CS241 Lecture 19

Graham Cooper

July 13th, 2015

Recal:

- put all local vars on the stack, including params of wain.

```
int wain(int a, int b){return a;}

sw $1, -4 ($30)
sw $2, -8($30)
lis $4
.word 8
sub $30, $30, $4
lw $3, 4($30)
add $30, $30, $4
jr $31
```

name	type	offset from $30
a	int	4
b	int	0

Problem: Can't know the offsets until all decls have been processed because $30 changes with each new decleration

```
int wain(int a, int b){
int c = 0;
return a;
}
```

name	type	offset from $30
a	int	8
b	int	4
c	int	0

Introduce two conventions:

- $4 always contains 4
- $29 points to the bottom of the stack frame

- if offsets are calculated with relation to $29 they will be constant

```
int waint(int a, int b){
int c = 0;
return a;
}

lis $4
.word 4
sub $29, $30, $4
sw $1, -4($30)
sw $2, -8($30)
sw $0, -12($30)
lis $5
.word 12
sub $30, $30, $12
lw $3, 0($29)
add $30, $30, $5
jr $31
```

name	typ	offset from $29
a	int	0
b	int	-4
c	int	-8

$29 is called the frame pointer
What about more complicated programs?

```
int wain(int a, int b){
return a + b;
}
```

In general: For every grammar rule A $\rightarrow \gamma$ build code (A) from code (γ)

Extend Previous Convention:
Use $3 as "output" for every expression

problem:
a + b:

$3 ← eval(a)
$3 ← eval(b)
$3 ← $3 + $3
value of a is lost!!

Need a place to store pending computation. Could use a register:

```
code(a+b) =
code(a) ($3 = a)
add $5, $3, $0 ($5 = a)
code(b) ($3 = b)
add $3, $5, \$3
```

What about a + (b + c)

```
code(a) ($3 = a)
add $5, $3, $0 ($5 = a)
code(b) ($3 = b)
add $6, $3, $0 ($6 = b)
code(c) ($3 = c)
add $3, $6, $3 ($3 = b + c)
add $3, $5, $3 ($3 = a + (b + c))
```

Similarily, a + (b + (c + d)) woudl need 3 edtra regs, we would eventualyl run out of registers.
More general solution: use the stack

```
code(a) ($3 = a)
push($3)
code(b) ($3 = b)
push($3)
code(c) ($3 = c)
push($3)
code(d) ($3 = d)
pop($5) ($5 = c)
add $5, $5, $3 ($3 = c +d)
pop($5) ($5 = b)
```

```
add $3, $5, $3 ($3 = b + (c + d))
pop($5) ($5 = a)
add $3, $5, $3 ($3 = a + (b + (c + d)))
```

In general:
expr1 ->expr2 + term
code(expr1) = code(expr2)
+push($3)
+code(term)
+pop($5)
+add $3, $5, $3

Singleton rules: easy
expr ->term: code(expr) = code(term)

Print: println(expr) prints expr, followed by newline
You did this already! A2P6, A2P7a

Runtime Environment: set of procedures supplied by the compiler (or the OS) to assist programs in their exectuion
msvcrt.dll
- procedure print will be part of the runtime evironment.
- We will provide print.merl

./wlp4gen < prog.wlp4i > prog.asm
java cs241.linkasm < prog.asm > prog.merl
linker prog.merl print.merl > exec.merl
java mips.{twoints or array}exec.merl

We can assume print is available it takes input int $1.

code(println(expr)) = code(expr) ($3 = expr)
+ add $1, $3, $0
+ call print

- remember to save and restore $31

Assignment:

stmt → lvalue = expr
for now: assume lvalue is ID (no ptrs yet)

```
stmt -> lvalue = expr
cod(stmt) = code(expr) ($3 <- expr)
sw $3, _($29)
```

The blank spot above we look up ID's offset in symbol table

What's left - if and while
- need boolean testing
Suggest:
Store the number 1 in register 11
May want to store print in register 10
Code so far:

```
.import print
list $4
.word 4
list $10
.word print
lis $11
,word 1
sub $29, $30, $4
(put all params, locals on stack)
MY CODE WOOO
add $30, $29, $4
jr $31
```

Stuff before my code is the prologue, stuff after my code is the epilogue

boolean testing

```
test -> expr1 < expr2
```

```
code(test) = code(expr1) ($3 <- expr1)
add $6, $3, $0 ($6, <- expr1)
code(expr2) ($3 <- expr2)
slt $3, $6, $3 ($3 <- expr1 < expr2)

test -> expr1 > expr2
treat as expr1 as above\\
```