CSE 331 PROJECT 3
MIPS LOAD-STORE
REPORT

In this project, we asked to write verilog project that make load and store operations on mips architecture. First of all we need to have those components:

- Program counter that gives instructions to the datapath.
- Control unit that controls the the data to be loaded or stored.
- Extenders for unsigned or signed for each common length.
- Register module, that reads the register to give them to datapath or writes register to given address.
- ALU(from previous project) calculates addresses with immediate values.
- Memory unit, that writes to given address with given word or read from memory.
- Muxes to select correct words to be loaded or writed.

Program Counter

In program counter I used a counter, that starts from zero. This counter makes increment of counter value in every posedge clock but processor designed to make one instruction in two cycle. This means in every second posedge, instruction is given to the datapath from program counter. Because of that I divided counter and modded with 2. To give printing I displayed instruction before new instruction come.

Control Unit

In control unit, I created an and gate for each instruction. This shows that for instruction opcode associated instruction is logic 1. We have several signals but for instructions we can group loads and stores since in deep they all do same operation. So, to write register we need a load operation. It can be any load operation. For AluSrc all instructions need to calculate correct memory address. This is common for all instructions. MemToReg decides whether readed value or calculated word from ALU goes to register. MemRead controls the writing to memory for load operations. MemWrite controls the reading operation for store operations. ALUops always will be 00 for addition. LuiCtrl controls is the immediate upper extended value is served to register or loaded value. LoadCtrl controls the is loaded ,raw, word is served to register or extended word is served to register. StoreCtrl controls the readed word is directly given to the memory or some extended word is given to the memory.

Extenders

We have different extenders for different purposes. Extend upper takes 16 bit immediate value and loads it to 32 bit word but it loads upper 16 bit part of 32 bit. After that if this is lui instruction, it loads this 32 bit value.

For SignExtends we have take word to be loaded or stored or immediate and then we need to extend its most significant bit to 32 bit. For immediate values to calculate memory addresses, we need to do take 16 bit and extend it with sign bit. For sb instruction, we need to extract least significant 8 bit of readed value. After that, we keep the this 8 bit but we extend it with eight bit of readed value. Since lb is signed operation. In homework document operations is given wrong. I talked this with professor and he said sing extend must extend most significant bit of loaded or stored value but we can't change this now, so we will accept both true. So I designed my system most significant bit extension base. Also for lb we must take least significant 8 bit and extend with 8^{th} bit. For lh sh, operation is same but this time we kept 16 bit. For lbu and lhu, extend bits will be 0.

Register Module

In register module, we initially read the register file and store it. We always reading read read registers. Only in posedge of clock we can make writing operation if signal_reg_write is 1. After every writing we update file.

ALU

In ALU, we only do addition operation to calculate address of words.

Memory Unit

In memory unit, we have two signals. MemRead will read the given address and take it the memory if signal implies that. MemWrite write given word if we have posedge clock. This is sync with register the write at same time. Before any writing or reading we initialize memory.

Muxes

We have different muxes. Their purpose is decide which value will be passed to next level. To write register we decide that is the loaded word from memory will be loaded to register or Load Immediate value. LuiCtrl decides that. We also have register output mux. Decides is extended value is passed to ALU or read_data 2 passed to ALU with help of AluSrc. We have 2x1 mux that decides stored value is sb or sh. They differ at 26^{th} bit of instruction. So we can choose the correct one but stored word can be the word to be loaded. So we again take 2x1 mux and decide that normal store word or extended store words will be loaded to memory with help of StoreCtrl. MemResult mux will decide that is readed word from memory will be loaded registers or value that calculated at ALU with help of memToReg. If we have load instruction, we need to load one word and we can process it later. For load instructions I loaded one word completely. I calculated extend values for all lb, lbu, lh, lhu. They have a pattern when we consider opcodes. They have 00, 01, 10, 11 different bits. We can select between them with 4x1 mux. Instruction's 26^{th} and 28^{th} bits have this pattern. After selecting correct instruction result we also need to decide we want to make load word or extended load word operation. With 2x1 mux we can decide which value we want to write to register with help of LoadCtrl.

Clock

For one instruction to complete its task takes 20ps. This means my clock cycle will be 20ps for one instruction. For testing if you want to see every instruction one by one you need to do timescale of one run 20 ps.

Registera load edilen data: 00000000000000000000000000000001
Instruction: 10000000011010000000000000000000
Result: 11111111111111110000111100000001
Registera load edilen data: 00000000000000000000000000000001
run
Memorye store edilen data: 00000000000000000000000000000000
Instruction: 10100000011010100000000000111111
Result: 00000000000000000000000001000010
Memorye store edilen data: 00000000000000000000000000000000
run
Registera load edilen data: 00000000000000000000000000000100
Instruction: 10000000010010010000000000000010
Result: 00000000000000000000000000000100
Registera load edilen data: 00000000000000000000000000000100
run
Memorye store edilen data: 00000000000000000000000000000000
Instruction: 10100000001010110000000000111111
Result: 00000000000000000000000001000000
Memorye store edilen data: 00000000000000000000000000000000
run
Registera load edilen data: 00000000000000000000000000010100
Instruction: 10010000000010000000000000010100
Result: 11111111111000000000000000010100
Registera load edilen data: 00000000000000000000000000010100
run
Memorye store edilen data: 00000000000000000111111111010010
Instruction: 10100100010100100000000011110110
Result: 00000000000000000000000011111000
Memorye store edilen data: 00000000000000000111111111010010
run
Registera load edilen data: 00000000000000000000000000000001
Instruction: 10010000001001010000000000000000
Result: 11111111111111110000111100000001
Registera load edilen data: 00000000000000000000000000000001
run
Memorye store edilen data: 11111111111111111000000000000110
Instruction: 10100100011001100000000011101101
Result: 00000000000000000000000011110000
Memorye store edilen data: 11111111111111111000000000000110
run
Registera load edilen data: 00000000000000000000000000010011
Instruction: 10000100011101000000000000010000
Result: 00011111100000000000000000010011
Registera load edilen data: 00000000000000000000000000010011
run
Instruction: 10101100000011110000000011011000
Memorye store edilen data: 00000000000000000000000001100000

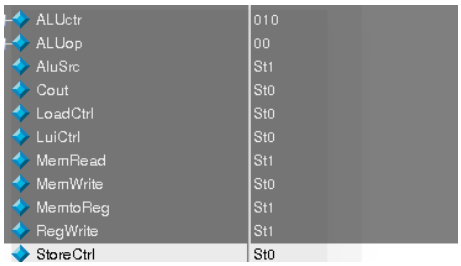
```
# Result: 00000000000000000000000011011000
# Memorye store edilen data: 00000000000000000000000001100000
run
# Instruction: 10000100010101010000000000010110
# Registera load edilen data: 11111111111111111000000000011000
# Result: 11111111110000001000000000011000
# Registera load edilen data: 11111111111111111000000000011000
run
# Instruction: 10101100001100000000000011011000
# Memorye store edilen data: 00111111110000000000000000010000
# Result: 00000000000000000000000011011001
# Memorye store edilen data: 00111111110000000000000000010000
run
# Instruction: 10010100001101100000000000011001
# Registera load edilen data: 00000000000000000000000000011010
# Result: 00000000000000000000000000011010
# Registera load edilen data: 00000000000000000000000000011010
run
# Instruction: 10010100011101110000000000011101
# Registera load edilen data: 00000000000000000000000000000000
# Result: 11111110000000000000000000000000
# Registera load edilen data: 00000000000000000000000000000000
run
# Instruction: 00111100000111100000000000000001
# Registera load edilen data: 00000000000000010000000000000000
# Result: 00000000000000010000000000000000
# Registera load edilen data: 00000000000000010000000000000000
run
# Instruction: 00111100000111111111111111111111
# Registera load edilen data: 11111111111111110000000000000000
# Result: 11111111111111110000000000000000
# Registera load edilen data: 11111111111111110000000000000000
run
# Instruction: 10001100000111010000000000110000
# Registera load edilen data: 00000000000000000000000000010000
# Result: 00000000000000000000000000010000
# Registera load edilen data: 00000000000000000000000000010000
run
# Instruction: 10001100000111000000000000110001
# Registera load edilen data: 00000000000000000000000000010001
# Result: 00000000000000000000000000010001
# Registera load edilen data: 00000000000000000000000000010001
```

We have these outputs from 18 instruction in given instr.txt.

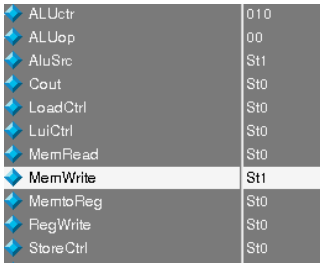
Control Signals

LB

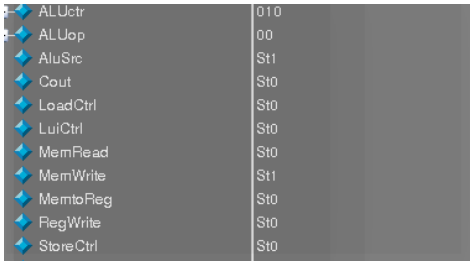
LBU



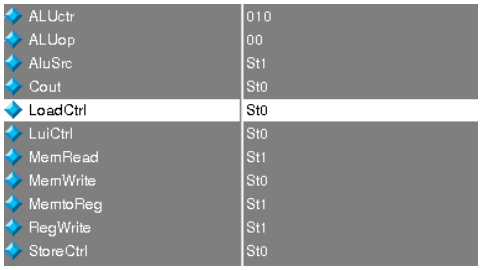
SB



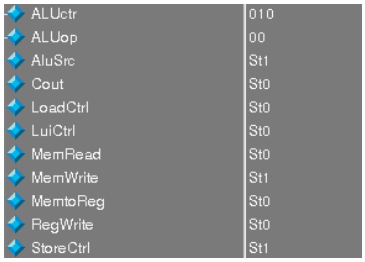
SH



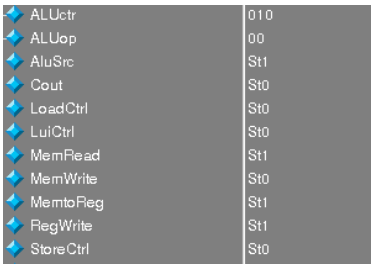
LH



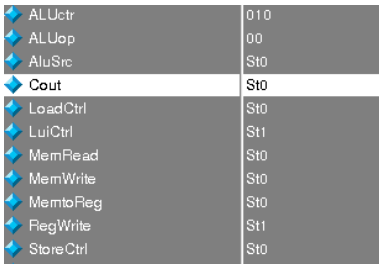
SW



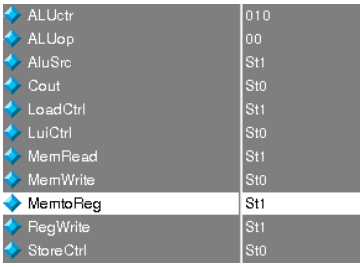
LHU



LUI

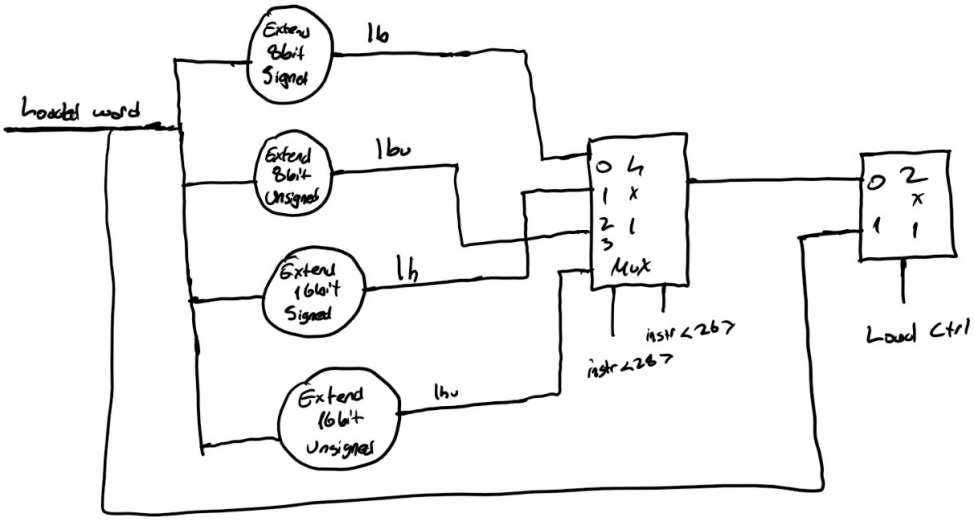


LW



Logic Behind Store and Load Operations

LOAD



STORE