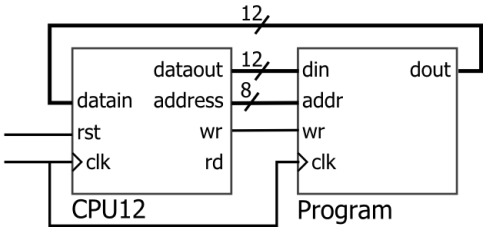
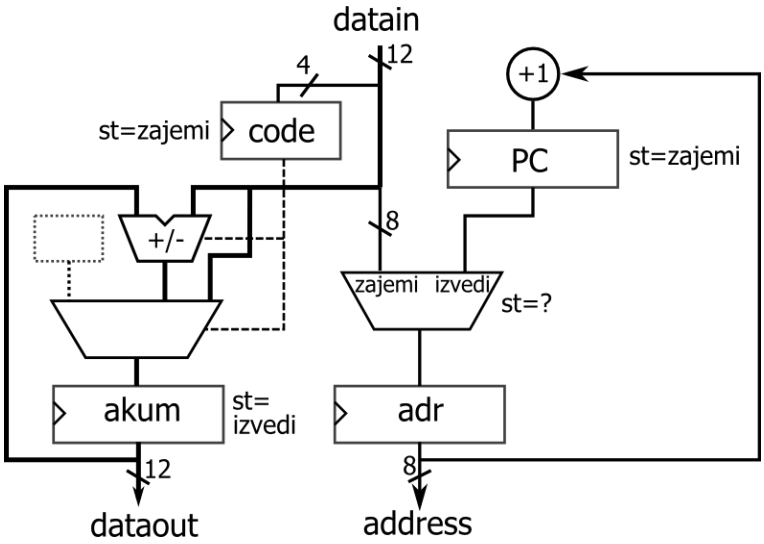
9. Vaja: 12-bitni procesor

Naredi model preprostega 12-bitnega procesorja, ki je zgrajen kot sekvenčni stroj z akumulatorjem (podobno kot procesor MCPU). Ukazi naj bodo 12-bitne binarne vrednosti: zgornji 4 biti so koda ukaza, spodnjih 8 bitov pa naslov v pomnilniku. V opisu vezja bomo uporabljali ukazne kode v obliki simboličnih konstant:

```
subtype  koda is std_logic_vector(3 downto 0);
constant  lda: koda := "0001";        -- naloži vrednost iz pomnilniške lokacije v akumulator
constant  sta: koda := "0010";        -- shrani iz akumulatorja v pomnilnik
constant  jmp: koda := "0100";        -- skok na novo pomnilniško lokacijo
constant  add: koda := "1000";        -- prištej k akumulatorju vrednost iz pomnilniške lokacije
constant  sub: koda := "1001";        -- odštej…
```





Vsak ukaz naj se opravi v dveh urnih ciklih: v prvem ciklu zajamemo kodo ukaza, v drugem pa jo izvedemo. Definirajmo nov podatkovni tip in signal st, ki shranjuje stanje. Ob resetu postavimo vse registre na 0 in stanje na zajemi.

Najprej bomo opisali podatkovno pot z akumulatorjem. V stanju zajemi se zgornji 4 biti datain shranijo v register code. V stanju izvedi pa se izvršijo tiste operacije, ki shranijo rezultat v akumulator.

```
if rst='1' then
  st <= zajemi;
  akum <= x"000";
  pc <= x"00";    adr <= x"00"; ...
elsif rising_edge(clk) then
    if st=zajemi then
      code <= datain(11 downto 8);
    else
      case code is
        when lda  =>
          akum <= unsigned(datain);
        when add  =>
          akum <= akum + unsigned(datain);
        ...
        when others  => null;
      end case;
    end if;
```

NOTRANJI SIGNALI

Deklaracija podatkovnega tipa za notranje stanje:

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

Deklaracija registrov, ki jim prištevamo vrednost:

```
signal  akum: unsigned(11 downto 0);
signal  adr, pc: unsigned(7 downto 0);
```

Deklaracija registra za ukazno kodo:

```
signal  code: std_logic_vector(3 downto 0);
```

Krmilni del skrbi za prehajanje stanj ter naslovne in krmilne signale za pomnilnik. Predstavili ga bomo v obliki algoritma, ki se veji glede na stanje:

- če je st=zajemi, bo v naslednjem ciklu:
  - st <= izvedi
  - pc <= adr+1; adr <= unsigned(datain(7 **downto** 0))
  - če je na datain ukaz jmp:
    - st <= zajemi
  - če je na datain ukaz lda: rd <= '1'
  - če je na datain ukaz sta: wr <= '1'
- če je st=izvedi, bo v naslednjem ciklu:
  - st <= zajemi
  - adr <= pc

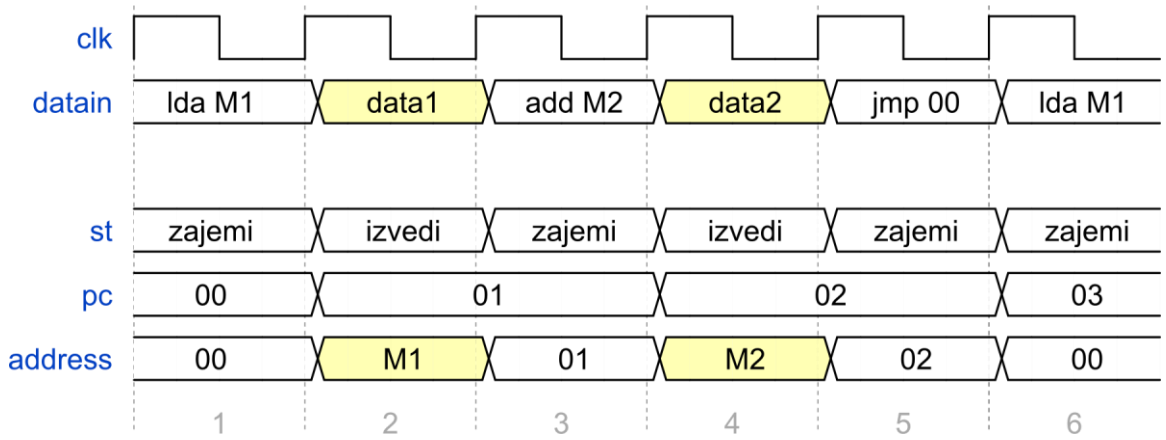
a) Naredi VHDL model procesorja CPU12 in preveri sintakso. Simbolične konstante ukazov naj bodo definirane v paketu procpak knjižnice **work**, ki ga vključi v opis vezja.

Pomnilnik bomo naredili v obliki statičnega pomnilnika z ločenim podatkovnim vhodom in izhodom ter sinhronim vpisom, ki je na voljo v programirljivih vezjih. Programsko kodo bomo pisali z uporabo konstant iz paketa procpak, zato je inicializacija pomnilnika podobna pravi zbirniški kodi:

```
type memory is array(0 to 3) of std_logic_vector(11 downto 0);

signal m : memory := (
   lda & x"03",
   add & x"03",
   jmp & x"00",
   x"005"
);
```

Poglejmo si potek izvajanja kratkega programa s tremi ukazi: najprej naloži v akumulator vrednost (data1) iz pomnilniške lokacije M1, nato prišteje vrednost (data2) iz lokacije M2, tretji ukaz pa je brezpogojni skok na začetek programa (naslov 00):



b) Naredi model statičnega pomnilnika v katerem bo programska koda za procesor. Uporabi sinhroni proces za vpis podatka iz vhoda din ob fronti ure in pogoju wr='1'. Branje podatkov naj bo asinhrono, na dout se pojavi vrednost, ki jo določa naslov adr. Naredi testno strukturo, ki naj povezuje procesor in pomnilnik ter določa časovni potek ure in signala rst. S testno strukturo preizkusi delovanje procesorja na simulatorju.