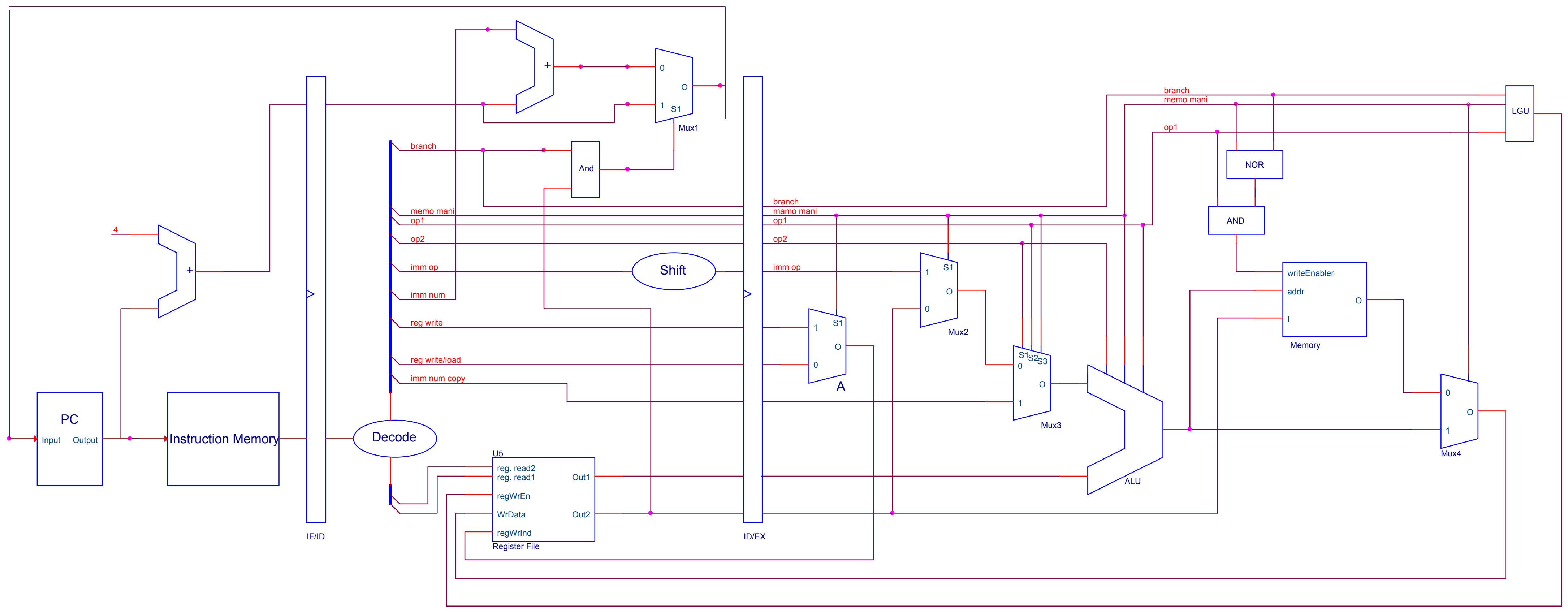

Decode bit spaces:

conditional branch: [00], type J, I, R, set to 1 if it's a bne instruction
memo mani(pulation): [01], type I, R, set to 0 if it reads/writes memory
op1: [02], type I, R, 0/1 meaning r/w to the memory in type I
op2: [03], type R
imm_op(erand): [03-07], type I, memory address offset
imm_num(ber): [02-11], type J, signed immediate number to be copied
reg_write(_index): [04-07], type R
reg_write_load: [12-15], type I, register being written in memory manipulation instructions
imm_num_copy: [04-11], type R' (type 4 in our taxonomy), signed number being copied
reg_read2: [08-11], type R, I
reg_read1: [12-15], type R, and type I if the memory is being written with a value from this register.

And for PC: Only if Out2 isn't 0 and branch bit is active, the branch would be taken in Mux1
NOR for Memory: only branch isn't active and memo_mani is 0, it is a type I instruction.
AND for Memory: only if it is a type I instruction and op1 is 1 (writing to mem) the writing is enabled.
LGU: output 1 to enable register writing for branch inactive & ((memo active & op1 = 0) or memo inactive), otherwise, disable register writer in case the output is not what we desired.

Multiplexers:
A: if memo_mani is 1, it's a load/store instruction, reg_write won't be used.
Mux2: if memo_mani is 0, we pass shifted imm_op to ALU to be added with reg_read_1

Mux3: if memo_mani is 0, we pass shifted imm_op (input 0), if memo_mani is 1, input 0 is reg_read_2.
If memo_mani is 1 and op1 is 0, pass reg_read_2,
if memo_mani is 1 and op1 is 1 and op2 is 0, pass reg_read_2,
otherwise, pass imm_num_copy.

ALU: if memo_mani is 1, do addition,
otherwise, if op1 is 0, op2 is 0, do addition,
op1 is 0, op2 is 1 do left shift, op1 is 1, op2 is 0 do XOR,
op1 is 1, op2 is 1, do copy (the inputs are right data with mux2 and mux3)

Mux4: if memo_mani is 0, the data to be written (or not) is from memory, otherwise it is computed by ALU.