**Chapter 5**

**Computer Systems Organization**

**A Guide to this Instructor’s Manual:**

We have designed this Instructor’s Manual to supplement and enhance your teaching experience through classroom activities and a cohesive chapter summary.

This document is organized chronologically, using the same headings that you see in the textbook. Under the headings you will find: lecture notes that summarize the section, Teaching Tips, Class Discussion Topics, and Additional Projects and Resources. Pay special attention to teaching tips and activities geared towards quizzing your students and enhancing their critical thinking skills.

In addition to this Instructor’s Manual, our Instructor’s Resources also contain PowerPoint Presentations, Test Banks, and other supplements to aid in your teaching experience.

|  |
| --- |
| **At a Glance** |

**Instructor’s Manual Table of Contents**

* Overview
* Learning Objectives
* Teaching Tips and Quick Quizzes
* Class Discussion Topics
* Additional Projects
* Additional Resources
* Key Terms
* Solutions to End-of-Chapter Exercises

|  |
| --- |
| Lecture Notes |

**Overview**

Chapter 5 moves up one level in the hierarchy of abstraction to talk about how the Von Neumann architecture works, built out of circuit pieces from the preceding chapter, along with other parts. It outlines the characteristics of the Von Neumann architecture, and then looks in detail at each piece of the processor required to implement the architecture. The chapter explains how information is organized, stored, and retrieved from random access memory. It describes different I/O devices, particularly mass storage devices, and the need for I/O controllers to handle the slow speed of I/O devices. It describes the ALU and the control unit and how all those pieces fit together to form a Von Neumann machine. It then discusses some reasons why the Von Neumann architecture is becoming limited and describes work in parallel processing to get around the Von Neumann bottleneck.

# **Learning Objectives**

* Enumerate the characteristics of the Von Neumann architecture
* Describe the components of a random access memory system, including how fetch and store operations work, and the use of cache memory to speed up access time
* Diagram the components of a typical arithmetic/logic unit (ALU) and illustrate how the ALU data path operates
* Describe the operation of the control unit and explain how it implements the stored program characteristic of the Von Neumann architecture
* List and explain the types of instructions in a typical instruction set, and how instructions are commonly encoded
* Diagram the organization of a typical Von Neumann machine
* Show the sequence of steps, using the book’s notation, in the fetch, decode, and execute cycle to perform a typical instruction

# **Teaching Tips**

**5.1 Introduction**

1. Introduce the terms **functional units** and **computer organization**. Functional units or subsystems perform tasks such as instruction processing, information storage, computation, and data transfer. Computer organization is the field within computer science that examines the functional units of a computer’s hardware. Note that this field is distinct from hardware design.
2. Emphasize the importance in computer science of using **levels of abstraction** to manage the complexity of computer and software systems. Introduce the term **hierarchy of abstractions**, and illustrate it with the example of transistors grouped into gates and gates into circuits. Note that the process of abstracting away details occurs at all levels of computer science.

**5.2 The Components of a Computer System**

1. Introduce the term **Von Neumann architecture**, and stress that it is the foundation for virtually all modern computers.
2. The Von Neumann architecture has four major subsystems called memory, input/output, the arithmetic/logic unit (ALU), and the control unit. The **central processing unit (CPU)** is generally a bundle of the ALU and the control unit.

**5.2.1 Memory and Cache**

1. Introduce the terms **memory** and **random access memory (RAM)**. All data stored in memory is represented using the binary representations from Chapter 4.
2. Introduce the term **read-only memory (ROM)** as a form of RAM that is prerecorded with its values and cannot be changed. This memory is generally used to store important system instructions data where the user cannot overwrite them.
3. Memory is divided into **cells** of a fixed size, and each cell is labeled with a unique number: its **address**. A computer’s **cell size** or **memory width** is the number of bits per cell, usually 8 bits these days. Cell is a term that is now relatively obsolete.
4. The number of cells in a memory relates to the number of bits in each memory address. N bits in the address mean 2N different addresses, thus the **maximum memory size** or **address space** is determined. Emphasize the difference between the address of a memory cell and its value. Use the metaphor of addresses on houses versus the people who live in a house.
5. Memory operations are called fetch and store; introduce the terms **nondestructive fetch** and **destructive store**. RAM takes a fixed time to fetch or store to every memory cell: **memory access time**, currently 5–10 **nanoseconds**. To change a single bit of memory, the whole cell must first be fetched and then rewritten after the change.
6. Memory registers are used to perform fetch and store operations. The **memory address register (MAR)** holds the address to operate on; the **memory data register (MDR)** receives the data being fetched or stored. Discuss the use of decoder circuits to convert the value in the MAR to a signal to a specific memory circuit. The **fetch/store controller** ensures that data flows either into the MDR or from the MDR.
7. Discuss different speeds of memory and why they are needed: RAM, registers, and **cache memory**. Cache memory stores values based on the **principle of locality**. The **cache hit rate** is the percentage of times a memory value is found in cache memory, rather than requiring an access of regular RAM.

**Quick Quiz 1**

1. \_\_\_\_\_\_\_\_\_\_\_ is the unique number associated with each cell in a random access memory.   
   Answer: An address
2. (True or false) Integers inside most computers are limited to the values that can fit in one memory cell.

Answer: False

1. (True or false) A destructive store operation destroys the contents of a memory cell.

Answer: True

1. In order to perform fetch and store operations, name two circuits besides memory cells that a RAM must have.

Answer: A decoder (to convert the address to a signal to a specific cell) and a fetch/store controller (to ensure data flows the right way)

1. How many controller units are used to select the correct row and column in a fetch command?

Answer: 2

1. Higher-speed memory used to store values likely to be accessed is called \_\_\_\_\_\_\_\_\_\_\_.

Answer: cache memory

**5.2.2 Input/Output and Mass Storage**

1. Introduce the term **input/output (I/O)** and emphasize that I/O devices include those intended to communicate with humans (keyboards, screens, and speakers), those used to communicate with external data store, and those used to communicate with other computers.
2. Introduce the terms **volatile memory** and **nonvolatile memory** to highlight the importance of **mass storage systems.**
3. There are two important kinds of mass storage systems: **direct access storage devices (DASDs)** and **sequential access storage devices (SASDs)**.
4. Hard drives, CDs, and DVDs are direct access storage devices. Drives divide the disk surface into **tracks** and **sectors** and retrieve one sector at a time. Sectors are addressed, much like cells in RAM, but each addressed section is much larger. Time to retrieve data is based on three factors: **seek time**, **latency**, and **transfer time**. Sequential access storage devices are much less common and are typically used for making backup copies of memory or drive data.

|  |  |
| --- | --- |
| ***Teaching Tip*** | Bring an old hard drive into the classroom—ideally one that can still function. Take the drive apart, and demonstrate its function to the class. Use it as a specific example for discussing access speed. |

1. Sequential access storage devices do not require that all units of data be identifiable via unique addresses. The whole of the data must be searched with the question, “Is this what I’m looking for?” If not, move on to the next unit of data and ask again until the data is found, or not found. Refer students to cassette tapes of the 1980s and early 1990s as an example. To find a song, you had to fast-forward through the data to get to the right place.
2. Introduce the term **I/O controller** and its importance as intermediary between the “slow” I/O devices (on the order of milliseconds or microseconds, not nanoseconds) and the rest of the computer. The I/O controller signals the process or when an I/O operation is done using an **interrupt signal**. Use an anthropomorphic metaphor of a busy person handing a tedious task to a friend—the interrupt signal could be a text, phone call, or raising a flag.

**5.2.3 The Arithmetic/Logic Unit**

1. Introduce the term **arithmetic/logic unit (ALU)**, and note it is part of the **central processing unit**. The ALU contains circuits to perform arithmetic operations and also comparisons and other logical operations. The **data path** is how information flows to **registers**, through the ALU circuitry to other registers, and ultimately out of the ALU.
2. Registers are very fast memory and have a specific purpose: often connected to specific circuits.
3. The path for electrical signals in a computer is called a **bus**. Emphasize how a bus works with the ALU.
4. Most ALU designs feed data from a register to all operation circuits and use a multiplexer to select the desired answer.

**5.2.4 The Control Unit**

1. Introduce the term **control unit** and its importance for implementing the **stored program** characteristic of the Von Neumann architecture. The control unit fetches instructionsfrom memory, decodes them, and executes them.
2. Introduce the term **machine language**, and describe their typical format. Mention that in very early days, computer programmers had to write their programs in binary.
3. Introduce the term **instruction set**: the set of instructions a given processor has been constructed to execute. Note that this is why you must download software designed for your specific machine: different processors speak different languages.
4. Describe two different approaches to instruction set design: **RISC machines** and **CISC machines**. Many modern systems have aspects of both RISC and CISC design.
5. Instruction sets contain four kinds of instructions: data transfer, arithmetic, comparison, and branch. The examples in the text are in the form that will be used in later chapters.
6. Remind students of the task of a control unit, and describe the basic components: the instruction decoder circuit and two registers: the **program counter (PC)** and the **instruction register (IR)**. Describe the purpose of each register and how they relate to the MAR and MDR from earlier. The instruction decoder is a decoder circuit that operates on the op code portion of the instruction in the IR.

**Quick Quiz 2**

1. What does the ALU do?  
   Answer: Performs arithmetic and logical operations
2. (True or false) Different computer chips, processors, use different machine language.

Answer: True

1. Give two of the kinds of instructions most instructions sets contain.

Answer: Data transfer, arithmetic, comparison, or branch

1. (True or false) The instruction register (IR) holds the address of the cell in memory where the next instruction is stored.

Answer: False

1. All data transfer instructions follow what principle?

Answer: Nondestructive fetch/destructive store

**5.3 Putting the Pieces Together—the Von Neumann Architecture**

1. Display Figure 5.24 that shows the whole Von Neumann machine; emphasize to students that they should now know how each part of that diagram works. Remind students that in most cases the ALU and the control unit are combined into the **central processing unit**.
2. Discuss the Von Neumann cycle: fetch/decode/execute. Emphasize that, while the cycle may be described algorithmically, the computer must implement the actual cycle in its hardware. Go over the detailed steps of the cycle for several specific example instructions; then ask your students to do the same for one or two additional instructions, using the same notation.

|  |  |
| --- | --- |
| ***Teaching Tip*** | Have students act out the pieces of the Von Neumann architecture, taking on roles as control, ALU, memory, and/or even I/O controller circuitry. |

**5.4 Non–Von Neumann Architectures**

1. Explain Moore’s Law and the problems with it in recent years: harder to place more gates on a circuit, problem sizes growing, and heat dissipation an increasing issue. Introduce the term **Von Neumann bottleneck**.
2. Introduce the term **non–Von Neumann architectures**. Explain that, while much new architecture is studied in research, **parallel processing** has been around for many years and is currently growing in importance.
3. Introduce the term **SIMD parallel processing**, where only the ALU is duplicated. It is particularly useful for **vector** processing, and was the dominant form for supercomputers starting in the 1980s.
4. Introduce the term **MIMD parallel processing**, also called **cluster computing**—more commonly used in recent years. Processors are duplicated in this model. MIMD parallelism is easily scalable; the downside is the need for communications over an interconnection network. **Grid computing** extends the MIMD model to different kinds of computers scattered across the Internet.
5. Introduce the term **parallel algorithms** as a separate field of study in computer science focused on the study of techniques that make efficient use of parallel architectures.

**Quick Quiz 3**

1. A parallel architecture where the ALU is duplicated many times is called \_\_\_\_\_\_\_\_\_\_\_\_.  
   Answer: SIMD
2. (True or false) BIONIC is an example of grid computing, a type of MIMD parallel processing.

Answer: True

1. The problem of sequential processors being unable to perform very large computations in a reasonable time is called \_\_\_\_\_\_\_\_\_\_\_\_.

Answer: the Von Neumann bottleneck

1. (True or false) The problem with MIMD is that once implemented, it is not scalable.

Answer: False

# **Class Discussion Topics**

1. Think about examples of abstractions we use with the computer on a regular basis. What are some metaphors provided by applications or your operating system?
2. Compare RAM with nonvolatile memory like the computer’s hard drive. List the features of each, and compare them with each other? Where they differ in design, why would the designers have made the choices they made?
3. Why might the MIMD model for parallel processing be preferred over the SIMD model?

# **Additional Projects**

1. Describe in a step-by-step way what would happen within the RAM memory system if we wanted to store the binary byte 11011000 to memory cell 452. What if we wanted to fetch the value from cell 102, add one to it, and store the value back into 102?
2. Using the notations from pages 254 and 255, write out the execute phase steps for the JUMP X instruction.
3. List the places in the processor where circuits described in Chapter 4 occur and what they are used for.

# **Additional Resources**

1. Web page and video that explain how flash memory works: <http://computer.howstuffworks.com/flash-memory.htm> <http://www.youtube.com/watch?v=kNGvZGz7dmE>
2. Intel has an online curriculum that talks about hardware and processor design, among other topics. While designed for K–12, it could be useful here as well: <http://educate.intel.com/en/TheJourneyInside>
3. The Computer History Museum has an online exhibit about supercomputers: <http://www.computerhistory.org/revolution/supercomputers/10/intro>
4. Download Berkley’s BIONIC software and become part of a MIMD:

http://boinc.berkeley.edu

**Key Terms**

* **Address space (max memory size):** The maximum amount of memory that a computer can physically hold. It is determined by the size of the address field.
* **Arithmetic/logic unit (ALU):** A computer subsystem that performs such mathematical and logical operations as addition, subtraction, and comparison for equality.
* **Cache hit rate:** The percentage of the time that the information needed is in cache memory.
* **Cache memory:** A special high-speed memory unit that keeps a copy of memory cells with a high likelihood of access in the near future.
* **CISC machine:** Complex instruct set computer; a machine that has a very large and complex instruction set.
* **Cluster computing:** In which independent systems such as mainframes, desktops, or laptops are interconnected by a local area network (LAN) like the Ethernet or a wide area network (WAN) such as the Internet; also called MIMD parallel processing.
* **Computer organization:** The branch of computer science that studies computers in terms of their major functional units and how they work.
* **Control unit:** The computer subsystem that fetches and executes instructions stored in the memory of the computer.
* **Data path:** The ALU circuits, registers, and interconnections between components.
* **Destructive store:** When you change the contents of a memory cell, you destroy its previous contents and replace it with the new contents.
* **Direct access storage devices:** A mass storage device in which each unit of information is associated with a unique address, but the time to access each piece of information may not be the same.
* **Fetch/store controller:** The component that determines whether a value will be placed into memory or taken out of memory.
* **Functional units:** Subunits of a computer that perform tasks such as instruction processing, information storage, computation, and data transfer.
* **Grid computing:** A MIMD model in which the individual processors can be computer systems belonging to a wide range of groups or individuals.
* **Hierarchy of abstractions:** A series of abstractions, each one more detailed and each one showing lower level components of a system.
* **I/O controller:** A special-purpose device that controls the operations of an input/output device.
* **Input/output (I/O) device:** The device that allows a computer system to communicate and interact with the outside world as well as store information.
* **Instruction register (IR):** The register that holds a copy of the instruction to be executed.
* **Instruction set:** The set of all operations that can be executed by a processor.
* **Interrupt signal:** A signal sent by the I/O controller to the CPU to let it know that it has completed an I/O operation.
* **Latency:** The time required to rotate the disk to the beginning of the desired sector.
* **Level of abstraction:** A specific way to view a system.
* **Machine language:** The programming language that a processor is directly able to understand and execute. It is written in binary.
* **Mass storage systems:** Systems or devices where information is kept for long periods of time and not lost when the computer is not being used.
* **Memory:** The functional unit of a computer that stores and retrieves the instructions and the data being executed.
* **Memory access time:** The time it takes to fetch or store the contents of a single memory cell.
* **Memory address:** The unique numeric identifier for a memory cell.
* **Memory address register (MAR):** The memory register that holds the address of the cell to be fetched or stored.
* **Memory cell:** The minimum unit of memory access.
* **Memory data register (MDR):** The memory register that holds the data value to be stored or the data value that was just fetched.
* **Memory width (cell size):** The number of bits in a single memory cell.
* **MIMD parallel processing:** A parallel processing model in which multiple processors all work independently on their own program to solve a single problem; MIMD stands for “multiple instruction stream/multiple data stream”; also called cluster computing.
* **Nanosecond:** One-billionth of a second.
* **Nondestructive fetch:** When you access the contents of a memory cell you only copy it; you do not destroy it.
* **Non–Von Neumann Architectures:** Computer designs based on models other than the standard Von Neumann architecture.
* **Nonvolatile memory:** Memory that does not lose its contents even when the power is turned off.
* **Parallel algorithms:** Algorithms that exploit the presence of multiple processors to solve a single problem.
* **Parallel processing:** Building computers with two or more processors that work in parallel.
* **Principle of locality:** When you access a memory cell, the likelihood is that you will access memory cells with addresses very close to this one in the near future.
* **Processor:** A system that is composed of the ALU together with the control unit.
* **Program counter (PC):** The register that holds the address of the next instruction to be executed.
* **Quantum computing:** A field of computer design using the principles of quantum mechanics in which a single bit of information can be not just a 0 or a 1 but in both states at the same time.
* **Random access memory (RAM):** A memory structure in which each cell has an address, and it takes the same amount of time to fetch or store any cell.
* **Read-only memory (ROM):** A memory structure that can only be accessed, not written into or changed.
* **Register:** A special, high-speed storage cell.
* **RISC machine:** Reduced instruction set computer; a machine that has a very small and simple instruction set but where each instruction is highly optimized and executes very quickly.
* **Sector:** A disk storage unit containing an address, a data block, and a fixed number of bytes. Sectors are arranged in concentric tracks on a disk.
* **Seek time:** The time required to move the read/write head to the correct track.
* **Sequential access storage device:** A mass storage device in which information is located by sequentially searching all the information that is stored.
* **Sequential execution:** When one instruction at a time is fetched from memory to the control unit, where it is decoded and executed.
* **SIMD parallel processing:** A parallel processing model in which multiple processors all execute the same instruction on their own local data; SIMD stands for “single instruction stream/multiple data stream.”
* **Stored program:** The instructions to be executed by the computer are represented as binary values and stored in memory.
* **Track:** A single concentric circle of information on a disk.
* **Transfer time:** The time required to read the desired sector into main memory.
* **Vector:** An ordered collection of values.
* **Volatile memory:** Memory that loses its contents when the power is turned off.
* **Von Neumann architecture:** The computational model designed by John Von Neumann and first implemented in the EDSAC computer of 1947; the structure and organization of virtually all modern computers.
* **Von Neumann bottleneck:** The inability of sequential, one-at-a-time processors to handle extremely large problems in a reasonable time scale.

**Solutions to End-of-Chapter Exercises**

**1.** The advantage of using a very large memory cell size is that the computer can store larger (in terms of absolute value) values in its memory cells. The disadvantage is that there would be fewer cells available, and with such big cells there would be a lot of wasted space when storing small numbers. The largest positive integer that could be stored in a system using sign/magnitude notation, with 64-bit cells is 263. If two such cells were used to store integers, the largest integer that could be stored by the system is 2127.

**2.** a. 20 bits (220 = 1,048,576, just over one million)

b. 24 bits

c. 27 bits

d. 30 bits

**3.** A memory unit with 640 KB has 640 × 210 memory cells, provided that the computer uses 8-bit cells. A memory unit with 512 MB would contain 512 × 220 memory cells. A memory unit with 2 GB would contain 2 × 230 cells.

**4.** Read-only memory can be used to store essential system instructions that users cannot overwrite. The information could originally be stored in the ROM by hardwiring the electronic circuitry.

**5.** The dimensions of a memory containing 1 MB of storage would be 1024 × 1024. The MAR would have to be 20 bits large. 10 bits would be sent to each of the row and column decoders, and the decoders would each have 1024 output lines.

**6.** The maximum size of the memory unit on this machine is 224. The dimensions of the two-dimensional memory are 212 × 212 or 4096 × 4096.

**7.** One could store 4 MB of memory using a 20 bit MAR by thinking of our 4 MB memory as being composed of four separate 1 MB units. Then if we had a separate 2-bit register that specified which of the four units we were accessing, we could use the 20 bit MAR to access into that one particular memory unit. In a sense, the new 2-bit register could be thought of as the two high-order bits of a 22-bit MAR.

**8.** We would have to make two fetches to get the complete instruction. Alternately, we might fetch only the first 16 bits, which could be the operation code, and then decode that part of the instruction. Then, during the execution phase, we might go back to memory to fetch the last 16 bits, which could be the address of the operand on which we are going to work. Basically, we would have to either extend the fetch phase or do part of the fetch in the execution cycle.

**9.** One page = 65 × 60 = 3900 characters.

3900 characters = 5 seconds

780 characters/second

A 1 gigaflop machine does 1,000,000,000 floating point operations per second, so in that time it can perform

5 × 1,000,000,000 = 5,000,000,000 floating point operations

**10.** One would need a multiplexor circuit that contains 5 selector lines because 25 = 32 is sufficient to identify one of 20 lines. With only 4 selector lines, we would not have enough selectivity because 24 = 16.

**11.** Major advantages: 1) Since each instruction does a lot more, the size of the programs will be less, and 2) the programmers will have an easier time since they are working with more sophisticated and powerful instructions.

Major disadvantage: The circuitry will be much bigger and more complex, taking up far more room on the chip. This will leave much less room to do things like optimization.

**12.** a. The number of characters per disk = 1024 char/sector × 20 sectors/track × 500 tracks/surface × 2 surfaces = 20,480,000 characters.

b. 7200 rev/min = 120 rev/sec or 1/120 = 0.00833 sec/rev = 8.33 msec/rev.   
 20 sectors per track = 20 sectors/rev or 1/20 = 0.05 rev/sector.   
 Finally,

8.33 msec/rev × 0.05 rev/sector = 0.4165 msec/sector

This is the transfer rate, which is a constant.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Best | Worst | Average |
| Seek Time | 0 | 0.5 + 499 × 0.05 = 25.45 | 0.5 + 150 × 0.05 = 8 (assume average move is 150 tracks) |
| Latency | 0 | 8.33  (one complete revolution) | 4.165  (one half revolution) |
| Transfer | 0.4165 | 0.4165 | 0.4165 |
| Total | 0.4165 | 34.1965 | 12.5815 |

**13.** Since we are changing the rotational speed, the latency and transfer times will change but the seek times remain the same.

9600 rev/min = 160 rev/sec or 1/160 = 0.00625 sec/rev = 6.25 msec/rev.   
 6.25 msec/rev × 0.05 rev/sector = 0.3125 msec/sector (constant transfer rate)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Best | Worst | Average |
| Seek Time | 0. | 0.5 + 499 × 0.05 = 25.45 | 0.5 + 150 × 0.05 = 8 (assume average move is 150 tracks) |
| Latency | 0 | 6.25  (one complete revolution) | 3.125  (one half revolution) |
| Transfer | 0.3125 | 0.3125 | 0.3125 |
| Total | 0.3125 | 32.0125 | 11.4375 |

**14.** How can we minimize access time to this data? Latency time and transfer time cannot be improved but we can reduce the time that we spend moving the disk arm to the correct track. To do this we should store the information to first fill one track and then the corresponding track on the other side of the disk. This will store 20 × 1024 × 2 = 40,960 bytes. Store the remaining information on an adjacent track. This will minimize seek time to access this information once the initial track is located.

**15.** Having a read/write head per track eliminates seek time in all cases.

**16.** If we assume that the arm is initially positioned at the beginning of the first sector of the first track on the disk then we would have to perform the following operations:

a. read the entire first track on both sides. Since the disk is double sided, we will need to spend two revolutions to read each track—the first revolution to read the top surface and the second revolution to read the bottom.

b move the arm inward one track and wait for the beginning of the track

c. repeat steps a and b 499 more times to read every track on the disk.

Step a takes two revolutions, which is 2 × 8.33 msec = 16.66 msec

Step b takes 0.55 msec to move the arm, but this takes place while the disk continues to rotate, so the 0.55 msec is absorbed in the worst case of waiting for one complete revolution for the beginning of the track to rotate around under the head. Therefore, step b takes a total of 8.33 msec.

So, the total time will be 16.66 + 499 (8.33 + 16.66) = 12.48 seconds

**17.** A sequential access storage device such as a tape could be a useful form of mass storage when one is sequentially copying the memory of a disk drive to back up or archive a system.

**18.** 56000 bits are being transferred per second so the transfer time is 1/56000 sec/bit. The number of instructions executable between bits is

500,000,000 instructions/sec × 1/56000 sec/bit = 8928 instructions/bit

**19.** a. The maximum number of distinct operation codes that can be recognized and executed by the processor on this machine is 26 = 64

b. The max memory size on this machine is 218

c. The number of bytes required for each operation is 6 (48 bits), since the IR is   
 6 + 18 + 18 = 42 bits.

**20.** With an op code field of 8 bits we could theoretically have 28 = 256 operations.

**21.** For reference: v-200 w -201 x - 202 y -203 z-204

a. set v to x - y + z (assume command SUBTRACT X,Y,Z exists which is z = x - y)

|  |  |  |  |
| --- | --- | --- | --- |
| *Memory Location* | *Op Code* | *Address Field* | *Comment* |
| 50 | SUBTRACT | 202, 203, 200 | v now has the value x - y |
| 51 | ADD | 200, 204, 200 | v now has the value x – y + z |

b. set v to value of (w + x) - (y + z)

|  |  |  |  |
| --- | --- | --- | --- |
| *Memory Location* | *Op Code* | *Address Field* | *Comment* |
| 50 | ADD | 201, 202, 200 | v now has the value w + x |
| 51 | ADD | 203, 204, 201 | w now has the value y + z |
| 52 | SUBTRACT | 200, 201, 200 | v now has the value (w + x) – (y + z) |

c. if (v ≥ w) then

set x to y

else

set x to z

|  |  |  |  |
| --- | --- | --- | --- |
| *Memory Location* | *Op Code* | *Address Field* | *Comment* |
| 50 | COMPARE | 200, 201 | Compare v and wand set condition codes |
| 51 | JUMPGE | 54 | If v ≥ w go to address 54 |
| 52 | MOVE | 204, 202 | Otherwise give x the value of z |
| 53 | JUMP | 55 | Go to address 55 |
| 54 | MOVE | 203, 202 | Give x the value of y |
| 55 |  |  | The next instruction begins here |

d. while y < z do

set y to value (y + w + z)

set z to value (z + v)

end of loop

|  |  |  |  |
| --- | --- | --- | --- |
| *Memory Location* | *Op Code* | *Address Field* | *Comment* |
| 50 | COMPARE | 203, 204 | Compare y and zand set condition codes |
| 51 | JUMPLT | 53 | If y < z go to address 53 |
| 52 | JUMP | 57 | Otherwise, go to address 57 |
| 53 | ADD | 203, 201, 203 | y now contains the value y + w |
| 54 | ADD | 203, 204, 203 | y now contains the value  y + w + z |
| 55 | ADD | 204, 200, 204 | z now contains the value  z + v |
| 56 | JUMP | 50 | Go to address 50 |
| 57 |  |  | The next instruction begins here |

**22**. a.

|  |  |  |  |
| --- | --- | --- | --- |
| *Memory Location* | *Op Code* | *Address Field* | *Comment* |
| 50 | LOAD | 300 | Register R now contains the value of a |
| 51 | ADD | 301 | R now contains the sum a + b |
| 52 | ADD | 401 | R now contains the sum a + b – 1 |
| 53 | STORE | 300 | Store the result into a |

b.

|  |  |  |  |
| --- | --- | --- | --- |
| *Memory Location* | *Op Code* | *Address Field* | *Comment* |
| 50 | COMPARE | 300, 402 | Compare a and 0 and set condition codes |
| 51 | JUMPLE | 54 | If a ≤ 0 go to address 54 |
| 52 | LOAD | 400 | Otherwise, register R now contains the value 1 |
| 53 | STORE | 301 | Store 1 into b |
| 54 |  |  | The next instruction begins here |

**23.** The error is on the second line. The instruction ADD 19 does not add the integer 19 to the value of y but, instead, the *contents* of memory cell 19. To correct this error, you would need to store the constant +19 into a memory cell, say location 600, and then write the instruction ADD 600.

**24.** a. MOVE X, Y

1. IRaddr1 → MAR Send address of X, currently in IRaddr1, to the MAR

2. FETCH Fetch the contents of cell X and put in the MDR

3. IRaddr2 → MAR Send address of Y, currently in IRaddr2, to the MAR

4. STORE Store the value in the MDR into memory cell Y

b. ADD X, Y

1. IRaddr1 → MAR Send address of X, currently in IRaddr1, to the MAR

2. FETCH Fetch the contents of cell X and put in the MDR

3. MDR → ALU Send contents of MDR to the ALU

4. IRaddr2 → MAR Send address of Y, currently in IRaddr2, to the MAR

5. FETCH Fetch the contents of cell Y and put in the MDR

6. MDR → ALU Send contents of MDR to the ALU

7. ADD Activate ALU and select output of add circuit

8. ALU → MDR Copy output of the ALU to the MDR

9. STORE Store the value in the MDR into memory cell Y, whose address is still in the MAR

**25.** ADD X, v, Y

1. IRaddr1 🡪 MAR Send address of X, currently in IRaddr1, to the MAR
2. FETCH Fetch the contents of cell X and put in the MDR
3. MDR 🡪 ALU Send contents of MDR to the ALU
4. IR addr2🡪 ALU Send the second address field directly into the ALU

without fetching since it is a constant, not an address

1. ADD Activate ALU and select output of add circuit; the ALU   
    now holds CON(X)+v
2. ALU 🡪 MDR Copy ALU to the MDR. MDR now holds CON(X)+v
3. IRaddr3 🡪 MAR Send address of Y, currently in IRaddr3, to the MAR
4. STORE Store the value CON(X)+v in the MDR into

memory cell Y which is in the MAR

1. What makes the SETI project so well suited for grid computing is that the data can be easily divided into a large number of independent subsets, each of which can be analyzed in its entirety without having to communicate with other systems. When Arecibo sends out a chunk of radio telescope data, that data can be analyzed by a single computer to determine whether it does or does not contain the special pattern for which it is searching. When it is done, it only needs to communicate a single word message: Yes (if it found something) or No (it did not). That independence makes it an ideal research project to divide up into the hundreds of thousands of pieces of data that will be sent out to the hundreds of thousands of computers that are part of BOINC.

**Challenge Work**

**1.** With 100 processors, one can perform up to 100 addition operations at each step. One possible algorithm will use up to 50 processors simultaneously.

In the while loop:

While i < 101 do the following

instead of

Sum = Sum + ai,

one could perform 50 operations simultaneously

(sum1 = a1 + a2

sum2 = a3 + a4

sum3 = a5 + a6

.

.

.

sum50 = a99 + a100)

resulting in 50 new sums.

On iteration 2, one could perform 25 operations simultaneously.

(secondsum1=sum1+sum2

secondsum2=sum3+sum4

.

.

.

secondsum25=sum49+sum50)

resulting in 25 new sums.

Iteration 3 adds 12 pairs giving 12 new sums plus 1 remaining from the previous level.   
Iteration 4 adds 6 pairs giving 6 new sums plus 1 remaining from the previous level.  
Iteration 5 adds 3 pairs giving 3 new sums plus 1 remaining from the previous level.  
Iteration 6 adds 2 pairs giving 2 new sums.  
Iteration 7 adds 1 pair, giving the final result.

Instead of 100 iterations, 7 iterations will be needed, which means the algorithm with the parallel processing is over 14 times faster. This is logarithmic time instead of the linear time required to do the job sequentially.

No more than 50 processors were needed and any more than 50 processors would be too many for this specific problem.

**2**. Students should choose a processor and use the Internet to find information regarding the specific architecture chosen, although it may be difficult to find some of these details. Stress writing skills here so that you get a report, not merely a list of attributes.