İTÜ Computer Engineering Department
Assoc.Prof. Feza BUZLUCA
Asst. Prof. Ayşe YILMAZER
26.04.2018

COMPUTER ARCHITECTURE 2ND MIDTERM SOLUTIONS

QUESTION 1: (35 Points)

a)

i)

In the given code, the value of FLAG as being zero represents that the synchronization is available and the value of FLAG as ($80) represents that the synchronization is not available.

- Let's assume that initially the value of the FLAG is zero, meaning that synchronization variable is available.
- P1 reads the value of FLAG from memory when executing CMP instruction and observes that the value of FLAG is zero.
- After P1 reads the value of FLAG but before it writes the new value of FLAG by running the MOV instruction to complete the synchronization operation, either P1 context switches out and P2 context switched in if time-sharing on a uniprocessor or P2 executes on another processor in the multiprocessor system.
- P2 executes CMP instruction and observes the value of FLAG as zero.
- When both of the processes continue executing since both P1 and P2 observed that the value of FLAG is zero, both of them will set the FLAG value to $80 and enter the critical section and causing a race condition.

The problem here is that the read-modify-write of the synchronization variable (FLAG) is not atomic because two instructions CMP and MOV can be divided by system interrupts (process switch). This suggests that we need an atomic instruction to implement a correct synchronization mechanism.

Senkronizasyon (semafor) değişkeninin (FALG) değeri sıfır ise kritik bölge erişime açıktır, değilse başka bir proses senkronizasyona girmiştir ve dolayısıyla kapalıdır

- *FLAG değişkenin ilk değerini sıfır olarak varsayalım.*
- *P1 prosesi CMP komutunu işletirken, FLAG değişkeninin değerini bellekten okur ve FLAG'in değerinin sıfır olduğunu gözlemler.*
- *P1 prosesi MOV komutunu yürütüp FLAG değerini değiştirerek senkronizasyonu tamamlamadan sistem kesmesi gelir, proses anahtarlaması olur P2 prosesi çalışmaya başlar.*
- *P2 prosesi CMP komutunu işletir (araya girerek, ya da diğer işlemci üzerinden erişerek) ve FLAG değişkeninin değerinin sıfır olduğunu görür.*
- *Her iki proses de FLAG değerini sıfır olarak okudukları için, çalışmaya devam ettiklerinde kritik kod bölümüne gireceklerdir. İki proses de ayni anda kritik alan içerisinde paylaşılan veriye erişim sağladıkları için veri bütünlüğü bozulacaktır.*

Buradaki problem, senkronizasyon değişkeni üzerinde gerçekleştirilen (oku-değiştir-yaz) işlemlerinin tek bir komut olmayıp CMP ve MOV komutlarının sistem kesmeleriyle bölünebilmesidir.

ii) Consider a system with a CPU and a DMAC (direct memory controller) and write the events step by step that cause the problem.

In this case, since we are talking about synchronization, a process (P1) is running on CPU to execute this code. And, let's imagine that another process (P2) is running on a device and accessing to the FLAG variable using DMA accesses.

- P1 reads the value of FLAG from memory when executing CMP instruction and observes that the value of FLAG is zero.
- After P1 reads the value of FLAG but before it writes the new value of FLAG to complete the synchronization operation, DMAC takes the bus and transfers the value FLAG to the memory of P2. Remember that DMA accesses are possible during execution of an instruction.
- P2 reads the value of FLAG executing CMP using DMA and observes the value of FLAG as zero.
- When both of the processes continue executing since both p1 and p2 observed that the value of FLAG is zero, both of them will set the FLAG value to $80 and enter the critical section and causing a race condition.

The problem here is that the read-modify-write of the synchronization variable (FLAG) is divisible with DMA accesses and this suggest that during read-modify-write of the synchronization bus must be locked to avoid the DMA accesses to the synchronization variable.

Bir işlemci üzerinde çalışan bir proses (P1) ve başka bir cihaz üzerinde çalışarak DMA üzerinden senkronizasyon değişkenine erişim yapabilen başka bir proses (P2) düşünelim.

- *FLAG değişkenin ilk değerini sıfır olarak varsayalım.*
- *P1 prosesi CMP komutunu işletirken, FLAG değişkeninin değerini bellekten okur ve FLAG'in değerinin sıfır olduğunu gözlemler.*
- *P1 prosesi FLAG değerini değiştirerek senkronizasyonu tamamlamadan DMAC yolu alır ve FLAG değişkeninin değerini P2'nin bellek alanına kopyalar. Hatırlatma: DMAC bir komutun içinde de yolu alarak belle erişebilir.*
- *P2 prosesi CMP komutunu işletir ve DMAC tarafından kopyalanan FLAG değişkeninin değerinin sıfır olduğunu görür.*
- *Her iki proses de FLAG değerini sıfır olarak okudukları için, çalışmaya devam ettiklerinde kritik kod bölümüne gireceklerdir. İki proses de ayni anda kritik alan içerisinde paylaşılan veriye erişim sağladıkları için veri bütünlüğü bozulacaktır.*

Buradaki problem, senkronizasyon işlemi sırasında, yani semafor değişkeni üzerinde oku-değiştir-yaz işlemi gerçekleştirilirken, sistem yolunun DMA erişimine açık olmasıdır.

b) (10 p)

```
TEST        TAS   FLAG      Tests the semaphore and sets it if necessary
            BMI   TEST
CRITICAL    ......          Critical section
            ......
END         CLR.B FLAG      Semaphore is cleared (unlock)
```

c) TAS instruction is an atomic instruction, executes as a single operation and performs the read-modify-write of the synchronization variable atomically.

- This instruction cannot be divided by interrupts
- During the execution of the TAS, AS (address strobe) remains asserted to lock the bus and disable the access to memory from other devices.

TAS komutu atomik, tek bir komuttur, senkronizasyon değişkeni üzerindeki oku-değiştir-yaz işlemleri bölünmez tek bir işlem olarak gerçekleştirilir.

- Bu komut kesmelerle bölünemez.
- TAS yürütülürken sistem yolu AS (address strobe) ile kilitlenir ve belleğe DMAC erişimleri engellenmiş olur.

QUESTION 2: (35 Points)

A CPU with 8-bit data bus, has two Interrupt Request inputs (**IRQ1** , **IRQ2**) and two related Interrupt Acknowledgement outputs (**INTA1**, **INTA2**). All signals are active at "1". If the CPU receives a request from any of the inputs (IRQx), it works with vectored interrupts and reads the interrupt vector number after the acknowledgement (INTAx=1) of the related interrupt when its Data acknowledgement input (**DACK**) is "1".

IRQ1 has higher priority than the IRQ2 (If there are simultaneous requests from both inputs, IRQ1 is accepted, INTA1=1).

In the interrupt (housekeeping) cycle of the IRQ1, interrupts from both inputs are disabled by clearing interrupt masks (IM1=0, IM2=0). Hence, interrupt service routines related to IRQ1 cannot be interrupted by other requests.

In the interrupt cycle of the IRQ2, interrupts only from IRQ2 are disabled by clearing the interrupt mask (IM2=0). Hence interrupt service routines related to IRQ2 can be interrupted only by requests from IRQ1.

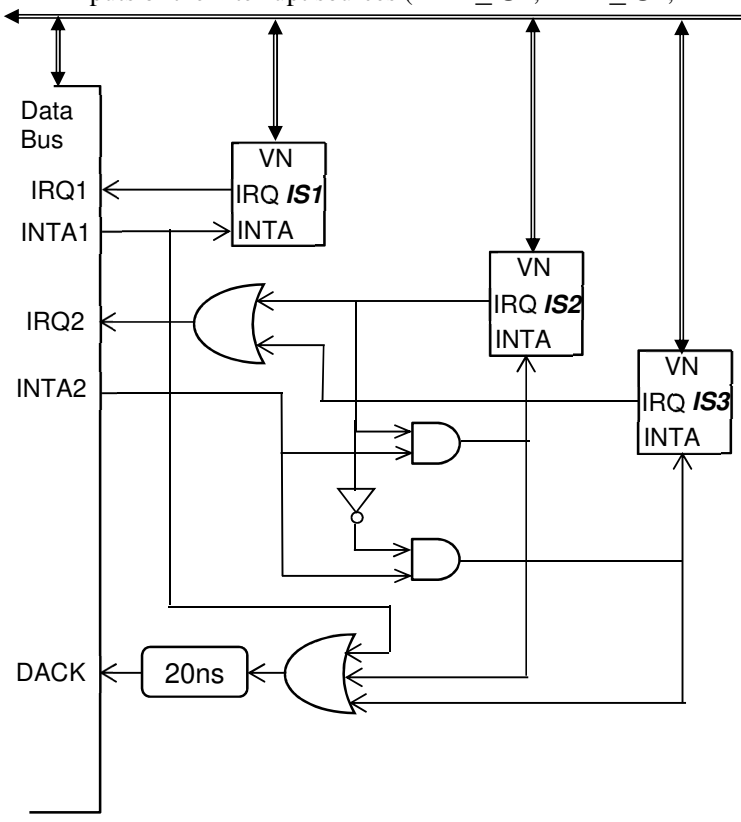
In this system, there are three devices that work as interrupt sources (*IS1*, *IS2*, *IS3*).

Each of these devices has an Interrupt Request output (**IRQ**), an Interrupt Acknowledgement input (**INTA**), and 8-bit vector number output (**VN**). 20 ns after the interrupt has been acknowledged (**INTA=1**) the device outputs its vector number at the **VN** and removes its request (**IRQ=0**).

Priority (precedence) order of the devices: *IS1* > *IS2* > *IS3*

The interrupt service routine related to *IS1* cannot be interrupted by *IS2* or *IS3*. Interrupt service routines related to *IS2* and *IS3* can be interrupted only by *IS1*.

a) Design and draw the system with the CPU, 3 devices (*IS1*, *IS2*, *IS3*) and the necessary logic gates. Write the logic expressions for the interrupt request inputs of the CPU (IRQ1 , IRQ2) and for the interrupt acknowledgement inputs of the interrupt sources (INTA_IS1, INTA_IS2, INTA_IS3). (15 points)



$$IRQ1 = IRQ_{IS1}$$
$$IRQ2 = IRQ_{IS2} + IRQ_{IS3}$$
$$INTA_{IS1} = INTA1$$
$$INTA_{IS2} = INTA2 \cdot IRQ_{IS2}$$
$$INTA_{IS3} = INTA2 \cdot \overline{IRQ_{IS2}}$$

b) Instruction cycle of the CPU has the following 4 states (phases):
1. Instruction fetch and decode: 60ns, 2. Operand fetch: 70ns, 3. Execution: 50ns, 4. Interrupt: 200ns.
Durations of all interrupt service routines (ISR) are 2000ns.
Assume that we start a clock (Clock = 0) when the CPU begins to run the program and all interrupts are enabled (IM1=1, IM2=1).
At Clock = 10ns the device *IS2* sends an interrupt request and then at Clock = 300ns the device *IS1* sends an interrupt request.
When (Clock =?) will the ISRs of the I*S1* and *IS2* start to run? Why? (20 points)

Remember; interrupt requests are checked after the completion of the instruction.
In this system, an instruction is completed at Clock= 180 ns (60+70+50).
Because of the interrupt request from IS2, the CPU enters the "4. interrupt" phase and starts housekeeping operations for IS2.
During the housekeeping operations the IS1 sends an interrupt request. The housekeeping operations (such as saving the return address and condition code) cannot be interrupted. Remember; interrupt requests are checked after the completion of the instruction.
After the housekeeping operations (200ns) at Clock= 380 ns (180 + 200) the ISR of IS2 starts to run.
ISR of IS2 at Clock= 380 ns.
Now IM1=1 IM2=0. Interrupts from IS1 are enabled.
After the completion of the 1st instruction of the ISR of the IS2 (Clock= 380 + 180 = 560 ns), the CPU checks interrupt requests, and enters the "4. interrupt" phase for IS1.
After the housekeeping operations (200ns) at Clock= 760 ns (560 + 200) the ISR of IS1 starts to run.
ISR of IS1 at Clock= 760 ns.

Hatırlatma: kesme istekleri komut tamamlandıktan sonra kontrol edilip değerlendirilir.
Bu sistemde bir komut Saat = 180 ns'de (60+70+50) tamamlanır.
IS2'den gelen kesme isteği nedeniyle MİB "4. kesme" çevrimine girer ve IS2 için hazırlık işlemlerine başlar.
Hazırlık işlemleri sırasında IS1 kesme isteği gönderir. Hazırlık işlemleri (geri dönüş adresinin ve durum bilgisinin saklanması gibi) kesilemezler Hatırlatma: kesme istekleri komut tamamlandıktan sonra kontrol edilip değerlendirilir.
Hazırlık işlemlerinden sonra (200ns) Saat = 380 ns'de (180 + 200) IS2'nin kesme hizmet programı çalışmaya başlar.
IS2'nin KHP'si Saat = 380 ns.
IM1=1 IM2=0 olur. IS1'den gelen kesme isteklerine izin vardır.
IS2'nin kesme hizmet programını birinci komutu tamamlandıktan sonra (Saat= 380 + 180 = 560 ns), MİB kesme isteklerini kontrol eder ve "4. kesme" çevrimine IS1'in hazırlık işlemleri için girer.
Hazırlık işlemlerinden sonra (200ns) Saat = 760 ns'de (560 + 200) IS1'in kesme hizmet programı çalışmaya başlar.
IS1 KHP'si Saat= 760 ns.

COMPUTER ARCHITECTURE 2ND MIDTERM (Question 3 of 3)

QUESTION 3: (30 Points)

a) Remember; interrupt requests are checked after the completion of the instruction.
In this system, an instruction is completed at Clock= 210 ns (50+20+70+30+40).
The IS sends the interrupt request at Clock = 250 ns as the CPU is in the fetch cycle of the second instruction.
The ISR can start after the housekeeping operations, if there are not any DMA requests.
Second instruction is completed at Clock = 420 ns (210+210).
The ISR starts at Clock = 620 ns (420+200).

Hatırlatma: kesme istekleri komut tamamlandıktan sonra kontrol edilip değerlendirilir.
Bu sistemde bir komut Saat = 210 ns'de (50+20+70+30+40) *tamamlanır.*
IS'den kesme isteği Saat = 250 ns'de MİB ikinci komutun alma çevrimindeyken gelir.
Eğer bu arada gelen DMA isteği yoksa hazırlık işlemlerinden sonra kesme hizmet programı çalışmaya başlar.
İkinci komut Saat = 420 ns'de (210+210) *tamamlanır.*
KHP Saat = 620 ns'de (420+200) başlar.

b)

DMAC sends bus requests as the CPU is running the ISR.
The first bus request comes as the CPU is decoding first instruction of the ISR. Because the CPU is not using the bus during decoding the DMAC can stat immediately to transfer the first word at Clock = 675 ns. Because the type of the DMAC is fly-by (implicit) and access time is 50 ns, the DMAC complete the transfer of the first word at Clock = 725 ns (675 +50). Other times can be calculated similarly.

1. Word: at Clock = 725 ns
2. Word: at Clock = 845 ns
3. Word: at Clock = 935 ns
4. Word: at Clock = 1035 ns

DMAC yol isteklerini, MİB kesme hizmet programını çalıştırırken gönderir.
İlk yol isteği, MİB KHP'nin ilk komutunu çözerken gelir. Komut çözme sırasında MİB yolu kullanmadığından DMAC ilk sözcüğün aktarımına = 675 ns'de hemen başlayabilir. DMAC örtülü "fly-by (implicit)" ve erişim süresi 50 ns olduğundan, DMAC ilk sözcüğün aktarımını Saat = 725 ns'de (675 +50) tamamlar. Diğer zamanlarda bezer şekilde hesaplanır.

1. Sözcük: Saat= 725 ns
2. Sözcük: Saat= 845 ns
3. Sözcük: Saat= 935 ns
4. Sözcük: Saat= 1035 ns