Proving a computer correct with the LCF_LSM hardware verification system

Mike Gordon

September 1983
© 1983 Mike Gordon

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986
Proving a Computer Correct
with the LCF_LSM Hardware Verification System

Mike Gordon
Computer Laboratory
Corn Exchange Street
Cambridge CB2 3QG

Abstract

A machine-generated correctness proof of a simple computer is described.

At the machine code level the computer has a memory and two registers: a 13-bit program counter and a 16-bit accumulator. There are 8 machine instructions: halt, unconditional jump, jump when the accumulator contains 0, add contents of a memory location to accumulator, subtract contents of a location from accumulator, load accumulator from memory, store contents of accumulator in memory, and skip. The machine can be interrupted by pushing a button on its front panel.

The implementation which we prove correct has 6 data registers, an ALU, a memory, and a microcode controller. This controller consists of a ROM holding 26 30-bit microinstructions, a microprogram counter, and some combinational microinstruction decode logic.

Formal specifications of the target and host machines are given, and we describe the main steps in proving that the host correctly fetches, decodes and executes machine instructions.

The utility of LCF_LSM for general manipulation is illustrated in two appendices. In Appendix 1 we show how to code a microassembler. In Appendix 2 we use the LCF_LSM inference rules to design a hard-wired controller equivalent to the original microcoded one.

N.B. This report should be read in conjunction with LCF_LSM: A system for specifying and verifying hardware. University of Cambridge computer laboratory technical report Number 41.
Proving a Computer Correct
with the LCF-LSM Hardware Verification System

Introduction

The example computer used in this report is taken from [Gordon1]; its specification and verification are done using the LCF-LSM system [Gordon2]. Familiarity with LCF-LSM is assumed.

As the proof of correctness is quite long, we will not give a complete transcript of it. We will, however, outline the main steps. The omitted details are mostly pure LCF manipulations. We hope eventually to design and implement derived inference rules which will perform these LCF inferences automatically.

Our aim in presenting the proof is to give an idea of what is involved in verifying a non-trivial system. It is not expected that the reader will follow all the details, but merely that he will get a feel for their rough shape. In particular we hope to demonstrates that:

1. Non-trivial hardware can be proved correct using existing techniques.

2. Machine assistance is essential for such proofs.

The entire specification and verification described here took several months, but this includes some extending and debugging of LCF-LSM (necessary, as this was our first big example). I estimate that it would take me two to four weeks to do another similar exercise now. The complete proof requires several hours CPU time on a 2 megabyte Vax750. I found it necessary to prove some of the bigger lemmas (e.g. DERIVED_HOST_EQN below) in batch mode overnight. These times seem to me to be reasonable in the context of the normal timescales associated with the design and implementation of hardware systems.

If a small change were made to the computer (e.g. adding a separate incrementer for the program counter, or changing a few microinstructions), then the proof could be redone in a day or so. The main overhead of setting up the various theories, and creating ML programs to do the necessary inferences, need only be done once. The correctness of small changes can be verified by editing these theories and programs, and then redoing the entire proof as a batch job.

To illustrate how the LCF-LSM system can be used for other things besides formal proof of correctness, we describe in appendices how a
microassembler can be expressed in ML, and how a hard-wired controller for the computer can be systematically derived. We believe that LCF_LSM provides a good programming environment for developing many kinds of CAD/CAM tools. Further evidence for this is the use of ML for mask-level graphical design in the Sticks & Stones system [Cardelli], this approach is entirely consistent with the framework used here; Cardelli's ideas could be nicely integrated into LCF_LSM.

Specification of The Target Machine

The front panel of the computer looks like:

![Diagram of computer front panel](image)

This computer has two registers: the program counter PC which is 13 bits wide, and the accumulator ACC which is 16 bits wide. It has a random access memory with 13-bit addresses, each of which points to a 16-bit word (thus the memory holds 8192 16-bit words).

On the front panel there is a four position knob which determines what happens when the button on the right of the panel is pressed. There are three sets of lights: thirteen PC display lights which show the contents of the program counter; sixteen ACC display lights which show the contents of the accumulator, and the idle light which is on when the computer is idling (i.e. not executing a program). There are also sixteen
two-position switches which are used for inputting data.

We shall refer to the four positions of the knob as 0, 1, 2 and 3. The effect of pushing the button when the idle light is on is:

\( knob = 0 \) The word determined by the state of the thirteen rightmost switches is loaded into the program counter.

\( knob = 1 \) The word determined by the state of the sixteen switches is loaded into the accumulator.

\( knob = 2 \) The contents of the accumulator is stored in the memory at the location held in the program counter.

\( knob = 3 \) The program stored in memory is executed starting at the location in the program counter. The idle light will go off, and stay off until execution stops; this happens either when a halt instruction is reached, or when an interrupt is generated by pushing the button.

Each instruction has a three bit opcode part \( op \) and a thirteen bit address part \( addr \). The format is:

\[
[ \ldots op \ldots ] \ldots addr \ldots \]

The eight instructions are:

\[
| 0 | 0 | 0 | \ldots addr \ldots | \text{HALT} \quad \text{Stop execution}
| 0 | 0 | 1 | \ldots addr \ldots | \text{JMP} L \quad \text{Jump to addr}
| 0 | 1 | 0 | \ldots addr \ldots | \text{JZ} L \quad \text{Jump to addr if } ACC=0
| 0 | 1 | 1 | \ldots addr \ldots | \text{ADD} L \quad \text{Add contents of addr to } ACC
| 1 | 0 | 0 | \ldots addr \ldots | \text{SUB} L \quad \text{Subtract contents of addr from } addr
| 1 | 0 | 1 | \ldots addr \ldots | \text{LD} L \quad \text{Load contents of addr into } ACC
| 1 | 1 | 0 | \ldots addr \ldots | \text{ST} L \quad \text{Store contents of } ACC \text{ in addr}
| 1 | 1 | 1 | \ldots addr \ldots | \text{SKIP} \quad \text{Skip to next instruction}

In order to formally specify the computer in LSM it is convenient to define a few auxiliary constants:

- \( PAD16 \): Pads a 13-bit word with 3 Os at left
- \( CUT16 \): Extracts 13 rightmost bits
- \( OPCODE \): Extracts 3 leftmost bits
- \( INC16 \): Adds 1 to a 16-bit word
- \( ADD16 \): Addition on 16-bit words
- \( SUB16 \): Subtraction on 16-bit words
- \( I5ZERO16 \): Test for 0 on 16-bit words

The following axioms are needed:
We can now specify the computer using two constants:

\begin{align*}
&\text{NEXT} \quad \text{COMPUTER} \\
&\text{mem}_{13} \cdot 16 \cdot \text{word}_{13} \cdot 16 \cdot \text{word}_{16} \rightarrow \text{dev} \\
&\text{mem}_{13} \cdot 16 \cdot \text{word}_{13} \cdot 16 \cdot \text{word}_{16} \rightarrow \text{dev} \\
\end{align*}

The function \text{NEXT} defines a single step of the machine, and hence specifies the semantics of the machine instructions. \text{NEXT}(m, w_1, w_2) gives the state after executing the instruction stored in \(m\) at address \(w_1\) when the accumulator holds \(w_2\), and is defined by the following axiom:

\[
\text{NEXT}(m, w_1, w_2) = \begin{cases} 
\text{let } \text{op} = \text{VAL}3(\text{OPCODE}(\text{FETCH}13 m w_1)) \text{ in} \\
\text{let } \text{addr} = \text{CUT}16.13(\text{FETCH}13 m w_1) \text{ in} \\
(\text{nop} = 0 \rightarrow (m, w_1, w_2, T)) \ |
\text{op} = 1 \rightarrow (m, \text{addr}, w_2, F) \ |
\text{op} = 2 \rightarrow (\text{VAL}3 w_2 = 0 \rightarrow (m, \text{addr}, w_2, F) \ |
(m, \text{INC}13 w_1, w_2, F)) \ |
\text{op} = 3 \rightarrow (m, \text{INC}13 w_1, \text{ADD}16 w_2 (\text{FETCH}13 m \text{addr}), F) \ |
\text{op} = 4 \rightarrow (m, \text{INC}13 w_1, \text{SUB}16 w_2 (\text{FETCH}13 m \text{addr}), F) \ |
\text{op} = 5 \rightarrow (m, \text{INC}13 w_1, \text{FETCH}13 m \text{addr}, F) \ |
\text{op} = 6 \rightarrow (\text{STORE}13 \text{addr} w_2 m, \text{INC}13 w_1, w_2, F) \ |
(m, \text{INC}13 w_1, w_2, F) \ |
\end{cases}
\]

\text{COMPUTER}(m, w_1, w_2, t) is the behaviour of the computer when the memory is \(m\), the program counter is \(w_1\), the accumulator is \(w_2\) and the run/idle status is \(t\) (\(t=T\) means the computer is idling; \(t=F\) means it is executing). The axiom defining this behaviour is:

\[
\text{COMPUTER}(m, w_1, w_2, t) = \begin{cases} 
\text{dev} \{ \text{knob}, \text{button}, \text{switches}, \text{pc}, \text{acc}, \text{idle} \}. \\
\{ \text{pc}=w_1, \text{acc}=w_2, \text{idle}=t \}; \\
\text{COMPUTER}(t) \rightarrow \\
(\text{button} \rightarrow ( (\text{VAL}2 \text{ knob} = 0) \rightarrow (m, \text{CUT}16.13 \text{ switches}, w_2, T) \ |
(\text{VAL}2 \text{ knob} = 1) \rightarrow (m, w_1, \text{switches}, T) \ |
(\text{VAL}2 \text{ knob} = 2) \rightarrow (\text{STORE}13 w_1 w_2 m, w_1, w_2, T) \ |
(m, w_1, w_2, F)) \ |
(m, w_1, w_2, T)) \ |
(\text{button} \rightarrow (m, w_1, w_2, T) \ |
\text{NEXT}(m, w_1, w_2))) \end{cases}
\]

The two axioms above are a complete specification of the computer. Note how concise they are.

**Specification of the Host Machine**

Next we describe an implementation. We will refer to this as the host machine and the machine it implements (namely the machine specified by \text{NEXT} and \text{COMPUTER} above) as the target machine. The host machine has a number of registers in addition to the program counter and accumulator of the target machine. The instruction currently being executed is held in the instruction register \(IR\); addresses being looked-up in the memory are held in the memory address register \(MAR\); arguments to the arithmetic and logic unit (the \(ALU\)) are held in the \(ARG\) register, and the results of the \(ALU\) are held in the buffer register \(BUF\).
The fetch-decode-execute cycle is driven by a microcoded control unit. The microcode is stored in a read-only memory (the ROM) which can hold 32 microinstructions, each 30 bits wide. The actual microcode is represented by a constant MICROCODE of LSM type mem5_30.

The architecture of the host is described by the following diagram:
This diagram shows both the control and data parts of the host; it is represented in LSM by a constant:

$$\text{HOST : word5\#mem13\_16\#word13\#word16\#word16\#word16\#word16\#word16\#word16\#word16\#word16\#word16\#dev}$$

Where, in $\text{HOST}(w,m,w0,w1,w2,w3,w4,w5)$: $w$ is a 5-bit word representing the address of the next microinstruction to be executed, $m$ is the memory, $w0$ is the contents of the MAR register, $w1$ is the contents of the PC register, $w2$ is the contents of the ACC register, $w3$ is the contents of the IR register, $w4$ is the contents of the ARG register and $w5$ is the contents of the BUF register.

The correctness of the host is represented in LSM by the formula:

$$\text{<m w0 w1 w2 w3 w4 w5 \&. COMPUTER(m,w1,w2,t) \&\& until ready do HOST(WORDS5(i->0|5),m,w0,w1,w2,w3,w4,w5)}$$

The start address in ROM of the microcode for controlling the idling computer is 0; the start address for the fetch-decode-execute cycle is 5. The host has a done-line called ready which signals the end of a sequence of microcycles which implement a single target macrocycle.

We hierarchically specify HOST by introducing constants:

$$\text{CONTROL : mem5\_30\#word5 -> dev}$$
$$\text{DATA : mem13\_16\#word13\#word16\#word16\#word16\#word16\#word16 -> dev}$$

And then an axiom:

$$\text{HOST}(w,m,w0,w1,w2,w3,w4,w5) \equiv \ \{ \{ \text{CONTROL(MICROCODE,w)} \mid \text{DATA}(m,w0,w1,w2,w3,w4,w5) \} \}$$
$$\text{hide[rsu,unar,mementl,wpc,rpc,wc,c,rac,wir,rir,auarg,aluentl,rbuf,ir]}$$

Thus the host is obtained by connecting together the control part (with the actual microcode) and the data part and then hiding the control lines. We specify these parts structurally by:

$$\text{CONTROL(r,w) \equiv \{ \{ \text{ROM r} \mid \text{MPC w} \mid \text{DECODE} \} \text{ hide[mpc,rom,nextaddress]} \}$$

and

$$\text{DATA}(m,w0,w1,w2,w3,w4,w5) \equiv \ \{ \{ \text{MEM m} \mid \text{MAR w0} \mid \text{PC w1} \mid \text{ACC w2} \mid \text{IR w3} \mid \text{ARG w4} \mid \text{BUF w5} \}
$$
$$\text{G0} \mid \text{G1} \mid \text{G2} \mid \text{G3} \mid \text{G4} \mid \text{ALU} \mid \text{BUS} \}$$
$$\text{hide[mem,mar,arg,buf,g0,g1,g2,g3,g4,alu,bus]}$$

Where for the component devices we have constants:

$$\text{ROM : mem5\_30 -> dev}$$
$$\text{MPC : word5 -> dev}$$
$$\text{DECODE : dev}$$
$$\text{MEM : mem13\_16 -> dev}$$
$$\text{MAR : word13 -> dev}$$
$$\text{PC : word13 -> dev}$$
ACC : word16 -> dev
IR : word16 -> dev
ARG : word16 -> dev
BUF : word16 -> dev
G0 : dev
g1 : dev
g2 : dev
g3 : dev
g4 : dev
ALU : dev
BUS : dev

We then specify axioms defining the behaviour of the memories, registers, gates, ALU, bus, and control logic. We have arbitrarily chosen to take these devices as primitive, we could have chosen to further decompose them structurally, and then proven that these structures have the desired behaviour. The only complicated specification is the one for the purely combinational microinstruction decode logic - i.e. the device DECODE. We defer giving this for a while. The read-only memory (for holding the microcode) and microprogram counter are specified by:

\[
\begin{align*}
\text{ROM } r & = \text{dev}[\text{mpc}.\text{rom}].\{\text{rom=FETCH5 } r \text{ mpc}\}.\text{ROM } r \\
\text{MPC } w & = \text{dev}[\text{nestaddress}.\text{mpc}].\{\text{mpc=w}\}.\text{MPC nestaddress}
\end{align*}
\]

To model the bus we use tri-state values, thus the LSM type of values on the bus is \textit{tri_word16} not \textit{word16}. The LSM functions

\[
\begin{align*}
\text{MK_TRI16} & : \text{word16} \rightarrow \text{tri_word16} \\
\text{DEST_TRI16} & : \text{tri_word16} \rightarrow \text{word16}
\end{align*}
\]

convert from 16-bit words to 16-bit tri-state words, and vice versa, respectively. These functions satisfy the (built-in) axiom:

\[
!w:\text{word16}. \text{DEST_TRI16(MK_TRI16 w)} = w
\]

This axiom is invoked by \textit{TRI_RULE} and \textit{TRI_FCONV} as described in Appendix 3 of [Gordon2].

LSM has a special built-in tri-state value \textit{FLOAT16} to represent the floating (or high impedance) state. There is also a built-in in infix:

\[
U16 : \text{tri_word16} # \text{tri_word16} \rightarrow \text{tri_word16}
\]

for representing the value which results when several values are simultaneously put on a bus. Thus if two tri-state words, \textit{w1} and \textit{w2} say, are put on the bus at the same time then the resulting value is \textit{w1 U16 w2}.

We have the built-in axiom:

\[
!w:\text{tri_word16}. \text{FLOAT16 U16 w} = w \land w \text{ U16 FLOAT16} = w
\]

This axiom is invoked by \textit{U_RULE} and \textit{U_FCONV} as described in Appendix 3 of [Gordon2]. It says that if only one non-floating value, \textit{w
say, is put on a bus, then the resulting value on the bus is $w$. We do not specify what happens if two or more different non-floating values are put on a bus (this should never happen; proving it doesn't is part of the work that has to be done in verifying correctness). In the definition of BUS below we choose to output not this tri-state value, but DEST_TRI16 of it instead. This saves us having to invoke DEST_TRI16 in all the devices that read from the bus.

We can now describe the behaviour of the main memory:

$$MEM \ m = \begin{cases} \text{dev(mar, bus, memctl, mem)}, & \text{mem}=2 \ \text{memctl} = 1 \Rightarrow \text{MK_TRI16(FETCH13 mar m)} \ | \ \text{FLOAT16}\},
\text{MEM}(\text{memctl} = 2 \Rightarrow \text{STORE13 mar bus m} \ | \ m) \end{cases}$$

Thus if the number denoted by the 2-bit word on line memctl is 1 then the contents of the location input on line mar is put on the line mem (which is connected to the bus), otherwise the line is floated. The new state of MEM is identical to the old state unless the value on memctl denotes 2 (i.e. is #10). If #10 is input on line memctl then the memory $m$ becomes STORE13 mar bus m - i.e. a memory identical to $m$ except that location mar contains value bus. (As mentioned above, the definition of BUS below ensures that the value on line bus is an ordinary 16-bit word, not a tri-state word).

The host uses three kinds of registers: 13 and 16-bit registers with load-enable control line:

$$\text{REG13} : \ \text{word13} \rightarrow \ \text{dev}$$
$$\text{REG16} : \ \text{word16} \rightarrow \ \text{dev}$$

With behaviour:

$$\text{REG13 w} = \text{dev}(\text{t, id, o13}), [\text{o13}=w]; \text{REG13}(\text{id} \rightarrow \text{CUT1d_13 t|w})$$
$$\text{REG16 w} = \text{dev}(\text{t, id, o16}), [\text{o16}=w]; \text{REG16}(\text{id} \rightarrow \text{t|w})$$

Using these we can define:

$$\text{ACC} \ w = (\text{REG16 w}) \ \text{rn} \ [t=\text{bus} ; \text{id}=\text{wacc}; o16=\text{acc}]$$
$$\text{IR} \ w = (\text{REG16 w}) \ \text{rn} \ [t=\text{bus} ; \text{id}=\text{wir} ; o16=\text{ir}]$$
$$\text{ARG} \ w = (\text{REG16 w}) \ \text{rn} \ [t=\text{bus} ; \text{id}=\text{warg}; o16=\text{arg}]$$
$$\text{MAR} \ w = (\text{REG13 w}) \ \text{rn} \ [t=\text{bus} ; \text{id}=\text{wmar}; o13=\text{mar}]$$
$$\text{PC} \ w = (\text{REG13 w}) \ \text{rn} \ [t=\text{bus} ; \text{id}=\text{wpc} ; o13=\text{pc}]$$

The third kind of register has no load-enable line; the buffer register is the only example of this used, so we define it directly:

$$\text{BUF} \ w = \text{dev(alu, buf)}, [\text{buf}=w] ; \text{BUF ahu}$$

The gate GI is the only gate which has a 13-bit input. We define it directly by:

$$\text{GI} = \text{dev(pc, rpe, gi)}, [\text{gi}=(\text{rpe} \rightarrow \text{MK_TRI16(PAD13_16 pc)} | \text{FLOAT16})] ; \text{GI}$$
The remaining four gates all have 16-bit inputs. We will define them in terms of a generic gate device \textit{GATE}:

\[
GATE \equiv dev\{i, cntl, o\}.\{o=(cntl \rightarrow MK\_TRI16 \& FLOAT16)\}; GATE
\]

Then:

\[
\begin{align*}
G0 & \equiv GATE\; \text{rn}\{i=\text{switches}; cntl=rsw; o=g0\} \\
G2 & \equiv GATE\; \text{rn}\{i=\text{acc}; cntl=racc; o=g2\} \\
G3 & \equiv GATE\; \text{rn}\{i=\text{ir}; cntl=\text{rir}; o=g3\} \\
G4 & \equiv GATE\; \text{rn}\{i=\text{buf}; cntl=\text{rbuf}; o=g4\}
\end{align*}
\]

The alu and bus are directly specified by:

\[
\begin{align*}
\text{ALU} & \equiv dev\{arg, \text{bus}, \text{alu}, \text{cntl}, \text{alu}\}. \\
& \{\text{alu}=(VAL2\; \text{alu} = 0 \rightarrow \text{bus} \mid \text{VAL2\; \text{alu} = 1 \rightarrow INC16\; \text{bus} \mid \text{VAL2\; \text{alu} = 2 \rightarrow ADD16\; \text{arg}\; \text{bus} \mid SUB16\; \text{arg}\; \text{bus})}; \\
\text{ALU} \}
\end{align*}
\]

\[
\begin{align*}
\text{BUS} & \equiv dev\{\text{mem}, g0, g1, g2, g3, g4\}. \\
& \{\text{bus}=\text{DEST\_TRI16}(\text{mem}\; U16\; g0\; U16\; g1\; U16\; g2\; U16\; g3\; U16\; g4)\}; \\
\text{BUS} \}
\end{align*}
\]

Note that the value output on line \textit{bus} has LSM type \textit{word16} - i.e. it is not a tri-state value.

Before defining the microinstruction decoding device \textit{DECODE}, we must describe the 30-bit microinstructions of the host. Bits 0 to 2 are are 3-bit opcode called the test field. Bits 8 to 12 hold a 5-bit microinstruction address called the A-address, and bits 3 to 7 hold another 5-bit address called the B-address. The remaining bits are all control fields, each field determining the value on a control line during a microcycle. The microinstructions are thus fully "horizontal".

The address of the next microinstruction to be executed is normally the contents of the A-address field, unless:

1. The value denoted by the test field is 1 and \textit{T} is input on line \textit{button}, or the value denoted by the test field is 2 and the value denoted by the 16-bit word on the \textit{acc} line is 0. In either of these two cases the next address is the contents the B-address field.

2. The value denoted by the test field is 3. In this case the next address is obtained by adding the value denoted by the 2-bit word on the \textit{knob} line to the value in the A-address field.

3. The value denoted by the test field is 4. In this case the next address is obtained by adding the value of bits 13 to 15 of the 16-bit word on line \textit{ir} (i.e. the opcode) to the value in the A-address field.
Thus if the test field contains \#001 then pressing the button causes a branch to the B-address; if the test field contains \#010 then a branch to the B-address occurs if the accumulator contains \#0000000000000000; if the test field contains \#011 then a branch to a microinstruction determined by the position of the knob occurs, and if the test field contains \#100 then a branch to a microinstruction determined by the opcode of the current machine instruction (i.e. the contents of the instruction register) occurs.

The actual microcode for the host is represented in LSM by a constant MICROCODE: mem5_30. This is defined by 26 axioms, one for each microinstruction:

\[
\begin{align*}
\text{FETCH6 MICROCODE(WORD5 0)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 1)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 2)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 3)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 4)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 5)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 6)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 7)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 8)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 9)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 10)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 11)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 12)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 13)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 14)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 15)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 16)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 17)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 18)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 19)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 20)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 21)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 22)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 23)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 24)} & \equiv \text{word30} \\
\text{FETCH6 MICROCODE(WORD5 25)} & \equiv \text{word30}
\end{align*}
\]

These axioms were generated by a simple microassembler written in ML. The ML code for this microassembler, together with the symbolic input which generates the above 26 axioms is listed in Appendix 1 of [Gordon4]. We do not specify the microinstructions with addresses 26 to 31.

To make the definition of \textit{DECODE} more readable we introduce some auxiliary functions.

\begin{align*}
\text{CNTL\_BIT} & : \text{num} \rightarrow \text{word30} \rightarrow \text{bool} \\
\text{CNTL\_FIELD} & : \text{num} \rightarrow \text{word30} \rightarrow \text{word30} \\
\text{A\_ADDR} & : \text{word30} \rightarrow \text{word30} \\
\text{B\_ADDR} & : \text{word30} \rightarrow \text{word30} \\
\text{TEST} & : \text{word30} \rightarrow \text{num}
\end{align*}

These are defined by axioms:

\[
\text{CNTL\_BIT } n \ w \quad \equiv \text{EL } n \ (\text{BITS30 } w)
\]
Thus if \( w \) is a microinstruction (i.e. a 30-bit word) then: \( \text{CNTL\_BIT} n w \) is the \( n \)th bit of \( w \); \( \text{CNTL\_FIELD}(m,n)w \) is the 2-bit word consisting of bits \( m \) and \( n \) of \( w \); \( \text{A\_ADDR} w \) is the A-address of \( w \); \( \text{B\_ADDR} w \) is the B-address of \( w \), and \( \text{TEST} w \) is the test field of \( w \). Notice how we have defined these functions by first converting \( w \) to a list of booleans and then using the list-processing functions \( \text{EL} \) and \( \text{SEG} \).

The interpretation of microinstructions described above is embodied in the specification of \( \text{DECODE} \):

\[
\begin{align*}
\text{DECODE} &= \quad \text{dev\_rom, knob, button, acc, ir, nest\_address, rsw, u\_status, mem\_cntl, upc, rpc,} \\
& \quad \text{ uacc, rac, wir, rir, warg, al\_cntl, rbuf, ready, idle}. \\
\text{nest\_address} &= ((\text{TEST} \text{ rom} = 1) \text{ AND button} \\
& \quad \rightarrow \text{ B\_ADDR} \text{ rom}) \\
& \quad \rightarrow \text{ B\_ADDR} \text{ rom} \mid \\
& \quad \rightarrow \text{ B\_ADDR} \text{ rom} \mid \\
& \quad \rightarrow \text{ B\_ADDR} \text{ rom} \mid \\
& \quad \rightarrow \text{ B\_ADDR} \text{ rom} \mid \\
\text{rsw} &= \text{ CNTL\_BIT} 28 \text{ rom}, \\
\text{u\_status} &= \text{ CNTL\_BIT} 27 \text{ rom}, \\
\text{mem\_cntl} &= \text{ CNTL\_FIELD} (25,26) \text{ rom}, \\
\text{upc} &= \text{ CNTL\_BIT} 24 \text{ rom}, \\
\text{rpc} &= \text{ CNTL\_BIT} 23 \text{ rom}, \\
\text{uacc} &= \text{ CNTL\_BIT} 22 \text{ rom}, \\
\text{rac} &= \text{ CNTL\_BIT} 21 \text{ rom}, \\
\text{wir} &= \text{ CNTL\_BIT} 20 \text{ rom}, \\
\text{rir} &= \text{ CNTL\_BIT} 19 \text{ rom}, \\
\text{warg} &= \text{ CNTL\_BIT} 18 \text{ rom}, \\
\text{al\_cntl} &= \text{ CNTL\_FIELD} (16,17) \text{ rom}, \\
\text{rbuf} &= \text{ CNTL\_BIT} 15 \text{ rom}, \\
\text{ready} &= \text{ CNTL\_BIT} 14 \text{ rom}, \\
\text{idle} &= \text{ CNTL\_BIT} 13 \text{ rom}; \\
\end{align*}
\]

This completes the specification of the host.

Proof of Correctness

To verify that the host machine correctly implements the target machine we must show:

\[
\begin{align*}
\text{m w0 w1 w2 w3 w4 w5 t.} \\
\text{COMPUTER}(m,w1,w2,t) \equiv \\
\text{ until ready do HOST(WORD5(t->0|5),m,w0,w1,w2,w3,w4,w5)}
\end{align*}
\]

It is convenient to introduce a constant:

\[
\text{COMPUTER\_IMP} : \text{bool\_mem13\_16\_word13\_word16\_word16\_word16\_word16\_word16\_word16}\]

Defined by:

-11-
COMPUTER_IMP(t,m,w0,w1,w2,w3,w4,w5) ==
until ready do HOST(WORD5(t->0|5),m,w0,w1,w2,w3,w4,w5)

Then correctness is simply:

:m w0 w1 w2 w3 w4 w5 t.
COMPUTER(m,w1,w2,t) ==
COMPUTER_IMP(t,m,w0,w1,w2,w3,w4,w5)

We start the proof by deriving a behaviour equation for HOST. This is done in two stages. First we use EXPAND_IMP to derive an equation for CONTROL which we call CONTROL_EQN.

!r w.
CONTROL(r,w) ==
dev[knob, button, acc, ir, rsu, wmar, momentl, upc, opc, 
wacc, rac, wir, rir, war, aluentl, rbuf, ready, idle].
| rsu=(CUTL_BIT 28(FETCH5 r w)),
wmar=(CUTL_BIT 27(FETCH5 r w)),
momentl=(CUTL_FIELD(25,26)(FETCH5 r w)),
| upc=(CUTL_BIT 24(FETCH5 r w)),
rpc=(CUTL_BIT 23(FETCH5 r w)),
wacc=(CUTL_BIT 22(FETCH5 r w)),
racc=(CUTL_BIT 21(FETCH5 r w)),
| wir=(CUTL_BIT 20(FETCH5 r w)),
| rir=(CUTL_BIT 19(FETCH5 r w)),
| war=(CUTL_BIT 18(FETCH5 r w)),
| aluentl=(CUTL_FIELD(16,17)(FETCH5 r w)),
rbuf=(CUTL_BIT 16(FETCH5 r w)),
| ready=(CUTL_BIT 14(FETCH5 r w)),
| idle=(CUTL_BIT 13(FETCH5 r w));
CONTROL
(r, 
| ((TEST(FETCH5 r w)) = 1) AND button ->
| B_ADDR(FETCH5 r w) | )
| ((TEST(FETCH5 r w)) = 2) AND ((VAL16 acc) = 0) ->
| B_ADDR(FETCH5 r w) | )
| ((TEST(FETCH5 r w)) = 3 ->
| WORDS(((VAL12 knob) + 1) + (VAL5(A_ADDR(FETCH5 r w)))) | )
| ((TEST(FETCH5 r w)) = 4 ->
| WORDS5(((VAL3(OPCODE ir)) + (VAL5(A_ADDR(FETCH5 r w))))) | )
| A_ADDR(FETCH5 r w)))))

Next we derive a behaviour equation for DATA which we call DATA_EQN. In order to derive this we must use EXPAND_DEF to deduce the following equations for the primitives which are defined in terms of generic devices.

!w. ACC w == dev[bus, wacc, acc]. [acc=w]; ACC(wacc -> bus | w)
!w. IR w == dev[bus, wir, ir]. [ir=w]; IR(wir -> bus | w)
!w. ARG w == dev[bus, warg, arg]. [arg=w]; ARG(warg -> bus | w)
!w. MAR w == dev[bus, wmar, mar]. [mar=w]; MAR(wmar -> CUT16_13 bus | w)
!w. PC w == dev[bus, upc, pc]. [pc=w]; PC(upc -> CUT16_13 bus | w)
G0 == dev[switches, rsu, g0]. [g0=(rsu -> MK_TRI16 switches | FLOAT16)]; G0
G2 == dev[acc, rac, g2]. [g2=(rac -> MK_TRI16 acc | FLOAT16)]; G2
G3 == dev[ir, rir, g3]. [g3=(rir -> MK_TRI16 ir | FLOAT16)]; G3
\[ G_4 \equiv \text{dev}[\text{buf}, \text{rbuf}, g_4]. \{ g_4 \to \text{MK\_TRI16 buf } | \text{FLOAT16}\}; G_4 \]

Using these theorems, together with the behaviour equations for the rest of the primitives, we can derive the following behaviour equation for the data part of the host.

\[
\begin{align*}
\text{DATA}(m, w_0, w_1, w_2, w_3, w_4, w_5) & \equiv \nonumber \\
& \text{dev}[^\text{memctl}, \text{unar}, \text{upc}, \text{pc}, \text{wacc}, \text{acc}, \text{wir}, \text{ir}, \\
& \text{warg, switches}, \text{rsz}, \text{rpc}, \text{accr}, \text{rir}, \text{rbuf}, \text{alucontl}]. \\
& \{ \text{pc} = w_1, \\
& \text{acc} = w_2, \\
& \text{tr} = w_3, \\
& \text{bus} = (\text{DEST\_TRI16} \\
& \hspace{1cm} (\text{VAL2 memoctl}) = 1 \to \text{MK\_TRI16(FETCH13 m w_0 | FLOAT16) U16} \\
& \hspace{2cm} (\text{rsz} \to \text{MK\_TRI16 switches | FLOAT16) U16} \\
& \hspace{2cm} (\text{rpc} \to \text{MK\_TRI16(PAD\_13 w_1) | FLOAT16) U16} \\
& \hspace{2cm} (\text{racc} \to \text{MK\_TRI16 w_2 | FLOAT16) U16} \\
& \hspace{2cm} (\text{rir} \to \text{MK\_TRI16 w_3 | FLOAT16) U16} \\
& \hspace{2cm} (\text{rbuf} \to \text{MK\_TRI16 w_5 | FLOAT16) U16})})]) ; \\
& \text{DATA} \\
& \hspace{1cm} (\text{VAL2 memoctl}) = 2 \to \text{STORE13 w_0 bus m | m}, \\
& \hspace{2cm} (\text{unar} \to \text{CUT16\_13 bus | w_0}), \\
& \hspace{2cm} (\text{upc} \to \text{CUT16\_13 bus | w_1}), \\
& \hspace{2cm} (\text{wacc} \to \text{bus | w_2}), \\
& \hspace{2cm} (\text{wir} \to \text{bus | w_3}), \\
& \hspace{2cm} (\text{warg} \to \text{bus | w_4}), \\
& \hspace{2cm} (\text{VAL2 alucontl}) = 0 \to \text{bus |} \\
& \hspace{3cm} \text{INC16 bus |} \\
& \hspace{3cm} (\text{VAL2 alucontl}) = 1 \to \text{ADD16 w_4 bus | SUB16 w_4 bus}))
\end{align*}
\]

Now we can put together the behaviour equations for the control and data parts to get, via \textit{EXPAND\_IMP}, the following equation for \texttt{HOST}:

\[
\begin{align*}
\text{HOST}(w_0, w_1, w_2, w_3, w_4, w_5) & \equiv \nonumber \\
& \text{dev}[^\text{knob}, \text{button}, \text{acc}, \text{ready}, \text{idle}, \text{pc}, \text{switches}]. \\
& \{ \text{ready} = \text{CNTL\_BIT 14(FETCH5 MICROCODE w)}, \\
& \hspace{1cm} (\text{CNTL\_BIT 13(FETCH5 MICROCODE w)}), \\
& \hspace{2cm} \text{pc} = w_1, \\
& \hspace{2cm} \text{acc} = w_2, \\
& \hspace{2cm} \text{bus} = (\text{DEST\_TRI16} \\
& \hspace{3cm} (\text{VAL2(CNTL\_FIELD(25,26)(FETCH5 MICROCODE w))) = 1 \to \text{MK\_TRI16(FETCH13 m w_0 | FLOAT16) U16} \\
& \hspace{4cm} (\text{CNTL\_BIT 28(FETCH5 MICROCODE w)} \to \text{MK\_TRI16 switches | FLOAT16) U16} \\
& \hspace{4cm} (\text{CNTL\_BIT 29(FETCH5 MICROCODE w)} \to \text{MK\_TRI16(PAD\_13 w_1) | FLOAT16) U16} \\
& \hspace{4cm} (\text{CNTL\_BIT 21(FETCH5 MICROCODE w)} \to \text{MK\_TRI16 w_2 | FLOAT16) U16} \\
& \hspace{4cm} (\text{CNTL\_BIT 19(FETCH5 MICROCODE w)} \to \text{MK\_TRI16 w_3 | FLOAT16) U16} \\
& \hspace{4cm} (\text{CNTL\_BIT 15(FETCH5 MICROCODE w)} \to \text{MK\_TRI16 w_5 | FLOAT16) U16})})]) ; \\
& \text{HOST} \\
& \hspace{1cm} (\text{TEST(FETCH5 MICROCODE w)} = 1) \text{ AND button} \to \text{B\_ADDR(FETCH5 MICROCODE w)} \\
& \hspace{2cm} (\text{TEST(FETCH5 MICROCODE w)} = 2) \text{ AND (VAL16 w_2 = 0) \to B\_ADDR(FETCH5 MICROCODE w)} \\
& \hspace{2cm} (\text{TEST(FETCH5 MICROCODE w)} = 3) \to \\
\end{align*}
\]
WORDS
((((VAL2 knob) + 1) + (VAL6(A_ADDR(FETCH5 MICROCODE w)))) | 
((TEST(FETCH5 MICROCODE w)) = 4 -> 
WORDS 
((VAL3(OPCODE w3)) + (VAL5(A_ADDR(FETCH5 MICROCODE w)))) | 
A_ADDR(FETCH5 MICROCODE w)))), 
((((VAL3(CONTL_FIELD(25,26)(FETCH5 MICROCODE w)))) = 2 -> 
STORE13 w0 bus m | 
m).
(CONTL_BIT 27(FETCH5 MICROCODE w) -> CUT16_13 bus | w0),
(CONTL_BIT 24(FETCH5 MICROCODE w) -> CUT16_13 bus | w1),
(CONTL_BIT 22(FETCH5 MICROCODE w) -> bus | w2),
(CONTL_BIT 20(FETCH5 MICROCODE w) -> bus | w3),
(CONTL_BIT 18(FETCH5 MICROCODE w) -> bus | w4).
((((VAL3(CONTL_FIELD(16,17)(FETCH5 MICROCODE w)))) = 0 -> 
bus | 
((VAL3(CONTL_FIELD(16,17)(FETCH5 MICROCODE w)))) = 1 -> 
INC16 bus | 
((VAL3(CONTL_FIELD(16,17)(FETCH5 MICROCODE w)))) = 2 -> 
ADD16 w4 bus | 
SUB16 w4 bus)))))

We call this HOST_EQN. Next we use the rule UNTIL to derive an equation for until ready do HOST(W,m,w0,w1,w2,w3,w4,w5). The resulting theorem, which we call HOST_UNTIL_EQN, is rather big. It is:

!host. 
(button knob switches w m w0 w1 w2 w3 w4 w5. 
<equation> 
=> 
(W m w0 w1 w2 w3 w4 w5. 
until ready do HOST(W,m,w0,w1,w2,w3,w4,w5) == 
dev{knob,button,acc,Idle,pc,switches}. 
[Idle=(CONTL_BIT 13(FETCH5 MICROCODE w)), 
pc=w1, 
acc=w2, 
bus=(DEST_TRI16 
((((VAL2(CONTL_FIELD(25,26)(FETCH5 MICROCODE w)))) = 1 -> 
MK_TRI16(FETCH13 m w0) | 
FLOAT16) U16 
((CONTL_BIT 28(FETCH5 MICROCODE w) -> 
MK_TRI16 switches | 
FLOAT16) U16 
((CONTL_BIT 29(FETCH5 MICROCODE w) -> 
MK_TRI16(PAD13_16 w1) | 
FLOAT16) U16 
((CONTL_BIT 21(FETCH5 MICROCODE w) -> 
MK_TRI16 w2 | 
FLOAT16) U16 
((CONTL_BIT15(FETCH5 MICROCODE w) -> 
MK_TRI16 w3 | 
FLOAT16) U16 
((CONTL_BIT15(FETCH5 MICROCODE w) -> 
MK_TRI16 w5 | 
FLOAT16)))))))).

until ready do 
HOST 
(host (button. 
knob. 
switches. 
((TEST(FETCH5 MICROCODE w)) = 1) AND button -> 
A_ADDR(FETCH5 MICROCODE w) | 
((TEST(FETCH5 MICROCODE w)) = 2) AND ((VAL16 w2) = 0) -> 
A_ADDR(FETCH5 MICROCODE w) | 
((TEST(FETCH5 MICROCODE w)) = 3 -> 
WORDS 
(((VAL2 knob) + 1) + 
-14-
Where \(<\text{equation}>\) is the formula:

\[
\text{host}(\text{button}, \text{knob}, \text{switches}, w, m, w_0, w_1, w_2, w_3, w_4, w_5) = \neg \text{let } b = \text{DEST\_TRI16} \\
(\text{VAL2}(\text{CNTL\_FIELD}(25, 26)(\text{FETCH5 MICROCODE } w))) = 1 \rightarrow \\
\text{MK\_TRI16}(\text{FETCH13 } m \ w_0) \\
\text{FLOAT16}(\text{M16}) \ U16 \\
(\text{CNTL\_BIT } 20(\text{FETCH5 MICROCODE } w) \rightarrow \\
\text{MK\_TRI16(\text{PAD13\_16 } w_1)} \\
\text{MK\_TRI16 switches} \\
\text{FLOAT16}(\text{M16}) \ U16 \\
(\text{CNTL\_BIT } 21(\text{FETCH5 MICROCODE } w) \rightarrow \\
\text{MK\_TRI16 w_2} \\
\text{FLOAT16}(\text{M16}) \ U16 \\
(\text{CNTL\_BIT } 19(\text{FETCH5 MICROCODE } w) \rightarrow \\
\text{MK\_TRI16 w_3} \\
\text{FLOAT16}(\text{M16}) \ U16 \\
(\text{CNTL\_BIT } 15(\text{FETCH5 MICROCODE } w) \rightarrow \\
\text{MK\_TRI16 w_5} \\
\text{FLOAT16}(\text{M16}))))) \\
\text{in} \text{(CNTL\_BIT } 14(\text{FETCH5 MICROCODE } w) \rightarrow \\
(w, m, w_0, w_1, w_2, w_3, w_4, w_5) \\
\text{host}(\text{button}, \text{knob}, \\
\text{switches}) \\
(\text{TEST}(\text{FETCH5 MICROCODE } w) = 1) \text{ AND button} \rightarrow \\
\text{ADDR}(\text{FETCH5 MICROCODE } w) \\
(\text{TEST}(\text{FETCH5 MICROCODE } w) = 2) \text{ AND } ((\text{VAL16 } w_2) = 0) \rightarrow \\
\text{ADDR}(\text{FETCH5 MICROCODE } w) \\
(\text{TEST}(\text{FETCH5 MICROCODE } w)) = 3 \rightarrow \\
\text{WORDS} \\
((\text{VAL2 } \text{knob} = 1) + \\
(\text{VAL5 (A\_ADDR(\text{FETCH5 MICROCODE } w)))) \\
(\text{TEST}(\text{FETCH5 MICROCODE } w)) = 4 \rightarrow \\
\text{WORDS} \\
((\text{VAL3 (OPCODE } w3)) + \\
(\text{VAL5 (A\_ADDR(\text{FETCH5 MICROCODE } w)))) \\
\text{A\_ADDR(\text{FETCH5 MICROCODE } w))}) \\
(\text{VAL5 (CNTL\_FIELD}(25, 26)(\text{FETCH5 MICROCODE } w)) = 2 \rightarrow \\
\text{STORR13 } w_0 \text{ bus } m \\
\text{m}, \\
(\text{CNTL\_BIT } 27(\text{FETCH5 MICROCODE } w) \rightarrow \text{CUT16}\_13 \text{ bus } w_0, \\
(\text{CNTL\_BIT } 24(\text{FETCH5 MICROCODE } w) \rightarrow \text{CUT16}\_13 \text{ bus } w_1), \\
(\text{CNTL\_BIT } 22(\text{FETCH5 MICROCODE } w) \rightarrow \text{bus } w_2, \\
(\text{CNTL\_BIT } 20(\text{FETCH5 MICROCODE } w) \rightarrow \text{bus } w_3). \\
\text{FLOAT16}(\text{M16}) \ U16
\(\text{CINTL\_BIT}\ 16(\text{FETCH5 MICROCODE}\ w) \rightarrow\text{bus}\ \mid\ w4),\)
\((\text{VAL2}(\text{CINTL\_FIELD}(16,17)(\text{FETCH5 MICROCODE}\ w))) = 0 \rightarrow\)
\(\text{bus}\ \mid\)
\((\text{VAL2}(\text{CINTL\_FIELD}(16,17)(\text{FETCH5 MICROCODE}\ w))) = 1 \rightarrow\)
\(\text{INC16}\ \text{bus}\ \mid\)
\((\text{VAL2}(\text{CINTL\_FIELD}(16,17)(\text{FETCH5 MICROCODE}\ w))) = 2 \rightarrow\)
\(\text{ADD16}\ w4\ \text{bus}\ \mid\)
\(\text{SUB16}\ w4\ \text{bus}\).\)

We now introduce a constant called \(\text{HOST\_fn}\) to name the function defined by \(<\text{equation}>\). This constant has \(\text{OL}\) type:

\[
\begin{align*}
\text{HOST\_fn}(\text{button, knob, switches, w,m,w0,w1,w2,w3,w4,w5}) &= \\
\text{let bus =} \\
\text{DEST\_TRI16} \\
((\text{VAL2}(\text{CINTL\_FIELD}(25,26)(\text{FETCH5 MICROCODE}\ w))) = 1 \rightarrow\)
\(\text{MK\_TRI16}(\text{FETCH13}\ m\ w0) \mid\)
\(\text{FLOAT16})\ \text{U16} \\
((\text{CINTL\_BIT}\ 29(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\(\text{MK\_TRI16}\ \text{switches} \mid\)
\(\text{FLOAT16})\ \text{U16} \\
((\text{CINTL\_BIT}\ 23(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\(\text{MK\_TRI16}(\text{PAD13}\_16\ w1) \mid\)
\(\text{FLOAT16})\ \text{U16} \\
((\text{CINTL\_BIT}\ 21(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\(\text{MK\_TRI16}\ w2 \mid\)
\(\text{FLOAT16})\ \text{U16} \\
((\text{CINTL\_BIT}\ 19(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\(\text{MK\_TRI16}\ w3 \mid\)
\(\text{FLOAT16})\ \text{U16} \\
((\text{CINTL\_BIT}\ 18(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\(\text{MK\_TRI16}\ w5 \mid\)
\(\text{FLOAT16})\ \text{U16}))\}
\text{in}\ \)
\((\text{CINTL\_BIT}\ 14(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\((w,m,w0,w1,w2,w3,w4,w5))\)
\text{HOST\_fn} \\
\text{(button, knob, switches, w,m,w0,w1,w2,w3,w4,w5}) \\
\end{align*}
\]

\(\text{HOST\_fn}\) is defined to satisfy the recursive definition generated by the \textit{UNTIL} rule (i.e. the equation \(<\text{equation}>\) for \textit{host} which is the antecedent of the implication in \textit{HOST\_UNTIL\_EQN}). The axiom defining \(\text{HOST\_fn}\) is thus the formula \(<\text{equation}>\) with the variable \textit{host} replaced by the constant \(\text{HOST\_fn}\), i.e.

\[
\begin{align*}
\text{HOST\_fn}(\text{button, knob, switches, w,m,w0,w1,w2,w3,w4,w5}) &= \\
\text{let bus =} \\
\text{DEST\_TRI16} \\
((\text{VAL2}(\text{CINTL\_FIELD}(25,26)(\text{FETCH5 MICROCODE}\ w))) = 1 \rightarrow\)
\(\text{MK\_TRI16}(\text{FETCH13}\ m\ w0) \mid\)
\(\text{FLOAT16})\ \text{U16} \\
((\text{CINTL\_BIT}\ 29(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\(\text{MK\_TRI16}\ \text{switches} \mid\)
\(\text{FLOAT16})\ \text{U16} \\
((\text{CINTL\_BIT}\ 23(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\(\text{MK\_TRI16}(\text{PAD13}\_16\ w1) \mid\)
\(\text{FLOAT16})\ \text{U16} \\
((\text{CINTL\_BIT}\ 21(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\(\text{MK\_TRI16}\ w2 \mid\)
\(\text{FLOAT16})\ \text{U16} \\
((\text{CINTL\_BIT}\ 19(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\(\text{MK\_TRI16}\ w3 \mid\)
\(\text{FLOAT16})\ \text{U16} \\
((\text{CINTL\_BIT}\ 18(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\(\text{MK\_TRI16}\ w5 \mid\)
\(\text{FLOAT16})\ \text{U16}))\}
\text{in}\ \)
\((\text{CINTL\_BIT}\ 14(\text{FETCH5 MICROCODE}\ w) \rightarrow\)
\((w,m,w0,w1,w2,w3,w4,w5))\)
\text{HOST\_fn} \\
\text{(button, knob, switches, w,m,w0,w1,w2,w3,w4,w5}) \\
\end{align*}
\]
(VAL2(CNTL_FIELD(16,17))(FETCH5 MICROCODE w))) = 1 ->
INC16 bus |
((VAL2(CNTL_FIELD(16,17))(FETCH5 MICROCODE w))) = 2 ->
ADD16 w4 bus |
SUB16 w4 bus)))))

We can now simplify HOST_UNTIL_EQN by specializing the variable host to the constant HOST_fn, and then doing modus ponens with the above axiom. This yields the following theorem:

until ready do HOST(u,m,w0,w1,w2,w3,w4,w5) ==
dev
| knob, button, acc, idle, pc, switches |
idle=(EL 13(BITS30(FETCH5 MICROCODE w))).
pcc=w1,
acc=w2,
buss=(DEST_TRI16
(VAL2(WORDS(Y(SEG(25,26))(BITS30(FETCH5 MICROCODE w)))))) =
(1 ->
MK_TRI16(FETCH13 m w0) |
FLOAT16) U16
((EL 20)(BITS30(FETCH5 MICROCODE w)) ->
MK_TRI16 switches |
FLOAT16) U16
((EL 21)(BITS30(FETCH5 MICROCODE w)) ->
MK_TRI16(PAD13_16 w1) |
FLOAT16) U16
((EL 19)(BITS30(FETCH5 MICROCODE w)) ->
MK_TRI16 w2 |
FLOAT16) U16
((EL 17)(BITS30(FETCH5 MICROCODE w)) ->
MK_TRI16 w3 |
FLOAT16) U16
((EL 15)(BITS30(FETCH5 MICROCODE w)) ->
MK_TRI16 w5 |
FLOAT16)))))));

until ready do
HOST_fn
(button, knob, switches,
(VAL2(CNTL_FIELD(0,2))(BITS30(FETCH5 MICROCODE w)))) = 1 AND button ->
WORDS5(Y(SEG(0,2))(BITS30(FETCH5 MICROCODE w))))) |
((VAL2(CNTL_FIELD(0,2))(BITS30(FETCH5 MICROCODE w)))) = 2 AND
((VAL2 w2) = 0) ->
WORDS5(Y(SEG(3,7))(BITS30(FETCH5 MICROCODE w))))) |
((VAL2(CNTL_FIELD(0,2))(BITS30(FETCH5 MICROCODE w)))) = 3 ->
WORDS5
(((VAL2 knob) + 1) +
(VAL5(WORDS5(Y(SEG(8,12))(BITS30(FETCH5 MICROCODE w)))))) |
((VAL5(Y(SEG(0,2))(BITS30(FETCH5 MICROCODE w)))) = 4 ->
WORDS5
(((VAL3(OPS_CODE w3)) +
(VAL5(WORDS5(Y(SEG(8,12))(BITS30(FETCH5 MICROCODE w)))))))) |
((VAL2(WORDS2(Y(SEG(25,26))(BITS30(FETCH5 MICROCODE w)))))) = 2 ->
STORE13 w0 bus m |

(EL 27)(BITS30(FETCH5 MICROCODE w)) -> CUT16_13 bus |
(EL 24)(BITS30(FETCH5 MICROCODE w)) -> CUT16_13 bus |
(EL 22)(BITS30(FETCH5 MICROCODE w)) -> bus w2, |
(EL 20)(BITS30(FETCH5 MICROCODE w)) -> bus w3, |
(EL 18)(BITS30(FETCH5 MICROCODE w)) -> bus w5, |
(VAL2(WORDS2(Y(SEG(16,17))(BITS30(FETCH5 MICROCODE w)))))) = 0 ->
bus |
((VAL2(WORDS2(Y(SEG(16,17))(BITS30(FETCH5 MICROCODE w)))))) = 1 ->
INC16 bus |

-17-
\((\text{VAL2}(\text{WORD2}(\text{SEG}(16, 17)(\text{BITS30}(\text{FETCH5 MICROCODE }w)))))) = 2 \rightarrow \\
\text{ADD16 } w4 \text{ bus } | \\
\text{SUB16 } w4 \text{ bus })\))

The variable \(w\) here is the address of the current microinstruction. From the definition of \text{COMPUTER_IMP} we see that two important cases are when this address is (a 5-bit word denoting) 0, and when it is 5. These cases correspond to the start of the microcode for idling and executing respectively. Using the various evaluation rules and the definition of MICROCODE we can derive the following two theorems (we precede them by their names).

\text{HOST\_UNTIL\_EQN\_0}

\begin{align*}
\text{until \ ready \ do} & \text{ HOST}($\#00000, m, w0, w1, w2, w3, w4, w5) \equiv \\
\text{dev}[\text{knob, button, acc, idle, pc, switches}]. & \\
\{\text{idle}=T, \text{pc}=w1, \text{acc}=w2\}; \\
\text{until \ ready \ do} & \text{ HOST} \\
(\text{HOST\_fn}) & \text{ (button,} \\
\text{knob,} & \text{ switches,} \\
(\text{button} \rightarrow \#00001 \mid \#00000), & \text{ m,} \\
\text{w0,} & \text{w1,} \\
\text{w2,} & \text{w3,} \\
\text{w4,} & \text{DEST\_TRI16 FLOAT16}) \\
\text{HOST\_UNTIL\_EQN\_5}
\end{align*}

\begin{align*}
\text{until \ ready \ do} & \text{ HOST}($\#00101, m, w0, w1, w2, w3, w4, w5) \equiv \\
\text{dev}[\text{knob, button, acc, idle, pc, switches}]. & \\
\{\text{idle}=F, \text{pc}=w1, \text{acc}=w2\}; \\
\text{until \ ready \ do} & \text{ HOST} \\
(\text{HOST\_fn}) & \text{ (button,} \\
\text{knob,} & \text{ switches,} \\
(\text{button} \rightarrow \#00000 \mid \#00110), & \text{ m,} \\
\text{w0,} & \text{w1,} \\
\text{w2,} & \text{w3,} \\
\text{w4,} & \text{DEST\_TRI16 FLOAT16})
\end{align*}

We must now consider each or the 26 microinstructions separately. Using the axioms defining MICROCODE followed by simplification using the various evaluation rules, we can derive:

\begin{align*}
\text{HOST\_fn(button, knob, switches, \#00000, m, w0, w1, w2, w3, w4, w5)} & = \#00000, m, w0, w1, w2, w3, w4, w5 \\
\text{HOST\_fn(button, knob, switches, \#00001, m, w0, w1, w2, w3, w4, w5)} & = \\
\end{align*}
HOST_fn
(button,
knob,
switches,
WORD0(((VAL3 knob) + 1) + 1),
m,
w0,
w1,
w2,
w3,
w4,
DEST_TRI16 FLOAT16)

HOST_fn(button,knob,switches,00010,m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button,
knob,
switches,
00000,
m,
w0,
CUT16_13 switches,
w2,
w3,
w4,
switches)

HOST_fn(button,knob,switches,00011,m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button,knob,switches,00000,m,w0,w1,switches,w3,w4,switches)

HOST_fn(button,knob,switches,00100,m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button,knob,switches,00111,m,w1,w1,w2,w3,w4,PAD13_16 w1)

HOST_fn(button,knob,switches,00101,m,w0,w1,w2,w3,w4,w5) ==
00101,m,w0,w1,w2,w3,w4,w5

HOST_fn(button,knob,switches,00110,m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button,knob,switches,01000,m,w1,w1,w2,w3,w4,PAD13_16 w1)

HOST_fn(button,knob,switches,00111,m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button,knob,switches,00000,STORE13 w0 w2 m,w0,w1,w2,w3,w4,w2)

HOST_fn(button,knob,switches,01000,m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button,
knob,
switches,
01001,
m,
w0,
w1,
w2,
FETCH13 m w0,
w5,
FETCH13 m w0)

HOST_fn(button,knob,switches,01001,m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button,
knob,
switches,
WORD0((VAL3(OPCODE w3)) + 10),
m,
w0,
w1,
w2,
w3,
w4,
DEST_TRI16 FLOAT16)
HOST_fn(button, knob, switches, #01010, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button, knob, switches, #00000, m, w0, w1, w2, w3, w4, DEST_TRI16 FLOAT16)
HOST_fn(button, knob, switches, #01011, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button, knob, switches, #00101, m, w0, CUT16_13 w3, w2, w3, w4, w3)
HOST_fn(button, knob, switches, #01100, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button, knob, switches, ((VAL16 w2) = 0 -> #01011 | #10001), m, w0, w1, w2, w3, w4, DEST_TRI16 FLOAT16)
HOST_fn(button, knob, switches, #01101, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn(button, knob, switches, #10011, m,w0,w1,w2,w3,w2,w2)
HOST_fn(button, knob, switches, #01110, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn(button, knob, switches, #10110, m,w0,w1,w2,w3,w2,w2)
HOST_fn(button, knob, switches, #01111, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button, knob, switches, #11000, m,CUT16_13 w3, w1, w2, w3, w4, w3)
HOST_fn(button, knob, switches, #10000, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button, knob, switches, #11001, m,CUT16_13 w3, w1, w2, w3, w4, w3)
HOST_fn(button, knob, switches, #10001, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button, knob, switches, #10010, m, w0, w1, w2, w3, w4, INC16(PAD13_16 w1))
HOST_fn(button, knob, switches, #10010, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button, knob, switches, #00101, m, w0, CUT16_13 w5, w2, w3, w4, w5)
HOST_fn(button, knob, switches, #10011, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button, knob, switches, #10100, m,CUT16_13 w3, w1, w2, w3, w4, w3)
HOST_fn(button, knob, switches, #10100, m,w0,w1,w2,w3,w4,w5) ==
HOST_fn
(button, knob, switches, #10101, m, w0, w1, w2, w3, w4, w5)
ADD16 w4(FETCH13 m w0))

HOST_fn(button, knob, switches, #10101, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn(button, knob, switches, #10001, m, w0, w1, w2, w3, w4, w5)

HOST_fn(button, knob, switches, #10110, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn(button, knob, switches, #10111, m, CUT16_13 w3, w1, w2, w3, w4, w5)

HOST_fn(button, knob, switches, #10111, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn(button, knob, switches, #10101, m, w0, w1, w2, w3, w4, w5)
SUB16 w4(FETCH13 m w0))

HOST_fn(button, knob, switches, #11000, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn(button, knob, switches, #10001, m, w0, w1, w2, w3, w4, w5)

HOST_fn(button, knob, switches, #11001, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn(button, knob, switches, #10001, STORE13 w0 w2 m, w0, w1, w2, w3, w4, w5)

By unwinding these equations we can derive a further 26 theorems.

HOST_fn(button, knob, switches, #00000, m, w0, w1, w2, w3, w4, w5) ==
#00000, m, w0, w1, w2, w3, w4, w5

HOST_fn(button, knob, switches, #00001, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn(button, knob, switches, #00001, m, w0, w1, w2, w3, w4, w5)

DEST_TRI16 FLOAT16)

HOST_fn(button, knob, switches, #00010, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn(button, knob, switches, #00011, m, w0, w1, w2, w3, w4, w5) ==
#00000, m, w0, w1, switches, w2, w3, w4, switches

HOST_fn(button, knob, switches, #00100, m, w0, w1, w2, w3, w4, w5) ==
#00000, STORE13 w1 w2 m, w1, w2, w3, w4, w2

HOST_fn(button, knob, switches, #00101, m, w0, w1, w2, w3, w4, w5) ==
#00101, m, w0, w1, w2, w3, w4, w5

HOST_fn(button, knob, switches, #00110, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn
(button,
knob,
switches,
WORDS((VAL3(OPCODE(FETCH13 m w1)) + 10),
  m, w1, w1, w2,
  FETCH13 m w1,
  w4,
  DEST_TRI16 FLOAT16)

HOST_fn(button, knob, switches, #00111, m, w0, w1, w2, w3, w4, w5) ==
#00000, STORE13 w0 w2 m, w0, w1, w2, w3, w4, w2

HOST_fn(button, knob, switches, #01000, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn
(button,
knob,
switches,
WORDS((VAL3(OPCODE(FETCH13 m w0)) + 10),
  m, w0, w1, w2,
  FETCH13 m w0,
  w4,
  DEST_TRI16 FLOAT16)

HOST_fn(button, knob, switches, #01001, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn
(button,
knob,
switches,
WORDS((VAL3(OPCODE w3)) + 10),
  m, w0, w1, w2, w3, w4,
  DEST_TRI16 FLOAT16)

HOST_fn(button, knob, switches, #01010, m, w0, w1, w2, w3, w4, w5) ==
#00000, m, w0, w1, w2, w3, w4, DEST_TRI16 FLOAT16

HOST_fn(button, knob, switches, #01011, m, w0, w1, w2, w3, w4, w5) ==
#00101, m, w0, CUT16-13 w3, w2, w3, w4, w3

HOST_fn(button, knob, switches, #01100, m, w0, w1, w2, w3, w4, w5) ==
HOST_fn
(button,
knob,
switches,
((VAL16 w2) = 0 -> #01011 | #10001),
  m, w0, w1, w2,
w3,
w4,
DEST_TR116 FLOAT16

HOST_fn(button, knob, switches, #01101, m, w0, w1, w2, w3, w4, w5) ==
#00101,
m,
CUT16..13 w3,
INC13 w1,
ADD16 w5(FETCH13 m(CUT16..13 w3)),
w3,
w2,
INC16(PAD13..16 w1)

HOST_fn(button, knob, switches, #01110, m, w0, w1, w2, w3, w4, w5) ==
#00101,
m,
CUT16..13 w3,
INC13 w1,
SUB16 w5(FETCH13 m(CUT16..13 w3)),
w3,
w2,
INC16(PAD13..16 w1)

HOST_fn(button, knob, switches, #01111, m, w0, w1, w2, w3, w4, w5) ==
#00101,
m,
CUT16..13 w3,
INC13 w1,
FETCH13 m(CUT16..13 w3),
w3,
w4,
INC16(PAD13..16 w1)

HOST_fn(button, knob, switches, #10000, m, w0, w1, w2, w3, w4, w5) ==
#00101,
STORE13(CUT16..13 w3)w2 m,
CUT16..13 w3,
INC13 w1,
w2,
w3,
w4,
INC16(PAD13..16 w1)

HOST_fn(button, knob, switches, #10001, m, w0, w1, w2, w3, w4, w5) ==
#00101,m,w0,INC13 w1,w2,w3,w4,INC16(PAD13..16 w1)

HOST_fn(button, knob, switches, #10010, m, w0, w1, w2, w3, w4, w5) ==
#00101,m,w0,CUT16..13 w3,w2,w3,w4,w5

HOST_fn(button, knob, switches, #10011, m, w0, w1, w2, w3, w4, w5) ==
#00101,
m,
CUT16..13 w3,
INC13 w1,
ADD16 w4(FETCH13 m(CUT16..13 w3)),
w3,
w4,
INC16(PAD13..16 w1)

HOST_fn(button, knob, switches, #10100, m, w0, w1, w2, w3, w4, w5) ==
#00101,
m,
w0,
INC13 w1,
ADD16 w4(FETCH13 m w0),
w3,
w4,
INC16(PAD13..16 w1)

HOST_fn(button, knob, switches, #10101, m, w0, w1, w2, w3, w4, w5) ==
#00101,m,w0,INC13 w1,w5,w3,w4,INC16(PAD13..16 w1)

-23-
HOST_fn(button, knob, switches, #10110, m, w0, w1, w2, w3, w4, w5) ==
#00101,
m,
CUT16_13 w3,
INC13 w1,
SUB16 w4(FETCH13 m(CUT16_13 w3)),
w3,
w4,
INC16(PAD13_16 w1)
HOST_fn(button, knob, switches, #10111, m, w0, w1, w2, w3, w4, w5) ==
#00101,
m,
w0,
INC13 w1,
SUB16 w4(FETCH13 m w0),
w3,
w4,
INC16(PAD13_16 w1)
HOST_fn(button, knob, switches, #11000, m, w0, w1, w2, w3, w4, w5) ==
#00101,m,w0,INC13 w1,FETCH13 m w0,w3,w4,INC16(PAD13_16 w1)
HOST_fn(button, knob, switches, #11001, m, w0, w1, w2, w3, w4, w5) ==
#00101,STORE13 w0 w2 m,w0,INC13 w1,w2,w3,w4,INC16(PAD13_16 w1)

From these theorems and HOST UNTIL_EQN_0 and
HOST UNTIL_EQN_5 one can derive theorems HOST_IDLE and
HOST_RUN where:

HOST_IDLE
!
m w0 w1 w2 w3 w4 w5.
until ready do HOST(#00000,m,w0,w1,w2,w3,w4,w5) ==
dev
{knob, button, acc, idle, pc, switches}.
{idle=F, pc=w1, acc=w2};
until
ready
do
HOST
(button ->
((VAL2 knob) = 0 ->
(#00000,m,w0,CUT16_13 switches,w2,w3,w4,switches) |
((VAL2 knob) = 1 ->
(#00000,m,w0,w1,switches,w3,w4,switches) |
((VAL2 knob) = 2 ->
(#00000,STORE13 w1 w2 m,w1,w2,w3,w4,w5) |
(#00101,m,w0,w1,w2,w3,w4,DEST_TRI16 FLOAT16)) |
(#00000,m,w0,w1,w2,w3,w4,DEST_TRI16 FLOAT16))

HOST_RUN
!
m w0 w1 w2 w3 w4 w5.
until ready do HOST(#00101,m,w0,w1,w2,w3,w4,w5) ==
dev
{knob, button, acc, idle, pc, switches}.
{idle=F, pc=w1, acc=w2};
until
ready
do
HOST
(button ->
(#00000,m,w0,w1,w2,w3,w4,DEST_TRI16 FLOAT16) |
((VAL3(OPCODE(FETCH13 m w1))) = 0 ->
(#00000,m,w1,w2,FETCH13 m w1,w4,DEST_TRI16 FLOAT16) |
((VAL3(OPCODE(FETCH13 m w1))) = 1 ->
(#00101,
m,
\( \text{w1,} \)
\( \text{CUT16\_13(FETCH13 m w1).} \)
\( \text{w2,} \)
\( \text{FETCH13 m w1.} \)
\( \text{w4,} \)
\( \text{FETCH13 m w1) |} \)
\( ((\text{VAL3(OPCODE(FETCH13 m w1))) = 2}) \rightarrow \)
\( ((\text{VAL16 w2} = 0}) \rightarrow \)
\( (#00101,} \)
\( m, \)
\( \text{w1,} \)
\( \text{CUT16\_13(FETCH13 m w1),} \)
\( \text{w2,} \)
\( \text{FETCH13 m w1,} \)
\( \text{w4,} \)
\( \text{FETCH13 m w1) |} \)
\( (#00101,m,w1,INC13 w1,w2,FETCH13 m w1,w4,INC16(PAD13\_16 w1)) \rightarrow \)
\( ((\text{VAL3(OPCODE(FETCH13 m w1))) = 3}) \rightarrow \)
\( (#00101,} \)
\( m, \)
\( \text{CUT16\_13(FETCH13 m w1),} \)
\( \text{INC13 w1,} \)
\( \text{ADD16 w2(FETCH13 m(CUT16\_13(FETCH13 m w1)))),} \)
\( \text{FETCH13 m w1,} \)
\( \text{w2,} \)
\( \text{INC16(PAD13\_16 w1)) |} \)
\( ((\text{VAL3(OPCODE(FETCH13 m w1))) = 4}) \rightarrow \)
\( (#00101,} \)
\( m, \)
\( \text{CUT16\_13(FETCH13 m w1),} \)
\( \text{INC13 w1,} \)
\( \text{SUB16 w2(FETCH13 m(CUT16\_13(FETCH13 m w1)))),} \)
\( \text{FETCH13 m w1,} \)
\( \text{w2,} \)
\( \text{INC16(PAD13\_16 w1)) |} \)
\( ((\text{VAL3(OPCODE(FETCH13 m w1))) = 5}) \rightarrow \)
\( (#00101,} \)
\( m, \)
\( \text{CUT16\_13(FETCH13 m w1),} \)
\( \text{INC13 w1,} \)
\( \text{FETCH13 m(CUT16\_13(FETCH13 m w1))),} \)
\( \text{FETCH13 m w1,} \)
\( \text{w4,} \)
\( \text{INC16(PAD13\_16 w1)) |} \)
\( ((\text{VAL3(OPCODE(FETCH13 m w1))) = 6}) \rightarrow \)
\( (#00101,} \)
\( \text{STORE13(CUT16\_13(FETCH13 m w1)) w2 m,} \)
\( \text{CUT16\_13(FETCH13 m w1),} \)
\( \text{INC13 w1,} \)
\( \text{w2,} \)
\( \text{FETCH13 m w1,} \)
\( \text{w4,} \)
\( \text{INC16(PAD13\_16 w1)) |} \)
\( (#00101,} \)
\( m,} \)
\( \text{w1,} \)
\( \text{INC13 w1,} \)
\( \text{w2,} \)
\( \text{FETCH13 m w1,} \)
\( \text{w4,} \)
\( \text{INC16(PAD13\_16 w1))}))})}

Recall that \( \text{COMPUTER\_IMP} \) is defined by:

\( \text{COMPUTER\_IMP}(t,m,w0,w1,w2,w3,w4,w5) = \)
\( \text{until ready do HOST(WORD5(t->0))},m,w0,w1,w2,w3,w4,w5) \)

Using some case analysis, the theorems \( \text{HOST\_UNTIL\_EQN\_0, HOST\_UNTIL\_EQN\_5, HOST\_IDLE} \) and \( \text{HOST\_RUN} \) we can derive:
where:

\[
\text{term 1} = (t \rightarrow \\
(\text{button} \rightarrow \\
((\text{VAL2 knob}) = 0 \rightarrow \\
T \\
((\text{VAL2 knob}) = 1 \rightarrow T \mid ((\text{VAL2 knob}) = 2 \rightarrow T \mid F)))) \mid \\
T) \\
(\text{button} \rightarrow \\
T \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 0 \rightarrow \\
T \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 1 \rightarrow \\
F \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 2 \rightarrow \\
((\text{VAL16 w2}) = 0 \rightarrow F \mid F) \mid \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 3 \rightarrow \\
F \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 4 \rightarrow \\
F \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 5 \rightarrow \\
F \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 6 \rightarrow F \mid F)))))))
\]

and

\[
\text{term 2} = (t \rightarrow \\
(\text{button} \rightarrow \\
((\text{VAL2 knob}) = 0 \rightarrow \\
m \\
((\text{VAL2 knob}) = 1 \rightarrow \\
m \\
((\text{VAL2 knob}) = 2 \rightarrow \text{STORE13 w1 w2 m | m}) \mid \\
m) \\
(\text{button} \rightarrow \\
m \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 0 \rightarrow \\
m \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 1 \rightarrow \\
m \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 2 \rightarrow \\
((\text{VAL16 w2}) = 0 \rightarrow m \mid m) \mid \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 3 \rightarrow \\
m \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 4 \rightarrow \\
m \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 5 \rightarrow \\
m \\
((\text{VAL3 OPCODE(FETCH13 m w1)}) = 6 \rightarrow \text{STORE13(CUT18_13(FETCH13 m w1))w2 m | m})))))
\]

and

\[
\text{term 3} = (t \rightarrow \\
(\text{button} \rightarrow \\
((\text{VAL2 knob}) = 0 \rightarrow \\
w0 \\
((\text{VAL2 knob}) = 1 \rightarrow w0 \mid ((\text{VAL2 knob}) = 2 \rightarrow w1 \mid w0)) \mid \\
w0)
\]

-26-
(\text{button} \rightarrow \\
\text{w}0 \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 0 \rightarrow \text{w}1 \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 1 \rightarrow \text{w}1 \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 2 \rightarrow \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 3 \rightarrow \\
\text{CUT}16\_13(\text{FETCH}13 \text{ m w}1) \ |
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 4 \rightarrow \\
\text{CUT}16\_13(\text{FETCH}13 \text{ m w}1) \ |
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 5 \rightarrow \\
\text{CUT}16\_13(\text{FETCH}13 \text{ m w}1) \ |
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 6 \rightarrow \\
\text{CUT}16\_13(\text{FETCH}13 \text{ m w}1) \ |
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) )})

and
\text{term}4 = (t \rightarrow \\
(\text{button} \rightarrow \\
(\text{VAL}2 \text{ knob}) = 0 \rightarrow \\
\text{CUT}16\_13 \text{ switches} | \\
(\text{VAL}2 \text{ knob}) = 1 \rightarrow \text{w}1 | (\text{VAL}2 \text{ knob}) = 2 \rightarrow \text{w}1 | \text{w}1) | \\
(\text{button} \rightarrow \\
\text{w}1 | \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 0 \rightarrow \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 1 \rightarrow \\
\text{CUT}16\_13(\text{FETCH}13 \text{ m w}1) | \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 2 \rightarrow \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 3 \rightarrow \\
\text{INC}13 \text{ w}1 | \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 4 \rightarrow \\
\text{INC}13 \text{ w}1 | \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 5 \rightarrow \\
\text{INC}13 \text{ w}1 | \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 6 \rightarrow \\
\text{INC}13 \text{ w}1 | 
(\text{INC}13 \text{ w}1))))))))

and
\text{term}5 = (t \rightarrow \\
(\text{button} \rightarrow \\
(\text{VAL}2 \text{ knob}) = 0 \rightarrow \\
\text{w}2 | \\
(\text{VAL}2 \text{ knob}) = 1 \rightarrow \\
\text{switches} | \\
(\text{VAL}2 \text{ knob}) = 2 \rightarrow \text{w}2 | \text{w}2) | \\
(\text{button} \rightarrow \\
\text{w}2 | \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 0 \rightarrow \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 1 \rightarrow \\
\text{w}2 | \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 2 \rightarrow \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 3 \rightarrow \\
\text{ADD}16 \text{ w}2(\text{FETCH}13 \text{ m} | \text{CUT}16\_13(\text{FETCH}13 \text{ m w}1)) | \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 4 \rightarrow \\
\text{SUB}16 \text{ w}2(\text{FETCH}13 \text{ m} | \text{CUT}16\_13(\text{FETCH}13 \text{ m w}1)) | \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 5 \rightarrow \\
\text{FETCH}13 \text{ m} | \text{CUT}16\_13(\text{FETCH}13 \text{ m w}1)) | \\
(\text{VAL}3(\text{OPCODE}(\text{FETCH}13 \text{ m w}1))) = 6 \rightarrow \text{w}2 | \text{w}2)))))))

-27-
and

\[ \text{term6} = (t \rightarrow (\text{button} \rightarrow \begin{cases} ((\text{VAL2 knob}) = 0 \rightarrow w3) \\ ((\text{VAL2 knob}) = 1 \rightarrow w3) \end{cases} | \begin{cases} ((\text{VAL2 knob}) = 2 \rightarrow w3) \end{cases}) | w3)) | (\text{button} \rightarrow \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 0 \rightarrow \begin{cases} \text{FETCH13 m w1} \end{cases} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 1 \rightarrow \begin{cases} \text{FETCH13 m w1} \end{cases} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 2 \rightarrow \begin{cases} \text{FETCH13 m w1} | \text{FETCH13 m w1} \end{cases} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 3 \rightarrow \begin{cases} \text{FETCH13 m w1} | \text{FETCH13 m w1} \end{cases} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 4 \rightarrow \text{FETCH13 m w1} | \text{FETCH13 m w1} \end{cases} \end{cases} \end{cases} \end{cases} \end{cases} \end{cases} \end{cases} \end{cases} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 5 \rightarrow \begin{cases} \text{FETCH13 m w1} | \text{FETCH13 m w1} \end{cases} \end{cases} \end{cases} \end{cases} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 6 \rightarrow \begin{cases} \text{FETCH13 m w1} | \text{FETCH13 m w1} \end{cases} \end{cases} \end{cases} \end{cases} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) \end{cases}) \end{cases}) \right) \]

and

\[ \text{term7} = (t \rightarrow (\text{button} \rightarrow \begin{cases} ((\text{VAL2 knob}) = 0 \rightarrow w4) \\ ((\text{VAL2 knob}) = 1 \rightarrow w4) \end{cases} | \begin{cases} ((\text{VAL2 knob}) = 2 \rightarrow w4) \end{cases}) | w4)) | (\text{button} \rightarrow \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 0 \rightarrow w4) \\ ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 1 \rightarrow w4) \\ ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 2 \rightarrow w4) \\ ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 3 \rightarrow w4) \end{cases} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 4 \rightarrow w4) \\ ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 5 \rightarrow w4) \\ ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 6 \rightarrow w4) \end{cases}) \right) \]

and

\[ \text{term8} = (t \rightarrow (\text{button} \rightarrow \begin{cases} ((\text{VAL2 knob}) = 0 \rightarrow \text{switches}) \\ ((\text{VAL2 knob}) = 1 \rightarrow \text{switches}) \end{cases}) | \begin{cases} ((\text{VAL2 knob}) = 2 \rightarrow w2 | \text{DEST\_TRI16 FLOAT16}) \end{cases}) | \text{DEST\_TRI16 FLOAT16} | (\text{button} \rightarrow \text{DEST\_TRI16 FLOAT16} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 0 \rightarrow \text{DEST\_TRI16 FLOAT16}) \end{cases} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 1 \rightarrow \text{FETCH13 m w1}) \end{cases} | \begin{cases} ((\text{VAL3 OPCODE(FETCH13 m w1)}) = 2 \rightarrow \text{FETCH13 m w1}) \end{cases} \right) \]

\[-28-\]
\[
((\text{VAL}16\ w2) = 0 \rightarrow \text{FETCH}13\ m\ w1 \mid \text{INC}16(\text{PAD}13\_16\ w1)) \mid \\
((\text{VAL}3(\text{OPCODE(FETCH}13\ m\ w1)))) = 3 \rightarrow \\
\text{INC}16(\text{PAD}13\_16\ w1) \mid \\
((\text{VAL}3(\text{OPCODE(FETCH}13\ m\ w1)))) = 4 \rightarrow \\
\text{INC}16(\text{PAD}13\_16\ w1) \mid \\
\text{INC}16(\text{PAD}13\_16\ w1)) \mid \\
((\text{VAL}3(\text{OPCODE(FETCH}13\ m\ w1)))) = 5 \rightarrow \\
\text{INC}16(\text{PAD}13\_16\ w1) \mid \\
((\text{VAL}3(\text{OPCODE(FETCH}13\ m\ w1)))) = 6 \rightarrow \\
\text{INC}16(\text{PAD}13\_16\ w1) \mid \\
\text{INC}16(\text{PAD}13\_16\ w1)))))))
\]

This is in the right form for applying the \textit{UNIQUENESS} rule. Before we can use this we must derive a similar equation for the target machine \textit{COMPUTER}. The required theorem is:

\begin{verbatim}
COMPUTER_EQN

COMPUTER(m,w1,w2,t) ==
  dev\{knob,button,switches,pc,acc, idle\}.
  {pc=w1,acc=w2,idle=t};
  COMPUTER(term2,term4,term5,term1)

\end{verbatim}

Where $\text{term}1$, $\text{term}2$, $\text{term}4$, $\text{term}5$ are the terms occuring in $\text{DERIVED\_HOST\_EQN}$ defined above.

Finally we just execute:

\begin{verbatim}
UNIQUENESS COMPUTER_EQN DERIVED_HOST_EQN
\end{verbatim}

which generates the theorem showing correctness, namely:

\begin{verbatim}
COMPUTER(m,w1,w2,t) == COMPUTER_IMP(t,m,w0,w1,w2,w4,w5)
\end{verbatim}

-29-
Appendix 1: Microassembler and symbolic microcode for the host

In this appendix we give the ML code for the microassembler, followed by
the input used to generate the axioms specifying MICROCODE.
Comments are enclosed between percent signs (i.e. %'s). Although this
code will only be understandable if you are familiar with ML, I hope it
illustrates how simple it is to implement the microassembler in ML.

% The code that follows generates a theory called MUCODE containing
axioms defining the constant MICROCODE which was declared in the
theory HOST %

new_theory 'MUCODE' ;;
new_parent 'HOST' ;;

% The identifiers below give symbolic names to subfields of a
microinstruction %

let rsw = [28]
and umar = [27]
and write = [26]
and read = [25]
and upc = [24]
and rpc = [23]
and uacc = [22]
and racc = [21]
and wr = [20]
and rir = [19]
and warg = [18]
and add = [17]
and inc = [18]
and sub = [16;17]
and rbuf = [15]
and ready = [14]
and idle = [13] ;;

% map2 f (11,12)

maps f in parallel down 11 and 12 %

letrec map2 f (11,12) =
  if null 11 & null 12 then []
  else f(hd 11,hd 12).map2 f (tl 11,tl 12);;

% mk_ADDR [b4; b3; b2; b1; b0] n

gives a list of the positions of bits that are 1 if n is represented in the
5-bit field defined by bit positions b4, b3, b2, b1, b0 %

let mk_ADDR l n =
  if n<0 or n>31 or (not(length l=5))
  then failwith 'mk_A_ADDR'
  else
    mapfilter
    (\n. n=0=>fail|n)
    (map2 $* (map(. t='0'=0|l)(mk_bin rep(5,n)), l));;

% mk_A_ADDR n

gives a list of the positions of bits in the A-address field which are 1
if number n is held in that field

mk_B_ADDR n

gives a list of the positions of bits in the B-address field which are 1
if number n is held in that field %
let mk_A_ADDR = mk_ADDR [12; 11; 10; 8; 5; 0]
and mk_B_ADDR = mk_ADDR [7; 6; 5; 4; 3; 2; 1];

% test_button (n, m)
% gives a list of the bits that are 1 if the address of the next
% microinstruction is (button->n|m) %
let test_button (n, m) =
(0)(mk_A_ADDR m)(mk_B_ADDR n);

% test_acc (n, m)
% gives a list of the bits that are 1 if the address of the next
% microinstruction is (acc=0->n|m) %
let test_acc (n, m) =
(1)(mk_A_ADDR m)(mk_B_ADDR n);

% jump n
% gives a list of the bits that are 1 if the address of the next
% microinstruction is n %
let jump = mk_A_ADDR;

% jump_knob n
% gives a list of the bits that are 1 if the address of the next
% microinstruction is knob+1 %
let jump_knob n =
(1; 0)(mk_A_ADDR n);

% jump_opcode n
% gives a list of the bits that are 1 if the address of the next
% microinstruction is opcode+10 %
let jump_opcode n =
(2)(mk_A_ADDR n);

let word30_ty = "$\text{word30}$";

% mk_micro_ins l
% takes a list of numbers representing the positions of bits in a
% microinstruction that should be 1, and constructs a micro-word (i.e. a
% value of type :word30) with the appropriate bits on %
let mk_micro_ins l =
mk_const
(implode
  ("#".
    map
      (n. mem n t="1" | "0")
    [29; 28; 27; 26; 25; 24; 23; 22; 21; 20; 19; 18; 17; 16; 15; 14; 13; 12; 11; 10;
     9; 8; 7; 6; 5; 4; 3; 2; 1; 0]),
  word30_ty);
% define_microinstruction(addr, cont)

% generates an axiom of the form: |- FETCH5 MICROCODE (WORDS n) == w
% where n is the ROM-address corresponding to the ML integer
% addr, and w is a microinstruction word generated from the list
% cont of bits that are on %

let define_microinstruction (addr, cont) =
  new_axiom
    (concat "MICROCODE" (tok_of_int addr),
     "FETCH5 MICROCODE (WORDS ~{int_to_term addr}) ==
     ~(mk_micro_ins cont)");

% The following generates axioms defining the host's microcode MICROCODE.
% The comments should be interpreted as follows: A -> B is a transfer with source
% and target B (this is sometimes written B: A); MEM(MAR) is the location in
% memory pointed to by the address in MAR. Each element of the list %

map define_microinstruction

[0, ready @ idle @ (test_button(1, 0)); % begin idling cycle %
  1, rsu @ upc @ (jump 0); % decode knob position %
  2, rsw @ upc @ (jump 0); % switches -> PC %
  3, rsw @ uacc @ (jump 0); % switches -> ACC %
  4, rpc @ umar @ (jump 7); % PC -> MAR %
  5, ready @ (test_button(0, 0)); % begin instruction execution %
  6, rpc @ umar @ (jump 8); % PC --> MAR %
  7, acc @ write @ (jump 0); % ACC --> MEM(MAR) %
  8, read @ wir @ (jump 9); % MEM(MAR) --> IR %
  9, (jump_opcode 10); % decode %
  10, (jump 0); % HALT %
  11, rir @ upc @ (jump 8); % JMP IR --> PC %
  12, (test_addr(11, 17)); % JER %
  13, racc @ warg @ (jump 19); % ADD ACC --> ARG %
  14, racc @ warg @ (jump 19); % SUB ACC --> ARG %
  15, rir @ umar @ (jump 24); % LD IR --> MAR %
  16, rir @ umar @ (jump 25); % ST IR --> MAR %
  17, rpc @ inc @ (jump 18); % PCl --> BUF %
  18, rbuf @ upc @ (jump 5); % BUF --> PC %
  19, rir @ umar @ (jump 20); % IR --> MAR %
  20, read @ add @ (jump 21); % ARG+MEM(MAR) --> BUF %
  21, rbuf @ uacc @ (jump 17); % BUF --> ACC %
  22, rir @ umar @ (jump 23); % IR --> MAR %
  23, read @ sub @ (jump 21); % ARG+MEM(MAR) --> BUF %
  24, read @ uacc @ (jump 17); % MEM(MAR) --> ACC %
  25, racc @ write @ (jump 17)];

close theory();
Appendix 2: Derivation of a hard-wired controller

From the theorem CONTROL_EQN given above, and the axioms defining MICROCODE we can derive the following theorems, one for each microinstruction:

CONTROL(MICROCODE, #00000) ==
    dev
    {knob, button, acc, tr, rsw, umar, mementl, wpc, rpc, wacc, race, wir, rir,
     warg, alucntl, rbuf, ready, idle}.
    [rsw=F,
     umar=F,
     mementl=#00,
     wpc=F,
     rpc=F,
     wacc=F,
     race=F,
     wir=F,
     rir=F,
     warg=F,
     alucntl=#00,
     rbuf=F,
     ready=T,
     idle=T];
CONTROL(MICROCODE, (button -> #00001 | #00000))'';

CONTROL(MICROCODE, #00001) ==
    dev
    {knob, button, acc, tr, rsw, umar, mementl, wpc, rpc, wacc, race, wir, rir,
     warg, alucntl, rbuf, ready, idle}.
    [rsw=F,
     umar=F,
     mementl=#00,
     wpc=F,
     rpc=F,
     wacc=F,
     race=F,
     wir=F,
     rir=F,
     warg=F,
     alucntl=#00,
     rbuf=F,
     ready=F,
     idle=F];
CONTROL(MICROCODE, WORDS(((VAL2 knob) + 1) + 1))'';

CONTROL(MICROCODE, #00010) ==
    dev
    {knob, button, acc, tr, rsw, umar, mementl, wpc, rpc, wacc, race, wir, rir,
     warg, alucntl, rbuf, ready, idle}.
    [rsw=T,
     umar=F,
     mementl=#00,
     wpc=T,
     rpc=F,
     wacc=F,
     race=F,
     wir=F,
     rir=F,
     warg=F,
     alucntl=#00,
     rbuf=F,
     ready=F,
     idle=F];
CONTROL(MICROCODE, #00000)'';

CONTROL(MICROCODE, #00011) ==
    dev
CONTROL(MICROCODE, #00000) =

dev
[knob, button, acc, ir, rsrw, uwar, memcntl, upc, rpc, wacc, racc, wir, rir,
 warg, aluncnl, rbuf, ready, idle].
[rsrw=F,
 uwar=F,
 memcntl=#00,
 upc=F,
 rpc=F,
 wacc=F,
 racc=F,
 wir=F,
 rir=F,
 warg=F,
 aluncnl=#00,
 rbuf=F,
 ready=F,
 idle=F];
CONTROL(MICROCODE, #00001) =

CONTROL(MICROCODE, #00010) =

dev
[knob, button, acc, ir, rsrw, uwar, memcntl, upc, rpc, wacc, racc, wir, rir,
 warg, aluncnl, rbuf, ready, idle].
[rsrw=F,
 uwar=F,
 memcntl=#00,
 upc=F,
 rpc=F,
 wacc=F,
 racc=F,
 wir=F,
 rir=F,
 warg=F,
 aluncnl=#00,
 rbuf=F,
 ready=F,
 idle=F];
CONTROL(MICROCODE, #00011) =

CONTROL(MICROCODE, #00100) =

CONTROL(MICROCODE, #00101) =

dev
[knob, button, acc, ir, rsrw, uwar, memcntl, upc, rpc, wacc, racc, wir, rir,
 warg, aluncnl, rbuf, ready, idle].
[rsrw=F,
 uwar=F,
 memcntl=#00,
 upc=F,
 rpc=F,
 wacc=F,
 racc=F,
 wir=F,
 rir=F,
 warg=F,
 aluncnl=#00,
 rbuf=F,
 ready=F,
 idle=F];
CONTROL(MICROCODE, #00110) =

CONTROL(MICROCODE, #00111) =

dev
[knob, button, acc, ir, rsrw, uwar, memcntl, upc, rpc, wacc, racc, wir, rir,
 warg, aluncnl, rbuf, ready, idle].
[rsrw=F,
 uwar=F,
 memcntl=#00,
 upc=F,
 rpc=F,
 wacc=F,
 racc=F,
 wir=F,
 rir=F,
 warg=F,
alunctl=@00,
rbuf=F,
ready=F,
idle=F;
CONTROL(MICROCODE,#01000)"

CONTROL(MICROCODE,#00111) ==
dev
{knot, button, acc, ir, rsu, umar, memcntl, upc, rpc, wace, race, wir, rir,
warg, alunctl, rbuf, ready, idle}.
{rsu=F,
 umar=F,
mencntl=#10,
upc=F,
rpc=F,
wace=F,
race=F,
wir=F,
rir=F,
warg=F,
alunctl=#00,
rbuf=F,
ready=F,
idle=F;
CONTROL(MICROCODE,#00000)"

CONTROL(MICROCODE,#01000) ==
dev
{knot, button, acc, ir, rsu, umar, memcntl, upc, rpc, wace, race, wir, rir,
warg, alunctl, rbuf, ready, idle}.
{rsu=F,
 umar=F,
mencntl=#01,
upc=F,
rpc=F,
wace=F,
race=F,
wir=F,
rir=F,
warg=F,
alunctl=#00,
rbuf=F,
ready=F,
idle=F;
CONTROL(MICROCODE,#01001)"

CONTROL(MICROCODE,#01001) ==
dev
{knot, button, acc, ir, rsu, umar, memcntl, upc, rpc, wace, race, wir, rir,
warg, alunctl, rbuf, ready, idle}.
{rsu=F,
 umar=F,
mencntl=#00,
upc=F,
rpc=F,
wace=F,
race=F,
wir=F,
rir=F,
warg=F,
alunctl=#00,
rbuf=F,
ready=F,
idle=F;
CONTROL(MICROCODE,WORDS((VAL3(OPCODE ir)) + 10))"

CONTROL(MICROCODE,#01010) ==
dev
{knot, button, acc, ir, rsu, umar, memcntl, upc, rpc, wace, race, wir, rir,
warg, alunctl, rbuf, ready, idle}.
{rsu=F,
 umar=F,
memcntl=\$00,
upc=F,
rpc=F,
uacc=F,
racc=F,
wir=F,
rr=F,
var=F,
alucntl=\$00,
rbuf=F,
ready=F,
idle=F;
CONTROL(MICROCODE, \$000000)

CONTROL(MICROCODE, \$01011) ==
dev
[knob, button, acc, ir, rsw, umar, memcntl, upc, rpc, uacc, racc, wir, rrr, var, alucntl, rbuf, ready, idle].
[rsw=F,
umar=F,
memcntl=\$00,
upc=T,
rpc=F,
uacc=F,
racc=F,
wir=F,
rr=F,
var=F,
alucntl=\$00,
rbuf=F,
ready=F,
idle=F];
CONTROL(MICROCODE, \$00101)

CONTROL(MICROCODE, \$01100) ==
dev
[knob, button, acc, ir, rsw, umar, memcntl, upc, rpc, uacc, racc, wir, rrr, var, alucntl, rbuf, ready, idle].
[rsw=F,
umar=F,
memcntl=\$00,
upc=F,
rpc=F,
uacc=F,
racc=F,
wir=F,
rr=F,
var=F,
alucntl=\$00,
rbuf=F,
ready=F,
idle=F];
CONTROL(MICROCODE, ((VAL16 acc) = 0 -> \$01011 | \$10001))

CONTROL(MICROCODE, \$01101) ==
dev
[knob, button, acc, ir, rsw, umar, memcntl, upc, rpc, uacc, racc, wir, rrr, var, alucntl, rbuf, ready, idle].
[rsw=F,
umar=F,
memcntl=\$00,
upc=F,
rpc=F,
uacc=F,
racc=T,
wir=F,
rr=F,
var=F,
alucntl=\$00,
rbuf=F,
ready=F,
idle=F];

-36-
CONTROL(MICROCODE, #10011) =

dev
{kob, button, acc, ir, rsw, umar, momentl, wpc, rpc, wacc, race, wir, rir, warg, alucntl, rbuf, ready, idle}.
{rsr=F, umar=F, momentl=#00, wpc=F, rpc=F, wacc=F, race=T, wir=F, rir=F, warg=T, alucntl=#00, rbuf=F, ready=F, idle=F};
CONTROL(MICROCODE, #10110) =

dev
{kob, button, acc, ir, rsw, umar, momentl, wpc, rpc, wacc, race, wir, rir, warg, alucntl, rbuf, ready, idle}.
{rsr=F, umar=T, momentl=#00, wpc=F, rpc=F, wacc=F, race=F, wir=F, rir=T, warg=F, alucntl=#00, rbuf=F, ready=F, idle=F};
CONTROL(MICROCODE, #11000) =

dev
{kob, button, acc, ir, rsw, umar, momentl, wpc, rpc, wacc, race, wir, rir, warg, alucntl, rbuf, ready, idle}.
{rsr=F, umar=F, momentl=#00, wpc=F, rpc=F, wacc=F, race=F, wir=F, rir=F, warg=F, alucntl=#00, rbuf=F, ready=F, idle=F};
CONTROL(MICROCODE, #11001) =

dev
{kob, button, acc, ir, rsw, umar, momentl, wpc, rpc, wacc, race, wir, rir, warg, alucntl, rbuf, ready, idle}.
{rsr=F, umar=F, momentl=#00, wpc=F, rpc=F, wacc=F, race=F, wir=F, rir=F, warg=F, alucntl=#00, rbuf=F, ready=F, idle=F};
CONTROL(MICROCODE, #10001) =

dev
{kob, button, acc, ir, rsw, umar, momentl, wpc, rpc, wacc, race, wir, rir, warg, alucntl, rbuf, ready, idle}.
{rsr=F, umar=F, momentl=#00, wpc=F, rpc=F, wacc=F, race=F, wir=F, rir=F, warg=F, alucntl=#00, rbuf=F, ready=F, idle=F}.
racc=F,
wire=F,
rtir=F,
warg=F,
alunhl=#01,
rbuf=F,
ready=F,
idle=F};
CONTROL(MICROCODE,#10010);";

CONTROL(MICROCODE,#10010) ==
dev
{knob,button,acc,ir,rsw,umar,memcnll,upc,rpc,wacc,racc,wir,rtir,
warg,alunhl,rbuf,ready,idle}.
{rswe=F,
umae=F,
mempnl=#00,
upce=T,
rpc=F,
waee=F,
racc=F,
wir=F,
rtir=F,
warg=F,
alunhl=#00,
rbuf=F,
ready=F,
idle=F};
CONTROL(MICROCODE,#00101);";

CONTROL(MICROCODE,#00111) ==
dev
{knob,button,acc,ir,rsw,umar,memcnll,upc,rpc,wacc,racc,wir,rtir,
warg,alunhl,rbuf,ready,idle}.
{rswe=F,
umae=T,
mempnl=#00,
upce=F,
rpc=F,
waee=F,
racc=F,
wir=F,
rtir=T,
warg=F,
alunhl=#00,
rbuf=F,
ready=F,
idle=F};
CONTROL(MICROCODE,#10100);";

CONTROL(MICROCODE,#10100) ==
dev
{knob,button,acc,ir,rsw,umar,memcnll,upc,rpc,wacc,racc,wir,rtir,
warg,alunhl,rbuf,ready,idle}.
{rswe=F,
umae=F,
mempnl=#01,
upce=F,
rpc=F,
waee=F,
racc=F,
wir=F,
rtir=F,
warg=F,
alunhl=#10,
rbuf=F,
ready=F,
idle=F};
CONTROL(MICROCODE,#10101);";

CONTROL(MICROCODE,#10101) ==
dev

-38-
CONTROL(MICROCODE, #10001) =
dev
[knob, button, acc, ir, rsu, unar, memcnt0, wpc, rpe, wacc, race, wir, rir, warg, alucntl, rbuf, ready, idle].
{rsu=F, unar=F, memcnt0=00,
 wpc=F, rpe=F, wacc=F, race=F, wir=F, rir=F, warg=F, alucntl=00, rbuf=F, ready=F, idle=F};
CONTROL(MICROCODE, #10110) =
dev
[knob, button, acc, ir, rsu, unar, memcnt0, wpc, rpe, wacc, race, wir, rir, warg, alucntl, rbuf, ready, idle].
{rsu=F, unar=F, memcnt0=00,
 wpc=F, rpe=F, wacc=F, race=F, wir=F, rir=F, warg=F, alucntl=00, rbuf=F, ready=F, idle=F};
CONTROL(MICROCODE, #10111) =
dev
[knob, button, acc, ir, rsu, unar, memcnt0, wpc, rpe, wacc, race, wir, rir, warg, alucntl, rbuf, ready, idle].
{rsu=F, unar=F, memcnt0=01,
CONTROL(MICROCODE, #11000) =
dev
[knob, button, acc, ir, rsu, unar, memcnt0, wpc, rpe, wacc, race, wir, rir, warg, alucntl, rbuf, ready, idle].
{rsu=F, unar=F, memcnt0=01,
 wpc=F, rpe=F, wacc=F, race=F, wir=F, rir=F, warg=F,
We can use these theorems to design a controller which just has a 5-bit register (to hold a state counter) instead of a ROM with microcode. The structure of this hard-wired controller is:

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT_STATE</td>
<td>STATE_REG(w)</td>
<td>DECODE_STATE</td>
</tr>
<tr>
<td>next_state</td>
<td>s</td>
<td>rsw  unmar</td>
</tr>
<tr>
<td></td>
<td></td>
<td>memcntl upc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rpc  wacc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>racw wir rir</td>
</tr>
<tr>
<td></td>
<td></td>
<td>aluncl rbuf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ready idle</td>
</tr>
</tbody>
</table>
```

Each state corresponds to a microinstruction address, and the decode logic `DECODE_STATE` and sequencing logic `NEXT_STATE` mimic the effect of the microcode as exhibited in the 26 theorems above. For abstractness let us take the state to be a number (rather than, say, a value of 01 type `word5`). The specification of `STATE_REG` is:

```
STATE_REG n == dev[next state, s].{s=n};STATE_REG next state
```

The purely combinational sequencing logic can be read off from the 26
Theorems:

\[
\text{\texttt{NEXT\_STATE}} ==
\begin{aligned}
&\text{dev}\{s,\text{button},\text{knob},\text{tr},\text{acc},\text{next\_state}\}, \\
&\text{next\_state} = \begin{cases}
  s=0 & \rightarrow (\text{button} \rightarrow 1|0) \\
  s=1 & \rightarrow (\text{VAL2 knob})+2 \\
  s=2 & \rightarrow 0 \\
  s=3 & \rightarrow 0 \\
  s=4 & \rightarrow 7 \\
  s=5 & \rightarrow (\text{button} \rightarrow 0|6) \\
  s=6 & \rightarrow 8 \\
  s=7 & \rightarrow 0 \\
  s=8 & \rightarrow 0 \\
  s=9 & \rightarrow (\text{VAL3 (OPCODE tr)} \rightarrow 1|0) \\
  s=10 & \rightarrow 0 \\
  s=11 & \rightarrow 6 \\
  s=12 & \rightarrow ((\text{VAL16 acc}) = 0 \rightarrow 11|17) \\
  s=13 & \rightarrow 19 \\
  s=14 & \rightarrow 22 \\
  s=15 & \rightarrow 24 \\
  s=16 & \rightarrow 25 \\
  s=17 & \rightarrow 18 \\
  s=18 & \rightarrow 5 \\
  s=19 & \rightarrow 20 \\
  s=20 & \rightarrow 21 \\
  s=21 & \rightarrow 23 \\
  s=22 & \rightarrow 23 \\
  s=23 & \rightarrow 21 \\
  ? & 
\end{cases}
\end{aligned}
\]

\texttt{NEXT\_STATE}

The purely combinational decode logic can also be constructed directly from the 28 theorems:

\[
\text{\texttt{DECODE\_STATE}} ==
\begin{aligned}
&\text{dev} \{s,\text{rsu,unar,memc1,upc,rpc,wa,acc,rac,wr,rr,wa,rw}\}, \\
&\text{rsu} = (s=3) \lor (s=6), \\
&\text{unar} = (s=4) \lor (s=7) \lor (s=10) \lor (s=15) \lor (s=16) \lor (s=18), \\
&\text{memc1} = (s=17) \lor (s=19) \lor (s=20) \lor (s=21) \lor (s=23), \\
&\text{upc} = (s=2) \lor (s=11), \\
&\text{rpc} = (s=14) \lor (s=24) \lor (s=25), \\
&\text{wa} = (s=5), \\
&\text{rac} = (s=12) \lor (s=13), \\
&\text{wr} = (s=8), \\
&\text{rr} = (s=11) \lor (s=16) \lor (s=19) \lor (s=22), \\
&\text{wa} = (s=13) \lor (s=14), \\
&\text{rsu} = (s=23) \lor (s=21) \lor (s=20) \lor (s=17) \lor (s=16) \lor (s=14), \\
&\text{wb} = (s=2) \lor (s=11), \\
&\text{ready} = (s=0) \lor (s=5), \\
&\text{idle} = (s=0)
\end{aligned}
\]

\texttt{DECODE\_STATE}

Using these devices we can then define a controller \texttt{CNTL} by:

\[
\text{\texttt{CNTL}} n == \text{\texttt{\{\{\text{\texttt{STATE\_REG}} n | \text{\texttt{NEXT\_STATE}} | \text{\texttt{DECODE\_STATE}}\}}\} hide\{s,\text{next\_state}\}}
\]

Although \texttt{CNTL} was derived directly from the microcode, perhaps we made a silly error when typing in the definitions. To check we didn't we can prove that \texttt{CNTL} satisfies the same equations as the microprogrammed controller \texttt{CONTROL}. First we use \texttt{EXPAND\_IMP} to derive a behaviour equation for \texttt{CNTL n}, then we simplify the cases \(n=0, n=1, n=2, \ldots, n=25\) separately to derive:
CNSL VAL5 #00000
\[\text{dev} \{\text{button, knob, ir, acc, rsu, umar, memcntl, upc, rpc, wace, race, wir, rir, warn, alunctl, rbuf, ready, idle}\}.
\{rsu=F, umar=F, memcntl=\#00, upc=F, rpc=F, wace=F, race=F, wir=F, rir=F, warn=F, alunctl=\#00, rbuf=F, ready=T, idle=T\};
CNSL (button \rightarrow 1 | 0)

CNSL VAL5 #00001
\[\text{dev} \{\text{button, knob, ir, acc, rsu, umar, memcntl, upc, rpc, wace, race, wir, rir, warn, alunctl, rbuf, ready, idle}\}.
\{rsu=F, umar=F, memcntl=\#00, upc=F, rpc=F, wace=F, race=F, wir=F, rir=F, warn=F, alunctl=\#00, rbuf=F, ready=F, idle=F\};
CNSL ((VAL2 knob) + 2)

CNSL VAL5 #00010
\[\text{dev} \{\text{button, knob, ir, acc, rsu, umar, memcntl, upc, rpc, wace, race, wir, rir, warn, alunctl, rbuf, ready, idle}\}.
\{rsu=T, umar=F, memcntl=\#00, upc=T, rpc=F, wace=T, race=F, wir=F, rir=F, warn=F, alunctl=\#00, rbuf=F, ready=F, idle=F\};
CNSL 0

CNSL VAL5 #00011
\[\text{dev} \{\text{button, knob, ir, acc, rsu, umar, memcntl, upc, rpc, wace, race, wir, rir, warn, alunctl, rbuf, ready, idle}\}.
\{rsu=T, umar=F, memcntl=\#00, upc=F, rpc=F, wace=F, race=F, wir=F, rir=F, warn=F, alunctl=\#00,\]

-42-
rbuf=F,
ready=F,
idle=F;
CNTL 0

CNTL(VALS $00100)$ ==

$\text{dev}$ button, knob, ir, acc, rsu, umar, memcntl, wpc, rpc, uacc, race, wir, rir,
warq, alunctl, rbuf, ready, idle$\}$.

{rsu=F,
umar=F,
memcntl=$00$,
wpc=F,
rpc=F,
wacc=F,
racq=F,
wrtr=F,
rit=F,
warq=F,
alunctl=$00$,
rbuf=F,
ready=F,
idle=F$} ;
CNTL 7

CNTL(VALS $00101)$ ==

$\text{dev}$ button, knob, ir, acc, rsu, umar, memcntl, wpc, rpc, uacc, race, wir, rir,
warq, alunctl, rbuf, ready, idle$\}$.

{rsu=F,
umar=F,
memcntl=$00$,
wpc=F,
rpc=F,
wacc=F,
racq=F,
wrtr=F,
rit=F,
warq=F,
alunctl=$00$,
rbuf=F,
ready=F,
idle=T$} ;
CNTL(button $\rightarrow 0 \mid 6$)

CNTL(VALS $00110)$ ==

$\text{dev}$ button, knob, ir, acc, rsu, umar, memcntl, wpc, rpc, uacc, race, wir, rir,
warq, alunctl, rbuf, ready, idle$\}$.

{rsu=F,
umar=F,
memcntl=$00$,
wpc=F,
rpc=T,
wacc=F,
racq=F,
wrtr=F,
rit=F,
warq=F,
alunctl=$00$,
rbuf=F,
ready=F,
idle=F$} ;
CNTL 8

CNTL(VALS $00111)$ ==

$\text{dev}$ button, knob, ir, acc, rsu, umar, memcntl, wpc, rpc, uacc, race, wir, rir,
warq, alunctl, rbuf, ready, idle$\}$.

{rsu=F,
umar=F,
memcntl=$00$,
wpc=F,
rpc=F,
wacc=F,
racq=T$} .

-43-
CCTL (VAL5 #01000) ==
  dev[button, knob, ir, acc, rsw, umar, memcntl, upc, rpc, wacc, racc, wir, rir, warg, aluncll, rbuf, ready, idle].
  rsu=F,
  umar=F,
  memcntl=#00,
  upc=F,
  rpc=F,
  wacc=F,
  racc=F,
  wir=F,
  rir=F,
  warg=F,
  aluncll=#00,
  rbuf=F,
  ready=F,
  idle=F;
CCTL 0

CCTL (VAL5 #01001) ==
  dev[button, knob, ir, acc, rsw, umar, memcntl, upc, rpc, wacc, racc, wir, rir, warg, aluncll, rbuf, ready, idle].
  rsu=F,
  umar=F,
  memcntl=#00,
  upc=F,
  rpc=F,
  wacc=F,
  racc=F,
  wir=F,
  rir=F,
  warg=F,
  aluncll=#00,
  rbuf=F,
  ready=F,
  idle=F;
CCTL((VAL3(OPCODE ir)) + 10)

CCTL (VAL5 #01010) ==
  dev[button, knob, ir, acc, rsw, umar, memcntl, upc, rpc, wacc, racc, wir, rir, warg, aluncll, rbuf, ready, idle].
  rsu=F,
  umar=F,
  memcntl=#00,
  upc=F,
  rpc=F,
  wacc=F,
  racc=F,
  wir=F,
  rir=F,
  warg=F,
  aluncll=#00,
  rbuf=F,
  ready=F,
  idle=F;
CCTL 0

CCTL (VAL5 #01011) ==
  dev[button, knob, ir, acc, rsw, umar, memcntl, upc, rpc, wacc, racc, wir, rir, warg, aluncll, rbuf, ready, idle].
  rsu=F,
  umar=F,
  memcntl=#00,
  upc=F,
ctl(val5 #0100) ==
  dev{button, knob, ir, acc, rsu, umar, memcntl, wpc, rpc, wacc, racc, wir, rir, warg, aluentl, rbuf, ready, idle}.
  [rsu=F,
   umar=F,
   memcntl=$00,
   wpc=F,
   rpc=F,
   wacc=F,
   racc=F,
   wir=F,
   rir=F,
   warg=F,
   aluentl=$00,
   rbuf=F,
   ready=F,
   idle=F};
ctl(val16 acc = 0 -> 11 | 17)

ctl(val5 #0110) ==
  dev{button, knob, ir, acc, rsu, umar, memcntl, wpc, rpc, wacc, racc, wir, rir,
   warg, aluentl, rbuf, ready, idle}.
  [rsu=F,
   umar=F,
   memcntl=$00,
   wpc=F,
   rpc=F,
   wacc=F,
   racc=F,
   wir=F,
   rir=F,
   warg=F,
   aluentl=$00,
   rbuf=F,
   ready=F,
   idle=F};
ctl 19

ctl(val5 #0110) ==
  dev{button, knob, ir, acc, rsu, umar, memcntl, wpc, rpc, wacc, racc, wir, rir,
   warg, aluentl, rbuf, ready, idle}.
  [rsu=F,
   umar=F,
   memcntl=$00,
   wpc=F,
   rpc=T,
   wacc=F,
   racc=T,
   wir=F,
   rir=F,
   warg=F,
   aluentl=$00,
   rbuf=F,
   ready=F,
   idle=F};
ctl 22

ctl(val5 #0111) ==
  dev{button, knob, ir, acc, rsu, umar, memcntl, wpc, rpc, wacc, racc, wir, rir,
CNTL(VAL5 #10000) ==
dev{button, knob, ir, acc, rsw, umar, memcntl, upc, rpc, wacc, race, wir, rir, warg, aluncll, rbuf, ready, idle}.
{rsw=F, umar=T, memcntl=#00, upc=F, rpc=F, wacc=F, race=F, wir=F, rir=T, warg=F, aluncll=#00, rbuf=F, ready=F, idle=F}.
CNTL 24

CNTL(VAL5 #10001) ==
dev{button, knob, ir, acc, rsw, umar, memcntl, upc, rpc, wacc, race, wir, rir, warg, aluncll, rbuf, ready, idle}.
{rsw=F, umar=F, memcntl=#10, upc=F, rpc=F, wacc=F, race=F, wir=F, rir=F, warg=F, aluncll=#01, rbuf=F, ready=F, idle=F}.
CNTL 25

CNTL(VAL5 #10010) ==
dev{button, knob, ir, acc, rsw, umar, memcntl, upc, rpc, wacc, race, wir, rir, warg, aluncll, rbuf, ready, idle}.
{rsw=F, umar=F, memcntl=#00, upc=T, rpc=F, wacc=F, race=F, wir=F, rir=F, warg=F, aluncll=#00, rbuf=T, ready=F, idle=F}.

-46-
CNTL. 5

-CNTL (VAL5 #10011) ==
  dev\{button, knob, ir, acc, rsu, unar, mencoder, upc, rpe, wace, racc, wir, rir,
  warg, aluncnt, rbuf, ready, idle\}.
  \{rsuw=F,\n    unar=F,\n    mencoder=#00,\n    upc=F,\n    rpe=F,\n    wace=F,\n    racc=F,\n    wir=F,\n    rir=F,\n    warg=F,\n    aluncnt=#00,\n    rbuf=F,\n    ready=F,\n    idle=F\};
CNTL 20

-CNTL (VAL5 #10100) ==
  dev\{button, knob, ir, acc, rsu, unar, mencoder, upc, rpe, wace, racc, wir, rir,
  warg, aluncnt, rbuf, ready, idle\}.
  \{rsuw=F,\n    unar=F,\n    mencoder=#01,\n    upc=F,\n    rpe=F,\n    wace=F,\n    racc=F,\n    wir=F,\n    rir=F,\n    warg=F,\n    aluncnt=#10,\n    rbuf=F,\n    ready=F,\n    idle=F\};
CNTL 21

-CNTL (VAL5 #10101) ==
  dev\{button, knob, ir, acc, rsu, unar, mencoder, upc, rpe, wace, racc, wir, rir,
  warg, aluncnt, rbuf, ready, idle\}.
  \{rsuw=F,\n    unar=F,\n    mencoder=#00,\n    upc=F,\n    rpe=F,\n    wace=T,\n    racc=F,\n    wir=F,\n    rir=F,\n    warg=F,\n    aluncnt=#00,\n    rbuf=T,\n    ready=F,\n    idle=F\};
CNTL 17

-CNTL (VAL5 #10110) ==
  dev\{button, knob, ir, acc, rsu, unar, mencoder, upc, rpe, wace, racc, wir, rir,
  warg, aluncnt, rbuf, ready, idle\}.
  \{rsuw=F,\n    unar=T,\n    mencoder=#00,\n    upc=F,\n    rpe=F,\n    wace=F,\n    racc=F,\n    wir=F,\n    rir=F,\n    warg=F,\n    -47-
CNTL (VAL5 #1011) ==
  dev{button, knob, ir, acc, rsw, umar, memcntl, upc, rpc, wacc, race, wir, rir,
    warg, aluncll, rbuf, ready, idle}.
  {rswe=F,
   umar=F,
   memcntl=#01,
   upc=F,
   rpc=F,
   wacc=F,
   race=F,
   wir=F,
   rir=F,
   warg=F,
   aluncll=#11,
   rbuf=F,
   ready=F,
   idle=F};
  CNTL 23

CNTL (VAL5 #11000) ==
  dev{button, knob, ir, acc, rsw, umar, memcntl, upc, rpc, wacc, race, wir, rir,
    warg, aluncll, rbuf, ready, idle}.
  {rswe=F,
   umar=F,
   memcntl=#01,
   upc=F,
   rpc=F,
   wacc=F,
   race=F,
   wir=F,
   rir=F,
   warg=F,
   aluncll=#00,
   rbuf=F,
   ready=F,
   idle=F};
  CNTL 21

CNTL (VAL5 #11001) ==
  dev{button, knob, ir, acc, rsw, umar, memcntl, upc, rpc, wacc, race, wir, rir,
    warg, aluncll, rbuf, ready, idle}.
  {rswe=F,
   umar=F,
   memcntl=#10,
   upc=F,
   rpc=F,
   wacc=F,
   race=F,
   wir=F,
   rir=F,
   warg=F,
   aluncll=#00,
   rbuf=F,
   ready=F,
   idle=F};
  CNTL 17

One can now compare these equations for CNTL with the previous set of 26 equations derived for CONTROL. It will be seen that they are equivalent.
If one wanted more security one could then go on to prove that:

\[
\text{COMPUTER}(m, w1, w2, t) == \\
\text{until \ ready} \\
\text{do \ [[ \ CNTL(t \to 0, 5) \ | \ DATA(m, w0, w1, w2, w3, w4, w5) \ ]] \\
\text{hide} \{ rsw, unwar, mmenfil, upo, rpo, wqco, race, \\
wir, rir, warg, alucnf1, rbuf, tr\}
\]

One could either do this directly, or by extending \textit{MICROCODE} so that:

\[
!w. \ \text{CONTROL}(\text{MICROCODE}, w) == \text{CNTL}(\text{VAL5} \ w)
\]

If this held, then the original proof of correctness would not need to be redone. Unfortunately, because we have not specified what happens when the microinstruction address lies in the range 26 to 31, this equation does not follow from the axioms given above.

References

[Cardelli]

[Gordon1]
M. Gordon. \textit{A Model of register Transfer Systems with Applications to Microcode and VLSI Correctness}. Internal Report CSR-82-81, Dept. of Computer Science, University of Edinburgh, 1981

[Gordon2]