Week 05 Laboratory Sample Solutions

Objectives

- understanding how array indices are calculated
- practicing using MIPS control instructions (branch)
- learning how MIPS memory access works (lw/sw)
- practicing running MIPS programs with spim, xspim or qtspim

---

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

Getting Started

Create a new directory for this lab called `lab05`, change to this directory, and fetch the provided code for this week by running these commands:

```
$ mkdir lab05
$ cd lab05
$ 1521 fetch lab05
```

Or, if you're not working on CSE, you can download the provided code as a zip file or a tar file.

EXERCISE — INDIVIDUAL:

Bigger MIPS

In the files for this lab, you have been given `print_bigger.s`, a MIPS assembler program that reads 10 numbers and then prints them:

```
$ cat numbers1.txt
12086
24363
47363
64268
34001
6800
60742
48867
26002
54999
$ 1521 spim -f print_bigger.s <numbers1.txt
12086
24363
47363
64268
34001
6800
60742
48867
26002
54999
```

Add code to `print_bigger.s` to make it equivalent to this C program:

```c
// Read 10 numbers into an array
// then print the numbers which are
// larger than the last number read.

#include <stdio.h>

int main(void) {
    int i, last_number;
    int numbers[10] = { 0 };

    i = 0;
    while (i < 10) {
        scanf("%d", &numbers[i]);
        last_number = numbers[i];
        i++;
    }
    i = 0;
    while (i < 10) {
        if (numbers[i] >= last_number) {
            printf("%d\n", numbers[i]);
        }
        i++;
    }
}
```

For example:

```
$ 1521 spim -f print_bigger.s <numbers1.txt
64268
60742
54999
$ cat numbers2.txt
53906
9064
40906
4504
4774
7892
15334
45515
55387
5681
$ 1521 spim -f print_bigger.s <numbers2.txt
53906
9064
40906
7892
15334
45515
55387
5681
```

HINT:

Start by choosing which register you will use for each variable.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest print_bigger
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab05_print_bigger print_bigger.s
```

You must run `give` before **Monday 26 October 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `print_bigger.s`

```
# Read 10 numbers into an array
# then print the numbers which are
# larger than the last number read.

# i in register $t0, last_number in $t4
# registers $t1, $t2 & $t3 used to hold temporary results

main:
    li   $t0, 0         # i = 0
loop0:
    bge  $t0, 10, end0  # while (i < 10) {

    li   $v0, 5         #   scanf("%d", &numbers[i]);
    syscall             #

    mul  $t1, $t0, 4    #   calculate &numbers[i]
    la   $t2, numbers   #
    add  $t3, $t1, $t2  #
    sw   $v0, ($t3)     #   store entered number in array

    move $t4, $v0       # last_number = numbers[i]

    addi $t0, $t0, 1    #   i++;
    j    loop0          # }
end0:

    li   $t0, 0         # i = 0
loop1:
    bge  $t0, 10, end1  # while (i < 10) {

    mul  $t1, $t0, 4    #   calculate &numbers[i]
    la   $t2, numbers   #
    add  $t3, $t1, $t2  #
    lw   $a0, ($t3)     #   load numbers[i] into $a0

    blt  $a0, $t4, skip #   if (numbers[i] >= last_number)

    li   $v0, 1         #   printf("%d", numbers[i])
    syscall

    li   $a0, '\n'      #   printf("%c", '\n');
    li   $v0, 11
    syscall

skip:

    addi $t0, $t0, 1    #   i++
    j    loop1          # }
end1:

    jr   $ra            # return

.data

numbers:
    .word 0 0 0 0 0 0 0 0 0 0  # int numbers[10] = {0};
```

EXERCISE — INDIVIDUAL:

MIPS Order Checking

In the files for this lab, you have been given unordered.s, a MIPS assembler program that reads 10 numbers and then prints 42:

Add code to `unordered.s` to make it equivalent to this C program:

```c
// Read 10 numbers into an array
// print 0 if they are in non-decreasing order
// print 1 otherwise

#include <stdio.h>

int main(void) {
    int i;
    int numbers[10] = { 0 };

    i = 0;
    while (i < 10) {
        scanf("%d", &numbers[i]);
        i++;
    }

    int swapped = 0;
    i = 1;
    while (i < 10) {
        int x = numbers[i];
        int y = numbers[i - 1];
        if (x < y) {
            swapped = 1;
        }
        i++;
    }

    printf("%d\n", swapped);
}
```

For example:

```
$ cat numbers1.txt
12086
24363
47363
64268
34001
6800
60742
48867
26002
54999
$ 1521 spim -f unordered.s <numbers1.txt
1
$ cat sorted.txt
1
2
3
4
5
6
7
8
9
10
$ 1521 spim -f unordered.s <sorted.txt
0
```

HINT:

Start by choosing which register you will use for each variable.

Reading the 10 numbers in every time can be inconvenient when debugging.

You can comment out the loop that reads the 10 numbers and just initialize the array to values useful for debugging.

See `unordered_hard_coded.s` for an example of this.

Don't forget to uncomment the loopthat reads the 10 numbers before autotesting.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest unordered
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab05_unordered unordered.s
```

You must run `give` before **Monday 26 October 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `unordered.s`

```
# Read 10 numbers into an array
# print 0 if they are in non-decreasing order
# print 1 otherwise

# i in register $t0, swapped in $t5, x in $t6, y in $t7
# registers $t1 - $t4 used to hold temporary results

main:

    li   $t0, 0         # i = 0
loop0:
    bge  $t0, 10, end0  # while (i < 10) {

    li   $v0, 5         #   scanf("%d", &numbers[i]);
    syscall             #

    mul  $t1, $t0, 4    #   calculate &numbers[i]
    la   $t2, numbers   #
    add  $t3, $t1, $t2  #
    sw   $v0, ($t3)     #   store entered number in array

    addi $t0, $t0, 1    #   i++;
    j    loop0          # }
end0:

    li   $t5, 0          #   swapped = 0
    li   $t0, 1          #   i = 1
loop2:
    bge  $t0, 10, end2   #   while (i < 10) {

    mul  $t1, $t0, 4     #
    la   $t2, numbers    #
    add  $t3, $t1, $t2   #      $t3 = &numbers[i]
    lw   $t6, ($t3)      #      y = numbers[i]

    addi $t4, $t3, -4     #      $t4 = &numbers[i - 1]
    lw   $t7, ($t4)      #      x = numbers[i - 1]

    bge  $t6, $t7, skip  #      if (x < y) {
    li   $t5, 1          #        swapped = 1
skip:

    addi $t0, $t0, 1     #      i++;
    j    loop2           #   }
end2:

    move $a0, $t5       # load swapped into $a0
    li   $v0, 1         # printf("%d", numbers[i])
    syscall

    li   $a0, '\n'       # printf("%c", '\n');
    li   $v0, 11
    syscall

    jr   $ra            # return

.data

numbers:
    .word 0 0 0 0 0 0 0 0 0 0  # int numbers[10] = {0};
```

EXERCISE — INDIVIDUAL:

MIPS Swapping

In the files for this lab, you have been given swap_numbers.s, a MIPS assembler program that reads 10 numbers and then prints them:

Add code to `swap_numbers.s` to make it equivalent to this C program:

```c
// Read 10 numbers into an array
// swap any pair of numbers which are out of order
// then print the array

#include <stdio.h>

int main(void) {
    int i;
    int numbers[10] = { 0 };

    i = 0;
    while (i < 10) {
        scanf("%d", &numbers[i]);
        i++;
    }

    i = 1;
    while (i < 10) {
        int x = numbers[i];
        int y = numbers[i - 1];
        if (x < y) {
            numbers[i] = y;
            numbers[i - 1] = x;
        }
        i++;
    }

    i = 0;
    while (i < 10) {
        printf("%d\n", numbers[i]);
        i++;
    }
}
```

For example:

```
$ 1521 spim -f swap_numbers.s <numbers1.txt
12086
24363
47363
34001
6800
60742
48867
26002
54999
64268
```

```
$ 1521 spim -f swap_numbers.s <numbers2.txt
9064
40906
4504
4774
7892
15334
45515
53906
5681
55387
```

HINT:

Start by choosing which register you will use for each variable.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest swap_numbers
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab05_swap_numbers swap_numbers.s
```

You must run `give` before **Monday 26 October 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `swap_numbers.s`

```
# read 10 numbers into an array
# swap any pairs of of number which are out of order
# then print the 10 numbers

# i in register $t0,  x in $t6, y in $t7
# registers $t1 - $t4 used to hold temporary results

main:

    li   $t0, 0         # i = 0
loop0:
    bge  $t0, 10, end0  # while (i < 10) {

    li   $v0, 5         #   scanf("%d", &numbers[i]);
    syscall             #

    mul  $t1, $t0, 4    #   calculate &numbers[i]
    la   $t2, numbers   #
    add  $t3, $t1, $t2  #
    sw   $v0, ($t3)     #   store entered number in array

    addi $t0, $t0, 1    #   i++;
    j    loop0          # }
end0:

    li   $t0, 1         # i = 1
loop2:
    bge  $t0, 10, end2  # while (i < 10) {

    mul  $t1, $t0, 4    #
    la   $t2, numbers   #
    add  $t3, $t1, $t2  #   $t3 = &numbers[i]
    lw   $t6, ($t3)     #   y = numbers[i]

    addi $t4, $t3, -4    #   $t4 = &numbers[i - 1]
    lw   $t7, ($t4)     #   x = numbers[i - 1]

    bge  $t6, $t7, skip2 #   if (x < y) {

    sw   $t7, ($t3)     #     numbers[i] = y
    sw   $t6, ($t4)     #     numbers[i - 1] = x
                        #   }
skip2:
    addi $t0, $t0, 1    #   i++;
    j    loop2          # }
end2:

    li   $t0, 0         # i = 0
loop1:
    bge  $t0, 10, end1  # while (i < 10) {

    mul  $t1, $t0, 4    #   calculate &numbers[i]
    la   $t2, numbers   #
    add  $t3, $t1, $t2  #
    lw   $a0, ($t3)     #   load numbers[i] into $a0
    li   $v0, 1         #   printf("%d", numbers[i])
    syscall

    li   $a0, '\n'      #   printf("%c", '\n');
    li   $v0, 11
    syscall

    addi $t0, $t0, 1    #   i++
    j    loop1          # }
end1:

    jr   $ra            # return

.data

numbers:
    .align 2            # ensure array aligned ona 4-byte boundary
    .word 0 0 0 0 0 0 0 0 0 0  # int numbers[10] = {0};
```

CHALLENGE EXERCISE — INDIVIDUAL:

MIPS Bubbles

In the files for this lab, you have been given bubblesort.s, a MIPS assembler program that reads 10 numbers and then prints them:

Add code to `bubblesort.s` to make it equivalent to this C program:

```c
// Read 10 numbers into an array
// bubblesort them
// then print them

#include <stdio.h>

int main(void) {
    int i;
    int numbers[10] = { 0 };

    i = 0;
    while (i < 10) {
        scanf("%d", &numbers[i]);
        i++;
    }

    int swapped = 1;
    while (swapped) {
        swapped = 0;
        i = 1;
        while (i < 10) {
            int x = numbers[i];
            int y = numbers[i - 1];
            if (x < y) {
                numbers[i] = y;
                numbers[i - 1] = x;
                swapped = 1;
            }
            i++;
        }
    }

    i = 0;
    while (i < 10) {
        printf("%d\n", numbers[i]);
        i++;
    }
}
```

For example:

```
$ 1521 spim -f bubblesort.s <numbers1.txt
6800
12086
24363
26002
34001
47363
48867
54999
60742
64268
$ 1521 spim -f bubblesort.s <numbers2.txt
4504
4774
5681
7892
9064
15334
40906
45515
53906
55387
```

HINT:

Start by choosing which register you will use for each variable.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest bubblesort
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab05_bubblesort bubblesort.s
```

You must run `give` before **Monday 26 October 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `bubblesort.s`

```
# read 10 numbers into an array bubblesort them then print the 10 numbers

# i in register $t0, swapped in $t5, x in $t6, y in $t7
# registers $t1 - $t4 used to hold temporary results

main:

    li   $t0, 0         # i = 0
loop0:
    bge  $t0, 10, end0  # while (i < 10) {

    li   $v0, 5         #   scanf("%d", &numbers[i]);
    syscall             #

    mul  $t1, $t0, 4    #   calculate &numbers[i]
    la   $t2, numbers   #
    add  $t3, $t1, $t2  #
    sw   $v0, ($t3)     #   store entered number in array

    addi $t0, $t0, 1    #   i++;
    j    loop0          # }
end0:

    li   $t5, 1          # swapped = 1
sort:
    beq  $t5, 0, end_sort# while (swapped) {
    li   $t5, 0          #   swapped = 0
    li   $t0, 1          #   i = 1
loop2:
    bge  $t0, 10, end2   #   while (i < 10) {

    mul  $t1, $t0, 4     #
    la   $t2, numbers    #
    add  $t3, $t1, $t2   #      $t3 = &numbers[i]
    lw   $t6, ($t3)      #      y = numbers[i]

    addi $t4, $t3, -4     #      $t4 = &numbers[i - 1]
    lw   $t7, ($t4)      #      x = numbers[i - 1]

    bge  $t6, $t7, skip  #      if (x < y) {

    sw   $t7, ($t3)      #        numbers[i] = y
    sw   $t6, ($t4)      #        numbers[i - 1] = x
    li   $t5, 1          #        swapped = 1
                         #      }
skip:
    addi $t0, $t0, 1     #      i++;
    j    loop2           #   }
end2:
    j    sort            # }

end_sort:

    li   $t0, 0         # i = 0
loop1:
    bge  $t0, 10, end1  # while (i < 10) {

    mul  $t1, $t0, 4    #   calculate &numbers[i]
    la   $t2, numbers   #
    add  $t3, $t1, $t2  #
    lw   $a0, ($t3)     #   load numbers[i] into $a0
    li   $v0, 1         #   printf("%d", numbers[i])
    syscall

    li   $a0, '\n'      #   printf("%c", '\n');
    li   $v0, 11
    syscall

    addi $t0, $t0, 1    #   i++
    j    loop1          # }
end1:

    jr   $ra            # return

.data
```

```
.data

numbers:
    .word 0 0 0 0 0 0 0 0 0 0  # int numbers[10] = {0};
```

CHALLENGE EXERCISE — INDIVIDUAL:

MIPS NUXI

We have two 32 bit values which the bytes have placed in an unknown order.

Fortunately we know the 4 bytes of the first value contains the ASCII values "UNIX"

Write a MIPS program nuxi.s which read the two values and prints the second value with its bytes correctly ordered.

For example:

```
$ 1521 spim -f nuxi.s
1481199189
-2023406815
-2023406815
$ 1521 spim -f nuxi.s
1431193944
-2023406815
558065031
$ 1521 spim -f nuxi.s
1230525774
-559038737
-1377898562
$ 1521 spim -f nuxi.s
1229871189
305419896
878056056
```

HINT:

A major part of this challenge is understanding the problem. Started by printing the signed integers above in hexadecimal.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest nuxi
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab05_nuxi nuxi.s
```

You must run `give` before **Monday 26 October 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `nuxi.s`

```
    .data
first:  .word 0
second: .word 0
result: .word 0

    .text
main:
    addiu   $sp,  $sp, -4   # store $ra on the stack, since
    sw  $ra, ($sp)  # main calls another fn "find_letter".


    li  $v0, 5      # scanf("%d", &first);
    syscall
    sw  $v0, first

    li  $v0, 5      # scanf("%d", &second);
    syscall
    sw  $v0, second


    li  $a0, 'U'    # int u = find_letter('U');
    jal find_letter
    move    $s0, $v0

    li      $a0, 'N'    # int n = find_letter('N');
    jal     find_letter
    move    $s1, $v0

    li      $a0, 'I'    # int i = find_letter('I');
    jal     find_letter
    move    $s2, $v0

    li      $a0, 'X'    # int x = find_letter('X');
    jal     find_letter
    move    $s3, $v0


    lb  $t0, second + 0     # result[u] = second[0]
    sb  $t0, result($s0)

    lb      $t0, second + 1     # result[n] = second[1]
    sb      $t0, result($s1)

    lb      $t0, second + 2     # result[i] = second[2]
    sb      $t0, result($s2)

    lb      $t0, second + 3     # result[x] = second[3]
    sb      $t0, result($s3)


    lw  $a0, result # printf("%d", result);
    li  $v0, 1
    syscall

    li  $a0, '\n'   # putchar('\n');
    li  $v0, 11
    syscall


    lw  $ra, ($sp)  # restore $ra from the stack
    addiu   $sp,  $sp, 4

    li  $v0, 0      # return 0;
    jr  $ra




# $a0 = input letter
# $v0 = returned index
find_letter:
        li      $v0, -1                                 # int ret = -1;
        li      $t0, 0                                  # int index = 0;

find_letter_while_cond:
```

```
        blt     $t0, 4, find_letter__while_body         # while (index < 4) {
        j       find_letter__while_end

find_letter__while_body:
        lb      $t1, first($t0)                         #       int byte = first_bytes[index];
        bne     $t1, $a0, find_letter__while_incr       #       if (byte == input) {

        move    $v0, $t0                                #               ret = index;
        j       find_letter__while_end                  #               break;
                                                        #       }
find_letter__while_incr:
        addiu   $t0, $t0, 1                             #       index++;
        j       find_letter__while_cond                 # }

find_letter__while_end:
        jr      $ra                                     # return ret;
```

Submission

When you are finished each exercises make sure you submit your work by running `give`.

You can run `give` multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Mon Oct 26 21:00:00 2020** to submit your work.

You cannot obtain marks by e-mailing your code to tutors or lecturers.

You check the files you have submitted here.

Automarking will be run by the lecturer several days after the submission deadline, using test cases different to those `autotest` runs for you. (Hint: do your own testing as well as running `autotest`.)

After automarking is run by the lecturer you can view your results here. The resulting mark will also be available via give's web interface.

Lab Marks

When all components of a lab are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 1521 classrun -sturec
```