Lasse Karagiannis, Simulering med Modelsim Uppgift 8 2016-11-05,
Uppdaterad 2016-11-06,
Uppdaterad 2016-11-12
Projektnamn: TEIS_ECS
Författare: Lasse Karagiannis
Kontakt: lasse.l.karagiannis@gmail.com

Innehållsförteckning

Lasse Karagiannis, Simulering med Modelsim Uppgift 8 2016-11-05,
Uppdaterad 2016-11-06,
Uppdaterad 2016-11-12

1 Verifiering

Verifiering med Modelsim har gjorts av enkel CPU. 50 instruktionscykel har stegats igenom, och värdet i CPU:ns register, adress och data bussar, samt lysdioderna status har analyserats. Registervärden och signalvärden på bussar har jämförts med den givna VHDL-koden. Tabellen testfall 1.1. visar ett antal testfall som undersökts.

1.1 Tabell med testfall

Testfall	Insignal	Utsignal	OK/Fel
Resetsignal som återställer konstruktionen till utgångsläget Reset_n <= ’0’	reset_n <= '0'	Nollställs PC, IR, R0, adressbuss och datautgång. Kontrollsignalerna sätts i inaktivt läge och tillstånd till Fetch_state	OK
NOP-instruktion	IR<=X"0000"	next_state = Execute_NOP_state	OK
LOAD-instruktion LOAD_R0 #A	IR< =X"100A"	CPU_REG_0 = ”0xA” under Execute_Load-state	OK
STORE-instruktion STORE_R0 #10	IR <= X"2010"	Addr_bus = 0b00010000 WE_n = '0' data_bus_out = REG_0 då next_state = STORE_2	OK
JMP-instruktion JMP #1	IR <= X"3001"	PC_reg <= IR[7:0] next_state <= Fetch_1_state;	OK, då next_state = EXECUTE_JMP

Lasse Karagiannis, Simulering med Modelsim Uppgift 8 2016-11-05,
Uppdaterad 2016-11-06,
Uppdaterad 2016-11-12

1.2 Beskrivning av testbänken

Testbänken innehåller ingenting annat än en uppdatering av klocksignalen, en reset-nivåändring från låg till hög. Då reset_n går hög kommer processorns state-machine att börja hämta instruktioner från minnet.

```
clock : PROCESS
-- variable declarations
BEGIN
    CLOCK_50 <= '0';
                wait for sys_clk_period/2;
                CLOCK_50 <= '1';
                wait for sys_clk_period/2;
END PROCESS clock;

SW(9) <= '0', '1' after 10*sys_clk_period;
```

1.3 Do filens beskrivning

Do-filen återfinns under mappen simulation, och skapas då testbänken genereras av verktyget.
Man kan lägga till signaler, genom att klicka på projektstrukturen (fönster med vit bakgrund) och expandera en trädstrukturen för en subkomponent som man vill studera. Då komponenter i hierarkin markeras så presenteras tillgängliga signaler i fönstret med blå bakgrund. Man höger klickar på dessa och väljer ”Add Wave”. Signalen läggs till i simuleringsfönstret, genom att en kodsnutt körs i terminalen.
Vill man simulera de manuellt tillagda signalerna från start av simuleringen, tillsammans med de automatiskt valda signalerna, så måste do-filen uppdateras med de kod-snuttar som verktyget genererade och körde genom terminalen då du lade till ytterliggare signaler.

```
add wave -position insertpoint  \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/reset_n
add wave -position insertpoint  \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/PC_reg
add wave -position insertpoint  \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/Addr_bus
add wave -position insertpoint  \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/bus_en_n
add wave -position insertpoint  \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/RD_n
add wave -position insertpoint  \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/data_bus_in
add wave -position insertpoint  \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/IR
add wave -position insertpoint  \
```

Lasse Karagiannis, Simulering med Modelsim Uppgift 8 2016-11-05,
Uppdaterad 2016-11-06,
Uppdaterad 2016-11-12
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/CPU_REG_0
add wave -position insertpoint \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/next_state
add wave -position insertpoint \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/WE_n
add wave -position insertpoint \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/data_bus_out
add wave -position insertpoint \
sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_inst_OUT_LED/LEDG

Dessa rader lades till efter raden med instruktionen *add wave **.
För att få ett kortare signalnamn så högerklickar man på signalen i fönstret för simuleringen och väljer ”Properties”. Där kan man skriva in ett alternativt kortare namn som visas i simuleringsfönstret.

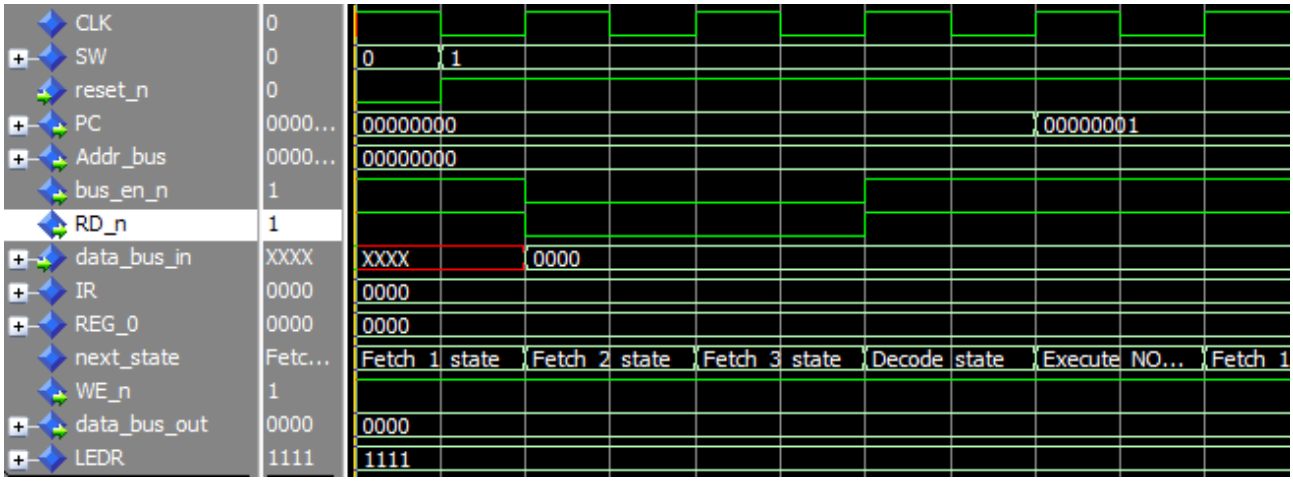
1.4 Resultatet från test

Resultatet från testerna visas i kapitel 2 Tidsanalys nedan

Lasse Karagiannis, Simulering med Modelsim Uppgift 8 2016-11-05,
Uppdaterad 2016-11-06,
Uppdaterad 2016-11-12

2 Tidsanalys

2.1 Cykler 1-14



Figur 1 Cykler 1-14, simulering Modelsim

CLK_PERIOD	1 to 10	11	12	13	14
SW	0	1	1	1	1
reset_n	0	1	1	1	1
PC	0b00000000	0b00000000	0b00000000	0b00000000	0b00000001
Addr_bus	0b00000000	0b00000000	0b00000000	0b00000000	0b00000000
bus_en_n	1	0	0	1	1
RD_n	1	0	0	1	1
data_bus_in	0xUUUU	0x0000	0x0000	0x0000	0x0000
IR	0x0000	0x0000	0x0000	0x0000	0x0000
REG_0	0x0000	0x0000	0x0000	0x0000	0x0000
next_state	Fetch_1_state	Fetch_2_state	Fetch_3_state	Decode	Execute_NOP
WE_n	1	1	1	1	1
data_bus_out	0x0000	0x0000	0x0000	0x0000	0x0000
LEDR	0b1111	0b01111	0b1111	0b1111	0b1111

Tabell 1 Signaler och registervärden cykler 1-14

Fetch_1_state ligger som next_state fram till att reset_n går går hög, samtliga register är nollställda under reset-fasen, vi har nämligen kod (klistrar in en del av den)

process(reset_n, clk_50)

Lasse Karagiannis, Simulering med Modelsim Uppgift 8 2016-11-05,
Uppdaterad 2016-11-06,
Uppdaterad 2016-11-12
begin

```
if reset_n = '0' then
  PC_reg <= X"00"; -- after reset, PC is zero (i.e address 0)
        IR <= X"0000"; -- NOP, no operation
        Addr_bus <= X"00" ; -- initialize registers
        CPU_REG_0 <= (others => '0'); -- initialize registers
        next_state <= Fetch_1_state;
        bus_en_n <= '1';
        data_bus_out <= "0000000000000000"; -- initialize registers
   WE_n <= '1';
   RD_n <= '1';
        CPU_state <= "00";

elsif(rising_edge(clk_50)) then
```

Lysdioderna lyser p.g.a. koden i OUT_LED

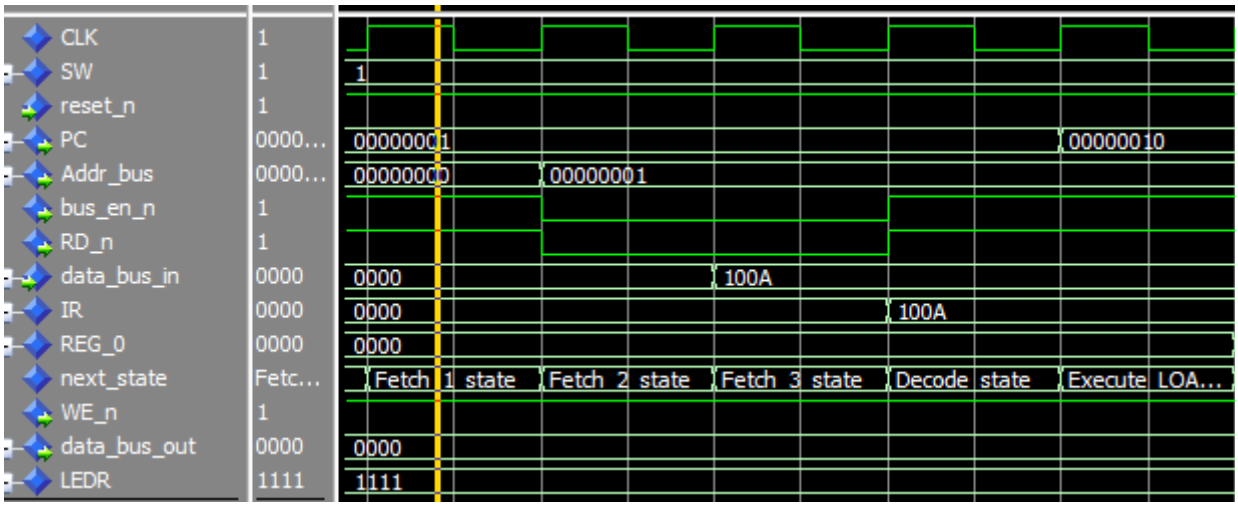
```
process(reset_n, clk_50)
                begin
                if reset_n = '0' then
                        LEDG <= "1111";      -- LED-signals go high during reset

                elsif(rising_edge(clk_50)) then
```

Instruktionsläsningen kräver 2 cykler, där IR har laddats med instruktionskoden för NOP 0x0000 och är giltlig under cykel 14.

Lasse Karagiannis, Simulering med Modelsim Uppgift 8 2016-11-05,
Uppdaterad 2016-11-06,
Uppdaterad 2016-11-12

2. 2 Cykler 15-19



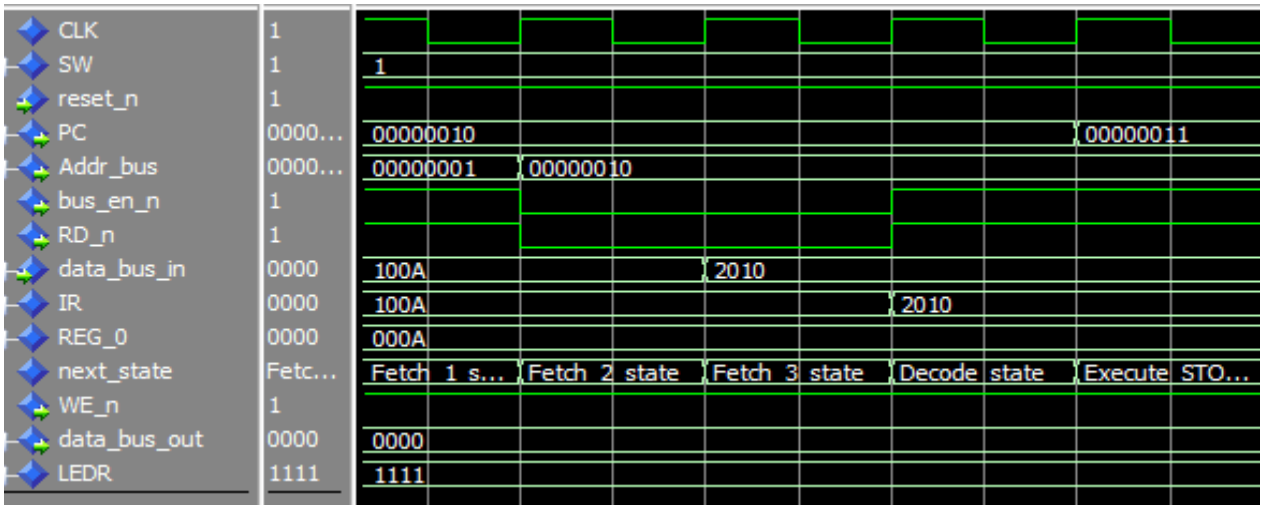
Figur 2 Cykler 15-19 simulering Modelsim

CLK_PERIOD	15	16	17	18	19
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000001	0b00000001	0b00000001	0b00000001	0b00000010
Addr_bus	0b00000000	0b00000001	0b00000001	0b00000001	0b00000001
bus_en_n	1	0	0	1	1
RD_n	1	0	0	1	1
data_bus_in	0x0000	0x0000	0x100A	0x100A	0x100A
IR	0x0000	0x0000	0x0000	0x100A	0x100A
REG_0	0x0000	0x0000	0x0000	0x0000	0x0000
next_state	Fetch_1	Fetch_2	Fetch_3	Decode	Execute_LOAD
WE_n	1	1	1	1	1
data_bus_out	0x0000	0x0000	0x0000	0x0000	0x0000
LEDR	0b1111	0b1111	0b1111	0b1111	0b1111

Tabell 2 Signaler och registervärden cykler 15-19

Här hämtas och exekveras intruktionen LOAD_R0 #A med op-kod 0x100A på adressposition 1 i kod-minnet. PC inkrementerades redan under exekveringsfasen av NOP, under cykel 14. Addr_bus kan inte adressera kod-minnet under decode-fasen av NOP, eftersom samma adressbuss används för både kod och data.
En hel cykel används (16) till att låta RD_n och bus_en_n bli stabil, och koppla PC till adressbussen.
Nästa cykel (17) liger kod-data på kod-bussen, för att läsas in till IR cykeln därpå. Man borde kunna slå ihop cykel 16 och 17, dvs. att RD_n och bus_en_n bara får vara låga under en cykel och IR läses samtidigt som kod ligger stabil på kodbussen utan något mellansteg.

2.3 Cykler 20 – 24



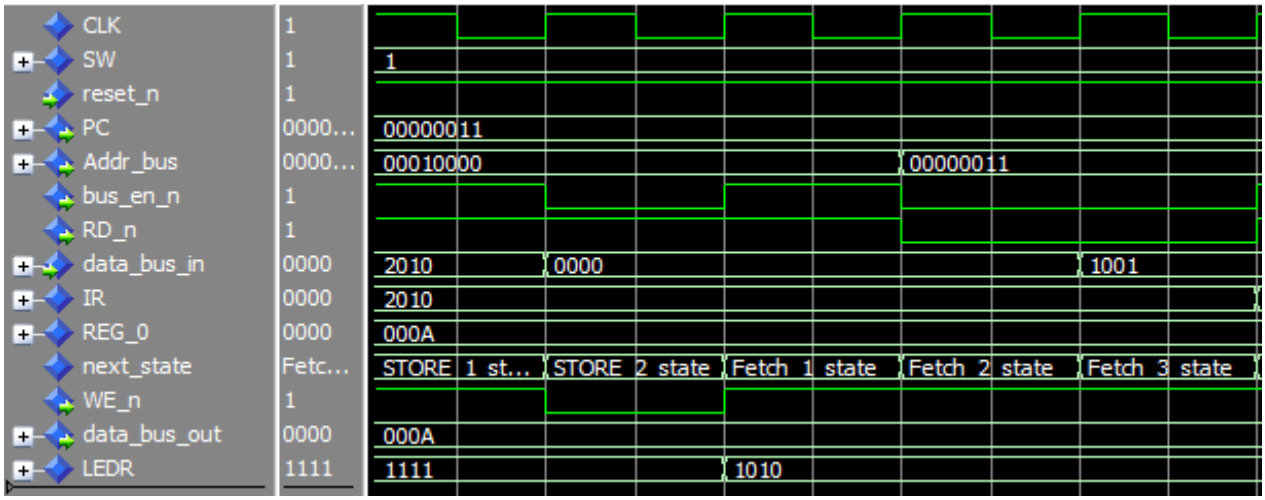
Figur 3 Cykler 20-24 simulering Modelsim

CLK_PERIOD	20	21	22	23	24
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000010	0b00000010	0b00000010	0b00000010	0b00000011
Addr_bus	0b00000001	0b00000010	0b00000010	0b00000010	0b00000010
bus_en_n	1	0	0	1	1
RD_n	1	0	0	1	1
data_bus_in	0x100A	0x100A	0x2010	0x2010	0x2010
IR	0x100A	0x100A	0x100A	0x2010	0x2010
REG_0	0x000A	0x000A	0x000A	0x000A	0x000A
next_state	Fetch_1	Fetch_2	Fetch_3	Decode	Execute_STORE
WE_n	1	1	1	1	1
data_bus_out	0x0000	0x0000	0x0000	0x0000	0x0000
LEDR	0b1111	0b1111	0b1111	0b1111	0b1111

Tabell 3 Signaler och registervärden cykler 20-24

Under cykel 20 sker exekvering av LOAD instruktionen där R0 laddas med argumentet 0xA. PC inkrementerades under decode-fasen av LOAD_R0 #A. Nästa cykel (21) är Fetch_1, och då är PC kopplad till adressbusesen, och då bus_en_n och RD_n går från låg till hög laddas IR med op-koden 0x2010, vars mnemonic är STORE_R0 #10. Under decode (cykel 24) inkrementeras PC återigen.

2.4 Cykler 25-29



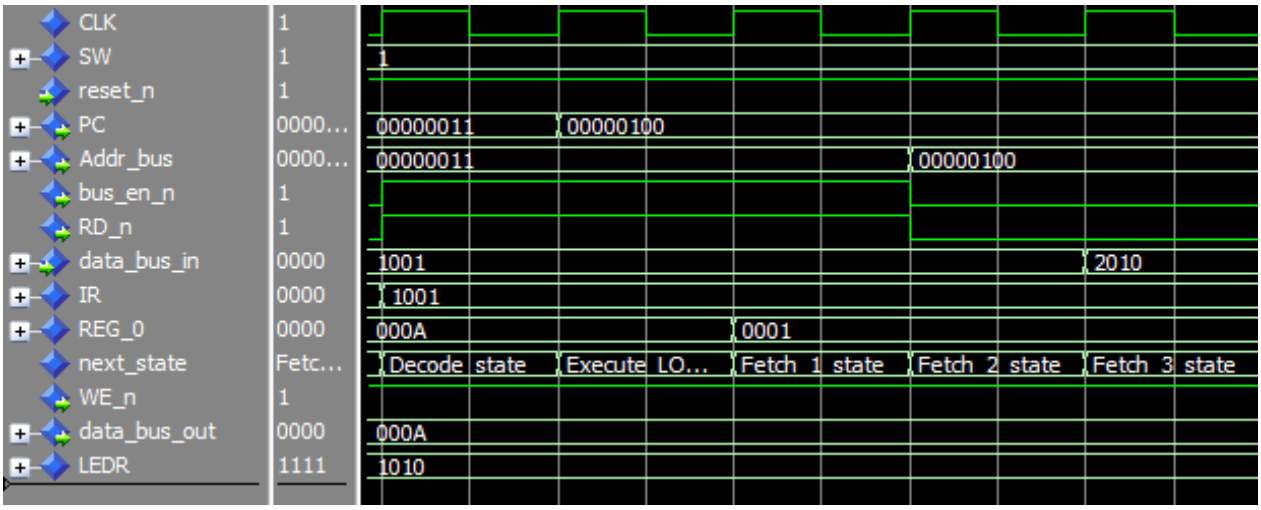
Figur 4 Cykler 25-29 simulering Modelsim

CLK_PERIOD	25	26	27	28	29
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000011	0b00000011	0b00000011	0b00000011	0b00000011
Addr_bus	0b00010000	0b00010000	0b00010000	0b00000011	0b00000011
bus_en_n	1	0	1	0	0
RD_n	1	1	1	0	0
data_bus_in	0x2010	0x0000	0x0000	0x0000	0x1001
IR	0x2010	0x2010	0x2010	0x2010	0x2010
REG_0	0x000A	0x000A	0x000A	0x000A	0x000A
next_state	STORE_1	STORE_2	Fetch_1	Fetch_2	Fetch_3
WE_n	1	0	1	1	1
data_bus_out	0x000A	0x000A	0x000A	0x000A	0x000A
LEDR	0b1111	0b1111	0b1010	0b1010	0b1010

Tabell 4 Signaler och registervärden cykler 25-29

Under Execute_STORE (cykel 25) kopplas IR:s argument 0x10 till Adressbussen och Reg_0 läggs ut på data_bussen. Under cykel 26 görs data giltlig genom att WE_n går låg. Senast cykel 27 lyser dioderna enligt bitmönstret 0b1010 (0xA).
PC har varit inkrementerad sedan flera klockcykler tidigare. Under Fetch_1 (cykel 28) koppas PC ut till Adressbssen och bus_en_n och RD_n går då återigen samtidigt låga. IR kommer laddas under Fetch_3 (cykel 30, se nästa avsnitt).

2.5 Cykler 30 – 34



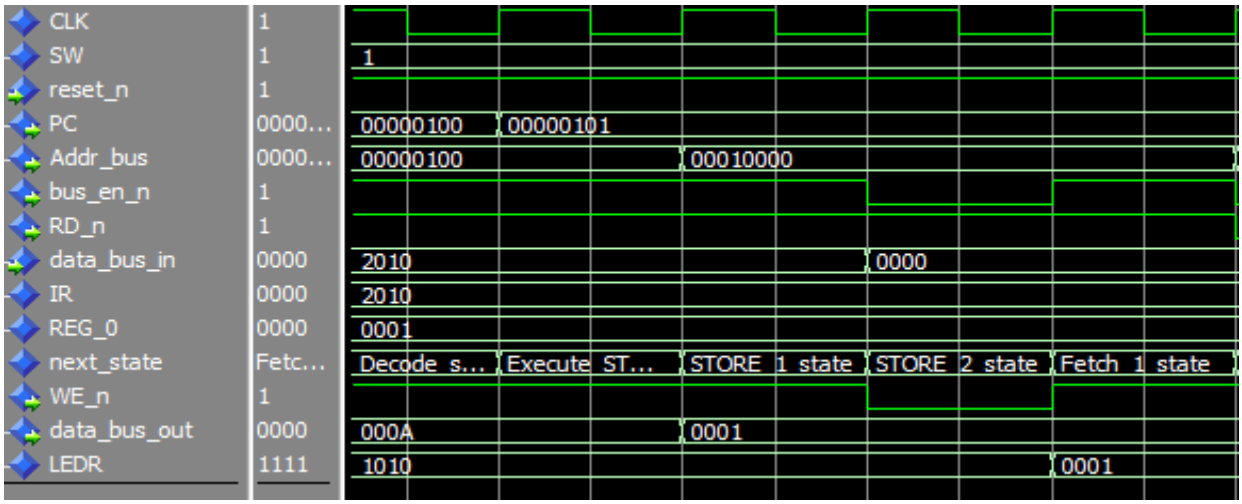
Figur 5 Cykler 30-34 simulering Modelsim

CLK_PERIOD	30	31	32	33	34
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000011	0b00000100	0b00000100	0b00000100	0b000001000
Addr_bus	0b00000011	0b00000011	0b00000011	0b00000100	0b000001000
bus_en_n	1	1	1	0	0
RD_n	1	1	1	0	0
data_bus_in	0x1001	0x1001	0x1001	0x1001	0x2010
IR	0x1001	0x1001	0x1001	0x1001	0x1001
REG_0	0x000A	0x000A	0x0001	0x0001	0x0001
next_state	Decode	Execute_LOAD	Fetch_1	Fetch_2	Fetch_3
WE_n	1	1	1	1	1
data_bus_out	0x000A	0x000A	0x000A	0x000A	0x000A
LEDR	0b1010	0b1010	0b1010	0b1010	0b1010

Tabell 5 Signaler och registervärden cykler 30-34

Op-koden som IR håller är 0x1001 vars mnemonic är LOAD_R0 #1. Under decode (cykel 31) inkrementeras PC. Under Execute_LOAD (cykel 32) laddas Reg_0 med OP-kodens argument. Under Fetch_1 (cykel 33) kopplas PC till adressbussen bus_en_n och RD_n går då låga. Nästa instruktion som är resultatet av adresseringen av kod-minnet läggs ut på kod-bussen under cykel 34, och läses in till IR under cykel 36.

2.6 Cykler 35-39



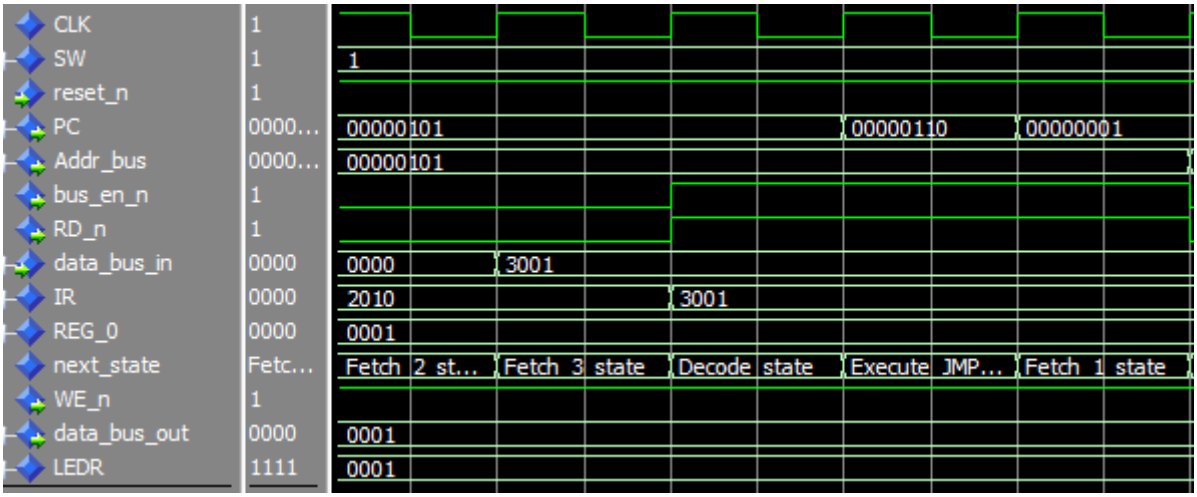
Figur 6 Cykler 35-39 simulering Modelsim

CLK_PERIOD	35	36	37	38	39
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000100	0b00000101	0b00000101	0b00000101	0b00000101
Addr_bus	0b00000100	0b00000100	0b00010000	0b00010000	0b00010000
bus_en_n	1	1	1	0	1
RD_n	1	1	1	1	1
data_bus_in	0x2010	0x2010	0x2010	0x0000	0x0000
IR	0x2010	0x2010	0x2010	0x2010	0x2010
REG_0	0x0001	0x0001	0x0001	0x0001	0x0001
next_state	Decode	Execute_STORE	STORE_1	STORE_2	Fetch_1
WE_n	1	1	1	0	1
data_bus_out	0x000A	0x000A	0x0001	0x0001	0x0001
LEDR	0b1010	0b1010	0b1010	0b1010	0b0001

Tabell 6 Signaler och registervärden cykler 35-39

Inläsningen till IR innebär att op-koden 0x2010, vilket är menomic för STORE_R0 #10.
Under deode inkrementeras PC (cykel 36), och Execute_STORE görs under cykel 37. Då kopplas IR:s argument till adressbussen och REG_0 läggs ut på databussen. WE_n går låg under följande cykel (38) och senast cykel 39 lyser lysdioderna i enlighet med vad som lades ut på databussen dvs. 0b0001.

2.7 Cykler 40 – 44



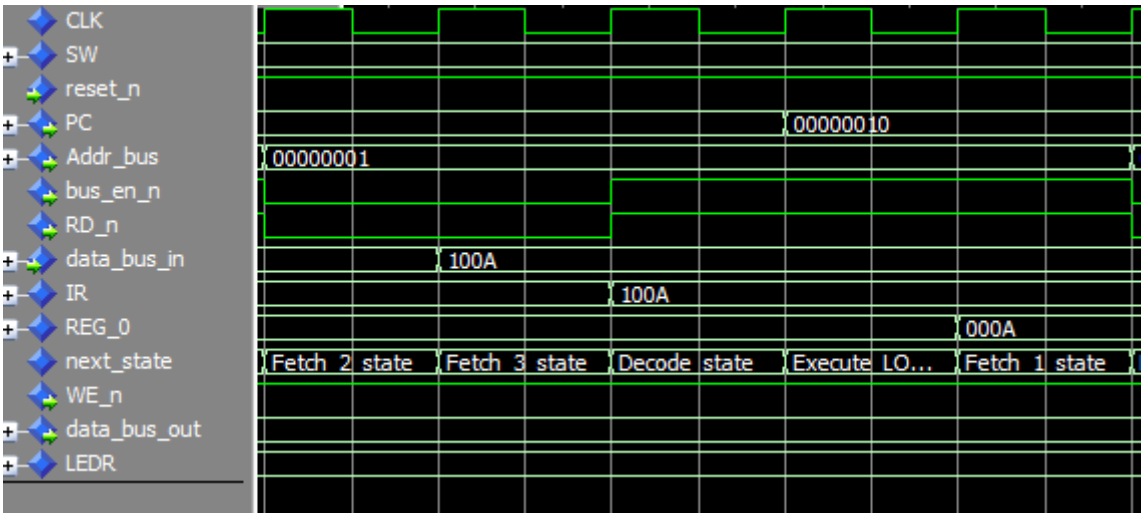
Figur 7 Cykler 40-44 simulering Modelsim

CLK_PERIOD	40	41	42	43	44
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000101	0b00000101	0b00000101	0b00000110	0b00000001
Addr_bus	0b00000101	0b00000101	0b00000101	0b00000101	0b00000101
bus_en_n	0	0	1	1	1
RD_n	0	0	1	1	1
data_bus_in	0x0000	0x3001	0x3001	0x3001	0x3001
IR	0x2010	0x2010	0x3001	0x3001	0x3001
REG_0	0x0001	0x0001	0x0001	0x0001	0x0001
next_state	Fetch_2	Fetch_3	Decode	Execute_JMP	Fetch_1
WE_n	1	1	1	1	1
data_bus_out	0x0001	0x0001	0x0001	0x0001	0x0001
LEDR	0b0001	0b0001	0b0001	0b0001	0b0001

Tabell 7 Signaler och registervärden cykler 40-44

Under Fetch_1 (cykel 40) kopplas PC till adressbussen och bus_en_n och RD_n går båda låga. Nästa instruktion, dvs. den på adress adresserad av PC finns på kodbussen under cykel 41, och inläsning till IR sker under nästa cykel (42). Op koden är 0x3001 vars mnemonic är JMP #1. Under decode inkrementeras PC (cykel 43), men under Execute_JMP (cykel 44) laddas PC med instruktionens argument dvs. IR:s argument-del kopplas till PC.

2.8 Cykler 45-49



Figur 8 Cykler 45-49 simulering Modelsim

CLK_PERIOD	45	46	47	48	49
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000001	0b00000001	0b00000001	0b00000010	0b00000010
Addr_bus	0b00000001	0b00000001	0b00000001	0b00000001	0b00000001
bus_en_n	0	0	1	1	1
RD_n	0	0	1	1	1
data_bus_in	0x3001	0x100A	0x100A	0x100A	0x100A
IR	0x3001	0x3001	0x100A	0x100A	0x0000
REG_0	0x0001	0x0001	0x0001	0x0001	0x000A
next_state	Fetch_2	Fetch_3	Decode	Execute_LOAD	Fetch_1
WE_n	1	1	1	1	1
data_bus_out	0x0001	0x0001	0x0001	0x0001	0x0001
LEDR	0b0001	0b0001	0b0001	0b0001	0b0001

Tabell 8 Signaler och registervärden cykler 45-49

Under Fetch_1 (cykel 45) kopplas PC till Adressbussen och bus_en_n och RD_n går båda låga samtidigt. Instrutkionen som läses in till IR (cykel 47) är från adress 1 i kodminnet LOAD_R0 #A med OP-kod 0x100A, som exekverats förut.
Under decode (cykel 48) inkrementeras PC, och under Execute_LOAD (cykel 49) har Reg_0 laddats med argument delen av IR.