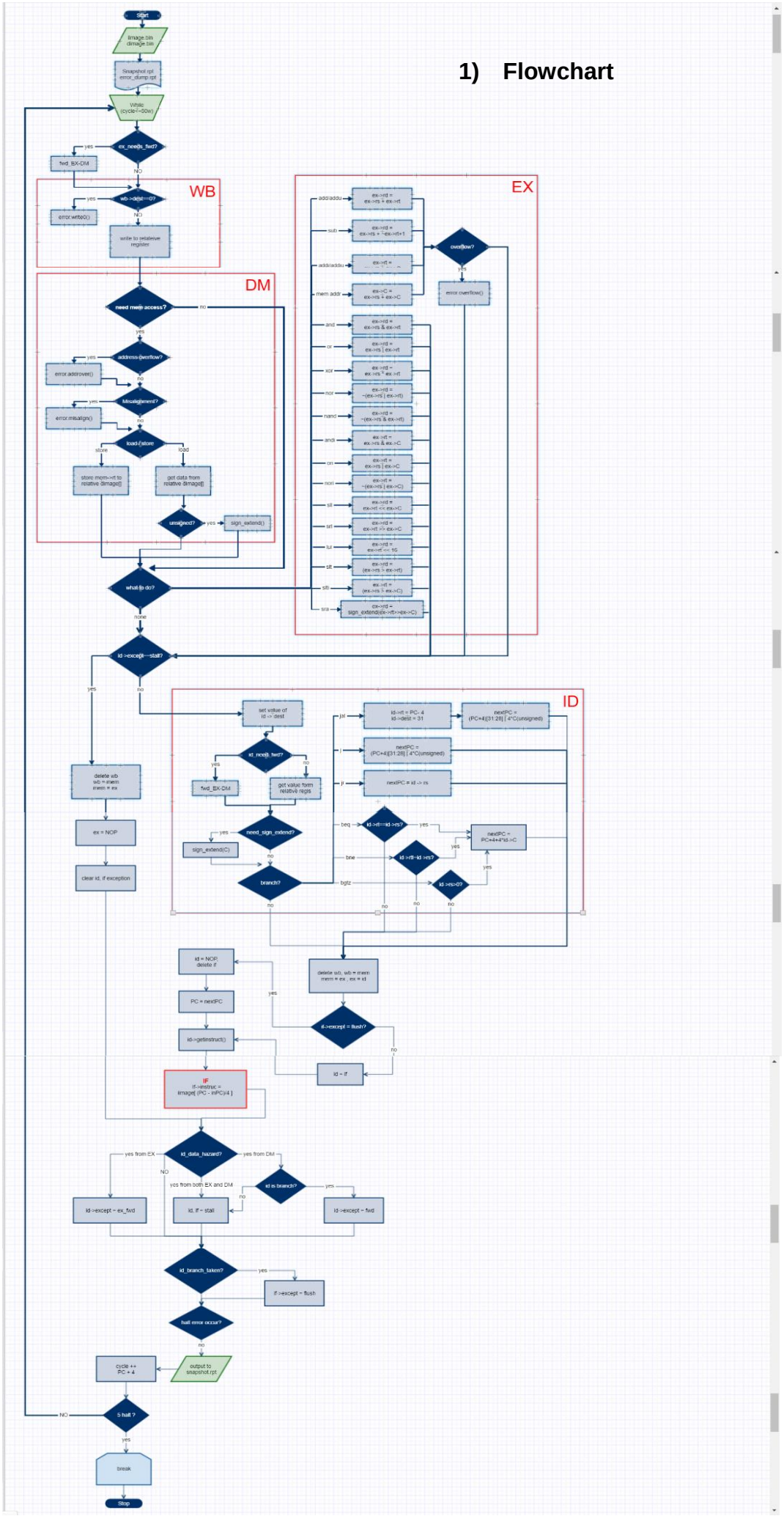
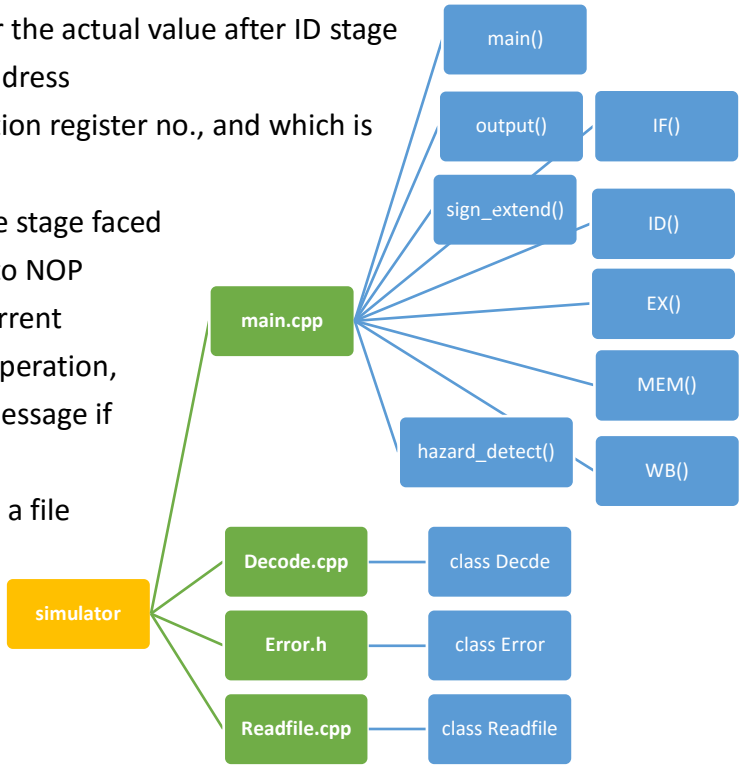
1) Flowchart


Start
image.bin
dimage.bin
Snapshot.rpt
error_dump.rpt
While
(cycle<=50w)
ex_needs_fwd?
fwd_EX-DM
WB
wb->dest==0?
error:write0()
write to relateive register
DM
need mem access?
address overflow?
error:addrover()
Misalignment?
error:misalign()
load/store
store
load
store mem->rt to relative dimage[]
get data from relative dimage[]
unsigned?
sign_extend()
EX
add/addu
sub
add/addu
mem addr
and
or
xor
nor
nand
andi
ori
nori
sll
srl
lui
slt
slti
sra
overflow?
error:overflow()
what to do?
none
id->except==stall?
ID
set value of id -> dest
id_needs_fwd?
fwd_EX-DM
get value form relative regis
need_sign_extend?
sign_extend(C)
branch?
jal
j
jr
beq
bne
bgtz
id->rt==id->rs?
id->rt!=id ->rs?
id->rs>=0?
nextPC = (PC+4)[31:28] | 4*C(unsigned)
nextPC = PC+4+4*id->C
delete wb
wb = mem
mem = ex
ex = NOP
clear id, if exception
id = NOP
delete if
PC = nextPC
id->getinstruct()
if->except = flush?
delete wb, wb = mem
mem = ex , ex = id
id = if
IF
if->instruc = imagef((PC - initPC)/4)
id_data_hazard?
yes from EX
yes from DM
yes from both EX and DM
id is branch?
id->except = ex_fwd
id_if = stall
id->except = fwd
id_branch_taken?
if->except = flush
halt error occur?
output to snapshot.rpt
cycle ++
PC + 4
S halt ?
break
Stop

2) Project Description

1. Variables :
   a. regis[32], represents as registers and initial to 0 (unsigned),
   b. iimage[256], represents as instruction-memory and initial to 0 (unsigned),
   c. dimage[256], represents as D-memory and initial to 0 (unsigned),
   d. PC, represents the address of current instruction,
   e. If, id, ex, mem, and wb (pointers), represent each stage's register;

2. Classes :
   a. **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],
   b. **Decode** is the class for stages' register, include following variables and functions:
      i. rd, rs, rt: the register no. or the actual value after ID stage
      ii. C : shamt, immediate or address
      iii. dest , dt : save the destination register no., and which is the source, -1 if none
      iv. except : the hazard that the stage faced
      v. the constructor will initial to NOP
      vi. getingstuc() will decode current instruction to specific it's operation,
   c. **Error** print out specific error message if detected,
      i. it's constructor will fopen() a file named error_dump.rpt,
      ii. a variable exit to show if a halt error occur;



3. Functions :
   a. sign_extented()
      i. fill the variable with 0 or 1 according to its sign and return it,
   b. output()
      i. Make sure $zero is constant zero,
      ii. Print out all register value(in hex), cycle number and PC value(in hex),
      iii. Print out the instruction of each stage and their hazard message,
   c. IF()
      i. If PC is smaller than iniPC, take NOP,
      ii. Read the instruction from iimage[] according to PC,

d. ID()
    i. Specify each variable's value according to the instruction, replace rd, rs, rt with actual value, by either forwarding or from relative regis[]
    ii. sign_extend() c if needed,
    iii. beq, bne, bgtz :
        1. if the condition is met, add PC by C + 4,
    iv. 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 |),
    v. jal : save [31:28] of PC to $31 ($ra), then do the same as j,
    vi. jr : assign PC as $ra,
e. EX()
    i. add, addu : add directory, overflow detection in add,
    ii. sub: transfer $rt to ~($rt+1), then do the same as add,
    iii. addi, addiu: add the value, overflow detection in addi,
    iv. and, or, xor, nor, nand : can be done directory by operator,
    v. andi, ori, nori : can be done directory,
    vi. lw, lh, lhu, lb, lbu, sw, sh, sb : compute the address same as addi did
    vii. 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),
    viii. sll, srl, lui : use shift operator directory,
    ix. sra :
        1. use shift operator as srl
        2. take the result after sign_extend() it,
f. MEM()
    i. lw :
        1. detect address overflow and misaligned(whether address is multiple of 4)
        2. assign the value to the specific register,
    ii. 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,
    iii. lb, lbu :
        1. detect address overflow

2. assign the value to the specific register
3. sign_extend() for lb,

iv. sw :
1. detect address overflow and data misaligned(whether address is multiple of 4)
2. assign the value to the specific address in dimage,

v. 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 >>, <<, |),

vi. 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 >>, <<, |),

g. WB()
i. Detected write to 0 error
ii. assign the source value to the specific regis[], according to wb->dest,

h. hazard_detect()
i. if id encounter data hazard from EX_MEM, it can forward only when id is running branch instruction, or stall otherwise
ii. if id encounter data hazard from EX_MEM, id->except will store the value that it needs forward in EX stage
iii. If will stall whenever id stalls
iv. Pre-do branch instruction to know if branch will be taken in next cycle, if do, If must flush;

4. In main(),
a. Preparation:
i. Utilize class Readfile to get input data, assign SP to regis[29] ($SP),
ii. Open a file name snapshot.rpt,
iii. New all stages' register to an instance(which will be NOP by default),

b. A while loop that will run to at most 500,000 cycles, or terminate if any 5 consecutive halt signal been detected:
i. Forward value to ex if needed,
ii. Run WB(), MEM(), EX(), respectively,
iii. Switch each stage-register pointer to point to next instruction that will be deal in next cycle,
iv. Ex will point to new Decode() if id is stalled, ID() and IF() will be run

otherwise

v. If If should be flush, id will point to new Decode(), or point to If otherwise,
vi. Use id->getinstruct() to know decode the new instructiong in order to do hazard detect,
vii. Run hazard_detect(),
viii. Use error.Exit() to know whether to end the program,
ix. Use output() to record current status and what to do in next cycle,
x. Add cycle by 1, and PC by 4 so that the simulator will execute next line in iimage,

c. Fclose() snapshot.rpt, error_dumt.rpt will be closed when destructor of Error is execute;

5. Special cases:
   a. Ignore NOP while detecting write to 0 error,
   b. How to detect number overflow?
      i. Use a variable to save the sign of two variable that going to be add if they're same
      ii. If the sign of the result is different to the variable, imply that number overflow has occur;

3) Test case Design

1. dimage.bin
   a. Random create other value;

2. iimage.bin
   a. load a few word to register first before do any operation,
   b. 6 data hazard solve by stall (insert NOP),
   c. 2 data hazard solve by forwarding,
   d. 1 control hazard (flush),
   e. create address overflow and Data misaligned in the last line of instruction;

3. Use the same function that I use to read inputs in simulator to transfer my iimage.bin and dimage.bin to small-endian.

```
    lw   $16, 0($0)
    addi $18, $16, 5
    lw   $12, 4($0)
    addi $18, $18, 1
    ori  $15, $1, 1
    addi $18, $16, 1
    ori  $17, $18, 1
    lh   $1, $18, 2
    sub  $2, $7, $18
    sub  $0, $8, $7
    lw   $12, 28($0)
    sw   $9, 4($0)
    lb   $10, 0($29)
    lw   $0, 0($0)
    addi $2, $10, 1
    subi $5, $2, -5
    nori $18, $18, 5
    lui  $18, $5, 5
    addi $29, $29, -4
    sb   $30, 1($0)
    lw   $4, 4($0)
    j    TO
    sw   $2, 4($0)
    xor  $30 0($29)
    addiu $29, $29, 4
    lw   $3, 4($0)
TO  sll  $0, $0, 0
    addi $3, $0, 12
    lw   $20, 28($0)
    add  $5, $20, $3
    add  $20,$5, $5
    lw   $0, -3($0)
    halt
    halt
    halt
    halt
    halt
```