“Um computador faz exatamente o que você manda ele fazer, mas isso pode ser muito diferente do que você gostaria que ele fizesse.”

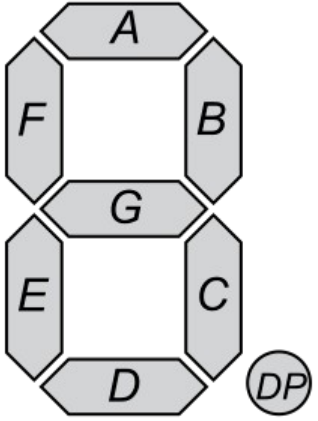
Chamando Funções

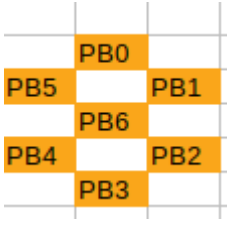
Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Display de 7 segmentos

- Ligue o Display de 7 segmentos **de cátodo comum** na porta B do seu microcontrolador
  - Ligue da maneira que você achar mais conveniente
    - Não ligue o LED do “ponto”
  - Coloque um **resistor de 200Ohms** ligando o terra do display e o terra do seu circuito
  - Dica: Coloque em uma planilha suas ligações para facilitar sua vida





UDESC

Stack Pointer

- Pilha cresce do final da memória para o começo
- Temos um “registrador de pilha”
  - Endereço mapeado na memória que mantém a pilha
  - SPH (Endereço 0x3E) mantém a parte alta do endereço
  - SPL (0x3D) mantém a parte baixa do endereço
  - Obs.: o PIC16f628a não possui nenhum mecanismo de auxílio para implementação de pilha

Stack Pointer

- No x86-64 o stack pointer é inicializado
  - Temos um Sistema Operacional benevolente para cuidar desses detalhes antes do programa iniciar
- No ATMega não podemos garantir que o stack pointer é inicializado corretamente
  - O manual é confuso e contraditório

The Stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. Initial Stack Pointer value equals the last address of the internal SRAM and the Stack Pointer must be set to point above start of the SRAM, see Table 8-3 on page 28.

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	

UDESC

Inicializando o stack pointer

- Por segurança, a primeira coisa que vamos fazer é sempre inicializar o stack pointer no microcontolador

```
.equ RAMEND, 0x8FF ; última posição de memória
.equ SPH, 0x3E ; endereço de memória de SPH
.equ SPL, 0x3D ; endereço de memória de SPL
```

Definição de constantes

```
;ajuste do stack pointer (por precaução)
ldi R16, hi8(RAMEND)
ldi R17, lo8(RAMEND)
out SPL, R17
out SPH, R16
;fim do ajuste do stack pointer
```

Esse trecho sempre vai ser a primeira coisa que vamos executar em nossos programas

Inicializando o stack pointer

- Por segurança, a primeira coisa que vamos fazer é sempre inicializar o stack pointer no microcontolador

```
.equ RAMEND, 0x8FF ; última posição de memória
.equ SPH, 0x3E ; endereço de memória de SPH
.equ SPL, 0x3D ; endereço de memória de SPL
```

```
;ajuste do stack pointer (por precaução)
ldi R16, hi8(RAMEND)
ldi R17, lo8(RAMEND)
out SPL, R17
out SPH, R16
;fim do ajuste do stack pointer
```

hi8/lo8 são diretivas do montador instruindo para pegar os 8 bits mais altos/baixos de determinado valor

Criando funções

- Criar uma função segue os mesmos conceitos aprendidos em x86-64 e MIPS
  - Precisamos definir
    - Onde passar os parâmetros
    - Onde retornar
    - Quais registradores são salvos

Falta de convenção

- Nos manuais não é especificado quais registradores usar
  - Na falta de uma convenção melhor, vamos usar a **convenção do GCC para AVR**
    - http://www1.vak.ru/proj/avr-gcc-regs.html
    - https://gcc.gnu.org/wiki/avr-gcc

Registradores salvos

- Os registradores entre R2 e R17, e os registradores R28 e R29 devem ser salvos pela função sendo chamada
  - Se a função usa esses registradores, ela **obrigatoriamente** deve retorná-los para seus estados originais antes de retornar

Parâmetros e retorno

- Passamos parâmetros nos registradores entre R25 e R8, **nessa ordem**
  - O primeiro parâmetro em R25, segundo em R24, …
- Retorno nos registradores entre R25 e R22
  - Retorno em R25, se precisarmos de mais um byte, usamos R24, …
- Essa é uma **relaxação** da convenção do GCC AVR
  - A convenção inclui detalhes como o alinhamento de variáveis nos registradores

Chamada de função

- No ATmega328P temos 3 opções para chamar uma função
  - call LABEL → salta para o label
    - **Instrução de 32 bits**
  - rcall LABEL
    - O mesmo que call, mas o label deve estar em uma região próxima a instrução
      - Utiliza endereçamento relativo ao PC na CPU
    - **Instrução de 16 bits**
      - Sempre que possível, utilize rcall
  - icall
    - Salta para o endereço armazenado no registrador Z
- Todas instruções de chamada de função **salvam o endereço de retorno na pilha**

Retorno

- Função ***ret*** retorna ao chamador
  - Utiliza o último valor salvo na pilha como endereço de retorno
    - Remover o valor da pilha

Função

- Vamos criar uma função calcula7segs
  - Recebe um byte indicando o número que desejamos exibir em um display de 7 segmentos
  - Retorna 1 byte com o valor necessário que deve ser enviado para qualquer porta onde o display está conectado
    - O valor acende/apaga os legs corretos do display

Parâmetros e retorno

- Temos um parâmetro de 1 byte
  - R25 (o valor que desejamos exibir)
- Temos um retorno de 1 byte
  - R25 (os zeros e uns que vamos enviar para alguma porta)

Esboço da função

```
calcula7segs:
    cpi r25,0 ;compare r25 com o imediato 0
    breq zero7Segs ;salte se igual
    cpi r25,1
    breq um7Segs
erro7Segs:
    ldi r25, 0b01100011
    ret
zero7Segs:
    ldi r25, 0b00111111 ; representação do zero em r25
    ret ;retorne ao chamador
um7Segs:
    ldi r25, 0b00000110
    ret
```

Esboço da função

Primeiro comparamos, depois realizamos um salto de acordo com a comparação. O resultado de uma comparação é armazenado em flags de um registrador de status. Mesma estratégia do x86.

```
calcula7segs:
    cpi r25,0 ;compare r25 com o imediato 0
    breq zero7Segs ;salte se igual
    cpi r25,1
    breq um7Segs
erro7Segs:
    ldi r25, 0b01100011
    ret
zero7Segs:
    ldi r25, 0b00111111 ; representação do zero em r25
    ret ;retorne ao chamador
um7Segs:
    ldi r25, 0b00000110
    ret
```

UDESC

Programa principal

```
#constantes
.equ DDRB, 0x04
.equ PORTB, 0x05


.equ RAMEND, 0x8FF
.equ SPH, 0x3E
.equ SPL, 0x3D


     .global  main
     .type main, @function
main:
     ;ajuste do stack pointer (por precaução)
     ldi R16, hi8(RAMEND)
     ldi R17, lo8(RAMEND)
     out SPL, R17
     out SPH, R16
     ;fim do ajuste do stack pointer


     ldi r16,0b11111111 ;carrega 0 para r16
     out DDRB,r16 ;Todos pinos de PB como saída
LOOP:
     ldi r25,1 ; exibir o número 1
     rcall calcula7segs ;salto curto relativo ao PC
     out PORTB,r25 ; carrega o retorno para PORTB
     rcall delay_omic ; delay
     jmp LOOP
```

Exercício

- Complete a função calcula7segs para que ela seja capaz de gerar os sinais para qualquer valor entre 0-9. Submeta no Moodle o seu programa (no “main”, você pode chamar a função passando um número qualquer) e também uma foto ou vídeo do seu circuito exibindo algum valor no display de 7 segmentos.

Referências

- S. Naimi, S. Naimi, M. Mazidi. **The Avr Microcontroller and Embedded Systems Using Assembly and C.** 2010**.**
- **megaAVR® Data Sheet.** Microchip, 2018.
- **ATmega328P Automotive - Complete Datasheet**. Microchip.
- **AVR Instruction Set Manual**. Microchip, 2016.
- D. Patterson; J. Henessy. **Organização e Projeto de Computadores**: **a Interface Hardware/Software.** 5a Edição. Elsevier Brasil, 2017.