5.5　仮想マシンコードの生成

$\mathcal{C}$ から直接にアセンブリを生成することもできるのだが，アセンブリ言語は書かれている命令を順番に実行していく**命令形言語** (*imperative language*) なので，関数型言語である $\mathcal{C}$ とはギャップがまだ大きい．そこで，$\mathcal{C}$ とアセンブリ言語の間に仮想マシン言語 $\mathcal{V}$[1] という中間言語を挟むことにする.[2]

$\mathcal{V}$ の定義を示す前に，$\mathcal{V}$ がどんな感じの言語かを見てみよう．以下のプログラムは，言語 $\mathcal{V}$ で書かれた，3 に 1 を加えるプログラムである．

$$
\begin{array}{ll}
l_f : & \\
& \mathbf{local}(4) \leftarrow \mathbf{param}(1) \\
& \mathbf{local}(0) \leftarrow \mathit{add}(\mathbf{local}(4), \mathbf{imm}(1)) \\
& \mathbf{returnlocal}(0) \\
l_{main} : & \\
& \mathbf{local}(0) \leftarrow \mathbf{call\ labimm}(l_f)(\mathbf{imm}(3))
\end{array}
$$

プログラムは命令の列である．プログラム中の l_f や l_{main} は**ラベル** (*label*) と呼ばれる識別子で，プログラム中の位置を表している．ラベルは処理のジャンプ先を指定する際に用いられる．例えば，プログラム中の $\ldots \leftarrow \mathbf{call}\ l_f(\ldots)$ 命令は関数呼び出しをするために l_f に処理を移す命令である．

このプログラム中の各命令の動作を順番に見てみよう．ラベル l_f から始まる部分にかかれている命令は以下のとおりである．

$\mathbf{local}(4) \leftarrow \mathbf{param}(1)$ ：ラベル l_f から始まる関数の第一引数の内容を，$\mathbf{local}(4)$ で指される記憶領域に格納する．アセンブリ言語において関数呼び出しを実装するには，呼び出された関数のローカルな記憶領域（この記憶領域のことを**フレーム** (*frame*) と呼ぶ）をどのように確保するか，その記憶領域をどのように使うか，引数や返り値をどのように受け渡しするかを決定する必要がある．これらの決まりごとを**呼び出し規約** (*calling convention*) という．言語 $\mathcal{V}$ においては，関数に渡された引数は $\mathbf{param}(1), \mathbf{param}(2), \ldots$ で参照し，ローカルな変数の格納先は $\mathbf{local}(0), \mathbf{local}(4), \ldots$ のようにローカルな記憶領域内部の場所を表す名前を付けておくことにより，あとでアセンブリ生成を行う際に呼び出し規約を完全に決められるようにしてある．

$\mathbf{local}(0) \leftarrow \mathit{add}(\mathbf{local}(4), \mathbf{imm}(1))$ ：フレーム中で $\mathbf{local}(4)$ という名前で指される領域に格納されている値（すなわち前の命令でセットされた関数の第一引数）と整数値 1 とを加算して $\mathbf{local}(0)$ に格納する．$\mathbf{imm}(n)$ は整数定数 n を表すオペランドで，imm というオペランド名はアセンブリ言語で命令語中に直接現れる定数を表す**即値** (*immediate*) に由来する．

---

[1]「仮想マシン言語」という名前は，命令形の中間言語の名前として本書で便宜的に使っている名前である．このような中間言語に相当する言語は多くのコンパイラやコンパイラの教科書で用いられているが，その名前は様々である．

[2]ソース言語とターゲット言語の間にどのような中間言語を挟むかはコンパイラを作る上で重要なデザインチョイスである．本書では $\mathcal{C}$ と $\mathcal{V}$ を中間言語として挟むが，より多くの中間言語を挟むコンパイラもある．

return : **local**(0) に格納されている値を関数の返り値として返す.

l_{main} はプログラムが起動されたときに実行が始まるプログラム中の箇所を指すラベルであり，命令 $\mathbf{local}(0) \leftarrow \mathbf{call}\ \mathbf{labimm}(l_f)(\mathbf{imm}(3))$ が書いてある．この命令は l_f から始まる命令列を関数と思って引数 $\mathbf{imm}(3)$ で呼び出し，返り値を記憶領域 $\mathbf{local}(0)$ に格納する．

$\mathcal{V}$ は以下の BNF で定義される言語である．

$$
\begin{array}{rcl}
op & ::= & \mathbf{param}(n) \mid \mathbf{local}(\mathit{ofs}) \mid \mathbf{labimm}(l) \mid \mathbf{imm}(n) \\
i & ::= & \mathbf{local}(\mathit{ofs}) \leftarrow op \mid \mathbf{local}(\mathit{ofs}) \leftarrow op(op_1, op_2) \mid l : \mid \mathbf{if}\ op\ \mathbf{then\ goto}\ l \\
& \mid & \mathbf{goto}\ l \mid \mathbf{local}(\mathit{ofs}) \leftarrow \mathbf{call}\ op_0(op_1, \ldots, op_n) \mid \mathbf{return}(op) \\
d & ::= & \langle l \,\|\, i_1 \ldots i_m \,\|\, n \rangle \\
P & ::= & \langle d_1 \ldots d_m \,\|\, i_1 \ldots i_n \,\|\, k \rangle
\end{array}
$$

l はラベル名を表すメタ変数，ofs は整数値である．プログラムは**命令** (*instruction*) の列である．各命令は，命令の種類と命令の引数（**オペランド** (*operand*)）によってどのように動作するかが決まる．言語 $\mathcal{V}$ のオペランドは値の記憶領域か定数値を表す情報で，具体的には以下のいずれかである．

param(n) : 関数に渡された n 番目の引数の格納場所を表す.

local(ofs) : 現在のフレームのうち,「基準となるアドレス」から ofs バイト目のアドレスを表す.[3]

imm(n) : 整数定数 n を表す.

labimm(l) : ラベル名 l を表す定数を表す．関数呼び出しを行う際に使用する．

では，各命令の意味を説明しよう．以下の説明で「op の値」という表現を用いることがある．これは，op が $\mathbf{param}(n)$ であれば n 番目の引数として渡された値を，$\mathbf{local}(\mathit{ofs})$ であればフレーム中の場所 ofs に格納されている値を，$\mathbf{imm}(n)$ であれば整数値 n を，それぞれ表す.

$\mathbf{local}(\mathit{ofs}) \leftarrow op$: op の値をフレーム中の $\mathbf{local}(\mathit{ofs})$ の指す記憶領域に格納する．

$\mathbf{local}(\mathit{ofs}) \leftarrow op(op_1, op_2)$: op_1 と op_2 の値を op で計算して，フレーム中の $\mathbf{local}(\mathit{ofs})$ の場所に格納する．

l : : プログラム中のラベル名 l で指される場所を表す.

$\mathbf{if}\ op\ \mathbf{then\ goto}\ l$: op の値が 0 でなければ l に制御を移す．そうでなければ何もしない．

$\mathbf{goto}\ l$: l に制御を移す.

[3]後述のフレームの内部構造のところでもう少し詳しく説明する．

$\mathbf{local}(\mathit{ofs}) \leftarrow \mathbf{call}\ op(op_1, \ldots, op_n)$: $op_1, \ldots, op_n$ の値を引数として op に格納されているラベルから始まる命令列を関数として呼び出す．関数が返ったら，返り値を $\mathbf{local}(\mathit{ofs})$ に格納する．

$\mathbf{return}(op)$: op に格納されている値を現在実行中の関数の返り値として返す．

関数定義 $\langle l \,\|\, i_1 \ldots i_m \,\|\, n\rangle$ は，関数のラベル名と，その関数本体の命令列と，関数内で使われるローカル変数に必要な記憶領域のサイズ n からなる．この記憶領域サイズは，後のコード生成フェーズで使用される．プログラムは $\langle d_1 \ldots d_m \,\|\, i_1 \ldots i_n \,\|\, k\rangle$ の形をしており，関数定義の列 $d_1 \ldots d_m$ と，メインのプログラムに対応する命令列 $i_1 \ldots i_n$ と，メインのプログラム内で使われるローカル変数のための記憶領域のサイズ k からなる．

$\mathcal{C}$ から $\mathcal{V}$ への変換を図 5.1 に示す．**変換の定義を簡潔に保つために，変換対象の $\mathcal{C}$ プログラムではすべての束縛変数が一意な名前にあらかじめ変換されているものとする．**例えば，let $x = 1$ in let $x = 2$ in x というプログラムは let $x_1 = 1$ in let $x_2 = 2$ in x_2 というプログラムにあらかじめ変換がなされているものとする．実際に，先に示した変換 $\mathcal{I}$ はすべての束縛変数が一意な名前を持つように変換を行っている．

式 e の変換 $\mathcal{VT}_{\delta,tgt}(e)$ は e の他に変数からオペランドへの部分関数 δ とオペランド tgt を引数として取り,「変数の記憶領域が δ に書いてあると仮定して e を評価した結果を tgt に格納する」仮想マシンコードを生成する．各ケースの説明は以下の通りである．

$\mathcal{VT}_{\delta,tgt}(x)$: x の評価結果を tgt に格納するコードを生成する必要がある．x が格納されている場所は $\delta(x)$ なので，$tgt \leftarrow \delta(x)$ を生成すればよい．

$\mathcal{VT}_{\delta,tgt}(n)$: n の評価結果を tgt に格納するコードを生成するので，$tgt \leftarrow \mathbf{imm}(n)$ を生成すればよい．

$\mathcal{VT}_{\delta,tgt}(\mathsf{true}), \mathcal{VT}_{\delta,tgt}(\mathsf{false})$: 考え方は $\mathcal{VT}_{\delta,tgt}(n)$ と全く同じだが，true は整数定数 1 で，false は整数定数 0 でエンコードしていることに注意．

$\mathcal{VT}_{\delta,tgt}(x_1\ op\ x_2)$: x_1, x_2 を格納している場所はそれぞれ $\delta(x_1), \delta(x_2)$ なので，これらを op で計算して $\mathbf{local}(tgt)$ に格納するコード $tgt \leftarrow op(\delta(x_1), \delta(x_2))$ を生成している．

$\mathcal{VT}_{\delta,tgt}(\mathsf{if}\ x\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2)$: $\mathbf{if}\ \delta(x)\ \mathbf{then\ goto}\ l_1$ 命令で x の値が格納されている $\delta(x)$ に非ゼロの値が入っていれば（すなわち x が true であれば）l_1 にジャンプする．もしここで値がゼロであれば（すなわち x が false であれば）その後ろがそのまま実行されるので，e_2 を評価するコード $\mathcal{VT}_{\delta,tgt}(e_2)$ を書いておき，その後ラベル l_1 のコードを飛び越せるように $\mathbf{goto}\ l_2$ を書いておく．ラベル l_1 以降には $\mathcal{VT}_{\delta,tgt}(e_1)$ で e_1 を評価するコードが書いてある．

$\mathcal{VT}_{\delta,tgt}(\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2)$:まず初めに e_1 を評価して $\delta(x)$ に格納するコード $\mathcal{VT}_{\delta,\delta(x)}(e_1)$ を置く．その後，e_2 の評価結果を tgt に格納するコード $\mathcal{VT}_{\delta,tgt}(e_2)$ を置く．

Definition of $\mathcal{VT}_{\delta,tgt}(e)$ （δ は識別子からオペランドへの部分関数，tgt は $\mathbf{local}(n)$ の形をしたローカル領域のアドレスである．また，以下の定義中 l_1 と l_2 は fresh なラベル名である．）

$$
\begin{array}{l}
\mathcal{VT}_{\delta,tgt}(x) = tgt \leftarrow \delta(x) \\
\mathcal{VT}_{\delta,tgt}(n) = tgt \leftarrow \mathbf{imm}(n) \\
\mathcal{VT}_{\delta,tgt}(\mathsf{true}) = tgt \leftarrow \mathbf{imm}(1) \\
\mathcal{VT}_{\delta,tgt}(\mathsf{false}) = tgt \leftarrow \mathbf{imm}(0) \\
\mathcal{VT}_{\delta,tgt}(x_1\ op\ x_2) = tgt \leftarrow op(\delta(x_1), \delta(x_2)) \\
\mathcal{VT}_{\delta,tgt}(\mathsf{if}\ x\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) = \begin{array}{l} \mathbf{if}\ \delta(x)\ \mathbf{then\ goto}\ l_1 \\ \mathcal{VT}_{\delta,tgt}(e_2) \\ \mathbf{goto}\ l_2 \\ l_1: \\ \mathcal{VT}_{\delta,tgt}(e_1) \\ l_2: \end{array} \\
\mathcal{VT}_{\delta,tgt}(\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2) = \begin{array}{l} \mathcal{VT}_{\delta,\delta(x)}(e_1) \\ \mathcal{VT}_{\delta,tgt}(e_2) \end{array} \\
\mathcal{VT}_{\delta,tgt}(x_1\ x_2) = tgt \leftarrow \mathbf{call}\ \delta(x_1)(\delta(x_2))
\end{array}
$$

Definition of $\mathcal{VT}_{\delta}(d)$

$$
\begin{array}{rcl}
\mathcal{VT}_{\delta}(\mathsf{let\ rec}\ f\ = \mathsf{fun}\ x \rightarrow e) & = & \left(l \parallel \begin{array}{l} \mathcal{VT}_{\delta\cup\delta_1\cup\delta_2,\mathbf{local}(0)}(e) \\ \mathbf{return}(\mathbf{local}(0)) \end{array} \parallel 4n+4 \right) \\
\text{where} & & \begin{array}{rcl} \delta_1 & = & \{x \mapsto \mathbf{param}(1)\} \\ \{x_1, \ldots, x_n\} & = & e \text{ 中に出現する変数の集合} \\ \delta_2 & = & \{x_1 \mapsto \mathbf{local}(4), x_2 \mapsto \mathbf{local}(8), \ldots, x_n \mapsto \mathbf{local}(4n)\} \\ \mathbf{labimm}(l) & = & \delta(f) \end{array}
\end{array}
$$

Definition of $\mathcal{VT}(P)$

$$
\begin{array}{rcl}
\mathcal{VT}((\{d_1, \ldots, d_n\}, e)) & = & \left(\begin{array}{l} \mathcal{VT}_{\delta}(d_1) \\ \ldots \\ \mathcal{VT}_{\delta}(d_n) \end{array} \parallel \begin{array}{l} l_{main}: \\ \mathcal{VT}_{\delta\cup\delta',\mathbf{local}(0)}(e) \\ \mathbf{return}(\mathbf{local}(0)) \end{array} \parallel 4m+4 \right) \\
\text{where} & & \begin{array}{l} \{f_1, \ldots, f_n\} = d_1, \ldots, d_n \text{ で定義されている関数名の集合} \\ \{x_1, \ldots, x_m\} = e \text{ 中の変数の集合} \\ \delta = \{f_1 \mapsto \mathbf{labimm}(f_1), \ldots, f_n \mapsto \mathbf{labimm}(f_n)\} \\ \delta' = \{x_1 \mapsto \mathbf{local}(4), x_2 \mapsto \mathbf{local}(8), \ldots, x_m \mapsto \mathbf{local}(4m)\} \end{array}
\end{array}
$$

図 5.1: $\mathcal{C}$ から $\mathcal{V}$ への変換 $\mathcal{VT}$.

$\mathcal{VT}_{\delta,tgt}(x_1\ x_2)$: 関数呼び出しを行い，その返り値を tgt に格納するコード $tgt \leftarrow \textbf{call}\ \delta(x_1)(\delta(x_2))$ を生成する．ジャンプ先のラベルは $\delta(x_1)$ に格納されている．また，$\delta(x_2)$ に引数が格納されている．

関数定義 d の仮想マシンコード生成を行う変換 $\mathcal{VT}_\delta(d)$ は，d 以外に δ を引数にとる．δ はトップレベルで定義されている関数名を受け取って，それを対応するコードが書かれているラベルオペランド $\textbf{labimm}(l)$ に写像する．$\mathcal{VT}_\delta(\mathsf{let\ rec}\ f\ = \mathsf{fun}\ x \rightarrow e)$ は，その後関数本体 e を評価するコード $\mathcal{VT}_{\delta\cup\delta_1\cup\delta_2,\textbf{local}(0)}(e)$ を生成する．$\delta\cup\delta_1\cup\delta_2$ は δ を以下の二つの写像で拡張したものである．

δ_1 : 仮引数名 x から $\textbf{param}(1)$ への写像.

δ_2 : e 中に現れるすべての変数からそれぞれ固有の記憶領域 $\textbf{local}(i)$ への写像.[4] ここでは，すべての値が 4 バイトで表現できるものとして，各変数に 4 バイトの記憶領域を割り当て，δ_2 を $\{x_1 \mapsto \textbf{local}(4), x_2 \mapsto \textbf{local}(8), \ldots, x_n \mapsto \textbf{local}(4n)\}$ としている．

末尾に e の評価結果（$\textbf{local}(0)$ に格納されている）を $\textbf{return}(\textbf{local}(0))$ で返す．この関数で必要とされるローカルな記憶領域のサイズは $4n$ である．

プログラム $(\{d_1,\ldots,d_n\},e)$ の変換においては，まず各 d_i の変換結果 $\mathcal{VT}_\delta(d_i)$ を生成する．δ は各 d_i で定義されている関数名 f_i からラベル名 $\textbf{labimm}(f_i)$ への写像である．その後メインの式である e を評価するコードを生成すればよい．このコードの先頭にはラベル l_{main} : を生成している．e を評価する際に，e 中の変数のための記憶領域を割り当てる必要があるが，これは上記の関数定義の仮想マシンコード生成と同じ考え方である．

[4] 変換 $\mathcal{I}$ においてすべての束縛変数の名前を一意になるように付け替えたのがここで地味に効いている．