Arquitectura de Computadores
Teste nº 2 – versão A – 12 de Junho de 2015

Nome Nº

Teste sem consulta e sem esclarecimento de dúvidas.
Duração: 2h 15m

Questão 1 (1 valor)

O bit M do registo *status word* do processador (PSW) indica se o CPU pode executar instruções privilegiadas (se o bit M é 1) ou se não pode (se o bit M é 0). Indique o valor desse bit M em cada uma das seguintes situações:
a) o processo executa o código do programa do utilizador: M = 0

b) o processo efetuou uma chamada ao sistema e executa código do kernel. M = 1

c) o código que atende a interrupção do relógio (*timer*) é executado. M = 1

Questão 2 (2 valores) Para suportar o que um dos livros usados na cadeira "Operating Systems: Three Easy Pieces" (OSTEP) chama *limited direct execution*, a arquitectura hardware deve suportar os três seguintes mecanismos:

- instruções privilegiadas,
- proteção de memória, e
- interrupção de relógio (*timer*).

Explique porque é que basta que um deles não exista para que o sistema operativo não possa garantir que um dado processo efetua a computação que lhe está atribuída sem interferência dos outros processos.

Os três mecanismos têm de existir em simultâneo
- se existisse proteção de memória e interrupções de relógio e não existissem instruções privilegiadas, os programas utilizadores poderiam manipular o hardware, destruindo por exemplo o conteúdo de um sistema de ficheiros no disco.
- se existissem instruções privilegiadas e interrupções de relógio mas não existisse proteção de memória um processo poderia ler e escrever nas imagens de outros processos
- - se existissem instruções privilegiadas e proteção de memória e não existissem interrupções de relógio, um processo poderia ocupar indefinidamente o CPU, com isso impedindo uma distribuição justa do recurso CPU

Questão 3 (2.5 valores) Considere a forma como são suportadas as chamadas ao sistema no sistema operativo Linux sobre um CPU como a do Pentium.

a) Para realizar uma chamada ao sistema, os registos do CPU são carregados com valores que especificam qual é a chamada ao sistema e quais são os parâmetros. Por exemplo, a biblioteca do C implementa a chamada ao sistema *write(1, msg, 14)* através da execução das seguintes instruções máquina

```
movl     $14, %edx  # numero de bytes a escrever (14)
movl     $msg, %ecx # endereço onde se encontram os bytes (msg)
movl     $1, %ebx   # canal 1
movl     $4, %eax   # 4 é o código para indicar que se quer fazer um write
int      $0x80
…
.data
msg: .ascii "hello, world \n"
```

Explique em detalhe o que se passa após a execução da instrução ***int $0x80*** nomeadamente quanto ao modo em que o CPU fica, valor no PC e o estado da pilha.

A instrução int provoca a entrada no código do sistema operativo para que seja executada uma chamada ao sistema. Para esse efeito a execução da instrução int 0x80 faz com que a unidade de controlo do CPU

- empilhe o IP e o conteúdo das flags
- mude o CPU para modo supervisor
- o IP é carregado com o conteúdo da posição de memória vector_interrupcoes[0x80]
- o endereço carregado no IP é onde está o código que executa a chamada ao sistema especificada pelo conteúdo do registo eax

b) O Pentium tem uma instrução máquina chamada *return from interrupt*, cuja mnemónica é ***iret***. Esta instrução, entre outras coisas, muda o modo de operação do CPU de modo sistema para modo utilizador. Explique porque é que a instrução *iret* existe e em que circunstâncias é usada.

Esta instrução é usada para finalizar uma rotina de tratamento de interrupções sendo usada para colocar o CPU em modo utilizador e voltar à instrução que o CPU se preparava para executar quanto ocorreu a interrupção. Para restaurar o modo do CPU é feito pop das flags guardadas pela instrução INT. Para retomar a execução de instruções no ponto a seguir à instrução int, é feito pop com destino ao IP.

Nome Nº

Questão 4 (5.5 valores)

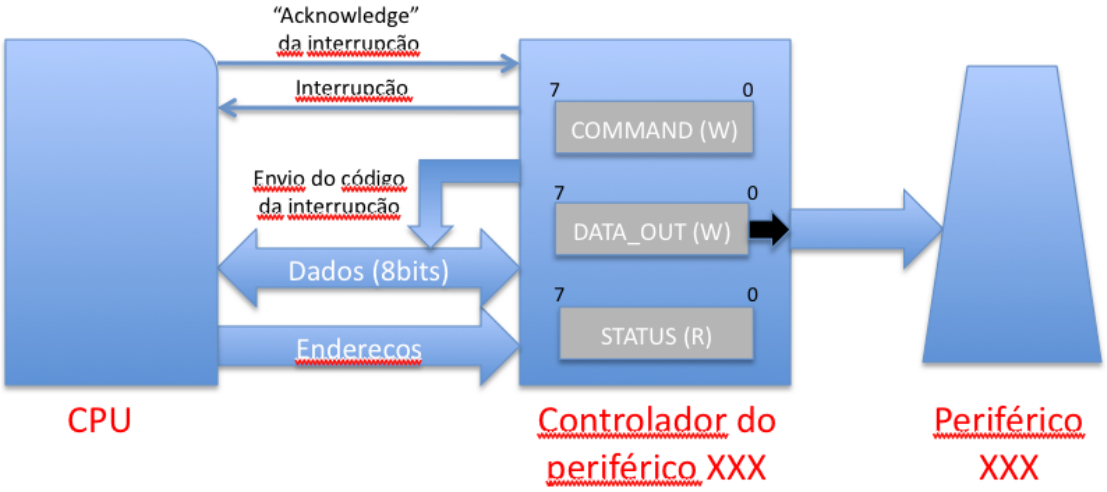
As várias alíneas desta pergunta supõem um ambiente, do ponto de vista do software, semelhante ao usado nas aulas práticas e no projecto:

- sistema operativo *Freedos*
- compilador *Turbo C* com as seguintes funções disponíveis
    - unsigned char inportb (unsigned int portid)
    - void outportb(unsigned int portid, unsigned char value)
    - void enable(void)
    - void disable(void)

Supondo a existência de uma rotina de tratamento de interrupções void my_isr() o endereço dessa subrotina é colocada na entrada 5 do vector de interrupções através da invocação da função setvect(5, my_isr).
Está ainda disponível um módulo de software que manipula um *buffer circular* e que disponibiliza as seguintes operações

- void buffer_init(): inicializa a estrutura de dados que descreve o *buffer*
- int buffer_full(): retorna 1 se não há espaço e 0 se há
- int buffer_empty(): retorna 1 se o *buffer* está vazio e 0 se não está
- void buffer_put(unsigned char c): coloca c na 1ª posição livre do *buffer*; assume que há espaço
- unsigned char buffer_get(): retorna o elemento que está há mais tempo no *buffer*; assume que há pelo menos um elemento no buffer

Do ponto de vista do hardware, o controlador do periférico XXX é o único controlador ligado à linha de interrupção do CPU. Depois do controlador accionar a linha de interrupção do CPU, aguarda que este active a linha de "Acknowledge" da interrupção; quando isto acontece o controlador coloca o código 0x11 no bus de dados e desactiva a linha de interrupção.



Todos os registos do controlador têm 8 bits e o bit 7 é o mais significativo. Os registos são os seguintes:

- **endereço 0x30 DATA_OUT**: registo que só pode ser escrito; o valor escrito neste registo é transferido para o periférico XXX
- **endereço 0x31 STATUS**: registo só de leitura; se o bit 1 está a 1 isso indica que o controlador está pronto a enviar mais um byte. Quando o CPU escreve no registo DATA_OUT este bit fica a 0.
- **endereço 0x32 COMMAND**: registo só de escrita; Quando o CPU escreve neste registo o valor 0x00 o controlador não envia interrupções para o CPU; quando o CPU escreve neste registo 0xFF o controlador acciona a linha de interrupção quando o bit 1 do registo STATUS está a 1

a) Escreva, usando o Turbo C, uma rotina com o protótipo
`void out_XXX( unsigned char c)`
que envia o byte c para o periférico XXX usando espera activa

```
void   out_XXX(unsigned char c){
       unsigned char s;
       do{
          s = inportb( STATUS)
       }while ( s & 0000 0010B )  // espera que seja possível enviar
       outportB( DATA_OUT, c );
}
```

b) Escreva, usando o Turbo C, uma rotina com o protótipo
void out_XXX(unsigned char c que envia o byte *c* para o periférico XXX usando interrupções. Escreva também a rotina de atendimento de interrupções out_XXX_int() cujo endereço está na entrada 0x11 do vector de interrupções.

```
void   out_XXX(unsigned char c){
       unsigned char s;
       while ( bufFull( )  ) ;  // espera que o buffer tenha espaço
       if ! bufEmpty()
           bufPut( c );
       else{
           bufPut( c );
           outportb( COMMAND, 0xFF);   // liga as interrupções
       }
}

interrupt myISRXX{
           unsigned char c;

           if !bufEmpty()
             outportb (DATA_OUT, bufGet( ));

           if bufEmpty()
             outportb( COMMAND, =0x00); // desliga as interrupções
}
```

c) Apresente também o código que deve ser executado antes da ocorrência da 1ªinterrupção proveniente do periférico XXX, relativamente à inicialização do periférico e do vector de interrupções.

```
bufferInit( );
setvect( 11, myISRXX);
outportb( COMMAND, 0xFF);   // liga as interrupções
```

Nome Nº

Questão 5 (1.5 valores) Considere a seguinte sequência de comandos no shell que se supõem serem executados sem erros. Os dois primeiros geram ficheiros objecto a partir de códigos fonte em C; o terceiro comando gera um ficheiro executável.

```
gcc –c –o p1.o  p1.c
gcc –c –o p2.o  p2.c
ld –o prog p1.o p2.o
```

Explique como é que o ligador *(ld)* gera o ficheiro executável.

O ligador usa a informação presente no cabeçalho dos dois ficheiros objectos para saber a dimensão do código e dados de cada módulo bem como a distância de cada símbolo ao início do módulo, De posse destes elementos. Com essa informação consegue preencher (editar) todos os endereços que eram desconhecidos na compilação.

Questão 6 (3 valores) Um dado CPU emite endereços com 24 bits. Entre o CPU e a memória central existe uma cache associativa por grupos com as seguintes características:

- uma linha tem 8 bytes
- cada conjunto ou grupo tem 4 linhas
- Há 1024 conjuntos

a) Qual é a capacidade da cache?

8 kBytes 4 * 1024 = 32 Mbytes*

b) O CPU emite o endereço E com 24 bits. Como é que o hardware verifica que o conteúdo de E está ou não na cache? Indique claramente como é que o endereço é dividido, indicando para que servem cada um dos 24 bits.

Os bits 0 a 2 contêm o deslocamento em bytes em relação ao início da linha

Os bits 3 a 12 selecionam o grupo. Como há 1024 entradas são precisos 10 buts

Os restantes 11 bits são a marca

Nome Nº

Questão 7 (3 valores) Um sistema em que os endereços virtuais têm 28 bits, usa uma MMU que suporta páginas com dimensão 64 KBytes (2^{16}).

a) Diga como é interpretado um endereço virtual. Justifique

Os bits 0 a 15 contêm o deslocamento 2^16 = 64K
Os bits 16 a 27 (12 bits) são o número da página

b) Quantas entradas tem a tabela de páginas de cada processo? Justifique.

Como há 12 bits para o nº da página, o nº de entradas é 2^12 isto é 4096

c) Para um dado processo em execução, as primeiras entradas da tabela de páginas são as seguintes:

Nº página virtual	Nº página física (em base 16)
0	Inválida
1	0xAFE
2	0xADA
3	0xEAD

As outras entradas são todas inválidas. Indique, justificando, os endereços físicos que correspondem aos seguintes endereços virtuais. Responda "Endereço Inválido" se o endereço virtual for inválido.

b1) 0x0022000 nº da página 0x002 → nº da página física 0xADA → end. Físico 0xADA2000

b2) 0x0110100 nº da página 0x011 → página inválida

b3) 0x0030100 nº da página 0x003 → nº da página física 0xEAD → end. Físico 0xEAD0100

b4) 0x0121100 nº da página 0x012 → página inválida

Questão 8 (1.5 valores) Explique porque é que um sistema de gestão de memória baseado em páginas necessita de um TLB (*Translation Lookaside Buffer*)

A tabela de páginas está em memória (RAM). Supondo esta tabela completamente preenchida pelo SO, existem acessos que têm associadas dois acessos à RAM

1) Para obter PF que corresponde a PV
2) Para fazer o acesso efectivo

Este procedimento duplica o tempo de acesso à memória e não pode ser tolerado. Para obviar a este inconveniente existe o TLB que é usado exclusivamente para armazenar correspondências PV, PF. Quando o

par PV, PF está no TLB já não é preciso fazer o acesso 1). Por outro lado o tempo de consulta do TLB é muito menor do que o tempo de acesso à RAM.