Teste 2 de Arquitetura de Computadores 17/06/2023
Duração 2h (1h30m + 30m)

- *Teste sem consulta e sem esclarecimento de dúvidas. Ver na última página informação sobre as instruções máquina do x86-64.*
- *A deteção de fraude conduz à reprovação de todos os envolvidos. Não é permitido o uso de qualquer dispositivo digital.*
- *A cotação de todas as perguntas é semelhante, entre 1,0 e 1,25 valores*

Nº: ***Nome:***

Q1 – Considere um CPU com a arquitetura *X86_64*. Diga como é habitualmente usado o registo *%rbp*.
O registo %rbp marca uma posição fixa no espaço de trabalho de uma função (frame em inglês) o que permite usar livremente o registo %rsp. Faz-se acesso às variáveis locais da função e aos parâmetros de entrada da função se existirem usando endereçamento baseado, sendo o registo %rbp usado como base.

Q2 – Considere o seguinte fragmento de código assembly x86_64. Diga qual o conteúdo dos registos %rax e %rbx e %rcx depois de executadas todas as instruções abaixo.

```
      mov  $5, %rax
      xor   %rcx, %rcx
      mov  $3, %rbx
l1:   cmp  $0,%rbx
      jz   end
      dec %rbx
      add %rax, %rcx
       jmp l1
end:
```

%rax 5 : %rbx 0 ; %rcx 15

Q3 – Considere o seguinte fragmento de código C

```
int x = 3;
…
while( x < 7)
      x++;
```

Complete o código assembly X86-64 apresentado a seguir que é uma possível tradução do código C. Após o *while*, a execução continua no código que está na etiqueta *cont*:

```
.section .data
X: .quad 3
.section . code
….
while:
       cmpq __$7__, X

       _jz__ cont    % também pode ser jge
       incq   X

       __jmp__ while
cont:
```

Q4 – Considere a seguinte declaração em C

```
unsigned int vec[32];
```

Suponha que o endereço inicial de *vec* está no registo %rsi e que a variável *i* está no registo %rcx. Como traduziria em assembly X86-64 a instrução C *vec[i] = 5; ?*

```
movl $5, (%rsi, %rcx, $4)
```

Q5 – Considere as seguintes declarações em C

```
typedef struct {
    unsigned short size;
    unsigned short free;
} estrut;
estrut es[32];
```

Suponha que o endereço inicial de *es* está no registo %rsi e que a variável *i* está no registo %rcx. A tradução da instrução *es[i].free = 1;* para o seguinte código assembly X86_64

```
      lea (%rsi,%rcx,4), %rbx
      movw $1, 4(%rbx)
```

está correta? Justifique a sua resposta.

Cada elemento da estrutura ocupa 4 bytes. A primeira instrução carrega em %rbx o endereço inicial de es[i]. O elemento *size* está à distância 0 do início da estrutura e o elemento *free* está à distância 2 do início da estrutura. Assim, a 2a instrução deveria ser movw $1, 2(%rbx)

Q6 – Considera a seguinte função C

```
void swap( unsigned char *a, unsigned char *b){
      unsigned char t = *a;
                *a = *b;
                *b = t;
}
```

Complete o código assembly X86_64 correspondente à função acima indicada

```
swap:   # rdi unsigned char *a,  rsi unsigned char *b
          push %rbp
          mov %rsp, %rbp
          movb (%rdi), %ral     #  char t1 = *a
          movb (%rsi), %rah     #  char t2 = *b
          movb %ral, (%rsi)     #  *b = t1
          movb %rah, (%rsi)     #  *a = t2
         mov %rbp, %rsp
         pop %rbp
          ret
```

Q7 – Considere dois níveis de memória:

- Um nível L1 caracterizado por uma capacidade *C1* e um tempo de acesso *Ta1*
- Um nível L2 com capacidade *C2* e tempo de acesso *Ta2*. *C2* é muito maior do que *C1* e *Ta1* é muito menor que *Ta2*

Sendo *h* o hit rate (nº de referências que encontram o dado pretendido no nível L1 / nº total de referências), qual deve ser o valor de *h* para que o tempo de acesso médio se aproxime de *Ta1*.

Tempo de acesso médio TAM = h*Ta1 + (1-h) *Ta2
Para TAM ser próximo de Ta1, h terá se ser muito próximo de 1

Q8 – Diga quais são os inconvenientes de uma cache de mapa direto.

Nas caches de mapa direto a possibilidade de "misses" por conflito é grande. Se existem dois endereços E1 na linha L1 da memória central e E2 na linha L2 em que os "bits do meio" sejam iguais a L e a parte da marca é diferente as linhas L1 e L2 da memória central têm de estar na linha L da cache, o que, se E1 e E2 forem referenciados num ciclo provoca repetidas substituiçoes da linha L da cache

Q9 – O CPU XPTO usa uma cache com um nível único em que cada linha tem 64 bytes. A cache usa uma organização associativa por grupos em que existem 128 grupos e cada grupo tem 8 linhas. A figura seguinte apresenta a forma como o hardware associado à cache divide o endereço enviado pelo CPU

Diga, justificando, quais os valores de B1 e B2. Indique também como é que o hardware da cache deteta se uma dada linha da memória central está na cache.

O deslocamento precisa de 6 bits (2^6=64). logo B2 = 6
A seleção da linha /grupo precisa de 7 bits (2^7=128). logo B1 = 13
Para verificar se a linha pretendida está na cache, os bits B1-1 a B2 selecionam o grupo G, sendo a marca M os bits 31 a B1. Usando uma memória associativa vê-se se alguma das 8 linhas que estão no grupo G tem a marca M. Se M está numa das linhas é um hit na cache, senão é um miss

---

Q10 Considere o seguinte pedaço de código em que é feita uma chamada ao sistema no Linux. Suponha que as posições de memória *buf* e *res* foram declaradas anteriormente.

```
L10: mov $1, %rax
L11: mov $0, %rdi
L12: leal buf, %rsi
L13: mov $1, %rdx
L14: syscall
L15: mov %rax, res
```

Entre as linhas L10 e L15 o CPU muda de modo utilizador para modo sistema? E de modo sistema para modo utilizador? Se sim, quando é que tal acontece?

Na linha L14 syscall provoca a entrada no código do sistema operativo, e portanto o CPU muda para modo sistema; quando se executa a linha 15 o CPU está novamente em modo utilizador

---

Q11 Diga quais são os componentes da máquina virtual que o sistema operativo cria para cada processo.

Memória virtual para carregar código, dados, pilha e heap
CPU virtual para executar as instruções
Tabela de canais abertos para interagir com periféricos e ficheiros

---

Q12 Um ficheiro executável contém instruções máquina de um dado CPU; algumas dessas instruções contêm endereços de memória. Explique porque é que esses endereços são endereços virtuais e não endereços físicos.

Quando os compiladores e os ligadores geram os ficheiros executáveis, é o desconhecido o endereço físico em que o programa vai ser carregado em memória. Assim sendo assumem que o programa gera endereços entre 0 e um valor máximo L. Esses endereços não podem ser endereços físicos e têm, em tempo de execução ser convertidos para endereços físicos. Para o mesmo endereço virtual V vai corresponder a um físico diferente dependendo da zona da RAM em que o SO decidiu carregar o programa

---

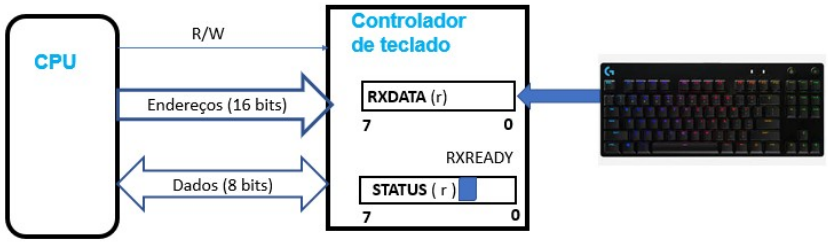
Q13– Explique como é que uma MMU (*memory management unit*) que contém um registo de recolocação (ou registo base) transforma endereços virtuais em endereços físicos.

Endereço físico = conteúdo do registo de recolocação + endereço virtual

Q14– Diga quais são as vantagens de uma MMU que faz a transformação de endereços virtuais em endereços físicos usando páginas, em relação a uma MMU que faz a mesma transformação usando um registo de recolocação.

Uma MMU com um registo de recolocação obriga ao carregamento da imagem contida no ficheiro executável contiguamente na RAM; isto pode provocar a fragmentação da RAM livre. Uma MMU baseada em páginas remove essa restrição permitindo usar todas as zonas de memória livres que existam num dado momento para carregar um programa; com esta tecnologia, páginas virtuais contíguas não precisam de corresponder a páginas físicas contíguas.

Q15– Considere a figura seguinte que mostra a interação entre um CPU, um controlador de entrada e saída que está ligada a um teclado que codifica as teclas premidas em 8 bits. O CPU pode ler e escrever em registos dos controladores de entrada / saída através de instruções de *in* e *out* que estão acessíveis num programa em C através das funções:
void outPort(unsigned short int port, unsigned char val);
unsigned char inPort(unsigned short int port);
O controlador tem os seguintes registos que estão acessíveis ao CPU:
●RXDATA (o CPU pode apenas ler): quando é lido um byte neste endereço, esses 8 bits são transferidos para um registo do CPU.
●STATUS (o CPU pode apenas ler): indica o estado em que se encontra a transferência entre o registo RXDATA e o teclado. O bit 3 deste registo (RXREADY) estará a 1 se já acabou a transferência do byte anterior e a 0 se a transferência ainda estiver a decorrer. Sempre que o CPU lê um byte no endereço RXDATA, este bit fica a 0.



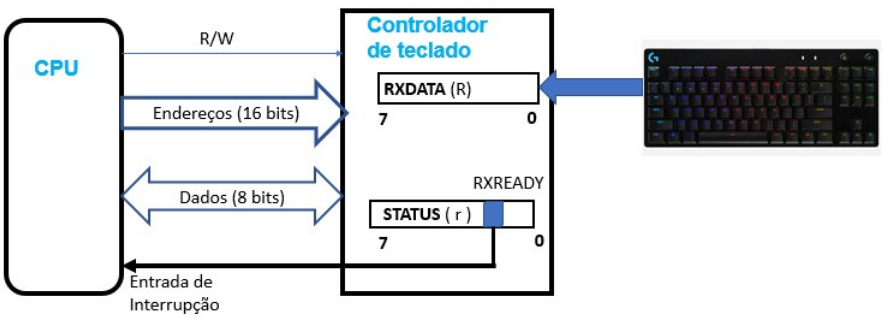
Pretende-se que escreva o código de uma função *unsigned char lerTeclado( )* que lê um byte do teclado, usando espera ativa.

unsigned char lerTeclado( ){

```
   while ((inport(STATUS) & 0x04)==0)
     ;
   return inport(RXDATA) ;
```

}

Q16 Considere a mesma situação descrita na pergunta anterior, mas em que ao controlador do teclado foi acrescentada a possibilidade de enviar uma interrupção ao CPU, tal como mostra a figura abaixo. Admita que a possibilidade de o controlador fazer interrupções está sempre ativada.



Suponha que está disponível uma estrutura de dados *buffer circular* de bytes que pode ser usada no seu programa através do tipo *bufcirc* com as seguintes operações definidas:

```
int bufFull( bufcirc *b);  // retorna 1 se o buffer estiver cheio; 0 se não estiver
int bufEmpty( bufcirc *b); // retorna 1 se o buffer estiver vazio; 0 se não estiver
unsigned char bufGet(bufcirc *b); // devolve o byte que está há mais tempo no buffer;
                                  // quando é chamada, *bc não pode estar vazio
void bufPut( bufcirc *bc, unsigned char b); // coloca o byte b na 1a posição
                                            // livre buffer; assume que *bc não está cheio
```

escreva o código de uma função *unsigned char lerTeclado(bufcirc *bc)* que, em vez de usar espera ativa, usa interrupções. Esta função consulta o buffer circular **bc* e retorna o valor que está no buffer há mais tempo, removendo esse valor do buffer. Se não há nada no buffer, retorna 0xFF.

```
unsigned char lerTeclado( bufcirc *bc ){
   if (bufEmpty(bc))
          return código indicando que não há dados disponíveis
   else
          return bufGet(bc):
}
```

Escreva também o código de uma rotina de tratamento de interrupções *interruptHandlerTeclado* que é invocada sempre que o controlador da porta do teclado gera uma interrupção (isto é, quando o bit RXREADY do registo de STATUS está a 1). Admita que esta rotina de tratamento de interrupções recebe como parâmetro de entrada o buffer circular que é partilhado com a função *lerTeclado( )*.

```
void  interruptHandlerTeclado( bufcirc *bc ){

  unsigned char t = inport(RXDATA);
  if (bufFull(bc))
          ;  // byte perde-se
   else
          bufPut(bc, t):

  interrupt_return; // executa a instrução máquina que termina
                    // uma rotina de tratamento de interrupções
}
```

Q17 Considere que um processo fez uma chamada ao sistema que implica a leitura de um bloco do disco. Explique as acções efetuadas pelo sistema operativo para executar a chamada ao sistema e também as que são efetuadas pelo controlador do disco.

1) Sistema operativo (SO) lança a operação de transferência indicando ao controlador do disco o no. do bloco a transferir e qual o endereçp em RAM onde estão os bytes a transferir. Bloqueia o processo que fez a chamada
2) o controlador inicia a transferência transferindo os bytes do bloco para a RAM através do uso de DMA (Direct Memory Access)
3) Quando acaba a transferência o controlador notifica o SO através de uma interrupção. O SO coloca o processo no estado READY (PRONTO)

Q18 Considere um disco de tecnologia magnética. Quais são os componentes do tempo de acesso a um setor do disco?

- tempo de espera na fila do controlador / Device Driver
- deslocamento da cabeça de R/W até à pista pretendida
- espera pela passagem do sector pretendido sob a cabeça
- transferência do setor enquamto ele passa debaixo da cabeça