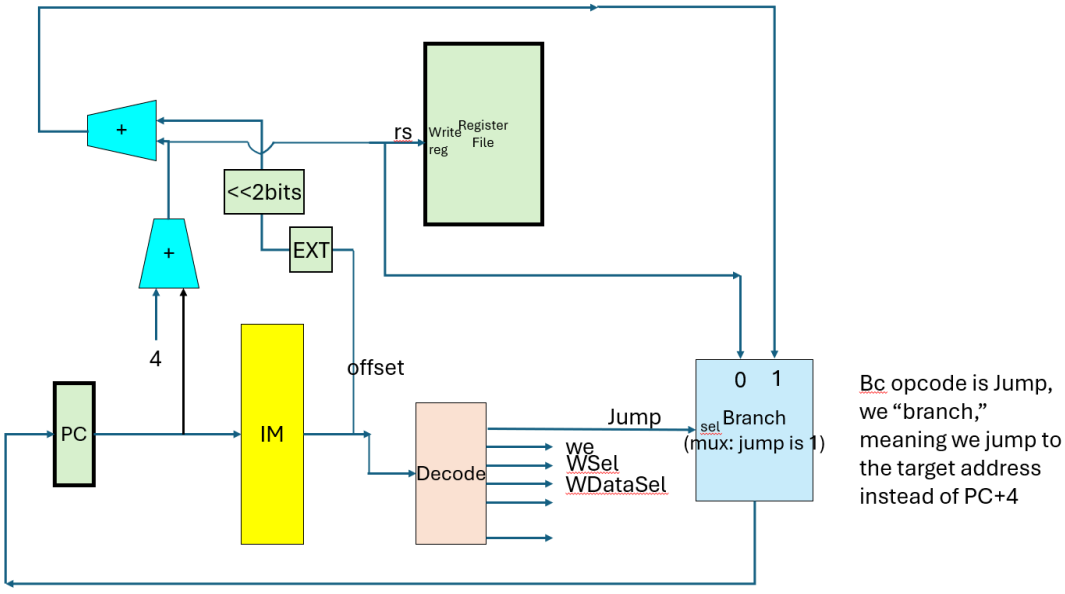
PART 1: Adding JAL and RET instructions

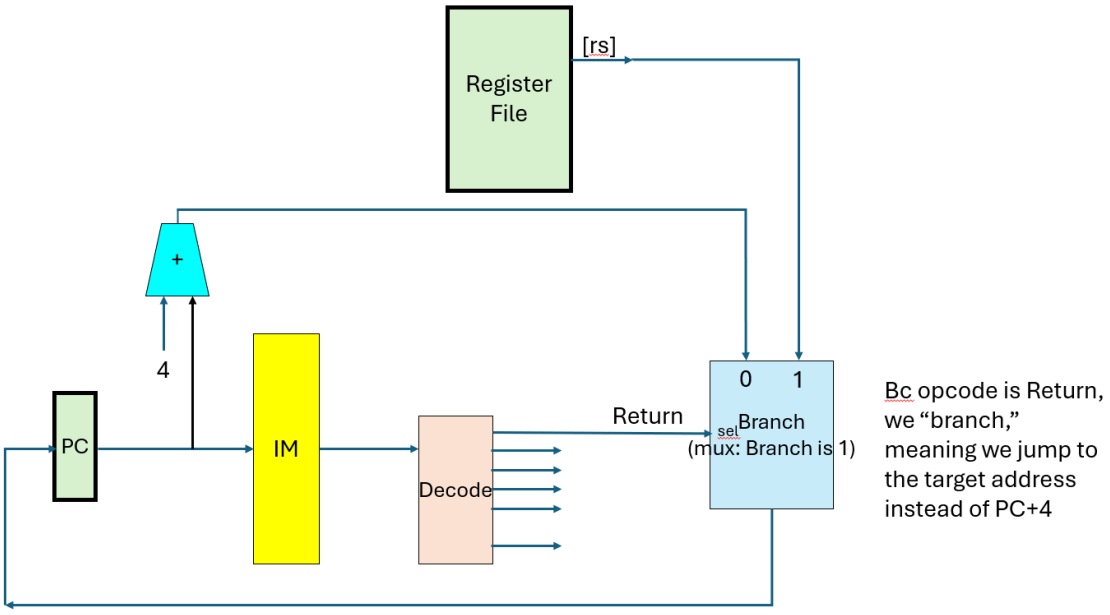
(1) Datapaths

Datapath for JAL rs, offset



Bc opcode is Jump, we "branch," meaning we jump to the target address instead of PC+4

Datapath for RET rs



Bc opcode is Return, we "branch," meaning we jump to the target address instead of PC+4

(2) I updated processor.v and decode.v

(3)
The register values match those I saw in Part A. The additional register R31 matches the expected output of a JAL instruction when PC=0.

The register values for all three programs and memory values for Programs 1 and 2 are below:

Program 1: Completion time 440

```
INS_MEM:        Time:430        PC: 36  Instruction: 33e00000
REGFILE:        Time:430        R0: 107 R1:   8 R2: 828 R3:   0 R4:   0 R5:   0 R6:   0 R7:   0 R8:   0 R31:   4

INS_MEM:        Time:440        PC:  4  Instruction: 28000000
REGFILE:        Time:440        R0: 107 R1:   8 R2: 828 R3:   0 R4:   0 R5:   0 R6:   0 R7:   0 R8:   0 R31:   4
```

Program 2: Completion time 440

```
INS_MEM:        Time:430        PC: 36  Instruction: 33e00000
REGFILE:        Time:430        R0: 107 R1:   8 R2:   0 R3:   0 R4:   0 R5:   0 R6:   0 R7:   0 R8:   0 R31:   4
DMEM[8..15]:    Time:430        100     101     102     103     104     105     106     107
DMEM[16..23]:   Time:430        116     117     118     119     120     121     122     123

INS_MEM:        Time:440        PC:  4  Instruction: 28000000
DMEM[8..15]:    Time:440        100     101     102     103     104     105     106     107
DMEM[16..23]:   Time:440        116     117     118     119     120     121     122     123
REGFILE:        Time:440        R0: 107 R1:   8 R2:   0 R3:   0 R4:   0 R5:   0 R6:   0 R7:   0 R8:   0 R31:   4
```

Program 3: Completion time 790

```
INS_MEM:        Time:780        PC: 64  Instruction: 33e00000
DMEM[8..15]:    Time:780        108     109     110     111     112     113     114     115
DMEM[16..23]:   Time:780        208     210     212     214     216     218     220     222
REGFILE:        Time:780        R0: 107 R1: 115 R2:   7 R3:   0 R4:  15 R5: 222 R6:  23 R7:   0 R8:   0 R31:   4

INS_MEM:        Time:790        PC:  4  Instruction: 28000000
REGFILE:        Time:790        R0: 107 R1: 115 R2:   7 R3:   0 R4:  15 R5: 222 R6:  23 R7:   0 R8:   0 R31:   4
DMEM[8..15]:    Time:790        108     109     110     111     112     113     114     115
DMEM[16..23]:   Time:790        208     210     212     214     216     218     220     222
```

PART 2: Recursive Vector Reduction

(1) I submitted extracredit.v as well as regfile.v, where I added the initialization of the stack pointer with: REG_FILE[30] <= 32'h80;

(2) Completion Time and Register Values of program

Completion time is **1250**, using 125 cycles:

Setup: 2 cycles (ADDI + JAL)
8 recursive entries: 8 × 8 = 64 cycles
Base case: 2 cycles (BNEZ + RET)
8 unwinds: 8 × 7 = 56 cycles
HALT: 1 cycle

We end up with the registers: R0: 100 R2: 828 R8: 0 R30: 128 R31: 8

```
INS_MEM:        Time:1240       PC: 76  Instruction: 33e00000
REGFILE:        Time:1240       R0: 100 R1:   0 R2: 828 R3:   0 R4:   0 R5:   0 R6:   0 R7:   0 R8:   0 R30: 128 R31:   8

INS_MEM:        Time:1250       PC:  8  Instruction: 28000000
REGFILE:        Time:1250       R0: 100 R1:   0 R2: 828 R3:   0 R4:   0 R5:   0 R6:   0 R7:   0 R8:   0 R30: 128 R31:   8
```

It makes sense that our last R0 is 100 instead of 107 (like it was in the iterative version) because of the LIFO nature of the stack.

Recursive Assembly Program:

```
// PROGRAM 1B
// Recursive vector reduction (recursive version of PROGRAM 1)
// Use R2 for sum; use R1 for idx; use R8 for remaining count
0: instruction = 32'h15080008; // ADDI R8, R8, 8;
4: instruction = 32'h2fe00002; // JAL FUNC: 2
8: instruction = 32'h28000000; // HALT
12: instruction = 32'h00000000; // NOP

16: instruction = 32'h25000004; // BNEZ R8, 4  FUNC:
20: instruction = 32'h33E00000;  // RET

24: instruction = 32'h1BDFFFFF; // SD R31, -1(R30)  RECURSE:
28: instruction = 32'h17DEFFFF; // ADDI R30, R30, -1
32: instruction = 32'h1BC1FFFF; // SD R1, -1(R30)
36: instruction = 32'h17DEFFFF; // ADDI R30, R30, -1

40: instruction = 32'h14210001; // ADDI R1, R1, 1
44: instruction = 32'h1508FFFF; // ADDI R8, R8, -1

48: instruction = 32'h2ffFFFF7; // JAL FUNC: -9

52: instruction = 32'h13C10000; // LD R1, 0(R30)
56: instruction = 32'h17DE0001; // ADDI R30, R30, 1
60: instruction = 32'h13DF0000; // LD R31, 0(R30)
64: instruction = 32'h17DE0001; // ADDI R30, R30, 1

68: instruction = 32'h10200000; // LD R0, 0(R1)
72: instruction = 32'h04401000; // ADD R2, R2, R0
76: instruction = 32'h33E00000; // RET
```