FunCPU

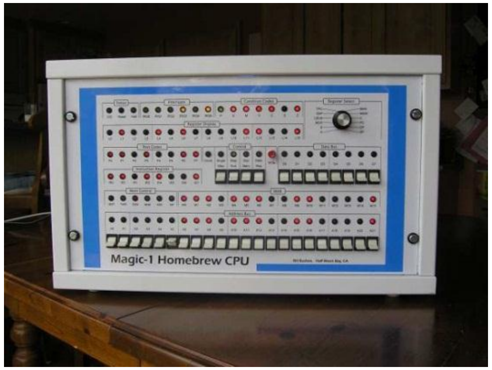
*7 Bit Homebrew CPU
Designed For
Functional Programming

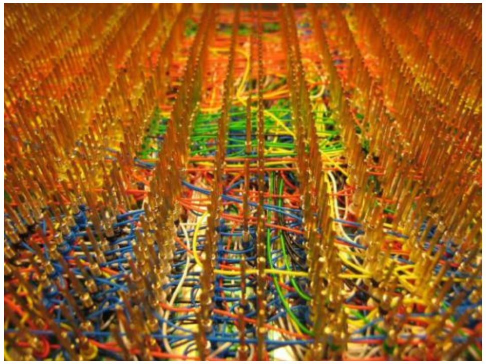
András Juhász „Árokparty”, 19.07.2014.

Motivation

- Software guy: to create something „real”
- IT engineer: to have a computer, which I fully understand
- Bored hobbyst: to find a challenge, a logical puzzle
- Myself: All of the above, plus: to create something „new“, unconventional, „exotic“.

Homebrew Scene





Magic 1 – Bill Buzbee

Harry Porter – Relay CPU

Big Mess Of Wire -

Steve Chamberlin

Time Fracture - John Doran

Mark-1 FORTH- Andrew Holme

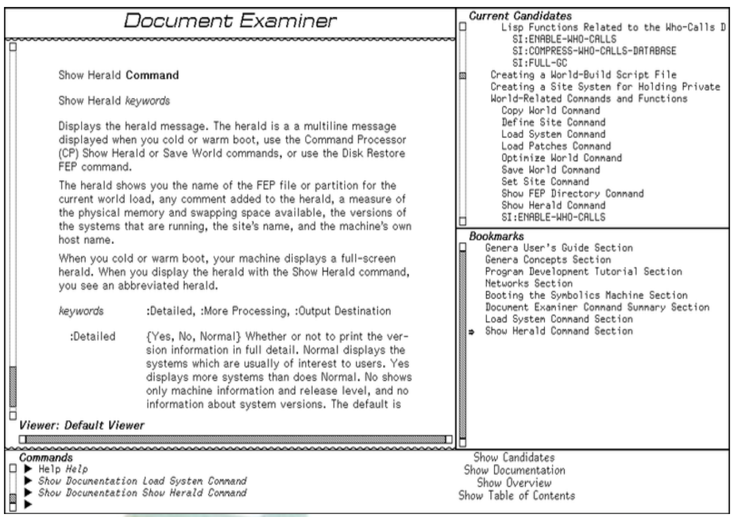
Goals

- Simple;
- Unconventional;
- Buildable/achievable;
- „Useful”;
- Native functional programming support.

- www.mycpu.blog.hu

„Lisp-machine“

- Graphical workstaton
- Genera lisp op.system
- Symbolics, TI,Xerox



Dedicated CPUs

- VLSI-PLM, then PLUM for Prolog
- Nikolaus Wirth - CPU for Module 2
- INMOS Transputer for Occam
- AT&T Hobbit – to support CRISP C
- Java processors
- Ericsson ECOMP for Erlang
- Source: Wikipedia

Overview

- Concept
- Challenge
- Examples
- Architecture
- Implementation
- Computability
- Improvements

The Concept

FunCPU

- Natively supports functional programming;
- Special assembly instruction set;
- Simple programming paradigm;
- Typed (tagged) architecture;
- Main focus on numerical computations;
- Turing-complete.

Unconventional

- No PC.
- No SP, no stack.
- There are no flags.
- No jumping/branching instructions.
- Not even the accumulator exists.
- No I/O operations.
- So, what is what we have?

"Small Cooker"

"Small Cooker"

Architecture

- 7 bit literals;
- 3 built-in,
- 32 user-definable functions;
- ROM – to store functions (256 bytes);
- RAM – to evaluate expressions (256 bytes);
- 8 bit data bus;
- 8/9 bit address bus.

Functions

Built-in:

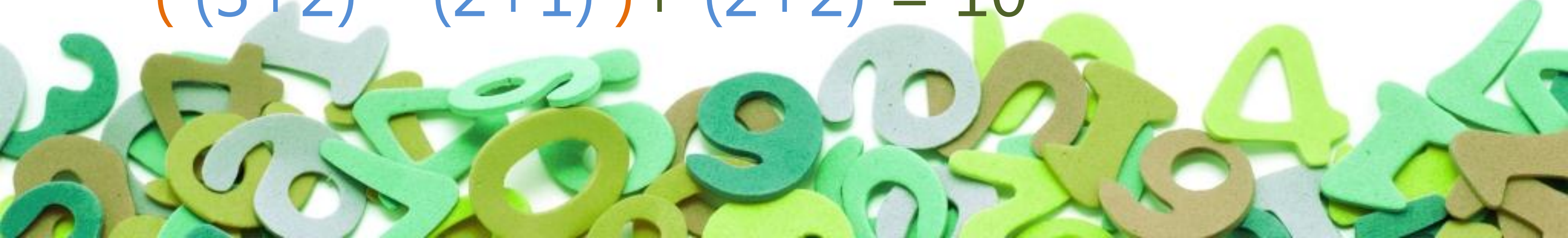
- **inc**(x) : x+1=inc(x)
- **dec**(x) : x-1=dec(x)
- **If** cond **then** exp1 **else** exp2

Plus user-definable functions.

Note: „0" represents True, any other value is False.

Computational Model

- Library: function definitions
  f(x,y)   :=x+y
  g(x,y,z) :=f(x,y)-f(y,z)
- Program: closed expression (const, func.)
  g(3,2,1)+f(2,2)
- Execution: rewriting rules
  (f(3,2) – f(2,1)) + (2+2) =
  ((3+2) – (2+1))+ (2+2) = 10



The Challenge

Challenges

- How to represent expressions;
- How to represent functions;
- Parenthesis, operation priorities;
- Argument passing;
- Evaluation strategy;
- How to model with integrated circuits;
- Physical implementation.

Encoding Scheme

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Hex	Value
0	0	0	0	0	0	0	0	00	zero / true
0	c 1	c 1	c 1	c 1	c 0	c 1	c 1	00- 7B	constant last const.
0	1	1	1	1	1	a	a	7C 7F	argument 1..4
1	F	f	f	f	f	a	a	80-FE	function
1	1	1	1	1	1	0	0	FC	dec
1	1	1	1	1	1	0	1	FD	If
1	1	1	1	1	1	1	0	FE	inc
1	1	1	1	1	1	1	1	FF	EOX

Function arity: 1..4

map 8 bit value to 3 bit class

Note: Constant functions must have at least one argument.

Function Encoding

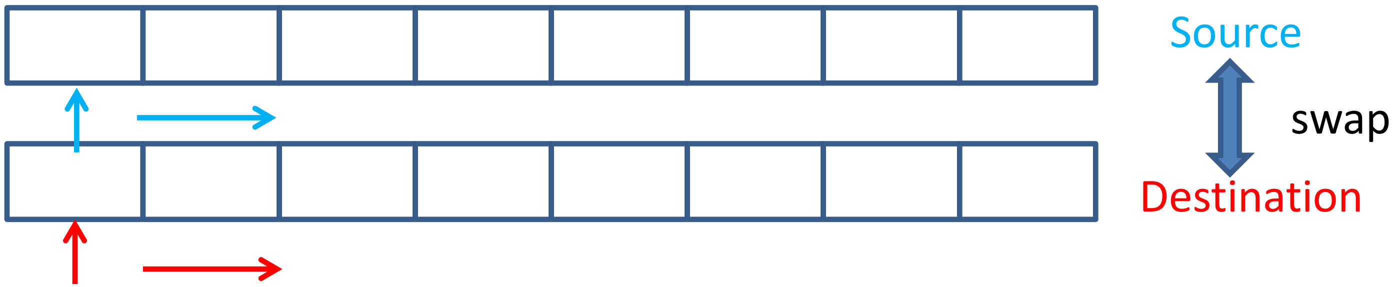
„`%1fff ffaa`"
Function address in binary: %`ffff f000`.
($00, $08, $10, … , $E8, $F0)

Number of arguments: aa=00, 01, 10, 11 denote 4, 3, 2, 1 arguments respectively.

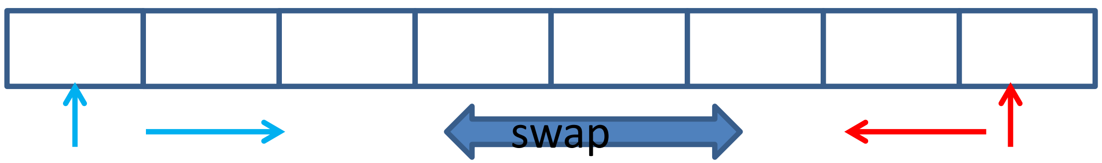
- Arity is encoded in function „id"-> efficient
- No real function „id", id gives instantly the function physical address.

Memory Models

1.) Separate Source and Destination Memory



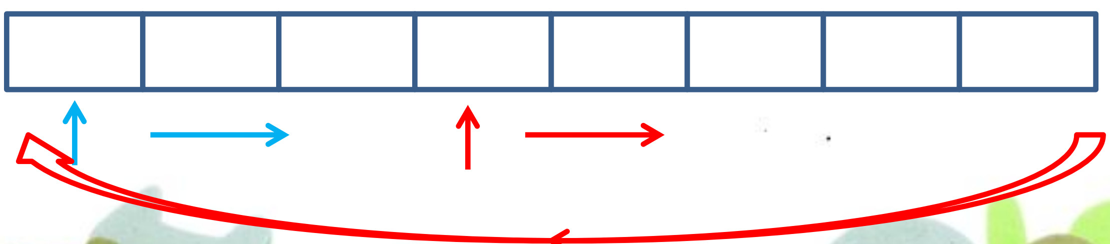
2.) Shared Memory, Source increasing/Destination decreasing



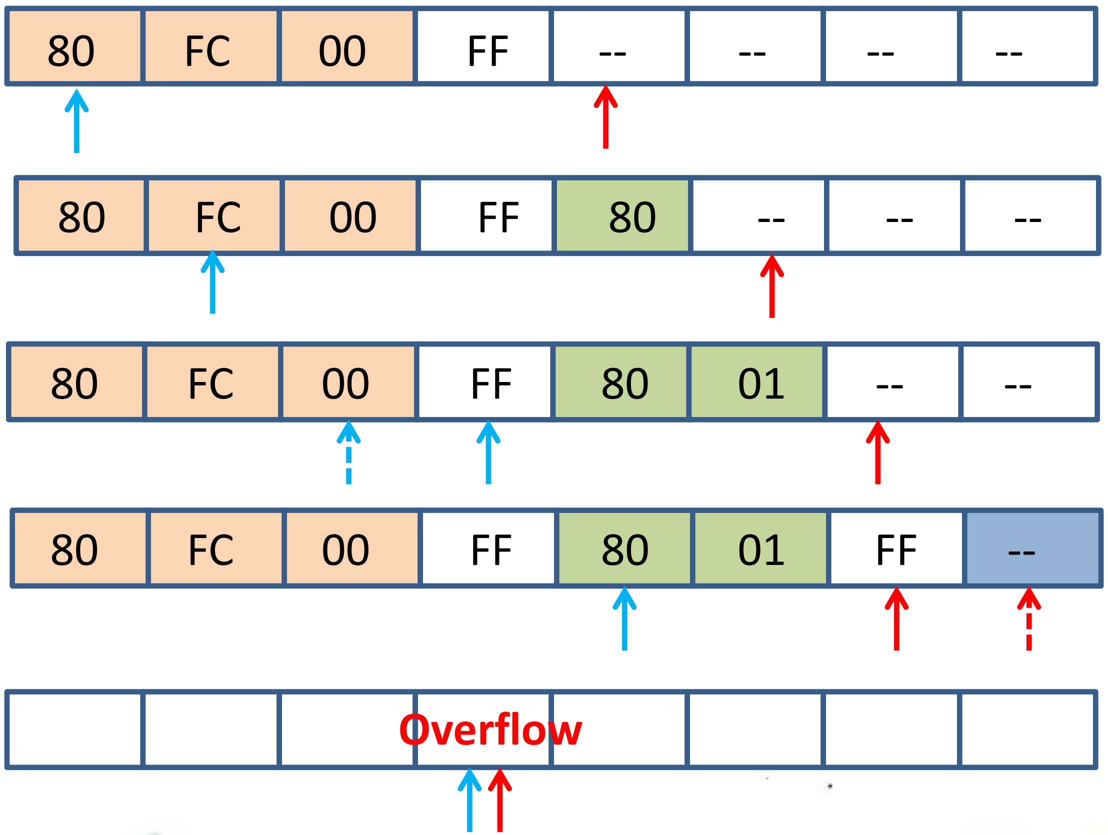
3.) Shared Memory, Source is „following” Destination

Optimized storage

„Step over”



Index Chase



80 FC 00 FF -- -- -- --
80 FC 00 FF 80 -- -- --
80 FC 00 FF 80 01 -- --
80 FC 00 FF 80 01 FF --
Overflow

Parenthesis

- 2*fac(fac(2)+fac(1))?
- Everything is right-associative (PPN):
- * 2 fac + fac 2 fac 1
- Function Scope? (simple, hw-based!):
- Store arity in AC in the beginning.
- AC:=AC-1+ arity of the next symbol.
- End of Scope, if AC=0.

Example: Parenthesis

- Scope of the first „fac“:
- if 2 fac + fac 2 fac 1 if … ac=1
- if 2 fac + fac 2 fac 1 if … ac=2=1-1+2
- if 2 fac + fac 2 fac 1 if … ac=2=2-1+1
- if 2 fac + fac 2 fac 1 if … ac=1=2-1
- if 2 fac + fac 2 fac 1 if … ac=1=2-1+1
- if 2 fac + fac 2 fac 1 if … ac=**0**=1-1

Evaluation Issue

- fac(n):= if n=0 then 1 else n*fac(n-1)
- fac(1)=?
- if 1=0 then 0 else 1*fac(0)
- If 1=0 then 0 else 1*(if 0=0 then 1 else 0*fac(-1))
- If 1=0 then 0 else 1*(if 0=0 then 1 else 0*(if -1=0 then 0 else -1*fac(-1))) ....

Evaluation Goals

- Must terminate (applicative results in infinite loop)
- Must be efficient.
- Must be simple (suitable to be represented in hardware directly).

Passing Arguments

- fac (n):= n*fac(n-1)
- fac(add(5,4))?
- ack(add(fac(fac(2)+fac(1)),fac(1)),1)?

- No stack;
- No dedicated memory for parameters;
- Still „arbitrary" depth of function calls.

Evaluation - If

- if cond exp1 exp2   =>

- exp1, if cond=00
- exp2, if cond<>00, but it is a literal
- Otherwise „if” is not evaluated.

Evaluating – inc, dec…

- inc exp   =>
- exp+1, if „exp” is constant.
- dec is reduced analogously.
- Other functions are evaluated/”called”, if all of their arguments are constants.
- Corollary: all parameters passed to functions are constants.

Evaluation Strategy

- Many strategies co-exist.
- Scan input expression symbol by symbol.
- 1.) Copy symbol from source to output.
- 2.) Evaluate functions (if, inc, dec, and user-defined) if possible.
- Next cycle: output becomes the input.
- Stop, if the first symbol is constant.

Termination

- inc, dec terminates immediately, thus reduces the length of expression.
- if cond exp1 exp2 – if cond is constant, then the length of expression is decreased.
- Eventually everything is based on/can be reduced to the three built-in functions.
- Sooner or later, all „if", „inc", „dec" are evaluated/reduced (if possible).

Examples

Elementary Functions

- I(x):=   x

7F FF

- C(x):=   c

„c” FF, where c=00..7B

constant 7C can be encoded as: inc(7B)

FC 7B FF

Few Predicates

- not(x):= if x then 1
  else 0

FE 7F 01 00 FF

- =(x,y):= if y=0 then x
  else =(dec(x),dec(y))

FE 7E 7F 81 F8 7F F8 7E FF

- >(x,y):= if x=0 then not(y)
  else >(dec(x),dec(y))

Some Functions

- add(x,y):= if y=0 then x
  else inc(add(x,dec(y)))

- mul(x,y):= if y=0 then 0
  else add(x,mul(x,dec(y)))

- fac(n):= if n=0 then 1
  else mul(n,fac(dec(n)))

Case...

```
case x
when        c0    then b0
when        c1    then b1
...
when        cn    then bn
otherwise               b      :=
if=(x,c0)   then b0
            else  if    =(x,c1)     then b1
                                         else  ...
```

Ackermann function

```
ack(m,n):=
if      m=0 then inc(n)
elsif  n=0  then ack(dec(m),1)
else              ack(dec(m),ack(m,dec(n)))
```

FE 7E FC 7F FE 7F 81 F8 7E 01 81 F8 7E 81 7E F8 7F FF

Functions Encoded

```
add:= 00/81:    FE 7E 7F FC 81 7F F8 7E FF

mul:= 10/89:    FE 7E 00 81 7F 89 7F F8 7E FF

fac:= 20/90:    FE 7F 01 89 90 F8 7F 7F FF
```

FE	if		
FC	inc	F8	dec
00, 01	constant	FF	end of exp.
7E, 7F	arguments	y, x	

1+1=2

add(x,y):= FE 7E 7F FC 81 7F F8 7E FF

--:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00:	81	01	01	FF	FE	01	01	FC	81	01	F8	01	FF	FC	81	01
10:	00	FF	FC	FE	01	00	FC	81	00	F8	01	FF	FC	FC	81	00
20:	00	FF	FC	FC	FE	00	00	FC	81	00	F8	00	FF	FC	FC	00
30:	FF	FC	01	FF	02	FF	...									

1+1=„02”

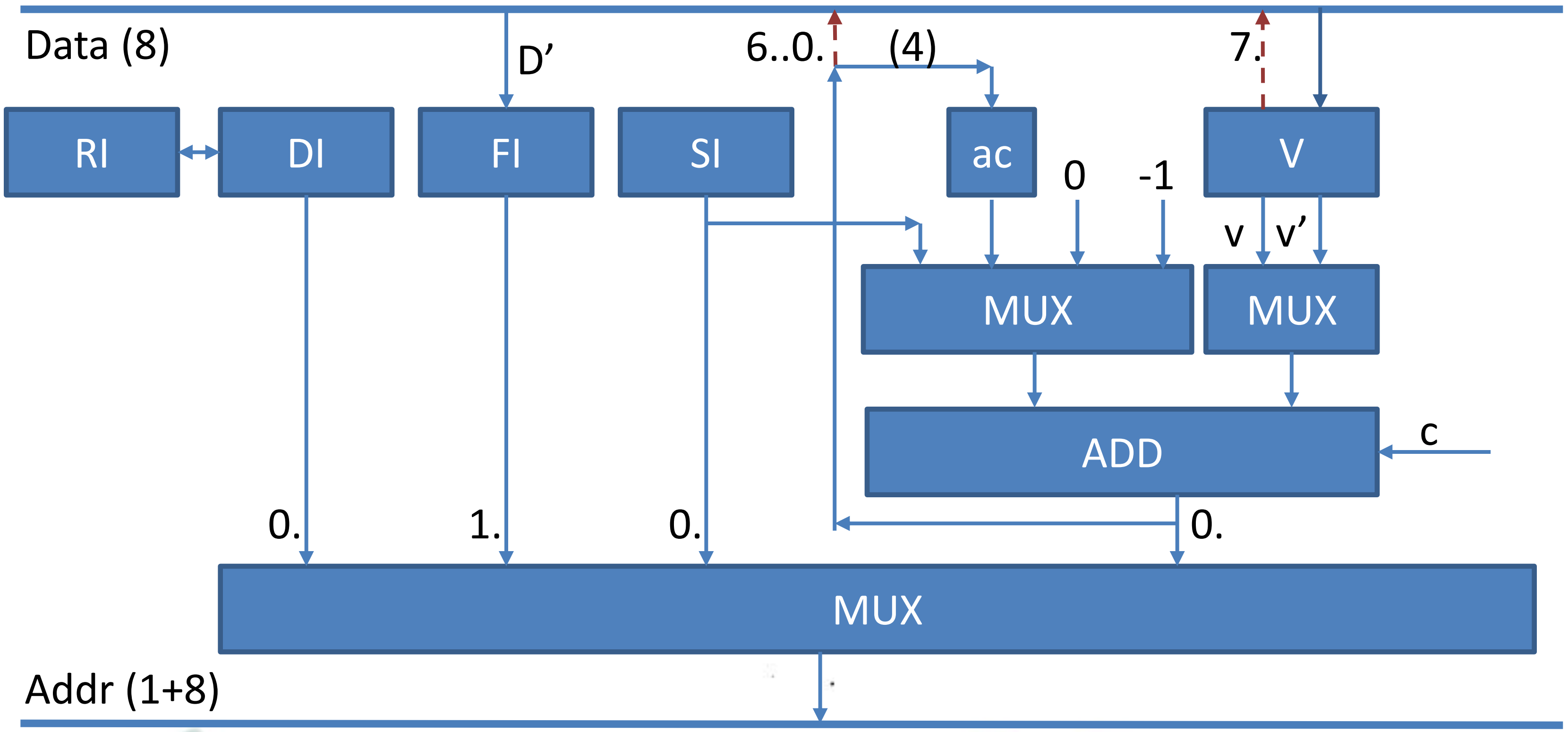
FE	if	81	Add
FC	inc	F8	dec
00, 01, 02	constant	FF	end of exp.
7E, 7F	arguments	y, x	

fac(5)=120

--:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00:	FF	FC	FC	FC	75	FF	FC	FC	76	FF	FC	77	**FF**	**78**	**FF**	FC
10:	FC	FE	03	71	FC	81	71	F8	03	FF	FC	FC	FC	FC	FC	81
20:	71	02	FF	FC	FC	FC	FC	FC	FE	71	02	FC	81	02	F8	71
30:	FF	FC	FC	FC	FC	FC	FC	81	02	70	FF	FC	FC	FC	FC	FC
40:	FC	FE	02	70	FC	81	70	F8	02	FF	FC	FC	FC	FC	FC	FC
50:	FC	81	70	01	FF	FC	FC	FC	FC	FC	FC	FC	FE	70	01	FC
60:	81	01	F8	70	FF	FC	FC	FC	FC	FC	FC	FC	FC	81	01	6F
70:	FF	FC	FC	FC	FC	FC	FC	FC	FC	FE	01	6F	FC	81	6F	F8
80:	01	FF	FC	FC	FC	FC	FC	FC	FC	FC	FC	81	6F	00	FF	FC
90:	FC	FC	FC	FC	FC	FC	FC	FC	FE	6F	00	FC	81	00	F8	6F
A0:	FF	FC	FC	FC	FC	FC	FC	FC	FC	FC	FC	81	00	6E	FF	FC
B0:	FC	FC	FC	FC	FC	FC	FC	FC	FC	FE	00	6E	FC	81	6E	F8
C0:	00	FF	FC	FC	FC	FC	FC	FC	FC	FC	FC	FC	6E	FF	FC	FC
D0:	FC	FC	FC	FC	FC	FC	FC	6F	FF	FC	FC	FC	FC	FC	FC	FC
E0:	FC	70	FF	FC	FC	FC	FC	FC	FC	FC	71	FF	FC	FC	FC	FC
F0:	FC	FC	72	FF	FC	FC	FC	FC	FC	73	FF	FC	FC	FC	FC	74

Architecture

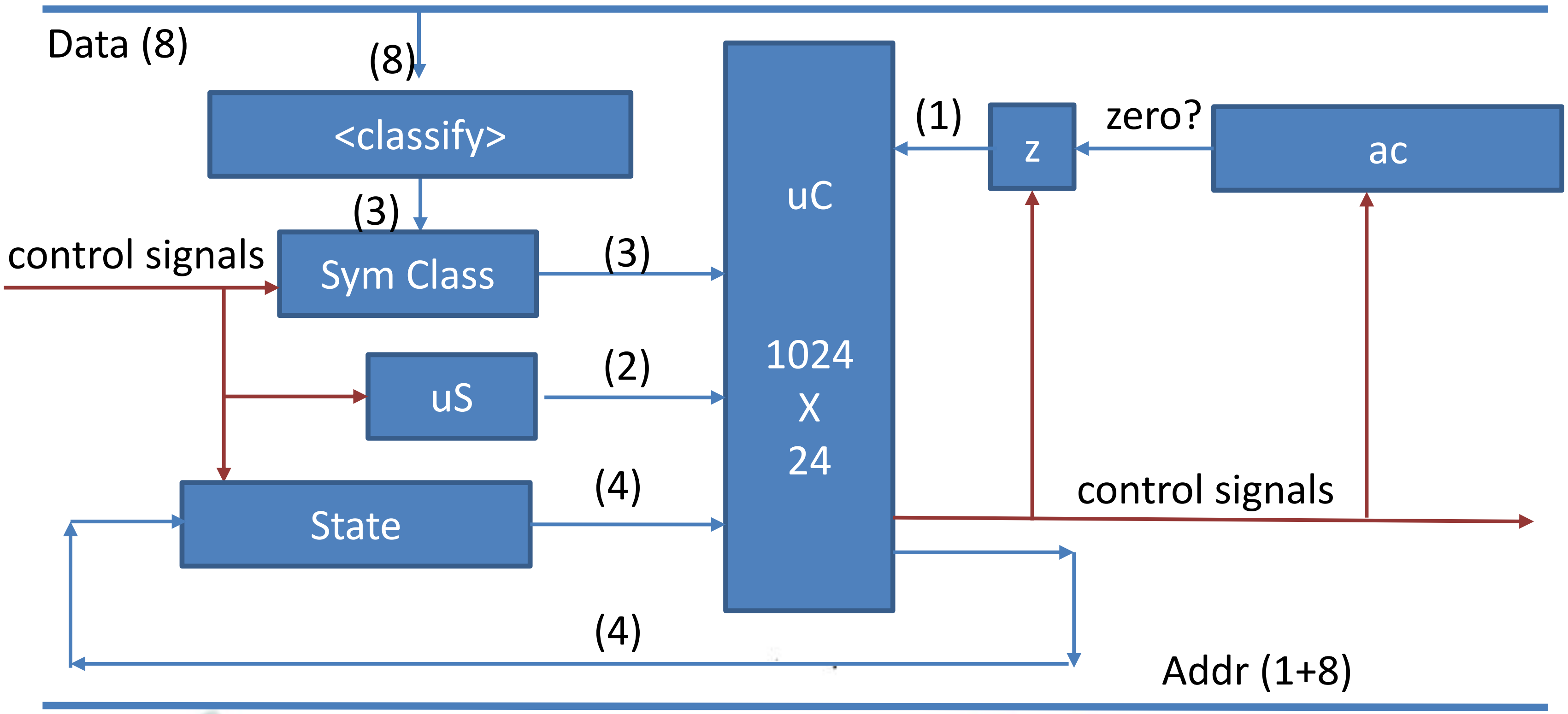
Architecture (registers)



Registers

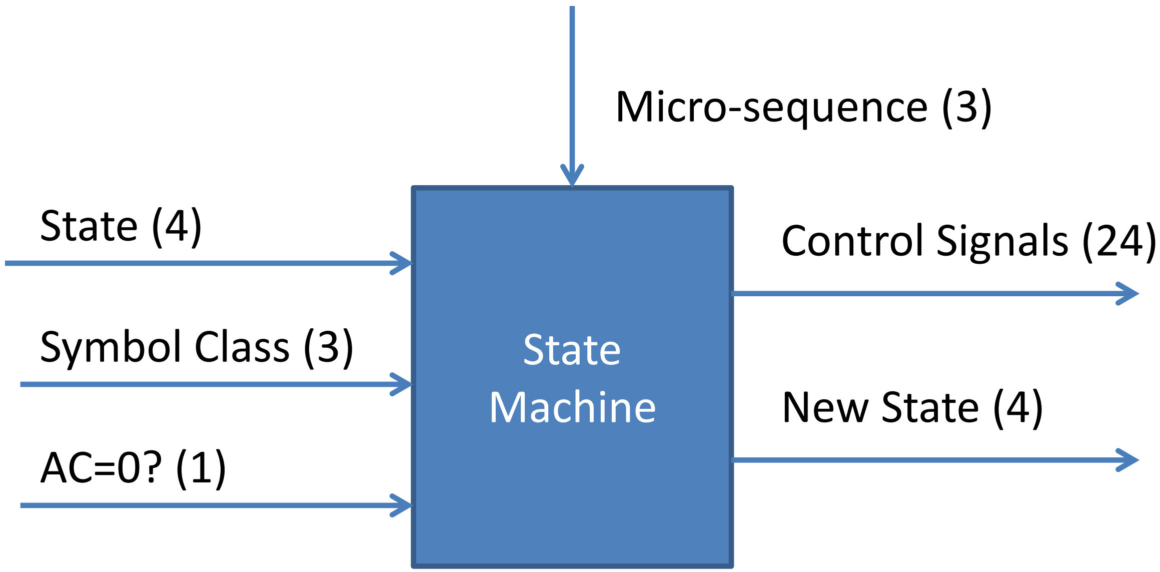
- SI – source index (8 bit)
- DI – destination index (8 bit)
- FI – function index (8 bit)
- RI – redex index (8 bit)
- AC – argument count (4 bit)
- V – value (8 bit)
- S – state (4), Z – zero (1), C – class (3)

Architecture (ctrl unit)



Control Unit

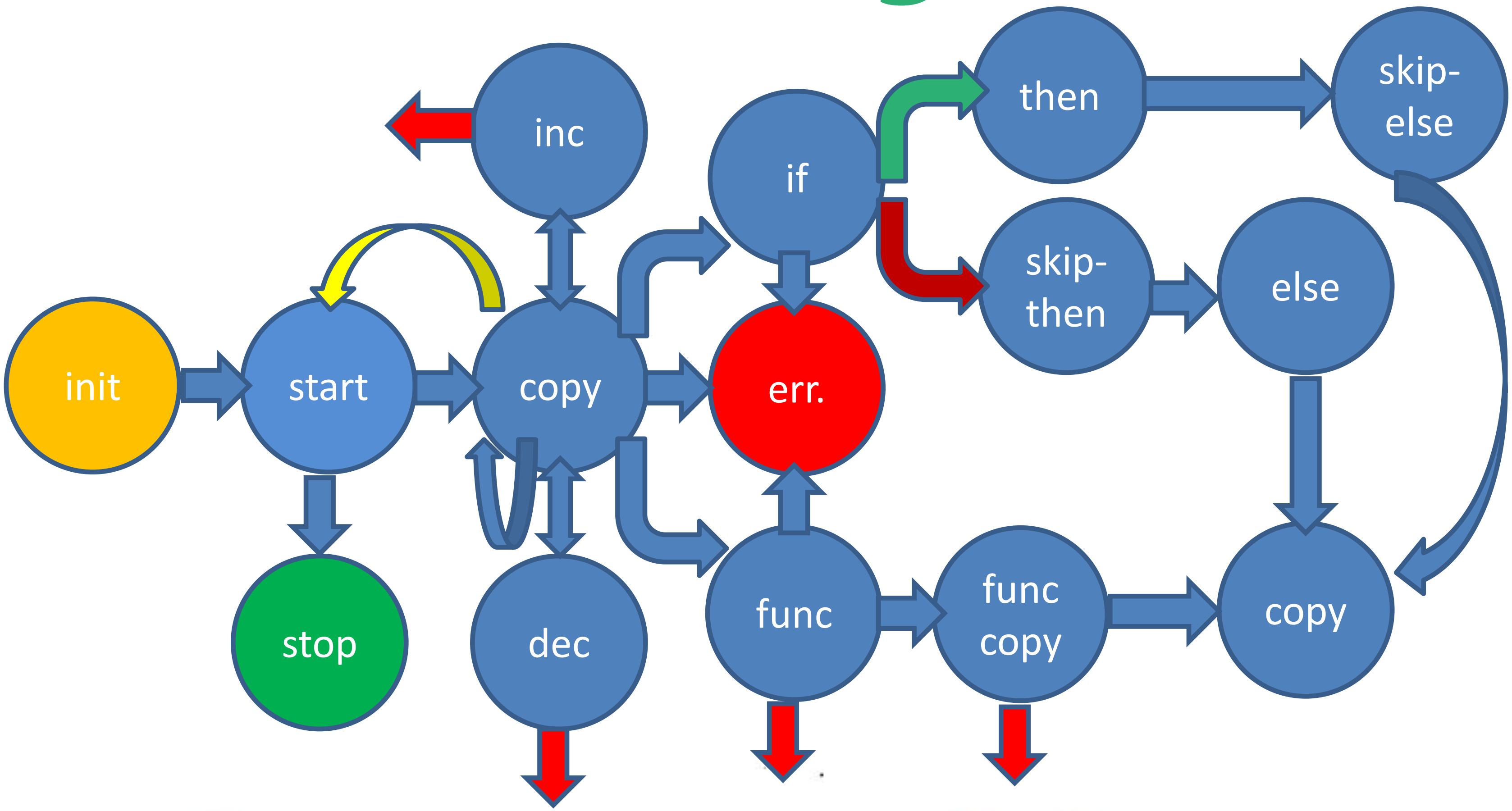
- Turing-machine
- 16 states



States

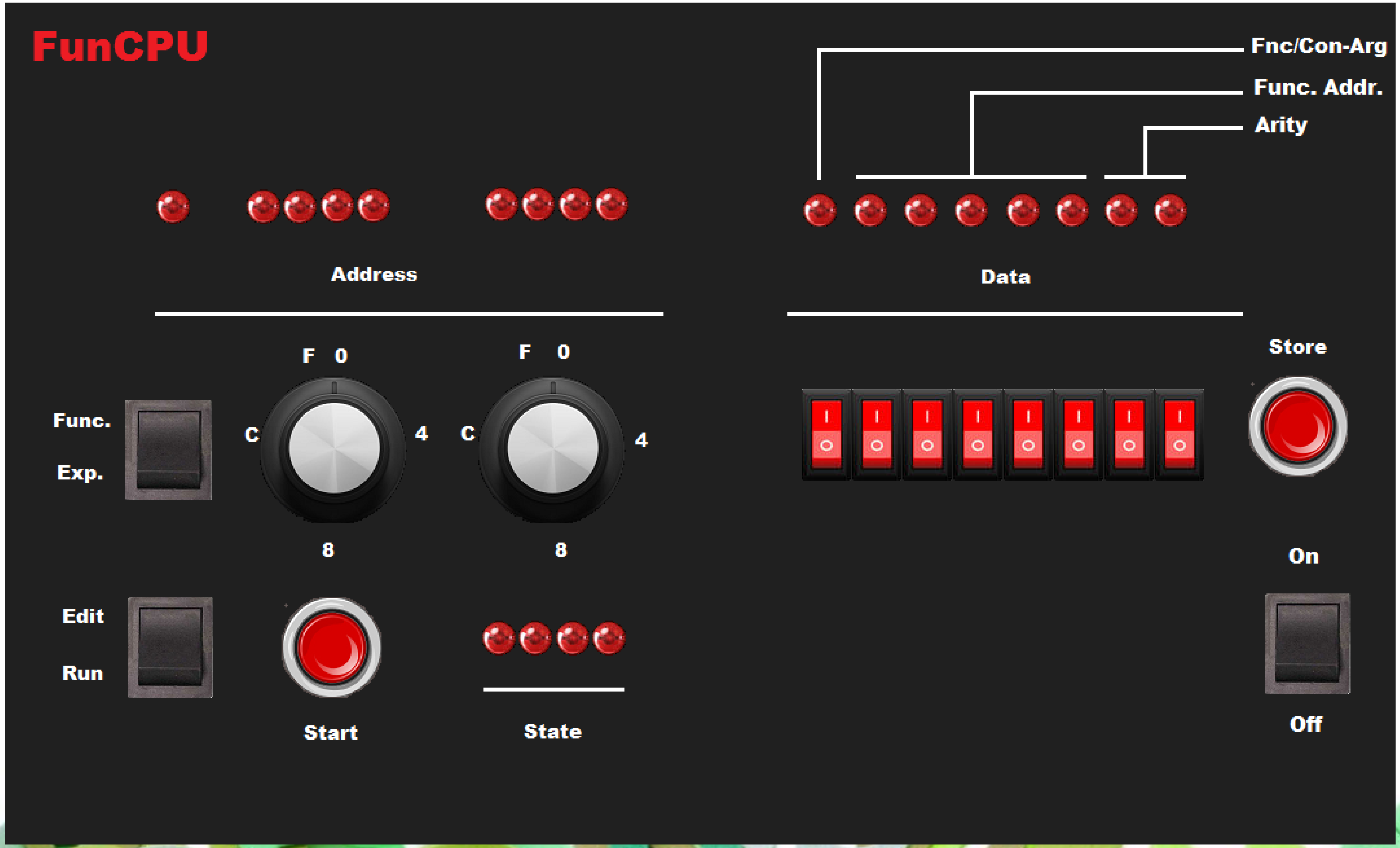
- Init - initialization
- Copy loop – copying
- If – evaluation of „if“
- inc/dec – evaluation of inc/dec
- Func/copy – evaluation of user functions
- Error – syntactical or internal error
- Stop – final state

State Diagram



Implementation

Control Board



Physical Implementation

- 74HC163/161 – FI, AC / uC, SI, DI
- 74HC273 - RI, V
- 74HC273 - S, Z, C
- 74HC153 - 4 to 1 MUX (Addr, ALU src 1)
- 74HC157 - 2 to 1 MUX (ALU src 2)
- 74HC241 – octal tristate buffer
- MCM2708 – 1Kb EPROM (uCode,Classifier)

Computability

Overcoming Limitations

- Increase argument count: pairing functions, e.g. Cantor:
- $\pi \quad (x, y) := \frac{1}{2} * (x + y) * (x + y + 1) + y$
- List: Gödel-encoding
- $list(x_1, x_2, \ldots, x_n) := 2^{x_1} * 3^{x_2} * \cdots * {p_n}^{x_n}$
- Only theoretical approaches due to the low precision of number representation.

Atomic Functions

- c(x) `CC FF`
- s(x) `FC`
- $\pi_i(x_1, x_2, x_3, x_4)$ `7F/7E/7D/7C FF`

- `FE 7F 7E FE F8 7F 7D 7C FF`

Composition

- $\mathrm{h}(x_1, x_2, x_3, x_4) = f(g_1(x_1, x_2, x_3, x_4), \ldots, g_k(x_1, x_2, x_3, x_4))$
- f **83**
- g_1 **93**, g_2 **A3**, g_3 **B3**, g_4 **C3**

83 93 7F 7E 7D 7C A3 7F 7E 7D 7C
B3 7F 7E 7D 7C C3 7F 7E 7D 7C FF

Primitive Recursion

- $\mathrm{h}(0, x_1, x_2) = f(x_1, x_2)$
- $\mathrm{h}(s(y), x_1, x_2) =$

  $\mathrm{g}(y, \mathrm{h}(y, x_1,\ x_2), x_1,\ x_2)$
- h **82**, f **91**, g **A3**

FE 7F 91 7E 7D A3 F8 7F

82 F8 7F 7E 7D 7E 7D FF

μ-operator

- $\mu(f)\ (x_1,\ x_2, x_3\ ) = z\ \ \equiv def$

  $f\ (z, x_1,\ x_2, x_3\ ) = 0$

  $f\ (i, x_1,\ x_2, x_3\ ) > 0\ \ \forall i < z$
- f **82**

FE 82 7F 7E 7D 7C 7F

82 FC 7F 7E 7D 7C FF

Improvements

Limitations

- Only integer types with limited precision.
- No advanced and/or user-defined types (e.g. list, record, tuples, etc.).
- Supports only 4 arguments.
- Small memory.
- No support for higher order functions.

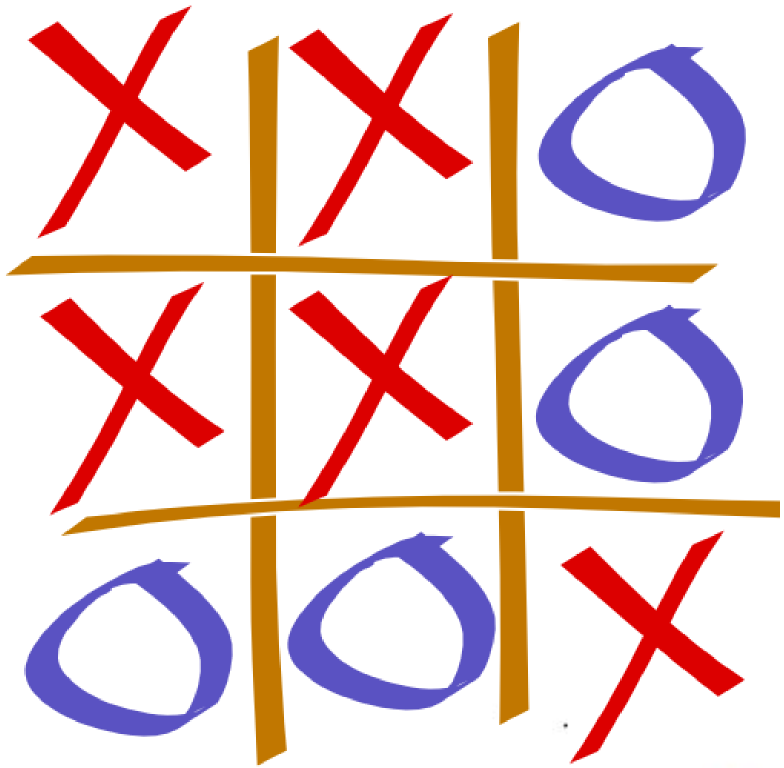
Limitations – cont.

- Expression evaluation is slow and inefficient, requires more storage due to low-level basic functions.
- Functions cannot be embedded in arbitrary depth:
  f(f(f(f(… is fine, but f(f(f(f(f(… is not.
- Note: f(f(f(f(const,f(… is ok again.
- Where f is a function with 4 arguments.

Enhancements

- More built-in functions (e.g. +, -, *, etc.)
- Increased memory
- Advanced timing: clock period reduced
- Inreased register size (higher precision, deeper function embedding)
- More efficient expression evaluation
- Exploit parallelism

More Challenges…



Questions?

References / Sources

- Homebrew Computers Web-Ring
  http://members.iinet.net.au/~daveb/simplex/ringhome.html
- Time Fracture – John Doran
- http://www.timefracture.org/
- Mark 1 FORTH - Andrew Holme
  http://www.aholme.co.uk/Mk1/Architecture.htm
- Magic-1 – Bill Buzbee
  http://www.homebrewcpu.com/
- Big Mess of Wires - Steve Chamberlin
  http://www.bigmessowires.com/bmow1/
- Relay Computer – Harry Porter
  http://web.cecs.pdx.edu/~harry/Relay/