Građe Međuispit 2 primjerak testa odgovori

Ishod 4

1. **Objasnite što je prekidni sustav i kako radi?**

Prekidni sustav je mehanizam koji omogućava trenutačno prekidanje izvršavanja trenutnog programa kako bi se obradio događaj visoke prioritetnosti. Kada se dogodi prekid, CPU zaustavlja trenutni zadatak, sprema trenutni kontekst (status i registri), te prebacuje izvršavanje na posebnu prekidnu rutinu. Prekidna rutina obrađuje događaj (npr. dolazne podatke s tipkovnice), a zatim CPU vraća pohranjeni kontekst i nastavlja izvršavanje prekinutog programa. Prekidni sustav omogućuje efikasnije upravljanje resursima računala i brži odgovor na događaje, što poboljšava ukupnu performansu sustava.

2. **Koje vrste pipeline hazarda poznajete?**

- *Structural hazard* – strukturni rizici zbog pristupa dijeljenim resursima
- *Data hazard* – podatkovni rizici zbog potrebe da se osigura međusobna ovisnost podataka u različitim instrukcijama.
- *Control hazard* – kontrolni rizici koje uzrokuju instrukcije koje mijenjaju PC registar, dakle grananja i skokovi u programu.

3. **Objasnite što se događa sa procesorom ako se dogodi iznimka (exception).**

Iznimka će uzrokovati da procesor izvrši sljedeće:
Spremite stanje procesora (PSTATE), npr. zastavice procesora, bitove maske prekida, razinu iznimke itd.
Spremanje povratne adrese (trenutni PC).

1. *Skok na dio za rukovanje iznimkom definiran vektorom u memoriji*
2. *Spremite registre, izvršite kod rukovatelja iznimke, vratite registre.*
3. *Povratak iz iznimke (instrukcija ERET).*

4. **Koristeći sedam segmentni display ispišite samo parne brojeve s kratkom petljom koja će omogućiti da između brojki postoji vremenski razmak.**

```
1.          .global _start
2.
3.          .equ SEG7_BASE, 0xFF200020
4.          .equ OUTER_DELAY, 0x100
5.          .equ INNER_DELAY, 0x1000
6.
7.          .data
8.  even_numbers:
9.          .word 0x3F  // 0
10.         .word 0x5B  // 2
11.         .word 0x66  // 4
12.         .word 0x7D  // 6
13.         .word 0x7F  // 8
14.
15.         .text
16. _start:
17.         LDR R0, =even_numbers
18.         LDR R1, =SEG7_BASE
19.
20. loop:
21.         LDR R2, [R0]
22.         STR R2, [R1]
23.         MOV R4, #OUTER_DELAY
24. outer_delay_loop:
25.         MOV R3, #INNER_DELAY
26. inner_delay_loop:
27.         SUBS R3, R3, #1
```

```
28.         BNE inner_delay_loop
29.         SUBS R4, R4, #1
30.         BNE outer_delay_loop
31.
32.         ADD R0, R0, #4
33.
34.         LDR R5, =even_numbers + 20
35.         CMP R0, R5
36.         BNE loop
37.
38.         LDR R0, =even_numbers
39.         B loop
40.
```

Ishod 5

1. **Je li slijedeća izjava točna: „Procesori imaju ogromne količine brze cache memorije dok grafičke kartice imaju malo cache memorije. Procesori imaju vrlo brz pristup ogromnim količinama radne memorije (RAM), dok grafičke kartice imaju spor pristup grafičkoj memoriji.” Objasnite svoj odgovor.**

   Slijedeća izjava nije točna, zato što procesori imaju malu količinu brze cache memorije i imaju brzi pristup radnoj memoriji dok grafičke kartice imaju veliku količinu radne memorije sa brzim pristupom, ali imaju manje cache memorije koje kompenziraju sa VRAM-om zbog sporijeg pristupa RAM-u.

2. **Je li slijedeća izjava točna: I kod SMP-a i kod NUMA-e postoji problem latencije memorije, koja je značajno sporija od cache memorije. Ako se pravilno dizajnira, sustav baziran na NUMA arhitekturi će u ogromnoj količini slučajeva ostvarivati bolje performanse i manje probleme sa kvalitetom usluge (latencije pristupa memoriji, brzine pristupa memoriji). Objasnite svoj odgovor.**

   Izjava je djelomično točna, zato što ako je aplikacija dizajnirana da je osviještena o SMP/NUMA arhitekturi i topologiji sustava, te može pravilno raspoređivati procese i podatke, tada u lokalnoj NUMA zoni gdje se aplikacija nalazi, može doći do boljih performansa i manje latencije pri radu.

3. **Kod heterogenih i asimetričnih arhitektura (primjer: CPU + GPU), koje vrste jezgri postoje i za kakve poslove bismo ih mogli idealno iskoristiti (1 bod)?**
   A. male CPU jezgre (akceleracija serijskih dijelova koda) i velike CPU jezgre (akceleracija paralelnih dijelova koda), uz kvalitetnu predikciju koji kod spada u koju grupu (serijski ili paralelni). Također, GPU-ove za masovno paralelne izračune.
   B. male CPU jezgre (akceleracija paralelnih dijelova koda) i velike CPU jezgre (akceleracija serijskih dijelova koda), uz kvalitetnu predikciju koji kod spada u koju grupu (serijski ili paralelni). Također, GPU-ove za masovno paralelne izračune.
   C. možemo sve izvršiti na paralelan način pošto je sav izvorni kod tretiran kroz kompajler i organiziran pri kompajliranju tako da se može izvoditi paralelno
   D. najbolje bi bilo napraviti optimalan heterogeni OpenCL kod korištenjem heterogene platforme (CPU+GPU) jer ćemo tako uvijek dobiti najbolje performanse.

   B je točan odgovor, dok je D djelomično točan.

   Zato što u B odgovoru:

A. koriste se male jezgre za paralelne zadatke koje mogu biti energetski učinkovite, manje prostora zauzimaju, može ih se više staviti na čip i mogu izvršavati više jednostavnih paralelnih zadataka istovremeno.
B. Velike jezgre se koriste za serijske zadatke zbog velike količine cache memorije, složenijih instrukcija, out of order execution-a, spekulativnog izvršavanja i ostalih značajka koje male jezgre nemaju, a potrebne su za bolje performance izvrsavanja serijskog dijela koda
C. Grafčke kartice su vrlo efikasne u izvršavanju masovnih paralelnih zadataka zbog velikog broja mikroprocesora/jezgri optimiziranih za paralelne operacije

Odgovor D je djelomično točan je nudi potencijalne visoke performanse korištenjem heterogenog API-a/platforme OpenCL na heterogenom hardveru (GPU+CPU), ali treba uzeti u obzir visoku složenost implementacije i potrebe za optimizacijom

4. **Koje su prednosti i mane privatnog vs dijeljenog pristupa priručnoj (cache) memoriji kod višejezgrenih procesora?**
   Kod privatnog pristupa cache memoriji prednosti su:
   - *Smanjena latencija*
   - *Izolacija performansi*
   - *Jednostavnije upravljanje*

   Mane su:

   - *Smanjena efikasnost korištenja memorije*
   - *Potreba za koherencijskim protokolima*

   Kod dijeljenog pristupa cache memoriji prednosti su:

   - *Bolja iskoristivost memorije*
   - *Jednostavnije upravljanje podacima*
   - *Poboljšana koherencija*

   Mane su:

   - *Povećana latencija*
   - *Konflikti pristupa*
   - *Kompleksnost dizajna*
5. **Iz aspekta paralelizma, zašto nam je bitno da paralelno procesiranje ima kvalitetno dizajniran memorijski sustav?**

   Kvalitetno dizajniran memorijski sustav omogućava efikasan i brz pristup podacima, smanjuje latenciju i povećava propusnost koje su neke od karakteristika potrebne za postizanje visokih performansi u sustavima sa paralelnim procesiranjem što omogućuje maksimalnim i najefikasnijim iskorištavanjem jezgri i procesora za izvršavanje zadataka.

6. **Koristeći Amdahl-ov zakon izračunajte koliko je ubrzanje u postocima ako su vrijednosti za petlje u primjeru u vježbama redom 2048, 1024, 5120 i 4096, a maksimalno ubrzanje koje možete postići za petlju je 3 puta.**
   Uvijek ubrzavamo *najduži dio koda*

$$\boldsymbol{S_{max} = \frac{1}{(1-P) + \frac{P}{S}}}$$

$$\mathbf{2048 + 1024 + 5120 + 4096 = \ 12288}$$

A $2048 - \frac{2}{12}$

B $1024 - \frac{1}{12}$

C $5120 - \frac{5}{12} \approx \mathbf{0,41} \ = \mathbf{P}, \ \ \mathbf{S = 3}$ (u zadatku piše)

D $4096 - \frac{4}{12}$

$$S_{\max} = \frac{1}{(1-0{,}41) + \frac{0{,}41}{3}} = \ 1.37614 = \mathbf{38\% - \ maksimalno\ ubrzanje}$$

Ishod 6

1. **Da li SIMD instrukcije procesora rade različitom brzinom na SMP ili NUMA arhitekturi (1 bod). Zašto? (2 boda)**

   SIMD instrukcije mogu raditi različitom brzinom na SMP i NUMA arhitekturi zbog razlika u latenciji pristupa memoriji što može utjecati na ukupnu brzinu izvršavanja SIMD instrukcija, posebno ako nije pažljivo upravljano alokacijom podataka.

2. **Koja je osnovna razlika između SMP i NUMA modela organizacije sustava sa jednim ili više procesorskih utora i jednom ili više procesorskih jezgri?**
   Osnovna razlika između SMP i NUMA modela organizacije sustava leži u načinu pristupa memoriji. U SMP sustavima, svi procesori imaju simetričan pristup svim dijelovima memorije s jednakom latencijom, dok u NUMA sustavima, procesori imaju različite latencije pristupa ovisno o tome je li memorija lokalna ili udaljena.
3. **Koja je cijena koju kod NUMA arhitekture moramo „platiti“ ukoliko koristimo remote memoriju (memoriju drugog fizičkog procesora)?**
   Ukoliko koristimo remote memoriju tada dolazi do povećane latencije, smanjene propusnosti interconnect sabirnice, povećane kompleksnosti upravljanja memorijom i regresije na performansama aplikacija.
4. **Kako bi u NUMA arhitekturi izgledala optimalna memorijska hijerarhija, ako na izboru imate sve komponente koje na tržištu možete pronaći?**

   Optimalna memorijska hijerarhija u NUMA arhitekturi bi izgledala ovako:
   a. Registri
   b. L1 Cache
   c. L2 Cache
   d. L3 Cache
   e. RAM i NVDIMM
      i. Lokalna memorija NUMA node-a
      ii. NVDIMM memorija
      iii. Udaljena memorija na drugom NUMA node-u
   f. NVMe SSD-ovi
   g. Magnetski diskovi (HDD)

5. **Na raspolaganju su vam dvije serverske SMP arhitekture sa dva i četiri socketa Aplikacija koju trebate instalirati na jedan od ta dva servera je jako osjetljiva na brzinu rada i latenciju memorije. Uz uvjet da obje arhitekture imaju dovoljno procesorske snage za Preddiplomski studij 2/2 izvršavanje aplikacije čak i na jednom socketu, koja arhitektura bi bila optimalniji izbor za izvršavanje aplikacije (1 bod)? Zašto (2 boda)?**
   Arhitektura sa dva socketa bi bila optimalnija jer je, pod zadanim uvjetima, lakša za napraviti i održavati. Pošto se nalazimo u SMP arhitekturi to znači da svi procesori moraju imati jednako vrijeme pristupa memoriji te je puno lakše to napraviti sa dva procesora nego sa četiri jer to vrijeme pristupa mora zadovoljavati potrebe svih procesora.

GRUPE ZADATAKA

Ishod 4

Koristeći sedam segmentni display izbrojite do od 1-5 sa kratkom petljom koja će omogućiti da između brojki postoji vremenski razmak.

```
1. .data
2. segments:   .word 0x06, 0x5B, 0x4F, 0x66, 0x6D    @ Segment values for 1, 2, 3, 4, 5
3.
4.     .text
5.     .global _start
6.
7. _start:
8.     LDR     R1, =0xFF200020     @ Base address for GPIO
9.     LDR     R2, =segments       @ Address of segment values
10.    MOV     R4, #0              @ Initialize counter to 0
11. loop:
12.    LDR     R3, [R2, R4, LSL #2] @ Load segment value based on counter
13.    STR     R3, [R1]            @ Write segment value to GPIO port
14.    BL      delay               @ Call delay function
15.    ADD     R4, R4, #1          @ Increment counter
16.    CMP     R4, #5              @ Compare counter to 5
17.    BNE     loop                @ If not equal, loop again
18. done:
19.    B       done                @ Infinite loop to end program
20. delay:
21.    MOV     R5, #1000           @ Outer loop count
22. outer_delay_loop:
23.    MOV     R6, #1000           @ Inner loop count
24. inner_delay_loop:
25.    SUBS    R6, R6, #1          @ Subtract 1 from R6
26.    BNE     inner_delay_loop    @ If R6 is not zero, branch to inner_delay_loop
27.    SUBS    R5, R5, #1          @ Subtract 1 from R5
28.    BNE     outer_delay_loop    @ If R5 is not zero, branch to outer_delay_loop
29.    BX      LR                  @ Return from delay
30.
```

Koristeći sedam segmentni display ispišite samo neparne brojeve sa kratkom petljom koja će omogućiti da između brojki postoji vremenski razmak.

```
 1. .data
 2. segments:   .word 0x06, 0x4F, 0x6D   @ Segment values for 1, 3, 5
 3.
 4. .text
 5. .global _start
 6.
 7. _start:
 8.     LDR     R1, =0xFF200020     @ Base address for GPIO (assuming the base address for the
seven segment display)
 9.     LDR     R2, =segments       @ Address of segment values
10.     MOV     R4, #0              @ Initialize counter to 0
11.
12. loop:
13.     LDR     R3, [R2, R4, LSL #2] @ Load segment value based on counter
14.     STR     R3, [R1]            @ Write segment value to GPIO port
15.     BL      delay               @ Call delay function
16.
17.     ADD     R4, R4, #1          @ Increment counter
18.     CMP     R4, #3              @ Compare counter to 3 (number of odd numbers)
19.     BNE     loop                @ If not equal, loop again
20.
21.     MOV     R4, #0              @ Reset counter to 0
22.     B       loop                @ Repeat the loop
23.
24. done:
25.     B       done                @ Infinite loop to end program
26.
27. delay:
28.     MOV     R5, #1000           @ Outer loop count
29. outer_delay_loop:
30.     MOV     R6, #1000           @ Inner loop count
31. inner_delay_loop:
32.     SUBS    R6, R6, #1          @ Subtract 1 from R6
33.     BNE     inner_delay_loop    @ If R6 is not zero, branch to inner_delay_loop
34.     SUBS    R5, R5, #1          @ Subtract 1 from R5
35.     BNE     outer_delay_loop    @ If R5 is not zero, branch to outer_delay_loop
36.     BX      LR                  @ Return from delay
37.
```

Koristeći sedam segmentni display ispišite samo parne brojeve sa kratkom petljom koja će omogućiti da između brojki postoji vremenski razmak.

```
 1. .data
 2. segments:   .word 0x5B, 0x66    @ Segment values for 2 and 4
 3.
 4.     .text
 5.     .global _start
 6.
 7. _start:
 8.     LDR     R1, =0xFF200020     @ Base address for GPIO (assuming the base address for the
seven segment display)
 9.     LDR     R2, =segments       @ Address of segment values
10.     MOV     R4, #0              @ Initialize counter to 0
11.
12. loop:
13.     LDR     R3, [R2, R4, LSL #2] @ Load segment value based on counter
14.     STR     R3, [R1]            @ Write segment value to GPIO port
15.     BL      delay               @ Call delay function
16.
17.     ADD     R4, R4, #1          @ Increment counter
18.     CMP     R4, #2              @ Compare counter to 2 (number of even numbers)
19.     BNE     loop                @ If not equal, loop again
20.
21.     MOV     R4, #0              @ Reset counter to 0
22.     B       loop                @ Repeat the loop
23.
24. done:
25.     B       done                @ Infinite loop to end program
26.
27. delay:
28.     MOV     R5, #1000           @ Outer loop count
29. outer_delay_loop:
30.     MOV     R6, #1000           @ Inner loop count
31. inner_delay_loop:
32.     SUBS    R6, R6, #1          @ Subtract 1 from R6
33.     BNE     inner_delay_loop    @ If R6 is not zero, branch to inner_delay_loop
34.     SUBS    R5, R5, #1          @ Subtract 1 from R5
35.     BNE     outer_delay_loop    @ If R5 is not zero, branch to outer_delay_loop
36.     BX      LR                  @ Return from delay
37.
```

Koristeći sedam segmentni display izbrojite od 1-9 ali na drugom mjestu zdesna uz dodanu kratku petlju koja će omogućiti da između brojki postoji vremenski razmak. (hint: LSL)

```
.data
segments:   .word 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67   @ Segment values for 0-9

    .text
    .global _start

_start:
    LDR     R1, =0xFF200020     @ Base address for GPIO (assuming the base address for the seven segment display)
    LDR     R2, =segments       @ Address of segment values
    MOV     R4, #1              @ Initialize counter to 1

loop:
    LDR     R3, [R2, R4, LSL #2] @ Load segment value based on counter
    LSL     R3, R3, #8          @ Shift segment value to the second position from the right
    STR     R3, [R1]            @ Write shifted segment value to GPIO port
    BL      delay               @ Call delay function

    ADD     R4, R4, #1          @ Increment counter
    CMP     R4, #10             @ Compare counter to 10
    BNE     loop                @ If not equal, loop again

    MOV     R4, #1              @ Reset counter to 1
    B       loop                @ Repeat the loop

done:
    B       done                @ Infinite loop to end program

delay:
    MOV     R5, #1000           @ Outer loop count
outer_delay_loop:
    MOV     R6, #1000           @ Inner loop count
inner_delay_loop:
    SUBS    R6, R6, #1          @ Subtract 1 from R6
    BNE     inner_delay_loop    @ If R6 is not zero, branch to inner_delay_loop
    SUBS    R5, R5, #1          @ Subtract 1 from R5
    BNE     outer_delay_loop    @ If R5 is not zero, branch to outer_delay_loop
    BX      LR                  @ Return from delay

```

Koristeći sedam segmentni display izbrojite od 9-1 ali na drugom mjestu zdesna uz dodanu kratku petlju koja će omogućiti da između brojki postoji vremenski razmak. (hint: LSL)

```
 1.     .data
 2. segments:   .word 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67   @ Segment
values for 0-9
 3.
 4.     .text
 5.     .global _start
 6.
 7. _start:
 8.     LDR     R1, =0xFF200020     @ Base address for GPIO (assuming the base address for the
seven segment display)
 9.     LDR     R2, =segments       @ Address of segment values
10.     MOV     R4, #9              @ Initialize counter to 9
11.
12. loop:
13.     LDR     R3, [R2, R4, LSL #2] @ Load segment value based on counter
14.     LSL     R3, R3, #8          @ Shift segment value to the second position from the right
15.     STR     R3, [R1]            @ Write shifted segment value to GPIO port
16.     BL      delay               @ Call delay function
17.
18.     SUB     R4, R4, #1          @ Decrement counter
19.     CMP     R4, #0              @ Compare counter to 0
20.     BNE     loop                @ If not equal, loop again
21.
22.     MOV     R4, #9              @ Reset counter to 9
23.     B       loop                @ Repeat the loop
24.
25. done:
26.     B       done                @ Infinite loop to end program
27.
28. delay:
29.     MOV     R5, #1000           @ Outer loop count
30. outer_delay_loop:
31.     MOV     R6, #1000           @ Inner loop count
32. inner_delay_loop:
33.     SUBS    R6, R6, #1          @ Subtract 1 from R6
34.     BNE     inner_delay_loop    @ If R6 is not zero, branch to inner_delay_loop
35.     SUBS    R5, R5, #1          @ Subtract 1 from R5
36.     BNE     outer_delay_loop    @ If R5 is not zero, branch to outer_delay_loop
37.     BX      LR                  @ Return from delay
38.
```

Koristeći Amdahlov zakon izračunajte koliko je ubrzanje u postocima ako su vrijednosti za petlje u primjeru u vježbama redom 2048, 1024, 5120 i 4096, a maksimalno ubrzanje koje možete postići za petlju je 4 puta.

$$\boldsymbol{S_{max} = \frac{1}{(1 - P) + \frac{P}{S}}}$$

$$\mathbf{2048 + 1024 + 5120 + 4096 = \ 12288}$$

$$\text{A } 2048 - \frac{2}{12}$$

$$\text{B } 1024 - \frac{1}{12}$$

$$\text{C } 5120 - \frac{5}{12} \approx \mathbf{0,41} \ = \mathbf{P}, \ \ \mathbf{S = 4} \text{ (u zadatku piše)}$$

$$\text{D } 4096 - \frac{4}{12}$$

$$\text{S}_{\text{max}} = \frac{1}{(1 - 0{,}41) + \frac{0{,}41}{4}} = \ 1.44404 = \mathbf{44}\% - \ \textbf{maksimalno ubrzanje}$$

Koristeći Amdahlow zakon izračunajte koliko je ubrzanje u postocima ako su vrijednosti za petlje u primjeru u vježbama redom 2048, 1024, 5120 i 4096, a maksimalno ubrzanje koje možete postići za petlju je 6 puta.

$$\boldsymbol{S_{max} = \frac{1}{(1 - P) + \frac{P}{S}}}$$

$$\mathbf{2048 + 1024 + 5120 + 4096 = \ 12288}$$

$$\text{A } 2048 - \frac{2}{12}$$

$$\text{B } 1024 - \frac{1}{12}$$

$$\text{C } 5120 - \frac{5}{12} \approx \mathbf{0,41} \ = \mathbf{P}, \ \ \mathbf{S = 6} \text{ (u zadatku piše)}$$

$$\text{D } 4096 - \frac{4}{12}$$

$$\text{S}_{\text{max}} = \frac{1}{(1 - 0{,}41) + \frac{0{,}41}{6}} = \ 1.51898 = \mathbf{52}\% - \ \textbf{maksimalno ubrzanje}$$

Koristeći Amdahlov zakon izračunajte koliko je ubrzanje u postocima ako su vrijednosti za petlje u primjeru u vježbama redom 2048, 1024, 5120 i 4096, a maksimalno ubrzanje koje možete postići za petlju je 3 puta.

$$\boldsymbol{S_{max} = \frac{1}{(1-P) + \frac{P}{S}}}$$

$$\mathbf{2048 + 1024 + 5120 + 4096 = \ 12288}$$

A $2048 - \frac{2}{12}$

B $1024 - \frac{1}{12}$

C $5120 - \frac{5}{12} \approx \mathbf{0,41} \ = \mathbf{P}, \ \ \mathbf{S = 3}$ (u zadatku piše)

D $4096 - \frac{4}{12}$

$$\mathrm{S_{max}} = \frac{1}{(1-0{,}41) + \frac{0{,}41}{3}} = \ 1.37614 = \mathbf{37}\% - \ \textbf{maksimalno ubrzanje}$$

Koristeći Amdahlov zakon izračunajte koliko je ubrzanje u postocima ako su vrijednosti za petlje u primjeru u vježbama redom 2048, 1024, 5120 i 4096, a maksimalno ubrzanje koje možete postići za petlju je 2 puta.

$$\boldsymbol{S_{max} = \frac{1}{(1-P) + \frac{P}{S}}}$$

$$\mathbf{2048 + 1024 + 5120 + 4096 = \ 12288}$$

A $2048 - \frac{2}{12}$

B $1024 - \frac{1}{12}$

C $5120 - \frac{5}{12} \approx \mathbf{0,41} \ = \mathbf{P}, \ \ \mathbf{S = 2}$ (u zadatku piše)

D $4096 - \frac{4}{12}$

$$\mathrm{S_{max}} = \frac{1}{(1-0{,}41) + \frac{0{,}41}{2}} = \ 1.25786 = \mathbf{26}\% - \ \textbf{maksimalno ubrzanje}$$

Koristeći Amdahlow zakon izračunajte koliko je ubrzanje u postocima ako su vrijednosti za petlje u primjeru u vježbama redom 2048, 1024, 5120 i 4096, a maksimalno ubrzanje koje možete postići za petlju je 8 puta.

$$\boldsymbol{S_{max} = \frac{1}{(1-P) + \frac{P}{S}}}$$

$$\mathbf{2048 + 1024 + 5120 + 4096 = \ 12288}$$

$$\text{A } 2048 - \frac{2}{12}$$

$$\text{B } 1024 - \frac{1}{12}$$

$$\text{C } 5120 - \frac{5}{12} \approx \mathbf{0,41} \ = \mathbf{P}, \ \ \mathbf{S = 8} \text{ (u zadatku piše)}$$

$$\text{D } 4096 - \frac{4}{12}$$

$$\text{S}_{\text{max}} = \frac{1}{(1-0{,}41) + \frac{0{,}41}{8}} = \ 1.55945 = \mathbf{56\% - \ maksimalno\ ubrzanje}$$

Koristeći Amdahlow zakon izračunajte koliko minimalno moramo ubrzati petlju, i u kojoj petlji to treba napraviti ako su vrijednosti za petlje u primjeru u vježbama redom 2048, 1024, 5120 i 4096, a želimo da ukupno ubrzanje bude veće od 50%.

Ubrzanje treće petlje

Ako ubrzamo treću petlju:

$$P = \frac{5120}{12288} \approx 0.4167$$

Koristeći Amdahlov zakon:

$$1.5 < \frac{1}{(1-0.4167) + \frac{0.4167}{A}}$$

$$1.5 < \frac{1}{0.5833 + \frac{0.4167}{A}}$$

$$\frac{2}{3} > 0.5833 + \frac{0.4167}{A}$$

$$\frac{2}{3} - 0.5833 > \frac{0.4167}{A}$$

$$0.0833 > \frac{0.4167}{A}$$

$$A > \frac{0.4167}{0.0833}$$

$$A > 5$$

Koristeći Amdahlow zakon izračunajte koliko minimalno moramo ubrzati petlju, i u kojoj petlji to treba napraviti ako su vrijednosti za petlje u primjeru u vježbama redom 2048, 1024, 5120 i 4096, a želimo da ukupno ubrzanje bude veće od 40%.