CPU DESIGN LAB REPORT

Name: Krishna Prasath
Roll No: CS21B043

- I have written the verilog code for a simple CPU design.
- I will start with explaining the modules and explain the code along the way

Explaining the modules:

1)HalfAdder

```
//Explained in report

module HalfAdder(sum, carry, A, B);
  input A, B;
  output sum, carry;

  xor (sum, A, B);
  and (carry, A, B);

endmodule
```

- ☐ Inputs: A,B
- ☐ Outputs are sum and carry, represented by S and C in picture
- ☐ The outputs are the sum and carry obtained by adding 2-1 bit numbers
- ☐ Sum is xor of input and carry is and of input
- ☐ Here, carry in is not considered

2) FullAdder

```
`timescale 1ns/1ns

//Explained in report
module FullAdder(P, Q, cin, sum, cout);

  input P, Q, cin;
  output cout, sum;

  wire sum1;
  wire carry1;
  wire carry2;


        HalfAdder uut (.sum(sum1), .carry(carry1), .A(P), .B(Q));

  HalfAdder uut1 (.sum(sum), .carry(carry2), .A(cin), .B(sum1));

  or (cout, carry1, carry2);

endmodule
```

- ☐ Here, Inputs are P,Q,cin
- ☐ Outputs are carryout,sum.
- ☐ Compared to half adder, carry in input is also considered here

3) rca8

```
`timescale 1ns/1ns

//This is a 8 bit orca with a carry in(cin) and a carry out(cout)
module rca8(

  input [7:0] A, B,
  input cin,
  output [7:0] sum,
  output cout
);

  wire [7:1] carry;//used to represent the carry between the full
adders
  FullAdder abc (A[0], B[0], cin, sum[0], carry[1]); //The first bit
  FullAdder uut[6:1] ( A[6:1], B[6:1], carry[6:1], sum[6:1],
carry[7:2]);
  FullAdder bcd (A[7], B[7], carry[7], sum[7], cout); //The last bit

endmodule
```

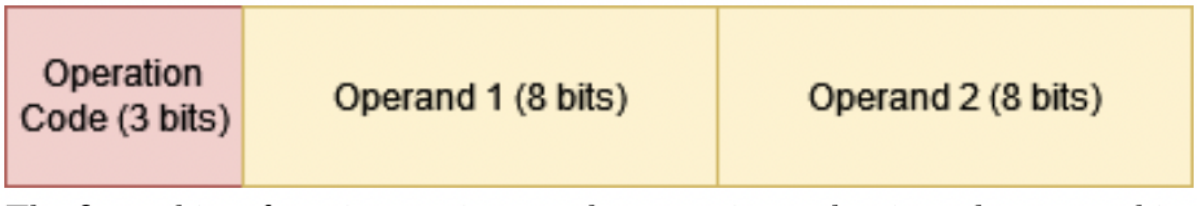
- ☐ This represents the code of a 8bit ripple carry adder
- ☐ Each time a full adder is used to find the sum and carry out from the input

4) decoder

```
`timescale 1ns/1ns

//A simple 3X8 decoder
module decoder(
  input wire [2:0]code,
  output wire [7:0]d
);
  wire not_a,not_b,not_c;
  wire a,b,c;
  // a,b,c represent the bits from left hand side respectively
  buf d1(a,code[2]);
  buf d2(b,code[1]);
  buf d3(c,code[0]);

  //Finding nots of three bits
  not a1(not_a,code[2]);
  not b1(not_b,code[1]);
  not c1(not_c,code[0]);

  and l0(d[0],not_a,not_b,not_c);//d0=a'b'c'
  and l1(d[1],not_a,not_b,c);  //d1=a'b'c
  and l2(d[2],not_a,b,not_c);  //d2=a'bc'
  and l4(d[3],not_a,b,c);  //d3=a'bc
  and l3(d[4],a,not_b,not_c);  //d4=ab'c'
  and l5(d[5],a,not_b,c);  //d5=ab'c
  and l6(d[6],a,b,not_c);  //d6=abc'
  and l7(d[7],a,b,c);    //d7=abc

endmodule
```

- ☐ A 3x8 decoder with code as the input and d as the 8 bit output
- ☐ Here a,b,c are the significant digits from left.

5) cpu.v



1) CU module

- ☐ This is the main file containing two modules specadd and cu
- ☐ In cu, we first start by getting the 19 bit input instruction
- ☐ We first use the decoder to convert the 3 bit op code into 8 bit select
- ☐ For example, 00000010 represents add
  00000100 represents subtract the 2 numbers, etc..
- ☐ Then we find the results of all the operations using the 2 operands

```
  //instruction[15:8] is operand 1
  //instruction[7:0] is operand 2
  rca8 uut00(instruction[15:8],instruction[7:0],x,add[7:0],cout);//8
bit ripple carry addition to calculate the result of adding the two 8
bit no
  not uut0[7:0](nott[7:0],instruction[15:8]);
//Used to calculate not of first operand(nott)
  and uut1[7:0](andt[7:0],instruction[15:8],instruction[7:0]);
//Used to calculate and of operands 1 and 2
  or uut2[7:0](ort[7:0],instruction[15:8],instruction[7:0]);
//Calculating or of operands 1 and 2
  rca8 uut3(instruction[15:8],inc[7:0],x,increment[7:0],cout1);
//incrementing the 8 bit no using orca
  rca8 uut4(instruction[15:8],dec[7:0],x,decrement[7:0],cout2);
//decrementing by adding 2's complement of 8bit 1 with operand1
  /*
  The next 3 lines are used to find subtraction result where notsec
is not of operand 2
  Then 8 bit 1 is added to subtr[7:0] to get subtraction result
  */
  not uut6[7:0](notsec[7:0],instruction[7:0]);
  rca8 uut7(instruction[15:8],notsec[7:0],x,subtr[7:0],cout4);
  rca8 uut8(subtr[7:0],inc[7:0],x,subtract[7:0],cout5);
```

- ☐ Add[7:0] is and result by rca
- ☐ Nott is not result
- ☐ Andt is and result
- ☐ Ort is or operation result
- ☐ For decrement, we add the operand 1 alone with 2's complement of 8 bit 1
- ☐ Similarly for subtraction, we find 1's complement of the operand 2, add it with operand 1 and further add1

2) Specadd Module

```
module specadd(
  input [7:0]op,
  input d,
  output [7:0]res
);
  and(res[0],d,op[0]);
  and(res[1],d,op[1]);
  and(res[2],d,op[2]);
  and(res[3],d,op[3]);
  and(res[4],d,op[4]);
  and(res[5],d,op[5]);
  and(res[6],d,op[6]);
  and(res[7],d,op[7]);
endmodule
```

```
except the one bit in select with 1
  specadd uut9(add[7:0],select[1],addres[7:0]);       //add u
  specadd uut10(subtract[7:0],select[2],subres[7:0]); //subtr
bit 2
  specadd uut11(increment[7:0],select[3],incres[7:0]);//incre
uses bit 3
  specadd uut12(decrement[7:0],select[4],decres[7:0]);//decre
uses bit 4
  specadd uut13(andt[7:0],select[5],andres[7:0]);     //and u
  specadd uut14(ort[7:0],select[6],orres[7:0]);       //or us
  specadd uut15(nott[7:0],select[7],notres[7:0]);     //not u
  /*
```

- ☐ Here, we use the property of select that for each operation ,only one bit has the value 1 and each time it is different.(1hot encoded)
- ☐ So , in select bit select[1] corresponds to add,
  Select[2] corresponds to subtract ..similarly for each
- ☐ Thus in specadd, we take the result and the value of the bit corresponding to the operation in select and and them.Thus is a bit corresponding to a particular operation is 1, only the result of that operation will remain.The and makes the result of the other operations become 8 bit zero
- ☐ I.E if select is 00000010, corresponding to add.Only the result of add is intact after specadd, the result of subtraction(subres), increment (incres)etc.. becomes zero
- ☐ Finally by doing or to all the results we get the required result.Since, x||0=x.

The Final Waveform: