**Lab 4B - RFALU with Data Memory (RFALUDM) - Load Operation:**

When a computer boots up, it will load operating system and instructions from the nonvolatile hard drive to the Instruction segment of the Memory. It may load some constants and data in the Data section of the memory. In this lab, we have not built the Instruction Memory yet. So we will supply the necessary instruction bits. We will imagine that some constants are in the Data Memory. So we will call the Data Memory module with the same *initialization* of the memory content as in Part A i.e., DM[40] = 8’h55 etc. and integrate it with Lab 3C – RFALU to make RFALUDM.

We must first 'load' two values 555... and aaa… after picking them from Data Memory to Registers 5 and 10. The difference between Lab 3C and 4B is that instead of writing 555 and aaa to RF initially as a starting point, here we actually load them *from Data Memory*. Being nonvolatile, they have the values 555 and aaa. (In the simulation, we achieved this by initializing the DM appropriately. The D type instructions use *based displacement addressing*. The base pointer is given by the content of register address given by the Rn field (9-5) and the displacement is given by the 9 bits (20-12) in the instruction. See the format below.

Rt (Destination Address for Load **o**r Source Address for Store

Displacement

Rn (Base)

Opcode

11

5

5

2

unused

9

We have to add the base and the displacement and *if we use* the ALU for that, the two operands entering the ALU must be matched to have the same size. A input to ALU will be 64 bit, the content of the register pointed to by Rn field and the B input for displacement to be added must be sign extended to be 64 bits. Sign Extender module can be implemented with one line (after the usual in and out declarations) as

“assign SEout = {{55{SEin[9]}}, SEin};” //This simply concatenates the most significant bit of the 9 bit displacement field SEin to the left 55 times to give 64 bits double word SEout.

What more, we will need a *multiplexer* in front of the B input of the ALU. For R Type, B must be register value read out by the Rm field but

for D Type, it should use the displacement. We have to use a 64 bit 2 to 1 mux. We need to modify the mux developed in Lab 1 suitably. The text

refers to this mux for ALU input B as "ALUSrc".

ALUSrc

5

Rn

Rm

ALU result

DM

Address

Rt

B

Write register

64

Write Data

SE

64

9

The output of the ALU will become the Effective Address for the Data Memory. We have to pick up the memory operand and connect it to Write data input of the Register File. We have to supply the Write address for the Register File from the Rt field of the instruction. See above.

Are we done yet? No. For R Type instruction, we had the ALU result come back to the Register File and get written into register pointed to by the Rd field (4-0) – same as Rt field (4-0). So the line coming to the Write data input to the RF must be either the ALU result or data read from Data Memory. This means we need a 64 bit 2 to 1 mux again (Here we go again). In the diagram below, this mux is placed next to DM and the ALU result comes as an alternate input and the output of the mux instead of going on the top of the picture comes at the bottom to reach the register write data input. (The reason for this connecting line being shown at the bottom is that more connections will come about on the top once we incorporate the B Type instruction).

Give the appropriate values in the test fixture module (do not forget mux selector values, clock, MemRead and RegWrite control signals) and see whether Load operation works. As before, Registers 5 and 10 will be loaded first from Data Memory (2 clock edges). To load register 5 from memory location 40 decimal, we must have Rn content as zero and 9 bit displacement as 9’b 0 0010 1000. Remember we have only positive values for displacement and they have to be aligned at 8 Byte boundaries for a total of 256 B (32 Double Words). To load register 10 from memory location 80 decimal, we must have Rn content as zero and 9 bit displacement as 9'b 0 0101 0000. Then the 4 ALU operations will be performed. Let the results of ALU operations be written in 4 different registers, say R1, R2, R3, and R4 (4 more clock edges). Remember many connectors will become wires.

MemRead

Rn

ALUSrc

MemtoReg

Mem. Address

5

Zero

A

5

64

DM

RF

Rt

ALU result

B

Write register (Address)

Write Data

4

ALU operation

from Opcode field and ALUOp

64

Write Data

clock

RegWrite

clock

MemWrite

SE

9

64

**Part 4C: RFALU with Data Memory (RFALUDM) – Store operations**

We have to pick up the register value given by the Rt field in the instruction and deliver it as Write data to the Data Memory. For the R type instruction, Read Register addresses 1 and 2 were Rn (9-5) and Rm (20-16) (Figure 4.14 (p.274)) to pick the two source operands. Now we have to pick the register value given by the *Rt* field (4-0). Let us say that we choose Read data **2** to transport it, then we have the following problem. When the instruction is R type, Read Register Address 2 must be Rm and when it is D type Store operation, it must be Rt. This necessitates a 5 bit 2 to 1 Multiplexer before the Read Register Address 2 connection as shown below: The text calls this mux as "Reg2Loc". Also the Read out data 2 must become Write data for Data Memory, *skipping the ALU* as shown.

Rn

Rm

DM

RF

Rt

Read Register Address 2

Write data

MemWrite

clock

The address for storing the operand must be the effective address calculated by ALU after adding the base and the sign extended displacement as done for Load operation.

DM

Address

B

9

64

SE

The connection diagram will be as follows:

MemRead

Mem. Address

Rn

5

Zero

A

5

Rt

Rm

64

DM

RF

Write register (Address)

ALU result

B

Write Data

4

ALU operation

from Opcode field and ALUOp

64

Write Data

RegWrite

clock

clock

MemWrite

SE

9

64

Let us check **storing**. Let us store the results R1 and R2 in Lab 3C of adding and subtracting 555… and aaa… in Memory locations DM[0] to DM[7] and DM[8] to DM[15]. They were probably initialized as 0 in the Memory module beforehand. Change the different fields of the instruction to effect storing. MemWrite will be the additional input stimulus we need to specify (in addition to clock) for DM.

To conclude, we have **added four more components** (a sign extender and three muxes) **and three more control signals** for selecting the correct input in those muxes in the new figure. The four components are - Component # 1 - a Sign Extender to make the displacement field in the D Type instruction to a 64 bit number and Component # 2 - a two to one 64 bit multiplexer for the ALU Source B and Component # 3 - a two to one 64 bit multiplexer for Register Write data from either Data Memory or from ALU result and Component # 4 - a two to one 5 bit mux to select Rm or Rt as the address to select the regidter content for ALU or for storing the register content in teh Data Mrmory. **See Fig. 4.15 in p.275** (or Fig. 4.17 in p.277). The three additional input stimuli will be the three mux select signals shown in the table below with their choices. These are shown clearly in the figures and also in Table 4.16 on p.276.

|  |  |  |
| --- | --- | --- |
| **Name of Control Signal** | **If 0** | **If 1** |
| Reg2Loc | Rm field | Rt field |
| ALUSrc | Register Read Data 2 | Sign Extender Output |
| MemtoReg | Register Write Data is ALU Output | Register Write Data is Data Memory Read Data |

.

Looking ahead, we have to build the Control Unit and then combine it with RFALUDM to finish the simple Single (long) Cycle non-pipelined Computer. We will be able to execute all instructions except the Branch instruction.

mc, f17