7. Vaja: mikroprocesor

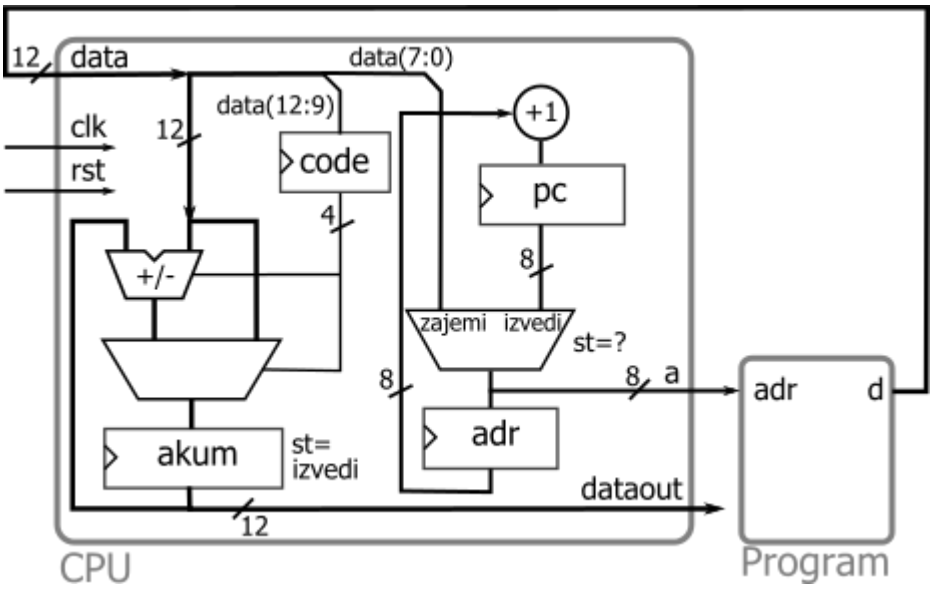
Naredili bomo model 12-bitnega mikroprocesorja (CPU) s programskim pomnilnikom (Program), v katerem so shranjeni ukazi in podatki.
Napisali bomo preprost program za procesor z uporabo ukazov v zbirniku:

LDA M – *naloži vrednost iz pomnilniške lokacije M v akumulator*

ADD M – *prištej k akumulatorju vrednost iz pomnilniške lokacije M*

JMP A – *skok na novo pomnilniško lokacijo z naslovom A*

12-bitni mikroprocesor s pomnilnikom



V opisu mikroprocesorja in programa bomo uporabljali ukazne kode v obliki simboličnih konstant, ki so definirane v knjižnici procpak.vhd.

```
subtype  koda is unsigned(3 downto 0);
constant lda: koda := "0001"; -- nalozi iz pomnilnika v akumulator
constant sta: koda := "0010"; -- shrani iz akumulatorja v pomnilnik
constant jmp: koda := "0100"; -- skok na novo pomnilnisko lokacijo
constant add: koda := "1000"; -- pristej vrednost iz pomnilnika
constant sub: koda := "1001"; -- odstej vrednost iz pomnilnika
```

Pomnilnik

V datoteki Program.vhd je opis pomnilnika vrste ROM v katerem je kratek program. Pomnilnik vsebuje 12-bitne mikroprocesorske ukaze in podatke. Ukaz je sestavljen iz 4-bitne kode, ki jo predstavljajo zgornji štiri biti, spodnjih osem bitov pa je naslov v pomnilniku. V pomnilnik smo zapisali kratek program iz treh ukazov: najprej naložimo v akumulator vrednost (005) iz pomnilniške lokacije 03, nato prištejemo isto vrednost, tretji ukaz pa je brezpogojni skok na drugi naslov (01).

```
type memory is array(0 to 3) of unsigned(11 downto 0);
signal ram : memory := (
             lda & x"03",
             add & x"03",
             jmp & x"01",
             x"005" );
```

Centralna procesna enota: CPU

Naredi model centralne procesne enote z akumulatorjem, ki ima na vhodu uro in reset signal in 12-bitni podatek **data**, na izhodu pa 8-bitni naslov **a** in 12-bitni signal **dataout**.

a. Definiraj priključke procesorja in dodaj na vrhu opisa vezja knjižnico z ukaznimi kodami:

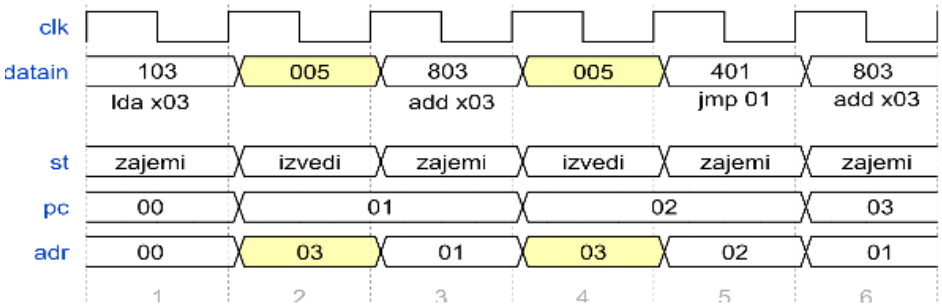
Circuit type: ☑ Sequential circuit

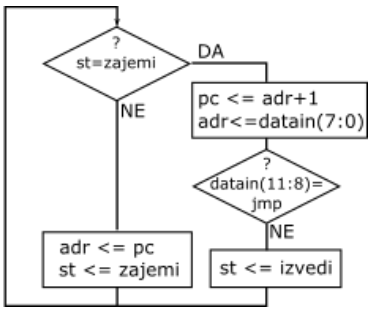
Name	In/Out	Type	MSB	LSB
rst	in	std_logic		0
data	in	unsigned	11	0
a	out	unsigned	7	0
dataout	out	unsigned	11	0

```
library work;
use work.procpak.all;
```

Deklariraj tudi notranje signale za registre tipa unsigned: 4-bitni **code**, 8-bitni **pc** in **adr**, 12-bitni **akum**

b. Naredi sinhroni proces, v katerem se izmenično izmenjujeta stanji *zajemi* in *izvedi*. V tem procesu opiši tudi programski števec in naslovni register, kot prikazuje diagram:





c. Opiši delovanje signala rst, ki postavi procesor v stanje zajemi in registre na 0. Določi še logiko za signal **a**, ki predstavlja naslov pomnilnika. Ob resetu ali stanju *izvedi* naj bo enak programskemu števcu **pc**, sicer pa vodnemu podatku **data** (spodnjih 8 bitov). S simulacijo preveri prehajanje stanj procesorja in časovni potek programskega števca in naslovnega registra.

d. Podatkovna pot procesorja vsebuje 4-bitni register **code** za ukazno kodo in 12-bitni akumulator **akum.** V stanju *zajemi* naj se zgornji 4 biti iz vodila **data** shranijo v register **code.** Glede na ukazno kodo v registru **code** se v stanju *izvedi* izvedejo operacije, ki shranijo rezultat v akumulator:

- lda: akum <= data
- add: akum <= akum + data
- sub: akum <= akum - data