FPGA Application Week 10 Homework

CYEE 10828241 Chen Da-Chuan

May 1, 2023

Contents

List of Figures

List of Tables

1 Homework 10-1 Read Only Memory

1.1 Objective

Create a ROM module storing 16-bit data.

1.2 Operation

Table 1: Homework 10-1 Operation detail

type	var	operation
input	addr	Address of data stored in ROM to be read
input	clk	Clock signal
output reg	q	Read ROM data

1.3 Code

1. Figure 1 Line13-13: Declare register to store ROM data.
2. Figure 1 Line23-26: Load data into ROM from file.
3. Figure 1 Line28-29: Output the specified data from ROM.
4. Figure 2 Line1-5: 5 instructions to be stored in ROM.
5. Figure 3 Line10-14: Set initial value of address and clock.
6. Figure 3 Line16-16: Add specified address by 1 every 100ns.
7. Figure 3 Line17-17: Alternate clock signal every 50ns.

```verilog
// Quartus Prime Verilog Template
// Single Port ROM

module rom_v
#(parameter DATA_WIDTH=16, parameter ADDR_WIDTH=8)
(
    input [(ADDR_WIDTH-1):0] addr,
    input clk,
    output reg [(DATA_WIDTH-1):0] q
);

    // Declare the ROM variable
    reg [DATA_WIDTH-1:0] rom[2**ADDR_WIDTH-1:0];

    // Initialize the ROM with $readmemb.  Put the memory contents
    // in the file single_port_rom_init.txt.  Without this file,
    // this design will not compile.

    // See Verilog LRM 1364-2001 Section 17.2.8 for details on the
    // format of this file, or see the "Using $readmemb and $readmemh"
    // template later in this section.

    initial
    begin
        $readmemb("rom_v16.txt", rom);
    end

    always @ (posedge clk)
    begin
        q <= rom[addr];
    end

endmodule

```

Figure 1: Main file

```
0000111100001111
0000111011110000
1010101010101010
1010101001010101
1010000011110000
```

Figure 2: ROM data

```verilog
`timescale 1ns/1ns
module test_rom_v;

reg  [7:0]  addr;
reg         clk;
wire [15:0] q;

rom_v DUT(.addr(addr), .clk(clk), .q(q));

initial
    begin
        addr = 0;
        clk = 0;
    end

always #100 addr = addr + 1;
always #50 clk = ~clk;

endmodule
```

Figure 3: Test file

1.4 Result

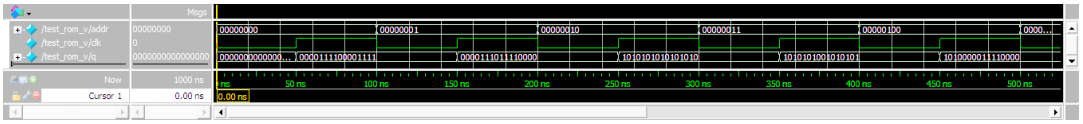

Figure 4: Homework 10-1 Result

Table 2: Exercise 10-1 Truth Table

clk ↑ (ns)	50	150	250	350
addr	00000000	00000001	00000010	00000011
q	0000111100001111	0000111011110000	1010101010101010	1010101001010101
description	q=rom[0] pass	q=rom[1] pass	q=rom[2] pass	q=rom[3] pass

2 Homework 10-2 Data Direct System

2.1 Objective

Create a data direct system selecting data either from ROM or register.

2.2 Operation

Table 3: Homework 10-2 Operation detail

type	var	operation
input	DA	Input data 1 from register
input	DB	Input data 2 from register
input	ROMX	Input data 2 from ROM
input	clock	Clock signal
input	dir	Data direction, selects either from register or ROM
output	DATAA	Output data 1
output	DATAB	Output data 2

2.3 Code

1. Figure 5 Line10-11: When triggered by "dir", "DB", or "ROMX", if "dir" is 1, select data from ROM.
2. Figure 5 Line12-13: When triggered by "dir", "DB", or "ROMX", if "dir" is 0, select data from register.
3. Figure 5 Line16-20: Output received and selected data.
4. Figure 6 Line9-17: Set initial values, and set "dir" as 1 at 500ns.
5. Figure 6 Line19-22: Alternate "clock" signal every 50ns, add "DA", "DB", "ROMX" by 1 every 100ns.

```
1  module direct_v (DA, DB, ROMX, clock, dir, DATAA, DATAB);
2  input [7:0] DA, DB, ROMX;
3  input clock, dir;
4  output reg[7:0] DATAA, DATAB;
5
6  reg [7:0] tempDB;
7
8  always@(dir or DB or ROMX)
9  begin
10     if (dir == 1)
11         tempDB <= ROMX;
12     else
13         tempDB<=DB;
14 end
15
16 always@(posedge clock)
17 begin
18     DATAA <= DA;
19     DATAB <= tempDB;
20 end
21
22 endmodule
```

Figure 5: Main file

```
1  `timescale 1ns/1ns
2  module test_direct;
3  reg [7:0]  DA, DB, ROMX;
4  reg        clock, dir;
5  wire [7:0] DATAA, DATAB;
6
7  direct_v DUT(.DA(DA), .DB(DB), .ROMX(ROMX), .clock(clock), .dir(dir), .DATAA(DATAA), .DATAB(DATAB));
8
9  initial
10     begin
11         DA = 8'h86;
12         DB = 8'h68;
13         ROMX = 8'h49;
14         clock = 0;
15         dir = 0;
16         #500 dir = 1;
17     end
18
19 always #50 clock = ~clock;
20 always #100 DA = DA + 1;
21 always #100 DB = DB + 1;
22 always #100 ROMX = ROMX + 1;
23
24 endmodule
```

Figure 6: Test file

2.4 Result

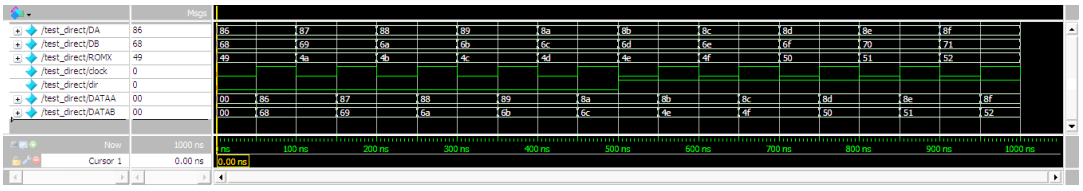

Figure 7: Homework 10-2 Result

Table 4: Exercise 10-2 Truth Table

clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
DA (8'h)	86	86	88	89	8a	8b	8c	8d	8e	8f
DB (8'h)	68	69	6a	6b	6c	6d	6e	6f	70	71
ROMX (8'h)	49	4a	4b	4c	4d	4e	4f	50	51	52
dir	0	0	0	0	0	1	1	1	1	1
DATAA (8'h)	86	87	88	89	8a	8b	8c	8d	8e	8f
DATAB (8'h)	68	69	6a	6b	6c	4e	4f	50	51	52
description	DATAA=DA DATAB=DB pass	DATAA=DA DATAB=DB pass	DATAA=DA DATAB=DB pass	qDATAA=DA DATAB=DB pass	DATAA=DA DATAB=DB pass	DATAA=DA DATAB=ROMX pass	DATAA=DA DATAB=ROMX pass	DATAA=DA DATAB=ROMX pass	DATAA=DA DATAB=ROMX pass	DATAA=DA DATAB=ROMX pass

3 Homework 10-3 Write Address Control System

3.1 Objective

Create a write address control system, which stores the writing address of the data processed by ALU.

3.2 Operation

Table 5: Homework 10-3 Operation detail

type	var	operation
input	R3_0	Address option 1
input	R11_8	Address option 2
input	clk	Clock signal
input	dir	Direction signal, selecting either address 1 or 2
output	addr	Output/selected address

3.3 Code

1. Figure 8 Line6-12: When triggered by clock, if "dir" is 0, output "R3_0", else output "R11_8".
2. Figure 9 Line9-15: Set initial values.
3. Figure 9 Line17-20: Alternate clock signal every 50ns, and add 1 to "R3_0", "R11_8", "dir" every 100ns.

```
1  module writeaddr_v (R3_0, R11_8, clk, dir, addr);
2  input  [3:0]     R3_0, R11_8;
3  input           clk, dir;
4  output reg [3:0] addr;
5
6  always@(posedge clk)
7  begin
8      if (dir == 0)
9          addr = R3_0;
10     else
11         addr = R11_8;
12 end
13
14 endmodule
```

Figure 8: Main file

```
1  `timescale 1ns/1ns
2  module test_writeaddr;
3  reg  [3:0]  R3_0, R11_8;
4  reg         clk, dir;
5  wire [3:0]  addr;
6
7  writeaddr_v DUT(.R3_0(R3_0), .R11_8(R11_8), .clk(clk), .dir(dir), .addr(addr));
8
9  initial
10     begin
11         R3_0 = 4'h6;
12         R11_8 = 4'h3;
13         clk = 0;
14         dir = 0;
15     end
16
17 always #50 clk = ~clk;
18 always #100 R3_0 = R3_0 + 1;
19 always #100 R11_8 = R11_8 + 1;
20 always #100 dir = dir + 1;
21
22 endmodule
23
```

Figure 9: Test file

3.4 Result

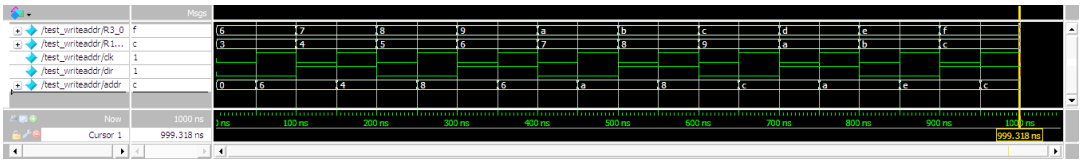

Figure 10: Homework 10-3 Result

Table 6: Exercise 10-3 Truth Table

clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
R3_0 (8'h)	6	7	8	9	a	b	c	d	e	f
R11_8 (8'h)	3	4	5	6	7	8	9	a	b	c
dir	0	1	0	1	0	1	0	1	1	1
addr (8'h)	6	4	8	6	a	8	c	a	e	c
description	addr=R3_0 pass	addr=R11_8 pass	addr=R3_0 pass	addr=R11_8 pass	addr=R3_0 pass	addr=R11_8 pass	addr=R3_0 pass	addr=R11_8 pass	addr=R3_0 pass	addr=R11_8 pass

4 Homework 10-4 Full Data Pipeline

4.1 Objective

Fully combine all of the previous coded modules to create a full data pipeline. Beside this, the relating modules are modified to accommodate $2^4 = 16$ 8-bit data in register.

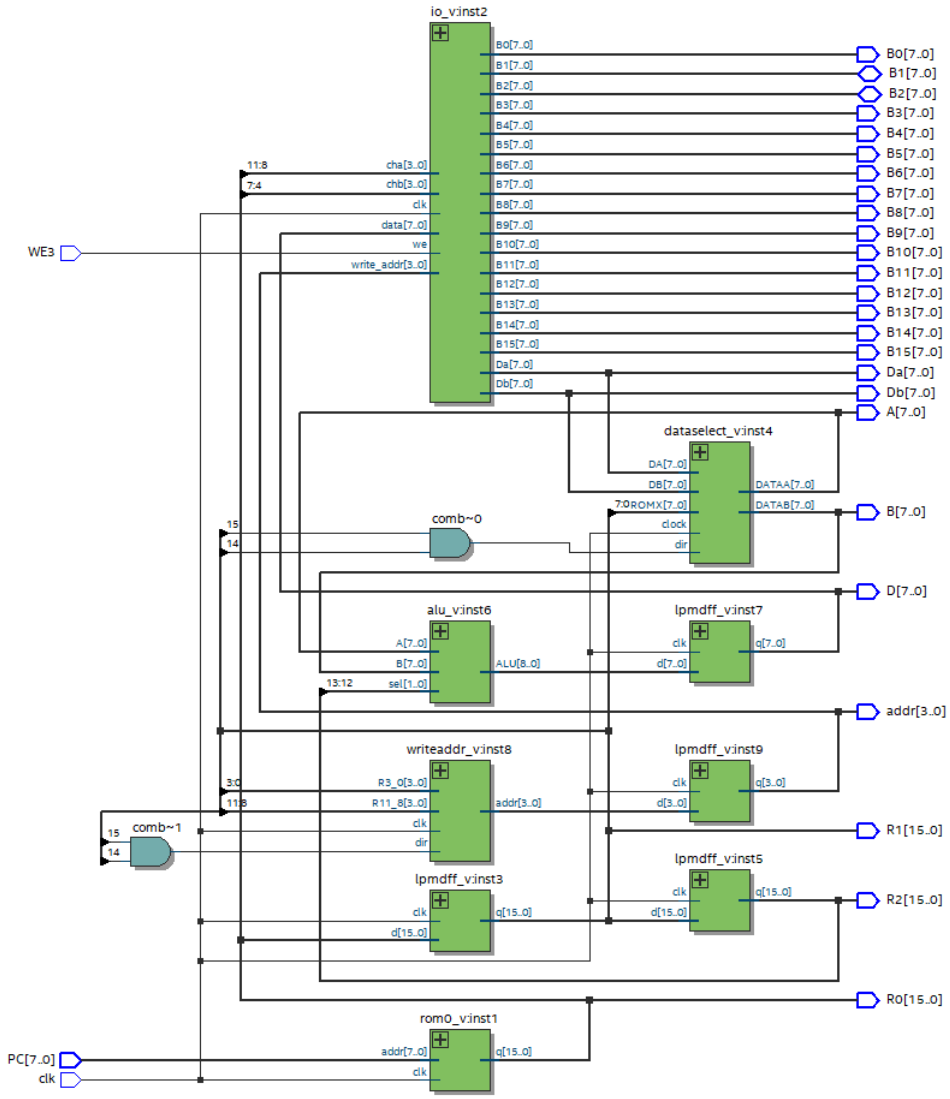

Figure 11: Homework 10-4 Data Pipeline

4.2 Operation

Table 7: Homework 10-4 Operation detail

type	var	operation
input	WE3	write enable, which writes data into register
input	PC	Address of data stored in ROM to be read
input	clk	Clock signal
output	R0	ROM output stage 0
output	R1	ROM output stage 1
output	R2	ROM output stage 2
output	B0	RAM address 1
output	B1	RAM address 2
output	B2	RAM address 3
output	B3	RAM address 4
output	B4	RAM address 5
output	B5	RAM address 6
output	B6	RAM address 7
output	B7	RAM address 8
output	B8	RAM address 9
output	B9	RAM address 10
output	B10	RAM address 11
output	B11	RAM address 12
output	B12	RAM address 13
output	B13	RAM address 14
output	B14	RAM address 15
output	B15	RAM address 16
output	Da	Register output data 1
output	Db	Register output data 2
output	A	ALU input data 1
output	B	ALU input data 2
output	D	ALU output data
output	addr	Where to write ALU output data

4.3 Code

1. Figure 12 Line16-18: ROM, which stores commands by PC and outputs to register.
2. Figure 12 Line21-24: Data register & I/O system, which stores data and outputs to data driven system.
3. Figure 12 Line28-29: Command register, holds command from ROM for 1 clock cycle and outputs to next command register.
4. Figure 12 Line32-33: Data driven system, which receives data from data register & I/O system and command register, and outputs to ALU.
5. Figure 12 Line37-38: Command register, holds command from previous command register for 1 clock cycle and outputs to write address control system.
6. Figure 12 Line41-41: ALU, which receives data from data driven system and outputs to data register.
7. Figure 12 Line45-46: Data register, which receives data from ALU and outputs to data register & I/O system.
8. Figure 12 Line49-50: Write address control system, which receives command and outputs to address register.
9. Figure 12 Line54-55: Address register, which receives address from write address control system and outputs to data register & I/O system.
10. Figure 13 Line11-14: Include data pipeline module.

11. Figure 13 Line16-22: Initialize "WE3", "clk", "PC" signals. Which controls the data pipeline.
12. Figure 13 Line23-23: Alternate "clk" signal every 50ns.
13. Figure 13 Line24-24: Add 1 to "PC" every 100ns.
14. Figure 13 Line27-28: Set the tri-state gate to be open or closed when internal control is not yet activated.

```
1  module data_v  (WE3,PC,clk,R0,B0,B1,B2,B3,B4,R1,R2,Da,Db,
2                  D, A, B,addr, B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15);
3   input WE3, clk;
4   input [7:0] PC;
5   inout [7:0] B1, B2;
6   output [15:0] R0,R1,R2;
7   output [7:0] B0, B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15;
8   output [7:0] Da,Db, A, B,D;
9
10  wire [8:0] ALU;
11  wire [3:0] addr0;
12  output [3:0] addr;
13
14
15
16  rom0_v inst1(.addr(PC), .clk(clk), .q(R0));
17  defparam inst1.DATA_WIDTH=16;
18  defparam inst1.ADDR_WIDTH= 8;
19
20
21  io_v inst2(.clk(clk), .we(WE3), .write_addr(addr),.cha(R0[11:8]), .chb(R0[7:4]
22             .data(D),.B0(B0), .B1(B1),.B2(B2),.B3(B3),.B4(B4),
23             .B5(B5),.B6(B6),.B7(B7),.B8(B8), .B9(B9), .B10(B10),.B11(B11),
24             .B12(B12), .B13(B13), .B14(B14), .B15(B15), .Da(Da), .Db(Db));
25
26
27
28   lpmdff_v inst3(.d(R0),.clk(clk), .q(R1));
29    defparam inst3.DATA_WIDTH=16;
30
31
32  dataselect_v inst4(.DA(Da),.DB(Db), .ROMX(R1[7:0]), .clock(clk),
33                  .dir(R1[14]&R1[15]), .DATAA(A), .DATAB(B));
34
35
36
37   lpmdff_v inst5(.d(R1),.clk(clk), .q(R2));
38    defparam inst5.DATA_WIDTH=16;
39
40
41   alu_v inst6(.A(A), .B(B), .sel(R2[13:12]), .ALU(ALU));
42
43
44
45   lpmdff_v inst7(.d(ALU[7:0]),.clk(clk), .q(D));
46    defparam inst7.DATA_WIDTH=8;
47
48
49  writeaddr_v inst8(.R3_0(R1[3:0]),.R11_8(R1[11:8]),.clk(clk),
50                .dir(R1[14]&R1[15]),.addr(addr0));
51
52
53
54   lpmdff_v inst9(.d(addr0), .clk(clk),.q(addr));
55    defparam inst9.DATA_WIDTH=4;
56
57
58
59   endmodule
60
```

Figure 12: Main file

```
1
2   `timescale 1 ns/1 ns
3   module test;
4   reg WE3, clk;
5   reg [7:0] PC;
6   wire [7:0] B1, B2;
7   wire [15:0] R0,R1,R2;
8   wire [7:0] B0, B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15;
9   wire [7:0] Da,Db, A, B,D;
10
11  data_v DUT( .WE3(WE3), .clk(clk),.PC(PC), .B1(B1), .B2(B2),.R0(R0),.R1(R1),
12              .R2(R2), .B0(B0), .B3(B3),.B4(B4), .B5(B5) ,.B6(B6), .B7(B7),
13              .B8(B8), .B9(B9), .B10(B10) , .B11(B11), .B12(B12) ,.B13(B13),
14              .B14(B14) , .B15(B15), .Da(Da) , .Db(Db), .A(A), .B(B), .D(D));
15
16   initial
17  begin
18     WE3 = 1 ;
19     clk = 0;
20     PC = 0;
21
22   end
23  always #50 clk = ~clk  ;
24  always #100 PC = PC + 1 ;
25
26
27  assign B1 = (B0[0]==1)? 8'bzzzzzzzz : 8'h40;
28  assign B2 = (B0[1]==1)? 8'bzzzzzzzz : 8'h88;
29
30  endmodule
31
```

Figure 13: Test file

4.4 Result

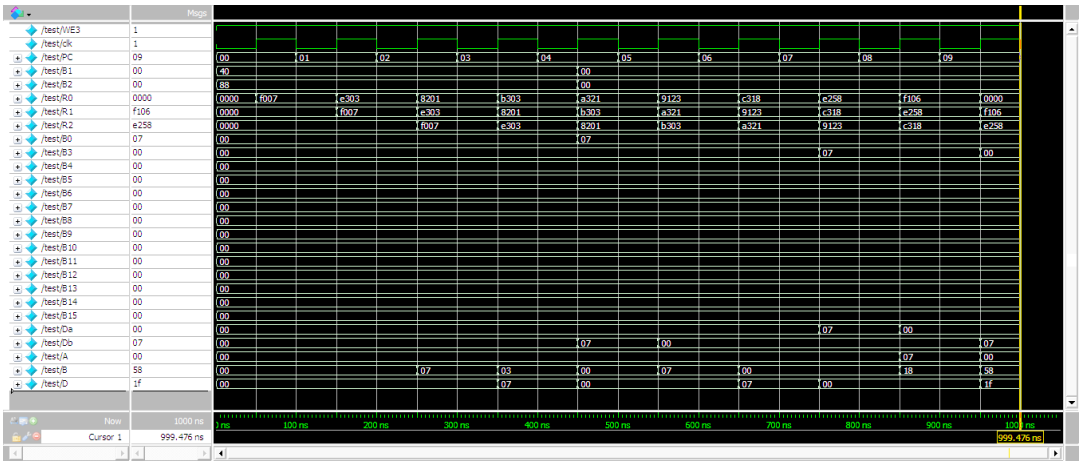

Figure 14: Homework 10-4 Result

Table 8: Exercise 10-4 Truth Table

clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
PC (8'h)	00	01	02	03	04	05	06	07	08	09
R0 (8'h)	f007	e303	8201	b303	a321	9123	c318	e258	f106	0000
R1 (8'h)	0000	f007	e303	8201	b303	a321	9123	c318	e258	f106
R2 (8'h)	0000	0000	f007	e303	8201	b303	a321	9123	c318	e258
A (8'h)	00	00	00	00	00	00	00	00	07	00
B (8'h)	00	00	07	03	00	07	00	00	18	28
D (8'h)	00	00	00	07	00	00	07	00	00	1f
B0 (8'h)	00	00	00	00	00	07	07	07	07	07
description	S1 RCf007 S2 N/A S3 N/A S4 N/A S5 N/A pass	S1 RCe303 S2 RA0 S3 N/A S4 N/A S5 N/A pass	S1 RC8201 S2 RA3 S3 PD07 S4 N/A S5 N/A pass	S1 RCb303 S2 RA2 S3 PD03 S4 A0|D07=D07 S5 N/A pass	S1 RCa321 S2 RA3 S3 PA0 S4 A3&D03=D00 S5 WA0 pass	S1 RC9123 S2 RA3 S3 PA0 S4 A2+A0=D00 S5 WA3 pass	S1 RCc318 S2 RA1 S3 PA2 S4 A3|A0=D07 S5 WA1 pass	S1 RCe258 S2 RA3 S3 PA2 S4 A3&A2=D00 S5 WA3 pass	S1 RCf106 S2 RA2 S3 PD18 S4 A1−A2=D00 S5 WA1 pass	S1 RC0000 S2 RA1 S3 PD58 S4 A3+D18=D1f S5 WA3 pass

"S": stage, "R": read, "W": write, "P": pick/choose, "C": command, "A": address, "D": data, "N/A": not available, "+": add operation, "−": minus operation, "&": and operation, "|": or operation

5 Homework 10-5 Instruction Explanation

"A4'd0100" stands for 4-bit binary address, "D8'd0000_1111" stands for 8-bit binary data.

- **16'hF40F** = 16'b1111_0100_0000_1111 = 2'b11 + 2'b11 + 4'b0100 + 8'b0000_1111
  = Execute (A4'b0100)|(D8'b0000_1111) Write in A4'b0100
- **16'hE300** = 16'b1110_0011_0000_0000 = 2'b11 + 2'b10 + 4'b0011 + 8'b0000_0000
  = Execute (A4'b0011)&(D8'b0000_0000) Write in A4'b0011
- **16'hF4FF** = 16'b1111_0100_1111_1111 = 2'b11 + 2'b11 + 4'b0100 + 8'b1111_1111
  = Execute (A4'b0100)|(D8'b1111_1111) Write in A4'b0100
- **16'hF3FF** = 16'b1111_0011_1111_1111 = 2'b11 + 2'b11 + 4'b0011 + 8'b1111_1111
  = Execute (A4'b0011)|(D8'b1111_1111) Write in A4'b0011
- **16'hD401** = 16'b1101_0100_0000_0001 = 2'b11 + 2'b01 + 4'b0100 + 8'b0000_0001
  = Execute (A4'b0100)−(D8'b0000_0001) Write in A4'b0100

6 Appendix

6.1 ALU Operation Code

Table 9: ALU Operation

code	operation
00	D=A+B
01	D=A-B
10	D=A&B
11	D=A|B

"A": input data 1, "B": input data 2, "D": output 1

6.2 Instruction Format

Table 10: Instruction Format

register hex	ro[15:12]				ro[11:08]				ro[07:04]				ro[03:00]			
register bin	ro15	ro14	ro13	ro12	ro11	ro10	ro09	ro08	ro07	ro06	ro05	ro04	ro03	ro02	ro01	ro00
instruction type A	1	1	ALU operation code		read A / write D address				B data							
instruction type B	1	0	ALU operation code		read A address				read B address				write D address			

"A": input data 1, "B": input data 2, "D": output 1

6.3 Pipeline CPU Stages

Table 11: Pipeline CPU Stages

clock cycle	1	2	3	4	5	6
stage 1 Fetch Command	F007	E300	F44F	F3FF	D401	F007
stage 2 Fetch Data		F007	E300	F44F	F3FF	D401
stage 3 Fetch ALU Operation			F007	E300	F44F	F3FF
stage 4 Execute				F007	E300	F44F
stage 5 Write Back					F007	E300

6.4 CPU State Machine

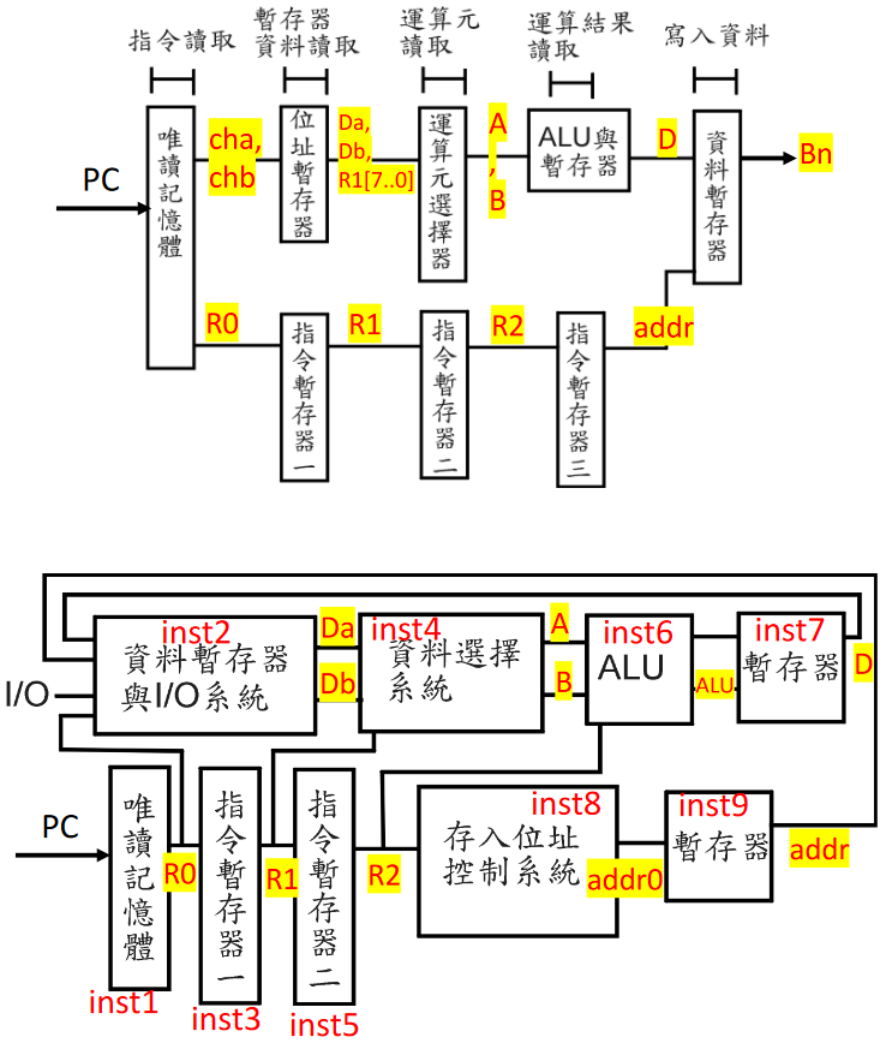

Figure 15: CPU steps