

### 3-ADDRESS CODE SIMULATOR DOCUMENTATION

#### A) SIMULATOR OVERVIEW

- 1) The 3-address code simulator is a two-pass simulator. The first scans and interprets the instruction and the arguments. The second pass executes the instructions until the end is reached or 1,000,000 instructions have been executed.
- 2) Errors found in the first pass will cause an error message and the scanning of the remaining instruction, but NO execution of the instructions.
- 3) Errors found in the second pass will cause an error message and an abort of the execution pass.
- 4) There will be two input files, the object that is to be executed will be read through standard input (STDIN), the file "datain" will be read if the executing code requests a read.
- 5) The output will go to Standard output (STOUT). The output includes the instructions scanned, accompanying messages, error info and output generated by executing the instructions.
- 6) The error info generated by either a scanning error or a execution error will include the current PC, the last five instructions executed and the contents of the symbol table.

#### B) INPUT FORMAT OVERVIEW

- 1) There are four argument formats:
  - a) Constants - can be either real, integer, or char type. Constant values are RIGHT JUSTIFIED within the 10 position field. A real number must include a digit before the decimal point. E notation is not allowed for either type.
  - b) Variables - are alpha-numeric values of up to 10 characters which begin with a alphabetic character and are LEFT JUSTIFIED. A variable can be either integer, real or char type. A variable's type is fixed for the entire program once it is set.
  - c) Temporaries - are of the form %t# where # is an integer value up to 8 digits long and are LEFT JUSTIFIED. A temporary can be a real, char integer or boolean type. A temporary's type can be changed by using it as result of an instruction.
  - d) Strings - are of the form %s# where # is an integer value up to 8 digits long and are LEFT JUSTIFIED. A string can only represent a string value.
- 2) This simulator is CASE SENSITIVE therefore follow formats exactly. When lower case is used, use lower case characters and when upper case characters are used, use upper case characters. Arguments must stay in the same case (x and X are two different variables). This is also true for temporary variables and strings.
- 3) There is a fixed sized symbol table in the simulator. If you receive an error of the form "Symbol Table Overflow - (area) - Inform JDA", copy down the whole error message or get a printout. Bring it to me and I will address the problem.

C) INSTRUCTION FORMATS AND FUNCTION

(Note: "--" signifies an empty argument field)

1) Addition

Col. 1	10	25	40
+	Arg1	Arg2	Result

- a) Function - The action of this instruction is  $\text{Result} := \text{Arg1} + \text{Arg2}$
- b) Arg1 - can be Constant, Variable or Temporary Format
  - its type must match Arg2, can be either integer or real
- c) Arg2 - can be Constant, Variable or Temporary Format
  - its type must match Arg1, can be either integer or real
- d) Result - Must be Temporary Format
  - its type is assigned that of Arg1 and Arg2 (both of which are either integer or real)

2) Subtraction

Col. 1	10	25	40
-	Arg1	Arg2	Result

- a) Function - The action of this instruction is  $\text{Result} := \text{Arg1} - \text{Arg2}$
- b) Arg1 - can be Constant, Variable or Temporary Format
  - its type must match Arg2, can be either integer or real
- c) Arg2 - can be Constant, Variable or Temporary Format
  - its type must match Arg1, can be either integer or real
- d) Result - Must be Temporary Format
  - its type is assigned that of Arg1 and Arg2 (both of which are either integer or real)

3) Multiplication

Col. 1	10	25	40
*	Arg1	Arg2	Result

- a) Function - The action of this instruction is  $\text{Result} := \text{Arg1} * \text{Arg2}$
- b) Arg1 - can be Constant, Variable or Temporary Format
  - its type must match Arg2, can be either integer or real
- c) Arg2 - can be Constant, Variable or Temporary Format
  - its type must match Arg1, can be either integer or real
- d) Result - Must be Temporary Format
  - its type is assigned that of Arg1 and Arg2 (both of which are either integer or real)

4) Division

Col. 1	10	25	40
/	Arg1	Arg2	Result

- a) Function - The action of this instruction is `Result := Arg1 / Arg2`.  
The division of two Integer types provides a Result with the portion after the decimal truncated.
- b) Arg1 - can be Constant, Variable or Temporary Format  
- its type must match Arg2, can be either integer or real
- c) Arg2 - can be Constant, Variable or Temporary Format  
- its type must match Arg1, can be either integer or real
- d) Result - Must be Temporary Format  
- its type is assigned that of Arg1 and Arg2 (both of which are either integer or real)

5) Negation

Col. 1	10	25	40
neg	Arg1	--	Result

- a) Function - The action of this instruction is `Result := - (Arg1)`
- b) Arg1 - can be Constant, Variable or Temporary Format  
- its type must be either integer or real
- c) Result - Must be Temporary Format  
- its type is assigned that of Arg1, which is either integer or real

6) Less Than

Col. 1	10	25	40
<	Arg1	Arg2	Result

- a) Function - The action of this instruction is `Result := Arg1 < Arg2`
- b) Arg1 - can be Constant, Variable or Temporary Format  
- its type must match Arg2, can be either real, integer or char
- c) Arg2 - can be Constant, Variable or Temporary Format  
- its type must match Arg1, can be either real, integer or char
- d) Result - Must be Temporary Format  
- its type gets assigned Boolean

7) Greater Than

Col. 1	10	25	40
>	Arg1	Arg2	Result

- a) Function - The action of this instruction is `Result := Arg1 > Arg2`
- b) Arg1 - can be Constant, Variable or Temporary Format  
- its type must match Arg2, can be either real, integer or char
- c) Arg2 - can be Constant, Variable or Temporary Format  
- its type must match Arg1, can be either real, integer or char
- d) Result - Must be Temporary Format  
- its type gets assigned Boolean

8) Less Than Equal

Col. 1	10	25	40
<=	Arg1	Arg2	Result

- a) Function - The action of this instruction is `Result := Arg1 <= Arg2`
- b) Arg1 - can be Constant, Variable or Temporary Format
  - its type must match Arg2, can be either real, integer or char
- c) Arg2 - can be Constant, Variable or Temporary Format
  - its type must match Arg1, can be either real, integer or char
- d) Result - Must be Temporary Format
  - its type gets assigned Boolean

9) Greater Than Equal

Col. 1	10	25	40
>=	Arg1	Arg2	Result

- a) Function - The action of this instruction is `Result := Arg1 >= Arg2`
- b) Arg1 - can be Constant, Variable or Temporary Format
  - its type must match Arg2, can be either real, integer or char
- c) Arg2 - can be Constant, Variable or Temporary Format
  - its type must match Arg1, can be either real, integer or char
- d) Result - Must be Temporary Format
  - its type gets assigned Boolean

10) Not Equal

Col. 1	10	25	40
<>	Arg1	Arg2	Result

- a) Function - The action of this instruction is `Result := Arg1 <> Arg2`
- b) Arg1 - can be Constant, Variable or Temporary Format
  - its type must match Arg2, can be either real, integer or char
- c) Arg2 - can be Constant, Variable or Temporary Format
  - its type must match Arg1, can be either real, integer or char
- d) Result - Must be Temporary Format
  - its type gets assigned Boolean

11) Equal

Col. 1	10	25	40
=	Arg1	Arg2	Result

- a) Function - The action of this instruction is `Result := Arg1 = Arg2`
- b) Arg1 - can be Constant, Variable or Temporary Format
  - its type must match Arg2, can be either real, integer or char
- c) Arg2 - can be Constant, Variable or Temporary Format
  - its type must match Arg1, can be either real, integer or char
- d) Result - Must be Temporary Format
  - its type gets assigned Boolean

12) Logical And

Col. 1	10	25	40
and	Arg1	Arg2	Result

- a) Function - The action of this instruction is Result := Arg1 and Arg2
- b) Arg1 - Must be Temporary Format
  - its type must be Boolean
- c) Arg2 - Must be Temporary Format
  - its type must be Boolean
- d) Result - Must be Temporary Format
  - its type gets assigned Boolean

13) Logical Or

Col. 1	10	25	40
or	Arg1	Arg2	Result

- a) Function - The action of this instruction is Result := Arg1 or Arg2
- b) Arg1 - Must be Temporary Format
  - its type must be Boolean
- c) Arg2 - Must be Temporary Format
  - its type must be Boolean
- d) Result - Must be Temporary Format
  - its type gets assigned Boolean

14) Logical Not

Col. 1	10	25	40
not	Arg1	--	Result

- a) Function - The action of this instruction is Result := not Arg1
- b) Arg1 - Must be Temporary Format
  - its type must be Boolean
- c) Result - Must be Temporary Format
  - its type gets assigned Boolean

15) Float

Col. 1	10	25	40
float	Arg1	--	Result

- a) Function - The action of this instruction is Result := Arg1 where an integer is moved into a real field
- b) Arg1 - can be Constant, Variable or Temporary Format
  - its type must be integer
- c) Result - can be Variable or Temporary Format
  - its type gets assigned real

16) Truncation

Col. 1	10	25	40
trunc	Arg1	--	Result

- a) Function - The action of this instruction is Result := Trunc(Arg1) where a real is moved into an integer field and truncated
- b) Arg1 - can be Constant, Variable or Temporary Format  
- its type must be real
- c) Result - can be Variable or Temporary Format  
- its type gets assigned integer

17) Assignment

Col. 1	10	25	40
:=	Arg1	--	Result

- a) Function - The action of this instruction is Result := Arg1 where a Arg1 is moved into the Result field and both are the same type
- b) Arg1 - can be Constant, Variable, or Temporary Format  
- its type must be real, integer or char
- c) Result - must be Variable Format  
- its type matches Arg1 or is assigned that type, which is either real, integer or char

18) Jump Subroutine

Col. 1	10	25	40
jsr	--	Arg2	--

- a) Function - The action of this instruction is an unconditional branch to the location supplied by Arg2. The address following the jsr is pushed onto a stack and later used as the return address (by the return operation)
- b) Arg2 - must be Constant Format  
- its type must be Integer

19) Return

Col. 1	10	25	40
ret	--	--	--

- a) Function - pops the element off of the return stack and branches to that address

20) Unconditional Branch

Col. 1	10	25	40
br	--	Arg2	--

- a) Function - The action of this instruction is an unconditional branch to the location supplied by Arg2
- b) Arg2 - must be Constant Format
  - its type must be Integer

21) Branch on Condition True

Col. 1	10	25	40
bct	Arg1	Arg2	--

- a) Function - The action of this instruction is a branch to the location supplied in Arg2 if the value of Arg1 is True
- b) Arg1 - must be Temporary Format
  - its type must be Boolean
- c) Arg2 - must be Constant Format
  - its type must be Integer

22) Branch on Condition False

Col. 1	10	25	40
bcf	Arg1	Arg2	--

- a) Function - The action of this instruction is a branch to the location supplied in Arg2 if the value of Arg1 is False
- b) Arg1 - must be Temporary Format
  - its type must be Boolean
- c) Arg2 - must be Constant Format
  - its type must be Integer

23) Write Real

Col. 1	10	25	40
wr	Arg1	--	--

- a) Function - The action of this instruction is to output Arg1 or Write(Arg1)
- b) Arg1 - can be Constant, Variable, or Temporary Format
  - its type must be Real

24) Write Integer

Col. 1	10	25	40
wi	Arg1	--	--

- a) Function - The action of this instruction is to output Arg1 or Write(Arg1)
- b) Arg1 - can be Constant, Variable, or Temporary Format
  - its type must be Integer

25) Write String

Col. 1	10	25	40
ws	Arg1	--	--

a) Function - The action of this instruction is to output Arg1 or Write(Arg1)

b) Arg1 - must be String Format (i.e. %s#)

26) Write Line

Col. 1	10	25	40
wl	--	--	--

a) Function - The action of this instruction is to output a carriage return and line feed

27) Read Real

Col. 1	10	25	40
rr	Arg1	--	--

a) Function - The action of this instruction is to read a real value from DataIn into Arg1 or Read(Arg1)

b) Arg1 - must be Variable Format

- If it has a type already assigned to it, it must be Real. Otherwise, it is assigned the type "Real"

28) Read Integer

Col. 1	10	25	40
ri	Arg1	--	--

a) Function - The action of this instruction is to read an integer value from DataIn into Arg1 or Read(Arg1)

b) Arg1 - must be Variable Format

- If it has a type already assigned to it, it must be Integer. Otherwise, it is assigned the type "Integer"

29) Read Line

Col. 1	10	25	40
rl	--	--	--

a) Function - The action of this instruction is to move the next line in the file DataIn or Readln

30) End of File Test

Col. 1	10	25	40
eof	--	--	Result

a) Function - The action of this instruction is to do an end of file test on DataIn and assign value to Result or Result := EOF(DataIn)

b) Result - must be Temporary Format

- its type gets assigned Boolean

31) End of Line Test

Col. 1	10	25	40
eoln	--	--	Result

- a) Function - The action of this instruction is to do an end of line test on DataIn and assign value to Result or  
Result := EOLN(DataIn)
- b) Result - must be Temporary Format  
- its type gets assigned Boolean

32) Halt

Col. 1	10	25	40
halt	--	--	--

- a) Function - The Halt instruction terminates execution of the simulator.

33) String Declaration

Col. 1	10	15
%s#	Length	Value

- a) Function - The action of this instruction is to declare and give the string name a length and value
- b) %s# - is the name of the string being declared where # is an integer up to 8 digits long
- c) Length - is the length of the value field, and is a 1 or 2 digit integer value which is RIGHT JUSTIFIED
- d) Value - is an alphanumeric value up 66 characters long

D) PROGRAMMING COMMENTS

- 1) Strings can be placed anywhere in the code. However, if they are interspersed throughout the code then they MUST be branched around or they will try to be executed. In general it is a good practice to place all string declarations after the halt instruction at the end of the code.
- 2) Not ALL errors can be caught (such as integer overflow) therefore if simulator has a runtime error check your code first to see if your doing an illegal runtime operation.

ADDITIONAL INSTRUCTION FORMATS AND FUNCTION

34) Load Array

Col. 1	10	25	40
ldar	Array_Name	Index	Arg

- a) Function - The action for this instruction is  
                   Array\_Name [Index] := Arg
- b) Array\_Name - must be an array variable
  - If the array designated by Array\_Name has been assigned a type, then the type of Arg must match that of the array.
  - If the array has does not have an assigned type, it gets the same type as that of Arg.
- c) Index - can be Constant, Variable or Temporary Format
  - Its type must be integer
- d) Arg - can be Constant, Variable or Temporary Format
  - Its type can be either integer, real or char

35) Store Array

Col. 1	10	25	40
star	Array_Name	Index	Arg

- b) Function - The action for this instruction is  
                   Arg := Array\_Name [Index]
- b) Array\_Name - must be an Array Variable
  - its type must be integer, real or char
- c) Index - can be a Constant, Variable or Temporary Format
  - Its type must be integer
- d) Arg - If Arg has not had a value (and therefore a type) assigned to it, then it is assigned the same type as that of the array. Otherwise, its type must match that of the array.

36) Write Char

Col. 1	10	25	40
wc	Arg1	--	--

- a) Function - The action of this instruction is to output Arg1 or Write(Arg1)
- b) Arg1 - can be character constant, Variable, or Temporary Format.
  - If Arg1 is a Variable or Temporary Format, its type must be Char

37) Read Char

Col. 1	10	25	40
rc	Arg1	--	--

- a) Function - The action of this instruction is to read a character value from DataIn and place it in Arg1 or Read(Arg1)
- b) Arg1 - must be Variable Format
  - If it has a type already assigned to it, it must be char. Otherwise, it is assigned the type "char"