LABORATORY FOR CONCURRENT COMPUTING SYSTEMS

COMPUTER SYSTEMS ENGINEERING
School of Electrical Engineering
Swinburne Institute of Technology
John Street, Hawthorn 3122, Victoria, Australia.

i2: An Intermediate Language for the CSIRAC II Dataflow Computer

Technical Report 31-002

G.K. Egan †
M.W. Rawling ¥
N.J.Webbψ

† School of Electrical Engineering
Swinburne Institute of Technology
John Street
Hawthorn 3122
Australia

¥ C.S.I.R.O. Division of Information Technology
ψRMIT Department of Computer Science

Version 2.0 December 1988

Also published as RMIT TR 112-68 R

Abstract:

This document describes version 2 of an intermediate language / parallel assembler for the CSIRAC II dataflow multi-processor.

1. INTRODUCTION

This document describes the syntax of version 2 of an intermediate target language (i2) for the CSIRAC II dataflow multi-processor. Version 1 of the i2 language was defined by Egan to act as an assembler for the CSIRAC II instruction set [1]. Version 2 includes inhancements to assist the implementation of GHC [2] and IF1 [3] translators.

2. SYNTAX OF 'i2'

2.1 Interpretation

The following points should be noted when interpreting the syntax of i2.

1) Mnemonics for Match Class, Function and Type are detailed in the Token and Node Definition Report [1].

2) An asterisk (*) after the word **define** indicates that the definition is shared, i.e. the body of the definition is not copied in an instantiation, instead, copies of shared graphs are distributed to all processing elements at load time. A call interface is not planted for these graphs but must be explicitly stated; this allows full control of the interface. Future versions will implement a 'general' call mechanism scheme (obtained by the special symbol '&' following **define**) which will allow the planned i2 optimiser to choose an appropriate implementation.

3) i2 does not interpret literals, they are passed directly to the 'dfo' output file. Some examples of literals follow:

   For vector types the length, lower bound precede the vector element values e.g.
   'R32V 4 -3 1.0 2.0 3.0 4.0' is a **real32_vector** with 4 elements and a lower bound -3.

   The following bit literals are equivalent **'B false', 'B 0', 'B F', 'b f'.**

   Bit_vectors are of the form
   'BV 8 0 10101010' or **'BV** 8 0 TFTFTFTF'.
   True and **false** may not be used as mnemonics for elements of **bit_vectors.**

   Examples of **Type** literals are **'T R32'** or **'T R32V** 8 -3'. In the case of vectors the length and lower bound are specified but the elements are omitted.

   An example of a **?** literals is **'Q divzero'**.

   String literals (character vectors) take the form: **'cv "here is a string"'**.

   Case is ignored in literals (except for characters and strings). The lower bound defaults to zero in all vector literals and need not be speciifed unless it is other than zero.

4) If the match class is omitted from a primitive node, then the default class for that node is used. Similarly, if a match class is specified without a node, then the default node is used for that match class, e.g.:

   dup defaults to **byp.dup**
   sto defaults to **sto.id**

5) A hash (#) before an arc identifier means that the address (a 'name' token) of the arc is to be used as an operand (see the definition of literals).

6) An exclamation mark (!) preceeding a node or graph input list causes trace to be set . Also, the definition of a subgraph (i.e. a function) may specify it to be traced by placing a '!' character before the formal input list.

7) Where an output of a node function with ordered outputs, e.g. **dst**, is not required, the predefined constant **null** should be used.

8) i2 currently defines the following constants:

null	for absorbing tokens
stdin	to read standard input
stdout	to write standard output
stderr	for standard error output
stdinctl	to set the acknowledge dest for input
stdoutctl	to set the acknowledge dest for output
stderrctl	to set the acknowledge dest for error output

NB. I/O acknowledgement addresses default to **null**.

9) Arcs may be renamed or aliased without planting identity nodes by using:

(arcname) -> alias1, alias2, ... ;

Non-deterministic merging of arcs uses the same mechanism when more than one arc appears in the input list. In this all arcs in the input list are merged and aliased to the arcs in the output list. i2 maintains consistency checks on aliased and merged arcs to ensure that 'sensible' semantics are adhered to.

10) i2 programs may be block structured, but do not require forward definition. Thus the ordering of arc and subgraph definitions is irrelevant at all levels.

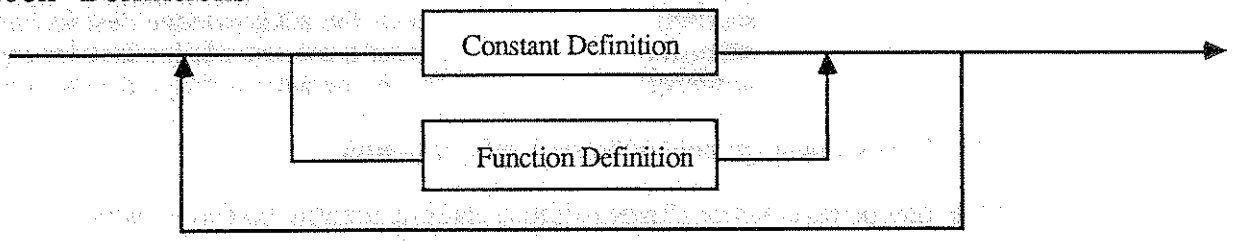
11) Support is included for all standard C pre-processor directives.

12) i2 is intended to be used as a source level loader and to this extent a driver script is used to run i2 which can take several i2 source files and catenate them prior to compilation (this is achieved by using #include directives). For more information see the i2 manual page under UNIX.

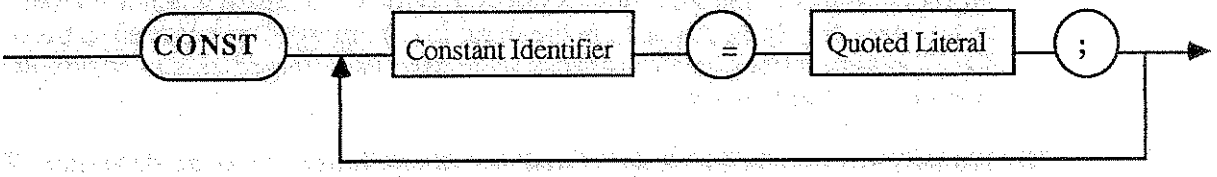
3.2 Syntax

Main Graph

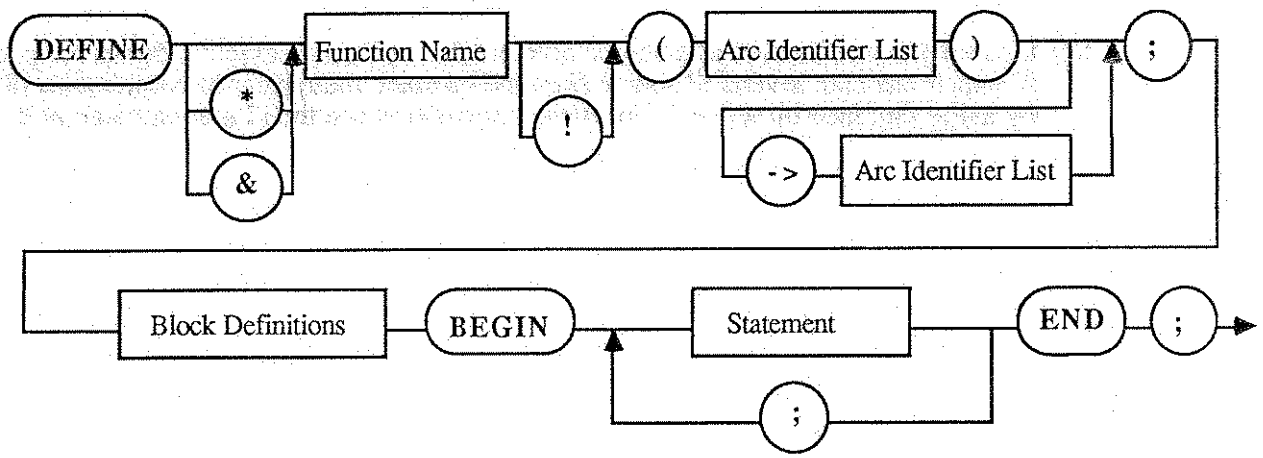
Block Definitions



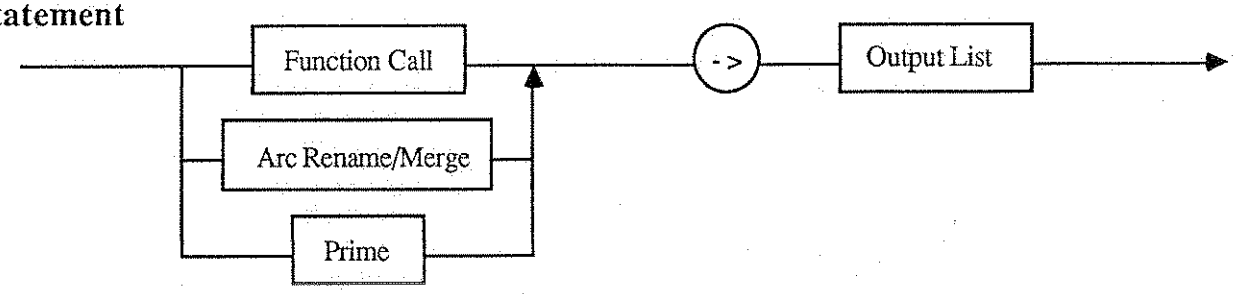
Constant Definition



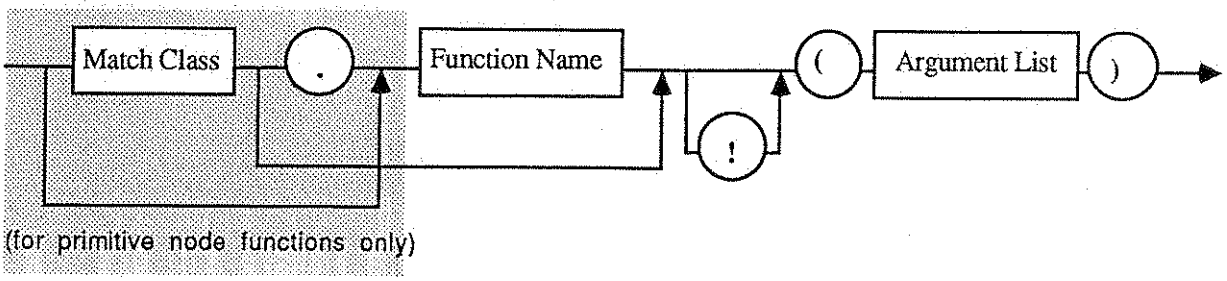
Function Definition



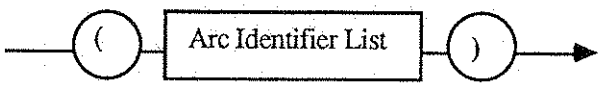
Statement



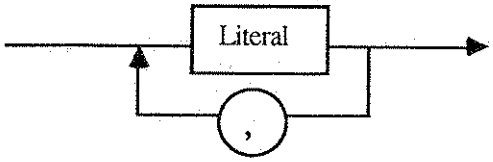
Function Call



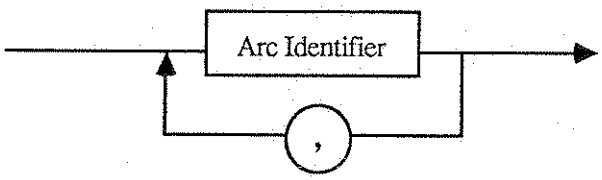
Arc Rename/Merge



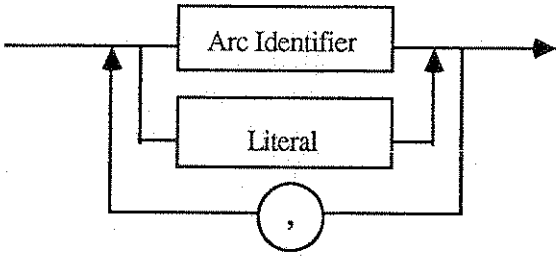
Prime



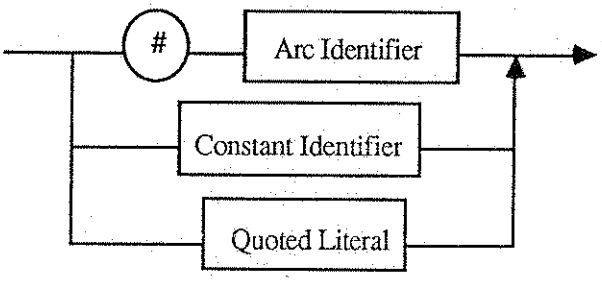
Arc Identifier List



Argument List and Output List



Literal



Quoted Literal

REFERENCES

[1] G.K. Egan, "The RMIT Data Flow Computer Token and Node Definitions", Technical Report 31-001, Laboratory for Concurrent Computing Systems, School of Electrical Engineering, Swinburne Institute of Technology, 1990.

[2] Rawling M.R., GHC on the CSIRAC II Dataflow Computer', TR 118090R, Department of Communication and Electrical Engineering, Royal Melbourne Institute of Technology, Oct. 1989.

[3] Webb N.J., 'Implementing an Applicative Language for the CSIRAC II Dataflow Computer', Department of Computer Science, Royal Melbourne Institute of Technology, *M.App.Sci. Thesis in preparation.*