Department of Computer Science
Ashoka University

Computer Organization and Systems

Assignment 5

Name: Rushil Gupta and Dhruman Gupta

Problem 1: Compute Floor Division by 9

Pseudocode

```
BEGIN
    Print "Enter s (s mod 9 != 0): "
    Read s
    z = 0
    WHILE s >= 9 DO
        s = s - 9
        z = z + 1
    END WHILE
    Print "s div 9 = " and z
END
```

Assembly Code

```
.data
prompt: .asciiz "Enter s (s mod 9 != 0): "
resultStr: .asciiz "s div 9 = "

.text
.globl main

main:
    # Print prompt message
    li $v0, 4
    la $a0, prompt
    syscall

    # Read integer input into register $t0 (s)
    li $v0, 5
    syscall
    move $t0, $v0

    # Initialize result as 0
    li $t1, 0


sub_loop:
    slti $t2, $t0, 9 # Check if s < 9
    bne $t2, $zero, sub_done # If s < 9, then we are done
    addi $t0, $t0, -9 # Subtract 9 from s
    addi $t1, $t1, 1 # Increment result
    j sub_loop

sub_done:
    # Print result message
    li $v0, 4
    la $a0, resultStr
    syscall

    # Print result
```

```
    li $v0, 1
    move $a0, $t1
    syscall

    # Exit program
    li $v0, 10
    syscall
```

Sample Run

```
Enter s (s mod 9 != 0): 25
s div 9 = 2
```

Problem 2: Insertion Sort of 12 Integers

Pseudocode

```
BEGIN
    Create an array of size 12.
    FOR i from 0 to 11 DO
        Print "Enter number: "
        Read integer into array[i]
    END FOR

    // Insertion Sort
    FOR i from 1 to 11 DO
        key = array[i]
        j = i - 1
        WHILE j >= 0 AND array[j] > key DO
            array[j + 1] = array[j]
            j = j - 1
        END WHILE
        array[j + 1] = key
    END FOR

    FOR i from 0 to 11 DO
        Print array[i] followed by a newline
    END FOR
END
```

Assembly Code

```
.data
array:    .space 48 # Reserve space for 12 integers (12 * 4 bytes)
prompt:   .asciiz "Enter number: "
newline:  .asciiz "\n"

.text
.globl main

main:
    li $t0, 0
    la $t1, array

input_loop:
    bge $t0, 12, start_sort

    li $v0, 4
    la $a0, prompt
    syscall

    li $v0, 5
    syscall
    move $t2, $v0

    sll $t3, $t0, 2
    add $t4, $t1, $t3
    sw $t2, 0($t4)

    addi $t0, $t0, 1
    j input_loop

start_sort:
    li $t0, 1 # Outer loop index
    la $t1, array # Array address
```

```
sort_outer:
    bge $t0, 12, print_array # If outer loop index >= 12, then print array

    sll $t2, $t0, 2
    add $t3, $t1, $t2 # Address of key
    lw $t4, 0($t3) # Load key into $t4
    move $t5, $t0 # Inner loop index

sort_inner:
    addi $t5, $t5, -1 # Decrement inner loop index
    blt $t5, $zero, insert_key # If inner loop index < 0, then insert key

    sll $t6, $t5, 2
    add $t7, $t1, $t6  # Address of element to compare
    lw $t8, 0($t7) # Load element
    bgt $t8, $t4, shift_element # If element > key, shift it
    j insert_key # Otherwise, insert key into $t5 + 1

shift_element:
    addi $t9, $t5, 1 # New index of element
    sll $t9, $t9, 2
    add $t9, $t1, $t9 # Address of new index
    sw $t8, 0($t9) # Shift element to new index
    j sort_inner

insert_key:
    addi $t5, $t5, 1 # Store key in $t5 + 1
    sll $t6, $t5, 2
    add $t6, $t1, $t6 # Address of key position
    sw $t4, 0($t6) # Insert key into correct position
    addi $t0, $t0, 1 # Increment outer loop index
    j sort_outer

print_array:
    li $t0, 0
    la $t1, array

print_loop:
    bge $t0, 12, exit_program

    sll $t2, $t0, 2
    add $t3, $t1, $t2
    lw $t4, 0($t3)
    li $v0, 1
    move $a0, $t4
    syscall

    li $v0, 4
    la $a0, newline
    syscall

    addi $t0, $t0, 1
    j print_loop

exit_program:
    li $v0, 10
    syscall
```

Sample Input and Output

```
Enter number: 12
Enter number: 11
...
Enter number: 1

1
2
3
...
12
```

Problem 3: Finding Pairs that Sum to a Target Value

Pseudocode

```
BEGIN
    Create an array of size 16.
    FOR i from 0 to 15 DO
        Print "Enter number: "; Read integer into array[i]
    END FOR

    Print "Target: "
    Read target value

    FOR i from 0 to 15 DO
        FOR j from i+1 to 15 DO
            IF (array[i] + array[j] == target) THEN
                Print array[i], a space, and array[j] followed by a newline
            END IF
        END FOR
    END FOR
END
```

Assembly Code

```
.data
array:    .space 64 # Reserve space for 16 integers (16 x 4 bytes)
newline:  .asciiz "\n"
space:    .asciiz " "
prompt1:  .asciiz "Enter number: "
prompt2:  .asciiz "Target: "

.text
.globl main

main:
    li $t0, 0  # Outer index i = 0
    la $t1, array

read_loop:
    bge $t0, 16, ask_target
    li $v0, 4
    la $a0, prompt1
    syscall

    li $v0, 5
    syscall

    sll $t2, $t0, 2
    add $t9, $t1, $t2
    sw $v0, 0($t9)

    addi $t0, $t0, 1
    j read_loop

ask_target:
    li $v0, 4
    la $a0, prompt2
    syscall

    li $v0, 5
    syscall
    move $s0, $v0
```

```
    li $t0, 0 # Outer index i = 0

outer_loop:
    bge $t0, 16, exit_program # If i >= 16, then exit program

    sll $t2, $t0, 2
    add $t9, $t1, $t2 # Address of element in index i
    lw $t4, 0($t9) # Load element

    addi $t8, $t0, 1 # Inner index j = i + 1

inner_loop:
    bge $t8, 16, next_outer # If j >= 16, then go to next outer loop

    sll $t2, $t8, 2
    add $t9, $t1, $t2 # Address of element in index j
    lw $t5, 0($t9) # Load element

    add $t7, $t4, $t5 # Sum of elements in index i and j
    beq $t7, $s0, print_pair # If sum = target, then print pair

    addi $t8, $t8, 1 # Increment inner index
    j inner_loop

print_pair:
    li $v0, 1
    move $a0, $t4
    syscall

    li $v0, 4
    la $a0, space
    syscall

    li $v0, 1
    move $a0, $t5
    syscall

    li $v0, 4
    la $a0, newline
    syscall

next_outer:
    addi $t0, $t0, 1 # Increment outer index
    j outer_loop # Jump

exit_program:
    li $v0, 10
    syscall
```

Sample Input and Output

```
Enter number: 1
Enter number: 2
...
Enter number: 16
Target: 13

1 12
2 11
3 10
4 9
5 8
6 7
```

Problem 4: Floating-Point Conversion (Part i)

Pseudocode

```
BEGIN
    Prompt "Enter a 32-bit binary number (A): "
    Read binary string into BUFFER
    CALL convert:
        Set ACCUMULATOR = 0
        For each character in the binary string:
            If character is newline or null, exit loop
            Shift ACCUMULATOR left by 1
            If character is '1', set the least significant bit of ACCUMULATOR
        End loop (ACCUMULATOR now holds the 32-bit integer value)

        Extract exponent:
            Discard the sign bit and the mantissa bits
            Shift appropriately and subtract 127 to unbias the exponent
        Extract mantissa:
            Remove sign and exponent bits from ACCUMULATOR
            Add implicit 1 to form the SIGNIFICAND

        Compute DIVISOR = 2^(23 - exponent)
        Divide SIGNIFICAND by DIVISOR:
            QUOTIENT becomes the integer part
            REMAINDER is used for fractional conversion

        Initialize digit counter to 0 and FRACTION accumulator to 0
        WHILE digit counter < 7 DO:
            Multiply REMAINDER by 10
            Divide by DIVISOR to get next digit and update REMAINDER
            Append digit to FRACTION accumulator
            Increment digit counter
        END WHILE
        Return QUOTIENT (integer part) and FRACTION (fractional part)
    END CALL
    Print converted value for A as "integer part.fractional part"

    Prompt "Enter a 32-bit binary number (B): "
    Read binary string into BUFFER
    CALL convert (as above) for B
    Print converted value for B as "integer part.fractional part"
END
```

Assembly Code

```
.data
buffer:    .space 33              # Space for 32-bit binary string + null terminator
prompt:    .asciiz "Enter a 32-bit binary number (A): "
prompt2:   .asciiz "Enter a 32-bit binary number (B): "
newline:   .asciiz "\n"
dot:       .asciiz "."

.text
.globl main

#------------------------------------------------------------
# Function: convert
# Input:  $a0 = pointer to the binary string
# Output: $v0 = integer part, $v1 = fractional part
#------------------------------------------------------------
convert:
    # Save return address ($ra) on stack
    addi    $sp, $sp, -4
    sw      $ra, 0($sp)

    # Convert the binary string to a 32-bit integer
    li      $t0, 0       # Accumulator for binary value
    move    $t1, $a0     # Input string

conv_loop:
    lb      $t2, 0($t1)  # Load character

    # If newline or null terminator, done
    beq     $t2, 10, conv_done_loop
    beq     $t2, 0, conv_done_loop

    sll     $t0, $t0, 1  # Shift left for next bit
    li      $t3, '1'
    beq     $t2, $t3, conv_set_bit # Set bit if character is '1'
    j       conv_next_char

conv_set_bit:
    ori     $t0, $t0, 1  # Set LSB if character is '1'

conv_next_char:
    addi    $t1, $t1, 1  # Next character
    j       conv_loop

conv_done_loop:
    # At this point $t0 holds the 32-bit binary value

    # Extract exponent to $t4
    move    $t4, $t0
    sll     $t4, $t4, 1    # Discard sign bit
    srl     $t4, $t4, 24   # Discard mantissa bits
    addi    $t4, $t4, -127 # Unbias exponent.

    # Extract raw mantissa (removing sign and exponent) to $t5
    move    $t5, $t0
    sll     $t5, $t5, 9   # Discard sign and exponent bits
    srl     $t5, $t5, 9   # Move back to the right

    # Compute significand = (2^23 + mantissa) and store to $t5
    li      $t6, 1
    sll     $t6, $t6, 23  # $t6 = 2^23
    add     $t5, $t5, $t6 # $t5 = significand (with implicit 1)
```

```
    # Compute divisor = 2^(23 - exponent) and store to $t6
    li      $t6, 1       # Reinitialize $t6.
    li      $t7, 23
    sub     $t7, $t7, $t4  # $t7 = 23 - exponent.
    sll     $t6, $t6, $t7  # $t6 = 2^(23 - exponent).

    # Divide significand by divisor
    div     $t5, $t6
    mflo    $t8        # $t8 = integer part (quotient)
    mfhi    $t9        # $t9 = remainder

    # Compute the fractional part
    li      $t7, 0     # Digit counter
    li      $t1, 0     # Fractional accumulator

conv_fraction_loop:
    addi    $t7, $t7, 1    # Increment digit counter
    mul     $t9, $t9, 10   # Multiply remainder by 10

    div     $t9, $t6
    mflo    $t2          # Next digit
    mfhi    $t9          # Update remainder

    mul     $t1, $t1, 10 # Shift accumulator
    add     $t1, $t1, $t2  # Add new digit
    bne     $t7, 7, conv_fraction_loop # Stop at 7 digits

    # Return: integer part in $v0, fractional part in $v1
    move    $v0, $t8
    move    $v1, $t1

    lw      $ra, 0($sp)  # Restore return address
    addi    $sp, $sp, 4  # Restore stack pointer
    jr      $ra          # Return

main:
    li      $v0, 4
    la      $a0, prompt
    syscall

    li      $v0, 8
    la      $a0, buffer
    li      $a1, 33
    syscall

    # Call conversion function for first input
    la      $a0, buffer
    jal     convert

    # Save conversion results
    move    $t0, $v0   # Integer part of A
    move    $t1, $v1   # Fractional part of A

    li      $v0, 4
    la      $a0, newline
    syscall

    # Print first converted value: integer part . fractional part.
    li      $v0, 1
    move    $a0, $t0
    syscall
```

```
    li      $v0, 4
    la      $a0, dot
    syscall

    li      $v0, 1
    move    $a0, $t1
    syscall

    li      $v0, 4
    la      $a0, newline
    syscall

    li      $v0, 4
    la      $a0, prompt2
    syscall

    li      $v0, 8
    la      $a0, buffer
    li      $a1, 33
    syscall

    la      $a0, buffer
    jal     convert
    move    $t2, $v0   # Integer part of B
    move    $t3, $v1   # Fractional part of B

    li      $v0, 4
    la      $a0, newline
    syscall

    # Print second converted value: integer part . fractional part.
    li      $v0, 1
    move    $a0, $t2
    syscall

    li      $v0, 4
    la      $a0, dot
    syscall

    li      $v0, 1
    move    $a0, $t3
    syscall

    # End program
    li      $v0, 10
    syscall
```

Sample Input and Output

```
Enter a 32-bit binary number (A): 01000000111011100001000000000000
7.4394531
Enter a 32-bit binary number (B): 01001011010100010001001010100000
13701792.0
```