

Computer Architecture & Organization Lab Manual

1. Experiment-1: ALU Design

- 1.1. Design and implement a simple 4-bit ALU capable of processing the functions and operations described in table-1. Design is in hardware using normal logic gates, 4-bit adder and 4-bit multiplexers.

Table-1: ALU functions

| Category | Function Selection | Function |
|------------|--------------------|-------------------------|
| Arithmetic | 000 | $F = A + 1$ [Increment] |
| | 001 | $F = A + B$ [ADD] |
| | 010 | $F = A - B$ [SUB] |
| | 011 | $F = A - 1$ [Decrement] |
| Logic | 100 | $F = A \text{ AND } B$ |
| | 101 | $F = A \text{ OR } B$ |
| | 110 | $F = A \text{ XOR } B$ |
| | 111 | $F = \text{NOT } A$ |

- 1.2. Your circuit should have the following:
- 1.2.1. 4-bit DIP switches for input-A
 - 1.2.2. 4-bit DIP switches for input-B
 - 1.2.3. 3-bit DIP switches for selection lines [S2, S1, S0]
 - 1.2.4. 4-bit LEDs for output

2. Experiment-2: ALU Design

- 2.1. Design and implement the required ALU design to perform functions described in table-1 in VHDL and verify your design on the FPGA kit.

You must connect the following to the FPGA:

- 2.1.1. 8-bit DIP switches for input-A
 - 2.1.2. 8-bit DIP switches for input-B
 - 2.1.3. 3-bit DIP switches for selection lines [S2, S1, S0]
 - 2.1.4. 8-bit LEDs for output
- 2.2. You must also provide a testbench showing executing all functions of the ALU and provide a visual output of running the testbench file in simulation (using GTKwave view).

3. Experiment-3: ROM/RAM Design

3.1. Read Only Memory [ROM]:

Design and implement a simple 16-bit ROM consisting of 32 Bytes.

3.1.1. The ROM has the following

- 3.1.1.1. 4-bit address [A0 - A3]
- 3.1.1.2. 16-bit data [D0 - D15] - connected to LEDs.
- 3.1.1.3. 1-bit Chip Enable [CE]
- 3.1.1.4. 1-bit Read Enable [RE]

3.1.2. The ROM must contain the values described in table-2.

Table-2: ROM Data

| Address | Value (Bin) | Value (Hex) |
|---------|------------------|-------------|
| 0 | 1000000000000000 | 8000 |
| 1 | 1100000000000000 | C000 |
| 2 | 1110000000000000 | E000 |
| 3 | 1111000000000000 | F000 |
| 4 | 1111100000000000 | F800 |
| 5 | 1111110000000000 | FC00 |
| 6 | 1111111000000000 | FE00 |
| 7 | 1111111100000000 | FF00 |
| 8 | 1111111110000000 | FF80 |
| 9 | 1111111111000000 | FFC0 |
| 10 | 1111111111100000 | FFE0 |
| 11 | 1111111111110000 | FFF0 |
| 12 | 1111111111111000 | FFF8 |
| 13 | 1111111111111100 | FFFC |
| 14 | 1111111111111110 | FFFE |
| 15 | 1111111111111111 | FFFF |

3.2. Random Access Memory [RAM]:

Design and implement a simple 16-bit RAM consisting of 32 Bytes

3.2.1. The RAM has the following

- 3.2.1.1. 4-bit address [A0 - A3]
- 3.2.1.2. 16-bit data [D0 - D15] - input - connected to DIP switch.
- 3.2.1.3. 16-bit data [D0 - D15] - output - connected to LEDs.
- 3.2.1.4. 1-bit Chip Enable [CE]
- 3.2.1.5. 1-bit Read Enable [RE]
- 3.2.1.6. 1-bit Write Enable [WE]

4. Experiment-4: Control Unit Design

- 4.1. Design and implement the control unit required for the MIPS processor as shown in block diagram in figure-1. The instruction set is provided in table-3 and table-4.

The control unit is supposed to have the instruction, \$rs and \$rt 8-bit values, immediate data and address values as inputs. And outputs destination register address, ALU control code, memory read or memory write signals, address value, PC update value and FR's (Flag Register) flags values; based on the instruction being executed.

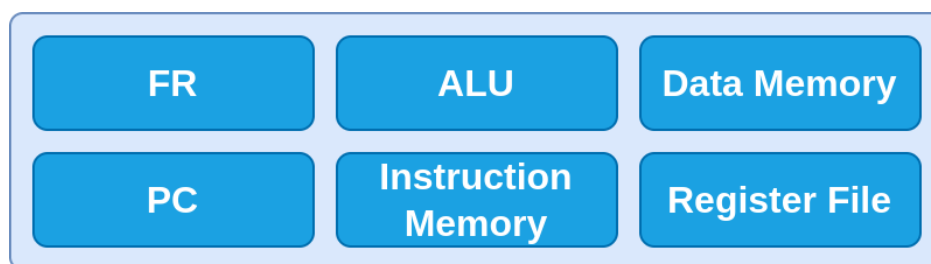


Figure-1: MIPS Processor Block Diagram

- 4.2. You are required to provide a testbench to verify your implementation, as following:
- 4.2.1. You should provide the instruction to execute.
 - 4.2.2. You should provide values for rs and rt, whenever needed.
 - 4.2.3. You should provide values for immediate data, whenever needed.
 - 4.2.4. You should provide values for jump addresses, whenever needed.
 - 4.2.5. You should provide various instructions to test different opcodes and functions of the instruction set.
- 4.3. **Output of the control unit will be verified using the testbench and visualized on GTKwave.**

Table-3: MIPS CPU types of instructions

| Type | Sub-Type | Fields | | | | | Comment |
|--------|----------|--------|----------------|---------------------|---------------------|----------|------------------------------------|
| R-Type | RRR | 15-12 | 11-9 | 8-6 | 5-3 | 2-0 | Arithmetic Instruction Format |
| | | 4 | 3 | 3 | 3 | 3 | |
| | | opcode | rs | rt | rd | Function | |
| I-Type | RRI | 15-12 | 11-9 | 8-6 | 5-0 | | Transfer, Branch, Immediate Format |
| | | 4 | 3 | 3 | 6 | | |
| | | opcode | rs | rt | Address / Immediate | | |
| | RI | 15-12 | 11-9 | 8-0 | | | |
| | | 4 | 3 | 9 | | | |
| | | opcode | rs | Address / immediate | | | |
| J-Type | J | 15-12 | 11-0 | | | | Jump Instruction Format |
| | | 4 | 12 | | | | |
| | | opcode | Target Address | | | | |

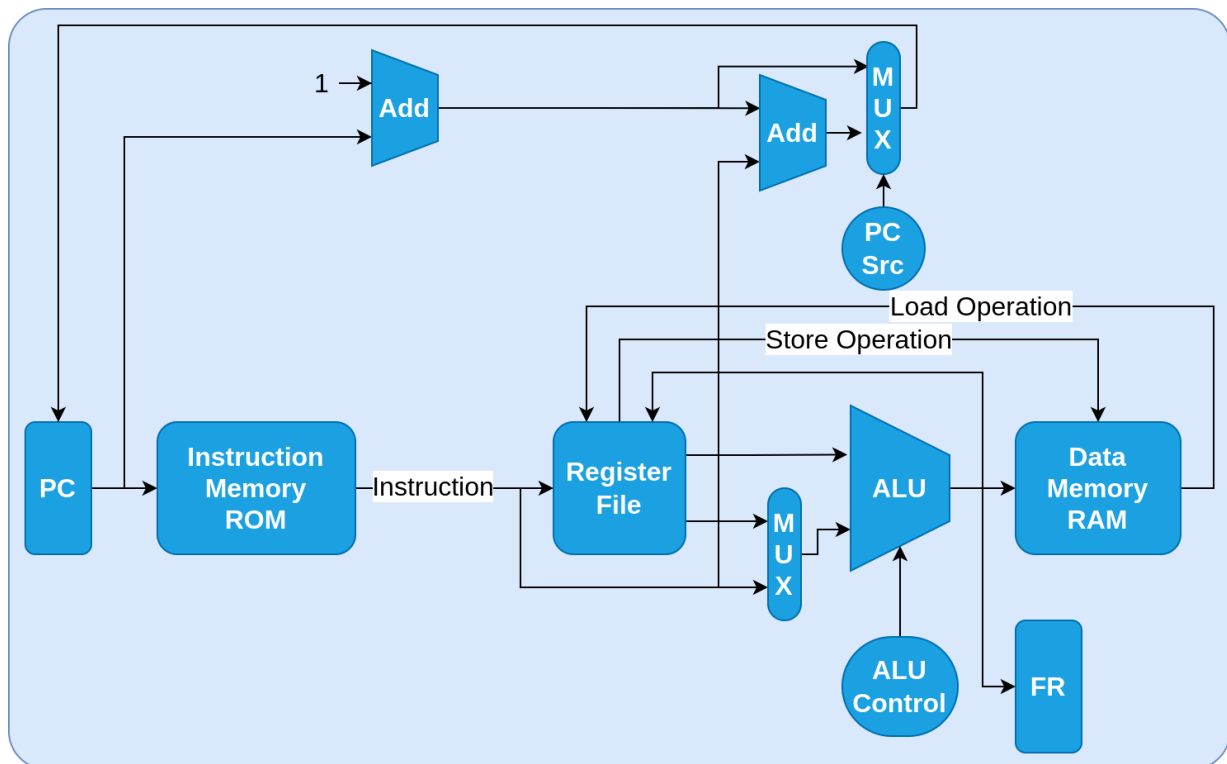


Figure-2: MIPS Processor Data Path

Table-4: MIPS CPU instruction set

| Mne- monic | Assembly Format | Fields | | | | | Description | | Type |
|---------------|----------------------|-------------|---------------|--------------------------|-----------------------------|----------------|---|------------------|------|
| - | - | opcode 4 | rs 3 | rt 3 | rd 3 | function 3 | Arithmetic | | R |
| add | ADD \$rs, \$rt, \$rd | 0001 | xxx | xxx | xxx | 000 | \$rd ← \$rs + \$rt | addition | RRR |
| sub | SUB \$rs, \$rt, \$rd | 0001 | xxx | xxx | xxx | 001 | \$rd ← \$rs - \$rt | subtraction | |
| mul | MUL \$rs, \$rt, \$rd | 0001 | xxx | xxx | xxx | 010 | \$rd ← \$rs * \$rt | Multiplication | |
| div | DIV \$rs, \$rt, \$rd | 0001 | xxx | xxx | xxx | 011 | \$rd ← \$rs / \$rt | Division | |
| and | AND \$rs, \$rt, \$rd | 0001 | xxx | xxx | xxx | 100 | \$d ← \$s AND \$t | and | |
| or | OR \$rs, \$rt, \$rd | 0001 | xxx | xxx | xxx | 101 | \$d ← \$s OR \$t | or | |
| xor | XOR \$rs, \$rt, \$rd | 0001 | xxx | xxx | xxx | 110 | \$d ← \$s XOR \$t | xor | |
| cpl | CPL \$rs, \$rt, \$rd | 0001 | xxx | xxx | xxx | 111 | \$d ← NOT \$s | complement | |
| - | - | opcode 4 | rs 3 | rt 3 | address / immediate 6 | | Immediate | | I |
| addi | ADD \$rs, \$rt, imm | 0010 | xxx | xxx | imm | | \$rt ← \$rs + imm | addition | RRI |
| subi | SUB \$rs, \$rt, imm | 0011 | xxx | xxx | imm | | \$rt ← \$rs - imm | subtraction | |
| andi | AND \$rs, \$rt, imm | 0100 | xxx | xxx | imm | | \$rt ← \$rs AND imm | and | |
| ori | OR \$rs, \$rt, imm | 0101 | xxx | xxx | imm | | \$rt ← \$rs OR imm | or | |
| xori | XOR \$rs, \$rt, imm | 0110 | xxx | xxx | imm | | \$rt ← \$rs XOR imm | xor | |
| - | - | opcode 4 | rs 3 | address / immediate 9 | | Immediate | | | I |
| li | LI \$rs, imm | 0111 | xxx | imm | | \$rs ← imm | | Load immediate | RI |
| lm | LM \$rs, addr | 1000 | xxx | addr | | \$rs ← M[addr] | | Load from memory | |
| sm | SM \$s, addr | 1001 | xxx | addr | | M[addr] ← \$rs | | Store to memory | |
| - | - | opcode 4 | address 12 | | | Jump Type | | | J |
| beq | BEQ \$rs, \$rt | 1010 | xxx | xxx | addr | | if(\$rs == \$rt) PC ← Addr else PC ← PC + 1 | Branch if equal | |

| Mne- monic | Assembly Format | Fields | | | | Description | | Type |
|---------------|-----------------|--------|-------|-----|------|--|------------------------|-------|
| bgt | BGT \$rs, \$rt | 1011 | xxx | xxx | addr | if(\$rs > \$rt) PC ← Addr else PC ← PC + 1 | Branch if greater than | |
| blt | BLT \$rs, \$rt | 1100 | xxx | xxx | addr | if(\$rs < \$rt) PC ← Addr else PC ← PC + 1 | Branch if less than | |
| bc | BC \$rs, \$rt | 1101 | xxx | xxx | addr | if(CF == 1) PC ← Addr else PC ← PC + 1 | Branch on carry | |
| bz | BZ \$rs, \$rt | 1110 | xxx | xxx | addr | if(ZF == 1) PC ← Addr else PC ← PC + 1 | Branch on zero | |
| br | BR addr_imm | 1111 | addr | | | PC ← Addr | Unconditional Branch | |
| nop | NOP | 0000 | ----- | | | No operation | No operation | other |

5. Experiment-5: MIPS based 8-bit RISC Processor

Use the components implemented in previous experiments to build a basic 8-bit processor consisting of an ALU, Register file, Instruction memory, data memory, PC and Flag register, as described in Figure-1. The data path is explained in Figure-2.

5.1. Processor elements are described as following:

5.1.1. Program counter (PC):

It is an 8-bit register. Its output (PC) represents the address of the current instruction (instr) to be executed while its input (PC next) represents the address of the next instruction.

5.1.2. Instruction memory: 16-bit ROM of 256 locations

It takes an 8-bit address from the PC register and reads a 16-bit data at the output port.

5.1.3. Register file:

Consists of 8 registers each of 16-bit in size. It has two reading ports (RD1 and RD2) and one writing port (WD3). Reading ports (RD1 and RD2) take 3-bit address inputs (A2, A1 and A0) which in turn select one of the 8 registers to be read on the reading output ports (RD1 and RD2). Writing port (WD3) takes 5-bit address input (A3) which in turn selects one of the 8 registers to which the 8-bit at (WD3) input port will be written if the WE signal is 1 on the rising edge of the clock signal.

5.1.4. Data memory:

has one output reading port (RD) and one input writing port (WR). 16-bit data at input (WR) port is written to the memory location specified by the address (A) if (WE) signal is 1 at the raising edge of the clock signal. The content of memory location selected by (A) input is always available at (RD) output port.

5.1.5. ALU (Arithmetic Logic Unit):

It has been designed in order to execute all the arithmetic/logical instructions. ALU takes ALU control (2:0) as inputs and generates the ALU functions according to it. Table-4 provides functions that can be executed by the ALU.

5.1.6. Flag register [FR] or Status register [SR]: 8-bit register

An 8-bit register having:

- 1-bit as Zero flag [ZF], set to 1 when the result is zero and 0 otherwise.
- 1-bit as Carry flag [CF], set to 1 when there's a carry and 0 otherwise.

- c. 1-bit as Sign flag [SF], set to 1 when result is negative and 0 otherwise.

- d. Flag Register can be viewed as following:

| | | | | | | | |
|---|---|---|---|---|----|----|----|
| - | - | - | - | - | SF | CF | ZF |
|---|---|---|---|---|----|----|----|

- 5.2. You are required to fully build the MIPS processor.
- 5.3. You are required to write a program, convert it to machine language based on the provided instruction set and store it in the instruction memory.
- 5.4. **Experiment output will be verified using the testbench and visualized on GTKwave.**
- 5.5. **Experiment output will be verified in real-time on the FPGA kit.**