

Virtualization

Introduction

G. Lettieri

Dipartimento di Ingegneria dell'Informazione
Università di Pisa

A/A 2014/15

What do people mean when they talk about “virtualization” w.r.t. computers?

What do people mean when they talk about “virtualization” w.r.t. computers?

- *Anything* you do on computers/Internet

What do people mean when they talk about “virtualization” w.r.t. computers?

- *Anything* you do on computers/Internet
- it cannot be touched \implies it is not real

What do people mean when they talk about “virtualization” w.r.t. computers?

- *Anything* you do on computers/Internet
- it cannot be touched \implies it is not real
- “fake” vs “real” experience

What do people mean when they talk about “virtualization” w.r.t. computers?

- *Anything* you do on computers/Internet
- it cannot be touched \implies it is not real
- “fake” vs “real” experience

A fake V behaves like some real T w.r.t. some observer O .

Typically (but not necessarily):

T

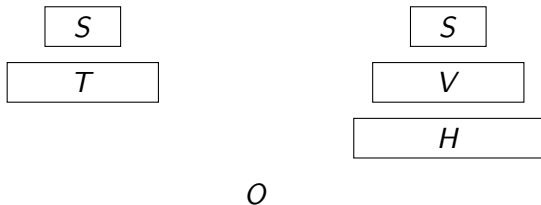
Typically (but not necessarily):

- real T (Target): some hardware



Typically (but not necessarily):

- real T (Target): some hardware
- Virtual V : made in software (running on hardware H , Host)



Typically (but not necessarily):

- real T (Target): some hardware
- Virtual V : made in software (running on hardware H , Host)
- Observer O : someone using software (S) originally made for T

Why virtualization?

- We don't want to change S

Why virtualization?

- We don't want to change S
- *and* one or more of:

Why virtualization?

- We don't want to change S
- *and* one or more of:
 - Hardware T is not available;

Why virtualization?

- We don't want to change S
- *and* one or more of:
 - Hardware T is not available;
 - V is *less expensive* than T

Why virtualization?

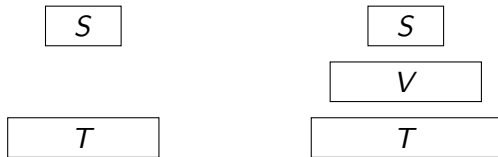
- We don't want to change S
- *and* one or more of:
 - Hardware T is not available;
 - V is *less expensive* than T
 - V is *more flexible* than T

Why virtualization?

- We don't want to change S
- *and* one or more of:
 - Hardware T is not available;
 - V is *less expensive* than T
 - V is *more flexible* than T
 - V offers a good protection model for S

Why virtualization? (2)

Useful also when $T = H$:



V adds a layer of indirection between S and T .

How to virtualize?

We are going to examine several techniques:

- emulation (Bochs, original JVM)

How to virtualize?

We are going to examine several techniques:

- emulation (Bochs, original JVM)
- binary translation (QEMU, recent JVMs)

How to virtualize?

We are going to examine several techniques:

- emulation (Bochs, original JVM)
- binary translation (QEMU, recent JVMs)
- hardware-assisted (KVM, Virtualbox)

How to virtualize?

We are going to examine several techniques:

- emulation (Bochs, original JVM)
- binary translation (QEMU, recent JVMs)
- hardware-assisted (KVM, Virtualbox)
- paravirtualization (original Xen)

The Small Scale Experimental Machine (1948)

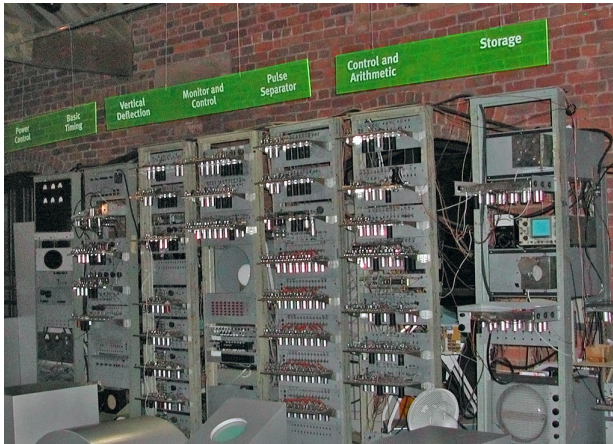


Figure : A modern replica of the SSEM, aka “Baby”, at the Museum of Science and Industry, Manchester. (credit: Wikipedia)

The SSEM CRT output

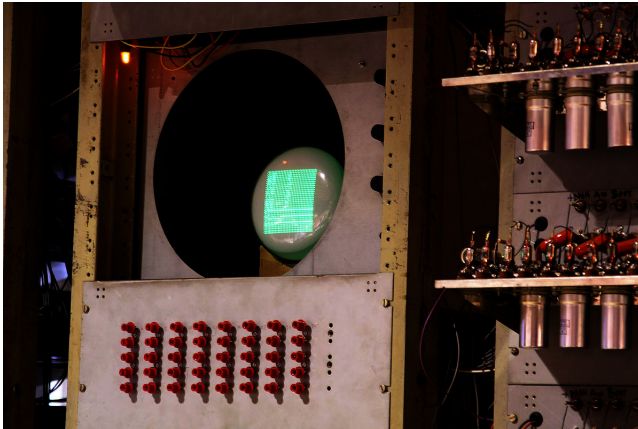
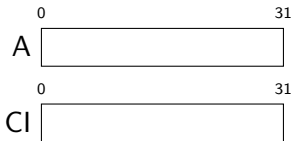
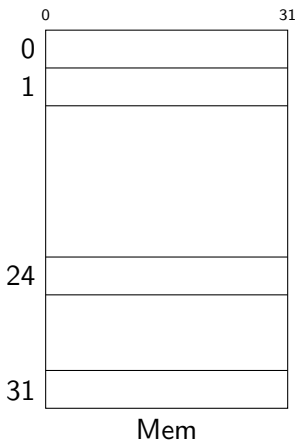
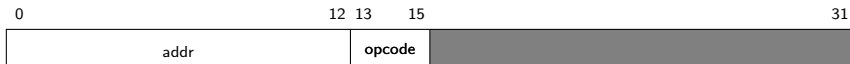


Figure : The CRT output showing the memory contents as a matrix of 32×32 big/small dots (credit: Wikipedia)

The SSEM ISA (1)



The SSEM ISA (2)



opcode		<i>effect</i>
0	CI	$\leftarrow \text{Mem}[\text{addr}]$
1	CI	$\leftarrow \text{CI} + \text{Mem}[\text{addr}]$
2	A	$\leftarrow -\text{Mem}[\text{addr}]$
3	Mem[addr]	$\leftarrow A$
4,5	A	$\leftarrow A - \text{Mem}[\text{addr}]$
6	<i>if</i> $A < 0$, $\text{CI} \leftarrow \text{CI} + 1$	
7	<i>halt</i>	

The emulator (1)

```
int32_t Mem[32];
int32_t A;
int32_t CI;

void exec() {
    for (;;) {
        /* advance CI */
        CI++;

        /* fetch the next instruction */
        int32_t PI = Mem[CI];

        /* decode the instruction */
        int32_t opcode = (PI & 0xE000) >> 13;
        int32_t addr = PI & 0x1FFF;
    }
}
```

The emulator (2)

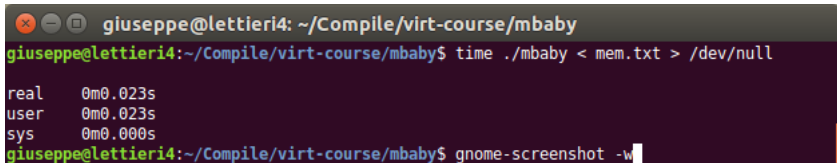
```
/* execute the instruction */  
switch (opcode) {  
case 0: CI = Mem[addr];           break;  
case 1: CI = CI + Mem[addr];      break;  
case 2: A = -Mem[addr];           break;  
case 3: Mem[addr] = A;           break;  
case 4: /* below */  
case 5: A = A - Mem[addr];        break;  
case 6: if (A < 0) CI = CI + 1;  break;  
case 7: return; /* terminates emulation */  
}  
}  
}
```

The (amended) first program

1	00011000000000100000000000000000	$A \leftarrow -\text{Mem}[24]$
2	01011000000001100000000000000000	$\text{Mem}[26] \leftarrow A$
3	01011000000001000000000000000000	$A \leftarrow -\text{Mem}[26]$
4	11011000000001100000000000000000	$\text{Mem}[27] \leftarrow A$
5	11101000000001000000000000000000	$A \leftarrow -\text{Mem}[23]$
6	11011000000000100000000000000000	$A \leftarrow A - \text{Mem}[27]$
7	00000000000001100000000000000000	<i>if</i> $A < 0$, $\text{CI} \leftarrow \text{CI} + 1$
8	00101000000001000000000000000000	$\text{CI} \leftarrow \text{CI} + \text{Mem}[20]$
9	01011000000000100000000000000000	$A \leftarrow A - \text{Mem}[26]$
10	10011000000001100000000000000000	$\text{Mem}[25] \leftarrow A$
11	10011000000001000000000000000000	$A \leftarrow -\text{Mem}[25]$
12	00000000000001100000000000000000	<i>if</i> $A < 0$, $\text{CI} \leftarrow \text{CI} + 1$
13	00000000000001110000000000000000	<i>halt</i>
14	01011000000001000000000000000000	$A \leftarrow -\text{Mem}[26]$
15	10101000000000100000000000000000	$A \leftarrow A - \text{Mem}[21]$
16	11011000000001100000000000000000	$\text{Mem}[27] \leftarrow A$
17	11011000000001000000000000000000	$A \leftarrow -\text{Mem}[27]$
18	01011000000001100000000000000000	$\text{Mem}[26] \leftarrow A$
19	01101000000000000000000000000000	$\text{CI} \leftarrow \text{Mem}[22]$
<hr/>		
20	10111111111111111111111111111111	-3
21	10000000000000000000000000000000	1
22	00100000000000000000000000000000	4
23	00000000000000000001111111111111	-262144
24	11111111111111111100000000000000	262143

Running the first program

... about 130,000 numbers were tested, involving some 3.5 million operations. The correct answer was obtained in a 52-minute run. (F.C. Williams, T. Kilburn, "Electronic Digital Computers", Nature, Vol. 162, p. 487, September 25, 1948.)

A terminal window with a dark background and light text. The window title is "giuseppe@lettieri4: ~/Compile/virt-course/mbaby". The prompt is "giuseppe@lettieri4:~/Compile/virt-course/mbaby\$". The command entered is "time ./mbaby < mem.txt > /dev/null". The output shows timing statistics: "real 0m0.023s", "user 0m0.023s", and "sys 0m0.000s". The prompt is now "giuseppe@lettieri4:~/Compile/virt-course/mbaby\$". The next command entered is "gnome-screenshot -w".

```
giuseppe@lettieri4: ~/Compile/virt-course/mbaby
giuseppe@lettieri4:~/Compile/virt-course/mbaby$ time ./mbaby < mem.txt > /dev/null
real    0m0.023s
user    0m0.023s
sys     0m0.000s
giuseppe@lettieri4:~/Compile/virt-course/mbaby$ gnome-screenshot -w
```

- Model both $T + S$ and $V + S$ as *State Machines*:

$\langle T\text{-state}, T\text{-next} \rangle$

$\langle V\text{-state}, V\text{-next} \rangle$

- Model both $T + S$ and $V + S$ as *State Machines*:

$\langle T\text{-state}, T\text{-next} \rangle$

$\langle V\text{-state}, V\text{-next} \rangle$

- Define *interp*: $V\text{-state} \rightarrow T\text{-state}$ (interpretation)

- Model both $T + S$ and $V + S$ as *State Machines*:

$\langle T\text{-state}, T\text{-next} \rangle$

$\langle V\text{-state}, V\text{-next} \rangle$

- Define *interp*: $V\text{-state} \rightarrow T\text{-state}$ (interpretation)
- Agree with O that she will only look at $T\text{-states}$ (either directly from T or from V through *interp*)

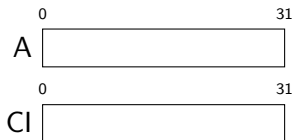
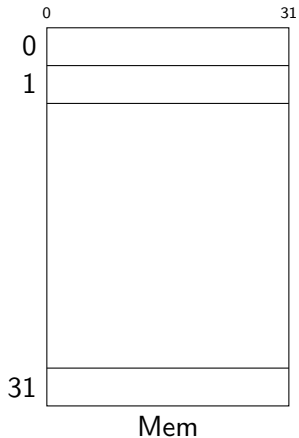
- Model both $T + S$ and $V + S$ as *State Machines*:

$\langle T\text{-state}, T\text{-next} \rangle$

$\langle V\text{-state}, V\text{-next} \rangle$

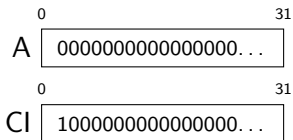
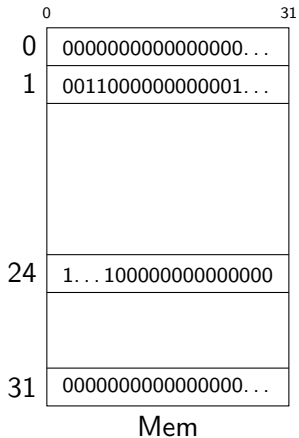
- Define *interp*: $V\text{-state} \rightarrow T\text{-state}$ (interpretation)
- Agree with O that she will only look at $T\text{-states}$ (either directly from T or from V through *interp*)
- Require that $V\text{-next}$ preserves the interpretation.

A formalization: T -state



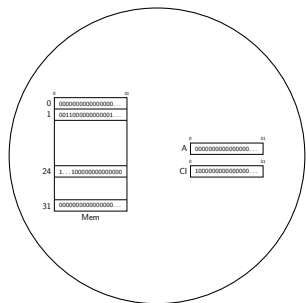
T

A formalization: T -state



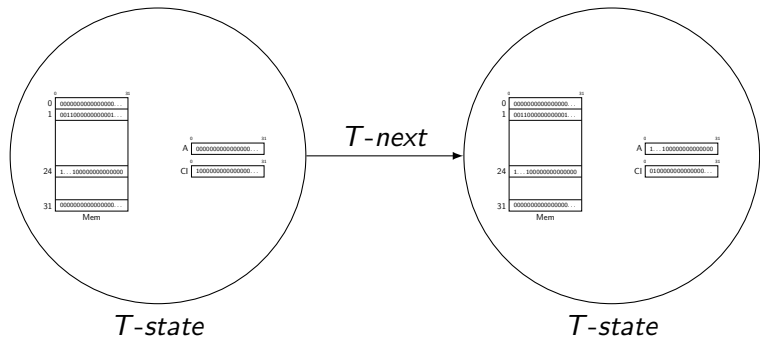
$T + S$

A formalization: T -transitions



T -state

A formalization: T -transitions



A formalization: *V-states*

```
int32_t Mem[32]; // 0, 0x100C, ...,  
              // [24] 0x3FFFF, ..., 0  
int32_t A;      // 0  
int32_t CI;     // 1  
  
void exec() {  
    for (;;) {  
        CI++;  
⇒      int32_t PI = Mem[CI];  
  
        int32_t opcode = (PI & 0xE000) >> 13;  
        int32_t addr = PI & 0x1FFF;  
  
        switch (opcode) {  
        case 0: CI = Mem[addr];          break;  
        case 1: CI = CI + Mem[addr];     break;  
        case 2: A = -Mem[addr];          break;  
        case 3: Mem[addr] = A;           break;  
        case 4:  
        case 5: A = A - Mem[addr];        break;  
        case 6: if (A < 0) CI = CI + 1;  break;  
        case 7: return;  
        }  
    }  
}
```

A formalization: *V*-state interpretation

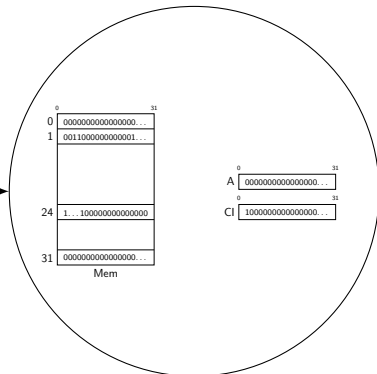
```
int32_t Mem[32]; // 0, 0x100C, ..., [24] 0x3FFFF, ..., 0
int32_t A; // 0
int32_t C1; // 1

void exec() {
  for (;;) {
    => int32_t P1 = Mem[C1];

    int32_t opcode = (P1 & 0xE000) >> 13;
    int32_t addr = P1 & 0x3FFF;

    switch (opcode) {
      case 0: C1 = Mem[addr]; break;
      case 1: C1 = C1 + Mem[addr]; break;
      case 2: A = -Mem[addr]; break;
      case 3: Mem[addr] = A; break;
      case 4:
      case 5: A = A - Mem[addr]; break;
      case 6: if (A < 0) C1 = C1 + 1; break;
      case 7: return;
    }
  }
}
```

interp



A formalization: V-transitions

```
int32_t Mem[32]; // 0, 0x1000, ...,
                // [24] 0xFFFF, ..., 0
int32_t A;      // 0
int32_t C1;     // 1

void exec() {
  for (;;) {
    C1++;
    ⇒ int32_t P1 = Mem[C1];

       int32_t opcode = (P1 & 0xE000) >> 13;
       int32_t addr = P1 & 0x1FFF;

       switch (opcode) {
         case 0: C1 = Mem[addr];      break;
         case 1: C1 = C1 + Mem[addr]; break;
         case 2: A = -Mem[addr];     break;
         case 3: Mem[addr] = A;      break;
         case 4:                      break;
         case 5: A = A - Mem[addr];   break;
         case 6: if (A < 0) C1 = C1 + 1; break;
         case 7: return;
       }
    }
}
```


A formalization: V-transitions

```
int32_t Mem[32]; // 0, 0x100C, ...,
int32_t A; // [24] 0x3FFF, ..., 0
int32_t C1; // 1

void exec() {
  for (;;) {
    C1++;
    => int32_t P1 = Mem[C1];
    int32_t opcode = (P1 & 0xE000) >> 13;
    int32_t addr = P1 & 0x1FFF;

    switch (opcode) {
    case 0: C1 = Mem[addr]; break;
    case 1: C1 = C1 + Mem[addr]; break;
    case 2: A = -Mem[addr]; break;
    case 3: Mem[addr] = A; break;
    case 4:
    case 5: A = A - Mem[addr]; break;
    case 6: if (A < 0) C1 = C1 + 1; break;
    case 7: return;
    }
  }
}
```

```
int32_t Mem[32]; // 0, 0x100C, ...,
int32_t A; // [24] 0x3FFF, ..., 0
int32_t C1; // 1

void exec() {
  for (;;) {
    C1++;
    => int32_t P1 = Mem[C1];
    int32_t opcode = (P1 & 0xE000) >> 13;
    int32_t addr = P1 & 0x1FFF;

    switch (opcode) {
    case 0: C1 = Mem[addr]; break;
    case 1: C1 = C1 + Mem[addr]; break;
    case 2: A = -Mem[addr]; break;
    case 3: Mem[addr] = A; break;
    case 4:
    case 5: A = A - Mem[addr]; break;
    case 6: if (A < 0) C1 = C1 + 1; break;
    case 7: return;
    }
  }
}
```

A formalization: V-transitions

```
int32_t Mem[32]; // 0..0x100C, ...,
int32_t A; // [24] 0x3FFF, ..., 0
int32_t Cl; // 1

void exec() {
  for (;;) {
    Cl++;
    =>
    int32_t Pi = Mem[Cl];
    int32_t opcode = (Pi & 0xE000) >> 13;
    int32_t addr = Pi & 0x1FFF;

    switch (opcode) {
    case 0: Cl = Mem[addr]; break;
    case 1: Cl = Cl + Mem[addr]; break;
    case 2: A = -Mem[addr]; break;
    case 3: Mem[addr] = A; break;
    case 4:
    case 5: A = A - Mem[addr]; break;
    case 6: if (A < 0) Cl = Cl + 1; break;
    case 7: return;
    }
  }
}
```

```
int32_t Mem[32]; // 0..0x100C, ...,
int32_t A; // [24] 0x3FFF, ..., 0
int32_t Cl; // 1

void exec() {
  for (;;) {
    Cl++;
    int32_t Pi = Mem[Cl];
    =>
    int32_t opcode = (Pi & 0xE000) >> 13;
    int32_t addr = Pi & 0x1FFF;

    switch (opcode) {
    case 0: Cl = Mem[addr]; break;
    case 1: Cl = Cl + Mem[addr]; break;
    case 2: A = -Mem[addr]; break;
    case 3: Mem[addr] = A; break;
    case 4:
    case 5: A = A - Mem[addr]; break;
    case 6: if (A < 0) Cl = Cl + 1; break;
    case 7: return;
    }
  }
}
```

```
int32_t Mem[32]; // 0..0x100C, ...,
int32_t A; // [24] 0x3FFF, ..., 0
int32_t Cl; // 2

void exec() {
  for (;;) {
    Cl++;
    =>
    int32_t Pi = Mem[Cl];
    int32_t opcode = (Pi & 0xE000) >> 13;
    int32_t addr = Pi & 0x1FFF;

    switch (opcode) {
    case 0: Cl = Mem[addr]; break;
    case 1: Cl = Cl + Mem[addr]; break;
    case 2: A = -Mem[addr]; break;
    case 3: Mem[addr] = A; break;
    case 4:
    case 5: A = A - Mem[addr]; break;
    case 6: if (A < 0) Cl = Cl + 1; break;
    case 7: return;
    }
  }
}
```

A formalization: V-transitions

V-next

```
int32_t Mem[32]; // 0, 0x100C, ...,
                // [24] 0x3FFF, ..., 0
int32_t A; // 0
int32_t Cl; // 1

void exec() {
  for (;;) {
    Cl++;
    => int32_t Pi = Mem[Cl];
       int32_t opcode = (Pi & 0xE000) >> 13;
       int32_t addr = Pi & 0x1FFF;

    switch (opcode) {
    case 0: Cl = Mem[addr]; break;
    case 1: Cl = Cl + Mem[addr]; break;
    case 2: A = -Mem[addr]; break;
    case 3: Mem[addr] = A; break;
    case 4:
    case 5: A = A - Mem[addr]; break;
    case 6: if (A < 0) Cl = Cl + 1; break;
    case 7: return;
    }
  }
}
```

V-state

```
int32_t Mem[32]; // 0, 0x100C, ...,
                // [24] 0x3FFF, ..., 0
int32_t A; // 0
int32_t Cl; // 1

void exec() {
  for (;;) {
    Cl++;
    => int32_t Pi = Mem[Cl];
       int32_t opcode = (Pi & 0xE000) >> 13;
       int32_t addr = Pi & 0x1FFF;

    switch (opcode) {
    case 0: Cl = Mem[addr]; break;
    case 1: Cl = Cl + Mem[addr]; break;
    case 2: A = -Mem[addr]; break;
    case 3: Mem[addr] = A; break;
    case 4:
    case 5: A = A - Mem[addr]; break;
    case 6: if (A < 0) Cl = Cl + 1; break;
    case 7: return;
    }
  }
}
```

...

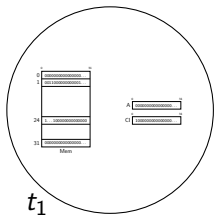
```
int32_t Mem[32]; // 0, 0x100C, ...,
                // [24] 0x3FFF, ..., 0
int32_t A; // 0
int32_t Cl; // 2

void exec() {
  for (;;) {
    Cl++;
    => int32_t Pi = Mem[Cl];
       int32_t opcode = (Pi & 0xE000) >> 13;
       int32_t addr = Pi & 0x1FFF;

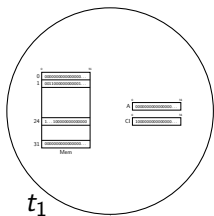
    switch (opcode) {
    case 0: Cl = Mem[addr]; break;
    case 1: Cl = Cl + Mem[addr]; break;
    case 2: A = -Mem[addr]; break;
    case 3: Mem[addr] = A; break;
    case 4:
    case 5: A = A - Mem[addr]; break;
    case 6: if (A < 0) Cl = Cl + 1; break;
    case 7: return;
    }
  }
}
```

V-state

A formalization: stepwise correctness



A formalization: stepwise correctness



t_1

interp

```
int32_t Mem[32] // 0x00000000, ..., 0x00000000
int32_t A; // 0x00000000, ..., 0
int32_t C1; // 0

void main() {
  for (i = 0; i < 32; i++) {
    Mem[i] = 0;
  }

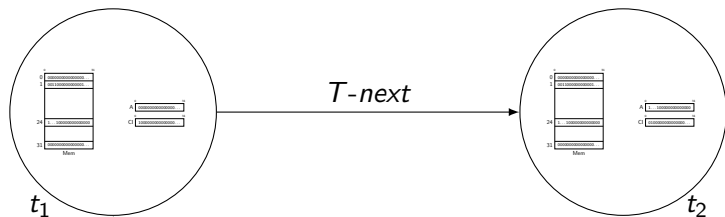
  int32_t P1 = Mem[C1];

  int32_t opcode = (P1 & 0xF000) >> 12;
  int32_t addr = P1 & 0x1FFF;

  switch (opcode) {
    case 0: C1 = Mem[addr]; break;
    case 1: C1 = C1 + Mem[addr]; break;
    case 2: A = Mem[addr]; break;
    case 3: Mem[addr] = A; break;
    case 4: break;
    case 5: A = A - Mem[addr]; break;
    case 6: if (A < 0) C1 = C1 + 1; break;
    case 7: return;
  }
}
```

V_1

A formalization: stepwise correctness



interp

```
int32_t Mem[32] // 0x00000000, ..., 0x00000000
int32_t A; // 0x00000000, ..., 0
int32_t C1; // 0

void main() {
  for (;;) {
    C1++;
    //
    int32_t P1 = Mem[C1];

    if (P1 > 0) {
      int32_t opcode = (P1 & 0xF000) >> 12;
      int32_t addr = P1 & 0xFFFF;

      switch (opcode) {
        case 0: C1 = Mem[addr]; break;
        case 1: C1 = C1 - Mem[addr]; break;
        case 2: A = Mem[addr]; break;
        case 3: Mem[addr] = A; break;
        case 4: A = A - Mem[addr]; break;
        case 5: if (A < 0) C1 = C1 + 1; break;
        case 7: return;
      }
    }
  }
}
```

V_1

A formalization: stepwise correctness

