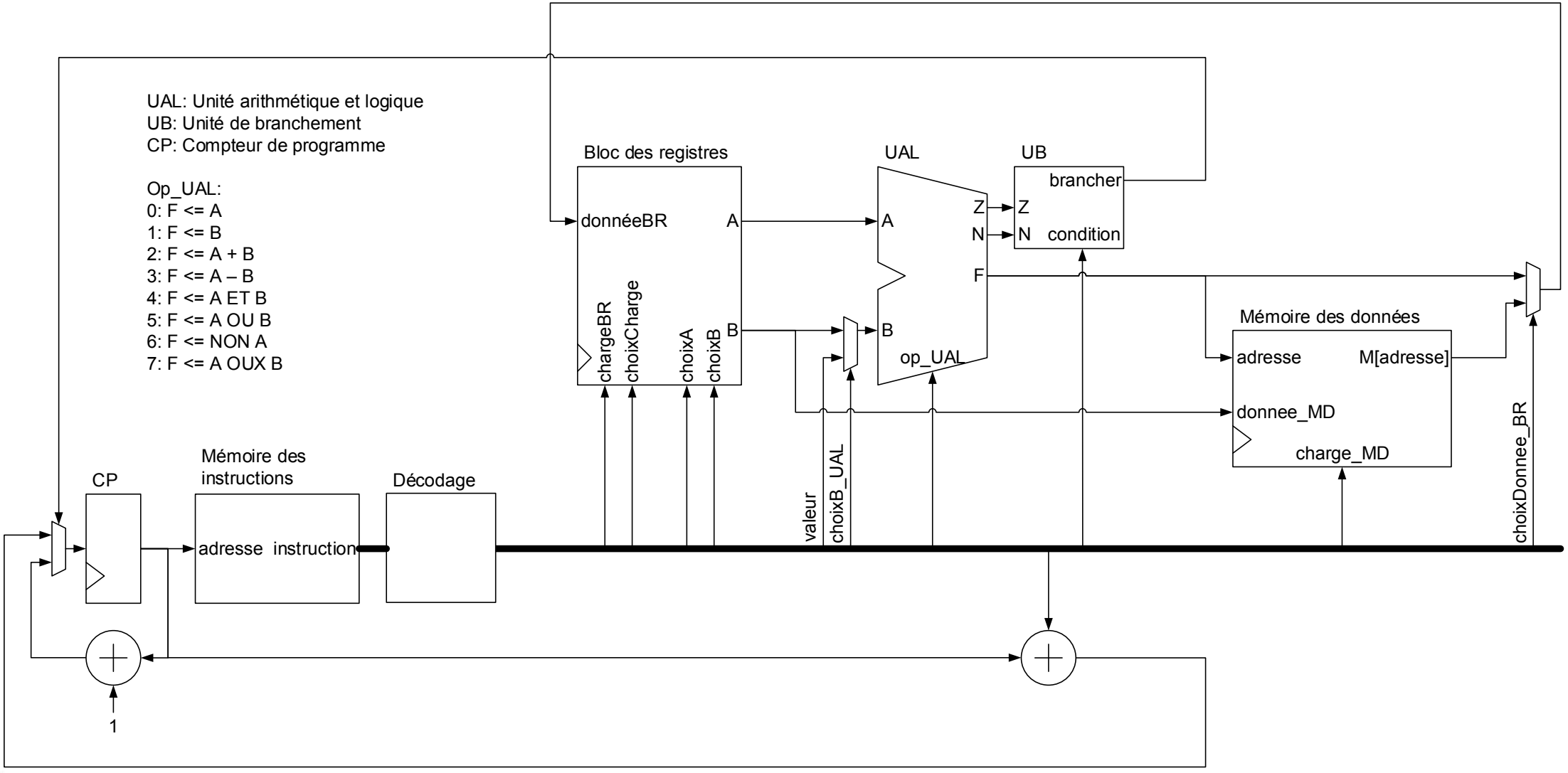
Encodage du jeu d'instructions d'un processeur à usage général

Encodage du jeu d’instructions d’un processeur à usage général
Sujets de ce thème

- Rappel: processeur PolyRISC
- Les jeux d’instructions
- Encodage du jeu d’instructions:
  - Opérations sur les registres
  - Opérations registre-valeur
  - Branchements
  - Opérations avec la mémoire
- Exemples
- Écrire un programme

Le processeur à usage général PolyRISC

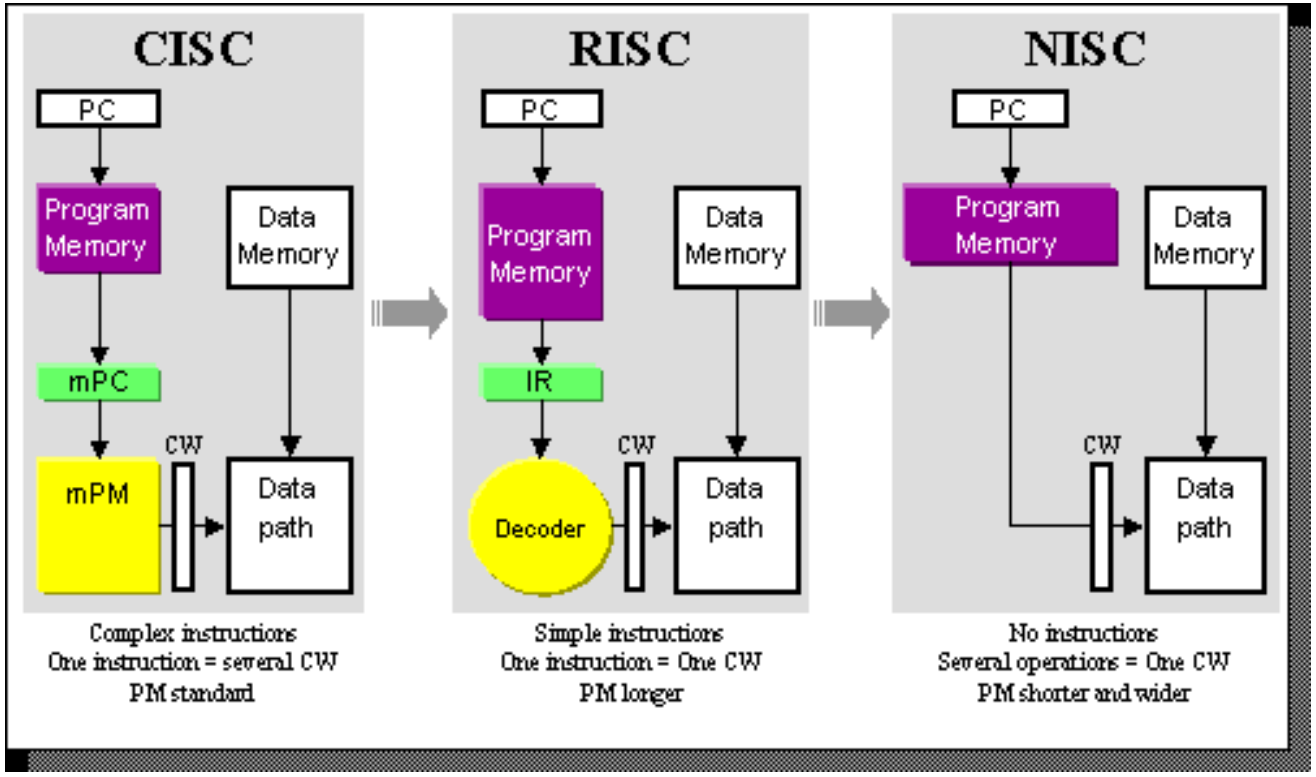


Jeu d'instructions d'un processeur à usage général

- Le jeu d'instructions (*Instruction Set Architecture* – ISA) est la spécification de toutes les instructions pouvant être effectuées par le processeur, leur encodage numérique, et l'inventaire des ressources mises à la disponibilité des programmeurs (comme des registres, des piles, des coprocesseurs, etc.).
- Le jeu d'instructions est le contrat entre les concepteurs du processeur et les programmeurs.
- On peut catégoriser les jeux d'instructions selon plusieurs dimensions. Une dimension concerne la forme du chemin des données du processeur, selon qu'il contient ou non des registres et de la mémoire.
- Les processeurs de type registre-registre (*register-register* ou *load-store*) sont les plus populaires. Pour ce type de processeur, les opérations peuvent être effectuées uniquement sur des valeurs gardées dans des registres. Il est possible de transférer des données entre les registres et la mémoire.
- On distingue aussi :
  - Architecture Harvard: mémoires séparées pour le programme et les données
  - Architecture Von Neumann: une seule mémoire

Choix d'approche pour l'encodage des instructions

- Le code d'une instruction doit spécifier sans ambiguïté tous les signaux de contrôle du chemin des données.
- Il y a trois grandes approches:
  - CISC: *complex instruction set computing*
  - RISC: *reduced instruction set computing*
  - NISC: *no instruction set computing*
- Nous choisissons l'approche RISC.
  - Les signaux de contrôle du chemin des données sont encodés, ce qui réduit la taille du programme.
  - La spécification du processeur monte alors au niveau du jeu d'instructions.
  - L'implémentation peut être différente pour deux processeurs qui ont le même comportement.



NISC Technology and Toolset, University of California Irvine, [en ligne: http://www.ics.uci.edu/~nisc/], consulté le 2009/11/11

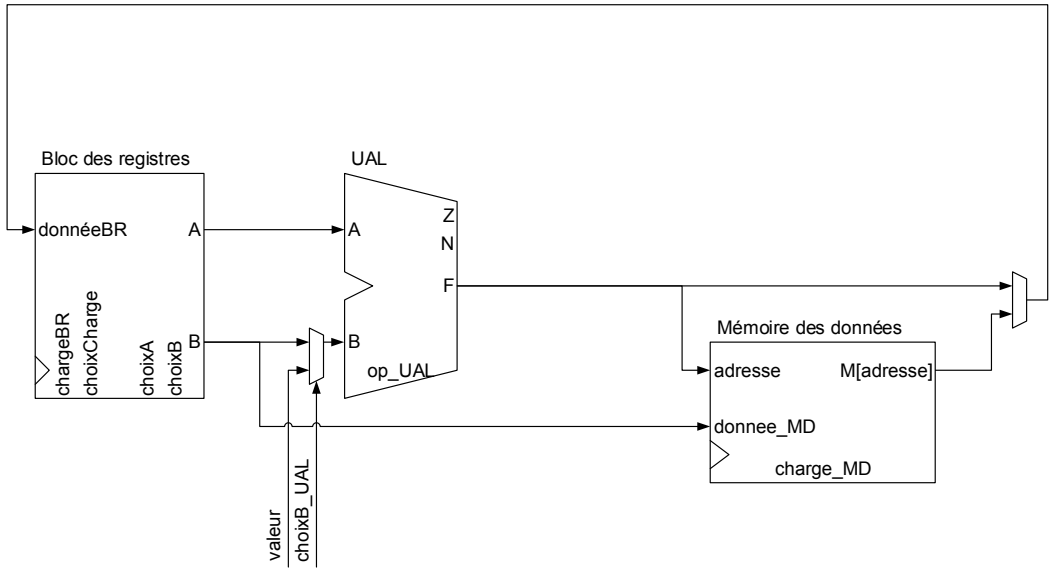
Encodage des instructions en mémoire

- Considérations pour le format des instructions et de leur encodage en mémoire
  - C'est un problème à plusieurs solutions possibles.
  - On favorise la régularité, ce qui simplifie le décodage et accélère le processeur.
  - L'encodage, le jeux d'instructions et les modes d'adressage sont intimement liés.
- Choix effectués ici:
  - Encodage sur un seul mot de 32 bits: chaque instruction s'exécute en un seul cycle.
  - Le mot d'instruction est divisé en 6 champs.
  - Modes d'adressage limités conformément à la philosophie RISC.

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Encodage des instructions en mémoire
Catégorie 0: opération sur les registres

- Le processeur a 32 registres de 16 bits de large.
- Avec 4 bits pour les détails, l'UAL pourrait effectuer 16 opérations différentes, dont:
  - A, B, A + B, A – B, A ET B, A OU B, NON(A), A OUX B
  - A(15:0) × B(15:0), décaleG(A, B), décaleD(A, B)
  - etc.



Bits 29:26
0: F <= A
1: F <= B
2: F <= A + B
3: F <= A – B
4: F <= A ET B
5: F <= A OU B
6: F <= NON A
7: F <= A OUX B

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Exemples d'encodage d'instructions

Bits 29:26
(cat. 0 et 1)

0: F <= A
1: F <= B
2: F <= A + B
3: F <= A – B
4: F <= A ET B
5: F <= A OU B
6: F <= NON A
7: F <= A OUX B

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
R7 := R5 + R2	0	2	7	5	-	2
R9 := R0 – R11	0	3	9	0	-	11
R8 := R4	0	0	8	4	-	-

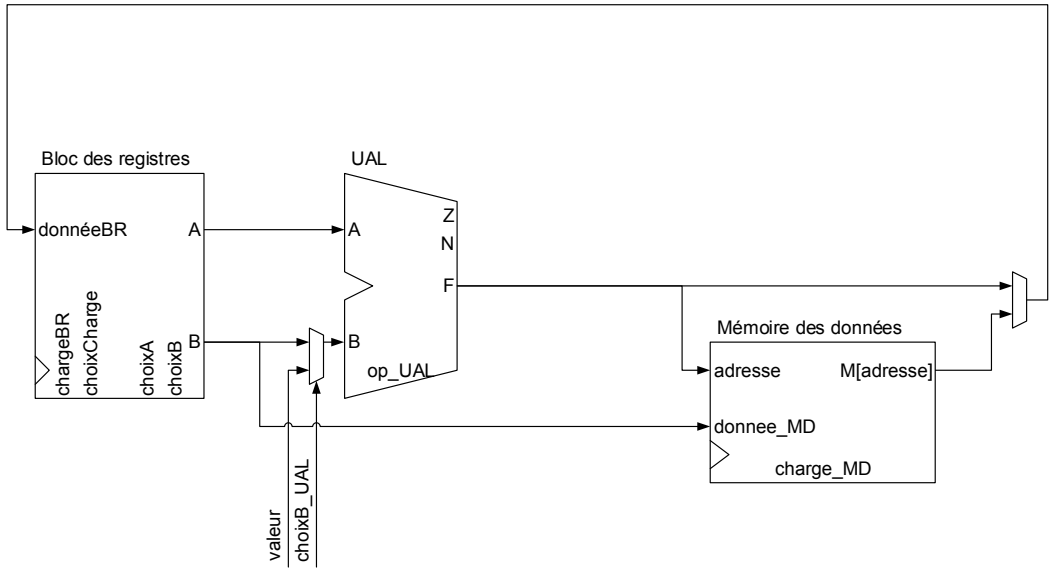
Bits 29:26
(cat. 2)

0: si =
1: si !=
2: si <
3: si >
4: si <=
5: si >=
6: toujours
7: jamais

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Encodage des instructions en mémoire
Catégorie 1: opération sur un registre et une valeur codée dans l'instruction

- Les bits 29:26 spécifient l'opération, comme pour la catégorie 0.
- Les bits 15:0 de l'instruction contiennent la valeur à utiliser.



Bits 29:26
0: F <= A
1: F <= B
2: F <= A + B
3: F <= A – B
4: F <= A ET B
5: F <= A OU B
6: F <= NON A
7: F <= A OUX B

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Exemples d'encodage d'instructions

Bits 29:26
(cat. 0 et 1)

0: F <= A
1: F <= B
2: F <= A + B
3: F <= A – B
4: F <= A ET B
5: F <= A OU B
6: F <= NON A
7: F <= A OUX B

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
R31 := R0 ET 0xFF	1	4	31	0	255	

Bits 29:26
(cat. 2)

0: si =
1: si !=
2: si <
3: si >
4: si <=
5: si >=
6: toujours
7: jamais

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Encodage des instructions en mémoire
Catégorie 2: branchement

- Les bits 29:26 permettent d'avoir 16 conditions de branchement possibles. En pratique, 8 suffisent.

  =, ≠, <, >, ≤, ≥,

  toujours (sans condition), jamais (= NOP)

- Les branchements sont limités à $\pm 2^{15}$ adresses par rapport au CP, encodées sur les bits 15:0.
- La valeur des bits 15:0 est étendue sur 32 bits et ajoutée à la valeur incrémentée du CP. On peut donc brancher vers l'avant (valeurs positives) ou l'arrière (valeurs négatives).

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Exemples d'encodage d'instructions

Bits 29:26
(cat. 0 et 1)

0: F <= A
1: F <= B
2: F <= A + B
3: F <= A – B
4: F <= A ET B
5: F <= A OU B
6: F <= NON A
7: F <= A OUX B

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
Si R8 = R9, goto PC + 25	2	0	8	9	25	

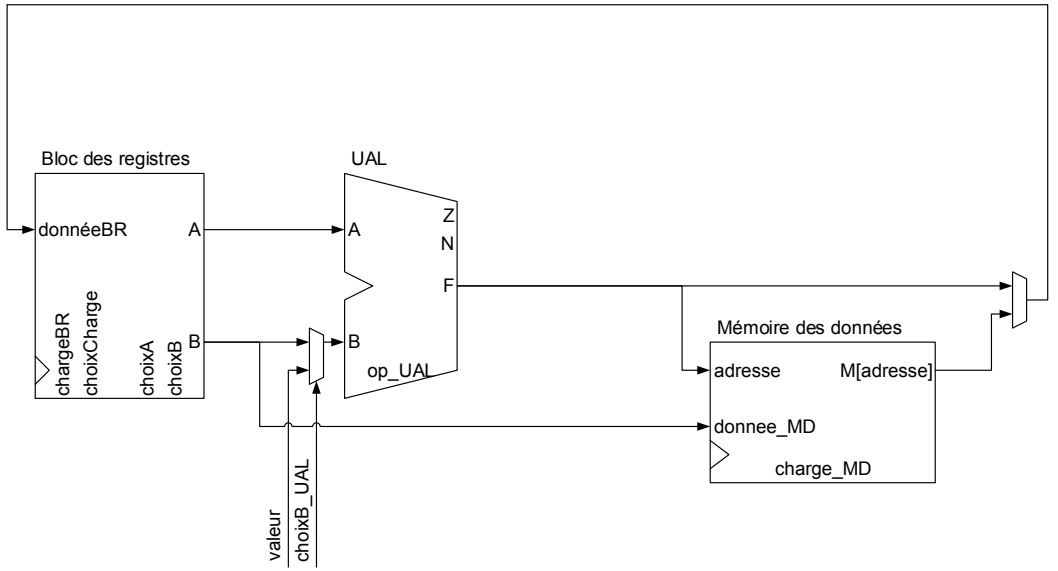
Bits 29:26
(cat. 2)

0: si =
1: si !=
2: si <
3: si >
4: si <=
5: si >=
6: toujours
7: jamais

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Encodage des instructions en mémoire
Catégorie 3: mémoire (*Load-Store*)

- L'adresse effective est composée de la somme d'un registre et d'une valeur de déplacement. Ceci facilite l'accès aux éléments d'un tableau.
- La mémoire des données peut avoir jusqu'à 2^{32} mots de 32 bits (16 Go).



Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Exemples d'encodage d'instructions

Bits 29:26
(cat. 0 et 1)

0: F <= A
1: F <= B
2: F <= A + B
3: F <= A – B
4: F <= A ET B
5: F <= A OU B
6: F <= NON A
7: F <= A OUX B

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
R12 := MD[R0 + 240]	3	0	12	0	240	
R14 := MD[R7 + 4]	3	0	14	7	4	
MD[R0 + 28] := R10	3	1	10	0	28	

Bits 29:26
(cat. 2)

0: si =
1: si !=
2: si <
3: si >
4: si <=
5: si >=
6: toujours
7: jamais

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Exemples d'encodage d'instructions

Bits 29:26
(cat. 0 et 1)

0: F <= A
1: F <= B
2: F <= A + B
3: F <= A – B
4: F <= A ET B
5: F <= A OU B
6: F <= NON A
7: F <= A OUX B

Bits 29:26
(cat. 2)

0: si =
1: si !=
2: si <
3: si >
4: si <=
5: si >=
6: toujours
7: jamais

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
R7 := R5 + R2	0	2	7	5	-	2
R9 := R0 – R11	0	3	9	0	-	11
R8 := R4	0	0	8	4	-	-
R31 := R0 ET 0xFF	1	4	31	0	255	
Si R8 = R9, goto PC + 25	2	0	8	9	25	
R12 := MD[R0 + 240]	3	0	12	0	240	
MD[R0 + 28] := R10	3	1	10	0	28	
R14 := MD[R7 + 4]	3	0	14	7	4	

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	

Exemple de programme

Adresse	Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
0	R0 := M[0]						
1	R1 := M[1]						
2	Si R0 > R1, goto 5						
3	R12 := R1 – R0						
4	goto 6						
5	R12 := R0 – R1						
6	M[3] := R12						
7	Goto 7						

Le problème du logiciel

- L'écriture directe d'un programme en codes de 32 bits est ardue.
- Pour simplifier la tâche, on utilise un assembleur qui donne automatiquement les codes numériques des instructions à partir de leur représentation symbolique.
- Pour les programmes très complexes, cependant, on utilise plutôt un compilateur qui accepte en entrée un programme dans un langage de haut niveau comme C ou Python et qui produit les codes des instructions.
- Le compilateur doit s'appuyer sur une connaissance approfondie des instructions pouvant être réalisées par le processeur ainsi que des architectures de son chemin des données et de son unité de contrôle.
- Un simulateur et un débogueur seraient bien aussi … ☺

Vous devriez maintenant être capable de …

- Expliquer comment effectuer l'encodage et le décodage des instructions. (B2)
- Interpréter et écrire des programmes simples pour le processeur. (B3)

Code	Niveau (http://fr.wikipedia.org/wiki/Taxonomie_de_Bloom)
B1	Connaissance – mémoriser de l'information.
B2	Compréhension – interpréter l'information.
B3	Application – confronter les connaissances à des cas pratiques simples.
B4	Analyse – décomposer un problème, cas pratiques plus complexes.
B5	Synthèse – expression personnelle, cas pratiques plus complexes.

Encodage bonifié des instructions en mémoire

- Encodage bonifié

Instruction	bits 31-30 catégorie	bits 29:26 détails	bits 25:21	bits 20:16	bits 15:5	bits 4:0
RC := RA ◊ RB	00	code de ◊	RC	RA	-	RB
RC := RA ◊ valeur	01	code de ◊	RC	RA	valeur(15:0)	
si (RA ◊ RB) goto (CP + valeur)	10	code de ◊	RB	RA	valeur(15:0)	
RC := MD[RA + valeur]	11	0000	RC	RA	valeur(15:0)	
MD[RA + valeur] := RB	11	0001	RB	RA	valeur(15:0)	
RC := GPIO_in	11	0010	RC	-	-	
GPIO_out := RB	11	0011	RB	-	-	