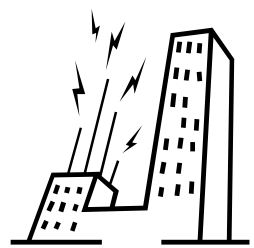
Architecture des ordinateurs V

Frédéric Boulanger

CentraleSupélec



Conception d'un microprocesseur

Architecture de von Neumann

Les ordinateurs actuels ont une architecture caractérisée en 1945 par John von Neumann :

- unité arithmétique et logique (UAL)

Architecture de von Neumann

Les ordinateurs actuels ont une architecture caractérisée en 1945 par John von Neumann :

- unité arithmétique et logique (UAL)
- unité de contrôle (séquençage des opérations)

Architecture de von Neumann

Les ordinateurs actuels ont une architecture caractérisée en 1945 par John von Neumann :

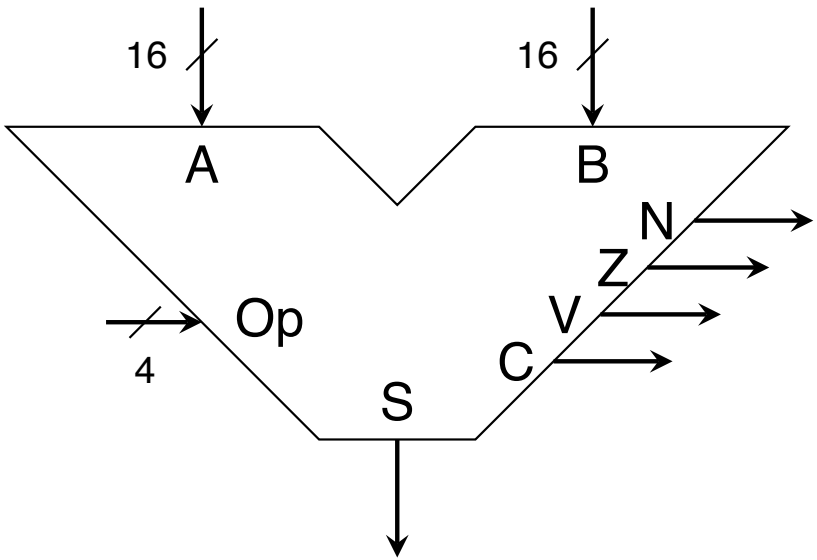
- unité arithmétique et logique (UAL)
- unité de contrôle (séquençage des opérations)
- mémoire (données et programme)

Architecture de von Neumann

Les ordinateurs actuels ont une architecture caractérisée en 1945 par John von Neumann :

- unité arithmétique et logique (UAL)
- unité de contrôle (séquençage des opérations)
- mémoire (données et programme)
- dispositifs d'entrée-sortie

Unité arithmétique et logique



Exemple de l'UAL du cours

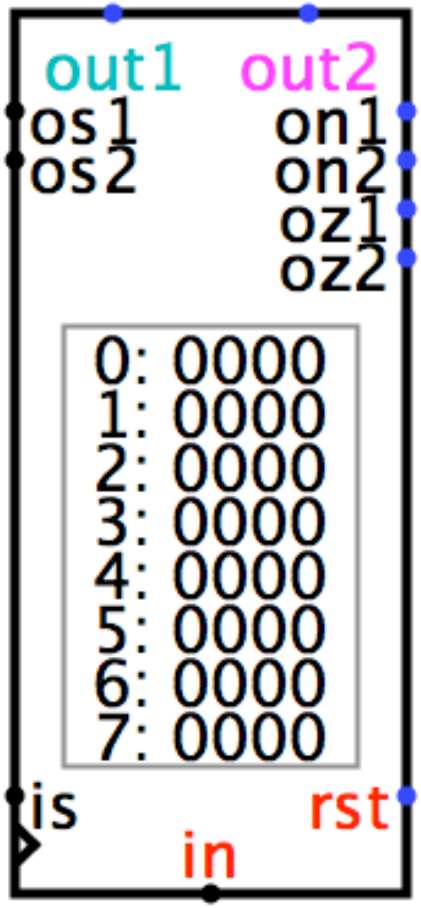
Op_3	Op_2	Op_1	Op_0	Opération
0	0	0	0	S = A
0	0	0	1	S = A ou B
0	0	1	0	S = non (A ou B)
0	0	1	1	S = non (A et B)
0	1	0	0	S = A et B
0	1	0	1	S = B
0	1	1	0	S = non B
0	1	1	1	S = A xor B
1	0	0	0	S = A - B
1	0	0	1	S = A + B
1	0	1	0	S = A + A
1	0	1	1	S = A - 1
1	1	0	0	S = A + 1
1	1	0	1	S = FFFF
1	1	1	0	S = 0
1	1	1	1	S = non A

Registres

Un processeur utilise des registres pour stocker les arguments et les résultats des opérations élémentaires.

L'accès aux registres est plus rapide que l'accès à la mémoire.

Notre processeur utilise un banc de 8 registres :



os1 n° du registre pour **out1**

os2 n° du registre pour **out2**

is n° du registre à charger avec **in**

on1 indique si le registre **os1** est négatif

on2 indique si le registre **os2** est négatif

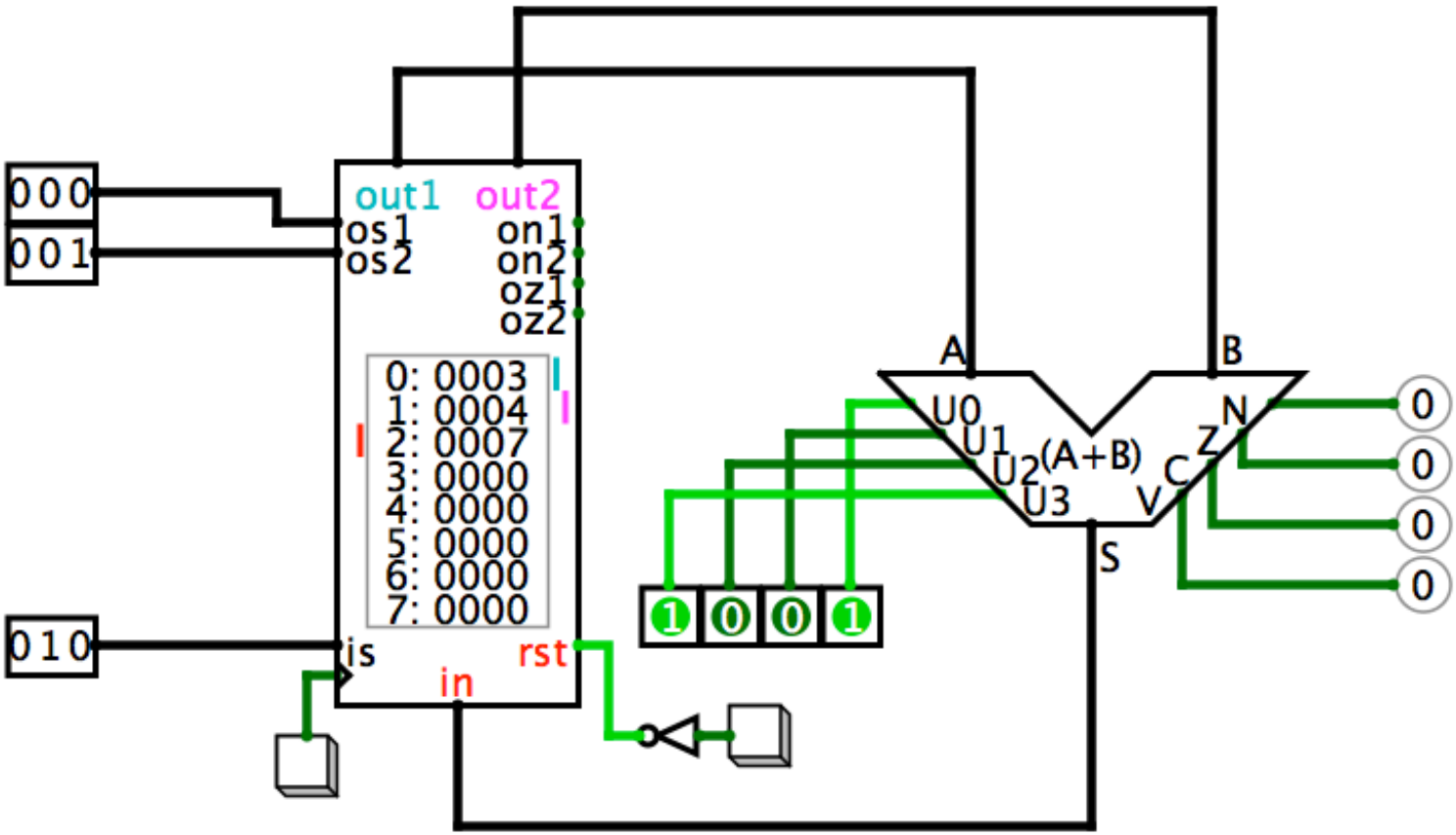
on1 indique si le registre **os1** est nul

on2 indique si le registre **os2** est nul

clk horloge de chargement de **in**

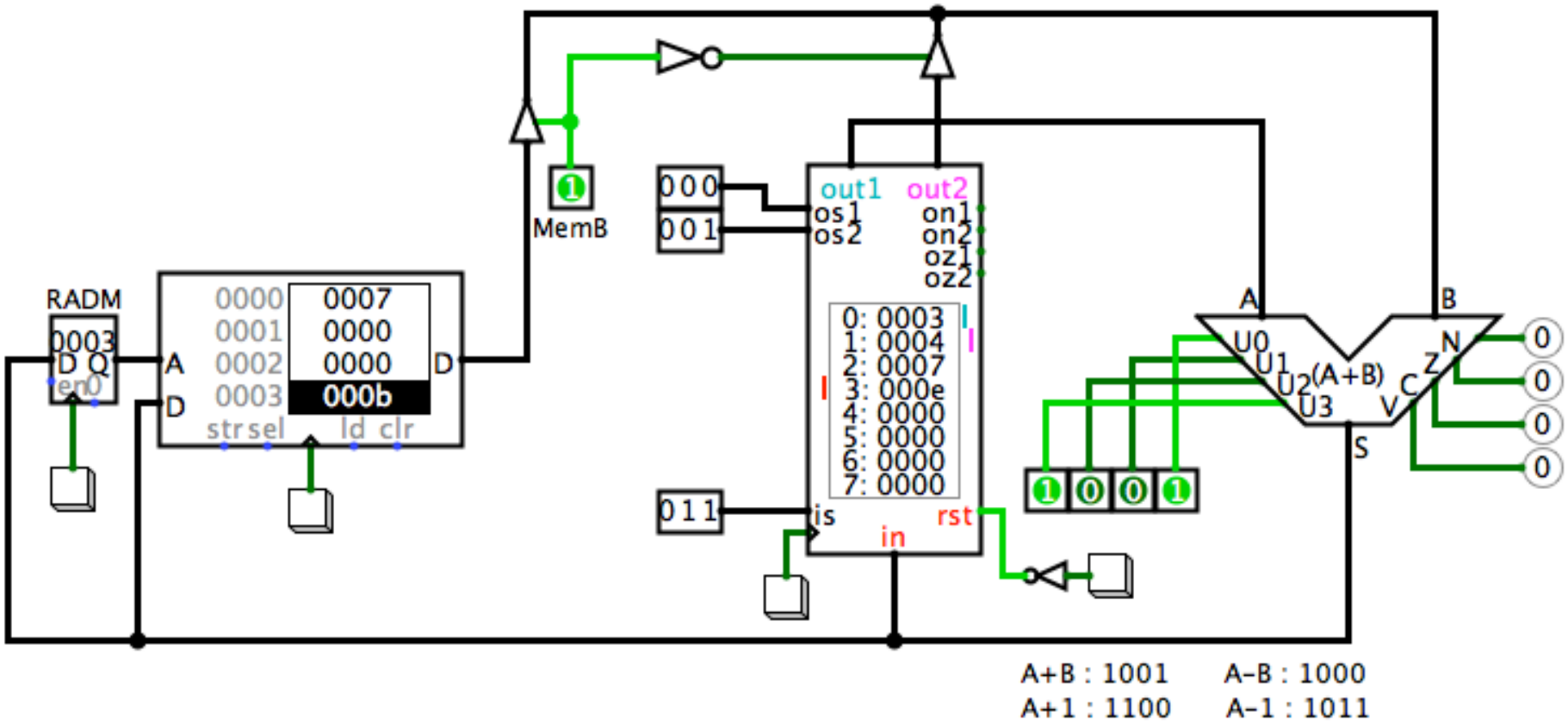
Une calculette

En assemblant l'UAL et le banc de registres, on obtient une calculette qui est le cœur de notre processeur :



Circuit

Une calculette avec mémoire



Circuit

Retour à von Neumann

- unité arithmétique et logique (UAL) OK

Retour à von Neumann

- unité arithmétique et logique (UAL) OK
- mémoire (données et programme) OK

Retour à von Neumann

- unité arithmétique et logique (UAL) OK
- mémoire (données et programme) OK
- unité de contrôle (séquençage des opérations) ?

Retour à von Neumann

- unité arithmétique et logique (UAL) OK
- mémoire (données et programme) OK
- unité de contrôle (séquençage des opérations) ?
- dispositifs d'entrée-sortie plus tard

Unité de contrôle (séquenceur)

- Décompose l'exécution d'une instruction en étapes

Unité de contrôle (séquenceur)

- Décompose l'exécution d'une instruction en étapes
- Donne à chaque étape les signaux :

Unité de contrôle (séquenceur)

- Décompose l'exécution d'une instruction en étapes
- Donne à chaque étape les signaux :
  - de sélection des registres du banc

Unité de contrôle (séquenceur)

- Décompose l'exécution d'une instruction en étapes
- Donne à chaque étape les signaux :
  - de sélection des registres du banc
  - de chargement des registres

Unité de contrôle (séquenceur)

- Décompose l'exécution d'une instruction en étapes
- Donne à chaque étape les signaux :
  - de sélection des registres du banc
  - de chargement des registres
  - du code opération de l'UAL

Unité de contrôle (séquenceur)

- Décompose l'exécution d'une instruction en étapes
- Donne à chaque étape les signaux :
  - de sélection des registres du banc
  - de chargement des registres
  - du code opération de l'UAL
  - de connexion de la mémoire ou des registres à l'entrée B

Unité de contrôle (séquenceur)

- Décompose l'exécution d'une instruction en étapes
- Donne à chaque étape les signaux :
  - de sélection des registres du banc
  - de chargement des registres
  - du code opération de l'UAL
  - de connexion de la mémoire ou des registres à l'entrée B
  - de lecture/écriture en mémoire

Unité de contrôle (séquenceur)

- Décompose l'exécution d'une instruction en étapes
- Donne à chaque étape les signaux :
  - de sélection des registres du banc
  - de chargement des registres
  - du code opération de l'UAL
  - de connexion de la mémoire ou des registres à l'entrée B
  - de lecture/écriture en mémoire
- A besoin de savoir quelle instruction on exécute :

  Registre d'instruction **RI**

Codage des instructions

Notre processeur doit pouvoir exécuter différentes instructions :

- chargement d'un registre depuis une case mémoire ;

Codage des instructions

Notre processeur doit pouvoir exécuter différentes instructions :

- chargement d'un registre depuis une case mémoire ;
- écriture de la valeur d'un registre dans une case mémoire ;

Codage des instructions

Notre processeur doit pouvoir exécuter différentes instructions :

- chargement d'un registre depuis une case mémoire ;
- écriture de la valeur d'un registre dans une case mémoire ;
- copie d'une valeur d'un registre à un autre ;

Codage des instructions

Notre processeur doit pouvoir exécuter différentes instructions :

- chargement d'un registre depuis une case mémoire ;
- écriture de la valeur d'un registre dans une case mémoire ;
- copie d'une valeur d'un registre à un autre ;
- addition de deux registres ;

Codage des instructions

Notre processeur doit pouvoir exécuter différentes instructions :

- chargement d'un registre depuis une case mémoire ;
- écriture de la valeur d'un registre dans une case mémoire ;
- copie d'une valeur d'un registre à un autre ;
- addition de deux registres ;
- soustraction de deux registres ;

Codage des instructions

Notre processeur doit pouvoir exécuter différentes instructions :

- chargement d'un registre depuis une case mémoire ;
- écriture de la valeur d'un registre dans une case mémoire ;
- copie d'une valeur d'un registre à un autre ;
- addition de deux registres ;
- soustraction de deux registres ;
- test de conditions :

Codage des instructions

Notre processeur doit pouvoir exécuter différentes instructions :

- chargement d'un registre depuis une case mémoire ;
- écriture de la valeur d'un registre dans une case mémoire ;
- copie d'une valeur d'un registre à un autre ;
- addition de deux registres ;
- soustraction de deux registres ;
- test de conditions :
  - exécuter une instruction si un résultat est nul

Codage des instructions

Notre processeur doit pouvoir exécuter différentes instructions :

- chargement d'un registre depuis une case mémoire ;
- écriture de la valeur d'un registre dans une case mémoire ;
- copie d'une valeur d'un registre à un autre ;
- addition de deux registres ;
- soustraction de deux registres ;
- test de conditions :
  - exécuter une instruction si un résultat est nul
  - exécuter une instruction si un résultat est négatif

Accès à la mémoire

- chargement d'un registre :

  `ldr rx, adresse`

  charge le registre `rx` avec le contenu de la case mémoire d'adresse `adresse`.

Accès à la mémoire

- chargement d'un registre :

  `ldr rx, adresse`

  charge le registre `rx` avec le contenu de la case mémoire d'adresse `adresse`.

  Par exemple, si la case mémoire d'adresse `0x002C` contient la valeur `0x004A`, l'instruction `ldr r1, 0x002C` charge la valeur `0x004A` dans le registre `r1`.

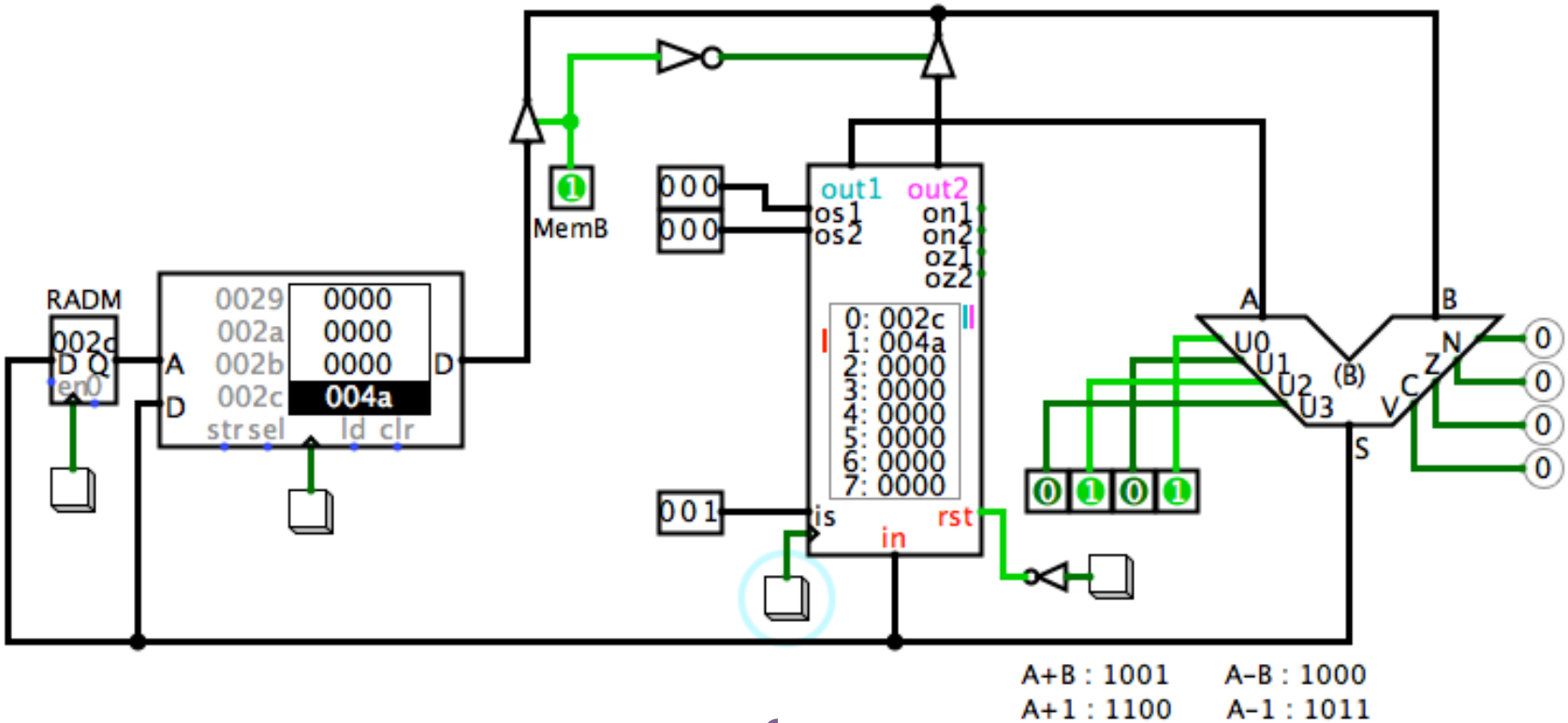
Accès à la mémoire

- chargement d'un registre :

  `ldr rx, adresse`

  charge le registre `rx` avec le contenu de la case mémoire d'adresse `adresse`.

  Par exemple, si la case mémoire d'adresse `0x002C` contient la valeur `0x004A`, l'instruction `ldr r1, 0x002C` charge la valeur `0x004A` dans le registre `r1`.



Accès à la mémoire

- écriture d'un registre :

  `str rz, adresse`

  écrit la valeur du registre `rz` dans la case mémoire d'adresse `adresse`.

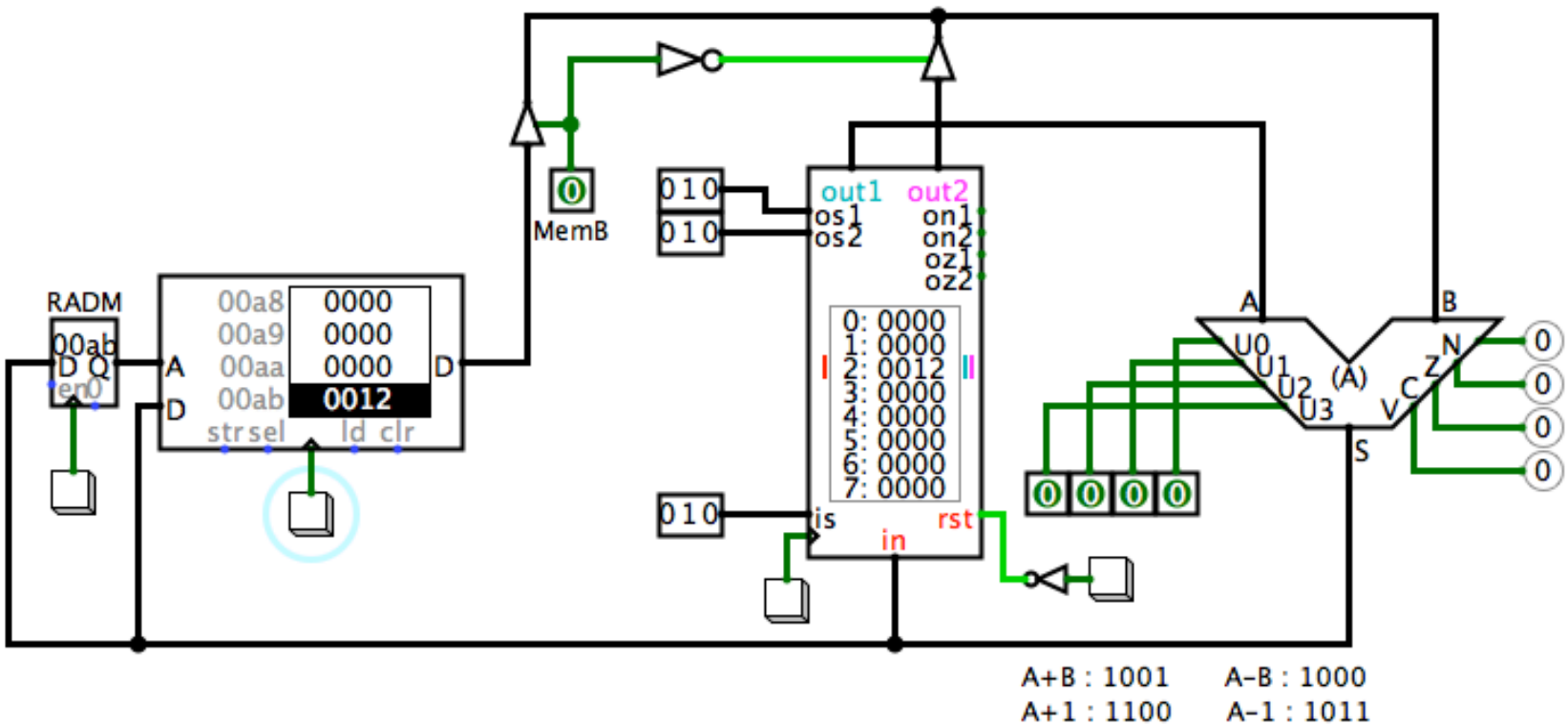
Accès à la mémoire

- écriture d'un registre :

  `str rz, adresse`

  écrit la valeur du registre `rz` dans la case mémoire d'adresse `adresse`.

  Par exemple, si le registre `r2` contient la valeur `0x0012`, l'instruction `str r2, 0x00AB` écrit la valeur `0x0012` dans la mémoire à l'adresse `0x00AB`.

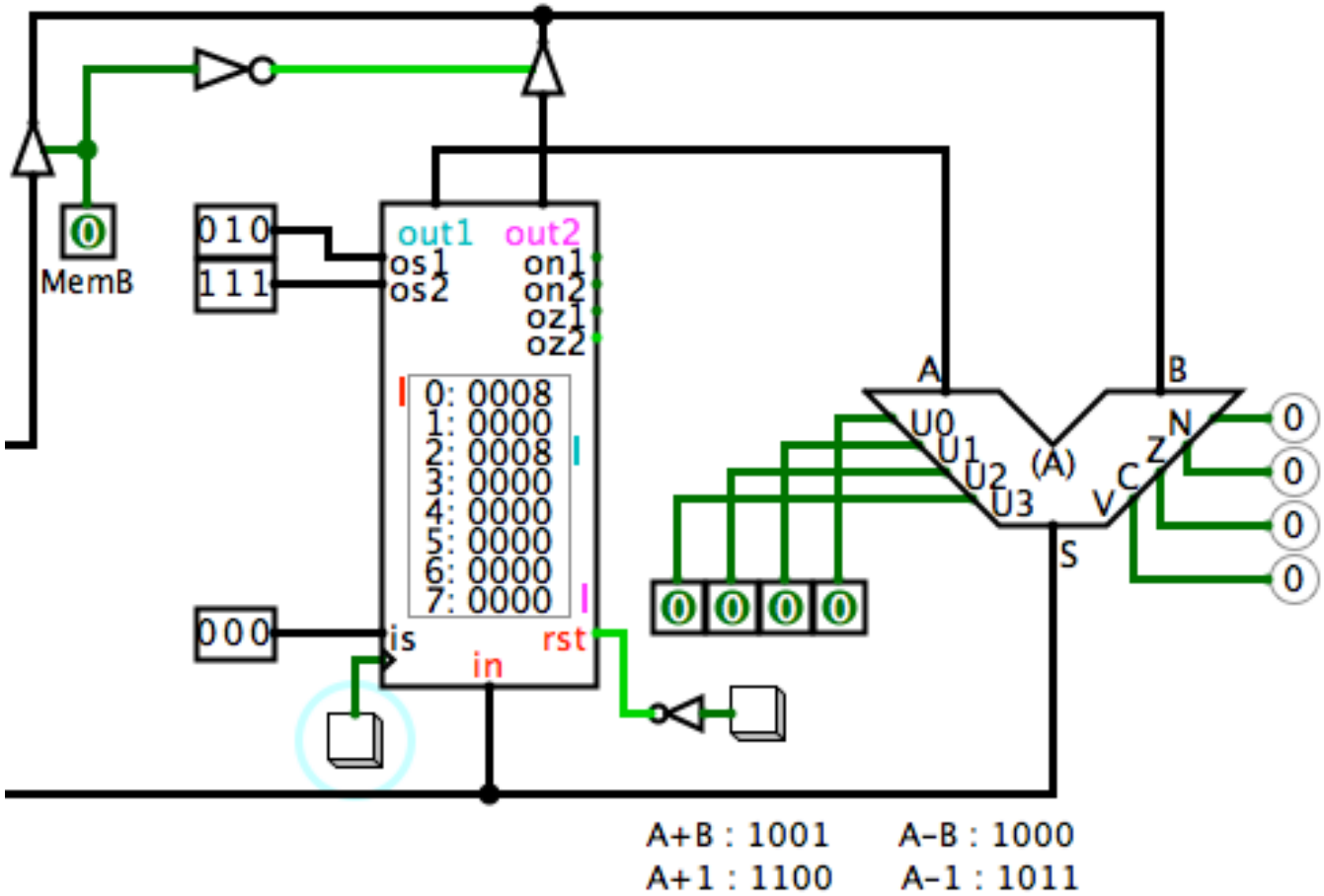


Copie d'une valeur d'un registre à un autre

`mov rx, ry`

copie la valeur du registre `ry` dans le registre `rx`.

Par exemple, si `r2` contient `0x0008`, l'instruction `mov r0, r2` place la valeur `0x0008` dans le regsitre `r0`.

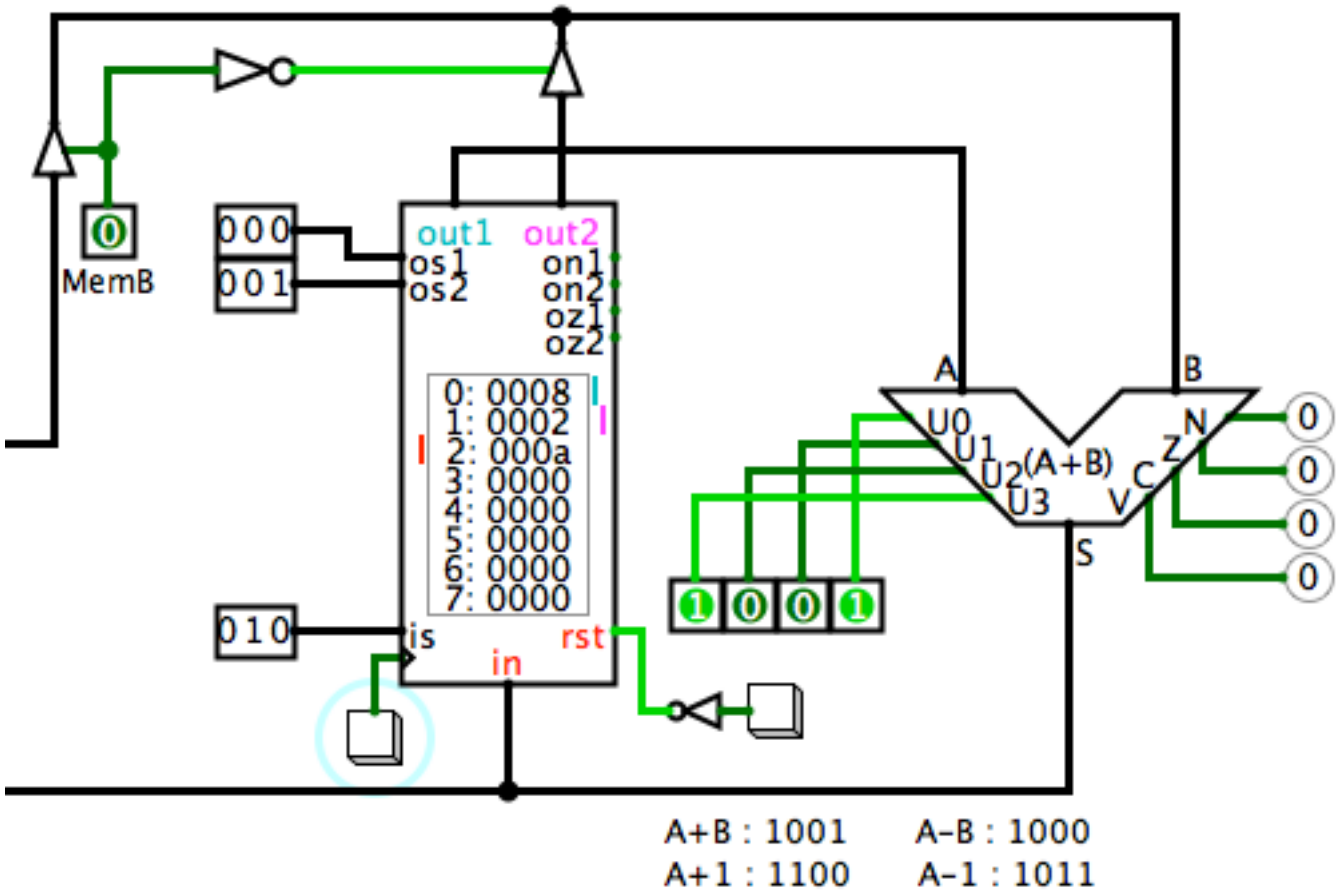


Addition

`add rx, ry, rz`

place la somme de `ry` et `rz` dans le registre `rx`.

Par exemple, si `r0` contient `0x0008` et `r1` contient `0x0002`, l'instruction `add r2, r0, r1` place la valeur `0x000A` dans le registre `r2`.

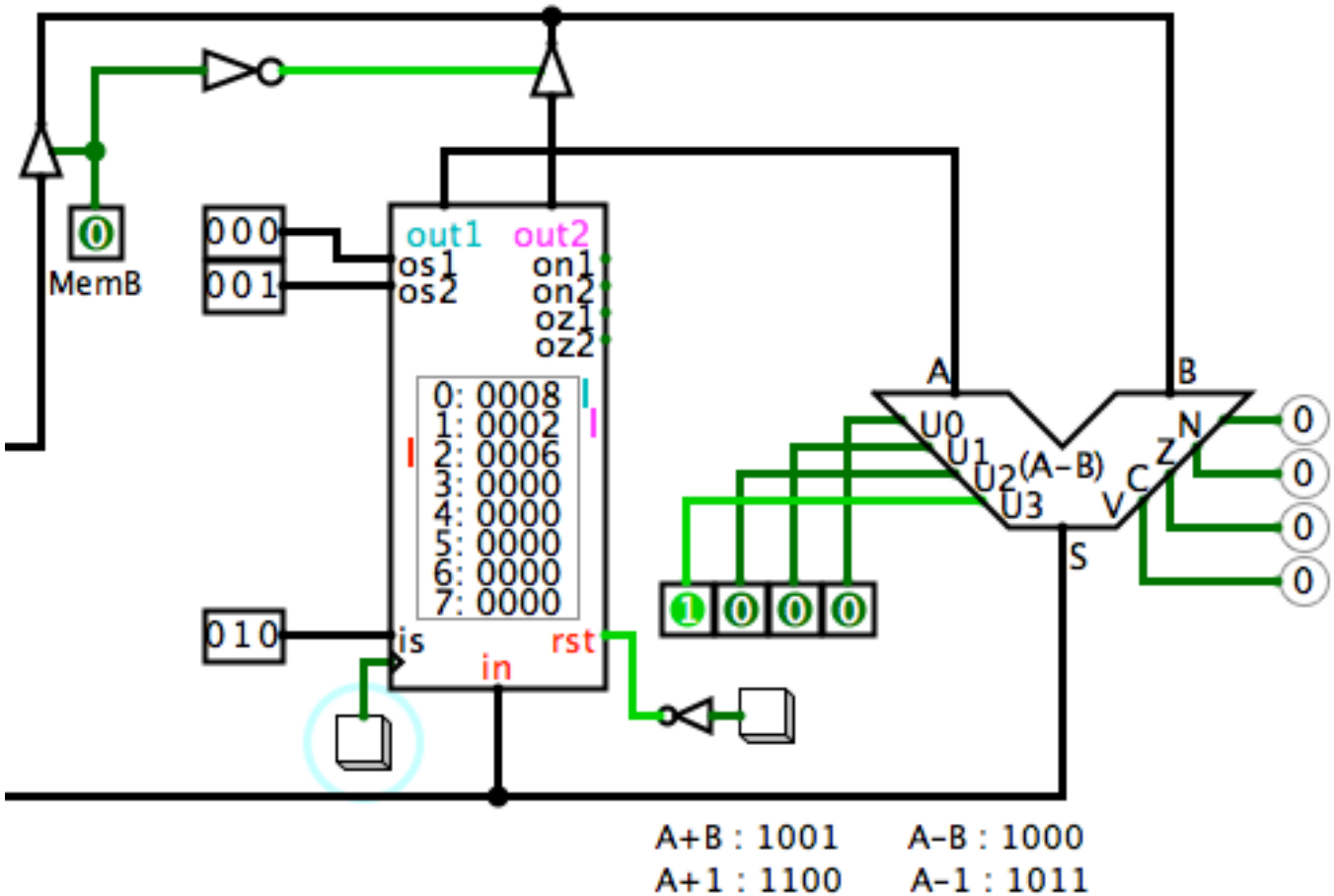


Soustraction

`sub rx, ry, rz`

place la différence de `ry` et `rz` dans le registre `rx`.

Par exemple, si `r0` contient `0x0008` et `r1` contient `0x0002`, l'instruction `add r2, r0, r1` place la valeur `0x0006` dans le registre `r2`.



Tests de condition

- `cmp rx, ry`
  compare les valeurs de `rx` et `ry`

Tests de condition

- `cmp rx, ry`
  compare les valeurs de `rx` et `ry`
- la comparaison se fait par soustraction : `rx - ry`

Tests de condition

- `cmp rx, ry`
  compare les valeurs de `rx` et `ry`
- la comparaison se fait par soustraction : `rx - ry`
- si le résultat est nul, les deux registres sont égaux

Tests de condition

- `cmp rx, ry`
  compare les valeurs de `rx` et `ry`
- la comparaison se fait par soustraction : `rx - ry`
- si le résultat est nul, les deux registres sont égaux
- si le résultat est négatif, `rx < ry`

Tests de condition

- `cmp rx, ry`
  compare les valeurs de `rx` et `ry`
- la comparaison se fait par soustraction : `rx - ry`
- si le résultat est nul, les deux registres sont égaux
- si le résultat est négatif, `rx < ry`
- pour utiliser le résultat de la comparaison, il faut mémoriser la valeur des indicateurs N et Z de l'UAL $\Rightarrow$ registre d'état **SR**.

Branchements

Pour modifier le comportement d'un programme selon le résultat d'une comparaison, on utilise des instructions de branchement :

- `beq adresse`
  saute à l'instruction qui se trouve à `adresse` si la dernière comparaison a donné un résultat nul.

Branchements

Pour modifier le comportement d'un programme selon le résultat d'une comparaison, on utilise des instructions de branchement :

- `beq adresse`
  saute à l'instruction qui se trouve à `adresse` si la dernière comparaison a donné un résultat nul.
- `beq` = *Branch if EQual*
  le saut se fait si les deux valeurs comparées étaient égales...

Branchements

Pour modifier le comportement d'un programme selon le résultat d'une comparaison, on utilise des instructions de branchement :

- `beq adresse`
  saute à l'instruction qui se trouve à `adresse` si la dernière comparaison a donné un résultat nul.
- `beq` = *Branch if EQual*
  le saut se fait si les deux valeurs comparées étaient égales...
- ... donc si l'indicateur Z était à 1

Branchements

Pour modifier le comportement d'un programme selon le résultat d'une comparaison, on utilise des instructions de branchement :

- `beq adresse`
  saute à l'instruction qui se trouve à `adresse` si la dernière comparaison a donné un résultat nul.
- `beq` = *Branch if EQual*
  le saut se fait si les deux valeurs comparées étaient égales...
- ... donc si l'indicateur Z était à 1
- On trouve la valeur de cet indicateur dans le registre d'état

Branchements

Pour modifier le comportement d'un programme selon le résultat d'une comparaison, on utilise des instructions de branchement :

- `beq adresse`
  saute à l'instruction qui se trouve à `adresse` si la dernière comparaison a donné un résultat nul.
- `beq` = *Branch if EQual*
  le saut se fait si les deux valeurs comparées étaient égales...
- ... donc si l'indicateur Z était à 1
- On trouve la valeur de cet indicateur dans le registre d'état
- Exemple :
```
cmp r0, r1
beq 0x1F2A
```

Branchements

- `blt adresse`
  saute à l'instruction qui se trouve à `adresse` si la dernière comparaison a donné un résultat négatif.

Branchements

- `blt adresse`
  saute à l'instruction qui se trouve à `adresse` si la dernière comparaison a donné un résultat négatif.
- `blt` = *Branch if Lower Than*
  le saut se fait si la première valeur était inférieure à la deuxième...

Branchements

- `blt adresse`
  saute à l'instruction qui se trouve à `adresse` si la dernière comparaison a donné un résultat négatif.
- `blt` = *Branch if Lower Than*
  le saut se fait si la première valeur était inférieure à la deuxième...
- ... donc si l'indicateur N était à 1

Branchements

- `blt adresse`
  saute à l'instruction qui se trouve à `adresse` si la dernière comparaison a donné un résultat négatif.
- `blt` = *Branch if Lower Than*
  le saut se fait si la première valeur était inférieure à la deuxième...
- ... donc si l'indicateur N était à 1
- On trouve la valeur de cet indicateur dans le registre d'état

Branchements

- `blt adresse`
  saute à l'instruction qui se trouve à `adresse` si la dernière comparaison a donné un résultat négatif.
- `blt` = *Branch if Lower Than*
  le saut se fait si la première valeur était inférieure à la deuxième...
- ... donc si l'indicateur N était à 1
- On trouve la valeur de cet indicateur dans le registre d'état
- Exemple :
```
cmp r0, r1
blt 0x1F2A
```

Branchements

- `b adresse`

  saute à l'instruction qui se trouve à `adresse`.

Branchements

- `b adresse`
  saute à l'instruction qui se trouve à `adresse`.
- `b` = *Branch*

Branchements

- `b adresse`
  saute à l'instruction qui se trouve à `adresse`.
- `b` = *Branch*
- Exemple :

```
                cmp r0, r1
                beq 0x1F2A
                code pour le cas où r0 ≠ r1
                b 0x1F30
0x1F2A          code pour le cas où r0 = r1
0x1F30          suite du code
```

Branchements

- `b adresse`
  saute à l'instruction qui se trouve à `adresse`.
- `b` = *Branch*
- Exemple :

```
                cmp r0, r1
                beq 0x1F2A
                code pour le cas où r0 ≠ r1
                b 0x1F30
0x1F2A          code pour le cas où r0 = r1
0x1F30          suite du code
```

- Équivalent de : `if r0 == r1: ... else: ...`

Branchements

Par défaut, les instructions s'exécutent en séquence :

- quand une instruction est exécutée, on passe à celle qui suit en mémoire.

Branchements

Par défaut, les instructions s'exécutent en séquence :

- quand une instruction est exécutée, on passe à celle qui suit en mémoire.
- il faut donc toujours connaître l'adresse de la prochaine instruction

Branchements

Par défaut, les instructions s'exécutent en séquence :

- quand une instruction est exécutée, on passe à celle qui suit en mémoire.
- il faut donc toujours connaître l'adresse de la prochaine instruction
- le processeur mémorise cette adresse dans un registre : le compteur ordinal (PC = *program counter*)

Branchements

Par défaut, les instructions s'exécutent en séquence :

- quand une instruction est exécutée, on passe à celle qui suit en mémoire.
- il faut donc toujours connaître l'adresse de la prochaine instruction
- le processeur mémorise cette adresse dans un registre : le compteur ordinal (PC = *program counter*)
- lors de l'exécution d'une instruction, le PC est incrémenté pour contenir l'adresse de l'instruction suivante

Branchements

Par défaut, les instructions s'exécutent en séquence :

- quand une instruction est exécutée, on passe à celle qui suit en mémoire.
- il faut donc toujours connaître l'adresse de la prochaine instruction
- le processeur mémorise cette adresse dans un registre : le compteur ordinal (PC = *program counter*)
- lors de l'exécution d'une instruction, le PC est incrémenté pour contenir l'adresse de l'instruction suivante
- lors d'un branchement, on charge l'adresse à laquelle il faut sauter dans le PC.

Récapitulons

Instructions

`ldr rx, adresse`	`rx` ← `Mem[adresse]`
`str rz, adresse`	`Mem[adresse]` ← `rz`
`mov rx, ry`	`rx` ← `ry`
`add rx, ry, rz`	`rx` ← `ry` + `rz`
`sub rx, ry, rz`	`rx` ← `ry` - `rz`
`cmp rx, ry`	SR ← N et Z selon `rx` - `ry`
`beq adresse`	`PC` ← `adresse` si Z vaut 1 dans SR
`blt adresse`	`PC` ← `adresse` si N vaut 1 dans SR
`b adresse`	`PC` ← `adresse`

Registres

- Registre d'adresse mémoire RADM.
- Registre d'instruction RI = code de l'instruction à exécuter.
- Registre d'état SR = résultat de `cmp`.
- Compteur ordinal PC = adresse de la prochaine instruction.

Modes d'adressage

Un mode d'adressage est une manière d'indiquer la valeur d'un opérande.

- adressage par registre

  L'opérande est contenu dans un *registre* : `mov r0, r1`

Modes d'adressage

Un mode d'adressage est une manière d'indiquer
la valeur d'un opérande.

- adressage par registre

  L'opérande est contenu dans un *registre* : `mov r0, r1`

- adressage direct

  L'opérande se trouve en mémoire à une adresse
  qui est *directement* indiquée : `ldr r0, 0x25FC`

Modes d'adressage

Un mode d'adressage est une manière d'indiquer
la valeur d'un opérande.

- adressage par registre

  L'opérande est contenu dans un *registre* : `mov r0, r1`
- adressage direct

  L'opérande se trouve en mémoire à une adresse
  qui est *directement* indiquée : `ldr r0, 0x25FC`
- Autre exemple d'adressage direct : `beq 0xAB2C`

Modes d'adressage

Un mode d'adressage est une manière d'indiquer
la valeur d'un opérande.

- adressage par registre
  L'opérande est contenu dans un *registre* : `mov r0, r1`
- adressage direct
  L'opérande se trouve en mémoire à une adresse
  qui est *directement* indiquée : `ldr r0, 0x25FC`
- Autre exemple d'adressage direct : `beq 0xAB2C`
- adressage immédiat
  L'opérande est donné *immédiatement* (sans passer par un registre
  ou une adresse) : `add r0, r0, #1`

Modes d'adressage

Un mode d'adressage est une manière d'indiquer
la valeur d'un opérande.

- adressage par registre
  L'opérande est contenu dans un *registre* : `mov r0, r1`
- adressage direct
  L'opérande se trouve en mémoire à une adresse
  qui est *directement* indiquée : `ldr r0, 0x25FC`
- Autre exemple d'adressage direct : `beq 0xAB2C`
- adressage immédiat
  L'opérande est donné *immédiatement* (sans passer par un registre
  ou une adresse) : `add r0, r0, #1`
- Autre exemple d'adressage immédiat : `cmp r1, #2`

Modes d'adressage

Un mode d'adressage est une manière d'indiquer
la valeur d'un opérande.

- adressage par registre
  L'opérande est contenu dans un *registre* : `mov r0, r1`
- adressage direct
  L'opérande se trouve en mémoire à une adresse
  qui est *directement* indiquée : `ldr r0, 0x25FC`
- Autre exemple d'adressage direct : `beq 0xAB2C`
- adressage immédiat
  L'opérande est donné *immédiatement* (sans passer par un registre
  ou une adresse) : `add r0, r0, #1`
- Autre exemple d'adressage immédiat : `cmp r1, #2`
- adressage direct par registre
  L'opérande se trouve en mémoire à une adresse contenue
  dans un registre : ldr r0,[r2] ou beq r6

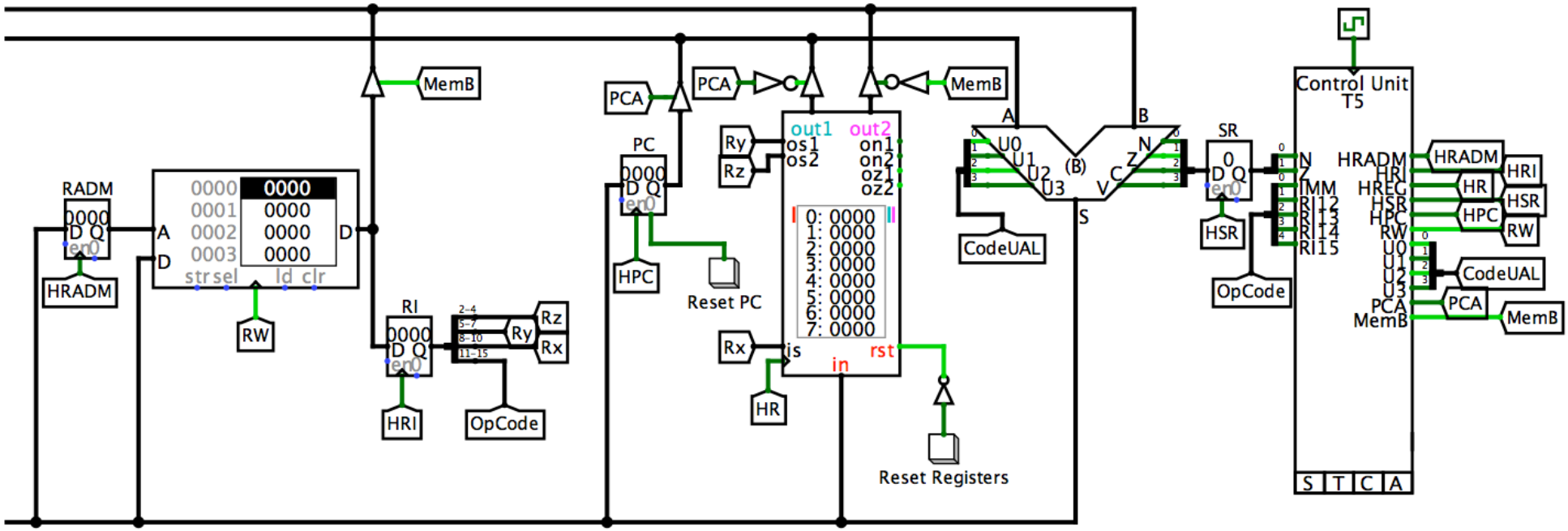
Format des instructions

Instruction	**code**	**imm**	**rx**	**ry**	**rz**	
	4 bits	1 bit	3 bits	3 bits	3 bits	2 bits
ldr rx,adresse	0000	1	rx			
	adresse					
str rz,adresse	0001	1			rz	
	adresse					
mov rx,ry	0010	0	rx	ry		
mov rx,#val	0010	1	rx			
	val					
add rx,ry,rz	0011	0	rx	ry	rz	
add rx,ry,#val	0011	1	rx	ry		
	val					
sub rx,ry,rz	0100	0	rx	ry	rz	
sub rx,ry,#val	0100	1	rx	ry		
	val					

Format des instructions

Instruction	**code**	**imm**	**rx**	**ry**	**rz**	
	4 bits	1 bit	3 bits	3 bits	3 bits	2 bits
cmp ry,rz	0101	0		ry	rz	
cmp ry,#val	0101	1		ry		
	val					
beq rz	0110	0			rz	
beq adresse	0110	1				
	adresse					
blt rz	0111	0			rz	
blt adresse	0111	1				
	adresse					
b rz	1000	0			rz	
b adresse	1000	1				
	adresse					

Chemin de données



Conception du séquenceur

- Le séquenceur est chargé de piloter les signaux du chemin de données

Conception du séquenceur

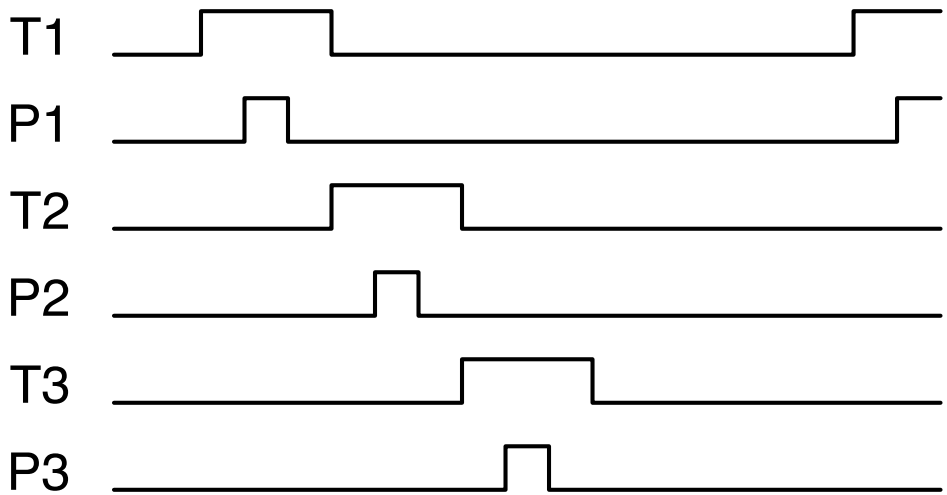
- Le séquenceur est chargé de piloter
  les signaux du chemin de données
- Il reçoit en entrée le code de l'instruction (RI) et les indicateurs (SR)

Conception du séquenceur

- Le séquenceur est chargé de piloter
  les signaux du chemin de données
- Il reçoit en entrée le code de l'instruction (RI) et les indicateurs (SR)
- Il décompose l'exécution d'une instruction en une suite d'étapes

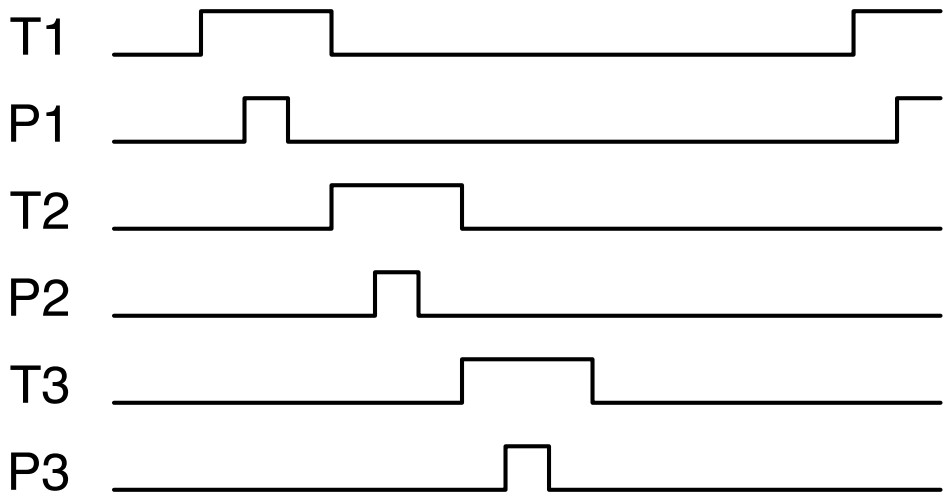
Conception du séquenceur

- Le séquenceur est chargé de piloter
  les signaux du chemin de données
- Il reçoit en entrée le code de l'instruction (RI) et les indicateurs (SR)
- Il décompose l'exécution d'une instruction en une suite d'étapes
- Il dispose pour cela d'une horloge :



Conception du séquenceur

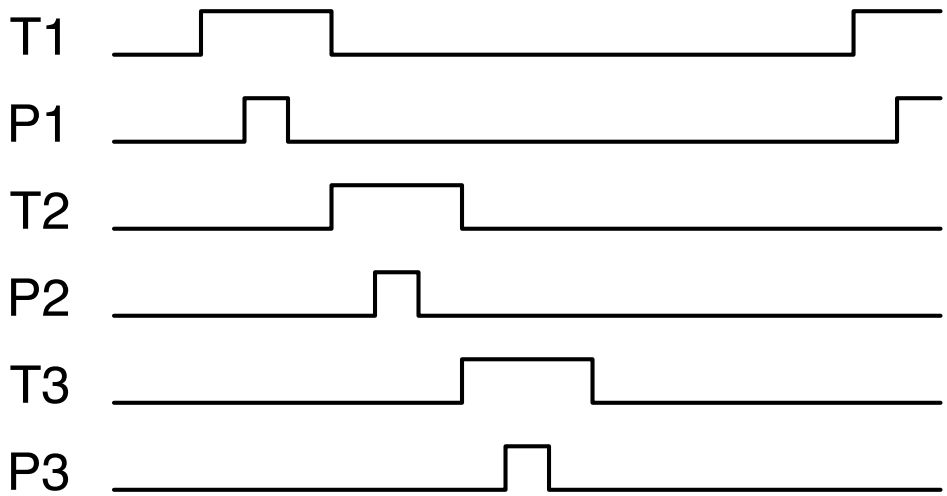
- Le séquenceur est chargé de piloter
  les signaux du chemin de données
- Il reçoit en entrée le code de l'instruction (RI) et les indicateurs (SR)
- Il décompose l'exécution d'une instruction en une suite d'étapes
- Il dispose pour cela d'une horloge :



- T_i configure la combinatoire (codes UAL, connexions)

Conception du séquenceur

- Le séquenceur est chargé de piloter
  les signaux du chemin de données
- Il reçoit en entrée le code de l'instruction (RI) et les indicateurs (SR)
- Il décompose l'exécution d'une instruction en une suite d'étapes
- Il dispose pour cela d'une horloge :



- T_i configure la combinatoire (codes UAL, connexions)
- P_i permet de charger les registres et d'écrire en mémoire

Conception du séquenceur

- Première étape : le fetch

Conception du séquenceur

- Première étape : le fetch
- Il s'agit d'aller chercher le code de l'instruction en mémoire

Conception du séquenceur

- Première étape : le fetch
- Il s'agit d'aller chercher le code de l'instruction en mémoire
- L'adresse de cette instruction se trouve dans le PC

Conception du séquenceur

- Première étape : le fetch
- Il s'agit d'aller chercher le code de l'instruction en mémoire
- L'adresse de cette instruction se trouve dans le PC
- Il faut placer cette adresse dans le RADM

Conception du séquenceur

- Première étape : le fetch
- Il s'agit d'aller chercher le code de l'instruction en mémoire
- L'adresse de cette instruction se trouve dans le PC
- Il faut placer cette adresse dans le RADM
- Le code de l'instruction apparaît en sortie de la mémoire

Conception du séquenceur

- Première étape : le fetch
- Il s'agit d'aller chercher le code de l'instruction en mémoire
- L'adresse de cette instruction se trouve dans le PC
- Il faut placer cette adresse dans le RADM
- Le code de l'instruction apparaît en sortie de la mémoire
- Il faut le charger dans le RI

Conception du séquenceur

- Première étape : le fetch
- Il s'agit d'aller chercher le code de l'instruction en mémoire
- L'adresse de cette instruction se trouve dans le PC
- Il faut placer cette adresse dans le RADM
- Le code de l'instruction apparaît en sortie de la mémoire
- Il faut le charger dans le RI
- Il faut incrémenter le PC

Conception du séquenceur

- Première étape : le fetch
- Il s'agit d'aller chercher le code de l'instruction en mémoire
- L'adresse de cette instruction se trouve dans le PC
- Il faut placer cette adresse dans le RADM
- Le code de l'instruction apparaît en sortie de la mémoire
- Il faut le charger dans le RI
- Il faut incrémenter le PC

Circuit Logisim

Conception du séquenceur

Tableau des temps

On indique pour chaque temps :

- la configuration du chemin de données (signaux T_i)

Conception du séquenceur

Tableau des temps

On indique pour chaque temps :

- la configuration du chemin de données (signaux T_i)
- les registres à charger (signaux P_i)

Conception du séquenceur

Tableau des temps

On indique pour chaque temps :

- la configuration du chemin de données (signaux T_i)
- les registres à charger (signaux P_i)
- pendant le fetch, cela ne dépend pas de l'instruction

Conception du séquenceur

Tableau des temps

On indique pour chaque temps :

- la configuration du chemin de données (signaux T_i)
- les registres à charger (signaux P_i)
- pendant le fetch, cela ne dépend pas de l'instruction
- pour les temps suivants, il faut tenir compte du code de l'instruction

Conception du séquenceur

Tableau des temps

On indique pour chaque temps :

- la configuration du chemin de données (signaux T_i)
- les registres à charger (signaux P_i)
- pendant le fetch, cela ne dépend pas de l'instruction
- pour les temps suivants, il faut tenir compte du code de l'instruction

On détermine ainsi le nombre de temps nécessaires
à l'exécution des instructions

Conception du séquenceur

Tableau des temps

On indique pour chaque temps :

- la configuration du chemin de données (signaux T_i)
- les registres à charger (signaux P_i)
- pendant le fetch, cela ne dépend pas de l'instruction
- pour les temps suivants, il faut tenir compte du code de l'instruction

On détermine ainsi le nombre de temps nécessaires
à l'exécution des instructions

Tableau des temps

Conception du séquenceur

Équations du séquenceur

À partir du tableau des temps :

- on détermine quand les signaux doivent être actifs

Conception du séquenceur

Équations du séquenceur

À partir du tableau des temps :

- on détermine quand les signaux doivent être actifs
- on en déduit les équations des signaux

Conception du séquenceur

Équations du séquenceur

À partir du tableau des temps :

- on détermine quand les signaux doivent être actifs
- on en déduit les équations des signaux

```
# Registre
HRADM = P1+P2+P4*(LDR+STR)

# UAL
U0 =
T4*IMM+T5*(LDR+STR+BEQ+BLT+B+(MOV*IMM)+ADD)

# Connexions
PCA = T1+T2+T3*IMM
```

Conception du séquenceur

Équations du séquenceur

À partir du tableau des temps :

- on détermine quand les signaux doivent être actifs
- on en déduit les équations des signaux

```
# Registre
HRADM = P1+P2+P4*(LDR+STR)

# UAL
U0 =
T4*IMM+T5*(LDR+STR+BEQ+BLT+B+(MOV*IMM)+ADD)

# Connexions
PCA = T1+T2+T3*IMM
```

À faire lors du BE n° 1

Suite...

Code