

**EAST WEST UNIVERSITY**

**Course Title:** Computer Architecture

**Course Code:** CSE360

**Section No:** 3

**Project Report**

**Project Title:** Processor Micro-program

**Submitted To:**

Dr. Anisur Rahman

Assistant Professor,

Department of Computer Science and Engineering,

East West University.

**Submitted By:**

Shaykh Siddique

ID: 2016-1-60-053

Arif Hasan Nayon

ID: 2016-1-60-076

MD. Shohag

ID: 2016-2-68-043

# Title:

Using microprogram as instructions directly: Consider that there is no “instruction set”, no program counter (but microprogram counter), no instruction fetch in the normal sense. Your machine and “program” is THE microprogram itself. You have to add some fields into microprogram word such as: ADD R0, R1, R2 which holds the appropriate values.

# Objective:

The objective of this project is to know about the basic micro operation and its uses in our modern computer architecture. The control signals associated with operations are stored in special memory units inaccessible by the programmer as Control Words. Control signals are generated by a program are similar to machine language programs. Micro-program will fetch instructions with micro-program counter and makes the operations like addition.

# Theory:

An alternative to a hardwired control is a micro-programmed control unit, in which the logic of the control unit is specified by a micro-program. Micro-program (or firmware) consists of a sequence of instructions in a microprogramming language. These are very simple instructions that specify micro-operations. Micro-program is midway between hardware and software.

* Control unit design approaches-
  + - * Micro-programmed implementation
      * Hardwired logic implementation

**Micro-Programmed Implementation:**

* The control signal values for each micro-operation are stored in a memory (Registers).

Reading the contents of the control store in a prescribed order is equivalent to sequencing through the micro-operations.

**Hardwired logic implementation:**

* The Control Unit is viewed and designed as a combinatorial and sequential logic circuit.

The Control Unit is implemented by using any of a variety of “standard” digital logic techniques. The logic circuit generate the fixed sequences of control signals.

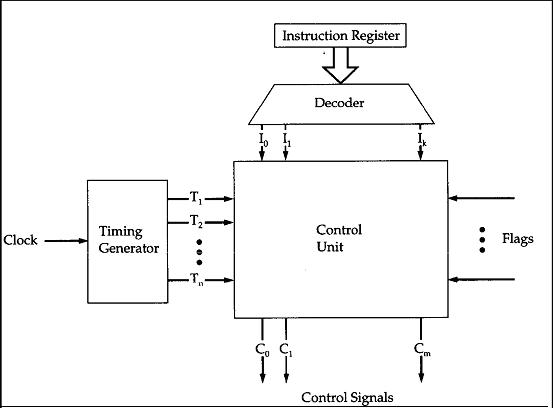
# Design and Implementation:

As Simulation criteria we can assume that, Instruction from Instruction Register will decoded and comes in control unit.

Instruction- ADD R1, R2, R3 will Considered as additional operation will held.

Here, C0, C1, … , Cn these are the control signals.

ADD will generate a control signal which will related to addition.



# 

Used four bit binary full adder for the addition.

**Algorithm:**

module full\_adder4(A, B, Cin, Sum, Cout);

input [3:0] A,B; // four-bit inputs

input Cin; // one-input input carry

output [3:0] Sum; // five-bit outputs

output Cout;

wire [2:0] carry;

// internal carry wires

adder fa0(A[0],B[0],Cin,Sum[0],carry[0]);

adder fa1(A[1],B[1],carry[0],Sum[1],carry[1]);

adder fa2(A[2],B[2],carry[1],Sum[2],carry[2]);

adder fa3(A[3],B[3],carry[2],Sum[3],Cout);

endmodule

# Debugging, test and run:

For testing the micro-program, giving the input values in register, we used a test bench program.

The value of two registers A and B are defined here.

Test Input:

initial begin

// Initialize Inputs

A = 0; B = 0; Cin = 0;

#5 A = 4'b0001; B = 4'b0001;

#5 A = 4'b0001; B = 4'b0010;

#5 A = 4'b0011; B = 4'b0101;

#5 A = 4'b0100; B = 4'b0110;

#5 A = 4'b0101; B = 4'b0110;

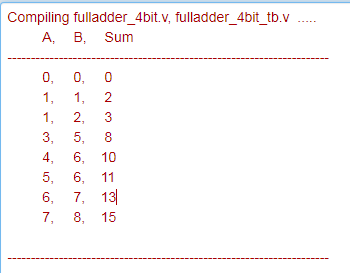
#5 A = 4'b0110; B = 4'b0111;

#5 A = 4'b0111; B = 4'b1000;

#10 $stop;

end

**Output:**



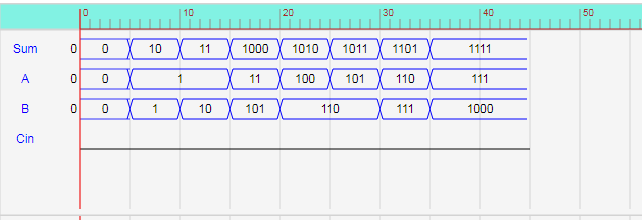
Tested Verilog Compiler:

* ModelSim from Mentor Graphics
* ISim from Xilinx ISE
* And Quartus 2

We used ModelSim from Mentor Graphics for simulation

# Result Analysis:

Relational waves for input and output values.



Here, A and B are Register and the output register is Sum/Cout .

# Conclusion and Future improvements:

The advantage to this approach is significant saving in control memory size (bits).

Disadvantage is more complexity and slower operation (doing 2 memory accesses for each microinstruction).

Micro-programmed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory. On this case Hardwired control is faster than micro-programmed control. A controller that uses this approach can operate at high speed as our software is a simulation software. It requires no additional hardware (decoders). And it is more flexible and reusable then hardwired control system.

For the typically large microprocessor systems today:

* + There are many instructions and associated register level hardware.
  + There are many control point to be manipulated.
  + The use of a micro-program on one machine to execute programs originally written to run on another machine.
  + By changing the microcode of a machine, you can make it execute software from another machine.

# Reference:

<http://nptel.ac.in/courses/Webcourse-contents/IIT-%20Guwahati/comp_org_arc/msword/m5_CPU_Design/5_Microprogrammed%20Control.doc>

<https://www.geeksforgeeks.org/computer-organization-hardwired-vs-micro-programmed-control-unit/>

# Source Codes:

Project -

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **module** adder(a, b, cin, s, cout);  **input** a, b, cin;  **output** s, cout;  **assign** s = a ^ b ^ cin;  **assign** cout = (a & b) | (cin & a) | (cin & b);  **endmodule**  **module** full\_adder4(**A**, **B**, Cin, Sum, Cout);  **input** [**3**:**0**] **A**,**B**; // four-bit inputs  **input** Cin; // one-input input carry  **output** [**3**:**0**] Sum; // five-bit outputs  **output** Cout;  **wire** [**2**:**0**] carry; // internal carry wires  adder fa0(**A**[**0**],**B**[**0**],Cin,Sum[**0**],carry[**0**]);  adder fa1(**A**[**1**],**B**[**1**],carry[**0**],Sum[**1**],carry[**1**]);  adder fa2(**A**[**2**],**B**[**2**],carry[**1**],Sum[**2**],carry[**2**]);  adder fa3(**A**[**3**],**B**[**3**],carry[**2**],Sum[**3**],Cout);  **endmodule** |

Test bench-

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | **module** test\_bench();  // Inputs  **reg** [**3**:**0**] **A**;  **reg** [**3**:**0**] **B**;  **reg** Cin;  // Outputs  **wire** [**3**:**0**] Sum;  **wire** Cout;  // Instantiate the Unit Under Test (UUT)  full\_adder4 uut (  .**A**(**A**),  .**B**(**B**),  .Cin(Cin),  .Sum(Sum)  );  **initial** **begin**  $display("**\t** A,**\t** B,**\t** Sum");  $display("---------------------------------------------------------------------");  $monitor("**\t**%d, **\t**%d, **\t**%d", **A**, **B**, Sum);  **end**  **initial** **begin**  // Initialize Inputs  **A** = **0**; **B** = **0**; Cin = **0**;  #**5** **A** = **4'b0001**; **B** = **4'b0001**;  #**5** **A** = **4'b0001**; **B** = **4'b0010**;  #**5** **A** = **4'b0011**; **B** = **4'b0101**;  #**5** **A** = **4'b0100**; **B** = **4'b0110**;  #**5** **A** = **4'b0101**; **B** = **4'b0110**;  #**5** **A** = **4'b0110**; **B** = **4'b0111**;  #**5** **A** = **4'b0111**; **B** = **4'b1000**;  #**10** $stop;  **end**  **endmodule** |