7. vaja: Mikroprocesor (1)

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

Pomnilnik

Mikroprocesor ima 12-bitni pomnilnik za podatke in programsko kodo. Uporabi datoteko ram.vhd z opisom pomnilnika s kratkim programom:

adr	data	pomen
0	0x103	ukaz LDA 03, naloži v akumulator podatek iz naslova 03
1	0x804	ukaz ADD 04, prištej akumulatorju podatek iz naslova 04
2	0x402	ukaz JMP 02, skoči na naslov 02
3	4	vrednost 4
4	5	vrednost 5

Pomnilnik definiramo kot tabelo:

```
type mem is array(0 to 4) of unsigned(11 downto 0);
signal rom: mem :=(x"103", x"804"...);


data <= rom(to_integer(adr));
```

Krmilna enota

Krmilna enota skrbi za osnovne korake izvajanja ukazov. Začetno stanje je S0, nato pa si izmenoma sledita stanji S1 in S2.

- S0, začetno stanje: postavi programski števec na 0 in pojdi na S1
- S1, zajemi: shrani ukaz iz pomnilnika in nastavi programski števec
- S2, izvedi: beri podatek iz pomnilnika in izvedi ukaz

Naredi krmilno enoto s spletnim orodjem, kjer so stanja določena kot konstante ali pa v jeziku VHDL, kjer stanja določimo z novim podatkovnim tipom.

spletno orodje

Če signalov s0,s1 in s2 ne deklariramo, jim orodje dodeli zaporedne vrednosti.

```
if (rst=0) s<=s0
else if (s=s0) s<=s1
else if (s=s1) s<=s2
...
```

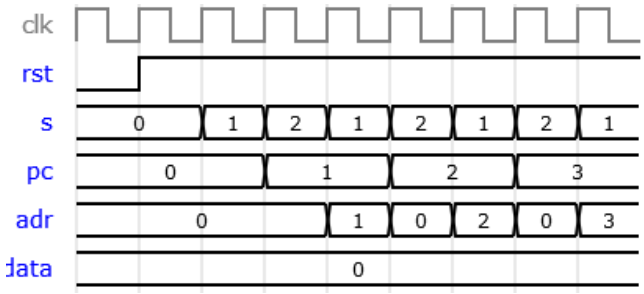
VHDL

Stanja določimo s podatkovnim tipom:

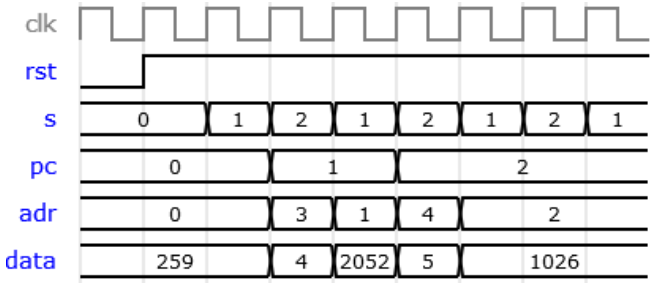
```
type stanje is (s0,s1,s2);
signal s: stanje;

process(clk)
begin
 if rising_edge(clk) then
  if rst = '0' then
     s <= s0;
  elsif s = s0 then
     s <= s1;
  ...
```

1. Naredi model krmilne enote z vhodno uro in enobitnim signalom **rst**, ki je aktiven ob 0. Dodaj še notranji 8-bitni števec **pc**, ki se ob resetu postavi na 0, stanje s1 pa naj ga poveča za 1. **Pazi**: programski števec se spreminja ob uri, zato se poveča ob spremembi iz s1 v s2 !

2. Dodaj v opis vezja dva nepredznačena notranja signala: 12-bitni **data** in 8-bitni **adr**. Signal **adr** je naslovni register, ki je ob resetu postavljen na 0. Ob prehodu iz stanja s1 naj dobi vrednost iz podatkovnega signala **data**(7:0), ob s2 pa naslovnega števca.



3. Dopolni delovanje programskega števca: ob s1 naj se poveča za 1, razen, če je takrat na vodilu **data**(11:8) vrednost 4 (ukaz JMP). V tem primeru naj programski števec dobi vrednost spodnjih osmih bitov vodila. Preizkusi na simulaciji:



Izvedba ukazov

Dodali bomo še ukazni register in akumulator ter logiko za izvedbo dveh ukazov: LDA in ADD.

1. Definiraj 12-bitni akumulator in 4-bitni ukazni register. Ukazni register naj dobi vrednost zgornjih štirih bitov podatkovnega vodila ob stanju s1, ob stanju s2 pa se ukaz dekodira in izvede operacija:
   a. ukaz=1: prenesi vrednost iz vodila v akumulator
   b. ukaz=8: prištej akumulatorju vrednost iz vodila
   c. ukaz=9: odštej od akumulatorja vrednost iz vodila

   Preizkusi delovanje s simulacijo in opazuj vrednost akumulatorja. Spremeni program v pomnilniku, tako da bo vseboval operacijo odštevanja in preizkusi s simulacijo.

2. V krmilno enoto dodaj ukaz za pogojni skok (JZE) – skok naj se izvede, če je vrednost v akumulatorju enaka 0. Napiši program s pogojnim skokom in preveri delovanje.

ukaz	LDA	JMP	JZE	ADD	SBT
koda	1	4	5	8	9