Lecture: An Assembly Language

Marco Patrignani

1 Language

1.1 Syntax

$$\begin{aligned}
&\textit{naturals} \quad \mathbf{n}, \mathbf{pc}, \mathbf{i}, \mathbf{j}, \mathbf{k} \\
&\textit{States } \mathbf{s} ::= \langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle \\
&\textit{Instructions } \mathbf{i} ::= \mathbf{add}\ \mathbf{r_i}\ \mathbf{r_j} \mid \mathbf{const}\ \mathbf{k}\ \mathbf{r_i} \mid \mathbf{jmp}\ \mathbf{r_i} \mid \mathbf{jz}\ \mathbf{r_i} \\
&\qquad \mid \mathbf{load}\ \mathbf{r_i}\ \mathbf{r_j} \mid \mathbf{store}\ \mathbf{r_i}\ \mathbf{r_j} \mid \mathbf{cmp}\ \mathbf{r_i}\ \mathbf{r_j} \mid \mathbf{set}\ \mathbf{r_i}\ \mathbf{r_j} \\
&\textit{Register file } \mathbf{R} ::= \mathbf{r_1} \mapsto \mathbf{n_1}, \mathbf{r_2} \mapsto \mathbf{n_2}, \cdots \\
&\textit{Registers } \mathbf{r} \quad \text{taken from an infinite, denumerable set} \\
&\textit{Flags } \mathbf{f} ::= \mathbf{zf} = \mathbf{b} \qquad \mathbf{b} = \mathbf{true} \mid \mathbf{false} \\
&\textit{Memories } \mathbf{m} ::= \mathbf{n_1} \mapsto \mathbf{m_1}, \mathbf{n_2} \mapsto \mathbf{m_2}, \cdots
\end{aligned}$$

Auxiliaries

$$\frac{\mathbf{pc} \mapsto \mathbf{n} \in \mathbf{m}}{\mathbf{m}(\mathbf{pc}) = \mathbf{n}}\ \text{(Memory Access)} \qquad \frac{\mathbf{r} \mapsto \mathbf{n} \in \mathbf{R}}{\mathbf{R}(\mathbf{r}) = \mathbf{n}}\ \text{(Register Access)}$$

$$\frac{\begin{array}{c}\mathbf{m} = \mathbf{n_1} \mapsto \mathbf{m_1}, \mathbf{n_2} \mapsto \mathbf{m_2}, \cdots, \mathbf{pc} \mapsto \mathbf{n'}, \cdots \\ \mathbf{m'} = \mathbf{n_1} \mapsto \mathbf{m_1}, \mathbf{n_2} \mapsto \mathbf{m_2}, \cdots, \mathbf{pc} \mapsto \mathbf{n}, \cdots\end{array}}{\mathbf{m}[\mathbf{pc} \mapsto \mathbf{n}] = \mathbf{m'}}\ \text{(Memory Update)}$$

$$\frac{\begin{array}{c}\mathbf{R} = \mathbf{r_1} \mapsto \mathbf{m_1}, \mathbf{r_2} \mapsto \mathbf{m_2}, \cdots, \mathbf{r} \mapsto \mathbf{n'}, \cdots \\ \mathbf{R'} = \mathbf{r_1} \mapsto \mathbf{m_1}, \mathbf{r_2} \mapsto \mathbf{m_2}, \cdots, \mathbf{r} \mapsto \mathbf{n}, \cdots\end{array}}{\mathbf{R}[\mathbf{r} \mapsto \mathbf{n}] = \mathbf{R'}}\ \text{(Register Update)} \qquad \frac{}{\|\mathbf{i}\| = \mathbf{n}}\ \text{(Instruction decoding)}$$

1.2 Semantics

$$\frac{\mathbf{m}(\mathbf{pc}) = \|\mathbf{add}\ \mathbf{r_i}\ \mathbf{r_j}\| \quad \mathbf{R}(\mathbf{r_i}) = \mathbf{n_i} \quad \mathbf{R}(\mathbf{r_j}) = \mathbf{n_j} \quad \mathbf{n_i} + \mathbf{n_j} = \mathbf{n_f}}{\langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle \to \langle \mathbf{m}, \mathbf{R}[\mathbf{r_i} \mapsto \mathbf{n_f}], \mathbf{f}, \mathbf{pc} + 1 \rangle}\ \text{(Eval-add)}$$

$$\frac{\mathbf{m}(\mathbf{pc}) = \|\mathbf{const}\ \mathbf{j}\ \mathbf{r_i}\|}{\langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle \to \langle \mathbf{m}, \mathbf{R}[\mathbf{r_i} \mapsto \mathbf{j}], \mathbf{f}, \mathbf{pc} + 1 \rangle}\ \text{(Eval-const)}$$

$$\frac{\mathbf{m}(\mathbf{pc}) = \|\mathbf{jmp}\ \mathbf{r_i}\| \quad \mathbf{R}(\mathbf{r_i}) = \mathbf{n_i}}{\langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle \to \langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{n_i} \rangle}\ \text{(Eval-jmp)}$$

$$\frac{\mathbf{m}(\mathbf{pc}) = \|\mathbf{jz}\ \mathbf{r_i}\| \quad \mathbf{R}(\mathbf{r_i}) = \mathbf{n_i} \quad \mathbf{f} = \mathbf{zf} = \mathbf{true}}{\langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle \to \langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{n_i} \rangle}\ \text{(Eval-jz-true)}$$

$$\frac{m(pc) = \|jz\ r_i\| \quad R(r_i) = n_i \quad f = zf = false}{\langle m, R, f, pc\rangle \to \langle m, R, f, pc + 1\rangle}\ (\text{Eval-jz-false})$$

$$\frac{m(pc) = \|load\ r_i\ r_j\| \quad R(r_i) = n_i \quad m(n_i) = k}{\langle m, R, f, pc\rangle \to \langle m, R[r_j \mapsto k], f, pc + 1\rangle}\ (\text{Eval-load})$$

$$\frac{m(pc) = \|store\ r_i\ r_j\| \quad R(r_i) = n_i \quad R(r_j) = n_j}{\langle m, R, f, pc\rangle \to \langle m[n_j \mapsto n_i], R, f, pc + 1\rangle}\ (\text{Eval-store})$$

$$\frac{m(pc) = \|cmp\ r_i\ r_j\| \quad R(r_i) = n_i \quad R(r_j) = n_j \quad f' = zf = (n_i == n_j ? true : false)}{\langle m, R, f, pc\rangle \to \langle m, R, f, pc + 1\rangle}\ (\text{Eval-cmp})$$

$$\frac{m(pc) = \|set\ r_i\ r_j\| \quad R(r_j) = n_j}{\langle m, R, f, pc\rangle \to \langle m, R[r_i \mapsto n_j], f, pc + 1\rangle}\ (\text{Eval-set})$$

The initial state runs execution from address 0 until it encounters an instruction that it cannot decode, the result value in that case is in r_0.

2 Compiler from the Source to this Target

2.1 Compiler Definition

The compiler $[\![\cdot]\!]$ takes in input: a source expression e, a list of registers K, a list of bindings V, an address where to write the instructions. It returns: a list of instructions is, a register where the output of that expression can be found r, an updated list of registers K', an updated list of bindings V.

$$\begin{aligned}
K &::= \varnothing \mid K, r \\
V &::= \varnothing \mid V, x : r \\
\|K\| &= n \qquad \text{where } K = r_1, \cdots, r_n \\
is = \varnothing \mid is, i & \\
\|is\| &= n \qquad \text{where } is = i_1, \cdots, i_n
\end{aligned}$$

Compiler for whole programs (assuming no instruction decodes to 0):

$$[\![f(x) \mapsto e]\!] = is; set\ r_0\ r_i; 0 \qquad \text{where } [\![e, \varnothing, x : r_0, 0]\!] = is, r_i, K, V$$

Compiler for partial programs:

$$[\![f(x) \mapsto e]\!] = is; set\ r_0\ r_i \qquad \text{where } [\![e, \varnothing, x : r_0, 100]\!] = is, r_i, K, V$$

Assume the context fills the instruction before address 100 and after address $100 + \|is\|$. We don't really model returns for simplicity

$$[\![z, K, V, a]\!] = \varnothing, r_i, K, V \qquad \text{where } x : r_i \in V$$

$$
\begin{aligned}
[\![\mathsf{true}, K, V, a]\!] &= \mathbf{const}\ 0\ r_i, r_i, K, r_i, V && \text{where } i = \|K\| + 1 \\
[\![\mathsf{false}, K, V, a]\!] &= \mathbf{const}\ 1\ r_i, r_i, K, r_i, V && \text{where } i = \|K\| + 1 \\
[\![\mathsf{n}, K, V, a]\!] &= \mathbf{const}\ n\ r_i, r_i, K, r_i, V && \text{where } i = \|K\| + 1 \\
[\![\mathsf{e} + \mathsf{e}', K, V, a]\!] &= is; is'; \mathbf{add}\ r_i\ r_j, r_i, K'', V'' && \text{where } [\![\mathsf{e}, K, V, a]\!] = is, r_i, K', V' \\
& && \quad [\![\mathsf{e}', K', V', a + \|is\|]\!] = is', r_j, K'', V'' \\
& && \quad i = \|K\| + 1 \\
& && \quad j = \|K'\| + 1
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathsf{e} == \mathsf{e}', K, V, a]\!] = \ & is; is'; \mathbf{cmp}\ r_i\ r_j \qquad\qquad , r_j, K'', V'' \\
& \mathbf{const}\ k\ r_{i+1}; \mathbf{jz}\ r_{i+1}; \mathbf{const}\ 0\ r_j \\
\text{where} \quad & [\![\mathsf{e}, K, V, a]\!] = is, r_i, K', V' \\
& [\![\mathsf{e}', K', V', a + \|is\|]\!] = is', r_j, K'', V'' \\
& i = \|K\| + 1 \\
& j = \|K'\| + 1 \\
& k = a + \|is\| + \|is''\| + 5
\end{aligned}
$$

$$
\begin{aligned}
& is; \mathbf{const}\ 0\ r_{i+1}; \mathbf{cmp}\ r_i\ r_{i+1}; \mathbf{const}\ k_1\ r_{i+1}; \mathbf{jz}\ r_{i+1} \\
& is''; \mathbf{set}\ r_i\ r_{i''}; \mathbf{const}\ k_2\ r_{i+1}; \mathbf{jmp}\ r_{i+1} \\
[\![\mathsf{if\ e\ then\ e_1\ else\ e_2}, K, V, a]\!] = \ & is'; \mathbf{set}\ r_i\ r_{i'}; \qquad\qquad , r_i, K'''', V'''' \\
\text{where} \quad & [\![\mathsf{e}, K, V, a]\!] = is, r_i, K', V' \\
& [\![\mathsf{e_1}, K', V', a_1]\!] = is', r_{i'}, K'', V'' \\
& [\![\mathsf{e_2}, K', V', a_2]\!] = is'', r_{i''}, K''', V''' \\
& k_1 = a + \|is\| + 4 + \|is''\| + 1 \\
& k_2 = a + \|is\| + 4 + \|is''\| + 3 + \|is'\| + 1 \\
& a_1 = a + \|is\| + 4 + \|is''\| + 3 \\
& a_2 = a + \|is\| + 4 \\
& K'''' = \mathtt{max}\,(K'', K''') \\
& V'''' = \mathtt{max}\,(V'', V''')
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathsf{let}\ x = \mathsf{e}\ \mathsf{in}\ \mathsf{e}', K, V, a]\!] = \ & is; is', r_{i'}, K'', V'' \\
\text{where} \quad & [\![\mathsf{e}, K, V, a]\!] = is, r_i, K', V' \\
& [\![\mathsf{e}', K', V', x : r_i, a_1]\!] = is', r_{i'}, K'', V'' \\
& a_1 = a + \|is\|
\end{aligned}
$$

3 Examples

Compiling this program:

$$[\![\mathsf{f(x) \mapsto let\ z = true\ in\ let\ y = 2\ in\ if\ z\ then\ 0\ else\ y + 3}]\!]$$

results in this assembly:

100 const 0 r_1	z = true
101 const 2 r_2	y = 2
102 const 0 r_3	if loads true
103 cmp r_1 r_3	if checks z == true
104 const 111 r_3	distance to then
105 jz r_3	if
106 const 3 r_4	else: 3
107 add r_2 r_4	y + 3
108 set r_3 r_2	set to if-result register
109 const 113 r_4	end offset
110 jmp r_4	goto end
111 const 0 r_4	then: 0
112 set r_3 r_4	set to if-result register
113 set r_0 r_3	set to program result register

Compiling this program:

$$[\![f(x) \mapsto x]\!]$$

results in this assembly:

100 set r_0 r_0	x

Compiling this program:

$$[\![f(x) \mapsto \text{if } x \text{ then true else false}]\!]$$

results in this assembly:

100 const 0 r_1	if loads true
101 cmp r_0 r_1	if checks x == true
102 const 108 r_1	distance to then
103 jz r_1	if
104 const 1 r_2	else: false
105 set r_0 r_2	set to if-result register
106 const 110 r_1	end offset
107 jmp r_1	goto end
108 const 0 r_3	then: true
109 set r_0 r_3	set to if-result register
110 set r_0 r_0	set to program result register