Implementing Functions

CS212 – Fall 2004

Functions

int main ( )
{ int i, j; } { ...; i = A( ); ... }

int A ( )
{ int x, y; } { ...; x = A( ); ... }

Basic Idea for Functions

- A new frame (on the stack) is created for each function call
  - We use the FBR (Frame Base Register) to indicate the current frame
  - When a function returns it should “clean up” its frame

int main ( )
{ int i, j; } { ...; i = A( ); ... }

int A ( )
{ int x, y; } { ...; x = A( ); ... }

What’s Kept in a Frame?

- We already have this principle:
  - When an expression is evaluated, the result is left on top of the stack
- What should be left on the stack after a function call?

What Else is Kept in a Frame?

- Another principle:
  - Every time a function is called, it has its own local variables
  - Thus it makes sense to keep a function’s local variables in its frame
- The parameters of a function are also “local variables”
  - They can be kept in the frame, too
Is That It? Nothing Else in a Frame?

- We are using assembly language
  - If we want to jump somewhere and then come back then we must remember where to come back to

```
int main ( )
{ int i, j; }
{ ...; i = A( ); ... }
```

```
int A ( )
{ int x, y; }
{ ...; x = A( ); ... }
```

How Do We Jump Back?

- We can store the return address (i.e., a saved PC value) in the frame, too
- We have provided SAM instructions to store and restore the PC

```
JSR address
  - push PC+1 onto stack; set PC to address
  - Jump to SubRoutine
RST
  - set PC to value on top of stack; SP–
  - Return from SubRoutine
```

Creating a Frame

- Creation of a frame is shared by the caller (calling code) and the callee (the function’s code)

- Caller’s responsibilities
  - Push space for return value
  - Push arguments
  - Create new frame (use LINK = push current FBR and set FBR to SP–1)
  - JSR to callee (push PC+1 and jump to callee)

- callee’s responsibilities
  - Push space for local variables
  - Continue with callee’s code

Clearing a Frame (Clean-up)

- Clearing of a frame is shared by the callee (the function’s code) and the caller (calling code)

- Caller’s responsibilities
  - Restore the FBR (UNLINK)
  - Clear the arguments from stack
  - Note: return value remains on stack

- Callee’s responsibilities
  - Clear local variables from stack
  - RST to caller (clear the saved PC and jump back to calling code)

Access to Frame’s Data

- Data stored in the frame are accessed via offset from the FBR
  - Let p be the number of parameters

```
The first local variable
  - STOREOFF 2
The second local variable
  - STOREOFF 3
The first parameter
  - STOREOFF –p
The second parameter
  - STOREOFF –p + 1
The return value
  - STOREOFF –p – 1
```

An Example

```
int factorial (int n) {
if n < 2 then return 1;
else return n * factorial(n-1);
}
```

```
factorial: PUSHOFF -1
PUSH IMM 2
LESS
JUMPC true
JUMP false
true: PUSH IMM 1
STOREOFF -2 // Store return value
RST // Return
false: PUSHOFF -1
ADDSP 1 // Space for return value
PUSH IMM -1
SUB
LINK
JSR factorial
UNLINK
ADJSIP 1
PUSHOFF -1
PUSH IMM 1
SUB
LINK
JSR factorial
UNLINK
ADJSIP 1
TIMES
STOREOFF -2
RST
// Store return value
// Return
```
Example Calling Code

- We need this “calling code” to help create factorial’s initial frame.

Code Pattern for Caller

- Caller’s responsibilities (frame creation)
  - Push space for return value
  - Push arguments
  - Create new frame (LINK)
  - JSR to callee (push PC+1 and jump to callee)
- Caller’s responsibilities (frame clean-up)
  - Restore the FBR (UNLINK)
  - Clear the arguments from stack

Code Pattern for Callee

- Callee’s responsibilities (frame creation)
  - Push space for local variables
- Callee’s responsibilities (frame clean-up)
  - Clear local variables from stack
  - RST to caller (clear the saved PC and jump back to calling code)

What about the “main” Function?

- The mainFunction can be called by other functions.
  - Thus, it needs to behave as a callee (i.e., it participates in building a frame)
- We need initial code to build the rest of main’s frame

- The mainFunction can be called by other functions.
  - Thus, it needs to behave as a callee (i.e., it participates in building a frame)
- We need initial code to build the rest of main’s frame

main:
  program:
  ADDSP 1 // Return value
  JSR main
  UNLINK // Restore FBR
  main:
  ADDSP 1 // Return value for main
  LINK
  JSR main
  UNLINK // Restore FBR
  main:
  ADDSP 1 // Return value for main
  LINK
  JSR main
  UNLINK // Restore FBR
  main:
  ADDSP 1 // Return value for main
  LINK
  JSR main
  UNLINK // Restore FBR
  main:
  ADDSP 1 // Return value for main
  LINK
  JSR main
  UNLINK // Restore FBR
  main: