High-Performance Scientific Computing
Lecture 8: Single-thread Performance

MATH-GA 2011 / CSCI-GA 2945 · October 24, 2012

Today

Tool of the day: Installing software

Closer to the machine

Making things go faster

Bits and pieces

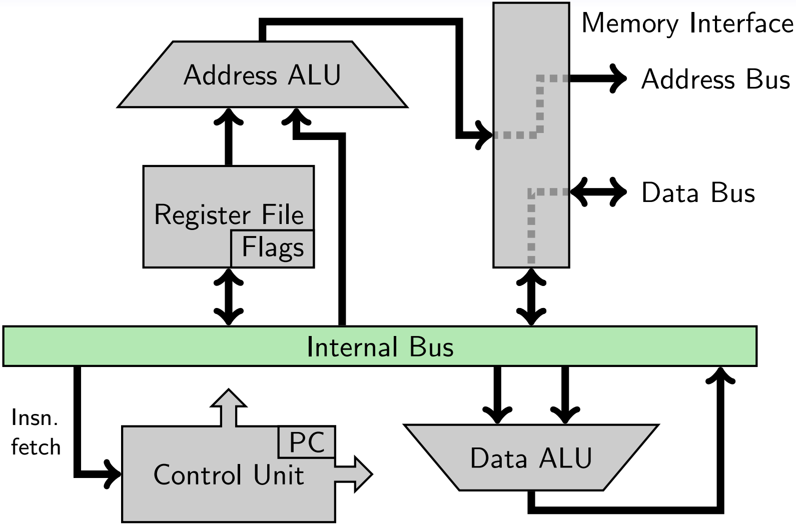
- HW4: tonight / early tomorrow
- HW6: due Saturday (ask for ext'n early)
- Last homework $\rightarrow$ project work after that
- Might issue problem sets for entertainment

Outline

Demo time

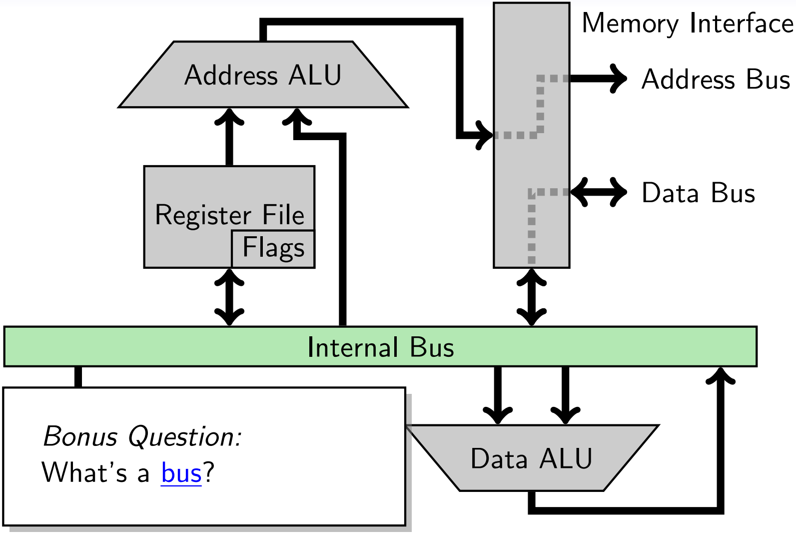
Outline

A Basic Processor



(loosely based on Intel 8086)

A Basic Processor



Bonus Question:
What's a bus?

(loosely based on Intel 8086)

Outline

A Very Simple Program

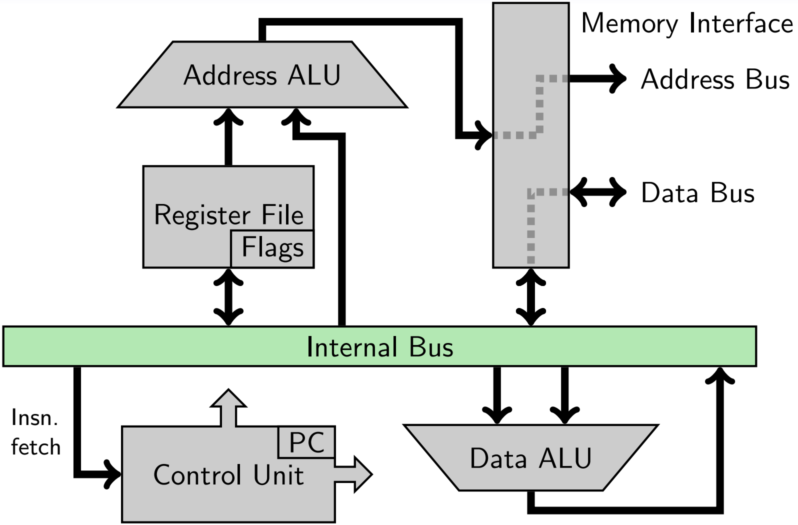
```
int a = 5;
int b = 17;
int z = a * b;
```

4:	c7 45 f4 05 00 00 00	movl	$0x5,−0xc(%rbp)
b:	c7 45 f8 11 00 00 00	movl	$0x11,−0x8(%rbp)
12:	8b 45 f4	mov	−0xc(%rbp),%eax
15:	0f af 45 f8	imul	−0x8(%rbp),%eax
19:	89 45 fc	mov	%eax,−0x4(%rbp)
1c:	8b 45 fc	mov	−0x4(%rbp),%eax

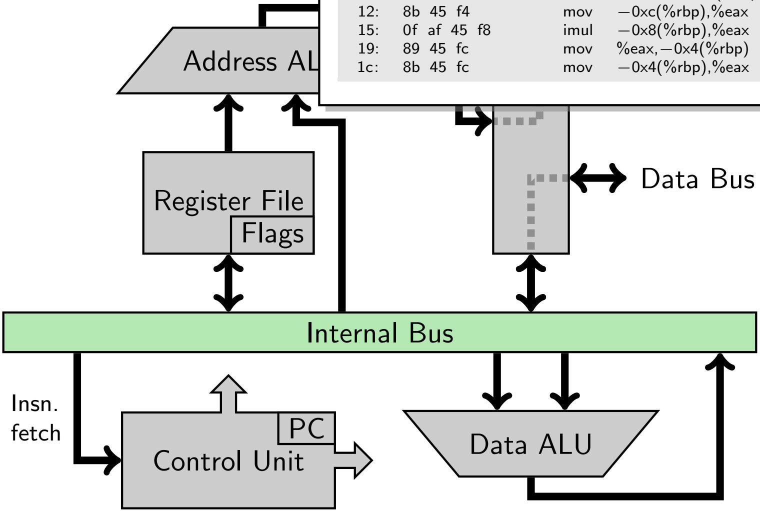
Things to know:

- Addressing modes (Immediate, Register, Base plus Offset)
- 0xHexadecimal
- "AT&T Form": (we'll use this)
  `<opcode><size> <source>, <dest>`

Another Look



A




```
 4:   c7 45 f4 05 00 00 00   movl  $0x5,-0xc(%rbp)
 b:   c7 45 f8 11 00 00 00   movl  $0x11,-0x8(%rbp)
12:   8b 45 f4               mov   -0xc(%rbp),%eax
15:   0f af 45 f8            imul  -0x8(%rbp),%eax
19:   89 45 fc               mov   %eax,-0x4(%rbp)
1c:   8b 45 fc               mov   -0x4(%rbp),%eax
```

A Very Simple Program: Intel Form

```
  4:   c7 45 f4 05 00 00 00   mov    DWORD PTR [rbp-0xc],0x5
  b:   c7 45 f8 11 00 00 00   mov    DWORD PTR [rbp-0x8],0x11
 12:   8b 45 f4               mov    eax,DWORD PTR [rbp-0xc]
 15:   0f af 45 f8            imul   eax,DWORD PTR [rbp-0x8]
 19:   89 45 fc               mov    DWORD PTR [rbp-0x4],eax
 1c:   8b 45 fc               mov    eax,DWORD PTR [rbp-0x4]
```

- “Intel Form”: (you might see this on the net)
  `<opcode> <sized dest>, <sized source>`
- Goal: Reading comprehension.
- Don’t understand an opcode?
  Google “`<opcode> intel instruction`”.

Machine Language Loops

```c
int main()
{
  int y = 0, i;
  for (i = 0;
       y < 10; ++i)
    y += i;
  return y;
}
```

```
 0:   55                      push   %rbp
 1:   48 89 e5                mov    %rsp,%rbp
 4:   c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
 b:   c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
12:   eb 0a                   jmp    1e <main+0x1e>
14:   8b 45 fc                mov    -0x4(%rbp),%eax
17:   01 45 f8                add    %eax,-0x8(%rbp)
1a:   83 45 fc 01             addl   $0x1,-0x4(%rbp)
1e:   83 7d f8 09             cmpl   $0x9,-0x8(%rbp)
22:   7e f0                   jle    14 <main+0x14>
24:   8b 45 f8                mov    -0x8(%rbp),%eax
27:   c9                      leaveq
28:   c3                      retq
```

Things to know:

- Condition Codes (Flags): Zero, Sign, Carry, etc.
- Call Stack: Stack frame, stack pointer, base pointer
- ABI: Calling conventions

Web demo

`http://assembly.ynh.io/`

demo time

Other web-based assembly viewers

- http://assembly.ynh.io/
  [https://github.com/ynh/cpp-to-assembly]
- http://gcc.godbolt.org/
- http://llvm.org/demo/

Assembly comprehension/optimizer

Outline

What is. . . a Memory Interface?

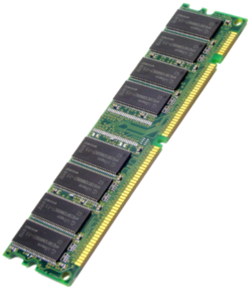
Memory Interface gets and stores binary words in off-chip memory.

Smallest granularity: Bus width

Tells outside memory

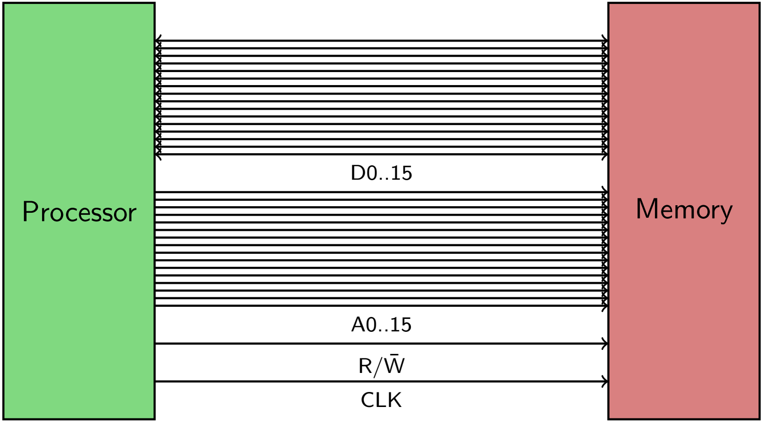
- "where" through *address bus*
- "what" through *data bus*

Computer main memory is "Dynamic RAM" (DRAM): Slow, but small and cheap.



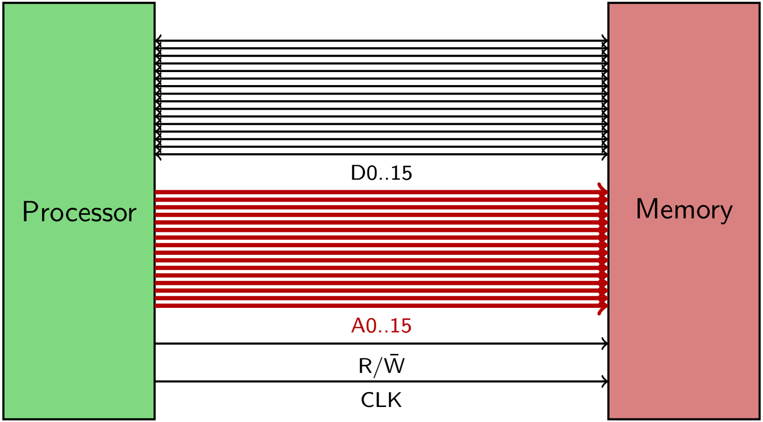
How does computer memory work?

One (reading) memory transaction (simplified):



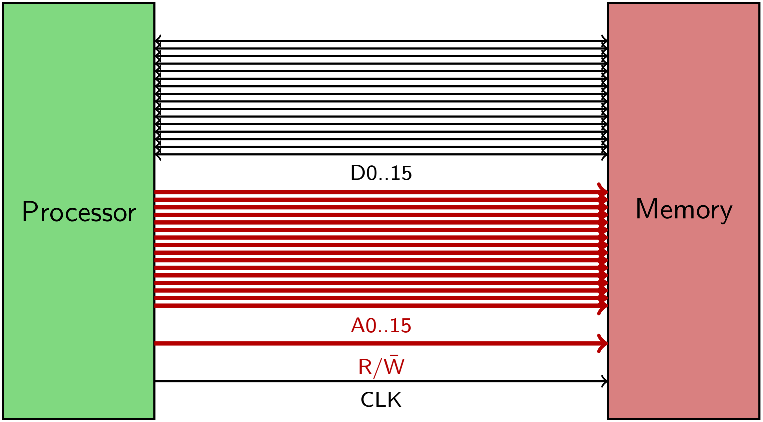
How does computer memory work?

One (reading) memory transaction (simplified):



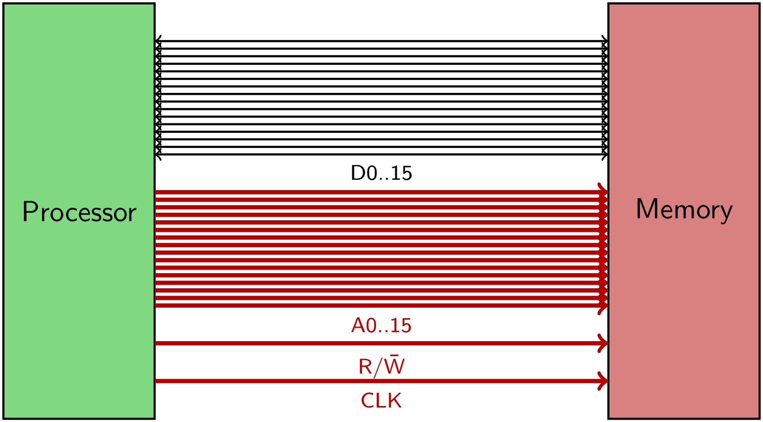
How does computer memory work?

One (reading) memory transaction (simplified):



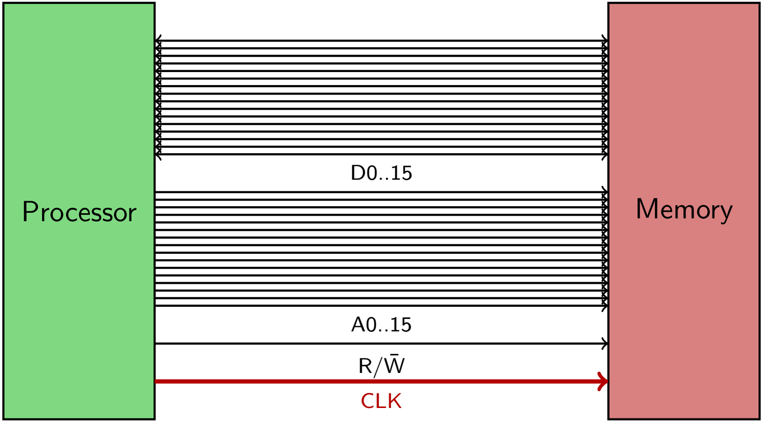
How does computer memory work?

One (reading) memory transaction (simplified):



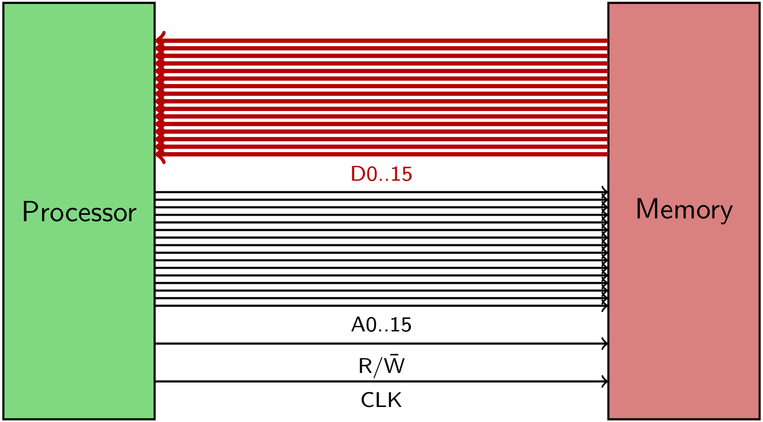
How does computer memory work?

One (reading) memory transaction (simplified):



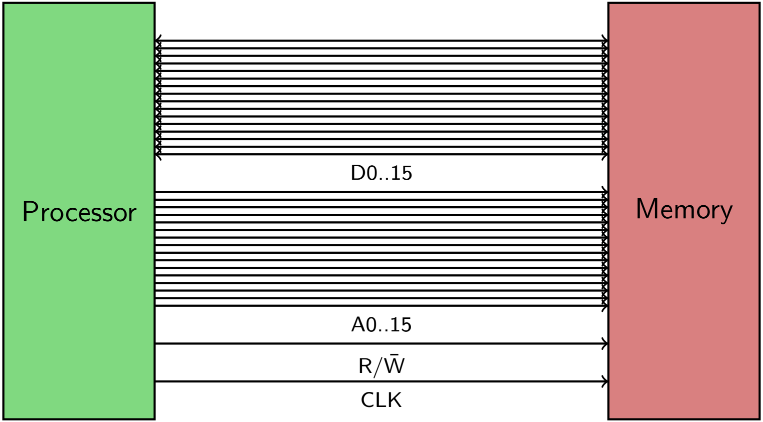
How does computer memory work?

One (reading) memory transaction (simplified):



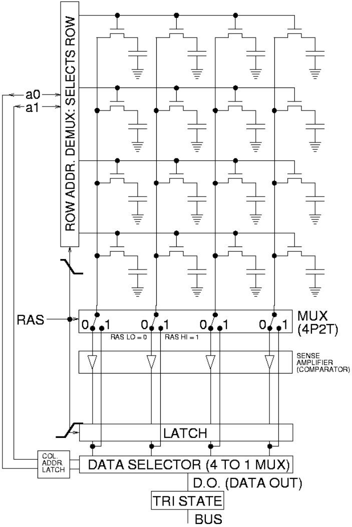
How does computer memory work?

One (reading) memory transaction (simplified):

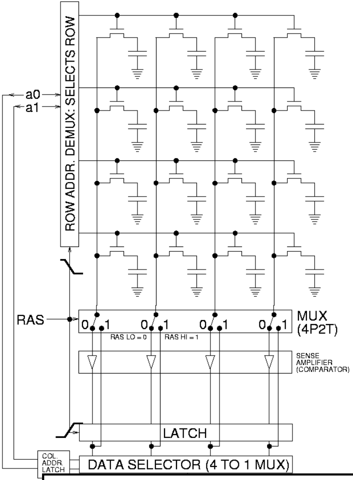


Observation: Access (and addressing) happens in bus-width-size "chunks".

DRAM

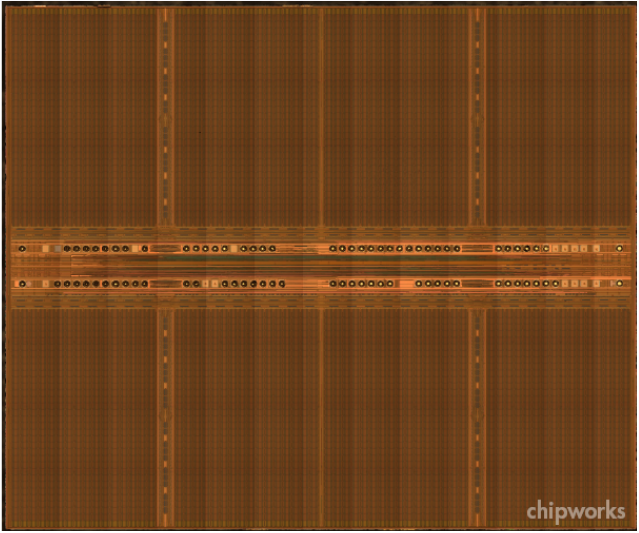

ROW ADDR. DEMUX: SELECTS ROW
a0
a1
RAS
0
1
MUX
(4P2T)
RAS LO = 0
RAS HI = 1
SENSE
AMPLIFIER
(COMPARATOR)
LATCH
COL.
ADDR.
LATCH
DATA SELECTOR (4 TO 1 MUX)
D.O. (DATA OUT)
TRI STATE
BUS

DRAM



Key: each cell is *tiny* → many of them!

DRAM die



Samsung 1 Gib DDR3 die

Outline

Outline

We know how a computer works!

All of this can be built in about 4000 transistors.
(e.g. MOS 6502 in Apple II, Commodore 64, Atari 2600)

So what exactly is Intel doing with the other 623,996,000 transistors?

Answer:

We know how a computer works!

All of this can be built in about 4000 transistors.
(e.g. MOS 6502 in Apple II, Commodore 64, Atari 2600)

So what exactly is Intel doing with the other 623,996,000 transistors?

Answer: *Make things go faster!*

Go-fast widgets

All this go-faster technology: **hard to see**.

Most of the time:

- program fast,
- programmer happy.

Sometimes that's not the case.

Go-fast widgets

All this go-faster technology: **hard to see**.

Most of the time:

- program fast,
- programmer happy.

Sometimes that's not the case.

Goal now: Break each widget in an understandable way.

Outline

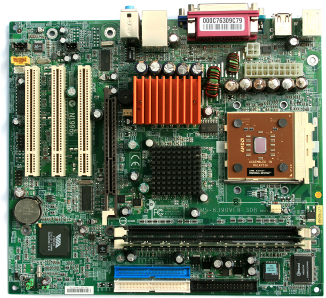
Source of Slowness: Memory

Memory is slow.

Distinguish two different versions of “slow”:

- Bandwidth
- Latency

→ Memory has *long latency*, but can have *large bandwidth*.



Size of die vs. distance to memory: big!

Dynamic RAM: long intrinsic latency!

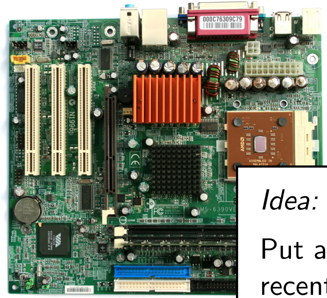
Source of Slowness: Memory

Memory is slow.

Distinguish two different versions of “slow”:

- Bandwidth
- Latency

→ Memory has *long latency*, but can have *large bandwidth*.



Size of die vs. distance to memory: big

Dynamic RAM: long intrinsic latency!

Idea:

Put a look-up table of recently-used data onto the chip.

→ “Cache”

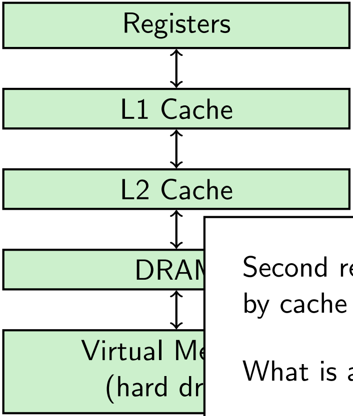
The Memory Hierarchy

Hierarchy of increasingly bigger, slower memories:

Registers	1 kB, 1 cycle
L1 Cache	10 kB, 10 cycles
L2 Cache	1 MB, 100 cycles
DRAM	1 GB, 1000 cycles
Virtual Memory (hard drive)	1 TB, 1 M cycles

The Memory Hierarchy

Hierarchy of increasingly bigger, slower memories:



1 kB, 1 cycle

10 kB, 10 cycles

1 MB, 100 cycles

Second red/blue pebble game: played by cache controller

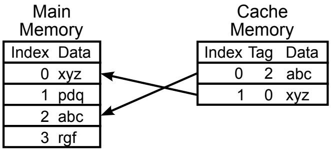
What is a *working set*?

How might *data locality* factor into this?

Cache: Actual Implementation

Demands on cache implementation:

- Fast, small, cheap, low power
- Fine-grained
- High "hit"-rate (few "misses")



Problem:

Goals at odds with each other: Access matching logic expensive!

Solution 1: More data per unit of access matching logic

$\rightarrow$ Larger "Cache Lines"

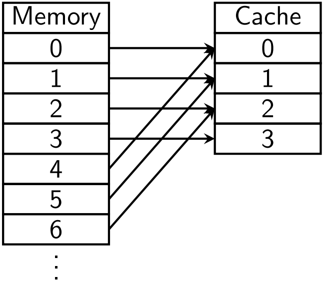
Solution 2: Simpler/less access matching logic

$\rightarrow$ Less than full "Associativity"

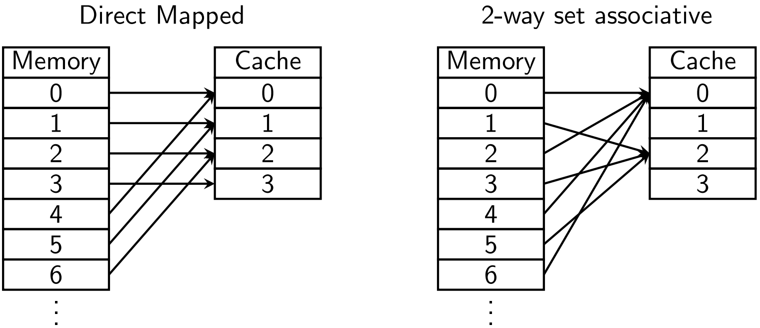
Other choices: Eviction strategy, size

Cache: Associativity

Direct Mapped

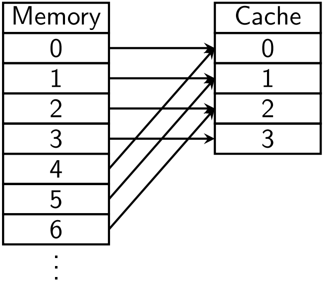


Cache: Associativity

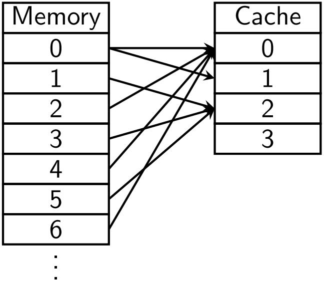


Cache: Associativity

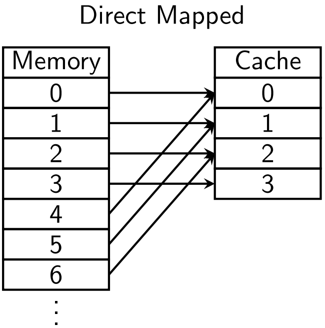
Direct Mapped

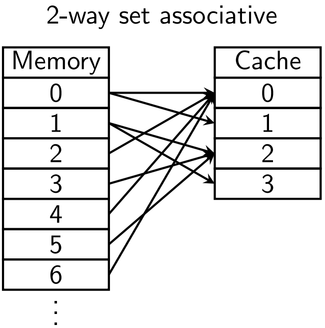


2-way set associative



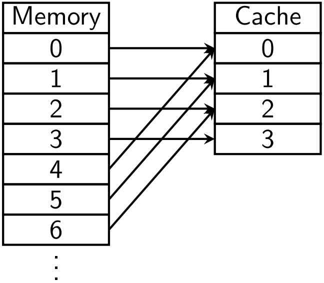
Cache: Associativity



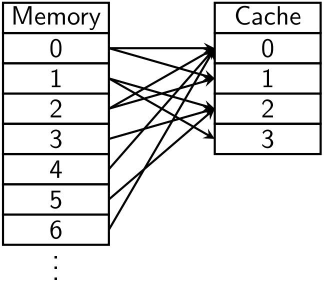


Cache: Associativity

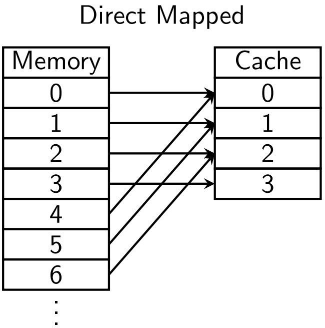
Direct Mapped

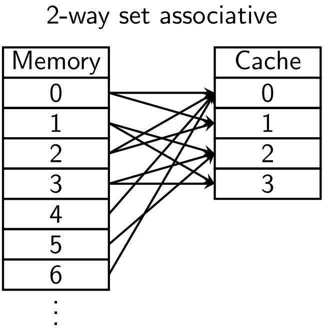


2-way set associative



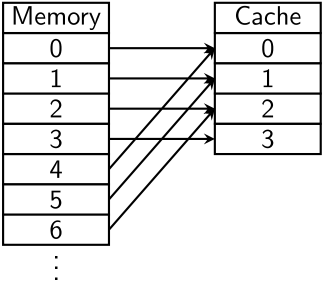
Cache: Associativity



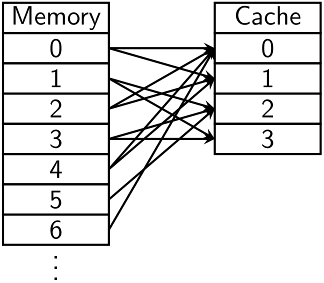


Cache: Associativity

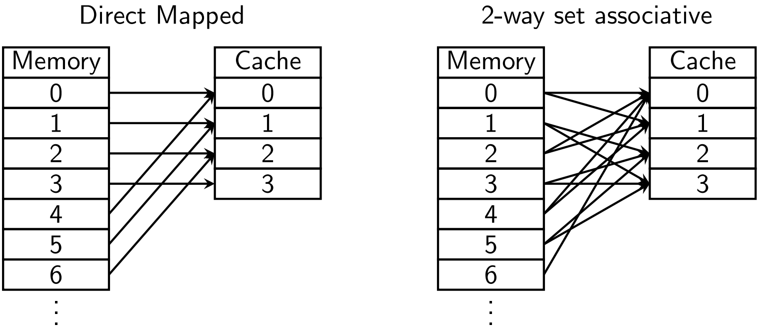
Direct Mapped



2-way set associative

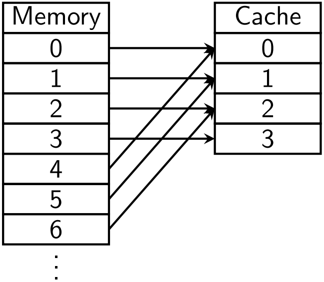


Cache: Associativity

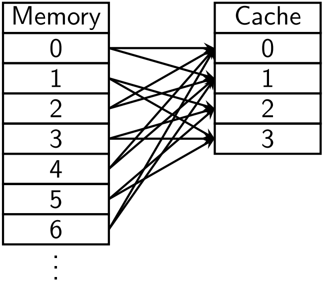


Cache: Associativity

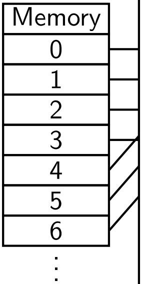
Direct Mapped

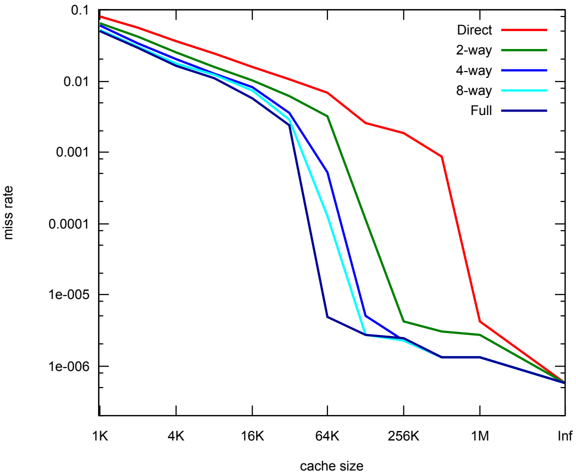


2-way set associative



Direc





Miss rate versus cache size on the Integer portion of SPEC CPU2000 [Cantin, Hill 2003]

CPUID demo time

Updating every kth integer

```
int go(unsigned count, unsigned stride)
{
  const unsigned array_size = 64 * 1024 * 1024;
  int *ary = (int *) malloc(sizeof(int) * array_size);

  for (unsigned it = 0; it < count; ++it)
  {
    for (unsigned i = 0; i < array_size; i += stride)
      ary[i] *= 17;
  }

  int result = 0;
  for (unsigned i = 0; i < array_size; ++i)
      result += ary[i];

  free(ary);
  return result;
}
```

Original benchmarks by Igor Ostrovsky

Updating every kth integer

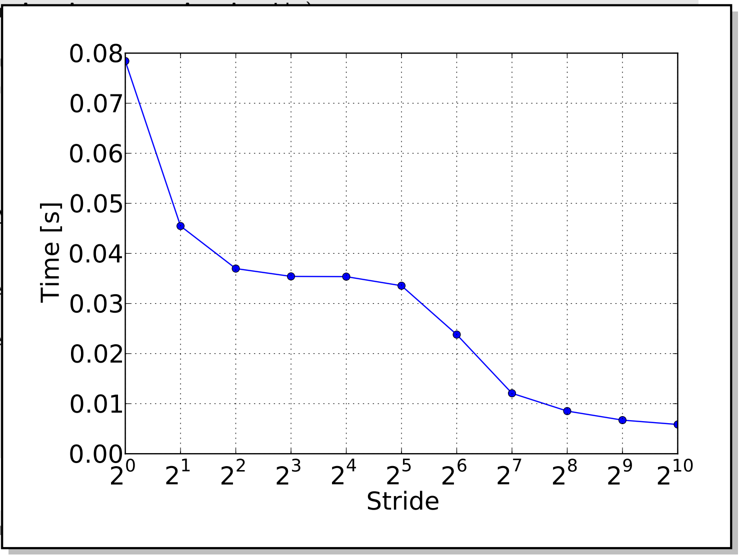
```
int go(u
{
  const (
  int *a

  for (u
  {
    for
      ary
  }

  int re
  for (u
      re

  free(a
  return
}
```

Original ben



Measuring bandwidths

```
int go(unsigned array_size , unsigned steps)
{
  int *ary = (int *) malloc(sizeof(int) * array_size );
  unsigned asm1 = array_size - 1;

  for (unsigned i = 0; i < 100*steps;)
  {
    #define ONE ary[(i++*16) & asm1] ++;
    #define FIVE ONE ONE ONE ONE ONE
    #define TEN FIVE FIVE
    #define FIFTY TEN TEN TEN TEN TEN
    #define HUNDRED FIFTY FIFTY
    HUNDRED
  }

  int result = 0;
  for (unsigned i = 0; i < array_size ; ++i)
      result += ary[i];

  free(ary);
  return result ;
}
```

Original benchmarks by Igor Ostrovsky

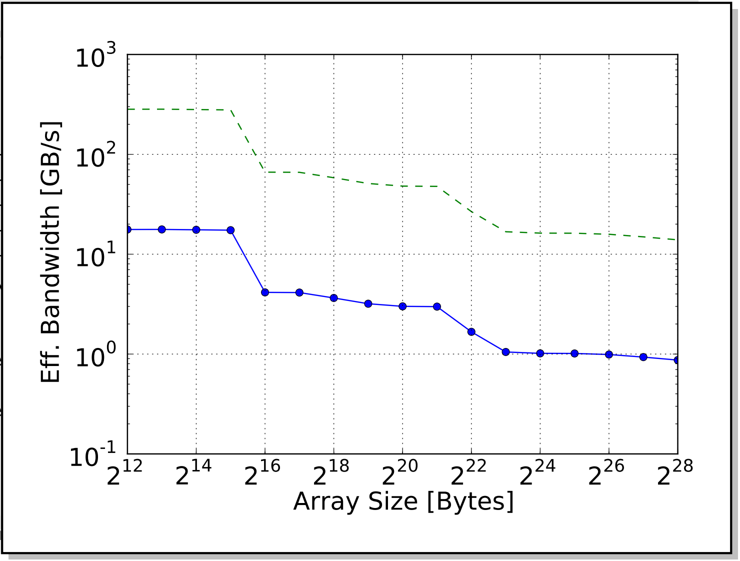
Measuring bandwidths

```
int go(unsigned array_size , unsigned steps)
{
  int *a
  unsign

  for (u
  {
    #de
    #de
    #de
    #de
    #de
    HUN
  }

  int re
  for (u
      re

  free(a
  return
}
```



Original ben

Another mystery

```
int go(unsigned array_size , unsigned stride , unsigned steps)
{
  char *ary = (char *) malloc(sizeof(int) * array_size );

  unsigned p = 0;
  for (unsigned i = 0; i < steps; ++i)
  {
    ary[p] ++;
    p += stride;
    if (p >= array_size)
      p = 0;
  }

  int result = 0;
  for (unsigned i = 0; i < array_size ; ++i)
      result += ary[i];

  free(ary);
  return result ;
}
```

Original benchmarks by Igor Ostrovsky

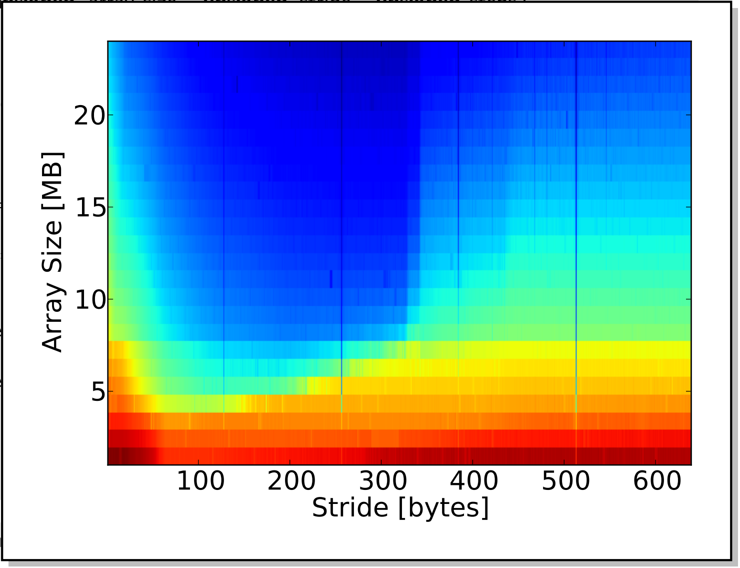
Another mystery

```
int go(unsigned array_size, unsigned stride, unsigned steps)
{
  char *

  unsign
  for (u
  {
    ary[
    p +=
    if (
      p
  }

  int re
  for (u
      re

  free(a
  return
}
```



Original ben

Core Message

Learned a lot about caches.

Also learned:

Honest measurements are *hard*.

A good attempt:

`http://www.bitmover.com/lmbench/`

Instructions:

`http://download.intel.com/design/intarch/papers/321074.pdf`

Programming for the Hierarchy

How can we rearrange programs to friendly to the memory hierarchy?

Examples:

- Large vectors x, a, b
  Compute
  $$x \leftarrow x + 3a - 5b.$$

Programming for the Hierarchy

How can we rearrange programs to friendly to the memory hierarchy?

Examples:

- Large vectors x, a, b
  Compute
  $$x \leftarrow x + 3a - 5b.$$
- Matrix-Matrix Multiplication

Outline

Source of Slowness: Sequential Operation

IF	Instruction fetch
ID	Instruction Decode
EX	Execution
MEM	Memory Read/Write
WB	Result Writeback

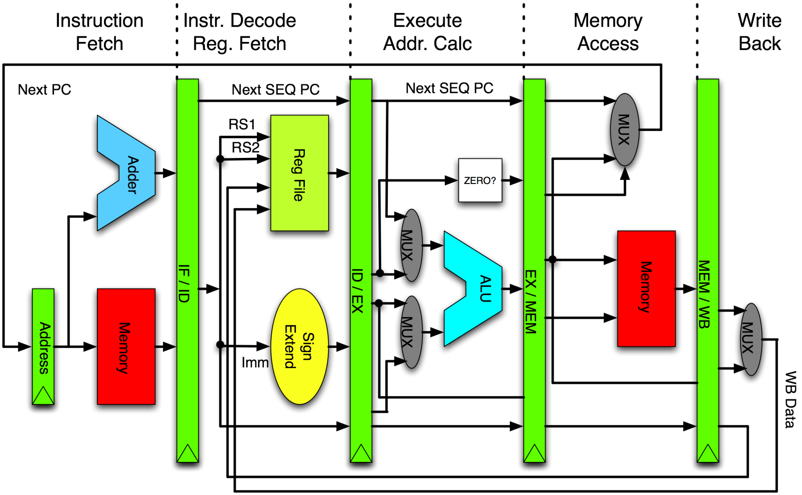
Solution: Pipelining

IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB

i ↓

t →

Pipelining



(MIPS, 110,000 transistors)

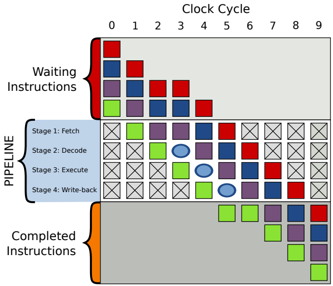
Issues with Pipelines

Pipelines generally help performance–but not always.

Possible issue: Dependencies. . .

- . . . on memory
- . . . on previous computation
- . . . on branch outcomes

"Solution": Bubbling



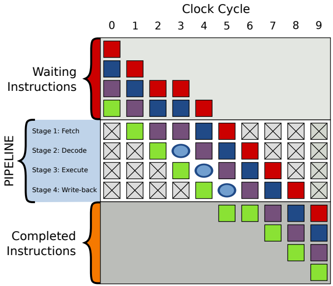
Issues with Pipelines

Pipelines generally help performance–but not always.

Possible issue: Dependencies. . .

- . . . on memory
- . . . on previous computation
- . . . on branch outcomes

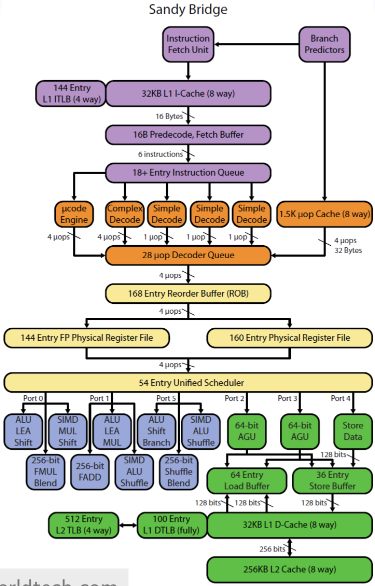
"Solution": Bubbling



For branches: could guess. . . ?

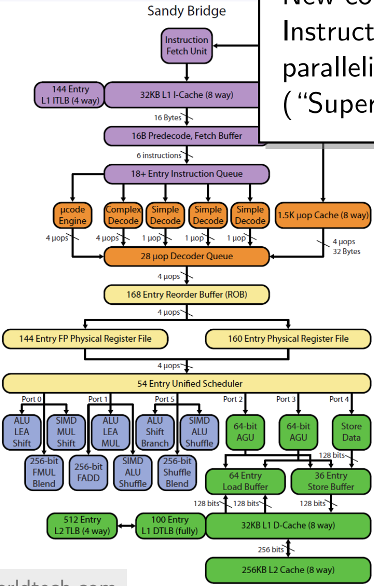
Performance mystery demo time

Sandy Bridge Pipeline



David Kanter / Realworldtech.com

Sandy Bridge Pi...



New concept: Instruction-level parallelism ("Superscalar")

David Kanter / Realworldtech.com

More Pipeline Mysteries

Outline

Floating point performance demo

Questions?

?

Image Credits

- DRAM: Wikipedia
- DRAM die: chipworksrealchips.com / Samsung
- Basic cache: Wikipedia
- Cache associativity: based on Wikipedia
- Cache associativity vs miss rate: Wikipedia,
- Cache Measurements: Igor Ostrovsky
- Pipelining: Wikipedia
- Bubbly Pipeline: Wikipedia