Project 4: Advanced MIPS Simulator

Part A) Intro & reflections

1. Introduction of simulator:

a) **Functionality – does your simulator achieve (and if so what exactly) for the following components?**

- **MC**
- **Slow-Pipe**
- **Fast-Pipe**
- **Cache**
- **(extra credit) sorting algorithm comparison**
- **Interface – what are the input / output? how does it interact with user?**
  Our program achieves MC, Slow-pipe, fast-pipe, cache, and extra credit
  We had a user input for the b(#byes), S(number of sets 0 and N(Nways). We also gave the user the option to choose if they wanted to run Multicycle, slow pipeline of fast pipeline.

2. Project experience reflection:

- **List out your group members, and explain your general working style for this project.**
  Group Members: Pratik Jasani, Alex Smith, Vishal Parikh
  Our general working style for this project was by sharing the workload and bringing together the code. We each did slow pipeline, fast pipeline and multi cycle. We later brought together this code and started working on the cache together.
- **Are there any changes of how you work now from the previous projects?**
  Since this project was much harder compared to the 2nd project we each shared the workload and also spent some time understanding each other's coding style. Since we each had different coding style it was a little harder to bring together our codes. We found this project to be more enjoyable than project 2.
- **Tell us a few things that you have learned from your team members (including what to do and what NOT to do) throughout project 2, 3, and 4.**
  We organized our code so that if someone read it they would be able to understand what is going on. We commented each function and lines which we did not do because we found those projects to be a little easier. We learned how to properly organize out output on the output screen by using the format function. We learned how someone more experienced at code program compared to

someone else who is used to programming using a different language. We also learned that not commenting large sections of code can be very frustrating for someone else to look at and understand.

3. Tool reflection:

- **What are some of the features that you find useful in Python?**
  Some of the features that we find useful in Python is that it is a dynamically typed language. Not having to specify data types or return types for variables and functions made coding easier and faster overall since the system keeps track of that for you. Also, the way indentation works in Python to separate statements and loops made it very easy to read and allowed for much cleaner code.
- **What would you advise other students about Python programming for this course in the future?**
  We would advise them to lean how to organize the code because once we get into the habit of not organizing it, it becomes a lot harder for someone else understand. It is also a lot harder to debug the code if it is not organized. We would advise to planout how you want to program instead of just imping in and start coding.

Part B) General result + Processor details for Program X

1. Main findings.

- **In general, what's your conclusion about the performance comparison (as far as program X is concerned) among multi-cycle CPU, simple pipelined CPU (without forwarding), and aggressive pipeline (with all forwarding and hardware solution for hazards)? Is it worthy of all the design efforts and hardware overhead to implement a pipelined CPU, with all the forwarding hardware?**
  In our opinion we think it is worthy of our design efforts. Although it takes a lot of effort to implement, as long as it makes a program faster or helps save time it is worth it. The difference may not be very noticable in small programs that don't have a large instruction count, but once the programs become larger and larger, the speed of the program's execution time becomes much faster.

- **Support your conclusion by data in a table or chart – use your simulator to collect abundant data by running program X with different seeds (use the slt version). Your table and chart should be well annotated and easy to understand.**

Seed Value	# of cycles in MC	# of cycles in SP	# of cycles in FP
29	800	597	342
227	804	598	343
334	804	598	343
735	816	601	346

As the chart shows, the number of cycles in the program is significantly reduced when using the Fast-Pipeline implementation. Using Program X, the Fast-Pipeline is able to save the CPU hundreds of cycles of execution time. Similarly, in a larger program with more instructions, this difference in performance optimization would increase.

2. Simulator showcase

- Demonstrate the interface and results of your simulator running program X (see appendix).

Diagnosis Mode Snippets

```
===DEBUG MODE: INSTRUCTIONS CYCLE BY CYCLE===
Multi-Cycle: 0   |PC:  0   ori $8, $ 0, 24   Taking 4 cycles F : ori D :   E :   M :   W :
Slow-Pipeline: 0   |PC:  0   ori $8, $ 0, 24   Taking 1 cycle F : ori D :   E :   M :   W :
Fast-Pipeline: 0   |PC:  0   ori $8, $ 0, 24   Taking 1 cycle F : ori D :   E :   M :   W :



===DEBUG MODE: INSTRUCTIONS CYCLE BY CYCLE===
Multi-Cycle: 4   |PC:  4   addi $9, $ 0, 64   Taking 4 cycles F : addi D : ori E :   M :   W :
Slow-Pipeline: 1   |PC:  4   addi $9, $ 0, 64   Taking 1 cycle F : addi D : ori E :   M :   W :
Fast-Pipeline: 1   |PC:  4   addi $9, $ 0, 64   Taking 1 cycle F : addi D : ori E :   M :   W :
```

```
===DEBUG MODE: INSTRUCTIONS CYCLE BY CYCLE===
Multi-Cycle: 113 |PC:  24  addu $10, $10 , $8   Taking 4 cycles F : addu D : sll E : beq M : addi W : sw
Slow-Pipeline: 76  |PC:  24  addu $8, $ 10, 20513   Taking 1 cycle F : addu D : NOP E : NOP M : sll W : NOP
Fast-Pipeline: 41  |PC:  24  addu $8, $ 10, 20513   Taking 1 cycle F : addu D : sll E : beq M : Delay W : Delay



===DEBUG MODE: INSTRUCTIONS CYCLE BY CYCLE===
Multi-Cycle: 117 |PC:  28  sub $8, $0 , $8   Taking 4 cycles F : sub D : addu E : sll M : beq W : addi
Slow-Pipeline: 77  |PC:  28  sub $8, $ 0, 16418   Taking 1 cycle F : sub D : addu E : NOP M : NOP W : sll
Fast-Pipeline: 42  |PC:  28  sub $8, $ 0, 16418   Taking 1 cycle F : sub D : addu E : sll M : beq W : Delay
```

```
===DEBUG MODE: INSTRUCTIONS CYCLE BY CYCLE===
Multi-Cycle: 72  |PC:  12  addi $9, $ 9, -4   Taking 4 cycles F : addi D : sw E : beq M : xor W : sub
Slow-Pipeline: 46  |PC:  12  addi $9, $ 9, -4   Taking 1 cycle F : addi D : sw E : NOP M : NOP W : NOP
Fast-Pipeline: 25  |PC:  12  addi $9, $ 9, -4   Taking 1 cycle F : addi D : sw E : beq M : Delay W : xor


Slow Cycle: 47  |PC:  12  NOP taking 2 cycles
Fast Cycle: 26  |PC:  12  Control Hazard: Taken branch delay taking 1 cycle
Fast Cycle: 27  |PC:  12  Data Hazard: Compute-branch delay taking 1 cycle

===DEBUG MODE: INSTRUCTIONS CYCLE BY CYCLE===
Multi-Cycle: 76  |PC:  16  beq $9, $0,5   Taking 3 cycles F : beq D : addi E : sw M : beq W : xor
Slow-Pipeline: 49  |PC:  16  beq $0, $ 9, 5   Taking 1 cycle F : beq D : NOP E : NOP M : addi W : sw
Fast-Pipeline: 28  |PC:  16  beq $0, $ 9, 5   Taking 1 cycle F : beq D : Delay E : Delay M : addi W : sw
```

Line by line instruction output for each mode of processing

```
===DEBUG MODE: INSTRUCTIONS CYCLE BY CYCLE===
Multi-Cycle: 706 |PC:  64  addi $9, $ 9, 4   Taking 4 cycles F : addi D : addi E : bne M : slt W : lw
Slow-Pipeline: 515 |PC:  64  addi $9, $ 9, 4   Taking 1 cycle F : addi D : addi E : NOP M : NOP W : NOP
Fast-Pipeline: 292 |PC:  64  addi $9, $ 9, 4   Taking 1 cycle F : addi D : addi E : bne M : Delay W : Delay


Slow Cycle: 516 |PC:  64  NOP taking 2 cycles
Fast Cycle: 293 |PC:  64  Control Hazard: Taken branch delay taking 1 cycle
Fast Cycle: 294 |PC:  64  Data Hazard: Compute-branch delay taking 1 cycle

===DEBUG MODE: INSTRUCTIONS CYCLE BY CYCLE===
Multi-Cycle: 710 |PC:  68  bne $9, $10,-6   Taking 3 cycles F : bne D : addi E : addi M : bne W : slt
Slow-Pipeline: 518 |PC:  68  bne $10, $ 9, -6   Taking 1 cycle F : bne D : NOP E : NOP M : addi W : addi
Fast-Pipeline: 295 |PC:  68  bne $10, $ 9, -6   Taking 1 cycle F : bne D : Delay E : Delay M : addi W : addi
```

Hazard Handling for both the Slow-Pipeline and the Fast-Pipeline

Multi cycle - slt

```
1: Debug Mode
0: Normal Mode : 0

What mode Do u want to run 1:Multi Cycle     2:Slow Pipeline     3:Fast Pipelin  : 1


====================STATISTICS=========================
====================MULTI-CYCLE========================
Total # of cycles: 800
Dynamic Instruction count = 212
Dynamic Instruction Breakdown:
         64 instructions take 3 cycles
         132 instructions take 4 cycles
         16 instructions take 5 cycles
```

Slow pipeline - slt

```
1: Debug Mode
0: Normal Mode : 0

What mode Do u want to run 1:Multi Cycle     2:Slow Pipeline     3:Fast Pipelin  : 2

Enter the block size b(# of bytes): 8
Enter the number of sets S:  8
Enter N: 1

====================STATISTICS=========================
====================SLOW PIPELINE======================
Total # of cycles: 4 + 212 + 381 = 597
Total Instruction entering pipeline = 212
Finishing up the last instruction: 4
Total NOPs = 381
Dynamic Instruction count = 212

NOP Breakdown:
         192 Data Hazards
         189 Control Hazards

====================Cache==============================
Cache Hit   = 24
Cache Miss  = 9
 % Hit rate  = 72.72727272727273%
```

Fast pipeline - slt

```
1: Debug Mode
0: Normal Mode : 0

What mode Do u want to run 1:Multi Cycle     2:Slow Pipeline     3:Fast Pipelin  : 3

Enter the block size b(# of bytes): 8
Enter the number of sets S:  8
Enter N: 1

======================STATISTICS=========================
==================FAST PIPELINE============================
Total # of cycles: 4 + 212 + 62 + 64 = 342
Total Instruction entering pipeline = 212
Finishing up the last instruction: 4
Dynamic Instruction count = 212

Delay Breakdown:
         64 Data Hazards
         62 Control Hazards
Multi-Cycle Comparison: Fast Pipeline is 458 cycles faster
Slow Pipeline Comparison: Fast Pipeline is 255 cycles faster

=======================Cache=========================
Cache Hit  = 24
Cache Miss  = 9
 % Hit rate  = 72.72727272727273%
```

Multi Cycle - sltu

```
1: Debug Mode
0: Normal Mode : 0

What mode Do u want to run 1:Multi Cycle     2:Slow Pipeline     3:Fast Pipelin  : 1


======================STATISTICS=========================
====================MULTI-CYCLE=========================
Total # of cycles: 788
Dynamic Instruction count = 209
Dynamic Instruction Breakdown:
         64 instructions take 3 cycles
         129 instructions take 4 cycles
         16 instructions take 5 cycles
```

Slow Pipeline - sltu

```
1: Debug Mode
0: Normal Mode : 0

What mode Do u want to run 1:Multi Cycle     2:Slow Pipeline     3:Fast Pipelin  : 2

Enter the block size b(# of bytes): 8
Enter the number of sets S:  8
Enter N: 1

=====================STATISTICS=========================
====================SLOW PIPELINE========================
Total # of cycles: 4 + 209 + 381 = 594
Total Instruction entering pipeline = 209
Finishing up the last instruction: 4
Total NOPs = 381
Dynamic Instruction count = 209

NOP Breakdown:
         192 Data Hazards
         189 Control Hazards

=====================Cache=========================
Cache Hit  = 24
Cache Miss  = 9
 % Hit rate  = 72.72727272727273%
```

Fast Pipeline - sltu

```
1: Debug Mode
0: Normal Mode : 0

What mode Do u want to run 1:Multi Cycle     2:Slow Pipeline     3:Fast Pipelin  : 3

Enter the block size b(# of bytes): 8
Enter the number of sets S:  8
Enter N: 1

=====================STATISTICS=========================
=================FAST PIPELINE=========================
Total # of cycles: 4 + 209 + 62 + 64 = 339
Total Instruction entering pipeline = 209
Finishing up the last instruction: 4
Dynamic Instruction count = 209

Delay Breakdown:
         64 Data Hazards
         62 Control Hazards
Multi-Cycle Comparison: Fast Pipeline is 449 cycles faster
Slow Pipeline Comparison: Fast Pipeline is 255 cycles faster

=====================Cache=========================
Cache Hit  = 24
Cache Miss  = 9
 % Hit rate  = 72.72727272727273%
```

- You will be graded by program correctness (verify your simulator with as many as possible small programs with MARS), and interface design (make the output unique, easy to understand, and nice!).

Part C) (40pts) General result + Cache behavior for Program Y

1. Main findings

- **In general, what's your conclusion about the cache performance (in terms of overall hit rate of program Y) of the various cache configurations? What are some of the important tradeoffs of cache design to minimize miss rate?** One of the thing we can do the minimize the miss rate of the cache is to make your cache bigger, but the tradeoff is that it will be more expensive. If we just have a small cache then it will be cheaper, but it will have low hit rate. We can also make the size of he block bigger so it can hold more words. We can see in the table below that as words per block gets larger the hit rate increases. As we increase the set size the hit rate also increases.

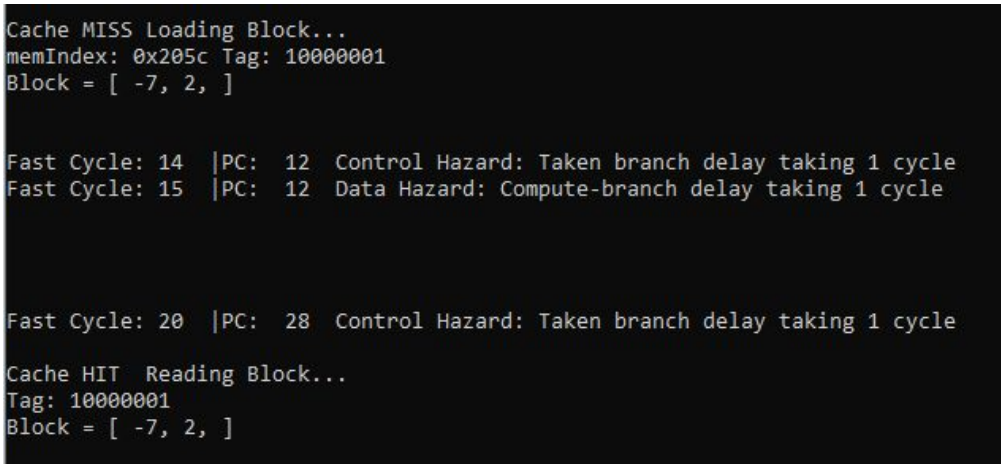
- Support your conclusion by data in a table or chart – use your simulator to collect abundant data by running program Y with various cache configurations. Your table and chart should be well annotated and easy to understand.

b(#bytes)	S	N	Hit rate(%)
32	1	1	65.625%
8	1	1	37.5%
4	1	1	0%
8	2	4	62.5%
8	4	4	71.875%
16	8	4	84.375%
4	1	8	20.3125%
4	1	16	28.125%

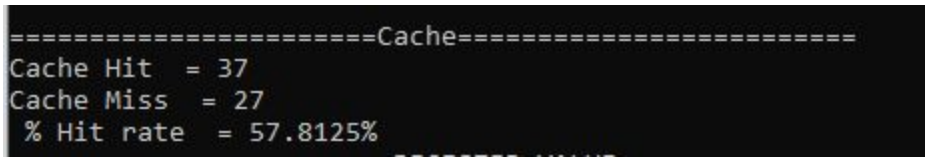
2. Simulator showcase Demonstrate the interface and results of your simulator running program Y (see appendix) on the three given cache settings:

a) a directly-mapped cache, block size of 2 words, a total of 8 blocks (b=8; N=1; S=8)

Debug Mode Snippet



Normal Mode



b) a fully-associative cache, block size of 4 words, a total of 4 blocks (b=16; N=4; S=1)

Debug Mode Snippet

```
Cache MISS!! Loading a block...
memIndex: 0x203c Tag: 1000000011
Block = [ -1537, 767, -385, 191, ]


Fast Cycle: 94  |PC:  12  Control Hazard: Taken branch delay taking 1 cycle
Fast Cycle: 95  |PC:  12  Data Hazard: Compute-branch delay taking 1 cycle




Fast Cycle: 100 |PC:  28  Control Hazard: Taken branch delay taking 1 cycle

Cache HIT!! Reading a block...
Tag: 1000000011
Block = [ -1537, 767, -385, 191, ]
```

Normal Mode

```
=======================Cache=========================
Cache Hit   = 50
Cache Miss  = 14
 % Hit rate  = 78.125%
```

a 2-way set-associative cache, block size of 8 bytes, 4 sets (b=8; N=2; S=4)

Debug Mode Snippet

```
Cache MISS!! Loading a block...
memIndex: 0x2090 Tag: 100000100
Block = [ -97, 0, ]



Fast Cycle: 528 |PC:  92  Control Hazard: Taken branch delay taking 1 cycle
Fast Cycle: 529 |PC:  92  Data Hazard: Compute-branch delay taking 1 cycle


Cache HIT!! Reading a block...
Tag: 100000010
Block = [ 47, -25, ]
```

Normal Mode

```
=======================Cache=========================
Cache Hit   = 40
Cache Miss  = 24
 % Hit rate  = 62.5%
```

If your simulator supports configurable cache setting via user input, showcase it with another setting of your choice:

c) an N-way set-associative cache, block size of b bytes, S sets (b = 4,N = 1,S = 2)

Debug Mode Snippet

```
Cache MISS Loading Block...
memIndex: 0x208c Tag: 10000010001
Block = [ -1537, ]



Fast Cycle: 486 |PC:  92  Control Hazard: Taken branch delay taking 1 cycle
Fast Cycle: 487 |PC:  92  Data Hazard: Compute-branch delay taking 1 cycle


Cache HIT  Reading Block...
Tag: 10000001001
Block = [ 191, ]
```

Normal Mode

```
=======================Cache=========================
Cache Hit   = 8
Cache Miss  = 56
 % Hit rate  = 12.5%
```

Part D) (10% extra credit): Sorting algorithm comparison

- Which sorting algorithms did you choose? What are the main findings?
  We chose Selection sort and bubble sort.
  - The selection sort algorithm(memory) sorts an array by repeatedly finding the minimum element from unsorted part and putting it at the beginning.
  - Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.
  - Selection sort is quicker sorting algorithm uses less cycles whereas bubble sort has a longer runtime
  - Higher cache hit rate
  - Less cycles

- Use a table or chart to support your main findings. Clearly provide all the parameters you have used to collect the data (how many rounds have you run, on what seeds, and what cache configurations did you use, etc).

Seed value	Cache config	Total #of cycles	Cache hit rate
24	B=8, S=8,N=1	Z1: 3264 Z2: 3428	Z1:88.84% Z2:93.92%
80	B=8, S=8,N=1	Z1: 3240 Z2: 3428	Z1:73.08% Z2:95.36%

- Use screenshots to demonstrate the interface and results of your simulator running program Z1 and Z2 (the two sorting algorithms).

- program Z1 on cache config B=8, S=8,N=1

```
1: Debug Mode
0: Normal Mode : 0

What mode Do u want to run 1:Multi Cycle     2:Slow Pipeline     3:Fast Pipelin  : 3

Enter the block size b(# of bytes): 8
Enter the number of sets S:  8
Enter N: 1

=====================STATISTICS=======================
=================FAST PIPELINE=========================
Total # of cycles: 4 + 1776 + 590 + 894 = 3264
Total Instruction entering pipeline = 1776
Finishing up the last instruction: 4
Dynamic Instruction count = 1776

Delay Breakdown:
         894 Data Hazards
         590 Control Hazards
Multi-Cycle Comparison: Fast Pipeline is 3626 cycles faster
Slow Pipeline Comparison: Fast Pipeline is 2336 cycles faster

=====================Cache=======================
Cache Hit  = 470
Cache Miss  = 59
 % Hit rate  = 88.84688090737241%
=====================REGISTER VALUE=======================
$0  = 0            $1  = 0           $2  = 0            $3  = 0
$4  = 0            $5  = 0           $6  = 0            $7  = 0
$8  = 0            $9  = 0           $10 = -1610612800     $11 = -805306400
$12 = 0            $13 = 0           $14 = 0            $15 = 0
$16 = 0            $17 = 0           $18 = 0            $19 = 0
$20 = 0            $21 = 0           $22 = 0            $23 = 0
$24 = 0            $25 = 0           $26 = 0            $27 = 0
$28 = 0            $29 = 0           $30 = 0            $31 = 0
=====================MEMORY VALUE=======================
$0x2000 = -805306400       $0x2004 = -1610612800     $0x2008 = -805306400       $0x200c = -402653200
$0x2010 = -262144          $0x2014 = -131072         $0x2018 = -65536           $0x201c = -32768
$0x2020 = -16384           $0x2024 = -8192           $0x2028 = -4096            $0x202c = -2048
$0x2030 = -1024            $0x2034 = -512            $0x2038 = 24               $0x203c = 1073741696
$0x2040 = 2147483392       $0x2044 = 0               $0x2048 = 0                $0x204c = 0
$0x2050 = 0                $0x2054 = 0               $0x2058 = 0                $0x205c = 0
$0x2060 = 0                $0x2064 = 0               $0x2068 = 0                $0x206c = 0
$0x2070 = 0                $0x2074 = 0               $0x2078 = 0                $0x207c = 0
$0x2080 = 0                $0x2084 = 0               $0x2088 = 0                $0x208c = 0
```

- program Z2 on cache config B=8, S=8,N=1

```
1: Debug Mode
0: Normal Mode : 0

What mode Do u want to run 1:Multi Cycle     2:Slow Pipeline     3:Fast Pipelin  : 3

Enter the block size b(# of bytes): 8
Enter the number of sets S:  8
Enter N: 1

=====================STATISTICS=========================
=================FAST PIPELINE==========================
Total # of cycles: 4 + 2067 + 526 + 831 = 3428
Total Instruction entering pipeline = 2067
Finishing up the last instruction: 4
Dynamic Instruction count = 2067

Delay Breakdown:
        831 Data Hazards
        526 Control Hazards
Multi-Cycle Comparison: Fast Pipeline is 4870 cycles faster
Slow Pipeline Comparison: Fast Pipeline is 2073 cycles faster

=====================Cache=========================
Cache Hit   = 649
Cache Miss  = 42
 % Hit rate  = 93.92185238784371%
=======================REGISTER VALUE=========================
$0  = 0           $1  = 0          $2  = 0           $3  = 0
$4  = 0           $5  = 0          $6  = 0           $7  = 0
$8  = 64          $9  = 68         $10 = 1073741696     $11 = 2147483392
$12 = 0           $13 = 0          $14 = 68          $15 = 0
$16 = 0           $17 = 0          $18 = 0           $19 = 0
$20 = 0           $21 = 0          $22 = 0           $23 = 0
$24 = 0           $25 = 0          $26 = 0           $27 = 0
$28 = 0           $29 = 0          $30 = 0           $31 = 0
=======================MEMORY VALUE=========================
$0x2000 = -402653200      $0x2004 = -1610612800     $0x2008 = -805306400      $0x200c = -402653200
$0x2010 = -262144         $0x2014 = -131072         $0x2018 = -65536          $0x201c = -32768
$0x2020 = -16384          $0x2024 = -8192           $0x2028 = -4096           $0x202c = -2048
$0x2030 = -1024           $0x2034 = -512            $0x2038 = 24              $0x203c = 1073741696
$0x2040 = 2147483392      $0x2044 = 0               $0x2048 = 0               $0x204c = 0
$0x2050 = 0               $0x2054 = 0               $0x2058 = 0               $0x205c = 0
$0x2060 = 0               $0x2064 = 0               $0x2068 = 0               $0x206c = 0
$0x2070 = 0               $0x2074 = 0               $0x2078 = 0               $0x207c = 0
$0x2080 = 0               $0x2084 = 0               $0x2088 = 0               $0x208c = 0
```

- Include the assembly code for your program Z1 and Z2 at the end.

```
#Z1
#Selection Sort
addi $8, $0, 0x40
loop_2:
addi $9, $0, 0x44
lw $10, 0x2000($8)
        loop_1:
        addi $9, $9, -4
        lw $11, 0x2000($9)
        slt $12, $11, $10
        beq $12, $0, else
                sw $10, 0x2000
                lw $10,
0x2000($9)
                sw $10,
0x2000($8)
                lw $11, 0x2000
                sw $11,
0x2000($9)
        else:
        bne $9, $0, loop_1
addi $8, $8, -4
bne $8, $0, loop_2
```

```
# Z2
# Bubble Sort
addi $13, $0, 0x40
addi $14, $0, 0x44
loop_2:
addi $8, $0, 4
addi $9, $0, 8
        loop_1:
        lw $10, 0x2000($8)
        lw $11, 0x2000($9)
        slt $12, $11, $10
        beq $12, $0, else
                sw $11, 0x2000
                lw $11,
0x2000($8)
                sw $11,
0x2000($9)
                lw $10, 0x2000
                sw $10,
0x2000($8)
        else:
        addi $8, $8, 4
        addi $9, $9, 4
        bne $9, $14, loop_1
addi $13, $13, -4
bne $13, $0, loop_2
```