ÇÖZÜMLER

Name, Lastname, Stu # :

COM/BİL 204 Final Exam/Dönem Sonu Sınavı 31.05.2015

1. How many times does the control unit refer to memory when it fetches and executes a two-word indirect addressing mode if the instruction is a computational type requiring an operand from memory? / İki-kelime uzunluğunda dolaylı adres modunda ve bellekten bir operand'a gerek duyan hesapsal bir emrin bellekten okunması ve icrası sırasında denetim biriminin belleğe kaç kez erişmesi gerekir

a. 2	b. 3	c. 4	d. 5	e. 6

2. Which of the following is true about independent and memory mapped I/O? / Bağımsız ve belleğe haritalanmış G/Ç hakkında aşağıda söylenenlerden hangisi doğrudur?

a. All processors can use memory mapped I/O but only processors with independent I/O can use either / Tüm işlemciler belleğe haritalanmış G/Ç kullanabilir fakat yalnız bağımsız G/Ç destekli işlemciler herhangi birini kullanabilir	b. There is no performance difference between independent and memory mapped I/O / Bağımsız ve belleğe haritalanmış G/Ç başarımı arasında fark yoktur	c. Both I/O methods are easy to use if symbolic addresses and two pass assembler are used / Sembolik adresler ve iki geçişli assembler kullanılırsa her iki yöntemi kullanmak kolaydır	d. None of them	e. All of them

3. Which of the following is true about a branch, a subroutine call and program interrupt? / Dallanma, altprogram çağırma ve program kesmeleri hakkında aşağıdakilerden hangisi doğrudur?

a. It is possible to use branch instead of subroutine call to speed up the execution / Çalışmayı hızlandırmak için altprogram çağrısı yerine dallanma kullanmak mümkündür	b. Interrupt is the slowest of all because it needs to store the state of the processor / İşlemcinin durumunu saklaması gerektiği için hepsi içinde en yavaş olanı kesmedir	c. Subroutine call can not be nested unless a hardware stack is used to store return addresses / Bir donanım yığın kullanılmadıkça altprogram çağrıları iç içe (nested) kullanılamaz	d. A new interrupt request to the processor causes problems while servicing an interrupt / Bir kesmeye servis verilirken işlemciye yeni bir kesme isteği gelmesi probleme neden olur	e. Branch instructions can not be used depending on the status bits (flags) of the processor / Dallanma emirleri işlemcinin durum bitlerine (bayraklarına) bağlı olarak kullanılamaz

4. Which of the following is likely to happen if handshaking signal is not received in time? / Elsıkışma işaretinin zamanında alınamaması durumunda aşağıdakilerden hangisinin olması muhtemeldir?

a. Handshaking signal is ignored and next data item is sent to the receiver / El sıkışma işareti gözardı edilir ve bir sonraki veri alıcıya yollanır	b. Handshaking signals are ignored just once and the next data item is sent to the receiver / El sıkışma işareti yalnız bir kereliğine gözardı edilir ve bir sonraki veri alıcıya yollanır	c. Sender should wait for the handshaking signal in an endless loop / Gönderici sonsuz döngü içinde el sıkışma işaretini beklemelidir	d. Assuming that the first attempt to transfer is failed, the sender sends the same data to the receiver immediately / İlk aktarım denemesinin başarısız olduğu varsayımı ile aynı veri anında alıcıya gönderilir	e. Communication breaks down unless a timeout mechanism is used / Bir zaman aşımı mekanizması kullanılmamış ise aktarım kesilir

5. Which of the following is true about synchronous serial data transfer and asynchronous serial data transfer / senkron seri veri aktarım ve asenkron seri veri aktarım hakkında aşağıdakilerden hangisi doğrudur?

a. Synchronous serial data transfer is less efficient because it needs to send synchronization bits in order not to lose synchronization / Senkronizasyonu kaybetmemek için senkronizasyon bitleri göndermesi gerektiğinden senkron seri veri aktarımı daha az verimlidir	b. Synchronous serial data transfer is more efficient because it does not need start and stop bits for each character / Her karakter için başla ve bitiş bitlerine gerek duymadığı için senkron seri veri aktarımı daha verimlidir	c. Synchronous serial data transfer can achieve higher bit rates because synchronization is handled by an electronic circuit / Senkronizasyon elektronik bir devre tarafından sağlandığı için senkron seri veri aktarımı daha yüksek bit hızlarına ulaşabilir	d. Synchronous serial data transfer is slower because it needs zero insertion to prevent a flag occuring in the frame / Çerçeve içinde bayrak oluşmaması için sıfır sokmaya gerek duyduğundan senkron seri aktarım daha yavaştır	e. Asynchronous serial data transfer is faster because synchronization needed by synchronous transfer needs extra time / Senkron aktarımın gerek duyduğu senkronizasyon fazladan zaman aldığı için asenkron seri veri aktarımı daha hızlıdır

6. Which of the following is the primary advantage of using interrupt initiated data transfer over transfer under program control without interrupts? / Aşağıdakilerden hangisi program denetimi altında veri aktarımı yapmaya göre kesme ile başlatılan veri aktarımının asıl avantajıdır?

a. Interrupt initiated data transfers are faster and moves more data in the same amount of time / Kesme ile başlatılan veri aktarımı daha hızlıdır ve aynı süre içinde daha fazla veri aktarır	b. Response time to interrupt is shorter since it does not require to save the state of the processor / İşlemcinin durumunu saklamak gerekmediği için kesmeye tepki süresi daha kısadır	c. The utilization of the processor is better because it is free to perform other tasks / İşlemci diğer işleri gerçekleştirebilecek şekilde serbest olduğu için kullanımı daha iyidir	d. Interrupt initiated data transfer are faster and moves more data in the same amount of time because it uses stack as well / Kesme ile başlatılan veri aktarımı daha hızlıdır ve aynı süre içinde daha fazla veri aktarır çünkü yığını da kullanır	e. Interrupt initiated data transfers consumes more energy because it checks the interrupt flag after each instruction / Her emir sonunda kesme bayrağını kontrol ettiği için kesme ile başlatılan veri aktarımı daha fazla enerji harcar

7. The only way to prioritize interrupts by a computer without priority hardware that branches to a common interrupt service routine upon an interrupt initiated by many sources is / Birçok kaynak tarafından başlatılan bir kesme üzerine ortak bir servis programına dallanan kesme öncelik donanımı olmayan bir bilgisayarın, kesmeleri önceliklendirebilmesi için tek yol

a. Interrupt service routine should poll all interrupt generating sources in descending priority order to find the highest priority source / En yüksek öncelikli kaynağı bulmak için kesme servis programı tüm kesme üreticilerini azalan öncelik sırasında yoklamalıdır	b. Connecting all sources of interrupt generators together in a way that they can resolve priority / Tüm kesme üreten kaynakları kendi aralarında önceliği çözümleyebilecek şekilde birbirine bağlayarak	c. Interrupt service routine should check the value of a specific register to determine the highest priority source / Kesme servis programı belirli bir kaydedici içeriğini kontrol ederek en yüksek öncelikli kesme kaynağını belirlemelidir	d. Interrupt service routine should poll all interrupt generating sources in ascending priority order to find the highest priority source / En yüksek öncelikli kaynağı bulmak için kesme servis programı tüm kesme üreticilerini artan öncelik sırasında yoklamalıdır	e. There is no way to prioritize interrupts without interrupt priority hardware / Kesme öncelik donanımı olmadan kesmeleri önceliklendirmek için bir yol yoktur

8. The read and write control lines in a DMA controller is bidirectional because / DMA denetleyicinin okuma/yazma denetim hatları çift yönlüdür çünkü

a. Programmer can read and write to DMA controller registers / Programcı DMA denetleyici denetim kaydicilerini okuma ve yazabilmelidir	b. Bidirectional bus buffers are cheaper to manufacture than unidirectional ones / Çift yönlü yol tamponları tek yönlü olanlardan daha ucuz üretilebilmektedir	c. Control lines of all devices connected to control bus needs to be bidirectional / Denetim yoluna bağlı olan tüm cihazların denetim hatları çift yönlü olmalıdır	d. They are used as output lines while DMA controller is running and input lines otherwise / DMA denetleyici çalışırken çıkış ve diğer zamanlarda giriş olarak kullanılır	e. There is no reason to be bidirectional, they could be unidirecional / Çift yönlü olmaları için gerekçe yoktur, tek yönlü de olabilirler

9. One of the useful functions of many high level languages is **int ToUpper(int c)** which converts the character **c** to upper case equivalent and returns it. An external command which converts all characters to upper case in an input file and stores it in an output file is needed. Design an assembly language program to process all the characters in an input file to create upper case equivalent in an output file given as command line arguments to ToUpper external command as **ToUpper *infile outfile***. You may assume that command line arguments are parsed (as given in arg_masm.asm file for the second homework) and both input and output files are open. Characters in ASCII table are located in such a way that, the hexadecimal code of "a" is 61h (97) and "A" is 41h (65), "b" is 62h and "B" 42h etc. All you have to do is conversion from lower case to upper case by masking off the 6th bit of all characters in the buffer read from the input file (by ReadFile) before writing the buffer to output file (by WriteFile). Also note that masking off the 6th position is corresponds to subtracting 32 (2^5) from the integer character code. Required code snippet from arg_masm.asm is given below.

Birçok yüksek seviyeli dilde bulunan yararlı bir fonksiyon verilen karakteri büyük harf eşdeğerine çevirerek geri döndüren **int ToUpper(int c)**'dir. Bir giriş dosyasındaki tüm karakterleri büyük harfe dönüştürdükten sonra bir çıkış dosyasına saklayan dış komut gerekmektedir. **ToUpper *giriş çıkış*** şeklinde dış komut dosyasına komut satırından arguman olarak verilen ***giriş*** dosyasındaki tüm karakterleri işleyerek eşdeğer büyük harflerden oluşan ***çıkış*** dosyasına yazan bir asssembly dili programı tasarlayınız. Komut satırı argumanlarının (ikinci ödev için verilen arg_masm.asm dosyasındaki gibi) ayrıştırıldığını ve giriş ve çıkış dosyalarının açıldığını varsayabilirsiniz. ASCII tablodaki karakterler öyle yerleştirilmişlerdir ki, "a" harfi onaltılı karakter kodu 61h (97) iken "A" harfi onaltılı karakter kodu 41h (65) ve "b" 62h ve "B" 42h vb. Bütün yapmanız gereken giriş dosyasından (ReadFile) ile okunan tampon içeriğindeki tüm karakterlerin 6. bit'ini (WriteFile) ile çıkış dosyasına yazmadan önce sıfırlamaktır. Karakterlerin 6. bitini sıfırlamanın karakter kodundan 32 (2^5) çıkarmaya karşı düştüğünü de fark edebilirsiniz. Aşağıda arg_masm.asm dosyasından gerekli kod parçası verilmiştir.

```
; Tampon tanımlamaları

BUFFER_SIZE   equ   32768              ; Dosya giris/cikis icin tampon boyu

TempBuffer  db BUFFER_SIZE dup(?)      ; Okuma (ve yazma için tampon)

; Komut satırını işleyen ve dosyaları açan kısım (kısa gösterim için verilmemiştir)

CopyLoop:

; giris dosyasindan oku

INVOKE ReadFile, [SrcFileHandle], offset TempBuffer, BUFFER_SIZE, offset BytesRead, 0

cmp [BytesRead], 0        ; dosya sonu ise

je EndCopy                ; kopyalamayi bitir

;

;  Your Assembly code should be placed here / Yazacağınız Assembly kod buraya yerleştirilmelidir

        mov  ecx, offset BytesRead
        mov  edx, offset TempBuffer
        mov  ebx, mask
dön:    and  [edx], ebx
        inc  edx
        dec  ecx
        jnz  dön

; cikis dosyasina yaz

INVOKE WriteFile, [DstFileHandle], offset TempBuffer, [BytesRead], offset BytesWritten, 0

jmp CopyLoop              ; dosyanin tumu islenene kadar tekrarla

EndCopy:
```