TP N° 7 – Programación en lenguaje ensamblador MIPS con MIPSX

Facultad de Informática

Objetivo: Comprender la estructura de un programa en lenguaje ensamblador MIPS junto con la convención de llamada a procedimientos y funciones. Comprender el funcionamiento general de una organización de memoria caché y su implicancia.

Recursos y Bibliografía:

Arq. MIPS Vol I, II and III.
Programa mipsx.
Apunte MIPS, sección 6.
Caché: Patterson y Hennessy. Computer Organization and Design. The hw/sw interface.

Ejercicio 0. Considere el siguiente programa:

```
unsigned char  m0[32];
unsigned char m1[32];
unsigned char m2[32];

main()
{
        int i;
        int res;
        cargar(m0);
        cargar(m1);
        cargar(m2);

        for (i=0; i<32; i++) {
                res = res + m0[i] + m2[i];
        }
}
```

El segmento de datos se carga en la dirección 0x420000 de memoria principal. unsigned char ocupa un byte.

a. El procesador MIPS cuenta con una caché de mapeo directo de 32 entradas, y en cada línea de caché se almacena una palabra. El acceso a memoria principal demora 60ns, y el tiempo de acceso a caché 2ns. Sin tener en cuenta las funciones cargar(), ¿Cuál es el tiempo de acceso medio para este programa y procesador?.
b. Ahora se cuenta con un procesador MIPS que contiene una caché de mapeo directo de 8 entradas, y en cada entrada de caché se almacenan 2 palabras. El acceso a memoria principal demora 60ns, y el tiempo de acceso a caché 2ns. Indique cuál es el tiempo de acceso medio para este programa.
c. ¿Qué sucede con la tasa de fallos y el tiempo medio si la caché fuese asociativa por conjuntos, de dos vías? (en cada vía la caché es idéntica a b.).

Ejercicio 1. a. ¿Qué fallos comete el siguiente código de función (o subrutina)[1] **[*]** en cuanto a la convención de uso de registros y llamada a procedimiento y funciones?

```
fun:
      addiu   $sp,$sp,-32
      sw    $ra,28($sp)
      sw    $fp,24($sp)
      move  $fp,$sp
      li    $s0,1
      li    $t1,2
      jal   fun
      addi  $k0, $v0, 3
      add   $v0, $t1, $k0
      add   $v0, $s1, $k0
      move  $sp,$fp
      lw    $ra,28($sp)
      lw    $fp,24($sp)
      addiu   $sp,$sp,32
      jr    $ra
```

b. ¿Qué problemas pueden suceder si un compilador mal desarrollado (o un programador) no utiliza la convención de llamadas a procedimientos?. Mencione tres problemas que pueden surgir.

Ejercicio 2. Traduzca el programa en C, que se encuentra debajo, a un programa en lenguaje ensamblador MIPS, como lo haría un compilador. Respete la convención de llamada a procedimientos. Considere que char ocupa un byte e int una palabra. Verifique su correcto funcionamiento en *mipsx*.

a. ¿por qué las variables **x** y **n** de pow no son las mismas que las variables globales **x** y **n** definidas antes de **main**?
b. Indique cuales son las direcciones efectivas de **x** y **n** global, y de las variables **i** y **pow** en **pow()**.
c. ¿Cuál es la dirección del último byte del segmento de texto? ¿Cuántas pseudo instrucciones tiene el programa?
d. Indique las direcciones de inicio de los segmentos de pila, segmento de texto y segmento de datos del programa.

```
int pow(char x, char n)
{
        int i;
        int pow = 1;

        for (i=0; i<n; i++)
                pow = pow * x;

        return pow;
}


char x = 30;
char n = 55;
int resultado = 0;

void main(void)
{
```

[1] [*] Subrutina o función: https://en.wikipedia.org/wiki/Function_(computer_programming)

```
        resultado = pow(x, n);
}
```

Ejercicio 3. Desarrollar una función llamada strlen que reciba como argumento la dirección de una cadena de texto (terminada con el caracter nulo) y devuelva la longitud de la cadena. Respete la convención de llamada a procedimiento. Verifique el funcionamiento de su subrutina desarrollada llamándola desde main en mipsx.

Ejercicio 4. Desarrollar una función que calcule el factorial de un número, utilizando la definición recursiva, y respetando la convención de llamada a procedimiento.

```
factorial(0) = 1
factorial(N) = N * factorial(N-1)
```