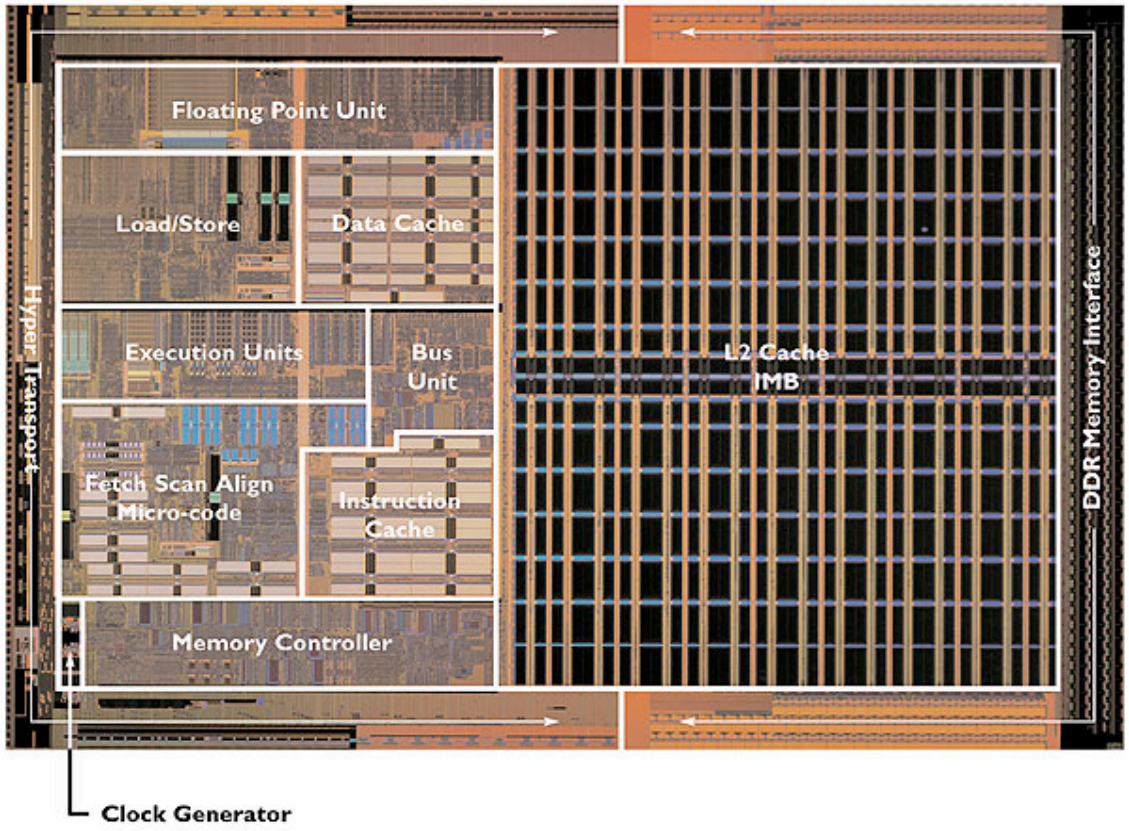
CS232: Computer Architecture II

Spring 2009



Who we are

- *Lecturer*:

  Prof. Craig Zilles

  - I do research on computer architecture and compilers

- *Section Instructors & Teaching Assistants*:

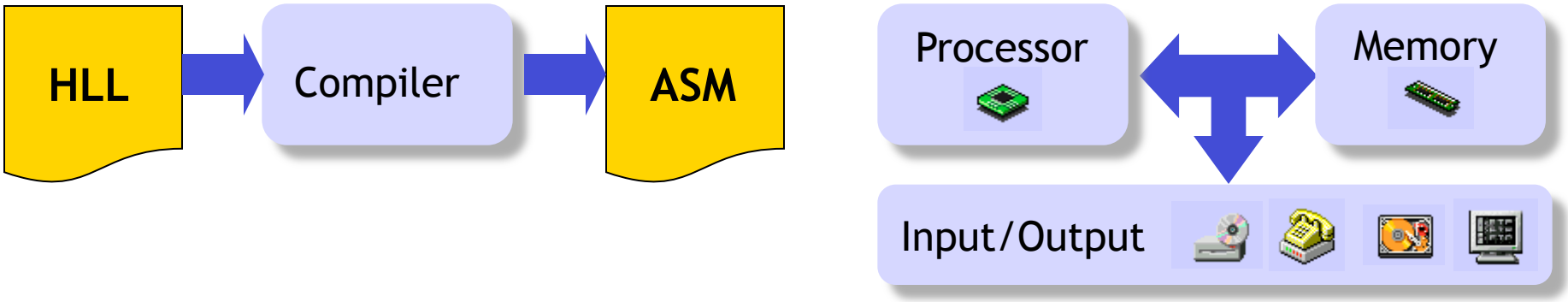
  Samer Fanek

  Abner Guzman Rivera

  Brett Jones

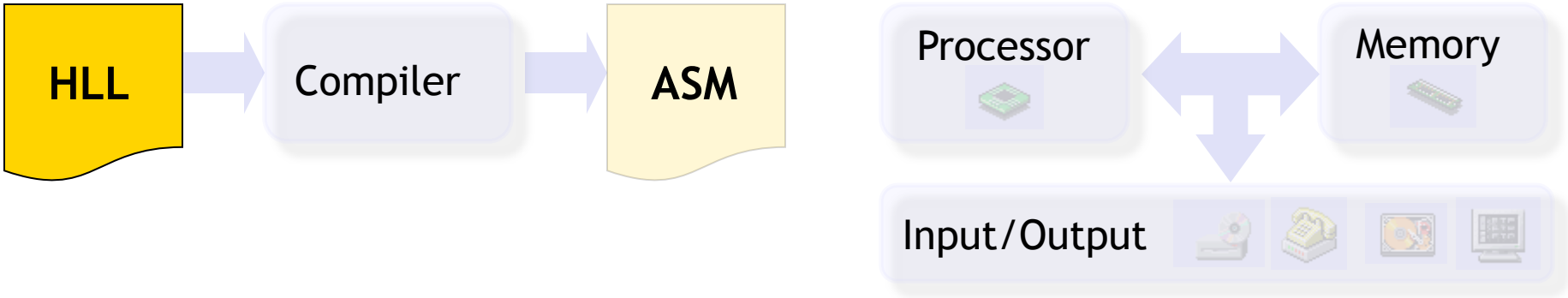
What is computer architecture about?

- Computer architecture is about building and analyzing computer systems.



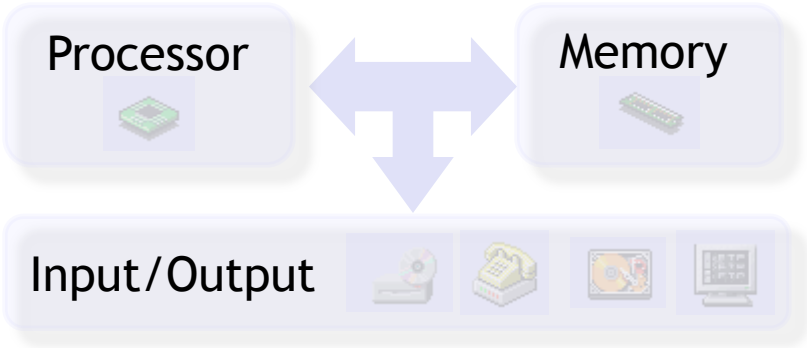
- In CS232, we will take a tour of the whole machine.
- Specifically, we'll…

Do low-level programming in a high-level language



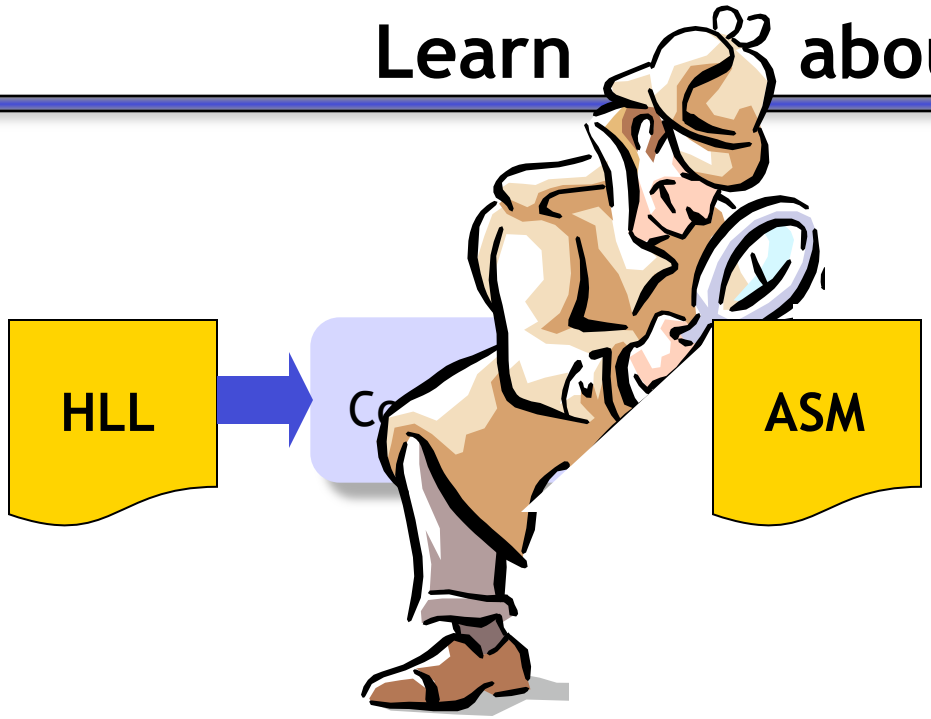
- We'll look at bit-wise logical and shifting operations in C.

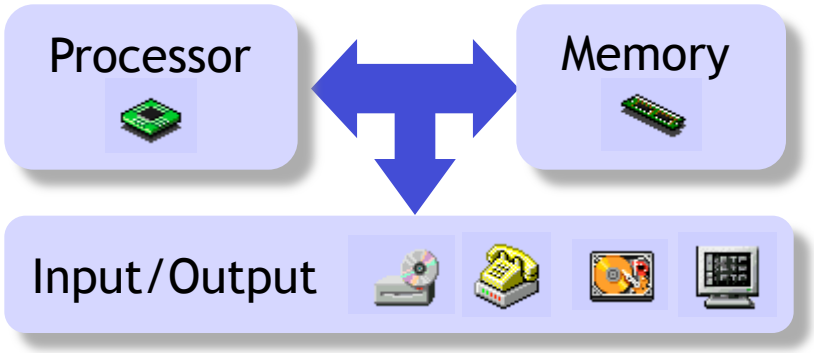
Study Instruction Set Architectures



- The Instruction Set Architecture (ISA) is the bridge between the hardware and the software.
  - We'll learn the MIPS ISA in detail
  - We'll get a brief introduction to the x86 ISA
  - We'll learn how HLL program constructs are represented to the machine
  - We won't learn how compilers work, but we'll learn what they do

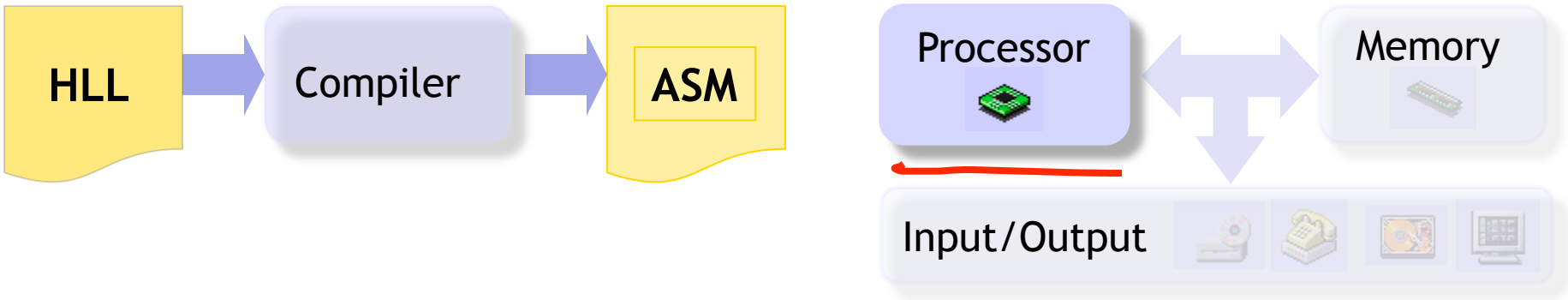
Learn about performance





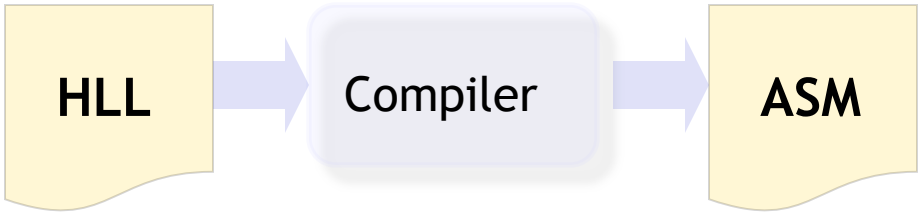
- We'll learn how to performance tune programs.
- We'll exploit explicit parallelism to make programs run faster
  - We'll optimize a program using SSE instructions

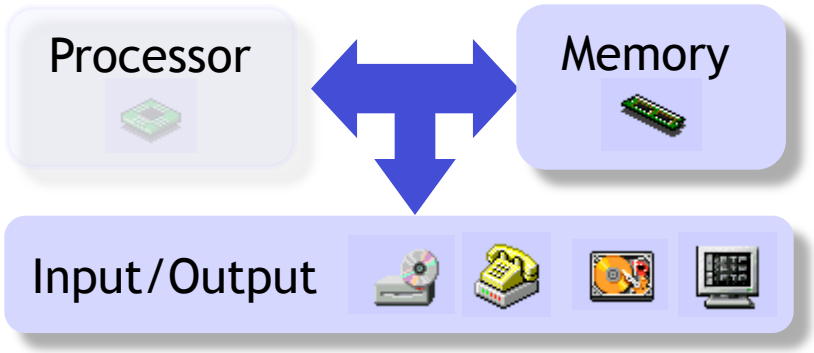
Learn about Modern Processor Organization



- The key technique we'll focus on is: Pipelining
  - Pipelining allows processors to work on multiple instructions at the same time.

Learn about Memory and I/O systems





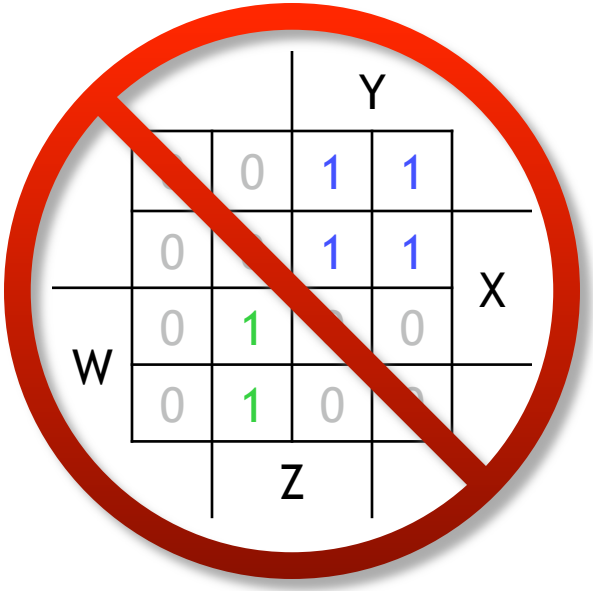
- We'll learn how virtual memory makes programming easy
- We'll learn how caches make memory fast
- We'll learn about buses and disks

Why should you care?

- It is interesting.
  - How do you make a processor that runs at 3Ghz?
- It will help you be a better programmer.
  - Understanding how your program is translated to assembly code lets you reason about correctness and performance.
  - Demystify the seemingly arbitrary (*e.g.*, bus errors, segmentation faults)
- Many cool jobs require an understanding of computer architecture.
  - The cutting edge is often pushing computers to their limits.
  - Supercomputing, games, portable devices, etc.
- Computer architecture illustrates many fundamental ideas in computer science
  - Abstraction, caching, and indirection are CS staples

CS231 vs. CS232

- This class expands upon the computer architecture material from the last few weeks of CS231, and we rely on many other ideas from CS231.
  - Understanding binary, hexadecimal and two's-complement numbers is still important.
  - Devices like multiplexers, registers and ALUs appear frequently. You should know what they do, but not necessarily how they work.
  - Finite state machines and sequential circuits will appear again.
- We do *not* spend much time with logic design topics like Karnaugh maps, Boolean algebra, latches and flip-flops.



Low-level Programming in “High-level” Languages

- Very often it is necessary to store a large number of very small data items.

Low-level Programming in "High-level" Languages

- Very often it is necessary to store a large number of very small data items.

- Example: A Social Security Number (SSN) registry
  - Needs to keep track of how which SSNs have already been allocated.
- How much space is required?

int SSN [1 Billion]

123 45 6789

10^9 = 1 Billion

int = 32b = 4B $\frac{8b}{1B}$

4 Billion 1b

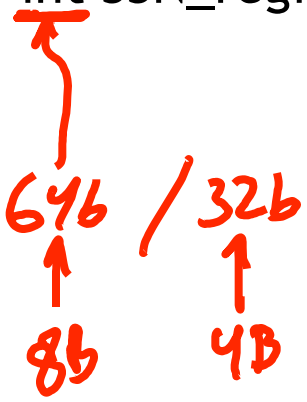
4 GB

Storing collections of bits as integers

- Store N bits in each N-bit integer, only need $10^9/N$ integers
  - Requires $10^9/8$ bytes = 125MBs of storage (fits on a CD)

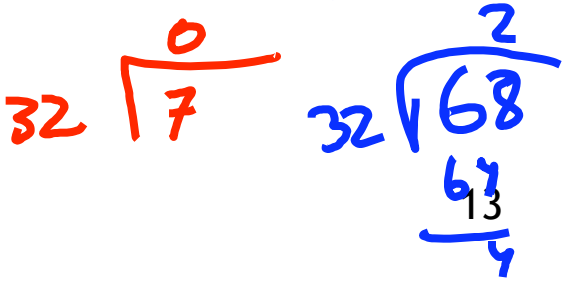
- Allocate array:

```
int array_size = 1000000000/(sizeof(int)*8)
unsigned int SSN_registry[array_size];
```



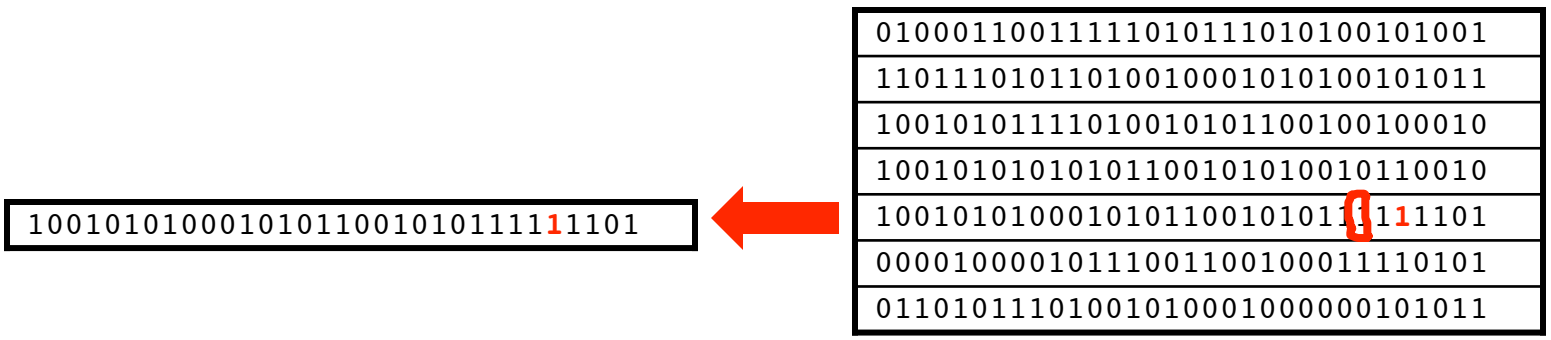
- Want two operations on this array:
  - check_SSN: returns 1 if SSN is used, 0 otherwise
  - set_SSN: marks an SSN as used.

SSN #7
SSN #68



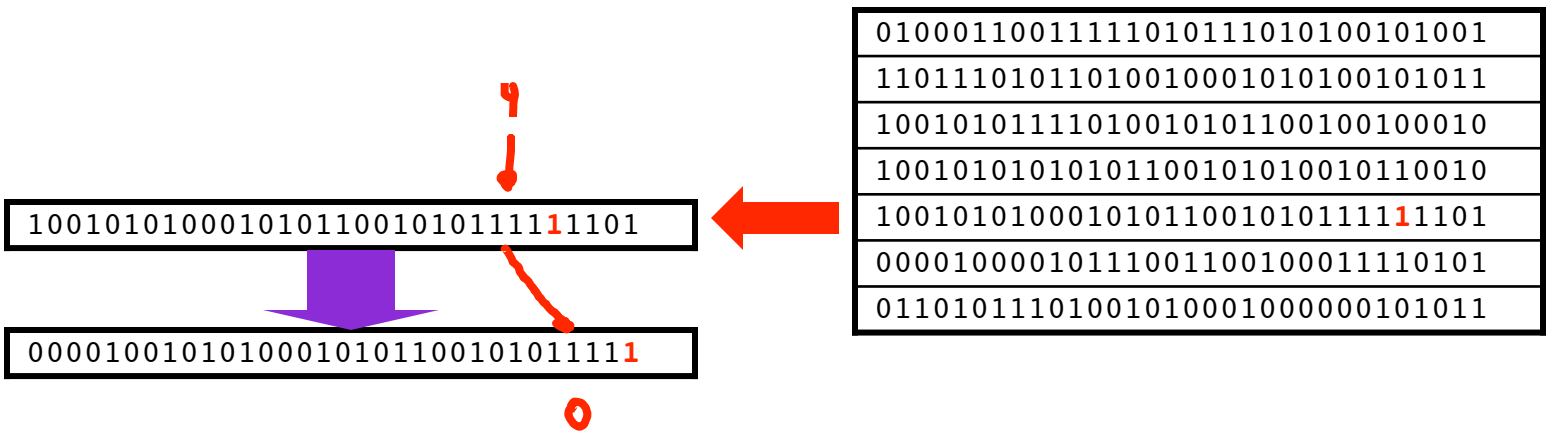
check_SSN

```
int check_SSN(unsigned int SSN_registry[], int ssn) {
    int word_index = ssn / (8*sizeof(int));
    int word = SSN_registry[word_index];
```



check_SSN

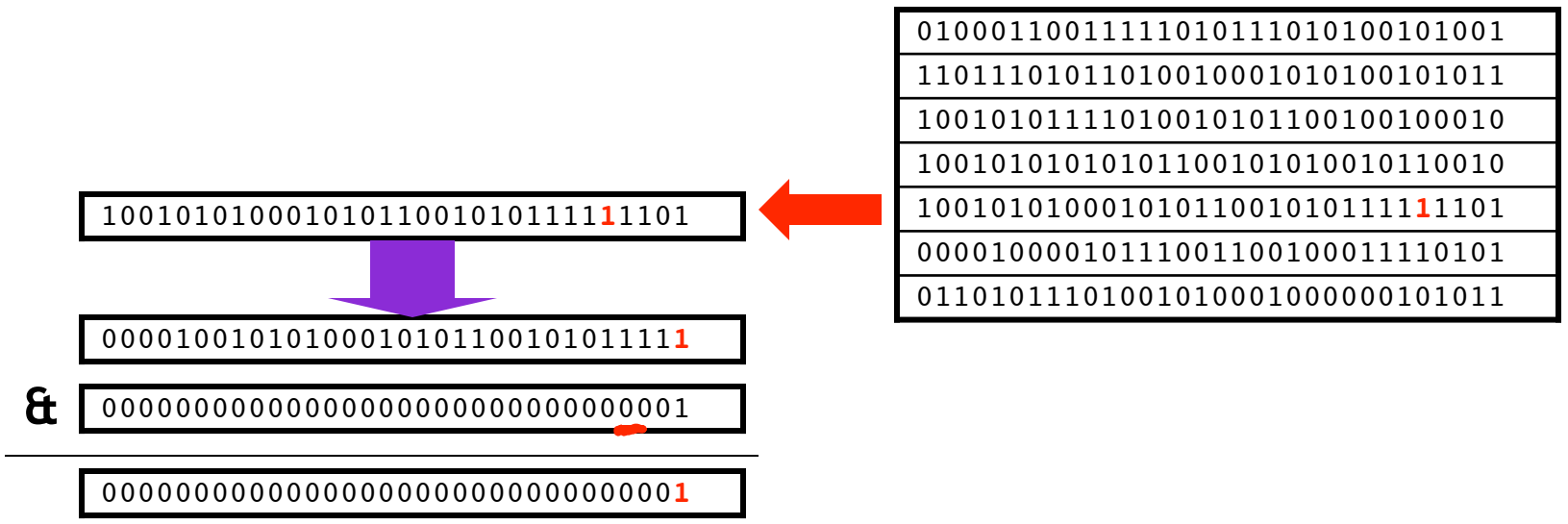
```
int check_SSN(unsigned int SSN_registry[], int ssn) {
    int word_index = ssn / (8*sizeof(int));
    int word = SSN_registry[word_index];
    int bit_offset = ssn % (8*sizeof(int)) // % is the remainder operation
    word = word >> bit_offset;  // >> shifts a value "right"
        (note: zeros are inserted at the left because it is an unsigned int)
```



check_SSN

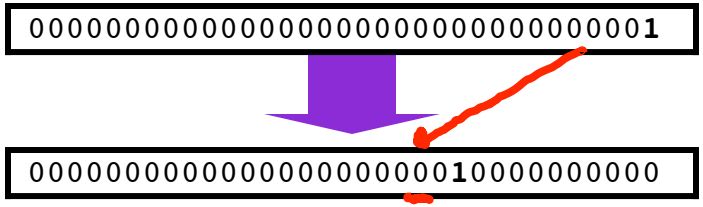
```
int check_SSN(unsigned int SSN_registry[], int ssn) {
    int word_index = ssn / (8*sizeof(int));
    int word = SSN_registry[word_index];
    int bit_offset = ssn % (8*sizeof(int))
    word = word >> bit_offset;
    word = (word & 1);  // & is the bit-wise logical AND operator
    return word;
}
```

(each bit position is considered independently)



set_SSN

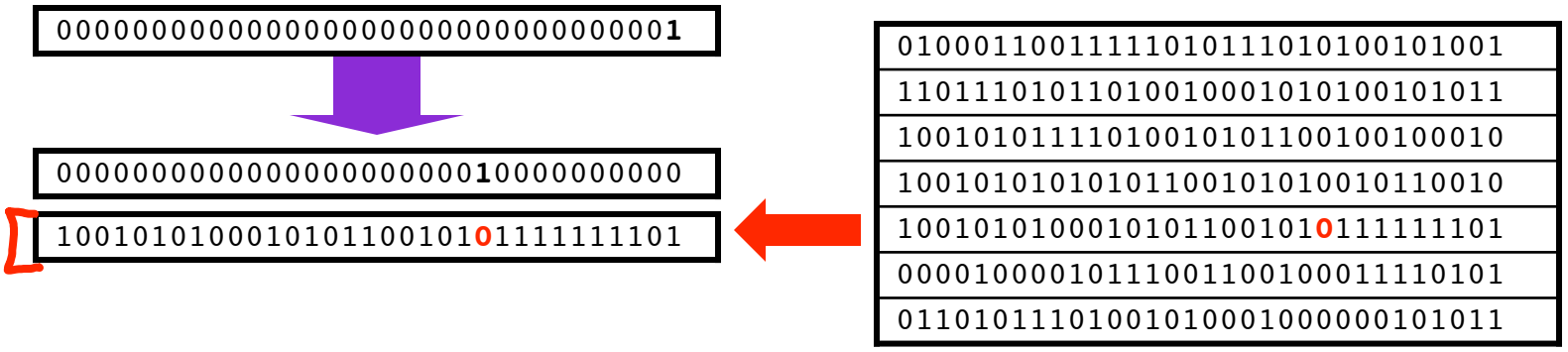
```
void set_SSN(unsigned int SSN_registry[], int ssn) {
    int bit_offset = ssn % (8*sizeof(int))
    int new_bit = (1 << bit_offset) // "left" shift the 1 to the desired spot
                                    (always shifts in 0's at the right)
```



01000110011111101011101010010100 1
11011101011010010001010100101011
10010101111010010101100100100010
10010101010101100101010010110010
10010101000101011001010111111101
00001000010111001100100011110101
01101011101001010001000000101011

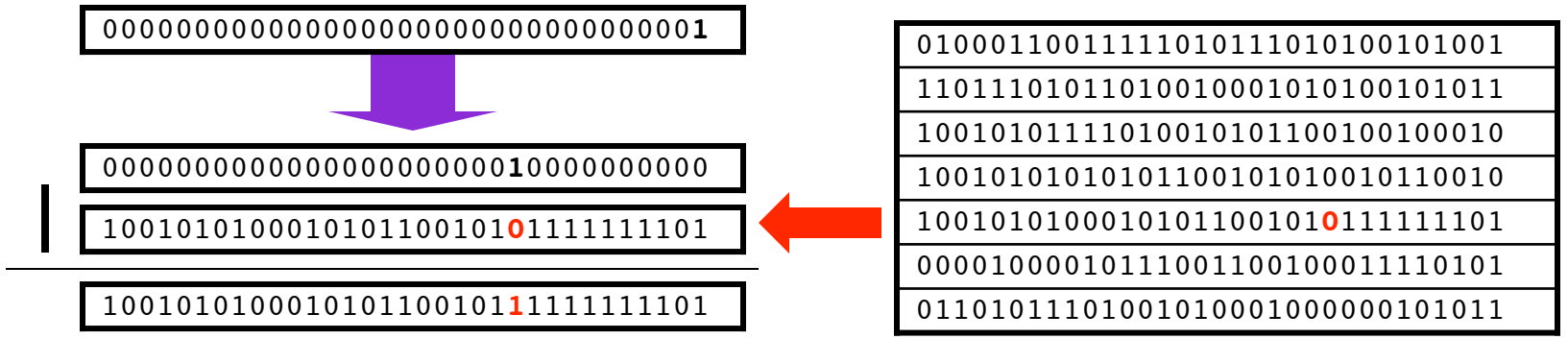
set_SSN

```
void set_SSN(unsigned int SSN_registry[], int ssn) {
    int bit_offset = ssn % (8*sizeof(int))
    int new_bit = (1 << bit_offset)
    int word_index = ssn / (8*sizeof(int));
    int word = SSN_registry[word_index];
```



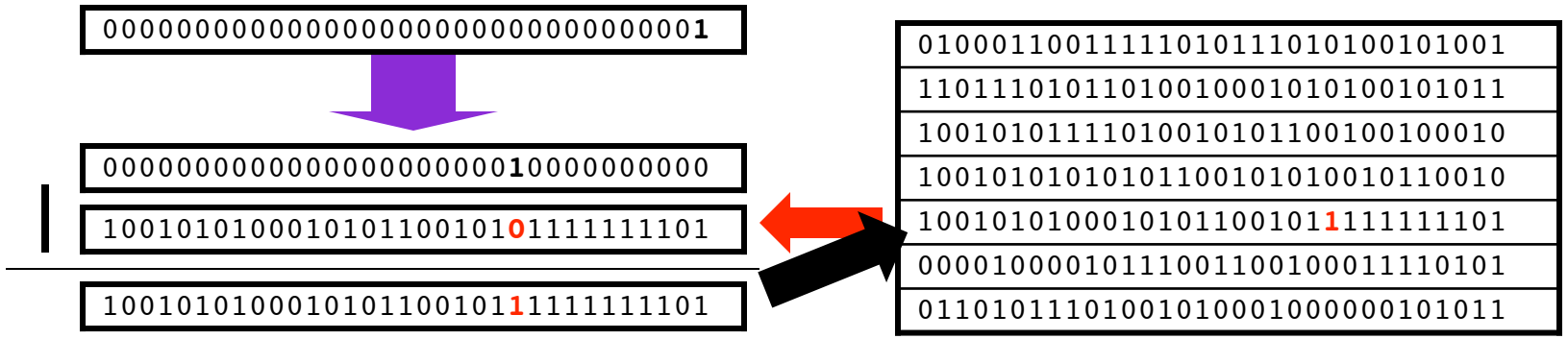
set_SSN

```
void set_SSN(unsigned int SSN_registry[], int ssn) {
    int bit_offset = ssn % (8*sizeof(int))
    int new_bit = (1 << bit_offset)
    int word_index = ssn / (8*sizeof(int));
    int word = SSN_registry[word_index];
    word = word | new_bit;  // bit-wise logical OR sets the desired bit
```



set_SSN

```
void set_SSN(unsigned int SSN_registry[], int ssn) {
    int bit_offset = ssn % sizeof(int)
    int new_bit = (1 << bit_offset)
    int word_index = ssn / sizeof(int);
    int word = SSN_registry[word_index];
    word = word | new_bit;
    SSN_registry[word_index] = word;  // write back the word into array
}
```



Shorthand for last 3 lines: SSN_registry[word_index] |= new_bit;

What you just saw

- Storage of a collection of booleans in an integer
- Use of bit-wise logical operations to read and write specific bits
- Use of shifts to move bits with an integer

- This stuff gets used all over the place in real code:
  - Bitmap graphics
  - Network packet headers
  - Operating system tracking free disk blocks

How the class will be organized

- The textbook provides the most comprehensive coverage
- Lecture and section will present course material
- Section problems useful for gauging your understanding of the material
  - Weekly, graded on effort, and good practice for the exams
- Machine problems are more open-ended applications of course material
  - Due most weeks, graded, can be done in groups (1-3 people)
- Homeworks used for closed-form, quantitative problems
  - Due occasionally, graded
- Exams: three in-class midterms and one final

- See the syllabus:

  http://www.cs.uiuc.edu/class/cs232/html/info.html

- Questions?

Sections start next week. MP's start next week!!

- Go to section!

- MP#1 will be due the next Wednesday night.
  - It will be out tomorrow
  - Make sure you have a working EWS account soon!!!
    - Contact the TA's if you are not an engineering major
  - It covers doing bit-wise logical and shifting in C

- MP#0 will be due next Friday.
  - It is already out.
  - It is a SPIM tutorial (this will make more sense on Friday)

General hints to reach CS232 nirvana

- **Remember the big picture.**
  What are we trying to accomplish, and why?
- **Read the textbook.**
  It's clear, well-organized, and well-written. The diagrams can be complex, but are worth studying. Work through the examples and try some exercises on your own. Read the "Real Stuff" and "Historical Perspective" sections.
- **Talk to each other.**
  You can learn a lot from other CS232 students, both by asking and answering questions. Find some good partners for the homeworks (but make sure you all understand what's going on).
- **Help us help you.**
  Come to lectures, sections and office hours. Send email or post on the newsgroup. Ask lots of questions! Check out the web page:

http://www-courses.cs.uiuc.edu/~cs232

What you will need to learn this month

- You must become "fluent" in MIPS assembly:
  - Translate from C to MIPS and MIPS to C

- Example problem from last mid-term 1:

  Question 3: Write a recursive function (30 points)

  Here is a function pow that takes two arguments (n and m, both 32-bit numbers) and returns n^m (i.e., n raised to the m^{th} power).

```
int
pow(int n, int m) {
   if (m == 1)
      return n;
   return n * pow(n, m-1);
}
```

  Translate this into a MIPS assembly language function.