# Section 5: Procedures & Stacks

- Stacks in memory and stack operations
- The stack used to keep track of procedure calls
- Return addresses and return values
- Stack-based languages
- The Linux stack frame
- Passing arguments on the stack
- Allocating local variables on the stack
- Register-saving conventions
- Procedures and stacks on x64 architecture

# Register Saving Conventions

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

- **Can a register be used for temporary storage?**

```
yoo:
    • • •
    movl $12345, %edx
    call who
    addl %edx, %eax
    • • •
    ret
```

```
who:
    • • •
    movl 8(%ebp), %edx
    addl $98195, %edx
    • • •
    ret
```

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

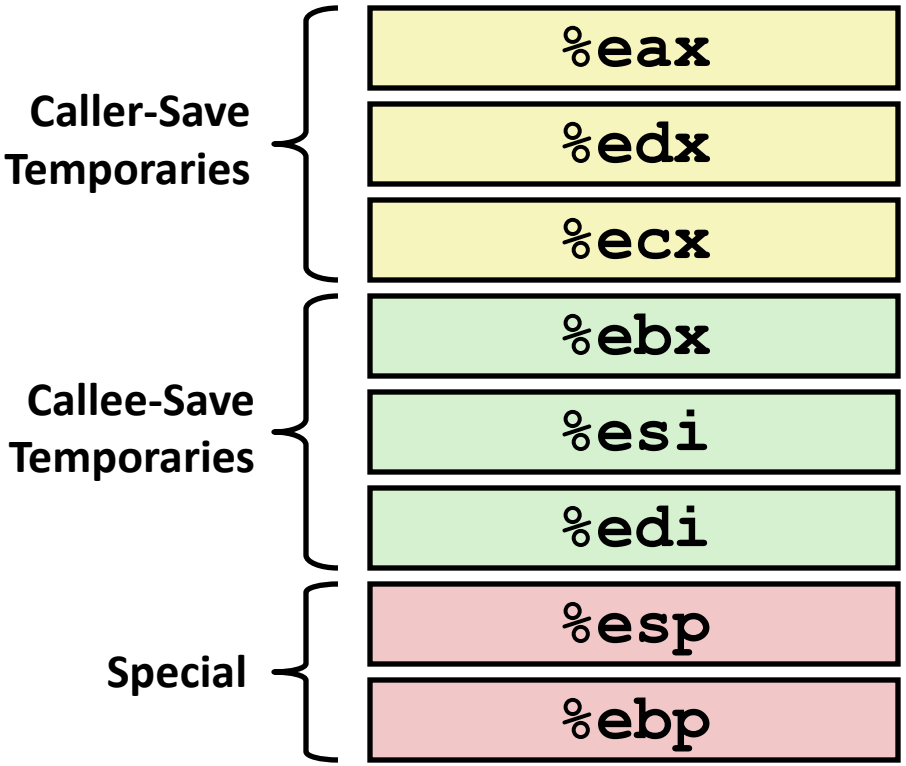
# Register Saving Conventions

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

- **Can a register be used for temporary storage?**
- **Conventions**
  - ***"Caller Save"***
    - Caller saves temporary values in its frame before calling
  - ***"Callee Save"***
    - Callee saves temporary values 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 form of callee save – restored to original values upon exit from procedure



# Example: Pointers to Local Variables

## Recursive Procedure

```
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

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

- Pass pointer to update location

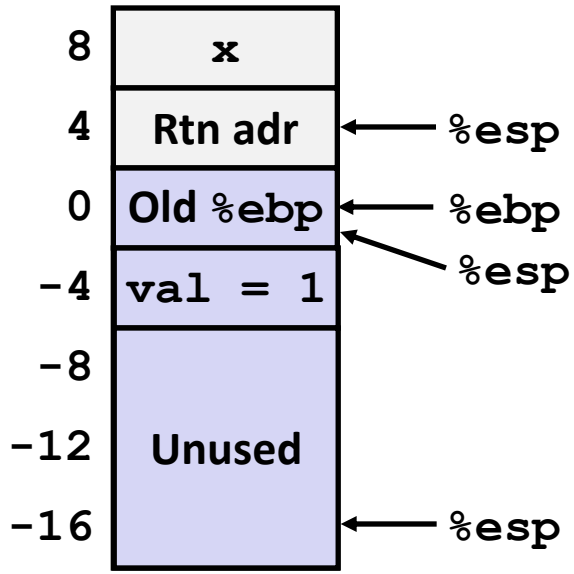
# Creating & Initializing Pointer

```
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`**

```
_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
```

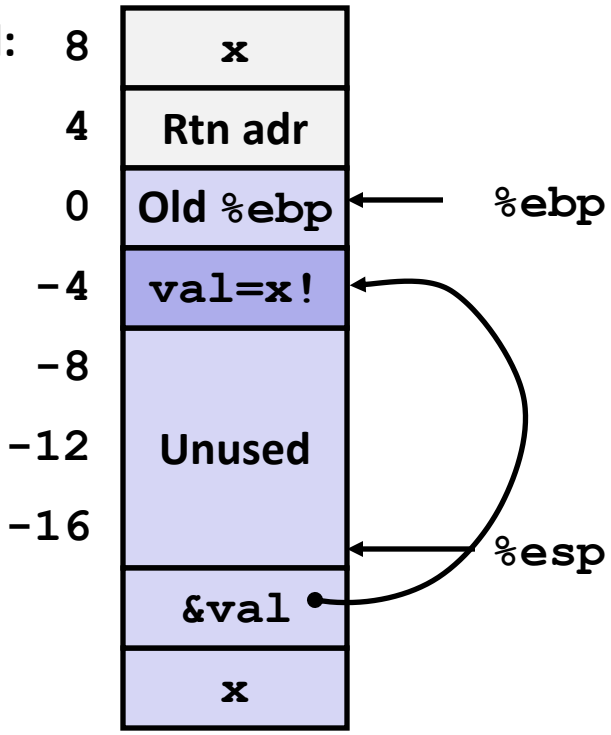


# Passing Pointer

```
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

Stack at time of call:




Calling `s_helper` from `sfact`

```
  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

- **Important points:**
  - IA32 procedures are a combination of *instructions* and *conventions*
    - Conventions prevent functions from disrupting each other
  - Stack is the right data structure for procedure call / return
    - If P calls Q, then Q returns before P
- **Recursion handled by normal calling conventions**
  - Can safely store values in local stack frame and in callee-saved registers
  - Put function arguments at top of stack
  - Result returned in **`%eax`**