Computer Organization

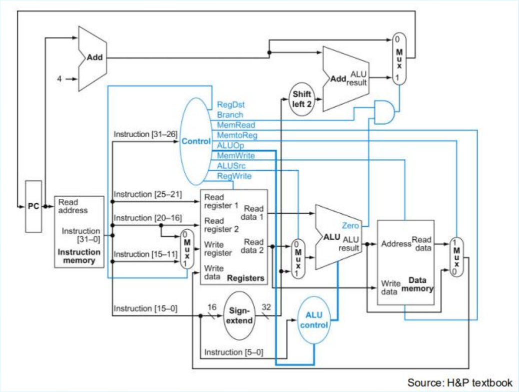
Lab11 CPU Design(3)

'single' cycle CPU
clock, I/O

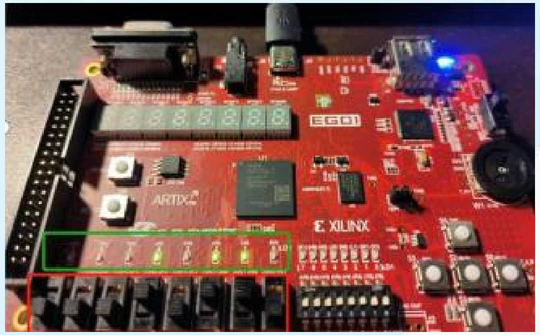
Topic

- **CPU Design(3)**
  - **A 'single' cycle CPU**
  - **Clock (IP core)**

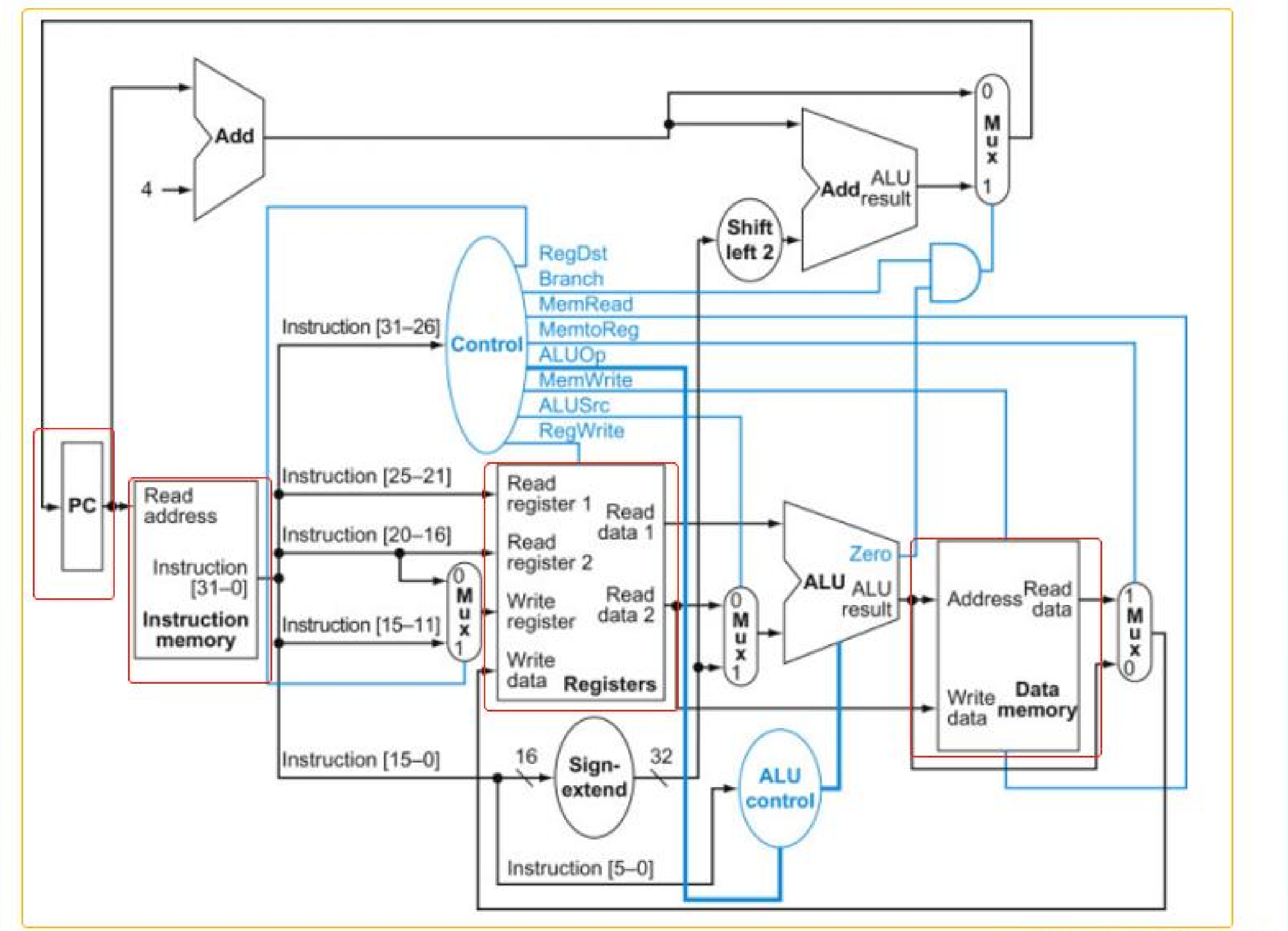
```
取指: last-cycle neg // update PC
      pos // fetch instruction
分析:组合逻辑
执行/计算：组合逻辑
访存:neg // get address
     neg // read out data
写回：next-cycle pos // write back data
```



- **CPU work with I/O device**
  - **Option1： MMIO(Memory-Mapped IO)**
    - **MemOrIO, Controller +**
  - **Option2： Specific Instruction**
  - **Collaborative work between CPU and I/O**



A 'single' cycle CPU



Source: H&P textbook

Q1. Does it **take time** for signals to be processed and transmitted within the module, as well as between modules?

Q2. Which sub modules within CPU **need the trigger from the clock**? When does the following event occur in a clock cycle?

1-1) **IFetch**: **update** the value of PC register
1-2) **IFetch**: **fetch** the instruction according to the value of PC

2-1-1) **Controller**: generate the **control signals**
2-2-1) **Decoder**: **get** the value of register(s)
2-2-2) **Decoder**: **generate** the extended **immediate**

3-1) **ALU**: **get** the **operands**
3-2) **ALU**: **generate** the **calculaton result**

4-1) **Dmemory**: **get** the **address**(from ALU) and **data**(from Decoder)
4-2) **Dmemory**: **read out the data**

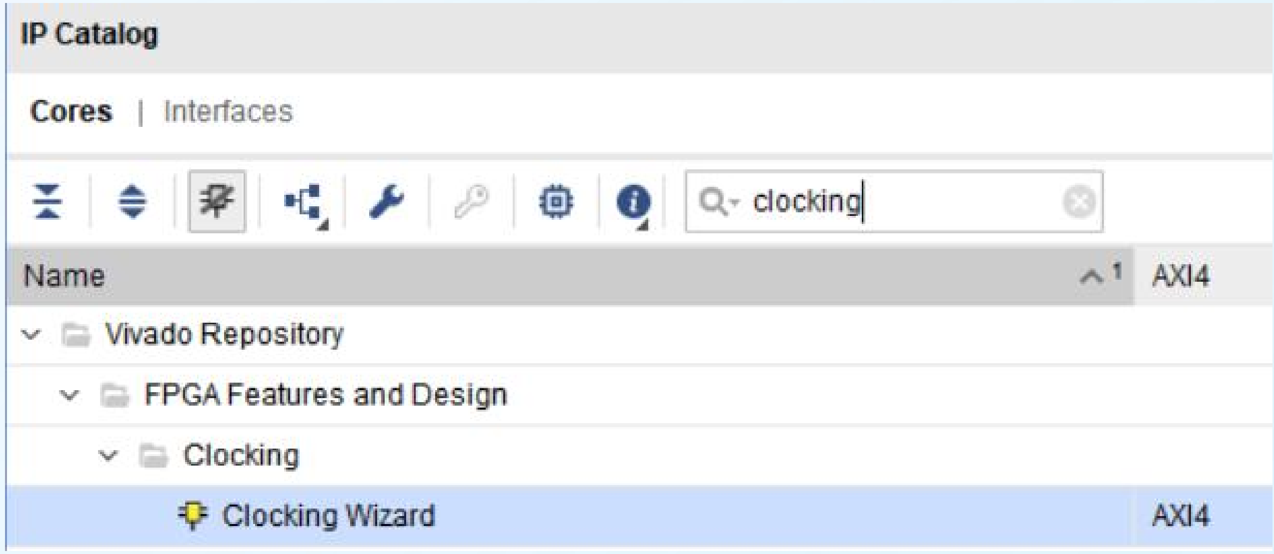
5-1) **Decoder**: **write back the data**

Clock

- Add **PPL clock IP core** to generate the needed clock:

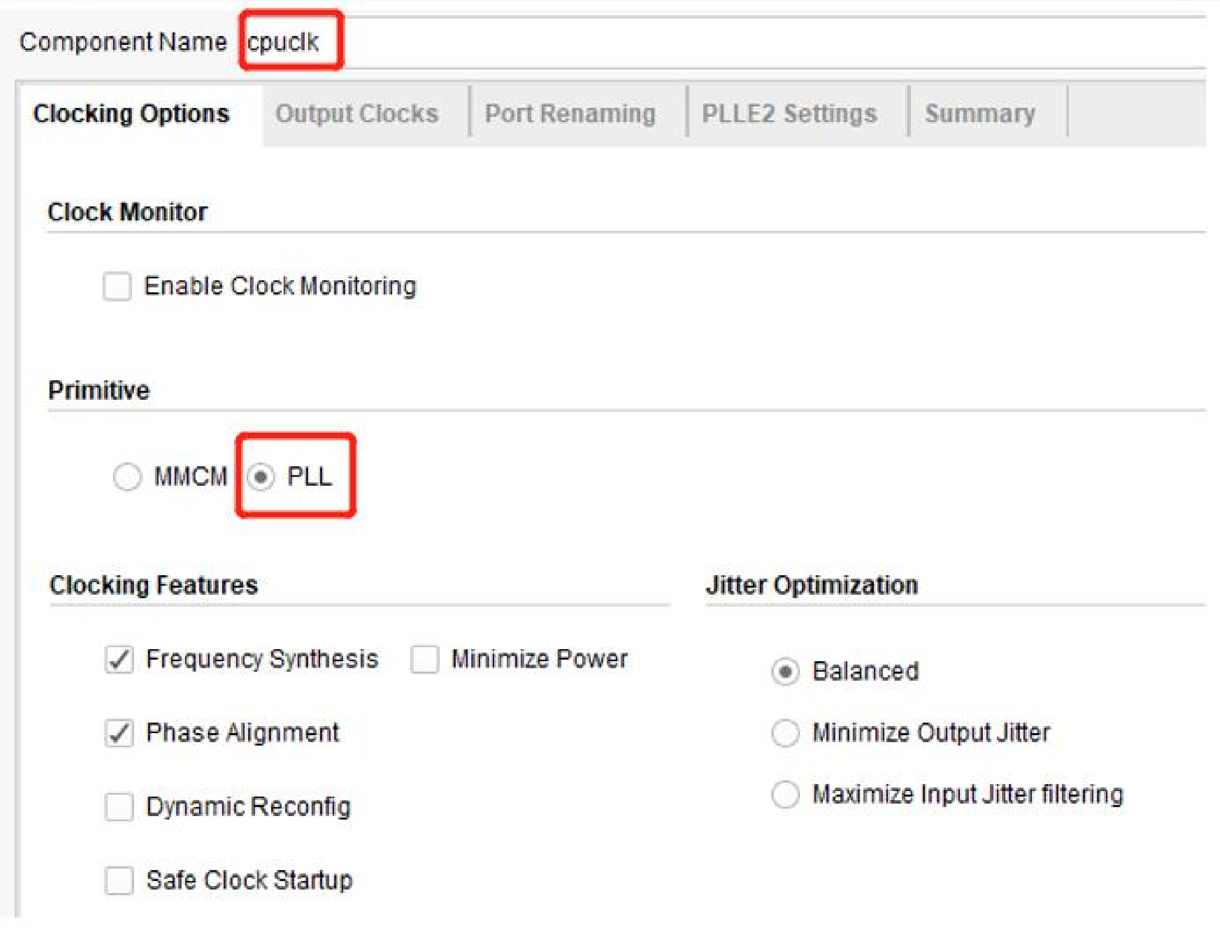
  1. The clock on the Minisys/EGO1 development board is **100Mhz** *(clk_in1)*
     - 100Mhz is too fast for a ‘single’ clock CPU

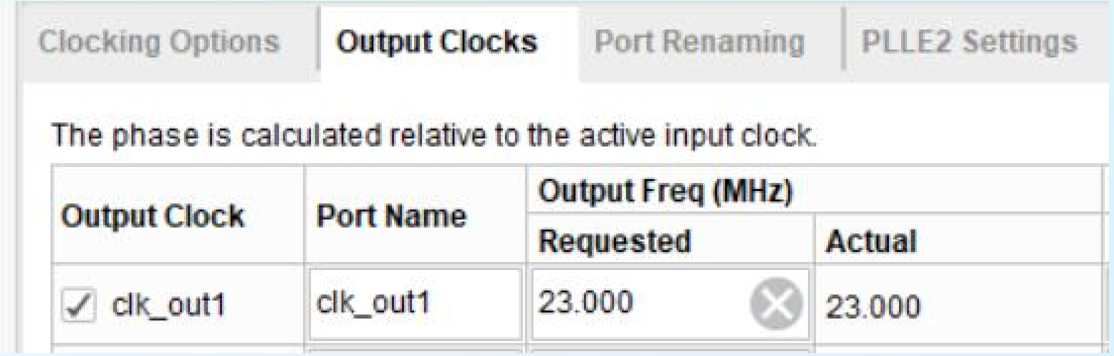
  2. **A clock of 23Mhz is more sutiable for the ‘single’ clock CPU** *(clk_out1)*

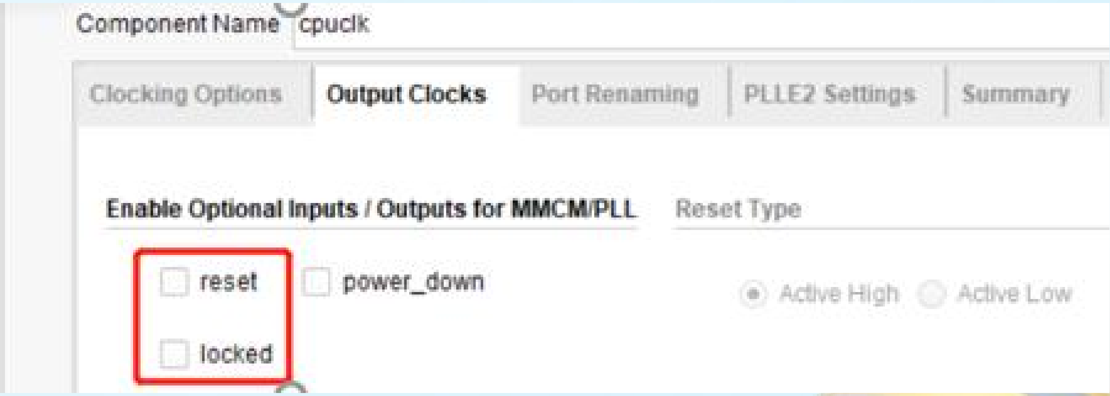


Clock continued

Custom the IP core, set its **name**, **Primitive**, **Output Freq** and **with out the reset and locked**.
Then **generate** the IP core with the settings.







The Function Verification of “cpuclk”

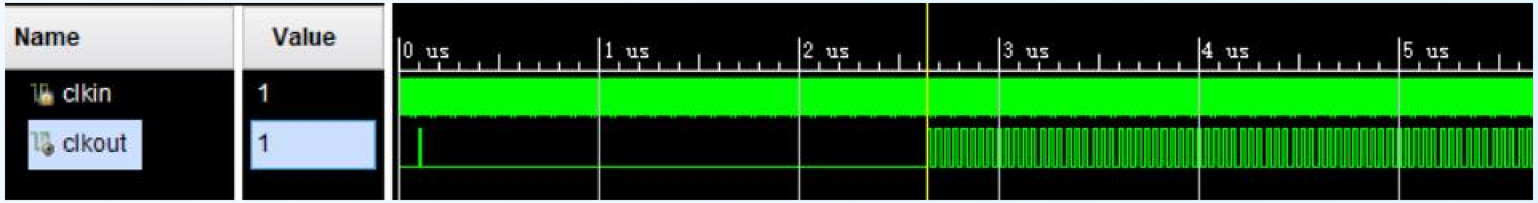
Functional Verification by **testbench** and **simulator**

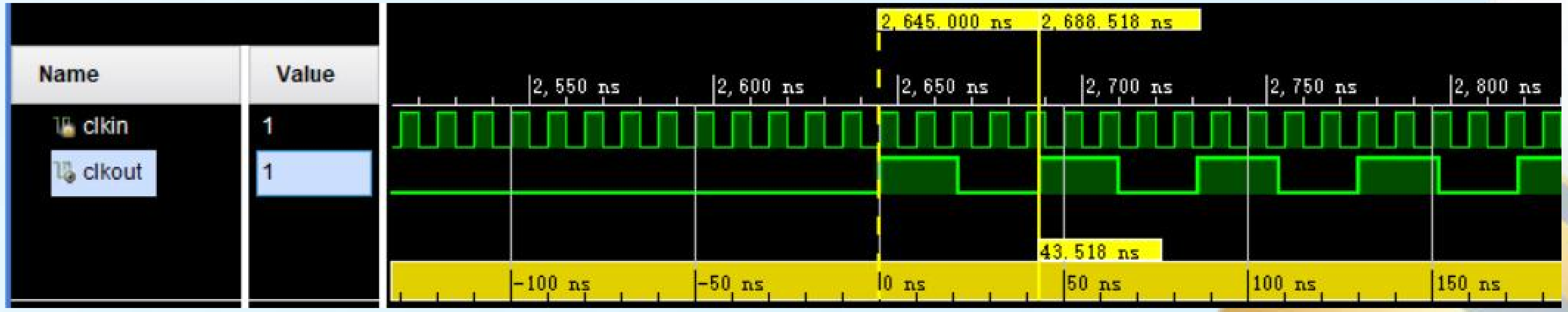
1) Create a verilog **testbench** module to instance the IP core “**cpuclk**” and bind its ports. set the frequence of the input on “cpuclk” as **100Mhz**.

2) Do the simulation to verify whether the output signal is a **23Mhz** clock signal while the input signal is **100Mhz**.

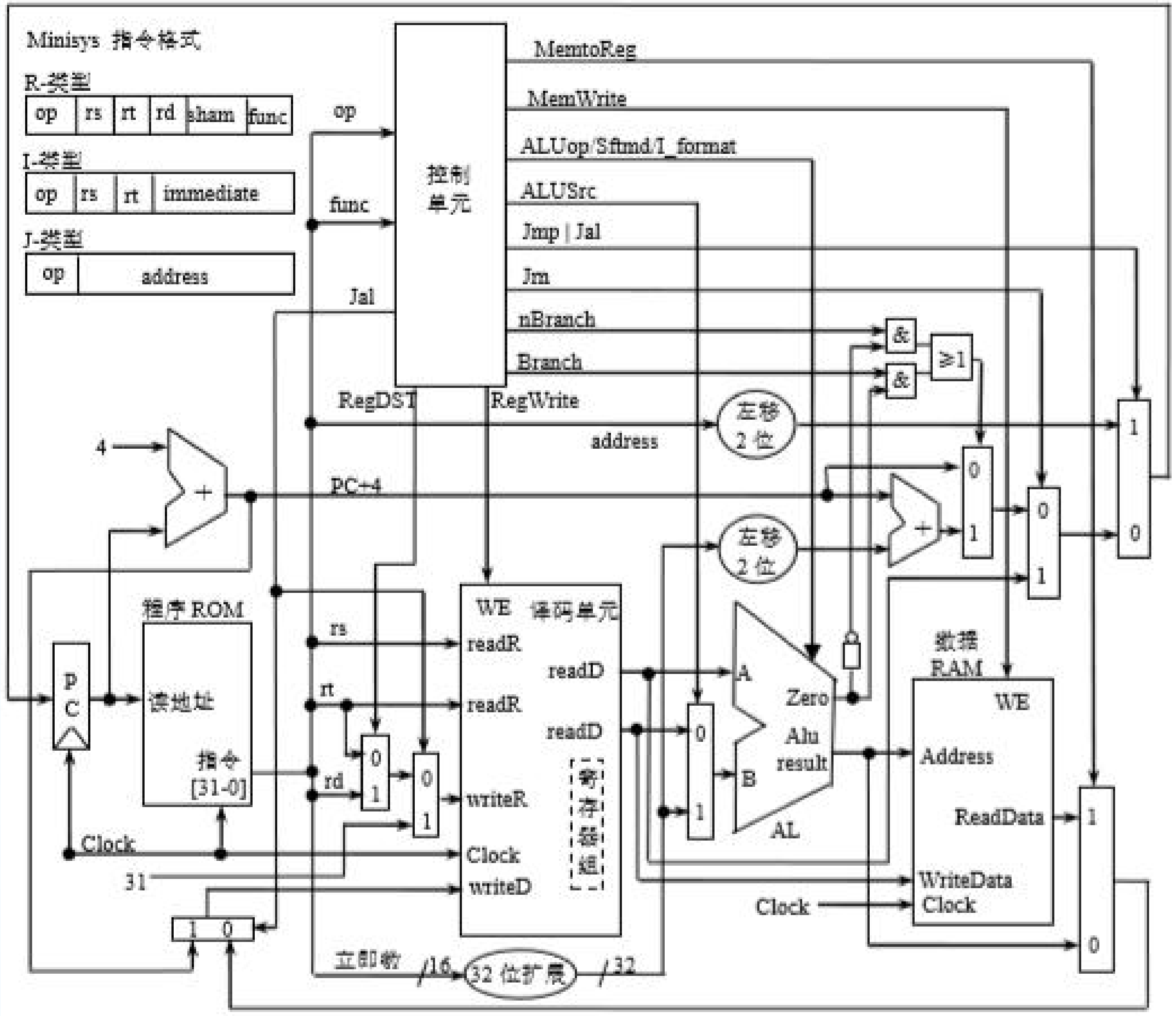
```
module cpuclk_tb( );  // a reference testbench for 'cpuclk'

    reg clkin;
    wire clkout;
    cpuclk clk1( .clk_in1(clkin), .clk_out1(clkout) );

    initial       clkin = 1'b0;
    always #5 clkin=~clkin;
endmodule
```

NOTE: The output of IP core 'cpuclk' need to work for a ‘long’ time to achieve stability.





Build and test the CPU



Build a **CPU top** module

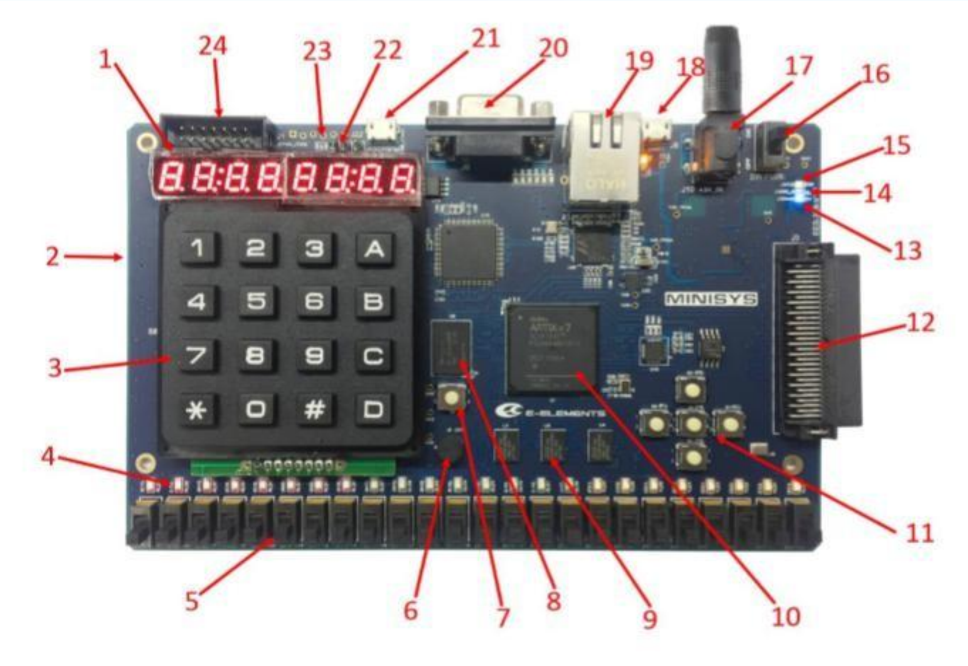
1) **Instantiating** the sub-modules: **clock**, **Decoder**, execution unit/**ALU**, **IFetch, Controller** and **Data-Memory**.

2) Complete the inter-module connection inside the CPU and the **binding** to the CPU **port.**

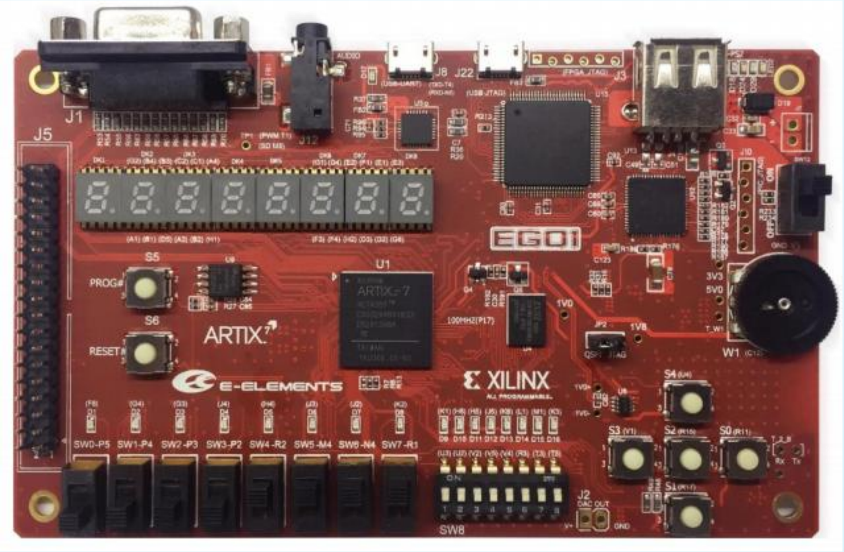
Q1. How to test the CPU ? how to determine the program, the data, how to check the testing result?

I/O interface

Minisys board with FPGA chip embeded



EGO1 board with FPGA chip embeded



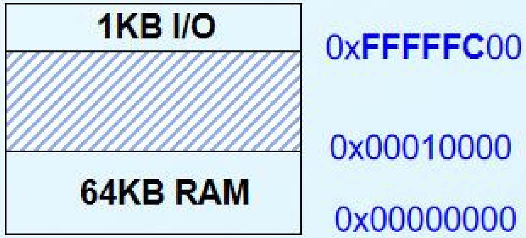
We have practiced a Cropped CPU on EGO1 in lab1

TIPS:

The handbook of board **Minisys** and **EGO1** could be found in the directory "**labs\Handbook_of_Minisys_EGO1**" on the course **BlackBoard sit**

CPU work with I/O

- **Option1： MMIO**
  - lw/sw + $(2^{32} - 64K)$
  - MemOrIO
  - Controller + new control signals



- **Option2： Specific Instruction(s)**
  - Specific instruction(s)
    - $2^{(6+6)}$ - number of R,I,J type
    - e.g. syscall

Code	Basic	Source
0x24020008	addiu $2, $0, 0x00000008	7: li $v0, 8 #to get a string
0x3c011001	lui $1, 0x00001001	8: la $a0, sid
0x34240008	ori $4, $1, 0x00000008	
0x24050009	addiu $5, $0, 0x00000009	9: li $a1, 9
0x0000000c	syscall	10: syscall
0x24020004	addiu $2, $0, 0x00000004	14: li $v0, 4 #to print a string
0x3c011001	lui $1, 0x00001001	15: la $a0, s1
0x34240000	ori $4, $1, 0x00000000	
0x0000000c	syscall	16: syscall
0x00000000	nop	18: nop
0x2402000a	addiu $2, $0, 0x0000000a	20: li $v0, 10 #to exit
0x0000000c	syscall	21: syscall

- **Collaborative work between CPU and I/O**

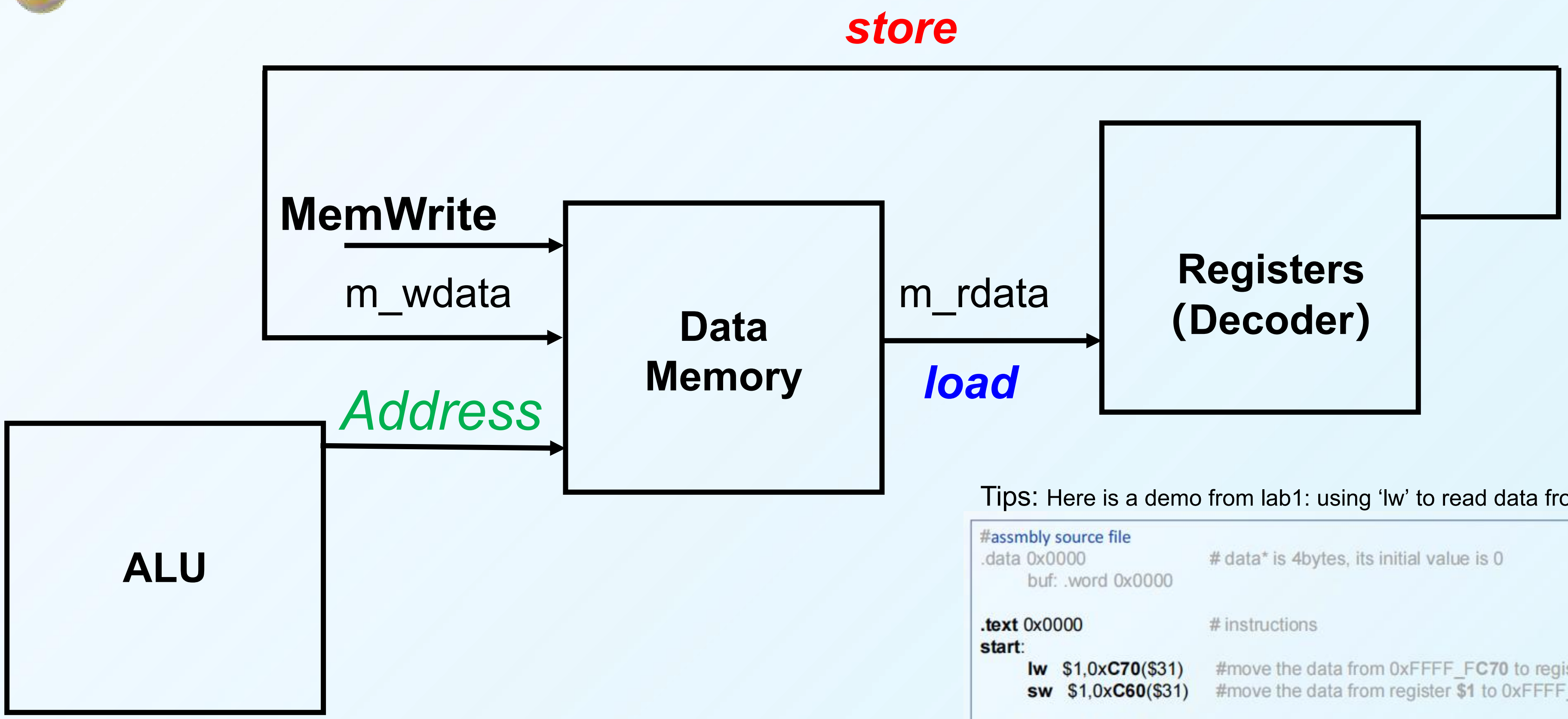
A Simple Design on the I/O Interface

This part mainly accomplishes the following work:

1. Add I/O function
2. 16-bit LED design
3. 16-bit DIP Switch design

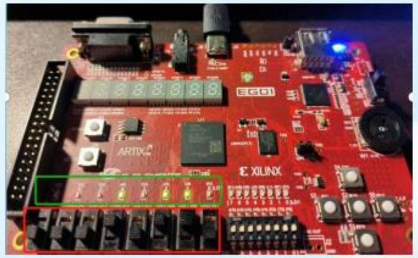
This is only one of the design solutions for I/O related data bus. Please develop a solution that suits your design needs

Option1: MMIO, reuse LW/SW to support I/O



Tips: Here is a demo from lab1: using 'lw' to read data from switchs, 'sw' to write data to leds

```
#assmbly source file
.data 0x0000                    # data* is 4bytes, its initial value is 0
        buf: .word 0x0000

.text 0x0000                    # instructions
start:
        lw   $1,0xC70($31)      #move the data from 0xFFFF_FC70 to register $1
        sw   $1,0xC60($31)      #move the data from register $1 to 0xFFFF_FC60

        lw   $1,0xC72($31)
        sw   $1,0xC62($31)

        j start                 # jump to the instructions labled by start
```

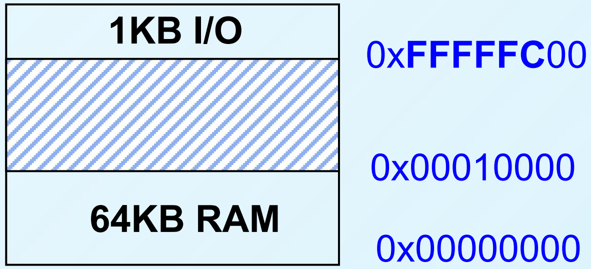


NOTE:

1) There is no specific instruction in Minisys to read data from input ports and write data to output ports.
2) To implement the read/write process on I/O, it needs to **share the load/store instructions** in Minisys.

MMIO: I/O Share Part of the Data Bus Address

The space of **32** bits address bus is **4GB**(0x0000_0000~0xFFFF_FFFF)



1024 bytes(0xFFFF_FC00~0xFFFF_FFFF) is designed to be allocated for the **I/O**.
Chip **S**elect and **address** are specified by specifying **10** IO port lines.

Here is an example for **24** LED lights and **24** DIP switches on Minisys board, both of them are divided into two groups, all the ports in one group share the same address.

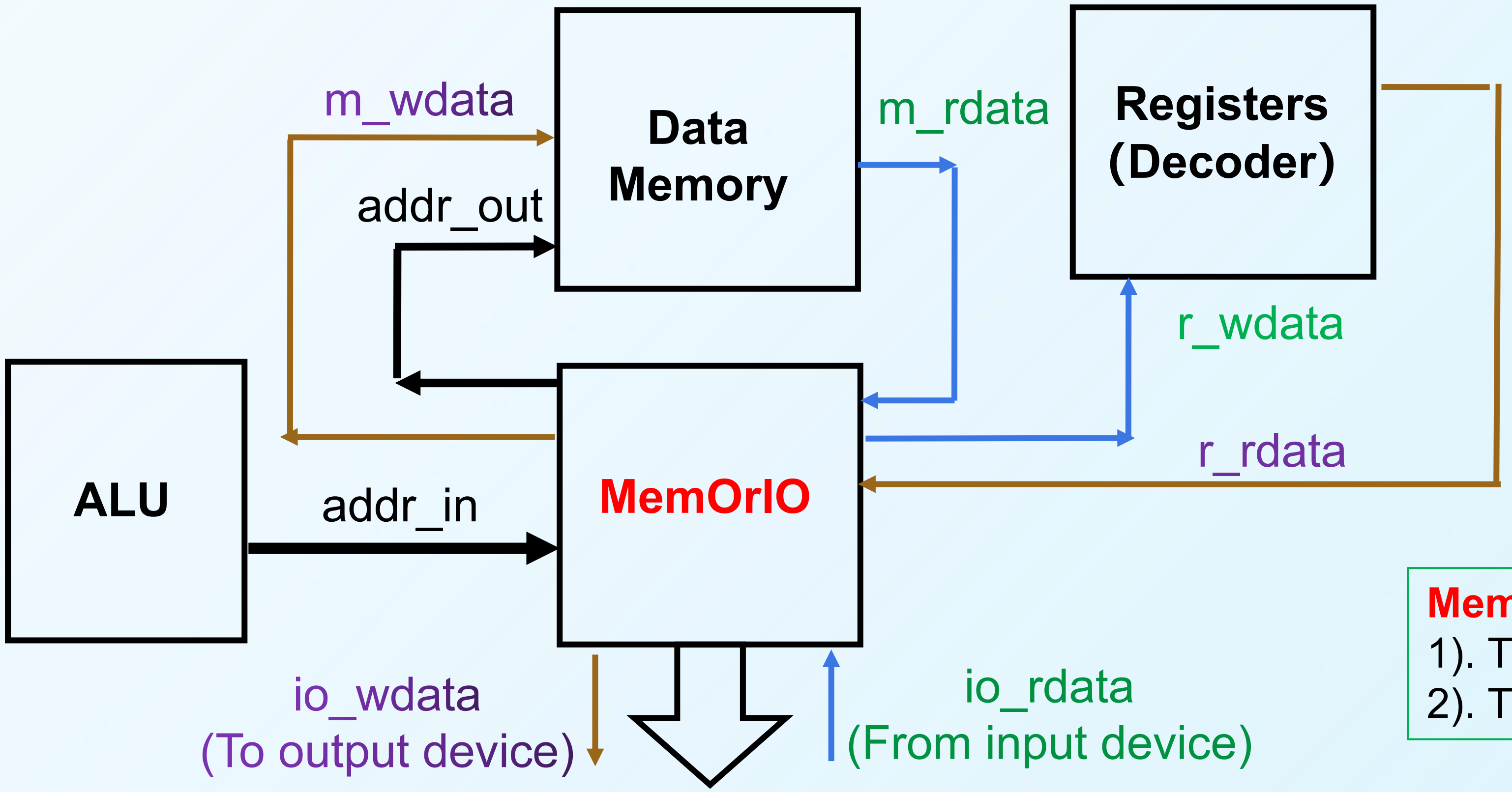
1. The CS(Chip Select) signal of the LED light is **ledCtrl**
2. The CS(Chip Select) signal of the DIP switch is **switchCtrl**

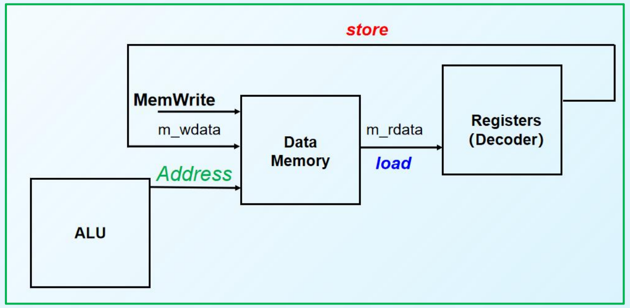
Range	LED(1~16)	LED(17~24)	Switch(1~16)	Switch(17~24)
Address	0x**FFFFFC60**	0x**FFFFFC62**	0x**FFFFFC70**	0x**FFFFFC72**

Note:
*1. In the computer field, there are usually two schemes for I/O address space design: I/O and memory **unified addressing** or **I/O independent addressing**. However there is no dedicated I/ O instruction in current Minisys-1. Here, both LW and SW instructions are used for RAM access and I/O access, which means Minisys-1 can only use I/O unified addressing.*

2. It is just a way for IO address implementation （MMIO: Memory-Mapped Input Output） , but not the only choice.

CPU： add a new module - MemOrIO

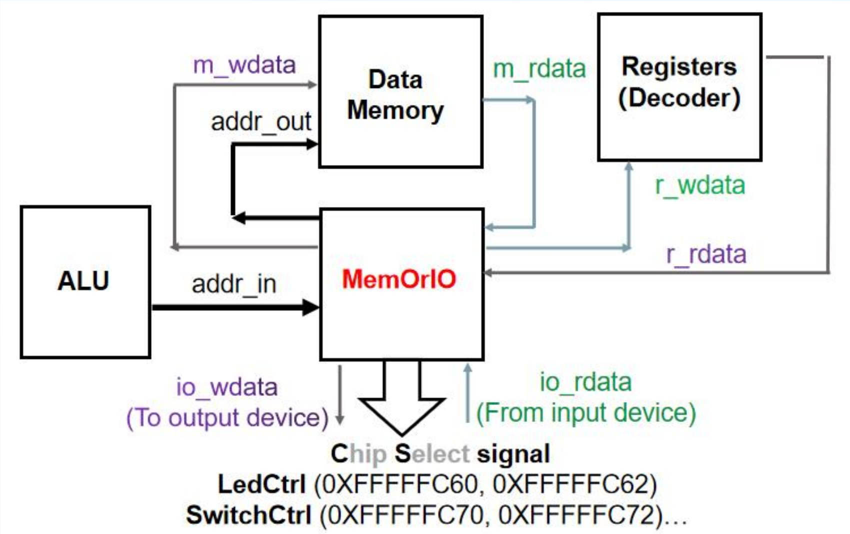




MemOrIO determine:
1). The destination of r_rdata
2). The source of r_wdata

Chip Select signal
LedCtrl (0XFFFFFC60, 0XFFFFFC62)
SwitchCtrl (0XFFFFFC70, 0XFFFFFC72)…

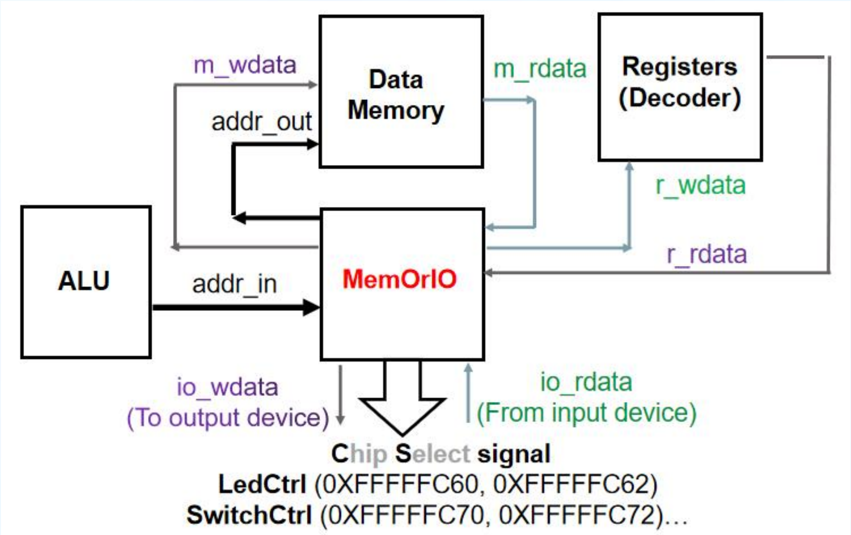
MemOrIO continued




```
module MemOrIO( mRead, mWrite, ioRead, ioWrite,addr_in, addr_out,
m_rdata, io_rdata, r_wdata, r_rdata, write_data, LEDCtrl, SwitchCtrl);

input mRead;                        // read memory, from Controller
input mWrite;                       // write memory, from Controller
input ioRead;                       // read IO, from Controller
input ioWrite;                      // write IO, from Controller

input[31:0] addr_in;                // from alu_result in ALU
output[31:0] addr_out;              // address to Data-Memory

input[31:0] m_rdata;                // data read from Data-Memory
input[15:0] io_rdata;               // data read from IO,16 bits
output[31:0] r_wdata;               // data to Decoder(register file)

input[31:0] r_rdata;          // data read from Decoder(register file)
output reg[31:0] write_data;        // data to memory or I/O (m_wdata, io_wdata)
output LEDCtrl;                     // LED Chip Select
output SwitchCtrl;                  // Switch Chip Select
```

*Tips: A demo about how the **Chip Select** signals work on I/O could be found in **labs/lab11_io** on course BlackBoard site*

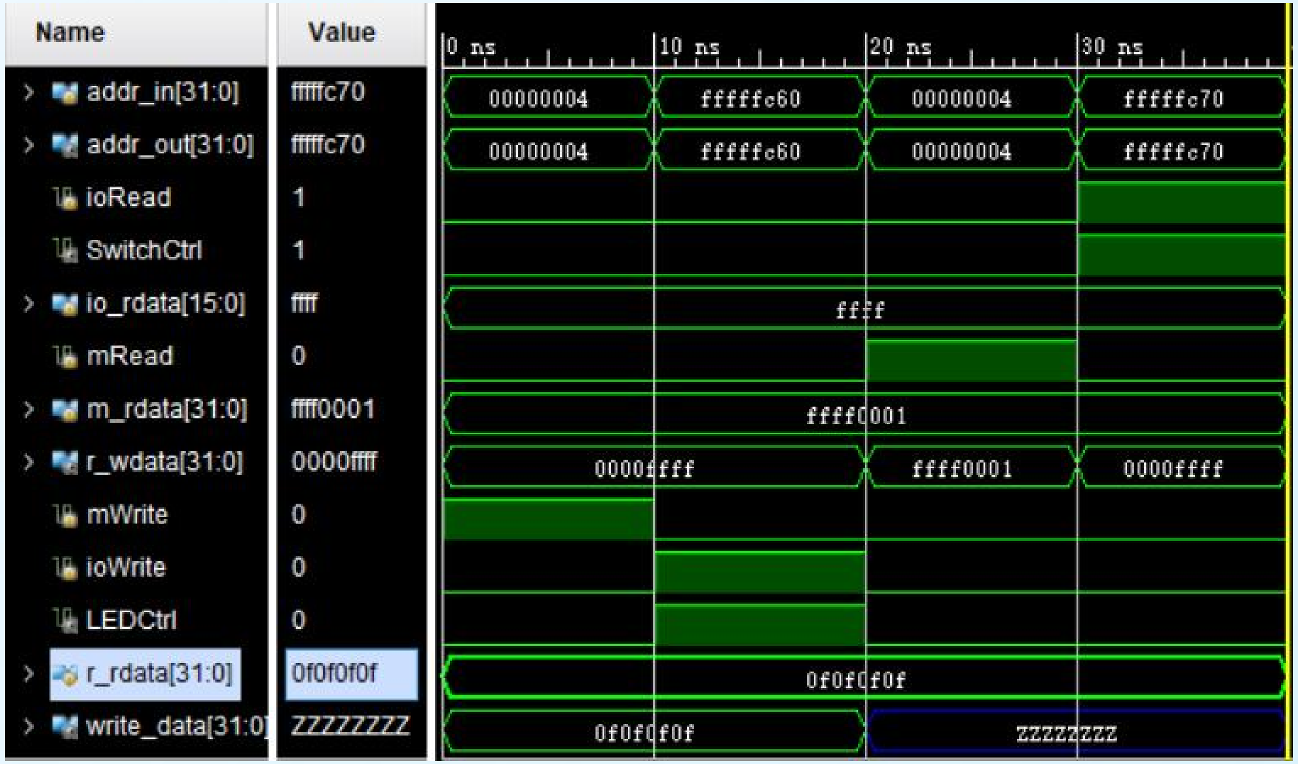
MemOrIO continued




```
assign addr_out= addr_in;
// The data wirte to register file may be from memory or io.
// While the data is from io, it should be the lower 16bit of r_wdata.
assign r_wdata = ？ ？ ？

// Chip select signal of  Led and Switch  are all active high;
assign LEDCtrl=  ？ ？ ？
assign SwitchCtrl= ？ ？ ？

always @* begin
        if((mWrite==1)||(ioWrite==1))
              //wirte_data could go to either memory or IO. where is it from?
                write_data = ？ ？ ？
        else
                write_data = 32'hZZZZZZZZ;
end
endmodule
```

The Function Verification of MemOrIO

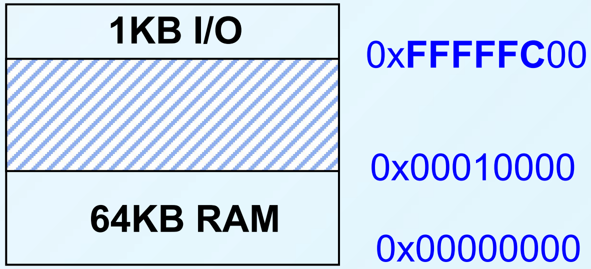
```
 // a reference for the testbench of MemOrIO
module MemOrIO_tb( );
   reg mRead,mWrite,ioRead,ioWrite;
   reg[31:0] addr_in,m_rdata,r_rdata;
   reg[15:0] io_rdata;
   wire LEDCtrl,SwitchCtrl;
   wire [31:0] addr_out,r_wdata,write_data;

   MemoryOrIO umio(addr_out, addr_in,
mRead, mWrite, ioRead, ioWrite,
m_rdata, io_rdata, r_rdata, r_wdata, write_data,
LEDCtrl, SwitchCtrl );
```




```
  initial begin    // r_rdata -> m_wdata(write_data)
  m_rdata = 32'h0xffff_0001;  io_rdata = 16'h0xffff;  r_rdata = 32'h0x0f0f_0f0f;    addr_in = 32'h4;   {mRead,mWrite,ioRead,ioWrite}= 4'b01_00;
  #10   addr_in = 32'hffff_fc60;   {mRead,mWrite,ioRead,ioWrite}= 4'b00_01;           // r_rdata -> io_wdata(write_data)
  #10  addr_in = 32'h0000_0004;    {mRead,mWrite,ioRead,ioWrite}= 4'b10_00;       // m_rdata -> r_wdata
  #10  addr_in = 32'hffff_fc70;     {mRead,mWrite,ioRead,ioWrite}= 4'b00_10;          // io_rdata -> r_wdata(write_data)
  #10 $finish;
   end
endmodule
```

Controller+

Add new ports to Controller for IO reading and writing support.




```
module  control32(Opcode,Function_opcode,Jr,Branch,nBranch,Jmp,Jal,
Alu_resultHigh,
RegDST, MemorIOtoReg, RegWrite,
MemRead, MemWrite,
IORead, IOWrite,
ALUSrc,ALUOp,Sftmd,I_format);
        ...
        input[21:0] Alu_resultHigh;     // From the execution unit Alu_Result[31..10]
        output MemorIOtoReg;     // 1 indicates that data needs to be read from memory or I/O to the register
        output RegWrite;         // 1 indicates that the instruction needs to write to the register
        output MemRead;          // 1 indicates that the instruction needs to read from the memory
        output MemWrite;         // 1 indicates that the instruction needs to write to the memory
        output IORead;           // 1 indicates  I/O read
        output IOWrite;          // 1 indicates  I/O write
        ...
```

Controller+ continued

1) Modify the logic of the '***MemWrite***'
2) Add '*MemRead*', '***IORead***' and '***IOWrite***' signals
3) Change '*MemtoReg*' to '***MemorIOtoReg***'.

```
 // The real address of LW and SW is Alu_Result, the signal comes from the execution unit
// From the execution unit Alu_Result[31..10], used to help determine whether to process Mem or IO
  input[21:0]  Alu_resultHigh;

   output      MemorIOtoReg;         //1 indicates that read date from memory or I/O to write to the register
   output      MemRead;              // 1 indicates that reading from the memory to get data
   output      IORead;               // 1 indicates  I/O read
   output      IOWrite;              // 1 indicates  I/O write

   assign RegWrite = (R_format || Lw || Jal || I_format) && !(Jr) ;        // Write memory or write IO
   assign MemWrite = ((sw==1) && (Alu_resultHigh[21:0] != 22'h3FFFFF)) ? 1'b1:1'b0;
   assign MemRead = ? ? ?                          // Read memory
   assign IORead  = ? ? ?                          // Read input port
   assign IOWrite     = ? ? ?                      // Write output port

// Read operations require reading data from memory or I/O to write to the register
   assign MemorIOtoReg = IORead || MemRead;
```

Option2: Specific instruction for I/O

- Use **specific instructions** and **independent address spaces** to access and address I/O devices(e.g. IN/OUT instructions used in **x86** ISA to access the I/O device)

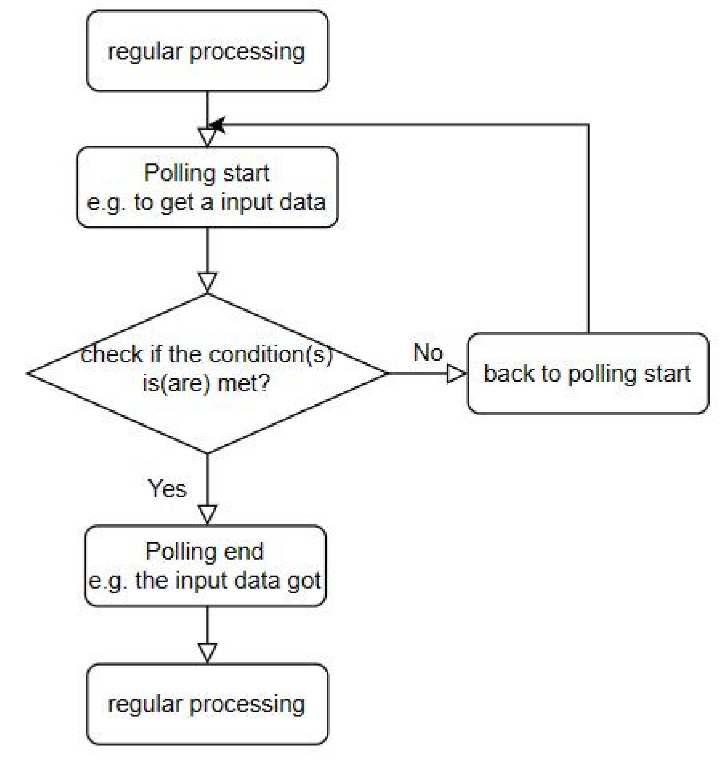
- Implementation the solution about Specific instruction for I/O on Minsys
  - **ISA** (add new instructions about I/O access)
  - **Assembler** (recognizes and supports the added instructions)
  - **CPU** (Modification)
    - **Control Path**
      - Controller: **distiguish the IO and the data-memory by the opcode and function code in the instruction instead of the address calculated by the ALU**
      - The **control signal** to the Data Path
    - **Data Path**
      - The **data path** between the **Decoder**, **Data-Memory**, **I/O module** and **ALU**

How about '**syscall**' ? ?
(0x0000_000c)

Collaborative work between CPU and I/O

How to collaborate between IO devices and CPUs?

- Issue1: speed mismatch between sender and receiver
  - solution: cache, fifo
- Issue2: out of sync on the communication between sender and receiver
  - solution1 of issue2
    Polling between waiting and checking:
    - **when the condition(s) is(are) met**(e.g. the input data is ready), **continue the regular process**
    - **when the condition(s) is(are) NOT met, do nothing except checking while 'waiting'.**

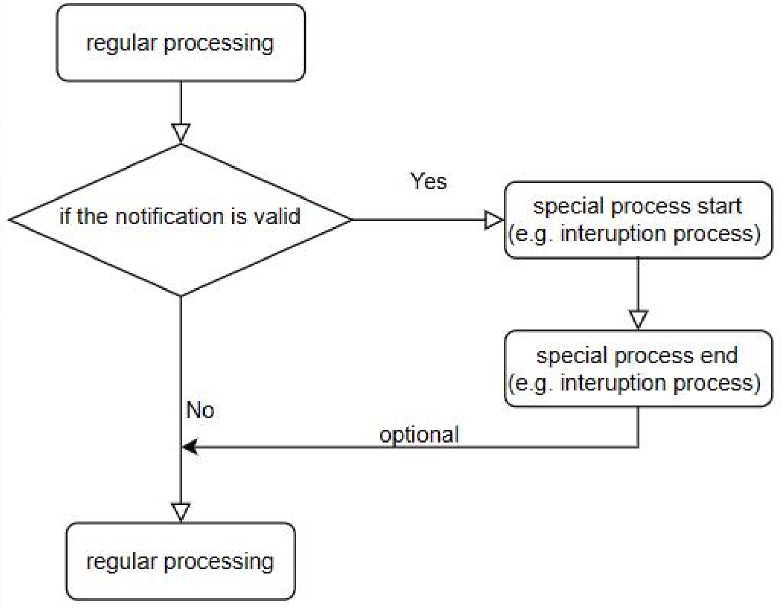


How to 'wait' ? ?
What's your solution about 'wait' ? ?

Collaborative work between CPU and I/O continued

Solution2 on the issue2(**out of sync on the communication between sender and receiver**)

- When there is **NO notification**, **do regular process** (other things) first.
- When there is a **notification**
  - **stop for specialtreatment**(e.g. excepiton/interrupt process)
  - after finish the specialtreatmen then return to(optional) the regular process to continue.



- How to Generate and identify the notification(s)?
  - The internal signals of CPU
    - e.g. exception about overflow (generated by the ALU)
  - The external signals of CPU
    - e.g. interuption from input device

Which module(s) in CPU receive and identify the notification ? ?

- How to 'wait'?
  - do something but meaningless
    - nop (sll $0,$0,0)
  - do nothing
    - adding chipsel / enable control on the sub-modules in CPU

Which solution about wait lead to lower power consumption ??

- How to determine the waiting time?
  - The time of the instruction cycle * the number of instructions

Practice

P1-1. Do the functional verification on the module cpuclk(which is introduce on the first part of this lab)

P1-2.Answer the Q2 on page 2 and Q1 on page 7 of this lab slides.

P2. Complete the following modules, do the function verification:

- 1. MemoryOrIO
- 2. Controller+
- 3. Sigle cycle CPU with I/O process

P3. Redesign and implement the solution about I/O data bus and I/O addressing that are suitable for your design. Build the single cycle CPU with the updated solution of I/O process and do the function verification.

P4. Design the solution on collaborate between IO devices and CPU in your teamwork of CPU, evaluate work in software and hardware collaborative development.