Machine-Level Representation of Programs: Loops, Procedures

Yipeng Huang

Rutgers University

November 10, 2026

Announcements

Class session plan

- ~~Today, 11/3: addressing modes (Book chapter 3.4), arithmetic (Book chapter 3.5). Bomblab phase_1.~~
- ~~Friday, 11/6: Control flow (conditionals, if, for, while, do loops, switch statements) in assembly. (Book chapter 3.6). Bomblab phase_2, phase_3.~~
- Tuesday, 11/10: Function calls in assembly. (Book chapter 3.7). Bomblab phase_4.
- Friday, 11/13: Arrays and data structures in assembly. (Book chapter 3.8). Bomblab phase_5, phase_6.

Quiz 6 and programming assignment 4

Quiz 6

1. Spanning Friday 11/6 - Friday 11/13.
2. 60 minutes.
3. Three tries.
4. Assembly data movement and control flow.

Programming assignment 4

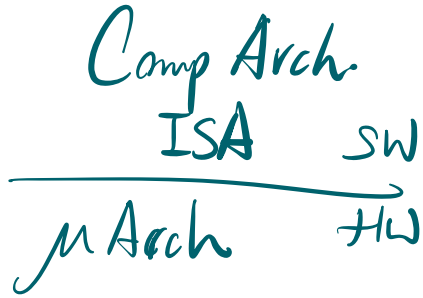
1. Due Tuesday 11/17.
2. Binary bomb lab.

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory



Branch statements

```c
unsigned long absdiff_ternary (
    unsigned long x, unsigned long y ){
        return x<y ? y-x : x-y;
}
```

```c
unsigned long absdiff_if_else (
    unsigned long x, unsigned long y ){
        if (x<y) return y-x;
        else return x-y;
}
```

```c
unsigned long absdiff_goto (
    unsigned long x, unsigned long y ){
        if (!(x<y)) goto Else;
        return y-x;
    Else:
        return x-y;
}
```

All C functions above translate (-fno-if-conversion) to assembly at right.

```
absdiff_if_else:
absdiff_goto:
    cmpq %rsi, %rdi
    jnb .ELSE
    movq %rsi, %rax
    subq %rdi, %rax
    ret
.ELSE:
    movq %rdi, %rax
    subq %rsi, %rax
    ret
```

Explanation

- `cmpq %rsi, %rdi`: Calculates %rdi − %rsi ($x - y$), sets condition codes.
- `jnb .ELSE`: Sets program counter / instruction pointer in %rip (.ELSE) if CF flag not set indicating no unsigned overflow.

Control Flow Abstractions: Syntactic Sugar

Observation: In C, an `if-else` statement, a ternary operator (`?  :`), and explicit `goto` statements can all compile to the *exact same assembly*.

At the machine level, all control flow is ultimately implemented as conditional jumps (goto). High-level structures are simply abstractions designed to make code readable and safe.

Historical Context: *Go To Considered Harmful*

In 1968, Edsger W. Dijkstra published a famous letter arguing that the `goto` statement should be abolished from high-level programming in favor of structured control flow (`if`, `while`). At the hardware level, however, `goto` is all the CPU understands!

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Conditional move statements

```c
unsigned long absdiff_ternary (
    unsigned long x, unsigned long y ){
        return x<y ? y-x : x-y;
}
```

```c
unsigned long absdiff_if_else (
    unsigned long x, unsigned long y ){
        if (x<y) return y-x;
        else return x-y;
}
```

```c
unsigned long absdiff_goto (
    unsigned long x, unsigned long y ){
        if (!(x<y)) goto Else;
        return y-x;
    Else:
        return x-y;
}
```

All C functions above translate (-fif-conversion or -O1) to assembly at

```
absdiff_ternary:
absdiff_if_else:
absdiff_goto:
    movq %rsi, %rdx // y
    subq %rdi, %rdx // y-x
    movq %rdi, %rax // x
    subq %rsi, %rax // x-y
    cmpq %rsi, %rdi
    cmovb %rdx, %rax
    ret
```

Explanation

- `cmpq %rsi, %rdi`: Calculates %rdi − %rsi ($x - y$), sets condition codes.
- `jnb .ELSE`: Sets program counter / instruction pointer in %rip (.ELSE) if CF flag not set indicating no unsigned overflow.

Modifying control flow vs. data flow in deep CPU pipelines

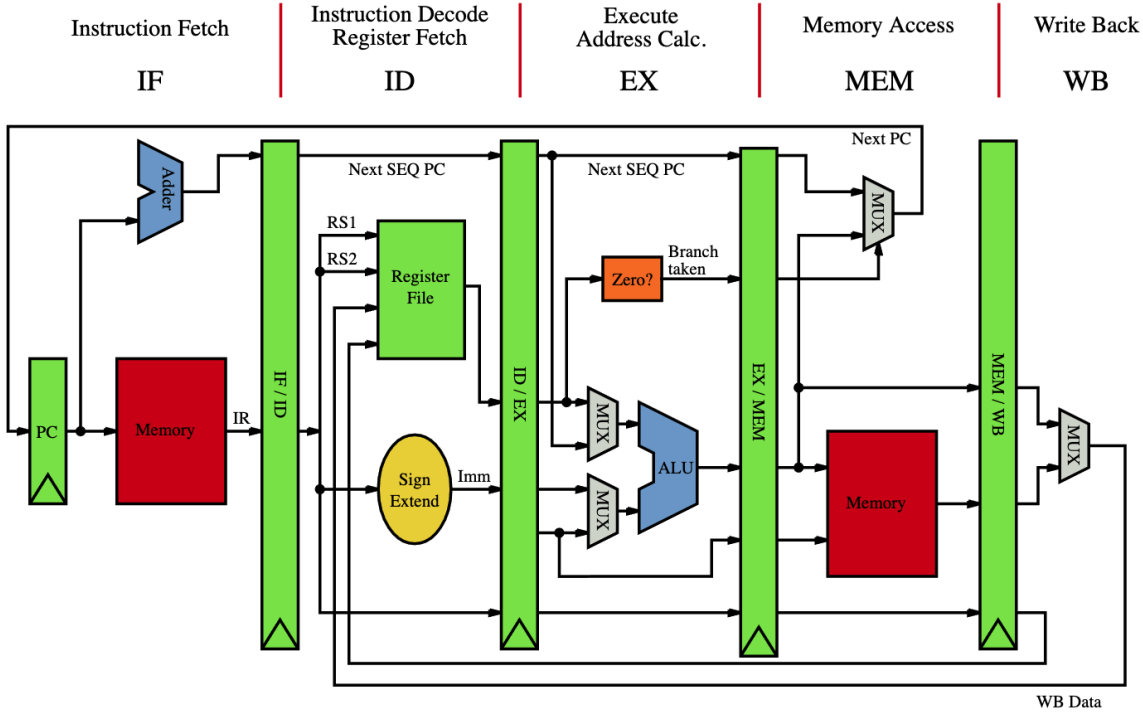


Figure: Pipelined CPU stages. Image credit wikimedia

P3 800 MHz 8×10^8 Hz

$\frac{1}{8} \times 10^{-8}$ s

$.125 \times 10^{-8}$ s

$= .125 \times 10^{-2}$ μs

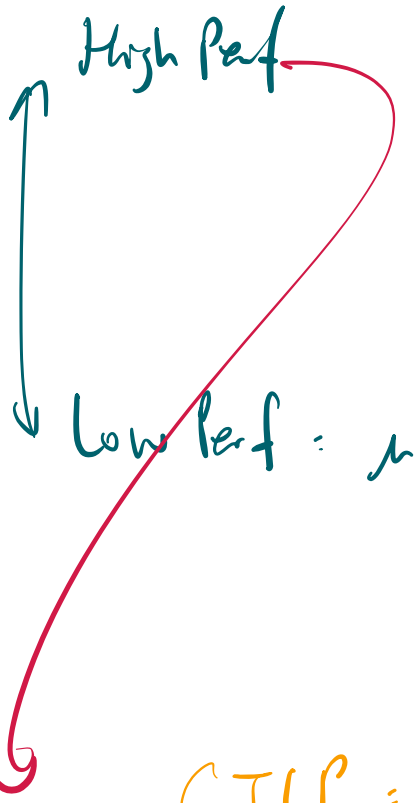
$= .125 \times 10^{1}$ ns

$= 1.25$ ns

P4 netburst. → 10 GHz

$\frac{1}{10}$ ns





Parallelism

- ILP : instruction level parallelism
- DLP : data level parallelism
- TLP : thread level parallelism

ILP :

- pipelining
- superscalar
- out of order exec.
- branch prediction

The Cost of Branching: Pipelines and Pentium 4

Why do compilers prefer Conditional Moves (`cmov`) over Jumps?

Modern CPUs use **pipelining** (fetching, decoding, and executing multiple instructions simultaneously). A conditional jump forces the CPU to guess the path (Branch Prediction).

- **Correct guess:** Zero performance penalty.
- **Incorrect guess:** The pipeline must be flushed, wasting dozens of cycles.

The Intel Netburst (Pentium 4) Lesson

Intel attempted to reach 10 GHz by using an ultra-deep pipeline (30+ stages). However, a single branch misprediction carried a catastrophic flush penalty, nullifying the clock speed gains on real-world code. This is why modern CPUs capped around 5 GHz and shifted focus to multi-core designs and branchless optimizations (`cmov`).

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Compiling for loops to while loops

C loop statements such as for loops, while loops, and do-while loops do not exist in assembly. They are instead constructed from conditional jump statements.

```
1  unsigned long count_bits_for (
2    unsigned long number
3  ) {
4    unsigned long tally = 0;
5    for (
6      int shift=0; // init
7      shift<8*sizeof(unsigned long); // ↩
           test
8      shift++ // update
9    ) {
10     // body
11     tally += 0b1 & number>>shift;
12   }
13   return tally;
14 }
```

```
1  unsigned long count_bits_while (
2    unsigned long number
3  ) {
4    unsigned long tally = 0;
5    int shift=0; // init
6    while (
7      shift<8*sizeof(unsigned long) // ↩
           test
8    ) {
9      // body
10     tally += 0b1 & number>>shift;
11     shift++; // update
12   }
13   return tally;
14 }
```

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Compiling while loops to do-while loops

```
unsigned long count_bits_while (
  unsigned long number
) {
  unsigned long tally = 0;
  int shift=0; // init
  while (
    shift<8*sizeof(unsigned long) // ↩
        test
  ) {
    // body
    tally += 0b1 & number>>shift;
    shift++; // update
  }
  return tally;
}
```

```
unsigned long count_bits_do_while (
  unsigned long number
) {
  unsigned long tally = 0;
  int shift=0; // init
  do {
    // body
    tally += 0b1 & number>>shift;
    shift++; // update
  } while (shift<8*sizeof(unsigned long↩
      )); // test
  return tally;
}
```

If initial iteration is guaranteed to run, then do one fewer test.

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Compiling do-while loops to goto statements

```c
unsigned long count_bits_do_while (
  unsigned long number
) {
  unsigned long tally = 0;
  int shift=0; // init
  do {
    // body
    tally += 0b1 & number>>shift;
    shift++; // update
  } while (shift<8*sizeof(unsigned long↩
      )); // test
  return tally;
}
```

```c
unsigned long count_bits_goto (
  unsigned long number
) {
  unsigned long tally = 0;
  int shift=0; // init
LOOP:
  // body
  tally += 0b1 & number>>shift;
  shift++; // update
  if (shift<8*sizeof(unsigned long)) { ↩
      // test
    goto LOOP;
  }
  return tally;
}
```

Loops get compiled into goto statements which are readily translated to assembly.

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Compiling goto statements to assembly conditional jump instructions

```
1  unsigned long count_bits_goto (
2    unsigned long number
3  ) {
4    unsigned long tally = 0;
5    int shift=0; // init
6  LOOP:
7    // body
8    tally += 0b1 & number>>shift;
9    shift++; // update
10   if (shift<8*sizeof(unsigned long)) { ↩
         // test
11     goto LOOP;
12   }
13   return tally;
14 }
```

All C loop statements so far translate to assembly at right.

```
count_bits_for:
count_bits_while:
count_bits_do_while:
count_bits_goto:
  xorl %ecx, %ecx # int shift=0; // init
  xorl %eax, %eax # unsigned long tally = 0;
.LOOP:
  movq %rdi, %rdx # number
  shrq %cl, %rdx  # number>>shift
  incl %ecx       # shift++; // update
  andl $1, %edx.  # 0b1 & number>>shift
  addq %rdx, %rax # tally += 0b1 & number>>shi
  cmpl $64, %ecx  # shift<8*sizeof(unsigned lo
  jne .LOOP       # goto LOOP;
  ret             # return tally;
```

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Procedures and function calls

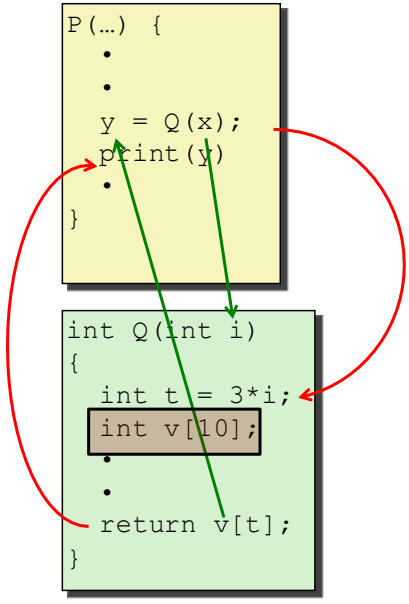


Figure: Steps of a C function call. Image credit CS:APP

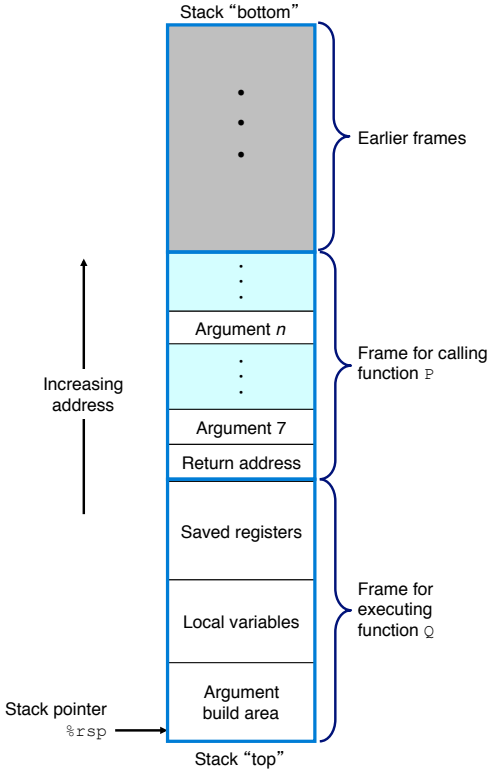
To create the abstraction of functions, need to:

- Transfer control to function and back
- Transfer data to function (parameters)
- transfer data from function (return type)

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Memory stack frames



Structure of stack for currently executing function Q()

- P() calls Q(). P() is the caller function. Q() is the callee function.

Stack instructions: `push src` and `pop dest`

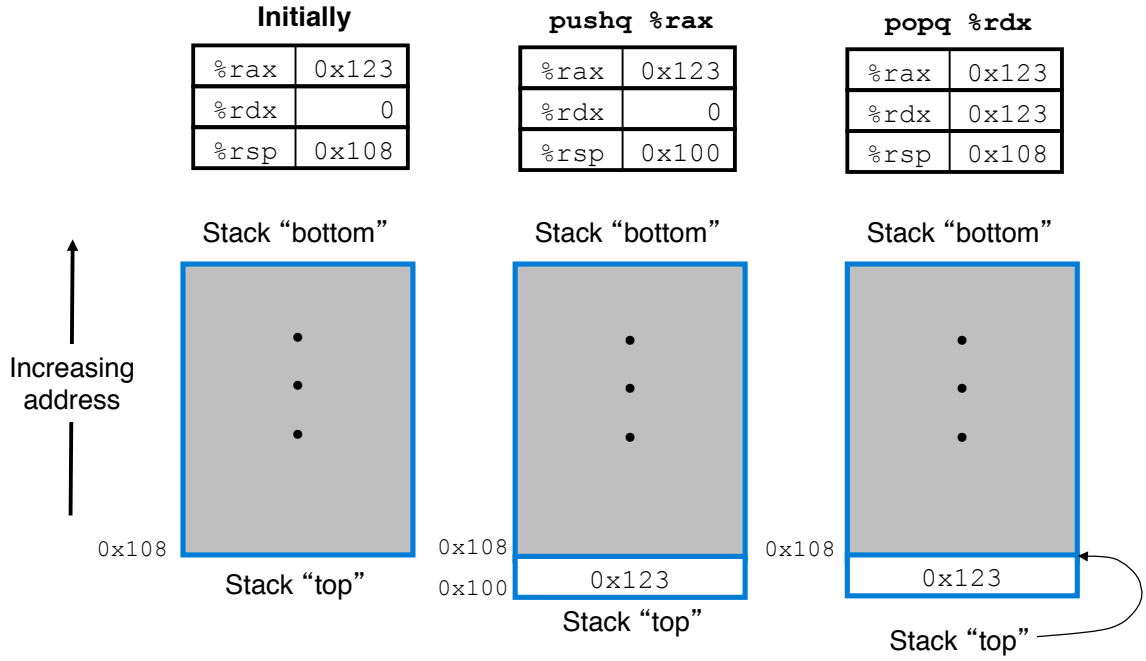


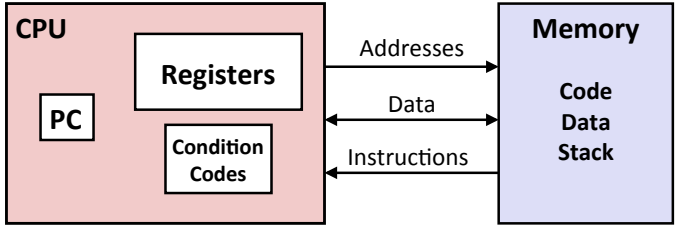
Figure: x86-64 offers dedicated instructions to work with stack in memory. In addition to moving data, the updating of %rsp is implied. Image credit: CS:APP.

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

CPU and memory state in support of procedures and functions

Assembly/Machine Code View



Programmer-Visible State

- **PC: Program counter**
  - Address of next instruction
  - Called "RIP" (x86-64)
- **Register file**
  - Heavily used program data
- **Condition codes**
  - Store status information about most recent arithmetic or logical operation
  - Used for conditional branching

- **Memory**
  - Byte addressable array
  - Code and user data
  - Stack to support procedures

Relevant state in CPU:

- %rip register / instruction pointer / program counter
- %rsp register / stack pointer

Relevant state in Memory:

- Stack

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Procedure call and return: `call` and `ret`

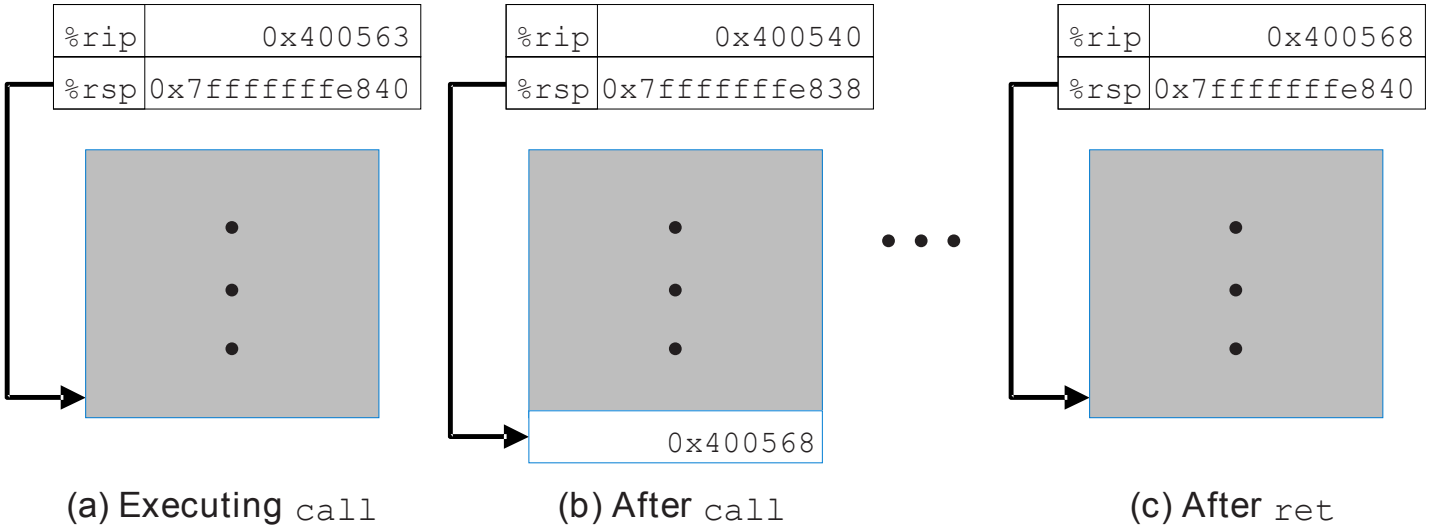


Figure: Effect of `call 0x400540` instruction and subsequent return. `call` and `ret` instructions update the instruction pointer, the stack pointer, and the stack to create the procedure / function call abstraction. Image credit: CS:APP.

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Example in GDB

```c
#include <stdio.h>

int return_neg_one() {
  return -1;
}

int main() {
  int num = return_neg_one();
  printf("%d", num);
  return 0;
}
```

```
return_neg_one:
    movl $-1, %eax
    ret
main:
    subq $8, %rsp
    movl $0, %eax
    call return_neg_one
    movl %eax, %edx
    ...
```

Compile, and then run it in GDB:

`gdb return`

In GDB, see evolution of %rip, %rsp, and stack:

- `(gdb) layout split`
- `(gdb) break return_neg_one`
- `(gdb) info stack`
- `(gdb) print /a $rip`
- `(gdb) print /a $rsp`
- `(gdb) x /a $rsp`

Step past return instruction, and inspect again:

- `(gdb) stepi`
- `(gdb) info stack`

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Procedures and function calls: Transferring data

For purposes of this class, the Bomb Lab, and the CS:APP textbook, we study the x86-64 Linux Application Binary Interface (ABI). Would be different on ARM or in Windows. So, don't memorize this, but it is helpful for PA4 Lab.

Passing parameters

Parameter	Register / stack	Subset registers	Mnemonic[1]
1st	%rdi	%edi, %di	Diane's
2nd	%rsi	%esi, %si	silk
3rd	%rdx	%edx, %dx, %dl	dress
4th	%rcx	%ecx, %cx, %cl	cost
5th	%r8	%r8d	$8
6th	%r9	%r9d	9
7th and beyond	Stack		

[1] `http://csappbook.blogspot.com/2015/08/dianes-silk-dress-costs-89.html`

PA4 Defusing a Binary Bomb: `sscanf();`

```
1 int sscanf (
2     const char *str, // 1st arg, %rdi
3     const char *format, // 2nd arg, %rsi
4     ...
5 )
```

Procedures and function calls: Transferring data

Passing function return data

Function return data is passed via:

- the 64-bit %rax register
- the 32-bit subset %eax register

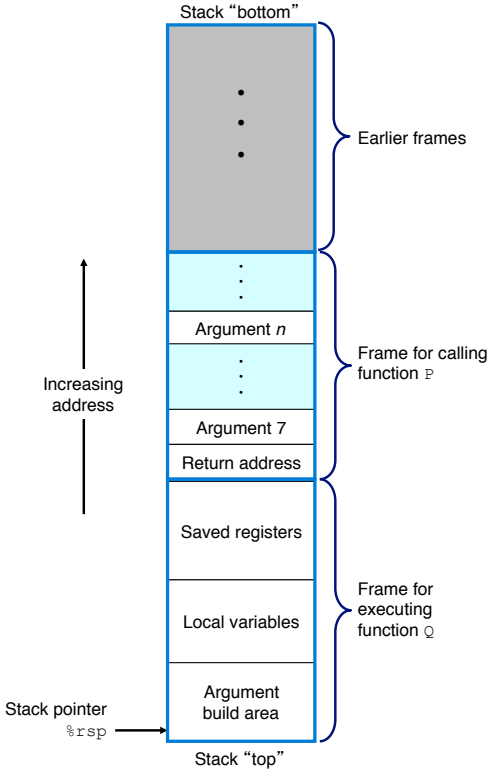
Example from textbook slides on assembly procedures

Slides 33 through 38.

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

Data transferred via memory



Structure of stack for currently executing function Q()

- P() calls Q(). P() is the caller function. Q() is the callee function.

Example from textbook slides on assembly procedures

Slides 40 through 44.

Machine-Level Representation of Programs: Loops, Procedures

- Modifying control flow via conditional branch statements
  - Conditional branch statements
- Modifying data flow via conditional move statements
- Loop statements
  - Compiling for loops to while loops
  - Compiling while loops to do-while loops
  - Compiling do-while loops to goto statements
  - Compiling goto statements to assembly conditional jump instructions
- Procedures and function calls
  - Memory stack frames
- Procedures and function calls: Transferring control
  - Procedure call and return: `call` and `ret`
  - Example in GDB
- Procedures and function calls: Transferring data
  - Data transferred via registers
  - Data transferred via memory

`3_recursion.c`: Putting it all together to support recursion

Discussion points

- Use info stack, info args in GDB to see recursion depth
- Difference between compiling with and without -g for debugging information.
- Memory costs of recursion.
- Compilers can recognize tail recursive calls to reduce memory use. Enabled with -foptimize-sibling-calls, -O2, -O3, and -Os.