İTÜ Computer Engineering Department
29.05.2018
Assoc.Prof. Feza BUZLUCA
Asst. Prof. Ayşe YILMAZER

COMPUTER ARCHITECTURE FINAL EXAMINATION SOLUTIONS
BİLGİSAYAR MİMARİSİ FİNAL SINAVI ÇÖZÜMLERİ

QUESTION 1: (20 P)
SORU 1:

a) (5 p)

$$S_{n\to\infty} = \frac{t_n}{t_p} \Longrightarrow \boldsymbol{t_p = 60ns}$$

Second task will be completed one clock cycle after the first task.
İkinci iş birinci işten bir saat çevrimi sonra tamamlanır.

$T_2 = T_1 + t_p = 300 + 60 =$ **$T_2 = 360$ ns.**

b) (5 p)

$$S_{n=100} = \frac{100 * t_n}{T_{100}} = \frac{100 * t_n}{T_1 + 99 * t_p} = \frac{18000}{300 + 99 * 60} = \frac{300}{104} = \boldsymbol{S_{n=100} = 2.88}$$

c) (10 p)

$$k = \frac{T_1}{t_p} = \frac{300}{60} = \boldsymbol{k = 5}$$

The task is divided into 5 sub operations.
For the given pipeline, maximum theoretical speedup is **5** (if the task can be divided into 5 sub operations with the same durations). However, the speedup achieved by pipeline P if the array has an infinite number of elements is $S_{n\to\infty} = 3$. It means that the task is not divided into balanced sub operations.
So, the answer is "**YES, we should examine the structure of the pipeline P to design another pipeline with a higher speedup $S_{n\to\infty}$**".

*T işlemi 5 alt işleme bölünmüştür. Buna göre, soruda tarif edilen iş hattının maksimum teorik hızlanması **5**'tir (eğer iş 5 adet eşit süreli alt işleme bölünebilirse). Ancak dizi sonsuz sayıda eleman içerdiğinde, iş hattı P'nin sağladığı hızlanma $S_{n\to\infty} = 3$'tür. Demek ki iş, alt işlemlere dengeli olarak bölünmemiştir.*
*Buna göre yanıt "**EVET, iş hattı P'nin içyapısını inceleyerek daha yüksek bir hızlanmaya ($S_{n\to\infty}$) sahip başka bir iş hattı tasarlamaya çabalamalıyız**".*

Another possible solution:
If the task T was divided into balanced sub operations the clock cycle should be 180/5 = 36 ns.
However, the cycle time of P is 60 ns. Even if we consider delays of the registers this duration is too longer than 36 ns.

Diğer bir olası çözüm:
Eğer T işlemi alt işleme dengeli olarak bölünmüş olsaydı saat çevriminin süresi 180/5 = 36 ns olurdu. Ancak P iş hattının saat çevriminin süresi 60ns'dir. Saklayıcı gecikmeleri dikkate alındığında bile bu süre 36 ns'den çok uzundur.

QUESTION 2: (20 P)
SORU 2:
a) (10 p)
There are two cases that we need to consider:
There is no memory access during the execution of program (i.e. data and instructions are supplied from the cache), so the memory accesses through the DMAC do not affect the program execution. The execution of the program would be 1000ns. This is the best possible timing.
In the second case, there is no overlap between program execution and the memory accesses using DMAC. In this case, the execution time of the program would be (time to transfer 10 words + execution time of the program without DMAC request) = 2000ns.
Time to transfer 10 words = 10 * (50 + 50) = 1000ns
Execution time of the program without DMAC request = 1000ns
This is the worst possible timing.
Therefore, we can say, the execution of the program takes **between 1000ns and 2000ns**.

Note: With a good explanation, the answers stating that the execution time of the program is between 1000ns and 1900ns also get the full points.

Göz önüne almamız gereken iki durum söz konusudur:
Programın çalışması sırasında bellek, dolayısıyla veri yolu erişimine gerek duyulmaz (örneğin tüm veri ve komutlar cep-bellekten sağlanır). Bu durumda programın çalışması DMA'yin veri aktarımından etkilenmez ve programın çalışması toplamda 1000ns sürer. Bu mümkün olan en kısa süredir.

Diğer durumda ise, programın çalışması ile DMA'yin veri aktarımı hiç örtüştürülemez (mesela programın komutları çalıştırılırken, işlemci sürekli veri yolunu kullanmaktadır) Bu durumda işlemci ve DMA veri yolunu dönüşümlü kullanırlar. Bu durumda programın çalışması:
*10 veri sözcüğünü aktarım zamanı (10 * 50 + 50 = 1000 ns)+ programın DMA erişimi ile bölünmeden çalışma süresi (1000 ns) = 2000ns.*

Bu ise olası en uzun suredir.

*Dolayısıyla diyebiliriz ki, programın çalışması **1000ns ile 2000ns arasında** sürer.*
Not: 2000ns yerine 1900ns da doğru olarak kabul edilmiştir.

b) (10 p)
With fly-by, DMAC provides single address. Accessing two memory units requires specifying both destination and source addresses and most of the time, it is very unlike to have same source and destination addresses. Therefore, **Flow-through DMAC** accesses must be used.

---

*Örtülü tipte DMA tek bir adres ile ya kaynak ya da hedef adresini belirtebilir. İki bellek ünitesi arasında veri aktarımı için hem kaynak hem de hedef adresi belirtilmesi gerekir. Çoğunlukla, kaynak ve hedef adresler ayni olmayacağından, **açık tipte** DMA kullanılmalıdır.*

QUESTION 3: ***SORU 3:*** (35 P)

a) Address: 20 bits
Data: 8bits
Data cache: 1KB = 2^{10} Byte
Block size: 8Byte = 2^3 Byte
$2^{10} / 2^3 = 2^7$ = 128 Frame in the Data cache

i) Byte Offset = 3 Bits,
Index = 7 Bits,
Tag = 10 Bits.

ii) $00014 (hex) = **0000 0000 0000 0001 0100** (bin)
Tag = $0
Index = $2
Byte offset = $4
Frame #2 is checked, cache is initially empty, so frame is invalid and access to $00014 is a **Miss**. Data block is brought into frame #2, frame set to valid, and tag set to $0. Byte #4 from block is read/write.

$00016 (hex) = **0000 0000 0000 0001 0110** (bin)
Tag = $0
Index = $2
Byte offset = $6
Frame #2 is checked, frame is valid, tags match, so access to $00016 is a **Hit**. Byte #6 is read/write.

$00019 (hex) = **0000 0000 0000 0001 1001** (bin)
Tag = $0
Index = $3
Byte offset = $1
Frame #3 is checked, cache was initially empty, so frame is invalid and access to $00019 is a **Miss**. Data block is brought into frame #3, frame set to valid, and tag set to $0. Byte #1 is read/write.

$00410 (hex) = **0000 0000 0100 0001 0000** (bin)
Tag = $1
Index = $2
Byte offset = $0
Frame #2 is checked, frame is valid, tags do not match, and so, access to $00410 is a **Miss**. Data block is brought into frame #2, frame set to valid, and tag set to $1. Byte #0 is read/write.

$00017 (hex) = **0000 0000 0000 0001 0111** (bin)
Tag = $0
Index = $2
Byte offset = $7

Frame #2 is checked, frame is valid, tags do not match, and so, access to $00017 is a **Miss**. Data block is brought into frame #2, frame set to valid, and tag set to $0. Byte #7 is read/write.

b) Address: 20 bits
Data: 8bits
Data cache: 1KB = 2^{10} Byte
Block size: 16Byte = 2^4 Byte
$2^{10} / 2^4 = 2^6$ = 64 Frame in the Data cache

$00014 (hex) = **0000 0000 0000 0001 0100** (bin)
Tag = $0
Index = $1
Byte offset = $4
Frame #1 is checked, cache is initially empty, so frame is invalid and access to $00014 is a **Miss**. Data block is brought into frame #1, frame set to valid, and tag set to $0. Byte #4 is read/write.

$00016 (hex) = **0000 0000 0000 0001 0110** (bin)
Tag = $0
Index = $1
Byte offset = $6
Frame #1 is checked, frame is valid, tags match, so access to $00016 is a **Hit**. Byte #6 is read/write.

$00019 (hex) = **0000 0000 0000 0001 1001** (bin)
Tag = $0
Index = $1
Byte offset = $9
Frame #1 is checked, frame is valid, tags match, so access to $00019 is a **Hit**. Byte #9 is read/write.

$00410 (hex) = **0000 0000 0100 0001 0000** (bin)
Tag = $1
Index = $1
Byte offset = $0
Frame #1 is checked, frame is valid, tags do not match, and so, access to $00410 is a **Miss**. Data block is brought into frame #1, frame set to valid, and tag set to $1. Byte #0 is read/write.

$00017 (hex) = **0000 0000 0000 0001 0111** (bin)
Tag = $0
Index = $1
Byte offset = $7

Frame #1 is checked, frame is valid, tags do not match, and so, access to $00017 is a **Miss**. Data block is brought into frame #1, frame set to valid, and tag set to $0. Byte #7 is read/write.

c) Address: 20 bits
Data: 8bits
Data cache: 1KB = 2^{10} Byte, 2-way set associative
Block size: 8Byte = 2^3 Byte
$2^{10} / 2^3 / 2 = 2^6 = 64$ Frame in the Data cache

$00014 (hex) = **0000 0000 0000 0001 0100** (bin)
Tag = $0
Index = $2
Byte offset = $4
Frames in set #2 are checked, cache is initially empty, so two frames in the set are invalid and so, the access to $00014 is a **Miss**. Ways #1 in the set is allocated. Data block is brought into frame #1 in set #1, frame set to valid, and tag set to $0. Byte #4 is read/write.

$00016 (hex) = **0000 0000 0000 0001 0110** (bin)
Tag = $0
Index = $2
Byte offset = $6
Frames in set #2 are checked, frame #1 is valid, tags match, so access to $00016 is a **Hit**. Byte #6 is read/write.

$00019 (hex) = **0000 0000 0000 0001 1001** (bin)
Tag = $0
Index = $3
Byte offset = $1
Frames in set #3 are checked, cache was initially empty, so two frames in the set are invalid and so, the access to $00019 is a **Miss**. Ways #1 in the set is allocated. Data block is brought into frame #1 in set #3, frame set to valid, and tag set to $0. Byte #1 is read/write.

$00410 (hex) = **0000 0000 0100 0001 0000** (bin)
Tag = $1

Index = $2
Byte offset = $0
Frames in set #2 are checked, frame #2 is invalid, frame #1 is valid but tags do not match, and so, the access to $00410 is a **Miss**. Ways #2 in the set is allocated. Data block is brought into frame #2, frame set to valid, and tag set to $1. Byte #0 is read/write.

$00017 (hex) = **0000 0000 0000 0001 0111** (bin)
Tag = $0
Index = $2
Byte offset = $7

Frames in set #1 are checked, frame #1 is valid but tags do not match, frame #2 is valid and tags match, and so, access to $00017 is a **Hit**. Byte #7 is read/write.

---

a) Adres: 20 bit
Data: 8 bit
Veri cep-belleği: 1KB = 2^{10} Byte
Blok boyu: 8Byte = 2^3 Byte
$2^{10} / 2^3 = 2^7$ = 128 çerçeve (cep-bellekte)

i) Veri Ofseti (Byte Offset)= 3 Bit,
Çerçeve no = 7 Bit,
Takı (tag) = 10 Bit.

*ii) $00014 (hex) = **0000 0000 0000 0001 0100** (bin)*
Takı = $0
Çerçeve No = $2
Veri Ofseti = $4
*Çerçeve 2'ye bakılır, cep bellek başlangıçta boş, dolayısıyla çerçeve geçersiz veri içerir $00014'e erişim **Iskadır**. Veri bloğu 2 nolu çerçeveye getirilir, çerçeve geçerli olarak işaretlenir, takısı $0 olarak yazılır ve 4 nolu sekizlik veriye erişilir.*

*$00016 (hex) = **0000 0000 0000 0001 0110** (bin)*
Takı = $0
Çerçeve No = $2
Veri Ofseti = $6
*Çerçeve 2'ye bakılır, çerçeve 2'de geçerli veri vardır, takılar eştir, $00016'ya erişim **Vurudur**. 6 nolu sekizlik veriye erişilir.*

*$00019 (hex) = **0000 0000 0000 0001 1001** (bin)*
Takı = $0
Çerçeve No = $3
Veri Ofseti = $1
*Çerçeve 3'ye bakılır, cep bellek başlangıçta boş, dolayısıyla çerçeve geçersiz veri içerir, $00019'a erişim **Iskadır**. Veri bloğu 3 nolu çerçeveye getirilir, çerçeve geçerli olarak işaretlenir, takısı $0 olarak yazılır ve 1 nolu sekizlik veriye erişilir.*

*$00410 (hex) = **0000 0000 0100 0001 0000** (bin)*
Takı = $1
Çerçeve No = $2
Veri Ofseti = $0
*Çerçeve 2'ye bakılır, çerçeve geçerli veri içerir fakat takılar ayni değildir, dolayısıyla $00410'a erişim **Iskadır**. Veri bloğu 2 nolu çerçeveye getirilir, çerçeve geçerli olarak işaretlenir, takısı $1 olarak yazılır ve 0 nolu sekizlik veriye erişilir.*

*$00017 (hex) = **0000 0000 0000 0001 0111** (bin)*
Takı = $0
Çerçeve No = $2

Veri Ofseti = $7

*Çerçeve 2'ye bakılır, çerçeve geçerli veri içerir fakat takılar ayni değildir, dolayısıyla $00017'ye erişim **Iskadır**. Veri bloğu 2 nolu çerçeveye getirilir, çerçeve geçerli olarak işaretlenir, takısı $0 olarak yazılır ve 7 nolu sekizlik veriye erişilir.*

b) *Adres: 20 bit*
Data: 8 bit
Veri cep belleği: 1KB = 2^{10} Byte
Blok boyu: 16Byte = 2^4 Byte
$2^{10} / 2^4 = 2^6$ = 64 Çerçeve (cep bellek içinde)

*$00014 (hex) = **0000 0000 0000 0001 0100** (bin)*
Takı = $0
Çerçeve No = $1
Veri Ofseti = $4
*Çerçeve 1'e bakılır, cep bellek başlangıçta boş, dolayısıyla çerçeve geçersiz veri içerir $00014'e erişim **Iskadır**. Veri bloğu 1 nolu çerçeveye getirilir, çerçeve geçerli olarak işaretlenir, takısı $0 olarak yazılır ve 4 nolu sekizlik veriye erişilir.*

*$00016 (hex) = **0000 0000 0000 0001 0110** (bin)*
Takı = $0
Çerçeve No = $1
Veri Ofseti = $6
*Çerçeve 1'e bakılır, çerçeve 21de geçerli veri vardır, takılar eştir, $00016'ya erişim **Vurudur**. 6 nolu sekizlik veriye erişilir.*

*$00019 (hex) = **0000 0000 0000 0001 1001** (bin)*
Takı = $0
Çerçeve No = $1
Veri Ofseti = $9
*Çerçeve 1'ye bakılır, çerçeve 1'de geçerli veri vardır, takılar eştir, $00019'a erişim **Vurudur**. 9 nolu sekizlik veriye erişilir.*

*$00410 (hex) = **0000 0000 0100 0001 0000** (bin)*
Takı = $1
Çerçeve No = $1
Veri Ofseti = $0
*Çerçeve 1'ye bakılır, çerçeve geçerli veri içerir fakat takılar ayni değildir, dolayısıyla $00410'a erişim **Iskadır**. Veri bloğu 1 nolu çerçeveye getirilir, çerçeve geçerli olarak işaretlenir, takısı $1 olarak yazılır ve 0 nolu sekizlik veriye erişilir.*

*$00017 (hex) = **0000 0000 0000 0001 0111** (bin)*
Takı = $0
Çerçeve No = $1
Veri Ofseti = $7

*Çerçeve 1'ye bakılır, çerçeve geçerli veri içerir fakat takılar ayni değildir, dolayısıyla $00017'ye erişim **Iskadır**. Veri bloğu 1 nolu çerçeveye getirilir, çerçeve geçerli olarak işaretlenir, takısı $0 olarak yazılır ve 7 nolu sekizlik veriye erişilir.*

c) *Adres: 20 bit*
Data: 8 bit
Veri cep belleği : 1KB = 2^{10} Byte, 2 çerçeve kümeli ve çağrışımlı
Blok boyu: 8Byte = 2^3 Byte
$2^{10} / 2^3 / 2 = 2^6$ = 64 çerçeve (cep bellek içinde)

$00014 (hex) = 0000 0000 0000 0001 0100 (bin)
Takı = $0
Çerçeve No = $2
Veri Ofseti = $4
*2 nolu küme içinde çerçevelere bakılır, cep bellek başlangıçta boş, dolayısıyla çerçeveler geçersiz veri içerir $00014'e erişim **Iskadır**. Veri bloğu 2 nolu küme içinde 1 nolu çerçeveye getirilir, çerçeve geçerli olarak işaretlenir, takısı $0 olarak yazılır ve 4 nolu sekizlik veriye erişilir.*

$00016 (hex) = 0000 0000 0000 0001 0110 (bin)
Takı = $0
Çerçeve No = $2
Veri Ofseti = $6
*2 nolu küme içinde çerçevelere bakılır, çerçeve 1'de geçerli veri vardır, takılar eştir, $00016'ya erişim **Vurudur**. 6 nolu sekizlik veriye erişilir.*

$00019 (hex) = 0000 0000 0000 0001 1001 (bin)
Takı = $0
Çerçeve No = $3
Veri Ofseti = $1
*3 nolu küme içinde çerçevelere bakılır, cep bellek başlangıçta boş, dolayısıyla çerçeveler geçersiz veri içerir $00019'e erişim **Iskadır**. Veri bloğu 3 nolu küme içinde 1 nolu çerçeveye getirilir, çerçeve geçerli olarak işaretlenir, takısı $0 olarak yazılır ve 1 nolu sekizlik veriye erişilir.*

$00410 (hex) = 0000 0000 0100 0001 0000 (bin)
Takı = $1
Çerçeve No = $2
Byte offset = $0
*2 nolu küme içinde çerçevelere bakılır, çerçeve 1'de geçerli veri vardır fakat takılar uyuşmaz, çerçeve 2 ise bostur. Dolayısıyla $00410'a erişim **Iskadır**. Veri bloğu 2 nolu küme içinde 2 nolu çerçeveye getirilir, çerçeve geçerli olarak işaretlenir, takısı $1 olarak yazılır ve 0 nolu sekizlik veriye erişilir.*

$00017 (hex) = 0000 0000 0000 0001 0111 (bin)
Takı = $0
Çerçeve No = $2
Veri Ofseti = $7
*2 nolu küme içinde çerçevelere bakılır, çerçeve 1 ve 2'de geçerli veri vardır, , çerçeve 1'in takısı eştir, dolayısıyla $00017'ya erişim **Vurudur**. 7 nolu sekizlik veriye erişilir.*

QUESTION 4: (25 P)
SORU 4:

a)
i) (5p)

CPU1:	**CPU2:**	**Main Memory:**
Exclusive (*Özel*)	Invalid (*Geçersiz*)	Valid (*Geçerli*)

ii) (5p)

CPU1:	**CPU2:**	**Main Memory:**
Modified (*Değiştirilmiş*)	Invalid (*Geçersiz*)	Invalid (*Geçersiz*)

iii) (5p)

CPU1:	**CPU2:**	**Main Memory:**
Shared (*Paylaşılan*)	Shared (*Paylaşılan*)	Valid (*Geçerli*)

b) (10 p)

CPU1:	**CPU2:**	**CPU3:**	**Main Memory:**
Shared	Invalid	Shared	Valid
Shared	Shared	Invalid	Valid
Shared	Shared	Shared	Valid