CS061:
Machine Organization & Assembly Language
Lab 7

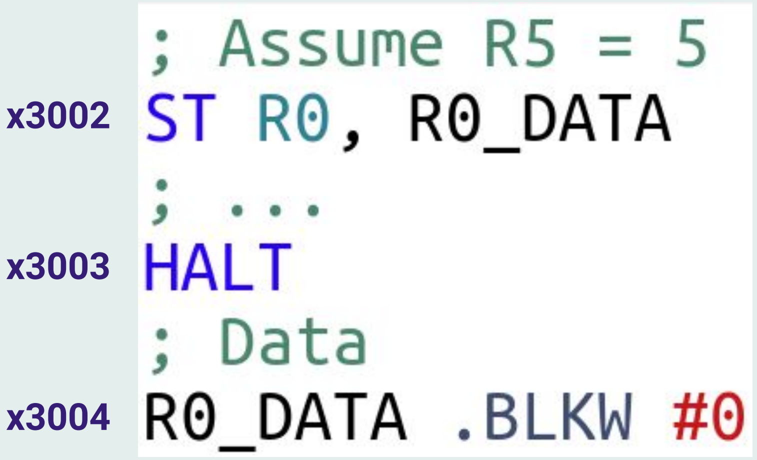
Agenda

1. Presentation:
    a. Store Direct Review
    b. Fixed Register Backups
    c. Lab Descriptions

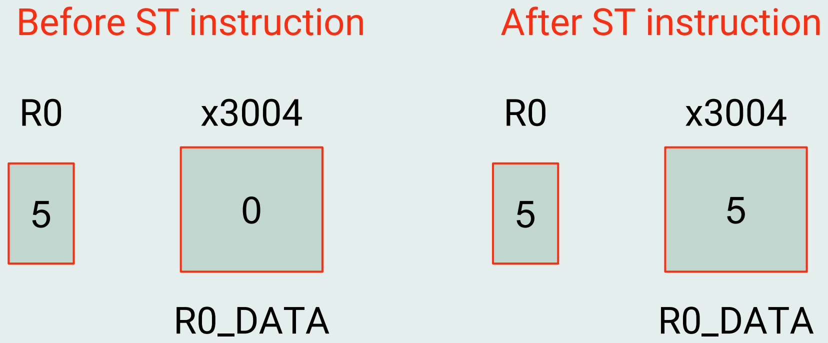
2. Work Time / Questions / Demos

Store Direct Review

- The **ST** instruction stores a value from a register to a memory address (aliased by a label).
- Direction: Register -> Memory Address



```
      ; Assume R5 = 5
x3002 ST R0, R0_DATA
      ; ...
x3003 HALT
      ; Data
x3004 R0_DATA .BLKW #0
```



Fixed Backup

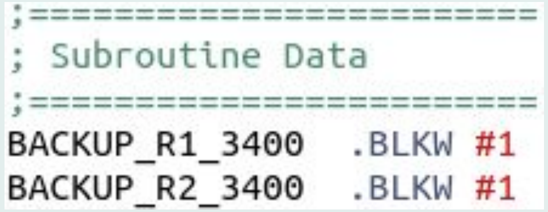
- A (*bad*) alternative to backing up registers.
- Backup registers to a fixed location in memory (to a label).

Steps

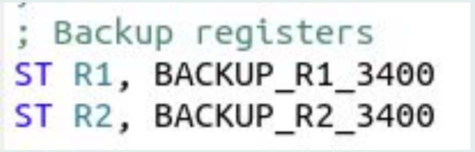
1. Create labels in the sub-routine data to store the register values.

2. At the start of the sub-routine, use “ST” to store register values to the label locations.

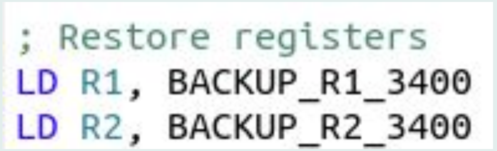
3. At the end of the sub-routine, use “LD” to load values from the label locations to the registers.



```
;==========================
; Subroutine Data
;==========================
BACKUP_R1_3400   .BLKW #1
BACKUP_R2_3400   .BLKW #1
```



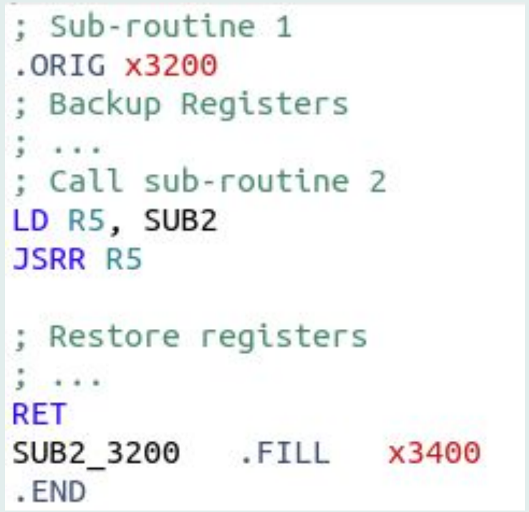
```
; Backup registers
ST R1, BACKUP_R1_3400
ST R2, BACKUP_R2_3400
```



```
; Restore registers
LD R1, BACKUP_R1_3400
LD R2, BACKUP_R2_3400
```

Why Bad?

- What happens if we call a subroutine inside another subroutine (recursion)?
  - Backup registers via stack?
  - Backup registers via labels?
  - Hint: what register is modified when a sub-routine is called?



```
; Sub-routine 1
.ORIG x3200
; Backup Registers
; ...
; Call sub-routine 2
LD R5, SUB2
JSRR R5

; Restore registers
; ...
RET
SUB2_3200   .FILL   x3400
.END
```

Debugging Subroutines (Review)

- **Live Demo**

- Tips:
  - When the blue arrow in the simulator is on a JSRR/JSR line, use the Step-In button!
  - Look at the values in the registers! Keep checking if they are what you expect!

Exercise Questions

- This lab provides you questions to answer as you do the lab.
- Please answer these as we will pick and choose some of the questions to ask you during the demo!

Exercise 1

- Use the template code provided.
    - Program tries to compute the factorial of a value in R1.

- Factorial:
    - E.g. 3! = 3 * 2 * 1 = 3
    - E.g. 5! = 5 * 4 * 3 * 2 * 1 = 120

- Factorial Sub-routine:
    - Compute factorial of value in R1 and store result in R0.
    - **Calls multiply sub-routine!**

- Multiply Sub-routine:
    - Multiply R1 * R2 and store product in R0.

- This code is broken **(don’t fix it in this exercise)**!

- Follow the lab manual & figure out why it’s broken.

Exercise 2

- Copy exercise 1 code to exercise 2 file!

- Backup/restore R7 using the fixed-backup technique (i.e. ST and LD).
  - Make sure to do this in **both** FACT and MULT sub-routines!

- The code still **doesn’t** work (don’t fix it yet!)
  - Step through the program and figure out why!

Exercise 3

- Copy exercise 2 code to exercise 3 file!

- Replace all the fixed register backups/restores with a stack!
  - Do it for both the FACT and MULT sub-routines.
  - Just like you used in lab 5 and lab 6!

- The code **works** now!
  - Again, step through the program and figure out why!
  - Why does a stack fix all the issues?

Demo Info

- **Lab Grade Breakdown:**
  - 3 points for attendance.
  - 7 points for demoing (+1 bonus point demo'd before/during Friday).
  - 3 point penalty if lab is demo'd during the next lab session.

- **Tips before you demo:**
  - ***Understand your code!*** (Know what each line does & the input/output)
  - ***Test your code!*** (Check for correct output and that there are no errors)