IA32 Stack

- Region of memory managed with stack discipline
- Grows toward lower addresses
- 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

Stack Pointer: \%esp

Stack “Top”

Stack Pointers: \%esp

Stack Grows Down

Increasing Addresses

Stack “Bottom”
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

%esp: 0x108
%eip: 0x804854e

0x8048b90:
pushl %eax

call 8048b90

%esp: 0x104
%eip: 0x8048b90

%esp: 0x110
%eip: 0x804854e
Procedure Return Example

8048591:  c3  ret

```
0x110 0x10c 0x108 0x104
0x110 0x10c 0x108 0x104
0x110 0x10c 0x108 0x104
0x110 0x10c 0x108 0x104
%esp  0x104  123  0x8048553
%esp  0x108  123  0x8048553
%esp  0x104  0x8048553
%esp  0x108  0x8048553
%eip  0x8048591
%eip  0x8048591
```

%eip: 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

Procedure `amI` is recursive
Stack Frames

• Contents
  – Local variables
  – Return information
  – Temporary space

• Management
  – Space allocated when enter procedure
    • “Set-up” code
  – Deallocated when return
    • “Finish” code
```c
yoo (...) {
  who();
  ...
}
```
Example

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

Stack

- yoo
- who
- amI
- %ebp
- %esp
Example

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

Stack

```
%ebp
%esp
```

```
yoo
who
amI
amI
amI
```
Example

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

Stack

```
%ebp
%esp
```

```
yoo
who
amI
amI
```
Example

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

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

Stack

```
%ebp
%esp
```

```
yoo
who
amI
amI
amI
```
Example

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

Stack

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

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

Stack

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

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

Stack

```
%ebp
%esp
```

Diagram:

- `yoo`
- `who`
- `amI`
- `amI`
IA32/Linux Stack Frame

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

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

int zip1 = 15213;
int zip2 = 91125;

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

Calling `swap` from `call_swap`

```c
void call_swap()
{
    swap(&zip1, &zip2);
    pushl $zip2    # Global Var
    pushl $zip1    # Global Var
    call swap
    ...
}
```

Resulting Stack

- &zip2
- &zip1
- Rtn adr
- %esp
Revisiting \textbf{swap}

\begin{verbatim}
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
\end{verbatim}

\textbf{swap}:
\begin{verbatim}
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 %esp,%ebp
    popl %ebp
    ret
\end{verbatim}

\begin{itemize}
    \item \textbf{Set Up}
    \item \textbf{Body}
    \item \textbf{Finish}
\end{itemize}
**swap Setup #1**

**Entering Stack**

- \%ebp
- \&zip2
- \&zip1
- Rtn adr

**Resulting Stack**

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

**swap:**

```assembly
pushl %ebp
movl %esp,%ebp
pushl %ebx
```
swap Setup #1

Entering Stack

swap:

```
pushl %ebp
movl %esp,%ebp
pushl %ebx
Rtn adr

&zip2
&zip1
Rtn adr

```
swap Setup #1

Entering Stack:

- %ebp
- &zip2
- &zipl
- Rtn adr

Resulting Stack:

- yp
- xp
- Rtn adr
- Old %ebp

swap:

pushl %ebp
movl %esp,%ebp
pushl %ebx
**swap Setup #1**

**Entering Stack**

```
%ebp
%esp
&zip2
&zip1
Rtn adr
```

```
pushl %ebp
movl %esp,%ebp
```

```
pushl %ebx
```

```
Rtn adr
Old %ebp
%ebp
%esp
yp
xp
```

```
swap:
pushl %ebp
```

```
movl %esp,%ebp
```

```
pushl %ebx
```
swap Setup #1

Entering Stack

Resulting Stack

Offset relative to %ebp

movl 12(%ebp),%ecx  # get yp
movl 8(%ebp),%edx  # get xp
...
swap Finish #1

swap's Stack

Resulting Stack

Observation: Saved and restored register %ebx

movl -4(%ebp),%ebx
movl %esp,%ebp
popl %ebp
ret
swap's Stack

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

swap Finish #2
swap Finish #2

swap’s Stack

Resulting Stack

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

swap's Stack

movl -4(%ebp),%ebx
movl %esp,%ebp
popl %ebp
ret
swap Finish #3

swap's Stack

Resulting Stack

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

swap’s Stack

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

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

080483a4 <swap>:
80483a4:  55          push   %ebp
80483a5:  89 e5       mov    %esp,%ebp
80483a7:  53          push   %ebx
80483a8:  8b 55 08    mov    0x8(%ebp),%edx
80483ab:  8b 4d 0c    mov    0xc(%ebp),%ecx
80483ae:  8b 1a       mov    (%edx),%ebx
80483b0:  8b 01       mov    (%ecx),%eax
80483b2:  89 02       mov    %eax(%edx)
80483b4:  89 19       mov    %ebx(%ecx)
80483b6:  5b          pop    %ebx
80483b7:  c9          leave
80483b8:  c3          ret

Calling Code

8048409:  e8 96 ff ff ff ff   call 80483a4 <swap>
804840e:  8b 45 f8         mov  0x8.ffffffff(%ebp),%eax
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`

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

who:
  ...
  movl 8(%ebp), %edx
  addl $91125, %edx
  ...
  ret
```
Register Saving Conventions

• When procedure \texttt{yoo} calls \texttt{who}:
  – \texttt{yoo} is the \texttt{caller}
  – \texttt{who} is the \texttt{callee}

• Can register be used for temporary storage?

• Conventions
  – \texttt{"Caller Save"}
    • Caller saves temporary in its frame before calling
  – \texttt{"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