1. List the instructions:

   IMemory[0] = 32'h8ca30002; 32'b 100011 | 00101 | 00011 | 0000000000000010
   Lw $3, 2(5)

   IMemory[1] = 32'h8c620007; 32'b 100011 | 00011 | 00010 | 0000000000000111
   Lw $2, 7($3)

   IMemory[2] = 32'h10200003; 32'b 000100 | 00001 | 00000 | 0000000000000011
   Beq $1, $0, 3

   IMemory[3] = 32'h8c0b0010; 32'b 100011 | 00000 | 01011 | 0000000000010000
   Lw $11, 16($0)

   IMemory[4] = 32'had6f0009; 32'b 101011 | 01011 | 01111 | 0000000000001001
   Sw $15, 9($11)

   IMemory[5] = 32'h10210001; 32'b 000100 | 00001 | 00001 | 0000000000000001
   Beq $1, $1, 1

   IMemory[6] = 32'h8ca30002; 32'b 100011 | 00101 | 00011 | 0000000000000010
   Lw $3, 2($5)

   IMemory[7] = 32'h8c0b0010; 32'b 100011 | 00000 | 01011 | 0000000000010000
   Lw $11, 16($0)

   IMemory[8] = 32'h002b6020; 32'b 000000 | 00001 | 01011 | 01100 | 00000 | 100000
   Add $12, $1, $11

2. Code Modifications:

   Stall = 0; // before

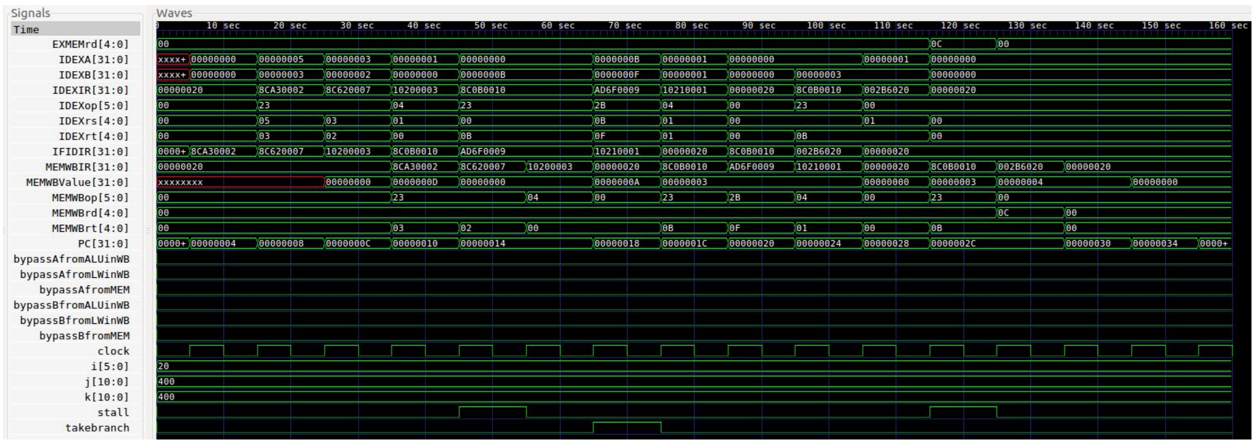
   Stall = (MEMWBIR[31:26] == LW) && ((((IDEXop == LW) | (IDEXop == SW)) && (IDEXrs == MEMWBrd)) | ((IDEXop == ALUop) && ((IDEXrs == MEMWBrd) | (IDEXrt == MEMWBrd)))); // after

3. Explanation:

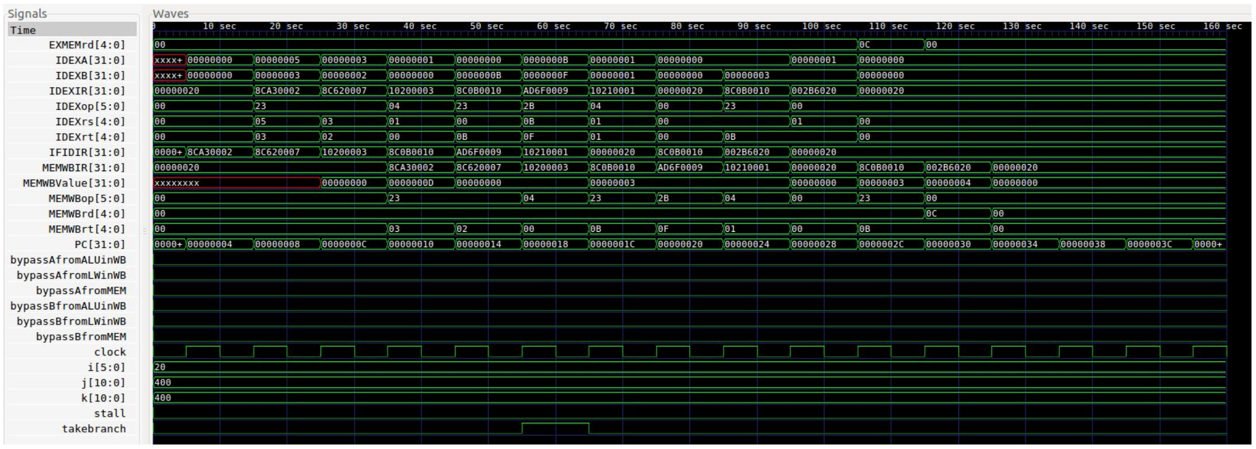
   Since the Load-Use Data Hazard occurred, we need to make sure the stall should be satisfied two conditions, 1. The instruction is Load, 2 the address calculation finished or in the ALU, the address has been passed from Write back stage to IDEX stage, the value should be able to use by rs or rt. So, the first part: (MEMWBIR[31:26] == LW) is to make sure the instruction is load, the second part: (((IDEXop == LW) | (IDEXop == SW)) && (IDEXrs == MEMWBrd)) is the part that address calculation, which means the rd is the target address should be available to IDEX stage, the third part: ((IDEXop == ALUop) && ((IDEXrs == MEMWBrd) | (IDEXrt == MEMWBrd))) is that the register value from write back stage should available in IDEX stage before into ALU.

4. Waveform:

   - With stall:



-Without stall:



There are some obvious observation, like in the MEMWBValue, we can tell if using stall, the value access at time 65sec to 75sec, it has one more value used. And for the IFIDIR, if has get less address access at point 50-55 sec, same ass EX stage. They just happened when address or value from WB stage be needed

5. Verilog code:

```
// Incomplete behavioral model of MIPS pipeline

module mipspipe_mp4 (clock);

// in_out
  input clock;

  // Instruction opcodes
  parameter LW = 6'b100011, SW = 6'b101011, BEQ = 6'b000100, nop = 32'b00000_100000,
ALUop = 6'b0;
  reg [31:0] PC, Regs[0:31], IMemory[0:1023], DMemory[0:1023], // instruction and data
memories
        IFIDIR, IDEXA, IDEXB, IDEXIR, EXMEMIR, EXMEMB, // pipeline latches
```

```
            EXMEMALUOut, MEMWBValue, MEMWBIR;

  wire [4:0] IDEXrs, IDEXrt, EXMEMrd, MEMWBrd, MEMWBrt; // hold register fields
  wire [5:0] EXMEMop, MEMWBop, IDEXop; // hold opcodes
  wire [31:0] Ain, Bin; // ALU inputs


  // declare the bypass signals
  wire takebranch, stall, bypassAfromMEM, bypassAfromALUinWB, bypassBfromMEM,
bypassBfromALUinWB, bypassAfromLWinWB, bypassBfromLWinWB;

   // Define fields of pipeline latches
   assign IDEXrs = IDEXIR[25:21]; // rs field
   assign IDEXrt = IDEXIR[20:16]; // rt field
   assign EXMEMrd = EXMEMIR[15:11]; // rd field
   assign MEMWBrd = MEMWBIR[15:11]; // rd field
   assign MEMWBrt = MEMWBIR[20:16]; // rt field -- for loads
   assign EXMEMop = EXMEMIR[31:26]; // opcode
   assign MEMWBop = MEMWBIR[31:26]; // opcode
   assign IDEXop = IDEXIR[31:26]; // opcode

  // The bypass to input A from the MEM stage for an ALU operation
  assign bypassAfromMEM = (IDEXrs == EXMEMrd) & (IDEXrs!=0) & (EXMEMop==ALUop); //
yes, bypass
  // The bypass to input B from the MEM stage for an ALU operation
  assign bypassBfromMEM = (IDEXrt == EXMEMrd) & (IDEXrt!=0) & (EXMEMop==ALUop); //
yes, bypass
  // The bypass to input A from the WB stage for an ALU operation
  assign bypassAfromALUinWB = (IDEXrs == MEMWBrd) & (IDEXrs!=0) &
(MEMWBop==ALUop);
  // The bypass to input B from the WB stage for an ALU operation
  assign bypassBfromALUinWB = (IDEXrt == MEMWBrd) & (IDEXrt!=0) &
(MEMWBop==ALUop);
  // The bypass to input A from the WB stage for an LW operation
  assign bypassAfromLWinWB = (IDEXrs == MEMWBIR[20:16]) & (IDEXrs!=0) &
(MEMWBop==LW);
  // The bypass to input B from the WB stage for an LW operation
  assign bypassBfromLWinWB = (IDEXrt == MEMWBIR[20:16]) & (IDEXrt!=0) &
(MEMWBop==LW);

  // The A input to the ALU is bypassed from MEM if there is a bypass there,
  // Otherwise from WB if there is a bypass there, and otherwise comes from the IDEX register
  assign Ain = bypassAfromMEM? EXMEMALUOut : (bypassAfromALUinWB |
bypassAfromLWinWB)? MEMWBValue : IDEXA;

  // The B input to the ALU is bypassed from MEM if there is a bypass there,
  // Otherwise from WB if there is a bypass there, and otherwise comes from the IDEX register
  assign Bin = bypassBfromMEM? EXMEMALUOut : (bypassBfromALUinWB |
bypassBfromLWinWB)? MEMWBValue : IDEXB;

  // The signal for detecting a stall based on the use of a result from LW
  assign stall = (MEMWBIR[31:26] == LW) && ((((IDEXop == LW) | (IDEXop == SW)) &&
(IDEXrs == MEMWBrd)) | ((IDEXop == ALUop) && ((IDEXrs == MEMWBrd) | (IDEXrt ==
MEMWBrd))));// modified

  // The signal for a taken branch: instruction is BEQ and registers are equal
  assign takebranch = (IFIDIR[31:26]==BEQ) && (Regs[IFIDIR[25:21]]==Regs[IFIDIR[20:16]]);

  reg [5:0] i; // used to initialize latches
  reg [10:0] j,k; // used to initialize memories

  initial
  begin
    PC = 0;
    IFIDIR = nop;
    IDEXIR = nop;
    EXMEMIR = nop;
    MEMWBIR = nop; // put nops in pipeline registers

    for (i = 0;i<=31;i = i+1) Regs[i] = i; // initialize registers

    IMemory[0] = 32'h8ca30002;
    IMemory[1] = 32'h8c620007;
    IMemory[2] = 32'h10200003;
    IMemory[3] = 32'h8c0b0010;
    IMemory[4] = 32'had6f0009;
    IMemory[5] = 32'h10210001;
    IMemory[6] = 32'h8ca30002;
    IMemory[7] = 32'h8c0b0010;
    IMemory[8] = 32'h002b6020;
    for (j=9; j<=1023; j=j+1) IMemory[j] = nop; // initialize instruction memories

    DMemory[0] = 32'h00000000;
    DMemory[1] = 32'h0000000d;
    DMemory[2] = 32'h00000000;
    DMemory[3] = 32'h00000000;
    DMemory[4] = 32'h00000003;
    DMemory[5] = 32'hffffffff;
    for (k=6; k<=1023; k=k+1) DMemory[k] = 0; // initialize data memories
  end

  always @ (posedge clock)
  begin

          if (~stall)  // the first three pipeline stages stall if there is a load hazard
    begin

            // FETCH: Fetch instruction & update PC
            if (~takebranch)
      begin
        IFIDIR <= IMemory[PC>>2];
        PC <= PC + 4;
      end
      else
      begin // a taken branch is in ID; instruction in IF is wrong; insert a nop and reset the PC
        IFIDIR <= nop;
        PC <= PC + ({{16{IFIDIR[15]}}, IFIDIR[15:0]}<<2);
      end

      // DECODE: Read registers
      IDEXA <= Regs[IFIDIR[25:21]];
      IDEXB <= Regs[IFIDIR[20:16]];
      IDEXIR <= IFIDIR;

      // EX: Address calculation or ALU operation
      if ((IDEXop==LW) |(IDEXop==SW)) // address calculation & copy B
        EXMEMALUOut <= Ain +{{16{IDEXIR[15]}}, IDEXIR[15:0]};
      else if (IDEXop==ALUop)
      begin
        case (IDEXIR[5:0]) // R-type instruction
          32: EXMEMALUOut <= Ain + Bin; // add operation
          default: ; // other R-type operations: subtract, SLT, etc.
        endcase
      end

      EXMEMIR <= IDEXIR;
      EXMEMB <= Bin; // pass along the IR & B register
```

```
        end
    else EXMEMIR <= nop; // Freeze first three stages of pipeline

    // MEM
    if (EXMEMop==ALUop) MEMWBValue <= EXMEMALUOut; // pass along ALU result
    else if (EXMEMop == LW) MEMWBValue <= DMemory[EXMEMALUOut>>2];
    else if (EXMEMop == SW) DMemory[EXMEMALUOut>>2] <=EXMEMB; // store

    MEMWBIR <= EXMEMIR; // pass along IR

    // WB
    if ((MEMWBop==ALUop) & (MEMWBrd != 0)) // update latches if ALU operation and
destination not 0
    Regs[MEMWBrd] <= MEMWBValue; // ALU operation
    else if ((MEMWBop == LW)& (MEMWBrt != 0)) // update latches if load and destination not 0
    Regs[MEMWBrt] <= MEMWBValue;

  end

endmodule
```