-Computer Architecture (COL 216)

Assignment 8: Simulating a Pipelined Processor

ENTRY NO. 2018CS10354_2018CS10348

THE FILES:

1. data_mem.cpp :

It consists two function :

```
void store(int position,struct data_memory *dm,char* val_name,int val)
```

: This function takes three argument according to position it stores the value(i.e val)

In the data_memory.

```
int get_location(char* var_name, struct data_memory *dm)
```

: This function takes two arguments and according to them it finds the var_name in data_memory and when it matches it returns that index value.

2. functions.cpp :

```
int reading(FILE *file,struct inst_mem*im,struct data_memory *dm)
```

: This function separates the code in instruction format and store in the instruction memory

3. inst_mem.cpp :

: This is the core part of the whole program where actually we are doing the whole backend part(pipelining).

So at first when there is a function “execute” which we call it from the main function after having the instruction in the instruction memory.

```
void execute(struct inst_mem *im, int k, struct data_memory *dm)
```

: it takes three arguments the first one is an instruction array ,the total number of instructions and the initialised data memory.

→First we initialise the pc with the value=0;cycle=0

→Then using the while loop we iterated through the array till we did not reach the last instruction.

→So after entering in the loop we call the “decode” function passing the argument instruction memory and the data memory.Then we reach to decode function and then inside the decode function first we check if there is any stall or not(we had defined a variable of type int which take care of whether there is stall=0 or stall=1).if we find there is no any stall the we continue the process,that is we first call IF() function which is the function equivalent to the theory of pipelining(the instruction fetching state),its work is to fetch instruction from instruction memory then we proceeds to the next function which is ID() which is the instruction decoding state which decodes the fetched instruction and fetches new instruction.then we move to EXE() function which execute the decoded function and decodes the previously fetched instruction,then we move to MEM() function which place the executed data to their respectives memory address / we fetch the data from the given address from memory along with it we do the the execution of previously decoded instruction.the we move to the final WB() function which write back the data to register and along with it also do memory part for the previous instruction.

The functionality of pipelining:

- We had created 8 registers file extra for the pipelining process.using struct “`struct pipeline”` it has 6 attributes inside this struct which is instruction which it will hold the,the pc value,the npc value,the data it might can hold(A/B),LMD(load or store value),alu_output.

  ```
  struct pipline IF_ID;
  ```

```
struct pipline ID_EX;

struct pipline EX_MEM;

struct pipline MEM_WB;

struct pipline deposit_IF_ID;

struct pipline deposit_ID_EX;

struct pipline deposit_EX_MEM;

struct pipline deposit_MEM_WB;
```

: These are the 8 registers that we are using for pipelining.

HOW ITS IS WORKING:

Step 1.

→when we call ID() function it sets the IF_ID.pc=current pc and IF_ID.npc=npc

Step 2.

→we reach here after the ID() function hence in the IF_ID will have the instruction so now when we are inside this ID() function according to theory here the instruction will get decoded so for that purpose for storing the previously fetched instruction we had defined the depost_IF_ID register.so when we are inside the ID() function we will decode the instruction which is in the deposit_IF_ID register and along with this we will put this deposit_IF_ID to the ID_EX register this means we have did the decoded part after it we will update the deposit_IF_ID to the newly received instruction[second part of the job of ID() stage that is fetch the new instruction].

Step 3.

→when we reach here, that is the EX() stage we will do the execution of previously decoded instruction which will be in the deposit_ID_EX register.same time we will also update the ID_EX=deposit_ID_EX [means one part of its job that is executed the instruction] now deposit_ID_EX will change to ID_EX means [second part of the job that is decode the previously fetched instruction].

Step 4.

→when we are in a MEM() function here we will store or load from memory according to the required action regulated by the type of instruction we will have here. So from deposit_EX_MEM we will get the instruction and the data/address which will be needed accordingly to load/store in the memory,and we will update the MEM_WB to deposit_EX_MEM[first part of job is done here of MEM() stage].Then it change the deposit_EX_MEM = EX_MEM [second part of the job is done for MEM() stage].

Step 5.

→when we are here means we are in final stage that is WB() stage,here we will write back the value to corresponding given register,for that we will be having the data in the deposit_MEM_WB,from this register we will do the write-back part which is the first job of this part (here we will have the final instruction of any instruction).Second part of its job is to change deposit_MEM_WB=MEM_WB(updation is just indication that second part is done).

THIS WHOLE STEP WILL EXECUTE IN ONE CYCLE,AND IN EACH CYCLE WE KNOW EVERY STAGES WORKS SO TO PASS THAT INSTRUCTION(RESULT) IN THE NEXT STAGE IN THE NEXT CYCLE WE WILL HAVE TO HOLD THE RESULT(MEANS EVERY STAGE HAS DEFINED JOB AND COMPLETING THAT JOB MEANS : RESULT HERE) OF EVERY STAGES FOR THAT PURPOSE WE HAD USE THE "deposit_if-id, deposit_id-ex, deposit_ex_mem, deposit_mem_wb" REGISTERS.

4. main.cpp :

```
int main(int argc, char *argv[])
```

: Here we are taking the file name from the command in which the code is written in mips language.

When it takes the input file first thing it do is call the reading function which had two things to do first is converting the code input file into the format in which all the data and main part gets separated along with labels,and the second part it do is return the no of line of the corresponding instruction along with it also stores the separated instruction to instruction memory.

After this we call execute function from here which executes the instruction according to the code provided.

5. operations.cpp :

```
void add(int rd,int reg1, int reg2)

void sub(int rd,int reg1, int reg2)

void and_(int rd,int reg1, int reg2)

void or_(int rd,int reg1, int reg2)

void slt(int rd,int reg1, int reg2)

void li(int dest,int val)
```

: These are similar type of function it takes the input and do required arithmetic and stores

In the register which is globally defined so that it can be reached from every function.

```
void lw(int dest,int base, int addr,struct data_memory *dm )
```

```
void sw(int dest,int base, int addr ,struct data_memory *dm)
```

: These two are similar type of function takes three arguments (dest,base,addr,and the data memory) do the arithmetic that is addr+base and if the function is “lw” then from the data_memory using that addr+base collects the data and assign it in the register at the corresponding index that is dest.

Similarly for “sw” from registerfile it receives the data using the dest index and stores it int the data_memory at the corresponding index that is base+addr.

```
void beq(int reg1,int reg2,int pc_dest)

void bne(int reg1,int reg2,int pc_dest)

void bgtz(int reg1,int pc_dest)

void bltz(int reg1,int pc_dest)

void jump(int num)
```

: These are the functions which take input do the arithmetic and check whether it is true or not suppose in beq and bne if it true the changes the pc to that pc_dest.(pc is similar to a program counter).

6. reg_file.cpp :

```
struct reg register_file[32]
```

: This is the structure we had defined which is an array of length 32.Inside it consists of two attributes: the char array for storing the name of the register and second a int for storing the data value at that register.

```
void init_reg_file()
```

: This is just a void function which initiates the register file with their corresponding name that we use in mips language like - “t1” “a0” etc.

```
int reg_num(char* first_name)
```

: This is the function which takes one argument that will be the name of the register for which it will provide the index of that register in the register_file array.

7. all.h :

: It is just the header file where we have defined the functions name that is getting used in other file.so to make us able to use those functions in the other file like inst_mem.cpp and so on.i have included this file in each component so wherever we might require the functions we directly will able to call that function only not we have to make that function again.

HOW WE HANDLE THE HAZARDS USING THE STALL METHOD:

Since we are having the deposit_IF_ID, deposit_ID_EX, deposit_EX_MEM, deposit_MEM_WB registers so when we are in ID() function we will be decoding the instruction so what we are doing here is if find that the current decoded instruction is related or not with the instruction which will be executing in the same cycle. So that instruction we are getting from the deposit_ID_EX[which will be executed in the same cycle] if the decoded instruction matches with the instruction which will be executing in the same cycle then we assign stall=1(a globally defined int variable).

LW/SW STALL:

So when next cycle comes, before starting execution of stages we had implemented if case whether there is stall==1 or not if it finds so, then we simply call only MEM(),WB() and again WB()[so to completely execute the instruction which where in execution and memory stages],and then assign some nop value to deposit_EX_MEM, and then we resume the normal execution starting from IF() to WB() functions.

BRANCHING STALL:

These cases we are handling in execution part, when we are in execution we will the current executing instruction so we check where it is simply jump or bltz or bgtz or bne if satisfies the condition we assign stall_branch=1(int variable define globally),also update the npc according to it.

So when next cycle come we first check stall_branch=1(we are written two if case one for lw/sw and the other one for branching) if it founds true means stall_branch==1 then we fill the deposit_IF_ID, deposit_ID_EX to nop because it will having the instruction of the consecutive next line so by assigning nop will overwrite the data and hence we can execute from the branched instruction.So after if case we normally resume the function like it will then execute IF(),ID(),EX(),MEM(),WB().