Projet:
un microprocesseur

Cédric Pasteur
cedric.pasteur@ens.fr
http://di.ens.fr/~pasteur

25 sept. 2012

Aperçu

But: créer un processeur pour exécuter le programme d'une montre digitale

- Simulateur de net-list
- Microprocesseur et son jeu d'instructions
- Programme de la montre

Moyen: à vous de voir

Un simulateur de net-list

Langage de description de net-list

- Portes logiques, registres et mémoires
- Très simple (pas de hiérarchie, etc)

Simulateur

- Entrée: circuit, entrées et nombre de pas
- Sortie: sorties du circuit
- (Suffisament) Efficace et robuste

Exemple: le full-adder

```
INPUT a, b
OUTPUT o, c
VAR
  _l_10_50, _l_11_49, _l_16_22,
  _l_17_21, _l_7_52, _l_9_51,
  a, b, c, c_n1_27, o, s_n_26
IN
o = s_n_26
c = OR _l_9_51 _l_11_49
s_n_26 = XOR _l_7_52 c_n1_27
_l_7_52 = XOR _l_16_22 _l_17_21
_l_9_51 = AND _l_16_22 _l_17_21
_l_10_50 = XOR _l_16_22 _l_17_21
_l_11_49 = AND _l_10_50 c_n1_27
c_n1_27 = 0
_l_16_22 = SELECT 0 a
_l_17_21 = SELECT 0 b
```

```
Step 1:
a ?
```

Exemple: le full-adder

```
INPUT a, b
OUTPUT o, c
VAR
  _l_10_50, _l_11_49, _l_16_22,
  _l_17_21, _l_7_52, _l_9_51,
  a, b, c, c_n1_27, o, s_n_26
IN
o = s_n_26
c = OR _l_9_51 _l_11_49
s_n_26 = XOR _l_7_52 c_n1_27
_l_7_52 = XOR _l_16_22 _l_17_21
_l_9_51 = AND _l_16_22 _l_17_21
_l_10_50 = XOR _l_16_22 _l_17_21
_l_11_49 = AND _l_10_50 c_n1_27
c_n1_27 = 0
_l_16_22 = SELECT 0 a
_l_17_21 = SELECT 0 b
```

```
Step 1:
a ? 1
```

Exemple: le full-adder

```
INPUT a, b
OUTPUT o, c
VAR
  _l_10_50, _l_11_49, _l_16_22,
  _l_17_21, _l_7_52, _l_9_51,
  a, b, c, c_n1_27, o, s_n_26
IN
o = s_n_26
c = OR _l_9_51 _l_11_49
s_n_26 = XOR _l_7_52 c_n1_27
_l_7_52 = XOR _l_16_22 _l_17_21
_l_9_51 = AND _l_16_22 _l_17_21
_l_10_50 = XOR _l_16_22 _l_17_21
_l_11_49 = AND _l_10_50 c_n1_27
c_n1_27 = 0
_l_16_22 = SELECT 0 a
_l_17_21 = SELECT 0 b
```

```
Step 1:
a ? 1
b ?
```

Exemple: le full-adder

```
INPUT a, b
OUTPUT o, c
VAR
  _l_10_50, _l_11_49, _l_16_22,
  _l_17_21, _l_7_52, _l_9_51,
  a, b, c, c_n1_27, o, s_n_26
IN
o = s_n_26
c = OR _l_9_51 _l_11_49
s_n_26 = XOR _l_7_52 c_n1_27
_l_7_52 = XOR _l_16_22 _l_17_21
_l_9_51 = AND _l_16_22 _l_17_21
_l_10_50 = XOR _l_16_22 _l_17_21
_l_11_49 = AND _l_10_50 c_n1_27
c_n1_27 = 0
_l_16_22 = SELECT 0 a
_l_17_21 = SELECT 0 b
```

```
Step 1:
a ? 1
b ? 1
```

Exemple: le full-adder

```
INPUT a, b
OUTPUT o, c
VAR
  _l_10_50, _l_11_49, _l_16_22,
  _l_17_21, _l_7_52, _l_9_51,
  a, b, c, c_n1_27, o, s_n_26
IN
o = s_n_26
c = OR _l_9_51 _l_11_49
s_n_26 = XOR _l_7_52 c_n1_27
_l_7_52 = XOR _l_16_22 _l_17_21
_l_9_51 = AND _l_16_22 _l_17_21
_l_10_50 = XOR _l_16_22 _l_17_21
_l_11_49 = AND _l_10_50 c_n1_27
c_n1_27 = 0
_l_16_22 = SELECT 0 a
_l_17_21 = SELECT 0 b
```

Step 1:

a ? 1

b ? 1

=> o = 0

=> c = 1

Step 2:

Exemple: le full-adder

```
INPUT a, b
OUTPUT o, c
VAR
  _l_10_50, _l_11_49, _l_16_22,
  _l_17_21, _l_7_52, _l_9_51,
  a, b, c, c_n1_27, o, s_n_26
IN
o = s_n_26
c = OR _l_9_51 _l_11_49
s_n_26 = XOR _l_7_52 c_n1_27
_l_7_52 = XOR _l_16_22 _l_17_21
_l_9_51 = AND _l_16_22 _l_17_21
_l_10_50 = XOR _l_16_22 _l_17_21
_l_11_49 = AND _l_10_50 c_n1_27
c_n1_27 = 0
_l_16_22 = SELECT 0 a
_l_17_21 = SELECT 0 b
```

```
Step 1:
a ? 1
b ? 1
=> o = 0
=> c = 1
```

```
Step 2:
a ?
```

Exemple: le full-adder

```
INPUT a, b
OUTPUT o, c
VAR
  _l_10_50, _l_11_49, _l_16_22,
  _l_17_21, _l_7_52, _l_9_51,
  a, b, c, c_n1_27, o, s_n_26
IN
o = s_n_26
c = OR _l_9_51 _l_11_49
s_n_26 = XOR _l_7_52 c_n1_27
_l_7_52 = XOR _l_16_22 _l_17_21
_l_9_51 = AND _l_16_22 _l_17_21
_l_10_50 = XOR _l_16_22 _l_17_21
_l_11_49 = AND _l_10_50 c_n1_27
c_n1_27 = 0
_l_16_22 = SELECT 0 a
_l_17_21 = SELECT 0 b
```

```
Step 1:
a ? 1
b ? 1
=> o = 0
=> c = 1
```

```
Step 2:
a ? 0
```

Exemple: le full-adder

```
INPUT a, b
OUTPUT o, c
VAR
  _l_10_50, _l_11_49, _l_16_22,
  _l_17_21, _l_7_52, _l_9_51,
  a, b, c, c_n1_27, o, s_n_26
IN
o = s_n_26
c = OR _l_9_51 _l_11_49
s_n_26 = XOR _l_7_52 c_n1_27
_l_7_52 = XOR _l_16_22 _l_17_21
_l_9_51 = AND _l_16_22 _l_17_21
_l_10_50 = XOR _l_16_22 _l_17_21
_l_11_49 = AND _l_10_50 c_n1_27
c_n1_27 = 0
_l_16_22 = SELECT 0 a
_l_17_21 = SELECT 0 b
```

```
Step 1:
a ? 1
b ? 1
=> o = 0
=> c = 1
```

```
Step 2:
a ? 0
b ?
```

Exemple: le full-adder

```
INPUT a, b
OUTPUT o, c
VAR
  _l_10_50, _l_11_49, _l_16_22,
  _l_17_21, _l_7_52, _l_9_51,
  a, b, c, c_n1_27, o, s_n_26
IN
o = s_n_26
c = OR _l_9_51 _l_11_49
s_n_26 = XOR _l_7_52 c_n1_27
_l_7_52 = XOR _l_16_22 _l_17_21
_l_9_51 = AND _l_16_22 _l_17_21
_l_10_50 = XOR _l_16_22 _l_17_21
_l_11_49 = AND _l_10_50 c_n1_27
c_n1_27 = 0
_l_16_22 = SELECT 0 a
_l_17_21 = SELECT 0 b
```

```
Step 1:
a ? 1
b ? 1
=> o = 0
=> c = 1
```

```
Step 2:
a ? 0
b ? 1
```

Exemple: le full-adder

```
INPUT a, b
OUTPUT o, c
VAR
  _l_10_50, _l_11_49, _l_16_22,
  _l_17_21, _l_7_52, _l_9_51,
  a, b, c, c_n1_27, o, s_n_26
IN
o = s_n_26
c = OR _l_9_51 _l_11_49
s_n_26 = XOR _l_7_52 c_n1_27
_l_7_52 = XOR _l_16_22 _l_17_21
_l_9_51 = AND _l_16_22 _l_17_21
_l_10_50 = XOR _l_16_22 _l_17_21
_l_11_49 = AND _l_10_50 c_n1_27
c_n1_27 = 0
_l_16_22 = SELECT 0 a
_l_17_21 = SELECT 0 b
```

```
Step 1:
a ? 1
b ? 1
=> o = 0
=> c = 1
```

```
Step 2:
a ? 0
b ? 1
=> o = 1
=> c = 0
```

MiniJazz

Un langage de plus haut-niveau

- Description hiérarchique
- Nappes de fils

Compilé vers le langage de net-list

- Vérifications diverses (noms, typage, etc)
- Déjà fait !!
- http://www.di.ens.fr/~pasteur/enseignement.html
- cf cours de compilation

Exemple: un adder n-bit

```
fulladder(a,b,c) = (s, r) where
  s = (a ^ b) ^ c;
  r = (a & b) + ((a ^ b) & c);
end where

adder<n>(a:[n], b:[n], c_in) = (o:[n], c_out) where
  if n = 0 then
    o = [];
    c_out = 0
  else
    (s_n1, c_n1) = adder<n-1>(a[1..], b[1..], c_in);
    (s_n, c_out) = fulladder(a[0], b[0], c_n1);
    o = s_n . s_n1
  end if
end where

main(a:[2], b:[2]) = (o:[2], c) where
  (o, c) = adder<2>(a,b,0)
end where
```

Le microprocesseur

Conception croisée

- Architecture du microprocesseur
- Jeu d'instructions
- Programme de la montre

Plusieurs approches

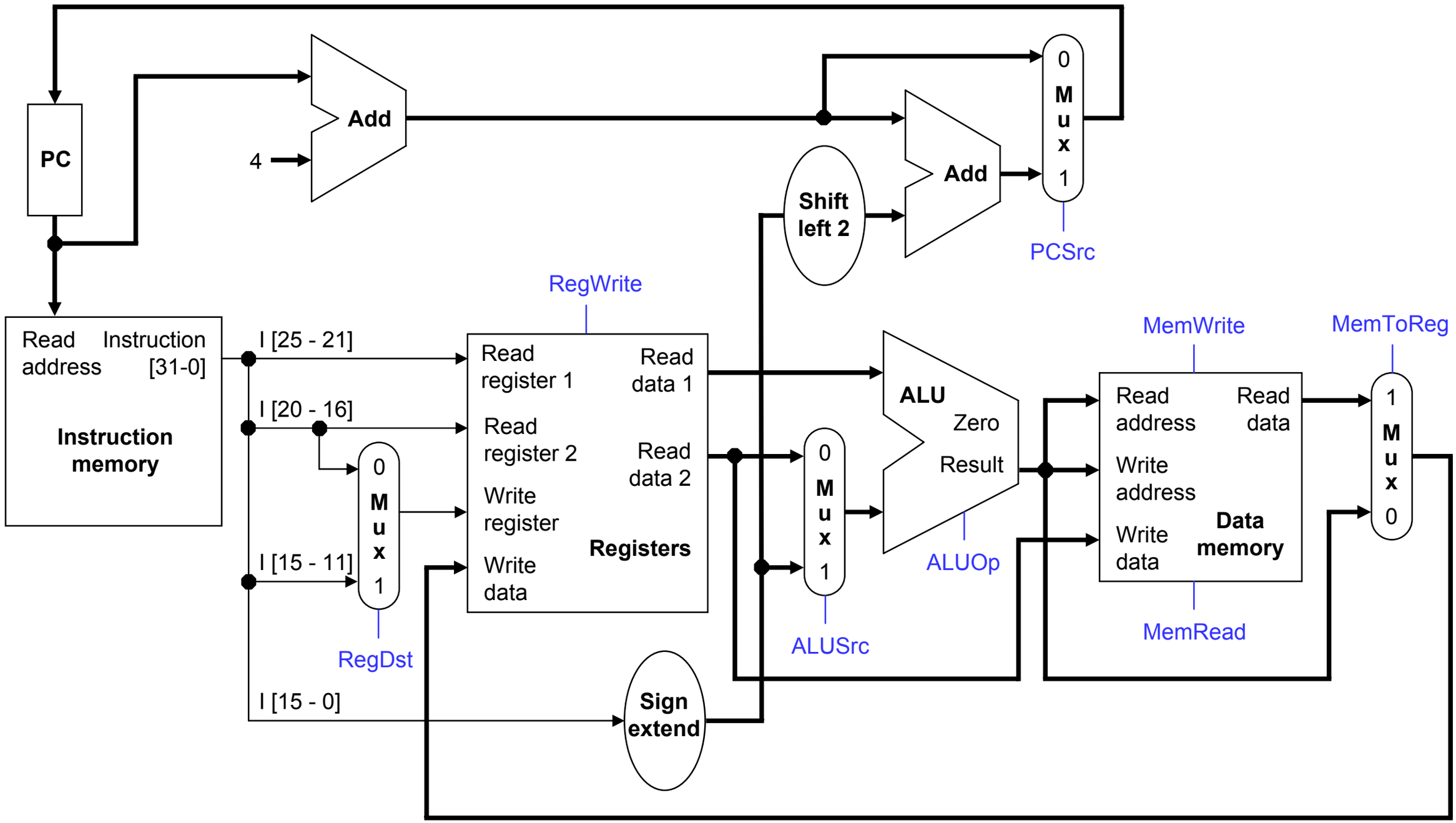
- Processeur généraliste
- Processeur spécialisé (compteur modulo 24)

Le microprocesseur

Un processeur compatible MIPS

- Projet de compilation: d'un langage de haut-niveau vers assembleur MIPS
- Projet de système digital: de l'assembleur MIPS à la simulation des portes logiques
- Pas forcément si compliqué

Le microprocesseur



La montre digitale

Doit gérer

- Heures, minutes, secondes
- Jours, mois, années

Une interface complète

- Mode horloge temps-réel
- Mode vitesse maximum

Aperçu

cpu.mj

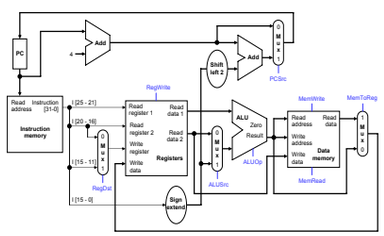
```
fulladder(a,b,cin)
 = (o, cout) where
   o =
...
```

MiniJazz

cpu.net

```
INPUT a, b
OUTPUT o, c
VAR
  _1_11,  _1_10,
  _1_11_63,
...
```

Architecture



Simulateur

23:59:59

Instructions

```
add Rd, Rs, Rt
addi Rd, Rs, im
sub Rd, Rs, Rt
....
```

clock.asm

```
addi $2, $3, 0
slt $3, $4, $5
jal f
....
```

Aperçu

cpu.mj

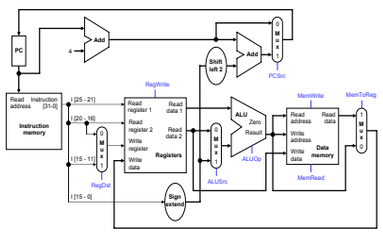
```
fulladder(a,b,cin)
 = (o, cout) where
   o =
...
```

MiniJazz

cpu.net

```
INPUT a, b
OUTPUT o, c
VAR
   _l_11,  _l_10,
   _l_11_63,
...
```

Architecture



1

Simulateur

23:59:59

Instructions

```
add Rd, Rs, Rt
addi Rd, Rs, im
sub Rd, Rs, Rt
....
```

clock.asm

```
addi $2, $3, 0
slt $3, $4, $5
jal f
....
```

Aperçu

cpu.mj

```
fulladder(a,b,cin)
 = (o, cout) where
  o =
...
```

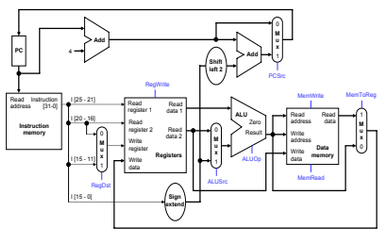
MiniJazz

cpu.net

```
INPUT a, b
OUTPUT o, c
VAR
  _l_11,  _l_10,
  _l_11_63,
...
```

2

Architecture



1

Simulateur

23:59:59

2

Instructions

```
add Rd, Rs, Rt
addi Rd, Rs, im
sub Rd, Rs, Rt
....
```

clock.asm

```
addi $2, $3, 0
slt $3, $4, $5
jal f
....
```

Aperçu

3

cpu.mj

```
fulladder(a,b,cin)
 = (o, cout) where
   o =
...
```

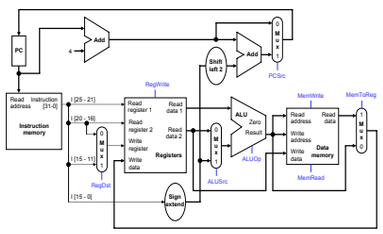
MiniJazz

cpu.net

```
INPUT a, b
OUTPUT o, c
VAR
   _l_11,  _l_10,
   _l_11_63,
...
```

2

Architecture



1

Simulateur

23:59:59

2

Instructions

```
add Rd, Rs, Rt
addi Rd, Rs, im
sub Rd, Rs, Rt
....
```

clock.asm

```
addi $2, $3, 0
slt $3, $4, $5
jal f
....
```

Aperçu

3

cpu.mj

```
fulladder(a,b,cin)
 = (o, cout) where
   o =
...
```

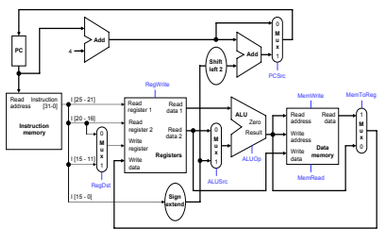
MiniJazz

cpu.net

```
INPUT a, b
OUTPUT o, c
VAR
   _l_11,  _l_10,
   _l_11_63,
...
```

2

Architecture



1

Simulateur

23:59:59

2

Instructions

```
add Rd, Rs, Rt
addi Rd, Rs, im
sub Rd, Rs, Rt
....
```

4

clock.asm

```
addi $2, $3, 0
slt $3, $4, $5
jal f
....
```

Outils

Liberté de choix

- Langage d'implémentation
- Choix d'architecture
- Organisation au sein du groupe

Conseils

- Soyez raisonnables !
- Langage OCaml
- Outils de collaboration
- Documentation

Organisation

Groupes de 3 ou 4 personnes

Séances

- 2 oct.: TP sur l'ordonnancement de net-list
- Plusieurs séances d'aide au projet (23 oct., 20 nov., 4 dec., 8 jan.)
- Deux rapports intermédiaires
  - Simulateur (23 oct.)
  - Architecture du microprocesseur (4 dec.)
- Soutenance (22 jan.) avec démo et rapport final

Conclusion

N’hésitez pas à me poser des questions

- Maintenant
- cedric.pasteur@ens.fr
- http://www.di.ens.fr/~pasteur/enseignement.html pour les énoncés