Semestralni rad iz predmeta Praktikum iz digitalnih signalnih procesora

Dušan Bižić, broj indeksa 2020/0090 i Dositej Cvetković, broj indeksa 2020/0224

22. januar 2023

Elektrotehnički fakultet, Univerzitet u Beogradu

Zadatak ovog semestralnog rada je bio da se napravi program u asemblerskom jeziku, za *Texas instruments* digitalne signalne procesore, koji radi inverziju 3×3 matrice sa 16-bitnim celim neoznačenim brojevima u opsegu od 0 do 31. U nastakvu je dat kod sa detaljnim objašnjenjem koraka izvršavanja.

```
   .bss   red1,3
   .bss   red2,3
   .bss   red3,3
   .bss   det, 1
   .bss   pom,1
   .bss   pom2,1
   .bss   min1,1
   .bss   min2,1
   .bss   min3,1
   .bss   min4,1
   .bss   min5,1
   .bss   min6,1
   .bss   min7,1
   .bss   min8,1
   .bss   min9,1
   .bss   delioc, 1
   .bss   deljenik, 1
   .bss   resenje,1
   .bss   brojac, 1
   .bss   kolicnik, 1
   .bss   znak,1
   .bss   deljenik_tmp,1
   .bss   delioc_tmp, 1
   .bss   brojac2, 1
;*******************************;


      .text
      .global _c_int0

; --------------------------------;

_c_int0:

    ;ucitivanje vrednosti

    CLRC  CNF
    SPM   0

    SETC  SXM
    LDP   #2h

    lar   AR0,#red1
    lar   AR1,#red2
    lar   AR2,#red3
```

Ovo je inicijalizacioni deo gde se deklarišu potrebne promenljive.

`CLRC CNF` on-chip namešta bit konfiguracije DARAM-a u smislu da li se dual-access RAM blokovi mapiraju u programskom prostoru ili u prostoru podataka.

`SPM 0` zadaje nultu vrednost shift-a u desno svih prethodnih transfera PREG-a.

`LDP #2h` inicijalizuje memorijsku stranu na kojoj će se upisivati promenljive, 2h nam kaže da će se upisivati od druge, što je memorijska lokacija 100h.

`lar` naredba postavlja pokazivače registara AR na alocirane vektore matrice.

```
mar   *,AR0
lacc #10
sacl *+
lacc #15
sacl *+
lacc #1
sacl *
rpt #1
mar *-

mar   *,AR1
lacc #7
sacl *+
lacc #8
sacl *+
lacc #9
sacl *
rpt #1
mar *-


mar   *,AR2
lacc #3
sacl *+
lacc #16
sacl *+
lacc #5
sacl *
rpt #1
mar *-
```

U ovom delu se unosi matrica koja će se u memoriji čuvati kao tri redna vektora. Unos se vrši tako što se `mar *, ARi` naredbom pokazivač na i-ti registar prebaci u *.

Dalje se koristi kombinacija `lacc #broj` i `sacl*+` naredbe da bi se u akumulator uneo broj, sačuvala vrednost u * i inkrementirao pokazivač na sledeću lokaciju tekućeg vektora.

`mar *-` i `mar *+` bez naznake registra se koriste kako bi se pokazivač na memorijsku lokciju u * inkrementovao ili dekrementovao. Za ovaj primer uneta je matrica

$$\mathbf{A} = \begin{bmatrix} 10 & 15 & 1 \\ 7 & 8 & 9 \\ 3 & 16 & 5 \end{bmatrix}$$

```
zac

mar *, AR1  ; ar1->a21
mar *+   ; ar1->a22

lt *+,AR2  ; ar1->a23
; TREG->a22

mar *+  ; ar2->a32
mar *+  ; ar2->a33
mpy *-,AR1  ; ar2 -> ar32
 ;PREG->a22*a33
pac

;ACC->a22*a33


lt *-,AR2  ; TREG->a23 ar1->a22
mpy *+,AR1  ; PREG-> a32*a23

;ar2->a33

spac  ; ACC-> a22*a33 - a32*a23

sacl min1  ;cuvanje minora a11
```

Naredne tri sekcije koda računaju minore m_{11}, m_{12} i m_{13} uz pomoć kojih će se računati determinanta preko formule $|\mathbf{A}| = a_{11}m_{11} - a_{12}m_{12} + a_{13}m_{13}$. Odgovornost se prebacuje na potrebne registre u zavisnosti od toga koji minor se računa. Elementi se množe i čuvaju u akumulator naredbama `lt*`, `mpy*` i `pac`.

`ACC -> a22*a33` komentar znači da se trenutno u akumulatoru čuva vrednost minora $m_{11} = a_{22}a_{33} - a_{32}a_{23}$.

```
mar *, AR0
sacl det
lt det
mpy *+  ;ar0 -> a12

pac
sacl det

zac
```

Trenutni minor se u P registru množi sa vrednošću a_{11} iz matrice, zatim se prebacuje u akumulator naredbom `pac` i rezultat se čuva u promenljivoj `det`. Tu će se čuvati konačni rezultat determinante.

Bitno je napomenuti da se naredbom `spac` od akumulatora oduzima vrednost P registra. Pri računanju minora m_{12} je potrebo uraditi $a_{21}a_{33} - a_{23}a_{31}$, što se postiže korišćenjem `spac` naredbe, Međutim potrebno je spremiti tekući minor da bude oduzet od prethodnog, tako da se i negira u akumulatoru naredvom `neg`. Ovim korakom dobijamo $-a_{12}m_{12}$

```
mar *, AR2  ; ar1->a22

;ar2->a33

lt *-, AR1  ; ar2->a32
mar *-  ; ar1->a21
mpy *+, AR2 ; ar1->a22
pac  ; ACC-> a21*a33



mar *-  ; ar2->a31
lt *+, AR1 ; ar2->a32
mar *+  ; ar1->a23
mpy *-, AR0 ; ar1->a22

spac ; ACC-> a21*a33-a23*a31
neg
sacl min2

; cuvanje minora a12
```

```
sacl pom
lt pom

mpy *+, AR2
pac
sacl pom

lacc det
add pom

sacl det
```

Oduzimanje minora $a_{12}m_{12}$ od $a_{11}m_{11}$, čuvanje u `det` i računanje poslednjeg minora.

```
zac  ;ar0 = a13 ar1 = a22 ar2 = a32

lt *-, AR1 ; ar2->a31
mar *-

mpy *+, AR2 ; PREG = a21*a32

;ar1 -> a22

pac
```

```
    lt *, AR1
    mpy *
    spac

    sacl min3
    ; cuvanje minora a33
```

Nakon što je iztačunat minor m_{13}, računa se konačna vrednost determinante prema napomenutoj formuli. Ako je vrednost determinante 0, preskače se ostatak postupka i završava se program. To je učinjeno dvema naredbama `lacc det` i `bz ako_je_det_0`. Poslednja naredba uradi branch na zadatu labelu ako je vrednost učitana u akumulatoru jednaka nuli.

```
    sacl pom
    lt pom

    mar *, AR0
    mpy *
    pac
    sacl pom

    lacc det
    add pom
    sacl det

    lacc det
    bz ako_je_det_0



;racunanje adjungovane


;AR0 = a13 AR1 = a22 AR2 = A31

    lt *-, AR2 ; ar0 => a12
    mar *+ ; ar2 => a32
    mpy *+ ; PREG => a32*a13

    ar2 => a33

    pac;

    lt *-, AR0  ; ar2 => a32
    mpy *+ ; ar0 => a13

    spac;

    sacl min4;


     ;AR0 = a13 AR2 = a32 AR1 = A22

    zac

    lt *-, AR2 ; ar0 => a12
    mar *-; ar2 => a31
    mpy *+, AR0  ; ar2 => a32
    pac ; ACC => a13*a31
    mar*-; ; ar0 => a11
    lt *+, AR2   ; ar0 => a12
```

Računanje adjungovane matrice se svodi na ponavljanju prethodnog postupka devet puta, tj. za svaki minor od m_{11} do m_{33}, sa tim što su minori od m_{11} do m_{13} sačuvani u promenljivama `min1`, `min2` i `min3` i u tim promenljivama nije bilo vršeno množenje sa elementima matrice na mestima adjungata, kao što i treba pri računanju minora za adjungovanu matricu. U narednom postupku nisu iskorišćene nove naredbe, tako da će se detaljno objašnjenje preskočiti.

```
    mar *+  ; ar2 => a33
    mpy *-,AR0   ; PREG => a11*a33

    ;ar2 => a32

    spac
    neg ; ACC = a11*a33 - a13*a31

    sacl min5


;AR0 = a12 AR1 = a22 AR2 = a32

    zac

    lt *-, AR2 ; ar0 => a11
    mar *-; ar2 => a31
    mpy *+, AR0  ; ar2 => a32
    pac ; ACC => a12*a31

    lt *+, AR2  ; ar0 => a12
    mpy*, AR0
    spac ; ACC => a12*a31 - a11*a32

    sacl min6


; AR0 = a12 AR1 = a22 AR2 = a32

    zac

    lt *+, AR1 ; ar0 => a13
    mar *+;      ; ar1 => a23
    mpy *-, AR0  ; ar1 => a22
    pac ; ACC = a12*a23

    lt *, AR1
    mpy *-,AR0 ; ar1 => a21
    spac ; ACC => a12*a23 - a13*a22

    sacl min7


; AR0 = a13 AR1 = a21 AR2 = a32

    zac

    lt *-, AR1   ; ar0 => a12
    mpy *+, AR0  ; ar1 => a22
    pac ; ACC => a13*a21

    mar *-; ar0 => a11
    lt *, AR1
    mar *+;      ; ar1 => a23
    mpy *-;      ; ar1 => a22
    spac ; ACC => a13*a21 - a11*a23

    sacl min8
```

U ovoj sekciji je naredbom `sacl min9` sačuvan poslednji minor. Time je izračunata adjungovama matrica i sačuvana u promenljivama od `min1` do `min9`.

Nakon komentara `;transponovanje` je izvršeno transponovanje prostim prebacivanjem minora iz respektivnih promenljivih nazad u početne redne vektor, u odgovarajućem redosledu.

```
; AR0 = a11 AR1 = a22 AR2 = a32

    zac

    lt *-, AR0 ; ar1 => a21
    mpy *+  ; ar0 => a12
    pac   ; ACC => a22*a11

    lt *, AR1
    mpy *
    spac  ; ACC => a22*a11 - a12*a21

    sacl min9



;transponovanje


; AR0 = a12 AR1 = a21 AR2 = a32
    mar*,AR0
    mar*-
    lacc min1
    sacl *+
    lacc min4
    sacl *+
    lacc min7
    sacl *
    rpt #1
    mar*-

    mar*,AR1
    lacc min2
    sacl *+
    lacc min5
    sacl *+
    lacc min8
    sacl *
    rpt #1
    mar*-

    mar *,AR2
    mar*-
    lacc min3
    sacl *+
    lacc min6
    sacl *+
    lacc min9
    sacl *
    rpt #1
    mar*-
```

```
;deljenje sa determinantom


   lacc det
   sacl delioc
   zac
   sacl brojac2

glavna_petlja:
   sub #9
   bgez ispisivanje_matrice

   lacc brojac2
   sub #6
   bgez iterate_AR2

   lacc brojac2
   sub #3
   bgez iterate_AR1

iterate_AR0:
   mar *, AR0
   lacc *
   sacl deljenik
   b deljenje

iterate_AR1
   mar *, AR1
   lacc *
   sacl deljenik
   b deljenje

iterate_AR2
   mar *, AR2
   lacc *
   sacl deljenik


deljenje:

   zac
   sacl brojac
   sacl resenje
   sacl kolicnik


   lt    deljenik
   mpy   delioc
   sph   znak

   lacc  deljenik
   abs
   sacl  deljenik_tmp

   lacc  delioc
   abs
   sacl  delioc_tmp
```

U ovoj sekciji se vrši deljenje svakog člana adjungovane i transponove matrice sa determinantom. Za početak se `det` učitava u promenljivu `delioc` i neće se menjati. Inicijalizuje se `brojac2` sa nulom.

Učitavanje tekuće vrednosti iz matrice u `deljenik` se vrši sekvencijalno, uz pomoć `brojac2` vrednosti. Vrednosti matrice su izdeljene u tri podgrupe, za svaki vektor. Iterira se po kolonama od elementa koji bi bio na poziciji 1,1.

Odmah se proverava da li je brojač veći ili jednak 9, ako jeste, znači da smo podelili devet elemenata sa `delioc` i algoritam deljenja se završava. Ako ovo nije ispunjeno, dalje se proverava da li je `brojac2` veči od 6, ako jeste, znači da su pređeni prvi i drugi vektor i prelazi se na poslednji (onaj na koji pokazuje `AR2`).

Dok vrednost u `brojac2` nije veća od one na trenutnom stepenu, odlazi se u podrutine `iterate ARi` gde $i = 0, 1, 2$. Podrutine su ispisane odmah nakon stepena proveravanja i to u obrnutom redosledu, tako da se stepen za proveru `AR0` direktno nadovezuje na `iterate AR0`. Za poslednji stepen važi, ako je `brojac2` veći ili jednak 3, onda je i dalje na vektoru `AR1`, bude brojač manji od 3, iterira se kroz `AR0`.

Sledeći korak jeste samo deljenje, nakon što je izabrana vrednost za `deljenik` iz prethodnog dela za iteraciju. Prvo se inicijalizuju sve potrebne promenljive na nulu, ovaj korak je ključan jer će se iste promenljive koristiti svaki put pri pozivanju podrutine `deljenje`.

Ovde se vrši provera znaka količnika množenjem `deljenik` i `delioc`. Znak se nalazi u višoj reči P registra i čuva u promenljivu `znak` naredbom `sph`. Za dalje potrebe, za pomnožene promenljive se koriste njihove apsolutne vrednosti. Apsolutne vrednosti se čuvaju u zasebnim promenljivama `deljenik_tmp` i `delioc_tmp`.

```
obrada_ostatka:
   lacc delioc_tmp
   sub deljenik_tmp

   bgez veci_delioc
   zac


   lacc  deljenik_tmp
   rpt   #15
   subc  delioc_tmp

   sacl  kolicnik
   sach  deljenik_tmp

   b  obrada_ostatka
```

Podrutina `obrada_ostatka` zapravo služi za određivanje celog dela količnika blokom od tri naredve koji se završava naredbom `subc` (conditional subtract). Naredba će u nižoj reči akumulatora sačuvati celobrojni `kolicnik`, a u višoj reči ostatak, koji postaje novi objekat deljenja i stoga se čuva u `deljenik_tmp`. Ispod labele `obrada_ostatka` se proverava da li je delioc veći od deljenika, ako jeste, ulazi se u zasebnu podrutinu određivanje izlomljenog dela rezultata.

```
veci_delioc:

   lacc deljenik_tmp, 1
   sacl deljenik_tmp

unutrasnja_petlja:
   lacc deljenik_tmp
   sub delioc_tmp
   bz  stepen_dva
   bgz vece_od_nula

   lacc resenje, 1
   sacl resenje
   lacc deljenik_tmp, 1
   sacl deljenik_tmp

   lacc brojac
   add #1
   sacl brojac
   sub #7
   bgz glavni_izlaz


   b unutrasnja_petlja
```

Postupak otpočinje rađenjem levog shift-a (SL) tekućeg deljenika. Levi shift idejno označava deo kada se prelazi na decimalu manje vrednosti. Ovo je zato što će za svaku decimalu biti odrađeno nešto nalik celobrojnom deljenju, i tome šta radi naredba `subc`, samo što se nakon oduzimanja proverava da li je rezultat manji od, veći ili jednak nuli. Ako je rezultat manji od nula, znači da je potrebno uraditi još jedan SL za `deljenik_tmp` i trenutna decimala ostaje na nuli.

Za rezultate koji će na kraju ispasti da su stepen $2^n, n < 0$ se ispostavlja da će rezultat oduzimanja u jednom trenutku biti nula. Ako se ovo ne uzme u obzir kao zaseban slučaj, otišlo bi se u podrutinu `vece_od_nula`. Ona se završava prerano i ne doda potreban broj 1 na najnižu decimalu. Zato je potrebno tretirati takav stepen dvojke kao zaseban slučaj i dodati potrebnu vrednost 1, koja bi inače bila izostavljena.

Ako ne dođe do prethodno navedenog slučaja, ili rezultat može prosto imati beskonačno mnogo delioca, on će biti ograničen na 8 bitova. Provera da li je odrađeno 8 bitova se postiže promenljivom `brojac`.

```
vece_od_nula:

   sacl deljenik_tmp
   lacc resenje
   sfl
   add #1
   sacl resenje
   lacc deljenik_tmp,1
   sacl deljenik_tmp

   lacc brojac
   add #1
   sacl brojac
   sub #7
   bgz glavni_izlaz

   b unutrasnja_petlja
```

U ovoj subrutini je trenutni deljenik bio veći od delioca, tj. sadrži ga na tekućoj decimalnoj poziciji. U binarnom zapisu to znači da če na tekućoj decimalnoj poziciji biti 1. Naredbom `sfl` se u akumulatoru uradi jedan SL, ovo se uradi za `resenje`, koje sadrži izlomljeni rezultat i doda se vrednost 1. Dalje se takođe uradi SL nad `deljenik_tmp` i algoritam nastavlja dok se ne izračunaju sve decimale, ili dok `brojac` ne dostigne vrednost 7, tj osmi bit u binarnom zapisu izlomljenog dela količnika.

```
stepen_dva:
   lacc resenje
   sfl
   add #1
   sacl resenje
   lacc znak
   bgez preskakanje_negacije


glavni_izlaz:
   lacc znak
   bgez preskakanje_negacije
   lacc kolicnik, 8
   add resenje
   neg
   sacl resenje
   b izlaz_end

preskakanje_negacije:
   lacc kolicnik, 8
   add resenje
   sacl resenje

izlaz_end:
   lacc resenje
   sacl *+

   lacc brojac2
   add #1
   sacl brojac2
   b glavna_petlja
```

Ako se nije desio slučaj negativnog stepena dvojke, izlazi se na `glavni_izlaz`.

Ideja čuvanja rezultata se ogleda u podeli značenja, viša i niža dva broja[1], ukupne širine reči koju možemo štampati. Umesto da se kao rezultat prosto prikaže celobrojni količnik i ostatak, odlučeno je da se viša dva heksidecimalna broja ispisa iskoriste za celobrojni deo količnika, a niža dva za izlomljeni deo koji je dobijen prethodnim algoritmom. Ovo znači da je rezultat prikazan u *fixed point* aritmetici, a decimalna tačka bi se nalazila na sredini ispisane reči.

Gledano u decimalnu tačke, sa desne strane idu negativni stepeni dvojke koji se smanjujum, a sa leve idu pozitivni koji se povećavaju.

Opravdanje* za dodeljivanje iste širine reči za rezultat izlomljenog i decimalnog dela leži u činjenici da su elementi inverznih matrica sa pozitivnim celobrojnim vrednostima do 31 u najvećem slučaju veoma male vrednosti, između 0 i 1. Stoga je potrebno dodeliti veću preciznost izlomljenom delu nego celobrojnom. Time rečeno, trebalo bi uzeti i veću preciznost za izlomljeni deo nego što je u ovom primeru. Međutim, demonstrativno je uzeta simetrična preciznost.

Negacija se vrši u drugom komplementu, a čuvanje rezultata rađenjem SL celobrojnog količnika za osam bita i sabiranjem sa izlomljenim delom. Trenutni količnik se čuva u * koja pokazuje na trenutnu poziciju registra u iteratoru.

[1] Rezultat se ispisuje sa četiri heksidecimalna broja, to je reč od 16 bitova, ovom podelom bi nižih osam bitova bilo iskorišćeno za čuvanje izlomljenog dela a viših 8 za čuvanje količnika.

```
;ispis glavnog rezulata
```

Ispisivanje glavnog rezultata u izlazni `qfile` se vrši ispisivanjem prvog reda, sada invertovane, matrice. Zatim drugog, pa trećeg, sve u jednoj koloni.

Naredbom `qfile` vraćamo pokazivače registara nazad na potreban redni vektor u memoriji. Ispis se vrši naredbom `out *, 8003h`.

```
ispisivanje_matrice:

   lar AR0, #red1
   mar *, AR0
   lacc *
   out *+,8003h
   lacc *
   out *+,8003h
   lacc *
   out *,8003h
   rpt #1
   mar*-

   lar AR1, #red2
   mar *, AR1
   lacc *
   out *+,8003h
   lacc *
   out *+,8003h
   lacc *
   out *,8003h
   rpt #1
   mar*-

   lar AR2, #red3
   mar *, AR2
   lacc *
   out *+,8003h
   lacc *
   out *+,8003h
   lacc *
   out *,8003h
   rpt #1
   mar*-
```

Analitičkim nalaženjem inverzne matrice dobija se sledeći rezultat

$$\mathbf{A}^{-1} = \begin{bmatrix} 0.0970 & 0.0550 & -0.1185 \\ 0.0075 & -0.0438 & 0.0774 \\ -0.0821 & 0.1073 & 0.0233 \end{bmatrix}$$

Izlaz u `qfile` je

```
0018
000e
ffe2
0001
fff5
0013
ffeb
001b
0005
```

Što je, nakon primene objašenjene logike ispisa rezultata i, po potrebi, drugog komplementa ekvivalentno matrici

$$\mathbf{A}^{-1} = \begin{bmatrix} 0.09375 & 0.0546875 & -0.1171875 \\ 0.0078125 & -0.04296875 & 0.07421875 \\ -0.08203125 & 0.10546875 & 0.01953125 \end{bmatrix}$$

Dobijena preciznost je relativno zadovoljavajuća, ali moguće je proširiti je uzimanjem u obzir opravdanja pod $\star$.

```
ako_je_det_0:
   END
```