
# Guía de Ejercicios Resuelta

[66.20] Organización de Computadoras
1er cuatrimestre de 2020

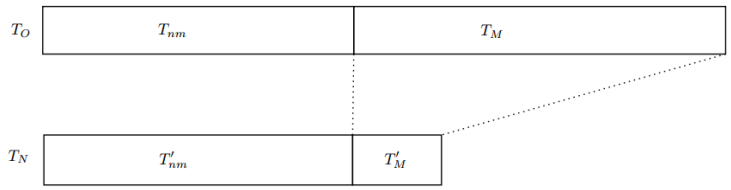
# Índice

# 1. Principios fundamentales

## 1.1. Ejercicio resuelto: [1.2 $3^{ed}$ CAAQA]

Se considera realizar una mejora a una máquina agregando hardware vectorial. Cuando un cálculo de modo vectorial se ejecuta en este hardware, este se realiza 10 veces más rápido que en modo de ejecución normal. Llamamos al porcentaje de tiempo que se puede utilizar este modo vectorial como porcentaje de vectorización.

Antes de resolver el ejercicio, planteamos el escenario mediante un diagrama:




Donde

- $T_O$ corresponde al tiempo de ejecución del programa antes de aplicar la optimización
- $T_N$ corresponde al tiempo de ejecución del programa luego de aplicar la optimización. Obtenemos la figura de *Speed Up* global, $SU_g = \frac{T_O}{T_N}$
- $T_M$ corresponde al tiempo de ejecución del programa en el que es factible aplicar la optimización. $f = \frac{T_m}{T_O}$ determina el porcentaje de vectorización.
- $T'_M$ corresponde al tiempo de ejecución del programa en el que la optimización se está ejecutando. Obtenemos la figura de *Speed Up* local, $SU_L = \frac{T_M}{T'_M}$
- $T_{nm}$ corresponde al tiempo en el que no se está aplicando ninguna optimización

1. Realizar un gráfico que represente el *Speed Up* en función del como porcentaje de vectorización. Nombrar al *Eje Y* como *Speed Up* Global ($SU_g$) y al *Eje X* como Porcentaje de Vectorización.

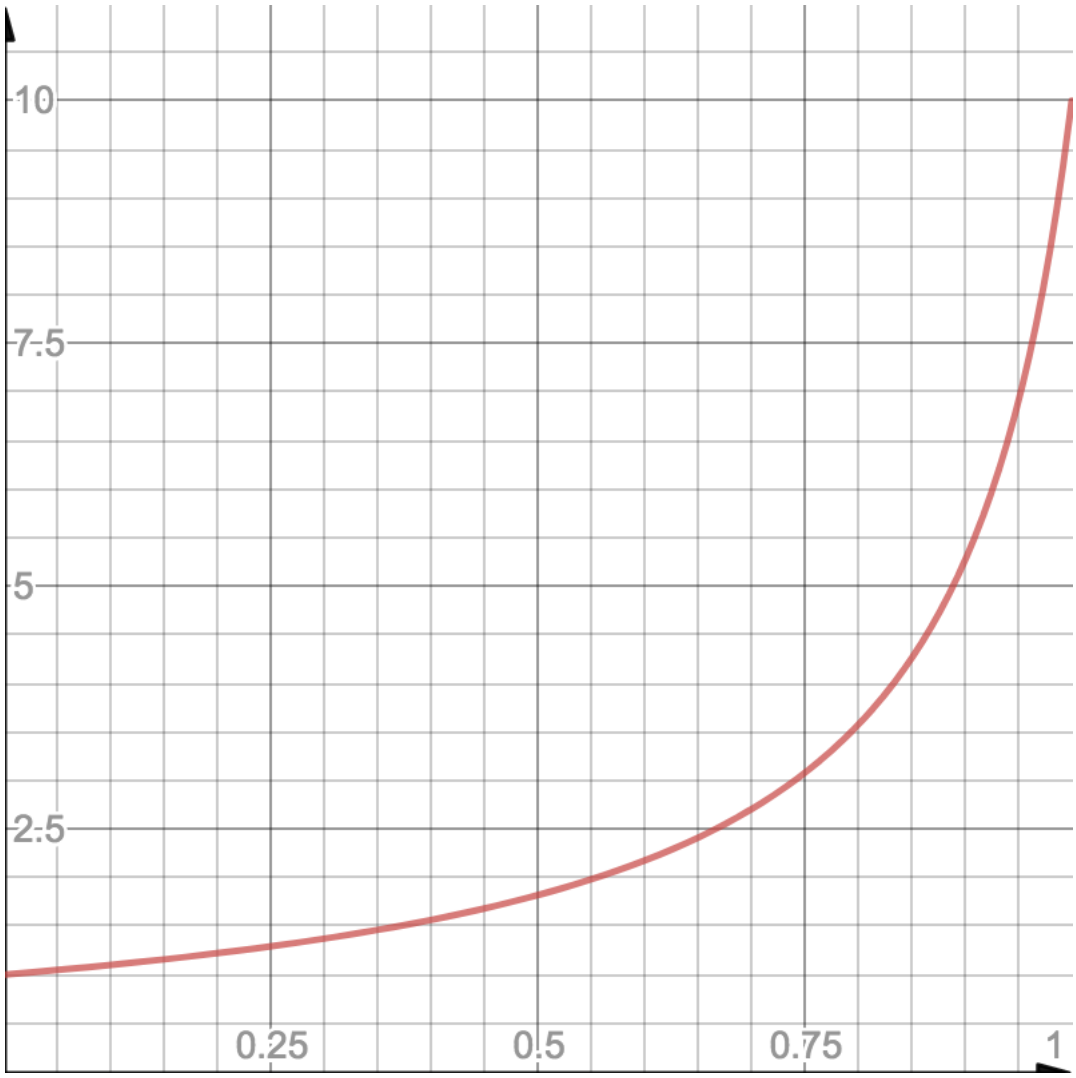
   ———

   Partiendo de la Ley de Amdahl, así como los datos del enunciado, tenemos que

   $$SU_g = \frac{1}{(1-f) + \frac{f}{10}} = \frac{1}{1 - 0,9 \times f}$$

   Sabemos también que $f \in [0, 1]$. Podemos calcular el valor de *Speed Up* en los extremos:

   - Si no es posible aplicar la mejora, $f = 0 \Rightarrow SU_g = 1$
   - Si la mejora es aplicable a todo el tiempo de ejecución, $f = 1 \Rightarrow SU_g = SU_L = 10$




2. ¿Qué porcentaje de vectorización en necesaria para alcanzar un *Speed Up* global de 2?

   ---

   Utilizando la ley de Amdahl podemos despejar $f$

$$2 = \frac{1}{1 - 0,9 \times f}$$
$$f = \frac{5}{9}$$

3. ¿Qué porcentaje de tiempo se emplea en modo vectorización cuando se alcanza un *Speed Up* global de 2?

   ---

   Es importante entender que esta pregunta no es igual a la anterior. El escenario para este punto nos plantea en el momento en el que la optimización ya se encuentra en ejecución.

   Partiendo de la definición de *Speed Up* , tenemos que

$$SU_g = \frac{T_O}{T_N} = 2 \Rightarrow f = \frac{5}{9}$$

$$SU_L = \frac{T_m}{T'_m} = 10, \quad siendo \quad T_m = T_O \times f = \frac{5}{9} T_O$$

También es importante notar que $T'_{nm} = T_{nm} = \frac{4}{9} \times T_O$, ya que el tiempo en el que no se aplica ninguna optimización no cambia.

El ejercicio nos pide calcular $f' = \frac{T'_m}{T_N}$. Se puede determinar que

$$T'_m = \frac{1}{10} \times T_m$$
$$T_N = T'_m + T'_{nm}$$
$$T_{nm} = T'_{nm}$$

$$f' = \frac{\frac{5}{90}}{\frac{5}{90} + \frac{4}{9}}$$

4. ¿Qué porcentaje de vectorización es necesaria para alcanzar la mitad del máximo *Speed Up* posible al usar el modo de vectorización?

   ———

   A partir de la Ley de Amdahl y el análisis realizado en el primer punto sabemos qué $SU_{max} = 10$, por lo que solo resta resolver

$$\frac{1}{5} = 1 - 0,9 \times f$$
$$f = \frac{8}{9}$$

5. Comparar los siguientes escenarios de mejora:

   a) Se duplica el SpeedUp local

   b) Se aumenta el porcentaje de vectorización con el compilador.

   Condición base $f_v = 0{,}7$

   ———

   En definitiva lo que nos piden es una comparación de esfuerzos. Duplicar el speed up local implica duplicar la velocidad, es una mejora del hardware. En cambio, aumentar la fracción de vectorización es una mejora en el compilador.

   Calculamos el Speed Up global base, es decir, sin aplicar ninguna de las mejores:

$$SU_g = \frac{1}{1 - 0{,}7 + \frac{0{,}7}{10}} = 2{,}70$$

   Si duplicamos el Speed Up local, obtendremos:

$$SU_g = \frac{1}{1 - 0{,}7 + \frac{0{,}7}{20}} = 2{,}985$$

   Ahora, para poder comparar ambas opciones, calculamos la fracción de vectorización que necesitaríamos para alcanzar el mismo Speed Up global:

$$SU_g = \frac{1}{1 - f + \frac{f}{10}} = 2{,}985$$

$$SU_g = \frac{1}{1 - 0{,}9xf} = 2{,}985 \Rightarrow f = 0{,}739$$

Entonces para alcanzar la misma mejora tenemos dos opciones: duplicar la velocidad o aumentar la fracción de vectorización de 0.7 a 0.739. La segunda requiere menos esfuerzo.

## 1.2.

Demostrar que

$$SU = \frac{1}{(1-f) + \frac{f}{SU_L}}$$

Donde $f$ es la fracción de tiempo a mejorar, y $SU_L$ es la mejora local.

---

$$SU_g = \frac{T_{nm} + T_m}{T'_{nm} + T'_m} = \frac{T_{nm} + T_m}{T_{nm} + T'_m}$$

$$= \frac{T_m/f}{T_{nm} + T'_m}$$

y como $T_{nm} = T_v - T_m = \frac{T_m}{f} - T_m \Rightarrow$

$$SU_g = \frac{T_m/f}{T_m/f - T_m + T'_m}$$

multiplicando por $f$ ambos lados:

$$SU_g = \frac{T_m}{T_m - fxT_m + fxT'_m} = \frac{1}{1 - f + fx\frac{T'_m}{T_m}}$$

$$\Rightarrow SU_g = \frac{1}{1 - f + \frac{f}{SU_l}}$$

## 1.3.

Un procesador de $300Mhz$ ejecuta un programa que presenta los siguientes tipos de instrucciones:

| Tipo de instrucción | Frecuencia ( %) | Ciclos |
|---|---|---|
| Aritmético-Lógica | 40 | 1 |
| Carga | 20 | 1 |
| Almacenamiento | 10 | 2 |
| Saltos | 20 | 3 |
| Punto Flotante | 10 | 5 |

1. Calcular las tasas de CPI y MIPS, para el programa completo.
2. Suponga que una optimización elimina un 30 % de las instrucciones aritmético-lógicas (o sea, 12 % del total de instrucciones A-L), 30 % de instrucciones load y 20 % de punto flotante. ¿Cuál es el speedup alcanzado?
3. Recalcular las tasas de CPI y MIPS para el programa completo. Explicar las diferencias respecto al punto (a).

1. Se puede calcular el CPI promedio como un promedio ponderado de los CPIs de los distintos grupos de instrucciones:

$$CPI = \sum_i f_i \times CPI_i$$

donde $f_i$ es la frecuencia de ejecución de instrucciones del grupo $i$

Entonces

$$CPI = 1 \times 0{,}4 + 1 \times 0{,}2 + 2 \times 0{,}1 + 3 \times 0{,}2 + 5 \times 0{,}1 = 1{,}9$$

$$MIPS = \frac{f_{ck}}{CPIx10^6} = \frac{300MHz}{1,9x10^6} = 157,9$$

2. Calculo la frecuencia de los tipos de instrucciones luego de reducirlos:

a) Aritmético-Lógica: $\frac{30\,\% \times 40\,\%}{100\,\%} = 12\,\% \Rightarrow 40\,\% - 12\,\% = 28\,\%$

b) Carga: $\frac{30\,\% \times 20\,\%}{100\,\%} = 6\,\% \Rightarrow 20\,\% - 6\,\% = 14\,\%$

c) Punto flotante: $\frac{20\,\% \times 10\,\%}{100\,\%} = 2\,\% \Rightarrow 10\,\% - 2\,\% = 8\,\%$

Con estos datos, calculo el CPI de manera análoga al punto anterior.

$$CPI = 0,28 \times 1 + 0,14 \times 1 + 0,1 \times 2 + 0,2 \times 3 + 0,08 \times 5 = 1,62$$

Si supongo que sólo hubo cambios a nivel hardware, es decir, considero que no cambia la cantidad de instrucciones:

$$SU = \frac{CPUtime_{old}}{CPUtime_{new}} = \frac{IC \times Clockcycle \times CPI_{old}}{IC \times Clockcycle \times CPI_{new}} = \frac{1,62}{1,30} = 1,25$$

3.

$$CPI = 1,62$$

$$MIPS = \frac{300MHz}{1,62 \times 10^6} = 185,19$$

El MIPS dio mayor al reducir la cantidad de instrucciones y el CPI dio menor. Sin embargo, no podemos concluir que por eso la computadora sea mejor. No podemos comparar con diferentes sets de instrucciones basándonos en MIPS porque el IC va a ser distinto para ambas. Además, MIPS varía entre programas en una misma computadora, o sea que no hay un único valor MIPS para una computadora.

## 1.4.

Se proponen 3 mejoras para una nueva arquitectura con los siguientes Speedups:

- Speedup1 = 30
- Speedup2 = 20
- Speedup3 = 10

Sólo una mejora es aplicable en cada momento (no se pueden solapar).

1. Si las mejoras 1 y 2 se pueden usar un 30 % del tiempo, ¿qué fracción del tiempo se debe usar la mejora 3 para lograr un speedup global de 10?

2. Asumir que la distribución del uso de las mejoras es del 30 %, 30 % y 20 para las mejoras 1, 2 y 3 respectivamente. Asumir que las 3 mejoras están en uso. ¿Qué fracción del tiempo mejorado no tiene una mejora en uso?

3. Asumir que para un *benchmark* la fracción del uso de las mejoras es del 15 % para 1 y 2 y del 70 % para la mejora 3. Se quiere maximizar la performance. Si sólo una mejora puede ser aplicada, ¿cuál debería ser elegida? Si 2 mejoras pueden ser aplicadas, ¿cuales deberían ser elegidas?

---

1. Sabiendo que el Speed Up global cuando se aplica más de una mejora por separado es:

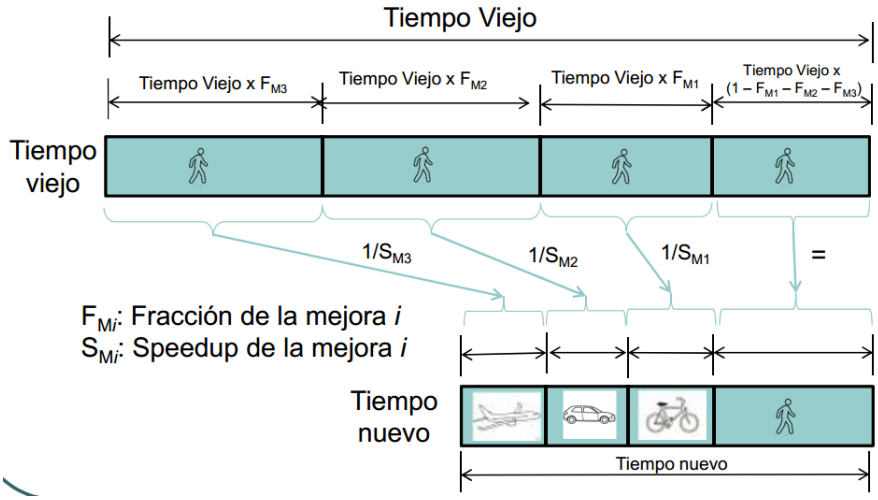
$$SU_g = \frac{1}{(1 - \sum_i F_{mi}) + \sum_i \frac{F_{mi}}{S_{mi}}}$$

$$SU_g = \frac{1}{(1 - f_1 - f_2 - f_3) + (\frac{f_1}{SU_1} + \frac{f_2}{SU_2} + \frac{f_3}{SU_3})}$$

$$\Rightarrow SU_g = \frac{1}{(1 - 0,3 - 0,3 - f_3) + (\frac{0,3}{30} + \frac{0,3}{20} + \frac{f_3}{10})}$$

$$\Rightarrow f_3 = 0,36$$

2. A partir del siguiente gráfico




Se puede ver que el tiempo mejorado ($T_n$) va a tener un intervalo de tiempo sin mejorar ($T'_{nm}$) que es igual al tiempo no mejorable del tiempo viejo ($T_{nm}$) (perdón, trabalenguas). O sea que $T'nm = T_{nm} = T_v \times (1 - f_1 - f_2 - f_3)$. Luego, divido este tiempo por $T_n$ porque me piden la fracción del tiempo mejorado.

$\Rightarrow x = \frac{(1-f_1-f_2-f_3)\times T_v}{T_n} = \frac{(1-f_1-f_2-f_3)\times T_v}{\frac{T_v}{SU_g}} = \frac{(1-f_1-f_2-f_3)}{(1-f_1-f_2-f_3)+\frac{f_1}{SU_1}+\frac{f_2}{SU_2}+\frac{f_3}{SU_3}} = 0,816$

3. Si una mejora pudiera ser aplicada: $f_1 = f_2 = 0,15$

   $f_3 = 0,7$

$$SU_1 = \frac{1}{1 - 0,15 + \frac{0,15}{30}} = 1,170$$

$$SU_2 = \frac{1}{1 - 0,15 + \frac{0,15}{20}} = 1,170$$

$$SU_3 = \frac{1}{1 - 0,7 + \frac{0,7}{10}} = 2,703$$

Entre la opción 1 y 2, es mejor la 1 porque tiene mayor Speed Up local. Entre las tres, es mejor la opción 3 ya que logra el mayor Speed Up global.

Si dos mejoras pudieran ser aplicadas:

$$SU_{12} = \frac{1}{1 - 0,15 - 0,15 + \frac{0,15}{30} + \frac{0,15}{20}} = 1,40$$

$$SU_{13} = \frac{1}{1 - 0,15 - 0,7 + \frac{0,15}{30} + \frac{0,7}{10}} = 4,44$$

$$SU_{23} = \frac{1}{1 - 0,15 - 0,7 + \frac{0,15}{20} + \frac{0,7}{10}} = 4,39$$

La mejor combinación es la 1-3

## 1.5.

Se dispone de un *benchmark* que contiene 195,578 instrucciones de punto flotante, y un número indeterminado de instrucciones de otro tipo.

Dicho *benchmark* fue ejecutado en un procesador embebido luego de haber sido compilado con las optimizaciones activadas. El procesador embebido está basado en un procesador RISC que incluye unidades de punto flotante, pero no dispone de ellas por distintas razones. El compilador permite calcular las operaciones de punto flotante mediante unidades punto flotante o mediante rutinas de software, dependiendo en las opciones utilizadas.

Salvo por la disponibilidad de unidades de punto flotante, ambos procesadores son idénticos, con una frecuencia $\nu = 16{,}67Mhz$

El *benchmark* se ejecutó en 1,08 segundos en el procesador RISC, mientras que tomó 13,6 segundos en la versión embebida. Asumir que el CPI del procesador RISC es 10, mientras que el CPI del procesador embebido es 6.

1. Para ambos procesadores, ¿cuántas instrucciones fueron ejecutadas?
2. Para ambos procesadores, ¿cuál es el valor de la tasa de MIPS?
3. En promedio, ¿cuántas instrucciones enteras son necesarias para ejecutar una operación de punto flotante en software?

---

Tenemos la siguiente información

| Procesador 1 (RISC) | Procesador 2 (Embebido) |
|---|---|
| Floating Point nativo | Floating Point emulado en instr. enteras |
| $\nu = 16,67Mhz$ | $\nu = 16,67Mhz$ |
| $T_{exr} = 1,08s$ | $T_{exe} = 13,6s$ |
| $CPI_r = 10$ | $CPI_e = 6$ |
| $IC_r = IC_{int} + IC_{fp}$ | $IC_e = IC_{int} + IC_{fp} \times Y$ |

Donde $Y$ es la cantidad de funciones enteras necesarias para emular una instrucción de punto flotante.

1. Sabiendo que $T_{ex} = CPI \times IC \times T_{ck} \Rightarrow$

$$IC_r = \frac{1,08 \times 16,67 \times 10^6}{10} = 1,8 \times 10^6$$

$$IC_r = \frac{13,6 \times 16,67 \times 10^6}{6} = 38,3 \times 10^6$$

2. Sabiendo que $MIPS = \frac{\nu_{ck}}{CPI \times 10^6} \Rightarrow$

$$MIPS_r = \frac{16,67 \times 10^6}{10 \times 10^6} = 1,667$$
$$MIPS_e = \frac{16,67 \times 10^6}{6 \times 10^6} = 2,778$$

3. A partir de los datos, queremos averiguar $Y$

$$IC_r = IC_{int} + IC_{fp}$$
$$IC_e = IC_{int} + IC_{fp} \times Y$$
$$IC_{fp} = 195578$$

Despejando con los valores obtenidos en el punto anterior, llegamos a $Y = 187$

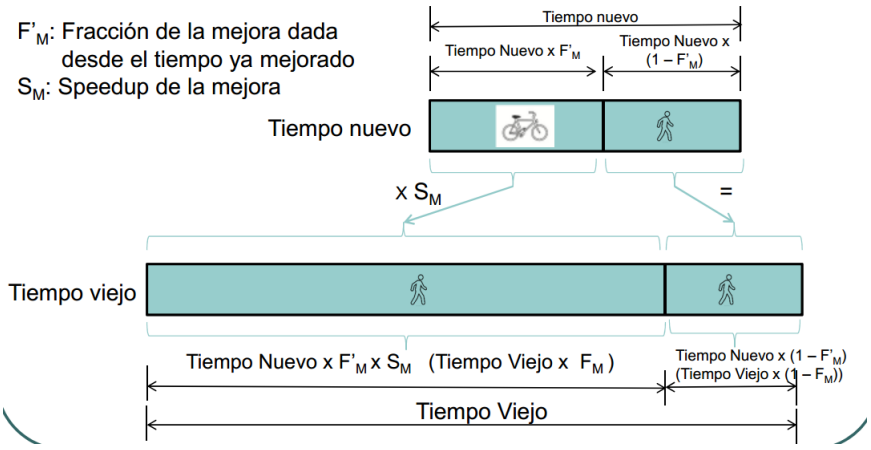
## 1.6. [1.3 $3^{ed}$ CAAQA]

Se realiza una mejora a una computadora a un dado modo de ejecución por un factor de 10. Esta mejora es utilizada el 50 % del tiempo, medido como un porcentaje del tiempo de ejecución cuando la mejora está siendo utilizada. Recordar que no se puede aplicar directamente este 50 % en la Ley de Amdahl para para calcular el speedup.

1. ¿Cuál es el speedup global que se alcanza con esta mejora?
2. ¿Qué porcentaje del tiempo original es empleada esta mejora?

---

Del siguiente gráfico se puede concluir




$$SU = \frac{T_n \times [(1 - F'_m) + F'_m \times SU_m]}{T_n} = 1 - F'm + F'm \times SU_m$$
$$SU = 1 - 0,5 + 0,5 \times 10 = 5,5$$

$$f = \frac{T_m}{T_v} = \frac{F'_m \times T_n \times SU_m}{Tv} = \frac{F'_m \times SU_m}{SU} = \frac{0,5 \times 10}{5,5} = 0,9090$$

## 1.7. [1.6 $3^{ed}$ CAAQA]

Un *benchmark* muy conocido para enteros es Dhrystone. La computadora A realiza $D_A$ ejecuciones del *benchmark* por segundo y realiza millones de instrucciones por segundo ($MIPS_A$). En la computadora B se ejecuta el mismo *benchmark* obteniendo sus propias métricas.

1. ¿Cuál es el problema en calcular la tasa de MIPS de la computadora B como $MIPS_B = MIPS_A \times (D_B/D_A)$?

---

$MIPS_B = MIPS_A \times (D_B/D_A)$ asume que $\frac{MIPS_a}{D_a} = \frac{MIPS_b}{D_b}$ lo cual, en definitiva es $\frac{IC_a}{D_a} = \frac{IC_b}{D_b}$. Esto asume que ambas computadoras tienen el mismo set de instrucciones y ejecutan el mismo programa identicamente compilado, lo cual no ocurre siempre.

## 1.8. [1.17 $3^{ed}$ CAAQA]

Una empresa tiene un *benchmark* que es considerado representativo para sus aplicaciones típicas. Se está considerando un procesador embebido para realizar las tareas pero no cuenta con una unidad de punto flotante y se deben emular por una secuencia de instrucciones de enteros. Este procesador tiene una tasa de 120 MIPS en el *benchmark* .

Un vendedor ofrece un coprocesador para mejorar la performance. Este coprocesador ejecuta cada instrucción de punto flotante por hardware. Cuando se combina el procesador y el coprocesador resulta una tasa de MIPS de 80 en el mismo *benchmark* .

Sea:

- I: Número de instrucciones de enteros ejecutadas en el *benchmark* .
- F: Número de instrucciones de punto flotante ejecutadas en el *benchmark* .
- Y: Número de instrucciones de entero para emular las instrucciones de punto flotante.
- W: Tiempo de ejecución del *benchmark* en el procesador solo.
- B: Tiempo de ejecución del *benchmark* en la combinación procesador/coprocesador.

1. Escribir una expresión para calcular la tasa de MIPS en cada configuración utilizando los símbolos anteriores.
2. Para la configuración sin coprocesador, se obtuvo $F = 8 \times 10^6$, $Y = 50$ y $W = 4$ segundos. Determinar I.
3. ¿Cuál es el valor de B?
4. ¿Cuál es la tasa de MFLOPS para el sistema con coprocesador?
5. Un colega quiere comprar el coprocesador a pesar que tiene una tasa de MIPS inferior cuando se lo utiliza en comparación con el procesador sólo. ¿Tiene razón? Justificar la respuesta.

---

1.

$$MIPS_e = \frac{I + F \times Y}{W \times 10^6}$$

$$MIPS_c = \frac{I + F}{B \times 10^6}$$

2.

$$I = MIPS \times W \times 10^6 - F \times Y = 80 \times 10^6$$

3.
$$B = \frac{I + F}{MIPSe \times 10^6} = 1,1s$$

4. MFLOPS: millones de operaciones de punto flotante por segundo.

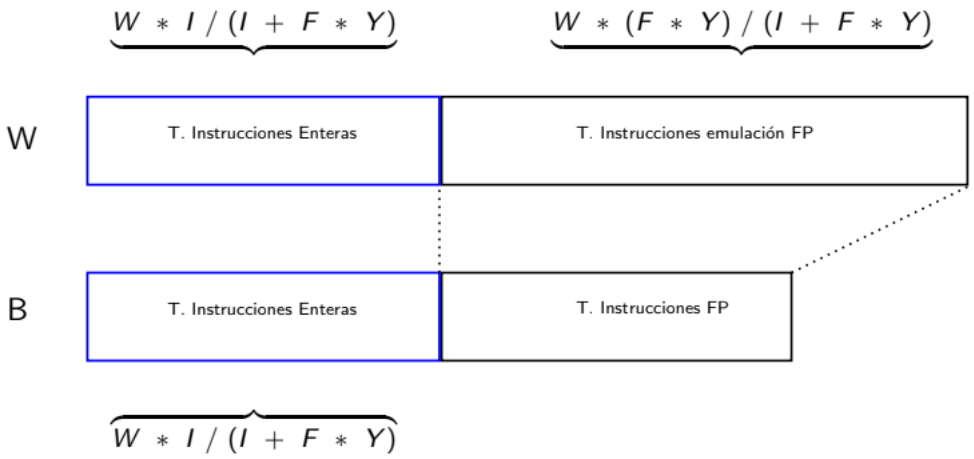
$$MFLOPS = \frac{F}{T_{expf} \times 10^6}$$

El tiempo $W$ indica el tiempo de ejecución total, compuesto por la ejecución de instrucciones enteras y de punto flotante emuladas. El tiempo $T_{exint}$ es igual en ambos casos.

$$W = T_{exfpemul} + T_{exint}$$
$$B = T_{exfp} + T_{exint}$$

Además,

$$T_{ex_int} = \frac{I \times W}{I + F \times Y}$$




$$\Rightarrow T_{expf} = B - T_{exint} = B - \frac{I}{MIPS_c}$$

$$\Rightarrow MFLOPS = \frac{F}{B - I/MIPS_c} = 80$$

5. Si, tiene razón. La métrica que determina el desempeño de un sistema es el tiempo de ejecución. Si comparamos el tiempo de ejecutar el benchmark al utilizar el coprocesador (B) con respecto a simular a través de operaciones enteras (W), el Speed up obtenido es de aproximadamente 3.6 ¿Qué significa que $MIPS_e$ sea mayor que $MIPS_c$? La ejecución en ambos sistemas no es equivalente. La versión que carece del coprocesador debe ejecutar un número extra de instrucciones por cada instrucción de punto flotante (Y). Esta cantidad desproporcionada de instrucciones destinadas a la emulación de punto flotante es lo que causa que $MIPS_e$ sea mayor que $MIPS_c$. En definitiva, la ejecución de un número mucho mayor de instrucciones incrementa el tiempo de ejecución junto con la tasa de MIPS: $IC_e >> IC_c$

## 1.9. Bonus

Se dispone de dos arquitecturas: "A" (con hardware de punto flotante) y "B" (sin hardware de PF), ambas con una frecuencia de reloj de 1200 MHz.

La arquitectura "B" sólo ejecuta instrucciones de enteros (las instrucciones de PF son emuladas exclusivamente con instrucciones de enteros).

Para un programa dado (workload), las frecuencias y ciclos de los diferentes tipos de instrucciones, se muestran a continuación:

| Tipo de instrucción | Frecuencia( %) | Ciclos en 'A' | Ciclos en 'B' |
|---|---|---|---|
| Multiplicación de PF | 15 | 8 | 36 |
| Suma de PF | 20 | 6 | 24 |
| División de PF | 10 | 24 | 60 |
| Instrucciones de Enteros | 55 | 2 | 2 |

1. Calcular la tasa MIPS de ambas máquinas
2. Si 'A' ejecuta 400 millones de instrucciones para este programa, ¿cuántas instrucciones de enteros deberá ejecutar 'B' para el mismo programa?
3. ¿En cuál de las máquinas se ejecuta más rápido el programa dado? Calcular los tiempos de ejecución usando los datos del punto anterior.

---

1. Sabiendo que $MIPS = \frac{\nu}{CPI \times 10^6}$

   Calculamos el $CPI$ para cada arquitectura:

$$CPI = \sum_i f_i \times CPI_i$$

$$CPI_A = 0,15 \times 8 + 0,2 \times 6 + 0,1 \times 24 + 0,55 \times 2 = 5,9$$

$$CPI_B = 0,15 \times 36 + 0,2 \times 24 + 0,1 \times 60 + 0,55 \times 2 = 17,3$$

   Notar que $CPI_B$ es el CPI de la arquitectura 'B' con la posibilidad de emular las instrucciones de punto flotante con instrucciones enteras, pero el CPI real de la arquitectura 'B' es igual a 2 (en la tabla: 2 ciclos por instrucción de enteros).

$$\Rightarrow MIPS_A = \frac{1200 \times 10^6}{5,9} = 203,4$$

$$MIPS_B = \frac{1200 \times 10^6}{17,3} = 69,4$$

$$MIPS_{Breal} = \frac{1200 \times 10^6}{2} = 600$$

   Si comparamos $MIPS_A$ con $MIPS_B$, el primero es mayor y esto tiene sentido considerando que el segundo requiere mayores ciclos para emular las instrucciones de punto flotante con instrucciones enteras. Ahora, si comparamos $MIPS_A$ con $MIPS_{Breal}$ hay que tener en cuenta que el primero sale de una arquitectura con instrucciones más complejas y por eso es mayor el MIPS.

2. Nos piden $IC_{Bint}$ sabiendo que $IC_A = 400 \times 10^6$

$$IC_A = IC_{Apf} + IC_{Aint}$$

Veo cuantos ciclos en 'A' por instrucción voy a tener

Multiplicación de PF: $400 \times 10^6 \times 0,15 = 60 \times 10^6$

Suma de PF: $400 \times 10^6 \times 0,2 = 80 \times 10^6$

División de PF: $400 \times 10^6 \times 0,1 = 40 \times 10^6$

Enteros: $400 \times 10^6 \times 0,55 = 220 \times 10^6$

La cantidad total de instrucciones enteras para 'B' va a ser la suma de las instrucciones enteras que emulan a las de punto flotante más las enteras.

$$IC_{Bint} = 60 \times 10^6 \times 36 + 80 \times 10^6 \times 24 + 40 \times 10^6 \times 60 + 220 \times 10^6 \times 2 = 6920 \times 10^6$$

Pero cada instrucción de enteros toma dos ciclos por instrucción. Entonces, dividido por 2 queda $IC_{Bint} = 3460 \times 10^6$

3.

$$T_{ex} = \frac{CPI \times IC}{\nu} \Rightarrow$$

$$T_{exA} = \frac{5,9 \times 400 \times 10^6}{1200 \times 10^6} = 1,97s$$

$$T_{exBreal} = \frac{2 \times 3460 \times 10^6}{1200 \times 10^6} = 5,77s$$

$$T_{exB} = \frac{17,3 \times 400 \times 10^6}{1200 \times 10^6} = 5,77s$$

# 2. Arquitectura de Conjunto de Instrucciones (ISA)

Cuadro 1: *Mix. de instrucciones*

| Instrucción | Frecuencia |
|---|---|
| *Load* | 26 % |
| *Store* | 10 % |
| Aritméticas | 14 % |
| Comparaciones | 13 % |
| Saltos condicionales | 16 % |
| Saltos incondicionales | 1 % |
| *Call / Return* | 2 % |
| Shift | 4 % |
| Logicas | 4 % |
| Misc. | 5 % |

## 2.1.

Considerar el agregado de un nuevo modo de acceso a MIPS. El nuevo modo suma dos registros y un valor de offset de 11 bits con signo para obtener la dirección efectiva. Utilizar el porcentaje de instrucciones indicado en el Cuadro 1. El compilador pasa de las instrucciones:

```
add R1, R1, R2
lw Rd, 100(R1) #(o store)
```

A utilizar:

```
lw Rd, 100(R1,R2)
```

1. Asumir que el nuevo modo de acceso es usado el 10 % de los loads y stores. ¿Cuál es el porcentaje de IC nuevo comparado con la tasa original?
2. Si el nuevo modo de direccionamiento aumenta en 5 % el tiempo de clock, ¿Cuál computadora será más rápida y por cuanto?
3. Describa una situación que limite el porcentaje de reemplazos realizados.

---

1. Del Cuadro 1 sabemos que el 36 % son instrucciones de tipo Load/Store, entonces el nuevo modo de direccionamiento podrá usarse en el 3,6 % de instrucciones. Calculo el ratio sacando la diferencia de instrucciones con el nuevo modo de direccionamiento dividido el original. El nuevo modo de direccionamiento va a tener la cantidad original menos la cantidad que nos ahorramos con la mejora y para eso usamos el dato del porcentaje de instrucciones donde la podemos aplicar.

$$IC_{mejorado} = \frac{IC_{original} - (IC \times 0,036)}{ICoriginal} = (1 - 0,036) \times IC_{original} = 0,964 \times IC_{original}$$

   Donde 0,964 es el ratio pedido.

2. Sabemos:

$$T_{ckViejo}$$
$$T_{ckNuevo} = 0,05 \times T_{ckViejo} + T_{ckViejo}$$
$$\Rightarrow T_{exViejo} = \frac{CPI_v \times IC_v \times T_{ckViejo}}{CPI_n \times IC_n \times T_{ckNuevo}} = \frac{1}{0,964 \times 1,05} = 0,988$$

   Esto quiere decir que $SpeedUp = 0,988 < 1$, o sea que es más lenta!

## 2.2.

Al diseñar un sistema de memoria, es importante contrastar la frecuencia de accesos de lectura y accesos de escritura, así como accesos para leer instrucciones y accesos para datos. Utilizando el Cuadro 1, indicar:

1. Porcentaje de accesos a memoria que son accesos a datos.
2. Porcentaje de accesos de datos que son lecturas
3. Porcentaje de accesos a memoria que son lecturas

---

1. 36 %
2. 26 %
3. not sure..

   .ª memory access happens when the CPU loads bits from main memory or stores bits to main memory. The CPU will first check to see if it can load the bits from a caché, in which case it will skip the memory access completely. One kind of memory access is a data memory access, when the CPU runs an instruction that explicitly loads or stores data from/to memory.º sea son accesos a datos (lw/sw) + instrucciones ejecutadas

## 2.3.

Calcular el CPI Efectivo para MIPS utilizando el Cuadro 1. Asumir que se obtuvieron los siguientes CPI para instrucciones. Asumir que el 60 % de los saltos son tomados.

| Instrucción | Ciclos |
|---|---|
| ALU | 1,0 |
| *Load/Store* | 1,4 |
| Saltos condicionales | |
| Tomados | 2,0 |
| No Tomados | 1,5 |
| Saltos | 1,2 |

Primero identifico qué tipo de instrucción es cada una de las del Cuadro 1.

| Instrucción | Familia de instrucciones |
|---|---|
| Load | Load/Store |
| Store | Load/Store |
| Aritméticas | ALU |
| Comparaciones | ALU |
| Saltos condicionales | Saltos condicionales |
| Saltos incondicionales | Saltos |
| Call / Return | Saltos |
| Shift | ALU |
| Logicas | ALU |
| Misc | ALU |

Ahora junto toda la información en una nueva tabla

| Tipo de instrucción | Frecuencia ( %) | Ciclos |
|---|---|---|
| Load/Store | 36 | 1,1 |
| ALU | 40 | 1,0 |
| Saltos condicionales Tomados | 9,6 | 2,0 |
| Saltos condicionales No Tomados | 6,4 | 1,5 |
| Saltos | 3 | 1,2 |

El CPI efectivo es aquel basado en un workload determinado con ciertas instrucciones y frecuencias. Es el promedio ponderado:

$$CPI_{ef} = 0,36 \times 1,1 + 0,4 \times 1 + 0,096 \times 2 + 0,064 \times 1,5 + 0,03 \times 1,2 = 1,12$$

## 2.4.

Considere dos implementaciones M1 y M2 del mismo ISA. Estamos interesados en la performance de dos programas P1 y P2, que tienen la siguiente mezcla de instrucciones:

| Operación | *P*1 | *P*2 |
|---|---|---|
| *Load/Store* | 40 % | 50 % |
| ALU | 50 % | 20 % |
| Saltos | 10 % | 30 % |

El CPI para cada computadora es:

| Operación | $M1$ | $M2$ |
|---|---|---|
| Load/Store | 2 | 2 |
| ALU | 1 | 2 |
| Saltos | 3 | 2 |

1. Asuma que la frecuencia del reloj de M1 es 2GHz. ¿Cuál debería ser la frecuencia del reloj de M2 para que ambas computadoras reporten el mismo Tiempo de ejecución para P1?
2. Asuma que ambas computadoras tienen la misma frecuencia de reloj, y que P1 y P2 ejecutan la misma cantidad de instrucciones. ¿Qué computadora es mas rápida para un conjunto de trabajo que consiste de igual número de ejecuciones de P1 y P2?
3. Determine un conjunto de trabajo tal que M1 y M2 tengan la misma performance cuando tienen la misma frecuencia de reloj.

---

1. $$\nu_1 = 2GHz$$
$$T_{ex} = CPI \times IC \times T_{ck} \Rightarrow$$
$$CPI_1 = 2 \times 0,4 + 0,5 + 3 \times 0,1 = 1,6$$
$$CPI_2 = 2 \times 0,4 + 2 \times 0,5 + 2 \times 0,1 = 2$$

   Luego, igualo los tiempos de ejecución para ambos

$$\frac{CPI_1 \times IC_1}{\nu_1} = \frac{CPI_2 \times IC_2}{\nu_2}$$

   Como son implementaciones del mismo ISA, $IC_1 = IC_2 \Rightarrow$

$$\nu_2 = \frac{\nu_1}{CPI_1} = 2,5GHz$$

2. Calculo los CPIs para ambos programas

   Para P1:
$$CPI_1 = 1,6$$
$$CPI_2 = 2$$

   Para P2:
$$CPI_1 = 2 \times 0,5 + 0,2 + 3 \times 0,3 = 2,1$$
$$CPI_2 = 2 \times 0,5 + 2 \times 0,2 + 2 \times 0,3 = 2$$

   Como tienen misma cantidad de instrucciones (IC) y misma frecuencia de reloj, el CPI define cuál es más rápida. Entonces, para P1 va a ser M1, pero para P2 va a ser M2. Para un conjunto de trabajo donde se ejecutan P1 y P2, va a ser más rápida M1 (la suma de sus cpi's es menor).

3. Si ejecuto P2 cuatro veces y P1 una sola, por ejemplo.

## 2.5. [2.6 3ed CAAQA]

Varios investigadores han sugerido que incorporar un modo de direccionamiento registro-memoria a una máquina Load/Store puede ser conveniente. La idea es reemplazar la siguiente secuencia

```
LOAD R1, 0(Rb)
ADD R2, R2, R1
```

por

```
ADD R2, 0(Rb)
```

Asumir que la nueva intrucción provoca un incremento en el ciclo de clock del 5 %. Utilizar la frecuencia de instrucciones del Cuadro 1. La nueva instrucción afecta únicamente al ciclo de clock y no al CPI.

1. ¿Qué porcentaje de loads deben ser eliminados en la máquina con la nueva instrucción para tener al menos el mismo rendimiento?
2. Mostrar una secuencia de intrucciones donde un load de R1 seguido inmediatamente por el uso de R1 (con algún tipo de opcode) no podría ser reemplazado por una única instrucción como la propuesta, asumir que el mismo opcode existe.

---

1. El mismo rendimiento significa $T_{exV} = T_{exN} \Rightarrow$

$$CPI_v \times IC_v \times T_v = CPI_n \times IC_n \times T_n$$

Como no afecta al CPI, puedo simplificar

$$IC_v \times T_v = IC_n \times 1,05 \times T_v \Rightarrow$$
$$\frac{1}{1,05} \times IC_v = IC_n \Rightarrow$$
$$IC_n = 0,952 \times IC_v$$

Esta es la relación entre cantidad de instrucciones para todo el set de instrucciones. Me piden el porcentaje de loads que deben ser eliminados, entonces si decimos que se redujo un 5 % la cantidad de instrucciones total. Sabemos que:

$$IC_{lost} = 0,05 \times IC_{original}$$

Además, del cuadro 1:

$$IC_{ldOriginal} = 0,26 \times IC_{original}$$

Por lo tanto, juntamos ambas:

$$IC_{ldLost} = 0,05 \times \frac{IC_{ldOriginal}}{0,26} = 0,19 \times IC_{ldOriginal}$$

Entonces se tiene que eliminar aproximadamente el 19 % de instrucciones load.

2. Consideremos los valores R1=47, Rb=1000 y mem[1000]=4. Ejecutando la primera versión el resultado va a ser 4+4=8. Ejecutando la segunda versión el resultado va a ser 47+4=51.

   Otra secuencia que tampoco se podría reemplazar por una instrucción:

```
ld R1, 0(R2) #R1 <- Mem(R2)
ld R3, 0(R1) #R3 <- Mem(R1)
```

   Esto intentaría hacer R3 <- Mem(Mem(R2)), pero es imposible reemplazar en una instrucción porque necesito las direcciones de R1 y R2.

## 2.6. [2.8 $3^{ed}CAAQA$]

Para el siguiente código C:

```
for (i=0; i <= 100; i++) {
    A[i] = B[i] + C
}
```

Suponer que A y B son arreglos de enteros de 64 bits y C es un entero de 64 bits. Suponer que todos los valores para datos y direcciones se encuentran en memoria (en las direcciones 0, 5000, 1500 y 2000 para A, B, C e i respectivamente) excepto cuando se operan sobre ellos. Suponer que los valores de los registros se pierden entre iteraciones del bucle.

1. Escribir el código MIPS correspondiente
2. Cuántas instrucciones son requeridas dinámicamente?
3. Cuántas referencias a memoria para datos son ejecutadas?
4. Cuál es el tamaño del código en bytes?

---

1.
```
        sw zero, 2000(fp)   # guardo 0 en pos 2000 (i)
        lw R0, 2000(fp)     # i -> R0
    loop:
        lw R1, 100          # 100 -> R1
        lw R0, 2000(fp)     # i -> R0
        bgtu R0, R1, FIN    # salgo si i>100
        lw R1, 1500(fp)     # C -> R1
        sll R0, R0, 3       # escalo x8 bytes (c/entero son 64 bits)
        ld R2, 5000(fp)     # B -> R2
        addu R2, R2, R0     # me muevo a B[i]
        ld R2, 0(R2)        # B[i] -> R2
        addu R2, R2, R1     # B[i] + C -> R2
        ld R3, 0(fp)        # A -> R3
        addu R3, R3, R0     # me muevo a A[i]
        sw R2, 0(R3)        # A[i] = B[i] + C
        addiu R0, R0, 1     # i++
        sw R0, 2000(fp)     # lo guardo porque se pierde entre iteraciones segun enunciado
    FIN:
```

2. Instrucciones requeridas dinámicamente: las que ejecuta el procesador.

$$2(primerasantesdelloop) + 15(instr.loop) \times 101(vecesqueseejecutaloop) = 1517$$

3. Referencias a memoria para datos: lw / sw

$$2 + 101 \times 7 = 709$$

4. Tamaño del código en bytes: cantidad de instrucciones por 4 (cada instrucción tiene 4 bytes)

$$17 \times 4B = 68B$$

## 2.7.

## 2.8.

El diseño de MIPS brinda 32 registros de propósito general y 32 registros de punto flotante. Si los registros son ”buenos”, ¿tener más registros es mejor?. Armar una lista y discutir todos los trade-off posibles que se le ocurra considerando el cuando y el cuanto para incremento de registros con el criterio de los diseñadores de ISA en mente. ———Tener más registros es ganar velocidad porque están más cerca del CPU y, por lo tanto, se tarda menos en acceder. Sin embargo, ocupan más espacio. Agrandan área de chip y aumenta uso de potencia, y las intrucciones son más largas (más bits).

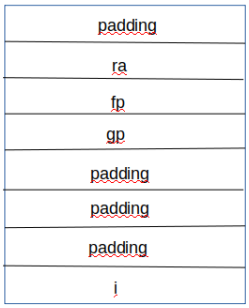
## 2.9.

## 2.10.

Codificar en Assembly MIPS y diagramar el stack de las siguientes funciones.

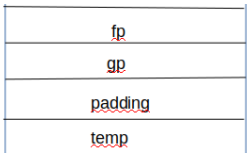
```
void proc ( int i ) {
    int j ;
    j = i +20;
}
int main ( int argc , char ** argv )
{
    const int i =10;
    proc (i) ;
    return 0;
}
```

———




```
.text
.ent main
.globl main
main:
    .frame fp, 28, ra
    subu sp, sp, 28
    .cprestore 16
    sw ra, 24(sp)
    sw fp, 20(sp)
    move fp, sp
    sw a0, 32(fp)
    sw a1, 36(fp)
    li a0, 10
    jal proc
```

```
    li v0, 0
fin:
    lw ra, 24(sp)
    lw fp, 20(sp)
    addu sp, sp, 28
    jr ra
.end main
```

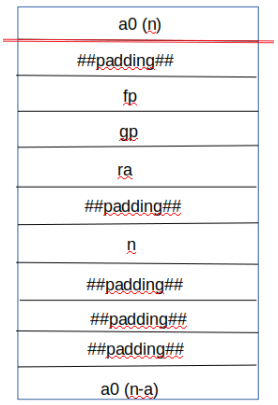



```
proc:
    subu sp, sp, 16
    sw fp, 12(sp)
    sw gp, 8(sp)
    sw a0, 16(sp)
    move fp, sp
    lw t0, 16(fp)
    addiu t1, t0, 20
fin:
    lw fp, 12(sp)
    lw gp, 8(sp)
    addu sp, sp, 16
    jr ra
```

## 2.11.

Codificar en Assembly MIPS y diagramar el stack de la siguiente función.

```
unsigned int
factorial ( unsigned int n )
{
    if ( n < 2)
        return 1;
    else
        return n * factorial (n -1) ;
}
```



```
.text
.ent factorial
.globl factorial
factorial:
    .frame fp, 40, ra
    subu sp, sp, 40
    .cprestore 28
    sw fp, 32(sp)
    sw ra, 24(sp)
    move fp, sp
    sw a0, 40(fp)
    sw a0, 16(fp)
    li t0, 2
    lw t1, 16(fp)
    bltu t1, t0, ret1
    li t0, 1
    subu a0, t1, t0
    sw a0, 16(fp)
    jal factorial
    lw t0, 40(fp)
    mul v0, v0, t0
    b fin
ret1:
    li v0, 1
fin:
    lw ra, 24(sp)
    lw fp, 32(sp)
    addu sp, sp, 40
    jr ra
.end factorial
```

## 2.12.

Codificar en C y Assembly MIPS una función que reciba calcule la longitud de un C-string

---

```
size_t strlen (const char *s)
{
    size_t len = 0;
    while (*s != 0)
    {
        len++;
        s++;
    }
    return len;
}
```

Stack frame:
16: const char *s [justo por encima del stack frame]
12: fp
8: gp
4: s
0: len

```
    #toda la parte de crear stack frame..
```

```
    sw zero, 0(fp)      # len = 0
loop:
    lw t0, 0(fp)        # len -> t0
    lw t1, 4(fp)        # s -> t1
    lb t2, 0(t1)        # *s -> t2
    beqz t2, fin
    lw t0, 0(fp)
    addiu t0, t0, 1     # len++
    sw t0, 0(fp)
    lw t1, 8(fp)
    addiu t1, t1, 1     #s++
    sw t1, 8(fp)
    b loop
fin:
    lw v0, 0(fp)
    #destruir stack frame
```