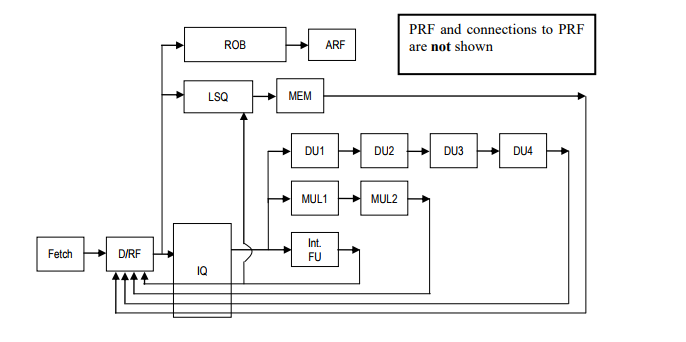
**DESIGN DOCUMENT FOR APEX PIPELINE**

This project implements the simulator for an out-of-order processor:

****

**Module 1 :-**

**Initialization**

After reading the set of Instructions from the text file all the instructions with its PC address are stored in following struct :-

struct instruction

{

char instruction\_string[100];

int line\_number;

int address;

};

Then stage\_info for all the stages, Register File, Code Memory, Data Memory base address and data fields are initialized.

struct flags

{

char zero[10];

char carry;

char Negative;

char psw\_flag[10];

};

struct stats

{

int cycle;

};

struct instruction

{

char instruction\_string[100];

int line\_number;

int address;

};

struct register1

{

int value;

char status[10];

};

struct register\_file

{

char registers[4];

struct register1 regtr;

};

struct stage

{

struct instruction\_info input\_instruction;

struct instruction\_info output\_instruction;

char stalled[10];

};

struct data\_memory {

int mem\_address;

int mem\_value;

char mem\_char;

};

**Module 2 :-**

**Simulation**

In the simulator loop stages are called in following order :-

issue\_que();

lsq();

robfr();

rename\_display();

decode();

execute\_div1();

execute\_div2();

execute\_div3();

execute\_div4();

execute\_mul1();

execute\_mul2();

execute();

memory();

Instruction will stop flowing to fetch stage till the instruction counter is reached.

After calling all the stages, cycle is incremented in simulator method after every loop. After incrementing the cycle all values which are in the input stage are transferred in output stage for that particular stage, so that in next cycle the next stage can pick all these values from output of previous stage and stored in the input buffer of current stage and can process the same.

**Fetch Stage :-**

Fetches the first instruction to start the Apex pipeline implementation. Next instructions will keep on coming in next cycle as along as fetch stage is not stalled or execution is completed. Fetch Stage will be stalled if decoded and execution stages are stalled.

**Decode Stage, IQ and LSQ :-**

Decode Input instruction fetches the first instruction in second cycle from Output of fetch instruction and it keep doing the same in every cycle as along as Decode is not stalled or execution is completed. In decode stage the instruction string which is fetched from fetch stage is decoded and Source Register, Destination Register, Literal Value. Op code all are stored in following Struct :-

struct stage

{

struct instruction\_info input\_instruction;

struct instruction\_info output\_instruction;

char stalled[10];

};

struct instruction\_info

{

int pc\_value;

char instruction\_str[100];

int source\_reg1;

int source\_reg2;

int dest\_reg1;

int dest\_reg2;

int target\_memory\_addr;

int target\_memory\_data;

int source\_reg1\_value;

int source\_reg2\_value;

int dest\_reg1\_value;

int dest\_reg2\_value;

int dest\_reg\_value;

int literal\_value;

char opcode[20];

};

In following pipeline no instruction waits in Decode stage unless Issue Queue is full or Load Store queue is full or ROB is full or there is no Physical Registers are available. Instruction waits for there source register to be valid in Issue Queue. Issue Queue is added between function unit and Decode Stage. All the Instruction are Moved to Issue Queue, ROB and Load store along with Issue Queue and ROB are moved to Load Store Queue from Decode Stage. Register Renaming is done for every Destination register of a instruction and values from physical register are committed to architecture when Instruction is at the head of the queue. There is a rename table to keep a track which physical register is mapped with which Architecture register. For every control Flow instructions unique CFID tag is added if that instruction gets flushed preceding instructions with that tag all are flushed.

Register status and Physical Register Details are stored in following structs:-

struct register1

{

int value;

char status[10];

};

struct phy\_reg

{

int value;

char status[10];

char allocated[10];

char renamed[5];

};

struct phy\_reg\_file

{

char registers[4];

struct phy\_reg phy\_regtr;

};

**Integer FU :-**

All the instruction other than Multiplication, Halt and Divide are executed here, arithmetic calculations for ADD, LOAD, STORE, SUBTRACT, AND, OR, EX-OR are done in these stage. Output for the arithmetic instruction are stored in input of current stage it doesn’t directly updates the destination register, it is done from Head of ROB and calculated values are forwarded using forwarding bus from Last Stage of FU’s for waiting Instructions in Issue Queue, Decode Stage and also Load store along with these two are Forwarded to Waiting instruction in LSQ. At the time of forwarding value two condition are checked that are bypassing and store forwarding. If address of load is different than subsequent valid store in LSQ then Load is bypassed as other register are waiting for the values from the load. If any STORE instruction is invalid then forwarding is not done, now if address is same as the latest store of LSQ and it is valid then load will not have to go to Memory stage it can directly be forwarded to ROB and will be completed.

**Implementation of BZ :-**

BZ checks if the Zero flag is TRUE if it is TRUE it takes branch, only when ZERO flag is TRUE it does branching otherwise BZ is passed from execution without performing any operation. Branching Instruction is decided after adding the PC value of current stage to the literal value of the instruction.

**Implementation of BNZ :-**

BNZ checks if the Zero flag is FALSE if it is FALSE it takes branch, only when ZERO flag is FALSE it does branching otherwise BNZ is passed from execution without performing any operation. Branching Instruction is decided after adding the PC value of current stage to the literal value of the instruction.

**Implementation of JUMP :-**

PC value is calculated by adding the literal value and the value of one source register. This PC value calculated is the address where instruction should get jumped.

**Multiplication FU 1 & 2:-**

In following 2 stages arithmetic calculation is done for Multiplication. Output value is forwarded from last Stage of Multiplication using forwarding bus if issue queue and decode is waiting for that value else it is directly updated in Register file. Value is kept in input of following stage and passed to next stage. It will reflect the register in when that instruction comes to head of ROB.

**Division FU 1, 2, 3 & 4:-**

In following 4 stages arithmetic calculation is done for Division. Output value is forwarded using forwarding bus if decode is waiting for that value else it is directly updated in Register file. Value is kept in input of following stage and passed to next stage. It will reflect the register in Write back stage.

If there is a instruction in Division FU 4 then flag is set by Division FU to notify Multiply FU, Integer FU stage, so that execution for Execution will be stalled for Multiply FU, Integer FU and also for decode and fetch, if the instruction in decode is other that Divide.

Stalled for following stages are released in Memory stage when instruction from Division FU 4 moves to Memory stage in next cycle.

**Memory Stage :-**

Memory Address calculated for STORE in previous stage is directly updated in data memory. Following is the Data Memory where values are stored.

struct data\_memory {

int mem\_address;

int mem\_value;

char mem\_char;

};

For the memory address which is calculated in previous stage for LOAD is used to retrieve the Memory value for that address so that same can be updated in Writeback for that destination Register.

Store and load have to wait in Memory Stage for 3 cycles to perform its operations.

**ROB :-**

Size of ROB is 32 it is First in First out queue. 2 valid instructions are committed at every cycle in program order. After decode step by step instruction are forwarded to ROB. ROB entry is completed from last stage of function unit and it is committed from the head of the ROB.

**Module 3 :-**

**Display**

Final register values and Memory values are displayed, also step by step cycle information is shown.