COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE HW 2B PART 2 - Adil Hydari

March 20, 2024

Problem 2.

Write a RISCV leaf procedure which accepts as parameters a character (in register x12) and the base address of an array A (in register x13). The array contains 100 characters. The procedure scans array A and return to register x14 the multitude of appearances of the character which was received as argument via register x12.

Solution

```
# x12: Target character
# x13: Base address of array A
# x14: Count of character

character_scan:
addi    sp, sp, -4        # Reserve space on the stack for the counter
sw      x0, 0(sp)         # Initialize counter to 0

addi    x5, x0, 100       # Initialize loop counter
add     x6, x0, x13       # x6 now is * to current element in the array

loop:
```

```
lb      x7, 0(x6)          # Load the current byte from the array into x7
beq     x7, x12, found     # if x7 matches x12, jump to found
addi    x6, x6, 1          # Move to the next character in the array
addi    x5, x5, -1         # Decrement loop counter
bnez    x5, loop           # If loop counter is not zero, continue looping
j       end                # Jump to end

found:
lw      x8, 0(sp)          # Load the current count
addi    x8, x8, 1          # Increment the count
sw      x8, 0(sp)          # Store the updated count back to the stack
addi    x6, x6, 1          # Move to the next character in the array
addi    x5, x5, -1         # Decrement loop counter
bnez    x5, loop           # Continue looping if there are more characters

end:
lw      x14, 0(sp)         # Load the final count into x14
addi    sp, sp, 4          # Clean up the stack
ret                        # Return from the procedure
```

Problem 5

Part 1: (5 pts) Implement the following C code in RISC-V assembly.
Part 2: (3 pts) What is the total number of RISC-V instructions needed to execute the function?

```
int fib(int n) {
        if (n == 0)
                return 0;
        else if (n == 1)
                return 1;
        else
                return fib(n - 1) + fib(n - 2);
}
```

Solution

Part 1

```
fib:
addi    sp, sp, -16     # Adjust stack * for 4 words
sw      ra, 12(sp)      # Save return address
sw      a0, 8(sp)       # Save the argument n

# case 1: if (n == 0) return 0
li      t0, 0           # Load immediate 0 into t0
beq     a0, t0, end_zero

# case 2: if (n == 1) return 1
li      t1, 1           # Load immediate 1 into t1
beq     a0, t1, end_one

# Recursive case: return fib(n-1) + fib(n-2)
addi    a0, a0, -1      # Calculate n-1
call    fib             # Recursive call fib(n-1)
sw      a0, 4(sp)       # Save the result of fib(n-1)

lw      a0, 8(sp)       # Restore n
addi    a0, a0, -2      # Calculate n-2
call    fib             # Recursive call fib(n-2)

lw      t2, 4(sp)       # Load the result of fib(n-1)
add     a0, a0, t2      # Add fib(n-1) + fib(n-2)

j       end_recursion

end_zero:
li      a0, 0           # Return 0 for base case n == 0
j       cleanup

end_one:
li      a0, 1           # Return 1 for base case n == 1
j       cleanup
```

```
end_recursion:

cleanup:
lw      ra, 12(sp)      # Restore return address
addi    sp, sp, 16      # Adjust stack back
ret                      # Return to caller
```

Part 2

- Instructions for setting up and cleaning up the stack, saving and restoring registers: 8 instructions.
- Instructions for checking base cases and jumping to the end labels: 6 instructions.
- Instructions for the recursive calls, including calculating n-1 and n-2, saving intermediate results, and adding results: 10 instructions.

The path that would give the least amount of instructions would be the one in which, in main, n is defined as 0 and returns true back to the main function.

Problem 6

Part 1: (9 marks) Compile the RISC-V assembly code for the following C code. Assume that k and m are either non- negative integers or unsigned integers, passed in x8 and x9 respectively. Assume that result returned in x8. This function does not have to make sense, it is a test on your knowledge of writing nested/recursive routines. Compile the assembly code for the following C code. *long long int int likely is just long long int*

```
int func (long long int int m, long long int int k ) {
        if (k <= 0)
                return m;
        else if (m >=14)
                return k;
        else return 5*m*k + 4*func(m+4,k-1) + 6*func(m+1,k-4);
}
```

Part 2: (3 marks) How many RISC-V instructions does it take to implement the C code from Part 1? If the variables m and k are initialized to 6 and 9 what is the total number of RISC-V instructions that is executed to complete the loop?

Solution

Part 1

```
.section .text
.global func
# Arguments:
#   m - x8 (a0)
#   k - x9 (a1)
# Return:
#   Result in x8 (a0)
func:
        addi sp, sp, -20 # allocation of space for 8 words
        sw ra, 16(sp) #save return address
        sw a0, 12(sp) # m = a0
        sw a1, 8(sp) # k = a1

        #case 1
        bge x0, a1, return_m # k <= 0

        #case 2
        li t0, 14 # temp = 14
        bge a0, t0, return_k # m >= 14

        #case 3 - recursive
        # 4*func(m+4,k-1)
        lw a0, 12(sp) # restore m to base value
        lw a1, 8(sp) # restore k to base value
        addi a0, a0, 4 # m+4
        addi a1, a1, -1 # k-1
        call func # recursive call
        addi t0, a0, 0 # store result in temp
        mul t0, t0, 4 # *4
```

```
# 6*func(m+1,k-4)
lw      a0, 12(sp)      # restore m
lw      a1, 8(sp)       # restore k
addi    a0, a0, 1       # m + 1
addi    a1, a1, -4      # k - 4
call    func            # recursive call
addi t1, a0, 0 # store result in temp
mul t1, t1, 6 # *6

# 5*m*k
lw a0, 12(sp)
lw a1, 8(sp)
mul a0, a0, 5
mul a0, a0, a1
addi t2, a0, 0

add t0, t0, t1 # 4*func(m+4,k-1) + 6*func(m+1,k-4)
add t0, t0, t2 # 5*m*k + 4*func(m+4,k-1) + 6*func(m+1,k-4)

j       func_end

return_m:
lw      a0, 12(sp)      # Load m into a0
j       func_end

return_k:
lw      a0, 8(sp)       # Load k into a0
j               func_end

func_end:
lw      ra, 16(sp)      # Restore return address
addi    sp, sp, 20      # Deallocate stack space
ret                     # Return to caller
```

Part 2

If m =6 and k=9, this means that the function ”func” should be called 13 times, including the first time it is called within the main() function. This means that we will have 34 instructions called 12 times in total for the first 12 iterations of the recursive call. The value 34 comes from the fact that we have 32 instructions in the function func() as well as 2 in both returnk() and returnm() and either or will be called and will jump to funcend(). On the last call of func(), we will not call return m or k, instead we will simply just jump to func end, as the conditionals will both be false, and result in the final return in which k ¡= 0 and m ¿= 14; this means we have 32 instructions on the 13th iteration. 34*12 + 32 = 440 instructions.