

## PITANJA ZA SVA TRI BLICA

Širina linije priručne memorije L1 obično se kreće oko: **32B**

Najznačajnijem bajtu 32-bitnog broja na adresi p na računalu s arhitekturom Sun SPARC možemo prispojiti sa : **\*((char\*)p)**

U kojoj datoteci se nalazi prototip funkcije clock? **time.h**

Treba zbrojiti dvije velike matrice čiji su elementi prirodni brojevi manji od 1000. Kojim tipom elemenata matrice bismo postigli ispravan rezultat uz minimalni broj promašaja PM?: **short**

Pristupanje elementima matrice **p(i\*m+j)**

Odnos povratne vrijednosti funkcije clock je: **CLOCKS-PER-SEC:1**

Broj linija priručne memorije L1 obično je: **500**

Petlju u kojoj varijabla i poprima vrijednosti 0,1,2,3...n-1 možemo izvesti s: **For(i=0;i<n;++i)**

Najznačajniji bajt 32b podatka x86: **\*(char\* p) +3**

Veličina L2? : **1MB**

Pogotci/promašaji (32b procesor, 32B linija, svaki 4. povećaj, podatak 16 bitni) **1:1**

Pogotci/promašaji (64b procesor, 32B linija, podatak 64 bitni) **3:1**

Pogotci/promašaji (32b procesor, 32B linija, svaki 4. povećaj, podatak 32 bitni) **1:1**

Pogotci/promašaji (64b procesor, 32B linija, svaki povećaj) **3:1**

Ako imamo dvije velike matrice s brojevima manjim od 1000 i zbrajamo ih, koji od navedenih tipova podataka je najpogodniji **SHORT**

Uobičajena velicina linije L1 - **512bitova = 64bajta**

Kod konvencije pozivanja cdecl na x86, na početku potprograma cesta je instrukcija sub esp X. X se postavlja u ovisnosti : **Broju bajtova lokalnih varijabli potprograma**

Tvrdnja koja je istinita za SSE podskup instrukcija: **sse instrukcije se koriste isključivo za operacije nad float podacima**

Potprogram koristi 1 lokalnu char varijablu. Za koliko je minimalno potrebno pomaknuti esp u prologu potprograma: **8**

Kako bi se omogućili pozivi s varijabilnim brojem argumenata. Konvencija pozivanja cdecl podrazumijeva da stogovni okvir čisti: **pozvani kod neposredno nakon pozivanja**

Kod x86, a\*b+c, a b c na stogu : **mov eax,[ebp+8]; imul eax,[ebp+12]; add eax,[ebp+16]**

Funkcija čiji je prototip int fja(float a) u x86 strojnom jeziku pozivamo: **push a, call fja**

U programima za x86 memorijske lokalne varijable se smjestaju: **na stog, iznad argumenata potprograma (manje adrese)**

Unutar funkcije neposredno nakon cdecl prologa izvodimo DWORD PTR [ebp+4], 0. Tvrdnja: **funkcija se neće moći vratiti u glavni program**

Kojom instr u akumulator smjestimo cjelobrojni argument potprograma: **mov eax,[ebp+12]**

Kod konvencije pozivanja cdecl na x86 potprogram tipično počinje instrukcijama: **push ebp, mov ebp, esp**

U nekom podatkovnom registru D3 nalazi se FFFFFFFF. naredba MOVE #0, D3 : **FFFF0000**

Na adresi \$3000 nalazi se podatak ABCDEFGH. A u heks. formi iznosi 41. U registru A0 nalazi se adresa 3000. Što će se dogoditi ako ADD #2,A0; MOVE.W (A0), D; **A0=00003002, D0=00004344**

Kazalo stoga na MC 68000 je: **registar A7**

Registar stanja na MC68000: **SR - 16 bitni registar - korisnicki i sistemski**

Kako moraju biti pisane naredbe u EASY 68K: **moraju biti uvucene za razmak ili tabulator**

Koliko naredbi je potrebno da bi se zbrojila 3 broja na stogu: **3**

Koji je standardni način pristupanja varijablama i parametrima: **ebp**

Kako staviti 1234 u nešto: **mov eax, 1234**

Kako završava potprogram: **pop ebp; ret**

Argumenti potprograma se prenose preko: **stoga**

Ako hoćete skočiti u 0 red 1 stupac koja je naredba: **JZR1**

Ako hoćete da zastavice ostavite nepromijenjene: **HCZ**

Izvršava se mikroinstrukcija s prvog stupca u prvom retku; u onom polju za slijedeću instrukciju piše JCC 4, koja se mikroinstrukcija slijedeća izvršava: **1. stupac, 4. redak**

Ako se trenutno izvodi mikronaredba koja se u mikromemoriji nalazi na x retku i y stupcu, a u polju AC ima JZR 5, gdje se nalazi sljedeća mikroinstrukcija koja će se izvoditi: **0. redak, 5. stupac**

Makroinstrukciju ASR je moguće izvesti u : **U više od dvije mikroinstrukcije**

Kojim registrima raspolaže CPE: **R0-R9, AC, T**

JPZ # u makromemoriji zauzima: **2 bajta (2B)**

U PCMP simulatoru, povratak na prethodni prozor ide preko: **ESC**

Što radi funkcija FF1: **Stavlja jedinicu na ulaz CI (zastavicu carry, zero ili tako nešto)**

Upis u makromemoriju je moguć izravno preko kojih registara: **Preko registara AC i T.**

Procesna jedinica sadrži: **nema ni PC ni SP već da se oni realiziraju korištenjem drugih registara**

Gdje se u prozoru PCMP nalazi dio za detaljno prikazivanje mikroinstrukcije: **U gornjem dijelu.**

Koja naredba može promijeniti zastavicu Z/C/Z i C: **SCZ, STZ i STC**

Koliko se puta u JPZ # koristi uvjetni skok: **Samo jednom (nisam siguran).**

**Koliko memorije zauzima neka makronaredba:** Sve makronaredbe koje nemaju konstante (AND A,B, SUB A,B...) zauzimaju po jedan bajt (**1B**) u makromemoriji jer se zapravo gleda samo operacijski kod, a naredbe tipa JPZ # koje imaju neku konstantu zauzimaju dva bajta (**2B**) u makromemoriji.

Kako se može ostvariti logički pomak u lijevo za jedan: **Zbrajanjem broja sa samim sobom.**

Kako se može izvesti posmak u lijevo: **Aritmetičkim posmakom te nakon toga nesto s carry**

Koliko bajtova u **mikromemoriji** zauzima naredba PUSH: **n\*32 bita.**

Dana je naredba  $R1 - 1 + CI \rightarrow R2$ . Da bi se omogućila smanjivanje R2, koje zastavice trebaju bit postavljene: **ako se treba smanjit onda bi trebao bit FF0**

Što se treba napraviti da se sadržaj iz R1 prenese u R2:  **$R1 - 1 + CI \rightarrow R2$  (FF1)**

Koja od ovih operacije ne koristi stog: **ASR**

Makronaredbe se izvode: **sljedno po redcima**