Specification for Design Under Test (DUT) for Laboratory Exercises

1. INTRODUCTION

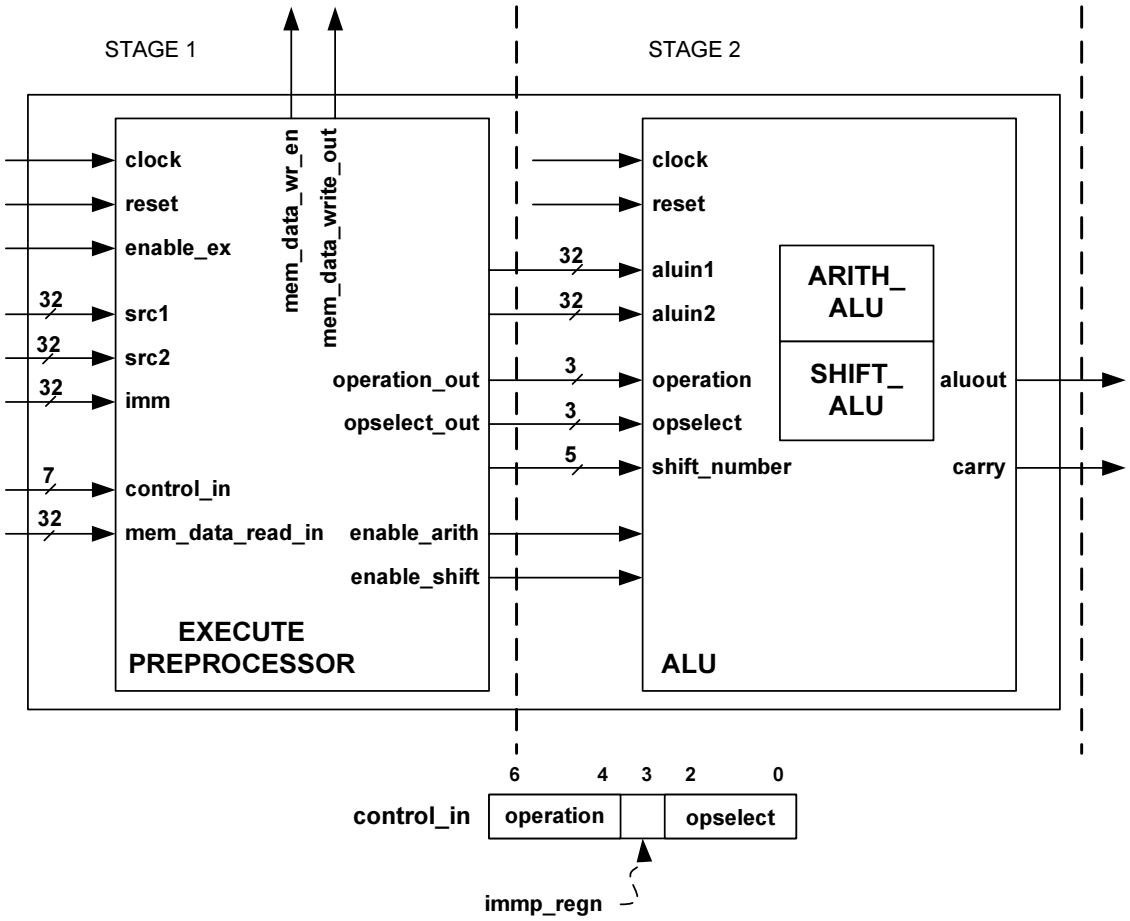


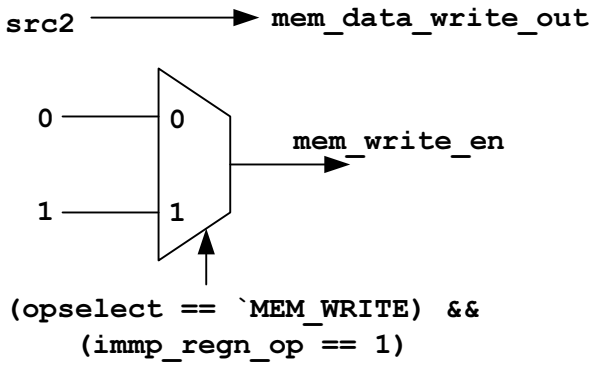
Figure 1: Top Level Block diagram of system and description of control_in

The DUT consists of a two stages pipeline that consists of an Execute pre-processor and an ALU based on a modification to the DLX ISA. The preprocessor performs the function of the preparation of the design inputs and controls for the ALU using the information seen at the input. The ALU then executes the relevant operation on the incoming data controlled by the `opselect, operation` and `shift_number` values. Each unit is independently clocked and reset. Almost all valid instructions under consideration pass through both the stages except for memory write instructions which deal with the `mem_data_wr_en` and `mem_data_write_out` signal and completely skip the ALU stage. Thus, when enabled the first stage creates the `aluin1, aluin2, operation, opselect, shift_number,`

enable_arith and enable_shift values. These values are then used by the ALU stage to create the relevant aluout and carry signals.

2 SYSTEM SPECIFICATION

2.1 Combinational Behavior of outputs:



2.2 Reset Behavior:

All sequential outputs go to 0 i.e. aluin1, aluin2, operation_out, opselect_out, shift_number, enable_arith and enable_shift for Stage 1 and aluout and carry for stage 2 go to 0.

2.3 Normal Behavior of EXECUTE PREPROCESSOR BLOCK:

All sequential outputs change only when enable_ex ==1 . Therefore, assuming enable_ex == 1, the following truth tables hold for each sequential output:

aluin1

enable_ex	aluin1
0	No change (NC)
1	src1

aluin2

enable_ex	opselect	control_in[3]	aluin2
0	X	X	NO CHANGE
1	ARITH_LOGIC	0	src2
1	ARITH_LOGIC	1	imm
1	MEM_READ	0	NO CHANGE
1	MEM_READ	1	mem_data_read_in
FOR ALL OTHER COMBINATIONS aluin2 REMAINS UNCHANGED			

operation_out

enable_ex	operation_out
0	No change (NC)
1	control_in[6:4]

opselect_out

enable_ex	opselect_out
0	No change (NC)
1	control_in[2:0]

shift_number

enable_ex	opselect	imm[2]	shift_number
0	X	X	0
1	SHIFT_REG	0	imm[10:6]
1	SHIFT_REG	1	src2[4:0]
FOR ALL OTHER COMBINATIONS shift_number IS 0			

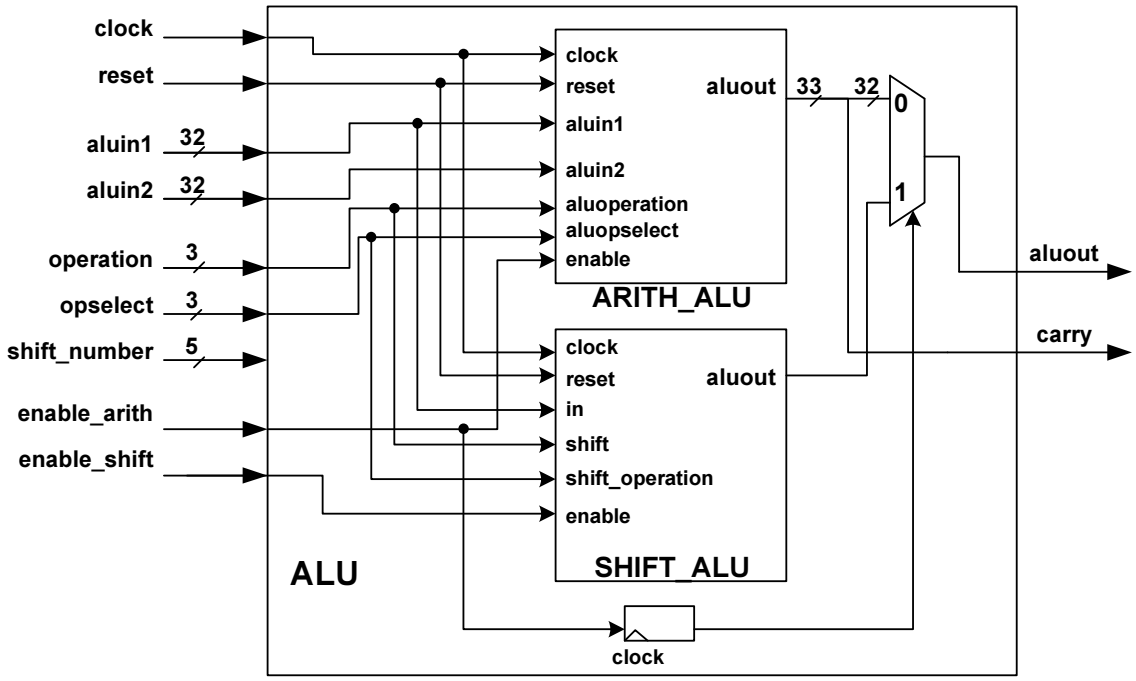
enable_arith

enable_ex	opselect	control_in[3]	enable_arith
0	X	X	0
1	ARITH_LOGIC	0	1
1	ARITH_LOGIC	1	1
1	MEM_READ	0	0
1	MEM_READ	1	1
FOR ALL OTHER COMBINATIONS enable_arith IS 0			

enable_shift

enable_ex	opselect	enable_shift
0	X	0
1	SHIFT_REG	1
FOR ALL OTHER COMBINATIONS shift_number IS 0		

2.4 Normal Behavior of ALU BLOCK



All the commands are valid only when the relevant enables i.e. `enable_arith == 1` or `enable_shift == 1` and the entire system is sensitive to the positive edge of the clock. Assuming, that `enable_execute == 1`, the following truth tables hold good for each ALU. **Please note that the in port of the Shift ALU is connected to aluin1.**

SHIFT_ALU

enable	shift_operation[1:0]	aluout
0	X	NO CHANGE
1	SHLEFTLOG	in << shift;0 padded lower bits
1	SHLEFTART	in << shift; 0 padded lower bits
1	SHRGHTLOG	in >> shift; 0 padded upper bits
1	SHRGHTART	in >> shift; sign extended upper bits
FOR ALL OTHER COMBINATIONS aluout REMAINS UNCHANGED		

ARITH_ALU

enable	aluopselect	operation	aluout = {carry, aluout[31:0}}
0	X	X	UNCHANGED
1	ARITH_LOGIC	ADD	Signed aluin1 **(+)** aluin2
1	ARITH_LOGIC	HADD	{h_carry,h_add[15:0]} = aluin1[15:0] **(+)** aluin2[15:0] carry = h_carry; aluout = ***sxt***(h_add)

1	ARITH_LOGIC	SUB	Signed aluin1 **(-)** aluin2
1	ARITH_LOGIC	NOT	(**BITWISE NOT**) aluin2; carry = 0;
1	ARITH_LOGIC	AND	aluin1 (**BITWISE AND**) aluin2
1	ARITH_LOGIC	OR	aluin1 (**BITWISE OR**) aluin2
1	ARITH_LOGIC	XOR	aluin1 (**BITWISE XOR**) aluin2
1	ARITH_LOGIC	LHG	aluout[31:16] = aluin2[15:0]; aluout[15:0] = 16'h0; carry = 0;
1	MEM_READ	LOADBYTE	***signext***{aluin2[7:0]}; carry = 0;
1	MEM_READ	LOADBYTEU	***zeropad***{aluin2[7:0]}; carry = 0;
1	MEM_READ	LOADHALF	***signext***{aluin2[15:0]}; carry = 0;
1	MEM_READ	LOADHALFU	***zeropad***{aluin2[15:0]}; carry = 0;
1	MEM_READ	LOADWORD	aluin2; carry = 0;
1	MEM_READ	***others***	aluin2; carry = 0;
FOR ALL OTHER COMBINATIONS aluout and carry REMAIN UNCHANGED			

2.5 Additional information:

2.5.1 Understanding `signext{}`: (Note that M > N)

`out[M:0] = signext{in[N:0]}` will perform: `out[N:0] = in[N:0]` and `out[M:N+1]` repeats `in[N]`.

2.5.2 Understanding `zeropad{}`:(Note that M > N)

`out[M:0] = zeropad{in[N:0]}` will perform: `out[N:0] = in[N:0]` and `out[M:N+1]` repeats 0.

2.5.3 Carry Values

1. The carry out from the Execute unit functions per arithmetic correctness for Subtraction and Addition.
2. For the following operations the **carry = 0**:
    a. Arithmetic: **`AND, OR, XOR, LHG`**
    b. All **`MEM_READ`** operations
    c. Shift operations : **`SHLEFTLOG, SHRGHTLOG, SHRGHTART`**
3. For **`SHLEFTART`** (Shift Left Arithmetic),
    a. if you start off with a negative number i.e. `in[31] ==1`, and if you do a `ARITHMETIC` left shift on this number (say `in <SHLEFTART> 4`) then `carry =1`.
    b. If you start off with a positive number i.e. `in[31] == 0`, then the `carry` = 0.

The aim is to preserve the initial sign value of the value being shifted so that a check can be made to see if the sign of the resulting number has changed.

Appendix 1
DEFINITIONS

```
`define     CLK_PERIOD          10

`define     REGISTER_WIDTH      32
`define     INSTR_WIDTH         32
`define     IMMEDIATE_WIDTH     16

`define     MEM_READ            3'b101
`define     MEM_WRITE           3'b100
`define     ARITH_LOGIC         3'b001
`define     SHIFT_REG           3'b000

// ARITHMETIC
`define     ADD                 3'b000
`define     HADD                3'b001
`define     SUB                 3'b010
`define     NOT                 3'b011
`define     AND                 3'b100
`define     OR                  3'b101
`define     XOR                 3'b110
`define     LHG                 3'b111

// SHIFTING
`define     SHLEFTLOG           3'b000
`define     SHLEFTART           3'b001
`define     SHRGHTLOG           3'b010
`define     SHRGHTART           3'b011

// DATA TRANSFER
`define     LOADBYTE            3'b000
`define     LOADBYTEU           3'b100
`define     LOADHALF            3'b001
`define     LOADHALFU           3'b101
`define     LOADWORD            3'b011
```