Centro Universitário Senac

**Fernando Alves da Silva**

**Victor Eleuterio Trindade**

**Danilo**

­­­­

Projeto CPU

Relatório de código em Assembly para Arduino Uno ATmega328/P

São Paulo

2016

Fernando Alves da Silva

Victor Eleuterio Trindade

Danilo

Projeto CPU

Relatório de código em Assembly para Arduino Uno ATmega328/P

Produção coletiva apresentada à disciplina de Organização e Arquitetura de Computadores do Centro Universitário Senac, como requisito parcial para a nota de graduação em Bacharelado em Ciência da Computação.

São Paulo

2016

Sumário

Introdução 4

Objetivo 5

Materiais e Métodos 6

Materiais 6

*AVR Studio* 6

*Arduino Uno ATmega328/P* 6

AVRDUDE 6

Displays de 7 segmentos 6

Métodos 6

Resultado 9

Conclusão 10

Referências 11

Apêndice 12

# Introdução

# Objetivo

Gerar e aplicar código em Assembly para o Arduino Uno ATmega328/P capaz de contar os segundos até o valor máximo dos displays(9 em cada display, chegando ao valor 99).

# Materiais e Métodos

## Materiais

Para a realização deste trabalho, fez-se necessário o uso dos seguintes materiais:

### *AVR Studio*

Plataforma de desenvolvimento integrado(IDP) para desenvolvimento e depuração de microcontroladores.

### *Arduino Uno ATmega328/P*

Plataforma física de computação de código aberto baseado numa placa microcontroladora.

### AVRDUDE

É uma aplicação de código aberto usada para embarcar o código no microprocessador do arduino.

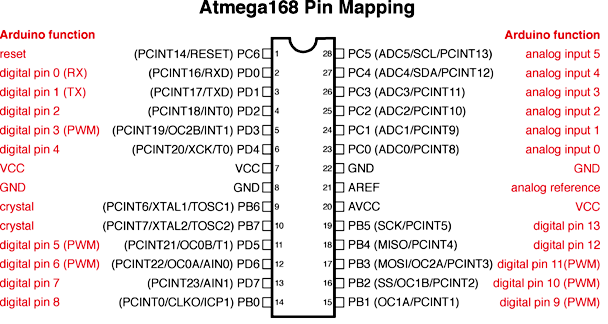
### Displays de 7 segmentos

Displays de 7 segmentos que mostram a contagem de segundos.

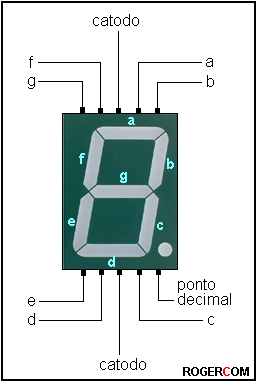
## Métodos

Fez-se necessário, antes de começar o desenvolvimento do projeto, realizar um estudo acerca da documentação do arduino e ao manual dos micro-controladores, para que se fizesse possível o entendimento relativo aos comandos do código em Assembly.

Uma vez realizada a pesquisa quanto a documentação, demos início a busca de material explicativo relativo aos displays de 7 segmentos que seriam utilizados para demonstrar a contagem dos segundos, além do mapeamento dos pinos, para realizar a configuração inicial do envio dos bits para cada segmento do display.



Dado o conteúdo necessário em mãos, demos início a fase de configurar e interligar as portas dos displays para as portas dos arduino.



Com base na imagem acima, utilizamos a função SBI(Seta o bit em um registrador de entrada e saída e dura 2 clocks) e ‘*setamos*‘ a porta D do arduino de 1 até 7 para os seguimentos de A a F do display de contagem de dezenas, e, em seguida, ‘*setamos’* a porta B do arduino de 0 até 4 e a porta C de 0 até 1 para os seguimentos de A até F do segundo display, de unidade. Dessa forma se fez possível o controle de cada um dos segmentos, ativando-os de acordo com o bit enviado pela placa(0 ou 1).

Com as portas devidamente interligadas, deu-se início à elaboração do código, que utilizando o comando RCALL(Chamada de Função que dura 3 clocks) chamava o primeiro display(da dezena) e carregava os bits utilizando a função LDI(carrega uma constante de 8 bits e dura 1 clock), formando o primeiro valor da dezena por 0b00011111 no registrador 26 de uso geral e 0b00000001 no registrador 27 de uso geral, que em seguida eram enviados para o display por meio do comando OUT(armazena o valor do registrador em um dispositivo de entrada e saída e dura 1 clock), formando o número zero no display.

Após o processo de definição do primeiro número no display da dezena, é chamada uma segunda função, que define o número a ser formado no display de unidade. Esta função utiliza o mesmo processo da função de dezena, com exceção da chamada de uma terceira função denominada *delay*, que carrega o valor 80 no registrador 16, o valor 250 no registrador 17 e o valor 200 no registrador 26, e, em seguida, utiliza o comando DEC(Decrementa o valor e leva 1 clock) e um NOP(sem operação, leva 1 clock)e chama a função novamente até que o valor do registrador 26 chegue a 0, ativando o condicional BRNE(não prossegue se não for igual a zero, leva 2 clocks se for diferente de zero e retorna,e 1 para continuar), e fazendo o mesmo com os registradores 17 e 16, com exceção do NOP. Desta forma, ao se somar todos os ciclos de clocks dessas chamadas, tem-se o valor de 20000000 clocks, que é equivalente a 1 segundo.

# Resultado

Obtivemos como resultado um contador que vai de 0 a 99, como esperado no objetivo, mas que não ficou totalmente preciso, uma vez que não consideramos o clock do delay 0,1 e 2, que carregam os valores nos registradores 16, 17 e 26. A falta de precisão fica visível por volta de 3 minutos.

# Conclusão

Com base nos resultados obtidos, é imprescindível ressaltar que gerar códigos em assembly(linguagem nativa) é um processo deveras trabalhoso, porém, vale destacar que este processo pode gerar aplicações de forma otimizada, de tal modo que se sobressaia à utilização de linguagens de alto nível, embora o código varie de acordo com a plataforma a ser utilizada.

Cabe ainda relevar que a documentação relativa a linguagens nativas é escassa, sendo presente somente em idiomas que não o português, dificultando o processo de criação e levando o programador a buscar outros meios de conhecimento, tais como fóruns de ajuda que contenham bons programadores assembly, que são tão escassos quanto a documentação.

Levando em consideração as informações obtidas, pode-se chegar ao consenso de que se faz necessário o incentivo ao estudo nesta área, para que o acesso a informações referente a linguagens nativas torne-se mais fácil, tendo um maior número de exemplificações e textos auxiliares.

# Referências

- 8-bit AVR Microcontrollers, ATMmega328/P DATASHEET COMPLETE. Disponível em: <<http://www.atmel.com/pt/br/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf>>, acesso em <26/11/2016>.

- AVR Microcontrollers, AVR Instruction Set Manual. Disponível em: <<http://www.atmel.com/images/atmel-0856-avr-instruction-set-manual.pdf>>, acesso em <26/11/2016>.

- Controlando displays de 7 segmentos através da porta paralela. Disponível em: <http://www.rogercom.com/pparalela/ControleDisplay.htm>, acesso em <27/11/2016>.

- ATmega168/328-Arduino Pin Mapping. Disponível em: <<https://www.arduino.cc/en/Hacking/PinMapping168>>, acesso em <27/11/2016>.

# Apêndice

;

; AssemblerArduino.asm

;

; Created: 16/11/2016 12:01:16

;

;;CLOCK Table

;;LDI = 1

;;SBI = 2

;;RCALL = 3

;;OUT = 1

;;RJMP = 2

.ORG 0x0000

set\_values:

;;LDI r27, 0b11111111

;;OUT DDRD, r27

;;Seta os bits 1-7 da porta D como saída

SBI PortD, 1

SBI PortD, 2

SBI PortD, 3

SBI PortD, 4

SBI PortD, 5

SBI PortD, 6

SBI PortD, 7

;;Seta os bits 0-5 da porta B como saída

SBI PortB, 0

SBI PortB, 1

SBI PortB, 2

SBI PortB, 3

SBI PortB, 4

;;Seta os bits 0-1 da porta C como saída

SBI PortC, 0

SBI PortC, 1

loop:

RCALL display1

display1:

;;ZERO

LDI r26, 0b00011111

LDI r27, 0b00000001

OUT DDRB, r26

OUT DDRC, r27

RCALL display0

;;UM

LDI r26, 0b00000110

LDI r27, 0b00000000

OUT DDRB, r26

OUT DDRC, r27

RCALL display0

;;DOIS

LDI r26, 0b00011011

LDI r27, 0b00000010

OUT DDRB, r26

OUT DDRC, r27

RCALL display0

;;TRÊS

LDI r26, 0b00001111

LDI r27, 0b00000010

OUT DDRB, r26

OUT DDRC, r27

RCALL display0

;;QUATRO

LDI r26, 0b00000110

LDI r27, 0b00000011

OUT DDRB, r26

OUT DDRC, r27

RCALL display0

;;CINCO

LDI r26, 0b00001101

LDI r27, 0b00000011

OUT DDRB, r26

OUT DDRC, r27

RCALL display0

;;SEIS

LDI r26, 0b00011101

LDI r27, 0b00000011

OUT DDRB, r26

OUT DDRC, r27

RCALL display0

;;SETE

LDI r26, 0b00000111

LDI r27, 0b00000000

OUT DDRB, r26

OUT DDRC, r27

RCALL display0

;;OITO

LDI r26, 0b00011111

LDI r27, 0b00000011

OUT DDRB, r26

OUT DDRC, r27

RCALL display0

;;NOVE

LDI r26, 0b00001111

LDI r27, 0b00000011

OUT DDRB, r26

OUT DDRC, r27

RCALL display0

RJMP loop

display0:

;;ZERO

LDI r27, 0b01111110

OUT DDRD, r27

RCALL delay

;;UM

LDI r27, 0b00001100

OUT DDRD, r27

RCALL delay

;;DOIS

LDI r27, 0b10110110

OUT DDRD, r27

RCALL delay

;;TRÊS

LDI r27, 0b10011110

OUT DDRD, r27

RCALL delay

;;QUATRO

LDI r27, 0b11001100

OUT DDRD, r27

RCALL delay

;;CINCO

LDI r27, 0b11011010

OUT DDRD, r27

RCALL delay

;;SEIS

LDI r27, 0b11111010

OUT DDRD, r27

RCALL delay

;;SETE

LDI r27, 0b00001110

OUT DDRD, r27

RCALL delay

;;OITO

LDI r27, 0b11111110

OUT DDRD, r27

RCALL delay

;;NOVE

LDI r27, 0b11011110

OUT DDRD, r27

RCALL delay

RET

;; 20 Mhz Clock delay

delay:

;; set decimal value 200 to register 16

LDI r16, 80

delay\_1:

;; set decimal value 250 to register 15

LDI r17, 250

delay\_2:

;; set decimal value 40 to register 14

LDI r26, 200­­­

delay\_3:

;; delay 200 \* 4 = 1000 clocks

DEC r26 ;; 1 clock

NOP ;; 1 clock

BRNE delay\_3 ;; 2 clocks on jump, 1 when continue

;; delay 1000 \* 250 = 250000 clocks­­­

DEC r17

BRNE delay\_2

;; delay 80 \* 250000 = 20000000 (1seg)

DEC r16

BRNE delay\_1

RET