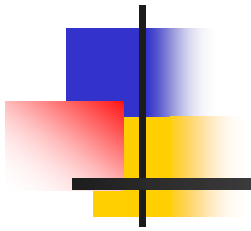


3.7 Summary

Alexandre David

Today

- Switch statements
- **IA 32 Procedures**
  - Stack Structure
  - Calling Conventions – Register usage 3.7.3
  - Illustrations of Recursion & Pointers

Register Saving Conventions

- **When procedure yoo calls who:**
  - **yoo** is the ***caller***
  - **who** is the ***callee***
- **Can register be used for temporary storage?**

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

```
who:
    • • •
    movl 8(%ebp), %edx
    addl $18243, %edx
    • • •
    ret
```

  - Contents of register **%edx** overwritten by **who**
  - This could be trouble → something should be done!
    - Need some coordination

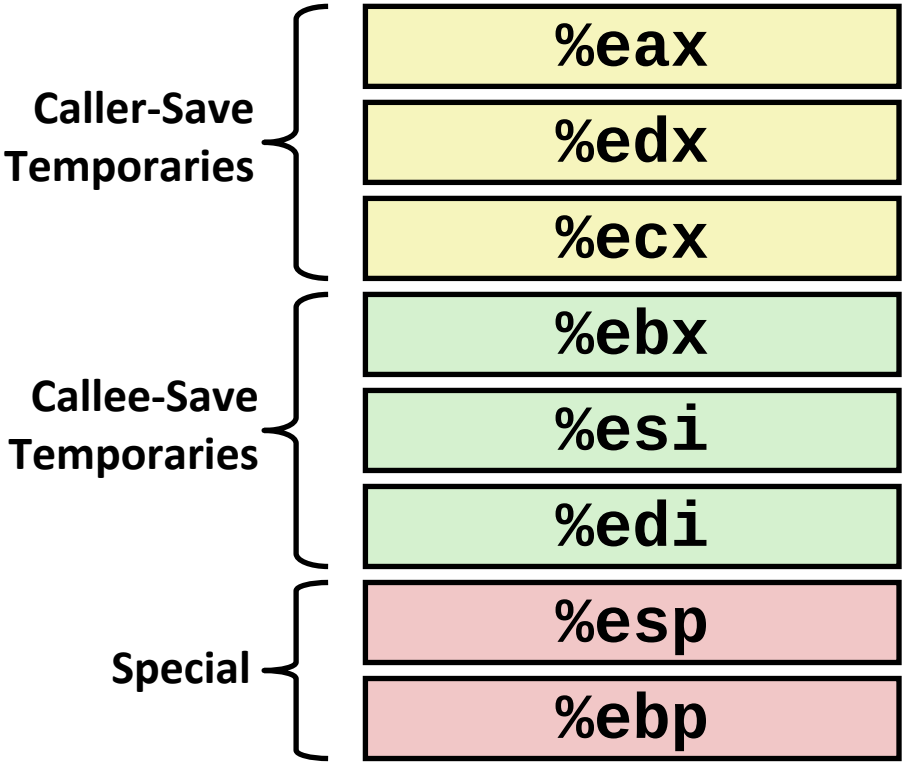
Register Saving Conventions

- **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 values in its frame before the call
  - ***"Callee Save"***
    - Callee saves temporary values in its frame before using

PP3.31

IA32/Linux+Windows 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



Today

- Switch statements
- **IA 32 Procedures**
  - Stack Structure
  - Calling Conventions
  - Illustrations of Recursion & Pointers – 3.7.5

Recursive Function

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

- **Registers**
  - `%eax`, `%edx` used without first saving
  - `%ebx` used, but saved at beginning & restored at end

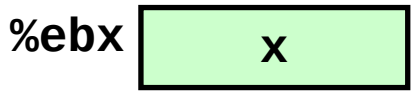
```
pcount_r:
    pushl %ebp
    movl  %esp, %ebp
    pushl %ebx
    subl  $4, %esp
    movl  8(%ebp), %ebx
    movl  $0, %eax
    testl %ebx, %ebx
    je  .L3
    movl  %ebx, %eax
    shrl  %eax
    movl  %eax, (%esp)
    call  pcount_r
    movl  %ebx, %edx
    andl  $1, %edx
    leal  (%edx,%eax), %eax
.L3:
    addl  $4, %esp
    popl  %ebx
    popl  %ebp
    ret
```

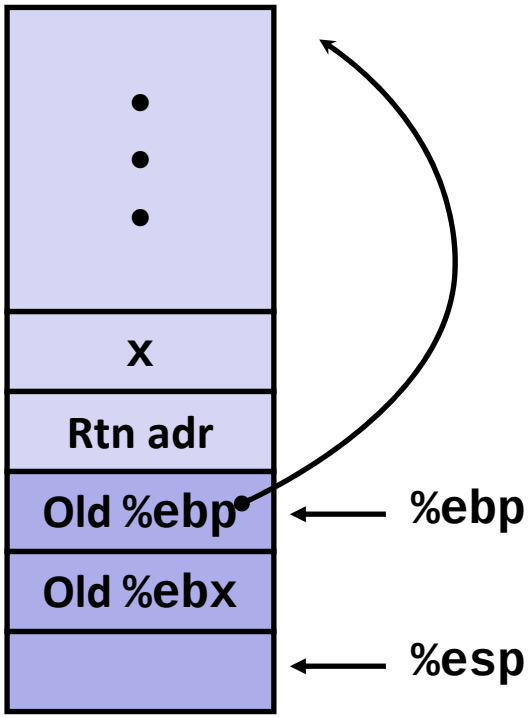
Recursive Call #1

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
pcount_r:
    pushl %ebp
    movl  %esp, %ebp
    pushl %ebx
    subl  $4, %esp
    movl  8(%ebp), %ebx
     . . .
```

- **Actions**
  - Save old value of **%ebx** on stack
  - Allocate space for argument to recursive call
  - Store x in **%ebx**



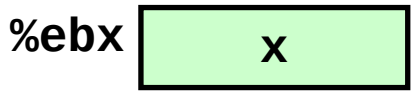


Recursive Call #2

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
    . . .
    movl  $0, %eax
    testl %ebx, %ebx
    je  .L3
    . . .
.L3:
    . . .
    ret
```

- **Actions**
  - If x == 0, return
    - with **%eax** set to 0

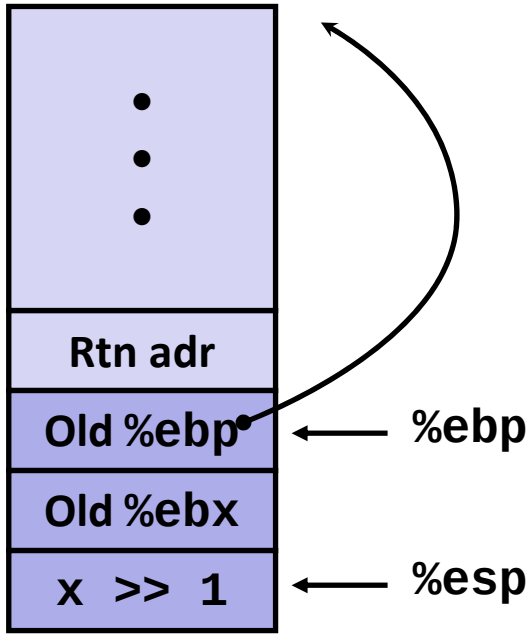


Recursive Call #3

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
  • • •
movl  %ebx, %eax
shrl  %eax
movl  %eax, (%esp)
call  pcount_r
  • • •
```

- **Actions**
  - Store x >> 1 on stack
  - Make recursive call
- **Effect**
  - %eax set to function result
  - %ebx still has value of x

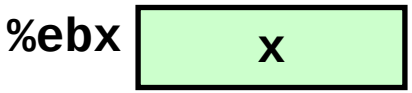


Recursive Call #4

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
  . . .
movl   %ebx, %edx
andl   $1, %edx
leal   (%edx,%eax), %eax
  . . .
```

- Assume
  - %eax holds value from recursive call
  - %ebx holds x
- Actions
  - Compute (x & 1) + computed value
- Effect
  - %eax set to function result

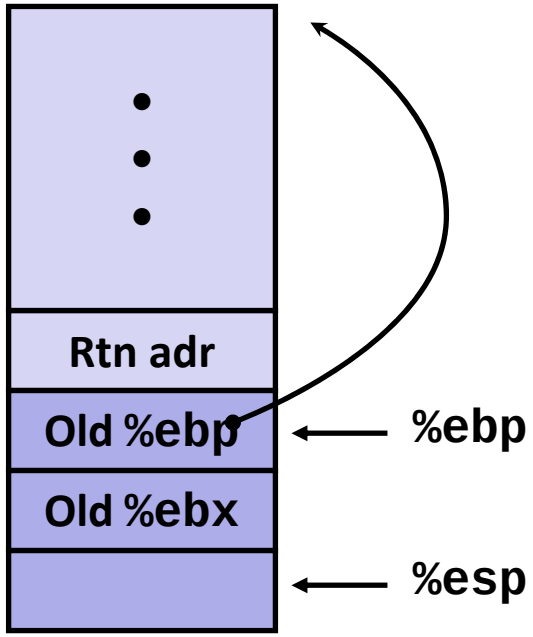


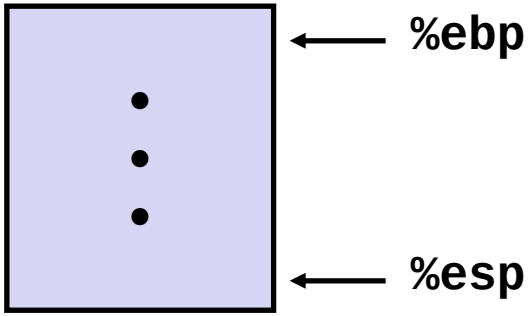
Recursive Call #5

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
    . . .
L3:
    addl$4, %esp
    popl%ebx
    popl%ebp
    ret
```

- Actions
  - Restore values of %ebx and %ebp
  - Restore %esp





%ebx

Old %ebx

Observations About Recursion

- **Handled Without Special Consideration**
  - Stack frames mean that each function call has private storage
    - Saved registers & local variables
    - Saved return pointer
  - Register saving conventions prevent one function call from corrupting another's data
  - Stack discipline follows call / return pattern
    - If P calls Q, then Q returns before P
    - Last-In, First-Out
- **Also works for mutual recursion**
  - P calls Q; Q calls P