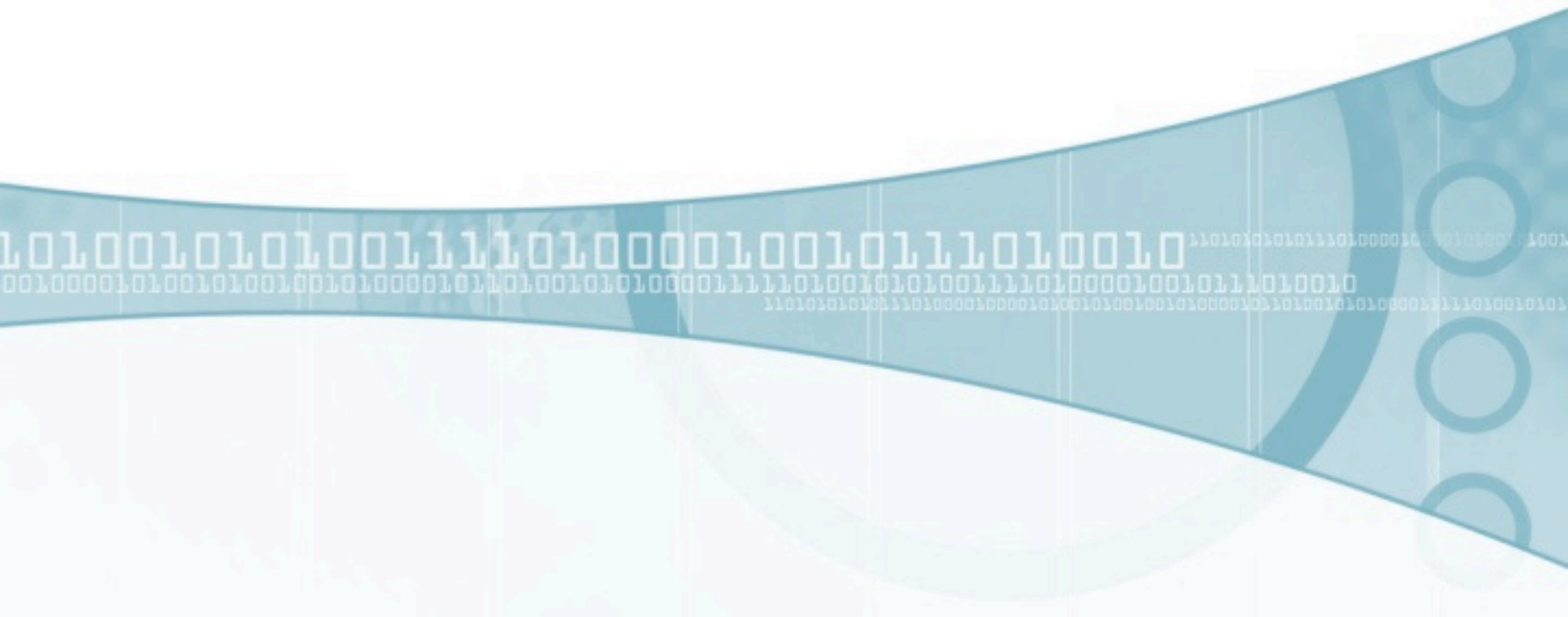
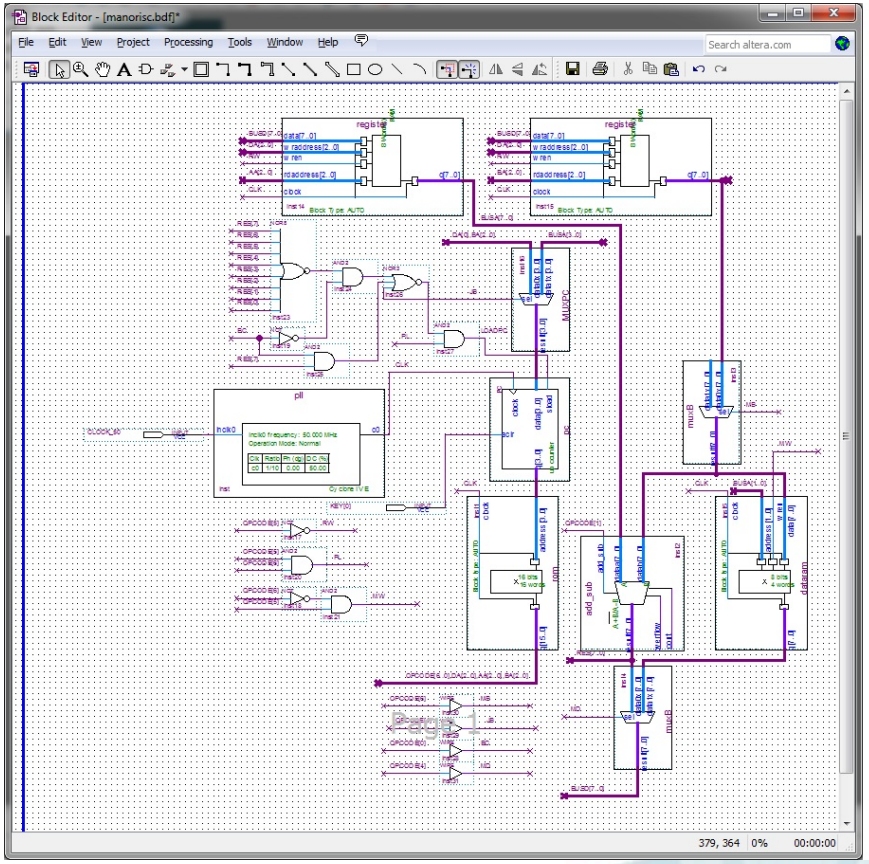
Langage de haut niveau, langage de bas niveau …

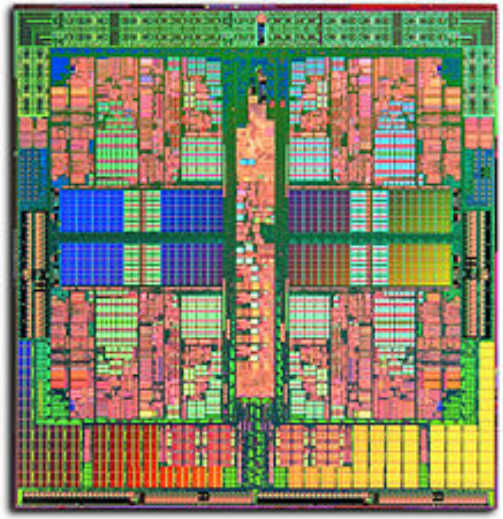


Exemple de programme

```
int main(){
    int i;
    int x = 4;
    int y = 3;
    int produit = 0;

    i = 0;
    while (i<y) {
        produit = produit + x;
        i = i + 1;
    }
}
```

Le processeur est un circuit …





Langage de haut niveau

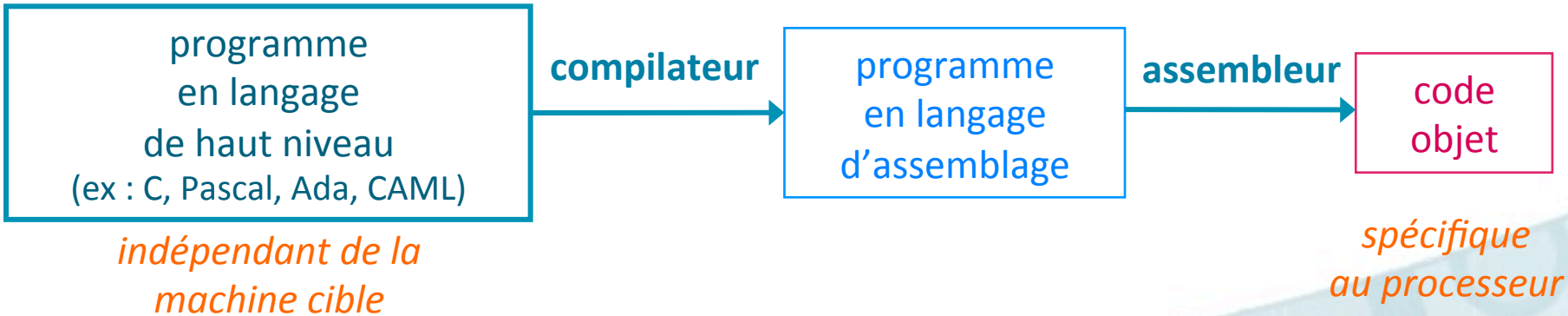
```
int main(){
      int i;
      int x = 4;
      int y = 3;
      int produit = 0;

      i = 0;
      while (i<y) {
            produit = produit + x;
            i = i + 1;
      }
}
```

- **Difficilement traitable par un circuit …**
  - opérations complexes à réaliser :
    - lire les caractères
    - reconstituer les mots
    - identifier les instructions et les structures de contrôle, …
  - pour chaque langage, il faudrait un nouveau circuit !
  - l'analyse du texte devrait être faite à chaque exécution
  - le programme prendrait beaucoup de place en mémoire (un caractère codé sur un octet)

- **Solution : langage de bas niveau**

Langage de bas niveau

- **Instructions codées en binaire**
  - facilement identifiables par le processeur
  - n'occupent pas trop de place en mémoire
  - peuvent être représentées par des mnémoniques pour plus de lisibilité (pour le programmeur)
- **Traduction**



Jeu d'instructions : exemple

- **Instructions d'accès à la mémoire**

`ldr rd, @donnée`	*lecture d'une donnée en mémoire*
`str rs, @donnée`	*écriture d'une donnée en mémoire*

- **Instructions de calcul**

`add rd, rs1, rs2`	*addition rd ← rs1 + rs2*
`add rd, rs1, #constante`	*addition rd ← rs1 + constante*
`sub rd, rs1, rs2`	*soustraction rd ← rs1 - rs2*
`cmp rs1, rs2`	*comparaison*
`mov rd,#constante`	*affectation rd ← constante*

- **Instructions de contrôle**

`b @instruction`	*branchement (PC ← @instruction)*
`bge @instruction`	*branchement si supérieur ou égal*

Du langage C au langage d'assemblage …

```
main(){
  int x = 4;
  int y = 3;
  int produit = 0;



  int i = 0;
  while (i<y) {

     produit = produit + x;
     i++;
  }
}
```

```
x:          .int 4
y:          .int 3
produit:    .int 0

main:       ldr r1, x
            ldr r2, y
            ldr r3, produit
            mov r4, #0
boucle:     cmp r4, r2
            bge fin
            add r3, r1, r3
            add r4, r4, #1
            b   boucle
fin:
            str r3, produit
```