Subprograms (Practice)

ICS312

Machine-Level and Systems Programming

Henri Casanova (henric@hawaii.edu)

(q1) Implement a function

- Implement this function in assembly:

```
def f(int x, int y):
  if x == 0:
    return 2*y
  else:
    return x + y
```

(q1) Solutions

- Implement this function in assembly:

```
def f(int x, int y):
  if x == 0:
    return 2*y
  else:
    return x + y
```

```
f:
  push ebp
  mov  ebp, esp
  cmp dword [ebp+8], 0
  jne else_f
  mov eax, [ebp+12]
  shl eax, 1
  jmp end_f
else_f:
  mov eax, [ebp+8]
  add eax, [ebp+12]
end_f:
  pop ebp
  ret
```

(q2) Implement a function

- Implement this function in assembly (do not destroy any register other than eax):

```
int f(int x, int *y) {
  *y += x;
  return *y;
}
```

(q2) Solutions

- Implement this function in assembly (do not destroy any register other than eax):

```
int f(int x, int *y) {
  *y += x;
  return *y;
}
```

```
f:
  push ebp
  mov  ebp, esp
  push ebx
  mov  eax, [ebp+12]
  mov  ebx, [ebp+8]
  add  [eax], ebx
  mov  eax, [eax]
  pop  ebx
  pop  ebp
  ret
```

(q3) Implement a function

- Implement this function in assembly (do not destroy any register other than eax):

```
int f(int *x, int *y) {
  return (*x == *y);
}
```

(q3) Solutions

- Implement this function in assembly (do not destroy any register other than eax):

```
int f(int *x, int *y) {
  return (*x == *y);
}
```

```
f:
  push ebp
  mov  ebp, esp
  push ebx
  mov  eax, [ebp+8]
  mov  eax, [eax]
  mov  ebx, [ebp+12]
  cmp  eax, [ebx]
  mov  eax, 1
  je   f_equal
  mov  eax, 0
f_equal:
  pop ebx
  pop ebp
  ret
```

(q4) Implement a function

- Implement this function in assembly (using the same local variable, and destroying no register):

```
int f(int *x, int y) {
  int z = 0;
  for (int i=0; i < y; i++) {
    z += *x
  }
  return z;
}
```

(q4) Solutions

- Implement this function in assembly (using the same local variable, and destroying no register):

```
int f(int *x, int y) {
  int z = 0;
  for (int i=0; i < y; i++) {
    z += *x
  }
  return z;
}
```

```
f:
  push ebp
  mov  ebp, esp
  sub  esp, 4
  push ecx

  mov  dword [ebp-4], 0
  mov  ecx, 0 ; loop idx
loop_begin:
  cmp  ecx, [ebp+12]
  jge  loop_end
  mov  eax, [ebp+8]
  mov  eax, [eax]
  add  [ebp-4], eax
  inc  ecx
  jmp  loop_begin
loop_end:
  mov  eax, [ebp-4]

  pop  ecx
  mov  esp, ebp
  pop  ebp
  ret
```

(q5) Debug Assembly

- What's wrong with this function, which is supposed to return the sum of two integers?

```
f:
  push ebp
  mov  ebp, esp
  pusha
  mov  ebx, [ebp+4]
  mov  ecx, [ebp+8]
  add  ebx, ecx
  mov  eax, ebx
  popa
  pop ebp
  ret
```

(q5) Solutions

- What's wrong with this function, which is supposed to return the sum of two integers?

```
f:
  push ebp
  mov  ebp, esp
  pusha
  mov  ebx, [ebp+4]
  mov  ecx, [ebp+8]
  add  ebx, ecx
  mov  eax, ebx
  popa
  pop ebp
  ret
```

Return address!! (should be [ebp+8], and next line should be [ebp+12])

Overwrites return value

(q6) Draw the Stack

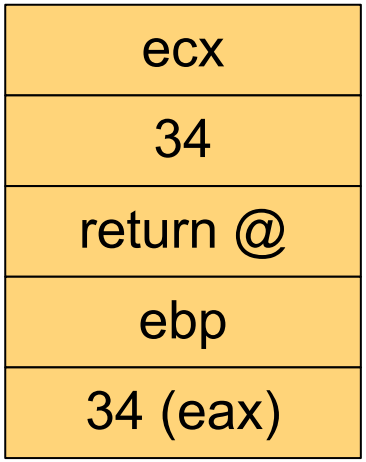
- Draw the stack at point HERE

```
push ecx
push dword 34
call f
add  esp, 8
. . .
func:
  push ebp
  mov  ebp, esp
  sub  esp, 4
  mov  eax, [ebp + 8]
  mov  [ebp - 4], eax
  ; HERE
```

(q6) Solutions

- Draw the stack at point HERE

```
push ecx
push dword 34
call f
add  esp, 8
. . .
func:
  push ebp
  mov  ebp, esp
  sub  esp, 4
  mov  eax, [ebp + 8]
  mov  [ebp - 4], eax
  ; HERE
```



(q7) Draw the Stack

- Draw the stack at its deepest

```
. . .
 f(12);
. . .
int f(int x){
  int z = 10;
  g(x, x+z);
}

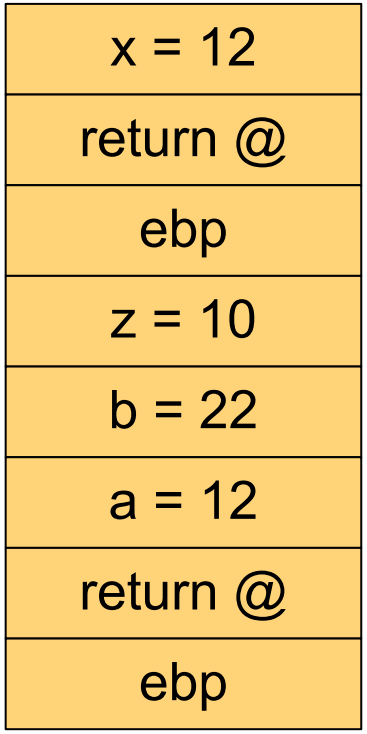
int g(int a, int b) {
  return a + 2 * b;
}
```

(q7) Solutions

- Draw the stack at its deepest

```
. . .
 f(12);
. . .
int f(int x){
  int z = 10;
  g(x, x+z);
}

int g(int a, int b) {
  return a + 2 * b;
}
```



(q8) Draw the Stack

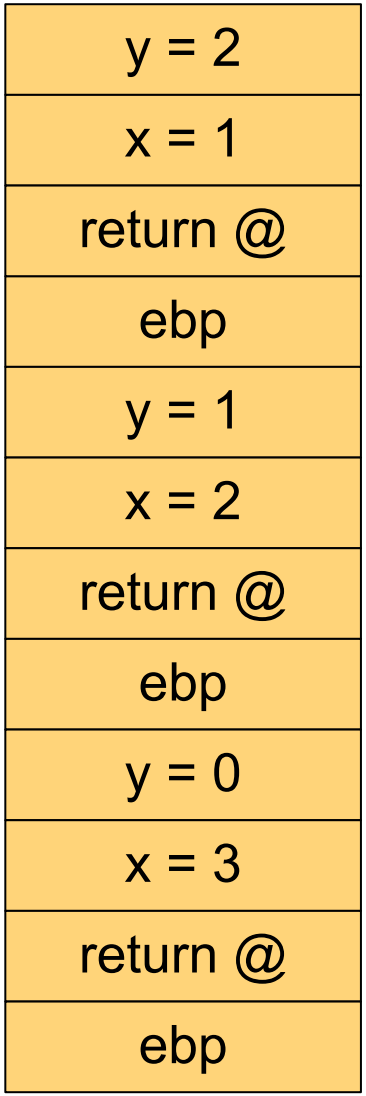
- Draw the stack at point HERE

```
. . .
 f(1, 2);
. . .
int f(int x, int y){
  if y == 0 {
    // HERE
    return x;
  } else {
    return f(x+1, y-1);
  }
}
```

(q8) Solutions

- Draw the stack at point HERE

```
. . .
 f(1, 2);
. . .
int f(int x, int y){
  if y == 0 {
    // HERE
    return x;
  } else {
    return f(x+1, y-1);
  }
}
```

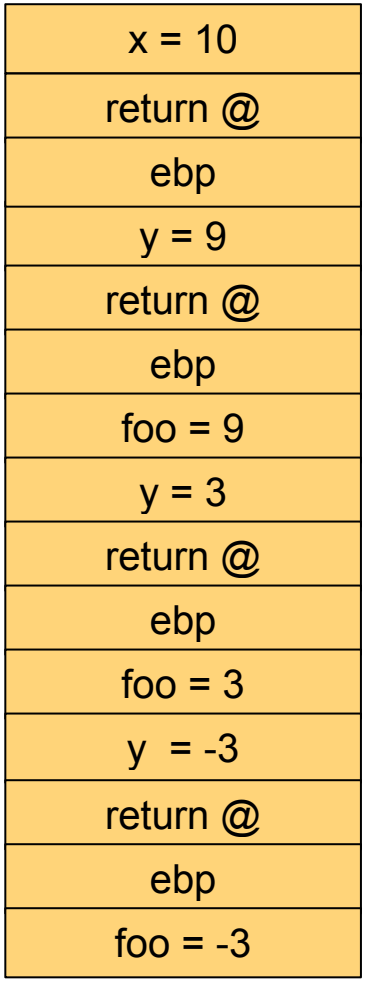


(q9) Draw the Stack

- Draw the stack at its deepest

```
. . .
 f(10);
. . .
int f(int x){
  return g(x-1);
}

int g(int y) {
  int foo = y;
  if (foo < 0) {
    return 66;
  } else {
    return g(foo - 6)
  }
}
```



(q10) How Deep?

- How many bytes have been added on the stack when its at its deepest (as a function of M)?

```
#define M . . .
. . .
f(M);
. . .
void f(int n) {
  int foo = 12;
  if (n > 0)
    f(n - foo);
  else
    return;
}
```

(q10) Solutions

- How many bytes have been added on the stack when its at its deepest (as a function of M)?

```
#define M . . .
. . .
f(M);
. . .
void f(int n) {
  int foo = 12;
  if (n > 0)
    f(n - foo);
  else
    return;
}
```

- Each call puts 16 bytes on the stack (parameter, return @, saved ebp, local variable foo)
- There are 1 + ⌈max(M, 0)/12⌉ calls to f
- Answer: 16 (1 + ⌈max(M, 0)/12⌉)