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

IIC2343 - Arquitectura de Computadores
Marzo 2022

Entrega el 8 de abril hasta las 23:59

Tarea 1

Objetivos

Para esta tarea, esperamos que se familiaricen con el lenguaje de programación en assembly RISC-V[1] (*risk-five*), aplicando los conocimientos adquiridos en clases hasta el momento, en particular respecto a números enteros, branching, y otras propiedades de un computador básico, junto con llevar a cabo una pequeña investigación bibliográfica.

Enunciado

Motivación

A modo de introducción, verán dos imágenes del videojuego Shadow Man (1999) en dos consolas diferentes, N64 y PS1:

Figura 1: Shadow Man en N64

Figura 2: Shadow Man en PS1

Más allá de las diferencias en los colores y texturas, se puede notar que los voxeles ("pixeles" en un espacio 3D) de la versión de PS1 son considerablemente más grandes que en la N64, y además parecieran no estar perfectamente ubicados, es aparente si se fijan en el hombro y muñeca derechas. Además, si ven un video comparativo, verán que en la PS1 pareciera que todo "tirita" un poco, esto y lo anterior, se debe a, entre otras diferencias de hardware y programación, que la PS1 no poseía una unidad de procesamiento para punto flotante (FPU por sus siglas en inglés).

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

Parte 1

En esta parte se debe entregar un breve informe (1-2 planas A4), en el que investigues sobre al menos 2 soluciones al problema de cómo programar aplicaciones 3D en equipos sin una FPU, mencionándolas y explicando a grandes rasgos cómo se implementan. Deberán elegir una de ellas y explicar en detalle su implementación. Recuerden citar referencias en medida de lo necesario.

3 puntos.

Parte 2

Una vez terminada su investigación, deberán implementar en assembly RISC-V su solución elegida, para esto deben entregar dos scripts diferentes.

2.1

El primero, deberá recibir como input en la sección `.data` dos tuplas de coordenadas en un espacio cartesiano, que representan figuras tridimensionales, la cantidad de aristas de las mismas, y luego calcular el centro de masa de ambas figuras, considerando que la densidad de cada figura es constante e igual para ambas.

La sección `.data` se deberá ver como sigue, respetando los labels que se indican (pueden agregar más si lo consideran necesario):

```
.data
    # --- No modificar labels ---
    N1: .word 3  # entero >= 3, indica el número de vertices
    C1: .word 10,20,10, 0,0,0, 5,-4,10  # secuencia de N1*3 números, representa a las vertices
    N2: .word 4 # entero >= 3, indica el número de vertices
    C2: .word
    # --- End no modificar labels---
    # de aca para abajo van sus variables en memoria
```

El resultado final, que corresponde a una coordenada, deberá almacenarse en los registros `a0, a1, a2`, para las coordenadas x, y, z, respectivamente.

2.2

El segundo script, deberá recibir como input 3 valores, que representan dos caras de un triángulo y el ángulo entre ellos, y deberán aproximar el valor del lado faltante, utilizando el teorema del coseno u otra función similar.

La sección `.data` se deberá ver como sigue, respetando los labels que se indican (pueden agregar más si lo consideran necesario):

```
.data
    # --- No modificar labels ---
    A: .word a # indica el largo de una arista
    B: .word b # '' '' ''
    D: .word d # es el angulo entre los lados
    # --- End no modificar labels---
    # de aca para abajo van sus variables en memoria
```

El resultado final, que corresponde a un valor numérico, deberá almacenarse en el registro `a0` al finalizar el programa.

3 puntos.

Evaluación parte 2

Assembly

El código deberá estar adecuadamente comentado, de manera de facilitar la corrección y *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 en el minicurso y la documentación.

```
1        # para sumar dos numeros, escribir
2        add a4, a3, a4
3        # NO HACER
4        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 3: Convención de llamada para registros. Free & Open RISC-V Reference Card, RISC-V Organization.

El desglose del puntaje será el siguiente: 1 punto por pasar el assembler y respetar la convención de llamada, en ambos scripts. Luego, para cada uno de los dos scripts, 0,5 punto por implementar una solución de acuerdo con lo expuesto en el informe. 0,5 punto por pasar 6 tests con diferentes inputs, y entregar correctamente una respuesta de acuerdo a su implementación, para un total de 1 punto por script.

Emulador

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, disponible en el repositorio del curso, 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 la sección de programación**.

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.

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

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 fecha de entrega es el viernes 8 de abril, a las 23:59 por Canvas.

El formato a entregar será un archivo por item, `.asm` con sus respectivas soluciones en assembly RISC-V de la tarea y un documento PDF con sus respuestas a la pregunta de desarrollo. Se habilitará en Canvas un buzón para cada ítem.