COL718 As2

2022MCS2057,2019CS50617

October 2022

1 New x86 Instruction

1.1 OperationType:

Depending on the kind of instruction we might need to make a new operation type for that instruction. This will come in handy later on to identify our special instruction when we push into the pipeline.

1.2 x86 Translation:

To add support for a new x86 instruction in Tejas we must first create a string matching to check for the instruction in a trace file. We add a new entry in the instructionClassTable hashtable for this new instruction. Next we create a new staticInstructionHandler for our instruction. We also need to create a function in the Instruction.java file that returns a new instruction of specified operation type when it is encountered in the trace file.

```java
//CacheFlush.java
public void handle(long instructionPointer,
        Operand operand1, Operand operand2, Operand operand3,
        InstructionList instructionArrayList,
        TempRegisterNum tempRegisterNum)
             throws InvalidInstructionException
                 {
                     if(operand1.isMemoryOperand() && operand2==null &&
                         operand3==null)
                     {

                         Instruction ins =
                             Instruction.getCacheFlushInstruction(operand1);
                         instructionArrayList.appendInstruction(ins);


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

```
                    }
```

```
//InstructionClassTable.java
String cfluschImmediate="clflush";
      instructionClassTable.put(cfluschImmediate,
           InstructionClass.CACHE_FLUSH);
instructionClassHandlerTable.put(
         InstructionClass.CACHE_FLUSH,
         new CacheFlush());
```

```
//Instruction.java
public static Instruction getCacheFlushInstruction(Operand
     memoryLocation)
   {
      Instruction ins =
           CustomObjectPool.getInstructionPool().borrowObject();
      ins.set(OperationType.clflush, memoryLocation, null, null);
      return ins;
   }
```

1.3 VISA Translation:

All instructions in the trace file are first converted to VISA instructions for simplicity and uniformity. So we must add support for the new instruction here as well. We follow the same procedure as before only now new don't create a string matching. We create a new dynamic instruction handler for the new instruction and point our x86 instruction to this handler. For address carrying instruction we also need to pass the resolved address in this instruction. To do this we look for the address in the trace file, indexed by the cisc instruction program counter, and place it in the Operand1 memory value of this new instruction so that we can access it inside the pipeline.

```
//CacheFlush.java in VISA
public int handle(int microOpIndex,
         Instruction microOp, DynamicInstructionBuffer
              dynamicInstructionBuffer)
   {
      long memoryReadAddress;
      memoryReadAddress = dynamicInstructionBuffer.
            getSingleLoadAddress(microOp.getCISCProgramCounter());

      if(memoryReadAddress != -1)
      {
         microOp.setSourceOperand1MemValue(memoryReadAddress);
         return ++microOpIndex;
      }
      else
      {
         return -1;
      }
   }
```

1.4 Pipeline:

Now our instruction is ready to be supplied to the pipeline. Before that we must assign a Functional unit which this instruction would require to process it. In our case we pick the memory FU for our task. Now we can access our instruction from within the pipeline and process it accordingly.

1.5 Notes:

We had to make a few extra changes to the config file because in the file that shipped with Tejas, the L1 cache was configured to be bigger than the L2 cache. To us, it seemed like a typo so we changed it to the size it is meant to be according to the configuration.

2 Clflush:

2.1 Changes in Cache:

We must create a function that takes in the address of a cache line and then proceeds to set that line invalid for every layer in the cache hierarchy and replaces this line, if modified back to main memory. We added the following function to the Cache class:

```
// Cache.java
public void clearLine(Long addr)
   {

      CacheLine ll = access(addr);
      if(ll!=null && (ll.getState()==MESI.MODIFIED ||
          ll.getState()==MESI.SHARED))
      {
         if(this.isLastLevel)
         {
            this.handleEvictedLine(ll);
         }
         else
         {
            this.nextLevel.clearLine(addr);
         }
         ll.setState(MESI.INVALID);
      }
      else
      {
         if(!this.isLastLevel)
            this.nextLevel.clearLine(addr);
      }
   }
```

2.2 Call Function From Pipeline:

We look for instruction of type clfush during the writeback stage. Whenever we encounter one we extract the address from using getSourceOperand1MemValue function and call it on the L1 cache.

```
//WriteBackLogic.java
else
    if(buffer[i].getInstruction().getOperationType()==OperationType.clflush)
          {
             Instruction ins1 = buffer[i].getInstruction();
             Long addr = ins1.getSourceOperand1MemValue();
             execEngine.getCoreMemorySystem().getL1Cache().clearLine(addr);

          }
```

3 syscall

3.1 Changes To TLB:

We create a function in the TLB class that clears all the entries. Since TLB is implemented as a hashtable doing this is pretty straghtforward:

```
//TLB.java
public void flushTlb()
   {
      TLBuffer.clear();
   }
```

3.2 Call from Pipeline:

Similarly calling this function from the write back stage:

```
//WriteBackLogic.java
else
    if(buffer[i].getInstruction().getOperationType()==OperationType.syscall)
          {
```

```
        execEngine.getCoreMemorySystem().getiTLB().flushTlb();
        execEngine.getCoreMemorySystem().getdTLB().flushTlb();
      }
```

4 Results

4.1 Without Modification:

```
core    =  0
Pipeline: outOfOrder
instructions executed = 98183
cycles taken  =  384815 cycles
IPC     =         0.2551     in terms of micro-ops
IPC     =         0.2099     in terms of CISC instructions
core frequency = 3200 MHz
time taken =       120.2547 microseconds

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

L1[0] Hits =  25444
L1[0] Misses  =  1439
L1[0] Accesses = 26883
L1[0] Hit-Rate = 0.94647175
L1[0] Miss-Rate =  0.053528253
L1[0] AvgNumEventsInMSHR = 1.8464
L1[0] AvgNumEventsInMSHREntry = 2.1682


L2[0] Hits =  7571
L2[0] Misses  =  1314
L2[0] Accesses = 8885
L2[0] Hit-Rate = 0.8521103
L2[0] Miss-Rate =  0.1478897
L2[0] AvgNumEventsInMSHR = 1.9401
L2[0] AvgNumEventsInMSHREntry = 1.0000
```

4.2 With Just syscall:

```
core    =  0
Pipeline: outOfOrder
instructions executed = 98192
cycles taken  =  392741 cycles
IPC     =         0.2500     in terms of micro-ops
IPC     =         0.2057     in terms of CISC instructions
core frequency = 3200 MHz
time taken =       122.7316 microseconds

iTLB[0] Hits  =  78504
iTLB[0] Misses = 830
iTLB[0] Accesses = 79334
iTLB[0] Hit-Rate =        0.9895
iTLB[0] Miss-Rate =       0.0105

dTLB[0] Hits  =  29229
dTLB[0] Misses = 359
dTLB[0] Accesses = 29588
dTLB[0] Hit-Rate =        0.9879
dTLB[0] Miss-Rate =       0.0121

L1[0] Hits =  25466
L1[0] Misses  =  1432
L1[0] Accesses = 26898
L1[0] Hit-Rate = 0.94676185
L1[0] Miss-Rate =  0.053238157
L1[0] AvgNumEventsInMSHR = 1.7704
L1[0] AvgNumEventsInMSHREntry = 2.1893


L2[0] Hits =  7576
L2[0] Misses  =  1314
L2[0] Accesses = 8890
L2[0] Hit-Rate = 0.8521935
L2[0] Miss-Rate =  0.14780653
L2[0] AvgNumEventsInMSHR = 1.8342
L2[0] AvgNumEventsInMSHREntry = 1.0000
```

4.3 With Just clflush:

```
core    =  0
Pipeline: outOfOrder
instructions executed = 98183
cycles taken  =  384815 cycles
IPC     =         0.2551     in terms of micro-ops
IPC     =          0.2099     in terms of CISC instructions
core frequency = 3200 MHz
time taken =       120.2547 microseconds

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

L1[0] Hits =  25440
L1[0] Misses  =  1440
L1[0] Accesses = 26880
L1[0] Hit-Rate = 0.9464286
L1[0] Miss-Rate =  0.05357143
L1[0] AvgNumEventsInMSHR = 1.8490
L1[0] AvgNumEventsInMSHREntry = 2.1646


L2[0] Hits =  7565
L2[0] Misses  =  1315
L2[0] Accesses = 8880
L2[0] Hit-Rate = 0.8519144
L2[0] Miss-Rate =  0.14808558
L2[0] AvgNumEventsInMSHR = 1.9401
L2[0] AvgNumEventsInMSHREntry = 1.0000
```

4.4 With Both:

```
core    =  0
Pipeline: outOfOrder
instructions executed = 98192
cycles taken  =  392741 cycles
IPC     =          0.2500     in terms of micro-ops
IPC     =          0.2057     in terms of CISC instructions
core frequency = 3200 MHz
time taken =       122.7316 microseconds

iTLB[0] Hits  =  78504
iTLB[0] Misses = 830
iTLB[0] Accesses = 79334
iTLB[0] Hit-Rate =        0.9895
```

```
iTLB[0] Miss-Rate =        0.0105

dTLB[0] Hits  =  29228
dTLB[0] Misses = 359
dTLB[0] Accesses = 29587
dTLB[0] Hit-Rate =         0.9879
dTLB[0] Miss-Rate =        0.0121

L1[0] Hits =  25462
L1[0] Misses  =  1433
L1[0] Accesses = 26895
L1[0] Hit-Rate = 0.9467187
L1[0] Miss-Rate =  0.053281277
L1[0] AvgNumEventsInMSHR = 1.7704
L1[0] AvgNumEventsInMSHREntry = 2.1893


L2[0] Hits =  7570
L2[0] Misses  =  1315
L2[0] Accesses = 8885
L2[0] Hit-Rate = 0.85199773
L2[0] Miss-Rate =  0.14800225
L2[0] AvgNumEventsInMSHR = 1.8342
L2[0] AvgNumEventsInMSHREntry = 1.0000
```