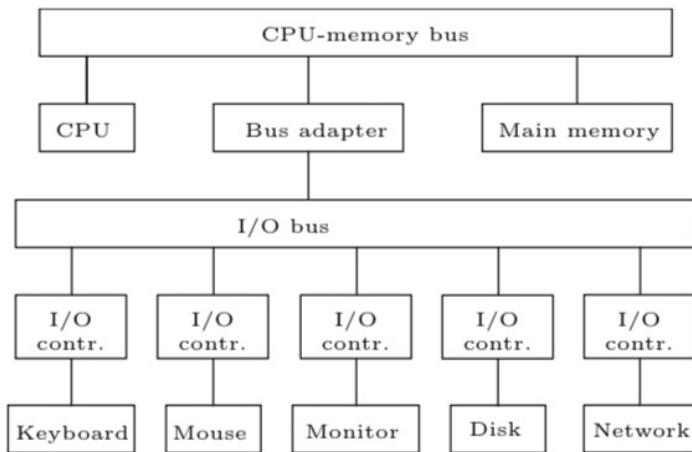


Introdução



Component	Hours/Week	No. Weeks	Total
Lectures	2	11	22
Lab classes	2	13	26
Prep. L0	4	1	4
Prep L2-L5	8	8	64
Proj.	8	5	40
Total			156

Lab	First lab class	Topic
Lab 0	13-02	VBox and Minix
Lab 2	20-02	Timer
Lab 3	06-03	Keyboard
Lab 4	13-03	Mouse
Lab 5	27-03	Video (graphics)

What	Week (starting on)
Proposal	17-04
Submission	26-05
Demonstration	First week of exams

Dispositivos I/O

- São a interface entre o CPU e o mundo exterior
praticamente independente do dispositivo

Controlador/Adaptador: componente eletrônico que controla um dispositivo I/O
REGISTROS

- 1. Control: faz operações de I/O
- 2. Status: obter o estado do dispositivo ou operações de I/O pendentes
- 3. Data: transferir dados de/para o dispositivo I/O
INPUT/OUTPUT

Nota: os nomes dos registros não são do ponto de vista do dispositivo de I/O, não são do CPU

Como é que o CPU acede a um controlador de I/O?

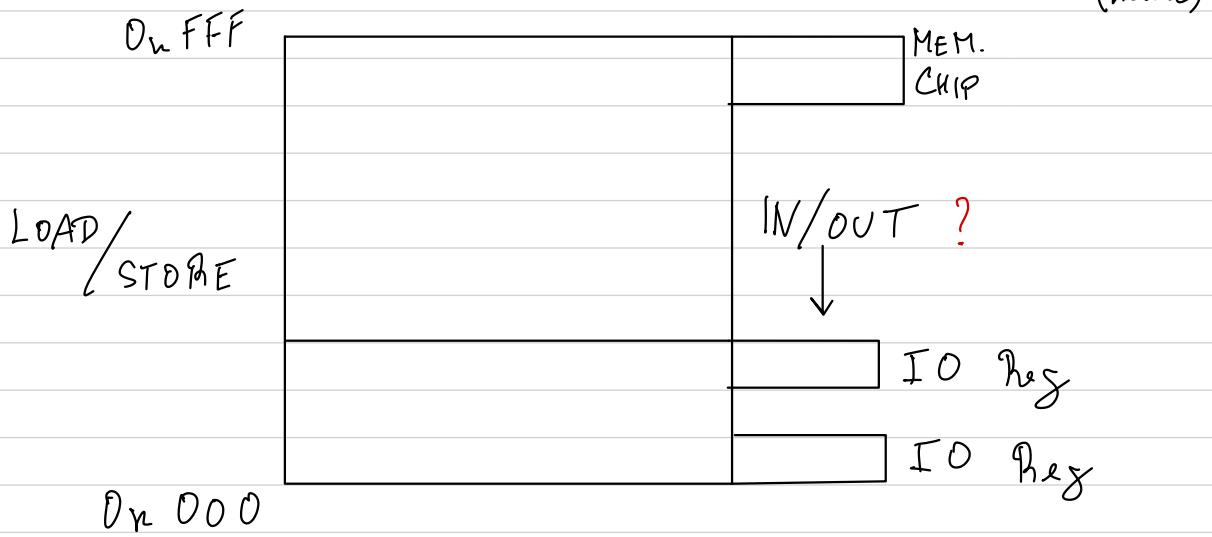
1. Memory-Mapped I/O: posições de endereço de endereçamento são atribuídas a dispositivos de I/O e o acesso a cada dispositivo de I/O é feito usando as instruções de acesso a memória do CPU

qualquer arquitetura de processador

2. Instruções de I/O especiais: o I/O usa um espaço de endereçamento diferente e a cada dispositivo o I/O é atribuída uma posição. Dene espaço de endereçamento ↴

Intel ✓ + CPU deve fornecer instruções especiais para aceder ao espaço de endereçamento I/O — instruções de I/O — que só não podem ser executadas em modo privilegiado (kernel)

ARM ✗



Intel: um porto é uma abstracção do registo do controlador de dispositivo

Instrução IN: input do porto - ler registo de I/O

Instrução OUT: output para o porto - escrever registo de I/O

Minix 3 & Virtual Box

Problema: C não fornece instruções para executar I/O

Solução: Minix 3 SYS_DEVIO

#include <minix/syslib.h>

```
int sys_inb(int port, u32_t *byte);  
int sys_outb(int port, u32_t byte);
```

Endereço de 32 bits interno seminal

→ implementa util_sys_inb (int port, u8-t *byte)

Notas:

- Os processos do usuário não podem diretamente aceder hardware por razões de fiabilidade e segurança
- As system calls do sistema operativo são de mais alto-nível do que a interface programática dos dispositivos IO
- A Kernel implementa as system calls e gere o hardware
- Uma estrutura em camadas permite decomposição e modularidade

KERNEL MONOLÍTICA

serviços no núcleo do kernel

V/S.

MICRO-KERNEL

serviços como módulos

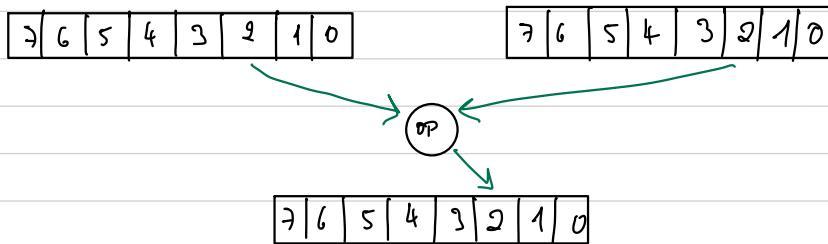
Conclusão: no ma Minix 3, processos que não correm em modo kernel podem usar a interface programática de dispositivos IO

C

- C é um subconjunto de C++

I/O em C: `stdin`; `stdout`; `stderr`; `printf`
(stdio.h)

Operações em Bits: operações booleanas, binárias ou unárias, sobre cada bit de operadores inteiros



&	AND
	OR inclusivo
^	OR exclusivo
~	complemento para um

CONJUNÇÃO
DISJUNÇÃO
DISJUNÇÃO EXCLUSIVA
NEGAÇÃO

Ex:

uchar mask = 0x80 // 1000 0000 b

```
if (flags & mask) // testa bit mais significativo
    flags |= mask; // ativa bit mais significativo
    flags ^= mask; // altera bit mais significativo
    flags &= ~mask; // resta bit mais significativo
```

Shift: OPERANDO OPERADORA n BITS

LEFT SHIFT $<<$: bits libertados à direita param a 0
RIGHT SHIFT $>>$: bits libertados à esquerda param a 0
Se o operando for nem binário, não?

- Multiplicação / Divisão de inteiros por 2^n

! // define SQRT_WAVE_BIT 0 1
! // define SQRT_WAVE_BIT 1 2
! // define BIT (n) ($0 \ll (n)$)

Convenção de Inteiros: operandos de operadores aritméticos/lógicos cujo tipo é inferior a `int` não são promovidos a `int` antes de realizar a operação! Solução: usar `(cast)`

Nota: usar `<std::int.h>` que define tipo de inteiros

A pontador: tipo de dado cujo valor é um endereço de memória
`int *p;` // o apontador p aponta para inteiros
`p = &n;` // p aponta para o endereço de n
`*p = 8;` // 8 passa a ser o valor da variável

Struct: contém espaço para guardar todos os seus membros simultaneamente

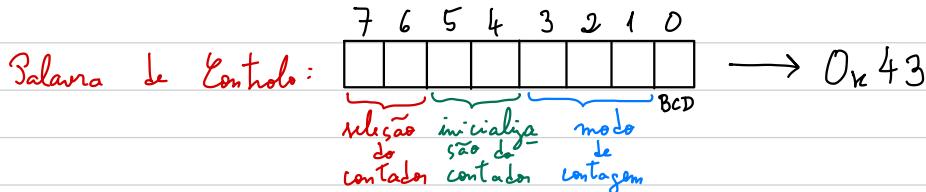
Union: contém espaço para guardar qualquer um dos seus membros de cada vez

i8254 Timer/Counter

- Temporizador / Contador programável

- 3 contadores de 16-bit, decrescentes, a partir de um valor carregado
- 6 modos de contagem que determinam o que acontece quando o contador é 0

Modo 3 - Geração de Onda Quadrada: OUT é alto durante metade do ciclo e baixo durante a restante metade do ciclo



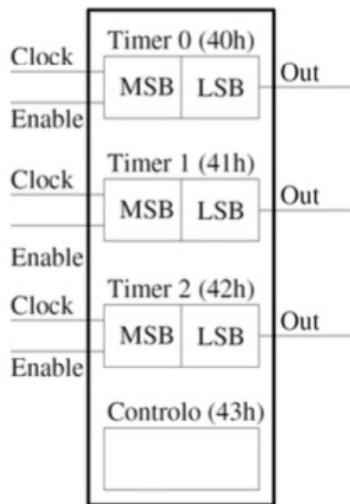
- O valor inicial da contagem deve ser escrito no ponto do contador correspondente
- Por razões de compatibilidade, os bits "don't care" (X) devem ser 0

Read-Back Command: permite ler a configuração programada e/ou o valor de contagem atual

↓
usar macros ou operações bit-a-bit (máscara/mshift)

- É necessário escrever no registo de controlo antes de aceder aos timers
- Ao iniciar, Minic 3 programa o Timer 0 para gerar uma onda quadrada de frequência fixa, usando essa interrupção para medir o tempo, incrementando numa variável global a um ritmo fixo e conhecido

$$f_{\text{entrada}} = 1193181 \text{ Hz}$$



Bit	Value	Function
7,6		Counter selection
	00	0
	01	1
	10	2
5,4		Counter Initialization
	01	LSB
	10	MSB
	11	LSB followed by MSB
3,2,1		Counting Mode
	000	0
	001	1
	x10	2
	x11	3
	100	4
	101	5
0		BCD
	0	Binary (16 bits)
	1	BCD (4 digits)

Read-Back Command Format

Bit	Value	Function
7,6		Read-Back Command
	11	
5		COUNT
	0	Read counter value
4		STATUS
	0	Read programmed mode
3		Select Timer 2
	1	Yes
2		Select Timer 1
	1	Yes
1		Select Timer 0
	1	Yes
0		Reserved

)— ativos em 0

Read-Back Status Format

Bit	Value	Function
7		Output
6		Null Count
5,4		Counter Initialization
3,2,1		Programmed Mode
0		BCD

Interrupts

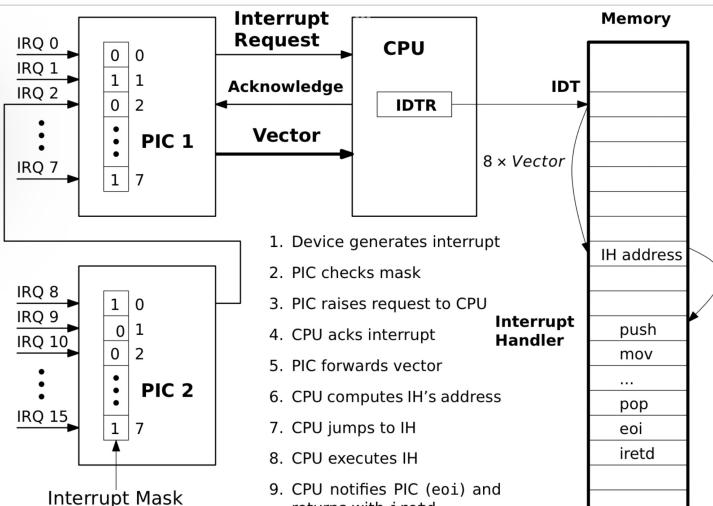
- As operações de I/O (lentas) não envolvem com as do processador (rápida)

Variamento: o processador varre o dispositivo I/O, isto é, lê um registo de estado

Interrupção: o dispositivo I/O notifica o processador através de mecanismos de interrupções

Priority Interrupt Controller (PIC)

PIC 1	Dispositivo	Valor
IRQ 0	Timer	0x08
IRQ 1	Teclado	0x09



Linha IRQ: determinada pelo designer de hardware

Vector: pode ser configurado na inicialização, configurando o PIC e o IDT (Interrupt Descriptor Table)

Interrupt Handler (IH): rotinas de serviço de interrupções (ISR) e parte do respetivo "device driver" que são executadas pelo hardware quando de uma interrupção e podem armazenar

Misc 3: "device drivers" não implementados como programas ao nível do utilizador e não da Kernel o que exige que só o mínimo seja feito ao nível da Kernel (GIH) e as operações específicas do dispositivo sejam realizadas por ele ao nível do utilizador

Generic Interrupt Handler (GIH): quando de uma interrupção

1. Notifica todos os "device drivers" interessados na interrupção
2. Se possível, reconhece a interrupção emitindo o comando EOI ao PC
3. Emite a instrução IRET

Como é que o GIH sofre (1)? O DD diz, usando a kernel call que pode ser vista como uma subscrição de notificações de interrupções:

linha IRQ do dispositivo
int sys_irqsetpolicy (int irqline, int policy, int *hook_id)

↳ input: id para kernel
↳ output: id para DD

IRQ_PENABLE informa GIH que pode dar EOI

Como é que o GIH notifica DD? Usa o mecanismo standard de comunicação entre processos (IPC): notificações (tipo especial de mensagens)

Como é que o DD recebe a notificação? IPC e biblioteca libdrivers

if (msg.m_notify.interrupts & irq_set) { ... }

valor baseado em hook_id (input):
bit corresponde à interrupções pendentes

↳ máscara para testar interrupção pendente

E se o GIH não envia EOI? Se um DD não define a política IRQ_PENABLE no seu pedido de subscrição de interrupções (sys_irqsetpolicy), o DD tem de o fazer assim que possível através da chamada sys_irqenable (int *hook_id)

sys_irqsetpolicy (int *hook_id): anula a subscrição de uma notificação de interrupção

sys_irqenable (int *hook_id): mascara a linha de interrupções associada com uma subscrição

LAB 2

1. timer-test-read-config (uint8_t time, enum timer-status-field field)

a. timer-get-conf (uint8_t time, uint8_t *st)

- i. sys-outb (int port, uint32_t value): value → port — (TIMER_CTRL, rb)
- ii. atl-sys-inb (int port, uint8_t *value): value → port — (TIMER_PORT, st)
- sys-inb (int port, uint32_t *value): port → value

b. timer-display-conf (uint8_t time, uint8_t st, enum timer-status-field field)

- i. timer-print-config (uint8_t time, enum timer-status-field field, union timer-status-field-val val)

2. timer-test-time-base (uint8_t time, uint32_t freq)

a. timer-set-frequency (uint8_t time, uint32_t freq)

i. TIMER_MIN_FREQ ; TIMER_FREQ

ii. timer-get-conf (uint8_t time, uint8_t *st)

iii. sys-outb (int port, uint32_t value): value → port — (TIMER_CTRL, cwe)

iv. value = TIMER_FREQ / FREQ

v. atl-get-LSB (uint16_t value, uint8_t *lb)

vi. atl-get-HSB (uint16_t value, uint8_t *msb)

vii. sys-outb (int port, uint32_t value): value → port — (TIMER_PORT, lsb)

viii. sys-outb (int port, uint32_t value): value → port — (TIMER_PORT, msb)

3. timer-test-int (uint8_t time)

a. timer-subscribe-int (uint8_t *bit_no): sig-set (bit mask)

i. *bit_no = hook_id

ii. sys-signet-policy (int sig_vec, int policy, int *sig_hook_id)

b. timer-int-handler (): counter ++

c. timer-print-delayed-time ()

d. timer-unsubscribe-int ()



id de DD para K id de K para DD

i8042 Keyboard

Scancode: normalmente, 1 byte colocado num buffer e enviado para o PC

Make Code: quando uma tecla é premida — MSB 0

Break Code: quando uma tecla é libertada — MSB 1

Nota: o primeiro byte de scancodes de 2 bytes é normalmente 0xE0

OUT-BUF: porto 0x60

1. KBC sinaliza que está vazio via "serial bus"
2. C@KBD envia o byte à cabeça do buffer para o KBC
3. KBC põe o byte em OUT-BUF
4. KBC gera uma interrupção "raising" IRQ1

OUT-BUF 0x60
STATUS-BEG 0x64 } sys-inb()

IN-BUF DATA 0x60
CTRL 0x64 } sys-outb(...)

- Para input ou output, é sempre necessário ler o status register
- Se o bit 1 estiver ativo, não escreve para o IN-BUF

PC-AT/PS2: 0x64: comandos 0x60: argumentos OUT-BUF: valores de retorno

KBC Command Syst:

—	—	DIS 2	DIS	—	—	INT 2	INT
7	6	5	4	3	2	1	0

DIS 2: desativa interface do rato

DIS: desativa interface do teclado

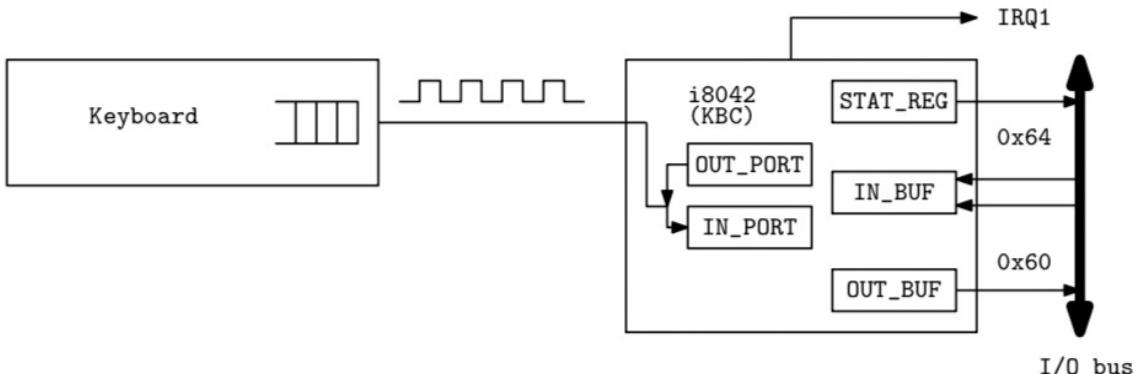
INT 2: ativa interrupções do rato em OBF

INT: ativa interrupções do teclado em OBF

• Uma resposta do comando pode ser "roubada" pelo IH — Desativa interrupções

• Respostas do KBD/KBC não são imediatas — o código tem de esperar

• Gestão de concorrência — assumi que os bytes gerados pelo KBD são diferentes das respostas



Bit	Name	Meaning (if set)
7	Parity	Parity error - invalid data
6	Timeout	Timeout error - invalid data
5	Aux	Mouse data
4	INH	Inhibit flag: 0 if keyboard is inhibited
3	A2	A2 input line: irrelevant for LCOM
2	SYS	System flag: irrelevant for LCOM
1	IBF	Input buffer full don't write commands or arguments
0	OBF	Output buffer full - data available for reading

Command	Meaning	Args (A)/ Return (R)
0x20	Read Command Byte	Returns Command Byte
0x60	Write Command Byte	Takes A: Command Byte
0xAA	Check KBC (Self-test)	Returns 0x55, if OK Returns 0xFC, if error
0xAB	Check Keyboard Interface	Returns 0, if OK
0xAD	Disable KBD Interface	
0xAE	Enable KBD Interface	

7	6	5	4	3	2	1	0
-	-	DIS2	DIS	-	-	INT2	INT

LAB 3

lab3.c

1. Kbd-test-scan()

- a. Keyboard-subscribe-int (& kbd-req-set)
- b. kbc-ih()
- c. Kbd-print-1scan-code (code.makecode, code.size, code.bytes)
- d. Keyboard-restore()
- e. Keyboard-unsubscribe-int()
- f. Keyboard-print-to-myself (cnt)

2. Kbd-test-roll()

- a. Keyboard-read-1scan-code-byte()
- b. Kbd-print-1scan-code (code.makecode, code.size, code.bytes)
- c. Keyboard-restore()
- d. Keyboard-print-to-myself (cnt)
- e. Keyboard-enable-interrupts()

3. Keyboard-test-timed-scan(n)

- a. Keyboard-subscribe-int (& keyboard-req-set)
- b. timer-subscribe-int (& timer-req-set)
- c. timer-int-handler()
- d. kbc-ih()
- e. Kbd-print-1scan-code (code.makecode, code.size, code.bytes)
- f. Keyboard-restore()
- g. Keyboard-unsubscribe-int()
- h. timer-unsubscribe-int()

Keyboard.c

1. Keyboard-subscribe-int (*bit_no) IRQ - EXCLUSIVE
 - a. sys-set-policy (KEYBOARD_IRQ, IRQ-REENABLE, &Keyboard-hook-id)
2. Keyboard-unsubscribe-int ()
 - a. sys-ingrm-policy (&Keyboard-hook-id)
3. Keyboard-read-scan-code-byte ()
 - a. Kbc-read-output (&output)
 - b. Keyboard-restore() / Keyboard-enable-interrupt()
4. Kbc-ih ()
 - a. Keyboard-read-scan-code-byte()
5. Keyboard-restore ()
6. Keyboard-enable-interrupt ()
 - a. Kbc-write-command (KBC-READ_CMD)
 - b. Kbc-read-output (&command-byte)
 - c. Kbc-write-command (KBC-WRITE_CMD)
 - d. Kbc-write-argument (command-byte | BIT(0))

Kbc.c

1. Kbc-read-status (*status)
 - a. util-sys-inb (KBC_STATUS_REG, status)
2. Kbc-read-output (*output)
 - a. Kbc-read-status (&status) /* loop while !DBF */
 - b. util-sys-inb (KBC_OUTT-BUF, output)
3. Kbc-write-command (command)
 - a. Kbc-read-status (&status) /* loop while !BF */
 - b. sys-outb (KBC_CMD_REG, command)
4. Kbc-write-argument (argument)
 - a. Kbc-read-status (&status) /* loop while !BF */
 - b. sys-outb (KBC-ARG-REG, argument)

Interrupções

hal.c:

1. wait8-t iq-set
2. subscribe-int (& iq-set)
3. & BIT (iq-set)
4. ih()
5. unsubscribe-int()

device.c:

1. hook-id = X // bit que vai ser ativo
2. subscribe-int (*bit_no)
3. ih()
4. unsubscribe-int()

subscribe-int (*bit_no):

1. *bit_no = hook_id
2. my-s-ing-set policy (IRQ, REENABLE, & hook_id)

unsubscribe-int():

1. my-s-ing rm policy (& hook_id)

my-s-ing-set policy (IRQ, REENABLE, & hook_id)

CALL

RETURN

"ativa este bit" (*bit_no = hook_id)

"des-subscribe-me com este id"

PS/2 Mouse

- O rato tem o seu próprio controlador e guarda o estado dos seus botões
- O movimento relativo à última posição do rato é registado em dois contadores de 9 bit (x, y) em complemento para dois
- O controlador envia esta informação para o PC num pacote de 3 bytes
- Um parâmetro de escala no controlador do rato afeta o valor dos contadores reportados pelo rato:

→ 1:1 os valores são os valores dos contadores
2:1 os valores não dão por uma função



Modo Stream: o rato envia os dados a uma velocidade máquina fixa programável
Modo Tremoto: o rato envia os dados só a pedido do KBC

Numa interrupção, é activa a linha IRQ 12 (IRQ 4 da PIC 2) uma vez por cada byte, pelo que o INT do rato deve ser um byte por interrupção

PC-AT/PS2: On64: comandos On60: argumentos OUT_BUF: valores de retorno

7	6	5	4	3	2	1	0
—	—	DIS 2	DIS	—	—	INT 2	INT

KBC Command Byte:

DIS 2: desativa interface do rato

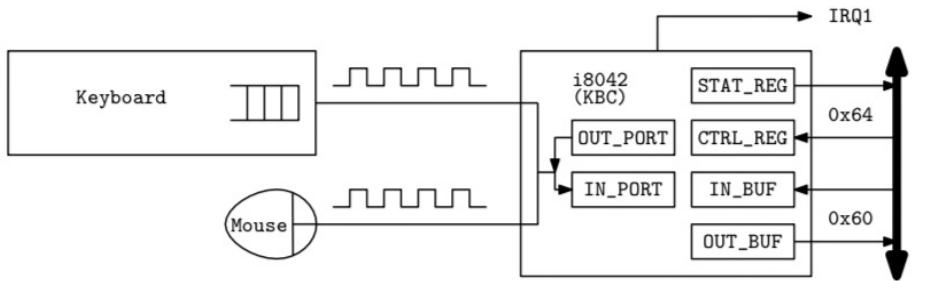
DIS: desativa interface de teclado

INT 2: ativa interrupção do rato em OBF

INT: ativa interrupção do teclado em OBF

Notas:

- escreva os comandos para KBC no porto On64
- argumentos e valores de retorno passados via porto On60
- não esqueça de verificar o bit IBF no STATUS_REG antes de escrever
- o comando OnD4 envia o seu argumento para o rato sem interpretação



	7	6	5	4	3	2	1	0
Byte 1	Y Ovfl	X Ovfl	MSB Y delta	MSB X delta	1	M.B.	R.B.	L.B.
Byte 2				X delta				
Byte 3				Y delta				

Command	Meaning	Args (A)/ Return (R)
0x20	Read Command Byte	Command byte (R)
0x60	Write Command Byte	Command byte (A)
0xA7	Disable Mouse	
0xA8	Enable Mouse	
0xA9	Check Mouse Interface	Returns 0, if OK
0xD4	Write Byte to Mouse	Byte (A)

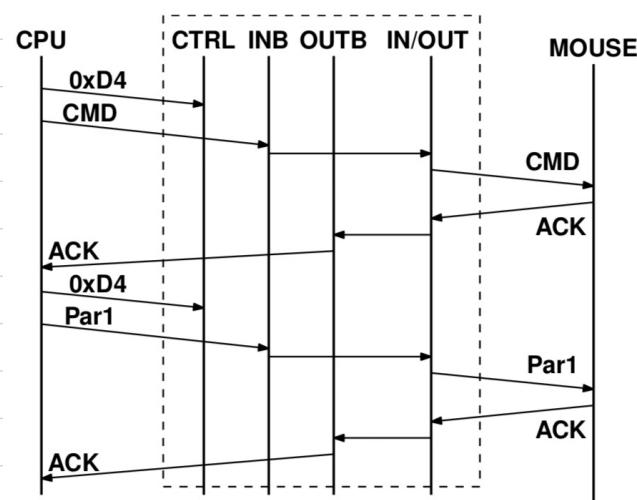
Bit	Name	Meaning (if set)
7	Parity	Parity error - invalid data
6	Timeout	Timeout error - invalid data
5	Aux	Mouse data
4	INH	Inhibit flag: 0 if keyboard is inhibited
3	A2	A2 input line: 0 data byte 1 command byte
2	SYS	System flag: 0 if system in power-on reset, 1 if system already initialized
1	IBF	Input buffer full don't write commands or arguments
0	OBF	Output buffer full - data available for reading

Comandos passados como argumento do comando OnD4

Command	Function	Description/Comments
0xFF	Reset	Mouse reset
0xFE	Resend	For serial communications errors
0xF6	Set Defaults	Set default values
0xF5	Disable (Data Reporting)	In stream mode, should be sent before any other command
0xF4	Enable (Data Reporting)	In stream mode only
0xF3	Set Sample Rate	Sets state sampling rate
0xF0	Set Remote mode	Send data on request only
0xEB	Read Data	Send data packet request
0xEA	Set Stream Mode	Send data on events
0xE9	Status Request	Get mouse configuration (3 bytes)
0xE8	Set Resolution	
0xE7	Set Scaling 2:1	Acceleration mode
0xE6	Set Scaling 1:1	Linear mode

Argumentos destes comandos também devem ser passados como argumentos do comando OnD4

OnD0
OUT-BUF { ACK: 0xFA → tudo OK
Respostas { NACK: 0xFE → byte inválido (pode ser erro de comunicação)
AOS BYTES ERROR: 0xFC → segundo byte inválido consecutivo



Quando o host recebe uma resposta 0xFE, deve tentar novamente o comando que originou o erro. Se um byte de argumento origina uma resposta 0xFE, o host deve retransmitir todo o comando, não só o byte do argumento.

O byte de reconhecimento não é a resposta ao comando! Para comandos com resposta, o controlador envia-a depois do último byte de reconhecimento.

Design Dirigido a Eventos

Evento: alteração no estado

Todo o processamento de I/O é dirigido por eventos (detetados por scan/poll)

Design Dirigido a Eventos:

- O controlo de fluxo é determinado pelo ambiente em vez do próprio programa
- O código é executado de forma reativa em resposta a eventos que podem ocorrer simultaneamente com a execução do programa

Filas de Eventos: permitem lidar com um evento aninhadamente à sua ocorrência

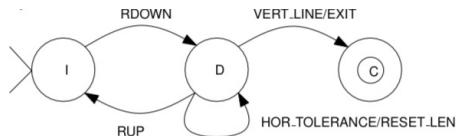
Handlers de Eventos: processam cada tipo de evento

Dispatchers: monitorizam as filas de eventos e chamam os handlers apropriados

Uma máquina de estados é dirigida a eventos — as transições de estado dependem da ocorrência de um evento

Eventos do dispositivo de I/O

Não processados pelos interrupt handlers correspondentes



Cur. State	Input	Next State	Output
I(initial)	RDOWN	D(rawing)	
D(rawing)	VERT_LINE	C(complete)	Exit
D(rawing)	RUP	I(initial)	
D(rawing)	HOR_TOLERANCE	D(rawing)	Reset length

Os IH podem ser dependentes da aplicação ou independentes da aplicação (neste caso, é necessário definir um evento handler dependente da aplicação e especificar a sua comunicação com o IH)

Em geral, IH podem ser reutilizados e introduzem um nível de indireção (podem adicionar flexibilidade, mas mais responsivos, requerem mais código ou são mais lentos)

Apoitadores para Funções

int (*fp) (int); declara fp como um apontador para uma função que toma um inteiro como argumento e retorna um inteiro

• Yendo int foo(int) numa função:

fp = foo; inicializa fp como apontador para foo

n = (*fp) (i); invoca a função apontada por fp, foo, com argumento i e atribui o valor de retorno à variável n

Ex: void (*cht[]) (void) = {ev0_handler, ev1_handler, ...}
(*cht[ev])();

Ex: void (*cht[][]) (void) = {{ev0_handler, ...}, {ev1_handler, ...}, ...};
(*cht[st][ev])();

LAB 4

- No IH, há só um byte do KBC
- Não é necessário verificar DBF nem AUX
- O KBC usa linhas IRQ diferentes para o teclado e o rato
- Atribuir pacotes usando:
 - Um array de 3 bytes para o pacote do rato
 - O índice da posição atual do array
- Através que quando se mostram os 3 bytes, todos pertencem ao mesmo pacote
- Se o dispositivo está em modo Stream (padronização) e foi ativado com o comando Enable (OnF4), então a ação deve desativar o dispositivo com um comando Disable (OnF5) antes de enviar qualquer outro comando.
- Se um byte ainda está no OUT-BUF, o KBC não vai gerar mais interrupções

Placa Gráfica

GPU: controla o hardware do ecrã e realiza algoritmos de "rendering"

ROM: inclui código que realiza operações lógicas de I/O de vídeo

Vídeo RAM: armazena a informação que é "rendered" no ecrã

Modo Texto: abstrai o ecrã como uma matriz de caracteres

Modo Gráfico: abstrai o ecrã numa matriz de pixels - pixels

RGB: (Red, Green, Blue) → intensidade de cada cor primária (N bits)

Modo Ponto de Cor: armazena a cor de cada pixel na VRAM

Cor Indicada: armazena um índice para uma Tabela - paleta/mapa de cores - com a definição, i.e., a intensidade das 3 cores primárias de cada cor

Modelo de Memória: determina como é que a memória de vídeo é organizada, isto é, onde é que o valor de cada pixel é armazenado na VRAM

Modo Linear:

- SABER** {
- o endereço base do "frame buffer"
 - as coordenadas do pixel
 - o número de bytes usados para codificar a cor

Problema: Como configurar o modo gráfico pretendido?

NÃO: ler/escrever diretamente nos registos da GPU

Solução: usar a VESA Video Bios Extension (VBE)

BIOS: Basic Input-Output System

1. Uma interface de firmware usada para acesar os recursos de hardware do PC
2. A implementação desta interface
3. A memória não volátil (ROM) que contém esta implementação

- O acesso a serviços da BIOS é via a interrupção de interrupções SERVIÇO INT XX
- Os argumentos são passados via registos do processador

Problema: Como fazer uma chamada à BIOS em Minix?

Solução: usar a chamada à Kernel de Minix 3 SYS_INT86 - Faz uma chamada em tempo real à BIOS em vez de um device driver no espaço do utilizador. Isto troca temporariamente do modo protegido de 32 bits para o modo real de 16 bits para acesar as chamadas à BIOS.

VBE: Video BIOS Extension

- A especificação da BIOS só suporta modos gráficos VGA (Video Graphics Adapter)

Registo AH: 0x4F

Registo AL: função

AH	AL
AX	

VRAM:

Antes de escrever para o frame buffer:

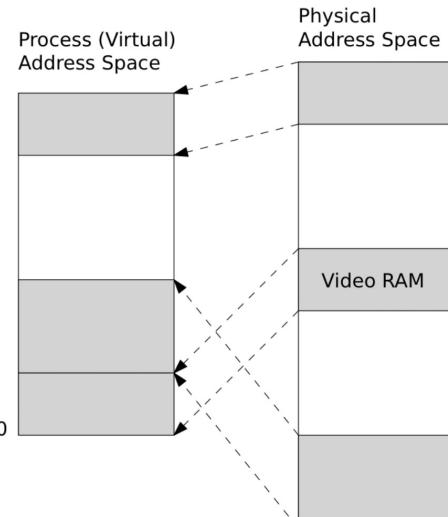
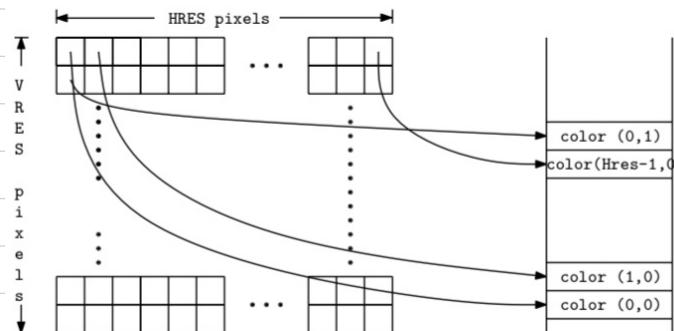
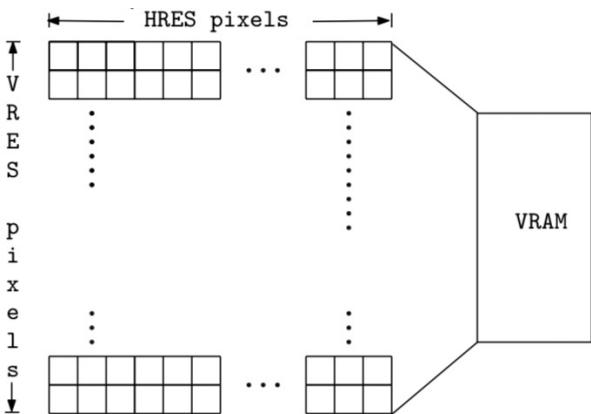
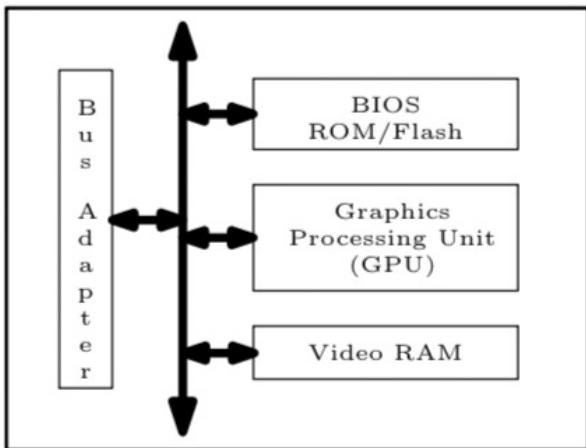
1. Usar vbe_get_mode_info (uint16_t mode, vbe_mode_info_t *vinfo)
2. Mapar a região de memória física no espaço (virtual) de endereçamento do processador

- A maioria das arquiteturas de computadores suportam um espaço de endereçamento virtual que é "decodificado" do espaço de endereçamento físico

- Cada processo tem o seu próprio espaço de endereçamento virtual
- O SO mapeia regiões da memória física no computador para os espaços de endereçamento virtual dos diferentes processos
- É preciso mapear a Video RAM para o espaço de endereçamento virtual!

Problema: Dado um endereço virtual, como pode um programa aceder a memória física mapeada para esse endereço virtual?

Solução: Usar apontadores (com atenção para o tamanho de um pixel)



Interrupt vector (xx)	Service
10h	video card
11h	PC configuration
12h	memory configuration
16h	keyboard

C

Ajuntador: tipo de dados cujo valor é um endereço de memória

* obtém o endereço de uma variável

* desreferencia o apontador - lê/escreve nas posições da memória para as quais ele aponta (e declara o apontador)

arrays:

- Os elementos de um array são armazenados em posições contíguas de memória
- O nome de um array é o endereço do primeiro elemento desse array
- C suporta aritmética de apontadores

! O apontador é deve ser declarado para apontar para variável do tipo dos elementos do array a

Problema: Os compiladores de C organizam os tipos de dados em memória com o objetivo de tornar os acessos mais rápidos ... mas ... às vezes é necessário "optar" a informação ao processador

Solução: # pragma pack(1)

pragma options align=reset

OU __attribute__((packed))

```
static void *video_mem; /* Address to which VRAM is mapped */
static unsigned hres;   /* Frame horizontal resolution */
static unsigned vres;   /* Frame vertical resolution */

void vg_fill(uint32_t color) {
    int i;
    uint32_t *ptr; /* Assuming 4 byte per pixel */
    ptr = video_mem;

    for(i = 0; i < hres*vres; i++, ptr++) {
        *ptr = color; /* Handle a pixel at a time */
    }
}
```

```
typedef struct reg86 {
    union {
        struct { /* 32-bit (double word) access */
            [...]
        };
        struct { /* 16-bit (word) access */
            [...]
        };
        struct { /* 8-bit (byte) access */
            u8_t intno; /* Interrupt number (input only) */
            u8_t : 8; /* unused */
            u16_t : 16; /* unused */
            [...] /* unused */
            u8_t al, ah; /* 8-bit general registers */
            u16_t : 16; /* unused */
            u8_t bl, bh; /* 8-bit general registers */
            u16_t : 16; /* unused */
            u8_t cl, ch; /* 8-bit general registers */
            u16_t : 16; /* unused */
            u8_t dl, dh; /* 8-bit general registers */
            [...] /* unused */
        };
    };
} reg86_t;
```

UNIÓES
ESTRUTURAS
ANÔNIMAS

XPM

PIXMAP: matriz de pixels — representação de uma imagem digital como um array de valores de cores de pixels — matriz das coordenadas do ecrã para valores de cores

XPM: X PIXMAP — formato de imagem em que o valor de cada cor de um PIXMAP é representado por sequências de caracteres (strings)

Um XPM para um PIXMAP pode ser guardado num ficheiro de texto ou num array em C

`uint8_t *xpm_load (xpm_map_t xmap, enum xpm_image_type type, xpm_image_t *img)`
lê um PIXMAP XPM xmap e retorna o PIXMAP como um array bi-dimensional `uint8_t`, tendo a codificação da cor para o PIXMAP de acordo com a especificada em type

`map = xpm_load (xmap, XPM_INDEXED, &img)`

Sprites

Sprite: imagem bi-dimisional que é integrada numa cena maior
• Permite a integração de picmaps independentes na cena
• Permite animação de imagens sem alterar o fundo do ecrã
• Um sprite pode ser animado apresentando uma sequência de picmaps

Funções com Número Variável de Argumentos: devem ter pelo menos um argumento com nome e os argumentos nem nome não fazendo numa lista de tamanho conhecido e com tipos conhecidos
`<stdarg.h>`

LAB 5

1. Configura a placa de vídeo para o modo gráfico desejado
↳ O Minix 3 inicia em modo texto, não em modo gráfico
2. Escreve para a VRAM para mostrar no ecrã o que é pedido
↳ Mapas VRAM para o espaço de endereçamento do processo
3. Reaja a placa de vídeo para o modo texto usado pelo Minix
↳ É necessário chamar a função fornecida

1. video-test-init(mode, delay)

```
reg86_t r;  
r.ax = 0x4F02; // VBE call, function 02 -- set VBE mode  
r.bx = 1<<14|0x105; // set bit 14: linear framebuffer  
r.intno = 0x10;  
if( sys_int86(&r) != OK ) {  
    printf("set_vbe_mode: sys_int86() failed \n");  
    return 1;  
}
```

1/2/4 ↗ vg_fill()

2. video-test-rectangle(mode, x, y, width, height, color)

- a. vg-draw-line(x, y, len, color)
- b. vg-draw-rectangle(x, y, width, height, color)

Problema: modos que usam 3 bytes

Solução: void *memcopy(void *dest, void *src, size_t n);

Ou typedef struct { uint8_t red; uint8_t green; uint8_t blue } rgb888t;

0	1	2	3
color(0,0)	color(0,1)	color(0,2)	color(0,3)
4	5	6	7
color(1,0)	color(1,1)	color(1,2)	color(1,3)
8	9	10	11
color(2,0)	color(2,1)	color(2,2)	color(2,3)
12	13	14	15
color(3,0)	color(3,1)	color(3,2)	color(3,3)

3. video-test-pattern (modo, n° , first, step)

a. vg-draw-rectangle (x, y, width, height, color)

$$\text{index}(\text{row}, \text{col}) = (\text{first} + (\text{row} * \text{no_rectangles} + \text{col}) * \text{step}) \% (1 \ll \text{Bits Per Pixel})$$

OU

$$R(\text{row}, \text{col}) = (R(\text{first}) + \text{col} * \text{step}) \% (1 \ll \text{Red Ytree Mask})$$

$$G(\text{row}, \text{col}) = (G(\text{first}) + \text{row} * \text{step}) \% (1 \ll \text{Green Ytree Mask})$$

$$B(\text{row}, \text{col}) = (B(\text{first}) + (\text{col} + \text{row}) * \text{step}) \% (1 \ll \text{Blue Ytree Mask})$$

4. video-test-xpm (*Xpm, xi, yi): mostra, nas coordenadas (x_i, y_i) do ecrã, o mapa de pixels no formato XPM carregado em Xpm

Modo VBE: On 105

XPM: um carácter para cada valor de cor

Problema: Como converter XPM para PIXMAP?

Solução: Xpm-load() (map, tpm, *map)

5. video-test-move (*Xpm[], xi, yi, xf, yf, speed, frame-rate): move uma imagem no ecrã (não sobre os eixos x ou y) ↑ Timer 0

RTC

RTC: circuito integrado que mantém a data e a hora do dia, mesmo quando o PC está desligado

O espaço de endereçamento interno é um array de pelo menos 64 registos de um byte, cujo conteúdo é não-volátil e que podem ser endereçados individualmente para leitura e escrita

RTC-ADDR-REG: porto 0x70 — deve ser carregado com o endereço do registo RTC a ser acedido

RTC-DATA-REG: porto 0x71 — é usado para transferir dados de / para o registo do RTC acedido

Problema: E se a data/hora atualizar enquanto se lê?

VIP: "Update in Progress" — o RTC ativa o VIP do REGISTER-A 244 µs antes de começar uma atualização e desativa-o assim que a atualização termina

UEI: "Update Ended Interrupt" — se ativo, o RTC vai interromper no fim do ciclo de atualização, o próximo ciclo vai ocorrer pelo menos 999 µs depois — o REGISTER-C deve ser lido no H, para limpar IARFF

Interrupções Periódicas: interrupções periódicas são geradas de modo que uma atualização ocorre sempre dentro a meio do período (244 µs depois)

E se ...: o DD é "anticipado" ao ler o tempo?
 Desativar interrupções antes de começar a ler e reativá-las

Problema: Atualizações anúncias podem tomar atualizações de data/hora inconsistentes

Solução: Ativar o bit SET do REGISTER-B antes da atualização
 Previne o RTC de atualizar os registros com os valores dos contadores
 Depois de atualizar, resetar o bit SET para o RTC atualizar

O RTC reporta valores "don't care" (11XXXXXX) para registros de alarme

REGISTER_A

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
UIP	DV2	DV1	DVO	RS3	RS2	RS1	RS0

REGISTER_B

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
SET	PIE	AIE	UIE	SQWE	DM	24/12	DSE

REGISTER_C

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
IRQF	PF	AF	UF	0	0	0	0

REGISTER_D

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
VRT	0	0	0	0	0	0	0

0		00	0	SECONDS
13	14 BYTES	0D	1	SECONDS ALARM
14		0E	2	MINUTES
			3	MINUTES ALARM
			4	HOURS
			5	HOURS ALARM
			6	DAY OF THE WEEK
			7	DAY OF THE MONTH
			8	MONTH
			9	YEAR
127		7F	10	REGISTER A
			11	REGISTER B
			12	REGISTER C
			13	REGISTER D

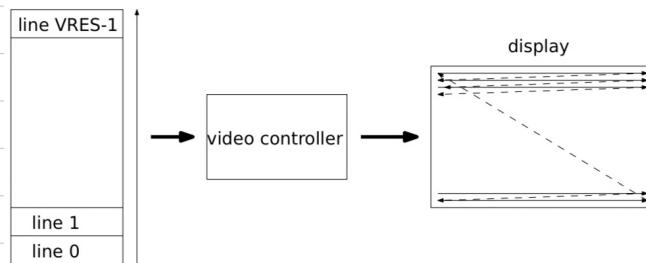
ADDRESS LOCATION	FUNCTION	DECIMAL RANGE	DATA MODE RANGE	
			BINARY	BCD
0	Seconds	0-59	00-3B	00-59
1	Seconds Alarm	0-59	00-3B	00-59
2	Minutes	0-59	00-3B	00-59
3	Minutes Alarm	0-59	00-3B	00-59
4	Hours, 12-hour Mode	1-12	01-0C AM, 81-8C PM	01-12AM, 81-92PM
	Hours, 24-hour Mode	0-23	00-17	00-23
5	Hours Alarm, 12-hour	1-12	01-0C AM, 81-8C PM	01-12AM, 81-92PM
	Hours Alarm, 24-hour	0-23	00-17	00-23
6	Day of the Week Sunday = 1	1-7	01-07	01-07
7	Date of the Month	1-31	01-1F	01-31
8	Month	1-12	01-0C	01-12
9	Year	0-99	00-63	00-99

Buffering

Graficos "Plástic": uma imagem é representada como uma matriz retangular ou grade de pixels quadrados

- Cada pixel tem um valor numérico, normalmente uma cor
- A posição do pixel na grade é determinada implicitamente
- O tamanho da imagem depende da resolução

Graficos Vectors: imagens não criadas a partir de formas geométricas como pontos, linhas e curvas num plano Cartesiano



- Para alterar o frame apresentado, é necessário alterar os conteúdos do frame buffer, na VRAM

Problema: se forem modificados os conteúdos do frame buffer de uma região que está a ser alcançada, a imagem no ecrã pode mostrar artefactos visuais

- Se não houver nenhuma alteração, o problema vai ser resolvido da forma que o ecrã é atualizado

Double Buffering: usar um segundo buffer, o "back buffer" na RAM

1. Criar o novo frame no "back buffer"
2. Copiar o novo frame do "back buffer" para o frame buffer na VRAM

Page Flipping: os dois buffers estão na VRAM

O controlador de vídeo tem um registo que aponta para o buffer a ser mostrado na tela, o "front buffer"

Em vez de copiar, só "toca" os dois buffers, i. e., troca os conteúdos do registo para apontar para o "back buffer" e trocar os papéis dos dois buffers (o "back buffer" torna-se o "front buffer" e v.v.)

VBE: função 0x07

Usar o intervalo de rastreamento vertical, i. e., o intervalo entre o rendering do pixel mais à direita da última linha de um frame e o rendering do pixel mais à esquerda da primeira linha do próximo frame

VGA: tem registo para configurar a geração de interrupções no rastreamento vertical

VBE: função 0x07 e sub-função 0x80 permitem tocar o frame buffer no próximo rastreamento vertical

Porta Série

Comunicação Paralela: dados são enviados simultaneamente em paralelo através de vários canais

Comunicação Síncrona: dados são enviados sequencialmente através de um canal

Problema: Como é que o receptor sincroniza?

Comunicação Síncrona: os relógios estão sincronizados

Comunicação Asíncrona: os relógios variam independentemente

- Os dados são agrupados em caracteres (normalmente 7/8 bits)
- Cada caractere é precedido por um bit de início (normalmente 0)
- Cada caractere é seguido por um bit de paragem (normalmente 1)

Bit de Paridade: usado para detecção de erros simples

Bit-Rate: máximo número de bits que é transmitido por unidade de tempo

Comunicação Simples: comunicação só em uma direção

Comunicação Full-Duplex: comunicação em ambas as direções, simultaneamente

Comunicação Half-Duplex: comunicação em ambas as direções, não simultaneamente

UART: controlador de comunicação assíncrona

8 números de portas consecutivas
1 linha IRQ

Port	Base Address	IRQ	Vector
COM1	0x3F8 (-0x3FF)	4	0x0C
COM2	0x2F8 (-0x2FF)	3	0x0B

Address	Read/Write	Mnem.	Description
0	R	RBR	Receiver Buffer Register
	W	THR	Transmitter Holding Register
1	R/W	IER	Interrupt Enable Register
	R	IIR	Interrupt Identification Reg.
2	W	FCR	FIFO Control Register
	R/W	LCR	Line Control Register
3	R/W	MCR	Modem Control Register
	R	LSR	Line Status Register
4	R	MSR	Modem Status Register
	R/W	SR	Scratchpad Register

IMPORTANT Addresses 0 and 1 are overloaded, accessing different registers if bit DLAB of the LCR register is set to 1:

Address	Read/Write	Mnem.	Description
0	R/W	DLL	Divisor Latch LSB
1	R/W	DLM	Divisor Latch MSB

Line Control Register (LCR) Allows the setting of the main asynchronous communication parameters: number of bits per character, number of stop bits and parity

DLL and DLM Allows the setting of the bit rate (by means of a frequency divider), via the Divisor Latches of the programmable bit-rate generator.

Line Status Register (LSR) Provides status information concerning the data transfer: whether a character was transmitted or received, and in the latter case whether an error was detected

Interrupt Enable Register (IER) Allows the selection of the events that may generate interrupts

Interrupt Identification Register (IIR) Provides information regarding the event that caused an interrupt

FIFO Control Register (FCR) Allows the control of FIFO buffering, both for reception and for transmission

IMPORTANT These registers may also include state/control bits that do not match exactly the purpose described above.

Bit	LCR			Meaning
1, 0				Number of bits per char
	0	0	5 bits per char	
	0	1	6 bits per char	
	1	0	7 bits per char	
	1	1	8 bits per char	
2		0	1 stop bit	
		1	2 stop bits (1 and 1/2 when 5 bits char)	
5, 4, 3				Parity control
	X	X	0	No parity
	0	0	1	Odd parity
	0	1	1	Even parity
	1	0	1	Parity bit is 1 (always)
	1	1	1	Parity bit is 0 (always)
6				Break control: sets serial output to 0 (low)
7 (DLAB)		1		Divisor Latch Access
		0		RBR (read) or THR (write)

LSR

Bit	Name	Meaning
0	Receiver Data	Set to 1 when there is data for receiving
1	Overrun Error	Set to 1 when a character received is overwritten by another one
2	Parity Error	Set to 1 when a character with a parity error is received
3	Framing Error	Set to 1 when a received character does not have a valid Stop bit
4	Break Interrupt	Set to 1 when the serial data input line is held in the low level for longer than a full "word" transmission
5	Transmitter Holding Register Empty	When set, means that the UART is ready to accept a new character for transmitting
6	Transmitter Empty Register	When set, means that both the THR and the Transmitter Shift Register are both empty
7	FIFO Error	Set to 1 when there is at least one parity error or framing error or break indication in the FIFO Reset to 0 when the LSR is read, if there are no subsequent errors in the FIFO