

C o m p u t e r O r g a n i z a t i o n



Lab10 CPU(2) IFetch, Data-Memory



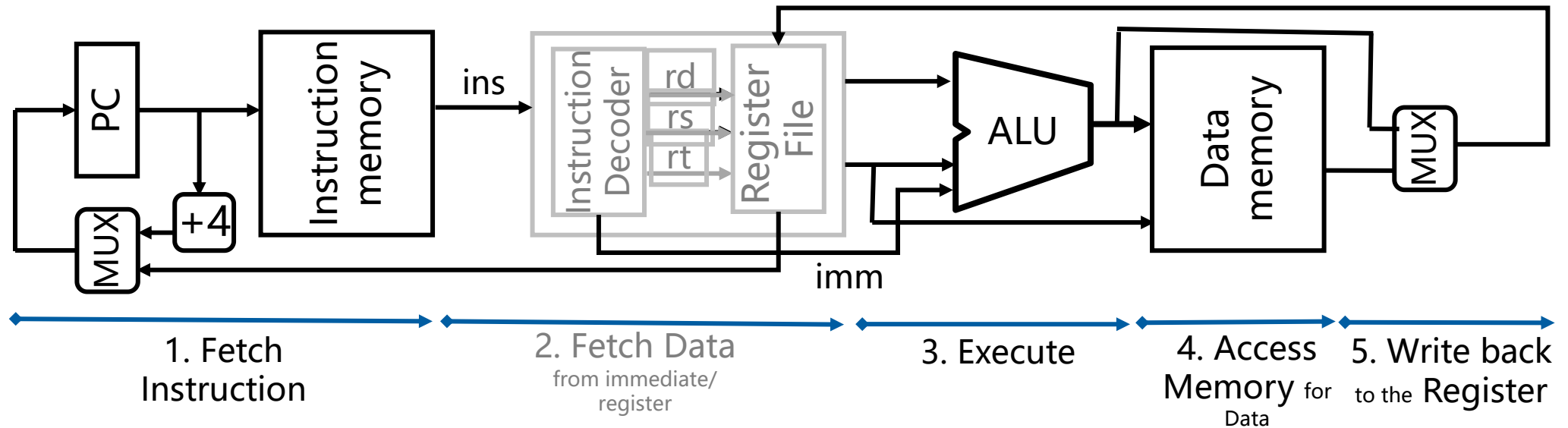
2

Topics

➤ CPU(2) -DataPath (2)

➤ Data-Memory

➤ IFetch



```
input clock;    // Clock signal
```

```
input memWrite;
```

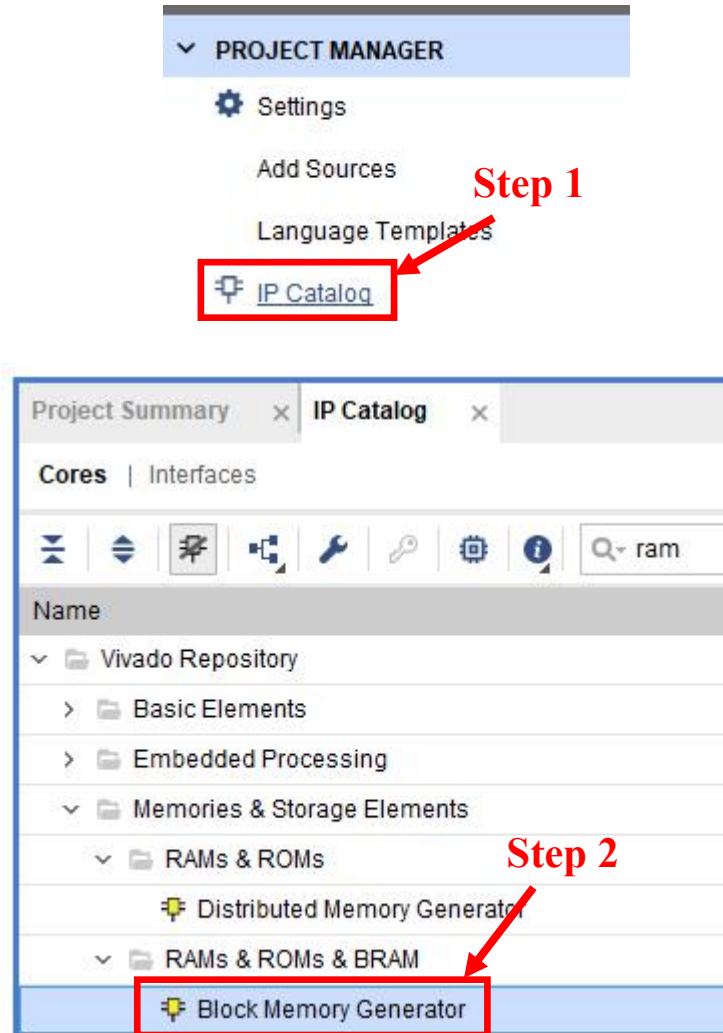
```
input[31:0] address;
```

```
input[31:0] writeData;
```

```
output[31:0] readData;
```

Using IP core Block Memory

Using the IP core Block Memory of Xilinx to implement the data memory.



Import the IP core in vivado project

1) in “**PROJECT MANAGER**” window
click “**IP Catalog**”

2) in “**IP Catalog**” window

> Vivado Repository

> Memories & Storage Elements

> RAMs & ROMs & BRAM

> **Block Memory Generator**

Customize Memory IP core

The screenshot shows the configuration interface for a Memory IP core. The 'Component Name' is set to 'RAM'. The 'Basic' tab is selected. Under 'Basic', 'Interface Type' is 'Native' and 'Memory Type' is 'Single Port RAM'. In the 'ECC Options' section, 'ECC Type' is 'No ECC'. In the 'Write Enable' section, 'Byte Write Enable' is unchecked and 'Byte Size (bits)' is '9'. In the 'Algorithm Options' section, 'Algorithm' is 'Minimum Area' and 'Primitive' is '8kx2'.

Component Name: RAM

Basic | Port A Options | Other Options | Summary

Interface Type: Native | ☐ Generate address interface with 32 bits

Memory Type: Single Port RAM | ☐ Common Clock

ECC Options

ECC Type: No ECC

☐ Error Injection Pins: Single Bit Error Injection

Write Enable

☐ Byte Write Enable

Byte Size (bits): 9

Algorithm Options

Defines the algorithm used to concatenate the block RAM primitives. Refer datasheet for more information.

Algorithm: Minimum Area

Primitive: 8kx2

Customize memory IP core

- 1) Component Name: **RAM**
- 2) Basic settings:
 - Interface Type: **Native**
 - Memory Type: **Single-port RAM**
 - ECC options: **no ECC check**
 - Algorithm options: **Minimum area**

Customize Memory IP core continued

Component Name RAM

Basic Port A Options Other Options Summary

Memory Size

Write Width 32 Range: 1 to 4608 (bits)

Read Width 32

Write Depth 16384 Range: 2 to 1048576

Read Depth 16384

Operating Mode Write First

Enable Port Type Always Enabled

Port A Optional Output Registers

☐ Primitives Output Register ☐ Core Output Register

☐ SoftECC Input Register ☐ REGCEA Pin

Port A Output Reset Options

☐ RSTA Pin (set/reset pin) Output Reset Value (Hex) 0

☐ Reset Memory Latch Reset Priority CE (Latch or Register Enable)

3) PortA Options settings:

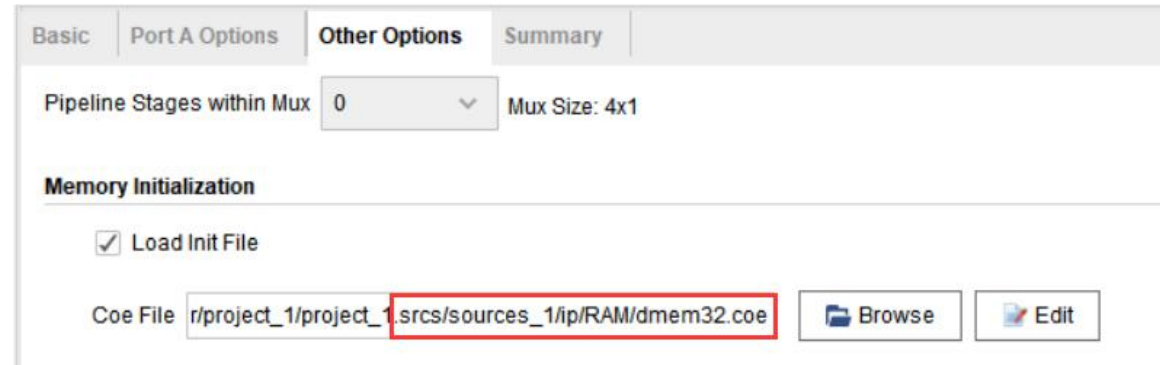
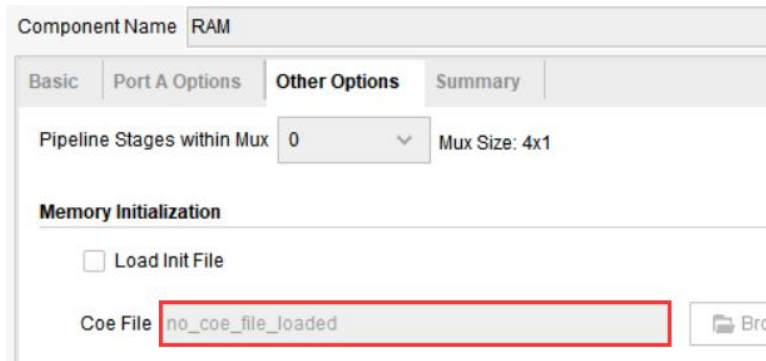
- Data read and write **bit width: 32 bits (4Byte)**
- Write/Read **Depth: 16384 (64KB)**
- Operating Mode: **Write First**
- Enable Port Type: **Always Enabled**
- PortA Optional Output Registers: **NOT SET**



Customize Memory IP core continued

4) Other Options settings:

- 1. When **specifying the initialization file** for customize the RAM on the 1st time, the IP core RAM just customized **WITHOUT initial file** and **corresponding path**, so set it to **no initial file** when creating RAM.
- 2. **After** the RAM IP core created
 - 2-1. **COPY** the initialization file **dmem32.coe** to **projectName.srscs/sources_1/ip/ComponentName**. (“projectName.srscs” is under the project folder, “componentName” here is RAM)
 - 2-2. Double-click the newly created RAM IP core, **RESET** it with the **initialization file**, select the **dmem32.coe** file that has been in the directory of projectName.srscs/sources_1/ip/RAM.



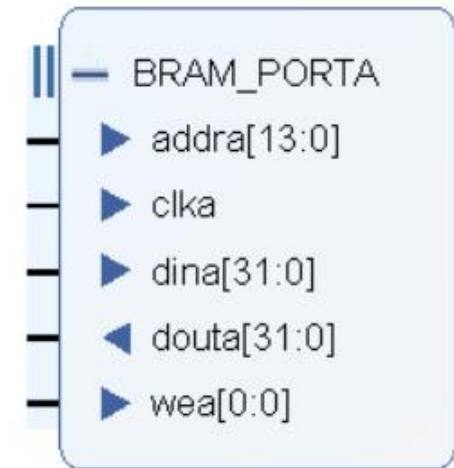
Tips: “dmem32.coe” file could be found in the directory “lab_tools” of course blackboard site

https://bb.sustech.edu.cn/webapps/blackboard/content/listContentEditable.jsp?content_id= 281670_1&course_id= 3602_1

Design Module With Memory IP Instanced

```
// Part of dmemory32 module
//Create a instance of RAM(IP core), binding the ports
RAM ram (
    .clka(clk),                // input wire clka
    .wea(memWrite),           // input wire [0 : 0] wea
    .addra(address[15:2]),     // input wire [13 : 0] addra
    .dina(writeData),         // input wire [31 : 0] dina
    .douta(readData)          // output wire [31 : 0] douta
);

/*The clock is from CPU-TOP, suppose its one edge has been used at the upstream module of data memory, such as IFetch,
Why Data-Memroy DO NOT use the same edge as other module ? */
assign clk = !clock;
```



Q: In the five stages of instruction processing, what operations must be arranged on the edge of the clock?
What's your design for a one-cycle CPU?

Function Verification

```
//The testbench module for dmemory32
module ramTb( );
reg clock = 1'b0;
reg memWrite = 1'b0;
reg [31:0] addr = 32'h0000_0010;
reg [31:0] writeData = 32'ha000_0000;
wire [31:0] readData;

dmemory32 uram
    (clock,memWrite,addr,writeData,readData);
always #50 clock = ~clock;

initial fork
    #120 memWrite = 1'b1;
    #200
        writeData = 32'h0000_00f5;
    #400
        memWrite = 1'b0;
    // ... to be completed
join

endmodule
```

NOTE:

Using bind port with name is Suggested!!

- 1) Set “**memWrite**” to 1'b0 means to read the data from the RAM unit identified by “**addr**”.
- 2) Set “**memWrite**” to 1'b1 and “**writeData**” to 0x0000_00f5 which means to write data 0xa000_00f5 to the RAM unit identified by “**addr**”.

Q1. While instance the module on page 3(module **dmemory32**(readData,address,writedata,memWrite,clock)) and using sequential binding as the testbench on the left hand, What will happen ?

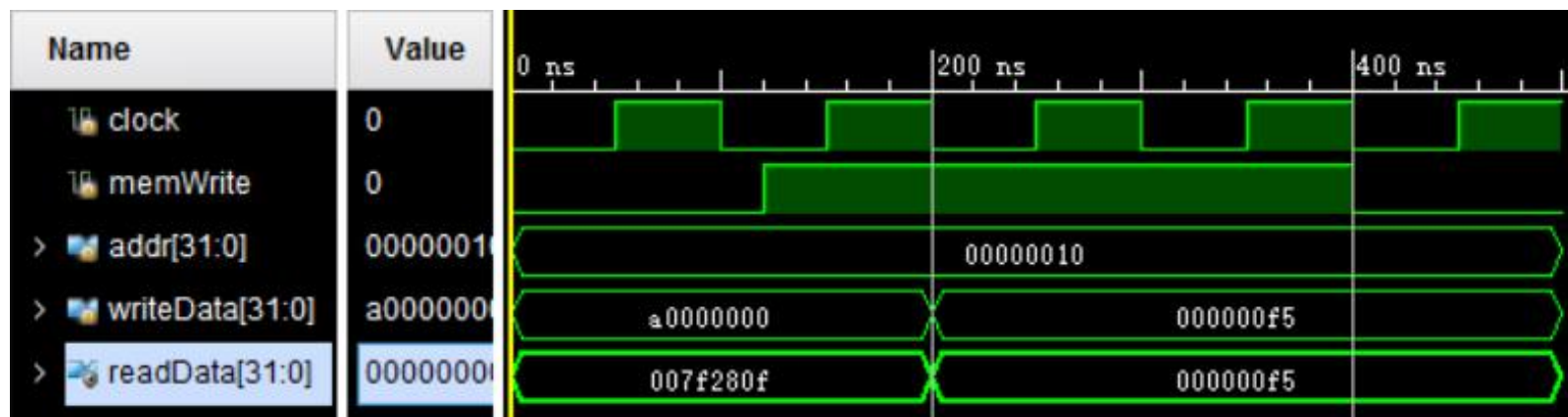
Q2. While the data has been written to the RAM unit, would it be recorded to the initial data file(dmем32.coe)?

Function Verification continued

```

1 memory_initialization_radix = 16;
2 memory_initialization_vector =
3 007f2812,
4 007f2811,
5 007f2810,
6 007f2810,
7 007f280f,
8 00000001,
9 00000002,
10 00000003,
11 00000005,
12 00000006,
13 00000007,
14 0000ffff,
15 00000000,
16 00000000,

```



Q1: On which edge of clock does the read and write operations occur? posedge or negedge?

Q2: What's value will be get while read the memory according to the “addr” 0x0000_0020?
how about “addr” 0x0000_0016?

Tips: “dmem32.coe” file could be found in the directory “lab_tools” of course blackboard site

https://bb.sustech.edu.cn/webapps/blackboard/content/listContentEditable.jsp?content_id= 281670_1&course_id= 3602_1

Practice1

1. Build the data memory module.
2. Verify its function by simulation(NOTE: The testbench on page 9 is JUST a reference)
 - **Read** the values one by one from memory unit where are specified in the red box of the screenshot on the right hand.
 - Write a word(value is 0x1000_0000) to the memory unit where is specified in the blue box of the screenshot on the right hand, then read it out.
3. List all the signals which are needed for data-memory module

```
1 memory_initialization_radix = 16;
2 memory_initialization_vector =
3 007f2812,
4 007f2811,
5 007f2810,
6 007f2810,
7 007f280f,
8 00000001,
9 00000002,
10 00000003,
11 00000005,
12 00000006,
13 00000007,
14 0000ffff,
15 00000000,
16 00000000,
17 00000000,
18 00000000,
19 00000000,
20 00000000,
21 00000000,
```

read these initial value

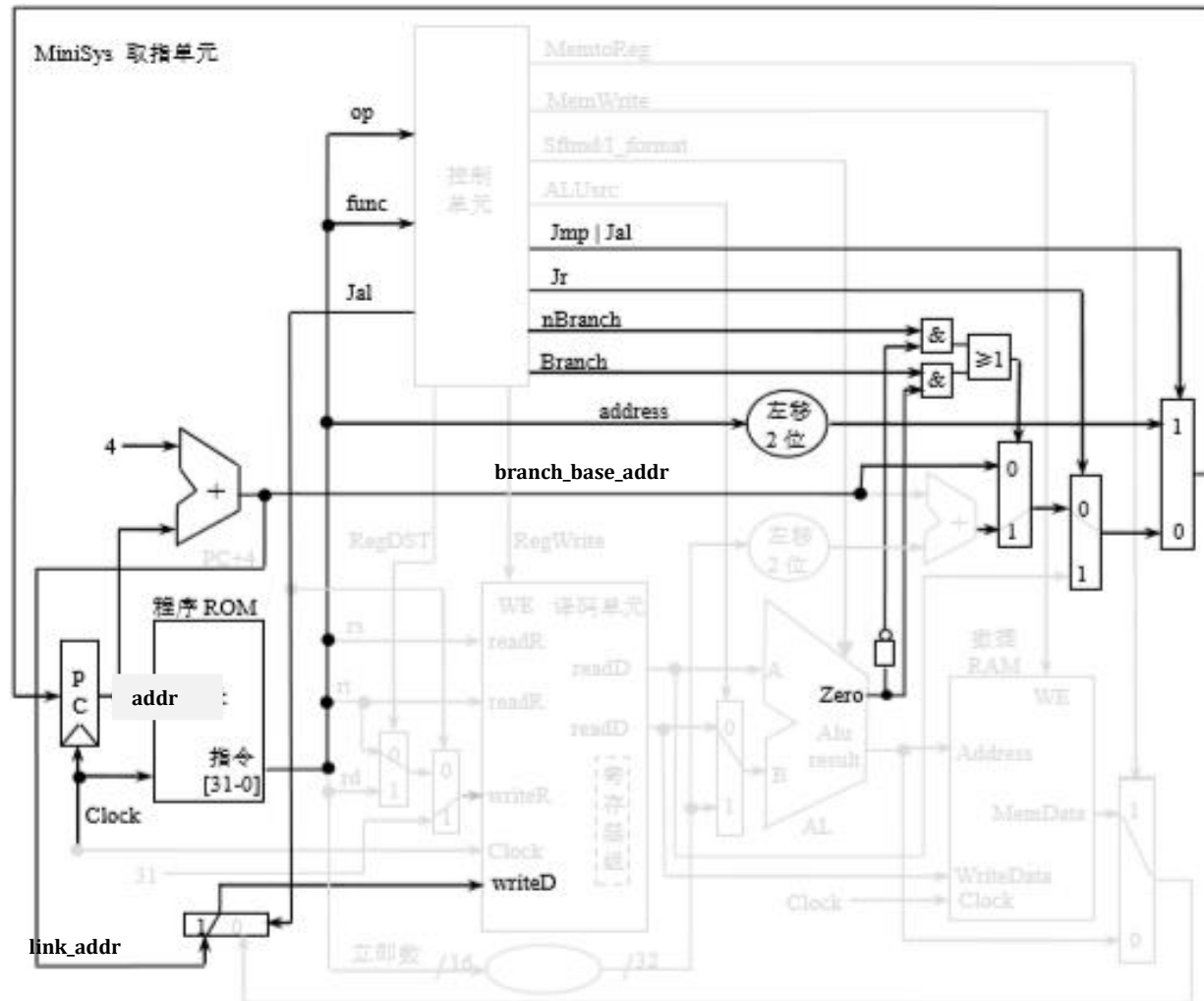
write this word with 0x1000_0000 then read it

| name | from | to | bits | function |
|-------------|-------------|-------------|------|---|
| clock | CPU-TOP | Data Memory | 1 | data memory write is sensitive with its negedge |
| rdata | Data Memory | Decoder | 32 | the word read from the data memory and send to decoder |
| memoryWrite | Controller | Data Memory | 1 | 1'b1 means to write the memory unit, else means not to write |
| address | ALU | Data Memory | 32 | the address which is used to identify the memory unit to be read or written |
| ... | | | | |

Tips: “**dmem32.coe**” file could be found in the directory “**lab_tools**” of course blackboard site

https://bb.sustech.edu.cn/webapps/blackboard/content/listContentEditable.jsp?content_id= 281670_1&course_id= 3602_1

Instruction Fetch



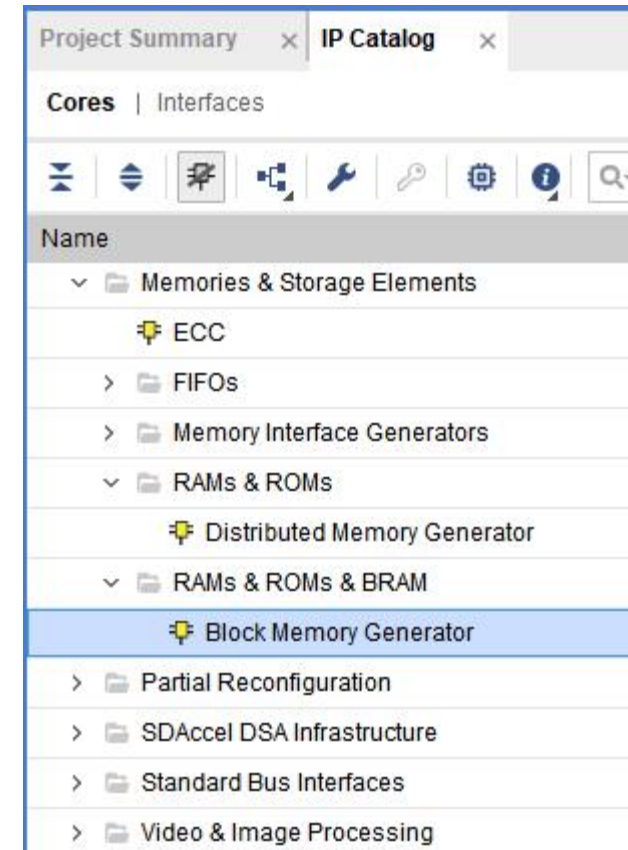
Instruction Fetch

- 1. **Store** the instructions(machine-code)
- 2. **Update** the value of the PC register
 - Reset
 - PC+4
 - Update the value of the PC register according to the jump instructions
 - branch(beq,bne) [I-type]
 - jal, j [J-type]
 - jr [R-type]
- 3. **Fetch** the instructions according to the value of the PC register

Using IP core As Instruction Memory

- **Step1: Find the IP core(Block Memory Generator) in IP Catalog**
- **Step2: Customize the IP core**
 - set **name**(component name), **type(ROM)**
 - set features of the ROM(**width** and **depth**), **operation mode** and **register output**
 - set **initial file**
- **Step3: Generate** the IP core, then it will be added to vivado project automatically

Tips: The setting steps of ROM IP core are same with which of the RAM IP core in Data-memory



Customize the IP core

Component Name **prgrom**

Basic | Port A Options | Other Options | Summary

Interface Type: Native

Memory Type: **Single Port ROM**

ECC Options

ECC Type: No ECC

Error Injection Pins: Single Bit Error Injection

Write Enable

Byte Write Enable

Byte Size (bits): 9

Algorithm Options

Defines the algorithm used to concatenate the block RAM primitives. Refer datasheet for more information.

Algorithm: Minimum Area

Primitive: 8kx2

Component Name **prgrom**

Basic | **Port A Options** | Other Options | Summary

Memory Size

Port A Width: 32 (Range: 1 to 4608 (bits))

Port A Depth: 16384 (Range: 2 to 1048576)

The Width and Depth values are used for Read Operation in Port A

Operating Mode: Write First

Enable Port Type: Always Enabled

Port A Optional Output Registers

Primitives Output Register

Core Output Register

SoftECC Input Register

REGCEA Pin

Port A Output Reset Options

RSTA Pin (set/reset pin) Output Reset Value (Hex): 0

Reset Memory Latch Reset Priority: CE (Latch or Register Enable)

READ Address Change A

Read Address Change A

Component Name **prgrom**

Basic | Port A Options | **Other Options** | Summary

Pipeline Stages within Mux: 0 Mux Size: 4x1

Memory Initialization

Load Init File

Coe File: no_coe_file_loaded

Browse Edit

Fill Remaining Memory Locations

Remaining Memory Locations (Hex): 0

Structural/UniSim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

Collision Warnings: All

Behavioral Simulation Model Options

Disable Collision Warnings

Disable Out of Range Warnings

NOTE: set the init file of prgrom after this IP core has been added into vivado project.
Same steps as the RAM IP core used in data memory.

Instance the IP core



```
prgrom instmem(  
    .clka(clock),  
    .addra(PC[15:2]),  
    .douta(Instruction)  
);
```

NOTES:

“**prgrom**” is the IP core which is generated in vivado follow the steps on page 13, 14 of this slides.

In One Cycle CPU, the process of **geting instrcutiion** should **happen** on the **posedge** of the clock. At this moment, IFetch module gets the instruction which is store at “**addra**” from the instruction memory “Instmem”

Q: Why using PC[15:2] instead of PC[13:0] to bind with port “addra”?

TIPS: The same reason as the address bus used in data memory

The Function Verification of “prgrom”

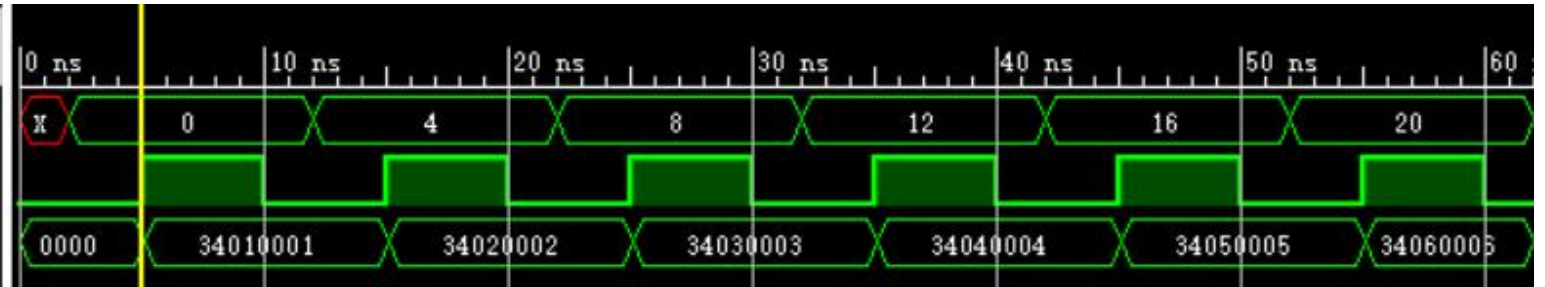
prgmip32.coe

```
1 memory_initialization_radix = 16;
2 memory_initialization_vector =
3 34010001,
4 34020002,
5 34030003,
6 34040004,
7 34050005,
8 34060006,
9 34070007,
10 34080008,
11 34090009,
12 340a000a,
13 340b000b,
14 340c000c,
```

Tips: “prgmip32.coe” file could be found in the directory “lab_tools” of course blackboard site

https://bb.sustech.edu.cn/webapps/blackboard/content/listContentEditable.jsp?content_id=_281670_1&course_id=3602_1

| Name | Value |
|---------------------|----------|
| > PC[31:0] | 0 |
| clock | 1 |
| > Instruction[31:0] | 00000000 |



```
module prgrom_tb( );    //a reference for the testbench ?
    reg[31:0] PC;
    reg clock=1'b0;
    wire [31:0] Instruction;
    prgrom instmem(.clka(clock),.addra(PC[15:2]),.douta(Instruction));
    always #5 clock = ~clock;
    initial begin
        clock = 1'b0;
        #2 PC = 32'h0000_0000;
        repeat(5) begin
            #10 PC = PC+4;
            #10 $finish;
        end
    end
end
endmodule
```

- Read the “Instruction” from “douta” port of Instruction memory “prgrom” on every posedge of the “clock”.
- In this testcase, the value of 'PC' is added with 4 each time.
- Q: How many instructions would be fetched in this testbench ?

IFetch Module

```
module IFetc32(Instruction, branch_base_addr, link_addr,
clock, reset,
Addr_result, Read_data_1, Branch, nBranch, Jmp, Jal, Jr, Zero);

    output[31:0] Instruction;           // the instruction fetched from this module
    output[31:0] branch_base_addr;     // (pc+4) to ALU which is used by branch type instruction
    output[31:0] link_addr;           // (pc+4) to Decoder which is used by jal instruction

    input        clock, reset;          // Clock and reset
// from ALU
    input[31:0] Addr_result;             // the calculated address from ALU
    input        Zero;                  // while Zero is 1, it means the ALUresult is zero

// from Decoder
    input[31:0] Read_data_1;            // the address of instruction used by jr instruction

// from Controller
    input        Branch;                // while Branch is 1,it means current instruction is beq
    input        nBranch;               // while nBranch is 1,it means current instruction is bneq
    input        Jmp;                   // while Jmp 1, it means current instruction is jump
    input        Jal;                   // while Jal is 1, it means current instruction is jal
    input        Jr;                    // while Jr is 1, it means current instruction is jr
```

Update the Value of the PC register

NOTES: The code here is JUST reference, NOT request.

```
reg[31:0] PC, Next_PC;
```

```
always @* begin
```

```
    if(((Branch == 1) && (Zero == 1)) || ((nBranch == 1) && (Zero == 0))) // beq, bne
```

```
        Next_PC = ... // the calculated new value for PC
```

```
    else if(Jr == 1)
```

```
        Next_PC = ... // the value of $31 register
```

```
    else Next_PC = ... // PC+4
```

```
end
```

```
always @(... clock) begin
```

```
    if(reset == 1)
```

```
        PC <= 32'h0000_0000;
```

```
    else begin
```

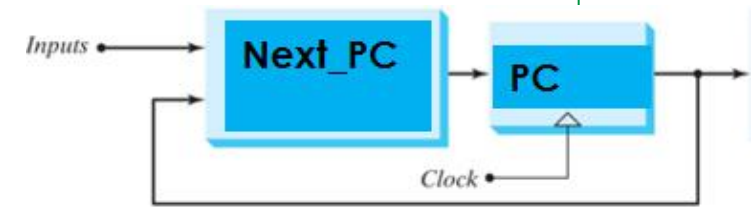
```
        if((Jmp == 1) || (Jal == 1)) begin
```

```
            PC <= ...;
```

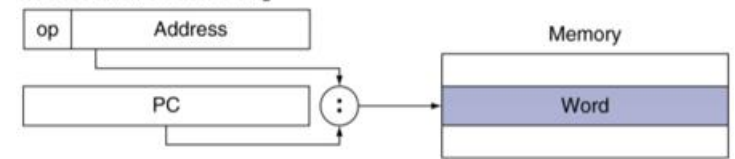
```
        end
```

```
    else PC <= ...;
```

```
end
```



5. Pseudodirect addressing



Q1: Complete the code to update 'Next_PC'

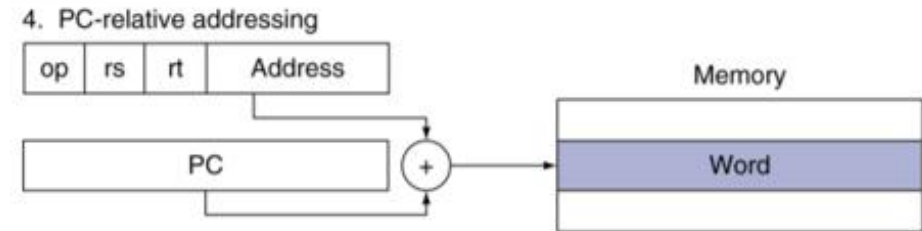
Q2: Could be 'PC' ready while read the 'prgrom'? Determine when to update the value of the PC register.

Q3: Is this Minisys ISA a Harvard structure or Von Neumann structure take a look at the initial value of PC

Outputs of IFetch: Prepare for Decoder and ALU

```
output[31:0] branch_base_addr;    // (pc+4) to ALU which is used by branch type instruction  
output[31:0] link_addr;           // (pc+4) to decoder which is used by jal instruction
```

Here for “pc+4”, the value of pc is the address of current processing instruction .



Don't forget to instance instruction memory, complete the port binding.

TIPS: The design here is for reference ONLY, NOT request.

Practice2

1. Make a Minisys source file with j, jal, jr, beq,bne and other NON-jumping instructions included.
2. Using the **Minisys1AssemblerV2.2** to assembler the source file on step 1, get the coe files .
3. Using the “prgmip32.coe” generated on step 2 as the initial file for the ROM in IFetch submodule to verify the its funciton:
 - 3-1) What’s the value of register PC while the reset is valid.
 - 3-2) While reset is invalid, on which edge of clock would the value of register PC be updated?
 - 3-3) What’s the updated value to register PC while the current instruction is j, jal, jr, beq,bne and other NON-jumping instructions.
 - 3-4) On which edge of clock would the instruction be fetched out?
 - 3-5) Is there any difference between the two output ports(“**branch_base_addr**” and “**link_addr**”)

Tips:1) There are j, jal, jr, beq,bne and other NON-jumping instructions in cputest.asm(which is in the Minisys1AssemblerV2.2.rar), you can modify it as an alternative to the 1st step.

2) “**Minisys1AssemblerV2.2.rar**” could be found in the directory “**lab_tools**” of course blackboard site

https://bb.sustech.edu.cn/webapps/blackboard/content/listContentEditable.jsp?content_id= 281670_1&course_id= 3602_1