Computer Organization

Lab9 CPU Design1: Decoder and DataMemory

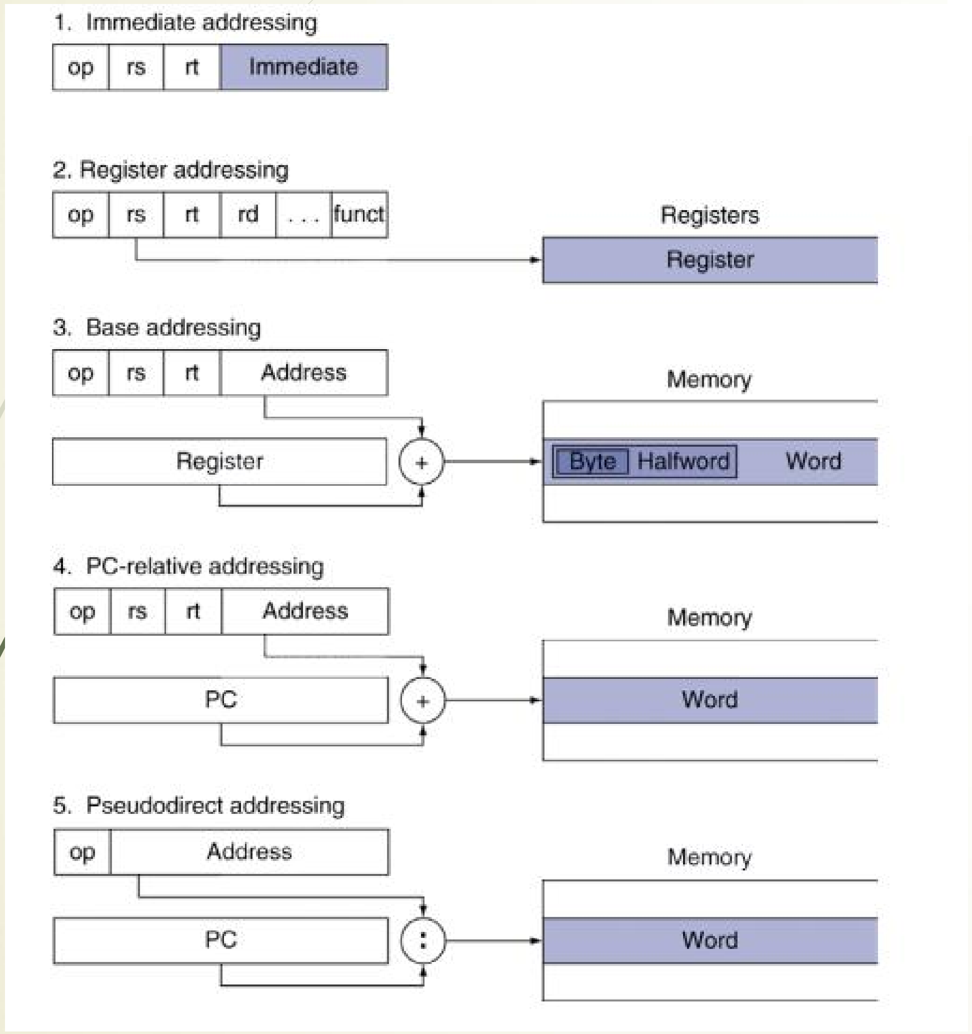
2021 Spring term

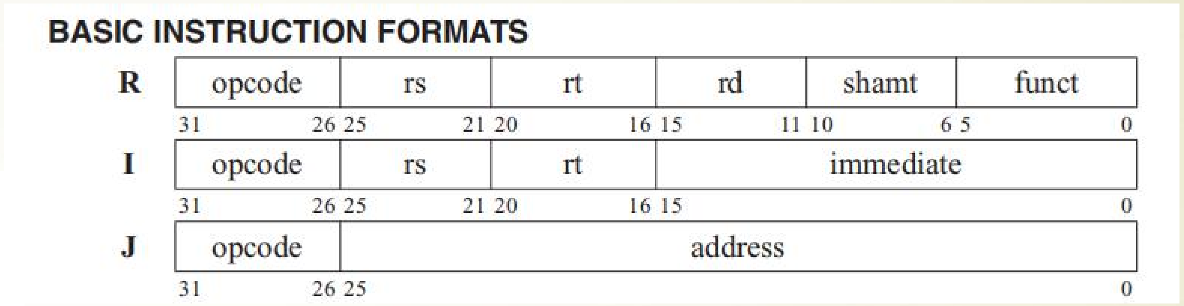
wangw6@sustech.edu.cn

Topics

- CPU Design(1)
  - ISA
  - Data Path(1)
    - Decoder
    - Data Memory
      - Customize block memory IP core
      - Instance IP core to build Data Memory

ISA-MIPS





BASIC INSTRUCTION FORMATS

R	opcode (31–26)	rs (25–21)	rt (20–16)	rd (15–11)	shamt (10–6)	funct (5–0)
I	opcode (31–26)	rs (25–21)	rt (20–16)	immediate (15–0)		
J	opcode (31–26)	address (25–0)				

MIPS addressing

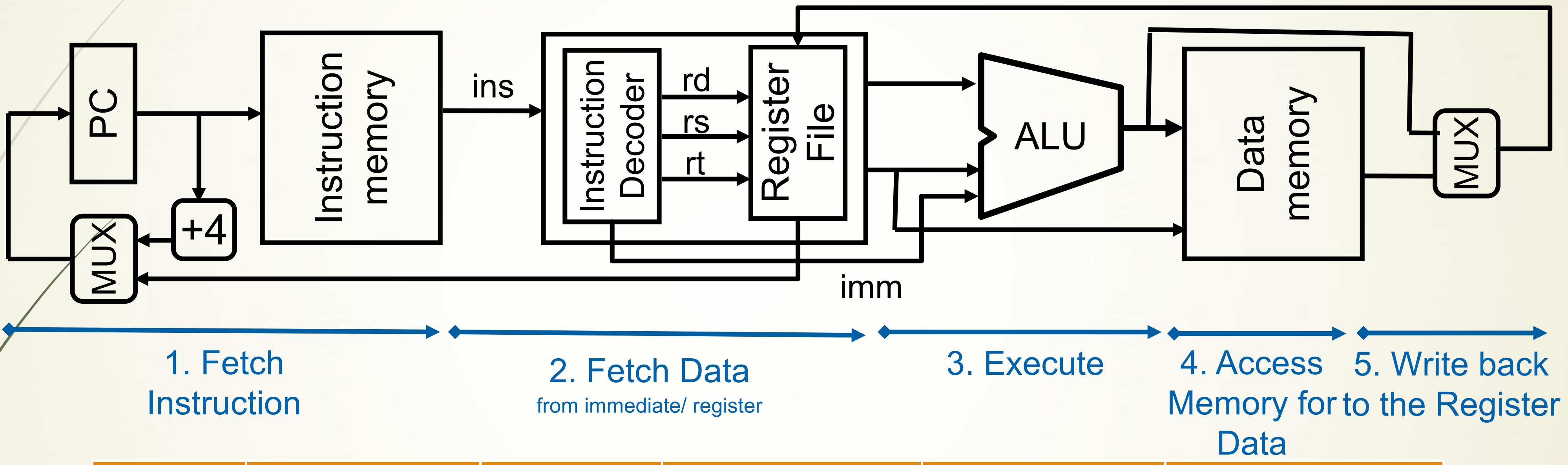
- 1. immediate addressing:
  - addi $s0,$s1,5 【I】
- 2. register addressing:
  - add $s0,$s1,$s2 【R】
- 3. base addressing:
  - lw $s0,0（$s1） 【I】
- 4. pc-relative addressing
  - bne $s0,$s1,EXIT 【I】
- 5. psudo-directive addressing
  - j EXIT 【J】

How CPU works

- **Fetch** the instruction from the instruction memory according to the value of the PC register
- **Analyze** the instruction to get how to process the data
- **Execute** the instruction
  - R type :
    - [add] : **find** the register and **read** its value, do the **arithmetic calculation** , **write** the **register**
  - I type :
    - [addi]: same as add except one of operands is from the immediate number of the instruction
    - [branch]: **find** the register and **read** its value, do the **comparation** and **write** the PC **register** with the target address(pc relative addressing)
    - [load/store]: **find** source data from the register/memory, **read** its value to **wirte** the targert place
  - J type :
    - [j] : **write** the PC register with the target address(presdo-directory addressing)
    - [jr] : **write** the PC register with the value of $31
    - [jal] : **write** $31 and the PC register

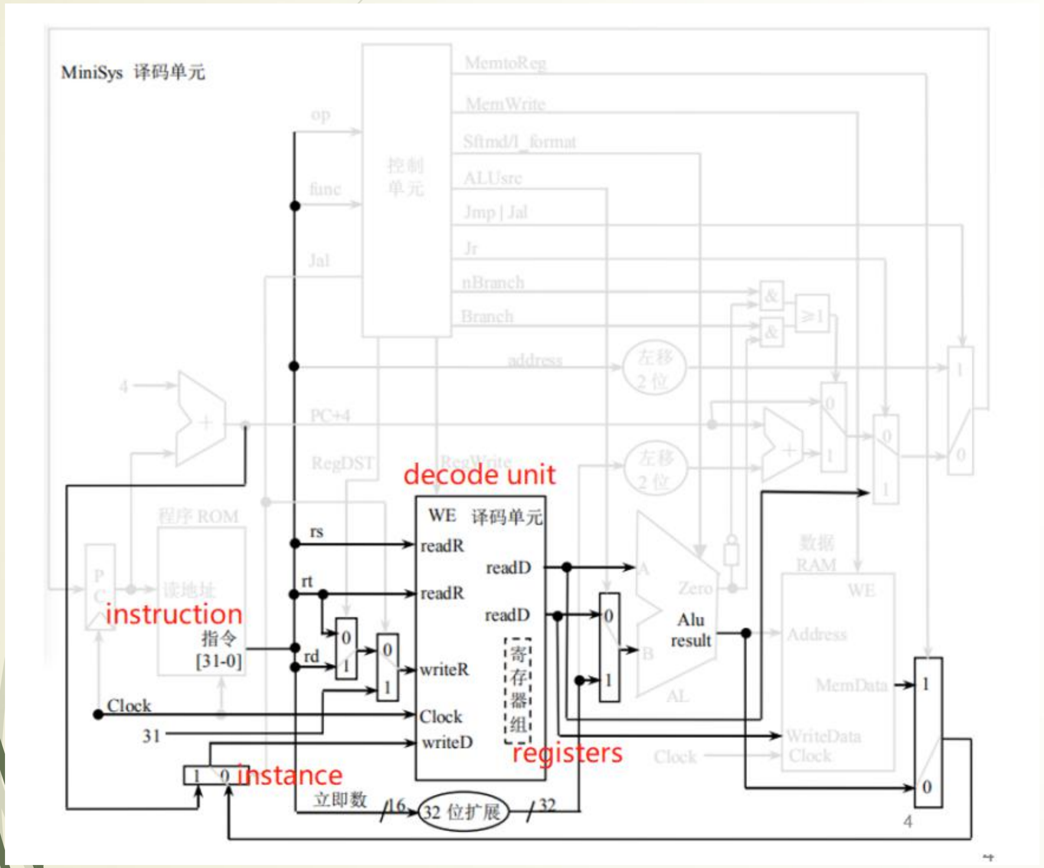
Core Modules in CPU

reference from Minisys, a little modification



	Instruction fetch	Data Fetch	Instruction Execute	Memory Access	Register WriteBack
add[R]	Y	Y	Y		Y
store[I]	Y	Y	Y	Y	
load[I]	Y	Y	Y	Y	Y
branch[I]	Y	Y	Y		
jump[J]	Y	Y	Y		
jal [J]	Y	Y	Y		

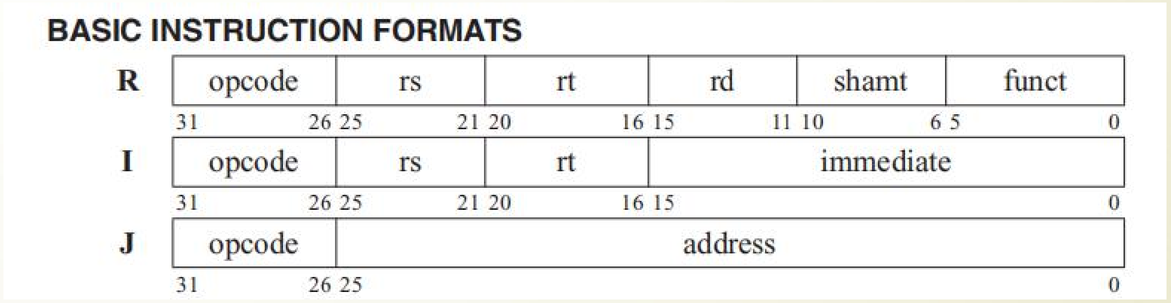
Decoder



- **Get data** from the instruction directly or indirectly

  opcode, function code : how to get data, where to get data

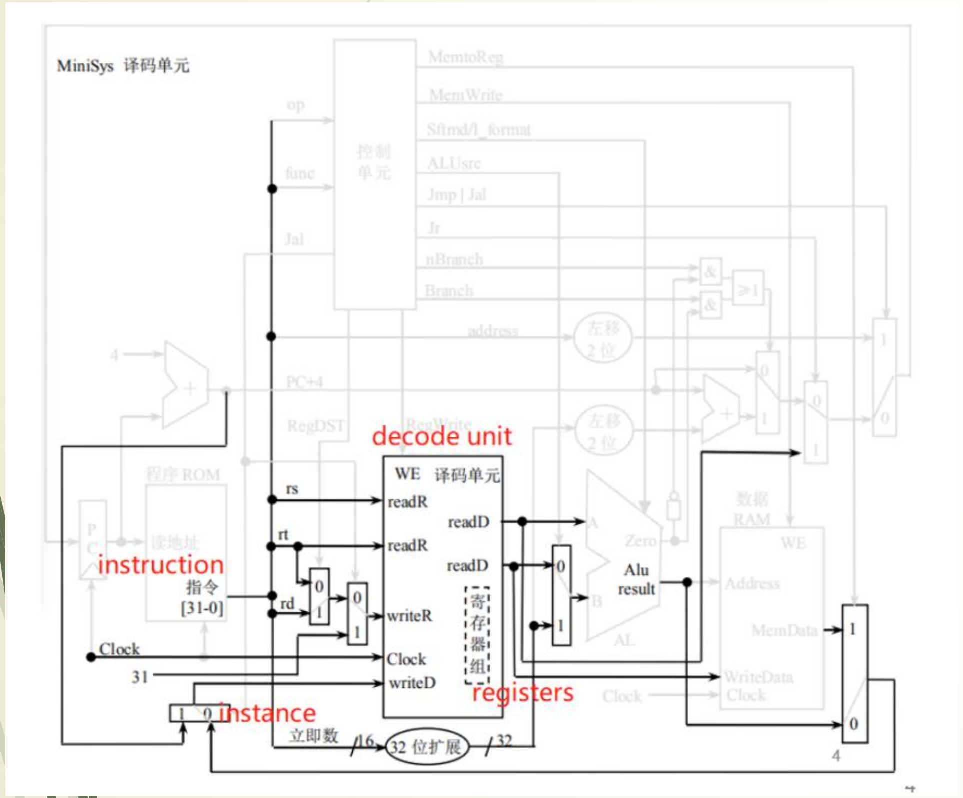
  - **immediate data** in the instruction([15:0]), (e.g. immi = Instruction[15:0]) need to be **signextended to 32bits**
  - **data in the register**, the address of the register is coded in the instruction. e.g. rs = Instruction[25:21];
  - **data in the memory**, the address of the memory unit need to be calculated by ALU with base address stored in the register and offset as immediate data in the instruction
- **Read/Write data from/to Register File**



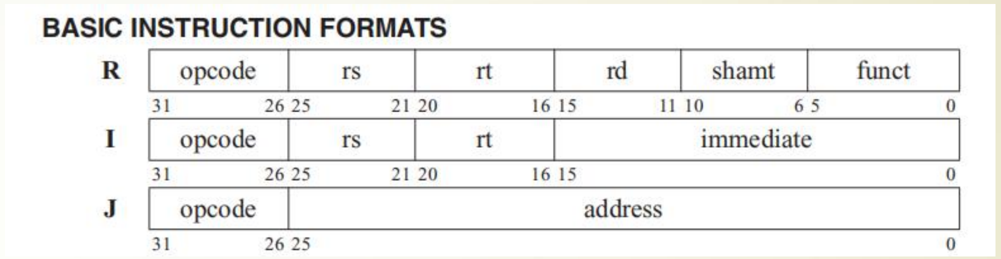
BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

Decoder continued



- Register File
- Inputs
  - read address
    - [R] add: rs,rt
    - [J] jr : [31]
  - write address
    - [R] add: rd
    - [J] jal : [31]
    - [I] addi: rt
  - write data
    - [R] add: alu_result
    - [J] jal : pc+4
    - [I] load: data from memory
  - control signal
    - [R] /[I]/[J] writeRegister
    - [J]: jal
    - [I] : memToReg
    - [R]/[I]: rd vs rt

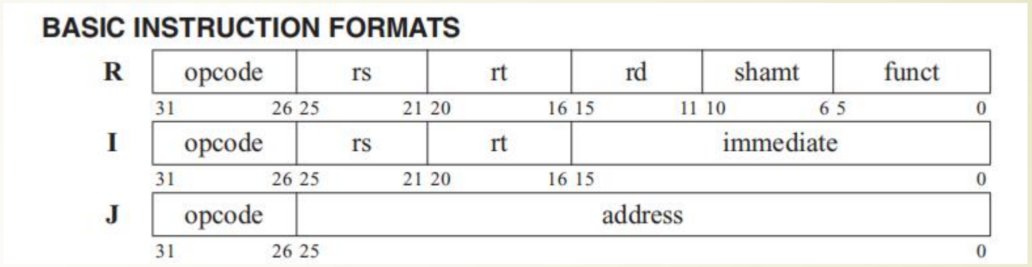


BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15		0
J	opcode	address				
	31 26	25				0

- Register File
- Outputs
  - read data1
  - read data2
  - extendedImmi

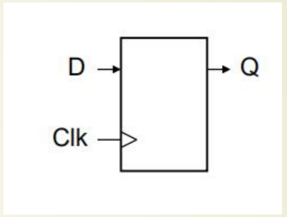
Decoder continued

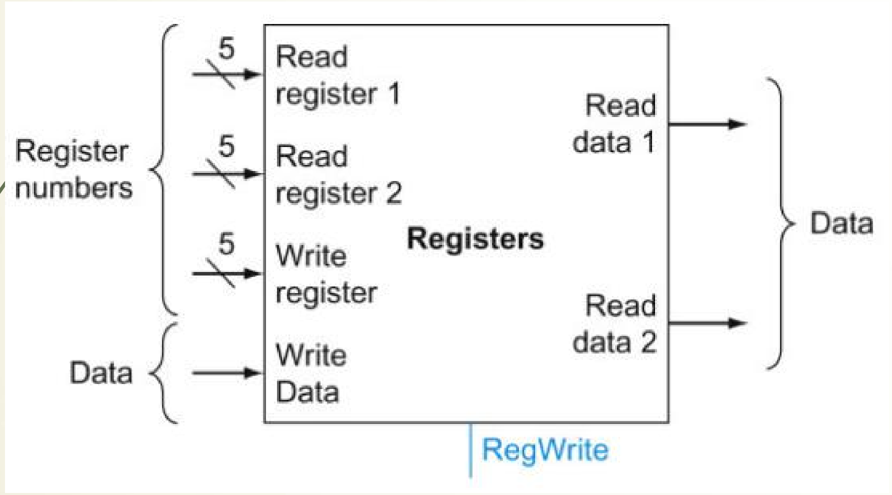


BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

Register File:
Almost all the instructions need to read or write register file in CPU
32 common registers with same bitwidth: 32

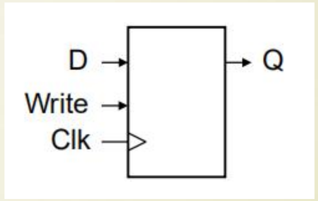




Q1:
How to avoid the confliction between register read and register write?

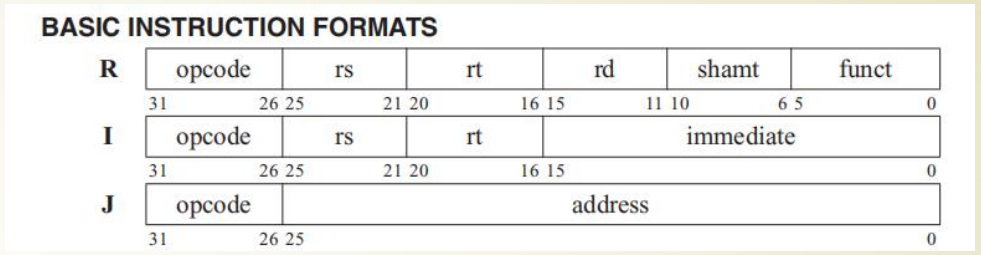
Q2: Which kind of circuit is this register file, combinatorial circuit or sequential circuit?

Q3: How to determine the size of address bus on register file?



```
//verilog tips:
reg[31:0] register[0:31];
assign Read data 1 = register[Read register 1]
register[Write register] <= WriteData;
```

Practice1

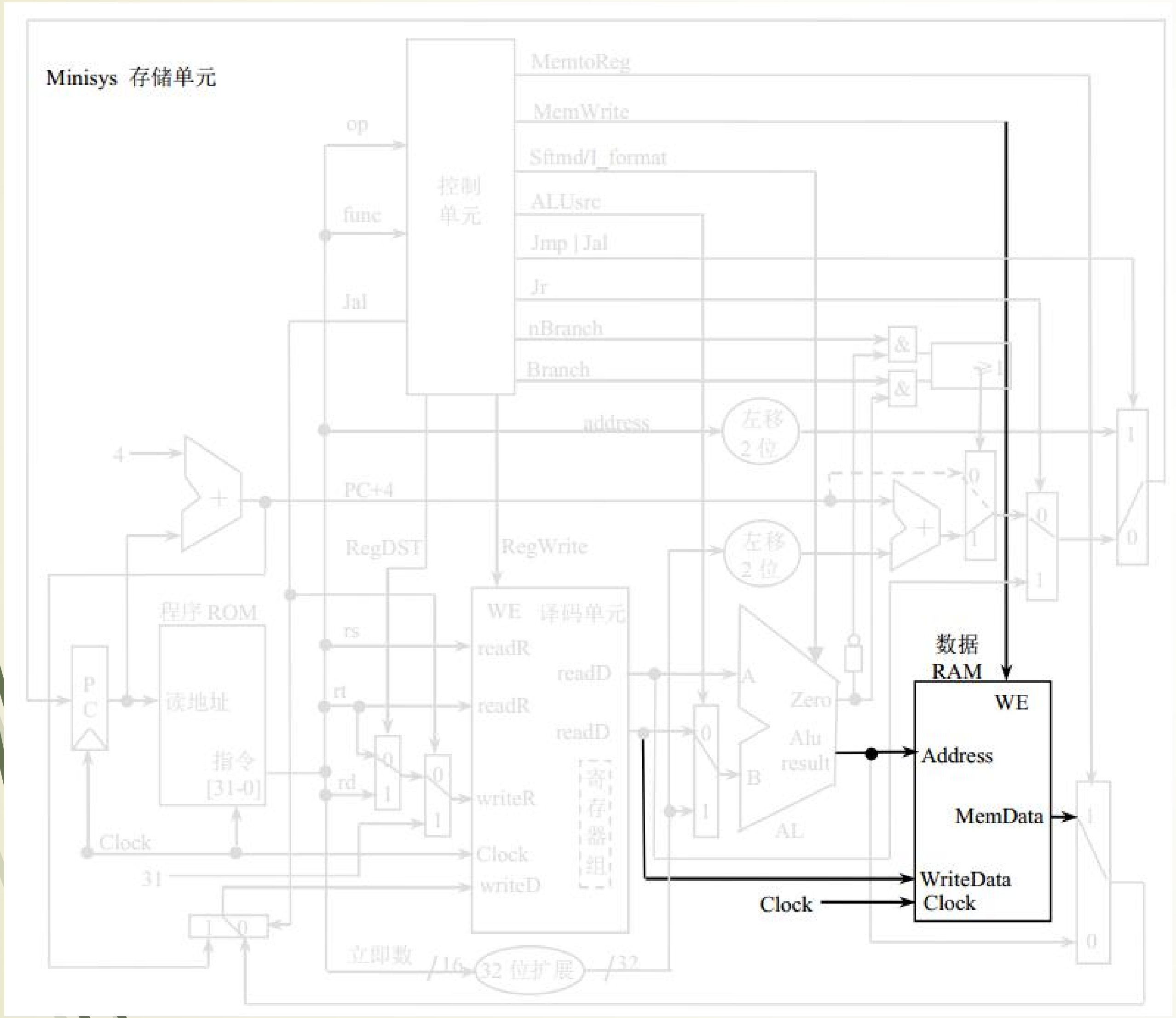


- 1. Create the decoder as specified on page 6~7
  - There are **32** registers in the decoder, each register is **32bits** width
  - All the registers are **readable** and **writeable** except $0, **$0 is readonly**.
  - The **read** should be done at any time while **write** only happens on the **posedge** of the clock.
  - The register file should support **R/I/J** type instructions.
    - such as: **add, addi, lw, sw, jal, jr**
- 2. Verify its function by simulation.
- 3. List the signals which are used in the decoder

NOTE: Signals' name are determined by designer

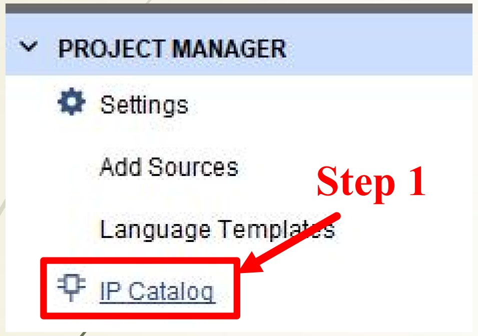
name	from	to	bits	function
regWrite	instruction	decoder	1	1 means write the register identified by writeAdress
immediate	instruction	ALU	32	the signextended immediate
readRegister1	instruction	decoder	5	the address of read register instruction[25:21]
…				

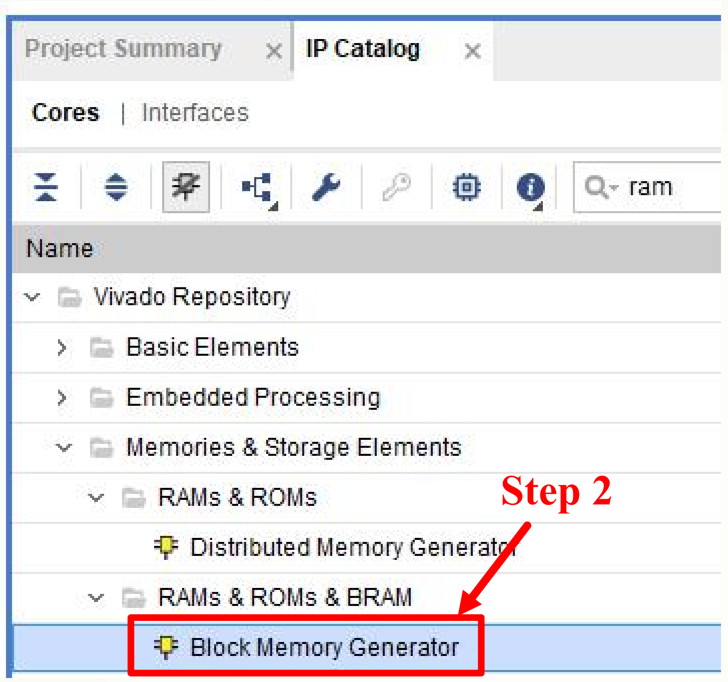
Memory



```
module dmemory32(readData,address,
writedata,memWrite,clock);

// Clock signal
input clock;

// the address of memory unit  which is tobe read/write
input[31:0] address;

// used to determine to write or read
input memWrite;

// data tobe wirte into the memory unit
input[31:0] writeData;

// data tobe read from memory  unit
output[31:0] readData;
```

Using IP core Block Memory



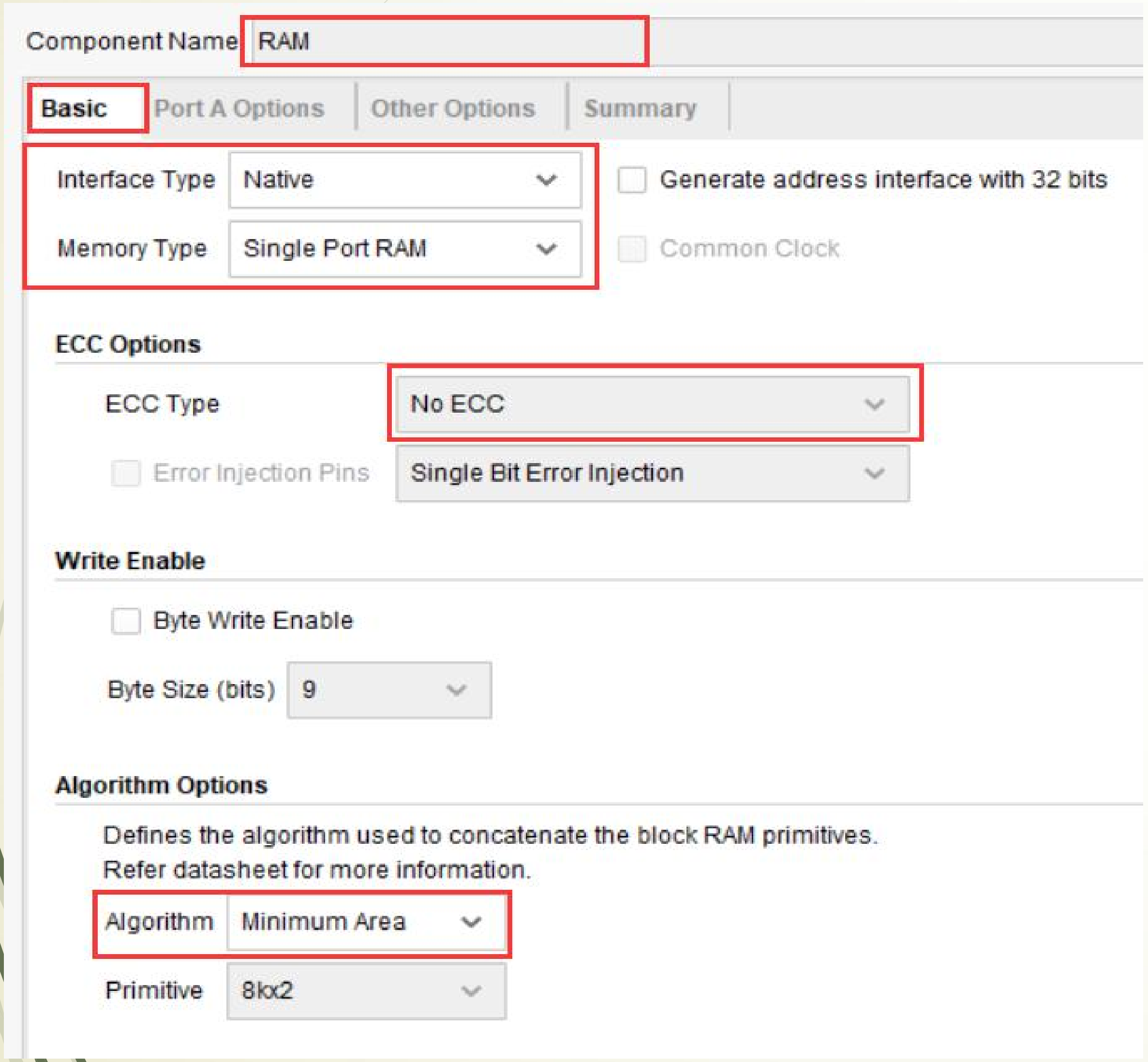


Using the **IP core Block Memory** from Xilinx to implement the data memory.

Import the **IP core** in vivado project
Step1:
in "PROJECT MANAGER" window
click "IP Catalog"

Step2:
in "IP Catalog" window
> Vivado Repository
> Memories & Storage Elements
> RAMs & ROMs & BRAM
> **Block Memory Generator**

Customize Memory IP core



Component Name RAM

Basic | Port A Options | Other Options | Summary

Interface Type: Native

Memory Type: Single Port RAM

☐ Generate address interface with 32 bits

☐ 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

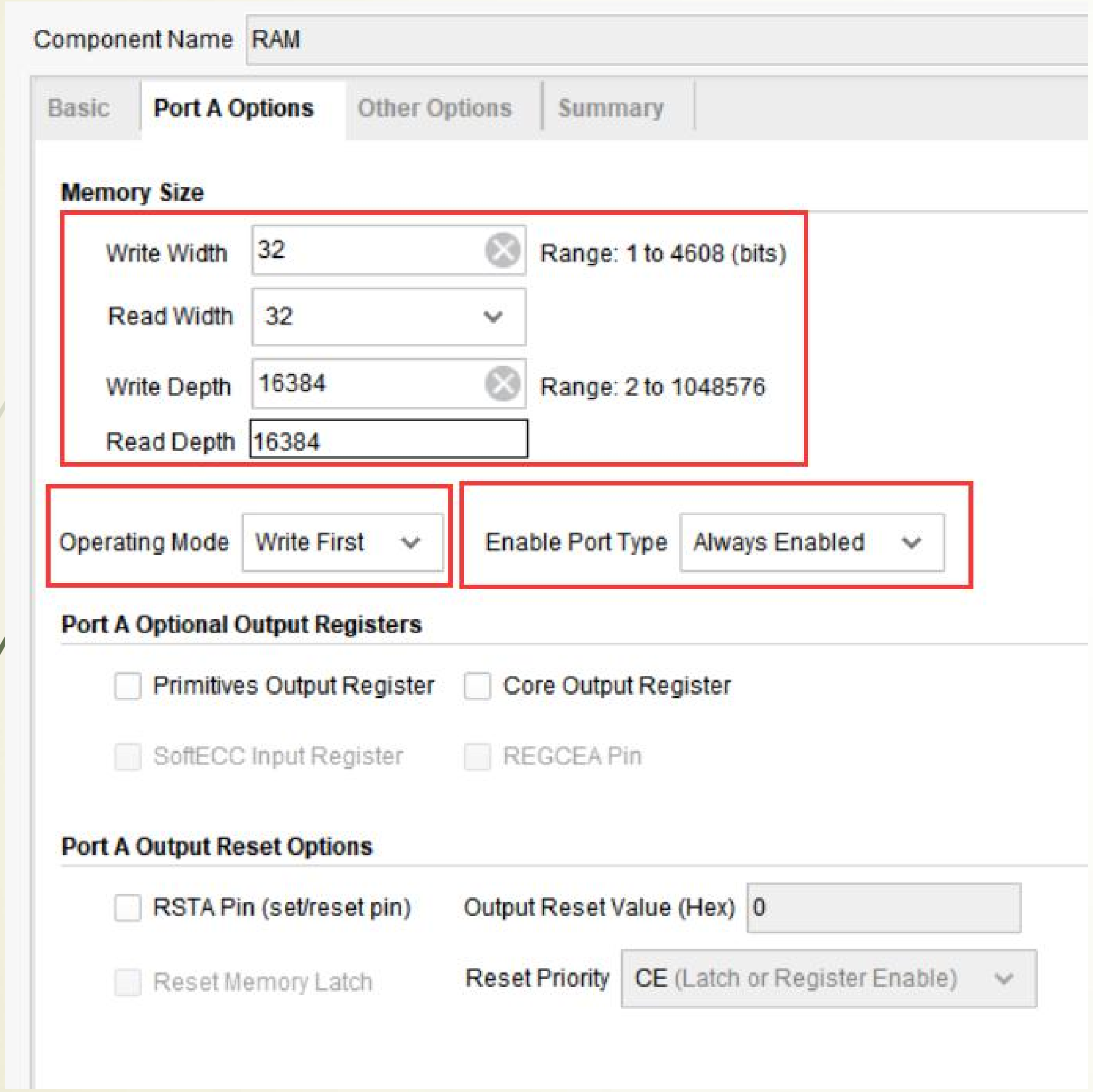
Primitive: 8kx2

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

customize memory IP core:

- Component Name: **RAM**
- **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

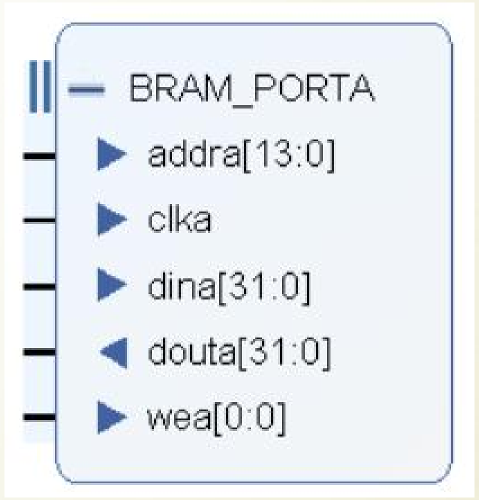
- [ ] RSTA Pin (set/reset pin)

Output Reset Value (Hex): 0

- [ ] Reset Memory Latch

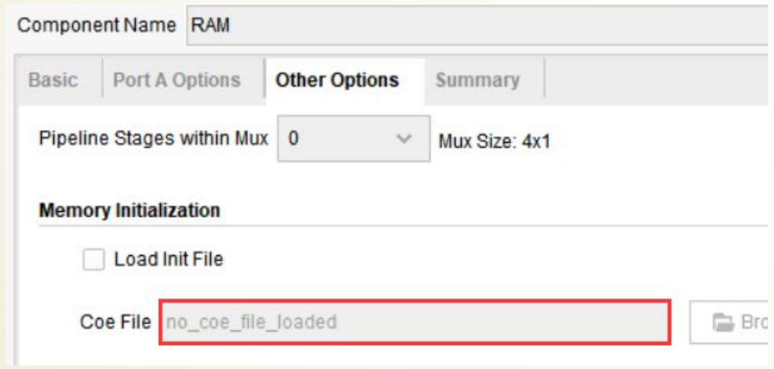
Reset Priority: CE (Latch or Register Enable)

- **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

- **Other Options** settings:
  - **1.** When specifying the initialization file for RAM, the IP core RAM just customized **WITHOUT initial file** and **corresponding path**, so set it to **no initial file** when creating RAM.
  - **2. After creating** the RAM
    - **2-1. copy** the initialization file dmem32.coe **to** projectName.srcs/sources_1/ip/RAM. (“projectName.srcs” is under the project folder)
    - **2-2.** Double-click the newly created RAM IP core, **reset** it with **an initialization file**, select the dmem32.coe file that has been in the directory of projectName.srcs/sources_1/ip/RAM.



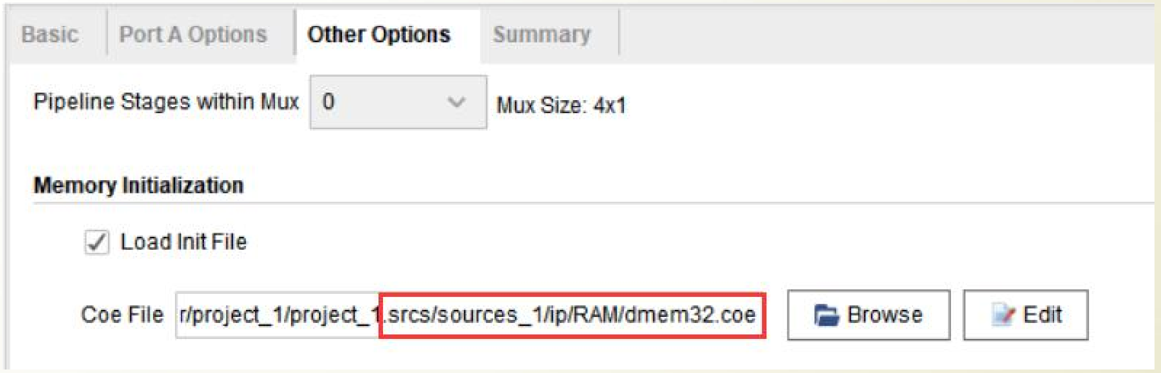
Component Name RAM

Pipeline Stages within Mux 0 Mux Size: 4x1

Memory Initialization

☐ Load Init File

Coe File no_coe_file_loaded



Pipeline Stages within Mux 0 Mux Size: 4x1

Memory Initialization

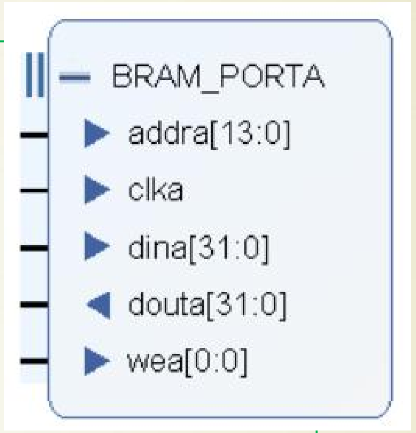
☑ Load Init File

Coe File r/project_1/project_1.srcs/sources_1/ip/RAM/dmem32.coe

Tips: “**dmem32.coe**” file could be found in the directory “**source/lab/lab9**” of course sakai site
https://sakai.sustech.edu.cn/portal/site/d50211ea-1586-4344-9d92-a6c42eb7f4e0/tool/b47e4c13-4220-4f61-b881-a211abee2a96?panel=Main

Design Module With Memory IP Instanced

```
// Part of dmemory32 module

//Generating a clk signal, which is the inverted clock of the clock signal
assign clk = !clock;

//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
 );
```



Notes: *Because the inherent delay of the chip is used, the address line of the RAM is not ready at the rising edge of the clock, which causes the data to be read incorrectly on the rising edge of the clock. Consider the method of using the inverted clock to make the read data more ready than the address. It takes about half a clock late to get the correct address.*

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 begin
   #200
       writeData = 32'ha0000_00f5;
       memWrite = 1'b1;
   #200
      memWrite = 1'b0;
end
endmodule
```

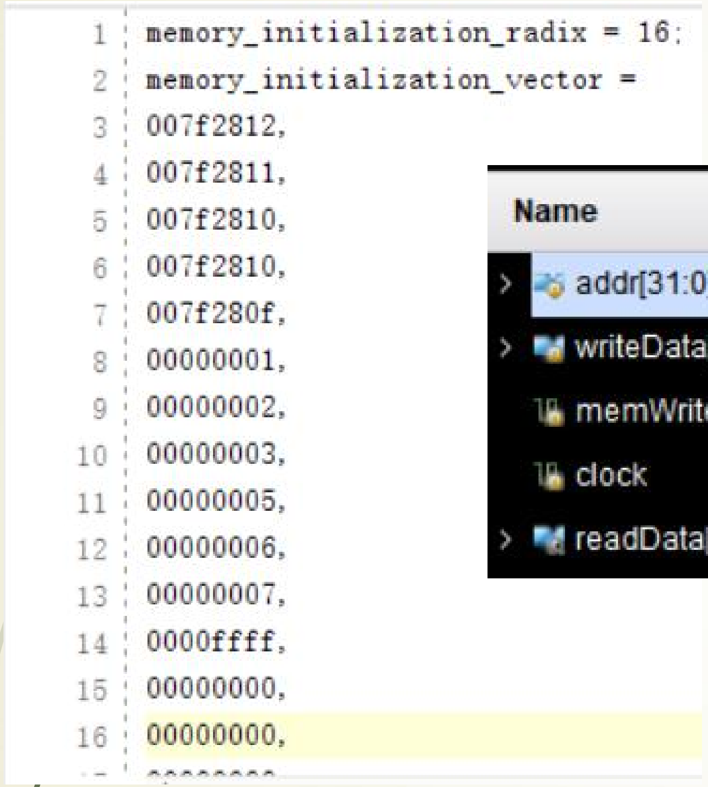
The “addr” is 0x0000_0010

1) Set “memWrite” to low level and read the data of the address in RAM.

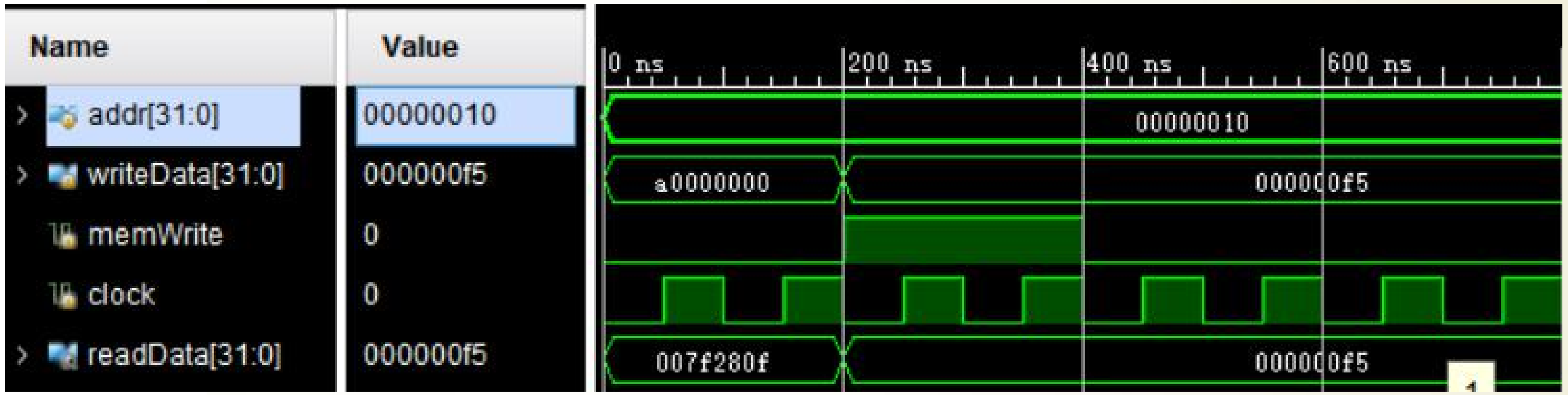
2) Set “memWrite” to high level and writeData to 0xa000_00f5

3) Set “memWrite” to low level and read the data of the address in RAM.

Function Verification continued



```
memory_initialization_radix = 16;
memory_initialization_vector =
007f2812,
007f2811,
007f2810,
007f2810,
007f280f,
00000001,
00000002,
00000003,
00000005,
00000006,
00000007,
0000ffff,
00000000,
00000000,
```



Name	Value
addr[31:0]	00000010
writeData[31:0]	000000f5
memWrite	0
clock	0
readData[31:0]	000000f5

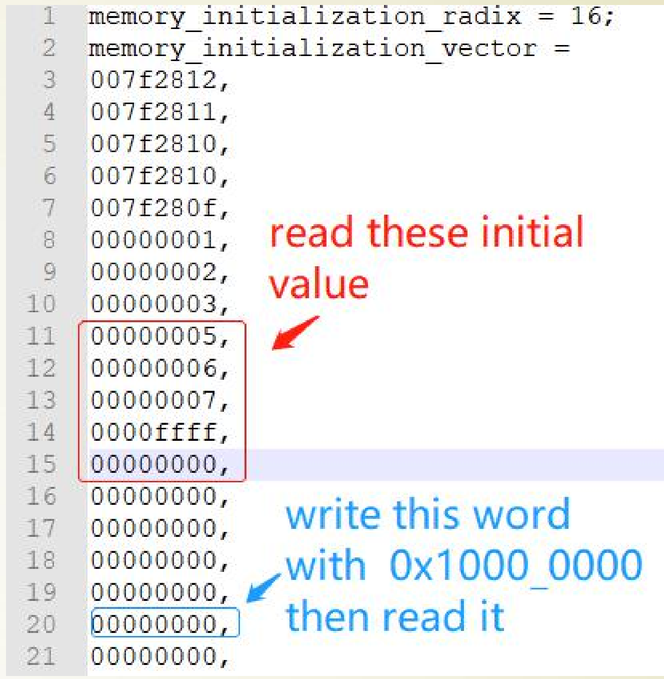
Q1: On which edge of clock does the read and write operations occur? posedge or negedge?
Q2: What's value will be get while read the memory according to the “addr” 0x0000_0016?

Tips: “**dmem32.coe**” file could be found in the directory “**source/lab/lab9**” of course sakai site
https://sakai.sustech.edu.cn/portal/site/d50211ea-1586-4344-9d92-a6c42eb7f4e0/tool/b47e4c13-4220-4f61-b881-a211abee2a96?panel=Main

Practice2

- 1. Build the data memory module following the steps list on page 11~15

  NOTE: Using the same "dmem32.coe" as page 17 to initialize the memory ip core.
- 2. Verify its function by simulation
  - Change the testbench by adding more testing
    - **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 screeshot on the left hand, then read it out.
- 3. List all the signals which are needed for data memory module

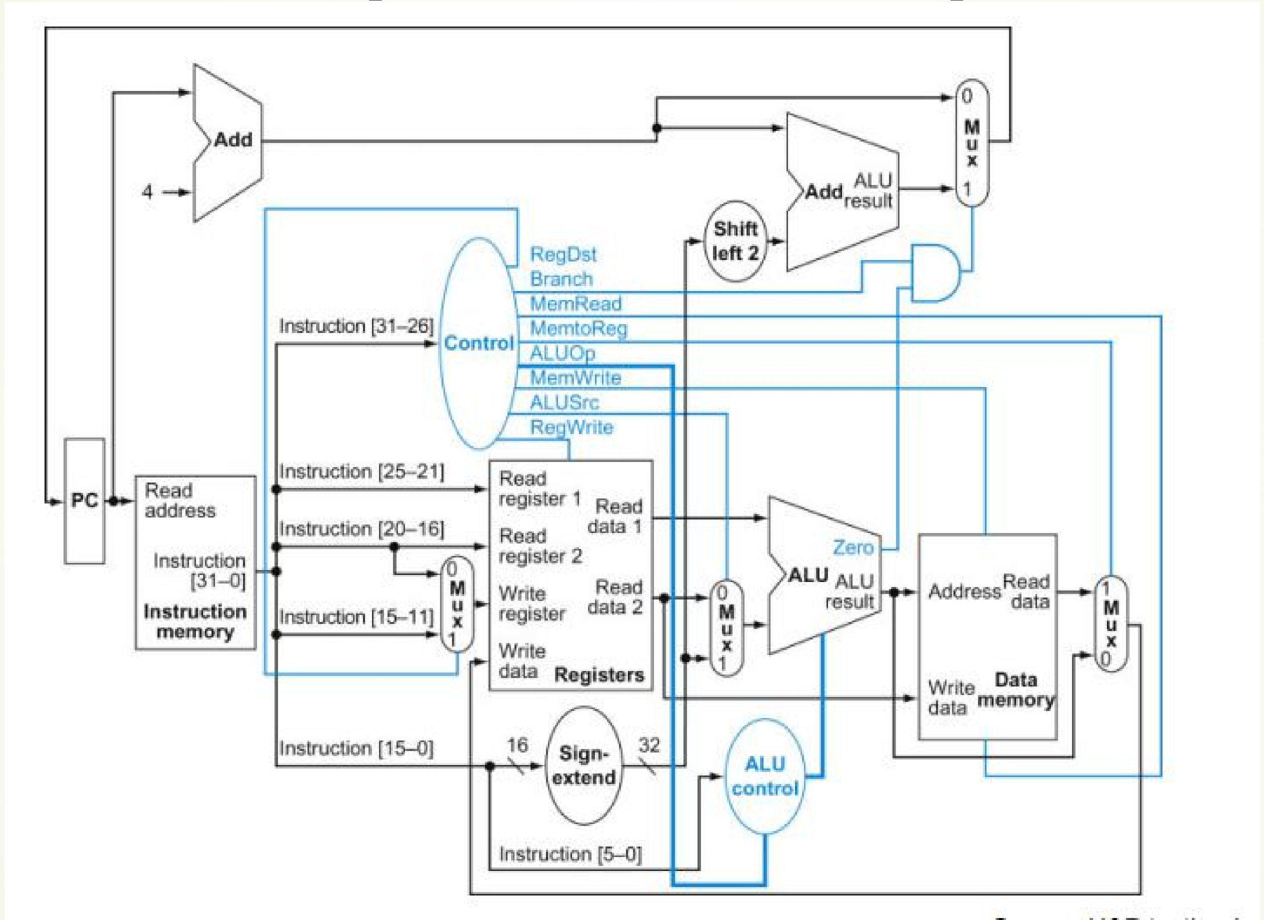


```
memory_initialization_radix = 16;
memory_initialization_vector =
007f2812,
007f2811,
007f2810,
007f2810,
007f280f,
00000001,
00000002,
00000003,
00000005,
00000006,
00000007,
0000ffff,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
```

name	from	to	bits	function
clock		Data Memory	1	memory write is sensitive with its negedge
rdata	Data Memory	Decoder	32	the word read from the data memory and send to decoder
…				

Tips: "**dmem32.coe**" file could be found in the directory "**source/lab/lab9**" of course sakai site
https://sakai.sustech.edu.cn/portal/site/d50211ea-1586-4344-9d92-a6c42eb7f4e0/tool/b47e4c13-4220-4f61-b881-a211abee2a96?panel=Main

TIPS: Control path & Data path of CPU



Source: H&P textbook

Control Path: Interprete instructions and generate signals to control the data path to execute instructions

Data Path: The parts in CPU with componets which are involved to execute instructions