Notre CPU

Le jeu d'instruction

- Le langage assembleur est le langage de plus bas niveau que l'on puisse imaginer.
- Les instructions sont supportées par le processeur nativement.
- La transformation en code exécutable par la machine est direct :
  - Les mnémoniques en toutes lettres n'ont qu'à être converties directement dans l'encodage binaire des instructions défini pour le jeu d'instruction.
- Il n'y a aucune abstraction dans l'assembleur par rapport au processeur sous-jacent.
- L'assembleur ne sert qu'à rentre plus intelligible par l'homme le programme machine mais la traduction est directe contrairement aux langages de haut-niveau.
- Les concepts manipulés par le langage sont directement ceux du processeur.

Notre CPU

Le jeu d'instruction : les labels

- L'adresse à laquelle se trouvera le code exécutable en mémoire a rarement besoin d'être connue lors de l'écriture du programme.
- L'être humain se perd facilement avec la démultiplication des quantité numériques.
- C'est pourquoi l'assembleur permet d'utiliser des LABEL :
  - Il s'agit un alias permettant de se référer à une adresse symboliquement.
  - C'est lors de la création de l'exécutable que les LABEL Sont remplacé par des valeurs d'adresse.

Notre CPU

Le jeu d'instruction : déplacement de données

- De la mémoire vers un registre :
  - MOVE @,R_d
  - Exemple : MOVE @1000,R_0 copie le contenu de la casse mémoire @1000 dans le registre R_0
- D'un registre vers la mémoire :
  - MOVE R_s, @
  - Exemple : MOVE $R_{0,}$ @1000, copie le contenu du registre R_0 dans casse mémoire @1000
- D'un registre vers un registre :
  - MOVE R_s,R_d
  - Exemple : MOVE R_0,R_1 copie le contenu du registre R_0 dans le registre R_1
- Initialiser un registre avec une constante :
  - MOVE #N,R_d
  - Exemple : MOVE #152,R_1 initialise le registre R_1 à la valeur 152

Notre CPU

Le jeu d'instruction : déplacement de données

- On peut vouloir transférer différent type de données de/vers la mémoire :
  - Des octets.
  - Des mots de 16 bits.
  - Des mots de 32 bits.
- Les instruction d'interaction avec la mémoire portent donc une information supplémentaire de type :
  - MOVE.B @1000,R_0 copie l'octet contenu à l'adresse 1000 dans R_0 .
  - MOVE.W @1000,R_0 copie le mot de 16 bits contenu aux adresses 1000, 1001 dans R_0 .
  - MOVE.L @1000,R_0 copie le mot de 32 bits contenu aux adresses 1000 … 1003 dans R_0 .

Notre CPU

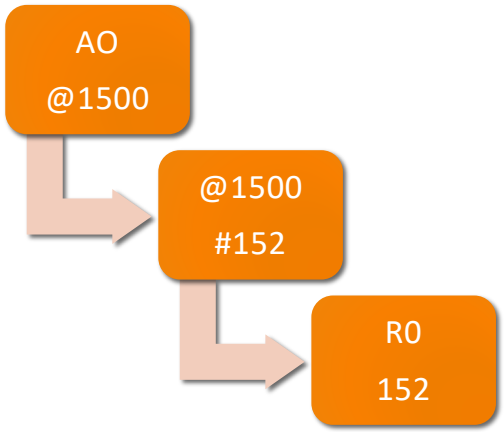
Le jeu d'instruction : Adressage Indirect

- Il est courant de devoir accéder de manière dynamique à une donnée :
  - L'adresse où se trouve l'information n'est pas une constante mais une variable.
  - Par exemple pour balayer les valeurs à un tableau.
  - C'est le concept d'indirection :
    - Je vous donne l'adresse de Paul qui vous donnera l'adresse de Jacques.
- De la mémoire vers un registre avec indirection :
  - MOVE.B (A_0),R_0 copie un octet dans R_0 ,l'adresse de l'octet est donné par registre A_0.
- D'un registre à la mémoire avec indirection :
  - MOVE.B R_0,(A_0), copie un octet contenu dans R_0 à l'adresse donnée par A_0.
- Exemple :
  - Si A_0 contient la valeur @2500.
  - MOVE.B (A_0),R_0 copie dans R_0 l'octet se trouvant à l'adresse @2500.

Notre CPU

Le jeu d'instruction : Adressage Indirect

- Exemple d'usage :
  - Si l'on veut balayer l'ensemble des valeurs de la composante rouge d'une image codée en RGB 8 bits.
  - On initialise un registre d'adresse avec l'adresse du 1° pixel.
  - On incrémente le registre d'adresse pour passer au pixel suivant.
  - MOVE.B (A_0),R_O



Notre CPU

Le jeu d'instruction : Arithmétique

- Addition:
  - ADD R_s ,R_d
  - R_s+R_d → R_d
- Soustraction :
  - SUB R_s ,R_d
  - R_d-R_s → R_d
- Multiplication :
  - MUL R_s ,R_d
  - R_s*R_d → R_d
- Division :
  - DIV R_s ,R_d
  - R_d/R_s → R_d
- Modulo :
  - MOD R_s ,R_d
  - R_d%R_s → R_d
- La source peut être une constante notée #N :
  - SUB #2,R_0 retranche 2 au registre R_0

Notre CPU

Le registre d'état

- Il s'agit d'un registre dont chaque bit est un drapeau dont l'état dépend du résultat du dernier calcul :
  - Bit Z : à 1 si le résultat du dernier calcul est nul, à 0 si non.
  - Bit N : à 1 si le résultat du dernier calcul est négatif, à 0 si non.
  - Bit C : bit de retenue à 1 si le résultat du dernier calcul produit une retenue, à 0 si non.
  - Bit O : bit d'overflow à 1 si le résultat du dernier calcul produit une dépassement, à 0 si non.

Notre CPU

Le jeu d'instruction : La comparaison

- Comment comparer des valeurs ?
  - En faisant une soustraction
  - CMP R_s ,R_d
  - R_d-R_s mais n'affecte pas le résultat à R_d , seul les bit du registre d'état reflètent le résultat
    - Si Z est à 1 → R_s==R_d
    - Si N est à 1 → R_s>R_d
    - Si N est à 0 → R_s<R_d

Notre CPU

Le jeu d'instruction : Branchement inconditionnel

- Comment rompre la linéarité de l'exécution ?
  - Après l'exécution d'une instruction le PI est normalement incrémenté pour traiter l'instruction suivante.
  - BRA @
    - Cette instruction force l'écrasement du PI par @ au lieu d'une incrémentation.
    - L'instruction suivante peut-être n'importe où en mémoire !
    - Nous rompons la linéarité de l'exécution des instructions dans le programme.

Notre CPU

Le jeu d'instruction : Branchement conditionnel

CMP R_0 ,R_1	Branchement en @5000 si
BEQ @5000	R_0 ==R_1
BNE @5000	R_0 !=R_1
BGT @5000	R_0 > R_1
BLT @5000	R_0 < R_1
BGE @5000	R_0 >= R_1
BLE @5000	R_0 <= R_1

Codons !

Structures conditionnelles

LABEL	INSTRUCTION	COMMENTAIRE
SI :	CMP R0,R1	
	BNE FIN_SI	
ALORS :	*Groupe d'instructions conditionnel* *Exécuté si R0==R1*	
FIN_SI :		

Codons !

Structures conditionnelles

LABEL	INSTRUCTION	COMMENTAIRE
SI :	CMP R0,R1	
	BNE SINON	
ALORS :	*Groupe d'instructions conditionnelles Exécuté si R0==R1*	
	BRA FIN_SI	
SINON :	*Groupe d'instructions conditionnelles Exécuté si R0!=R1*	
FIN_SI :		