Software Implementation

Aleksandr Popov
0924595
a.popov@student.tue.nl

Devin van Broekhoven
0839004
d.v.broekhoven@student.tue.nl

Jiddo van Vliet
0821894
j.g.v.vliet@student.tue.nl

Yuntao Li
0910663
y.li.2@student.tue.nl

Dan Cristian Chirascu
0923784
d.c.chirascu@student.tue.nl

Group 17

April 14, 2015

Department of Mathematics and Computer Science

1 Introduction

This document describes the process of translating Java code into PP2 Assembly for the project DBL Embedded Systems. Large part of this document is comprised of the coding standard.

2 Data Representation

As our program is implemented as a state machine, where we constantly check some conditions and go to a different state if necessary, every function is required to be implemented as a subroutine. Moreover, it must not contain any loops that may potentially take an infinite amount of time (i.e. there must not be a loop with a guard depending on external signals). All the data that has to be transferred between executions of the function must be stored in memory in a space allocated by `DW` or `DS` commands in the `DATA` section of the program.

3 Common Guidelines

A description of guidelines for subroutine writing follows.

1. Subroutine may not wait in a loop for a certain value of PP2 `INPUT` or `ADCONVS` registers, it may only check them with an `if`.
2. Arguments to the function are passed on stack in reverse order. Use `SP+1`, `SP+2`, etc. to access them.
3. Return value, if specified, must be put into `R0` register. Subroutines may not rely on the value in this register remaining unchanged after they call another subroutine.
4. To check or change the current state subroutines must access variable located at `GB+STATE` in memory.
5. Subroutines must `PUSH` all the registers that are going to be used and `PULL` them again at the end of its execution, so that the only register that may change after subroutine execution is `R0`.
6. Instead of writing values for the motors directly to `OUTPUT`, subroutine `PWM` must be called with corresponding arguments to ensure that the voltage on motors is not too high.
7. All variables that need to be preserved between executions of a function must be stored in memory by means of declaring them in `DATA` section.
8. In order to find out which inputs and outputs should be used at any given moment author of code should refer to Machine Design document.

4 Coding Standard

This section contains a translation of some common Java commands into PP2 Assembly.

- ***Function call***
  Consider the following Java code:

```
1: <type> var = FUNCTION(arg0,arg1,...);
```

  It can be translated to PP2 Assembly as follows:

```
1: PUSH arg0
2: PUSH arg1
3: ...
4: BRS function
5: ADD SP n
6: STOR R0 [GB+VAR]
```

  Here `n` stands for the number of arguments.

- ***Function definition***
  Consider the following Java code:

```
1: <type> function(<type0> arg0, <type1> arg1, ...) {
2: ...
3: return var;
4: }
```

  It can be translated to PP2 Assembly as follows:

```
1: function:
2: PUSH R1
3: ...
4: PUSH Rn
5: LOAD R1 [SP+(n+1)]
6: ...
7: LOAD Rk [SP+(n+k+1)]
8: ...
9: LOAD R0 [GB+VAR]
10: PULL Rn
11: ...
12: PULL R1
13: RTS
```

  Here `n` stands for the number of used registers, `k` stands for the number of arguments to the function.

- ***while loop***
  Consider the following Java code:

```
1: while (var != 1000) {
2: ...
3: }
```

It can be translated to PP2 Assembly as follows:

```
1: LOAD R1 [GB+VAR]
2: loop:
3: CMP R1 1000
4: BEQ exit
5: ...
6: BRA loop
7: exit:
8: ...
```

Here `n` stands for the number of used registers, `k` stands for the number of arguments to the function.

Similar constructions can be used to implement different types of loops (e.g. `for` loop, `do-while` loop).

Other constructions are trivial and are used exactly as described in lectures for the subject 2IC30 Computer Systems.

5 Code explanation

PP2 Assembly code for the machine, which can be found in separate files attached to this project, is directly traceable back to Java code with respect to the guidelines given in the previous sections. Each subroutine represents some state, except for the main `switch` operator and some helper functions, i.e. `pwm`, `abort` and `display`.

For debugging purposes the program is fitted with variable `DEBUG` and some simple code that allows user to monitor current state or value from the black/white detector on the PP2 display.

Code is supplied with comments that explain in detail how it works and why certain decisions were made.