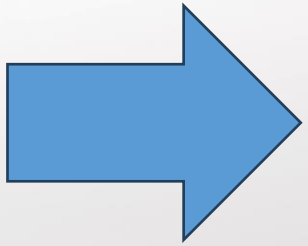
# Atividade 2 executar iXOR, iADD, iSUB, iLESS (a)

- **FSM**: criar o estado sALU e deixar como condição *default* da máquina de estados

```
131     process(ck, rst)
132     begin
133         if rst = '1' then
134             state <= sFETCH;
135         elsif ck'event and ck = '1' then
136             case state is
137                 when sFETCH =>
138                     state <= sEXE;
139                 when sEXE =>
140                     if inst = iREAD then
141                         state <= sREAD;
142                     else
143                         state <= sEND;
144                     end if;
145                 when sEND =>
146                     state <= sEND;
147                 when others =>
148                     state <= sFETCH;
149             end case;
150         end if;
151     end process;
```



```
131     process(ck, rst)
132     begin
133         if rst = '1' then
134             state <= sFETCH;
135         elsif ck'event and ck = '1' then
136             case state is
137                 when sFETCH =>
138                     state <= sEXE;
139                 when sEXE =>
140                     if inst = iEND then
141                         state <= sEND;
142                     elsif inst = iREAD then
143                         state <= sREAD;
144                     else
145                         state <= sALU;
146                     end if;
147                 when sEND =>
148                     state <= sEND;
149                 when others =>
150                     state <= sFETCH;
151             end case;
152         end if;
153     end process;
```

- **Decodificar** as instruções iXOR (4), iSUB (5), iADD (6), iLESS (7) no trecho de código:

```
121     inst <= iREAD when ir(15 downto 12) = x"0" else     -- decode the current instruction
122             --complete
123             iEND;
```

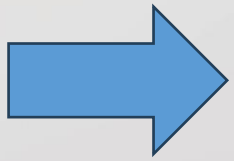
# Atividade 2: executar iXOR, iADD, iSUB, iLESS (b)

- **Completar** os sinais de controle com o estado sALU, pois neste estado devem ser alterados o PC e o banco de registradores

```
125     wPC <= '1' when state = sREAD --complete
126         else '0';
127     wReg <= '1' when state = sREAD  --complete
128         else '0';
```

- **Acrescentar** as operações iXOR, iSUB, iLESS na ALU

```
101   -- arithmetic and logic unit
102   --
103    outalu <=   RS2           when inst = iWRITE else  -- data to be
104                --complete
105                RS1 + RS2;     --  default operation: iADD
```



```
outalu <=   RS2           when inst = iWRITE else
            ...           when inst = ...    else
            ...           when inst = ...    else
            ...           when inst = ...    else
            RS1 + RS2;    --  default operation: iADD
```

# Atividade 2: executar iXOR, iADD, iSUB, iLESS (c)

- **Programa** de teste

Já temos:

R0 ← 1111

R1 ← 2222

R2 ← 3333

R3 ←4444

**Vamos programar** (acrescentar o código binário no testbench)

R0 ← R0 + R3 - espera-se 5555 em R0

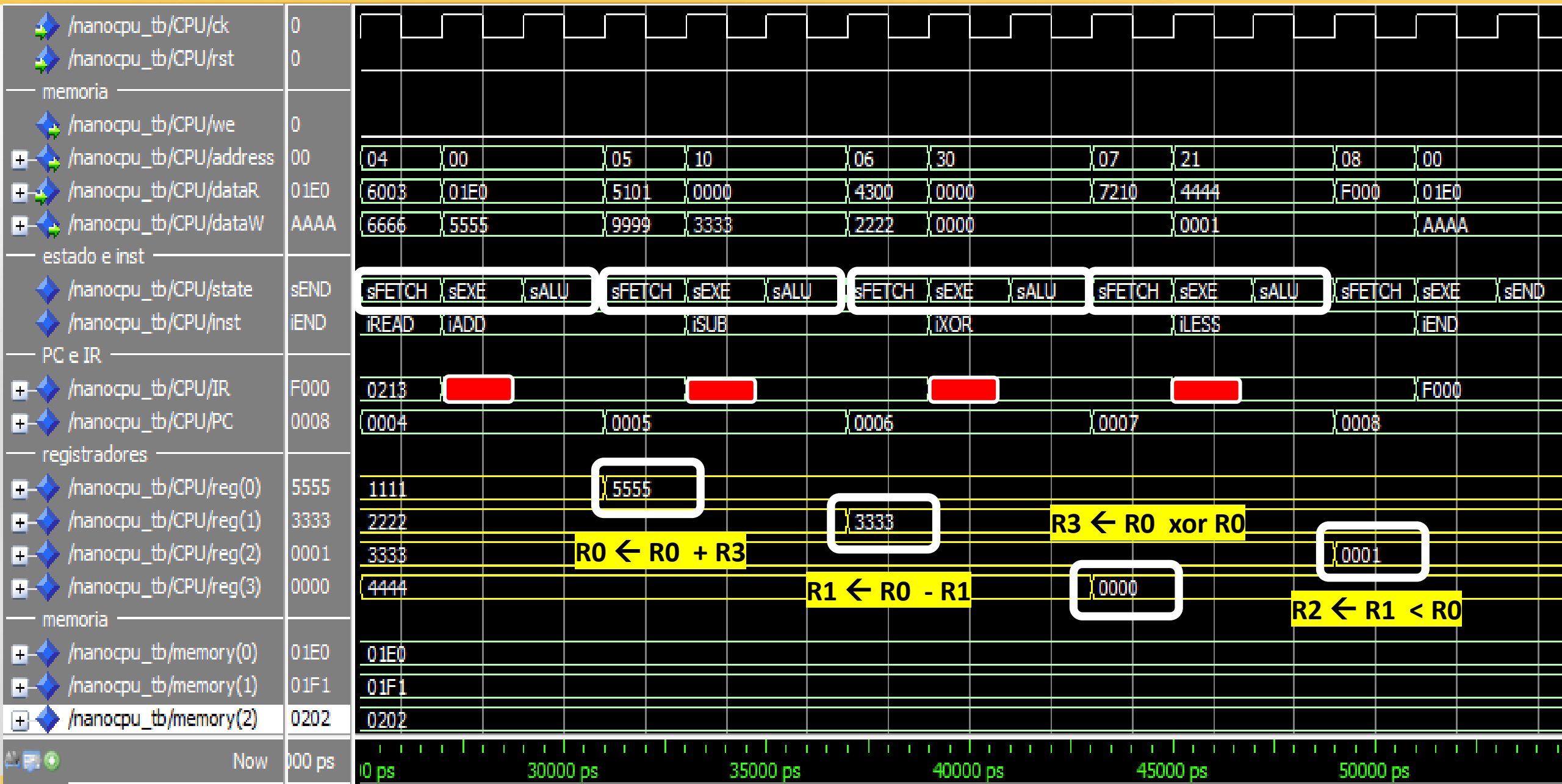
R1 ← R0 – R1 - espera 3333 em R1

R3 ← R0 xor R0 - espera-se que o R3 seja **zerado** (método comum para zerar registradores)

R2 ← R1 < R0 - 3333 < 5555, logo R2 deve ser 1

PUCRS ESCOLA POLITÉCNICA

# Atividade 2: executar iXOR, iADD, iSUB, iLESS (d)



# Atividade 3 escrita na memória (a)

## Modificações necessárias no código:

(1) Acrescentar o estado **sWRITE** na lista de estados

```
67      type stateType is (sFETCH, sEXE, sREAD, sALU, sEND); --complete
```

(2) Ativar o we (1) no estado **sWRITE**

```
81          we <= '0';    -- complete
```

(3) Decodificar a instrução **iWRITE**

(4) Acrescentar no **wPC** o estado **sWRITE**

(5) Acrescentar na FSM o estado **sWRITE**

**Simulação:** Escrever o conteúdo dos registradores R0 (5555), R1 (3333), R2 (0001) na posições 15 $(0F)_{16}$, 16 $(10)_{16}$ e 17 $(11)_{16}$

PUCRS ESCOLA POLITÉCNICA

# Atividade 3: escrita na memória (b)