Mid-term Report

Nguyen Tieu Phuong 20210692

6th November 2023

Problem

Write a program that gets an integer i from the user and creates the table shown below on the screen (example inputs provided). Subroutines are required for power, square, and hexadecimal (in 32 bit arithmetic, attend to overflowed results). Hint: Hexadecimal can be done with shifts and masks because the size is 32 bits.

i	power(2,i)	square(i)	Hexadecimal(i)
10	1024	100	0xA
7	128	49	0x7
16	65536	256	0x10

An example.

Solution

User input

For getting the user input, we can simply display a dialog that reads an integer. For simplicity, this program only supports positive integers (greater than zero). We can accomplish this by implementing a prior check for validity of the input. If this condition is not satisfied, an error output is produced and the program quits.

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

```
# Read the user input
li          $v0, 5
syscall
move        $s0, $v0

# Check for valid input
blez        $s0, exit
nop
```

Calculating power(2, i)

A condition check is necessary. Since the maximum possible integer supported by MIPS is only (2ˆ31 - 1), we find the upper bound for the possible value of i to be log(2) of (2ˆ31 - 1), which yields the value 30.

```
li      $t0, 30                  # The maximum possible input value
bgt     $s0, $t0, overflow_pow   # Overflow if input greater than 30
```

To calculate the power(2, i), we create a loop that multiplies the number by 2 for i times. The multiplication is best optimized by using the bit shifting operation.

```
#--------------------------------------------------------------------
# Procedure pow
# @brief              raise 2 to the power of i
# @param[in]          s0               the power
# @param[out]         s1               2^i
#--------------------------------------------------------------------

pow:  li          $s1, 1           # init $s1, 2^0 = 1
      li          $t1, 0           # increment counter
      pow_loop:   sll        $s1, $s1, 1      # multiply i by 2
                  j          test             # done? check condition
      test:       addi       $t1, $t1, 1      # advance the index j
                  slt        $t2, $t1, $s0    # set $t2 to 1 if j < i
                  bne        $t2, $0, pow_loop  # repeat if j < i
      end_pow_loop:    jr      $ra
```

Calculating square(i)

This subroutine is fairly simple. We use a `mul` instruction to multiply `i` with itself - and save the result to a register.

```
#--------------------------------------------------------------------
# Procedure square
# @brief              return i*i
# @param[in]          s0               the number i
# @param[out]         s2               the result i*i
#--------------------------------------------------------------------
```

```
square:    mul     $s2, $s0, $s0
           jr      $ra
```

A condition check is necessary. Since the maximum possible integer supported by MIPS is only (2^31 - 1), we find the upper bound for the possible value of i to be the square root of (2^31 - 1), which yields the value 46340.

```
li      $t0, 46340              # The maximum possible input value
bgt     $s0, $t0, overflow_sq
```

Calculating hexadecimal(i)

```
#--------------------------------------------------------------------
# Procedure hex
# @brief               print i in hexadecimal, digit by digit
# @param[in]           s0               the number i
# @param[out]          s3               the hexadecimal value of i
#--------------------------------------------------------------------

hex:  li      $t1, 28       # set counter to 28
      li      $t2, 0x0F     # set mask to 0x0F
convert_loop:    srlv    $t3, $s0, $t1    # shift user input right
                 and     $t3, $t3, $t2    # mask the lower 4 bits
# Convert remainder to hex digit
      addi    $t3, $t3, 48        # convert to ASCII digit
      blt     $t3, 58, skip       # skip if digit is less than '9'
      addi    $t3, $t3, 7         # adjust for letters 'A' to 'F'
skip:
continue:  li    $v0, 11          # Print hex digit
           move  $a0, $t3
           syscall
           subi    $t1, $t1, 4   # Decrement counter by 4 bits
           addi     $t9, $0, -4
           bne     $t1, $t9, convert_loop  # Check if done

done:      jr      $ra
```

Hexadecimal representation of i can be calculated by, firstly, processing using 2 techniques: masking and shifting.

- Masking is for isolating 8 4-bit groups of a 32-bit long integer in MIPS, and

converting each group into the corresponding hex character.
- Shifting is for manipulating the original input to prepare for masking.

Upon successful isolation, we convert the number onto the hex digit by referring to the ASCII digit, i.e. the ASCII code of the characters. For digits from 0 to 9, the ASCII code can be obtained by adding 48, the ASCII code for '0'. For digits from A to F, to get the suitable alphabet, we add 7, the number needed to go from the chunk of ASCII number to ASCII uppercase.

Output formatting

To print a proper table, we set up several headers and formatting strings. It is necessary to pay close attention to the number of tabs, so that the output lined up together.

Souce code

- The first instruction under the label should be written in the same line for easy debugging after compilation.
- The instructions inside a label are written in the same indentation (not more) than the label.
- Remember to NOT use pseudo-code in the theory exam.

```
.data

prompt: .asciiz "\nEnter an integer:"

header: .asciiz "\ni\t\tpower(2,i)\tsquare(i)\thex(i)\n"

tab: .asciiz "\t\t"

error: .asciiz "OVF\t\t"

input_err: .asciiz "\nInvalid input"

.text

.globl main

main:

# Print the prompt

li $v0, 4

la $a0, prompt
```

```
syscall


# Read the user input
li $v0, 5
syscall
move $s0, $v0
# Check for valid input
blez $s0, exit
nop
proceed:
# Print the table header
li $v0, 4
la $a0, header
syscall


# Call the subroutine
jal print_table


print_table:
# Print i
li $v0, 1
move $a0, $s0
syscall
li $v0, 4
la $a0, tab
syscall
```

```
# pow(2,i)

li $t0, 30 # The maximum possible input value for pow (2^30 = 1073741824 < 2^31-1)

bgt $s0, $t0, overflow_pow # If the input is greater than $t0, the result will
overflow

nop

jal pow # Else goto pow

j print_pow

overflow_pow:

li $v0, 4

la $a0, error

syscall

j end_pow

print_pow:

li $v0, 1

move $a0, $s1

syscall

li $v0, 4

la $a0, tab

syscall

end_pow:



# square(i)

li $t0, 46340 # The maximum possible input value for square (sqrt of max int:
2^31-1)

bgt $s0, $t0, overflow_sq # If the input is greater than $t0, the result will
overflow
```

```
nop

jal square # else goto square

j print_sq

overflow_sq:

li $v0, 4

la $a0, error

syscall

j end_sq

print_sq:

li $v0, 1

move $a0, $s2

syscall

li $v0, 4

la $a0, tab

syscall

end_sq:


# hex(i)

jal hex

finish:

li $v0, 10

syscall



# ------------------------------------------------------------------------------

# Procedure pow

# @brief raise 2 to the power of i
```

```
# @param[in] s0 the power that 2 will be raised into

# @param[out] s1 2^i

# ------------------------------------------------------------------------------

pow:

li $s1, 1 # init $s1, 2^0 = 1

li $t1, 0 # increment counter

pow_loop:

sll $s1, $s1, 1 # multiply i by 2

j test # done? check condition

test:

addi $t1, $t1, 1 # advance the index j

slt $t2, $t1, $s0 # set $t2 to 1 if j < i

bne $t2, $0, pow_loop # repeat if j < i

end_pow_loop:

jr $ra



# ------------------------------------------------------------------------------

# Procedure square

# @brief return i*i

# @param[in] s0 the number i

# @param[out] s2 the result i*i

# ------------------------------------------------------------------------------

square:

# Calculate square(i)

mul $s2, $s0, $s0

jr $ra
```

```
# --------------------------------------------------------------------------------
# Procedure hex
# @brief print i in hexadecimal, digit by digit
# @param[in] s0 the number i
# @param[out] s3 the hexadecimal value of i
# --------------------------------------------------------------------------------
hex:
# Convert decimal to hex
li $t1, 28 # set counter to 28 (number of bits in a decimal integer)
li $t2, 0x0F # set mask to 0x0F (to extract the lower 4 bits)
convert_loop:
srlv $t3, $s0, $t1 # shift user input right by counter bits
and $t3, $t3, $t2 # mask the lower 4 bits

# Convert remainder to hex digit
addi $t3, $t3, 48 # convert to ASCII digit by add '0'
blt $t3, 58, skip # skip if digit is less than '9'
addi $t3, $t3, 7 # adjust for letters 'A' to 'F'
skip:
continue:
# Print hex digit
li $v0, 11
move $a0, $t3
syscall
subi $t1, $t1, 4 # Decrement counter by 4 bits
```

```
addi $t9, $0, -4

bne $t1, $t9, convert_loop # Check if counter is -4 (done)

done:

jr $ra



exit:

li $v0, 4

la $a0, input_err

syscall

li $v0, 10

syscall
```

THE END