

We understand that a processor is made up of digital logic circuits (hardware).

— The micro-architecture of a processor describes how these

circuits are organized to create a processor.

Processor = Controller + Data Paths

— Internally the processor performs simple tasks by

executing microinstructions on a data path.

- Microinstructions are grouped to create micro-routines

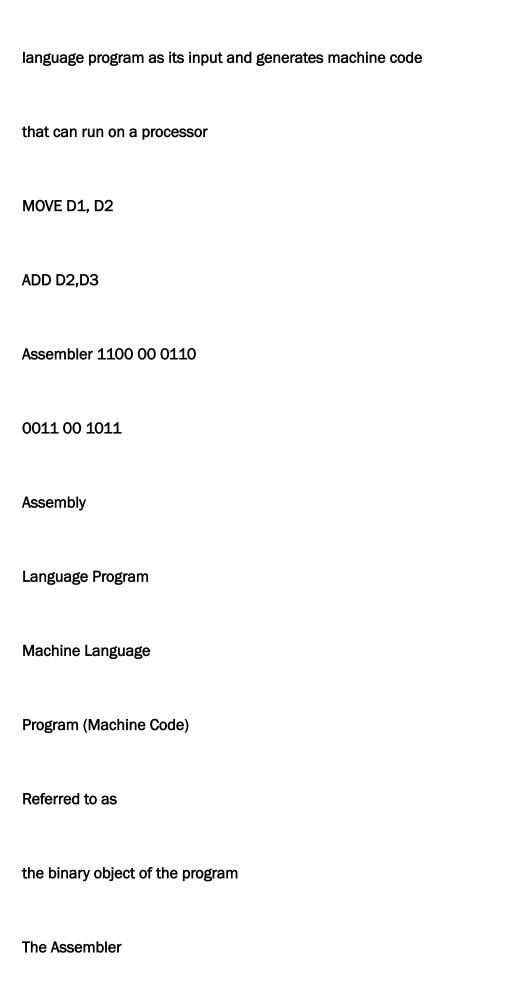
that perform useful operations.

So far
+
In a typical processor these micro-routines are organized
to:
1. Fetch an instruction from RAM and place the instruction into
the processor's Instruction Register.
2. Decode the instruction (i.e. determine what is to be done)
3. Execute the task that the instruction requires.
This simple cycle is repeated for each instruction that is
given to the processor by a program.
So far
+
We discuss:

1. Why Assembly Languages are useful.
2. Concepts to be considered when implementing an Assembly
Language:
➤Instruction Formats
➤ Register Access
➤ Addressing Modes
➤ Branching
In this Lecture
+
■ The machine level of a computer
system is positioned above the
micro-architecture level.
■ The machine level is the interface

between hardware and software.
■ A programmer can write instructions
to the processor at the machine level
using what is called machine code.
■ The Machine Level defines the
Instruction Set Architecture (ISA) of
a processor.
■ Describes how a processor can be
used.
The Machine Level
+
— A programmer can use machine code to write programs.
Enter strings of bits to tell the processor what to do.

Tedious, error prone, unproductive.
— Few programmers write programs at the machine code level
today.
— Those who do use a language that is called assembly language to
write instructions.
— The assembly language for a processor defines mnemonics for
machine code.
For example the mnemonic ADD might stand for the code 0011.
Instead of writing the instruction 0011 programmer writes the
assembly language instruction ADD
Assembly Language
+
■ An Assembler is a program that accepts an assembly



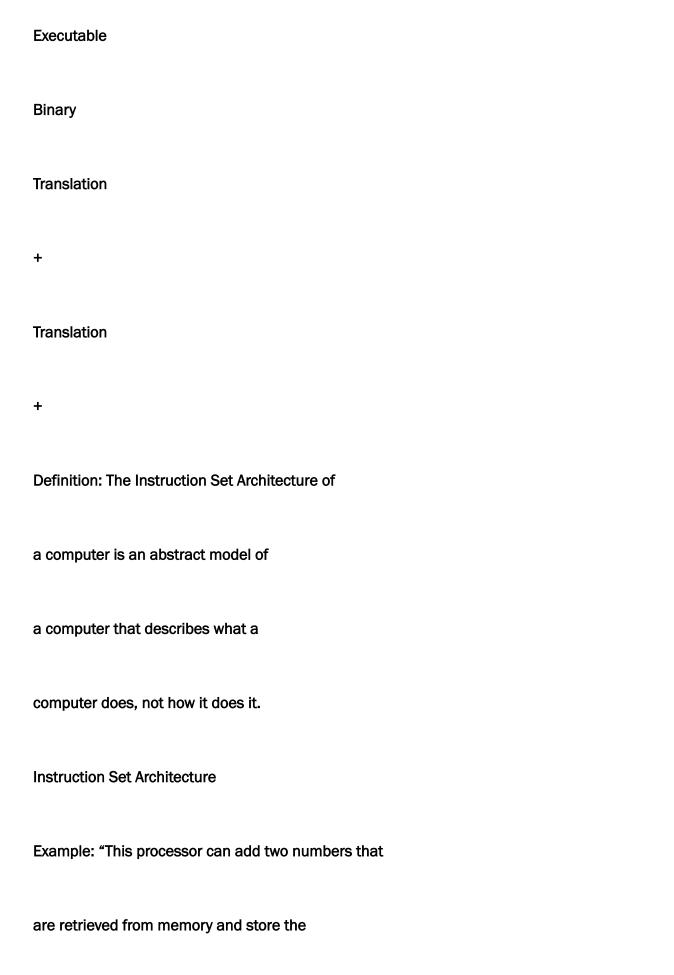
— Most programmers write programs using High Level
Languages (e.g. C, C++, Java, etc).
— High Level Programming Languages are for human beings.
So that humans can see what a program is supposed to do.
— In order to be run on a computer system a HLL program must
be translated to a form that the processor can understand.
— This is done through a process called compiling.
A Compiler is a program that takes a HLL program as its input
and produces machine code as its output.
A C compiler translates a C program to machine code.
Translation

1. Compiler translates C code (for example) to assembly language
program
2. Assembler produces machine code from assembly language
In most cases, a program will have several modules that must all be
combined to make one executable program. Very often these other
programs are pre-defined programs that are supplied as object
libraries.
3. All object code that is to make up a program are brought together to
produce an executable binary program (machine code) by a program
that is called a linker.
(Note a compiler will often carry out these stages without stopping,
unless otherwise instructed)

Translation is done in several stages:

Binary Object	
•	
p = q + r	
Compiler	
Assembly Language	
version of program	
Linker	
Object	
Libraries	
Assembler	

Translation



result to a location in memory".
Details of implementation are in the organization of the
units that make up the computer (micro-architecture).
+
■ An ISA can then be seen as an abstract specification of a
processor which is written for programmers who write
compilers.
■ The compiler must be written to produce instructions that the
processor can "understand".
■ Programmers who write operating systems also need to be
familiar with the ISA of the computer system.
■ The operating system (e.g. Windows, or Linux).
Other Definitions

▼ An ISA defines
▼ An Instruction Set
▼ Set of instructions that the processor can execute (the language that
processor understands).
▼ Format in which instruction must be written
▼ How many bits are in an instruction
▼ How to specify what operation is to be done
▼ How to specify the data on which the operation is to be carried out
▼ A set of registers that can be used by instructions
▼ Referred to as the register set
▼ Memory Model
▼ How the processor views memory. Typically, how an address that the

processor issues relates to a location in memory.
ISA Properties
+
■ When encoded, an instruction is a
set of bits that can be portioned
into bits that contain:
■ An operation code (called opcode
for short)
■ Specifies an operation that is to
be performed (eg. Add two
numbers)
■ A number of operands

■ Values on which the operation is

to be performed
■ For ease of understanding we will
discuss instructions using
mnemonics for operation codes
and symbols to represent
operands.
OPCODE Operand 1 Operand n
Instructions
+
We will introduce a small set of operations for illustration
purposes only:
LOAD means retrieve a value from memory and store it in a register
STORE means copy a value from a register to a memory location

ADD means add two numbers to give their sum
We will also specify operands as follows:
• R1, R2, and R3 refer to registers that contain values which are to be
used in an operation
X,Y, and Z refer to memory locations where values are stored
Note: The use of mnemonics here is only for the convenience of explaining the
concept of an instruction and as such our notation will not be strict to any
specific notation such as assembly language.
Instructions
+
An ISA will describe the format of an instruction that can be
executed by the processor
Every instruction will have an operation code

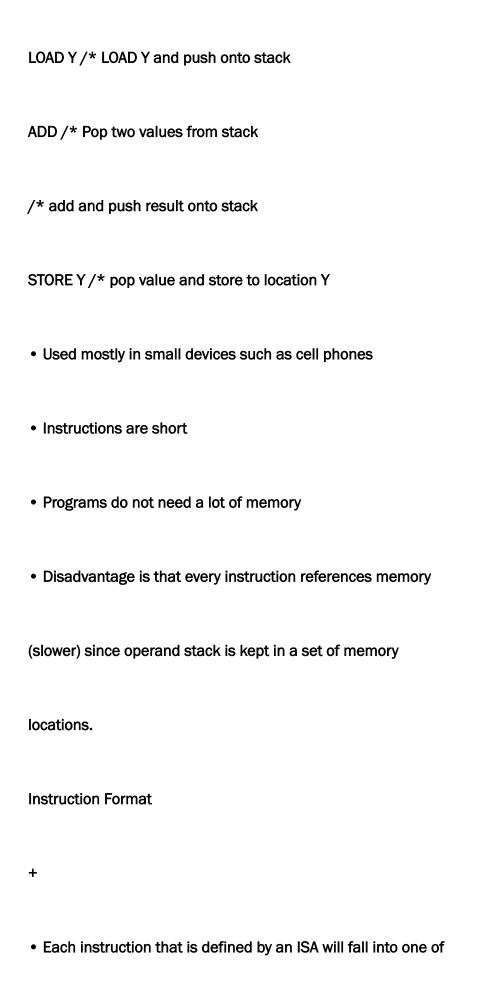
The number of operands may defer depending on the ISA
Common instruction formats include:
Three Operand Instructions
Two Operand Instructions
One Operand Instructions
Zero Operand Instructions
Instruction Format
+
■ Consider the C program statement Z = X + Y;
■ This might be translated into instructions which carry out the
following tasks
■ Retrieve a value, X from a location in memory
■ Retrieve a second value, Y, from a location in memory

■ Add the values of X and Y
■ Store the result of the addition to a memory location Z.
■ The following examples will demonstrate how this might be
done using different instruction formats
Instruction Format
+
Three operand format uses three operands (usually registers) for all
instructions except LOAD and STORE.
• Example:
LOAD X, R1 /* Retrieve X, store in register R1
LOAD Y, R2 /* Retrieve Y, store in register R2
ADD R1, R2, R3 /* Add values, store result in R3
STORE R3, Z/* Store result to memory location Z

Three operand formats are typical of RISC (Reduced Instruction Set
Computer) architectures
The order of the operands will differ among different ISA's
Destination is sometimes the last operand.
Instruction Format
+
Two operand format:
• Example:
LOAD X ,R1, /* Retrieve X, store in register R1
LOAD Y, R2, /* Retrieve Y, store in register R2
ADD R1, R2 /* Add R1 to R2 (literally speaking)
STORE R2, Z/* Store result to memory location Z
Second operand is used to store result

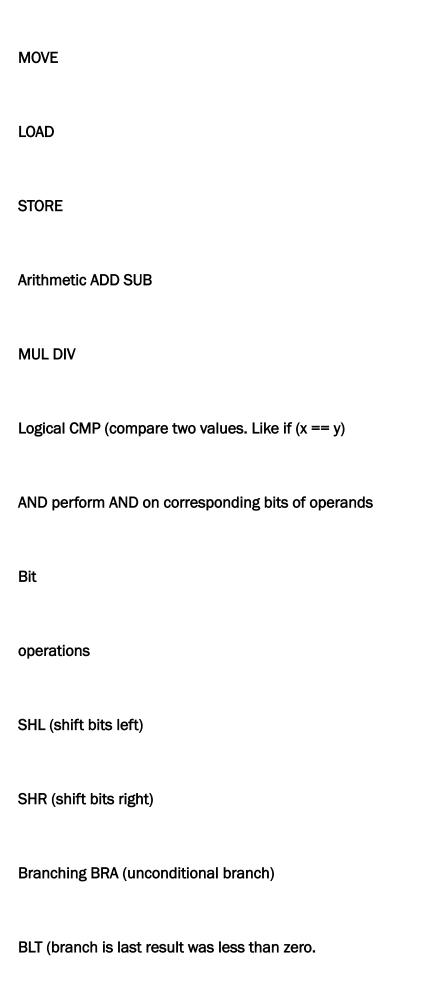
Most commonly used format for personal computer
processors
Instruction shorter than 3 operand instruction (less memory)
Processor can have a moderate number of registers
Instruction Format
+
One operand format:
Accumulator Architecture
Implicit operand is a special register which is called the
accumulator
Example:
LOAD X /* Retrieve X, store in accumulator
MOVE R2 /* Copy value from accumulator to register R2

LOAD Y /* Retrieve Y, store in accumulator
ADD R2 /* Add R2 to accumulator
STORE Z /* Store accumulator value to location Z
Popular in older computers
Small number of registers needed
Mostly used in calculators
Instruction Format
+
Zero operand format:
Stack Architecture
Implicit use of an operand stack
• Example:
LOAD X /* Load X and push onto stack



Data Movement (copy data from one location to another)
Arithmetic (perform common arithmetic operations)
Logical Operations (perform logical and Boolean operations)
Operations on Bits (e.g., complement bits, shift bits, etc,)
Branching operations that change the flow of execution (e.g.,
looping)
These are general classes
Instruction Types
Instruction Types
Some Examples (again, we use mnemonics)
Data
Movement

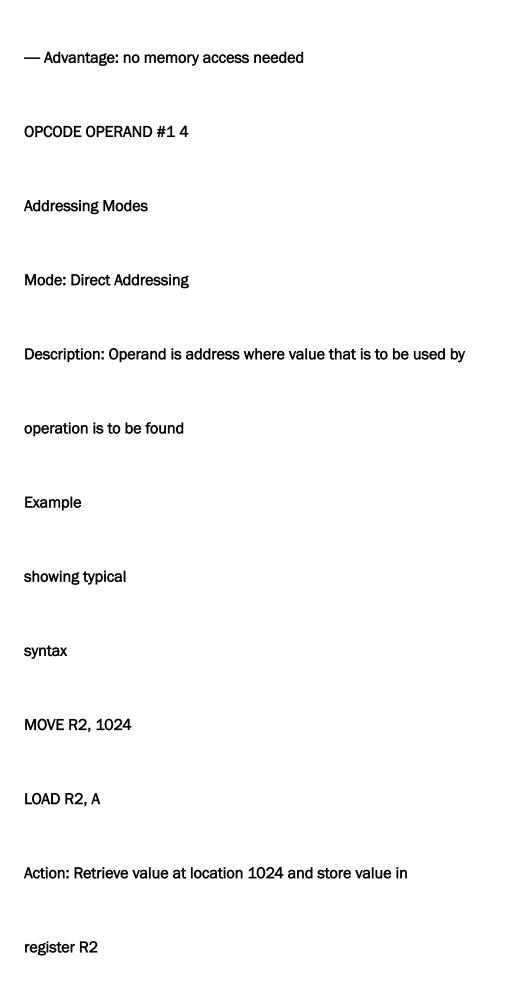
several categories:



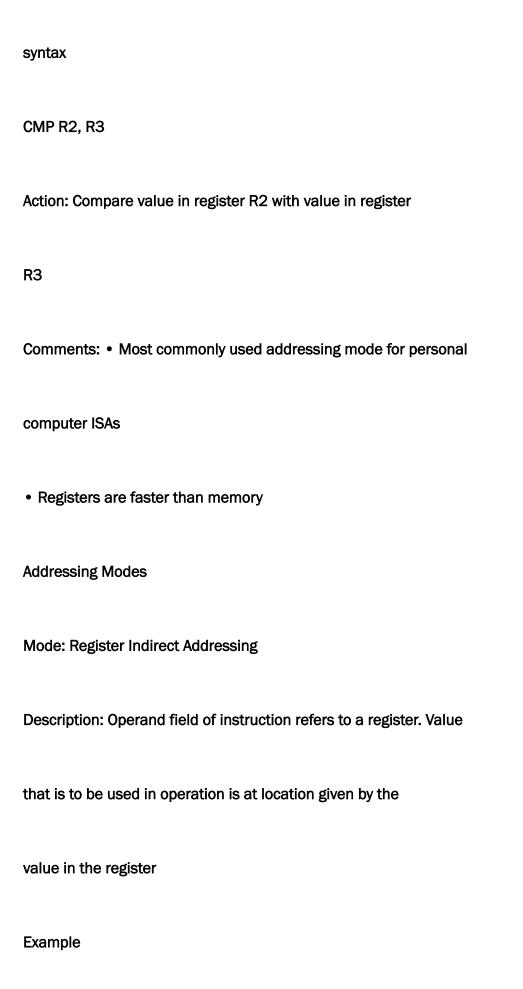
Mnemonics will vary with ISA but most use this type of syntax
+
— The ISA of a processor will define a set registers that can be
referred to in an instruction at the machine level
— Some registers that are visible at the micro-architecture level
cannot be referred to at the ISA level (they are hidden from a
programmer)
Programmer does not need to be concerned with the MAR, MDR, IR, for
example.
These are used internally (at the microarchitecture level)
— Program Counter (PC) is usually visible
— Also, general purpose registers
For example, the Motorola 68000 (68K) ISA defines:

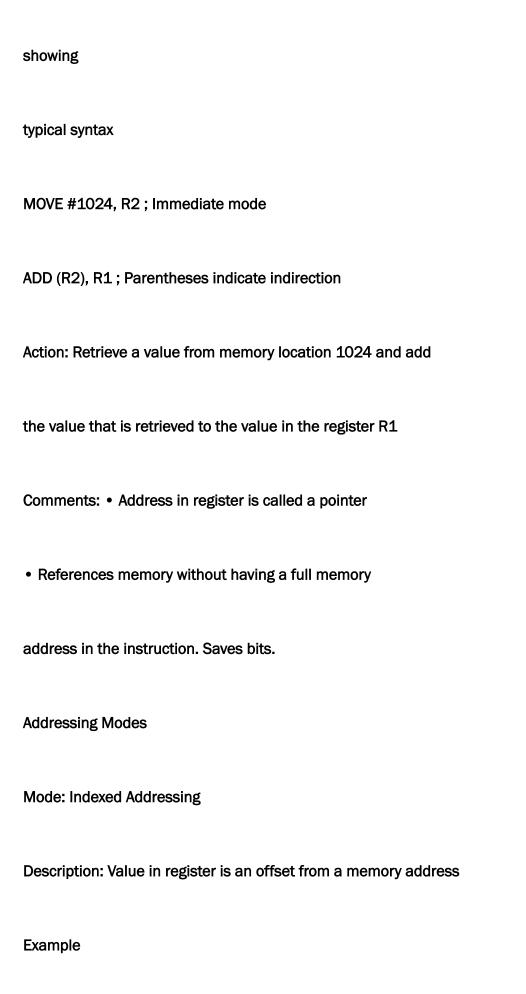
÷ 16 General purpose registers (D0 to D15)
÷ 16 Address Registers (A0-A15) that are used as pointers
Register Set
+
An ISA will define several different modes in which an
operand can be specified in an instruction.
These include:
Immediate (literal) Addressing
Direct (absolute) Addressing
Register Addressing
Register Indirect Addressing
Indexed Addressing
Based Indexed Addressing

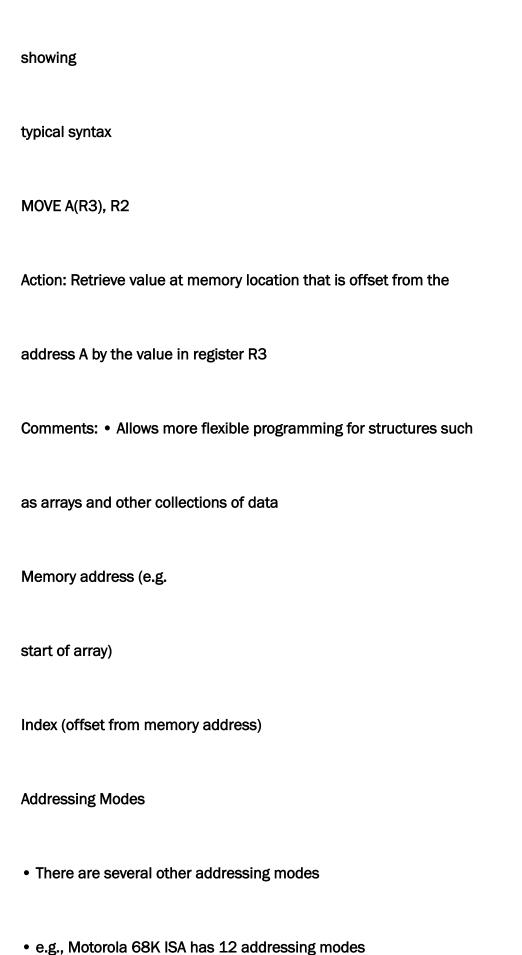
Stack Addressing (to be covered when discussing the stack)	
We will illustrate these using two operand format	
instructions.	
Addressing Modes	
Mode: Immediate (also called Literal)	
Description: Operand is the value that is included in the instruction  Example	
showing typical	
syntax	
MOVE R2, #4	
Action: Move the value 4 into the register R2	
Comments: — Largest value is limited to number of bits in operand field	
— Value is constant, change requires changing program	

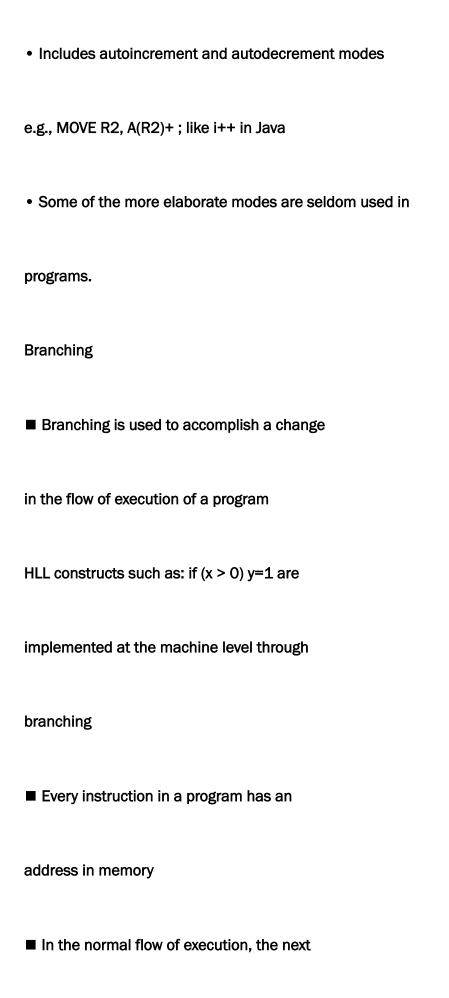


Comments: — Address is determined at compile time and cannot be
changed thereafter
— Useful for global variables if program is always loaded
in same locations
OPCODE OPERAND #1 1024
Addressing Modes
Addressing Modes
Mode: Register Addressing
Description: Operand is a register. Value in register is to be used
in operation.
Example
showing
typical

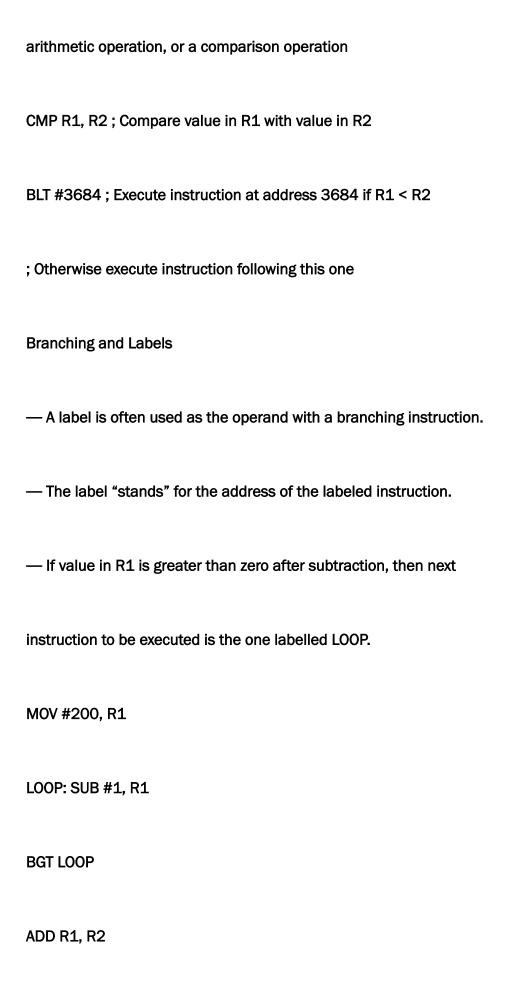


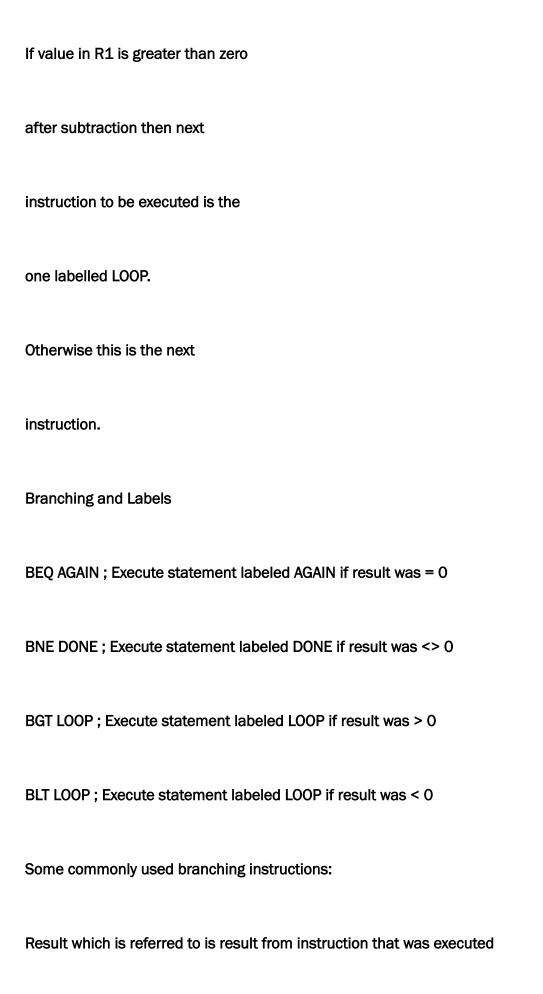






instruction to be executed is the one
following the one that is being executed.
■ A branching instruction can result in
another instruction being executed
instead.
loop
Skip
over
Branching and Labels
Example:
BRA #3684; Execute instruction at address 3684 after this one
; This is an unconditional branch
A conditional branch is used with another instruction that executes an



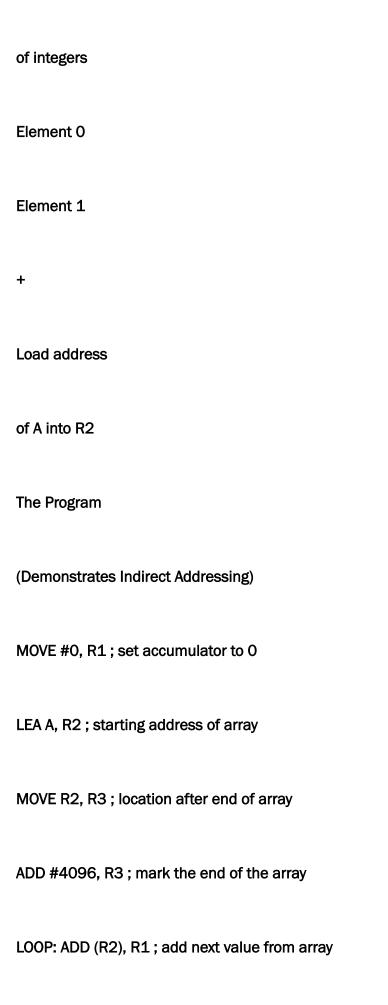


immediately before the branching instruction
How it Is Done
■ A note on how branching is accomplished:
At the microarchitecture level there is a special register which called the
Program Status Word (PSW)
This register is continuously updated to store information on the status
of the processor
Contains bits (Condition Code Bits) that are updated after each ALU
operation:
• Z is set to 1 if last ALU result was 0
N is set to 1 if last ALU result was negative
C is set to 1 if there was a carry out on the last ALU operation
V is set to 1 if there was an overflow on the last operation

The PSW has many other bits that store information such as the
processor mode (e.g. privileged vs. unprivileged)-
How it Is Done
Other status data
ALU
ZNCV
Out
PROGRAM
STATUS WORD
(usually made
from latches)
These wires go to the
Control Unit

The microinstructions that implement a branching instruction will
test the condition code bits of the PSW and set the Program
Counter to effect the branch.
For example, given the instructions:
AGAIN:CMP R1, R2; Subtraction occurs
BEQ AGAIN
To execute the BEQ instruction, the microinstructions in the
micro-routine that implement the BEQ will:
check the Z bit of the PSW
if (Z ==1) move address of statement that is labelled AGAIN to PC
Illustration of Indirect Addressing
▼ Problem:

▼ An array A (in memory) stores 1024
integers. Compute the sum of the
integers and store the result in the
register R1.
Notes:
▼ Array elements are stored in
consecutive memory locations
▼ Name of array is address of first
element (element 0)
▼ Each element is 4 bytes (integer)
▼ Element n is at starting address + n
* 4 (offset)
Depiction of an array



ADD #4, R2; increment index
CMP R2, R3; check for end of array
BLT LOOP; keep going. Not yet at end
•
•
+
A processor is described by its Instruction Set Architecture
The ISA defines the instructions that a processor can understand
The micro-architecture of a processor interprets each instruction in a program
An instruction has an operation code and zero, or more operands
The operation (code) is a unique code for each instruction that the processor can
interpret

The operand field(s) of an instruction can be interpreted in several ways which
are referred to as addressing modes
An operand can be a register, or a memory address
Literal addressing, direct addressing, and indirect addressing are the most
common modes
Branching causes a change in the flow of instructions that are executed
Branching is accomplished using the bits of the Program Status Word that are
updated from the ALU status bits.