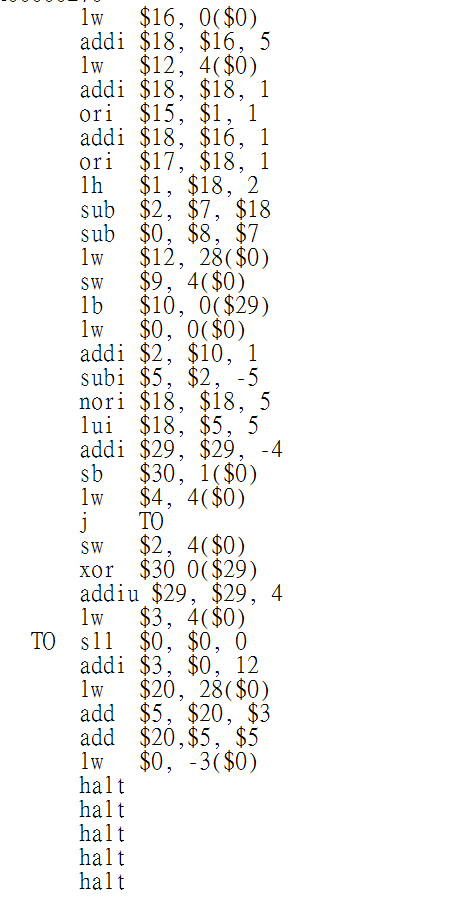


**1) Flowchart**

1. **Project Description**
2. Variables :
   1. regis[32], represents as registers and initial to 0 (unsigned),
   2. iimage[256], represents as instruction-memory and initial to 0 (unsigned),
   3. dimage[256], represents as D-memory and initial to 0 (unsigned),
   4. PC, represents the address of current instruction,
   5. If, id, ex, mem, and wb (pointers), represent each stage’s register;
3. Classes :
   1. **Readfile** will get input data from iimage.bin and dimage.bin, transfer the data to big-endian and store to array iimage[256] and dimage[256],
   2. **Decode** isthe class for stages’ register, include following variables and functions:
      1. rd, rs, rt: the register no. or the actual value after ID stage
      2. C : shamt, immediate or address
      3. dest , dt : save the destination register no., and which is the source, -1 if none
      4. except : the hazard that the stage faced
      5. the constructor will initial to NOP
      6. getingstuc() will decode current instruction to specific it’s operation,
   3. **Error** print out specific error message if detected,
      1. it’s constructor will fopen() a file named error\_dump.rpt,
      2. a variable exit to show if a halt error occur;
4. Functions :
   1. sign\_extented()
      1. fill the variable with 0 or 1 according to its sign and return it,
   2. output()
      1. Make sure $zero is constant zero,
      2. Print out all register value(in hex), cycle number and PC value(in hex),
      3. Print out the instruction of each stage and their hazard message,
   3. IF()
      1. If PC is smaller than iniPC, take NOP,
      2. Read the instruction from iimage[] according to PC,
   4. ID()
      1. Specify each variable’s value according to the instruction, replace rd, rs, rt with actual value, by either forwarding or from relative regis[]
      2. sign\_extend() c if needed,
      3. beq, bne, bgtz :
         1. if the condition is met, add PC by C + 4,
      4. j :
         1. Add PC by 4 first, and keep [31:28] only (by operator &)
         2. Multiple C by 4 (by <<2), replace [27:0] of PC (by operator |),
      5. jal : save [31:28] of PC to $31 ($ra), then do the same as j,
      6. jr : assign PC as $ra,
   5. EX()
      1. add, addu : add directory, overflow detection in add,
      2. sub: transfer $rt to ~($rt+1), then do the same as add,
      3. addi, addiu: add the value, overflow detection in addi,
      4. and, or, xor, nor, nand : can be done directory by operator,
      5. andi, ori, nori : can be done directory,
      6. lw, lh, lhu, lb, lbu, sw, sh, sb : compute the address same as addi did
      7. slt, slti :
         1. compare the sign bit, result is known if different,
         2. otherwise, just compare the value, the smaller is the smaller one (since I store all value as unsigned),
      8. sll, srl, lui : use shift operator directory,
      9. sra :
         1. use shift operator as srl
         2. take the result after sign\_extend() it,
   6. MEM()
      1. lw :
         1. detect address overflow and misaligned(whether address is multiple of 4)
         2. assign the value to the specific register,
      2. lh, lhu :
         1. detect address overflow and data misaligned(whether address is multiple of 2)
         2. assign the value to the specific register
         3. sign\_extend() for lh,
      3. lb, lbu :
         1. detect address overflow
         2. assign the value to the specific register
         3. sign\_extend() for lb,
      4. sw :
         1. detect address overflow and data misaligned(whether address is multiple of 4)
         2. assign the value to the specific address in dimage,
      5. sh :
         1. detect address overflow and detect Data misaligned(whether it’s multiple of 2)
         2. assign the value to the specific dimage and make sure the other half word of the dimage is constant (by operator >>, <<, |),
      6. sb :
         1. detect address overflow
         2. assign the value to the specific address in dimage and make sure the other part of the dimage is constant (by operator >>, <<, |),
   7. WB()
      1. Detected write to 0 error
      2. assign the source value to the specific regis[], according to wb->dest,
   8. hazard\_detect()
      1. if id encounter data hazard from EX\_MEM, it can forward only when id is running branch instruction, or stall otherwise
      2. if id encounter data hazard from EX\_MEM, id->except will store the value that it needs forward in EX stage
      3. If will stall whenever id stalls
      4. Pre-do branch instruction to know if branch will be taken in next cycle, if do, If must flush;
5. In main(),
   1. Preparation:
      1. Utilize class Readfile to get input data, assign SP to regis[29] ($SP),
      2. Open a file name snapshot.rpt,
      3. New all stages’ register to an instance(which will be NOP by default),
   2. A while loop that will run to at most 500,000 cycles, or terminate if any 5 consecutive halt signal been detected:
      1. Forward value to ex if needed,
      2. Run WB(), MEM(), EX(), respectively,
      3. Switch each stage-register pointer to point to next instruction that will be deal in next cycle,
      4. Ex will point to new Decode() if id is stalled, ID() and IF() will be run otherwise
      5. If If should be flush, id will point to new Decode(), or point to If otherwise,
      6. Use id->getinstruct() to know decode the new instructiong in order to do hazard detect,
      7. Run hazard\_detect(),
      8. Use error.Exit() to know whether to end the program,
      9. Use output() to record current status and what to do in next cycle,
      10. Add cycle by 1, and PC by 4 so that the simulator will execute next line in iimage,
   3. Fclose() snapshot.rpt, error\_dumt.rpt will be closed when destructor of Error is execute;
6. Special cases:
   1. Ignore NOP while detecting write to 0 error,
   2. How to detect number overflow?
      1. Use a variable to save the sign of two variable that going to be add if they’re same
      2. If the sign of the result is different to the variable, imply that number overflow has occur;
7. **Test case Design**
8. dimage.bin
   * 1. Random create other value;
9. iimage.bin
10. load a few word to register first before do any operation,
11. 6 data hazard solve by stall (insert NOP),
12. 2 data hazard solve by forwarding,
13. 1 control hazard (flush),
14. create address overflow and Data misaligned in the last line of instruction;
15. Use the same function that I use to read inputs in simulator to transfer my iimage.bin and dimage.bin to small-endian.