4270 Lab report
Group 12

Lab 2: MIPS Assembler

Objective

The objective for this lab was develop an assembler for MIPS ISA. It will take a program written in MIPS assembly and will convert into MIPS machine code. The output of MIPS assembler will be an input to MIPS simulator.

Results

```
MU-MIPS SIM:> run 22
Running simulator for 22 cycles...

MU-MIPS SIM:> mdump 0x10010000 0x10010024
------------------------------------------------------------
Memory content [0x10010000..0x10010024] :
------------------------------------------------------------
        [Address in Hex (Dec) ] [Value]
        0x10010000 (268500992) :        0x00000005
        0x10010004 (268500996) :        0x00000003
        0x10010008 (268501000) :        0x00000006
        0x1001000c (268501004) :        0x00000008
        0x10010010 (268501008) :        0x00000009
        0x10010014 (268501012) :        0x00000001
        0x10010018 (268501016) :        0x00000004
        0x1001001c (268501020) :        0x00000007
        0x10010020 (268501024) :        0x00000002
        0x10010024 (268501028) :        0x0000000a

MU-MIPS SIM:> run 800
Running simulator for 800 cycles...

MU-MIPS SIM:> mdump 0x10010000 0x10010024
------------------------------------------------------------
Memory content [0x10010000..0x10010024] :
------------------------------------------------------------
        [Address in Hex (Dec) ] [Value]
        0x10010000 (268500992) :        0x00000001
        0x10010004 (268500996) :        0x00000002
        0x10010008 (268501000) :        0x00000003
        0x1001000c (268501004) :        0x00000004
        0x10010010 (268501008) :        0x00000005
        0x10010014 (268501012) :        0x00000006
        0x10010018 (268501016) :        0x00000007
        0x1001001c (268501020) :        0x00000008
        0x10010020 (268501024) :        0x00000009
        0x10010024 (268501028) :        0x0000000a

MU-MIPS SIM:>
```

Figure 1. Test result of bubble sort

```
MU-MIPS SIM:> run 100
Running simulator for 100 cycles...

MU-MIPS SIM:> rdump
-------------------------------------
Dumping Register Content
-------------------------------------
# Instructions Executed : 100
PC      : 0x004000dc
-------------------------------------
[Register]      [Value]
-------------------------------------
[R0]    : 0x00000000
[R1]    : 0x00000000
[R2]    : 0x00000000
[R3]    : 0x00000000
[R4]    : 0x00000000
[R5]    : 0x00000000
[R6]    : 0x00000000
[R7]    : 0x00000000
[R8]    : 0x00000015
[R9]    : 0x00000022
[R10]   : 0x00000037
[R11]   : 0x0000000a
[R12]   : 0x0000000a
[R13]   : 0x00000000
[R14]   : 0x00000000
[R15]   : 0x00000000
[R16]   : 0x00000000
[R17]   : 0x00000000
[R18]   : 0x00000000
[R19]   : 0x00000000
[R20]   : 0x00000000
[R21]   : 0x00000000
[R22]   : 0x00000000
[R23]   : 0x00000000
[R24]   : 0x00000000
[R25]   : 0x00000000
[R26]   : 0x00000000
[R27]   : 0x00000000
[R28]   : 0x00000000
[R29]   : 0x00000000
[R30]   : 0x00000000
[R31]   : 0x00000000
-------------------------------------
[HI]    : 0x00000000
[LO]    : 0x00000000
-------------------------------------
MU-MIPS SIM:>
```

Figure 2. Test result of Fibonacci number

Discussion

Work Distribution

Workload was divided evenly among all three group members and the total work was separated into approximately 3 milestones. Problem 1, problem 2, and problem 3, and write the report.

Zach	Sorting algorithm (other than bubble sort).
Zhu	Bubble sort and the MIPS Assembler.
Theo	Fibonacci number.

Table 1. Work Distributions

Implementation Decisions

Assembler

For the assembler, I created two array for all the opcode and registers we will use in this lab section, then based on the opcode of different instructions, grab the register information for rs, rt or rd or immediate value and offset, shift them for corresponding bits and add up all the components to get the machine code. For some of the instructions I also did some bit shifting for the immediate value based on the sign bit of the immediate value.
During the coding of the assembler, we also fixed some problems in our lab 1 MIPS simulator and finally we have our simulator works perfect.

Bubble sort

For the bubble sort, I first wrote the code in C which is:

```
void bubble_sort(int arr[], int len) {
        int i, j, temp;
        for (i = 0; i < len - 1; i++)
                for (j = 0; j < len - 1 - i; j++)
                        if (arr[j] > arr[j + 1]) {
                                temp = arr[j];
                                arr[j] = arr[j + 1];
                                arr[j + 1] = temp;
                        }
}
```

After that, I try to figure out how to store all the values into the data segment of the memory, before trying to sort them out. I noticed that the address for memory is 32bits long and the immediate value is only 16bits long, so I used an addiu instruction with a sll instruction to shift 16bit value by 16 bits to get 32bis long. After that, I used sw instruction with different offset to insert all the values to the memory.

```
addiu $a3,$zero,0x1001
sll $a0,$a3,16
addiu $t0,$zero,5
sw $t0,0($a0)
```

```
addiu $t0,$zero,3
sw $t0,4($a0)
addiu $t0,$zero,6
sw $t0,8($a0)
addiu $t0,$zero,8
sw $t0,12($a0)
addiu $t0,$zero,9
sw $t0,16($a0)
addiu $t0,$zero,1
sw $t0,20($a0)
addiu $t0,$zero,4
sw $t0,24($a0)
addiu $t0,$zero,7
sw $t0,28($a0)
addiu $t0,$zero,2
sw $t0,32($a0)
addiu $t0,$zero,a
sw $t0,36($a0)
```

First 22 instructions just look like that, and if you convert those instructions to machine code and test them, you will get all the values in the memory. The result is shown in Figure 1.
After that I start to implement the bubble sort. I initialized all the parameter to zero and initialized the upper bound of i and j. Then get the address of A[j] and A[j+1]. After that, compare those two by slt and used beq and bne for different branch. I did bubble sort by restore the correct order into the memory. The instructions are shown below:

```
addiu $t5,$zero,0
addiu $s0,$zero,0
addiu $a1,$zero,9
addiu $s1,$zero,0
sub $a2,$a1,$s0
sll $t0,$s1,2
add $t2,$a0,$t0
lw $t1,0($t2)
lw $t3,4($t2)
slt $t4,$t3,$t1
beq $t4,$zero,2
bne $t4,$zero,4
addiu $s1,$s1,1
bne $s1,$a2,-8
beq $s1,$a2,6
add $t5,$zero,$t1
sw $t3,0($t2)
```

```
sw $t5,4($t2)
addiu $s1,$s1,1
bne $s1,$a2,-e
addiu $s0,$s0,1
bne $s0,$a1,-12
syscall
```

Convert all the instructions to machine code, and run lots of time (in my case I chose 800) to complete the for loop, finally I got every values sorted, which also shown in Figure 1.

Selection Sort

For problem 2 we chose the selection sort algorithm because it seemed fairly simple to implement. We used the example C implementation from wikipedia as a reference and transcribed it into MIPS assembly. We started on selection sort after the bubble sort was working so we could use the bubble sort assembly as a reference which was helpful.

```
/* a[10] to a[n-1] is the array to sort */
int i,j;
int a[10];

/* advance the position through the entire array */
/*   (could do j < n-1 because single element is also min element) */
for (j = 0; j < n-1; j++)
  {
    /* find the min element in the unsorted a[j .. n-1] */

    /* assume the min is the first element */
    int iMin = j;
    /* test against elements after j to find the smallest */
    for (i = j+1; i < n; i++) {
        /* if this element is less, then it is the new minimum */
        if (a[i] < a[iMin]) {
            /* found new minimum; remember its index */
            iMin = i;
        }
    }

    if(iMin != j)
    {
        swap(a[j], a[iMin]);
    }
```

}

Figure 4. C implementation of selection sort (from Wikipedia) - used as reference.

We ran out of time near the end to finish testing the selection sort. Still need to modify the assembler to handle some extra registers, because selection sort implementation required more registers than the other two lab problems. Additionally there were some unimplemented instructions that needed changed at the last minute. E.g. we didn't implement unconditional jump properly yet so we used beq $zero,$zero,<addr> as replacement. Lastly, the jump immediates were originally written in decimal so needed to be changed to hex immediates.

Comparison of Bubble and Selection Sorts

For both algorithms, the average-case complexity is O(n^2) comparisons. Selection sort usually outperforms bubble sort especially with large lists. One of the main advantages of bubble sort is that the best case performance improves when the list is already pre-sorted or partially sorted.

At this point we don't have a good way of reasoning about the difference in number of instructions. The bubble sort implementation required at least 822 cycles to run, but the 800 number is not precise. Probably the best way to compare them would be to trace execution in both cases i.e. unroll the loops and compare the raw numbers of instructions and total number of cycles. If we use the assembly files as a heuristic it appears that bubble sort uses less instructions and cycles than the selection sort. Some of the differences can also be attributed to implementation details e.g. writing the assembly in the optimal way such as to minimize number of branches, jumps etc.

Fibonacci number

For problem 3, we have first check the concept of fibonacci number:

//r1 = $t0 = 2 numbers ago
//r2 = $t1 = 1 number ago
//r3 = $t2 = current number (where the end result will be stored)
//r4 = $t3 = counter
//r5 = $t4 = USER INPUT number within sequence
the current number will be the fibonacci number
After that, we generated instructions based on the concept which are
addiu $t4,$zero,A
addiu $t3,$zero,0
addiu $t0,$zero,0
addiu $t1,$zero,1
addiu $t2,$zero,0
beq $t3,$t4,18

```
addiu $t0,$t1,0
addiu $t1,$t2,0
add $t2,$t0,$t1
addiu $t3,$t3,1
bne $t3,$t4,-10
syscall
```

Test result of that is shown in figure 3, we test the the fibonacci number of 10 in this case and run it 100 times to complete the process. After that, we do rdump to check the values in the register, the number which is stored in register t2 (R10) is showing 37 in hex and 37(hex) is 55 in decimal, which is the fibonacci number of 10.

Conclusions

For this lab we developed an assembler for MIPS ISA. It takes a program written in MIPS assembly and converts into MIPS machine code. The output of MIPS assembler will be an input to MIPS simulator completed in Lab 1. In our case we got the bubble sort and fibonacci number implementations working, but still need to do work on selection sort to get it working. The main things we learned in this lab were the basics of writing an assembler and how to write MIPS assembly. Additionally, the requirement to use our simulator from Lab 1 was helpful in that it gave us an opportunity to improve and fix mistakes from the first two weeks.