Arhitektura računara 2021/22 – Teme za prvi projektni rad

Napomena za izradu projektnih zadataka

Potrebno je uraditi samo jedan od ponuđenih projektnih zadataka. Projektni rad se mora moći kompajlirati, izvršiti i testirati. Uz priloženi rad treba da se obezbijede i testni primjeri.

Zadatak 1.1 – Asemblerski program za obradu podataka

(16) Napisati asemblerski program za obradu podataka. Algoritam odabrati na stranici kursa. Omogućiti pokretanje iz konzole nad specificiranim ulaznim fajlom.

(5) Optimizovati program uvođenjem SSE ili AVX paralelizma i dokumentovati ubrzanje.

(4) Napisati isti program u C ili C++ programskom jeziku i isprobati različite nivoe kompajlerskih optimizacija. Uporediti performanse sa prethodnim implementacijama i dokumentovati rezultate.

Zadatak 1.2 – Kernel operativnog sistema

Realizovati kernel operativnog sistema koji koristi proizvoljan *bootloader* (npr. GRUB). Implementirati:

(10) Izračunavanje i prikaz sume dva cijela broja koja korisnik unosi putem tastature

(5) Prikaz sistemskog vremena ili prikaz neke informacije o hardveru računara

(5) Promjenu boje teksta pri svakom otkucaju sistemskog tajmera

(5) Prelazak u 64-bitni režim rada**,** pri čemu straničenje treba biti podešeno tako da u memoriji istovremeno bude prisutna bar jedna 1GB stranica, bar jedna 2MB stranica i bar jedna 4KB stranica

Zadatak 1.3 – Simulator petofazne protočne obrade

(15) Realizovati (u proizvoljnom jeziku) simulator petofazne protočne obrade (IF, ID, EX, MEM, WB). Podržati sljedeće instrukcije: ADD, SUB, MUL, DIV, LOAD i STORE (sa njihovim standardnim značenjem). ADD, SUB, LOAD i STORE zahtijevaju 1 ciklus za izvršavanje (EX faza). MUL i DIV zahtijevaju 2 ciklusa za izvršavanje (EX faza). LOAD i STORE zahtijevaju 3 ciklusa za pristup memoriji. Smatrati da se IF, ID i WB faze uvijek izvrše u jednom ciklusu. Svaka instrukcija raspolaže sa maksimalno tri registarska operanda (odredišni registar i izvorišni registar kod unarnih operacija, odnosno odredišni registar i dva izvorišna registra kod binarnih operacija). Kao ulaz se prosljeđuje niz instrukcija. U tabelarnoj formi prikazati izlaz koji za svaki ciklus (kolonu) prikazuje faze (ili zastoje) u kojima se nalaze ulazne instrukcije (vrste).

(5) Omogućiti prosljeđivanje podataka između protočnih stepeni.

(5) Ispisati potencijalne hazarde (WAR, RAW i WAW zavisnosti) između instrukcija u ulaznom nizu.

Zadatak 1.4 – Simulator arhitekture instrukcionog skupa

(10) U proizvoljnom programskom jeziku napisati simulator sopstvene arhitekture instrukcionog skupa sa bar 4 registra opšte namjene dužine 64 bita (dozvoljeno je koristiti tip podataka dužine 8 bajta, npr. *long long* ili *uint64_t* za registre). Simulator treba da funkcioniše kao interpreter asemblerskih instrukcija. Treba biti omogućeno da se izvorni asemblerski kod učita iz fajla. Pravilno izvršiti potrebnu sintaksičku i semantičku analizu koda u svrhu detekcije nevalidnih dijelova koda. Instrukcijski skup simulirane mašine (gosta) mora da obuhvata:

- osnovne aritmetičke operacije (ADD, SUB)
- osnovne bitske logičke operacije (*AND*, *OR*, *NOT*)
- instrukciju za pomjeranje podataka između registara (*MOV*)
- instrukciju za unos podataka sa standardnog ulaza (slično odgovarajućem sistemskom pozivu)
- instrukciju za ispis podataka na standardni izlaz (slično odgovarajućem sistemskom pozivu)

(5) Implementirati rad sa memorijom. Simulirana mašina (gost) treba da ima 64-bitni adresni prostor. Omogućiti direktno i indirektno adresiranje. Sadržaj svake memorijske adrese treba biti dužine 1 bajt. Pravilno omogućiti da se adrese mogu navesti kao brojevi. Omogućiti pristup svim adresama iz adresnog prostora, uključujući i upis i čitanje, upotrebom odgovarajućih instrukcija (*MOV* ili *LOAD/STORE*).

(5) Implementirati instrukcije potrebne za bezuslovno i uslovno grananje (JMP, CMP, JE, JNE, JGE, JL).

(2) Implementirati jednostavnu *single-step debugging* podršku. Omogućiti izvršavanje i pregled vrijednosti svih registara i specificiranih memorijskih adresa na postavljenim zaustavnim tačkama u asemblerskom kodu tokom izvršavanja. U ovom režimu treba biti omogućen prelaz na sljedeću instrukciju (NEXT ili STEP konzolne komande) i prelaz na sljedeću zaustavnu tačku (CONTINUE).

(3) Omogućiti da se asemblerske instrukcije mogu prevesti u mašinski kod gosta i smjestiti na proizvoljnu lokaciju u adresni prostor gosta (analogno *code* ili *text* segmentu). Definisati i iskoristiti registar koji će da služi kao programski brojač (instrukcijski pokazivač). Omogućiti da mašina radi u režimu u kom se izvršavaju mašinske instrukcije locirane u memoriji gosta. Prelazak u režim u kom se izvršavaju instrukcije iz memorije se može izvršiti upotrebom namjenske instrukcije. Konstruisati primjer asemblerskog koda za gosta koji je samomodifikujući (nakon prevođenja u mašinski kod gosta) i u kom se modifikovani dio koda izvršava i prije i poslije relevantne modifikacije.

Obezbijediti nekoliko jediničnih testova kojima se demonstriraju funkcionalnosti iz stavki u specifikaciji projektnog zadatka.

Sve detalje koji nisu eksplicitno navedeni implementirati na proizvoljan način. Pridržavati se:

- principa objektno-orijentisanog programiranja i SOLID principa
- principa pisanja čistog koda i pravilnog imenovanja varijabli, funkcija, klasa i metoda
- konvencija za korišteni programski jezik