Organizacija računalnikov (OR) – 1.DN

1.

Jasno zapišite zaporedje mikroukazov, ki se izvedejo pri izvedbi strojnega ukaza "LI Rd,Immed" (primer: "LI r3, 150").

addrsel=pc dwrite=1 regsrc=databus, goto pcincr

- addrsel=pc: Ta mikroukaz nastavi izbirnik naslova (address selector) na vrednost programskega števca (PC). To pomeni, da se bo naslednji naslov, ki bo uporabljen, vzet iz PC.
- dwrite=1: Ta mikroukaz nastavi bit za pisanje na 1. To pomeni, da bo naslednji podatek, ki bo prebran iz databus-a, shranjen v registru, ki ga določa naslednji mikroukaz.
- regsrc=databus: Ta mikroukaz nastavi vir registra na databus. To pomeni, da bo naslednji podatek, ki bo shranjen v registru, prebran iz databus-a.
- goto pcincr: Ta mikroukaz nastavi PC na naslednji naslov v programu. To pomeni, da se bo procesor premaknil na naslednji ukaz v programu.

V primeru "LI r3, 150" se bo torej vrednost 150 shranila v registrsko spremenljivko r3, števec programa pa se bo povečal za ena.

1 Preberemo vrednost Immed in jo shranimo v spremenljivko v pomnilniku.

- Vrednost, navedena po ukazu LI (v našem primeru 150), se prebere in shrani v spremeljivko v pomnilniku. Stanje podatkovne enote se ne spremeni

2 Preberemo vrednost Rd in jo shranimo v spremenljivko v pomnilniku.

- Vrednost, navedena kot ciljni registerski spremeljivki po ukazu LI (v našem primeru je to “r3”) se prebere in shrani v spremeljivko v pomnilniku. Stanje podatkovne enote se ne spremeni.

3 Shranimo vrednost Immed v registrsko spremenljivko Rd.

- Vrednost Immed, ki je bila prej shranjena v spremeljivko v pomnilniku prebere in shrani v registersko spremelnjivko Rd (v našem primeru je to “r3”). Stanje podatkovne enote se spremeni, saj se v registerski spremeljivki r3 shrani nova vrednost

4 Povečamo programski števec (PC).

- Vrednost programskega števca se poveča za ena. To je potrebno, da se lahko izvede naslednji ukaz programa. Stanje podatkovne enote se spremeni, saj se spremeni programski števec.

addi Rd,Rs,immed (16)

Rd <- Rs + immed PC <- PC + 2

addi (add immediate) ukazuje procesorju, da se operand immed (takojšnje število) doda k operandu Rs in rezultat shrani v register Rd.

Nato povečamo programski števec (PC) za 2, da se procesor premakne na naslednji ukaz.

subi Rd,Rs,immed (17)

Rd <- Rs - immed PC <- PC + 2

#subi (subtract immediate) ukazuje procesorju, da se operand immed odšteje od operanda Rs in rezultat shrani v register Rd.

Nato povečamo programski števec (PC) za 2, da se procesor premakne na naslednji ukaz.

muli Rd,Rs,immed (18)

Rd <- Rs * immed PC <- PC + 2

#muli (multiply immediate) ukazuje procesorju, da se operand Rs pomnoži z operandom immed in rezultat shrani v register Rd.

Nato povečamo programski števec (PC) za 2, da se procesor premakne na naslednji ukaz.

divi Rd,Rs,immed (19)

Rd <- Rs / immed PC <- PC + 2

#divi (divide immediate) ukazuje procesorju, da se operand Rs deli z operandom immed in rezultat shrani v register Rd.

Nato povečamo programski števec (PC) za 2, da se procesor premakne na naslednji ukaz.

remi Rd,Rs,immed (20)

Rd <- Rs % immed PC <- PC + 2

#remi (remainder immediate) ukazuje procesorju, da izračuna ostanek pri deljenju registra Rs z immed in rezultat shrani v register Rd.

Nato povečamo programski števec (PC) za 2, da se procesor premakne na naslednji ukaz v programu.

andi Rd,Rs,immed (21)

Rd <- Rs AND immed PC <- PC + 2

#andi (and immediate) ukazuje procesorju, da operand Rs in operand immed poveže po logičnem AND in rezultat shrani v register Rd.

Nato povečamo programski števec (PC) za 2, da se procesor premakne na naslednji ukaz v programu.

ori Rd,Rs,immed (22)

Rd <- Rs OR immed PC <- PC + 2

#ori (or immediate) ukazuje procesorju, da operand Rs in operand immed poveže po logičnem OR in rezultat shrani v register Rd.

Nato povečamo programski števec (PC) za 2, da se procesor premakne na naslednji ukaz v programu.

3.

```
main: li r0, 5 # Nastavi vrednost registra r0 na 5
      addi r1, r0, 4 # Ukaz doda 4 k vrednosti registra r0 in rezultat shrani v
register r1
      subi r2, r1, 1 # Ukaz odšteje 1 od vrednosti registra r1 in rezultat shrani v
register r2
      divi r3, r2, 2 # Ukaz deli vrednost registra r2 z 2 in rezultat shrani v
register r3
      muli r4, r3, 3 # Ukaz pomnoži vrednost registra r3 z 3 in rezultat shrani v
register r4
      remi r5, r4, 5 # 5 Ukaz izračuna ostanek pri deljenju registra r4 z 5 in
rezultat shrani v register r5
```

Ukaz	Število urninih period
li	4
addi	5
subi	5
divi	5
muli	5
remi	5

Skupno število trajanja urinih period: **29**

5.

Za uporabo dodatne izhodne naprave sem v address decoder mogel dodati še en izhod (poimenoval sem ga RGB_LED).

Ta izhod sem nato v glavnem vezju peljal na novi tunnel "RGB_LED".

Nato sem dodal D Register na katerega sem pripeljal 3 tunnele (data, RGB_LED in CLK) ter jih ustrezno povezal na register.

Na izhodu registra (Q) sem dodal 16bit splitter, preko katerega izloščim prve 3 bite, ki sem jih nato povezal na RGB LED diodo.

V program (test_program_IO_v05.s) sem dodal naslednji ukaz "sw r1, 49152", kateri poskrbi,

da vrednost registra r1 shranimo v pomnilniški naslov 49152 (1100000...). Končni rezultat je delujoča RGB LED dioda.

Vse spremenjene datoteke ter video delovanja sem naložil na github:
https://github.com/zigapovhe/or_dn1