

TP 1: Conjunto de instrucciones MIPS

Martín Queija, *Padrón Nro. 96.455*
tinqueija@gmail.com

Estanislao Ledesma, *Padrón Nro. 96.622*
estanislaomledesma@gmail.com

Agustin Luques, *Padrón Nro. 96.803*
agus.luques@hotmail.com

2do. Cuatrimestre de 2016
66.20 Organización de Computadoras – Práctica Martes
Facultad de Ingeniería, Universidad de Buenos Aires

November 1, 2016

\AM@currentdocname .png

.png

\AM@currentdocname .png

.png

\AM@currentdocname .png

.png

1 Diseño e implementación

La función `mips32_plot` implementada con código mips utiliza un stack frame de tamaño variable, ya que esta hace uso de un buffer de almacenamiento en LTA para los syscall de impresión en el archivo, cuyo tamaño se setea previo a la compilación (mediante `define`). El mismo está compuesto por las cuatro áreas características: SRA (saved register area), la cual almacena los datos de los registros `s`, `fp`, `sp`, `ra` y `gp`; FRA (float register area), la cual tiene espacio de almacenamiento para los registros de punto flotante utilizados en el programa (principalmente para los ciclos); LTA (local and temporary area), con espacio para todos los componentes del parámetro de la función (`parms`, cuyo puntero se haya en la ABA de la caller), para `BUF_PTR` (tamaño del buffer para `flush`), `RA_SAVE` buffer `flush` (para guardar `ra` antes de `flush`), `a0` previo al `flush` y espacios para algunos registros temporales (`t`) y `s`, cuando se llama a alguna función auxiliar. También posee el buffer, de tamaño `BUF_SZ`, con lo cual, a partir de esta posición, todas las posiciones del stack se indican con constantes definidas. Por último, también posee el ABA (argument building area) con el espacio típico: cuatro palabras (una por cada registro `a`). El diagrama del mismo se encuentra en la tabla 1.

También se implementaron cuatro pseudo-funciones, `print-headers` (para imprimir los encabezados), `shade_a_buffer` (para escribir el `shade` en el buffer), `flush_buffer` (para `flush`ear todo lo escrito en el buffer) e `inttostring` (para obtener el string de un entero a partir de este último). Se trata de pseudo-funciones ya que no se reservó espacio para stack frames de cada una, sino que se trabajó con registros directamente que escriben en el buffer alocado de la función principal `mips32_plot`. Para acceder a estas, siempre primero se salva los registros en la LTA de la `mips32_plot`, `ra`, `a0`, y todos aquellos que sean sobrescritos en la función llamada, que luego deban ser utilizados.

2 Compilación

Para compilar la función implementada en Mips, se utilizó `Gxemul` para emular un procesador MIPS R3000 con una imagen de sistema operativo NETBSD 3.0 con plataforma `pmax`. Este se ejecutó con soporte X11, mediante la generación de un túnel entre el HostOS y el GuestOS con SSH/SSHD. La transferencia de archivos (los necesarios para compilar el `tp`) se hizo con la utilidad SCP. Para compilar `mips32_plot.S` junto con el `main.c` y demás archivos necesarios se utilizó el `makefile` provisto por la cátedra. Para esto se emiten los comandos “`make clean`”, “`make makefiles`” y “`make`” en ese orden, obteniendo así el ejecutable. El archivo de salida se puede enviar al sistema operativo HostOS mediante SCP nuevamente.

3 Código MIPS

```
#include <mips/regdef.h>
#include <sys/syscall.h>

#ifdef BUF_SZ
```

```

#define BUF_SZ 8192
#endif

#define BUF_OFFSET 100 + BUF_SZ
#define BUF_LOW_OFFSET 104

#define FP0_OFFSET BUF_OFFSET + 4
#define FP1_OFFSET FP0_OFFSET + 4
#define FP2_OFFSET FP1_OFFSET + 4
#define FP3_OFFSET FP2_OFFSET + 4
#define FP4_OFFSET FP3_OFFSET + 4
#define FP5_OFFSET FP4_OFFSET + 4
#define FP6_OFFSET FP5_OFFSET + 4
#define FP7_OFFSET FP6_OFFSET + 4
#define FP8_OFFSET FP7_OFFSET + 4
#define FP9_OFFSET FP8_OFFSET + 4
#define FP10_OFFSET FP9_OFFSET + 4
#define FP11_OFFSET FP10_OFFSET + 4

#define S0_OFFSET FP11_OFFSET + 4
#define S1_OFFSET S0_OFFSET + 4
#define S2_OFFSET S1_OFFSET + 4
#define S3_OFFSET S2_OFFSET + 4
#define S4_OFFSET S3_OFFSET + 4
#define S5_OFFSET S4_OFFSET + 4
#define S6_OFFSET S5_OFFSET + 4
#define S7_OFFSET S6_OFFSET + 4

#define SP_OFFSET S7_OFFSET + 4
#define GP_OFFSET SP_OFFSET + 4
#define FP_OFFSET GP_OFFSET + 4
#define RA_OFFSET FP_OFFSET + 4
#define STACK_SZ RA_OFFSET + 4

        .text
        .abicalls
        .align 2
        .globl mips32_plot
        .ent    mips32_plot

mips32_plot:
        .frame   $fp, STACK_SZ, ra

        .set     noreorder
        .cpld    t9
        .set     reorder

```

```

subu sp, sp, STACK_SZ                                # 4(SRA) + 15 + BUF_SZ (LTA) +

.cprestore GP_OFFSET                                #guardo gp en 80 + sp
sw $fp, FP_OFFSET(sp)                               #guardo $fp en 84 + sp
sw ra, RA_OFFSET(sp)                                #guardo ra en 88 + sp

move $fp, sp                                          # $fp = sp (desde aca se usa $fp)

sw a0, STACK_SZ($fp)                                #guardo el parametro(*parms) en

#
# Guardo la data de *params en el stack local.
#

lw t0, 0(a0)
sw t0, 16($fp)                                       #guardo UL_Re en 16 + $fp

lw t0, 4(a0)
sw t0, 20($fp)                                       #guardo UL_Im en 20 + $fp

lw t0, 8(a0)
sw t0, 24($fp)                                       #guardo LR_Re en 24 + $fp

lw t0, 12(a0)
sw t0, 28($fp)                                       #guardo LR_Im en 28 + $fp

lw t0, 16(a0)
sw t0, 32($fp)                                       #guardo D_Re en 32 + $fp

lw t0, 20(a0)
sw t0, 36($fp)                                       #guardo D_Im en 36 + $fp

lw t0, 24(a0)
sw t0, 40($fp)                                       #guardo CP_Re en 40 + $fp

lw t0, 28(a0)
sw t0, 44($fp)                                       #guardo CP_Im en 44 + $fp

lw t0, 32(a0)
sw t0, 48($fp)                                       #guardo X_RES en 48 + $fp

lw t0, 36(a0)
sw t0, 52($fp)                                       #guardo Y_RES en 52 + $fp

lw t0, 40(a0)
sw t0, 56($fp)                                       #guardo SHADES en 56 + $fp

lw t0, 44(a0)

```

```

sw t0 , 60($fp)                                #guardo FILE POINTER en

#
# Guardo los registros flotantes y caller saved registers que voy a usar
#
error:                                           #Si vuelvo de error restauro todo, en v
s.s $f0 , FP0-OFFSET($fp)
s.s $f1 , FP1-OFFSET($fp)
s.s $f2 , FP2-OFFSET($fp)
s.s $f3 , FP3-OFFSET($fp)
s.s $f4 , FP4-OFFSET($fp)
s.s $f5 , FP5-OFFSET($fp)
s.s $f6 , FP6-OFFSET($fp)
s.s $f7 , FP7-OFFSET($fp)
s.s $f8 , FP9-OFFSET($fp)
s.s $f9 , FP9-OFFSET($fp)
s.s $f10 , FP10-OFFSET($fp)
s.s $f11 , FP11-OFFSET($fp)

sw s0 , S0-OFFSET($fp)
sw s1 , S1-OFFSET($fp)
sw s2 , S2-OFFSET($fp)
sw s3 , S3-OFFSET($fp)
sw s4 , S4-OFFSET($fp)
sw s5 , S5-OFFSET($fp)
sw s6 , S6-OFFSET($fp)
sw s7 , S7-OFFSET($fp)

l.s $f6 , 40($fp) #f6: cpr = parms->cp_re
l.s $f7 , 44($fp) #f7: cpi = parms->cp_im

sw zero , 64($fp)                                #inicializo el BUF_PTR

#####
#####

jal print_header

#Inicializaciones para

addu s1 , zero , zero #s1: y = 0
l.s $f0 , 20($fp) #f0: ci = parms->UL_Im
lw t1 , 52($fp) #t1: parms-> y_res

for1:
bge s1 , t1 , fin #si y >= y_res => fin

```



```

#Inicializaciones para
addu s2, zero, zero
l.s $f1, 16($fp)
lw t2, 48($fp)
#s2: x = 0
#f1: cr = parms->UL_Re
#t2: parms-> x_res

for2:
    bge s2, t2, fin_for1
    mov.s $f2, $f1
    mov.s $f3, $f0
#si x >= x_res => fin_f
#f2: zr = cr
#f3: zi = ci

#Inicializaciones para
addu s3, zero, zero
lw t3, 56($fp)
#s3: c = 0
#t3: parms->shades

for3:
    bge s3, t3, fin_for2
    mul.s $f4, $f2, $f2
    mul.s $f5, $f3, $f3
    add.s $f4, $f4, $f5
    l.s $f5, CTEFOR3
    c.lt.s $f5, $f4
#si CTEFOR3 < absz => code = 1
    bc1t fin_for2
    mul.s $f4, $f2, $f2
    mul.s $f5, $f3, $f3
    sub.s $f4, $f4, $f5
    add.s $f4, $f4, $f6
    mul.s $f5, $f2, $f3
    l.s $f8, CTEMULT
    mul.s $f5, $f5, $f8
    add.s $f5, $f5, $f7
    mov.s $f2, $f4
    mov.s $f3, $f5
    addi s3, s3, 1
    b for3
#si code = 1 =>
#f4: zr * zr
#f5: zi * zi
#f4: zr * zr - zi * zi
#f4: sr = zr * zr - zi
#f5: zr * zi
#f8: CTEMULT
#f5: 2 * zr * zi
#f5: si = 2 * zr * zi +
#f2: zr = sr
#f3: zi = si
#s3: ++c

fin_for2:

sw s1, 80($fp)
sw s2, 84($fp)
sw s3, 88($fp) #### ESTOS SON LOS REGISTROS QUE USA EL CICLO
sw t1, 92($fp) ####tienen lugares especiales en el stack si necesitas gu
sw t2, 96($fp)
sw t3, 100($fp)

move a0, s3
jal shade_a_buffer

lw s1, 80($fp)

```

```

    lw s2, 84($fp)
    lw s3, 88($fp)
    lw t1, 92($fp)   ### REPONGO LOS REGISTROS QUE USA EL CICLO
    lw t2, 96($fp)
    lw t3, 100($fp)

    addi s2, s2, 1
    l.s $f8, 32($fp)
    add.s $f1, $f1, $f8
    b for2
    #s2: ++x
    #f8: parms->d_re
    #f1: cr += parms->d_re
    #vuelve a for2

fin_for1:
    addi s1, s1, 1
    l.s $f8, 36($fp)
    sub.s $f0, $f0, $f8
    b for1
    #s1: ++y
    #f8: parms->d_im
    #f0: ci -= parms->d_im
    #vuelve a for1

fin:
#Antes de terminar, flusheo lo que queda en el buffer
    sw ra, 68($fp)
    sw a0, 72($fp)
    lw a2, 64($fp)
    jal flush_buffer
    lw ra, 68($fp)
    lw a0, 72($fp)
    #Establezco en a2 la ca

#Reestablezco los registros flotantes y caller saved registers a sus va

    l.s $f0, FP0_OFFSET($fp)
    l.s $f1, FP1_OFFSET($fp)
    l.s $f2, FP2_OFFSET($fp)
    l.s $f3, FP3_OFFSET($fp)
    l.s $f4, FP4_OFFSET($fp)
    l.s $f5, FP5_OFFSET($fp)
    l.s $f6, FP6_OFFSET($fp)
    l.s $f7, FP7_OFFSET($fp)
    l.s $f8, FP9_OFFSET($fp)
    l.s $f9, FP9_OFFSET($fp)
    l.s $f10, FP10_OFFSET($fp)
    l.s $f11, FP11_OFFSET($fp)

    lw s0, S0_OFFSET($fp)
    lw s1, S1_OFFSET($fp)
    lw s2, S2_OFFSET($fp)
    lw s3, S3_OFFSET($fp)
    lw s4, S4_OFFSET($fp)

```

```

        lw s5, S5_OFFSET($fp)
        lw s6, S6_OFFSET($fp)
        lw s7, S7_OFFSET($fp)

        move sp, $fp                                #restauro valor de sp
        lw ra, RA_OFFSET(sp)                        #restauro valor de ra
        lw $fp, FP_OFFSET(sp)                      #restauro valor de $fp

        addu sp, sp, STACK_SZ                        #subo stack pointer

        jr ra                                        #vuelvo a la funcion caller

#####

#####

#Int to String: el int va en a0, cantidad de chars guardados en v0

inttostr:
        addi t0, zero, 10                          #establezco divisor base 10
        move t1, zero                               #t1=0
        la t2, stringbuffer                        #t2=stringbufferadress
        add t3, a0, zero                            #t3=a0 (int a imprimir)
loop1:   div t3, t0                                  #t3/10 division:t4 resto:t5
        mflo t4
        mfhi t5
        addi t1, t1, 1
        bne t3, 0, loop1
        add t3, a0, zero                            #t3=a0 (int a imprimir)
        move v0, t1                                 #v0=cant chars guardados
loop2:   div t3, t0                                  #vuelvo a dividir esta vez guardando en e
        mflo t4
        mfhi t5
        addi t1, t1, -1
        add t6, t2, t1
        addi t5, t5, 48
        sb t5, 0(t6)
        bne t3, 0, loop2
        jr ra

#####

#Shade a buffer: Escribe el int en a0 al buffer, si el buffer se llena se flush

shade_a_buffer:
        addi s0, $fp, BUF_LOW_OFFSET               #s0 = BUFFERLOW
        lw s1, 64($fp)                             #Cargo a s1 el BUF_PTR
        add s5, s0, s1                             #s5 = BUFFERLOW + BUF_PTR

        sw ra, 68($fp)

```

	jal inttostr	
	lw ra, 68(\$fp)	
	move s2, v0	
	move s6, zero	#s6 = 0
	la s7, stringbuffer	#s7 = stringbufferaddress
loop:	add s4, s7, s6	#s4 = stringbufferaddress + s6
	addi s6, s6, 1	#s6 = s6 + 1
	lb t2, 0(s4)	#guardo el primer byte del int de dir(s
	sb t2, 0(s5)	#lo guardo en s5
	addi s1, s1, 1	#incremento BUF_PTR
	addi s5, s5, 1	#incremento el puntero al buffer
	bne s1, BUF_SZ, next	#si BUF_PTR no es igual a BUF_SZ NO flu
	sw ra, 68(\$fp)	
	sw a0, 72(\$fp)	
	li a2, BUF_SZ	#Establezco la cantidad a flushear (tod
	jal flush_buffer	
	lw ra, 68(\$fp)	
	lw a0, 72(\$fp)	
	addi s1, zero, 0	#flushee, entonces reseteo el buffptr
	add s5, s0, zero	#reseteo el puntero al buffer
next:	addi s2, s2, -1	#decremento la cantidad de chars guarda
	bne s2, 0, loop	#si la cantidad de chars es != 0 loopeo
	addi t4, zero, 10	
	sb t4, 0(s5)	#ascii code new line
	addi s1, s1, 1	#incremento BUF_PTR
	addi s5, s5, 1	#incremento el puntero al buffer
	bne s1, BUF_SZ, next2	#si BUF_PTR no es igual a BUF_SZ NO flu
	sw ra, 68(\$fp)	
	sw a0, 72(\$fp)	
	li a2, BUF_SZ	#Establezco la cantidad a flushear (tod
	jal flush_buffer	
	lw ra, 68(\$fp)	
	lw a0, 72(\$fp)	
	addi s1, zero, 0	#flushee, entonces reseteo el buffptr
	add s5, s0, zero	#reseteo el puntero al buffer
	jr ra	
next2:	sw s1, 64(\$fp) # guardo el buff ptr en el stack	
	jr ra	

```
#####
```

#Flush Buffer: flushea el buffer en su totalidad al File Pointer en 60(sp), req

```
flush_buffer:
    li v0, SYS_write           #carga la instrucción para escribir en
    lw a0, 60(sp)             #carga el file pointer en a0
    addi a1, $fp, BUFLOW_OFFSET #carga el offset del buffer en a1
    syscall
    blt v0, zero, error       #si v0 < 0 go to error
    sw zero, 64($fp)          #Reseteo el buff ptr
    jr ra
```

```
#####
```

#Print header: Imprime los datos del encabezado de la imagen

```
print_header:
    move s6, zero              #s6 = 0 (buffer pointer)
    addi s0, $fp, BUFLOW_OFFSET #s0 = BUFFERLOW
    add s6, s6, s0             #s6 = BUFFERLOW + s6 (buffer pointer)
    addi t0, zero, 80          #80 = 'P'
    sb t0, 0(s6)               #'P' en buffer
    addi s6, s6, 1             #s6++
    addi t0, zero, 50          #50 = '2'
    sb t0, 0(s6)               #'2' en buffer
    addi s6, s6, 1             #s6++
    addi t0, zero, 10          #10 = '\n'
    sb t0, 0(s6)               #'\n' en buffer
    addi s6, s6, 1             #s6++
    la s7, stringbuffer        #s7 = stringbufferaddress
    lw a0, 48($fp)              #Cargo en a0 el valor de x_res
    sw ra, 68($fp)              #Guardo la dirección de retorno en 68($fp)
    jal inttostr                #llamo a inttostr
    lw ra, 68($fp)              #Recupero la dirección de retorno

    add t2, zero, zero         #t2 = 0
looph1: add t1, t2, s7          #t1 = stringbufferaddress + t2(contador de caracteres)
    lb t0, 0(t1)               #primer elemento del stringbuffer en t0
    sb t0, 0(s6)               #almaceno primer elemento del stringbuffer en s6
    addi t2, t2, 1             #t2++
    addi s6, s6, 1             #s6++
    bne v0, t2, looph1         #cantidad de caracteres a imprimir != 0
    addi t0, zero, 10          #10 = '\n'
    sb t0, 0(s6)               #'\n' en buffer
    addi s6, s6, 1             #s6++

    lw a0, 52($fp)             #Cargo en a0 el valor de y_res
```

```

        sw ra, 68($fp)           #Guardo la dirección de retorno en 68($fp)
        jal inttostr             #llamo a inttostr
        lw ra, 68($fp)           #Recupero la dirección de retorno
        la s7, stringbuffer      #s7 = stringbufferaddress
        add t2, zero, zero       #t2 = 0 (contador de caracteres)
looph2: add t1, t2, s7           #t1 = stringbufferaddress + t2(contador de caracteres)
        lb t0, 0(t1)             #primer elemento del stringbuffer en t0
        sb t0, 0(s6)             #almaceno primer elemento del stringbuffer en s6
        addi t2, t2, 1           #t2++
        addi s6, s6, 1           #s6++
        bne v0, t2, looph2       #cantidad de caracteres a imprimir != 0
        addi t0, zero, 10        #10 = '\n'
        sb t0, 0(s6)             #'n' en buffer
        addi s6, s6, 1           #s6++

# Repito pasos anteriores con shades
        lw a0, 56($fp)           #Cargo en a0 el valor de shades
        sw ra, 68($fp)           #llamo a inttostr
        jal inttostr             #llamo a inttostr
        lw ra, 68($fp)           #s7 = stringbufferaddress
        la s7, stringbuffer
        add t2, zero, zero
looph3: add t1, t2, s7
        lb t0, 0(t1)
        sb t0, 0(s6)
        addi t2, t2, 1
        addi s6, s6, 1
        bne v0, t2, looph3
        addi t0, zero, 10        #10 = '\n'
        sb t0, 0(s6)
        addi s6, s6, 1

        addi s0, $fp, BUF_LOW_OFFSET #s0 = BUFFER_LOW
        sub a2, s6, s0           #Establezco la cantidad a flushear
        sw ra, 68($fp)           #Guardo la dirección de retorno en 68($fp)
        sw a0, 72($fp)           #Guardo a0 en 72($fp)
        jal flush_buffer         #llamo a flush_buffer
        lw ra, 68($fp)           #Recupero la dirección de retorno
        lw a0, 72($fp)           #Recupero a0
        jr ra

.end mips32_plot

.data
stringbuffer: .space 32
CTE_FOR3: .float 4.0
CTE_MULT: .float 2

```

References

- [1] Conjunto de Julia, WikiPedia, https://es.wikipedia.org/wiki/Conjunto_de_Julia.
- [2] Arquitectura de Computadores-John L. Hennessy - David A. Patterson -
Un Enfoque Cuantitativo