Computer Architecture: CPU Design

Lamiah Khan and Megan Vo

Professor: Robert Marano

05/10/2024

MIPs Green Sheet:

Logic Documentation

Logic	Format	Definition
Add	R	R[rd] = R[rs] + R[rt]
Subtract	R	R[rd] = R[rs] - R[rt]
Mult	R	{Hi, Lo} = R[rs] * R[rt]
Div	R	Lo = R[rs] / R[rt], Hi = R[rs] % R[rt]
And	R	R[rd] = R[rs] & R[rt]
Slt	R	R[rd] = (R[rs] < R[rt]) ? 1:0
Jr	R	PC = R[rs]
Lw	I	R[rt] = M[R[rs] + SignExtImm]
Sw	I	M[R[rs] + SignExtImm] = R[rt]
Beq	I	if (R[rs] = R[rt]) PC = PC+4+BranchAddr
Addi	I	R[rt] = R[rs] + SignExtImm
J	J	PC = JumpAddr
Jal	J	R[31] = PC+8; PC = JumpAddr
Slti	I	R[rt] = (R[rs] <SignExtImm) ? 1:0

Where: SignExtImm = { 16{immediate[15]}, immediate}, ZeroExtImm = { 16{1b'0}, immediate }
BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }, JumpAddr = { PC+4[31:28], address, 2'b0 }

Command Documentation

Command	Value	Purpose
$high	1	High value
$low	2	Low value
$v0-$v7	3-10	Functions will return value
$ar0-ar9	19-28	Arguments
$t0-$t9	29-44	Temporary values
$gpr0-$gpr90	45-108	GPRs
$pc	109-117	Program counter
$g03-Sgp3	11-18	Global pointers
$ddp	117	Dynamic Data Pointer
$sp	118	Stack pointer
$fp	119	Frame pointer
$zero	0	zero
$ra0-$ra3	120-127	Returns addresses

Instruction Format

Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R- Type	Opcode			Rs			Rt			Rd			funct			
I - Type	Opcode			Rs			Rt			7-bit immediate						
J-Type	Opcode			13-bit Address Bit												

R-Type Opcodes and Base Conversion

Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Opcode			Rs			Rt			Rd			funct			
add	0	0	0	x	x	x	x	x	x	x	x	x	0	0	0	0
sub	0	0	0	x	x	x	x	x	x	x	x	x	0	0	0	1
mult	0	0	0	x	x	x	x	x	x	x	x	x	0	0	1	0
nor	0	0	0	x	x	x	x	x	x	x	x	x	0	0	1	1
slt	0	0	0	x	x	x	x	x	x	x	x	x	0	1	0	0
div	0	0	0	x	x	x	x	x	x	x	x	x	0	1	0	1
mfhi	0	0	0	x	x	x	x	x	x	x	x	x	0	1	1	0
mflo	0	0	0	x	x	x	x	x	x	x	x	x	0	1	1	1
jr	0	0	0	x	x	x	x	x	x	x	x	x	1	0	0	0

I-Type Opcodes and Base Conversion

Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Opcode			Rs			Rt			7-bit immediate						
Slti	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x
Lw	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x
Sw	1	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x
Beq	1	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x
Addi	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x

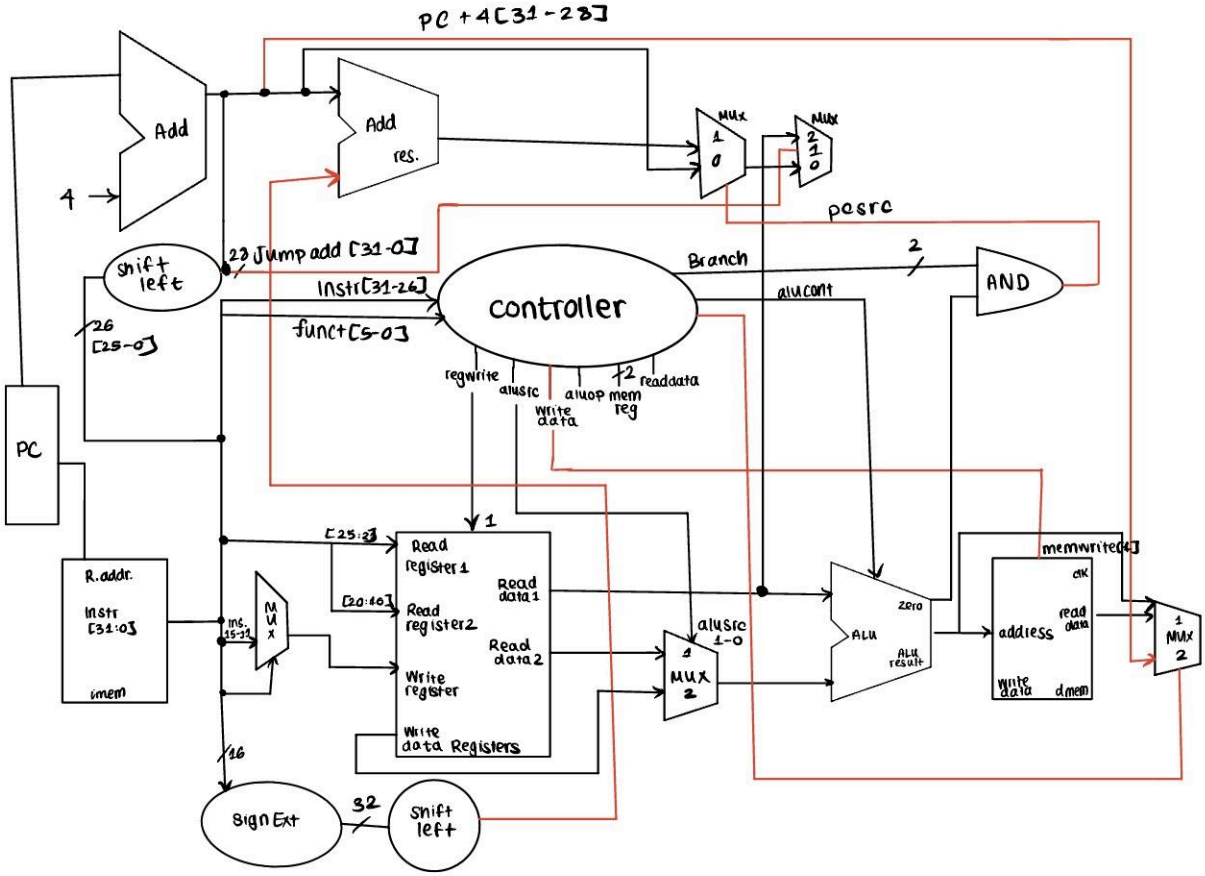
J-Type Opcodes and Base Conversion

Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Opcode			13-bit Address Bit												
J	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x
Jal	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x

Overall Design Explanation:

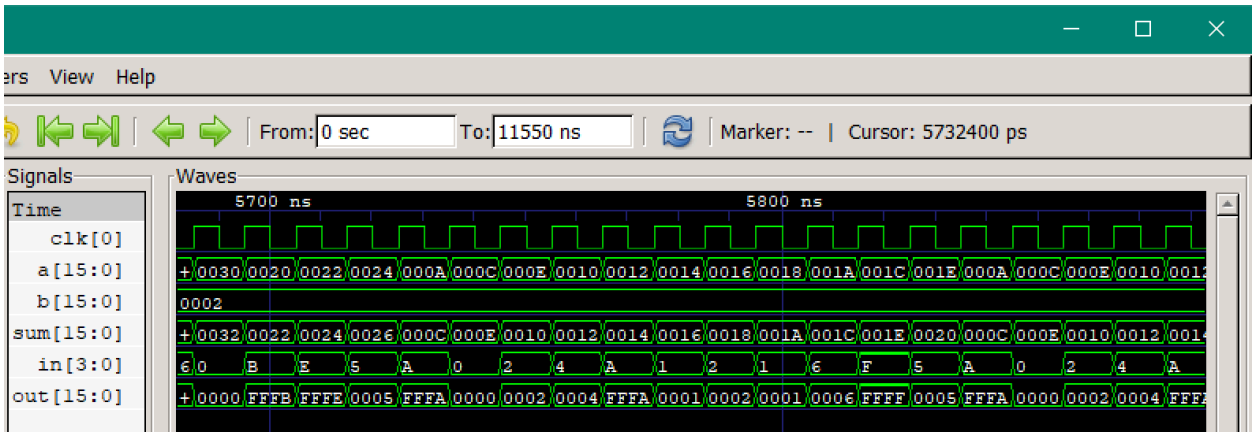
In this project, a CPU following the Von Neumann architecture will be programmed via Verilog. The CPU will retrieve and execute a subset of MIPs instruction, specifically targeting two example programs. Specifically, when the computer is run, the CPU will fetch and print instructions from a memory unit. The CPU follows a Reduced Instruction Set Computer (RISC) processor, as it is emulating the MIPS processor. In order to design the CPU efficiently, various components were implemented. This includes the ALU, ALU decoder, adder, clock, main computer, controller, SPU, datapath, D-flip flop, Data memory, integrated memory, main decoder, 2:1 multiplexer, register file, shift operations, and sign extender. In terms of size, it was decided that the CPU would be 32 bits, and the RAM would be 4GB. Additionally, the RAM layout will inherit the Big Endian format. Overall, the design components consist of R-type, J-type, and I-type instruction, as defined by the provided MIPS Green Sheet.

Main Design Diagram:



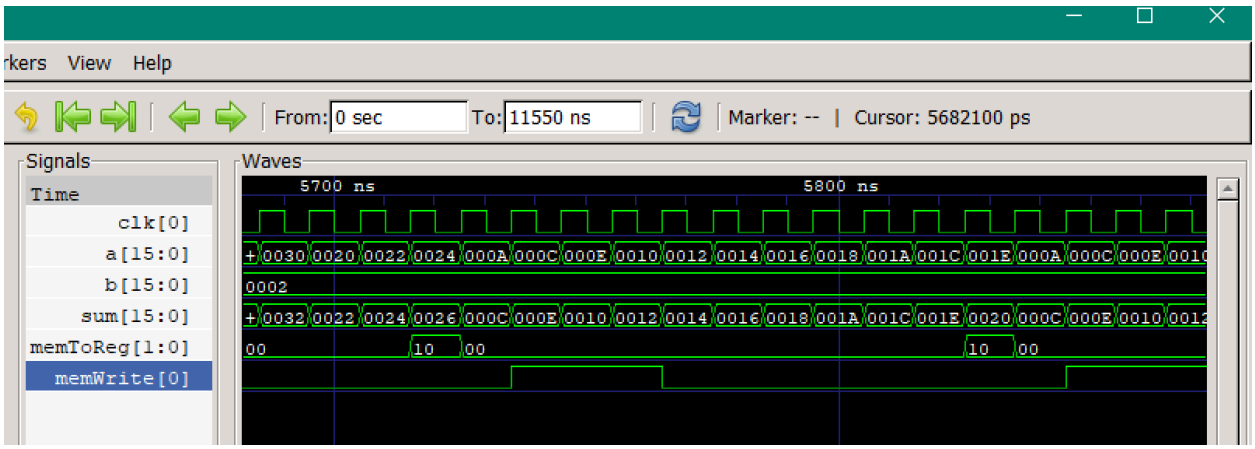
R-Type Timing Diagram:

The R-type instruction format is utilized for register-based instructions. This includes arithmetic or logic operations ran by the ALU, such as addition, subtraction, etc. The following waveform is generated from ETKWave, and displays various signals generated during the execution of an R-type instruction:



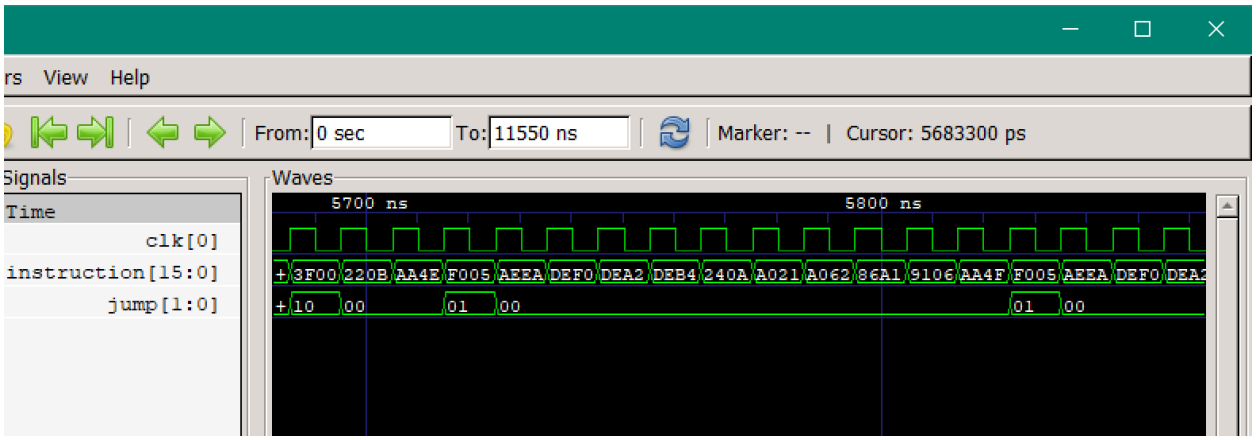
I-Type Timing Diagram:

I-type instructions are for operands involving immediate values, and memory operations. In the context of the CPU, this includes the integrated memory and data memory. The following wave form displays signals generated by an I-type instruction:



J-Type Timing Diagram:

Similarly, a J-type instruction is used during jump commands. The figure below displays the waveform of a J-type instruction. Similar waveforms also appear in our Fibonacci program, as shown in a later section.



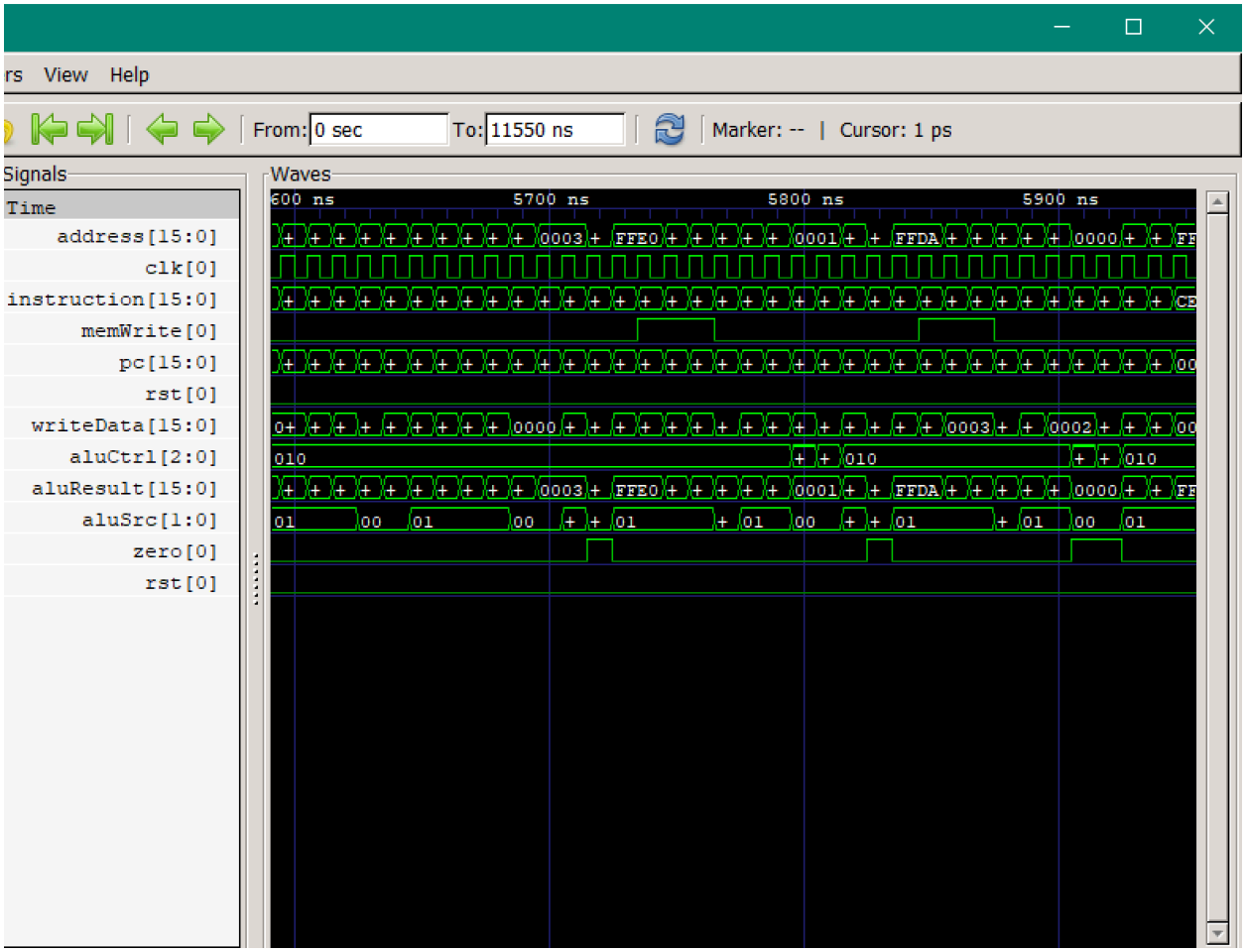
Fibonacci Program Example:

A Fibonacci sequence was run on the computer, in order to demonstrate the CPU's ability to perform recursion or nested features. The following is the MIPs program, and the associated machine code for the program:

```
1     main:     addi $2, $0, 5
2               addi $3, $0, 12
3               addi $7, $3, -9
4               or $4, $7, $2
5               and $5, $3, $4
6               add $5, $5, $4
7               beq $5, $7, exit
8               slt $4, $3, $4
9               beq $4, $0, next
10              addi $5, $0, 0
11    next:     slt $4, $7, $2
12              add $7, $4, $5
13              sub $7, $7, $2
14              sw $7, 68($3)
15              lw $2, 80($0)
16              j exit
17    exit:     sw $2, 84($0)
```

```
1     20020005
2     2003000c
3     2067fff7
4     00e22025
5     00642824
6     00a42820
7     10a7000a
8     0064202a
9     10800001
10    20050000
11    00e2202a
12    00853820
13    00e23822
14    ac670044
15    8c020050
16    08000011
17    20020001
18    ac020054
```

The waveform signal outputted with an input of 5 is as follows:



Once again, the waveform is clear, and the results are accurate.

Conclusion:

In conclusion, the CPU successfully ran both a simple MIPs program, and a sample Fibonacci sequence. Therefore, the CPU is successful in its building. Moreover, from the results, the waveforms derived are clear. For future improvements, pipelining should be considered. Regardless, the CPU is functioning, and follows correct Von Neumann architecture concepts.