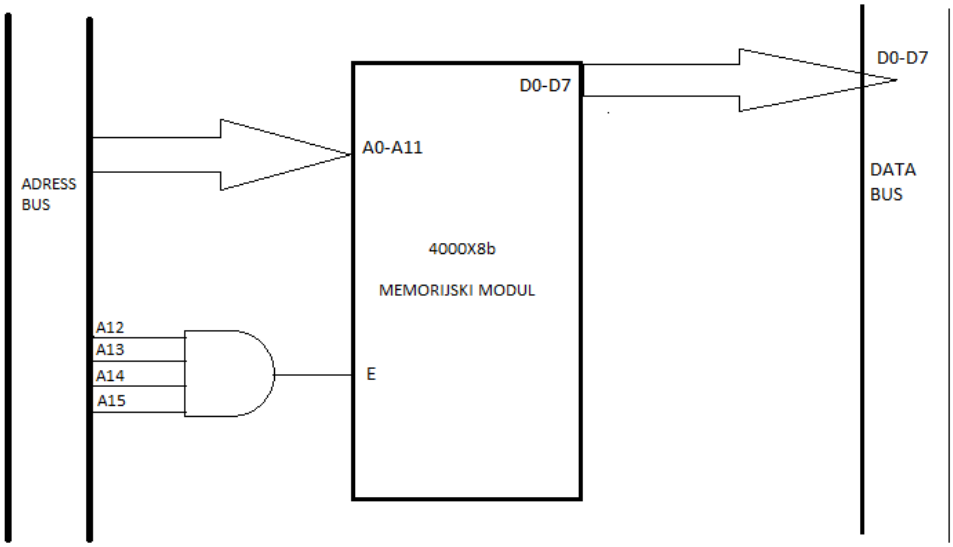
ARHITEKTURA RAČUNALA 2
Akademska godina 2009/10.
Rjesenja završnog ispita 18.01.2010.

(by Acid_Spunk, Bubenšvanc & futo)

1. Na procesor sa 16-bitnom adresnom i 8-bitnom podatkovnom sabirnicom potrebno je spojiti memorijski modul s jednim ulazom za omogućavanje veličine 4k x 8 bita.Modul se treba javljati u dijelu adresnog prostora koji počinje od adrese `0xF000`.

RJEŠENJE (by Acid_Spunk):

Znači ovdje se radilo o jednostavnom ROM modulu. Adresni prostor počinje od adrese 0xF000, pa stoga adresni ulazi A11-A15 moraju biti spojeni na I sklop i zatim na ulaz za omogućavanje E, a ostali direktno na adresne priključke modula. Podatkovni priključci D0-D7 spajaju se na podatkovnu sabirnicu. Upravljački signali se ne spominju u zadatku, pa zato veza sa upravljačkom sabirnicom ne postoji.



2. Pokaži kako bismo sljedeće instrukcije uklopili u temeljnu protočnu organizaciju arhitekture MIPS:

- **`Lwinc Rd, offset, Rs  #  Rd <- Mem[Rs+offset]; Rs=Rs+4`**
- **`Addmem Rd, offset, Rs  #  Rd <- Rd + Mem[Rs+offset]`**

Koje bi dodatno sklopovlje trebalo uvrstiti u put podataka za svaku od navedene dvije instrukcije, te kako bi se njihovo uvrštavanje odrazilo na svojstva temeljne protočne strukture arhitekture?

RJEŠENJE (by Acid_spunk):

U ovom zadatku trebalo je dodati neke elemente u protocnu strukturu MIPS, kako bismo mogli izvrsiti zadane instrukcije. Od elemenata koji su nam na raspolaganju mozemo koristiti sklopove za zbrajanje, posmak, mnozenje, itd itd.

U slajdovima imamo podatak da, sto se tice vremenskog trajanja ALU operacija, posmak traje 1 delta T, operacije AND,ADD i COMP traju 2 delta T, a mnozenje traje 6, a dijeljenje 2 delta T.

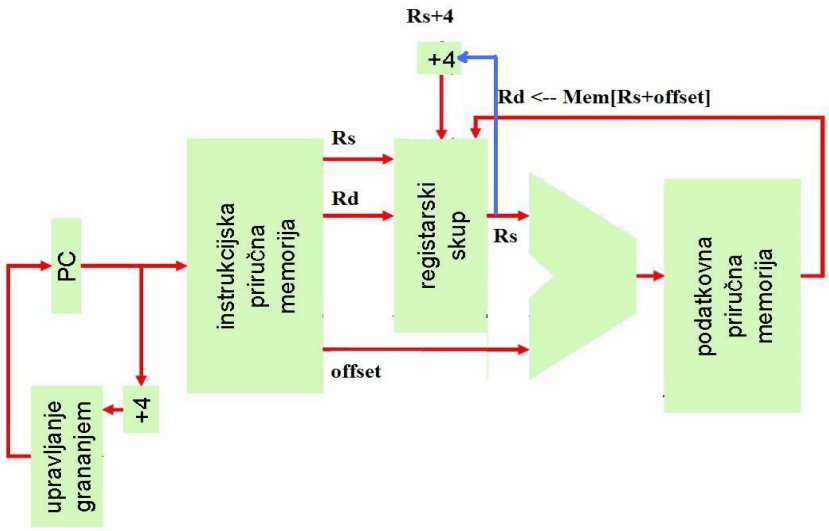
a) `lwinc Rd, offset, Rs # Rd <- Mem[Rs + offset]; Rs=Rs+4`

Dakle naša instrukcija mora zbrojiti registar Rs sa konstantom offset, zapisati rezultat u registar Rd, i uvecati Rs za 4.

Ovo se postize tako, da dodamo jedan sklop za zbrajanje, koji cemo spojiti na izlaz Rs. Rezultat zbrajanja ce biti upisan u registar Rs.

Kako ce se to odraziti na protocnu strukturu? Nisam siguran, ali mislim da se vrijeme izvrsavanja produlji za 1 delta T, buduci da zbrajanje {Rs+4} traje jednako dugo kao i zbrajanje {Rs + offset}, odnosno 2 delta T, morat cemo jos jedan period vremena iskoristiti za upis rezultata {Rs+4} u registar Rs, što se mora izvrsiti u dodatnom ciklusu prije pristupa memoriji.

Slika rjesenja:

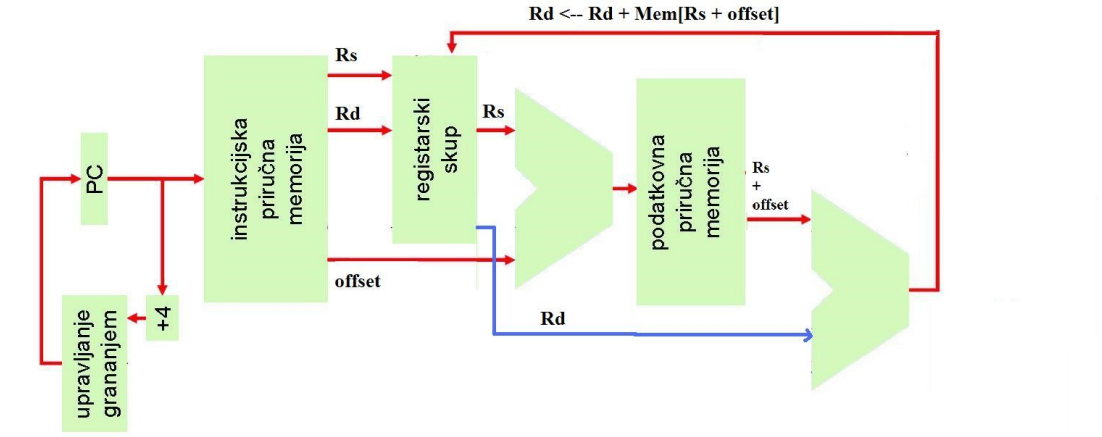


b) `addmem Rd, offset, Rs # Rd <-Rd + Mem[Rs + offset]`

Ovdje je situacija nesto drugacija nego kod prosle instrukcije, zbog toga sto rezultat zbrajanja {Rs + offset} predstavlja adresu u podatkovnoj memoriji s koje moramo dohvatiti podatak, a zatim taj podatak zbrojiti sa vrijednosti registra Rd i zapisati rezultat natrag u Rd.

Ovdje se zbrajanje ne moze napraviti paralelno, pa zato moramo uvesti novi segment, EX2, tj. novi ALU sklop, i to nakon segmenta MEM (pristupa podatkovnoj memoriji). Buduci da znamo da zbrajanje traje 2 perioda izvodjenja, tako ce cjelokupno izvodjenje instrukcije trajati duze za (2 delta T).

Slika rjesenja:



3. Zadana je sljedeća programska petlja u strojnom jeziku za računala arhitekture MIPS ($s1,$s3 i $t0 predstavljaju registre opće namjene):

```
Loop:
     lw   $t0, 0($s1)          #load word
     addi $t0, $t0, 25         #add immediate
     sw   $t0, 0($s1)          #store word
     addi $s1, $s1, 4          #add immediate
     bne  $s1, $s3, Loop       #branch if not equal
```

- Skicirati kod u C-u koji bi na klasičnom SISD procesoru mogao biti preveden u zadani strojni kod.
- Za petlju iz prethodnog pitanja prikazati kakav bi kod bio generiran za procesor s dvostrukim statičkim izdavanjem koji istovremeno može izvoditi:

  - jednu instrukciju tipa ALU/branch
  - jednu instrukciju tipa load/store

  Pretpostaviti da se učitani registar ne može koristiti u instrukciji neposredno nakon instrukcije čitanja memorije. Odrediti CPI koji se postiže u tijelu petlje ako zanemarimo instrukcije `nop`.
- Pokušaj ponuditi raspored u kojem jedan prolaz kroz petlju troši samo četiri ciklusa
- (BONUS) Pokušaj ponuditi raspored u kojem četiri prolaza kroz petlju troši osam ciklusa

RJEŠENJE (by futo):

a) Ekivalentni dio c koda:

```
int *s1 = &start;
int *s3 = &end; // proizvoljno...

while (s1 < s3){
  int t = *s1;
  t += 25;
  *s1  = t;
  s1++; // dovoljno ++, pointer je int pointer
}
```

b), c)

ciklus	alu/branch	sw/lw
1	`nop`	`lw $t0, 0($s1)`
2	`addi $s1, $s1, 4`	`nop`
3	`addi $t0, $t0, 25`	`nop`
4	`bne $s1, $s3, Loop`	`sw $t0, 0($s1)`

Znači, u prvom koraku učitamo u t0, onda ne smijemo opet koristiti isti registar (piše u zadatku) pa možemo povećati s1 (adresu s koje čitamo), zatim povećamo t0, i možemo ga spremiti opet na adresu koja piše u s1, nakon toga provjera za skok.

Primjetiti da je to odmah rješenje i za **b** i **c** zadatke, **CPI = 4/5** (broj redova u tablici/ukupno instrukcija("dugih riječi", dvostruko izdavanje..). Može tako, potvrdio asistent Hrkač na ispitu.

4. Intelov procesor 80486 imao je 32-bitnu adresnu sabirnicu. Priručna memorija prvih modela tog procesora bila je četverostruko asocijativna, a sastojala se od 256 linija po 32 bajta. Odrediti omjer neto veličine te priručne memorije i ukupnog broja bitova potrebnih za njenu realizaciju, pod pretpostavkom minimalnog broja servisnih bitova.

RJEŠENJE (by Acid_Spunk):

Imamo adresu od 32 bita, PM je 4x asocijativna, 256 linija po 32 bajta.

Trazi se s_neto / s_ukupno , ako imamo minimalan broj servisnih bitova

n=256
b=32B
a=4

s_neto = n*b = 256*32*8 = 65536 bita

w(i) = log2(n/a) = log2(64) = 6 bita

w(p) = log2(b) = 5 bita

w(o) = 32 - w(i) - w(p) = 21 bit

Minimalni broj servisnih bitova je 2 (bitovi V i D = Valid i Dirty)

s_ukupno= (w(o) + s_bitovi + b)*n = (21 + 2 + 32*8) * 256 = 279 * 256 = 71424 bita

s_neto/s_ukupno = 65536/71424= 0,917 = 92%

RJEŠENJE (by Bubenšvanc): priloženo uz dokument (slikani postupak)

5. Zadan je procesor s 8-bitnom adresnom i podatkovnom sabirnicom. Priručna memorija procesora je dvostruko asocijativna, a sastoji se od 8 linija po 8 bajta. Pretpostavimo da su sve linije priručne memorije zauzete, te da se u poljima oznaka linija nalaze sljedeći podatci (redoslijed odgovara rastućim indeksima linija): [0, 1, 0, 2, 0, 3, 0, 2]

Pod pretpostavkom istog zadanog stanja priručne memorije, odrediti ishode (pogodak ili promašaj) slijedećih pristupa: `$a2, $65, $89, $ca, $54, $30, $bf, $18`

RJEŠENJE (by Acid_Spunk):

Imamo 8 bitnu adresu, PM je 2x asocijativna, ima 8 linija po 8 bajta. Na početku su sve linije PM zauzete.

Prvo izračunajmo format adrese:

w(i) = log2(n/a) = log2(4) =2 bita

w(p) =log2(b) = log2(8) = 3 bita

w(o) = 8 - 2 - 3 = 3 bita

Format adrese : `ooo`**`ii`**`ppp`

Znači imamo 4*2 linije, indeksirane sa 4 različita indeksa, 0, 1, 2 i 3.

Zadano je početno stanje PM, i ono izgleda ovako:

indeks linije	oznaka 1	oznaka 2
0	0	1
1	0	2
2	0	3
3	0	2

Zadani su pristupi stranica, **pod pretpostavkom da su početni uvjeti za svaki pristup jednaki.** To znači da ne promatramo pristupe slijedno i ne radimo promjene kada se dogodi promašaj, nego samo promatramo svaki pristup sam za sebe.

Adrese kojima se pristupa pretvorit ćemo u binarni zapis, i iz njega pročitati oznaku, indeks i pomak. Potom uspoređujemo dobivene podatke sa stanjem memorije i zaključujemo radi li se o pogotku ili promašaju:

$a2 => 101|00|010 indeks 0, oznaka 5 **PROMAŠAJ**

$65 => 011|00|101 indeks 0 oznaka 3 **PROMAŠAJ**

$89 => 100|01|001 indeks 1 oznaka 4 **PROMAŠAJ**

$ca => 110|01|010 indeks 1 oznaka 6 **PROMAŠAJ**

$54 => 010|10|100 indeks 2 oznaka 2 **PROMAŠAJ**

$30 => 001|10|000 indeks 2 oznaka 1 **PROMAŠAJ**

$bf => 101|11|111 indeks 3 oznaka 5 **PROMAŠAJ**

$18 => 000|11|000 indeks 3 oznaka 0 **POGODAK**

Znači, od 8 pristupa imamo **samo 1 pogodak**.

RJEŠENJE (by Bubenšvanc): priloženo uz dokument (slikani postupak)

6. Razmotrimo proces koji zauzima sljedeće lokacije virtualne memorije:

- adrese 0x00000000 – 0x00003490 : program i statistički podatci

- adrese 0xfffff234 – 0xffffffff : stog

Pretpostavimo da se proces izvodi na procesoru s dvorazinskim straničenjem (npr. Pentium I), pri čemu je struktura virtualne adrese kako slijedi:

- gornjih 10 bitova: indeks u straničnom imeniku
- srednjih 10 bitova: indeks u straničnoj tablici
- donjih 12 bitova: pomak unutar stranice

Odrediti koliko memorije zauzimaju strukture za ostvarivanje adresnog preslikavanja straničenjem (stranični imenici i tablice), ako svaka stavka u straničnom imeniku odnosno straničnoj tablici zauzima 4B. Kolika je to ušteda u odnosu na linearnu straničnu tablicu?

RJEŠENJE (by Bubenšvanc): priloženo uz dokument (slikani postupak)