ECE4504
Project Phase 3: Caching

Emil Bengtsson

November 2018

Implementation

The goal of this phase of the project was to implement and evaluate a cache in the baseline EduMIPS64 simulator.

I implemented the cache in its own class using HashMaps and arrays. I chose the size of the shared cache to be 512 bytes and the split caches to be 256 bytes each. I used a random replacement strategy.

I then changed the code of the baseline EduMIPS64 simulator to try to access data from the cache before the memory. If it fails it is counted as a cache miss and the data is loaded from memory into the cache.

Experimental Setup

For evaluation we were asked to find benchmarks on our own. I didn't find much on the internet so I wrote my own. I wrote three benchmarks that I think should show differences between different block sizes, set associativities, and whether the cache is split or shared.

The first benchmark of the three I wrote is called "lotsofcode" and it contains a lot of assembly instructions including quite a few branch instructions. The idea is that this might show some differences between different set associativities and also differences between a shared and split cache.

The second benchmark I wrote is called "smallforloop" and it is a small for loop. It should fit in the cache but the idea is that it should show conflict misses.

The third benchmark program I wrote is called "storeandload". It contains a lot of store and load instructions and should hopefully show some differences between different block sizes and also between split and shared cache.

I will evaluate the implementations by running the benchmark programs with all combinations of the three settings. I will then compare the statistics of the runs.

I will refer to the three benchmark programs I use by their names: lotsofcode, smallforloop, and storeandload.

Results

Cache miss rate of the benchmarks using a shared instruction and data cache:

settings		Cache Miss Rate		
Set Associativity	Block Size	lotsofcode	smallforloop	storeandload
1	16	0.18182	0.29630	0.11837
2	16	0.17398	0.16667	0.08333
4	16	0.15047	0.16667	0.08239
8	16	0.10972	0.16667	0.07765
16	16	0.12382	0.16667	0.07860
1	32	0.11442	0.31481	0.10417
2	32	0.10658	0.09259	0.04735
4	32	0.11599	0.09259	0.04261
8	32	0.12069	0.09259	0.03788
16	32	0.10972	0.09259	0.03883
1	64	0.10345	0.29630	0.10701
2	64	0.09718	0.05556	0.03030
4	64	0.10031	0.05556	0.02178
8	64	0.09248	0.05556	0.01894
16	64	0.09404	0.05556	0.01894
1	128	0.09718	0.38889	0.18939
2	128	0.09091	0.03704	0.01610
4	128	0.09404	0.03704	0.01326
8	128	0.09404	0.03704	0.01136
16	128	0.08464	0.03704	0.01420
1	256	0.07053	0.38889	0.33049
2	256	0.06897	0.03704	0.01326
4	256	0.06740	0.03704	0.00852
8	256	0.06583	0.03704	0.01042
16	256	0.06583	0.03704	0.01136

Cache miss rate of the benchmarks using a split instruction and data cache:

settings		Cache Miss Rate		
Set Associativity	Block Size	lotsofcode	smallforloop	storeandload
1	16	0.40439	0.16667	0.07576
2	16	0.39028	0.16667	0.07576
4	16	0.37147	0.16667	0.07576
8	16	0.36364	0.16667	0.07576
16	16	0.37618	0.16667	0.07576
1	32	0.26019	0.09259	0.03788
2	32	0.24451	0.09259	0.03788
4	32	0.24765	0.09259	0.03788
8	32	0.23041	0.09259	0.03788
16	32	0.23354	0.09259	0.03788
1	64	0.16144	0.05556	0.01894
2	64	0.15047	0.05556	0.01894
4	64	0.15047	0.05556	0.01894
8	64	0.14577	0.05556	0.01894
16	64	0.15204	0.05556	0.01894
1	128	0.10972	0.03704	0.01042
2	128	0.10815	0.03704	0.01042
4	128	0.10815	0.03704	0.01042
8	128	0.10972	0.03704	0.01042
16	128	0.10972	0.03704	0.01042
1	256	0.07053	0.03704	0.00568
2	256	0.07053	0.03704	0.00568
4	256	0.07053	0.03704	0.00568
8	256	0.07053	0.03704	0.00568
16	256	0.07053	0.03704	0.00568

Comment on the results

smallforloop:

The results of the smallforloop benchmark was pretty much what I expected they would be. On the shared cache, when the set associativity is set to one, you get a high cache miss rate. This was mainly because cache lines with instructions or data had to be placed in the same spot in the cache, and that made them replace each other a lot. When the set associativity was set to something greater than one, and when a split cache was used, the cache lines with instructions and the cache lines with data did not replace each other as much, and the cache miss rate was quite low.

lotsofcode:

The results of the lotsofcode benchmark were quite interesting I think. When using a shared cache, the cache miss rate got lowered every time the clock size was increased. It also got lowered a little bit when the set associativity was increased. This was even more apparent when using a split cache.

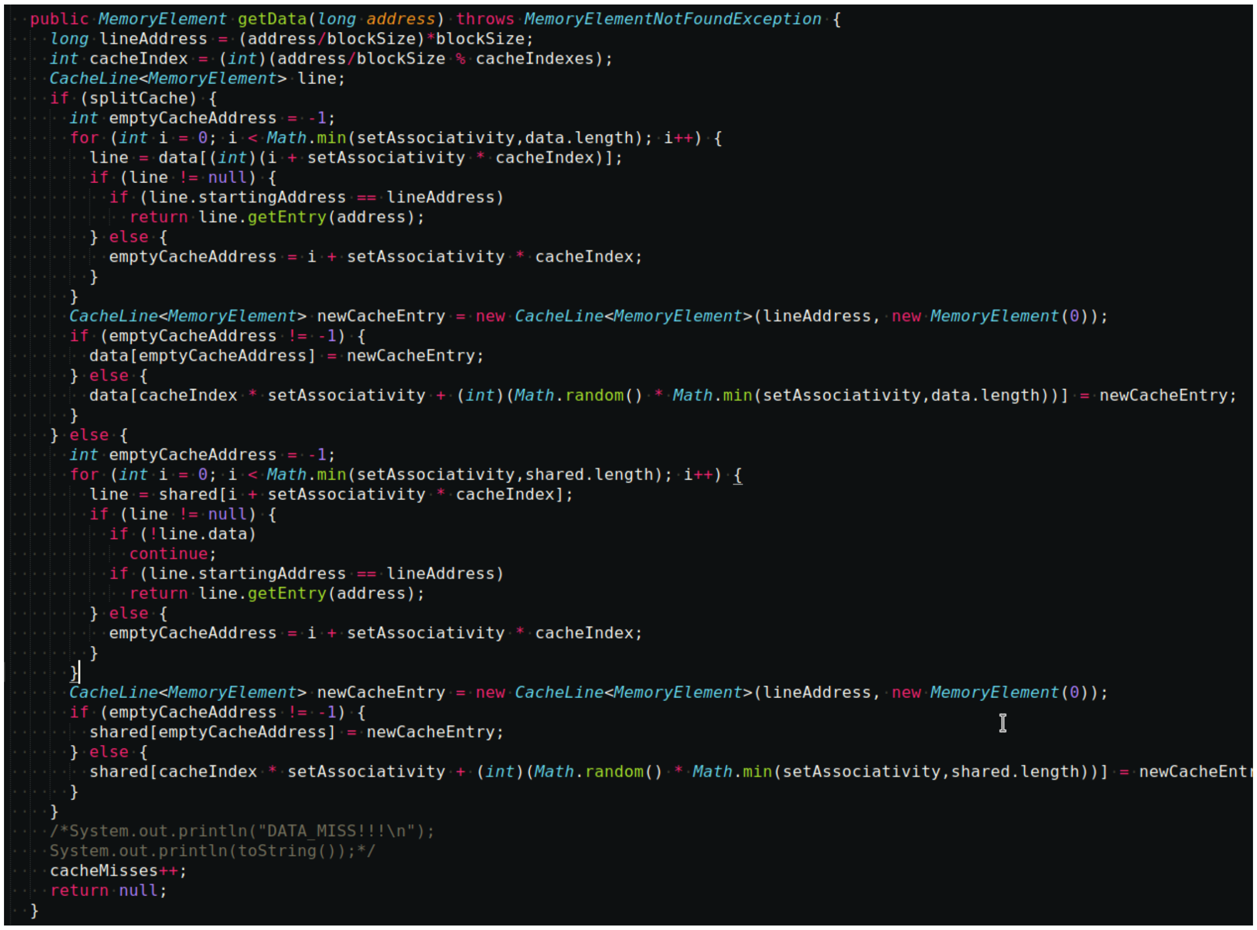
storeandload:

The storeandload benchmark got some interesting results too. When using a shared cache, the cache miss rate got lower as the set associativity increased. I think this was because the instruction cache lines and the data cache lines did not have to replace each other as much, which means there were less conflict misses. When using a split cache, the set associativity did not affect the miss rate, which is interesting. The block size lowered the miss rate of the benchmark significantly when the set associativity was more than one, but not always when it was one. I think the reason why the miss rate increased on the shared cache when the set associativity was one was that there were more conflict misses.

All in all the split cache did better when there were a lot of store and load operations and the shared cache did better when there were a lot of instructions and not so many store and load operations.

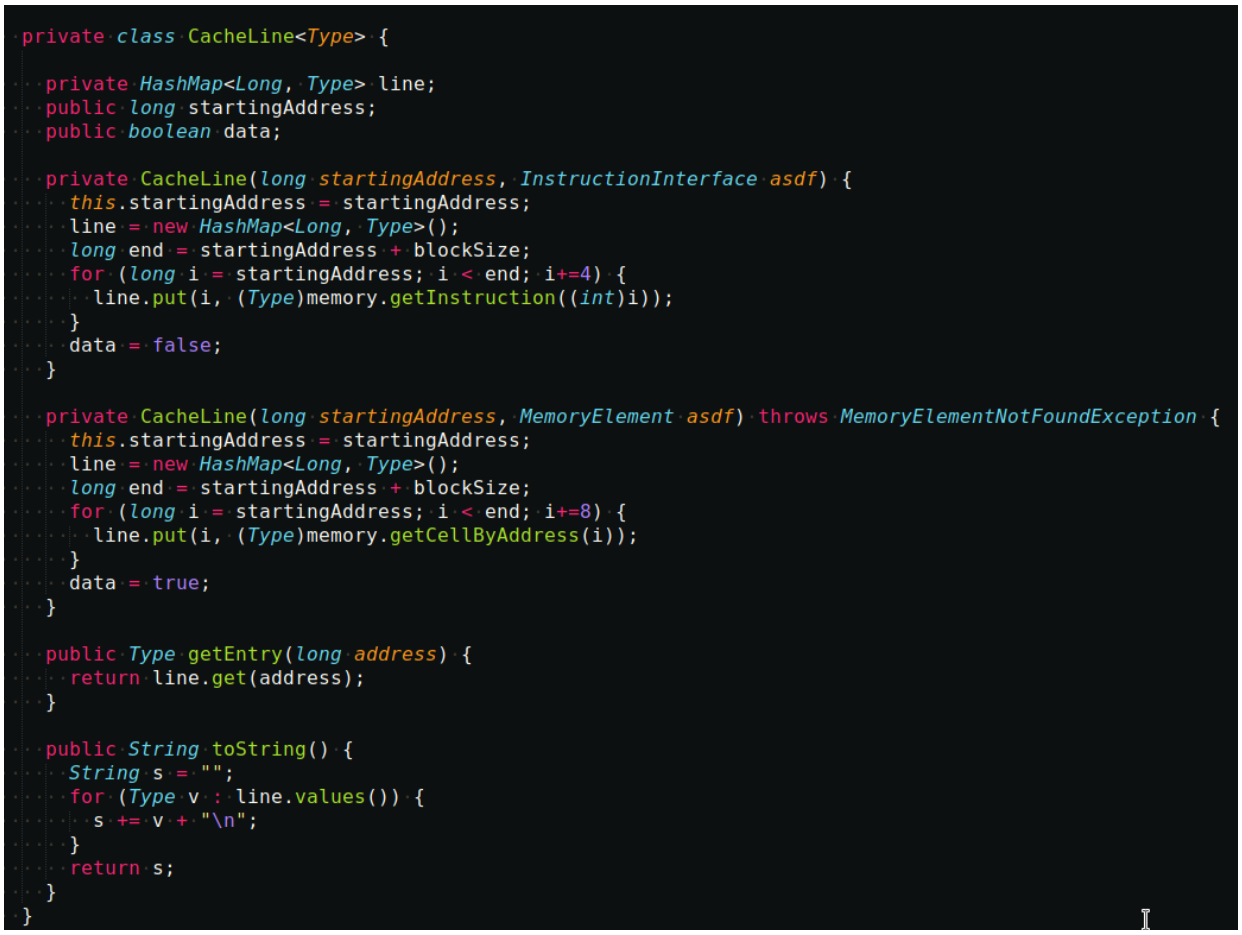
Screen shots

Here are some screen shots of important parts of the code and the program:



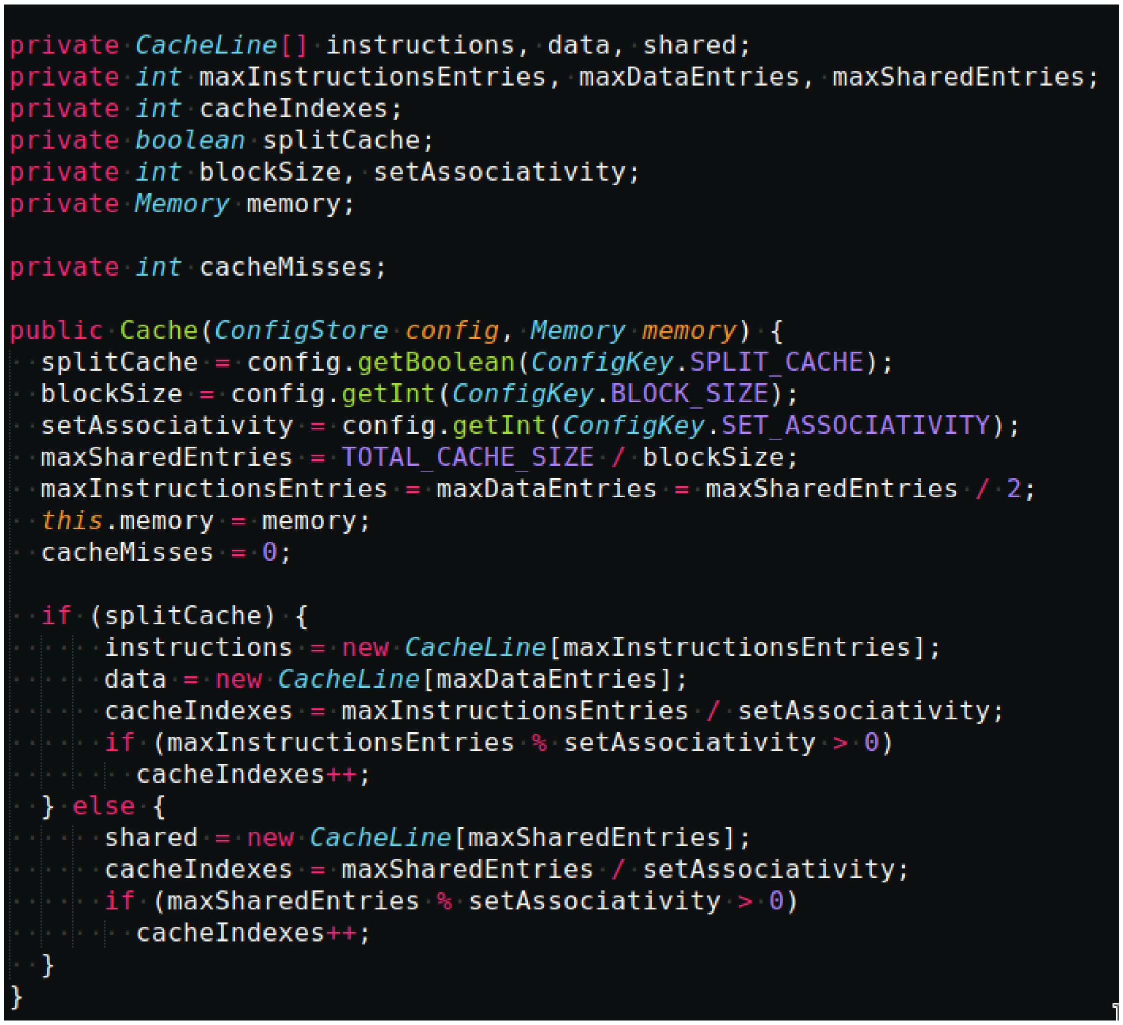
```
  public MemoryElement getData(long address) throws MemoryElementNotFoundException {
    long lineAddress = (address/blockSize)*blockSize;
    int cacheIndex = (int)(address/blockSize % cacheIndexes);
    CacheLine<MemoryElement> line;
    if (splitCache) {
      int emptyCacheAddress = -1;
      for (int i = 0; i < Math.min(setAssociativity,data.length); i++) {
        line = data[(int)(i + setAssociativity * cacheIndex)];
        if (line != null) {
          if (line.startingAddress == lineAddress)
            return line.getEntry(address);
        } else {
          emptyCacheAddress = i + setAssociativity * cacheIndex;
        }
      }
      CacheLine<MemoryElement> newCacheEntry = new CacheLine<MemoryElement>(lineAddress, new MemoryElement(0));
      if (emptyCacheAddress != -1) {
        data[emptyCacheAddress] = newCacheEntry;
      } else {
        data[cacheIndex * setAssociativity + (int)(Math.random() * Math.min(setAssociativity,data.length))] = newCacheEntry;
      }
    } else {
      int emptyCacheAddress = -1;
      for (int i = 0; i < Math.min(setAssociativity,shared.length); i++) {
        line = shared[i + setAssociativity * cacheIndex];
        if (line != null) {
          if (!line.data)
            continue;
          if (line.startingAddress == lineAddress)
            return line.getEntry(address);
        } else {
          emptyCacheAddress = i + setAssociativity * cacheIndex;
        }
      }
      CacheLine<MemoryElement> newCacheEntry = new CacheLine<MemoryElement>(lineAddress, new MemoryElement(0));
      if (emptyCacheAddress != -1) {
        shared[emptyCacheAddress] = newCacheEntry;
      } else {
        shared[cacheIndex * setAssociativity + (int)(Math.random() * Math.min(setAssociativity,shared.length))] = newCacheEnt
      }
    }
    /*System.out.println("DATA_MISS!!!\n");
    System.out.println(toString());*/
    cacheMisses++;
    return null;
  }
```

Figure 1: Code for the method in the cache that is used to get data



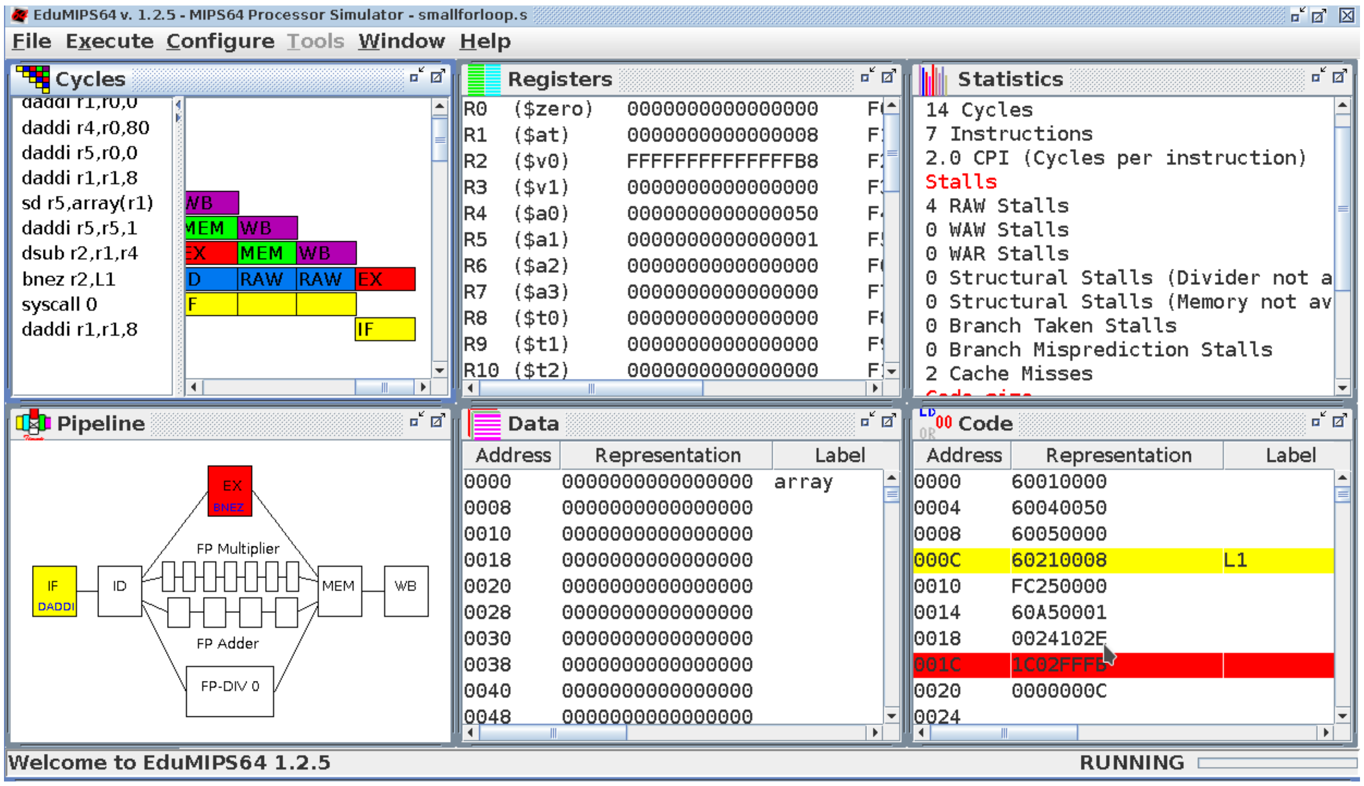
```java
private class CacheLine<Type> {

  private HashMap<Long, Type> line;
  public long startingAddress;
  public boolean data;

  private CacheLine(long startingAddress, InstructionInterface asdf) {
    this.startingAddress = startingAddress;
    line = new HashMap<Long, Type>();
    long end = startingAddress + blockSize;
    for (long i = startingAddress; i < end; i+=4) {
      line.put(i, (Type)memory.getInstruction((int)i));
    }
    data = false;
  }

  private CacheLine(long startingAddress, MemoryElement asdf) throws MemoryElementNotFoundException {
    this.startingAddress = startingAddress;
    line = new HashMap<Long, Type>();
    long end = startingAddress + blockSize;
    for (long i = startingAddress; i < end; i+=8) {
      line.put(i, (Type)memory.getCellByAddress(i));
    }
    data = true;
  }

  public Type getEntry(long address) {
    return line.get(address);
  }

  public String toString() {
    String s = "";
    for (Type v : line.values()) {
      s += v + "\n";
    }
    return s;
  }
}
```

Figure 2: The CacheLine class, used in the cache



```java
private CacheLine[] instructions, data, shared;
private int maxInstructionsEntries, maxDataEntries, maxSharedEntries;
private int cacheIndexes;
private boolean splitCache;
private int blockSize, setAssociativity;
private Memory memory;

private int cacheMisses;

public Cache(ConfigStore config, Memory memory) {
  splitCache = config.getBoolean(ConfigKey.SPLIT_CACHE);
  blockSize = config.getInt(ConfigKey.BLOCK_SIZE);
  setAssociativity = config.getInt(ConfigKey.SET_ASSOCIATIVITY);
  maxSharedEntries = TOTAL_CACHE_SIZE / blockSize;
  maxInstructionsEntries = maxDataEntries = maxSharedEntries / 2;
  this.memory = memory;
  cacheMisses = 0;

  if (splitCache) {
      instructions = new CacheLine[maxInstructionsEntries];
      data = new CacheLine[maxDataEntries];
      cacheIndexes = maxInstructionsEntries / setAssociativity;
      if (maxInstructionsEntries % setAssociativity > 0)
        cacheIndexes++;
  } else {
      shared = new CacheLine[maxSharedEntries];
      cacheIndexes = maxSharedEntries / setAssociativity;
      if (maxSharedEntries % setAssociativity > 0)
        cacheIndexes++;
  }
}
```

Figure 3: The constructor of the cache class



EduMIPS64 v. 1.2.5 - MIPS64 Processor Simulator - smallforloop.s

File Execute Configure Tools Window Help

Cycles

```
daddi r1,r0,0
daddi r4,r0,80
daddi r5,r0,0
daddi r1,r1,8
sd r5,array(r1)      WB
daddi r5,r5,1        MEM  WB
dsub r2,r1,r4        EX   MEM  WB
bnez r2,L1           D    RAW  RAW  EX
syscall 0            F
daddi r1,r1,8                       IF
```

Registers

```
R0   ($zero)   0000000000000000
R1   ($at)     0000000000000008
R2   ($v0)     FFFFFFFFFFFFFFB8
R3   ($v1)     0000000000000000
R4   ($a0)     0000000000000050
R5   ($a1)     0000000000000001
R6   ($a2)     0000000000000000
R7   ($a3)     0000000000000000
R8   ($t0)     0000000000000000
R9   ($t1)     0000000000000000
R10  ($t2)     0000000000000000
```

Statistics

```
14 Cycles
7 Instructions
2.0 CPI (Cycles per instruction)
Stalls
4 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls (Divider not a
0 Structural Stalls (Memory not av
0 Branch Taken Stalls
0 Branch Misprediction Stalls
2 Cache Misses
```

Pipeline

Data

Address	Representation	Label
0000	0000000000000000	array
0008	0000000000000000	
0010	0000000000000000	
0018	0000000000000000	
0020	0000000000000000	
0028	0000000000000000	
0030	0000000000000000	
0038	0000000000000000	
0040	0000000000000000	
0048	0000000000000000	

Code

Address	Representation	Label
0000	60010000	
0004	60040050	
0008	60050000	
000C	60210008	L1
0010	FC250000	
0014	60A50001	
0018	0024102E	
001C	1C02FFFB	
0020	0000000C	
0024		

Welcome to EduMIPS64 1.2.5 RUNNING

Figure 4: Program running the smallforloop benchmark with split cache, a block size of 256, and a set associativity of 16