Computer Architecture Project3

student name : 김대한
student id : 201711020
prof. : 김대훈

Project goal

MIPS 인스트럭션을 실행할 수 있는 MIPS 에뮬레이터에 pipeline 기능을 확장한다. 총 다섯 단계의 pipeline step이 존재하며 각각 IF(Instruction Fetch), ID(Instruction Decoding), EX(Excution), MEM(Memory), WB(Write back)이다. pipeline 과정에서 발생하는 hazard를 막고 각 단계 사이에 state register를 구현해야 한다. input으로는 MIPS 기반의 바이너리 object file(*.o)을 받으며 특정 플래그를 설정할 수 있다. 입력된 바이너리를 통해 각 인스트럭션은 pipeline system에서 일련의 과정을 수행하고 register 및 data memory 수정 및 업데이트한다.

Environment

wsl2 ubuntu 20.04.4

Execution

gcc -o runfile file_name(*.c)로 컴파일한 뒤 ./runfile <antp or atp> [-m addr1:addr2] [-d] [-p] [n- num_instruction] <input file> 명령어 수행. <>의 경우 프로그램 수행에 필수적인 옵션이다. 필수 옵션을 명시하지 않았을 경우 실행오류가 발생한다. [] 경우 선택옵션으로 명시하지 않았을 경우 기본 값으로 실행된다. 아래는 옵션에 대한 설명이다.

- -atp : Always Taken분기 예측을 사용한다.
- -antp : Always Not Taken 분기 예측을 사용한다.
- -m : 메모리 주소 범위(addr1~addr2) 내용을 출력하며 시작주소는 text section의 경우 0x400000, data section의 경우 0x10000000이다.
- -d : 는 한 인스트럭션이 수행될 때마다 모든 레지스터의 값을 출력한다.
- -p : 각파이프라인 단계에서 실행뙤고 있는 instruction의 PC를 출력한다.
- -n : num_instruction 개수만큼 인스터럭션이 수행된다.

Code Description

//memory
struct INSTM.(char inst[33], int loc) : instruction memory
struct DATAM.(char inst[33), int loc) : data memory

//state register
struct IF_ID_stateReg : IF/ID stage State Register
struct ID_EX_stateReg : ID/EX stage State Register
struct EX_MEM_stateReg : EX/MEM stage State Register
struct MEM_WB_stateReg : MEM/WB stage State Register

//value container
struct HAZARD_control : container for control signal for hazard
struct ALU_VALUES : ALU UNIT input type

//type changer
char* toBi(int i, int size) : change the int type decimal to binary str
int signExtend() : change the imm in IF/ID state register

//file loader
void preprocess() : load the instruction and data to memory

//Unit
void controlUnit() : get instr[31-26] from the IF/ID set the opcode and ALU control signal
void hazardUnit_ID() : Hazard Unit in ID stage
void hazardUnit_EX() : Hazard Unit in EX stage/?
ALU_VAL forwardUnit() : Forward Unit in Ex stage
void ALU(ALU_VAL alu_val) : ALU

//pipeline stage
void IFetch() : fetch the one text memory instruction
void Dec() : decode the fetched instruction and read the registor
void Exec() : execute mathamatic operation with ALU control signal
void Mem() : access the memory for load and store
void WB() : write the data to register
void update(): update the state register input to output

//print code
void memState(int m1, int m2) : print memory state of m1~m2 addr
void regState() : print register state
void pipeState() : print pipeline state

State Register

```
struct IF_ID_stateReg {
        int pPC; //PC for pipeline
        int PC; //real PC
        //inst info
        char op[7] ; //inst[31-26]
        char rs[6] ;//inst[25-21]
        char rt[6] ; //inst[20-16]
        char rd[6] ; //inst[15-11]
        char sh[6] ; //inst[11-6]
        char fc[7] ; //inst[5-0]
        char im[17] ; //inst[15-0]
        char tg[27] ; //inst[25-0]
        int stall; //for stall

struct ID_EX_stateReg {
        int pPC; //pipleline PC
        int PC; //pc + 4
        int jump; //jump pc
        int branch; //branch pc
        //data
        int read_data1 ; //rs
        int read_data2 ; //rt, rs
        int write_addr1; //rt
        int write_addr2; //rd
        int rs; //for forwarding check
        int rt; //for forwarding check
        int imm //imm;
        //control info
        int ALUOP;//func code
        int Jump; //j, jr, jal
        int Branch; //beq, bne
        int PCSrc; //whether jump to the branch PC or not
        int MemRead; //read Mem, lw, lb
        int MemtoReg; //send Mem data, lw, lb
        int MemWrite; //write mem, sw, sb
        int ALUSrc; // choice alu source between reg_data with imm
        int RegWrite; //write reg
        int RegDst; //choise write reg between rt and rd
        int Shift; //Shift operation check
        int Lui; //Lui operation check
        int stall; //for stall
}
```

```
struct EX_MEM_stateReg{
        int pPC; //pipleline PC
        int PC; //pc + 4
        int branch; //pc to brach
        //data
        int read_data2; //data read in the register
        int write_addr; //address of register to write
        int ALU_value; //result of alu exectutation
        int ALU_zero; //for branch judgement
        //control info
        int Branch;
        int RegWrite;
        int MemRead;
        int MemtoReg;
        int MemWrite;
}


struct MEM_WB_stateReg {
        int pPC; //pipleline PC
        int PC; //pc + 4
        int read_data; //alu_value, or data from memory and so on...
        int write_addr; //
        int ALU_value; //result of ALU operation
        //control info
        int MemtoReg;
        int RegWrite;
}
```

Main Flow

1. Instruction Fetch , IF()

in real process there are lots of computation cost to access the instruction data. but in this project, access cost is neglectable. So pre-decoding of the instruction is executed for the convenience.

2. Instruction Decode , ID()

in this stage, get the decoded instruction at the Control Unit which is determined the control signal. And then access the register for bring the value for operation. moreover, get the jump and branch address from the instruction and PC. condition of the control hazard(antp, atp) some of the function is added. for the prevent the **simultaneous register access**, write process has priority to the read process by ordering IF, WB, DEC EX, MEM in the code. By using this displacement prevents the structure hazard of the register.

3. Executaion, EX()

this stage register get the EX stage value which executed the arithmetic operation at the ALU. some of the state register value is follow up to the next state register and some of is neglected. before the caculation, lastest value is updated by using forward unit. and the case of the control signal operation is performed various way.

4. Memory MEM()

at this stage access the data memory by using the alu result(ALU_value). and then get or wirte the data. Also, branch result is judged by hazard unit. depending on the hazard some of the instruction is stalled or flushed.

5. Write Back, WB()

this stager is generally considered last stage but for implementation this stage is conducted before the ID(). update the register value.

6. update()

every state register input value is updated by update() function. this Approch of implementation is able to spacial separation between each sate register

Hazard

in the program totally ‘5’ control hazards are exist. at the case of L(lw, lb) hazard, value for ALU input is comfirmed after MEM stage. so we need to add one stall at ID stage and replace the ID/EX state register to noop. in case of J(jump) hazrd, address to jump is confirmed after ID stage so next instruction is flushed. in case of B(branch atp) hazrd, predictor predict branch is move to branch PC. similar to J hazard, next instructionis flushed and banch PC instruction is conducted. Bf1(branch fail antp) hazard, Bf2(branch fail, atp) hazard is the case branch prediction is fail so 3 instruction is flushed after the branch instruction.