CSE 2021
COMPUTER ORGANIZATION

HUGH CHESSER
CSEB 1012U

W4-M

System Calls - Examples

Example # 1: <u>Print a string</u>
```
.data
    str:     .asciiz "the answer is"
.text
    addi $v0,$zero,4
    la $a0,str    # pseudoinstruction
    syscall
```
Example # 2: <u>Input an integer</u>
```
    addi $v0,$zero,5
    syscall
```

Example # 3: <u>Print an integer</u>
```
    addi $v0,$zero,1
    addi $a0,$s0,$zero
    syscall
```

Example # 4: <u>Read String</u>
```
    addi $v0,$zero,8
    la $a0,Buff  #$a0=address of Buff
    addi $a1,$zero,60 #$a1=max. len.
    syscall
```

Service	System Call Code ($v0)	Arguments	Result
print_int	1	$a0 = int	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string address	
read_int	5		int (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		terminate prog

Putting it all together (2)

Activity: Write a MIPS program which does the following:

1. Accepts an integer N using the following prompt
    <p align="center" style="color:green">Please input a value for N =</p>
2. Computes the sum of integers from 1 to N, i.e., $(1 + 2 + \ldots + N)$ if $N > 0$
3. Displays the result (X) as
    <p align="center" style="color:green">The sum of the integers from 1 to N is X</p>
2. Waits for the next number N.
3. If $N <= 0$, the program exits with the following farewell
    <p align="center" style="color:green">Chao - Have a good day</p>

Run the program in the spim simulator to verify the results

main routine

```
1   .data
2       str1:        .asciiz "\nPlease input a value for N = "
3       str2:        .asciiz "The sum of the integers from 1 to N is...\n\t\t\t"
4       str3:        .asciiz "\n\"Chao\" - Have a good day"
5   .text
6   main:
7       addi $v0,$zero,4
8       la $a0,str1          # pseudoinstruction
9       syscall             # Print out user prompt
10  #-------------------------------------------
11      addi $v0,$zero,5    # input integer
12      syscall
13  #-------------------------------------------
14      add $a0,$zero,$v0   # Move input integer as to $a0 - input to sum
15      slt $t0,$v0,$zero
16      bne $t0,$zero,end  # if input <0 end
17      jal sum             # Jump to subroutine sum
18  #-------------------------------------------
19      addi $sp, $sp, -4
20      sw $v0, 4($sp)      # Put the sum on the stack
21      addi $v0,$zero,4
22      la $a0,str2
23      syscall             # Print message for sum
24  #-------------------------------------------
25      addi $v0, $zero,1
26      lw $a0, 4($sp)      # Get sum from stack
27      addi $sp, $sp,4
28      syscall             # Print sum
29  #-------------------------------------------
30  end:
31      addi $v0,$zero,4
32      la $a0,str3
33      syscall             # Print out good bye message
34      add $a0,$a0,$zero
35      addi $v0,$zero,17
36      syscall             # exit2
37  #-------------------------------------------
```

sum subroutine

```
38  sum:
39      addi $sp, $sp, -8   # Set up the stack
40      sw $ra, 4($sp)      # Save return address
41      addi $t0, $a0, 0    # Initialize the sum
42      li $v0, 0           # Initialize return value
43      beq $t0,$zero, ret  # If argument is 0 then return
44      addi $t1, $t0, -1   # Compute n-1
45      sw $t0, 8($sp)      # Save caller saved regs
46      addi $a0, $t1, 0    # Move n-1 into argument register
47      jal sum             # Call sum
48  #-------------------------------------------
49      lw $t0, 8($sp)      # Restore caller saved reg
50      add $v0, $t0, $v0   # Add return value to $t0
51      lw $ra, 4($sp)      # Get the return address
52      addi $sp,$sp,8
53  ret:
54      jr $ra              # Return
```

```
1  ######################################################################
2  #       Program Name:        Sum of Positive Integers
3  #       Programmer:          H. Chesser
4  #       Date last Modified:  Sept 24, 2012
5  ######################################################################
6
7
8  #####################
9  # Procedure  <main> #
10 #####################
11
12 # The assembler directive .data tells the assembler that all data allocation directives should
13 # allocate data in a portion of memory called the data segment
14
15             .data
16 Prompt:     .asciiz "\n Please input a value for N = "
17 Result:     .asciiz "The sum of the integers from 1 to N is...\n\t\t\t "
18 Bye:        .asciiz "\n*** \"Chao\" - Have a good day *** \n"
19
20 # The assembler directive .globl <label> declare that <label> is global so it can be referenced
21 # from other files
22             .globl  main
23
24 # The assembler directive .text <address> tells the assembler that subsequent items must be
25 # placed in the user text segment. In SPIM, these items may only be instructions or words. If
26 # the optional argument <address> is specified, subsequent items are stored starting at address
27 # <address>
28             .text
29 ########################## save registers ##############################
30 main:
31             addi    $sp,    $sp,    -12
32             sw      $ra,    8($sp)
33             sw      $a0,    4($sp)
34             sw      $v0,    0($sp)
35
36 ########################## main body ##############################
37 again:      addi    $v0,    $zero,  4       # Sys Call # 4 (Print String)
38             la      $a0,    Prompt          # load address of Prompt into $a0
39             syscall                         # print the Prompt message
40
41             addi    $v0,    $zero,  5       # Sys Call # 5 (Read Integer)
42             syscall                         # reads the value of N into $v0
43
44             slti    $t0,    $v0,    1       # Set $t0 = 1 if $v0 < 1, else $t0 = 0
45             bne     $t0,    $zero,  End     # if ($v0 < 1), go to End
46
47             add     $a0,    $v0,$zero       # $v0 = $a0
48             jal     myadd                   # call myadd
49
50             addi    $t1,    $v0,    0       # $t1 = $v0
51
52             addi    $v0,    $zero,  4       # Sys Call # 4 (Print String)
```

```
75 ######################
76 # Procedure  <myadd> #
77 ######################
78
79
80 ########################## save registers ##############################
81 myadd:
82             addi    $sp,    $sp,    -8
83             sw      $ra,    4($sp)
84             sw      $s1,    0($sp)
85
86 ########################## myadd body ##############################
87             addi    $s1,    $zero,  0       # $s1 = 0
88 Loop:
89             add     $s1,    $s1,    $a0     # $s1 = $s1 + $a0
90             addi    $a0,    $a0,    -1      # $a0 = ($a0 - 1)
91             slti    $t0,    $a0,    1       # $t0 = 1 if $a0 < 1, else $t0 = 0
92             beq     $t0,    $zero,  Loop    # if ($v0 > 1), go to Loop
93             addi    $v0,    $s1,    0
94
95 ########################## restoring registers ##############################
96             lw      $s1,    0($sp)
97             lw      $ra,    4($sp)
98             addi    $sp,    $sp,    8
99
100 ########################## procedure return ##############################
101            jr $ra
```

Agenda for Today

1. Addition, Subtraction
2. Overflow
3. Multiplication
4. Division
5. Floating Point: IEEE 754 single and double precision formats

Patterson: Sections 3.1 – 3.5

Wednesday – Quiz #1

Addition and Subtraction

In MIPS, addition and subtraction for signed numbers use 2's complement arithmetic

Example 1: Add 10_{ten} and 15_{ten}

bit 1	bit 2	Prev. Carry	Sum	Next Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1: Truth Table for addition

Example 2: Subtract 15_{ten} from 10_{ten}

Addition and Subtraction

In MIPS, addition and subtraction for signed numbers use 2's complement arithmetic

Example 1: Add 10_{ten} and 15_{ten}

Step 1: Represent the operands in 2's complement

$10_{ten} = $ 0000 0000 0000 0000 0000 0000 0000 1010$_{two}$

$15_{ten} = $ 0000 0000 0000 0000 0000 0000 0000 1111$_{two}$

Step 2: Perform bit by bit addition using table 1.

$25_{ten} = $ 0000 0000 0000 0000 0000 0000 0001 1001$_{two}$

Example 2: Subtract 15_{ten} from 10_{ten}

The problem is reduced to $(10_{ten} + (-15_{ten}))$

$10_{ten} = $ 0000 0000 0000 0000 0000 0000 0000 1010$_{two}$

$-15_{ten} = $ 1111 1111 1111 1111 1111 1111 1111 0001$_{two}$

$-5_{ten} = $ 1111 1111 1111 1111 1111 1111 1111 1011$_{two}$

bit 1	bit 2	Prev. Carry	Sum	Next Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1: Truth Table for addition

Overflow (1)

Recall that:

Smallest signed integer: 1000 0000 0000 0000 0000 0000 0000 0000$_{two}$

$$= -(2^{31})_{ten} = -2{,}147{,}483{,}648_{ten}$$

Largest signed integer: 0111 1111 1111 1111 1111 1111 1111 1111$_{two}$

$$= (2^{31} - 1)_{ten} = 2{,}147{,}483{,}647_{ten}$$

What happens if the result of an operation is more than the largest signed integer or less than the smallest signed integer?

Example: Add $2{,}147{,}483{,}640_{ten}$ and 28_{ten}

28_{ten} = 0000 0000 0000 0000 0000 0000 0001 1100$_{two}$

$2{,}147{,}483{,}640_{ten}$ = 0111 1111 1111 1111 1111 1111 1111 1000$_{two}$

$28_{ten} + 2{,}147{,}483{,}640_{ten}$ = 1000 0000 0000 0000 0000 0000 0001 0100$_{two}$

                                     = $-2{,}147{,}483{,}628_{ten}$

Overflow caused the value to be perceived as a negative integer

Overflow (2)

When can overflow occur?

Operation	Operand A	Operand B	Result indicating overflow
A + B	A >= 0	B >= 0	< 0
A + B	A < 0	B < 0	>= 0
A – B	A >= 0	B < 0	< 0
A – B	A < 0	B >= 0	>= 0

Integer Multiplication, Division

Both operations really imply a series of additions and subtractions

Example: Multiply 10_{ten} and 3_{ten}:

10_{ten} $\qquad = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1010_{two}$

3_{ten} $\qquad = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{two}$

$10_{ten} * 1_{ten}$ $\qquad = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1010_{two}$

$10_{ten} * 2_{ten}$ $\qquad = \underline{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0100_{two}}$

$\qquad\qquad = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 1110_{two} = 0x1e = 30_{ten}$

Example: Divide 28_{ten} by 9_{ten}:

28_{ten} $\qquad = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 1100_{two}$

9_{ten} $\qquad = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001_{two}$

$$
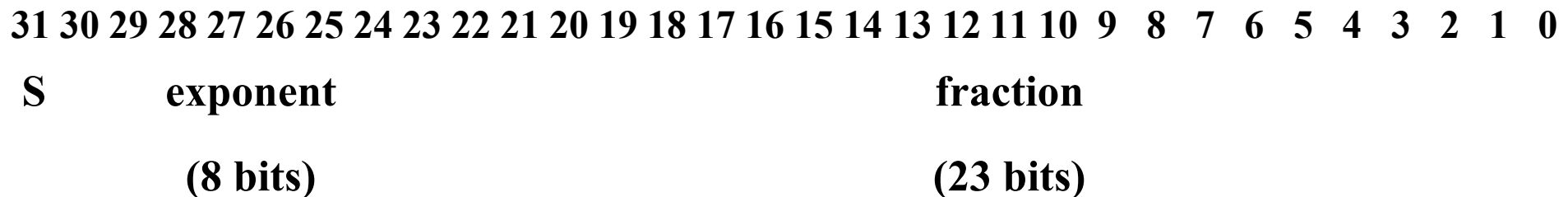\begin{array}{r}
11 \\
1001\overline{)11100} \\
\underline{1001} \\
1010 \\
\underline{1001} \\
1
\end{array}
$$

Floating Point: Single Precision

1. In MIPS, decimal numbers are represented with the IEEE 754 binary representation that uses the **normalized** standard scientific binary notation defined as

$$(-1)^S \times (1 + \text{fraction})_{two} \times 2^{\text{exponent - bias}}$$

2. A number in normalized scientific notation has a mantissa that has no leading 0's and must be of the form $(1 + \text{fraction})$. For example, the binary representations 2.0×2^{-5}, 0.5×2^{-3}, 4.0×2^{-6}, and 1.0×2^{-4} are all equivalent but only 1.0×2^{-4} is the normalized scientific binary notation.

3. MIPS allows for two floating point representations: Single precision and double precision.

4. Single precision has a bias of 127 while double precision has a bias of 1023.

5. In single precision, the floating point representation is 32 bit long and has the following form

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

S   exponent   fraction

(8 bits)   (23 bits)

where S represents the sign bit, which is 1 for negative numbers and 0 for positive numbers.

Activity 2:

Represent $-0.75_{ten,}$ 1.4_{ten} in single precision of IEEE 754 binary representation.

1. In double precision, the value of bias in

$$(-1)^S \times (1 + \text{fraction})_{\text{two}} \times 2^{\text{exponent - bias}}$$

   is 1023.

2. In single precision, the floating point representation is 64 bit long and has the following form

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

S   **exponent**   **fraction**

   (11 bits)   **(Total of 52 bits)**

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

fraction (continued)

Activity 3:

Represent -0.75_{ten} in double precision of IEEE 754 binary representation.

Activity 4:

Show that the largest magnitude that can be represented using single precision is $\pm 6.8_{\text{ten}} \times 10^{38}$, while the smallest fraction that can be represented is $\pm 5.9_{\text{ten}} \times 10^{-39}$.

Floating Point Registers

Name	Example	Comments
32 floating point registers each is 32 bits long	$f0,$f1,$f2,$f3,$f4 ,…,$f31	**MIPS floating point registers are used in pairs for double precision numbers**
Memory w/ 2^{30} words	**Memory[0], Memory[4], … Memory[4294967292]**	**Memory is accessed one floating point (single or double precision) at a time**

The following is the established register usage convention for the floating point registers:

$f0,$f1,$f2,$f3:   Function-returned values

$f4,$f5,…, $f11:   Temporary values

$f12,$f13,$f14,$f15:   Arguments passed into a function

$f16,$f17,$f18,$f19:   More Temporary values

$f20,$f21,…, $f31:   Saved values

Floating Point Instructions

Category	Instruction	Example	Meaning	Comments
Arithmetic	**FP add single**	`add.s $f2,$f4,$f6`	$f2 ← $f4+$f6	**Single Prec.**
	FP subtract single	`sub.s $f2,$f4,$f6`	$f2 ← $f4-$f6	**Single Prec.**
	FP multiply single	`mul.s $f2,$f4,$f6`	$f2 ← $f4×$f6	**Single Prec.**
	FP divide single	`div.s $f2,$f4,$f6`	$f2 ← $f4/$f6	**Single Prec.**
	FP add double	`add.d $f2,$f4,$f6`	$f2 ← $f4+$f6	**Double Prec.**
	FP subtract double	`sub.d $f2,$f4,$f6`	$f2 ← $f4-$f6	**Double Prec.**
	FP multiply double	`mul.d $f2,$f4,$f6`	$f2 ← $f4×$f6	**Double Prec.**
	FP divide double	`div.d $f2,$f4,$f6`	$f2 ← $f4/$f6	**Double Prec.**
Data Transfer	**load word copr.1**	`lwc1 $f2,100($s2)`	$f2 ← Mem[$s2+100]	**Single Prec.**
	store word copr.1	`swc1 $f2,100($s2)`	Mem[$s2+100] ← $f2	**Single Prec.**
Conditional branch	**FP compare single (eq, ne, lt, le, gt, ge)**	`c.lt.s $f2,$f4`	if($f2<$f4)cond = 1, else cond = 0	**Single Prec.**
	FP compare double (eq, ne, lt, le, gt, ge)	`c.lt.d $f2,$f4`	if($f2<$f4)cond = 1, else cond = 0	**Double Prec.**
	Branch on FP true	`bc1t 25`	if cond==1 go to PC+100+4	**Single/ Double Prec.**
	Branch on FP false	`bc1f 25`	if cond==0 go to PC+100+4	**Single/ Double Prec.**

Example

```
# calculate area of a circle
        .data
Ans:     .asciiz      "The area of the circle is: "
Ans_add:.word         Ans                           # Pointer to String (Ans)
Pi:      .double       3.1415926535897924
Rad:     .double       12.345678901234567
Rad_add:.word         Rad                           # Pointer to float (Rad)
        .text
main: lw $a0, Ans_add($0)             # load address of Ans into $a0
      addi $v0, $0, 4                 # Sys Call 4 (Print String)
      syscall
#----------------                     # load float (Assembler Instruction)
      la $s0, Pi                      # load address of Pi into $s0
      ldc1 $f2, 0($s0)               # $f2 = Pi
#----------------                     # load float (MIPS Instruction)
      lw $s0, Rad_add($0)            # load address of Rad into $s0
      ldc1 $f4, 0($s0)               # $f4 = Rad
      mul.d $f12, $f4, $f4
      mul.d $f12, $f12, $f2
      addi $v0, $0, 3                 # Sys Call 3 (Print Double)
      syscall
exit: jr $ra
```