Akshay Karve
Alexander Volkov
ESE 345 Final Project
12/2/2019

Goals

The goal of this project is to design and simulate a simple 4 stage multimedia unit. In this course, we've learned about the design process of a CPU and all the building blocks and now this information was used in a real-life application. This project made us face many challenges and gave an opportunity to apply the knowledge we've been taught.

Design process

We've started with analyzing the requirements for this project. We had to create an ALU capable of performing r3 and r4 type instructions as well as have 4 pipelined stages for all types of instructions. The multimedia unit also needed a 32 registers Register File, a 64 rows Instruction Buffer, program counter, forwarding unit, control unit, and pipeline stage registers.

First block we began to design was ALU. The way we implemented ALU is by creating two blocks within ALU each responsible for either r3 or r4 type of operation with a multiplexor for the output result. r3 type of operations was arithmetic or logical operations that take 1 or 2 inputs and one output. Using opcode this block was implemented with a case switch for every one of the instructions. All instructions are implemented using subword parallelism meaning each 128 bit register has 4 different words or 8 different halfwords that we can perform operations on. R4 type is (a*b)+/-c. It's operating with integers or long values and it's implemented with saturation. Then depending on R bit (23rd bit of the instruction), ALU selects an output of r3 or r4 type.

Register file takes 3 register addresses to be read from the decoder, register address to be written to and write enable bit from EXE/WB register, load index for loading immediate. Data to be written into the register is multiplexed between immediate value and the result of ALU depending on the immediate bit (24th bit). It outputs data from 3 registers to ID/EXE register.

The decoder takes instruction and splits it. 24th bit an immediate bit indicating whether it's a load immediate or r type instruction. 23th bit is r type distinguishing between r3 and r4 type of operation. 23-21 is the load index for loading immediate. 22-20 is opcode for r4 instruction, 19-15 is opcode for r3 instruction. 19-15 is r3, 14-10 is r2, 9-5 is r1, 4-0 is rd.

Instruction buffer takes in pc that acts as counter and outputs an instruction pc is pointing at in the array. It's preloaded with machine code from an external file.

Control unit take in immediate bit (24), rtype bit (23) and opcode (22-15). It outputs immediate bit (24), rtype bit (23), ALUop and regWrite signal. regWrite is disabled on nop opcode.

The assembler is implemented in python. It takes code instructions from a file and parses them into a corresponding machine code file with correct opcode and register address values. That machine code is then written to a file that the instruction buffer is reading from.

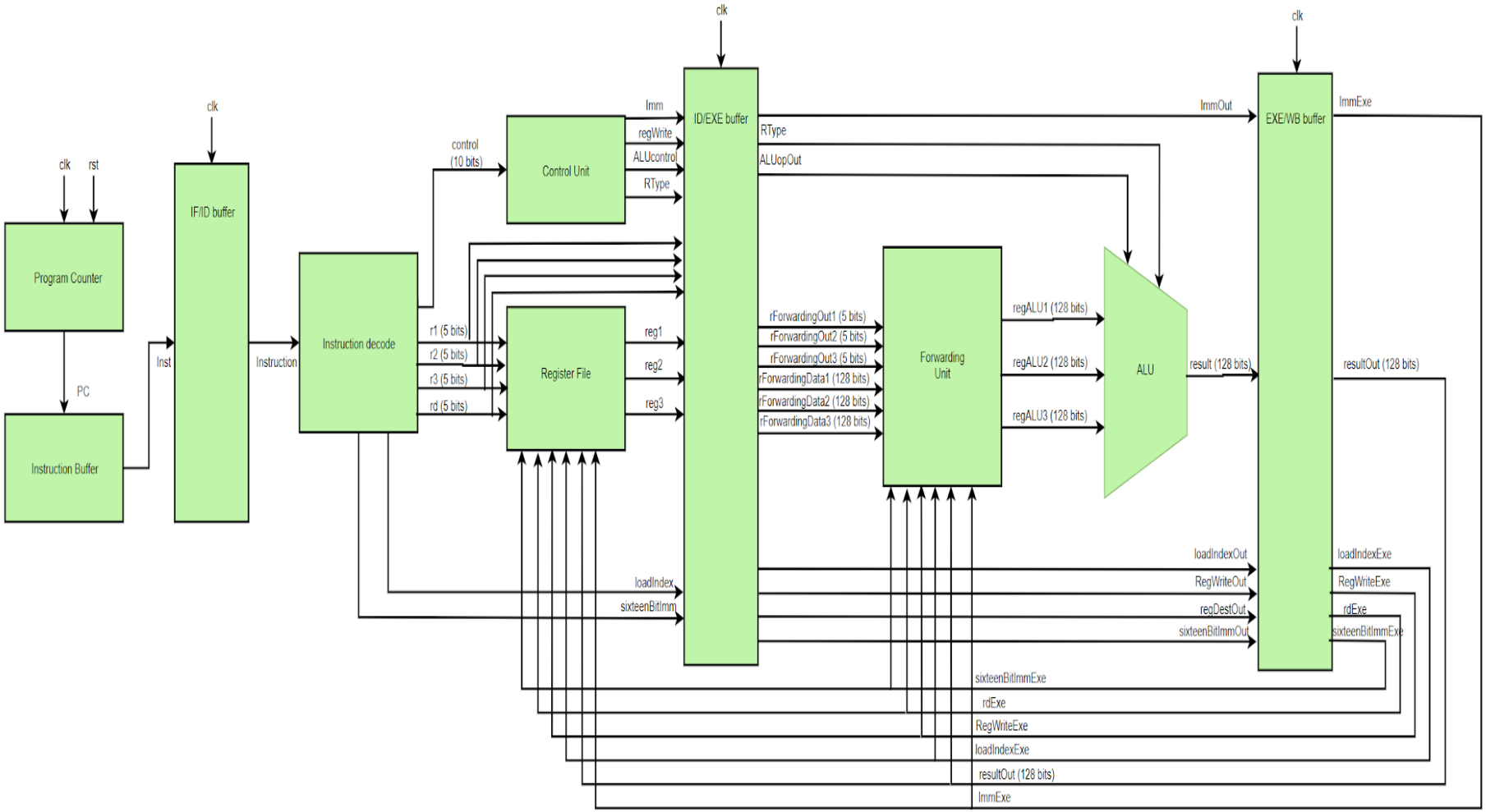
Pipeline registers are the only blocks that are clocked with an exception for the Program Counter. They simply pass through values from the previous stage to the next stage on every rising edge.

Conclusions

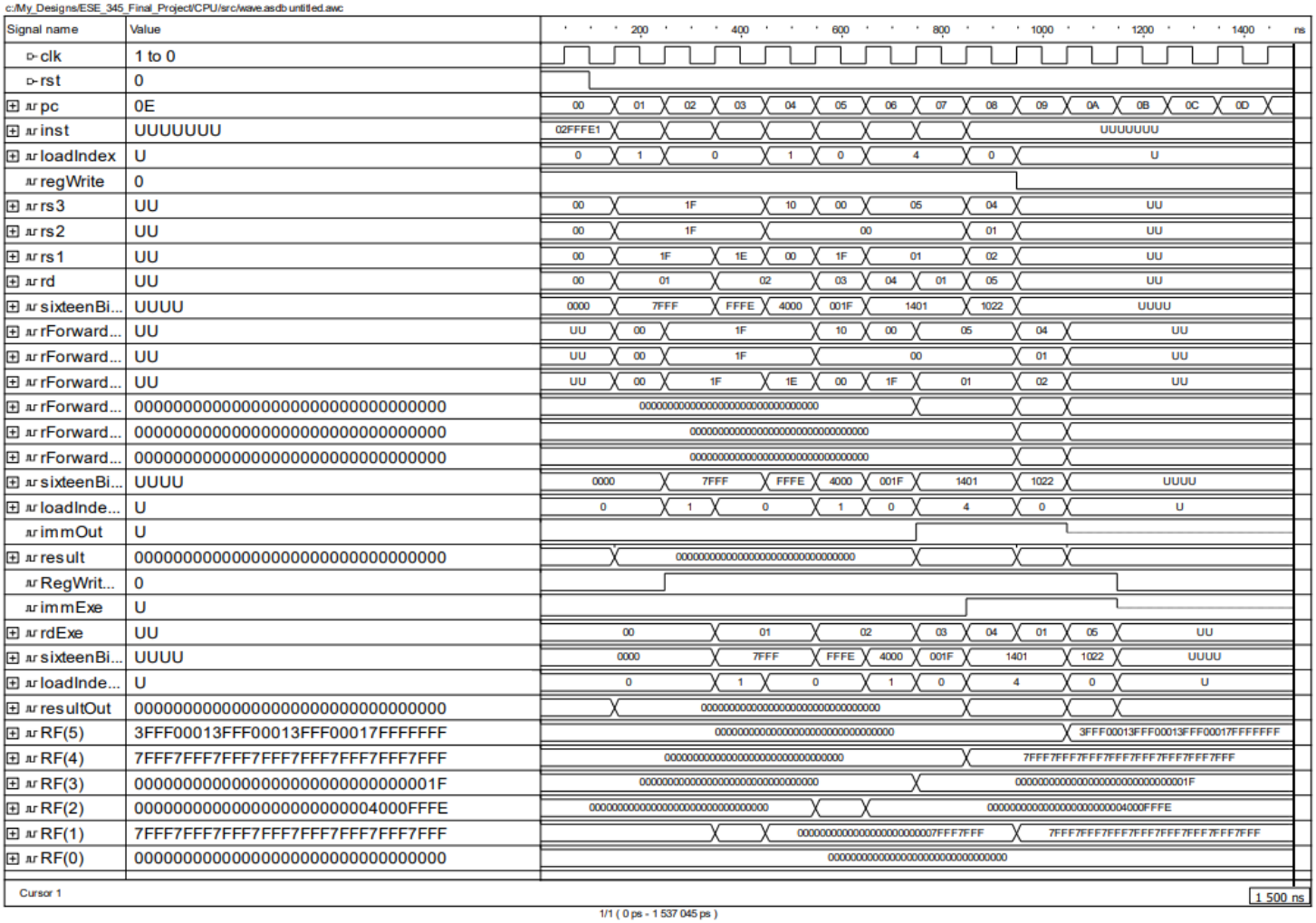
This project helped us apply the knowledge we've been learning this semester. We faced many challenges trying to create this multimedia unit but we successfully overcame them and learned a great deal along the way. It's fascinating to be there with every step of the design process observing how many little blocks start working together as a unit that allows us to do our instructions. Subword parallelism was an interesting challenge. It took some reading and researching to figure out how to implement our ALU in the best way possible.

Sample Instructions Inputted:
li 1 1 32767
li 1 0 32767
li 2 0 65534
li 2 1 16384
li 3 0 31
bcw 4 1 0
bcw 1 1 0
simal 5 2 1 4

The registers after these instructions should be:
REG31# 00000000000000000000000000000000
REG30# 00000000000000000000000000000000
REG29# 00000000000000000000000000000000
REG28# 00000000000000000000000000000000
REG27# 00000000000000000000000000000000
REG26# 00000000000000000000000000000000
REG25# 00000000000000000000000000000000
REG24# 00000000000000000000000000000000
REG23# 00000000000000000000000000000000
REG22# 00000000000000000000000000000000
REG21# 00000000000000000000000000000000
REG20# 00000000000000000000000000000000
REG19# 00000000000000000000000000000000
REG18# 00000000000000000000000000000000
REG17# 00000000000000000000000000000000
REG16# 00000000000000000000000000000000
REG15# 00000000000000000000000000000000
REG14# 00000000000000000000000000000000
REG13# 00000000000000000000000000000000
REG12# 00000000000000000000000000000000
REG11# 00000000000000000000000000000000
REG10# 00000000000000000000000000000000
REG09# 00000000000000000000000000000000
REG08# 00000000000000000000000000000000
REG07# 00000000000000000000000000000000
REG06# 00000000000000000000000000000000
REG05# 3FFF00013FFF00013FFF00017FFFFFFF
REG04# 7FFF7FFF7FFF7FFF7FFF7FFF7FFF7FFF
REG03# 0000000000000000000000000000001F
REG02# 0000000000000000000000004000FFFE
REG01# 7FFF7FFF7FFF7FFF7FFF7FFF7FFF7FFF
REG00# 00000000000000000000000000000000


clk
rst
Program Counter
PC
Instruction Buffer
Inst
clk
IF/ID buffer
Instruction
Instruction decode
control
(10 bits)
Control Unit
r1 (5 bits)
r2 (5 bits)
r3 (5 bits)
rd (5 bits)
Register File
Imm
regWrite
ALUcontrol
RType
reg1
reg2
reg3
clk
ID/EXE buffer
loadIndex
sixteenBitImm
RType
ALUopOut
rForwardingOut1 (5 bits)
rForwardingOut2 (5 bits)
rForwardingOut3 (5 bits)
rForwardingData1 (128 bits)
rForwardingData2 (128 bits)
rForwardingData3 (128 bits)
Forwarding
Unit
regALU1 (128 bits)
regALU2 (128 bits)
regALU3 (128 bits)
ALU
result (128 bits)
ImmOut
clk
EXE/WB buffer
loadIndexOut
RegWriteOut
regDestOut
sixteenBitImmOut
ImmExe
resultOut (128 bits)
loadIndexExe
RegWriteExe
rdExe
sixteenBitImmExe
sixteenBitImmExe
rdExe
RegWriteExe
loadIndexExe
resultOut (128 bits)
ImmExe

c:/My_Designs/ESE_345_Final_Project/CPU/src/wave.asdb untitled.awc



1/1 (0 ps - 1 537 045 ps)