5197 - Sistema Digitais

Bacharelado de Informática

UEM – DIN - Prof. Elvio

v. 17a

Roteiro

- A Família AVR
- Introdução ao ATmega328
- Características Gerais
- Memórias

Família AVR

- AVR = Advanced Virtual RISC
- Fundadores são Alf Egil Bogen e Vegard Wollan, alunos de doutorado no Norwegian Institute of Technology
- São processadores RISC, com arquitetura Harvard e barramento de 8 bits
  - Recentemente foi lançando o AVR UC3 de 32 bits
- Possuem 32 registradores de uso geral de 8 bits
- Maioria das instruções executam em 1 ciclo de relógio
  - Produz 20 MIPS com relógio de 20 MHz
- Projetado para uso em linguagem C

Família AVR

- Dispositivos diferenciam-se pelo:
  - tamanho das memórias
  - números de linhas de E/S (depende do encapsulamento)
  - blocos funcionais disponíveis (ADC, UART, etc.)
- Três conjuntos:

	TinyAVR	MegaAVR	XMegaAVR
Encapsulamento	6 a 32 pinos	32 a 100 pinos	32 a 100 pinos
Memória "*flash*"	0,5 a 16 KB	4 a 256 KB	8 a 384 KB
Memória RAM	0,03 a 1 KB	0,5 a 16 KB	1 a 32 KB
Memória EEPROM	0 a 0,5 KB	0,25 a 4 KB	0,5 a 4 KB
Pinos E/S máx.	4 a 28	22 a 86	26 a 78
Relógio máx.	20 MHz	20 MHz	32 MHz

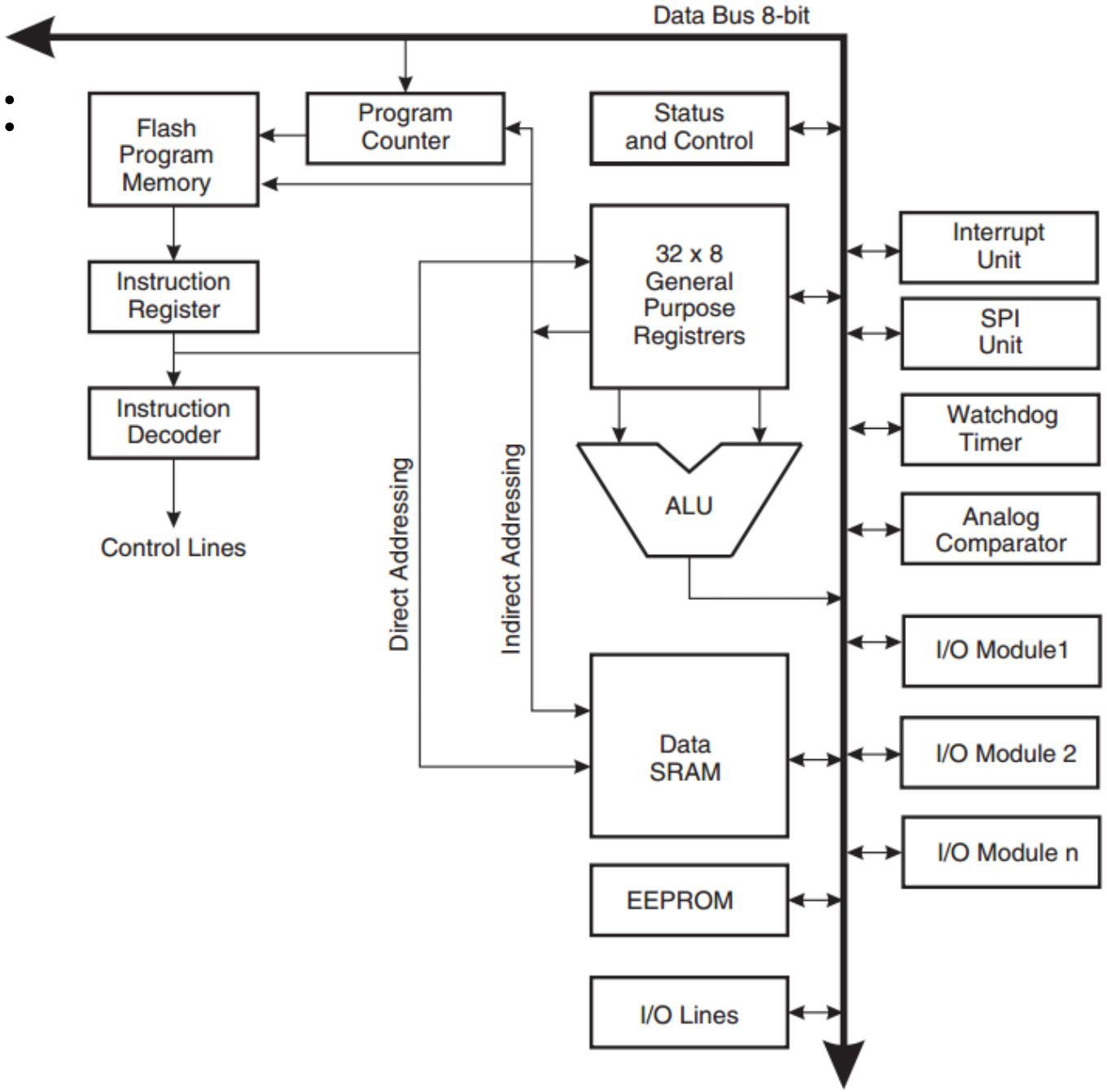
Atmel Studio 7.0

- Plataforma integrada de desenvolvimento para processadores Atmel (inclui AVR)
- Permite desenvolvimento e depuração de código escrito em C, C++ e Assembly
- Inclui biblioteca, exemplos de código, e outras informações
- Inclui editor, compilador e simulador da CPU
- Disponível (apenas) para Windows

ATmega328: Características Gerais

- Memórias: 32 KB "*flash*", 2 KB RAM e 1 KB EEPROM
- Periféricos: 1 UART, 2 SPI, 1 I2C, 1 Comparador
- Portas E/S: até 23
- Arquitetura Harvard: memórias de programa e dados separadas e utilizando barramentos distintos
- Utiliza "*pipeline*"
- Possui 32 registadores de uso geral de 8 bits
- 6 registradores de 8 bits podem ser combinados em 3 registradores de 16 bits para endereçamento indireto
- ALU (Arithmetic Logic Unit) permite operações entre registradores e entre registrador e constante
- Instruções de 16 bits de largura
- Memória "*flash*" dividida em "*boot*" e aplicação

ATmega328: CPU



Registrador SREG

- Contém status da instrução aritmética executada
- Esta informação pode ser usada para alterar o fluxo do programa
- Este registrador não é automaticamente armazenado no caso de interrupção (isto deve ser feito pelo software)

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	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	

Registrador SREG

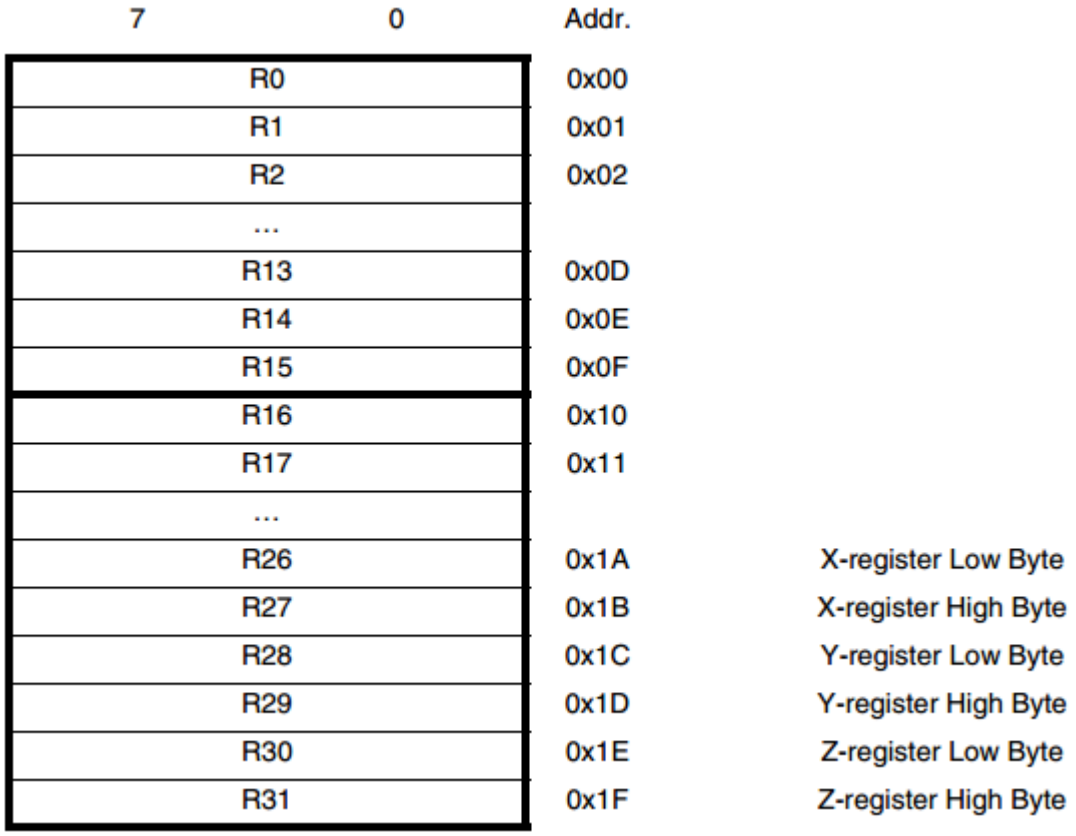
- Global Interrupt Enable (I): habilitação de interrupção
  - É ressetado pelo hardware quando ocorre uma interrupção e setado pela instrução RETI
  - Pode ser alterado pelas instruções SEI (Global Interrupt Enable) e CLI (Global Interrupt Disable)
- Bit Copy Storage (T): cópia de bit
  - Instruções BLD (Bit load from T to Register) e BST (Bit Store from Register to T) leem e armazenam um bit nesta posição do SREG
- Half Carry Flag (H): indica "*half carry*"
  - Útil em operações utilizando BCD
- Sign Bit (S): sinal
  - É um exclusive-or entre os bits N e V
  - S = 1 se N e V forem diferentes; S = 0 se N e V forem iguais

Registrador SREG

- Two's Complement Overflow Flag (V)
  - Indica "*overflow*" na representação complemento de 2
- Negative Flag (N): valor negativo
  - Indica resultado negativo em operação aritimética ou lógica
- Zero Flag (Z)
  - Indica resultado igual a zero em operação aritimética ou lógica
- Carry Flag (C)
  - Indica uma situação de "*carry*" em operação aritimética ou lógica

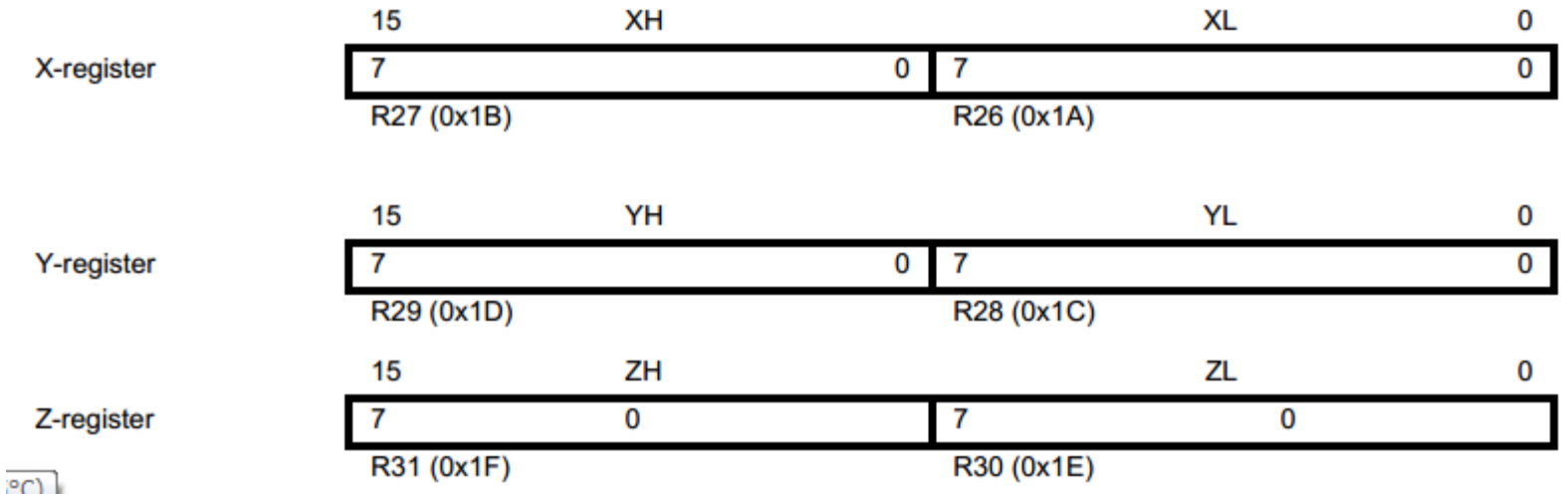
Registradores

- Operações:
  - 1 x 8-bit entrada -> 1 x 8-bit resultado
  - 2 x 8-bit entrada -> 1 x 8-bit resultado
  - 2 x 8-bit entrada -> 1 x 16-bit resultado
  - 1 x 16-bit entrada -> 1 x 16-bit resultado



Registradores

- Registradores R26-R31 podem ser usados em duplas (16 bits) para endereçamento indireto da memória
- Formam os registradores X, Y e Z



Stack Pointer

- Usado para armazenar dados temporários, como variáveis locais ou endereços de retorno de instruções ou chamadas de subrotinas
- Utiliza a RAM e seu tamanho é limitado por ela

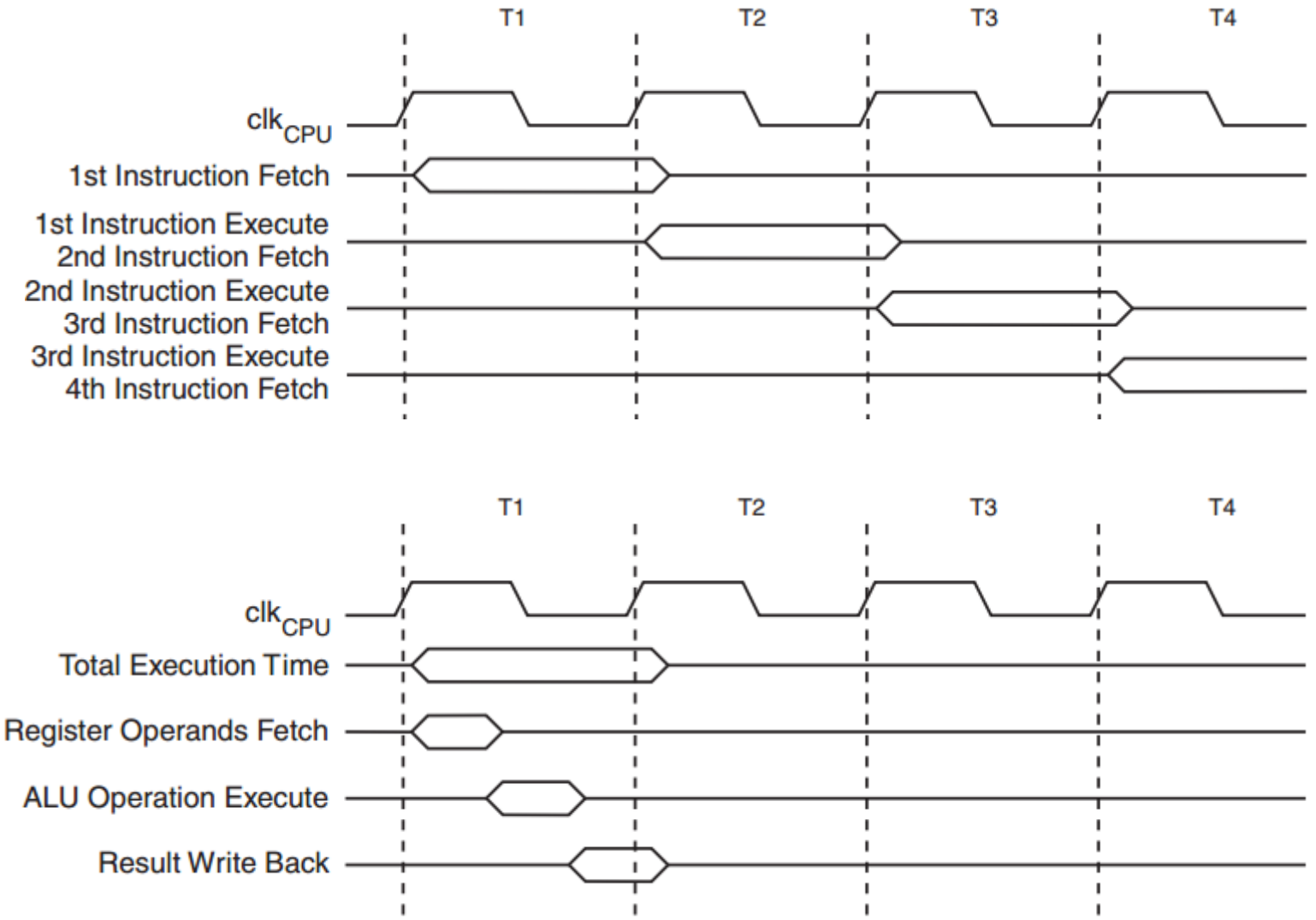
Instruction	Stack pointer	Description
PUSH	Decremented by 1	Data is pushed onto the stack
CALL ICALL RCALL	Decremented by 2	Return address is pushed onto the stack with a subroutine call or interrupt
POP	Incremented by 1	Data is popped from the stack
RET RETI	Incremented by 2	Return address is popped from the stack with return from subroutine or return from interrupt

Stack Pointer

- Implementado em 2 registradores

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	**SP15**	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
0x3D (0x5D)	**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	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	
	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	

Execução


T1
T2
T3
T4
clk CPU
1st Instruction Fetch
1st Instruction Execute
2nd Instruction Fetch
2nd Instruction Execute
3rd Instruction Fetch
3rd Instruction Execute
4th Instruction Fetch
T1
T2
T3
T4
clk CPU
Total Execution Time
Register Operands Fetch
ALU Operation Execute
Result Write Back

Interrupções

- Cada interrupção tem seu bit de habilitação (que deve operar em conjunto com o bit global de habilitação)
- Cada interrupção tem seu endereço específico de tratamento (*interruption vector*), que estão colocados nos endereços mais baixos da memória de programa
- Interrupções também seguem prioridades
  - RESET é a de maior prioridade, seguida de INT0 (interrupção externa) e assim por diante

Interrupções

- Interrupção sensível a borda (fica armazenada)
  - Possui "*flag*" de indicação
  - Hardware limpa automaticamente o "*flag*"
  - "*Flag*" pode também ser ressetado pelo software
  - Se a interrupção ocorre enquanto ela está desabilitada, ela permanece setada
- Interrupção sensível a nível (não fica armazenada)
  - Pode ou não possuir "*flag*" de indicação
  - Se a situação de interrupção deixar de existir antes da interrupção ser habilitada, ela não provoca interrupção

Interrupções

- Tratamento da interrupção ocupa pelo menos 4 ciclos:
  - 1 ciclo para salvar o Program Counter (PC) no Stack
  - 3 ciclos para o "*jump*"
  - Pode demorar mais se em estado de "*sleep*" ou se executando instrução multiciclo (execução precisa ser concluída)
- Retorno da interrupção ocupa 4 ciclos
  - Program Counter (PC) é recuperado do Stack
  - Bit I no SREG é setado
  - Execução retorna ao fluxo original

Interrupções

- Exemplo de código

Assembly Code Example

```
in r16, SREG     ; store SREG value
cli    ; disable interrupts during timed sequence
sbi EECR, EEMPE  ; start EEPROM write
sbi EECR, EEPE
out SREG, r16    ; restore SREG value (I-bit)
```

C Code Example

```
char cSREG;
cSREG = SREG; /* store SREG value */
/* disable interrupts during timed sequence */
_CLI();
EECR |= (1<<EEMPE); /* start EEPROM write */
EECR |= (1<<EEPE);
SREG = cSREG; /* restore SREG value (I-bit) */
```

Interrupções

- Exemplo de código

SEI - Set Global Interrupt Flag

SLEEP - Sleep mode
Implementa o "*sleep mode*" definido no MCU Control Register (MCUCR)

Assembly Code Example

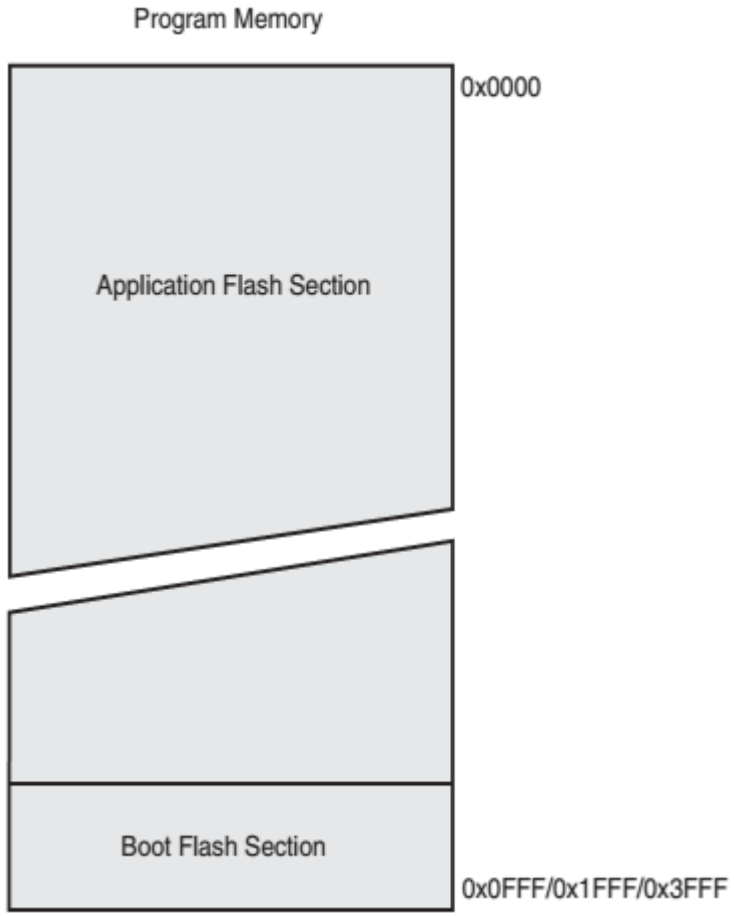
```
sei   ; set Global Interrupt Enable
sleep; enter sleep, waiting for interrupt
; note: will enter sleep before any pending interrupt(s)
```

C Code Example

```
__enable_interrupt(); /* set Global Interrupt Enable */
__sleep(); /* enter sleep, waiting for interrupt */
/* note: will enter sleep before any pending interrupt(s) */
```

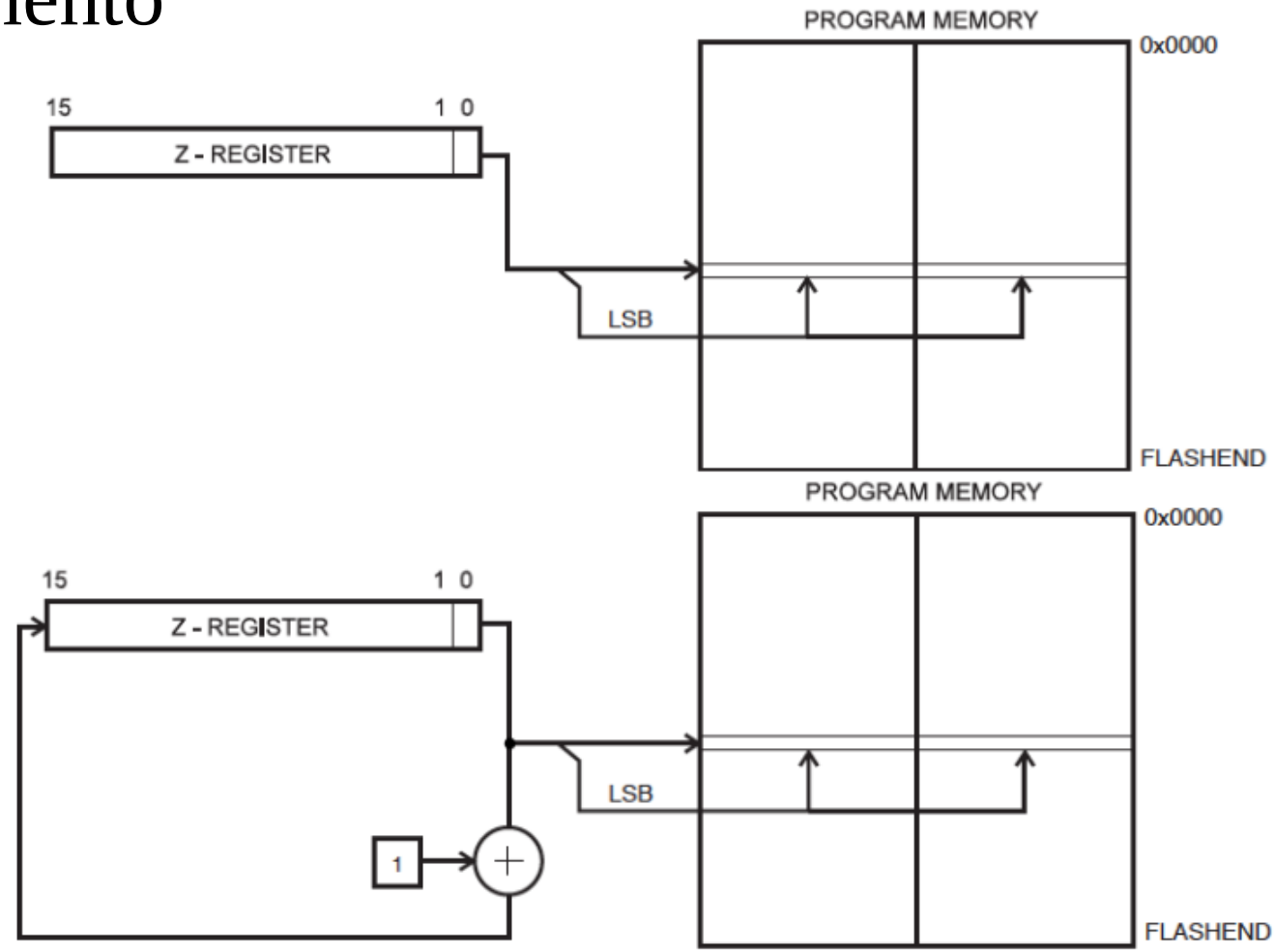
Memória Flash

- Memória não-volátil
- 32 KB para o ATmega328 (organizada em 16 KB x 16 bits)
- Usada para armazenar código
- Permite no mínimo 10.000 ciclos de escrita
- Dividida em secções "*Boot Loader*" e "*Application Program*"



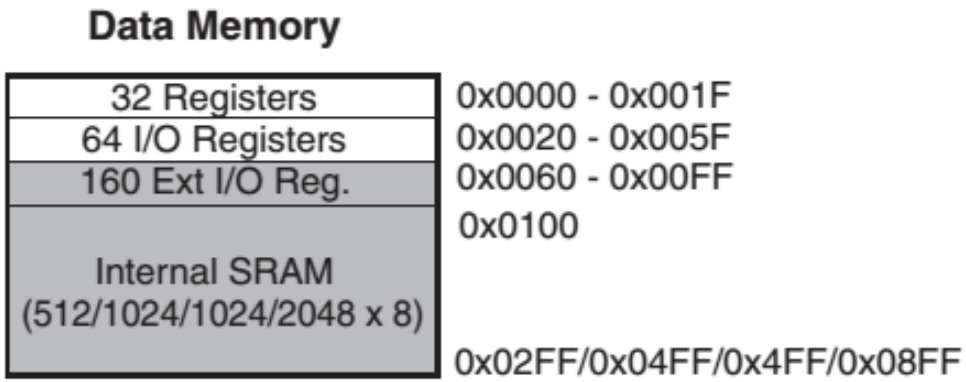
Memória Flash

- Endereçamento



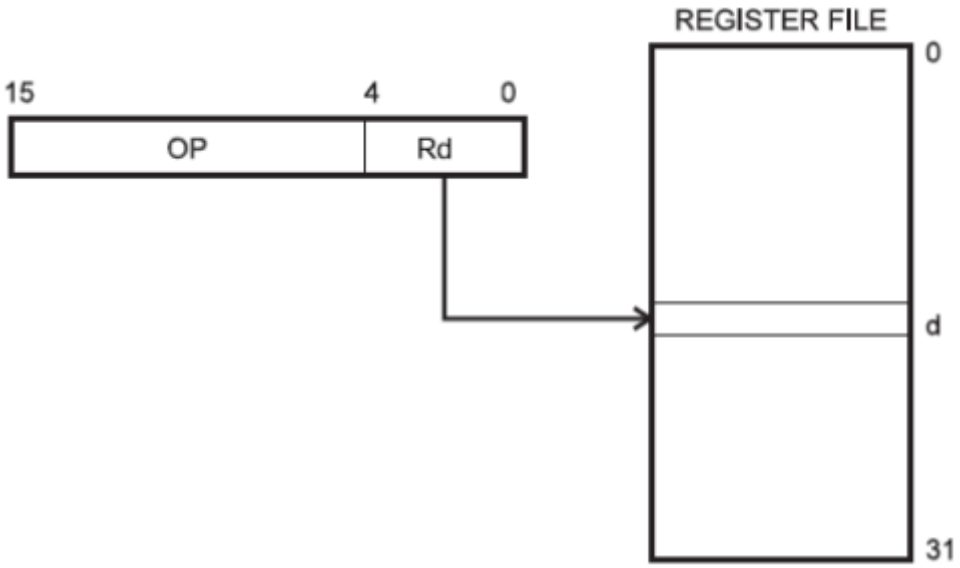
Memória Volátil

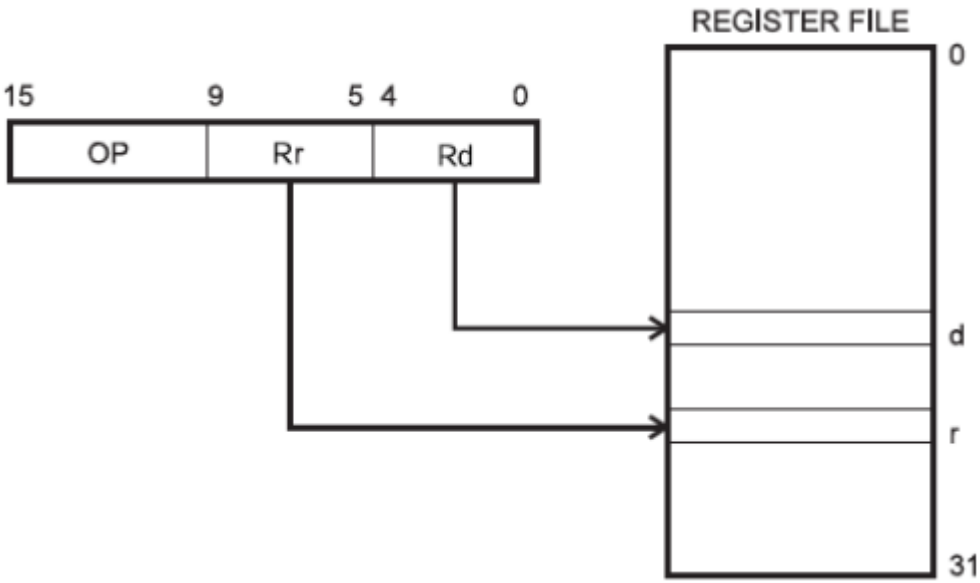
- Memória RAM
  - Memória volátil
  - 2 KB de RAM para ATmega328
  - Usada para armazenar variáveis, pilha...
- Registradores
  - 32 registradores de uso geral
  - 64 registradores de E/S
  - 160 registradores de E/S estendido



Memória Volátil

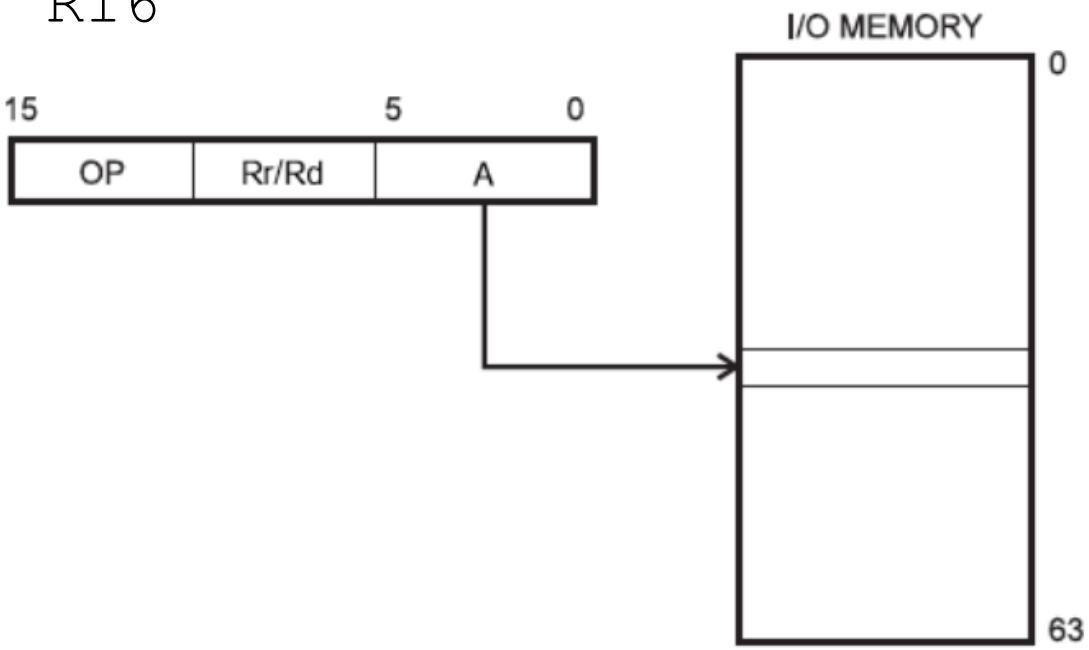
- Endereçamento dos Registradores de uso geral
  - Direto com 1 registrador
    - `INC R16`
    - `CLR R22`
  - Direto com 2 registradores
    - `ADD R16, R17`
    - `CP R22, R5`
    - `MOV R0, R1`





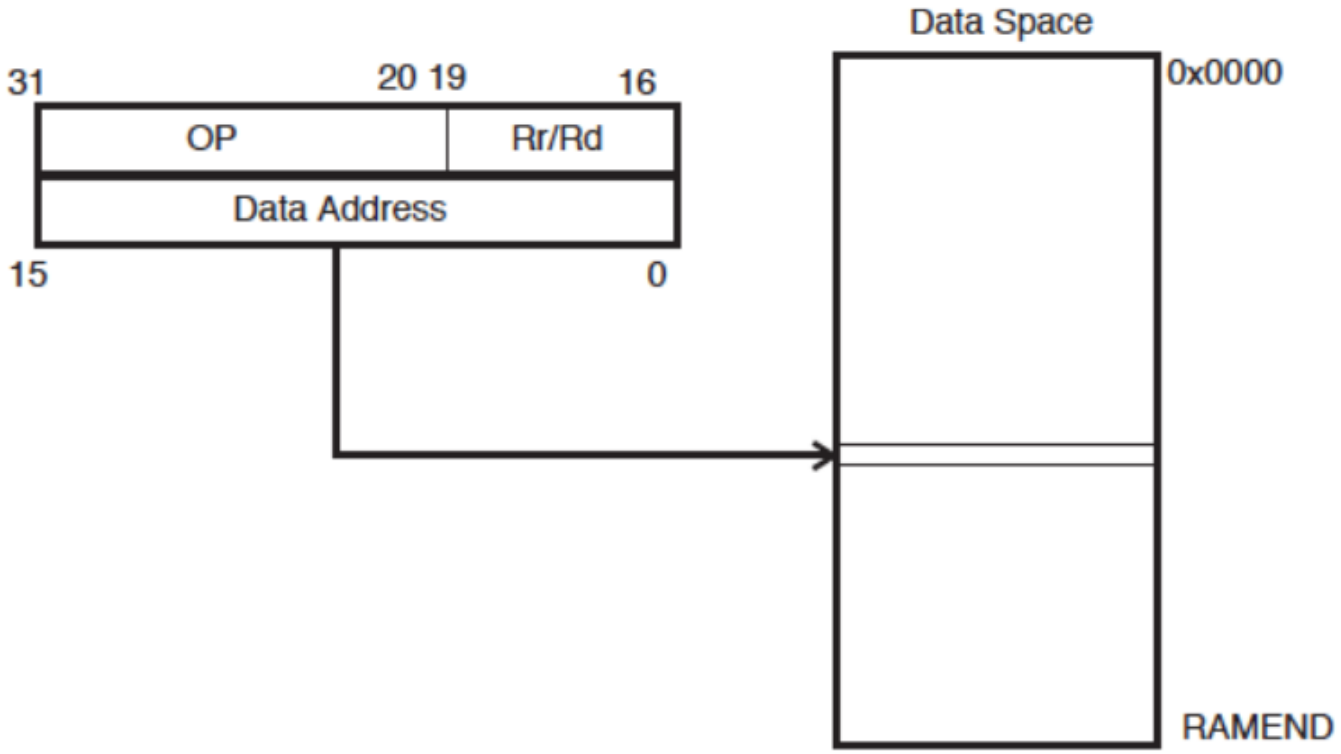
Memória Volátil

- Endereçamento dos Registradores de E/S
  - Direto
    - `IN R16, PIND`
    - `OUT PORTC, R16`



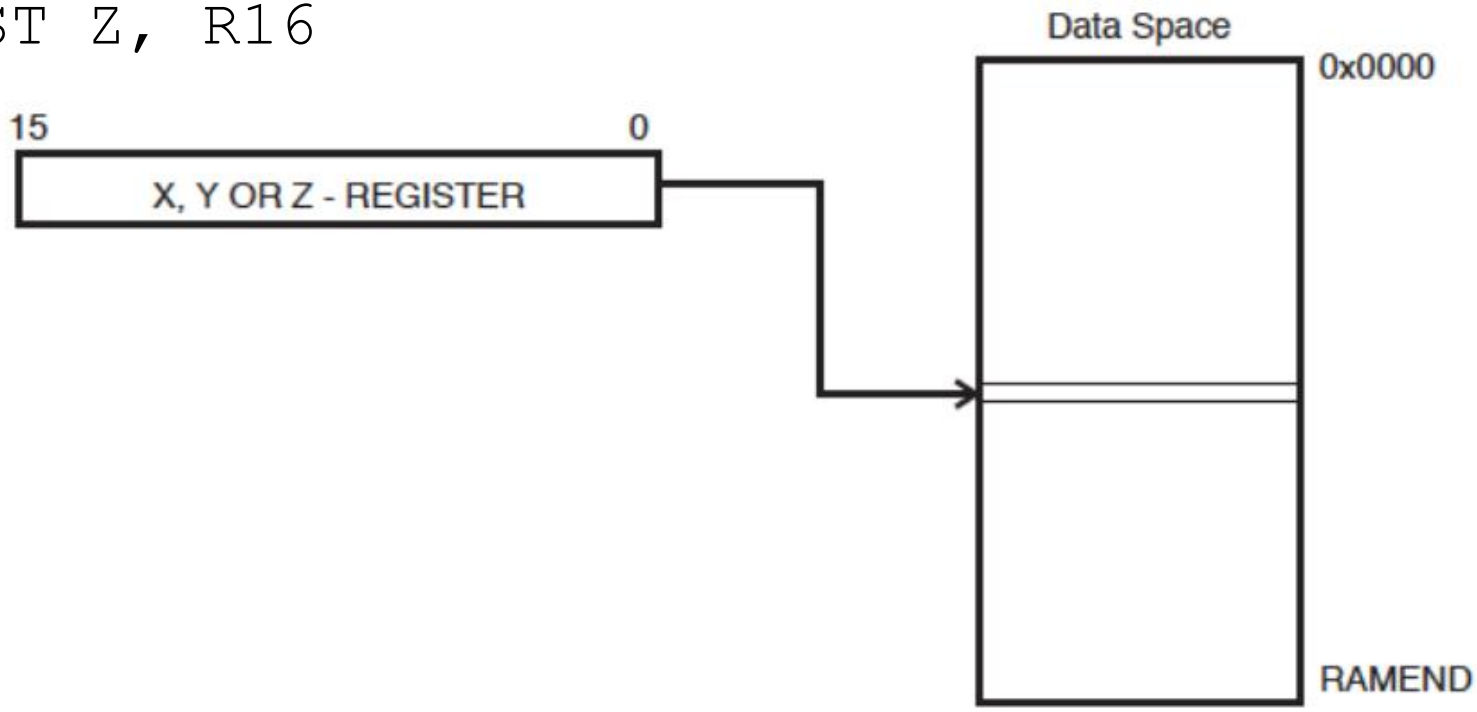
Memória Volátil

- Endereçamento da Memória RAM
  - Direto
    - `STS 0x1000, R16`



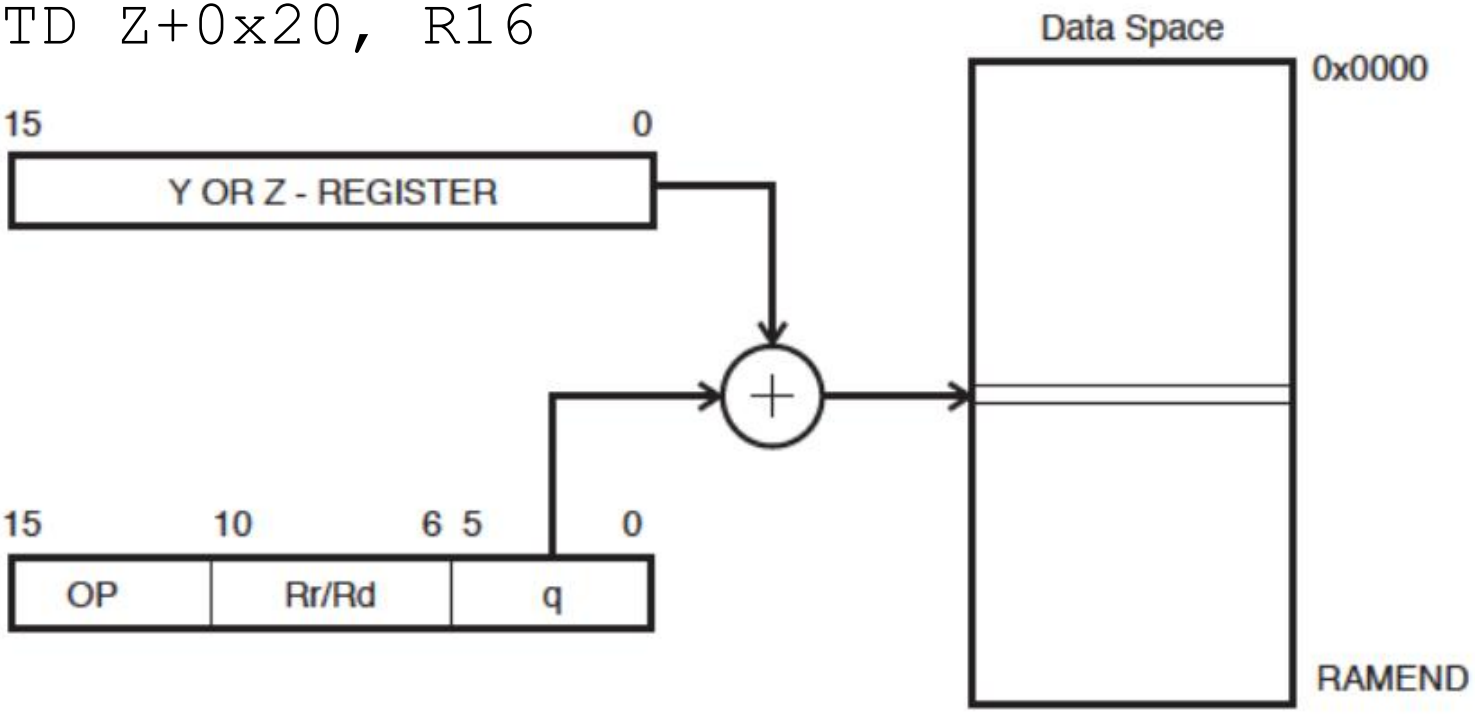
Memória Volátil

- Endereçamento da Memória RAM
  - Indireto
    - `LD R16, Y`
    - `ST Z, R16`



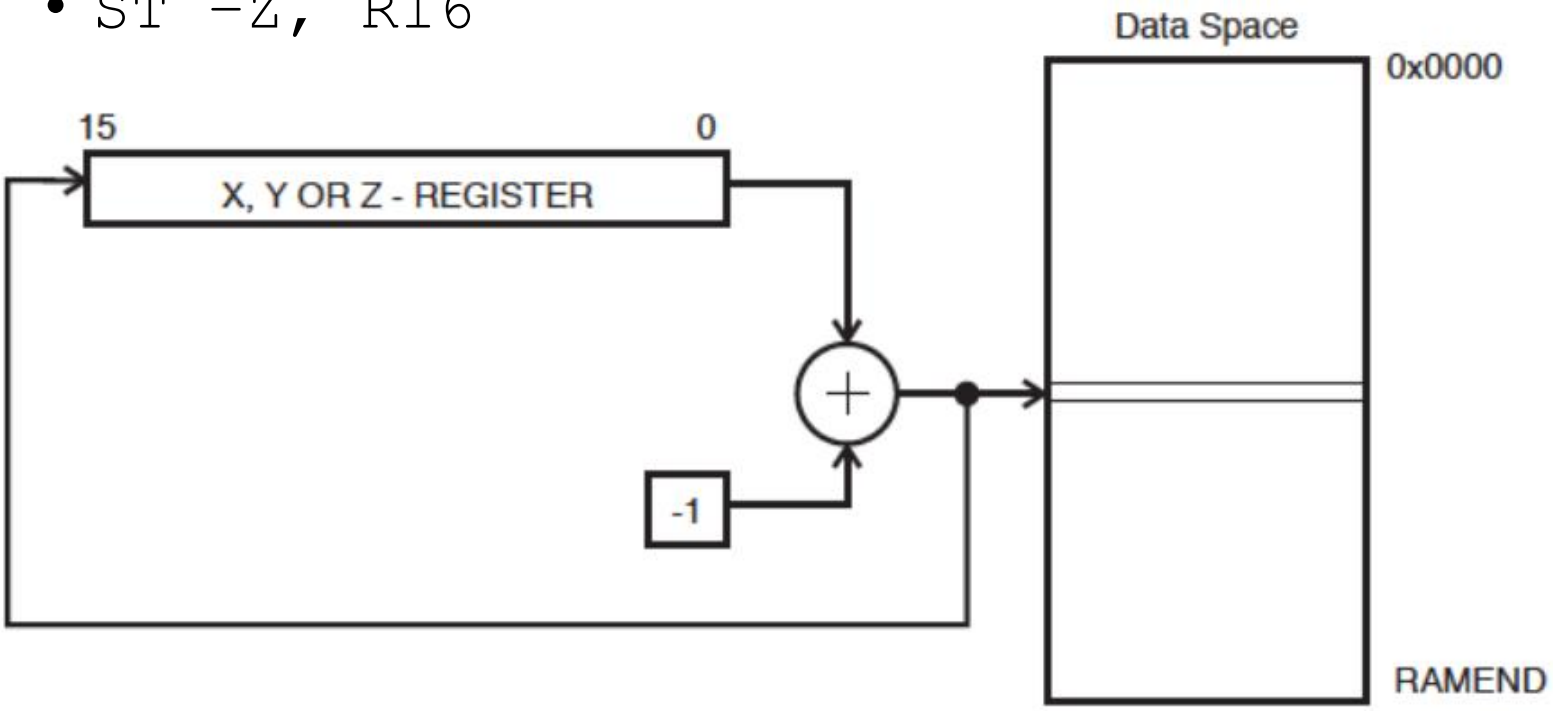
Memória Volátil

- Endereçamento da Memória RAM
  - Indireto com deslocamento
    - `LDD R16, Y+0x10`
    - `STD Z+0x20, R16`



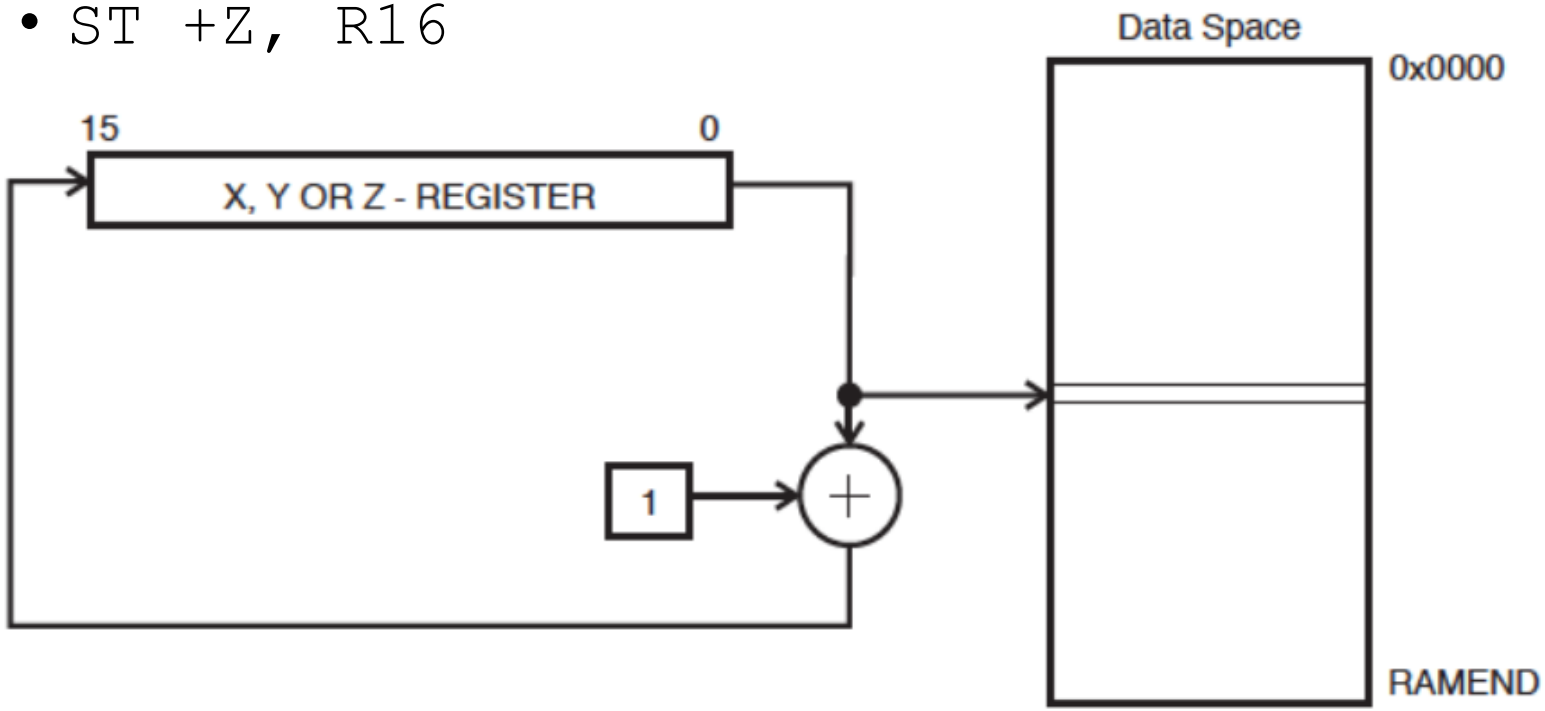
Memória Volátil

- Endereçamento da Memória RAM
  - Indireto com pré-decremento
    - `LD R16, -Z`
    - `ST -Z, R16`



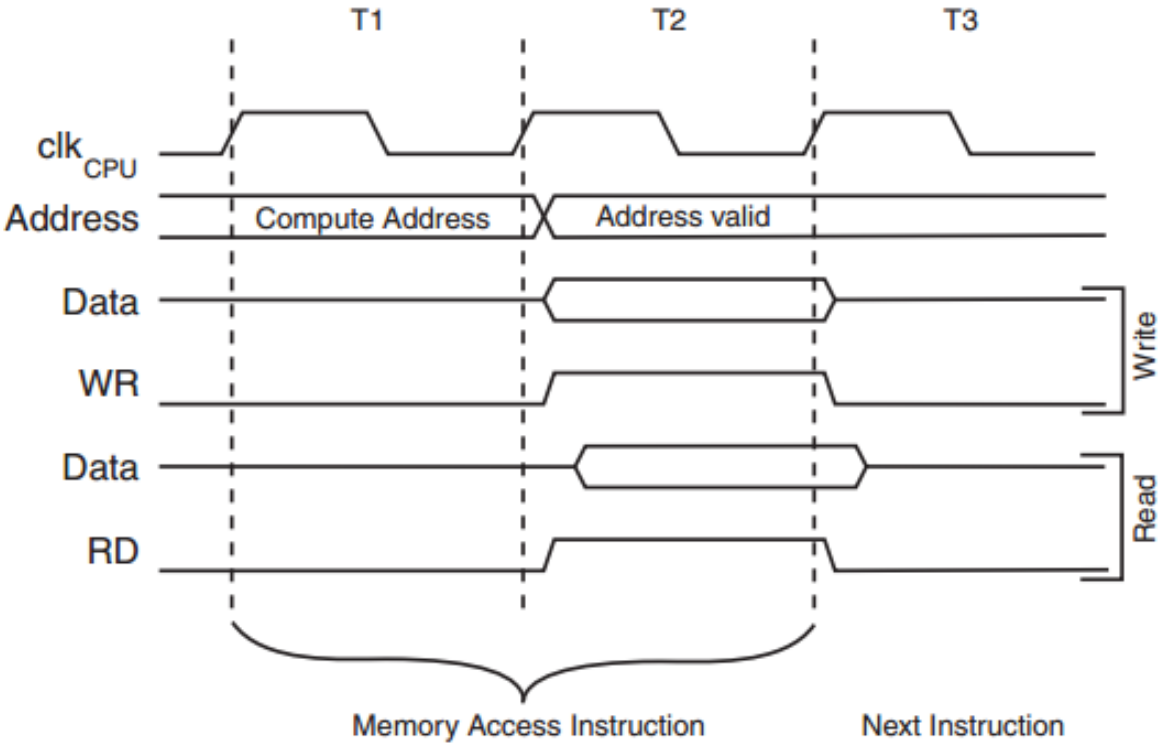
Memória Volátil

- Endereçamento da Memória RAM
  - Indireto com pós-incremento
    - `LD R16, +Z`
    - `ST +Z, R16`



Memória RAM

- Escrita e leitura



Memória EEPROM

- Memória não-volátil
- 1 KB para o ATmega328
- Usada para armazenar dados
- Permite no mínimo 100.000 ciclos de escrita
- Acesso feito pelos registradores:
  - EEPROM Address Registers (EEARH e EEARL)
  - EEPROM Data Register (EEDR)
  - EEPROM Control Register (EECR)
  - Leituras da EEPROM causam atraso de 4 ciclos para execução da próxima instrução
  - Escritas na EEPROM causam atraso de 2 ciclos para execução da próxima instrução

Memória EEPROM

- Registradores EEPROM Address (EEAR[H,L])
  - Contém um endereço na área da EEPROM

Bit	15	14	13	12	11	10	9	8	
0x22 (0x42)	–	–	–	–	–	–	–	**EEAR8**	**EEARH**
0x21 (0x41)	**EEAR7**	**EEAR6**	**EEAR5**	**EEAR4**	**EEAR3**	**EEAR2**	**EEAR1**	**EEAR0**	**EEARL**
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	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	X	
	X	X	X	X	X	X	X	X	

Memória EEPROM

- Registrador EEPROM Data (EEDR)
  - Contém o dado a ser escrito na ou lido da EEPROM no endereço apontado por EEAR

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	MSB							LSB	EEDR
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	

- Registrador EEPROM Control (EECR)
  - Especifica a operação a ser realizada na EEPROM

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

Memória EEPROM

- Registrador EEPROM Control (EECR)
  - EEPROM Programming Mode Bits (EEPM[1-0]): Define o modo de escrita na EEPROM

EEPM1	EEPM0	Programming Time	Operation
0	0	3.4ms	Erase and Write in one operation (Atomic Operation)
0	1	1.8ms	Erase Only
1	0	1.8ms	Write Only
1	1	–	Reserved for future use

Memória EEPROM

- Registrador EEPROM Control (EECR)
  - EEPROM Ready Interrupt Enable (EERIE)
    - Habilita a EEPROM Ready Interrupt
    - A EEPROM Ready Interrupt é gerada quando o bit EEPE é ressetado (exceto em escritas à EEPROM e com a instrução SPM – Store Program Memory)
  - EEPROM Master Write Enable (EEMPE)
    - Habilita a escrita na EEPROM
    - O hardware resseta este bit automaticamente após a escrita
  - EEPROM Write Enable (EEPE)
    - Habilita a escrita na EEPROM do dado presente no registrador EEDR e no endereço especificado pelo registrador EEAR
  - EEPROM Read Enable (EERE)
    - Habilita a leitura da EEPROM no endereço especificado pelo registrador EEAR

Memória EEPROM

- Escrita (Assembly)

Assembly Code Example

```
EEPROM_write:
  ; Wait for completion of previous write
  sbic EECR,EEPE
  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 EEMPE
  sbi  EECR,EEMPE
  ; Start eeprom write by setting EEPE
  sbi  EECR,EEPE
  ret
```

SBIC - Skip if Bit in I/O Register is Cleared
Verifica bit de registrador de E/S e pula a próxima instrução se bit é zero.

RJMP - Relative Jump
Pula para o endereço indicado

OUT - Store Register to I/O Location

SBI - Set Bit in I/O Register

RET - Return from Subroutine

Memória EEPROM

- Escrita (C)

C Code Example

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
  /* Wait for completion of previous write */
  while(EECR & (1<<EEPE))
    ;
  /* Set up address and Data Registers */
  EEAR = uiAddress;
  EEDR = ucData;
  /* Write logical one to EEMPE */
  EECR |= (1<<EEMPE);
  /* Start eeprom write by setting EEPE */
  EECR |= (1<<EEPE);
}
```

Memória EEPROM

- Leitura (Assembly)

Assembly Code Example

```
EEPROM_read:
  ; Wait for completion of previous write
  sbic EECR,EEPE
  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
  ret
```

SBIC - Skip if Bit in I/O Register is Cleared
Verifica bit de registrador de E/S e pula a próxima instrução se bit é zero.

RJMP - Relative Jump
Pula para o endereço indicado

OUT - Store Register to I/O Location

SBI - Set Bit in I/O Register

IN – Load an I/O Location to Register

RET - Return from Subroutine

Memória EEPROM

- Leitura (C)

C Code Example

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
  /* Wait for completion of previous write */
  while(EECR & (1<<EEPE))
    ;
  /* Set up address register */
  EEAR = uiAddress;
  /* Start eeprom read by writing EERE */
  EECR |= (1<<EERE);
  /* Return data from Data Register */
  return EEDR;
}
```