Sean Atangan
May 8, 2020
Professor Rao
ECE 366

Cache Simulator

PART A) Reflection

Group Members: Sean Atangan

Activity Log:

April 28th	Fix Project 2, Clean code, Clean output, Develop Plan for Cache Simulator
April 29th	Write program intro, obtain user input, create bit configuration variables to match needed cache info (offset bits, setID bits, block#, tag, block age)
May 2nd	Build function cacheSend() to be called from LW and SW sub-functions, passing variables to be used for cache. Calls a cacheConfiguration function depending on user input.
May 4th	Added needed functions for test programs: Ori, Addu Debugged the following functions: Slt, Beq,
May 5th	Built cacheConfig1 and cacheConfig2 functions, debugged cacheConfig2
May 6th	Built cacheConfig3 and cacheconfig4 functions, Inserted print statements for user specified Detailed display view
May 7th	Debugging cacheConfig3, cacheConfig4, LRU functionality
May 8th	Write report

What are some features that were useful?

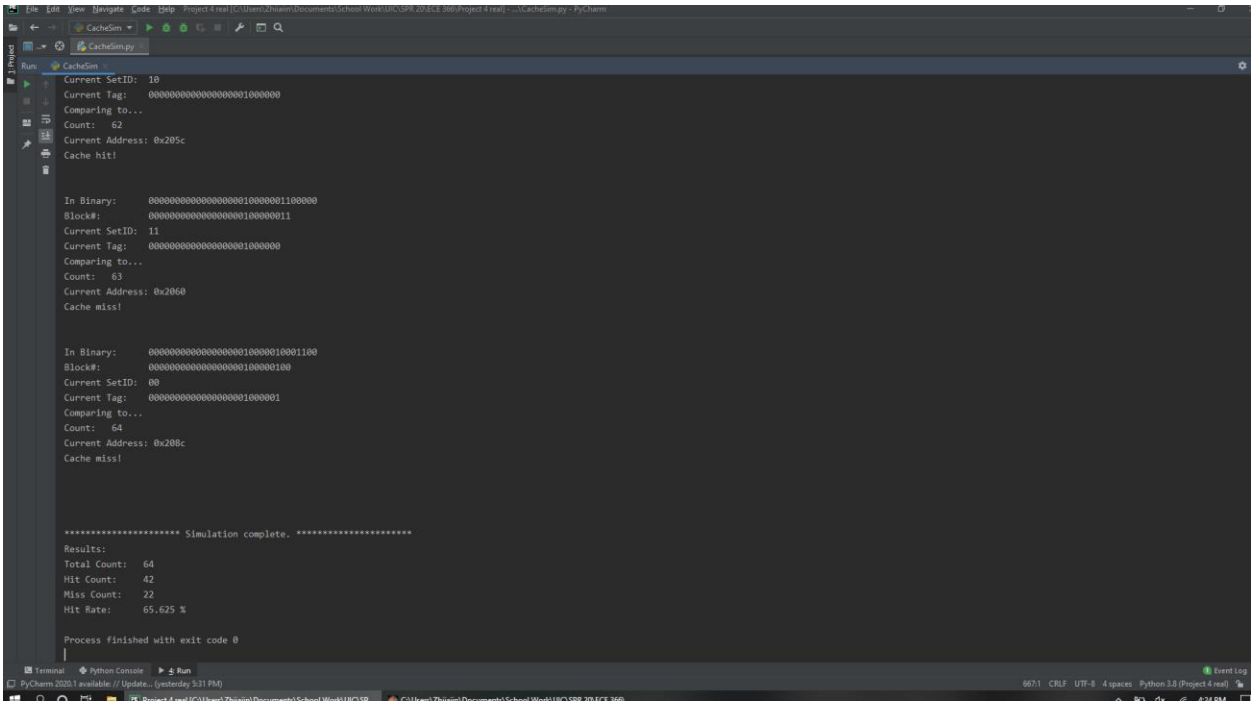
I found that the python debugger pdb.set_trace() was extremely useful for finding where my code was breaking or when cache information did not match the data cache simulator in MARS. I also really like how python utilizes indentations to determine scopes, which makes coding easier and cleaner that writing in C or C++.

What would I advise other students about the projects in this course?

I would recommend students to dig really deeply into the course material especially on days of which the content is relevant to the project. This may be already obvious to some, but really understanding how the machine you are building works, down to the smallest details, will help with the efficiency of your code as well as allow for other on the spot learning that may arise from coding in a new programming language. Otherwise, these projects are as fun as they are challenging, and they do a great service of teaching one how to think like an engineer. They also stress how crucial it is in the real world to be able to work in a team, and can uncover how difficult it can be to work on a real life problem all on your own.

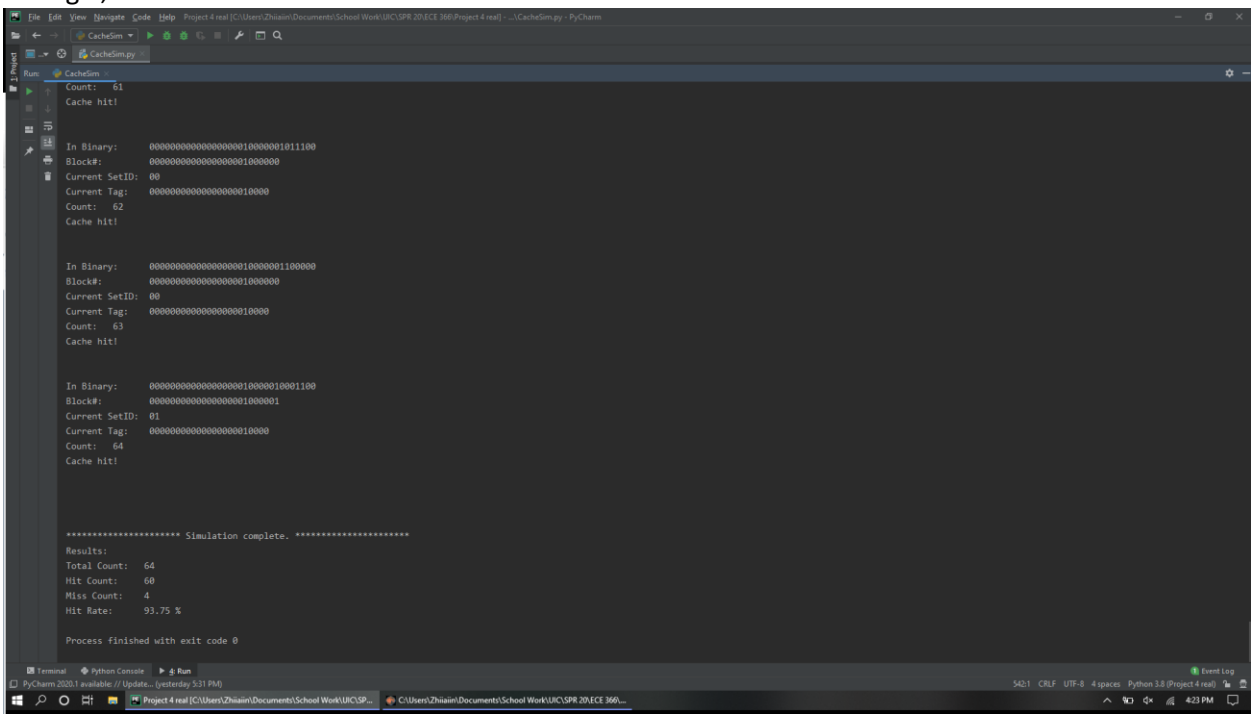
Sean Atangan
May 8, 2020
Professor Rao
ECE 366
PART B) Showcase

Config 1, detailed mode



```
Current SetID:  10
Current Tag:    00000000000000000001000000
Comparing to...
Count:   62
Current Address: 0x205c
Cache hit!


In Binary:      00000000000000000010000001100000
Block#:         00000000000000000100000011
Current SetID:  11
Current Tag:    00000000000000000001000000
Comparing to...
Count:   63
Current Address: 0x2060
Cache miss!


In Binary:      00000000000000000100000010001100
Block#:         00000000000000000100000100
Current SetID:  00
Current Tag:    00000000000000000001000001
Comparing to...
Count:   64
Current Address: 0x208c
Cache miss!



********************* Simulation complete. *********************
Results:
Total Count:   64
Hit Count:     42
Miss Count:    22
Hit Rate:      65.625 %

Process finished with exit code 0
```

Config 2, detailed mode



```
Count:   61
Cache hit!


In Binary:      00000000000000000010000001011100
Block#:         00000000000000000001000000
Current SetID:  00
Current Tag:    00000000000000000010000
Count:   62
Cache hit!


In Binary:      00000000000000000010000001100000
Block#:         00000000000000000001000000
Current SetID:  00
Current Tag:    00000000000000000010000
Count:   63
Cache hit!


In Binary:      00000000000000000100000010001100
Block#:         00000000000000000001000001
Current SetID:  01
Current Tag:    00000000000000000010000
Count:   64
Cache hit!



********************* Simulation complete. *********************
Results:
Total Count:   64
Hit Count:     60
Miss Count:    4
Hit Rate:      93.75 %

Process finished with exit code 0
```

Sean Atangan
May 8, 2020
Professor Rao
ECE 366
Config 3, detailed mode

```
Count:   61
Cache hit!


In Binary:      00000000000000000000000001011100
Block#:         00000000000000000000010000001
Current SetID:  01
Current Tag:    0x20
Count:   62
Cache hit!


In Binary:      00000000000000000000000001100000
Block#:         00000000000000000000010000001
Current SetID:  01
Current Tag:    0x20
Count:   63
Cache hit!


In Binary:      00000000000000000000000010001100
Block#:         00000000000000000000010000010
Current SetID:  10
Current Tag:    0x20
Count:   64
Cache hit!



********************* Simulation complete. *********************
Results:
Total Count:   64
Hit Count:     61
Miss Count:    3
Hit Rate:      95.3125 %

Process finished with exit code 0
```

Config 4, detailed mode

```
Count:   61
Cache hit!


In Binary:      00000000000000000000000001011100
Block#:         00000000000000000000010000001
Current SetID:  01
Current Tag:    0x20
Count:   62
Cache hit!


In Binary:      00000000000000000000000001100000
Block#:         00000000000000000000010000001
Current SetID:  01
Current Tag:    0x20
Count:   63
Cache hit!


In Binary:      00000000000000000000000010001100
Block#:         00000000000000000000010000010
Current SetID:  10
Current Tag:    0x20
Count:   64
Cache hit!



********************* Simulation complete. *********************
Results:
Total Count:   64
Hit Count:     61
Miss Count:    3
Hit Rate:      95.3125 %

Process finished with exit code 0
```

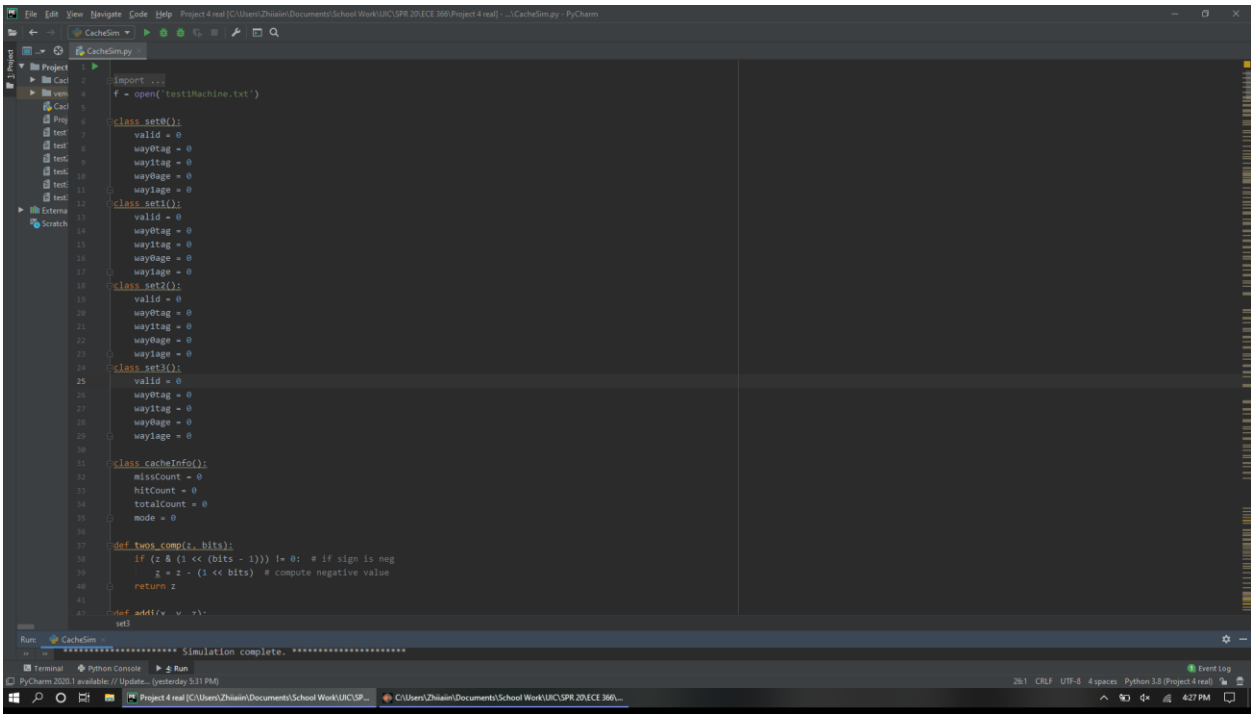
Sean Atangan
May 8, 2020
Professor Rao
ECE 366
FastMode (Similar for all)

```
Please select a cache configuration
 Enter...
1: 1 way, 1 set, 32 bytes
2: 1 way, 4 sets, 128 bytes
3: 2 way, 4 sets, 64 bytes
4: 4 way, 1 set, 64 bytes
1
1
Choose a mode
1: Detailed Display Mode
2: Fast Mode


2
2


********************* Simulation complete. *********************
Results:
Total Count:   64
Hit Count:     42
Miss Count:    22
Hit Rate:      65.625 %

Process finished with exit code 0
```

Other Code Snippets

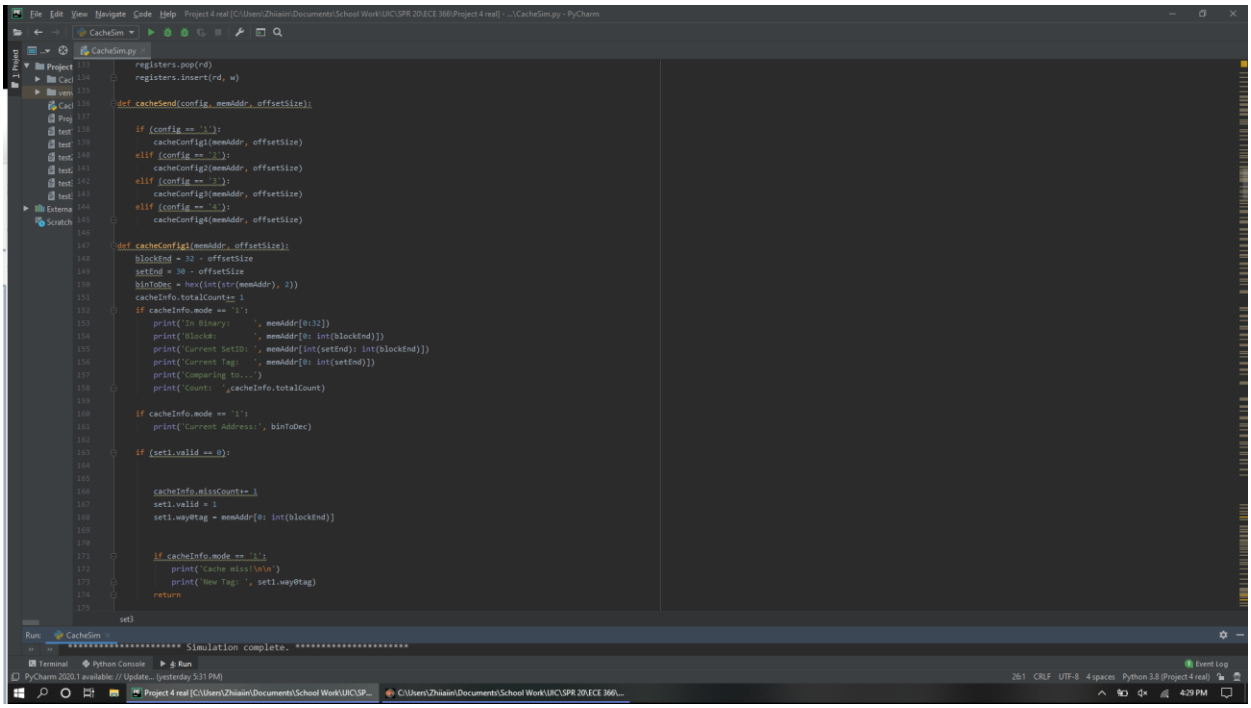
```
    registers.insert(y, w)

def branchtaken(z, pc):
    imm = hex(int(z, 2))
    imm = twos_comp(int(imm, 16), 16)
    pc = pc + 4 + (imm * 4)
    return pc

def storeword (x,y,z, userChoice, offset):

    z = twos_comp(z,16)
    t = registers[y]
    addr = registers[x] + z
    memory[hex(addr)] = t
    if cacheInfo.mode == 1:
        print('********************* STORE WORD: ********************* ')
        print('Hex Address:   ', hex(addr))
    hexAsBin = bin(int(hex(addr), 16))[2:].zfill(32)
    cacheSend(userChoice, hexAsBin, offset)

def loadword (x,y,z, userChoice, offset):  # rs,rt,imm

    z = twos_comp(z, 16)
    addr = registers[x] + z
    t = memory[hex(addr)]
    if cacheInfo.mode == 1:
        print('********************* LOAD WORD: ********************* ')
        print('Hex Address:   ', hex(addr))
    registers.pop(y)
    registers.insert(y, t)
    HexAsBin = bin(int(hex(addr), 16))[2:].zfill(32)
    cacheSend(userChoice, HexAsBin, offset)

def add(x, y, z):
    w = registers[z] + registers[y]
    registers.pop(x)
    registers.insert(x, w)

def sub(x, y, z):  # rd, rs, rt
    w = registers[y] - registers[z]
    registers.pop(x)
```

Sean Atangan
May 8, 2020
Professor Rao
ECE 366



```
import ...
f = open('test1Machine.txt')

class set0():
    valid = 0
    way0tag = 0
    way1tag = 0
    way0age = 0
    way1age = 0
class set1():
    valid = 0
    way0tag = 0
    way1tag = 0
    way0age = 0
    way1age = 0
class set2():
    valid = 0
    way0tag = 0
    way1tag = 0
    way0age = 0
    way1age = 0
class set3():
    valid = 0
    way0tag = 0
    way1tag = 0
    way0age = 0
    way1age = 0

class cacheInfo():
    missCount = 0
    hitCount = 0
    totalCount = 0
    mode = 0

def twos_comp(z, bits):
    if (z & (1 << (bits - 1))) != 0:  # if sign is neg
        z = z - (1 << bits)  # compute negative value
    return z

def addi(x, y, z):
```

```
********************* Simulation complete. *********************
```



```
    registers.pop(rd)
    registers.insert(rd, w)

def cacheSend(config, memAddr, offsetSize):

    if (config == '1'):
        cacheConfig1(memAddr, offsetSize)
    elif (config == '2'):
        cacheConfig2(memAddr, offsetSize)
    elif (config == '3'):
        cacheConfig3(memAddr, offsetSize)
    elif (config == '4'):
        cacheConfig4(memAddr, offsetSize)

def cacheConfig1(memAddr, offsetSize):
    blockEnd = 32 - offsetSize
    setEnd = 30 - offsetSize
    binToDec = hex(int(str(memAddr), 2))
    cacheInfo.totalCount+= 1
    if cacheInfo.mode == '1':
        print('In Binary:     ', memAddr[0:32])
        print('Block:         ', memAddr[0: int(blockEnd)])
        print('Current SetID: ', memAddr[int(setEnd): int(blockEnd)])
        print('Current Tag:   ', memAddr[0: int(setEnd)])
        print('Comparing to...')
        print('Count: ',cacheInfo.totalCount)

    if cacheInfo.mode == '1':
        print('Current Address:', binToDec)

    if (set1.valid == 0):


        cacheInfo.missCount+= 1
        set1.valid = 1
        set1.way0tag = memAddr[0: int(blockEnd)]


        if cacheInfo.mode == '1':
            print('Cache miss!\n\n')
            print('New Tag: ', set1.way0tag)
        return
```

```
********************* Simulation complete. *********************
```

Sean Atangan
May 8, 2020
Professor Rao
ECE 366

Overall Conclusions

	Test 1	Test 2	Test 3
Config 1 Hit Rate	**65.63%**	**72.88%**	**80.95%**
Config 2 Hit Rate	**93.75%**	**93.22%**	**93.65%**
Config 3 Hit Rate	**95.31%**	**94.91%**	**95.24%**
Config 4 Hit Rate	**95.31%**	**94.91%**	**95.24%**

I found that either Cache Configuration 3 of 4 yields the best performance. This data implies that more parameters are important for maximizing performance, and that different parameters for cache need to be customized depending on what hardware will use it. The main factors seem to be how much of the block number bits do you want to conserve verses how efficient do you want memory to be accessed. These are things to consider when working on CPUs and determining which configurations will optimize performance, at the same time ultimately save the company the most money during production.