3. a)

dvostruko asocijativna memorija -> a = 2, z je zrnatost – ako nije posebno navedeno uzima se da je ona 1, tj z = 1

Kapacitet -> s = 512 B, veličina linije -> b = 16 B

Slijed 32-bitnih mem. referenci: 0x0001 0x0086 0x00d4 0x0001 0x0187 0x00d5 0x01d2 0x0281 0x0002 0x008c 0x0189 0x00dd

Prosječno vrijeme pristupa memoriji? Vrijeme pristupa priručnoj memoriji 1T, pristup RAM 100 T.

Imamo jednadžbu s = n * b, odakle dobijemo n(broj linija) -> n = 512/16 = 32

Znači imamo, a = 2, s = 512B, b = 16B, n = 32, z = 1.

E sada, zanima nas struktura adrese da možemo iz memorijskih referenci odrediti što nam je što, struktura je w(adrese) = w(o)+w(i)+w(p) gdje su **o** oznaka, **i** indeks, **p** pomak.

Zatim imamo formule w(p) = log_2(b/z) i w(i) = log_2(n/a) odakle dobijemo broj bitova unutar adrese potrebnih za pomak i indeks, a i oznaku preko w(o) = w(adresa) – w(i) – w(p), a u našem slučaju w(adresa) = 32.

Iz formula dobijemo w(p) = 4 , w(i) = 4, w(o) = 24. Super! Sad ide „najzabavniji“ dio. Prije nego što crtamo tablicu, važno je prepoznati kada je pogodak(hit – nađeno u priručnoj memoriji), kada je promašaj (miss – dohvat iz RAM-a), te promašaj sa zamjenom (miss with replacement - promašaj, ali se mijenja neki prošli sadržaj koji je dohvaćen iz RAM-a u priručnu memoriju).

Pravilo je: hit – isti indeks i ista oznaka

miss – novi indeks

miss sa promjenom – isti indeks sa novom oznakom

Mali note: Ja sam to shvatio tako da ovisno o tome koliko je asocijativna memorija, toliko „mjesta“ ima u liniji, u našem slučaju je to 2.

Pa hajdemo onda!

Reference	Zapis nama važnih bitova	Indeks	Oznaka	Hit/Miss
0x0001				
0x0086				
0x00d4				
0x0001				
0x0187				
0x00d5				
0x01d2				
0x0281				
0x0002				
0x008c				
0x0189				
0x00dd				

Struktura nam je 24 (oznaka) + 4 (indeks) + 4 (pomak), dakle važna su nam prva dva dijela, oznaka i indeks, pa pišemo: 0x0001 = sve nule i zadnja 4 bita su 0001, sve nule zadnjih 8 bitova 1000 0110 itd.

Reference	Zapis važnih bitova	Indeks	Oznaka	Hit/Miss
0x0001	0000 0001			
0x0086	1000 0110			
0x00d4	1101 0100			
0x0001	0000 0001			
0x0187	1000 0111			
0x00d5	1101 0101			
0x01d2	1101 0010			
0x0281	1000 0001			
0x0002	0000 0010			
0x008c	1000 1100			
0x0189	1000 1001			
0x00dd	1101 1101			

Iz važnih bitova nađemo indeks!

Reference	Zapis važnih bitova	Indeks	Oznaka	Hit/Miss
0x0001	0000 0001	0		
0x0086	1000 0110	8		
0x00d4	1101 0100	13(d)		
0x0001	0000 0001	0		
0x0187	1000 0111	8		
0x00d5	1101 0101	13(d)		
0x01d2	1101 0010	13(d)		
0x0281	1000 0001	8		
0x0002	0000 0010	0		
0x008c	1000 1100	8		
0x0189	1000 1001	8		
0x00dd	1101 1101	13(d)		

Sada nađemo vrijednost oznake, ne vidimo sva 32 bita ali pretpostavka je takva da su sve nule slijeva, tj. proširenje najznačajnijeg bita.

Reference	Zapis važnih bitova	Indeks	Oznaka	Hit/Miss
0x0001	0000 0001	0	0	
0x0086	1000 0110	8	0	
0x00d4	1101 0100	13(d)	0	
0x0001	0000 0001	0	0	
0x0187	1000 0111	8	1	
0x00d5	1101 0101	13(d)	0	
0x01d2	1101 0010	13(d)	1	
0x0281	1000 0001	8	2	
0x0002	0000 0010	0	0	
0x008c	1000 1100	8	0	

0x0189	1000 1001	8	1	
0x00dd	1101 1101	13(d)	0	

I sada promatramo dobivenu tablicu i gledamo je li hit ili miss ili m(r). Prvi je automatski miss pošto je prazna memorija. Drugo je miss i treće je miss pošto su drugi indeksi u pitanju.

E sada dolazimo do četvrte reference koja je istog indeksa i iste oznake, to je hit! Peta referenca ima isti indeks ali drugačiju oznaku, da nam je jednostruko asocijativno, to bi bilo miss with replacement, ali pošto je dvostruko, to je samo miss i ide na drugo „mjesto“.

Reference	Zapis važnih bitova	Indeks	Oznaka	Hit/Miss
0x0001	0000 0001	0	0	M
0x0086	1000 0110	8	0	M
0x00d4	1101 0100	13(d)	0	M
0x0001	0000 0001	0	0	H
0x0187	1000 0111	8	1	M
0x00d5	1101 0101	13(d)	0	
0x01d2	1101 0010	13(d)	1	
0x0281	1000 0001	8	2	
0x0002	0000 0010	0	0	
0x008c	1000 1100	8	0	
0x0189	1000 1001	8	1	
0x00dd	1101 1101	13(d)	0	

Šesta referenca je istog indeksa i iste oznake i to je hit! Sedma je isti indeks, druga oznaka, dakle miss. Osma referenca, isti indeks druga oznaka, no naša linija već ima ta dva popunjena „mjesta“! Što sad? Zadano je u zadatku da je tip LRU – least recently used. Dakle, uzimamo podatak iz RAM-a (miss) i stavljamo ga na mjesto po LRU, a to je prvo „mjesto“ i=8; o=0! Deveta je hit, deseta je je isti indeks, ali druga oznaka, zato što smo onaj prvi 8 0 zamijenili sa 8 2, i pošto je LRU u pitanju, sada 8 0 stavljamo na drugo „mjesto“, dakle na mjesto 8 1! Jedanaesta je ista priča kao deseta, dakle m(r) koji ide na prvo mjesto. I zadnje dvanaesto je hit.

Reference	Zapis važnih bitova	Indeks	Oznaka	Hit/Miss
0x0001	0000 0001	0	0	M
0x0086	1000 0110	8	0	M
0x00d4	1101 0100	13(d)	0	M
0x0001	0000 0001	0	0	H
0x0187	1000 0111	8	1	M
0x00d5	1101 0101	13(d)	0	H
0x01d2	1101 0010	13(d)	1	M
0x0281	1000 0001	8	2	M(R)
0x0002	0000 0010	0	0	H
0x008c	1000 1100	8	0	M(R)
0x0189	1000 1001	8	1	M(R)
0x00dd	1101 1101	13(d)	0	H

Sada kada smo popunili tablicu možemo izračunati prosječno vrijeme pristupa memoriji, a to je:

Broj referenci = 12 . Suma vremena potrebnih je: (broj hitova(ne iz bonga) * pristup priručnoj memoriji) + (broj missova * pristup RAM-u) = 8*100T + 4*1T= 804T, prosječno vrijeme je 804/12=67T.

E sada dolazi dodatno kompliciranje stvari, ali mislim da nakon što prođete ovaj zadatak da će vam svi mogući zadaci ovog konteksta biti jasni.

b) Promjena prosječnog vremena pristupa uz osmerostruko asocijativnu L2 kapaciteta 1 MB, linije 16 B, LRU, latencija 10T.

Pišemo za L2: a = 8, b= 16 B, s = 1 MB = 2^{20} B, z = 1.

Sada opet s = n * b, dobijemo n = 2^{16} .

Opet koristimo w(p) = log_2(b/z) i w(i) = log_2(n/a) i dobijemo da nam je w(p) = 4, w(i) = 13, te iz w(adresa) = w(o) + w(i) + w(p) dobijemo w(o) = 15. I to je to, nema više tablica.

Šalim se. Sad dolazi pravo sranje. Koristimo tablicu od gore i radimo novu, proširenu. Isto gledamo bitove prvo, pa određujemo indeks i oznaku i onda gledamo hit ili miss. **Zapis važnih bitova u tablici je da se olakša posao određivanja indeksa i oznake, ako znate napamet ili nešto tako, slobodno izostavite. Za L2 zapis važnih bitova ću pisati bez dijela koji je pomak(on nam na kraju i nije važan, ali pisao sam ga da bude jasno šta je šta) da stane.**

Znači: o 15, i 13, p 4

Reference	Zapis važnih bitova L1	Indeks L1	Oznaka L1	Hit/Miss L1	Zapis važnih bitova L2	Indeks L2	Oznaka L2	Hit/Miss L2
0x0001	0000 0001	0	0	M	13 nula			
0x0086	1000 0110	8	0	M	9 nula i 1000			
0x00d4	1101 0100	13(d)	0	M	9 nula i 1101			
0x0001	0000 0001	0	0	H	13 nula			
0x0187	1000 0111	8	1	M	8 nula i 1 1000			
0x00d5	1101 0101	13(d)	0	H	9 nula i 1101			
0x01d2	1101 0010	13(d)	1	M	8 nula i 1 1101			
0x0281	1000 0001	8	2	M(R)	7 nula i 10 1000			
0x0002	0000 0010	0	0	H	13 nula			
0x008c	1000 1100	8	0	M(R)	9 nula i 1000			
0x0189	1000 1001	8	1	M(R)	8 nula i 1 1000			
0x00dd	1101 1101	13(d)	0	H	9 nula i 1101			

Sada očitamo indekse i oznake i upišemo u tablicu kao i za L1.

Napomena: Kod indeks L2 značenje npr. 24(16+8) je ovo: 24 – vrijednost iz binarnog zapisa 0 0000 0001 1000 = 24, gdje sa (16 + 8) označavam bitove koje sam zbrajao da dobijem taj indeks.

Reference	Zapis važnih bitova L1	Indeks L1	Oznaka L1	Hit/Miss L1	Zapis važnih bitova L2	Indeks L2	Oznaka L2	Hit/Miss L2
0x0001	0000 0001	0	0	M	13 nula	0	0	
0x0086	1000 0110	8	0	M	9 nula i 1000	8	0	
0x00d4	1101 0100	13(d)	0	M	9 nula i 1101	13(d)	0	
0x0001	0000 0001	0	0	H	13 nula	0	0	
0x0187	1000 0111	8	1	M	8 nula i 1 1000	24(16+8)	0	
0x00d5	1101 0101	13(d)	0	H	9 nula i 1101	13(d)	0	
0x01d2	1101 0010	13(d)	1	M	8 nula i 1 1101	29(16+13)	0	
0x0281	1000 0001	8	2	M(R)	7 nula i 10 1000	40(32+8)	0	
0x0002	0000 0010	0	0	H	13 nula	0	0	
0x008c	1000 1100	8	0	M(R)	9 nula i 1000	8	0	
0x0189	1000 1001	8	1	M(R)	8 nula i 1 1000	24	0	
0x00dd	1101 1101	13(d)	0	H	9 nula i 1101	13(d)	0	

E sada kada imamo i to popunjeno, treba odrediti kada je hit, miss ili miss with replacement.

Prva tri slučaja su ista za L1 i L2, missovi. Četvrti slučaj je hit kod L1 i kod L2, ali pošto je bio hit kod L1, hit kod L2 je nevažan(pisat ću ga kao H--). Peti je miss, šesti isto kao četvrti. Sedmi je miss za L1, gledamo L2 i tamo je isto miss! Osmi je miss(r) kod L1, kod L2 je on miss. Deveti je hit kod L1 pa L2 je H--. E sada deseti je po L1 m(r), ali kada gledamo u L2 vidimo da smo našli, dakle prvi normalan hit! Jedanaeasti je isti kao deseti, i dvanaeasti je hit kod L1, dakle kod L2 je H--.

Reference	Zapis važnih bitova L1	Indeks L1	Oznaka L1	Hit/Miss L1	Zapis važnih bitova L2	Indeks L2	Oznaka L2	Hit/Miss L2
0x0001	0000 0001	0	0	M	13 nula	0	0	M
0x0086	1000 0110	8	0	M	9 nula i 1000	8	0	M
0x00d4	1101 0100	13(d)	0	M	9 nula i 1101	13(d)	0	M
0x0001	0000 0001	0	0	H	13 nula	0	0	H--
0x0187	1000 0111	8	1	M	8 nula i 1 1000	24(16+8)	0	M
0x00d5	1101 0101	13(d)	0	H	9 nula i 1101	13(d)	0	H--
0x01d2	1101 0010	13(d)	1	M	8 nula i 1 1101	29(16+13)	0	M
0x0281	1000 0001	8	2	M(R)	7 nula i 10 1000	40(32+8)	0	M
0x0002	0000 0010	0	0	H	13 nula	0	0	H--
0x008c	1000 1100	8	0	M(R)	9 nula i 1000	8	0	H
0x0189	1000 1001	8	1	M(R)	8 nula i 1 1000	24	0	H
0x00dd	1101 1101	13(d)	0	H	9 nula i 1101	13(d)	0	H--

I zadnja stvar koja nam preostaje je izračunati poboljšanje! I dalje imamo 12 referenci, iz dijela tablice za L1 vidimo da su 4 hita L1, iz tablice za L2 vidimo da su 2 hita L2, a preostalih 6 referenci su missovi!

Dakle sada prosječno vrijeme je : 4*1T + 2*10T + 6*100 T = 624 T podijeljeno sa 12 je 52T!

I eto nam rješenja, uvođenjem L2 prosječno vrijeme nam se smanjilo za 67T – 52T = 15T.

Napomena: Nisam siguran događa li se m(r) kod L1 u prisustvu L2 ili je to samo miss bez replacementa, ali pretpostavljam da se događa m(r) zato što nema smisla da jednom popunjena L1 ostane isto popunjena cijelo vrijeme tako da zaključujem da se događa m(r).

Npr. imamo liniju kod L1 indeks 8 sa 2 mjesta, na prvom mjestu je sa oznakom 0, na drugom sa oznakom 1 i isto to na nekim indeksima u L2. Dolazi referenca gdje je indeks 8 i oznaka 2, događa se m(r) i ide na prvo mjesto te je sada raspored ind 8 ozn 2, ind 8 ozn 1. I sada kada bi se tražilo indeks 8 ozn 0 našlo bi se u L2, jer u L1 je zamijenjeno.

5.

Za ove zadatke sa n-komponentnim vektorima ne treba znati puno, samo par naredaba i to je to.

Što god došlo, bit će neka jednostavna operacija tipa množenje, oduzimanje, zbrajanje, dijeljenje(malo kompliciranije) ili u ovom slučaju skalarni umnožak(što je kompliciranije).

Koje naredbe moramo znati?

ldv – učitavanje u vekt registar

stv – spremanje vekt registra u memoriju, st i ld su spremanje i učitavanje običnih registara

addi – zbrajanje registra sa konstantom i spremanje u registar

subv – oduzimanje dva vekt registra

addv – zbrajanje dva vekt registra, add i sub su zbrajanje i oduzimanje dva skalara

mulsv – množenje dva vekt registra, mul je mnozenje 2 skalara

I to je to. Sintaksu ćete skužiti iz rješenja ovog zadatka.

Zadana nam je i funkcija sumv v,r koja računa sumu svih brojeva u vektorskom registru.

Zadano nam je 64-komponentni vektori, veličina vektorskih registra v0-v7 je 256 bita, početna adresa vektora A i B su $a i $b, a skalara c i s su $c i $s.

Treba napisati c = s(A*B) gdje je A*B skalarni umnožak.

Prva stvar koju gledamo je, hm, naši vektorski registri su 256 bita, 32 bajta, a 64 - komponentne vektore imamo. 64 puta po 4 bajta jeeee… 256 bajta. Jebote, treba nam petlja. Da su 32 – komponentni ne bi nam trebala jer bi onda vektor a stao u prva 4 registra, a vektor b u druga 4 pa bi jedan prolazak kroz instrukcije bio dovoljan da se obavi posao. Pa krenimo sa pisanjem.

Napomena: Palo mi je na pamet da možda postoje stvari koje nisam napisao na ispitu, ali moguce da sam trebao, ali su mi oni svejedno dali sve bodove na zadatku. Nakon ovog sa ispita ću napisati dodatno to što mi je palo na pamet.

```
            addi r7, $a, #256 //da možemo usporediti jesmo li završili ili ne
            ld r4, 0($s) // učitavamo skalar s u registar r4
PETLJA:     ldv v0, 0($a) // učitavamo u registre
            ldv v1, 32($a)
            ldv v2, 64($a)
            ldv v3, 96($a)
            ldv v4, 0($b)
            ldv v5, 32($b)
            ldv v6, 64($b)
```

```
        ldv v7, 96($b)
        mulv v0, v0, v4 // množimo a0*b0, a1*b1,…
        mulv v1, v1, v5
        mulv v2, v2, v6
        mulv v3, v3, v7
        sumv v0, r0 // zbrajamo vrijednosti unutar vektorskih registara i spremamo u registre
        sumv v1, r1 // jer je skalarni produkt= a0*b0 + a1*b1 + … + an*bn
        sumv v2, r2
        sumv v3, r3
        add r0, r0, r1 //zbrajanjem registara imamo u r0 konacni rezultat skalarnog umnoska
        add r0, r0, r2
        add r0, r0, r3
        mul r5, r0, r4 //množimo skalar s sa konacnim rezultatom skalarnog umnoska r0
        st r5, 0($c) //spremamo rezultat na adresu od c
        addi $a, $a, #128 //dodamo 128 tako da mozemo jos 1 kroz petlju,tj do kraja podatke
        addi $b, $b, #128
        addi r6, $a, #0 // sada u r6 stavljamo $a zbog iduceg koraka – provjere petlje
        bne r7, r6, PETLJA // provjera je li r7 jednako r6
```

E sad, caka je ta dok sam rješavao i pisao ovo, shvatio sam da po ovome gore rezultat od prvog prolaska kroz petlju u r5 bi se spremio na adresu $c, a onda u drugom prolasku kroz petlju bi se ***drugi*** rezultat spremio na $c, znači ne bi imali 1.prolaz + 2. prolaz nego bi imali samo zadnje spremljeno na adresu $c što bi bilo samo 2. prolaz.

```
        addi r7, $a, #256 //da možemo usporediti jesmo li završili ili ne
        ld r4, 0($s) // učitavamo skalar s u registar r4
        sub r5, r4, r4 // postavimo da je r5=0
PETLJA: ldv v0, 0($a) // učitavamo u registre
        ldv v1, 32($a)
        ldv v2, 64($a)
        ldv v3, 96($a)
        ldv v4, 0($b)
```

ldv v5, 32($b)

ldv v6, 64($b)

ldv v7, 96($b)

mulv v0, v0, v4 // množimo a0*b0, a1*b1,…

mulv v1, v1, v5

mulv v2, v2, v6

mulv v3, v3, v7

sumv v0, r0 // zbrajamo vrijednosti unutar vektorskih registara i spremamo u registre

sumv v1, r1 // jer je skalarni produkt= a0*b0 + a1*b1 + … + an*bn

sumv v2, r2

sumv v3, r3

add r0, r0, r1 //zbrajanjem registara imamo u r0 konacni rezultat skalarnog umnoska

add r0, r0, r2

add r0, r0, r3

mul r1, r0, r4 //množimo skalar s sa konacnim rezultatom skalarnog umnoska r0

add r5, r5, r1 // u prvom prolasku je ovo 0+rez prvog prolaska, u drugom je to rezultat prvog prolaska(r5) + rezultat drugog prolaska

st r5, 0($c) //spremamo rezultat na adresu od c

addi $a, $a, #128 //dodamo 128 tako da mozemo jos 1 kroz petlju,tj do kraja podatke

addi $b, $b, #128

addi r6, $a, #0 // sada u r6 stavljamo $a zbog iduceg koraka – provjere petlje

bne r7, r6, PETLJA // provjera je li r7 jednako r6