Virtualization

Introduction

G. Lettieri

Dipartimento di Ingegneria dell'Informazione
Università di Pisa

A/A 2014/15

Folk definition

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

Folk definition

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

- *Anything* you do on computers/Internet

Folk definition

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

Folk definition

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

- *Anything* you do on computers/Internet
- it cannot be touched $\Longrightarrow$ it is not real
- "fake" vs "real" experience

Folk definition

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

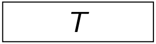
- *Anything* you do on computers/Internet
- it cannot be touched $\implies$ it is not real
- "fake" vs "real" experience

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

Computer Science definition

Typically (but not necessarily):

Computer Science definition



Typically (but not necessarily):

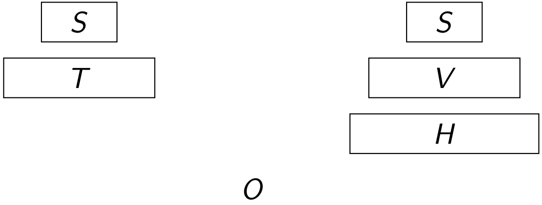
- real T (Target): some hardware

Computer Science definition

Typically (but not necessarily):

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

Computer Science definition



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- Observer O: someone using software (S) originally made for T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?

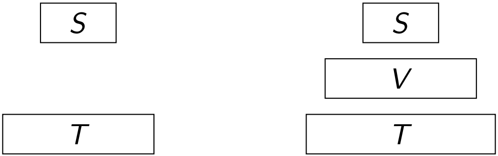
- We don't want to change S
- *and* one or more of:
  - Hardware T is not available;
  - V is *less expensive* than T
  - V is *more flexible* than T

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)

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)

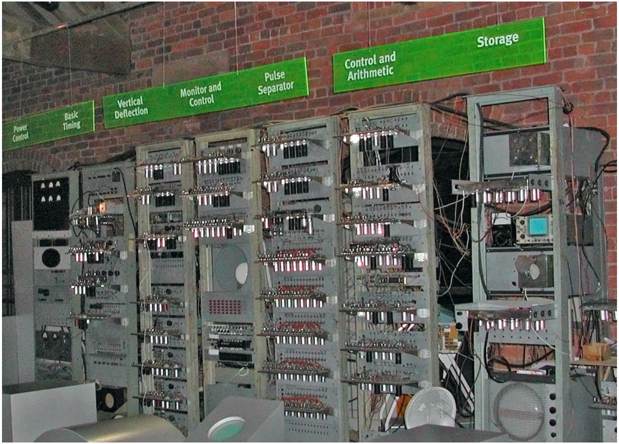


Figure: A modern replica of the SSEM, aka “Baby”, at the Museum of Science and Industry, Manchester. (credit: Wikipedia)

The SSEM CRT output

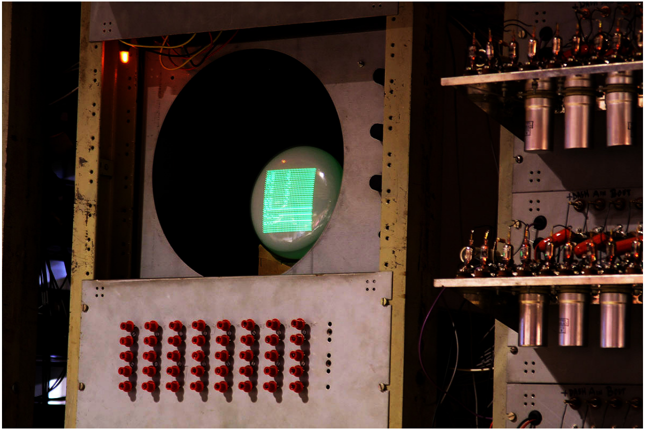
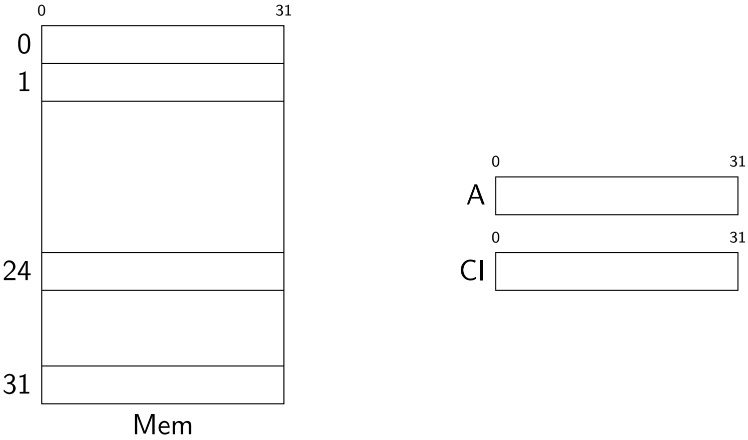
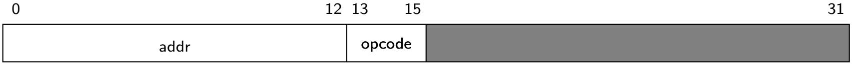


Figure: The CRT output showing the memory contents as a matrix of 32×32 big/small dots (credit: Wikipedia)

The SSEM ISA (1)



The SSEM ISA (2)



opcode		effect
0	CI	$\leftarrow$ Mem[addr]
1	CI	$\leftarrow$ CI + Mem[addr]
2	A	$\leftarrow$ −Mem[addr]
3	Mem[addr]	$\leftarrow$ A
4,5	A	$\leftarrow$ A − Mem[addr]
6	*if* $A < 0$, CI $\leftarrow$ CI + 1	
7	*halt*	

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	$A \leftarrow -\mathrm{Mem}[24]$
2	01011000000001100000000000000000	$\mathrm{Mem}[26] \leftarrow A$
3	01011000000000100000000000000000	$A \leftarrow -\mathrm{Mem}[26]$
4	11011000000001100000000000000000	$\mathrm{Mem}[27] \leftarrow A$
5	11101000000000100000000000000000	$A \leftarrow -\mathrm{Mem}[23]$
6	11011000000000010000000000000000	$A \leftarrow A - \mathrm{Mem}[27]$
7	00000000000000110000000000000000	*if* $A < 0,\ \mathrm{CI} \leftarrow \mathrm{CI} + 1$
8	00101000000001000000000000000000	$\mathrm{CI} \leftarrow \mathrm{CI} + \mathrm{Mem}[20]$
9	01011000000000010000000000000000	$A \leftarrow A - \mathrm{Mem}[26]$
10	10011000000001100000000000000000	$\mathrm{Mem}[25] \leftarrow A$
11	10011000000000100000000000000000	$A \leftarrow -\mathrm{Mem}[25]$
12	00000000000000110000000000000000	*if* $A < 0,\ \mathrm{CI} \leftarrow \mathrm{CI} + 1$
13	00000000000001110000000000000000	*halt*
14	01011000000000100000000000000000	$A \leftarrow -\mathrm{Mem}[26]$
15	10101000000000010000000000000000	$A \leftarrow A - \mathrm{Mem}[21]$
16	11011000000001100000000000000000	$\mathrm{Mem}[27] \leftarrow A$
17	11011000000000100000000000000000	$A \leftarrow -\mathrm{Mem}[27]$
18	01011000000001100000000000000000	$\mathrm{Mem}[26] \leftarrow A$
19	01101000000000000000000000000000	$\mathrm{CI} \leftarrow \mathrm{Mem}[22]$
20	10111111111111111111111111111111	-3
21	10000000000000000000000000000000	1
22	00100000000000000000000000000000	4
23	00000000000000000011111111111111	-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.)

```
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
```

A formalization

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

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

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

A formalization

- 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*: $V\text{-}state \to T\text{-}state$ (interpretation)

A formalization

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

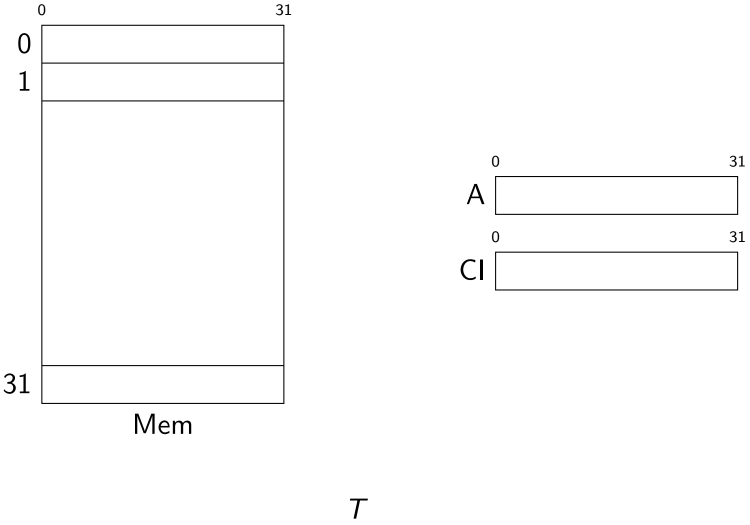
A formalization

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

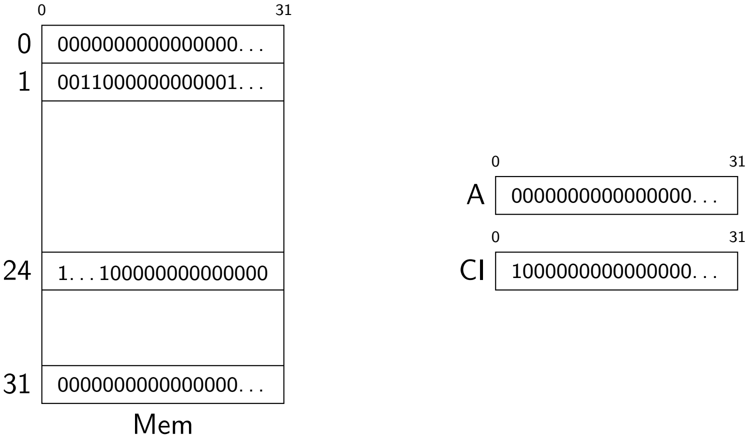
$$\langle T\text{-}state, T\text{-}next \rangle$$
$$\langle V\text{-}state, V\text{-}next \rangle$$

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

A formalization: *T-state*

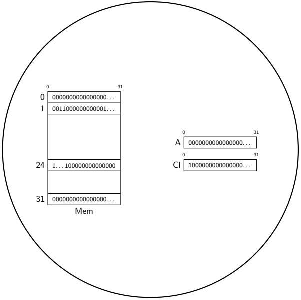


A formalization: *T-state*



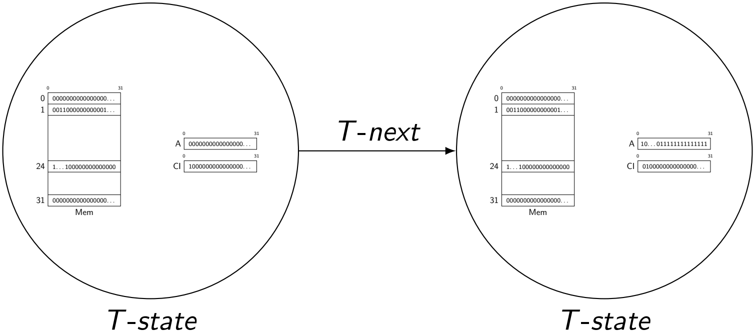
$$T + S$$

A formalization: T-transitions



T-state

A formalization: T-transitions

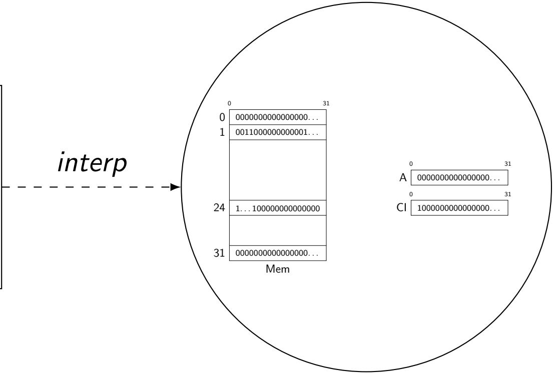


A formalization: *V-states*

```
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 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-transitions

```
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;
    }
  }
}
```

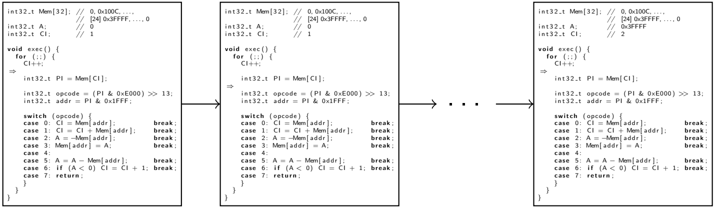
A formalization: V-transitions

```
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;
    }
  }
}
```

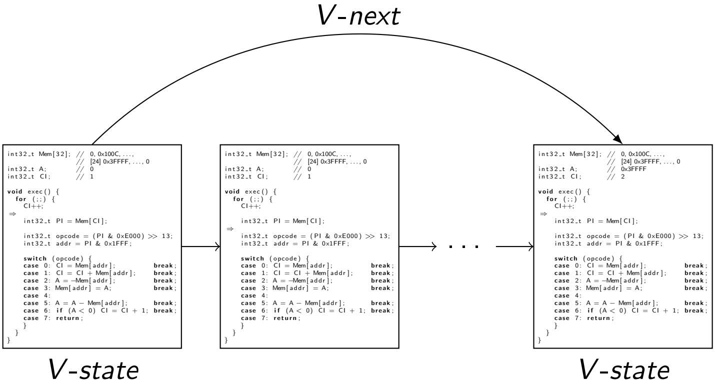
→

```
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;
    }
  }
}
```

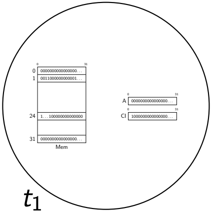
A formalization: V-transitions



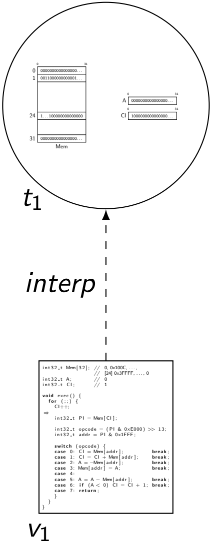
A formalization: *V*-transitions



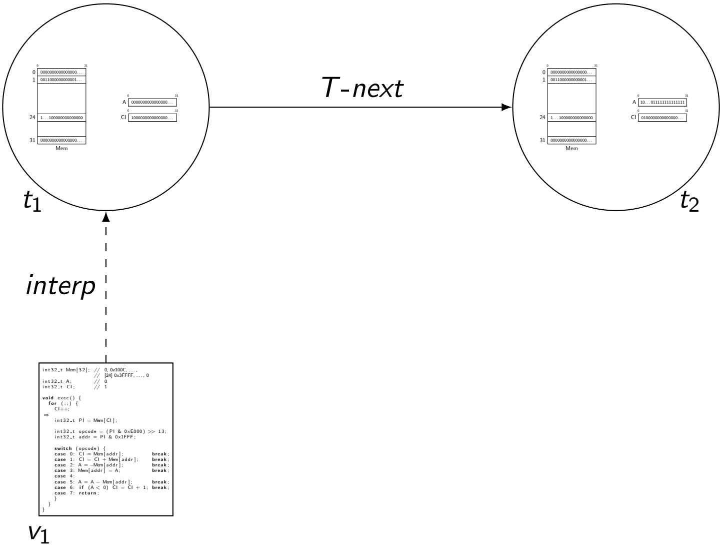
A formalization: stepwise correctness



A formalization: stepwise correctness



A formalization: stepwise correctness



A formalization: stepwise correctness