Last Time

- For loops
  - for loop → while loop → do-while loop → goto version
  - for loop → while loop → goto “jump to middle” version

- Switch statements
  - Jump tables: \texttt{jmp *.L62 (, edx, 4)}
  - Decision trees (not shown)
  - Decision trees (not shown)
    when we have \textit{sparse} cases

Machine Programming III: Control Flow

- Stacks
- Procedures
- Parameter passing
IA32 Stack

- Region of memory managed with a stack discipline
- Grows toward lower addresses
- Customarily shown “upside-down”

- Register %esp contains lowest stack address = address of “top” element

Register %esp contains lowest stack address = address of “top” element

IA32 Stack: Push

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

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

---

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:**
  - 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
Procedure Call Example

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

```
call 8048b90
```

| 0x110 | 0x110 |
| 0x10c | 0x10c |
| 0x108 | 123 |
| 0x104 | 0x8048553 |

| %esp | 0x108 |
| %esp | 0x104 |
| %eip | 0x804854e |
| %eip | 0x8048553 |

%eip: program counter

---

```
call 8048b90
```

```
call 8048b90 <main>
```

```
pushl %eax
```

| 0x110 | 0x110 |
| 0x10c | 0x10c |
| 0x108 | 123 |
| 0x104 | 0x8048553 |

| %esp | 0x108 |
| %esp | 0x104 |
| %eip | 0x804854e |
| %eip | 0x8048553 |

%eip: program counter

```
+ 0x000063d
```
Procedure Return Example

8048591: c3 ret

Stack-Based Languages

- Languages that support recursion
  - e.g., C, Pascal, Java
  - Code must be re-entrant
    - Multiple simultaneous instantiations of single procedure
  - Need some place to store state of each instantiation
    - Arguments
    - Local variables
    - Return pointer

- Stack discipline
  - State for a given procedure needed for a limited time
    - Starting from when it is called to when it returns
  - Callee always returns before caller does

- Stack allocated in frames
  - State for a single procedure instantiation
Call Chain Example

Example Call Chain

Procedure amI is recursive (calls itself)

Stack Frames

Contents
- Local variables
- Return information
- Temporary space

Management
- Space allocated when procedure is entered
  - “Set-up” code
- Space deallocated upon return
  - “Finish” code
Example

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

Stack

```c
who

%ebp
%esp
```

Example

```c
who(...) {
  ...
  ami();
  ...
  ami();
  ...
}
```

Stack

```c
who

%ebp
%esp
```
Example

```c
amI(...) {
    
    amI();
    
}
```

Stack

Example

```c
amI(...) {
    
    amI();
    
}
```

Stack
Example

```c
amI(...) {
  .
  .
  amI();
  .
  .
}
```

Stack

```
%ebp
%esp
```

Example

```c
amI(...) {
  .
  .
  amI();
  .
  .
}
```

Stack

```
%ebp
%esp
```
Example

```c
amI(...) {
    ...
    amI();
    ...
}
```

Example

```c
who(...) {
    ...
    amI();
    ...
    amI();
    ...
}
```
Example

```c
amI(...) {
  ...
  ...
  ...
  ...
}
```

Stack

```
yoo
who
amI
%ebp
%esp
```

Example

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

Stack

```
yoo
who
%ebp
%esp
```
Example

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

Stack

- `yoo`
- `%ebp`
- `%esp`

IA32/Linux Stack Frame

- **Current Stack Frame ("Top" to Bottom)**
  - Old frame pointer
  - Local variables
    - If can’t be just kept in registers
  - Saved register context
    - When reusing registers
  - "Argument build area"
    - Parameters for function about to be called

- **Caller Stack Frame**
  - Return address
    - Pushed by `call` instruction
  - Arguments for this call

Frame pointer `%ebp`

Caller Frame

Arguments

Saved Registers

Local Variables

Stack pointer `%esp`

Argument Build

Saved %ebp

Old %ebp

Return Addr
Revisiting swap

```c
int zip1 = 15213;
int zip2 = 98195;

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

Calling swap from call_swap

```c
call_swap:
    • • •
pushl $zip2     # Global Var
pushl $zip1     # Global Var
call swap
    • • •
```

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

Resulting Stack

```
%esp

Rtn adr

&zip2

&zip1
```

Calling swap from call_swap

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

swap:

```c
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx

    movl 12(%ebp),%ecx
    movl 8(%ebp),%edx
    movl (%ecx),%eax
    movl (%edx),%ebx
    movl %eax,(%edx)
    movl %ebx,(%ecx)

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

Set Up

Body

Finish
swap Setup #1

Entering Stack

Resulting Stack

\[
\begin{align*}
\text{swap:} & \quad \text{pushl } \%ebp \\
& \quad \text{movl } \%esp, \%ebp \\
& \quad \text{pushl } \%ebx
\end{align*}
\]
swap Setup #1

Entering Stack

%ebp

%esp

&zip2

&zip1

Rtn adr

Resulting Stack

%ebp

%esp

yp

xp

Rtn adr

Old %ebp

swap:

pushl %ebp

movl %esp,%ebp

pushl %ebx

CSE 351 - Inaugural Edition - Spring 2010

29
swap Setup #1

Entering Stack  
\[ \begin{align*}
\cdot & \quad \cdot \\
\cdot & \quad \cdot \\
\text{$\&\text{zip1}$} & \quad \text{12} \\
\text{\textit{Rtn adr}} & \quad \%\text{esp} \\
\text{\textit{zip2}} & \quad \%\text{ebp} \\
\end{align*} \]

Resulting Stack  
\[ \begin{align*}
\cdot & \quad \cdot \\
\cdot & \quad \cdot \\
\text{\textit{yp}} & \quad \text{8} \\
\text{\textit{xp}} & \quad \%\text{ebp} \\
\text{\textit{Rtn adr}} & \quad \%\text{esp} \\
\end{align*} \]

\[
\text{movl } 12(\%\text{ebp}),\%\text{ecx} \quad \# \text{ get } \textit{yp} \\
\text{movl } 8(\%\text{ebp}),\%\text{edx} \quad \# \text{ get } \textit{xp} \\
\ldots
\]

Observation: Saved and restored register \%\text{ebx}
swap Finish #2

swap's Stack

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

Resulting Stack
swap Finish #2

swap's Stack

\[
\begin{array}{c}
\text{YP} \\
\text{xp} \\
\text{Rtn adr} \\
\text{Old %ebp} \\
\text{Old %ebx} \\
\end{array}
\]

\[
\begin{array}{c}
\text{YP} \\
\text{xp} \\
\text{Rtn adr} \\
\text{Old %ebp} \\
\text{Old %esp} \\
\end{array}
\]

\[
\text{movl } -4(\%ebp),\%ebx \\
\text{movl } \%ebp,\%esp \\
\text{popl } \%ebp \\
\text{ret}
\]

swap Finish #3

swap's Stack

\[
\begin{array}{c}
\text{YP} \\
\text{xp} \\
\text{Rtn adr} \\
\text{Old %ebp} \\
\text{Old %ebx} \\
\end{array}
\]

\[
\begin{array}{c}
\text{YP} \\
\text{xp} \\
\text{Rtn adr} \\
\text{Old %ebp} \\
\text{Old %esp} \\
\end{array}
\]

\[
\text{movl } -4(\%ebp),\%ebx \\
\text{movl } \%ebp,\%esp \\
\text{popl } \%ebp \\
\text{ret}
\]
swap Finish #4

swap's Stack

\[
\begin{align*}
\text{\textbullet} & \quad \text{\textbullet} \\
\text{\textbullet} & \quad \text{\textbullet} \\
\text{YP} & \\
\text{xp} & \\
\text{Rtn adr} & \\
\text{Old \%ebp} & \quad \%ebp \\
\text{Old \%ebx} & \quad \%esp
\end{align*}
\]

Resulting Stack

\[
\begin{align*}
\text{\textbullet} & \quad \text{\textbullet} \\
\text{\textbullet} & \quad \text{\textbullet} \\
\text{YP} & \\
\text{xp} & \\
\text{Rtn adr} & \\
\%ebp & \\
\%esp & \\
\end{align*}
\]

\[
\begin{align*}
\text{movl} & \quad -4(\%ebp),\%ebx \\
\text{movl} & \quad \%ebp,\%esp \\
\text{popl} & \quad \%ebp \\
\text{ret} &
\end{align*}
\]

Observation
- Saved & restored register \%ebx
- Didn't do so for \%eax, \%ecx, or \%edx
Disassembled swap

080483a4 <swap>:

080483a4:  55          push   %ebp
080483a5:  89 e5          mov    %esp,%ebp
080483a7:  53          push   %ebx
080483a8:  8b 55 08      mov    0x8(%ebp),%edx
080483ab:  8b 4d 0c      mov    0xc(%ebp),%ecx
080483ae:  8b 1a          mov    (%edx),%ebx
080483b0:  8b 01          mov    (%ecx),%eax
080483b2:  89 02          mov    %eax,(%edx)
080483b4:  89 19          mov    %ebx,(%ecx)
080483b6:  5b          pop    %ebx
080483b7:  c9          leave
080483b8:  c3          ret

Calling Code

08048409:  e8 96 ff ff ff          call 080483a4 <swap>

0x0804840e + 0xffffff96 = 0x080483a4

Register Saving Conventions

- When procedure you calls who:
  - you is the caller
  - who is the callee

- Can Register be used for temporary storage?

  you:
  
  • • •
  movl $15213, %edx
  call who
  addl %edx, %eax
  • • •
  ret

  who:

  • • •
  movl 8(%ebp), %edx
  addl $98195, %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

IA32/Linux Register Usage

- `%eax`, `%edx`, `%ecx`
  - Caller saves prior to call if values are used later

- `%eax`
  - also used to return integer value

- `%ebx`, `%esi`, `%edi`
  - Callee saves if wants to use them

- `%esp`, `%ebp`
  - special
Recursive Factorial

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

- **Registers**
  - `%ebx` used, but saved at beginning & restored at end
  - `%eax` used without first saving
    - expect caller to save
    - pushed onto stack as parameter for next call
    - used for return value

---

Pointer Code

**Recursive Procedure**

```c
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**

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

- Pass pointer to update location
Creating & Initializing Pointer

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

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

Initial part of `sfact`

```c
_sfact:
    pushl %ebp
    movl %esp,%ebp
    subl $16,%esp
    movl 8(%ebp),%edx
    movl $1,-4(%ebp)
```

- Variable `val` must be stored on stack
  - Because: Need to create pointer to it
- Compute pointer as \(-4\) (%ebp)
- Push on stack as second argument
Passing Pointer

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

Calling `s_helper` from `sfact`

```asm
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
```
IA 32 Procedure Summary

- 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
  - 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

Diagram:
- Caller Frame
- Arguments
- Return Addr
- Old %ebp
- Saved Registers + Local Variables
- Argument Build
- %ebp
- %esp