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
Stack Operation Examples

- `pushl %eax`
- `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>`
  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**

---

### Procedure Return Example

8048591: `c3`  `ret`

- **Return**

---

%esp 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();
  ...
}
```

**Call Chain**

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

Procedure `amI` recursive

---

**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**

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

Frame Pointer `%ebp`

Stack Pointer `%esp`

Call Chain

```
yoo
  proc
    who
      amI
```

Stack "Top"
**Stack Operation**

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

**Call Chain**

- `who()`: 1
- `yoo()`: 2
- `who()`: 1

**Frame Pointer**

- `%ebp`

- `%esp`

**Stack Operation**

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

**Call Chain**

- `yoo()`: 1
- `who()`: 2
- `yoo()`: 1

**Frame Pointer**

- `%ebp`

- `%esp`

**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**

Calling `swap` from `call_swap`

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

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

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

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

Entering Stack

Resulting Stack

Offset (relative to %ebp)

%ebp

%.zip2 12 yp
%.zip1 8 xp
Rtn adr 4 Rtn adr

Old %ebp

%esp

0

Observation

- Saved & restored register %ebx

swap Finish #1

swap Finish #2

body

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

swap Finish #3

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

ret
**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**

**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

**Register Saving Conventions**

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

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

Swap Finish #4

Offset

12 yp
8 xp
4 Rtn adr

Exiting Stack
%esp

%ebp

%ebp

%ebp

%esp

%esp

%esp

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

who:
  * *
  movl 8(%ebp), %edx
  addl $91125, %edx
  * *
  ret
Recursive Factorial

Recursive Factorial

Registers

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

Rfact Body

Recursion

Registers

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

int rfact(int x)
{
    int rval;
    if (x <= 1)
        return 1;
    rval = rfact(x-1);
    return rval * x;
}
Return from Call

Rtn adr
Old %ebp
Old %ebx
x-1

imull %ebx,%eax

x

x

x

Rtn adr
Old %ebp
Old %ebx
x-1

%eax (x-1)!
%ebx x
Assume that \( rfact(x-1) \) returns \((x-1)\!\) in register %eax

Rtn Completion

%eax x!
%ebx Old %ebx

%eax x!
%ebx Old %ebx

%eax x!
%ebx Old %ebx

%eax x!
%ebx Old %ebx

Moving from Call

%eax x!
%ebx Old %ebx

%eax x!
%ebx Old %ebx

%eax x!
%ebx Old %ebx

%eax x!
%ebx Old %ebx

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);
    }
}
```

Pass pointer to update location

Top-Level Call

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

Creating & Initializing Pointer

Initial part of sfact

```c
    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

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

#### Calling s_helper from sfact

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

#### Using Pointer

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

### 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

---

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