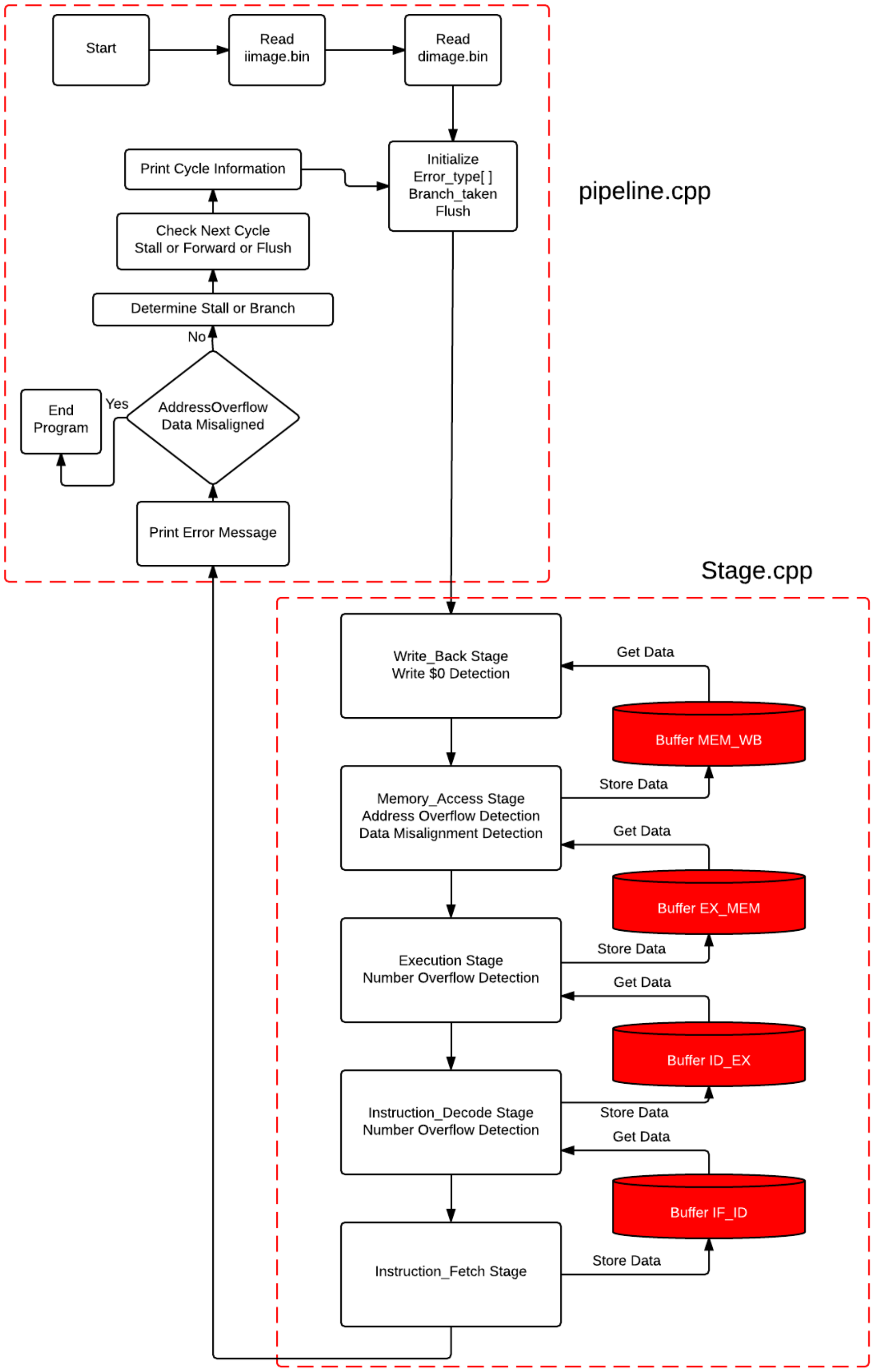
103062331 Archi_Project1_Report

1) Project Description

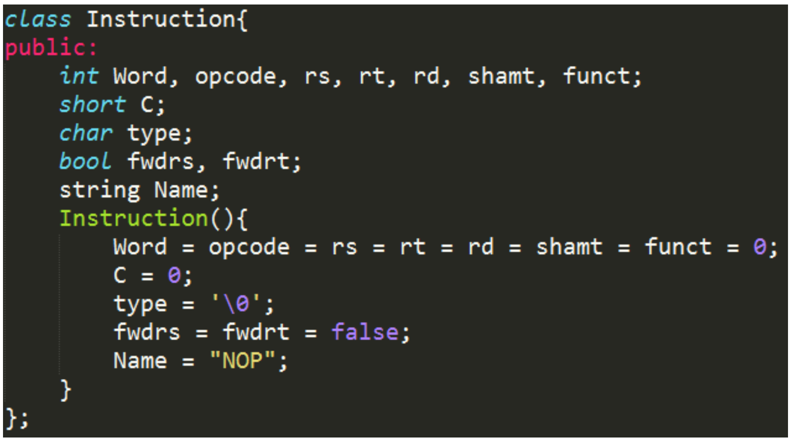
1-1) Program Flow Chart



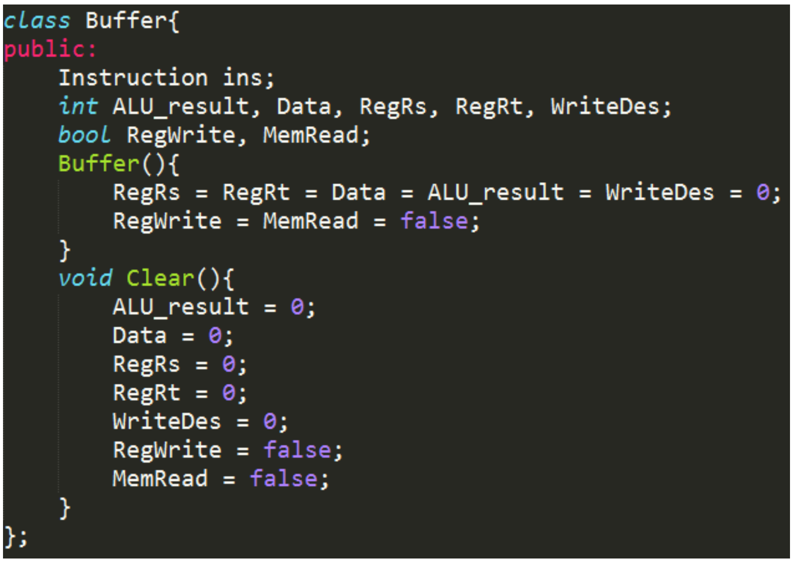
Start
Read iimage.bin
Read dimage.bin
Initialize Error_type[ ] Branch_taken Flush
Print Cycle Information
Check Next Cycle Stall or Forward or Flush
Determine Stall or Branch
No
AddressOverflow Data Misaligned
Yes
End Program
Print Error Message
pipeline.cpp
Stage.cpp
Write_Back Stage Write $0 Detection
Get Data
Buffer MEM_WB
Store Data
Memory_Access Stage Address Overflow Detection Data Misalignment Detection
Get Data
Buffer EX_MEM
Store Data
Execution Stage Number Overflow Detection
Get Data
Buffer ID_EX
Store Data
Instruction_Decode Stage Number Overflow Detection
Get Data
Buffer IF_ID
Store Data
Instruction_Fetch Stage

1-2) Detailed Description

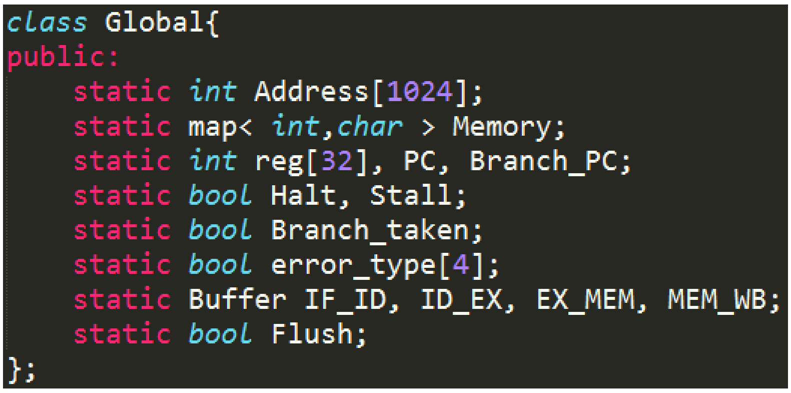
1. Classes Define



```
class Instruction{
public:
    int Word, opcode, rs, rt, rd, shamt, funct;
    short C;
    char type;
    bool fwdrs, fwdrt;
    string Name;
    Instruction(){
        Word = opcode = rs = rt = rd = shamt = funct = 0;
        C = 0;
        type = '\0';
        fwdrs = fwdrt = false;
        Name = "NOP";
    }
};
```



```
class Buffer{
public:
    Instruction ins;
    int ALU_result, Data, RegRs, RegRt, WriteDes;
    bool RegWrite, MemRead;
    Buffer(){
        RegRs = RegRt = Data = ALU_result = WriteDes = 0;
        RegWrite = MemRead = false;
    }
    void Clear(){
        ALU_result = 0;
        Data = 0;
        RegRs = 0;
        RegRt = 0;
        WriteDes = 0;
        RegWrite = false;
        MemRead = false;
    }
};
```



```
class Global{
public:
    static int Address[1024];
    static map< int,char > Memory;
    static int reg[32], PC, Branch_PC;
    static bool Halt, Stall;
    static bool Branch_taken;
    static bool error_type[4];
    static Buffer IF_ID, ID_EX, EX_MEM, MEM_WB;
    static bool Flush;
};
```

Memory (Store Data), reg[32] for 32 registers

PC for Program Counter, Branch_PC for PC to be branched

Halt for ending the program, Stall for whether to stall

Branch_taken for whether to branch.

error_type[4] for four types of errors.

2. Read iimage.bin

   Int Word (Store Instruction). Use get() to get a char each time

   Word = (Word << 8) | (unsigned char) ch ; (Four times to get a 32-bit)

   Store Program Counter into PC

   Store Word into (map<int, int> Address), which is easier to access.

3. Read dimage.bin

   The same as (2.)

   Store Stack Pointer into reg[29] ($sp)

   Store Data into (map<int, char> Memory), which is easier to access.

4. Write_Back Stage

   Take instruction from MEM_WB buffer.

   Write data back to register.

5. Memory_Access Stage

   Take instruction from EX_MEM buffer.

   Access the memory to get data and store data

   Store data and instruction into MEM_WB beffer.

6. Execution Stage

   Take instruction from ID_EX buffer.

   Use ALU to do calculation.

   Store data and instruction into EX_MEM beffer.

7. Instruction_Decode Stage

   Take instruction from IF_ID buffer.

   Decode the instruction.

   Store data and instruction into ID_EX beffer.

8. Instruction_Fetch Stage

   Fetch instruction from Address[PC]

   Store data and instruction into IF_ID beffer.

9. Print Error Messages

   Use for loop to check error_type[0 to 3], if error occurs, output the error messages.

10. Check Halt

    If Halt is true, end the program.

    Else check next cycle’s stage whether to Stall or Forward or Flush

    Output the register and PC status at this cycle, and continue.

11. Close File

Close snapshot.rpt and error_dump.rpt.

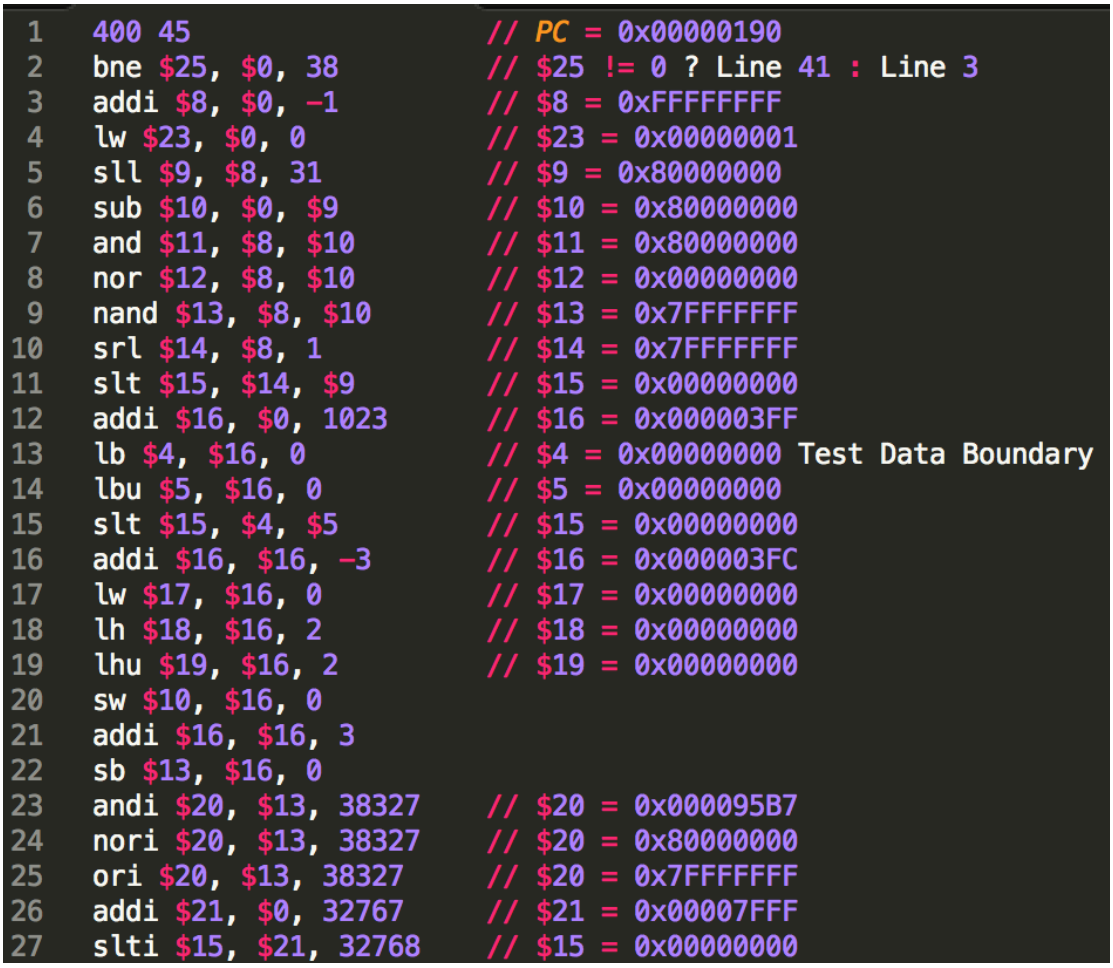
2) Test case Design

2-1) Detail Description of Test case

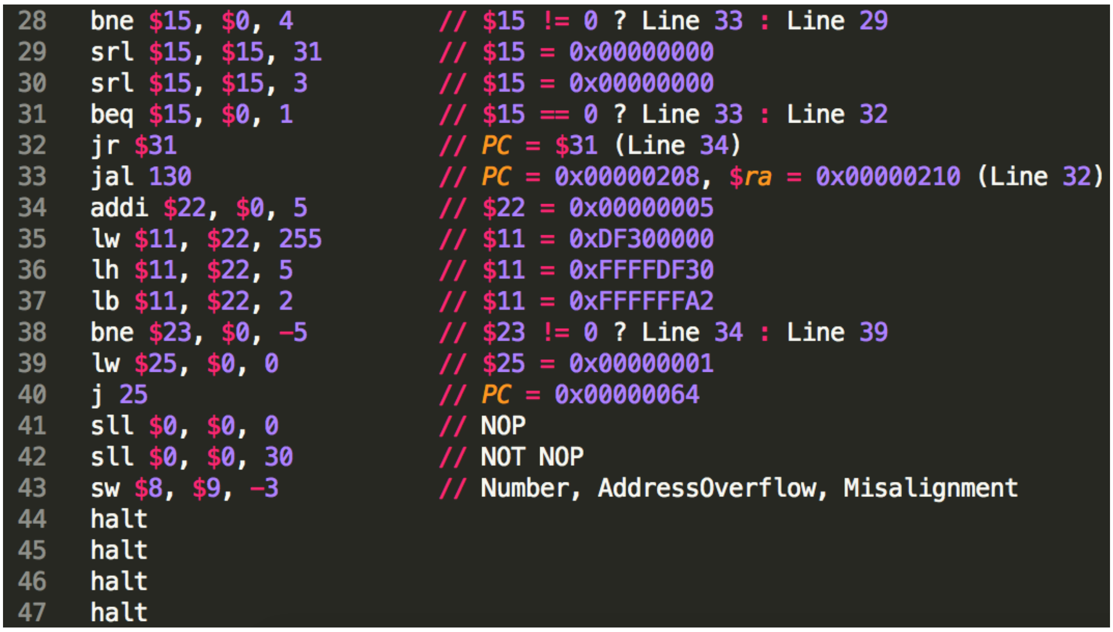
Basically, my test case will test every function except bgtz.

And test Write $0, Number Overflow, Address overflow, Data Misalignment

I'll show it as a graph step by step.



```
 1  400 45                   // PC = 0x00000190
 2  bne $25, $0, 38          // $25 != 0 ? Line 41 : Line 3
 3  addi $8, $0, -1          // $8 = 0xFFFFFFFF
 4  lw $23, $0, 0            // $23 = 0x00000001
 5  sll $9, $8, 31           // $9 = 0x80000000
 6  sub $10, $0, $9          // $10 = 0x80000000
 7  and $11, $8, $10         // $11 = 0x80000000
 8  nor $12, $8, $10         // $12 = 0x00000000
 9  nand $13, $8, $10        // $13 = 0x7FFFFFFF
10  srl $14, $8, 1           // $14 = 0x7FFFFFFF
11  slt $15, $14, $9         // $15 = 0x00000000
12  addi $16, $0, 1023       // $16 = 0x000003FF
13  lb $4, $16, 0            // $4 = 0x00000000 Test Data Boundary
14  lbu $5, $16, 0           // $5 = 0x00000000
15  slt $15, $4, $5          // $15 = 0x00000000
16  addi $16, $16, -3        // $16 = 0x000003FC
17  lw $17, $16, 0           // $17 = 0x00000000
18  lh $18, $16, 2           // $18 = 0x00000000
19  lhu $19, $16, 2          // $19 = 0x00000000
20  sw $10, $16, 0
21  addi $16, $16, 3
22  sb $13, $16, 0
23  andi $20, $13, 38327     // $20 = 0x000095B7
24  nori $20, $13, 38327     // $20 = 0x80000000
25  ori $20, $13, 38327      // $20 = 0x7FFFFFFF
26  addi $21, $0, 32767      // $21 = 0x00007FFF
27  slti $15, $21, 32768     // $15 = 0x00000000
```



```
28   bne $15, $0, 4          // $15 != 0 ? Line 33 : Line 29
29   srl $15, $15, 31        // $15 = 0x00000000
30   srl $15, $15, 3         // $15 = 0x00000000
31   beq $15, $0, 1          // $15 == 0 ? Line 33 : Line 32
32   jr $31                  // PC = $31 (Line 34)
33   jal 130                 // PC = 0x00000208, $ra = 0x00000210 (Line 32)
34   addi $22, $0, 5         // $22 = 0x00000005
35   lw $11, $22, 255        // $11 = 0xDF300000
36   lh $11, $22, 5          // $11 = 0xFFFFDF30
37   lb $11, $22, 2          // $11 = 0xFFFFFFA2
38   bne $23, $0, -5         // $23 != 0 ? Line 34 : Line 39
39   lw $25, $0, 0           // $25 = 0x00000001
40   j 25                    // PC = 0x00000064
41   sll $0, $0, 0           // NOP
42   sll $0, $0, 30          // NOT NOP
43   sw $8, $9, -3           // Number, AddressOverflow, Misalignment
44   halt
45   halt
46   halt
47   halt
```

About dimage.bin :

I randomly generate the data inside it (1024 Bytes)

I just modify some specific position which I will use in the test case.