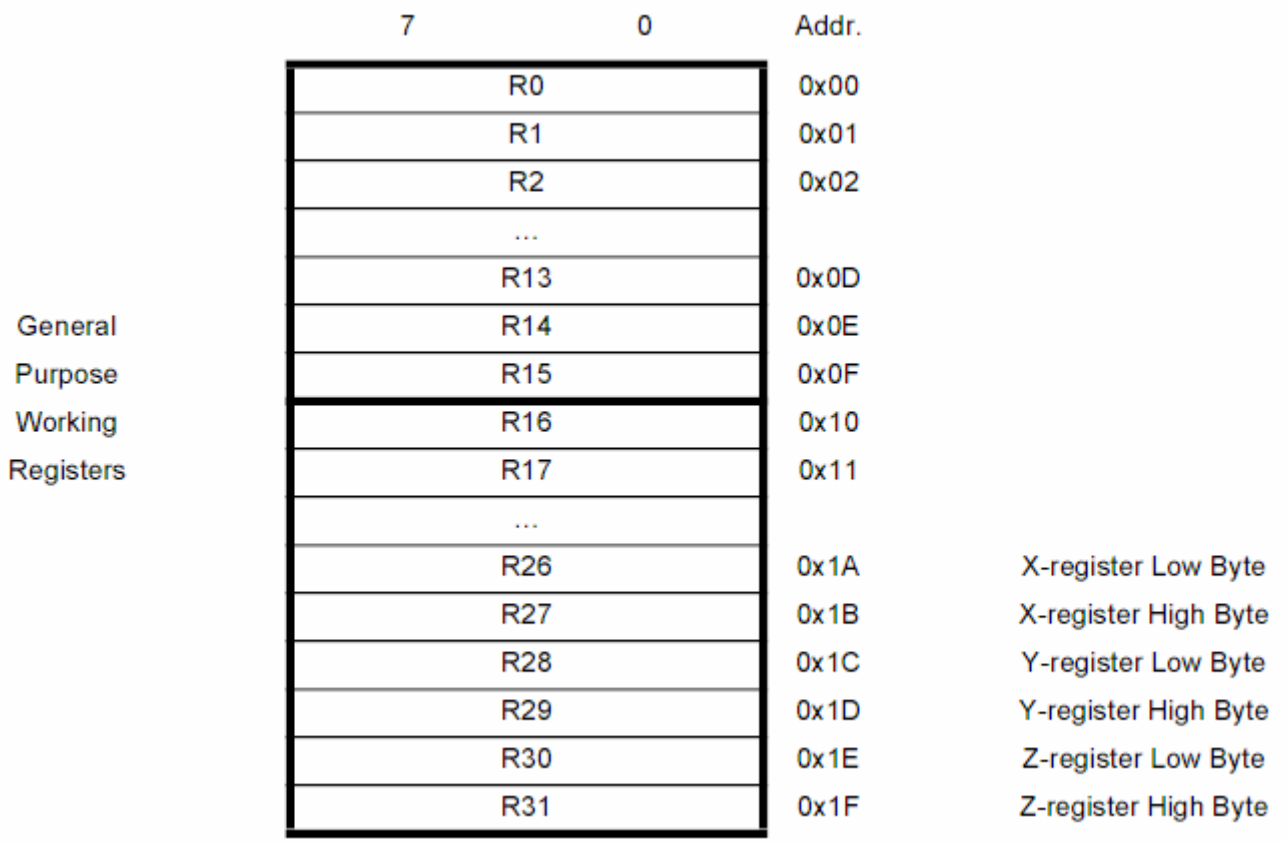
Detalhamento da CPU

- Core da CPU
  - Arquitetura



Detalhamento da CPU

- Core da CPU
  - Registros de trabalho (8 bits)



Detalhamento da CPU

- Core da CPU
  - Registro de *Status*

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

I : Habilitação global de interrupções

T : Bit auxiliar para instruções BLD (Bit LoaD) e BST (Bit STore)

H : Half Carry (usado em operações com codificação decimal)

S : Sinal, S = N ⊕ V

V : Overflow em complemento de 2

N : Indica resultado negativo

Z : Indica Zero

C : Vai um

Detalhamento da CPU

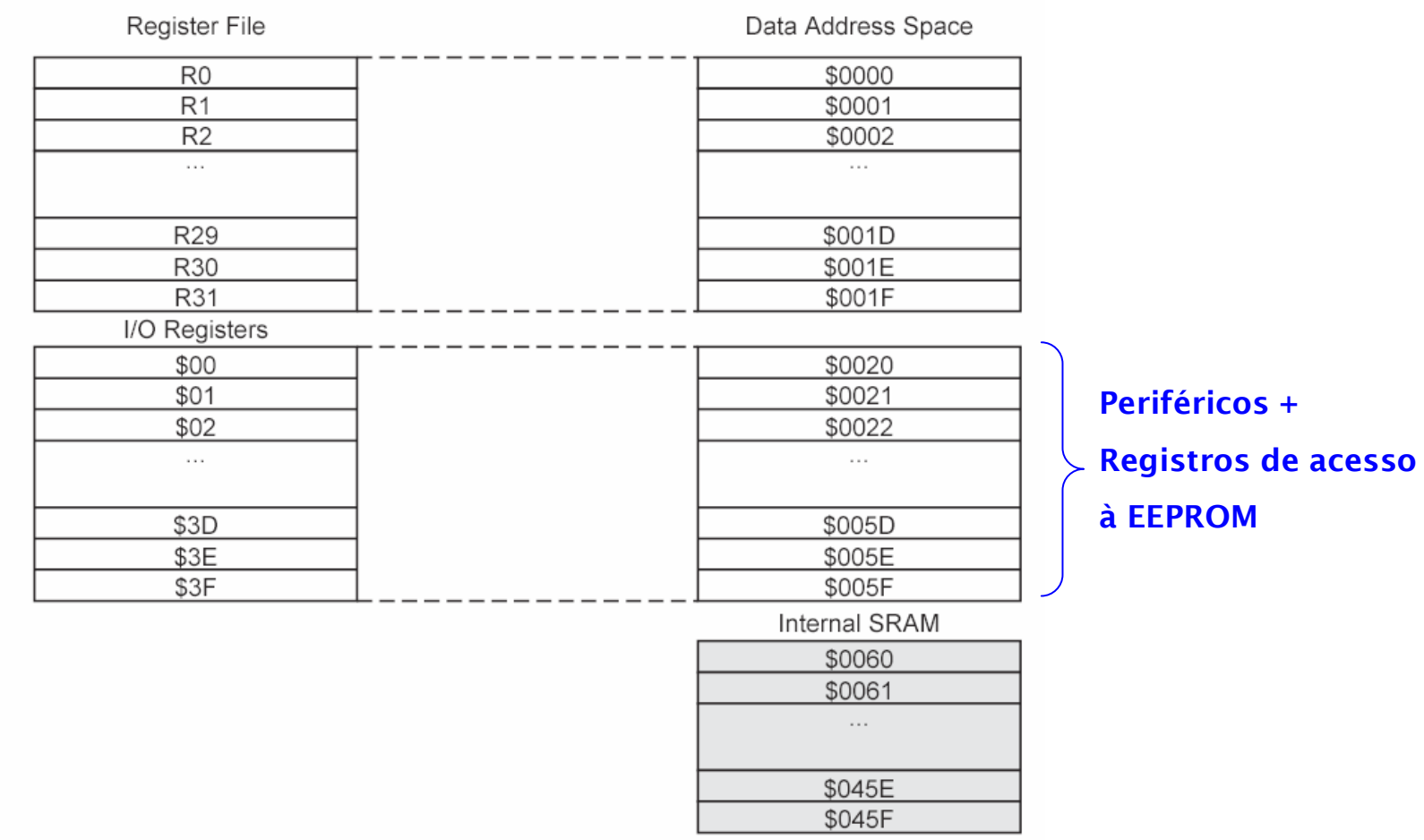
- Core da CPU
  - Pilha
    - Formato decrescente:
      - instrução PUSH decrementa em 1 a pilha
      - instrução POP incrementa em 1 a pilha
      - Instruções RET e RETI decrementam em 2 a pilha
    - Registros apontadores da pilha:

Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

    - SPH:SPL deve ser iniciado na RAM acima do endereço 0x060

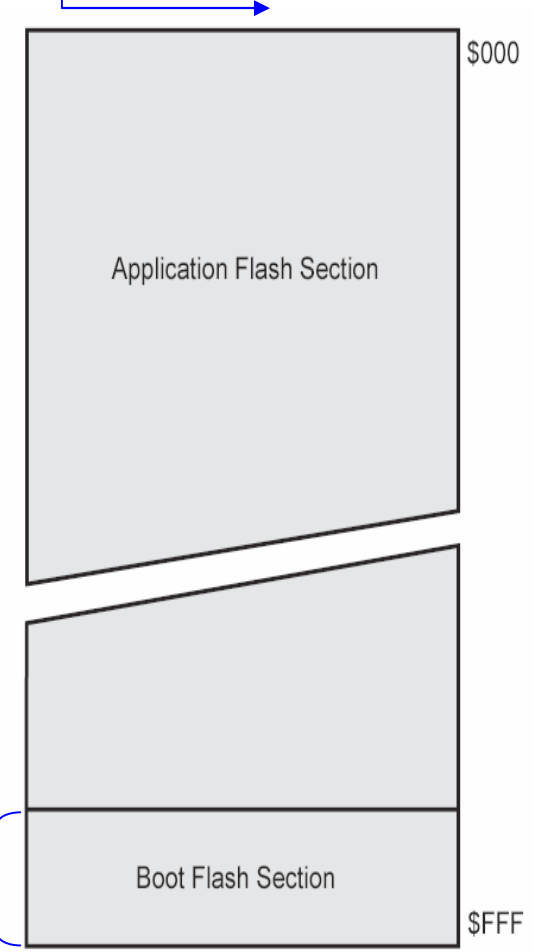
Detalhamento da CPU

- Memória volátil de dados (registros+E/S+RAM)



Detalhamento da CPU

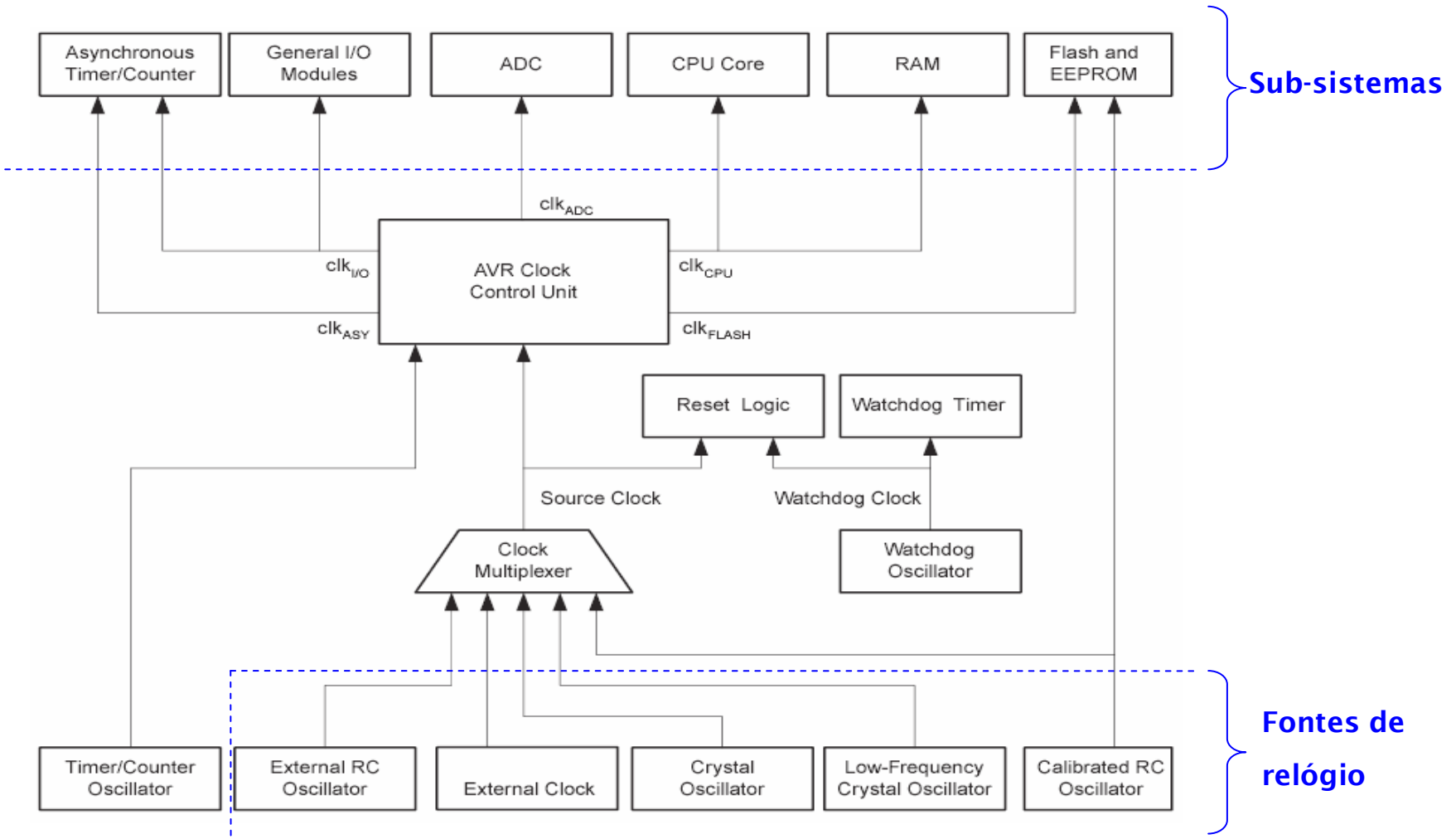
- Memória de programa (FLASH)
  - 8KB organizados em 4K x 16 bits divididos em duas seções (instruções são de 16 ou 32 bits)
  - 10000 ciclos de escrita/apagamento garantidos
  - Acessível por instruções LPM/SPM (SPM é restrito)
- Memória não volátil de dados (EEPROM)
  - 512B organizados em 512 x 8 bits
  - Acesso individual
  - Registros específicos de acesso
  - Escrita em 8,5 ms



Pode usar instruções SPM para programar a seção de aplicação, possibilitando boot remoto.

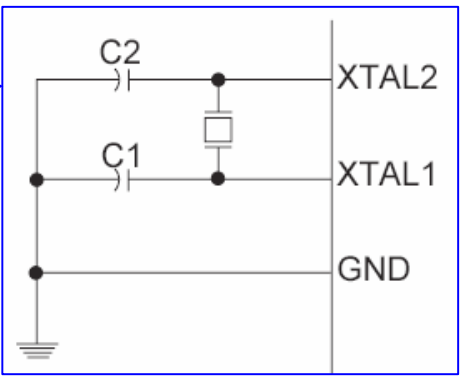
Detalhamento da CPU

- Relógio do sistema

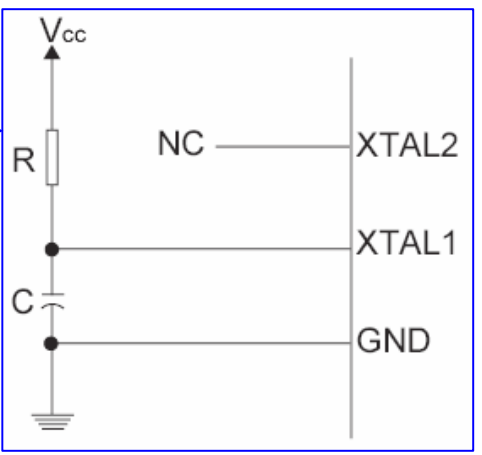


Detalhamento da CPU

- Relógio do sistema
  - Oscilador a cristal de Quartzo/ressonador cerâmico
    - Faixa: alguns KHz - 16MHz
    - Conexão externa:
    - Precisão: muito boa, mas depende do cristal e dos capacitores externos



  - Oscilador a circuito RC externo
    - Faixa: alguns KHz - 12MHz
    - Conexão externa:
    - Freqüência: 1/(3RC)
    - Precisão: depende em grande parte da precisão do componentes externos

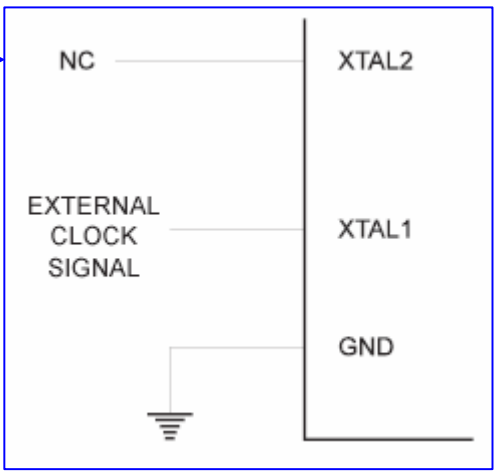


Detalhamento da CPU

- Relógio do sistema
  - Oscilador a circuito RC interno calibrado
    - Freqüências: 1, 2, 4 e 8 MHz
    - Conexão externa: nenhuma
    - Precisão: ±3% do valor nominal a 25 ^{o}C
      ajuste fino pelo registro OSCAL
  - Oscilador a cristal de Quartzo de baixa freqüência
    - Faixa: nominal para 32.768 Hz
    - Conexão externa: similar ao cristal externo
    - Precisão: depende em grande parte da precisão do componentes externos
    - Capacitores externos desnecessários se CKOPT = 0

Detalhamento da CPU

- Relógio do sistema
  - Relógio externo
    - Faixa: DC a 16 MHz
    - Conexão externa:
    - Variação de 2% na freqüência entre dois ciclos consecutivos podem levar a comportamento imprevisível



Detalhamento da CPU

- Relógio do sistema
  - Configuração pelo Fuse Low Byte, acessível apenas pela gravadora.

Fonte de relógio	Freqüência	Bit CKOPT	CKSEL 3	2	1	0
Oscilador externo no pino 9 (XTAL1)	0 - 16 MHz	X	0	0	0	0
Circuito RC interno calibrado	1,0 MHz	X	0	0	0	1
Circuito RC interno calibrado	2,0 MHz	X	0	0	1	0
Circuito RC interno calibrado	4,0 MHz	X	0	0	1	1
Circuito RC interno calibrado	8,0 MHz	X	0	1	0	0
Circuito RC externo	< 0,9 MHz	X	0	1	0	1
Circuito RC externo	0,9 MHz - 3,0 MHz	X	0	1	1	0
Circuito RC externo	3,0 MHz - 8,0 MHz	X	0	1	1	1
Circuito RC externo	8,0 MHz - 12,0 MHz	X	1	0	0	0
Cristal externo (baixa freqüência)	alguns KHz	X	1	0	0	1
Cristal externo (média freqüência)	0,9 MHz - 3,0 MHz	X	1	1	0	1
Cristal externo (alta freqüência)	3,0 MHz - 16,0 MHz	X	1	1	1	1
Ressonador cerâmico externo	0,4 MHz - 0,9 MHz	1	1	0	1	0

Detalhamento da CPU

- Relógio do sistema
  - Observações:
    - Para todas as fontes de relógio, existe um tempo mínimo entre o RESET e o início de operação da CPU (alguns ms)
    - ATmega8 vem de fábrica configurado para relógio RC interno calibrado a 1MHz
    - No modo com cristal de quartzo, a faixa de variação de XTAL2 pode ser configurada conforme CKOPT:
      - CKOPT = 0: XTAL2 vai de 0 a V_{DD} , aumentando a imunidade a ruído externo e permitindo ser usada como fonte de relógio para outros dispositivos. Entretanto, implica em aumento de consumo.
      - CKOPT = 1: XTAL2 varia dentro de uma faixa estreita, reduzindo a imunidade a ruído externo mas também reduzindo o consumo.

Detalhamento da CPU

- Gerenciamento de energia
  - Ativados quando a instrução SLEEP é executada
  - Configuração pelo registro MCU Control Register (MCUCR)

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SE : Habilita modos de economia de energia

SM2..0 : Seleção do modo, de acordo com a tabela abaixo:

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby(1)

Detalhamento da CPU

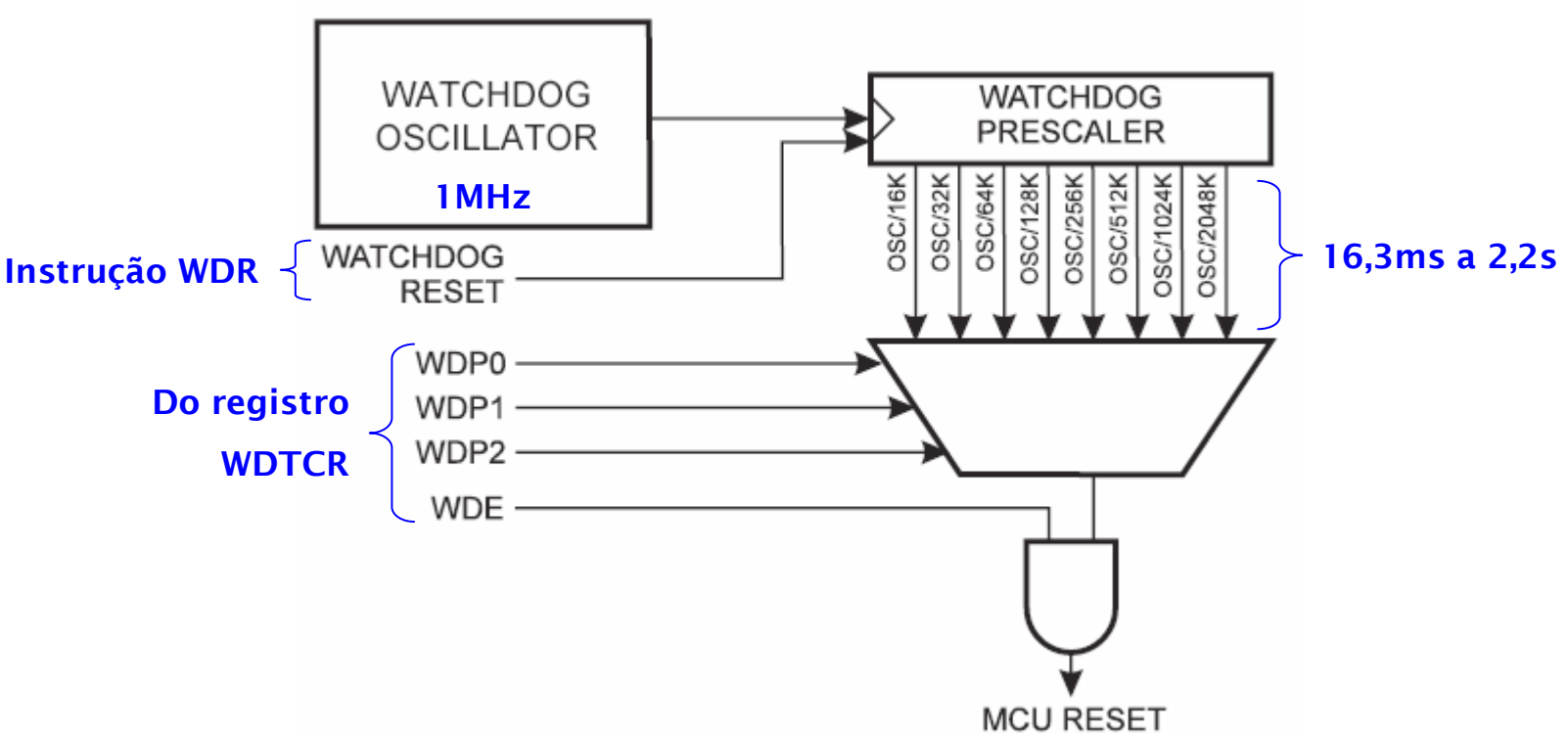
- Gerenciamento de energia
  - Modos de economia de energia:

	Active Clock Domains					Oscillators		Wake-up Sources					
Sleep Mode	**clk_{CPU}**	**clk_{FLASH}**	**clk_{IO}**	**clk_{ADC}**	**clk_{ASY}**	**Main Clock Source Enabled**	**Timer Osc. Enabled**	**INT1 INT0**	**TWI Address Match**	**Timer 2**	**SPM/ EEPROM Ready**	**ADC**	**Other I/O**
Idle			X	X	X	X	X(2)	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X	X	X	
Power Down								X(3)	X				
Power Save					X(2)		X(2)	X(3)	X	X(2)			
Standby(1)						X		X(3)	X				

Notes:
1. External Crystal or resonator selected as clock source.
2. If AS2 bit in ASSR is set.
3. Only level interrupt INT1 and INT0.

Detalhamento da CPU

- Cão-de-guarda
  - Se ativo, reinicia a CPU se seu contador interno estourar.
  - A instrução WDR reinicia o contador.
  - Estrutura interna:



Detalhamento da CPU

- Interrupções
  - Vetores de interrupção e RESET

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow

Detalhamento da CPU

- Interrupções
  - Vetores de interrupção e RESET

Vector No.	Program Address(2)	Source	Interrupt Definition
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	SPM_RDY	Store Program Memory Ready

Detalhamento da CPU

- Interrupções
  - Priorização: prioridade maior quanto menor for o número do vetor de interrupção.
  - Cada locação do vetor deve conter uma instrução RJMP para a função de gerenciamento.
  - Para uma interrupção ser gerada e atendida pela CPU, deve-se ter as três condições abaixo:
    - A ocorrência do evento que gatilha a interrupção, colocando o bit local do registro de flags do periférico em 1 (se houver registro de flags)
    - O bit de habilitação local da interrupção deve estar em 1
    - O bit de I (SREG) de habilitação global deve estar em 1

Detalhamento da CPU

- Interrupções
  - Habilitação global de interrupções
    - Instrução SEI : seta o bit de I de SREG
    - Instrução CLI : reseta o bit de I de SREG
  - Comportamento do bit de SREG quando do atendimento a uma interrupção:
    - I é automaticamente colocado em 0 quando se inicia o atendimento a uma interrupção.
    - I é colocado em 1 quando se encerra o atendimento com a instrução RETI.
    - Para permitir que uma interrupção seja atendida durante o atendimento de uma outra interrupção, o bit I pode ser setado no início da rotina que será interrompida.
  - Tempo de resposta para atendimento: 4 ciclos de relógio (modo normal) a 8 ciclos (modo SLEEP).

Detalhamento da CPU

- Interrupções
  - Locação na FLASH dos vetores de interrupção e RESET

BOOTRST[1]	IVSEL	Reset Address	Interrupt Vectors Start Address
1	0	0x000	0x001
1	1	0x000	Boot Reset Address + 0x001
0	0	Boot Reset Address	0x001
0	1	Boot Reset Address	Boot Reset Address + 0x001

Vetor 1

Vetores 2 - 19

Detalhamento da CPU

- Registros de configuração
  - Acessíveis apenas pela gravadora
  - Lock Bit Byte: registro que limita acesso externo e interno à memória FLASH e à EEPROM.
  - Signature Bytes: três bytes que identificam o fabricante, dispositivo e tamanho da memória FLASH
  - Calibration Byte: contém 4 bytes de calibração do circuito oscilador RC interno, correspondentes às freqüências nominais de 1, 2, 4 e 8MHz. O valor de calibração para 1MHz é carregado automaticamente no registro OSCCAL logo após RESET. Para as outras freqüências, OSCCAL deve ser alterado pelo programa.

Detalhamento da CPU

- Registros de configuração
  - Fuse High Byte (default 11011001): controle sobre
    - (bit 7) RSTDISBL: Ativação do RESET externo
    - (bit 6) WDTON: Cão-de-guarda
    - (bit 5) SPIEN: Programação em modo serial (o mesmo da gravadora BSD)
    - (bit 4) CKOPT: Opções dos circuitos osciladores para relógio do sistema
    - (bit 3) EESAVE: Memória EEPROM preservada durante operação de apagamento na programação.
    - (bit 2..1) BOOTSZ1..0: Tamanho da seção de boot da Flash
    - (bit 0) BOOTRST: (1) Vetor de Reset na Flash ou (0) para BLS.

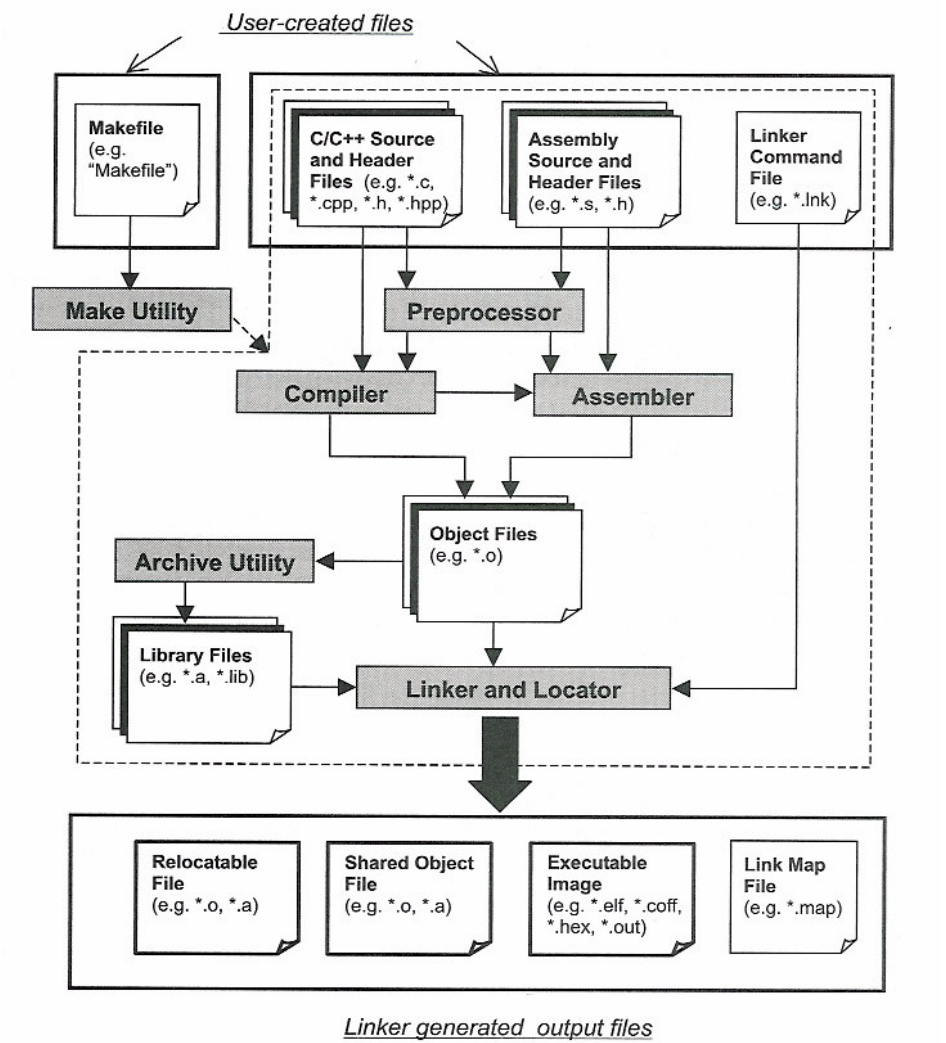
Detalhamento da CPU

- Registros de configuração
  - Fuse Low Byte: controle sobre pinos de E/S, duração da inicialização e relógios do sistema.

Fuse Low Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown out detector trigger level	1 (unprogrammed)
BODEN	6	Brown out detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed)(1)
SUT0	4	Select start-up time	0 (programmed)(1)
CKSEL3	3	Select Clock source	0 (programmed)(2)
CKSEL2	2	Select Clock source	0 (programmed)(2)
CKSEL1	1	Select Clock source	0 (programmed)(2)
CKSEL0	0	Select Clock source	1 (unprogrammed)(2)

Desenvolvimento de software

- Obtenção de imagens [Li & Yao, 2003]



Desenvolvimento de software

- Componentes de um projeto de software em C:
  - Arquivos fonte (.c/.cpp)
  - Arquivos cabeçalho
  - Arquivo makefile
- Projetos com um único arquivo fonte:
  - Indicados para pequenos programas
  - Requerem clareza na escrita e comentários sobre as funções
- Projetos com vários arquivos fonte:
  - Geralmente refletem uma construção modular e mais elegante da resolução do problema
  - Cada arquivo fonte deve ter um arquivo cabeçalho correspondente (boa prática), em que são definidas as funções acessíveis por outros módulos

Desenvolvimento de software

- Exemplo: Projeto com um único arquivo fonte.
  - Arquivo main.c: (1/2)

```
/*****************************************************************************
**** Arquivo: main.c
**** Projeto: testepb0
**** Autores: Geovany A. Borges
**** Ultimas alterações:
        - 03/06/2006 (G.A. Borges): primeira versão.
**** Resumo: exemplo de programa principal.
**** Especificidades do microcontrolador ATMEGA8:
****    - Habilitar relógio por cristal externo a 16MHz: LFUSE = 0xEF.
****    - Colocar um cristal de 16MHz.
*****************************************************************************/
#include <avr/io.h>

// Macros:
#define INVERTE_PB0() PORTB ^= 0x01

// Protótipos de funções locais:
void inicializacao(void);

// Variáveis globais:
int x;
```

Desenvolvimento de software

- Exemplo: Projeto com um único arquivo fonte.
  - Arquivo main.c: (2/2)

```
/*****************************************************************************
// Função principal
*****************************************************************************/
void main (void)
{
    inicializacao();
    while(1) INVERTE_PB0(); // inverte PB0
}

/*****************************************************************************
// Função de inicialização dos periféricos
*****************************************************************************/
void inicializacao(void)
{
    // Coloca PB0 como saída
    DDRB = _BV(DDB0);
}
```

Desenvolvimento de software

- Exemplo: influência do estilo de programação na compreensibilidade (1/2).

```
void PntUpdateOpPos()
{
unsigned long corbeille ;
Reel
     estOpPos[NO] ; // à jeter
int  i ;            // indice de boucle
//------------------
EstGetOpPos( &corbeille , _OpDeltaPos , estOpPos ) ;

// Cumul obligatoirement dans Update pour intégrer si pas de Get à chaque échantillon.
for ( i = NO ; i-- ; ) _OpPos[i] += _OpDeltaPos[i] ;

#if 0 // 0 le 7-3-1 remonter le 2kPi + haut chez l'utilisateur ?
// Saturation. Le faire dans Update est +,simple et plus précis.
if ( _OpPos[Theta] < 0.f ) _OpPos[Theta] += DeuxPif ;
else if ( _OpPos[Theta] >= DeuxPif ) _OpPos[Theta] -= DeuxPif ;
#endif
} // PntUpdateOpPos
```

Desenvolvimento de software

- Exemplo: influência do estilo de programação na compreensibilidade (2/2).

```
// Init de la table de transcodage de la SBPA en binaire.
sbpa = 0x3FFF ; // valeur initiale 14 bits tous à 1 , sera la valeur 0
for ( i = 0 ; i < TailleLFSR ; i++ )
{
    // Ranger
    _LFSR[sbpa ^ 0x3FFF ] = i ;
    // Suivant avec ou exclusif avec les bits 0 2 4 13
#if 0
    bit = sbpa ;       // bit 13 en 13
    aux = sbpa << 9 ; // le bit 4 vient en 13
    bit ^= aux ;       // bit 13 ^ 4
    aux <<= 2 ;        // 2 vient en 13
    bit ^= aux ;       // 13 ^ 4 ^ 2 c'est associatif
    aux <<= 2 ;        // 0 vient en 13
    bit ^= aux ;       // 13 ^ 4 ^ 2 ^ 0
    sbpa = ( bit & 0x2000 ) | ( sbpa >> 1 ) ;
#else
    bit = sbpa ;       // bit 0 en 0
    aux = sbpa >> 2 ; // le bit 2 vient en 0
    bit ^= aux ;       // bit 2 ^ 0
    aux >>= 2 ;        // 4 vient en 0
    bit ^= aux ;       // 4 ^ 2 ^ 0 c'est associatif
    aux >>= 9 ;        // 13 vient en 0
    bit ^= aux ;       // 13 ^ 4 ^ 2 ^ 0
    sbpa = ( ( sbpa << 1 ) | ( bit & 0x1 ) ) & 0x3FFF ;
#endif
} // for i
```

Desenvolvimento de software

- Procedimentos para programação modular em C
  - Identificação dos módulos de um projeto
  - Cada módulo deve ser auto-suficiente
  - Definição das funcionalidades e dependências
  - Definição de rotinas de inicialização, interface, internas e de encerramento dos módulos
  - Cada módulo possui um arquivo fonte e um arquivo cabeçalho
  - Implementar módulos reutilizáveis
  - Realizar documentação, ao menos no código durante sua concepção

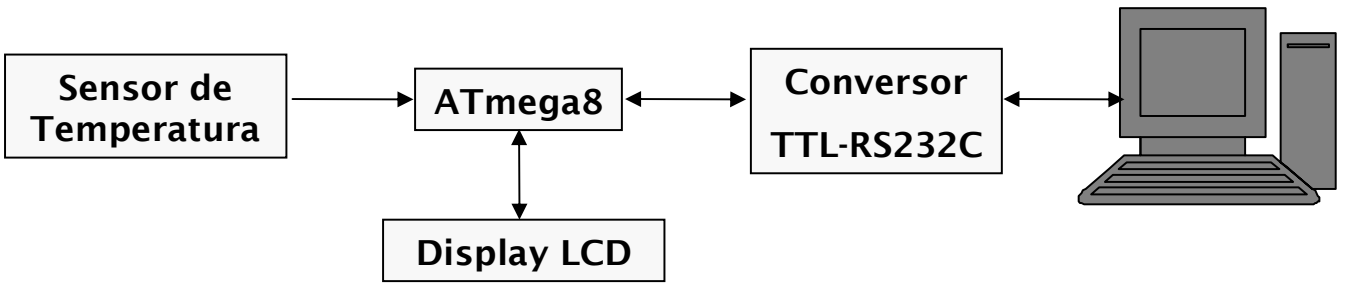
Desenvolvimento de software

- Exemplo:

  Sistema de aquisição de dados de sensor de temperatura com visualização em display LCD alfanumérico e comunicação RS232 com microcomputador. No microcomputador, um software residente solicita de forma assíncrona a última medição de temperatura.

Desenvolvimento de software

- Exemplo:
  - Realização física:



  - Identificação dos módulos:
    - Módulo principal
    - Módulo sensor
    - Módulo lcd
    - Módulo serial

Desenvolvimento de software

- Exemplo:
  - Módulo principal:
    - Arquivos: main.c
    - Funcionalidades:
      - Inicialização do sistema
      - Laço infinito em que realiza comunicação serial
      - Interrupção periódica (T = 5ms)
    - Dependências: sensor, lcd e serial

Desenvolvimento de software

- Exemplo:
  - Módulo principal:
    - Implementação

```
// arquivo main.c
#include “sensor.h”
#include “lcd.h”
#include “serial.h”
volatile double Temp;
volatile int Counter5msTicks=0;
void main(void)
{
    init();
    while(1){
          for(Counter5msTicks=0;
Counter5msTicks<4;);
          serial_dispatch(Temp);
          lcd_printf(“\rTemp = %f Celsius”, Temp);
    }
}
```

Desenvolvimento de software

- Exemplo:
  - Módulo principal:
    - Implementação

void init(void)	

Chamadas externas:	`serial_init(SERIAL_9600),`
	`lcd_init();`
	`sensor_init();`

Funcionalidades locais:	Inicialização do temporizador

SIGNAL (SIG_OUTPUT_COMPARE1A)	
Chamadas externas:	`Temp = sensor_read();`
Funcionalidades locais:	`++Counter5msTicks;`

Desenvolvimento de software

- Exemplo:
  - Módulo serial:
    - Implementação

```
// arquivo serial.c
#include “serial.h”

// protótipos locais
char serial_receive(char *c, timeout_ms);
void serial_send_double(double valor);

// funções de interface:
void serial_init(int baudrate){...}
void serial_dispatch(double Temp){...}

// apenas funções locais:
char serial_receive(char *c, timeout_ms){...}
void serial_send_double(double valor){...}
```

```
// arquivo serial.h
// defines de interface
#define SERIAL_19200          2
#define SERIAL_9600           1
#define SERIAL_4800           0

// protótipos de interface
void serial_init(int baudrate);
void serial_dispatch(double Temp);
```

Desenvolvimento de software

- Exemplo:
  - Módulo serial:
    - Implementação

void serial_init(int baudrate)	
Chamadas externas:	
Funcionalidades locais:	Inicialização da USART Taxa dada por baudrate 8 bits, sem paridade, 1 stop

void serial_dispatch(double Temp)	
Chamadas externas:	

Funcionalidades locais:	`status = serial_receive(&d,1);`
	`se status = 0, retornar`
	`se d = 'T', executar serial_send_double(Temp)`

char serial_receive(char *c, timeout_ms){...}	
Chamadas externas:	
Funcionalidades locais:	`se não chegar nada pela USART em timeout_ms, retorna 0` `se chegar algo, escrever em *c e retorna 1`

Desenvolvimento de software

- Toolchain avr-gcc: GCC significa
  - avr-gcc/avr-g++: compiladores C/C++
  - avr-as: assembler
  - avr-gdb: depurador
  - avr-ld: linker
  - binutils: utilitários que manipulam arquivos binários
    - avr-adr2line: dado um endereço e um executável, usa informação de depuração para determinar qual linha/arquivo fonte se encontra o código que gerou as instruções do executável;
    - avr-ar: manipulador de arquivos, que altera, retira e coloca partes de código objeto;
    - avr-nm: lista símbolos contidos em um arquivo objeto;
    - avr-objcopy: copia partes de um arquivo objeto em outro;
    - avr-objdump: mostra informações sobre um ou mais arquivos objeto;

Desenvolvimento de software

- avr-ranlib: gera um índice do conteúdo de um arquivo objeto e armazena o índice nele mesmo. Isto orna mais rápida a tarefa do linker;
- avr-readelf: mostra informações de um arquivo .elf (*executable and linking format*);
- avr-size: mostra o tamanho das seções e o tamanho total do código binário contido em um arquivo objeto;
- avr-strings: para um arquivo de entrada, imprime toda seqüência de caracteres de comprimento maior ou igual a 4;
- avr-strip: recria um arquivo objeto sem os símbolos;

- make: automação do processo de compilação
- Informações na internet:
  - http://www.gnu.org/manual/manual.html

Desenvolvimento de software

- Procedimentos de construção da imagem
  - Construção em linha de comando
    - Formado da chamada de avr-gcc:

```
avr-gcc [options] file
```

opções normalmente usadas:

Opções do compilador:

`-c`	Compilar apenas, gerando arquivo objeto alocável
`-o main.o`	Define o nome do arquivo final da compilação.
`-lm`	Quando usado como linker, inclui biblioteca libm.a (matemática). Qualquer biblioteca compilada se apresenta sob a forma limnome.a, e sua inclusão no projeto é feita com -lnome. -Ldir especifica diretório onde as bibliotecas devem ser buscadas
`-On`	Otimização, em que n=0,1,2,3 ou s é o nível. -00 é sem otimização. -01 é otimização com melhor compromisso entre velocidade e tamanho de código. -O2 implica em maior otimização de velocidade sem aumentar muito o tamanho. -O3 é otimização com uso de funções inline. -Os implica em otimização de tamanho de código.
`-Wa,options`	Passa as opções `options` para o assembler (`avr-as`).
`-g`	Inclui símbolos de depuração no código, deixando-o muito maior, mas permitindo depuração.

Desenvolvimento de software

- Procedimentos de construção da imagem
  - Construção em linha de comando
    - Formado da chamada de avr-gcc:

```
avr-gcc [options] file
```

opções normalmente usadas:

Opções específicas para processadores AVR, obtidas por avr-gcc --target-help:

`-mint8`	Assume o tipo int de 8 bits
`-mmcu=atmega8`	Processador ATmega8.
`-minit-stack=nnnn`	O endereço `nnnn` é assumido como topo da pilha inicial

Maiores detalhes, consultar a documentação do WinAVR:
WinAVR/doc/avr-libc/avr-libc-user-manual/index.html

Desenvolvimento de software

- Procedimentos de construção da imagem
  - Construção em linha de comando
    - Versão do compilador:

```
avr-gcc --version
```

    - Compilação

```
avr-gcc -c -mmcu=atmega8 -I. -O0 -
funsigned-char -funsigned-bitfields -fpack-
struct -fshort-enums -Wall -Wstrict-
prototypes -Wa,-adhlns=main.lst  -std=gnu99
main.c -o main.o
```

      O arquivo main.lst contém a listagem em assembly
    - Linker: obtenção do arquivo main.elf (executable and linking format)
    - Geração do arquivo imagem: main.hex (intel hex)

Desenvolvimento de software

- Automação dos procedimentos
  - Com o aumento de complexidade dos projetos, o procedimento de compilação precisa ser automatizado. Em muitos casos, obter um outro formato de imagem pode implicar em muitas mudanças no processo de construção.
  - Utilitátio make.exe: é um utilitário de automação do processo de compilação. Para tanto, ele faz uso de um arquivo auxiliar (makefile), único para cada projeto, que especifica todo o processo de construção.

Desenvolvimento de software

- Automação dos procedimentos
  - Arquivo makefile de demonstração:
    - Consultar makefile.pdf para sua listagem.
    - Variáveis:
      - MCU: define o microcontrolador (atmega8)
      - FORMAT: define o formato da imagem final (ihex)
      - TARGET: define o nome da imagem (main)
      - OPT: define o nível de otimização (0, 1, 2, 3 ou s)
      - SRC: deve conter a listagem de todos os arquivos fonte
      - CFLAGS: Flags do compilador avr-gcc.
      - AFLAGS: Flags do assembler avr-as.
      - LDFLAGS: Flags do linker avr-ld
      - e várias outras...

Desenvolvimento de software

- Automação dos procedimentos
  - Arquivo makefile de demonstração:
    - Forma de chamada:

      `make rule`

      em que rule é a regra usada. Para construir um projeto por completo, deve-se digitar `make all`.
    - Formato de uma regra:

      `target` `...` `:` *`prerequisites`* `...`

      `command`

      `...`

      em que `target` é o nome da regra (alvo da execução dos comandos), que nem sempre são arquivos. Se não é um arquivo, chama-se de `PHONY target`. `prerequisites` são as regras ou arquivos que devem estar disponíveis para a execução da regra. Se for uma regra, elas será executada antes da atual. Se for um arquivo, make procura pela regra capaz de criá-lo. Se não existir, procura por um arquivo no diretório. `command` são os comandos executados por aquela regra.

Desenvolvimento de software

- Automação dos procedimentos
  - Arquivo makefile de demonstração:
    - Exemplos de regras
      - Sem comandos (exemplo de PHONY target):

```
all: begin gccversion sizebefore $(TARGET).elf $(TARGET).hex $(TARGET).eep \
   $(TARGET).lss $(TARGET).sym sizeafter finished end
```

      - Compilar apenas arquivos .c:

```
%.o : %.c
   @echo
   @echo $(MSG_COMPILING) $<
   $(CC) -c $(ALL_CFLAGS) $< -o $@
```

    - Símbolos especiais: nem sempre alvos e pré-requisitos são fixos, sendo necessário usar as instâncias das regras.
      - $< : primeiro pré-requisito da regra
      - $@ : alvo da regra
      - $(VARIABLE) : conteúdo da variável VARIABLE.

Desenvolvimento de software

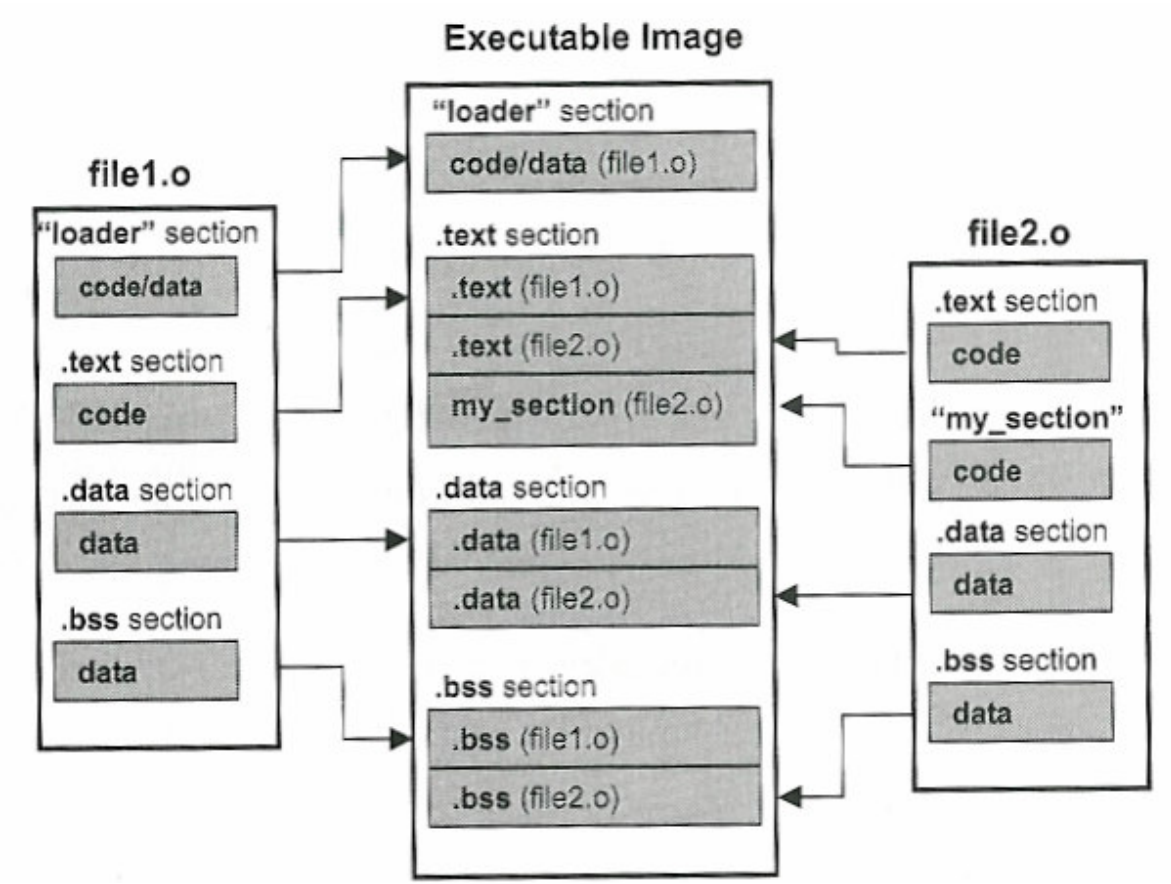
- Automação dos procedimentos
  - Arquivo makefile de demonstração:
    - Principais arquivos resultantes de `make all`:
      - main.lss: arquivo de listagem assembly, com detalhes de conversão de código. Se a opção -g estiver ativa em CFLAGS, então se terá também o código C correspondente.
      - main.sym: listagem de todos os símbolos do alvo main.elf
      - main.o: arquivo objeto resultado da compilação de main.c
      - main.eep: arquivo ihex a ser gravado na EEPROM
      - main.elf: arquivo binário ELF
      - main.d: arquivo de dependências
    - Limpeza do projeto: `make clean`

Desenvolvimento de software

- Seções de memória
  - Durante todo o processo de construção, o código, variáveis e constantes são alocadas em segmentos denominados de seções.
  - Seções comuns no AVR:
    - .text: seção com trechos de programa
    - .data: seção com constantes, usadas por variáveis inicializadas no código.
    - .bss: seção com variáveis globais ou variáveis estáticas, que não possuem valor inicial nos programa. As variáveis da seção .bss recebem 0 durante o processo de inicialização da imagem.
    - .noinit: o mesmo que .bss, mas as variáveis não serão afetadas durante a inicialização.
    - .eeprom: seção com variáveis para a EEPROM

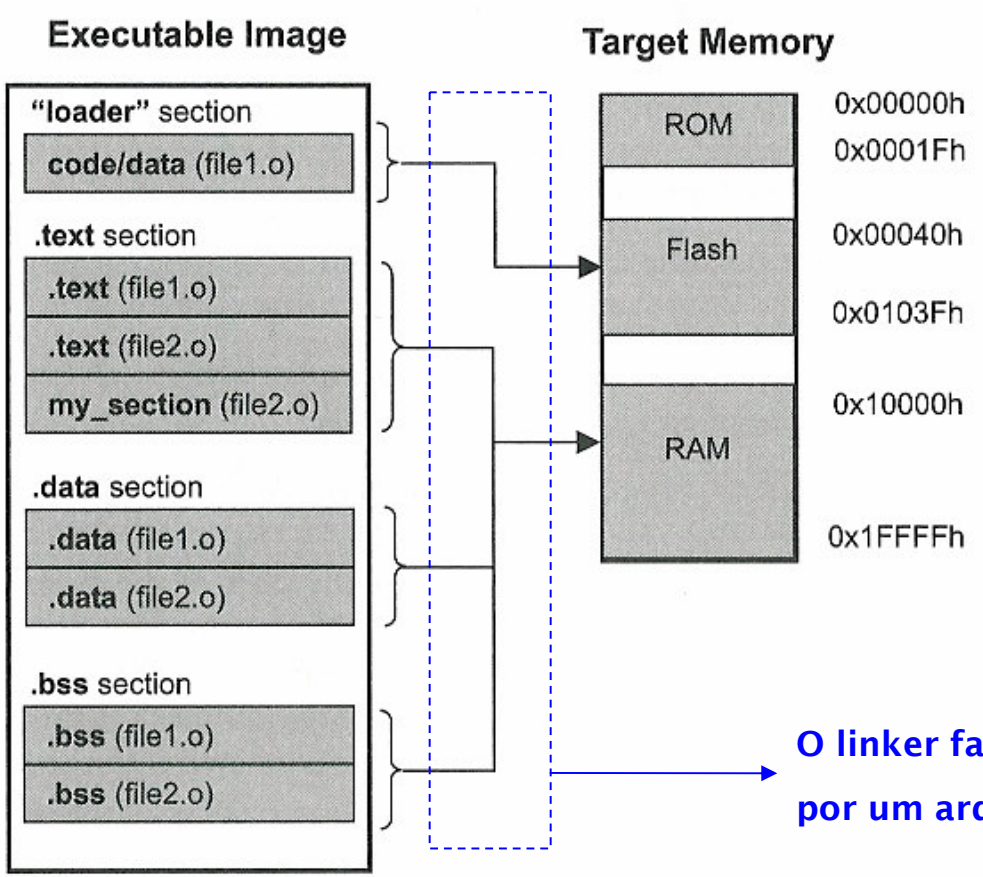
Desenvolvimento de software

- Seções de memória
  - Constituição do objeto alvo [Li & Yao, 2003]



Desenvolvimento de software

- Seções de memória
  - Mapeamento na memória [Li & Yao, 2003]



O linker faz o mapeamento guiado por um arquivo script.

Desenvolvimento de software

- Seções de memória
  - Conteúdo de um arquivo script do linker
    - Definição de seções: SECTIONS{...}
    - Definição do mapeamento de memória: MEMORY{...}
  - No caso do toolchain avr-gcc, os arquivos script se encontram em \WinAVR\avr\lib\ldscripts
  - Para o ATmega8 (arquitetura avr4, mmcu=atmega8), o arquivo default chama-se avr4.x:

```
MEMORY
{
  text   (rx)   : ORIGIN = 0, LENGTH = 8K
  data   (rw!x) : ORIGIN = 0x800060, LENGTH = 0xffa0
  eeprom (rw!x) : ORIGIN = 0x810000, LENGTH = 64K
}
```

Início da FLASH

Início da SDRAM

    - Observa-se que o tamanho das memória RAM não corresponde ao que se tem com o ATmega8: Cuidado!

Desenvolvimento de software

- Dissecando um projeto:
  - main.c:

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

// Macros:
#define INVERTE_PB5() PORTB ^= (1<<5)

// Protótipos de funções locais:
void inicializacao(void);

// Variáveis globais:
int x;
int y = 10;
char strg[] = {"Ola fantastico mundo dos microcontroladores"};

/*****************************************************************************************
// Função principal
*****************************************************************************************/
void main (void)
{
    int z;

    inicializacao();

    while(1){
        INVERTE_PB5(); // inverte PB5
        z = (x += y);
        y--;
    }
}

/*****************************************************************************************
// Função de inicialização dos periféricos
*****************************************************************************************/
void inicializacao(void)
{
    // Coloca PB5 como saída
    DDRB = (1<<5);
}
```

Desenvolvimento de software

- Dissecando um projeto:
  - Compilando: `make all`
  - Otimização: -O0 (nenhuma)
  - Tamanho das seções: `make sizeafter`

```
Size after:
main.elf  :
section    size      addr
.text       186         0
.data        46   8388704
.bss          2   8388750
.noinit       0   8388752
.eeprom       0   8454144
Total       234
```

    - Tamanho do programa na FLASH: .text + .data
    - Espaço usado pelas variáveis na RAM: .data + .bss

Desenvolvimento de software

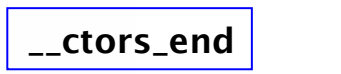
- Dissecando um projeto:
  - Arquivo main.lss:
    - Cabeçalho

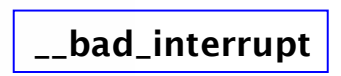
```
main.elf:     file format elf32-avr

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000ba  00000000  00000000  00000094  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         0000002e  00800060  000000ba  0000014e  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000002  0080008e  0080008e  0000017c  2**0
                  ALLOC
  3 .noinit       00000000  00800090  00800090  0000017c  2**0
                  CONTENTS
  4 .eeprom       00000000  00810000  00810000  0000017c  2**0
                  CONTENTS
```

Desenvolvimento de software

- Dissecando um projeto:
  - Arquivo main.lss:
    - Seção .text – vetores de interrupção

```
Disassembly of section .text:

00000000 <__vectors>:
   0:   12 c0           rjmp    .+36        ; 0x26
   2:   2b c0           rjmp    .+86        ; 0x5a
   4:   2a c0           rjmp    .+84        ; 0x5a
   6:   29 c0           rjmp    .+82        ; 0x5a
   8:   28 c0           rjmp    .+80        ; 0x5a
   a:   27 c0           rjmp    .+78        ; 0x5a
   c:   26 c0           rjmp    .+76        ; 0x5a
   e:   25 c0           rjmp    .+74        ; 0x5a
  10:   24 c0           rjmp    .+72        ; 0x5a
  12:   23 c0           rjmp    .+70        ; 0x5a
  14:   22 c0           rjmp    .+68        ; 0x5a
  16:   21 c0           rjmp    .+66        ; 0x5a
  18:   20 c0           rjmp    .+64        ; 0x5a
  1a:   1f c0           rjmp    .+62        ; 0x5a
  1c:   1e c0           rjmp    .+60        ; 0x5a
  1e:   1d c0           rjmp    .+58        ; 0x5a
  20:   1c c0           rjmp    .+56        ; 0x5a
  22:   1b c0           rjmp    .+54        ; 0x5a
  24:   1a c0           rjmp    .+52        ; 0x5a
```





Desenvolvimento de software

- Dissecando um projeto:
  - Arquivo main.lss:
    - Seção .text – seção de reset

```
00000026 <__ctors_end>:
  26:   11 24           eor r1, r1
  28:   1f be           out 0x3f, r1      ; 63
  2a:   cf e5           ldi r28, 0x5F     ; 95
  2c:   d4 e0           ldi r29, 0x04     ; 4
  2e:   de bf           out 0x3e, r29     ; 62
  30:   cd bf           out 0x3d, r28     ; 61
```

SREG = 0

SP = 0x045F, topo da RAM

    - Seção .text – copia valores constantes de variáveis inicializadas

```
00000032 <__do_copy_data>:
  32:   10 e0           ldi r17, 0x00     ; 0
  34:   a0 e6           ldi r26, 0x60     ; 96
  36:   b0 e0           ldi r27, 0x00     ; 0
  38:   ea eb           ldi r30, 0xBA     ; 186
  3a:   f0 e0           ldi r31, 0x00     ; 0
  3c:   02 c0           rjmp    .+4           ; 0x42

0000003e <.do_copy_data_loop>:
  3e:   05 90           lpm r0, Z+
  40:   0d 92           st  X+, r0

00000042 <.do_copy_data_start>:
  42:   ae 38           cpi r26, 0x8E     ; 142
  44:   b1 07           cpc r27, r17
  46:   d9 f7           brne    .-10          ; 0x3e
```

X-Register = 0x0060 (RAM)

Z-Register = 0x00BA (FLASH, após final do programa)

Procedimento de cópia

Desenvolvimento de software

- Dissecando um projeto:
  - Arquivo main.lss:
    - Seção .text – inicializa com 0 as variáveis não inicializadas explicitamente no programa fonte (seção .bss)

```
00000048 <__do_clear_bss>:
  48:   10 e0           ldi r17, 0x00   ; 0
  4a:   ae e8           ldi r26, 0x8E   ; 142
  4c:   b0 e0           ldi r27, 0x00   ; 0
  4e:   01 c0           rjmp    .+2             ; 0x52

00000050 <.do_clear_bss_loop>:
  50:   1d 92           st  X+, r1

00000052 <.do_clear_bss_start>:
  52:   a0 39           cpi r26, 0x90   ; 144
  54:   b1 07           cpc r27, r17
  56:   e1 f7           brne    .-8             ; 0x50
  58:   01 c0           rjmp    .+2             ; 0x5c
```

X-Register = 0x008E (início da .bss na RAM)

Procedimento escrita de 0 nas variáveis não inicializadas (np caso x)

    - Seção .text – rotina default de tratamento de interrupções

```
0000005a <__bad_interrupt>:
  5a:   d2 cf           rjmp    .-92            ; 0x0
```

Vetor de RESET, ou seja RESET por software.

Desenvolvimento de software

- Dissecando um projeto:
  - Arquivo main.lss:
    - Seção .text – função main()

```
0000005c <main>:
  5c:   cd e5           ldi r28, 0x5D   ; 93
  5e:   d4 e0           ldi r29, 0x04   ; 4
  60:   de bf           out 0x3e, r29   ; 62
  62:   cd bf           out 0x3d, r28   ; 61
  64:   20 d0           rcall   .+64          ; 0xa6
  66:   90 91 38 00     lds r25, 0x0038
  6a:   80 e2           ldi r24, 0x20   ; 32
  6c:   89 27           eor r24, r25
  6e:   80 93 38 00     sts 0x0038, r24
  72:   20 91 8e 00     lds r18, 0x008E
  76:   30 91 8f 00     lds r19, 0x008F
  7a:   80 91 60 00     lds r24, 0x0060
  7e:   90 91 61 00     lds r25, 0x0061
  82:   82 0f           add r24, r18
  84:   93 1f           adc r25, r19
  86:   90 93 8f 00     sts 0x008F, r25
  8a:   80 93 8e 00     sts 0x008E, r24
  8e:   89 83           std Y+1, r24    ; 0x01
  90:   9a 83           std Y+2, r25    ; 0x02
  92:   80 91 60 00     lds r24, 0x0060
  96:   90 91 61 00     lds r25, 0x0061
  9a:   01 97           sbiw    r24, 0x01   ; 1
  9c:   90 93 61 00     sts 0x0061, r25
  a0:   80 93 60 00     sts 0x0060, r24
  a4:   e0 cf           rjmp    .-64          ; 0x66
```

SP = 0x045D, -2 do topo da RAM

Chamada de inicializacao()

Laço principal

Desenvolvimento de software

- Dissecando um projeto:
  - Arquivo main.lss:
    - Seção .text – função inicializacao()

```
000000a6 <inicializacao>:
  a6:   cf 93           push    r28
  a8:   df 93           push    r29
  aa:   cd b7           in  r28, 0x3d   ; 61
  ac:   de b7           in  r29, 0x3e   ; 62
  ae:   80 e2           ldi r24, 0x20   ; 32
  b0:   80 93 37 00     sts 0x0037, r24
  b4:   df 91           pop r29
  b6:   cf 91           pop r28
  b8:   08 95           ret
```

Salva Y-Register na pilha

Y-Register = endereço do topo da pilha

DDRB (mapeado em RAM) = 0x20

Restaura Y-Register da pilha

Aparentemene não há motivo para salvar Y-Register na pilha!

Desenvolvimento de software

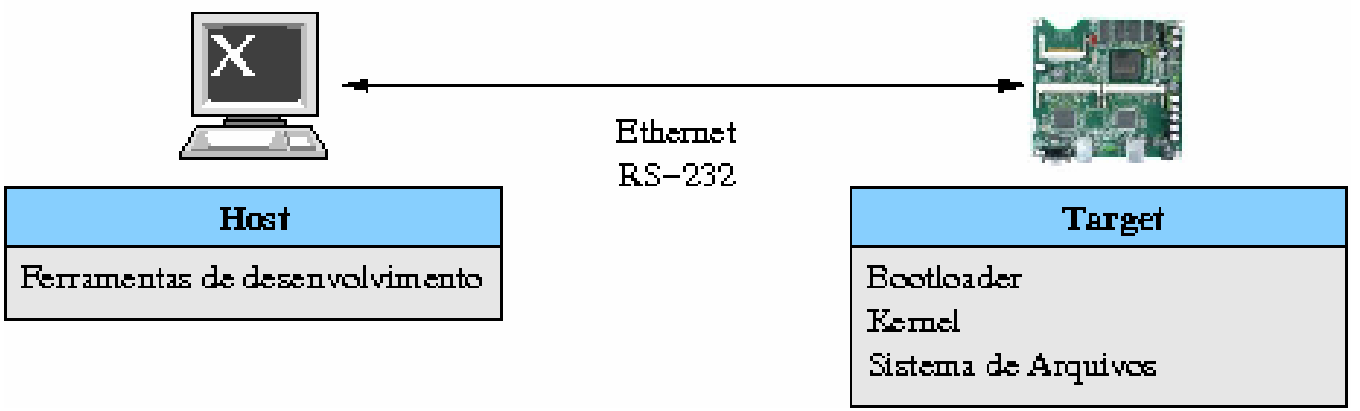
- Dissecando um projeto:
  - Arquivo main.lss: recompilado com otimização -O2
    - Tamanho antes: 234 bytes (.text + .data + .bss)
    - Tamanho depois: 166 bytes (.text + .data + .bss)
    - Única (e enorme) diferença:

```
0000005c <inicializacao>:
  5c:   80 e2           ldi r24, 0x20   ; 32
  5e:   87 bb           out 0x17, r24   ; 23
  60:   08 95           ret

00000062 <main>:
  62:   cf e5           ldi r28, 0x5F   ; 95
  64:   d4 e0           ldi r29, 0x04   ; 4
  66:   de bf           out 0x3e, r29   ; 62
  68:   cd bf           out 0x3d, r28   ; 61
  6a:   f8 df           rcall   .-16          ; 0x5c
  6c:   90 e2           ldi r25, 0x20   ; 32
  6e:   88 b3           in  r24, 0x18   ; 24
  70:   89 27           eor r24, r25
  72:   88 bb           out 0x18, r24   ; 24
  74:   fc cf           rjmp    .-8           ; 0x6e
```

  - Arquivo main.lss: recompilado com otimização -Os
    - Mesmo resultado que compilação -O2

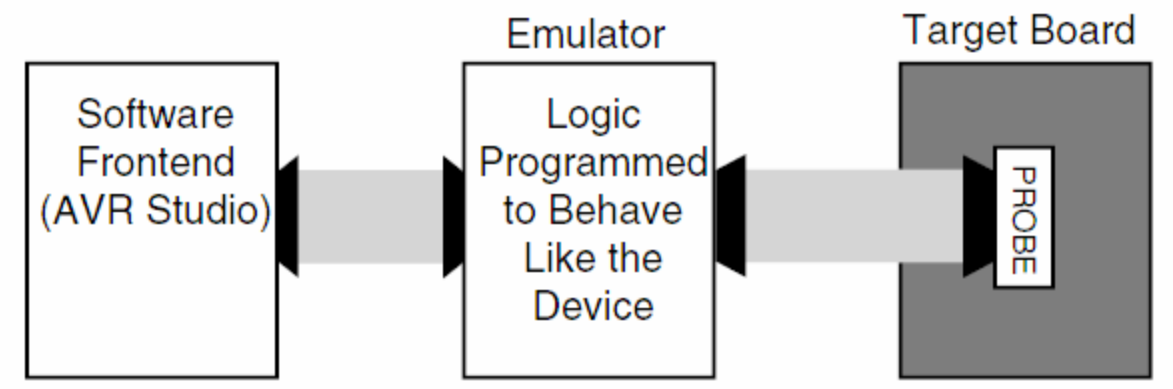
Desenvolvimento de software

- Depuração de software
  - Problemas da depuração de software em desenvolvimento cruzado:
    - Observação tempo-real de eventos
    - Sincronismo
    - Passo-a-passo



Desenvolvimento de software

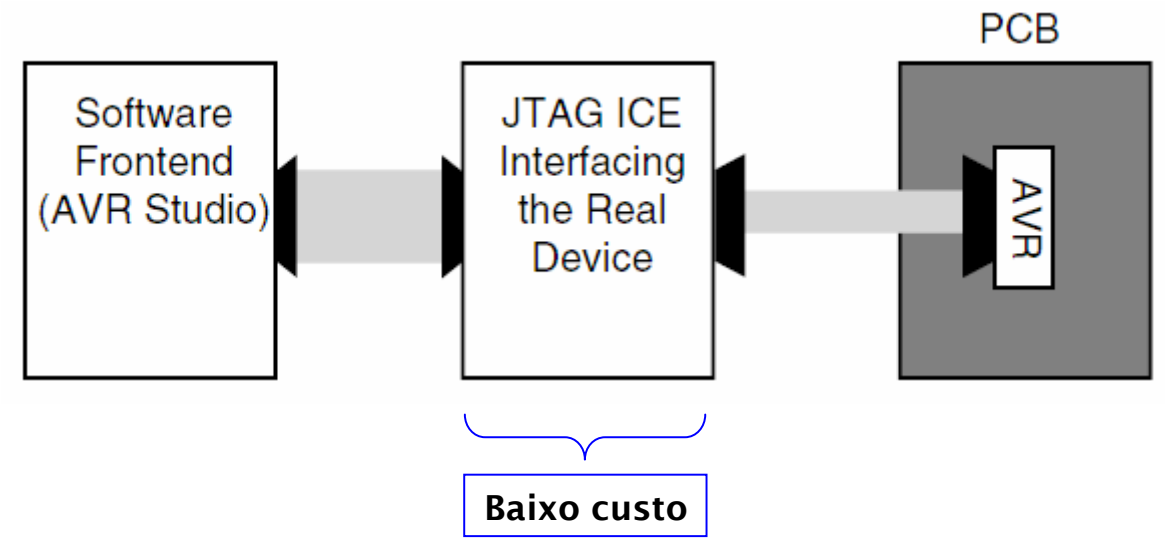
- Depuração de software
  - Abordagens por hardware:
    - Usando emuladores de hardware



      - Emuladores com processador
      - Emuladores sem processador

Desenvolvimento de software

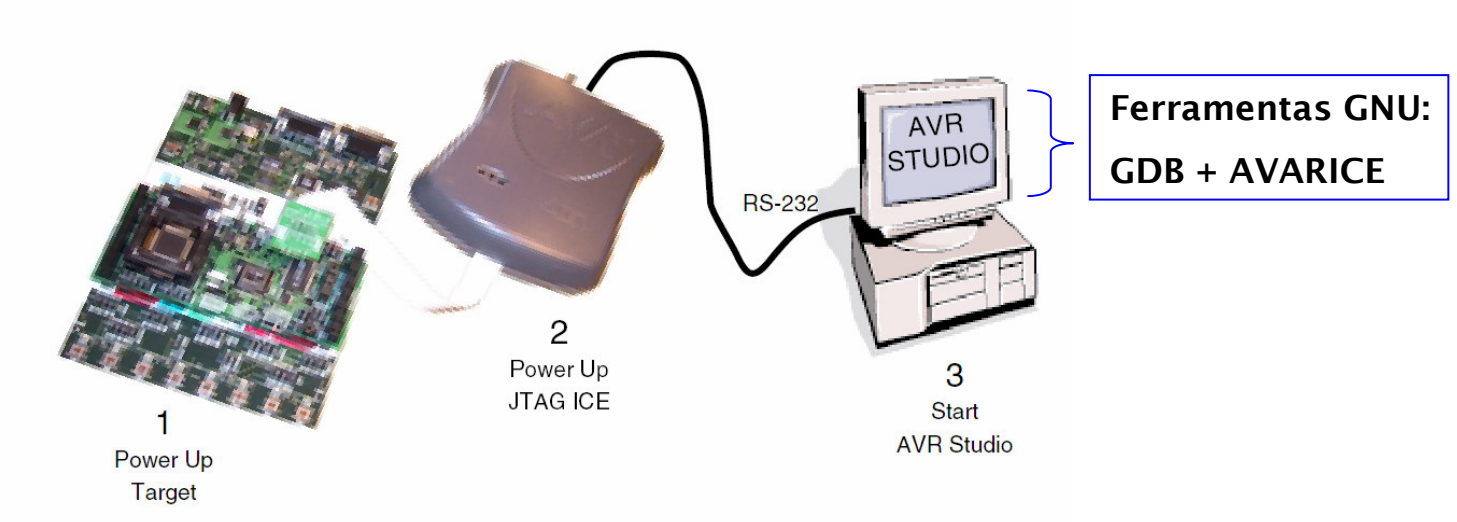
- Depuração de software
  - Abordagens por hardware:
    - Usando depuradores no chip (e.g., JTAG)



Desenvolvimento de software

- Depuração de software
  - Abordagens por hardware:
    - Usando depuradores no chip (e.g., JTAG)

      Depurador JTAG da ATMEL (acima ATmega16):



      Funcionalidades: Run Mode, Stopped Mode, Breakpoints, Passo-a-passo, visualização de registros

Desenvolvimento de software

- Depuração de software
  - Abordagens por software:
    - Depuradores (e.g., GDB)
    - Simuladores (e.g., AVRStudio)
    - Sinalização de eventos (Run mode apenas):
      - LEDs
      - Portas de E/S + Osciloscópio
      - Buzzer
      - printf direcionado para porta serial ou display LCD

Desenvolvimento de software

- Depuração de software
  - Exemplo de sinalização de eventos: printf

```
// Macros:
#define USE_DEBUG   1
#define DEBUG_POINT(m) if(USE_DEBUG) {printf("\n%i",m);}

void main (void)
{
    DEBUG_POINT(1);
    funcao_a();
    DEBUG_POINT(2);
    funcao_b();
    DEBUG_POINT(3);
}

void funcao_a(void)
{
    DEBUG_POINT(11);
}

void funcao_b(void)
{
    DEBUG_POINT(21);
}
```

Desenvolvimento de software

- Depuração de software
  - Exemplo de sinalização de eventos: USE_DEBUG=1, OPT=s

```
0000005c <funcao_a>:
  5c:   8b e0           ldi r24, 0x0B   ; 11
  5e:   90 e0           ldi r25, 0x00   ; 0
  60:   9f 93           push    r25
  62:   8f 93           push    r24
  64:   80 e6           ldi r24, 0x60   ; 96
  66:   90 e0           ldi r25, 0x00   ; 0
  68:   9f 93           push    r25
  6a:   8f 93           push    r24
  6c:   3d d0           rcall   .+122       ; 0xe8
  6e:   0f 90           pop r0
  70:   0f 90           pop r0
  72:   0f 90           pop r0
  74:   0f 90           pop r0
  76:   08 95           ret

00000078 <funcao_b>:
  78:   85 e1           ldi r24, 0x15   ; 21
  7a:   90 e0           ldi r25, 0x00   ; 0
  7c:   9f 93           push    r25
  7e:   8f 93           push    r24
  80:   80 e6           ldi r24, 0x60   ; 96
  82:   90 e0           ldi r25, 0x00   ; 0
  84:   9f 93           push    r25
  86:   8f 93           push    r24
  88:   2f d0           rcall   .+94        ; 0xe8
  8a:   0f 90           pop r0
  8c:   0f 90           pop r0
  8e:   0f 90           pop r0
  90:   0f 90           pop r0
  92:   08 95           ret
```

Coloca 0x0011 na pilha

Onde está o if ???

Chamada a printf

Coloca 0x0015 na pilha

Chamada a printf

Tamanho do código: 2720 bytes

Desenvolvimento de software

- Depuração de software
  - Exemplo de sinalização de eventos: USE_DEBUG=0, OPT=s

```
0000005c <funcao_a>:
  5c:   08 95           ret

0000005e <funcao_b>:
  5e:   08 95           ret

00000060 <main>:
  60:   cf e5           ldi r28, 0x5F   ; 95
  62:   d4 e0           ldi r29, 0x04   ; 4
  64:   de bf           out 0x3e, r29   ; 62
  66:   cd bf           out 0x3d, r28   ; 61
  68:   00 c0           rjmp    .+0           ; 0x6a
```

Onde está o printf ???

Onde está o printf ???

Onde está o printf ???

Onde estão as chamadas a funcao_a e funcao_b

Desenvolvimento de software

- Depuração de software
  - Limitações no uso de printf para depuração:
    - Tempo de execução longo
    - No avr-libc, printf não é compilado como uma função re-entrante!

Desenvolvimento de software

- Depuração de software
  - Exemplo de sinalização de eventos: LEDs

```
// Macros:
#define USE_DEBUG   1
#define DEBUG_POINT(l2,l1,l0) if(USE_DEBUG) {\
    PORTB&=(0xF8|((l2)?_BV(PB2):0) | ((l1)?_BV(PB1):0) | ((l0)?_BV(PB0):0) );\
    PORTB|=(((l2)?_BV(PB2):0)|((l1)?_BV(PB1):0)|(((l0)?_BV(PB0):0)) );\
    }

void main (void)
{
    DEBUG_POINT(0,1,0);
    funcao_a();
    DEBUG_POINT(1,0,0);
    funcao_b();
    DEBUG_POINT(0,0,0);
}

void funcao_a(void)
{
    DEBUG_POINT(0,1,1);
}

void funcao_b(void)
{
    DEBUG_POINT(1,0,1);
}
```

Desenvolvimento de software

- Depuração de software
  - Exemplo de sinalização de eventos: USE_DEBUG=1, OPT=s

```
0000005c <funcao_a>:
  5c:   c2 98           cbi 0x18, 2 ; 24
  5e:   88 b3           in  r24, 0x18   ; 24
  60:   83 60           ori r24, 0x03   ; 3
  62:   88 bb           out 0x18, r24   ; 24
  64:   08 95           ret

00000066 <funcao_b>:
  66:   c1 98           cbi 0x18, 1 ; 24
  68:   88 b3           in  r24, 0x18   ; 24
  6a:   85 60           ori r24, 0x05   ; 5
  6c:   88 bb           out 0x18, r24   ; 24
  6e:   08 95           ret

00000070 <main>:
  70:   cf e5           ldi r28, 0x5F   ; 95
  72:   d4 e0           ldi r29, 0x04   ; 4
  74:   de bf           out 0x3e, r29   ; 62
  76:   cd bf           out 0x3d, r28   ; 61
  78:   88 b3           in  r24, 0x18   ; 24
  7a:   8a 7f           andi    r24, 0xFA   ; 250
  7c:   88 bb           out 0x18, r24   ; 24
  7e:   c1 9a           sbi 0x18, 1 ; 24
  80:   ed df           rcall   .-38        : 0x5c
```

DEBUG_POINT(0,1,1);

DEBUG_POINT(1,0,1);

DEBUG_POINT(0,1,0);

Tamanho do código: 108 bytes

Desenvolvimento de software

- Especificidades da linguagem C – GNU
  - Tipos de variáveis e tamanho (arquitetura AVR)
    - [unsigned] char : 8 bits
    - [unsigned] int : 16 bits
    - [unsigned] long int : 32 bits
    - [unsigned] long long : 64 bits
    - Ponteiros: 16 bits
    - float, double: 32 bits

Desenvolvimento de software

- Especificidades da linguagem C – GNU
  - Variáveis voláteis
    - Aplicado em situações em que o conteúdo de uma variável pode ser alterada sem que o compilador perceba que a alteração é possível.
    - Casos em que se necessita declarar uma variável como volátil:
      - Registros de periféricos mapeados em memória
      - Variáveis globais utilizadas em interrupções
      - Variáveis globais utilizadas em sistemas multitarefas
    - Efeito: variáveis voláteis não são otimizadas (mesmo com -O3)
    - Exemplos de declarações:
      - volatile char c;
      - volatile float x = M_PI;

Desenvolvimento de software

- Especificidades da linguagem C – GNU
  - Funções inline

```
void ioinit (void);

void main (void)
{
    ioinit ();
}

void ioinit (void)
{
    /* Habilita PB0 como saída */
    DDRB = _BV(DDB0);
}
```

```
00000052 <.do_clear_bss_start>:
  52:   a0 36           cpi r26, 0x60   ; 96
  54:   b1 07           cpc r27, r17
  56:   e1 f7           brne    .-8             ; 0x50
  58:   04 c0           rjmp    .+8             ; 0x62

0000005a <__bad_interrupt>:
  5a:   d2 cf           rjmp    .-92            ; 0x0

0000005c <ioinit>:
  5c:   81 e0           ldi r24, 0x01   ; 1
  5e:   87 bb           out 0x17, r24   ; 23
  60:   08 95           ret

00000062 <main>:
  62:   cf e5           ldi r28, 0x5F   ; 95
  64:   d4 e0           ldi r29, 0x04   ; 4
  66:   de bf           out 0x3e, r29   ; 62
  68:   cd bf           out 0x3d, r28   ; 61
  6a:   f8 df           rcall   .-16            ; 0x5c
  6c:   00 c0           rjmp    .+0             ; 0x6e

0000006e <_exit>:
  6e:   ff cf           rjmp    .-2             ; 0x6e
```

Desenvolvimento de software

- Especificidades da linguagem C – GNU
  - Funções inline

```
inline void ioinit (void);

void main (void)
{
    ioinit ();
}

void ioinit (void)
{
    /* Habilita PB0 como saída */
    DDRB = _BV(DDB0);
}
```

```
00000052 <.do_clear_bss_start>:
  52:   a0 36           cpi r26, 0x60   ; 96
  54:   b1 07           cpc r27, r17
  56:   e1 f7           brne    .-8             ; 0x50
  58:   04 c0           rjmp    .+8             ; 0x62

0000005a <__bad_interrupt>:
  5a:   d2 cf           rjmp    .-92            ; 0x0

0000005c <ioinit>:
  5c:   81 e0           ldi r24, 0x01   ; 1
  5e:   87 bb           out 0x17, r24   ; 23
  60:   08 95           ret

00000062 <main>:
  62:   cf e5           ldi r28, 0x5F   ; 95
  64:   d4 e0           ldi r29, 0x04   ; 4
  66:   de bf           out 0x3e, r29   ; 62
  68:   cd bf           out 0x3d, r28   ; 61
  6a:   81 e0           ldi r24, 0x01   ; 1
  6c:   87 bb           out 0x17, r24   ; 23
  6e:   00 c0           rjmp    .+0             ; 0x70

00000070 <_exit>:
  70:   ff cf           rjmp    .-2             ; 0x70
```

Onde está a diferença?

Desenvolvimento de software

- Especificidades da linguagem C – GNU
  - Funções re-entrantes (-D_REENTRANT)

    Alguns sites reportam que as funções da biblioteca libc não são re-entrantes. Portanto, deve-se evitar usá-las simultaneamente em interrupções e no nível principal. Isto somente pode ser feito se for garantida atomicidade na sua execução.

    Aparentemente, não suportada para a arquitetura AVR

AVR-LIBC

- Tipos inteiros para variáveis (stdint.h)
  - Com sinal:
    - int8_t: inteiro 8 bits
    - int16_t: inteiro 16 bits
    - int32_t: inteiro 32 bits
    - int64_t: inteiro 64 bits
  - Sem sinal:
    - uint8_t: inteiro 8 bits sem sinal
    - uint16_t: inteiro 16 bits sem sinal
    - uint32_t: inteiro 32 bits sem sinal
    - uint64_t: inteiro 64 bits sem sinal

AVR-LIBC

- Laços de espera (avr/delay.h)
  - Funções baseadas no número de ciclos de instrução

```
#define F_CPU 1000000UL  // 1 MHz
//#define F_CPU 14.7456E6
#include <avr/delay.h>
```

```
__inline__ void _delay_loop_1 (uint8_t __count)
__inline__ void _delay_loop_2 (uint16_t __count)
__inline__ void _delay_us (double __us)
__inline__ void _delay_ms (double __ms)
```

  - `count`: contagem de 4 ciclos de relógio
  - Contagem máxima determinada pelo tipo de `count`

AVR-LIBC

- Gerenciamento de energia (avr/sleep.h)
  - Funções baseadas na instrução SLEEP

```
void set_sleep_mode (uint8_t mode)
void sleep_mode (void)
```

  - Definições para ATMEGA8 (em avr/sleep.h)

```
#define SLEEP_MODE_IDLE         0
#define SLEEP_MODE_PWR_DOWN     1
#define SLEEP_MODE_PWR_SAVE     2
#define SLEEP_MODE_ADC          3
#define SLEEP_MODE_STANDBY      4
#define SLEEP_MODE_EXT_STANDBY  5
```

AVR-LIBC

- Gerenciamento de energia (avr/sleep.h)
  - Exemplo: timerint1_ctc do CD de iniciação

```
volatile unsigned int Counter1msTicks = 0; // permite contagens de até 65,535 s

SIGNAL (SIG_OUTPUT_COMPARE1A)
{
    // Decrementa Counter1msTicks a cada interrupção se diferente de 0.
    if(Counter1msTicks!=0){
        --Counter1msTicks;
    }
}
```

```
    /* Gerenciamento de consumo de energia */
    set_sleep_mode(SLEEP_MODE_IDLE);
```

```
void delay(unsigned int time_ms) // limitado em até 65535 ms.
{
    unsigned int counter = 1;

    cli(); // Inicia região crítica: garante acesso exclusivo a Counter1msTicks
    Counter1msTicks = time_ms;
    sei(); // Encerra região crítica
    while(counter>0){
        sleep_mode(); // entra em modelo SLEEP. Acorda somente quando ocorre uma interrupção.
        cli(); // Inicia região crítica: garante acesso exclusivo a Counter1msTicks
        counter = Counter1msTicks;
        sei(); // Encerra região crítica
    }
}
```

AVR-LIBC

- E/S padrão (stdio.h)
  - Direcionamento:
    - Deve-se definir as funções básicas de tratamento E/S de caracteres

```
FILE* fdevopen( int(*     put)(char),
                int(*     get)(void),
                int opts  __attribute__((unused))
              )
```

→ Não usado!

AVR-LIBC

- E/S padrão (stdio.h)
  - Exemplo: terminal do CD de iniciação

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

/*****************************************************************************
*** Defines:
*****************************************************************************

/*****************************************************************************
*** Protótipos:
*****************************************************************************

void USART_Init(void);
int USART_Transmit( char data );
int USART_Receive( void );
```

AVR-LIBC

- E/S padrão (stdio.h)
  - Exemplo: terminal do CD de iniciação

```
/*************************************************************************************
*** main:
*************************************************************************************
int main(void){
    char ch;

    USART_Init();

    /*registrando as funções de leitura e escrita de um byte pela serial*/
    fdevopen(USART_Transmit, USART_Receive, 0);

    //escrevendo "Ola, mundo!" pela serial
    printf ("Ola, mundo");

    while(1){
        // As duas linhas abaixo fazem a mesma coisa: o caracter que chegar
        // pela serial será ecoado por ela. Entretanto, as funções usadas são dife
        // A diferença fundamental está na compexidade de cada uma delas.
        // Para ter uma melhor compreensão disto, descomente apenas uma das linhas
        // o programa para ver a diferença de tamanho do programa compilado.
        scanf ("%c", &ch);  printf ("%c", ch); // Solução 1.
//      putchar(getchar()); // Solução 2.
    }
}
```

AVR-LIBC

- E/S padrão (stdio.h)
  - Exemplo: terminal do CD de iniciação

```
/*
Função que faz a inicialização da usart para comunicacao assíncrona com o PC
a 9600 baud rate com 16MHz de cristal externo.
A recepção e transmissão são habilitadas, sobrepondo as funções dos pinos TxD e RxD.
Nenhuma interrupção é habilitada.
*/
void USART_Init(void)
{
    /*zerando o conteúdo do registrador de dados da serial*/
    UDR=0;
    //Seta baud rate de 9600 para fosc= 16MHz
    UBRRH = 0;
    UBRRL = 103;
    //Habilita transmissão e recepção serial
    UCSRB = _BV(TXEN)| _BV(RXEN);
    //Setar tamanho palavra
    //URSEL = 1 -> seleciona acesso para UCSRC (0 is UBRRH) reading and writing
    //UMSEL = 0 -> Asynchronous Operation
    //UPM1 e UPM0 = 00 -> sem paridade
    //USBS = 1 -> seleciona 2 bits de parada a ser inserido pelo transmissor
    //UCSZ 2 1 0 = 011 -> escolhe tamanho de caractere de 8 bits p/ dado
    UCSRC = _BV(URSEL) | _BV(UCSZ1) | _BV(UCSZ0);
}
```

AVR-LIBC

- E/S padrão (stdio.h)
  - Exemplo: terminal do CD de iniciação

```
/*
Função que faz a leitura da um byte da serial usando pooling
*/
int USART_Receive( void )
{
    /* Wait for data to be received */
    while ( !(UCSRA & (1<<RXC)) );

    /* Get and return received data from buffer */
    return UDR;
}

/*
Função que faz a transmissão de um byte pela serial.
*/
int USART_Transmit(  char data )
{
    /* Wait for empty transmit buffer */
    while ( !( UCSRA & (1<<UDRE)) );

    /* Put data into buffer, sends the data */
    UDR = data;
    return 0;
}
```

AVR-LIBC

- Registros de E/S (io.h)
  - Todos os registros são definidos pelo nome em io.h
  - Manipulação de registros:

```
outb(PORTB, inb(PORTB) & 0xFE);
```

ou

```
PORTB &= 0xFE;
```

  - Manipulação de bits:

Bit manipulation

```
#define _BV(bit)  (1 << (bit))
```

IO register bit manipulation

```
#define bit_is_set(sfr, bit)  (_SFR_BYTE(sfr) & _BV(bit))
#define bit_is_clear(sfr, bit)  (!(_SFR_BYTE(sfr) & _BV(bit)))
#define loop_until_bit_is_set(sfr, bit)  do { } while (bit_is_clear(sfr, bit))
#define loop_until_bit_is_clear(sfr, bit)  do { } while (bit_is_set(sfr, bit))
```

AVR-LIBC

- EEPROM (avr/eeprom.h)
  - Funções de acesso à EEPROM interna

```
#define eeprom_is_ready()   bit_is_clear(EECR, EEWE)
#define eeprom_busy_wait()   do {} while (!eeprom_is_ready())
uint8_t eeprom_read_byte (const uint8_t *addr)
uint16_t eeprom_read_word (const uint16_t *addr)
    void eeprom_read_block (void *buf, const void *addr, size_t n)
    void eeprom_write_byte (uint8_t *addr, uint8_t val)
    void eeprom_write_word (uint16_t *addr, uint16_t val)
    void eeprom_write_block (const void *buf, void *addr, size_t n)
```

AVR-LIBC

- Interrupções e sinais (avr/interrupt.h e avr/signal.h)
  - Definição de rotinas de gerenciamento usando as macros INTERRUPT() e SIGNAL().
  - Exemplos
    - SIGNAL (SIG_OUTPUT_COMPARE1A)
    - INTERRUPT(SIG_ADC)

```
SIGNAL (SIG_OUTPUT_COMPARE2)
{
    // Decrementa Counter10usTicks a cada interrupção se diferente de 0.
    if(Counter10usTicks!=0){
        --Counter10usTicks;
    }
}
```

  - Diferença entre INTERRUPT e SIGNAL:
    - INTERRUPT: a rotina inicia com uma instrução SEI
    - SIGNAL: a rotina inicia com todas as interrupções desabilitadas (situação default do microcontrolador)

AVR-LIBC

- Interrupções e sinais (avr/interrupt.h e avr/signal.h)
  - Identificadores de fonte de interrupção para ATmega8

1	__vector_default
2	SIG_INTERRUPT0
3	SIG_INTERRUPT1
4	SIG_OUTPUT_COMPARE2
5	SIG_OVERFLOW2
6	SIG_INPUT_CAPTURE1
7	SIG_OUTPUT_COMPARE1A
8	SIG_OUTPUT_COMPARE1B
9	SIG_OVERFLOW1
10	SIG_OVERFLOW0

11	SIG_SPI
12	SIG_UART0_RECV
13	SIG_UART0_DATA
14	SIG_UART0_TRANS
15	SIG_ADC
16	SIG_EEPROM_READY
17	SIG_COMPARATOR
18	SIG_2WIRE_SERIAL
19	SIG_SPM_READY

Consultar tabela da página 44 do manual do ATmega8

AVR-LIBC

- Outras facilidades
  - Suporte a Bootloader
  - CRC
  - Geração de bit de paridade
  - Manipulação de blocos de memória da FLASH
  - Cão-de-guarda
  - Operações matemáticas em ponto flutuante
  - Geração de números aleatórios
  - Manipulação de strings
  - Teste de caracteres

Referências

[Li & Yao, 2003] LI, Q.; YAO, C. *Real-Time Concepts for Embedded Systems*. CMPBooks, 2003.