BLM5207 Computer Organization

Prof. Dr. Nizamettin AYDIN
naydin@yildiz.edu.tr
http://www3.yildiz.edu.tr/~naydin

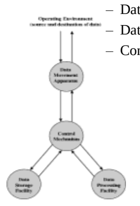
Introduction to a Simple Computer

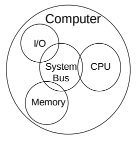
1

1

What is a computer?

- In terms of what?
  - Functional
    - All computer functions are:
      - Data processing
      - Data storage
      - Data movement
      - Control
  - Structural
    - Corresponding computer components are:
      - CPU
      - Memory
      - I/O
      - System interconnection

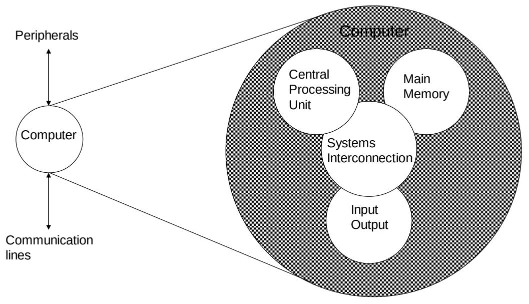




2

2

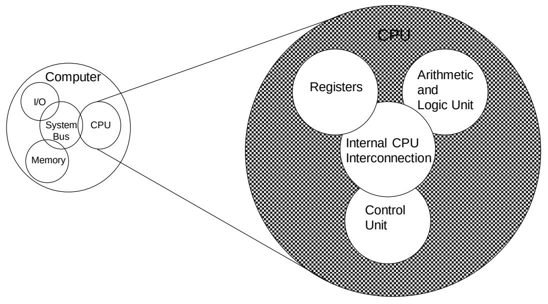
Structure - Top Level



3

3

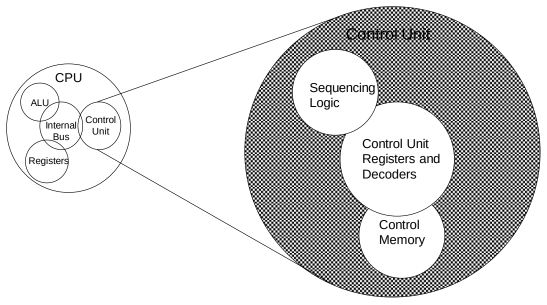
Structure - The CPU



4

4

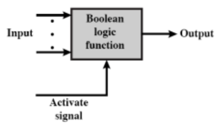
Structure - The Control Unit



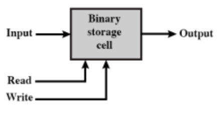
5

5

Fundamental computer elements



(a) Gate



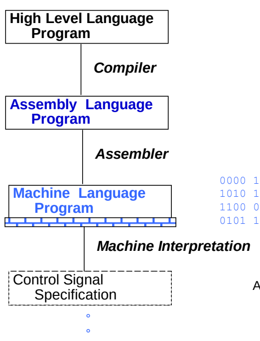
(b) Memory cell

- Data storage:
  - Provided by memory cells
- Data processing:
  - Provided by gates
- Data movement:
  - The paths between components are used to move data from/to memory
- Control:
  - The paths between components can carry control signals

6

6

Levels of Representation


High Level Language Program
Compiler
Assembly Language Program
Assembler
Machine Language Program
Machine Interpretation
Control Signal Specification
°
°

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

```
lw    $15,   0($2)
lw    $16,   4($2)
sw    $16,   0($2)
sw    $15,   4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

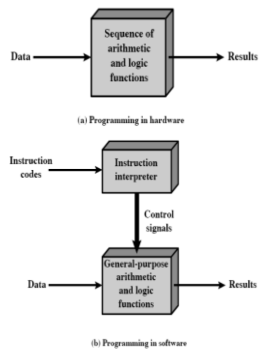
```
ALUOP[0:3] <= InstReg[9:11] & MASK
```

7

7

Program Concept

- Programming in hardware
  - Hardwired systems are inflexible
- Programming in software
  - General purpose hardware can do different tasks, given correct control signals
  - Instead of re-wiring, supply a new set of control signals


Data
Sequence of arithmetic and logic functions
Results
(a) Programming in hardware
Instruction codes
Instruction interpreter
Control signals
Data
General-purpose arithmetic and logic functions
Results
(b) Programming in software

8

8

What is a program?

- A sequence of steps
- For each step, an arithmetic or logical operation is done
- For each operation, a different set of control signals is needed
- For each operation a unique code is provided
  - e.g. ADD, MOVE
- A hardware segment accepts the code and issues the control signals
- We have a computer!

9

9

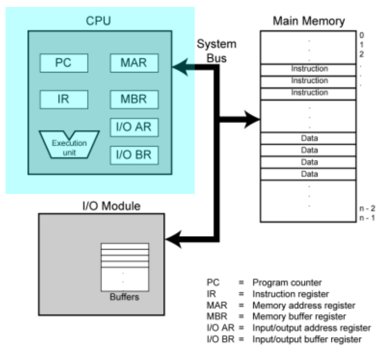
Components

- The Central Processing Unit contains
  - The Control Unit
  - The Arithmetic and Logic Unit
- Data and instructions need to get into the system and results out
  - Input/output
- Temporary storage of code and results is needed
  - Main memory

10

10

Computer Components-CPU

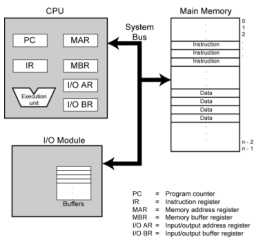


11

11

CPU Structure

- CPU must:
  - Fetch instructions
  - Interpret instructions
  - Fetch data
  - Process data
  - Write data

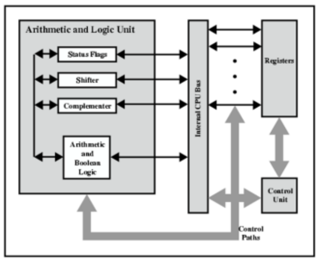

CPU
PC
MAR
IR
MBR
I/O AR
I/O BR
Execution unit
System Bus
Main Memory
Instruction
Instruction
Instruction
Data
Data
Data
Data
I/O Module
Buffers
PC = Program counter
IR = Instruction register
MAR = Memory address register
MBR = Memory buffer register
I/O AR = Input/output address register
I/O BR = Input/output buffer register

12

12

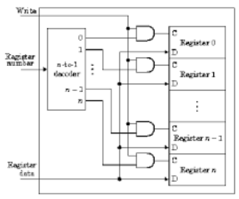
CPU Structure

- CPU must:
  - Fetch instructions
  - Interpret instructions
  - Fetch data
  - Process data
  - Write data



Registers

- CPU must have some working space (temporary storage)
  - Called registers



- Manipulated directly by the Control Unit
- Number and function vary between processor designs
- One of the major design decisions
- Size in bits or bytes (not MB like memory)
- Can hold data, an address or an instruction
- Top level of memory hierarchy

Registers in the µP perform two roles:

- User-visible registers
    - Enable the machine- or assembly language programmer to minimize main memory references by optimizing use of registers
  - General Purpose registers
  - Data registers
  - Address registers
  - Condition Code Registers (flags)
    - Sets of individual bits
      - e.g. result of last operation was zero
    - Can be read (implicitly) by programs
      - e.g. Jump if zero
    - Can not (usually) be set by programs

Registers in the µP perform two roles:

- Control and status registers
    - Used by the control unit to control the operation of the processor and by priviliged, operating system programs to control the execution of programs
  - Program Counter (PC)
    - Also called instruction pointer
    - Contains the address of an instruction to be fetched
  - Instruction Decoding Register (IR)
    - Stores instruction fetched from memory
  - Memory Address Register (MAR)
    - Contains the addres of location in memory
  - Memory Buffer Register (MBR)
    - Also called Memory Data Register (MDR)
    - Contains a word or data to be written to memory or the word most recently read

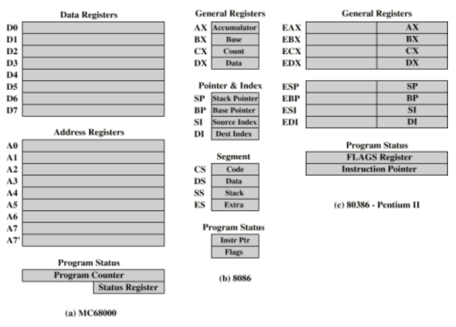
Program Status Word (Status Registers)

- A set of bits containing status information
- Includes Condition Codes (flags)
  - Sign
    - sign of last result
  - Zero
    - set when the result is 0
  - Carry
    - set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high order bit
  - Equal
    - set if a logical compare result is equality
  - Overflow
    - used to indicate arithmetic overflow
  - Interrupt enable/disable
    - used to enable or disable interrupts

Register Operations

- Stores values from other locations such as
  - registers and memory
- Addition and subtraction
- Shift or rotate data
- Test contents for conditions such as zero or positive

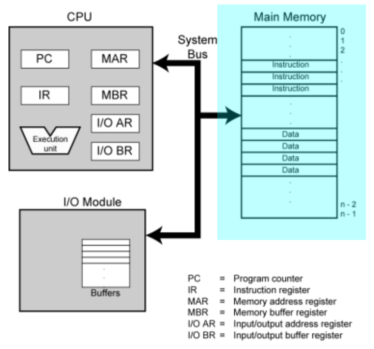
Example Register Organizations



19

19

Computer Components-Memory

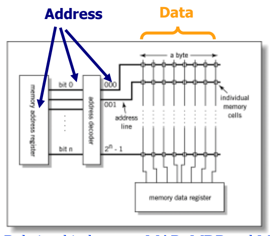


20

20

Operation of Memory

- Each memory location has a unique address
- Address from an instruction is copied to the MAR which finds the location in memory



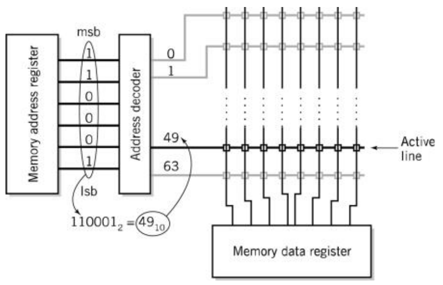
- CPU determines if it is a store or retrieval
- Transfer takes place between the MDR and memory
- MDR is a two-way register

Relationship between MAR, MDR and Memory

21

21

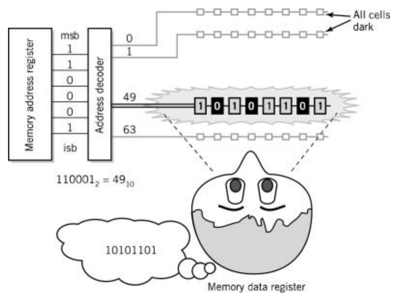
MAR-MDR Example



22

22

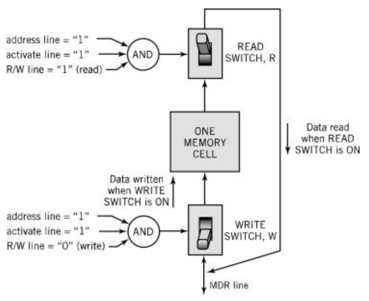
Visual Analogy of Memory



23

23

Individual Memory Cell



24

24

Memory Capacity

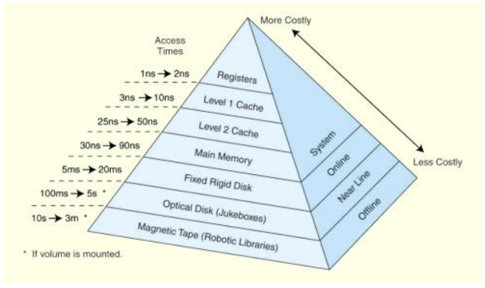
- Determined by two factors
  1. Number of bits in the MAR
     - 2^K where K = width of the register in bits
  2. Size of the address portion of the instruction
     - 4 bits allows 16 locations
     - 8 bits allows 256 locations
     - 32 bits allows 4,294,967,296 or 4 GB
- Important for performance
  - Insufficient memory can cause a processor to work at 50% below performance

25

25

Memory Hierarchy

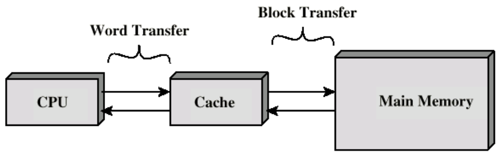
- This storage organization can be thought of as a pyramid:



26

26

Cache



- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module

27

27

Virtual Memory

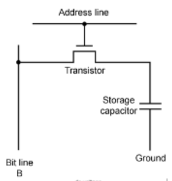
- Cache memory enhances performance by providing faster memory access speed.
- Virtual memory enhances performance by providing greater memory capacity, without the expense of adding main memory.
  - Instead, a portion of a disk drive serves as an extension of main memory.
- If a system uses paging, virtual memory partitions main memory into individually managed page frames, that are written (or paged) to disk when they are not immediately needed.

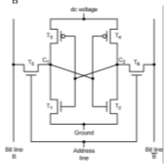
28

28

RAM: Random Access Memory

- DRAM (Dynamic RAM)
  - Most common, cheap
  - Volatile:
    - must be refreshed (recharged with power) 1000's of times each second
- SRAM (static RAM)
  - Faster than DRAM and more expensive than DRAM
  - Volatile
    - Frequently small amount used in cache memory for high-speed access used





29

29

ROM - Read Only Memory

- Permanent storage
  - Non-volatile memory to hold software that is not expected to change over the life of the system
- Used in...
  - Microprogramming
  - Library subroutines
  - Systems programs (BIOS)
    - initial boot instructions and diagnostics
  - Function tables

30

30

Types of ROM

- Written during manufacture
  - Very expensive for small runs
- Programmable (once)
  - PROM
  - Needs special equipment to program
- Read “mostly”
  - Erasable Programmable (EPROM)
    - Erased by UV
  - Electrically Erasable (EEPROM)
    - Takes much longer to write than read
  - Flash memory
    - Erase whole memory electrically

31

31

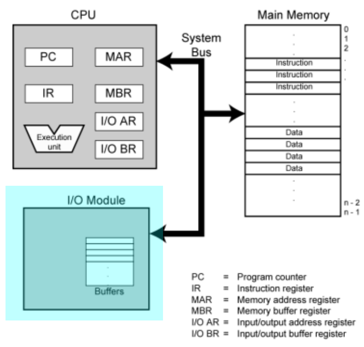
Types of External Memory

- SSD
  - Fast
  - Expensive (relatively)
- Magnetic Disk
  - RAID
  - Removable
- Optical
  - CD-ROM
  - CD-Recordable (CD-R)
  - CD-R/W
  - DVD
- Magnetic Tape

32

32

Computer Components-I/O

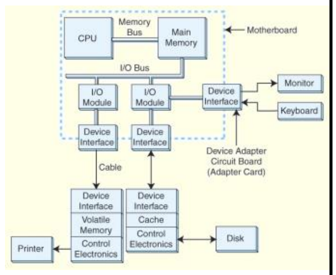


33

33

Input/Output Problems

- Wide variety of peripherals
  - Delivering different amounts of data
  - At different speeds
  - In different formats
- All slower than CPU and RAM
- Need I/O modules

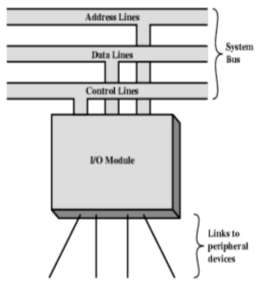


34

34

Input/Output Module

I/O Module Block Diagram



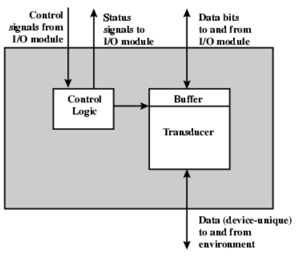
- Interface to CPU and Memory
- Interface to one or more peripherals
- I/O Module Function:
  - Control & Timing
  - CPU Communication
  - Device Communication
  - Data Buffering
  - Error Detection

35

35

External Devices

External Device Block Diagram


Control signals from I/O module
Status signals to I/O module
Data bits to and from I/O module
Control Logic
Buffer
Transducer
Data (device-unique) to and from environment

- External Devices:
  - Human readable
    - Screen, printer, keyboard
  - Machine readable
    - Monitoring and control
  - Communication
    - Modem
    - Network Interface Card (NIC)

36

36

I/O Steps

- CPU checks I/O module device status
- I/O module returns status
- If ready, CPU requests data transfer
- I/O module gets data from device
- I/O module transfers data to CPU
- Variations for output, DMA, etc.

37

37

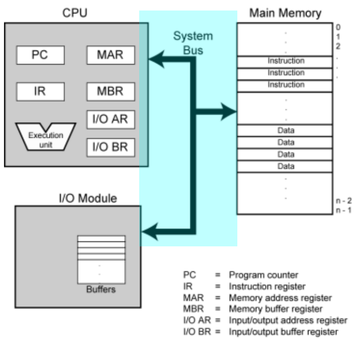
I/O Architectures

- I/O can be controlled in four general ways:
  - Programmed I/O
    - Reserves a register for each I/O device.
    - Each register is continually polled to detect data arrival.
  - Interrupt-Driven I/O
    - Allows the CPU to do other things until I/O is requested.
  - Direct Memory Access (DMA)
    - Offloads I/O processing to a special-purpose chip that takes care of the details.
  - Channel I/O
    - Uses dedicated I/O processors.

38

38

Computer Components- Bus

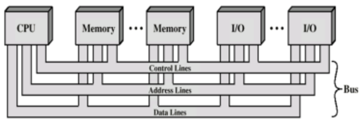


39

39

Bus

- The physical connection that makes it possible to transfer data from one location in the computer system to another
- Group of electrical conductors for carrying signals from one location to another
- 4 kinds of signals
  - Data
    - Alphanumeric
    - Numerical
    - instructions
  - Addresses
  - Control signals
  - Power (sometimes)



40

40

Bus

- Connect
  - CPU and Memory
  - I/O peripherals:
    - on same bus as CPU/memory or separate bus
- Physical packaging commonly called backplane
  - Also called system bus or external bus
  - Example of broadcast bus
  - Part of printed circuit board called motherboard that holds CPU and related components

41

41

Bus Characteristics

- Protocol
  - Documented agreement for communication
  - Specification that spells out the meaning of each line and each signal on each line
- Throughput, i.e., data transfer rate in bits per second
- Data width in bits carried simultaneously

42

42

Bus types

- Data Bus
  - Carries data
  - Width is a key determinant of performance
    - 8, 16, 32, 64 bit
- Address bus
  - Identify the source or destination of data
  - Bus width determines maximum memory capacity of system
    - e.g. 8080 has 16 bit address bus giving 64k address space
- Control Bus
  - Control and timing information
    - Memory read/write; I/O read/write; Transfer acknowledge; Bus request; Bus grant; Interrupt request; Interrupt acknowledge; Clock; Reset

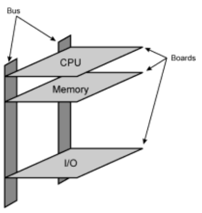
43

43

What do buses look like?

- Parallel lines on circuit boards
- Ribbon cables
- Strip connectors on mother boards
  - e.g. PCI
- Sets of wires

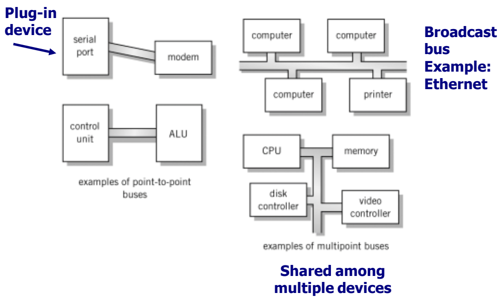
Physical Realization of Bus Architecture



44

44

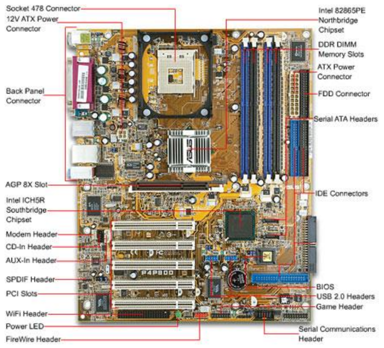
Point-to-point vs. Multipoint


Plug-in device
serial port
modem
control unit
ALU
examples of point-to-point buses
computer
computer
computer
printer
Broadcast bus Example: Ethernet
CPU
memory
disk controller
video controller
examples of multipoint buses
Shared among multiple devices

45

45

Motherboard

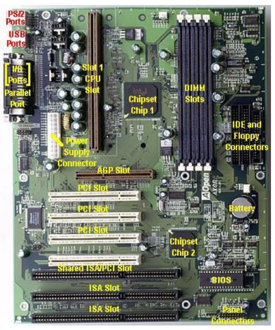


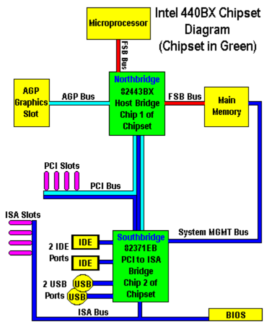
- Printed circuit board that holds CPU and related components including backplane

46

46

Motherboard



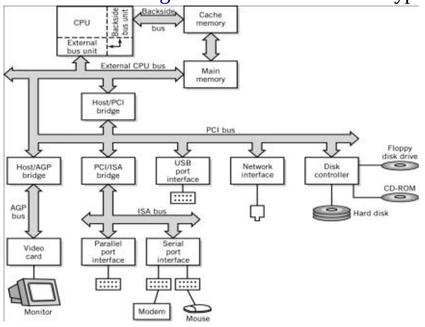

Intel 440BX Chipset Diagram (Chipset in Green)
Microprocessor
FSB Bus
AGP Graphics Slot
AGP Bus
Northbridge 82443BX Host Bridge Chip 1 of Chipset
FSB Bus
Main Memory
PCI Slots
PCI Bus
ISA Slots
2 IDE Ports
IDE
IDE
2 USB Ports
USB
USB
Southbridge 82371EB PCI to ISA Bridge Chip 2 of Chipset
System MGMT Bus
ISA Bus
BIOS

47

47

Typical PC Interconnections

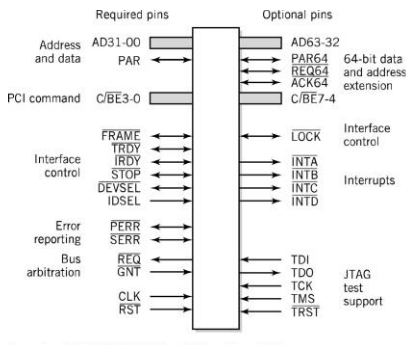
- Bus interface bridges connect different bus types



48

48

PCI Bus Connections



Source: Copyright © PCI Pin List/PCI Special Interest Group, 1999.

49

49

50

50

Instructions

- Instruction
  - Direction given to a computer
  - Causes electrical signals to be sent through specific circuits for processing
- Instruction set
  - Design defines functions performed by the processor
  - Differentiates computer architecture by the
    - Number of instructions
    - Complexity of operations performed by individual instructions
    - Data types supported
    - Format (layout, fixed vs. variable length)
    - Use of registers
    - Addressing (size, modes)

51

51

Instruction Elements

- OPCODE: task
- Source OPERAND(s) } **Addresses**
- Result OPERAND
  - Location of data (register, memory)
    - Explicit:
      - included in instruction
    - Implicit:
      - default assumed

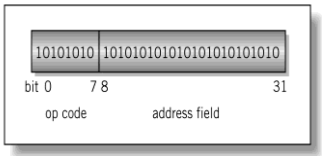
OPCODE	Source OPERAND	Result OPERAND

52

52

Instruction Format

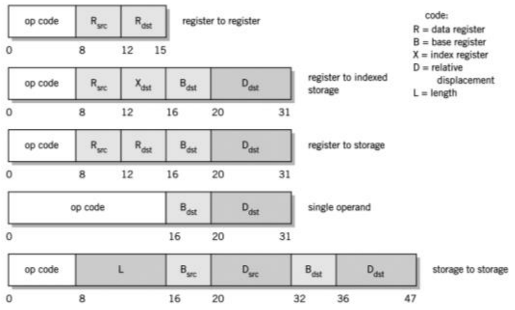
- Machine-specific template that specifies
  - Length of the op code
  - Number of operands
  - Length of operands
    - Simple 32-bit Instruction Format



53

53

Instruction Formats: CISC

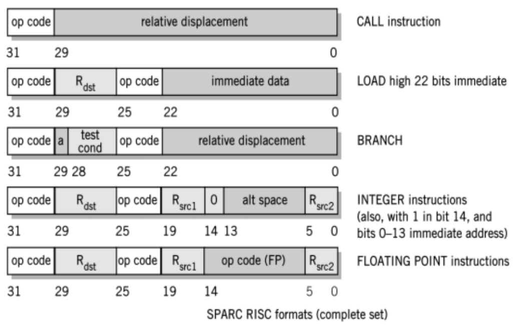


IBM mainframe formats (partial set)

54

54

Instruction Formats: RISC



SPARC RISC formats (complete set)

55

55

Instruction Types

- Data Transfer (load, store)
  - Most common, greatest flexibility
  - Involve memory and registers
  - What's a word ?
    - 16? 32? 64 bits?
- Arithmetic
  - Operators + - / * ^
  - Integers and floating point
- Logical or Boolean
  - Relational operators: > < =
  - Boolean operators **AND, OR, XOR, NOR,** and **NOT**
- Single operand manipulation instructions
  - Negating, decrementing, incrementing

56

56

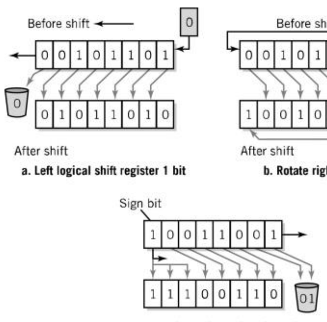
More Instruction Types

- Bit manipulation instructions
  - Flags to test for conditions
- Shift and rotate
- Program control
- Stack instructions
- Multiple data instructions
- I/O and machine control

57

57

Register Shifts and Rotates



a. Left logical shift register 1 bit

b. Rotate right 1 bit

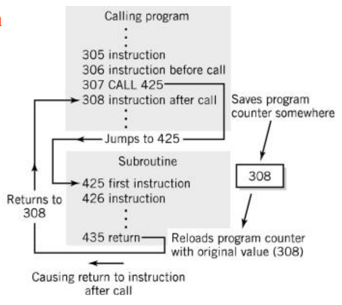
c. Right arithmetic shift 2 bits

58

58

Program Control Instructions

- Program control
  - Jump and branch
  - Subroutine call and return



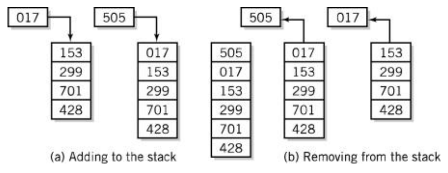
Calling program
305 instruction
306 instruction before call
307 CALL 425
308 instruction after call
Saves program counter somewhere
Jumps to 425
Subroutine
425 first instruction
426 instruction
308
Returns to 308
435 return
Reloads program counter with original value (308)
Causing return to instruction after call

59

59

Stack Instructions

- Stack instructions
  - LIFO method for organizing information
  - Items removed in the reverse order from that in which they are added



(a) Adding to the stack

Push

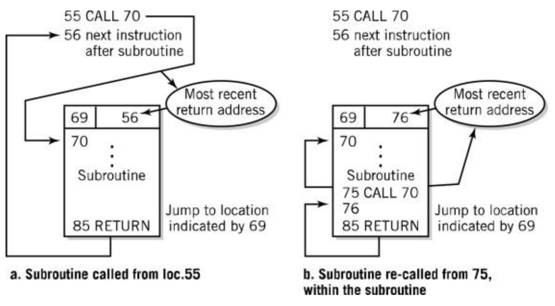
(b) Removing from the stack

Pop

60

60

Fixed Location Subroutine Return Address Storage



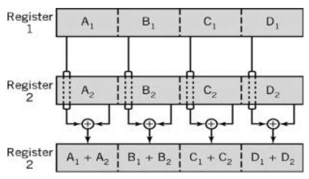
55 CALL 70
56 next instruction after subroutine
Most recent return address
69 56
70
Subroutine
85 RETURN
Jump to location indicated by 69
a. Subroutine called from loc.55
55 CALL 70
56 next instruction after subroutine
Most recent return address
69 76
70
Subroutine
75 CALL 70
76
85 RETURN
Jump to location indicated by 69
b. Subroutine re-called from 75, within the subroutine

61

61

Multiple Data Instructions

- Perform a single operation on multiple pieces of data simultaneously
  - SIMD:
    - Single Instruction, Multiple Data
  - Intel MMX™: 57 multimedia instruction



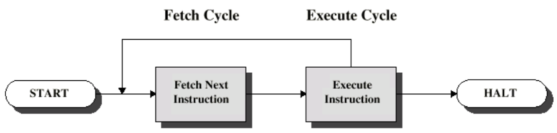
  - Commonly used in vector and array processing applications

62

62

Instruction Cycle

- Two steps:
  - Fetch
  - Execute

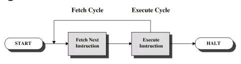


63

63

Fetch Cycle

- Program Counter (PC) holds address of next instruction to fetch



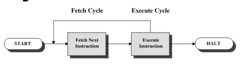
- Processor fetches instruction from memory location pointed to by PC
- Increment PC
  - Unless told otherwise
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions

64

64

Execute Cycle

- Processor-memory

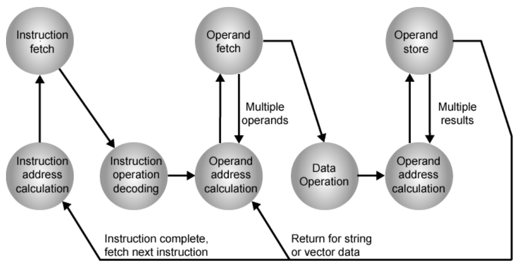


  - Data transfer between CPU and main memory
- Processor I/O
  - Data transfer between CPU and I/O module
- Data processing
  - Some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
    - e.g. jump
- Combination of above

65

65

Instruction Cycle State Diagram

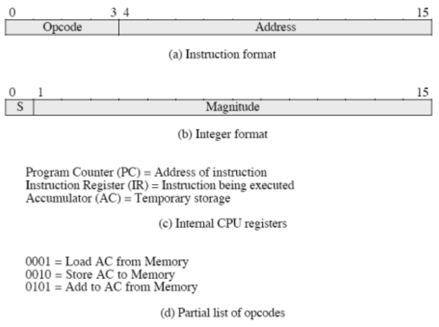


Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data

66

66

A simple example – A hypotetical machine



(a) Instruction format

(b) Integer format

Program Counter (PC) = Address of instruction
Instruction Register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from Memory
0010 = Store AC to Memory
0101 = Add to AC from Memory

(d) Partial list of opcodes

67

67

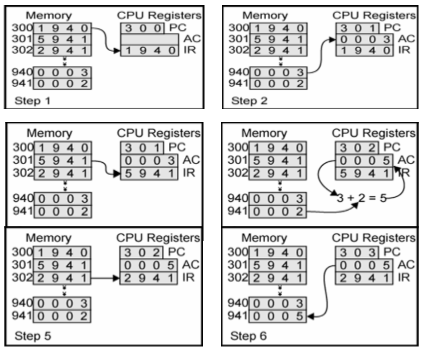
A simple example –

- Next figure illustrates a partial program execution.
- It adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the address 941.
- Here 3 instructions (3 fetch and 3 execute cycles) are required

68

68

Example of Program Execution

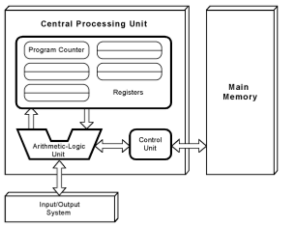


69

69

von Neumann Architecture (1945)

- Stored program concept
- Memory is addressed linearly
- Memory is addressed without regard to content



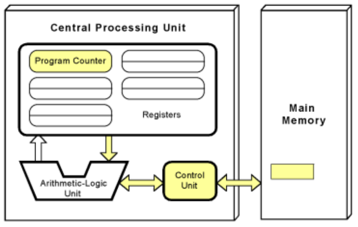
  – This is a general depiction of a von Neumann system:
  – These computers employ a fetch-decode-execute cycle to run programs as follows . . .

70

70

von Neumann Architecture

- The control unit fetches the next instruction from memory using the program counter to determine where the instruction is located.

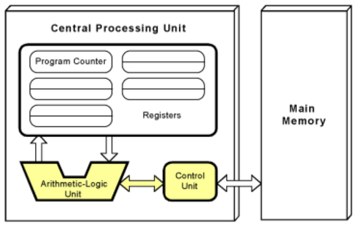


71

71

von Neumann Architecture

- The instruction is decoded into a language that the ALU can understand.

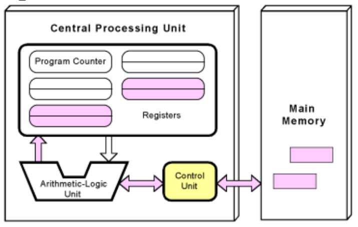


72

72

von Neumann Architecture

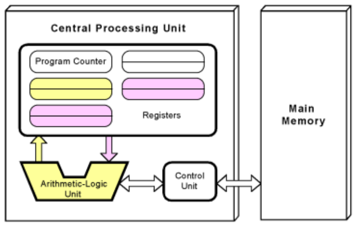
- Any data operands required to execute the instruction are fetched from memory and placed into registers within the CPU.



73

von Neumann Architecture

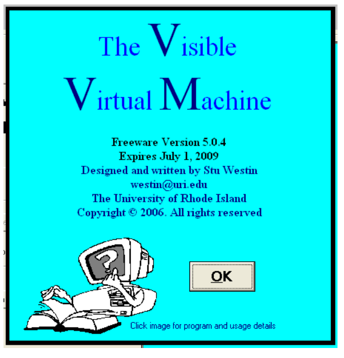
- The ALU executes the instruction and places results in registers or memory.



74

75

A virtual processor for understanding instruction cycle


The Visible Virtual Machine

Freeware Version 5.0.4
Expires July 1, 2009
Designed and written by Stu Westin
westin@uri.edu
The University of Rhode Island
Copyright © 2006. All rights reserved

76

The VVM Machine

- The Visible Virtual Machine (VVM) is based on a model of a simple computer device called the Little Man Computer which was originally developed by Stuart Madnick in 1965, and revised in 1979.
- The VVM is a virtual machine because it only appears to be a functioning hardware device.
- In reality, the VVM "hardware" is created through a software simulation.
- One important simplifying feature of this machine is that it works in decimal rather than in the traditional binary number system.
- Also, the VVM works with only one form of data –
  - decimal integers.

77

Hardware Components of VVM

- I/O Log
  - This represents the system console which shows the details of relevant events in the execution of the program.
    - Examples of events are the program begins, the program aborts, or input or output is generated.
- Accumulator Register
  - This register holds the values used in arithmetic and logical computations.
  - It also serves as a buffer between input/output and memory.
    - Legitimate values are any integer between -999 and +999.
    - Values outside of this range will cause a fatal VVM Machine error.
    - Non integer values are converted to integers before being loaded into the register.
- Instruction Cycle Display
  - This shows the number of instructions that have been executed since the current program execution began.

78

Hardware Components of VVM

- Instruction Register (Instr. Reg.).
  - This register holds the next instruction to be executed.
  - divided into two parts:
    - a one-digit operation code, and a two digit operand.
  - The Assembly Language mnemonic code for the operation code is displayed below the register.
- Program Counter Register (Prog. Ctr.).
  - The two-digit integer value in this register "points" to the next instruction to be fetched from RAM.
  - Most instructions increment this register during the *execute* phase of the instruction cycle.
  - Legitimate values range from 00 to 99.
    - A value beyond this range causes a fatal VVM Machine error.
- RAM.
  - The 100 *data-word* Random Access Storage is shown as a matrix of ten rows and ten columns.
  - The two-digit memory addresses increase sequentially across the rows and run from 00 to 99.
    - Each storage location can hold a three-digit integer value between -999 and +999.

79

79

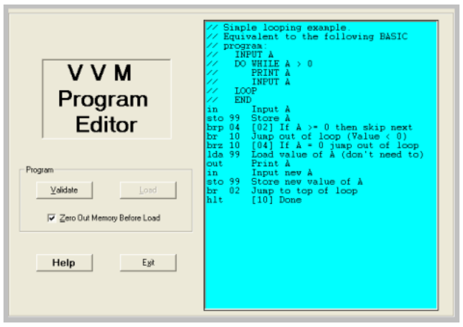
Data and Addresses

- All data and address values are maintained as decimal integers.
- The 100 data-word memory is addresses with two-digit addressed in the range 00-99.
- Each memory location holds one data-word which is a decimal integer in the range -999 - +999.
- Data values beyond this range cause a data overflow condition and trigger a VVM system error.

80

80

VVM

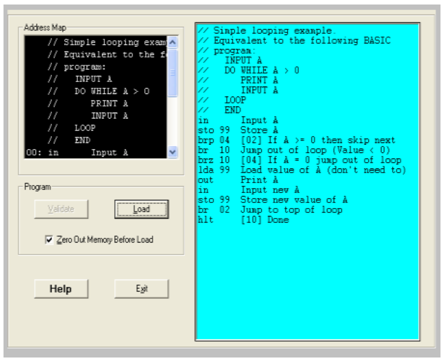


```
// Simple looping example.
// Equivalent to the following BASIC
// program:
//   INPUT A
//   DO WHILE A > 0
//      PRINT A
//      INPUT A
//   LOOP
//   END
in      Input A
sto 99  Store A
brp 04  [02] If A >= 0 then skip next
br  10  Jump out of loop (Value < 0)
brz 10  [04] If A = 0 jump out of loop
lda 99  Load value of A (don't need to)
out     Print A
in      Input new A
sto 99  Store new value of A
br  02  Jump to top of loop
hlt     [10] Done
```

81

81

VVM



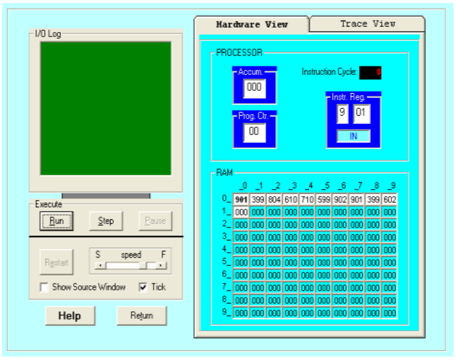
```
    // Simple looping exam
    // Equivalent to the f
    // program:
    //   INPUT A
    //   DO WHILE A > 0
    //      PRINT A
    //      INPUT A
    //   LOOP
    //   END
00: in      Input A
```

```
// Simple looping example.
// Equivalent to the following BASIC
// program:
//   INPUT A
//   DO WHILE A > 0
//      PRINT A
//      INPUT A
//   LOOP
//   END
in      Input A
sto 99  Store A
brp 04  [02] If A >= 0 then skip next
br  10  Jump out of loop (Value < 0)
brz 10  [04] If A = 0 jump out of loop
lda 99  Load value of A (don't need to)
out     Print A
in      Input new A
sto 99  Store new value of A
br  02  Jump to top of loop
hlt     [10] Done
```

82

82

VVM



	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9
0_	901	399	804	610	710	599	902	901	399	602
1_	000	000	000	000	000	000	000	000	000	000
2_	000	000	000	000	000	000	000	000	000	000
3_	000	000	000	000	000	000	000	000	000	000
4_	000	000	000	000	000	000	000	000	000	000
5_	000	000	000	000	000	000	000	000	000	000
6_	000	000	000	000	000	000	000	000	000	000
7_	000	000	000	000	000	000	000	000	000	000
8_	000	000	000	000	000	000	000	000	000	000
9_	000	000	000	000	000	000	000	000	000	000

83

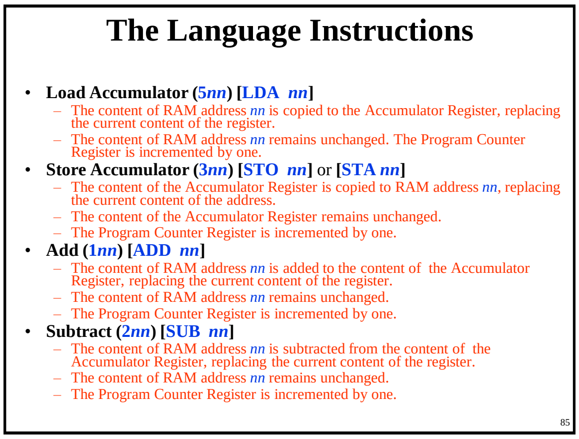
83

VVM System Errors

- Data value out of range.
  - This condition occurs when a data value exceeds the legitimate range -999 - +999.
  - The condition will be detected while the data resides in the *Accumulator Register*.
    - Probable causes are an improper addition or subtraction operation, or invalid user input.
- Undefined instruction.
  - This occurs when the machine attempts to execute a three-digit value in the *Instruction Register* which can not be interpreted as a valid instruction code.
    - Probable causes of this error are attempting to use a data value as an instruction, an improper *Branch* instruction, or failure to provide a *Halt* instruction in your program.
- Program counter out of range.
  - This occurs when the Program Counter Register is incremented beyond the limit of 99.
    - The likely cause is failure to include a *Halt* instruction in your program, or a branch to a high memory address.
- User cancel.
  - The user pressed the "Cancel" button during an *Input* or *Output* operation.

84

84

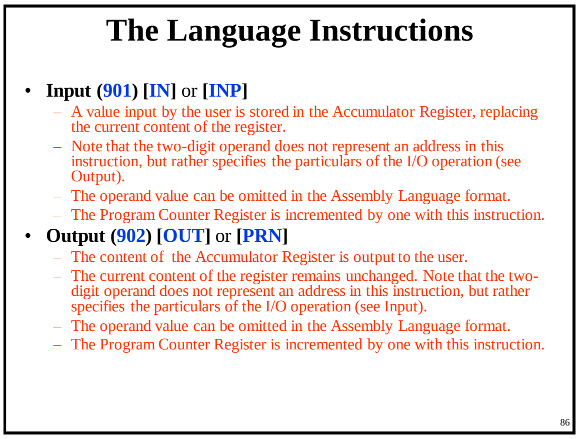


The Language Instructions

- **Load Accumulator (5*nn*) [LDA *nn*]**
  - The content of RAM address *nn* is copied to the Accumulator Register, replacing the current content of the register.
  - The content of RAM address *nn* remains unchanged. The Program Counter Register is incremented by one.
- **Store Accumulator (3*nn*) [STO *nn*] or [STA *nn*]**
  - The content of the Accumulator Register is copied to RAM address *nn*, replacing the current content of the address.
  - The content of the Accumulator Register remains unchanged.
  - The Program Counter Register is incremented by one.
- **Add (1*nn*) [ADD *nn*]**
  - The content of RAM address *nn* is added to the content of the Accumulator Register, replacing the current content of the register.
  - The content of RAM address *nn* remains unchanged.
  - The Program Counter Register is incremented by one.
- **Subtract (2*nn*) [SUB *nn*]**
  - The content of RAM address *nn* is subtracted from the content of the Accumulator Register, replacing the current content of the register.
  - The content of RAM address *nn* remains unchanged.
  - The Program Counter Register is incremented by one.

85

85

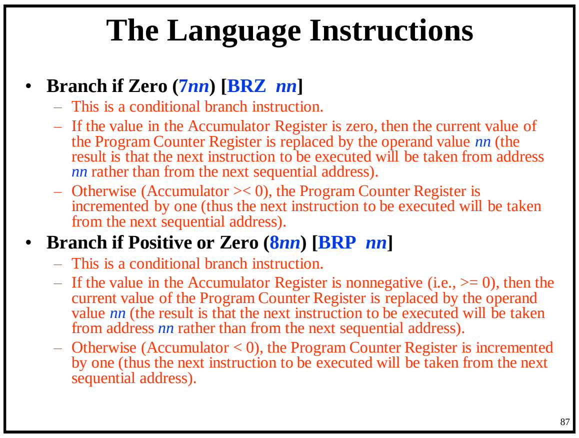


The Language Instructions

- **Input (901) [IN] or [INP]**
  - A value input by the user is stored in the Accumulator Register, replacing the current content of the register.
  - Note that the two-digit operand does not represent an address in this instruction, but rather specifies the particulars of the I/O operation (see Output).
  - The operand value can be omitted in the Assembly Language format.
  - The Program Counter Register is incremented by one with this instruction.
- **Output (902) [OUT] or [PRN]**
  - The content of the Accumulator Register is output to the user.
  - The current content of the register remains unchanged. Note that the two-digit operand does not represent an address in this instruction, but rather specifies the particulars of the I/O operation (see Input).
  - The operand value can be omitted in the Assembly Language format.
  - The Program Counter Register is incremented by one with this instruction.

86

86

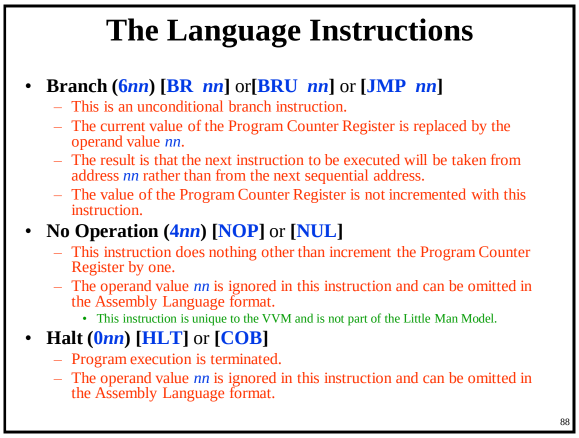


The Language Instructions

- **Branch if Zero (7*nn*) [BRZ *nn*]**
  - This is a conditional branch instruction.
  - If the value in the Accumulator Register is zero, then the current value of the Program Counter Register is replaced by the operand value *nn* (the result is that the next instruction to be executed will be taken from address *nn* rather than from the next sequential address).
  - Otherwise (Accumulator >< 0), the Program Counter Register is incremented by one (thus the next instruction to be executed will be taken from the next sequential address).
- **Branch if Positive or Zero (8*nn*) [BRP *nn*]**
  - This is a conditional branch instruction.
  - If the value in the Accumulator Register is nonnegative (i.e., >= 0), then the current value of the Program Counter Register is replaced by the operand value *nn* (the result is that the next instruction to be executed will be taken from address *nn* rather than from the next sequential address).
  - Otherwise (Accumulator < 0), the Program Counter Register is incremented by one (thus the next instruction to be executed will be taken from the next sequential address).

87

87

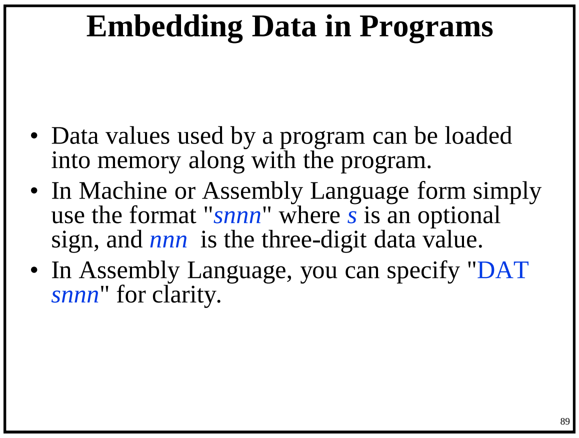


The Language Instructions

- **Branch (6*nn*) [BR *nn*] or[BRU *nn*] or [JMP *nn*]**
  - This is an unconditional branch instruction.
  - The current value of the Program Counter Register is replaced by the operand value *nn*.
  - The result is that the next instruction to be executed will be taken from address *nn* rather than from the next sequential address.
  - The value of the Program Counter Register is not incremented with this instruction.
- **No Operation (4*nn*) [NOP] or [NUL]**
  - This instruction does nothing other than increment the Program Counter Register by one.
  - The operand value *nn* is ignored in this instruction and can be omitted in the Assembly Language format.
    - This instruction is unique to the VVM and is not part of the Little Man Model.
- **Halt (0*nn*) [HLT] or [COB]**
  - Program execution is terminated.
  - The operand value *nn* is ignored in this instruction and can be omitted in the Assembly Language format.

88

88

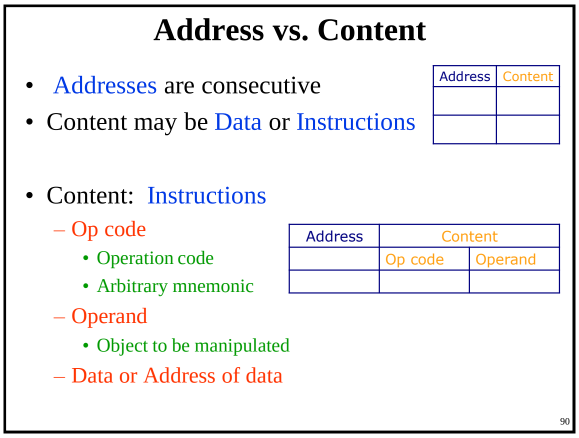


Embedding Data in Programs

- Data values used by a program can be loaded into memory along with the program.
- In Machine or Assembly Language form simply use the format "*snnn*" where *s* is an optional sign, and *nnn* is the three-digit data value.
- In Assembly Language, you can specify "DAT *snnn*" for clarity.

89

89



Address vs. Content

- Addresses are consecutive
- Content may be Data or Instructions

Address	Content

- Content: Instructions
  - Op code
    - Operation code
    - Arbitrary mnemonic
  - Operand
    - Object to be manipulated
  - Data or Address of data

Address	Content	
	Op code	Operand

90

90

Assembly Language

- Specific to a CPU
- 1 to 1 correspondence between assembly language instruction and binary (machine) language instruction
- Mnemonics (short character sequence) represent instructions
- Used when programmer needs precise control over hardware,
  - e.g., device drivers

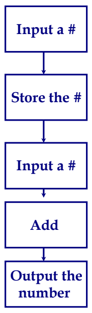
91

91

Simple Program: Add 2 Numbers

- Assume data is stored in mailboxes with addresses > 80
- Write instructions

Mailbox	Mnemonic	Code	Instruction Description
00	IN	901	;input 1st Number
01	STO 85	399	;store data
02	IN	901	;input 2nd Number
03	ADD 85	199	;add 1st # to 2nd #
04	OUT	902	;output result
05	COB	000	;stop
85	DAT 00	000	;data



92

92

Find Positive Difference of 2 Numbers

00	IN	901	
01	STO 10	310	
02	IN	901	
03	STO 11	311	
04	SUB 10	210	
05	BRP 08	808	;test
06	LDA 10	510	;if negative, reverse order
07	SUB 11	211	
08	OUT	902	;print result and
09	COB	000	;stop
10	DAT 00	000	;used for data
11	DAT 00	000	;used for data

93

93

94

94