Name: Nimish Sakalkale
Aniket Rajmane
Prashant Kolhe

APEX PIPELINE SIMULATOR

CS-520

Group Members:

Nimish Sakalkale

Aniket Rajmane

Prashant Kolhe

Name: Nimish Sakalkale
Aniket Rajmane
Prashant Kolhe

Simulator For Apex

Modules Involved

1) Simulator.java
2) Fetch.java
3) Decode.java
4) IQEntry.java
5) IQ.java
6) LSQENtry.java
7) LSQ.java
8) ROBEntry.java
9) ROB.java
10) Int_FU.java
11) MUL_Fu.java
12) LS_Fu_stage_1.java
13) LS_Fu_stage_2.java
14) LS_Fu_stage_3.java
15) MemoryClass.java
16) RegFIleClass.java
17) Utility.java

Overview:

Every object of class represents that entity/stage. Memory has been implemented by array of integers with size of 10000. Register and its valid bits for these registers are implemented using HashMap.

ROB, IQ, LSQ has been implemented using ArrayLists.

Memory entity is represented by MemoryClass.java

Valid bits are used to handle flow dependencies between instructions.

RegFileClass contains structures which handles archRegisterFIle, validBits for physical registers, AliasTable , Allocated Bits for architectural registers , free list and Renamed Bits.

Also walkback method has been implemented to restore alias table entries if branch is taken.

Name: Nimish Sakalkale
Aniket Rajmane
Prashant Kolhe

Assumption:

All the instructions are written in UPPER CASE.

Instructions having lower case letters will not be processed.

It has been assumed that all memory locations as well as all register are initialized to value zero.

Short description of each module

1) Simulator.java
   It contains main function of java program.
   It contains definitions for
   a) Initialize(): It initializes pipeline. Sets program counter to 20,000. And also sets clock to 0.
   b) runCycles (int cycles) : method which simulates the pipeline for given number of cycles. It does the operations of Fetch, Decode, Execute, Mem, WB.
   c) Display() : This function displays contents of all 5 stages as well as indicates whether given stage was stalled at given clock. And display contents of registers and there valid bits. Additionally it also display contents of 100 memory locations.
   d) It also contains functions which calls flush functions of stages.

2) Fetch.java
   It contains member variables which stores current program counter and also currently fetched instruction.
   It contains definitions for:
   1) doFetch(long PC) : It fetches instruction from file which is available at passed program counter. If it contains blank line it treats it as NOP.
   2) getCurrentIstruction() : This function returns instruction which was fetched from memory.
   3) Flush() : it flushes the contents of Fetch stage
   4) Display() : Display contents of fetch stage.

3) Decode.java

   Adds Unused Registers To FreeList.

   Then decodes the contents of lastly fetched instruction into opcode, destination registers and source registers. After decoding, assign Free Register to destination register.

   Then get Source Reg Value either from ROB or Architectural register. Stores them into member variables which serves as latches in pipeline depending on instruction type which

can be inferred from opcode. Additionally it checks valid bits of source registers and if IQ is full or ROB is full, if any of this condition is invalid then it stalls pipeline.IF all above conditions are met then it sets valid bit of destination register to false we insert instruction and decode info in IQ and ROB and proceeds to execution.

It contains member variables to store decoded information of instruction and value of registers.

It contains functions for

a) LoadDataInDecode() : It reads data from fetch stage which needs to be processed in decode stage.
b) doDecode() : It reads data from fetch stage and decodes it as explained in above summary and stores decoded data into member variables.
c) Flush() : it flushes the contents of decode stage
d) Display() : Display contents of decode stage.
e) addUnusedRegistersToFreeList () : If physical register is valid and has not been assigned to any architectural register then add it to Free list.
f) assignFreeReg () : Check if free physical register available and add Entry To alias table. Mark this new physical register as allocated and invalid
g) getSourceRegValue() : While retrieving architectural register value,If value from ROB is valid take ROB value or take value from architectural register file.
h) It also contains getters for member variables.

4) IQEntry.java
Instruction Queue Entry. It defines the format for each entry which will be made in Instruction Queue after decode.
It contains method for
    a) isIQEntryValid(): It check whether source registers are valid or not. If valid returns true otherwise false.
    b) Display(): It displays contents of current IQ entry.

5) IQ.java
Instruction Queue. It contains maximum 6 instructions which are decoded and waiting for execution.
It contains methods for
    a) Enqueue(): It will add an entry into instruction queue.
    b) Dequeue(): It will remove the first instruction from IQ which is ready to execute and pass it to the execution unit.
    c) flushIQEntriesAfterBranchTaken(): This method will flush all entries in Instruction queue after a taken branch

d) broadcast(): It will broadcast result of executed instructions to dependent instructions in the IQ.
e) isEmpty():It returns true if IQ is empty.
f) isFull(): It returns true if IQ is full.
g) display(): It displays the contents of IQ.

6) LSQEntry.java
Load Store Queue Entry. It defines the format for each entry which will be made in Load Store Queue after decode.
It contains method for
a) isLSQEntryValid():It check whether sources of entry are valid or not. If valid returns true otherwise false.
b) Display(): It displays the contents of current LSQ entry.

7) LSQ.java
Load Store Queue. It contains maximum 4 memory instructions which are decoded and waiting for execution.
It contains methods for
a) LSQEnqueue(): It will add an entry into load store queue.
b) LSQDequeue(): It will remove the first valid memory instruction from LSQ which is ready to execute and pass it to the execution unit.
c) addValueInLSQ(): This method will store memory address calculated by the load store execution unit.
d) updateLSQEntry(): This method will mark LSQ entry as completed and update it with memory value.
e) flushLSQEntriesAfterBranchTaken(): This method will flush all entries in load store queue after a taken branch.
f) broadcast(): It will broadcast result of executed instructions to dependent instructions in the LSQ.
g) isEmpty():It returns true if LSQ is empty.
h) isFull(): It returns true if LSQ is full.
i) display(): It displays the contents of LSQ.

8) ROBEntry.java
Reorder Buffer Entry. It defines the format for each entry which will be made in Reorder Buffer after decode.
It contains method for
a) Display(): It displays contents of the current ROB entry.

9) ROB.java

Reorder buffer. It contains maximum 16 instructions which are decoded and waiting for retirement.

It contains method for

a) Enqueue(): It will add an entry into ROB
b) retireValidInstructionsFromHead(): It will remove the first instruction from ROB which has completed its execution and retire it.
c) flushROBEntriesFromBranch(): This method is written to handle branch instructions. This will squash instructions in ROB inserted after branch instruction is resolved as taken.
d) markROBEntryCompleted() : FU will call this to mark execution of that instruction as completed.
e) markROBEntryCompletedForStore() : FU will call this to mark execution of store instruction as completed.
f) getHeadEntry(): This will return instruction at head of ROB.
g) getTail(): This will return instruction at tail of ROB.
h) isROBListEmpty(): This will tell whether ROB list is empty or not.
i) isROBListFull() : This will tell whether ROB list is full or not.
j) getROBEntryValue(String phyReg): This will return value calculated in ROB where provided physical register was destination.
k) isROBEntryValid():This will return whether instruction in ROB is completed or not where provided physical register was destination.
l) getSourceRegValue(): While retrieving architectural register value ,If value from ROB is valid take ROB value or take value from architectural register file.
b) display() : It displays contents of the current ROB.
c) It also contain some getter an setters.

10) INT_Fu.java

Integer Functional Unit. It is responsible for the execution arithmetic, logical and branch instructions. It has latency of 1 cycle.

It contains method for

a) LoadDataInINT_Fu(): It reads the entry from instruction queue which is ready to execute.
b) doINT_Fu(): It perform the actual execution of instruction and stores the result in member variables.
c) flush(): It flushes the contents of this stage.
d) display(): It displays the contents of this functional unit.

11) MUL_Fu.java

Multiply Functional Unit. It is responsible for the execution multiplication instructions. It has latency of 4 cycle.

It contains method for

a) LoadDataInMUL_Fu(): It reads the entry from instruction queue which is ready to execute.
b) doMUL_Fu(): It perform the actual execution of instruction and stores the result in member variables.
c) flush(): It flushes the contents of this stage.
d) display(): It displays the contents of this functional unit.

12) LS_Fu_stage_1.java

It contains methods for

a) LoadDataInLS_Fu_stage_1():It reads data from LSQ entry .
b) doLS_Fu_stage_1():It calls LoadDataInLS_Fu_stage_1() and stores the results into latches.
c) flush(): It flushes the contents of this stage.
d) display(): It displays the contents of this functional unit.

13) LS_Fu_stage_2.java

It contains methods for

a) LoadDataInLS_Fu_stage_2():It reads data from LS_Fu_stage_1 and stores results into latches.
b) doLS_Fu_stage_2():It calls LoadDataInLS_Fu_stage_1() and stores the results into latches.
c) flush(): It flushes the contents of this stage.
d) display(): It displays the contents of this functional unit.

14) LS_Fu_stage_3.java

It contains methods for

a) LoadDataInLS_Fu_stage_3():It reads data from LS_Fu_stage_2 and stores results into latches.
b) doLS_Fu_stage_3():It computes the memory addresses and values depending on whether instruction is LOAD or STORE.
c) flush(): It flushes the contents of this stage.
d) display(): It displays the contents of this functional unit.

15) MemoryClass.java

This class represents memory of 10000 bytes.

It contains methods

a) GetValueFromMemory() : returns value of specified memory location.

b) SetMemoryValue() : sets value of specified memory location to provided value.
c) Display() : displays contents if first 100 memory locations.

16) RegFIleClass.java

This class contains data for register file, special register X , valid bits for architectural registers. It also contains structures which handles archRegisterFIle, validBits for physical registers, AliasTable , Allocacated Bits for architectural registers , free list and Renamed Bits.

This class contains definitions for

a) Display() : displays contents of all the structures specified in above summary.
b) Flush() : rests contents of all structures.
c) Contains getters and setters for member variables.

17) Utility.java

Contains isInteger() function to check whether given string can be parsed into integer or not.
isLoadBeforeStore():- If store is in LSQ, then check if load is earlier than that.
.