Computer Science Courses
ANU College of Engineering & Computer Science

rPeANUt

rPeANUt is a RISC version of the older PeANUt illustrative computer. The simulator is written completely in Java. The original version was written by Eric McCreath in 2011. The source code is available in the jar file under a GPL licence.

The specification is available: rPeANUtv2.1spec.pdf

The current jar: rPeANUt2.4c7f.jar (supports `#define`, `#include`, Java 7).
Older versions: rPeANUt2.4a.jar, rPeANUt2.4av7.jar (Java 7), rPeANUt2.4.jar (2014).

rPeANUt requires a Java Runtime Environment (JRE) or Java Development Kit (JDK) version 7 installed on your system (I use the Oracle JDK Runtime Environment, however, other Java implementation would generally also work). Once you have the JRE going rPeANUt should run in Windows, Mac, or Linux without too much trouble.

To run the simulator with the GUI simply download the jar and execute:

```
java –jar rPeANUt2.4c7.jar
```

To compile and run on the command-line an existing program `code.s` using the simulator, just execute:

```
java –jar rPeANUt2.4c7.jar code.s
```

rPeANUt has the following command line options:

- `-dump`: this does a dump of the frame buffer once the computer halts.
- `-count`: this produces a count of the instructions executed once the program halts.
- `-check`: this checks whether a script can compile or not
- `-screen`: this runs the program with only the display screen
- `-load`: this opens the rPeANUt editor with the specified file
- `-objdump`: this does a dump of the memory directly after it has assembled the specified program
- `-help`: provides this info

Short Macros

The `short macro' is a facility that will be introduced in rPeANUt in 2015. Its purpose is to improve code readability. The `short macro' has the form:

```
#define symbol number_or_register_name ; optional comment
```

It is a restricted form of the C `#define` directive. An example would be:

```
#define _FRAMEBUF 0x7C40    ; // rPeANUt address of start of frame buffer
#define WORDSZ 32           ; // size of a word in rPeANUt
#define xOffs -1            ; // stack offsets for func()
#define x R1                ; // registers holding vals of parameters & locals
#define i R0                ;
#define y R2                ;
```

```
#define z R3                ;
func:                       ; void func(int x) {
        load    SP #xOffs x ;
        load    #WORDSZ i   ;   int i = WORDSZ; // same as load #32 R0
        move    x y         ;   int y = x; // same as move R1 R2
        load    x #_FRAMEBUF z; int z = _FRAMEBUF[x];
        ...
```

Multi-line Macros

The general format for these is described in the rPeANUt manual. A simple example is

```
macro                       ; putchar(c);
putc &char
        load    #&char R1
        store   R1 0xfff0
mend
0x100:  putc 'h'            ; printf("%s", "Hello");
        putc 'e'
        putc 'l'
        putc 'l'
        putc 'o'
```

An example which calls `writeStr()` from `simpleIO.s` is as follows:

```
macro                         ; func(val);
call1   &func &val
        load    &val R0       ;   // note: overwrites R0
        push    R0
        call    &func
        pop     R0
mend

hello:  block   #"hello!"
....
        call1    writeStr hello ;  printf("%s", "hello!");
```