Thực Hành
Thiết kế vi mạch số với HDL
Báo cáo bài tập lớn

Đề bài : Thiết kế vi xử lý đơn giản bằng verilog HDL

Nhóm thực hiện:
Hồ Nhựt Minh 50901549
Ngô Thành Nhân 50901808

12/2012

Nội dung

I. Giới thiệu

Nội dung đề tài:

Thiết kế một vi xử lý đơn giản trên board Altera DE2 với chức năng như sau :

1. Chương trình nguồn được lưu trên block RAM dưới dạng mã máy
2. CPU sẽ đọc tuần tự các mã lệnh này để thực thi (mã lệnh được mô tả bên dưới)
3. Cho phép hiển thị nội dung ô nhớ trong RAM ra led 7 đoạn (Advanced: Memory mapped IO)

II. Thiết kế

1. Thanh ghi

Vi xử lý hỗ trợ 4 thanh ghi
- R0: 00
- R1: 01
- R2: 10
- R3: 11

2. Tập lệnh

Lệnh có định dạng opcode động, cố định 10 bit.

Chia làm 3 loại lệnh cơ bản:

- **Lệnh liên quan đến ALU**

Lệnh	Opcode (4bit)	Source1 (2bit)	Source2 (2bit)	Destination (2bit)	Chức năng
ADD	0000	xx	xx	xx	Destination = source1 + source2
SUB	0001	xx	xx	xx	Destination = source1- source2
AND	0010	xx	xx	xx	Destination = source1 & source2
OR	0011	xx	xx	xx	Destination = source1 | source2
NOT	0100	xx	Don’t care	xx	Destination = ~ source1

- **Lệnh liên quan đến Program counter**

Lệnh	Opcode (5bit)	Don't care (5bit)	Chức năng
SIZ	01010	xxxxx	Bỏ qua lệnh kế tiếp nếu cờ zero được bật
NOP	01011	xxxxx	Không làm gì cả

Lệnh	Opcode (3bit)	Address (7bit)	Chức năng
JUMP	011	xxxxxxx	Nhảy đến và thực thi chương trình tại địa chỉ có giá trị address

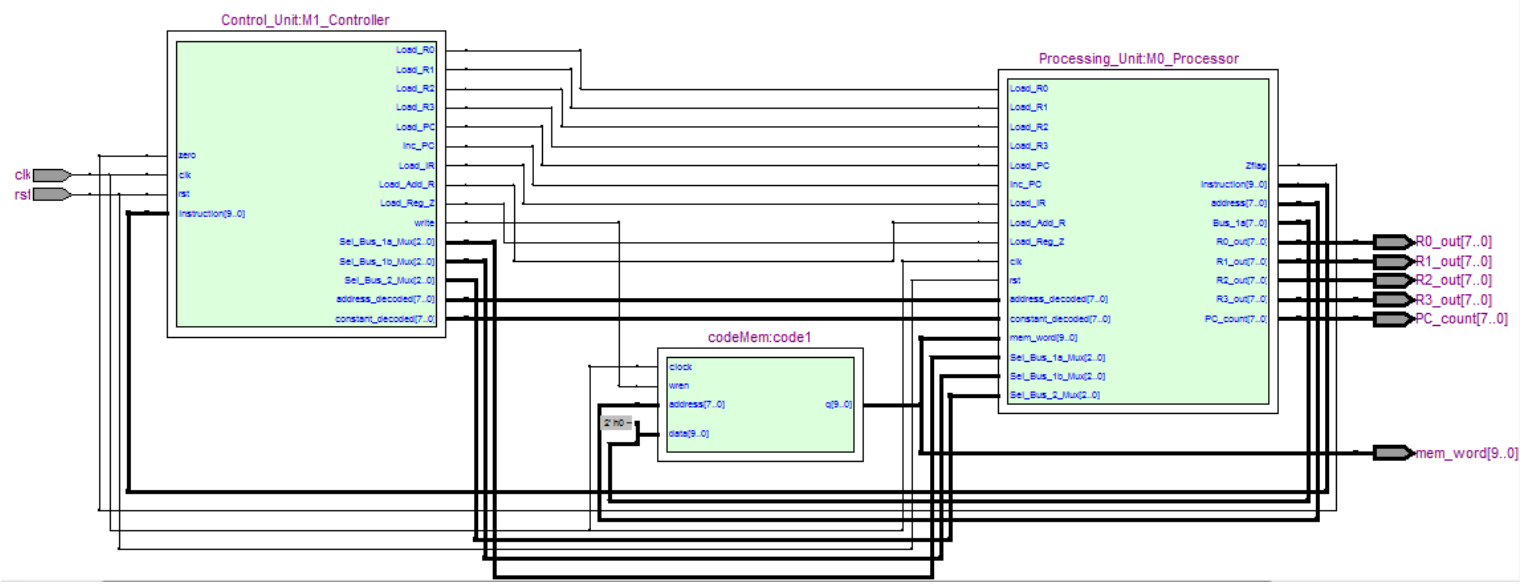
- **Lệnh liên quan đến dữ liệu và Ram**

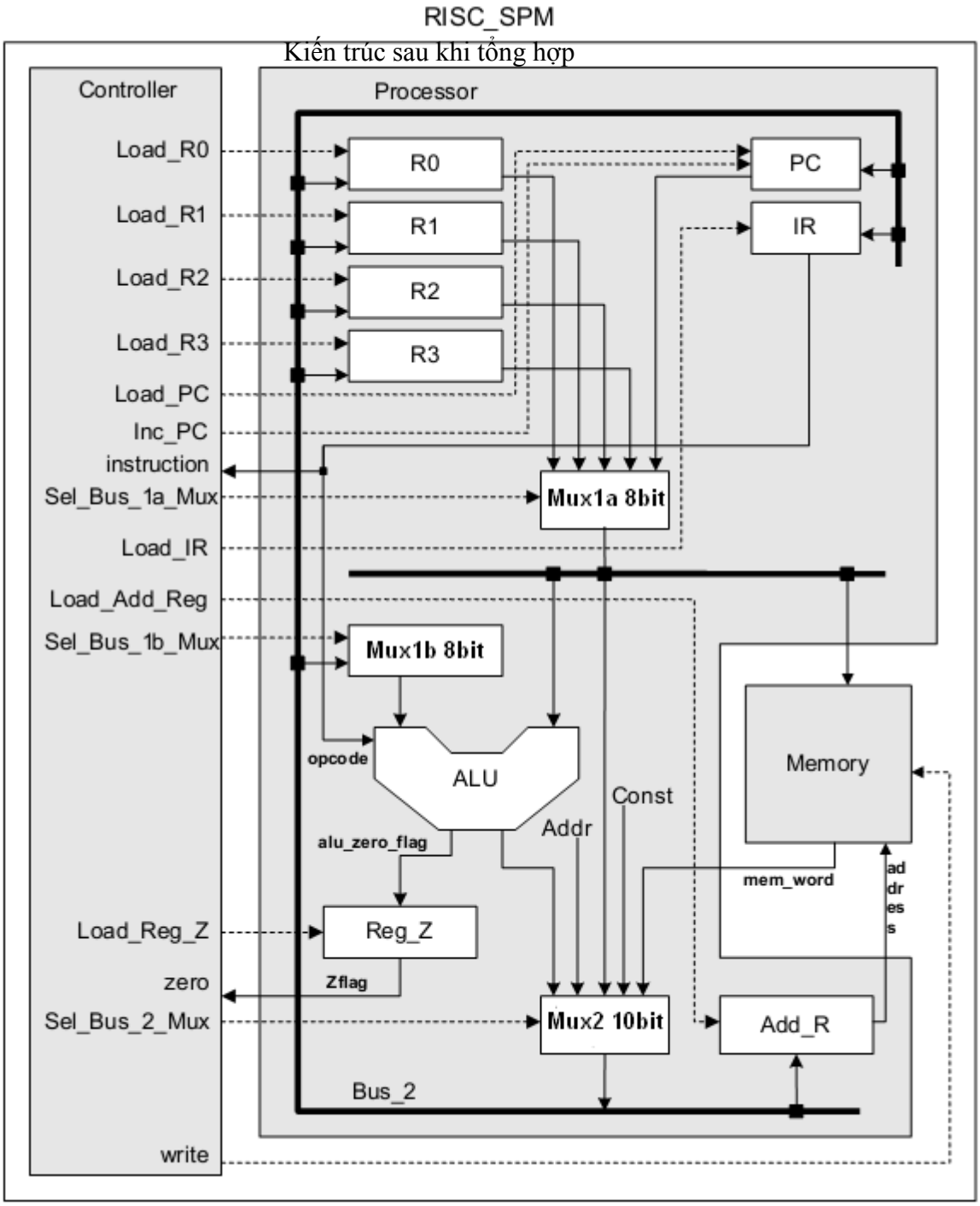
Lệnh	Opcode (3bit)	Address (7bit)	Chức năng
STORE	100	xxxxxxx	Lưu giá trị của R0 vào bộ nhớ tại địa chỉ Address
LOAD	101	xxxxxxx	Lấy giá trị từ bộ nhớ tại địa chỉ Address và lưu vào R0

Lệnh	Opcode (2bit)	Constant (8bit)	Chức năng
Save	11	xxxxxxxx	Lưu giá trị của một số Integer 8 bit vào R0

3. Kiến trúc

Các Module chính gồm có: Processing Unit, Control Unit và Memory Unit.





Sơ đồ tổng quan của hệ thống

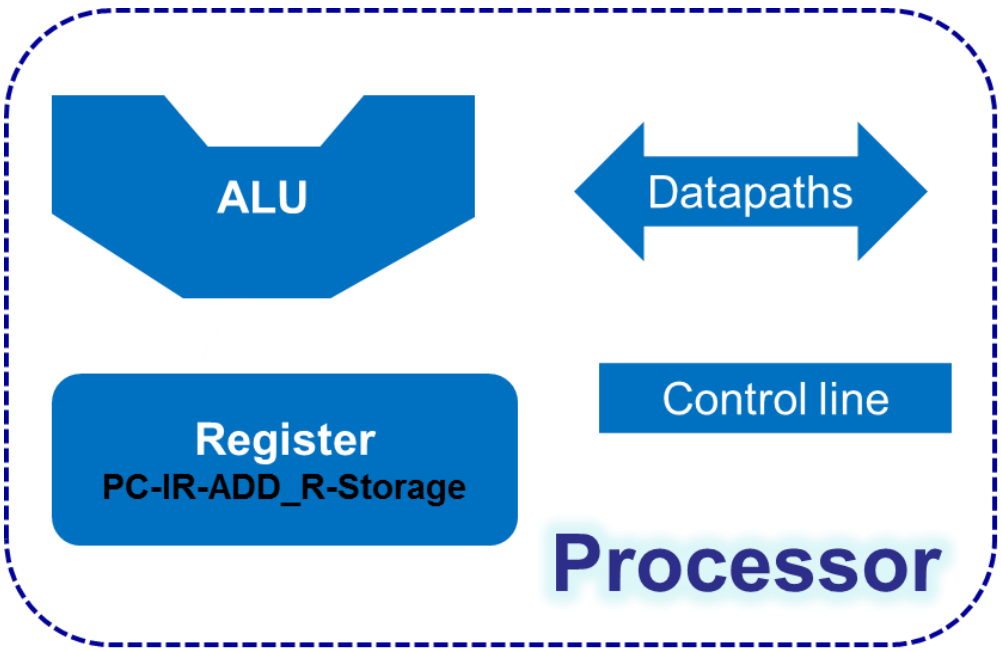
```
Processing_Unit M0_Processor (...);
// Thực thi chương trình

Control_Unit M1_Controller (...);
// Điều khiển các tín hiệu cho Processor và Memory
```

```
codeMem code1(...);
// Chứa chương trình và dữ liệu
```

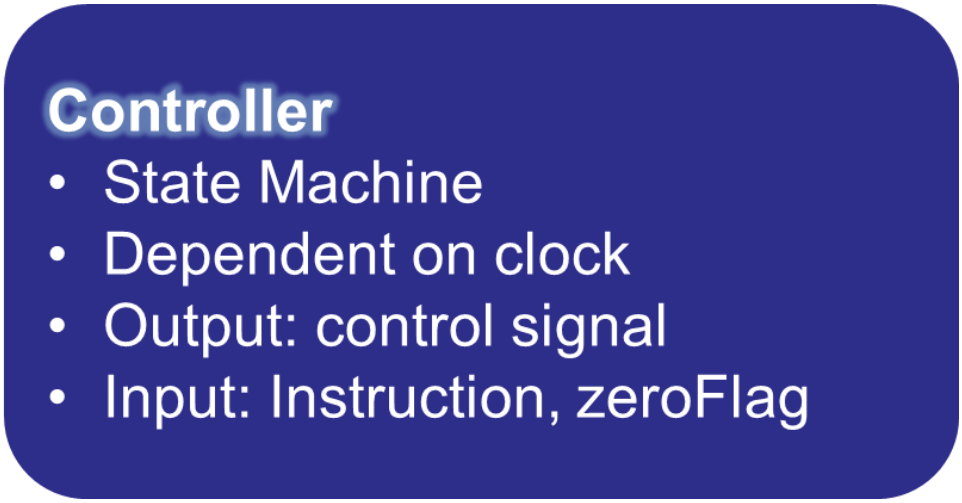
- **Cấu trúc các module**

a) Cấu trúc của Processor



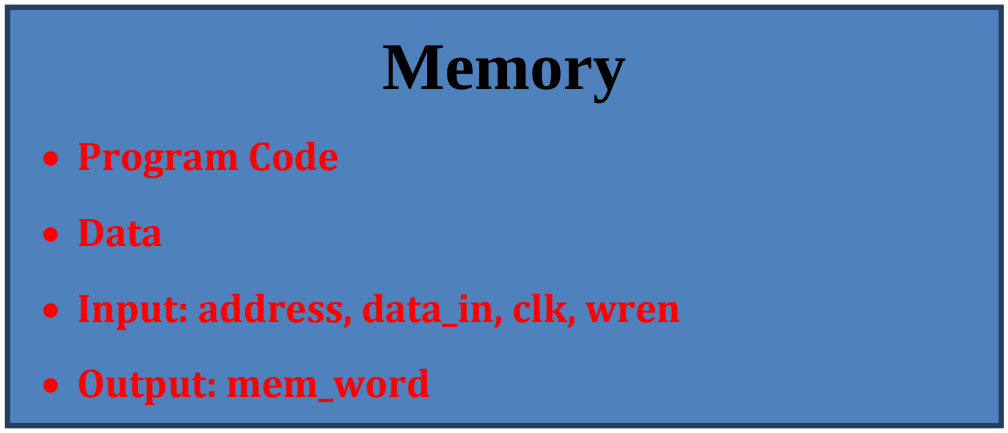

```
output [word_size-1: 0]   instruction;
output [address_size-1:0] address;
output [data_size-1: 0]   Bus_1a,Bus_1b;
output  Zflag;
input [address_size-1:0] address_decoded;
input [data_size-1 :0] constant_decoded;
input [word_size-1: 0]    mem_word;
input         Load_R0, Load_R1, Load_R2, Load_R3, Load_PC, Inc_PC;
input [Sel1_size-1: 0]    Sel_Bus_1a_Mux,Sel_Bus_1b_Mux;
input [Sel2_size-1: 0]    Sel_Bus_2_Mux;
input             Load_IR, Load_Add_R,  Load_Reg_Z;
input             clk, rst;
```

b) *Cấu trúc của Controller*




```
output reg Load_R0, Load_R1,  Load_R2, Load_R3,  Load_PC,Inc_PC;
output  [Sel1_size-1:0]Sel_Bus_1a_Mux,Sel_Bus_1b_Mux;
output  [Sel2_size-1:0]Sel_Bus_2_Mux;
output reg Load_IR, Load_Add_R, Load_Reg_Z,  write;
output  [address_size-1:0] address_decoded;
output  [data_size-1:0] constant_decoded;
input  [word_size-1:0]instruction;
input  zero, clk, rst;
```

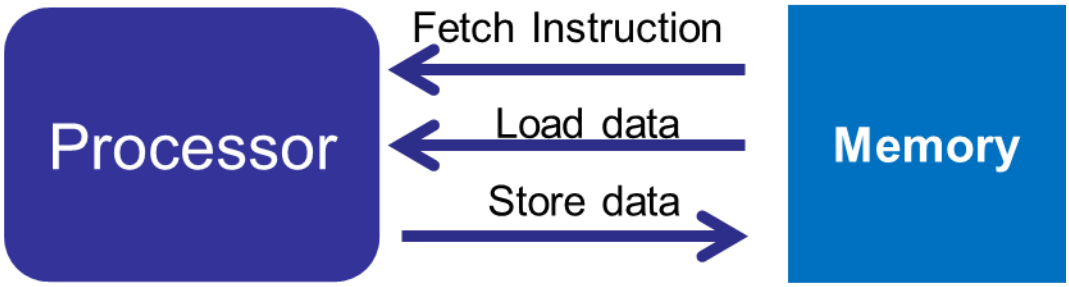
c) *Cấu trúc của Memory*



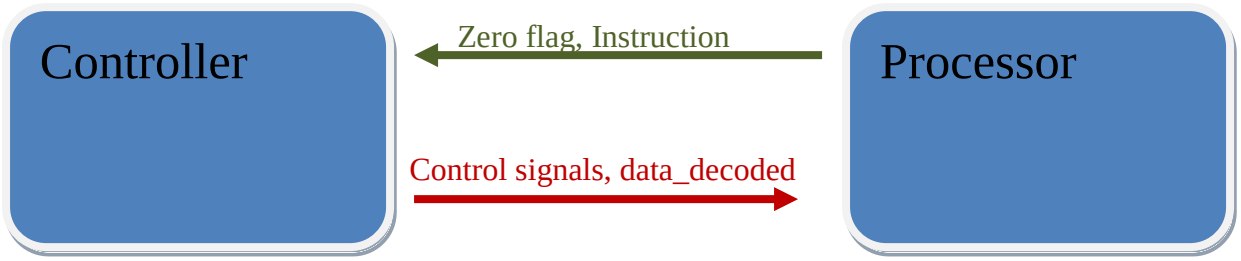

```
output [word_size-1: 0] data_out;
input [word_size-1: 0] data_in;
input [address_size-1: 0] address;
input clk, write;
```

- **Giao tiếp giữa các module**

a) Giao tiếp giữa Processor và Memory



b) Giao tiếp giữa Processor và Controller



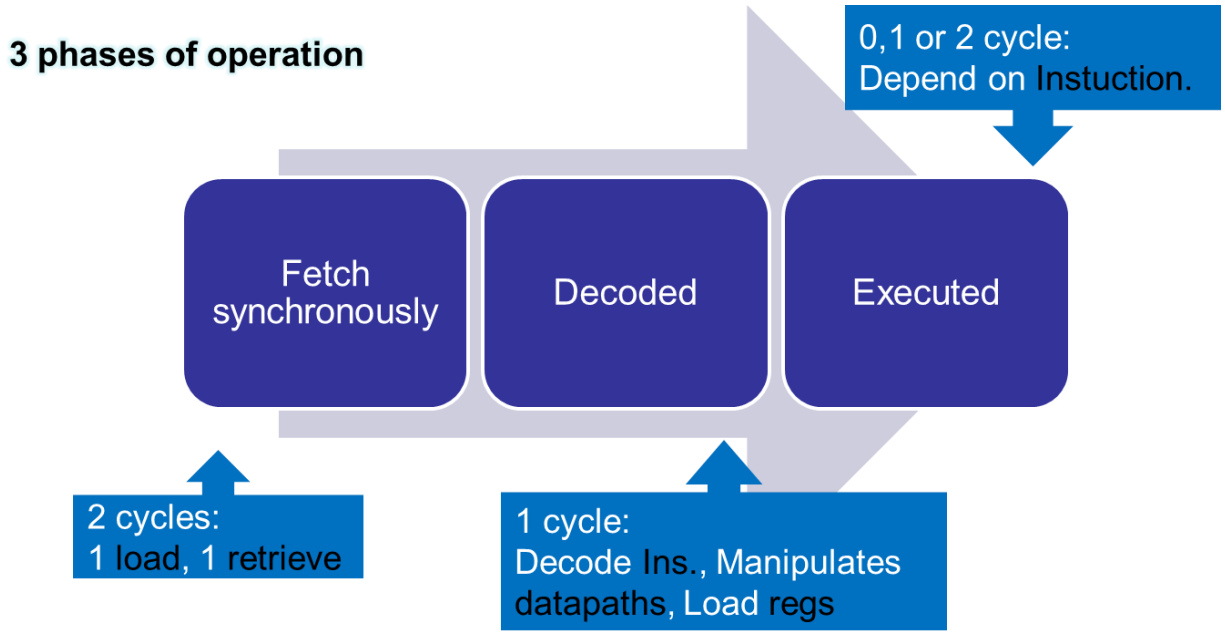
Trong đó :

Control signals và data_decoded bao gồm :

```
output reg Load_R0, Load_R1,  Load_R2, Load_R3,  Load_PC,Inc_PC;
// điều khiển các thanh ghi R0,R1,R2,R3,PC
output  [Sel1_size-1:0]Sel_Bus_1a_Mux,Sel_Bus_1b_Mux;
//điều khiển multiplexer 5 kênh 1a và 1b, để chọn ra input cho
bus 1a và bus 1b
output  [Sel2_size-1:0]Sel_Bus_2_Mux;
// điều khiển multiplexer 2 để chọn ra dữ liệu cho bus 2
output reg Load_IR, Load_Add_R, Load_Reg_Z,  write;
// load giá trị bus 2 vào instruction register, Address
Register, Load zero vao Reg_Z, write
output  [address_size-1:0] address_decoded;
// giá trị address trong các lệnh jump, store, load
output  [data_size-1:0] constant_decoded;
// giá trị số trong câu lệnh save
```

- **Chu kỳ lệnh và máy trạng thái**

a) Chu kỳ lệnh:

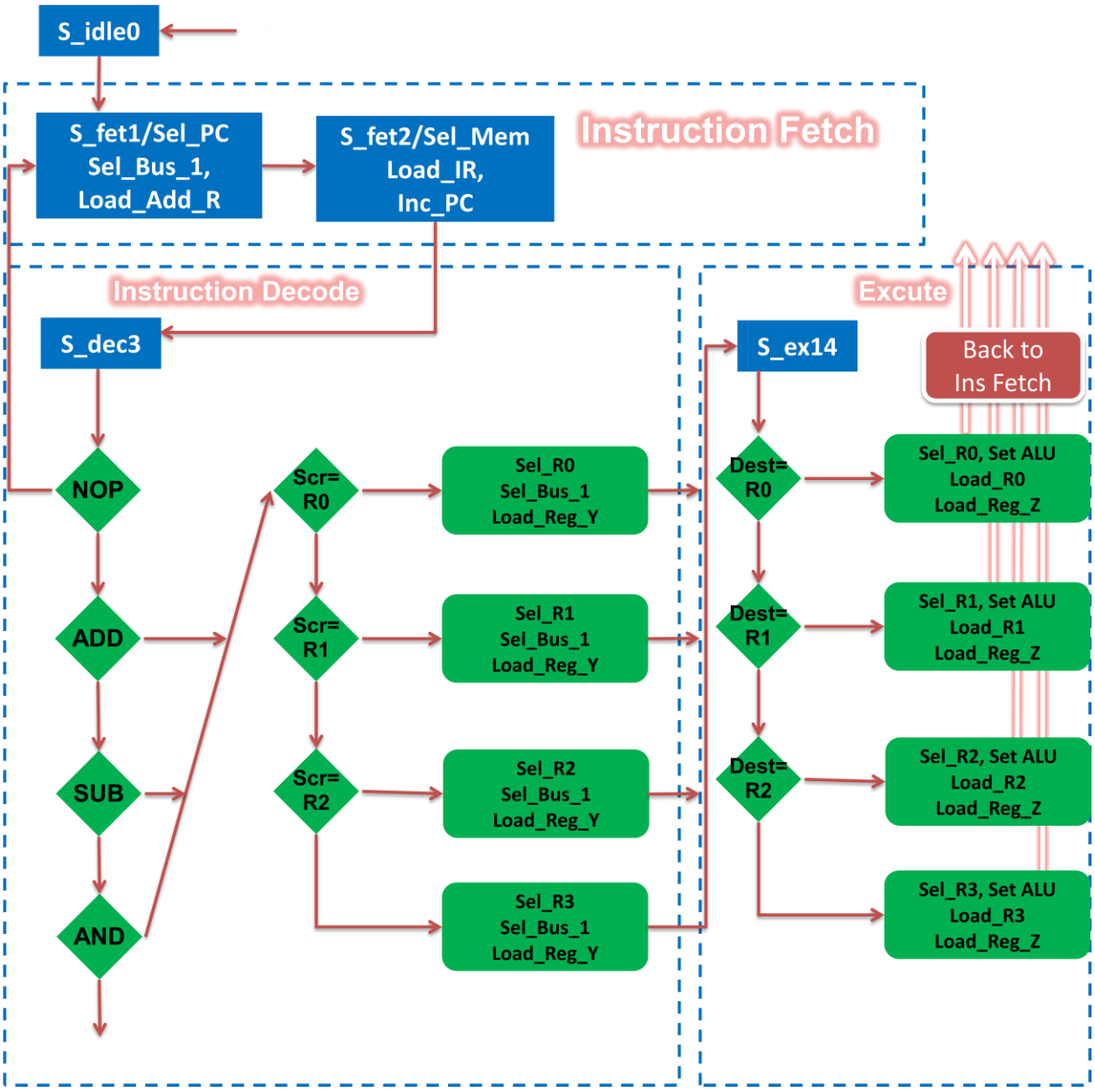


b) Các trạng thái :

Trạng thái	Mô tả chức năng
S_idle	Trạng thái lúc reset
S_fet1	Thực hiện load giá trị PC vào thanh ghi address của memory
S_fet2	Load giá trị memory[address] vào trong thanh ghi lệnh IR
S_fet3	Trạng thái dùng để tránh hiện tượng delay khi đọc Block RAM
S_dec	Giải mã lệnh chứa trong IR, giải mã address và constant ở các câu lệnh jump, store, load, save
S_ex1	Thực thi lệnh có liên quan đến ALU
S_rd1	Load địa chỉ vào memory[address]
S_rd2	Đọc giá trị lên R0
S_wr1	Ghi giá trị R0 vào memory[address]
S_jump1	Thực hiện lệnh jump address
S_jump2	
S_halt	Dừng toàn bộ hoạt động của hệ thống
S_nop	Không làm gì và chờ đến clock kế tiếp

c) Ví dụ:

Ví dụ về sự chuyển đổi trạng thái và các tín hiệu điều khiển của Controller khi thực hiện các lệnh NOP, ADD, SUB, AND :



4. Hiện thực:

- **Tiến trình hiện thực trên ModelSim:**
  - Viết file testbench
  - Compile và chạy mô phỏng
  - Xem kết quả (các file ảnh đính kèm)

- **Tiến trình hiện thực trên board DE2:**
  - ProgramCode cho vi điều khiển được viết trong file code.txt dưới dạng mã ASM
  - Trình biên dịch compiler.exe sẽ tạo ra file code.mif
  - File code.mif dùng để initial blockRAM
  - Nạp file RISC_processor.sof xuống board DE2 bằng Quartus
  - Xem kết quả: mỗi chu kì ứng với 1s

III. Kết quả

- Đã thiết kế được một vi xử lý đơn giản, tương đối hoàn thiện.
- Đáp ứng được yêu cầu đề bài, hiện thực các lệnh: ADD, SUB, AND, OR, NOT, JUMP, LOAD, STORE
- Mở rộng thêm các lệnh: NOP, SAVE, SIZ
- Đối với mô phỏng:
  - Test các module bộ phận (ALU, Control_Unit, Processing_Unit, Memory_Unit) và tổng quát (RISC_SPM) trên giả lập ModelSim.
  - Chạy được chương trình trên ModelSim (theo code)
- Đối với board DE2:
  - Chạy được trên mạch thật với Memory là Mảng initial
  - Chạy được một số lệnh với block RAM thực
- Viết chương trình compiler bằng C++ , dịch ra file code.mif dùng cho BlockRAM

IV. Kết luận

- ✓ Kiến trúc RISC đơn giản để hiện thực một vi điều khiển
- ✓ Hiện thực theo mô hình State-Machine dễ kiểm soát lỗi và phát triển
- ✓ Tập lệnh cố định 10bit đơn giản và có thể được nâng cấp

V. Phụ lục

Ảnh testbench (file đính kèm)