

Proyecto 01 - Organización de Computadores

Juan Antonio González Orbe, Alexander Goussas

3 de Julio, 2022

Contents

1	Instrucciones	2
1.1	Entregables	2
2	Autores y Extras	3
3	Implementación	3
3.1	Funciones	3
3.2	Compilación y Ejecución del Programa	4
3.3	Archivo <code>make.in</code> y Desarrollo del Proyecto	6
3.4	Tests y GitHub Workflows	9
3.5	Video demostrativo	10
4	Código fuente	11
4.1	Versión <code>.asm</code>	11
4.2	Versión <code>.c</code>	29
5	Referencias	35

1 Instrucciones

Este proyecto consiste en implementar un programa en lenguaje ensamblador que simule el funcionamiento de una cabina telefónica.

- El usuario puede ingresar monedas de diferentes denominaciones desde 5 centavos, y puede ingresar tantas monedas como desee.
- El usuario ingresa el número al que desea llamar. (Validar)
- El costo de la llamada por minuto será generado de manera aleatoria entre un valor de 10 y 40 centavos de dólar.
- Luego se deberá simular la llamada, y el usuario podrá colgar, o la llamada puede terminar debido a que se le terminó el saldo.
- La cabina deberá mostrar una alerta cuando el saldo sea menor a \$0.05
- La cabina dará vuelto en caso de ser necesario. Recuerde que la mínima denominación son monedas de 5 centavos.

1.1 Entregables

- Código en lenguaje ensamblador.
- Sus códigos deben estar apropiadamente documentados. (En un sólo idioma)
- Documento en PDF que contenga:
 - Consideraciones sobre el uso de su programa
 - Capturas de Pantalla de su programa funcionando
 - Referencias
- **EXTRAS:**
 - Implementación del programa en C

2 Autores y Extras

Este proyecto es realizado por:

Nombre	Usuario GH	E-Mail (ESPOL)
<i>Juan Antonio González Orbe</i>	anntnzb	juangonz@espol.edu.ec
<i>Alexander Goussas</i>	aloussase	agoussas@espol.edu.ec

El código fuente puede ser encontrado en **GitHub**, en el siguiente repositorio:

[<https://github.com/aloussase/CCPG1049-2022-P1>](https://github.com/aloussase/CCPG1049-2022-P1)

3 Implementación

Este documento adjunta el código fuente de las dos versiones implementadas del programa (.c y .asm), sin embargo, única y exclusivamente será documentada la versión solicitada, que es la versión en **MIPS**. En la sección "Código fuente" estará incluido el contenido adjunto, por la parte final del documento, esto para no ser intrusivo al lector. El código fuente se encuentra *bien documentado* por lo que leer e interpretar el programa debe fácil.

El número telefónico solicitado por el programa no sirve algún propósito, por lo que se ha optado por agregar una funcionalidad extra, que es verificar que la cantidad de dígitos sea igual a 10. El número indicado anteriormente es la cantidad de dígitos que contempla un número telefónico celular Ecuatoriano válido, ésto es, sin extensión internacional (+593) y con los 2 primeros dígitos iguales a 09.

A continuación se encuentra una explicación de funciones empleadas, creación y ejecución del programa.

3.1 Funciones

Las funciones del proyecto se encuentran de forma individual, cada una en un archivo independiente. Éstas pueden ser encontradas en la carpeta `src/`.

```
is_valid_coin.asm  
is_valid_phone_number.asm  
pow.asm  
rand.asm  
readline.asm  
strlen.asm
```

Por cada archivo (función) se encontrará una pequeña documentación de la función y que registros se han empleado.

3.2 Compilación y Ejecución del Programa

SPIM es un simulador de **MIPS**, éste es capaz de correr código ensamblador dirigido a **MIPS**, así como **MIPS Assembler and Runtime Simulator (MARS)**. La diferencia mas evidente entre estos dos simuladores es que **SPIM** permite correr comandos desde la consola, mientras **MARS** en sí es un **IDE (Integrated Development Environment)**.

Para este proyecto se ha optado por emplear **SPIM** ya que facilita la automatización de comandos a través de la consola, por ejemplo, es fácil integrar **SPIM** con algún *target* de **Makefile**.

Mediante **Makefile** podemos crear *targets* dirigidos a **SPIM** que nos permite simplemente correr:

```
make asm
```

Y como resultado ver la ejecución del programa en la misma terminal.

A partir de lo anterior, es importante mencionar que el código escrito está diseñado para poder ser ejecutado en **SPIM** y **MARS** intercambiabilmente. Se ha evitado el empleo de instrucciones o directivas específicas de alguna plataforma para así fomentar la portabilidad.

En **MARS** es posible definir *macros* para así encapsular código y evitar repeticiones, similar al concepto de funciones, pero aquí la sustitución es textual, no existe algún tipo de transformación.

A continuación un ejemplo de un macro para encapsular el conjunto de instrucciones para imprimir un número entero:

```
.macro print_int (%n)
li $v0, 1
add $a0, $zero, %n
syscall
.end_macro

print_int (10) # ==> 10
```

Desafortunadamente, los *macros* son una característica exclusiva de **MARS**, por lo que incluirla en el proyecto violaría el principio de portabilidad propuesto por los autores del proyecto. En otras palabras, los *macros* realmente no son parte del lenguaje **MIPS**, éstos son una extensión al lenguaje.

Hasta el momento de éste escrito, se contempla que la persona designada a revisar y calificar este proyecto empleará **MARS**, por lo que, técnicamente, es posible usar estos mencionados *macros*. Sin embargo, para evitar complicaciones se ha decidido no emplear dicha característica.

3.3 Archivo `make.in` y Desarrollo del Proyecto

Este archivo contiene el *pseudo-código* del programa en lenguaje **MIPS**, que posteriormente será transformado a su version final (implementación real del programa) con nombre de archivo `main.asm`.

La decisión de trabajar de esta manera permite incluir directivas creadas por los autores como `@include <archivo.asm>` sin tener que tener todo declarado en el mismo archivo. A partir de esto, cada función creada para el programa se encuentra independientemente en la carpeta `src/`, como se menciona en la sección Funciones.

Sea el ejemplo, en `src/` se puede encontrar los siguientes archivos:

- `strlen.asm`
- `check_number_valid.asm`

Mediante esta estructurada, se permite separar el programa solicitado con las funciones que éste requiere, esto implica que si hay un error en la función `strlen`, no es necesario nadar buscando en alguna parte del archivo, sino ir a su respectivo archivo directamente.

La transformación de *pseudo-código* a código válido en **MIPS** es posible gracias a un simple y pequeño script ubicado en la carpeta `scripts/` escrito en POSIX Shell, éste busca y reemplaza las directivas creadas para así generar el archivo **MIPS** real. Este proceso es único y exclusivo para los desarrolladores del proyecto, el archivo final es llamado `main.asm` y es el entregable del proyecto en sí.

```

[annt@munich ~/lib/repos/CCPG1049-2022-P1]$ make asm
rm -Rf main.asm
./scripts/preprocess.sh main.in main.asm
INFO: Preprocessing main.in
INFO: Substituting contents of src/strlen.asm
INFO: Substituting contents of src/readline.asm
INFO: Substituting contents of src/is_valid_coin.asm
INFO: Substituting contents of src/is_valid_phone_number.asm
INFO: Substituting contents of src/pow.asm
INFO: Substituting contents of src/rand.asm
spim -file main.asm
Loaded: /nix/store/k95m806mszq32pgbvqfr9sc4akfc5qwm-xspim-9.1.22
Ingreso monedas (-1 para terminar): 0.5
Ingreso monedas (-1 para terminar): 0.5
Ingreso monedas (-1 para terminar): 0.2
Moneda incorrecta
Ingreso monedas (-1 para terminar): 0.25
Ingreso monedas (-1 para terminar): -1
Su saldo es: 1.25000000
El valor por minuto de llamada es de 30 ctvs
Ingrese el numero a llamar: 0993446587
Iniciar la llamada?[S/n] S
1. Llamada en curso ... Presiona C para colgar

2. Llamada en curso ... Presiona C para colgar

3. Llamada en curso ... Presiona C para colgar
c
Duracion de la llamada (minutos): 3
Costo total de la llamada: 0.90000004
Cambio: 0.34999996

```

Figure 1: Ejemplo de ejecución del programa en ASM (MIPS) #1

```
[annt@munich ~/lib/repos/CCPG1049-2022-P1]$ make c && ./main
gcc -o main main.o -lreadline
Ingrese monedas: 0.5
Ingrese monedas: 0.5
Ingrese monedas: 0.5
Ingrese monedas: 0.25
Ingrese monedas: 0.05
Ingrese monedas: -1
Su saldo es 1.80

Costo de la llamada: $ 0.15
Ingrese el numero a llamar: 1234567890
Iniciar la llamada? [Si/No] Si
1. Llamada en curso ... Presiona C para colgar
2. Llamada en curso ... Presiona C para colgar
3. Llamada en curso ... Presiona C para colgar
4. Llamada en curso ... Presiona C para colgar
C
Costo de la llamada: $ 0.60
Cambio: $ 1.20
[annt@munich ~/lib/repos/CCPG1049-2022-P1]$
```

Figure 2: Ejemplo de ejecución del programa en C #1

3.4 Tests y GitHub Workflows

En el repositorio de GitHub se puede encontrar un **workflow** que permite correr una serie de tests pertinentes a la carpeta `tests/`. Éstos fueron creados para depurar posibles errores en las implementaciones individuales de las funciones (esto gracias al hecho de haberlas separado por archivo independientes); la creación de los mismos facilitó el desarrollo de las funciones ya que el chequeo era aislado del programa.

Los tests se corren cada vez que algun contribuidor ejerce un *push* al repositorio remoto.

La siguiente imagen adjunta muestra una serie de tests exitosos:

20 workflow runs

Event ▾

Status ▾

Branch ▾

Actor ▾

✓

Makefile: minor tweaks

run_tests #20: Commit f6d35ca pushed by anntnzb

master

📅 yesterday

🕒 37s

...

✓

Fix "li.s" instruction issue with MARS

run_tests #19: Commit 05c6a01 pushed by anntnzb

mars-lis-instruction

📅 yesterday

🕒 23s

...

✓

Makefile: minor tweaks

run_tests #18: Commit 7b552cb pushed by anntnzb

master

📅 yesterday

🕒 26s

...

✓

add transpiled asm file

run_tests #17: Commit 2e98f3c pushed by anntnzb

master

📅 yesterday

🕒 22s

...

✓

kill call loop with either 'c' or 'C'

run_tests #16: Commit 17eaac8 pushed by anntnzb

master

📅 3 days ago

🕒 22s

...

✓

TODO: update

run_tests #15: Commit 9a01de6 pushed by anntnzb

call-kill-c

📅 3 days ago

🕒 22s

...

✓

TODO: update

run_tests #14: Commit 9a01de6 pushed by anntnzb

master

📅 3 days ago

🕒 24s

...

✓

implement ask_for_balance and finish main

run_tests #13: Commit 8092f17 pushed by aloussase

master

📅 4 days ago

🕒 23s

...

✓

add linear congruency PRNG

run_tests #12: Commit d090ddf pushed by aloussase

master

📅 4 days ago

🕒 41s

...

✓

replace li.s with l.s for mars compatibility

run_tests #11: Commit 0eb6b26 pushed by aloussase

master

📅 4 days ago

🕒 34s

...

✓

update todo

run_tests #10: Commit 7d2607c pushed by aloussase

master

📅 4 days ago

🕒 41s

...

✓

Finish is_valid_coin + tests

run_tests #9: Commit 9fe82bd pushed by aloussase

master

📅 4 days ago

🕒 25s

...

Figure 3: GitHub Workflows

3.5 Video demostrativo

Se proporciona un video demostrativo (demo) de la ejecución del programa en MIPS y en C en el siguiente link.

4 Código fuente

4.1 Versión .asm

```
1      .data
2
3  cost_per_minute_prefix:      .ascii "El valor por minuto de llamada es de: "
4  cost_per_minute_suffix:      .ascii " ctvs\n"
5
6  ask_for_phone_number_prompt:  .ascii "Ingrese el numero a llamar: "
7  ask_for_phone_number_errmsg:  .ascii "\033[31mERROR:\033[m Numero invalido\n"
8
9  simulate_call_prompt:         .ascii "Iniciar la llamada? [S/n] "
10 simulate_call_message:        .ascii ". Llamada en curso ... Presiona C para col
11
12 ask_for_balance_prompt:        .ascii "Ingrese monedas (-1 para terminar): "
13 ask_for_balance_errmsg:        .ascii "Moneda incorrecta\n"
14
15 balance_report_message:        .ascii "Su saldo es: "
16 call_duration_message:         .ascii "Duracion de la llamada (minutos): "
17
18 total_call_cost_message:       .ascii "Costo total de la llamada: "
19
20 change_message:                .ascii "Cambio: "
21
22 nickel:                       .float  0.05
23 dime:                         .float  0.10
24 quarter:                     .float  0.25
```

```

25  half:                                .float  0.50
26  tolerance:                          .float  0.000001
27
28  minus_one:                          .float  -1.0
29  zerof:                              .float  0.0
30  hundredf:                          .float  100.0
31
32  ask_for_phone_number_buffer:         .byte  12
33  simulate_call_buffer:               .byte  3
34
35          .text
36          .globl main
37
38          # STRLEN
39          #
40          # Input:
41          #   $a0: The string for which to calculate the length.
42          # Output:
43          #   $v0: The length of the given string.
44  strlen:
45          li    $v0, 0
46
47  strlen_loop:
48          lb     $t0, 0($a0)
49          beqz   $t0, strlen_exit
50
51          addi   $v0, $v0, 1

```

```

52      addi      $a0, $a0, 1
53      j         strlen_loop
54
55  strlen_exit:
56      jr        $ra
57      # READLINE
58      #
59      # Input:
60      #  $a0: A null terminated string to use as prompt
61      #  $a1: The address of a buffer to store the line
62      #  $a2: The max. number of characters to read
63  readline:
64      li        $v0, 4
65      syscall
66
67      # syscall 8 will read max $a1 chars into $a0
68      move $a0, $a1
69      move $a1, $a2
70      li        $v0, 8
71      syscall
72
73      jr        $ra
74  # IS_VALID_COIN
75  #
76  # Input:
77  #  $f0: A floating point number to test for coinness
78  # Output:

```

```

79 # $v0: 1 if $a0 is a valid coin, 0 otherwise
80 is_valid_coin:
81     # NOTE: These need to be defined in the including file.
82     l.s $f1, nickel
83     l.s $f2, dime
84     l.s $f3, quarter
85     l.s $f4, half
86
87     l.s $f9, tolerance          # accepted error margin
88
89     # Here I am using a different register for each branch to
90     # avoid having to reset the same register over and over
91     # again.
92
93     sub.s $f5, $f1, $f0          # if ((0.05 - arg) < 0.000001)
94     abs.s $f5, $f5
95     c.lt.s $f5, $f9              #
96     bc1t  is_valid_coin_success  # return true;
97
98     sub.s $f6, $f2, $f0          # if ((0.1 - arg) < 0.000001)
99     abs.s $f6, $f6
100    c.lt.s $f6, $f9              #
101    bc1t  is_valid_coin_success  # return true;
102
103    sub.s $f7, $f3, $f0          # if ((0.25 - arg) < 0.000001)
104    abs.s $f7, $f7
105    c.lt.s $f7, $f9              #

```

```

106         bc1t    is_valid_coin_success    #   return true;
107
108         sub.s    $f8, $f4, $f0            # if ((0.5 - arg) < 0.000001)
109         abs.s    $f8, $f8
110         c.lt.s   $f8, $f9                #
111         bc1t    is_valid_coin_success    #   return true;
112
113         li $v0, 0                        # return false
114         jr $ra
115
116 is_valid_coin_success:
117         li $v0, 1
118         jr $ra
119         # IS_VALID_PHONE_NUMBER: checks whether the provided number (as a string)
120         #                               is a valid phone number
121         # Input:
122         #   $a0: The string to validate.
123         # Output:
124         #   $v0: 1 if the string is a valid phone number, 0 otherwise.
125 is_valid_phone_number:
126         addi $sp, $sp, -8
127         sw    $ra, 0($sp)                # Save return address in stack
128         sw    $s0, 4($sp)
129
130         move $s0, $a0                    # Save $a0
131         jal   strlen                      # Calculate the length of the
132         move $t1, $v0                    # Save return value of strlen

```

```

133         move $a0, $s0                                # Restore $a0
134
135         li    $v0, 0
136         li    $t0, 11
137         bne   $t0, $t1, is_valid_phone_number_exit    # Return 0 if len($a0) != 11
138
139 is_valid_phone_number_loop:
140         lb    $t0, 0($a0)                              # $t0 = *$a0;
141
142         li    $v0, 1
143         beqz  $t0, is_valid_phone_number_exit          # Return true if we reach the e
144         beq   $t0, 10, is_valid_phone_number_exit      # Or is it's a newline
145
146         li    $v0, 0                                    # Prepare false return value in
147
148         li    $t1, 48                                  # 48 is ascii code for 0
149         blt   $t0, $t1, is_valid_phone_number_exit    # *$a0 < '0'
150
151         li    $t1, 57                                  # 57 is 9
152         bgt   $t0, $t1, is_valid_phone_number_exit    # *$a0 > '9'
153
154         addi  $a0, $a0, 1                              # Increment the string pointer
155         j     is_valid_phone_number_loop
156
157 is_valid_phone_number_exit:
158         lw    $ra, 0($sp)
159         lw    $s0, 4($sp)

```



```

160          addi $sp, $sp, 8
161
162          jr   $ra
163 # POW: raise a number to the nth power
164 #
165 # input:
166 #   $a0: the base
167 #   $a1: the exponent
168 # output:
169 #   $v0: $a0 raised to the $a1
170 pow:
171     li    $t0, 1          # number of iterations
172     move  $v0, $a0        # start with the initial value of base
173
174 pow_loop:
175     beq   $t0, $a1, pow_exit
176
177     mult  $v0, $a0
178     mflo  $v0
179
180     addi  $t0, $t0, 1
181
182     j     pow_loop
183
184 pow_exit:
185     jr    $ra
186 # RAND: get a random number

```

```

187 #
188 # input:
189 #   - $a0: previous pseudorandom number returned by this routine or a seed value
190 # output:
191 #   - $v0: a 31 bit random number
192 # requires:
193 #   - pow
194 rand:
195     addi $sp, $sp, -8
196     sw   $ra, 0($sp)
197     sw   $a0, 4($sp)
198
199     li   $a0, 2                # m = 2^31
200     li   $a1, 31
201     jal  pow
202
203     lw   $a0, 4($sp)          # restore $a0
204
205     li   $t0, 1103515245      # a
206     li   $t1, 12345           # c
207
208     #  $X_n = (aX + c) \bmod m$ 
209
210     mult $t0, $a0              # aX
211     mflo $t0
212
213     add  $t0, $t0, $t1         # aX + c

```

```

214
215     div  $t0, $v0                # (aX + c) mod m
216     mfhi $t0
217
218     lw   $ra, 0($sp)
219     addi $sp, $sp, 8
220
221     move $v0, $t0
222     jr   $ra
223
224
225 ask_for_balance:
226     addi $sp, $sp, -4
227     sw   $ra, 0($sp)
228
229     l.s  $f15, minus_one         # load -1.0 into $f15 to check exit cond
230     l.s  $f16, zerof            # initialize return value to zero
231     l.s  $f0,  zerof            # reset $f0
232
233 ask_for_balance_loop:
234     add.s $f16, $f16, $f0        # add to the balance
235
236     la   $a0, ask_for_balance_prompt # print prompt
237     li   $v0, 4
238     syscall
239
240     li   $v0, 6                  # read a float, result in $f0

```

```

241      syscall
242
243      c.eq.s $f0, $f15                # exit if user entered -1
244      bc1t   ask_for_balance_exit
245
246      jal   is_valid_coin             # check if input is a valid coin denomi
247                                             # (argument is already in $f0)
248      bnez $v0, ask_for_balance_loop  # loop back if it is
249
250      la    $a0, ask_for_balance_errmsg # else print error message
251      li    $v0, 4
252      syscall
253
254      sub.s $f16, $f16, $f0          # subtract invalid coin because it will
255      j      ask_for_balance_loop    # and loop
256
257 ask_for_balance_exit:
258      mov.s $f0, $f16                # move return value to $f0
259      lw     $ra, 0($sp)
260      addi   $sp, $sp, 4
261      jr     $ra
262
263 ask_for_phone_number:
264      add $sp, $sp, -12
265      sw   $ra, 0($sp)
266      sw   $s0, 4($sp)
267      sw   $s1, 8($sp)

```

```

268
269     la  $s0, ask_for_phone_number_prompt
270     la  $s1, ask_for_phone_number_buffer
271
272     move $a0, $s0
273     move $a1, $s1
274     li   $a2, 12
275     jal readline
276
277 ask_for_phone_number_loop:
278     move $a0, $s1
279     jal  is_valid_phone_number           # Check whether the input is valid.
280
281     bne  $v0, $zero, ask_for_phone_number_exit # Exit if the number is valid.
282
283     la  $a0, ask_for_phone_number_errmsg    # Print an error message.
284     li  $v0, 4
285     syscall
286
287     move $a0, $s0                         # Ask for input again.
288     move $a1, $s1
289     li   $a2, 12
290     jal readline
291
292     j    ask_for_phone_number_loop         # Loop.
293
294 ask_for_phone_number_exit:

```

```

295         lw  $ra, 0($sp)
296         lw  $s0, 4($sp)
297         lw  $s1, 8($sp)
298         add $sp, $sp, 12
299         jr  $ra
300
301 simulate_call:
302         addi $sp, $sp, -16
303         sw   $ra, 0($sp)
304         sw   $s0, 4($sp)
305         sw   $s1, 8($sp)
306         sw   $s2, 12($sp)
307
308         li $s0, 0                # duration of the call in m
309         la $s1, simulate_call_buffer # store user answer.
310         la $s2, simulate_call_message
311
312         la  $a0, simulate_call_prompt # ask the user if they want
313         move $a1, $s1
314         li  $a2, 3
315         jal readline
316
317         lb  $t0, 0($s1)          # exit is user entered 'S'.
318         bne $t0, 83, simulate_call_exit # 83 is ascii code for 'S'.
319
320 simulate_call_loop:
321         addi $s0, $s0, 1        # increase the number of mi

```

```

322
323     li    $v0, 1                                # print call in progress me
324     move $a0, $s0
325     syscall
326
327     li    $v0, 4
328     move $a0, $s2
329     syscall
330
331     li $v0, 12                                # read a character
332     syscall
333
334     li    $t0, 67
335     li    $t1, 99
336     beq $v0, $t0, simulate_call_exit          # exit if the user enters e
337     beq $v0, $t1, simulate_call_exit
338     j simulate_call_loop
339
340 simulate_call_exit:
341     move $v0, $s0                                # return call duration in m
342     lw    $ra, 0($sp)
343     lw    $s0, 4($sp)
344     lw    $s1, 8($sp)
345     lw    $s2, 12($sp)
346     addi $sp, $sp, 16
347     jr    $ra
348

```

```

349  main:
350      add $sp, $sp, -4
351      sw  $ra, 0($sp)
352
353      #
354      # Ask the user for balance and print it.
355      #
356
357      jal    ask_for_balance
358      la     $a0, balance_report_message
359
360      mov.s  $f20, $f0                # Save the balance in $f20.
361
362      li     $v0, 4
363      syscall
364
365      mov.s  $f12, $f0
366      li     $v0, 2
367      syscall
368
369      li     $a0, 10                  # Print a newline.
370      li     $v0, 11
371      syscall
372
373      #
374      # Get a random number to be the per minute cost of the phone call.
375      #

```



```

376      # The seed is fixed so we always get the same random value on
377      # every execution of the program. We need a source of entropy.
378      #
379
380      li  $a0, 1
381      jal rand
382
383      li  $t0, 40                                # Normalize the result to be 0 <= x <=
384      div $v0, $t0
385      mfhi $s0
386
387      la  $a0, cost_per_minute_prefix
388      li  $v0, 4
389      syscall
390
391      move $a0, $s0                                # Print the random number.
392      li  $v0, 1
393      syscall
394
395      la  $a0, cost_per_minute_suffix
396      li  $v0, 4
397      syscall
398
399      #
400      # Ask for phone number and simulate call.
401      #
402

```

```

403     jal    ask_for_phone_number    # Ask for a phone number.
404     jal    simulate_call           # Simulate call, number of minutes is in $v0
405     move   $t0, $v0                # Save return value in $t0.
406
407     la     $a0, call_duration_message
408     li     $v0, 4
409     syscall
410
411     move   $a0, $t0                # Print number of minutes.
412     li     $v0, 1
413     syscall
414
415     li     $a0, 10                 # Print a newline.
416     li     $v0, 11
417     syscall
418
419     #
420     # Calculate and print the final cost of the phone call.
421     #
422
423     l.s     $f1, hundredf
424
425     addi    $sp, $sp, -8           # Needed for converting ints to floats.
426     sw      $s0, 0($sp)
427     sw      $t0, 4($sp)
428
429     lwc1    $f0, 0($sp)           # Price per minute.

```

```

430      cvt.s.w  $f2, $f0
431
432      lwc1      $f0, 4($sp)          # Call duration.
433      cvt.s.w  $f3, $f0
434
435      addi      $sp, $sp, 8          # Pop 2 items off the stack.
436
437      div.s     $f12, $f2, $f1      # f12 = price_per_minute / 100
438      mul.s     $f12, $f12, $f3     # f12 = f12 * call_duration
439
440      la        $a0, total_call_cost_message
441      li        $v0, 4
442      syscall
443
444      li        $v0, 2              # Print the total cost of the call.
445      syscall
446
447      li        $a0, 10             # Print a newline.
448      li        $v0, 11
449      syscall
450
451      #
452      # Calculate and print the change.
453      #
454
455      la        $a0, change_message
456      li        $v0, 4

```

```
457      syscall
458
459      sub.s $f12, $f20, $f12
460      li    $v0, 2
461      syscall
462
463      li    $a0, 10
464      li    $v0, 11
465      syscall
466
467      #
468      # End
469      #
470
471      lw    $ra, 0($sp)
472      add $sp, $sp, 4
473
474      jr $ra
```

4.2 Versión .c

```
1  #include <assert.h>
2  #include <stdio.h>
3  #include <stdint.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <time.h>
7  #include <readline/readline.h>
8
9  #define EQLFLT(x, y) (((x) - (y)) < 0.00001)
10
11  int
12  is_valid_coin(double coin)
13  {
14      return EQLFLT(coin, 0.05) | EQLFLT(coin, 0.10) | EQLFLT(coin, 0.25) | EQLFLT(coin, 0.50);
15  }
16
17  float
18  ask_for_balance()
19  {
20      double coin;
21      double saldo = 0;
22
23      char* line;
24      while ((line = readline("Ingrese monedas: ")) != NULL)
25      {
26          coin = atof(line);
```

```

27
28     if (coin == -1.0)
29     {
30         free(line);
31         break;
32     }
33
34     if (!is_valid_coin(coin))
35     {
36         fprintf(stderr, "Moneda incorrecta\n");
37     }
38     else
39     {
40         saldo += coin;
41     }
42
43     free(line);
44 }
45
46 return saldo;
47 }
48
49 int
50 is_valid_phone_number(char* number)
51 {
52     if (strlen(number) != 10)
53         return 0;

```

```

54
55     while (*number)
56     {
57         /*
58             !(A && B) == !A || !B
59             !(A >= B) == A < B
60         */
61         if (!(*number >= '0' && *number <= '9'))
62             return 0;
63         number++;
64     }
65
66     return 1;
67 }
68
69 void
70 ask_for_phone_number()
71 {
72     char* line;
73
74     while ((line = readline("Ingrese el numero a llamar: ")) != NULL)
75     {
76         if (is_valid_phone_number(line))
77         {
78             free(line);
79             return;
80         }

```

```

81
82     fprintf(stderr, "Numero incorrecto\n");
83     free(line);
84 }
85 }
86
87 int
88 simulate_call()
89 {
90     int minutes = 0;
91     char* ans = readline("Iniciar la llamada? [Si/No] ");
92
93     if (strcmp(ans, "Si") != 0)
94         return minutes;
95
96     free(ans);
97
98     while (1)
99     {
100         minutes += 1;
101         printf("%d. Llamada en curso ... Presiona C para colgar\n", minutes);
102
103         int c = getchar();
104         if (c == 'c' || c == 'C')
105             break;
106     }
107

```



```

108     return minutes;
109 }
110
111 int
112 run()
113 {
114
115     float saldo = ask_for_balance();
116     printf("Su saldo es %.2f\n", saldo);
117     printf("\n");
118
119     int price_per_minute = (rand() + 10) % 40;
120     printf("Costo de la llamada: $ 0.%d\n", price_per_minute);
121
122     ask_for_phone_number();
123     int minutes = simulate_call();
124
125     float final_price = (price_per_minute / 100.0) * minutes;
126     printf("Costo de la llamada: $ %.2f\n", final_price);
127     // TODO: Duracion de la llamada
128
129     printf("Cambio: $ %.2f\n", saldo - final_price);
130
131     return 0;
132 }
133
134 int

```

```
135  main()
136  {
137      srand(time(NULL));
138      return run();
139  }
```

5 Referencias

- Computer Architecture with MIPS
- SPIM: A MIPS32 Simulator - SourceForge
- MARS (MIPS Assembler and Runtime Simulator)