Machine-Level Representation of Programs: Data and Locality

Yipeng Huang

Rutgers University

November 11, 2025

Table of contents

Announcements

Class session plan

- ~~Thursday, 10/30: addressing modes (Book chapter 3.4), arithmetic (Book chapter 3.5). Bomblab phase_1.~~
- ~~Tuesday, 11/4: Control flow (conditionals, if, for, while, do loops, switch statements) in assembly. (Book chapter 3.6). Bomblab phase_2, phase_3.~~
- ~~Thursday, 11/6: Function calls in assembly. (Book chapter 3.7). Bomblab phase_4.~~
- Tuesday, 11/11: Arrays and data structures in assembly. (Book chapter 3.8). Bomblab phase_5, phase_6.

Table of contents

Procedures and function calls

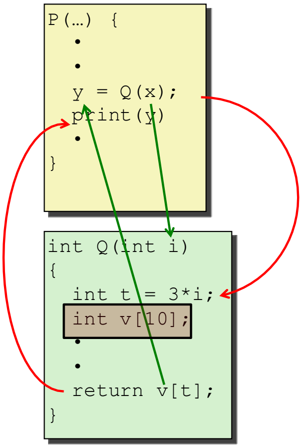
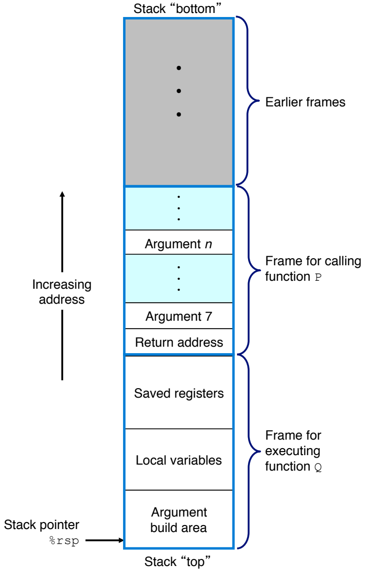


Figure: Steps of a C function call. Image credit CS:APP

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)

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`

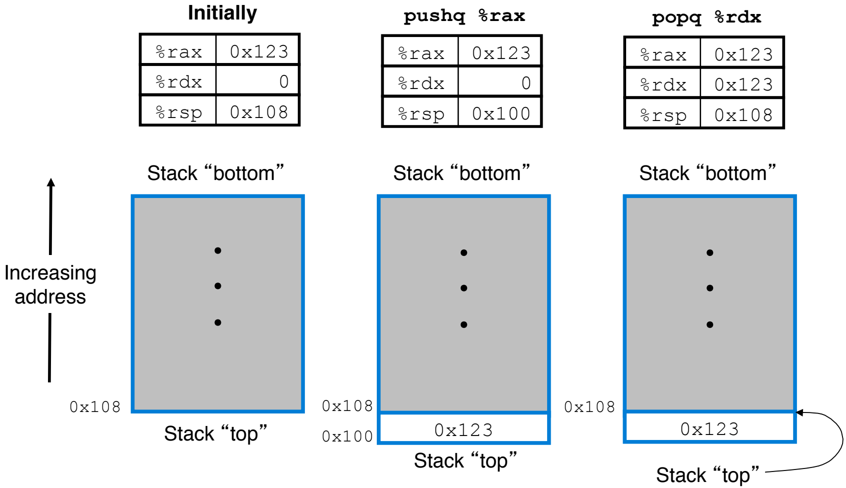
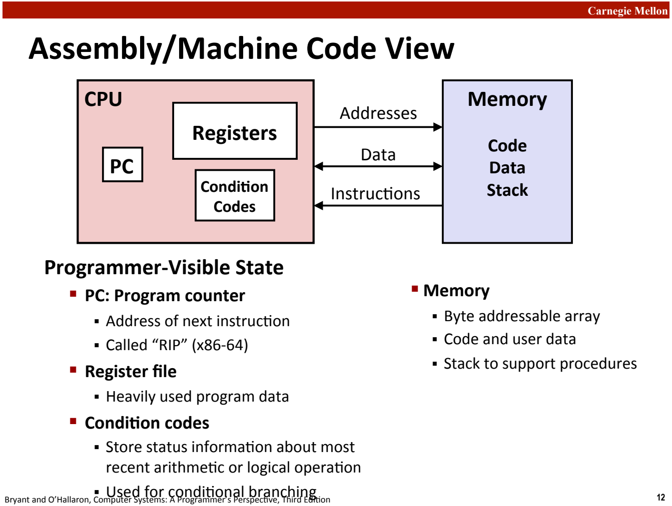


Figure: x86-64 offers dedicated instructions to work with stack in memory. In addition to moving data, the updating of %rsp is implied. Image credit: CS:APP.

Table of contents

CPU and memory state in support of procedures and functions



Relevant state in CPU:

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

Relevant state in Memory:

- Stack

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

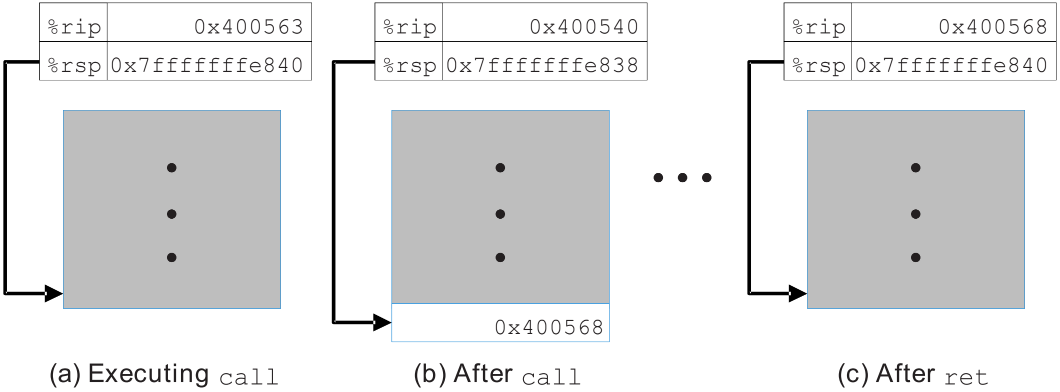


Figure: Effect of `call 0x400540` instruction and subsequent return. `call` and `ret` instructions update the instruction pointer, the stack pointer, and the stack to create the procedure / function call abstraction. Image credit: CS:APP.

Example in GDB

```
#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`

Table of contents

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();`

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

Procedures and function calls: Transferring data

Passing function return data

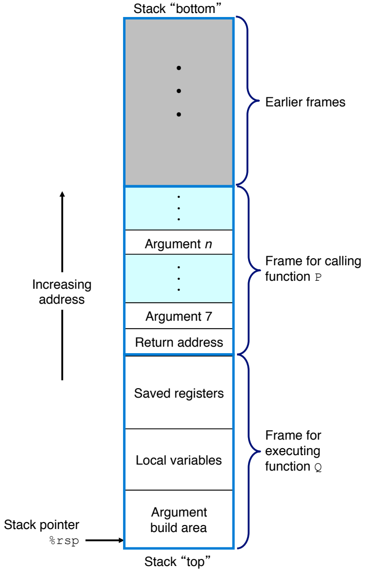
Function return data is passed via:

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

Example from textbook slides on assembly procedures

Slides 33 through 38.

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.

Table of contents

`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.

Table of contents

Cache, memory, storage, and network hierarchy trends

- Assembly programming view of computer: CPU and memory.
- Full view of computer architecture and systems: +caches, +storage, +network

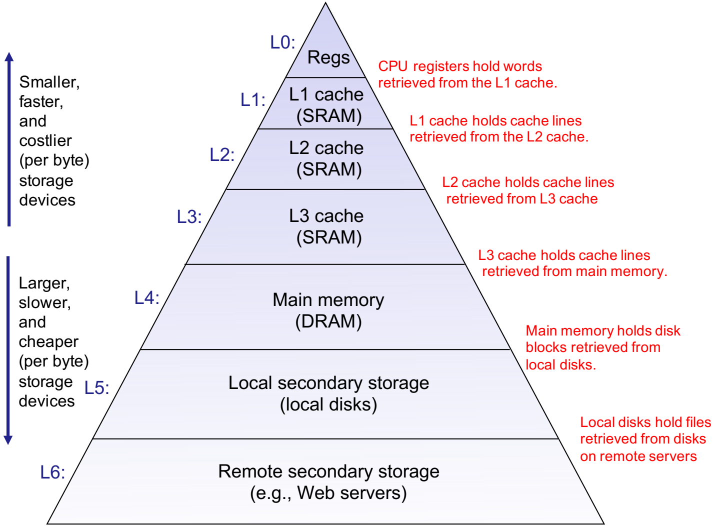


Figure: Memory hierarchy. Image credit CS:APP

Cache, memory, storage, and network hierarchy trends

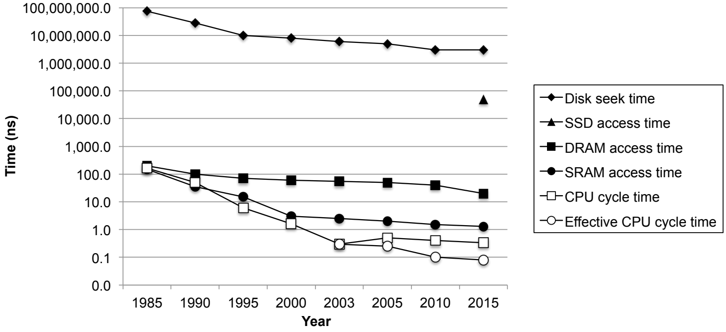


Figure: Widening gap: CPU processing time vs. memory access time. Image credit CS:APP

Topic of this chapter:

- Technology trends that drive CPU-memory gap.
- How to create illusion of fast access to capacious data.

Static random-access memory (registers, caches)

- SRAM is bistable logic
- Access time: 1 to 10 CPU clock cycles
- Implemented in the same transistor technology as CPUs, so improvement has matched pace.

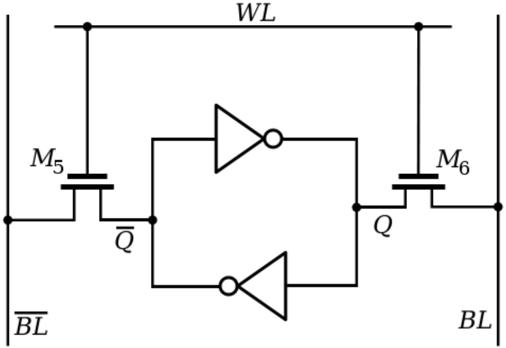


Figure: SRAM operating principle. Image credit Wikimedia

Dynamic random-access memory (main memory)

- Needs refreshing every 10s of milliseconds
- 8GB typical in laptop; 1TB on each ilab machine
- Access time: 100 CPU clock cycles
- Memory gap: DRAM technological improvement slower relative to CPU/SRAM.

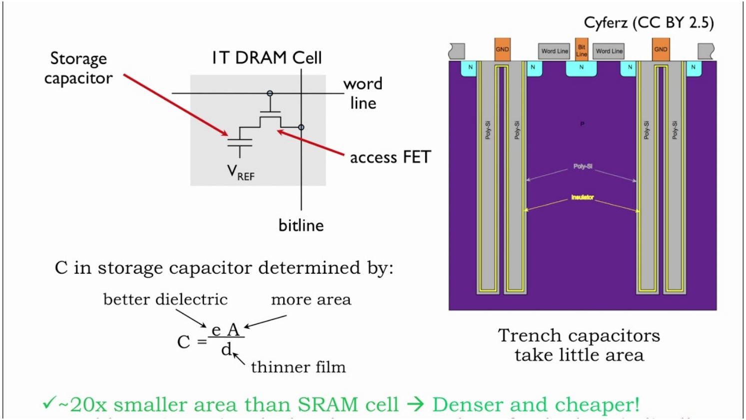


Figure: DRAM operating principle. Image credit ocw.mit.edu

CPU / DRAM main memory interface

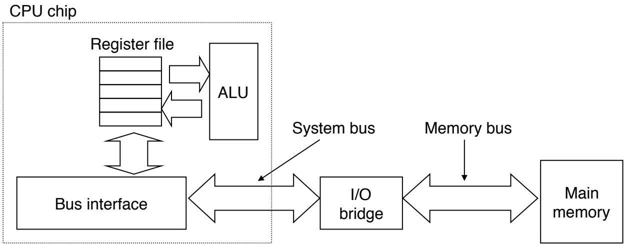


Figure: Memory Bus. Image credit CS:APP

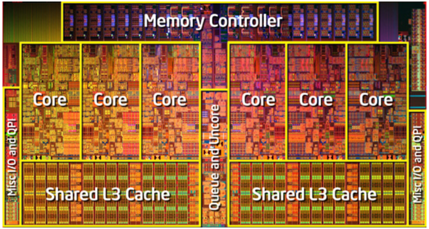


Figure: Intel 2020 Gulftown die shot. Image credit AnandTech

- DDR4 bus standard supports 25.6GB/s transfer rate

Solid state and hard disk drives (storage)

Technology

- SSD: flash nonvolatile memory stores data as charge.
- HDD: magnetic orientation.
- Access time: 100K CPU clock cycles

For in-depth on storage, file systems, and operating systems, take:

- CS214 Systems Programming
- CS416 Operating Systems Design

Since summer 2021, LCSR (admins of iLab) have moved the storage systems that supports everyone's home directories to SSD. `https://resources.cs.rutgers.edu/docs/file-storage/storage-technology-options/`

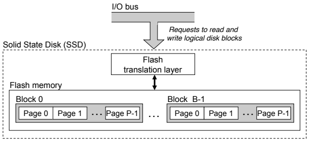


Figure: SSD. Image credit CS:APP

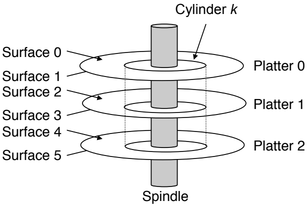


Figure: HDD. Image credit CS:APP

I/O interfaces

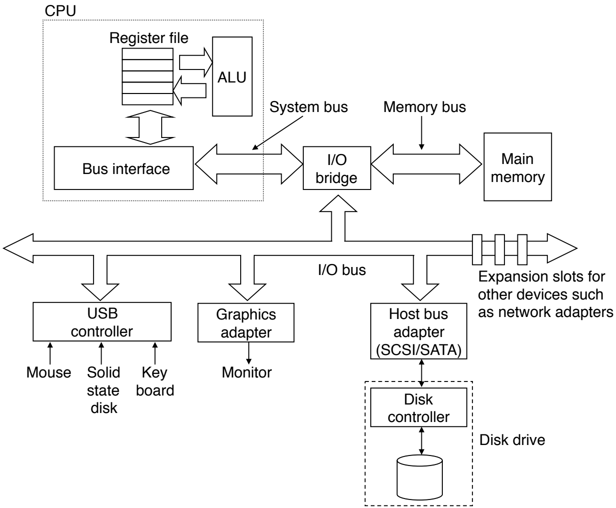


Figure: I/O Bus. Image credit CS:APP

Storage interfaces

- SATA 3.0 interface (6Gb/s transfer rate) typical
- PCIe (15.8 GB/s) becoming commonplace for SSD
- But interface data rate is rarely the bottleneck.

For in-depth on computer network layers, take:

- CS352 Internet Technology

Cache, memory, storage, and network hierarchy trends

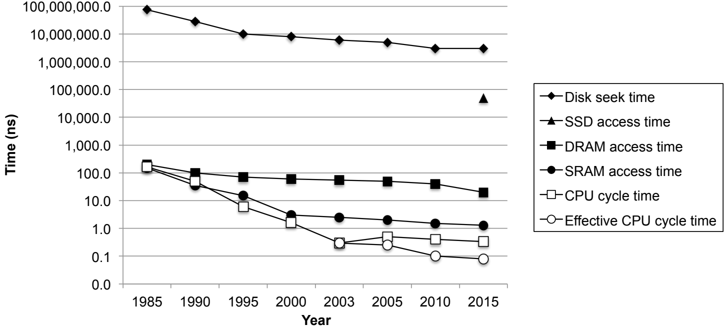


Figure: Widening gap: CPU processing time vs. memory access time. Image credit CS:APP

Topic of this chapter:

- Technology trends that drive CPU-memory gap.
- How to create illusion of fast access to capacious data.

Table of contents

Locality: How to create illusion of fast access to capacious data

From the perspective of memory hierarchy, locality is using the data in at any particular level more frequently than accessing storage at next slower level.

First, let's experience the puzzling effect of locality in `sumArray.c`

- sumArrayRows()
- sumArrayCols()

Well-written programs maximize locality

- Spatial locality
- Temporal locality

Spatial locality

```c
double dotProduct (
  double a[N],
  double b[N],
) {
  double sum = 0.0;
  for(size_t i=0; i<N; i++){
    sum += a[i] * b[i];
  }
  return sum;
}
```

Spatial locality

- Programs tend to access adjacent data.
- Example: stride 1 memory access in `a` and `b`.

Temporal locality

```
double dotProduct (
  double a[N],
  double b[N],
) {
  double sum = 0.0;
  for(size_t i=0; i<N; i++){
    sum += a[i] * b[i];
  }
  return sum;
}
```

Temporal locality

- Programs tend to access data over and over.
- Example: `sum` gets accessed N times in iteration.

Table of contents

CPU / cache / DRAM main memory interface

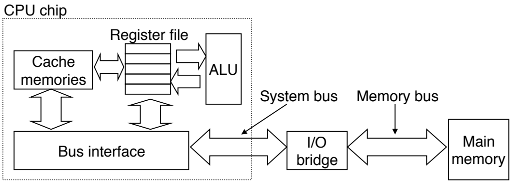
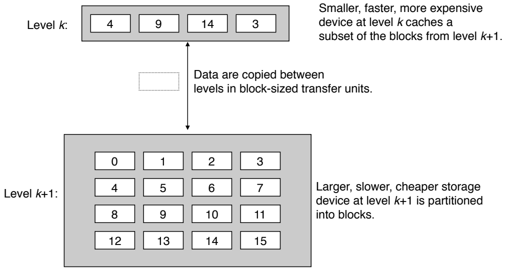


Figure: Cache resides on CPU chip close to register file. Image credit CS:APP



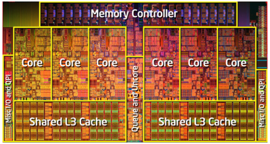


Figure: Intel 2020 Gulftown die shot. Image credit AnandTech

Figure: Cache stores a temporary copy from the slower main memory. Image credit CS:APP

CPU / cache / DRAM main memory interactions

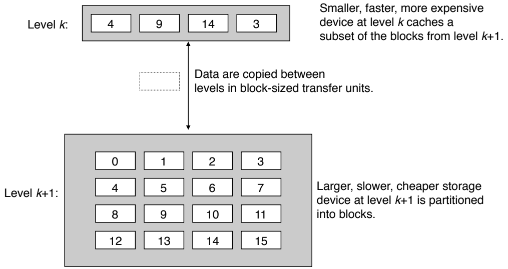


Figure: Cache stores a temporary copy from the slower main memory. Image credit CS:APP

When CPU loads (LD) from memory

- Cache read hit
- Cache read miss

When CPU stores (ST) to memory

- Cache write hit
- Cache write miss