A **gadget** is a partial register-machine graph.

It has one entry wire, and one or more exit wires.

The gadget operates on input and output registers specified in the gadget’s name.

The gadget may use other registers, called scratch registers, for temporary storage.

The gadget assumes the scratch registers are initially set to 0, and **must** ensure that they are set back to 0 when the gadget exits.
The gadget “zero $R_0$” sets register $R_0$ to be zero, whatever its initial value:
Gadget: “add $R_1$ to $R_2$”

The gadget “add $R_1$ to $R_2$” adds the initial value of $R_1$ to register $R_2$, storing the result in $R_2$ but restoring $R_1$ to its initial value.
The gadget “copy $R_1$ to $R_2$” copies the value of register $R_1$ into register $R_2$, leaving $R_1$ with its initial value:
Gadget: “copy $R_1$ to $R_2$ and $R_3$”
Gadget: “copy $R_1$ to $R_2$ and $R_3$”

- Entry
- Zero $R_2$
- Add $R_1$ to $R_2$
- Zero $R_3$
- Add $R_1$ to $R_3$
- Exit
Gadget: “copy $R_1$ to $R_2$ and $R_3$”
Gadgets: “multiply $R_1$ by $R_2$ to $R_0$”

We can implement “multiply $R_1$ by $R_2$ to $R_0$” by repeated addition:
The gadget “push $X$ to $L$”:

Given input values $X = x$, $L = \ell$ and $Z = 0$, it returns the output values $X = 0$, $L = \langle x, \ell \rangle = 2^x(2\ell + 1)$ and $Z = 0$: 
\[ L = 2^{x-x}(2\ell + 1), \]
\[ Z + 2L = 2^{x-x}(2\ell + 1) \]
\[ Z = 0 \]

\[ X = x, \]
\[ L = \ell, \]
\[ Z = 0 \]

\[ Z + L = 2^{x-x}(2\ell + 1) \]
\[ L = 2^x(2\ell + 1), \]
\[ Z = 0 \]
The gadget “pop $L$ to $X$”:

If $L = 0$ then return $X = 0$ and go to “empty”. If $L = \langle x, \ell \rangle = n$ then return $X = x$ and $L = \ell$, and go to “done”.

Diagram: [Diagram of gadget with states and transitions labeled appropriately]
\[
\begin{align*}
L &= n, \\
X &= y, \\
Z &= 0
\end{align*}
\]
Gadgets

\[ R_1 = x \rightarrow \text{copy } R_1 \text{ to } R_2 \rightarrow R_1 = R_2 = x \]

\[ X = x, L = \ell \rightarrow \text{push } X \text{ to } L \rightarrow X = 0, L = \langle x, \ell \rangle \]

\[ L = \ell \rightarrow \text{pop } L \text{ to } X \rightarrow \text{done} \rightarrow X = x', L = \ell' \text{ where } \ell = \langle x', \ell' \rangle \]

\[ X = L = \ell = 0 \]
The Universal Register Machine

The *universal register machine* carries out the following computation, starting with $R_0 = 0$, $R_1 = e$ (code of a program), $R_2 = a$ (code of a list of arguments) and all other registers zeroed:

- decode $e$ as a RM program $P$
- decode $a$ as a list of register values $a_1, \ldots, a_n$
- carry out the computation of the RM program $P$ starting with $R_0 = 0$, $R_1 = a_1, \ldots, R_n = a_n$ (and any other registers occurring in $P$ set to 0).
Mnemonics for the registers of $U$ and the role they play in its program:

$R_0$ result of the simulated RM computation (if any).

$R_1 \equiv P$ Program code of the RM to be simulated

$R_2 \equiv A$ list of RM Arguments (or register contents) of the simulated machine

$R_3 \equiv PC$ Program Counter—label number of the current instruction

$R_4 \equiv N$ label number(s) of the Next instruction(s)—also used to hold code of current instruction

$R_5 \equiv C$ code of the Current instruction body

$R_6 \equiv R$ value of the Register to be used by current instruction

$R_7 \equiv S$ and $R_8 \equiv T$ are auxiliary registers.

$R_9$... other scratch registers.
Overall structure of the URM

1 copy $PC$th item of list in $P$ to $N$ (halting if $PC >$ length of list);
goto 2

2 if $N = 0$ then halt, else decode $N$ as $\langle y, z \rangle$; $C ::= y$; $N ::= z$;
goto 3

{at this point either $C = 2i$ is even and current instruction is $R_i^+ \rightarrow L_z$,
or $C = 2i + 1$ is odd and current instruction is $R_i^- \rightarrow L_j, L_k$ where $z = \langle j, k \rangle$}

3 copy $i$th item of list in $A$ to $R$; goto 4

4 execute current instruction on $R$; update $PC$ to next label; restore register values to $A$; goto 1
The Universal Register Machine

START → push $R_0$ to $A$ → copy $P$ to $T$ → pop $T$ to $N$ → pop $A$ to $R_0$ → HALT

push $R$ to $A$ → copy $N$ to $PC$ → R$^-$ → pop $N$ to $PC$ → push $R$ to $S$

copy $P$ to $T$ → empty → R$^+$

copy $N$ to $PC$ → empty → N$^+$ → empty → done

copy $N^-$ to $C$ → empty → done

copy $P$ to $T$ → empty → done

push $R$ to $S$ → empty → done

pop $S$ to $R$ → done

empty

done
Universal Register Machines