Architecture des ordinateurs

IFIPS Cycle Apprentissage
2005-2006

Cécile Germain-Renaud
cecile.germain@lri.fr

Plan

1. Représentation de l'information
   Comment faire exploser un satellite
2. Architecture Logicielle
   Compiler le langages de haut niveau
3. Micro-Architecture
   Fabriquer une UC avec des circuits très simples
4. Hiérarchie Mémoire
   Les caches sont partout

Contrôle des connaissances

- 2 petits contrôles (30 minutes) par séquence, aux séances 2 et 3

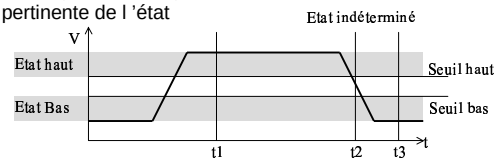
1. Représentation de l'information

Plan

- Définitions
- Représentation des caractères
- Représentation des entiers
- Représentation des réels

L 'information - Définition

- Analogique : grandeur continue
- ou Numérique (= digitale) :
  Information = connaissance d 'un état parmi un nombre fini d 'états possibles
- Support : grandeur physique continue - tension
- Le temps intervient pour obtenir une mesure pertinente de l 'état



Quantité d'information

- 1 bit = quantité d 'information liée à la connaissance d '1 état parmi 2 -> codage par 0 et 1
- Avec n bits, on peut coder 2^n états, donc la quantité d'information contenue dans la connaissance d'1 état parmi *N* est

$$\lceil \mathrm{Log}_2(N) \rceil \text{bits}$$

- Sur 2 bits
  - France 00
  - GB 01
  - Italie 10
  - Espagne 11

Ordres de grandeurs

Kilo	2**10	10**3
Mega	2**20	10**6
Giga	2**30	10**9
Tera	2**40	10**12
Peta	2**50	10**15

Quelle information ?

- Tous les processeurs
  - Nombres
    - Entiers naturels et relatifs
    - Réels
  - Caractères
  - Instructions

Pourquoi pas d 'autres ?

Quel codage?

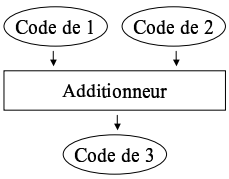
Doit permettre de réaliser des

opérateurs matériels

fonctionnels et rapides.

Opérateur matériel = circuit électronique non programmable

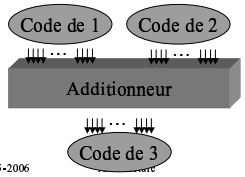
Le plus simple : un additionneur



Codage et opérateurs

Sur un nombre fixe de bits

- caractéristique du processeur et de la technologie des Circuits Intégrés
- typiquement 32 ou 64 bits



Quelle information ?

- Certains processeurs représentent et traitent les vecteurs d 'entiers et de réels
  - Processeurs « multimédia »
  - Console de jeux
- En général, pas les enregistrements (record), listes et autres types complexes

Plan

- Définitions
- Représentation des caractères
- Représentation des entiers
- Représentation des réels

Codage des caractères

- ASCII **American Standard Code for Information Interchange : 7 bits + 1 bit de parité**
  Caractères alphanumériques latins non accentués + caractères spéciaux
- Latin -1 : **8 bits, standard ISO 8859-1. Caractères alphanumériques latins accentués**
- Unicode : **16 bits, caractères d 'alphabets variés**

Plan

- Définitions
- Représentation des caractères
- Représentation des entiers
- Représentation des réels

Plan

- Définitions
- Représentation des caractères
- Représentation des entiers
- Représentation des réels

Codage des entiers naturels

Sur *n* bits, écriture en base 2

Sur 8 bits 10010001 code $1+2^3+2^7$

Sur *n* bits, les nombres représentables sont

$[0…2^n-1]$

Sur 8 bits : [0, 255]

Notations (humaines)

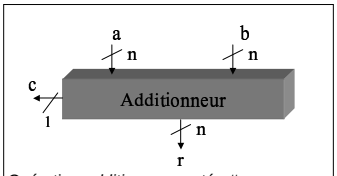
- Binaire : 00101100
  Peu pratique pour le traitement manuel
- Hexadécimale : 1 digit hexa par quartet
  0x2C

La notation hexadécimale ne préjuge pas du contexte d 'interprétation

exemple : ‘ A ’ est codé par 0x41

L 'additionneur

- Additionneur : circuit matériel qui effectue l 'addition naïve des entiers naturels sur *n* bits et tronque éventuellement la retenue



Opération *additionnage* notée #
a, b opérandes; r résultat ; c retenue (Carry)

a	b	r	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

r = a XOR b
c = a ET b

Additionneur 1 bit

L 'additionneur

Le résultat du calcul dans l 'additionneur peut être faux : *retenue*

Sur 3 bits

001 # 001 = 010 Correct : 1 + 1 = 2
001 # 111 = 000 Faux : 1 + 7 ≠ 0

est l'addition modulo 2^n

Addition des naturels

Le résultat du calcul dans l 'additionneur peut être faux

Un programme de test simple :

```
entier_non_signé : val ;
tant que val <> 0 faire
    val = val + 1;
fin tant que
afficher (' Quelle erreur ! ');
```

Le résultat dépend
- du langage de programmation
- du compilateur et des options

Codage des entiers relatifs

Codage en complément à 2 sur n bits
- Si N ≥ 0, N est codé comme un entier naturel
- Si N < 0, N est codé par la représentation en naturel de

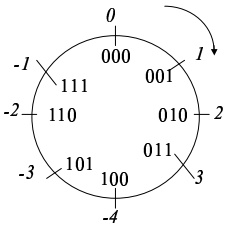
$$2^n + N$$

- Nombres représentables

$$[-2^{n-1} \ldots 2^{n-1} - 1]$$

Codage en complément à 2 sur 3 bits

Nombre	Code
0	000
1	001
2	010
3	011
-1	De 8 -1 = 7, soit 111
-2	De 8 - 2 = 6, soit 110
-3	De 8 – 3 = 5, soit 101
-4	De 8 – 4 = 4, soit 100



Propriétés du codage en complément à 2

1. Le bit de poids fort
   des positifs est 0
   des négatifs est 1
2. La représentation de -1 est le mot où tous les bits sont à 1 : FF (8bits), FFFF (16bits) etc.

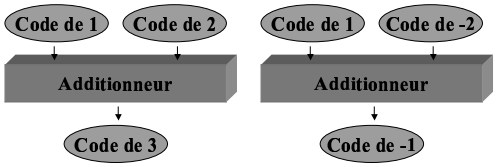
Opérations arithmétiques

L 'opposé d 'un relatif s 'obtient en complémentant bit à bit et en ajoutant 1 sur n bits

Ex. 2 est codé par 010 sur 3 bits. Comment coder –2 ?
Complément bit à bit : 101
1
-2 est donc codé par 110
Dans l'autre sens, 110 -> 001 -> 010

Opérations arithmétiques

Le circuit additionneur peut effectuer aussi l 'addition des relatifs *s 'ils sont codés en complément à 2* : 1 seul circuit



Addition des relatifs

Le résultat du calcul dans l 'additionneur peut être faux : *overflow*

Sur 3 bits :
001 # 001 = 010 correct : 1 + 1 -> 2
001 # 110 = 111 correct : 1 + (-2) = -1
111 # 111 = 110 correct : -1 + -1 = -2
011 # 001 = 100 faux : 3 + 1 n 'est pas codable
110 # 101 = 011 faux :-2 + (-3) n 'est pas codable

Addition des relatifs

On démontre :

L 'additionnage des codes de deux relatifs N et M est égal à N + M si et seulement si N + M est codable

L'additionneur est aussi satisfaisant que possible

L 'additionnage des codes de deux relatifs N et M est égal à N + M si et seulement si

- N et M sont de signes contraires
- Ou bien N et M sont de même signe, et le bit de signe du résultat est égal à la retenue

Facile à tester en matériel

Conclusion

- Avec les codages choisis, l'additionneur peut effectuer l'addition des naturels et des relatifs
- Mais le résultat peut être faux s'il n 'appartient pas à l 'ensemble d'entiers représentables
  - Analyser l 'application et le programme pour garantir que le résultat reste codable
  - Si ce n 'est pas possible, spécifier le comportement sur overflow et utiliser les ressources de l 'environnement pour le détecter et réaliser l 'action spécifiée.

Plan

- Définitions
- Représentation des caractères
- Représentation des entiers
- Représentation des réels

Codage des réels

- Problèmes :
  - Arrondir
  - Représenter équitablement des nombres très grands et très petits
- Principe : représentation normalisée par mantisse et exposant, avec un choix judicieux des valeurs de l 'exposant

Codage IEEE 754 simple précision

E = interprétation en naturel

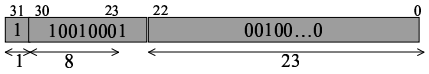
Pour E ≠ 0 et 255,

La valeur codée est : $(-1)^s . 2^{E-127} . 1,f$

Où 1,f doit être interprété en base 2

Soit e = E -127. Alors -126 ≤ e ≤127

Codage IEEE 754 simple précision



Que représente 0xC8900000 ?

1100 1000 1001 0000 …0

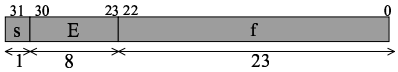
s = 1 ; E = 10010001 = 145 donc e = 18

f = 001 0000 …0 donc m = $1 + 2^{-3}$

Le nombre codé est $-2^{18}(1 + 2^{-3})$

Codage IEEE 754 simple précision

Les cas exceptionnels



Nom	E	f	valeur
Dénormalisé	0	non 0	$(-1)^s.2^{-127}.0,f$
Zéro	0	0	0
Infini	255	non 0	$(-1)^s$Infini
NaN	255	0	NaN

La norme IEEE 754

- Codage simple précision (32 bits), double précision (64 bits), simple étendue et double étendue
- Spécification des arrondis suivant les opérations, de l 'arithmétique étendue; Objectif : reproductibilité

2. Architecture logicielle

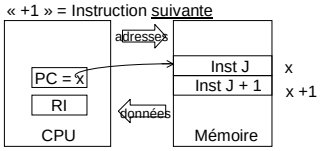
Plan

1. Le modèle de Von-Neumann
2. Format des instructions
3. Instructions arithmétiques
4. Instructions d'accès mémoire
5. Instructions de contrôle
6. Procédures

Le modèle d'ordinateur de Von Neumann

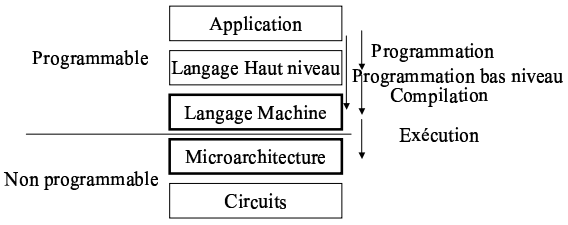
- *Mémoire = Instruction et données*
- *Processeur : Tant que <vrai> faire*
  Lire instruction : RI <- Mem[PC] ; PC <- PC + "1"
  Exécuter instruction : suite d'actions dépendant de l'instruction

RI : Registre Instructions
« +1 » = Instruction suivante



EN MATERIEL

Les niveaux d'un système informatique



Architecture logicielle

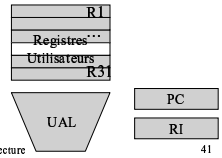
- Le processeur tel que le voit le compilateur, ou un programmeur d'applications spécifiques bas niveau (ex. DSP)
- Spécifié par un langage-machine = jeu d'instructions = "assembleur"
- Peut correspondre à une grande variété de processeurs matériels : x86 du 8086 (1980) au Pentium IV (2000).

Le CPU – Partie opérative

• Contraintes
- en matériel
- calcul uniquement sur les registres
Caractéristique RISC (Reduced Instruction Set Computer) Toutes machines actuelles

• Registres : Organes de stockage à l'intérieur du processeur
Chacun contient 1 mot

• UAL : opérations additionnage, soustraction, division et multiplication par puissances de 2 etc.

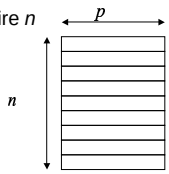


Caractéristiques élémentaires du CPU

- Largeur du chemin de données = taille de l'UAL : 32 ou 64 bits
  DIDE : 32 bits
- Fréquence : inverse du temps de cycle, dépend de
  - Temps de propagation de l'UAL
  - Temps de propagation des signaux dans le CPU
- Chaque registre a la même taille que le chemin de données

Caractéristiques élémentaires mémoire

- Tableau de mots de largeur fixe *p*
  - *Mémoire organisée par octets p* = 8
  - *Mémoire organisée par mots de 16 bits p* = 16
- Accessible chacun par leur adresse
- Taille = nombre de mots mémoire *n*



Un exemple

Instruction d'affectation LHN

```
A, B, C :: entiers
A:= B + C;
```

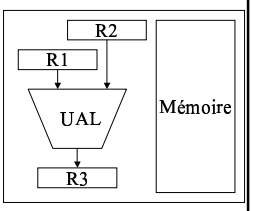
- Comment sont stockées les variables A, B et C ?
- Comment effectuer le calcul ?

Un exemple

Instruction d'affectation LHN

```
A, B, C :: entiers
A:= B + C;
```

Charger B depuis la mémoire vers le CPU
LOAD R1, "adresse de B"
Charger C depuis la mémoire vers le CPU
LOAD R2, "adresse de C"
Effectuer : R3 <- R1 + R2
ADD R3, R2, R1
Stocker le résultat en mémoire
STORE R3, "adresse de C"



Langage de Haut Niveau

- Structures de données : Scalaires, Tableaux, Listes, Produits,…
- Affectations : A := B +C + D
- Contrôle
  - Séquence : Instruction 1 ; Instruction2
  - Boucles
  - Appels de procédures et fonctions

Types de données

- Un type décrit en général : encombrement mémoire et les opérateurs admissibles
- Ici les opérateurs sont matériels, donc les types sont :
  - Entiers signés et non signés : taille du chemin de données
  - Octets : 8 bits
  - Flottants : 32 ou 64 bits

DIDE

- Dans toute la suite, on considère l'architecture logicielle DIDE, qui correspond à une version très simplifiée des processeurs actuels non x86.
- Chemin de données 32 bits
- Mémoire organisée par octets.

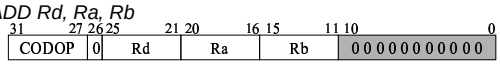
Plan

1. Le modèle de Von-Neumann
2. Format des instructions
3. Instructions arithmétiques
4. Instructions d'accès mémoire
5. Instructions de contrôle
6. Procédures

Format des instructions

- Qu'est-ce qu'une instruction ?
  - Chaîne de bit respectant un format

ADD Rd, Ra, Rb



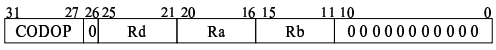
Exemple ADD R3, R1, R2

00000 0 00011 00001 00010 00000000000

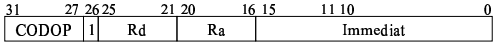
0x00611000

Format des instructions

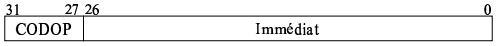
- F1



- F2



- F3



"L'assembleur"

- Représentation alphanumérique du format, pour la commodité de l'interaction homme-machine, par exemple ADD R1,R2,R3
  - Programmation directe
  - Résultat de la compilation (sous Unix, Option -S)
- Ou programme de codage : alphanum -> binaire.
  - Unix *as*

Plan

1. Le modèle de Von-Neumann
2. Format des instructions
3. Instructions arithmétiques
4. Instructions d'accès mémoire
5. Instructions de contrôle
6. Procédures

Instruction arithmétiques et logiques

- Arithmétiques
  - ADD, SUB et variantes : addition, soustraction
- Logiques
  - AND, OR, XOR et variantes : opérations logiques bit à bit
- Exemples
  - Etat initial R1 = 0x00001000 R2 = 0xEF125634
  - ADD R3, R2, R1 : R3 <-
  - AND R3, R2, R1 : R3 <-

Opérandes

- 2 opérandes d'entrée et 1 résultat = 3 opérandes à décrire dans l'instruction
- Modes d'accès à un opérande
  - Registre : l'opérande est dans un registre
  - Immédiat : l'opérande est dans l'instruction
    Dans ce cas l'opérande est limité à 16 bits
- Le résultat est toujours en registre !
- Format F1 : tous les opérandes en registre
- Format F2 : un des opérandes est un immédiat

Instructions arithmétiques format F1

- F1 = mode registre-registre
- Syntaxe ADD Rd, Ra, Rb
- Effet Rd <- Ra + Rb

Instructions arithmétiques format F2

- F2 = mode registre-immédiat
- Syntaxe ADDI Rd, Ra, $Immediat_{16}$
- Effet Rd <- Ra + ES(Immédiat)
- Application : compiler l'évaluation des expressions contenant des constantes

  Y := X + 12

  Se compile en

  < charger X dans R1>

  ADDI R2, R1, 12

  <ranger R2 dans Y>

Instructions arithmétiques format F2

- Exemples
  - ADDI R3, R1, 3 ou ADDI R3, R1, 0x0003
    - Initial R1 = 0x00001000 - Final R3 = 0x00001003

00000 1 00011 00001 0000000000000011
0x04610003

  - ADDI R3, R1, -2 ou ADDI R3, R1, 0xFFFE
    - Initial R1 = 0x00001000 - Final R3 = 0x00000FFFE

00000 1 00011 00001 1111111111111110
0x0461FFFE

Instructions logiques format F2

- F2 = mode registre-immédiat
- Syntaxe ANDI Rd, Ra, $Immédiat_{16}$
- Effet Rd <- Ra AND (0x0000||Immédiat)
- Exemples
  - ANDI R3, R1, 3 ou ANDI R3, R1, 0x0003
    00100 1 00011 00001 0000000000000011

    0x24610003

  - ANDI R3, R1, -2 ou ANDI R3, R1, 0xFFFE
    00100 1 00011 00001 1111111111111110

    0x2461FFFE

Instruction LUI

- Comment compiler Y := X + 0x12345678 ?
  ~~ADDI R3, R1, 0x12345678~~
- Format F2, champ Ra non significatif (à 0)
- Syntaxe LUI Rd, $Immediat_{16}$
- Effet $Rd_{31:16}$ <- Immediat
  $Rd_{15:0}$ <- 0X0000

```
LUI   R2, 0x1234      ; R2 <- 0x12340000
ADDI  R3, R2, 0x5678  ; R2 <- 0x12345678
ADD   R3, R1, R2      ; R3 <- X + 0x12345678
```

Instructions de décalage

- Format F2, bits 5 à 15 non significatifs (0)
- SLL : Shift Logical Left
  - Syntaxe SLL Rd, Imm_5,Ra
  - Effet Rd <- Ra décalé à gauche de Imm5 interprété en non signé
    Multiplication par 2^{Imm5}
  - Exemple
    SLL R2, 3, R1
    Initial R1 = 0x00010000
    Résultat R3 = 0x01000000

Instructions de décalage

- Format F2, bits 5 à 15 non significatifs (0)
- SLR : Shift Logical Right
  - Syntaxe SLR Rd, Imm_5,Ra
  - Effet Rd <- Ra décalé à droite de Imm5 interprété en non signé – C'est une décalage logique
    Division par 2^{Imm5} pour les entiers naturels
- SAR : Shift Arithmetic Right
  - Syntaxe SAR Rd, Imm_5,Ra
  - Effet Rd <- Ra décalé à droite de Imm5 interprété en non signé – C'est une décalage arithmétique
    Division par 2^{Imm5} pour les entiers relatifs

Plan

Chargement

Calcul d'adresse mode registre-immédiat

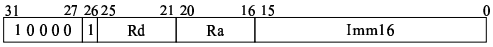
- *Syntaxe*

  LOAD Rd, Imm16 (Ra)

- *Effet*

  Rd <- Mem [Ra + ES(Imm16)]

- *Format F2*



Chargement - Exemple

On considère une mémoire organisée par octets

Etat initial

R1 = 0x00001000

12	*1000*
34	*1001*
56	*1002*
78	*1003*
AB	*1004*
CD	*1005*
EF	*1006*
09	*1007*

Effet des instructions

- LOAD R2, 0(R1)

  R1 <- 0x12345678

- LOAD R2, 4(R1)

  R1 <- 0xABCDEF09

Remarque : big endian

Rangement

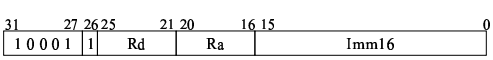
- *Syntaxe*

  STORE Rd, Imm16 (Ra)

- *Effet*

  Mem [Ra + ES(Imm16)] <- Rd

- *Format F2*



Instructions d'accès mémoire

Etat initial

R1 = 0x00001000
R2 = 0xEF125634

12	*1000*
34	*1001*
56	*1002*
78	*1003*
AB	*1004*
CD	*1005*
EF	*1006*
09	*1007*

Effet des instructions

STORE R2, 0(R1)

EF	*1000*
12	*1001*
56	*1002*
34	*1003*
AB	*1004*
CD	*1005*
EF	*1006*
09	*1007*

Représentation des variables en mémoire

- Les types élémentaires des LHN correspondent à un nombre précis d'octets

Caractères	« char »	1
Entiers courts	« short »	2
Entiers	« int »	4
Flottants simple p	« float »	4
Flottants double p	« double »	8

Représentation des variables en mémoire

- Dans quel ordre placer les variables multi-octets ?

int x = 0x12345678 ;

Mémoire organisée Big Endian

0x00001000	12
0x00001001	34
0x00001002	56
0x00001003	78

Mémoire organisée Little Endian

0x00001000	78
0x00001001	56
0x00001002	34
0x00001003	12

Représentation des variables en mémoire

- Alignement : à la compilation, les variables sont alignées sur leurs frontières naturelles = adresses multiples de leurs tailles

```
char c = 'A';
int x = 0x12345678 ;
char d ;
short z = 0xABCD;
float x = 0x01234567;
```


Représentation des variables en mémoire

- Tableaux : stockés séquentiellement à partir de l'adresse du premier élément

```
char t[5];
int u[4] = {    0x12345678 ;
                0x01020304;
                0xA5A6A7A8;
                0x11121314;}
```

- Adresse(tab[i]) = Adresse(tab[0]) + i*sizeof(élément_tableau)

Calculs d'adresse

LHN

```
int v [N], k ;
temp = v[k] ;
v[k] = v [k+1];
v[k+1] = temp
```

Assembleur

```
<Initialement, R1 = @v[0], R2 = k>
SLL     R2, 4, R2       ; deplacement <- k*4
ADD     R2, R2, R1      ; R2 <- @ v[k]
LOAD    R3, 0(R2)       ;  temp = R3 <- v[k]
LOAD    R4, 4(R2)       ; R4 <- v[k+1]
LOAD    R4, 0(R2)       ; v[k] <- R4
STORE R3, 4(R2)         ; v[k+1] <- temp
```

Plan

1. Le modèle de Von-Neumann
2. Format des instructions
3. Instructions arithmétiques
4. Instructions d'accès mémoire
5. Instructions de contrôle
6. Procédures

Instructions de branchement

- Comment réaliser les ruptures de séquence ?
- Branchement inconditionnel : BA
- Format F3
- Syntaxe BA $Immediat_{27}$
- Effet PC <- PC + ES($Immediat_{27}||00_2$)

Instructions de branchement

- NB : PC pointe l'instruction qui suit le branchement
- Les instructions occupent 4 octets et sont alignées
- Utiliser des étiquettes symboliques

	BA 1
+4	ADD …
+8	SUB…

```
        BA Adb
        ADD …
Adb:    SUB…
```

Instructions de branchement conditionnel

- Format F3
- Syntaxe B*cond* Immediat$_{27}$
- Effet PC <- PC + ES(Immediat$_{27}$||0x0)
  si *cond* vraie

Instructions de branchement conditionnel

- La condition résulte de l'instruction arithmétique la plus récente
  - Les instructions arithmétiques positionnent un bit dans le Registre Code Condition RCC
  - Le test de ces bits ou de combinaisons de ces bits correspond
    - Aux opérateurs booléens naturel sur les entiers : =, >, >=, <, <=
    - A des informations utiles au niveau LM : retenue, overflow

Z	Ov	N	C

  - Exemple

```
SUB   R3, R2, R1
BE    cible
```

branche si R2 = R1 en testant le bit Z de RCC

Positionnement du RCC

- Instructions CMP et CMPI
- Syntaxe CMP Ra, Rb et CMPI Ra, Imm16
- Effet Calcul de Ra – Rb ou Ra – ES(Imm16),
- mais le résultat n'est stocké dans aucun registre. Seul RCC est affecté.
- Les dépassements de capacité sont gérés correctement
- Exemple

```
CMP R1, R2
BGT cible
```

branche si R1 > R2 lorsque le contenu de R1 et R2 est interprété en entiers relatifs :
0x00000000 > 0xFFFFFFFF

Compilation des conditionnelles

Si <condition>		Bcond Ivrai
alors		Instructions du bloc 2
Bloc1		BA Isuite
sinon	Ivrai:	Instructions Bloc1
Bloc2	suite:	…

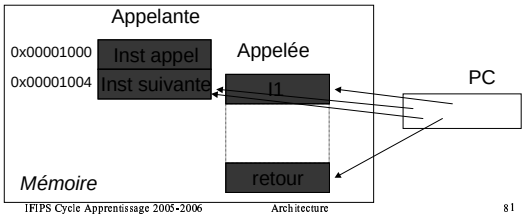
Plan

1. Le modèle de Von-Neumann
2. Format des instructions
3. Instructions d'accès mémoire
4. Instructions arithmétiques
5. Instructions de contrôle
6. Procédures

Procédures

Appel et retour de procédure

- L'appelante connaît l'adresse de l'appelée, mais le contraire n'est pas vrai
- Il faut assurer la sauvegarde de l'adresse de retour



Sauvegarde de l'adresse de retour

- Instruction JL
- Format F3
- Syntaxe JL deplacement$_{27}$
- Effet R31 <- PC
  PC <- PC + (0_b||deplacement$_{27}$||0x0)
- NB : la sauvegarde est prise en charge par le matériel

Retour de procédure

- Il faut un branchement sur registre !
- Syntaxe JMP Rd
- Effet PC <- Rd

Pile d'exécution

- Appels de procédures imbriqués : l'adresse de retour sera perdue
- Conventions logicielles sur
  - Le stockage en mémoire de l'adresse de retour et des paramètres
  - Le registre pointeur de pile