COL718 Assignment - 2

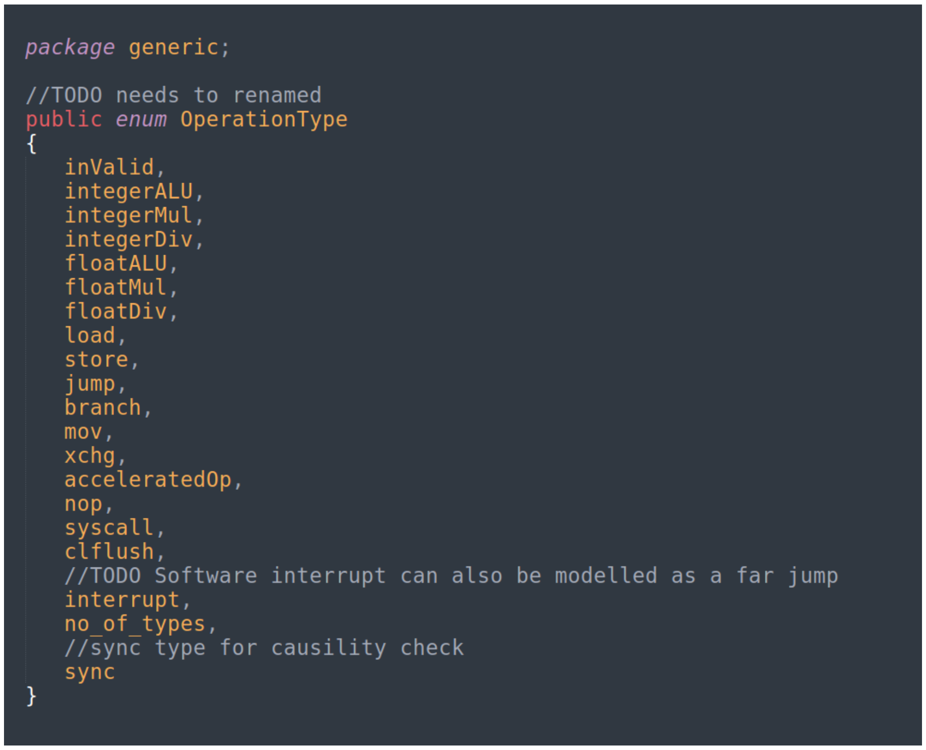
B.V.N.S.Aditya, 2019CS50471
Ch. Sriram Varma, 2019CS50426

Part - 1: System calls

Changes are made in src/simulator folder

Step - 1: Identifying Instruction as System call

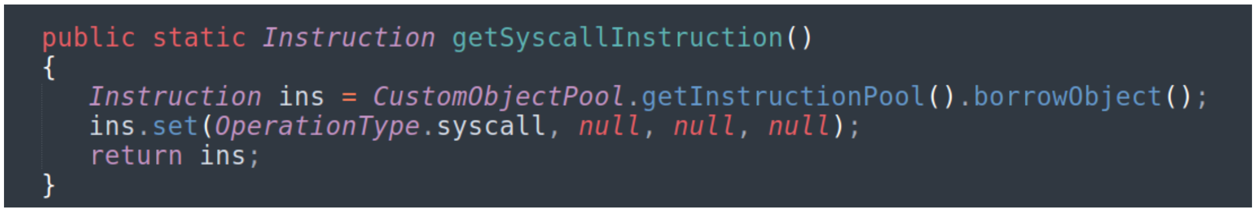
generic/OperationType.java



```java
package generic;

//TODO needs to renamed
public enum OperationType
{
    inValid,
    integerALU,
    integerMul,
    integerDiv,
    floatALU,
    floatMul,
    floatDiv,
    load,
    store,
    jump,
    branch,
    mov,
    xchg,
    acceleratedOp,
    nop,
    syscall,
    clflush,
    //TODO Software interrupt can also be modelled as a far jump
    interrupt,
    no_of_types,
    //sync type for causility check
    sync
}
```

Here, we have added syscall, clflush as Operation types

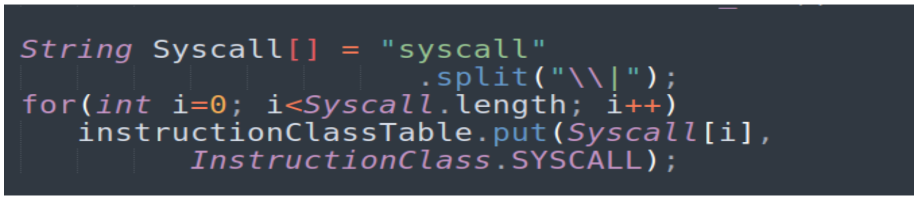
generic/Instruction.java



```java
public static Instruction getSyscallInstruction()
{
    Instruction ins = CustomObjectPool.getInstructionPool().borrowObject();
    ins.set(OperationType.syscall, null, null, null);
    return ins;
}
```

Here, we defined getSyscallInstruction(). Syscall instructions don’t take any operands. So, all the 3 operands are set to null.

emulatorinterface/translator/x86/instruction/InstructionClassTable.java

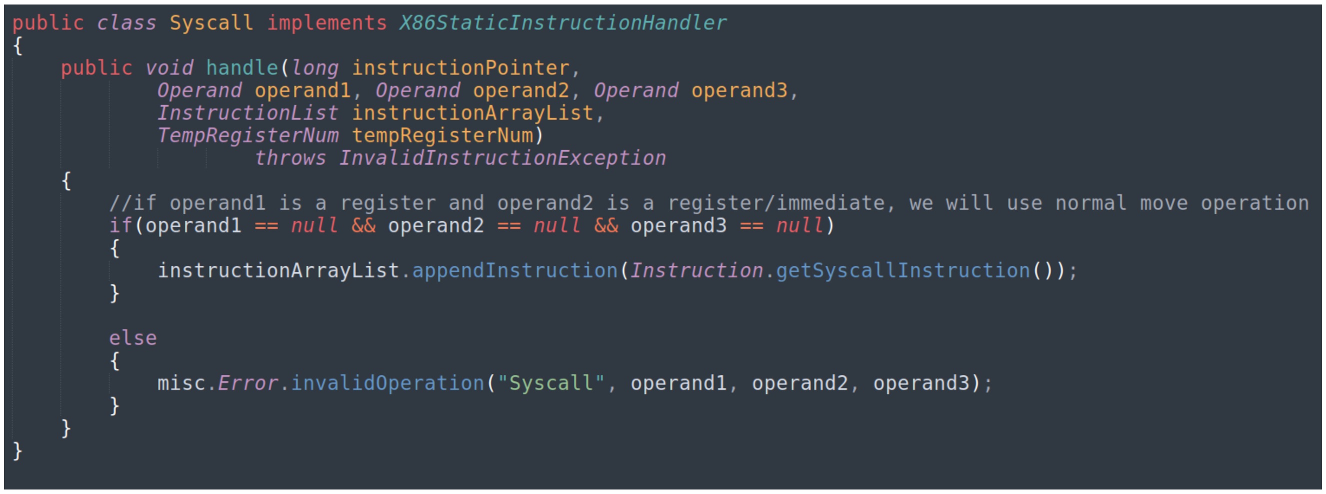


```
String Syscall[] = "syscall"
                        .split("\\|");
for(int i=0; i<Syscall.length; i++)
    instructionClassTable.put(Syscall[i],
            InstructionClass.SYSCALL);
```

Here, instruction will get parsed and it is identified as Syscall. SYSCALL is added as instruction in InstructionClass.java

Step - 2: Handling Syscall Instruction

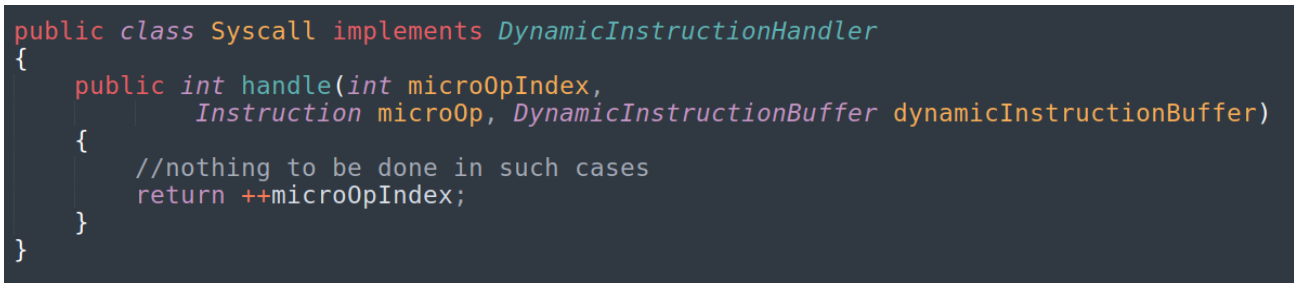
emulatorinterface/translator/x86/instruction/Syscall.java



```
public class Syscall implements X86StaticInstructionHandler
{
    public void handle(long instructionPointer,
            Operand operand1, Operand operand2, Operand operand3,
            InstructionList instructionArrayList,
            TempRegisterNum tempRegisterNum)
                    throws InvalidInstructionException
    {
        //if operand1 is a register and operand2 is a register/immediate, we will use normal move operation
        if(operand1 == null && operand2 == null && operand3 == null)
        {
            instructionArrayList.appendInstruction(Instruction.getSyscallInstruction());
        }

        else
        {
            misc.Error.invalidOperation("Syscall", operand1, operand2, operand3);
        }
    }
}
```

Here, if all 3 operands are null, then Syscall instruction is added to instructionArrayList. If not, it will raise an error

Then, we have changed translator/visaHandler/VisaHandlerSelector.java. In this file, we have created DynamicInstructionHandler for syscall and initialized it to an instance of Syscall(). For this, we have created a file named Syscall.java in the same folder.



```
public class Syscall implements DynamicInstructionHandler
{
    public int handle(int microOpIndex,
            Instruction microOp, DynamicInstructionBuffer dynamicInstructionBuffer)
    {
        //nothing to be done in such cases
        return ++microOpIndex;
    }
}
```

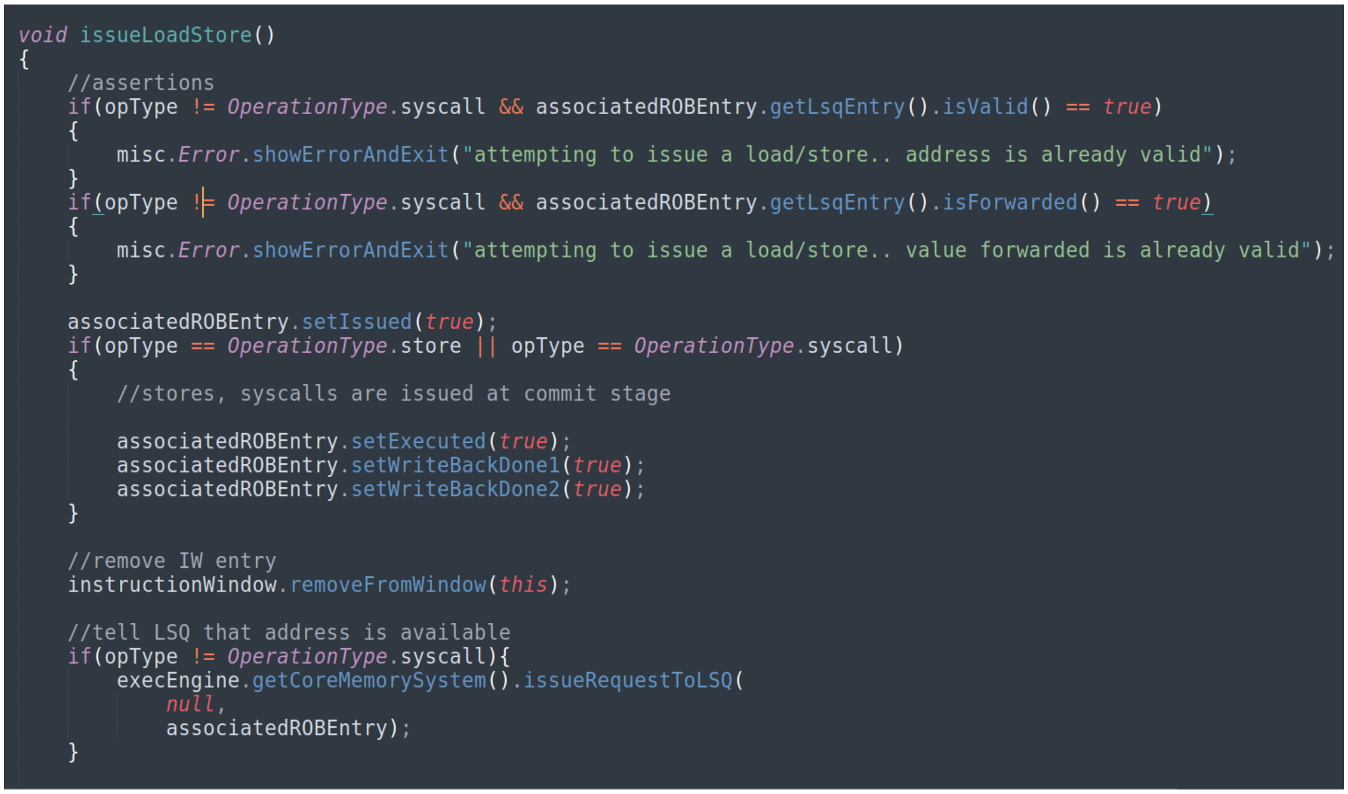
handle() method is defined for this class which handles the visa instruction. This method just increments microOpIndex and returns it.

Step - 3: Executing the instruction

So far, the instruction is identified and handled. Now, to execute it we have to change the pipeline.

There is no need to change fetch and decode units because they are written in generic way and they can fetch and decode syscall instructions as well. Now, syscall instruction should flush TLBs at commit time because of Precise Exceptions. If it flushes before commit time, then the outsider can see that instructions are not executing in program order. “Store” instruction is also executed at commit time. So, we looked at how store instruction is handled and executed in the same way.

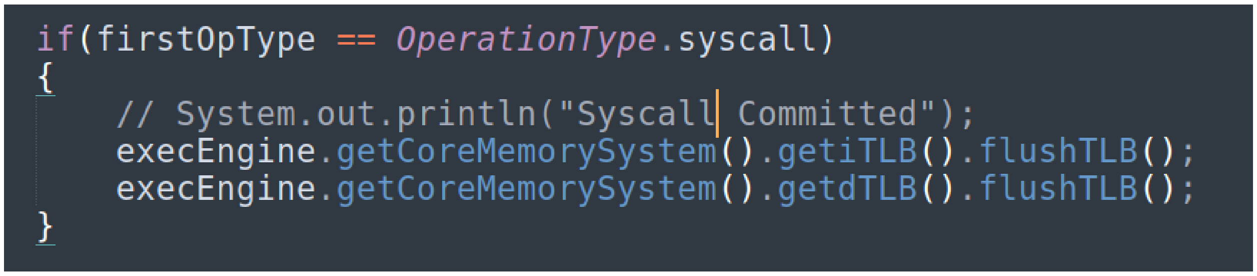
pipeline/outOfOrder/IWEntry.java



```java
void issueLoadStore()
{
    //assertions
    if(opType != OperationType.syscall && associatedROBEntry.getLsqEntry().isValid() == true)
    {
        misc.Error.showErrorAndExit("attempting to issue a load/store.. address is already valid");
    }
    if(opType != OperationType.syscall && associatedROBEntry.getLsqEntry().isForwarded() == true)
    {
        misc.Error.showErrorAndExit("attempting to issue a load/store.. value forwarded is already valid");
    }

    associatedROBEntry.setIssued(true);
    if(opType == OperationType.store || opType == OperationType.syscall)
    {
        //stores, syscalls are issued at commit stage

        associatedROBEntry.setExecuted(true);
        associatedROBEntry.setWriteBackDone1(true);
        associatedROBEntry.setWriteBackDone2(true);
    }

    //remove IW entry
    instructionWindow.removeFromWindow(this);

    //tell LSQ that address is available
    if(opType != OperationType.syscall){
        execEngine.getCoreMemorySystem().issueRequestToLSQ(
            null,
            associatedROBEntry);
    }
```

As syscall is issued similar to loads, we modified “issueLoadStore()” function and “issueOthers()” function. In issueLoadStore function, the ROB entry of syscall instruction is marked as executed. But we are not adding Syscall to LSQ.

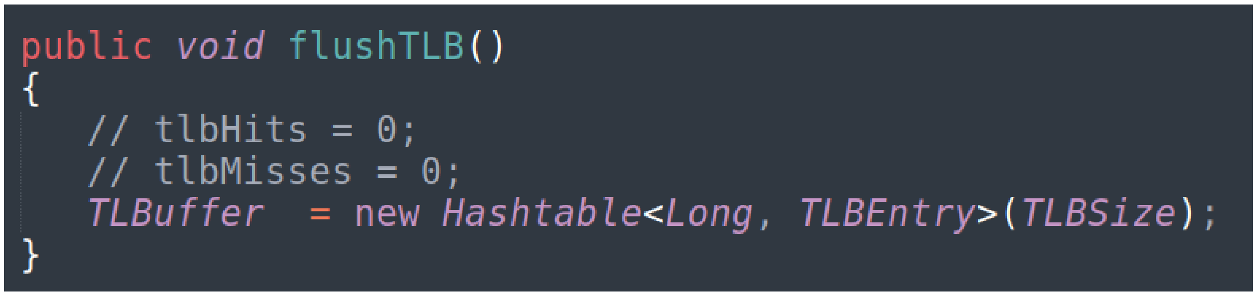
pipeline/outOfOrder/ReorderBufferjava



```java
if(firstOpType == OperationType.syscall)
{
    // System.out.println("Syscall Committed");
    execEngine.getCoreMemorySystem().getiTLB().flushTLB();
    execEngine.getCoreMemorySystem().getdTLB().flushTLB();
}
```

Now, we changed ReorderBuffer.java such that syscall instruction is executed before committing. We added this snippet in the performCommit() function. It will call flushTLB() function written in TLB.java getiTLB, getdTLB are methods for CoreMemorySystem() which returns iTLB, dTLB respectively.

memorysystem/TLB.java



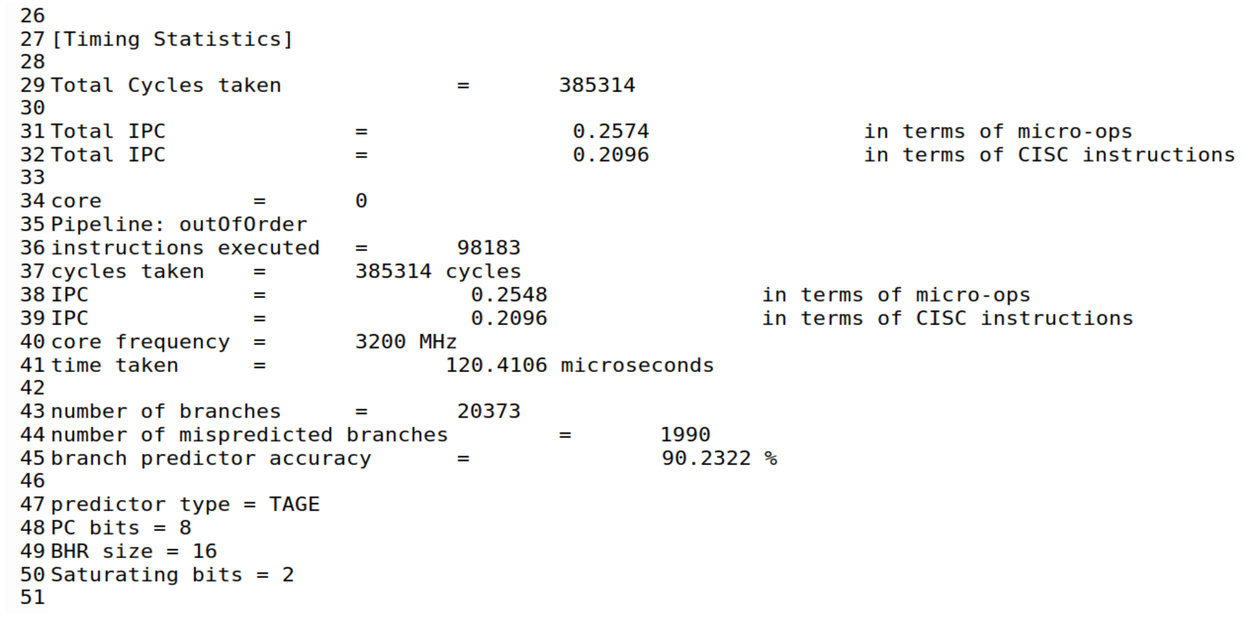
```java
public void flushTLB()
{
   // tlbHits = 0;
   // tlbMisses = 0;
   TLBuffer  = new Hashtable<Long, TLBEntry>(TLBSize);
}
```

TLB entries are stored in TLBuffer, so for flushing the TLB we have to clear TLBuffer. So, it is initialized to a new hashtable. So, all older entries will be removed.

<u>Output:</u>

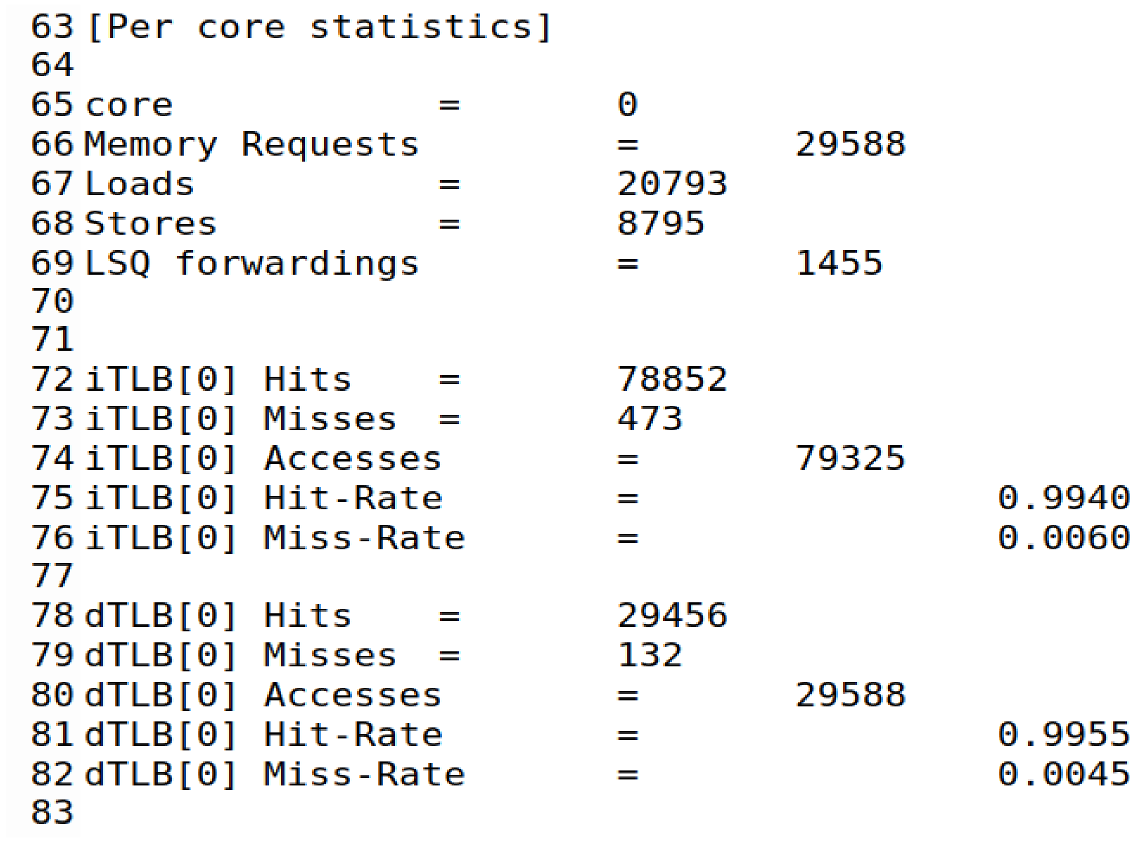
Before Adding Syscall Instruction:

<u>IPC:</u>



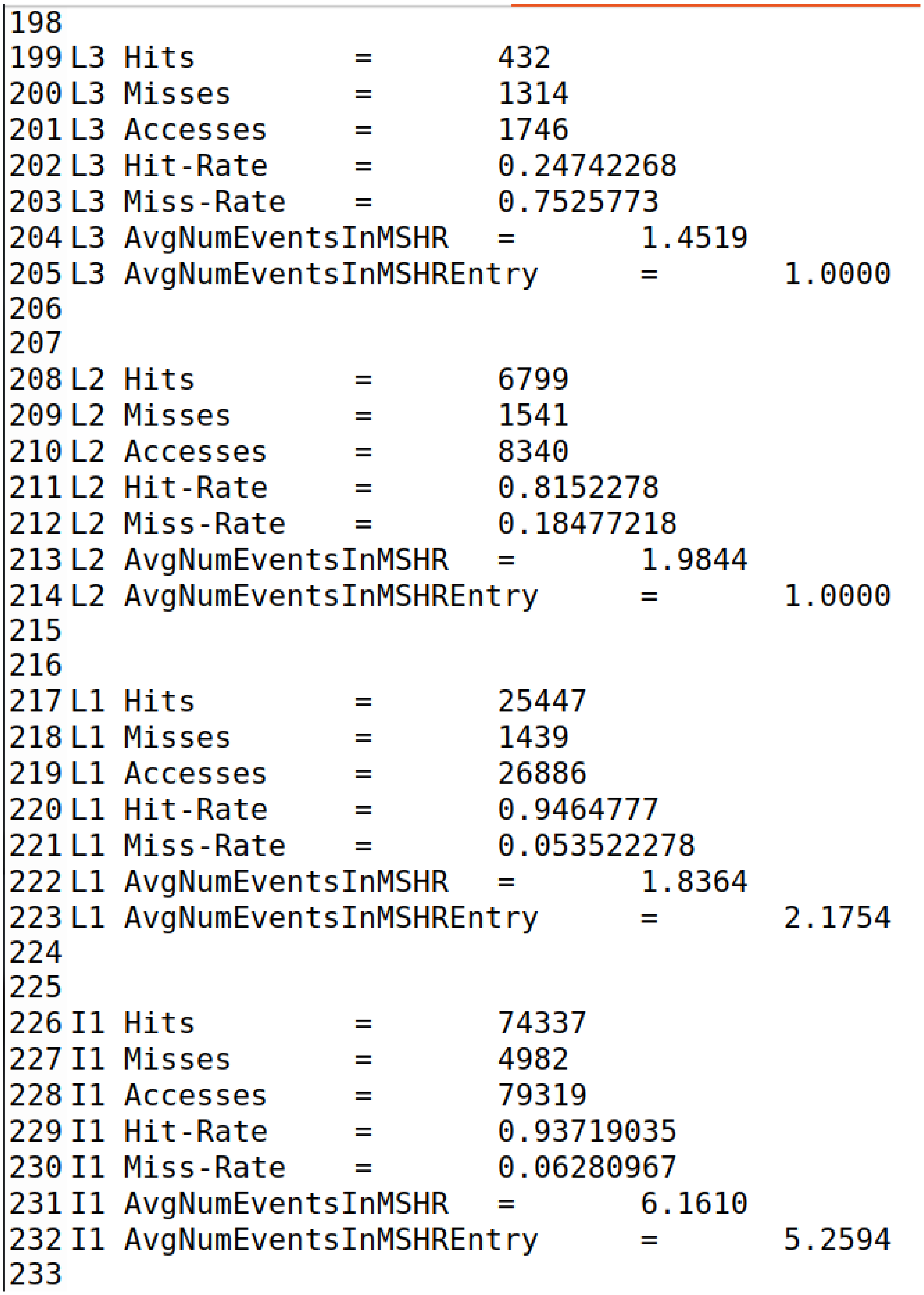
```

[Timing Statistics]

Total Cycles taken              =       385314

Total IPC               =                 0.2574                in terms of micro-ops
Total IPC               =                 0.2096                in terms of CISC instructions

core            =       0
Pipeline: outOfOrder
instructions executed   =       98183
cycles taken    =       385314 cycles
IPC             =                 0.2548                in terms of micro-ops
IPC             =                 0.2096                in terms of CISC instructions
core frequency  =       3200 MHz
time taken      =               120.4106 microseconds

number of branches      =       20373
number of mispredicted branches         =       1990
branch predictor accuracy       =               90.2322 %

predictor type = TAGE
PC bits = 8
BHR size = 16
Saturating bits = 2

```

<u>TLB Stats:</u>



```
[Per core statistics]

core            =       0
Memory Requests         =       29588
Loads           =       20793
Stores          =       8795
LSQ forwardings         =       1455


iTLB[0] Hits    =       78852
iTLB[0] Misses  =       473
iTLB[0] Accesses        =       79325
iTLB[0] Hit-Rate        =               0.9940
iTLB[0] Miss-Rate       =               0.0060

dTLB[0] Hits    =       29456
dTLB[0] Misses  =       132
dTLB[0] Accesses        =       29588
dTLB[0] Hit-Rate        =               0.9955
dTLB[0] Miss-Rate       =               0.0045

```

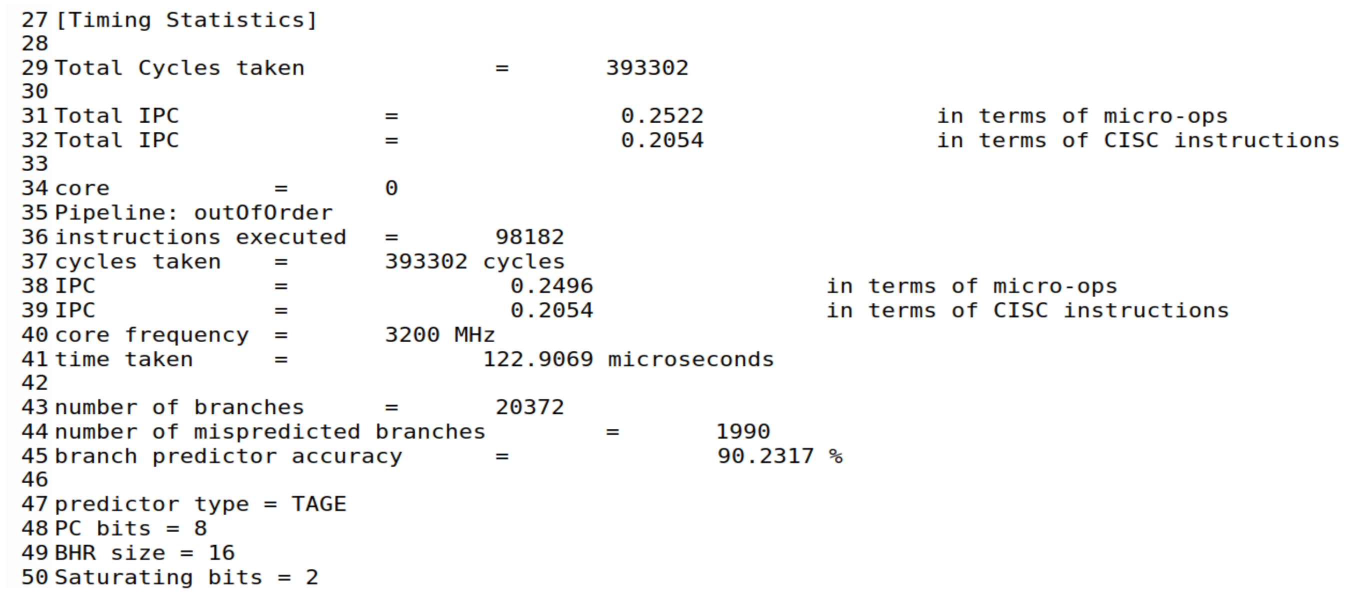
Cache Stats:



```

L3 Hits          =       432
L3 Misses        =       1314
L3 Accesses      =       1746
L3 Hit-Rate      =       0.24742268
L3 Miss-Rate     =       0.7525773
L3 AvgNumEventsInMSHR   =       1.4519
L3 AvgNumEventsInMSHREntry      =       1.0000


L2 Hits          =       6799
L2 Misses        =       1541
L2 Accesses      =       8340
L2 Hit-Rate      =       0.8152278
L2 Miss-Rate     =       0.18477218
L2 AvgNumEventsInMSHR   =       1.9844
L2 AvgNumEventsInMSHREntry      =       1.0000


L1 Hits          =       25447
L1 Misses        =       1439
L1 Accesses      =       26886
L1 Hit-Rate      =       0.9464777
L1 Miss-Rate     =       0.053522278
L1 AvgNumEventsInMSHR   =       1.8364
L1 AvgNumEventsInMSHREntry      =       2.1754


I1 Hits          =       74337
I1 Misses        =       4982
I1 Accesses      =       79319
I1 Hit-Rate      =       0.93719035
I1 Miss-Rate     =       0.06280967
I1 AvgNumEventsInMSHR   =       6.1610
I1 AvgNumEventsInMSHREntry      =       5.2594

```

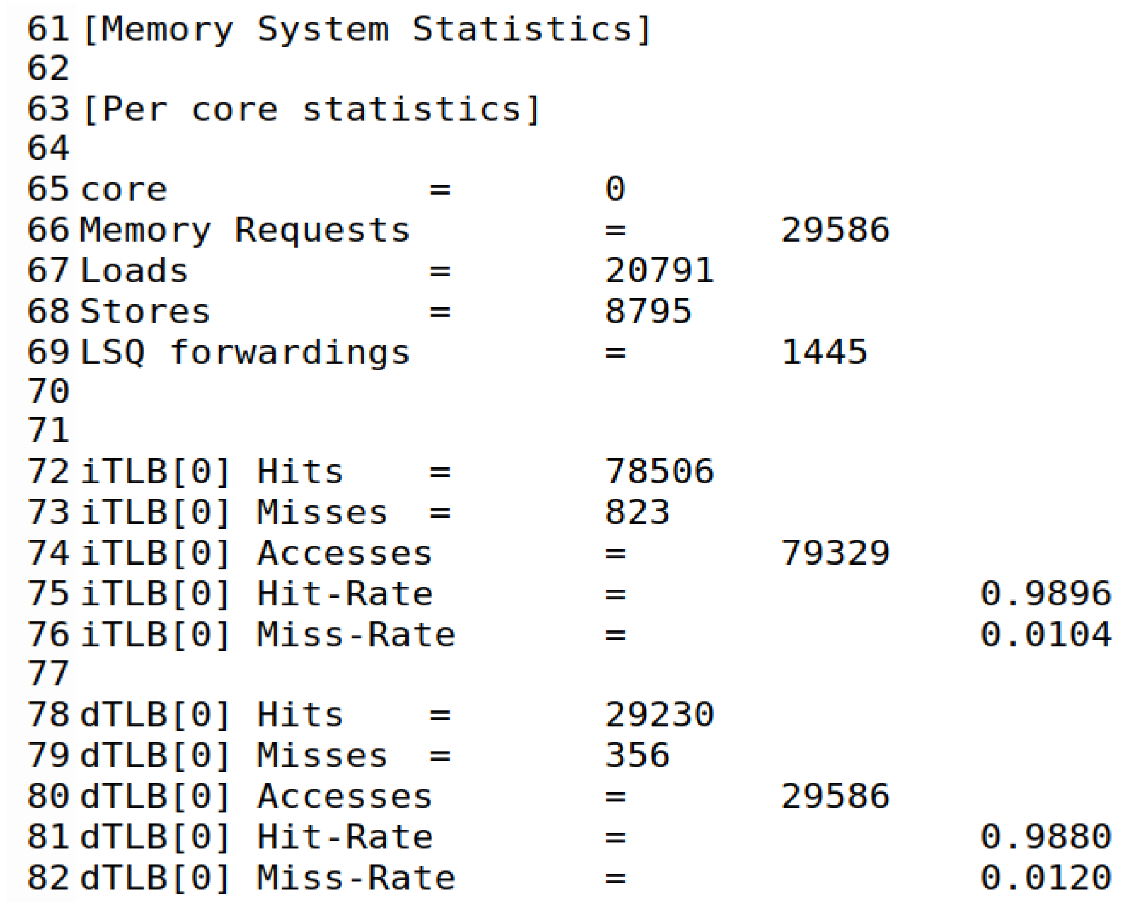
After Adding Syscall Instruction:

IPC:



```
[Timing Statistics]

Total Cycles taken              =       393302

Total IPC               =               0.2522                  in terms of micro-ops
Total IPC               =               0.2054                  in terms of CISC instructions

core            =       0
Pipeline: outOfOrder
instructions executed   =       98182
cycles taken    =       393302 cycles
IPC             =               0.2496                  in terms of micro-ops
IPC             =               0.2054                  in terms of CISC instructions
core frequency  =       3200 MHz
time taken      =               122.9069 microseconds

number of branches      =       20372
number of mispredicted branches         =       1990
branch predictor accuracy       =               90.2317 %

predictor type = TAGE
PC bits = 8
BHR size = 16
Saturating bits = 2
```

TLB Stats:



```
[Memory System Statistics]

[Per core statistics]

core            =       0
Memory Requests         =       29586
Loads           =       20791
Stores          =       8795
LSQ forwardings         =       1445


iTLB[0] Hits    =       78506
iTLB[0] Misses  =       823
iTLB[0] Accesses        =       79329
iTLB[0] Hit-Rate        =               0.9896
iTLB[0] Miss-Rate       =               0.0104

dTLB[0] Hits    =       29230
dTLB[0] Misses  =       356
dTLB[0] Accesses        =       29586
dTLB[0] Hit-Rate        =               0.9880
dTLB[0] Miss-Rate       =               0.0120
```

Cache Stats:

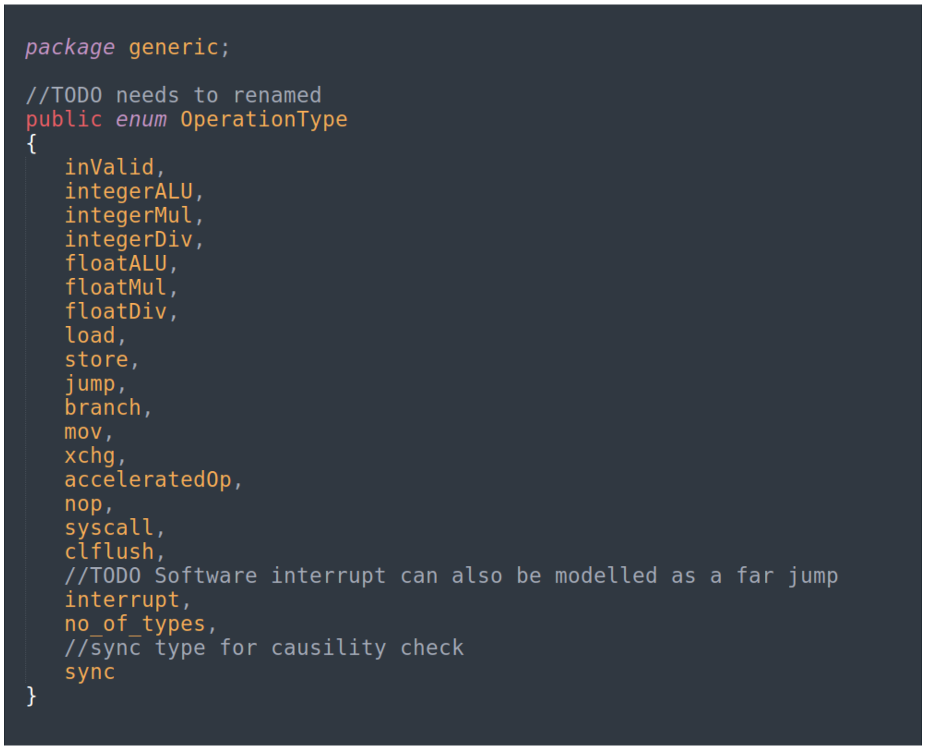
```

L3 Hits        =       430
L3 Misses      =       1314
L3 Accesses    =       1744
L3 Hit-Rate    =       0.24655963
L3 Miss-Rate   =       0.7534404
L3 AvgNumEventsInMSHR   =       1.4300
L3 AvgNumEventsInMSHREntry      =       1.0000


L2 Hits        =       6808
L2 Misses      =       1540
L2 Accesses    =       8348
L2 Hit-Rate    =       0.8155247
L2 Miss-Rate   =       0.18447532
L2 AvgNumEventsInMSHR   =       1.8957
L2 AvgNumEventsInMSHREntry      =       1.0000


L1 Hits        =       25466
L1 Misses      =       1433
L1 Accesses    =       26899
L1 Hit-Rate    =       0.9467266
L1 Miss-Rate   =       0.053273357
L1 AvgNumEventsInMSHR   =       1.7761
L1 AvgNumEventsInMSHREntry      =       2.2159


I1 Hits        =       74340
I1 Misses      =       4979
I1 Accesses    =       79319
I1 Hit-Rate    =       0.93722814
I1 Miss-Rate   =       0.06277184
I1 AvgNumEventsInMSHR   =       5.7481
I1 AvgNumEventsInMSHREntry      =       5.2051
```

<u>Observations:</u>
We can see that there is an increase in iTLB, dTLB misses because whenever we encounter Syscall instruction we flush iTLB, dTLB. So, iTLB and dTLB misses increased. As a result, IPC decreased after adding Syscall Instruction. There is no significant change in Cache stats because cache is not modified because of Syscall instruction.

Part - 2: Clflush (Flushing Cache Line)

Step - 1: Identifying Instruction as clflush

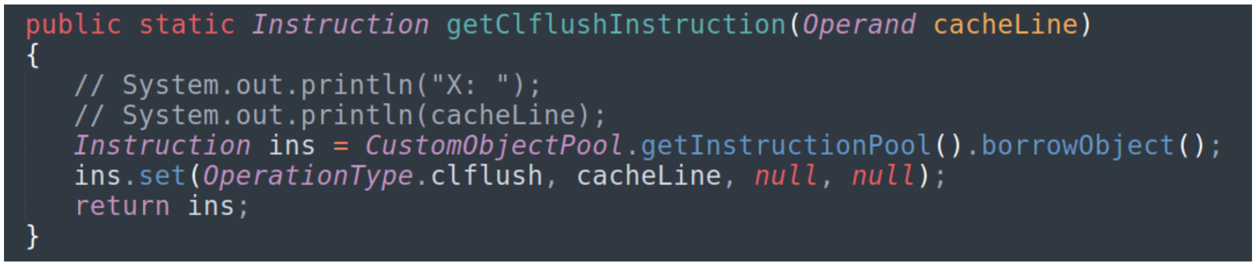
generic/OperationType.java



```java
package generic;

//TODO needs to renamed
public enum OperationType
{
    inValid,
    integerALU,
    integerMul,
    integerDiv,
    floatALU,
    floatMul,
    floatDiv,
    load,
    store,
    jump,
    branch,
    mov,
    xchg,
    acceleratedOp,
    nop,
    syscall,
    clflush,
    //TODO Software interrupt can also be modelled as a far jump
    interrupt,
    no_of_types,
    //sync type for causility check
    sync
}
```

Here, we have added clflush as OperationType

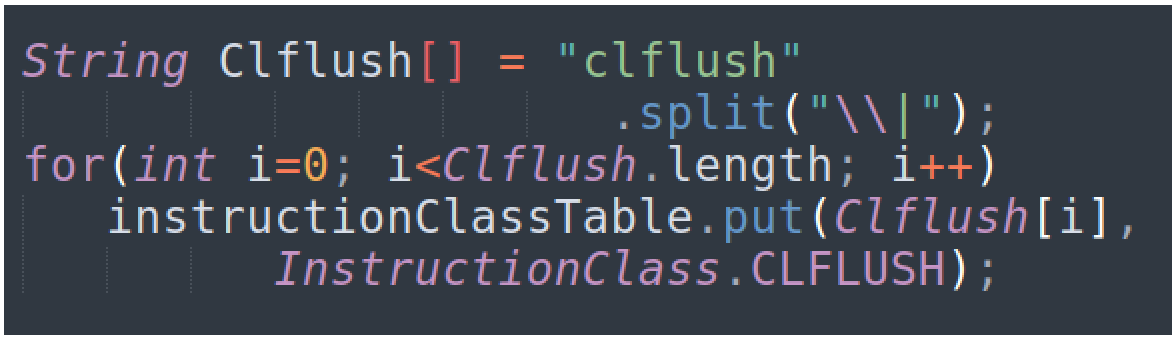
generic/Instruction.java



```java
public static Instruction getClflushInstruction(Operand cacheLine)
{
    // System.out.println("X: ");
    // System.out.println(cacheLine);
    Instruction ins = CustomObjectPool.getInstructionPool().borrowObject();
    ins.set(OperationType.clflush, cacheLine, null, null);
    return ins;
}
```

We have defined getClflushInstruction() method, which takes one operand (cacheLine address) as input. Other two operands are set to null.

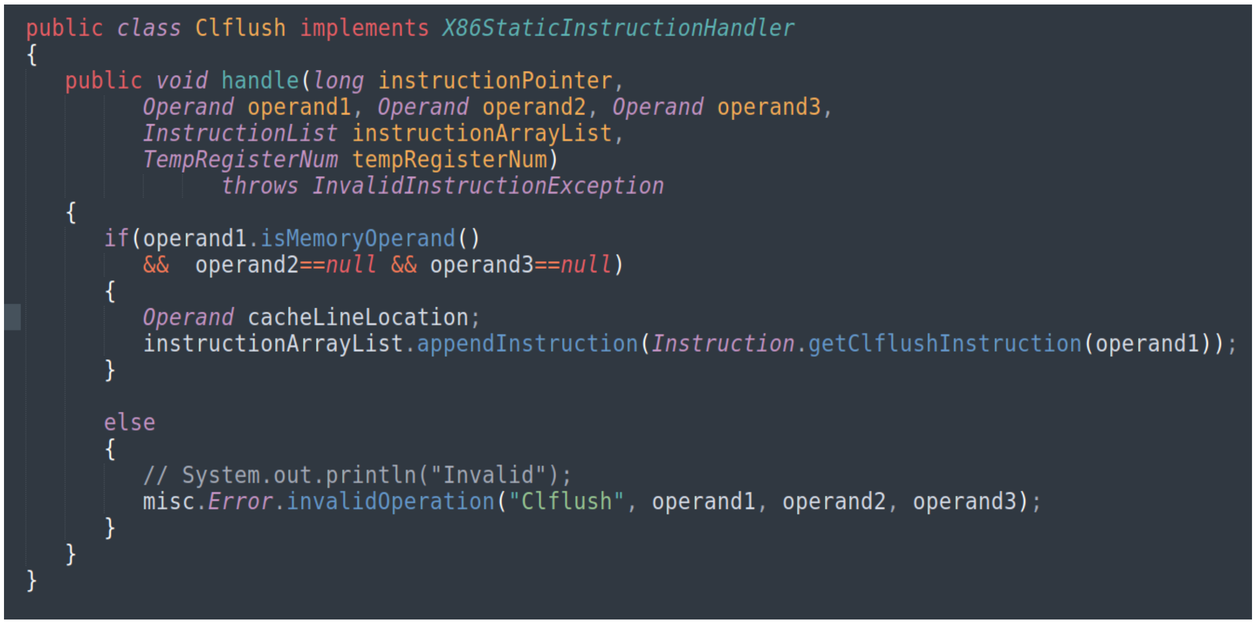
emulatorinterface/translator/x86/instruction/InstructionClassTable.java



```java
String Clflush[] = "clflush"
                        .split("\\|");
for(int i=0; i<Clflush.length; i++)
    instructionClassTable.put(Clflush[i],
          InstructionClass.CLFLUSH);
```

Here, instruction will be parsed and it will be identified as clflush instruction. CLFLUSH is added as an instruction in InstructionClass.java.

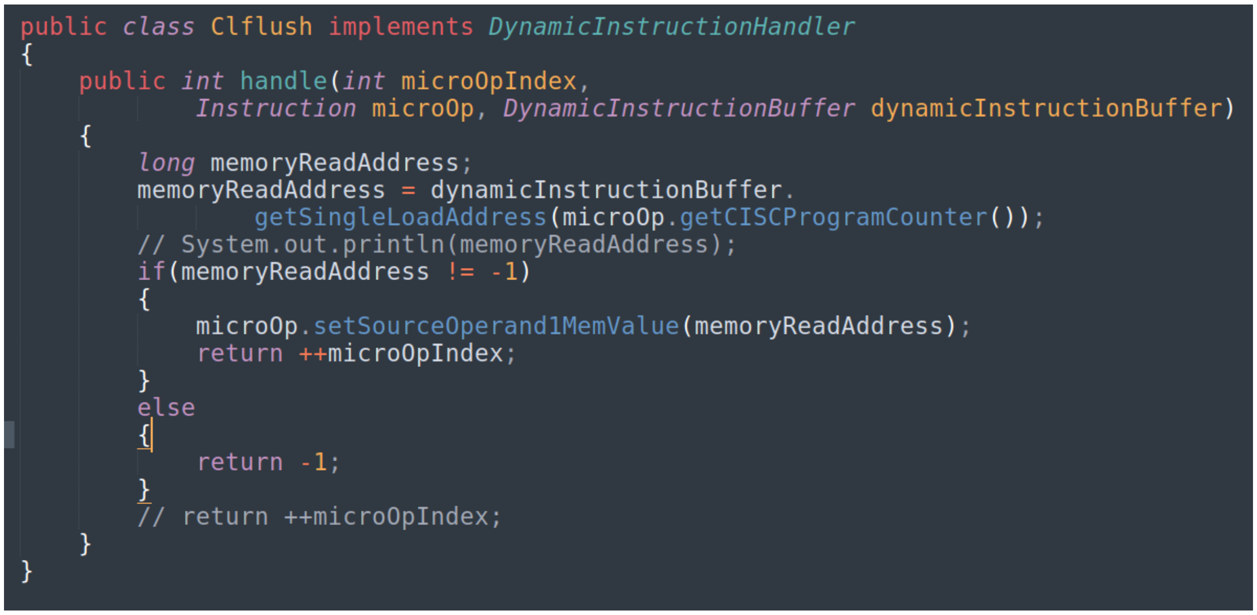
Step - 2: Handling Clflush Instruction
emulatorinterface/translator/x86/instruction/Clflush.java



```java
public class Clflush implements X86StaticInstructionHandler
{
    public void handle(long instructionPointer,
            Operand operand1, Operand operand2, Operand operand3,
            InstructionList instructionArrayList,
            TempRegisterNum tempRegisterNum)
                throws InvalidInstructionException
    {
        if(operand1.isMemoryOperand()
            &&  operand2==null && operand3==null)
        {
            Operand cacheLineLocation;
            instructionArrayList.appendInstruction(Instruction.getClflushInstruction(operand1));
        }

        else
        {
            // System.out.println("Invalid");
            misc.Error.invalidOperation("Clflush", operand1, operand2, operand3);
        }
    }
}
```

We have created a new file Clflush.java in the same folder. The first operand should be a memory operand (it will have the address of the cache line to be flushed) and other operands should be null. If not, it will raise an error. It will add Clflush instruction to InstructionArrayList.

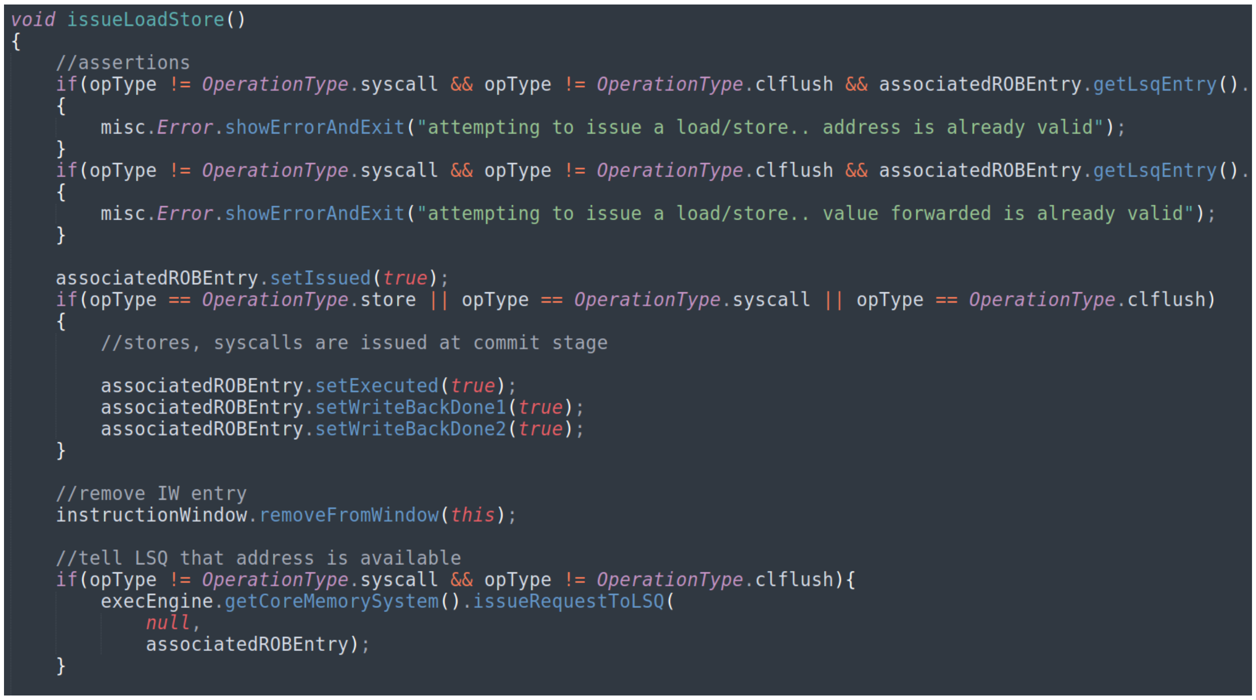
Then, we have changed translator/visaHandler/VisaHandlerSelector.java. In this file, we have created DynamicInstructionHandler for clflush and initialized it to an instance of Clflush(). For this, we have created a file named Clflush.java in the same folder.


```java
public class Clflush implements DynamicInstructionHandler
{
    public int handle(int microOpIndex,
            Instruction microOp, DynamicInstructionBuffer dynamicInstructionBuffer)
    {
        long memoryReadAddress;
        memoryReadAddress = dynamicInstructionBuffer.
                getSingleLoadAddress(microOp.getCISCProgramCounter());
        // System.out.println(memoryReadAddress);
        if(memoryReadAddress != -1)
        {
            microOp.setSourceOperand1MemValue(memoryReadAddress);
            return ++microOpIndex;
        }
        else
        {
            return -1;
        }
        // return ++microOpIndex;
    }
}
```

Clflush should get the memory address of the operand. So, we use getSingleLoadAddress() method of DynamicInstructionBuffer. It takes CICSProgramCounter() as input and returns a memory address. This memory address is stored in sourceOperand Memory Value.

Step - 3: Executing clflush Instruction
Now, we have to change the pipeline to execute the Clflush instruction. Similar to Syscall instruction, this instruction should be executed at the commit stage because of Precise Exceptions.

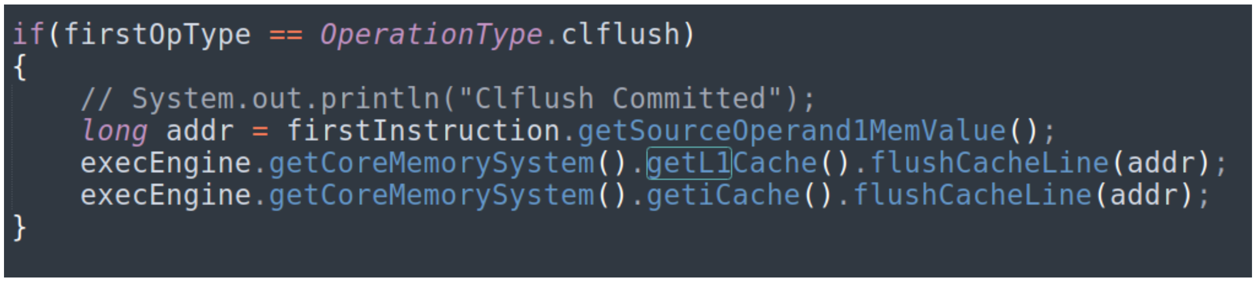
pipeline/outOfOrder/IWEntry.java



```java
void issueLoadStore()
{
    //assertions
    if(opType != OperationType.syscall && opType != OperationType.clflush && associatedROBEntry.getLsqEntry().
    {
        misc.Error.showErrorAndExit("attempting to issue a load/store.. address is already valid");
    }
    if(opType != OperationType.syscall && opType != OperationType.clflush && associatedROBEntry.getLsqEntry().
    {
        misc.Error.showErrorAndExit("attempting to issue a load/store.. value forwarded is already valid");
    }

    associatedROBEntry.setIssued(true);
    if(opType == OperationType.store || opType == OperationType.syscall || opType == OperationType.clflush)
    {
        //stores, syscalls are issued at commit stage

        associatedROBEntry.setExecuted(true);
        associatedROBEntry.setWriteBackDone1(true);
        associatedROBEntry.setWriteBackDone2(true);
    }

    //remove IW entry
    instructionWindow.removeFromWindow(this);

    //tell LSQ that address is available
    if(opType != OperationType.syscall && opType != OperationType.clflush){
        execEngine.getCoreMemorySystem().issueRequestToLSQ(
            null,
            associatedROBEntry);
    }
```

The clflush instruction is issued similarly to syscall instruction. So, the corresponding ROB entry is marked as executed. But it won't be added to LSQ.

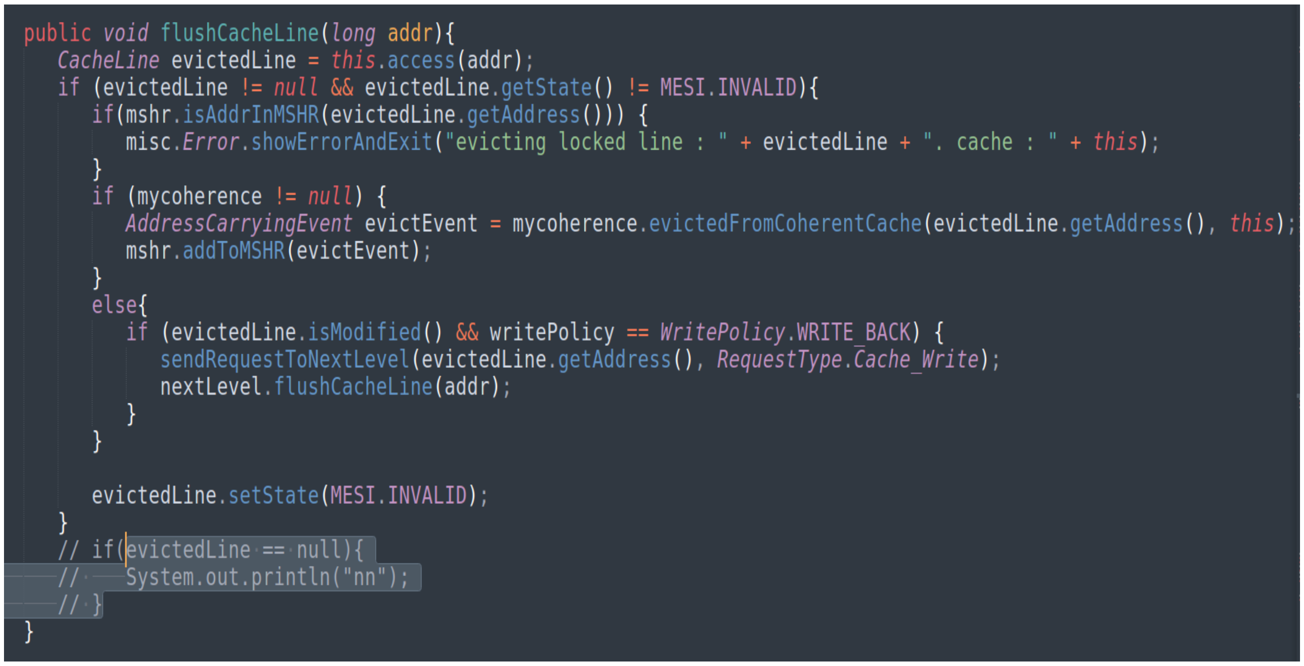
pipeline/outOfOrder/ReorderBufferjava



```java
if(firstOpType == OperationType.clflush)
{
    // System.out.println("Clflush Committed");
    long addr = firstInstruction.getSourceOperand1MemValue();
    execEngine.getCoreMemorySystem().getL1Cache().flushCacheLine(addr);
    execEngine.getCoreMemorySystem().getiCache().flushCacheLine(addr);
}
```

Before performing commits, we added the above snippet. It will read the memory address from the source operand (we have stored this earlier). Now, it will call flushCacheLine() function which takes addr as argument. We are flushing both data cache and instruction cache.

We have added a function in cache.java to flush the cache line.



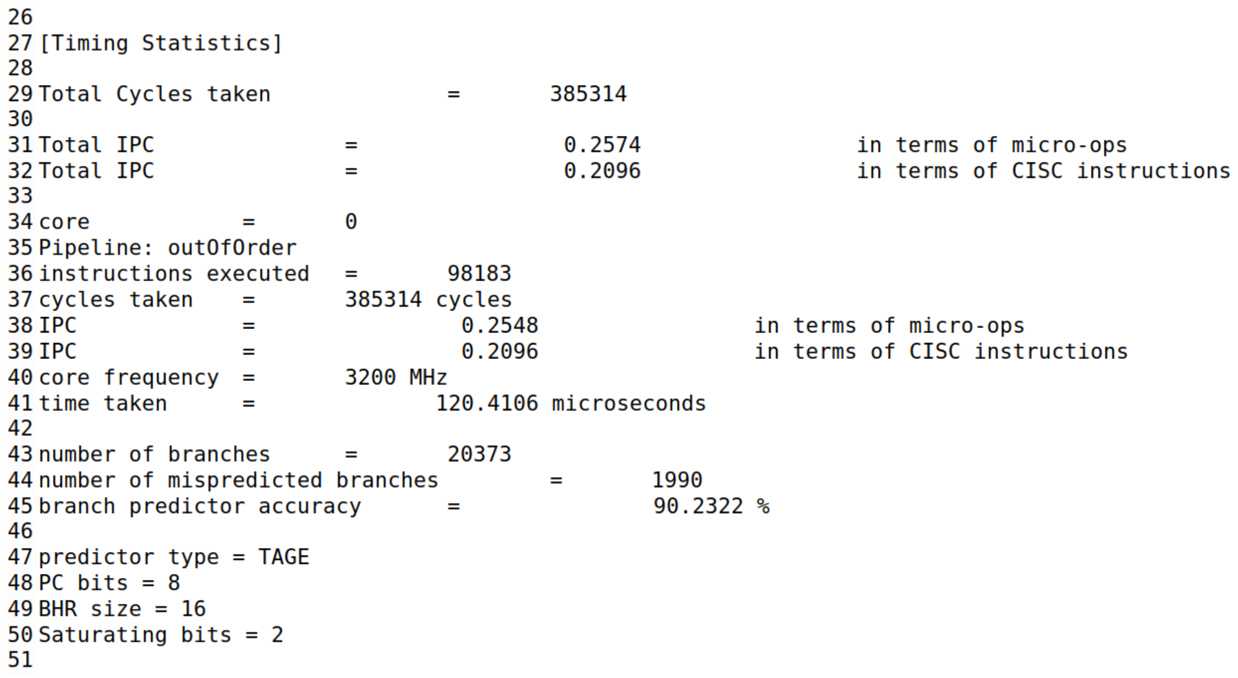
```java
public void flushCacheLine(long addr){
   CacheLine evictedLine = this.access(addr);
   if (evictedLine != null && evictedLine.getState() != MESI.INVALID){
      if(mshr.isAddrInMSHR(evictedLine.getAddress())) {
         misc.Error.showErrorAndExit("evicting locked line : " + evictedLine + ". cache : " + this);
      }
      if (mycoherence != null) {
         AddressCarryingEvent evictEvent = mycoherence.evictedFromCoherentCache(evictedLine.getAddress(), this);
         mshr.addToMSHR(evictEvent);
      }
      else{
         if (evictedLine.isModified() && writePolicy == WritePolicy.WRITE_BACK) {
            sendRequestToNextLevel(evictedLine.getAddress(), RequestType.Cache_Write);
            nextLevel.flushCacheLine(addr);
         }
      }

      evictedLine.setState(MESI.INVALID);
   }
   // if(evictedLine == null){
   //    System.out.println("nn");
   // }
}
```

This function takes the memory address as an argument and it should evict the cache line containing the address. The access function defined in Cache.java takes an address and returns the cache line containing it. If the cache line is NULL or if it is INVALID, there is no need to do anything. If not, we check if the cache line is locked or not. If it is locked, it raises an error. If it is a coherent cache, we create an event to evict the cache line and add this event to the queue of events (similar to what is done in handleEvictedLine() function in Cache.java). If the evicted Line is modified and the write policy is Write Back, then it has to be written to the memory. Instead, we write it to the cache line in the next cache level. Then ,we flush the cache line in the next cache level. So, eventually the modified value will be written to the memory. Finally, the cache line is marked as INVALID.

Output:

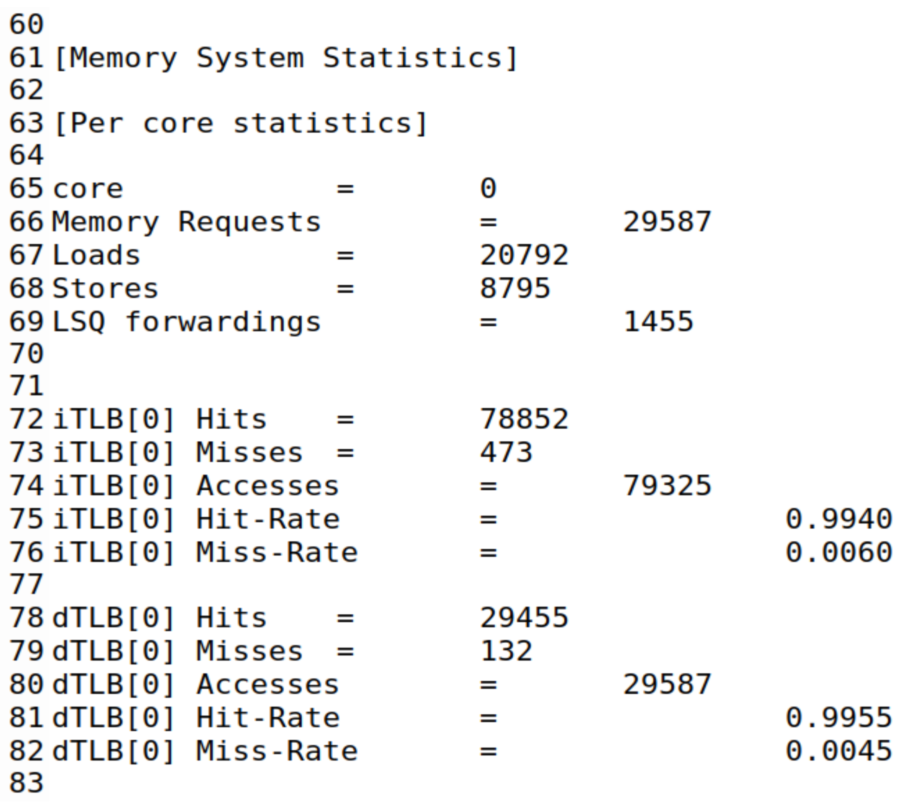
After Adding Only Clflush Instruction:

IPC Stats:



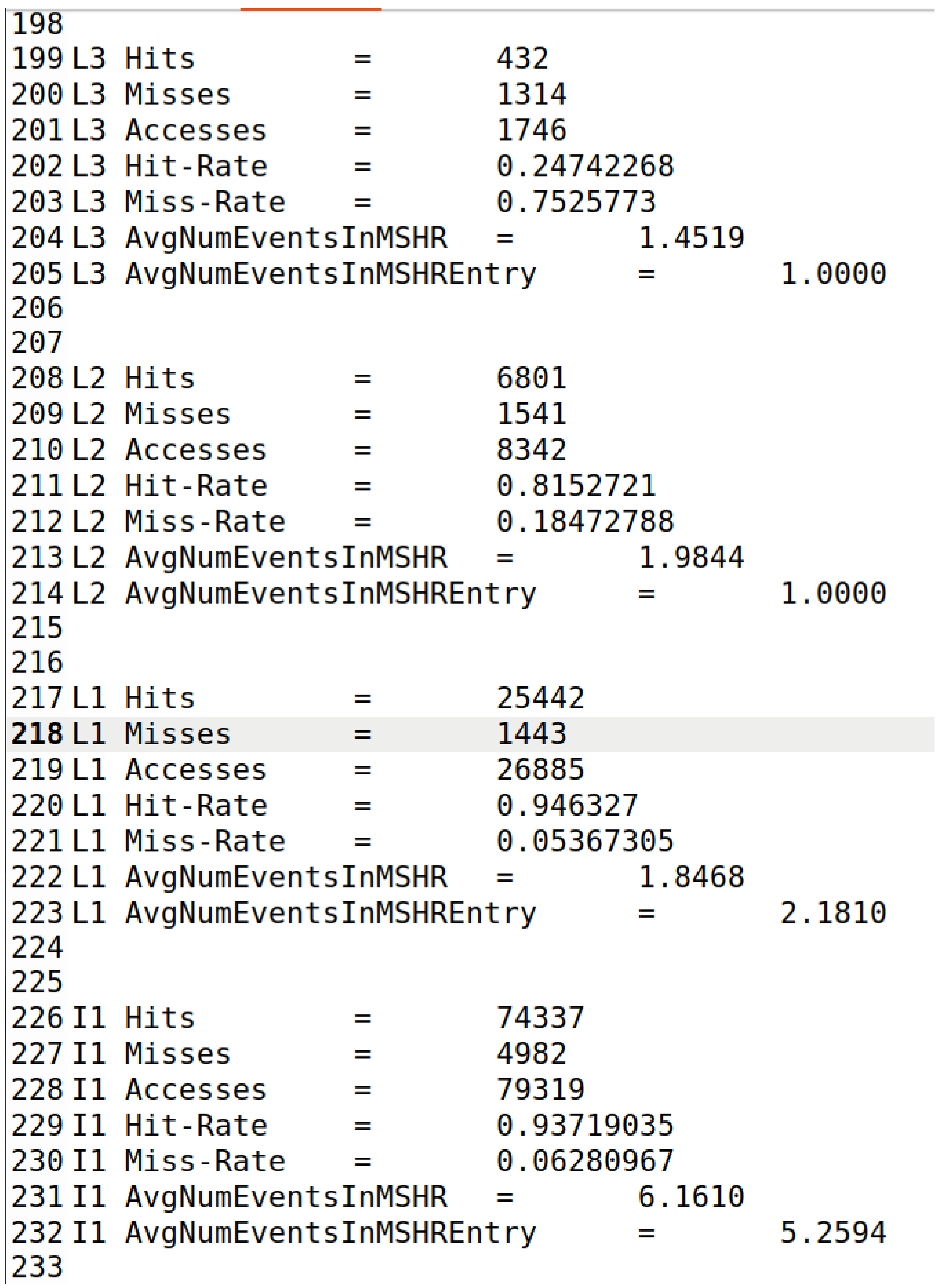
```

[Timing Statistics]

Total Cycles taken              =       385314

Total IPC               =               0.2574                  in terms of micro-ops
Total IPC               =               0.2096                  in terms of CISC instructions

core            =       0
Pipeline: outOfOrder
instructions executed   =       98183
cycles taken    =       385314 cycles
IPC             =               0.2548                  in terms of micro-ops
IPC             =               0.2096                  in terms of CISC instructions
core frequency  =       3200 MHz
time taken      =               120.4106 microseconds

number of branches      =       20373
number of mispredicted branches         =       1990
branch predictor accuracy       =               90.2322 %

predictor type = TAGE
PC bits = 8
BHR size = 16
Saturating bits = 2

```

TLB Stats:



```

[Memory System Statistics]

[Per core statistics]

core            =       0
Memory Requests         =       29587
Loads           =       20792
Stores          =       8795
LSQ forwardings         =       1455


iTLB[0] Hits    =       78852
iTLB[0] Misses  =       473
iTLB[0] Accesses        =       79325
iTLB[0] Hit-Rate        =               0.9940
iTLB[0] Miss-Rate       =               0.0060

dTLB[0] Hits    =       29455
dTLB[0] Misses  =       132
dTLB[0] Accesses        =       29587
dTLB[0] Hit-Rate        =               0.9955
dTLB[0] Miss-Rate       =               0.0045

```

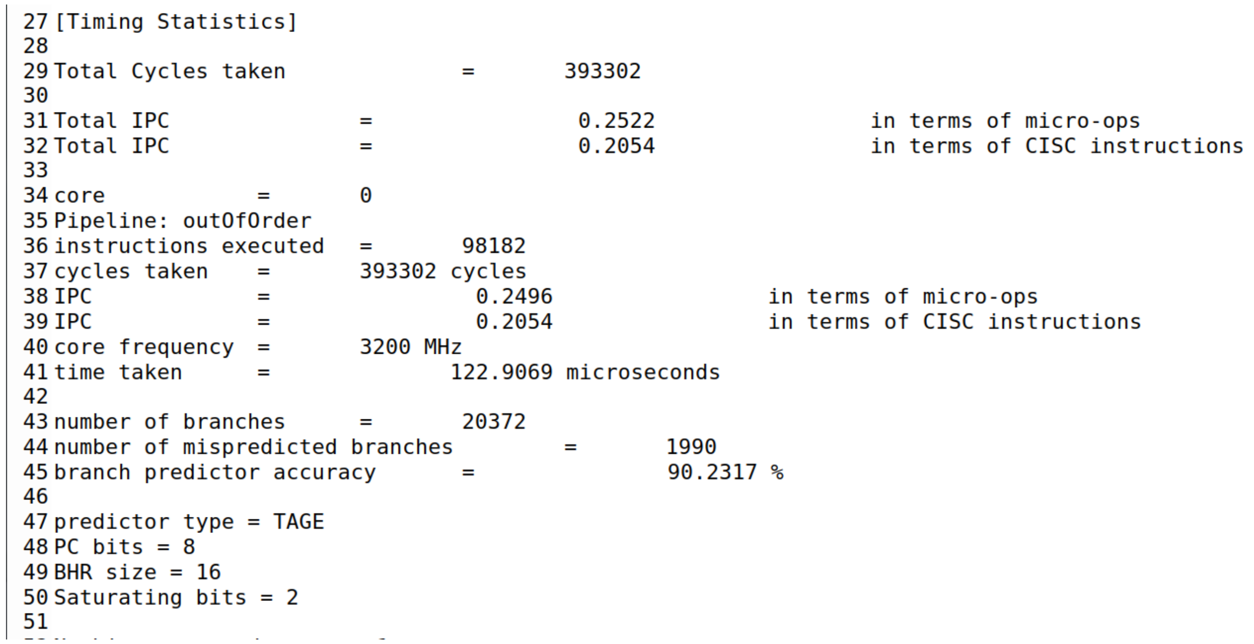
Cache Stats:



```

L3 Hits         =       432
L3 Misses       =       1314
L3 Accesses     =       1746
L3 Hit-Rate     =       0.24742268
L3 Miss-Rate    =       0.7525773
L3 AvgNumEventsInMSHR   =       1.4519
L3 AvgNumEventsInMSHREntry      =       1.0000


L2 Hits         =       6801
L2 Misses       =       1541
L2 Accesses     =       8342
L2 Hit-Rate     =       0.8152721
L2 Miss-Rate    =       0.18472788
L2 AvgNumEventsInMSHR   =       1.9844
L2 AvgNumEventsInMSHREntry      =       1.0000


L1 Hits         =       25442
L1 Misses       =       1443
L1 Accesses     =       26885
L1 Hit-Rate     =       0.946327
L1 Miss-Rate    =       0.05367305
L1 AvgNumEventsInMSHR   =       1.8468
L1 AvgNumEventsInMSHREntry      =       2.1810


I1 Hits         =       74337
I1 Misses       =       4982
I1 Accesses     =       79319
I1 Hit-Rate     =       0.93719035
I1 Miss-Rate    =       0.06280967
I1 AvgNumEventsInMSHR   =       6.1610
I1 AvgNumEventsInMSHREntry      =       5.2594

```

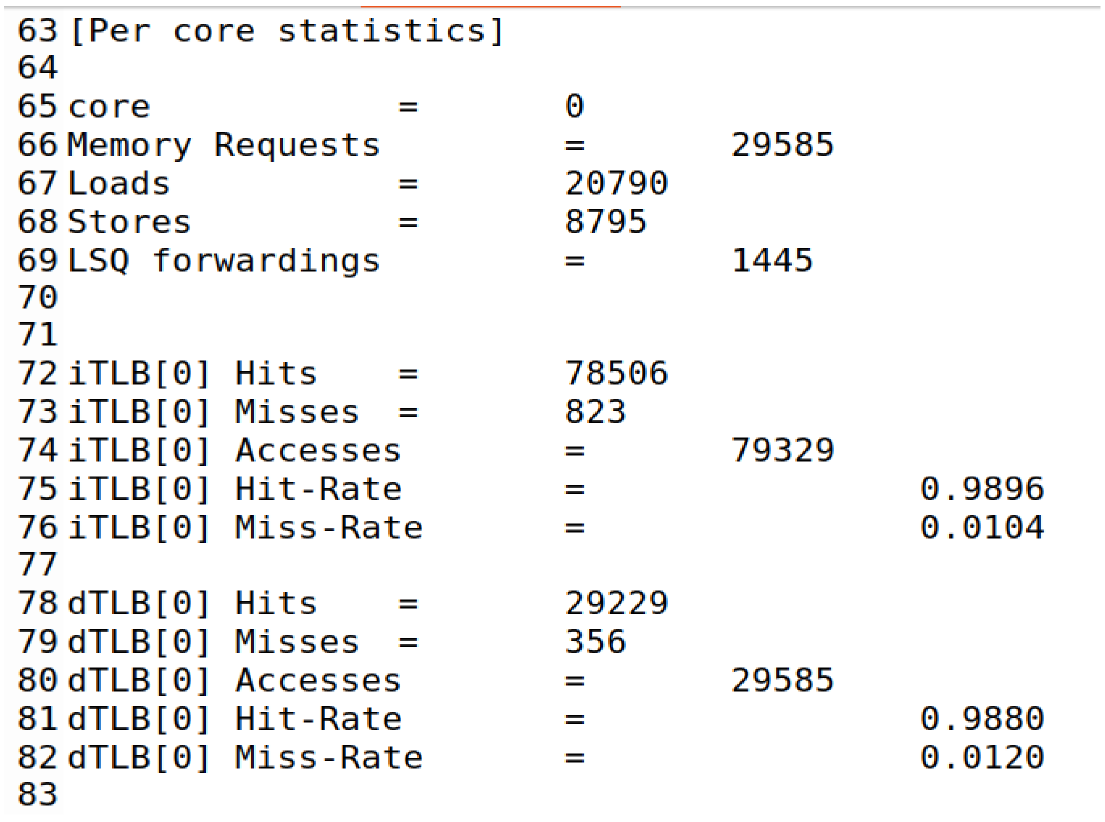
After Adding Both Syscall and Clflush Instructions

IPC Stats:



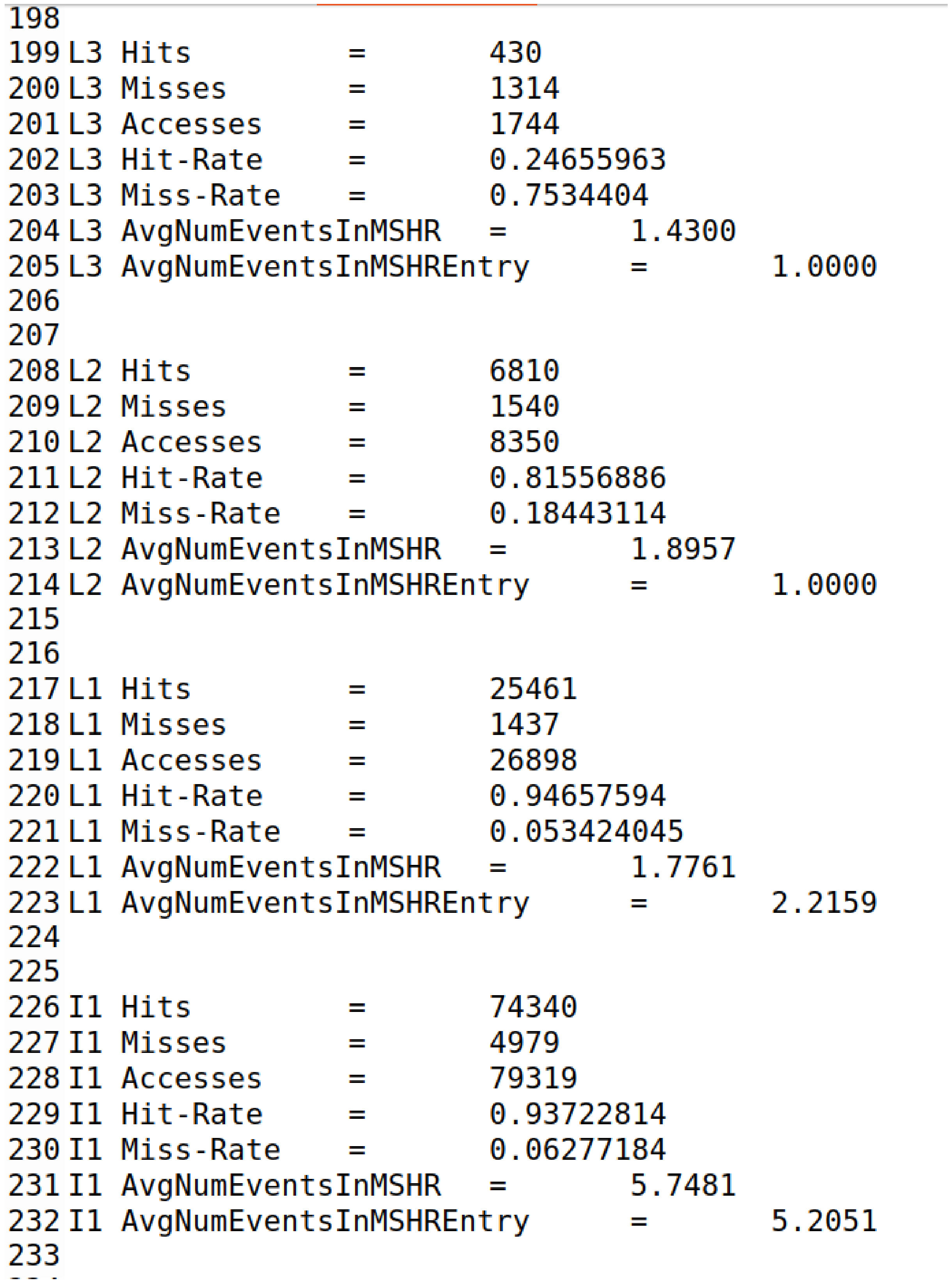
```
[Timing Statistics]

Total Cycles taken              =       393302

Total IPC               =               0.2522                  in terms of micro-ops
Total IPC               =               0.2054                  in terms of CISC instructions

core            =       0
Pipeline: outOfOrder
instructions executed   =       98182
cycles taken    =       393302 cycles
IPC             =               0.2496                  in terms of micro-ops
IPC             =               0.2054                  in terms of CISC instructions
core frequency  =       3200 MHz
time taken      =               122.9069 microseconds

number of branches      =       20372
number of mispredicted branches         =       1990
branch predictor accuracy       =               90.2317 %

predictor type = TAGE
PC bits = 8
BHR size = 16
Saturating bits = 2

```

TLB Stats:



```
[Per core statistics]

core            =       0
Memory Requests         =       29585
Loads           =       20790
Stores          =       8795
LSQ forwardings         =       1445


iTLB[0] Hits    =       78506
iTLB[0] Misses  =       823
iTLB[0] Accesses        =       79329
iTLB[0] Hit-Rate        =               0.9896
iTLB[0] Miss-Rate       =               0.0104

dTLB[0] Hits    =       29229
dTLB[0] Misses  =       356
dTLB[0] Accesses        =       29585
dTLB[0] Hit-Rate        =               0.9880
dTLB[0] Miss-Rate       =               0.0120

```

Cache Stats:



```

L3 Hits          =       430
L3 Misses        =       1314
L3 Accesses      =       1744
L3 Hit-Rate      =       0.24655963
L3 Miss-Rate     =       0.7534404
L3 AvgNumEventsInMSHR    =       1.4300
L3 AvgNumEventsInMSHREntry      =       1.0000


L2 Hits          =       6810
L2 Misses        =       1540
L2 Accesses      =       8350
L2 Hit-Rate      =       0.81556886
L2 Miss-Rate     =       0.18443114
L2 AvgNumEventsInMSHR    =       1.8957
L2 AvgNumEventsInMSHREntry      =       1.0000


L1 Hits          =       25461
L1 Misses        =       1437
L1 Accesses      =       26898
L1 Hit-Rate      =       0.94657594
L1 Miss-Rate     =       0.053424045
L1 AvgNumEventsInMSHR    =       1.7761
L1 AvgNumEventsInMSHREntry      =       2.2159


I1 Hits          =       74340
I1 Misses        =       4979
I1 Accesses      =       79319
I1 Hit-Rate      =       0.93722814
I1 Miss-Rate     =       0.06277184
I1 AvgNumEventsInMSHR    =       5.7481
I1 AvgNumEventsInMSHREntry      =       5.2051

```

Observations:
Comparison between statistics before adding and statistics after adding only clflush instruction:
There is only 1 clflush instruction in the given trace.
There is no change in TLB stats because clflush instruction is not modifying TLBs. The L1 cache misses increased by 4 because we flushed a cache line given in clflush instruction. So, next time the address is accessed it results in a cache miss. The address is not present in L2, L3 caches. So, no cache line is evicted from L2, L3 caches. So, L2, L3 cache stats remain the same. Overall IPC is the same after adding clflush.

These observations remain same for comparison between statistics after adding syscall and statistics after adding both instructions