Universal register machine, U

High-level specification

Universal RM U carries out the following computation, starting with $\mathtt{R}_0 = 0$, $\mathtt{R}_1 = e$ (code of a program), $\mathtt{R}_2 = a$ (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 $\mathtt{R}_0 = 0, \mathtt{R}_1 = a_1, \ldots, \mathtt{R}_n = a_n$ (and any other registers occurring in P set to 0).

Mnemonics for the registers of U and the role they play in its program:

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

$R_2 \equiv A$ code of current register contents of simulated RM

$R_3 \equiv PC$ program counter—number of the current instruction (counting from 0)

$R_4 \equiv N$ code of the current instruction body

$R_5 \equiv C$ type of the current instruction body

$R_6 \equiv R$ current value of the register to be incremented or decremented by current instruction (if not HALT)

$R_7 \equiv S$, $R_8 \equiv T$ and $R_9 \equiv Z$ are auxiliary registers.

Overall structure of U's program

$\boxed{1}$ copy PCth item of list in P to N; goto $\boxed{2}$

$\boxed{2}$ if $\mathtt{N} = 0$ then copy 0th item of list in A to $\mathtt{R}_0$ and halt, else (decode N as $\langle\langle y, z \rangle\rangle$; $\mathtt{C} ::= y$; $\mathtt{N} ::= z$; goto $\boxed{3}$)

{at this point either $\mathtt{C} = 2i$ is even and current instruction is $\mathtt{R}_i^+ \to \mathtt{L}_z$,
or $\mathtt{C} = 2i + 1$ is odd and current instruction is $\mathtt{R}_i^- \to \mathtt{L}_j, \mathtt{L}_k$ where $z = \langle j, k \rangle$}

$\boxed{3}$ copy ith item of list in A to R; goto $\boxed{4}$

$\boxed{4}$ execute current instruction on R; update PC to next label; restore register values to A; goto $\boxed{1}$

Overall structure of U's program

1 copy $\mathtt{PC}$th item of list in $\mathtt{P}$ to $\mathtt{N}$; goto 2

2 if $\mathtt{N} = 0$ then copy 0th item of list in $\mathtt{A}$ to $\mathtt{R}_0$ and halt, else (decode $\mathtt{N}$ as $\langle\langle y, z\rangle\rangle$; $\mathtt{C} ::= y$; $\mathtt{N} ::= z$; goto 3)

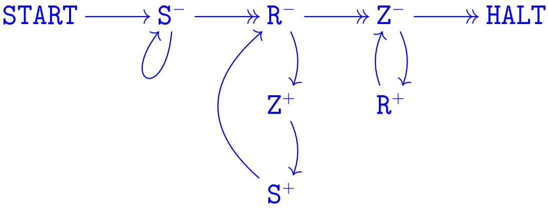
{at this point either $\mathtt{C} = 2i$ is even and current instruction is $\mathtt{R}_i^+ \to \mathtt{L}_z$, or $\mathtt{C} = 2i+1$ is odd and current instruction is $\mathtt{R}_i^- \to \mathtt{L}_j, \mathtt{L}_k$ where $z = \langle j, k\rangle$}

3 copy ith item of list in $\mathtt{A}$ to $\mathtt{R}$; goto 4

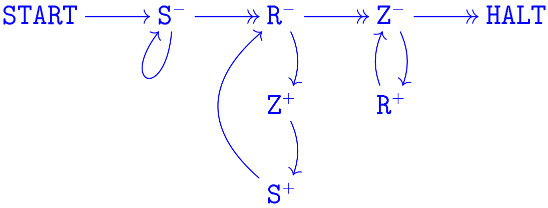
4 execute current instruction on $\mathtt{R}$; update $\mathtt{PC}$ to next label; restore register values to $\mathtt{A}$; goto 1

To implement this, we need RMs for manipulating (codes of) lists of numbers...

The program START→ $\boxed{\texttt{S ::= R}}$ →HALT
to copy the contents of R to S can be implemented by



The program $\texttt{START}\rightarrow\boxed{\texttt{S} ::= \texttt{R}}\rightarrow\texttt{HALT}$
to copy the contents of $\texttt{R}$ to $\texttt{S}$ can be implemented by

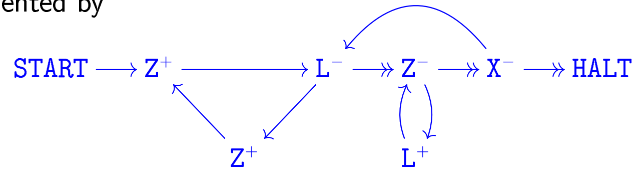


precondition:
$\texttt{R} = x$
$\texttt{S} = y$
$\texttt{Z} = 0$

postcondition:
$\texttt{R} = x$
$\texttt{S} = x$
$\texttt{Z} = 0$

The program START→ *push* X *to* L →HALT

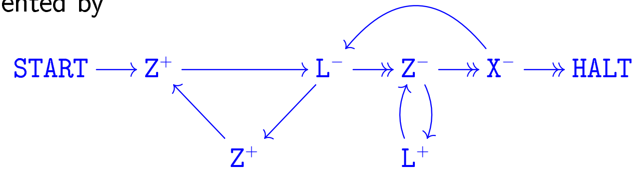
to carry out the assignment $(\mathtt{X}, \mathtt{L}) ::= (0, \mathtt{X} :: \mathtt{L})$ can be implemented by



$2^{\mathtt{X}}(2\mathtt{L}+1)$

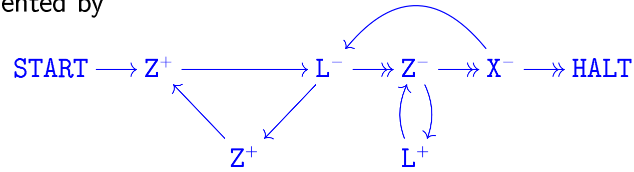
The program START$\rightarrow$ *push* X *to* L $\rightarrow$HALT

to carry out the assignment $(\mathtt{X}, \mathtt{L}) ::= (0, \mathtt{X} :: \mathtt{L})$ can be implemented by



The program START→ *push* X *to* L →HALT

to carry out the assignment $(\mathtt{X}, \mathtt{L}) ::= (0, \mathtt{X} :: \mathtt{L})$ can be implemented by



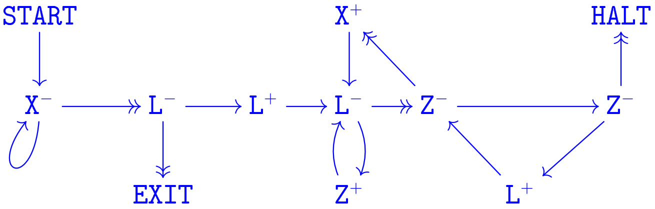
precondition:
$\mathtt{X} = x$
$\mathtt{L} = \ell$
$\mathtt{Z} = 0$

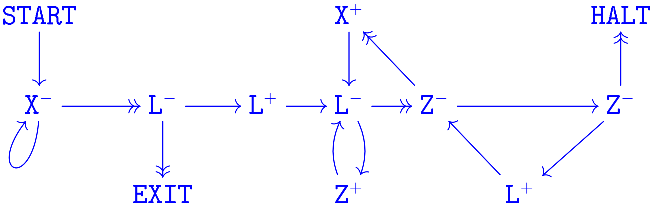
postcondition:
$\mathtt{X} = 0$
$\mathtt{L} = \langle\langle x, \ell \rangle\rangle = 2^x(2\ell + 1)$
$\mathtt{Z} = 0$

The program START→ [*pop* L *to* X] →HALT / ↠EXIT specified by

"*if* L $= 0$ *then* (X $::= 0$; *goto* EXIT) *else*
let L $= \langle\langle x, \ell \rangle\rangle$ *in* (X $::= x$; L $::= \ell$; *goto* HALT)"

can be implemented by




START
X$^-$
L$^-$
L$^+$
X$^+$
L$^-$
Z$^-$
HALT
Z$^-$
EXIT
Z$^+$
L$^+$

Overall structure of U's program

$\boxed{1}$ copy $\mathtt{PC}$th item of list in $\mathtt{P}$ to $\mathtt{N}$; goto $\boxed{2}$

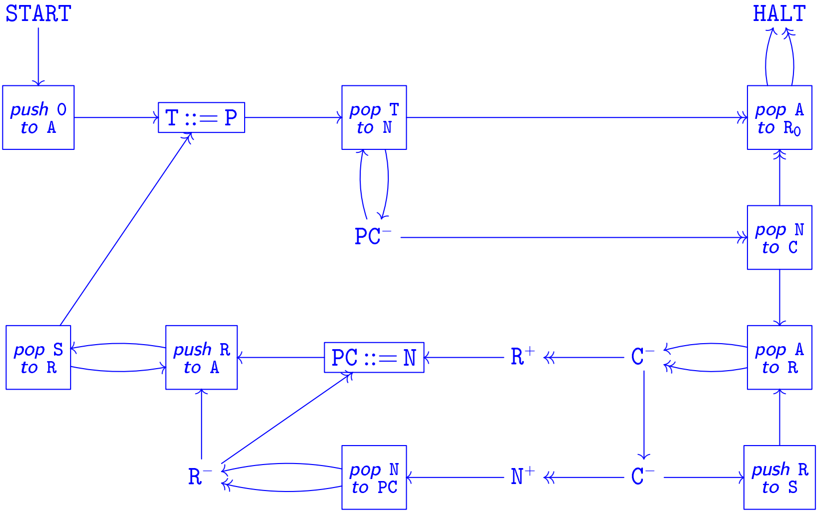
$\boxed{2}$ if $\mathtt{N} = 0$ then copy 0th item of list in $\mathtt{A}$ to $\mathtt{R}_0$ and halt, else (decode $\mathtt{N}$ as $\langle\!\langle y, z \rangle\!\rangle$; $\mathtt{C} ::= y$; $\mathtt{N} ::= z$; goto $\boxed{3}$)

{at this point either $\mathtt{C} = 2i$ is even and current instruction is $\mathtt{R}_i^+ \to \mathtt{L}_z$,
or $\mathtt{C} = 2i + 1$ is odd and current instruction is $\mathtt{R}_i^- \to \mathtt{L}_j, \mathtt{L}_k$ where $z = \langle j, k \rangle$}

$\boxed{3}$ copy ith item of list in $\mathtt{A}$ to $\mathtt{R}$; goto $\boxed{4}$

$\boxed{4}$ execute current instruction on $\mathtt{R}$; update $\mathtt{PC}$ to next label; restore register values to $\mathtt{A}$; goto $\boxed{1}$

The program for U



The Halting Problem

Computable functions

Recall:

Definition. $f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ is (register machine) computable if there is a register machine M with at least $n+1$ registers $\mathtt{R}_0$, $\mathtt{R}_1$, ..., $\mathtt{R}_n$ (and maybe more)
such that for all $(x_1, \ldots, x_n) \in \mathbb{N}^n$ and all $y \in \mathbb{N}$,

the computation of M starting with $\mathtt{R}_0 = 0$, $\mathtt{R}_1 = x_1$, ..., $\mathtt{R}_n = x_n$ and all other registers set to 0, halts with $\mathtt{R}_0 = y$

if and only if $f(x_1, \ldots, x_n) = y$.

Definition. A register machine H decides the Halting Problem if for all $e, a_1, \ldots, a_n \in \mathbb{N}$, starting H with

$$\mathtt{R}_0 = 0 \qquad \mathtt{R}_1 = e \qquad \mathtt{R}_2 = \ulcorner [a_1, \ldots, a_n] \urcorner$$

and all other registers zeroed, the computation of H always halts with $\mathtt{R}_0$ containing 0 or 1; moreover when the computation halts, $\mathtt{R}_0 = 1$ if and only if

> *the register machine program with index e eventually halts when started with $\mathtt{R}_0 = 0, \mathtt{R}_1 = a_1, \ldots, \mathtt{R}_n = a_n$ and all other registers zeroed.*

Definition. A register machine H decides the Halting Problem if for all $e, a_1, \ldots, a_n \in \mathbb{N}$, starting H with

$$\mathtt{R}_0 = 0 \qquad \mathtt{R}_1 = e \qquad \mathtt{R}_2 = \ulcorner [a_1, \ldots, a_n] \urcorner$$

and all other registers zeroed, the computation of H always halts with $\mathtt{R}_0$ containing 0 or 1; moreover when the computation halts, $\mathtt{R}_0 = 1$ if and only if

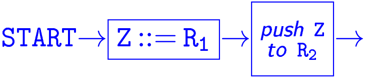
> *the register machine program with index e eventually halts when started with $\mathtt{R}_0 = 0, \mathtt{R}_1 = a_1, \ldots, \mathtt{R}_n = a_n$ and all other registers zeroed.*

Theorem. No such register machine H can exist.

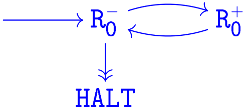
Proof of the theorem

Assume we have a RM H that decides the Halting Problem and derive a contradiction, as follows:

- Let H' be obtained from H by replacing START→ by

  (where Z is a register not mentioned in H's program).
- Let C be obtained from H' by replacing each HALT (& each erroneous halt) by

  .
- Let $c \in \mathbb{N}$ be the index of C's program.

Proof of the theorem

Assume we have a RM H that decides the Halting Problem and derive a contradiction, as follows:

$$C \text{ started with } \mathtt{R}_1 = c \text{ eventually halts}$$
$$\text{if \& only if}$$
$$H' \text{ started with } \mathtt{R}_1 = c \text{ halts with } \mathtt{R}_0 = 0$$

Proof of the theorem

Assume we have a RM H that decides the Halting Problem and derive a contradiction, as follows:

$$C \text{ started with } \mathtt{R}_1 = c \text{ eventually halts}$$
$$\text{if \& only if}$$
$$H' \text{ started with } \mathtt{R}_1 = c \text{ halts with } \mathtt{R}_0 = 0$$
$$\text{if \& only if}$$
$$H \text{ started with } \mathtt{R}_1 = c, \mathtt{R}_2 = \ulcorner [c] \urcorner \text{ halts with } \mathtt{R}_0 = 0$$

Proof of the theorem

Assume we have a RM H that decides the Halting Problem and derive a contradiction, as follows:

$$C \text{ started with } \mathtt{R}_1 = c \text{ eventually halts}$$
$$\text{if \& only if}$$
$$H' \text{ started with } \mathtt{R}_1 = c \text{ halts with } \mathtt{R}_0 = 0$$
$$\text{if \& only if}$$
$$H \text{ started with } \mathtt{R}_1 = c, \mathtt{R}_2 = \ulcorner [c] \urcorner \text{ halts with } \mathtt{R}_0 = 0$$
$$\text{if \& only if}$$
$$\mathit{prog}(c) \text{ started with } \mathtt{R}_1 = c \text{ does not halt}$$

Proof of the theorem

Assume we have a RM H that decides the Halting Problem and derive a contradiction, as follows:

$$C \text{ started with } \mathtt{R}_1 = c \text{ eventually halts}$$
$$\text{if \& only if}$$
$$H' \text{ started with } \mathtt{R}_1 = c \text{ halts with } \mathtt{R}_0 = 0$$
$$\text{if \& only if}$$
$$H \text{ started with } \mathtt{R}_1 = c, \mathtt{R}_2 = \ulcorner [c] \urcorner \text{ halts with } \mathtt{R}_0 = 0$$
$$\text{if \& only if}$$
$$prog(c) \text{ started with } \mathtt{R}_1 = c \text{ does not halt}$$
$$\text{if \& only if}$$
$$C \text{ started with } \mathtt{R}_1 = c \text{ does not halt}$$

Proof of the theorem

Assume we have a RM H that decides the Halting Problem and derive a contradiction, as follows:

C started with $\mathtt{R}_1 = c$ eventually halts

if & only if

H' started with $\mathtt{R}_1 = c$ halts with $\mathtt{R}_0 = 0$

if & only if

H started with $\mathtt{R}_1 = c, \mathtt{R}_2 = \ulcorner [c] \urcorner$ halts with $\mathtt{R}_0 = 0$

if & only if

$prog(c)$ started with $\mathtt{R}_1 = c$ does not halt

if & only if

C started with $\mathtt{R}_1 = c$ does not halt

—contradiction!