Instruction Set Completeness Theorem: Concept, Relevance, Proof, and Example for Processor Architecture

William F. Gilreath (wfgilreath@yahoo.com)
March 2017

Abstract

The Instruction Set Completeness Theorem is first formally defined and discussed in the seminal work on one-instruction set computing—the book *Computer Architecture: A Minimalist Perspective*.

Yet the original formalism of the Instruction Set Completeness Theorem did not provide a definitive, explicit mathematical proof of completeness, analyze both singular and plural instruction sets that were either complete or incomplete, nor examine the significance of the theorem to computer architecture instruction sets.

A mathematical proof of correctness shows the equivalence of the Instruction Set Completeness Theorem to a Turing machine, a hypothetical model of computation, and thereby establishes the mathematical truth of the Instruction Set Completeness Theorem. With a more detailed examination of the Instruction Set Completeness Theorem develops several surprising conclusions for both the instruction set completeness theorem, and the instruction sets for a computer architecture.

Keywords: Church-Turing Thesis, computation model, computer architecture, computer organization, instruction set, instruction set completeness theorem, plural instruction set, processor architecture, one-instruction computer, one-instruction set computer, singular instruction set, Turing machine, von Neumann architecture

1. Introduction

The Instruction Set Completeness Theorem is formally stated by Gilreath and Laplante [Gilr 2003c] by:

> "Any processor with a complete instruction set must be stateful of computation. That is, a means to alter the state or next state transition by explicit operation or by implicit computation is needed to deterministically affect processor state."

The five principles of the Instruction Set Completeness Theorem are stated by Gilreath and Laplante. Later, an important point is made of the equivalence of a Turing machine to an instruction set—indirectly stating the equivalence of the Instruction Set Completeness Theorem to a Turing machine. The statement by Gilreath and Laplante [Gilr 2003d] is:

> "An instruction set is complete if it is equivalent to a single tape Turing machine."

This statement is correct, but no mathematical proof is given by the authors. Showing the isomorphism between the Instruction Set Completeness Theorem and a Turing machine is a formal mathematical proof by demonstrating the correspondence.

1.1 Instruction Set Completeness Theorem

The Instruction Set Completeness Theorem is the formal definition for a general proposition of the principles for the completeness of an instruction set of a processor architecture. There are five overall principles for the Instruction Set Completeness Theorem, with a distinction made for four of the principles of implicitness and explicitness of a similar principle.

The five principles are declared by Gilreath and Laplante [Gilr 2003c] and are organized around explicit or implicit nature of the principle. The principles for the Instruction Set Completeness Theorem are:

1. Statefulness - The instruction set must be stateful by state and a next state function.
2. Explicit State Change - There must be an instruction to directly and explicitly modify state.
3. Explicit Next State Change - There must be an instruction to directly and explicitly modify next state function.
4. Implicit State Change - There must be an instruction to indirectly and implicitly modify state.
5. Implicit Next State Change - There must be an instruction to indirectly and implicitly modify next state function.

The State Change Principle and the Next State Change Principle are divided into explicit and implicit operation.

1.2 Simplify the Instruction Set Completeness Theorem

The Instruction Set Completeness Theorem has five principles, but four of the principles are organized around the explicit or implicit nature of the underlying principle.

1.2.1 Explicit and Implicit Principles

The Instruction Set Completeness Theorem is for a processor architecture—a computer system. Each instruction of a computer is either explicit or implicit in operation. The contradistinction between explicit and implicit is:

1. Explicit – operation intentionally by user
2. Implicit – operation automatically by processor

Simply put, an explicit principle is for operations intentional by the user, and the implicit principle is for operations automatic by the processor. Either way, an operation is governed by the same principle—whether it is intentional by the user, or automatic by the processor.

1.2.2 Simplify Four Principles to Two Principles

These four principles of the Instruction Set Completeness Theorem are then actually two principles when the distinction of explicit and implicit is removed. The two resulting principles are:

1. State Change Principle – There must be an instruction to modify state.
2. Next State Change Principle – There must be an instruction to modify next state function.

1.2.3 Reduction to Three Principles

The statefulness principle of the Instruction Set Completeness Theorem is neither explicit or implicit thus is not simplified. Combining the three principles is the core of the Instruction Set Completeness Theorem. The three fundamental principles are:

1. Statefulness Principle – The instruction set must be stateful by state and a next state function.
2. State Change Principle – There must be an instruction to modify state.
3. Next State Change Principle – There must be an instruction to modify next state function.

1.3 Significance of Instruction Set Completeness Theorem

The Instruction Set Completeness Theorem is relevant to processor architecture because a question asked by Gilreath and Laplante [Gilr 2003a] is:

> "What is the minimal functionality of an instruction needed to perform all other kinds of operations in a processor?"

and later state [Gilr 2003a]:

> "The question of instruction set completeness, is to determine what is the minimal instruction set functionality necessary to have effective computability."

Thus "completeness" of an instruction set for a processor architecture is for computation, or the capability for computing tractable problems with a particular processor architecture. A processor architecture that follows the Instruction Set Completeness Theorem is by definition a complete instruction set—effectively computing any tractable problem.

Considering the question in the converse or opposite, an incomplete instruction set is by design unable to compute all tractable computation problems. An incomplete instruction set limits a processor architecture to a subset of all possible problems that are computable.

The relevance of the Instruction Set Completeness Theorem is more germane to a singular instruction set, a one-instruction set computer, more so than with a plural instruction set. A computer with only one instruction must follow all the principles of the Instruction Set Completeness Theorem for the singular instruction set effectively computable.

2. Proof

The mathematical proof is simply to show an isomorphism or mapping between the Instruction Set Completeness Theorem and a Turing machine. A Turing machine is formally defined, the Instruction Set Completeness Theorem is simplified, and then the correspondence between a Turing machine and the Instruction Set Completeness Theorem is given.

2.1 Turing Machine

The Turing machine was developed by Alan Turing in 1936 [Turi 1936] who created the "automatic machine" or "a-machine" as Turing originally called his construct, for a mathematical solution and proof of computability—what is effectively computable. Savage [Sava 1997] describes the utility of a Turing machine:

> "The Turing machine (TM) is believed to be the most general computational model that can be devised (the Church-Turing thesis). Despite many attempts, no computational model has yet been introduced that can perform computations impossible on a Turing machine."

A Turing machine is elegant in simplicity and operation, so ideal theoretically, but impractical for application in the real world.

2.1.1 Turing Machine Description

Gilreath and Laplante [Gilr 2003b] describe a Turing machine as:

> "The Turing machine is a simple model consisting of an infinite tape and read/write unit that can write a single symbol at a time to the tape or read a symbol from the tape. While the symbols can be from any alphabet, a simple binary alphabet is sufficient. In any case, the read/write head moves one position either to the right or left depending on the symbol read on the tape at the position immediately below the read/write head. The read/write unit may also output a new symbol to the tape (overwriting what was previously there)."

The important point made by Gilreath and Laplante [Gilr 2003b] about a Turing machine is:

> "The most important observation is that anything computable by a Turing machine is computable by a more sophisticated and complex computer or processor."

Thus, logically a Turing machine is an ideal mathematical abstraction for the proof of the Instruction Set Completeness Theorem.

2.1.2 Turing Machine Definition

For the mathematical proof of the Instruction Set Completeness Theorem, a comprehensive definition of a Turing machine is first required. The definitions for a Turing machine given by

Linz [Linz 2011], Rosen [Rose 2012], and Hopcroft *et al.* [Hopc 2006] are utilized in the overall definition.

The formal definition for a Turing machine is described with a 7-tuple consisting of the elements:

$$M = \{Q, \Sigma, \Gamma, f, q_0, \beta, F\}$$

where the elements are defined as:

Q	– The finite set of states.
Σ	– The set of symbols in the alphabet.
Γ	– The set of symbols in the tape alphabet where $\Gamma \subseteq \Sigma$.
f	– The transition function where $f : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.
q_0	– The initial or starting state where $q_0 \in Q$.
β	– The blank symbol where $\beta \in \Gamma$ where $\beta \notin \Sigma$.
F	– The set of final or accepting states where $F \subseteq Q$.

The blank symbol β is a tape symbol thus an output symbol, but not an input symbol.

The transition function f is a partial function from the 2-tuple of state and input symbol ($q_x \in Q$, $s_i \in \Gamma$) to the 3-tuple of state, output symbol, and direction ($q_y \in Q$, $s_o \in \Gamma \cup \{\beta\}$, $d \in \{L, R\}$) where $f : (q_x, s_i) \rightarrow (q_y, s_o, d)$. For the transition function f some (state, symbol) pairs of the domain the partial function are possibly undefined—an implicit halt state for Turing machine M.

2.2 Turing Machine Equal to Instruction Set Completeness Theorem

A Turing machine is equal to the Instruction Set Completeness Theorem, or the principles of the Instruction Set Completeness Theorem. But, the Instruction Set Completeness Theorem is redundant in duplicating principles for inclusiveness.

2.2.1 Instruction Set Completeness Theorem Simplified Principles

The simplified Instruction Set Completeness Theorem principles, without explicit and implicit operation, are:

1. Statefulness Principle
2. State Change Principle
3. Next State Change Principle

2.2.2 Map Instruction Set Completeness Theorem Principles onto Turing Machine

Each of the three principles has a corresponding element in the definition of a Turing machine. Thus for each Instruction Set Completeness Theorem Principle, there is an element or elements from the 7-tuple formal definition of a Turing machine.

1. Statefulness Principle – M_s where $M_s \subseteq M$. The statefulness principle involves two sets of elements elements: state $\{Q, q_0, F\}$ of a Turing machine M, and also next state function $f = f_{Mc} \cup f_{Mn}$ of a Turing machine M. The statefulness principle elements are $M_s = \{Q, q_0, F, f\}$.

2. State Change Principle – M_c where $M_c \subseteq M$. The state change principle involves two sets of elements: input symbols $\{\Sigma\}$ of a Turing machine M, and also the next state function f of a Turing machine M. The state change principle elements are $M_c = \{f_{Mc}, \Sigma\}$.

   The state transition function f_{Mc} reads an input symbol $s_i \in \Sigma$ from the tape in a state q_x from the tape, and transitions to another state q_y:

   $f_{Mc}: (q_x, s_i) \rightarrow (q_y)$

   When the state $q_y \notin \{F\} \cup \{Q\}$ the resulting state is undefined.

3. Next State Change Principle – M_n where $M_n \subseteq M$. The next state change principle involves two sets of elements: output symbols $\{\Gamma, \beta\}$ of a Turing machine M, and also next state change principle f of a Turing machine M. The next state change principle elements are $M_n = \{f_{Mn}, \Gamma, \beta\}$.

   The next state transition function f_{Mn} is in state q_x to write a symbol s_o in $\{\Gamma, \beta\}$ to the tape, and then shift the tape in a direction $d = \{L, R\}$ to the left or right.

   $f_{Mn}: (q_x, s_i) \rightarrow (s_o, d)$ where $d = \{L, R\}$.

2.2.3 Constructing a Turing Machine

The original Turing machine M can be constructed using the set of elements from the three principles of the Instruction Set Completeness Theorem. This requires the transition function f, and the set of elements for symbols and state.

2.2.3.1 Transition Functions

The union of the image or range for the functions of state change f_{Mc}, next state change f_{Mn}, result in the overall transition function f for Turing machine M. Thus for the two functions:

$$f_{Mc}: (q_x, s_i) \rightarrow (q_y) \wedge f_{Mn}: (q_x,s_i) \rightarrow (s_o, d).$$

Both functions have the same domain, but map to a different range. The union of both functions f_{Mc} and f_{Mn} in the range is then the overall state change function f:

$$f : (q_x, s_i) \rightarrow (q_y) \cup (s_o, d) \equiv f : (q_x, s_i) \rightarrow (q_y, s_o, d).$$

Thus the overall state change function f is then:

$$f : (q_x, s_i) \rightarrow (q_y, s_o, d).$$

2.2.3.2 State Change, Next State Change, Statefulness Elements

The union of the subsets of the Turing machine state change M_c, next state M_n is the subset of state change and next state or M_{cn}:

$$M_{cn} = \{f_{Mc,} \Gamma, \Sigma\} \cup \{f_{Mn}, \Gamma, \beta\} = \{f, \Gamma, \beta, \Sigma\}$$

The set of elements $M_{cn} \subseteq M$, thus is a Turing machine M, but stateless.

The transition function f is defined for symbols the blank symbol β, and the set of tape symbols $\Gamma \subseteq \Sigma$ the set of symbols in the alphabet.

With the union with the set of elements for statefulness M_s, with the set of elements for state change and next state M_{cn} forms a stateful Turing machine M:

$$M_{cns} = \{f, \Gamma, \beta, \Sigma\} \cup \{Q, q_0, F, f\} = \{f, \Gamma, \beta, \Sigma, Q, q_0, F\} \equiv M$$

Thus the overall union of the three sets of elements: M_s, M_n, M_c, results in the 7-tuple of elements that define a Turing machine M. So each of the three principles of the Instruction Set Completeness Theorem correspond to elements in the definition of a Turing Machine.

$\therefore$ All three principles of the Instruction Set Completeness Theorem together form an overall 7-tuple of elements that define a Turing machine.

Since by the Church-Turing Thesis, Horsten [Hors 2006] states:

> "…states that every function on the natural numbers that is effectively computable, is computable by a Turing machine."

More simply stated, anything computable is computable by means of a Turing machine; therefore anything computable is computable on a processor architecture that uses the principles of the Instruction Set Completeness Theorem.

Q.E.D.

2.3 Summary of Mathematical Proof

The Instruction Set Completeness Theorem is shown to directly correspond to a Turing machine. Therefore, the Instruction Set Completeness Theorem is the same as a Turing machine, thus by the Church-Turing Thesis is equivalent computationally.

The three principles of the Instruction Set Completeness Theorem, in the context of corresponding to a Turing machine are then:

1. statefulness – have state, a memory of previous operation in current operation.
2. state change – read input data to compute next state
3. next state change – write output data for computation

An important consideration is that the Instruction Set Completeness Theorem is not specific to a processor architecture, or any intrinsic features for a processor architecture.

3. Instruction Sets

Instruction sets for a processor architecture can be either plural or singular. A plural instruction set consists of more than one instruction, and a singular instruction set has only one instruction. The cardinality of the instruction sets is either one instruction or many (greater than one) instructions.

More simply put, a plural instruction set is for a non-one-instruction set computer (NOISC), whereas a singular instruction set is for a one-instruction set computer (OISC). The concept of instruction set completeness is applicable to both plural and singular instruction sets of a processor architecture. A complete instruction set for a processor instruction, singular or plural, follows all the principles of the Instruction Set Completeness Theorem.

3.1 Complete Instruction Sets

Consider a complete instruction set for a processor architecture, such a processor can be evaluated for each of the principles of the Instruction Set Completeness Theorem for both plural and singular instruction sets.

3.1.1 Example Plural Instruction Set Processor Architecture

The example for a plural instruction set is the RISC I processor architecture [Patt 1981] which has thirty-one instructions and one addressing mode. The RISC I processor architecture is complete, therefore meets the principles for the Instruction Set Completeness Theorem. The RISC I instruction set is:

Assembly Language Definition for RISC I

Instr.	*Operands*	*Comments*	
ADD	*Rs,S2,Rd*	*Rd ← Rs + S2*	*integer add*
ADDC	*Rs,S2,Rd*	*Rd ← Rs + S2 + carry*	*add with carry*
SUB	*Rs,S2,Rd*	*Rd ← Rs - S2*	*integer subtract*
SUBC	*Rs,S2,Rd*	*Rd ← Rs - S2 - carry*	*subtract with carry*
SUBR	*Rs,S2,Rd*	*Rd ← S2 - Rs*	*integer subtract*
SUBCR	*Rs,S2,Rd*	*Rd ← S2 - Rs - carry*	*subtract with carry*
AND	*Rs,S2,Rd*	*Rd ← Rs & S2*	*logical AND*
OR	*Rs,S2,Rd*	*Rd ← Rs | S2*	*logical OR*
XOR	*Rs,S2,Rd*	*Rd ← Rs xor S2*	*logical EXCLUSIVE OR*
SLL	*Rs,S2,Rd*	*Rd ← Rs shifted by S2*	*shift left*
SRL	*Rs,S2,Rd*	*Rd ← Rs shifted by S2*	*shift right logical*
SRA	*Rs,S2,Rd*	*Rd ← Rs shifted by S2*	*shift right arithmetic*
LDL	*(Rx)S2,Rd*	*Rd ← M[Rx+S2]*	*load long*
LDSU	*(Rx)S2,Rd*	*Rd ← M[Rx+S2]*	*load short unsigned*
LDSS	*(Rx)S2,Rd*	*Rd ← M[Rx+S2]*	*load short signed*
LDBU	*(Rx)S2,Rd*	*Rd ← M[Rx+S2]*	*load byte unsigned*
LDBS	*(Rx)S2,Rd*	*Rd ← M[Rx+S2]*	*load byte signed*
STL	*(Rx)S2,Rm*	*M[Rx+S2] ← Rm*	*store long*
STS	*(Rx)S2,Rm*	*M[Rx+S2] ← Rm*	*store short*
STB	*(Rx)S2,Rm*	*M[Rx+S2] ← Rm*	*store byte*

JMP	*CON,S2(Rx)*	*pc ← Rx + S2*	*conditional jump*
JMPR	*CON,Y*	*pc ← pc + Y*	*conditional relative*
CALL	*S2(Rx),Rd*	*CWP--; Rd ← pc, next*	*call reg. indexed*
		pc ← Rx + S2	*and change window*
CALLR	*Y,Rd*	*CWP--; Rd ← pc, next*	*call relative*
		pc ← pc + Y	*and change window*
RET	*(Rx)S2*	*pc ← Rx + S2, next CWP++*	*return, change window*
RETINT	*(Rx)S2*	*pc ← Rx + S2, next CWP++*	*also enable interrupts*
CALLINT	*Rd*	*CWP--; Rd ← last pc*	*also disable interrupts*
LDHI	*Y,Rd*	*Rd<31:13> ← Y; Rd<12:0> ← 0*	*load immediate high*
GTLPC	*Rd*	*Rd ← last pc*	*to restart delayed jump*
GETPSW	*Rd*	*Rd ← PSW*	*read status word*
PUTPSW	*Rm*	*PSW ← Rm*	*set status word*

Table of Instruction Set for RISC I Processor Architecture

For each of the Instruction Set Completeness Theorem principles, the processor architecture instruction set has a corresponding subset of instructions within the instruction set that follow the Instruction Set Completeness Theorem. Likewise, the processor architecture has registers in the register set follow the Instruction Set Completeness Theorem. In summary, for the three principles, one principle involves the register set, and the other two principles involve subsets of the instruction set.

3.1.2 Plural Instruction Set Completeness Theorem RISC I Instructions

The processor architecture RISC I follows each of the Instruction Set Completeness Theorem principles is:

1. Statefulness Principle – The instruction set must be stateful by state and a next state function.

   For the RISC 1, state and next state function that make the processor architecture stateful are, for state the Processor Status Word (PSW) register, and for the next state function the Program Counter (PC) register.

2. State Change Principle – There must be an instruction to modify state.

   The state change is from the Processor Status Word register, instructions that affect the PSW register relate to the state change. For the RISC 1, the instructions are:

   GETPSW, PUTPSW explicit affect for PSW register.

3. Next State Change Principle – There must be an instruction to modify next state function.

   The next state change is the Program Counter register, instructions that affect the PC register relate to the next state change. For the RISC 1, the instructions are:

   CALL, CALLR, CALLINT, RET, RETINT explicit for PC register.

> The instructions in the instruction set: JMP, JMPR implicit for PC, implicit for PSW in effect, these instructions impact state change and next state change principles. The arithmetic instructions for addition and subtraction indirectly affect the carry bit in the PSW register...thus the state change principle.

The other instructions in the instruction set can possibly follow the Instruction Set Completeness Theorem, but are not necessarily.

3.1.3 Example Singular Instruction Set Processor Architecture

There are many complete singular instructions that are used in a one-instruction computer processor architecture—a variety of one instructions [Esol 2017a]. The example used is the subtract and branch on result less than or equal to zero [Esold 2017d] singular instruction or “subleq” instruction.

3.1.3.1 SUBLEQ Instruction

The description of SUBLEQ or “subleq” instruction is given in C programming language [Kern 1988] pseudocode by [Nürn 2003] as:

```
subleq a, b, c:

    *b-= *a;
    if (*b <= 0) goto c;
```

Code in C for the SUBLEQ Instruction

The subleq singular instruction is a variation on the subtract and branch if negative instruction (SBN) [Esol 2017c] and reverse subtract and skip on borrow (RSSB) instruction [Esol 2017b].

Some other complete singular instructions are bitwise logical such as NOR [Demi 2012]. Another singular instruction, the Move instruction, is complete on a transport triggered architecture (TTA) [Corp 1997].

3.1.3.2 SUBLEQ Instruction Operations

The subleq instruction consists of two operations, the subtraction operation, and the jump conditionally operation. In assembly pseudocode with both operations forming the subleq instruction:

```
subleq opa, opu, addr:

    sub opa, opu, opu ; opu = opa - opu
    jle opu, #0, addr ; if(result <= #0) goto addr
```

Code in Assembly for the SUBLEQ Instruction

3.1.4 Singular Instruction Set Completeness Theorem SUBLEQ

With a single instruction, the principles of the instruction set completeness theorem relate to the operations within the single instruction and the registers more so than the instruction set.

1. Statefulness Principle – The instruction set must be stateful by state and a next state function.

   The state and next state function that make the processor architecture stateful are, for state the status register (SR), and for the next state function the Program Counter (PC) register.

2. State Change Principle – There must be an instruction to modify state.

   The state change is from the status register, the operation that affects the status register relate to the state change. The subtraction operation indirectly affects the status register.

3. Next State Change Principle – There must be an instruction to modify next state function.

   The next state change is the program counter register, the operation that affects the PC register is the conditional jump operation.

3.2 Incomplete Instruction Sets

An incomplete instruction set does not follow one of the three principles of the Instruction Set Completeness Theorem. Starting with the complete instruction sets of the RISC I and SUBLEQ processor architectures, both can be modified for incompleteness with a specific Instruction Set Completeness Theorem principle.

3.2.1 Incomplete Plural Instruction Set

The modification to the complete plural instruction set for the RISC I processor architecture involves removing instructions from the instruction set, and then removing registers from the processor architecture.

The registers removed are:

1. PC the program counter register.
2. PSW the program status word register (i.e., status register).

The instructions removed are:

1. GETPSW, PUTPSW instructions for PSW register.
2. CALL, CALLR, CALLINT, RET, RETINT, GTLPC instructions for PC register.
3. JMP, JMPR for PC register and PSW register.

For all the principles of the Instruction Set Completeness Theorem the removal of the instructions and registers affects each principle:

1. stateful

Remove the PC register and PSW register.

2. explicit modify state

Remove the GETPSW, PUTPSW instructions.

3. explicit modify next state function

Remove the CALL, CALLR, CALLINT, RET, RETINT, GTLPC instructions.

4. implicit modify state

Remove the JMP, JMPR instructions.

5. implicit modify next state function

Remove the JMP, JMPR instructions.

The removal of the JMP, JMPR instructions affects both the implicit modify state function and implicit modify next state function principles. Removing the other instructions affects the principles of explicit modify state function and explicit modify next state function.

The removal of the registers from the processor architecture impacts the statefulness principle, and without state, the other principles of the Instruction Set Completeness Theorem.

The removal of the PSW register requires that the PSW register be a parameter for each instruction, both passed in and out with the instruction. The removal of the PC register requires that a condition, a next address for the condition, and the next address otherwise as a parameter passed into the instruction.

The loss of either register impacts global state of the processor architecture, and as both an input and output parameter of each instruction, localizes state. If the same memory address is used for both the input and output parameter for each register, then there is global state.

The instructions that utilize the carry, ADDC, SUBC, SUBCR, are redundant with the PSW register (that contains the carry bit) as a parameter for input and output, so are refactored from the RISC I instruction set. The carry bit is through the PSW parameter passed into and from each instruction.

The resulting incomplete instruction set with the removed registers for the RISC I is summarized in the table.

Incomplete Assembly Language Definition for RISC I

Instr.	Operands	Comments	
ADD	PSW_{in},Rs,S2,Rd,PSW_{out},CON,PC_{cond},PC_{next}	Rd $\leftarrow$ Rs + S2	integer add
SUB	PSW_{in},Rs,S2,Rd,PSW_{out},CON,PC_{cond},PC_{next}	Rd $\leftarrow$ Rs - S2	integer subtract
SUBR	PSW_{in},Rs,S2,Rd,PSW_{out},CON,PC_{cond},PC_{next}	Rd $\leftarrow$ S2 - Rs	integer subtract
AND	Rs,S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ Rs & S2	logical AND
OR	Rs,S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ Rs | S2	logical OR
XOR	Rs,S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ Rs xor S2	logical EXCLUSIVE OR
SLL	Rs,S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ Rs shifted by S2	shift left
SRL	Rs,S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ Rs shifted by S2	shift right logical
SRA	Rs,S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ Rs shifted by S2	shift right arithmetic
LDL	(Rx)S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ M[Rx+S2]	load long
LDSU	(Rx)S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ M[Rx+S2]	load short unsigned
LDSS	(Rx)S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ M[Rx+S2]	load short signed
LDBU	(Rx)S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ M[Rx+S2]	load byte unsigned
LDBS	(Rx)S2,Rd,CON,PC_{cond},PC_{next}	Rd $\leftarrow$ M[Rx+S2]	load byte signed
STL	(Rx)S2,Rm,CON,PC_{cond},PC_{next}	M[Rx+S2] $\leftarrow$ Rm	store long
STS	(Rx)S2,Rm,CON,PC_{cond},PC_{next}	M[Rx+S2] $\leftarrow$ Rm	store short
STB	(Rx)S2,Rm,CON,PC_{cond},PC_{next}	M[Rx+S2] $\leftarrow$ Rm	store byte
LDHI	Y,Rd,CON,PC_{cond},PC_{next}	Rd<31:13> $\leftarrow$ Y; Rd<12:0> $\leftarrow$ 0	load immediate high to restart delayed jump

Table of Incomplete Instruction Set for RISC I Processor Architecture

The resulting instruction set has only eighteen instructions, the load-store instructions for data movement into the processor architecture, and the operational instructions for arithmetic and logical operations.

3.2.2 Incomplete Singular Instruction Set

An incomplete single instruction set cannot be affected by the removal of instructions from the instruction set. With only one instruction in the instruction set, there are no other instructions to remove to impact the principles of the Instruction Set Completeness Theorem.

subleq sr_{in}, opa, opu, addr, sr_{out}, pc_{next}:

```
    sub sr_in, opa, opu, opu, sr_out, pc_next
    jle sr_in, opu, #0, addr, sr_out, pc_next
```

Code in Assembly for the Incomplete SUBLEQ Instruction

But, by removing the stateful registers and making them a parameter to the one instruction, the instruction set is incomplete. Statefulness is local to each instruction, although it is possible by using the same memory address to have global state.

4. Analysis

The Instruction Set Completeness Theorem is commensurate with the Church-Turing Thesis, only specifically for the attributes of a processor architecture and the instruction set for such a processor architecture.

4.1 Instruction Set Completeness Theorem

A processor architecture has operations, or instructions that are explicit and implicit in execution. Hence, the Instruction Set Completeness Theorem makes a distinction in such operations in the principles, but in the mathematical proof such a distinction is unnecessary.

4.2 Mapping between Turing Machine and Instruction Set Completeness Theorem

The mathematical proof for the truth of the Instruction Set Completeness Theorem is demonstrating the correspondence between the principles of the Instruction Set Completeness Theorem and the elements in the definition of a Turing machine, and then taking the elements and constructing the original definition of a Turing machine. This is a mapping between the two abstractions, but showing that one is equivalent to the other—the two are synonymous. A processor architecture that follows the Instruction Set Completeness Theorem is capable of any computation that a Turing machine can compute.

4.3 Processor Instruction Sets

Both plural and singular instruction sets must be complete for computability. For both instruction sets, simply removing statefulness is enough to make an instruction set incomplete—although this requires a change in the processor architecture and the instruction set.

4.3.1 Instruction Sets

Modifying the instruction set by removing instructions that allow both explicit and implicit operation on the program counter (PC) and/or processor status word (PSW) registers removes completeness by not following a principle of the Instruction Set Completeness Theorem. The emphasis is on instruction sets, but the modified or removed instructions in the instruction set impact state and statefulness of the processor architecture.

4.3.2 Serial Processor

For the von Neumann processor architecture [vonN 1998], statefulness is from the program counter register (PC) and status register (SR). The principles of the Instruction Set Completeness Theorem follow the requirement for statefulness, and then modifying processor architecture state, and the function that modifies the state.

Thus, for an alternative, serial non-von Neumann processor architecture, the elements of the processor architecture for statefulness are the paramount, and then the instructions in the instruction set that impact the elements of statefulness. Example of non-Von ???

4.3.3 Parallel Processor

For a parallel processor (the processor architecture has been implicitly a serial processor) the Instruction Set Completeness Theorem applies equally. As parallel computer architecture involves multiple or many processors that communicate, each individual processor must follow the Instruction Set Completeness Theorem.

Simply stated, the overall union of each processor in the parallel processor architecture must follow the Instruction Set Completeness Theorem for the overall parallel system to follow the Instruction Set Completeness Theorem. For a parallel processor architecture, there is a mathematical union or functional composition of each individual processor architecture. However, the Instruction Set Completeness Theorem equally applies to a parallel processor as it does to a serial processor.

5. Future Work

The original definition and discussion of the Instruction Set Completeness Theorem creates the groundwork, and this research monograph further examines and develops the Instruction Set Completeness Theorem. The most significant development is a formal mathematical proof of the Instruction Set Completeness Theorem. But, despite the amount of endeavour, there is still future work around the Instruction Set Completeness Theorem.

Future work on the Instruction Set Completeness Theorem involves three specific areas:

1. Examination of the Instruction Set Completeness Theorem for a serial non-von Neumann computer architecture.

2. Examination of the Instruction Set Completeness Theorem for a parallel computer architecture.

3. Develop incomplete instruction sets for each of the principles of the Instruction Set Completeness Theorem.

Such efforts evaluate the Instruction Set Completeness Theorem more widely across different processor architectures, and also by creating deliberately incomplete instruction sets for a hypothetical processor architecture.

6. Conclusion

The Instruction Set Completeness Theorem principles directly correspond to elements of a Turing machine. The Instruction Set Completeness Theorem is equivalent to a Turing machine, and thus is a restatement of effective computability from the Church-Turing thesis.

The emphasis of the Instruction Set Completeness Theorem is for a processor architecture with state that is stateful, and with instructions in an instruction set—either plural or singular instruction sets. The Instruction Set Completeness Theorem is not specific to any processor architecture.

The illustrative example is from the von Neumann computer architecture, where state is from the program counter and status registers. For other alternative processor architectures, the stateful elements of the processor architecture are significant, and the instructions in the instruction set that modify both state and the change of state—the Instruction Set Completeness Theorem.

Consequently, the Instruction Set Completeness Theorem is not only a list of requirements for completeness for effectively computable—but also a model of what is significant about a processor architecture—any processor architecture that is complete by the Church-Turing Thesis.

7. References

1. [Corp 1997] Corporaal, Henk. *Microprocessor Architectures: From VLIW to TTA*, John Wiley and Sons Inc., Hoboken, New Jersey, 1997.

2. [Demi 2012] Demin, Alexander. "The NOR Machine: Build a CPU with Only One Instruction," *Pragmatic Bookshelf, PragPub*, Number 33, March 2012, pp. 6 – 24, https://pragprog.com/magazines/2012-03/the-nor-machine.

3. [Esol 2017a] "OISC - Esolang," https://esolangs.org/wiki/OISC, Accessed Feburary 17, 2017.

4. [Esol 2017b] "RSSB - Esolang," https://esolangs.org/wiki/RSSB, Accessed Feburary 17, 2017.

5. [Esol 2017c] "SBN - Esolang," https://esolangs.org/wiki/SBN, Accessed Feburary 17, 2017.

6. [Esol 2017d] "Subleq - Esolang," https://esolangs.org/wiki/subleq, Accessed Feburary 17, 2017.

7. [Gilr 2003a] Gilreath, William F. and Laplante, Philip A. *Computer Architecture: A Minimalist Perspective*, Springer Science+Business Media, New York, New York, 2003, p. 55.

8. [Gilr 2003b] *ibid*, p. 56.

9. [Gilr 2003c] *ibid*, p. 63.

10. [Gilr 2003d] *ibid*, p. 66.

11. [Hors 2006] Horsten, Leon. "Formalizing Church's Thesis," *Church's Thesis After 70 Years*, Transaction Books, Piscataway, New Jersey, 2006, p 253.

12. [Hopc 2006] Hopcroft, John E., Motwani, Rajeev, and Ullman, Jeffrey D. Introduction to Automata Theory, Languages, and Computation, 3rd edition. Pearson Education, Boston, Massachusetts, 2007, p. 319.

13. [Kern 1988] Kernighan, Brian W. and Ritchie, Dennis M. *The C Programming Language*, 2nd edition, Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1988.

14. [Linz 2011] Linz, Peter. *An Introduction to Formal Languages and Automata*, 5th edition, Jones and Bartlett Learning, Sudbury, Massachusetts, 2011, p. 223.

15. [Nürn 2003] Nürnberg, Peter J., Wiil, Uffe K., and Hicks, David L., "A Grand Unified Theory for Structural Computing," *Metainformatics, International Symposium*, MIS 2003, Graz, Austria, September 17-20, p. 4, 2003.

16. [Patt 1981] Patterson, David A. and Sequin, Carlo H. "RISC I: A Reduced Instruction Set VLSI Computer," *Proceedings of the 8th International Symposium on Computer Architecture (ISCA '81)*, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 443-457, 1981.

17. [Rose 2012] Rosen, Kenneth H. *Discrete Mathematics and Its Applications*, 7th edition. McGraw-Hill, New York, New York, 2012, p. 889.

18. [Sava 1997] Savage, John E. *Models of Computation: Exploring the Power of Computing*, Addison-Wesley Longman Publishing Company, Inc. Boston, Massachusetts, 1997, p. 209.

19. [Turi 1936] Turing, A. "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of London Mathematical Society*, Number 2, Volume 42, 1936, pp. 230–236. Correction, *ibid*, Volume 43, 1937, pp. 544–546.

20. [vonN 1998] von Neumann, John, "First Draft of a Report on the EDVAC," *IEEE Annals of the History of Computing*, Volume 15, Number 4, 1993, pp. 27-75.