Plancha 4: Ensamblador de RISC-V

2019 – Arquitectura del Computador
Licenciatura en Ciencias de la Computación
Entrega: fecha a determinar

1. Introducción

Esta plancha de ejercicios introduce la arquitectura RISC-V y su lenguaje de ensamblador.

2. Procedimiento

Resuelva cada ejercicio en computadora. Cree un subdirectorio dedicado para cada ejercicio, que contenga todos los archivos del mismo. Para todo ejercicio que pida escribir código, genere un programa completo, con su función `main` correspondiente; evite dejar fragmentos sueltos de programas.

Asegúrese de que todos los programas que escriba compilen correctamente con `gcc`. Se recomienda además pasar a este las opciones `-Wall` y `-Wextra` para habilitar advertencias sobre construcciones cuestionables en el código.

Una vez terminada la plancha, suba una entrega al repositorio Subversion de la materia, creando una etiqueta en el subdirectorio correspondiente a su grupo:
`https://svn.dcc.fceia.unr.edu.ar/svn-no-anon/lcc/R-222/Alumnos/2019/`

3. Ejercicios

1) Considere que tiene un dispositivo embebido que utiliza una CPU RV64I sin la extensión M. Necesita hacer divisiones de enteros arbitrarios en su programa, pero no dispone de instrucciones para esto.

Implemente en ensamblador una función de división por software:

```
int div(int a, int b);
```

donde `a` sea el dividendo, `b` sea el divisor y el valor de retorno sea el cociente.

Incluya una función `main` en C.

2) Implemente en ensamblador de RV64IM la función factorial, de la misma forma en que la implementó para x86_64 en el ejercicio 5 de la plancha 2. Haga tanto la versión iterativa como la recursiva.

3) ¿Cómo se representa el color? A lo largo de décadas, se han desarrollado muchos formatos distintos para dispositivos distintos: televisores, cámaras, monitores, impresoras, y también para almacenar archivos de imagen o video. Dos de los formatos más usados son RGB y Y'C_BC_R.

RGB es un modelo de color aditivo que trabaja en base a 3 colores: rojo, verde y azul; todo se representa como combinación de distintas intensidades de estos tres. Se codifica con una tripleta (R, G, B) donde cada componente, denominada canal, representa la intensidad de cada color de

los nombrados. Comúnmente, cada canal varía en el rango de 0 a 255, es decir que ocupa 1 byte. Algunos ejemplos: el valor $(0, 0, 0)$ representa el negro, $(255, 255, 255)$ es el blanco, $(255, 0, 0)$ es rojo a la máxima intensidad, $(80, 0, 0)$ es un rojo más oscuro y $(255, 0, 255)$ es violeta. Es común representar estos 3 bytes en uno solo y usar la notación `#RRGGBB` donde `RR`, `GG` y `BB` son los dos dígitos en hexadecimal de cada canal.

$\text{Y'C}_\text{B}\text{C}_\text{R}$, por su parte, separa la codificación del brillo de la codificación de color. Se trata también de una tripleta, (Y, C_B, C_R). En este caso, Y representa el brillo (hablando estrictamente, la luma) sin indicar nada sobre el color (croma); C_B, por su parte, indica la diferencia entre el brillo y el color azul, mientras que C_R lo hace lo propio con el rojo.

Imagine que tiene una colección de archivos de video, algunos codificados con RGB y otros con $\text{Y'C}_\text{B}\text{C}_\text{R}$. Por razones de espacio, desearía convertir todos los videos a escala de grises.

a) Escriba en ensamblador de RISC-V una función:

```
unsigned char rgb_to_gray1(unsigned rgb);
```

que convierta un pixel de color codificado en RGB en un pixel en escala de grises. Los 8 bits menos significativos de la entrada indican el canal rojo, los 8 siguientes el verde y los posteriores 8 el azul.

Asuma que cada canal tiene el mismo peso en el cálculo del brillo en escala de grises.

b) Al aplicar la función anterior, las imágenes quedan muy oscuras. Esto es porque, en realidad, los humanos percibimos cada color primario de RGB con un peso distinto. Las proporciones aproximadas que se suelen aplicar son: 30 % rojo, 59 % verde, 11 % azul.

Escriba una función `rgb_to_gray2` que aplique esta corrección.

c) Haga una función análoga para $\text{Y'C}_\text{B}\text{C}_\text{R}$:

```
unsigned char ycbcr_to_gray(unsigned ycbcr);
```

La entrada codifica Y en los 8 bits menos significativos, C_B en los 8 que siguen y C_R en los 8 posteriores.

Para probar todas las implementaciones anteriores, escriba una función `main` en C.

4) `j`, `call`, `ret`, `la`, `mv`, `fmv.s` y `fabs.d` no son instrucciones propiamente dichas, sino que son algunos ejemplos de pseudoinstrucciones: el ensamblador las traduce automáticamente a una secuencia de instrucciones más simples.

Indique secuencias de una o más instrucciones con las que podría reemplazar cada una de estas pseudoinstrucciones.

5) Las extensión F incorpora punto flotante de precisión simple a la arquitectura RISC-V. Escriba en ensamblador una función:

```
float det(float a, float b, float c, float d);
```

que calcule el determinante de una matriz 2×2:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Para este ejercicio, escriba la función `main` en ensamblador también.