

Machine-Level Programming: Procedures

These slides based on some provided by the authors of our textbook:
Randal Bryant & David O'Hallaron

Topics

- IA32 stack discipline
- Register saving conventions
- Creating pointers to local variables

IA32 Stack

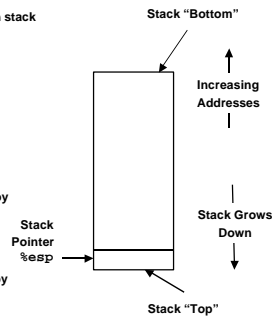
- Region of memory managed with stack discipline
- Register `%esp` indicates lowest allocated position in stack
– i.e., address of top element

Pushing

- `pushl Src`
- Fetch operand at `Src`
- Decrement `%esp` by 4
- Write operand at address given by `%esp`

Popping

- `popl Dest`
- Read operand at address given by `%esp`
- Increment `%esp` by 4
- Write to `Dest`



CompOrg Fall 2002 - Procedures & Stack

2

Stack Operation Examples

		<code>pushl %eax</code>		<code>popl %edx</code>	
0x110		0x110		0x110	
0x10c		0x10c		0x10c	
0x108	123	0x108	123	0x108	123
		0x104	213		
<code>%eax</code>	213	<code>%eax</code>	213	<code>%eax</code>	213
<code>%edx</code>	555	<code>%edx</code>	555	<code>%edx</code>	213
<code>%esp</code>	0x108	<code>%esp</code>	0x104	<code>%esp</code>	0x108

CompOrg Fall 2002 - Procedures & Stack

3

Procedure Control Flow

Use stack to support procedure call and return

Procedure call:

`call label` Push return address on stack; Jump to `label`

Return address value

- Address of instruction beyond `call`
- Example from disassembly

```
804854e: e8 3d 06 00 00 call 8048b90 <main>
8048553: 50          pushl %eax
- Return address = 0x8048553
```

Procedure return:

- `ret` Pop address from stack; Jump to address

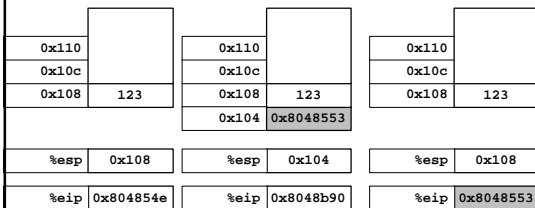
CompOrg Fall 2002 - Procedures & Stack

4

Procedure Call / Return Example

```
804854e: e8 3d 06 00 00 call 8048b90 <main>
8048553: 50          pushl %eax
```

`call 8048b90` `ret`



%eip is program counter

CompOrg Fall 2002 - Procedures & Stack

5

Stack-Based Languages

Languages that Support Recursion

- e.g., C, Pascal, Java
- Code must be "*Reentrant*"
 - Multiple simultaneous instantiations of single procedure
- Need some place to store state of each instantiation
 - Arguments
 - Local variables
 - Return pointer

Stack Discipline

- State for given procedure needed for limited time
 - From when called to when return
- Callee returns before caller does

Stack Allocated in *Frames*

- state for single procedure instantiation

CompOrg Fall 2002 - Procedures & Stack

6

Call Chain Example

Code Structure

```

yoo(...)
{
  .
  .
  who();
  .
}

who(...)
{
  .
  .
  amI();
  .
}

amI(...)
{
  .
  .
  amI();
  .
}

```

- Procedure amI recursive

Call Chain

```

yoo
  ↓
who
  ↓
amI
  ↓
amI
  ↓
amI

```

CompOrg Fall 2002 - Procedures & Stack 7

IA32 Stack Structure

Stack Growth

- Toward lower addresses

Stack Pointer

- Address of next available location in stack
- Use register %esp

Frame Pointer

- Start of current stack frame
- Use register %ebp

CompOrg Fall 2002 - Procedures & Stack 8

IA32/Linux Stack Frame

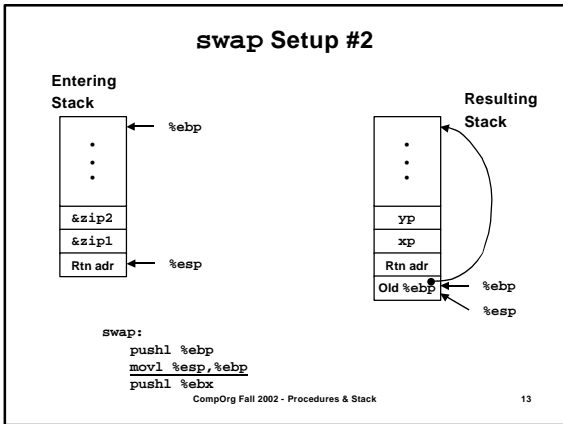
Callee Stack Frame ("Top" to Bottom)

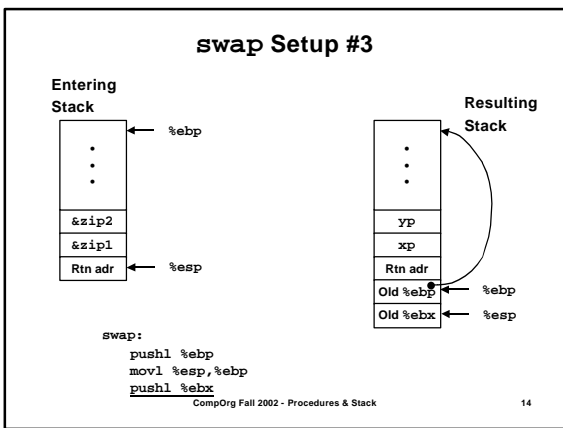
- Parameters for called functions
- Local variables
 - If can't keep in registers
- Saved register context
- Old frame pointer

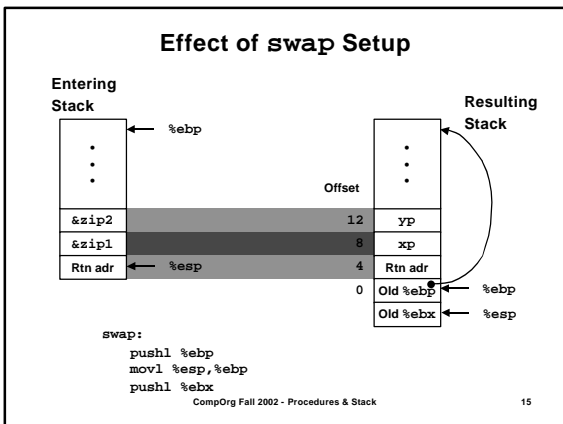
Caller Stack Frame

- Return address
 - Pushed by call instruction
- Arguments for this call

CompOrg Fall 2002 - Procedures & Stack 9







swap Finish #1

swap's Stack

swap's Stack

```

movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
  
```

Observation

- Saved & restored register %ebx

CompOrg Fall 2002 - Procedures & Stack 16

swap Finish #2

swap's Stack

swap's Stack

```

movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
  
```

CompOrg Fall 2002 - Procedures & Stack 17

swap Finish #3

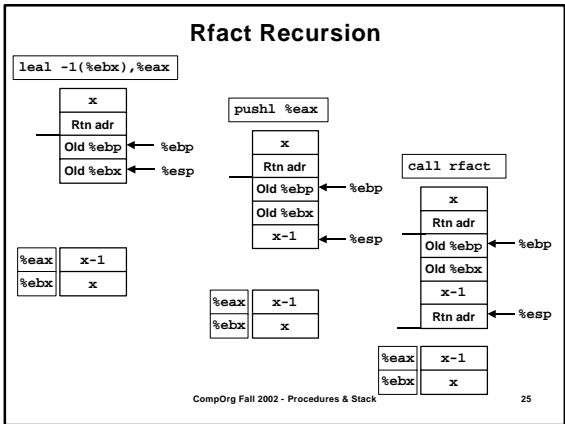
swap's Stack

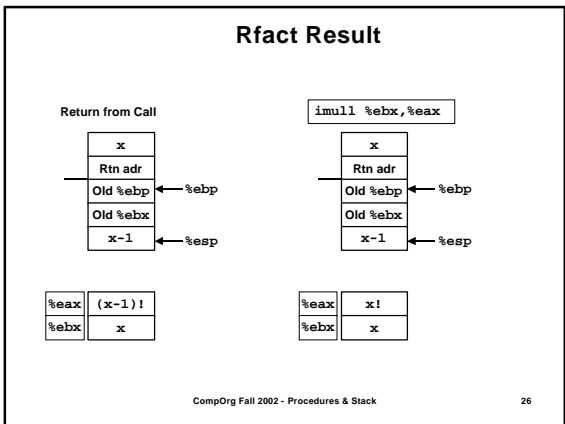
swap's Stack

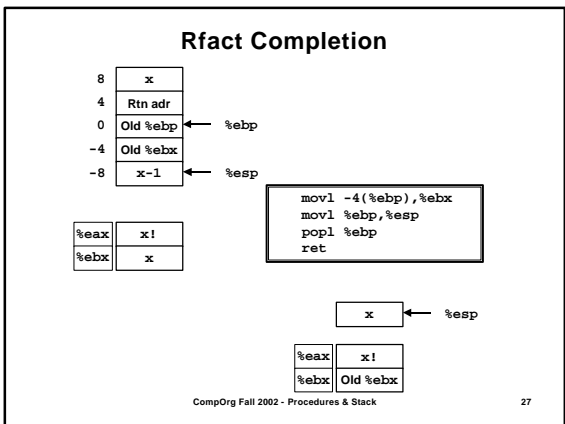
```

movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
  
```

CompOrg Fall 2002 - Procedures & Stack 18







Pointer Code

Recursive Procedure

```
void s_helper
(int x, int *accum)
{
    if (x <= 1)
        return;
    else {
        int z = *accum * x;
        *accum = z;
        s_helper (x-1, accum);
    }
}
```

Top-Level Call

```
int sfact(int x)
{
    int val = 1;
    s_helper(x, &val);
    return val;
}
```

- Pass pointer to update location
- Uses tail recursion
 - But GCC only partially optimizes it

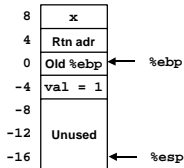
CompOrg Fall 2002 - Procedures & Stack

28

Creating & Initializing Pointer

Initial part of sfact

```
_sfact:
    pushl %ebp      # Save %ebp
    movl %esp,%ebp # Set %ebp
    subl $16,%esp  # Add 16 bytes
    movl 8(%ebp),%edx # edx = x
    movl $1,-4(%ebp) # val = 1
```



Using Stack for Local Variable

- Variable val must be stored on stack
 - Need to create pointer to it
- Compute pointer as -4(%ebp)
- Push on stack as second argument

```
int sfact(int x)
{
    int val = 1;
    s_helper(x, &val);
    return val;
}
```

CompOrg Fall 2002 - Procedures & Stack

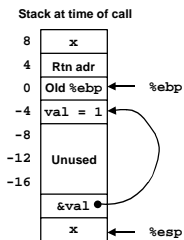
29

Passing Pointer

Calling s_helper from sfact

```
leal -4(%ebp),%eax # Compute &val
pushl %eax         # Push on stack
pushl %edx         # Push x
call s_helper      # call
movl -4(%ebp),%eax # Return val
• • •             # Finish
```

```
int sfact(int x)
{
    int val = 1;
    s_helper(x, &val);
    return val;
}
```



CompOrg Fall 2002 - Procedures & Stack

30

Using Pointer

```
void s_helper
(int x, int *accum)
{
    . . .
    int z = *accum * x;
    *accum = z;
    . . .
}
```

```
. . .
movl %ecx,%eax # z = x
imull (%edx),%eax # z *= *accum
movl %eax,(%edx) # *accum = z
. . .
```

- Register %ecx holds x
- Register %edx holds accum
 - Use access (%edx) to reference memory

CompOrg Fall 2002 - Procedures & Stack

31

Tail Recursion

Tail Recursive Procedure

```
int t_helper
(int x, int val)
{
    if (x <= 1)
        return val;
    return
        t_helper(x-1, val*x);
}
```

General Form

```
t_helper(x, val)
{
    . . .
    return
        t_helper(Xexpr, Vexpr)
}
```

Top-Level Call

```
int tfact(int x)
{
    return t_helper(x, 1);
}
```

Form

- Directly return value returned by recursive call

Consequence

- Can convert into loop

CompOrg Fall 2002 - Procedures & Stack

32

Removing Tail Recursion

Optimized General Form

```
t_helper(x, val)
{
    start:
    . . .
    val = Vexpr;
    x = Xexpr;
    goto start;
}
```

Resulting Code

```
int t_helper
(int x, int val)
{
    start:
    if (x <= 1)
        return val;
    val = val*x;
    x = x-1;
    goto start;
}
```

Effect of Optimization

- Turn recursive chain into single procedure
- No stack frame needed
- Constant space requirement
 - Vs. linear for recursive version

CompOrg Fall 2002 - Procedures & Stack

33

Generated Code for Tail Recursive Proc.

Optimized Form

```
int t_helper  
(int x, int val)  
{  
  start:  
  if (x <= 1)  
    return val;  
  val = val*x;  
  x = x-1;  
  goto start;  
}
```

Registers

```
$edx x  
$ecx val
```

Code for Loop

```
# %edx = x  
# %ecx = val  
L53:      # start:  
        cmpl $1,%edx    # x : 1  
        jle L52        # if <= goto done  
        movl %edx,%eax  # eax = x  
        imull %ecx,%eax # eax = val * x  
        decl %edx      # x--  
        movl %eax,%ecx  # val = val * x  
        jmp L53        # goto start  
L52:      # done:
```

CompOrg Fall 2002 - Procedures & Stack

34

Summary

The Stack Makes Recursion Work

- Private storage for each *instance* of procedure call
 - Instantiations don't clobber each other
 - Addressing of locals + arguments can be relative to stack positions
- Can be managed by stack discipline
 - Procedures return in inverse order of calls

IA32 Procedures Combination of Instructions + Conventions

- Call / Ret instructions
- Register usage conventions
 - Caller / Callee save
 - %ebp and %esp
- Stack frame organization conventions

CompOrg Fall 2002 - Procedures & Stack

35
