A Programming Language Embedded in *Magic: The Gathering*

Howe Choong Yin
Independent Researcher, Singapore

Alex Churchill
Independent Researcher, Cambridge, United Kingdom

Abstract

Previous work demonstrated that the trading card game *Magic: The Gathering* is Turing complete, by embedding a universal Turing machine inside the game. However, this is extremely hard to program, and known programs are extremely inefficient. We demonstrate techniques for disabling *Magic* cards except when certain conditions are met, and use them to build a microcontroller with a versatile programming language embedded within a *Magic* game state. We remove all choices made by players, forcing all player moves except when a program instruction asks a player for input. This demonstrates *Magic* to be at least as complex as any two-player perfect knowledge game, which we demonstrate by supplying sample programs for Nim and the Collatz conjecture embedded in *Magic*. As with previous work, our result applies to how real *Magic* is played, and can be achieved using a tournament-legal deck; but the execution is far faster than previous constructions, generally one cycle of game turns per program instruction.

2012 ACM Subject Classification Theory of computation → Representations of games and their complexity

Keywords and phrases Programming, computability theory, Magic: the Gathering, two-player games, tabletop games

Digital Object Identifier 10.4230/LIPIcs.FUN.2024.18

Category FUN with Algorithms

Related Version *Full Version*: `https://www.toothycat.net/~hologram/Magic/Magic_Microcontroller_full.pdf`

1 Introduction and Previous Work

Magic: The Gathering (also known as *Magic*) is the world's largest tabletop collectible card game, played by hundreds of thousands of players in tournaments and by millions more players casually. In 2020, Churchill, Biderman & Herrick published an embedding of a universal Turing machine inside *Magic* [1]. This is the first widely played tabletop game to be shown Turing complete in the format in which it is usually played, as opposed to some infinite generalisation. For example, chess is EXPTIME-complete with infinite board and pieces, but has a finite number of states in the 8x8 board used for tournament play. Churchill et al. showed that the question "will this *Magic* game ever terminate" cannot be answered in the general case, even for two-player *Magic* played with all the usual tournament restrictions. However, this paper did not contain any concrete example computations.

The author of [2] investigated the runtime performance of this Universal Turing Machine (UTM) embedded within *Magic*. He established a compilation sequence from an arbitrary Turing machine with N states, into a 2-tag system, into the 2-state UTM(2,18), and thence into *Magic*. He supplied a simple Turing machine to compute 2+3 in a unary adder. However, he found that no simulation was able to establish how long it would take to compute 2+3 in the UTM. The straightforward compiler's output from this simple Turing machine for

computing 2+3 results in a tape over 40 million symbols long (15 million in the program and 25 million in the data). The UTM simulation of the Turing machine needs to constantly move between the program and data sections, resulting in absurdly inefficient computation times. No simulation was able to establish how long it would take to compute 2+3 in the *Magic* game.

The same author created an optimised unary adder Turing machine with just 2 states, and compiled a simplified computation of 1+1. This completed in a mere 3,958,876,878 game cycles (of each of the two players taking a turn). Even after nearly 8 billion game turns, because of the multiple steps in translating the calculation into the UTM(2,18), the output consisted of hundreds of creature tokens, which needed to be interpreted by carefully counting how many tokens of type Myr were mixed in among the hundreds of tokens with type Aetherborn, all in order to retrieve the output of "2".

In this paper we set out a different construction, which embeds a full microcontroller within *Magic: The Gathering*. In Section 2 we describe at a high level some key features of the construction, and Section 3 specifies the programming language supported by the microcontroller. Section 4 provides the details of we implement the framework, and we work through an example instruction in Section 5. Section 6 contains our conclusions and discussion of the implications. Finally, Appendix A provides several sample programs, Appendix B explains which cards we use to modify other cards, and Appendix C supplies a decklist that could be brought to a *Magic* tournament to assemble the construction.

2 Outline of the Construction

The rest of this paper describes a construction that simulates a fully general programming language within *Magic: The Gathering*. Compared to [1] this is also Turing complete, but is much more efficient and easy to program, and reports its outputs much more clearly. It also allows reading input from each of the players of the two-player *Magic* game in which it is embedded, and can be programmed to terminate in a win for either player or a draw, or of course can keep running indefinitely.

As with the Turing machine construction, we start by assuming one player, Alice, draws a combination of cards that allows her to take over the game, draw all the rest of her deck, and remove all cards from the hand of the opponent, Bob. After the initial setup is completed, she removes all player choices, so that neither player has any option but to let the program execution continue, short of conceding the game. Thus the outcome of this tournament-legal *Magic* game is entirely determined by the result of the program.

The program is written in a language of 12 symbols, represented by basic land cards, which are allowed to occur any number of times in a player's deck. Alice's deck needs to start off containing a lot of other cards, but once the microcontroller is set up and she has drawn all her cards, she returns to her deck a sequence of cards from among these 12, which encodes the program to be executed.

The program is read one card at a time. During Alice's combat step we put one of these cards, which we call the "program permanent", onto the battlefield for long enough to read it, then move it to the bottom of Alice's deck. We in fact make Alice have three combat steps each turn, and in each one, a program permanent is read and possibly other game actions are (automatically) taken. Each instruction in the programming language is a sequence of three symbols, interpreted by some set of "instruction permanents" that do extra things during Alice's or Bob's turn.

All instruction permanents that don't apply to the current instruction are "inactive",

i.e. have all their abilities removed. To do this we turn them into creatures with a certain creature type (e.g. Angel), have a card which makes all Angels also Saprolings, make all Saprolings lands, and remove abilities from all lands. Crucially, we can conditionally allow a certain group of instruction permanents to regain their abilities by temporarily "phasing out" the card which makes (e.g.) Angels into Saprolings. (Objects in *Magic* which are "phased out" are treated by the rules as if they don't exist.) We do this during one of Alice's combat steps, so those instruction permanents have their abilities during the rest of Alice's turn (including her later combat steps) and all of the other player Bob's turn.

Permanents in *Magic* have zero or more colours, from the set of white, blue, black, red and green. Permanents that are creatures have zero or more creature types, also drawn from a well-defined set, but this set has over 200 types in it. We use both of these characteristics extensively. All "inactive" permanents are made green Saproling creatures, and all green creatures are given protection from certain creature types. We make extensive use of the capabilities *Magic* offers to edit existing cards by changing colour words (using the card **Mind Bend**), creature type text (using **Artificial Evolution**), colours (using **Prismatic Lace**), and creature types (using a variety of cards according to circumstances).

In particular, we carefully apply restrictions so that any time an ability triggers that would normally let its controller choose a target, there is precisely one legal target (or occasionally no targets, which prevents the ability from going on the stack at all). We ensure that all creatures attack and block where possible using **Grand Melee**, but make most creatures unable to attack or block using **Stormtide Leviathan**. Any time a creature is able (and thus forced) to attack, we arrange that either it can't be blocked at all, or there is precisely one creature forced to block it.

3 The Programming Language

The programming language we implement has the following features:

- Twelve registers $r_0 \dots r_{11}$, each able to contain an arbitrarily large nonnegative integer.
- An unlimited number of memory slots, each addressed by a nonnegative integer address; each memory slot can hold a single arbitrarily large nonnegative integer.
- A single Boolean flag that is set by certain instructions such as comparisons. The flag can be read by certain instructions; most notably, jump instructions can be made conditional on whether the flag is true or false.

The program is written in a language of 12 symbols, and each instruction is a sequence of three symbols. For example, the sequence 0 1 2 (represented by cards Plains Island Swamp) encodes the instruction "Add 1 2", which will result in increasing the value of register r_1 by the value of r_2. We provide the following instructions in the language:

5 Y Z $(Y \neq Z)$	Set r_Y r_Z	Set r_Y to the value of r_Z.
5 Y Y	Zero r_Y	Set r_Y to zero.
0 Y Z	Add r_Y r_Z	Set r_Y to $r_Y + r_Z$.
4 1 Z	Add1 r_Z	Set r_Z to $r_Z + 1$.
4 2 Z	Halve r_Z	Set r_Z to half the value of r_Z, rounding down. Set the flag to the remainder from the division.
1 Y Z $(Y \neq Z)$	SubCond r_Z r_Y	If $r_Z \geq r_Y$, set r_Z to $r_Z - r_Y$ and set the flag to 0. Otherwise, set the flag to 1.

1 Z Z	Sub1Cond r_Z	If $r_Z \geq 1$, set r_Z to $r_Z - 1$ and set the flag to 0. Otherwise, set the flag to 1.
6 Y Z	Mult r_Z r_Y	Set r_Z (note, not r_Y) to $r_Y \times r_Z$.
7 Y Z	DivCeil r_Y r_Z	Set r_Y to $\lceil r_Y / r_Z \rceil$. If the division was exact, set the flag to 0, otherwise set it to 1. If $r_Z = 0$ or $Y = Z$, this is undefined behaviour.
2 Y 0	AInput r_Y	Set r_Y to a nonnegative integer of Alice's choice.
2 Y 1	BInput r_Y	Set r_Y to a nonnegative integer of Bob's choice.
2 Y 2	SetF r_Y	Set r_Y to the flag's value.
2 Y 3	SetNF r_Y	Set r_Y to the Boolean negation of the flag's value (1 if it's 0 and vice versa).
2 Y 4	Rand6 r_Y	Set r_Y to a random nonnegative integer less than 6.
2 Y 5	Rand20 r_Y	Set r_Y to a random nonnegative integer less than 20.
10 Y Z	NumBuild $12Y+Z$	Set r_0 to $12Y+Z$, except if the last instruction that was executed was also a NumBuild instruction, in which case multiply r_0 by 144 and add $12Y+Z$ to it. As its name suggests, this instruction can be used repeatedly to build any nonnegative integer value in r_0, two base-12 digits at a time.
8 Y Z	Store r_Z r_Y	Store the value of r_Y at memory address r_Z.
9 Y Z	Load r_Y r_Z	Load the value at memory address r_Z into r_Y.
3 0 Z'	JumpFwd Z'	Jump forward by Z' instructions. If $Z = 0$, Z' is r_0, and otherwise, Z' is Z. Thus instead of a useless command to jump 0 instructions, we gain the ability to jump an arbitrary or computed distance.
3 1 Z'	JumpBwd Z'	Jump backward by Z' instructions. Backwards jumps by more than the length of the program[1] do nothing.
3 2 Z	JumpFwdNF Z'	Jump forward by Z' instructions if the flag is 0/false.
3 3 Z	JumpBwdNF Z'	Jump backward by Z' instructions if the flag is 0/false.
3 4 Z	JumpFwdF Z'	Jump forward by Z' instructions if the flag is 1/true.
3 5 Z	JumpBwdF Z'	Jump backward by Z' instructions if the flag is 1/true.
3 6 0	CallFwd r_0	Call a function r_0 instructions ahead: Jump forward $3r_0$ cards and push $P - 3r_0$ onto the return stack. P is the length of the program in cards[1]. If $r_0 = 0$ or $3r_0 > P$, this is undefined behaviour.
3 7 Z	Return Z'	Return from a function Z' instructions long: Pop a value S from the return stack, and jump forward $max(0,\ S-3Z')$ cards. If the return stack was empty, end the game in a draw.
3 6 1	CallBwd r_0	Call a function r_0 instructions behind: Jump backward $3r_0$ cards and push $3r_0$ onto the return stack. If $3r_0 \geq P$, this is undefined behaviour. May not be used to call a function from within itself.

[1] If the program is less than 6 cards long, P is the first multiple of the length that is at least 6.

3 6 2	CallBwdR r_0	Call a function r_0 instructions behind (direct-recursion-capable): Same as CallBwd, but push $P + 3r_0$, so that this may be used to call a function from within itself.
11 Y Z ($Y \neq Z$)	FLess r_Z r_Y	Set the flag to 1 if $r_Z < r_Y$, 0 otherwise. Flag-combining.
11 Z Z	FIsZero r_Z	Set the flag to 1 if $r_Z = 0$, 0 otherwise. Flag-combining.
4 0 0	HaltD	End the game in a draw.
4 0 1	HaltA	End the game with Alice winning.
4 0 2	HaltB	End the game with Bob winning.

"Flag-combining" is a property used by some instructions whose only purpose is to set the flag. It means that flag values given by subsequent instructions are combined by logical OR instead of replacing the flag's value. This state ends when any of the following is executed:

- Any instruction that uses the flag's value.
- Any jump instruction (including calls and returns, and regardless of whether the jump is taken or not).

For example, this can be used to check whether r_0 and r_1 are equal, with FLess r_0 r_1 followed by FLess r_1 r_0.

The program is cyclic, which is to say it wraps around: after passing the final instruction (by executing it or jumping past it) execution continues with the first instruction. Similarly you can jump backwards beyond the start of the program and end up near the end of the program.

Encountering a sequence that does not match any of the instructions above is undefined behaviour.

These instructions are a superset of those required for a random-access register machine such as Melzak's Q-machine, which is Turing complete.[3]

4 Implementation of the Microcontroller

In this section we describe the various gadgets that make up the microcontroller as described in the previous section. We defer discussion of how to set up the board state (including modifying card types, creature types, colours, etc) to Appendix B.

Note that this PDF contains tooltips: hover over a bold card name to see its full rules text, and hover over a dashed underline to see how specific modifications are accomplished or reminders of how creature types are used.[2]

4.1 The program

The program is a sequence of the following cards: **Plains**, **Island**, **Swamp**, **Mountain**, **Forest**, **Wastes**, **Snow-Covered Plains**, **Snow-Covered Island**, **Snow-Covered**

[2] These display in Adobe Reader but may not display in web browsers. Readers using a web browser may like to read the details at [4] instead, where we provide more detailed tooltips.

Swamp, **Snow-Covered Mountain**, **Snow-Covered Forest**, and **Snow-Covered Wastes**[3]. We call these cards "symbol cards" and assign them numbers 0, 1, 2, ..., 11 in the order listed.

Most of the time, one of the cards in the program will be on the battlefield under Alice's control; we call this card the *program permanent*. The rest of the cards will be in Alice's library, with the next symbol in the program on top of the library, continuing in program order from top to bottom, then continuing from the start of the program until the symbol before the current symbol.

The program is made up of instructions that each consist of 3 cards. X, Y, and Z shall refer to the numbers of the three cards that form the instruction currently being executed, in that order. For example, the instruction "Add r_1 r_2" is represented by 0 1 2 (Plains Island Swamp), and X is 0, Y is 1 and Z is 2.

4.2 Global environment control

On the battlefield is a **Grand Melee** ("All creatures attack each combat if able. All creatures block each combat if able") and a **Stormtide Leviathan** ("Creatures without flying or islandwalk can't attack"). Most creatures are unable to attack; when we want a creature to be able to attack, we will give it flying or islandwalk. Bob's creatures with islandwalk will be unblockable because Alice controls the program permanent which is given type Island.

Both players have life total 1, and each has a **Worship** keeping their life totals at 1 through the damage they will be dealt. (Note that **Worship** only modifies the *result* of damage; the damage itself is still dealt, so effects triggered on combat damage still trigger.)

4.3 Advancing through the program

Alice has a **Vaevictis Asmadi, the Dire**, whose rules text reads "Whenever Vaevictis Asmadi, the Dire attacks, for each player, choose target permanent that player controls. Those players sacrifice those permanents. Each player who sacrificed a permanent this way reveals the top card of their library, then puts it onto the battlefield if it's a permanent card.".

We use the techniques in Appendix B to make **Vaevictis Asmadi, the Dire** into a 1/1 Sliver Beast Reflection. It attacks in Alice's combat phase. Its ability is forced (see subsection 4.6) to target the program permanent for Alice, and one particular permanent for Bob. When it resolves, Alice sacrifices the program permanent, and **Wheel of Sun and Moon** enchanting Alice sends it to the bottom of Alice's library; then the next card (on top of Alice's library) is put onto the battlefield, becoming the new program permanent. Meanwhile, Bob has been given a **Tajuru Preserver** stopping Bob's permanent from being sacrificed, and thus Bob does not reveal the top card of his library.

Alice controls **Tetsuko Umezawa, Fugitive** ensuring that **Vaevictis Asmadi, the Dire**, and all her other creatures with power or toughness 1 or less, can't be blocked.

4.4 Disabling and conditionally enabling permanents

For each of the 12 program cards, letting n be its number, we choose a creature type X_n. X_0 is Aetherborn, X_1 is Beeble, and so on through Cephalid, Drake, Eldrazi, Faerie, Gremlin, Homarid, Illusion, Juggernaut, Kavu, and Lhurgoyf.

[3] To be released on 7th June 2024. To play the Microcontroller before that date, instead use Persistent Petitioners and Infinite Reflection as described at [4].

We have Alice control twelve token copies of **Dralnu's Crusade**, whose printed rules text reads "All Goblins get +1/+1. All Goblins are black and are Zombies in addition to their other creature types." However we use the techniques in Appendix B to edit each one to instead to affect creatures with a different one of the 12 creature types X_n ($n \in \{0, 1, \ldots, 11\}$), and add type Saproling. **Life and Limb** makes Saprolings into lands, and also makes them green, and then **Blood Sun** removes their abilities. So all creatures with any of the 12 creature types X_i have their abilities removed.

Each of these **Dralnu's Crusade**s is made into a creature with creature type Sliver, with power and toughness 2/1, and given flying. For each of the 12 basic land cards, the corresponding **Dralnu's Crusade** is equipped with a **Strata Scythe** imprinted with that basic land card ("Equipped creature gets +1/+1 for each land on the battlefield with the same name as the exiled card").

All twelve **Dralnu's Crusade**s are forced to attack, and the one corresponding to the current program permanent will be boosted to 3/2; the previously mentioned **Tetsuko Umezawa, Fugitive** ensures that the others can't be blocked.

Bob controls a **Dream Fighter** ("Whenever Dream Fighter blocks or becomes blocked by a creature, Dream Fighter and that creature phase out") with creature type Sliver and granted reach (so it can block creatures with flying). This blocks the 3/2 **Dralnu's Crusade**, and they both phase out, so that creatures of the type corresponding to the current program permanent are no longer made Saprolings. We have a **Shadow Sliver** ensuring that only Slivers can block these **Dralnu's Crusade**s.

Figure 1 After Vaevictis Asmadi, the Dire puts a Wastes onto the battlefield, the only one of Alice's attacking Dralnu's Crusades which Bob's Dream Fighter can block is the one given +1/+1 by a Strata Scythe imprinted with Wastes.

This reads the first card of an instruction; creatures of the corresponding type will regain their abilities after **Dream Fighter**'s ability resolves. We also choose a thirteenth creature type (Monkey) denoted X_A, and have another **Dralnu's Crusade** making X_A creatures also Saprolings, but this one is not a creature; rather we make it an artifact so **Bludgeon Brawl** makes it an Equipment, and attach it to Bob's **Dream Fighter**, so it will phase out when the **Dream Fighter** does. This means that creatures of type X_A get to regain their abilities after the first card is read, no matter what it is, but they will lose their abilities again when Bob's turn starts and the **Dream Fighter** phases back in.

We will also often want to use this conditional mechanism on noncreature permanents; to do that, we make token creature copies of them using **Urza, Prince of Kroog**, combined with **Memnarch** if necessary.

4.5 Reading the second and third cards

Alice controls a **Bloodthirster** ("Whenever Bloodthirster deals combat damage to a player, untap it. After this combat phase, there is an additional combat phase. Bloodthirster can't attack a player it has already attacked this turn.") It is made a 1/1 Sliver Beast Reflection and given double strike. This also attacks, can't be blocked because of **Tetsuko Umezawa, Fugitive**, and adds a second combat phase. The second combat phase is used to read the second card, using another copy of the above setup, but with all the creatures involved granted an additional creature type of X_A so that they do not attack or block in the first combat phase (because they don't have their abilities at that point). So there is a second **Vaevictis Asmadi, the Dire** to advance through the program, given types Sliver Beast Reflection X_A; a second **Dream Fighter** with reach and types Sliver X_A; and another batch of **Dralnu's Crusade**s for another 13 creature types $Y_0, Y_1, \ldots, Y_{11}$, and Y_A (Antelope, Basilisk, Camel, Dauthi, Efreet, Fox, Gnome, Hippo, Inkling, Jellyfish, Kor, Lammasu, and Metathran), the first 12 having **Strata Scythe**s.

Similarly, since the **Bloodthirster** has double strike, it will also give Alice a third combat phase, which is used by another copy of the setup (with creature type Y_A) to make another 13 creature types $Z_0, Z_1, \ldots, Z_{11}$, and Z_A conditional on the third card of the instruction (Aurochs, Brushwagg, Camarid, Druid, Elephant, Ferret, Graveborn, Hamster, Imp, Jackal, Kithkin, Licid, and Masticore).

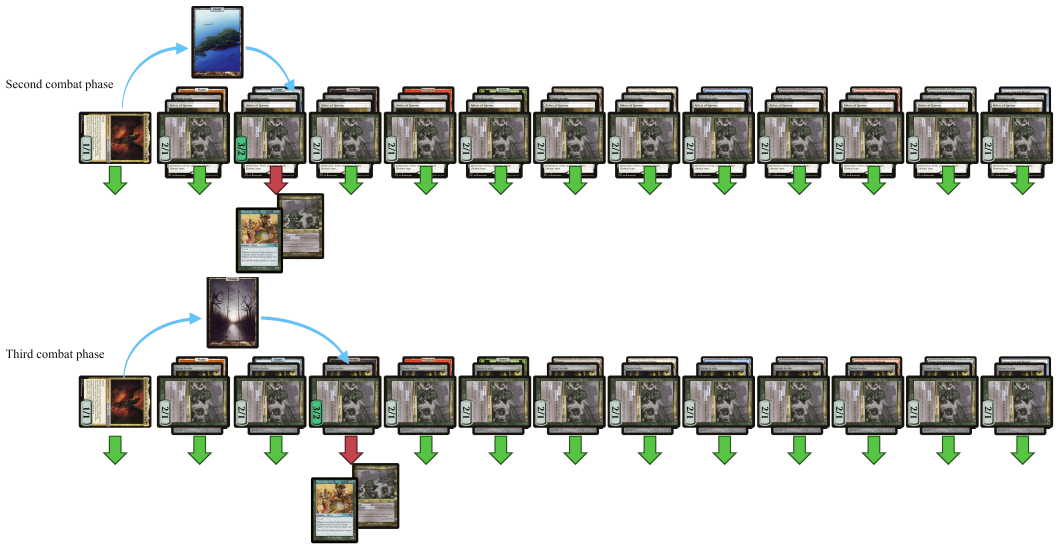

Figure 2 More Dralnu's Crusades attack in the second and third combat phases, and another of each is blocked and phases out. They couldn't attack before because of their types X_A or Y_A.

4.6 Constraining targets

Almost all creatures will be green; any creature that isn't naturally green and isn't specified to be differently coloured is made green by **Prismatic Lace**. A few creatures will be red or blue instead, but even those are made green while inactive by **Life and Limb**.

A **Masked Gorgon** edited to give green and blue creatures protection from Reflections means that only red creatures are legal targets for Reflections' abilities. Similarly, a **Masked Gorgon** edited to give green and red creatures protection from Beasts means that only blue creatures are legal targets for Beasts' abilities.

We will refer to the creature types Reflection and Beast as tR and tU respectively, indicating their function. ("U" is the usual abbreviation for "blue" in *Magic*.)

As mentioned earlier, each **Vaevictis Asmadi, the Dire** has been made both a Beast and a Reflection, so that green creatures, blue creatures, and red creatures are all illegal targets for it; the intended target under Alice's control is a noncreature land. The same types are applied to the **Bloodthirster** for a different purpose, to stop it from blocking on Bob's turn (which is something else protection does), because it untaps itself and stays untapped (as its own ability prevents it attacking in the later combat phases).

Spectral Guardian makes noncreature artifacts illegal targets for anything. Alice and Bob both control a **Sterling Grove** to make other enchantments illegal targets for anything. Alice's is made an artifact so it gains shroud from **Spectral Guardian**. Bob's one does not itself have shroud, and thus it is the only legal target under Bob's control for **Vaevictis Asmadi, the Dire**, but Bob has a **Tajuru Preserver** so he does not sacrifice anything. We do however give Bob's **Sterling Grove** protection from blue, which will be useful later to stop some other things from targeting it.

Alice and Bob both have **Ivory Mask**, making both players illegal targets for anything.

4.7 Order of continuous effects, and one more of them

The *Magic* Comprehensive Rules [5] specific a system of "*layers*" for working out what happens when multiple effects apply to the same permanent. For example, effects that make one permanent a copy of another object apply in layer 1. All effects that change a permanent's type (such as creature, land, etc) or subtype (Angel, Goblin, etc) apply in layer 4. Anything that adds or removes abilities applies in layer 6, and so on. Within a layer, if multiple effects try to affect the same permanent, each object or effect has a "*timestamp*", generally when that object or effect was created. Within this document, we denote timestamps with circled numbers: an effect with timestamp ① will take effect earlier than timestamp ②.

The continuous effects mentioned so far are timestamped as follows:

①: **Stormtide Leviathan**
②: **Dralnu's Crusade**s and **Blood Sun**
③: **Life and Limb** and protection-granting and shroud-granting effects

Any other continuous effect is timestamped ① unless otherwise stated.

The exception to timestamp order is "*dependency*": if two effects would apply within the same layer, but one will change the existence of the second or which objects the second acts on or what it does to them, the first one applies first even if the second has an earlier timestamp. This applies to our construction where abilities that will be removed by the **Blood Sun** wait for it to be applied (and thus end up not being applied themselves).

The program permanent is a land so it is granted type Island by the **Stormtide Leviathan**, but it may be a Forest as well. We do not want **Life and Limb** to make the program permanent a creature. So we also have Alice control an **Illusionary Terrain**, made a creature with type Z_A, within timestamp ① after the **Stormtide Leviathan**, turning all Islands to Islands. This is not as ineffective as it sounds: rule 305.7 [5] says that this removes all other types, so that the program permanent's subtype is set to Island and no others. Note that, because there are no Saprolings before ②, the **Stormtide Leviathan** does not have a dependency on the **Life and Limb**.

By being made a Z_A, this **Illusionary Terrain** has its abilities removed by **Blood Sun** most of the time, most importantly during Alice's upkeep. Because the effect of setting basic lands' types is applied in layer 4, before the abilities are removed in layer 6, it still

functions despite the abilities being removed. However, the removal of abilities does shut off its cumulative upkeep.

4.8 Registers

There are twelve registers named r_0, r_1, …, r_{11}, each of which holds a nonnegative integer value.

Each register r_n is a token copy of **Joraga Warcaller** under Bob's control, given copiable creature type Z_n and additionally given creature type Rabbit, given indestructible and vigilance, coloured red at timestamp ④, with base power and toughness noncopiably set to 2/2 but with two –1/–0 counters, and a number of +1/+1 counters on it equal to its register value. (As always, see Appendix B or the tooltips for how all these changes are accomplished.) **Joraga Warcaller**'s rules text says "Other Elf creatures you control get +1/+1 for each +1/+1 counter on Joraga Warcaller". Thus, after Z is read from the program, the active register r_Z regains its ability and adds its value to the power and toughness of each other Elf that Bob controls. (This does not include the other registers; their Elf type is overwritten.)

Note that each register's power is equal to its value, whether it is r_Z (having base power 2 and two –1/–0 counters) or not (having base power 1 from the **Life and Limb**, +1 power from a **Dralnu's Crusade**, and two –1/–0 counters).

All the registers are made red by **Prismatic Lace** at timestamp ④, later than that of the **Life and Limb**, so that they are always red even when inactive. But the inactive registers are still Saprolings even though they're red.

r_0 also has Rhino added to its creature types; this will be useful for some instructions that use specifically this register.

Figure 3 The first three registers. In this case r_0 has value 3, r_1 value 0 and r_2 value 2. If $Z = 0$, the Dralnu's Crusade making Aurochs into Saprolings is phased out and Bob's Elves get +3/+3.

For each $n \in \{0, 1, \ldots, 11\}$, Bob has a **Riders of Gavony** giving Z_n creatures protection from Yetis. Each of these is made into a noncreature artifact with mana value 0 and attached to the **Dralnu's Crusade** that applies to Y_n, so that they phase out together. As a result, this means that each register except for r_Y has protection from Yetis. Then, a creature can be given the types Reflection and Yeti so that it can only target r_Y; we call this combination tr_Y.

Bob also has another **Riders of Gavony** giving Saprolings protection from Zubera creatures. This means the inactive registers (those other than r_Z) can't be targeted by any creature that's a Zubera. Then, as above, creature types Reflection and Zubera together mean that a creature can only target r_Z; we call this combination tr_Z.

4.9 Memory

We provide an unlimited number of memory slots, each addressed by a nonnegative integer address. Each memory slot can hold a single arbitrarily large nonnegative integer.

A nonzero value V at a memory address A is represented by a Mouse token with base power and toughness V/V under Alice's control with A +1/+1 counters on it. A zero value is represented by an absence of such a token.

4.10 The flag

There is a Boolean flag that some instructions use. It is represented by a card, being in Bob's library for 0/false and in Bob's hand for 1/true; this card must not have any abilities that function in those zones other than characteristic-defining abilities. We assume that Bob has at least one such card in his deck (most lands, planeswalkers, instants and sorceries would be suitable). We remove all other cards from Bob's hand, library and graveyard.

Wheel of Sun and Moon enchanting Bob allows us to set the flag to 0 by making Bob discard the card. We give Bob a **Tomorrow, Azami's Familiar**, allowing us to set the flag to 1 by making Bob draw a card, while not making Bob lose the game if the flag was already 1.

To stop Bob from playing this card, we give Bob a **Nevermore** and/or an **Aggressive Mining** made into an artifact as appropriate.

4.11 Further environment control

To prevent any player choice involving the program permanent's abilities, we use **Root Maze** to make each new program permanent enter the battlefield tapped and **Choke** to keep them tapped (recall that they are all made Islands). Also, Bob's **Sterling Grove** has its activated ability shut off by **Suppression Bonds** attached to it. **Stony Silence** and **Cursed Totem** shut off activated abilities of artifacts and creatures.

Both players control a copy of **Recycle**, skipping both player's draw steps. **Mirror Gallery** disables the "legend rule". And we give both players a **Corrosive Mentor** so that black creatures controlled by either player have wither.

4.12 Instructions

Sadly length constraints prevent us including here the details of how each instruction is implemented. See [6] for the full implementation details.

5 Example Instruction

For demonstration purposes, here is how an example turn cycle looks. Let us say the next three cards on the top of Alice's library are Wastes, Plains, Swamp. This triplet encodes symbols 5 0 2, a Set instruction.

At the start of Alice's turn, most creatures are Saprolings and therefore have no abilities. Recall that all creatures are forced to attack and block where able, but only creatures with

flying or islandwalk are allowed to attack. In Alice's first combat phase, twelve flying 2/1 **Dralnu's Crusade**s attack along with the **Bloodthirster** and **Vaevictis Asmadi, the Dire**, whose ability puts the Wastes onto the battlefield. This makes the **Dralnu's Crusade** affecting X_5 get +1/+1 from its **Strata Scythe**, and so it gets blocked by Bob's first **Dream Fighter** and phases out. Creatures with type X_5 or X_A regain their abilities (unless they also have another type making them a Saproling such as Y_n).

In the second combat phase (granted by **Bloodthirster**), another **Vaevictis Asmadi, the Dire** and twelve more **Dralnu's Crusade**s attack, as their type X_A is no longer causing their flying ability to be removed. This Vaevictis's ability puts the Wastes onto the bottom of Alice's library and the next card of the program in its place, the Plains. The **Dralnu's Crusade** affecting Y_0 gets +1/+1 from its **Strata Scythe** and gets blocked by Bob's second **Dream Fighter**, whose type X_A is no longer having its reach ability removed. Creatures with type Y_0 or Y_A regain their abilities.

In the third combat phase, Alice's **Archpriest of Iona** with types X_5 Y_A tr_Y Cleric has finally regained its abilities. Its ability triggers, and is forced to target r_0, because its types $\underset{\cdots}{tr_Y}$ mean it can't target any green or blue creatures or any of the other registers. r_0 gains flying, so it'll be able to block, and gets a temporary +1/+1.

When the third set of **Dralnu's Crusade**s and the third **Vaevictis Asmadi, the Dire** attack, Alice's **Shape Stealer** with types X_5 Y_A is also forced to attack. The **Dralnu's Crusade**s all have shadow, so r_0 can't block any of them; the only creature r_0 can block is the **Shape Stealer**. **Shape Stealer**'s ability gives it base power equal to r_0's value +1, which is why it has the –1/–0 counter so its actual power is r_0's value. It is given $\underset{\cdots}{\text{wither}}$ because it is black, so the damage is dealt as –1/–1 counters, cancelling out all the +1/+1

Figure 4 The five steps of instruction 5 0 2, Set r_0 r_2.

counters on the register and setting r_0's value to 0.

In Bob's combat phase, **Halana and Alena, Partners** triggers. Because it is an Elf, its power is equal to r_2's value. And because it has types tr_Y as well, just like with Alice's **Archpriest of Iona**, the only legal target for its trigger is r_0. So it adds r_2 +1/+1 counters to r_Y. Then nothing else happens on the rest of Bob's turn, and we move back to Alice's turn, when the three copies of **Vaevictis Asmadi, the Dire** will read three more cards from the program.

6 Implications and Conclusion

6.1 Readability and programmability

In sharp contrast to the impenetrable millions of tokens produced by the Turing machine in [1], the game state will be clearly readable when a program in this construction terminates. After the computation of sample program 1 "Calculate 10 cubed" (see Appendix A), there will be one Mouse creature token with power and toughness 1000/1000. When sample program 2 "Prime Factors" halts, for each prime factor of the input number, there will be one Mouse creature token with power and toughness equal to that factor. When sample program 4 "Nim" halts, the result of the *Magic* game will be victory for Alice or Bob according to who won the embedded game of Nim.

The programming language provided is comparable to other microcode programming languages and assembly languages. It has some quirks but is perfectly usable to write moderate-sized programs. Readers are invited to write their own programs in the simulator we wrote to test the sample programs [7].

6.2 Tournament playability

The construction uses many different *Magic* cards, far more than are normally included in tournament decks. But it is legal to bring a deck with more than 60 cards to a tournament; players sometimes play decks with over 200 cards [8]. The only restriction is that you must be able to physically shuffle the deck in a reasonable amount of time [9].

Appendix C contains a decklist of a 360-card deck which could be brought to a Legacy tournament. The deck's composition breaks down as 160 land cards to be used for the program; 136 distinct named cards used in the microcontroller; 40 cards used during setup to edit the text and characteristics of the cards used on the microcontroller; and 24 cards used to generate an unbounded amount of mana, draw all the remaining cards, set up the construction and remove all Bob's cards.

With the correct draw, a player can take control of the game as early as the first turn, and set up the construction. Getting that correct draw is much less likely than with a 60-card deck, but this is a theoretical result anyway; the difference between a one in a million chance and a one in several trillion is not particularly relevant.

There are minimal constraints on Bob's deck (one card to serve as the flag must have no abilities that function in the graveyard or hand), which will easily be satisfied by any normal deck. So it is perfectly possible for a hapless player to sit down expecting a tournament *Magic* game, have the opponent take over and set up the Microcontroller, and find that they can only win the game by winning (say) a game of chess instead.

6.3 Computational implications

The previous construction in [1] was Turing complete, so this does not increase the amount of computation possible inside *Magic*. However, the addition of input commands during program execution adds a lot to the programs that can be usefully written, in terms of ability to simulate multi-player games involving choices – see e.g. sample program 4 which implements Nim. The language is clearly powerful enough to similarly write programs for chess, checkers, go, or any similar two-player perfect knowledge game.

A common joke upon the publication of [1] was "Now we can write *Magic Online* [a digital implementation of *Magic*] in *Magic*". With the Turing machine-based construction, all players would have had to pre-register all their moves before computation started. By contrast, if a digital card game were implemented using the construction in this paper, players could choose their moves during gameplay in response to the moves made by their opponent.

Similarly, this result shows that the complexity of *Magic* includes any game or algorithm which involves a finite (but potentially unbounded) sequence of choices, including responses to another player's choices. It is of course still not an especially practical environment to perform any real computation.

We include sample program 3, which searches for a counterexample to the Collatz conjecture, as a concrete demonstration of the possibility brought up in discussions of [1]. If Alice should set up the microcontroller and start this program, the game is a victory for Bob as soon as the program finds a cycle of numbers that is a counterexample to the Collatz conjecture. If (as is widely suspected) no such counterexample exists [10], or if it instead finds a sequence that goes on forever without repeating, the game is a draw by infinite loop. This provides an explicit *Magic* game state where all player choices have been removed but the end result of the game is unknown to current mathematics.

6.4 Further research

It is clear that *Magic* is as computationally complex as it's possible for a perfect knowledge game to be. But not all two-player games are perfect knowledge, and *Magic* contains many cards and mechanics that use hidden information. Our construction doesn't use any of these, but it's possible future constructions could. This would allow embedding a wider variety of games into *Magic*, such as two-player games where both players choose their moves simultaneously.

It is also possible that there exist other tabletop games which support embedding this kind of construction. But any such game would need to be in the small subset of tabletop games which allow all of the following:

- An unlimited number of player actions rather than a fixed number of turns
- An unlimited number of at least one resource
- Some way to constrain the actions players can perform
- Enough flexibility in player choices to allow forcing one action to result in another

It may well be the case that *Magic* is the only widely played tabletop game meeting these criteria which has enough depth of rules and scope for player creativity to allow this kind of construction. If that is the case, we are very grateful to Wizards of the Coast for providing such a versatile set of building blocks for us to play with.

References

1. Alex Churchill, Stella Biderman, and Austin Herrick. Magic: The Gathering is Turing complete. In *10th International Conference on Fun with Algorithms (FUN 2020)*, 2020.
2. Jan Biel. How to run any program in a Magic: The Gathering Turing machine, 2019. `https://www.youtube.com/watch?v=YzXoFldEux4`.
3. Z.A. Melzak. An informal arithmetical approach to computability and computation. *Canadian Mathematical Bulletin*, 4(3):279–293, 1961.
4. Choong Yin Howe and Alex Churchill. A more easily programmable system constructible in Magic: The Gathering, Apr 2024. `https://cyh31.neocities.org/amepscimtg/explanation`.
5. Wizards of the Coast. Magic: The Gathering comprehensive rules, Jan 2024. `https://magic.wizards.com/en/rules`.
6. Alex Churchill and Choong Yin Howe. Magic microcontroller website, Apr 2024. `https://www.toothycat.net/~hologram/Magic/`.
7. Alex Churchill and Choong Yin Howe. Magic microcontroller simulator, Apr 2024. `https://www.toothycat.net/~hologram/Magic/MTGProgSimulatorText.html`.
8. William "Huey" Jensen. Retro deck highlight: Huey's 2002 battle of wits, Jun 2023. `https://strategy.channelfireball.com/home/retro-deck-highlight-hueys-2002-battle-of-wits/`.
9. Wizards of the Coast. Magic: The Gathering tournament rules, Jan 2024. `https://wpn.wizards.com/en/rules-documents`.
10. Jeffrey C. Lagarias. The 3x+1 problem: An overview. *The Ultimate Challenge: The 3x+1 Problem*, pages 3–29, 2010.

Appendix A Sample Programs

Readers are encouraged to explore the functionality of these sample programs firsthand by executing them within the provided simulator interface [7] where they are preset options.

Sample Program 1 10 Cubed

Symbols	Instruction	Comments
10 0 10	NumBuild 10	Initialise r_0 to 10
5 1 0	Set r_1=r_0	r_1 is the output
6 0 1	Multiply r_1 r_0	Now r_1 is 100
6 0 1	Multiply r_1 r_0	Now r_1 is 1000
4 0 0	HaltD	We're done.

The complete program is: 10 0 10 5 1 0 6 0 1 6 0 1 4 0 0 – or in cards: Snow-Covered Forest, Plains, Snow-Covered Forest, Wastes, Island, Plains, Snow-Covered Plains, Plains, Island, Snow-Covered Plains, Plains, Island, Forest, Plains, Plains.

After 5 of Alice's turns and 4 of Bob's, the game will end in a draw. Register r_1 will have 1000 +1/+1 counters on it, the result of the calculation.

Sample Program 2 Prime Factors

Symbols	Instruction	Comments
Register usage: r_0: constant 2. r_1: input number remaining to be factorised. r_2: copy of r_1 for divisions. r_3: current divisor. r_4: how many factors found so far.		
2 1 0	AInput 1	Read the input number into r_1
4 1 3	Add1 r_3	Initialise divisor to 1
10 0 2	NumBuild 2	Initialise r_0 to constant 2
Main loop: test the next number		
4 1 3	Add1 r_3	Increment the divisor we're testing
5 2 1	Set r_2 r_1	Prepare to test r_1
7 2 3	DivCeil r_2 r_3	Divide and check remainder
3 5 4	JumpBwdF 4	If flag, r_3 is not a factor
Found a factor: store it and the quotient		
8 3 4	Store r_3 r_4	It is. Save r_3 to a new memory slot,
4 1 4	Add1 r_4	and increment number of factors found
5 1 2	Set r_1 r_2	Remember the new divided total
11 0 2	FLess r_2 r_0	Is r_2 now 1?
3 3 8	JumpBwdNF 8	If not, continue. Could be another factor of r_3 so recheck it.
4 0 0	HaltD	If so, halt

After execution finishes, there will be one memory entry for each prime factor in the input number. For example, if Alice chooses 120, the program finishes after 57 turn cycles, and memory consists of {2, 2, 2, 3, 5}.

Sample Program 3 Collatz $(3n + 1)$

Symbols	Instruction	Comments
Register usage: r_0: built numbers. r_1, r_2, r_3: constants 1, 2, 3. r_4: source of the current chain. r_6: temp read memory. r_7: current number being checked. r_8: either half r_7 or $3r_7 + 1$.		
4 1 1	Add1 r_1	Initialise r_1 to 1
10 0 2	NumBuild 2	Create constant 2
5 2 0	Set r_2 r_0	Store constant 2 in r_2
10 0 3	NumBuild 3	Create constant 3
5 3 0	Set r_3 r_0	Store constant 3 in r_3
10 4 2	NumBuild $12 \times 4 + 2$	Start searching at 50
5 4 0	Set r_4 r_0	Initialise current search root
5 7 4	Set r_7 r_4	Start checking at current search root
Label 0: we have a new r_7 to investigate		
11 2 7	FLess r_7 r_2	Is $r_7 = 1$?
10 1 4	NumBuild $12 \times 1 + 4$	Create longjump distance 16
3 4 0	JumpFwdF r_0	If so, jump to label 3
9 6 7	Load r_6 r_7	Load memory r_7 into r_6
11 6 6	FIsZero r_6	Is this a new number?
3 4 3	JumpFwdF 3	If so, go to label 1
11 2 6	FLess r_6 r_2	Is this a number that we know gets to 1?
3 4 11	JumpFwdNF 1	If so, jump to label 3
4 0 2	HaltB	If not, we found a loop & disproved the Collatz conjecture!
Label 1: r_7 is a number we've not seen before		
5 8 7	Set r_8 r_7	Prepare to halve r_8
4 2 8	Halve r_8	Halve r_8. Did that leave remainder?
3 2 3	JumpFwdNF 3	If not, r_8 is what we want at Label 2
5 8 7	Set r_8 r_7	Set r_8 to r_7...
6 3 8	Mult r_8 r_3	...×3...
4 1 8	Add1 r_8	...+1.
Label 2: r_8 is the next number in the sequence		
8 8 7	Store r_7 r_8	Store r_7 in memory r_8
5 7 8	Set r_7 r_8	Now investigate r_8
10 1 7	NumBuild $12 \times 1 + 7$	Create longjump distance 19
3 1 0	JumpBwd r_0	Go back to label 0
Label 3: A number r_4 got down to 1. Label the chain with 1s.		
5 7 4	Set r_7 r_4	Restart at r_4
9 6 7	Load r_6 r_7	Load memory r_7 into r_6
11 2 6	FLess r_6 r_2	Is $r_6 = 1$?
3 4 3	JumpFwdF 3	If so, skip to end of the loop
8 1 7	Store r_7 r_1	Save 1 into r_7
5 7 6	Set r_7 r_6	Set r_7 to the number we read
3 1 6	JumpBwd 6	Go back 6
4 1 4	Add1 r_4	We're done with r_4's chain. Next number!
3 0 7	JumpFwd 7	Loop around to just before label 0

Space constraints prevent us from including Sample Program 4 "Nim" here. See [7] to see and run this sample program and all the others.

Appendix B Card Modification Techniques

Here we detail the techniques used while setting up the microcontroller to accomplish the various modifications to cards described in Section 4 and in the instructions (whose implementation is omitted from this paper for space reasons, but can be seen at [6]). We assume Alice has generated an arbitrarily large amount of mana and drawn all the cards she needs using **Dimir Guildmage**. We are able to repeatedly cast the instants and sorceries used below by repeatedly casting **Archaeomancer** and bouncing it with **Capsize**.

- Editing creature types: **Artificial Evolution**
- Editing colour words: **Mind Bend**
- Copiably setting creature type and/or colour and making creatures 1/1: **Croaking Counterpart** combined with **Artificial Evolution** and **Spectral Shift**
- Adding copiable creature types: **Glasspool Mimic** in conjunction with **Artificial Evolution**
- Non-copiably setting creature type: Use **Blade of Shared Souls** to temporarily make the creature a copy of **Proteus Machine**. Use **Backslide** to turn it face down, then turn it face up and set its creature type.
- Adding non-copiable creature types: **Olivia Voldaren**, modified by **Artificial Evolution** to change Vampire to another type.
- Copiably setting power and toughness, for positive toughness: **Saw in Half** after adjusting as necessary with **Belbe's Armor**, **Enrage**, and/or **Drana, Kalastria Bloodchief**.
- Copiably setting power and toughness to 0/0 and adding type artifact: Have an **Engineered Plague** on Shapeshifter creatures. Cast **Hulking Metamorph** prototyped, and decline to copy anything; it's now 2/2. Cast **Saw in Half** on it, producing token copies that are base 1/1, net 0/0, and then they can copy other creatures while setting base P/T to 0/0. **Grumgully, the Generous** adds a +1/+1 counter to keep it alive.
- Non-copiably setting power and toughness to the same number: **Gigantoplasm**
- Copiably setting mana value to 0: **Vizier of Many Faces** copiably removes the mana cost, making the mana value 0. (This is usually done so that **Bludgeon Brawl** does not make this give a power boost.) We can repeat this if necessary using **Lithoform Engine** to copy the Embalm ability, untapped by **Twiddle**.
- Copiably setting card type to (only) artifact: **Imposter Mech**, after targeting the original with **Donate**.
- Adding type artifact: **Memnarch**
- Copiably adding type creature: **Urza, Prince of Kroog**, in conjunction with **Memnarch** if necessary, and with the creature type edited by **Artificial Evolution**.
- Copiably adding flying: **Irenicus's Vile Duplication** copiably adds flying.
- Adding keyword ability counters – flying, indestructible, reach, first strike, double strike, lifelink, deathtouch, vigilance, trample: **Kathril, Aspect Warper**, having used **Dimir Guildmage** to discard other cards used in the construction: **Healer's Flock**; **Darksteel Myr**; **Halana and Alena, Partners**; **Sylvia Brightspear**; **Questing Beast**; and **Quartzwood Crasher**. **Regrowth** gets back the discarded cards afterwards.
- Adding protection from a colour: Have a **Council Guardian** enter the battlefield, and use **Ballot Broker** to make sure the right colour wins the vote. Then use **True Polymorph** to turn it into what it should be.

- Adding islandwalk: **Fishliver Oil**, copied with **Mythos of Illuna**
- Adding "Whenever this creature deals combat damage to a player, draw a card.": Use **Blade of Shared Souls** to temporarily make the creature a copy of an **Ascendant Spirit**, which itself is copiably an Angel by **Glasspool Mimic**+**Artificial Evolution**, and then activate its last ability to add this ability. Generate the snow mana with a Snow-Covered land repeatedly untapped by **Twiddle**.
- Removing flying: Use **Blade of Shared Souls** to temporarily make the creature a copy of **Mist Dragon**, and use its ability to remove flying.
- Adding +1/+1 counters: **Kathril, Aspect Warper** and **Resourceful Defense**
- Adding -1/-0 counters: **Jabari's Influence** to get the first one, and multiple copies of **Resourceful Defense** to get more.
- Adding +1/+0 counters: **Dwarven Armorer** produces +1/+0 counters. Multiple copies of **Resourceful Defense** multiply the counters and then move them, as necessary.
- Creating arbitrary tokens: **Rotlung Reanimator** with **Artificial Evolution** creates tokens of arbitrary types, and **Saw in Half** sets their sizes after adjusting with **Belbe's Armor**.
- Setting colours: **Prismatic Lace**
- Giving Bob control of cards: **Donate**

Example: We make each **Vaevictis Asmadi, the Dire** into a 1/1 Sliver Beast Reflection by casting **Croaking Counterpart** targeting a real **Vaevictis Asmadi, the Dire**, then responding by casting **Artificial Evolution** targeting **Croaking Counterpart** to change Frog to Sliver. Once the 1/1 Sliver is present, we return **Artificial Evolution** to our hand using **Archaeomancer**. We cast **Glasspool Mimic**; edit it with **Artificial Evolution** replacing Shapeshifter with Beast; cast **Capsize** with buyback on the **Archaeomancer**; recast **Archaeomancer** using **Leyline of Anticipation** to get back **Artificial Evolution** again; then use **Artificial Evolution** one more time to replace Rogue with Reflection. Then **Glasspool Mimic** resolves and becomes a 1/1 Sliver Beast Reflection copy of **Vaevictis Asmadi, the Dire**. We can create further token copies of this with **Mythos of Illuna** (all the creature type additions are copiable), and then use **Capsize** to return the original **Glasspool Mimic** to our hand for the next time we need to use it.

Appendix C Decklist

Table 4 on the following page contains a decklist suitable for bringing to a Legacy tournament which could set up the microcontroller. Adjust the number of basic lands according to the program you wish to write; the following decklist contains enough to write the Collatz sample program. You can also leave out cards that are only used in instructions that aren't present in the program you wish to write. E.g. **Oubliette** is only used in Call and Return instructions.

Table 4 Decklist to play the *Magic* Microcontroller in a Legacy tournament

Card	Purpose	Card	Purpose
4 Ancient Tomb	Bootstrap	4 Lotus Petal	Bootstrap
4 Grim Monolith	Infinite mana	4 Power Artifact	Infinite mana
4 Gemstone Array	Infinite mana	4 Dimir Guildmage	Draw rest of deck
13 Plains	Program	13 Snow-Covered Plains	Program
13 Island	Program	18 Snow-Covered Island	Program
13 Swamp	Program	13 Snow-Covered Swamp	Program
18 Mountain	Program	8 Snow-Covered Mountain	Program
18 Forest	Program	10 Snow-Covered Forest	Program
13 Wastes	Program	10 Snow-Covered Wastes	Program
1 Memnarch	Make token copies	1 Mythos of Illuna	Make token copies
1 Capsize	Set up	1 Archaeomancer	Set up
1 Artificial Evolution	Edit cards	1 Mind Bend	Edit cards
1 Prismatic Lace	Edit cards	1 Spectral Shift	Edit cards
1 Glasspool Mimic	Add types	1 Olivia Voldaren	Add types
1 Proteus Machine	Set creature types	1 Backslide	Set creature types
1 Argent Mutation	Set up	1 Leyline of Anticipation	Set up
1 Gigantoplasm	Edit power/toughness	1 Croaking Counterpart	Set up
1 Saw in Half	Edit power/toughness	1 Belbe's Armor	Edit power/toughness
1 Enrage	Edit power/toughness	1 Drana, Kalastria Bloodchief	Edit power/toughness
1 Engineered Plague	Edit power/toughness	1 Grumgully, the Generous	Edit power/toughness
1 Hulking Metamorph	Edit power/toughness	1 Vizier of Many Faces	Edit mana value
1 Lithoform Engine	Edit mana value	1 Twiddle	Edit mana value
1 Imposter Mech	Edit types	1 Donate	Edit control
1 Astral Dragon	Add types and abilities	1 Irenicus's Vile Duplication	Add abilities
1 Urza, Prince of Kroog	Add types	1 Kathril, Aspect Warper	Add abilities
1 Council Guardian	Add abilities	1 Ballot Broker	Add abilities
1 True Polymorph	Add abilities	1 Fishliver Oil	Add abilities
1 Blade of Shared Souls	Change types and abilities	1 Ascendant Spirit	Add abilities
1 Mist Dragon	Remove abilities	1 Resourceful Defense	Add counters
1 Jabari's Influence	Add counters	1 Fishliver Oil	Add abilities
1 Dwarven Armorer	Add counters	1 Sealock Monster	Add types
1 True Polymorph	Add abilities	1 Grand Melee	Control combat
1 Stormtide Leviathan	Control combat	1 Tetsuko Umezawa, Fugitive	Control combat
1 Worship	Control combat	1 Vaevictis Asmadi, the Dire	Advance program
1 Wheel of Sun and Moon	Advance program	1 Tajuru Preserver	Advance program
1 Dralnu's Crusade	Conditional mechanism	1 Life and Limb	Conditional mechanism
1 Blood Sun	Conditional mechanism	1 Strata Scythe	Conditional mechanism
1 Healer's Flock	Keyword abilities	1 Dream Fighter	Conditional mechanism
1 Shadow Sliver	Conditional mechanism	1 Bludgeon Brawl	Conditional mechanism
1 Bloodthirster	Advance program	1 Masked Gorgon	Constrain targets
1 Spectral Guardian	Constrain targets	1 Sterling Grove	Constrain targets
1 Ivory Mask	Constrain targets	1 Illusionary Terrain	Control types
1 Joraga Warcaller	Registers	1 Riders of Gavony	Registers
1 Nevermore	Control choices	1 Aggressive Mining	Control choices
1 Root Maze	Control choices	1 Tomorrow, Azami's Familiar	Flag
1 Choke	Control choices	1 Suppression Bonds	Control choices
1 Stony Silence	Control choices	1 Cursed Totem	Control choices
1 Recycle	Control state	1 Mirror Gallery	Control state
1 Corrosive Mentor	Add abilities	1 Halana and Alena, Partners	Instructions
1 Archpriest of Iona	Instructions	1 Shape Stealer	Instructions
1 Necrogen Mists	Instructions	1 Halfdane	Instructions
1 Tanuki Transplanter	Instructions	1 Furtive Homunculus	Instructions
1 Omnath, Locus of Mana	Instructions	1 Wrathful Red Dragon	Instructions
1 Belligerent Brontodon	Instructions	1 Smog Elemental	Instructions
1 Hornet Nest	Instructions	1 Phantom Steed	Instructions
1 War Elemental	Instructions	1 Tocatli Honor Guard	Instructions
1 Ward Sliver	Instructions	1 Godhead of Awe	Instructions
1 Wandering Wolf	Instructions	1 Behind the Scenes	Instructions
1 Spinneret Sliver	Instructions	1 Quartzwood Crasher	Instructions
1 Arwen, Weaver of Hope	Instructions	1 Aether Flash	Instructions
1 Charisma	Instructions	1 Skeleton Key	Instructions
1 Serpent of Yawning Depths	Instructions	1 Field Marshal	Instructions
1 Questing Beast	Instructions	1 Vigor	Instructions
1 Toralf, God of Fury	Instructions	1 Gideon's Intervention	Instructions
1 Progenitor Mimic	Instructions	1 Volcano Hellion	Instructions
1 Artificer Class	Instructions	1 Syr Elenora, the Discerning	Instructions
1 Slithering Shade	Instructions	1 Suntail Hawk	Instructions
1 Strength-Testing Hammer	Instructions	1 Ancient Gold Dragon	Instructions
1 Rat Colony	Instructions	1 Goblin Pyromancer	Instructions
1 Celestial Convergence	Instructions	1 Alert Heedbonder	Instructions
1 Spiritual Sanctuary	Instructions	1 Abyssal Specter	Instructions
1 Catacomb Dragon	Instructions	1 Taii Wakeen, Perfect Shot	Instructions
1 Excruciator	Instructions	1 Aegar, the Freezing Flame	Instructions
1 Khorvath Brightflame	Instructions	1 Sylvia Brightspear	Instructions
1 Mangara's Equity	Instructions	1 Darksteel Myr	Instructions
1 Ojutai, Soul of Winter	Instructions	1 Angrath's Marauders	Instructions
1 Kangee, Aerie Keeper	Instructions	1 Shimmer	Instructions
1 Fiery Emancipation	Instructions	1 Chains of Mephistopheles	Instructions
1 Captain's Claws	Instructions	1 Steely Resolve	Instructions
1 Reaper King	Instructions	1 Akron Legionnaire	Instructions
1 Discordant Spirit	Instructions	1 Blinding Angel	Instructions
1 Meishin, the Mind Cage	Instructions	1 Bower Passage	Instructions
1 Darksteel Myr	Instructions	1 Moonsilver Spear	Instructions
1 Spellbane Centaur	Instructions	1 Melira's Keepers	Instructions
1 Spiteful Shadows	Instructions	1 Empyrial Archangel	Instructions
1 Lich	Instructions	1 Justice	Instructions
1 Rotlung Reanimator	Instructions	1 Oubliette	Instructions
1 Willbreaker	Instructions	1 Razorjaw Oni	Instructions
1 Sosuke, Son of Seshiro	Instructions	1 Syphon Sliver	Instructions
1 Dormant Sliver	Instructions	1 Skanos Dragonheart	Instructions
1 Corpsejack Menace	Instructions	1 Okk	Instructions
1 Bishop of Binding	Instructions	1 Tamiyo, Collector of Tales	Instructions
1 Shimmer	Instructions	1 Sporemound	Instructions
1 Polyraptor	Instructions	1 Chief of the Scale	Instructions
1 Gruul Ragebeast	Instructions	1 Sliver Hivelord	Instructions