CS241 Lecture 21

Graham Cooper

July 20th, 2015

Memory Allocation

- alloc.merl - must be linked LAST

Add to the prologue:

- .import init
- .import new
- .import delete

Function init

- sets up allocator's data structures
- call it once in the prologue
- takes a parameter in $2
  - if calling with mips.array (ie. int wain(int * ...))
  - $2 = length of the array
  - else $2 = 0

New/Delete

New

new: $1 = # of words needed
- returns ptr to memory in $3
- if allocation fails, return 0

```
code(new int [expr]) =
code(expr) ($3 <- expr)
add $1, $3, $0
call new
bne $3, $0, $1
add $3, $11, $0
```

Delete

delete: $1 = ptr to mem to be deallocated

```
code(delete [] expr) =
code(expr) ($3 <- expr)
beq $3, $11, skipY
add $1, $3, $0
call delete
skipY:
```

You now have a compiler that handles pointers!!!

Procedures

big picture:
int f(...) {...}
int g(...) {...}
...
int wain(,) {...}

Every function is going to have a prologue/epilogue

Main Prologue/Epilog

- save $1, $2 on stack
- import print, init, new, delete
- set $4, $11 etc if desired
- set $29
- call init
- ... CODE
- reset stack and jr $31

Procedure specific PRologues

- don't need to set .imports, set constants
- set $29!!!!
- save registers
- ... CODE
- restore registers
- reset stack ptr
- return

Saving and Restoring

- procedure should save and store all regs that it will modify
- How do you know which regs?
  - If not sure, save and restore all of them
  - Our code geen scheme doesn't use any registers past 6, and we also use 29 - 31
  - If your scheme uses more regs, just keep track of them
- REMEMBER TO SAVE AND RESTORE REG 29

Two Approaches to Saving and restoring

caller-save cs. callee-save
suppose f calls g

callee-save: g saves all regs it modifies (this is what we are used to), then f doesn't worry about what g does

caller-save: f saves all registers that contain critical data, do this before calling g, g does not have to worry as f has taken care of the saving

Our approach has been: - caller-save for $31, -callee-save for everything else

Q: Who saves $29? Callr or callee?
IF callee-save

- save $29 along with other regs
- if you haven't yet set $29, need to count back in the stack to find the beginning of the frame
- maybe easier to set $29 first then save the regs, but we can't set $29 before we save it!!
- so AT LEAST save $29 first, then set $29, then save the other registers
- OR let the caller (f) save $29 before calling g

```
f(){ ...
...
g()
...
}

f:
push($29)
push($31)
lis $5
.word g
jalr $5
pop($31)
pop($29)
```

Labels

What if a program looks like:

```
int init(...){...}
int print(...) {...}
int else1(...) {...}
```

procedure names match the names of existing labels.
-won't assemble

Solution:

- make sure it can't happen
- use a naming scheme for labels that prevents duplication
- for functions, f, g, h, use the labels Ff, Fg, Fh, ie. reserve labels starting with F as denoted functions
- My compiler will not generate any labels that start with an F

Parameters

- could use regs
- may not be enought
- or push params on stack

```
factor ->ID(expr1, expr2, ...exprn)
code(factor) =
push($29)
push($31)
code(expr1) ($3 <- expr1)
push($3)
...
code(exprn)
push($3)
lis $5
.word F__
jalr $5
```

```
POP ALL ARGUMENTS
pop($31)
pop($29)

procedure -> INT ID (Params) {dcls stmts return expr;}
code(procedure) =
sub $29, $30, $4
push regs
code(dcls) (local vars)
code(stmts)
code(expr)
pop regs
add $30, $29, $4
jr $31
```

What does the stack look like?

```
int f(...){
...
g(...)
...
}
```

	g's frame
local vars	g's frame
saved regs	g's frame
args for g	f's frame
$31	f's frame
$29	f's frame
	wain;s frame

Suppose g is :

```
int g(int a, int b, int c){
int d = 0; int e = 0; int f = 0;
...
}
```

g's symbol table:

Name	Type	Offset
a	int	0
b	int	-4
c	int	-8
d	int	-12
e	int	-16
f	int	-20

Params a,b,c below $29
Local vrs d,e,f above $29
In between are saved regs
so symbol table offsets are wrong!

Params should have positive offsets
Local vars shoudl have negative offsets!

Fix SymbolTable: Add 4 times number of args to all offsets in symboltable

Name	Type	Offset
a	int	12
b	int	8
c	int	4
d	int	0
e	int	-4
f	int	-8

d,e,f are still wrong, we hould push local variables first then save registers!!