Universidad de Buenos Aires
Facultad de Ingeniería

66.20 Organización de computadoras

Trabajo práctico 1: Conjunto de instrucciones MIPS
1^{er} cuatrimestre de 2015

Paula Saffioti	Padrón: 92001	Email: paula.saffioti@gmail.com
Kaoru Heanna	Padrón: 91891	Email: kaoru.heanna@gmail.com
Julián Scialabba	Padrón: 92181	Email: julian.scialabba@gmail.com

1. Documentación

El objetivo del trabajo práctico es desarrollar una versión del comando tac de UNIX utilizando el conjunto de instrucciones MIPS. Ya contabámos con el comando implementado en lenguaje C y debíamos traducirlo a assembly MIPS. Para hacer esto optamos por dividir el código en funciones pequeñas, con el objetivo de poder implementarlas y testearlas independientemente. El algoritmo base del programa que se resume en la siguiente lista de pasos:

1. Abrir un archivo pasado por parámetro o recibir un stream de datos desde stdin.
2. Leer el archivo o el stream línea por línea hasta el final y guardarlo en un array. El tamaño de las líneas debe ser dinámico.
3. Recorrer el array en sentido inverso imprimiendo por stdout el contenido de cada línea.
4. Cerrar el archivo si corresponde.

El paso 1 fue implementado en C. Para el caso de los archivos lo resolvimos con la función fopen. Como resultado de este paso, nos quedó un puntero al archivo para operar con él. En caso de algún error, lo lanzamos por stderr.

Los pasos 2 y 3 fueron realizados íntegramente en assembly MIPS. Para el manejo de la memoria dinámica utilizamos las funciones ya implementadas mymalloc y myfree. Tuvimos que modificar el código que utilizaba realloc para que también pueda hacer uso de mymalloc.

El paso 4 para cerrar el archivo nuevamente fue implementado en C con la función fclose.

El único inconveniente durante el desarrollo de trabajo práctico lo encontramos con la implementación de fgets. El problema era que haciamos la llamada al syscall encargado de la lectura de archivos (el 14) con los parámetros necesarios pero no nos devolvía el bloque leído. Tras mucho intentar, consultamos en el grupo de la materia esta cuestión y al no obtener respuesta lo solucionamos llamando a fgets desde el código assembly.

2. Comandos

Para obtener el ejecutable creamos un makefile. Por lo que para compilar el programa solo hace falta hacer:

```
$ make
```

El makefile es el siguiente:

```
PROG  = tp1
CFLAGS = -g -Wall
DSRC = src/
all: $(PROG)

deleteOld:
        rm $(PROG)
```

```
tp1: $(DSRC)main.c $(DSRC)printHelp.S $(DSRC)printVersion.S $(
    DSRC)isEndOfLine.S $(DSRC)strLength.S $(DSRC)concatBuffer.S $(
    DSRC)tacFile.S $(DSRC)printLines.S $(DSRC)resizeArrayLines.S
    $(DSRC)mergeStrings.S $(DSRC)mymalloc.S $(DSRC)storeNewLine.S
    $(DSRC)readFile.S
        $(CC) $(CFLAGS) -o $@ $>

clean:
        rm -f $(PROG) *.so *.o *.a *.core

permissions:
        chmod +x $(PROG)
```

3. Corridas de prueba

Para corroborar el funcionamiento del programa contábamos con una serie de archivos de prueba.

Salidas de ejemplo:

return.txt:

of -1 is returned and errno is set to indicate the error. Upon successful completion a value of 0 is returned. Otherwise, a value

basic.txt

4. D
3. C
2. B
1. A

return.txt basic.txt

of -1 is returned and errno is set to indicate the error.
Upon successful completion a value of 0 is returned. Otherwise, a value
4. D
3. C
2. B
1. A

Para poder hacer todas las pruebas de manera rápida escribimos un script que compilaba el programa, luego lo ejecutaba para los archivos mencionados dándolos vuelta dos veces y finalmente verificaba que el archivo resultante fuera igual al original con md5sum. El script es el siguiente:

```
testFile() {
        echo -e "\nTEST␣"$1".txt"
        ./tp1 test-files/$1.txt | ./tp1 > output/$1.txt
        res1=`cat test-files/$1.txt | md5`
        res2=`cat output/$1.txt | md5`
        if [ $res1 == $res2 ];
        then
```

```
                echo -e "OK\n"
        else
                echo -e "ERROR\n"
        fi
}

echo Compilando...
make

testFile empty
testFile basic
testFile empty-lines
testFile large-file
testFile return
testFile status
```

4. Código fuente

4.1. main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern void printHelp();
extern void printVersion();
extern int tacFile(FILE* fp);

int main(int argc, char** argv) {

    if ((argc == 2) && ((strcmp(argv[1], "-h") == 0) || (strcmp(
       argv[1], "--help") == 0))) {
        printHelp();
        return (EXIT_SUCCESS);
    }

    if ((argc == 2) && ((strcmp(argv[1], "-V") == 0) || (strcmp(
       argv[1], "--version") == 0))) {
        printVersion();
        return (EXIT_SUCCESS);
    }

    int result;

    if (argc < 2) { //no tengo archivo de entrada, uso standard
       input
        result = tacFile(stdin);
        return (result);
    }
```

```
    int i;
    for (i = 1; i < argc; i++) {
        FILE *fp;
        fp = fopen(argv[i], "r");
        if (fp == NULL) {
            fprintf(stderr, "%s", argv[i]);
            fprintf(stderr, ": nombre de archivo o comando
                inv lido.\n");
            return (EXIT_FAILURE);
        }
        tacFile(fp);
        fclose(fp);
    }
    return (EXIT_SUCCESS);
}
```

4.2. printHelp.S

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

        .text
        .align  2
        .globl  printHelp
        .ent    printHelp
printHelp:
        .frame  $fp,40,ra       #4SRA + 2LTA + 4ABA
        .set    noreorder
        .cpload t9
        .set    reorder
        subu    sp,sp,40
        .cprestore 24
        sw      $fp,28(sp)
        sw      ra,32(sp)
        move    $fp, sp
        la      a0, msg
        la      t9, strLength
        jal     ra, t9

        move    t0, v0
        sw      t0,16($fp)
        li      v0, SYS_write
        li      a0, 1
        la      a1, msg
        lw      t0,16($fp)
        move    a2, t0
        syscall

        lw      $fp,28(sp)
```

```
        lw      ra,32(sp)
        lw      gp,24(sp)
        addu    sp,sp,40
        jr      ra
        .end    printHelp

        .rdata
msg:
        .asciiz "Usage:\n        tp0 -h\n        tp0 -V\n        
           tp0 [file...]\nOptions:\n        -V, --version Print 
           version and quit.\n  -h, --help Print this information 
           and quit.\nExamples:\n        tp0 foo.txt bar.txt\n   
           tp0 gz.txt\n"
```

4.3. printVersion.S

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

        .text
        .align  2
        .globl  printVersion
        .ent    printVersion
printVersion:
        .frame  $fp,40,ra       #4SRA + 2LTA + 4ABA
        .set    noreorder
        .cpload t9
        .set    reorder
        subu    sp,sp,40
        .cprestore 24
        sw      $fp,28(sp)
        sw      ra,32(sp)
        move    $fp, sp
        la      a0, msg
        la      t9, strLength
        jal     ra, t9

        move    t0, v0
        sw      t0,16($fp)
        li      v0, SYS_write
        li      a0, 1
        la      a1, msg
        lw      t0,16($fp)
        move    a2, t0
        syscall

        lw      $fp,28(sp)
        lw      ra,32(sp)
        lw      gp,24(sp)
        addu    sp,sp,40
```

```
        jr      ra
        .end    printVersion

        .rdata
msg:
        .asciiz "tp1 1.1\nCopyright    2015 FIUBA.\nEsto es
           software libre: usted es libre de cambiarlo y
           redistribuirlo.\nNo hay NINGUNA GARANTA, hasta donde
           permite la ley.\n\nEscrito por Julian Scialabba,
           Kaoru Heanna y Paula Saffioti.\n"
```

4.4. isEndOfLine.S

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

        .text
        .align  2

        .globl  isEndOfLine
        .ent    isEndOfLine
isEndOfLine:
#creo frame
        .frame  $fp, 8, ra   # 2(SRA) + 0(FPA) + 0(LTA) + 0(ABA)
           (leaf function)
        .set    noreorder
        .cpload t9
        .set    reorder
        subu    sp, sp, 8
# guardo registros
        .cprestore 0
        sw      $fp, 4(sp)
        move    $fp, sp
        sw      a0, 8($fp)              #guardo el primer
           argumento en el ABA de mi caller

        # Use v0 for the result.
        li      v0, 0
        # el codigo ascii de new line es 10
        li      t0, 10
        beq     a0, t0, endOfLineTrue
        li      v0, 0
        j       isEndOfLine_return
endOfLineTrue:
        li      v0, 1
        j       isEndOfLine_return

isEndOfLine_return:
        # Destruimos el frame.
        # Restauro callee-saved regs
```

```
        lw      $fp, 4(sp)
        lw      gp, 0(sp)
        addu            sp, sp, 8

        # Retorno.
        #
        jr      ra
        .end    isEndOfLine
```

4.5. strLength.S

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

        .text
        .align  2
        .globl  strLength
        .ent    strLength

strLength:
        .frame  $fp, 16, ra
        .set    noreorder
        .cpload t9
        .set    reorder

        #creo stack frame
        subu    sp, sp, 16

        # Guardo $gp y $sp en SRA
        .cprestore 8
        sw      $fp, 12(sp)

        #De ahora en adelante utilizo $fp como sp
        move    $fp, sp

        # Guardo argumento en ABA de caller
        sw a0, 16($fp)

        # Creo variable incremental y la guardo en primer
           registro de LTA
        move t0, zero
        sw t0, 0($fp)

_for:
        # Condicion de corte del for
        addu t1, a0, t0  # Apunto en t1 al siguiente byte del
           string recibido por param
        lb t1, 0(t1) # Cargo el character apuntado en t1
        beq t1, zero, _end_for
        addu t0, t0, 1
```

```
        j _for

_end_for:

        # Preparo todo para retorno
        move v0, t0
        lw gp, 8($fp)
        lw $fp, 12(sp)
        addu sp, sp, 16
        jr ra

        .end strLength
```

4.6. concatBuffer.S

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

# en LTA guardo: line, lineLength, buffer, bufferLength,
    concatAddress?

        .text
        .align          2

        .globl          concatBuffer
        .ent            concatBuffer
concatBuffer:
#creo frame
        .frame          $fp, 56, ra   # 4(SRA) + 0(FPA) + 6(LTA)
           + 4(ABA) (leaf function)
        .set            noreorder
        .cpload         t9
        .set            reorder
        subu            sp, sp, 56

# guardo callee-saved registers
        .cprestore      40              # guardo gp
        sw              $fp, 44(sp)
        move            $fp, sp
        sw              ra, 48($fp)
#guardo a0 y a1 en el ABA de mi caller
        sw              a0, 56($fp)
        sw              a1, 60($fp)

lineLength:
        move            t0, a0                  # t0 = line. pos
           16
        sw              t0, 16($fp)
        move            t1, zero                # t1 =
           lineLength. pos 20
```

```
        sw              t1, 20($fp)
        beq             t0, zero, bufferLength  #si line esta
           vacio, queda lineLength en 0
        move            a0, t0
        la              t9, strLength
        jal             ra, t9
        move            t1, v0                  # guardo en t1 lo
            que devolvio strLength
        sw              t1, 20($fp)

bufferLength:
        move            t2, a1                  # t2 = buffer.
           pos 24
        sw              t2, 24($fp)
        move            t3, zero                # t3 =
           bufferLength. pos 28
        sw              t3, 28($fp)
        beq             t2, zero, concat # si buffer esta vacio,
           queda bufferLength en 0
        move            a0, t2
        la              t9, strLength
        jal             ra, t9
        move            t3, v0                  # guardo en t3 lo
            que devolvio strLength
        sw              t3, 28($fp)
        bne             t3, zero, concat
        lw              v0, 16($fp)             # si bufferLength
            es 0, devuelvo line
        j               return

concat:
        lw              a0, 16($fp)             # string 1
        lw              a1, 20($fp)             # len1
        lw              a2, 24($fp)             # string 2
        lw              a3, 28($fp)             # len2
        la              t9, mergeStrings
        jal             ra, t9
        j               return                  # ya tengo
           guardado en v0 el string a retornar

return:
# Restauro callee-saved regs
        lw              gp, 40(sp)
        lw              $fp, 44(sp)
        lw              ra, 48(sp)

# Destruimos el frame.
        addu            sp, sp, 56
```

```
# Retorno.
        jr              ra
        .end            concatBuffer
```

4.7. printLines.S

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

        .text
        .align  2
        .globl  printLines
        .ent    printLines

printLines:
        .frame  $fp,40,ra
        .set    noreorder
        .cpload t9
        .set    reorder
        subu    sp,sp,40
        .cprestore 24
        sw      $fp,28(sp)
        sw      ra,32(sp)
        move    $fp, sp

        sw      a0, 40($fp)
        sw      a1, 44($fp)
        move    t0, a0
        move    t1, a1
        subu    t0, t0, 1
        sw      t0, 16($fp)
        sll     t3, t0, 2
        addu    t1, t1, t3  #Apunto a ultimo char*
        sw      t1, 20($fp)
_loop:
        blt     t0, zero, _end_loop
        lw      t1, 20($fp)
        lw      t2, 0(t1)

        # Llamada strLength
        move    a0, t2
        la      t9, strLength
        jal     ra, t9
        move    t3, v0 #longitud

        # Llamo a syscall write
        li      v0, SYS_write
        li      a0, 1
        move    a1, t2 #Contenido
        move    a2, t3
```

```
        syscall

        # Libero memoria de char*
        lw      t1, 20($fp)
        lw      a0, 0(t1)
        la      t9, myfree
        jal     ra, t9

        # Actualizo estado variables
        lw      t0, 16($fp)
        lw      t1, 20($fp)
        subu    t0, t0, 1
        subu    t1, t1, 4
        sw      t0, 16($fp)
        sw      t1, 20($fp)

        j   _loop

_end_loop:
        lw      $fp,28(sp)
        lw      ra,32(sp)
        lw      gp,24(sp)
        addu    sp,sp,40
        jr      ra

        .end    printLines
```

4.8. resizeArrayLines.S

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

        .text
        .align  2
        .globl  resizeArrayLines
        .ent    resizeArrayLines

resizeArrayLines:
        .frame  $fp,40,ra
        .set    noreorder
        .cpload t9
        .set    reorder
        subu    sp,sp,40
        .cprestore 24
        sw      $fp,28(sp)
        sw      ra,32(sp)
        move    $fp, sp

        sw      a0, 40($fp)
        sw      a1, 44($fp)
```

```
        move    t0, zero
        sw      t0, 16($fp)

        addu    t1, a1, 1
        sll     t1, t1, 2
        move    a0, t1
        la      t9, mymalloc
        jal     ra, t9
        sw      v0, 20($fp)

        lw      t0, 16($fp)
        lw      t1, 44($fp)
_loop:
        bge     t0, t1, _end_loop

        lw      t2, 40($fp)
        lw      t3, 20($fp)

        sll     t4, t0, 2
        addu    t2, t2, t4
        addu    t3, t3, t4
        lw      t4, 0(t2)
        sw      t4, 0(t3)

        addu    t0, t0, 1
        j   _loop

_end_loop:

        lw      a0, 40($fp)
        la      t9, myfree
        jal     ra, t9

        lw      v0, 20($fp)
        lw      $fp,28(sp)
        lw      ra,32(sp)
        lw      gp,24(sp)
        addu    sp,sp,40
        jr      ra

        .end    resizeArrayLines
```

4.9. mergeStrings.S

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

# a0: char* string1 -> t0
# a1: size_t len1 -> t1
```

```
# a2: char* string2 -> t2
# a3: size_t len2 -> t3

        .text
        .align          2

        .globl          mergeStrings
        .ent            mergeStrings
mergeStrings:
#creo frame
        .frame          $fp, 56, ra   # 4(SRA) + 0(FPA) + 6(LTA)
           + 4(ABA) (leaf function)
        .set            noreorder
        .cpload         t9
        .set            reorder
        subu            sp, sp, 56

# guardo callee-saved registers
        .cprestore      40              # guardo gp
        sw              $fp, 44(sp)
        move            $fp, sp
        sw              ra, 48($fp)
#guardo a0,a1,a2,a3 en el ABA de mi caller
        sw              a0, 56($fp)
        sw              a1, 60($fp)
        sw              a2, 64($fp)
        sw              a3, 68($fp)

        move            t0, a0
        sw              t0, 16($fp)             # string1
        move            t1, a1
        sw              t1, 20($fp)             # len1
        move            t2, a2
        sw              t2, 24($fp)             # string2
        move            t3, a3
        sw              t3, 28($fp)             # len2

allocMemory:
        addu            t4, t1, t3              # t4 = len1 +
           len2. pos 32
        sw              t4, 32($fp)
        addiu           t4, t4, 1               # t4 <- t4 +1
        move            a0, t4                  # cantidad de
           bytes a allocar
        la              t9, mymalloc
        jal             ra, t9
        sw              v0, 36($fp)             # dir(concat) =
           address allocated memory. pos 36
```

```
copyString1:
        move            t6, zero                # t6 = i
        lw              t0, 16($fp)             # t0 = &string1
        lw              t1, 20($fp)             # t1 = len1
        lw              t5, 36($fp)             # t5 = &concat
_forString1:
        bge             t6, t1, copyString2             # si (i
           >= len1) voy a copyString2

        addu            t7, t0, t6              # t7 = &string1[i
           ]
        lb              t8, 0(t7)               # t8 = string1[i]

        addu            t7, t5, t6              # t7 = &concat[i]
        sb              t8, 0(t7)               # concat[i] <-
           string1[i]

        addiu           t6, t6, 1               # i++
        j               _forString1

copyString2:
        move            t6, zero                # t6 = i
        lw              t2, 24($fp)             # t2 = &string2
        lw              t3, 28($fp)             # t3 = len2
        lw              t5, 36($fp)             # t5 = &concat
        lw              t1, 20($fp)             # t1 = len1
_forString2:
        bge             t6, t3, endMerge                # si (i
           >= len2) voy a endMerge
        addu            t7, t2, t6              # t7 = &string2[i
           ]
        lb              t8, 0(t7)               # t8 = string2[i]

        addu            t7, t6, t1              # t7 = j <- i +
           len1
        addu            t7, t5, t7              # t7 = &concat[j]
        sb              t8, 0(t7)               # concat[j] =
           string2[i]

        addiu           t6, t6, 1               #i++
        j               _forString2

endMerge:
        lw              t5, 36($fp)             # t5 = dir(concat
           )
        lw              t4, 32($fp)             # t4 = len1 +
           len2
        addu            t7, t5, t4              # t7 = &concat[
           len1+len2]
```

```
        sb              zero, 0(t7)             # concat[len1+
            len2] = 0
        lw              v0, 36($fp)             # devuelvo dir(
            concat)

return:
        # Restauro callee-saved regs
        lw              gp, 40(sp)
        lw              $fp, 44(sp)
        lw              ra, 48(sp)

# Destruimos el frame.
        addu            sp, sp, 56

# Retorno.
        jr              ra
        .end            mergeStrings
```

4.10. storeNewLine.S

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

        .text
        .align  2

        .globl  storeNewLine
        .ent    storeNewLine
storeNewLine:
        .frame  $fp,48,ra

        .set    noreorder
        .cpload t9
        .set    reorder

        subu    sp,sp,48

        .cprestore 32

        sw      ra,36(sp)
        sw      $fp,40(sp)
        move    $fp,sp

        sw      a0,20($fp)
        sw      a1,24($fp)
        sw      a2,16($fp)
        sw      a3,28($fp)

        lw      a0,20($fp)
        la      t9,isEndOfLine
```

```
        jal     ra,t9

        beq     v0,zero,end

        lw      a0,16($fp)

        lw      t0,24($fp)
        lw      a1,0(t0)
        la      t9,resizeArrayLines
        jal     ra,t9
        sw      v0,16($fp)

        lw      t0,24($fp)
        lw      t0,0(t0)
        lw      t1,16($fp)

        sll     t3, t0, 2
        addu    t1, t1, t3

        lw      t0,28($fp)
        lw      t2,0(t0)
        sw      t2,0(t1)
        li      a0,1
        la      t9,mymalloc
        jal     ra,t9

        lw      t0,28($fp)
        sw      v0,0(t0)

        lw      t0,24($fp)
        lw      t1,0(t0)
        addu    t1,t1,1
        sw      t1,0(t0)


end:
        lw      v0,16($fp)
        lw      gp,32(sp)
        lw      ra,36(sp)
        lw      $fp,40(sp)
        addu    sp,sp,48
        jr      ra

        .end    storeNewLine
        .rdata
```

4.11. readFile.S

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

```
# a0 = buffer
# a1 = bufIncrSize
# a2 = filePointer

        .text
        .align  2
        .globl  readFile
        .ent    readFile

readFile:
        .frame  $fp, 32, ra
        .set    noreorder
        .cpload t9
        .set    reorder
        subu    sp, sp, 32
        .cprestore 16
        sw      $fp, 20(sp)
        sw      ra, 24(sp)
        move    $fp,sp
        sw      a0, 32($fp)
        sw      a1, 36($fp)
        sw      a2, 40($fp)

        jal fgets

        lw      gp, 16(sp)
        lw      $fp, 20(sp)
        lw      ra, 24(sp)
        addu    sp,sp,32
        jr              ra
        .end readFile
```

4.12. tacFile.S

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

        .text
        .align  2

        .globl  tacFile
        .ent    tacFile
tacFile:
        .frame  $fp,64,ra

        .set    noreorder
        .cpload t9
        .set    reorder
```

```
        subu    sp,sp,64

        .cprestore 48

        sw      ra,56(sp)
        sw      $fp,52(sp)
        move    $fp,sp

        sw      a0,64($fp)

        li      t0,0
        sw      t0,16($fp)
        li      a0,1
        la      t9,mymalloc
        jal     ra,t9
        sw      v0,20($fp)

        li      a0,10
        la      t9,mymalloc
        jal     ra,t9
        sw      v0,36($fp)


        li      a0,1
        la      t9,mymalloc
        jal     ra,t9
        sw      v0,24($fp)

cond_while:

        lw      a0,36($fp)
        li      a1,10
        lw      a2,64($fp)
        la      t9,readFile
        jal     ra,t9

        bne     v0,zero,bloque_while
        b       end

bloque_while:
        sw      v0,36($fp)

        lw      a0,24($fp)
        lw      a1,36($fp)
        la      t9,concatBuffer
        jal     ra,t9
        sw      v0,28($fp)

        lw      a0,24($fp)
```

```
        la      t9,myfree
        jal     ra,t9

        lw      t0,28($fp)
        sw      t0,24($fp)

        lw      a0, 36($fp)
        la      t9,strLength
        jal     ra,t9
        move    t0,v0
        subu    t0,t0,1
        lw      t1,36($fp)
        addu    t2,t1,t0
        lb      t3,0(t2)
        sw      t3,32($fp)

        lw      a0,32($fp)
        addu    a1,$fp,16
        lw      a2,20($fp)
        addu    a3,$fp,24
        la      t9,storeNewLine
        jal     ra,t9
        sw      v0,20($fp)

        b       cond_while
end:

        lw      a0,16($fp)
        lw      a1,20($fp)
        la      t9,printLines
        jal     ra,t9

        lw      a0,24($fp)
        la      t9,myfree
        jal     ra,t9

        lw      a0,20($fp)
        la      t9,myfree
        jal     ra,t9

        lw      a0,36($fp)
        la      t9,myfree
        jal     ra,t9

        li      v0,1

        lw      gp,48(sp)
        lw      ra,56(sp)
        lw      $fp,52(sp)
```

```
addu    sp,sp,64
jr      ra

.end    tacFile

.rdata
.align 2
```

5. Enunciado