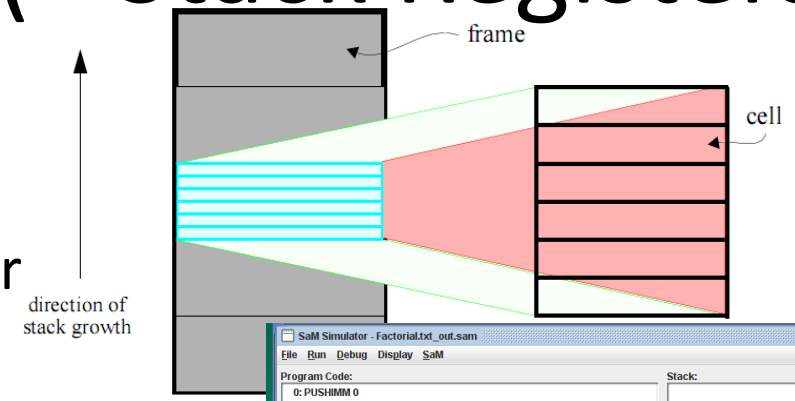
SaM Functions

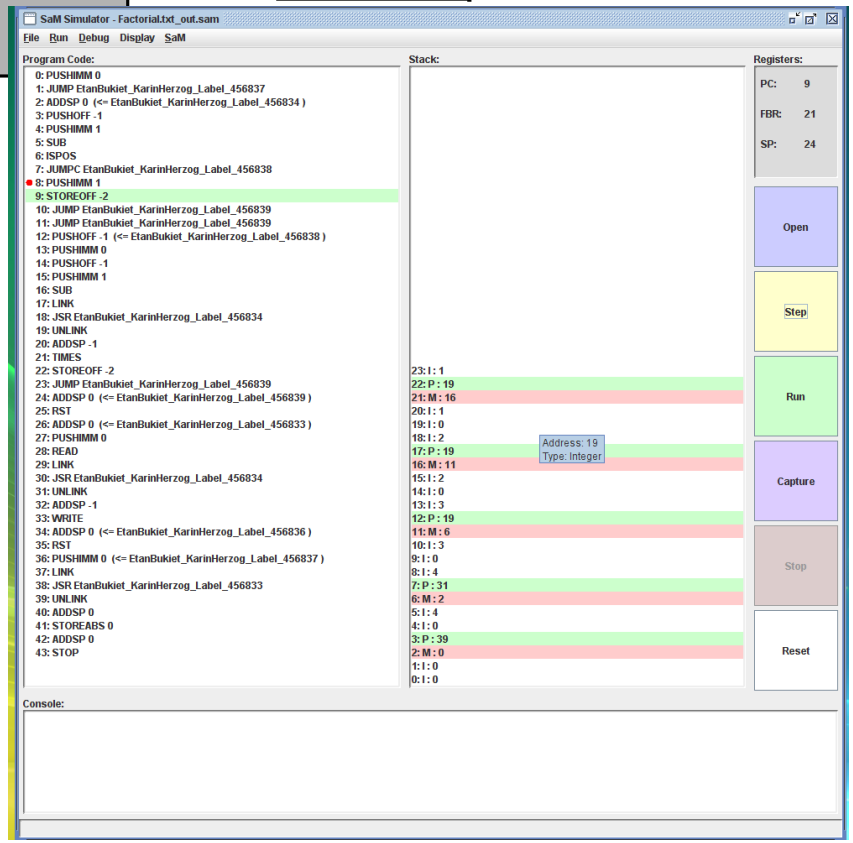
About Functions

- Purpose: Encapsulate procedural information
- Control flow and call order
  - Callee vs. Caller
- Scope (Global vs. Local)
- Parameters
- Recursion
  - Each function call as a separate invocation

The Stack (+ Stack Registers)

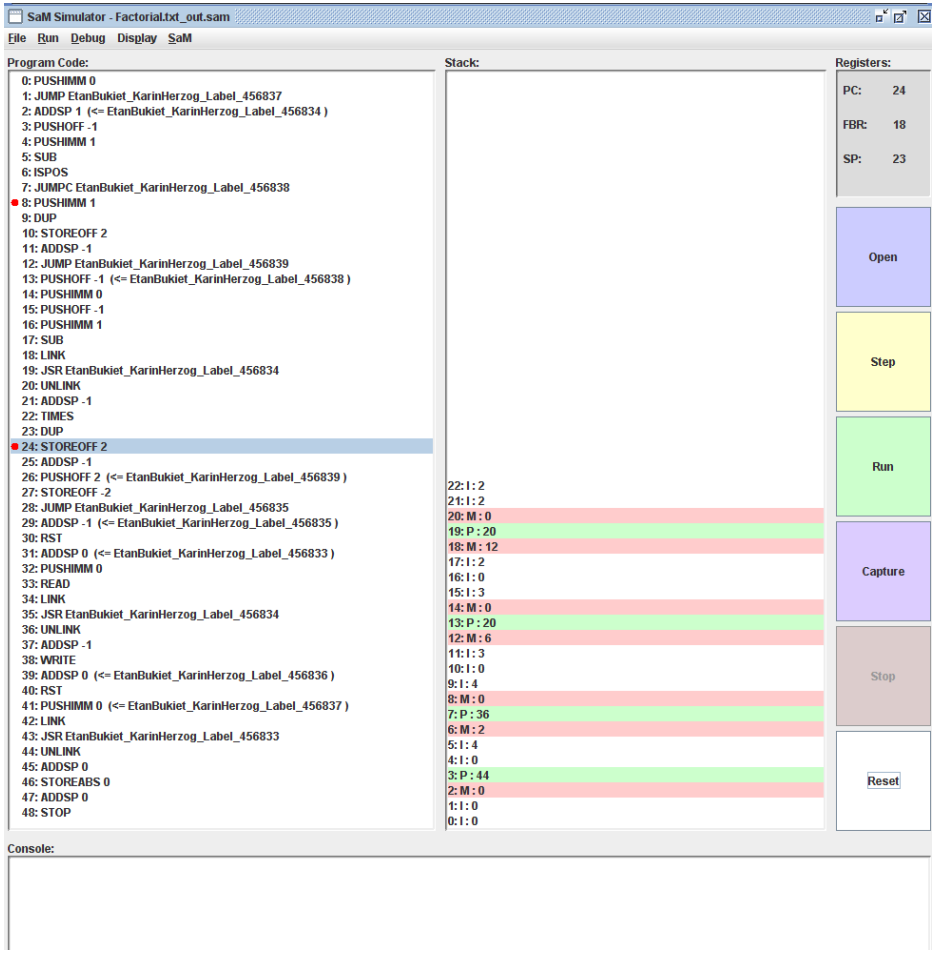
- Stack Registers
  - Frame Base Register
  - Stack Point
  - Program Counter
- Functional Frames
  - Return Value
  - Parameters
  - Return Frame
  - Return Address
  - Local Variables





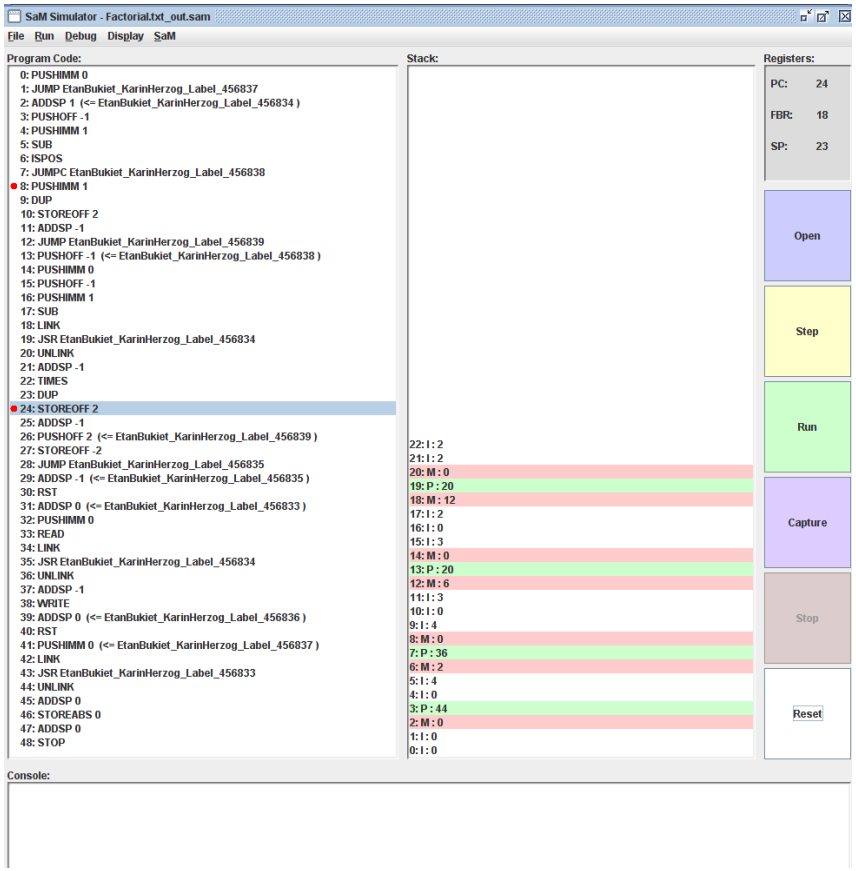
Program Counter, Program Addresses

- Program Counter
- Labels
- Breakpoints



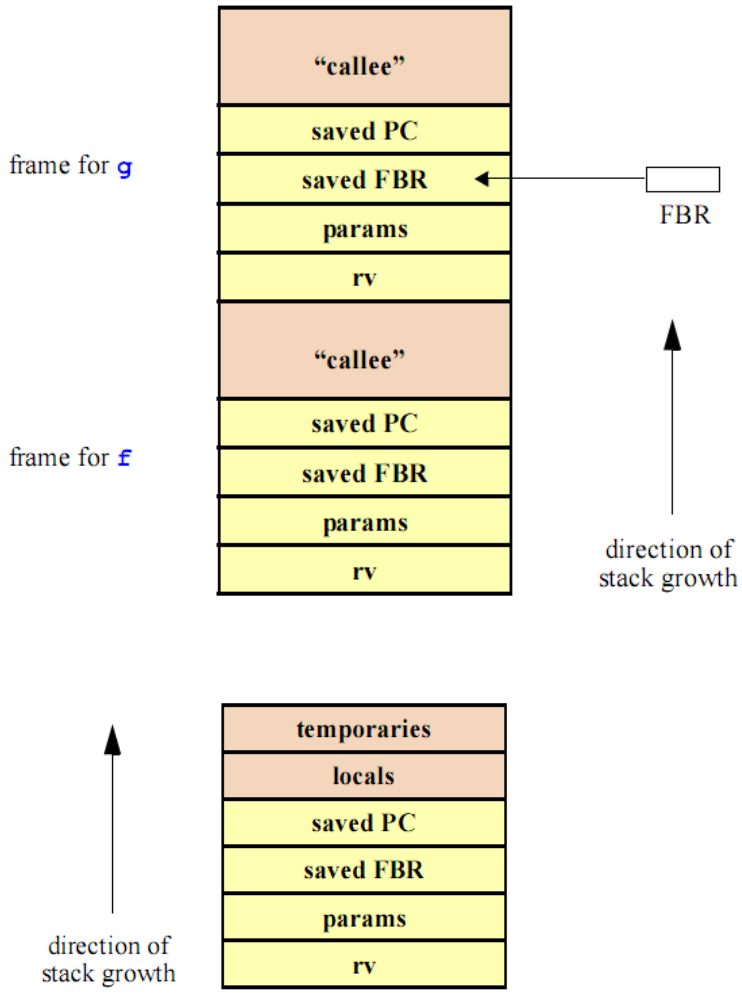
Function Calling Convention

- Caller:
  PUSHIMM 0 // default return
  // push arguments
  LINK
  JSR function_label
  UNLINK
  ADDSP -n // remove params
- Callee:
  function_label:
  ADDSP lv // local vars
  ADDSP -lv
  RST



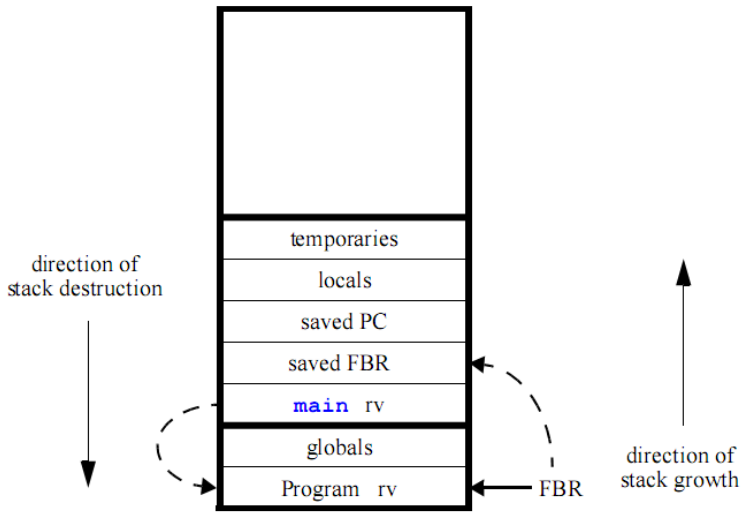
JSR and RST ?

- JSR ?
  - Stack[SP] <= PC+1
  - SP++
  - PC <= label
- RST ?
  - PC <= Stack[SP]
  - SP--



“Program” a wrapper for main?

- The FBR of “Program” is address 0
- The return variable of “Program” is address 0
- The variables of “Program” are any global variables



Code Walkthrough (Bali)

```
prototypes
{
        int factorial(int n);
}

int factorial(int n) { int myLocal; }
{
        unless(n > 1)       { myLocal = 1; }
        else                { myLocal = n * factorial(n-1); }

        return myLocal;
}

int main() { }
{
        print factorial(readInt());
}
```

Code Walkthrough (SaM)

- PUSHIMM 0 // Allocate a return value for the program
- JUMP Label_456837 // Jump to the program initialization code

- Label_456834: // begin function: factorial
- ADDSP 1 // allocate local variable (s)
- // begin unless(n > 1)
- PUSHOFF -1 // retrieve arg0
- PUSHIMM 1 // Push 1 to the stack to begin
- SUB // do the comparison
- ISPOS // ditto
- JUMPC Label_456838 // if (n > 1) // jump to the “else” block

- PUSHIMM 1 // begin (if n <= 1) ie: the “unless” part
- DUP
- STOREOFF 2 // store the value of 1 in the return value
- ADDSP -1
- JUMP Label_456839 // jump until after the “unless…else”

Code Walkthrough (cont…)

- Label_456838: // else part of the “unless…else”
- PUSHOFF -1 // retrieve n
- PUSHIMM 0 // initialize the return value for recursive func: factorial
- PUSHOFF -1 // push argument 1
- PUSHIMM 1 // ditto
- SUB
- LINK // copies curr FBR to stack and assigns FBR = SP
- JSR Label_456834 // jump to the function: factorial
- UNLINK // when we return, put the FBR back
- ADDSP -1 // de-allocate function arguments
- TIMES // perform the multiple n * f(n-1)
- DUP
- STOREOFF 2 // store result in local variable
- ADDSP -1
- Label_456839: // end of “unless…else”
- PUSHOFF 2 // put myLocal on top of the stack
- STOREOFF -2 // and put it as the return value
- Label_456835: // function clean up code
- ADDSP -1 // de-allocate local variables
- RST // end function: factorial

Code Walkthrough (cont…)

- Label_456833: // begin function: main
- ADDSP 0 // allocate local variables (none in this case)
- PUSHIMM 0 // allocate return value for function call (func: factorial)
- READ // readInt built in function call
- LINK // copies curr FBR to stack and assigns FBR = SP
- JSR Label_456834 // jump to the function: “factorial”
- UNLINK // when we return, put the FBR back
- ADDSP -1 // de-allocate argument (s) (the readInt call)
- WRITE // print the return value
- Label_456836: // clean up for function: main
- ADDSP 0 // remove local variables (none of these)
- RST // return to exit the program initialization code

- Label_456837: // begin program initialization code
- PUSHIMM 0 // allocate return value for function: main
- LINK // copies curr FBR to stack and assigns FBR = SP
- JSR Label_456833 // jump to the function: “main”
- UNLINK // when we return, put the FBR back
- STOREABS 0 // move “main’s” return value to the last slot on the stack
- ADDSP 0 // de-allocate global variables
- STOP // end the program

Code Example 2 (C)

```
int GCF(int x, int y) { }
{
unless (!((x % y) == 0))      { return y;          }
else                          { return GCF(y, x % y); }
}
```