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.

1. **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.
1. **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.

1. **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.

1. **Kod heterogenih i asimetričnih arhitektura (primjer: CPU + GPU), koje vrste jezgri postoje i za kakve poslove bismo ih mogli idealno iskoristiti (1 bod)?**
2. 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.
3. 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.
4. možemo sve izvršiti na paralelan način pošto je sav izvorni kod tretiran kroz kompajler i organiziran pri kompajliranju tako da se može izvoditi paralelno
5. 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:

1. 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.