Formal Derivation of a Computer

by

Li-Guo Wang and M P Fourman

Formal Derivation of a Computer

LFCS Report Series

ECS-LFCS-91-142

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

March 1991

FORMAL DERIVATION OF A COMPUTER

Li-Guo Wang[1] and M P Fourman

Laboratory for Foundations of Computer Science
Department of Computer Science, University of Edinburgh
JCMB, The King's Buildings, Edinburgh EH9 3JZ, U.K.
lgw@lfcs.ed.ac.uk

February 1991

Abstract

A synthesis logic and derivation of Mike Gordon's computer using the logic are presented. The derivation shows us that it is possible to construct realistic complex hardware using formal proof. The basic idea of the paper is 'deriving as design' and 'derivation as implementation'. The main part of the paper describes the process of derivation from specification to implementation. Other relevant aspects such as axiomatic synthesis logic, specification and implementation languages are only discussed briefly.

1 Introduction

For today's formal method for hardware design an important step is to push theory from laboratory research to practical application. To achieve the goal three things should be done 1. Formal method should be made as easy as possible for users. 2. Believable examples should be given which are realistic or can match realistic complex hardware. 3. A stronger theoretical foundation for automatic design should be presented. In order to achieve 1 and 2, we present, in this paper, an axiomatic synthesis logic and a derivation of Mike Gordon's computer. (3 will be dealt with in another paper, "Formal derivation of a class of computers")

Our work belongs to proof-based synthesis: specification-drived construction of implementation using formal proof. Our basic idea is 'deriving as design' and 'derivation as implementation'. The main contribition of the paper is that through the derivation of Mike Gordon's computer to show that it is possible to design realistic complex hardware using formal method.

We select Mike Gordon's computer as an example because it is suitable in degree of complication and in size: it matches realistic computer and its derivation can be contained in a short paper. Through the well-known computer which has been verified from implementation to specification we show a proof-based synthesis from specification to implementation (there is a little differences between our implementation which is for easy to understand synthesis and Mike Gordon's implementation) and a methodology of the synthesis. In fact our method can be used for more realistic application and we have got a stronger general result which is suitable for deriving a class of computers.

[1]The research is supported by Siemens.

In the paper we mainly describe the derivation process from the specification of behaviour level (state transition) to implementation of register-trasfer level for Mike Gordon's computer. Other relevant aspects such as axiomatic synthesis logic, specification and implementation languages are only discussed briefly.

In section 2 a simple Model-like hardware implementation language is discussed. In section 3 axiomatic synthesis logic is presented. In section 4 the specification of Mike Gordon's computer is described. In section 5 the implementation is derived from the specification using the logic. In section 6 conclusions and relevant work are discussed.

2 Hardware Specification and Implementation Languages

2.1 Specification Language

General logics such as Hol [3] and Lambda [1] are selected as specification language. We use *italic* font to write abstract or general term and predicate which correspond to the variables of implementation language below and use roman font to write concrete ones which correspond to the constant.

2.2 Implementation Language

This is a toy language of structural description style. The language is designed as minimal as possible for the paper.

2.2.1 Syntax

```
implementation ::= component | component & implementation
component ::= device (port_list)
port_list ::= port | port, port_list
device ::= variable | constant
port ::= variable | constant
```

The following font conventions will be used:

- variables will be written in *italic* font.
- constants will be written in roman font.
- capital letters are for devices and lowercase letters for ports.

2.2.2 Semantics

An implementation is a set of components which are syntactically connected by '&'. A component has device as its name and ports as its external lines. The constant expresses concrete deveice or port which have been established and variable expresses abstract ones which are taken temporarily and will be established. If two components have common port then it means there is a connection through the port.

2.2.3 Examples

REGISTER (input, *control*, output) expresses a component: its name is REGISTER and it has three ports: two ports: input and output have been established, port *control* is abstract and will be established. NAND (a, b, c) & NOT_1 (d, e) only means that devices NAND and NOT_1 are put together. NAND (a, b, c) & NOT_2 (c, d) expresses a combination of devices NAND and NOT_2 connected through their common port c.

3 Synthesis Logic

Synthesis logic (SL) is designed for hardware synthesis. Basic idea of SL is that implementation and specification are viewed as two different worlds and their unique connection is through basic rules of SL. To catch basic relationship between implementation and specification SL unites general logic (GL, for specification), construction logic (CL, for derivation) and implementation logic (IL, for implementation) for purpose of proof-based hardware synthesis.

3.1 Basic Conception

When describing SL we are at meta-language level and we will use sans serif letter which express general case at the level. I, J and K will (sometimes with subscript) be used for implementation and P, Q and S (sometimes with subscript) for specification. $\vdash \mathsf{P}$ / $\vdash \mathsf{Q}$ is called as 'proof' which means from P to prove Q. If P is axiom of GL then it is written as $\vdash \mathsf{Q}$. For example, if a is an arbitrary constant then we have $\vdash \mathsf{P\ a}$ / $\vdash \forall \mathsf{x}.\ \mathsf{P\ x}$ and also $\vdash \mathsf{P} \Rightarrow (\mathsf{Q} \Rightarrow \mathsf{P})$. $\models \mathrm{I} = \mathrm{J}$ is called as 'transform' which means implementation 'I' and 'J' are equal and can substitute each other. $\mathrm{I} \models \mathrm{S}$ is called as 'construction' which means implementation 'I' satisfies specification 'S'.

3.2 General Logic

We will directly use results proved in GL. General form for construction is
$IMP\ (x_1, x_2, \cdots, x_m, y_1, y_2, \cdots, y_n)$
$\models \forall x_1, x_2, \cdots, x_m . \exists y_1, y_2, \cdots, y_n .\ relation\ (x_1, x_2, \cdots, x_m, y_1, y_2, \cdots, y_n)$.
For short we omit pre-quantification and have below form:
$IMP\ (x_1, x_2, \cdots, x_m, y_1, y_2, \cdots, y_n) \models relation\ (x_1, x_2, \cdots, x_m, y_1, y_2, \cdots, y_n)$.

3.3 Implementation Logic

In applying IL we use *italic* capital letters to write abstract implementations which are taken, in top-down synthesis process, as temporary expressions which will be substituted by concrete implementations finally. The axioms below describe basic relations about the structure of implementation:

Axiom1: $\models \mathsf{I}\ \&\ \mathsf{I} = \mathsf{I}$.
Axiom2: $\models \mathsf{I}\ \&\ \mathsf{J} = \mathsf{J}\ \&\ \mathsf{I}$.
Axiom3: $\models (\mathsf{I}\ \&\ \mathsf{J})\ \&\ \mathsf{K} = \mathsf{I}\ \&\ (\mathsf{J}\ \&\ \mathsf{K})$.

3.4 Construction Logic

3.4.1 Basic Rules

Rule_1 and Rule_2 describe basic structure of the Construction Logic. Rule_1: if implementations I_1 and I_2 satisfy specifications S_1 and S_2 respectively then implementation $\mathsf{I}_1\ \&\ \mathsf{I}_2$ satisfies specification $\mathsf{S}_1 \wedge \mathsf{S}_2$. Rule_2: if implementation I_P satisfies specification P and impelementation $\mathsf{I}_\mathsf{P}\ \&\ \mathsf{I}_\mathsf{I}$ satisfies specification Q then implementation I_I satisfies specification $\mathsf{P} \Rightarrow \mathsf{Q}$. Rule_3 describes the relation between the General Logic and Construction Logic: if in general logic we can derive Q from P, ie $\vdash P \;/\; \vdash Q$, we take P and Q as specifications and we have implementation I satisfies P then we can derive that I satisfies Q. Rule$_4$ describe the relation between Implementation Logic and Construction Logic: if implementation I satisfies specification S and in imlementation logic we have got implementations I and J are equal then we have J satisfies S. We will directly use the results from GL and IL and omit details about how to get them in GL and IL

Rule$_1$:
$$\frac{\begin{array}{c} I_1 \models S_1 \\ I_2 \models S_2 \end{array}}{I_1 \,\&\, I_2 \models S_1 \wedge S_2}$$

Rule$_2$:
$$\frac{\begin{array}{c} I_P \models P \\ I_P \,\&\, I_I \models Q \end{array}}{I_I \models P \Rightarrow Q}$$

Rule$_3$:
$$\frac{\begin{array}{c} \vdash P / \vdash Q \\ I \models P \end{array}}{I \models Q}$$

Rule$_4$:
$$\frac{\begin{array}{c} I \models S \\ \models I = J \end{array}}{J \models S}$$

A special case of Rule$_3$ is:

$$\frac{\begin{array}{c} \vdash P \Rightarrow Q \\ I \models P \end{array}}{I \models Q}$$

3.4.2 Auxiliary Means

They are for easy of desciption of synthesis process. The Rule$_5$ is for top-down synthesis and inference is often used instead of rule.

- Substitution

If "I" expresses abstract implementation at application level of rule then we have Rule$_5$

$$\frac{I \models S}{\mathit{I} \models S}$$

- Inference

If C_i is called as premise which is proof, transform or construction ($i \in \{1, 2, \cdots, n\}$) and C is called as conclusion which is construction then

1. Rule is inference.

2. If $\frac{\begin{array}{c} C_1 \\ C_2 \\ \vdots \\ C_i \\ \vdots \\ C_n \end{array}}{C}$ is inference and $\frac{\begin{array}{c} C_{i1} \\ C_{i2} \end{array}}{C_i}$ is rule then $\frac{\begin{array}{c} C_1 \\ C_2 \\ \vdots \\ C_{i1} \\ C_{i2} \\ \vdots \\ C_n \end{array}}{C}$ is inference.

3.5 Goal, Deriving and Derivation

Goal: a construction, $I \models S$, is called goal if its specification, 'S', is to be desired.

Deriving:

1. If conclusion of a inference is goal and all premises of the inference called as known premises are as following: constructions are given axioms in CL, proofs have been proved from GL and

transforms are axioms from IL then the inference is call as deriving of the goal.
2. If conclusion of a inference is goal and its premises are known premises or can be taken as sub-goal for which we have had correspoding derivings then all derivings and the inference together are call deriving of the goal.

Derivation: If a construction, $\mathsf{I} \models \mathsf{S}$, as goal has had its deriving then the construction is called as derivation and its implementation, 'I', is called as derived implementation.

4 Specification of Mike Gordon's Computer

Mike Gordon's computer is a simple general-purpose computer invented for formal specification and verification. At the target level the computer has a memory, two registers and an idle light. The memory has a 13-bit address space of 16-bit words. The two registers are the 13-bit program counter PC and the 16-bit accumulator ACC. The idle light shows run/idle status. The formal specification of the computer at the target level is assembler language oriented. It describes semantics of the front panel operation and machine instruction set of the computer. For the specification its inputs are 'button', 'switches' and 'knob' and its outputs are 'memory', 'pc', 'acc' and 'idle'. The specification describes the state-transition of 4-tuples: memory, pc, acc and idle from time t_1 to t_2.

For the specification below types are taken:

Primitive type:
$t_1, t_2 : \mathrm{num}, \mathrm{T}, \mathrm{F} : \mathrm{bool}, \mathrm{word}_2, \mathrm{word}_3, \mathrm{word}_{13}, \mathrm{word}_{16}, \mathrm{memory}_{13\_16}.$
Derived type:
$\mathrm{knob} : \mathrm{num} \rightarrow \mathrm{word}_2$
$\mathrm{pc} : \mathrm{num} \rightarrow \mathrm{word}_{13}$
$\mathrm{switches}, \mathrm{acc} : \mathrm{num} \rightarrow \mathrm{word}_{16}$
$\mathrm{idle}, \mathrm{button} : \mathrm{num} \rightarrow \mathrm{bool}$
$\mathrm{memory} : \mathrm{num} \rightarrow \mathrm{memory}_{13\_16}$
$\mathrm{ADD}_{16}, \mathrm{SUB}_{16} : \mathrm{word}_{16} \rightarrow \mathrm{word}_{16} \rightarrow \mathrm{word}_{16}$
$\mathrm{CUT}_{16\_13} : \mathrm{word}_{16} \rightarrow \mathrm{word}_{13}$
$\mathrm{INC}_{13} : \mathrm{word}_{13} \rightarrow \mathrm{word}_{13}$
$\mathrm{OPCODE} : \mathrm{word}_{16} \rightarrow \mathrm{word}_3$
$\mathrm{FETCH}_{13} : \mathrm{memory}_{13\_16} \rightarrow \mathrm{word}_{13} \rightarrow \mathrm{word}_{16}$
$\mathrm{STORE}_{13} : \mathrm{word}_{13} \rightarrow \mathrm{word}_{16} \rightarrow \mathrm{memory}_{13\_16} \rightarrow \mathrm{memory}_{13\_16}$
$\mathrm{VAL}_n : \mathrm{word}_n \rightarrow \mathrm{num}$

specification $\overset{\mathrm{def}}{=}$
$\forall t_1 . \exists t_2 .$
(memory t_2, pc t_2, acct_2, idle t_2) =
(idle t_1 $\Rightarrow$
(button t_1 $\Rightarrow$
((VAL$_2$ (knob t_1) = 0) $\Rightarrow$
(memory t_1, CUT$_{16\_13}$ (switches t_1), acc t_1, T) |
((VAL$_2$ (knob t_1) = 1) $\Rightarrow$
(memory t_1, pc t_1, switches t_1, T) |

```
   (($VAL_2$ (knob $t_1$) = 2) $\Rightarrow$
    ($STORE_{13}$ (pc $t_1$) (acc $t_1$) (memory $t_1$), pc$t_1$, acc $t_1$, T) |
    (memory $t_1$, pc$t_1$, acc $t_1$, F))
  (memory $t_1$, pc$t_1$, acc $t_1$, T)) |
 (button $t_1$ $\Rightarrow$
  (memory $t_1$, pc$t_1$, acc $t_1$, T) |
  EXECUTE (memory $t_1$, pc $t_1$, acc $t_1$)))
$\wedge$
EXECUTE (memory t)1, pc $t_1$, acc $t_1$) =
 let op = $VAL_3$ (OPCODE ($FETCH_{13}$ (memory $t_1$) (pc $t_1$))) in
 let addr = $CUT_{16\_13}$($FETCH_{13}$(memory $t_1$ (pc $t_1$)) in
 ((op = 0) $\Rightarrow$ (memory $t_1$, pc $t_1$, acc $t_1$, T) |
  (op = 1) $\Rightarrow$ (memory $t_1$, addr, acc $t_1$, F) |
  (op = 2) $\Rightarrow$
   (($VAL_{16}$ acc $t_1$ = 0) $\Rightarrow$
    (memory $t_1$, addr, acc $t_1$, F) |
    (memory $t_1$, $INC_{13}$ (pc $t_1$), acc $t_1$, F) |
  (op = 3) $\Rightarrow$
   (memory $t_1$, $INC_{13}$ (pc $t_1$),
    $ADD_{16}$ (acc $t_1$) ($FETCH_{13}$ (memory $t_1$) addr), F) |
  (op = 4) $\Rightarrow$
   (memory $t_1$, $INC_{13}$ (pc $t_1$),
    $SUB_{16}$ (acc $t_1$) ($FETCH_{13}$ (memory $t_1$) addr), F) |
  (op = 5) $\Rightarrow$
   (memory $t_1$, $INC_{13}$ (pc $t_1$), $FETCH_{13}$ (memory $t_1$) addr, F) |
  (op = 6) $\Rightarrow$
   ($STORE_{13}$ addr (acc $t_1$)(memory $t_1$), $INC_{13}$ (pc $t_1$), acc $t_1$, F) |
   (memory $t_1$, $INC_{13}$ (pc $t_1$), acc $t_1$, F))
```

5 Deriving Implementation from Specification

Starting point of the derivation is goal:

$$\textit{COMPUTER}\,(\text{button}, \text{knob}, \text{switches}, \text{memory}, \text{pc}, \text{acc}, \text{idle}) \models \text{specification}$$

5.1 Refinement of Specification

We use '$\vdash$ imply_and_form / $\vdash$ *specification*' to refine the specification then by $Rule_3$ our goal will become '*COMPUTER* $\models$ imply_and_form'. The imply_and_form is as following:

imply_and_form = $path_1 \wedge path_2 \wedge \cdots \wedge path_{15}$

$path_1$ = idle t_{11} $\wedge$ $\neg$button t_{11} $\Rightarrow$
  (memory t_{21} = memory t_{11} $\wedge$ pc t_{21} = pc t_{11} $\wedge$ acc t_{21} = acc t_{11} $\wedge$ idle t_{21} = T)

$path_2$ = idle t_{12} $\wedge$ button t_{12} $\wedge$ VAL_2 (knob t_{12}) = 0 $\Rightarrow$
  (memory t_{22} = memory t_{12} $\wedge$ pc t_{22} = CUT_{16_13} (switches t_{12}) $\wedge$ acc t_{22} = acc $t_{12}$$\wedge$
  idle t_{22} = T)

$path_3$ = idle t_{13} $\wedge$ button t_{13} $\wedge$ VAL_2 (knob t_{13}) = 1 $\Rightarrow$
  (memory t_{23} = memory t_{13} $\wedge$ pc t_{23} = pc t_{13} $\wedge$ acc t_{23} = switches t_{13} $\wedge$ idle t_{23} = T)

$\text{path}_4 = \text{idle } t_{14} \wedge \text{button } t_{14} \wedge \text{VAL}_2\ (\text{knob } t_{14}) = 2 \Rightarrow$
$\quad(\text{memory } t_{24} = \text{STORE}_{13}\ (\text{pc } t_{14})\ (\text{acc } t_1)\ (\text{memory } t_{14}) \wedge \text{pc } t_{24} = \text{pc } t_{14} \wedge$
$\quad\ \text{acc } t_{24} = \text{acc } t_{14} \wedge \text{idle } t_{24} = \text{T})$
$\text{path}_5 = \text{idle } t_{15} \wedge \text{button } t_{15} \wedge \text{VAL}_2\ (\text{knob } t_{15}) = 3 \Rightarrow$
$\quad(\text{memory } t_{25} = \text{memory } t_{15} \wedge \text{pc } t_{25} = \text{pc } t_{15} \wedge \text{acc } t_{25} = \text{acc } t_{15} \wedge \text{idle } t_{25} = \text{F})$
$\text{path}_6 = \neg\text{idle } t_{16} \wedge \text{button } t_{16} \Rightarrow$
$\quad(\text{memory } t_{26} = \text{memory } t_{16} \wedge \text{pc } t_{26} = \text{pc } t_{16} \wedge \text{acc } t_{26} = \text{acc } t_{16} \wedge \text{idle } t_{26} = \text{T})$
$\text{path}_7 = \neg\text{idle } t_{17} \wedge \neg\text{button } t_{17} \wedge \text{op}_7 = 0 \Rightarrow$
$\quad(\text{memory } t_{27} = \text{memory } t_{17} \wedge \text{pc } t_{27} = \text{pc } t_{17} \wedge \text{acc } t_{27} = \text{acc } t_{17} \wedge \text{idle } t_{27} = \text{T})$
$\text{path}_8 = \neg\text{idle } t_{18} \wedge \neg\text{button } t_{18} \wedge \text{op}_8 = 1 \Rightarrow$
$\quad(\text{memory } t_{28} = \text{memory } t_{18} \wedge \text{pc } t_{28} = \text{addr}_8 \wedge \text{acc } t_{28} = \text{acc } t_{18} \wedge \text{idle } t_{28} = \text{F})$
$\text{path}_9 = \neg\text{idle } t_{19} \wedge \neg\text{buttont}_{19} \wedge \text{op}_9 = 2 \wedge \text{VAL}_{16}\ (\text{acc } t_{19}) = 0 \Rightarrow$
$\quad(\text{memory } t_{29} = \text{memory } t_{19} \wedge \text{pc } t_{29} = \text{addr}_9 \wedge \text{acc } t_{29} = \text{acc } t_{19} \wedge \text{idle } t_{29} = \text{F})$
$\text{path}_{10} = \neg\text{idle } t_{110} \wedge \neg\text{button } t_{110} \wedge \text{op}_{10} = 2 \wedge \text{VAL}_{16}\ (\text{acc } t_{110}) \neq 0 \Rightarrow$
$\quad(\text{memory } t_{210} = \text{memory } t_{110} \wedge \text{pc } t_{210} = \text{INC}_{13}\ (\text{pc } t_{110}) \wedge \text{acc } t_{210} = \text{acc } t_{110} \wedge \text{idle } t_{210} = \text{F})$
$\text{path}_{11} = \neg\text{idle } t_{111} \wedge \neg\text{button } t_{111} \wedge \text{op}_{11} = 3 \Rightarrow$
$\quad(\text{memory } t_{211} = \text{memory } t_{111} \wedge \text{pc } t_{211} = \text{INC}_{13}\ (\text{pc } t_{111}) \wedge$
$\quad\ \text{acc } t_{211} = \text{ADD}_{16}\ (\text{acc } t_{111})\ (\text{FETCH}_{13}\ (\text{memory } t_{111})\ \text{addr}_{11}) \wedge \text{idle } t_{211} = \text{F})$
$\text{path}_{12} = \neg\text{idle } t_{112} \wedge \neg\text{button } t_{112} \wedge \text{op}_{12} = 4 \Rightarrow$
$\quad(\text{memory } t_{212} = \text{memory } t_{112} \wedge \text{pc } t_{212} = \text{INC}_{13}\ (\text{pc } t_{112}) \wedge$
$\quad\ \text{acc } t_{212} = \text{SUB}_{16}\ (\text{acc } t_{112})\ (\text{FETCH}_{13}\ (\text{memory } t_{112})\ \text{addr}_{12}) \wedge \text{idle } t_{212} = \text{F})$
$\text{path}_{13} = \neg\text{idle } t_{113} \wedge \neg\text{button } t_{113} \wedge \text{op}_{13} = 5 \Rightarrow$
$\quad(\text{memory } t_{213} = \text{memory } t_{113} \wedge \text{pc } t_{213} = \text{INC}_{13}\ (\text{pc } t_{113}) \wedge$
$\quad\ \text{acc } t_{213} = \text{FETCH}_{13}\ (\text{memory } t_{113})\ \text{addr}_{13} \wedge \text{idle } t_{213} = \text{F})$
$\text{path}_{14} = \neg\text{idle } t_{114} \wedge \neg\text{button } t_{114} \wedge \text{op}_{14} = 6 \Rightarrow$
$\quad(\text{memory } t_{214} = \text{STORE}_{13}\ \text{addr}_{14}\ (\text{acc } t_{114})\ (\text{memory } t_{114}) \wedge \text{pc } t_{214} = \text{INC}_{13}\ (\text{pc } t_{114}) \wedge$
$\quad\ \text{acc } t_{214} = \text{acc } t_{14} \wedge \text{idle } t_{214} = \text{F})$
$\text{path}_{15} = \neg\text{idle } t_{115} \wedge \neg\text{button } t_{115} \wedge \text{op}_{15} = 7 \Rightarrow$
$\quad(\text{memory } t_{215} = \text{memory } t_{115} \wedge \text{pc } t_{215} = \text{INC}_{13}\ (\text{pc } t_{115}) \wedge \text{acc } t_{215} = \text{acc } t_{115} \wedge \text{idle } t_{215} = \text{F})$

Here t_{1i} is an arbitrary constant. t_{2i} is a function of t_{1i}. $i \in \{1, 2, \cdots, 15\}$
$\text{op}_j = \text{VAL}_3\ (\text{OPCODE}\ (\text{FETCH}_{13}\ (\text{memory } t_{1j})\ (\text{pc } t_{1j}))\quad j \in \{7, 8, \cdots, 15\}$
$\text{addr}_k = \text{CUT}_{16.13}\ (\text{FETCH}_{13}\ (\text{memory } t_{1k})\ (\text{pc } t_{1k}))\quad k \in \{8, 9, 11, 12, 13, 14\}$

Our task has now been reduced to proving:
$IMP_i \models \text{path}_i\ (i \in \{1, 2, \cdots, 15\})\ (COMPUTER = IMP_1\ \&\ IMP_2\ \&\ \cdots\ \&\ IMP_{15})$.
We will give some details through the derivation of path_2 for understanding derivation process and then will give general analysis through the dercibing main steps of the derivation of path_{12}. In next section let us first give some axioms which describe basic devices at rtl of implementation.

5.2 Relative Axioms and Theorems

For succinctness of description we suppose that only at the first appearance of a device we give its ports and omit the ports in following derivation if they are same. For deriving data part below additional types and axioms are given to describe basic devices used for data part at rtl:

Primitive types: $\text{FLOAT}_{16} : \text{tri\_word}_n$
Derived type:

$i_{g1}, o_{r1}, a : \text{num} \rightarrow \text{word}_{13}$
$i_{g2}, o_b, i_{r1}, i_{r2}, o_{r2}, i_f, o_f, d, i_{a1}, i_{a2}, o_a : \text{num} \rightarrow \text{word}_{16}$
$c_{g1}, c_{g2}, c_{r1}, c_{r2}, r, w, inc, add, sub : \text{num} \rightarrow \text{bool}$
$o_{g1}, o_{g2}, i_{bj}, o_m : \text{num} \rightarrow \text{tri\_word}_{16} \;\; \text{j} \in \{1, 2, \cdots, \text{n}\}$
$\text{PAD}_{13\_16} : \text{word}_{13} \rightarrow \text{word}_{16}$
$\text{MK\_TRI}_{16} : \text{word}_{16} \rightarrow \text{tri\_word}_{16}$
$\text{DEST\_TRI}_{16} : \text{tri\_word}_{16} \rightarrow \text{word}_{16}$
$\cup_{16} : \text{tri\_word}_{16} \rightarrow \text{tri\_word}_{16} \rightarrow \text{tri\_word}_{16}$
We have axioms for DEST_TRI, MK_TRI and $\cup_n$:
$\vdash \forall w : \text{word}_n.\ \text{DEST\_TRI}_n\ (\text{MK\_TRI}_n\ w) = w$
$\vdash \forall w : \text{tri\_word}_n.\ (\text{FLOAT}_n\ \cup_n\ w = w) \wedge (w\ \cup_n\ \text{FLOT}_n = w)$

Axioms:
$\text{G}_{13}\ (i_{g1}, c_{g1}, o_{g1}) \models \forall t.\ o_{g1}\ t = (c_{g1}\ t \Rightarrow \text{MK\_TRI}_{16}\ (\text{PAD}_{13\_16}\ (i_{g1}\ t)) \mid \text{FLOAT}_{16})$
$\text{G}_{16}\ (i_{g2}, c_{g2}, o_{g2}) \models \forall t.\ o_{g2}\ t = (c_{g2}\ t \Rightarrow \text{MK\_TRI}_{16}\ (i_{g2}\ t) \mid \text{FLOAT}_{16})$
$\text{BUS}\ (i_{b1}, i_{b2}, \cdots, i_{bn}, o_b) \models$
$\forall t.\ o_b\ t = \text{DEST\_TRI}_{16}\ (i_{b1}\ t \cup_{16} i_{b2}\ t \cup_{16} \cdots \cup_{16} i_{bn}\ t)$
$\text{REG}_{13}\ (i_{r1}, c_{r1}, o_{r1}) \models \forall t.\ o_{r1}\ (t+1) = (c_{r1}\ t \Rightarrow \text{CUT}_{16\_13}\ (i_{r1}\ t) \mid o_{r1}\ t)$
$\text{REG}_{16}\ (i_{r2}, c_{r2}, o_{r2}) \models \forall t.\ o_{r2}\ (t+1) = (c_{r2}\ t \Rightarrow\ i_{r2}\ t \mid o_{r2}\ t)$
$\text{BUF}\ (i_f, o_f) \models \forall t.\ o_f\ (t+1) = i_f\ t$
$\text{MEM}\ (memory, a, d, r, w, o_m) \models$
$\forall t.(o_m\ t = (r\ t \Rightarrow \text{MK\_TRI}_{16}\ (\text{FETCH}_{13}\ (memory\ t)\ (a\ t)) \mid \text{FLOAT}_{16})) \wedge$
$(memory\ (t+1) = (w\ t \Rightarrow\ \text{STORE}_{13}\ (a\ t)\ (d\ t)\ (memory\ t) \mid memory\ t))$
$\text{ALU}\ (i_{a1}, i_{a2}, inc, add, sub, o_a) \models$
$\forall t.\ o_a\ t = (inc\ t \Rightarrow \text{INC}_{16}\ i_{a2}\ t \mid (add\ t \Rightarrow \text{ADD}_{16}\ (i_{a1}\ t)\ (i_{a2}\ t) \mid$
$(\text{sub t} \Rightarrow \text{SUB}_{16}\ (\text{i}_{\text{a1}}\ \text{t})\ (\text{i}_{\text{a2}}\ \text{t}) \mid \text{i}_{\text{a2}}\ \text{t})))$
Theorem_1 : (the types of the variables in the theorem will be determined in context of applycations)

$$\begin{array}{l}
IMP_r \models r\ t \\
IMP_w \models w\ t \\
GATE\ (i_g, r, o_g) \models \forall t.\ r\ t \Rightarrow (o_g\ t = f\ (i_g\ t)) \\
BUS\ (\cdots, o_g, \cdots, o_b) \models \forall t.\ o_b\ t = g\ (\cdots \cup o_g\ t \cup \cdots) \\
REG\ (o_b, w, o_r) \models \forall t.\ w\ t \Rightarrow (o_r\ (t+1) = h\ (o_b\ t)) \\
\hline
IMP_r\ \&\ IMP_w\ \&\ GATE\ \&\ BUS\ \&\ REG \models o_r\ (t+1) = h\ g\ f\ (i_g\ t)
\end{array}$$

Below additional types and axioms are for control part at rtl:

Primitive types:
$control_i : \text{word}_{16}$
$n_i : \text{word}_3$
$tf : \text{bool}$
$address_i, addressa_i, addressb_iq : \text{word}_5$
Derived types:
$addrs_i, addrs_{ad}, addrs_s, addrs_a, addrs_b, nextaddrs : \text{num} \rightarrow \text{word}_5$
$control : num \rightarrow word_{16}$
$condition, code_j : \text{num} \rightarrow \text{bool} \;\; \text{j} \in \{1, 2, \cdots, \text{m}\}$
$test : num \rightarrow word_3$
$f_{br} : (\text{bool} * \text{bool} * \text{word}_5 * \text{word}_5) \rightarrow \text{word}_5$

$\mathrm{WORD_n} : \mathrm{num} \to \mathrm{word_n}$
We have axiom for the $\mathrm{WORD_n}$ and $\mathrm{VAL_n}$:
$\vdash \forall w : \mathrm{word_n}.\ \mathrm{WORD_n}(\mathrm{VAL_n}\ w) = w$

Axioms: (Note: constant '$address_i, control_i, n_i, addressa_i, addressb_i, tf, f_{br}$'
will be established in derivation process.)

$\mathrm{INSTRUCTION}\ (addrs_i, control, test, addrs_a, addrs_b)$
$\models \mathrm{instruction_0}\ (addrs_i, control, test, addrs_a, addrs_b) \wedge$
$\mathrm{instruction_1}\ (addrs_i, control, test, addrs_a, addrs_b) \wedge$
$\cdots$
$\mathrm{instruction_n}\ (addrs_i, control, test, addrs_a, addrs_b)$
$\vdash_{def} \mathrm{instruction_i}\ (addrs_i, control, test, addrs_a, addrs_b) =$
$\forall t.\ (addrs_i\ t = address_i \Rightarrow$
$control\ t = control_i \wedge test\ t = n_i \wedge addrs_a\ t = addressa_i \wedge addrs_b\ t = addressb_i)$

$\mathrm{DECODE}\ (control, code_1, code_2, \cdots, code_m)$
$\models \mathrm{decode_0}\ (control, code_1, code_2, \cdots, code_m) \wedge$
$\mathrm{decode_1}\ (control, code_1, code_2, \cdots, code_m) \wedge$
$\cdots$
$\mathrm{decode_n}\ (control, code_1, code_2, \cdots, code_m)$
$\vdash_{def} \mathrm{decode_i}\ (control, code_1, code_2, \cdots, code_m) =$
$\forall t.\ (control\ t = control_i \Rightarrow$
$\cdot\ code_1\ t = tf \wedge code_2\ t = tf \wedge \cdots \wedge code_m\ t = tf)\quad i \in \{0, 1, \cdots, n\}$

$\mathrm{BRANCH}\ (test, condition, addrs_a, addrs_b, nextaddrs)$
$\models \forall t.\ nextaddrs\ t = f_{br}\ ((test\ t), (condition\ t), (addrs_a\ t), (addrs_b\ t))$
$\mathrm{ADDRS}\ (nextaddrs, addrs_{ad}) \models \forall t.\ addrs_{ad}\ (t+1) = nextaddrs\ t$
$\mathrm{HAND} \models \forall t.\ condition\ t$
$\mathrm{START} \models \forall t.\ addrs_s\ t = address_0$

Note: HAND is a special device which express operation by our hand for button, knob and switch. START is too a special device: electric stimulation which, as 'first push', start the running of the microprogram of the computer.

5.3 Deriving Path$_2$ as Example

5.3.1 Deriving Data Part

$$\frac{\begin{array}{l} IMP_{2\_c} \models \mathrm{idle}\ t_{12} \wedge \mathrm{button}\ t_{12} \wedge \mathrm{VAL_2}\ (\mathrm{knob}\ t_{12}) = 0 \\ IMP_{2\_c}\ \&\ IMP_2 \\ \models \mathrm{memory}\ t_{22} = \mathrm{memory}\ t_{12} \wedge \mathrm{pc}\ t_{22} = \mathrm{CUT_{16\_13}}\ (\mathrm{switches}\ t_{12}) \wedge \mathrm{acc}\ t_{22} = \mathrm{acc}\ t_{12} \wedge \mathrm{idle}\ t_{22} = \mathrm{T} \end{array}}{IMP_2 \models \mathrm{path_2}}$$

$$\frac{\begin{array}{l} IMP_{2\_m} \models \mathrm{memory}\ t_{22} = \mathrm{memory}\ t_{12} \\ IMP_{2\_p} \models \mathrm{pc}\ t_{22} = \mathrm{CUT_{16\_13}}\ (\mathrm{switches}\ t_{12}) \\ IMP_{2\_a} \models \mathrm{acc}\ t_{22} = \mathrm{acc}\ t_{12} \\ IMP_{2\_i} \models \mathrm{idle}\ t_{22} = \mathrm{T} \end{array}}{\begin{array}{l} IMP_{2\_c}\ \&\ IMP_2 \\ \models \mathrm{memory}\ t_{22} = \mathrm{memory}\ t_{12} \wedge \mathrm{pc}\ t_{22} = \mathrm{CUT_{16\_13}}\ (\mathrm{switches}\ t_{12}) \wedge \mathrm{acc}\ t_{22} = \mathrm{acc}\ t_{12} \wedge \mathrm{idle}\ t_{22} = \mathrm{T} \end{array}}$$

$$\begin{array}{l}
IMP_{\neg write} \models \neg\mathrm{write}\ \mathrm{t}_{12} \wedge \neg\mathrm{write}\ (\mathrm{t}_{12}+1) \wedge \cdots \wedge \neg\mathrm{write}\ (\mathrm{t}_{22}-1) \\
\mathrm{MEM}\ (\mathrm{memory}, a, d, \mathit{read}, \mathrm{write}, o) \models \forall\mathrm{t}.\ \neg\mathrm{write}\ \mathrm{t} \Rightarrow \mathrm{memory}\ (\mathrm{t}+1) = \mathrm{memory}\ \mathrm{t} \\
\hline
IMP_{\neg write}\ \&\ \mathrm{MEM} \models \mathrm{memory}\ \mathrm{t}_{22} = \mathrm{memory}\ \mathrm{t}_{12}
\end{array}$$

Here we introduce a special device '$\mathrm{HAND_{sw}}$' which keeps the value of switch same from time t_{12} to $\mathrm{t}_{22}-1$. Intuitively the device is just our hand.

$$\begin{array}{l}
\mathrm{HAND_{sw}} \models \mathrm{switches}\ (\mathrm{t}_{22}-1) = \mathrm{switches}\ \mathrm{t}_{12} \\
IMP_{rsw} \models \mathrm{rsw}\ (\mathrm{t}_{22}-1) \\
IMP_{wpc} \models \mathrm{wpc}\ (\mathrm{t}_{22}-1) \\
\mathrm{GATE_{sw}}\ (\mathrm{switches}, \mathrm{rsw}, \mathrm{o_g}) \models \forall\mathrm{t}.\ \mathrm{rsw}\ \mathrm{t} \Rightarrow (\mathrm{o_g}\ \mathrm{t} = \mathrm{MK\_TRI_{16}}\ (\mathrm{switches}\ \mathrm{t})) \\
\mathrm{BUS}\ (\cdots, \mathrm{o_g}, \cdots, \mathrm{o_b}) \models \forall\mathrm{t}.\ \mathrm{o_b}\ \mathrm{t} = \mathrm{DEST\_TRI_{16}}\ (\cdots \cup_{16}\ \mathrm{o_g}\ \mathrm{t} \cup_{16} \cdots) \\
\mathrm{REG_{pc}}\ (\mathrm{o_b}, \mathrm{wpc}, \mathrm{pc}) \models \forall\mathrm{t}.\ \mathrm{wpc}\ \mathrm{t} \Rightarrow (\mathrm{pc}\ (\mathrm{t}+1) = \mathrm{CUT_{16\_13}}\ (\mathrm{o_b}\ \mathrm{t})) \\
\hline
\mathrm{HAND_{sw}}\ \&\ IMP_{rsw}\ \&\ IMP_{wpc}\ \&\ \mathrm{GATE_{sw}}\ \&\ \mathrm{BUS}\ \&\ \mathrm{REG_{pc}} \models \mathrm{pc}\ \mathrm{t}_{22} = \mathrm{CUT_{16\_13}}(\mathrm{switches}\ \mathrm{t}_{12})
\end{array}$$

$$\begin{array}{l}
IMP_{\neg wacc} \models \neg\mathrm{wacc}\ \mathrm{t}_{12} \wedge \neg\mathrm{wacc}\ (\mathrm{t}_{12}+1) \wedge \cdots \wedge \neg\mathrm{wacc}\ (\mathrm{t}_{22}-1) \\
\mathrm{REG_{acc}}\ (i_{acc}, \mathrm{wacc}, \mathrm{acc}) \models \forall\mathrm{t}.\ \neg\mathrm{wacc}\ \mathrm{t} \Rightarrow \mathrm{acc}\ (\mathrm{t}+1) = \mathrm{acc}\ \mathrm{t} \\
\hline
IMP_{\neg wacc}\ \&\ \mathrm{REG_{acc}} \models \mathrm{acc}\ \mathrm{t}_{22} = \mathrm{acc}\ \mathrm{t}_{12}
\end{array}$$

$$\begin{array}{l}
IMP_{\neg write}\ \&\ \mathrm{MEM} \models \mathrm{memory}\ \mathrm{t}_{22} = \mathrm{memory}\ \mathrm{t}_{12} \\
\mathrm{HAND_{sw}}\ \&\ IMP_{rsw}\ \&\ IMP_{wpc}\ \&\ \mathrm{GATE_{sw}}\ \&\ \mathrm{BUS}\ \&\ \mathrm{REG_{pc}} \\
\models \mathrm{pc}\ \mathrm{t}_{22} = \mathrm{CUT_{16\_13}}(\mathrm{switches}\ (\mathrm{t}_{22}-1)) \\
IMP_{\neg wacc}\ \&\ \mathrm{REG_{acc}} \models \mathrm{acc}\ \mathrm{t}_{22} = \mathrm{acc}\ \mathrm{t}_{12} \\
IMP_{2\_i} \models \mathrm{idle}\ \mathrm{t}_{22} = \mathrm{T} \\
\hline
IMP_{\neg write}\ \&\ \mathrm{MEM}\ \&\ \mathrm{HAND_{sw}}\ \&\ IMP_{rsw}\ \&\ IMP_{wpc}\ \&\ \mathrm{GATE_{sw}}\ \&\ \mathrm{BUS}\ \&\ \mathrm{REG_{pc}}\ \&\ IMP_{\neg wacc}\ \& \\
\mathrm{REG_{acc}}\ \&\ IMP_{2\_i} \\
\models \mathrm{memory}\ \mathrm{t}_{22} = \mathrm{memory}\ \mathrm{t}_{12} \wedge \mathrm{pc}\ \mathrm{t}_{22} = \mathrm{CUT_{16-13}}\ \mathrm{t}_{12} \wedge \mathrm{acc}\ \mathrm{t}_{22} = \mathrm{acc}\ \mathrm{t}_{12} \wedge \mathrm{idel}\ \mathrm{t}_{22} = \mathrm{T}
\end{array}$$

We have derived the implementation of the data part for path_2:
$\mathrm{HAND_{sw}}\ \&\ \mathrm{MEM}\ (\mathrm{memory}, a, d, \mathit{read}, \mathrm{write}, o)\ \&\ \mathrm{GATE_{sw}}\ (\mathrm{swtches}, \mathrm{rsw}, \mathrm{o_g})\ \&$
$\mathrm{BUS}\ (\cdots, \mathrm{o_g}, \cdots, \mathrm{o_b})\ \&\ \mathrm{REG_{pc}}\ (\mathrm{o_b}, \mathrm{wpc}, \mathrm{pc})\ \&\ \mathrm{REG_{acc}}\ (i_{acc}, \mathrm{wacc}, \mathrm{acc})$.
Remain abstract implementations, $IMP_{2\_c}, IMP_{\neg write}, IMP_{\neg acc}, IMP_{rsw}, IMP_{wpc}$ and IMP_{2_i}, will be derived in control part.

5.3.2 Deriving Control Part

In below derivation the implementations, 'INSTRUCTION, DECODE and BRANCH', will be constructed step by step following the derivation process so they will be variables until whole implementation of the computer is derived.

$$\begin{array}{l}
\mathrm{START} \models \mathrm{addrs}\ \mathrm{t}_{12} = \mathrm{address}_0 \\
INSTRUCTION_{c0}\ (\mathrm{addrs}, \mathrm{control}, \mathit{test}, \mathit{addrs}_a, \mathit{addrs}_b) \\
\models \mathrm{addrs}\ \mathrm{t}_{12} = \mathrm{address}_0 \Rightarrow \mathrm{control}\ \mathrm{t}_{12} = \mathrm{control}_0 \\
DECODE_0\ (\mathrm{control}, \mathrm{idel}, \mathrm{write}, \mathrm{wacc},) \\
\models \mathrm{control}\ \mathrm{t}_{12} = \mathrm{control}_0 \Rightarrow \mathrm{idle}\ \mathrm{t}_{12} = \mathrm{T} \wedge \mathrm{write}\ \mathrm{t}_{12} = \mathrm{F} \wedge \mathrm{wacc}\ \mathrm{t}_{12} = \mathrm{F} \\
\hline
\mathrm{START}\ \&\ INSTRUCTION_{c0}\ \&\ DECODE_0 \\
\models \mathrm{idle}\ \mathrm{t}_{12} = \mathrm{T} \wedge \mathrm{write}\ \mathrm{t}_{12} = \mathrm{F} \wedge \mathrm{wacc}\ \mathrm{t}_{12} = \mathrm{F}
\end{array}$$

$\mathrm{START} \models \mathrm{addrs}\ \mathrm{t}_{12} = \mathrm{address}_0$
$\mathit{INSTRUCTION}_{a0}\ (\mathrm{addrs}, \mathrm{control}, \mathrm{test}, \mathit{addrs}_a, \mathrm{addrs}_b)$
$\models \mathrm{addrs}\ \mathrm{t}_{12} = \mathrm{address}_0 \Rightarrow \mathrm{VAL}_3\ (\mathrm{test}\ \mathrm{t}_{12}) = 1 \wedge \mathrm{addrs}_b\ \mathrm{t}_{12} = \mathrm{address}_1$
$\mathrm{HAND}_{\mathrm{button}} \models \mathrm{button}\ \mathrm{t}_{12}$
$\mathit{BRANCH}_0\ (\mathrm{test}, \mathrm{button}, \mathrm{addrs}_b, \mathrm{nextarres})$
$\models \mathrm{VAL}_3\ (\mathrm{test}\ \mathrm{t}_{12}) = 1 \wedge \mathrm{button}\ \mathrm{t}_{12} \Rightarrow \mathrm{nextaddrs}\ \mathrm{t}_{12} = \mathrm{addrs}_b\ \mathrm{t}_{12}$
$\mathrm{ADDRS}\ (\mathrm{nextaddrs}, \mathrm{addrs}) \models \mathrm{addrs}\ (\mathrm{t}_{12} + 1) = \mathrm{nextaddrs}\ \mathrm{t}_{12}$

$\mathrm{START}\ \&\ \mathrm{HAND}_{\mathrm{button}}\ \&\ \mathit{INSTRUCTION}_{a0}\ \&\ \mathit{BRANCH}_0\ \&\ \mathrm{ADDRS} \models \mathrm{addrs}\ (\mathrm{t}_{12} + 1) = \mathrm{address}$

$\mathrm{START}\ \&\ \mathrm{HAND}_{\mathrm{button}}\ \&\ \mathit{INSTRUCTION}_{a0}\ \&\ \mathit{BRANCH}_0\ \&\ \mathrm{ADDRS} \models \mathrm{addrs}\ (\mathrm{t}_{12} + 1) = \mathrm{address}_1$
$\mathit{INSTRCUCTION}_{c1} \models \mathrm{addrs}\ (\mathrm{t}_{12} + 1) = \mathrm{address}_1 \Rightarrow \mathrm{control}\ (\mathrm{t}_{12} + 1) = \mathrm{control}_1$
$\mathit{DECODE}_1\ (\mathit{control}, \mathit{write}, \mathit{wacc})\ (\mathit{control}, \mathit{write}, \mathit{wacc})$
$\models \mathrm{control}\ (\mathrm{t}_{12} + 1) = \mathrm{control}_1 \Rightarrow \mathrm{write}\ (\mathrm{t}_{12} + 1) = \mathrm{F} \wedge \mathrm{wacc}\ (\mathrm{t}_{12} + 1) = \mathrm{F}$

$\mathrm{START}\ \&\ \mathrm{HAND}_{\mathrm{button}}\ \&\ \mathit{INSTRUCTION}_{a0}\ \&\ \mathit{BRANCH}_0\ \&\ \mathrm{ADDRS}\ \&\ \mathit{INSTRCUCTION}_{c1}\ \&$
$\mathit{DECODE}_1 \models \mathrm{write}\ (\mathrm{t}_{12} + 1) = \mathrm{F} \wedge \mathrm{wacc}\ (\mathrm{t}_{12} + 1) = \mathrm{F}$

$\mathrm{START}\ \&\ \mathrm{HAND}_{\mathrm{button}}\ \&\ \mathit{INSTRUCTION}_{a0}\ \&\ \mathit{BRANCH}_0\ \&\ \mathrm{ADDRS} \models \mathrm{addrs}\ (\mathrm{t}_{12} + 1) = \mathrm{address}_1$
$\mathit{INSTRCUCTION}_{a1}\ (\mathrm{addrs}, \mathrm{control}, \mathrm{test}, \mathrm{addrs}_a, \mathrm{addrs}_b)$
$\models \mathrm{address}\ (\mathrm{t}_{12} + 1) = \mathrm{address}_1 \Rightarrow \mathrm{VAL}_3\ (\mathrm{test}\ (\mathrm{t}_{12} + 1)) = 3 \wedge \mathrm{addrs}_a\ (\mathrm{t}_{12} + 1) = \mathrm{address}_2$
$\mathrm{HAND}_{\mathrm{knob}} \models \mathrm{VAL}_2\ (\mathrm{knob}\ \mathrm{t}_{12}) = 0 \wedge \mathrm{knob}\ \mathrm{t}_{12} = \mathrm{knob}\ (\mathrm{t}_{12} + 1)$
$\mathit{BRANCH}_1\ (\mathrm{test}, \mathrm{knob}, \mathrm{addrs}_a, \mathrm{nextaddrs})$
$\models \mathrm{nextaddrs}\ (\mathrm{t}_{12} + 1) = (\mathrm{VAL}_3\ (\mathrm{test}\ (\mathrm{t}_{12} + 1)) = 3$
$\quad \Rightarrow \mathrm{WORD}_5\ (\mathrm{VAL}_2\ (\mathrm{knob}\ (\mathrm{t}_{12} + 1)) + \mathrm{VAL}_5\ (\mathrm{addrs}_a\ (\mathrm{t}_{12} + 1)))$
$\mathrm{ADDRS} \models \mathrm{addrs}\ (\mathrm{t}_{12} + 2) = \mathrm{nextaddrs}\ (\mathrm{t}_{12} + 1)$

$\mathrm{START}\ \&\ \mathrm{HAND}_{\mathrm{button}}\ \&\ \mathit{INSTRUCTION}_{a0}\ \&\ \mathit{BRANCH}_0\ \&\ \mathrm{ADDRS}\ \&\ \mathit{INSTRUCTION}_{a1}\ \&$
$\mathrm{HAND}_{\mathrm{knob}}\ \&\ \mathit{BRANCH}_1 \models \mathrm{addrs}\ (\mathrm{t}_{12} + 2) = \mathrm{address}_2$

$\mathrm{START}\ \&\ \mathrm{HAND}_{\mathrm{button}}\ \&\ \mathit{INSTRUCTION}_{a0}\ \&\ \mathit{BRANCH}_0\ \&\ \mathrm{ADDRS}\ \&\ \mathit{INSTRUCTION}_{a1}\ \&$
$\mathrm{HAND}_{\mathrm{knob}}\ \&\ \mathit{BRANCH}_1 \models \mathrm{addrs}\ (\mathrm{t}_{12} + 2) = \mathrm{address}_2$
$\mathit{INSTRCUCTION}_{c2} \models \mathrm{address}\ (\mathrm{t}_{12} + 2) = \mathrm{address}_2 \Rightarrow \mathrm{control}\ (\mathrm{t}_{12} + 2) = \mathrm{control}_2$
$\mathit{DECODE}_2\ (\mathrm{control}, \mathrm{write}, \mathrm{wacc}, \mathrm{rsw}, \mathrm{wpc})$
$\models \mathrm{control}\ (\mathrm{t}_{12} + 2) = \mathrm{control}_2 \Rightarrow$
$\quad \mathrm{rsw}\ (\mathrm{t}_{12} + 2) = \mathrm{T} \wedge \mathrm{wpc}\ (\mathrm{t}_{12} + 2) = \mathrm{T} \wedge \mathrm{write}\ (\mathrm{t}_{12} + 2) = \mathrm{F} \wedge \mathrm{wacc}\ (\mathrm{t}_{12} + 2) = \mathrm{F}$

$\mathrm{START}\ \&\ \mathrm{HAND}_{\mathrm{button}}\ \&\ \mathit{INSTRUCTION}_{a0}\ \&\ \mathit{BRANCH}_0\ \&\ \mathrm{ADDRS}\ \&\ \mathit{INSTRUCTION}_{a1}\ \&$
$\mathrm{HAND}_{\mathrm{knob}}\ \&\ \mathit{BRANCH}_1\ \&\ \mathit{INSTRCUCTION}_{c2}\ \&\ \mathit{DECODE}_2$
$\models \mathrm{rsw}\ (\mathrm{t}_{12} + 2) = \mathrm{T} \wedge \mathrm{wpc}\ (\mathrm{t}_{12} + 2) = \mathrm{T} \wedge \mathrm{write}\ (\mathrm{t}_{12} + 2) = \mathrm{F} \wedge \mathrm{wacc}\ (\mathrm{t}_{12} + 2) = \mathrm{F}$

$\mathrm{START}\ \&\ \mathrm{HAND}_{\mathrm{button}}\ \&\ \mathit{INSTRUCTION}_{a0}\ \&\ \mathit{BRANCH}_0\ \&\ \mathrm{ADDRS}\ \&\ \mathit{INSTRUCTION}_{a1}\ \&$
$\mathrm{HAND}_{\mathrm{knob}}\ \&\ \mathit{BRANCH}_1 \models \mathrm{addrs}\ (\mathrm{t}_{12} + 2) = \mathrm{address}_2$
$\mathit{INSTRCUCTION}_{a2}$
$\models \mathrm{address}\ (\mathrm{t}_{12} + 2) = \mathrm{address}_2 \Rightarrow \mathrm{VAL}_3\ (\mathrm{test}\ (\mathrm{t}_{12} + 2)) = 0 \wedge \mathrm{addrs}_a\ (\mathrm{t}_{12} + 2) = \mathrm{address}_0$
$\mathit{BRANCH}_2\ (\mathit{test}, \mathit{addrs}_a, \mathit{nextaddrs})$
$\models \mathrm{VAL}_3\ (\mathrm{test}\ (\mathrm{t}_{12} + 2)) = 5 \Rightarrow \mathrm{nextaddrs}\ (\mathrm{t}_{12} + 2) = \mathrm{addrs}_a\ (\mathrm{t}_{12} + 2)$
$\mathrm{ADDRS} \models \mathrm{address}\ (\mathrm{t}_{12} + 3) = \mathrm{nextaddrs}\ (\mathrm{t}_{12} + 2)$

$\mathrm{START}\ \&\ \mathrm{HAND}_{\mathrm{button}}\ \&\ \mathit{INSTRUCTION}_{a0}\ \&\ \mathit{BRANCH}_0\ \&\ \mathrm{ADDRS}\ \&\ \mathit{INSTRUCTION}_{a1}\ \&$
$\mathrm{HAND}_{\mathrm{knob}}\ \&\ \mathit{BRANCH}_1\ \&\ \mathit{INSTRCUCTION}_{a2}\ \&\ \mathit{BRANCH}_2 \models \mathrm{address}\ (\mathrm{t}_{12} + 3) = \mathrm{address}_0$

$$\frac{\begin{array}{l}\text{START} \ \& \ \text{HAND}_{\text{button}} \ \& \ \mathit{INSTRUCTION}_{a0} \ \& \ \mathit{BRANCH}_0 \ \& \ \text{ADDRS} \ \& \ \mathit{INSTRUCTION}_{a1} \ \& \\ \text{HAND}_{\text{knob}} \ \& \ \mathit{BRANCH}_1 \ \& \ \mathit{INSTRCUCTION}_{a2} \ \& \ \mathit{BRANCH}_2 \models \text{address}\ (\text{t}_{12}+3) = \text{address}_0 \\ \mathit{INSTRCUCTION}_{c0} \models \text{address}\ (\text{t}_{12}+3) = \text{address}_0 \Rightarrow \text{control}\ (\text{t}_{12}+3) = \text{control}_0 \\ \mathit{DECODE}_0 \models \text{control}\ (\text{t}_{12}+3) = \text{control}_0 \Rightarrow \text{idle}\ (\text{t}_{12}+3) = \text{T}\end{array}}{\begin{array}{l}\text{START} \ \& \ \text{HAND}_{\text{button}} \ \& \ \mathit{INSTRUCTION}_{a0} \ \& \ \mathit{BRANCH}_0 \ \& \ \text{ADDRS} \ \& \ \mathit{INSTRUCTION}_{a1} \ \& \\ \text{HAND}_{\text{knob}} \ \& \ \mathit{BRANCH}_1 \ \& \ \mathit{INSTRCUCTION}_{a2} \ \& \ \mathit{BRANCH}_2 \ \& \ \mathit{INSTRCUCTION}_{c0} \ \& \\ \mathit{DECODE}_0 \models \text{idle}\ (\text{t}_{12}+3) = \text{T}\end{array}}$$

Using $\vdash \forall P.\ (P = \text{T} \Leftrightarrow P) \wedge (P = \text{F} \Leftrightarrow \neg P)$ and defining

$\models_{\text{def}} \mathit{INSTRUCTION}_{1,2,3} = \mathit{INSTRUCTION}_{a0} \ \& \ \mathit{INSTRUCTION}_{c0} \ \& \ \mathit{INSTRUCTION}_{a1} \ \& \ \mathit{INSTRUCTION}_{c1} \ \& \ \mathit{INSTRUCTION}_{a2} \ \& \ \mathit{INSTRUCTION}_{c2}$

$\models_{\text{def}} \mathit{BRANCH}_{0,1,2} = \mathit{BRANCH}_0 \ \& \ \mathit{BRANCH}_1 \ \& \ \mathit{BRANCH}_2$

$\models_{\text{def}} \mathit{DECODE}_{0,1,2} = \mathit{DECODE}_0 \ \& \ \mathit{DECODE}_1 \ \& \ \mathit{DECODE}_2$

we get:

$\text{START} \ \& \ \text{HAND}_{\text{button}} \ \& \ \text{HAND}_{\text{knob}} \ \& \ \mathit{INSTRUCTION}_{c0} \ \& \ \mathit{DECODE}_0$
$\models \text{idle t}_{12} \wedge \text{button t}_{12} \wedge \text{VAL}_2\ (\text{knob t}_{12}) = 0$

$\text{START} \ \& \ \text{HAND}_{\text{button}} \ \& \ \text{HAND}_{\text{knob}} \ \& \ \mathit{INSTRUCTION}_{c0} \ \& \ \mathit{DECODE}_0 \ \&$
$\text{ADDRS} \ \& \ \mathit{INSTRUCTION}_{0,1,2} \ \& \ \mathit{BRANCH}_{0,1,2} \ \& \ \mathit{DECODE}_{0,1,2}$
$\models \neg\text{write t}_{12} \wedge \neg\text{wacc t}_{12} \wedge$
$\neg\text{write}\ (\text{t}_{12}+1) \wedge \neg\text{wacc}\ (\text{t}_{12}+1) \wedge$
$\text{rsw}\ (\text{t}_{12}+2) \wedge \text{wpc}\ (\text{t}_{12}+2) \wedge \neg\text{write}\ (\text{t}_{12}+2) \wedge \neg\text{wacc}\ (\text{t}_{12}+2) \wedge$
$\text{idle}\ (\text{t}_{12}+3)$

5.3.3 Deriving Path$_2$

Let $\vdash_{\text{def}} \text{t}_{22} = \text{t}_{12} + 3$ then we get:

$$\frac{\begin{array}{l}\text{START} \ \& \ \text{HAND}_{\text{button}} \ \& \ \text{HAND}_{\text{knob}} \ \& \ \mathit{INSTRUCTION}_{c0} \ \& \ \mathit{DECODE}_0 \\ \models \text{idle t}_{12} \wedge \text{button t}_{12} \wedge \text{VAL}_2\ (\text{knob t}_{12}) = 0 \\ \text{START} \ \& \ \text{HAND}_{\text{button}} \ \& \ \text{HAND}_{\text{knob}} \ \& \ \mathit{INSTRUCTION}_{c0} \ \& \ \mathit{DECODE}_0 \ \& \\ \mathit{INSTRUCTION}_{0,1,2} \ \& \ \mathit{BRANCH}_{0,1,2} \ \& \ \mathit{DECODE}_{0,1,2} \ \& \ \text{ADDRS} \ \& \\ \text{HAND}_{\text{sw}} \ \& \ \text{GATE}_{\text{sw}} \ \& \ \text{BUS} \ \& \ \text{REG}_{\text{pc}} \ \& \ \text{REG}_{\text{acc}} \ \& \ \text{MEM} \\ \models \text{memory t}_{22} = \text{memoryt}_{12} \wedge \text{pc t}_{22} = \text{CUT}_{16\_13}\ (\text{swetches t}_{12}) \wedge \text{acc t}_{22} = \text{acc t}_{12} \wedge \text{idle t}_{22} = \text{T}\end{array}}{\begin{array}{l}\mathit{INSTRUCTION}_{0,1,2} \ \& \ \mathit{BRANCH}_{0,1,2} \ \& \ \mathit{DECODE}_{0,1,2} \ \& \ \text{ADDRS} \ \& \ \text{HAND}_{\text{sw}} \ \& \ \text{GATE}_{\text{sw}} \ \& \\ \text{BUS} \ \& \ \text{REG}_{\text{pc}} \ \& \ \text{REG}_{\text{acc}} \ \& \ \text{MEM} \models \text{path}_2\end{array}}$$

Finally let us give detail description with details about ports for our result:

$\mathit{INSTRUCTION}_{0,1,2}\ (\text{addrs, control, test, addrs}_a, \text{addrs}_b)\ \&$
$\mathit{DECODE}_{0,1,2}\ (\text{control}, \cdots, \text{idle, write, wacc, rsw, wpc}, \cdots)\ \&$
$\mathit{BRANCH}_{0,1,2}\ (\text{test, buton, knob, addrs}_a, \text{addrs}_b, \text{nextaddrs})\ \&$
$\text{ADDRS}\ (\text{nextaddrs, addrs})\ \&$
$\text{HAND}_{\text{sw}}\ \&$
$\text{GATE}_{\text{sw}}\ (\text{switches, rsw, o}_g)\ \&$
$\text{BUS}\ (\cdots, \text{o}_g, \cdots, \text{o}_b)\ \&$
$\text{REG}_{\text{pc}}\ (\text{o}_b, \text{wpc, pc})\ \&$
$\text{REG}_{\text{acc}}\ (i_{acc}, \text{wacc, acc})\ \&$
$\text{MEM}\ (\text{memory}, a, d, \mathit{read}, \text{write}, o)$
$\models \text{path}_2$

5.4 General Analysis of Derivation

It is not possible in a short paper to give all details about the derivation of whole computer. We will analyse, however, through describing the main steps in the derivation of path_{12} to give general picture to understand whole derivation of the computer.

5.4.1 Deriving Data Part

For deriving data part main work is to compose basic device along data flow.

• **Introducing Register and Gate**
Device is delivery of signal (data) flow. Signal is usually delivered through a series of devices. For bus-based signal delivery the basic structure of implementation is 'output→bus→input', eg the $\mathrm{Theorem}_1$ describes signal flow 'gate→bus→register' in single bus structure. Additional registers and gates will be introduced in below cases: 1. device has more than one input and bus is not enough to deliver all signals. 2. common signal is used many times. 3. time match is needed. 4. clearer design is prefered. However all these are embodied (hided) in the derivation process.

For the device MEM $(memory, a, d, r, w, o_m)$ after selecting bus to deliver signal d, $\mathrm{REG}_{\mathrm{mar}}$ is introduced for signal a. Similarly $\mathrm{REG}_{\mathrm{arg}}$ for signal i_{a1} of ALU. The o_m of MEM is used more than one time so $\mathrm{REG}_{\mathrm{ir}}$ is introduced. For time-match of ALU a delay, BUF, is introduced.

• **Connecting Ports**
For function composition of specification when output of a function is input of another function a signal delivery will be needed between the output and the input. There are also signal delivery between input of function and signal feeder, eg in FETCH_{13} (memory t_{12}) (pc t_{12}) from 'pc t_{12}' to the second input of FETCH_{13} through $\mathrm{GATE}_{\mathrm{pc}}$, BUS and $\mathrm{REG}_{\mathrm{mar}}$. And for the signal delivery some selected connections among devices will lead to set common ports, like as showing of the $\mathrm{Theorem}_1$.

• **Main Steps for Deriving Data Part of Path_{12}**
After derivation of path_2 for deriving path_{12} $\mathrm{REG}_{\mathrm{mar}}, \mathrm{GATE}_{\mathrm{mar}}, \mathrm{REG}_{\mathrm{ir}}, \mathrm{GATE}_{\mathrm{ir}}, \mathrm{ALU}, \mathrm{BUF}$, $\mathrm{GATE}_{\mathrm{buf}}, \mathrm{REG}_{\mathrm{arg}}$ and $\mathrm{GATE}_{\mathrm{arg}}$ are introduced. Meanwhile relevant ports are connected in the derivation for signal deliveries.

Repeating to use the $\mathrm{Theorem}_1$ below signal deliveries can be obtained:
(1). from pc, the output of $\mathrm{REG}_{\mathrm{pc}}$, to $\mathrm{o}_{\mathrm{mar}}$, the output of $\mathrm{REG}_{\mathrm{mar}}$
(2). from $\mathrm{o}_{\mathrm{buf}}$, the output of BUF, to pc
(3). from o_{ir}, the output of $\mathrm{REG}_{\mathrm{ir}}$, to $\mathrm{o}_{\mathrm{mar}}$
(4). from acc, the output of $\mathrm{REG}_{\mathrm{acc}}$, to $\mathrm{o}_{\mathrm{arg}}$, output of $\mathrm{REG}_{\mathrm{arg}}$
(5). from $\mathrm{o}_{\mathrm{buf}}$ to acc
Usinfg similar inferences below signal deliveries can be obtained:
(6). from memory, $\mathrm{o}_{\mathrm{mar}}$, the input of MEM, to o_{ir}
(7). from pc to $\mathrm{o}_{\mathrm{buf}}$
(8). from memory, $\mathrm{o}_{\mathrm{mar}}$ to $\mathrm{o}_{\mathrm{buf}}$

5.4.2 Deriving Control Part

For deriving control part main work is to construct micro-instruction, devices decode and branch step by step along control flow. In fact it is to introduce concrete specification instead of abstract specification structure for devices: instruction, decode and branch.

• Introducing Device and Connectting Ports
Unfinished constructions in the derivation of data part, whose implementations are variables, and whose specifications are about control signals of gate, register, mem and alu are taken as the goals of the derivation of the control part. In the deriving control part *INSTRUCTION, DECODE, BRANCH*, ADDRS and START are introduced and their ports are connneted. *INSTRUCTION, DECODE, BRANCH* are introduced step by step following the derivation process so they keep as variables until the whole implementation is derived. The process of deriving *INSTRUCTION, DECODE* is too the process of deriving micro-program. START is introduced for various possibilities of times and pathes to start running micro-program.

• Deriving Microprogram
For deriving micro-program main cases are as following: (1). Based on a single-bus structure at any time only one signal which is not $FLOAT_{16}$ can appear in the bus. This leads to form a linear sequence of micro-instructions. (2). Following the order of composition of function the order of the linear sequence of micro-instructions is formed. (3). But for arguments of function, whose correspoding micro-program can be parallel in essence, so when the micro-program is arranged as sequence form the order of the micro-program can arbitrarily be selected. (4). For predicates which connected by '$\wedge$' their correspoding micro-progarms can be parallel so when the micro-programs are arranged as sequence form whose order can arbitrarily selected. But for getting shorter program some common part of microprograms are put the beginning or end of programs. (5). The 'let-in' part of the specification will lead to form a common part of micro-program, which is put the beginning of the microprogram. (6). The 'if-then-else' part of the specification will lead to form branch-structure of the microprogram.

• Main Steps about Deriving Control Part of $\mathbf{Path_{12}}$
In the derivation of $path_{12}$ for '$FETCH_{13}$ (memory t_{12}) (pc t_{12})', a sequence of micro-instructions about '$rpc, wmar \rightarrow read, wir$' is derived. Similarly for
'acc $t_{212} = SUB_{16}$ (acc t_{112}) ($FETCH_{13}$ (memory t_{112}) $addr_{12}$)' it is '$racc, warg \rightarrow rir, wmar \rightarrow read, sub \rightarrow rbuf, wacc$'. For 'pc $t_{212} = INC_{13}$ (pc t_{112})' it is '$rpc, inc \rightarrow rbuf, wpc$'
$BRANCH_4$ (test, ir, $addrs_a$, nextaddrs)
$\models \forall t.\text{nextaddrs}\ t = (\text{test}\ t = 4 \Rightarrow WORD_5\ (VAL_3\ (OPCODE\ (ir\ t)) + VAL_5\ (addrs_a\ t)))$
is introduced for the branch-structure.

5.4.3 Deriving Path and Merging Pathes

When data and control parts of a path have been derived using $Rule_3$ the path can be derived. When all pathes have been derived their constructions are merged to derive whole implementation of the computer. The device $HAND_{sw}$ is a part of the implementation. But as tranditional view it is omitted.

5.5 Deriving Whole Implementation

Merging the 15 constructions of the derived pathes whole implementation of the computer is obtained:

Data Part:
$\mathrm{GATE_{sw}}$ (swtiches, rsw, g_{sw}) &
$\mathrm{GATE_{pc}}$ (pc, rpc, g_{pc}) &
$\mathrm{GATE_{acc}}$ (acc, racc, g_{acc}) &
$\mathrm{GATE_{ir}}$ (ir, rir, g_{ir}) &
$\mathrm{GATE_{buf}}$ (buf, rbuf, g_{buf}) &
BUS ($g_{sw}, g_{pc}, g_{acc}, g_{ir}, g_{buf}, o_m, o_b$) &
$\mathrm{REG_{pc}}$ (o_b, wpc, pc) &
$\mathrm{REG_{acc}}$ (o_b, wacc, acc) &
$\mathrm{REG_{mar}}$ (o_b, wmar, mar) &
$\mathrm{REG_{ir}}$ (o_b, wir, ir) &
$\mathrm{REG_{arg}}$ (o_b, warg, arg) &
BUF (o_a, buf) &
MEM (memory, mar, o_b, read, write, o_m) &
ALU (arg, o_b, inc, add, sub, o_a) &
Control Part:
ADDRS (nextaddrs, addrs) &
DECODE (control, rsw, rpc, racc, rir, rbuf, wpc, wacc, wmar, wir, warg, read, write, inc, add, sub, idle) &
BRANCH (test, button, knob, ir, acc, $\mathrm{addrs_a}$, $\mathrm{addrs_b}$, nextaddrs) &
INSTRUCTION (addrs, control, test, $\mathrm{addrs_a}$, $\mathrm{addrs_b}$)[2]

5.6 Deriving Other Implementations

For same goal selecting different axioms and passing different derivings different implementations can be derived. In essence verification is on designed implementation but synthesis is to design using formal method. So synthesis has, in essence, ability to show various different design ways. For Mike Gordon's computer, another possible implementation, for example, is that 1. Do not introduce $\mathrm{REG_{arg}}$, $\mathrm{REG_{mar}}$ but only $\mathrm{REG_{ir}}$. 2. Directly connect the output of $\mathrm{REG_{acc}}$ and the i_{a1} (the first input of ALU), the output of $\mathrm{REG_{ir}}$ and the i_{a2} (the second input of ALU), the output of $\mathrm{REG_{acc}}$ and 'd' (the second input of MEM). 3. Use multi-bus-gate structure (gates for every input): (1). $\mathrm{BUS_1}$: inputs: the swetches, the output of ALU and the output of $\mathrm{REG_{ir}}$, output: the input of $\mathrm{REG_{acc}}$ and the input of $\mathrm{REG_{pc}}$. (2). $\mathrm{BUS_2}$: input: the output of $\mathrm{REG_{pc}}$ and the output of $\mathrm{REG_{ir}}$, output: 'a' (the first input of MEM). The implementation has less registers, more buses, shorter micro-program and faster speed. We will give more detail in other paper about the problem.

6 Conclusions and Further Work

On the synthesis logic from given formal specification at bl implementation at rtl can be derived. For the derivation main steps are to compose basic devices along data flow and to construct micro-program and auxiliary devices, decode and branch, along control flow.

[2]The details about INSTRUCTION, DECODE and BRANCH will be given in appendix1

The paper show us a way: based on strong general logics a succinct synthesis logic can be built. The logic catches basic relation between implementation and specification. On the logic we find a specification-derived design methodology, proof-based synthesis: implementation can be derived step by step from formal specification using formal proof. We using the example of synthesis of Mike Gordon's computer to show that it does be realistic to design complex hardware using formal method. However we think the more important is not to find a new logic but is how to use logic for practical design problem. In essence it is problem: 'how to proof'. A further problem is to find more general specification scheme to catch broader proof goal. In the [5] we will discuss that on an abstract specification scheme how to derive a class of computers.

Slogan, 'deriving as design' and 'derivation as implementation' brings us both benefit and trouble: more correctness gurantee and more tedious proof. There is a vast gap between formal method and informal design and there is not a simple bridge for the gap. But various methodologies maybe play some roles for the gap. Problem is often reduced to find better methodology. Along the direction [5] presents some basic relations about structures among specification, proof and implementation.

When the first computer in the world was designed people thought that there was only a step between user's idea and machine implementation: machine code programming. But with the lapse of time people understand that there is a vast gap between them. The gap is just a world of computer science: computer scientists and their causes are contained in the world.

Acknowledgement
Li-Guo Wang sincerely thank to Mr Kayhan Imre, Dr Zhaohui Luo, Mr Jonathan Puddicombe and Mr Sandy Robertson for beneficial discussions and warmhearted help in logic, hardware and English.

References

[1] M.P.Fourman,R.L.Harris, Lambda-Logic And Mathematics Behind Design Automation, 26th ACD/IEEE Design Automation Conference, 1988.

[2] Gordon, M. Proving a Computer Correct using the LCF-LSM Hardware verification System, Report No. 42, Computer laboratory, Cambridge University, 1983.

[3] Gordon, M. HOL: A Proof Generating System for Higher-Order Logic, University of Cambridge, Computer Laboratory, Tech Reprot No. 103, 1987.

[4] J. Joyce, G. Birtwistle and M. Gordon, Proving a Computer Correct in Higher Order Logic, Report No. 100, Computer laboratory, Cambridge University, 1986.

[5] Li-Guo Wang, Formal Derivation of A Class of Computers, Draft, March 1991.

Appendix1: Microprogram, Decode and Branch

General form are:
$\mathrm{INSTRUCTION}_{ci}$ (addrs, control, test, addrs_a, addrs_b)
$\models \forall t.\ \mathrm{addrs}\ t = \mathrm{address}_i \Rightarrow \mathrm{control}\ t = \mathrm{control}_i$

$\mathrm{INSTRUCTION}_{ai}\,(\mathrm{addrs}, \mathrm{control}, \mathrm{test}, \mathrm{addrs}_a, \mathrm{addrs}_b)$
$\models \forall t.\ \mathrm{addrs}\ t = \mathrm{address}_i \Rightarrow \mathrm{test}\ t = n_i \wedge \mathrm{addrs}_a\ t = \mathrm{addressa}_i \wedge \mathrm{addrs}_b\ t = \mathrm{addressb}_i$
$\mathrm{DECODE}_i\,(\mathrm{control}, \mathrm{code}_1, \mathrm{code}_2, \cdots, \mathrm{code}_m)$
$\models \forall t.\ \mathrm{control}\ t = \mathrm{control}_i \Rightarrow \mathrm{code}_1\ t = \mathrm{ft} \wedge \mathrm{code}_2\ t = \mathrm{ft} \wedge \cdots \wedge \mathrm{code}_m\ t = \mathrm{ft}$

If we select the i_{th} bit of $\mathrm{control}_i$ from 1 to 16 corresponds to rsw, rpc, racc, rir, rbuf, wpc, wacc, wmar, wir, warg, read, write, inc, add, sub, idle respectively and assign concrete bits to address, addressa and addressb then we have:

address	control	n	addressa	addressb
00000	0000000000000001	001	00000	00001
00001	0000000000000000	011	00010	00000
00010	1000010000000000	000	00000	00000
00011	1000001000000000	000	00000	00000
00100	0100000100000000	000	00000	00000
00101	0000000000000000	001	00110	00000
00110	0100000100000000	000	01000	00000
00111	0010000000010000	000	00000	00000
01000	0000000010100000	000	01001	00000
01001	0000000000000000	100	01010	00000
01010	0000000000000000	000	00000	00000
01011	0001010000000000	000	00101	00000
01100	0000000000000000	010	10001	01011
01101	0010000001000000	000	10011	00000
01110	0010000001000000	000	10110	00000
01111	0001000100000000	000	11000	00000
10000	0001000100000000	000	11001	00000
10001	0100000000001000	000	10010	00000
10010	0000110000000000	000	00101	00000
10011	0001000100000000	000	10100	00000
10100	0000000000100100	000	10101	00000
10101	0000101000000000	000	10001	00000
10110	0001000100000000	000	10111	00000
10111	0000000000100010	000	10101	00000
11000	0000001000100000	000	10001	00000
11001	0010000000010000	000	10001	00000

Note: because of START the micro-instractions from address 11010 to 11111 are not necessary.

$\mathrm{BRANCH}\,(\mathrm{test}, \mathrm{button}, \mathrm{knob}, \mathrm{ir}, \mathrm{acc}, \mathrm{addrs}_a, \mathrm{addrs}_b, \mathrm{nextaddrs})$
$\forall t.\ \mathrm{nextaddrs}\ t =$
$((\mathrm{VAL}_3\,(\mathrm{test}\ t) = 1) \wedge (\mathrm{button}\ t) \Rightarrow \mathrm{addrs}_b\ t\ |$
$((\mathrm{VAL}_3\,(\mathrm{test}\ t) = 2) \wedge (\mathrm{VAL}_{16}\,(\mathrm{acc}\ t) = 0) \Rightarrow \mathrm{addrs}_b\ t\ |$
$((\mathrm{VAL}_3\,(\mathrm{test}\ t) = 3) \Rightarrow \mathrm{WORD}_5\,(\mathrm{VAL}_2\,(\mathrm{knob}\ t) + \mathrm{VAL}_5\,(\mathrm{addrs}_a\ t)\ |$
$((\mathrm{VAL}_3\,(\mathrm{test}\ t) = 4) \Rightarrow \mathrm{WORD}_5\,(\mathrm{VAL}_3\,(\mathrm{OPCODE}\,(\mathrm{ir}\ t)) + \mathrm{VAL}_5\,(\mathrm{addrs}_a\ t))\ |\ (\mathrm{addrs}_a\ t)))))$