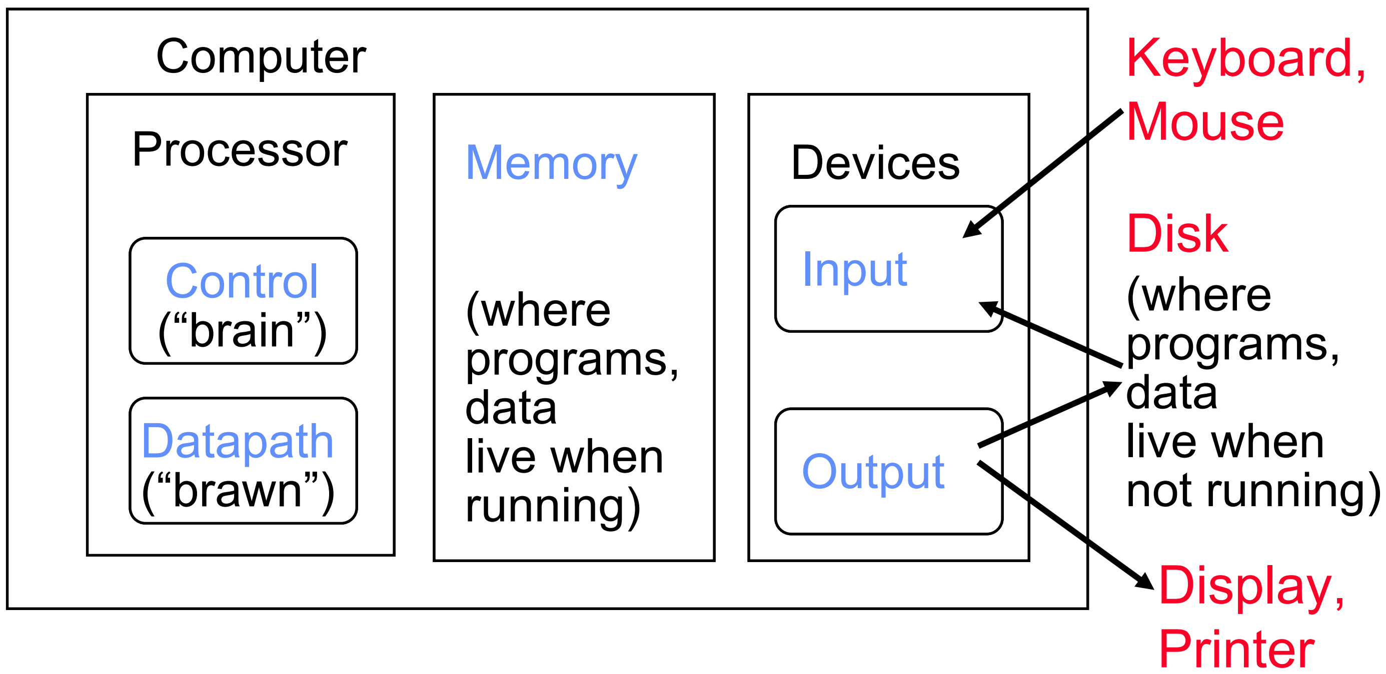
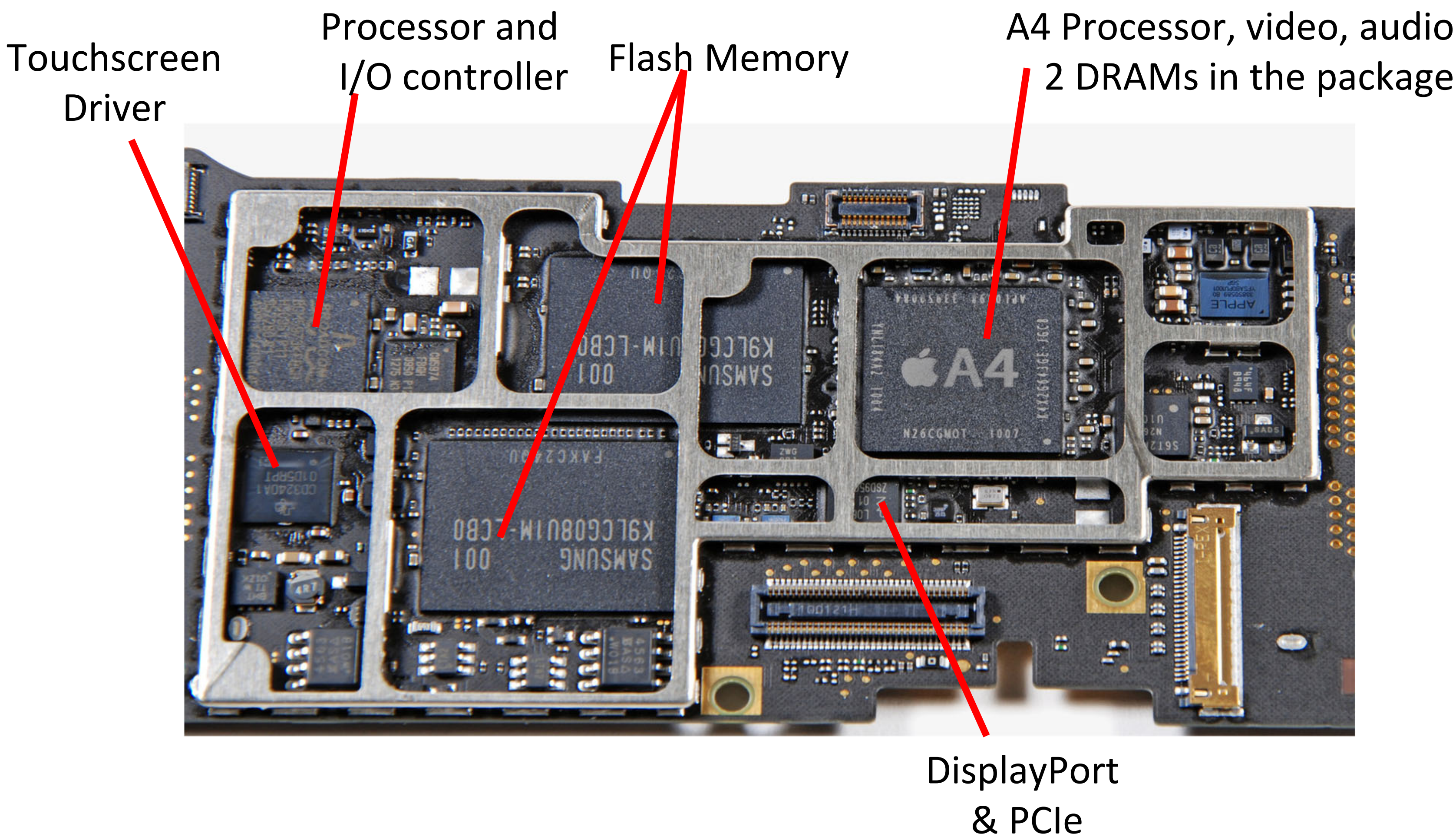
Five Components of a Computer

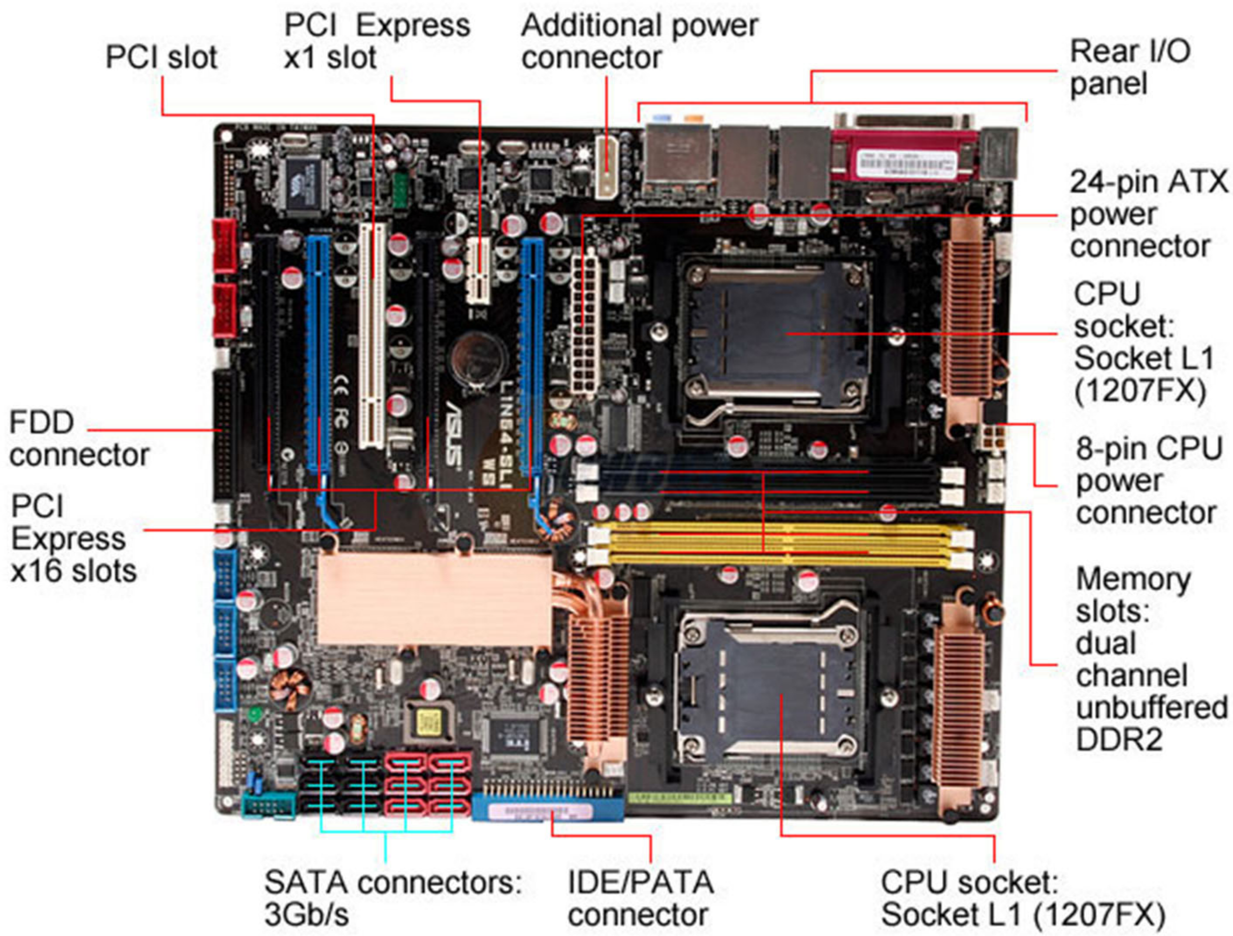
- Any computer, no matter how primitive or advanced, can be divided into five parts:



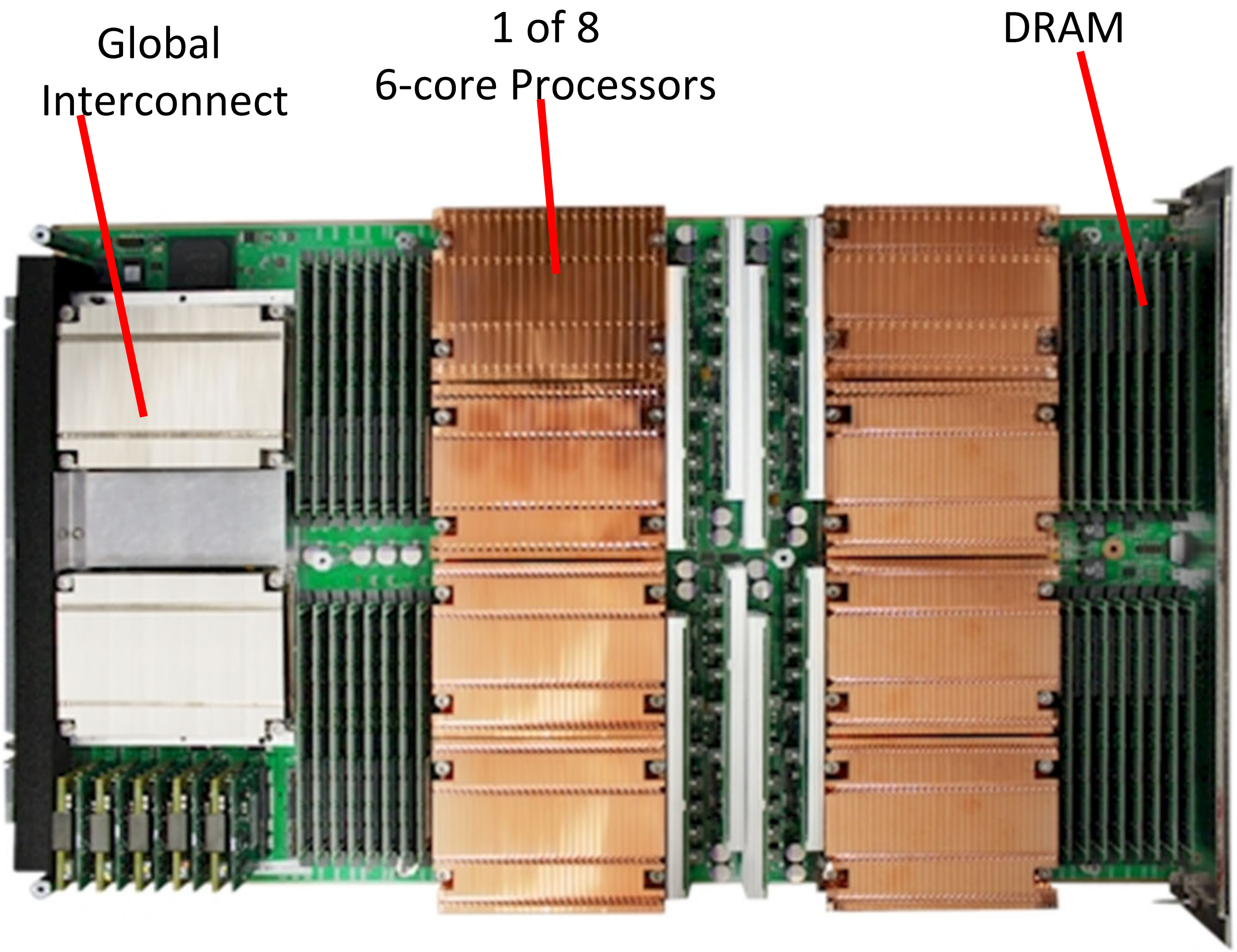
First Example: iPad



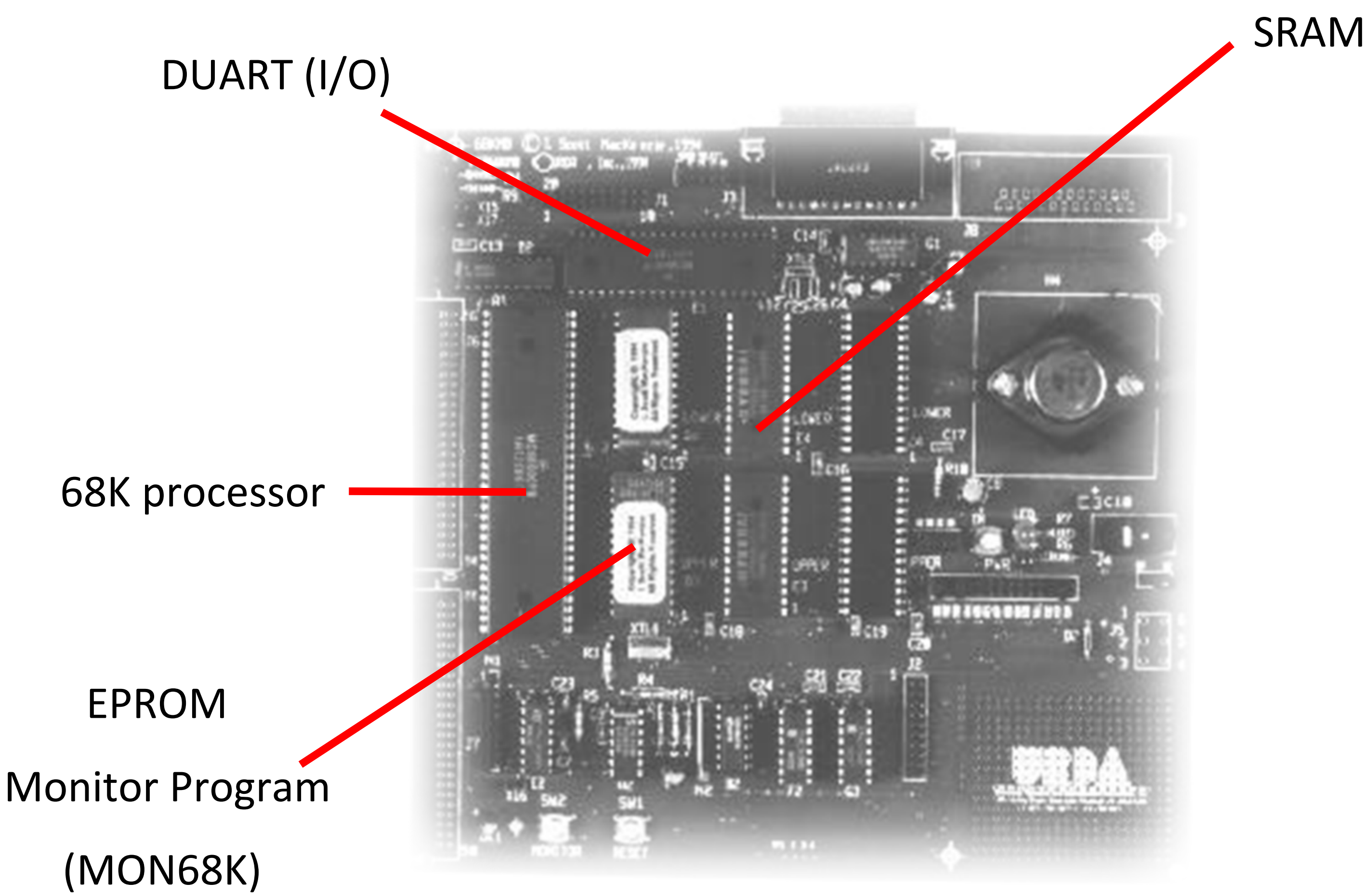
Second Example: x86 Motherboard



Third Example: Cray XT6 Supercomputer Blade



Fourth Example: 68000 Mini-board (68KMB)



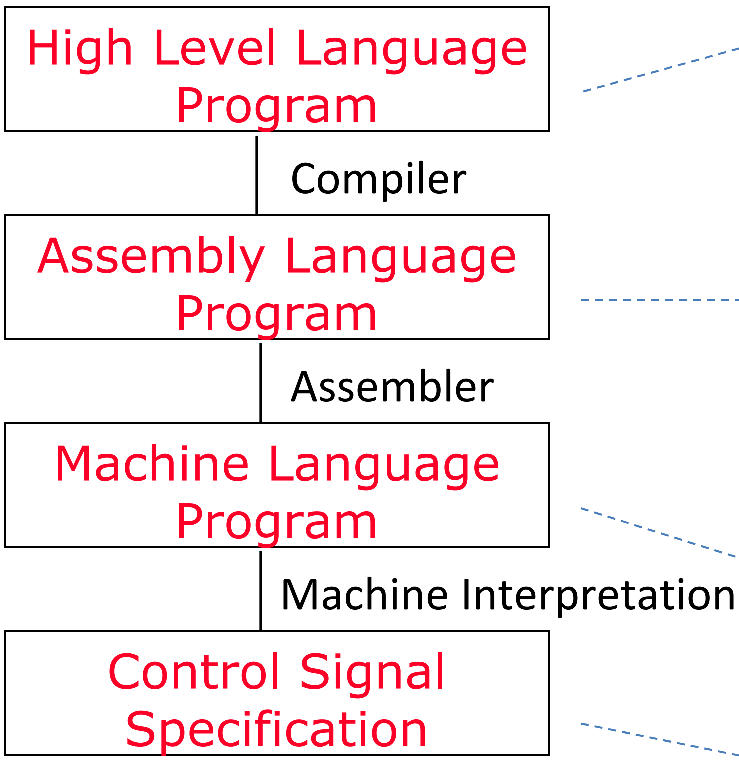
Computers Are Pretty Simple – Huh?

- Computers only work with binary signals
  - Signal on a wire is either 0 or 1
    - Usually called a “bit”
  - More complex stuff (numbers, characters, strings, pictures)
    - Must be build from multiple bits

- Built out of simple logic gates that perform Boolean logic
  - AND, OR, NOT, Adders, Comparators, etc.
- And memory cells that preserve bits over time
  - Flip-flops, registers, SRAM cells, DRAM cells, Flash, etc.

- To get hardware to do anything, need to break it down to bits
  - Strings of bits that tell the hardware what to do are called
    - instructions
  - A sequence of instructions is called
    - machine code or machine-language program

Running An Application




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

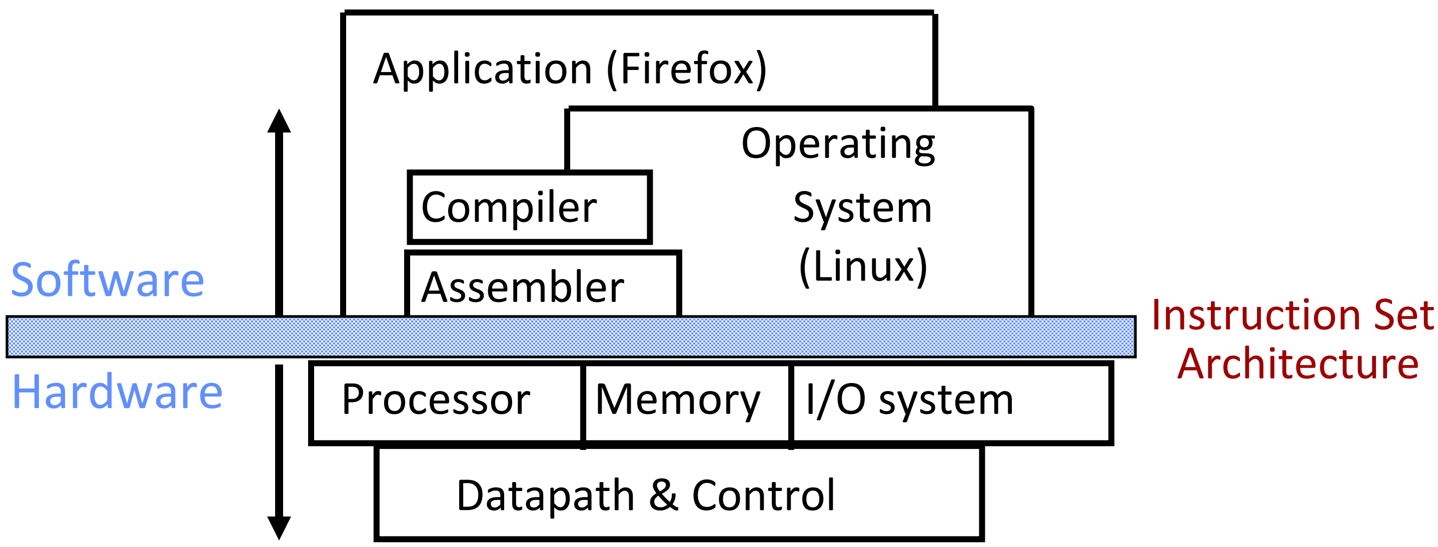
```
move.l   0(a0),d0
move.l   4(a0),d1
move.l   d1,0(a0)
move.l   d0,4(a0)
```

```
0010 0000 0010 1000 0000 0000 0000 0000
0010 0010 0010 1000 0000 0000 0000 0100
0010 0001 0100 0001 0000 0000 0000 0000
0010 0001 0100 0000 0000 0000 0000 0100
```

High/Low on control lines

Instruction-Set Architecture (ISA)

- Instruction-Set Architecture
  - Important abstraction
  - Interface between hardware and lowest-level software
  - Portion of computer visible to programmer
    - Assembly language, programmer’s model, instructions/format

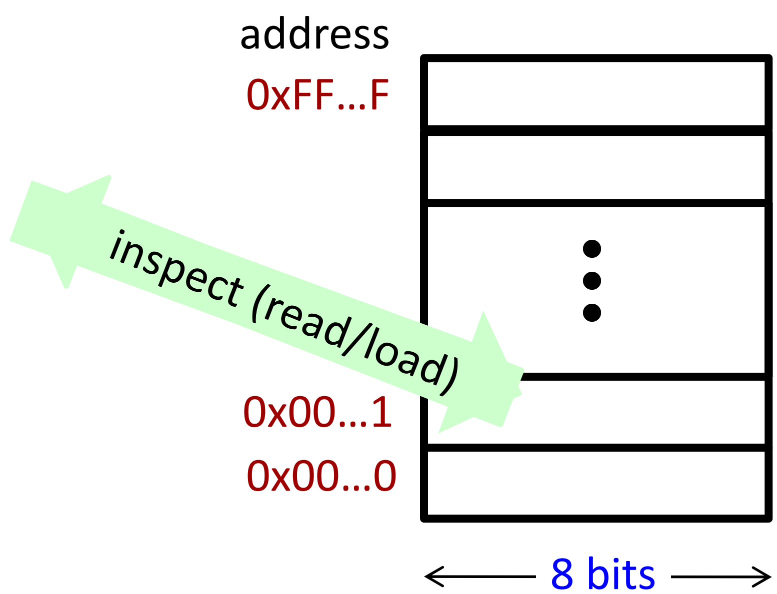


Instruction-Set Architecture (ISA) Examples

- ARM, MIPS, Intel IA32 (x86), Sun SPARC, PowerPC, IBM 390, Intel IA64, M68000, etc.
  - These are all ISAs
- Many different implementations can implement the same ISA (family)
  - 8086, 386, 486, Pentium, Pentium II, Pentium IV implement IA32
  - Of course they continue to extend it, while maintaining binary compatibility
- ISAs last a long time
  - x86 has been in use since the 70s
  - IBM 390 stared as IBM 360 in 60s
  - Stable interface between software and hardware
- Micro-architecture
  - Processor design techniques used to implement the ISA

A First Look at Memory

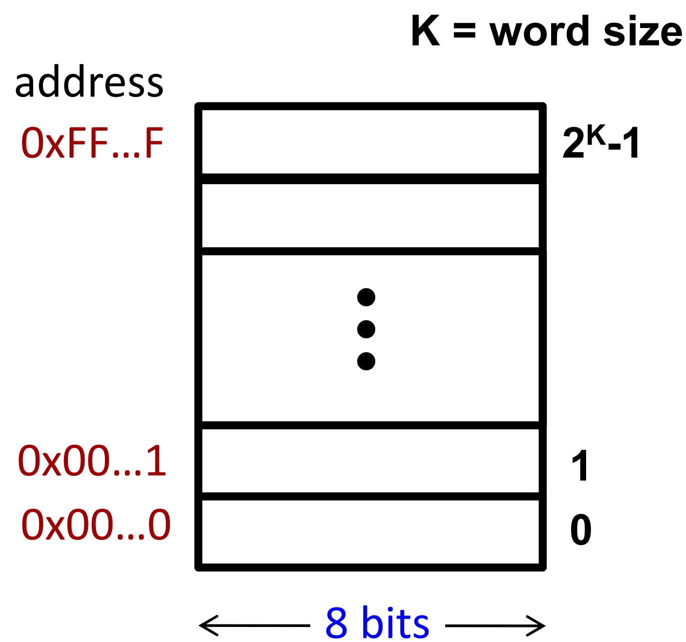
- Memory holds both <u>instructions</u> and <u>data</u>
- Logically organized as an array of locations, each with an address



A First Look at Memory

- Memory holds both <u>instructions</u> and <u>data</u>
- Logically organized as an array of locations, each with an address

- Process
  - Running program + State
  - Each process "sees" full address space due to an abstraction, called virtual memory
- Compiler + run-time system
  - Control memory allocation



Word Size of Processor

- Nominal size of integer-valued data
  - Includes addresses used to index memory
- Until recently, most machines used 32-bit (4-byte) words
  - Limits addresses to 4 Gigabytes
  - Become too small for memory-intensive applications
- Most current machines use 64-bit (8-byte) words
  - Potential address space: $2^{64} \approx 1.8 \times 10^{19}$ bytes (16 exabytes)
- For backward-compatibility many processors support different word sizes
  - Always a power-of-2 in the number of bytes: 1,2,3,4,5,6, …

C Data Types

- Typical size of C objects in bytes

Data Object	32-bit Machine	64-bit Machine
char	1 byte	1 byte
int	4 bytes	4 bytes
short int	2 bytes	2 bytes
long int	4 bytes	8 bytes
long long	8 bytes	8 bytes
float	4 bytes	4 bytes
double	8 bytes	8 bytes
long double	8 bytes	16 bytes
pointer	4 bytes	8 bytes

Word-Oriented View of Memory

- Addresses specify locations of bytes in memory
  - Word address is address of first byte in word
  - Address of successive words differ by 4 (32-bit) or 8 (64-bit)

				0x01C
				0x018
				0x014
				0x010
Byte 15	Byte 14	Byte 13	Byte 12	0x00C
				0x008
				0x004
Byte 3	Byte 2	Byte 1	Byte 0	0x000

K=32-bits

Byte 15	Byte 14	0x00E
Byte 13	Byte 12	0x00C
		0x00A
		0x008
		0x006
		0x004
Byte 3	Byte 2	0x002
Byte 1	Byte 0	0x000

K=16-bits

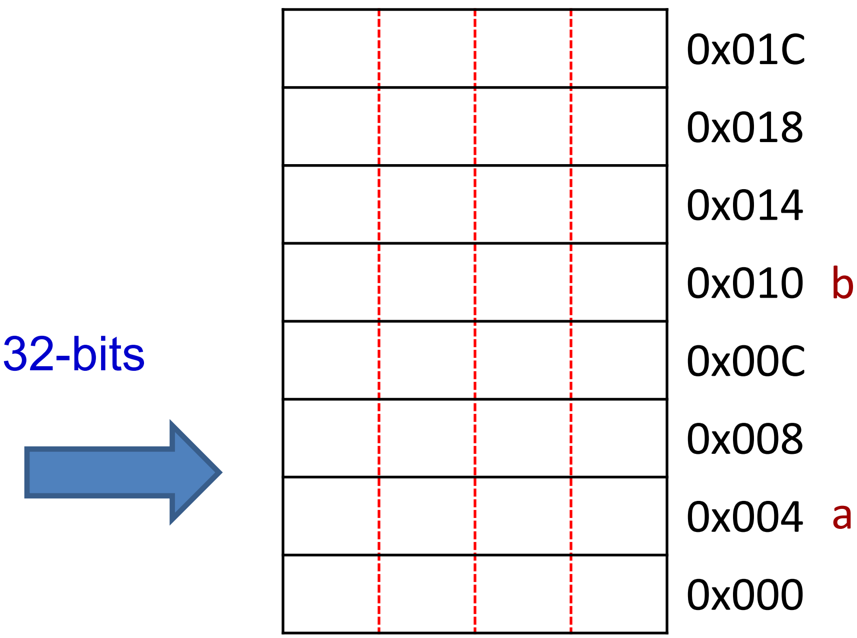
Byte 15	0x00F
Byte 14	0x00E
Byte 13	0x00D
Byte 12	0x00C
	0x00B
	0x00A
	0x009
	0x008
	0x007
	0x006
	0x005
	0x004
Byte 3	0x003
Byte 2	0x002
Byte 1	0x001
Byte 0	0x000

K=8-bits

Variable Declarations and Memory

- Variable declarations
  - int a, b;
  - find two locations in memory in which to store two integers (1 word each)

- Memory Layout
  - Assume word size of 32-bits



Variable Assignments and Memory

- Assignment (LHS = RHS)
  - int a = 0, b = -1;
  - LHS must evaluate to a memory location (a variable)
  - RHS must evaluate to a value (possibly an address)

				0x01C	
				0x018	
				0x014	
FF	FF	FF	FF	0x010	b
				0x00C	
				0x008	
00	00	00	00	0x004	a
				0x000	

Variable Assignments and Memory

- Assignment (LHS = RHS)
  - int a = 0, b = -1;
  - LHS must evaluate to a memory location (a variable)
  - RHS must evaluate to a value (possibly an address)

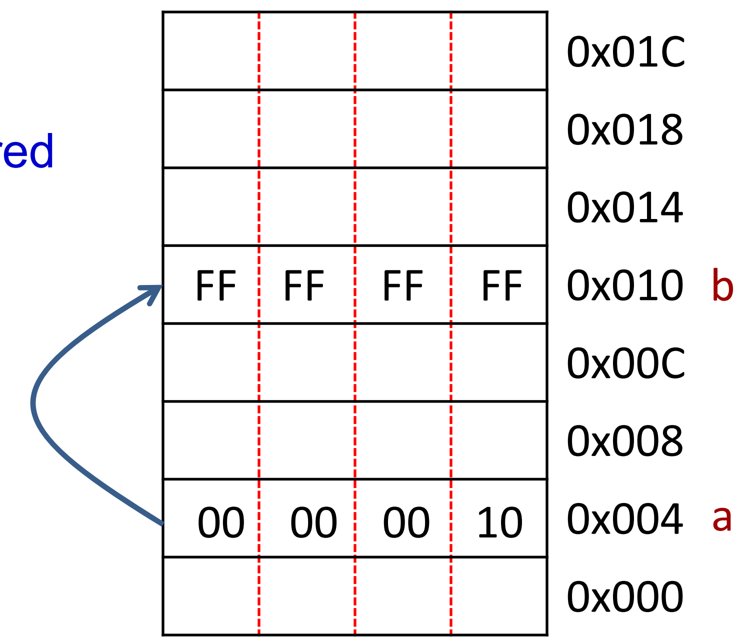
- Example
  - a = a + b;

				0x01C	
				0x018	
				0x014	
FF	FF	FF	FF	0x010	b
				0x00C	
				0x008	
00 / FF	00 / FF	00 / FF	00 / FF	0x004	a
				0x000	

Pointers and Memory

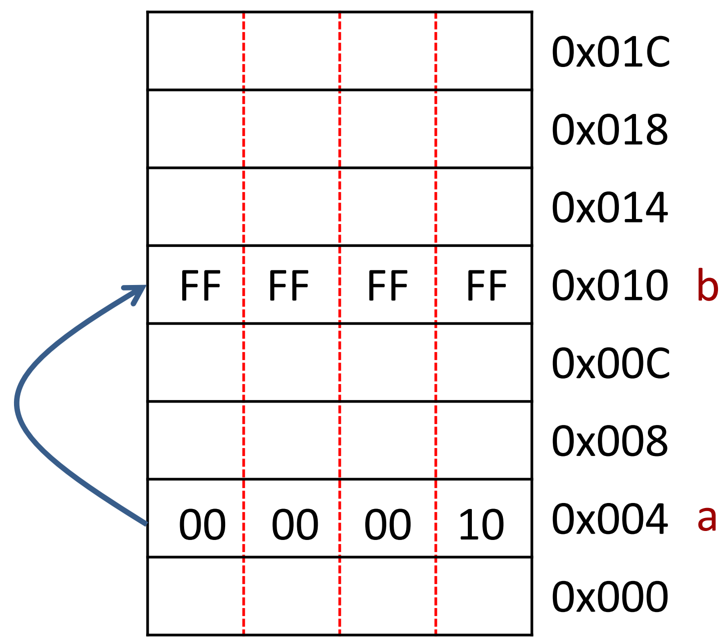
- Pointer declaration
  - int *a;
  - declares a variable a that is a pointer to an integer data item

- Pointer assignment
  - a = &b;
  - assigns a the address where b is stored



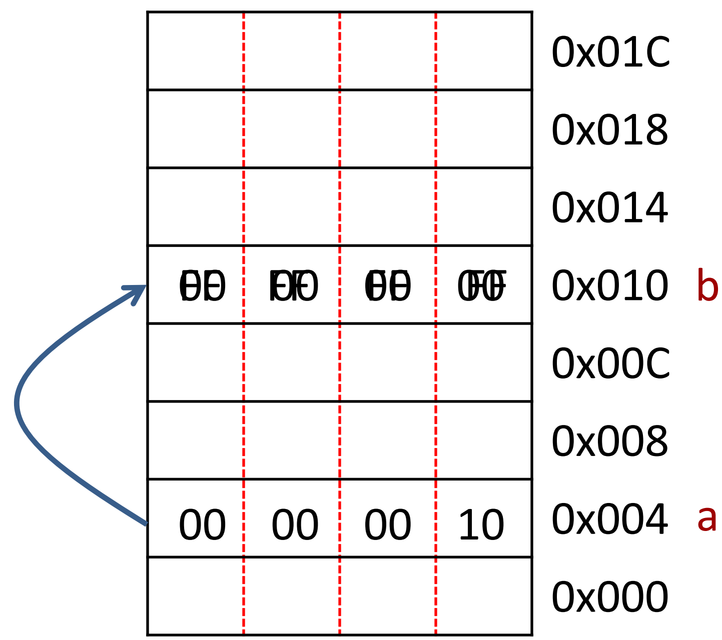
Pointers and Memory

- Pointer dereferencing
  - *a;
  - the value at the memory address given by the value of *a



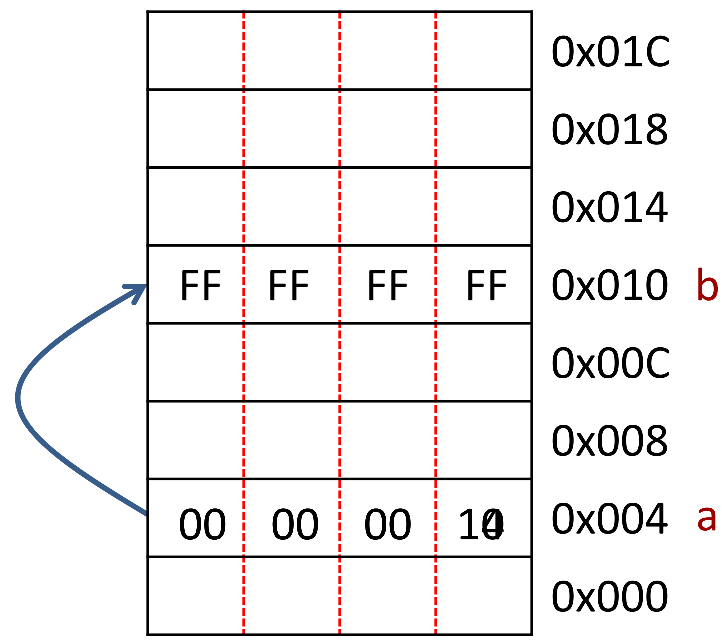
Pointers and Memory

- Pointer dereferencing
  - *a;
  - the value at the memory address given by the value of *a
- Example
  - b = *a + 1;



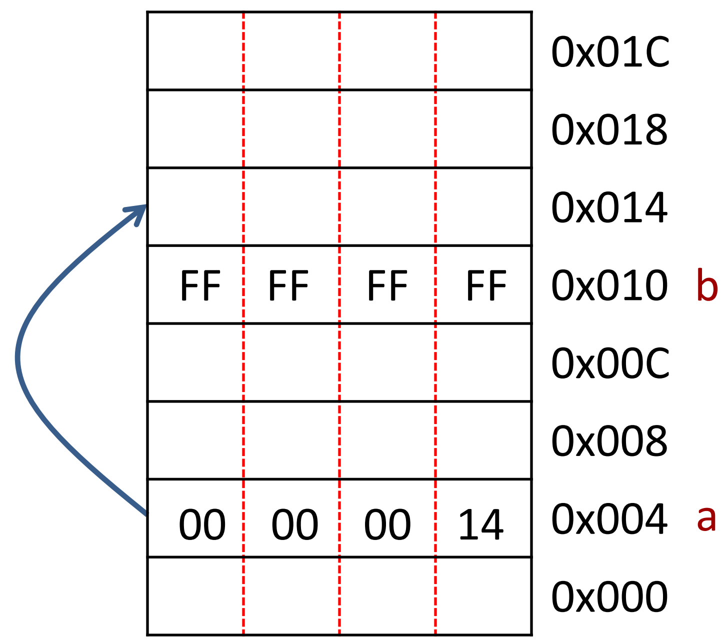
Pointers and Memory

- Pointer arithmetic
  - a = a + 1;
  - the value of the pointer is incremented to point at the next data object (integer) in memory



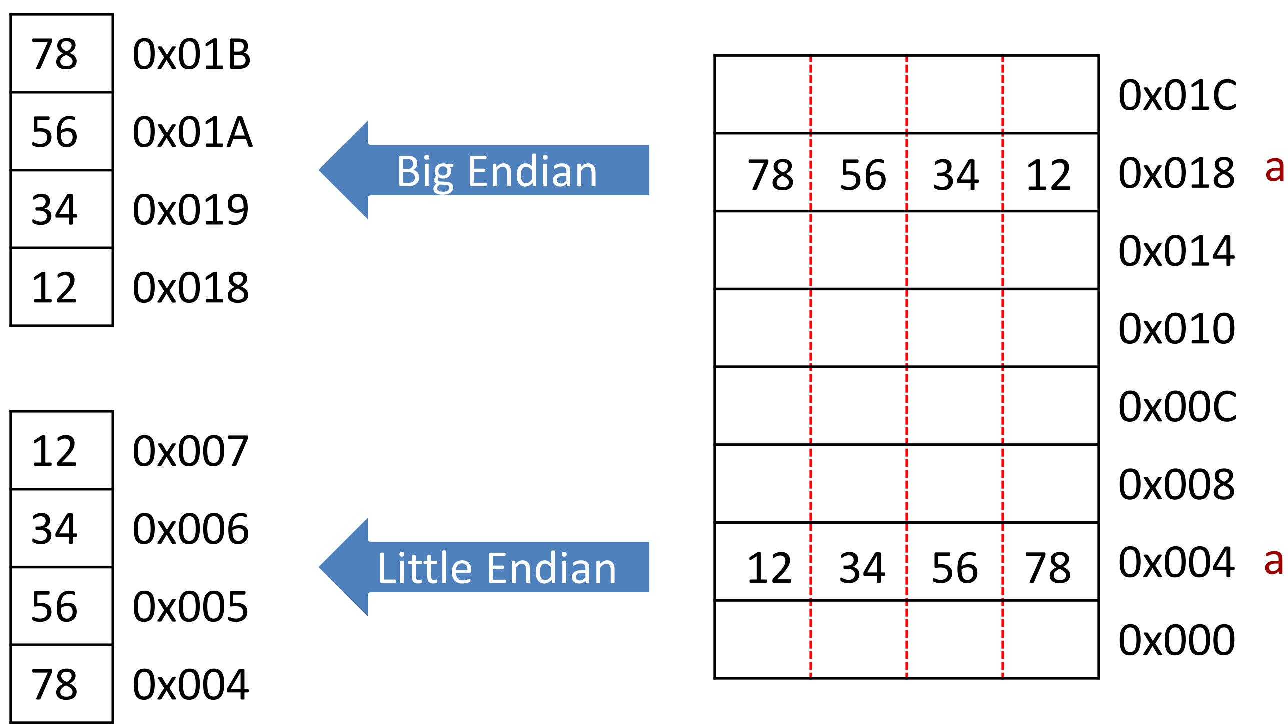
Pointers and Memory

- Pointer arithmetic
  - a = a + 1;
  - the value of the pointer is incremented to point at the next data object (integer) in memory



Byte Ordering

- How do we order the bytes in a word?
  - int a = 0x12345678;



Endianness: Big or Little?

- Big Endian
  - Address of most-significant byte = (lowest) address of word
  - IBM 360, M68000, MIPS, SPARC
- Little Endian
  - Address of least-significant byte = (lowest) address of word
  - Intel x86, ARM, DEC Vax & Alpha

When Endianness Matters

- When you
  - store words then load bytes

```c
#include <stdio.h>

int main() {
int  fourBytes = 0x12345678;
char oneByte  = *(char*) &fourBytes;
if (oneByte == 0x78)
    puts("little endian");
else
   puts("big endian");

return 0;
}
```

Big Endian

	78	0x007
	56	0x006
	34	0x005
fourBytes	12	0x004

Little Endian

	12	0x007
	34	0x006
	56	0x005
fourBytes	78	0x004

When Endianness Matters

- When you
  - share data between machines with different endianness
  - When moving between 32-bit and 64-bit word sizes

x86-32 (Little Endian)

12	0x007
34	0x006
56	0x005
78	0x004

X86-64 (Little Endian)

00	0x00B
00	0x00A
00	0x009
00	0x008
12	0x007
34	0x006
56	0x005
78	0x004

32-bit (Big Endian)

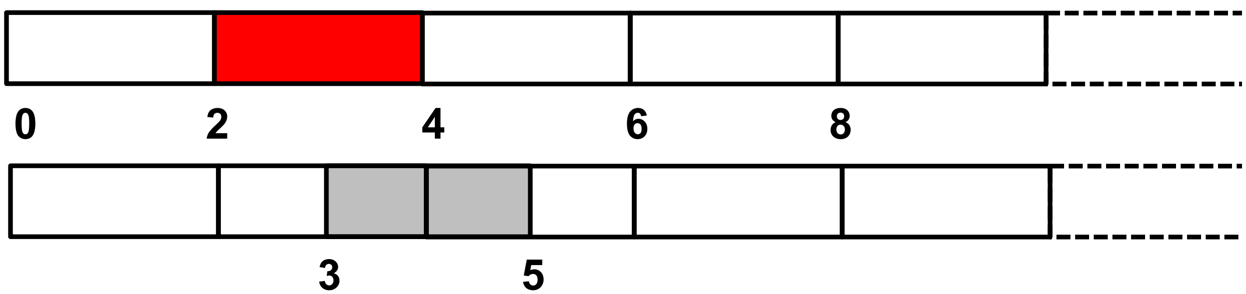
78	0x007
56	0x006
34	0x005
12	0x004

64-bit (Big Endian)

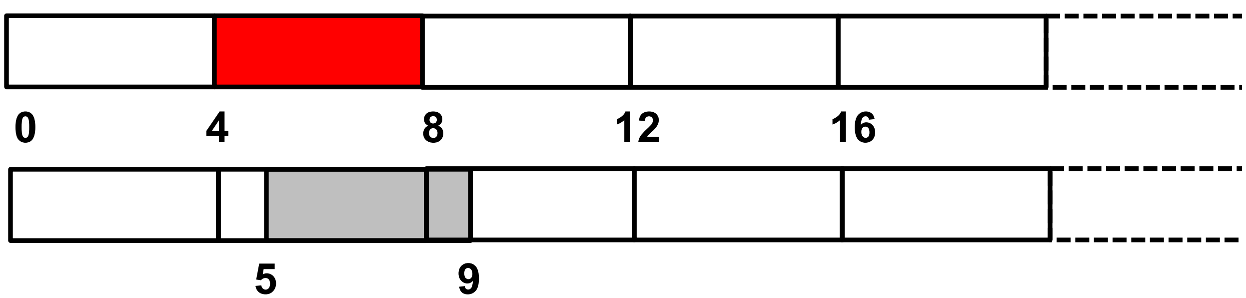
78	0x00B
56	0x00A
34	0x009
12	0x008
00	0x007
00	0x006
00	0x005
00	0x004

Alignment Issues

- Some ISAs require words to be at an <u>even</u> addresses
  - Consider the following word (2-byte) access



  - Consider the following word (4-byte) access



C Example

What is the size of this structure?

```
struct foo {
    char   sm;      /* 1 byte  */
    short  med;     /* 2 bytes */
    char   sm1;     /* 1 byte  */
    int    lrg;     /* 4 bytes */
}
```

Byte offset	0	1	2	3	4	5	6	7	8	9
	sm	X	med		sm1	X	lrg			

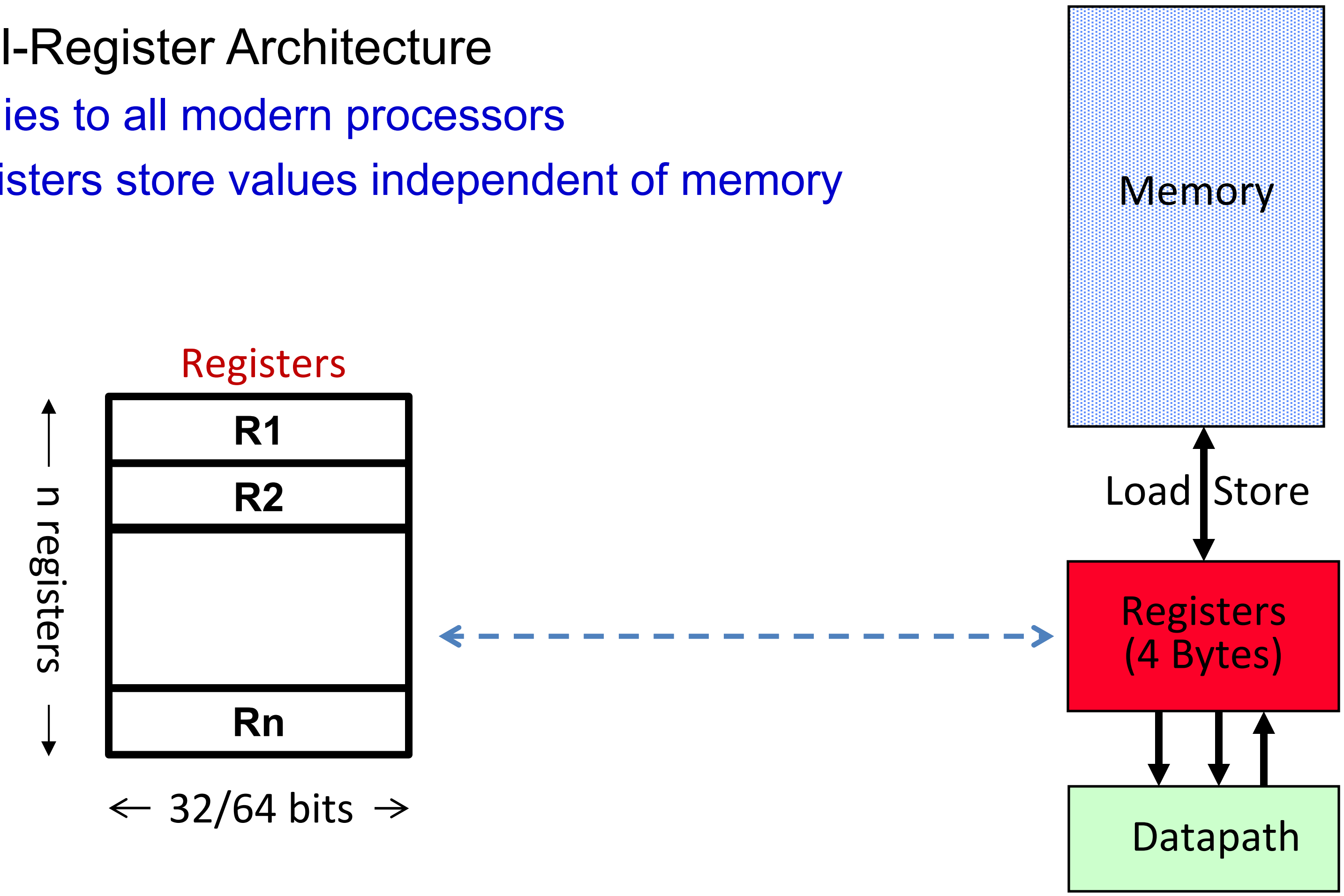
- Historically
  - Early machines (IBM 360, circa 1964) required alignment
  - Removed in the 1970s to reduce impact on programmers (IBM 370, x86)
  - Reintroduced by RISC to improve performance
  - Removed by some RISCs to simplify software

Today

- Major Components of Computer
- Instruction-Set Architecture (ISA)
- Memory
- Registers
- Instruction Cycle

A First Look at Registers

- General-Register Architecture
  - Applies to all modern processors
  - Registers store values independent of memory

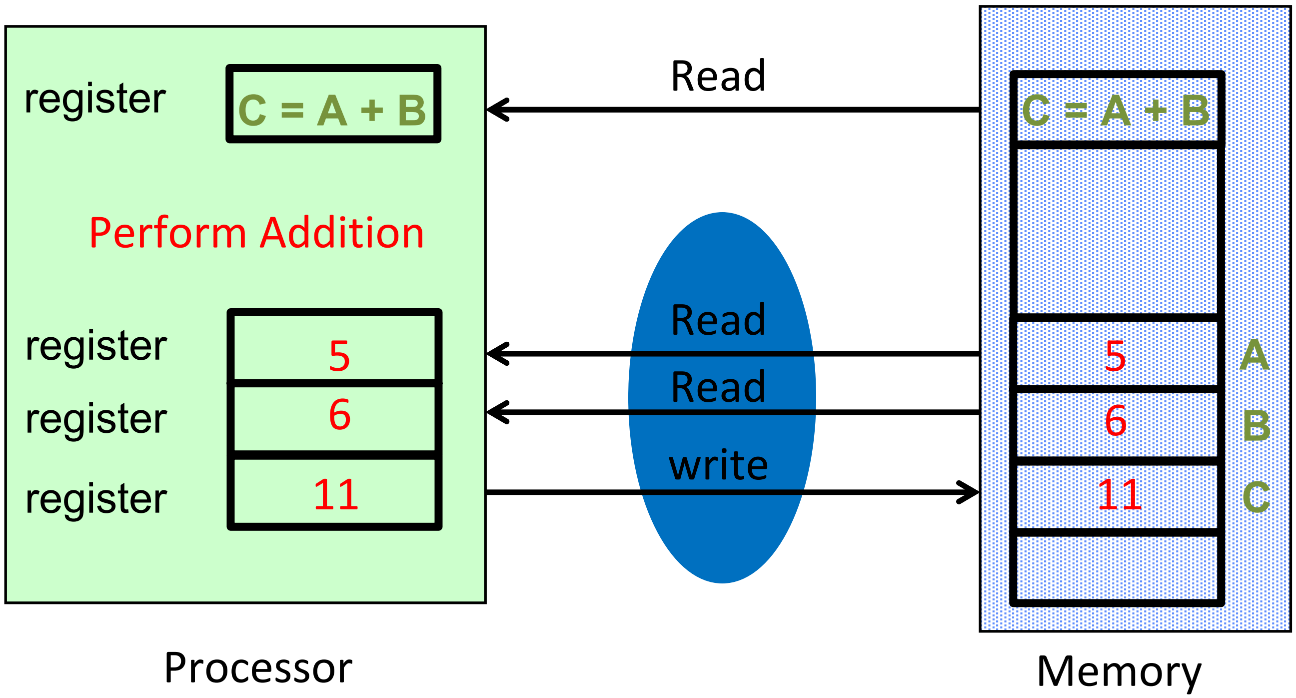


Today

- Major Components of Computer
- Instruction-Set Architecture (ISA)
- Memory
- Registers
- Instruction Cycle

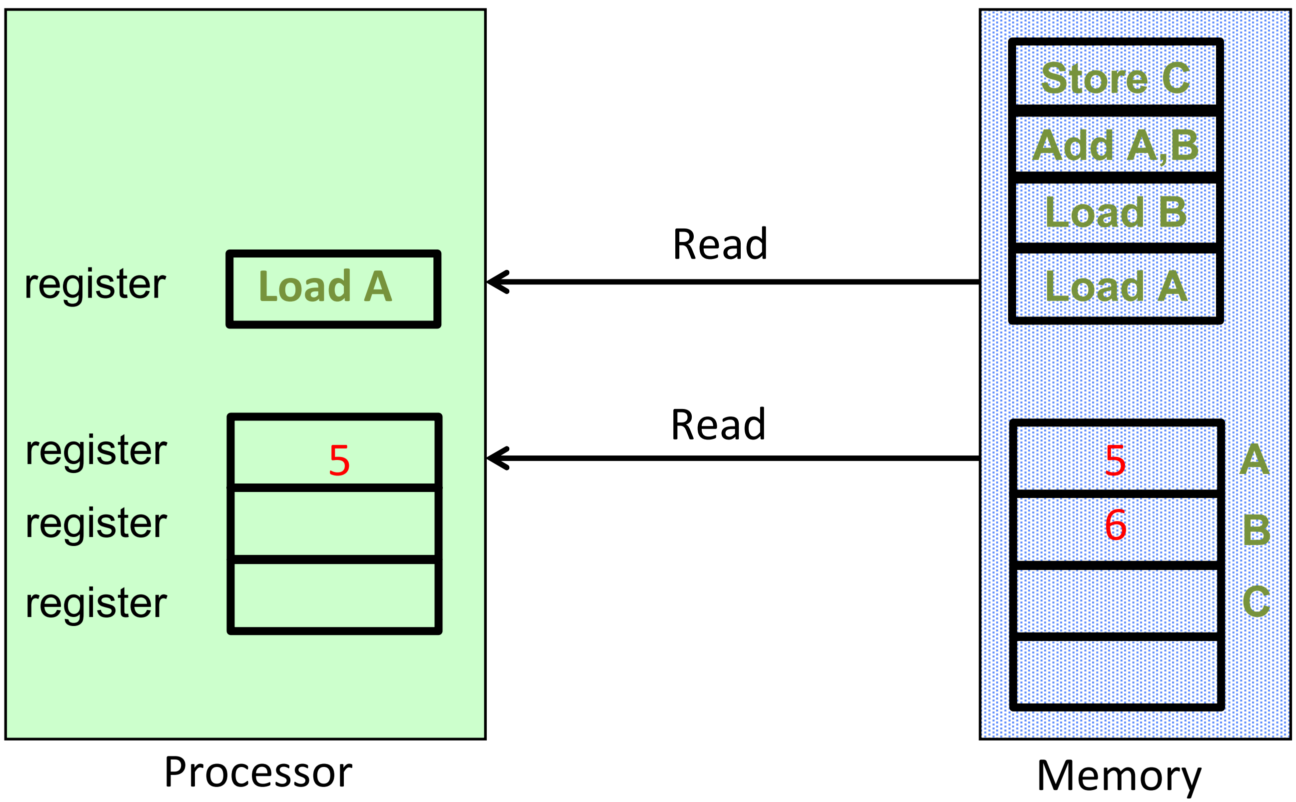
A First Look at the Processor

- Responsible for reading program instructions from memory and executing them



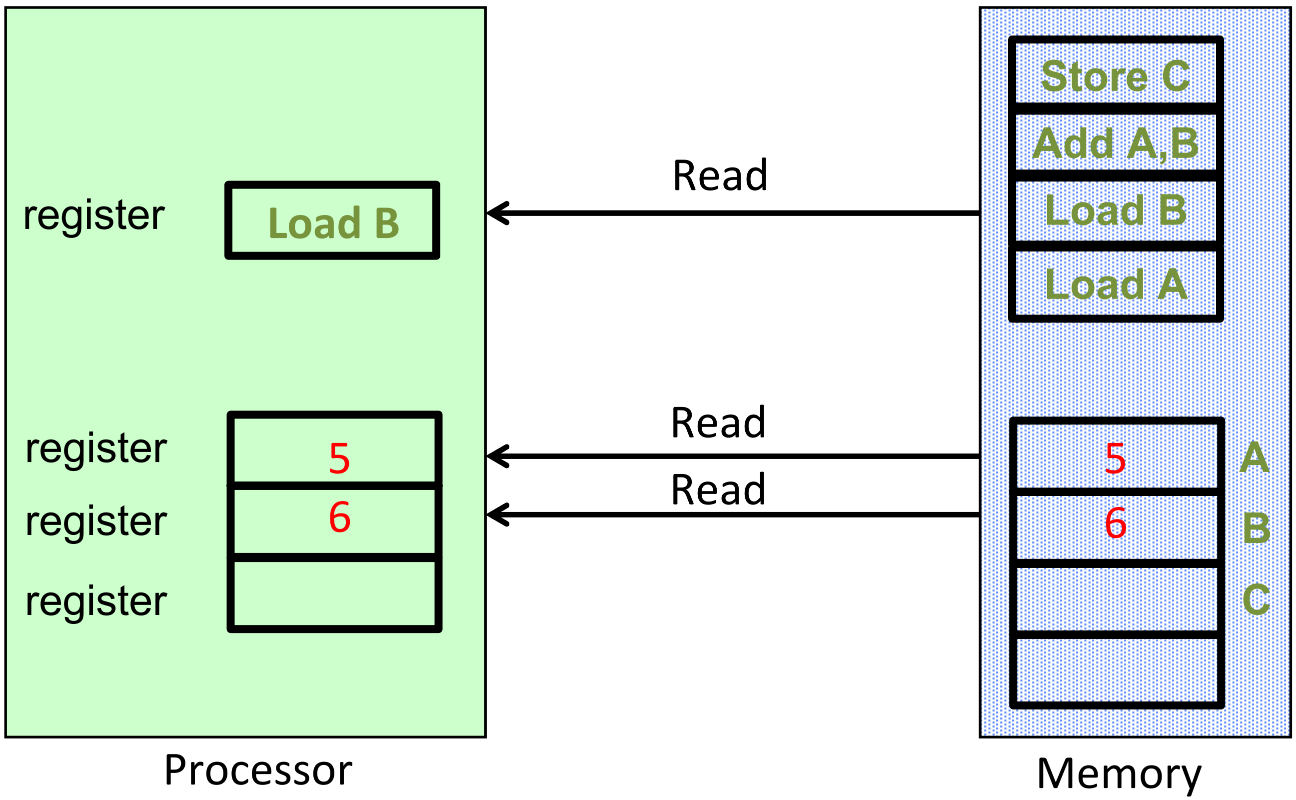
Instruction Granularity

- A statement such as C = A + B is implemented as a sequence of primitive machine instructions that access memory at most once



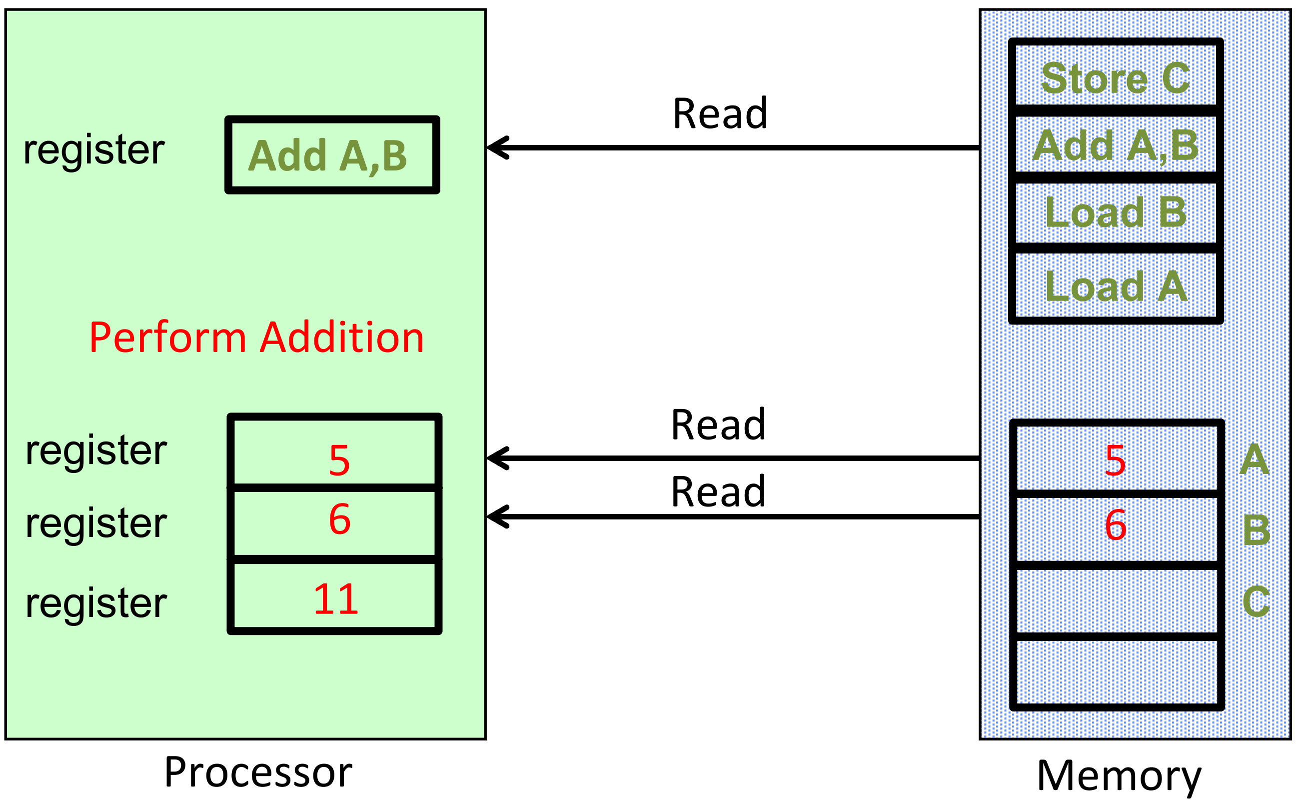
Instruction Granularity

- A statement such as C = A + B is implemented as a sequence of primitive machine instructions that access memory at most once



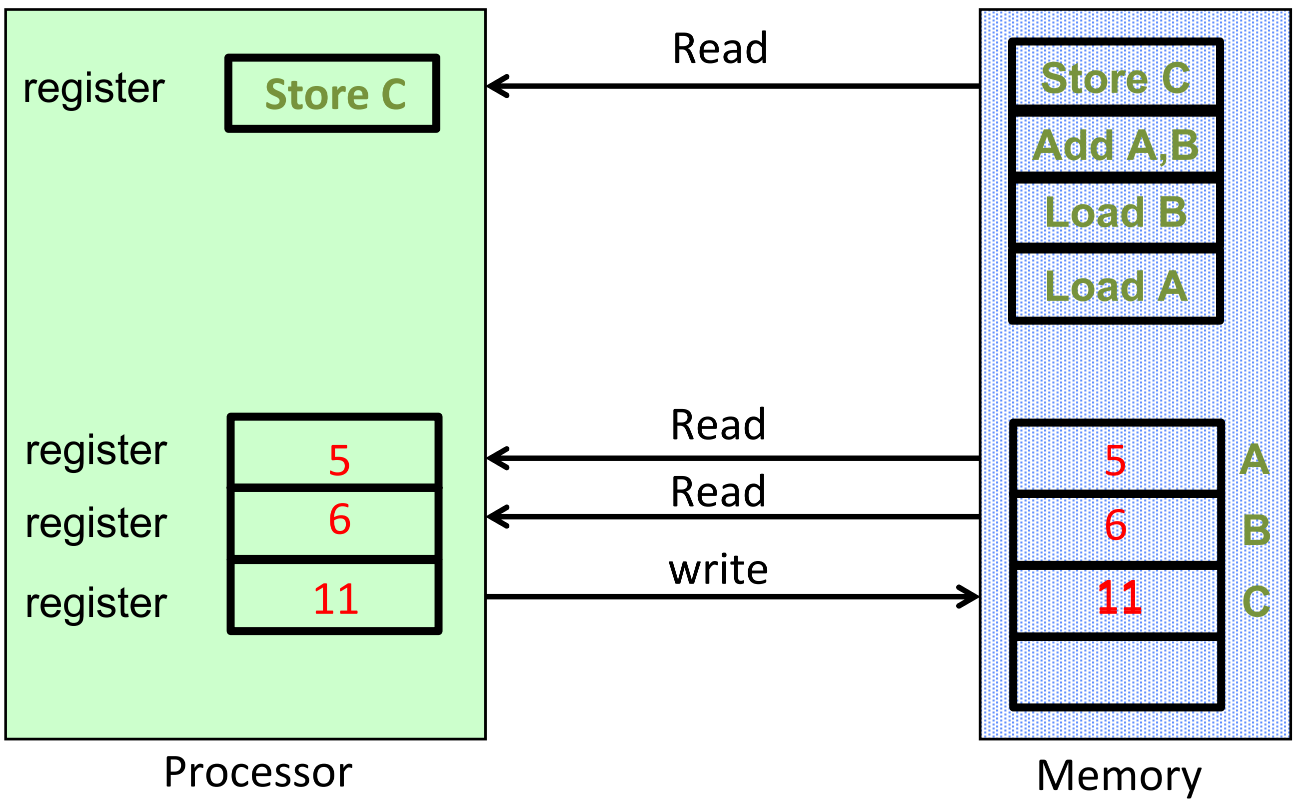
Instruction Granularity

- A statement such as C = A + B is implemented as a sequence of primitive machine instructions that access memory at most once



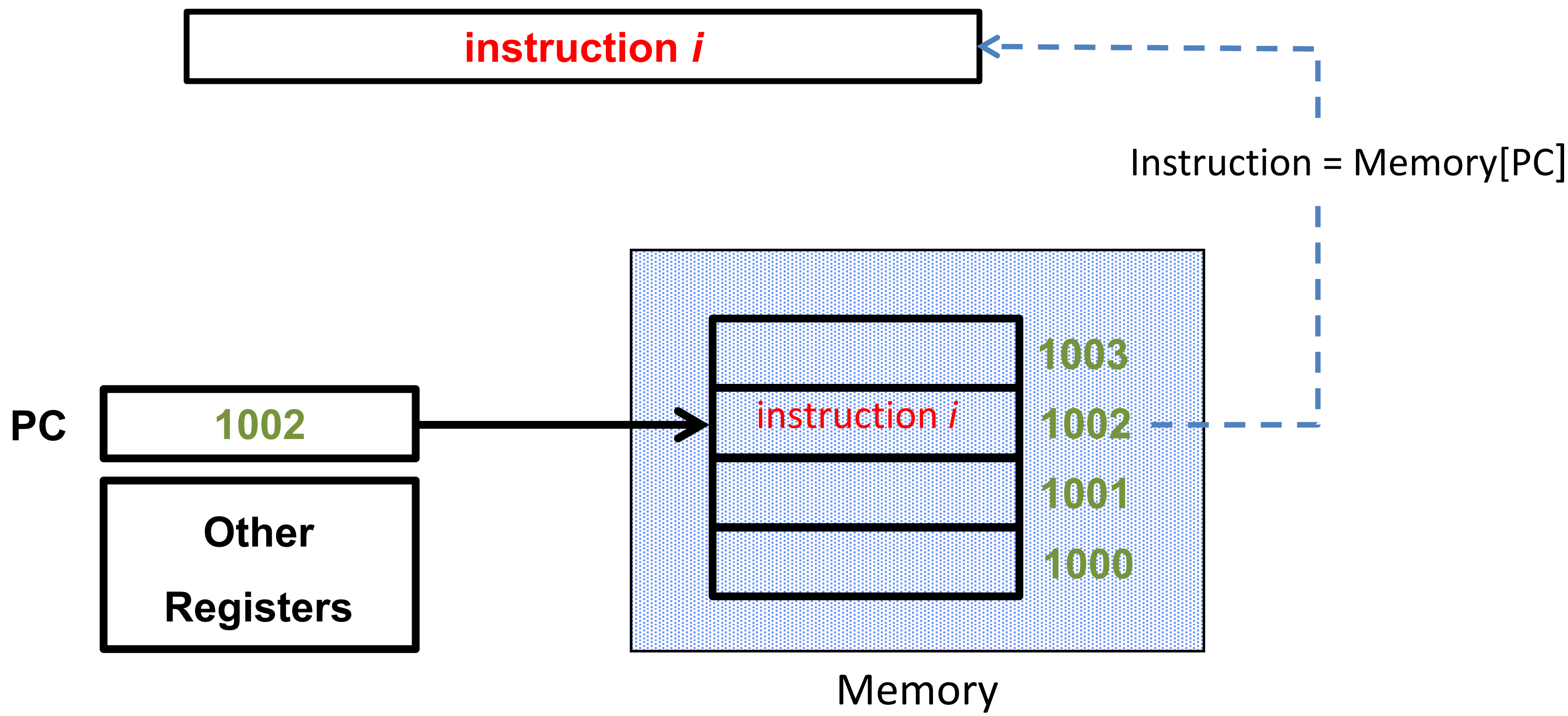
Instruction Granularity

- A statement such as C = A + B is implemented as a sequence of primitive machine instructions that access memory at most once



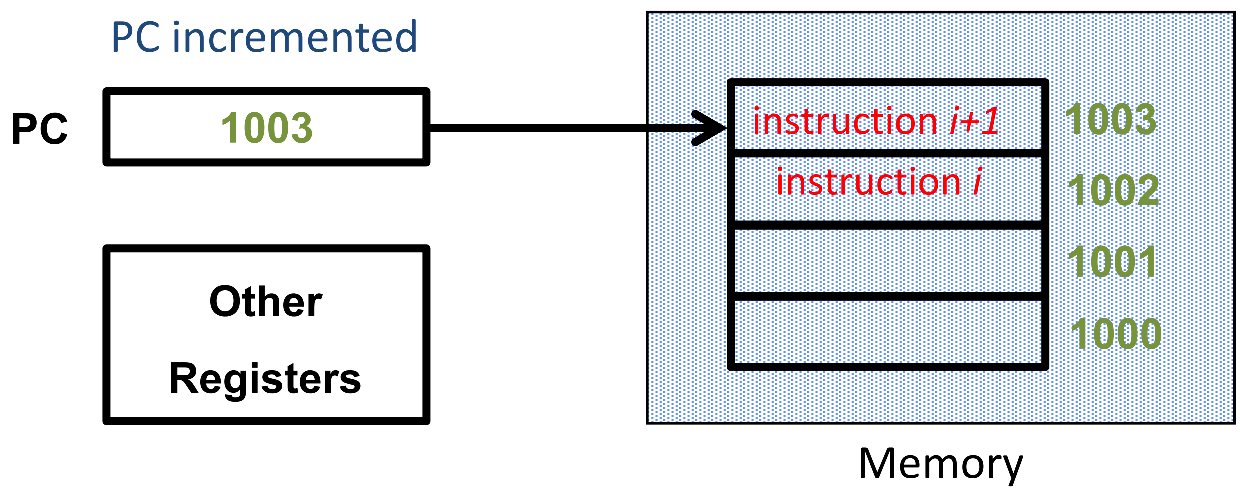
Program Counter

- PC is a <u>register</u> inside the processor
- PC holds the address of the <u>next instruction</u> to be fetched and executed



Program Counter

- PC is a register inside the processor
- PC holds the address of the next instruction to be fetched and executed
- After instruction is automatically read from memory, the PC is incremented to point to next instruction in memory



Instruction Cycle

- Set PC = address of first instruction in program
- Repeat
  - <u>Fetch</u> the instruction
    - Get an instruction from memory location contained in PC
    - Increment PC by size of instruction
  - <u>Decode</u> the instruction
    - Identify operation defined by instruction
    - If instruction requires data from memory, fetch data from memory
  - <u>Execute</u> the instruction
    - Perform operation defined by instruction
    - If instruction requires data to be stored, store data in memory
- Until (last instruction encountered)

Accessing the Program Counter

- The contents of the PC cannot be directly accessed by programs
- The content is manipulated by
  - branch, jump, and jump-to-subroutine instructions
  - exceptions (traps, interrupts)

Summary

- Instruction-Set Architecture (ISA)
  - Long-lived, stable interface between HW and SW
- Memory
  - Holds program instructions and data
  - Slow, large capacity, one-dimensional array
- Registers
  - Store values independent of memory
  - Small, but fast
- Program Counter
  - Holds address of next instruction to be read
  - Updated after instruction is read to point to next instruction
- Instruction Cycle
  - Fetch, decode, execute phases