Báo cáo thực hành môn Kiến trúc máy tính

Họ tên: Nguyễn Thanh Hưng

MSSV: 20225633

1. Assignment 1

Code:

.data

A:  .word   -2, 6, -1, 3, -2

.text

main:

    # Initialize data for the integer list

    la      $a0,    A

    li      $a1,    5

    # Call the mspfx procedure

    jal     mspfx

    nop

    # Print the results

    li      $v0,    1               # Print integer

    move    $a0,    $v0             # Move the result to $a0

    syscall

    # Exit the program

    li      $v0,    10              # Exit program

    syscall

mspfx:

    # Procedure to explore each element of the integer list using indexing method

    addi    $v0,    $zero,  0       # Initialize length in $v0 to 0

    addi    $v1,    $zero,  0       # Initialize max sum in $v1 to 0

    addi    $t0,    $zero,  0       # Initialize index i in $t0 to 0

    addi    $t1,    $zero,  0       # Initialize running sum in $t1 to 0

loop:

    sll     $t2,    $t0,    2       # Multiply i by 4 to get 4i

    add     $t3,    $t2,    $a0     # Put 4i+A (address of A[i]) in $t3

    lw      $t4,    0($t3)          # Load A[i] from mem(t3) into $t4

    add     $t1,    $t1,    $t4     # Add A[i] to running sum in $t1

    slt     $t5,    $v1,    $t1     # Set $t5 to 1 if max sum < new sum

    bne     $t5,    $zero,  mdfy    # If max sum is less, modify results

    j       test                    # Done?

mdfy:

    addi    $v0,    $t0,    1       # New max-sum prefix has length i+1

    addi    $v1,    $t1,    0       # New max sum is the running sum

test:

    addi    $t0,    $t0,    1       # Advance the index i

    slt     $t5,    $t0,    $a1     # Set $t5 to 1 if i < n

    bne     $t5,    $zero,  loop    # Repeat if i < n

    j       done

    nop

done:

    jr      $ra                     # Return to the main program

Giải thích code:

1. Data

A: .word -2, 6, -1, 3, -2 - Khai báo một mảng tên A gồm 5 phần tử kiểu word (32 bit) được khởi tạo với các giá trị -2, 6, -1, 3, -2.

1. Text

* Main:
  + `la $a0, A` - Câu lệnh `la` (Load Address) tải địa chỉ của biến `A` vào thanh ghi `$a0`.
  + `li $a1, 5` - Câu lệnh `li` (Load Immediate) tải giá trị 5 (số phần tử trong mảng) vào thanh ghi `$a1`.
  + `jal mspfx` - Câu lệnh `jal` (Jump and Link) gọi đến procedure `mspfx` và lưu địa chỉ để quay lại `main` vào thanh ghi `$ra`.
  + `nop` - Câu lệnh `nop` (No Operation) không thực hiện tác vụ gì, thường dùng để placeholder.
  + Các lệnh in và thoát chương trình thông thường (`li`, `move`, `syscall`).
* Mspfx:
  + Khởi tạo các biến đếm (`$v0`), Max sum (`$v1`), index (`$t0`), running sum (`$t1`) bằng 0.
* Loop:
  + `sll $t2, $t0, 2` - Dịch trái giá trị `$t0` (index `i`) 2 bit để nhân `i` với 4 (do 1 word = 4 bytes).
  + `add $t3, $t2, $a0` - Cộng địa chỉ gốc `$a0` (của mảng `A`) với `$t2` để tính toán địa chỉ của phần tử thứ `i` trong mảng `A`.
  + `lw $t4, 0($t3)` - Load giá trị của phần tử thứ `i` từ địa chỉ `$t3` vào thanh ghi `$t4`.
  + `add $t1, $t1, $t4` - Cộng giá trị vừa đọc (`$t4`) vào running sum (`$t1`).
  + `slt $t5, $v1, $t1` - So sánh Max sum (`$v1`) với running sum (`$t1`). Nếu Max sum nhỏ hơn, đặt `$t5` bằng 1.
  + `bne $t5, $zero, mdfy` - Nếu `$t5` khác 0 (Max sum nhỏ hơn), nhảy đến khối `mdfy`.
  + `j test` - Nhảy đến khối `test` để kiểm tra điều kiện kết thúc vòng lặp.
* Mdfy:
  + `addi $v0, $t0, 1` - Cập nhật Max-sum prefix length (`$v0`) bằng `i + 1` (index `i` cộng 1).
  + `addi $v1, $t1, 0` - Cập nhật Max sum (`$v1`) bằng running sum (`$t1`).
* Test:
  + `addi $t0, $t0, 1` - Tăng index `i` (`$t0`) lên 1.
  + `slt $t5, $t0, $a1` - So sánh index `i` (`$t0`) với số phần tử (`$a1`). Nếu `i` nhỏ hơn `n`, đặt `$t5` bằng 1.
  + `bne $t5, $zero, loop` - Nếu `i` nhỏ hơn `n`, nhảy lại vòng lặp `loop`.
  + `j done` - Nhảy đến khối `done` để kết thúc procedure.

1. Done

* jr $ra - Câu lệnh jr (Jump Register) sử dụng giá trị trong thanh ghi $ra (địa chỉ quay lại) để trở về main.

1. Kết quả của chương trình sau khi thực hiện phép tính Max Sum của 1 đoạn con trong 1 Array:

(-2) + 6 + (-1) + 3 = 6



1. Assignment 2

Code:

.data

A:      .word   7, -2, 5, 1, 5, 6, 7, 3, 6, 8, 8, 59, 5

Aend:   .word

.text

main:

    # Initialize data

    la      $a0,        A                       #$a0 = Address(A[0])

    la      $a1,        Aend

    addi    $a1,        $a1,    -4                      #$a1 = Address(A[n-1])

    # Call the sort procedure

    j       sort

after\_sort:

    # Exit the program

    li      $v0,        10                   #exit

    syscall

end\_main:

sort:

    # Check if the list has only one element

    beq     $a0,        $a1,    done            #single element list is sorted

    # Call the max procedure

    j       max

after\_max:

    # Swap the last element with the maximum value

    lw      $t0,        0($a1)                #load last element into $t0

    sw      $t0,        0($v0)                          #copy last element to max location

    sw      $v1,        0($a1)                          #copy max value to last element

    addi    $a1,        $a1,    -4                      #decrement pointer to last element

    # Repeat the sort for the smaller list

    j       sort

done:

    # Sort is complete, jump to after\_sort

    j       after\_sort

max:

    # Initialize max pointer and max value

    addi    $v0,        $a0,    0                       #init max pointer to first element

    lw      $v1,        0($v0)                          #init max value to first value

    addi    $t0,        $a0,    0                       #init next pointer to first

loop:

    # Check if next=last, if yes, return

    beq     $t0,        $a1,    ret                     #if next=last, return

    # Advance to the next element

    addi    $t0,        $t0,    4                       #advance to next element

    lw      $t1,        0($t0)                          #load next element into $t1

    # Check if next < max

    slt     $t2,        $t1,    $v1                     #(next)<(max) ?

    bne     $t2,        $zero,  loop                    #if (next)<(max), repeat

    # Update max pointer and max value

    addi    $v0,        $t0,    0                       #next element is new max element

    addi    $v1,        $t1,    0                       #next value is new max value

    # Repeat the loop

    j       loop

ret:

    # Return to after\_max

    j       after\_max

Giải thích code:

1. Data:

* A: .word 7, -2, 5, 1, 5, 6, 7, 3, 6, 8, 8, 59, 5 - Khai báo mảng A gồm 13 số nguyên khởi tạo với các giá trị tương ứng.
* Aend: .word - Nhãn đánh dấu cuối mảng A (không chứa giá trị thực).

1. Text:

* Main:
  + `la $a0, A` - Load địa chỉ phần tử đầu tiên của mảng `A` vào `$a0`.
  + `la $a1, Aend` - Load địa chỉ nhãn `Aend` (cuối mảng) vào `$a1`.
  + `addi $a1, $a1, -4` - Điều chỉnh `$a1` để trỏ đến phần tử cuối cùng của mảng `A` (do `Aend` không chứa giá trị thực).
  + `j sort` - Nhảy đến procedure `sort` để sắp xếp mảng.
* After sort:
  + `li $v0, 10` - Chuẩn bị giá trị 10 cho lệnh thoát (`syscall`).
  + `syscall` - Thực hiện lệnh thoát chương trình.
* End main: Nhãn đánh dấu kết thúc procedure `main`.
* Sort:
  + `beq $a0, $a1, done` - Kiểm tra xem mảng chỉ có 1 phần tử (`$a0` = `$a1`). Nếu đúng (danh sách đơn), nhảy đến `done`.
  + `j max` - Nhảy đến procedure `max` để tìm phần tử lớn nhất.
* After max:
  + `lw $t0, 0($a1)` - Load giá trị phần tử cuối (`$a1`) vào `$t0`.
  + `sw $t0, 0($v0)` - Lưu giá trị phần tử cuối vào vị trí được trỏ bởi `$v0` (giả định là vị trí lưu giá trị max).
  + `sw $v1, 0($a1)` - Lưu giá trị max (`$v1`) vào vị trí phần tử cuối (`$a1`).
  + `addi $a1, $a1, -4` - Giảm `$a1` 4 để trỏ đến phần tử cuối mới (sau khi hoán đổi).
  + `j sort` - Nhảy lại `sort` để sắp xếp mảng nhỏ hơn (không tính phần tử lớn nhất vừa tách ra).
* Done:
  + Nhãn đánh dấu kết thúc procedure `sort`.
  + `j after\_sort` - Nhảy đến `after\_sort` để thoát chương trình.
* Max:
  + `addi $v0, $a0, 0` - Khởi tạo `$v0` (con trỏ max) để trỏ đến phần tử đầu tiên.
  + `lw $v1, 0($v0)` - Load giá trị phần tử đầu tiên vào `$v1` (max hiện tại).
  + `addi $t0, $a0, 0` - Khởi tạo `$t0` (con trỏ duyệt) để trỏ đến phần tử đầu tiên.
* Loop:
  + `beq $t0, $a1, ret` - Kiểm tra xem `$t0` đã trỏ đến phần tử cuối (`$a1`). Nếu đúng, nhảy đến `ret`.
  + `addi $t0, $t0, 4` - Di chuyển `$t0` 4 đơn vị để trỏ đến phần tử kế tiếp.
  + `lw $t1, 0($t0)` - Load giá trị phần tử kế tiếp vào `$t1`.
  + `slt $t2, $t1, $v1` - So sánh giá trị phần tử kế tiếp (`$t1`) với max hiện tại (`$v1`).
  + `bne $t2, $zero, loop`
  + `addi $v0, $t0, 0` - Cập nhật `$v0` (con trỏ max) để trỏ đến phần tử kế tiếp (lớn hơn).
  + `addi $v1, $t1, 0` - Cập nhật `$v1` (max hiện tại) bằng giá trị phần tử kế tiếp.
* Ret:
  + Nhãn đánh dấu kết thúc procedure `max`.
  + `j after\_max` - Nhảy đến `after\_max` để tiếp tục quá trình sắp xếp.

1. Assignment 3

Code:

    # Bubble Sort Procedure

.data

array:  .word       5, 3, 8, 4, 2, 1, 9, 7, 6, 0

size:   .word       10

.text

.globl main

main:

    la      $a0,        array               # Load array address

    lw      $a1,        size                # Load array size

    jal     bubble\_sort                     # Call bubble\_sort procedure

    li      $v0,        10                  # Exit program

    syscall

bubble\_sort:

    addi    $sp,        $sp,        -12     # Allocate space for temporary variables

    sw      $ra,        0($sp)              # Save return address

    sw      $s0,        4($sp)              # Save $s0

    sw      $s1,        8($sp)              # Save $s1

    move    $s0,        $a0                 # Move array address to $s0

    move    $s1,        $a1                 # Move array size to $s1

    li      $t0,        1                   # Set flag to 1 (true)

    li      $t1,        0                   # Initialize counter to 0

loop:

    beqz    $t0,        end                 # If flag is 0, exit loop

    li      $t0,        0                   # Reset flag to 0 (false)

    li      $t1,        1                   # Reset counter to 1

inner\_loop:

    blt     $t1,        $s1,        compare # If counter < size, go to compare

    j       loop                            # Otherwise, go back to outer loop

compare:

    sll     $t2,        $t1,        2       # Multiply counter by 4 to get offset

    add     $t2,        $s0,        $t2     # Calculate address of current element

    lw      $t3,        0($t2)              # Load current element

    lw      $t4,        -4($t2)             # Load previous element

    ble     $t3,        $t4,        no\_swap # If current <= previous, go to no\_swap

    sw      $t4,        0($t2)              # Swap current and previous elements

    sw      $t3,        -4($t2)

    li      $t0,        1                   # Set flag to 1 (true)

no\_swap:

    addi    $t1,        $t1,        1       # Increment counter

    j       inner\_loop                      # Go back to inner loop

end:

    lw      $ra,        0($sp)              # Restore return address

    lw      $s0,        4($sp)              # Restore $s0

    lw      $s1,        8($sp)              # Restore $s1

    addi    $sp,        $sp,        12      # Deallocate space for temporary variables

    jr      $ra                             # Return

1. Assignment 4

Code:

    # Insertion Sort

.data

A:      .word   5, 2, 3, 1, 4

Aend:   .word

.text

    la      $a0,    A

    addi    $t0,    $0,         13                      # N

    addi    $t1,    $0,         0                       # i = 0

Loop1:

    slt     $t2,    $t1,        $t0                     #Check i < N

    bne     $t2,    1,          End\_loop1

    nop

    addi    $t3,    $t0,        -1                      # j = N - 1

Loop2:

    slt     $t4,    $t1,        $t3                     #Check j > i

    bne     $t4,    1,          end\_loop2

    nop

    sll     $t3,    $t3,        2

    add     $a1,    $a0,        $t3

    lw      $s0,    0($a1)                              # = arr[j]

    lw      $s1,    -4($a1)                             # = arr[j-1]

    slt     $t5,    $s0,        $s1

    bne     $t5,    1,          if\_loop2

    nop

    sw      $s0,    -4($a1)

    sw      $s1,    0($a1)

    srl     $t3,    $t3,        2

    addi    $t3,    $t3,        -1                      # j--

    j       Loop2

    nop

end\_loop2:

    addi    $t1,    $t1,        1                       # i++

    j       Loop1

    nop

if\_loop2:

    srl     $t3,    $t3,        2

    addi    $t3,    $t3,        -1                      # j--

    j       Loop2

    nop

End\_loop1:

    lw      $ra,        0($sp)              # Restore the return address from the stack

    addi    $sp,        $sp,        4       # Deallocate the space on the stack

    jr      $ra                             # Return from the procedure