Prof. Dr. M. Ruckert

Praktikum IT Systeme I

September 2011

Problem Statement

(nach Donald E. Knuth, The Art of Computer Porgramming, Fascicle 1, MMIX, Section 1.3.2', Exercise 34. page 50)

Asssume that an MMIX computer has been wired up to the traffic signals at the corner of Del Mar Boulevard and Berkeley Avenue (Pasadena, CA).

http://maps.google.com/maps?ll=34.141973,241.890431&t=k&z=20

The MMIX operating system provides TRAPs to activate or deactivate the lights specifying the sum of four TETRA codes in $255 as follows:

Del Mar traffic light:	#00 off,	#01 green	#02 amber	#04 red
Del Mar pedestrian light:	#00 off,	#0100 WALK	#0200 DON'T WALK	
Berkeley traffic light:	#00 off,	#010000 green	#200000 amber	#400000 red

Berkeley pedestrian light:	#00 off,	#01000000 WALK	#02000000 DON'T WALK.	

The instruction `TRAP 0,#10,0` can be used to turn the lights on and the instruction `TRAP 0,#11,0` can be used to turn the lights off.

Cars or pedestrians wishing to travel on Berkeley across the boulevard must activate a sensor; if this condition never occurs, the light for Del Mar should remain green. The MMIX operating system provides TRAPs to read the sensor status and to wait until the sensor is activated. The instruction `TRAP 0,#14,0` returns the sensor status in register $255; the value is nonzero if and only if the sensor has been activated since the previous status read. The instruction `TRAP 0,#13,0` will read the sensor status as before, but in case the sensor was not activated, it will wait for activation before it returns.

Cycle times are as follows:

Del Mar traffic light is green ≥ 30 sec, amber 8 sec;
Berkeley traffic light is green 20 sec, amber 5 sec.

When a trafic light is green or amber for one direction, the other direction has a red light. When the traffic light is green, the corresponding WALK light is on, except that DON'T WALK flashes for 12 sec just before a green light turns to amber , as follows:

repeat 8 times:

DON'T WALK	0.5 sec
off	0.5 sec
DON'T WALK	4 sec (and remains on through amber and red cycles).

If the sensor is activated while the Berkeley light is green, the car or pedestrian will pass on that cycle. But if it is activated during the amber or red portions, another cycle will be necessary after the Del Mar traffic has passed.

To wait a specific amount of time, the MMIX operating system provides the instruction `TRAP 0,#12,0` which will wait until the number of milliseconds specified in register $255 has elapsed before returning.

Assignment 1: Bitmanipulation and TRAP

Write a complete MMIX program that controls these lights, following the stated protocol - except for the flashing of the pedestrian lights (keep them just green).

Assignment 2: Control structures

Implement the flashing of lights using a loop structure.

Assignment 3: Subroutines

Implement the flashing of lights using a subroutine with two parameters:
- the bit pattern of the light to flash (#0200 for Del Mar and #02000000 for Berkeley).
- the number of flashes (in this case 8)

Problem Statement (Continued)

Die DARE (Deutsche Ampel Regelungen und Experimente) Corporation vereinfacht das Programmieren ihrer Ampelschaltanlagen dadurch, dass sie die Programmierung mittels einer einfachen Steuersprache ermöglicht:

Befehle:

```
ON,nummer
OFF,nummer
WAIT,decisekunde
SENSORCLEAR,nummer
SENSORWAIT,nummer
REPEAT,nummer
END,0
CONTINUE,0
```

Jeder Befehl ist genau 2 Byte lang: ein Byte für den Opcode, ein Byte für das Argument.

- Die Befehle `ON` und `OFF` bekommen als Argument die Nummer des entsprechenden Bits in $255 für die TRAPs #10 und #11.
- Die Befehle `SENSORCLEAR` und `SENSORWAIT` bekommen als Argument die Sensornummer (hier immer 0).
- Der `REPEAT` Befehl bekommt als Argument die Anzahl der Wiederholungen und wiederholt die nachfolgenden Befehle bis zum nächsten `END` Befehl.
- Der `CONTINUE` Befehl beginnt dass Programm von vorne.

Das Program beginnt mit dem Label `PROGRAM`.

Beispiel:
Das folgende Programm blinkt das rote Fußgängerlicht vom Del Mar Boulevard 8 mal im Halbsekundentakt und lässt es dann 4 Sekunden an. Dannach wiederholt sich die Sequenz.

```
PROGRAM   BYTE REPEAT,8
          BYTE ON,DelMar+DontWalk
          BYTE WAIT,5
          BYTE OFF,DelMar+DontWalk
          BYTE WAIT,5
          BYTE END,0
          BYTE ON,DelMar+DontWalk
          BYTE WAIT,40
          BYTE CONTINUE,0
```

Damit das Programm so wie oben angegeben mit mmixal assembliert werden kann bedarf es Definitionen wie:

```
ON        IS   1
OFF       IS   2
...
```

```
DelMar    IS   0
green     IS   0
amber     IS   1
red       IS   2
Walk      IS   8
DontWalk  IS   9
Berkeley  IS   16
```

Assignment 4: Interpreter (ohne Kontrollstrukturen)

Schreiben sie einen Interpreter, der beliebige Steuerprogramme (noch ohne REPEAT-Schleifen) interpretiert. Testen Sie den Interpreter mit einem Steuerprogramm für die Kreuzung Del Mar Boulevard und Berkeley Avenue (noch ohne Blinken der Fußgängerampel).

Hinweis:
Ein Interpreter funktioniert wie ein einfacher Simulator für eine virtuelle Maschine, die das gegebene Programm abarbeitet. Man hat also folgenden Pseudo-Code:

```
pc = 0;
while(0==0);
{
  befehl = PROGRAM[pc];
  switch (opcode(befehl))
  { case ON:
      ...
      pc++;
      break;
    case OFF:
      ...

    case CONTINUE:
      pc=0;
      break;
    default: Fehler;
      ...
  }
}
```

Assignment 5: Interpreter (ohne geschachtelte Kontrollstrukturen)

Schreiben sie einen Interpreter, der beliebige Steuerprogramme mit REPEAT-Schleifen (ohne Schachtelung) interpretiert.

Hinweis:
Der Interpreter braucht fuer die REPEAT-Schleife zwei zusätzliche Variable. Einmal für den pc (program counter=adresse des Befehls) am Schleifenanfang und einen Zähler für die verbleibenden Widerholungen.

Assignment 6: Interpreter (optional)

Implementieren Sie einen Interpreter, der auch geschachtelte REPEAT Schleifen verarbeiten kann.

Hinweis:
Für beliebig geschachtelte Schleifen genügen zwei Variable nicht. Man braucht einen Stack.
Nun kann man entweder für die beiden zusätzlichen Variablen ein Array nehmen, das man mittels einer Indexvariable wie einen Stack verwendet, oder man schreibt den Interpreter als ein Unterprogram, das sich selbst rekursiv aufruft und nutzt damit den Hardware-Stack.