PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2343 - Arquitectura de Computadores
Septiembre 2021

Tarea 2

Fecha de entrega: 9 de noviembre de 2021, hasta las 23:59.

Objetivos

Para esta tarea, esperamos que profundicen su conocimiento del lenguaje de programación en assembly RISC-V[1] (*risk-five*), aplicando los conocimientos adquiridos en clases hasta el momento, en particular respecto al uso de números de punto flotante, I/O e interrupciones.

Parte I: DEBT - V

Se avecina un viaje de una semana con tus amigues y estás cansade de tener que utilizar la calculadora para saldar deudas con elles. Debido a esto, decides anticiparte y crear un programa llamado DEBT - V que te permita ingresar los gastos que han hecho, y que al final del viaje haga un ajuste de cuentas para determinar quién le debe pagar a quién y cuánto dinero.

Funcionamiento

Deberás realizar un programa que reciba un arreglo con la información de los gastos realizados en el viaje, con el siguiente formato: (*quien paga*, *monto*, *cuantas personas participan*, *quienes participan*). Tú tendrás que idear cómo hacer los cálculos necesarios para saber quiénes deben pagar y quiénes tienen saldo a favor, entregando un output final de la forma 'X debe a Y W pesos'.

Para que tu programa esté correcto es importante que el ajuste de cuentas se haga una vez que TERMINA el viaje, es decir, NO se debe entregar un output 'X debe a Y W pesos' por cada gasto realizado, sino un ajuste global. Por ejemplo, si al final del viaje Pedro le debe 2000 pesos a Ana y Ana le debe 2000 a Juan, lo correcto es que el output sea '*Pedro* debe a *Juan* 2000 pesos' y no '*Pedro* debe a *Ana* 2000 pesos y *Ana* debe a *Juan* 2000 pesos'. Para lograr esto, te serán entregados como input (además del arreglo de gastos) el número total de gastos realizados, la cantidad de amigues que fueron al viaje y los ID de cada une.

[1] `https://riscv.org/`

Variables

- `N`: cantidad del amigues del grupo que fue al viaje.
- `amigos`: arreglo de IDs de cada amigue, de la forma:

  `ID_amigo_1, ID_amiga_2, ... , ID_amigo_N`

- `G`: cantidad de gastos realizados en el viaje
- `gastos`: arreglo de gastos realizados en el viaje, de la forma:

  `ID_quien_paga, monto, n_participantes, ID_participante_1, ... , ID_participante_n`

  * El `ID` de cada amigue equivale a su posición en el arreglo de IDs.
  * Si `n_participantes` es igual a -1 significa que todo el grupo de amigues participó en el gasto. Es decir, si se encuentra un -1, a continuación vendrá el siguiente gasto y no los `IDs` de los participantes, pues no es necesario especificarlos.

Input

Cada programa deberá tener la siguiente estructura

```
.globl  start
.data
    # --- VIAJE ---
    # --- No modificar labels ---
    N: .word 3 # cantidad de amigues en el viaje
    amigues: .word 0, 1, 2 # ID de cada amigue
    G: .word 3 # cantidad de gastos realizados al final del viaje
    gastos: .float 0.0, 3000.0, 2.0, 0.0, 1.0, 1.0, 5000.0, 1.0, 2.0, 1.0, 1000.0, -1.0
              # gastos realizados al final del viaje
    # --- End no modificar labels---
    # --- END VIAJE ---
    # de aca para abajo van sus variables en memoria
.text
    start:
        # aca va su codigo :3
```

Output

Utilizando environment calls (ecall), deberás imprimir en consola el output de manera similar al siguiente ejemplo:

```
1
debe a
2
2000.0
3
debe a
```

```
1
8000.0
```

4 puntos.

Parte II: ALARM

En esta sección, deberás escribir un pequeño script que implemente una cuenta regresiva, utilizando el Timer Tool del emulador. Al iniciar, el programa debe preguntarle al usuario el tiempo que desea para la cuenta regresiva, en segundos, para luego iniciar dicha cuenta regresiva. Una vez que termine la cuenta regresiva, el programa debe indicar que la cuenta regresiva ha terminado y preguntar por un nuevo valor para la cuenta regresiva. En caso de que se ingrese 0 o un valor negativo, el proceso tiene que salir, utilizando la ecall 10 para esto, no puede caerse del fondo del programa. La terminal del emulador se debería ver más o menos así:

```
Ingrese un tiempo:
10
Han pasado 10 segundos!
Ingrese un tiempo:
2
Han pasado 2 segundos!
Ingrese un tiempo:
0
-- program is finished running (0) --
```

2 puntos.

Evaluación

Assembly

El código deberá estar adecuadamente comentado, de manera de facilitar la corrección/*debugging*, además de respetar las convenciones de llamada para los registros, especificadas en la segunda página del *green card*[2] de RISC-V. Por ejemplo, para guardar una variable en el registro `x5`, deberán hacerlo a través de `t0`. En su código nunca debiesen llamar a un registro a través de la notación `xN`, además de usar los registros de acuerdo con la descripción provista.

Además podrían haber **descuentos de hasta 1 punto por código muy ilegible**, aunque mantengan en mente que el código se escribe para la máquina, no otras personas.

```
1  # para sumar dos numeros, escribir
2  addi a2, a2, 3
3  addi a3, a3, zero
4  add a4, a3, a4
5  # NO HACER
6  addi x12, x12, 3
7  addi x13, x13, x0
8  add x14, x13, x14
```

RISC-V Calling Convention			
Register	ABI Name	Saver	Description
x0	zero	---	Hard-wired zero
x1	ra	Caller	Return address
x2	sp	Callee	Stack pointer
x3	gp	---	Global pointer
x4	tp	---	Thread pointer
x5-7	t0-2	Caller	Temporaries
x8	s0/fp	Callee	Saved register/frame pointer
x9	s1	Callee	Saved register
x10-11	a0-1	Caller	Function arguments/return values
x12-17	a2-7	Caller	Function arguments
x18-27	s2-11	Callee	Saved registers
x28-31	t3-t6	Caller	Temporaries
f0-7	ft0-7	Caller	FP temporaries
f8-9	fs0-1	Callee	FP saved registers
f10-11	fa0-1	Caller	FP arguments/return values
f12-17	fa2-7	Caller	FP arguments
f18-27	fs2-11	Callee	FP saved registers
f28-31	ft8-11	Caller	FP temporaries

Figura 1: Convención de llamada para registros. Free & Open RISC-V Reference Card, RISC-V Organization.

NO CUMPLIR CON LA CONVENCIÓN DE LLAMADA SUPONDRÁ UN DESCUENTO DE 1 PUNTO EN LA NOTA FINAL

Emulador (IMPORTANTE)

Si bien son libres de programar en el editor que prefieran e incluso compilar/emular en la herramienta que les sea más cómoda, la corrección será con el emulador RARS anteriormente mencionado, por lo que su tarea deberá pasar el *assembler* y ejecutar en dicho emulador. **No pasar la etapa del *assembler* en RARS implica 0 puntos en el ítem particular que no pueda ser ejecutado**.

Se recomienda encarecidamente evitar el uso de tildes (‘, ’, ¨, etc.) u otros caracteres especiales en el código, ya que es probable que si hacen esto, el código se muestre con caracteres inválidos en el computador del ayudante por diferencias en cómo funciona el encoding/decoding de caracteres entre los diferentes sistemas operativos, y esto hará que su programa no pase el assembler.

Nota tarea

La nota se calculará de la siguiente manera:

$$Puntos + 1 = Nota\ tarea$$

La nota de la tarea se redondea a la decena.

Entrega

La entrega de la tarea será a través de la plataforma SIDING en los formularios habilitados para ello, hasta el día 9 de noviembre de 2021, a las 23:59. **No se entregará más plazo.**

El formato a entregar será un archivo `.txt` con su solución en assembly RISC-V de la parte 1. Además deberán entregar otro archivo `.txt` con el script RISC-V de su respuesta para el ítem 2.

[2] `https://inst.eecs.berkeley.edu/~cs61c/fa17/img/riscvcard.pdf`