Mestrado Integrado em Eng. de Comunicações

Stack

Portos I/O

Microprocessadores I

2º Ano – A12

Stack Pointer Register

SP

- *Registo de 8 bits cujo endereço é 81H (SFR RAM interna)*
- *Contém o endereço do item colocado no topo da pilha (stack). Valor após reset, 07H;*
- *Duas instruções permitem manipular a stack: PUSH e POP*
  - *A operação de PUSH coloca um dado na stack, enquanto a operação de POP retira o dado*

Stack Pointer Register

- *A stack cresce no sentido ascendente da memória*
- *Após o estado de reset este registo aponta para a posição 07H*

Pergunta:
Ao fazer o primeiro PUSH, onde é colocado o dado?

Stack Pointer Register

Atenção:

- *Como a stack após o reset está localizada na posição 7H, que corresponde a zona dos bancos de registos, é conveniente mudar a sua localização caso queiramos usar os bancos.*

  Por ex.: mov SP, #7FH

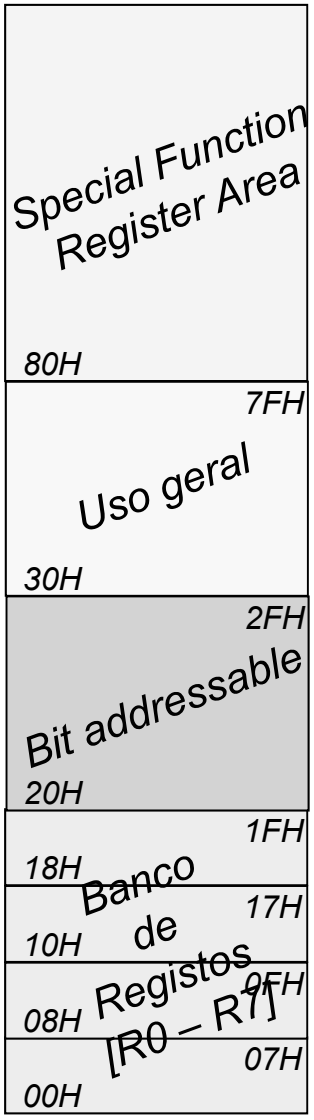
Stack Pointer Register

Atenção: Exemplo

Nota #2:
A pilha pode ser inicializada para a posição 7FH.
Como o acesso é indirecto através do SP, a pilha é implementada na IDATA e o espaço de SFR está salvaguardado.

Perigo #1:
Ao ser colocado valores na pilha o segundo banco será afectado

Posição apontada por SP após reset – 07H



Exemplo:
Inicializamos a SP para

mov SP, #7FH

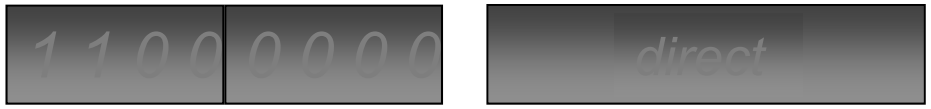
Stack Pointer Register

- ***Stack***
  - *O conjunto de instruções do 8051 fornece duas operações para manipulação da stack:*
    - ✓ ***push: insere um dado/valor na stack***
    - ✓ ***pop: retira o último dado/valor inserido na stack***

Stack Pointer Register

- ***Stack: PUSH***

2 bytes
2 cycles



- *Na operação de PUSH,*
  - *O valor do stack pointer é incrementado em uma unidade.*
  - *O conteúdo da posição da RAM indicada como argumento da operação é copiado para a posição apontada pelo stack pointer.*
  - *Nenhuma flag é afectada.*

$$SP \longleftarrow SP + 1$$
$$(SP) \longleftarrow (direct)$$

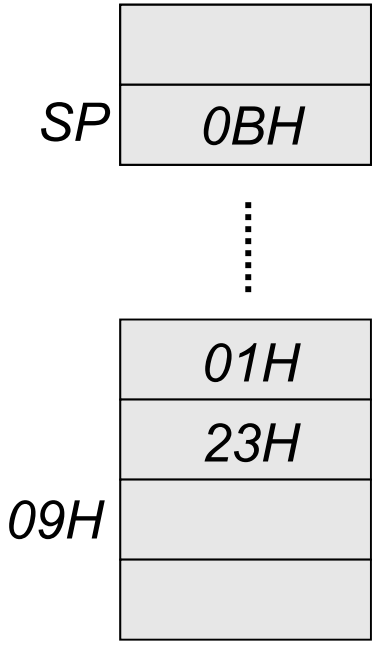
Stack Pointer Register

- ***Stack: PUSH - exemplo***

Um fragmento de programa tem o registo DPTR inicializado a 0123H e o stack pointer aponta para a posição 09H.
Explique qual é o estado da stack após o push do registo DPTR.

Programa:

```
PUSH    DPL
PUSH    DPH
```



Stack Pointer Register

- ***Stack: POP***

2 bytes
2 cycles

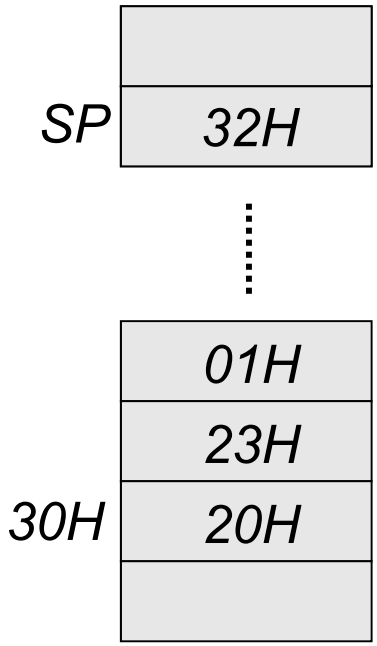
- *Na operação de POP,*
  - *O conteúdo da posição da RAM interna apontada pelo stack pointer é lido e o valor do stack pointer é decrementado.*
  - *O valor lido é carregado na posição da RAM indicada como argumento da operação.*
  - *Nenhuma flag é afectada.*

$$(direct) \longleftarrow ((SP))$$
$$(SP) \longleftarrow (SP) - 1$$

Stack Pointer Register

- ***Stack: POP - exemplo***

Analise o seguinte fragmento de programa e explique que valores assumirão os registos DPH, DPL e SP após a sua execução.



Programa:

```
POP   DPH
POP   DPL
POP   SP
```

Stack Pointer Register

- STACK – porquê?

Ao programar necessitamos de utilizar rotinas;

A invocação de rotinas no 8051 é feita utilizando as instruções **ACALL** ou **LCALL**;

O retorno de uma rotina é feito usando a instrução **RET**;

O que faz o 8051 ao executar uma instrução de CALL?

Guarda na *Stack* o valor do *Program Counter* (PC: PCH e PCL), que é o endereço na memória de código da próxima instrução a executar (instrução imediatamente após o CALL);

Carrega para o PC o endereço da rotina – começa a executar as instruções da rotina;

Ao executar a instrução RET, carrega para o PC o valor armazenado na *Stack*, ou seja, o endereço da instrução seguinte ao CALL.

Stack Pointer Register

- ***Directiva 'USING'***

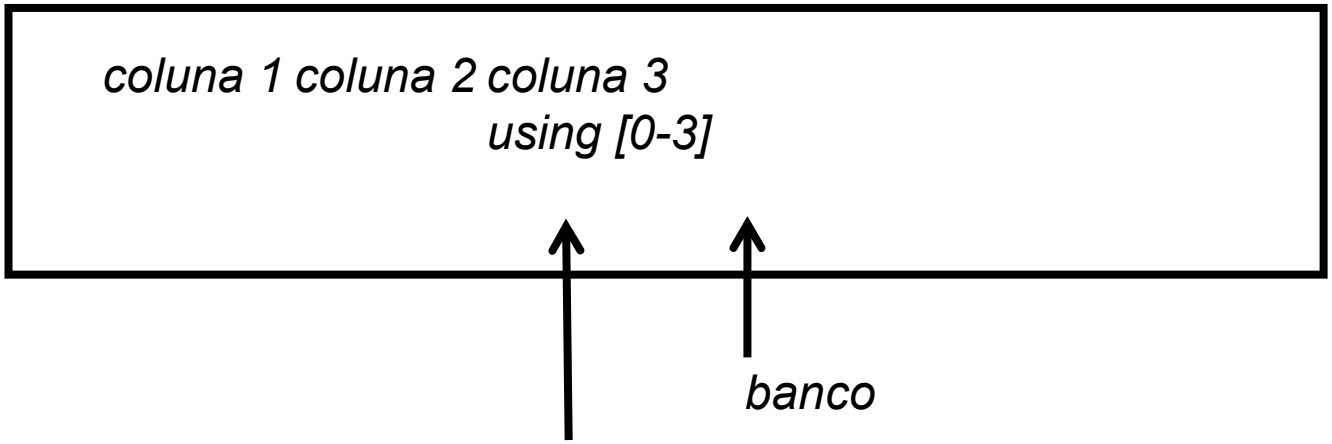
  - *As instruções de push e pop recebem como argumento o endereço da posição da memória interna cujo conteúdo será guardado na stack.*

  - *Isto significa que teremos que calcular o endereço dos registos Rn, n $\in$ [0-7], quando estivermos a trabalhar com diferentes bancos.*

  - *O assembler do 8051 fornece a directiva USING para facilitar a programação.*

Stack Pointer Register

- ***Directiva 'USING'***



- *No código uso as labels ARn, n ∈ [0 – 7], para os registos Rn, n ∈ [0 – 7], ou ACC para representar o acumulador, A.*

Stack Pointer Register

- ***Directiva 'USING': Exemplo***

```
using 3
push AR0

using 1
push AR7
```

- *O primeiro push colocará o conteúdo da posição 18H na stack (endereço de R0 no banco 3).*
- *O segundo push colocará o conteúdo da posição 0FH na stack (endereço de R7 no banco 1).*

Nota:

- *Antes de usar a directiva 'using' deve-se comutar de banco através da programação do registo PSW.*

Stack Pointer Register

- ***Directiva 'USING': Exemplo completo***

```
mov PSW, #00011000B     ; Comuta para o banco 3
using 3
push AR0

mov PSW, #00001000B     ; Comuta para o banco 1
using 1
push AR7
```

Nota:

- *Antes de usar a directiva 'using' deve-se comutar de banco através da programação do registo PSW.*

Portos de Input/Output

- *A família MSC-51 possui quatro portos de I/O de 8-bit. Cada um possui uma latch (SFR: P0 a P3), um driver de saída e um buffer de entrada:*
  - **P0 (80h)***: Porto de I/O de 8-bit. Implementa o LSB (A7:A0) do barramento de endereços e o barramento de dados (D7:D0) – multiplexagem temporal – para interface à memória externa;*
  - **P1 (90h)***: Porto de I/O de 8-bit ;*
  - **P2 (A0h)***: Porto de I/O de 8-bit. Implementa o MSB (A15:A8) do barramento de endereços para interface à memória externa.*
  - **P3 (B0h)***: Porto de I/O de 8-bit. Vários pinos do porto têm funções específicas (alternativas), como p.ex. entrada de temporizadores/contadores e entrada de interrupções externas.*

P2:P0 – Barramentos

- Quando é endereçada memória externa, os *drivers* de saída de P0 e P2 são utilizados, bem como o *buffer* de entrada de P0;

- Os valores nos pinos dos dois portos são temporariamente substituídos pelo endereço (PC ou DPTR) e pelo valor a ler/escrever na memória (P0), se o endereço tiver mais de 8-bit, caso contrário os pinos de P2 continuam a ter o conteúdo do SFR P2;

- Durante o acesso à memória externa, o SFR P2 não é alterado, mas o SFR P0 fica todo a 1's;

- Terminada a operação sobre a memória externa, o conteúdo dos portos é recuperado e colocado nos pinos.

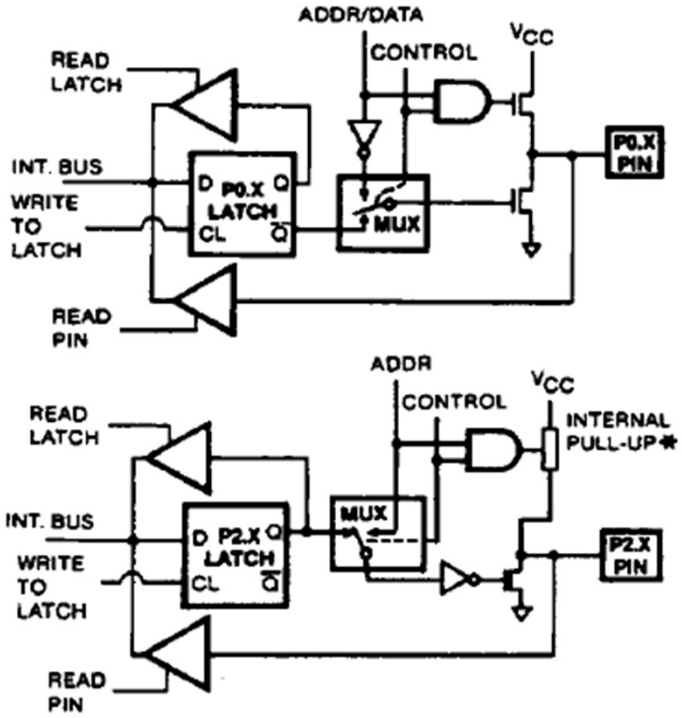
Funções alternativas do P3

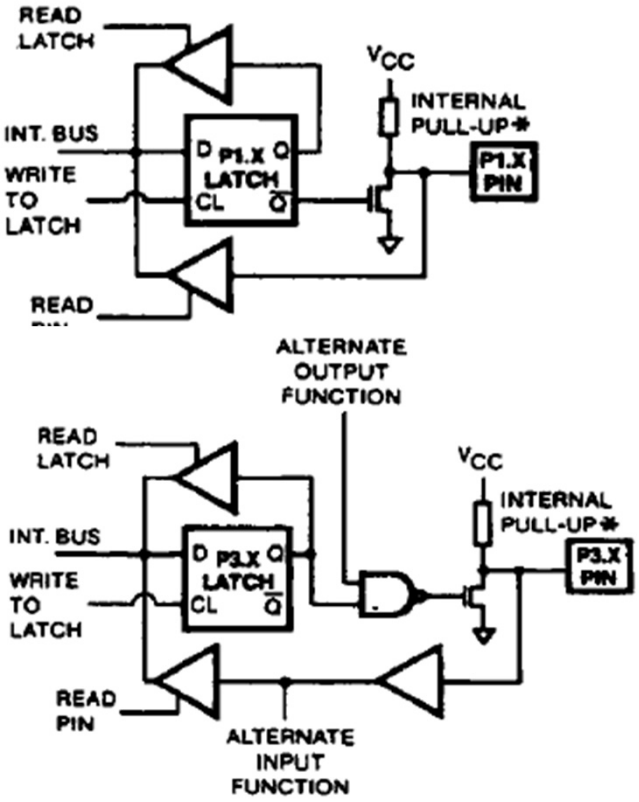
- P3.0 - RxD - Entrada Série;
- P3.1 - TxD - Saída Série;
- P3.2 - /INT0 - Interrupção externa 0;
- P3.3 - /INT1 - Interrupção externa 1;
- P3.4 - T0 - Entrada externa para o temporizador/contador 0;
- P3.5 - T1 - Entrada externa para o temporizador/contador 1;
- P3.6 - /WR - Sinal de controlo para operações de escrita na memória de dados externa;
- P3.7 - /RD - Sinal de controlo para operações de leitura na memória de dados externa;

As funções alternativas só podem ser activadas se na *latch* do SFR o bit correspondente for colocado a 1, caso contrário o pino do porto mantém-se a 0.

Portos: *driver* e *buffer*

- Os portos P1 a P3 possuem resistência de *pull-up* interna, P0 é de colector aberto;
- Cada linha de I/O pode ser configurada como entrada ou saída (P0 e P2 não podem ser utilizados como I/O quando implementam os barramentos de endereços e dados);
- Para ser usado como entrada o bit da *latch* tem de ser colocado a 1 de modo a desligar o *driver* de saída, então para P1 a P3 o pino fica a 1, mas pode ser colocado a 0 ou 1 por uma fonte exterior.





Portos de Entrada/Saída

Configurar os pinos dos portos como entrada ou saída:

Saída: basta escrever 0 ou 1 no bit do SFR para que esse valor apareça no pino correspondente;

Entrada: é necessário desligar o *driver* de saída, para tal coloca-se a 1 o bit do SFR. Nos portos P1 a P3 o pino é colocado a $+V_{cc}$ e a fonte externa pode então forçá-lo à massa (0) ou a $+V_{cc}$(1) – portos quasi-bidireccionais. Os pinos de P0 quando configurados como entradas flutuam, porque não há *pull-up* interno, nesta condição P0 pode ser utilizado como entrada de alta-impedência.

Escrita no Porto

Algumas instruções usam a *latch* do SFR, enquanto outras usam o estado do pino. As instruções que usam o bit da *latch* são aquelas que lêem o valor da *latch*, alteram-no de acordo com a instrução e reescrevem-no de novo para a *latch* (*read-modify-write*).

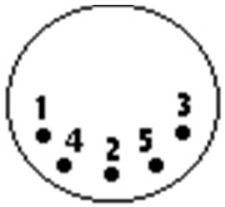
- **ANL:** ANL P0,A
- **ORL:** ORL P1,A
- **XRL:** XRL P2,A
- **JBC:** JBC P3.0,LB
- **CPL:** CPL P1.1
- **INC:** INC P0
- **DEC:** DEC P1
- **DJNZ:** DJNZ P2,*label*
- **MOV PX.Y,C:** MOV P3.2,C
- **CLR PX.Y:** CLR P0.3
- **SETB PX.Y:** SETB P1.4

- As três últimas instruções da lista da esquerda também provocam a *read-modify-write* na *latch*. Estas lêem o byte do porto, todos os 8-bit, modificam o bit em causa e escrevem o novo byte para a *latch*.
- As instruções *read-modify-write* são direccionadas à *latch* e não ao pino de modo a evitar uma possível má interpretação do nível de tensão no pino. O pino pode estar a ser utilizado para "atacar" a base de um transístor, quando um 1 é escrito no pino, o transístor liga. Se o micro lesse o mesmo bit no pino iria ler a tensão na base do transístor e interpretá-la como um 0. Ao ler a *latch* em vez do pino garante-se que o valor correcto, 1, é retornado.

Teclado PS/2

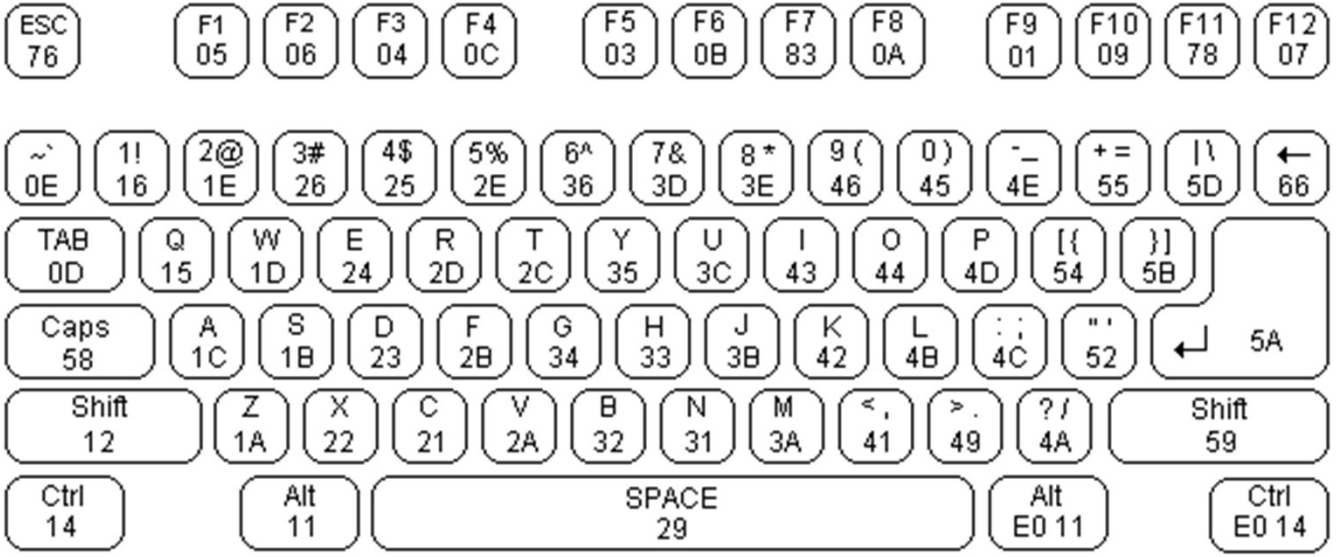
- Este tipo de teclados são baratos e muito populares devido à sua utilização em PCs. O seu preço é comparável ao de um mini-teclado de 16 teclas;
- Nos modelos PS/2, o interface entre um teclado e um PC é implementado através de fichas DIN. Estas são ilustradas na figura:

- ***Din de 5-pinos (AT/XT)***
  1. *Clock – relógio;*
  2. *Data – dados;*
  3. *NC – não conectado;*
  4. *Ground – massa/terra;*
  5. *Vcc (5 Volt).*



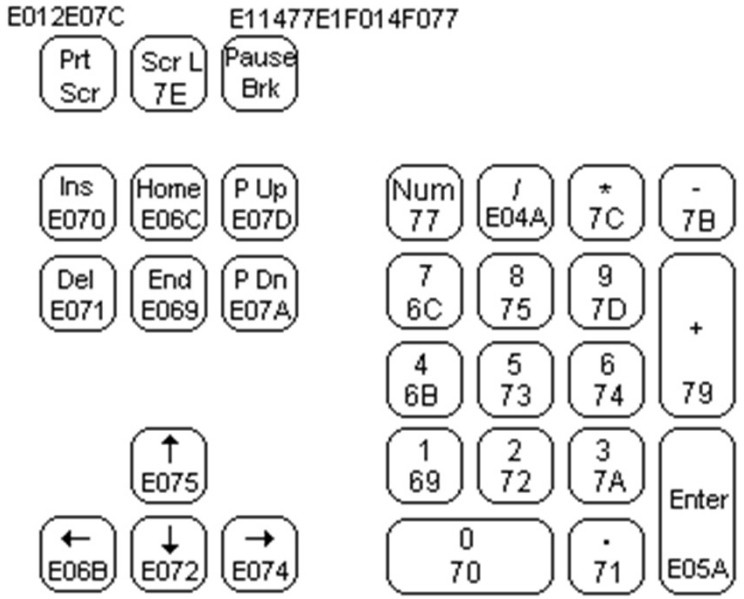
- ***Mini-Din de 6-pinos (PS/2)***
  1. *Data – dados;*
  2. *NC – não conectado;*
  3. *Ground – massa/terra;*
  4. *Vcc (5 Volt);*
  5. *Clock – relógio;*
  6. *NC – não conectado.*

Teclado PS/2 – Scan Codes



- ***Sequência enviada pelo teclado (em hexadecimal):***
  - *Nas teclas "normais", o código da tecla é enviado quando a tecla é pressionada e o byte 0xF0 mais o código da tecla é enviado quando a tecla é liberta.*
  - *Tecla "A": 0x1C (tecla pressionada) 0xF0 0x1C (tecla liberta);*
  - *Tecla "1": 0x16 (tecla pressionada) 0xF0 0x16 (tecla liberta);*

Teclado PS/2 – Scan Codes

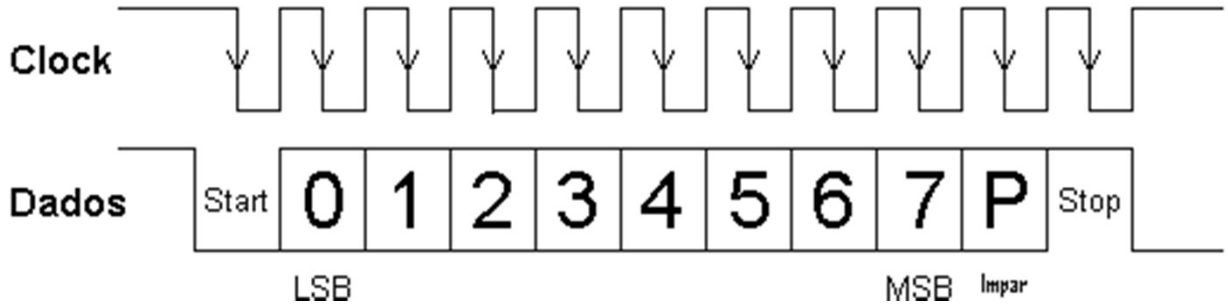


- ***Sequência enviada pelo teclado (em hexadecimal):***
  - *Nas teclas "especiais", o código da tecla (2 bytes, sendo que o primeiro é 0xE0) é enviado quando a tecla é pressionada e os mesmos dois bytes mais o byte 0xF0 no meio são enviados quando a tecla é solta.*
  - *Tecla "Ins": 0xE0 0x70 (tecla pressionada) 0xE0 0xF0 0x70 (tecla liberta);*
  - *Tecla "End": 0xE0 0x69 (tecla pressionada) 0xE0 0xF0 0x69 (tecla liberta);*

Teclado PS/2 - Comandos

- Os teclados PS/2 podem receber comandos (fora do âmbito do projecto) e enviam também bytes que permitem determinar o estado do teclado:
  - 0xFA – *Acknowledge* – confirmação;
  - 0xAA – *Power on* – teclado ligado, teste inicial OK;
  - 0xEE – *Echo* – faz o eco aos comandos recebidos;
  - 0xFE – *Resend* – pedido de reenvio de um comando;
  - 0x00 e 0xFF – *Error* – Erro ou *overflow* no buffer do teclado;

Teclado PS/2 - Interface



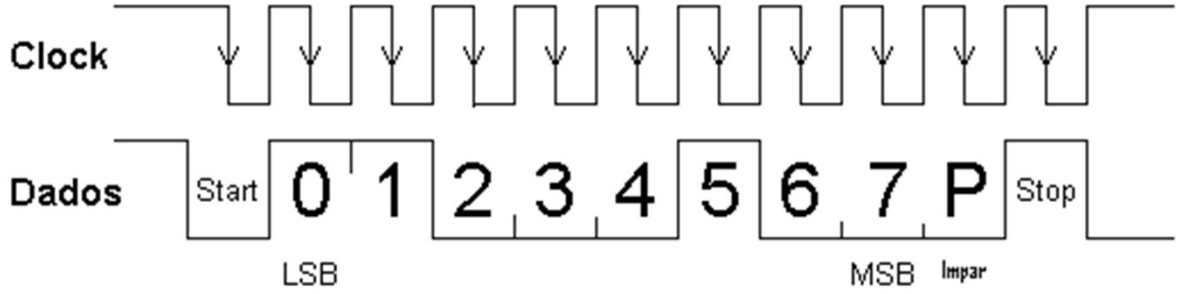
- *Em repouso a linha de relógio e de dados encontram-se a "1", estado iddle;*
- *Quando uma tecla é premida o teclado fixa a linha de dados a "0" e gera um bordo descendente na linha de relógio (transição de "1" para "0"). A linha de dados é mantida a "0" pelo menos durante 30µs. Esta é a condição de início (start);*
- *O primeiro bit a transmitir (LSB) é colocado na linha de dados, após o que o teclado gera outra transição descendente na linha de relógio. Esta operação é repetida até ser transmitido o bit 7 (MSB);*
- *O bit de paridade é calculado e enviado. A paridade é ímpar, o que significa que o número total de "1"s (b0-b7 mais o bit de paridade) enviados tem de ser ímpar.*
- *Depois o bit de stop, sempre igual a "1" é enviado e as duas linhas são colocadas no estado de iddle;*
- *A frequência do sinal de clock durante uma transmissão está próxima de 15.8KHz.*

Teclado PS/2 - Interface

Exemplo

Enviar tecla "D" pressionada

Código da tecla=0x23=00100011b



Teclado PS/2 - TPC

- Fazer um programa que leia um código do teclado (um byte:k7-k0), que coloque verifique se a tecla é um dígito e caso seja o coloque no *display* de 7-segmentos. Se não for dígito deve acender o "*dot*" do *display;*
- Deverá armazenar na memória de código duas tabelas, uma com os dígitos a colocar no *display* de 7-segmentos e outra com o código das teclas de 0 a 9 do teclado;
- O programa tem de utilizar dois pinos de I/O como entradas. Sugere-se o P1.0 para entrada de relógio e o P1.1 para entrada de dados;
- Os bits de *start*, *stop* podem ser ignorados. A paridade que no protocolo é ímpar tem de ser verificada.