Poročilo za dodatno nalogo pri OR

Andrej Sušnik

January 7, 2022

Contents

1 Izvedba vseh ukazov

Preden sem se lotil pisanja kakšnega bolj zahtevnega programa ali dodajanja dodatnih naprav, sem najprej realiziral vse ukaze, ki jih podpira zbirnik. Predstavil bom par bolj zanimivih ukazov.

1.1 Ukaz "subi Rd, Rs, immed"

Ukaz subi od vredosti registra Rs odšteje takojšnjo vrednost in jo shrani v Rd.

```
# subi Rd,Rs,immed
17: addrsel=pc imload=1
    aluop=sub  op2sel=immed dwrite=1  regsrc=aluout, goto pcincr
```

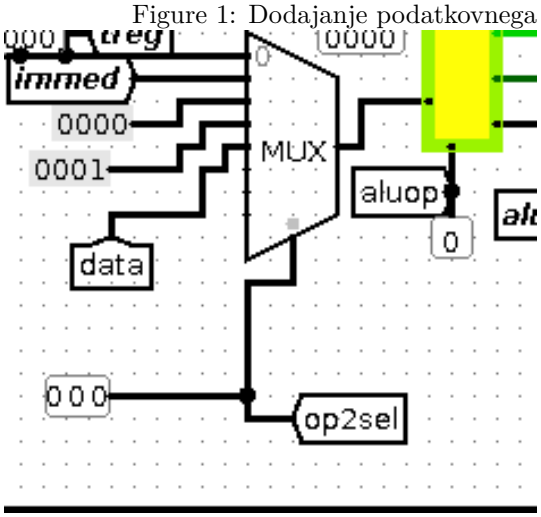
Ukaz	Fetch	Impl	Post op	$\sum$
subi Rd, Rs, Immed	2	2	1	5

Trenutna implementacija ukaza potrebuje 5 urinih period za izvedbo. Zdelo se mi je, da bi se dalo to optimizirati, če ne bi bilo treba nalagati takojšnje vrednosti v register za takojšnje vrednost. To je mogoče storiti saj je med izvajanjem prvega dela mikroukaza takojšnja vrednost na podatkovnem vodilu. Torej je treba dodati povezavo med ALE in podatkovnim vodilom. Da to lahko storimo pa moramo kontrolni signal op2sel povečati za 1 bit. V datoteki micro_assembler.pl moramo dodati opcijo 'databus' v op2sel in vse ostale zamakniti za 1 bit.

```
'op2sel=treg' =>    0 << 4,
'op2sel=immed' =>   1 << 4,
'op2sel=const0' =>  2 << 4,
'op2sel=const1' =>  3 << 4,
'op2sel=databus' => 4 << 4,
```

Nato še povežemo podatkovno vodilo na multiplexer za izbiro drugega operanda in multiplexer-ju povečamo število kontrolnih bitov na 3, kar nam omogoča 8 različna opcij za drugi operand.

Figure 1: Dodajanje podatkovnega vodila na multiplexer



Po teh spremembah lahko mikro ukaz zapišemo tako. In s tem prihranimo eno urino periodo.

```
# subi Rd,Rs,immed
17: aluop=sub  op2sel=databus dwrite=1  regsrc=aluout, goto pcincr
```

Ukaz	Fetch	Impl	Post op	$\sum$
subi Rd, Rs, Immed	2	1	1	4

1.2 Ukaz "push Rd" in "pop Rd"

Ukaza push in pop uporabljamo za delo s skladom. Za kazalec na sklad uporabljamo register R7.

```
# push Rd
68: aluop=sub op2sel=const1 regsrc=aluout swrite=1
    addrsel=sreg datawrite=1 datasel=dreg, goto fetch
# pop Rd
69: addrsel=sreg regsrc=databus dwrite=1
    aluop=add op2sel=const1 regsrc=aluout swrite=1, goto fetch
```

Pri ukazu push najprej kazalec na sklad zmanjšamo za 1, nato pa shranimo vrednost iz Rd na pomnilniško mesto, ki ga kaže kazalec na sklad. Pri ukazu pop najprej vrednost iz pomnilnika kamor kaže kazalec na sklad shranimo v register Rd, in nato kazalec na sklad povečamo za 1.

Ukaz	Fetch	Impl	Post op	$\sum$
push Rd	2	2	0	4
pop Rs	2	2	0	4

Oba ukaza potrebujeta za izvedbo 4 urine periode.

1.3 Ukaz "jge Rs, Rt, immed"

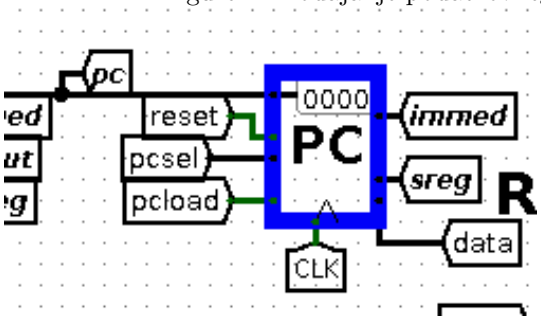
Ukaz jge uporabljamo za pogojne skoke. Skocimo ce ni prizgana zastavica N.

```
# jge Rs, Rt, immed
38: addrsel=pc imload=1
    aluop=sub op2sel=treg, if n then pcincr else jump
```

Ukaz	Fetch	Impl	Post op	$\sum$
jge Rs,Rt,immed	2	2	1	5

Pri implementaciji skocnih ukazov sem ugotovil, da podobno kot pri aritmeticnih ukazi z takojsnjim operandom, takojsnejega operanda ne potrebujemo shranati v register, ker ga lahko preberemo z podatkovnega vodila. Kontrolntrolni signal pcsel je bilo treba iz dveh bitov povecati na 3 bite in do enote programskega stevca pripeljati podatkovno vodilo.

Figure 2: Dodajanje podatkovnega vodila PC enoto



2 Dodajanje direktiv assemblerju

Da bi bil assembler bolj podoben armu sem v assembler dodal možnost shranjevanja podatkov v ram-u. Del programa kjer se nahajajo podatki označimo z ukazom .data. Ukaz se mora vedno pojaviti za našim programom.

Primer uporabe: direktiv:

```
inf: jmp inf


.data
TAB: .word 1000 10 15
LEN: .word 3
NIZ: .ascii a n d r e j
```

2.1 Direktiva .word

Najprej moramo v listo možnih ukazov dodati:

```
  '.word' => [ 0, 'w' ],
```

Nato pa moramo dodati še logiko v while, ki procesira datoteko

```
  if ($atype eq 'w') {
    my @args = split(' ', $arg);

    foreach my $ar (@args) {
      $Mcode[$PC] = ($ar =~ m{^0x}) ? hex($ar) : $ar;
      $PC++;
    }

    $PC--;
  }
```

Implementacija dovoljuje dekleracijo ene pomnilniške besede ali pa tudi tabele.

Testni program, ki sešteje vrednosti v tabeli:

```
main: li r0, TAB # store address of TAB into register r0
      li r1, 0 # use r1 as a counter
      li r2, 0 # store sum of the array in r2
      lw r5, LEN # store len in r5
loop: lwri r3, r0, r1
      add r2, r2, r3
      inc r1
      bne r1, r5, loop


inf: jmp inf


.data
TAB: .word 1000 10 15
LEN: .word 3
```