Announcements

- Program #5 due Thursday
- Reading
  - Bryant & O’Hallaron 3.7 (today)
  - Notes (Thursday)

Recall Computer from Project 3

- Definitions of load word and store word:
  - lw Rn Rm <value>
    - \( Rn = \text{memory}[Rm + <value>] \)
  - sw Rn Rm <value>
    - \( \text{Memory}[Rm + <value>] = Rn \)
- Remember
  - R0 is always 0
  - bal Rn Rm label
    - saves currentPC + 1 into Rn
    - loads R1 with the value of <label> offset by the value of Rm
Calling Functions

- When you call a function, eventually you return
  - Need to know where to return to
- Consider Machine from project #3
  - bal instruction saved program counter to register
  - Can use that saved address to get back
  - Consider this code:

```assembly
foo:          bal R3 R0 bar
            bal R4 R0 other
            halt
bar:          li R10 1234
            li R12 90
            bal R0 R3 0
other:        li R10 5678
            bal R3 R0 bar
            write R10
            bal 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
Call Stack

- R15 contains address of top of stack
- Stack grows down from the end of memory
- Sample Code:
- Setup Stack
  - li R15 65535
- Push R14 onto stack
  - sw R14 R15 0
  - li R2 1
  - neg R2
  - add R15 R2 R15
- Pop top of Stack into R14
  - li R2 1
  - add R15 R2 R15
  - lw R14 R15 0

Improved Function Calls

main:
  li R15 65535
  li R3 10
  neg R3
  bal R14 R0 foo
  halt

foo:
  sw R14 R15 0
  li R2 1
  neg R2
  add R15 R2 R15
  write R15
  beq R3 R0 done
  write R3
  li R2 1
  add R2 R3 R3
  bal R14 R0 foo

done:
  li R2 1
  add R15 R2 R15
  lw R14 R15 0
  write R15
  bal 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

```
Stack

Stack Pointer  var3
               var2
               var1
old Stack Pointer  Return address of current function
```

- **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

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

```
foo:          sw R14 R15 0
           li R2 5
           neg R2
           add R15 R2 R15
           li R5 30
           sw R5 R15 1
           li R6 25
           sw R6 R15 2
           add R5 R6 R7
           sw R7 R15 3
           write R7
           li R2 5
           add R15 R2 R15
           lw R14 R15 0
           bal 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
Parameter Example

```c
void foo(int a)
{
    int b, c, d;
    b = 25;
    c = a + b;
    printf("%d\n", a);
    printf("%d\n", c);
}
```

```assembly
main:
    li R15 65535
    li R3 1234
    li R4 4
    neg R4
    add R15 R4 R13
    sw R3 R13 0
    bal R14 R0 foo
    halt

foo:
    sw R14 R15 0
    li R2 5
    neg R2
    add R15 R2 R15
    lw R5 R15 1
    li R6 25
    sw R6 R15 2
    add R5 R6 R7
    sw R7 R15 3
    write R5
    write R7
    li R2 5
    add R15 R2 R15
    lw R14 R15 0
    bal R0 R14 0
```

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?