CS6013 - Modern Compilers: Theory and Practise

Runtime management

V. Krishna Nandivada

IIT Madras

Runtime management

Parameter passing

Call-by-value
- store values, not addresses
- never restore on return
- arrays, structures, strings are a problem

Call-by-reference
- pass address
- access to formal is indirect reference to actual

Call-by-value-result
- store values, not addresses
- always restore on return
- arrays, structures, strings are a problem

Parameter passing - varargs

What about variable length argument lists?

1. if caller knows that callee expects a variable number
   1. caller can pass number as 0^{th} parameter
   2. callee can find the number directly
2. if caller doesn't know anything about it
   1. callee must be able to determine number
   2. first parameter must be closest to `FP`

Consider `printf` :
- number of parameters determined by the format string
- it assumes the numbers match

MIPS procedure call convention

Registers:

Number	Name	Usage
0	`zero`	Constant 0
1	at	Reserved for assembler
2, 3	v0, v1	Expression evaluation, scalar function results
4–7	a0–a3	first 4 scalar arguments
8–15	t0–t7	Temporaries, caller-saved; caller must save to preserve across calls
16–23	s0–s7	Callee-saved; must be preserved across calls
24, 25	t8, t9	Temporaries, caller-saved; caller must save to preserve across calls
26, 27	k0, k1	Reserved for OS kernel
28	gp	Pointer to global area
29	sp	Stack pointer
30	s8 (fp)	Callee-saved; must be preserved across calls
31	ra	Expression evaluation, pass return address in calls

MIPS procedure call convention

Philosophy:

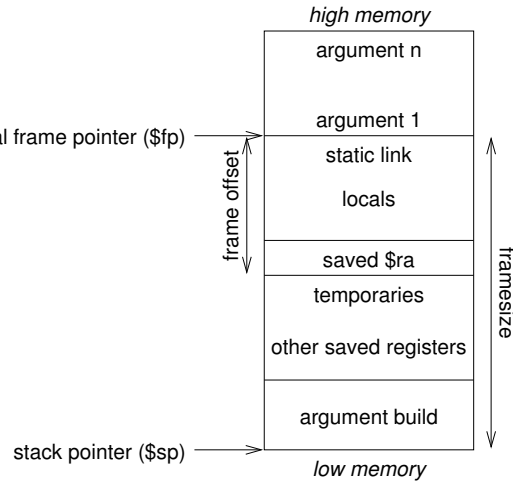
> *Use full, general calling sequence only when necessary; omit portions of it where possible (e.g., avoid using fp register whenever possible)*

Classify routines as:

- non-leaf routines: routines that call other routines
- leaf routines: routines that do not themselves call other routines
  - leaf routines that require stack storage for locals
  - leaf routines that do not require stack storage for locals

MIPS procedure call convention

The stack frame



The “locals” can be accessed by a callee.

MIPS procedure call convention

Pre-call:

1. Pass arguments: use registers $a0 . . . $a3; remaining arguments are pushed on the stack along with save space for $a0 . . . $a3
2. Save caller-saved registers if necessary
3. Execute a `jal` instruction: jumps to target address (callee’s first instruction), saves return address in register $ra

MIPS procedure call convention

Prologue:

1. Leaf procedures that use the stack and non-leaf procedures:
   1. Allocate all stack space needed by routine:
      - local variables
      - saved registers
      - sufficient space for arguments to routines called by this routine

      ```
      subu $sp,framesize
      ```
   2. Save registers ($ra, etc.):

      ```
      sw $31,framesize+frameoffset($sp)
      sw $17,framesize+frameoffset-4($sp)
      sw $16,framesize+frameoffset-8($sp)
      ```

      where `framesize` and `frameoffset` (usually negative) are compile-time constants
2. Emit code for routine

MIPS procedure call convention

Epilogue:

1. Copy return values into result registers (if not already there)
2. Restore saved registers

   ```
   lw reg,framesize+frameoffset-N($sp)
   ```
3. Get return address

   ```
   lw $31,framesize+frameoffset($sp)
   ```
4. Clean up stack

   ```
   addu $sp,framesize
   ```
5. Return

   ```
   j $31
   ```

Code Generation