NYCU-EE IC LAB – Spring2023

Final project

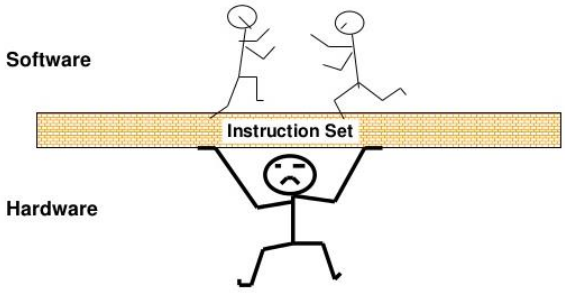
Design: Customized ISA Processor

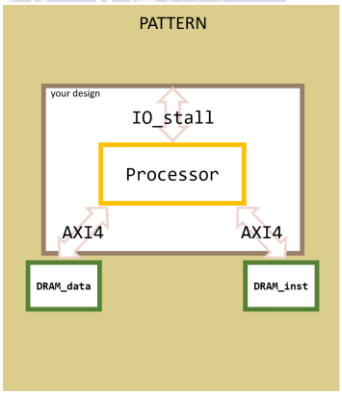
Data Preparation

1. Extract test data from TA's directory:
   % tar xvf ~iclabta01/Final_Project_2023_spring.tar

Design Description

A reduced instruction set computer (RISC) is a computer instruction set (AKA: the instruction set architecture (ISA)) which allows a computer's microprocessor to have fewer cycles per instruction (CPI) than a complex instruction set computer (CISC) [from wikipedia]. In this exercise, you are asked to design a 16-bits CPU with 16 registers and the CPU should be available for the required instruction set, which shows below (very similar to MIPS).





The following are the three formats for the core instruction set

Type	MSB	- Format – (16-bits)			LSB
R	opcode (3-bits)	rs (4-bits)	rt (4-bits)	rd (4-bits)	func (1-bits)
I	opcode (3-bits)	rs (4-bits)	rt (4-bits)	immediate (5-bits)	
J	opcode (3-bits)	Address (13-bits)			

Register s(rs), Register t(rt) and Register d(rd) represent the address of registers. Since the instruction takes 4 bits to store the address, it means we have 16 registers, from r0 to r15. Each register reserve 16 bits to store data, e.g. rs = 4'b0000 means one of operands is "r0". rt = 4'b0101 means one of operands is "r5", and so on.

Detailed instructions are shown below

Function Name	Meaning	Type	Instruction Binary Encode
ADD	rd = rs + rt	R	000-rs-rt-rd-1
SUB	rd = rs – rt	R	000-rs-rt-rd-0
Set less than	if(rs<rt) rd=1 else rd=0	R	001-rs-rt-rd-1
Mult	rd = rs * rt (16'b LSB)	R	001-rs-rt-rd-0
Load	rt = DM[sign(rs+immediate)× 2 + offset]	I	011-rs-rt-iiiii
Store	DM[sign(rs+immediate)× 2 + offset] = rt	I	010-rs-rt-iiiii
Branch on equal	if(rs==rt) pc=pc+1+immediate(sign) else pc=pc+1 (Please check out the red notes below to learn more about the concept of pc.)	I	101-rs-rt-iiiii
Jump	Next instruction address = address (range: 0x1000~0x1fff)	J	100-address

*pc: program counter, meaning the next instruction. For R type, store and load instructions, pc should automatically +1 to get the next instruction.

*Each instruction has 16bits, which means the instruction address should +2 to get the next instruction. Therefore, you should remember to multiply the immediate of branch on equal function by 2 as well to get the correct address.

*offset = 0x1000

*The first instruction address you acquire data should be "0x00001000".

```
@1000
25 66 // I_inst: load 0110011000100101 rs =  3, rt =  1, imm =  5
@1002
40 76 // I_inst: load 0111011001000000 rs = 11, rt =  2, imm =  0
@1004
60 72 // I_inst: load 0111001001100000 rs =  9, rt =  3, imm =  0
```

```
@1000
e5 ff
@1002
03 00
@1004
f2 ff
```

One instruction DRAM and one data DRAM are used in this lab. You should be careful if you wish to use the pipelining architecture, e.g. basic five-stage pipeline in a RISC machine, you must deal with data hazard problem.

Inputs & Output

I/O	Bit	Signal Name	Description
Input	1	clk	Positive edge trigger clock within clock period below 20.0ns
Input	1	rst_n	Asynchronous reset active low reset
Output	1	IO_stall	Pull high when core is busy. It should be low for one cycle whenever you finished an instruction. TA will check the values of your registers when your IO_stall is low. (refer to

			appendix to see how to check the register value in PATTERN)

- All inputs will be changed at clk ***negative*** edge.
- There is ***only 1 reset*** before the first pattern, thus, your design must be able to reset automatically.
- The test pattern will check the value in all registers at clock ***negative*** edge if stall is low.
- The test pattern will check the value in data DRAM every 10 instruction at clock negative edge if IO_stall is low.
- All the registers should be zero after the reset signal is asserted.
- The value in all registers should be **unchanged** when stall is low.

AXI 4 Interface (Connected with DRAM in Pattern)

In this project, the signal naming rule for AXI4 related signals is: AXI4 signal name (lower case) + _m_inf (suffix). Because we have two AXI4 interface, one of them is read only while the other one is for read write, so the port number for read port should be multiplied by two while for writing port it should only be multiplied by one. For example, the port width for ARADDR [63:0] and AWADDR [31:0] should be:

```
parameter ID_WIDTH = 4 , ADDR_WIDTH = 32, DATA_WIDTH = 16, DRAM_NUMBER=2, WRIT_NUMBER=1;

output  wire [WRIT_NUMBER * ID_WIDTH-1:0]        awid_m_inf;
output  wire [WRIT_NUMBER * ADDR_WIDTH-1:0]    awaddr_m_inf;
output  wire [DRAM_NUMBER * ID_WIDTH-1:0]        arid_m_inf;
output  wire [DRAM_NUMBER * ADDR_WIDTH-1:0]    araddr_m_inf;
```

ARADDR[63:32] is the address port for DRAM_inst

ARADDR [31:0] is the address port for DRAM_data

AWADDR [31:0] is the address port for DRAM_data

Write Address Channel

Signal	Source	Description
AWID[3:0]	Master	Write address ID. This signal is the identification tag for the write address group of signals. (In this project, we only use this to recognize master, reordering method is not supported)
AWADDR[31:0]	Master	Write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
AWLEN[6:0]	Master	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
AWSIZE[2:0]	Master	Burst size. This signal indicates the size of each transfer in the burst. (In this project, we only support 3'b001 which is 2 Bytes (matched with Data Bus-width) in each transfer)
AWBURST[1:0]	Master	Burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. (only INCR(2'b01) is supported in this Project)
AWVALID	Master	Write address valid. This signal indicates that valid write address and control information are available: 1 = address and control information available 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, **AWREADY**, goes HIGH.
AWREADY	Slave	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.

Write Data Channel

Signal	Source	Description
WDATA[15:0]	Master	Write data. The write data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide. (In this project, we only support 16 bit data width: WDATA[15:0])
WLAST	Master	Write last. This signal indicates the last transfer in a write burst.
WVALID	Master	Write valid. This signal indicates that valid write data and strobes are available: 1 = write data and strobes available 0 = write data and strobes not available.
WREADY	Slave	Write ready. This signal indicates that the slave can accept the write data: 1 = slave ready 0 = slave not ready.

Write Response Channel

Signal	Source	Description
BID[3:0]	Slave	Response ID. The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.
BRESP[1:0]	Slave	Write response. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. (only OKAY(2'b00) is supported in this Project)
BVALID	Slave	Write response valid. This signal indicates that a valid write response is available: 1 = write response available 0 = write response not available.
BREADY	Master	Response ready. This signal indicates that the master can accept the response information. 1 = master ready 0 = master not ready.

Read Address Channel

Signal	Source	Description
ARID[3:0]	Master	Read address ID. This signal is the identification tag for the read address group of signals. (In this project, we only use this to recognize master, reordering method is not supported)
ARADDR[31:0]	Master	Read address. The read address bus gives the initial address of a read burst transaction.
ARLEN[6:0]	Master	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
ARSIZE[2:0]	Master	Burst size. This signal indicates the size of each transfer in the burst. (In this project, we only support 3'b001 which is 2 Bytes (matched with Data Bus-width) in each transfer)
ARBURST[1:0]	Master	Burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. (only INCR(2'b01) is supported in this Project)
ARVALID	Master	Read address valid. This signal indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledge signal, **ARREADY**, is high. 1 = address and control information valid 0 = address and control information not valid.
ARREADY	Slave	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.

Read Data Channel

Signal	Source	Description
RID[3:0]	Slave	Read ID tag. This signal is the ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.
RDATA[15:0]	Slave	Read data. The read data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide. (In this project, we only support 16 bit data width: RDATA[15:0])
RRESP[1:0]	Slave	Read response. This signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. (only OKAY(2'b00) is supported in this Project)
RLAST	Slave	Read last. This signal indicates the last transfer in a read burst.
RVALID	Slave	Read valid. This signal indicates that the required read data is available and the read transfer can complete: 1 = read data available 0 = read data not available.
RREADY	Master	Read ready. This signal indicates that the master can accept the read data and response information: 1= master ready 0 = master not ready.

Specifications

1. Top module name: **CPU** (top file name: **CPU.v**)

2. It is **asynchronous** reset and **active-low** architecture. **rst_n** would active once at the beginning. All the registers should be zero after the reset signal asserts.
3. All the data are in **signed** format in all registers.
4. The maximum clock period is set to **20ns**, and **you can determine the clock period by yourself**.
5. The input delay and the output delay should be **half** of the clk period, if clock period is 10ns, the input delay and the output delay will be 5ns.
6. IO_stall signal cannot be continuous high for 2000 cycles during functionality check, cannot be continuous high for 100000 during performance check.
7. The output loading is set to **0.05**. The wire load is **top** and target library is **slow.db**
8. The synthesis result of data type cannot include any **LATCH** (in syn.log).
9. In this project, you should **modify the syn.tcl (e.g. link library for your memory.db) by yourself**. **compile_ultra** will be used to synthesis. Since memories are used in this project, the syn.tcl in Lab05 may be a good reference one.
10. You **CANNOT PASS** the demo if there are **timing violation** messages in the gate level simulation WITHOUT notimingchecks option
11. No **ERROR** is allowed in every simulation/synthesis.
12. **You must use SRAM in your design or you wouldn't get any point in this project.**
13. The maximum area should be smaller than **2,000,000.**
14. **You cannot change the register name or type from the original file.**
15. PATTERN will not generate any overflow cases during calculation.
16. The program counter will not over access range of memory when the jump instruction asserts. So, you need to think how to write your pattern by yourself.
17. For achieving the data dependence in real case, the next load/store address will be constrained in certain range for data memory and the address for jump/branch instruction will also be constrained in certain range. Please refer to appendix.
18. Please refer the Check_list.pdf file for APR routing specs.
19. The latency between BVALID and BREADY, RREADY and RVALID should not be larger than **300** cycles.
20. DRAM must be declared in PATTERN, **do not declare it in your DESIGN.**
21. Don't use any wire/reg/submodule/parameter name called *error*, *latch*, *congratulation* or *fail* otherwise you will fail the lab. Note: * means any char in front of or behind the word. e.g: error_note is forbidden.
22. Don't write Chinese comments or other language comments in the file you turned in.
23. Verilog commands
    //synopsys dc_script_begin, //synopsys dc_script_end

//synopsys translate_off, //synopsys translate_on
Are only allowed during the usage of including and setting designware IPs, other design compiler optimizations are forbidden.

24. Using the above commands are allowed, however any error messages during synthesize and simulation, regardless of the result will lead to failure in this lab.
25. Any form of display or printing information in verilog design is forbidden. You may use this methodology during debugging, but the file you turn in should not contain any coding that is not synthesizable.

APR Specifications

1. Floor plan:
   a. Core to IO boundary must be more than 100.
2. Power plan:
   a. Power ring must be wire group, interleaving, at least 4 pairs, width at least 9
   b. Stripes sets distance should be less than 200, width at least 4.
3. Timing slack shouldn't have any negative slacks after setup/hold time analysis (including SI).
4. The DRV of (fanout, cap, tran) should be all 0 after post_Route setup/hold time analysis (including SI).
5. No LVS violation after "verify connectivity".
6. No DRC violation after "verify geometry".
7. Core filler must be added.
8. SRAM (hard macro) must be placed inside core region.

Grading Policy

1st Demo (100%), 2nd Demo (70% of total)

- Synthesis, RTL & Gate Level Simulation Correctness, APR and Post Level Simulation Correctness: 60%
  You can only get this score and pass the demo with correct synthesis, APR specs, post layout and gate-level simulation (including functionality check and performance evaluation).
- Performance: 40%
  1. **Performance: (Total Cycle × Clock Period)$^{1.5}$ x Core Area**
  2. You can only get performance score when you pass the demo.

If you fail the 1.a, 2.a, 2.b APR specs and pass all the other checks, you can still pass demo, but each violation will deduct 5 points. (the sub-points in 2.a and 2.b are counted separately)
TA will demo your design with 2 different patterns, one is for functionality check, the other is for performance evaluation.

For functionality check, the DRAM latency is in between 1 to 20, not fixed.
For performance evaluation, the DRAM latency is above 100 but below 1000, not fixed.

Note

1. Please submit your files under 09_SUBMIT before **12:00 noon**.
   Due Day: 1st Demo: June 5th 2nd Demo: June 7th

   In this lab, you can adjust your clock cycle time. Consequently, make sure to key in your pre-layout and post-layout clock cycle time after the command like the figure below.

```
./01_submit 10.0 15.0
```

   It means that the TA will demo your:

   a. RTL and Gate Level design with pre-layout clock cycle time

   b. APR Level design with post-layout clock cycle time.

   c. After that, you should check the following files under **09_SUBMIT/ Final_Project_iclabXXX.tar /**

      RTL design: CPU_iclabXXX.v (XXX is your account no.)
      clock_cycle_iclabXXX_pre.txt
      DC file: syn_ iclabXXX.tcl
      Memory file: 04_MEM_iclabXXX
      (with all your memory files: .v / .db / .vclef / .lib)
      file_list_iclabXXX.f
      APR file: CHIP_iclabXXX.sdc
      CHIP_iclabXXX.inn
      CHIP_iclabXXX.io
      CHIP_iclabXXX.v
      CHIP_iclabXXX.sdf
      CHIP_iclabXXX.inn.dat
      clock_cycle_iclabXXX_post.txt

      If you miss any files on the list, you will fail this lab.
      If any error occurs when restoring your design, you will FAIL the lab
      Then use the command like the figure below to check the files are uploaded or not.

```
[Exercise/09_SUBMIT]% ./02_check 1st_demo
```

Check your APR files carefully by the list provided in check_list.pdf, any

APR file restore error will lead to failure in this lab.

2. Template folders and reference commands:

   01_RTL/ (for RTL simulation) **./01_run**

   02_SYN/ (for Synthesis) **./01_run_dc**

   Remember modify .tcl to fit your memory db name

   (Check the design which contains **Latch** and **Error** or not in **syn.log**)

   (Check the design's timing in /Report/***CPU.timing*** to see if the slack is **MET**)

   03_GATE / (Gate Level simulation) **./01_run**

   You can key in **./09_clean_up** to clear all log files and dump files in each folder

   04_MEM/ (Memory location)

   You should generate your memories and put the required files (.v and .db) here

   05_APR/(back-end APR) **./09_clean_up** (clean all log and command files)

   06_POST /(Post-layout simulation) **./01_run**

Appendix

- TA will check the value in data memory **every ten instructions** when IO_stall is low.
- TA will check your registers by the following command in PATTERN:

(RTL & GATE level)

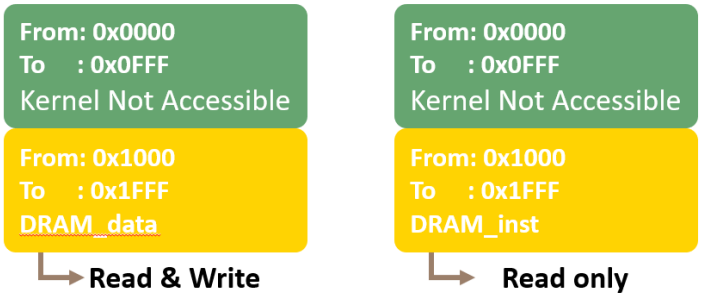
```
if((My_CPU.core_r0  !== golden_reg[0 ]) || (My_CPU.core_r8  !== golden_reg[8 ]) ||
   (My_CPU.core_r1  !== golden_reg[1 ]) || (My_CPU.core_r9  !== golden_reg[9 ]) ||
   (My_CPU.core_r2  !== golden_reg[2 ]) || (My_CPU.core_r10 !== golden_reg[10]) ||
   (My_CPU.core_r3  !== golden_reg[3 ]) || (My_CPU.core_r11 !== golden_reg[11]) ||
   (My_CPU.core_r4  !== golden_reg[4 ]) || (My_CPU.core_r12 !== golden_reg[12]) ||
   (My_CPU.core_r5  !== golden_reg[5 ]) || (My_CPU.core_r13 !== golden_reg[13]) ||
   (My_CPU.core_r6  !== golden_reg[6 ]) || (My_CPU.core_r14 !== golden_reg[14]) ||
   (My_CPU.core_r7  !== golden_reg[7 ]) || (My_CPU.core_r15 !== golden_reg[15]))begin
```

(APR & POST)

```
if((My_CHIP.core_r0  !== golden_reg[0 ]) || (My_CHIP.core_r8  !== golden_reg[8 ]) ||
   (My_CHIP.core_r1  !== golden_reg[1 ]) || (My_CHIP.core_r9  !== golden_reg[9 ]) ||
   (My_CHIP.core_r2  !== golden_reg[2 ]) || (My_CHIP.core_r10 !== golden_reg[10]) ||
   (My_CHIP.core_r3  !== golden_reg[3 ]) || (My_CHIP.core_r11 !== golden_reg[11]) ||
   (My_CHIP.core_r4  !== golden_reg[4 ]) || (My_CHIP.core_r12 !== golden_reg[12]) ||
   (My_CHIP.core_r5  !== golden_reg[5 ]) || (My_CHIP.core_r13 !== golden_reg[13]) ||
   (My_CHIP.core_r6  !== golden_reg[6 ]) || (My_CHIP.core_r14 !== golden_reg[14]) ||
   (My_CHIP.core_r7  !== golden_reg[7 ]) || (My_CHIP.core_r15 !== golden_reg[15]))begin
```

Please make sure your registers can be read by PATTERN after synthesis and APR, otherwise if TA's pattern can not access your register, you will fail your demo.

- Memory available range



- Store, load, branch and jump address

  The address value of the above instructions must be within the available address range for DRAM (0x00001000~0x00001FFF).

- Instruction

  Ex. 000-0001-0010-0011-0 means r1 - r2 and saves the result into r3.

  The first instruction address you acquire data should be "0x00001000".

- Data Dependence Prediction

  For achieving the data dependence in real case, the next load/store address will be constrained in certain range for data memory and the address for jump/branch instruction will also be constrained in certain range. The range is from (current address – 64 + 2) to (current address + 64), e.g. if current address is dec(1000), the next load/store address will be from dec(938) to dec(1064).