TrAL: The Truman Assembly Language

TruPro

The Truman processor is a hypothetical machine that we will refer to as the TruPro. TruPro has four general purpose registers (R0, R1, R2 and R3) and one special purpose register (IP). The general purpose registers can hold constants, values loaded from memory, or addresses. IP can hold only addresses. The TruPro's address space is organized into locations called words. Words can hold arbitrary binary data. The address space extends from address 0 to some bounded but unknown maximum address.

R3 is the stack pointer and should not be used except to access the stack. IP is the instruction pointer. It contains the address of the next instruction to be executed and is not directly accessible to the programmer.

TrAL

TrAL is the TRuman Assembly Language for the TruPro. Labels in TrAL match the regular expression
`[_a-zA-Z0-9]`$^+$.

By convention, labels that mark memory locations that hold variables begin with letters and have the same name as the variables. Labels that mark instructions (that is, branch targets) begin with an underscore.

Addressing Modes

TruPro supports five addressing modes: immediate, register direct, memory direct, register indirect and relative. The following table explains the meaning of, and TrAL notation for, each of these addressing modes.

Addressing Mode	**TrAL Notation**	**Example**	**meaning**
Immediate	#n	move R0, #1	Load R0 with constant value 1
Register Direct	Rn	move R0, R1	Load R0 with contents of R1
Memory Direct	Memory_label	move R0, var	Load R0 with contents of memory location that has label "var"
Register Indirect	(Rn)	move R0, (R1)	Load R0 with the contents of the memory location whose address is stored in R1.
Relative	(Rn, #m)	move R0, (R1, #-2)	Load R0 with the contents of the memory location whose address is two words before the address stored in R1

Instruction Set

The TruPro implements a simple two-address load-store architecture with 14 instructions. There are no memory to memory data movement instructions. The instructions, the addressing modes they support, and the meaning of each instruction are shown in the following table.

With the exception of the "move" instruction, every two-address instruction requires a register in direct mode as the destination (first) address of the instruction. The address of the src may be any of the listed addressing modes. For one-address instructions, the allowed addressing modes for the single address are shown.

Move

The move instructions is unique in that it allows either the source or destination address to be of any mode, provided that the other address is register direct.

If both addresses are register direct, the first address is the destination. Otherwise, the address that is register direct is the destination.

	Allowable modes for one-address instructions and the second address of two-address instructions						
Instruction	**Imm**	**RegD**	**MemD**	**RegI**	**Rel**	**Meaning**	**Example**
move Rn, src	X	X	X	X	X	Rn←src	move R0, #1
move dest, Rn		X	X	X	X	dest←Rn	move var, R1
add Rn, src	X	X	X	X	X	Rn←Rn + src	add R1, R2
sub Rn, src	X	X	X	X	X	Rn←Rn - src	sub R1, m
mul Rn, src	X	X	X	X	X	Rn←Rn * src	mul R1, (R2)
div Rn, src	X	X	X	X	X	Rn←Rn / src	div R1, (R2, #1)
neg Rn		X				Rn←Rn * -1	neg R0
not Rn		X				if Rn == 0 Rn←1 else Rn←0	not (R2)
lea Rn, src			X			Rn←address_of(src)	lea Rn, m
brun dest	X	X				IP←dest	brun _L0
brez Rn, dest	X	X				if Rn == 0 IP←dest	brez R0, _L2
brpo Rn, dest	X	X				if Rn > 0 IP←dest	brpo R1, _myloop
brne Rn, dest	X	X				if Rn < 0 IP←dest	brne R2, R0
outb Rn		X				print Rn to console as int	outb R0
halt						halt the program	halt

Data Directives

A data allocation directive is used to reserve memory for variables and other values. It begins with an optional label and colon. Following the optional label is the keyword `data` and an integer detailing the number of memory words to reserve.

For example, to allocate one word of memory and attach the label "x" to the memory location, use this directive:

```
x: data 1
```

We could allocate an array with 16 memory locations using:

```
an_array: data 16
```

Instruction Format

An instruction begins with an optional label. The label must be followed by a colon. Following the optional label and colon is the instruction mnemonic followed by the operands. If there are two operands in the instruction, they are separated by commas.

For example, the following instruction is found at the label "_loop4_start". It loads the value stored in the memory location "x" into register 1.

```
_loop4_start: move R1, X
```

Comments

A comment begins with a semicolon and continues to the end of the line.

```
_prequel0: move R1, (R3, #-4)      ; Load parameter into R1
```

Code and Data Layout

Code and data are loaded into the TruPro address space beginning at address 0. Once loaded, the program begins executing the instruction located at that address. Execution continues until the processor executes the `halt` instruction, encounters an unknown instruction, or generates an exception.

The TruPro recognizes the following exceptions:

- Division by zero
- Unknown instruction
- Illegal addressing mode

Here is a short example program that adds the contents of memory locations a and b, stores the result in c and then prints the value out.

```
_Start: move R0, a   ; load contents of a into R0
        move R1, b   ; load contents of b into R1
        add R0, R1   ; compute sum R0 = R0 + R1
        move c, R0   ; store R0 into c
        outb R0      ; print R0 to console
        halt         ; stop the program
a:      data 1       ; reserve memory for a,
b:      data 1       ; b, and
c:      data 1       ; c
```