

Organisation de l'enseignement

- Volume horaire

- ▶ Cours : 3×1 heure 45
- ▶ TD : 3×1 heure 45
- ▶ TP : 6×1 heure 45
- ▶ Contrôle : 1×1 heure 45

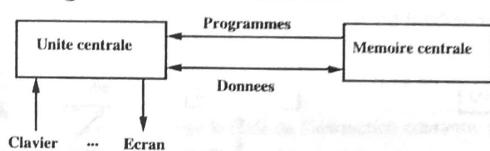
- Objectif

- ▶ Introduction à l'exécution d'un programme écrit dans un langage de haut niveau sur une architecture matérielle
- ▶ Introduction à l'échange d'informations entre un processeur et son environnement
- ▶ Introduction aux caractéristiques des processeurs actuels

Un ordinateur, c'est quoi ?

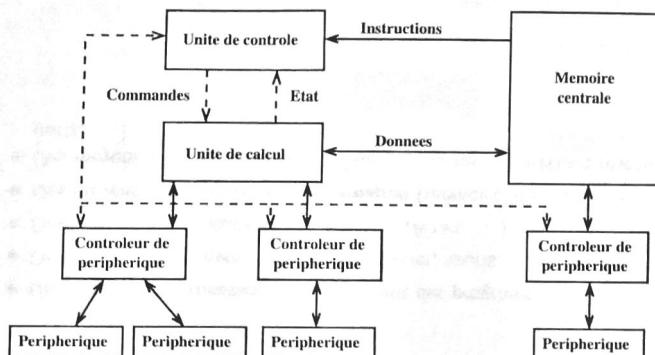
- Un ou plusieurs processeurs qui exécutent des programmes,
- Des moyens pour envoyer des ordres (clavier, souris, ...),
- Des moyens pour récupérer des résultats (écran, ...),
- Des moyens pour stocker de l'information (mémoire, disques, ...)
- Des moyens pour dialoguer avec d'autres dispositifs (interface réseau, ports, ...)

Organisation générale d'un ordinateur



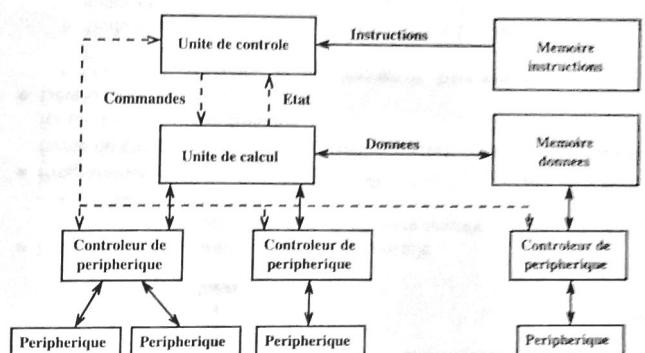
- Exécution de programmes par l'unité centrale
 - ▶ Lecture et écriture de données en mémoire centrale
 - ▶ Interactions avec l'extérieur
- Programmes et données stockées et véhiculées logiquement sous la forme de chiffres binaires ou bits (binary digits), physiquement sous la forme de signaux électroniques
- Développement d'un programme
 - ▶ Ecriture du programme dans un langage de "haut niveau" (e.g. Ada, C, ...)
 - ▶ Traduction du programme en langage machine (instructions plus rudimentaires)
 - ▶ Exécution du programme en langage machine

Modèle de Von Neuman



- Une mémoire commune aux données et aux programmes

Modèle de Harvard



- Deux mémoires séparées pour les données et les programmes

Principe général de fonctionnement d'un processeur

• Unité centrale du processeur

- ▶ Comprend l'unité de contrôle et l'unité de calcul
- ▶ S'occupe de l'interprétation et de l'exécution des programmes
 - ★ Unité de calcul = Unité Arithmétique et Logique + zone de stockage de données temporaires (registres)
 - ★ Unité de contrôle = envoi des ordres à l'unité de calcul pour l'exécution des instructions du programme ⇒ interprétation de chaque instruction

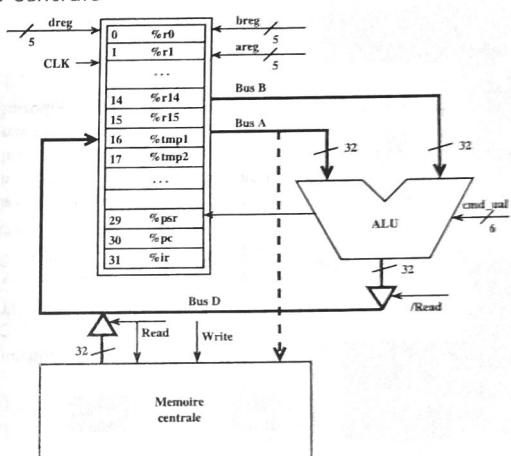
• Mémoire centrale

- ▶ Modèle de Von Neuman : les instructions et les données sont dans la même mémoire ⇒ architecture plus simple, pas d'accès en parallèle
- ▶ Modèle de Harvard : les instructions et les données sont dans deux mémoires distinctes (intérêt : accès simultané aux deux mémoires)

• Accès aux périphériques via des contrôleurs de périphériques

- ▶ Connexion physique du périphérique
- ▶ Synchronisation du périphérique avec l'unité centrale par un protocole assurant que toute donnée échangée est reçue une et une seule fois

L'unité centrale



Principe général de l'exécution d'une instruction

- Adresse mémoire de l'instruction courante dans le Compteur Ordinal (registre PC : Program Counter)
- Code de l'instruction courante dans le registre instructions (registre IR : Instruction Register)
- Etapes pour l'exécution d'une instruction
 - Chargement du code de l'instruction courante dans le registre IR

$IR \leftarrow \text{Mémoire}[PC]$

- Décodage de l'instruction
- Lecture éventuelle des opérandes
- Exécution de l'opération
- Sauvegarde éventuelle du résultat

Petit exemple introductif : interprétation des instructions

$PC \leftarrow 0; /*$ Le programme démarre à l'adresse 0 */

TantQue vrai Faire

```
Selon MEM[PC]
 00002 : /* Instruction clr */
    ACC ← 0;
    PC ← PC + 1;
 00102 : /* Instruction ld */
    ACC ← MEM[PC+1];
    PC ← PC + 2;
 00112 : /* Instruction st */
    MEM[MEM[PC+1]] ← ACC;
    PC ← PC + 2;
 01012 : /* Instruction add */
    ACC ← ACC + MEM[PC+1];
    PC ← PC + 2;
 01002 : /* Instruction jmp */
    PC ← MEM[PC+1];
FinSelon;
FinTantQue;
```

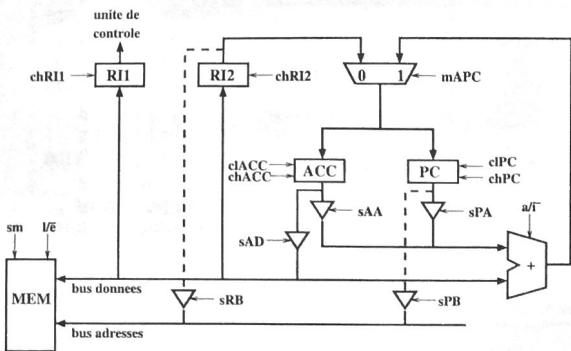
Petit exemple introductif

- Mise en œuvre d'un processeur simpliste
 - Une mémoire de 16 mots de 4 bits
 - Un registre de données ACC sur 4 bits
 - Un jeu d'instructions très réduit
- Exemple de programme
 - clr | Mise à 0 du registre ACC
 - ld #v | Charge de la valeur v dans ACC
 - st a | Copie du contenu de ACC en mémoire à l'adresse a
 - add a | ACC ← ACC + contenu du mot mémoire d'adresse a
 - jmp a | a est l'adresse mémoire de l'instruction suivante
- Codage des instructions
 - clr 0000 ||| add a 0101 a₃a₂a₁a₀
 - ld #v 0010 v₃v₂v₁v₀ ||| jmp a 0100 a₃a₂a₁a₀
 - st a 0011 a₃a₂a₁a₀

Petit exemple introductif : unité de calcul

- Principe général de fonctionnement
 - Adresse de l'instruction courante dans un registre PC
 - Code de l'instruction courante (4 ou 8 bits) dans un ou deux registres RI1, RI2
 - L'unité de calcul charge le code de l'instruction courante, puis exécute l'opération correspondante
- Ressources de l'unité de calcul
 - Un registre PC sur 4 bits
 - Deux registres RI1 et RI2 chacun sur 4 bits
 - Un registre ACC sur 4 bits
 - Une unité arithmétique capable de faire des additions et des incrémentations
- Mécanisme pour l'échange d'informations avec la mémoire
 - Choix de l'opération à effectuer ⇒ deux signaux sm et I/ē
 - * sm = 0 : pas d'opération
 - * sm = 1 et I/ē = 0 : écriture en mémoire
 - * sm = 1 et I/ē = 1 : lecture de la mémoire
 - Donnée lue ou écrite ⇒ un bus bidirectionnel
 - Adresse de lecture ou d'écriture ⇒ un bus monodirectionnel

Petit exemple introductif : unité de calcul



Petit exemple introductif : traitement des instructions

• Instruction *st a*

- ▶ Chargement du code de l'instruction dans le registre d'instruction
 $RI1 \leftarrow \text{MEM[PC]} : sPB = 1, I/\bar{e} = 1, sm = 1, chRI1 = 1$
- ▶ Passage au mot suivant du programme
 $PC \leftarrow PC + 1 : sPA = 1, mAPC = 1, chPC = 1$
- ▶ Chargement de *v* dans le registre d'instruction
 $RI2 \leftarrow \text{MEM[PC]} : sPB = 1, I/\bar{e} = 1, sm = 1, chRI2 = 1$
- ▶ Passage au mot suivant du programme
 $PC \leftarrow PC + 1 : sPA = 1, mAPC = 1, chPC = 1$
- ▶ Copie du contenu de *ACC* en mémoire à l'adresse *a*
 $\text{MEM}[RI2] \leftarrow ACC : sRB = 1, sAD = 1, sm = 1$

• Instruction *add a* (début)

- ▶ Chargement du code de l'instruction dans le registre d'instruction
 $RI1 \leftarrow \text{MEM[PC]} : sPB = 1, I/\bar{e} = 1, sm = 1, chRI1 = 1$
- ▶ Passage au mot suivant du programme
 $PC \leftarrow PC + 1 : sPA = 1, mAPC = 1, chPC = 1$
- ▶ Chargement de *v* dans le registre d'instruction
 $RI2 \leftarrow \text{MEM[PC]} : sPB = 1, I/\bar{e} = 1, sm = 1, chRI2 = 1$

Petit exemple introductif : traitement des instructions

• Instruction *clr*

- ▶ Chargement du code de l'instruction dans le registre d'instruction
 $RI1 \leftarrow \text{MEM[PC]} : sPB = 1, I/\bar{e} = 1, sm = 1, chRI1 = 1$
- ▶ Passage au mot suivant du programme
 $PC \leftarrow PC + 1 : sPA = 1, mAPC = 1, chPC = 1$
- ▶ Mise à 0 du registre *ACC*
 $ACC \leftarrow 0 : chACC = 1$

• Instruction *ld v*

- ▶ Chargement du code de l'instruction dans le registre d'instruction
 $RI1 \leftarrow \text{MEM[PC]} : sPB = 1, I/\bar{e} = 1, sm = 1, chRI1 = 1$
- ▶ Passage au mot suivant du programme
 $PC \leftarrow PC + 1 : sPA = 1, mAPC = 1, chPC = 1$
- ▶ Chargement de *v* dans le registre d'instruction
 $RI2 \leftarrow \text{MEM[PC]} : sPB = 1, I/\bar{e} = 1, sm = 1, chRI2 = 1$
- ▶ Passage au mot suivant du programme
 $PC \leftarrow PC + 1 : sPA = 1, mAPC = 1, chPC = 1$
- ▶ Copie de *v* dans le registre *ACC*
 $ACC \leftarrow RI2 : chACC = 1$

Petit exemple introductif : traitement des instructions

• Instruction *add a* (fin)

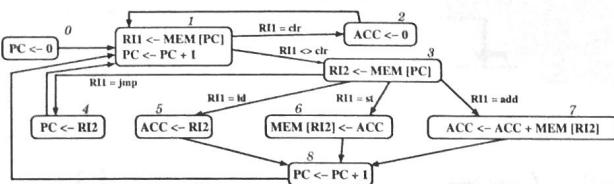
- ▶ Passage au mot suivant du programme
 $PC \leftarrow PC + 1 : sPA = 1, mAPC = 1, chPC = 1$
- ▶ Ajout de *MEM[a]* au contenu de *ACC*
 $ACC \leftarrow ACC + \text{MEM}[RI2] : sAA = 1, sRB = 1, I/\bar{e} = 1, sm = 1, a/\bar{i} = 1, mAPC = 1, chACC = 1$

• Instruction *jmp a*

- ▶ Chargement du code de l'instruction dans le registre d'instruction
 $RI1 \leftarrow \text{MEM[PC]} : sPB = 1, I/\bar{e} = 1, sm = 1, chRI1 = 1$
- ▶ Passage au mot suivant du programme
 $PC \leftarrow PC + 1 : sPA = 1, mAPC = 1, chPC = 1$
- ▶ Chargement de *v* dans le registre d'instruction
 $RI2 \leftarrow \text{MEM[PC]} : sPB = 1, I/\bar{e} = 1, sm = 1, chRI2 = 1$
- ▶ Copie de *a* dans le registre *PC*
 $PC \leftarrow RI2 : chPC = 1$

Petit exemple introductif : unité de commande

- Initialisation de PC à 0, puis répétition à l'infini du traitement d'une instruction



- Construction du circuit correspondant en utilisant les mêmes principes que pour le circuit de calcul du $n^{\text{ème}}$ terme de la suite de Fibonacci
- Un circuit possible
 - Une bascule D pour chaque état de l'unité de commande
 - Génération des commandes en fonction de l'état actif
 - Exemple : chACC = 1 pour les états 5 et 7

Langages de programmation des processeurs

- Le processeur ne comprend que le binaire \Rightarrow toute instruction s'exprime sous la forme d'une séquence de 0 et de 1
- L'écriture d'un programme en binaire est difficile (peu lisible \Rightarrow risque élevé d'erreur)
- Association d'un nom à chaque instruction pour faciliter le travail
- Une instruction du processeur est une opération élémentaire de l'unité de calcul \Rightarrow un programme nécessite de très nombreuses instructions et manque souvent de structure
- Utilisation de langages structurés de plus haut niveau (une instruction du langage correspond à une séquence d'instructions du processeur) : Java, Pascal, C, ...
- Compilateurs pour traduire les programmes en langage de haut niveau en programmes en langage du processeur
- Un compilateur différent pour chaque processeur

Caractéristiques principales du CRAPS

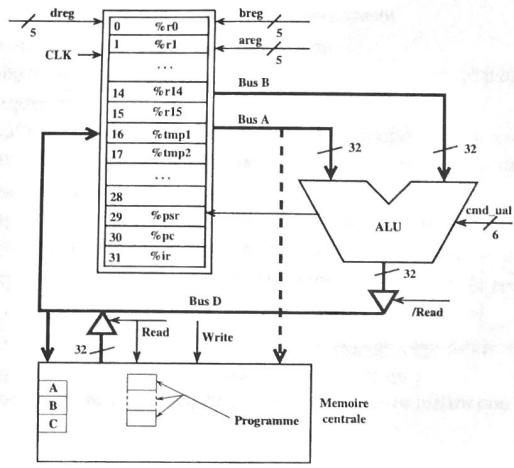
- Version simplifiée du SPARC version 8
- Processeur 32 bits
- Mémoire de 2^{32} octets
- Architecture de type *big-endian*
- Chaque instruction est codée sur un mot de 32 bits
- Processeur RISC (Reduced Instruction Set Computer) : jeu d'instructions réduit
- Machine de type *load/store* : un petit nombre d'instructions spécialisées pour le transfert des données entre la mémoire centrale et les registres
- 16 registres généraux, un compteur ordinal, un registre instruction, un registre d'états, ...

Exemple de programme

```
if (A <= B)
    C = A + B;
else
    C = A - B;
```

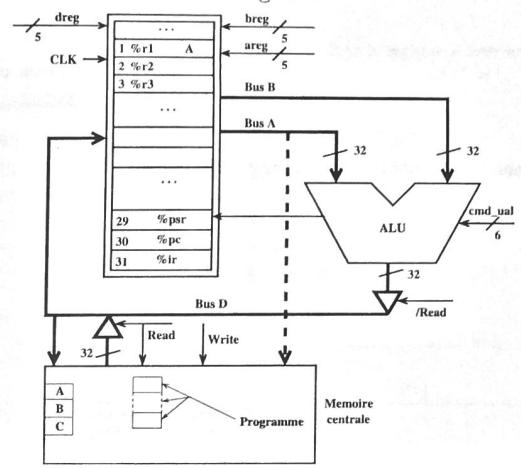
- Addition/soustraction de deux variables avec résultat dans une troisième variable
- Comparaison avec saut éventuel
- Saut inconditionnel

Point de départ : A et B en mémoire



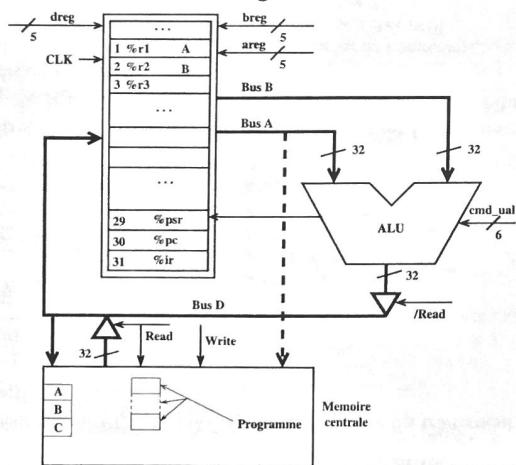
Jean-Luc Scherbang - ENSEEIHT - Dpt. SdN Architecture des ordinateurs 1

Chargement de A dans un registre



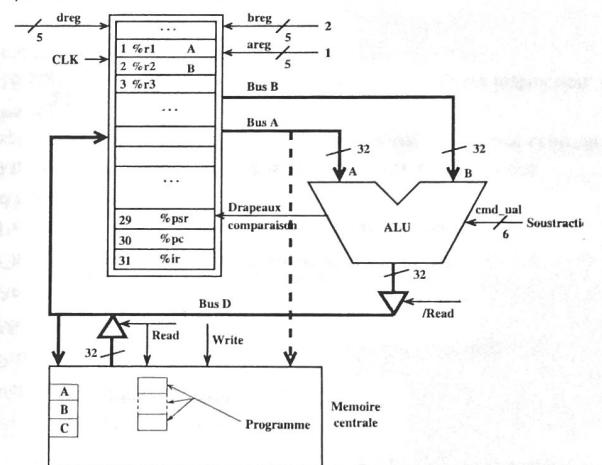
Jean-Luc Scharbarg - ENSEEIHT - Dpt. SdN Architecture des ordinateurs I Février 2020 22 / 58

Chargement de B dans un registre



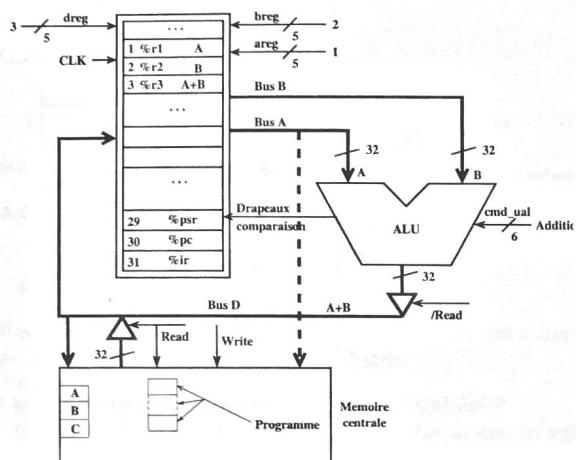
[View Details](#) | [Edit](#) | [Delete](#)

Comparaison de A et de B



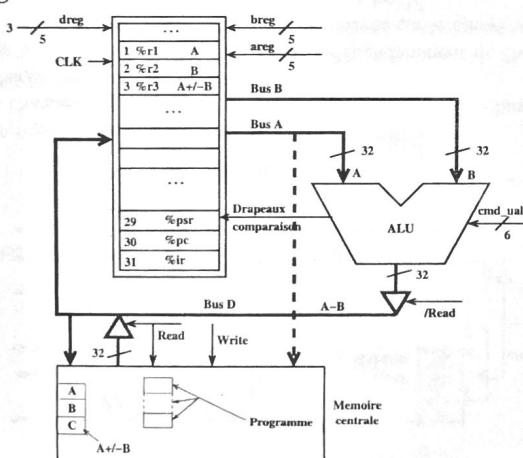
Février 2020 22 / 58

Calcul de $A + B$



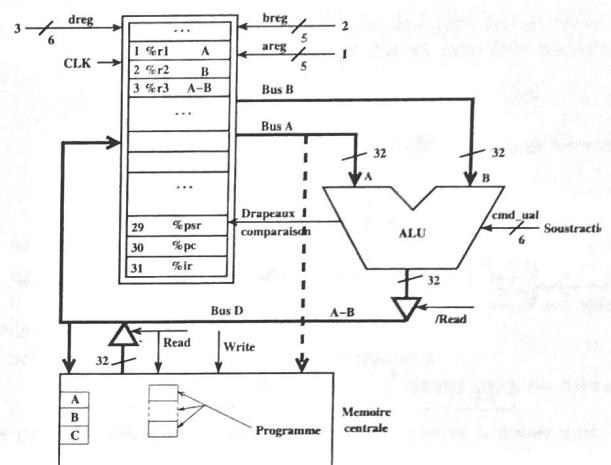
Jean-Luc Scharbarg - ENSEEIHT - Dpt. SdN - Architecture des ordinateurs I

Sauvegarde du résultat en mémoire



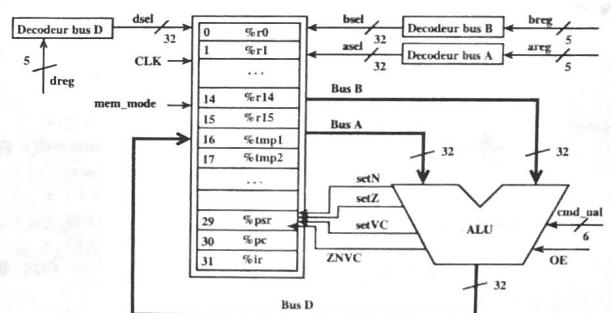
Février 2020 27 / 58

Calcul de $A - B$



Jean-Luc Scharbarg - ENSEEIHT - Dpt. SdN - Architecture des ordinateurs I

Architecture pour l'addition

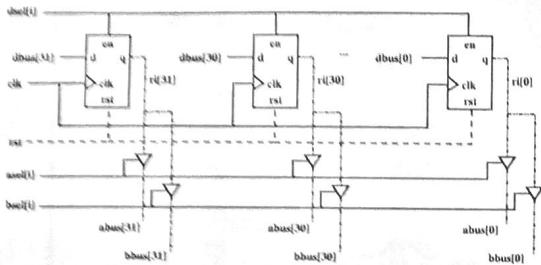


- 16 registres utilisateurs (%r0 à %r15), 1 registre d'état (%psr)
- 3 bus pour les échanges de données entre l'UAL et les registres

Jean-Luc Scharbarg - ENSEEIHT - Dpt. SdN - Architecture des ordinateurs I

Février 2020 28 / 58

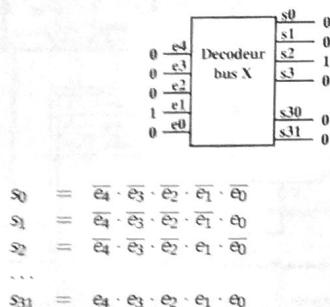
Le composant de mémorisation : le registre %ri



- Le registre contient la valeur $\%ri[31] \dots \%ri[0]$
 - A chaque front montant de l'horloge clk , la valeur sur le bus D est chargée dans le registre ssi $dsel[i] = 1$
 - rst permet la remise à 0 du registre, indépendamment de l'horloge
 - La valeur contenue dans le registre est placée sur le bus A et/ou le bus B en fonction des valeurs de $asel[i]$ et $bsel[i]$

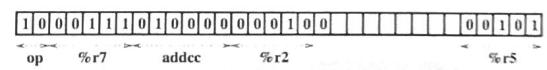
Principe des décodeurs associés aux bus

- Activation de la ligne de sortie s_i avec $i = e_5e_4e_3e_2e_1e_0$



Déroulement de l'instruction `addcc %r2 %r5 %r7`

- Code de l'instruction



❶ Mise en place des entrées pour l'opération d'addition

- Le contenu du registre $\%r2$ est placé sur le bus A ($asel[2] = 1$)
 - Le contenu du registre $\%r5$ est placé sur le bus B ($bsel[5] = 1$)
 - La commande d'addition est envoyée à l'unité arithmétique et logique ($cmd_alu = 010000$)

❷ Opération d'addition

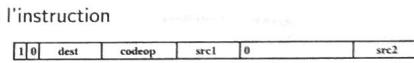
- ▶ L'unité arithmétique et logique effectue l'addition des deux valeurs et place le résultat sur le bus D
 - ▶ L'unité arithmétique et logique génère les indicateurs ZNVC

❸ Sauvegarde du résultat

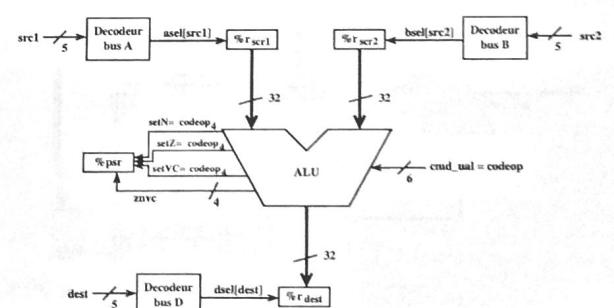
- Le résultat de l'addition est mémorisé dans `%r7` (`dsel[7] = 1`)
 - Les valeurs de ZNVC sont mémorisées dans le registre `%psr` (`setN = 1, setZ = 1, setVC = 1`)

Jean-Luc Scharbarg - ENSEEIHT - Dpt. SdN Architecture des ordinateurs 1

Synthèse d'une instr

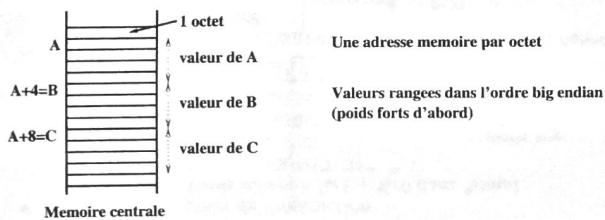


3. Mise d'ensemble du déroulement de l'instruction



Opération d'addition avec opérandes et résultat en mémoire

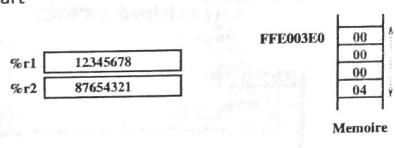
- Opération à exécuter : $C \leftarrow A + B$



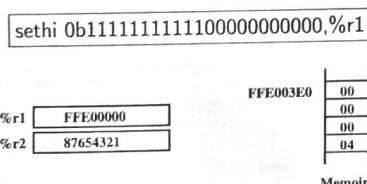
- Pas d'instructions arithmétiques ou logiques avec des opérandes situés en mémoire \Rightarrow il faut passer par des registres
- Trois étapes pour l'opération d'addition
 - Chargement des opérandes A et B dans des registres
 - Addition du contenu des deux registres avec résultat dans un registre
 - Stockage du résultat en mémoire

Illustration de la séquence d'instructions mise en œuvre

- Exemple considéré : copie de la valeur située à l'adresse mémoire FFE003E0₁₆ dans le registre %r2
- Point de départ



- Copie des poids forts de l'adresse dans %r1



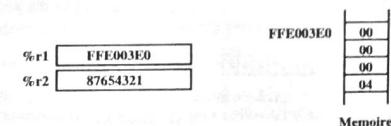
Copie d'une valeur de la mémoire vers un registre

- Informations nécessaires pour l'exécution de l'opération de copie
 - Code stipulant qu'il s'agit d'une opération de copie de la mémoire vers un registre
 - Le code opération d'une instruction CRAPS est sur 6 bits
 - Adresse mémoire concernée
 - Espace d'adressage du CRAPS : 2³² octets \Rightarrow une adresse mémoire est sur 32 bits
 - Numéro du registre concerné
 - 32 registres \Rightarrow numéro de registre sur 5 bits
- 43 bits pour coder la copie d'une valeur de la mémoire vers un registre
- Processeur RISC \Rightarrow toute instruction est codée sur 32 bits
- Il n'est pas possible de coder en une instruction la copie d'une valeur de la mémoire vers un registre
- Solution adoptée par le CRAPS : 3 instructions
 - Chargement, en deux étapes, de l'adresse mémoire concernée dans un registre,
 - Utilisation d'un adressage indirect : chargement de la donnée dont l'adresse mémoire se trouve dans le registre

Illustration de la séquence d'instructions mise en œuvre

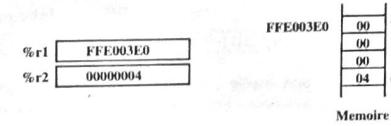
- Copie des poids faibles de l'adresse dans %r1

orcc %r1, 0b1111100000,%r1



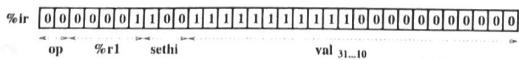
- Chargement de la donnée dont l'adresse mémoire est dans %r1 dans le registre %r2

ld [%r1],%r2



Exécution de l'instruction *sethi val_{31..10}, %r1*

• Codage de l'instruction

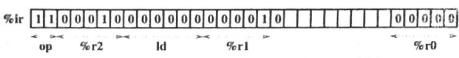


- Etapes de l'exécution de l'instruction

Jean-Luc Scharbarg - ENSEEIHT - Dpt. SdN Architecture des ordinateurs 1 Printemps 2024 37 / 58

Exécution de l'instruction *Id* [%r1], [%r2]

- L'instruction exécutée est : $Id [\%r1 + \%r0], \%r2$
 - Codage de l'instruction



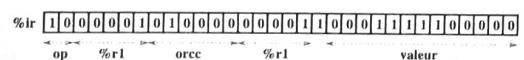
- Etapes de l'exécution de l'instruction

- ▶ Calcul de l'adresse mémoire $\%r1 + \%r0$ dans $\%tmp1$
 - * $\%r1$ sur le bus A ($asel[1] = 1$)
 - * $\%r0$ sur le bus B ($bsel[0] = 1$)
 - * Addition (`cmd.alu 010000`)
 - * Résultat dans $\%tmp1$ ($dsel[16] = 1$)
 $\%tmp1 \quad 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0$
 - ▶ Lecture en mémoire à l'adresse contenue dans $\%tmp1$, la donnée lue est chargée dans $\%r2$
 - * $\%tmp1$ sur le bus A ($asel[16] = 1$)
 - * Ordre de lecture envoyé à la mémoire, La valeur contenue en mémoire à l'adresse présente sur le bus A est placée sur le bus D
 - * Résultat dans $\%r2$ ($dsel[2] = 1$)
 $\%r2 \quad 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0$

Jean-Luc Scherbares - ENSEIGNANT - Doc. SdN - Architectures de systèmes - 1 Février 2020 39 / 59

Exécution de l'instruction `orcc %r1, val` : `0, %r1`

- Codage de l'instruction



- Etapes de l'exécution de l'instruction

- Extension des 13 bits de poids faibles de $\%ir$ sur 32 bits, résultat dans $\%tmp1$
 - $\%ir$ sur le bus A ($asel[31] = 1$)
 - Extension de 13 bits à 32 bits (signext13, **cmd_alu 100000**)
 - Résultat dans $\%tmp1$ ($dsel[16] = 1$)
 - $\%tmp1$

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
 - Addition entre $\%r1$ et $\%tmp1$
 - $\%r1$ sur le bus A ($asel[1] = 1$)
 - $\%tmp1$ sur le bus B ($bsel[16] = 1$)
 - Addition (**cmd_alu 010000**)
 - Résultat dans $\%r1$ ($dsel[1] = 1$)
 - $\%r1$

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

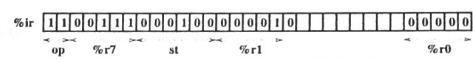
Jean-Luc Scharbarg - ENSEEIHT - Dpt. SdN Architecture des ordinateurs 1 Février 2020 38 / 58

Copie d'une valeur d'un registre vers la mémoire

- Séquence d'instructions mise en œuvre

| | |
|--------------|--------------------------|
| <i>sethi</i> | $val_{31..10}, \%r1$ |
| <i>orcc</i> | $\%r1, val_{9..0}, \%r1$ |
| <i>st</i> | $\%r7, [\%r1]$ |

- Codage de l'instruction *st %r7, [%r1]*



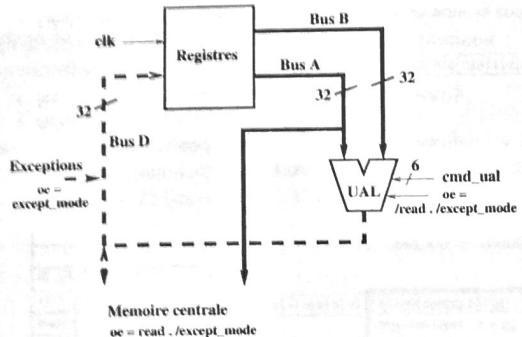
- Etapes de l'exécution de l'instruction

- ▶ Calcul de l'adresse mémoire $\%r1 + \%r0$ dans $\%tmp1$
 - * Identique au calcul mis en œuvre pour l'instruction *ld*
 - ▶ Ecriture en mémoire à l'adresse contenue dans $\%tmp1$, la donnée à écrire est dans $\%r7$
 - * $\%tmp1$ sur le bus A (*asel[16] = 1*)
 - * $\%r7$ sur le bus B (*bsel[7] = 1*)
 - * Copie du bus B sur le bus D (**cmd_alu 101000**)
 - * Ordre d'écriture envoyé à la mémoire, La valeur présente sur le bus D est écrite en mémoire à l'adresse présente sur le bus A

est écrite en mémoire à l'adresse présentee
Jean-Luc Scharbarg ENSEEIHT Doi. SdN Archivé automatiquement

Jean-Luc Scherbares - ENSEIGNANT - Doc. SdN - Architectures de systèmes - 1 Février 2020 39 / 59

Architecture d'ensemble intégrant la mémoire

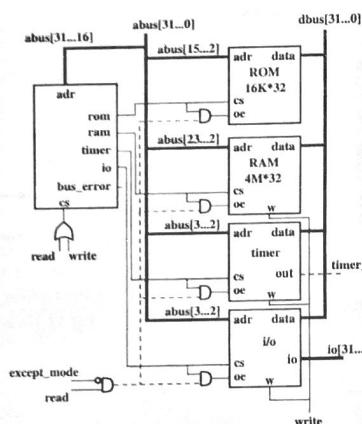


Cartographie de la mémoire

| | |
|-------------------------|--------------------------|
| 0000 0000 ... 0000 FFFF | ROM (64K) |
| 0040 0000 ... 0040 000B | |
| 0060 0000 ... 0060 000F | Timer Entrees/sorties |
| 0100 0000 ... 01FF FFFF | RAM (16 MO) |

- ROM : Read Only Memory, mémoire en lecture seule
- RAM : Random Access Memory, mémoire en lecture/écriture
- Timer : pour la configuration, la mise en route et l'arrêt du timer
- Les entrées/sorties sont vues comme des adresses mémoire
- Accès à des adresses mémoire en dehors des plages définies ⇒ exception *bus_error*

Structure du sous-système mémoire



Instructions de branchement pour les ruptures de séquence

- Structures de contrôle nécessitant des ruptures de séquence

► Instructions conditionnelles

| | |
|-----------------------------------|--|
| Si condition alors traitement1 | Si condition allera sinon traitement1 allera finsi |
| Sinon traitement2 | sinon : traitement2 |
| Finsi | finsi : |

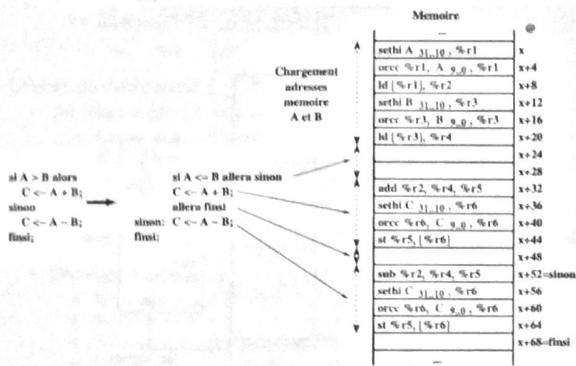
► Iterations

| | | |
|---------------------------------------|---------|--|
| Tantque Condition faire traitement | tq: | Si non condition allera fintq traitement allera tq |
| Fintantque | fintq : | |

- Deux types de ruptures de séquences

- Les branchements inconditionnels : **allera etiq**
- Les branchements conditionnels : **Si condition allera etiq**

Exemple de programme



Codage de l'adresse de l'étiquette associée au branchement

| Mémoire | Adressage | Opération | Opérande | Réultat |
|------------------------|------------|-----------|-------------------|------------|
| ... | x+48 | ba finsi | | |
| sub %r2, %r4, %r5 | x+52=sinon | sub | %r2, %r4, %r5 | |
| sethi C _1_1_0 , %r6 | x+56 | sethi | C _1_1_0 , %r6 | |
| urec %r6, C _9_0 , %r6 | x+60 | urec | %r6, C _9_0 , %r6 | |
| st %r5, [%r6] | x+64 | st | %r5, [%r6] | x+68=finsi |
| ... | | | | |

- Branchement absolu : c'est l'adresse absolue de l'instruction cible du branchement qui est codée dans l'instruction (exemple : x + 68)
 - Adresse de l'étiquette sur 32 bits
 - Ne peut pas être codée dans une instruction CRAPS
- Branchement relatif : c'est le déplacement jusqu'à l'instruction cible du branchement qui est codé dans l'instruction (exemple : +20)
 - Déplacement limité par le nombre de bits utilisé pour le codage
 - C'est la solution utilisée par le CRAPS

Principes des instructions de branchement

- L'instruction de branchement inconditionnel (*ba etiq*) force la mise à jour du registre *%pc*
 - remplace l'incrémentation de *%pc* par l'affectation de *%pc* avec l'adresse de l'étiquette
- L'effet de l'instruction de branchement conditionnel (*ble etiq*) dépend de la valeur de la condition
 - Condition vraie ⇒ affectation de *%pc* avec l'adresse de l'étiquette
 - Condition fausse ⇒ incrémentation de *%pc* (pas de rupture de séquence)
- La condition associée au branchement conditionnel porte sur les indicateurs ZNVC
- Mise en œuvre de l'opération : *si A ≤ B allera sinon*
 - Calcul de $A - B$ pour obtenir les indicateurs ZNVC
 - subcc %r2, %r4, %r0*
 - Branchement à *sinon* si résultat négatif ou nul

Z or (N xor V)

Exécution de l'instruction *ba finsi*

- Codage de l'instruction

%ir

- Etapes de l'exécution de l'instruction

- Extension des 22 bits de *%ir* sur 32 bits, résultat dans *%tmp1*
 - %ir* sur le bus A (*asel[31] = 1*)
 - Extension de 22 bits à 32 bits (*signext22, cmd_alu = 100001*)
 - Résultat dans *%tmp1* (*dsel[16] = 1*)
- Addition entre *%tmp1* et *%pc*, résultat dans *%pc*
 - %tmp1* sur le bus A (*asel[16] = 1*)
 - %pc* sur le bus B (*bsel[30] = 1*)
 - Addition (*cmd_al = 000000*)
 - Résultat dans *%pc* (*dsel[30] = 1*)

Exécution de l'instruction *ble sinon*

- Codage de l'instruction

%ir [0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0]
 op cond op2 disp22

- Instruction sans effet lorsque la condition est fausse

- Etapes de l'exécution de l'instruction lorsque la condition est vraie

- Extension des 22 bits de %ir sur 32 bits, résultat dans %tmp1

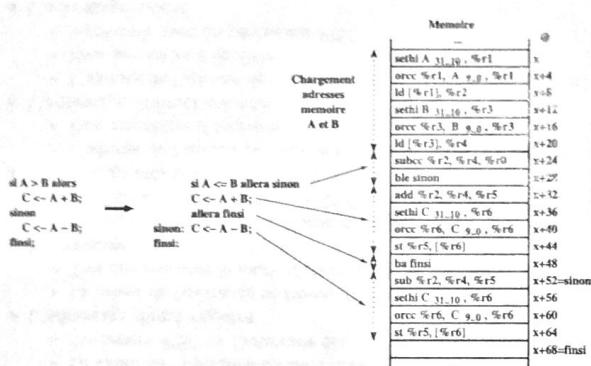
- * %ir sur le bus A (asel[31] = 1)
- * Extension de 22 bits à 32 bits (signext22, cmd.alu = 100001)
- * Résultat dans %tmp1 (dsel[16] = 1)

%tmp1 [0 1 1 0 0 0]

- Addition entre %tmp1 et %pc, résultat dans %pc

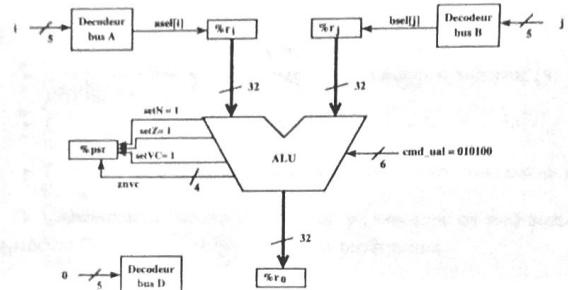
- * %tmp1 sur le bus A (asel[16] = 1)
- * %pc sur le bus B (bsel[30] = 1)
- * Addition (cmd.al = 000000)
- * Résultat dans %pc (dsel[30] = 1)

Le programme complet



Exécution d'une opération de comparaison

- Comparer les valeurs entières contenues dans deux registres %ri et %rj
- Calcul de %ri - %rj avec mise à jour des drapeaux
- On ne garde pas le résultat \Rightarrow on l'envoie dans %r0
- Instruction à exécuter : subcc %ri, %rj, %r0
- Le registre %r0 contient toujours 0 \Rightarrow écriture sans effet



Classes d'instructions d'un processeur

- Instructions arithmétiques et logiques

- Addition, soustraction, incrémentation, décrémentation
- Et logique, ou logique, non logique, ...
- Décalages logiques et arithmétiques, rotations
- Multiplication, division

- Instructions de mouvement

- De la mémoire vers un registre
- D'un registre vers la mémoire
- D'un registre vers un registre

- Instructions de branchements

- Branchements inconditionnels ou conditionnels
- Branchements absous ou relatifs

- Instructions système

Différents modes d'adressage

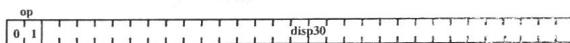
- L'adressage immédiat
 - ▶ La valeur de l'opérande est directement dans l'instruction
 - ▶ Processeur RISC ⇒ l'intervalle des valeurs possibles est limité
- L'adressage direct registre
 - ▶ La valeur de l'opérande se trouve dans un registre
 - ▶ Très certainement le mode d'adressage le plus utilisé
- L'adressage direct mémoire
 - ▶ L'adresse de l'opérande se trouve dans l'instruction
 - ▶ Inutilisable avec un processeur RISC
- L'adressage indirect registre
 - ▶ L'adresse de l'opérande se trouve dans un registre
 - ▶ Peut permettre d'accélérer la manipulation des pointeurs
- L'adressage indirect mémoire
 - ▶ L'adresse de l'opérande se trouve dans l'instruction
 - ▶ Peut permettre d'accélérer la manipulation des pointeurs
 - ▶ Inutilisable avec un processeur RISC
- L'adressage indexé
 - ▶ L'adresse de l'opérande est obtenue après ajout d'un index
 - ▶ Intéressant pour la manipulation des tableaux

Codage des instructions

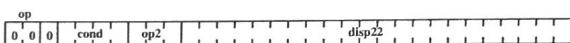
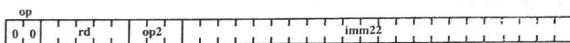
- Doit permettre de déterminer
 - ▶ L'instruction à exécuter
 - ▶ Les opérandes de cette instruction
- Le codage doit faciliter le décodage par le processeur
 - ▶ Les codes opérations sont structurés
 - ▶ Les opérandes sont toujours codés au même endroit
 - ▶ Pour les instructions arithmétiques ou logiques, la commande à envoyer à l'UAL est "en clair"

Structure binaire des groupes d'instructions du CRAPS

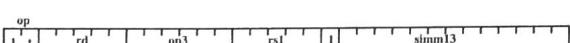
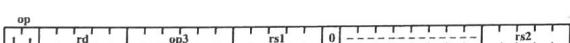
● Format 1 : call



● Format 2 : sethi et branchements



● Format 3 : accès mémoire, instructions arithmétiques

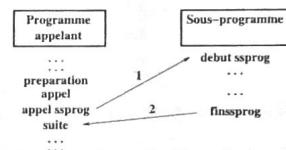


t = 1 : accès mémoire, t = 0 : instruction arithmétique

La mise en œuvre de sous-programmes

● Principe général de l'appel d'un sous-programme

- ▶ Préparation des données nécessaires à l'exécution du sous-programme
⇒ passage des paramètres
- ▶ Branchement à la première instruction du sous-programme ⇒ rupture de séquence
- ▶ Exécution de la séquence d'instructions correspondant au sous-programme
- ▶ A la fin de cette séquence, retour au programme appelant (à l'instruction qui suit l'appel au sous-programme)



La nécessité d'utiliser une pile

- L'utilisation de registres ne permet pas d'imbriquer des appels de sous-programmes
 - Un seul registre pour l'adresse de retour \Rightarrow tout nouvel appel écrase l'adresse de retour précédente
 - Les paramètres d'un sous programme donné sont toujours dans les mêmes registres \Rightarrow Problématique pour mettre en œuvre une fonction récursive
- Un sous-programme utilise (modifie) des registres \Rightarrow le programme appelant devrait connaître ces registres
- La solution : l'utilisation d'une pile
 - Structure permettant de sauvegarder un nombre de données déterminé par la taille de la pile
 - Deux opérations possibles
 - * empiler : sauvegarde d'une donnée en sommet de pile
 - * dépiler : récupération de la donnée en sommet de pile
- La pile est le plus souvent située en mémoire centrale

La mise en œuvre d'une pile

- Choix de l'ordre dans lequel on empile
 - Pile montante : par adresses décroissantes
 - Pile descendante : par adresses croissantes
- Pointeur de sommet de pile : permet de savoir où on si situe dans la pile. Deux solutions
 - Pointeur sur la dernière donnée empilée
 - Pointeur sur le premier emplacement libre
- La pile du CRAPS
 - Pile montante
 - Pointeur de sommet de pile : le registre %r30
 - %r30 pointe sur la dernière donnée empilée
 - Opération empiler : *push %ri*
sub %r30, 4, %r30
st %ri, [%r30]
 - Opération dépiler : *pop %ri*
ld [%r30], %ri
add %r30, 4, %r30