7 The Target Language: k86 Machine

The target machine's address space is not pre-divided into registers and the main memory. A particular memory architecture is programmed by declaring memory areas (using `AREA`) each of which is addressed by a pair of an area name and an offset in the area. Conventional machines that have registers `r0, r1` and etc. can be mimicked by declaring an area, say, `r` and using locations `r(0), r(1)` and so on.

A single memory location can have either an integer, a string (of any length), a label, or a location.

k86 programs are interpreted by our interpreter. The interpreter, after executing a k86 program, reports the execution cost of the program. The execution cost consists of the number of instruction executions, memory consumption, memory accesses, and etc. The interpreter is parameterized by a cost table that determines the costs of using particular areas. By using different cost tables we can measure a k86 program's execution cost on various memory architectures.

The syntax of the target language k86 is:

pgm	::=	$decl^+ instr^+$	program
decl	::=	`AREA` *id*	memory area declaration
instr	::=	`MOVE` *m m*	
	|	`ADD` *m m m*	
	|	`SUB` *m m m*	
	|	`MUL` *m m m*	
	|	`DIV` *m m m*	
	|	`TOZ` *m m*	
	|	`JMP` *m*	
	|	`JMPZ` *m m*	
	|	`JMPN` *m m*	
	|	`LAB` *label*	
	|	`READ` *m*	
	|	`WRITE` *m*	
m	::=	*integer* | "*string*" | *id* | *label*	
	|	*m*`(`*m*`)`	
	|	*m*`@`$^+$	
id, *label*			alpha-numeric identifier

For example, a summation program in k86:

```
AREA A          // sum 0 upto A
```

```
AREA S          // S has the result
LAB     START
        READ  A
        MOVE  0  S
LAB     LOOP
        ADD    S@  A@  S
        SUB    A@  1   A
        JMPZ   A@  EXIT
        JMP    LOOP
LAB     EXIT
        WRITE  S@
LAB     END
```

Let's define the meaning of k86 command.

$$
\begin{array}{lcllll}
M & \in & Mem & = & Loc \to Val & \text{memory} \\
 & & Loc & = & Area \times Int & \text{location} \\
 & & Val & = & Loc \cup Int \cup Lab \cup String & \text{value} \\
pc & \in & Int & & & \text{program counter} \\
m & \in & Addr & & & \text{address term} \\
\ell & \in & Lab & & & \text{label}
\end{array}
$$

One execution step

$$(M, pc) \to (M', pc')$$

indicates that a memory M changes to M' after an instruction at pc is executed, and the next program counter becomes pc'. Notation $M[x \mapsto y]$ indicates a new memory that is equivalent to M except at x, where its value is y.

$$
\begin{array}{lcl}
(M, pc : \texttt{MOVE}\ m_1\ m_2) & \to & (M[val(M, m_2) \mapsto val(M, m_1)], pc + 1) \\
(M, pc : \texttt{ADD}\ m_1\ m_2\ m_3) & \to & (M[val(M, m_3) \mapsto val(M, m_1) \oplus val(M, m_2)], pc + 1) \\
(M, pc : \texttt{SUB}\ m_1\ m_2\ m_3) & \to & (M[val(M, m_3) \mapsto val(M, m_1) \ominus val(M, m_2)], pc + 1) \\
(M, pc : \texttt{MUL}\ m_1\ m_2\ m_3) & \to & (M[val(M, m_3) \mapsto val(M, m_1) \otimes val(M, m_2)], pc + 1) \\
(M, pc : \texttt{DIV}\ m_1\ m_2\ m_3) & \to & (M[val(M, m_3) \mapsto val(M, m_1) \oslash val(M, m_2)], pc + 1) \\
(M, pc : \texttt{TOZ}\ m_1\ m_2) & \to & (M[val(M, m_2) \mapsto cast(val(M, m_1))], pc + 1) \\
(M, pc : \texttt{JMP}\ m) & \to & (M, pcfy(val(M, m))) \\
(M, pc : \texttt{JMPZ}\ m_1\ m_2) & \to & (M, \text{if } val(M, m_1) = 0 \text{ then } pcfy(val(M, m_2)) \text{ else } pc + 1) \\
(M, pc : \texttt{JMPN}\ m_1\ m_2) & \to & (M, \text{if } val(M, m_1) < 0 \text{ then } pcfy(val(M, m_2)) \text{ else } pc + 1) \\
(M, pc : \texttt{READ}\ m) & \to & (M[val(M, m) \mapsto get()], pc + 1) \\
(M, pc : \texttt{WRITE}\ m) & \to & (M, pc + 1) \\
(M, pc : \texttt{LAB}\ \ell) & \to & (M, pc + 1) \\
(M, pc : \texttt{LAB END}) & \to & (M, pc)
\end{array}
$$

Auxiliary function $val(M, m)$ indicates a value represented by an address term m at a given memory M:

$$val: Mem \times Addr \rightarrow Val$$

$val(M, a)$	$= \langle a, 0 \rangle$	aread id
$val(M, x)$	$= x$	integer/string/label
$val(M, m_1(m_2))$	$= val(M, m_1) \oplus val(M, m_2)$	offset address
$val(M, m\texttt{@})$	$= M(val(M, m))$	dereference

Casting function $cast(v)$ casts a location to an integer by removing the area name from the location:

$$cast: Loc \rightarrow Int$$

$$cast\langle a, z \rangle = z.$$

$v_1 \oplus v_2$ is defined only when

- both v_1 and v_2 are integer. Then the $\oplus$ is the integer addition.
- both v_1 and v_2 are locations whose areas are identical. Then $v_1 \oplus v_2$ is the location in the area whose offset is the sum of v_i's offsets.
- one of them is an integer and the other is a location. Then the result is the location in the same area whose offset is incremented by the integer.

Similarly for $\ominus$,$\otimes$, and $\oslash$.

- "`AREA sp`" declares new memory area `sp`.
- "`MOVE 2 sp`" moves constant 2 to location $\langle \texttt{sp}, 0 \rangle$.
- "`MOVE "cs" sp`" moves string "`cs`" to location $\langle \texttt{sp}, 0 \rangle$.
- "`MOVE sp@ sp`" moves value at location $\langle \texttt{sp}, 0 \rangle$ to location $\langle \texttt{sp}, 0 \rangle$.
- "`MOVE sp(-1)@ sp`" moves value at $\langle \texttt{sp}, -1 \rangle$ to location $\langle \texttt{sp}, 0 \rangle$.

Function $pcfy(\ell)$ returns a label ℓ's program counter value, and $input()$ gets an integer from the outside world.

Interpretation (execution) of a program P is the sequence of states starting from an empty memory and the initial program counter at label `START`:

$$([], pcfy(\texttt{START})) \rightarrow \cdots$$

Note that the end of the program is when the program counter reaches the `END` label. Therefore any program must have the two labels (`START` and `END`).