

Announcements

- Program #5 on the web
 - If checked out, do a "cvs update"
- Reading
 - Notes

Recall Computer from Project 1

- Consider a slight extension:
 - Load Rn Rm <value>
 - $Rn = \text{memory}[Rm + \text{<value>}]$
 - Store Rn Rm <value>
 - $\text{Memory}[Rm + \text{<value>}] = Rn$
- Remember
 - R0 is always 0
 - Branch Rn Rm label
 - saves $\text{currentPC} + 1$ into Rn

Calling Functions

- When you call a function, eventually you return
 - Need to know where to return to
- Consider Machine from project #1
 - Branch instruction saved program counter to register
 - Can use that saved address to get back
 - Consider this code:

```
foo:   Branch R3 R0 bar
       Branch R4 R0 other
       Halt
bar:   Load R10 R0 #1234
       Branch R0 R3 0
other: Load R10 R0 #5678
       Branch R3 R0 bar
       Output R10
       Branch R0 R4 0
```

Calling Functions

- Previous Code

- Allows functions to call and return
- But, need to coordinate which registers are used
 - caller and callee need to agree
 - what about recursion?

- Improved Solution

- Keep a stack of return addresses for function
- Dedicate a register to managing the stack
 - called stack pointer (R15)
- Pick a standard register to save return address in
 - caller always uses this one (R14)
 - callee knows where return address is and saves it
 - push return address onto stack onto stack

Call Stack

- R15 contains address of top of stack
- Stack grows down from the end of memory
- Sample Code:
- Setup Stack
 - Load R15 R0 #65535
- Push R14 onto stack
 - Store R14 R15 0
 - Load R2 R0 #1
 - Negate R2
 - Add R15 R2 R15
- Pop top of Stack into R14
 - Load R2 R0 #1
 - Add R15 R2 R15
 - Load R14 R15 0

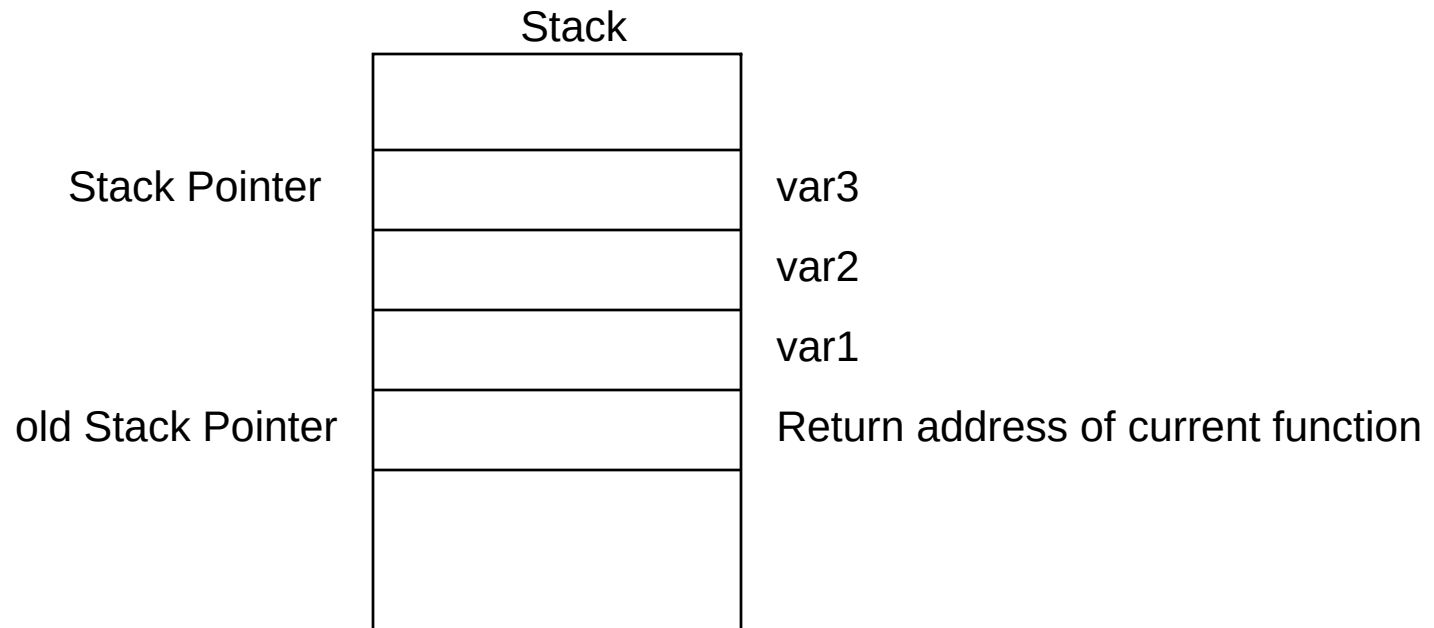
Improved Function Calls

```
main:      Load R15 R0 #65535
           Load R3 R0 #10
           Negate R3
           Branch R14 R0 foo
           Halt
foo:        Store R14 R15 0
           Load R2 R0 #1
           Negate R2
           Add R15 R2 R15
           Bnn R3 done
           Output R3
           Load R2 R0 #1
           Add R2 R3 R3
           Branch R14 R0 foo
done:       Load R2 R0 #1
           Add R15 R2 R15
           Load R14 R15 0
           Branch R0 R14 0
```

Local Variables

- Where to store variables local to a function?
 - Option 1:
 - in specific memory locations (like global vars)
 - what about recursion?
 - Option 2:
 - on the stack
 - specific address of a variables depends on call stack

Local Variables



- How to find local variables on the stack?
 - Use stack pointer to define relative position of items
 - When a function is called
 - save return address on stack
 - move stack pointer by size of locals plus return address

Local Variable Example

```
void foo()
{
    int a, b, c, d;

    a = 30;
    b = 25;
    c = a + b;
    printf("%d\n", c);
}
```

```
foo:   Store R14 R15 0
       Load R2 R0 #5
       Negate R2
       Add R15 R2 R15
       Load R5 R0 #30
       Store R5 R15 1
       Load R6 R0 #25
       Store R6 R15 2
       Add R5 R6 R7
       Store R7 R15 3
       Output R7
       Load R2 R0 #5
       Add R15 R2 R15
       Load R14 R15 0
       Branch R0 R14 0
```

Passing Parameters

- Parameters are like local variables
 - values are only defined for lifetime of function
 - but, they have initial values
- Implementation
 - Option 1:
 - use stack just like local variables
 - copy initial values into locations prior to branch
 - Option 2:
 - put some parameters into registers
 - registers are faster than memory
 - finite number of them means still need option #1
 - put first n parameters into registers, then on stack

Return Values

- Also sort of like a local variable
- Return statement is an assignment to the variable
- Implementation
 - Option 1:
 - Put values on stack in well defined spot
 - Caller can access it after function returns
 - Option 2:
 - Dedicate a register for the return value
 - What about returning a struct?