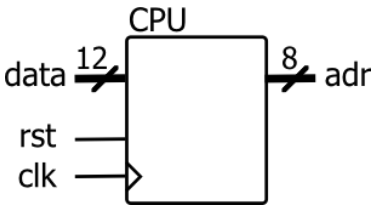
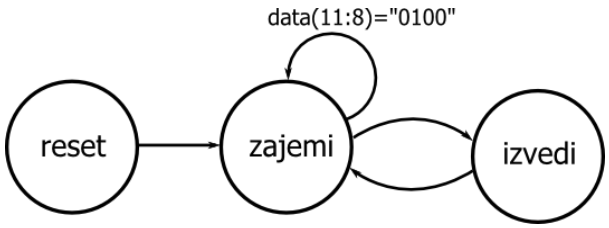
Naredili bomo model 12-bitne centralno procesne enote (CPU) z akumulatorjem, ki zna izvajati preproste strojne ukaze iz programskega pomnilnika. Strojni ukazi so 12-bitne kombinacije: zgornji 4 biti določajo vrsto ukaza, spodnjih 8 pa določa pomnilniški naslov. Za začetek bomo naredili enoto s tremi strojnimi ukazi:

"0001nnnnnnnn": naloži vrednost s pomnilniške lokacije nnnnnnnn v akumulator
"0100nnnnnnnn": skoči na naslov nnnnnnnn
"1000nnnnnnnn": prištej k akumulatorju vrednost iz pomnilniške lokacije nnnnnnnn

7 Centralno procesna enota (CPU)



Začetni model centralno procesne enote naj ima na vhodu uro in reset ter 12-bitno podatkovno vodilo **data,** na izhodu pa 8-bitno naslovno vodilo **adr**, na katerega bo kasneje priključen pomnilnik s programom. Enota začne po resetu zajemati in izvajati ukaze iz pomnilnika. Korake izvajanja ukazov krmili sekvenčni stroj (avtomat) s tremi stanji:



a. Naredi vezje cpu.vhd z definicijami zunanjih priključkov in signali, med katerimi je nov podatkovni tip in notranji signal **st** za sekvenčni stroj:

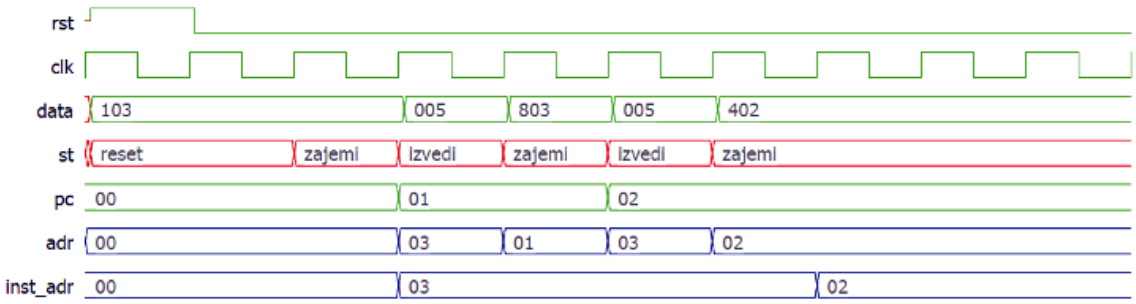
```
type stanje is (reset, zajemi, izvedi);
signal st: stanje;
```

Deklariraj še dva notranja signala: 8-bitni programski števec **(pc)** in 8-bitni ukazni naslov (**inst_adr**), ki sta tipa unsigned z začetno vrednostjo 0.

b. Naredi sinhroni proces, ki opisuje prehajanje stanj avtomata s stavkom case. Avtomat naj gre v stanje *reset* ob rst='1', nato pa naj ob fronti ure izmenjuje stanji *zajemi* in *izvedi.*
Poleg prehodov stanj določi še vrednosti notranjih registrov:
   - v stanju *reset* naj se programski števec (**pc**) postavi na x"00"
   - v stanju *zajemi* naj se programski števec poveča za 1, razen ob ukazu "0100", ko naj se postavi na vrednost data(7 downto 0)
   - v stanju *zajemi* na se shrani ukazni naslov: inst_adr <= data(7 downto 0);

c. Izven procesa opiši še kombinacijko logiko, ki določa signal **adr** (uporabi stavek **when...else**):
   - x"00" ob st=reset,
   - **pc** ob st=zajemi,
   - **inst_adr** ob st=izvedi

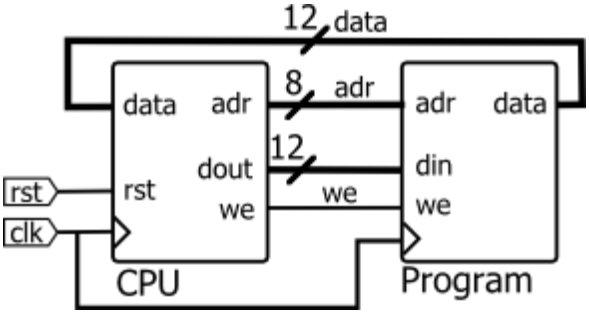
Preizkusi delovanje krmilnega dela CPU na simulaciji s testno strukturo testcpu.vhd, ki ima nastavljene vhode, kot prikazuje diagram:



d. Podatkovna pot procesorja vsebuje 12-bitni akumulator **akum** v katerega se shranjujejo rezultati operacij in 4-bitni register **inst_code** za ukazno kodo, ki ju deklariraj kot notranja signala tipa unsigned.
   - v stanju *zajemi* naj se zgornji 4 biti iz vodila **data** shranijo v register **inst_code**
   - v stanju *izvedi* naj se izvedejo operacije, ki shranijo rezultat v akumulator:
     - "0001": akum <= data
     - "1000": akum <= akum + data

Ponovno naredi simulacijo s testno strukturo in opazuj vrednost akumulatorja. Po izvedbi ukaza s kodo "103" se bo postavil na 5, po izvedbi naslednjega ukaza pa povečal na 10 (šestnajstiško X"00A").

8 Mikroprocesor s pomnilnikom



Mikroprocesor bomo zgradili iz komponent in lastne knjižnice. 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 jmp: koda := "0100"; -- skok na novo pomnilnisko lokacijo
constant add: koda := "1000"; -- pristej vrednost iz pomnilnika
```

a. Naloži datoteke procpak.vhd, proc.vhd in program.vhd in popravi **cpu.vhd** tako, da bo uporabljala simbolične konstante (lda, add in jmp) iz knjižnice. Knjižnico uporabimo tako, da jo vključimo na začetek opisa vezja s stavkom:

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

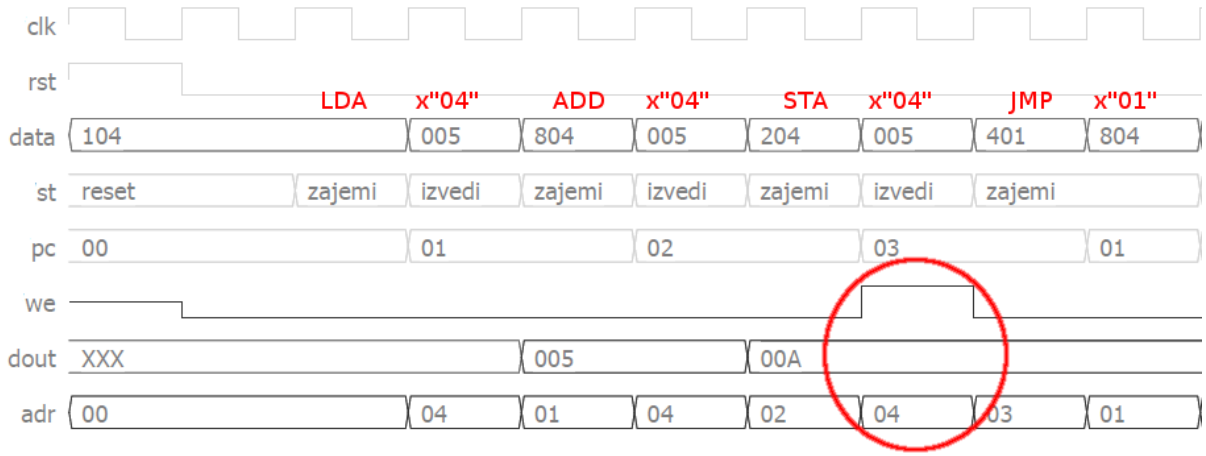
V datoteki **program.vhd** je opis pomnilnika vrste ROM v katerem je kratek program. Pomnilnik vsebuje 12-bitne mikroprocesorske ukaze in podatke. Ukaz je sestavljen iz 4-bitne kode, ki jo predstavljajo zgornji štiri biti, spodnjih osem bitov pa je naslov v pomnilniku. V pomnilnik smo zapisali kratek 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 255) of unsigned(11 downto 0);
signal ram : memory := (
 lda & x"03",
 add & x"03",
 jmp & x"01",
 x"005",
 others => x"000" );
```

Popravi model pomnilnika, tako da bo omogočal shranjevanje podatkov. Uporabi vhodna signala **din** in **we.**

- Dodaj logiko za pisanje v sinhroni proces. Kadar je signal **we='1',** se v pomnilnik **ram** na naslovu **adr** vpiše vhodni podatek iz **din**.

b. Med priključke mikroprocesorja dodaj 12-bitni izhod **dout**, ki naj bo enak akumulatorju, ter enobitni izhodni signal **we**. Signal **we** določa pisanje podatkov v pomnilnik in se mora postaviti na 1 v ciklu izvajanja ukaza STA, v vseh ostalih ciklih pa naj bo na '0'. Glej priloženo sliko simulacije.



c. Dodaj v program ukaz **sta** in preizkusi delovanje na simulatorju.

```
type memory is array(0 to 255) of unsigned(11 downto 0);
signal ram : memory := (
      lda & x"04",
      add & x"04",
      sta & x"04",
      jmp & x"01",
      x"005",
      others => x"000" );
```