Architecture project

**stages: all stages return true/false and exit the loop if false is returned.**

* BTB – Array , has insert and delete . Max size of 16. HashMap – check about size limit.

Implements a random LRU mechanism . Take the value with you to the table of integer reservation station.

* Fetch –
  + Queue of instructions , size of 16. When queue is full , don’t enqueue and don’t increment pc.
  + Find pc in BTB . Update Pc if found. Else increment by 4.
  + Always put taken = false. To the integer reservation table.
* Issue ( not class)
  + If decoded a non valid command , set halt to true and exit the loop.

Log – non valid command on screen.

* + Peek , decode the instruction . If there’s room in the reservation station and ROB dequeue from instruction queue and push to corresponding station. When insterting to reservation station go to Register status table and take value to Vj/Vk or robAddress to Qj/Qk. Also insert to ROB.
  + On reservation station in addition to all prior fields add ROB fields. This is the field which will write into at write stage.
  + Also notify float/integer status table and update relevant register with new rob.
  + Increment ID and add log record.
* Execution –
  + For FP adders/FP mults/ Integer ALU/Load Buf/Store Buf check if there’s an operation which is ready, meaning both values are available. Also, check if the appropriate ALU Is ready, meaning it’s not doing a previous operation. Only then send to the ALU.
  + Hold for each ALU a counter, which will reflect state of alu and how many cycles remain to finish.
  + For an existing running alu , continue the op and decrement the counter. One can only enter the ALU if its count already set to zero.
  + Compute the result in first cycle and hold in result\_ALu’s variable.
  + Search by index (if 2 ops are ready at the same time).
  + On load operations make sure mem unit is available. So store wouldn’t be done as well.
  + When started get record from trace (using id ) and set execution cycle.
* Write result
  + Write result to rob table in the relevant rob index.
  + Notify all reservation station tables that ROBx has a value now which is y.
  + Notify rob. Rob should set value in its row and also search for store ops waiting for the value of the resolved rob.
  + Delete row from reservation station.
  + Do this for all reservation station tables.
  + Get record from trace (using id ) and set write cycle. (if no write is actually done write -1 )
  + If commiting Halt , set halt to true.
  + When rob is notified on branch/jump then Check BTB.:
    - If pc is in BTB and branch is taken – ok.
    - If pc is not in BTB and branch Is not taken - ok.
    - If pc is in BTB and branch isn’t taken then :
      * flush all rows under branch op in ROB (remember that it’s cyclic so one after branch and to the head.)
      * Notify all reservation stations for each rob deleted. That this rob isn’t valid and the whole reservation station needs to be deleted.

Also notify Float/integer stauts tables . If reached a row with the deleted rob, go to rob and find the recent set to this register before the branch.

* + - * Flush instruction queue.
      * Erase row from btb.
    - If pc is not in BTB and branch is taken then :
      * flush all rows under branch op in ROB (remember that it’s cyclic so one after branch and to the head.)
      * Notify all reservation stations for each rob deleted. That this rob isn’t valid and the whole reservation station needs to be deleted.

Also notify Float/integer stauts tables . If reached a row with the deleted rob, go to rob and find the recent set to this register before the branch.

* + - * Flush instruction queue.
      * add row to btb.
* Commit
  + When bit ready in head of the Rob is high , commit.
  + For normal operations just update register (float/integer status table).
  + For store write to memory .
  + Hold counter for number of cycles to write result .One for all except store. Only when 0 advanced to next row. And remove row when done.
  + When committing store make sure Mem unit Is available, only if available store.
  + Get record from trace (using id ) and set commit cycle.
  + After commit advance header.

\*\*\* Every reservation station table will have the following methods:

Insert, Delete, Write to CDB , Get notification on update.

\*\* ROB will have , insert , delete , get notification on update(of value and destination if needed ) . will hold bit ready as well.

Structure:

**BtbRow** :

* PC
* Predicted PC

**IntegerReserveRow**

* Opcode – 4 bit
* ID - int
* Vj- 32 bit
* Vk – 32 btit
* Qj - int
* Qk - int
* ROB - int
* Busy bit - bool
* PC/Address - int
* Taken/NotTaken (bool)

**FpReserveRow**

* Opcode
* ID - int
* Vj - FP
* Vk- FP
* Qj
* Qk
* ROB
* Busy bit

**Fp Status**

* Value - FP
* Rob - short

**Int status**

* Value – int
* Rob - short

**ROB**

* Opcode – 4 bit
* ID - int
* Destination - int
* Value – Object(FP/int/null)
* Ready – (bool) set to true only when value is ready . For all opcodes except store commit when ready is true. For store make sure ready and destination is not null then commit.

**TraceRecord**

* ID
* Instruction (string that represent 8 bytes in hex)
* Cycle\_issued
* Cycle\_exeucted\_start
* Write\_cdb
* Cycle\_commit

**Config**

* **IntDelay**
* **AddDelay**
* **MulDelay**
* **MemDelay**
* **RobEntries**
* **AddNrReservatopm**
* **MulNrReservatopm**
* **IntNrReservatopm**
* **MemNrLoadBuffers**
* **MemNrStoreBuffers**

**Trace**

* **ArrayList of TraceRecord**
* Id – int , starts at 0 . ID generator for the instructions .

**Functions:**

* **Int AddRecord(int Insturction ) – Returns ID.**
* **TraceRecord GetRecord(ID).**

**FileHandler**

**Functions**

* **Config ReadConfig(string cfgFile)**
* **Int[] ReadMainMem(string MemIn)**
* **WriteMemOut(int[] memOut, string fileName)**
* **writeRegInt ( int[] regInt, string FileName)**
* **writeRegOut(float[] regFp, string FileName)**
* **WriteTraceToFile(Trace trace, string FileName)**

**UnitCounter (static)**

* **MemCounter**
* **IntCounter**
* **FpAddCounter**
* **FpMulCounter**

**Utils**

* **Function : AddressToRowNum**

**InstructionContainer**

* **Instruction(int)**
* **Taken (bool)**

**RobQueue**

* **Array of rob**
* **Add,delete(location), flushAfter – all use head and tails pointer on a cyclic array.**

Global

* PC
* Array of int registers status(16)
* Array of float registers status (16)
* Cycle counter (int)
* UnitCounter static object (not defined actually in global)
* Array of Rob in the given size. Implement a cyclic queue with head and tail pointer.
* Queue of instructions (16) – all instructionContainer Type
* Integer reservation station array – of given size
* FP adders reservation station array – of given size
* FP multipliers reservation station array – of given size
* BTB HashMap<int,int> – size 16. Dictionary that has the pc as the key and the pc predicted as the value.
* Main Memory – 1024 rows of 32 bit.
* Halt = 0 (bool) . 0 by default.

**Main**

1. GetConfig and initialize
2. Main loop while(!halt):
   1. Fetch
   2. Issue
   3. Execute
   4. Write CDB
   5. Commit

**Questions :**

1. **Bound on rob and reservation stations depth ? is byte enough (256) . also relevant for reading from the file.**
2. **If two ops are ready to exeute at the same time, which one should we choose? Can we choose by index ? or have to use lru mechanism .**

|  |  |
| --- | --- |
| **Yossi** | **David** |
| **Fetch** | **Trace Class** |
| **Utils** | **Write CDB** |
| **Execution** | **File Handler** |
| **Commit** | **Issue** |
| **ROB queue** |  |
|  |  |