```
MODULE      ProgramAccess
INTERFACE   (Offset7..Offset0, Direct12..Direct0,
            Load, Select2..Select0,
            Reset, Clock
            -> ProgAddr12..ProgAddr0, PC12..PC0);

" Description

" This is the implementation of the ProgramAccess Module.
" It takes in 8 bits of signed offset data, 13 bits of direct data,
" 4 control signals, a reset and clock, and returns the appropriate
" ProgramAddress and PC by selecting the right inputs and adding them.

" The control signals work in combinations to create the following functions:
" 1 0 0 0 -> direct jump                    -> PC = a
" 0 0 0 1 -> hold                           -> PC = PC + 0
" 0 0 1 1 -> increment                      -> PC = PC + 1
" 0 0 1 0 -> add r                          -> PC = PC + r
" 1 1 X 0 -> load last 8 bits of offset     -> PC = [PC12..PC8, Off7..Off0]
" 1 1 X 1 -> load first 5 bits of offset    -> PC = [Off4..Off0, PC7..PC0]

" Revision History:
" 02/22/23   Steven Lei   Created file
" 02/22/23   Steven Lei   Added logic and tested
" 02/22/23   STeven Lei   Updated comments

" Inputs

Offset7..Offset0                pin;                        " 8 bits of signed offset data
Direct12..Direct0               pin;                        " 13 bits of direct data
Load                            pin;                        " load signal for PC
Select2..Select0                pin;                        " select signals for MUX
Reset                           pin;                        " reset
Clock                           pin;                        " clock

" Outputs

ProgAddr12..ProgAddr0           pin;                        " 13 bit program address output, same
as PC
PC12..PC0                       pin ISTYPE 'REG, KEEP';     " 13 bit program counter output


" Intermediate Terms

SOffset12..SOffset0             node ;         " sign extend the 8 bit offset data to 13 bit
SHOffset12..SHOffset0           node;          " load the first 5 high bits to PC
SLOffset12..SLOffset0           node;          " load the last 8 low bits to PC

CarryIn12..CarryIn0             node ;         " 13 bit CarryIn used in adder.
A12..A0                         node ;         " first input of the adder
B12..B0                         node;          " second input of the adder
Sum12..Sum0                     node ;         " adder sum


" Buses
```

```
Direct = [Direct12..Direct0];
SOffset = [SOffset12..SOffset0];
SHOffset = [SHOffset12..SHOffset0];
SLOffset = [SLOffset12..SLOffset0];

A = [A12..A0];
B = [B12..B0];
Sum = [Sum12..Sum0];
CarryIn = [CarryIn12..CarryIn0];

PC = [PC12..PC0];
ProgAddr = [ProgAddr12..ProgAddr0];

" Constants

ZEROES = [0,0,0,0,0,0,0,0,0,0,0,0,0];
ONE =    [0,0,0,0,0,0,0,0,0,0,0,0,1];


EQUATIONS

PC.CLK = Clock;
PC.CLR = !Reset;

" We will convert the 8 bit offset data depending on our signal
" We can sign extend the 8 bit offset to 13 bits by appending the high order bit
" For loading the first 5 hi bits, we can load the 5 hi bits into PC and keep the rest
" For loading the last 8 low bits, we can load the 8 low bits into PC and keep the rest


SOffset = [Offset7, Offset7, Offset7, Offset7, Offset7, Offset7..Offset0];
SHOffset =[Offset4..Offset0, PC7..PC0];
SLOffset =[PC12..PC8, Offset7..Offset0];

" Here we use logic to select the adder inputs
" A will be dependent on the Load signals
" and B (the MUX) will be dependent on the select signals
" as described in the description

A = (!Load & PC);                                   " If load = 1 then PC is not loaded in A
B = ((!Select2 & !Select1 & !Select0 & Direct)      " 1 0 0 0 PC = a
  #  (!Select2 & !Select1 & Select0 & ZEROES)       " 0 0 0 1 PC = PC+0
  #  (!Select2 & Select1 & Select0 & ONE)               " 0 0 1 1 PC = PC+1
  #  (!Select2 & Select1 & !Select0 & SOffset)      " 0 0 1 0 PC = PC + r
  #  (Select2 & !Select0 & SLOffset)                     " 1 X 0 0 PC = [PC12..PC8,
Off7..Off0]
  #  (Select2 & Select0 & SHOffset)                   " 1 X 0 1 PC = [Off4..Off0, PC7..PC0]
  );

" From the 8 bit adder assignment, we know the following
" For each sum, S = A $ B $ CarryIn
" For each CarryOut, C(n) = (A & B) # (C(n-1) & (A $ B))

" CarryOut[N] = CarryIn[N+1]
" First CarryIn is zero, we take care of incrementation by adding ONE
```

```
CarryIn0 = 0;
CarryIn[12..1]=(A[11..0] & B[11..0]) # (CarryIn[11..0] & (A[11..0] $ B[11..0]));

Sum = (A $ B $ CarryIn);
PC:= Sum;                          "the final PC value will just be the sum
ProgAddr = PC;                     "and ProgAddr is just the PC


END ProgramAccess
```