Lecture: An Assembly Language

Marco Patrignani

1 Language

1.1 Syntax

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

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

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

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

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

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

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

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

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

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

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

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

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 $\mathbf{K}$, a list of bindings V, an address where to write the instructions. It returns: a list of instructions $\mathbf{is}$, a register where the output of that expression can be found $\mathbf{r}$, an updated list of registers $\mathbf{K'}$, an updated list of bindings V.

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

Compiler for whole programs (assuming no instruction decodes to $\mathbf{0}$):

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

Compiler for partial programs:

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

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

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

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

$$
[\![\mathsf{if\ e\ then\ e_1\ else\ e_2}, \mathbf{K}, V, \mathbf{a}]\!] = \left|
\begin{array}{l}
\mathbf{is}; \mathbf{const\ 0\ r_{i+1}}; \mathbf{cmp\ r_i\ r_{i+1}}; \mathbf{const\ k_1\ r_{i+1}}; \mathbf{jz\ r_{i+1}} \\
\mathbf{is}''; \mathbf{set\ r_i\ r_{i''}}; \mathbf{const\ k_2\ r_{i+1}}; \mathbf{jmp\ r_{i+1}} \\
\mathbf{is}'; \mathbf{set\ r_i\ r_{i'}};
\end{array}
\right. , \mathbf{r_i}, \mathbf{K}'''', V''''
$$

$$
\begin{aligned}
\text{where } \;& [\![\mathsf{e}, \mathbf{K}, V, \mathbf{a}]\!] = \mathbf{is}, \mathbf{r_i}, \mathbf{K}', V' \\
& [\![\mathsf{e_1}, \mathbf{K}', V', \mathbf{a_1}]\!] = \mathbf{is}', \mathbf{r_{i'}}, \mathbf{K}'', V'' \\
& [\![\mathsf{e_2}, \mathbf{K}', V', \mathbf{a_2}]\!] = \mathbf{is}'', \mathbf{r_{i''}}, \mathbf{K}''', V''' \\
& \mathbf{k_1} = \mathbf{a} + \|\mathbf{is}\| + 4 + \|\mathbf{is}''\| + 1 \\
& \mathbf{k_2} = \mathbf{a} + \|\mathbf{is}\| + 4 + \|\mathbf{is}''\| + 3 + \|\mathbf{is}'\| + 1 \\
& \mathbf{a_1} = \mathbf{a} + \|\mathbf{is}\| + 4 + \|\mathbf{is}''\| + 3 \\
& \mathbf{a_2} = \mathbf{a} + \|\mathbf{is}\| + 4 \\
& \mathbf{K}'''' = \mathtt{max}\,(\mathbf{K}'', \mathbf{K}''') \\
& V'''' = \mathtt{max}\,(V'', V''')
\end{aligned}
$$

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

2.2 Compiler Correctness

Let $\Gamma \vdash \gamma$ say that γ binds the same variables of Γ to values of the right type.

Lemma 2.1 (Forward simulation).

$$\begin{array}{l}\text{if } \Gamma \vdash \mathsf{e} : \tau \text{ and } \Gamma \vdash \gamma \text{ and } \mathsf{e}\gamma \hookrightarrow^{*} \mathsf{v} \text{ and } \mathtt{dom}\,(\mathbf{K}) = \mathtt{img}\,(V) \\ \text{ and } \llbracket \mathsf{e}, \mathbf{K}, V, \mathbf{pc} \rrbracket = \mathbf{is}, \mathbf{r_i}, \_, \_ \text{ and } \mathsf{e}\gamma \sim_V \langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle \\ \text{ then } \langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle \to^{\|\mathbf{is}\|} \langle \mathbf{m'}, \mathbf{R'}, \mathbf{f'}, \mathbf{pc'} \rangle \text{ and } \mathsf{v} \sim_{\mathbf{r_i}} \langle \mathbf{m'}, \mathbf{R'}, \mathbf{f'}, \mathbf{pc'} \rangle\end{array}$$

Proof. By induction on the typing derivation of e.

2.2.1 Relations for this Proof

We need a set of cross-language relations.

$$\text{(State Relation (closed))}\quad \frac{\begin{array}{c}\llbracket \mathsf{e}, \varnothing, \mathbf{x} : \mathbf{r_0}, \mathbf{pc} \rrbracket = \mathbf{i_0}, \cdots, \mathbf{i_j}, \_, \_, \_ \\ \forall k \in 0..j \ \mathbf{m(pc + k)} \mapsto \mathbf{i_k}\end{array}}{\mathsf{e} \sim^{c} \langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle}$$

A source closed expression is related to a target state if:

- in the memory, starting at address $\mathbf{pc}$, there is the list of instructions that result of the compilation of e;

$$\text{(Value Relation)}\quad \frac{\mathsf{v} \sim^{0} \mathbf{R(r)}}{\mathsf{v} \sim_{\mathbf{r}} \langle \mathbf{m'}, \mathbf{R'}, \mathbf{f'}, \mathbf{pc'} \rangle}$$

$$\text{(Base Value Relation - true)}\quad \frac{}{\mathsf{true} \sim^{0} \mathbf{0}} \qquad \text{(Base Value Relation - false)}\quad \frac{}{\mathsf{false} \sim^{0} \mathbf{1}}$$

$$\text{(Base Value Relation - n)}\quad \frac{}{\mathsf{n} \sim^{0} \mathbf{n}}$$

A source value is related to a target state at a certain register if:

- in the target state, at the register, there is a number that is in the base relation for values.

$$\text{(State Relation (ope))}\quad \frac{\begin{array}{c}\mathsf{e} \sim^{c} \langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle \\ V \vdash \gamma \sim \mathbf{R}\end{array}}{\mathsf{e}\gamma \sim_V \langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle} \qquad \text{(Substitution Relation - empty)}\quad \frac{}{V \vdash \varnothing \sim \mathbf{R}}$$

$$\text{(Substitution Relation - subst)}\quad \frac{\begin{array}{c}V(\mathsf{x}) = \mathbf{r} \\ \mathsf{v} \sim^{0} \mathbf{R(r)}\end{array}}{V \vdash \gamma; [\mathsf{v} \,/\, \mathsf{x}] \sim \mathbf{R}}$$

An open source expression is related to a target state if:

- the expression is related according to the closed expression relation;
- the list of substitutions γ is accounted for in the registers $\mathbf{R}$.

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	$\mathsf{z = true}$
101 const 2 r_2	$\mathsf{y = 2}$
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	$\mathsf{y + 3}$
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: $\mathsf{0}$
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:

$$[\![\mathsf{f(x) \mapsto x}]\!]$$

results in this assembly:

100 set r_0 r_0	x

Compiling this program:

$$[\![\mathsf{f(x) \mapsto if\ x\ then\ 0\ else\ 1}]\!]$$

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: $\mathsf{1}$
105 set r_0 r_2	set to if-result register
106 const 110 r_1	end offset
107 jmp r_1	goto end

$\mathbf{108\ const\ 0\ r_3}$ then: $\mathsf{0}$

$\mathbf{109\ set\ r_0\ r_3}$ set to if-result register

$\mathbf{110\ set\ r_0\ r_0}$ set to program result register

4 Supporting Fully-Abstract Compilation

In order to support fully-abstract compilation we extend the language with a primitive that zeroes-out all registers except one.

$$\textit{Instructions}\ \mathbf{i} ::= \cdots \mid \mathbf{zero\ r}$$

$$\frac{\mathbf{m(pc)} = \|\mathbf{zero\ r}\| \qquad \mathbf{R(r)} = \mathbf{n} \qquad \forall i \geq 0.\, \mathbf{R}' = \mathbf{r_i} \mapsto \mathbf{0}}{\langle \mathbf{m}, \mathbf{R}, \mathbf{f}, \mathbf{pc} \rangle \to \langle \mathbf{m}, \mathbf{R}'[\mathbf{r} \mapsto \mathbf{n}], \mathbf{f}, \mathbf{pc} + \mathbf{1} \rangle}\ \text{(Eval-zero)}$$

Fully-abstract compiler for partial programs:

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

Note that we avoid many issues by insisting that all source functions are typed at $\mathsf{Nat} \to \mathsf{Nat}$. Any change in any of those types would require the introduction of a typecheck in order for the compiler to be fully-abstract.