Recitation 11/20

J Compiler Overview

see above

J Language - what happens when I run these?

- 1 2 +
- 1 2 + 3 * -6
- 3 9 6 5 8 - + %
- 5 7 1 gt 5
- 90 0 11 eq if 25 else 3 endif + ; 1 if 3 else 0 endif
- defun testIfs

  5 get_arg1 16 gt if dup 8 gt if dup 4 gt if 0 else 1 endif else 2 endif else 0 endif;

  return

  defun main

  1 while 10 set_arg1 testIfs endwhile

  return

J to RISC: what does the compiler do?

The compiler translates the J code to RISC-V instructions.

Because J is a stack-based language, many instructions will involve popping from and pushing to the stack, marked by the stack pointer (sp)

It is important to note that **you are not simulating the running of the instructions**

In other words, you are not responsible for keeping track of the state of the stack, i.e. what's in the stack after each instruction. Hence, this does not require the use of an internal data structure such as a stack or a deque.

J to RISC: what happens when I run a binary operation?

1 2 +

Read 1 as a LITERAL, store in a register x5

Decrement stack pointer, and push x5 to stack

Read 2 as a LITERAL, store in a register x5

Decrement stack pointer, and push x5 to stack

Pop from stack and store in x5, increment stack pointer

Pop from stack and store in x6, increment stack pointer

Add x5 and x6 and store in x5

Decrement stack pointer, and push x5 to stack

J to RISC - what happens when I call a function?

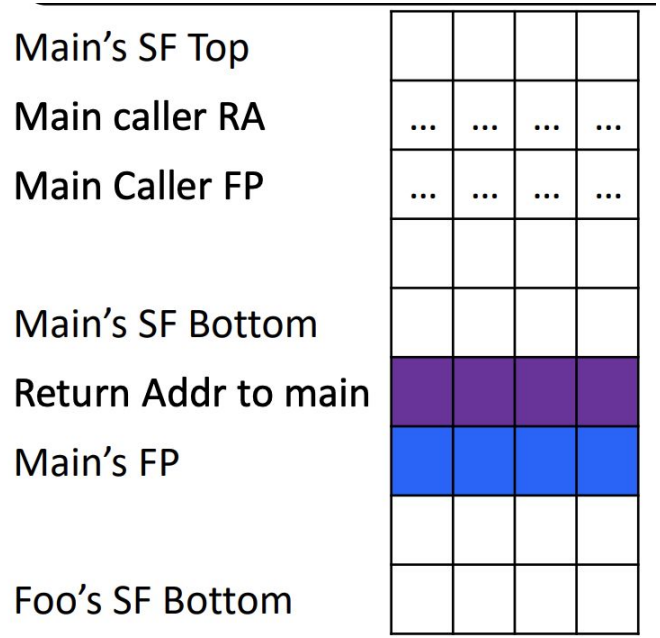
Defun Main

10 50

4 set_arg1

Foo

return



When entering a function:

1. decrement stack pointer
2. save return address
3. save frame pointer
4. frame pointer = stack pointer + 8

When leaving a function:

1. Move stack pointer to frame pointer
2. restore return address
3. restore frame pointer

HW overview

another title:)

jc.c

main()

- Read .j file
- Create output file with .s extension
- Read, compile, write, repeat

```
while (next_token(...)) {
        handle_token()
}
```

- Close files and cleanup!

token.c

next_token(...)

- Read in the next token, skipping space, next line, and comments
- Be careful with multiple space, \t, EOF, and comments

parse_token(...)

- Parse a string token to a token struct
- Potential helpers:
  - Dec_str_to_int(...)
  - Hex_str_to_int(...)
- Be careful with negative numbers, 0x53d

compiler.c

bool handle_token(token t, …) -> what should the arguments be?

Helpers that could be helpful:

pop_stack(...)

push_stack(...)

handle_if(...)

handle_while(...)

handle_token() plan of attack

LITERAL

PLUS, MINUS, MUL, DIV, MOD, AND, OR, NOT

LT, LE, EQ, GE, GT

DEFUN, IDENT, RETURN

SET_ARG, GET_ARG

DUP, SWAP, ROT

IF, WHILE

handle_if(...)

- Handle if
- Handle else
- Handle endif
- Handle bad tokens
- Handle everything else
- How do we handle nested?

```
if x6:
…if branch…
else:

…else branch…
endif
```

```
beqz x6, ELSE
…if branch…
j END_IF
ELSE:
…else branch…
ENDIF:
```

handle_while(...)

- Handle while
- Handle endwhile
- Handle bad tokens
- Handle everything else
- How do we handle nested?

```
WHILE:
beqz x6, END_WHILE
…inside…
j WHILE
END_WHILE:
```