**Software Implementation and Integration**

28 March 2015

2IO70

Version 1

The purpose of this document is to show the data representation and coding standard we chose that is used to write the Assembly Language.

Group 16

Rolf Verschuuren

Wigger Boelens

Stefan van den Berg

Dat Phung

Maarten Keet

Tudor Petrescu

**Table of contents**

[Java to PHP 3](#_Toc415262171)

[Validation of PHP to Java 3](#_Toc415262172)

[Validation of Assembly to PHP compiler 3](#_Toc415262173)

[Appendix 1: Explanation of the compiler 5](#_Toc415262179)

[Appendix 2: Explanation of the compiler functions 7](#_Toc415262180)

# Java to PHP

The Java to PHP conversion is usually natural, the two languages sharing most syntax but there are some differences we must note down. We are not required to create a class in PHP. The initialization will differ in PHP from Java, but they share the same core in the end. Also while we have some of the variables initialized globally in Java, in PHP they will be local. Having no class will make the class initialization irrelevant in PHP and that’s why its missing. The later functions in the Java code right after the function TimerManage are included in the PHP code using “include “functions.php”;”. In TimerManage, % operation is replaced by the mod() function. Due to our PHP compiler limitations we are required to use variables as arguments when calling certain functions like for example storeData. The PHP code has been added as appendix 3.

# Validation of PHP to Java

Because of the natural similarity and ease of conversion, the PHP codes correctness can be correlated to its java counterpart, the correctness of the java code was validated in the Validation part of the Software Design.

# Validation of Assembly to PHP compiler

The compiler works in phases. We will go through these phases 1 by 1 to explain how the compiler does its job: compiling PHP-like code to assembly. Throughout the phases the compiler keeps track of the line number of the PHP code it is currently compiling and uses that, when an error occurs, to give information where the error is. The compiler is written in PHP5.6 and uses a command line interface.

## Preprocessing

In the first phase, the input code will be made ready for the next steps. A few things happen in this phase: First the file is read into the memory. The next step is that all comments, newlines and extra spaces are stripped from the file. The file is then split into single lines using the “;” symbol that denotes the end of a line. While doing this the compiler writes the data to two arrays: the data array for everything between “//\*\*DATA\*\*” and “//\*\*CODE\*\*” and the code array for everything after “//\*\*CODE\*\*”. Everything before //\*\*DATA\*\* is ignored. The data array gets compiled immediately.

The preprocessor further removes some special statements that are needed to make valid php such as “global” and changes some shortcuts in their full version. For example $abc++ will be changed into $abc+=1. This ensures that the compiler only needs to be able to handle $abc+=1.

## Splitting

In the second phase the code is split up by function. Every function gets his own array with all the lines that are in that function. The code not inside of a function goes into a separate array.

## Compiling

The third phase is the most important one. It starts by compiling the code that is at the start and not inside a function. While compiling it keeps track of what functions are called and adds these, if they are not already compiled, to the toCompile queue. This helps in making sure there is no dead code, as a function that is never called, will not be compiled. The compiler adds the function “main”, which is the default start point of the code, to the queue and starts processing it.

After compiling the main function it will continue in the next function in the toCompile queue and keep doing this till the toCompile queue is empty.

The compiling itself is not a lot more than a lot of regex and switch statements that look at the input and make an output from that. At the first notion of a variable a register is assigned to it. The code then uses this register in place of the variable. Some more difficult statements, like the function display which displays something, will BRS to premade assembly code that handles that. The compiler keeps track of which segments of the premade assembly code are used.

When the compiler meets an if statement, it saves the code inside it to a new function named “condtionali” where i is the amount of conditionals that have already been seen. It then places this function in the toCompile queue. It also saves the location of the end of the if statement, so it will later know where to return when the if function has ended.

## Combining

After there are no functions left in the toCompile queue, the combining phase starts. In this phase all the functions and the code outside the functions are combined into a single array. This phase also adds the used premade functions at the top and inserts the return statements at the correct position.

## Formatting

The last phase is the last interesting. It goes through the, now compiled code, and formats it. It uses either the length of the longest function name or the number 25 depending on which is larger to insert spaces in front of every line of code in a way everything lines up nicely.

The last step the compiler takes is writing the compiled code to a file and using the assembler provided to create the hex code.

# Appendix 1: Explanation of the compiler

The compiler works in phases. We will go through these phases 1 by 1 to explain how the compiler does its job: compiling PHP-like code to assembly. Throughout the phases the compiler keeps track of the line number of the PHP code it is currently compiling and uses that, when an error occurs, to give information where the error is. The compiler is written in PHP5.6 and uses a command line interface.

**Preprocessing**

In the first phase, the input code will be made ready for the next steps. A few things happen in this phase: First the file is read into the memory. The next step is that all comments, newlines and extra spaces are stripped from the file. The file is then split into single lines using the “;” symbol that denotes the end of a line. The code is divided in three segments. The first segment starts at //\*\*COMPILER, everything before this statement is ignored.

The preprocessor further removes some special statements that are needed to make valid php such as “global” and changes some shortcuts in their full version. For example $abc++ will be changed into $abc+=1. This ensures that the compiler only needs to be able to handle $abc+=1.

**Splitting**

In the second phase the code is split up by function. Every function gets his own array with all the lines that are in that function. The code not inside of a function goes into a separate array.

**Compiling**

The third phase is the most important one. It starts by compiling the code that is at the start and not inside a function. While compiling it keeps track of what functions are called and adds these, if they are not already compiled, to the toCompile queue. This helps in making sure there is no dead code, as a function that is never called, will not be compiled. The compiler adds the function “main”, which is the default start point of the code, to the queue and starts processing it.

After compiling the main function it will continue in the next function in the toCompile queue and keep doing this till the toCompile queue is empty.

The compiling itself is not a lot more than a lot of regex and switch statements that look at the input and make a output from that. At the first notion of a variable a register is assigned to it. The code then uses this register in place of the variable. Some more difficult statements, like the function display which displays something, will BRS to premade assembly code that handles that. The compiler keeps track of which segments of the premade assembly code are used.

When the compiler meets an if statement, it saves the code inside it to a new function named “condtionali” where i is the amount of conditionals that have already been seen. It then places this function in the toCompile queue. It also saves the location of the end of the if statement, so it will later know where to return when the if function has ended.

For every line it compiles, it takes the corresponding line of PHP and inserts it as a comment in the assembly. This is to help in debugging.

**Combining**

After there are no functions left in the toCompile queue, the combining phase starts. In this phase all the functions and the code outside the functions are combined into a single array. This phase also adds the used premade functions at the top and inserts the return statements at the correct position.

**Formatting**

The last phase is the least interesting. It goes through the, now compiled code, and formats it. It uses either the length of the longest function name or the number 25 depending on which is larger to insert spaces in front of every line of code in a way everything lines up nicely. It also makes sure the comments line up nicely.

The last step the compiler takes is writing the compiled code to a file and using the assembler provided to create the hex code.

# Appendix 2: Explanation of the compiler functions

storeRam($location, $value)

Store a value in the ram.

$location The location (a variable) to store the value in the ram

$value The value to store, needs to be a variable

return void

getRam($location)

Get a value from the ram.

$location The location (a variable) where the value is stored

return The value that is stored at the location

display($what, $onWhat, $location = '000001')

Display something on either the display or the leds.

Possible values for $onwhat:

* leds: the leds at the top
* leds2: the leds to the right
* display: the display

$what What to display, must be a variable

$onWhat On what to display

$location Where to show the value when using the display, defaults to the right position

return void

pow($number,$power)

Get the power of a number

$number The number to power

$power The power value

return Int; The result

mod($what, $variable)

Take the modulo of a number

$what Modulo what

$variable Variable to modulo over

return void

getInput($writeTo, $type)

Get button or analog input. When you just want the input of 1 button, use getButtonPressed instead.

$writeTo Variable to write the input to

$type Type of input, possible values are: buttons, analog

return void

getButtonPressed($button)

Check if a button is pressed. Puts the result into R5.

$button Which button to check (input a variable)

return Int; Whether or not the button is pressed.

installCountdown($functionName)

Install the countdown.

$functionName The name of the function where the timer should go to

return void

startCountdown()

Start the countdown.

Retrun void

pushStack($variable)

Push a variable to the stack

$variable The variable to push to the stack

return void

pullStack($variable)

Pull a variable from the stack.

$variable The variable where the pulled variable is put into

return void

setCountdown($countdown)

Set the timer interrupt to a value. It will first reset the timer to 0.

$countdown How long the countdown should wait, in timer ticks

return void

getData($location, $offset)

Get data. Use offset 0 when it is just a single value.

$location The location where the variable is stored

$offset The offset of the location

return The value of the data segment

storeData($variable, $location, $offset)

Store data. Use offset 0 when it is just a single value.

$variable The variable to store

$location The name of the location where the variable is stored

$offset The offset of the location

return void

sleep($howLong)

Pause the program.

$howLong How long to sleep in clockticks

return void

initVar($variable,$places)

Initialize a variable that is used in that data segment.

$variable The name of the variable

$places How long the array is

return void

branch($branchTO)

Branch to a function.

$branchTO where to branch to

return void

moveFunction($branchTO)

Move a function in the assembly code.

$branchTO Where to branch to

return void

**Appendix 3: PHP Program**

1 <?php

2 */\* vim: set expandtab tabstop=4 shiftwidth=4 softtabstop=4: \*/*

3

4 */\*\**

5  *\* Sort of a simulation of the PP2 program controlling the Fischer Technik in order to sort black and white discs.*

6  *\* @team Group 16*

7  *\* @author Stefan van den Berg*

8  *\* @author Rolf Verschuuren*

9  *\* @author Wigger Boelens*

10  *\* @since 13/3/2015*

11  *\*/*

12 **include** 'functions.php';

13 *//\*\*COMPILER\*\**

14 moveFunction('timerInterrupt', 1);

15 moveFunction('timerManage', 50);

16

17 *//\*\*DATA\*\**

18 initVar('offset', 1);

19 initVar('stackPointer', 1);

20 initVar('outputs', 12);

21 initVar('state', 1);

22

23 *//\*\*CODE\*\**

24 define('TIMEMOTORDOWN', 150); *//how long the sorter takes to move down*

25 define('BELT', 2000);

26 define('BELTROUND', 2000);*//Time for the belt to make a rotation*

27 define('SORT', 200);*//Clockticks to make a rotation*

28 define('COUNTDOWN', 30000);

29 *//outputs*

30 define('LENSLAMPPOSITION', 2);

31 define('LENSLAMPSORTER', 6);

32 define('HBRIDGE0', 0);

33 define('HBRIDGE1', 1);

34 define('CONVEYORBELT', 7);

35 define('FEEDERENGINE', 3);

36 define('DISPLAY', 8);

37 define('LEDSTATEINDICATOR', 9);

38

39 *//not a state*

40 **function** main()

41 {

42 **global** $counter, $location;

43

44 *//store the offset of the program, this is used in the interrupt*

45 storeData(R5, 'offset', 0);

46 *//install the countdown*

47 installCountdown('timerInterrupt');

48

49 *//save the location of the stackPointer, so we can clear the stack*

50 storeData(SP, 'stackPointer', 0);

51

52 *//the variables that are the same throughout the program:*

53 $counter = 0;

54 $location = 0;

55 $sleep = 0;

56

57

58 *//stop everything*

59 $temp = 0;

60 storeData($temp, 'outputs', HBRIDGE1);

61 storeData($temp, 'outputs', LENSLAMPPOSITION);

62 storeData($temp, 'outputs', LENSLAMPSORTER);

63 storeData($temp, 'outputs', LEDSTATEINDICATOR);

64 storeData($temp, 'outputs', DISPLAY);

65 storeData($temp, 'outputs', CONVEYORBELT);

66 storeData($temp, 'outputs', FEEDERENGINE);

67

68 *//sh0w the state*

69 $state = 0;

70 storeData($state, 'state', 0);

71

72 *//set HBridge so the sorter starts moving up*

73 $temp = 10;

74 storeData($temp, 'outputs', HBRIDGE0);

75 unset($temp, $state);

76

77 *//go to the first state*

78 initial();

79 }

80

81 *//state 0*

82 **function** initial()

83 {

84 **global** $sleep;

85 *//disable the lights on the right hand side*

86 $temp = 0;

87 display($temp, 'leds2');

88

89 $temp = getData('stackPointer', 0);

90 setStackPointer($temp);

91

92 timerManage();

93

94 *//check if the sorter push button is pressed*

95 $push = getButtonPressed(5);

96 **if** ($push == 1) {

97 *//move sorter down*

98 $temp = 0;

99 storeData($temp, 'outputs', HBRIDGE0);

100 $temp = 10;

101 storeData($temp, 'outputs', HBRIDGE1);

102

103 *//update state*

104 $temp = 1;

105 storeData($temp, 'state', 0);

106 unset($temp);

107

108 *//reset sleep for the next function*

109 $sleep = 0;

110 calibrateSorter();

111

112 }

113 unset($push);

114

115 *//loop*

116 initial();

117 }

118

119 *//state 1*

120 **function** calibrateSorter()

121 {

122 **global** $sleep;

123 timerManage();

124

125 *//the sorter is now moving down,*

126 *//we're waiting for it to reach its bottom position*

127 **if** ($sleep == TIMEMOTORDOWN) {

128 *//stop the sorter*

129 $temp = 0;

130 storeData($temp, 'outputs', HBRIDGE1);

131

132 *//update the state*

133 $state = 2;

134 storeData($state, 'state', 0);

135 unset($state);

136

137 *//reset sleep for the next state*

138 $sleep = 0;

139 resting();

140 }

141

142 *//loop*

143 $sleep++;

144 calibrateSorter();

145 }

146

147 *//state 2*

148 **function** resting()

149 {

150 timerManage();

151

152 *//the program is now waiting for the user to press start/stop*

153 $startStop = getButtonPressed(0);

154 **if** ($startStop == 1) {

155 *//sleep so we don't go to pause immediately*

156

157

158 *//power up the lamps*

159 $temp = 12;

160 storeData($temp, 'outputs', LENSLAMPPOSITION);

161 unset($temp);

162 timerManage();

163 sleep(1000);

164 $temp = 12;

165 storeData($temp, 'outputs', LENSLAMPSORTER);

166 unset($temp);

167 timerManage();

168 sleep(2000);

169

170

171 *//start up the belt and feeder*

172 $temp = 9;

173 storeData($temp, 'outputs', CONVEYORBELT);

174 $temp = 9;

175 storeData($temp, 'outputs', FEEDERENGINE);

176 unset($temp);

177

178 *//set and start the countdown for the moment there are no more disks*

179 *//this countdown will reset every time a disk is found*

180 *//when it triggers, timerInterrupt will be ran.*

181 setCountdown(COUNTDOWN);

182 startCountdown();

183

184 *//update the state*

185 $state = 3;

186 storeData($state, 'state', 0);

187 unset($state);

188

189 running();

190 }

191 unset($startStop);

192

193 *//loop*

194 resting();

195 }

196

197 *//state 3*

198 **function** running()

199 {

200 timerManage();

201

202 *//check if we need to pause*

203 $startStop = getButtonPressed(0);

204 **if** ($startStop == 1) {

205 *//stop the feeder engine*

206 $temp = 0;

207 storeData($temp, 'outputs', FEEDERENGINE);

208 unset($temp);

209

210 *//exit after 1 rotation of the belt*

211 setCountdown(BELT \* 10);

212

213 *//update the state*

214 $state = 9;*//TODO: echte state*

215 storeData($state, 'state', 0);

216 unset($state);

217

218 runningTimer();

219

220 }

221 unset($startStop);

222

223 *//check if a disk is at the position detector*

224 $position = getButtonPressed(7);

225 **if** ($position == 1) {

226 *//reset the countdown, because a disk was just detected*

227 setCountdown(COUNTDOWN);

228

229 *//update the state*

230 $state = 4;

231 storeData($state, 'state', 0);

232 unset($state);

233 runningWait();

234 }

235 unset($position);

236

237 *//loop*

238 running();

239 }

240

241 *//state 4*

242 **function** runningWait()

243 {

244 timerManage();

245

246 *//check if we need to pause*

247 $startStop = getButtonPressed(0);

248 **if** ($startStop == 1) {

249 *//stop the feeder engine*

250 $temp = 0;

251 storeData($temp, 'outputs', FEEDERENGINE);

252 unset($temp);

253

254 *//exit after 1 rotation of the belt*

255 setCountdown(BELT \* 10);

256

257 *//update the state*

258 $state = 9;

259 storeData($state, 'state', 0);

260 unset($state);

261

262 runningTimer();

263

264 }

265 unset($startStop);

266

267 *//check if a disk is at the position detector*

268 $position = getButtonPressed(7);

269 **if** ($position == 0) {

270 *//reset the countdown, because a disk was just detected*

271 setCountdown(COUNTDOWN);

272

273 *//update state*

274 $state = 5;

275 storeData($state, 'state', 0);

276 unset($state);

277

278 runningTimerReset();

279

280 }

281 unset($position);

282

283 *//check if a white disk is at the colour detector*

284 $colour = getButtonPressed(6);

285 **if** ($colour == 1) {

286 *//move the sorter up so the disk goes to the correct box*

287 $temp = 10;

288 storeData($temp, 'outputs', HBRIDGE0);

289

290 *//stop the feeder engine*

291 $temp = 0;

292 storeData($temp, 'outputs', FEEDERENGINE);

293 unset($temp);

294

295 *//update state*

296 $state = 6;

297 storeData($state, 'state', 0);

298 unset($state);

299

300 motorUp();

301 }

302 unset($colour);

303

304 *//loop*

305 runningWait();

306 }

307

308 *//state 5*

309 **function** runningTimerReset()

310 {

311 timerManage();

312

313 *//update state*

314 $state = 4;

315 storeData($state, 'state', 0);

316 unset($state);

317

318 runningWait();

319 }

320

321 *//state 6*

322 **function** motorUp()

323 {

324 **global** $sleep;

325 timerManage();

326

327 *//check if we need to pause*

328 $startStop = getButtonPressed(0);

329 **if** ($startStop == 1) {

330 *//stop the feeder engine*

331 $temp = 0;

332 storeData($temp, 'outputs', FEEDERENGINE);

333 unset($temp);

334

335 *//exit after 1 rotation of the belt*

336 setCountdown(BELT \* 10);

337

338 *//update the state*

339 $state = 10;

340 storeData($state, 'state', 0);

341 unset($state);

342

343 motorUpTimer();

344

345 }

346 unset($startStop);

347

348 *//check if the sorter push button is pressed*

349 $push = getButtonPressed(5);

350 **if** ($push == 1) {

351 *//stop the sorter engine, because its at its highest position*

352 $temp = 0;

353 storeData($temp, 'outputs', HBRIDGE0);

354 unset($temp);

355

356 *//update state*

357 $state = 7;

358 storeData($state, 'state', 0);

359 unset($state);

360

361 *//set sleep for the next function*

362 $sleep = 0;

363

364 whiteWait();

365 }

366 unset($push);

367

368 *//loop*

369 motorUp();

370 }

371

372 *//state 7*

373 **function** whiteWait()

374 {

375 **global** $sleep;

376 timerManage();

377

378 *//we are waiting for the white disk to be sorted*

379 **if** ($sleep == SORT) {

380 *//start moving the sorter down*

381 $temp = 10;

382 storeData($temp, 'outputs', HBRIDGE1);

383 unset($temp);

384

385 *//make sure the timerinterrupt is correct*

386 setCountdown(COUNTDOWN);

387

388 *//update state*

389 $state = 8;

390 storeData($state, 'state', 0);

391 unset($state);

392

393 *//reset sleep for the next function*

394 $sleep = 0;

395 motorDown();

396

397 }

398

399 *//check if we need to pause*

400 $startStop = getButtonPressed(0);

401 **if** ($startStop == 1) {

402 *//stop the feeder engine*

403 $temp = 0;

404 storeData($temp, 'outputs', FEEDERENGINE);

405 unset($temp);

406

407 *//exit after 1 rotation of the belt*

408 setCountdown(BELT \* 10);

409

410 *//update the state*

411 $state = 11;

412 storeData($state, 'state', 0);

413 unset($state);

414

415 whiteWaitTimer();

416 }

417 unset($startStop);

418

419 *//loop*

420 $sleep++;

421 whiteWait();

422 }

423

424 *//state 8*

425 **function** motorDown()

426 {

427 **global** $sleep;

428 timerManage();

429

430

431 *//check if a white disk is at the colour detector*

432 $colour = getButtonPressed(6);

433 **if** ($colour == 1) {

434 *//move the sorter up so the disk goes to the correct box*

435 $temp=0;

436 storeData($temp,'outputs',HBRIDGE1);

437 $temp = 10;

438 storeData($temp, 'outputs', HBRIDGE0);

439 unset($temp);

440

441 *//update state*

442 $state = 6;

443 storeData($state, 'state', 0);

444 $sleep=0;

445 unset($state);

446

447 motorUp();

448 }

449 unset($colour);

450

451

452 *//the sorter is moving down, we are waiting for that to complete*

453 **if** ($sleep == TIMEMOTORDOWN) {

454 *//stop the sorter, its where it should be*

455 $temp = 0;

456 storeData($temp, 'outputs', HBRIDGE1);

457 $temp = 7;

458 storeData($temp, 'outputs', FEEDERENGINE);

459 unset($temp);

460

461 *//update state*

462 $state = 4;

463 storeData($state, 'state', 0);

464 *//reset sleep for the next function*

465 $sleep = 0;

466 unset($state);

467

468 runningWait();

469 }

470

471 *//check if we need to pause*

472 $startStop = getButtonPressed(0);

473 **if** ($startStop == 1) {

474 *//stop the feeder engine*

475 $temp = 0;

476 storeData($temp, 'outputs', FEEDERENGINE);

477 unset($temp);

478

479 *//exit after 1 rotation of the belt*

480 setCountdown(BELT \* 10);

481

482 *//update the state*

483 $state = 12;

484 storeData($state, 'state', 0);

485 unset($state);

486

487 motorDownTimer();

488 }

489 unset($startStop);

490

491 *//loop*

492 $sleep++;

493 motorDown();

494

495 }

496

497 *//state 9*

498 **function** runningTimer()

499 {

500 timerManage();

501

502 *//update state*

503 $state = 13;

504 storeData($state, 'state', 0);

505 unset($state);

506

507 runningStop();

508 }

509

510 *//state 10*

511 **function** motorUpTimer()

512 {

513 timerManage();

514

515 *//update state*

516 $state = 14;

517 storeData($state, 'state', 0);

518 unset($state);

519

520 motorUpStop();

521 }

522

523 *//state 11*

524 **function** whiteWaitTimer()

525 {

526 timerManage();

527

528 *//update state*

529 $state = 15;

530 storeData($state, 'state', 0);

531 unset($state);

532

533 whiteWaitStop();

534 }

535

536 *//state 12*

537 **function** motorDownTimer()

538 {

539 timerManage();

540

541 *//update state*

542 $state = 16;

543 storeData($state, 'state', 0);

544 unset($state);

545

546 motorDownStop();

547 }

548

549 *//state 13*

550 **function** runningStop()

551 {

552 timerManage();

553

554 *//check if a white disk is at the colour detector*

555 $colour = getButtonPressed(6);

556 **if** ($colour == 1) {

557 *//stop the sorter engine, because its at its highest position*

558 $temp = 10;

559 storeData($temp, 'outputs', HBRIDGE0);

560

561 *//stop the feeder engine*

562 $temp = 0;

563 storeData($temp, 'outputs', FEEDERENGINE);

564 unset($temp);

565

566 *//update state*

567 $state = 10;

568 storeData($state, 'state', 0);

569 unset($state);

570

571 motorUpStop();

572 }

573 unset($colour);

574

575 *//loop*

576 runningStop();

577 }

578

579 *//state 14*

580 **function** motorUpStop()

581 {

582 timerManage();

583

584 *//check if the sorter push button is pressed*

585 $push = getButtonPressed(5);

586 **if** ($push == 1) {

587 *//stop the engine of the sorter*

588 $temp = 0;

589 storeData($temp, 'outputs', HBRIDGE0);

590 unset($temp);

591

592 *//update state*

593 $state = 11;

594 storeData($state, 'state', 0);

595 unset($state);

596

597 whiteWaitStop();

598 }

599 unset($push);

600

601 *//loop*

602 motorUpStop();

603 }

604

605 *//state 15*

606 **function** whiteWaitStop()

607 {

608 **global** $sleep;

609 timerManage();

610

611 *//check if the white disk has been sorted*

612 **if** ($sleep == SORT) {

613 *//it has, so lets start moving the sorter down*

614 $temp = 10;

615 storeData($temp, 'outputs', HBRIDGE1);

616 $temp = 0;

617 storeData($temp, 'outputs', FEEDERENGINE);

618 unset($temp);

619

620 *//update state*

621 $state = 12;

622 storeData($state, 'state', 0);

623 unset($state);

624

625 $sleep = 0;

626 motorDownStop();

627 }

628

629 *//loop*

630 $sleep++;

631 whiteWaitStop();

632 }

633

634 *//state 16*

635 **function** motorDownStop()

636 {

637 **global** $sleep;

638 timerManage();

639

640 *//check if the sorter has moved down*

641 **if** ($sleep == TIMEMOTORDOWN) {

642 *//it has, so lets stop it*

643 $temp = 0;

644 storeData($temp, 'outputs', HBRIDGE1);

645 unset($temp);

646

647 *//update the state*

648 $state = 9;

649 storeData($state, 'state', 0);

650 unset($state);

651

652 $sleep = 0;

653 runningStop();

654 }

655

656 *//loop*

657 $sleep++;

658 motorDownStop();

659 }

660

661 *//not a state*

662 **function** timerInterrupt()

663 {

664 timerManage();

665 *//show that we are in the timer interrupt*

666 $temp = 5;

667 display($temp, 'display');

668

669 *//start moving the sorter up, to start the calibration*

670 $temp = 10;

671 storeData($temp, 'outputs', HBRIDGE0);

672

673 *//stop the rest*

674 $temp = 0;

675 storeData($temp, 'outputs', LENSLAMPPOSITION);

676 storeData($temp, 'outputs', LENSLAMPSORTER);

677 storeData($temp, 'outputs', LEDSTATEINDICATOR);

678 storeData($temp, 'outputs', DISPLAY);

679 storeData($temp, 'outputs', CONVEYORBELT);

680 storeData($temp, 'outputs', FEEDERENGINE);

681

682

683 *//reset, because we will no longer be in timerInterrupt*

684 display($temp, 'display');

685 unset($temp);

686

687 *//go back to initial*

688 $temp = getData('offset', 0);

689 $temp2 = getFuncLocation('initial');

690 $temp += $temp2;

691

692

693 addStackPointer(2);

694 pushStack($temp);

695 addStackPointer(-1);

696 }

697

698 *//not a state*

699 **function** abort()

700 {

701 *//free some memory*

702 unset($engines);

703

704 *//prevent timerinterrupt*

705 setCountdown(1000);

706 $temp = getData('stackPointer', 0);

707 setStackPointer($temp);

708

709 *//stop everything*

710 $temp = 0;

711 storeData($temp, 'outputs', HBRIDGE1);

712 storeData($temp, 'outputs', HBRIDGE0);

713 storeData($temp, 'outputs', LENSLAMPPOSITION);

714 storeData($temp, 'outputs', LENSLAMPSORTER);

715 storeData($temp, 'outputs', LEDSTATEINDICATOR);

716 storeData($temp, 'outputs', DISPLAY);

717 storeData($temp, 'outputs', CONVEYORBELT);

718 storeData($temp, 'outputs', FEEDERENGINE);

719 unset($temp);

720

721 *//apply the changes to actually stop it*

722 timerManage();

723

724 *//update the state*

725 $state = 17;

726 storeData($state, 'state', 0);

727

728

729 *//show we aborted*

730 $state = 7;

731 display($state, 'leds2', 0);

732 unset($state);

733

734 aborted();

735 }

736

737 *//state 17*

738 **function** aborted()

739 {

740 *//prevent timer interrupt*

741 setCountdown(1000);

742 timerManage();

743 *//check if we can start again*

744 $startStop = getButtonPressed(0);

745 **if** ($startStop == 1) {

746 *//start moving the sorter up, to start the calibration*

747 $temp = 10;

748 storeData($temp, 'outputs', HBRIDGE0);

749 unset($temp);

750

751 *//update the state*

752 $state = 0;

753 storeData($state, 'state', 0);

754 unset($state);

755

756 initial();

757 }

758 unset($startStop);

759 aborted();

760

761 }

762

763 *//not a state*

764 **function** timerManage()

765 {

766 **global** $location, $counter, $engine, $sleep;

767

768 **if** ($location == 0) {

769 $engines = 0;

770 }

771

772 *//makes sure that when $counter >12 it will reset to 0*

773 mod(12, $counter);

774

775 *//get the voltage of output $location*

776 $voltage = getData('outputs', $location);

777

778 *//power up the output when it needs to*

779 **if** ($voltage > $counter) {

780 $voltage = $location;

781 $voltage = pow(2, $voltage);

782 $engines += $voltage;

783 }

784

785 *//check if we did all outputs*

786 **if** ($location == 7) {

787 *//actually output the result*

788 sleep(1);

789 display($engines, 'leds');

790

791

792 unset($voltage);

793 *//check if abort is pressed*

794 $abort = getButtonPressed(1);

795 **if** ($abort == 1) {

796 abort();*//STOP THE MACHINE!*

797 }

798 unset($abort);

799

800 *//check if we are in a new iteration*

801 **if** ($counter == 6) {

802 *//set the first part of the display*

803 $temp = getData('state', 0);

804 mod(10, $temp);

805 display($temp, 'display', 1);

806 unset($temp);

807 }

808 *//check if we are at the end of the iteration*

809 **if** ($counter == 11) {

810 *//set the second part of the display;*

811 pushStack($sleep);

812

813 $temp = getData('state', 0);

814 *//get the last digit of the state*

815 *//we have no variables left, so we use $sleep*

816

817 $sleep = $temp;

818 mod(10, $sleep);

819 $temp -= $sleep;

820 $temp /= 10;

821 *//display the last digit*

822 display($temp, 'display', 2);

823

824 pullStack($sleep);

825 unset($temp);

826 }

827

828

829 *//set the variables for the next run*

830 $engines = 0;

831 $location = 0;

832 $counter++;

833

834 *//and return to where we came from*

835 **return**;

836 }

837

838 *//loop*

839 $location++;

840 branch('timerManage');

841 }490 *// calibration*

491 storeData(1, "outputs", HBRIDGE0);

492 *//update the state*

493 $state = 0;

494 initial();

495 }

496 *//loop*

497 aborted();

498

499 }

500

501 void timerManage() {

502

503

504 *//make sure that when counter can not*

505 *// be higher than 12*

506 mod(13, $counter);

507 *//get the voltage of output $location*

508 int $voltage = getData("outputs",

509 $location);

510 *//power up the output when it needs to*

511 **if** ($voltage > $counter) {

512 $engines += pow(2, $voltage);

513 }

514 *//check if we are in a new itteration*

515 **if** ($counter == 0) {

516 *//set the first part of the display*

517 $temp = getData("state", 0);

518 mod(10, $temp);

519 display($temp, "display", "1");

520

521

522 }

523 *//check if we are at the end of the*

524 *// itteration*

525 **if** ($counter == 12) {

526 *//set the second part of the display;*

527 $temp = getData("state", 0);

528 $temp = $temp / 10;

529 mod(10, $temp);

530 display($temp, "display", "01");

531

532 }

533 *//check if we did all outputs*

534 **if** ($location > 7) {

535 display($engines, "leds", "");

536 *//set the variables for the next run*

537 $engines = 0;

538 $location = 0;

539 $counter++;

540

541 *//check if abort is pressed*

542 $abort = getButtonPressed(1);

543 **if** ($abort == 1) {

544 abort();*//stop the machine*

545 }

546 **return**;

547 }

548

549

550 $location++;

551 timerManage();

552 }

553 }