Name, Lastname, Stu # :

1	2	3	Toplam
1	1	3, 4	PÖÇ

KTU Faculty of Engineering, Dept. of Computer Engineering
COM/BİL 2000 Final Exam

24.05.2019

---

1. a. Which of the serial communication protocols has higher efficiency for transferring large amounts of data from a hard disk drive to buffers in memory? Explain the reason in **one sentence**. <=> Bir sabit diskten bellekteki tamponlara büyük miktarda veri aktarımı için hangi seri iletişim protokolü daha yüksek verimlidir? Nedenini **bir cümle** ile açıklayınız.

Sabit diskten okunabilecek veri içinde yalnız metin karakteri bulunması garanti edilemeyeceği için (çalıştırılabilir dosyalar, ses ve görüntü dosyaları, sıkıştırılmış dosyalar vb) iletim sırasında her tür veriyi iletebilen, bit yönelimli senkron seri veri haberleşme protokolu daha yüksek verimlidir.

b. What is the data throughput (in Mb/s or Gb/s) of the SATA III, USB 3.0, pci-e v3.0 serial interfaces? <=> SATA III, USB 3.0 ve pci-e v3.0 seri arayüzlerinin veri oranı (Mb/s veya Gb/s) nedir?

SATA III 6 Gb/s   USB 3.0 4.8 Gb/s   pci-e V3.0 8 Gb/s

2. A Motorola 680x microprocessor based digital controller runs at 1 MHz clock frequency and has a static RAM with 150 ns access time. The contents of 256 byte RAM at address **BLOK** needs to be output to the Asynchronous Communication Interface Adapter (ACIA) buffer at address **KUYRUK** by the assembly language program given below. Instructions timings in clock cycles is indicated at the **Clock cycles** column. <=> Motorola 680x işlemcili bir sayısal denetim sistemi 1 MHz saat frekansı ile çalışmakta ve 150 ns erişim süresi olan statik RAM'i bulunmaktadır. Aşağıda verilen assembly dili program ile **BLOK** adresindeki 256 byte RAM içeriği **KUYRUK** adresindeki Asenkron Haberleşme Birimi (ACIA) tamponuna gönderilmelidir. Emir zamanlamaları **Saat periyodu** kolonunda belirtilmiştir.

a. Compute total time (ie. total clock cycles x actual clock period) to transfer the 256 byte block using Programmed Input/Output given below. <=> Aşağıda verilen Programlı G/Ç ile bloğun toplam aktarım süresini (yani toplam saat sayısı x gerçek saat periyodu) hesaplayınız.

Döngü başlamadan 1 kez çalışan ve başlangıç değerlerini kaydedicilere yükleyen LDX BLOK ve LDA B,#$FF emirleri 5+2 = 7 saat ve döngü içinde 256 kez çalışan diğer emirler 5+3+4+2+4 = 18 saat periyodu olduğu için $1\cdot(5+2)+256(5+3+4+2+4) = 7+256\cdot 18 = 4615 \Rightarrow @ 1\,MHz = 4615\,\mu$

b. Compute total time if the same block is transferred by the Direct Memory Access technique. <=> Aynı bloğun Doğrudan Bellek Erişimi tekniği ile aktarılma toplam süresini hesaplayınız.

Bellek erişim süresi hızında (150 ns) 256 byte doğrudan belleğe erişim (DMA) aktarımı $256\cdot 150\,ns = 256\cdot 150\cdot 10^{-9} = 38{,}4\,\mu s$ sürede tamamlanır.

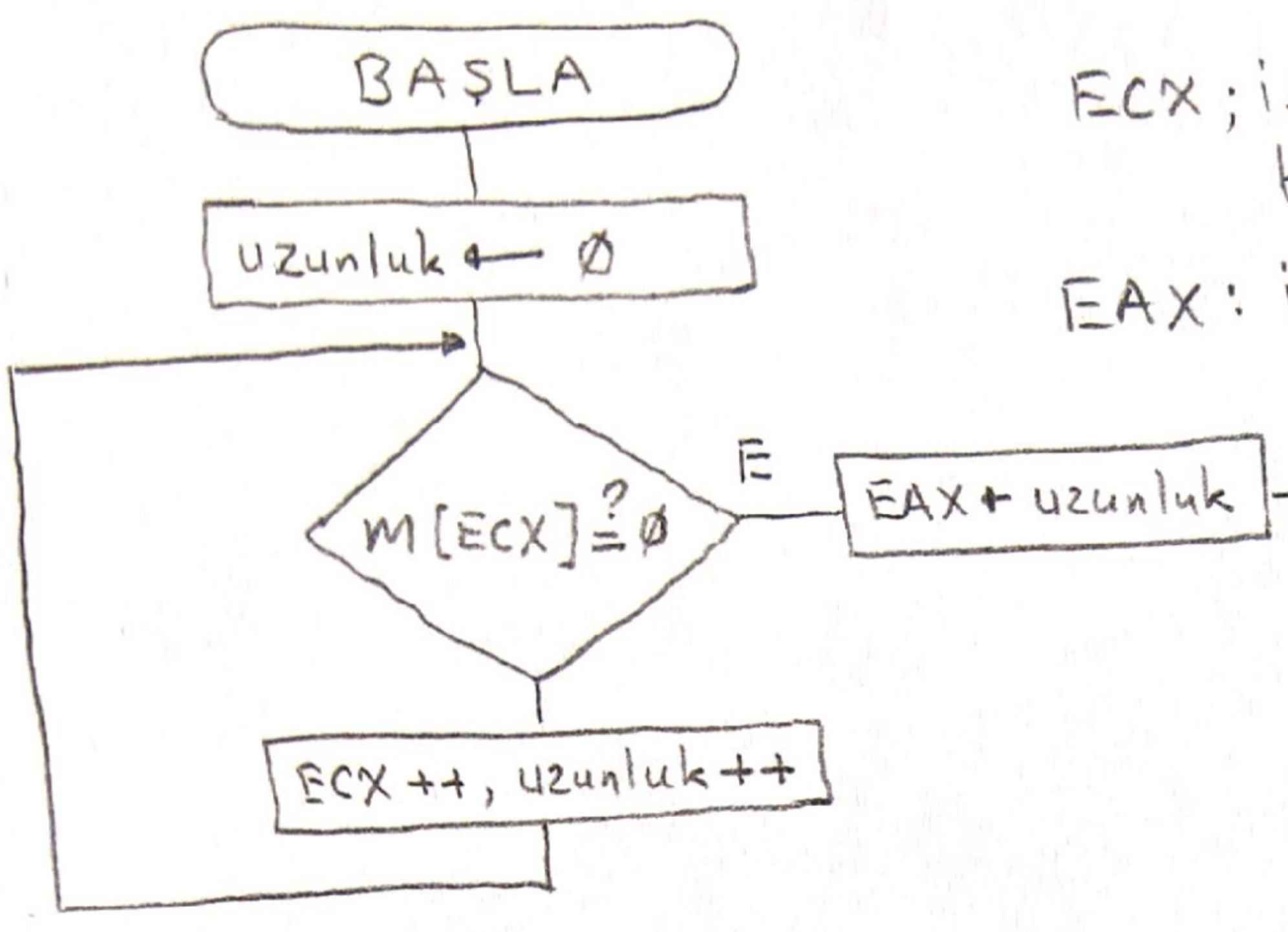
c. Compute speed-up (the ratio of Programmed I/O to DMA I/O) possible by the use of DMA block transfer. <=> DMA blok aktarımı ile mümkün olan hızlanmayı (Programlı G/Ç süresinin DMA G/Ç süresine oranı) hesaplayınız.

```
                 Clock cycles / Saat periyodu
      LDX BLOK            5      // IX <- BLOK (IX is an Index Register // IX indisli adresleme kaydedicisidir)
      LDA B #$FF          2      // B <- 255
DÖN:  LDAA, X             5      // A <- M[IX]
      STAA, KUYRUK        3      // Out (Memory Mapped) <- M[KUYRUK]
      INX                 4      // IX <- IX + 1
      DEC B               2      // B <- B - 1
      BNE DÖN             4      // Branch if not Z=1
```

$$\text{Hızlanma} = \frac{\text{Program denetimli G/Ç süresi}}{\text{Doğrudan bellek erişimli G/Ç süresi}} = \frac{4615\,\mu s}{38{,}4\,\mu s} \cong 120$$

3. From **en.wikibooks.org/wiki/C_Programming/string.h/strlen** : In the C standard library, ***strlen*** is a string function that determines the length of a C character string. Character strings are stored in an array of a data type called char. The end of a string is found by searching for the first null character in the array. Code ***StringLength*** function with prototype **unsigned int *StringLength*(char *)** to be called from C/C++ programs. <=> **en.wikibooks.org/wiki/C_Programming/string.h/strlen** 'den : Standart C kütüphanesinde, **strlen** bir C karakter dizisinin uzunluğunu belirleyen bir karakter dizisi işlevidir. Karakter dizileri **char** adı verilen veri tipi dizisinde saklanmaktadır. Karakter dizisinin sonu, dizideki ilk **null** karakteri arayarak bulunur. C/C++ programlardan çağrılabilecek ve işlev ön ürünü (function prototype) **unsigned int *StringLength*(char *)** olan ***StringLength*** işlevini kodlayınız.

a. Draw the flowchart of the **StringLength** algorithm. <=> **StringLength** algoritmasının akış çizgesini çiziniz.



ECX ; işleve parametre olarak verilen karakter dizisinin başlangıç adresini tutar (fastcall için)
EAX : işlevin geri döndürdüğü işaretsiz tamsayıyı tutar

b. Code **StringLength** algorithm in any high level language. <=> **StringLength** algoritmasını herhangi bir yüksek seviyeli dil ile kodlayınız.

```
uzunluk = 0
while (*p<>null)
{
   p++
   uzunluk ++
}
 return uzunluk
```

c. Code **StringLength** algorithm in Assembly language. <=> **StringLength** algoritmasını Assembly dili ile kodlayınız.

```
      xor eax, eax              ; uzunluk değerini tutan eax kaydedicisini sıfırla
dön:  mov dl, byte [ecx+eax]    ; karakter dizisinin sıradaki değerini al
      cmp dl, 0x0               ; null 'mı?
      je  çık                   ; dizi sonu bulundu, döngüden çık, çağıran programa geri dön
      inc eax                   ; dizi sonu olmadığı için sayacı artır
      j   dön                   ; dizi sonu aramaya devam et
Çık:  ret                       ; dizi uzunluğunu döndür
```