Northwind Automatic Card Dealer ACD-2

Service Manual

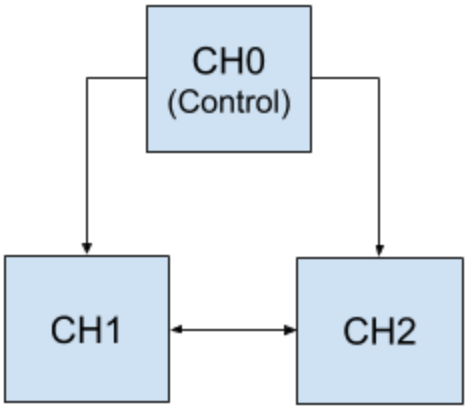
Overview

The Northwind Automatic Card Dealer (ACD) is a state of the art card distribution device suitable to a number of different games. Its dealing behavior is customizable to the end user to fit any possible needs.

Architecture

The ACD consists of 2 or more connected Card Handling (CH) units. The CH units are each identical and customizable. Their behavior is described later in this document.

Several models of ACD are available. By default they will randomly assign cards to each of their output units. The **ACD-2** consists of 3 CH units and is perfect for 2-player games.



Behavior

By default, all ACD models will assign an equal number of randomly selected cards to all outputs.

Customizing Behavior

Each CH unit can be configured with microcode instructions. Each CH unit has a series of registers and can handle **a single card at a time**:

Register	Description
CVAL	Returns the value of the current card held by this unit from 1 (Ace) to 13 (King). Will return 0 if no card currently in this unit. Read-only.
CVALB	Returns the bounded value of the current card held by this unit from 1 (Ace) to 10 (10, Jack, Queen, King). Will return 0 if no card currently in this unit. Read-only.
R0 R1 R2 R3 R4	8-bit signed general purpose register
UP LEFT RIGHT DOWN	Output ports allowing a card to be sent to an adjacent unit. Not all ports are connected on all units.

Input and Output

There is an input queue of cards attached to each unit. Units can asynchronously write to these queues, but all operations which read from the queue are blocking if the queue is currently empty. The input queue for any given CH node is shared between all other nodes capable of sending cards to that node. Any attempt to write more cards to an input queue will block the writer until the queue has been emptied.

Nodes can get rid of cards in two ways: First, nodes associated with an output position (often a player) can add the current card to the dealing queue with the KEEP instruction. Second, they may send cards to adjacent nodes with the SEND instruction.

The adjacency of nodes is shown in their layout diagrams. For instance, in the ACD-4, node P1 is adjacent to nodes P2 and P4.

The **Control** node is treated differently - it traditionally has the full deck as its input queue, but it has no way to deal a card, and other nodes cannot SEND cards to it.

Timing

All CH units are running in parallel. A shared clock advances all units one instruction simultaneously (excepting units currently blocked waiting for input

or sending to a full queue). The concept of a blocked node is determined at the start of each instruction. If a value is written to queue, the reader will not unblock until the next clock cycle. If a value is removed from a full queue, the writer will not unblock until the next cycle.

In the case of simultaneous writes or unblocking, the node of highest ID will be allowed to act first. The control node is considered to have ID 0.

Microcode Instructions

The following instructions are available for use in CH:

Instruction	**Syntax**	**Description**
READ	READ	Reads a new card from the deck. Crashes if the CARD register currently has contents.
RRAND (Read random)	RRAND	Reads a random card from the input queue. Crashes if the CARD register currently has contents.
KEEP	KEEP	Retains the card currently in the CARD register in the list of cards eligible to be dealt from this unit. Crashes if no card exists. The next time the player associated with this CH requests a card, that card will be provided to them. This does **NOT** mean that the card will be immediately dealt in a game.
SEND	SEND LEFT	Sends the card currently in the CARD register in the given direction. Crashes if no adjacent node is connected in that direction or if no card is in the CARD register
ADD	ADD R0 SUIT R1	R1 = (R0 + SUIT)
	ADD R0 2 R0	R0 = R0 + 2
SUB (Subtract)	SUB R0 SUIT R1	R1 = (R0 - SUIT)
	SUB R0 2 R0	R0 = R0 - 2

NEG (Negate)	NEG R0	R0 = 0 - R0
: (Label)	FOO:	Creates a label for use with jump instructions. Labels must begin with a capital letter and consist of letters and numbers only. Any pattern of R followed by numbers, any instruction name or any register name is reserved.
JMP (Jump)	JMP FOO	Jump to the instruction at the label FOO
JE (Jump if Equal)	JE R0 R1 FOO	Jump to FOO if R0 = R1
	JE R0 6 FOO	Jump to FOO if R0 = 6
JNE (Jump if not equal)	JNZ R0 R1 FOO	Jump to FOO if R0 ≠ R1
	JNZ R0 7 FOO	Jump to FOO if R0 ≠ 7
JGT (Jump if greater than)	JGT R0 R1 FOO	Jump to FOO if R0 > R1
	JGT R0 5 FOO	Jump to FOO if R0 > 5
JGE (Jump if greater than or equal)	JGE R0 R1 FOO	Jump to FOO if R0 ≥ R1
	JGE R0 4 FOO	Jump to FOO if R0 ≥ 4
JLT (Jump if less than)	JLT R0 R1 FOO	Jump to FOO if R0 < R1
	JLT R0 3 FOO	Jump to FOO if R0 < 3
JLE (Jump if less than or equal)	JLE R0 R1 FOO	Jump to FOO if R0 ≤ R1
	JLE R0 2 FOO	Jump to FOO if R0 ≤ 2
NOP (No operation)	NOP	No operation (do nothing)
# (comment)	# This is a comment	Ignores all text on this line

Testing

The ACD-2 unit allows you to step through its program instruction by instruction to inspect state and debug. If a program successfully runs (or is stepped) to completion, you will have the option to run a Validation pass against it.

Validation does a sanity check on the program with a series of inputs, ensuring that:

- The player total is not always the same
- The dealer total is not always the same
- The dealer's visible card is not always the same

Tips and Tricks

If the CH runs out of instructions it will crash. Remember to loop!

One common technique for dealing cards is the **circular buffer**. If you don't want to deal a card, pass it to another CHU. If you see it again, keep passing!

The stock coding for the ACD-2 unit is as follows:

Control:

```
START:
READ
SEND LEFT
READ
SEND RIGHT
JMP START
```

CH1/CH2:

```
START:
READ
KEEP
JMP START
```

Sample Programs

This program will send all odd cards to CH2 and all even cards to CH1

Control:

```
LOOP:
READ
SEND LEFT
READ
SEND RIGHT
JMP LOOP
```

CH1:

```
LOOP:
```

```
READ
# Store the card value
ADD CVAL 0 R0

# Determine if the card is even or odd
MOD:
JLT R0 2 GOTMOD
SUB R0 2 R0
JMP MOD

GOTMOD:
# Keep the card if it's even
JE R0 0 PRESERVE
SEND RIGHT
JMP LOOP

PRESERVE:
KEEP
JMP LOOP
```

CH2:

```
LOOP:
READ
# Store the card value
ADD CVAL 0 R0

# Determine if the card is even or odd
MOD:
JLT R0 2 GOTMOD
SUB R0 2 R0
JMP MOD

GOTMOD:
# Keep the card if it's odd
JE R0 1 PRESERVE
SEND LEFT
JMP LOOP

PRESERVE:
KEEP
JMP LOOP
```