Implementing Conditional Branches

I-Type Instructions

Previously made circuit that worked for I-type instructions, then combined with R-type circuit

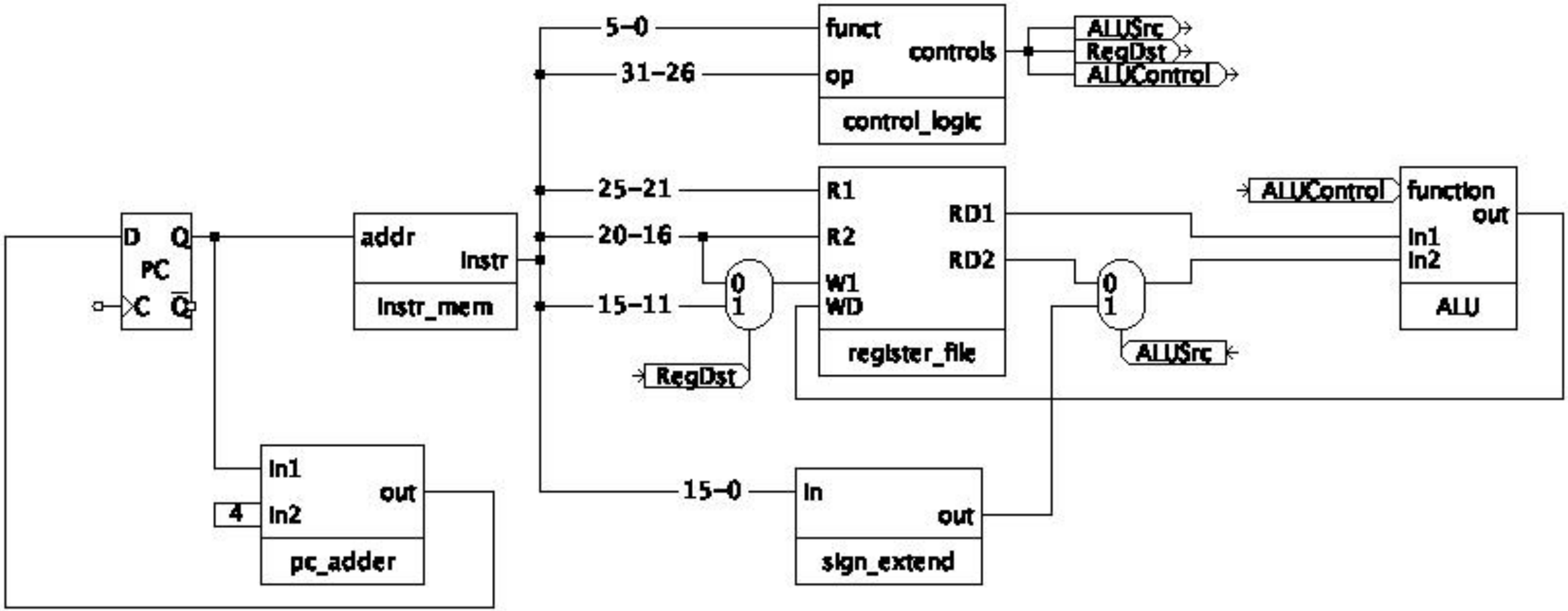
In reality, our circuit was only for subset of I-type instructions

Even though branches seem very different functionally, they are still I-type because of how they are stored

I-Type Instructions

We follow similar process again:

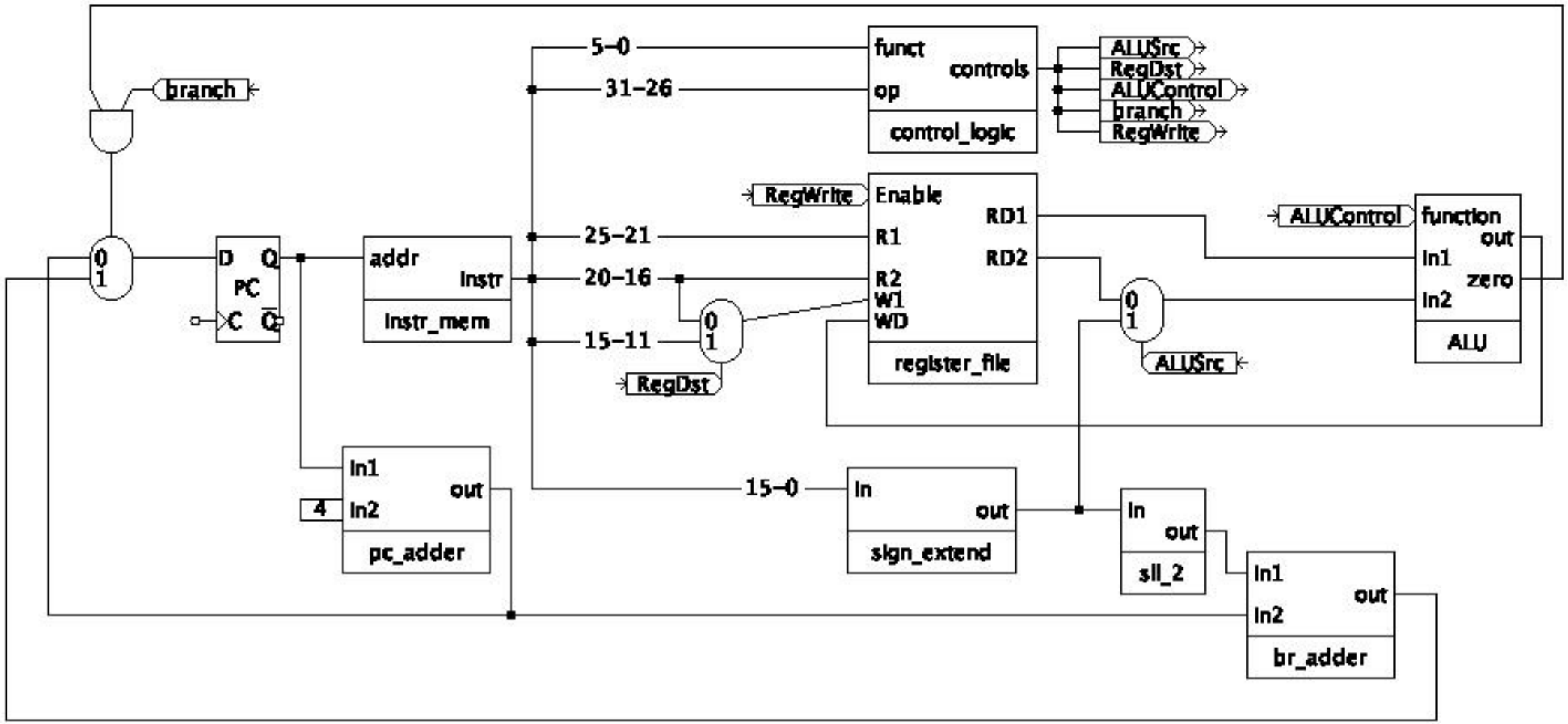
- Add capability for branches to circuit
- Use muxes to ensure backward compatibility


5-0
31-26
funct
op
controls
control_logic
ALUSrc
RegDst
ALUControl
25-21
20-16
15-11
R1
R2
W1
WD
RD1
RD2
register_file
0
1
RegDst
ALUControl
function
out
In1
In2
ALU
0
1
ALUSrc
D
Q
PC
C
Q
addr
Instr
Instr_mem
In1
4
In2
out
pc_adder
15-0
In
out
sign_extend

Adding branches

In the big picture, there are three ways branch is different from, e.g., `addi`. Branch instruction must

- Compute PC value of branch target
- Determine whether condition is satisfied
- *Not* write to general-purpose registers


branch
0
1
D
Q
PC
C
Q
addr
Instr
Instr_mem
5-0
31-26
25-21
20-16
15-11
RegDst
0
1
funct
op
controls
control_logic
ALUSrc
RegDst
ALUControl
branch
RegWrite
RegWrite
Enable
R1
R2
W1
WD
RD1
RD2
register_file
0
1
ALUSrc
ALUControl
function
In1
In2
out
zero
ALU
In1
4
In2
out
pc_adder
15-0
In
out
sign_extend
In
out
sll_2
In1
In2
out
br_adder

Modifications to hardware

`register_file`:

- Enable bit added – if input is 1, registers writable; otherwise, registers do not change
- Needed because we cannot "cut wires" for different inputs – `WD` will have *something* going into it even for branch instruction
- Register must be able to ignore bogus inputs – no register is modified during branch instruction

`ALU output zero`:

- Answering yes/no question: was output 0?
- Single bit – 1 for yes, 0 for no
- Feels backwards at first – 1 if output 0, 0 if output anything nonzero
- Used for checking branch condition

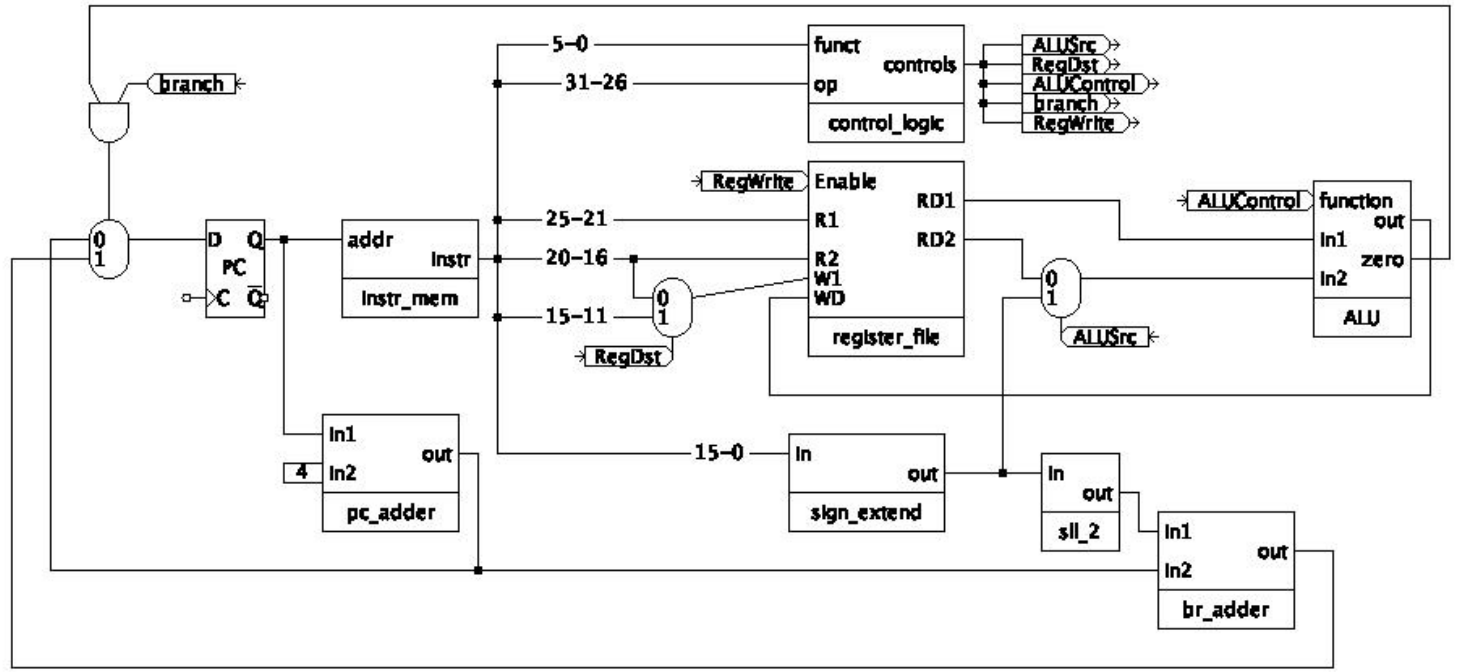
New control logic

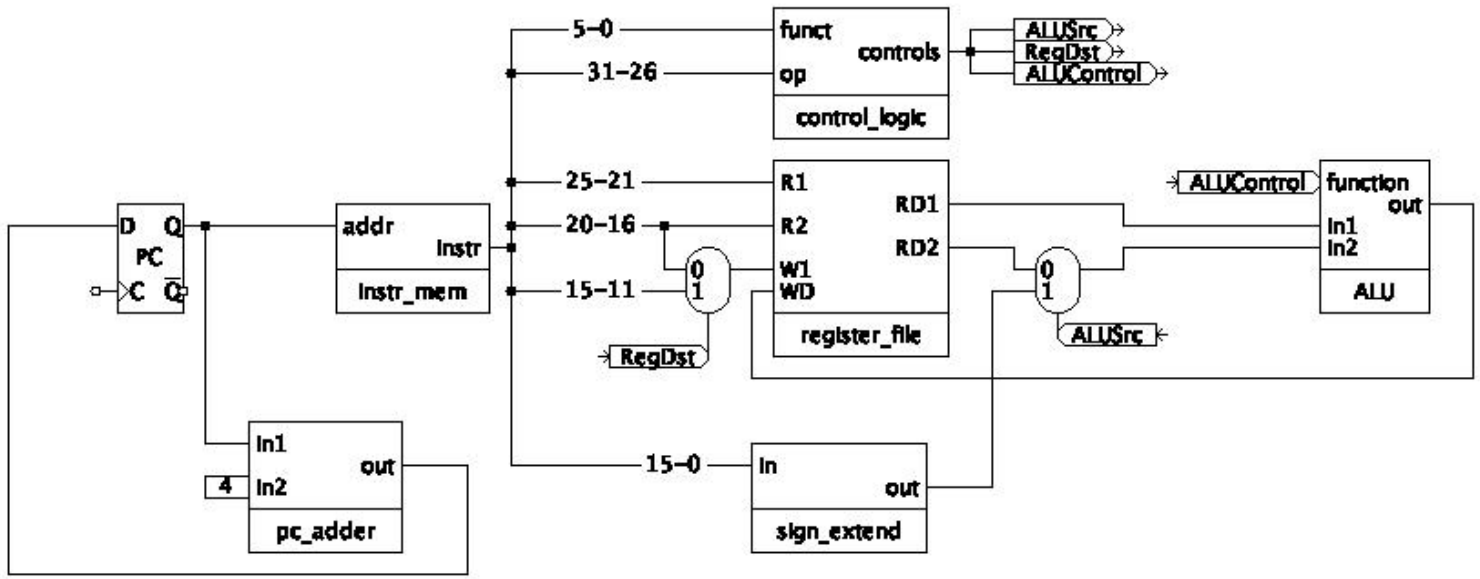
`branch`:

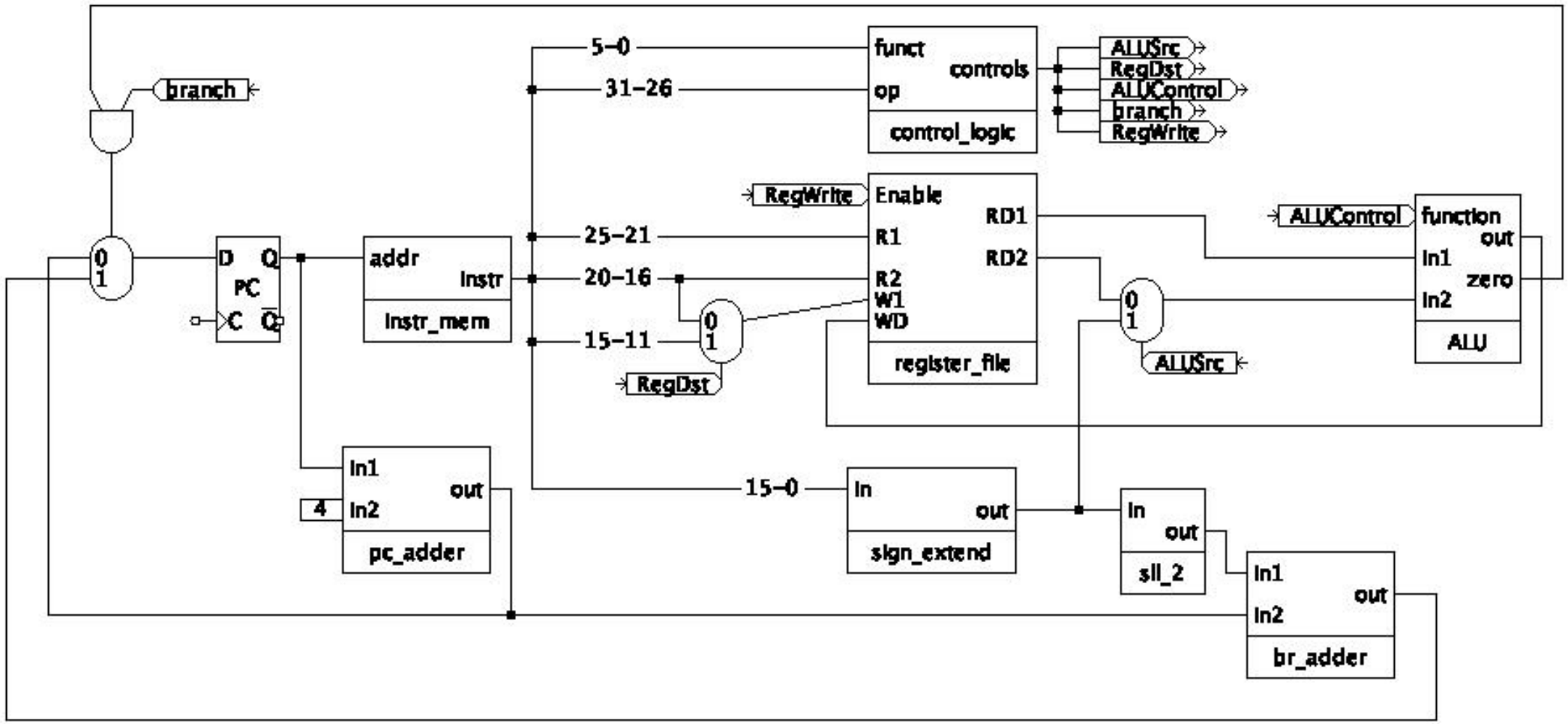
- Whether instruction is branch – 1 for yes, 0 for no
- Branch taken only if branch is 1 *and* operands are equal

`RegWrite`:

- Whether register should be written to – 1 for yes, 0 for no
- Do not want to accidentally overwrite register during branch


branch
5–0
31–26
funct
op
controls
control_logic
ALUSrc
RegDst
ALUControl
branch
RegWrite
RegWrite
Enable
R1
R2
W1
WD
RD1
RD2
register_file
25–21
20–16
15–11
RegDst
0
1
D Q
PC
C Q
addr
Instr
Instr_mem
ALUControl
function
In1
In2
out
zero
ALU
ALUSrc
In1
In2
out
4
pc_adder
15–0
In
out
sign_extend
sll_2
br_adder


5–0
31–26
funct
op
controls
control_logic
ALUSrc
RegDst
ALUControl
25–21
20–16
15–11
R1
R2
W1
WD
RD1
RD2
register_file
RegDst
0
1
D Q
PC
C Q
addr
Instr
Instr_mem
ALUControl
function
In1
In2
out
ALU
ALUSrc
In1
In2
out
4
pc_adder
15–0
In
out
sign_extend


branch
0
1
D
Q
PC
C
Q
addr
Instr
Instr_mem
5-0
31-26
25-21
20-16
15-11
RegDst
0
1
funct
op
controls
control_logic
ALUSrc
RegDst
ALUControl
branch
RegWrite
RegWrite
Enable
R1
R2
W1
WD
RD1
RD2
register_file
0
1
ALUSrc
ALUControl
function
In1
In2
out
zero
ALU
In1
4
In2
out
pc_adder
15-0
In
out
sign_extend
In
out
sll_2
In1
In2
out
br_adder