Tahmini Ders İçeriği
(Tentative Couse Schedule – Syllabus)

1. Hafta: Sayı sistemleri, onluk/ikilik taban sayı gösterimleri, mantıksal kapılar, computer system overview, başarım (performance)

2. Hafta: 2'lik tabanda işaretli sayılar, mikroişlemci tarihi, benchmarking, başarım,

3. Hafta: Başarım, Amdahl yasası, RISC-V development Environment, Verilog HDL ile Birleşik (Combinational) devreler

4. Hafta:, Verilog HDL ile sıralı (sequential) mantıksal devre ve sonlu durum makinası tasarımı, timing analysis

5. Hafta: Aritmetik devre tasarımları: Toplama, çıkarma, çarpma, bölme, trigonometri, square-root, hyperbolic, exponential, logarithm

6. Hafta: Fixed ve Floating-Point sayı gösterimleri

7. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

8. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

9. Hafta: Tek-çevrim işlemci tasarımı (single-cycle CPU)

10. Hafta: Çok-çevrim işlemci tasarımı (multi-cycle CPU)

11. Hafta: Boruhatlı işlemci tasarımı (pipelined CPU)

12. Hafta: Bellek sistemi ve hiyerarşisi

13. Hafta: İleri mimari konuları: Branch prediction, superscalar cpu, out-of-order execution, multi-core systems

14. Hafta: Gömülü sistemler, mikrodenetleyiciler, SoCs

VİZE SORU VE ÇÖZÜMLERİ

1) **[15 PUAN]** Aşağıdaki boşlukları uygun şekilde doldurun.

3'lük tabanda işlem yaparken 0, 1, 2 rakamları kullanılabilir.

2'lik tabanda verilen 10100011101010101011110001 sayısı 8'lik tabanda 50725361 olarak ifade edilir.

2'lik tabanda verilen 10100011101010101011110001 sayısı 16'lık tabanda A3AAF1 olarak ifade edilir.

6'lık tabanda verilen 135.13 sayısı 10'luk tabanda 59.25 olarak ifade edilir.

1*(6^2) + 3*(6^1) + 5*(6^0) + 1*(1/6) + 3*(1/36) = 5 + 18 + 36 + 1/6 + 1/12 = 59.25

İşaretli 2'ye tümleyen (2's complement) metodunda 1100101011 sayısı 10'luk tabanda -213 olarak ifade edilir.

-1*(2^9) + 1*(2^8) + 1*(2^5) + 1*(2^3) + 1*(2^1) + 1*(2^0) = -512+256+32+8+2+1 = -213

https://github.com/mbaykenar/computer-architecture/blob/main/exams/BZ403_midterm_soln.pdf

VİZE SORU VE ÇÖZÜMLERİ

2) **[20 PUAN]** Aşağıda C dilinde verilen kodu RISC-V assembly dilinde yazınız.

```
int a,b,c,d,e;

a = 7; b = 4; c = 6; d = 3;

if ((a-b) == (c-d))

   e = (a-b) + (c-d);

else

   e = (a+b) – (c+d);
```

NOT: a,b,c,d,e değişkenleri sırasıyla a0,a1,a2,a3,a4 registerlarında tutulacaktır. RISCV instructionları yazarken registerları a0 ya da karşılığı olan x10 şeklinde istediğiniz gibi yazabilirsiniz. RISCV register tablosu ve RV32I instruction listesi aşağıda verilmiştir:

https://github.com/mbaykenar/computer-architecture/blob/main/exams/BZ403_midterm_soln.pdf

VİZE SORU VE ÇÖZÜMLERİ

```
main:
    # Load values of a, b, c, and d into registers
    addi a0, x0, 7
    addi a1, x0, 4
    addi a2, x0, 6
    addi a3, x0, 3

    # Compute (a - b)
    sub t0, a0, a1

    # Compute (c - d)
    sub t1, a2, a3

    # Compare (a - b) with (c - d)
    beq t0, t1, equal

    # If not equal, compute (a + b)
    add t2, a0, a1

    # Compute (c + d)
    add t3, a2, a3

    # Compute (a + b) - (c + d)
    sub t4, t2, t3

    # Copy the result in e (a4)
    add a4, x0, t4
    j end

equal:
    # If equal, compute (a - b) + (c - d)
    add t4, t0, t1

    # Copy the result in e (a4)
    add a4, x0, t4

end:
    # End of program
    nop
```

https://github.com/mbaykenar/computer-architecture/blob/main/exams/BZ403_midterm_soln.pdf

VİZE SORU VE ÇÖZÜMLERİ

3) **[20 PUAN]** Aşağıdaki sorulara cevap veriniz:

a) Bilgisayarda stack bellekte nasıl bir yerdedir? Stack'in işlevi nedir? Stack overflow hatası hangi durum ya da durumlarda meydana gelebilir?

Stack pointer genel olarak bellekte yüksek bir adreste tanımlı olarak başlar ve aşağı doğru sabit bir alan kaplar. Prosedür çağrıldığında prosedür bitip de program eski durumuna dönebilmesi için bazı registerların bellekte saklanması gerekir. Bu registerların saklanacağı adres alanı stack olarak tanımlanır ve LIFO mantığıyla çalışır.

Stack adres uzayının alt tarafında dinamik verilerin saklanması için kullanılan heap alanı bulunur. Dinamik bellek gereksiniminin çok fazla olması, ya da çok fazla prosedür çağrılarak stack'e çok fazla veri yazılması nedeniyle yeni prosedür çağrılarında stack alanının dışında bir alanı kullandığında stack overflow hatası meydana gelir.

https://github.com/mbaykenar/computer-architecture/blob/main/exams/BZ403_midterm_soln.pdf

VİZE SORU VE ÇÖZÜMLERİ

b) RISC ve CISC nedir? Aralarındaki farklar nelerdir?

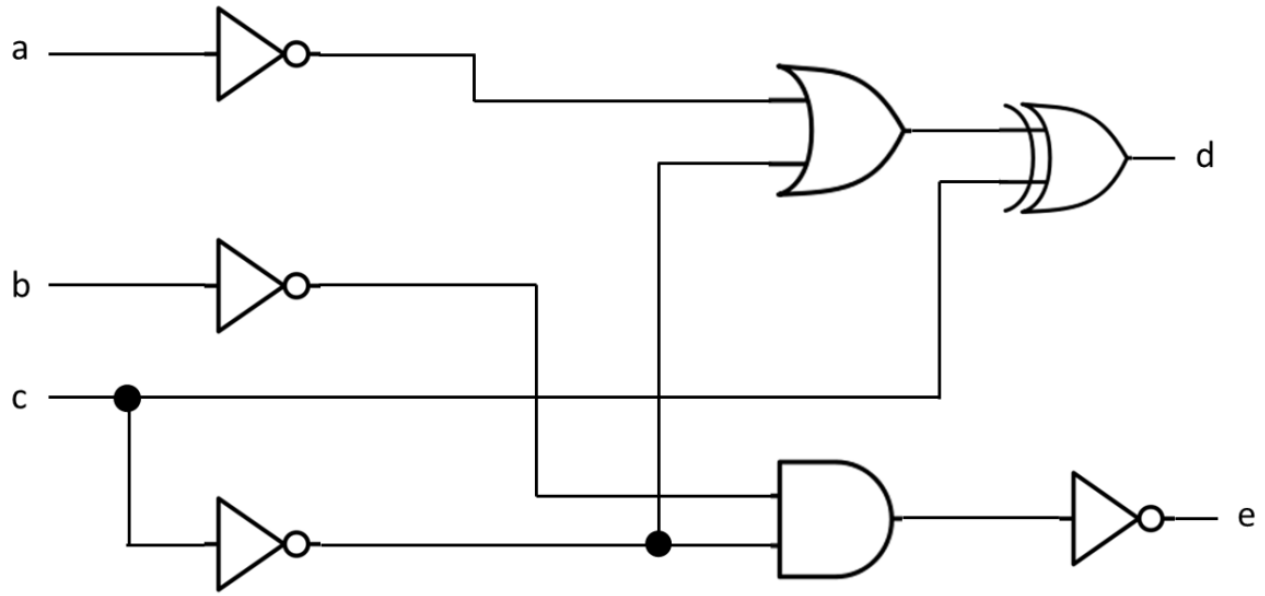
RISC açılımı reduced instruction set architecture ve CISC açılımı complex instruction set architecture. İlk çıkan işlemciler CISC ISA yapısındaydı, CISC yapısında karmaşık instructionlar kullanılarak bir instruction ile pek çok iş yapabilmek mümkün olabiliyordu. Bunun faydası, sınırlı kapasitesi olan bellekte az yer harcamaktı. Yine CISC mimarisinde işlemlerin operandları hem registerlar hem de bellekteki veriler olabiliyordu. 80'li yıllardan itibaren daha basit instructionlara sahip ve sadece registerlar üzerinde işlem gerçekleştiren RISC ISA mimarileri popülerleşti. RISC felsefesi:

- Instruction'ların hepsi eşit uzunlukta olsun
- Az sayıda register yeterli, örneğin 32 adet
- Az sayıda instruction type olsun ve field'lar aynı bit alanlarında olsun
- ALU işlemleri registerlar arasında olsun, direk bellekten olmasın

https://github.com/mbaykenar/computer-architecture/blob/main/exams/BZ403_midterm_soln.pdf

VİZE SORU VE ÇÖZÜMLERİ

c) Aşağıda verilen devredeki en uzun yol gecikmesi (critical path delay) nedir? Kapı gecikmeleri NOT 1 ns, AND ve OR 2 ns, XOR 4 ns. Kablo gecikmesi, bir kapının çıkışından diğer kapının girişine, giriş ve çıkış sinyalleri için de 1 ns. Devrenin giriş ve çıkışındaki kablo gecikmeleri hesaba katmayı unutmayın.



a sinyalinde d sinyaline giden hat en uzun yol gecikmesine sahiptir. a(kablo) + not + kablo + or + kablo + xor + kablo = 1 + 1 + 1 + 2 + 1 + 4 + 1 = 11 ns

d) c şıkkında verilen devrede çıkış sinyallerini giriş sinyalleri cinsinden bool fonksiyonu şeklinde yazınız.

d = (a' + c') ^ c

e = (b'c')'

https://github.com/mbaykenar/computer-architecture/blob/main/exams/BZ403_midterm_soln.pdf

VİZE SORU VE ÇÖZÜMLERİ

4) **[15 PUAN]**

Aynı instruction set architecture için iki farklı işlemci tasarımı düşünün. ISA'da bulunan instructionlar, CPI (cycle per instruction) değerlerine göre A, B, C ve D sınıfları olmak üzere dört sınıfa ayrılabilir.

P1 işlemcisi 2,5 GHz saat hızına sahiptir ve A,B,C,D instructionları için CPI değerleri sırasıyla 1, 2, 3 ve 3'tür.
P2 işlemcisi 3 GHz saat hızına sahiptir ve A,B,C,D instructionları için CPI değerleri sırasıyla 2, 2, 2 ve 2'dir.

Toplam instruction sayısı 1 milyon olan bir program incelendiğinde instruction tipleri şu şekilde dağılım göstermiştir: %10 A sınıfı, %20 B sınıfı, %50 C sınıfı ve %20 D sınıfı. Bu program için:

a) Hangi işlemci daha hızlıdır ve yavaş olana göre hızlı olma oranı kaçtır? Yaptığınız işlemleri, hesapları göstererek söyleyiniz.

CPI(P1) = 0.1*1 + 0.2+2 + 0.5*3 + 0.2*3 = 0.1 + 0.4 + 1.5 + 0.6 = 2.6

CPI(P2) = 0.1*2 + 0.2+2 + 0.5*2 + 0.2*2 = 0.2 + 0.4 + 1 + 0.4 = 2.0

YürütmeZamanı(P1) = Instruction Sayısı * CPI * Period = 10^6 * 2.6 * (1/2.5) *10^-9 = (2.6/2.5)*10^-3 = 1.04 ms

YürütmeZamanı(P2) = Instruction Sayısı * CPI * Period = 10^6 * 2.0 * (1/3.0) *10^-9 = (2.0/3.0)*10^-3 = 0.66 ms

YürütmeZamanı(P1) > YürütmeZamanı(P2) → P2, P1'den 1.04/0.66 = 1.56 → %56 daha hızlıdır

https://github.com/mbaykenar/computer-architecture/blob/main/exams/BZ403_midterm_soln.pdf

VİZE SORU VE ÇÖZÜMLERİ

b) İki işlemci tasarımı için ortalama CPI değerlerini hesaplayınız. Yaptığınız işlemleri, hesapları göstererek söyleyiniz.

CPI(P1) = 0.1*1 + 0.2+2 + 0.5*3 + 0.2*3 = 0.1 + 0.4 + 1.5 + 0.6 = 2.6

CPI(P2) = 0.1*2 + 0.2+2 + 0.5*2 + 0.2*2 = 0.2 + 0.4 + 1 + 0.4 = 2.0

c) İki işlemci için de toplam gereken saat vuruşu sayısı nedir hesaplayınız. Yaptığınız işlemleri, hesapları göstererek söyleyiniz.

SaatVuruşu(P1) = Instruction Sayısı * CPI = 10^6 * 2.6

SaatVuruşu(P2) = Instruction Sayısı * CPI = 10^6 * 2.0

https://github.com/mbaykenar/computer-architecture/blob/main/exams/BZ403_midterm_soln.pdf

VİZE SORU VE ÇÖZÜMLERİ

5) **[30 PUAN]**

3 adet sayıdan en büyüğünü bulan algoritmayı risc-v assembly dilinde yazınız. Gereksinimler:

- 3 adet sayı, bir array içerisinde bulunmaktadır (Örneğin my_array[0], my_array[1], my_array[2]). Arrayın 3 elemanı da 32-bit genişliğindedir ve bellekte bulunmaktadır. Arrayin ilk elemanının (my_array[0]) adresinin x10 registerında bulunduğu bilinmektedir.

- İşlem sonucunda bulunan en büyük sayı belleğe kaydedilecektir ve bellekte yazılacağı adres x11 registerındadır.

```
# initializations
# NOTE: You did not have to write initialization instructions
# These are just for testing purpose

initializations:
    addi x10,x0,0x100
    addi x11,x0,0x110
    addi t0, x0, 4
    sw t0, 0(x10)
    addi t0, x0, 7
    sw t0, 4(x10)
    addi t0, x0, 10
    sw t0, 8(x10)
```

```
# Find max of 3 numbers

main:
    addi t0, x0, 0
    lw t1, 0(x10)
    lw t2, 4(x10)
    lw t3, 8(x10)
    bgt t1, t0, check_t1_t2   # Compare t1 with t0
    # If t1 <= t0, go to comparing t2 with t0
    bgt t2, t0, check_t2_t3   # Compare t2 with t0
    # If t2 <= t0, the maximum number is t3
    addi t0, t3, 0          # Move t3 to t0
    j store_result
```

```
check_t1_t2:
    addi t0, t1, 0          # For now t1 is the largest
    # If t1 is also larger than t2, then check t1-t3
    bgt t1, t2, check_t1_t3   # Compare t1 with t3
    # If t2 is larger than t1, then check t2-t3
    j check_t2_t3   # if t2 >= t3 then t2 is the largest

check_t1_t3:
    addi t0, t1, 0          # For now t1 is the largest
    bgt t1, t3, store_result  # Compare t1 with t3
    # If t1 <= t3, the maximum number is t3
    addi t0, t3, 0          # Move t3 to t0
    j store_result

check_t2_t3:
    addi t0, t2, 0          # For now t2 is the largest
    bgt t2, t3, store_result  # Compare t2 with t3
    # If t2 <= t3, the maximum number is t3
    addi  t0, t3, 0         # Move t3 to t0
```

```
store_result:
    sw t0, 0(x11)

end:
    nop
```

RISC-V RV32I ISA CPU DESIGN

Amaç RV32I'da mevcut 47 instruction'un hepsini çalıştırabilen bir işlemci tasarımı

Tek dönem içerisinde gerçekleştirmek çok mümkün gözükmüyor, aşama aşama ilerlemek daha makul

İlk etapta implement edilecek instructionlar: **lw**, **sw**, **add**, **sub**, **and**, **or**, **beq** (Hennessy, Patterson)

Maksat, mikromimari (microarchitecture) tasarımı hakkında bilgi sahibi olmak

Mikromimari tasarımı, performansı ne ölçüde etkiliyor:

Compiler, bilgisayarın performans denkleminde buyruk sayısını ve CPI değerini etkiliyor.

Mikromimari ise CPI ve periyot (clock cycle time) parametreleri üzerinde etkili oluyor.

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$