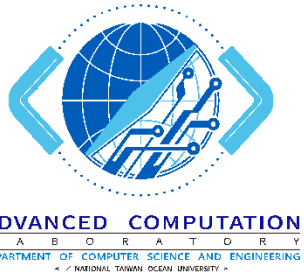


Introduction to Computer Science

William Hsu
Advanced Computation Laboratory
Department of Computer Science and Engineering
Department of Environmental Biology and Fisheries Science
National Taiwan Ocean University

Chapter 2: Data Manipulation

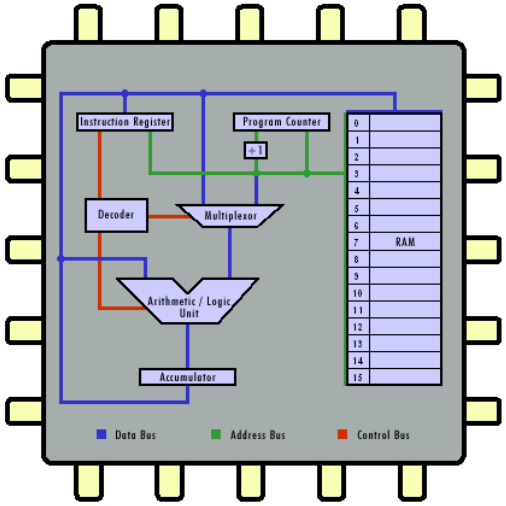
The successor to Intel's Haswell CPU technology, "Broadwell", has been revealed and shown working here at the Intel Developer Forum in San Francisco. The chip is built using a whole new production process that shrinks Haswell's 22nm transistors down to 14nm. That is tiny.

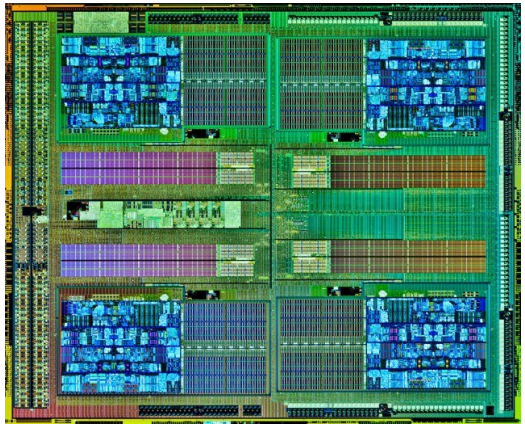



- 2.1 Computer Architecture
- 2.2 Machine Language
- 2.3 Program Execution
- 2.4 Arithmetic/Logic Instructions
- 2.5 Communicating with Other Devices
- 2.6 Program Data Manipulation
- 2.7 Other Architectures

Computer Architecture

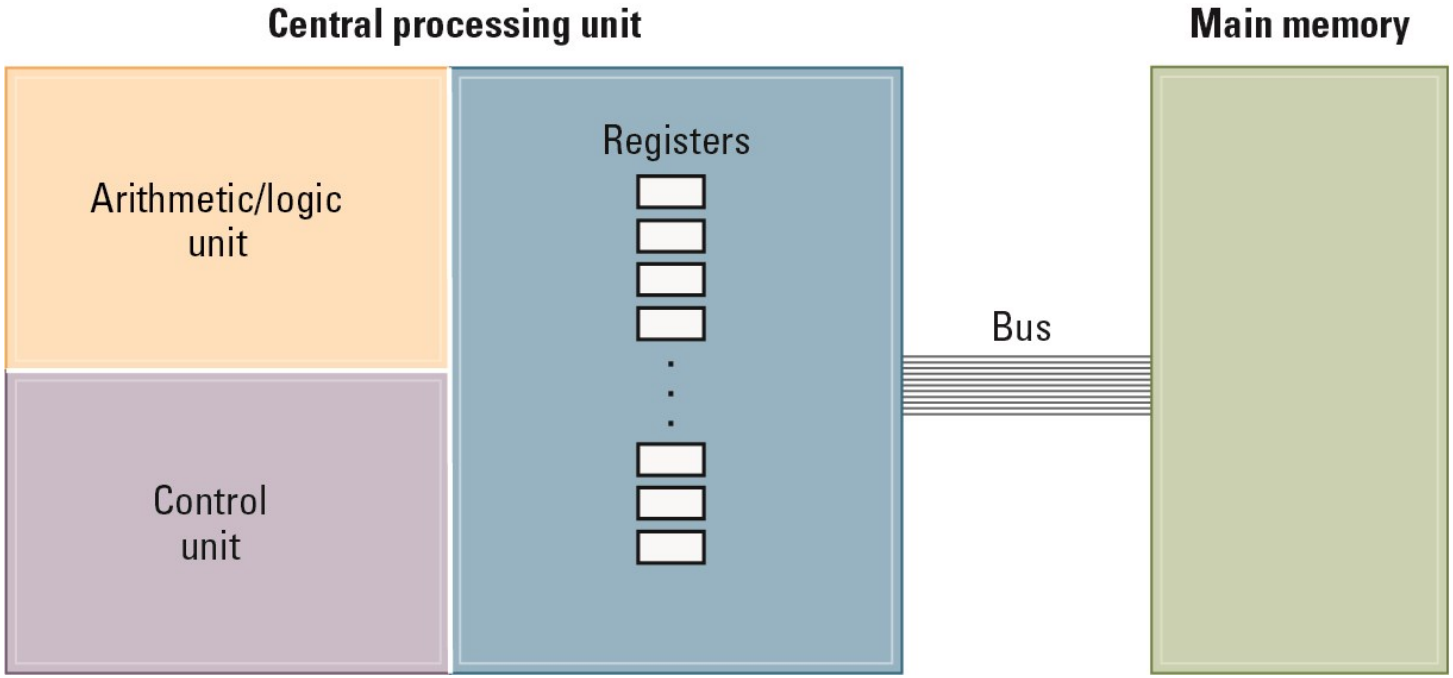
- Central Processing Unit (CPU) or processor
  - Arithmetic/Logic unit versus Control unit.
  - Registers
    - General purpose.
    - Special purpose.





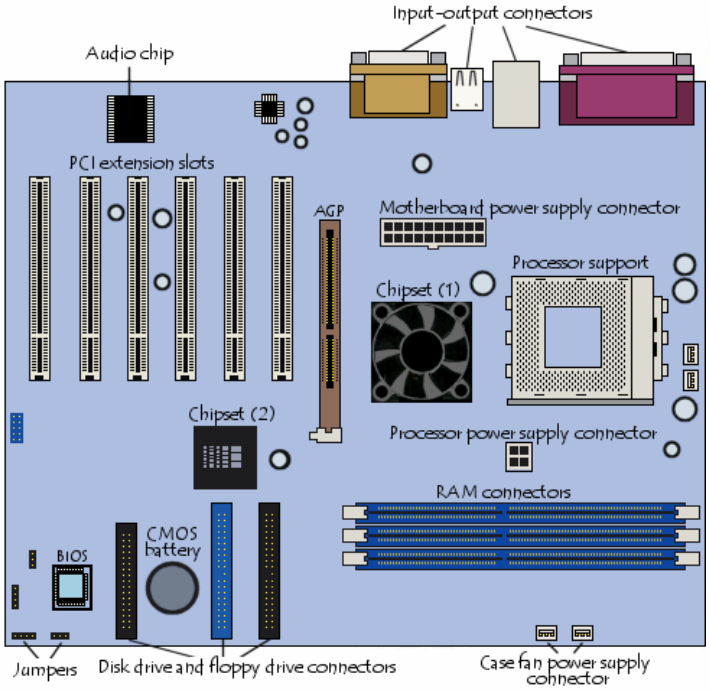
AMD's world #1 5GHz CPU.

CPU and Main Memory Connected via a Bus

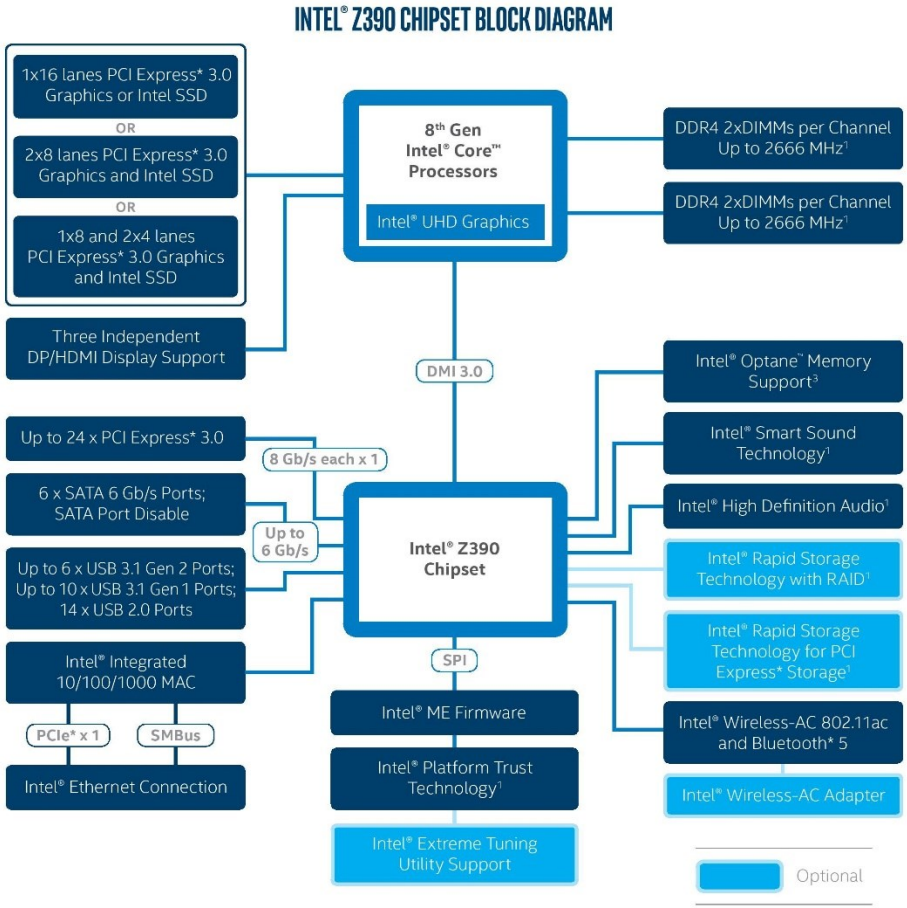


Computer Architecture

- Bus.
- Motherboard.

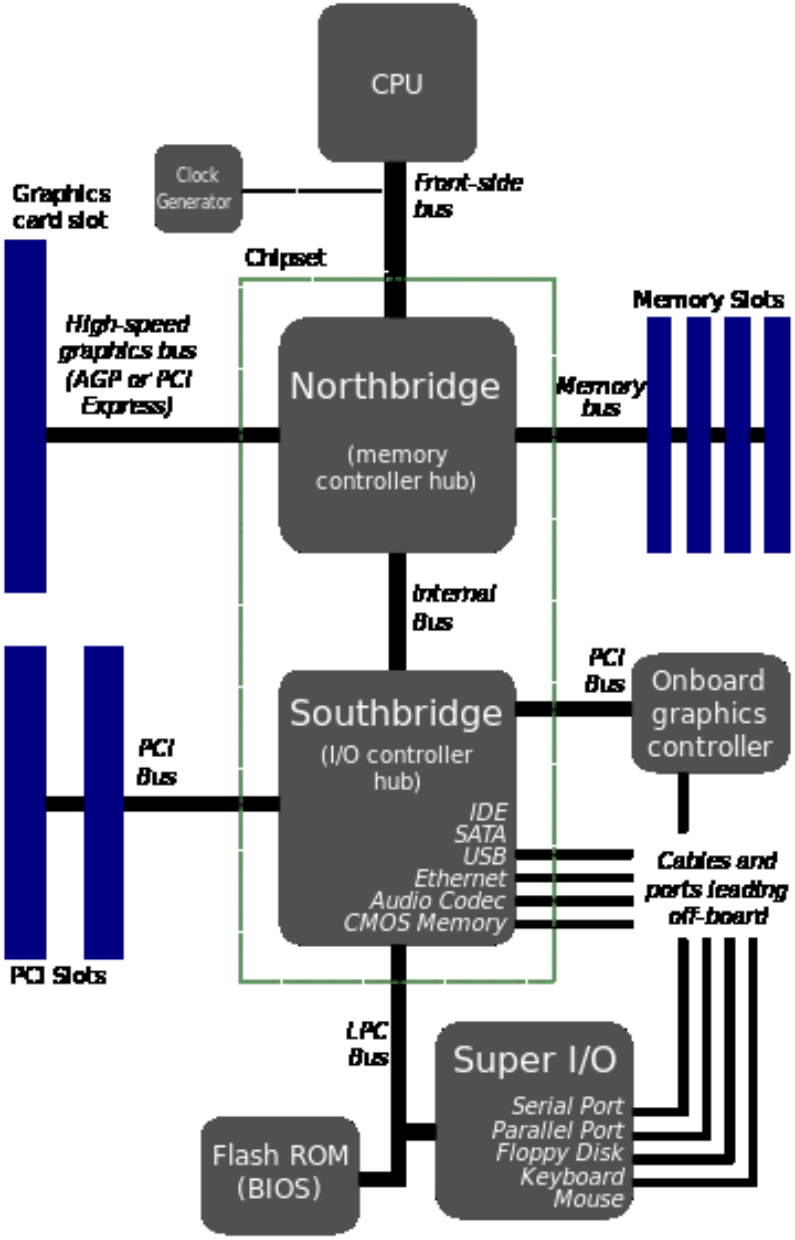


Northbridge removed

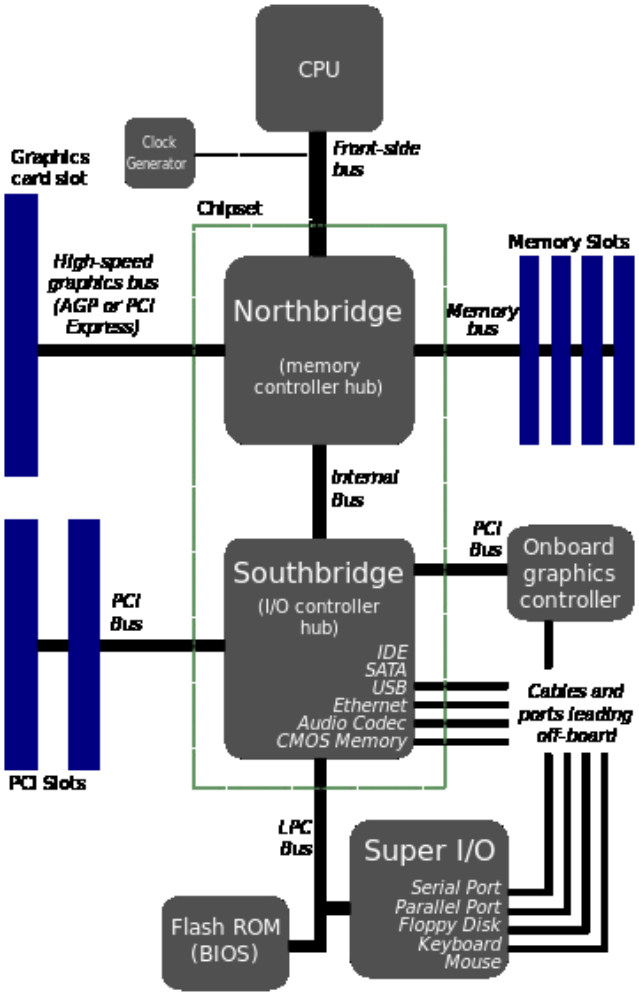


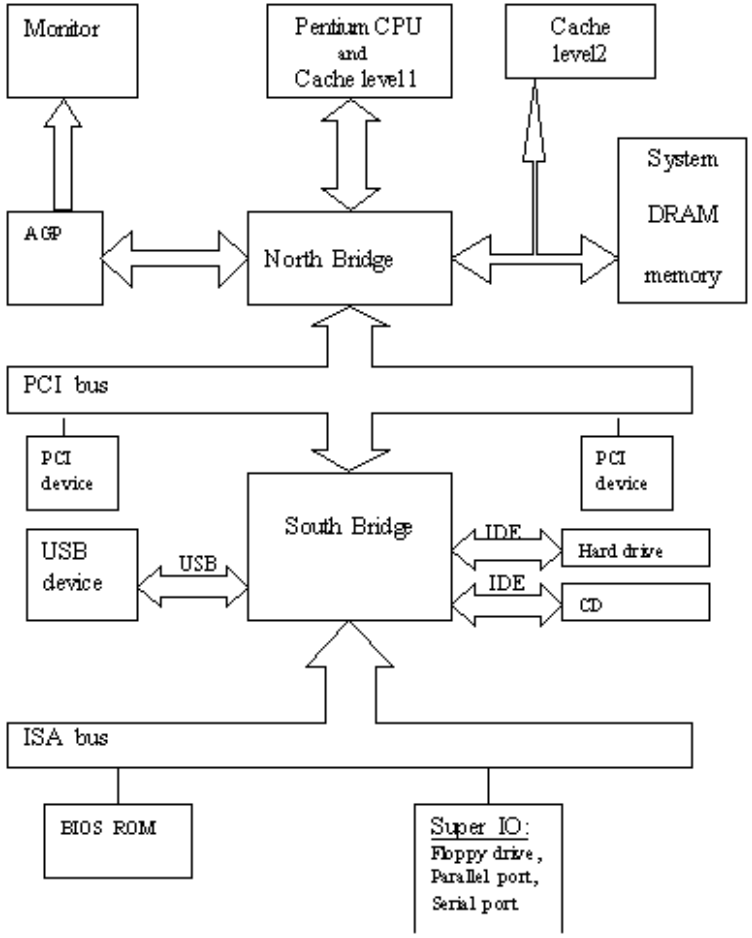
IO Bridges

- Northbridge is connected directly to the CPU via the frontside bus (FSB)
  - Responsible for tasks that require the highest performance.
- Southbridge, also known as I/O controller hub.



Block diagrams





Stored Program Concept

- A program can be encoded as bit patterns and stored in main memory.
- From there, the CPU can then extract the instructions and execute them. In turn, the program to be executed can be altered easily.

Terminology

- **Machine instruction:** An instruction (or command) encoded as a bit pattern recognizable by the CPU.
- **Machine language:** The set of all instructions recognized by a machine.

Machine Language Philosophies

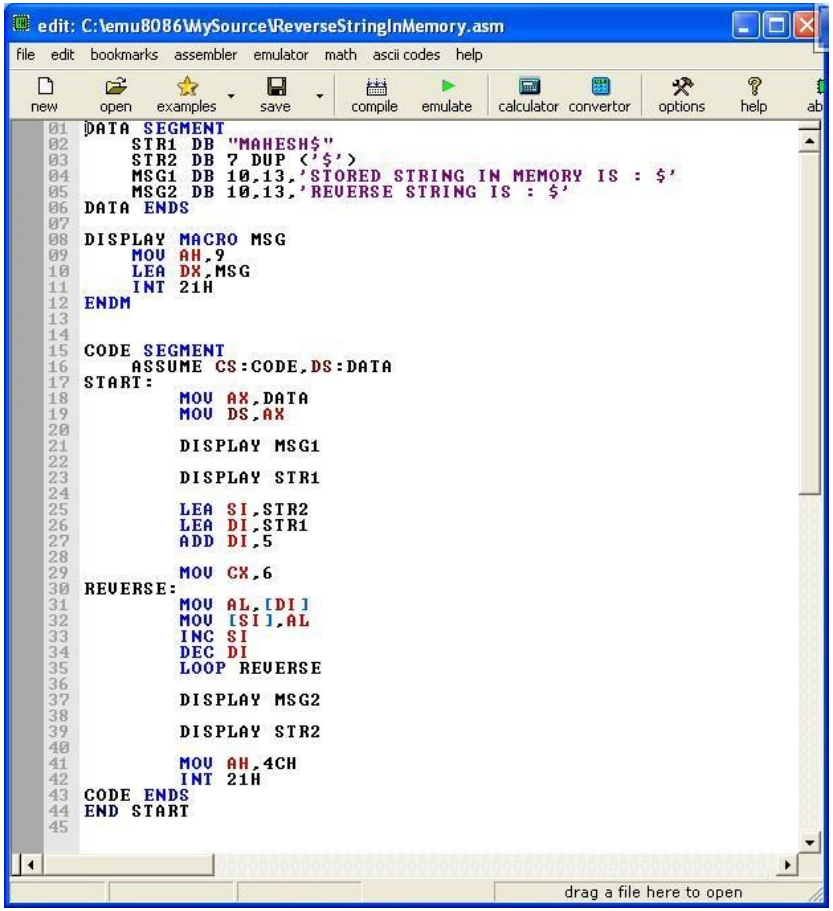
- Reduced Instruction Set Computing (RISC)
  - Few, simple, efficient, and fast instructions.
  - Examples: PowerPC from Apple/IBM/Motorola and ARM.
- Complex Instruction Set Computing (CISC)
  - Many, convenient, and powerful instructions.
  - Example: Intel.

CISC vs RISC

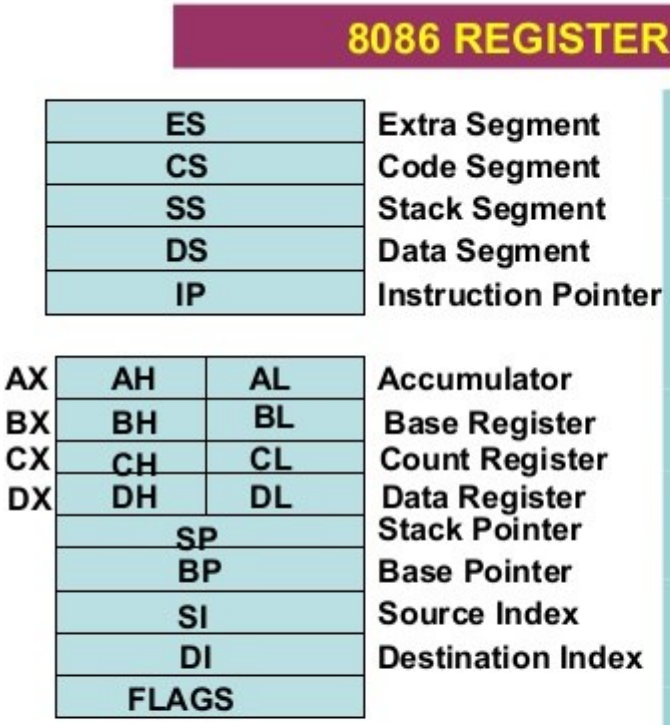
- CISC
  - Has more complex hardware.
  - More compact software code.
  - Takes more cycles per instruction.
  - Can use less RAM as no need to store intermediate results.
- RISC
  - Has simpler hardware.
  - More complicated software code.
  - Takes one cycle per instruction.
  - Can use more RAM to handle intermediate results.

Machine Instruction Types

- Data Transfer: copy data from one location to another.
  - (e.g. LOAD, STORE)
- Arithmetic/Logic: use existing bit patterns to compute a new bit patterns.
  - (e.g. +, -, *, /, AND, OR, SHIFT, ROTATE)
- Control: direct the execution of the program.
  - (e.g. JUMP, BRANCH)



Registers 8086



Type	Register size	Name of the Register
General purpose registers	16 bit	AX, BX, CX, DX
	8 bit	AL, AH, BL, BH, CL, CH, DL, DH
Pointer registers	16 bit	SP, BP
Indexed registers	16 bit	SI, DI
Instruction Pointer	16 bit	IP
Segment registers	16 bit	CS, DS, SS, ES
Flags	16 bit	Flag register

Registers (core iX, 64bit)

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

Adding Values Stored in Memory

Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

Dividing Values Stored in Memory

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

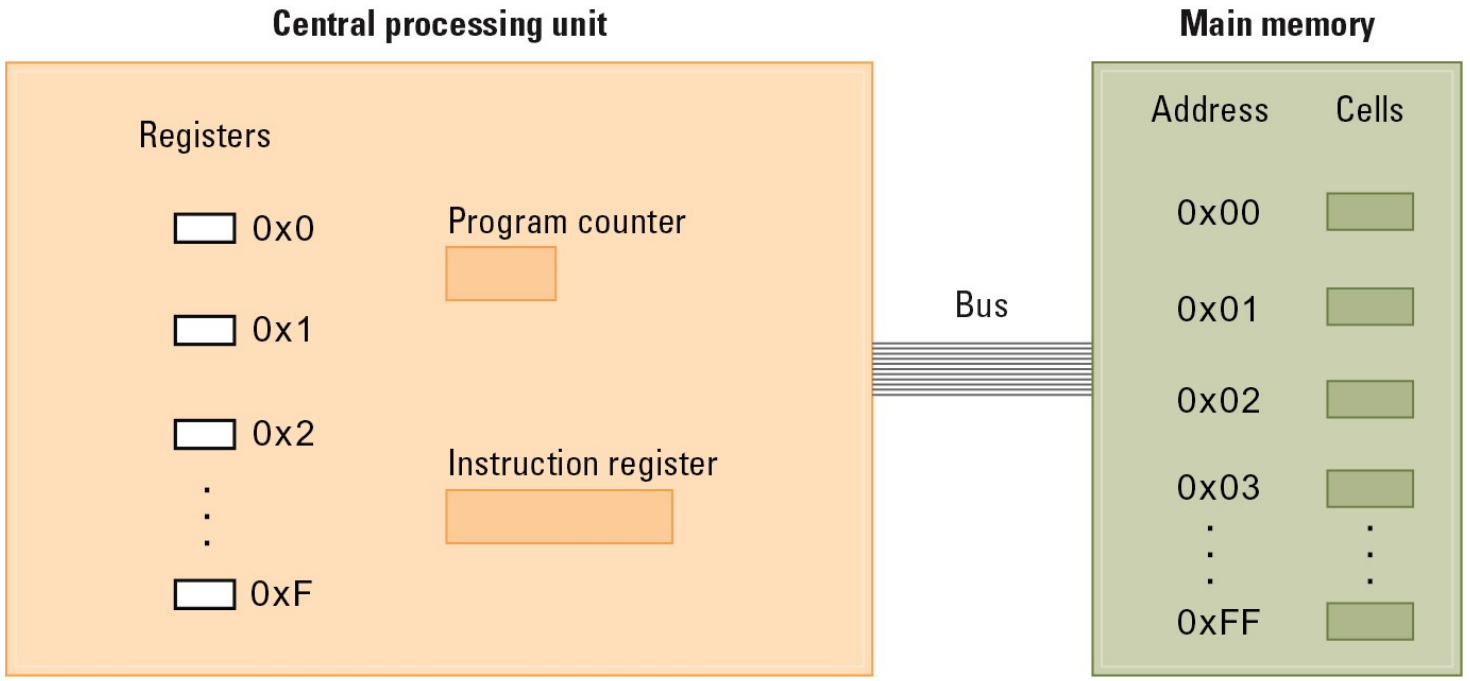
Step 3. If this second value is zero, JUMP to Step 6.

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

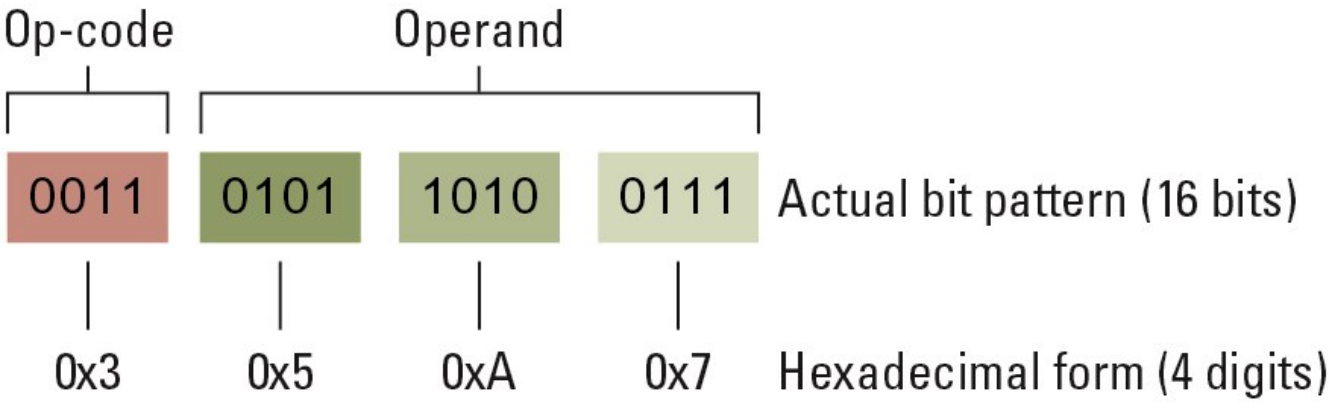
The Architecture of the Machine Described in Appendix C



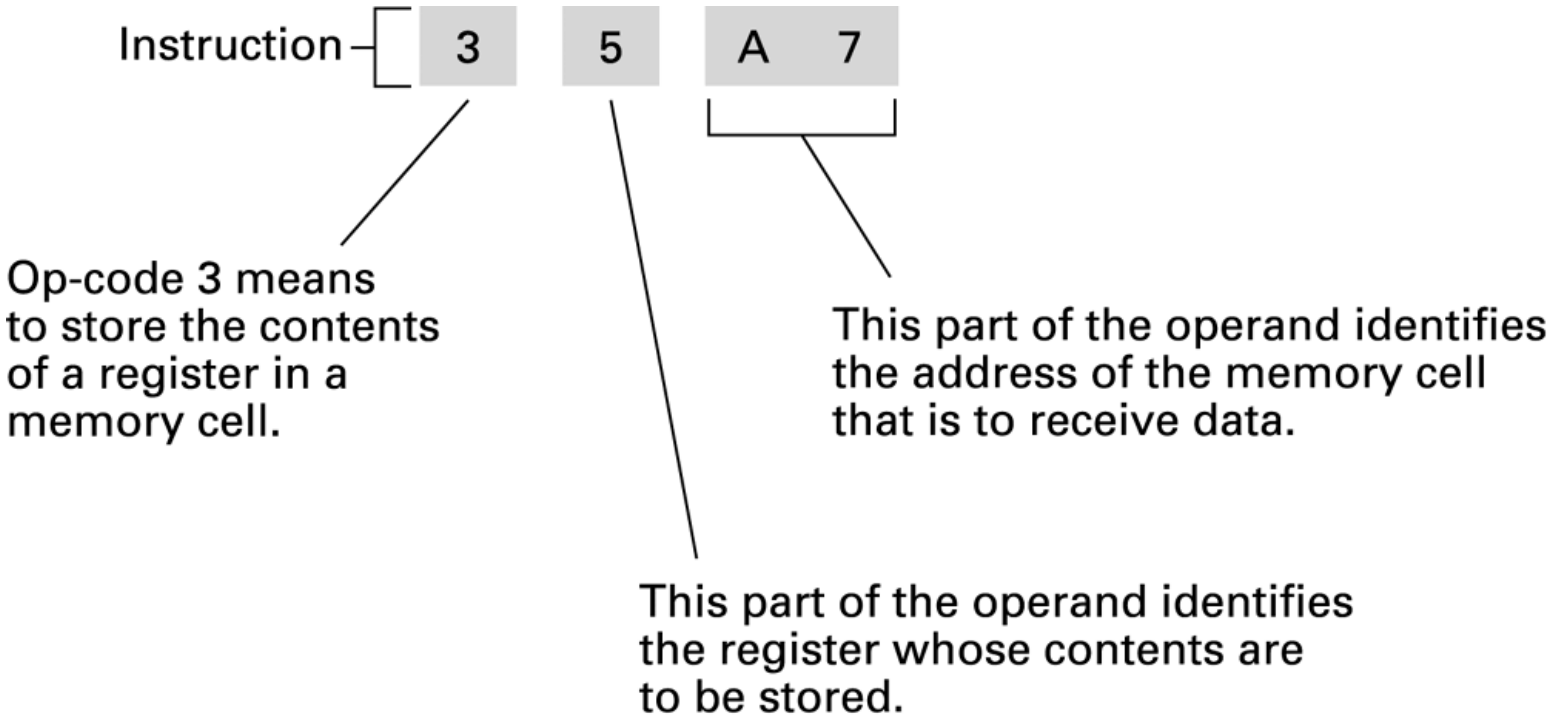
Parts of a Machine Instruction

- **Op-code:** Specifies which operation to execute.
- **Operand:** Gives more detailed information about the operation.
  - Interpretation of operand varies depending on op-code.

The Composition of an Instruction for the Machine in Appendix C



Decoding the Instruction 35A7

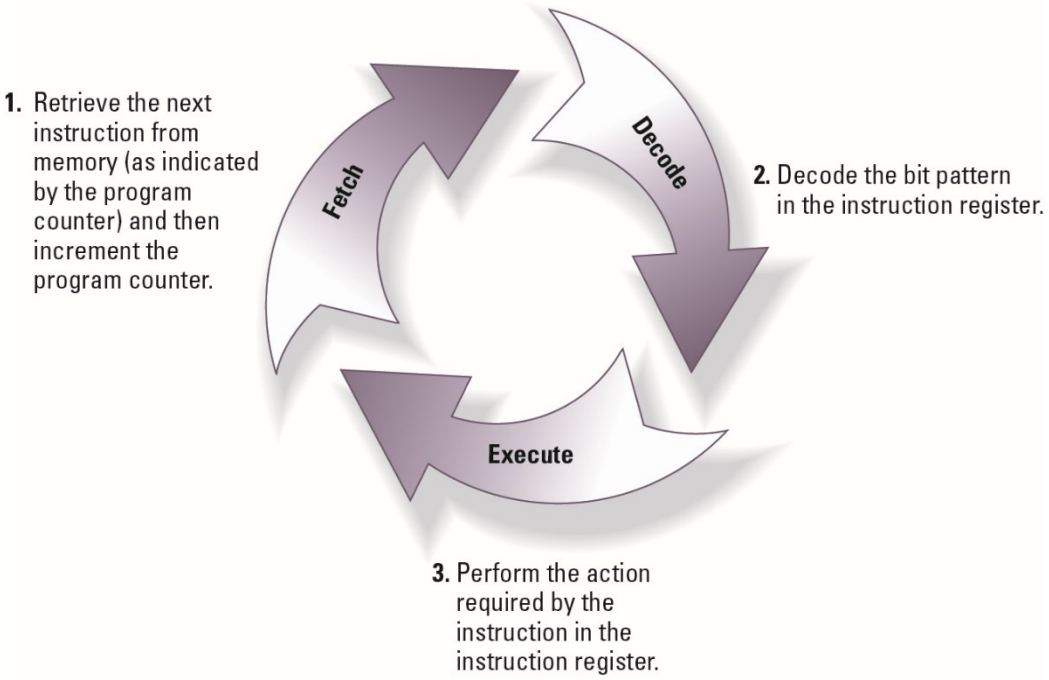


An Encoded Version of the Instructions in Figure 2.2

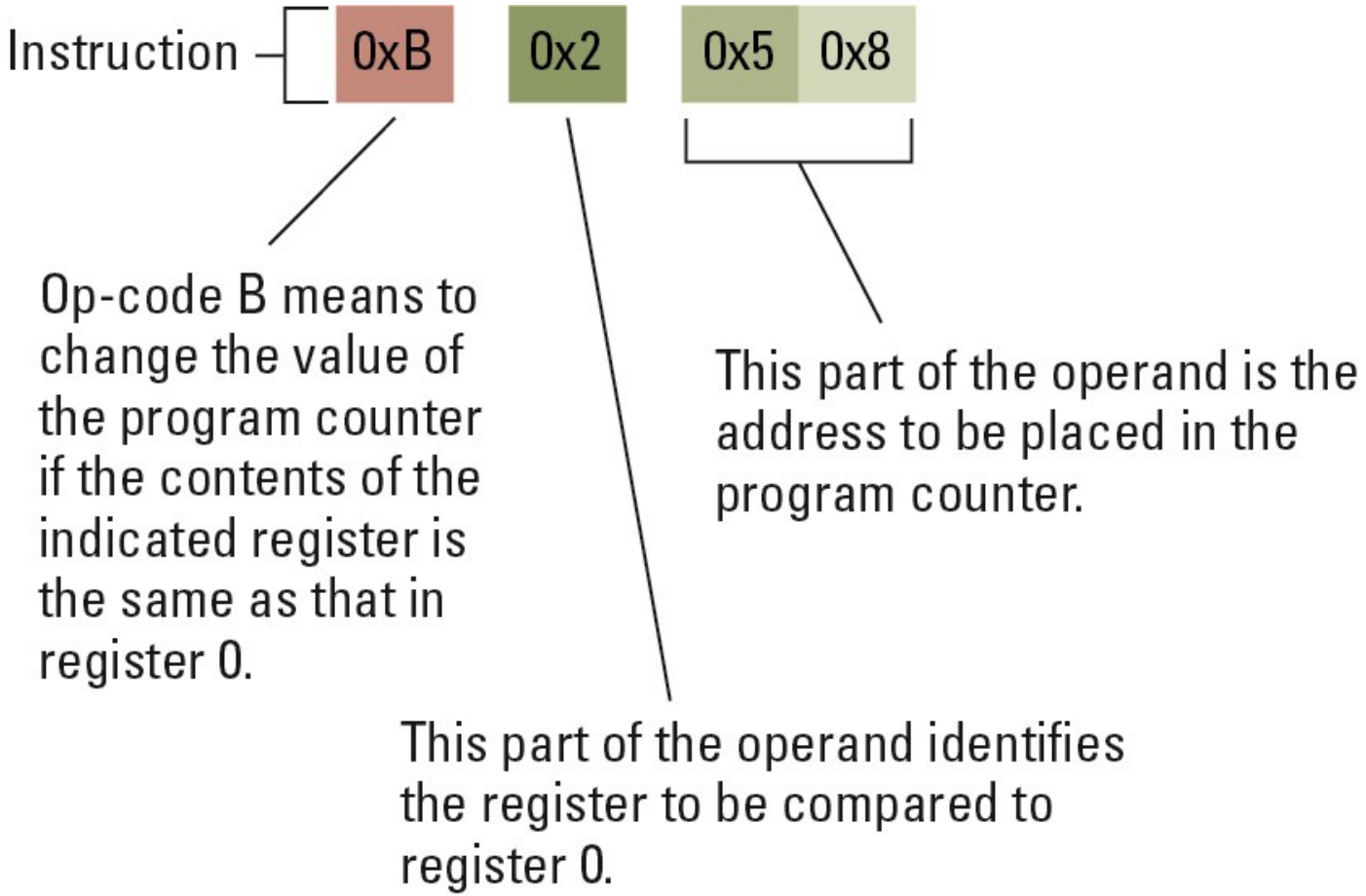
Encoded instructions	Translation
0x156C	Load register 0x5 with the bit pattern found in the memory cell at address 0x6C.
0x166D	Load register 0x6 with the bit pattern found in the memory cell at address 0x6D.
0x5056	Add the contents of register 0x5 and 0x6 as though they were two's complement representation and leave the result in register 0x0.
0x306E	Store the contents of register 0x0 in the memory cell at address 0x6E.
0xC000	Halt.

Program Execution

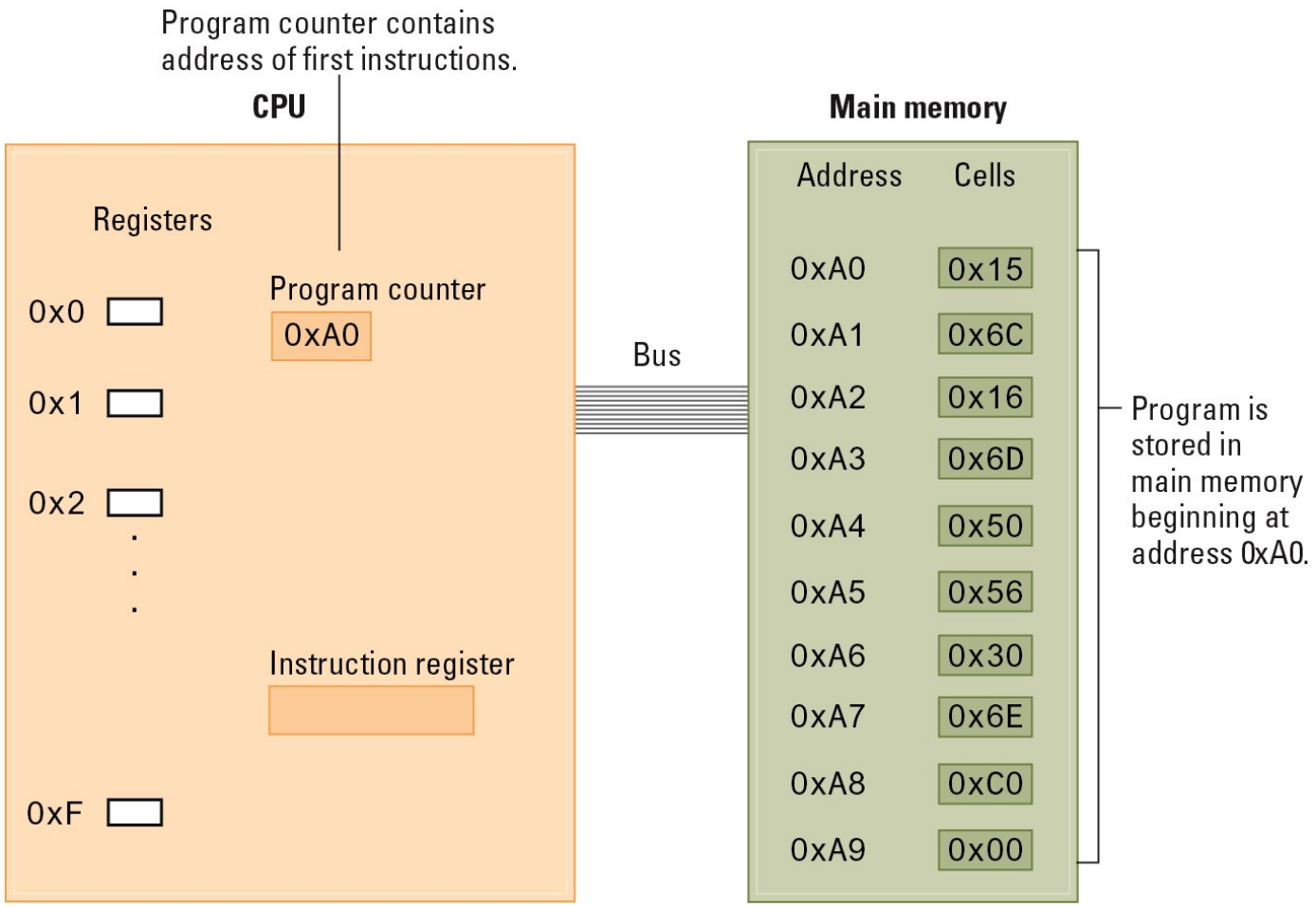
- Controlled by two special-purpose registers:
  - Program counter: address of next instruction.
  - Instruction register: current instruction.
- Machine Cycle
  - Fetch
  - Decode
  - Execute
  - (Store)



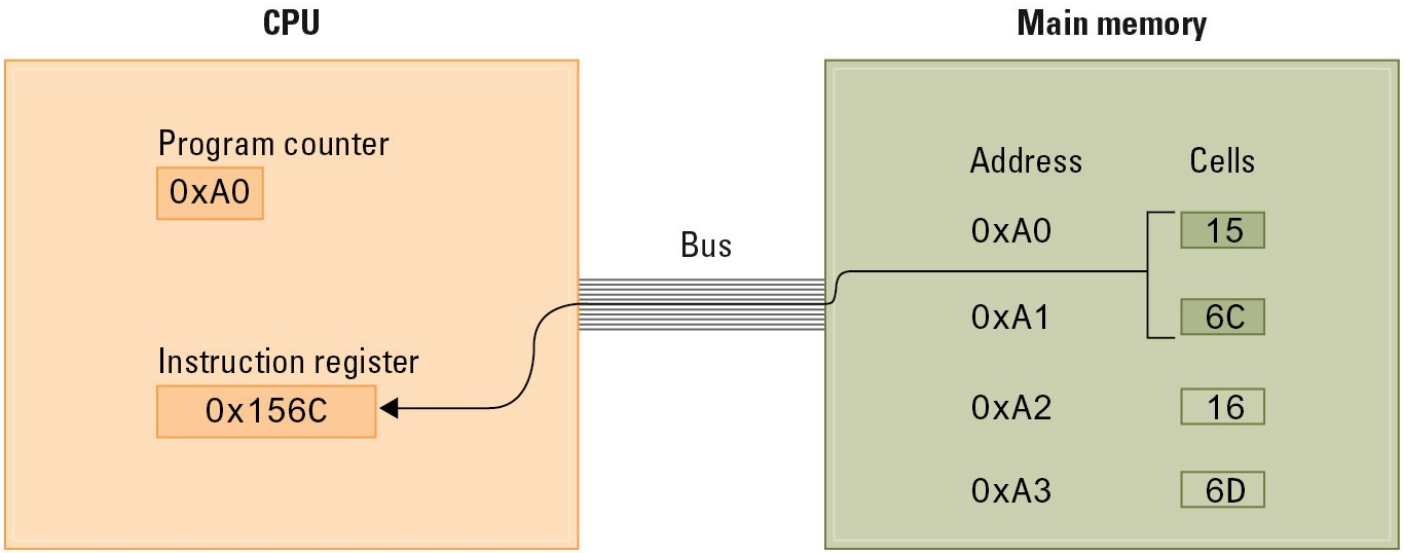
Decoding the Instruction B258



The Program from Figure 2.7 Stored in Main Memory Ready for Execution

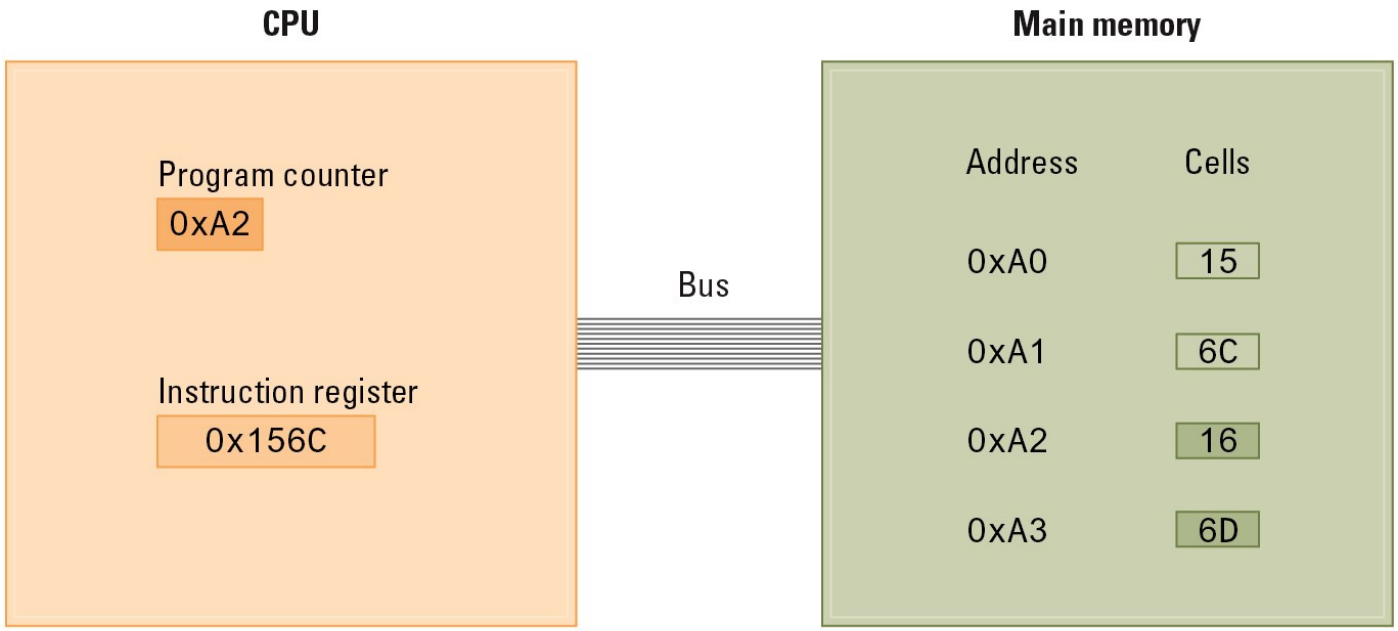


Performing the Fetch Step of the Machine Cycle



a. At the beginning of the fetch step, the instruction starting at address 0xA0 is retrieved from memory and placed in the instruction register.

Performing the Fetch Step of the Machine Cycle (continued)



b. Then the program counter is incremented so that it points to the next instruction.

How about CISC architecture

```
[17:16:10 0 2][wwyhsu@Citadel:~ $> objdump -d a.out

a.out:     file format elf64-x86-64


Disassembly of section .init:

00000000004003c8 <_init>:
  4003c8:       48 83 ec 08             sub    $0x8,%rsp
  4003cc:       48 8b 05 25 0c 20 00    mov    0x200c25(%rip),%rax        # 600ff8 <_DYNAMIC+0x1d0>
  4003d3:       48 85 c0                test   %rax,%rax
  4003d6:       74 05                   je     4003dd <_init+0x15>
  4003d8:       e8 43 00 00 00          callq  400420 <__libc_start_main@plt+0x10>
  4003dd:       48 83 c4 08             add    $0x8,%rsp
  4003e1:       c3                      retq
```

Endian Order

- Depending on which computing system you use, you will have to consider the byte order in which multi-byte numbers are stored, particularly when you are writing those numbers to a file.
- The two orders are called Little Endian and Big Endian.

- See: https://hunt8r.gitbooks.io/notes/content/topics/endiannes s.html

Little Endian

- Little Endian means that the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address. (The little end comes first.)

  For example, a 4 byte `long int`

  Byte3 Byte2 Byte1 Byte0

  will be arranged in memory as follows:

  Base Address+0 Byte0

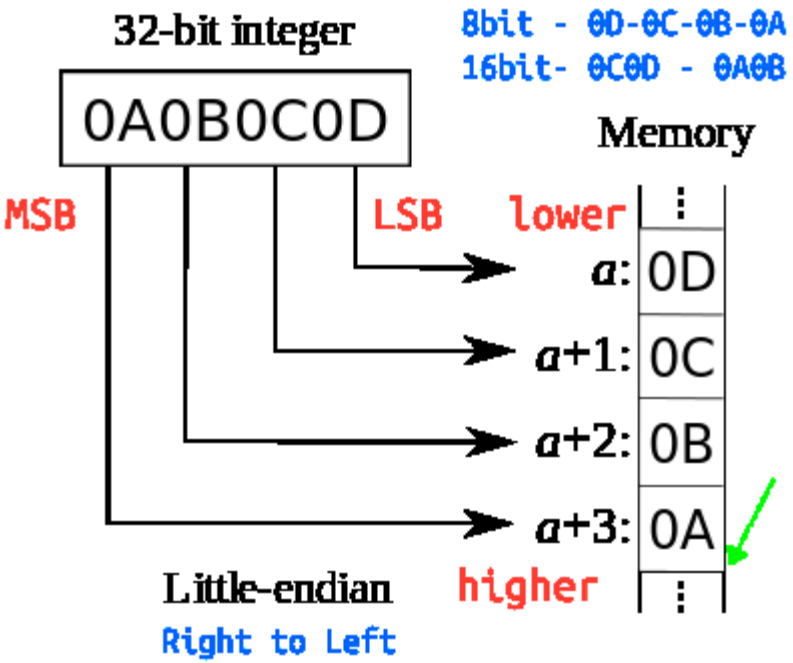
  Base Address+1 Byte1

  Base Address+2 Byte2

  Base Address+3 Byte3

- ***Intel processors*** (those used in **PC**'s) use "Little Endian" byte order.
- **Intel CPU**'s
- DEC Alphas
- **Domain Names (www.google.com)**
- **hexdump (**because the dumping program is unable to know what kind of data it is dumping, the only orientation it can observe is monotonically increasing addresses.)

Little Endian



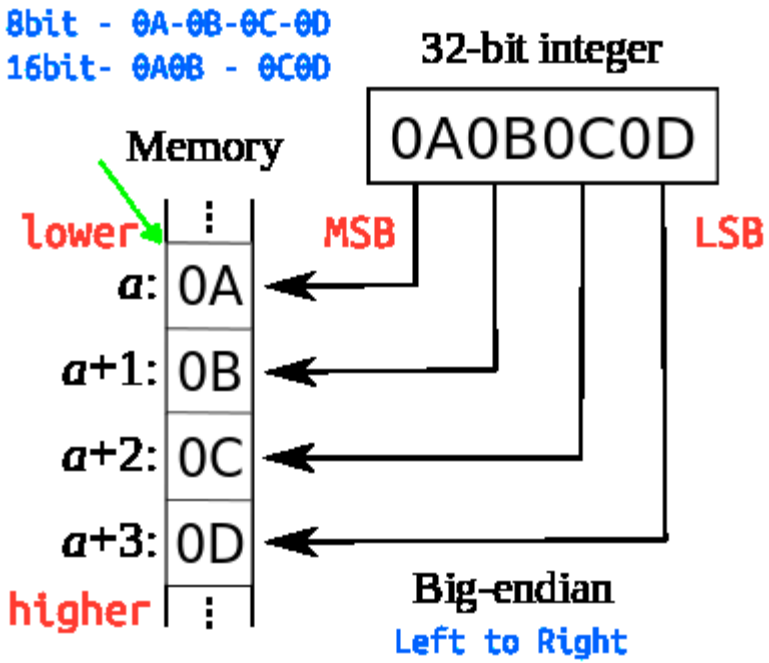
Big Endian

- Big Endian means that the high-order byte of the number is stored in memory at the lowest address, and the low-order byte at the highest address. (The big end comes first.)

Base Address+0	Byte3
Base Address+1	Byte2
Base Address+2	Byte1
Base Address+3	Byte0

- ***Motorola processors*** (those used in Mac's) use "Big Endian" byte order.
- IBM's 370 mainframes
- **TCP/IP**

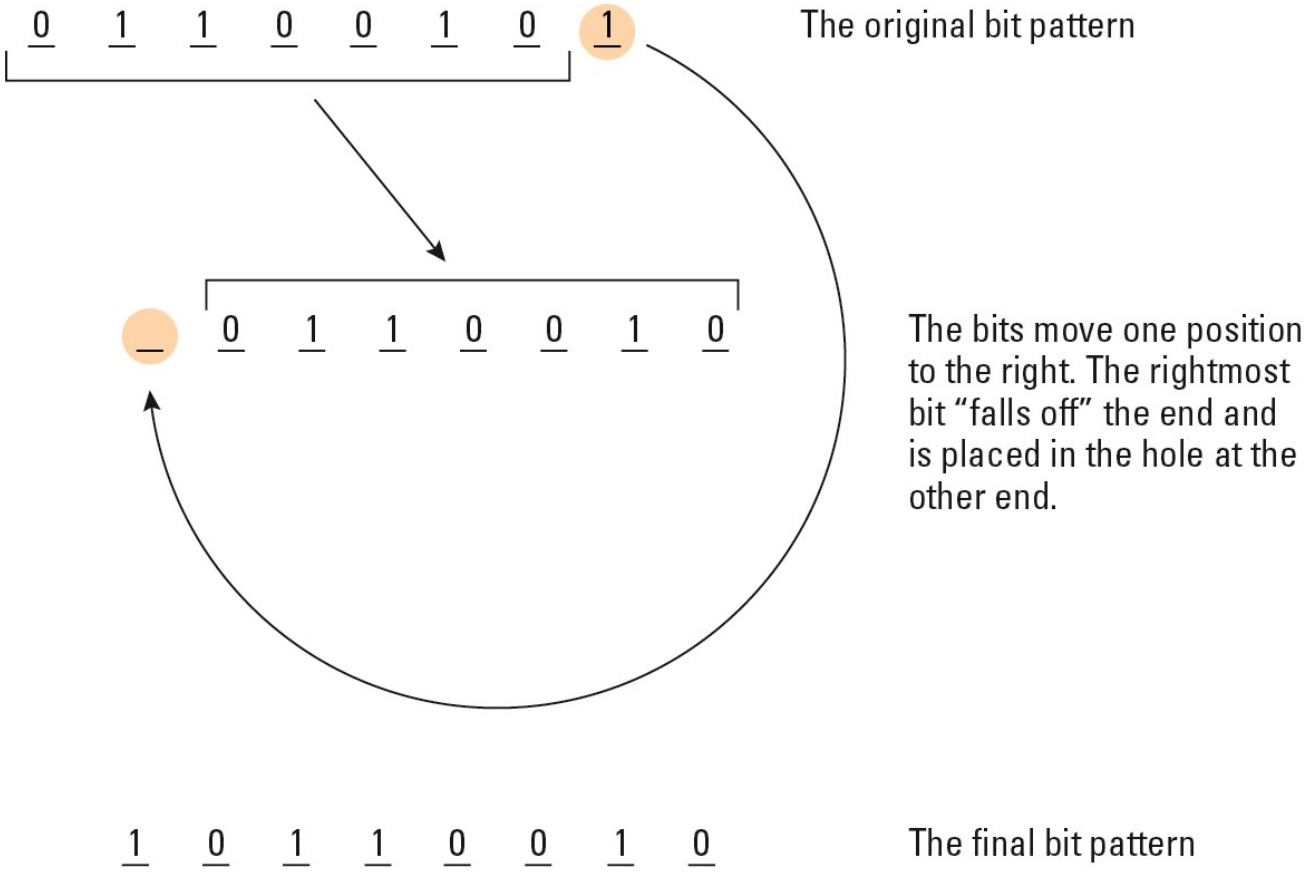
Big Endian


8bit - 0A-0B-0C-0D
16bit- 0A0B - 0C0D
32-bit integer
0A0B0C0D
Memory
lower
MSB
LSB
a: 0A
a+1: 0B
a+2: 0C
a+3: 0D
higher
Big-endian
Left to Right

Arithmetic/Logic Operations

- Logic: AND, OR, XOR
  - Masking
- Rotate and Shift: circular shift, logical shift, arithmetic shift
- Arithmetic: add, subtract, multiply, divide
  - Precise action depends on how the values are encoded (two's complement versus floating-point).

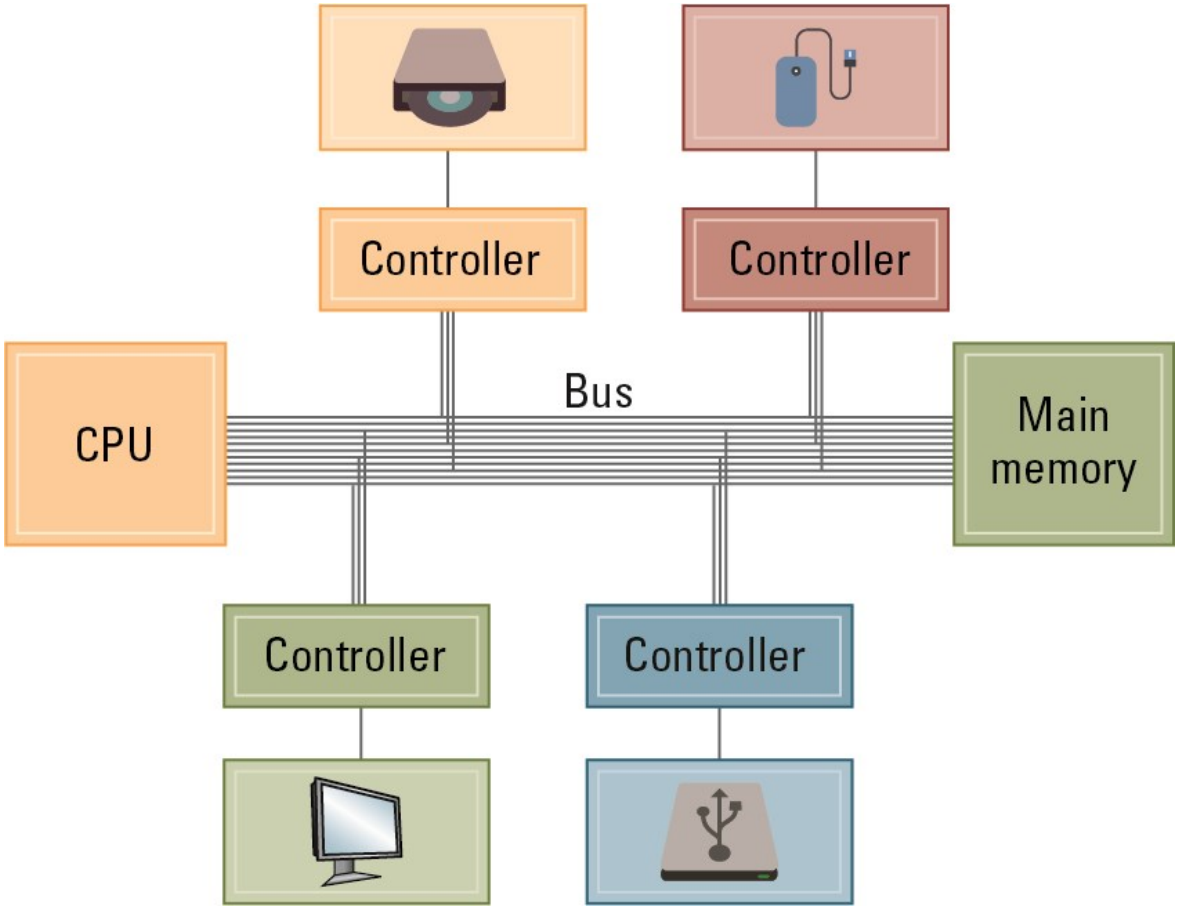
Rotating the Bit Pattern 65_h (hexadecimal) One Bit to the Right



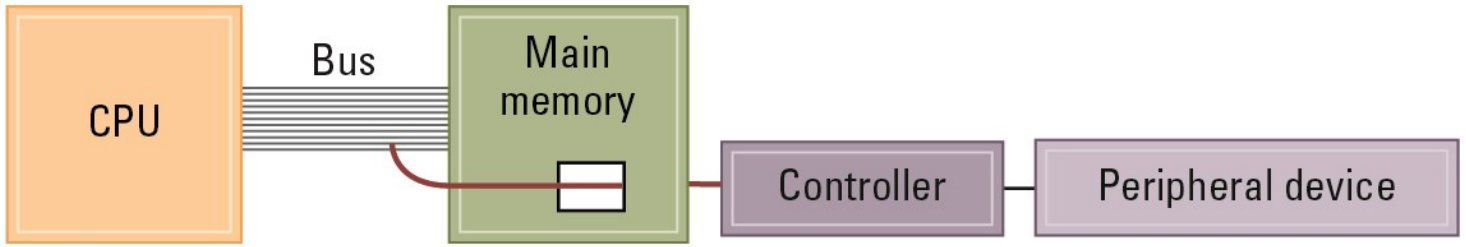
Communicating with Other Devices

- **Controller**: An intermediary apparatus that handles communication between the computer and a device.
  - Specialized controllers for each type of device
  - General purpose controllers (USB and FireWire)
- **Port**: The point at which a device connects to a computer
- **Memory-mapped I/O**: CPU communicates with peripheral devices as though they were memory cells

Controllers Attached to a Machine's Bus

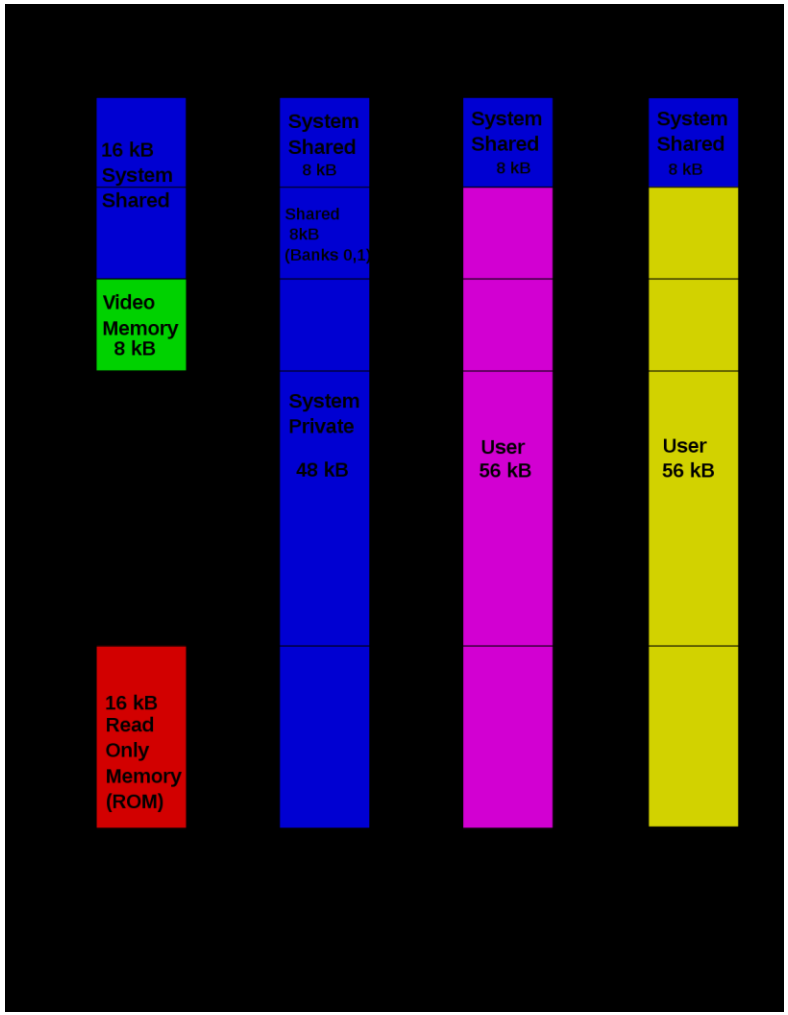


A Conceptual Representation of Memory-Mapped I/O



Memory-Mapped I/O

Memory banking technology - Nintendo

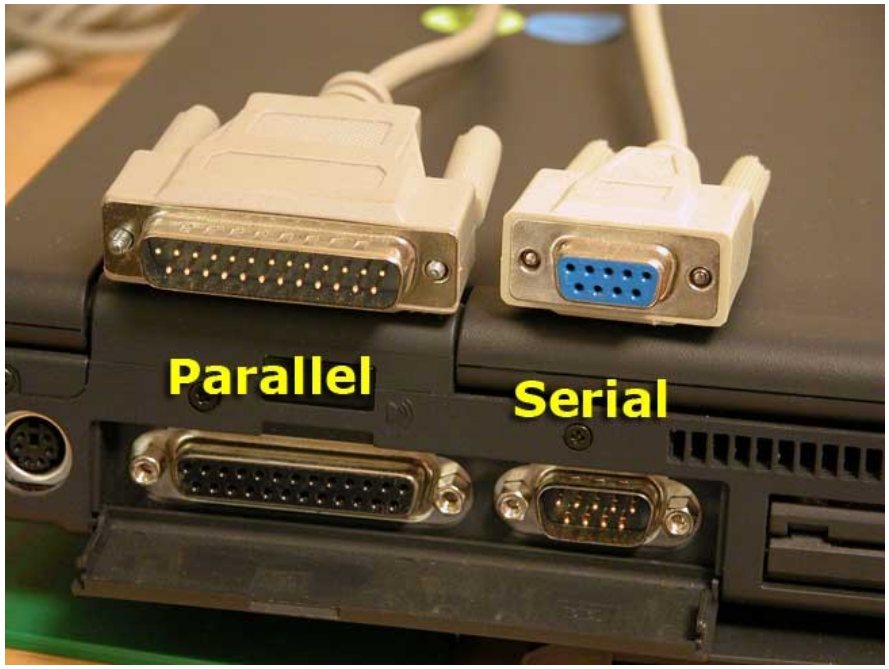


Communicating with Other Devices (continued)

- **Direct memory access (DMA):** Main memory access by a controller over the bus.
- **Von Neumann Bottleneck:** Insufficient bus speed impedes performance.
- **Handshaking:** The process of coordinating the transfer of data between components .

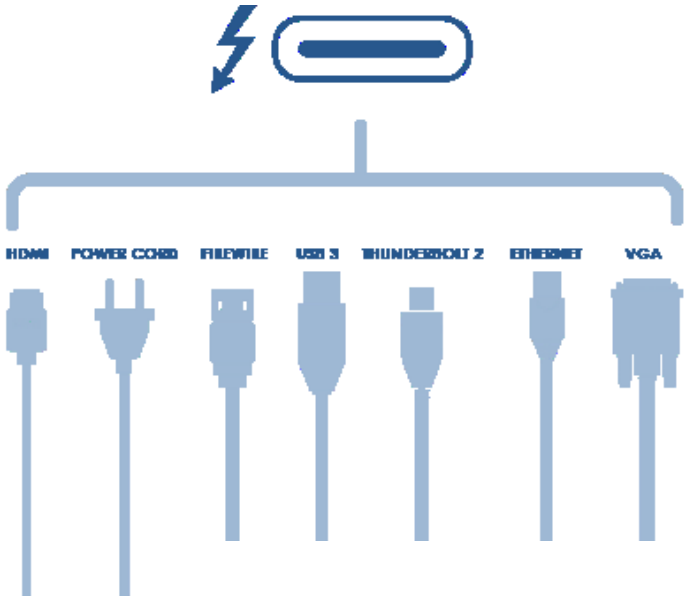
Communicating with Other Devices (continued)

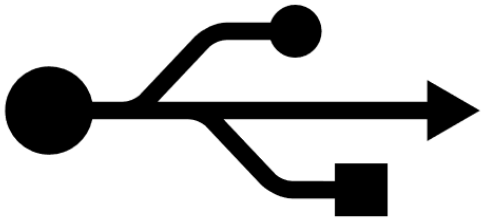
- **Parallel Communication:** Several communication paths transfer bits simultaneously.
- **Serial Communication:** Bits are transferred one after the other over a single communication path.

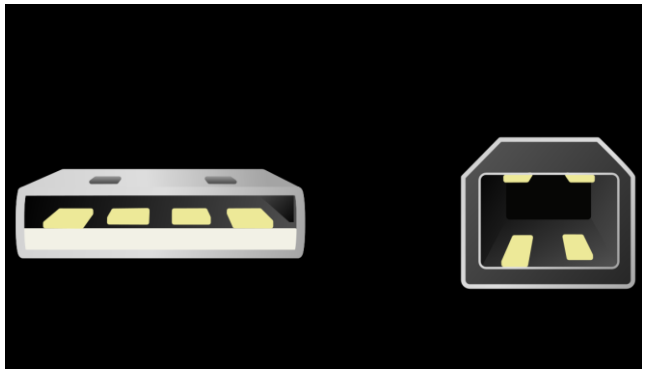


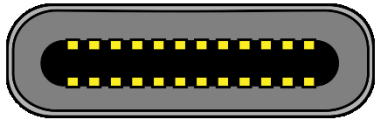
Communicating with Other Devices (continued)

- **Universal serial bus (USB)**
- **Thunderbolt**





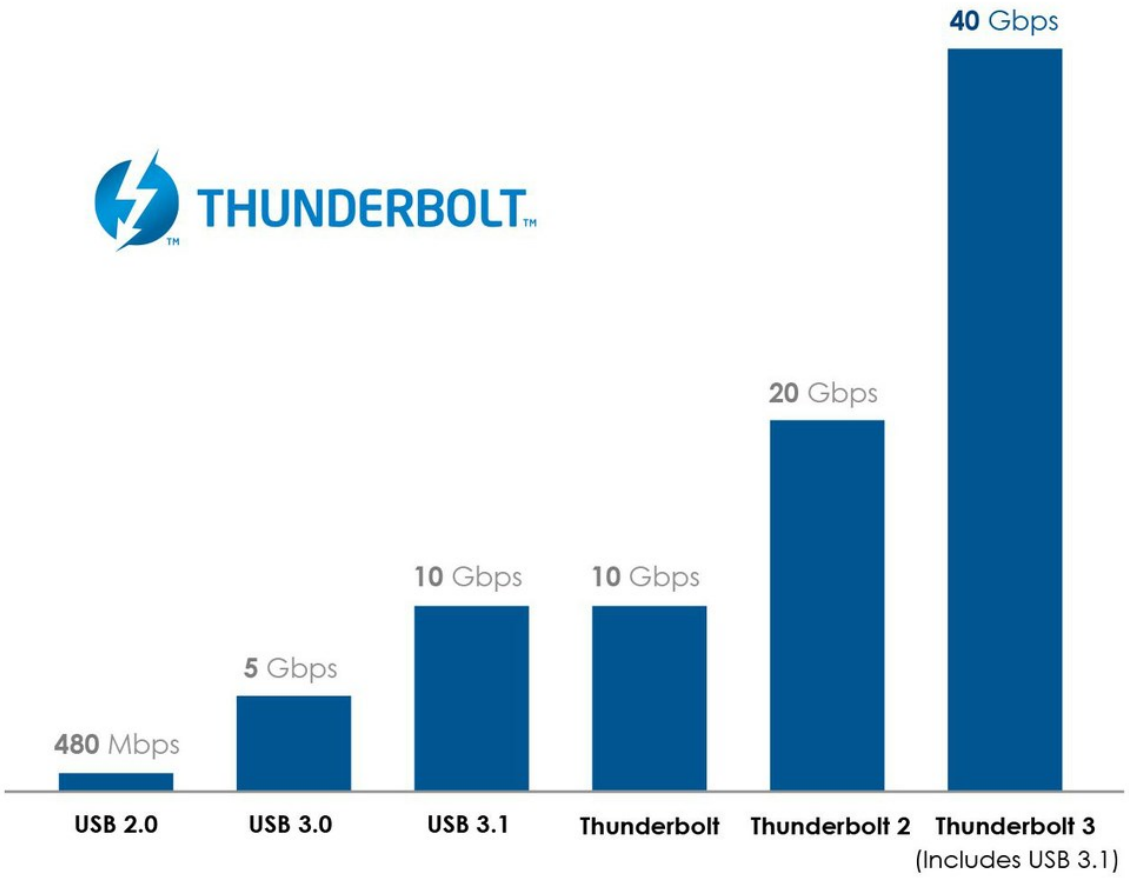




Type-C

IUNDERBOLT™

Transfer speed: USB vs Thunderbolt



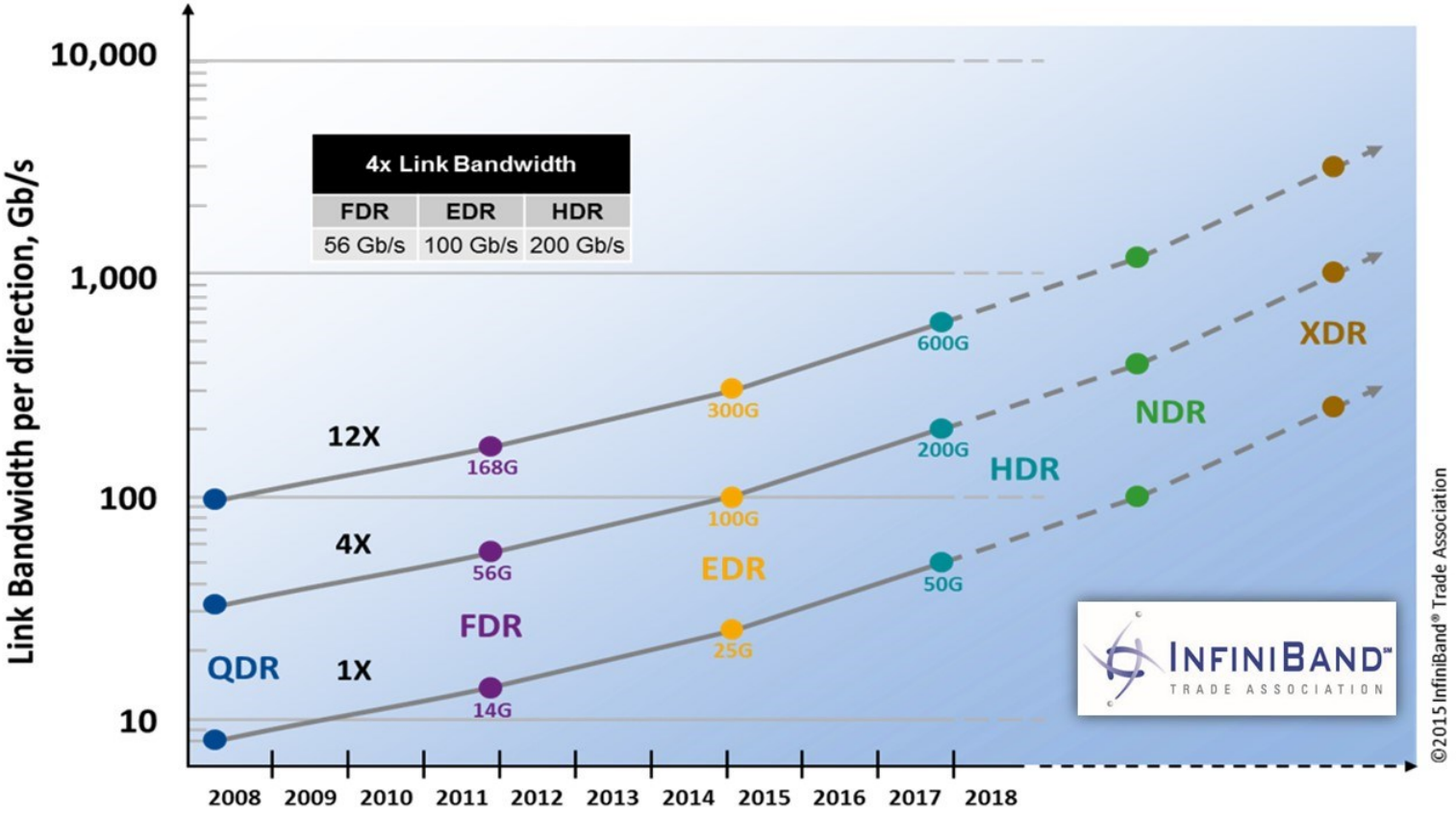
Why serial instead of parallel?

- Signal frequency cause synchronization problems.
- Crosstalk.

Anything faster?

- **InfiniBand** (abbreviated **IB**) is a computer-networking communications standard used in high-performance computing that features very high throughput and very low latency.
  - Data interconnection, switches, storage – server connections.

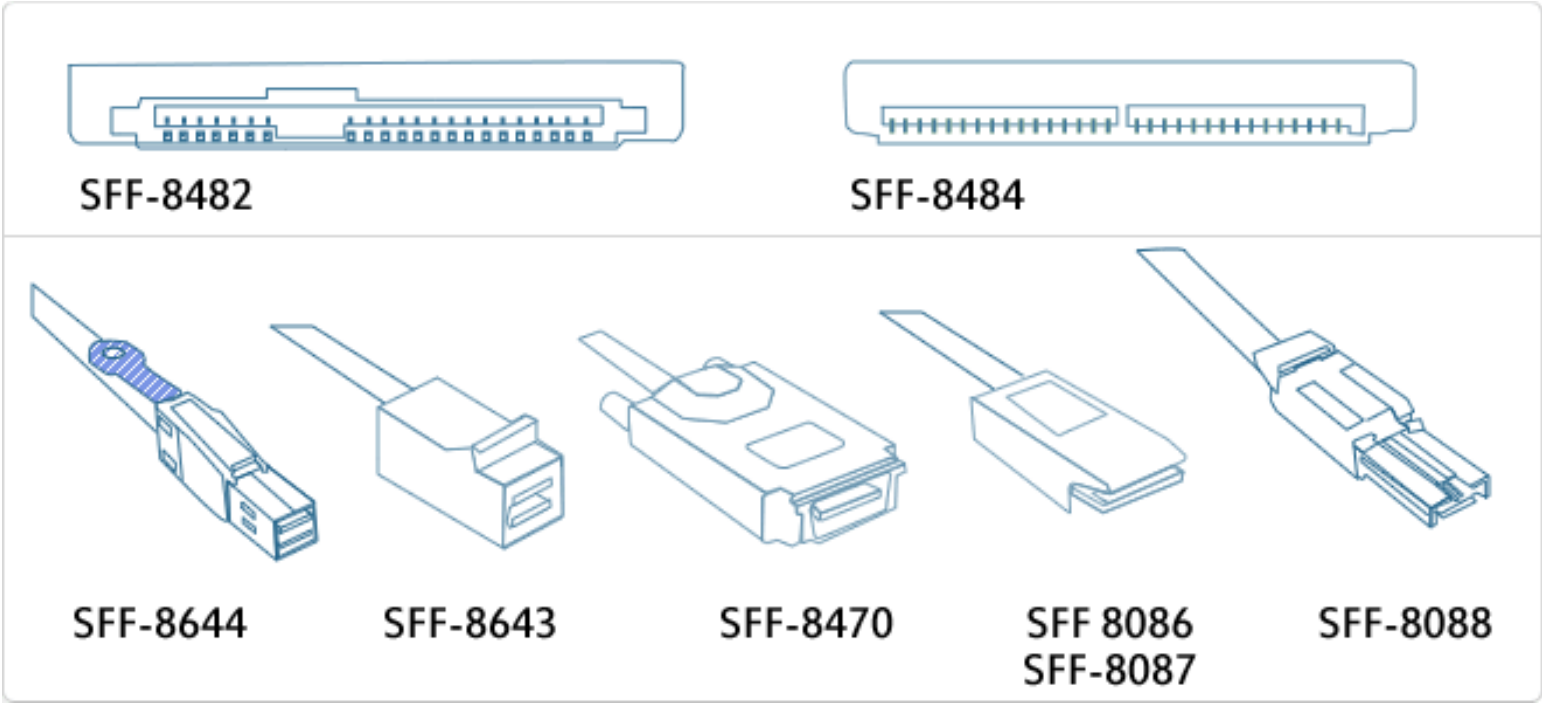
Infiniband speed



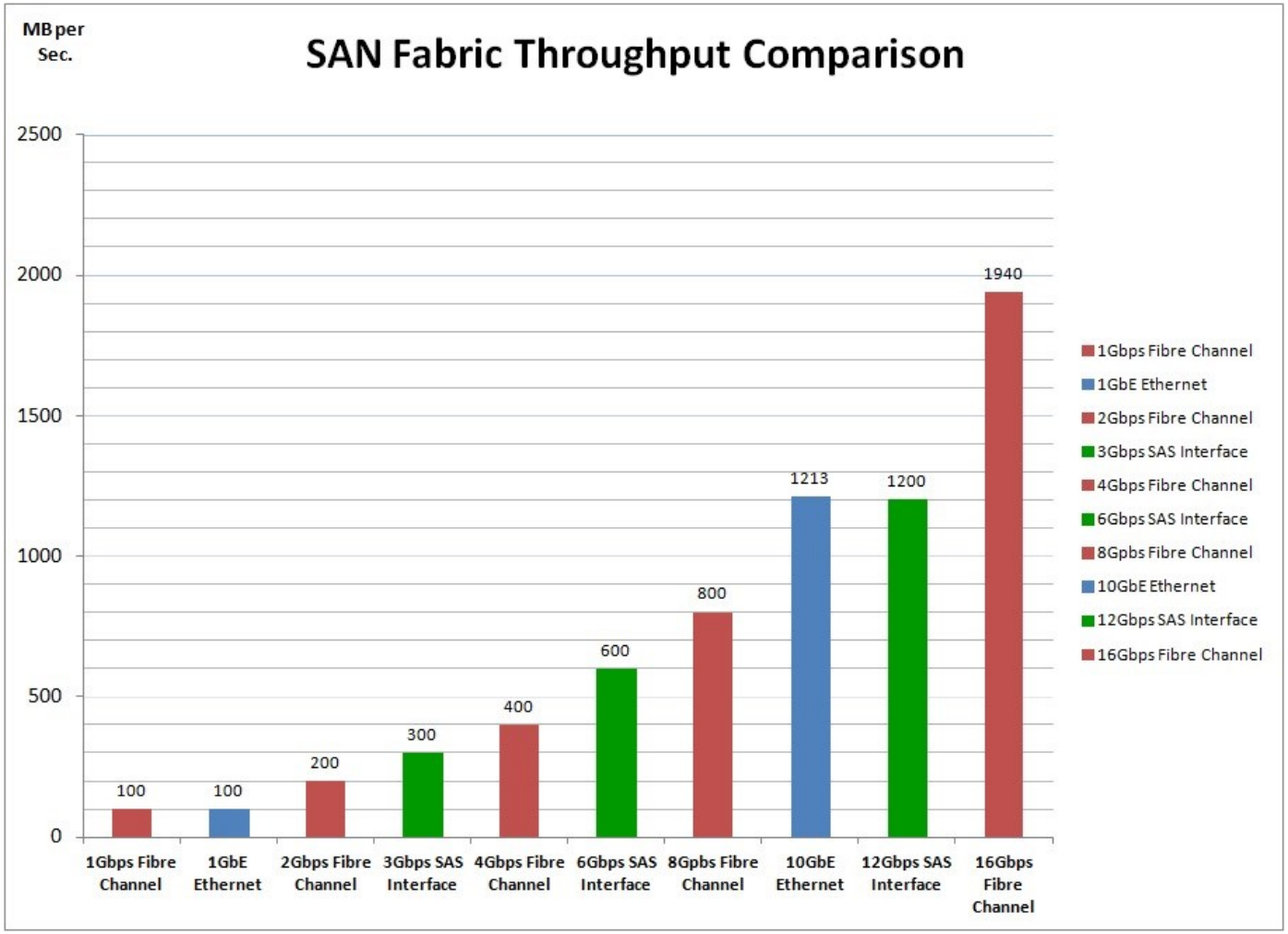
Serial attached SCSI

- **Serial Attached SCSI** (**SAS**) is a point-to-point serial protocol that moves data to and from computer-storage devices such as hard drives and tape drives.
- Replaces the older Parallel SCSI (Parallel Small Computer System Interface ("scuzzy") bus technology in mid 1980s.
- SAS offers optional compatibility with Serial ATA (SATA), versions 2 and later.
  - This allows the connection of SATA drives to most SAS backplanes or controllers.
  - The reverse, connecting SAS drives to SATA backplanes, is not possible

External SAS connectors



Serial attached SCSI



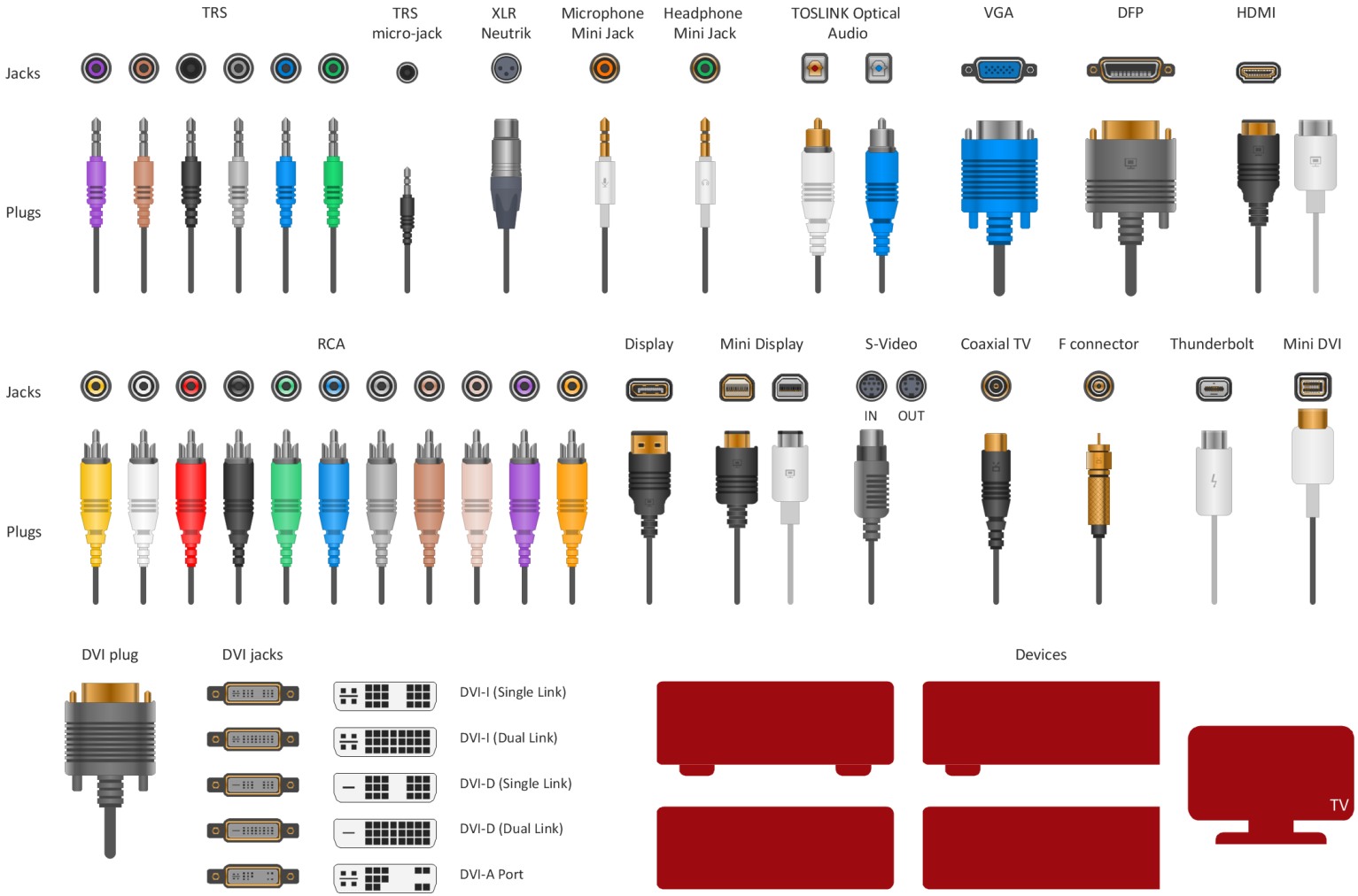
Data Communication Rates

- Measurement units:
  - Bps: Bits per second
  - Kbps: Kilo-bps (1,000 bps)
  - Mbps: Mega-bps (1,000,000 bps)
  - Gbps: Giga-bps (1,000,000,000 bps)
- Bandwidth: Maximum available rate.

Communication Speed over the Years

Technology	Rate		Year
FireWire (IEEE 1394) 200	**196.608 Mbit/s**	24.576 MB/s	1995
FireWire (IEEE 1394) 400	**393.216 Mbit/s**	49.152 MB/s	1995
USB high speed	**480 Mbit/s**	60 MB/s	2000
FireWire (IEEE 1394b) 800[56]	**786.432 Mbit/s**	98.304 MB/s	2002
Fibre Channel 1 Gb SCSI	**1 062.5 Mbit/s**	100 MB/s	
FireWire (IEEE 1394b) 1600[56]	**1.573 Gbit/s**	196.6 MB/s	2007
Fibre Channel 2 Gb SCSI	**2125 Mbit/s**	200 MB/s	
eSATA (SATA 300)	**3 Gbit/s**	375 MB/s	2004
USB SuperSpeed	**5 Gbit/s**	625 MB/s	2010
eSATA (SATA 600)	**6 Gbit/s**	750 MB/s	2011
USB SuperSpeed+	**10 Gbit/s**	1250 MB/s	2013
Thunderbolt	2 × **10 Gbit/s**	2 × 1250 MB/s	2011
External PCI Express 2.0 ×4	**16 Gbit/s**	2000 MB/s	
Thunderbolt 2	**20 Gbit/s**	2500 MB/s	2013
External PCI Express 2.0 ×8	**32 Gbit/s**	4000 MB/s	
Thunderbolt 3	**40 Gbit/s**	5000 MB/s	2015
External PCI Express 2.0 ×16	**64 Gbit/s**	8000 MB/s	

Standards


TRS
TRS micro-jack
XLR Neutrik
Microphone Mini Jack
Headphone Mini Jack
TOSLINK Optical Audio
VGA
DFP
HDMI
Jacks
Plugs
RCA
Display
Mini Display
S-Video
IN
OUT
Coaxial TV
F connector
Thunderbolt
Mini DVI
DVI plug
DVI jacks
DVI-I (Single Link)
DVI-I (Dual Link)
DVI-D (Single Link)
DVI-D (Dual Link)
DVI-A Port
Devices
TV

Programming Data Manipulation

- Programing languages shields users from details of the machine:
  - A single Python statement might map to one, tens, or hundreds of machine instructions.
  - Programmer does not need to know if the processor is RISC or CISC.
  - Assigning variables surely involves LOAD, STORE, and MOVE op-codes.

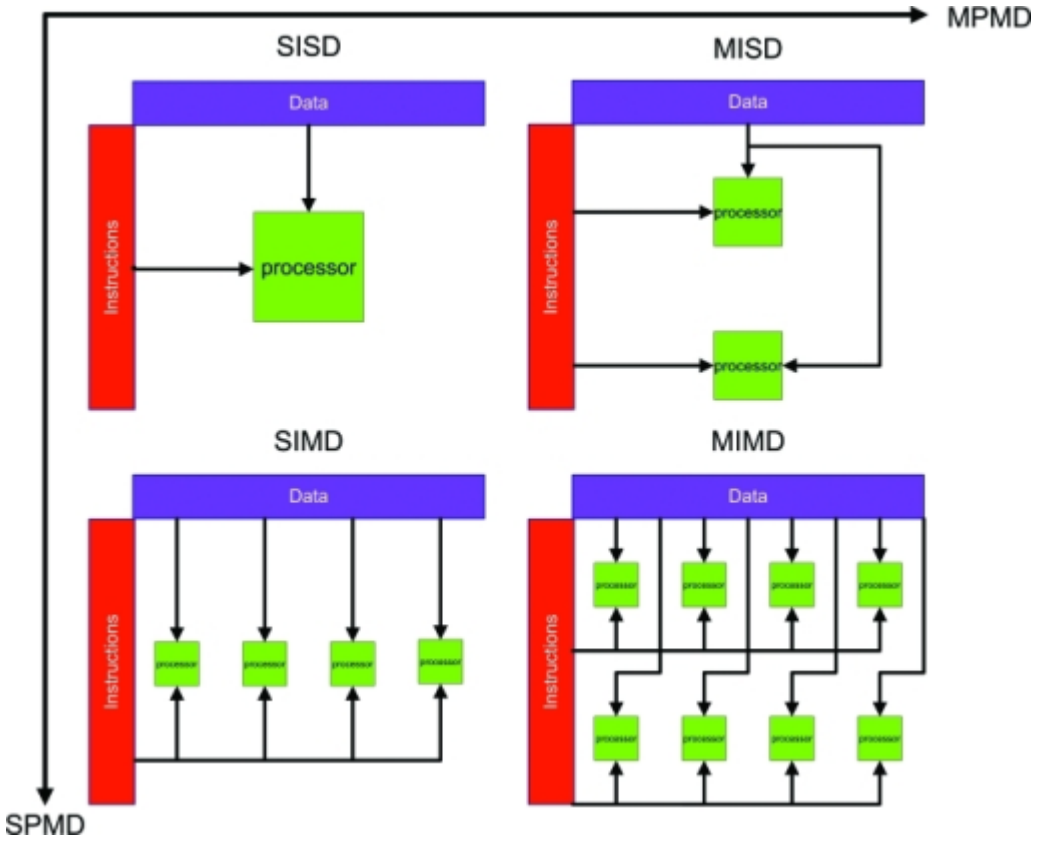
Example Marathon Training Data

Time Per Mile				Total Elapsed Time	
Minutes	Seconds	Miles	Speed (mph)	Minutes	Seconds
9	14	5	6.49819494584	46	10
8	0	3	7.5	24	0
7	45	6	7.74193548387	46	30
7	25	1	8.08988764044	7	25

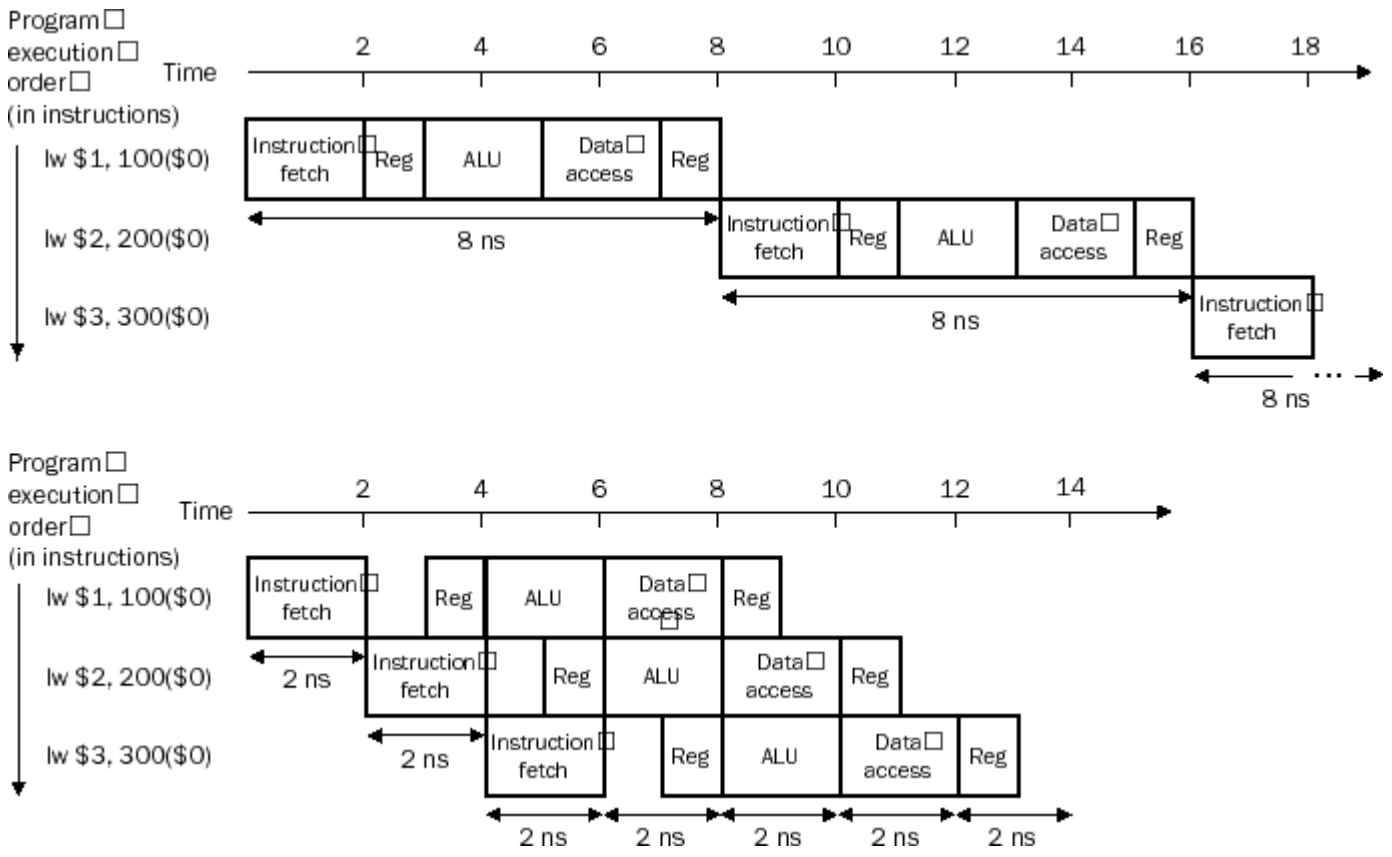
Other Architectures

- Technologies to increase throughput:
  - Pipelining: Overlap steps of the machine cycle.
  - Superscalar: Multiple instructions per cycle.
  - Parallel Processing: Use multiple processors simultaneously
    - SISD: No parallel processing
    - MIMD: Different programs, different data
    - SIMD: Same program, different data
    - MISD?

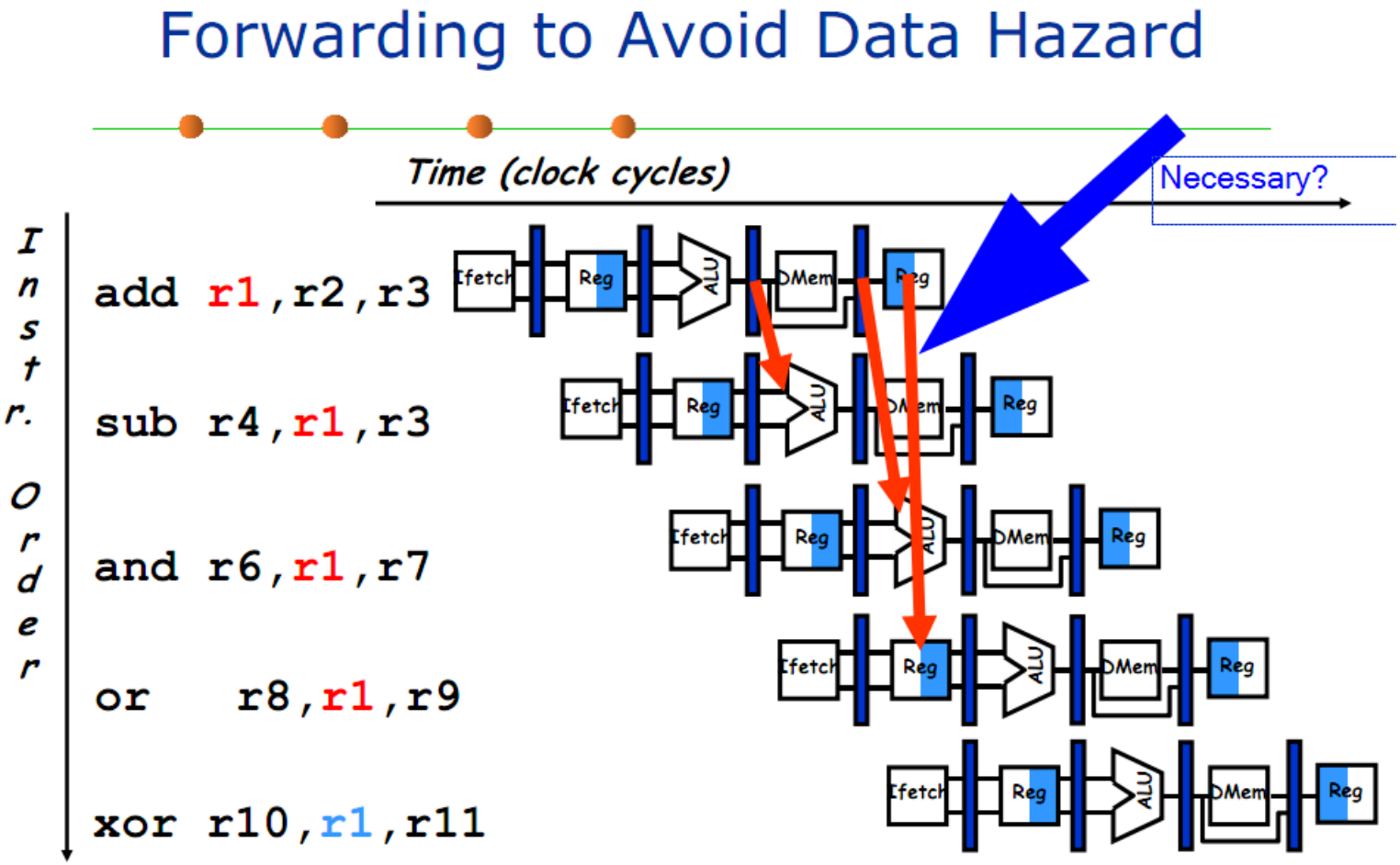
Parallel Processing



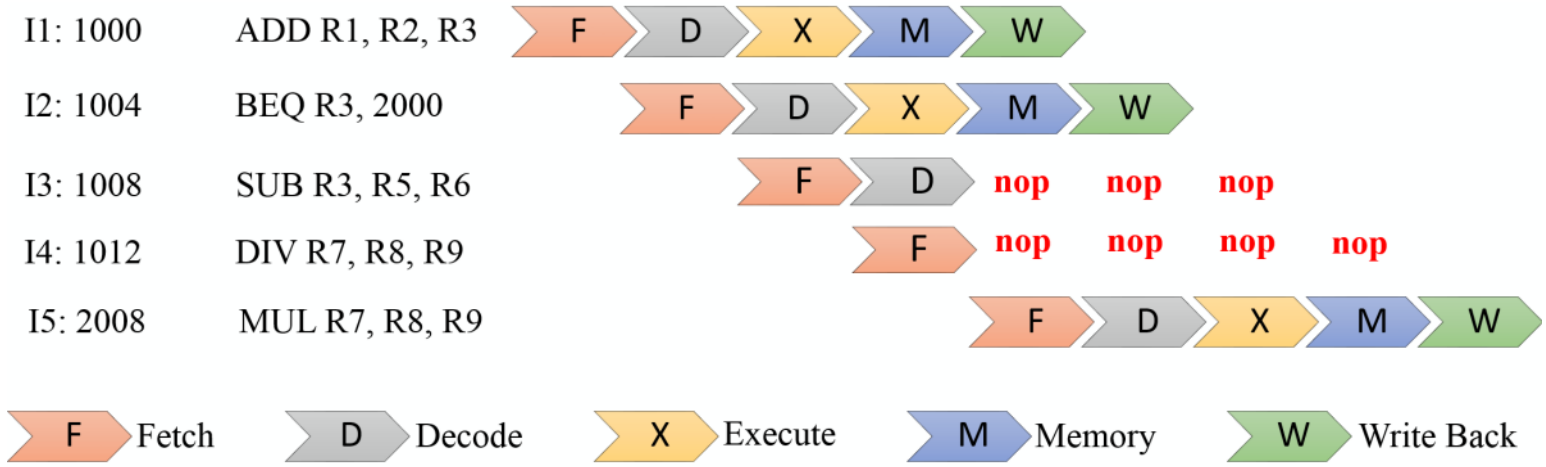
Pipelining


Program execution order (in instructions)
Time
2
4
6
8
10
12
14
16
18
lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)
Instruction fetch
Reg
ALU
Data access
Reg
8 ns
Program execution order (in instructions)
Time
2
4
6
8
10
12
14
2 ns

Data Hazards



5 State stalls



Branch prediction

	Time								
	1	2	3	4	5	6	7	8	9
a = 0	F	D	E	S					
a += 1		F	D	E	S				
if (a < 10)			F	D	E	S			
a += 2				F	D		E	S	
a += 3					F	D		E	S

Superscalar