香港中文大學
The Chinese University of Hong Kong

CENG3420

Lab 1-3: RISC-V Assembly Language Programing III

Chen BAI, Yuxuan Zhao
Department of Computer Science & Engineering
Chinese University of Hong Kong
cbai@cse.cuhk.edu.hk,
yxzhao21@cse.cuhk.edu.hk

Spring 2023

Outline

Recap

Example 1 – Array Definition I

Example

```
.data
    a: .word 1 2 3 4 5
```

- How do we access the 4th element (the integer 4)?

Example 2 – If-ElseIf-Else Statement I

```
_start:
    andi t0, t0, 0          # clear register t0
    andi t1, t1, 0          # clear register t1
    andi t2, t2, 0          # clear register t2
    andi t3, t3, 0          # clear register t3
    andi t4, t4, 0          # clear register t4
    andi t5, t5, 0          # clear register t5
    li t0, 2                # t0 = 2
    li t3, -2               # t3 = -2
    slt t1, t0, zero        # t1 = t0 < 0 ? 1 : 0
    beq t1, zero, ElseIf    # go to ElseIf if t1 = 0
    j EndIf                 # end If statement
ElseIf:
    sgt t4, t3, zero        # t4 = t3 > 0 ? 1 : 0
    beq t4, zero, Else      # go to Else if t4 = 0
    j EndIf                 # end Else statement
Else:
    seqz t5, t4, zero       # t5 = t4 == 0 ? 1 : 0
EndIf:
    j EndIf                 # end If-ElseIf-Else statement
```

Example 3 – While Loop I

```
_start:
    andi t0, t0, 0          # clear register t0
    andi t1, t1, 0          # clear register t1
    andi t2, t2, 0          # clear register t2
    li t1, 100              # t1 = 100
loop:
    add t2, t2, t0          # t2 = t2 + t0
    addi t0, t0, 1          # ++t0
    blt t0, t1, loop        # iterate if t0 < t1
end:
    j end                   # end of While loop
```

Example 4 – For Loop I

```
_start:
    andi t0, t0, 0          # clear register t0
    andi t1, t1, 0          # clear register t1
loop:
    andi t2, t2, 0          # clear t2 before starting the loop
    add t1, t1, t0          # t1 = t1 + t0
    addi t0, t0, 1          # ++t0
    slti t2, t0, 100        # t2 = t0 < 100 ? 1 : 0
    bne t2, zero, loop      # go to loop if t2 != 0
end:
    j end                   # end of For loop
```

Recursive Program in RISC-V Assembly

How to Call Nested Functions?

```
.globl _start
.text
_start: li a0, 20
        li a1, 23
        jal ra, func # we call a function: func
                     # func implements (a0 x 2 + a1)
                     # and put the result in t1
        addi t1, a2, 0 # a2 = func(a0, a1)
        j end
func:   addi sp, sp -12
        sw ra, 8(sp)
        sw a0, 4(sp)
        sw a1, 0(sp)
        slli a0, a0, 1
        jal ra, add_two_numbers # add_two_numbers implements (a0 + a1)
        lw ra, 8(sp)
        lw a0, 4(sp)
        lw a1, 0(sp)
        addi sp, sp, 12
        jalr zero, 0(ra)
add_two_numbers: addi sp, sp -8 # we assign 8x4 bytes in the stack
                                # stack: top (high address) -> bottom (low address)
        sw a0, 4(sp)   # we save arguments in the stack
        sw a1, 0(sp)
        add a2, a0, a1 # the a0 and a1 can be used directly since the
                       # original values of a0 and a1 are saved in the stack
        lw a0, 4(sp)   # we restore arguments
        lw a1, 0(sp)
        addi sp, sp, 8 # NOTICE: we need to free the stack we have allocated!
        jalr zero, 0(ra)
end:
        # we add t1 again
        addi t1, t1, 1
```

RARS example: compiling a recursive procedure

- What is the final value of t1?

How to Call Nested Functions?

```
.globl _start
.text
_start: li a0, 20
        li a1, 23
        jal ra, func # we call a function: func
                     # func implements (a0 x 2 + a1)
                     # and put the result in t1
        addi t1, a2, 0 # a2 = func(a0, a1)
        j end
func:   addi sp, sp -12
        sw ra, 8(sp)
        sw a0, 4(sp)
        sw a1, 0(sp)
        slli a0, a0, 1
        jal ra, add_two_numbers # add_two_numbers implements (a0 + a1)
        lw ra, 8(sp)
        lw a0, 4(sp)
        lw a1, 0(sp)
        addi sp, sp, 12
        jalr zero, 0(ra)
add_two_numbers: addi sp, sp -8 # we assign 8x4 bytes in the stack
                                # stack: top (high address) -> bottom (low address)
        sw a0, 4(sp)   # we save arguments in the stack
        sw a1, 0(sp)
        add a2, a0, a1 # the a0 and a1 can be used directly since the
                       # original values of a0 and a1 are saved in the stack
        lw a0, 4(sp)   # we restore arguments
        lw a1, 0(sp)
        addi sp, sp, 8 # NOTICE: we need to free the stack we have allocated!
        jalr zero, 0(ra)
end:
        # we add t1 again
        addi t1, t1, 1
```

RARS example: compiling a recursive procedure

- What is the final value of t1?
- t1 = 0x40

Compiling a Recursive Program

A procedure for calculating factorial

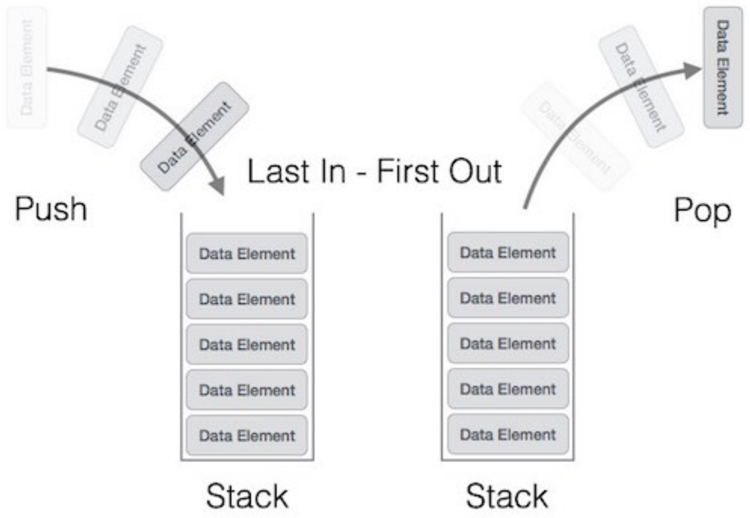
```
int fact (int n)
{
    if (n < 1) return 1;
    else return (n * fact (n-1));
}
```

- A recursive procedure (one that calls itself!)

```
fact (0) = 1
fact (1) = 1 * 1 = 1
fact (2) = 2 * 1 * 1 = 2
fact (3) = 3 * 2 * 1 * 1 = 6
fact (4) = 4 * 3 * 2 * 1 * 1 = 24
. . .
```

- Assume n is passed in `a0`; result returned in `ra`

Stack



Push
Last In - First Out
Pop
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Stack
Stack

Compiling a Recursive Program (cont.)

```
fact:
    addi  sp, sp, -8       # adjust the stack pointer
    sw    ra, 4(sp)        # save the return address
    sw    a0, 0(sp)        # save the argument n
    slti  t0, a0, 1        # test for n < 1
    beq   t0, zero, L1     # if n >=1, go to L1
    addi  t1, zero, 1      # else return 1 in t1
    addi  sp, sp, 8        # adjust stack pointer
    jr    ra               # return to caller
L1:
    addi  a0, a0, -1       # n >=1, so decrease n
    jal   fact             # call fact with (n-1)
                           # this is where fact returns
bk_f:
    lw    a0, 0(sp)        # restore argument n
    lw    ra, 4(sp)        # restore return address
    addi  sp, sp, 8        # adjust stack pointer
    mul   t1, a0, t1       # t1 = n * fact(n-1)
    jr    ra               # return to caller
```

Another Example I

Example

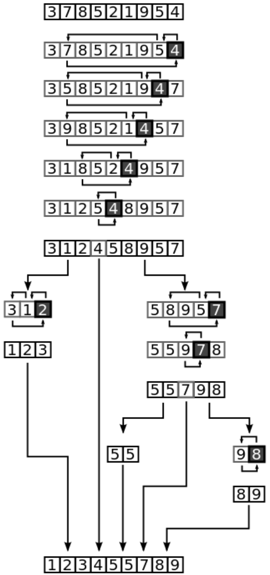
Another Example II

```
.globl _start
.text                         # recursive implementation of factorial
fact:                         # arg: n in a0, returns n! in a1
    addi  sp, sp, -8          # reserve our stack area
    sw ra, 0(sp)              # save the return address
    li t0, 2                  # t0 = 2
    blt a0, t0, ret_one       # go to ret_one if a0 < t0
    sw a0, 4(sp)              # save our n
    addi a0, a0, -1
    jal fact                  # call fact (n-1), a1 <- fact(n-1)
    lw t0, 4(sp)              # t0 <- n
    mul a1, t0, a1            # a1 <- n * fact(n-1)
    j done
ret_one:
    li a1, 1
done:
    lw ra, 0(sp)              # restore return address from stack
    addi sp, sp, 8            # free our stack frame
    jr ra                     # and return
_start:
    li a0, 5                  # compute 5!
    jal fact                  # call 'fact'
    li a0, 1                  # print it
    ecall
```

Quicksort

Overview of Quicksort

Quicksort is a divide and conquer algorithm. Quicksort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quicksort can then recursively sort the sub-arrays.

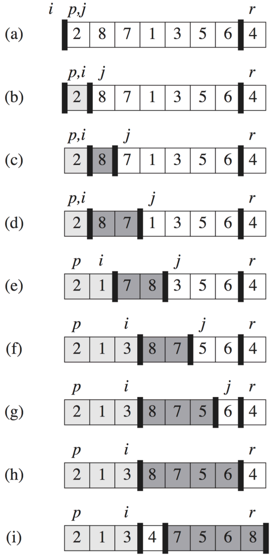


Quicksort: Array Partitioning (Lab 1-2)

- Pick an element, called a pivot, from the array.
- Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way).

```
function PARTITION(A, lo, hi)
    pivot ← A[hi]
    i ← lo-1;
    for j = lo; j ≤ hi-1; j ← j+1 do
        if A[j] ≤ pivot then
            i ← i+1;
            swap A[i] with A[j];
        end if
    end for
    swap A[i+1] with A[hi];
    return i+1;
end function
```

Example of Array Partition



[1]

[1] In this example, p = lo and r = hi.

Quicksort: Sorting

- Recursively apply the array partition to the sub-array of elements with smaller values and separately to elements with greater values.

```
function QUICKSORT(A, lo, hi)
    if lo < hi then
        p ← partition(A, lo, hi);
        quicksort(A, lo, p - 1);
        quicksort(A, p + 1, hi);
    end if
end function
```

Lab 1-3 Assignment

Lab Assignment

Implement Quicksort *w.r.t.* the following array in ascending order with RISC-V assembly programming:

Sort the array for this assignment

```
-1 22 8 35 5 4 11 2 1 78
```

Submission Method:

Submit one zip file into **Blackboard**, including

- Three source codes, i.e., Lab 1-1, Lab 1-2, and Lab 1-3.
  - Pay attention to the submission name format, i.e., `name-sid-lab1-x.asm`. For example, `zhangsan-1234567890-lab1-1.asm`, `zhangsan-1234567890-lab1-2.asm`, and `zhangsan-1234567890-lab1-3.asm`.
- One report. (name format: `name-sid-report.pdf`) The report summarizes your implementations and all screenshots of lab results. A report template is uploaded to Piazza.
- Deadline: 23:59, 22 Feb (Wed), 2023