Sémantique

Le processeur réalise un graphe de transitions, dont la sémantique est mathématiquement précise.

Sommets: configurations $(mem, eax, ebx, \ldots, pc)$, où
$mem : \mathbb{N} \to [0..255]$, $eax, ebx, ..., pc \in [0..\texttt{0xffffffff}]$.

(pour des registres 32 bits)

Relation de transition:
$(mem, eax, ebx, \ldots, pc) \xrightarrow{\delta} (mem', eax', ebx', \ldots, pc')$.

Règles de sémantique opérationnelle (ex.):

$$mem, eax, ebx, ecx, \ldots, pc \vdash \texttt{mov \%eax, \%ebx} \Rightarrow mem, eax, eax, ecx, \ldots, pc$$

Les instructions du langage machine

L’exemple de la famille Pentium.

Instructions de base

- Transferts: `mov` ⟨source⟩, ⟨dest⟩

  (constante/registre/mémoire[absolu/indirect]→registre/mémoire[absolu/indirect])

- Instructions arithmétiques et logiques: `sub` ⟨source⟩, ⟨dest⟩

  (aussi, `add`, `mul`, `div`, `xorb`, `andb`, `orb`, `sar`, `sal`, ...)

- Comparaisons: `cmp` ⟨source⟩, ⟨dest⟩

  (même format que `mov`)

- Sauts, conditionnels ou non: `jmp` ⟨addr⟩

  (⟨addr⟩=constante/relative/indirecte) (aussi, `je` [égal]/`jne`,

  `jl` [less]/`jge`, `jg` [greater]/`jle`, `jb` [below]/`jnb`, `ja` [above]/`jna`)

- Sous-programmes: `call` ⟨addr⟩, `ret`

- Divers: `push`, `pop`, `movzbl`, `cmpxchg8b`, ...

Les modes d'adressage

- immédiat (constante) `$0xc`

  (`sub $0xc, %eax`: soustraire 12 de `%eax`)

- registre `%eax`

  (`mov $0xc, %eax`: mettre 12 dans `%eax`)

- mémoire absolue `0xba48`

  (`mov %eax, 0xba48`: transférer `%eax` aux adresses `0xba48-0xba4b`)

- mémoire indirecte `0x8(%ebp)`

  (`mov 0x8(%ebp),%edi`: [supposant `%ebp=0x14`], transférer les 4 octets aux adresses `0x14+8-0x14+11` dans `%edi`)

- relatif

  (`<main+19>:jge <main+102> 0x7d 0x51`: aller `0x51` octets plus loin si $\geq$)

- + autres modes (les Pentium en sont particulièrement riches...)

Instructions dérivées: sous-programmes

- `call` ⟨addr⟩ $\sim$ `push` ⟨pc-après-call⟩; `jmp` ⟨addr⟩
- `ret` $\sim$ `pop` tmp; `jmp (`tmp`)`
- `push` ⟨source⟩ `sub $4, %esp`; `mov` ⟨source⟩`, (%esp)`
- `pop` ⟨dest⟩ `mov (%esp),` ⟨dest⟩; `add $4, %esp`

(Exercice: que font ces instructions? Indication: `%esp` pointe sur le sommet de la pile; on empile vers le bas de la mémoire.)

Sémantique des modes d'adressage

$Cfg = (mem, eax, ebx, \ldots, st, pc)$

$$
\begin{array}{rcl}
[\![\$n]\!]^w_{rd}\,(Cfg) & = & n \\
[\![\%\texttt{eax}]\!]^w_{rd}\,(mem, eax, \ldots, pc) & = & eax \\
[\![\%\texttt{eax}]\!]^w_{wr}\,(mem, eax, \ldots, pc)(N) & = & (mem, N, \ldots, pc) \\
 & \ldots & \\
[\![n]\!]^w_{rd}\,(mem, eax, \ldots, pc) & = & mem(n)..mem(n+w-1) \\
 & & (\text{où } a_1..a_n = a_1 + 256a_2 + \ldots + 256^{n-1}a_n) \\
[\![n]\!]^w_{wr}\,(mem, eax, \ldots, pc)(N) & = & (mem[n..n+w-1 \mapsto N], eax, \ldots, pc) \\
 & & (\text{où si } mem' = mem[n..m \mapsto N], mem'(k) = mem(k) \text{ si } k < n \text{ ou } k \geq m, \\
 & & mem'(n)..mem'(m-1) = N \bmod \texttt{0x100000000}) \\
[\![n(\%\texttt{eax})]\!]^w_{rd}\,(mem, eax, \ldots, pc) & = & mem(eax+n)..mem(eax+n+w-1) \\
[\![n(\%\texttt{eax})]\!]^w_{wr}\,(mem, eax, \ldots, pc)(N) & = & (mem[eax+n..eax+n+w-1 \mapsto N], \\
 & & eax, \ldots, pc)
\end{array}
$$

Sémantique

$$\frac{Cfg' = [\![dst]\!]^4_{wr}\,(Cfg)([\![src]\!]^4_{rd}\,(Cfg))}{Cfg \vdash \texttt{mov}\ src, dst \Rightarrow Cfg'}$$

$$\frac{\begin{array}{c} Cfg = (mem, \ldots, st, pc) \quad x = [\![src]\!]^4_{rd}\,(Cfg) \quad y = [\![dst]\!]^4_{rd}\,(Cfg) \quad z = x - y \\ x' = signed(x) \quad y' = signed(y) \qquad z' = x' - y' \quad Cfg' = (mem, \ldots, st', pc) \\ st' = [z > \texttt{0xffffffff}, z < 0, z = 0, z' > \texttt{0x7fffffff} \vee z' < -\texttt{0x80000000}, z' < 0, \ldots] \end{array}}{\begin{array}{c} Cfg \vdash \texttt{cmp}\ src, dst \Rightarrow Cfg' \\ (signed(x) = \text{if } x \geq \texttt{0x80000000} \text{ alors } x - \texttt{0x100000000} \text{ sinon } x) \end{array}}$$

$$\frac{\begin{array}{c} Cfg = (mem, \ldots, st, pc) \quad st = (\_, \_, \_, \_, \_, b, ...) \\ Cfg' = (mem, \ldots, st, \text{si } b \text{ alors } pc + 2 \text{ sinon } pc + 2 + offset) \end{array}}{Cfg \vdash \texttt{jge}\ offset \Rightarrow Cfg'}$$

Exercices

- Écrire des règles sémantiques pour `jmp`, `sub`, `push`, `pop`, `call`, `ret`, `andb` (et bit à bit), `negb` (négation bit à bit).
- Pourquoi $mem'(n)..mem'(m-1) = N$ mod `0x100000000` dans la définition de $mem' = mem[n..m \mapsto N]$?
- La représentation des entiers utilisée est dite en *complément à deux*. Montrer que $-x = \texttt{negb}(x) + 1$. Que calcule $\texttt{andb}(x, -x)$? En déduire un test efficace si x contient 0, 1, ou plusieurs bits à 1.

2	$=$	`0x00000002`	-2	$=$	`0xfffffffe`
-16	$=$	?	18	$=$	?

Additionner les représentations de -16 et 18; de -16 et 2: que trouve-t-on?