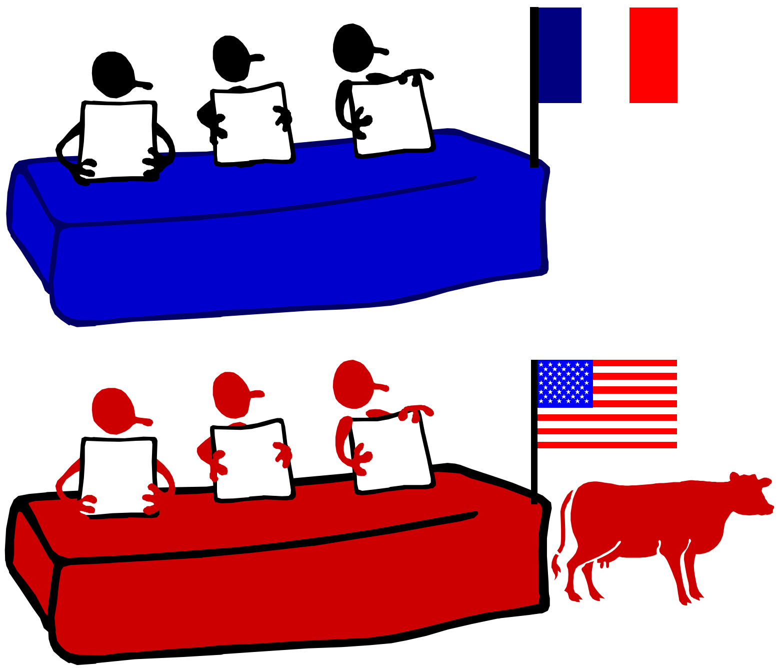
# Defensive Finite State Automata

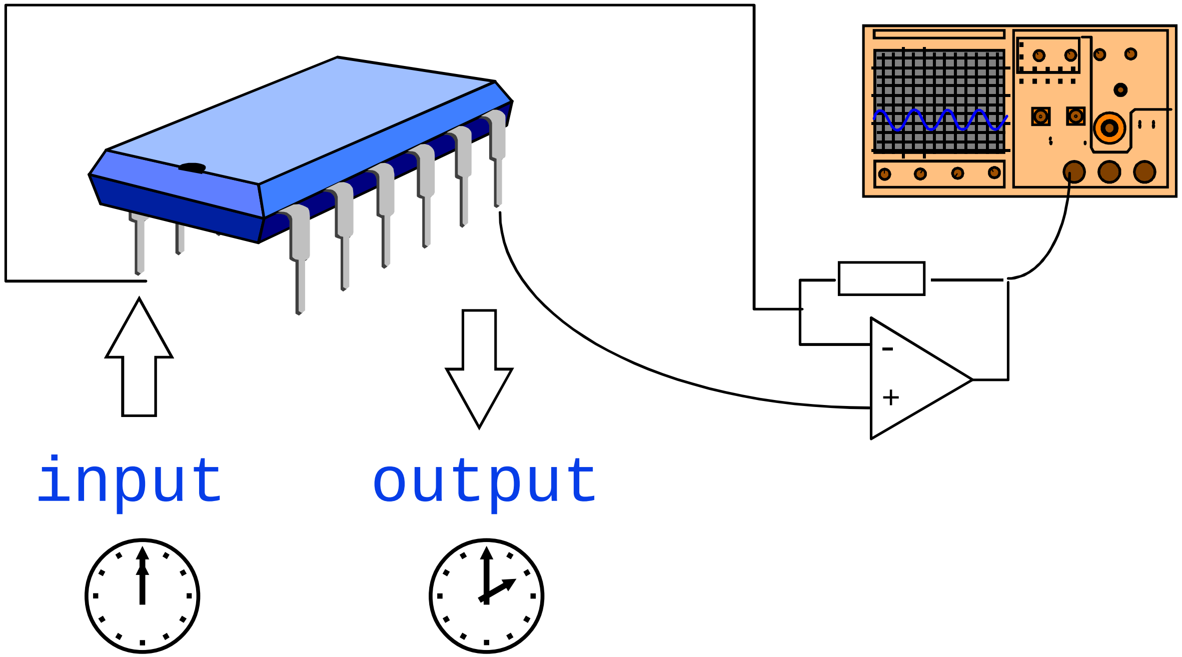
# *POWER ATTACKS*

- Seattle, 1999.
- US and French delegates negotiate under which conditions beef could be imported to France.
- «The Sun » sends a journalist to investigate:



# *SIDE CHANNEL ATTACKS*

**Measure the circuit's processing time and current consumption to infer what happens inside it.**



# Logistics vs. Strategy

- How to get countermeasures?

  Logistics

- Where to use these countermeasures?

  Strategy

Here we address strategy.

# The Subleq Machine

Subleq is a Turing-complete machine having only one instruction.

**subleq a b c**

① *(b)=*(b)-*(a)

② if the result is negative or zero, go to c else execute the next instruction.

# The Subleq Machine

Since subleq has only three arguments and since there is no confusion of instructions possible (there is only one!), a subleq code can be regarded as a sequence of triples.

$a_1\ b_1\ c_1$

$a_2\ b_2\ c_2$

$a_3\ b_3\ c_3$

:

# …interleaved with data

Since data can be embedded in the code, the sequence of triples can be interleaved with data. For instance:

**$a_1$ $b_1$ $c_1$**
**$data_1$ $data_2$**
**$a_2$ $b_2$ $c_2$**
**$data_3$**
**$a_3$ $b_3$ $c_3$**
:

# How does it work?

```
*b = *b-*a;

if (*b≤0)
        program_counter = c;
  else
        program_counter = program_counter+3;
```

# Genealogy

Subleq is an OISC ("One Instruction Set Computer) which comes from the Minsky machine concept.

The Minsky machine is a register machine with only two instructions: "**increment**" and "**decrement-and-branch**".

# Allowing for comfort

Memory is loaded with instructions and data altogether (no distinction).

Hence the code can potentially self-modify and consider that any cell is a, b or c.

We can pre-store constants (like 0,1 etc)

e.g. we devote a cell called Z to contain zero, N to contain -1

# What does this do?

```
subleq Z Z c
```

# JMP c

```
subleq Z Z c
```

# What does this do?

```
subleq a a $+1
```

# CLR a

```
subleq a a $+1
```

# What does this do?

```
CLR    b
subleq a Z $+1
subleq Z b $+1
CLR    Z
```

# MOV a b

```
subleq b b $+1        *b=0
subleq a Z $+1        Z=-*a
subleq Z b $+1        *b=0-(-*a)=*a
subleq Z Z $+1        Z=0
```

# What does this do?

```
subleq a Z $+1
subleq b Z $+1
CLR    c
subleq Z c $+1
CLR    Z
```

# ADD a b c

```
subleq a Z $+1      Z=0-*a
subleq b Z $+1      Z=-*a-*b
subleq c c $+1      *c=*c-*c=0
subleq Z c $+1      *c=0+*a+*b
sublez Z Z $+1      Z=0
```

# What does this do?

```
CLR    t
CLR    s
subleq a t $+1
subleq b s $+1
subleq s t $+1
CLR    c
CLR    s
subleq t s $+1
subleq s c $+1
```

# SUB a b c

```
subleq t t $+1    *t=0
subleq s s $+1    *s=0
subleq a t $+1    *t=-*a
subleq b s $+1    s=-*b
subleq s t $+1    t=-*a+*b
subleq c c $+1    *c=0
subleq s s $+1    *s=0
subleq t s $+1    *s=0-(-*a+*b)=*a-*b
subleq s c $+1    *c=0-(*a-*b)=*b-*a
```

not really optimal code just to illustrate

# What does this do?

```
CLR    t
subleq a t $+1
CLR    s
subleq t s $+1
subleq b s c
```

# BLE a b c

```
subleq t t $+1      t=0
subleq a t $+1      *t=-*a
subleq s s $+1      *s=0
subleq t s $+1      *s=*a
subleq b s c        *s=*a-*b
                if *a-*b≤0 goto c
```

# What does this do?

```
CLR    t
subleq a t $+1
CLR    s
subleq b s $+1
subleq s t $+1
subleq N t c
```

# BHI a b c

```
subleq t t $+1      *t=0
subleq a t $+1      *t=-*a
subleq s s $+1      *s=0
subleq b s $+1      *s=-*b
subleq s t $+1      *t=-*a+*b
subleq N t c        *t=-*a+*b-(-1)
                if *b-*a+1≤0 goto c
```

# What have we got so far?

| | |
|---|---|
| JMP a | goto a |
| MOV a b | *b=*a |
| SUB a b c | *c=*b-*a |
| ADD a b c | *c=*b+*a |
| BHI a b c | if *b-*a+1≤0 goto c |
| | if *b<*b+1≤*a goto c |
| | if *b<*a goto c |
| | if *a>*b goto c |
| BLE a b c | if *a-*b≤0 goto c |
| | if *a≤*b goto c |
| CLR a | *a=0 |

# What does this do?

```
MOV    b v
MOV    a w
CLR    c
subleq N c $+1
subleq w v $+4
subleq Z Z $-8
```

# What does this do?

```
MOV    b v              *v=*b
MOV    a w              *w=*a
CLR    c                *c=0
subleq N c $+1          *c=*c-(-1)
subleq w v $+4
subleq Z Z $-8
```

# What does this do?

```
MOV    b v          *v=*b
MOV    a w          *w=*a
CLR    c            *c=0
subleq N c $+1      *c++
subleq w v $+4      *v=*v-*w if(*v≤0)
subleq Z Z $-8      else
```

# What does this do?

```
MOV    b v           *v=*b
MOV    a w           *w=*a
CLR    c             *c=0
subleq N c $+1       *c++
subleq w v $+4       *v=*v-*w if(*v≤0)
subleq Z Z $-8       else
```

# What does this do?

```
*v=*b
*w=*a
*c=0
*c++
*v=*v-*w if(*v≤0)
else
```

# DIV a b c

```
*v=*b

*c=0
*c++
*v=*v-*a if(*v≤0)
else
```

# DIV a b c

```
*v=*b
*c=0
*c++
*v=*v-*a if(*v≤0)
else
```

# DIV a b c

```
a = 5; b = 45; c = 0; Label[more]; c++; b = b - a;
If[b ≤ 0, Goto[finish], Goto[more]]; Label[finish];
Print[c];
```

9

```
*v=*b
*c=0
*c++
*v=*v-*a if(*v≤0)
else
```

for nonzero arguments (else need one more test)

# What does this do?

```
CLR    u;v;w
MOV    b v
subleq N w $+1
subleq u u $+1
subleq a u  $+1
CLR    c
subleq u c $+1
subleq w v $+4
subleq Z Z $-8
```

# What does this do?

```
CLR    u;v;w        *u=*v=*w=0
MOV    b v          *v=*b
subleq N w $+1      *w=0-(-1)=1
subleq u u $+1      *u=0
subleq a u  $+1     *u=-*a
CLR    c            *c=0
subleq u c $+1
subleq w v $+4
subleq Z Z $-8
```

# What does this do?

```
*v=*b
*w=0-(-1)=1

*u=-*a
*c=0
```

```
subleq u c $+1
subleq w v $+4
subleq Z Z $-8
```

# What does this do?

```
*v=*b
*w=1

*u=-*a
*c=0
```

```
subleq u c $+1
subleq w v $+4
subleq Z Z $-8
```

# What does this do?

```
*v=*b
*w=1

*u=-*a
*c=0
```

```
subleq u c $+1        *c=*c-*u=*c+*a
subleq w v $+4
subleq Z Z $-8
```

# What does this do?

```
subleq u c $+1
subleq w v $+4
subleq Z Z $-8
```

```
*v=*b
*w=1

*u=-*a
*c=0
*c=*c+*a
*v=*v-*w  if…
```

# What does this do?

```
                    *v=*b
                    *w=1

                    *u=-*a
                    *c=0
subleq u c $+1      *c=*c+*a
subleq w v $+4      *v=*v-1   if…
subleq Z Z $-8
```

# What does this do?

```
                              *v=*b
                              *w=1

                              *u=-*a
                              *c=0
subleq u c $+1                *c=*c+*a
subleq w v $+4                *v--          if…
subleq Z Z $-8
```

# What does this do?

```
                   *v=*b
                   *w=1

                   *u=-*a
                   *c=0
subleq u c $+1     *c=*c+*a
subleq w v $+4     *v--; if(*v≤0)
subleq Z Z $-8
```

# What does this do?

```
                    *v=*b
                    *w=1

                    *u=-*a
                    *c=0
subleq u c $+1      *c=*c+*a
subleq w v $+4      *v--; if(*v≤0)
subleq Z Z $-8      else
```

# What does this do?

```
*v=*b
*w=1

*c=0
```

| | |
|---|---|
| `subleq u c $+1` | `*c=*c+*a` |
| `subleq w v $+4` | `*v--; if(*v≤0)` |
| `subleq Z Z $-8` | `else` |

# What does this do?

```
                                   *v=*b


                                   *c=0
subleq u c $+1                     *c=*c+*a
subleq w v $+4                     *v--; if(*v≤0)
subleq Z Z $-8                     else
```

# What does this do?

```
                       *v=*b

                       *c=0
subleq u c $+1         *c=*c+*a
subleq w v $+4         *v--; if(*v≤0)
subleq Z Z $-8         else
```

# What does this do?

```
*v=*b

*c=0
*c=*c+*a
*v--; if(*v≤0)
else
```

# MUL a b c

```
*v=*b



*c=0
*c=*c+*a
*v--; if(*v≤0)
else
```

# MUL a b c

```
*v=*b
*c=0
*c=*c+*a
*v--; if(*v≤0)
else
```

# MUL a b c

```
In[110]:= a = 3; b = 4; c = 0; Label[more]; c += a; b--;
          If[b ≤ 0, Goto[finish], Goto[more]]; Label[finish];
          Print[c];

          12
```

```
*v=*b
*c=0
*c=*c+*a
*v--; if(*v≤0)
else
```

for nonzero arguments (else need one more test)

# What does this do?

```
     MOV   a  L1
     data  Z
     data  Z
L1:  data  Z
```

# BRX a

```
     MOV    a  L1    *L1=*a
     data   Z
     data   Z
L1:  data   Z
```

**a very powerful instruction**

# What does this do?

```
    subleq b Z L1
    subleq Z Z L2
L1  subleq Z Z $+1
    subleq Z b c
L2  subleq Z Z $+1
```

# BEQ b c

```
    subleq b Z L1   Z=-*b if Z≤0
    subleq Z Z L2   else reset Z
L1  subleq Z Z $+1  reset Z
    subleq Z b c    *b=*b-0 if *b≤0
L2  subleq Z Z $+1
```

c

# What else do we need?

Boolean operations such as AND, XOR.

Assuming that we have AND, we can design the XOR:

$$A + B = \sum_{i=0}^{7} 2^i (B_i + A_i) = \sum_{i=0}^{7} 2^i (B_i \oplus A_i) + \sum_{i=0}^{7} 2^{i+1} B_i A_i = A \oplus B + 2(A \wedge B)$$

# Where is all this going?

The machine can do everything a smartcard can do.

Still, it's execution is hyper-regular.

Eliminates instruction-dependent leakage. Only leakage is data-dependent.

# Where is all this going?

A “reductionist” approach.

Push all security issues into the subleq machine.

If the subleq machine is side-channel resistant then no matter what algorithm we implement on it, the implementation is side-channel resistant!

# Where is all this going?

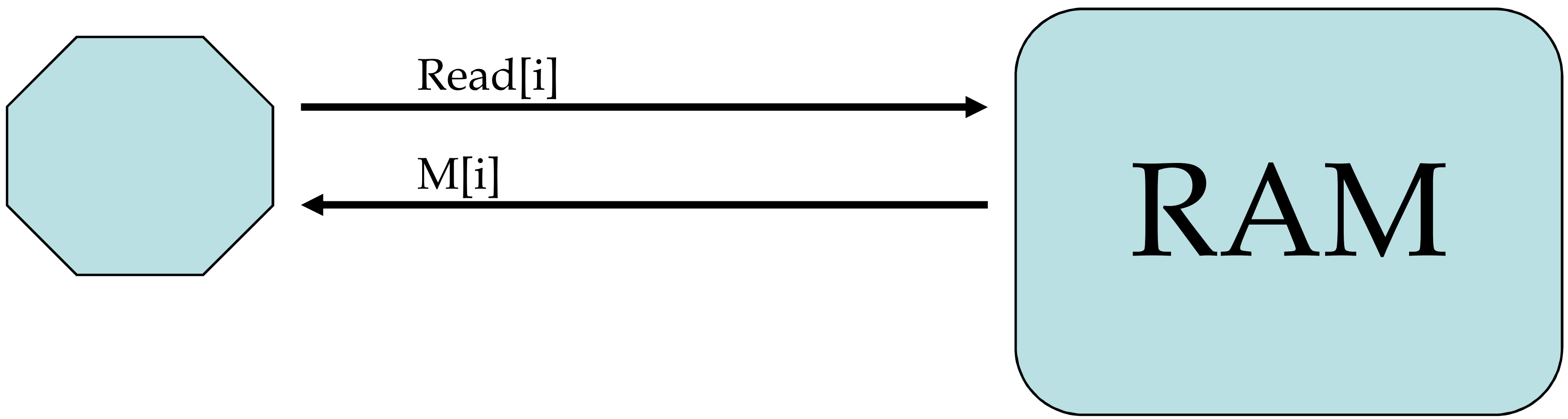
But any algorithm can be coded on the machine.

Hence it suffices to concentrate all effort on protecting the machine.

But the machine is very simple, hence (conceivably!) much easier to secure than an AES or RSA coprocessor.

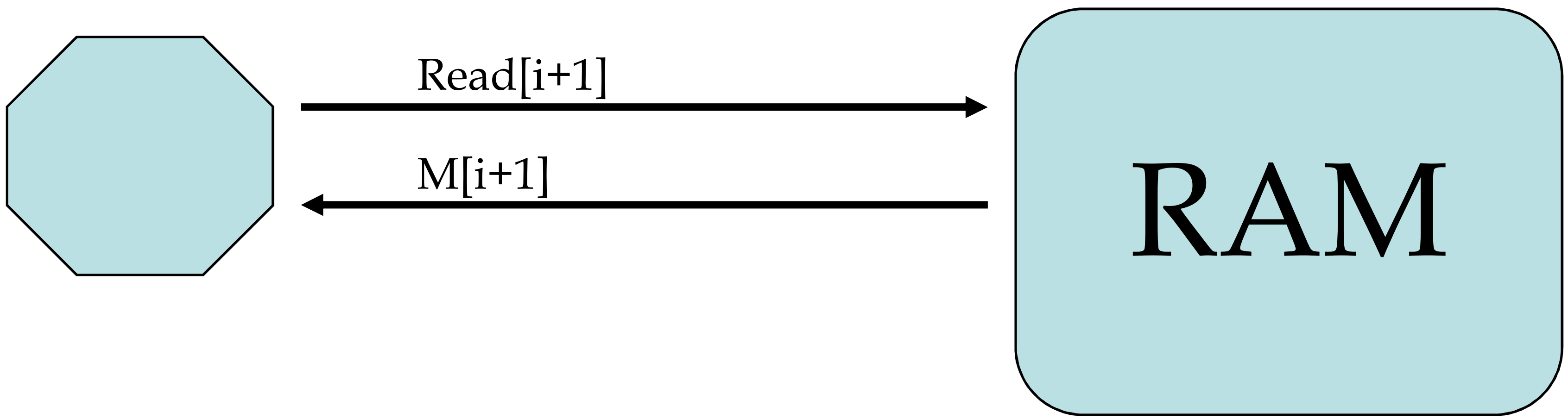
# Hardware Architecture

- We assume that we have a RAM initialized with the code.



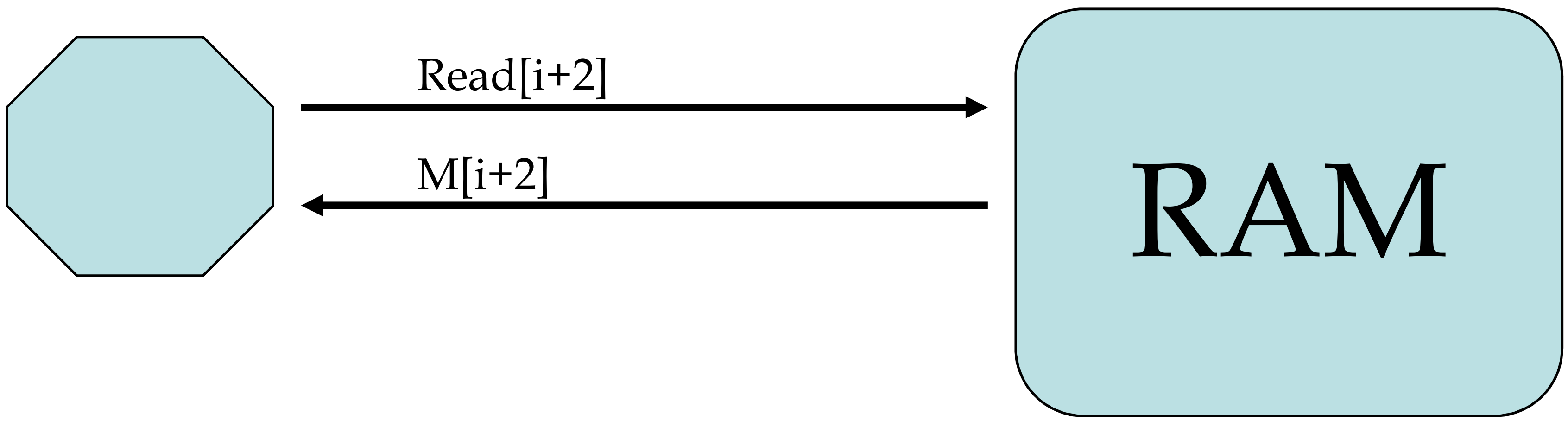
# Hardware Architecture

- We assume that we have a RAM initialized with the code.



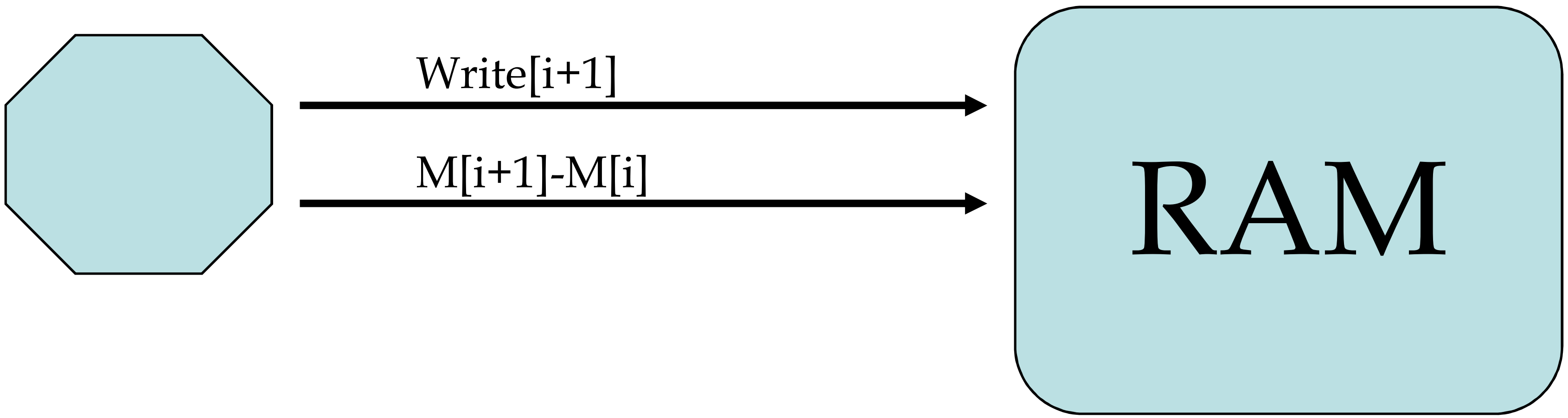
# Hardware Architecture

- We assume that we have a RAM initialized with the code.



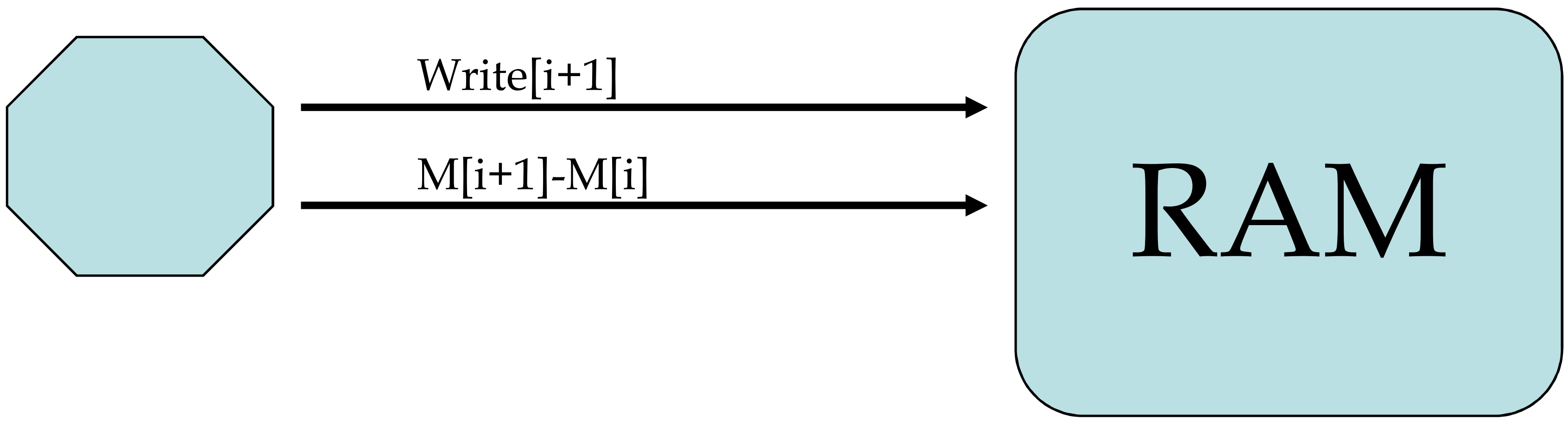
# Hardware Architecture

- We assume that we have a RAM initialized with the code.



# Hardware Architecture

- We assume that we have a RAM initialized with the code.



# What Have We Done?

Implemented the machine in FPGA (600 CLBs), wrote a compiler.

Circa 7 subleqs per 8-bit assembler instruction.

But the machine is so simple that clock can be very fast.

Explored variants:

SUBXORLEQ, SUBLEQXOR, SUBANDLEQ, etc.

Paper underway (soon on ePrint).