Insert Name Here (change something, at least so it’s not identical)

CS 47

HW #2

1. Problem #1

.include "cs47\_macro.asm"

.data

prompta: .asciiz "Please enter a positive integer: "

promptb: .asciiz "A number greater than 0. It's Elementary School, Watson.\n"

resulta: .asciiz "You're answer is: "

resultb: .asciiz "On Index "

resultc: .asciiz " fibs are "

resultd: .asciiz ", "

newl: .asciiz "\n"

.text

begin:

print\_str(prompta)

read\_int($a1)

addi $t2, $zero, 2

div $a1, $t2

mflo $a1

mfhi $t2

print\_str(newl)

bltz $a1, bzzz

addi $t0, $zero, 0

addi $t1, $zero, 1

addi $sp, $sp, -4

sw $t0, ($sp)

addi $sp, $sp, -4

sw $t1, ($sp)

addi $a1, $a1, -1

fibloop:

add $t0, $t0, $t1

add $t1, $t1, $t0

addi $sp, $sp, -4

sw $t0, ($sp)

addi $sp, $sp, -4

sw $t1, ($sp)

addi $a1, $a1, -1

bgez $a1, fibloop

move $sp, $t1

done:

beqz $t2, other

print\_str(resulta)

print\_reg\_int($t1)

j news

other:

print\_str(resulta)

print\_reg\_int($t0)

news:

print\_str(newl)

print\_str(newl)

j begin

bzzz:

print\_str(promptb)

j begin

1. Problem #2

#$t0 is general use number

#=========================

#$a1 is index of var\_a

#$a2 is index of var\_b

#$a3 is index of var\_c

#=========================

#$s1 is length of var\_a

#$s2 is length of var\_b

#=========================

#$t1 is var\_a

#$t2 is var\_b

#$t3 is var\_c

#=========================

.include "cs47\_macro.asm"

.data

.align 2

var\_a: .word 2 3 5 5 8 10 11 17 18 20

var\_b: .word 5 6 7 8 14 15 17

var\_m: .word 10

var\_n: .word 7

var\_c: .word 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

newl: .asciiz "\n"

.text

.globl start

start:

add $a1, $zero, $zero #index of var\_a

add $a2, $zero, $zero #index of var\_b

add $a3, $zero, $zero #index of var\_c

lw $s1, var\_m #length of var\_a

lw $s2, var\_n #length of var\_b

loopcheck:

sub $t0, $s1, $a1 #If index of a < length of a...

bgtz $t0, check #Jump to second check, else...

sub $t0, $s2, $a2 #If index of b < length of b...

bgtz $t0, addbtoc #Jump to add b to c as a is empty, else...

blez $t0, bbye #Since both arrays are empty, exit.

check:

sub $t0, $s2, $a2 #Since a is not empty, and if index of b < length of b...

bgtz $t0, looproutine #Jump to routine, else...

j addatoc #Since a is not empty, and b is empty, add a to c.

looproutine:

la $t1, var\_a #Load var\_a array into $t1.

mul $t0, $a1, 4 #Get pointer reference by mult index by 4.

add $t0, $t1, $t0 #Go to the index from pointer in array.

lw $t1, ($t0) #Get value from index in var\_a, store into $t1.

#Loaded val of var\_a at current index of $a1 into $t1

la $t2, var\_b #Load var\_b array into $t2.

mul $t0, $a2, 4 #Get pointer reference by mult index by 4.

add $t0, $t2, $t0 #Go to the index from pointer in array.

lw $t2, ($t0) #Get value from index in var\_b, store into $t2.

#Loaded val of var\_b at current index of $a2 into $t2

sub $t0, $t1, $t2

bgtz $t0, addbtoc

blez $t0, addatoc

#If a-b is greater than 0, a is bigger add b to c, else a is smaller, add a to c

addbtoc:

move $t4, $t2

add $a2, $a2, 1 #add 1 to index of var\_b

j addtoc

addatoc:

move $t4, $t1

add $a1, $a1, 1 #add 1 to index of var\_a

j addtoc

addtoc:

la $t3, var\_c

mul $t0, $a3, 4

add $t0, $t3, $t0

sw $t4, ($t0) #stores the value $t4 from the merge logic into $t3 ($t0)

add $a3, $a3, 1 #index of var\_c

j loopcheck

bbye:

li $v0, 10

syscall

1. 2’s Complement, 610 and -310:
   1. 610 = 01102

-310 = 11012

* 1. 00000110

x 11111101

00000000110

00000000000

00000011000

00000110000

00001100000

00011000000

00110000000

+ 01100000000

101111011102  (convert to only 8-bits) -> 111011102 = -1810

* 1. Extend to 8-Bit:
     1. Zero Extend:

610 → 000001102 → 610

-310 → 000011012 → 1310

* + 1. Signed Extend:

610 → 000001102 → 610

-310 → 111111012 → -310

1. Truth Tables:
   * 1. F(x, y, z) = (xy)' + z

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **X** | **Y** | **Z** | **XY** | **(XY)’** | **(XY)’+ Z** |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 |

* 1. F(x, y, z) = (x'yz') + (xy'z)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X** | **Y** | **Z** | **X’** | **Y’** | **Z’** | **X’YZ’** | **XY’Z** | **F(X,Y,Z)** |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

1. Boolean Arithmatic Manipulation:
   1. x’y’z’ + xy’z’ + x’yz’ + xyz’ =

z’(x’y’ + xy’ + x’y + xy) =

z’((x + x’)y’ + (x + x’)y) =

z’((1)y’ + (1)y) =

z’(y’ + y) =

z’(1) = z’

* 1. (a’ + c)(a’ + d’)(b + c)(b + d’) =

(a’ + cd’)(b + cd’) =

a’b + cd’

1. Karnaugh Maps:
   1. f (A, B, C, D) = ∑ m (1, 2, 3, 4, 6, 7, 9, 11, 12, 13, 14, 15)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB\CD** | **00** | **01** | **11** | **10** |
| **00** | 0 (0) | 1 (1) | 1 (3) | 1 (2) |
| **01** | 1 (4) | 0 (5) | 1 (7) | 1 (6) |
| **11** | 1 (C) | 1 (D) | 1 (E) | 1 (F) |
| **10** | 0 (8) | 1 (9) | 1 (B) | 0 (A) |

Prime Implicants: AD, BC, A’C, AB, CD, B’D, BD’

Essential Prime Implicants: BD’, B’D, A’C

SOP Expression: BD’ + B’D + A’C

* 1. f (w, x, y, z) = ∑ m (0, 5, 10, 15) + d(2, 7, 8, 13)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **WX\YZ** | **00** | **01** | **11** | **10** |
| **00** | 1 (0) | 0 (1) | 0 (3) | X (2) |
| **01** | 0 (4) | 1 (5) | X (7) | 0 (6) |
| **11** | 0 (C) | X (D) | 1 (E) | 0 (F) |
| **10** | X (8) | 0 (9) | 0 (B) | 1 (A) |

Prime Implicants: XZ, X’Z’

Essential Prime Implicants: XZ, X’Z’

SOP Expression: XZ + X’Z’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | Is Single Digit |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB\CD** | **00** | **01** | **11** | **10** |
| **00** | 1 (0) | 1 (1) | 1 (3) | 1 (2) |
| **01** | 1 (4) | 1 (5) | 1 (7) | 1 (6) |
| **11** | 0 (C) | 0 (D) | 0 (E) | 0 (F) |
| **10** | 1 (8) | 1 (9) | 0 (B) | 0 (A) |

Essential Prime Implicants: B’C’, A’

Expression: A’ + B’C’

Converting Expression:

* A’ + B’C’ (Initial Expression)
* A’ + (B + C)’ (De Morgan’s Law)
* (A(B+C))’ (De Morgan’s Law)
* (AB + AC)’ (Distributive)
* (AB)’(AC)’ (De Morgan’s Law)
* Different Notation: (A NAND B) AND (A NAND C)
* AND can be rewritten as two NANDs

