A Simple RISK Architecture

Stefan Muller
Illinois Institute of Technology

ABSTRACT

This paper introduces RISK-V, a minimal computer architecture that targets a widely-available hardware platform not previously used for computing.

1 INTRODUCTION

Reduced Instruction Set Computer, or RISC, architectures [1] have been studied since the early 1980s, with the goal of providing a simple, easily implementable, instruction set architecture. Less well-studied (that is, not studied until now) are RISK architectures, which are even simpler to implement because they target readily-available and portable[1] hardware, namely the board game *Risk*.

2 COMPUTING PLATFORM

Our architecture is built on top of a simplified, two-player version of Risk, which we will now describe. Because of its substantial departures from the usual rules, we refer to our architecture as RISK-V (the V standing for "Very Simple"). The game is played on a board resembling a world map that might be found in a history textbook or an ancient globe in an underfunded school. The map is divided into 24 territories, whose ownership is divided between the two players at the start of the game. Because randomness in computing hardware makes writing programs difficult, we perform this assignment deterministically. Each player maintains some (strictly positive) number of *armies* on each territory they hold. Initially, each territory contains one army.

The two players take turns attempting to conquer the world. A turn consists of three phases, as follows:

(1) *Reinforcement.* The player may place any number of armies in any territory they control. This may be repeated any number of times.
(2) *Attack.* The player may make zero or more attacks. In an attack, the player selects a territory they control to attack from, and an adjacent territory controlled by the other player to attack. The player may attack with any number A of armies between 1 and $N-1$, where N is the number of armies in the attacking territory. Let D be the number of armies in the defending territory. If $A \leq D$, the attack is unsuccessful. The number of armies in the defending and attacking territories are decreased by A (except that if $A = D$, one army is left in the defending territory). If $A > D$, the attack is successful. The attacking player gains control of the defending territory; $N - A$ armies are left in the attacking territory and $A - D$ armies move to the defending territory.
(3) *Tactical Move.* The player may make one tactical move, moving some number of armies between two adjacent territories both controlled by the player. At least one army must be left in each territory.

Performing an invalid move raises a fault.

[1]Unless you have some sort of special edition version that comes in a wooden box or something.

SIGBOVIK 2023,
2023.

3 RISK-V ASSEMBLY

RISK-V assembly language contains 5 instructions. Arguments in angle brackets are required, arguments in parentheses are optional.

- `REINFORCE` ⟨*territory*⟩ ⟨*n*⟩
  Reinforce ⟨*territory*⟩ with *n* armies.
- `ATTACK` ⟨*territory*$_1$⟩ ⟨*territory*$_2$⟩ (*n*)
  Attack ⟨*territory*$_2$⟩ from ⟨*territory*$_1$⟩ with *n* armies (or the maximum number if unspecified). The current player's turn moves to the attack phase if it is not already there.
- `TACTICAL` ⟨*territory*$_1$⟩ ⟨*territory*$_2$⟩ (*n*)
  Tactical move from ⟨*territory*$_1$⟩ to ⟨*territory*$_2$⟩ with *n* armies (or the maximum number if unspecified). The current player's turn moves to the tactical phase.
- `END TURN`
  Ends the current player's turn. May be executed in any phase of the turn. Must be executed to end the turn even if a tactical move has been executed.
- `IF UNSUCCESSFUL GOTO ⟨ label ⟩`
  Go to the instruction labeled ⟨ `label` ⟩ if the last attack or tactical move was unsuccessful. A tactical move is unsuccessful if not enough armies were available in the "from" territory.

4 CONVENTIONS AND PATTERNS

We will use the territories as registers holding nonnegative integers. Because territories cannot be empty, we will consider a territory with N armies to hold the number $N-1$. By convention, the return value of the program is placed in Kamchatka. We refer to the players as A and B.

Addition. We can set $a = a+b$ using the instruction `TACTICAL` b a. This, of course, requires that the territories holding a and b are adjacent and both controlled by the current player, and that it is the tactical move phase.

Subtraction. We can set $a = \max(0, a-b)$ using the instruction `ATTACK` b a, assuming a and b are adjacent, b is controlled by the current player, a is controlled by the other player, and it is the attack phase.

Negating Conditions. The only built-in jump instruction is `IF UNSUCCESSFUL GOTO`. We can construct code that branches to a label on a successful move with an `IF UNSUCCESSFUL GOTO` that jumps over 1) a move that always fails and 2) another `IF UNSUCCESSFUL GOTO`.

```
IF UNSUCCESSFUL continue
```

```
  # Always fails
  ATTACK WAUSTRALIA EAUSTRALIA
  IF UNSUCCESSFUL GOTO l
continue:
```

Duplicate. The addition and subtraction operations described above result in the value in a being lost, so it is useful to duplicate the value in a territory before performing one of these operations. As an example, the following code copies the value in Kamchatka to Mongolia and Yakutsk (in the process erasing the value in Kamchatka; if desired, the value can be moved back to the original register). The source and both destination territories must be controlled by the current player, and at least one destination must be adjacent to the source. The code moves one army at a time from Kamchatka to Mongolia as long as this is possible, and also reinforces Yakutsk with one army after each move.

```
  # Copy Kamchatka to Mongolia, Yakutsk
  # All three are held by Player B
copy:
  TACTICAL KAMCHATKA MONGOLIA 1
  # Now A's turn; A ends turn
  END TURN
  END TURN
  IF UNSUCCESSFUL GOTO end_copy
  REINFORCE YAKUTSK 1
  # Always fails
  ATTACK WAUSTRALIA EAUSTRALIA
  IF UNSUCCESSFUL GOTO copy
end_copy:
```

5 CASE STUDY: FIBONACCI

The code in Figure 1 calculates the N^{th} Fibonacci number, where N is the value in Ontario at the start of the program.

6 DISCUSSION AND FUTURE WORK

We have implemented a simulator for RISK-V[2], on which we have tested the code provided in this paper. Future work would be to build compiler backends targeting the RISK-V architecture.

REFERENCES

[1] David A. Patterson and Carlo H. Sequin. 1981. RISC I: A Reduced Instruction Set VLSI Computer. In *Proceedings of the 8th Annual Symposium on Computer Architecture (ISCA '81)*. IEEE Computer Society Press, Washington, DC, USA, 443–457.

[2]A simulator hardly seems necessary, as the architecture targets a readily-available platform, but is provided for legacy users who require compatibility with outdated, silicon-based computing hardware

```
  # Invariants:
  # n = N - value in Ontario
  # a = fib(n-1) (stored in Irkutsk)
  # b = fib(n) (stored in Kamchatka)
  # Double N
  # Copy N to WesternUS, Greenland
copy0:
  TACTICAL ONTARIO WESTERNUS 1
  # Now B's turn; B ends turn
  END TURN
  END TURN
  IF UNSUCCESSFUL GOTO end_copy0
  REINFORCE GREENLAND 1
  # Always fails
  ATTACK EAUSTRALIA WAUSTRALIA
  IF UNSUCCESSFUL GOTO copy0
end_copy0:
  TACTICAL GREENLAND ONTARIO
  END TURN
  END TURN
  TACTICAL WESTERNUS ONTARIO
  END TURN
  # Initialize b to 1
  REINFORCE KAMCHATKA 1
  # Set Northwest Territory to 4
  REINFORCE NORTHWESTTERRITORY 3
  # Start while
loop:
  REINFORCE NORTHWESTTERRITORY 3
  # Copy b to Mongolia, Yakutsk
copy:
  TACTICAL KAMCHATKA MONGOLIA 1
  # Now A's turn; A ends turn
  END TURN
  END TURN
  IF UNSUCCESSFUL GOTO end_copy
  REINFORCE YAKUTSK 1
  # Always fails
  ATTACK WAUSTRALIA EAUSTRALIA
  IF UNSUCCESSFUL GOTO copy
end_copy:
  # b = a + b
  TACTICAL IRKUTSK KAMCHATKA
  END TURN
  END TURN
  TACTICAL YAKUTSK KAMCHATKA
  END TURN
  END TURN
  # a = Mongolia
  TACTICAL MONGOLIA IRKUTSK
  END TURN
  END TURN
  # IF N-- < 2 THEN END
  ATTACK NORTHWESTTERRITORY ONTARIO 2
  END TURN
  END TURN
  IF UNSUCCESSFUL GOTO loop
```

Figure 1: RISK-V Assembly for the Fibonacci function.