Gate Model of Computer System

Keonwoong Min San Jose State University keonwoong.min@sjsu.edu

Abstract: Project\_02 was a behavioral model of a computer system and I could know what things are going on the processor. This project is to observe how things are going on at gate level such that this project is possible to be implemented on a silicon.

1. Requirement for System - include CS147DV instruction set description as well.
2. Design and implementation of Memory.
3. Design and implementation of Processor.
4. Test strategy and test implementation (include text output and waveforms from simulator).
5. Conclusion
6. **REQUIREMENT FOR SYSTEM**

**a) Components of Gate Level Model**

Logic gates are used to build circuits at gate level modeling and Verilog has 8 different gates by default. Implicit connection could be used in instantiation and first parameter is output wire and others are input wires. However, this might be harder to find the error.

* and / nand
* or / nor
* xor / xnor
* buf/ not (one input, one output only)

ex) and and\_instantiation\_1(OUT, IN1, IN2)

**b) Half adder**

Half adder is needed for a full adder

- Half adder requires 1 xor gate and 1 and gate.

- 2 inputs, A and B, and 2 outputs, Y and C.

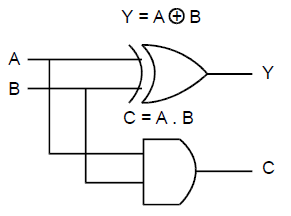
****

Figure 1.1 Half Adder Prototype

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y(A XOR B)** |
| 0 | 0 | **0** |
| 0 | 1 | **1** |
| 1 | 0 | **1** |
| 1 | 1 | **0** |

Figure 1.2 Half Adder Output Y

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C(A AND B)** |
| 0 | 0 | **0** |
| 0 | 1 | **0** |
| 1 | 0 | **0** |
| 1 | 1 | **1** |

Figure 1.3 Half Adder Output C

**b) Full adder**

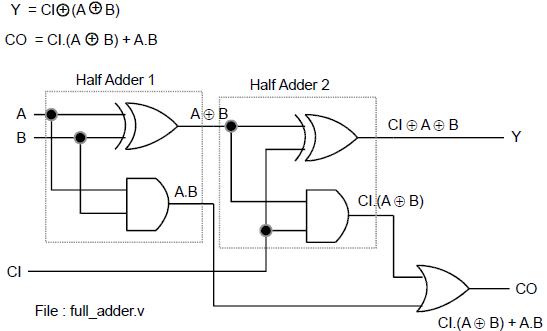
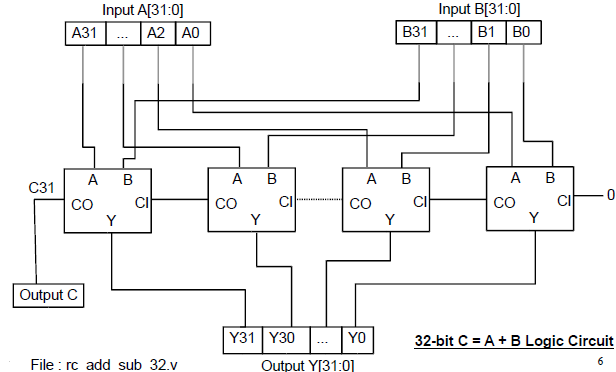


Figure 1.4 Full Adder Prototype

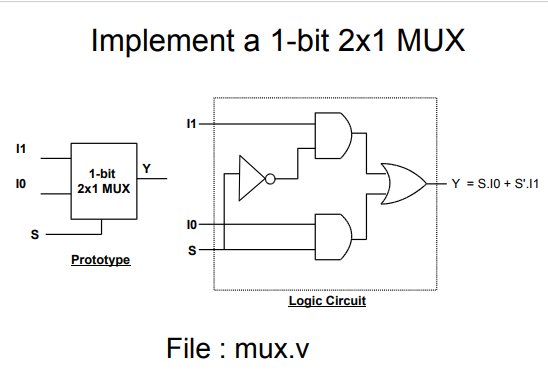
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **CI** | **Y** | **CO** |
| 0 | 0 | 0 | **0** | **0** |
| 0 | 0 | 1 | **1** | **0** |
| 0 | 1 | 0 | **1** | **0** |
| 0 | 1 | 1 | **0** | **1** |
| 1 | 0 | 0 | **1** | **0** |
| 1 | 0 | 1 | **0** | **1** |
| 1 | 1 | 0 | **0** | **1** |
| 1 | 1 | 1 | **1** | **1** |

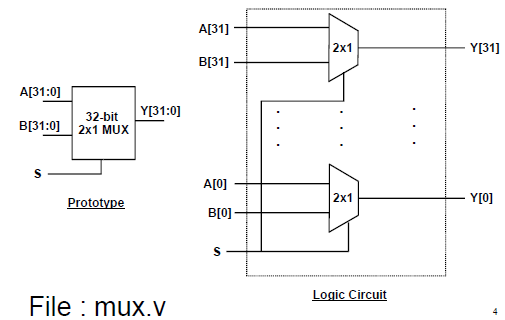
**Figure 1.5 Truth Table of Full Adder**

****

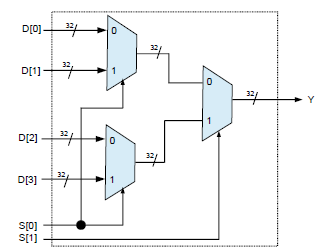
**Figure 1.6 32-bit Binary Ripple Carry Adder Prototype**

**c) Multiplexer**

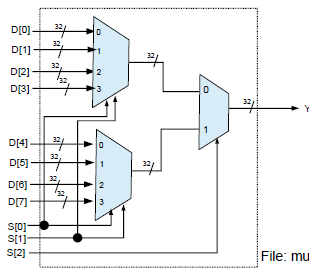
**Figure 1.7 1-bit 2x1 MUX Prototype**

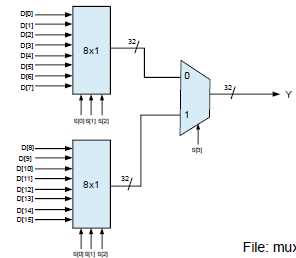
****

**Figure 1.8 32-bit 2x1 MUX Prototype**

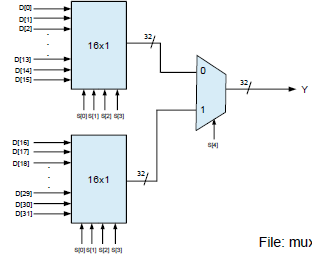
****

**Figure 1. 32-bit 4x1 MUX**

**Figure 1. 32-bit 8x1 MUX**

****

**Figure 1. 32bit 16x1 MUX**

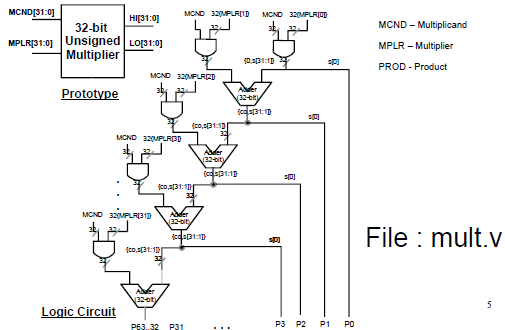
****

**Figure 32-bit 32x1 MUX**

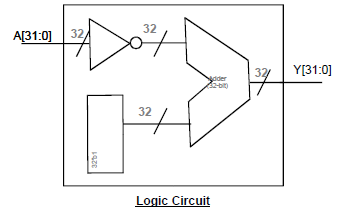
**d) Signed Multiplication Circuit**

**- To make a Signed Multiplication Circuit, we need a 32-bit Unsigned multiplier and 2’s complement.**

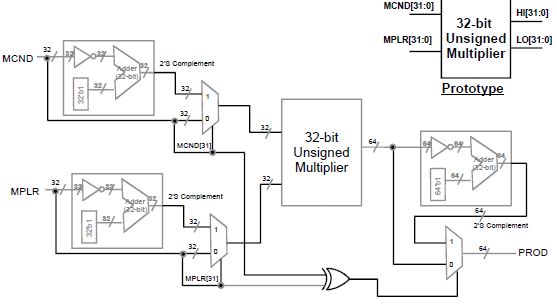
**Multiplier**

****

**Figure 1.9 32-bit Unsigned Multiplier**

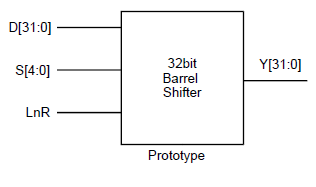
**2’s Complement**

**Figure 1.10 2’s Complement**

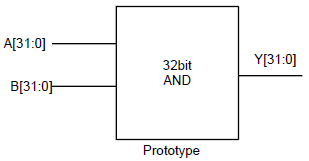
****

**Figure 1.11 32-bit Signed Multiplication Circuit**

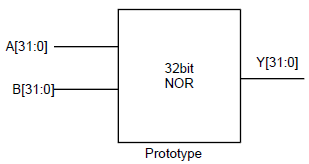
**e) Barrel Shifter**

**Figure 1.12 Barrel Shifter Prototype**

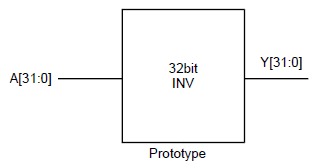
**f) 32-bit logic gates**

****

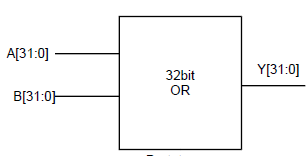
**Figure 1.13 32-bit AND**

****

**Figure 1.14 32-bit NOR**

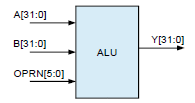
****

**Figure 1.15 32-bit INV**

****

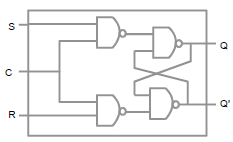
**Figure 1.16 32-bit OR**

**g) ALU**

****

**Figure 32-bit ALU**

**h) SR-Latch , D-Latch, Flipflop, and Register**

****

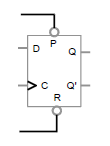
**Figure SR-Latch**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **C** | **S** | **R** | **Qt** | **Qt+1** | **Q`t+1** |  |
| 0 | x | x | 0 | 0 | 1 | Hold |
| 0 | x | x | 0 | 1 | 1 | Hold |
| 1 | 1 | 0 | x | 1 | 0 | Set in normal operation |
| 1 | 0 | 1 | x | 0 | 1 | Reset in normal operation |
| 1 | 0 | 0 | 0 | 0 | 1 | Hold |
| 1 | 0 | 0 | 1 | 1 | 0 | Hold |
| 1 | 1 | 1 | x | 1 | 1 | Undefined |

**Figure SR-Latch Truth Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **C** | **D** | **Qt** | **Qt+1** | **Q`t+1** |  |
| 0 | x | 0 | 0 | 1 | Hold |
| 0 | x | 1 | 1 | 0 | Hold |
| 1 | 1 | x | 1 | 0 | Set in normal operation |
| 1 | 0 | x | 0 | 1 | Reset in normal operation |

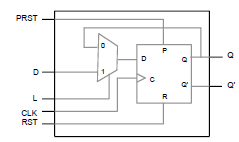
**Figure D-Latch Truth Table**

****

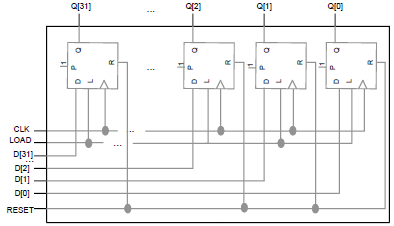
**Figure 1-bit Flipflop Prototype**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **C** | **D** | **P(nP)** | **R(nR)** | **Qt** | **Qt+1** | **Q`t+1** |  |
| X | x | 0 | 0 | X | ? | ? |  |
| X | X | 0 | 1 | X | 1 | 0 | Preset |
| X | X | 1 | 0 | X | 0 | 1 | Reset |
| 0 | X | 1 | 1 | 0 | 0 | 1 | Hold |
| 0 | X | 1 | 1 | 1 | 1 | 0 | Hold |
| 1 | 0 | 1 | 1 | X | 0 | 1 | Reset in normal operation |
| 1 | 1 | 1 | 1 | X | 1 | 0 | Preset in normal operation |

**Figure D-Flipflop Table**

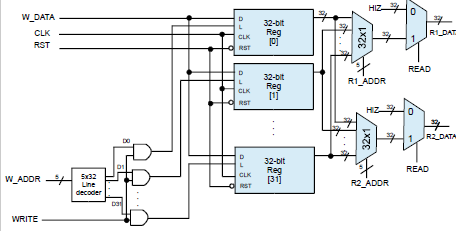
****

**Figure 1-bit Register**

****

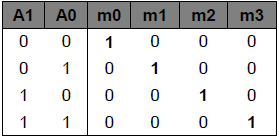
**Figure 32-bit Register**

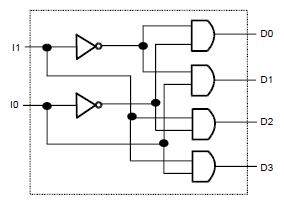
**To implement the 32x32-bit Register file, we need a 5 to 32 Line decoder which will come later in this report.**

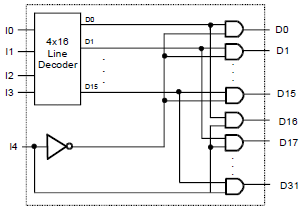
****

**Figure 32x32-bit Register File**

**i) Decoder**

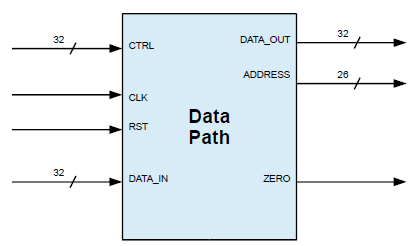
**Figure 2 to 4 Decoder truth table**

**Figure 2 to 4 Line Decode**

**Figure 5 to 32 Line Decoder**

**j) Data Path**

**- Data path is used to implement the processor of this project with a control unit.**

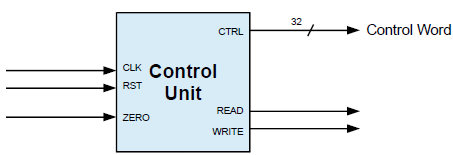
****

**Figure Data Path Prototype**

**k) Control Unit**

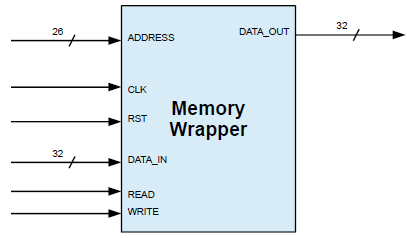
**- The control unit is not implemented as a gate level since it is too complicated for out level.**

**- It is used to implement PROC\_CS147\_SEC05, the processor of this project.**

**Figure Control Unit Prototype**

**l) Memory Wrapper**

**- Memory Wrapper and PROC\_CS147\_SEC05 are used to build a little computer, DAVINCI.**

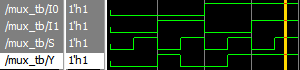
**Figure Memory Wrapper Prototype**

**2. Design and Implementation of Memory**

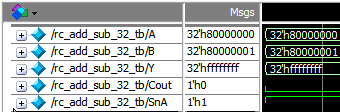
**3. Design and Implementation of Processor**

**4. Test Strategy and Test Implementation(include text output and waveforms from simulator)**

**5. Conclusion**

****

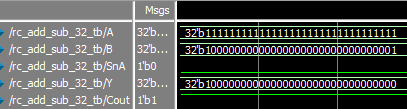
**Figure 1-bit 2x1 Mux Testbench result**

****

**Figure Ripple Counter Testbench**

A = 32'b10000000000000000000000000000000;

B = 32'b10000000000000000000000000000001; SnA = 1'b1; (sub)

****

**Figure Ripple Counter Testbench**

A = 32'b11111111111111111111111111111111; B = 32'b10000000000000000000000000000001; SnA = 1'b0; (add)

Y is 1000 0000 0000 0000 0000 0000 0000 0000

Cout, carry out, is 1

Such that the total number is

1 1000 0000 0000 0000 0000 0000 0000 0000

This Davinci v1.0, a minimum computer system, provides instruction set cs147DV in lecture note 01. This system contains Processor and Memory. In the Processor, there are a 32 x 32 Register file, an ALU, and a Control Unit as shown in Figure 1.1.

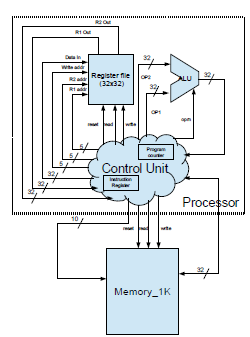
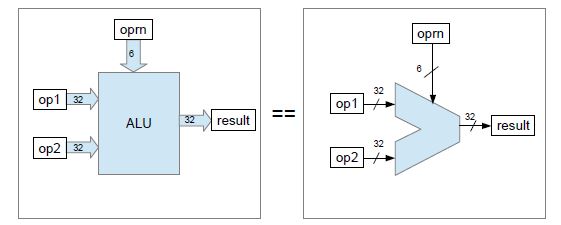


Figure 1.1 Diagram of Control System Model

ALU is a fundamental unit block of central processing unit (CPU). It is a digital electronic circuit that generally supports many basic arithmetic and bitwise logic functions on a computer. The input data is called operands that is operated with operations in ALU.

Since an ALU supports basic arithmetic and logic functions, it takes any mathematic and logical programs with two operand operations such as in Figure 3.1. It also has a special shape to represent it as shown in Figure 1.2. Figure 1.2 Diagram of ALU

Computer System needs to know when the sub-digital electronic circuit system is done unless if causes delay problem. Clock provides synchronization point to solve the delay problem as shown in Figure 1.2 and 1.3.

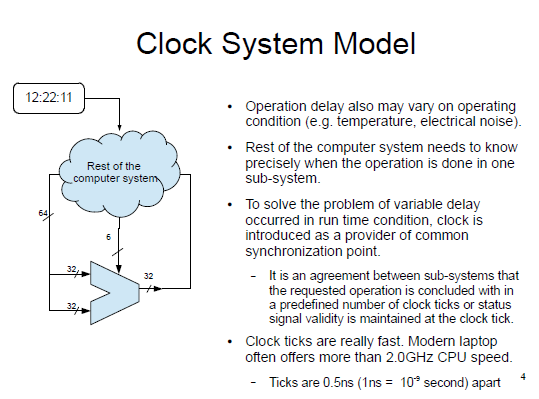


Figure 1.2 Diagram of Clock System Model

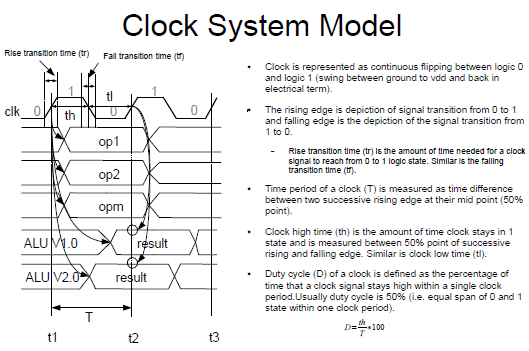


Figure 1.3 Clock System Model

Memory is in the DaVinci v1.0 system. It contains 32bit x 64M storage which is 256 MB in total and this storage provides data and instruction. All the input, output, and inout ports are described in the Figure 1.4

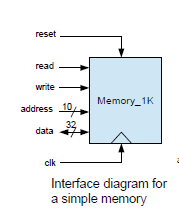


Figure 1.4 Diagram of Memory

Register file is also part of a Davinci v1.0 system. It has a 32x32 storage which stores 32 registers and each register is a word size, 32bit. It contains group of registers.

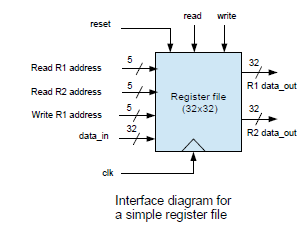


Figure 1.5 Diagram of Register File

Control unit is a brain of the processor. All the components get the control signal from the control unit. . It also has a program counter register to hold the power on instruction and an instruction register to hold the current instruction. The stack pointer register is used in control unit for the instruction push and pop.

Control unit has 5 states, instruction fetch, instruction decode, execution, memory access, and write back as shown in Figure 1.6

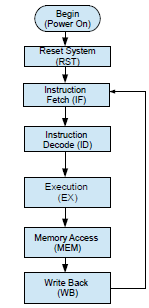


Figure 1.6 Diagram of stage of control unit

**2. DESIGN AND IMPLEMENTATION OF MEMORY**

**Design of Memory**

- Memory has a 32bit x 64M storage and one output, one inout for data and five input ports for read signal, write signal, address, clock cycle, and reset. as shown in Figure 2.1. The inout port needs to be controlled depends on which operation. It operates at positive edge of clock and wipes out the data at negative edge of reset signal. Instruction is gotten from the “mem\_content\_01.dat” by default with using parameter.

**A. Input port**

a) READ and WRITE inputs are the control signal for this memory. CLK is for clock cycle and RST is for the reset signal

b) ADDR port has 26bit since the storage has 64 million sections. `ADDRESS\_INDEX\_LIMIT is declared in the “prj.definition.v” file, and included by using `include at the first line, above the module declaration.

**B. Inout port**

a) Inout port, DATA, has 32bit buses since each part of the storage can contains 32bit of data. `DATA\_INDEX\_LIMIT is declared in the “prj.definition.v” file.

**C. Reg**

a) sram\_32x64m is a storage with 256MB b) data\_ret is used when it is in read operation to refresh the data from the storage. It needs to get the address to take out the right data in the right place.

**C. Integer**

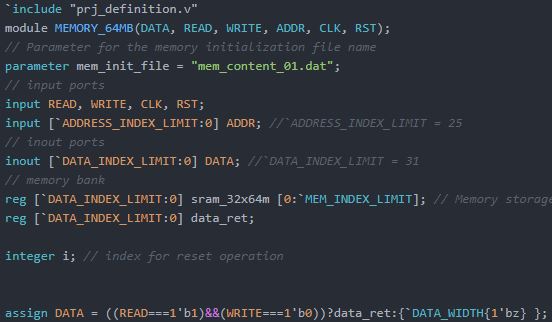
a) the integer type of ‘i’ is used for the for loop later in the implementation.

**D. Hold inout port, DATA**

a) Since the DATA is inout port, it cannot operate read and write operation at the same time such that when it is not in read operation, DATA needs to be set to Hiz and to date\_reg otherwise.

**E. Parameter**

a) a parameter, mem\_init\_file is used to declare a constant value. It is set to “mem\_content\_01.dat” which contains instruction sets from a certain address.

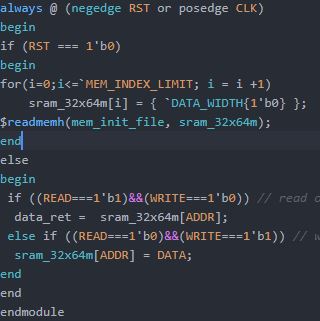
Figure 2.1 Memory design and implementation

**F. Implementation**

a) always@(negedge RST or posedge CLK) is used to operate codes at the negative edge of reset or the positive edge of the clock as shown in Figure 2.2

b) It wipes out all the data in the storage at the negative edge of reset by using for loop. And read memory in hexadecimal from the parameter, mem\_init\_file.

c) When it is in the read operation, it picks out the data from the right place by the address and when it is in the write operation, it stores the data from the DATA port to the storage.

****Figure 2.2 Memory design and implementation

**3. DESIGN AND IMPLEMENTATION OF PROCESSOR**

**Design of Processor**

- Processor has three different modules, Control Unit, Register File, and ALU. Control Unit is connected to Register File, ALU, and Memory to control all three modules. Instruction set has three different types, R-type, I-type, and J-type.

**Design of ALU**

- ALU supports 9 functions, addition, subtraction, multiplication, shift right, shift left, bitwise, or, nor, and set less than. It gets the result with 2 operands and 1 operation. It also contains ZERO output and used whenever the result is 0.

**A. Input port**

a) OP1 and OP2 are input ports for one operand and the other operand. It has 32bit bus as shown in Figure 3.1.

b) OPRN is the input port for the operation which has 6bit bus as shown in Figure 3.1.

**B. Output port**

a) OUT is one of output ports in ALU for the result, so it has 32bit bus.

b) ZERO is the other output port for the result of zero.

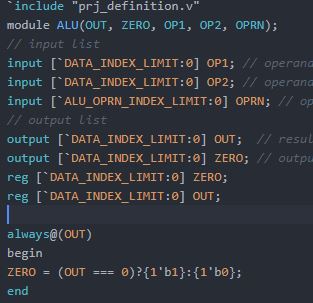
**C. Reg**

a) ZERO and OUT reg for output ports, OUT and ZERO

**C. Implementation**

a) ZERO is set to 1 whenever the result is 0, and set to 0 otherwise as shown in Figure 3.1

b) Whenever OP1 or OP2 or OPRN changes OUT is calculated based on the OPRN case which contains addition, subtraction, multiplication, shift right, shift left, bitwise, or, nor, and set less than in order.

****Figure 3.2 ALU design and implementation

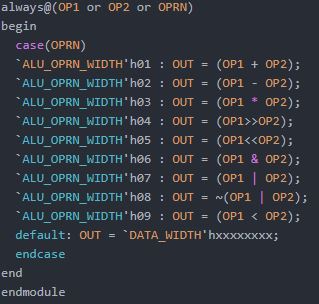
****

Figure 3.3 ALU design and implementation

**Design of Register File**

- Register File has 7 input ports and 2 output ports. It also has a storage 32x32 such that it contains 32 registers.

**A. Input & Output port**

a) CLK, RST input ports for clock cycle and reset signal, READ and WRITE input ports for the read operation and write operation.

b) DATA\_W is used for the data to write into the register file as shown in Figure 3.4.

C) ADDR\_R1 and ADDR\_R2 gets address to choose the register that the control unit want to read. ADDR\_W gets the address for the write operation to write in the specific spot in the register file.

**B. Reg**

a) register\_file\_32x32 is used for the storage.

b) DATA\_R1 and DATA\_R2 are registers for corresponding output ports.

**C. Implementation**

a) Initialize storage - As the register file module begins, it initialize the storage as 1 bit of 0 using for loop as shown in Figure 3.4

b) always@ - always@(negedge RST or posedge CLK) is used like memory, meaning whenever at the negative edge of reset signal and positive edge of clock.

c) Wiping out - At the negative edge of reset signal, it wipes out all the data and put 1 bit of 0 into the storage.

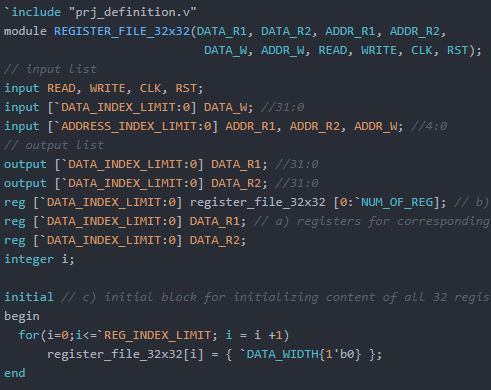


Figure 3.4 Register File design and implementation

d) Read operation - When it is in read operation, data in the storage with the specific address is assigned to the output port such as

DATA\_R1 = register\_file\_32x32[ADDR\_R1]; as shown in Figure 3.5.

e) Write operation – Data is set to the register file in the specific address as shown in Figure 3.5

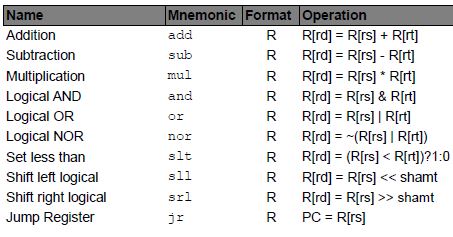
Figure 3.5 Register File design and implementation

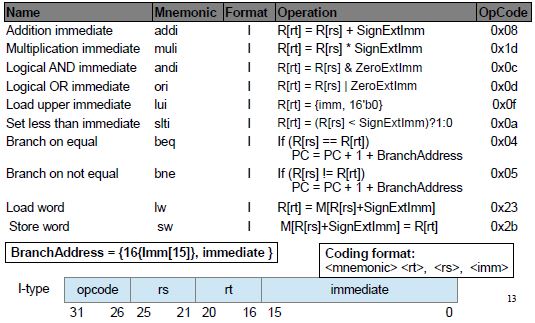
**Design of Control Unit**

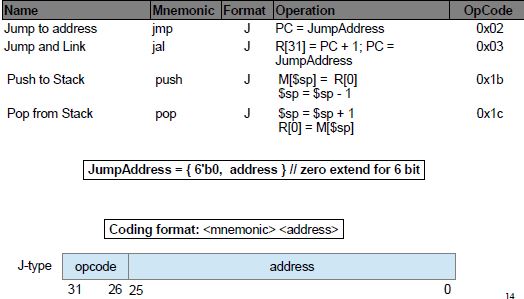
- Control Unit has input ports and output ports for Memory, Register File, and ALU, since it is a main brain in this DaVinci\_01.v.

A module “PROC\_SM” which is a state machine that determines the next state. It contains 5 states, PROC\_FETCH, PROC\_DECODE, PROC\_EXE, PROC\_MEM, and PROC\_WB. Each state contains own instructions to do depends on the opcode and fuct.

Control Unit has two special registers, Program counter and Instruction register. Program counter holds the memory address to fetch and Instruction register holds the current instruction.

Figure 3.1 Control Unit Instruction Set, R-type

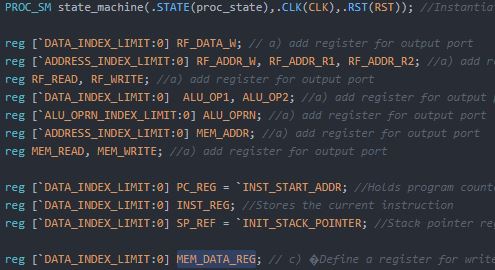
Figure 3.1 Control Unit Instruction Set, I-type

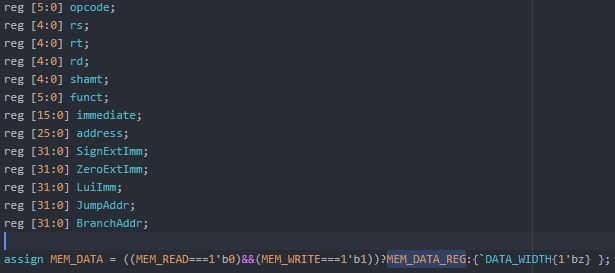
Figure 3.1 Control Unit Instruction Set, J-type

**A. Reg**

a) Reg’s are declared corresponding to the output ports and MEM\_DATA\_REG is for the inout port, MEM\_DATA, in memory as shown in Figure 3.7

b) Other registers are to parse the instruction and to save some data such as immediate, SignExtImm for later use as shown in Figure 3.7

Figure 3.6 Control Unit design and implementation

Figure 3.7 Control Unit design and implementation

**A. State Machine**

a) State machine is used to determine the next state based on the current state in Control Unit as shown in Figure 3.8.

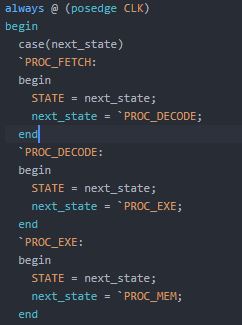
****

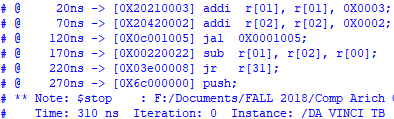
Figure 3.8 Control Unit design and implementation

**4. TEST STRATEGY AND TEST IMPLEMENTATION**

To Test the implementation code of davinci\_01, test bench is used on ModelSim simulator. Following depicts the result of the test bench on Davinci\_01.

**Text Output**

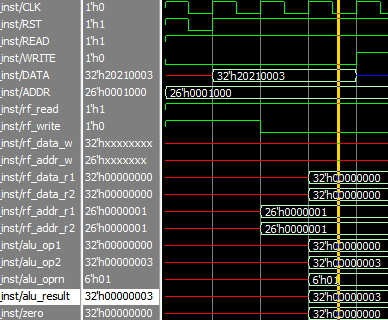
DA\_VINCI\_TB read the text file, "myfibo.txt". It operates 2 addition immediate and jump and link to subtraction. Then it goes back to one after the jump and link which is push as shown in Figure 4.1.

Figure 4.1 Text Output

**Processor**

**A. addi r[1], r[1], 0x3**

1. DATA is 32’h 20210003 as declared in the “myfibo.txt” file.
2. ADDR is set to power on address which is 32’00001000 declared in “prj.definition.v”.
3. Both addr\_r1 and addr\_r2 has same instruction since it is Addi.
4. OP1 is 32’h0000000, OP2 is 32’h00000003, and OPRN is 6’h1 which is addition and the esult is 32’h00000003.

 Figure 4.2 Processor Waverform First Add Immediate

**B. addi r[2], r[2], 0x2**

1. DATA is 32’h 20420002 as declared in the “myfibo.txt” file.
2. Both addr\_r1 and addr\_r2 has same instruction since it is Addi.
3. OP1 is 32’h0000000, OP2 is 32’h00000002, and OPRN is 6’h1 which is addition.
4. Result is 32’h00000002.
5. Write the result into the Register file when it is on write operation as shown in Figure 4.3.

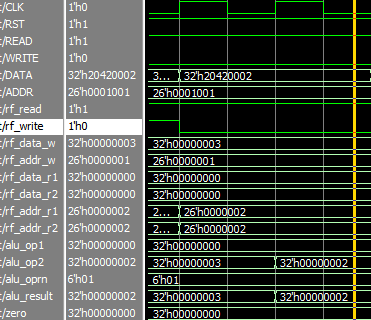


Figure 4.3 Processor Waverform Second Add Immediate

**C. jal, Jump to subtration instruction**

1. As the DATA gets to 32’h0c001005, it jumps to 32’h00220022 as shown in Figure 4.4

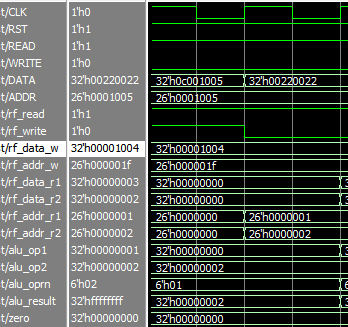


Figure 4.4 Processor Waverform Jump and Link

**D. sub r[0], r[1], r[2]**

1. As the DATA is 32’h00220022, it operates the subtraction.
2. Both addr\_r1 and addr\_r2 has same instruction since it is subtraction.
3. OP1 is 32’h0000001, OP2 is 32’h00000002, and OPRN is 6’h1 which is addition.
4. Result is 32’hffffffff.
5. Write the result into the Register file when it is on write operation as shown in Figure 4.5.

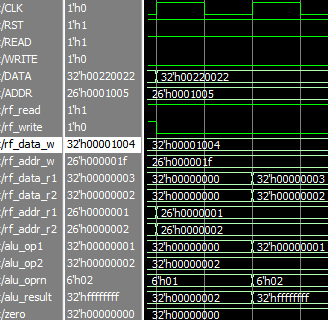


Figure 4.5 Processor Waverform Subtraction

**D. jr r[31] back to jal + 1 and Push**

1. As the DATA is 32’h 03e00008, it supposed to jump back to right next to the jal which is 32’h6c000000.

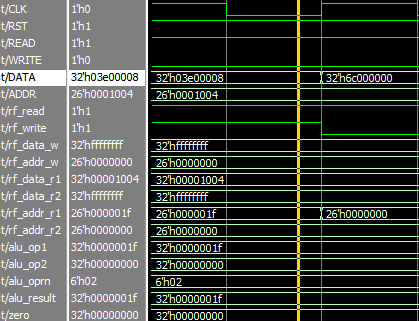


Figure 4.6 Processor Waverform Jump Register to Push

**6. CONCLUSION**

I spent tremendous of time to model the Davinci v1.0, a minimum computer system. It was hard to get an intuitive about the computer system in hardware perspective when I just listened the lecture. I thought I understood at the moment I learned lecture and read the notes, but I was not quite understanding it correctly. I made many errors and mistakes while I was doing this project. Especially, push, pop, sw, and lw were quite challenging to figure out where I need to deal with the Program counter or Stack pointer register. Modifying Davinci test bench was not hard work but I wasted long time to figure out the algorithm error that I made. If statement did not work for my project, so I had to use Case statement instead and I still do not understand why If-statement did not work for the module PROC\_SM. I was frustrating for a long time but what I learned from this project is valuable and beneficial. It was a great experience to do this project.

**7. REFERENCES**

1. Computer Organization and Design (5th Edition) by David A. Patterson and John L. Hennessy, 2014

2. CS147 Lecture 01 by Kaushik Patra, 2014