Emulators on the HP Prime Part 2 : Turing

Mark Power, mark.power@btinternet.com

In my first article on emulators on the Prime I presented a Chip-8 emulator that emulated systems from the 1970s. This time, following an inspiring trip to The National Museum of Computing (http://www.tnmoc.org) next door to Bletchley Park, I'm going further back to 1936, the dawn of modern computing and Alan Turing's theoretical machine.

I first learnt about Alan Turing and his machines when studying Computer Science in 1984. The concept of a Turing Machine was explained to us and we had homework to produce some simple programs to run on the Computer Science Department's Turing Machine Emulator. Being the impatient type, I didn't want to cycle back to the college and type my programs into the ICL 2960[1] mainframe to try them out, so having just got a HP-41CV and Extended Functions Module I implemented possibly the world's slowest Turing Machine emulator. If you were at the HPCC Mini-Conference in 2009 you might remember me showing this. If you want to run it on your HP41, look up Datafile V28N5.

Turing Machine

Hopefully you'll have come across the concept of a Turing Machine, but in case you haven't here's a recap.

The machine consists of:

- A tape that contains characters that can be read and written by a tape head. The tape can be moved left or right, and can extend to any length. It is a theoretical machine after all.
- A device for holding the current state.
- A table of tuples that are interpreted as the machine is run and determine whether to write a character to tape, or move the tape left or right.

Turing Machine implementations commonly use 5-tuples, but the version I played with used simplified 4-tuples described by Emil Post in 1947[2] of the form:

currentState characterReadFromTape characterToWrite/L/R newState

States are represented in the form Q1, Q2… Qn.

The 3rd part of the tuple either writes the given character to the tape in the current position, or moves the tape, or perhaps more accurately the tape head, left or right.

To emphasise the theoretical nature of the Turing Machine, we were encouraged not to use binary for values on the tape that I've seen in some recent examples (look up Raspberry Pi Turing Machine on the Internet). Instead we used repeated

[1] They are putting together an operational ICL 2966 in The National Museum of Computing

[2] https://en.wikipedia.org/wiki/Post–Turing_machine

1s to represent positive integers. So a tape of "1" represents the integer 0, "11" represents 1, "111" represents 2, "1111" represents 3, etc.

An example tuple to move right along a tape containing 1s until it gets to the end of the string of 1s would be:

Q1 1 R Q1

Assuming we start n state Q1 and there is a 1 under the tape head, move the head right along the tape and go to state Q1. The machine then repeats this operation until it gets to the end of the tape. As there is no tuple describing what to do next it stops.

If you want it to do something else you would add another tuple like:

Q1 B R Q2

In this implementation B represents a blank position, so the combination of the two tuples moves along a tape of 1s until the end and then goes into state Q2.

Duplicate

A complete example of the 4-tuples that duplicates a number on a tape is:

Q1 1 R Q1	Move over the first number
Q1 B L Q2	When we get to the B (blank) at the end move left
Q2 1 M Q2	Place Mark: Replace the 1 with a M (mark)
Q2 M R Q3	Move right past the M (mark)
Q3 1 R Q3	Move over any 1s
Q3 B R Q4	Move right over the B at the end of the number
Q4 1 R Q4	Move over the 1s that are the duplicate
Q4 B 1 Q5	At the end of the duplicate put in an extra 1
Q5 1 L Q5	Now move back to the left over all the 1s
Q5 B L Q5	and Bs
Q5 M 1 Q6	Until you get to the M (mark), replace it with a 1
Q6 1 L Q2	Go left and if over a 1 jump to Place Mark above
	If we're at the start of the tape, we try to find tuple
	"Q2 B x Qn" but as that doesn't exist, stop the machine

Example:

Starting with "1111" on the tape, with the tape head at the leftmost "1" and in state Q1, this table of tuples gives a result tape of "B1111B1111"

This concept of putting a mark on the tape and using it while shuttling backwards and forwards is the basis of multiplication and division.

Multiplication

Q1 1 R Q2	Q5 1 Y Q5	Q8 1 L Q8	Q9 X 1 Q1
Q2 1 X Q2	Q5 Y R Q6	Q8 B L Q8	Q2 B R Q10
Q2 X R Q3	Q6 1 R Q6	Q8 Y 1 Q4	Q10 1 R Q10
Q3 1 R Q3	Q6 B R Q7	Q5 B L Q9	Q10 B R Q11
Q3 B R Q4	Q7 1 R Q7	Q9 B L Q9	Q11 1 R Q11
Q4 1 R Q5	Q7 B 1 Q8	Q9 1 L Q9	Q11 B 1 Q12

Description:

Loop 1: Move right and put an X marker on the next 1. Move right until over the second number on the tape.

Loop 2: Go right over "1" and place a Y marker. Go all the way right to the answer area and place a 1. Go left back to the Y marker, replace it with a 1 and go to Loop 2. When the Y is at the end of the second number, over a B (blank), back up to the X, replace it with a "1" and go to Loop 1.

So you'll see the machine shuttle backwards and forwards duplicating the second number, the number of times given by the first number

Example:

Starting with "1111B11111" on the tape, the head at the leftmost character and in a state of Q1, this set of tuples fives a result tape of "1111B11111B1111111111111" i.e. 3x4=12.

Integer Division

Q1 1 R Q2	Q9 B L Q10	Q14 1 X Q15	Q20 1 R Q20
Q2 B R Q3	Q10 1 L Q10	Q15 X R Q15	Q20 B 1 Q5
Q3 1 R Q3	Q10 B L Q10	Q15 1 R Q15	Q21 1 Y Q12
Q3 B R Q4	Q10 X 1 Q11	Q15 B R Q15	Q21 B R Q22
Q4 B 1 Q5	Q11 1 L Q11	Q15 Y 1 Q16	Q22 B 1 Q23
Q2 1 R Q6	Q11 B E Q5	Q16 1 R Q21	Q22 1 R Q22
Q6 1 R Q6	Q9 1 Y Q12	Q14 B R Q17	Q23 1 L Q23
Q6 B L Q7	Q12 1 L Q12	Q17 1 R Q17	Q23 B L Q24
Q7 1 X Q7	Q12 B L Q12	Q17 B R Q17	Q24 1 L Q24
Q7 X R Q8	Q12 Y L Q12	Q17 Y 1 Q18	Q24 B R Q8
Q8 B R Q8	Q12 X 1 Q13	Q18 1 R Q18	
Q8 1 R Q9	Q13 1 L Q14	Q18 B R Q20	

Description:

Like multiply but in reverse, this shuttles backwards and forwards putting a 1 into the answer (quotient) area every time it finds that the divisor exists within the dividend. Two markers are used like in the multiply

Examples:

“1111111B111” gives “B1111111B111B1111” i.e. 6/2=3

“111111B111” gives “B111111B111B111” i.e. 5/2=2

“11B1” gives “E11B1B” with the “E” indicating a divide by zero error

Prime Implementation

For a change, this is one of my smaller Prime programs. I wrote it directly on the Prime whilst on holiday, so if you have an hour or so to kill you can type it in. Alternatively download the program and the sample tuples from http://hpcc.org/programs/hpprime/Turing/Turing.zip

Because I typed this in on the Prime and the orange alphabetic characters are so difficult to see in low energy lit rooms, I have abbreviated all the variable names. So:

- p is the program, or table of tuples
- cs is the current state and ns is the new state
- tape is the tape of characters
- tp is the tape position under the head
- b is the character which is used for blank positions on the tape and to separate numbers on the tape
- inst is the instruction to perform: “R” moves the head right, “L” moves the head left, any other character is written to tape
- ct is the current character read from the tape/under the tape head

The table of tuples are held in the Notes area to keep them separate from the emulator. I haven’t gone the whole hog and turned the emulator into an App, unlike the Chip-8 emulator.

Syntax checking isn’t implemented so you must be careful typing in the tuples. Open the Notes app (press Shift 0), press New and enter a suitable name such as “TM DUP”. Then enter the tuples, one per line with a single space between the different parts.

Remember that the emulator will stop if the state and character under the tape head are not represented in the list of tuples.

```
#pragma mode(separator(.,;) integer(h32))

p,ns,ct,tape,tp,q;
inst,mode,cs,b;
black:=RGB(0,0,0);
red:=RGB(255,0,0);
white:=RGB(255,255,255);
modes:={"Key press","Slow","Medium","Fast","Very fast"};

WaitForRelease()
BEGIN
        REPEAT
        UNTIL STRING(MOUSE)="{{},{}}";

        REPEAT
        UNTIL GETKEY=-1;
END;

Init(filename)
BEGIN
        DIMGROB_P(G2,320,240);

        p:=Notes(filename);
        cs:="Q1";
        ns:="";
        ct:="";
        tp:=1;
        b:="B";
        q:="";
        inst:="";
        WaitForRelease();
END;
```

```
Fetch()
BEGIN
        LOCAL ss,sq,f,rhs,keepRunning;

        ct:=MID(tape,tp,1);
        ss:=cs+" "+ct+" ";
        sq:=INSTRING(p,ss);

        IF sq THEN
                // Found start of quad
                q:=MID(p,sq,14);
                f:=INSTRING(q,CHAR(10));
                IF f THEN
                        // Truncate quad to EOL
                        q:=LEFT(q,f-1);
                END;

                rhs:=MID(q,SIZE(ss)+1);
                inst:=LEFT(rhs,1);
                ns:=MID(rhs,3);
                keepRunning:=1;
        ELSE
                keepRunning:=0;
        END;

        RETURN(keepRunning);
END;

MoveRight()
BEGIN
        tp:=tp+1;

        // Extend tape if pointer is past end
```

```
        IF tp>SIZE(tape) THEN
                tape:=tape+b;
        END;
END;

MoveLeft()
BEGIN
        IF tp==1 THEN
                // Extend tape to left
                tape:=b+tape;
        ELSE
                tp:=tp-1;
        END;
END;

WriteToTape()
BEGIN
        tape(tp):=inst;
END;

Execute()
BEGIN
        CASE
                IF inst=="R" THEN MoveRight() END;
                IF inst=="L" THEN MoveLeft() END;
                DEFAULT WriteToTape();
        END;

        cs:=ns;
END;
```

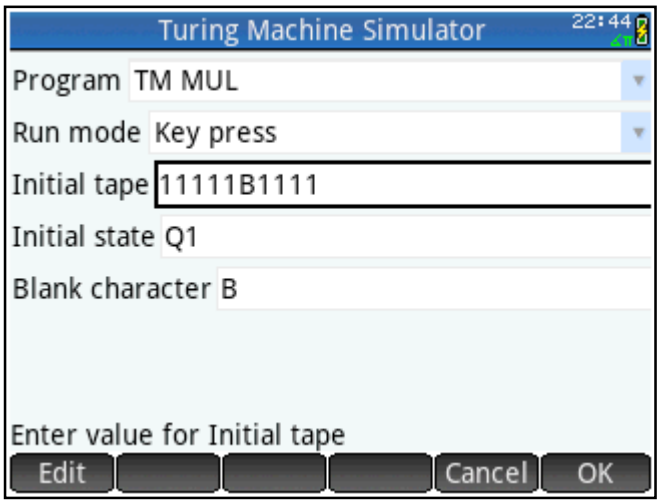
```
Output(m)
BEGIN
        LOCAL ls:="";
        LOCAL ms:="";
        LOCAL rs:="";
        LOCAL t:="";

        CASE
                IF tp==1 THEN
                        ms:=LEFT(tape,1);
                        rs:=MID(tape,2);
                 END;

                IF tp==SIZE(tape) THEN
                        ls:=LEFT(tape,SIZE(tape)-1);
                        ms:=MID(tape,SIZE(tape));
                 END;

                 DEFAULT
                        ls:=LEFT(tape,tp-1);
                        ms:=MID(tape,tp,1);
                        rs:=MID(tape,tp+1);
                 END;

        t:=ls+"["+ms+"]"+rs;

        RECT(G2,white);
        TEXTOUT_P(q,G2,0,0,2,black);
        TEXTOUT_P(t,G2,0,20,2,black);
        TEXTOUT_P(m,G2,0,40,2,black);
        BLIT_P(G0,G2);
END;
```

```
HandleMode()
BEGIN
	CASE
		IF mode==1 THEN WAIT(-1); END;
		IF mode==2 THEN WAIT(1); END;
		IF mode==3 THEN WAIT(0.5); END;
		IF mode==4 THEN WAIT(0.25); END;
		DEFAULT
		 // As fast as possible
	END;

	WaitForRelease();
END;

EXPORT Turing()
BEGIN
	LOCAL filename,v;
	tape:="1";
	cs:="Q1";
	b:="B";
	v:=INPUT({{filename,Notes()}, {mode,modes}, {tape,[2]}, {cs,[2]}, {b,[2]}}, "Turing Machine Simulator", {"Program", "Run mode", "Initial tape", "Initial state", "Blank character"});

	IF v THEN
		Init(filename);
		Output("");
		HandleMode();
		WHILE Fetch() DO
			Output("");
			Execute();
			HandleMode();
```

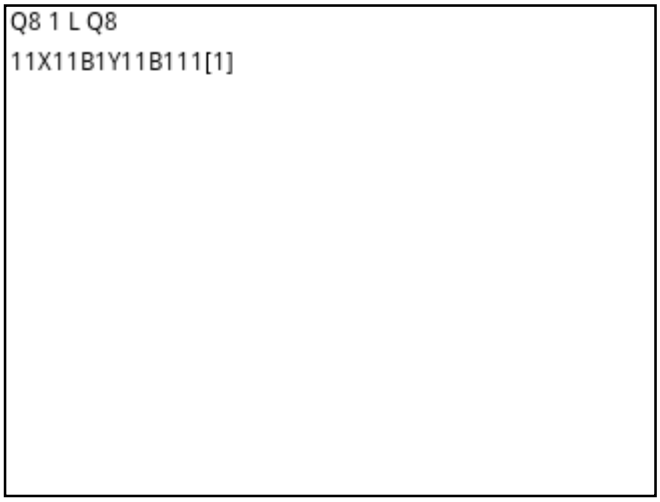
```
            END;
            Output("Finished");
            WaitForRelease();
            FREEZE;
        END;
        RETURN(tape);
END;
```

Instructions

Type in or download the example tuples and paste them into the Communication Kit with your Prime connected. I called them “TM DUP”, “TM MUL” and “TM DIV”. These will appear in setup screen when you run the Turing program.



Start screen where you can select the program to run, whether you want to manually step through the program or let it run automatically and at what speed, the tape you want to start with, the initial state and character used for blanks on the tape.

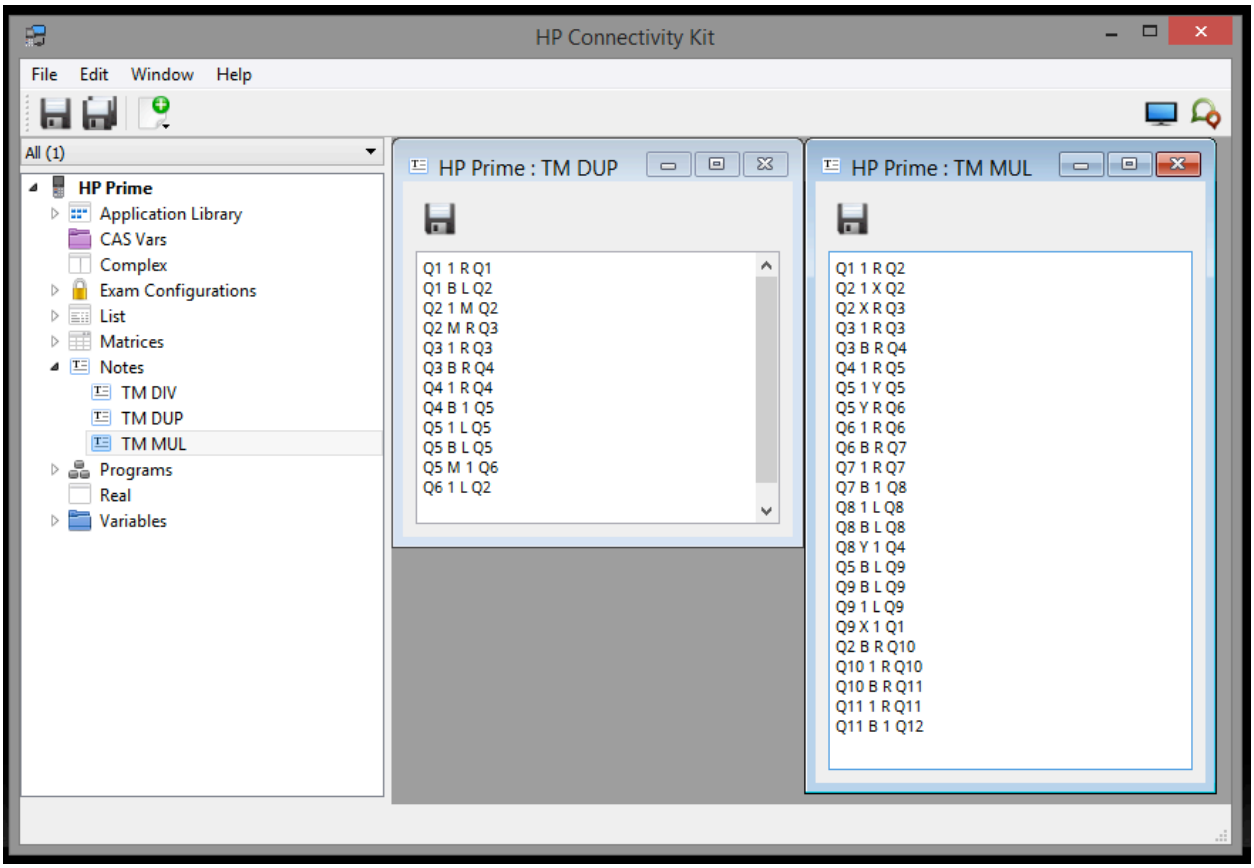


The multiplication program part way through showing the tuple just executed, the tape with X and Y markers, the partial result and the tape head, indicated by [ ],at the rightmost end of the tape having just written a 1.

```
Q11 B 1 Q12
11111B1111B111111111111[1]
Finished
```

The multiplication ended. In this case showing the result of multiplying 4 by 3 and showing 12 (represented by 13 1s). When you press a key to leave the program the tape is returned to the stack.

The picture below shows the Notes in the Connectivity Kit with the Turing Machine tuples for Duplicate and Multiply.



Alan Turing, OBE, FRS
1912-1954