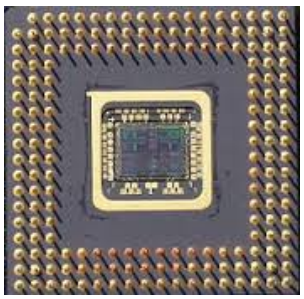
Proyecto de Arquitectura de Computadoras

Curso 2018-2019, v1.0

S-MIPS



Procesador S-MIPS

Introducción

El proyecto consiste en diseñar en LogiSim un procesador que implemente la arquitectura de juegos de instrucciones S-MIPS (Simplified-MIPS).

S-MIPS es una arquitectura de 32 bits. El procesador tiene 32 registros de 32 bits, nombrados R0 a R31, de propósito general, de ellos R0 siempre tiene el valor constante 0 independientemente de las operaciones que se realizen sobre él y R31 (SP) que actúa como puntero de la pila (stack pointer). Cuenta con dos registros adicionales **Hi** y **Lo** donde se almacena el resultado de la división y la multiplicación.

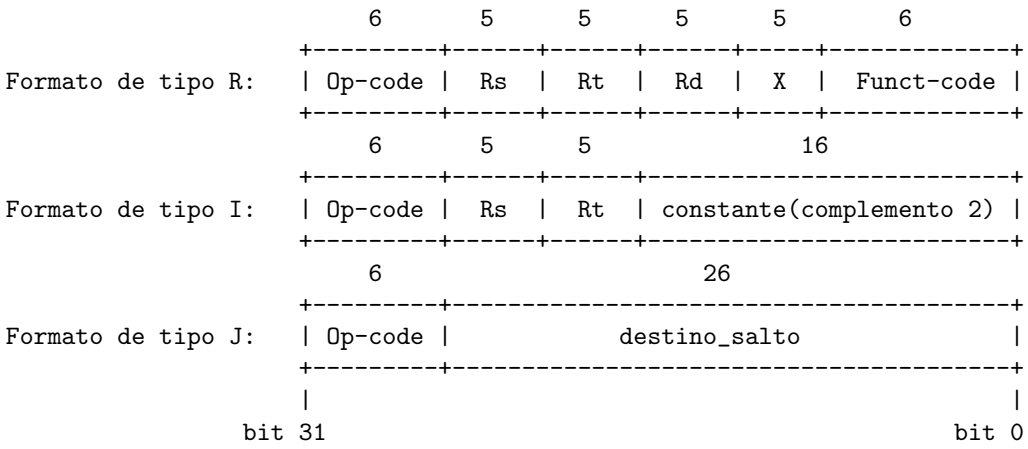
El procesador debe implementarse en el módulo **S-MIPS** del circuito **S-MIPS Board** que se brinda. No debe modificar el circuto **RAM** ni el circuto **S-MIPS Board**. Durante la evaluación se utilizarán estos circuitos tal y como se les entregó.

Instrucciones y su formato

Notas generales:

- $\mathbf{R_s}$, $\mathbf{R_t}$ y $\mathbf{R_d}$ especifican registros de propósito general.
- Un elemento entre corchetes ([ ]) indica .[el] contenido de". Por ejemplo $[R3]+[R22]$ se refiere a la suma de los valores almacenados en los registros R3 y R22.
- **[PC]** especifica la dirección de la instrucción en ejecución. Por ejemplo saltar a la próxima instrucción es $[PC] = [PC] + 4$
- **I** se refiere a los bits de la instrucción y el subindice indica a cuales de estos bits se refiere. $[I_{15..0}]$ se refiere al contenido de los 16 primeros bits de la instrucción, que en el caso de las instrucciones de tipo I es la constante.
- $\|$ indica la concatenación de bits.
- Los sobreíndices indican la repetición de un valor binario. 0^7 se refiere a: `0000000`.
- **M{I}** se refiere a los 32 bits almacenados en la memoria RAM en la dirección divisible por 4 más cercana al valor de I. $(I - I\,\%4)$.

Las instrucciones de S-MIPS tienen una longitud constante de 32 bits. Hay 3 formatos de instrucciones distintos:



Instrucciones

No operation: nop

Formato de tipo R:	000000	00000	00000	00000	00000	000000

Efectos de la instrucción:
$PC \leftarrow [PC] + 4$
Ensamblador: *nop*

Addition: add

Formato de tipo R:	000000	Rs	Rt	Rd	00000	100000

Efectos de la instrucción:
$R_d \leftarrow [R_s] + [R_t]$
$PC \leftarrow [PC] + 4$
Ensamblador: $add\ \ R_d, R_s, R_t$

Subtract: sub

Formato de tipo R:	000000	Rs	Rt	Rd	00000	100010

Efectos de la instrucción:
$R_d \leftarrow [R_s] - [R_t]$
$PC \leftarrow [PC] + 4$
Ensamblador: $sub\ \ R_d, R_s, R_t$

Multiply: mult

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 000000 |   Rs  |   Rt  | 00000 | 00000 | 011000 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
$Hi||Lo \leftarrow [R_s] * [R_t]$
$PC \leftarrow [PC] + 4$
Ensamblador: $mult\ \ R_s, R_t$

Unsigned multiply: mulu

Idéntica a la instucción **mult** excepto:
- Funct-code = 011001
- El contenido de R_s y R_t se considera como enteros sin signo.

Divide: div

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 000000 |   Rs  |   Rt  | 00000 | 00000 | 011010 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
$Lo \leftarrow [R_s]/[R_t]$
$Hi \leftarrow [R_s] mod [R_t]$
$PC \leftarrow [PC] + 4$
Ensamblador: $div\ \ R_s, R_t$

Unsigned divide: divu

Idéntica a la instucción **div** excepto:
- Funct-code = 011011
- El contenido de R_s y R_t se considera como enteros sin signo.

Set less than: slt

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 000000 |   Rs  |   Rt  |   Rd  | 00000 | 101010 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
si $[R_s] < [R_t]$ entonces $R_d \leftarrow 0^{31}||1$
sino $R_d \leftarrow 0^{32}$
$PC \leftarrow [PC] + 4$
Ensamblador: $slt\ \ R_d, R_s, R_t$

Unsigned set less than: sltu

Idéntica a la instucción **slt** excepto:
- Funct-code = 101011
- El contenido de R_s y R_t se considera como enteros sin signo.

Logical and: and

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 000000 |   Rs  |   Rt  |   Rd  | 00000 | 100100 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
$R_d \leftarrow [R_s] AND [R_t]$
$PC \leftarrow [PC] + 4$
Ensamblador: $and\ \ R_d, R_s, R_t$

Logical or, nor, xor: or, nor, xor

Idénticas a la instucción **and** excepto:
- Funct-code = 100101 para la instruccion **or**
- Funct-code = 100111 para la instruccion **nor**
- Funct-code = 101000 para la instruccion **xor**
- Se realiza la función lógica apropiada en vez de **and**.

Addition inmediate: addi

```
                    +---------+------+------+--------------------------+
Formato de tipo I:  | 001000  |  Rs  |  Rt  |        constante         |
                    +---------+------+------+--------------------------+
```

Efectos de la instrucción:
$R_t \leftarrow [R_s] + ([I_{15}]^{16} || [I_{15.,0}])$
$PC \leftarrow [PC] + 4$
Ensamblador: $addi\ \ R_t, R_s, constante$, por ejemplo: $addi\ \ R_3, R_8, 34$

Set less than immediate: slti

```
                    +--------+-------+-------+-----------------------+
Formato de tipo I:  | 001010 |   Rs  |   Rt  |       constante       |
                    +--------+-------+-------+-----------------------+
```

Efectos de la instrucción:
si $[R_s] < ([I_{15}]^{16} || [I_{15.,0}])$ entonces $R_t \leftarrow 0^{31} || 1$
sino $R_t \leftarrow 0^{32}$
$PC \leftarrow [PC] + 4$
Ensamblador: $slti\ \ R_t, R_s, constante$

Set less than immediate unsigned: sltiu

Idéntica a la instucción **slti** excepto:
- Funct-code = 001011
- El contenido de R_s y R_t se considera como enteros sin signo.

Logical and immediate: andi

```
                    +--------+-------+-------+-----------------------+
```

```
Formato de tipo I:   | 001100 |   Rs  |   Rt  |       constante        |
                     +--------+-------+-------+------------------------+
```

Efectos de la instrucción:
$R_t \leftarrow [R_s] AND (0^{16} || [I_{15..0}])$
$PC \leftarrow [PC] + 4$
Ensamblador: $andi\ \ R_t, R_s, constante$

Logical or immediate & xor immediate: ori, xori

Idénticas a la instucción **andi** excepto:
- Funct-code = 001101 para la instruccion **ori**
- Funct-code = 001110 para la instruccion **xori**
- Se realiza la función lógica apropiada en vez del **and** lógico.

Load word: lw

```
                     +--------+-------+-------+------------------------+
Formato de tipo I:   | 100011 |   Rs  |   Rt  |        offset          |
                     +--------+-------+-------+------------------------+
```

Efectos de la instrucción:
$R_t \leftarrow M\{[R_s] + [I_{15}]^{16} || [I_{15..0}]\}$
$PC \leftarrow [PC] + 4$
Ensamblador: $lw\ \ R_t, offset(R_s)$ por ejemplo: $lw\ \ R_3, 16(R_0)$

Store word: sw

```
                     +--------+-------+-------+------------------------+
Formato de tipo I:   | 101011 |   Rs  |   Rt  |        offset          |
                     +--------+-------+-------+------------------------+
```

Efectos de la instrucción:
$M\{[R_s] + [I_{15}]^{16} || [I_{15..0}]\} \leftarrow [R_t]$
$PC \leftarrow [PC] + 4$
Ensamblador: $sw\ \ R_t, offset(R_s)$

Pop from stack: pop

```
                     +--------+-------+-------+-------+-------+--------+
Formato de tipo R:   | 111000 | 00000 | 00000 |  Rd   | 00000 | 000000 |
                     +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
$R_d \leftarrow M\{[SP]\}$
$SP \leftarrow [SP] + 4$
$PC \leftarrow [PC] + 4$
Ensamblador: $pop\ \ R_d$

Push to stack: push

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 111000 |   Rs  | 00000 | 00000 | 00000 | 000001 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
$SP \leftarrow [SP] - 4$
$M\{[SP]\} \leftarrow R_s$
$PC \leftarrow [PC] + 4$
Ensamblador: $push\ R_s$

Branch on equal: beq

```
                    +--------+-------+-------+------------------------+
Formato de tipo I:  | 000100 |   Rs  |   Rt  |        offset          |
                    +--------+-------+-------+------------------------+
```

Efectos de la instrucción:
si $[R_s] == [R_t]$ entonces $PC \leftarrow [PC] + 4 + ([I_{15}]^{14}||[I_{15.,0}]||0^2)$ o sea: $PC \leftarrow [PC] + 4 + 4 * offset$
sino $PC \leftarrow [PC] + 4$
Ensamblador: $beq\ R_s, R_t, offset$

Branch on not equal: bne

```
                    +--------+-------+-------+------------------------+
Formato de tipo I:  | 000101 |   Rs  |   Rt  |        offset          |
                    +--------+-------+-------+------------------------+
```

Efectos de la instrucción:
si $[R_s] <> [R_t]$ entonces $PC \leftarrow [PC] + 4 + ([I_{15}]^{14}||[I_{15.,0}]||0^2)$ o sea: $PC \leftarrow [PC] + 4 + 4 * offset$
sino $PC \leftarrow [PC] + 4$
Ensamblador: $bne\ R_s, R_t, offset$

Branch on less than or equal zero: blez

```
                    +--------+-------+-------+------------------------+
Formato de tipo I:  | 000110 |   Rs  |   Rt  |        offset          |
                    +--------+-------+-------+------------------------+
```

Efectos de la instrucción:
si $[R_s] <= 0$ entonces $PC \leftarrow [PC] + 4 + ([I_{15}]^{14}||[I_{15.,0}]||0^2)$ o sea: $PC \leftarrow [PC] + 4 + 4 * offset$
sino $PC \leftarrow [PC] + 4$
Ensamblador: $blez\ R_s, offset$

Branch on greater than zero: bgtz

```
                    +--------+-------+-------+------------------------+
Formato de tipo I:  | 000111 |   Rs  |   Rt  |        offset          |
                    +--------+-------+-------+------------------------+
```

Efectos de la instrucción:
si $[R_s] > 0$ entonces $PC \leftarrow [PC] + 4 + ([I_{15}]^{14}|||[I_{15.,0}]||0^2)$ o sea: $PC \leftarrow [PC] + 4 + 4 * offset$
sino $PC \leftarrow [PC] + 4$
Ensamblador: $bgtz\ R_s, offset$

Branch on less than zero: bltz

```
                    +--------+-------+-------+------------------------+
Formato de tipo I:  | 000001 |   Rs  |   Rt  |        offset          |
                    +--------+-------+-------+------------------------+
```

Efectos de la instrucción:
si $[R_s] < 0$ entonces $PC \leftarrow [PC] + 4 + ([I_{15}]^{14}|||[I_{15.,0}]||0^2)$ o sea: $PC \leftarrow [PC] + 4 + 4 * offset$
sino $PC \leftarrow [PC] + 4$
Ensamblador: $bltz\ R_s, offset$

Jump: j

```
                    +--------+---------------------------------------+
Formato de tipo J:  | 000010 |                destino                |
                    +--------+---------------------------------------+
```

Efectos de la instrucción:
$PC \leftarrow [PC_{31.,28}]|||[I_{25.,0}]||0^2$
Ensamblador: $j\ destino$

Jump register: jr

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 000000 |   Rs  | 00000 | 00000 | 00000 | 001000 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
$PC \leftarrow [R_s]$
Ensamblador: $jr\ R_s$

Move from Hi: mfhi

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 000000 | 00000 | 00000 |  Rd   | 00000 | 010000 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
$R_d \leftarrow [Hi]$
$PC \leftarrow [PC] + 4$
Ensamblador: $mfhi\ R_d$

Move from Lo: mflo

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 000000 | 00000 | 00000 |  Rd   | 00000 | 010010 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
$R_d \leftarrow [Lo]$
$PC \leftarrow [PC] + 4$
Ensamblador: $mflo\ R_d$

Instrucciones especiales

Las siguientes instrucciones no son realistas pero son necesarias para simular una entrada de teclado y una terminal sin complicar la interfaz de simulación.

Halt program: halt

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 111111 | 00000 | 00000 | 00000 | 00000 | 111111 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
- Detiene la simulación del programa actual.
Ensamblador: *halt*

Write to terminal: tty

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 111111 |   Rs  | 00000 | 00000 | 00000 | 000001 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
Envia un caracter a la pantalla conectada al procesador (TTY). La pantalla es un circuito síncrono. Para enviar un caracter, se ponen los 7 bits menos significativos de Rs en la salida TTY DATA del procesador, se activa la salida TTY ENABLE y en el próximo ciclo la pantalla mostrará el caracter ASCII correspondiente a los 7 bits de TTY DATA.
Ensamblador: $tty\ R_s$

Set random: rnd

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 111111 | 00000 | 00000 |   Rd  | 00000 | 000010 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
Almacena en Rd un número aleatorio.
Ensamblador: $rnd\ R_d$

Set key: kbd

```
                    +--------+-------+-------+-------+-------+--------+
Formato de tipo R:  | 111111 | 00000 | 00000 |   Rd  | 00000 | 000100 |
                    +--------+-------+-------+-------+-------+--------+
```

Efectos de la instrucción:
Esta instrucción lee un caracter del teclado y lo almacena en Rd. La entrada KBD AVAILABLE del procesador indica si hay algún caracter esperando en el buffer del teclado y la entrada KBD DATA, es un

número de 7 bits que corresponde al código ASCII del caracter que está en la punta del buffer. El buffer del teclado funciona como una cola. Los caracteres se añaden a la cola cuando se teclea. En cada ciclo de reloj, si KBD ENABLE está activa, el teclado elimina el caracter que está en la punta de la cola.

Si en el momento que esta instrucción se ejecuta, KBD AVAILABLE está desactivada, en Rd se almacena el valor -1.

Ensamblador: *kbd* R_d

Program Counter

El valor del registro Program Counter (PC) representa la dirección de memoria de la próxima instrucción que se va a ejecutar. Como en S-MIPS las instrucciones ocupan 4 bytes, cada vez que el procesador ejecuta una instrucción que no sea de salto el valor de PC aumenta en 4.

Las instrucciones de salto (beq, bne, blez, ...) suman su argumento (en complemento a 2) al de PC. El argumento indica el número de instrucciones a saltarse,por tanto es necesario multiplicar el argumento por 4 antes de sumarlo al program counter.

Como resultado de esto, si un programa quisiera saltarse una instrucción, debe hacer un salto con argumento 1:

```
add R3, R0, 46
add R4, R0, 46
beq R3, R4, 1
halt
```

En este ejemplo la instrucción halt simpre se salta. Otro resultado de este comportamiento es que `beq R0, R0, 0` es lo mismo que `nop` y `beq R0, R0, -1` es un ciclo infinito.

Acceso a la memoria

La memoria, para el programador de S-MIPS, es un array plano de 1MB, direccionable a nivel de 1 byte. La instrucción **lw** cuenta con 32 bits para direccionar la memoria RAM pero en el board utilizado solo hay instalados 1MB de memoria por tanto solo se van a utilizar para direccionar los valores en la RAM los 20 primeros bits de esta. Nótese además que las transferencias entre la RAM y el procesador ocurren siempre en bloques de 4 bytes (32 bits).

El procesador puede ignorar cualquier dirección de memoria que no sea divisible entre 4 e interpretarla como si fuera el número divisible por 4 más cercano por debajo. O sea las instrucciones `lw R3, 0(R0)`, `lw R3, 1(R0)`, `lw R3, 2(R0)` y `lw R3, 3(R0)` hacen lo mismo.

Igualmente si un programa hace:

```
lw R3, 0(R0)
```

al registro R3 se copiarán los bytes 0, 1, 2 y 3 de la RAM.

S-MIPS es una arquitectura Little-Endian, que significa que, en cada palabra de 32 bits, el byte con dirección más pequeña es el menos significativo y el de dirección más grande es el más significativo. Esto luce "al revés" de cómo se escriben normalmente los números.

Supongamos que los primeros bytes de la RAM tienen estos valores:

0	1	2	3	4	5	
1	2	0	0	1	2	$\cdots$

Al hacer

```
lw R3, 0(R0)
```

el byte 0 de la RAM, que tiene el valor 1 (00000001) va hacia la parte menos significativa de R3 y el byte 3, con valor 0 (00000000), va hacia la más significativa. El valor final de R3 sería $2^9 + 2^0 = 513$

Las escrituras se comportan de igual forma. Si se hace:

```
add R3, R0, 1027
sw  R3, 0(R0)
```

se escribirán para el byte 0 de la RAM la parte menos significativa del valor 1027 y para el byte 3 la parte más significativa.

1027 es 00000000 00000000 00000100 00000011 en binario. El byte más significativo es 0 y el menos significativo es 3 ($1027 = 2^{10} + 2^1 + 2^0$). Así que la memoria quedaría así:

0	1	2	3	4	5	
3	4	0	0	1	2	$\cdots$

Interfaz con la memoria

El módulo RAM del circuito S-MIPS Board implementa una memoria asíncrona de 1MB. Esta memoria está organizada en 65536 bloques de 16 bytes. Esta RAM es más lenta que el procesador, y le toma varios ciclos leer y escribir datos. La cantidad de ciclos que toma hacer una lectura o una escritura están, respectivamente, en las salidas RT (Read Time) y WT (Write Time) que no cambian durtante la ejecución de un programa.

(Aunque estas salidas sean constantes, no se deben usar sus valores "a mano" dentro del CPU. Hay que obtenerlos de la RAM. Los profesores vamos a probar distintas combinaciones de esos valores durante la verificación del procesador y este debe comportarse en concordancia.)

Como la RAM tiene 65536 bloques, su entrada ADDR es de 16 bits. Después de pasados RT ciclos de CPU de establecerse esa entrada, si la entrada CS (Chip Select) está en 1 y la entrada ¬R/W está en 0, la RAM proporcina los 16 bytes del bloque seleccionado mediante las 4 salidas de 32 bits O0, O1, O2 y O3.

Análogamente, hay 4 entradas de 32 bits, I0, I1, I2 e I3, por las que se envían a la RAM valores para ser escritos en la dirección ADDR. Sin embargo, la escritura puede hacerse parcialmente, usando la entrada MASK.

La RAM está dividida en 4 bancos que actúan como columnas o slices. Cada bloque de 16 bytes de la RAM por tanto está dividido en 4 palabras de 4 bytes (32 bits). Los primeros 32 bytes de la RAM lucen así:

Banco 0	Banco 1	Banco 2	Banco 3	
00 01 02 03	04 05 06 07	08 09 10 11	12 13 14 15	← Bloque 0
16 17 16 19	20 21 22 23	24 25 26 27	28 29 30 31	← Bloque 1
32 33 34 35	36 37 38 39	40 41 42 43	44 45 46 47	← Bloque 2
48 49 50 51	52 53 54 55	56 57 58 59	60 61 62 63	← Bloque 3
⋮	⋮	⋮	⋮	⋮

La entrada MASK es una entrada de 4 bits que selecciona cuál o cuales bancos van a ser modificados por una operación de escritura. El bit menos significativo de MASK selecciona el Banco 0, el más significativo selecciona el Banco 3.

Por ejemplo, si la entrada MASK tiene el valor 8 (en binario 1000) y la entrada ADDR es 0, la escritura solo afectaría los bytes 12, 13, 14 y 15 de la RAM, pues estos están en el bloque 0 y en el banco 3. En esos 4 bytes se escribiría el valor de la entrada I3 (Data Input 3).

Si la entrada MASK fuera 12 y ADDR fuera 2 (en binario 1100), la escritura modificaría los bytes 40, 41, 42, 43 y 44, 45, 46, 47 (bloque 2, bancos 2 y 3). En esos 16 bytes se escribirían los valores de las entradas I2 e I3.

Las entrada MASK no afecta las operaciones de lectura. Las salidas O0, O1, O2, O3 siempre contienen el bloque completo solicitado en ADDR.

Las escrituras se realizan cuando CS es 1 y ¬R/W es 1 y toman la cantidad de ciclos de CPU que indica la salida WT.

Caché

La implementación del procesador S-MIPS debe incluir una caché de instrucciones y una cache de datos para acelearar el proceso de lectura y/o escritura en la RAM. En ambos casos se puede aprovechar el hecho de que a partir de una sola lectura de la RAM se obtiene un bloque completo de 4 x 32bits.

Nota: En caso de no implementar ningun tipo de caché la nota máxima a obtener es de 4ptos.

Costo y Presupuesto

El procesador va a tener un presupuesto máximo. Cada componente utilizada tiene un costo asociado y el costo total del procesador no puede tener un precio mayor de 100. Adjuto a este PDF encontrará un script `price.py` que le permite calcular el costo de un circuito, ejecute `python price.py -h` para ver la ayuda. Los costos específicos de cada componente se pueden consultar en el código fuente de este script.

Evaluación

El proyecto debe realizarse en equipos de no más de dos (2) estudiantes. El plazo para entregar los proyectos es el viernes 26 de abril de 2019. Después de la entrega, se realizará una revisión oral con un profesor y los miembros de cada equipo.

Se puede entregar y revisarse en cualquier momento antes de la expiración del plazo.

Pasar todos los casos de prueba es un requisito para realizar la revisión oral. El proyecto no se aprueba si el circuito no ha pasado todas la pruebas antes de o en el día de la entrega. Ninguno de los casos de prueba obligatorios comprueban la existencia y/o funcinamiento de una memoria caché.

La nota del proyecto se decide en la revisión oral.