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
Stack Operation Examples

```
pushl %eax

%eax  213
%edx  555
%esp  0x108

0x108  123
0x10c  
0x110  

popl %edx

%eax  213
%edx  555
%esp  0x108

0x108  123
0x10c  
0x110  
```
Procedure Control Flow

Use stack to support procedure call and return

Procedure call:

\texttt{call label}  Push return address on stack; Jump to \texttt{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 = \texttt{0x8048553}

Procedure return:

- \texttt{ret}  Pop address from stack; Jump to address
Procedure Call / Return Example

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

call  8048b90

ret

%esp  0x108
%eip  0x804854e

%esp  0x104
%eip  0x8048b90

%esp  0x104
%eip  0x8048553

%esp  0x108
%eip  0x8048553

%eip  is program counter
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
Call Chain Example

Code Structure

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

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

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

- Procedure `amI` recursive

Call Chain

```
yoo
   ↓
   who
      ↓
      amI
         ↓
         amI
            ↓
            amI
```
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
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
Register Saving Conventions

When procedure `yoo` calls `who`:

- `yoo` is the caller, `who` is the callee

Can Register be Used for Temporary Storage?

- Contents of register `%edx` overwritten by `who`

Conventions

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

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

who:
  • • •
  movl 8(%ebp), %edx
  addl $91125, %edx
  • • •
  ret
```
IA32/Linux Register Usage

- Surmised by looking at code examples

### Integer 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

### Diagram

- **Caller-Save Temporaries**
  - `%eax`
  - `%edx`
  - `%ecx`
- **Callee-Save Temporaries**
  - `%ebx`
  - `%esi`
  - `%edi`
- **Special**
  - `%esp`
  - `%ebp`
**swap**

```c
int zip1 = 15213;
int zip2 = 91125;

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

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

**call_swap:**

```c
    ... ...
pushl $zip2    # Global Var
pushl $zip1    # Global Var
call swap
    ... ...
```

**Resulting Stack**

```
.. ..
&zip2
&zip1
Rtn adr
```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}

```assembly
swap:
    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**

\[
\text{\%ebp} \quad \text{\%esp}
\]

- &zip2
- &zip1
- Rtn adr

**Resulting Stack**

\[
\text{\%ebp} \quad \text{\%esp}
\]

- yp
- xp
- Rtn adr
- Old \%ebp

\[
\text{\%esp}
\]

---

**swap:**

- pushl \%ebp
- movl \%esp,\%ebp
- pushl \%ebx
swap Setup #2

Entering Stack

\[
\begin{array}{c}
\cdot \\
\cdot \\
\cdot \\
&\text{zip2} \\
&\text{zip1} \\
\text{Rtn adr} \\
\hline
\%ebp \\
\%esp
\end{array}
\]

Resulting Stack

\[
\begin{array}{c}
\cdot \\
\cdot \\
\cdot \\
\\text{yp} \\
\\text{xp} \\
\\text{Rtn adr} \\
\hline
\\%ebp \\
\%esp
\end{array}
\]

\[
\text{swap:} \\
\text{\hspace{1cm} pushl } \%ebp \\
\text{\hspace{1cm} movl } \%esp,\%ebp \\
\text{\hspace{1cm} pushl } \%ebx
\]

\[-13-\]
swap Setup #3

swap:

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

- Entering Stack:
  - %ebp
  - &zip2
  - &zip1
  - Rtn adr

- Resulting Stack:
  - yp
  - xp
  - Rtn adr
  - Old %ebp
  - Old %ebx
  - %ebp
  - %esp

Diagram illustrating the stack setup before and after executing the `swap` function.
Effect of swap Setup

Entering Stack

Resulting Stack

\[
\begin{align*}
\text{Offset} & \quad 12 & \quad \text{yp} \\
\text{Offset} & \quad 8 & \quad \text{xp} \\
\text{Offset} & \quad 4 & \quad \text{Rtn adr} \\
\text{Offset} & \quad 0 & \quad \text{Old %ebp} \\
\text{Offset} & \quad & \quad \text{Old %ebx}
\end{align*}
\]

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

**Observation**
- Saved & restored register %ebx

```
movl -4(%ebp), %ebx
movl %ebp, %esp
popl %ebp
ret
```
swap Finish #2

mos\text{-}l \ -4(\%ebp),\%ebx
mos\text{-}l \%ebp,\%esp
popl \%ebp
ret

\textbf{swap'\textquotesingle S Stack}

\begin{center}
\begin{tabular}{|c|c|}
\hline
Offset & Contents \\
\hline
12 & yp \\
8 & xp \\
4 & Rtn adr \\
0 & Old \%ebp \\
-4 & Old \%ebx \\
\hline
\end{tabular}
\end{center}

\textbf{swap'\textquotesingle S Stack}

\begin{center}
\begin{tabular}{|c|c|}
\hline
Offset & Contents \\
\hline
12 & yp \\
8 & xp \\
4 & Rtn adr \\
0 & Old \%ebp \\
\hline
\end{tabular}
\end{center}
swap Finish #3

swap's Stack

Offset
12  yp
8  xp
4  Rtn adr
0  Old %ebp

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

- Saved & restored register %ebx
- Didn’t do so for %eax, %ecx, or %edx

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