4DV4-MiniProject-02
Simple MIPS CPU

COMPENG 4DV4: Very Large Scale Integration System Design

Deadline

Last day to submit this project is Tuesday March 10 (10/3/2026) by 11:59 p.m

1 Data Preparation

1. Extract the assignment using:

```
tar -xvf 1101_hw2.tar
```

2. The directory includes the following components:

Folder	File	Description
00_TESTBED	testbed.vp	Protected testbench module
	inst_mem.vp	Protected instruction memory module
	data_mem.vp	Protected data memory module
	define.v	Definition file
	testbed_temp.v	Testbench template
00_TESTBED/PATTERN	p*	Instruction bitstreams
	inst_assemble.dat	Assembly mapping of instruction patterns
	data.dat	Expected final data memory content
	status.dat	Expected processing status
01_RTL	core.v	Your CPU design
	rtl.f	RTL file list
01_run and 99_clean_up scripts included for simulation support		

2 Introduction

The CPU serves as the main computational controller in a digital system. In this assignment, you must implement a simplified MIPS-style CPU core integrating the program counter, ALU, and register file. A protected testbench environment is provided, but you are required to design a supplemental testbench to verify your implementation. You will also be required to do the logical synthesis and calculate the power and perform timing analysis.

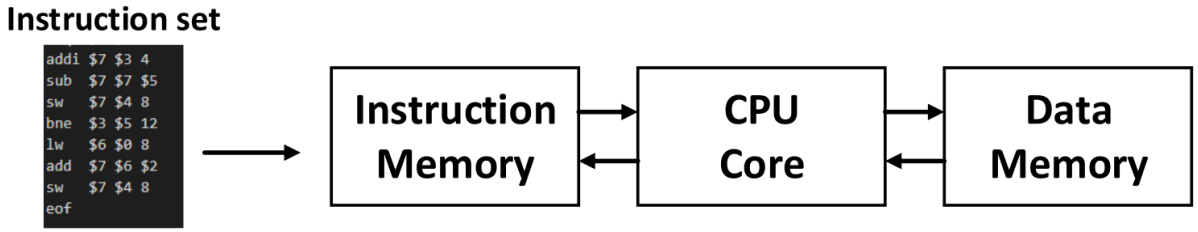


Figure 1: CPU workflow

3 System Architecture

Your core interacts with instruction and data memory modules via specified interfaces. The processor executes a small set of MIPS-like instructions for arithmetic, logical, memory, and control-flow operations.

4 Specifications

1. Top module: `core`
2. I/O interface signals:

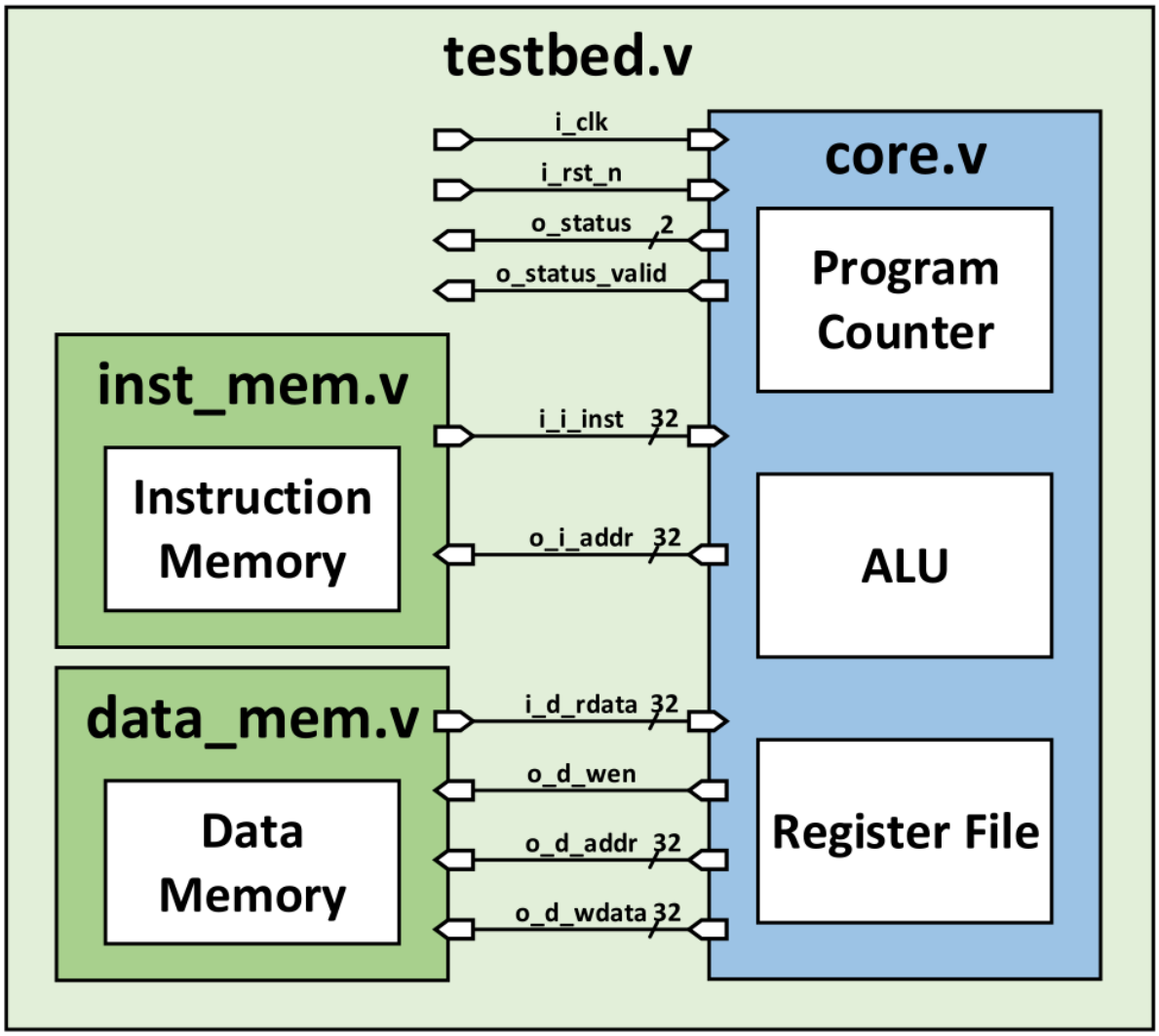


Figure 2: Block Diagram

Signal	I/O	Width	Description
i_clk	I	1	System clock
i_rst_n	I	1	Active-low asynchronous reset
o_i_addr	O	32	Fetch address issued to instruction memory
i_i_inst	I	32	Instruction word received from instruction memory
o_d_wen	O	1	Data memory write enable (0 = read, 1 = write)
o_d_addr	O	32	Data memory address
o_d_wdata	O	32	Write data for memory store instructions
i_d_rdata	I	32	Read data from memory load instructions
o_status	O	2	Processing status of each completed instruction
o_status_valid	O	1	Assert high for one cycle when status is valid

3. All outputs must be registered (updated on clock rising edge).
4. Reset must clear all outputs and zero initialize all registers.
5. Memories are pre-initialized to zero.
6. Register file must implement 32 unsigned 32-bit registers.
7. Instruction fetch latency: instruction is valid 1 cycle after issuing PC address.
8. Load latency: read data valid 1 cycle after issuing memory read address.
9. `o_status_valid` must pulse high for exactly one cycle per instruction.
10. Stop CPU execution immediately when:
    - `o_status = 3` (EOF instruction reached)
    - `o_status = 2` (Overflow detected)
11. Maximum instruction count per pattern: 1024
12. Maximum total execution cycles: 120000

5 Design Description

5.1 Program Counter Behavior (for every instruction except branching)

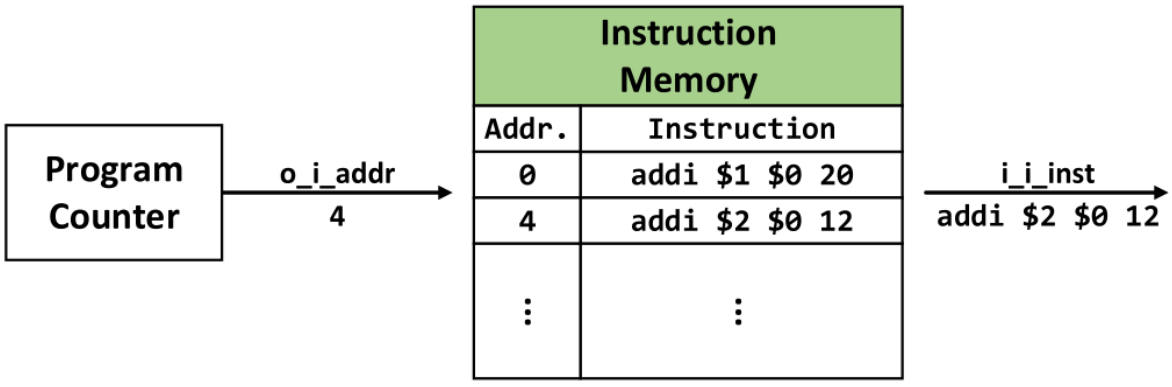


Figure 3: Program counter

$$PC = PC + 4$$

Exceptions: conditional branches apply relative addressing based on offset field.

5.2 Register File

Create 32 general-purpose unsigned registers, each 32 bits wide.

5.3 Instruction Formats

R-Type

[31:26] opcode | [25:21] $s2 | [20:16] $s3 | [15:11] $s1 | [10:0] unused

I-Type

[31:26] opcode | [25:21] $s2 | [20:16] $s1 | [15:0] immediate

EOF Instruction

[31:26] opcode (eof) | [25:0] unused

5.4 Instruction Set Requirements

`Operation`	Opcode	Type	Behavior
`add`	6'd1	R	$s1 = $s2 + $s3
`sub`	6'd2	R	$s1 = $s2 - $s3
`addi`	6'd3	I	$s1 = $s2 + immediate
`lw`	6'd4	I	$s1 = Mem[$s2 + immediate]
`sw`	6'd5	I	Mem[$s2 + immediate] = $s1
`and`	6'd6	R	Bitwise: $s1 = $s2 & $s3
`or`	6'd7	R	Bitwise: $s1 = $s2 — $s3
`nor`	6'd8	R	Bitwise: $s1 = ($s2 — $s3)
`beq`	6'd9	I	If equal, branch relative offset
`bne`	6'd10	I	If not equal, branch relative offset
`slt`	6'd11	R	$s1 = ($s2 ¡ $s3)? 1 : 0
`eof`	6'd12	EOF	Stop execution

Immediate values in I-type format are treated as unsigned.

5.5 Memory Interfaces

- Instruction memory: 1024×32
- Address mapping uses i_addr[11:2]
- Data memory: 64×32
- Address mapping uses i_addr[7:2]

```
module inst_mem (
    input                   i_clk,    // 1-bit
    input                   i_rst_n,  // 1-bit
    input  [ 31 : 0 ]       i_addr,   // 32-bit
    output [ 31 : 0 ]       o_inst    // 32-bit
);
```

```
module data_mem (
    input                   i_clk,      // 1-bit
    input                   i_rst_n,    // 1-bit
    input                   i_wen,      // 1-bit
    input  [ 31 : 0 ]       i_addr,     // 32-bit
    input  [ 31 : 0 ]       i_wdata,    // 32-bit
    output [ 31 : 0 ]       o_rdata     // 32-bit
);
```

5.6 Overflow Conditions

Overflow status asserted when:

- Additive arithmetic results exceed numeric range.
- Any instruction computes an invalid memory address mapping.

5.7 Status Output

o_status	Meaning
2'd0	R-Type instruction completed successfully
2'd1	I-Type instruction completed successfully
2'd2	Overflow detected
2'd3	Program terminated (EOF)

All instruction streams end with an EOF instruction.

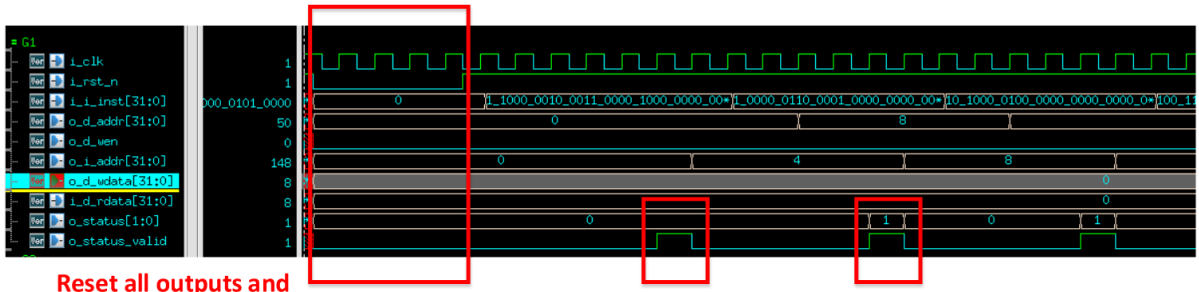
6 Testbench Instructions

You must implement:

- Clock and reset logic
- Waveform dump (e.g. vcd)
- Verification logic for status and memory

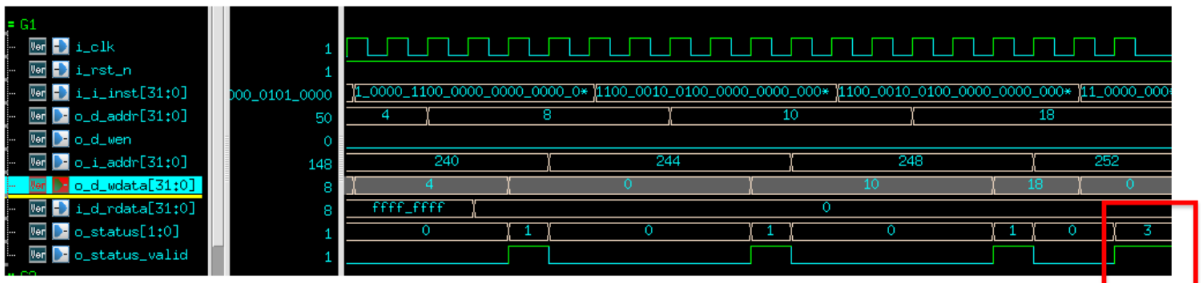
7 Sample Waveform

- Status check

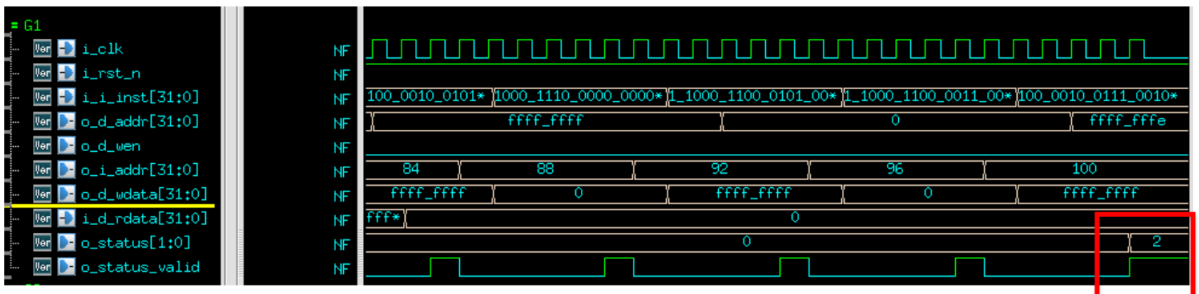


Reset all outputs and register file to 0

o_status is 0 if R-type instruction success,
o_status is 1 if I-type instruction success

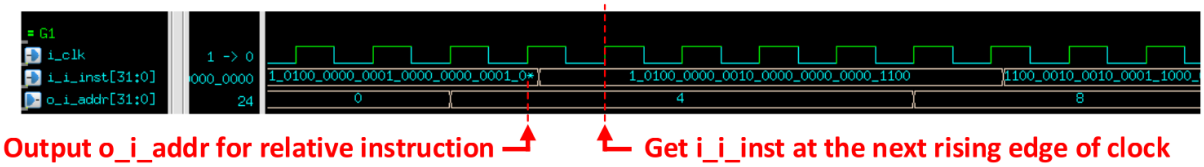


o_status is 3 if instruction is EOF

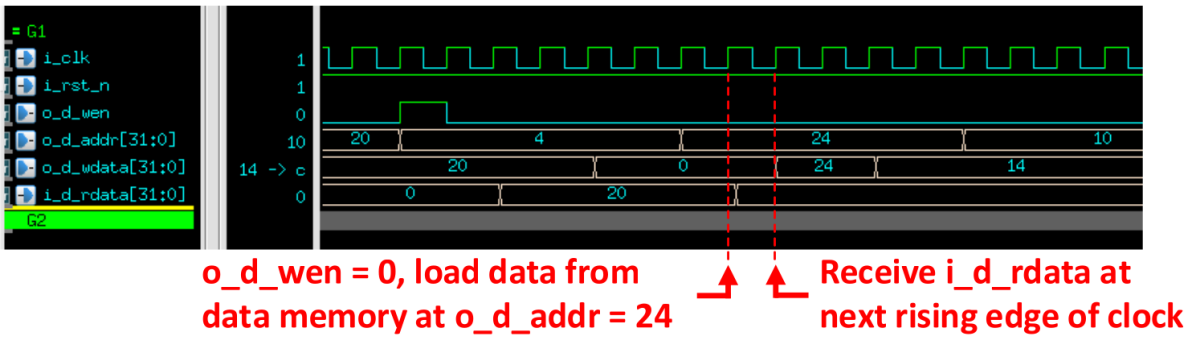


o_status is 2 if overflow occur

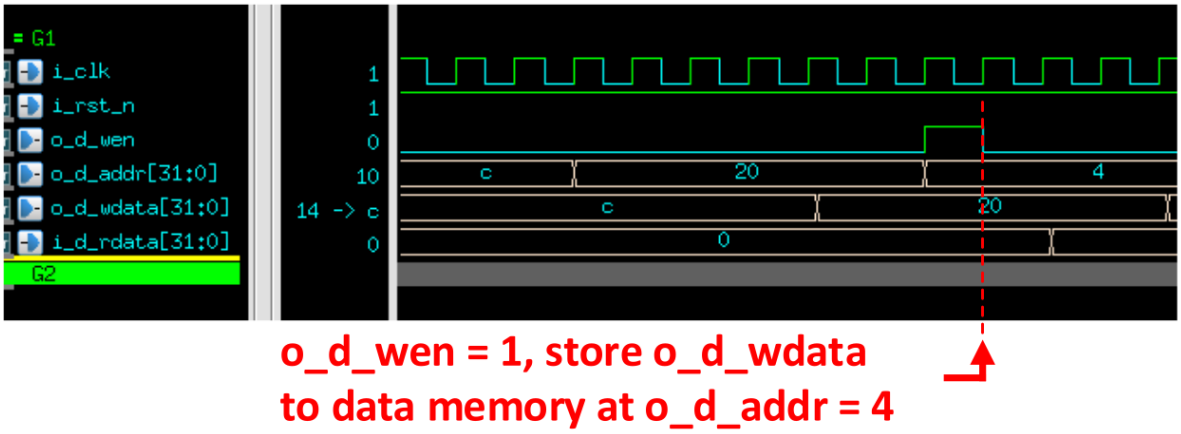
- Read instruction from instruction memory



- Load data from data memory



- Save data to data memory



Submission

1. Create folder: `Proj2_studentID` that includes all your design files (rtl.f, core.v, ..) and a report showing your work with explanations and screenshots.
2. Compress and upload it to its corresponding Dropbox on avenue

8 Grading Policy

Different score for different level design

1. Level A: achieve following request and get 90pts

   - Function work, and pass all patterns in RTL-simulation
   - Finish synthesis, and pass all patterns in gate-level simulation
   - Your performance score is less than 80000
   - Cycle time for gate-level simulation is less than 100ns

   Score for performance to be considered: Score = Power1 × Time1 + Power2 × Time2 + ... + Power7 × Time7 Unit: power(mW), Time(ns)

```
Start to Send IOT Data & Compare ...


P00:  ** Correct!! **
P01:  ** Correct!! **
P02:  ** Correct!! **
P03:  ** Correct!! **
P04:  ** Correct!! **
P05:  ** Correct!! **
P06:  ** Correct!! **
P07:  ** Correct!! **
P08:  ** Correct!! **
P09:  ** Correct!! **
P10:  ** Correct!! **
P11:  ** Correct!! **

--------------------------------------------------

Congratulations! All data have been generated successfully!

-------------------------PASS-------------------------

Simulation complete via $finish(1) at time 16590 NS + 0
./testfixture.v:227       #(`CYCLE/2); $finish;
```

```
                          Internal  Switching  Leakage    Total
Power Group               Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network             5.462e-04 1.347e-04 1.455e-06 6.824e-04 (75.88%)  i
register                  9.190e-05 4.246e-05 2.589e-05 1.603e-04 (17.82%)
combinational             1.589e-05 3.080e-05 1.001e-05 5.670e-05 ( 6.30%)
sequential                   0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                       0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                       0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                    0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 2.080e-04   (23.13%)
  Cell Internal Power  = 6.540e-04   (72.72%)
  Cell Leakage Power   = 3.736e-05   ( 4.15%)
                         ---------
Total Power            = 8.994e-04  (100.00%)

X Transition Power     = 1.331e-05
Glitching Power        = 1.149e-06

Peak Power             =    0.0857
Peak Time              = 15665.999
```

2. Level B: achieve following request and get 80pts

   - Function work, and pass all patterns in RTL-simulation
   - Finish synthesis, and pass all patterns in gate-level simulation
   - Your performance score is less than 100000
   - Cycle time for gate-level simulation is less than 100ns

3. Level C: achieve following request and get 70pts
    - Function work, and pass all patterns in RTL-simulation
    - Finish synthesis, and pass all patterns in gate-level simulation
    - Your performance score is more than 100000
    - Cycle time for gate-level simulation is less than 100ns
4. Level D: achieve following request and get 60pts
    - Function work, and pass all patterns in RTL-simulation
    - Finish synthesis, but fail at least one pattern in gate-level simulation
5. Level E: achieve following request and get 50pts
    - Fail at least one pattern in RTL-simulation
6. Report: 10pts

Design Hint

A recommended finite state machine (FSM) structure:

- Idle
- Instruction Fetch
- Instruction Decode
- Execution / Memory Access
- Write Back
- Next PC Update
- Termination