Operating System and Computer Architecture

RISC-V Assembly Programming

Sami Taktak

2025

Learning Objectives

— Manipulate registers and understand function calling conventions

— Implement control structures (loops, conditions) in assembly language

— Use memory operations (load/store) to manipulate arrays

— Design and translate simple algorithms into RISC-V assembly language

— Debug and trace the execution of assembly programs

Work Required

— Exercises require the use of the online simulator:
`https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/`

— Test and validate each program on the simulator

— Keep screenshots of the results obtained

— Total indicative duration: 2h00

Use the online RISC-V simulator at: `https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/`

* Exercise 1 Basic Arithmetic

Objective: Calculate: result = (a + b) × (c - d) + e
where a=10, b=5, c=8, d=3, e=7
Complete the following code and verify that the final result in `x10` equals 82.

```
main:
    addi x5, x0, 10       # a = 10
    addi x6, x0, 5        # b = 5
    addi x7, x0, 8        # c = 8
    addi x8, x0, 3        # d = 3
    addi x9, x0, 7        # e = 7

    # Your code here
    # Store final result in x10
```

Expected Result: x10 = 82

** Exercise 2 Loop Implementation

Objective: Sum first n numbers: 1 + 2 + 3 + ... + 10
Complete the following code:

```
# Sum first n numbers: 1 + 2 + 3 + ... + n
# Input: n = 10
# Output: sum in x10

main:
    addi x11, x0, 10      # n = 10
    addi x10, x0, 0       # sum = 0
    addi x12, x0, 1       # counter = 1

loop:
    # Your code here

    # Should output: 55
```

Expected Result: x10 = 55

** Exercise 3 Memory Operations

Objective: Store values [5, 10, 15, 20, 25] in memory and calculate their sum.
Complete the following code:

```
# Store values [5, 10, 15, 20, 25] in memory
# Then calculate their sum

.data
array: .word 0, 0, 0, 0, 0

.text
main:
    # Load base address
    lui x10, 0x10000      # Base address = 0x10000000

    # Store values (complete this part)

    # Sum all values (complete this part)
```

Expected Result: x10 = 75

*** Exercise 4 Function Implementation

Objective: Implement a factorial function using proper calling conventions.
Calculate factorial(5) = 120

```
# Calculate factorial(5)

main:
    addi x10, x0, 5       # n = 5
    jal x1, factorial     # call factorial
    # Result should be in x10 = 120

factorial:
    # Your implementation here
    # Remember to:
    # 1. Save return address if needed
    # 2. Handle base case (n=0 or n=1)
    # 3. Implement iterative calculation
    # 4. Return result in x10
```

Expected Result: x10 = 120

*** Exercise 5 Bonus Challenge: Fibonacci Sequence

Optional - If time permits
Objective: Calculate the nth Fibonacci number iteratively.
Implement a function that calculates the nth Fibonacci number:

- Input: n in `x10`
- Output: fib(n) in `x10`
- Test with n=10 (should output 55)

Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Testing Notes and Tips

Using the Cornell RISC-V Interpreter

1. Write your code into the editor
2. Click **"Assemble"** to check for syntax errors
3. Click **"Run"** to execute the program
4. Check the register values in the register panel on the right
5. For memory operations, check the memory viewer to see stored values
6. Use **"Step"** button for debugging line by line

Important Tips

- The interpreter doesn't support all pseudo-instructions - use basic instructions only
- Watch out for infinite loops - set a reasonable instruction limit
- The simulator initializes all registers to 0 by default
- For Exercise 2.3, you may need to adjust the base address depending on the simulator

Common Errors to Avoid

- Forgetting to multiply index by 4 for word addressing
- Using wrong branch instruction (signed vs unsigned)
- Infinite loops due to incorrect exit conditions
- Not initializing registers before use

Debugging Checklist

If a program doesn't work, check:

- Branch conditions (`beq`, `bne`, `blt`, `bge`, etc.)
- Memory addresses (must be word-aligned for `lw`/`sw`)
- Loop counters and exit conditions
- Register initialization