# UNIVERSITY OF GLASGOW

**Degrees of MEng, BEng, MSc and BSc in Engineering**

# REAL TIME COMPUTER SYSTEMS 3 (ENG3043)

**Friday 20th December 2019**
**13:00-15:00**

**Answer ALL questions in section A and TWO questions from Section B.**

**Section A = 40 Marks**
**Section B = 60 Marks**

*The numbers in square brackets in the right-hand margin indicate the marks allotted to the part of the question against which the mark is shown. These marks are for guidance only.*

**SOME USEFUL INSTRUCTIONS ARE PROVIDED AT THE END OF PAPER**

**An electronic calculator may be used provided that it does not have a facility for either textual storage or display, or for graphical display.**

Continued overleaf

**SECTION A [40 marks]**

**Attempt all questions**

**Q1** (a) Briefly explain what a real-time computer system is. [3]

(b) Describe the main difference between soft and hard deadline systems and use diagrams to illustrate the consequence if missing the deadlines for the two systems. [4]

(c) Anti-lock braking system (ABS) is one of the most significant safety systems in automobiles. If you are asked to design an Electronic Control Unit (ECU) what factors to be considered to meet the requirements of ABS ? [3]

**Q2** (a) Both low-level languages (LLL) such as assembly languages and high-level languages (HLL) such as C language could be found in a programme. Explain the pros and cons of both types of languages. [4]

(b) You are asked to debug someone's code written in 8086 assembly language, identify errors in the following instructions

(i) MOV CH, BX;

(ii) MOV ES, DS;

(iii) MOV CS, AX;

(iv) MOV [DI], [BX]. [4]

(c) If d8 is 04H, BX contains 1234H, and DS contains 2000H, what does the instruction of MOV AX, d8[BX] do? [2]

Continued overleaf

**Q3** (a) Processors interact with peripherals using either “standard I/O” or “memory-mapped I/O”. Explain the difference between them. [4]

(b) Last May (2018) Qualcomm released the world’s first dedicated processor Snapdragon XR1 for augmented reality (AR) devices, comment on what main features the processor would possess. [3]

(c) Comment on which processor hardware may influence your design for a timing-critical application e.g. real-time system. [3]

**Q4** In most high-level languages parameters are passed between procedures or functions using the stack. ***Figure Q4*** shows the assembly language equivalent of the machine code for the 8086 processor produced by a compiler in order to implement the function inportb.

(a) Draw a diagram showing the contents of the stack right before the `in` instruction is executed [5]

(b) Explain why the logical address for IP is changed from `cs:029B` to `cs:029E` ? [3]

(c) Which register is used to return the result from `_inportb` ? [2]

```
#CINOUT#6: flag
  cs:0297 C646FD00        mov byte ptr [bp-03],00
#CINOUT#9:invalue = inportb(0x226);
  cs:029B B82602           mov      ax, 0226
  cs:029E 50               push     ax
  cs:029F E80301           call     _inportb
  cs:02A2 59               pop      cx
  cs:02A3 8846FF           mov      [bp-01], al
                    :
_inportb
  cs:03A5 55               push     bp
  cs:03A6 8BEC             mov      bp, sp
  cs:03A8 8B5604           mov      dx, [bp+04]
  cs:03AB EC               in       al, dx
  cs:03AC 32E4             xor      ah, ah
  cs:03AE 5D               pop      bp
  cs:03AF C3               ret
```

***Figure Q4.***

Continued overleaf

## SECTION B [60 marks]

**Attempt ANY 2 questions**

Q5 (a) A UART (Universal Asynchronous Receiver Transmitter) allows data to be transferred between computers using a single wire in each direction plus a common ground. The speed of data transmission is measured using baud rate. Explain baud rate and illustrate the function of a UART in sending and receiving data over a serial (RS-232) link between two computers with aid of a graph. [6]

(b) The arrival of characters on a serial connection from another computer is being detected by polling. The characters are then displayed on the screen of the receiving computer. ***Table 5b*** below shows the 16-bit values (in decimal) read from the 8254 timer immediately after successive characters are written to the computer screen.

| 8254 Timer Count |
|---|
| 2102 |
| 1278 |
| 454 |
| 65160 |
| 64338 |
| 60630 |
| 60586 |

***Table 5b***

From these measurements determine

(i) the baud rate being used in the transmission; [12]

(ii) the time taken to scroll the screen by one line; [3]

(iii) the time taken to print one value on the screen without scrolling [3]

(c) If the baud rate is continuously increased, at some point we will see missing characters on the screen. One of the solutions is to implement interrupts. If the following code (***Figure 5c***) is implemented, explain how this can solve the missing characters problem. [6]

```
void interrupt handler(__CPPARGS)
{
   buffer[storeindex]=inportb(dataport);
   count=count+1;
   outport(0xdcdC,count);
   storeindex=storeindex+1;
   if (storeindex>2047) storeindex=0;
   outportb(0x20,0x20);
}
```

***Figure 5c***

Continued overleaf

Q6 (a) The time stamp counter register is available in Pentium-type processors for counting and timing purpose. An 80486 PC has to be adapted to fulfil this function. The following code (**Figure. Q6A**) is written in assembly language for this purpose. Explain how this register can be read using the assembly language instruction `ReadTSC`.

[4]

```
int ReadTSC()
{
   asm {
       push ax;
       push cx;
       push dx;
       db 0x0f,0x31;
       mov [tsclo],ax;
       mov cx,16;
       db 0x66;
       ror ax,cl;
       mov [tschi],ax;
       pop dx;
       pop cx;
       pop ax;
        }
return 0;
}
```

***Figure. Q6A***

(b) In practice, why is it often sufficient to work with only part of the complete time stamp counter register? Comparing with a Programmable Interval Timer 8254 with a clock frequency of 1.1934 MHz, what is the main advantage of the `rdtsc` over 8254? [6]

Throughout the remainder of this question you should assume that 3.2GHz is the clock frequency of the Pentium processor on which the code is running.

***Question 6 continued overleaf…***

Continued overleaf

(c) A square wave is input to the bottom bit of an input port at address `0xdcdc` and the following code (**Figure Q6B**) is executed repetitively.

```
signal=inport(0xdcdc);
signal=signal & 1;
if(signal!=oldsignal)  {
        ReadTSC(tsclo,tschi);
        Time[sample]=tschi;
        delta=Time[sample]-Time[sample-1];
        freqHz=factor/delta;
        printf("Frequency = %f\r",freqHz);
        sample++;
      }
oldsignal=signal;
```

***Figure Q6B.***

(i) What time interval would lead to an increase of 1 in the value of `tsclo`? Similarly, what time interval corresponds to `tschi` increasing by 1? [4]

(ii) In this code what time in the input signal is detected by the statement `if(signal!=oldsignal)`? Therefore what times associated with the signal are stored in the array `Time`? What characteristic of the signal is represented by `delta`? [4]

(iii) Several successive entries from the array `Time`, which contains a series of `tschi` values (in decimal), are shown below. From these values calculate the frequency of the input square wave. [6]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16578 | 16626 | 16674 | 16723 | 16771 | 16820 | 16868 | 16916 |

(iv) At both high frequencies e.g. 25.0 kHz and low frequency e.g. 0.25 Hz, this program will encounter problems. Explain the nature of this problem and suggest a possible solution. [6]

Continued overleaf

Q7 (a) In a real-time system the computer may have to respond to multiple external events which happen at approximately the same time, and it must appear to handle them simultaneously. Explain the role of interrupts and software buffers in achieving rapid response to external events. [6]

(b) Rate monotonic scheduling (RMS) is one of scheduling algorithms for non-interacting periodic tasks in a single core microprocessor. Explain how the total task utilisation $U_T$ and utilisation upper bound $U_b$ can be used to assess if a task set of $n$ tasks are schedulable. [6]

(c) Assuming there are four periodic tasks in a super train control system with the following features as shown in ***Table Q7*** C=execution time/task duration, D=Deadline and T=task period.

| Task | C | T | D |
|---|---|---|---|
| T1 | 1 | 8 | 8 |
| T2 | 2 | 5 | 3 |
| T3 | 2 | 10 | 10 |
| T4 | 3 | 12 | 12 |

***Table Q7***

(i) Determine if the RMS algorithm can handle these tasks without causing any catastrophic consequences. [10]

(ii) To meet another requirement, the period of T2 must be changed to 20, can the task set still be scheduled? [5]

(iii) Comment on the trend of the upper bound as the number of tasks increases and how this influences your system design. [3]

Continued overleaf

## Appendix I

## Commonly used 8086 instructions

| | **Data operations** | |
|---|---|---|
| Mov | Copy byte or word.<br>The operands may be any sensible combination of a numerical value, a register or a memory location (denoted by square brackets [ ] ).<br>As with all operations it reads right to left. | Mov ax,bx ; register<br>Mov al,0ffh ; immediate<br>Mov ax,[bx or bp + si or di + constant]<br>; indirect |
| In,out | I/O port | Out 40h,al or in ax,dx |
| Push,pop | Store onto or retrieve from stack | Push ax or Pop cx |
| Xchg | Exchange | Xchg al,bh |
| Xlat | Translate | Xlat = mov al,[bx+al] |

| | **Arithmetic / logic operations** | |
|---|---|---|
| Add,sub | Add or subtract | Add bx,ax ; answer in bx |
| Adc,sbb | Add or subtract with carry/borrow | Adc al,bl ; carry set by previous operation |
| Inc,dec | Add or subtract 1 | Inc bx |
| Neg | Take 2s complement | Neg ax |
| Cmp | Compare – sets flags as if a subtraction had been done | Cmp ax,1200 |
| Rol,ror | Rotate left or right 8 or 16 bit registers excluding carry bit. | |
| Rcl,rcr | Rotate left or right 8 or 16 bit registers including carry bit. | |
| Shl,shr | Logical shift left or right. | |
| Sal,sar | Arithmetic shift left or right. | |
| Not | Invert all the bits | |
| And, or, xor | And, or, xor | |

Continued overleaf

|  | **String Instructions** |  |
|---|---|---|
| Movsb,movsw | String moves, byte or word oriented | From ds:si to es:di |
| Rep | Repeat operation with implicit decrement of cx until 0. Also alters si and di. |  |
|  | **Processor Control** |  |
| Sti,cli | Set and clear interrupt enable flag |  |
| Rdtsc | Read time stamp counter | 64 bit counter transferred to edx (upper 32 bits) and eax (lower 32 bits). |
| Nop | Do nothing |  |
| hlt | Halt the processor – sometimes used when waiting for interrupts to ensure consistent response time. |  |

|  | **Branch – jump, call or interrupt** |  |
|---|---|---|
| Jmp | Unconditional jump – short (+127 / -128 bytes), near (within segment) or far (anywhere). |  |
| Jz,jnz,jne etc. | Conditional jump – if zero flag, if zero flag not set or if not equal. | There are many of these and many have alternative mnemonics (je =jz). All are short jumps i.e. restricted to +127 / -128 bytes. |
| Loop | Decrement cx and jump back to this label, repeating the process until cx=0. |  |
| Call,ret | Push return address and branch to code. Pop return address and back to the instruction after the call. |  |
| Int,iret | Software interrupt, return from interrupt with flags restored |  |

Continued overleaf

## Appendix II

### UART Registers

| Base Address | DLAB | I/O Access | Abbrv | Register Name |
|---|---|---|---|---|
| +0 | 0 | Write | THR | Transmitter Holding Buffer |
| +0 | 0 | Read | RBR | Receiver Buffer |
| +0 | 1 | Read/Write | DLL | Divisor Latch Low Byte |
| +1 | 0 | Read/Write | IER | Interrupt Enable Register |
| +1 | 1 | Read/Write | DLM | Divisor Latch High Byte |
| +2 | x | Read | IIR | Interrupt Identification Register |
| +2 | x | Write | FCR | FIFO Control Register |
| +3 | x | Read/Write | LCR | Line Control Register |
| +4 | x | Read/Write | MCR | Modem Control Register |
| +5 | x | Read | LSR | Line Status Register |
| +6 | x | Read | MSR | Modem Status Register |
| +7 | x | Read/Write | SR | Scratch Register |

### FIFO Control Register (FCR)

This is a relatively new register that was not a part of the original 8250 UART implementation. The purpose of this register is to control how the First In/First Out (FIFO) buffers will behave on the chip and to help you fine-tune their performance in your application. This even gives you the ability to "turn on" or "turn off" the FIFO. Keep in mind that this is a "write only" register. Attempting to read in the contents will only give you the Interrupt Identification Register (IIR), which has a totally different context.

| Bit | Notes | | | |
|---|---|---|---|---|
| 7&6 | Bit 7 | Bit 6 | Interrupt Trigger Level (16 bytes) | Interrupt Trigger Level (64 bytes) |
| | 0 | 0 | 1 byte | 1 byte |
| | 0 | 1 | 4 bytes | 16 bytes |
| | 1 | 0 | 8 bytes | 32 bytes |
| | 1 | 1 | 14 bytes | 56 bytes |
| 5 | Enable 64 Byte FIFO (16750) | | | |
| 4 | Reserved | | | |
| 3 | DMA Mode Select | | | |
| 2 | Clear Transmit FIFO | | | |
| 1 | Clear Receive FIFO | | | |
| 0 | Enable FIFOs | | | |

**Action required :**

The FIFO control register is at the base address of the UART +2. Find in the code where data is output to base address + 2. Interpret the value written to this address and verify that the interrupt is triggered when it is 14/16 full. Modify this value so that the trigger point is 8/16.

End of question paper