Tomasulo Algorithm Project

ECE6100

Bihai Zhang

GT ID: 903547409

1. Summary of Tomasulo Algorithm

Tomasulo algorithm is a hardware algorithm for dynamic scheduling of instructions. As a result, the hardware can achieve out-of-order execution of instructions. So, the execution units are more efficiently utilized instead of wasting time in idle status. The core innovation of this algorithm is register renaming in hardware. Robert Tomasulo invented reservation stations to support register renaming in hardware. And a common data bus (CDB) was introduced to let each reservation station receive operands.

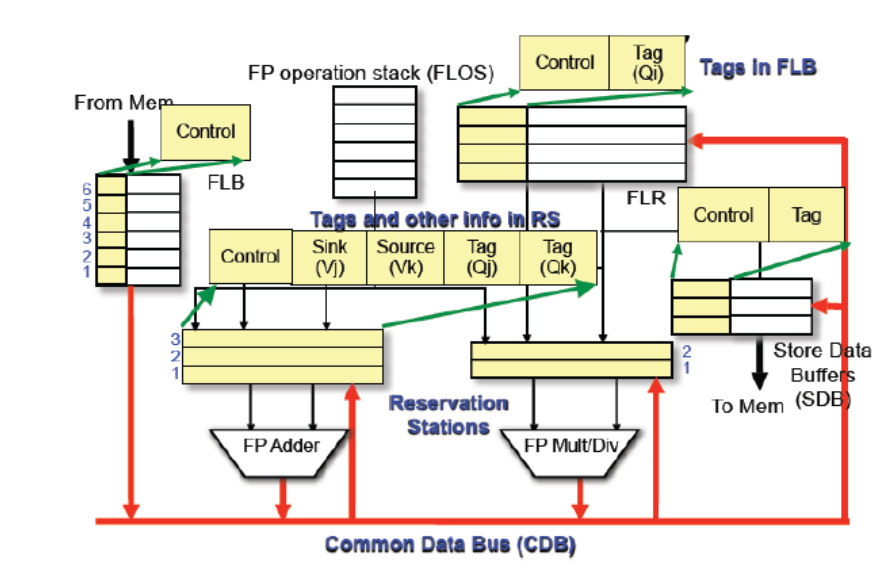


Figure 1 Tomasulo Algorithm

As is shown in Figure 1, instructions are loaded to corresponding reservations stations in a FIFO order. The reservation station store the operation to perform, the value of operands/where the operands will be generated. Firstly, a RS issue an instruction. Then, when all operands are available, execution of this instruction will begin in the cycle following the cycle when all its operands are known, assuming that computing resources are available. Finally, when the execution ends, if the CDB is available, the computing unit will broadcast the result to all RS via the CDB. Each RS will check if the value is needed.

The program uses a main loop to simulate the cycles of the processing unit. In each cycle, the program will make decisions according to the rules set by the Tomasulo algorithm.

1. Resource Code

The resource code is appended in the Appendix.

1. Example Runs

In the following subsections, RS Add1, Add2, Add3, Mult1, Mult2, Load1, Load2, Load3, Stor1, Stor2, Stor3 are numbered as 1,2,3,4,5,6,7,8,9,10,11 in the register result status. The default value stored in R0, R2, R4, R6, R8 and R10 are 1, 2, 3 ,4, 5 and 6. Cycles required for execution are assumed to be the same as the long example.

* 1. WAR

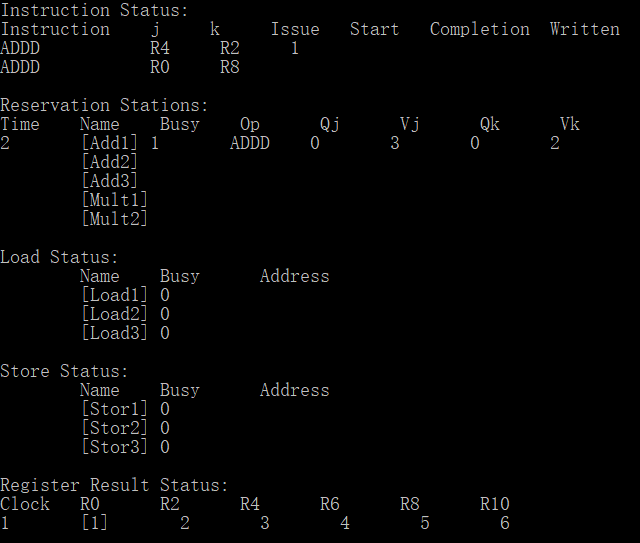
Instructions:

ADDD R0 R4 R2

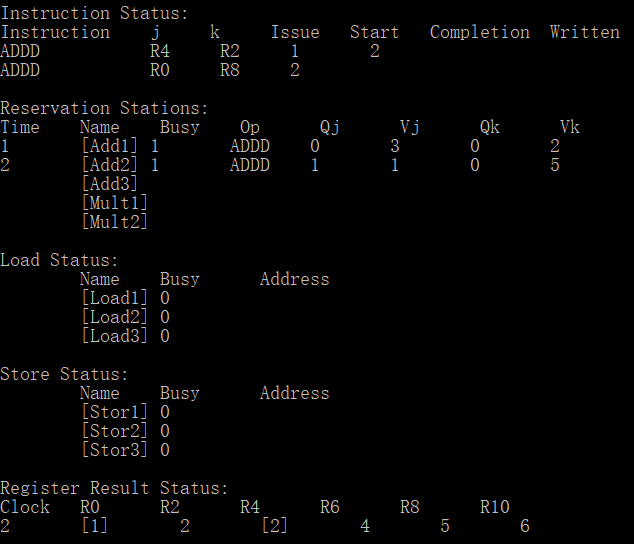
ADDD R4 R0 R8

The second instruction must write to R4 after the first instruction read the value in R4. So, it is WAR.

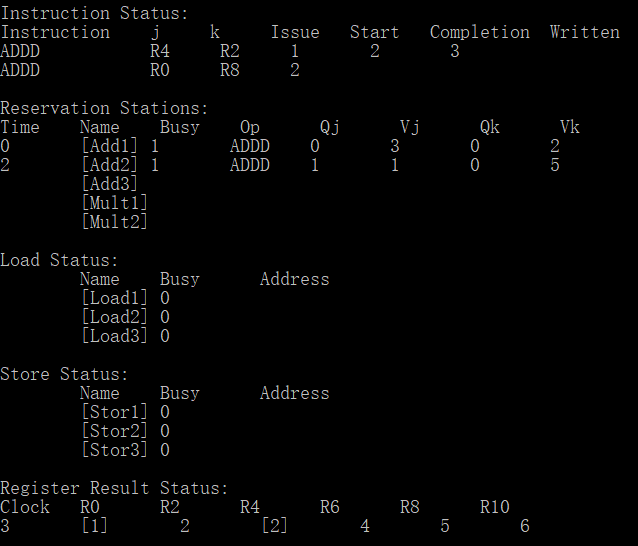
Cycle 1



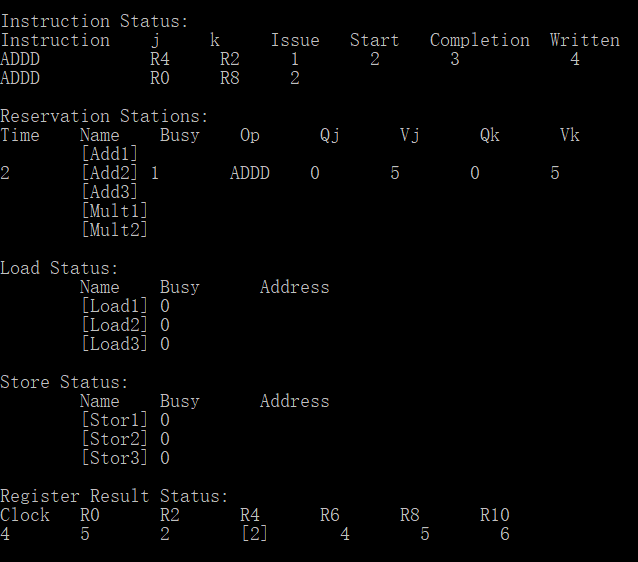
Cycle 2



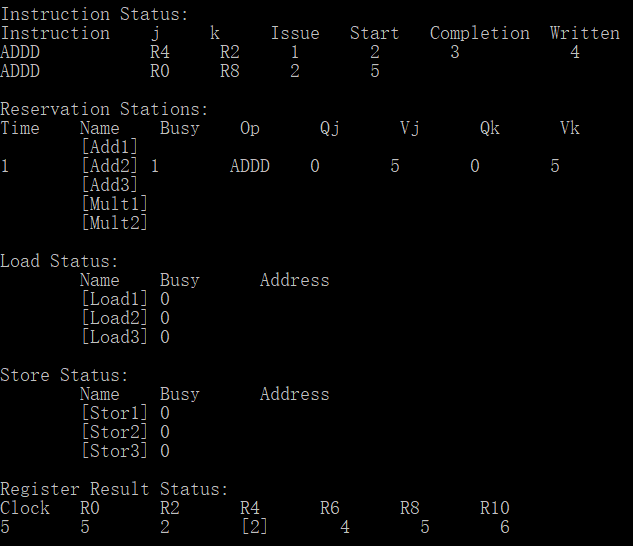
Cycle 3



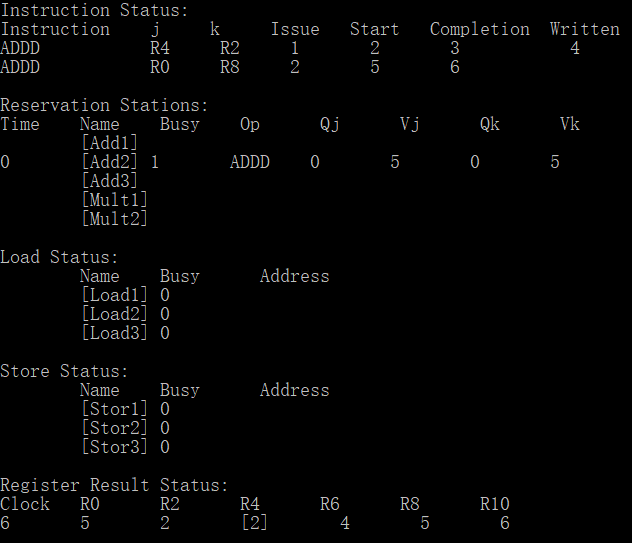
Cycle 4



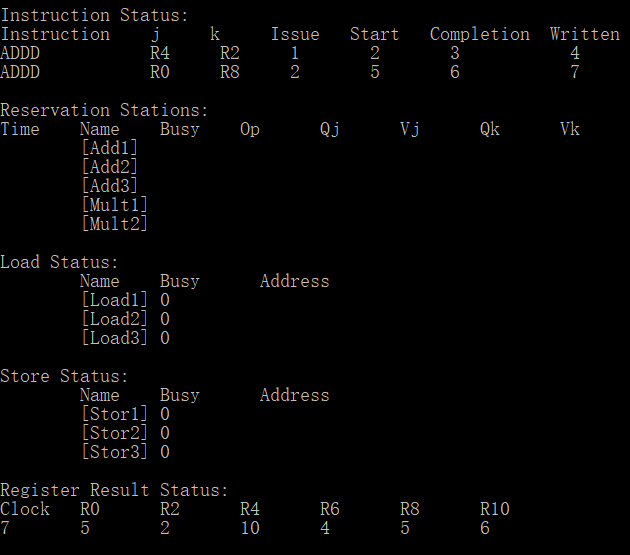
Cycle 5



Cycle 6



Cycle 7



* 1. RAW

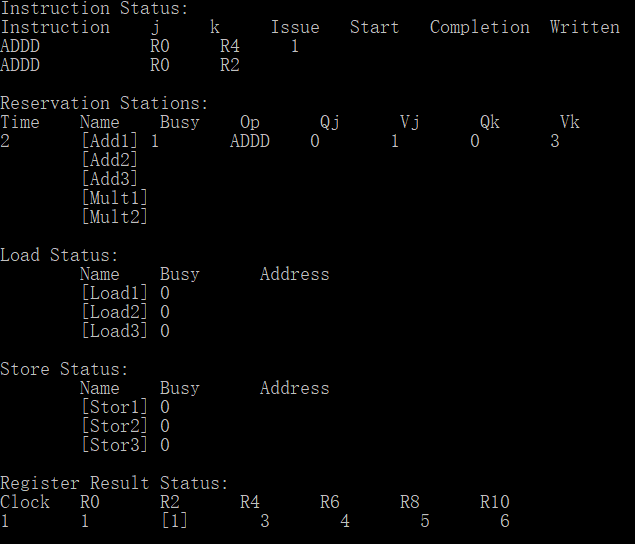
Instructions:

ADDD R2 R0 R4

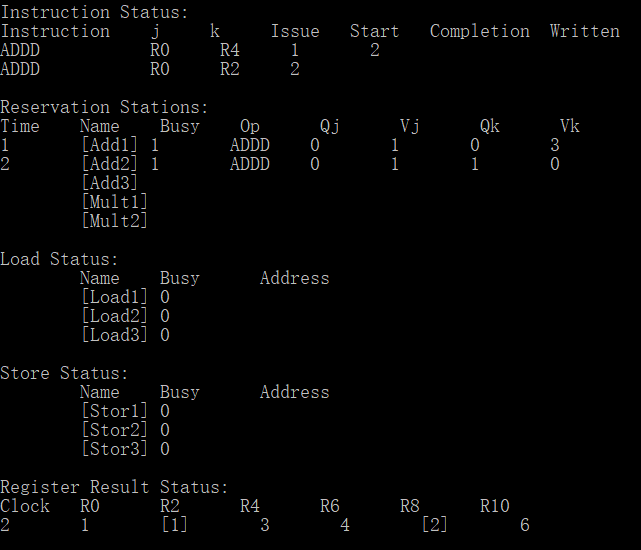
ADDD R8 R0 R2

The second instruction must read the value in R2 after the first instruction writes to R2. So, it is RAW.

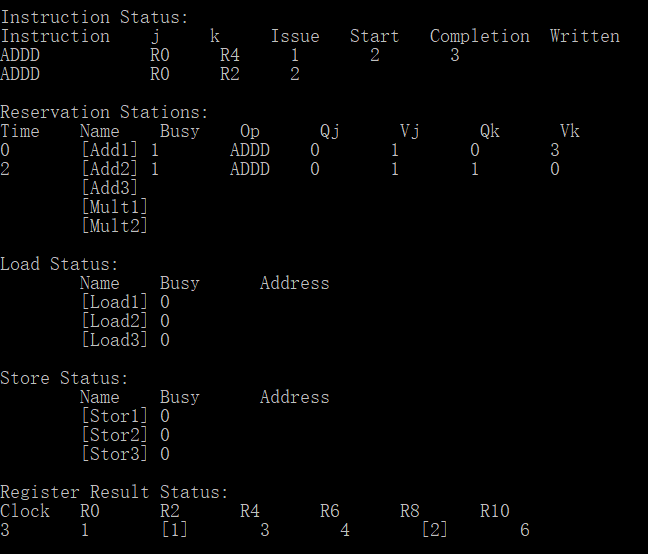
Cycle 1



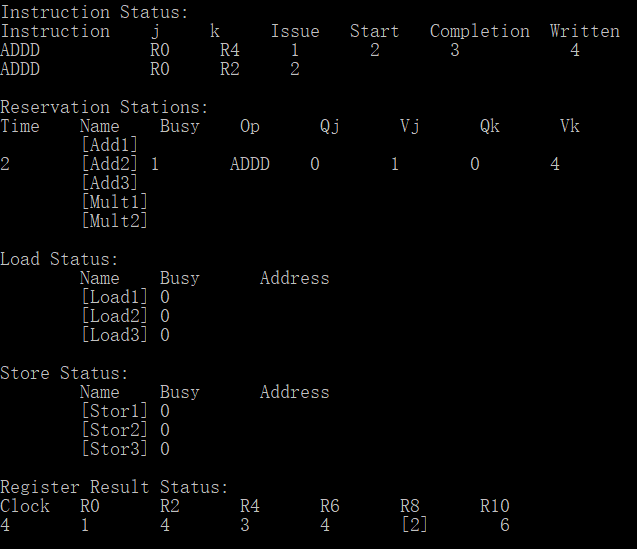
Cycle 2



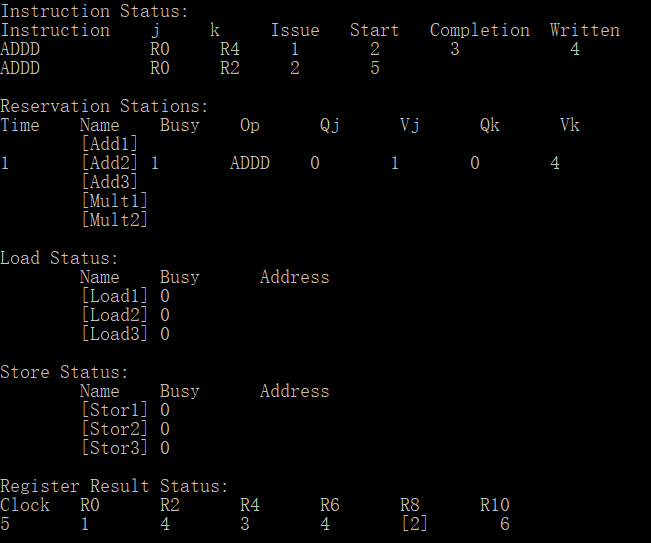
Cycle 3



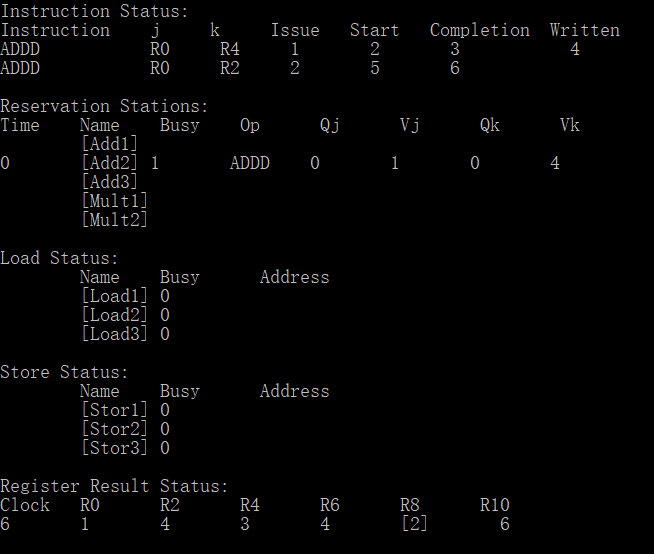
Cycle 4



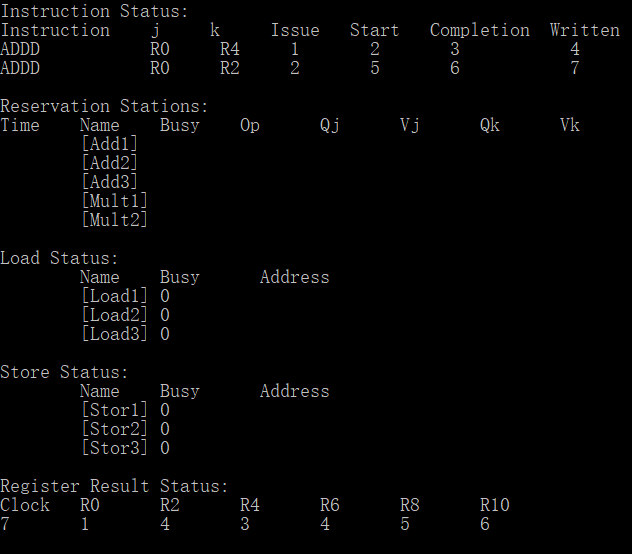
Cycle 5



Cycle 6



Cycle 7



* 1. WAW

Instructions:

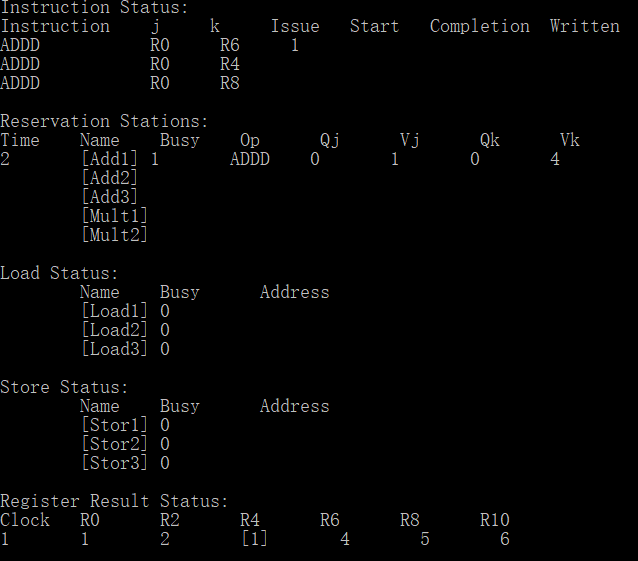
ADDD R4 R0 R6

ADDD R2 R0 R4

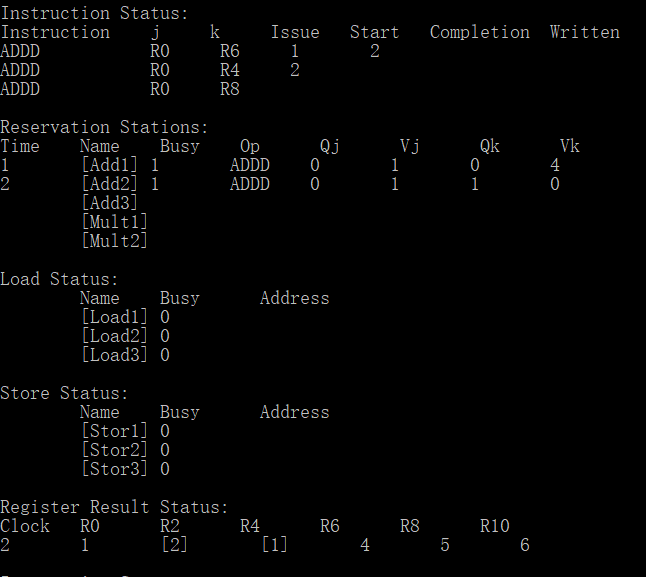
ADDD R4 R0 R8

There is WAW hazard via R4 between the first instruction and the last instruction.

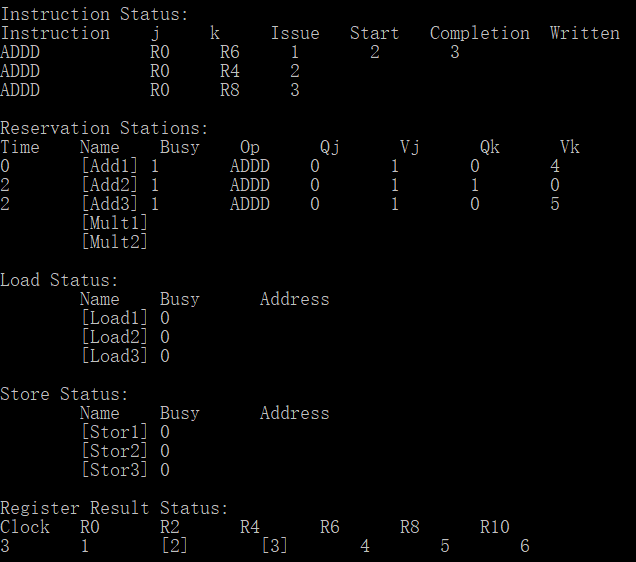
Cycle 1



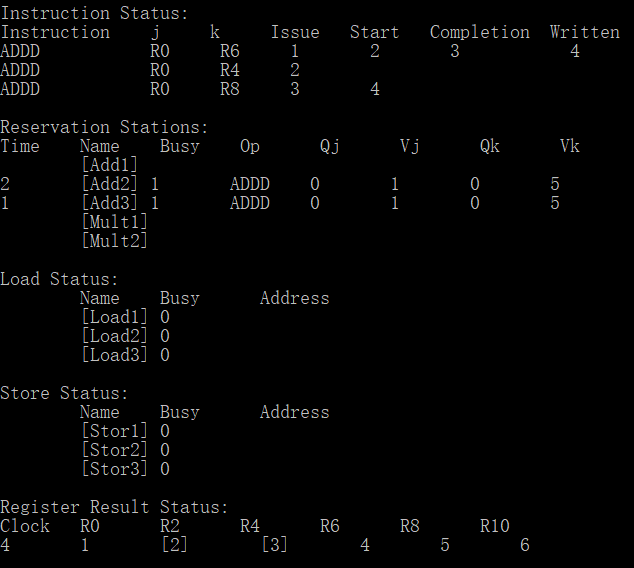
Cycle 2



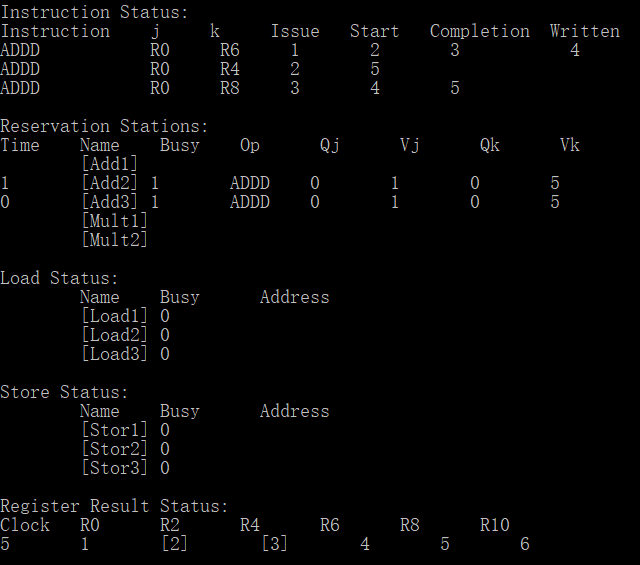
Cycle 3



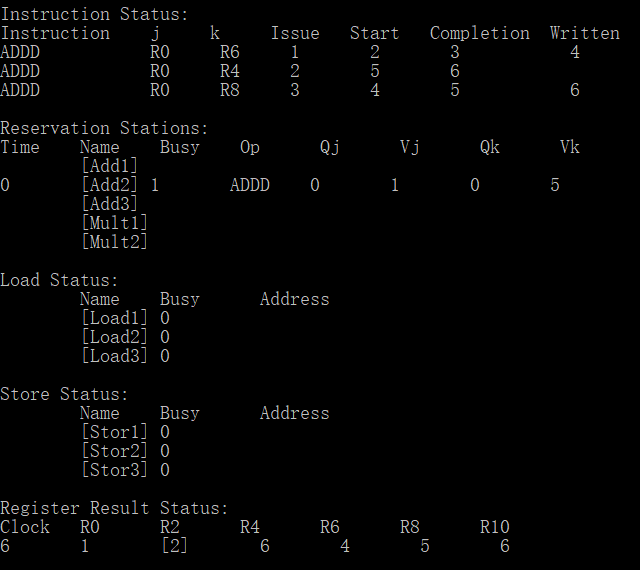
Cycle 4



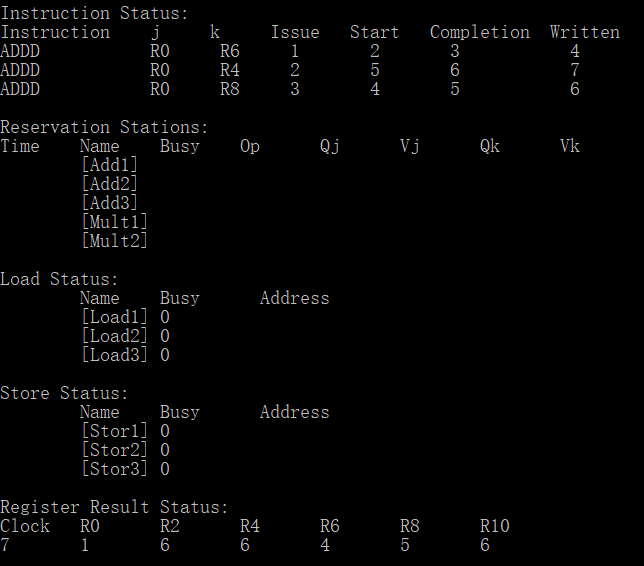
Cycle 5



Cycle 6



Cycle 7



* 1. The Long Example

Instructions:

LD R6 10

LD R2 12

MULTD R0 R2 R4

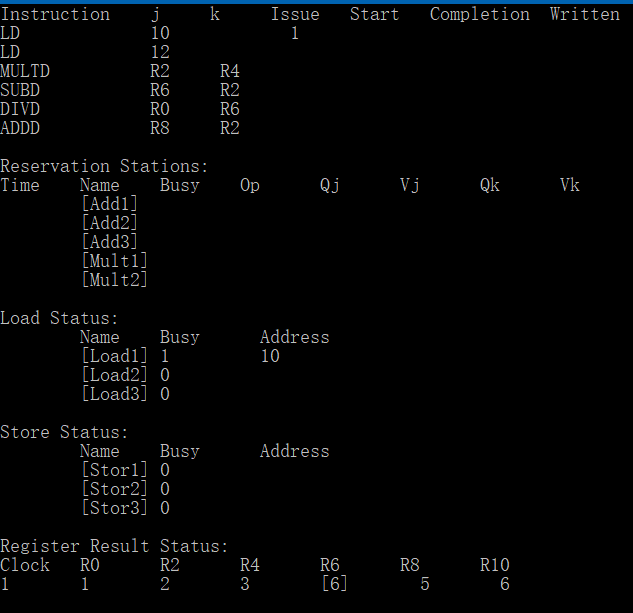
SUBD R8 R6 R2

DIVD R10 R0 R6

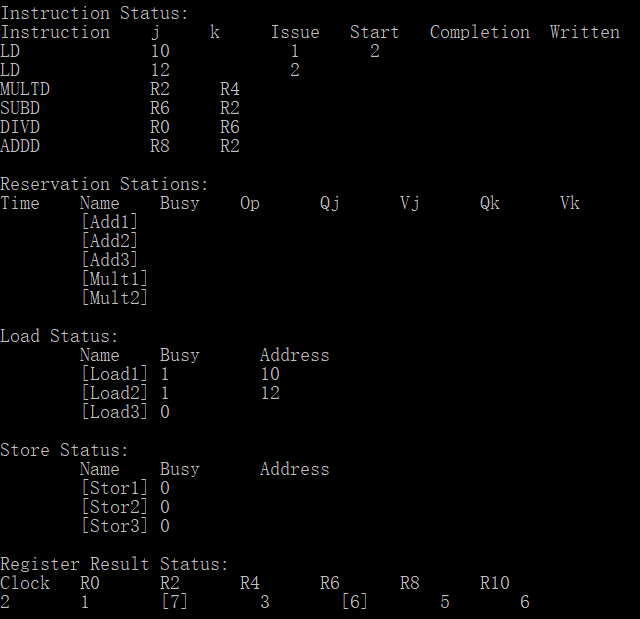
ADDD R6 R8 R2

For simplicity, it is assumed that the LD operation load an immediate directly to the register.

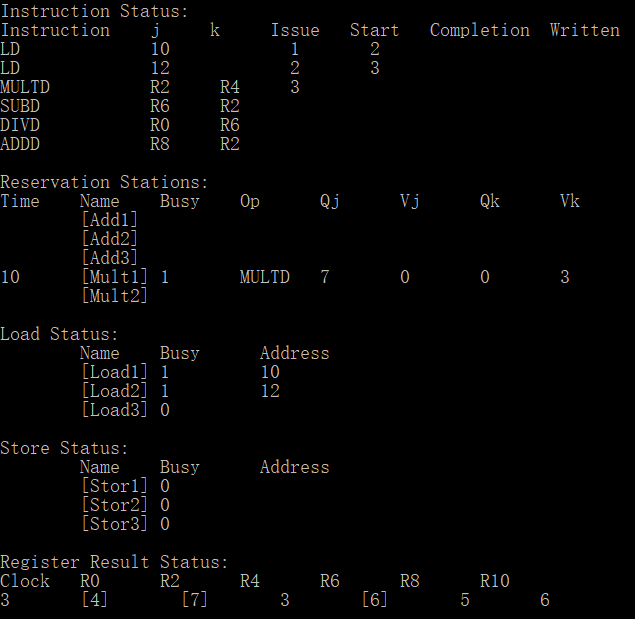
Cycle 1



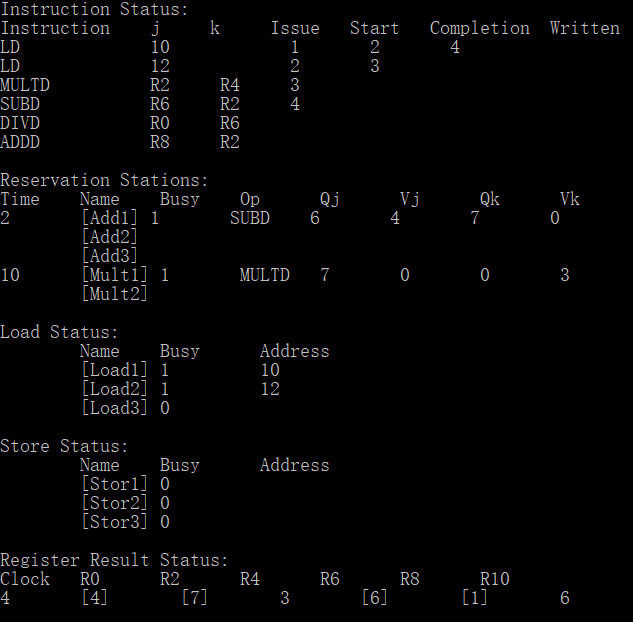
Cycle 2



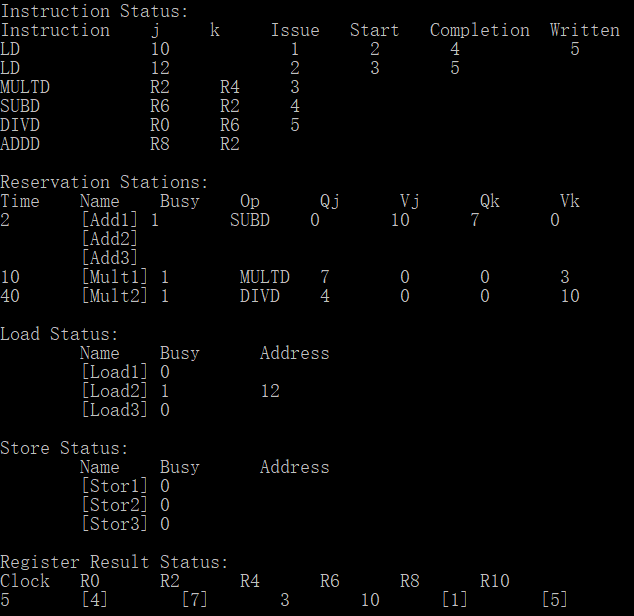
Cycle 3



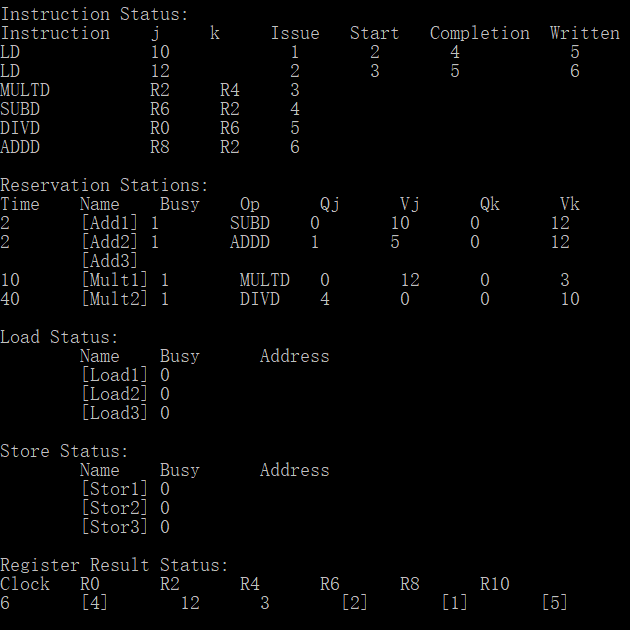
Cycle 4



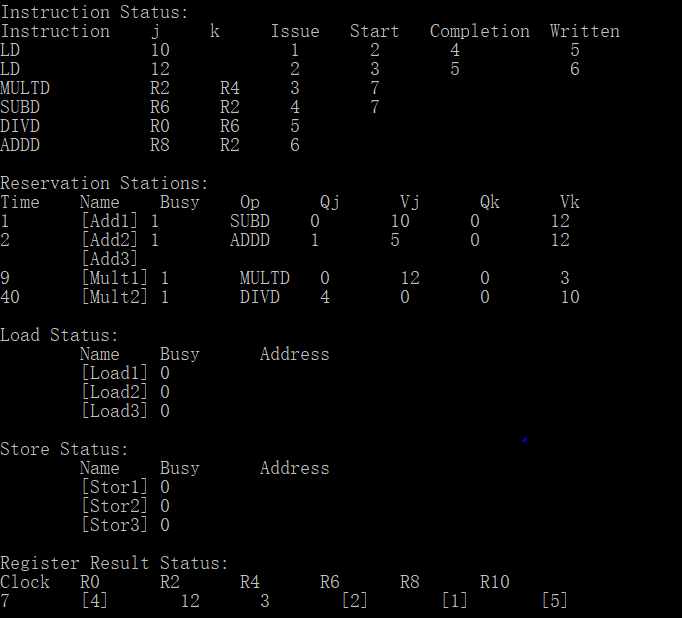
Cycle 5



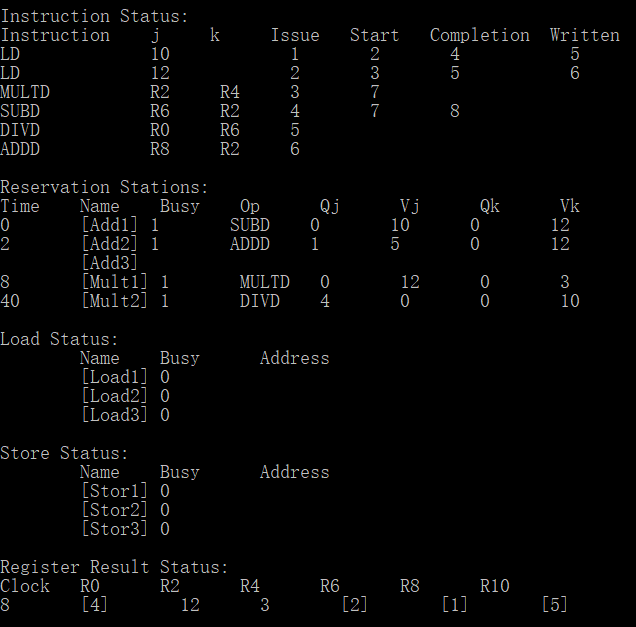
Cycle 6



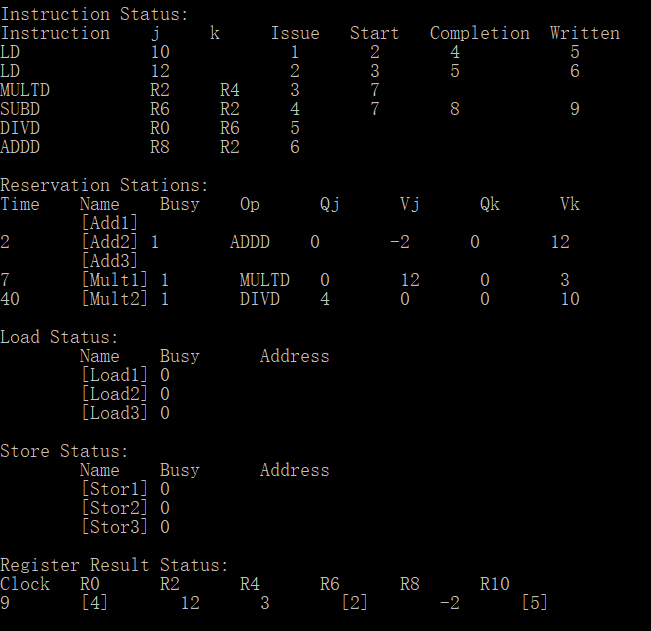
Cycle 7



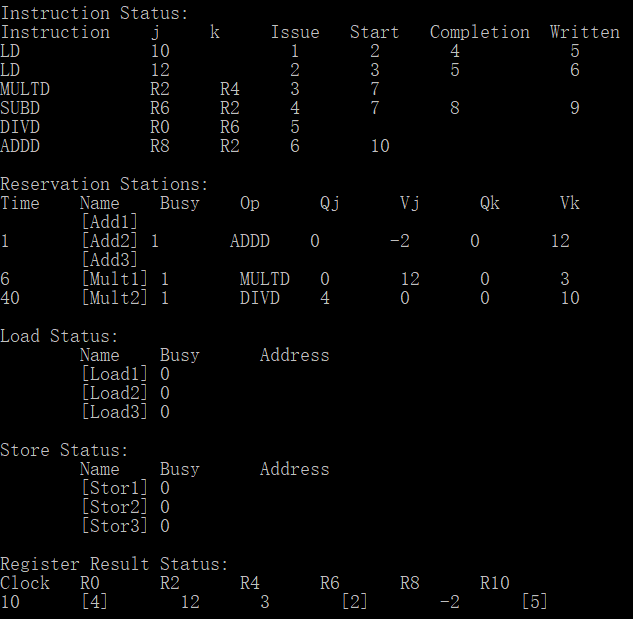
Cycle 8



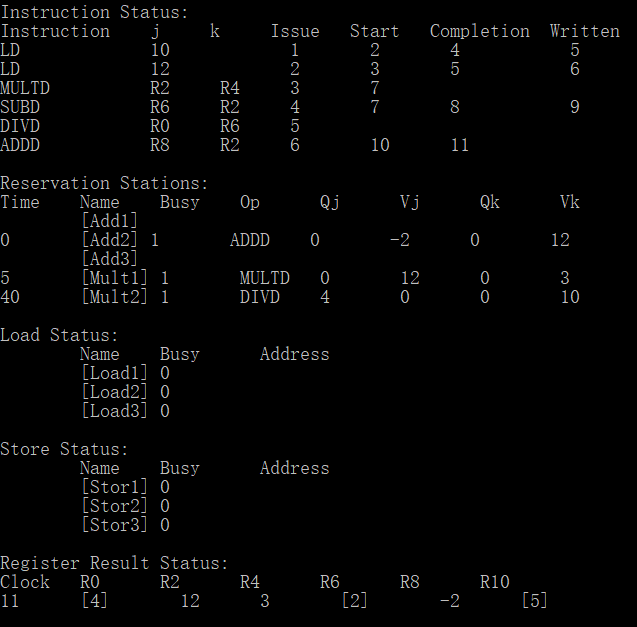
Cycle 9



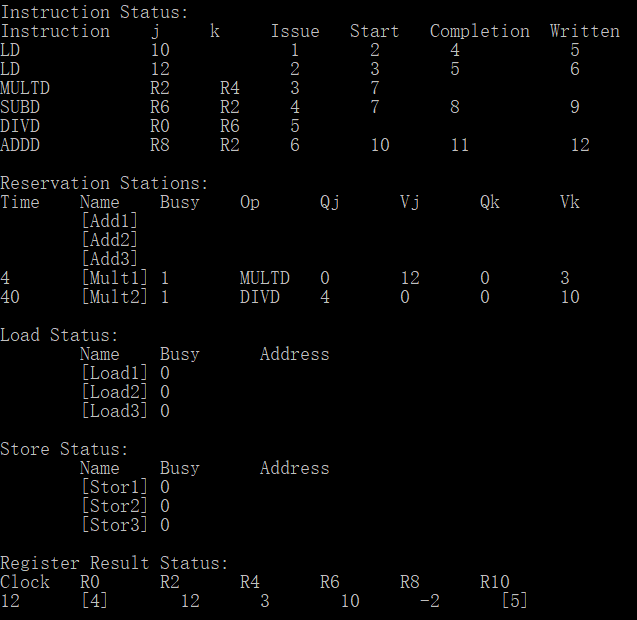
Cycle 10



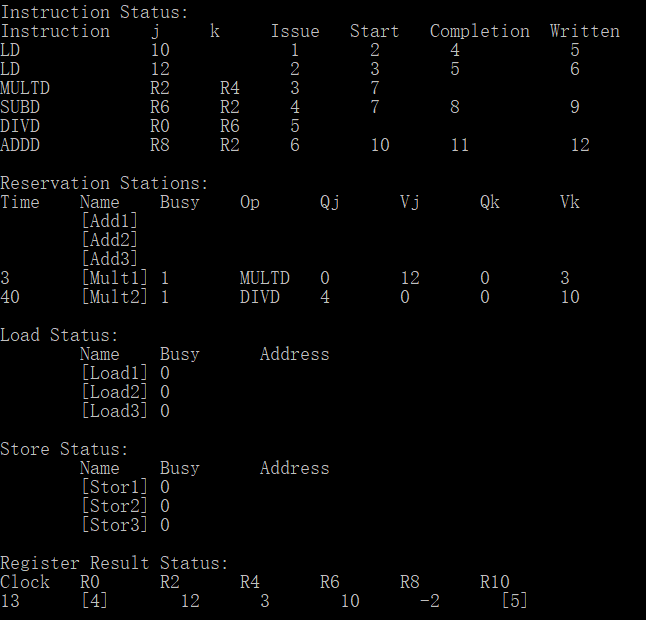
Cycle 11



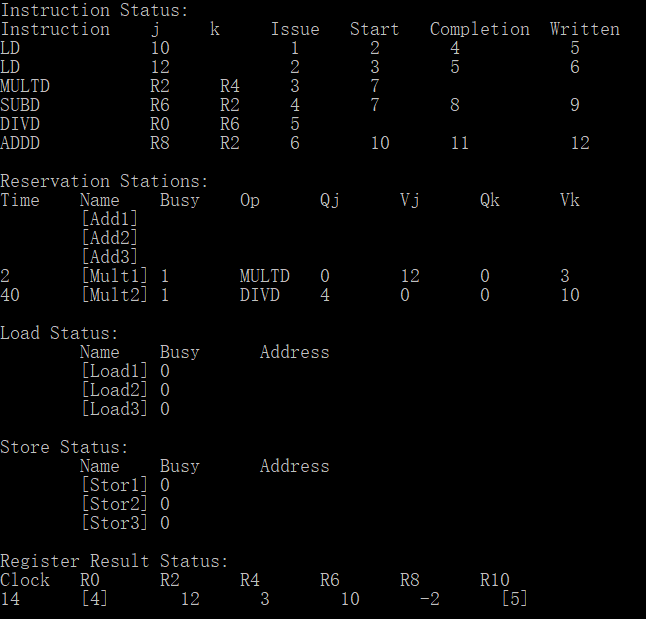
Cycle 12



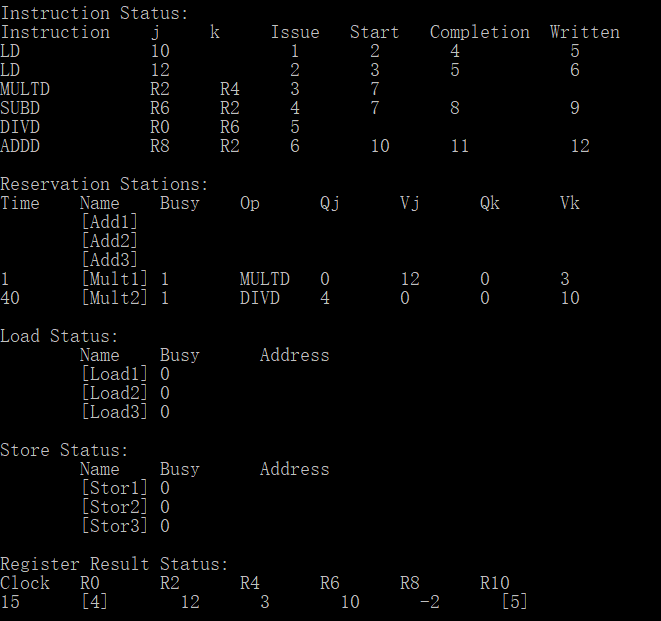
Cycle 13



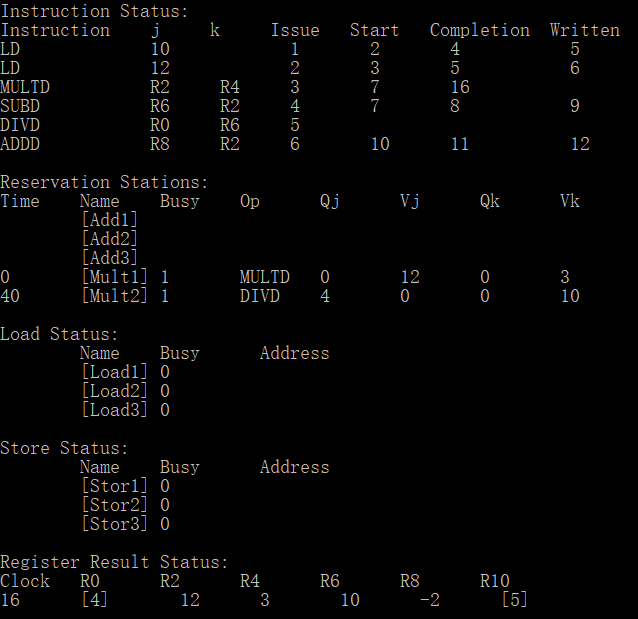
Cycle 14



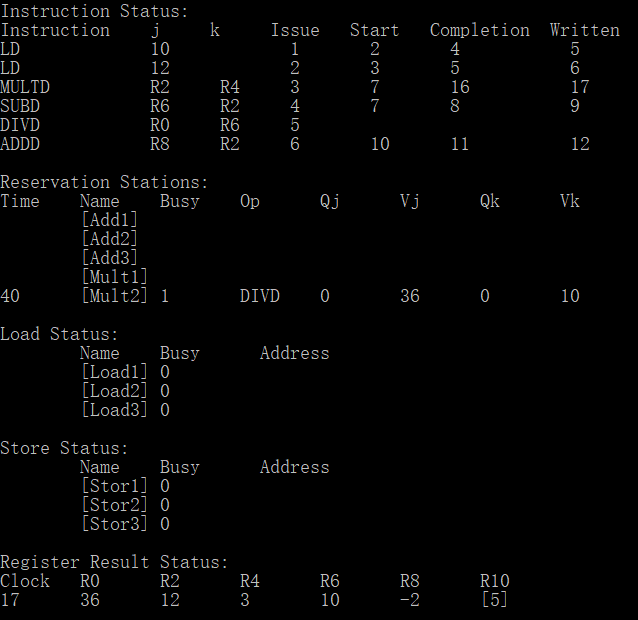
Cycle 15



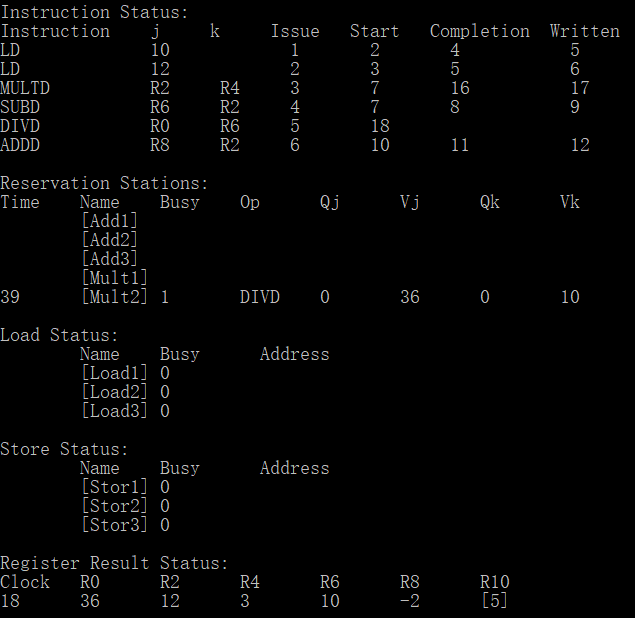
Cycle 16



Cycle 17



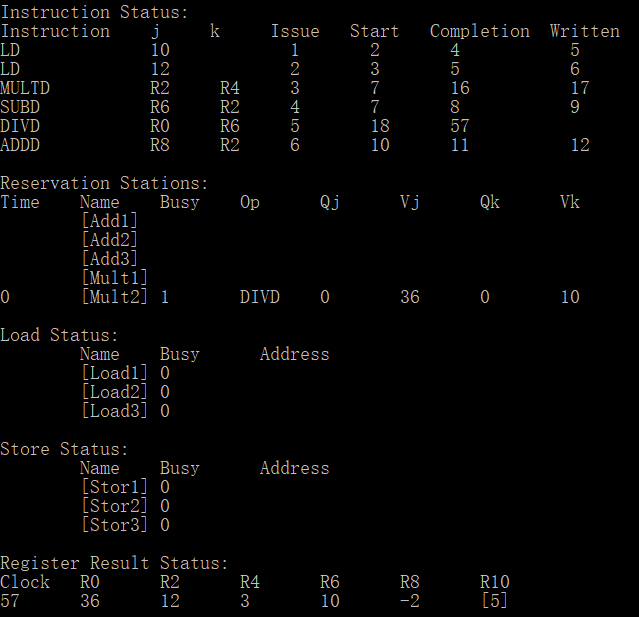
Cycle 18



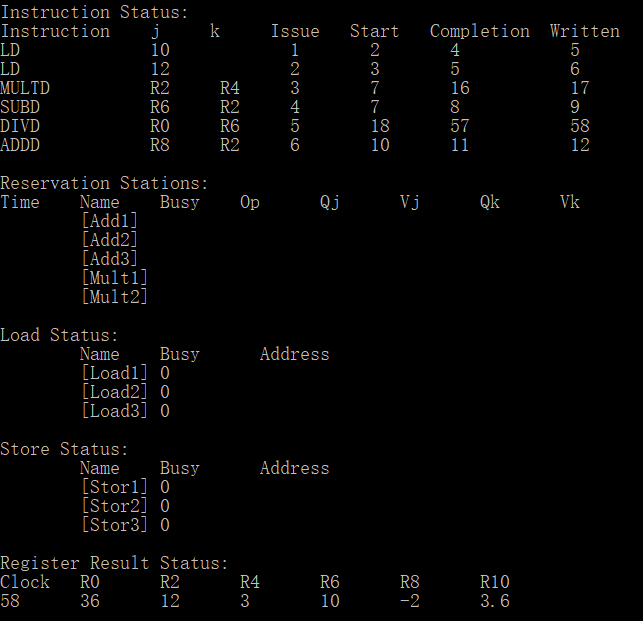
From cycle 18, only the DIVD instruction is in operation. For simplicity, the intermediate steps will be omitted.

Cycle 57

In this cycle, the execution of DIVD instruction is completed.



Cycle 58



* 1. Example with Stalls

Instructions:

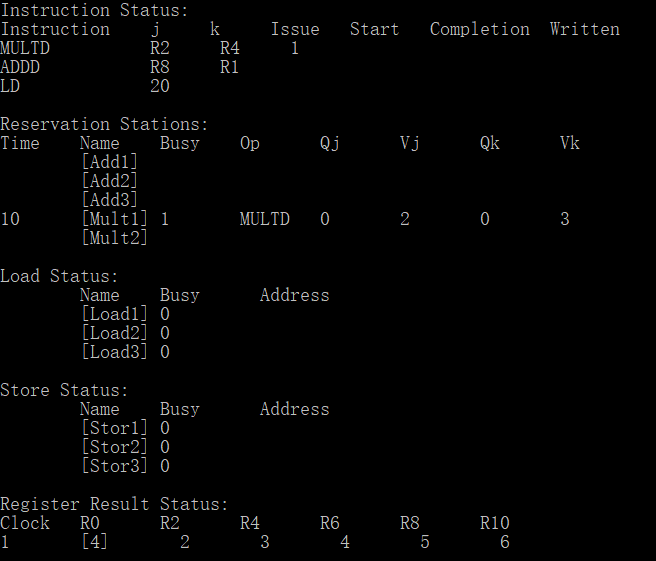
MULTD R0 R2 R4

ADDD R6 R8 R10

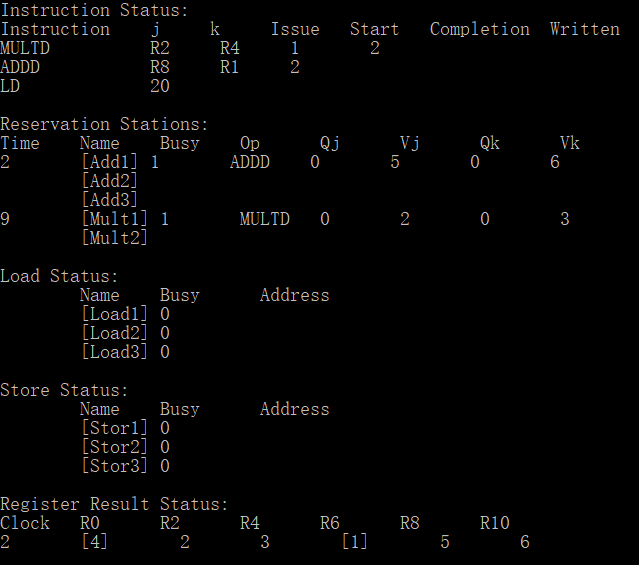
LD R8 20

The LD instruction wants to load a value to R8. Although the load buffer is available, it cannot be executed until ADDD has read the value in R8. So, there is stall for the LD instruction.

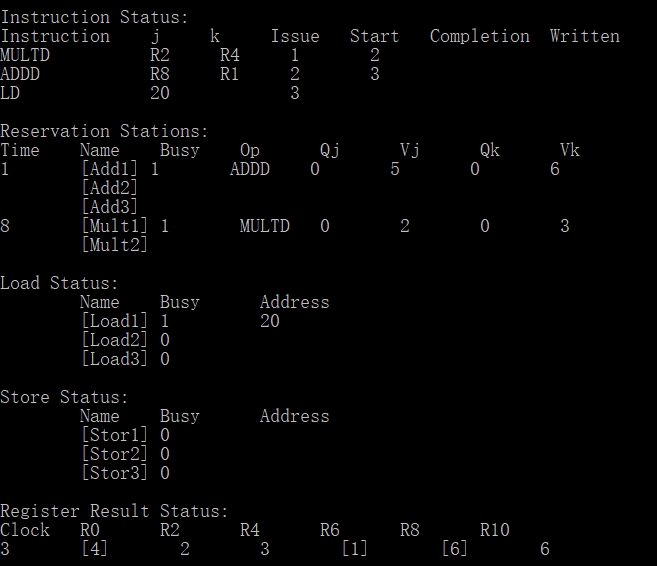
Cycle 1



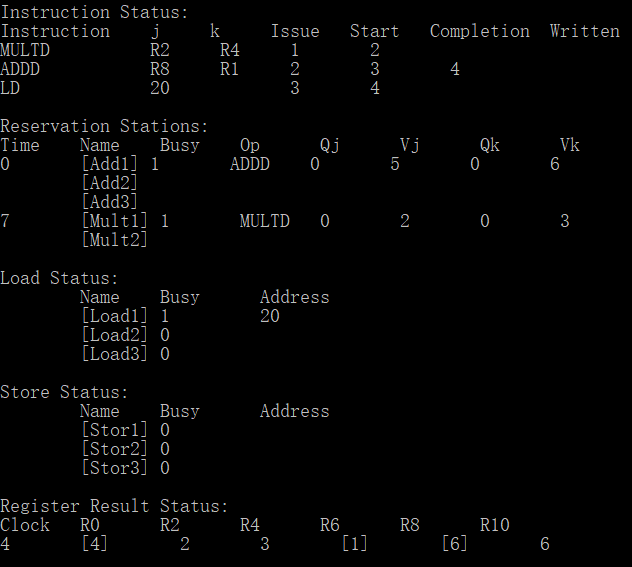
Cycle 2



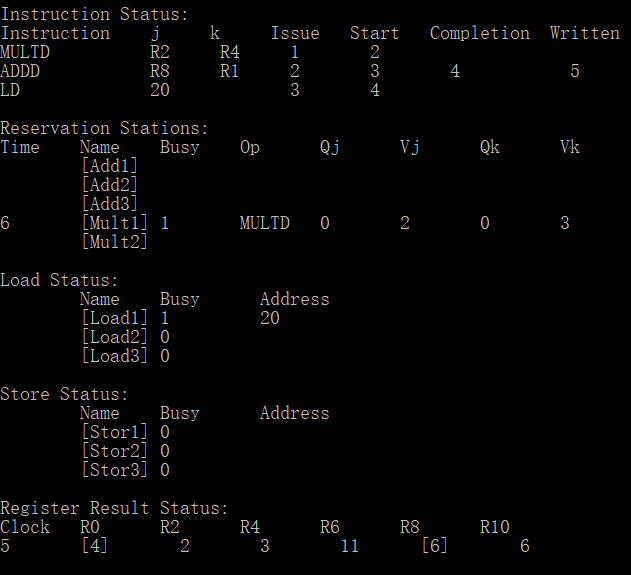
Cycle 3



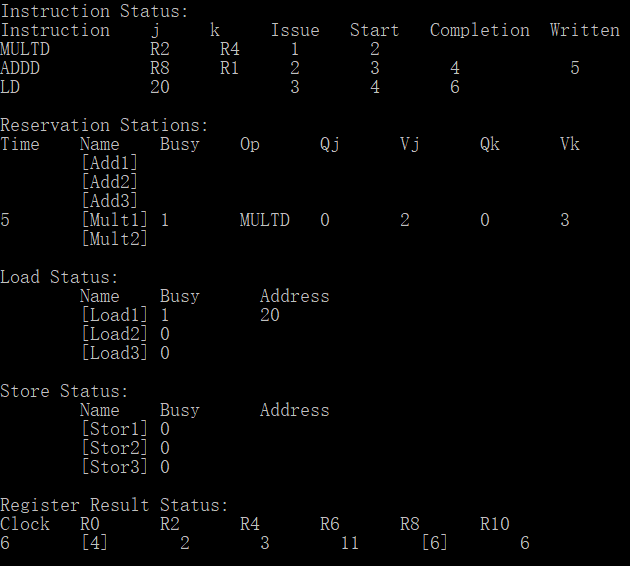
Cycle 4



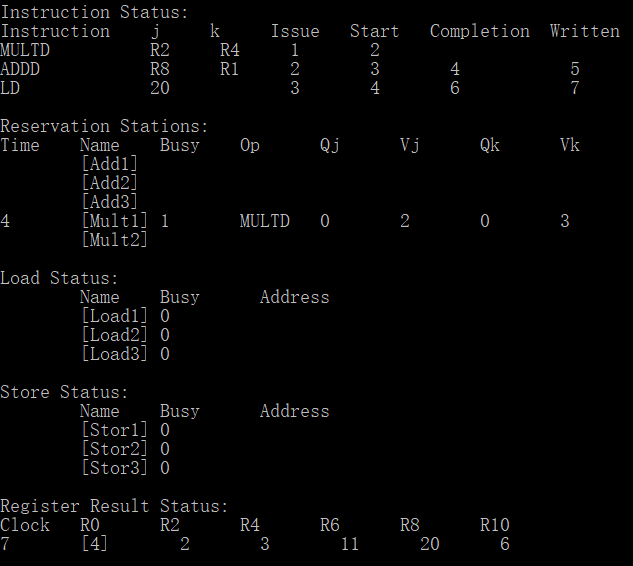
Cycle 5



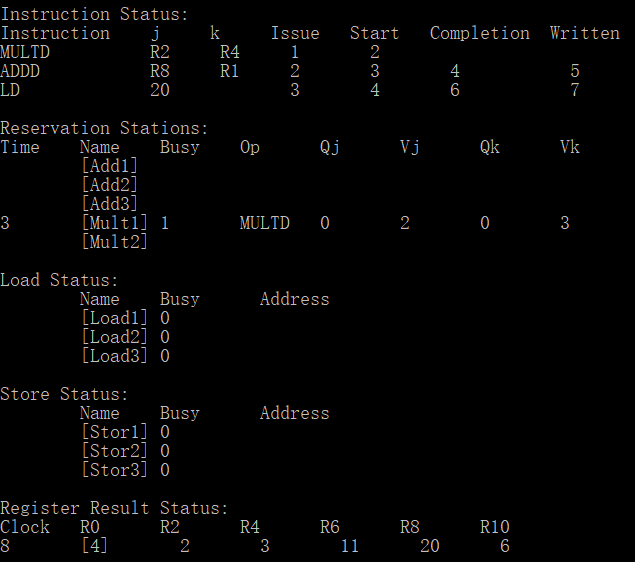
Cycle 6



Cycle 7

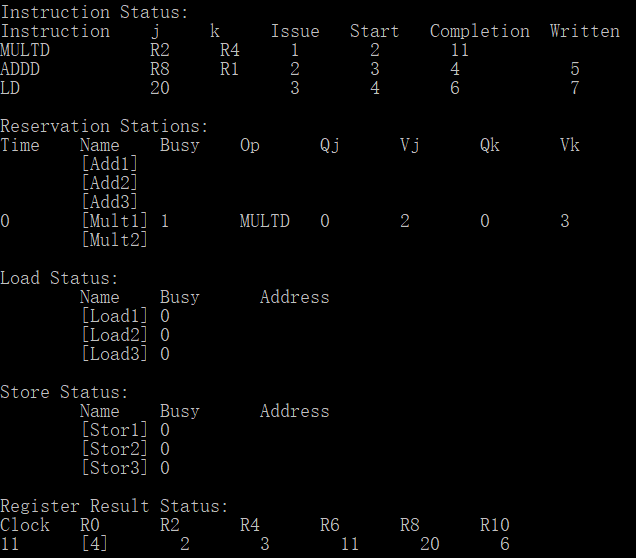


Cycle 8

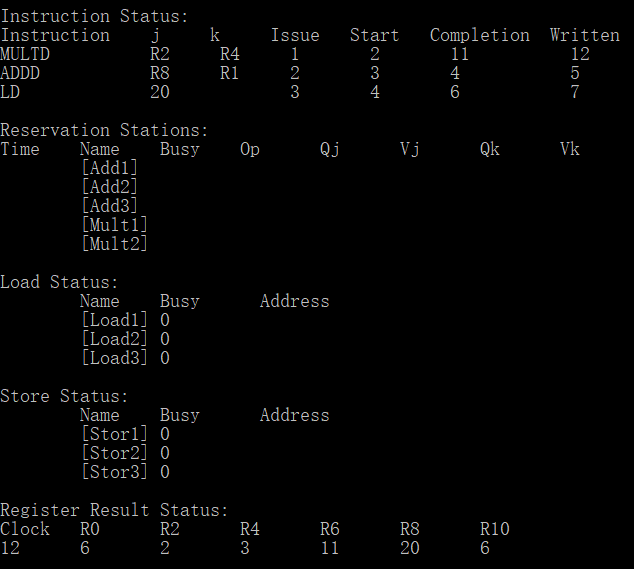


From cycle 8, there is only the MULTD instruction in operation. For simplicity, let’s jump to cycle 11.

Cycle 11



Cycle 12



Appendix

1. //
2. //  main.cpp
3. //  this program implements the tomasulo algorithm
4. //  instruction types supported: LD, SD, MULTD, DIVD, ADDD, SUBD
5. //  <store register>: [R0, Rn] where n is the number of registers (configured in assumptions)
6. //  Note: the program takes advantage of strncmp with 2 characters to find the correct register,
7. //  so if the number of registers is too large, the program may not execute correctly

10. #include <iostream>
11. #include <iomanip>
12. #include <cmath>
13. #include <fstream>
14. #include <stdio.h>
15. #include <string.h>
16. #include <stdlib.h>
18. #define MAXCHAR 1000
20. **using** **namespace** std;
22. // defining the data type for all instructions
23. **typedef** **struct** instruction
24. {
25. **char** type[100];
26. **char** dest\_reg[100];
27. **char** reg\_j[100];
28. **char** reg\_k[100];
29. **double** num;
30. **double** rs=0;
31. **double** issue=-1;
32. **int** start=0;
33. **double** completion=-1;
34. **double** written=-1;
35. **int** result;
36. **int** load=0;
37. **bool** completing=**false**;
38. **bool** writing=**false**;
39. } instruction;
40. // defining the data type for memory
41. **typedef** **struct** memory
42. {
43. **char** name[2];
44. **double** data;
45. **int** tag=0;
46. **bool** busy=**false**;
47. } memory;
48. // defining the data type for all RS
49. **typedef** **struct** reservation\_station
50. {
51. **char** name[100]="NONE";
52. **double** num;
53. **double** data\_j=NULL;
54. **double** data\_k=NULL;
55. **double** result;
56. **int** tag\_j=0;
57. **int** tag\_k=0;
58. **int** instr=-1;
59. **int** cycle\_count=0;
60. **int** cycles\_required=-999;
61. **bool** executing=**false**;
62. **bool** busy=**false**;
64. } reservation\_station;
66. **typedef** **struct** load\_store\_rs
67. {
68. **char** name[100]="NONE";
69. **double** num;
70. **double** address=NULL;
71. **int** instr=-1;
72. **int** tag=0;
73. **int** cycle\_count=0;
74. **int** cycles\_required=-999;
75. **bool** executing=**false**;
76. **bool** busy=**false**;
77. } load\_store\_rs;
79. **void** header(**int** n);
80. **template**<**typename** T> **void** printElement(T t, **const** **int**& width);
81. **template**<**typename** T> **void** printInstructionStatus(T t, **const** **int**& width);
82. **template**<**typename** T> **void** printLoadStatus(T t, **const** **int**& width);
83. **template**<**typename** T> **void** printStoreStatus(T t, **const** **int**& width);
84. **template**<**typename** T> **void** printStationStatus(T t, **const** **int**& width);
85. **template**<**typename** T> **void** printRegisterStatus(T t, **const** **int**& width);
87. **int** main() {
88. // ==================== ASSUMPTIONS ====================
89. // all assumptions are conssitent with the project description
90. **int** addCycles = 2;
91. **int** subCycles = 2;
92. **int** multCycles = 10;
93. **int** diviCycles = 40;
94. **int** loadCycles = 3;
95. **int** storeCycles = 3;
96. **const** **int** addReservationStations = 3;
97. **const** **int** mulReservationStations = 2;
98. **const** **int** loadReservationStations = 3;
99. **const** **int** storeReservationStations = 3;
100. **const** **int** maxInstructions = 16;
101. **const** **int** numRegisters = 6;
102. // initializing the values in registers, add or delete entries depending on the number of registers
103. **float** dataRegisters[numRegisters] = {1, 2, 3, 4, 5, 6};
104. //char\* filename = "raw.txt";
105. //char\* filename = "waw.txt";
106. **char**\* filename = "war.txt";
107. //char\* filename = "sdld.txt";
108. //char\* filename = "stall.txt";
109. //char\* filename = "long.txt";
111. // ==================== STRUCTURE INITIALIZATION ====================
112. instruction instruction\_list[maxInstructions];
113. memory  data\_registers[numRegisters];
115. **char** rs\_num[3];
116. **for** (**int** i=0; i < numRegisters; i++){
117. strcpy(data\_registers[i].name, "R");
118. sprintf(rs\_num, "%i", i\*2);
119. strcat(data\_registers[i].name, rs\_num);
120. data\_registers[i].data = dataRegisters[i];
121. }
123. reservation\_station add\_rs[addReservationStations];
124. reservation\_station mul\_rs[mulReservationStations];
126. **int** j;
127. **for** (j=0; j < addReservationStations; j++) {
128. add\_rs[j].num = j + 1;
129. }
130. **int** k;
131. **for** (k=0; k < mulReservationStations; k++) {
132. // each reservation station needs a unique number
133. mul\_rs[k].num = j + k + 1;
134. }
136. load\_store\_rs load\_rs[loadReservationStations];
137. load\_store\_rs store\_rs[storeReservationStations];
139. **for** (**int** i=0; i < loadReservationStations; i++) {
140. load\_rs[i].num = addReservationStations + mulReservationStations + i + 1;
141. }
143. **for** (**int** i=0; i < storeReservationStations; i++) {
144. store\_rs[i].num = addReservationStations + mulReservationStations + loadReservationStations + i + 1;
145. }
146. // 1,2,3 add rs;4,5 mul rs;6,7,8 load rs;9,10,11 store rs
147. // ==================== VARIABLE INITIALIZATION ====================
148. **int** lineCount = 0;
149. **int** completedInstr = 0;
150. **int** issuedInstr = 0;
151. **int** writtenInstr = 0;
152. **int** clockCycles = 0;
153. **int** completed\_rs = -1;
154. **double** cdb\_data = 0;
155. **bool** issueSuccessful = **false**;
156. **bool** cdb\_busy = **false**;
158. // ==================== READ IN INSTRUCTIONS ====================
159. **FILE** \*fp;
160. **char** mystring[MAXCHAR];

163. fp = fopen(filename, "r");
164. **if** (fp == NULL){
165. printf("Could not open file %s",filename);
166. **return** 1;
167. }
168. **while** (fgets(mystring, MAXCHAR, fp) != NULL){
169. **const** **char** t[2] = "\t";
170. **const** **char** s[2] = " ";
171. **char** \*token;
172. //cpoying instruction type
173. token = strtok(mystring, t);
174. instruction\_list[lineCount].num = atof(token);
175. strcpy(instruction\_list[lineCount].type, strtok(token, s));
176. // copying destination register
177. token = strtok(NULL, s);
178. strcpy(instruction\_list[lineCount].dest\_reg, token);
179. // copying operand j
180. token = strtok(NULL, s);
181. **if** (strcmp(instruction\_list[lineCount].type, "LD") == 0) {
182. instruction\_list[lineCount].load = atoi(token);
183. }
184. **else** {
185. strcpy(instruction\_list[lineCount].reg\_j, token);
186. }
188. /\* read in last part of instruction, if not load or store \*/
189. **if** (strcmp(instruction\_list[lineCount].type, "LD") == 0 or strcmp(instruction\_list[lineCount].type, "SD") == 0)
190. {
191. strcpy(instruction\_list[lineCount].reg\_k, " ");
192. }
193. **else**
194. {
195. token = strtok(NULL, s);
196. strcpy(instruction\_list[lineCount].reg\_k, token);
197. }
199. lineCount += 1;
200. }
201. fclose(fp);
203. // ==================== MAIN SIMULATION LOOP ====================
205. **while** (writtenInstr < lineCount)
206. {
207. // ================== WRITING INSTRUCTIONS ==================
208. **if** (completed\_rs != -1) {
209. // broadcasting data
210. **for** (**int** i=0; i < addReservationStations; i++) {
211. **if** (add\_rs[i].tag\_j == completed\_rs) {
212. add\_rs[i].data\_j = cdb\_data;
213. add\_rs[i].tag\_j = 0;
214. }
215. **if** (add\_rs[i].tag\_k == completed\_rs) {
216. add\_rs[i].data\_k = cdb\_data;
217. add\_rs[i].tag\_k = 0;
218. }
220. // clear reservation station
221. **if** (add\_rs[i].num == completed\_rs) {
222. add\_rs[i].busy = **false**;
223. add\_rs[i].cycle\_count = 0;
224. add\_rs[i].cycles\_required = -999;
225. add\_rs[i].executing = **false**;
226. }
227. }
228. **for** (**int** i=0; i < mulReservationStations; i++) {
229. **if** (mul\_rs[i].tag\_j == completed\_rs) {
230. mul\_rs[i].data\_j = cdb\_data;
231. mul\_rs[i].tag\_j = 0;
232. }
233. **if** (mul\_rs[i].tag\_k == completed\_rs) {
234. mul\_rs[i].data\_k = cdb\_data;
235. mul\_rs[i].tag\_k = 0;
236. }
238. // clear reservation station
239. **if** (mul\_rs[i].num == completed\_rs) {
240. mul\_rs[i].busy = **false**;
241. mul\_rs[i].cycle\_count = 0;
242. mul\_rs[i].cycles\_required = -999;
243. mul\_rs[i].executing = **false**;
244. }
245. }
246. **for** (**int** i=0; i < loadReservationStations; i++) {
247. **if** (load\_rs[i].tag == completed\_rs) {
248. load\_rs[i].address = cdb\_data;
249. load\_rs[i].tag = 0;
250. //load\_rs[i].executing = true;
251. }
252. // clear reservation station
253. **if** (load\_rs[i].num == completed\_rs) {
254. load\_rs[i].busy = **false**;
255. load\_rs[i].cycle\_count = 0;
256. load\_rs[i].cycles\_required = -999;
257. load\_rs[i].executing = **false**;
258. }
259. }
260. **for** (**int** i=0; i < storeReservationStations; i++) {
261. **if** (store\_rs[i].tag == completed\_rs) {
262. store\_rs[i].address = cdb\_data;
263. store\_rs[i].tag = 0;
264. //store\_rs[i].executing = true;
265. }
266. // clear reservation station
267. **if** (store\_rs[i].num == completed\_rs) {
268. store\_rs[i].busy = **false**;
269. store\_rs[i].cycle\_count = 0;
270. store\_rs[i].cycles\_required = -999;
271. store\_rs[i].executing = **false**;
272. }
273. }
274. // write to memory
275. **for** (**int** i=0; i < numRegisters; i++) {
276. **if** (data\_registers[i].tag == completed\_rs) {
277. data\_registers[i].data = cdb\_data;
278. data\_registers[i].busy = **false**;
279. data\_registers[i].tag = 0;
280. }
281. }
283. // just for bookkeeping - instruction written cycle number
284. **for** (**int** j=0; j < issuedInstr; j++) {
285. **if** (instruction\_list[j].rs == completed\_rs) {
286. **if** (instruction\_list[j].written == -1) {
287. instruction\_list[j].written = clockCycles+1;
288. }
289. }
290. }
292. writtenInstr += 1;
293. // reset
294. completed\_rs = -1;
295. cdb\_data = 0;
296. }

299. // ================== ISSUING INSTRUCTIONS ==================
300. **if** (issuedInstr < lineCount) {
301. // issue instruction 0...then 1...then n..etc. (& increment instruction cycle if successful)
302. **if** (strcmp(instruction\_list[issuedInstr].type, "ADDD") == 0 || strcmp(instruction\_list[issuedInstr].type, "SUBD") == 0) {
303. // if addition or subtraction, adding to reservation station
304. **for** (**int** l=0; l < addReservationStations; l++) {
305. **if** (issueSuccessful == **false**) {
306. **if** (add\_rs[l].busy == **false**) {
307. **for** (**int** i=0; i < numRegisters; i++) {
308. // j operand
309. **if** (strcmp(instruction\_list[issuedInstr].reg\_j, data\_registers[i].name) == 0) {
310. add\_rs[l].data\_j = data\_registers[i].data;
311. **if** (data\_registers[i].tag == 0) {
312. }
313. **else** {
314. add\_rs[l].tag\_j = data\_registers[i].tag;
315. }
316. }
317. // k operand
318. **if** (strncmp(instruction\_list[issuedInstr].reg\_k, data\_registers[i].name, 2) == 0) {
319. **if** (data\_registers[i].tag == 0) {
320. add\_rs[l].data\_k = data\_registers[i].data;
321. }
322. **else** {
323. add\_rs[l].tag\_k = data\_registers[i].tag;
324. }
325. }
326. // destination register
327. **if** (strcmp(instruction\_list[issuedInstr].dest\_reg, data\_registers[i].name) == 0) {
328. data\_registers[i].tag = add\_rs[l].num;
329. }
330. }
331. add\_rs[l].busy = **true**;
332. instruction\_list[issuedInstr].rs = add\_rs[l].num;
333. add\_rs[l].cycle\_count = 0;
334. instruction\_list[issuedInstr].issue = clockCycles+1;
335. issueSuccessful = **true**;
336. **if** (strcmp(instruction\_list[issuedInstr].type, "ADDD") == 0) {
337. strcpy(add\_rs[l].name, "ADDD");
338. add\_rs[l].cycles\_required = addCycles;
340. }
341. **else** {
342. add\_rs[l].cycles\_required = subCycles;
343. strcpy(add\_rs[l].name, "SUBD");
344. }
345. }
346. }
347. }
348. }
349. // if loading
350. **else** **if** (strcmp(instruction\_list[issuedInstr].type, "LD") == 0) {
351. **for** (**int** l=0; l < loadReservationStations; l++) {
352. **if** (issueSuccessful == **false**) {
353. **if** (load\_rs[l].busy == **false**) {
354. **for** (**int** i=0; i < numRegisters; i++) {
355. // load value
356. load\_rs[l].address = instruction\_list[issuedInstr].load;
357. // destination register
358. **if** (strncmp(instruction\_list[issuedInstr].dest\_reg, data\_registers[i].name, 2) == 0) {
359. data\_registers[i].tag = load\_rs[l].num;
360. }
361. }
362. load\_rs[l].busy = **true**;
363. load\_rs[l].cycles\_required = loadCycles;
364. instruction\_list[issuedInstr].rs = load\_rs[l].num;
365. load\_rs[l].cycle\_count = 0;
366. instruction\_list[issuedInstr].issue = clockCycles+1;
367. issueSuccessful = **true**;
368. }
369. }
370. }
371. }
372. // if storing
373. **else** **if** (strcmp(instruction\_list[issuedInstr].type, "SD") == 0) {
374. **for** (**int** l=0; l < storeReservationStations; l++) {
375. **if** (issueSuccessful == **false**) {
376. **if** (store\_rs[l].busy == **false**) {
377. **for** (**int** i=0; i < numRegisters; i++) {
378. **if** (strcmp(instruction\_list[issuedInstr].dest\_reg, data\_registers[i].name) == 0) {
379. **if** (data\_registers[i].tag == 0) {
380. store\_rs[l].address = data\_registers[i].data;
381. }
382. **else** {
383. store\_rs[l].tag = data\_registers[i].tag;
384. }
385. }
386. // destination register
387. **if** (strncmp(instruction\_list[issuedInstr].reg\_j, data\_registers[i].name, 2) == 0) {
388. data\_registers[i].tag = store\_rs[l].num;
389. }
390. }
391. store\_rs[l].busy = **true**;
392. store\_rs[l].cycles\_required = storeCycles;
393. instruction\_list[issuedInstr].rs = store\_rs[l].num;
394. store\_rs[l].cycle\_count = 0;
395. instruction\_list[issuedInstr].issue = clockCycles+1;
396. issueSuccessful = **true**;
397. }
398. }
399. }
400. }
401. // if multiplication or division
402. **else** {
403. // adding to reservation station
404. **for** (**int** l=0; l < mulReservationStations; l++) {
405. **if** (issueSuccessful == **false**) {
406. **if** (mul\_rs[l].busy == **false**) {
407. **for** (**int** i=0; i < numRegisters; i++) {
408. // destination register
409. **if** (strcmp(instruction\_list[issuedInstr].dest\_reg, data\_registers[i].name) == 0) {
410. data\_registers[i].tag = mul\_rs[l].num;
411. }
412. // j operand
413. **if** (strcmp(instruction\_list[issuedInstr].reg\_j, data\_registers[i].name) == 0) {
414. **if** (data\_registers[i].tag == 0) {
415. mul\_rs[l].data\_j = data\_registers[i].data;
416. }
417. **else** {
418. mul\_rs[l].tag\_j = data\_registers[i].tag;
419. }
420. }
421. // k operand
422. **if** (strncmp(instruction\_list[issuedInstr].reg\_k, data\_registers[i].name, 2) == 0) {
423. **if** (data\_registers[i].tag == 0) {
424. mul\_rs[l].data\_k = data\_registers[i].data;
425. }
426. **else** {
427. mul\_rs[l].tag\_k = data\_registers[i].tag;
428. }
429. }
430. }
431. mul\_rs[l].busy = **true**;
432. instruction\_list[issuedInstr].rs = mul\_rs[l].num;
433. mul\_rs[l].cycle\_count = 0;
434. instruction\_list[issuedInstr].issue = clockCycles+1;
435. issueSuccessful = **true**;
436. **if** (strcmp(instruction\_list[issuedInstr].type, "MULTD") == 0) {
437. strcpy(mul\_rs[l].name, "MULTD");
438. mul\_rs[l].cycles\_required = multCycles;
440. }
441. **else** {
442. mul\_rs[l].cycles\_required = diviCycles;
443. strcpy(mul\_rs[l].name, "DIVD");
444. }
445. }
446. }
447. }
448. }
450. }

453. // ================== COMPLETING AND EXECUTING INSTRUCTION CHECK ==================
454. // ordered by increasing reservation station number
455. cdb\_busy = **false**;
456. // add reservations
457. **for** (**int** i=0; i < addReservationStations; i++){
459. // executing instructions
460. **if** (add\_rs[i].busy and add\_rs[i].executing) {
461. **if** (add\_rs[i].cycle\_count < add\_rs[i].cycles\_required) {
462. add\_rs[i].cycle\_count += 1;
463. }
464. }
465. // setting mark of beginning of execution
466. **for** (**int** j=0; j < issuedInstr; j++) {
468. **if** (instruction\_list[j].rs == add\_rs[i].num) {
469. **if** (add\_rs[i].cycle\_count==1) {
470. instruction\_list[j].start = clockCycles+1;
471. }
472. }
473. }
474. // completing instruction
475. **if** (add\_rs[i].cycle\_count == add\_rs[i].cycles\_required) {
476. **for** (**int** j=0; j < issuedInstr; j++) {
477. // just tracking completion cycle - simply for bookkeeping
478. **if** (instruction\_list[j].rs == add\_rs[i].num) {
479. **if** (instruction\_list[j].completion == -1) {
480. instruction\_list[j].completion = clockCycles+1;
481. completed\_rs = add\_rs[i].num;
482. completedInstr += 1;
483. }
484. }
485. }
486. //writing to CDB
487. **if** (!cdb\_busy) {
488. **if** (strcmp(add\_rs[i].name, "ADDD") == 0) {
489. cdb\_data = add\_rs[i].data\_j + add\_rs[i].data\_k;
490. }
491. **else** {
492. cdb\_data = add\_rs[i].data\_j - add\_rs[i].data\_k;
493. }
494. cdb\_busy = **true**;
495. //completed\_rs = add\_rs[i].num;
496. }
497. }
498. }
499. // mult reservations
500. **for** (**int** i=0; i < mulReservationStations; i++) {
502. // executing instructions
503. **if** (mul\_rs[i].busy and mul\_rs[i].executing) {
504. // only increment if not yet reached
505. **if** (mul\_rs[i].cycle\_count < mul\_rs[i].cycles\_required) {
506. mul\_rs[i].cycle\_count += 1;
507. }
508. }
509. // setting mark of beginning of execution
510. **for** (**int** j=0; j < issuedInstr; j++) {
512. **if** (instruction\_list[j].rs == mul\_rs[i].num) {
513. **if** (mul\_rs[i].cycle\_count==1) {
514. instruction\_list[j].start = clockCycles+1;
515. }
517. }
518. }
519. // completing instruction
520. **if** (mul\_rs[i].cycle\_count == mul\_rs[i].cycles\_required) {
521. **for** (**int** j=0; j < issuedInstr; j++) {
522. // just tracking completion cycle - simply for bookkeeping
523. **if** (instruction\_list[j].rs == mul\_rs[i].num) {
524. **if** (instruction\_list[j].completion == -1) {
525. instruction\_list[j].completion = clockCycles+1;
526. completed\_rs = mul\_rs[i].num;
527. completedInstr += 1;
528. }
529. }
530. }
531. //writing to CDB
532. **if** (!cdb\_busy) {
533. **if** (strcmp(mul\_rs[i].name, "MULTD") == 0) {
534. cdb\_data = mul\_rs[i].data\_j \* mul\_rs[i].data\_k;
535. }
536. **else** {
537. cdb\_data = mul\_rs[i].data\_j / mul\_rs[i].data\_k;
538. }
539. cdb\_busy = **true**;
540. //completed\_rs = mul\_rs[i].num;
541. }
542. }
543. }
544. // load reservation
545. **for** (**int** i=0; i < loadReservationStations; i++) {
547. // executing instructions
548. **if** (load\_rs[i].busy and load\_rs[i].executing) {
549. // only increment if not yet reached
550. **if** (load\_rs[i].cycle\_count < load\_rs[i].cycles\_required) {
551. load\_rs[i].cycle\_count += 1;
552. }
553. }
554. // setting mark of beginning of execution
555. **for** (**int** j=0; j < issuedInstr; j++) {
557. **if** (instruction\_list[j].rs == load\_rs[i].num) {
558. **if** (load\_rs[i].cycle\_count==1) {
559. instruction\_list[j].start = clockCycles+1;
560. }
562. }
563. }
564. // completing instruction
565. **if** (load\_rs[i].cycle\_count == load\_rs[i].cycles\_required) {
566. **for** (**int** j=0; j < issuedInstr; j++) {
567. // just tracking completion cycle - simply for bookkeeping
568. **if** (instruction\_list[j].rs == load\_rs[i].num) {
569. **if** (instruction\_list[j].completion == -1) {
570. instruction\_list[j].completion = clockCycles+1;
571. completed\_rs = load\_rs[i].num;
572. completedInstr += 1;
573. }
574. }
575. }
576. //writing to CDB
577. **if** (!cdb\_busy) {
578. cdb\_data = load\_rs[i].address;
579. cdb\_busy = **true**;
580. //completed\_rs = load\_rs[i].num;
581. }
582. }
583. }
584. // store reservation
585. **for** (**int** i=0; i < storeReservationStations; i++) {

588. // executing instructions
589. **if** (store\_rs[i].busy and store\_rs[i].executing) {
590. cout << "cycle count" << store\_rs[i].cycle\_count << endl;
591. cout << "cycles required" << store\_rs[i].cycles\_required << endl;
592. // only increment if not yet reached
593. **if** (store\_rs[i].cycle\_count < store\_rs[i].cycles\_required) {
594. store\_rs[i].cycle\_count += 1;
595. }
596. }
597. // setting mark of beginning of execution
598. **for** (**int** j=0; j < issuedInstr; j++) {
600. **if** (instruction\_list[j].rs == store\_rs[i].num) {
601. **if** (store\_rs[i].cycle\_count==1) {
602. instruction\_list[j].start = clockCycles+1;
603. }
605. }
606. }
607. //completing instruction
608. **if** (store\_rs[i].cycle\_count == store\_rs[i].cycles\_required) {
609. **for** (**int** j=0; j < issuedInstr; j++) {
610. // just tracking completion cycle - simply for bookkeeping
611. **if** (instruction\_list[j].rs == store\_rs[i].num) {
612. **if** (instruction\_list[j].completion == -1) {
613. instruction\_list[j].completion = clockCycles+1;
614. completed\_rs = store\_rs[i].num;
615. completedInstr += 1;
616. }
617. }
618. }
619. //writing to CDB
620. **if** (!cdb\_busy) {
621. cdb\_data = store\_rs[i].address;
622. cdb\_busy = **true**;
623. //completed\_rs = store\_rs[i].num;
624. }
625. }
626. }
628. **for** (**int** l=0; l < addReservationStations; l++) {
629. **if** (add\_rs[l].busy == **true**) {
630. **if** (add\_rs[l].tag\_j == 0 and add\_rs[l].tag\_k == 0) {
631. add\_rs[l].executing = **true**;
632. }
633. }
634. }
635. **for** (**int** l=0; l < loadReservationStations; l++){
636. **if** (load\_rs[l].busy == **true**){
637. load\_rs[l].executing = **true**;
638. }
639. }
640. **for** (**int** l=0; l < storeReservationStations; l++) {
641. **if** (store\_rs[l].busy == **true**) {
642. **if** (store\_rs[l].tag == 0) {
643. store\_rs[l].executing = **true**;
644. }
645. }
646. }
647. **for** (**int** l=0; l < mulReservationStations; l++) {
648. **if** (mul\_rs[l].busy == **true**) {
649. **if** (mul\_rs[l].tag\_j == 0 and mul\_rs[l].tag\_k == 0) {
650. mul\_rs[l].executing = **true**;
651. }
652. }
653. }

656. // ================== PRINTING TO CONSOLE ==================
657. printInstructionStatus("test", 6);
658. **for** (**int** j=0; j < lineCount; j++) {
659. printElement(instruction\_list[j].type, 15);
660. **if** (strcmp(instruction\_list[j].type, "LD") == 0) {
661. printElement(instruction\_list[j].load, 6);
662. printElement(" ", 8);
663. }
664. **else** {
665. printElement(instruction\_list[j].reg\_j[0], 0);
666. printElement(instruction\_list[j].reg\_j[1], 6);
667. printElement(instruction\_list[j].reg\_k[0], 0);
668. printElement(instruction\_list[j].reg\_k[1], 6);
669. }
670. **if** (instruction\_list[j].issue == -1) {
671. printElement(" ", 8);
672. }
673. **else** printElement(instruction\_list[j].issue, 8);
674. **if** (instruction\_list[j].start!=0)
675. {
676. printElement(instruction\_list[j].start,8);
677. }
678. **else** printElement(" ", 8);
679. **if** (instruction\_list[j].completion == -1) {
680. printElement(" ", 12);
681. }
682. **else** printElement(instruction\_list[j].completion, 12);
683. **if** (instruction\_list[j].written == -1) {
684. printElement(" ", 0);
685. }
686. **else** printElement(instruction\_list[j].written, 0);
687. cout << endl;
688. }
689. printStationStatus("test", 6);
690. **for** (**int** i=0; i < addReservationStations; i++) {
691. **if** (add\_rs[i].cycles\_required == -999) {
692. printElement("", 8);
693. }
694. **else** {
695. printElement(add\_rs[i].cycles\_required - add\_rs[i].cycle\_count, 8);
696. }
697. printElement("[Add", 0);
698. printElement(add\_rs[i].num, 0);
699. printElement("]", 2);
700. **if** (add\_rs[i].busy == 0) {
701. printElement(" ", 8);
702. printElement(" ", 8);
703. printElement(" ", 8);
704. printElement(" ", 8);
705. }
706. **else** {
707. printElement(add\_rs[i].busy, 8);
708. printElement(add\_rs[i].name, 8);
709. printElement(add\_rs[i].tag\_j, 8);
710. printElement(add\_rs[i].data\_j, 8);
711. printElement(add\_rs[i].tag\_k, 8);
712. printElement(add\_rs[i].data\_k, 8);
713. }
714. cout << endl;
715. }
716. **for** (**int** i=0; i < mulReservationStations; i++) {
717. **if** (mul\_rs[i].cycles\_required == -999) {
718. printElement("", 8);
719. }
720. **else** {
721. printElement(mul\_rs[i].cycles\_required - mul\_rs[i].cycle\_count, 8);
722. }
723. printElement("[Mult", 0);
724. printElement(mul\_rs[i].num-3, 0);
725. printElement("]", 2);
726. **if** (mul\_rs[i].busy == 0) {
727. printElement(" ", 8);
728. printElement(" ", 8);
729. printElement(" ", 8);
730. printElement(" ", 8);
731. }
732. **else** {
733. printElement(mul\_rs[i].busy, 8);
734. printElement(mul\_rs[i].name, 8);
735. printElement(mul\_rs[i].tag\_j, 8);
736. printElement(mul\_rs[i].data\_j, 8);
737. printElement(mul\_rs[i].tag\_k, 8);
738. printElement(mul\_rs[i].data\_k, 8);
739. }
740. cout << endl;
741. }
742. printLoadStatus("test", 6);
743. **for** (**int** i=0; i < loadReservationStations; i++) {
744. printElement("", 8);
745. printElement("[Load", 0);
746. printElement(load\_rs[i].num-5, 0);
747. printElement("]", 2);
748. printElement(load\_rs[i].busy, 10);
749. **if** (load\_rs[i].busy == **true**) {
750. printElement(load\_rs[i].address, 0);
751. }
752. cout << endl;
753. }
754. printStoreStatus("test", 6);
755. **for** (**int** i=0; i < storeReservationStations; i++) {
756. printElement("", 8);
757. printElement("[Stor", 0);
758. printElement(store\_rs[i].num-8, 0);
759. printElement("]", 2);
760. printElement(store\_rs[i].busy, 10);
761. **if** (store\_rs[i].busy == **true**) {
762. printElement(store\_rs[i].address, 0);
763. }
764. cout << endl;
765. }
766. printRegisterStatus("test", 6);
767. printElement(clockCycles+1, 8);
768. **for** (**int** i=0; i < numRegisters; i++) {
769. **if** (data\_registers[i].tag == 0) {
770. printElement(data\_registers[i].data, 8);
771. }
772. **else** {
773. printElement("[", 0);
774. printElement(data\_registers[i].tag, 0);
775. printElement("]", 8);
776. }
777. }
778. cout << endl << endl;
780. // ================== INCREMENT COUNTERS ==================
781. **if** (issueSuccessful) {
782. issuedInstr += 1;
783. issueSuccessful = **false**;
784. }
785. clockCycles += 1;
786. }
788. **return** 0;
789. }

792. // ================== PRINT FUNCTIONS ==================
793. **void** header(**int** n)
794. {
795. cout << "Cycle " << n << endl << endl;
796. }
798. **template**<**typename** T> **void** printElement(T t, **const** **int**& width)
799. {
800. **const** **char** separator    = ' ';
801. cout << left << setw(width) << setfill(separator) << t;
802. }
804. **template**<**typename** T> **void** printInstructionStatus(T t, **const** **int**& width)
805. {
806. cout << "Instruction Status:" << endl;
807. printElement("Instruction", 15);
808. printElement("j", 6);
809. printElement("k", 6);
810. printElement("Issue", 8);
811. printElement("Start", 8);
812. printElement("Completion", 12);
813. printElement("Written", 0);
814. cout << endl;
815. }
817. **template**<**typename** T> **void** printLoadStatus(T t, **const** **int**& width)
818. {
819. cout << endl << "Load Status:" << endl;
820. printElement("", 8);
821. printElement("Name", 8);
822. printElement("Busy", 10);
823. printElement("Address", 0);
824. cout << endl;
825. }
827. **template**<**typename** T> **void** printStoreStatus(T t, **const** **int**& width)
828. {
829. cout << endl << "Store Status:" << endl;
830. printElement("", 8);
831. printElement("Name", 8);
832. printElement("Busy", 10);
833. printElement("Address", 0);
834. cout << endl;
835. }
837. **template**<**typename** T> **void** printStationStatus(T t, **const** **int**& width)
838. {
839. cout << endl << "Reservation Stations:" << endl;
840. printElement("Time", 8);
841. printElement("Name", 8);
842. printElement("Busy", 8);
843. printElement("Op", 8);
844. printElement("Qj", 8);
845. printElement("Vj", 8);
846. printElement("Qk", 8);
847. printElement("Vk", 8);
848. cout << endl;
849. }
851. **template**<**typename** T> **void** printRegisterStatus(T t, **const** **int**& width)
852. {
853. cout << endl << "Register Result Status:" << endl;
854. printElement("Clock", 8);
855. printElement("R0", 8);
856. printElement("R2", 8);
857. printElement("R4", 8);
858. printElement("R6", 8);
859. printElement("R8", 8);
860. printElement("R10", 8);
861. cout << endl;
862. }