2016

Computer Architecture Lab

LAB 4 : CPU STRUCTURE, PIPELINE PROGRAMMING AND HAZARDS, EXCEPTIONS AND INTERRUPTS

JEFFREY HUANG | RUID : 159-00-4687

Assignment 1

```
# Jeffrey Huang
# RUID: 159-00-4687
# NETID: jh1127
# Assignment 1
                                       #   RegDst  ALUSrc  MemtoReg    RegWrite    MemRead     Branch  ALUOp[1:0]  jump
    addi $t0, $t0, 128                 #     0       1        0            1           1          0      00          00
    sw $t0, 32($s0)                    #     x       1        x            0           0          0      00          00
    bneqz $t1, EXIT                    #     0       0        0            0           0          1      01          00
    xor $s0, $t1, $t2                  #     1       1        1            1           0          0      10          00
    jal print                          #     x       1        x            0           1          x      xx          10
```

Assignment 2

```
# Jeffrey Huang
# RUID: 159-00-4687
# NETID: jh1127
# Assignment 2
-------------------------------------------------------------------------------------------------------------------------------
Part 1.A:               #   1      2      3      4      5      6      7      8      9      10     11
    add $t0, $t1, $t2   #   IF     ID     EX     MEM    WB
    lw $a0, B($s0)      #          IF     ID     EX     MEM    WB
    add $a1, $a0, $t0   #                 IF     STALL  STALL  ID     EX     MEM    WB
-------------------------------------------------------------------------------------------------------------------------------
Part 1.B:               #   1      2      3      4      5      6      7      8      9      10     11     12     13     14
    and $t0, $t2, $t4   #   IF     ID     EX     MEM    WB
    add $t6, $t0, $t7   #          IF     STALL  STALL  ID     EX     MEM    WB
    sw $t6, 0($a0)      #                 IF     STALL  STALL  STALL  STALL  ID     EX     MEM    WB
    sub $t6, $t1, $t3   #                        IF     STALL  STALL  STALL  STALL  STALL  STALL  ID     EX     MEM    WB
-------------------------------------------------------------------------------------------------------------------------------
Part 2.A:               #   1      2      3      4      5      6      7      8      9      10     11     12
    add $t0, $t1, $t2   #   IF     ID     EX     MEM    WB
    lw $a0, B($s0)      #          IF     ID     EX     MEM    WB
    add $a1, $a0, $t0   #                 IF     STALL  ID     EX     MEM    WB
-------------------------------------------------------------------------------------------------------------------------------
Part 2.B:               #   1      2      3      4      5      6      7      8      9      10     11
    and $t0, $t2, $t4   #   IF     ID     EX     MEM    WB
    add $t6, $t0, $t7   #          IF     STALL  ID     EX     MEM    WB
    sw $t6, 0($a0)      #                 IF     STALL  STALL  ID     EX     MEM    WB
    sub $t6, $t1, $t3   #                        IF     STALL  STALL  STALL  ID     EX     MEM    WB
-------------------------------------------------------------------------------------------------------------------------------
```

Assignment 3

```
# Jeffrey Huang
# RUID: 159-00-4687
# NETID: jh1127
# Assignment 3

# quad_sol.s
# This assembly program calculates the integer solutions of a quadratic polynomial.
# Inputs : The coefficients a,b,c of the equation a*x^2 + b*x + c = 0
# Output : The two integer solutions.
#
# All numbers are 32 bit integers


    .globl main
main:   # Read all inputs and put them in floating point registers.
        li      $v0, 4    # Load print_string syscall code to register v0 for the 1st string.
        la      $a0, str1 # Load actual string to register $a0
        syscall           # Make the syscall
        li      $v0, 5    # Load read_int syscall code to register v0 for the coefficient a of a quadratic polynomial
        syscall           # Make the syscall
        move    $t1, $v0  # Move input from register $v0 to register $t1
        li      $v0, 4    # Load print_string syscall code to register v0 for the 2nd string.
        la      $a0, str2 # Load actual string to register $a0
        syscall           # Make the syscall
        li      $v0, 5    # Load read_int syscall code to register v0 for the coefficient a of a quadratic polynomial
        syscall           # Make the syscall
        move    $t2, $v0  # Move input from register $v0 to register $t2
        li      $v0, 4    # Load print_string syscall code to register v0 for the 3rd string.
        la      $a0, str3 # Load actual string to register $a0
        syscall           # Make the syscall
        li      $v0, 5    # Load read_int syscall code to register v0 for the coefficient a of a quadratic polynomial
        syscall           # Make the syscall
        move    $t3, $v0  # Move input from register $v0 to register $t3
        # ! -- NOTHING ABOVE CAN BE CHANGED
        # In the following lines all the necessary steps are taken to
        # calculate the discriminant of the quadratic equation.
        # As is known D = b^2 - 4*a*c



        mul     $t5,$t1,$t3  # 36: t5 = t1*t3, where t1 holds a and t3 holds c
        li      $t0, 2       # 34: Load constant number to integer register
        mul     $t4,$t2,$t2  # 35: t4 = t2*t2, where t2 holds b
        li      $s0, 0       # 46: Square Root Partial Result, sqrt(D).
        li      $t7, 1       # 47: Decrement step.
        mul     $t5,$t5,4    # 37: Multiply value of s0 with 4, creating 4*a*c
        sub     $t6,$t4,$t5  # 38: Calculate D = b^2-4*a*c
        tlt     $t6,$0       # 39: If D is less than 0 issue an exception

        # The following lines calculate the Integer result of the square root
        # of  a positive integer number D with a recursive algorithm.
        # x[n+1] = x[n] -(1+2*n), where n is the integer square root of an integer
        # number. x[0], of the step before the loop is D. The algorithm stops
        # when x[n+1] is less than zero.


        move    $s1,$t6      # 48: Move value in register t6 to register s1 for safety purposes.
    sqrtloop:
        sub  $s1,$s1,$t7      # 50: Subtract the decrement step from the x[n]
        addi    $t7,$t7,2     # 53: Increase by 2 the decrement step
        bltz $s1,endsqrt      # 51: Check if x[n+1] is less than zero, if yes stop
        addi    $s0,$s0,1     # 52: Increase partial result
        b sqrtloop            # 54: Branch unconditionally to sqrtloop label

    endsqrt:
        mul     $s5,$t1,$t0  # 60: Calculate 2*a and save it to s5
        neg     $s2,$t2      # 57: Calculate -b and save it to s2
        add     $s3,$s2,$s0  # 58: Calculate -b+sqrt(D) and save it to s3
        sub     $s4,$s2,$s0  # 59: Calculate -b-sqrt(D) and save it to s4
        div     $s6,$s3,$s5  # 61: Calculate first integer solution
        div     $s7,$s4,$s5  # 62: Calculate second integer solution
        # ! -- NOTHING BELOW CAN BE CHANGED.
        #Print the calculated solutions.
        li  $v0,4           # Load print_string syscall code to register v0 for the 1st result string.
        la  $a0, str4       # Load actual string to register $a0
        syscall             # Make the syscall
```

```
        li  $v0, 1          # Load print_int syscall code to register v0 for the 1st result string.
        move    $a0, $s6    # Load actual integer to register $a0
        syscall             # Make the syscall

        li  $v0,4           # Load print_string syscall code to register v0 for the 1st result string.
        la  $a0, str5       # Load actual string to register $a0
        syscall             # Make the syscall
        li  $v0, 1          # Load print_float syscall code to register v0 for the 1st result string.
        move    $a0, $s7    # Load actual float to register $f12
        syscall             # Make the syscall
        li  $v0, 10         # Load exit syscall code to register v0.
        syscall             # Make the syscall


.data
str1 : .asciiz "Please enter coefficient a of equation a*x^2 + b*x + c: "
str2 : .asciiz "Please enter coefficient b of equation a*x^2 + b*x + c: "
str3 : .asciiz "Please enter coefficient c of equation a*x^2 + b*x + c: "
str4 : .asciiz "The first integer solution is: "
str5 : .asciiz "\nThe second integer solution is: "
```

Assignment 3 – Problem 1

The code above is the rewritten code to reduce the cycles needed for execution. The percentage of reduction above in terms of number of stalls is by 33% because the original number of stalls is 24 but with the newly increased number of stalls, the number of stalls when executed is 16. Hence, 8 stalls were removed and then that means that the decrement is by 33%.

Assignment 3 – Problem 2

The execution of the program would change if instruction forwarding was supported. This means that instead of having to wait for the value to be loaded into a register at the write step, the register will be reloaded prior to the execution of the write step so the next step is able to execute their code with the proper output. This means that fewer cycles would be required and less re-organizing of the code is required as shown above.

Assignment 4

The code used for this assignment was provided as a file io.s. This program puts the user input as a buffer, reads the buffer until a new line is found and then prints the character out one by one as an input and enters a new line when there is a “\n” found. “\n” in Ascii value is 10 so that means that when the input character is equal to 10, the counter goes down until there are only 6 characters inputted. The exception response code is when are receiver and transmitter interrupts. If the mapped IO is not enabled in the QtSpim, the program will not be able to map the inputs, as a result not process anything until mapped IO is turned on so that the program may create a buffer, and read each character one by one.

Assignment 5

```
# Jeffrey Huang
# RUID: 159-00-4687
# NETID: jh1127
# Assignment 6

.data
    ask_index:      .asciiz     "Please enter the value of n that you would like to get use: \n"
    tell_result:    .asciiz     "Result of a[n] = "
## Arithemtic Overflow Interrupt Message
.kdata
    error:      .asciiz     "Arithemtic Overflow.\n"
.text

main:
    la $a0, ask_index       # Load and print string asking for string
    li $v0, 4               # load syscall print_string into $v0
    syscall                 # make the syscall.
    li $v0, 5               # load syscall read_into into $v0
    syscall                 # make the syscall.
    move $t0, $v0           # move the number read into $t0
    addi $t0, $t0, 1        # making number input the nuber of inputs.
    li $t1, 3               # initializing i = 2
    li $t2, 0               # a[0] = 0
    li $t3, 1               # a[1] = 1
    li $t4, 1               # a[2] = 1
    j loop

loop:
    bge $t1, $t0, END       # for loop from 3 to n
    addi $t1, $t1, 1        # incremement the counter
    add $t5, $t2, $t3       # adding a[n-3] and a[n-2]
    add $t5, $t5, $t4       # adding $t5 to a[n-1]
    move $t2, $t3           # moving the $t3 into $t2
    move $t3, $t4           # moving the $t4 into $t3
    move $t4, $t5           # moving the $t5 into $t4
    j loop                  # jump to loop

END:
    la $a0, tell_result     # print string with tell header
    li $v0, 4               # load syscall print_string into $v0
    syscall                 # make the syscall.
    move $a0, $t4           # move the number to print into $a0.
    li $v0, 1               # load syscall print_int into $v0.
    syscall                 # make the syscall
    li $v0, 10              # syscall code 10 is for exit.
    syscall                 # make the syscall.

## Arithmetic Overflow Interrupt
.ktext 0x80000180
    move $k0, $v0           # moving to interrupt register
    move $k1, $a0           # moving to interrupt register
    la $a0, error           # loading syscall argument for print_string
    li $v0, 4               # loading syscall service to print_string
    syscall                 # making syscall to print_string
    move $v0, $k0           # retrieving interrupt register
    move $a0, $k1           # retrieving interrupt register
    mfc0 $k0, $14
    lui $k0, 0x0040         # loading upper to base address
    ori $k0, 0x0000         # loading lower to base address
    mtc0 $k0, $14
    eret
```

Output for Assignment 5

Console

```
Please enter the value of n that you would like to get use:
1000
Arithemtic Overflow.
Please enter the value of n that you would like to get use:
13
Result of a[n] = 927
```

Console

```
Please enter the value of n that you would like to get use:
40
Arithemtic Overflow.
Please enter the value of n that you would like to get use:
39
Arithemtic Overflow.
Please enter the value of n that you would like to get use:
38
Arithemtic Overflow.
Please enter the value of n that you would like to get use:
37
Result of a[n] = 2082876103
```

NOTE: The arithmetic overflow only occurs when the value is greater than 37