1. Domača naloga

MiMo

Mentor: Avtor:

Robert Rozman Jožef Poklukar

## Mikroukaz SW Rd, Immed

## addrsel=pc irload=1

Irload omogoči branje ukaza iz pomnilnika v ukazni register. Iz pomnilnika prebere ukaz iz naslova, ki ga določimo z multiplekserjem addrsel, v tem primeru prebere programski števec.

## pcload=1 pcsel=pc, opcode\_jump

Pcload omogoči programski števec, da se posodobi, pcsel pa določi naslednjo vsebino, torej v tem primeru prišteje pc+1. Nato opcode\_jump skoči na naslov prebranega ukaza.

## addrsel=pc imload=1

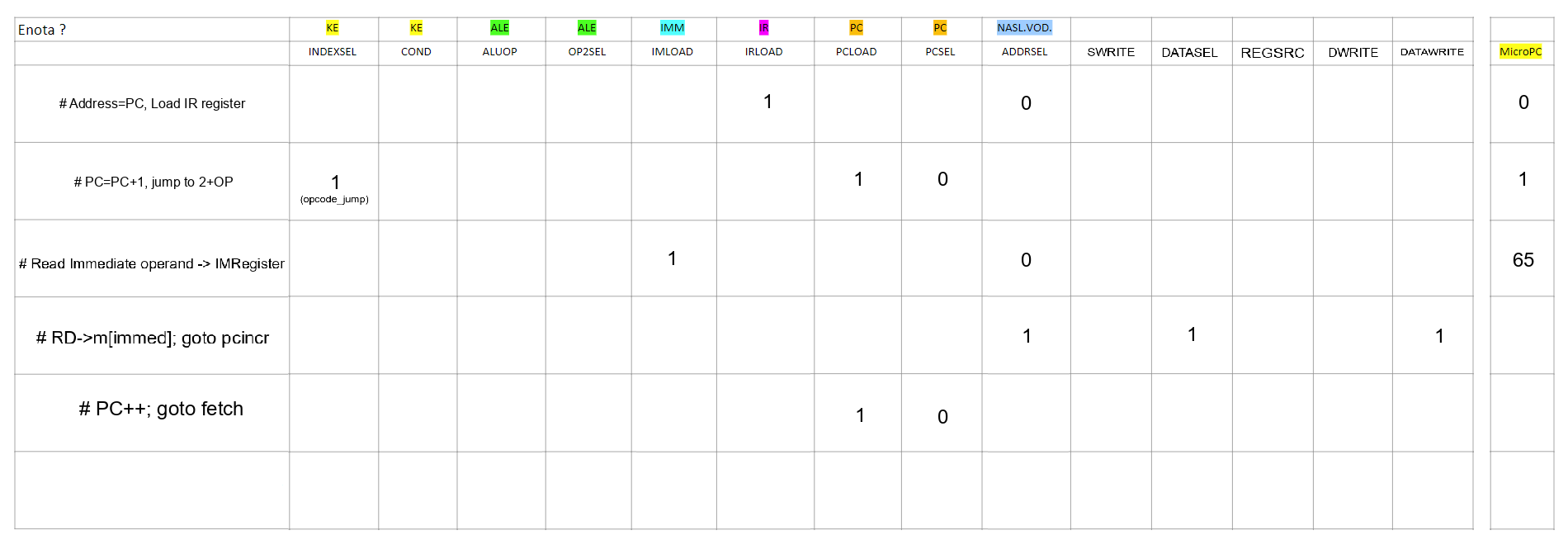
Ponovno preberemo iz pomnilnika, le da tokrat preberemo takojšnji operand. Zapišemo ga v ImmReg, ki smo ga omogočili z imload=1.

## addrsel=immed datawrite=1 datasel=dreg, goto pcincr

Da željeni register shranimo, najprej nastavimo addsel multiplekser na immed, kar pomeni, da se bo Rd shranil na naslov, ki smo ga vnesli kot takojšnji operand. Pomnilniku moramo sedaj še povedati, da bo pisal in ne bral, ter kaj mora zapisati. Da bo pisal povemo z datawrite=1, kaj pa bo pisal pa določimo z multiplekserjem datasel. V ukazni register določili Rd, torej datasel nastavimo na dreg.

V tem trenutku naš programski števec kaže na naslov, od koder smo prebrali takojšnji operand, zato ga moramo najprej povečati še za 1. Zato skočimo na podprogram z ukazom goto pcincr.

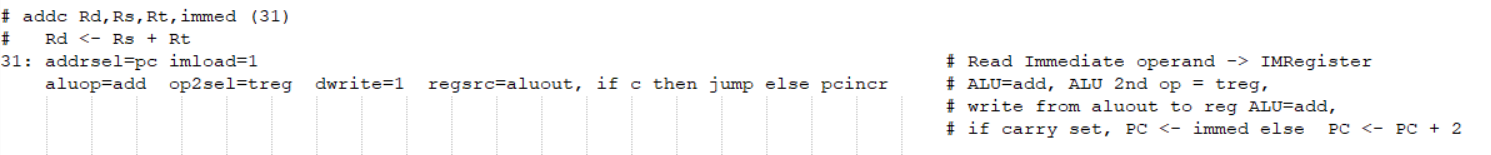
## pcload=1 pcsel=pc, goto fetch

Kot poprej prištejemo števcu 1 in tokrat skočimo nazaj na fetch, kjer se bo prebral naslednji ukaz.

# Dodajanje mikroukazov

## addc Rd, Rs, Rt, immed

Ideja ukaza je, da če pride do prenosa pri seštevanje Rs in Rt registrov naj program skoči na naslov immed.

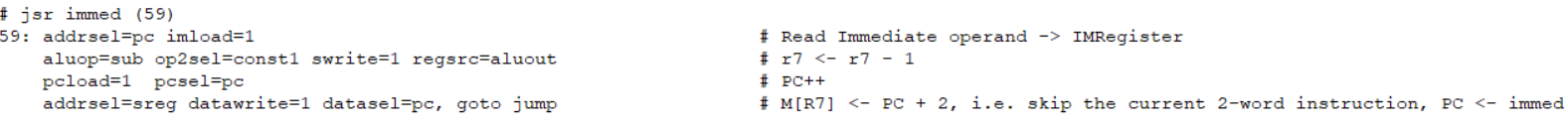
Realiziral sem ga tako, ga tako da sem najprej naložil immed register. Nato sem seštel Rs in Rt in rezultat shranil v register Rd. Ob seštevanju se je lahko vključila zastavica c, ki sem jo preveril z if stavkom. Če je zastavica vključena program skoči na podprogram jump, ki naloži v programski števec immed register, sicer se pa kliče pcincr, ki naredi PC+1 ter se vrne na fetch.



## jsr immed

Ukaz mora shraniti naslov naslednjega zaporednega ukaza na sklad v pomnilnik, ter skočiti na podprogram, katerega naslov je podan v immed.

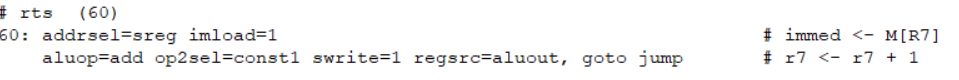
Začel sem z nalaganjem immed operanda v immReg. Nato sem r7, ki je kazalec sklada, odštel 1, torej da kaže na prosto mesto v pomnilniku. Povečal sem PC na PC+1, da kaže na naslednji ukaz za jsr, ter nato shranil ta naslov na sklad v pomnilnik. Izvajanje ukaza sem zaključil s skokom na naslov v ImmReg.



## rts

Naredi obratno kot jsr, torej vzame naslov iz sklada, ter ga naloži v PC.

Najprej sem prebral vrednost iz sklada in jo shranil v ImmReg. Števec sklada sem nato povečal za 1, da kaže na naslednjo vrednost. Tudi ta se zaključi s skokom na naslov, ki je v ImmReg.



## neg Rs

Ukaz negira register, oziroma mu spremeni predznak.

Za to sem uporabil logiko dvojiškega komplementa. Registru sem najprej negiral vse bite z not funkcijo v ALE. Nato sem Rs prištel še 1, ter ga shranil nazaj od koder sem ga bral. Tokrat PC ni treba povečevati za 2, ker ni takojšnjega operanda, torej sem takoj klical fetch.

Slika, ki vsebuje besede besedilo, pisava, bela, vrstica

Opis je samodejno ustvarjen

# Testni program

* li r7, 2048

V register r7 naložim naslov od koder navzdol se bodo shranjevale vrednosti na skladu.

* li r0, 63

Vrednost 63 naložim v r0, da imam vrednost s katero bom opravljal

* inc r0

Povečam r0 za, torej mora biti sedaj vrednost 64.

* muli r1, r0, 2

Pomnožim r0 z 2 ter rezultat shranim v r1. Vrednost r1 mora bti 128.

* div r0, r1, r0

R1 delim z r0, ter rezultat, ki bi moral priti enak 2 shranim v r0.

* jsr reset

Kliče podprogram, ki resetira vse registre na 0 razen r7

* li r0, 3

Ponovno naložim neko vrednost v register r0.

* move r1, r0

Kopira vrednost r0 v r1.

* neg r1

Negira r1, torej mora imeti sedaj vrednost -3.

* bgt r0, r1, asc

blt r1, r0, desc

Če je r0 večji od r1 potem gre v podprogram asc, če je manjši pa v desc.

* inf: jnez r7, inf

Neskončna zanka za konec programa.

* asc: dec r1

beq r1, r0, inf

br asc

Ta zanka pomanjšuje r1 dokler ni r1 enak r0. Desc podprogram naredi obratno, torej r1 povečuje do r0.

V spodnji tabeli je prikazano število period, ki jih posamezen ukaz porabi za svoje izvajanje. Na koncu tabele pa je tudi seštevek vseh skupaj, če bi se vsak ukaz izvedel enkrat.

Slika, ki vsebuje besede besedilo, številka, posnetek zaslona, pisava

Opis je samodejno ustvarjen

# Vključitev V/I

## Spreminjanje vezja

Če želimo uporabiti vhodno – izhodne naprave, moramo prebrati zadnja dva bita nasovnega vodila. ADDR DEC. to naredi in glede na njiju vklopi eno izmed 4 možnih V/I naprav. Do sedaj je bil trdoožičen, da je vedno vklopil pomnilnik, kar sem odstranil in dodal logiko za V/I.

Slika, ki vsebuje besede besedilo, pisava, vrstica, posnetek zaslona

Opis je samodejno ustvarjen

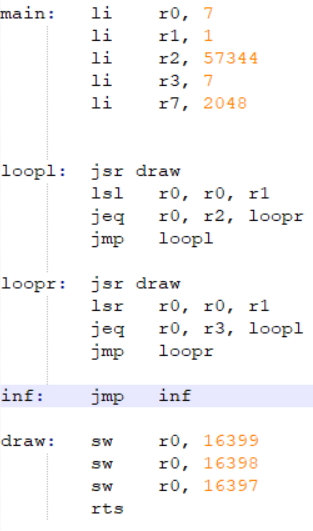
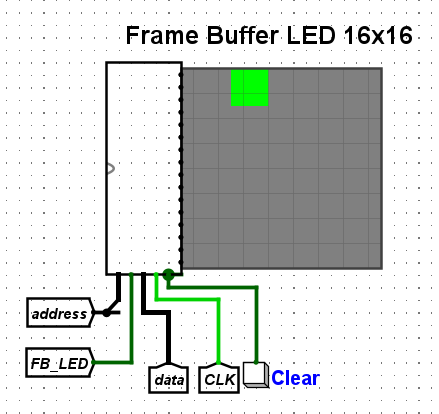
Spodnja slika prikazuje logiko, ki prebere zadnja dva bita Naslovnega vodil in ju kot kontrolna bita usmeri v multiplekser. Ta logično enico pošlje eni izmed naprav in jo s tem vklopi.

Slika, ki vsebuje besede diagram, posnetek zaslona, vrstica, pisava

Opis je samodejno ustvarjen

## Testni program

Napisal sem testni program, ki kvadratek 3x3 odbije levo in desno po izhodnem zaslonu. Program pošilja vrednosti izhodni napravi, ki jih ta prikaže tako, da kjer je 1 se prižge zeleno sicer je sivo. V programu sem vrednosti registrov s shift operacijo premikal levo in desno ter rezultate pošiljal v Frame Buffer.



# Dodajanje lastne izhodne naprave

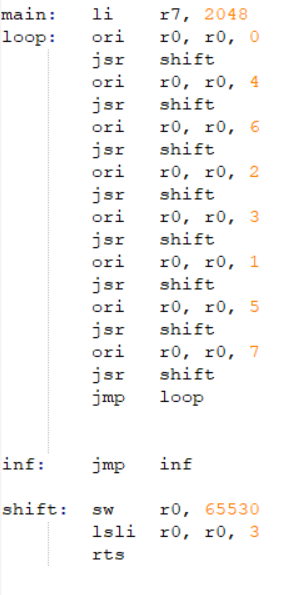
## Potrebna opravila za vključitev dodatne V/I naprave:

* Dodati je potrebno kanal iz ADDR DEC., kjer do biti na 11
* Naprava mora imeti sledeče vhode:
  + Data (podatki, ki jih bo uporabila / shranila)
  + CLK (ura, da bo delovala le ob urnih periodah)
  + Enabled (ADDR DEC. bo moral biti na 11, da bo aktivirana)
  + Clear (za ponastavitev naprave)
* V mojem primeru bo Izhodna naprava RGB LED luči. Za preprosto izvedbo sem naredil brez naslova. Lahko bi naredili za vsak naslov svoj komplet lučk.
* Za shranjevanje podatkov bom porabil D celico

## Postopek:

* Najprej sem dodal novo vezje.
* V vezju sem dodal 4 vhode (data, store, clock, reset), ki so vsi vezni na D celico.
* Izhod pomnilne celice, ki se spremeni le ko je store na 1, sem s splitter-jem razbil na posamezne bite in jih vezal po tri na vsako RGB LED luč.
* Zadnji bit ki je ostal sem vezal na navadno LED luč.
* Da se bo luči videlo na glavnem vezju sem za izhode vezal po tri bite. Te bite sem nato na glavnem vezju ponovno vezal la luči.
* Za vhode na glavnem vezju sem povezal kanal data iz podatkovnega vodila na vhod data, na clock kanal CLK, na clear sem vezal gumb Clear\_RGB, ter na store na novo ustvarjen kanal RGBSel, ki prihaja iz ADDR DEC.

## Testni program

Za testni program sem spisal premikanje barv od najtemnejše do najstvetlejše:

