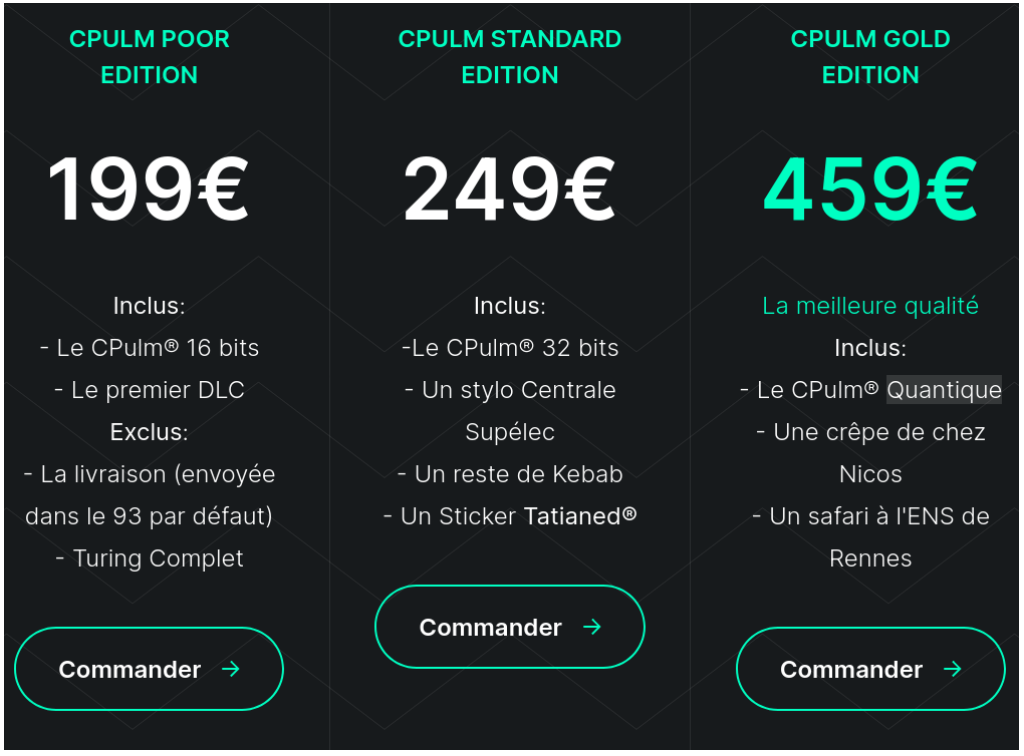
CPUlm

Le processeur ~~*qui affiche l'heure*~~ de **demain**

Hubert Gruniaux, Gabriel Desfrene,
Clément Rouvroy & Vincent Jules

23 Janvier 2024

L'Offre *CPUlm*



- Confidentiel
- Photos interdites

ISA : Une histoire de 32

- Architecture 32-bits,
- Adressage mémoire sur 32-bits,
- 32 registres généraux, mais
  - `r0 = 0`
  - `r1 = 1`
- Instructions à 3 opérandes

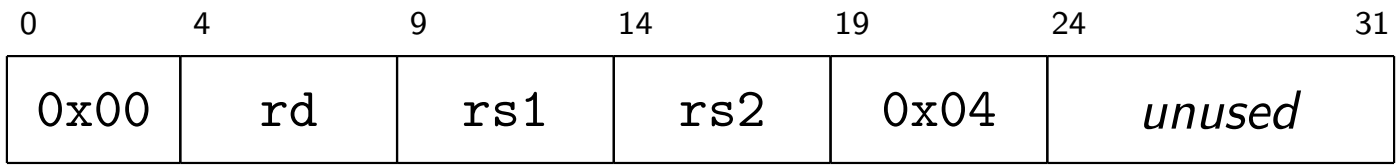


Figure – Encodage de `ADD`

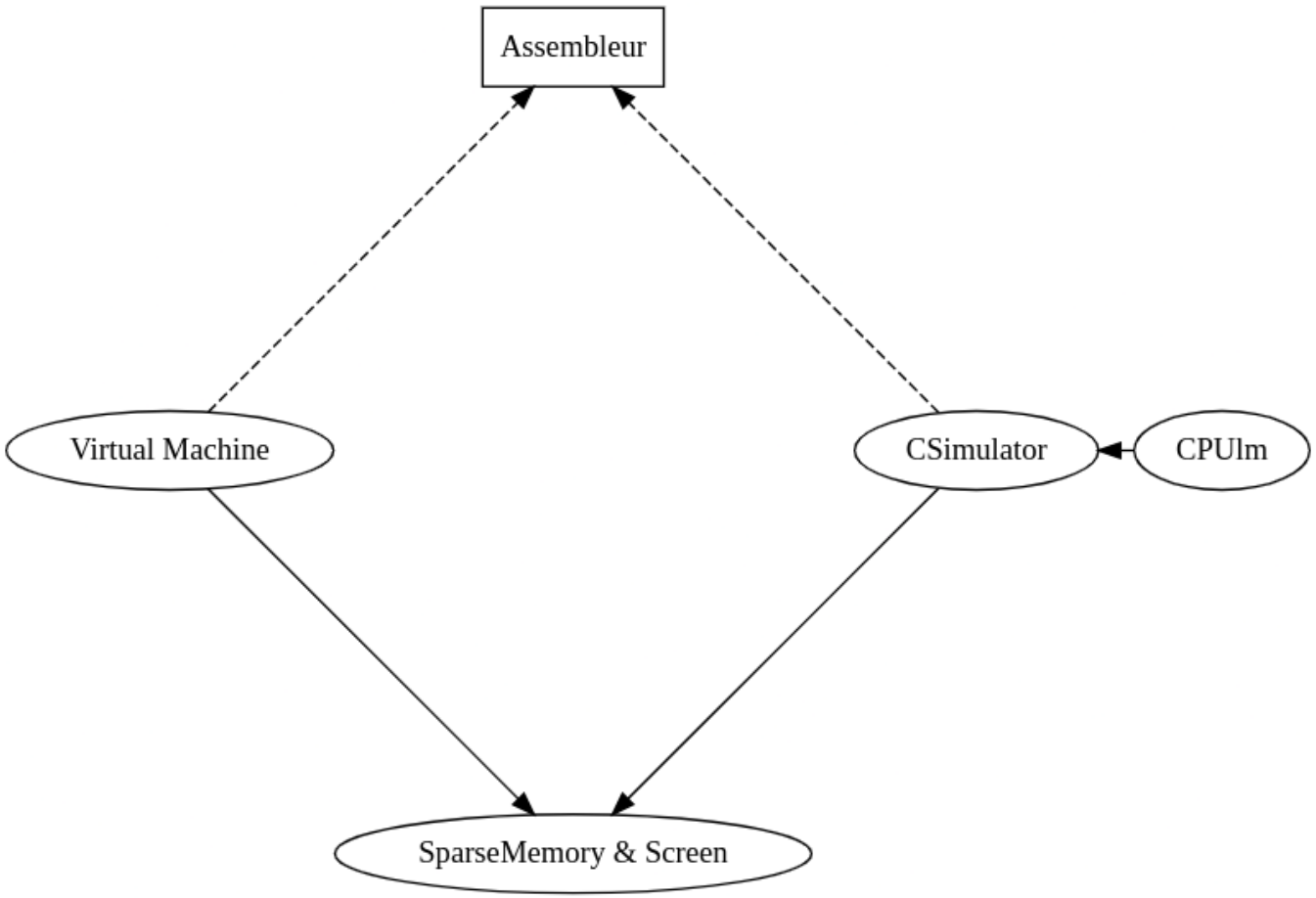
ISA : Instructions

Il y a 18 instructions supportées par le CPUlm :

- `AND`
- `OR`
- `NOR`
- `XOR`
- `ADD`
- `SUB`
- `MUL`
- `DIV`
- `LSL`
- `ASR`
- `LSR`
- `LOAD`
- `LOADI`
- `STORE`
- `JMP`
- `JMPC`
- `JMPI`
- `JMPIC`

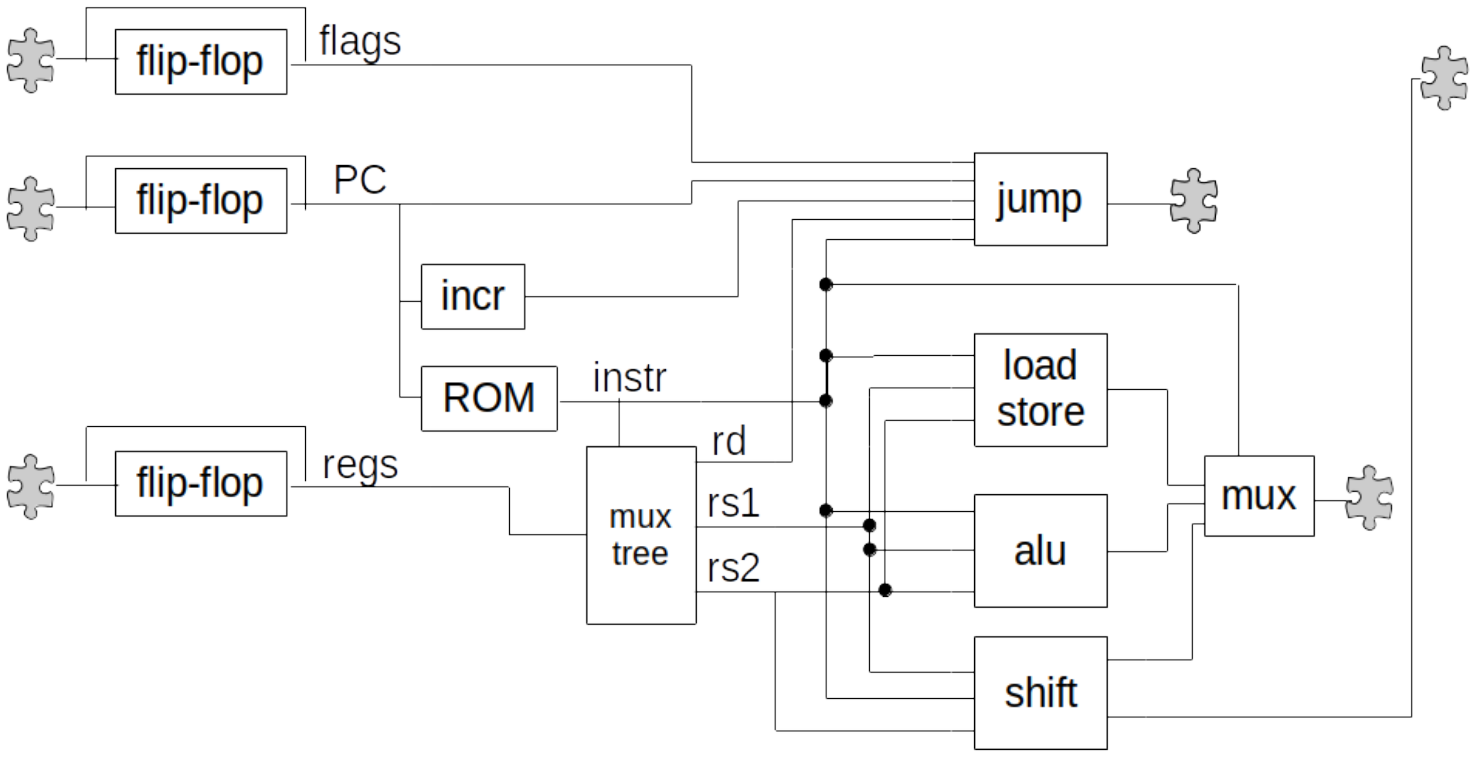
Elles sont encodées sur 32-bits.

L'Écosystème *CPUlm*



Hardware

Le *CPUlm*


flip-flop
flags
flip-flop
PC
incr
ROM
instr
flip-flop
regs
rd
mux
tree
rs1
rs2
jump
load
store
alu
shift
mux

Le *CPUlm*

4 blocs :

- *jump* $(\texttt{instruction}, \texttt{flags}, \texttt{pc}, \texttt{pc_incr}, \texttt{rd}) \mapsto \texttt{pc}$
- *load-store* $(\texttt{instruction}, \texttt{rd}, \texttt{rs1}) \mapsto \texttt{rd}$
- *alu* $(\texttt{instruction}, \texttt{rs1}, \texttt{rs2}) \mapsto \texttt{rd}, \texttt{flags}$
- *shift* $(\texttt{instruction}, \texttt{rs1}, \texttt{rs2}) \mapsto \texttt{rd}$

Le *CPUlm* : l'*ALU*

En entrée : `instruction`, `rs1`, `rs2`

- `and`, `or`, `nor`, `xor`
- `add`, `sub`
- `mul`
- `div`
- `flags`

Le *CPUlm* : l'*ALU*

En entrée : `instruction`, `rs1`, `rs2`

- `and`, `or`, `nor`, `xor`
- `add`, `sub`
- `mul` ($\simeq$ 4500 portes logiques : 40%)
- `div` ($\simeq$ 4500 portes logiques : 40%)
- `flags`

Le *CPUlm* sans la multiplication ni la division : 2000 portes logiques

Un Compilateur *NetList* vers *C*

- Évaluation paresseuse de la *NetList*. Sont toujours calculés :
  - `REG x`
  - sorties
  - `ROM as ws rd we wa wd`

Un Compilateur *NetList* vers *C*

- Évaluation paresseuse de la *NetList*. Sont toujours calculés :
  - `REG x`
  - sorties
  - `ROM as ws rd we wa wd`
- Découpage de la *NetList* en fonctions avec plusieurs instructions
- Interface avec l'utilisateur

Software

L'Assembleur *CPUlm*

```
    .text
; Ce programme incrémente r3
; jusqu'à ce que r3 = max
main:
    loadi   r2  $max
    load    r2  r2
    loadi   r3  0
loop:
    sub     r4  r2  r3
    jmp.z   $end
    inc     r3  r3
    jmp     $loop
end:
    halt
    .data
max: .int 5
```

```
0x00    LOADI sp 0x1001 r0 1
    main:
0x01    LOADI r2 0x1000 r0 1
0x02    LOAD r2 r2
0x03    LOADI r3 0x0000 r0 1
    loop:
0x04    SUB r4 r2 r3
0x05    JMPIC +3 Z
0x06    ADD r3 r3 r1
0x07    JMPI -3
    end:
0x08    LOADI rpriv 0xffff r0 1
0x09    LOADI rpriv 0xffff rpriv 0
0x0a    JMP rpriv
```

Encodage des chaînes de caractères

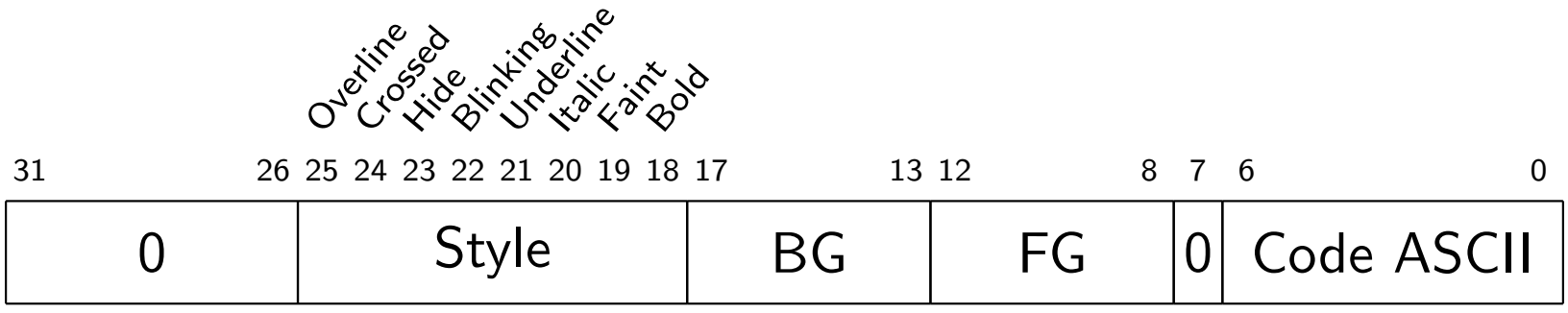


Figure – Encodage d'un caractère dans l'assembleur *CPUlm*.

La Machine Virtuelle (VM)

- Tester et deboguer un programme sur le CPU :

  $\Longrightarrow$ C'est **fastidieux**.
- Solution : la **Machine Virtuelle**
  - Execution étape par étape,
  - *Breakpoints*,
  - Lecture et écriture des registres,
  - Un désassembleur,
  - Etc.

La technologie SparseMemory [1]

- Problème : Impossible d'allouer 16Go de mémoire...
- Solution : Allouer de façon paresseuse la mémoire.
  - Page mémoire de 1024 mots $\rightarrow$ 4Kio
  - Table de hachage pour le mapping
- Supporte la technologie $\mathbb{SCREEN}$ (version 64×16)
  - API de détection des écritures en RAM

1. Technologie gratuite

La technologie SCREEN [2]

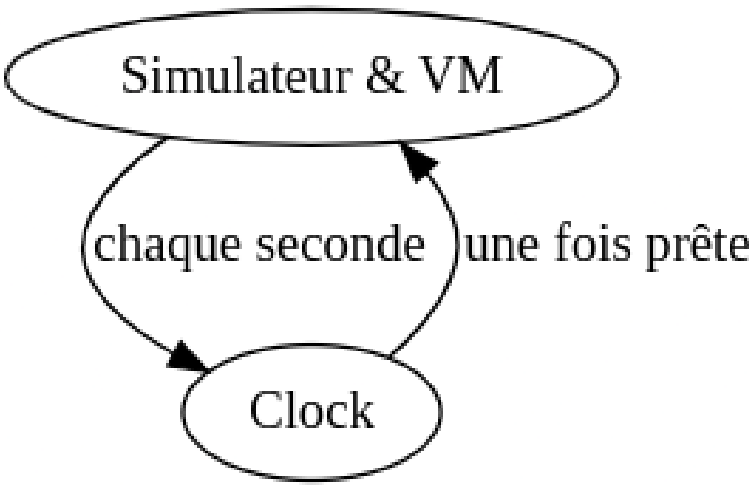
- Écran ASCII, résolution 64×16 caractères (1024 mots)
  - Correspond exactement à une page mémoire.
- Pleins de couleurs et de styles différents
- Fonctionnement efficace
  - Mapping d'une région de la RAM
  - Utilisation des séquences VT-100 (mise à jour intelligente)

2. Technologie gratuite

L'Horloge *CPUlm*

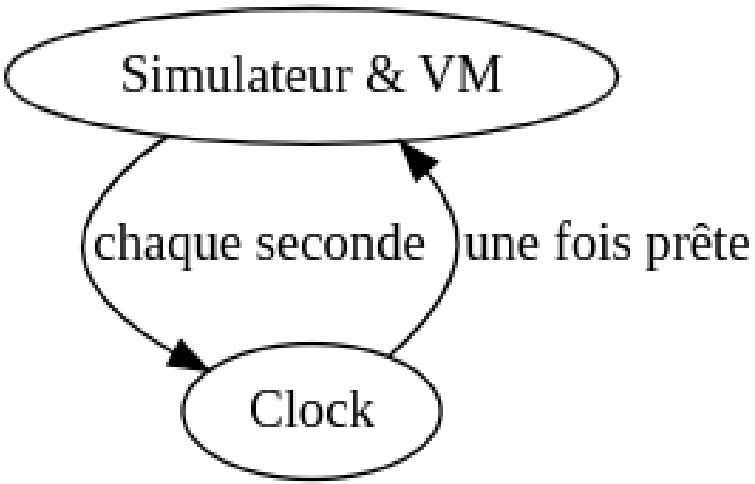
Fonctionnement

- Comment la clock est à l'heure



Fonctionnement

- Comment la clock est à l'heure



- Mettre à jour le temps dans les registres
- Mettre à jour le temps sur l'écran

Roadmap

- Janvier : *CPUlm Now*
- Mars : *Premium++*

Roadmap

- Janvier : *CPUlm Now*
- Mars : *Premium++*
- Juillet : *CPUlm CEnkorPT*
- Noël : *CPUlm Kids*

Conclusion

Achetez sur `cpulm.cr-dev.io` !