Welcome to the Lab

CMPUT 229

University of Alberta

Fall 2021

Outline

1. About the Lab
2. Reverse-Polish-Notation
3. Stacks
4. Lab Implementation
5. Assignment Tips
6. CheckMyLab
7. Questions?

Lab Requirements

- Assembly control flow
- Loading and storing from memory
- Using syscalls

Reverse-Polish-Notation (RPN)

- Also known as postfix notation
- In this method of writing mathematical expressions, the operator follows the operands
- Differs from the more common infix notation, where the operator is between the operands
- Examples:

postfix	infix	result
1 2 +	1 + 2	3
5 4 - 1 -	5 - 4 - 1	0

Stacks

- Stacks are a way of storing data in memory
- Is a First-In-First-Out (FIFO) data structure
- Stack terminology:
  - Elements are ”pushed” onto the stack, and ”popped” from the stack
  - The last element to be pushed onto the stack (that is still in the stack) is referred to as the ”top” element
- The only element that can be popped at any given time is the top element

RPN Expressions in This Lab

- Your assignment will require parsing an RPN expression that evaluates to a value
- These RPN expressions will be passed as input to your function in the form of an array of tokens.
- This array is composed of four different types of tokens
    - OPERAND
    - PLUS
    - MINUS
    - TERMINATION
- OPERAND tokens represent operands, PLUS and MINUS tokens represent operators, and the TERMINATION token signifies the end of the expression

Assignment

- Write a function called *calculator* that computes and prints the result of a Reverse-Polish-Notation expression
- Input:
  - *a0*: The address of memory containing an array of tokens making up an RPN expression
  - *a1*: The address of memory at which to begin growing your stack

- Effect:
  - Prints the result of the expression stored in *a0* to standard output

Recommended Strategy

- Iterate over every token in the array passed to your function
- For each token:
  - If it is an OPERAND:
    - Push the operand to your stack
  - If it is a PLUS:
    - Pop two operands from the stack, add the values together, then push the resulting value to the stack
  - If it is a MINUS:
    - Pop two operands from the stack. If A is the first value you popped and B is the second, compute B - A and store the resulting value back to the stack
  - If it is a TERMINATION:
    - Pop a value from the stack, print it, then return from your function

System Calls

- A list of system calls (syscalls) supported by RARS can be found in the RARS help page
- The syscall you will be using in this lab is PrintInt, which prints the integer stored in the a0 register to standard output
- The PrintInt syscall is executed after setting the value of a7 to 1 and using the ecall instruction
- Note: in this lab, do not print any newlines in order to ensure that the grading scripts understand your solution

Stack Growth Direction

- The grading scripts for this lab require that your stack grows backwards in memory
    - Therefore, if the base address of your stack was at 0x10010004, you would push the first item at 0x10010004, push the second at 0x10010000, and so on
    - While it may seem more intuitive to grow in the other direction, this design more closely replicates the stacks you will be encountering in this course
    - Marks will be deducted if your stack grows in the wrong direction

Assignment Tips

- Read specifications very carefully. Pay special attention to what you have to include - we don't want a `main:` label.
- Test your assignments on the lab machines before you submit. That's where we'll be marking them.
- Look at the marksheet to get an idea of how the grading will be done.
- Style marks are easy marks. Format your code like the `example.s` file we provided, and write good comments.
- Be sure to submit code that runs and compiles. Otherwise you will lose many marks.
- Every function in RISC-V needs a return statement. At the end of your function's execution, return with the instruction jr ra

Using CheckMyLab

- CheckMyLab is a great resource for testing your solution before submission
- In order to use it:
  1. Create a copy of your solution file
  2. In this new file, delete the .include ”common.s” line
  3. Copy the entire code from the common.s file, and paste it near the top of your copied file, where the .include ”common.s” line used to be
  4. Submit this modified copy of your solution to CheckMyLab

- This will show you which test cases your solution passed or failed, and how they failed
- Note: do not submit this modified copy for marks, as it will not work with the grading scripts

Questions?