Discovering Nontrivial and Functional Behavior in Register Machines

Anthony Joseph

Past Student
University of Technology Sydney
anthonyjoseph.nks@gmail.com

Nontrivial and functional behavior in register machines is examined. Register machines are simple implementations of modern information and communications technology and provide a computationally simple vehicle for investigating examples of nontrivial and functional behavior. They also provide opportunities for optimizing information and communication technologies to use fewer resources or perform functions more quickly.

A simple two-register, four-instruction register machine was analyzed using soft and hard analytical techniques. Examples of nontrivial and functional behavior were identified by observing two-register, four-instruction register machines with various initial conditions. These register machines were identified by an exhaustive search of all possible register machine configurations meeting a particular definition. A subsequent investigation into the randomness in register machine components involved a frequency analysis, comparing program counter and register values against the discrete uniform distribution.

It is possible to observe examples of cyclical and conditional behavior, register-dependent and register-independent behavior, randomness in the register machine's program counter and registers, and foundation arithmetic functions. Further analysis of this register machine configuration yields opportunities for synthesizing multiple functions into a single register machine and optimizing functional register machines by brute-force testing all possible register machines.

1. Introduction

Register machines are implementations of a simple computing device that perform operations on a fixed set of data registers. According to Stephen Wolfram, register machines are "specifically designed to be very simple idealizations of present-day computers" [1, p. 97]. Therefore, all modern information and communication technologies use register machines of various implementations to store, access, and manipulate data. Register machines are comprised of three related components: a register, a program, and a program counter.

1.1 Register

A register, or set of registers with a constant width, stores an encoded value. The horizontal axis (1 to 8) indicates the value of the register and the vertical axis (1 to 31) indicates the number of instructions executed in the program. Red-colored register values indicate that a value higher than the register's width is currently stored in that register. This is similar to the concept of "arithmetic overflow," where the result of a calculation is greater than the register that stores or represents the data. Unlike some physical implementations of a register machine such as a microcontroller, microprocessor, or other low-level hardware devices, exceeding a register's capacity will not clear the register's value or halt the register machine. Figure 1 shows an example of a single register.

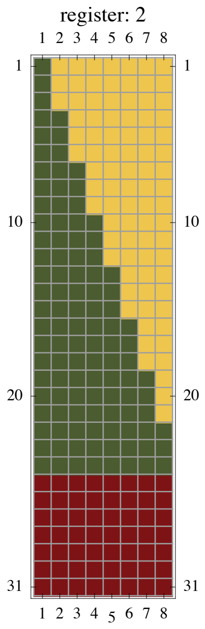

Figure 1. Register example.

1.2 Program

A program describes the behavior of the register machine. Programs are a set of instructions that operate on registers. There are a wide variety of implementations of instructions that are available on different hardware platforms. Wolfram uses a simple implementation with two instructions [1, p. 97]:

- an increment operation, which increases the value of the register by one and then executes the next instruction in the program, and

- a decrement-jump operation, which decreases the value of a register by one and "jumps" to another instruction in the program. If the register value is 0, then the program executes the next instruction. The decrement-jump operation is the main operation that yields nontrivial behavior.

In both cases, an instruction contains:

- the current instruction identified as an integer,
- the next instruction to be executed,
- the register that is being manipulated, and
- the modifier being applied (i.e., 1 for an increment operation and -1 for a decrement operation).

The author used a Minsky register machine implementation [2], which is available via the Wolfram Demonstrations Project [3]. This implementation is based on Wolfram's implementation [1, p. 98] with the added instruction of a "halted instruction," where a program would stop execution when it completed execution.

A no-operation "NOP" can be included but is not considered within this study, as "any NOP instruction can be removed from the formal description of the underlying Minsky register machine without altering its function" [2]. Therefore, including the halted instruction will allow the discovery of any possible arithmetic or logical functions as the register machine should halt after it has performed its function.

Unfortunately, there are no standard representations for register machines. Wolfram [1, p. 98] used as a representation a sequence of squares with directional arrowheads, arrows, and color to define the instruction and whether an increment or decrement jump operation is performed—the destination register for a decrement-jump operation and the register to be operated on, respectively. The author has used a Mealy finite state machine representation to describe these programs as described in Figure 2 using the nomenclature in Figures 3 to 6.

1.3 Program Counter

A program counter indicates the instruction that is executed at a particular time. The horizontal axis (1 to 4) indicates the instruction executed and the vertical axis (1 to 21) indicates the number of instructions executed in the program. Figure 7 shows an example of a program counter.

An enumeration is used to describe all unique register machine programs and is described in Appendix A. All of the register machines specified in this paper can be simulated using [3].

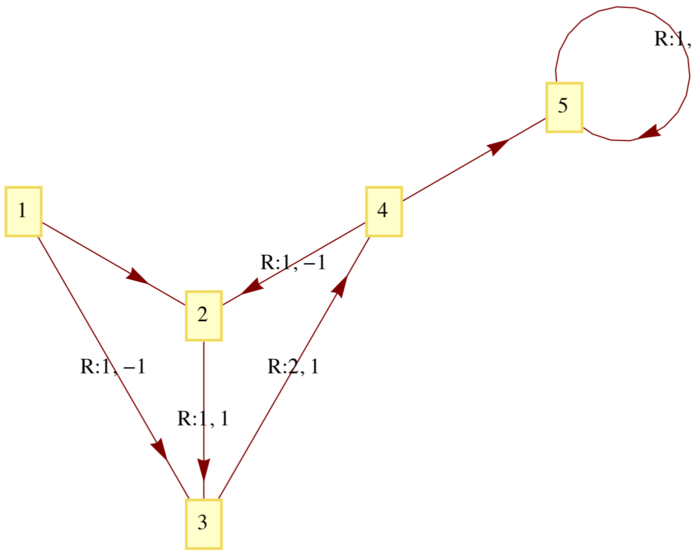

Figure 2. Program example: program number 2984.

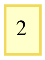

Figure 3. Program nomenclature: an instruction.

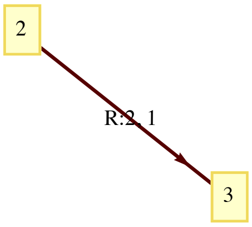

Figure 4. Program nomenclature: an increment instruction number 2 adds 1 bit to register 2 and then executes instruction 3.

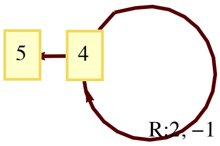

Figure 5. Program nomenclature: a decrement-jump instruction number 4, where if the value of register 2 is not 0 then 1 bit is subtracted from register 2 and instruction 4 is executed again. Otherwise, if the value of register 2 is 0, then instruction 5 is executed.

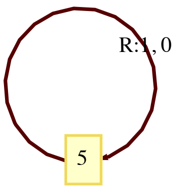

Figure 6. Program nomenclature: halted instructions are instructions used to stop the register machine's operation by repeating the halted instruction indefinitely.

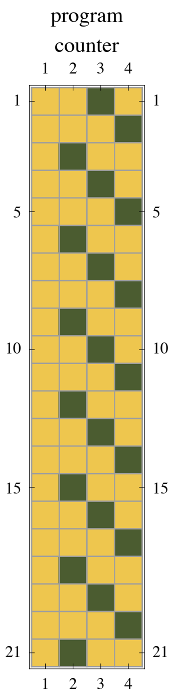

Figure 7. Program counter example.

2. Investigation Context

The author investigated the Minsky register machines during the New Kind of Science Summer School 2007, building on the results presented in Wolfram's *A New Kind of Science*. The initial objective was to investigate examples of nontrivial register machines but was extended to include examples of functional behavior due to the frequent occurrences of functions during the investigation. Wolfram *Mathematica* 6 was used with the Minsky register machine implementation.

Due to time and hardware limitations, the author only studied register machines with two registers with 8 bits in width and programs with four instructions with a program execution time of 50 instructions. Fifty instructions was chosen to avoid the halting problem: if a program did not halt after 50 instructions, then it was assumed to never halt.

The number of possible register machine programs (similar in concept to "rules" in cellular automata theory) is calculated by $(\rho(\iota + 1))^{\iota}$, where ι is the number of instructions and ρ is the number of registers. This definition is described in further detail in Appendix A. With this configuration, there are 10 000 possible programs to study.

The author observed every possible four-instruction, two-register configuration program with various initial program counters and register values. For example, if two registers have initial values of 2 and 4 respectively and final values of 6 and 0 respectively, then the program may yield an addition function and would merit further investigation. Similarly, if the registers or program counters yielded nontrivial behavior, then the program was analyzed further. A subsequent, exhaustive analysis of all possible four-instruction, two-register machines that assessed a register machine against formal definitions of nontrivial and functional behavior was performed, with the results documented in Section 3. It required approximately 24 hours of continuous processing time on an Apple MacBook, late-2008 edition with a 2.4 GHz dual-core processor and 4GB of RAM running two *Mathematica* 8.0.4 kernels.

3. Observations

The author observed that with the Minsky register machine, 8700 register machines (87%) reached a halted state and therefore achieved a function. However, this does not imply that nontrivial behavior cannot be observed from register machines that do not halt, such as the example register machine 2985 shown in Figure 8.

Many of the register machines exhibited the following behaviors.

- Nontrivial behavior that included register-independent and register-dependent behavior, conditional and cyclical behavior, and randomness in register machine components where:
  - Register-dependent register machines halt in at least one instruction due to at least one register having a zero value for any initial register value, while register-independent behavior involves a register machine not halting for any initial register value.

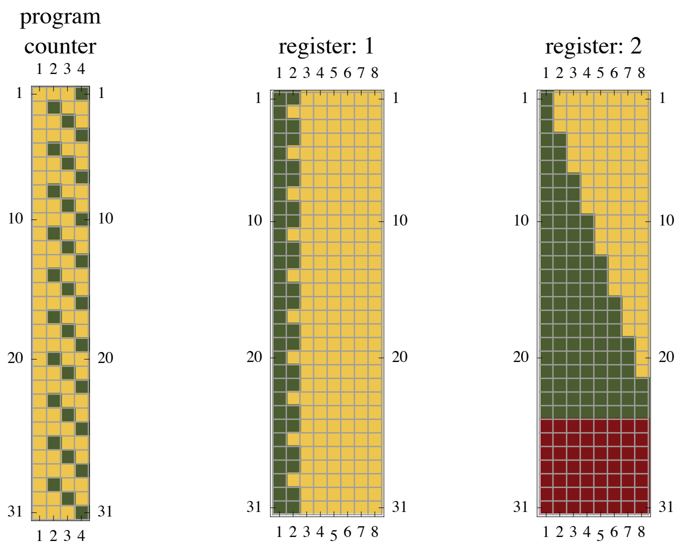

Figure 8. Example of nontrivial behavior from a register machine (program number 2985) that does not halt.

- Conditional register machine behavior when its register values determines which instructions are evaluated. Similarly, cyclical behavior is where the register machine does not halt and continually executes the same instructions in the same order, such as in Figure 8.
- Randomness, which according to Wolfram's definition as "standard methods of perception and analysis could not find any short description from which the thing could faithfully be reproduced" [1, p. 557] in a register machine's individual register and program counter values.

- Functional behavior where the register machine performed an arithmetic or logical operation, such as adding the value of two registers and storing the results in one of the registers.

3.1 Nontrivial Behavior

Using these definitions for all possible initial register values for two-register, four-instruction register machines, 4092 register machines (40.9%) showed register-independent behavior and 5908 register machines (59.1%) showed register-dependent behavior. Only 176 register machines (1.8%) exhibited conditional behavior and 1300 register machines (13%) exhibited cyclical behavior.

A simple frequency analysis of randomness was conducted to assess the randomness of the behavior of a register machine. This consisted of assessing the fit of the non-halted program counter or register values against the discrete uniform distribution by using a goodness-of-fit test with a significance value of $\alpha = 0.05$. The statistical analysis ignores register machines whose initial program counter and register values halt immediately or cannot be analyzed against the discrete uniform distribution: more specifically, those that only contain a single unique value. Figure 9 is a plot of the average p-values from the distribution fit test: comparing the program counter values against the discrete uniform distribution for all initial program counter values, organized by initial register values and program numbers.

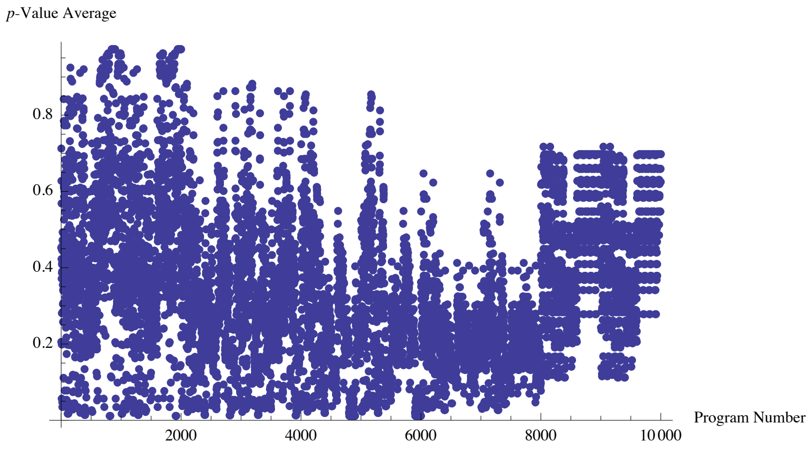

Figure 9. Average p-value of distribution fit test of the program counter values versus the program number, organized by initial register values.

Statistically, only 1804 register machine program counters (18.0%) exhibited statistically significant (i.e., p-value > significance value of 0.05) randomness for all initial program counter and register values.

Performing a similar analysis on the registers yields the interesting plots seen in Figures 10 and 11.

The discrete "lines" in Figures 10 and 11 are explained by a small, finite set of potential register values a register machine could generate. In register 1 and register 2 respectively, 7616 (76.2%) and 7524 (75.2%) register machine programs exhibited statistically significant randomness. Appendix B contains the actual values and their corresponding frequencies.

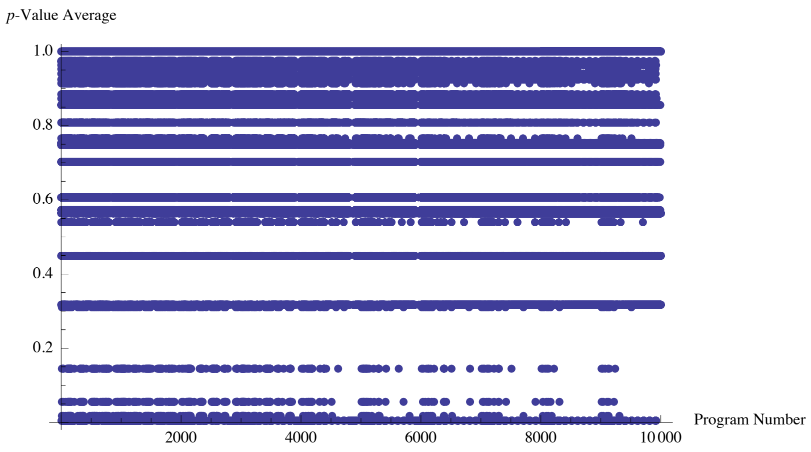

Figure 10. The p-value of a distribution fit test of register 1 values versus the program number.

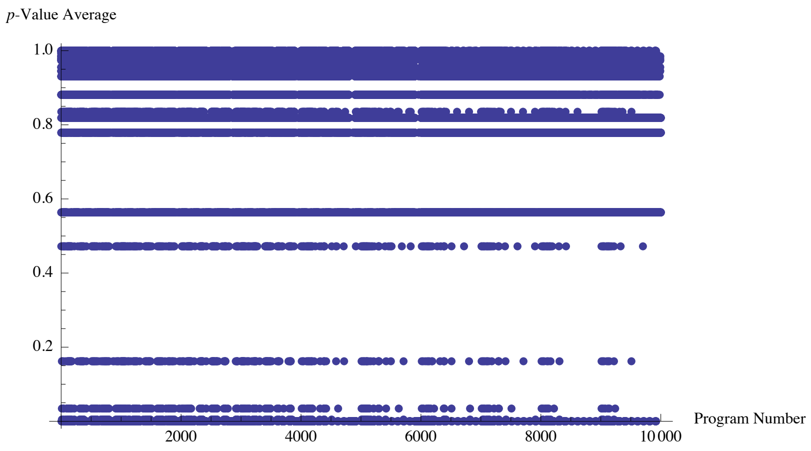

Figure 11. The p-value of a distribution fit test of register 2 values versus the program number.

3.2 Functional Behavior

A brute-force analysis of all possible four-instruction, two-register configuration programs was conducted to discover register machines that, given a particular initial program counter value, achieve a particular arithmetic or logical function for all possible initial register values. The following arithmetic and logical functions were discovered:

- Add function: 378 register machines
- Subtract function: 820 register machines

- Multiplication function: 31 register machines
- Divide function: 48 register machines, of which four performed the divide operation and 44 performed the divide operation and added the result to an existing register value
- Clear function: 5168 register machines cleared the first register, 5168 registers cleared the second register, and 2330 register machines cleared both registers

Appendix C contains a list of the register machines that performed these mathematical functions.

4. Register Machine Examples

4.1 Nontrivial Behavior

4.1.1 Register-Dependent and Register-Independent Behavior

In the example of nontrivial behavior shown in Figure 12, program number 2681 exhibits a pattern where register 1's value has 1 added and then 2 subtracted repeatedly until register 1 is empty. The increment operation in instruction 2 and the decrement-jump operations in instructions 3 and 4 yield this behavior with halting occurring because of the decrement operation in instruction 4.

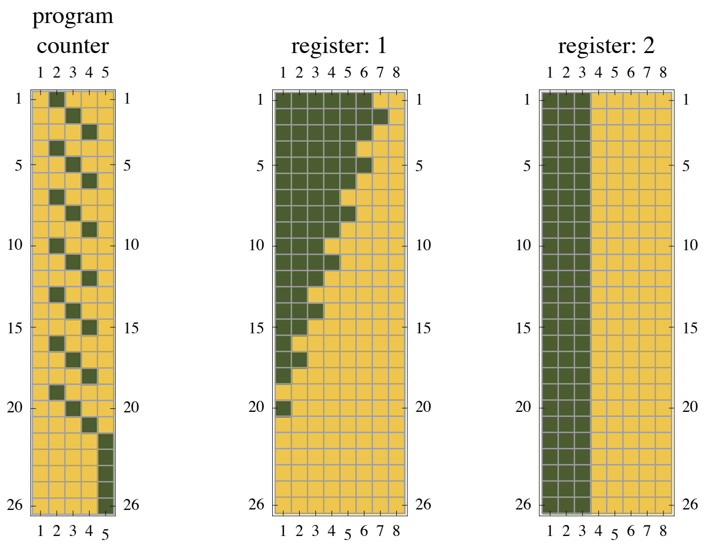

Figure 12. Program number 2681's registers with an initial program counter value of 2, register 1's initial value of 6 bits, and register 2's initial value of 3 bits.

The example shown in Figure 13 is very similar to the example shown in Figure 12, except in this case register 2's value has 2 bits added and then 1 bit subtracted repeatedly. This example is of significance as it is structurally similar to Figure 12, but it does not halt. This is due to the infinite loop caused by instructions 2 and 4, which increment and decrement the same register and never activate the decrement case where register 1 is equal to 0.

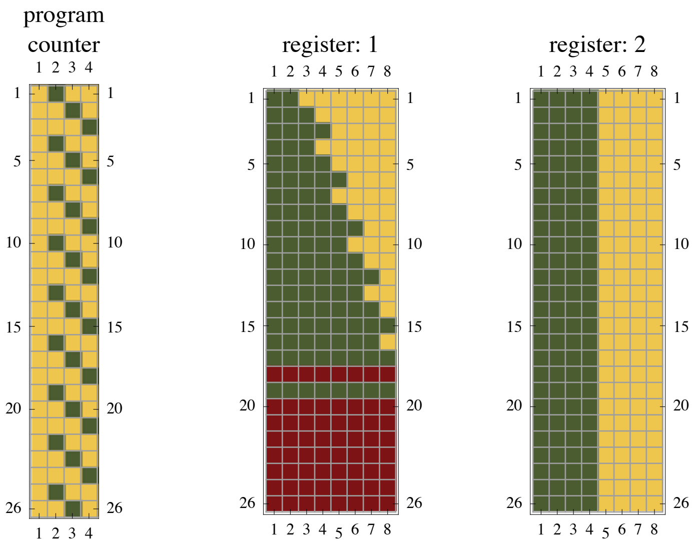

Figure 13. Program number 2881's registers with an initial program counter value of 2, register 1's initial value of 2 bits, and register 2's initial value of 4 bits.

The example in Figure 14 shows a non-halting register machine that does not cause an overflow in any register.

These programs are of interest as they exhibit register-independent behavior. The example program in Figure 15 exhibits register-dependent behavior: with an initial program counter value of 1, register 1 is cleared and then register 2 is cleared while incrementing and decrementing register 1. However, if the program counter is initialized at 2, register 2 is cleared while incrementing and decrementing register 1's value.

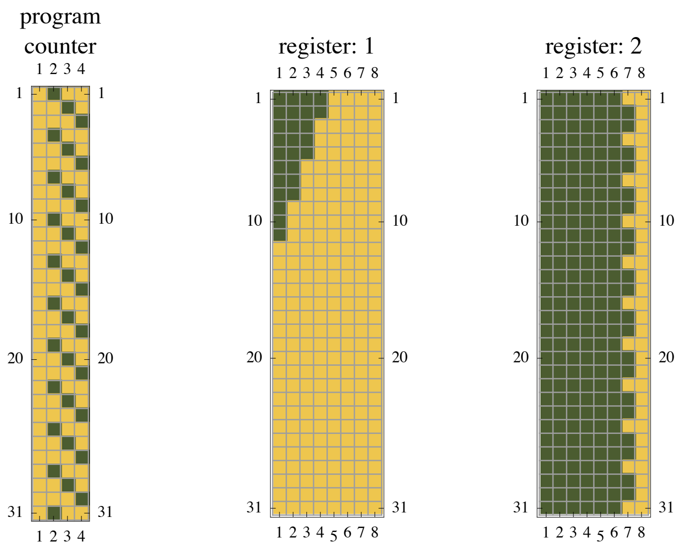

Figure 14. Program number 3691's registers with an initial program counter value of 2, register 1's initial value of 4 bits, and register 2's initial value of 6 bits.

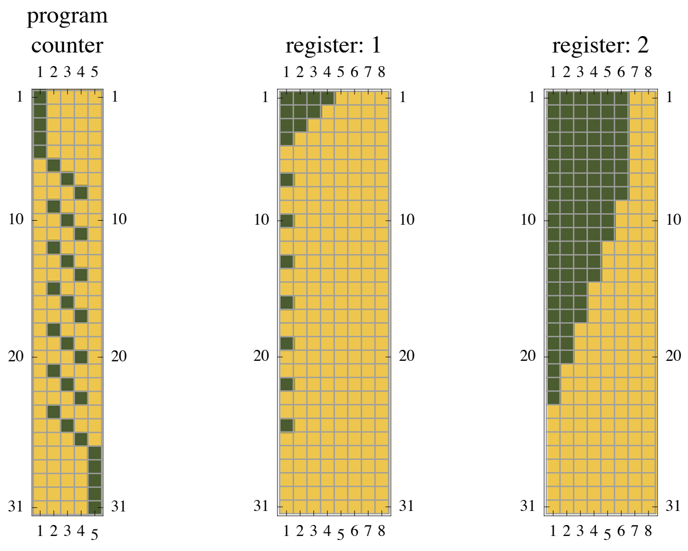

Figure 15. Program number 3680's registers with an initial program counter value of 1, register 1's initial value of 4 bits, and register 2's initial value of 6 bits.

4.1.2 Conditional and Cyclical Behavior

There are other examples of register machines that not only exhibit nontrivial behavior but also exhibit conditional behavior, where the register machine's behavior depends on certain conditions being satisfied. These conditions usually involve a register being set to 0 and a program using a decrement-jump operation to yield nontrivial behavior. For the program number 386, starting the program at different initial program counter values yields a variety of behaviors (images are shown in Appendix D):

- If register 1 has an odd value and the initial program counter value is 1, then the program clears the value of register 1 and halts without altering register 2. Similarly, if register 1 has an even number and the initial program counter value is 4, then the program clears the value of register 1 without altering register 2. This is caused by the fact that instruction 4 is the last instruction to be executed when register 1 is empty, so the program halts.
- If the initial program counter value is 2 with an even number stored in register 2, then the program adds the value of register 2 plus 1 to register 1 (i.e., register 1's value = register 1's initial value + register 2's initial value + 1), then clears register 1. This is similar to the previous case, as instruction 4 is the last instruction to be executed when the register machine halts after clearing register 1.
- However, if the last instruction to be executed in the last decrement operation was instruction 1, then the register machine enters an infinite loop, continuously incrementing and decrementing register 1's value as shown in Appendix D. Similar behavior is observed for initializing this register machine at an initial program counter value of 3 or 4.

Another example of conditional behavior is program number 5169. This program:

- subtracts the initial value of register 1 from register 2,
- clears register 1, and
- oscillates between (register 1's initial value − register 2's initial value) and (register 1's initial value − register 2's initial value − 1).

In Figure 16, as register 1's initial value is 4 and register 2's initial value is 6, then register 2's final register value oscillates between 1 and 2.

4.1.3 Randomness in a Register Machine's Program Counter

Program number 1274 is an example of randomness in a register machine's program counter; this program subtracts the value of register 2 from register 1 and stores the result in register 1 as shown in Figure 17.

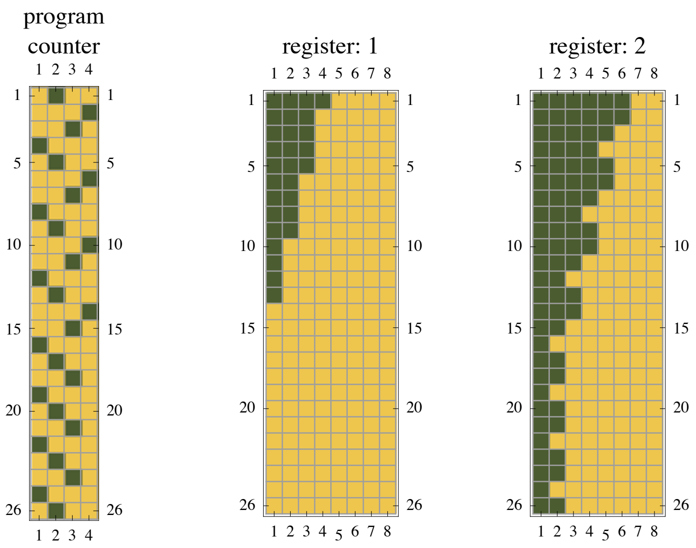

Figure 16. Program number 5169's registers with an initial program counter value of 2, register 1's initial value of 4 bits, and register 2's initial value of 6 bits.

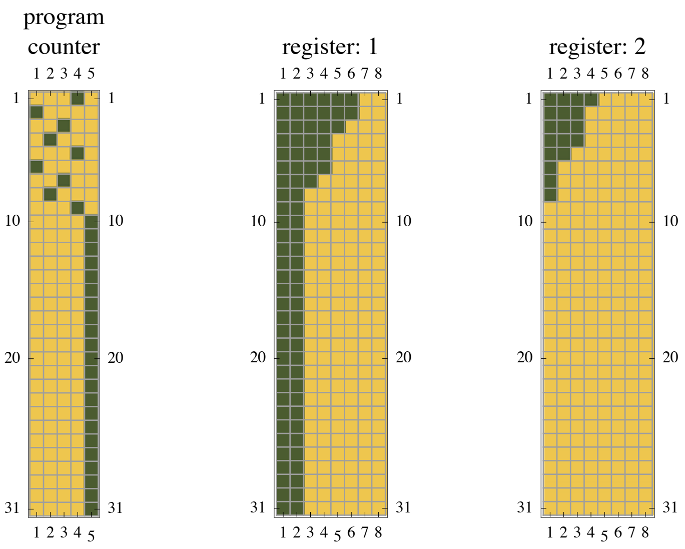

Figure 17. Program number 1274's registers with an initial program counter value of 4, register 1's initial value of 6 bits, and register 2's initial value of 4 bits.

However, under specific circumstances, such as register 2's initial value being 1 bit greater than register 1's value, the program counter exhibits a different kind of randomness, as shown in Figure 18.

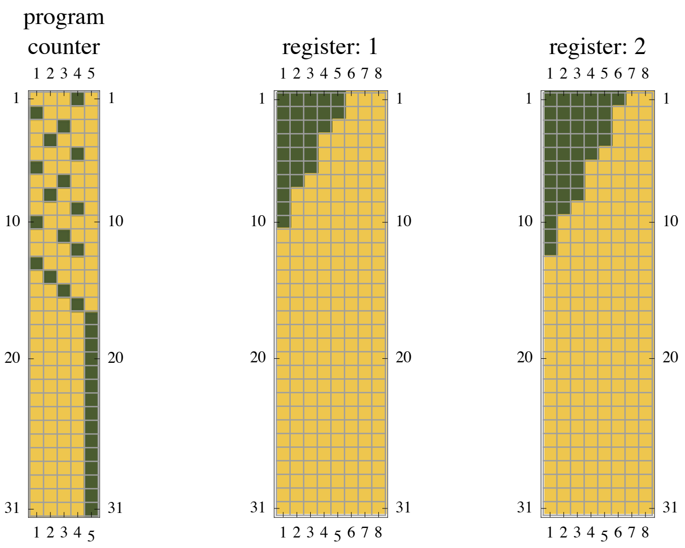

Figure 18. Program number 1274's registers with an initial program counter value of 4, register 1's initial value of 6 bits, and register 2's initial value of 4 bits.

4.1.4 Randomness in a Register Machine's Registers

Figures 19 and 20 are two examples of randomness in a register machine's registers.

- Program number 1825 appears to have two distinct functions:
  - If register 2's value is even, then the value of register 2 is divided by 2, the result is added to register 1, and the program halts.
  - If register 2's value is odd, then the value of register 2 is divided by 2 and the integer component of the result is added to register 1; then register 1 is cleared and incremented, and then the program halts.

While the behavior of this register machine can be easily explained, such a large variance in its behavior warranted its inclusion in a study on randomness in register machines.

- Program number 2715 is a functionally simple register machine that clears both registers. The way it clears registers is not simple. Register 2 clears 3 bits and then clears 1 bit from register 1 until register 2 is empty, and then register 1 has 1 bit cleared after every three instructions. Figure 21 shows how the register machine performs this behavior under different initial conditions.

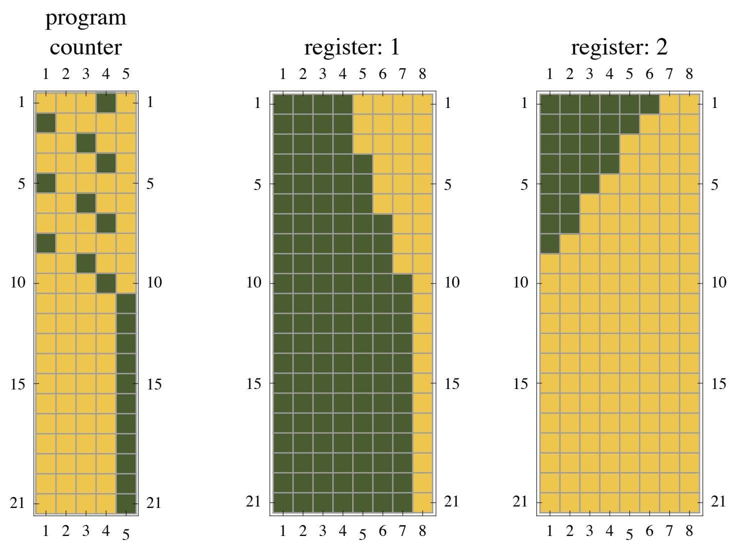

Figure 19. Program number 1825's registers with an initial program counter value of 4, register 1's initial value of 4 bits, and register 2's initial value of 6 bits.

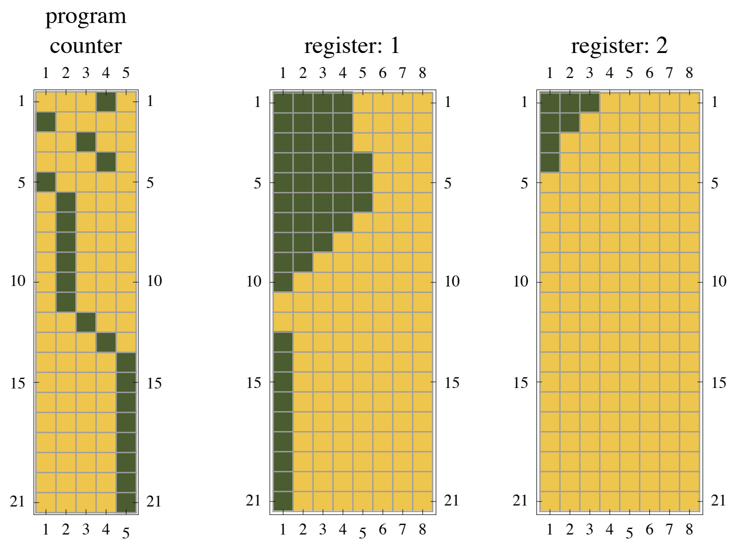

Figure 20. Program number 1825's registers with an initial program counter value of 4, register 1's initial value of 4 bits, and register 2's initial value of 3 bits.

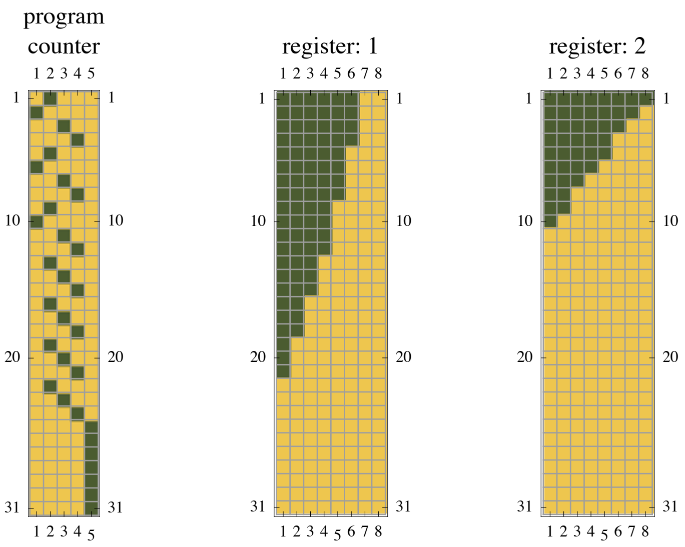

Figure 21. Program number 2715's registers with an initial program counter value of 2, register 1's initial value of 6 bits, and register 2's initial value of 8 bits.

4.2 Functional Behavior Examples

Considering register machines are theoretical implementations of practical information and communications technologies, register machines would be expected to perform basic arithmetic operations in a similar manner to a low-level information and communications technology device such as a microcontroller or microprocessor. All of the register machines that demonstrate functional behavior halt after performing their intended operation. It is also assumed that the register value represents a decimal value, so if a register's value is 5 bits, then it is storing a decimal value of 5, implying that this simple register machine can only process positive, integral values. Let $register\alpha_i$ be the value of register α at instruction i, so for example $register1_{\text{initial}}$ is the initial value of register 1 and $register2_{\text{final}}$ is the final value of register 2. Final value in this context is the value of a register after a register machine has halted.

4.2.1 The Addition Function

The register machine in Figure 22 (program number 4921) adds the value of register 1 to register 2 and clears register 1, or $register2_{\text{final}} = register1_{\text{initial}} + register2_{\text{initial}}$ and $register1_{\text{final}} = 0$. However, this register machine can also increment the value of register 1 and clear register 1; this can be accomplished by simply starting program number 4921 with an initial program counter value of 1. See Appendix D for examples of other addition functions.

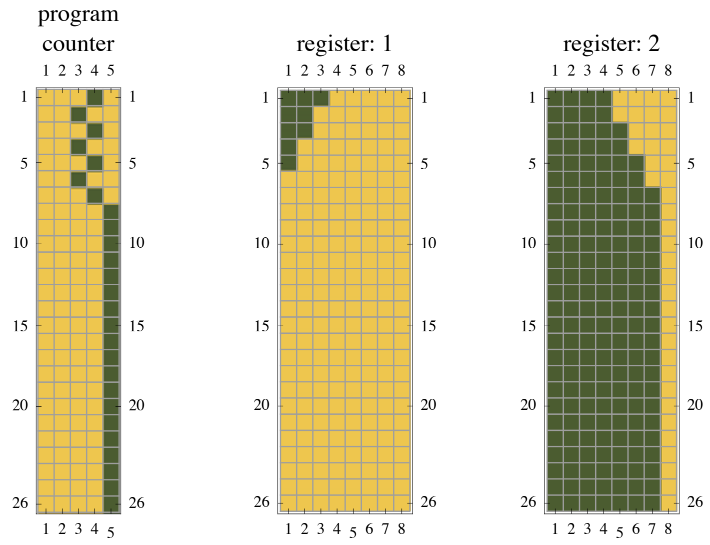

Figure 22. Program number 4921's registers with an initial program counter value of 4, register 1's initial value of 3 bits, and register 2's initial value of 4 bits. The result of adding register 1's value to register 2 and clearing register 1 is that register 1's final value is 0 and register 2's final value is 7.

4.2.2 The Subtraction Function

In program number 4721, register 2's value is subtracted from register 1 and register 1 is cleared, or

$$register2_{\text{final}} = \text{Max}\begin{pmatrix} register2_{\text{initial}} - register1_{\text{initial}} \\ 0 \end{pmatrix} \text{and}$$

$$register1_{\text{final}} = 0.$$

The maximum function is required, as this register machine assumes there is no way to represent negative values (see Figure 23).

Similar behavior can be observed from Figure 22, where setting the program counter's initial value to 1 decrements register 2's value and clears register 1.

4.2.3 The Multiplication Function

Program number 3882, shown in Figures 24 and 25, represents a register machine that doubles the value of register 2 and adds it to register 1, clearing register 2 in the process. While multiplication can be performed by repeatedly adding the value of one register to another, this program adds the value of register 2 twice to register 1 while decrementing register 2, making the multiplier a constant value. Put simply, this register machine performs the following operations:

$$\mathit{register1}_{\text{final}} = \mathit{register1}_{\text{initial}} + 2 * \mathit{register2}_{\text{initial}} \text{ and}$$
$$\mathit{register2}_{\text{final}} = 0.$$

Furthermore, initializing the program counter at 2 results in the program multiplying one more than the value of register 2 by 2 and adding the result to register 1 while clearing register 2, so

$$\mathit{register1}_{\text{final}} = \mathit{register1}_{\text{initial}} + 2\left(\mathit{register2}_{\text{initial}} + 1\right) \text{ and}$$
$$\mathit{register2}_{\text{final}} = 0$$

(see Figure 26).

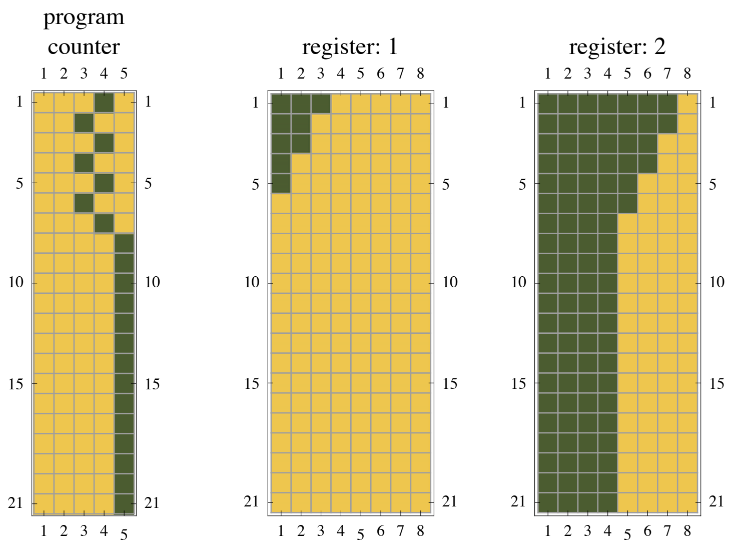

Figure 23. Program number 4721's registers with an initial program counter value of 4, register 1's initial value of 3 bits, and register 2's initial value of 7 bits. The result of subtracting register 1's value from register 2 and clearing register 1 is that register 1's final value is 0 and register 2's final value is 4.

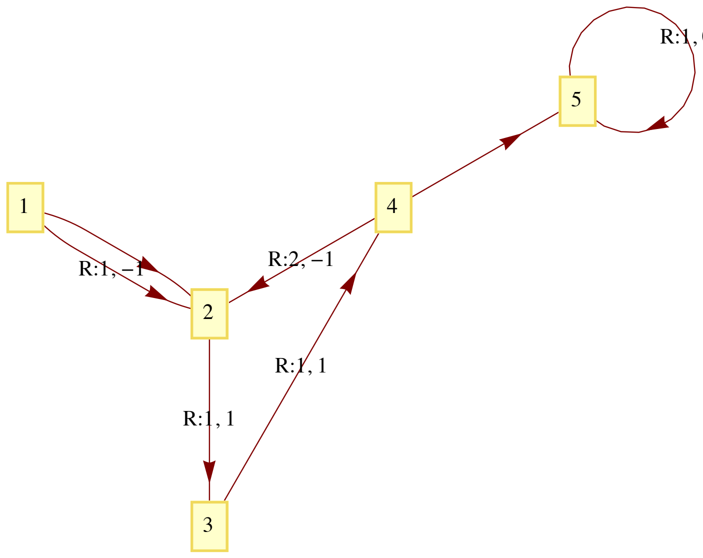

Figure 24. Program number 3882.

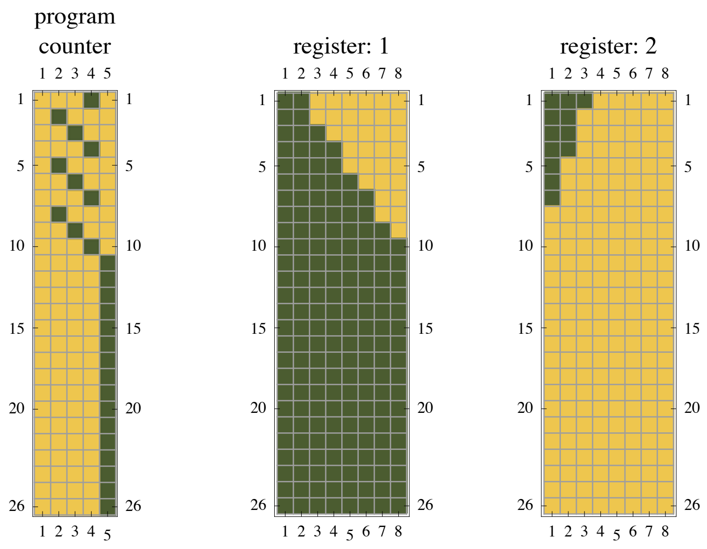

Figure 25. Program number 3882's registers with an initial program counter value of 4, register 1's initial value of 2 bits, and register 2's initial value of 4 bits. The result of multiplying register 2's value by 2, adding the result to register 1, and clearing register 2 is that register 1's final value is 8 and register 2's final value is 0.

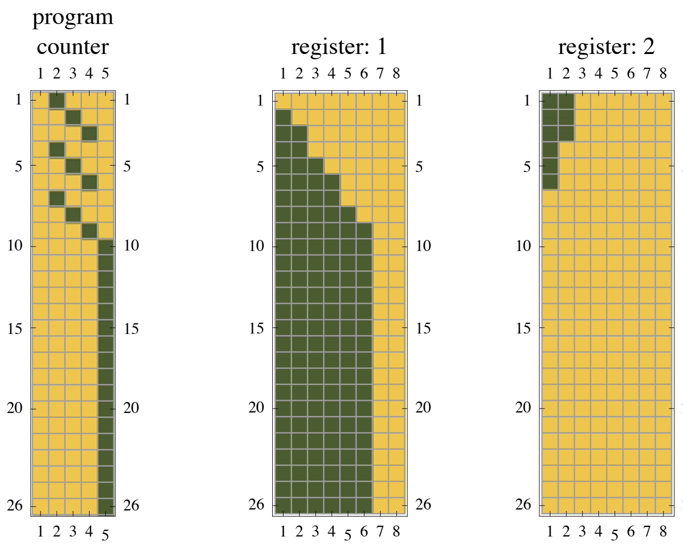

Figure 26. Program number 3882's registers with an initial program counter value of 2, register 1's initial value of 0 bits, and register 2's initial value of 2 bits. The result of multiplying one more than register 2's value by 2, adding the result in register 1, and clearing register 2 is that register 1's final value is 6 and register 2's final value is 0.

4.2.4 The Divide Function

Program number 3780 is another example of an implementation of a nontrivial mathematical operation: the divide operation. This program clears register 1, takes the integer component of dividing register 2's initial value by 2, increments it, and stores the result in register 2. Expressed mathematically, this is

$$\mathit{register1}_{\text{final}} = \left\lfloor \frac{\mathit{register2}_{\text{initial}}}{2} \right\rfloor + 1 \text{ and } \mathit{register2}_{\text{final}} = 0.$$

The implementation of this register machine program is very similar to Figure 26, except it is interesting to observe that the decrement-jump's behavior does not yield any nontrivial behavior: it performs a functional role in this context (see Figures 27 and 28).

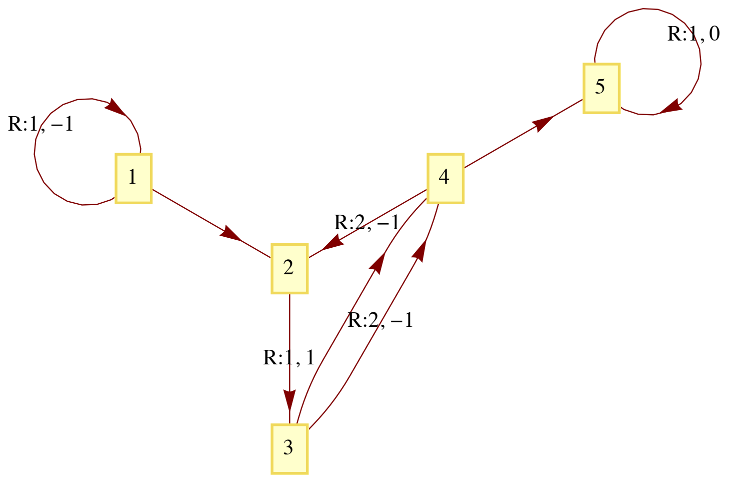

Figure 27. Program number 3780.

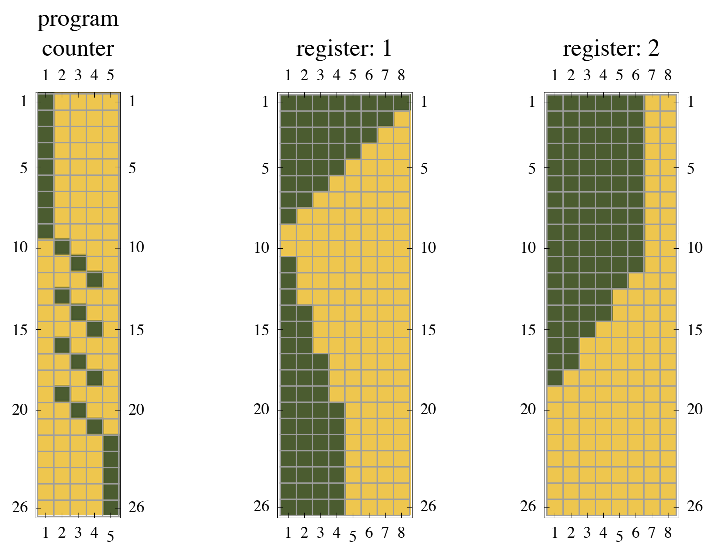

Figure 28. Program number 3780's registers with an initial program counter value of 1, register 1's initial value of 8 bits, and register 2's initial value of 6 bits. The result of clearing register 1, taking the integer component of dividing register 2's value by 2, incrementing it, and storing the result in register 1 is that register 1's final value is 4 and register 2's final value is 0.

Finally, initializing the program counter at any other value allows the register machine to add the integer component of dividing register 2's initial value by 2 to register 1, which is the most useful implementation of a "divide by two" register machine. The example shown in Figure 29 initializes the program counter at 3. Expressed mathematically, this is $\mathit{register1}_{\text{final}} = \left\lfloor \frac{\mathit{register2}_{\text{initial}}}{2} \right\rfloor + \mathit{register1}_{\text{initial}}$.

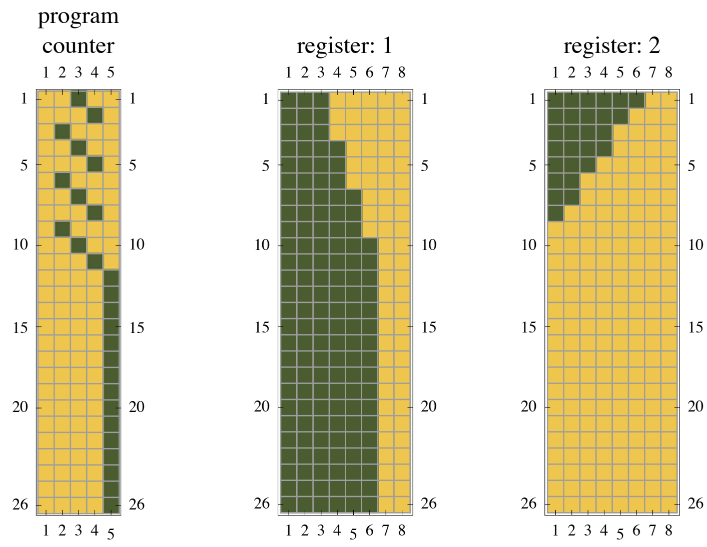

Figure 29. Program number 3780's registers with an initial program counter value of 3, register 1's initial value of 3 bits, and register 2's initial value of 6 bits. The result of taking the integer component of dividing register 2's value by 2, incrementing it, and storing the result in register 1 is that register 1's final value is 6 and register 2's final value is 0.

4.2.5 The Clear Function

The register machine programs 6420 and 7531 clear various registers.

Program number 6420 shown in Figure 30 only clears register 1's initial value and then halts. Observing the program's action in Figure 31, it can be seen that the program only operates on register 1 and ignores register 2's value, and it is clear that the initial program counter value does not have any practical impact on the functional behavior of the register machine except for the time it takes for the register machine to halt.

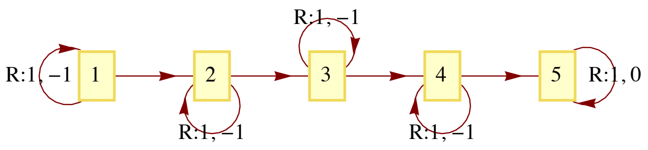

Figure 30. Program number 6420.

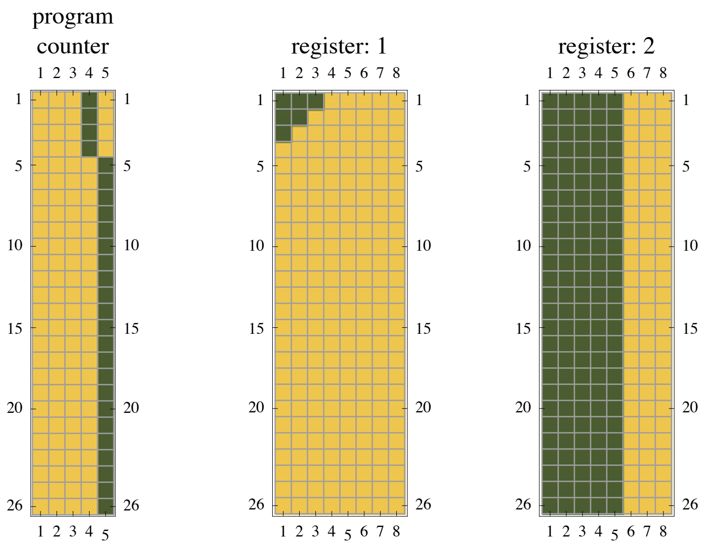

Figure 31. Program number 6420's registers with an initial program counter value of 4, register 1's initial value of 3 bits, and register 2's initial value of 5 bits. The result of clearing register 1 is that register 1's final value is 0 and register 2's final value is 5.

Program number 7531, shown in Figures 32 and 33, has a similar behavior to program number 6420, except it clears register 2 instead of register 1.

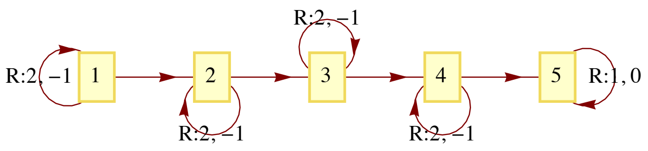

Figure 32. Functional behavior example: program number 7531's program.

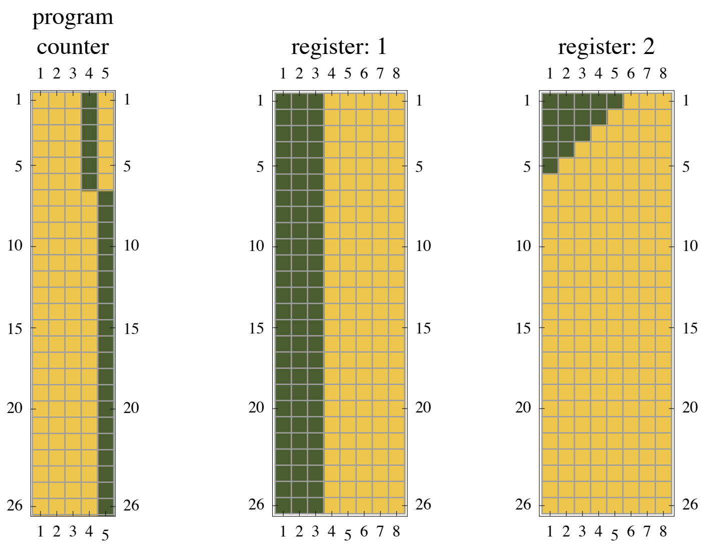

Figure 33. Program number 7531's registers with an initial program counter value of 4, register 1's initial value of 3 bits, and register 2's initial value of 5 bits. The result of clearing register 2 is that register 1's final value is 3 and register 2's final value is 5.

5. Conclusions

5.1 Investigation Results

From the results obtained, it is clear that register machines with a four-instruction, two-register configuration exhibit nontrivial and functional behavior. The set of register machine programs yielded multiple examples of nontrivial behavior, so further investigation for examples of nontrivial behavior with more complicated register machine configurations is warranted.

The examples of functional behavior exhibited by the simple four-instruction, two-register configuration is similar to the behavior typically seen in embedded platforms, where a "working" register is used as both an input and an output for a function. For example, an addition operation would add the contents of a register to the working register, storing the result in the working register. The contents of the working register would then be copied to another register for future use or used immediately for a subsequent operation. This supports the notion that register machines are good theoretical models of modern microcontroller and microprocessor technology.

5.2 Program Synthesis

An interesting observation from experimenting with these register machines is that certain register machines can perform different functions by starting at different initial conditions, especially by starting a register machine with different initial program counter values. In Section 4, examples of multiplying or dividing register values by a constant and by changing the initial program counter value are considered; different functional outputs could be observed from the register machine.

Another example is program number 5721, whose program is displayed in Figure 34. If the program counter is initialized to 1, then the register machine clears register 2 and then clears register 1 (see Figure 35). Initializing the program counter to 2 results in the register machine clearing register 1 before clearing register 2 (see Figure 36). It is also interesting to note that the program counter's pattern is very different from Figure 35.

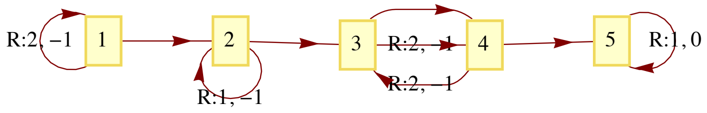

Figure 34. Program number 5721.

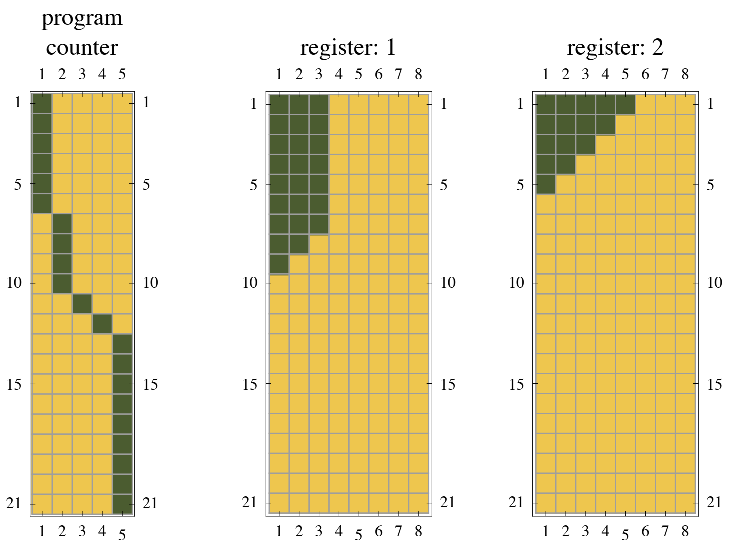

Figure 35. Program number 5721's registers with an initial program counter value of 1, register 1's initial value of 3 bits, and register 2's initial value of 5 bits. The result is that register 1 and register 2's final values are 0.

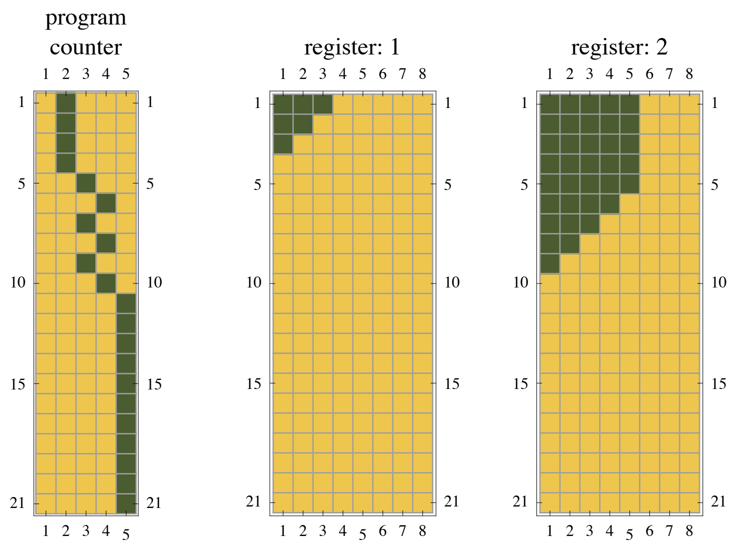

Figure 36. Program number 5721's registers with an initial program counter value of 2, register 1's initial value of 3 bits, and register 2's initial value of 5 bits. The result is that register 1 and register 2's final values are 0.

Finally, initializing the program counter to 3 results in the register machine clearing only register 2 and then halting without altering the value of register 1 (see Figure 37).

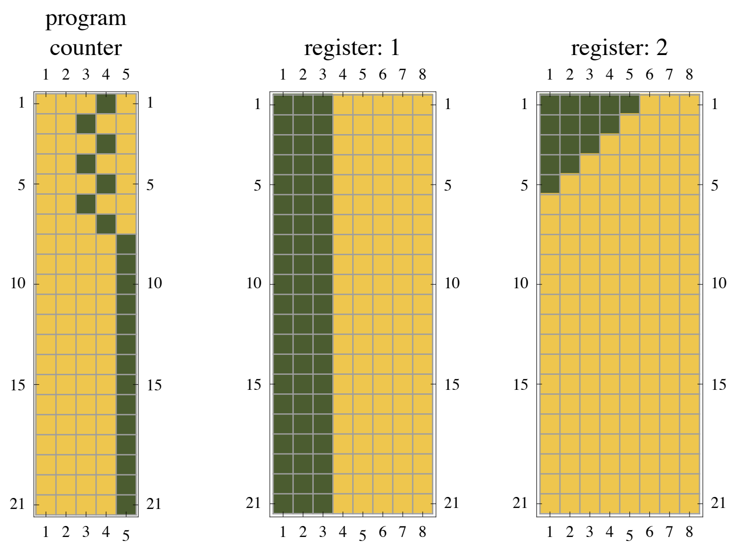

Figure 37. Program number 5721's registers with an initial program counter value of 3, register 1's initial value of 3 bits, and register 2's initial value of 5 bits. The result is that register 1's final value is 3 and register 2's final value is 0.

While this behavior has simple implications, this does suggest that the creation can be optimized for low-level logic devices or other Boolean logic using logic gates, such as complex programmable logic devices (CPLDs) and field programmable gate arrays (FPGAs) by creating these “super-programs,” which exhibit unique behavior by using different initial conditions. Theoretically, this is advantageous as this would offer lower overall usage of hardware components such as logic gates, potentially reducing the cost or size of an end product.

5.3 Optimizing Embedded Software

From observing these simple examples of functional behavior from two-register, four-instruction register machines, it is clear that certain register machines are quicker than other register machines at performing certain functions. That is, they execute fewer instructions to achieve a function. A trivial example involves clearing two registers. Program numbers 2551, 2741, and 6531 all clear both registers but require a different number of instructions to perform the task.

Program number 2551 with an initial program counter value of 1 and an initial value of 5 in both registers requires 25 instructions to clear both registers (see Figures 38 and 39). Further analysis suggests that it requires $3\,i + j + 5$ instructions to clear both registers and halt, where i and j are the initial values of registers 1 and 2, respectively.

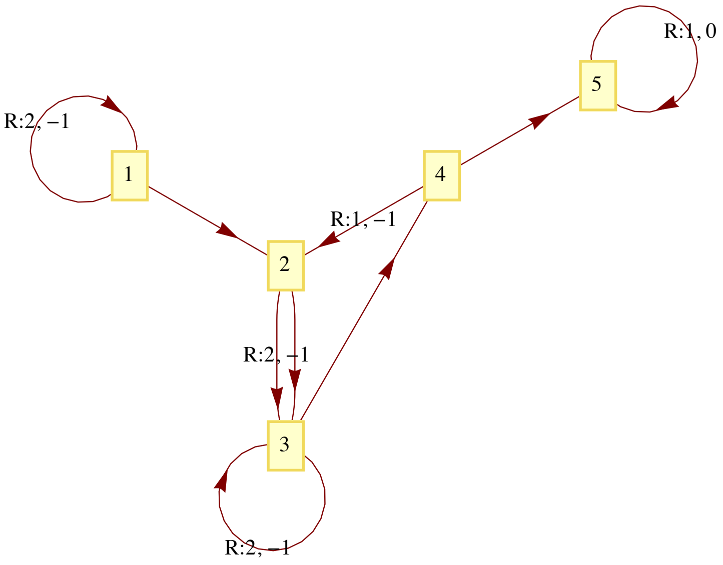

Figure 38. Program number 2551.

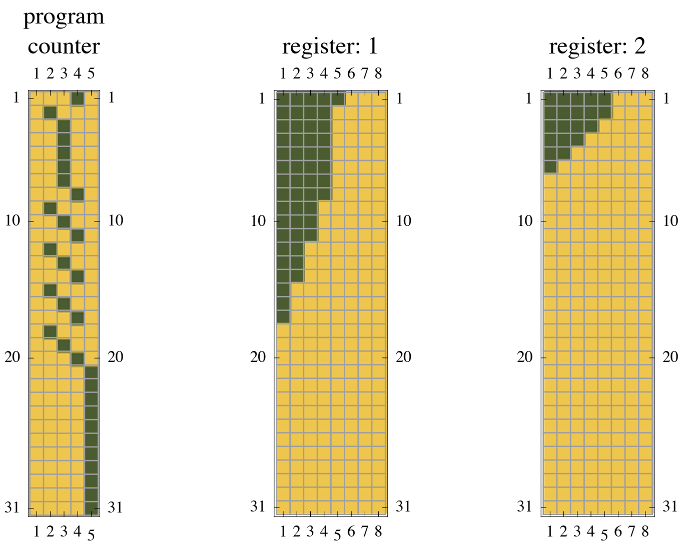

Figure 39. Program number 2551's registers with an initial program counter value of 5 and register 1 and register 2's initial value of 5 bits. This operation requires 25 instructions to halt after completing its task.

Program number 2741 with an initial program counter value of 1 and an initial value of 5 in both registers requires 16 instructions to clear both registers (see Figures 40 and 41). This register machine has a very different structure from that shown in Figure 39; it requires $3\left(\left\lfloor \frac{i}{2} \right\rfloor + 1\right) + j + 2$ instructions to clear both registers and halt, where i is the initial value of the first register and j is the initial value of the second register.

Program number 6531 with an initial program counter value of 3 and both registers having an initial value of 5 requires 12 instructions to clear both registers (see Figures 42 and 43). A similar, subsequent analysis suggests that it takes $i + j + 5$ instructions to clear both registers and halt.

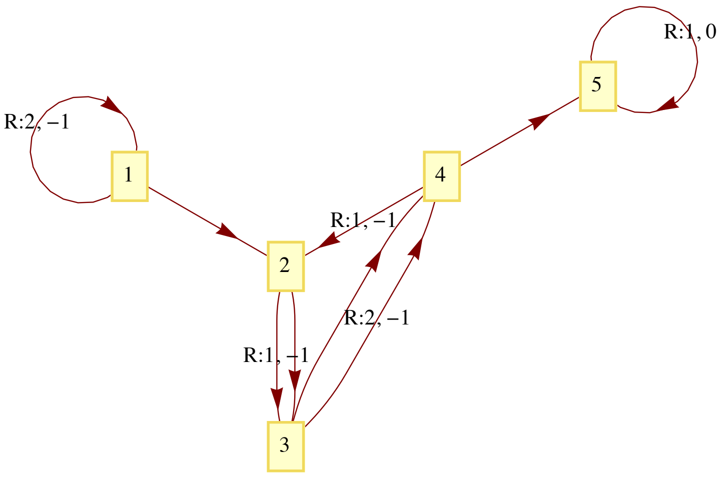

Figure 40. Program number 2741's program.

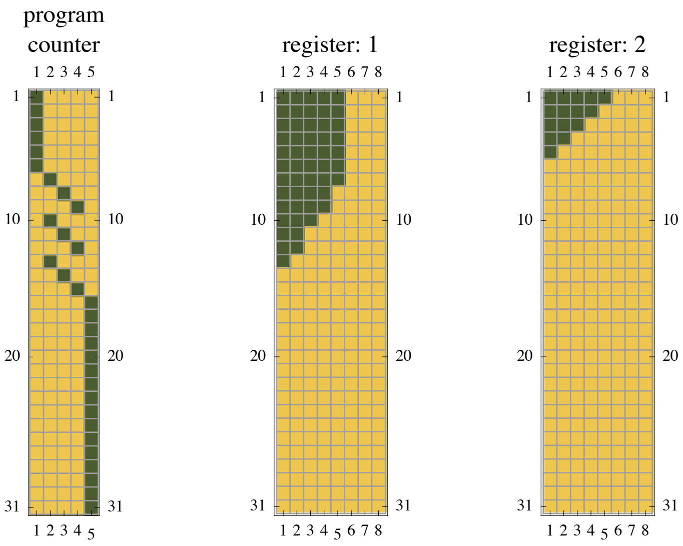

Figure 41. Program number 2741's registers with an initial program counter value of 1 and register 1 and register 2's initial value of 5 bits. This operation requires 16 instructions to halt after completing its task.

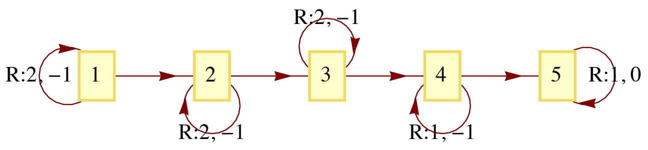

Figure 42. Program number 6531.

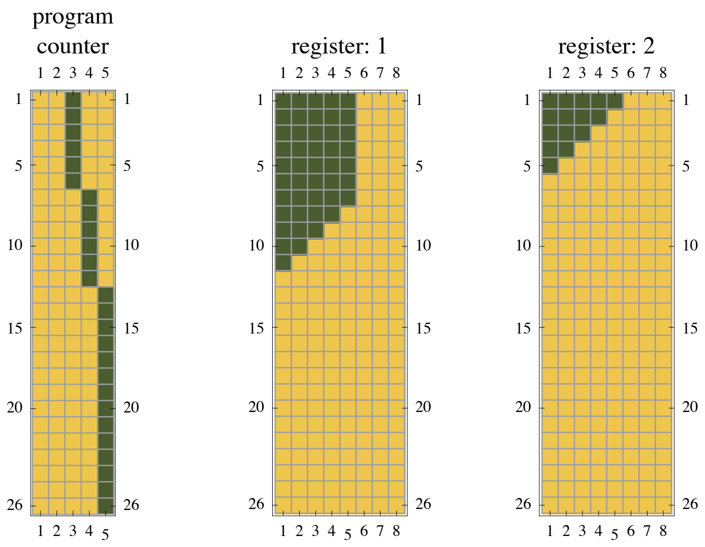

Figure 43. Program number 6531's registers with an initial program counter value of 1 and register 1 and register 2's initial value of 5 bits. This operation requires 15 instructions to halt after completing its task.

Plotting the functions that calculate the number of instructions required to clear all registers generates the plot in Figure 44 with the associated contour plots in Figure 45.

Graphically, program 6531 has the best performance out of the three functions tested for this set of two-register, four-instruction register machines. This raises an interesting concept: is it possible to test all possible programs to discover optimal programs that meet a particular required output? Current computer program optimization techniques involve recognizing patterns in inefficient programs and altering the instructions within these programs for a more efficient code. A simple example of such an optimization routine would involve removing redundant or unused instructions in a program. While being initially computationally expensive, a "mathematically optimal" program could be found through brute-force testing of all possible programs to find the program that performs the intended function while being optimized for a particular goal, such as finding the fastest or smallest program.

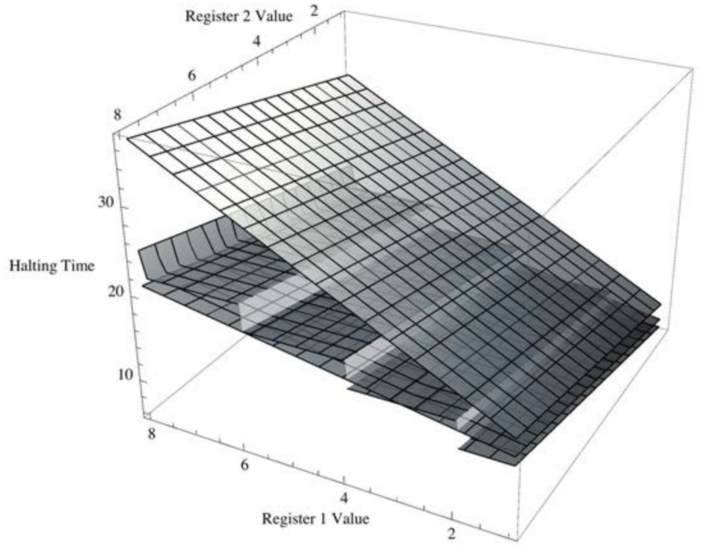

Figure 44. Three-dimensional plot of functions that calculate time to halt: program 2552 halts in $3i + j + 5$ instructions, program 2742 halts in $3\left(\left\lfloor \frac{i}{2} \right\rfloor + 1\right) + j + 2$ instructions, and program 6531 halts in $i + j + 5$ instructions.

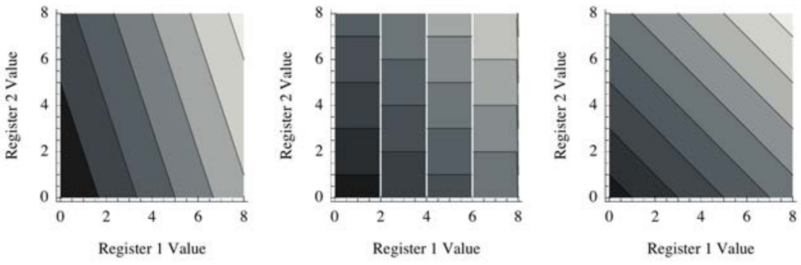

Figure 45. Contour plots of functions that calculate time to halt: program 2552 halts in $3i + j + 5$ instructions, program 2742 halts in $3\left(\left\lfloor \frac{i}{2} \right\rfloor + 1\right) + j + 2$ instructions, and program 6531 halts in $i + j + 5$ instructions.

After discussions with Wolfram and Todd Rowland, the author was introduced to the concept of superoptimizing as coined by Alexia (Henry) Massalin: a process if "given an instruction set, the superoptimizer finds the shortest program to compute a function" [4]. Unfortunately, Massalin's superoptimizer originally required several hours to explore programs of 12 instructions on a 16MHz computer [4]. However, given the rapid performance, reliability, and capacity improvements in modern hardware, could superoptimization be used as a design tool for firmware and embedded software developers to optimize performance or resource-intensive routines against a set of goals—optimizing for performance, energy use, or other metrics besides code size? These results could be adapted into a set of existing "rules" for optimization—peephole optimization—similar to the concept proposed by Sorav Bansal and Alex Aiken [5] where a database of outputs is created and desired outputs are searched for with the additional capability of optimizing for other design goals.

6. Future Research Directions

6.1 Register Machines

Future research into register machines would involve exploring more sophisticated register machines with more instructions and registers and larger register widths. From studying these simple register machines, examples of nontrivial behavior can be observed. In addition, the following basic mathematical and logic functions were identified:

- add the contents of a register to another register,
- subtract the contents of a register from another register,
- multiply the contents of a register by a constant value,
- divide the contents of a register by a constant value, and
- clear a register's contents.

Given the computational simplicity of the register machine, if a more precise definition of nontrivial behavior is used it would be possible to automatically discover further examples of nontrivial behavior by testing all possible register machine configurations with various initial conditions. Joost Joosten et al. conduct a highly detailed analysis of the complexity associated with Turing machines, in particular by considering another measure of descriptional complexity, where they define a Turing machine as being nontrivial (in this paper's context) "if its shortest description [where the description is the Turing machine and its input] cannot be much more shorter than the length of the string [the Turing machine's output] itself" [6]. More sophisticated pattern recognition techniques could assist in detecting examples of randomness beyond the frequency analysis conducted. In addition, this paper assumed that the data stored in a register was stored in a 1:1 ratio; that is, a value of 5 was represented by 5 bits. Other data representation systems could also be investigated, such as binary, octal, or binary-coded decimals, to discover further examples of functional behavior in a similar way to the representations considered for Turing machines in [6].

6.2 Practical Superoptimization

From the results in Section 5.3, the following set of circumstances now make superoptimization a viable and deterministic method of optimizing embedded software programs:

- cheaper, more accessible, and powerful computing infrastructure including grid- and cloud-computing systems using modern service models like platform as a service (PaaS) through providers such as Google App Engine and Windows Azure;
- improved support for embedded software development such as simulators, emulators, and profilers; and
- a need to be able to optimize software programs running on off-the-shelf hardware to meet a variety of non-functional requirements.

Therefore, future superoptimizer studies could study applications of superoptimizing in other programming languages or investigate different scenario types relevant to contemporary software engineering, such as reducing energy consumption or heat generated. In addition, complex programmable logic devices (CPLDs) often use proprietary programming languages such as the very high speed integrated circuit (VHSIC) hardware description language (VHDL) as defined in IEEE Standard 1076-2008, which would be amenable to superoptimization given the large industry adoption of the language, availability of emulation tools, and current access to high-performance computing infrastructure. Potential superoptimization scenarios could include optimizing a program for reduced execution time, smaller code size, fewer logic gates used, reducing heat emissions, or reducing energy consumption.

Acknowledgments

This paper is based on work the author did during the New Kind of Science Summer School 2007. The author is very grateful for the assistance and guidance provided by Dr. Stephen Wolfram, Dr. Todd Rowland, Dr. Eric Rowland, and the other administrators, tutors, Wolfram Research, Inc. personnel, and fellow students at the summer school.

The author would also like to thank his friend Mr. Deon Poncini and supervisors who provided valuable feedback as well as family, friends, and colleagues who provided advice, feedback, and support.

Wolfram *Mathematica*, Apple MacBook, Google App Engine, and Windows Azure are trademarks of their respective owners.

Appendix

A. Register Machine Functions

The following algorithms are implemented in *Mathematica* and are used in the “Register Machine” Demonstration available on the Wolfram Demonstrations site [3].

A.1 Register Machine Enumeration

The following algorithm is used to decode an enumeration, ranging from zero to the total number of register machine programs as defined.

```
convertEnumerationToState[value_,
   numberOfInstructions_, numberOfRegisters_] := Module[
   {currentState, nextState, registerNumber, increment},
  currentState = Quotient[value - 1,
      ((numberOfInstructions + 1) * numberOfRegisters)] + 1;
  nextState = If[Mod[value - 1, (numberOfInstructions + 1) *
         numberOfRegisters] >= numberOfInstructions *
        numberOfRegisters, Mod[Quotient[value - 1,
          (numberOfInstructions + 1) * numberOfRegisters],
        (numberOfInstructions + 1) * numberOfRegisters] + 2,
      Quotient[Mod[value - 1, (numberOfInstructions + 1) *
          numberOfRegisters], numberOfRegisters] + 1];
  registerNumber = Mod[value - 1, numberOfRegisters] + 1;
  increment = If[Mod[value - 1,
        (numberOfInstructions + 1) * numberOfRegisters] >=
       numberOfInstructions * numberOfRegisters, 1, -1];
  currentState -> {nextState, registerNumber, increment}];
```

A.2 Total Register Machine Program Algorithm

Definition 1. The total number of register machine programs can be calculated by the function $(\rho(\iota + 1))^{\iota}$, where ι is the number of instructions and ρ is the number of registers.

Consider an individual instruction: if there are ι instructions in a register, then there must be ι+1 possible instructions including the halted state.

Consider that any register machine instruction has:

- a current instruction itself,
- the next instruction to be executed, and
- the register that is being manipulated.

Then there are: 1 possible current instruction, ι +1 possible next instructions (considering the halted instruction as a possible instruction), and ρ possible registers. Therefore there are $\rho(\iota+1)$ possible instructions.

Now select ι instructions with replacement, which suggests there are $(\rho\,(\iota + 1))^{\iota}$ possible programs to select from.

Therefore, the total number of programs can be expressed as $(\rho\,(\iota + 1))^{\iota}$.

For example, consider a one-register, two-instruction register machine. Using this function, there are nine possible programs. The possible register machine programs are shown in Figures A1 through A9.

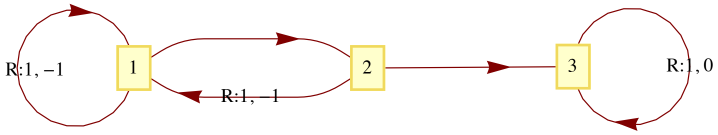

Figure A1. Program 1 of nine possible programs with 1 register and 2 possible instructions.

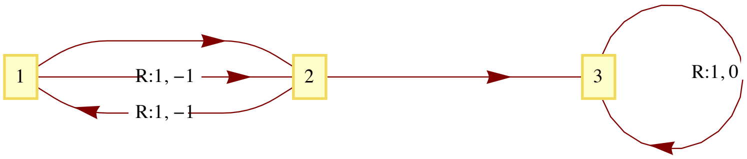

Figure A2. Program 2 of nine possible programs with 1 register and 2 possible instructions.

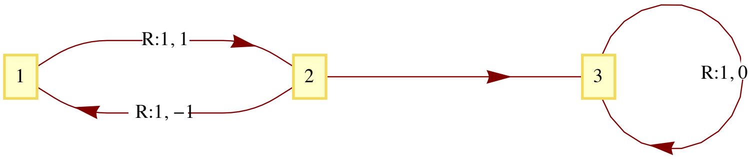

Figure A3. Program 3 of nine possible programs with 1 register and 2 possible instructions.

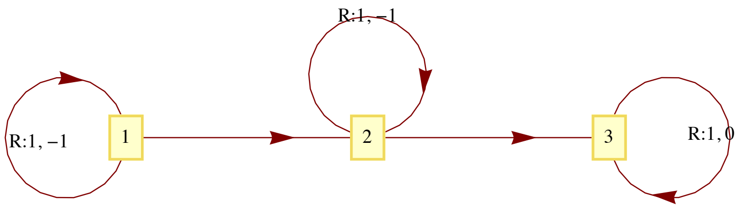

Figure A4. Program 4 of nine possible programs with 1 register and 2 possible instructions.

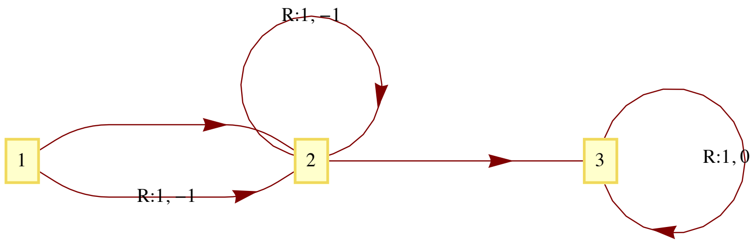

Figure A5. Program 5 of nine possible programs with 1 register and 2 possible instructions.

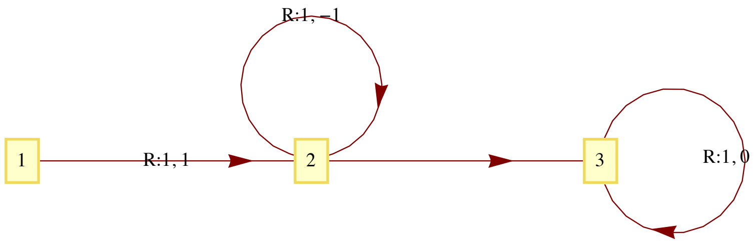

Figure A6. Program 6 of nine possible programs with 1 register and 2 possible instructions.

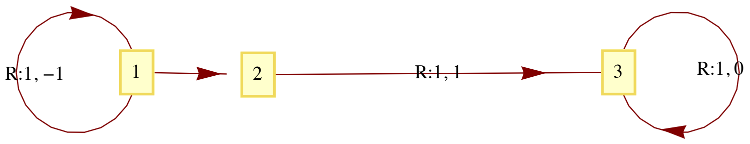

Figure A7. Program 7 of nine possible programs with 1 register and 2 possible instructions.

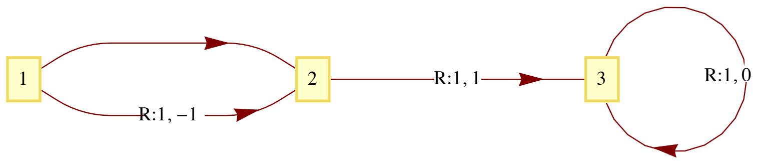

Figure A8. Program 8 of nine possible programs with 1 register and 2 possible instructions.

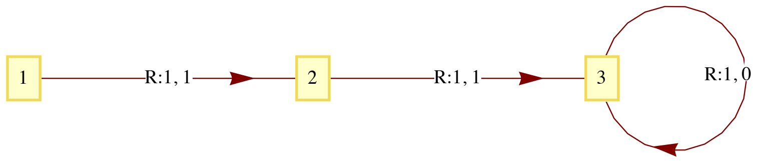

Figure A9. Program 9 of nine possible programs with 1 register and 2 possible instructions.

B. Raw *p*-Values of Frequency Analysis of Randomness in Register Values

The *p*-values and the respective counts for the distribution fit test of register 1's value against the discrete uniform distribution are shown at the left of Table B1.

The *p*-values and the respective counts for the distribution fit test of register 2's value against the discrete uniform distribution are shown at the right of Table B1.

p-Value	Count
0.00485213	338 328
0.0178312	4348
0.0556449	4960
0.144973	5396
0.310289	6262
0.317311	176 640
0.449329	161 520
0.539749	8446
0.563703	192 800
0.564718	156 860
0.572407	162 454
0.606531	151 138
0.702359	117 994
0.74768	167 496
0.753004	143 906
0.765857	10 938
0.808363	65 380
0.855695	92 488
0.873007	73 172
0.884549	50 114
0.914033	12 792
0.924313	59 818
0.939992	54 978
0.952577	24 124
0.963099	40 644
0.97244	20 522
0.97314	17 220
0.97365	31 916
0.974754	15 346
1.	192 000

p-Value	Count
0.0000310387	338 328
0.000454396	4348
0.00477391	4960
0.0344301	5396
0.161964	6262
0.472102	8446
0.563703	192 800
0.778801	176 640
0.818731	161 520
0.835225	10 938
0.881015	151 138
0.930627	156 860
0.945023	162 454
0.955375	167 496
0.973735	143 906
0.97874	117 994
0.984748	92 488
0.988102	12 792
0.992123	73 172
0.993373	65 380
0.996969	59 818
0.997839	54 978
0.998178	50 114
0.99896	17 220
0.999319	40 644
0.999923	31 916
0.999934	24 124
0.99999	15 346
1.	20 522

Table B1.

C. Arithmetic Function Program List

Sections C.1 through C.4 list the programs that for at least one initial program counter value performed a particular arithmetic function. These program numbers correspond to the enumeration defined in Appendix A and can be used in the Wolfram Demonstrations Project [3].

C.1 Addition Programs

The following 189 register machines add register 2's contents to register 1, expressed mathematically as

$$register1_{\text{final}} = register1_{\text{initial}} + register2_{\text{initial}}:$$

119, 319, 519, 719, 1119, 1319, 1381, 1382, 1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1519, 1619, 1689, 1719, 1790, 1799, 1849, 1860, 3119, 3319, 3381, 3382, 3383, 3384, 3385, 3386, 3387, 3388, 3389, 3390, 3519, 3619, 3719, 3819, 3919, 5119, 5319, 5381, 5382, 5383, 5384, 5385, 5386, 5387, 5388, 5389, 5390, 5519, 5619, 5719, 5801, 5802, 5803, 5804, 5805, 5806, 5807, 5808, 5809, 5810, 5811, 5812, 5813, 5814, 5815, 5816, 5817, 5818, 5819, 5820, 5821, 5822, 5823, 5824, 5825, 5826, 5827, 5828, 5829, 5830, 5831, 5832, 5833, 5834, 5835, 5836, 5837, 5838, 5839, 5840, 5841, 5842, 5843, 5844, 5845, 5846, 5847, 5848, 5849, 5850, 5851, 5852, 5853, 5854, 5855, 5856, 5857, 5858, 5859, 5860, 5861, 5862, 5863, 5864, 5865, 5866, 5867, 5868, 5869, 5870, 5871, 5872, 5873, 5874, 5875, 5876, 5877, 5878, 5879, 5880, 5881, 5882, 5883, 5884, 5885, 5886, 5887, 5888, 5889, 5890, 5891, 5892, 5893, 5894, 5895, 5896, 5897, 5898, 5899, 5900, 5919, 7119, 7319, 7381, 7382, 7383, 7384, 7385, 7386, 7387, 7388, 7389, 7390, 7519, 7619, 7719, 7919, 9119, 9319, 9381, 9382, 9383, 9384, 9385, 9386, 9387, 9388, 9389, 9390, 9519, 9619, 9719, and 9919.

The following 189 register machines add register 1's contents to register 2, expressed mathematically as

$$register2_{\text{final}} = register1_{\text{initial}} + register2_{\text{initial}}:$$

10, 210, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 410, 610, 690, 699, 710, 800, 949, 960, 1010, 1210, 1410, 1610, 2010, 2210, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2410, 2610, 2710, 2810, 2910, 4010, 4210, 4291, 4292, 4293, 4294, 4295, 4296, 4297, 4298, 4299, 4300, 4410, 4610, 4710, 4810, 4901, 4902, 4903, 4904, 4905, 4906, 4907, 4908, 4909, 4910, 4911, 4912, 4913, 4914, 4915, 4916, 4917, 4918, 4919, 4920, 4921, 4922, 4923, 4924, 4925, 4926, 4927, 4928, 4929, 4930, 4931, 4932, 4933, 4934, 4935, 4936, 4937, 4938, 4939, 4940, 4941, 4942, 4943, 4944, 4945, 4946, 4947, 4948, 4949, 4950, 4951, 4952, 4953, 4954, 4955, 4956, 4957, 4958, 4959, 4960, 4961, 4962, 4963, 4964, 4965, 4966, 4967, 4968, 4969, 4970, 4971, 4972, 4973, 4974, 4975, 4976, 4977, 4978, 4979, 4980, 4981, 4982, 4983, 4984, 4985, 4986, 4987, 4988, 4989, 4990, 4991, 4992, 4993, 4994, 4995, 4996, 4997, 4998, 4999, 5000, 6010, 6210, 6291, 6292, 6293, 6294, 6295, 6296, 6297, 6298, 6299, 6300, 6410, 6610, 6710, 6810, 8010, 8210, 8291, 8292, 8293, 8294, 8295, 8296, 8297, 8298, 8299, 8300, 8410, 8610, 8710, and 8810.

C.2 Subtraction Programs

The following 415 register machines subtract register 2's contents from register 1, expressed mathematically as

$$register1_{\text{final}} = \text{Max}\begin{pmatrix} register1_{\text{initial}} - register2_{\text{initial}} \\ 0 \end{pmatrix}:$$

198, 398, 818, 838, 858, 878, 1007, 1017, 1027, 1037, 1047, 1057, 1066, 1067, 1077, 1097, 1105, 1107, 1113, 1115, 1117, 1125, 1127, 1135, 1137, 1145, 1147, 1155, 1157, 1165, 1167, 1175, 1177, 1195, 1197, 1207, 1217, 1227, 1237, 1247, 1257, 1266, 1267, 1277, 1287, 1297, 1307, 1313, 1317, 1327, 1337, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 1357, 1365, 1367, 1377, 1397, 1407, 1417, 1427, 1437, 1447, 1457, 1467, 1477, 1497, 1507, 1513, 1517, 1527, 1537, 1547, 1557, 1567, 1577, 1597, 1607, 1617, 1627, 1637, 1647, 1649, 1657, 1660, 1667, 1677, 1683, 1687, 1697, 1707, 1713, 1717, 1727, 1737, 1747, 1750, 1757, 1767, 1777, 1793, 1797, 1907, 1917, 1927, 1937, 1947, 1957, 1967, 1977, 1997, 2198, 3061, 3062, 3063, 3064, 3065, 3066, 3067, 3068, 3069, 3070, 3098, 3105, 3113, 3115, 3125, 3135, 3145, 3147, 3155, 3161, 3162, 3163, 3164, 3165, 3166, 3167, 3168, 3169, 3170, 3175, 3195, 3197, 3261, 3262, 3263, 3264, 3265, 3266, 3267, 3268, 3269, 3270, 3313, 3341, 3342, 3343, 3344, 3345, 3346, 3347, 3348, 3349, 3350, 3361, 3362, 3363, 3364, 3365, 3366, 3367, 3368, 3369, 3370, 3461, 3462, 3463, 3464, 3465, 3466, 3467, 3468, 3469, 3470, 3513, 3561, 3562, 3563, 3564, 3565, 3566, 3567, 3568, 3569, 3570, 3613, 3615, 3661, 3662, 3663, 3664, 3665, 3666, 3667, 3668, 3669, 3670, 3713, 3761, 3762, 3763, 3764, 3765, 3766, 3767, 3768, 3769, 3770, 3913, 3915, 3961, 3962, 3963, 3964, 3965, 3966, 3967, 3968, 3969, 3970, 5069, 5080, 5105, 5113, 5115, 5125, 5135, 5145, 5155, 5165, 5170, 5175, 5195, 5313, 5341, 5342, 5343, 5344, 5345, 5346, 5347, 5348, 5349, 5350, 5513, 5601, 5602, 5603, 5604, 5605, 5606, 5607, 5608, 5609, 5610, 5611, 5612, 5613, 5614, 5615, 5616, 5617, 5618, 5619, 5620, 5621, 5622, 5623, 5624, 5625, 5626, 5627, 5628, 5629, 5630, 5631, 5632, 5633, 5634, 5635, 5636, 5637, 5638, 5639, 5640, 5641, 5642, 5643, 5644, 5645, 5646, 5647, 5648, 5649, 5650, 5651, 5652, 5653, 5654, 5655, 5656, 5657, 5658, 5659, 5660, 5661, 5662, 5663, 5664, 5665, 5666, 5667, 5668, 5669, 5670, 5671, 5672, 5673, 5674, 5675, 5676, 5677, 5678, 5679, 5680, 5681, 5682, 5683, 5684, 5685, 5686, 5687, 5688, 5689, 5690, 5691, 5692, 5693, 5694, 5695, 5696, 5697, 5698, 5699, 5700, 5713, 5913, 7105, 7113, 7115, 7125, 7135, 7145, 7155, 7165, 7175, 7195, 7313, 7341, 7342, 7343, 7344, 7345, 7346, 7347, 7348, 7349, 7350, 7513, 7713, 7913, 8195, 9105, 9113, 9115, 9125, 9135, 9145, 9155, 9165, 9175, 9195, 9313, 9341, 9342, 9343, 9344, 9345, 9346, 9347, 9348, 9349, 9350, 9513, 9713, and 9913.

The following 415 register machines subtract register 1's contents from register 2, expressed mathematically as

$$register2_{\text{final}} = \text{Max}\begin{pmatrix} register2_{\text{initial}} - register1_{\text{initial}} \\ 0 \end{pmatrix}:$$

4, 6, 8, 16, 18, 26, 28, 36, 38, 46, 48, 56, 58, 66, 68, 76, 78, 86, 88, 108, 118, 128, 138, 148, 158, 168, 175, 178, 188, 204, 208, 218, 228, 238, 248, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 268, 276, 278, 288, 308, 318, 328, 338, 348, 358, 368, 375, 378, 388, 398, 404, 408, 418, 428, 438, 448, 458, 468, 478, 488, 508, 518, 528, 538, 548, 558, 568, 578, 588, 604, 608, 618, 628, 638, 648, 658, 659, 668, 678, 684, 688, 708, 718, 728, 738, 748, 749, 758, 760, 768, 778, 788, 794, 798, 808, 818, 828, 838, 848, 858, 868, 878, 888, 1087, 1287, 1907, 1927, 1947, 1967, 2004, 2006, 2016, 2026, 2036, 2046, 2056, 2058, 2066, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2086, 2088, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2187, 2204, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2404, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2604, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 2680, 2704, 2706, 2771, 2772, 2773, 2774, 2775, 2776, 2777, 2778, 2779, 2780, 2804, 2806, 2871, 2872, 2873, 2874, 2875, 2876, 2877, 2878, 2879, 2880, 3087, 4004, 4006, 4016, 4026, 4036, 4046, 4056, 4066, 4076, 4079, 4086, 4169, 4180, 4204, 4251, 4252, 4253, 4254, 4255, 4256, 4257, 4258, 4259, 4260, 4404, 4604, 4701, 4702, 4703, 4704, 4705, 4706, 4707, 4708, 4709, 4710, 4711, 4712, 4713, 4714, 4715, 4716, 4717, 4718, 4719, 4720, 4721, 4722, 4723, 4724, 4725, 4726, 4727, 4728, 4729, 4730, 4731, 4732, 4733, 4734, 4735, 4736, 4737, 4738, 4739, 4740, 4741, 4742, 4743, 4744, 4745, 4746, 4747, 4748, 4749, 4750, 4751, 4752, 4753, 4754, 4755, 4756, 4757, 4758, 4759, 4760, 4761, 4762, 4763, 4764, 4765, 4766, 4767, 4768, 4769, 4770, 4771, 4772, 4773, 4774, 4775, 4776, 4777, 4778, 4779, 4780, 4781, 4782, 4783, 4784, 4785, 4786, 4787, 4788, 4789, 4790, 4791, 4792, 4793, 4794, 4795, 4796, 4797, 4798, 4799, 4800, 4804, 6004, 6006, 6016, 6026, 6036, 6046, 6056, 6066, 6076, 6086, 6204, 6251, 6252, 6253, 6254, 6255, 6256, 6257, 6258, 6259, 6260, 6404, 6604, 6804, 8004, 8006, 8016, 8026, 8036, 8046, 8056, 8066, 8076, 8086, 8204, 8251, 8252, 8253, 8254, 8255, 8256, 8257, 8258, 8259, 8260, 8404, 8604, 8804, and 9086.

C.3 Multiplication Programs

The following 16 register machines multiply register 1's contents by 2 and then add this value to register 2, expressed mathematically as

$register1_{\text{final}} = register1_{\text{initial}} + 2 * register2_{\text{initial}}$:

1189, 2189, 3189, 3881, 3882, 3883, 3884, 3885, 3886, 3887, 3888, 3890, 5189, 7189, and 9189.

The following 16 register machines multiply register 1's contents by 2 and then add this value to register 2, expressed mathematically as $register2_{\text{final}} = register2_{\text{initial}} + 2 * register1_{\text{initial}}$:

100, 2100, 2991, 2992, 2993, 2994, 2995, 2996, 2997, 2998, 2999, 3000, 3100, 4100, 6100, and 8100.

C.4 Division Programs

Register machines 3780 and 3851 take the integral part of dividing register 2's value by 2, incrementing the value, and storing the result in register 1, otherwise expressed mathematically as

$$register1_{\text{final}} = \left\lfloor \frac{register2_{\text{initial}}}{2} \right\rfloor + 1.$$

Register machines 2692 and 2942 similarly take the integral part of dividing register 1's value by 2, incrementing the value, and storing the result in register 2, otherwise expressed mathematically as

$$register2_{\text{final}} = \left\lfloor \frac{register1_{\text{initial}}}{2} \right\rfloor + 1.$$

Much like the other arithmetic register machines mentioned in Section 4.2, the following 22 register machines take the integral part of dividing register 2's value by 2 and adding the result to register 1, expressed mathematically as

$$register1_{\text{final}} = register1_{\text{initial}} + \left\lfloor \frac{register2_{\text{initial}}}{2} \right\rfloor:$$

1079, 1159, 1179, 1379, 1579, 1679, 1779, 3159, 3780, 3782, 3783, 3784, 3785, 3786, 3787, 3788, 3789, 3790, 5159, 5184, 7159, and 9159.

The following 22 register machines take the integral part of dividing register 1's value by 2 and adding the result to register 2, expressed mathematically as

$$register2_{\text{final}} = register2_{\text{initial}} + \left\lfloor \frac{register1_{\text{initial}}}{2} \right\rfloor:$$

50, 70, 170, 270, 470, 670, 770, 2050, 2691, 2692, 2693, 2694, 2695, 2696, 2697, 2698, 2699, 2700, 4050, 4093, 6050, and 8050.

D. Other Register Machine Examples

D.1 Complex Register Machines

The register machine outputs of program 386 (Figure D1) in Figures D2 and D3 show how register machines can be observed to either halt or run indefinitely after register 1's contents is cleared.

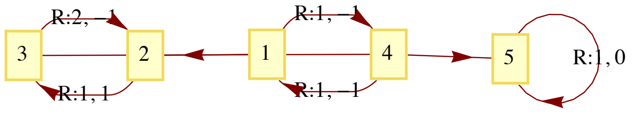

Figure D1. Program number 386.

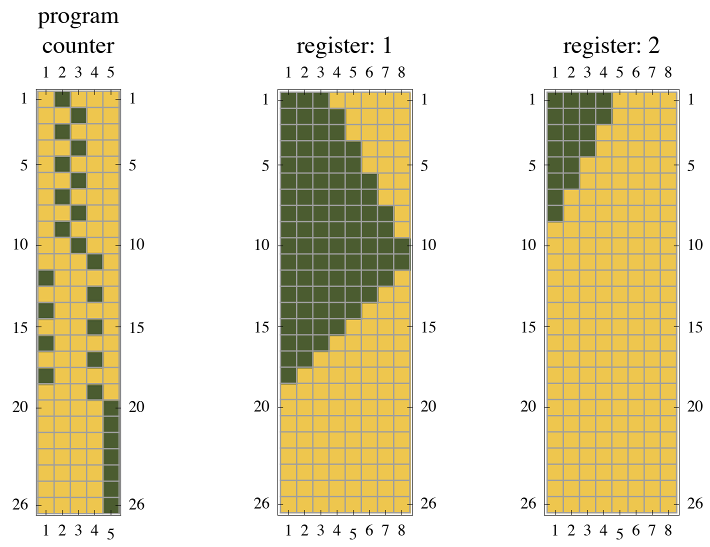

Figure D2. Program number 386's registers with an initial program counter value of 2, register 1's initial value of 3 bits, and register 2's initial value of 4 bits.

D.2 Functional Register Machines

D.2.1 The Addition Operation

The register machine shown in Figures D4 and D5 increments the contents of the second register before halting, a specific implementation of the addition function.

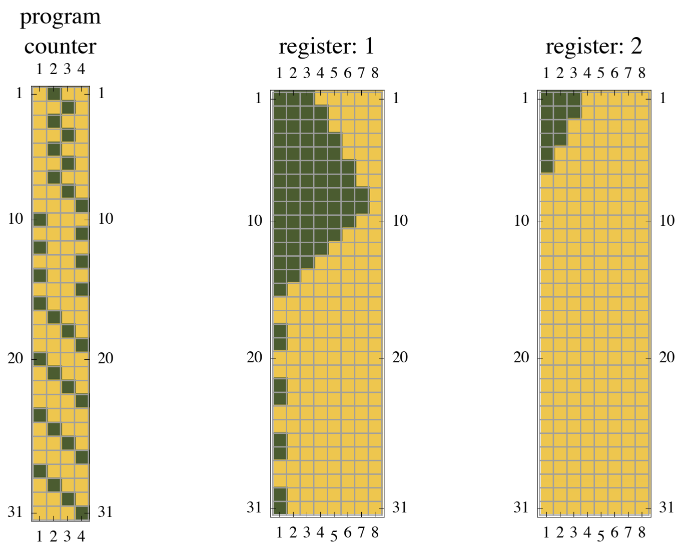

Figure D3. Program number 386's registers with an initial program counter value of 2, register 1's initial value of 3 bits, and register 2's initial value of 3 bits.

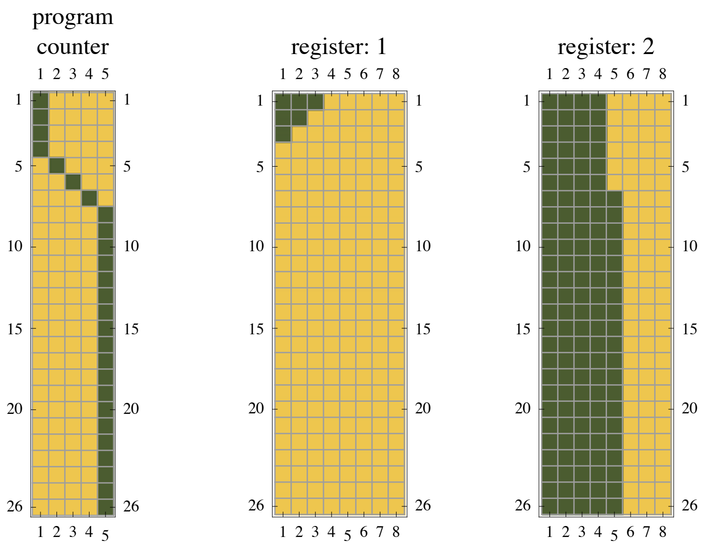

Figure D4. Program number 4920's registers with an initial program counter value of 1, register 1's initial value of 3 bits, and register 2's initial value of 4 bits. The result of incrementing register 2's value and clearing register 1 is that register 1's final value is 0 and register 2's final value is 5.

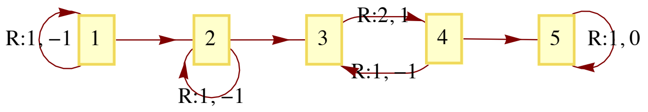

Figure D5. Program number 4920.

D.2.2 The Subtraction Operation

Similarly, the register machine shown in Figures D6 and D7 decrements the contents of the second register prior to halting; the increment operation is replaced by a decrement-jump operation at instruction 3.

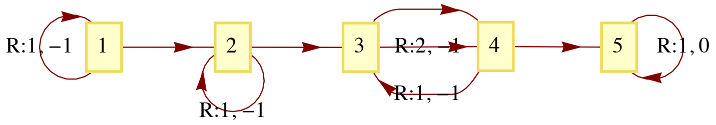

Figure D6. Program number 4720.

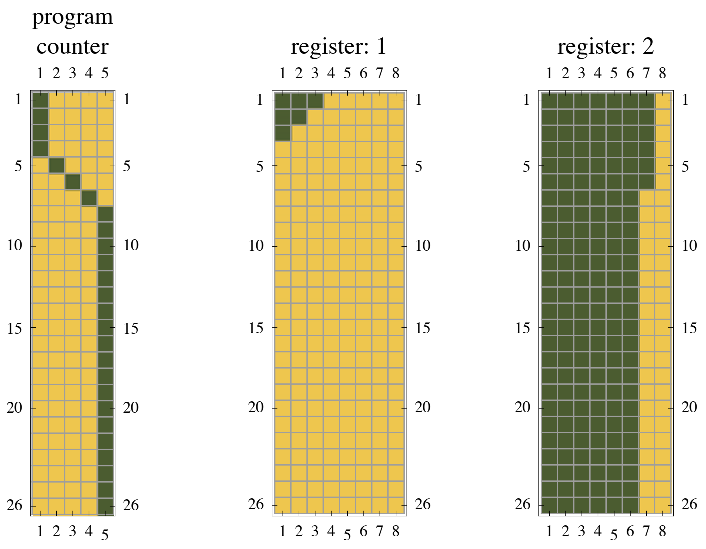

Figure D7. Functional behavior example: program number 4720's registers with an initial program counter value of 1, register 1's initial value of 3 bits, and register 2's initial value of 7 bits. The result of decrementing register 2's value and clearing register 1 is that register 1's final value is 0 and register 2's final value is 6.

D.2.3 The Multiplication Operation

Figures D8 and D9 are examples of register machines that perform multiplication operations on register 2 by storing the results in register 1.

The value of register 1 can also be incremented twice by initializing the program counter to 1 instead of 4 (see Figure D10).

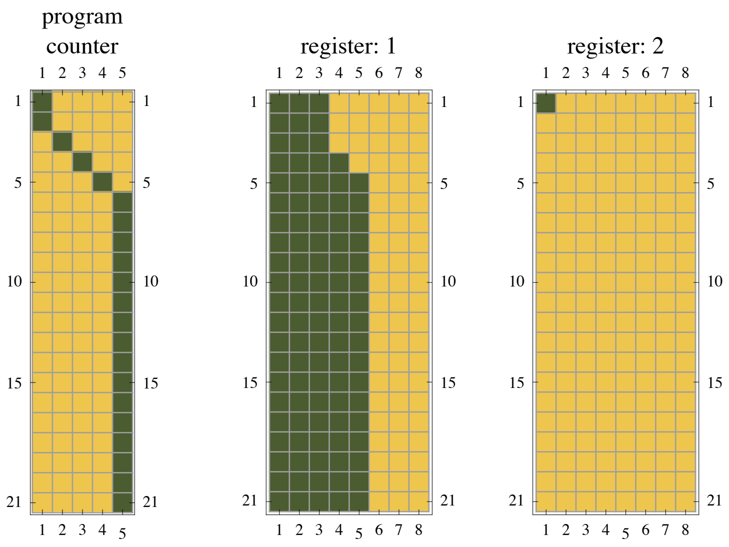

Figure D8. Program number 3881's registers with an initial program counter value of 2, register 1's initial value of 3 bits, and register 2's initial value of 1 bit. The result of multiplying 1 more than register 2's value by 2, adding the result in register 1, and clearing register 2 is that register 1's final value is 5 and register 2's final value is 0.

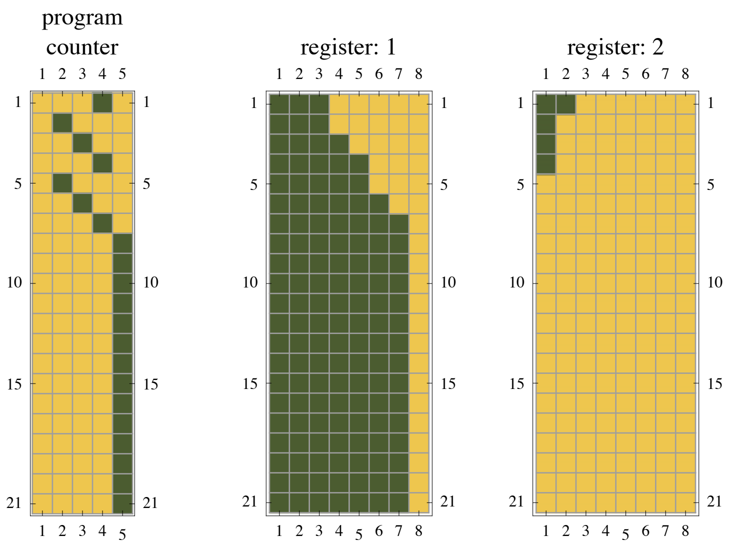

Figure D9. Program number 3881's registers with an initial program counter value of 4, register 1's initial value of 3 bits, and register 2's initial value of 2 bits. The result of multiplying register 2's value by 2, adding the result to register 1, and clearing register 2 is that register 1's final value is 7 and register 2's final value is 0.

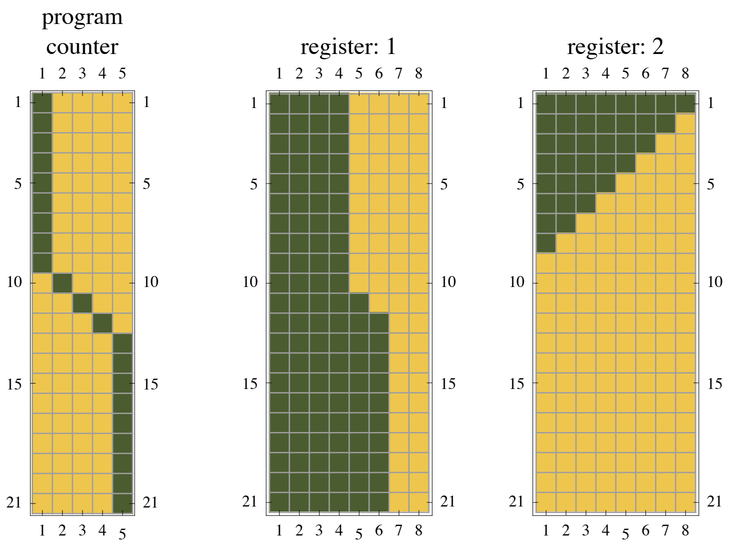

Figure D10. Program number 3881's registers with an initial program counter value of 1, register 1's initial value of 4 bits, and register 2's initial value of 8 bits. The result of incrementing register 1's value twice, adding the result to register 1, and clearing register 2 is that register 1's final value is 6 and register 2's final value is 0.

References

[1] S. Wolfram, *A New Kind of Science*, Champaign IL: Wolfram Media, Inc, 2002.

[2] P. Chapman. "Minsky Register Machine." (Jan 13, 2003) http://www.igblan.free-online.co.uk/igblan/ca/minsky.html.

[3] A. Joseph. "Register Machines" from the Wolfram Demonstrations Project—A Wolfram Web Resource. http://demonstrations.wolfram.com/RegisterMachines.

[4] H. Massalin, "Superoptimizer: A Look at the Smallest Program" in *Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS87)*, Berkeley, CA (R. Katz, ed.), Los Alamitos, CA: IEEE Computer Society Press, 1987 pp. 122–126. doi:10.1145/36177.36194.

[5] S. Bansal and A. Aiken, "Automatic Generation of Peephole Superoptimizers," *ACM SIGPLAN Notices*, **41**(11), 2006 pp. 394–403. doi:10.1145/1168918.1168906.

[6] J. J. Joosten, F. Soler-Toscano, and H. Zenil, "Program-Size versus Time Complexity Slowdown and Speed-up Phenomena in the Micro-cosmos of Small Turing Machines," *International Journal of Unconventional Computing*, 7(5), 2011 pp. 353–387.