Computer Systems Fundamentals

i_love_mips.c

print a string in C

```
#include <stdio.h>

int main(void) {
    printf("%s", "I love MIPS\n");

    return 0;
}
```

i_love_mips.s

print a string in MIPS assembly

```
main:
        la      $a0, string     # ... pass address of string as argument
        li      $v0, 4          # ... 4 is printf "%s" syscall number
        syscall

        # return 0
        li      $v0, 0
        jr      $ra

        .data
string:
        .asciiz "I love MIPS\n"
```

i_love_mips.1.s

print a string in MIPS assembly

On: 2023-02-12

Note: in this program, comments that are preceded by two # characters are what we would expect to see in submitted labs and assignments comments that are preceded by one # character are additional information for lectures and revision

Constant for the magic number of the print string syscall

```
SYSCALL_PRINT_STRING = 4 # As defined by the MIPSY ABI (application binary interface)

## Constant for the return value of the main function
EXIT_SUCCESS = 0         # In C, this is defined in `stdlib.h`

# Define some data
# Everything after the `.data` directive until the next `.text` directive is "data" and not "code"
.data
        string: .asciiz "I love MIPS\n"
# `string` is a label, it is a name that can be used to refer to the address of the data (a pointer)
# The `.asciiz` directive is used to define a null-terminated string (just like "double quotes" in C)
# we can also use the `.ascii` directive and manually add a null byte at the end of the string (e.g. `.ascii "I love MIPS\n\0")

# Define some "text"
# "text" is the executable code of the program
# Everything after the `.text` directive until the next `.data` directive is "code" and not "data"
# There is also a implicit `.text` directive at the beginning of the program so everything before the first `.data` directive is "code" and not "data"
.text
# The `.globl` directive is used to define a global symbol, this isn't necessary for MIPSY
# but you might see it in other MIPS assembly programs
# (why is the word "global" spelled with an "a"? who knows?)
.globl main
# `main` is a label (just like `string`), it is a name that can be used to refer to this address
# but instead of referring to the address of some data, it refers to the address of some code (a function)
# this means that a function is just a pointer
# Just like in C, all MIPS assembly programs must have a `main` function
# the `main` function is the entry point of the program
# so the first line of code executed by the program is the first line of the `main` function
main:
        ## printf("%s", "I love MIPS\n")
        la      $a0, string                 ## pass address of string "I love MIPS\n" as first (and only) argument
        li      $v0, SYSCALL_PRINT_STRING   ## magic number for the print string syscall ( printf("%s") )
        syscall

        # the `LA` instruction is used to load an *address*
        # this means that the operand is a label, or a memory location (pointer)

        # the `LI` instruction is used to load an *immediate*
        # an immediate is a fancy word for a constant value, like a number (6, 0, -42, etc.) or a character ('a', ' ', '\n', etc.)
        # immediates can also be binary numbers (0b1010, 0b1111, etc.), octal numbers (0o123, 0o777, etc.), and hexadecimal numbers (0x123, 0xABC, etc.)
        # immediates can also be constants defined earlier in the program (e.g. `SYSCALL_PRINT_STRING` as this is equivalent to the number 4)

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

        # the `JR` instruction the `return` part
        # The value of the register `$v0` is the return value of the function
```

add.c

add 17 and 25 then print the result

```
#include <stdio.h>

int main(void) {
    int x = 17;
    int y = 25;

    printf("%d\n", x + y);

    return 0;
}
```

add.simple.c

add 17 and 25 then print the result

```
#include <stdio.h>

int main(void) {
    int x, y, z;

    x = 17;
    y = 25;

    z = x + y;

    printf("%d", z);
    printf("%c", '\n');

    return 0;
}
```

add.s

add 17 and 25 then print the result

```
main:
        # x in $t0
        # y in $t1
        # z in $t2

        li      $t0, 17         # x = 17;
        li      $t1, 25         # y = 25;

        add     $t2, $t1, $t0 # z = x + y

        move    $a0, $t2  # printf("%d", z);
        li      $v0, 1
        syscall

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

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

add.1.s

add 17 and 25 then print the result

On: 2023-02-12

Constant for the magic number of the print integer syscall

```
SYSCALL_PRINT_INT  = 1
# Constant for the magic number of the print character syscall
SYSCALL_PRINT_CHAR = 11

# Constant for the return value of the main function
EXIT_SUCCESS       = 0

.text
.globl main
main:
        # x in $t0
        # y in $t1
        # z in $t2

        li      $t0, 17                   # x = 17
        li      $t1, 25                   # y = 25

        add     $t2, $t1, $t0             # z = x + y

        # printf("%d", z);
        move    $a0, $t2                  # move the result of the addition into the first argument to the syscall, if we are copying the contents of one register to another register we use MOVE
        li      $v0, SYSCALL_PRINT_INT    # magic number for the print int syscall ( printf("%d") )
        syscall

        # printf("%c", '\n');
        li      $a0, '\n'                 # load the newline character as the first argument to the syscall, as a character is an immediate value we use LI
        li      $v0, SYSCALL_PRINT_CHAR   # magic number for the print char syscall ( printf("%c") )
        syscall

        # return 0
        li      $v0, EXIT_SUCCESS
        jr      $ra
```

hello_world.s

Print hello world in MIPS.

```
        .text
main:

        li      $v0, 4                  # syscall 4: print_string
        la      $a0, hello_world_msg    #
        syscall                         # printf("Hello world\n");


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

        .data

hello_world_msg:
        .asciiz "Hello world\n"
```

math.c

Perform some basic arithmetic.

```
#include <stdio.h>

int main(void) {
    int x = 17;
    int y = 25;

    printf("%d\n", 2 * (x + y));

    return 0;
}
```

math.simple.c

Perform some basic arithmetic.

```
#include <stdio.h>

int main(void) {
    int x = 17;
    int y = 25;

    int z = x + y;
    z = 2 * z;

    printf("%d", z);
    putchar('\n');

    return 0;
}
```

math.s

Do some basic arithmetic in MIPS.

```
main:
        # Locals:
        # - $t0: int x
        # - $t1: int y
        # - $t2: int z

        li      $t0, 17         # int x = 17;
        li      $t1, 25         # int y = 25;

        add     $t2, $t0, $t1   # int z = x + y;
        mul     $t2, $t2, 2     # z = z * 2;

        li      $v0, 1          # syscall 1: print_int
        move    $a0, $t2        #
        syscall                 # printf("%d", z);

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

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

math.fewer_registers.s

Do some basic arithmetic in MIPS, but with one less register.

```
main:
        # Locals:
        # - $t0: int x
        # - $t1: int y

        li      $t0, 17         # int x = 17;
        li      $t1, 25         # int y = 25;

        li      $v0, 1          # syscall 1: print_int
        add     $a0, $t0, $t1   # (x + y)
        mul     $a0, $a0, 2     # * 2
        syscall                 # printf("%d", 2 * (x + y));

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

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

square_and_add.c

Square two numbers and sum their squares.

```c
#include <stdio.h>

int main(void) {
    int a, b;

    printf("Enter a number: ");
    scanf("%d", &a);

    printf("Enter another number: ");
    scanf("%d", &b);

    printf("The sum of the squares of %d and %d is %d\n", a, b, a*a + b*b);

    return 0;
}
```

square_and_add.simple.c

Square two numbers and sum their squares.

```c
#include <stdio.h>

int main(void) {
    int a, b;

    printf("Enter a number: ");
    scanf("%d", &a);

    printf("Enter another number: ");
    scanf("%d", &b);


    printf("The sum of the squares of ");
    printf("%d", a);
    printf(" and ");
    printf("%d", b);
    printf(" is ");

    a = a * a;
    b = b * b;
    printf("%d", a + b);
    putchar('\n');

    return 0;
}
```

square_and_add.s

Square and add two numbers and print the result.

```
        .text
main:
        # Locals:
        # - $t0: int a
        # - $t1: int b

        li      $v0, 4                  # syscall 4: print_string
        la      $a0, prompt1_msg        #
        syscall                         # printf("Enter a number: ");

        li      $v0, 5                  # syscall 5: read_int
        syscall                         #
        move    $t0, $v0                # scanf("%d", &a);

        li      $v0, 4                  # syscall 4: print_string
        la      $a0, prompt2_msg        #
        syscall                         # printf("Enter another number: ");

        li      $v0, 5                  # syscall 5: read_int
        syscall                         #
        move    $t1, $v0                # scanf("%d", &b);


        li      $v0, 4                  # syscall 4: print_string
        la      $a0, result_msg_1       #
        syscall                         # printf("The sum of the squares of ");

        li      $v0, 1                  # syscall 1: print_int
        move    $a0, $t0                #
        syscall                         # printf("%d", a);

        li      $v0, 4                  # syscall 4: print_string
        la      $a0, result_msg_2       #
        syscall                         # printf(" and ");

        li      $v0, 1                  # syscall 1: print_int
        move    $a0, $t1                #
        syscall                         # printf("%d", b);

        li      $v0, 4                  # syscall 4: print_string
        la      $a0, result_msg_3       #
        syscall                         # printf(" is ");

        mul     $t0, $t0, $t0           # a = a * a;
        mul     $t1, $t1, $t1           # b = b * b;

        li      $v0, 1                  # syscall 1: print_int
        add     $a0, $t0, $t1           #
        syscall                         # printf("%d", a + b);

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


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

        .data
prompt1_msg:
        .asciiz "Enter a number: "
prompt2_msg:
        .asciiz "Enter another number: "
result_msg_1:
        .asciiz "The sum of the squares of "
result_msg_2:
        .asciiz " and "
result_msg_3:
        .asciiz " is "
```

COMP1521 24T2: Computer Systems Fundamentals is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs1521@cse.unsw.edu.au
CRICOS Provider 00098G