Southern Luzon State University
College of Engineering
Lucban, Quezon

CPE23L- COMPUTER ARCHITECTURE & ORGANIZATION

Final Project Documentation: Water Refilling Box

Submitted by:

Bolo, Lyza April P.
Carreos, Clarice Mae G.
Corpuz, Reimarc G.

BSCPE IV - IE

Submitted to:
Engr. Carla May C. Ceribo

Submitted on:
January 11, 2024

INTRODUCTION

Verilog, as a hardware description language (HDL), plays a crucial role in designing and simulating digital circuits. In building a Central Processing Unit (CPU) using Verilog, registers, including instruction registers (IR), play a key role in storing and managing data during instruction execution. This Verilog code uses registers strategically to make the CPU work better by improving how the data moves around and stays organized. The instruction register specifically holds the binary representation of the current instruction fetched from memory. Its primary function is to facilitate the decoding of instructions, determining the operation the CPU needs to perform.

The Verilog code outlines the CPU architecture, defining the interconnections of components like Arithmetic and Logical Units (ALUs), multiplexers, and control units. Registers act as a temporary storage for different types of data movements, ensuring smooth instruction execution. The register-transfer level (RTL) design paradigm is applied to describe various data movements and control flow, connecting high-level functionality with low-level hardware implementation.

Clock and reset signals are incorporated to synchronize register and CPU component operations. This Verilog code forms the basis for simulating, testing, and compiling a fully operational CPU, highlighting Verilog's effectiveness in digital circuit design.

METHODOLOGY & DISCUSSION

Verilog module PROJECT contains a case statement with only one case (water). This case appears to simulate a water refilling operation based on the values of certain control signals. The mul_res variable is used to store the result of a multiplication operation, and the code includes a loop that prints decreasing values of j. Variable j is the time duration of refilling the water as it stops at 0 second.

Control Signals:

- *oper_type: Identifies the type of operation.*
- *rdst, rsrc1, rsrc2: Register addresses for source and destination operands.*
- *imm_mode: Indicates whether immediate value (isrc) is used in the operation.*
- *isrc: Immediate value for certain operations.*

Initialization:

- *The GPR array is initialized with certain values representing the number of seconds for each peso or the result of the multiplication operation from 15 bit and below.*
- *The SGPR variable is used to store a portion of the result of the multiplication operation from 16 bit and above.*

Water Refilling Operation (water case) mul operation:

- *The operation is triggered based on the control signals in the IR register.*
- *Depending on the value of imm_mode, either a multiplication of two GPR values or a GPR value and an immediate value (isrc) is performed.*
- *The result (mul_res) is then split into two parts, and the lower 16 bits are stored in the destination GPR, while the upper 16 bits are stored in SGPR.*
- *The loop attempts to print decreasing values starting from mul_res, but there are issues with the loop structure.*

Testbench (PROJECT_tb):

- *The testbench initializes the GPR array with values for pesos.*
- *Two water refilling operations are simulated with different inputs.*
- *The simulation includes displaying information about the operation and the time duration.*

The execution example provided states that if 3 pesos are inserted, and 1 peso is equivalent to 2 seconds, the time duration is 6 seconds. The desired output is a countdown sequence: 6, 5, 4, 3, 2, 1, 0.

VERILOG CODE

```
 `timescale 1ns / 1ps

`define oper_type IR[31:27]
`define rdst      IR[26:22]
`define rsrc1     IR[21:17]
`define imm_mode  IR[16]
`define rsrc2     IR[15:11]
`define isrc      IR[15:0]


`define water          5'b00101

module PROJECT();

  reg [31:0] IR;
  reg [15:0] GPR [31:0];
  reg [15:0] SGPR;
  reg [31:0] mul_res;

  integer j;

  always @*
  begin
    case (`oper_type)

      `water:
        begin
          if (`imm_mode)
            mul_res = GPR[`rsrc1] * `isrc;
          else
            mul_res = GPR[`rsrc1] * GPR[`rsrc2];

          GPR[`rdst] = mul_res[15:0];
          SGPR = mul_res[31:16];

          for (j = mul_res; j >= 0; j = j - 1)
          begin
           $display("%d", j);
          end
        end

    endcase
  end

endmodule
```

TESTBENCH

```
`timescale 1ns / 1ps

`define oper_type IR[31:27]
`define rdst      IR[26:22]
`define rsrc1     IR[21:17]
`define imm_mode  IR[16]
`define rsrc2     IR[15:11]
`define isrc      IR[15:0]

`define water          5'b00101

module PROJECT_tb;

  PROJECT dut();

  initial begin
    dut.GPR[1] = 2;
    dut.GPR[2] = 3;
    dut.GPR[3] = 4;
  end

  initial
  begin

    $display("--------------------------------------------------FIRST CUSTOMER");
    $display("----------------------------------------------------------------");
    dut.IR = 0;
    dut.`imm_mode = 0;
    dut.`oper_type = 5;
    dut.`rsrc1 = 1;
    dut.`rsrc2 = 2;
    dut.`rdst = 0;
    #10;
    $display("WATER REFILLING BOX");
    $display("1 pesos =  %0d seconds", dut.GPR[1]);
    $display("INSERTED: %0d pesos", dut.GPR[2]);
    $display("TIME DURATION: %0d seconds", dut.GPR[0]);
    $display("----------------------------------------------------------------");

    $display("-------------------------------------------------SECOND CUSTOMER");
    $display("----------------------------------------------------------------");
    dut.IR = 0;
    dut.`imm_mode = 0;
    dut.`oper_type = 5;
    dut.`rsrc1 = 1;
    dut.`rsrc2 = 3;
    dut.`rdst = 0;
```

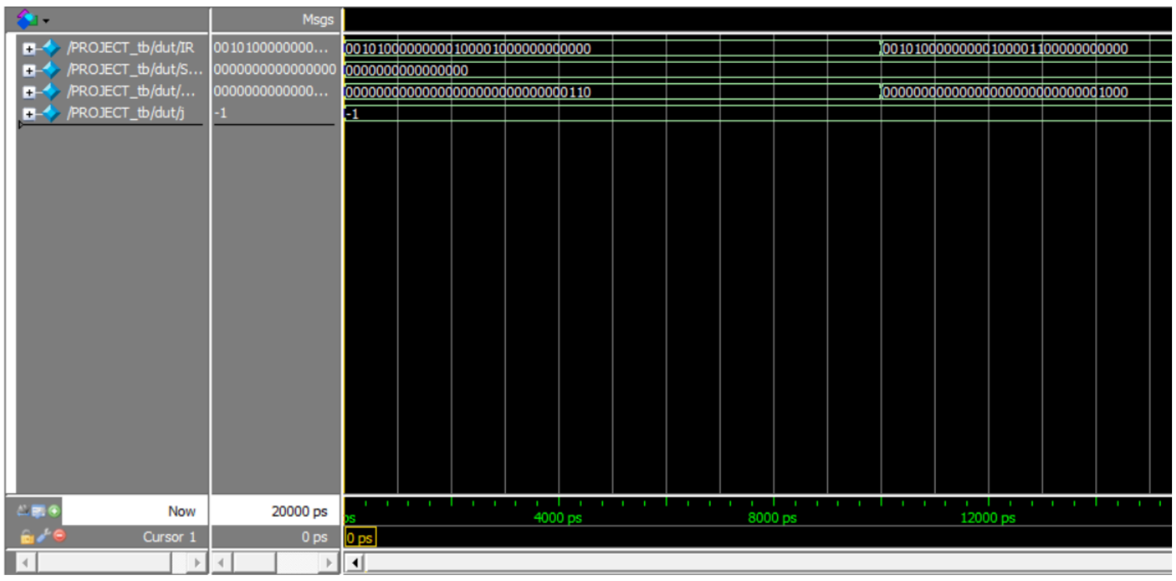
```
        #10;
        $display("WATER REFILLING BOX");
        $display("1 pesos =  %0d seconds", dut.GPR[1]);
        $display("INSERTED: %0d pesos", dut.GPR[3]);
        $display("TIME DURATION: %0d seconds", dut.GPR[0]);
        $display("-----------------------------------------------------------------");

    end

endmodule
```

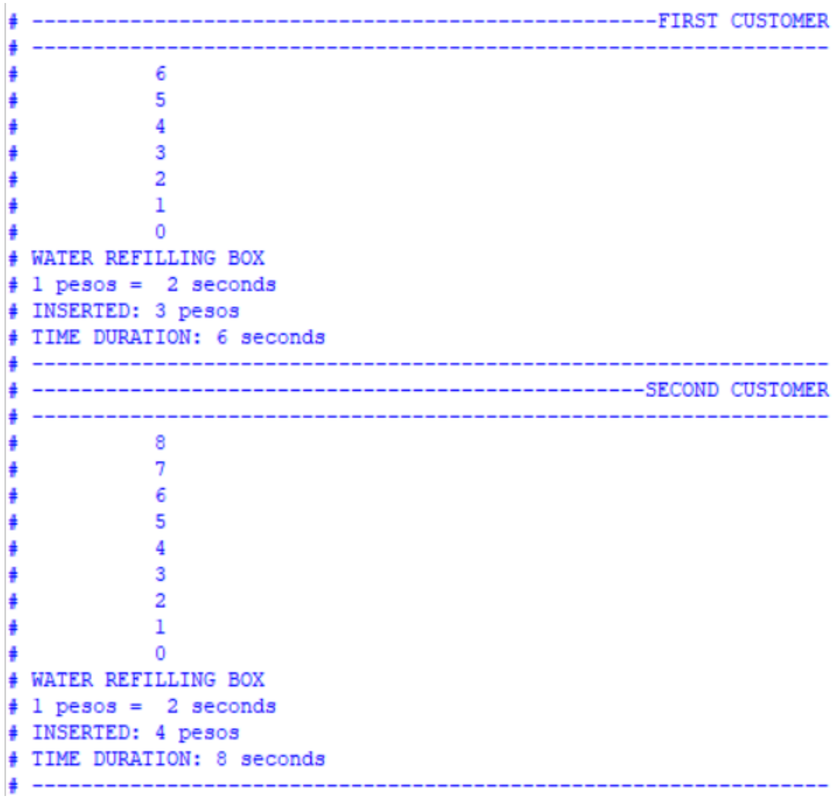
VERILOG DISPLAY OUTPUTS

a. WAVEFORM



```
/PROJECT_tb/dut/IR    0010100000000...   0010100000000100001000000000000      0010100000000100001100000000000
/PROJECT_tb/dut/S...  0000000000000000   0000000000000000
/PROJECT_tb/dut/...   0000000000000...   00000000000000000000000000000110     00000000000000000000000000001000
/PROJECT_tb/dut/j     -1                 -1
Now                   20000 ps           4000 ps   8000 ps   12000 ps
Cursor 1              0 ps               0 ps
```

b. TRANSCRIPT

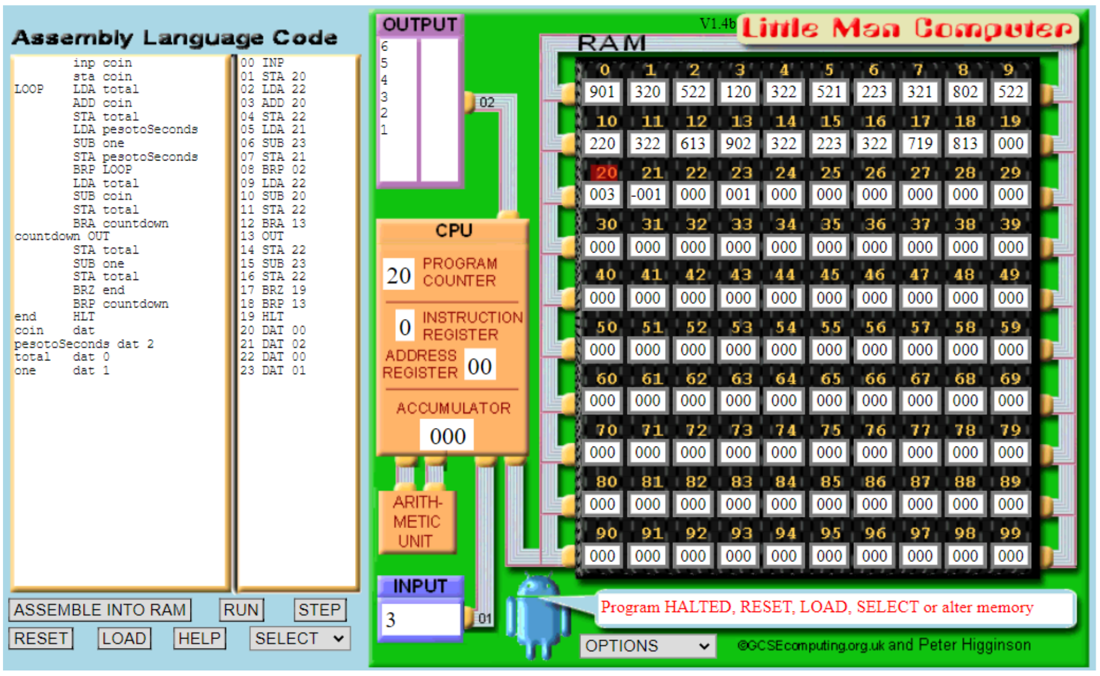


```
# ---------------------------------------------------FIRST CUSTOMER
# -----------------------------------------------------------------
#          6
#          5
#          4
#          3
#          2
#          1
#          0
# WATER REFILLING BOX
# 1 pesos =  2 seconds
# INSERTED: 3 pesos
# TIME DURATION: 6 seconds
# -----------------------------------------------------------------
# --------------------------------------------------SECOND CUSTOMER
# -----------------------------------------------------------------
#          8
#          7
#          6
#          5
#          4
#          3
#          2
#          1
#          0
# WATER REFILLING BOX
# 1 pesos =  2 seconds
# INSERTED: 4 pesos
# TIME DURATION: 8 seconds
# -----------------------------------------------------------------
```

ASSEMBLY LANGUAGE CODE

```
 inp coin
        sta coin
LOOP    LDA total
        ADD coin
        STA total
        LDA pesotoSeconds
        SUB one
        STA pesotoSeconds
        BRP LOOP
        LDA total
        SUB coin
        STA total
        BRA countdown
countdown OUT
        STA total
        SUB one
        STA total
        BRP countdown
        HLT
coin    dat
pesotoSeconds dat 2
total   dat 0
one     dat 1
```

LMC DISPLAY OUTPUT



Assembly Language Code

```
        inp coin
        sta coin
LOOP    LDA total
        ADD coin
        STA total
        LDA pesotoSeconds
        SUB one
        STA pesotoSeconds
        BRP LOOP
        LDA total
        SUB coin
        STA total
        BRA countdown
countdown OUT
        STA total
        SUB one
        STA total
        BRZ end
        BRP countdown
end     HLT
coin    dat
pesotoSeconds dat 2
total   dat 0
one     dat 1
```

```
00 INP
01 STA 20
02 LDA 22
03 ADD 20
04 STA 22
05 LDA 21
06 SUB 23
07 STA 21
08 BRP 02
09 LDA 22
10 SUB 20
11 STA 22
12 BRA 13
13 OUT
14 STA 22
15 SUB 23
16 STA 22
17 BRZ 19
18 BRP 13
19 HLT
20 DAT 00
21 DAT 02
22 DAT 00
23 DAT 01
```

OUTPUT

```
6
5
4
3
2
1
```

V1.4b Little Man Computer

RAM

	0	1	2	3	4	5	6	7	8	9
0	901	320	522	120	322	521	223	321	802	522
10	220	322	613	902	322	223	322	719	813	000
20	003	-001	000	001	000	000	000	000	000	000
30	000	000	000	000	000	000	000	000	000	000
40	000	000	000	000	000	000	000	000	000	000
50	000	000	000	000	000	000	000	000	000	000
60	000	000	000	000	000	000	000	000	000	000
70	000	000	000	000	000	000	000	000	000	000
80	000	000	000	000	000	000	000	000	000	000
90	000	000	000	000	000	000	000	000	000	000

CPU

PROGRAM COUNTER: 20

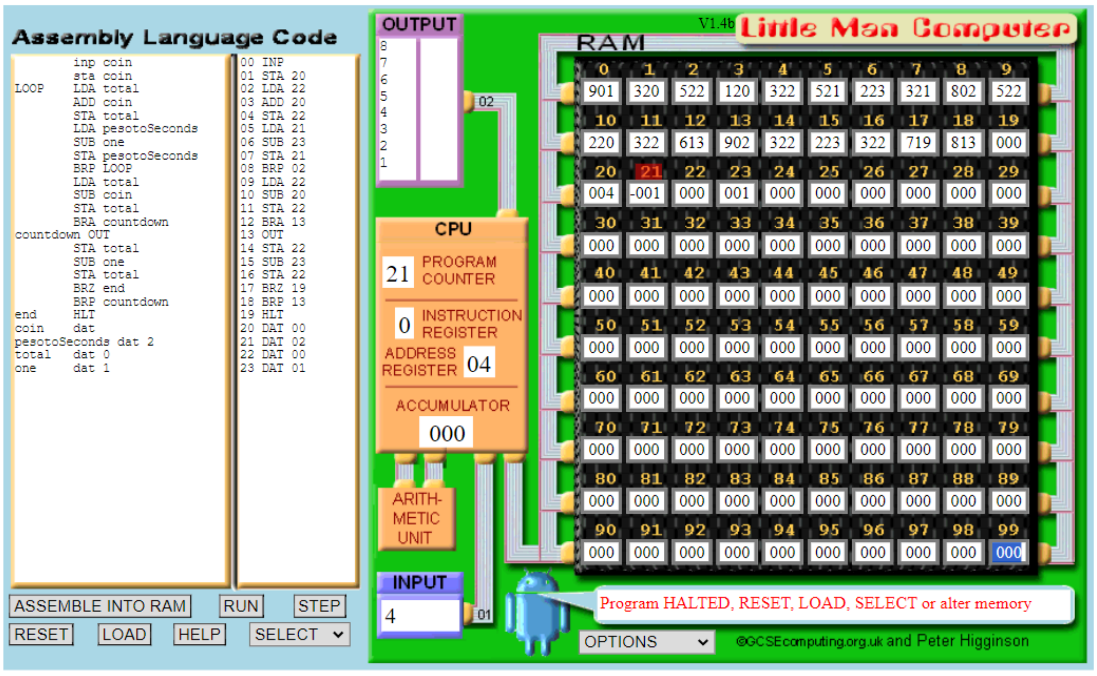
INSTRUCTION REGISTER: 0

ADDRESS REGISTER: 00

ACCUMULATOR: 000

INPUT: 3

Program HALTED, RESET, LOAD, SELECT or alter memory



```
        inp coin
        sta coin
LOOP    LDA total
        ADD coin
        STA total
        LDA pesotoSeconds
        SUB one
        STA pesotoSeconds
        BRP LOOP
        LDA total
        SUB coin
        STA total
        BRA countdown
countdown OUT
        STA total
        SUB one
        STA total
        BRZ end
        BRP countdown
end     HLT
coin    dat
pesotoSeconds dat 2
total   dat 0
one     dat 1
```

```
00 INP
01 STA 20
02 LDA 22
03 ADD 20
04 STA 22
05 LDA 21
06 SUB 23
07 STA 21
08 BRP 02
09 LDA 22
10 SUB 20
11 STA 22
12 BRA 13
13 OUT
14 STA 22
15 SUB 23
16 STA 22
17 BRZ 19
18 BRP 13
19 HLT
20 DAT 00
21 DAT 02
22 DAT 00
23 DAT 01
```

OUTPUT
```
8
7
6
5
4
3
2
1
```

PROGRAM COUNTER 21
INSTRUCTION REGISTER 0
ADDRESS REGISTER 04
ACCUMULATOR 000
INPUT 4

Program HALTED, RESET, LOAD, SELECT or alter memory

```
  0   1   2   3   4   5   6   7   8   9
901 320 522 120 322 521 223 321 802 522
 10  11  12  13  14  15  16  17  18  19
220 322 613 902 322 223 322 719 813 000
 20  21  22  23  24  25  26  27  28  29
004 -001 000 001 000 000 000 000 000 000
 30  31  32  33  34  35  36  37  38  39
000 000 000 000 000 000 000 000 000 000
 40  41  42  43  44  45  46  47  48  49
000 000 000 000 000 000 000 000 000 000
 50  51  52  53  54  55  56  57  58  59
000 000 000 000 000 000 000 000 000 000
 60  61  62  63  64  65  66  67  68  69
000 000 000 000 000 000 000 000 000 000
 70  71  72  73  74  75  76  77  78  79
000 000 000 000 000 000 000 000 000 000
 80  81  82  83  84  85  86  87  88  89
000 000 000 000 000 000 000 000 000 000
 90  91  92  93  94  95  96  97  98  99
000 000 000 000 000 000 000 000 000 000
```

CONCLUSION

This project helped us develop a strong understanding of how key registers, such as the instruction register and general-purpose registers (GPRs), function within a CPU. The instruction register (IR) holds 32 binary bits and stores the current instruction, while the GPRs also hold 32 bits and store memory data. This provided us with knowledge on how different operations like add, subtract, multiply, move, and jump work inside the CPU. In this project, the mul operation was primarily used, demonstrated by the multiplication of the time and payment. Ultimately through the use of Verilog codes, this project made us realize how data works inside the CPU architecture.