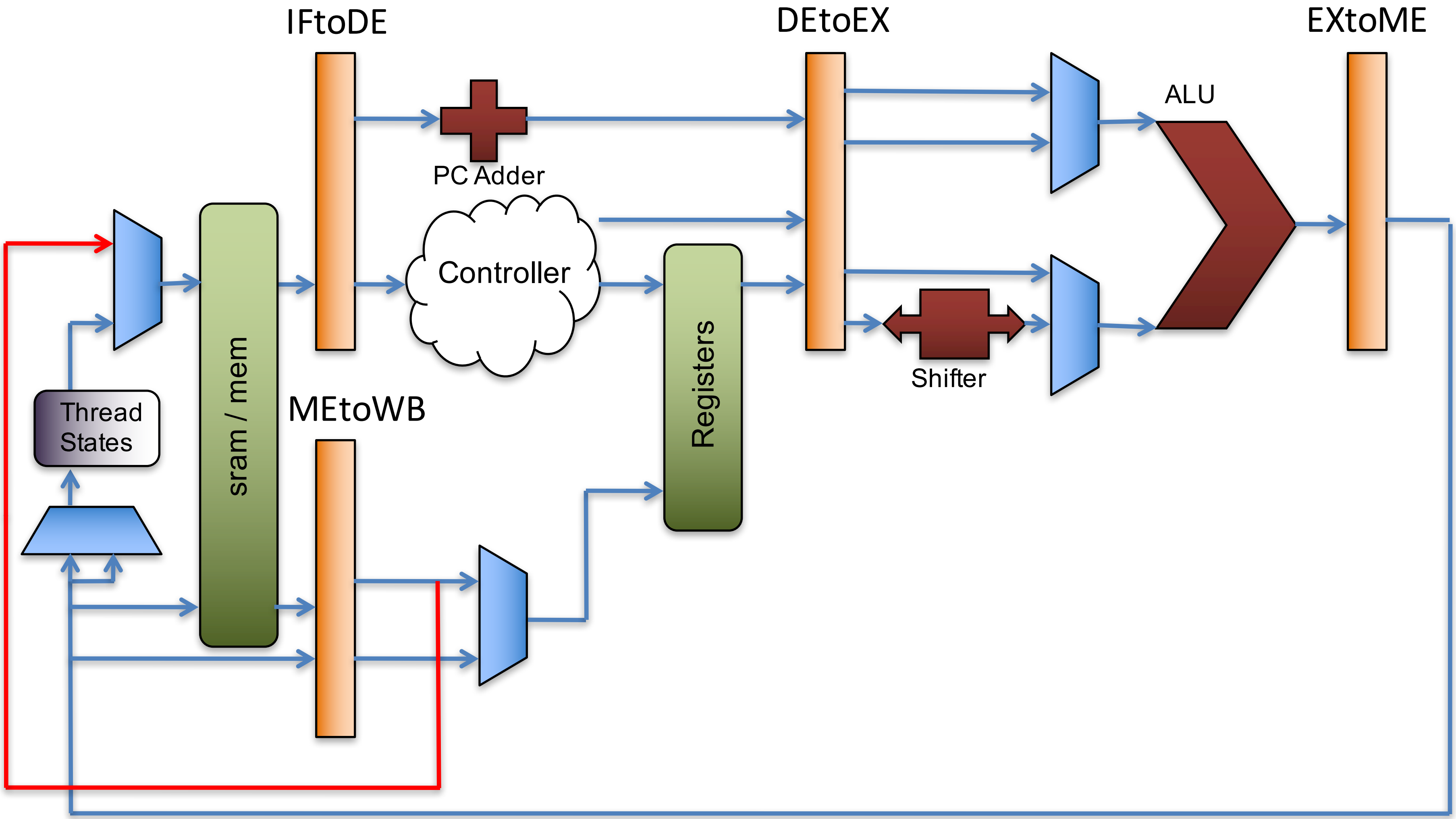
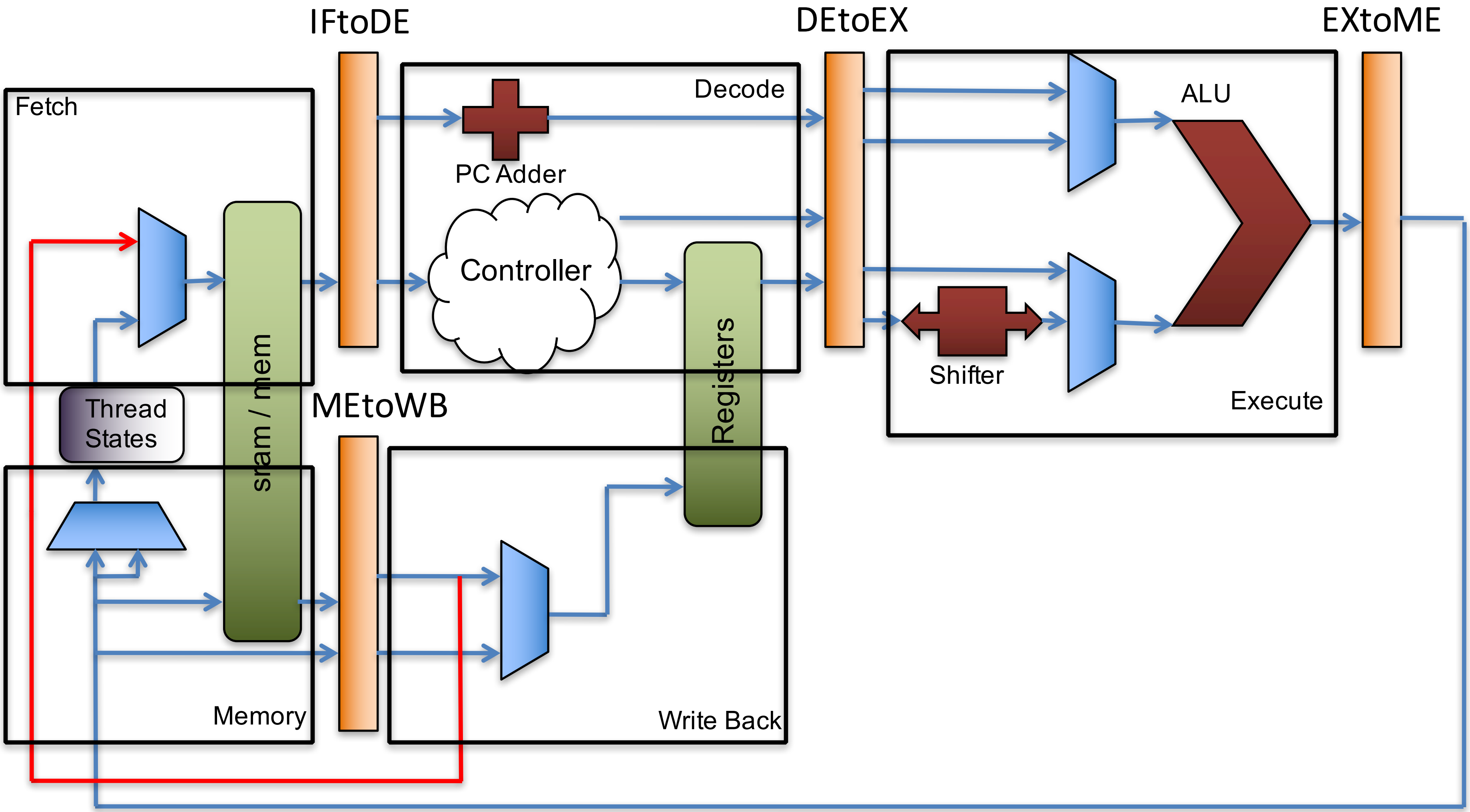
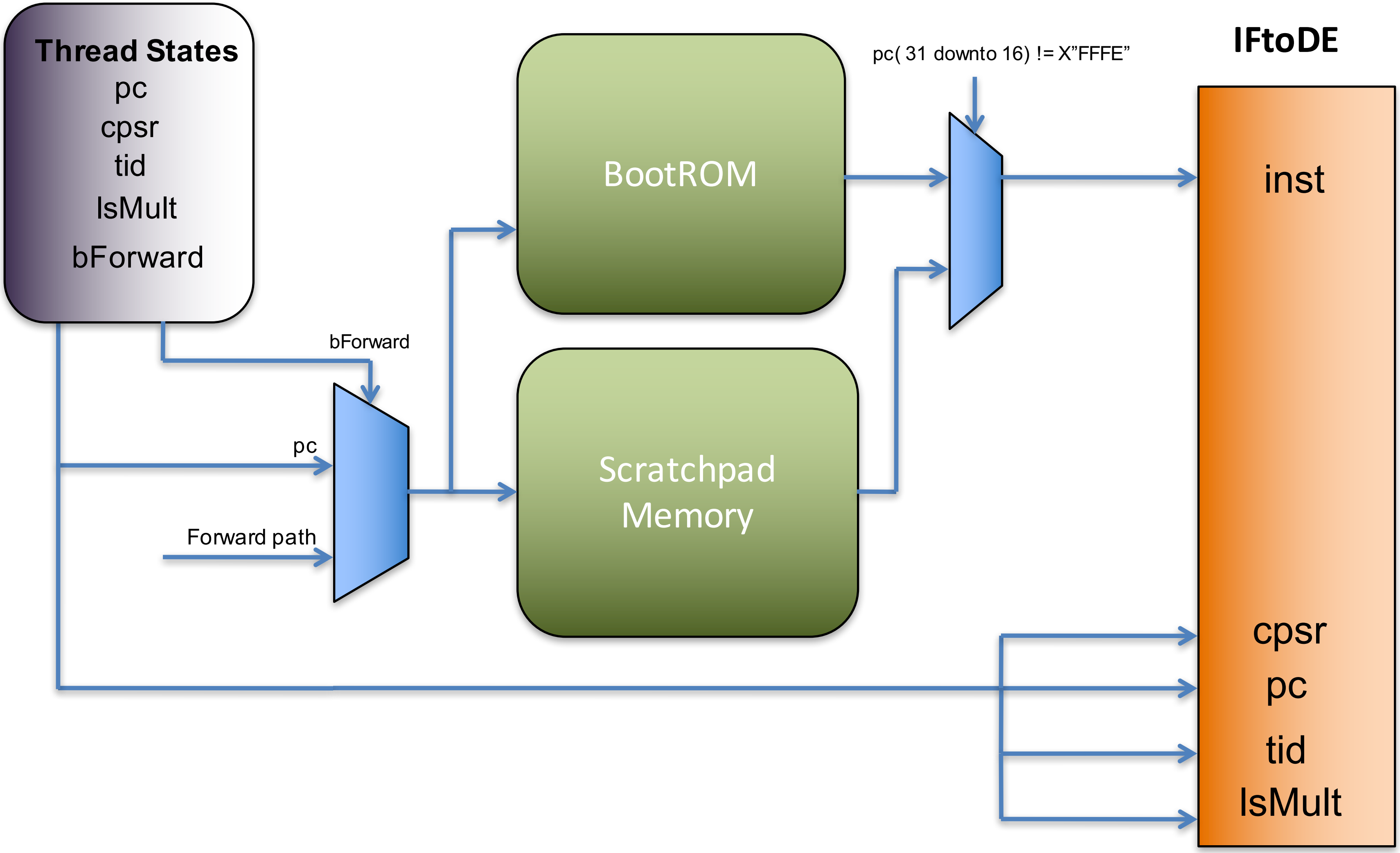
Ptarm 5 stage pipeline



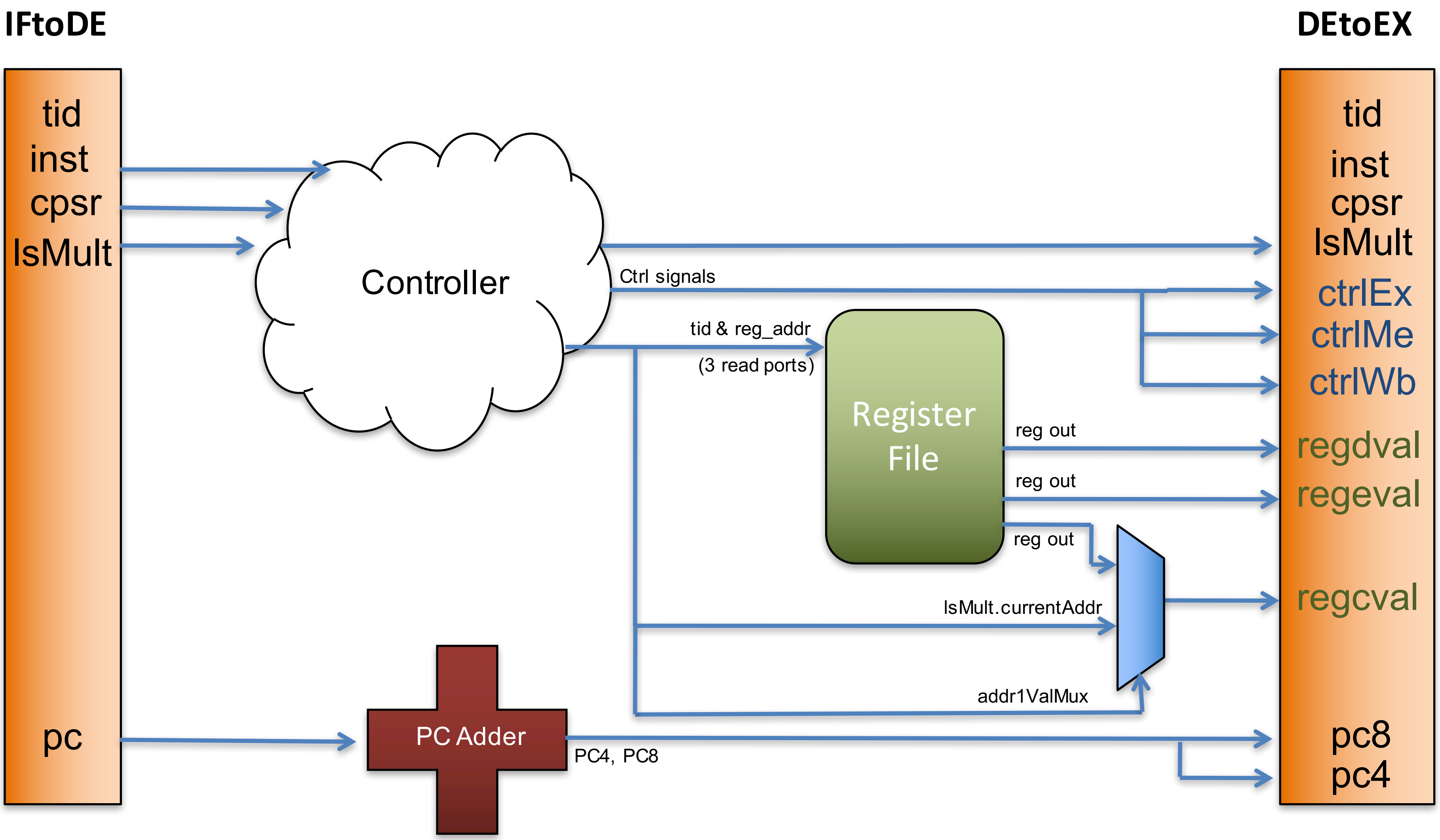
Ptarm 5 stage pipeline



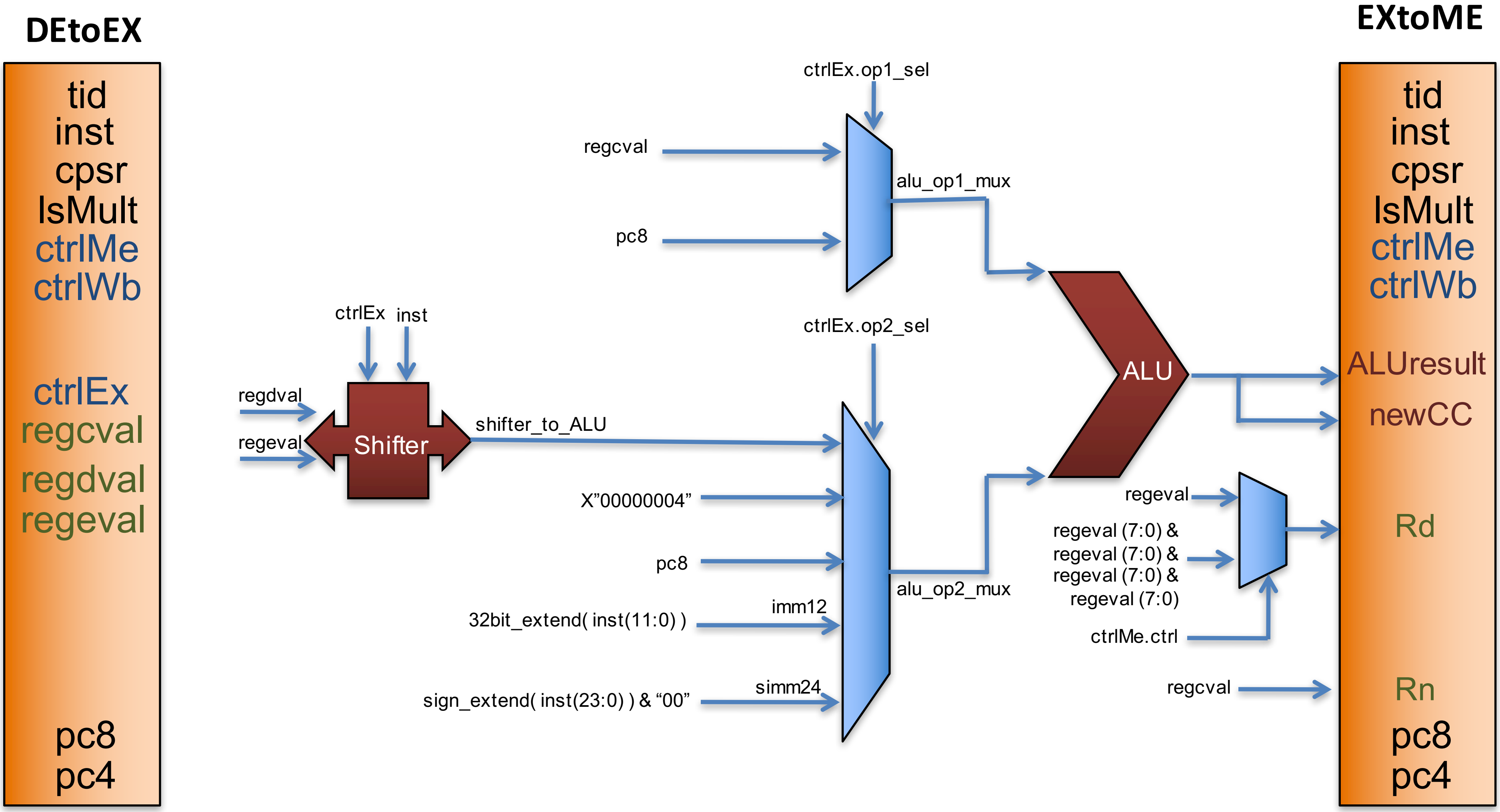
Fetch Stage


Thread States
pc
cpsr
tid
IsMult
bForward
bForward
pc
Forward path
BootROM
Scratchpad Memory
pc(31 downto 16) != X”FFFE”
IFtoDE
inst
cpsr
pc
tid
IsMult

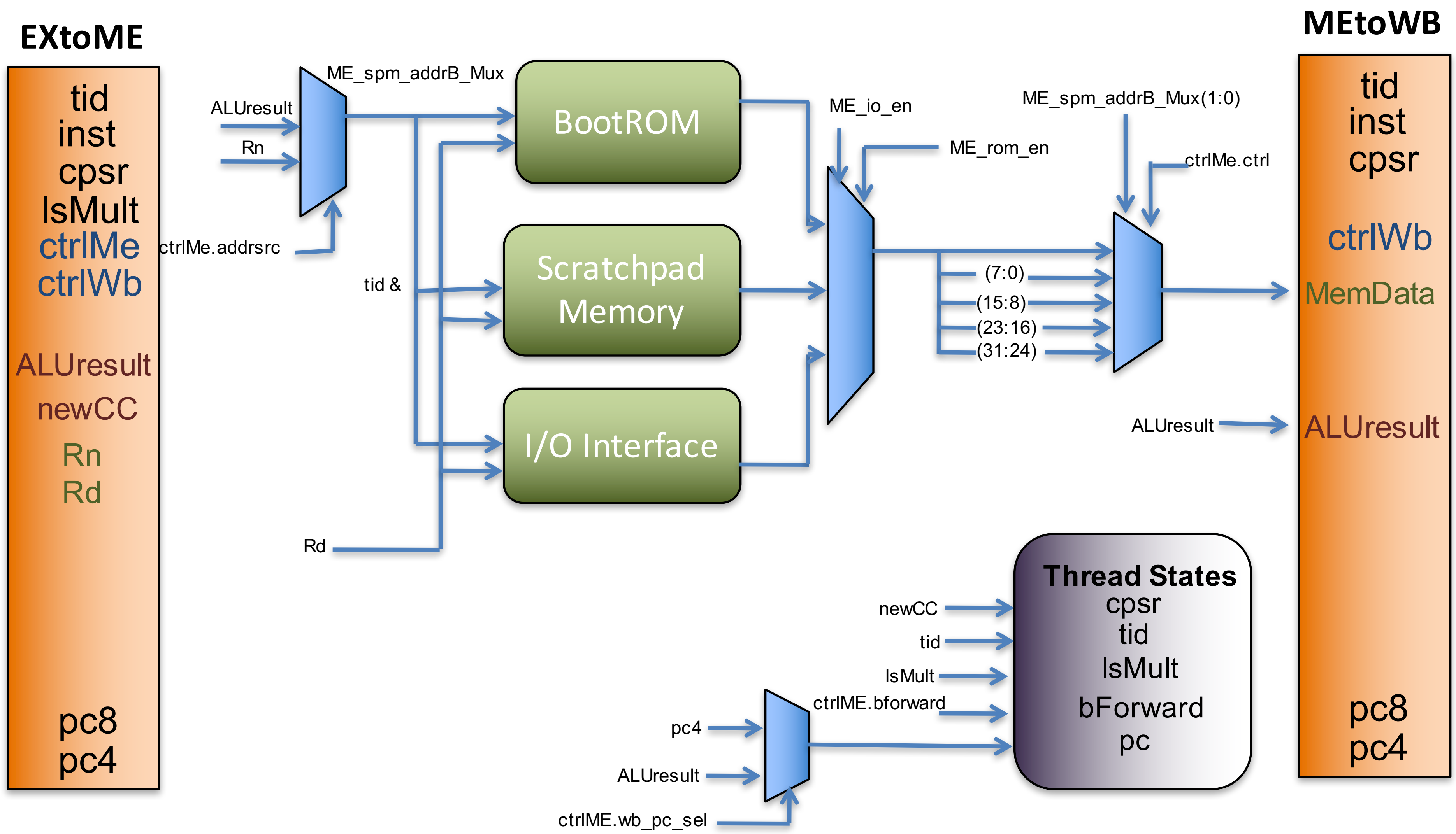
Decode Stage



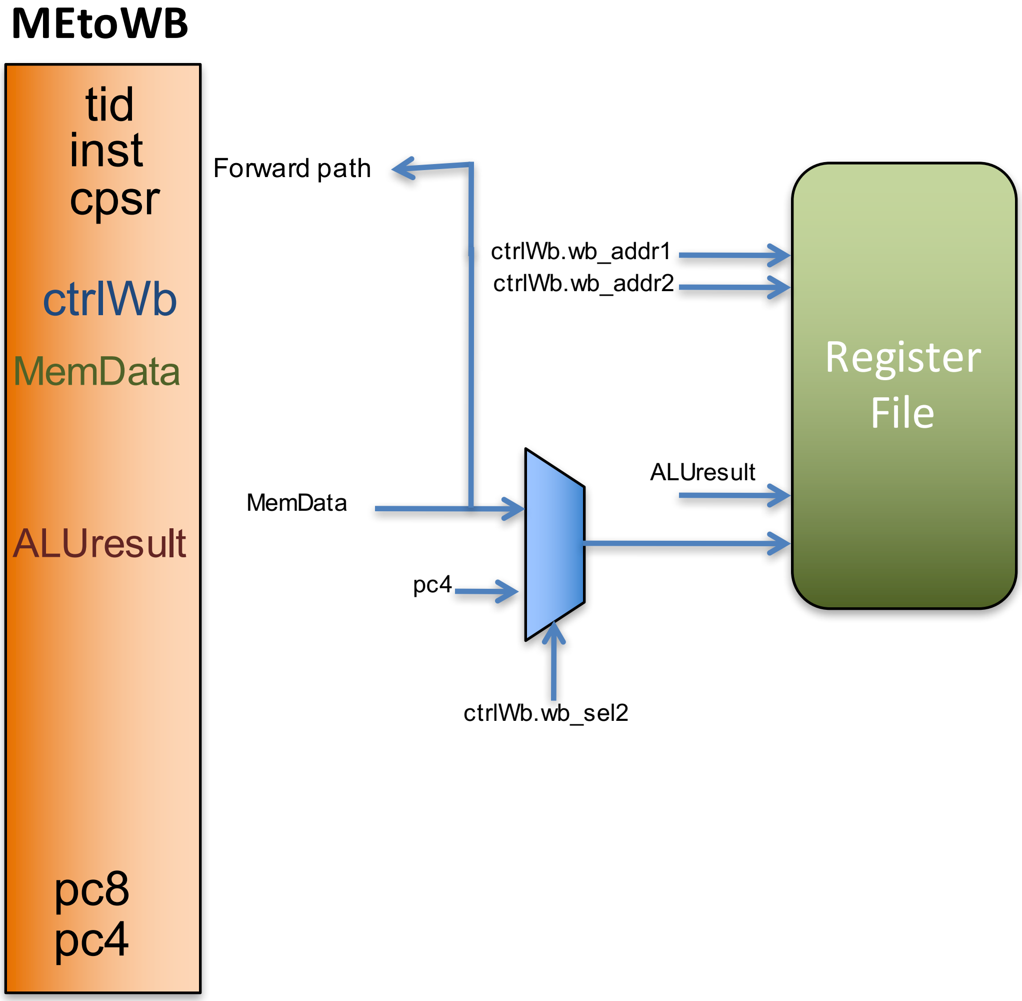
Execute Stage



Memory Stage



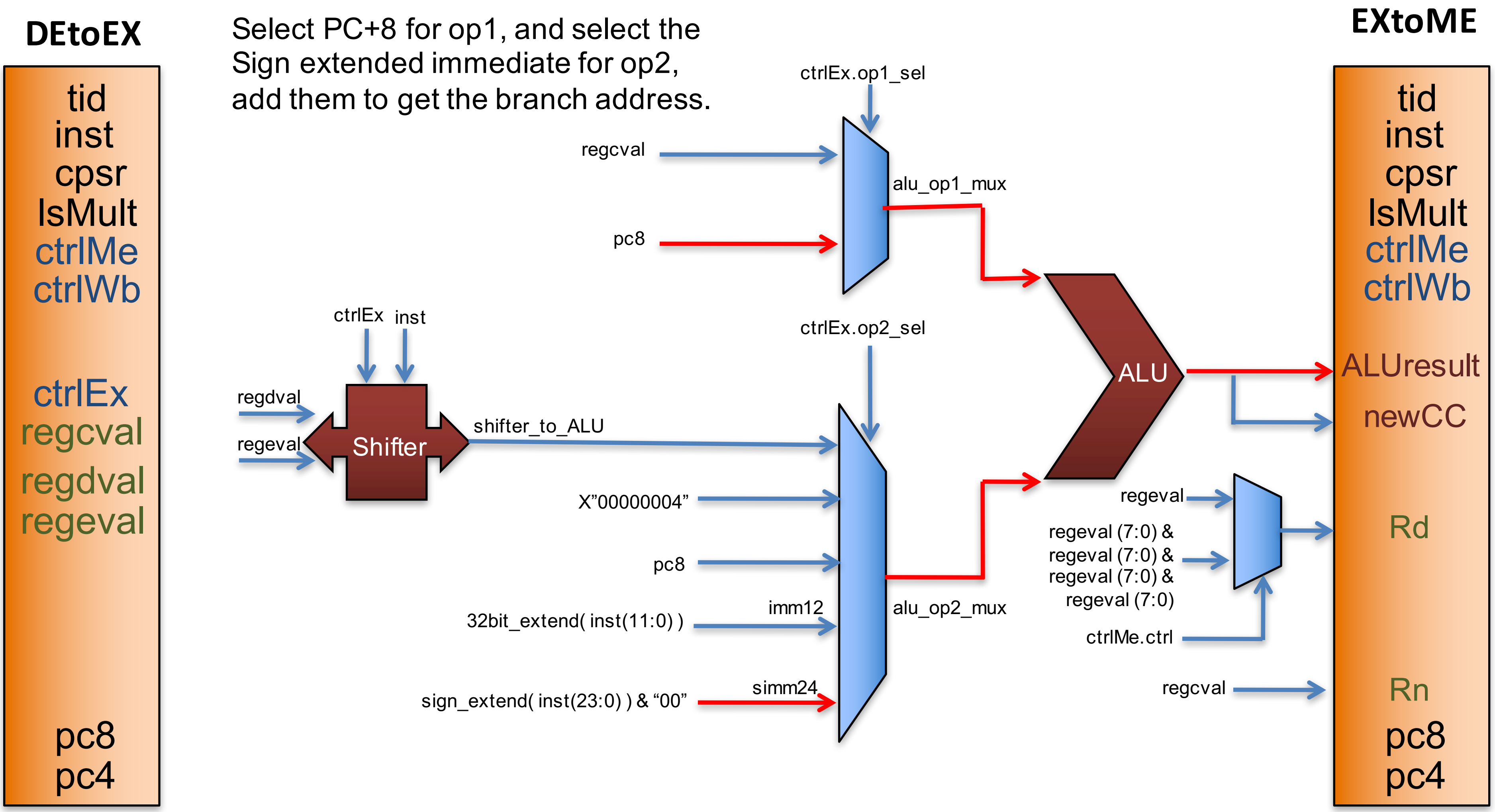
Write Back Stage



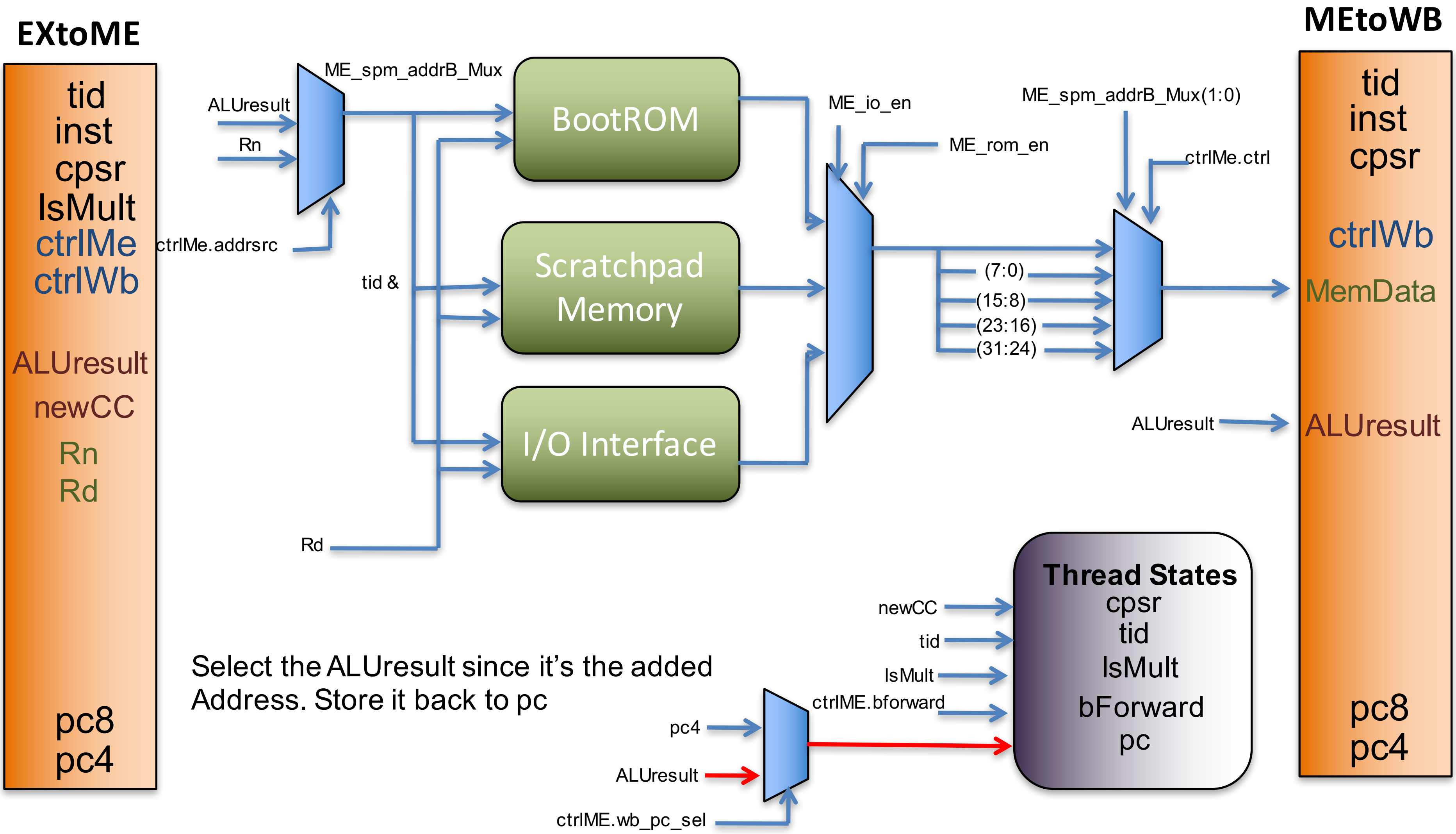
Branch/Branch Link Instructions

Execute Stage

Select PC+8 for op1, and select the Sign extended immediate for op2, add them to get the branch address.

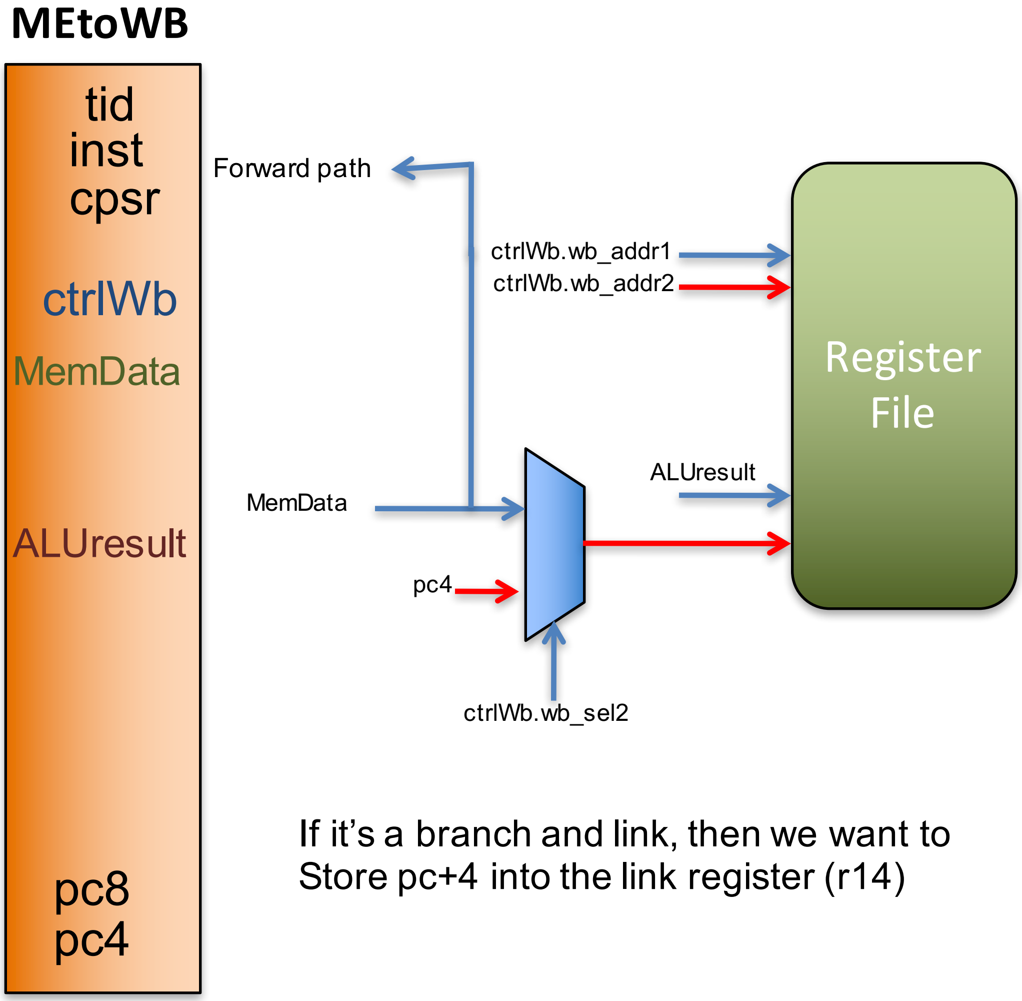


Memory Stage



Select the ALUresult since it's the added Address. Store it back to pc

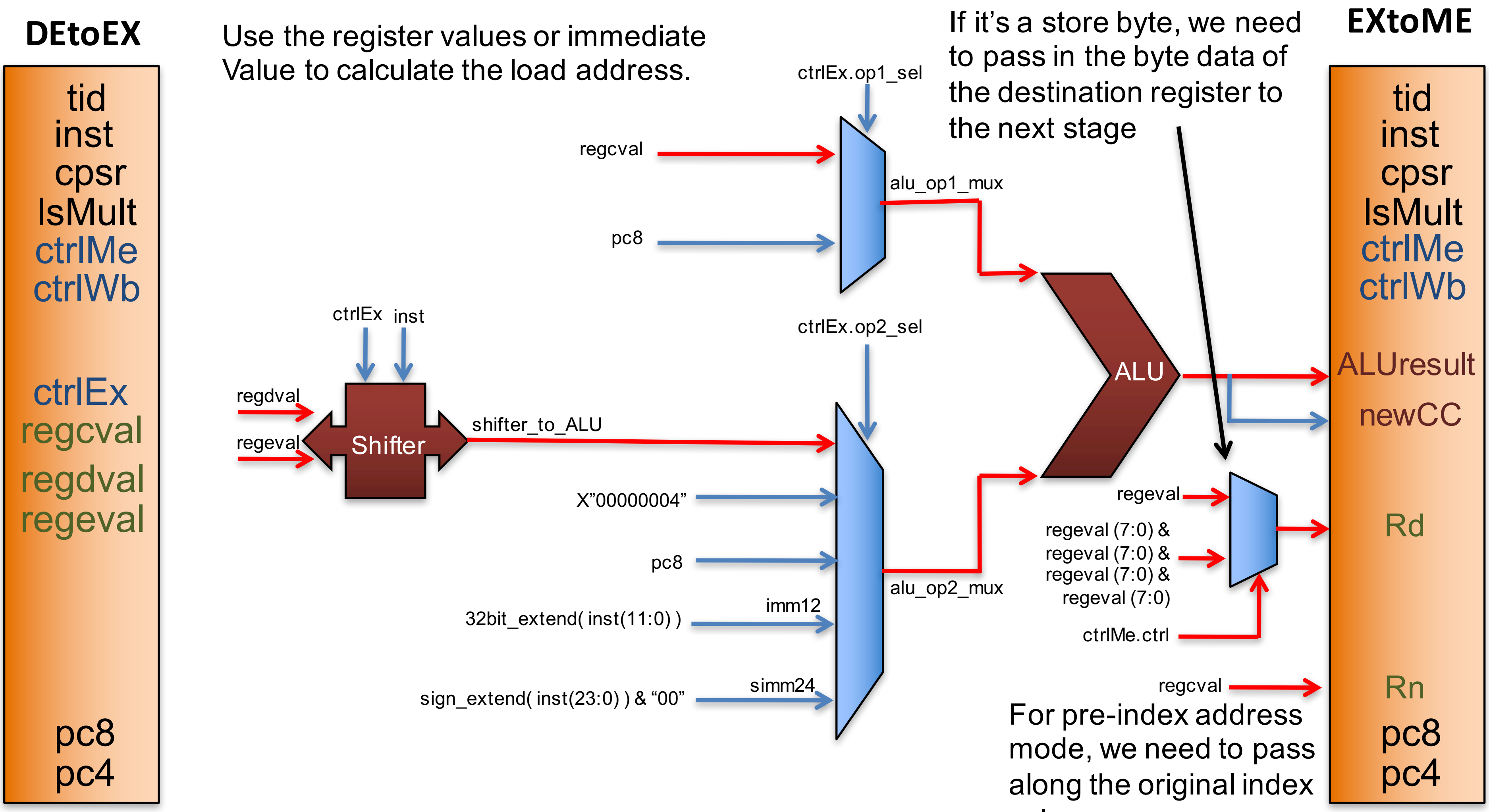
Write Back Stage



If it’s a branch and link, then we want to Store pc+4 into the link register (r14)

Load/Store Instructions

Execute Stage

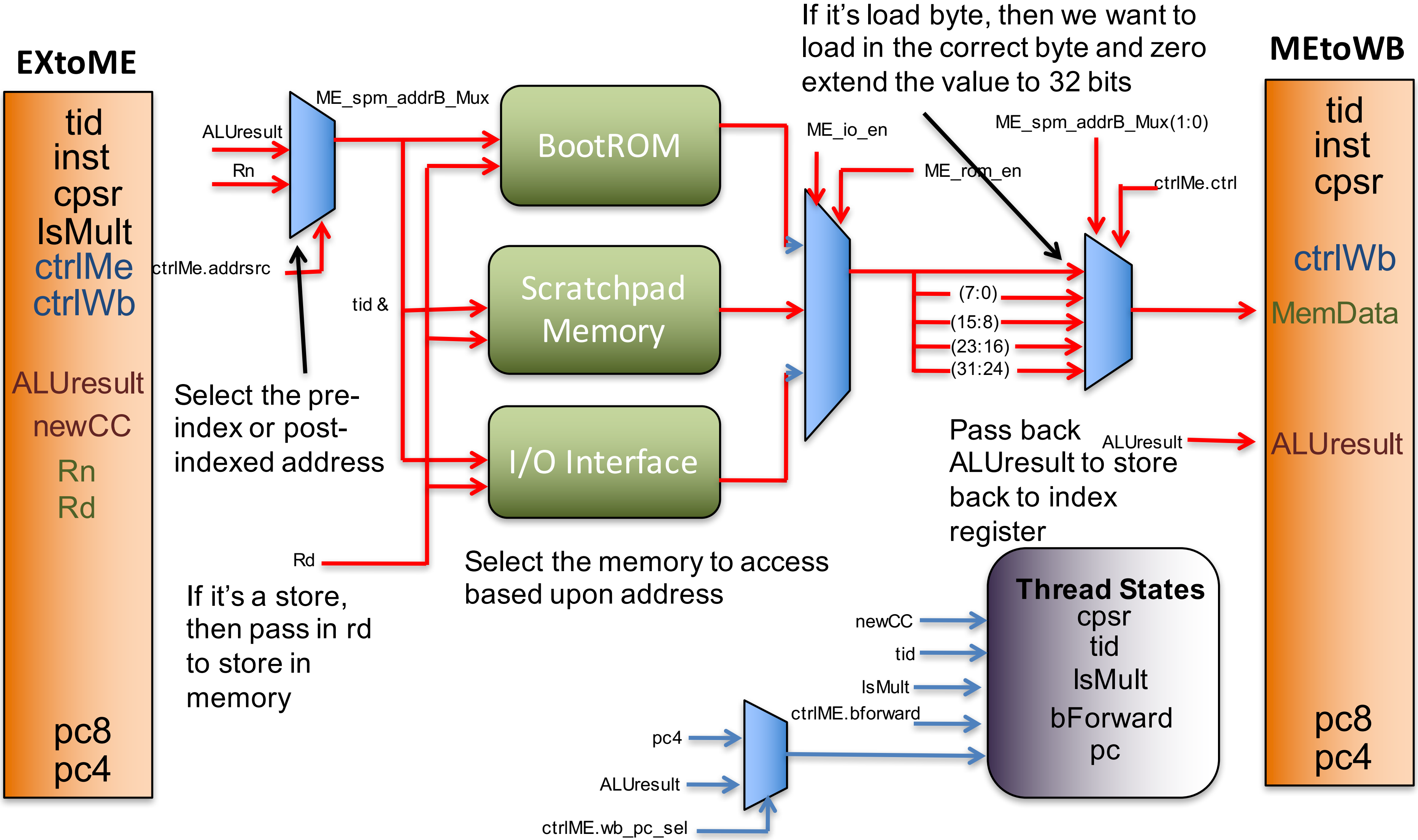


Use the register values or immediate Value to calculate the load address.

If it's a store byte, we need to pass in the byte data of the destination register to the next stage

For pre-index address mode, we need to pass along the original index value

Memory Stage



If it's load byte, then we want to load in the correct byte and zero extend the value to 32 bits

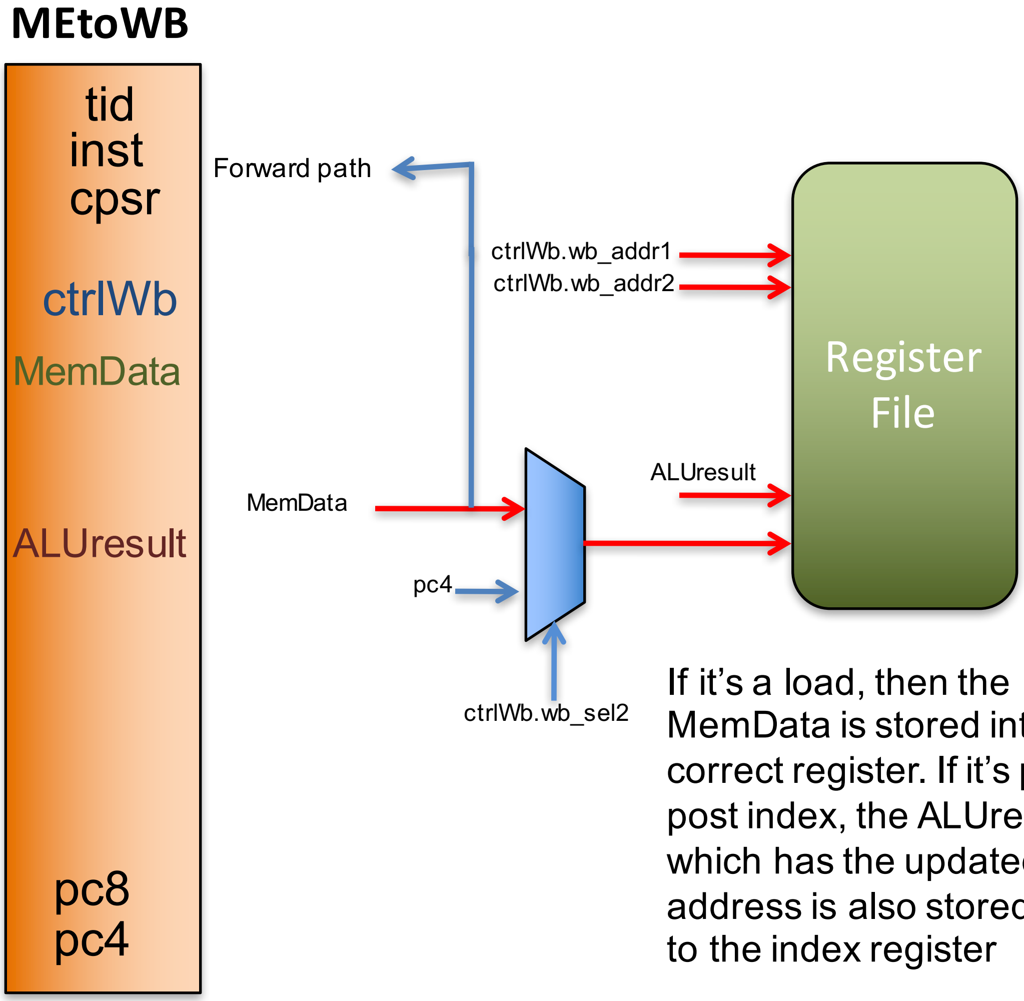
Select the pre-index or post-indexed address

Select the memory to access based upon address

If it's a store, then pass in rd to store in memory

Pass back ALUresult to store back to index register

Write Back Stage



MEtoWB

tid
inst
cpsr

ctrlWb

MemData

ALUresult

pc8
pc4

Forward path

ctrlWb.wb_addr1

ctrlWb.wb_addr2

Register File

ALUresult

MemData

pc4

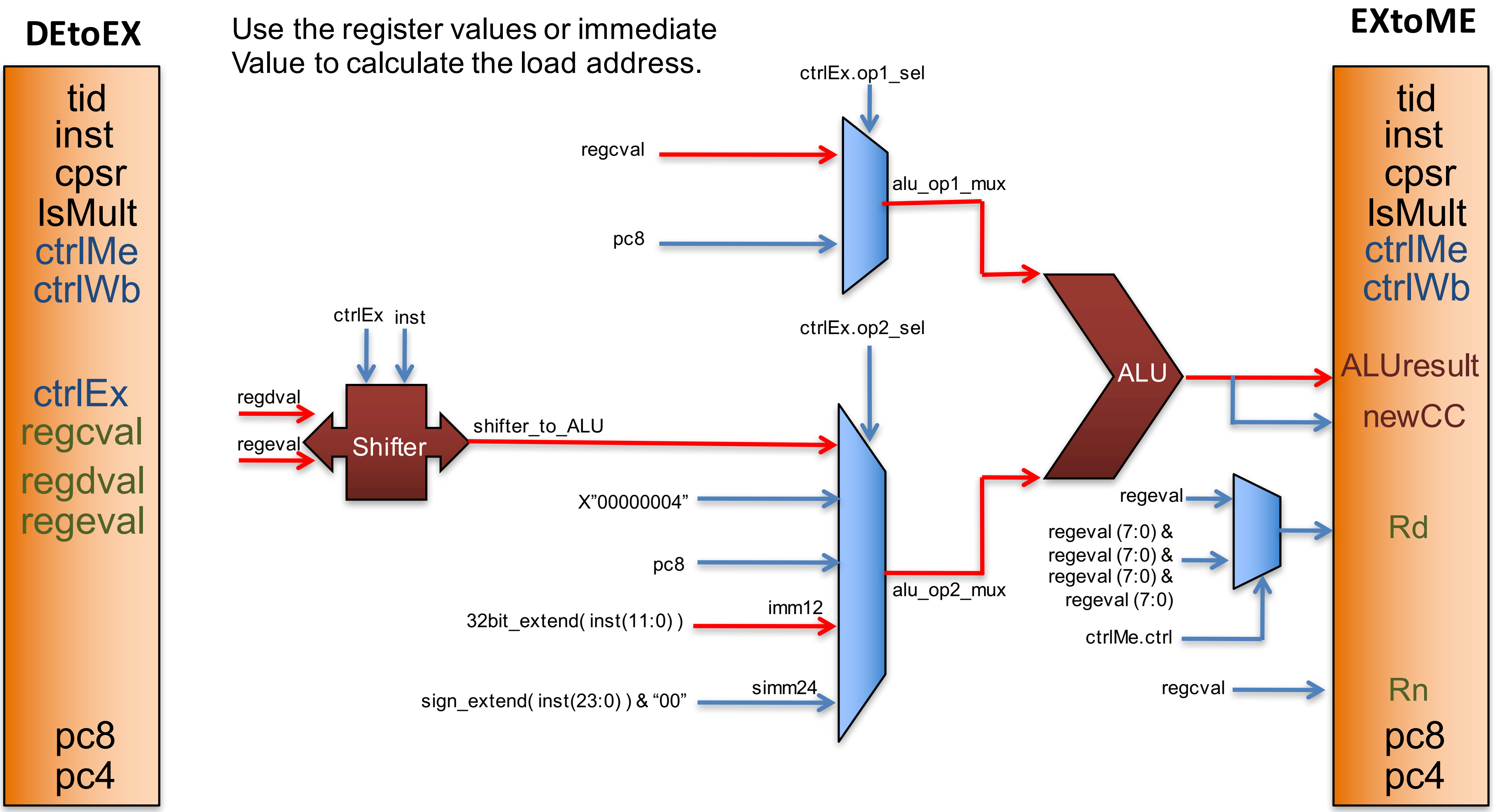
ctrlWb.wb_sel2

If it's a load, then the MemData is stored into the correct register. If it's pre or post index, the ALUresult, which has the updated address is also stored back to the index register

Load to R15 Instruction

Execute Stage

Use the register values or immediate Value to calculate the load address.



DEtoEX

tid
inst
cpsr
IsMult
ctrlMe
ctrlWb
ctrlEx
regcval
regdval
regeval
pc8
pc4

ctrlEx inst

regdval
regeval

Shifter

shifter_to_ALU

ctrlEx.op1_sel

regcval
pc8

alu_op1_mux

ctrlEx.op2_sel

X"00000004"
pc8
imm12
32bit_extend(inst(11:0))
simm24
sign_extend(inst(23:0)) & "00"

alu_op2_mux

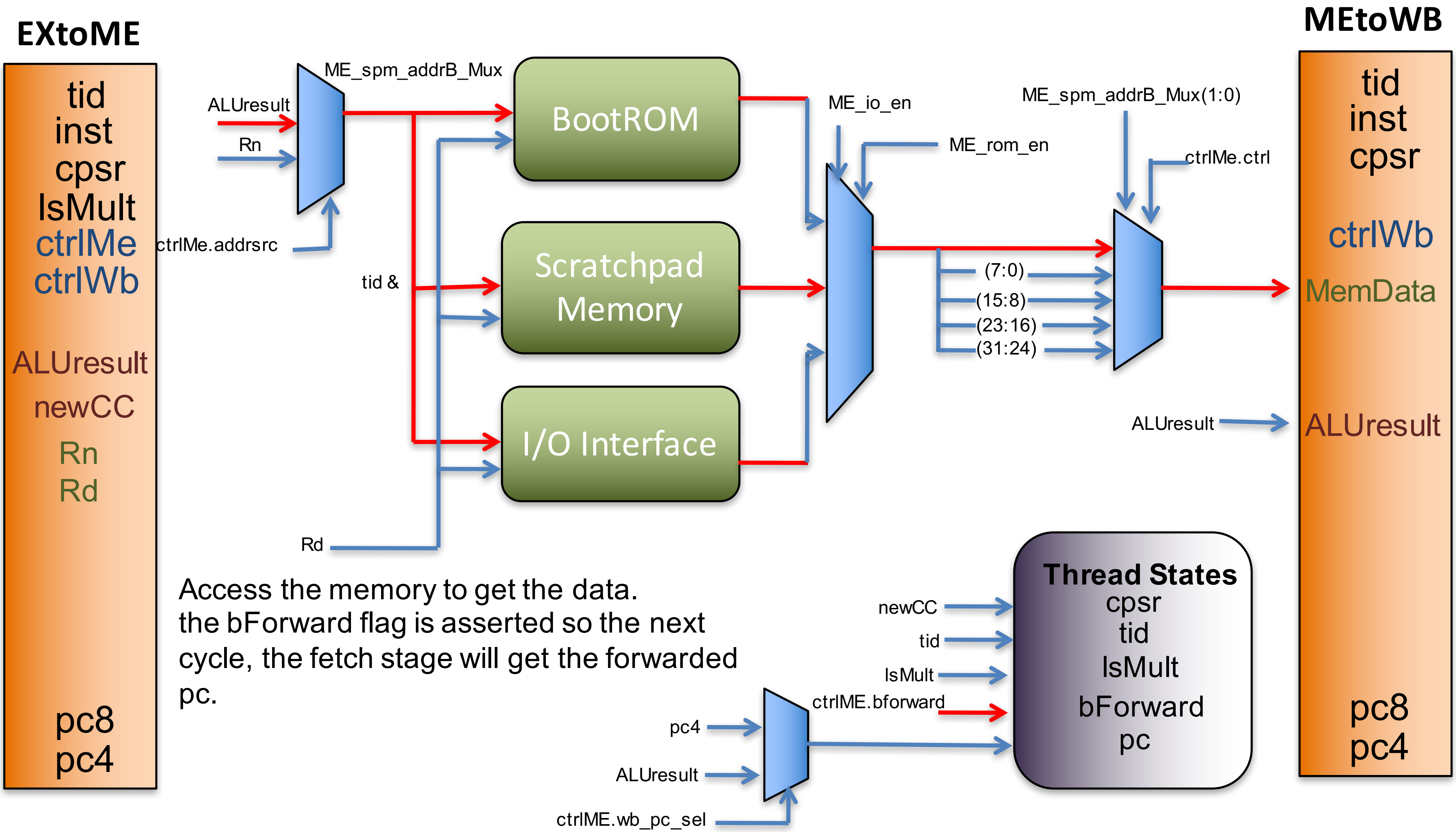
ALU

regeval
regeval (7:0) &
regeval (7:0) &
regeval (7:0) &
regeval (7:0)
ctrlMe.ctrl
regcval

EXtoME

tid
inst
cpsr
IsMult
ctrlMe
ctrlWb
ALUresult
newCC
Rd
Rn
pc8
pc4

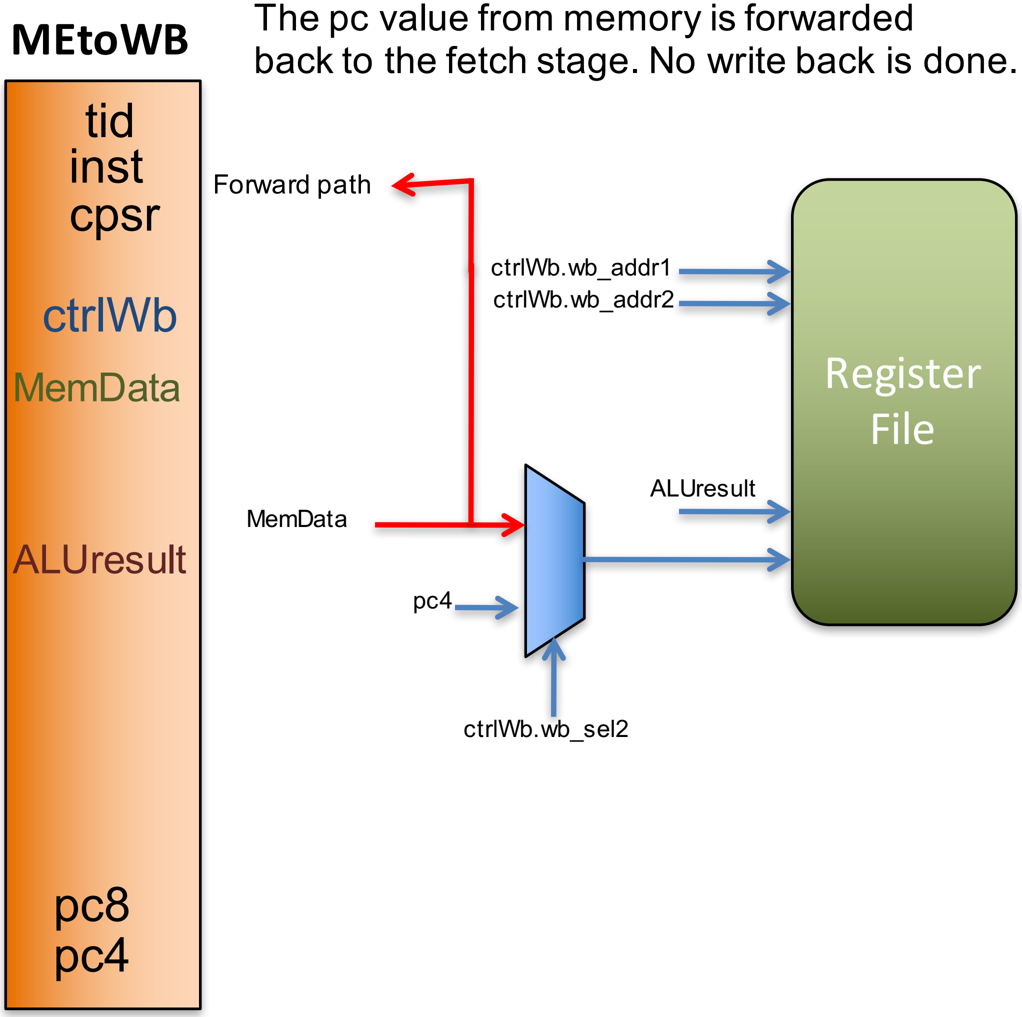
Memory Stage



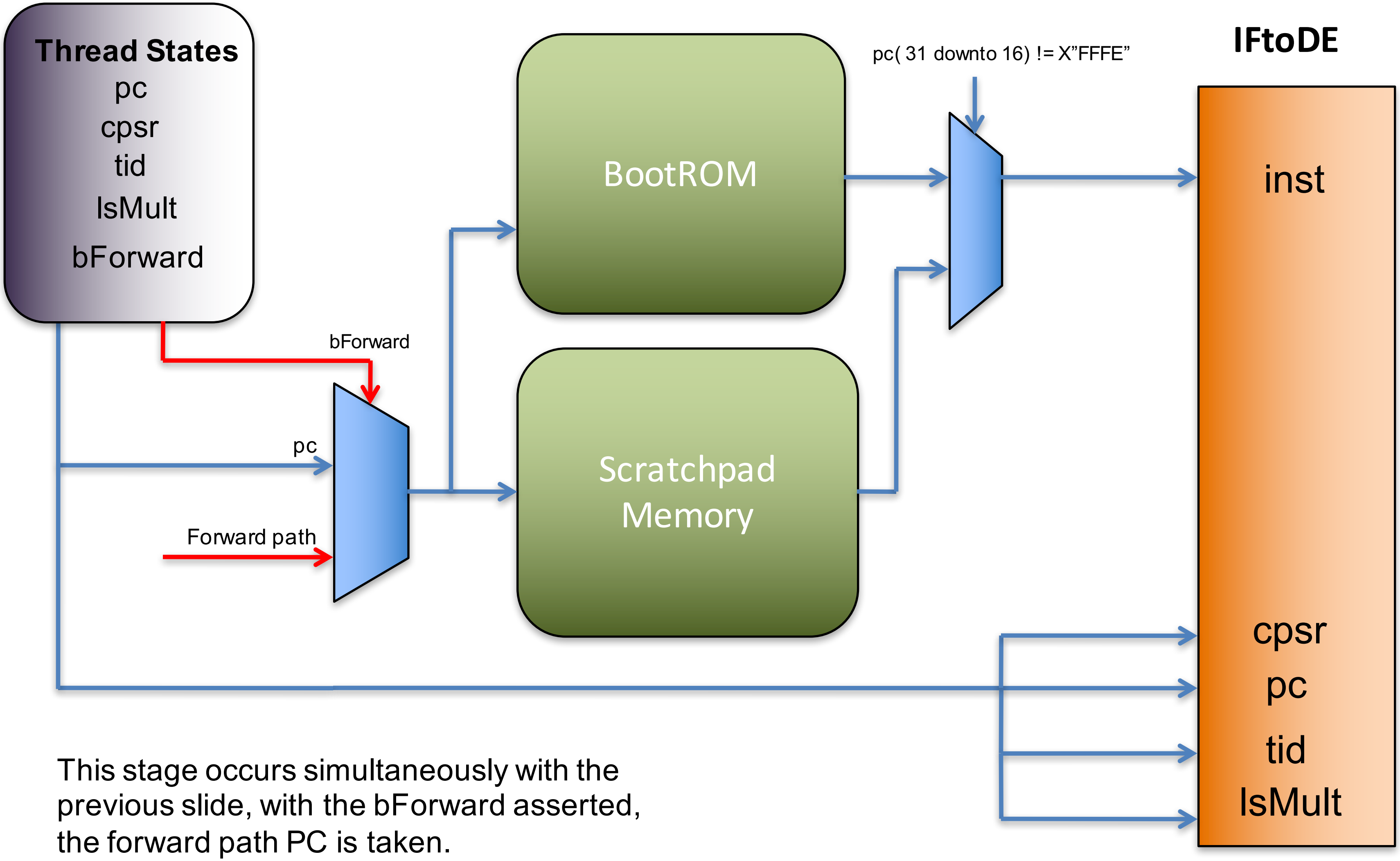
Access the memory to get the data.
the bForward flag is asserted so the next cycle, the fetch stage will get the forwarded pc.

Write Back Stage

The pc value from memory is forwarded back to the fetch stage. No write back is done.



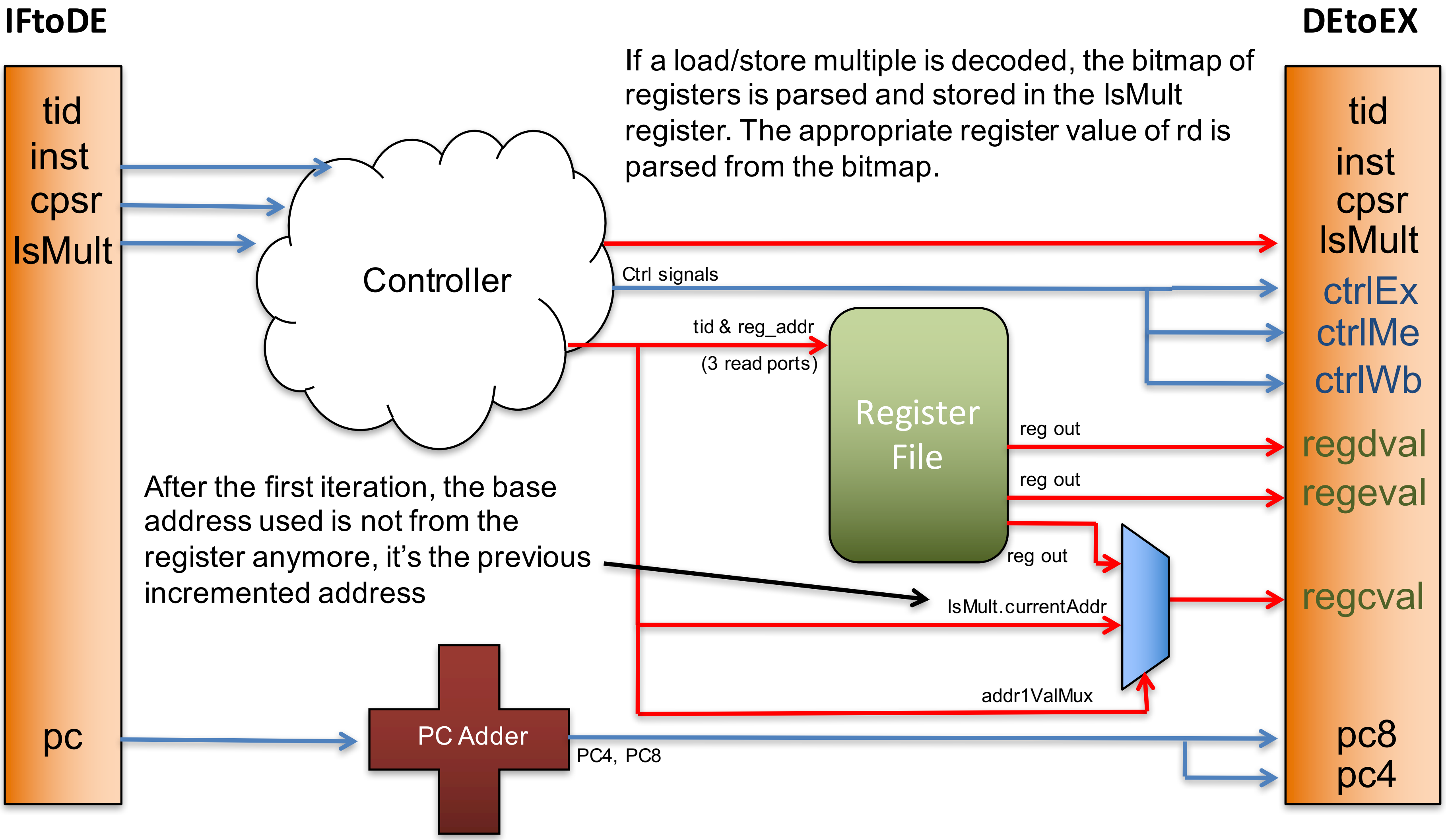
Fetch Stage



This stage occurs simultaneously with the previous slide, with the bForward asserted, the forward path PC is taken.

Load/Store Multiple Instructions

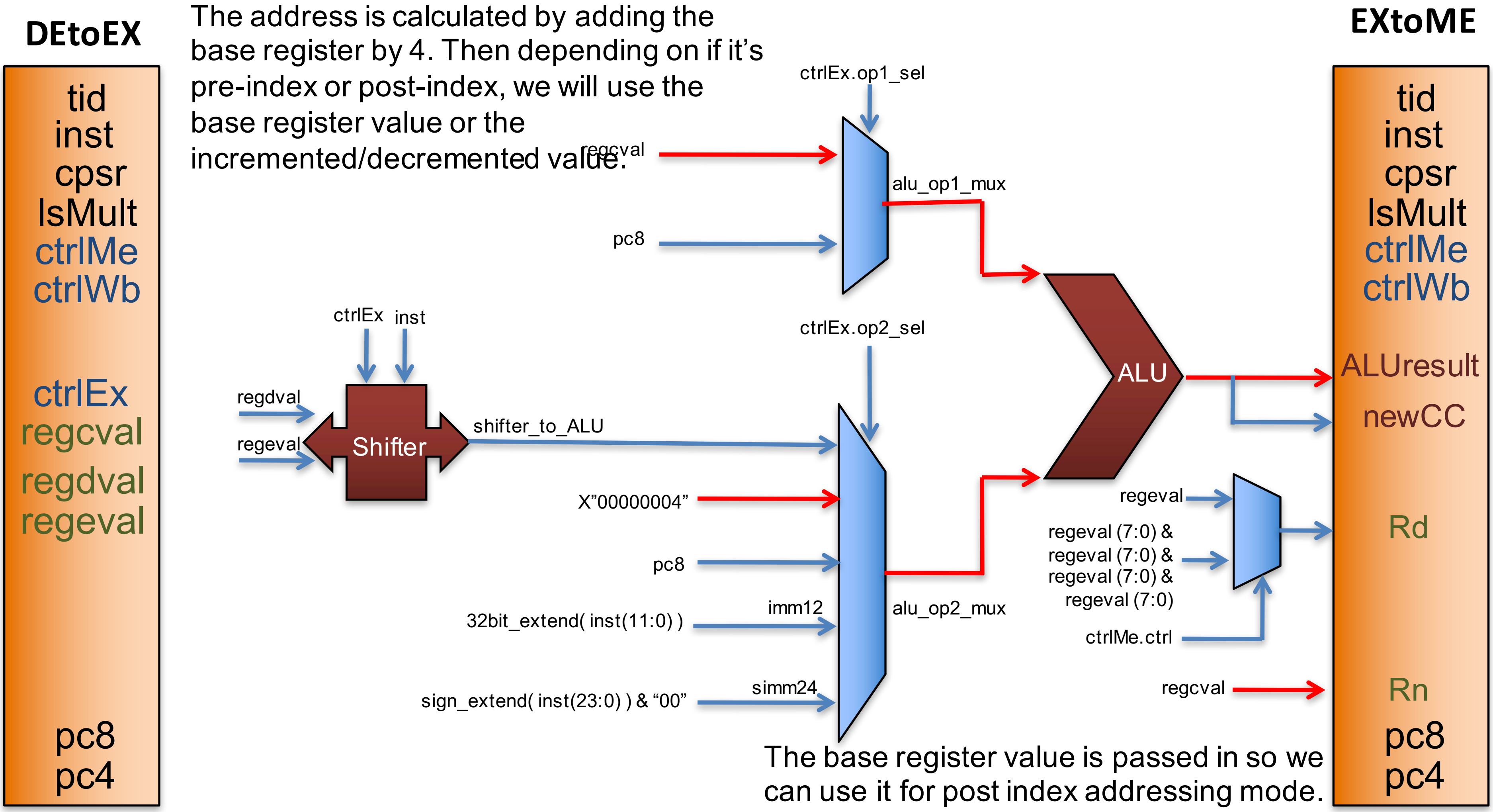
Decode Stage



If a load/store multiple is decoded, the bitmap of registers is parsed and stored in the IsMult register. The appropriate register value of rd is parsed from the bitmap.

After the first iteration, the base address used is not from the register anymore, it's the previous incremented address

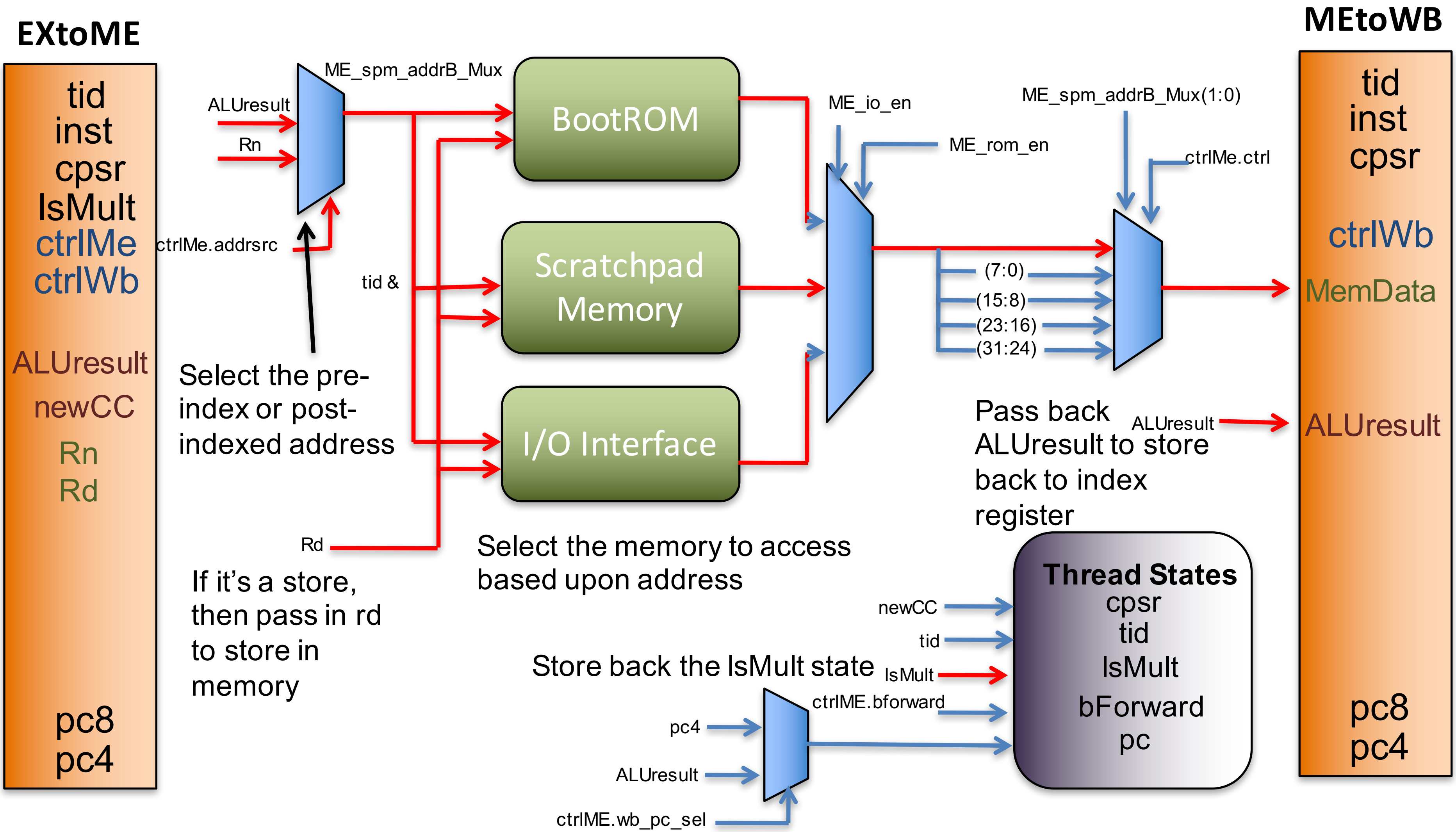
Execute Stage



The address is calculated by adding the base register by 4. Then depending on if it's pre-index or post-index, we will use the base register value or the incremented/decremented value.

The base register value is passed in so we can use it for post index addressing mode.

Memory Stage



Select the pre-index or post-indexed address

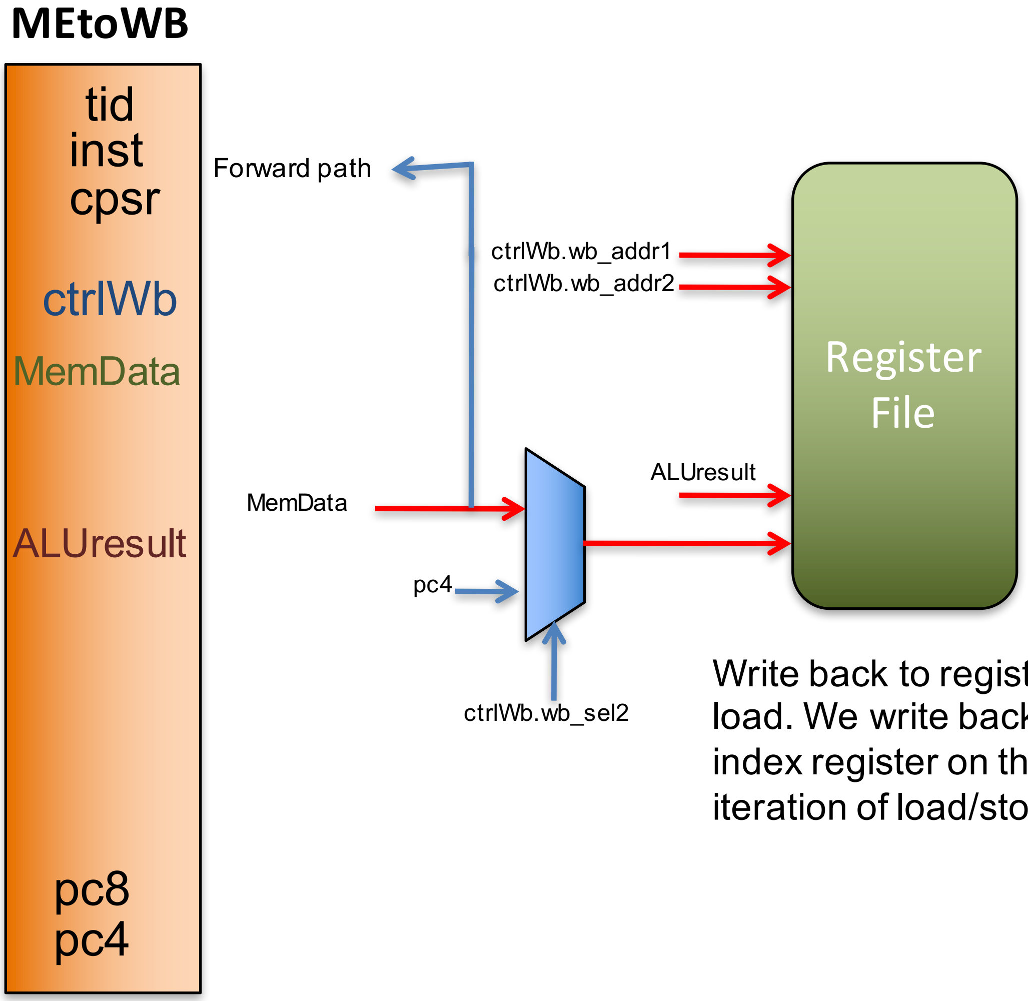
If it's a store, then pass in rd to store in memory

Select the memory to access based upon address

Pass back ALUresult to store back to index register

Store back the IsMult state

Write Back Stage



Write back to register if it’s a load. We write back the index register on the last iteration of load/store.