Giving credit where credit is due

- Most of slides for this lecture are based on slides created by Drs. Bryant and O’Hallaron, Carnegie Mellon University.
- I have modified them and added new slides.

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

IA32 Stack
- Region of memory managed with stack discipline
- Grows toward lower addresses
- Register %esp indicates lowest stack address
- Address of top element

IA32 Stack Pushing
- pushl Src
- Fetch operand at Src
- Decrement %esp by 4
- Write operand at address given by %esp

IA32 Stack Popping
- popl Dest
- Read operand at address given by %esp
- Increment %esp by 4
- Write to Dest
Stack Operation Examples

<table>
<thead>
<tr>
<th>$%esp$</th>
<th>$%eax$</th>
<th>$%edx$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10c</td>
<td>0x10c</td>
<td>0x10c</td>
</tr>
<tr>
<td>0x108</td>
<td>123</td>
<td>0x108</td>
</tr>
<tr>
<td>0x104</td>
<td>213</td>
<td>0x104</td>
</tr>
</tbody>
</table>

pushl $%eax$

<table>
<thead>
<tr>
<th>$%esp$</th>
<th>$%eax$</th>
<th>$%edx$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10c</td>
<td>0x10c</td>
<td>0x10c</td>
</tr>
<tr>
<td>0x108</td>
<td>123</td>
<td>0x108</td>
</tr>
<tr>
<td>0x104</td>
<td>213</td>
<td>0x104</td>
</tr>
</tbody>
</table>

popl $%edx$

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>
```

- Return address = 0x8048553

Procedure return:

``` ret
```

Pop address from stack; Jump to address

Procedure Call Example

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

Procedure Return Example

```
8048591: c3             ret
```

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

Stack-Chain Example

```
code

yoo(…)
{
  who();
}

who(…)
{
  amI();
  amI();
}

amI(…)
{
  amI();
}
```

Call Chain Example
Stack Frames

Contents
- Local variables
- Return information
- Temporary space

Management
- Space allocated when enter procedure
- “Set-up” code
- Deallocated when return
- “Finish” code

Pointers
- Stack pointer %esp indicates stack top
- Frame pointer %ebp indicates start of current frame

Stack Operation

Call Chain

```
who();
```

```
ctx();
```
IA32/Linux Stack Frame

Current Stack Frame ("Top" to Bottom)
- Parameters for function about to call
  - "Argument build"
- 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

Revisiting swap

void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}

swapping:

1. Push old return address:
   pushl %ebp
2. Copy frame pointer:
   movl %esp,%ebp
3. Push ebx:
   pushl %ebx
4. Copy arguments:
   movl 12(%ebp),%ecx
   movl 8(%ebp),%edx
   movl (%ecx),%eax
   movl (%edx),%ebx
5. Swap arguments:
   movl -4(%ebp),%ebx
   movl %ebx,%ecx
   movl %ebx,%eax
6. Restore return address:
   movl -4(%ebp),%esp
7. Return:
   ret

Resulting Stack:

- %ebp
- %esp
- %eax
- %ebx
- %ecx
- %edx
- %ebx
- %eax
- %esp
- %esp
- %esp

Calling swap from call_swap

void call_swap()
{
  swap(&zip1, &zip2);
}

swap Setup #1

Entering Stack

Resulting Stack

- %ebp
- %esp
- %esp
- %esp

Swap:

- pushl %ebp
- movl %esp,%ebp
- pushl %esp

swap Setup #2

Entering Stack

Resulting Stack

- %ebp
- %esp
- %esp
- %esp

Swap:

- pushl %ebp
- movl %esp,%ebp
- pushl %esp

swap Setup #3

Entering Stack

Resulting Stack

- %ebp
- %esp
- %esp
- %esp

Swap:

- pushl %ebp
- movl %esp,%ebp
- pushl %esp

Page 5
Effect of swap Setup

Entering Stack

Resulting Stack

Body

offset (relative to %ebp)

movl 12(%ebp),%eax # get yp
movl 8(%ebp),%edx  # get xp

Observation

Saved & restored register %ebx

Didn't do so for %eax, %ecx, or %edx

swap Finish #1

swap Finish #2

swap Finish #3

swap Finish #4

Register Saving Conventions

When procedure yoo calls who:
- yoo is the caller, who is the callee

Can Register be Used for Temporary Storage?

yoo:

movl $15213, %edx
movl 8(%ebp), %edx
addl $15112, %edx
ret

who:

movl -4(%ebp), %ebx
movl 4(%ebp), %edx
addl $15112, %edx
ret

Contents of register %edx overwritten by who

-
Register Saving Conventions

When procedure `yoo` calls `who`:
- `yoo` is the caller, who is the callee

Can Register be Used for Temporary Storage?

Conventions
- "Caller Save"
  - Caller saves temporary in its frame before calling
- "Callee Save"
  - Callee saves temporary in its frame before using

Can Register be Used for Temporary Storage?

- "Caller Save"
  - Caller saves temporary in its frame before calling
- "Callee Save"
  - Callee saves temporary in its frame before using

Integers and Registers

- Two have special uses:
  - `%ebp`
  - `%esp`
- Three managed as callee-save:
  - `%ebx`
  - `%esi`
  - `%edi`
- Old values saved on stack prior to using:
- Three managed as caller-save:
  - `%eax`
  - `%edx`
  - `%ecx`
- Do what you please, but expect any callee to do so, as well.
- Register `%eax` also stores returned value

Recursive Factorial

```c
int rfact(int x) {
    int rval;
    if (x <= 1) return 1;
    rval = rfact(x-1); return rval * x;
}
```

Registers
- `%eax` used without first saving
- `%ebx` used, but save at beginning & restore at end

Rfact Body

Recursion

```c
int rfact(int x) {
    int rval;
    if (x <= 1) return 1;
    rval = rfact(x-1); return rval * x;
}
```

Registers
- `%ebx` Stored value of `x`
- `%eax` Temporary value of `x-1`
  - Returned value from `rfact(x-1)`
  - Returned value from this call

IA32/Linux Register Usage

<table>
<thead>
<tr>
<th>Integer Registers</th>
<th>Caller-Save Temporaries</th>
<th>Callee-Save Temporaries</th>
<th>Special</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax, %edx, %ecx</td>
<td>%esp</td>
<td>%ebp</td>
<td>%esp</td>
</tr>
<tr>
<td>%ebx, %esi, %edi</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rfact Stack Setup

```c
.globl rfact
.rfact .type rfact,@function
.rfact:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 8(%ebp),%ebx
    cmpl $1,%ebx
    jle .L78
    leal -1(%ebx),%eax
    pushl %eax
    call rfact
    imull %ebx,%eax
    jmp .L79
.L78:
    movl $1,%eax
.L79:
    movl -4(%ebp),%eax
    movl %ebp, %esp
    popl %ebp
    ret
```

Rfact Recursion

```c
.int rfact(int x) {
    int rval;
    if (x <= 1) return 1;
    rval = rfact(x-1); return rval * x;
}
```

```c
.globl rfact
.rfact .type rfact,@function
.rfact:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 8(%ebp),%ebx
    cmpl $1,%ebx
    jle .L78
    leal -1(%ebx),%eax
    pushl %eax
    call rfact
    imull %ebx,%eax
    jmp .L79
.L78:
    movl $1,%eax
.L79:
    movl -4(%ebp),%eax
    movl %ebp, %esp
    popl %ebp
    ret
```

```
```
Rfact Result

Return from Call

\[
\begin{align*}
&\text{Pre} \text{ ebp} \\
&\text{Old} \text{ ebp} \\
&\text{Rtn adr} \\
&\text{ebp} \\
&\text{eax} \\
&\text{ebx} \\
&\text{esp} \\
&\text{x-1} \\
\end{align*}
\]

Assume that \( rfact(x-1) \) returns \( (x-1)! \) in register \( \text{eax} \)

Rfact Completion

\[
\begin{align*}
&\text{Pre} \text{ ebp} \\
&\text{Old} \text{ ebp} \\
&\text{Rtn adr} \\
&\text{ebp} \\
&\text{eax} \\
&\text{ebx} \\
&\text{esp} \\
&\text{x-1} \\
\end{align*}
\]

Pointer Code

Recursive Procedure

\[
\text{void s_helper} \ (\text{int x, int *accum}) \\
\{ \ \\
\quad \text{if (x <= 1)} \ \\
\quad \quad \text{return;} \ \\
\quad \text{else} \ \\
\quad \quad \{ \ \\
\quad \quad \quad \text{int z = *accum * x;} \ \\
\quad \quad \quad \text{*accum = z;} \ \\
\quad \quad \quad \text{s_helper(x-1,accum);} \ \\
\quad \quad \} \ \\
\} \ \\
\]

* Pass pointer to update location

Calling s_helper from sfact

\[
\begin{align*}
&\text{leal -4(%ebp),eax} \ # \text{Compute &val} \ \\
&\text{pushl %eax} \ # \text{Push on stack} \ \\
&\text{pushl %edx} \ # \text{Push } x \ \\
&\text{call s_helper} \ # \text{Call} \ \\
&\text{movl -4(%ebp),%eax} \ # \text{Return val} \\
&\text{ret} \ # \text{Finish} \ \\
\end{align*}
\]

Passing Pointer

Stack at time of call

\[
\begin{align*}
&\text{Pre} \text{ ebp} \\
&\text{Old} \text{ ebp} \\
&\text{Rtn adr} \\
&\text{ebp} \\
&\text{eax} \\
&\text{ebx} \\
&\text{esp} \\
&\text{x-1} \\
\end{align*}
\]

Using Pointer

\[
\begin{align*}
&\text{void s_helper} \ (\text{int x, int *accum}) \\
&\{ \ \\
&\quad \text{int val = 1;} \ \\
&\quad \text{s_helper(x, &val);} \ \\
&\quad \text{return val;} \ \\
&\} \ \\
\]

* Register \( \text{ecx} \) holds \( x \) \\
* Register \( \text{edx} \) holds pointer to \( \text{accum} \) \\
* Use access \( \text{(%edx)} \) to reference memory
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