Architecture des ordinateurs

Exemple d'architecture : x86 (IA-32 ou Pentium)

L1 MIASHS

UFR Mathématiques et Informatique

(2014 - 2015)

Registres généraux (32 bits)

EAX : registre accumulateur

- Pour les opérations arithmétiques
- Pour le stockage de la valeur de retour des appels systèmes

ECX : registre compteur

EBX : registre de base

EDX : registre de données

- Pour les opérations arithmétiques
- Pour les opérations d'E/S

AX : 16 bits de poids faible de EAX (idem BX, CX, DX)

AL : octet de poids faible de AX (idem BL, CL, DL)

AH : octet de poids fort de AX (idem BH, CH, DH)

Registres spécialisés (32 bits)

- Registres d'adresses :
  - ESI : pointeur source (*Extended Source Index*)
  - EDI : pointeur destination (*Extended Destination Index*)
  - EBP : pointeur de base (*Extended Base Pointer*)
  - ESP : pointeur de pile (*Extended Stack Pointer*)
- Autres registres :
  - EIP : pointeur d'instruction (*Extended Instruction Pointer*)
  - EFLAGS : registre d'états (drapeaux)
  - CS, SS, DS, ES, FS, GS : registres de segment (adresses et données du pgme)

Drapeaux

Zero Flag (ZF)

→ 1 si les deux opérandes utilisés sont égaux, 0 sinon

Overflow Flag (OF)

→ 1 si le dernier résultat a provoqué un overflow, 0 sinon

Carry Flag (CF)

→ 1 si la dernière opération a généré une retenue, 0 sinon

Sign Flag (SF)

→ 1 si la dernière opération a généré un résultat négatif, 0 sinon

Parity Flag (PF)

→ 1 si la dernière opération a généré un résultat impair, 0 sinon

Interrupt Flag (IF)

→ 1 si les interruptions sont autorisées, 0 sinon

Instructions x86

- Opérations de transfert (copie de données)
  - Entre la mémoire et les registres
  - Opérations sur la pile
- Opérations arithmétiques
- Opérations logiques
- Opérations de décalage et rotation : multiplications et divisions rapides
- Opérations de branchements :
  - Sauts
  - Boucles
  - Appels de fonctions
- Opérations sur les chaînes de caractères

Instructions de transfert

- Copie de données entre mémoire et registre : `MOV dst, src`
- Échange des contenus : `XCHG R1, R2`
  - Entre 2 registres : `XCHG EAX, EBX`
  - Entre un registre et une case mémoire : `XCHG EAX, [0xgbfffeedc]`
- Opérations de pile : `PUSH` et `POP`

Instructions de transfert : Résumé

Instruction	Description
`MOV dst, src`	Déplace `src` dans `dst`
`XCHG ds1, ds2`	Échange `ds1` et `ds2`
`PUSH src`	Place `src` au sommet de la pile
`POP dst`	Supprime le sommet de la pile et le place dans `dst`

Instructions arithmétiques

- Addition entière (en complément à 2) : `ADD dst, src`
  - `src` et `dst` peuvent être une valeur, un registre ou une adresse mémoire
  - Positionne les FLAGS : Carry (CF) et Overflow (OF)
- Addition avec retenue : `ADC dst, src`
  - Additionne les deux opérandes
  - Positionne la retenue dans CF
- Multiplication entière positive : `MUL src`
  - 1 seule opérandre : multiplication par EAX
  - Résultat stocké dans deux registres :
    - EDX (bits de poids fort)
    - EAX (bits de poids faible)
- Multiplication entière en complément à 2 (entiers relatifs) : `IMUL src`

Instructions arithmétiques : Résumé

Instruction	Description
`ADD dst, src`	Ajoute `src` à `dst`
`ADC dst, src`	Ajoute `src` à `dst` avec retenue
`SUB dst, src`	Soustrait `src` à `dst`
`SBB dst, src`	Soustrait `src` à `dst` avec retenue
`MUL src`	Multiplie `EAX` par `src` (résultat dans `EDX`|`EAX`)
`IMUL src`	Multiplie `EAX` par `src` en complément à 2
`DIV src`	Divise `EDX`|`EAX` par `src` (`EAX` = quotient, `EDX` = reste)
`IDIV src`	Divise `EDX`|`EAX` par `src` (en complément à 2)
`INC dst`	`dst` + 1
`DEC dst`	`dst` - 1
`NEG dst`	`-dst`

Instructions logiques : Résumé

Instruction	Description
`NOT dst`	Place (`NOT dst`) dans `dst`
`AND dst, src`	Place (`src AND dst`) dans `dst`
`OR dst, src`	Place (`src OR dst`) dans `dst`
`XOR dst, src`	Place (`src XOR dst`) dans `dst`

Instructions de décalage/rotation

2 opérandes : un registre suivi d'un nombre `nb` de décalages

- Décalage logique à gauche : `SHL dst, nb`
  - Insertion de `nb` 0 au niveau du bit de poids faible
  - Permet d'effectuer efficacement la multiplication par 2^{nb}
- Décalage arithmétique à droite : `SAR dst, nb`
  - Insertion de `nb` copies du bit de poids fort à gauche
  - Permet la division rapide d'un entier relatif par 2^{nb}
- Rotation à gauche : `ROL dst, nb`
  - Rotation de `nb` bits vers la gauche
  - Les bits sortants sont immédiatement réinjectés à droite
- Rotation à droite avec retenue : `RCR dst, nb`
  - Rotation de `nb` bits vers la droite en passant par la retenue
  - Le bit sortant à droite est mémorisé dans la retenue qui est elle-même réinjectée à gauche

Instructions de décalage/rotation : Résumé

Instruction	Description
`SAL dst, nb`	Décalage arithmétique à gauche de `nb` bits de `dst`
`SAR dst, nb`	Décalage arithmétique à droite de `nb` bits de `dst`
`SHL dst, nb`	Décalage logique à gauche de `nb` bits de `dst`
`SHR dst, nb`	Décalage logique à droite de `nb` bits de `dst`
`ROL dst, nb`	Rotation à gauche de `nb` bits de `dst`
`ROR dst, nb`	Rotation à droite de `nb` bits de `dst`
`RCL dst, nb`	Rotation à gauche de `nb` bits de `dst` avec retenue
`RCR dst, nb`	Rotation à droite de `nb` bits de `dst` avec retenue

Instructions de branchement

- Instruction de comparaison : `CMP sr1, sr2`
  - Effectue une soustraction (comme sub) mais ne stocke pas le résultat
  - Seul le drapeau ZF est modifié
- Saut conditionnel vers l'étiquette spécifiée : `Jxx addr`
  - `JE, JNE` : jump if (not) equal

    → Saute si le drapeau ZF est égal à 1 (ou 0)
  - `JGE, JNGE` : jump if (not) greater or equal

    → Saute si le résultat de `CMP` est (ou n'est pas) supérieur ou égal à
  - `JL, JNL` : jump if (not) less than

    → Saute si le résultat de `CMP` est (ou n'est pas) strictement inférieur à
  - `JO, JNO` : jump if (not) overflow
  - `JC, JNC` : jump if (not) carry
  - `JP, JNP` : jump if (not) parity
  - `JCXZ, JECXZ` : jump if CX (ou ECX) is null

Instructions de branchement : Résumé

Instruction	Description
`CMP sr1, sr2`	Compare `sr1` et `sr2`
`JMP addr`	Saut vers l'adresse `addr`
`Jxx addr`	Saut conditionné par `xx` vers l'adresse `addr`
`LOOP addr`	Répétition de la boucle `nb` fois (`nb` dans `ECX`)
`LOOPx addr`	Répétition de la boucle conditionnée par `x`