BZ403 Bahar 2023 VİZE Tarih: 12.05.2023 09:30 – 11:30

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

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

2'lik tabanda verilen 101000111010101011110001 sayısı 8'lik tabanda 50725361 olarak ifade edilir.

2'lik tabanda verilen 101000111010101011110001 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

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:

31 … 0	
x0 / zero	Hardwired zero
x1 / ra	Return address
x2 / sp	Stack pointer
x3 / gp	Global pointer
x4 / tp	Thread pointer
x5 / t0	Temporary
x6 / t1	Temporary
x7 / t2	Temporary
x8 / s0 / fp	Saved register, frame pointer
x9 / s1	Saved register
x10 / a0	Function argument, return value
x11 / a1	Function argument, return value
x12 / a2	Function argument
x13 / a3	Function argument
x14 / a4	Function argument
x15 / a5	Function argument
x16 / a6	Function argument
x17 / a7	Function argument
x18 / s2	Saved register
x19 / s3	Saved register
x20 / s4	Saved register
x21 / s5	Saved register
x22 / s6	Saved register
x23 / s7	Saved register
x24 / s8	Saved register
x25 / s9	Saved register
x26 / s10	Saved register
x27 / s11	Saved register
x28 / t3	Temporary
x29 / t4	Temporary
x30 / t5	Temporary
x31 / t6	Temporary

32

RV32I

Integer Computation

- add { _ immediate }
- subtract
- { and / or / exclusive or } { _ immediate }
- { shift left logical / shift right arithmetic / shift right logical } { _ immediate }
- load upper immediate
- add upper immediate to pc
- set less than { _ immediate } { _ unsigned }

Control transfer

- branch { equal / not equal }
- branch { greater than or equal / less than } { _ unsigned }
- jump and link { _ register }

Loads and Stores

- load / store { byte / halfword / word }
- load { byte / halfword } unsigned

Miscellaneous instructions

- fence loads & stores
- fence.instruction & data
- environment { break / call }
- control status register { read & clear bit / read & set bit / read & write } { _ immediate }

31 … 25	24 … 20	19 … 15	14 … 12	11 … 7	6 … 0	
imm[31:12]				rd	0110111	U lui
imm[31:12]				rd	0010111	U auipc
imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu
imm[11:0]		rs1	000	rd	0000011	I lb
imm[11:0]		rs1	001	rd	0000011	I lh
imm[11:0]		rs1	010	rd	0000011	I lw
imm[11:0]		rs1	100	rd	0000011	I lbu
imm[11:0]		rs1	101	rd	0000011	I lhu
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	S sb
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	S sh
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	S sw
imm[11:0]		rs1	000	rd	0010011	I addi
imm[11:0]		rs1	010	rd	0010011	I slti
imm[11:0]		rs1	011	rd	0010011	I sltiu
imm[11:0]		rs1	100	rd	0010011	I xori
imm[11:0]		rs1	110	rd	0010011	I ori
imm[11:0]		rs1	111	rd	0010011	I andi

0000000	shamt	rs1	001	rd	0010011	I slli
0000000	shamt	rs1	101	rd	0010011	I srli
0100000	shamt	rs1	101	rd	0010011	I srai
0000000	rs2	rs1	000	rd	0110011	R add
0100000	rs2	rs1	000	rd	0110011	R sub
0000000	rs2	rs1	001	rd	0110011	R sll
0000000	rs2	rs1	010	rd	0110011	R slt
0000000	rs2	rs1	011	rd	0110011	R sltu
0000000	rs2	rs1	100	rd	0110011	R xor
0000000	rs2	rs1	101	rd	0110011	R srl
0100000	rs2	rs1	101	rd	0110011	R sra
0000000	rs2	rs1	110	rd	0110011	R or
0000000	rs2	rs1	111	rd	0110011	R and
0000 pred	succ	00000	000	00000	0001111	I fence
0000 0000	0000	00000	001	00000	0001111	I fence.i
000000000000		00000	000	00000	1110011	I ecall
000000000001		00000	000	00000	1110011	I ebreak
csr		rs1	001	rd	1110011	I csrrw
csr		rs1	010	rd	1110011	I csrrs
csr		rs1	011	rd	1110011	I csrrc
csr		zimm	101	rd	1110011	I csrrwi
csr		zimm	110	rd	1110011	I csrrsi
csr		zimm	111	rd	1110011	I csrrci

```
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
```

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.

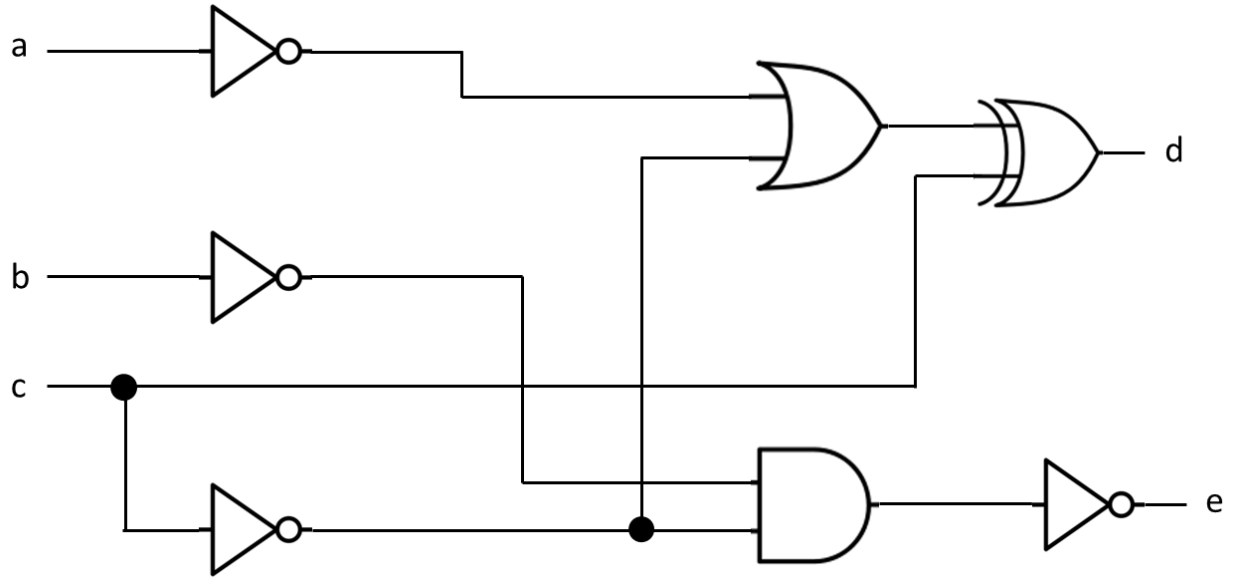
Stack adres uzayının alt tarafında dinamik verilerin saklanması için kullanılan heap alanı bulunur. Dinamik bellek gereksiniminin çok fazla olması, ya da çok fazla prosedür çağrılarak stack'e çok fazla veri yazılması nedeniyle yeni prosedür çağrılarında stack alanının dışında bir alanı kullandığında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

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

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')'

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

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

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
```

NOT: Vize çözümlerinizi pdf formatında beni (mbaykenar) collaborator olarak eklediğiniz private github repo üzerinde paylaşacaksınız. Piazza üzerinden yüklenen bu word formatındaki vize dosyası üzerinde cevaplarınızı yazıp "isim_soyisim_bz403_vize.pdf" olarak github reponuzda oluşturacağınız "vize" adındaki klasörün içine upload edeceksiniz.