Exercise C

bin\_and\_hex.asm

# bin\_and\_hex.asm

# ENCM 369 Winter 2016 Lab 5 Exercise C Partial Solution

#

# Author: S. Norman

# BEGINNING of start-up & clean-up code. Do NOT edit this code.

.data

exit\_msg\_1:

.asciiz "\*\*\*About to exit. main returned "

exit\_msg\_2:

.asciiz ".\*\*\*\n"

main\_rv:

.word 0

.text

# adjust $sp, then call main

addi $t0, $zero, -32 # $t0 = 0xffffffe0

and $sp, $sp, $t0 # round $sp down to multiple of 32

jal main

nop

# when main is done, print its return value, then halt the program

sw $v0, main\_rv

la $a0, exit\_msg\_1

addi $v0, $zero, 4

syscall

nop

lw $a0, main\_rv

addi $v0, $zero, 1

syscall

nop

la $a0, exit\_msg\_2

addi $v0, $zero, 4

syscall

nop

addi $v0, $zero, 10

syscall

nop

# END of start-up & clean-up code.

# int main(void)

#

.text

.globl main

main:

addi $sp, $sp, -32

sw $ra, 0($sp)

li $a0, 0x76543210

jal test

li $a0, 0x89abcdef

jal test

li $a0, 0

jal test

li $a0, -1

jal test

add $v0, $zero, $zero # r.v. = 0

lw $ra, 0($sp)

addi $sp, $sp, 32

jr $ra

# void test(int test\_value)

#

# arg / var memory location

# test\_value 44($sp)

# char str[40] 40 bytes starting at 0($sp)

#

.data

STR1: .asciiz "\n\n"

.text

.globl test

test:

addi $sp, $sp, -64

sw $a0, 44($sp)

sw $ra, 40($sp)

addi $a0, $sp, 0 # $a0 = &str[0]

lw $a1, 44($sp) # $a1 = test\_value

jal write\_in\_hex

addi $a0, $sp, 0 # $a0 = &str[0]

addi $v0, $zero, 4 # $v0 = code to print a string

syscall

addi $a0, $zero, '\n' # $a0 = '\n'

addi $v0, $zero, 11 # $v0 = code to print a char

syscall

addi $a0, $sp, 0 # $a0 = &str[0]

lw $a1, 44($sp) # $a1 = test\_value

jal write\_in\_binary

addi $a0, $sp, 0 # $a0 = &str[0]

addi $v0, $zero, 4 # $v0 = code to print a string

syscall

la $a0, STR1 # $a0 = STR1

addi $v0, $zero, 4 # $v0 = code to print a string

syscall

lw $ra, 40($sp)

addi $sp, $sp, 64

jr $ra

# void write\_in\_hex(char \*str, int word)

#

# arg / var register

# str $a0

# word $a1

# digit\_list $t9

#

.data

hex\_digits:

.asciiz "0123456789abcdef"

.text

.globl write\_in\_hex

write\_in\_hex:

ori $t0, $zero, '0'

sb $t0, 0($a0) # str[0] = '0'

ori $t0, $zero, 'x'

sb $t0, 1($a0) # str[1] = 'x'

ori $t0, $zero, '\_'

sb $t0, 6($a0) # str[6] = '\_'

sb $zero, 11($a0) # str[11] = '\0'

la $t9, hex\_digits # digit\_list = hex\_digits

srl $t1, $a1, 28 # $t1 = word >> 28

andi $t2, $t1, 0xf # $t2 = $t1 & 0xf

add $t3, $t9, $t2 # $t3 = &digit\_list[$t2]

lb $t4, ($t3) # $t4 = digit\_list[$t2]

sb $t4, 2($a0) # str[2] = $t4

srl $t1, $a1, 24 # $t1 = word >> 24

andi $t2, $t1, 0xf # $t2 = $t1 & 0xf

add $t3, $t9, $t2 # $t3 = &digit\_list[$t2]

lb $t4, ($t3) # $t4 = digit\_list[$t2]

sb $t4, 3($a0) # str[3] = $t4

srl $t1, $a1, 20 # $t1 = word >> 20

andi $t2, $t1, 0xf # $t2 = $t1 & 0xf

add $t3, $t9, $t2 # $t3 = &digit\_list[$t2]

lb $t4, ($t3) # $t4 = digit\_list[$t2]

sb $t4, 4($a0) # str[4] = $t4

srl $t1, $a1, 16 # $t1 = word >> 16

andi $t2, $t1, 0xf # $t2 = $t1 & 0xf

add $t3, $t9, $t2 # $t3 = &digit\_list[$t2]

lb $t4, ($t3) # $t4 = digit\_list[$t2]

sb $t4, 5($a0) # str[5] = $t4

srl $t1, $a1, 12 # $t1 = word >> 12

andi $t2, $t1, 0xf # $t2 = $t1 & 0xf

add $t3, $t9, $t2 # $t3 = &digit\_list[$t2]

lb $t4, ($t3) # $t4 = digit\_list[$t2]

sb $t4, 7($a0) # str[7] = $t4

srl $t1, $a1, 8 # $t1 = word >> 8

andi $t2, $t1, 0xf # $t2 = $t1 & 0xf

add $t3, $t9, $t2 # $t3 = &digit\_list[$t2]

lb $t4, ($t3) # $t4 = digit\_list[$t2]

sb $t4, 8($a0) # str[8] = $t4

srl $t1, $a1, 4 # $t1 = word >> 4

andi $t2, $t1, 0xf # $t2 = $t1 & 0xf

add $t3, $t9, $t2 # $t3 = &digit\_list[$t2]

lb $t4, ($t3) # $t4 = digit\_list[$t2]

sb $t4, 9($a0) # str[9] = $t4

andi $t2, $a1, 0xf # $t2 = word & 0xf

add $t3, $t9, $t2 # $t3 = &digit\_list[$t2]

lb $t4, ($t3) # $t4 = digit\_list[$t2]

sb $t4, 10($a0) # str[10] = $t4

jr $ra

# write\_in\_binary(char \*str, int word)

#

# Students have to replace the code for this procedure

# with code that implements the given C code.

.text

.globl write\_in\_binary

write\_in\_binary:

# underscore $t0

# digit0 $t1

# digit1 $t2

# bn $t3

# p $t4

# str $a0

# word $a1

addi $t0, $zero, '\_' # underscore = '\_'

addi $t1, $zero, '0' # digit0 = '0'

addi $t2, $zero, '1' # digit1 = '1'

sb $zero, 39($a0) # str[39] = '\0'

add $t3, $zero, $zero # bn = 0

addi $t5, $zero, 39 # $t5 = 39

add $t4, $a0, $t5 # p = str + 39

L1: addi $t4, $t4, -1 # p--

andi $t5, $a1, 1 # $t5 = word & 1

bne $t5, $zero, L2 # if ($t5 != 0) goto L2

sb $t1, 0($t4) # \*p = digit0

j L3 # goto L3

L2: sb $t2, 0($t4) # \*p = digit1

L3: addi $t5, $zero, 31 # $t5 = 31

beq $t3, $t5, L5 # if (bn == $t5) goto L5

andi $t5, $t3, 3 # $t5 = bn & 3

addi $t6, $zero, 3 # $t6 = 3

bne $t5, $t6, L4 # if ($t5 != $t6) goto L4

addi $t4, $t4, -1 # p--

sb $t0, 0($t4) # \*p = underscore

L4: addi $t3, $t3, 1 # bn++

srl $a1, $a1, 1 # word = word >> 1

j L1 # goto L1

L5: jr $ra

Exercise D

L1: lbu $t9, ($s1) # 0x0040\_1034

beq $t9, $zero, L2 # 0x0040\_1038

. . . # 26 instructions

addi $s1, $s1, 1 # 0x0040\_10a4

j L1 # 0x0040\_10a8

L2: or $s3, $s1, $zero # 0x0040\_10ac

beq $t9, $zero, L2:

$t9 is the 25th register

$zero is the 0th register

(0x004010ac-0x0040103c) = 0x70 = 112 = 28 words

beq is an I-type instruction with an opcode of 4 (Table B.1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| op | rs | rt | imm |  |
| 000100 | 11001 | 00000 | 0000000000011100 | = 0x1320001c |
| 4 | 25 | 0 | 28 |  |

j L1:

The instruction L1 refers to has an address of 0x00401034 (4198452)

j is a J-type instruction with an opcode of 2

|  |  |  |
| --- | --- | --- |
| op | addr |  |
| 000010 | 00010000000001000000110100 | = 0x08401034 |
| 2 | 4198452 |  |

Exercise E

CPU Times for compilation with gcc main.c functions.c

|  |  |
| --- | --- |
| index\_version (s) | pointer\_version (s) |
| 10.7470000000 | **9.2200000000** |
| **10.6860000000** | 9.4220000000 |
| **10.6850000000** | 9.5000000000 |
| 10.7320000000 | 9.3600000000 |
| **10.6860000000** | 9.3440000000 |
| 10.7630000000 | **9.1580000000** |
| 10.7170000000 | **9.1410000000** |
| **10.5920000000** | **9.3280000000** |
| **10.6540000000** | 9.4860000000 |
| 10.9500000000 | **9.3140000000** |
| Average: 10.6606000000 | Average: 9.2322000000 |

How much time was used by index\_version?

10.661 seconds

How much time was used by pointer\_version?

9.232 seconds

CPU Times for compilation with gcc –O2 main.c functions.c

|  |  |
| --- | --- |
| index\_version (s) | pointer\_version (2) |
| 1.7010000000 | 1.7000000000 |
| 1.7160000000 | 1.7000000000 |
| 1.7000000000 | 1.7000000000 |
| 1.7000000000 | 1.7160000000 |
| **1.7000000000** | 1.7160000000 |
| **1.7000000000** | **1.7000000000** |
| **1.7000000000** | **1.7000000000** |
| **1.7000000000** | **1.7000000000** |
| **1.7000000000** | **1.7000000000** |
| 1.7160000000 | **1.6840000000** |
| Average: 1.7000000000 | Average: 1.6968000000 |

With optimization, pointer\_version is not significantly faster than index\_version

Asking for optimization is a more important factor for array-processing speed.

CPU Times for compilation with gcc –O2 –funroll-loops main.c functions.c

|  |  |
| --- | --- |
| index\_version (s) | pointer\_version (s) |
| **1.2940000000** | 1.3110000000 |
| 1.3100000000 | **1.2950000000** |
| 1.3100000000 | **1.3100000000** |
| 1.3100000000 | **1.3100000000** |
| 1.3100000000 | 1.3260000000 |
| **1.2940000000** | **1.3110000000** |
| 1.3100000000 | **1.3100000000** |
| **1.2940000000** | 1.3260000000 |
| **1.3100000000** | 1.3260000000 |
| **1.3100000000** | 1.3260000000 |
| Average: 1.3004000000 | Average: 1.3072000000 |

Exercise F

Loop instructions:

movl $0, -4(%rbp)

jmp .L2

.L3:

movl -4(%rbp), %eax

cltq

leaq 0(,%rax,4), %rdx

movq 16(%rbp), %rax

addq %rdx, %rax

movl (%rax), %eax

addl %eax, -8(%rbp)

addl $1, -4(%rbp)

.L2:

movl -4(%rbp), %eax

cmpl 24(%rbp), %eax

jl .L3

The machine code for movl $0x0,-0x8(%rbp) is 0xc745f800000000 (line 0xf)

for (i = 0; i < ARRAY\_SIZE; i++) gets replaced with

for (i = 0; i < 4000; i++)

The machine code for cmpl $0xf9f,-0x4(%rbp) is 0x817dfc9f0f0000 (line 0x43)