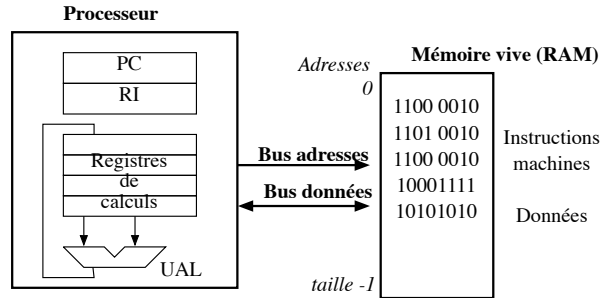


LE PROCESSEUR

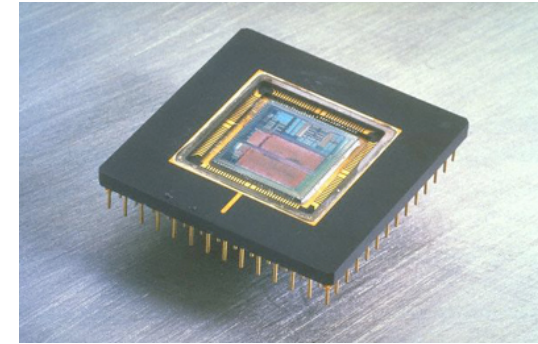
Introduction

- Exécution d'instructions machines
- Instructions et données des programmes se trouvent dans une mémoire (Architecture de Von Neuman - 1948)



- Machine algorithmique particulière

LE PROCESSEUR

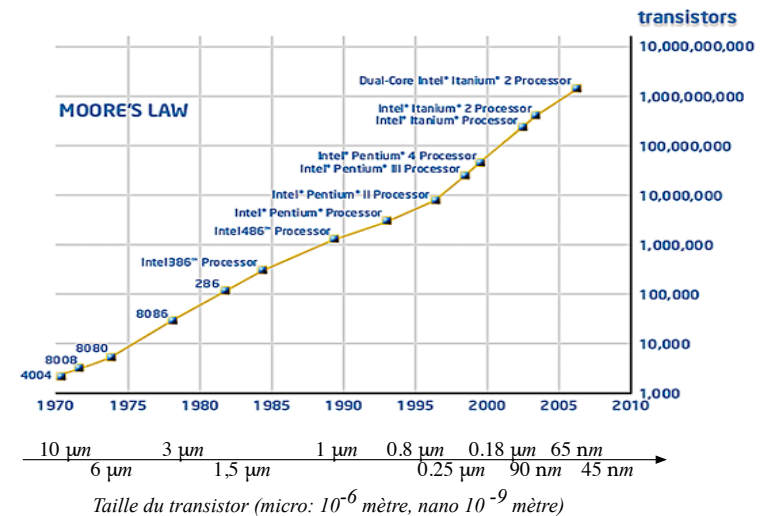


Brève historique

- Processeurs dans les micro-ordinateurs

Année	Nom (fabriquant)	Bus données/adr	Horloge	Ordinateur
1975-80	6502 (MOS technologie)	8 /16 bits	1 Mhz	Atari-Apple II-Commodore 64
1979	8088 (Intel)	8/16 bits	5 Mhz	PC
1980	8086(Intel)	16/16 bits	5 Mhz	PC
1980-90	68000(Motorola)	16/24 bits	8 puis 16 Mhz	MAC
1990	80386 puis 80486 (Intel)	32/32 bits	25 puis 50 Mhz	PC
1995	PowerPC (Apple-IBM-Motorola)	32/64 bits	50 puis 100	MAC
1995	Pentium (Intel)	32/64	60	PC
2000	PowerPC/Pentium	32/36	300	MAC/PC
2006	Core/core duo (Intel)	32/36	1 à 3 GigaHz	MAC et PC
2010	Core i7 (Intel)	64/64	3 à 4 GigaHz	MAC et PC

Evolution processeur (Source Intel)



Architecture générale

- **CISC: Complexe Instruction Set Computer**
 - Jeu d'instructions complexe pour "aider" les compilateurs
 - Nombreux modes d'adressage
 - Instructions de tailles différentes
 - Instructions de durées différentes
 - Jusqu'en 1995
- **RISC: Reduce Instruction Set Computer**
 - Jeu d'instructions restreint
 - Permet de "pipeliner" l'exécution des instructions
 - Plusieurs instructions s'exécute en même temps
 - Chacune est dans une phase différente à un moment donné
 - Exemple: ARM : 4 phases
 - Instructions de taille unique
 - Instructions de durée unique (pas tout à fait vrai pour load/store)
 - A partir du PowerPC et du Pentium

Algorithme du processeur

- Soit *Mem* le tableau représentant la mémoire
- Soit *PC* la variable contenant l'adresse de l'instruction en cours d'exécution
- Soit *RI* une variable contenant l'instruction machine à exécuter
- Le jeu d'instruction du processeur est composé de *N* instructions
- **PC=0** ; /*il faut bien commencer quelque part, la première instruction doit être à l'adresse 0 en mémoire*/
- **Tant que vrai faire**
 - **RI=Mem[PC]** ; /*Une instruction : 1 mot mémoire, variable suivant les processeurs*/
 - **PC=PC +1** ; /*On passe à l'instruction suivante, cas particulier pour les branchements PC = adresse de branchement*/
 - **Suivant RI :**
 - **Instruction1 : exécuter instruction1**
 - **Instruction2 : exécuter instruction2**
 -
 - **InstructionN : exécuter instructionN**

• Définition des caractéristiques du processeur

- Définition de la taille des données et des adresses(Bus)
- Définition du jeu d'instructions : opérations possibles, registres, mode d'adressage, instructions spéciales pour langage haut niveau (saut à sous programme...)
- Choix du codage des instructions (taille des instructions)

• Exemple :

- ARM : 32 bits d'adresse, données 32 ; instructions sur 32 bits
- 68000 : adresse sur 24 ou 32 bits, données sur 8 ou 16 bits, instructions sur 8,16, 24 32,64 bits

Etude sur un processeur "Ecole"

- Taille des données et des adresses: 1 mot (par exemple 8 bits)
- -> Taille maximale de la mémoire 256 octets
- Instructions sur 2 mots
- Un seul registre de calcul appelé ACC (Accumulateur)
- Jeu d'instructions :
 - Load #Vi ! Acc= Vi (Valeur immédiate)
 - St Adresse ! Mem[Adresse]=Acc
 - Jmp Ad ! PC=Ad, difference avec le ARM : déplacement
 - Add Ad ! Acc =Acc+ Mem[Ad]

Codage des instructions

- Format et codage des instructions
- Mot1, Mot2
- Mot1 contient le code opération
 - 4 valeurs (en décimal): *Load* : 0, *St* : 1, *Jmp* : 2, *Add* : 3
- Mot2 contient
 - soit la valeur immédiate (*Load*),
 - soit l'adresse (*St*, *Jmp*, *Add*)

Exemple de programme en assembleur « Ecole »

-Adresse	Contenu	
0	0	Load # 3
1	3	
2	3	Etiqu1 : Add [5]
3	5	
4	2	Jmp Etiqu1
5	2	

Algorithme détaillé du processeur Ecole

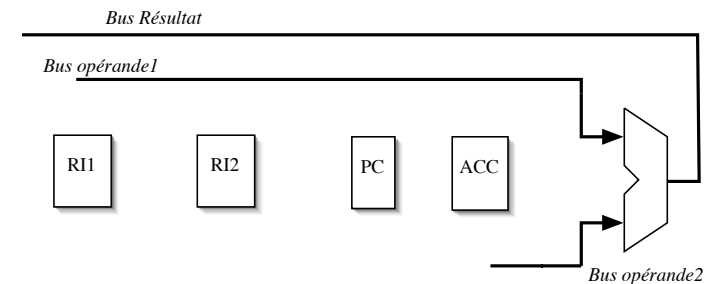
- PC=0 ;
- Tant que vrai faire
 - RI1=Mem[PC] ;
 - PC=PC +1 ;
 - RI2=Mem[PC] ; /*Instruction sur 2 mots → 2 registres instr*/
 - PC=PC +1 ;
 - Suivant RI1 :
 - 0 : ACC=RI2 /* Load */
 - 1 : Mem[RI2]=ACC /* Store */
 - 2 : PC=RI2 /* Jump */
 - 3 : ACC=ACC+ Mem[RI2] /* Add */

Architecture PC/PO

- **Registres** : PC, RI1, RI2, ACC
- **Operations** :
 - +1,
 - Addition,
 - Calcul des conditions RI1= 0, 1, 2, 3 ?
- **Entrées sorties** : bus données (E/S), Bus adresse (S)
- On suppose que la lecture/écriture en mémoire peut se passer en 1 cycle d'horloge du processeur.
- Poignée de main simplifiée pour les E/S, plus besoins de signaux EntréePrise et SortiePrise.

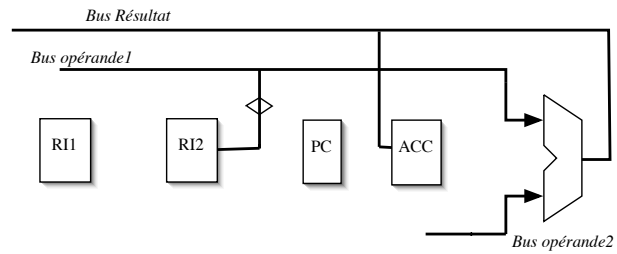
La partie opérative

PO "standard"



Load : $ACC=RI2$

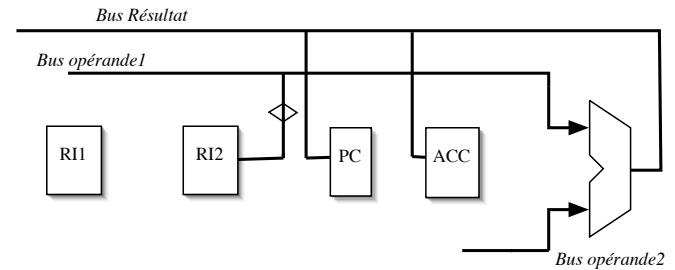
RI2 → Opérande 1,
Bus Resultat → ACC



Opération de l'UAL : RES=Opérande 1

Jump: $PC=RI2$

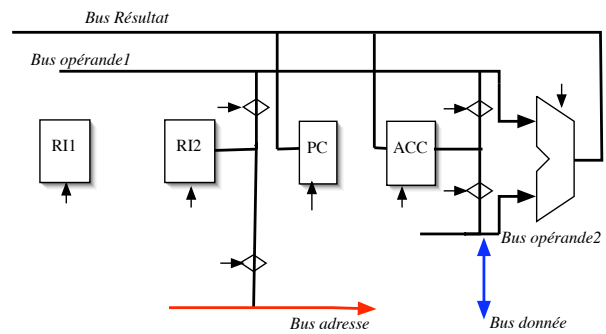
RI2 → Opérande 1
Bus Resultat → PC



Opération de l'UAL : RES=Opérande 1

Add: $ACC=ACC + Mem[RI2]$

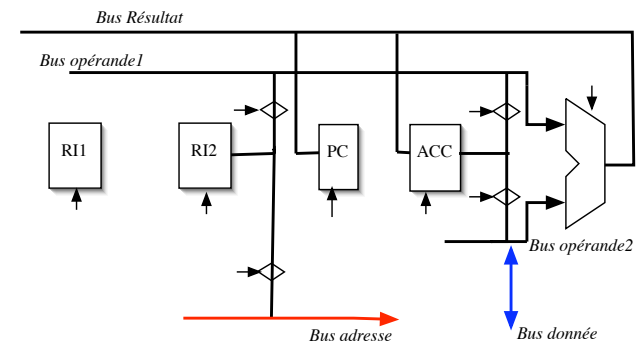
RI2 → Bus Adresse,
Bus Donnée → opérante 2
ACC → opérante 1
Bus Resultat → ACC



Opération de l'UAL : RES=Opérande 1 + Opérande 2

Store: $Mem[RI2]=ACC$

RI2 → Bus Adresse,
ACC → Bus Donnée



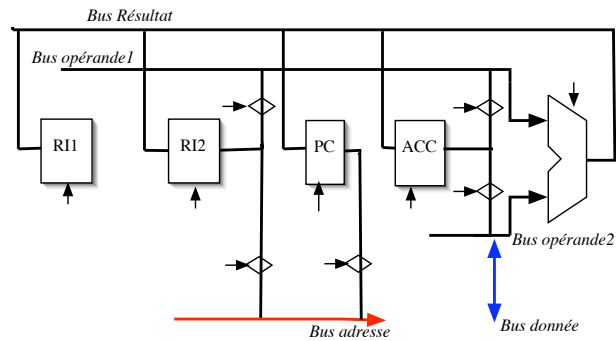
Opération de l'UAL : Indifférent

RI1=Mem[PC], RI2=Mem[PC]

PC → Bus Adresse,

Bus Donnée → Opérande2

Bus Résultat → RI1 ou RI2

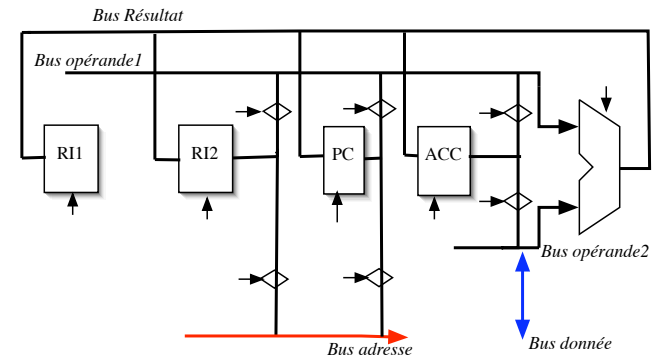


Opération de l'UAL : Res=Opérande 2

PC= PC+1

PC → Opérande1,

Bus Resultat → PC

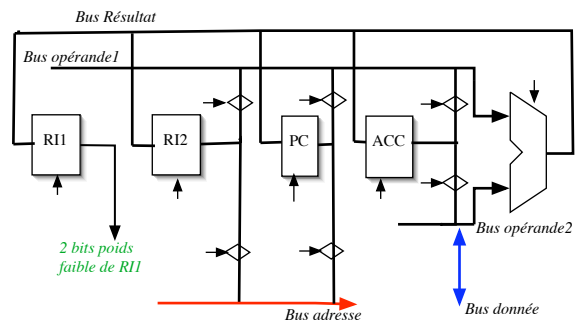


Opération de l'UAL : Res=Opérande 1 + 1

SUIVANT RI1:

Pris directement dans RI1 , inutile de passer dans l'UAL (1 état supplémentaire dans la PC)

00, 01, 10 et 11 : Bit0RI1 / Bit1RI1



Opération de l'UAL : Indifférent

La partie opérative

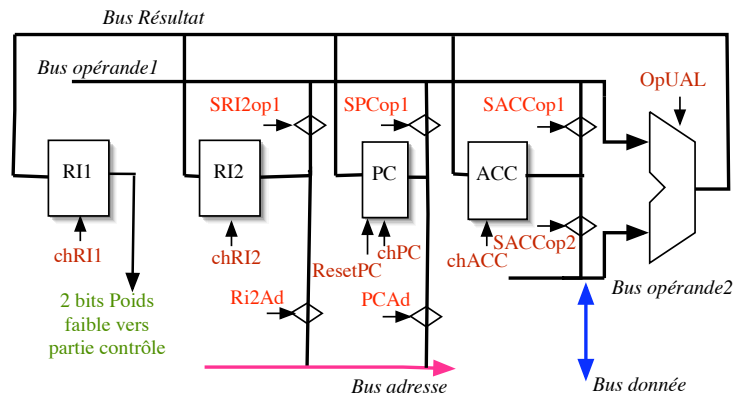
•En résumé 4 opérations dans l'UAL :

- Res=op1+op2
- Res=op1+1
- Res=op1
- Res=op2

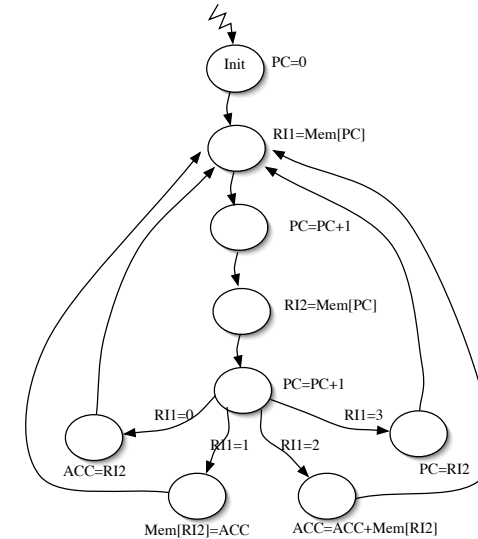
•Il faut rajouter les signaux de contrôle

- Commandes des sélections des bus opérande 1 et 2
- Liaison Bus Interne/externe (Adresse et donnée)
- OpUAL
- Chargement/initialisation registres

La partie opérative avec les signaux de contrôle



La partie contrôle



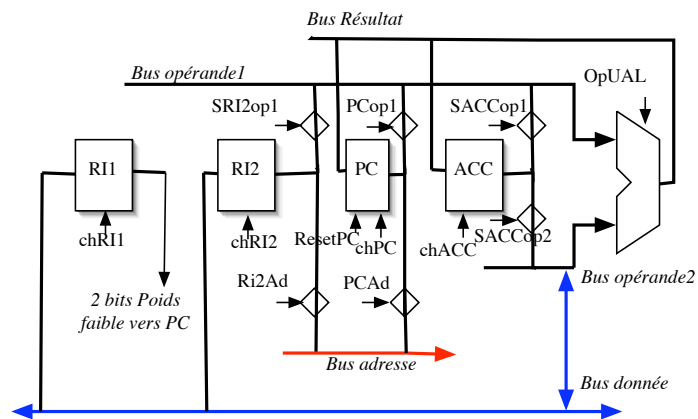
Exemples: Signaux de contrôle

- Signaux mentionnés à 1
- Etat $ACC=ACC+Mem[RI2]$
 - SACCop1, (ACC \rightarrow BusOpérande1)
 - Op2Don, Ri2Ad, ReadMemoire (Mem[RI2] \rightarrow BusOpérande2)
 - OpUAL: code pour l'addition (BusOpérande1+ BusOpérande \rightarrow BusRésultat)
 - chACC (BusRésultat \rightarrow ACC)
- Etat: Mem[RI2]=ACC
 - SACCop2, Op2Don, (ACC \rightarrow Busdonnée)
 - Ri2Ad, WriteMemoire (BusDonnée \rightarrow Mem[RI2])

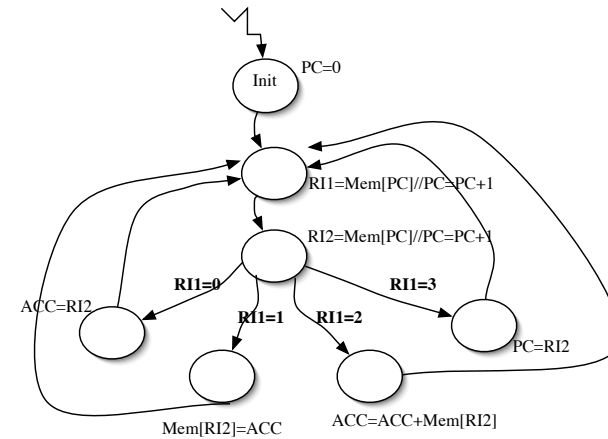
Améliorations

- Parallélisation de calcul
 - RI2=Mem[PC] // PC=PC+1
 - RI1=Mem[PC] // PC=PC+1
 - On gagne 2 périodes d'horloge
- Il faut rajouter dans la PO :
 - Le bus donnée doit arriver à l'entrée de RI1 et RI2 sans passer par l'UAL
 - Du coup, le bus résultat ne doit plus arriver à l'entrée de RI1 et RI2
 - On n'a plus besoin de l'opération $Res=op2$ dans l'UAL

La partie opérative bis



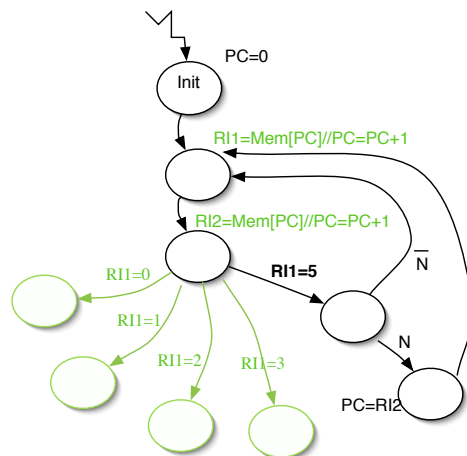
La partie contrôle bis



Extension du jeu d'instructions

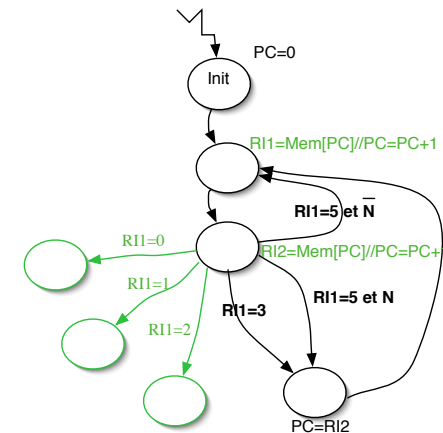
- Un branchement conditionnel suivant le flag N (branchement si négatif en complément à 2)

- BrN adresse
- Code 5



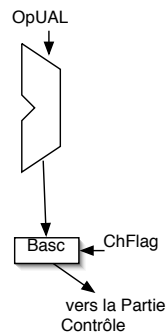
Amélioration

- Un état inutile et un état commun avec l'instruction *jmp*



Modification de la PO

- **Flag N donné par l'UAL**
- **Mémorisation du flag N dans une bascule (commande ChFlag)**
 - Par exemple lors d'instructions de calcul (AddS du ARM), chargement de la bascule dans l'état du calcul du AddS
- **Sortie de la bascule vers la partie contrôle**

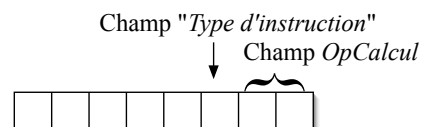


Extension du jeu d'instructions Instructions de calcul

- On augmente le nombre d'opérations de calcul possible : add, sub, and, or ...
- On peut choisir un code quelconque pour ces instructions
- Un état par opérations de calcul après l'éclatement
- On peut essayer de simplifier l'automate (au détriment de la PO) en se servant du code de l'instruction comme entrée des commandes de l'UAL. Il n'y a plus qu'un état pour ces 4 opérations.
- Attention il faut rajouter un signal de commande supplémentaire permettant d'utiliser ou non cette fonctionnalité

Instructions de calcul

- **Exemple :**
 - Instructions: *jmp, st, load, add, sub, or, and*
 - Changement du codage des instructions:
 - Champ *OpCalcul* dans l'instruction
 - 1 bit pour déterminer le type d'instruction (calcul ou autres)
 - Choix des codes des instructions (en binaire)
 - add 000
 - sub 001
 - or 010
 - and 011
 - jmp 100
 - st 101
 - load 110

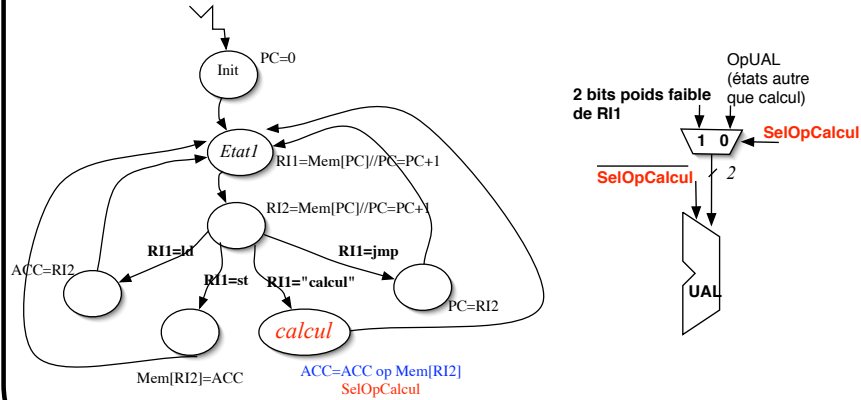


Instructions de calcul

- **UAL conçu en fonction du choix des codes des instructions :**
 - Opération à effectuer dans l'UAL (sur 3 bits):
 - Res= op1+op2 000
 - Res= op1-op2 001
 - Res= op1 or op2 010
 - Res= op1 and op2 011
 - Res= op1+1 100
 - Res= op1 101

Extension du jeu d'instructions

- Le calcul effectué dans l'UAL est défini
 - Soit directement par les 2 bits de poids faible de RI1,
 - Soit par les 2 bits envoyés par la PC; par exemple pour Etat1 : opUAL= 100 (Incr)

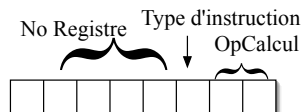


Augmentation du nombre de registres de calcul

- On veut avoir N registres Reg_j de calcul au lieu d'un seul
 - Toute les instructions avec ces registres:
 - store Reg_j, Adresse
 - add Reg_j, Adresse
 - load Reg_j, Vimmediate
 - Si on ajoute un état par instruction possible, le nombre d'états devient très grand
 - Même idée que précédemment on n'utilise qu'un seul état, le choix du registre est déterminé directement dans la PO
 - On ajoute dans l'instruction un champ « Numéro de registre »
 - On sélectionne le registre à envoyer sur les bus opérands grâce à ce champ et à un décodeur (idem pour le choix du registre à charger)

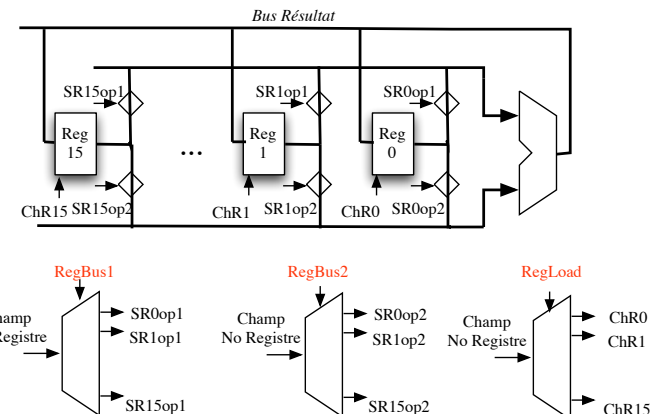
Augmentation du nombre de registres

- Changement du codage des instructions
 - 16 registres: Reg0 à Reg15
 - Champ "No Registre" (4 bits), Champ "Type intructions", Champ *OpCalcul*
- store R_j : NoRegistre, 1, 00 (indifférent)
- add, sub, or, and R_j : NoRegistre, 0, Opcalcul
- load R_j, Vi : NoRegistre, 1, 00 (indifférent)
- jmp : 0000 (indifférent), 1, 00 (indifférent)



Augmentation du nombre de registres

- Il faut que le chargement et la sélection des registres sur les bus n'est lieu que dans les états où les registres sont concernés: commandes supplémentaires de la PC : **Regload, RegBus1, RegBus2**



Augmentation du nombre de registres

•La partie contrôle

