بسم الله الرحمن الرحیم

گزارش تمرین 8

درس مبانی سیستم نهفته و بیدرنگ

جواب مسئله1)

محاسبه Cyclomatic complexity

CC = E – N + 2P

E = # Edges

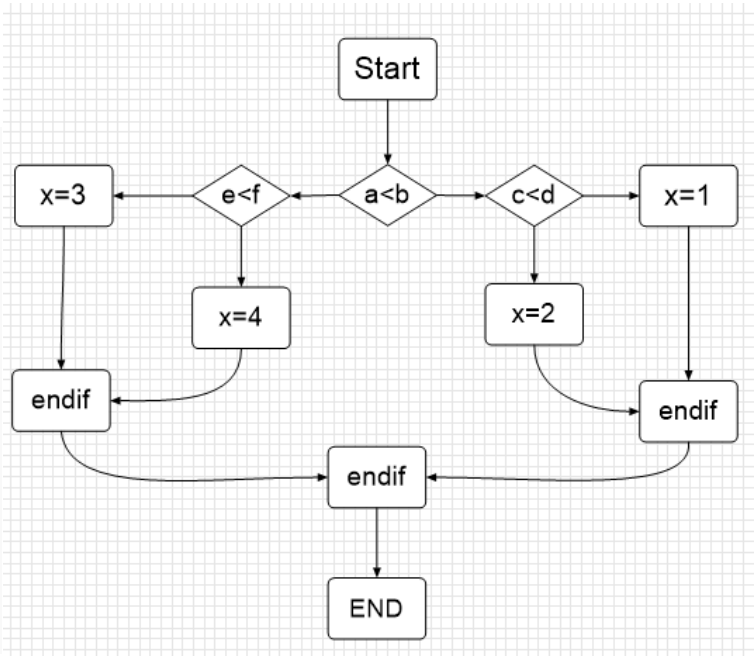
N = # Nodes

P = # functions (components)

Simplified formula CC = # Decisions (if statements) + 1

a)

```
if (a < b) {
    if (c < d)
        x = 1;
    else
        x = 2;
} else {
    if (e < f)
        x = 3;
    else
        x = 4;
}
```



E = 14 , N = 12, P = 1, # Decisions(if) = 3

CC = 14 – 12 + 2 =4 CC = 3 + 1 = 4

b)

```
switch (state) {
    case A:
        if (x = 1) { r = a + b; state = B; }
        else { s = a e b; state = C; }
        break;
    case B:
        s = c + d;
        state = A;
        break;
    case C:
        if (x < 5) { r = a - f; state = D; }
        else if (x == 5) { r = b + d; state = A; }
        else { r = c + e; state = D; }
        break;
    case D:
        r = r + 1;
        state = D;
        break;
}
```

# Decisions = 1 switch(state) + 1 if in case A + 2 if in case C = 4
CC = 4 + 1 = 5

C)

```
for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
        x[i][j] = a[i][j] * c[i];
```

# Decisions = 1 in outer for + 1 in inner for = 2
CC = 2 + 1 = 3

مسئله2) 5-27 مرجع Wolf را در رابطه با آزمون شرط های انشعاب برای کدهای داده شده حل کنید (بخش 5-10 را پیش از حل مسئله مطالعه کنید.)

Q5-27 Use the branch condition testing strategy to determine a set of tests for each of the following statements.

a.
```
if (a < b || ptr1 == NULL) proc1();
else proc2();
```

b.
```
switch (x) {
case 0: proc1(); break;
case 1: proc2(); break;
case 2: proc3(); break;
case 3: proc4(); break;
default; dproc(); break;
}
```

c.
```
if (a <5 && b > 7) proc1();
else if (a < 5) proc2();
else if (b > 7) proc3();
else proc4();
```

جواب مسئله 2)

برای حل کردن این مسئله اول بهتر است در مورد آزمون شرط های انشعاب اشاره داشته باشیم.

آزمون وضعیت انشعاب (Branch Condition Testing) یکی از تکنیک‌های آزمون مبتنی بر کنترل جریان (Control Flow Testing) در مهندسی نرم‌افزار است. این آزمون برای بررسی دقیق تمامی مسیرهای ممکن در کد استفاده می‌شود و هدف آن پوشش دادن تمامی وضعیت‌ها و نتایج منطقی مرتبط با شرایط انشعاب (مانند if, switch-case و غیره) است.

بخش آ)

```
if (a < b || ptr1 == NULL)
    proc1();
 else
    proc2();
```

Test Case	a	b	ptr1	Condition 1 (a < b)	Condition 2 (ptr1 == NULL)	Logical Expression	Action
TC 1	3	5	NULL	TRUE	TRUE	TRUE	proc1()
TC 2	3	5	Not NULL	TRUE	FALSE	TRUE	proc1()
TC 3	7	5	NULL	FALSE	TRUE	TRUE	proc1()
TC 4	7	5	Not NULL	FALSE	FALSE	FALSE	proc2()

بخش ب)

Test Case	x	Action
Test Case 1	0	proc1()
Test Case 2	1	proc2()
Test Case 3	2	proc3()
Test Case 4	3	proc4()
Test Case 5	5	dproc()

```
switch (x) {
    case 0: proc1(); break;
    case 1: proc2(); break;
    case 2: proc3(); break;
    case 3: proc4(); break;
    default: dproc(); break;
}
```

بخش ث)

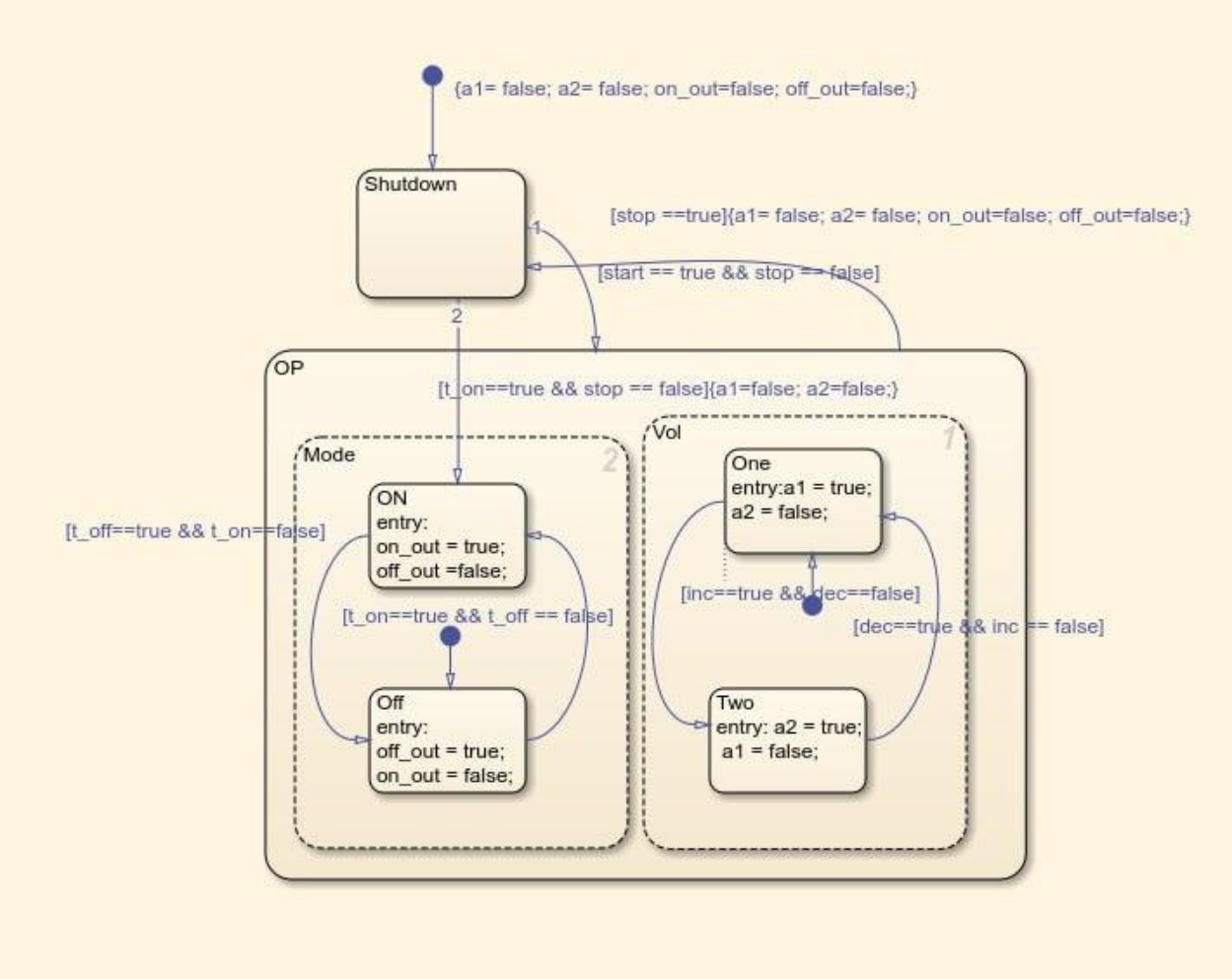
Test Case	a	b	Condition 1 (a < 5)	Condition 2 (b > 7)	Action
Test Case 1	3	8	TRUE	TRUE	proc1()
Test Case 2	3	6	TRUE	FALSE	proc2()
Test Case 3	6	8	FALSE	TRUE	proc3()
Test Case 4	6	6	FALSE	FALSE	proc4()

```
if (a < 5 && b > 7)
    proc1();
else if (a < 5)
    proc2();
else if (b > 7)
    proc3();
else
    proc4();
```

مسئله3)

بخش الف) برای این مدل با فرض آزمون با همان ترتیب ورودی مشخص شده در صورت سوال گزارش های مربوط به coverageرا در Stateflow/Simulink استخراج کرده و در گزارش ذکر کنید. چه درصدی از حالتها و گذارها مورد آزمون قرار گرفته است؟

الف) **ماشین حالت تمرین 4**



Summary

Model Hierarchy/Complexity

	Decision		Execution	
1. classActivity	12	55%	100%	
2. . . . Alarm	11	55%	NA	
3. . . . . SF: Alarm	10	55%	NA	
4. . . . . SF: OP	6	67%	NA	
5. . . . . SF: Mode	3	67%	NA	
6. . . . . SF: Vol	3	67%	NA	
7. . . . Compare To Constant		NA	100%	
8. . . . Compare To Constant1		NA	100%	
9. . . . Compare To Constant2		NA	100%	

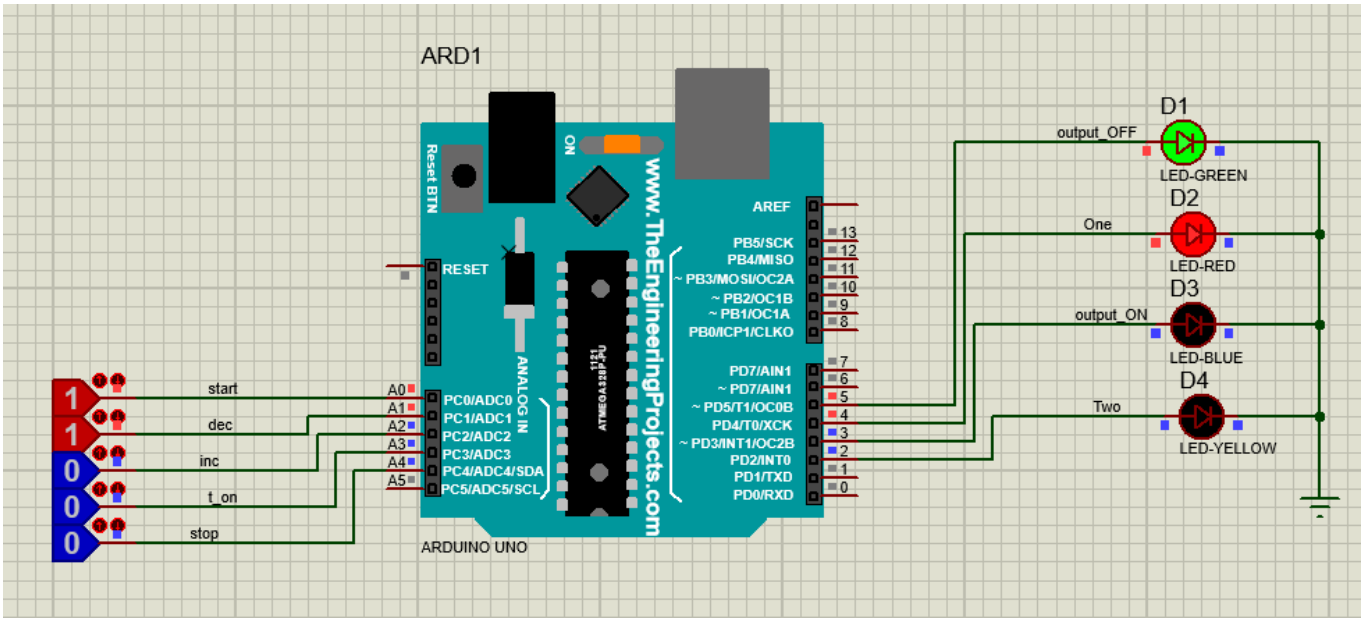
1. **ساختار مدل:(Hierarchy)**
   - مدل به صورت سلسله مراتبی سازمان‌دهی شده است :
     - **classActivity** بخش بالاترین سطح مدل است و شامل زیربخش‌هایی مانند :
       - **Alarm** زیرمجموعه آن
       - **OP** حالات عملیاتی
       - **Mode** حالت‌ها
       - **Vol**حالات ولوم/حجم
2. **متریک‌های پوشش:**
   - **پوشش تصمیم‌گیری:(Decision Coverage)**
     - این متریک درصد تصمیم‌گیری‌های (شاخه‌های منطقی) تست‌شده در طول شبیه‌سازی را نشان می‌دهد.

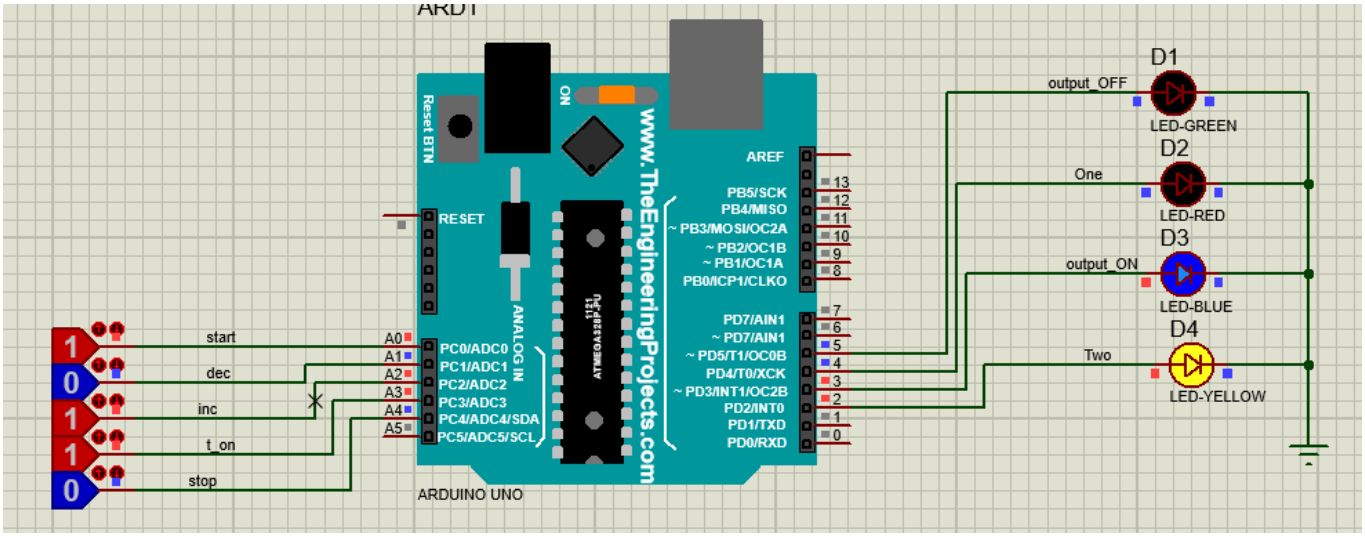
- به عنوان مثال، **classActivity**دارای**55%** **پوشش تصمیم‌گیری** است، یعنی فقط 55% از نقاط تصمیم‌گیری در تست مورد بررسی قرار گرفته‌اند.

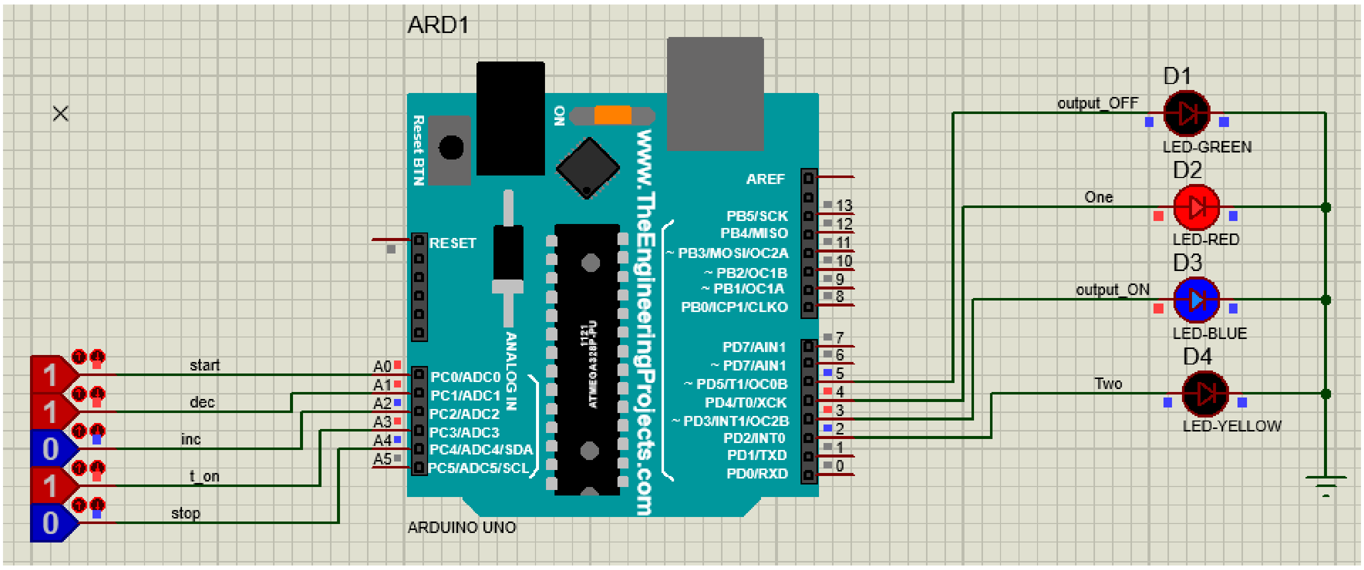
- **پوشش اجرا:(Execution Coverage)**
  - این متریک درصد عناصر مدل (مثل بلوک‌ها، حالات یا انتقال‌ها) که در طول شبیه‌سازی اجرا شده‌اند را نشان می‌دهد.
  - به عنوان مثال، پوشش اجرا برای **classActivity** برابر با **100%**است، یعنی تمام عناصر این بخش اجرا شده‌اند.

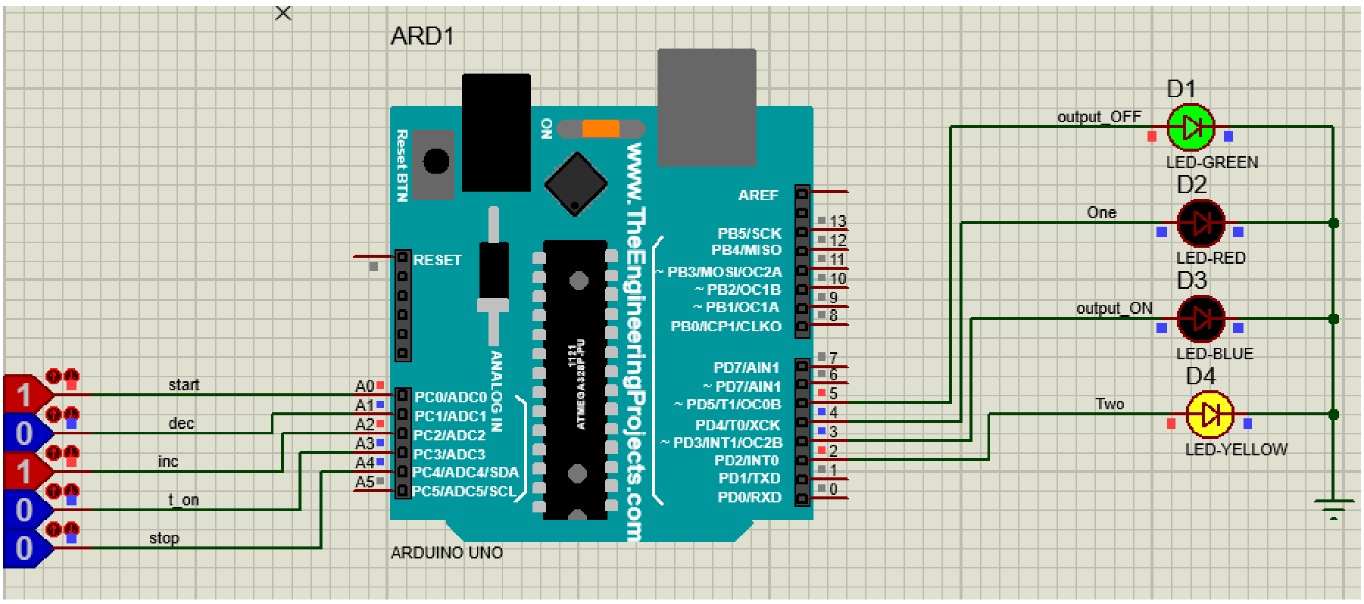
3. **جزئیات بخش‌ها**:
   - **SF: OP**، **SF: Mode** و**SF: Vol** هرکدام دارای **67%پوشش تصمیم‌گیری** هستند، که نشان می‌دهد این بخش‌ها بیشتر از classActivity و Alarm تست شده‌اند.
   - بلوک‌هایی مثل Compare to Constant دارای **100% پوشش اجرا** هستند، به این معنا که این بلوک‌ها به طور کامل تست شده‌اند، هرچند که پوشش تصمیم‌گیری برای آن‌ها قابل اعمال نیست.

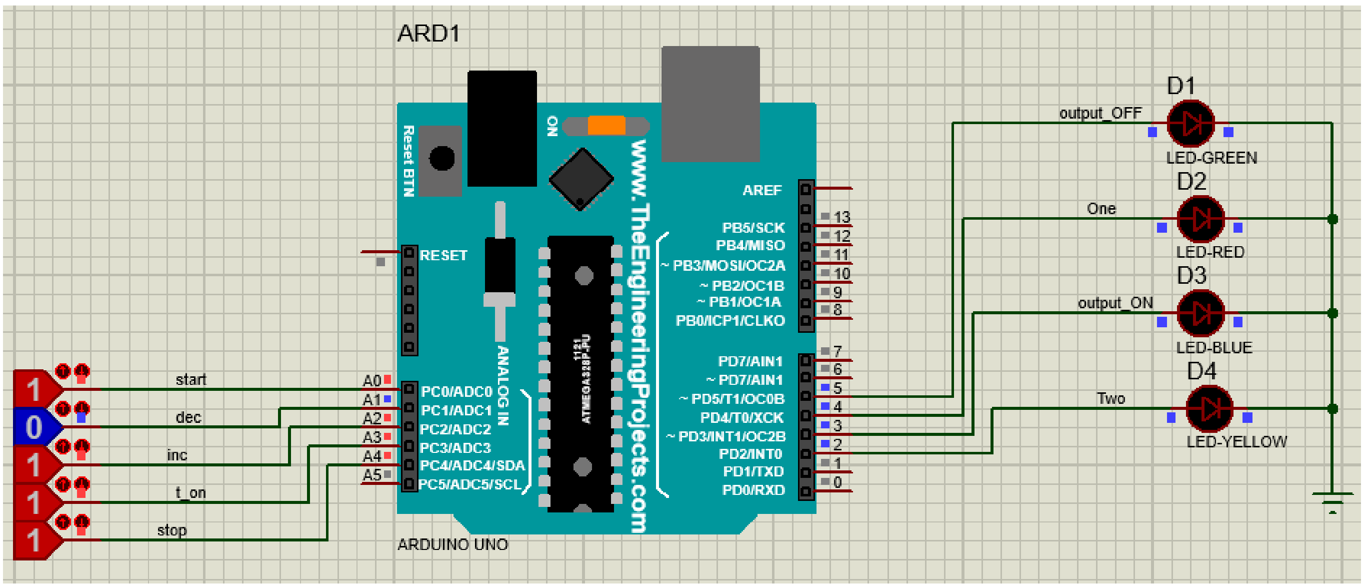
ب) شبیه ساز پروتئوس



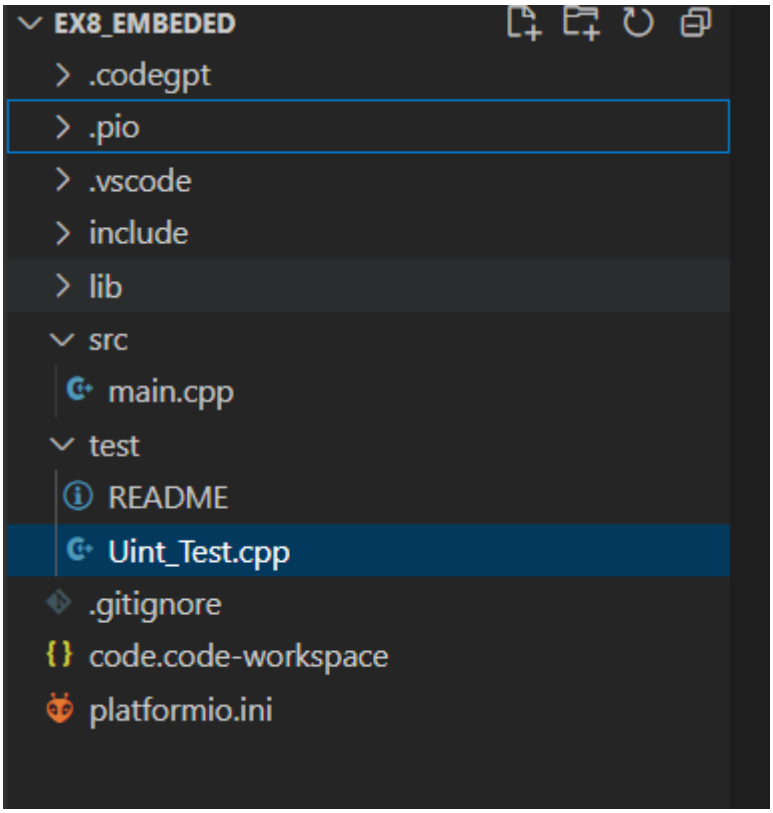



ARD1
ARDUINO UNO
start
dec
inc
t_on
stop
output_OFF
One
output_ON
Two
D1
LED-GREEN
D2
LED-RED
D3
LED-BLUE
D4
LED-YELLOW


ARD1
ARDUINO UNO
start
dec
inc
t_on
stop
output_OFF
One
output_ON
Two
D1
LED-GREEN
D2
LED-RED
D3
LED-BLUE
D4
LED-YELLOW


ARD1
ARDUINO UNO
start
dec
inc
t_on
stop
output_OFF
One
output_ON
Two
D1
LED-GREEN
D2
LED-RED
D3
LED-BLUE
D4
LED-YELLOW

(Test unit



در این فایل همه حالت ها تست شده است که بطور نمونه تست حالت خاموش کردن و اضافه کردن بصورت زیر است:

```
446  void test_off_and_inc() {
447      classActivity::ExtU_classActivity_T inputs;
448      inputs.start_e = 1;
449      inputs.dec = 0;
450      inputs.inc = 1;
451      inputs.t_on = 0;
452      inputs.t_off = 1;
453      inputs.stop = 0;
454
455      classActivity_instance.setExternalInputs(&inputs);
456      classActivity_instance.step();
457
458      const classActivity::ExtY_classActivity_T &outputs = classActivity_instance.getExternalOutputs();
459
460      TEST_ASSERT_EQUAL(false, outputs.a1);
461      TEST_ASSERT_EQUAL(true, outputs.a2);
462      TEST_ASSERT_EQUAL(true, outputs.off_out);
463      TEST_ASSERT_EQUAL(false, outputs.on_out);
464
465  }
```

در این تست, اول ورودی ها را بصورت دستی مقدار دهی می کنیم بعد از فراخوانی تابع ()step خروجی ها را تست میکنیم این تابع را در تابع اصلی با استفاده از RUN_TEST(...) اجرا میکنیم.

تابع اصلی:

```
561  int main(int argc, char** argv) {
562      UNITY_BEGIN(); // Start Unity testing framework
563
564      RUN_TEST(test_stop_on1);
565      RUN_TEST(test_stop_on2);
566      RUN_TEST(test_stop_on3);
567      RUN_TEST(test_stop_on4);
568      RUN_TEST(test_stop_on5);
569
570      RUN_TEST(test_stop_off_and_Start_on);
571      RUN_TEST(test_stop_off_and_T_on);
572      RUN_TEST(test_start_on_inc);
573      RUN_TEST(test_start_on_dec);
574      RUN_TEST(test_start_on_offOn);
575      RUN_TEST(test_start_on_StateOn);
576      RUN_TEST(test_On_and_Inc);
577      RUN_TEST(test_On_and_Dec);
578      RUN_TEST(test_off_and_inc);
579      RUN_TEST(test_off_and_dec);
580      RUN_TEST(test_stopAndStart_off1);
581      RUN_TEST(test_stopAndStart_off2);
582      RUN_TEST(test_stopAndStart_off3);
583
584      UNITY_END();    // End Unity testing framework
585
586      return 0;
587  }
```

خروجی تست ها:

```
PS C:\Users\Fartash\Documents\PlatformIO\Projects\EX8_Embeded> pio test -e native
Verbosity level can be increased via `-v, -vv, or -vvv` option
Collected 1 tests

Processing * in native environment
-----------------------------------------------------------------------------------------------------------------------------------------------
Building...
Testing...
test\Uint_Test.cpp:564: test_stop_on1   [PASSED]
test\Uint_Test.cpp:565: test_stop_on2   [PASSED]
test\Uint_Test.cpp:566: test_stop_on3   [PASSED]
test\Uint_Test.cpp:567: test_stop_on4   [PASSED]
test\Uint_Test.cpp:568: test_stop_on5   [PASSED]
test\Uint_Test.cpp:570: test_stop_off_and_Start_on      [PASSED]
test\Uint_Test.cpp:571: test_stop_off_and_T_on  [PASSED]
test\Uint_Test.cpp:572: test_start_on_inc       [PASSED]
test\Uint_Test.cpp:573: test_start_on_dec       [PASSED]
test\Uint_Test.cpp:574: test_start_on_offOn     [PASSED]
test\Uint_Test.cpp:575: test_start_on_StateOn   [PASSED]
test\Uint_Test.cpp:576: test_On_and_Inc [PASSED]
test\Uint_Test.cpp:577: test_On_and_Dec [PASSED]
test\Uint_Test.cpp:578: test_off_and_inc        [PASSED]
test\Uint_Test.cpp:579: test_off_and_dec        [PASSED]
test\Uint_Test.cpp:580: test_stopAndStart_off1  [PASSED]
test\Uint_Test.cpp:581: test_stopAndStart_off2  [PASSED]
test\Uint_Test.cpp:582: test_stopAndStart_off3  [PASSED]
-------------------------------------------------------- native:* [PASSED] Took 2.13 seconds --------------------------------------------

================================================================= SUMMARY ==================================================================
Environment    Test    Status    Duration
-------------  ------  --------  ------------
native         *       PASSED    00:00:02.129
============================================== 18 test cases: 18 succeeded in 00:00:02.129 ===============================================
PS C:\Users\Fartash\Documents\PlatformIO\Projects\EX8_Embeded>
```

ج) معیار ضریب اسپاگتی(Spaghetti Factor (SF)) معرفی شده در اسلایدهای درس را برای کد تولید شده توسط ابزار به صورت دستی محاسبه کنید

Spaghetti Factor(SF) = SCC + (Globals*5) + (SLOC/20)

- SCC = Strict Cyclomatic Complexity
- Globals = # of read/write global variables referenced
- SLOC = # source lines of code (e.g., C statements)

SCC

- SCC = E – N + 2P
- SCC = #Decisions + 1 #Decisions in code(if, if else, branch , loop) = 11
- SCC = 11 + 1 = 12

Globals

- classActivity_DW (read/write).
- classActivity_U (read).
- classActivity_Y (write).
  Total Globals = 3.

SLOC

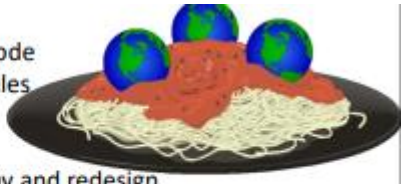
Source Lines of Code (SLOC) are the number of executable lines. We exclude comments, declarations, and blank lines:

- From the provided code, count the actual C statements (e.g., assignments, conditionals).
- Approximation: 100 lines of executable code(SLOC ≅ 100)

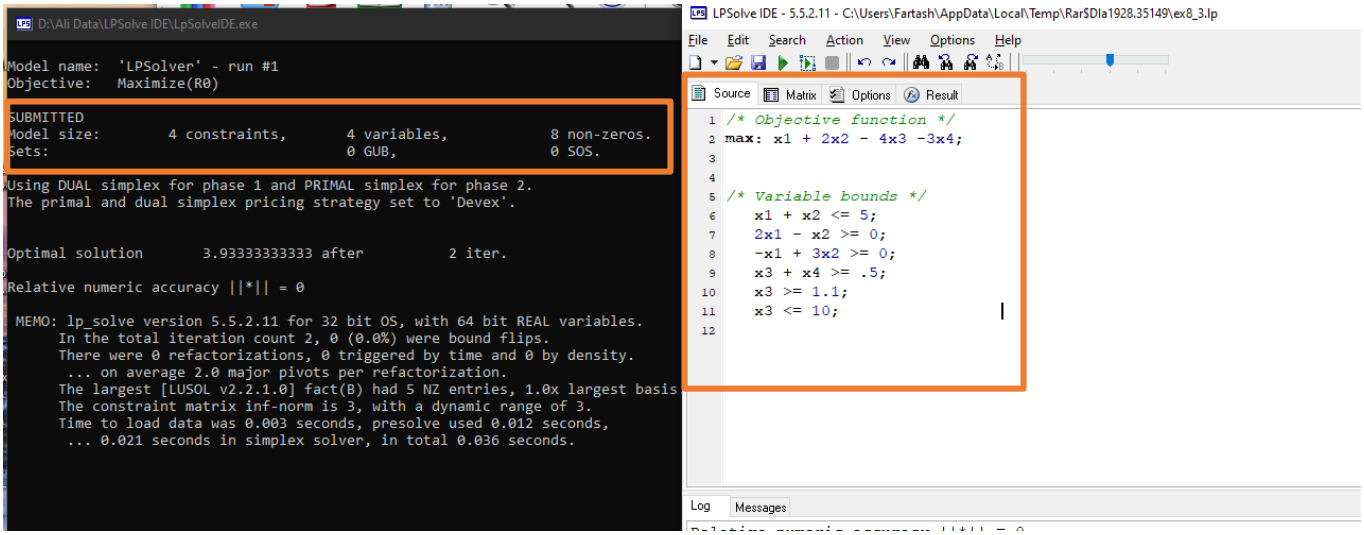
Spaghetti Factor(SF) = 12 + 5* 3 + (100/20) = 32

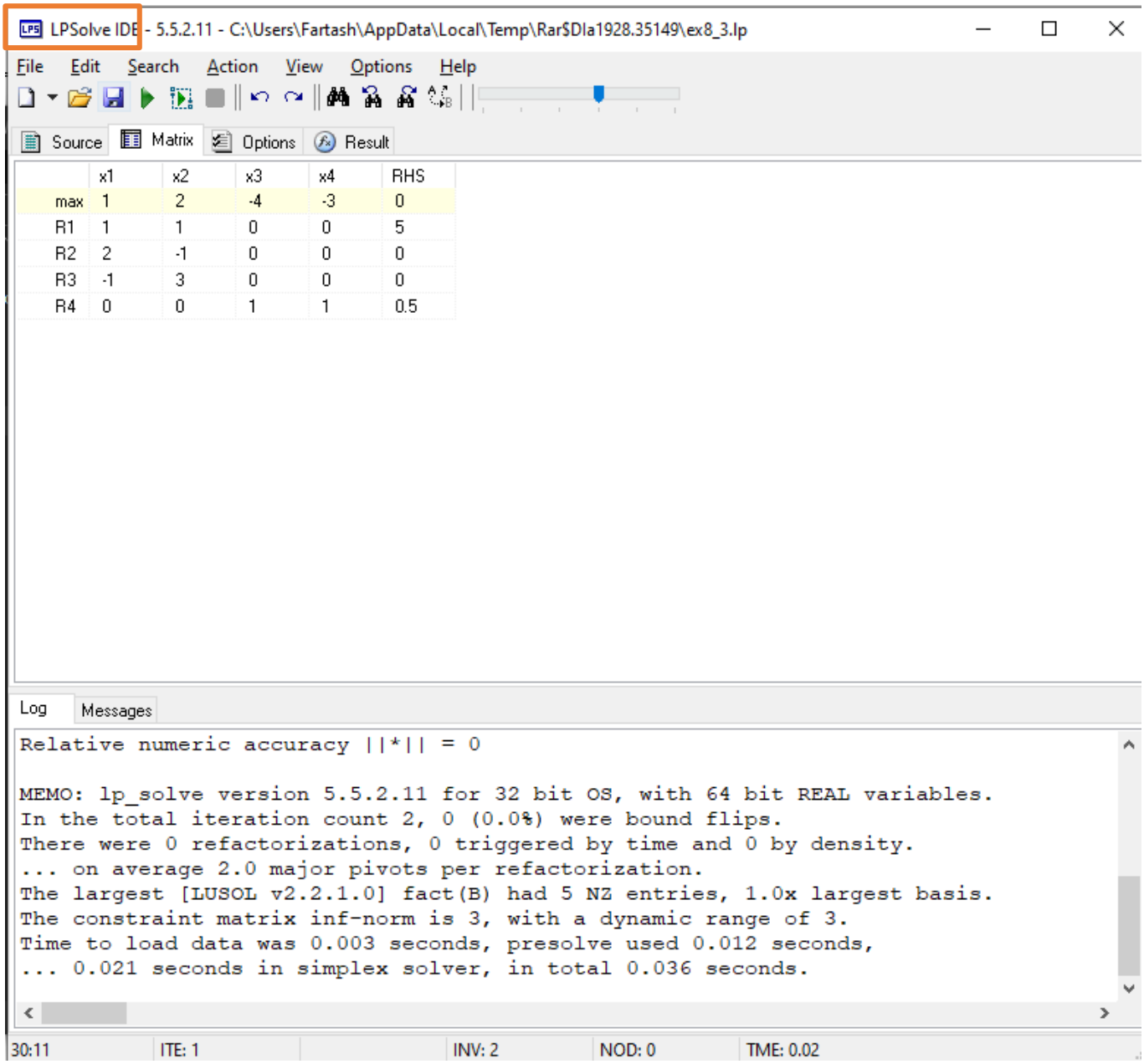
با SF = 32، کد پیچیدگی بالایی را نشان می دهد و می تواند از ساده سازی و ساختار بهتر برای بهبود قابلیت نگهداری بهره مند شود.

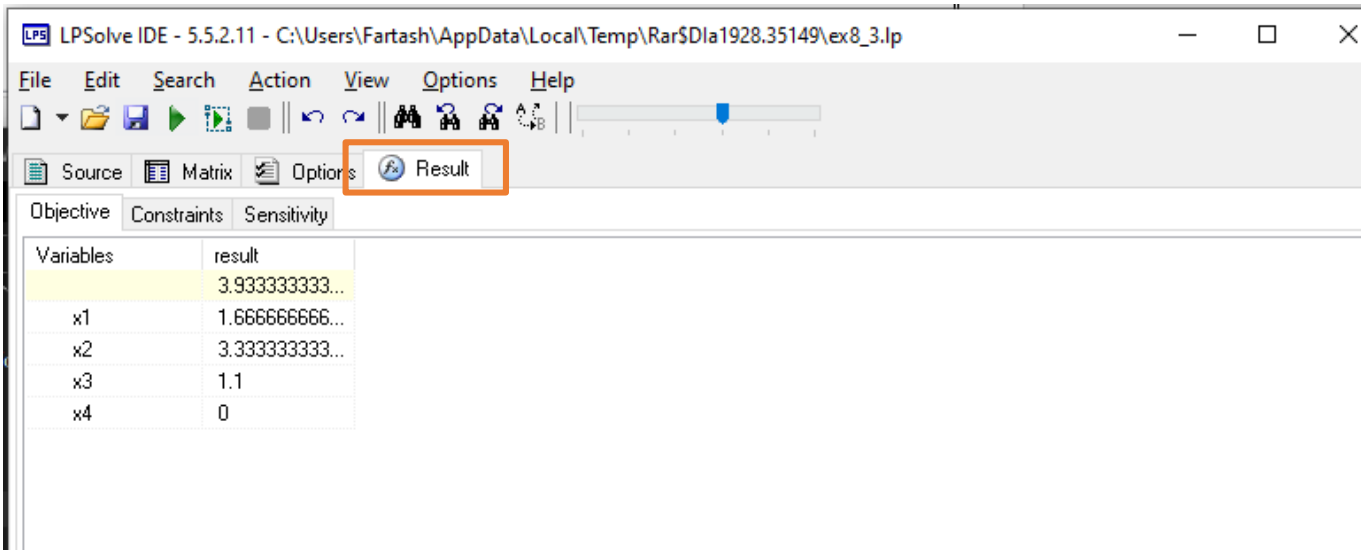
- Scoring:
  - 5-10 - This is the sweet spot for most code
  - 15 - Don't go above this for most modules
  - 20 - Look closely; possibly refactor
  - 30 - Refactor the design
  - 50 - Untestable; throw the module away and redesign
  - 75 - Unmaintainable; throw the module and its design away; start over
  - 100 - Nightmare; throw it out and re-architect



بخش امتیازی قسمت الف)







بخش دوم:

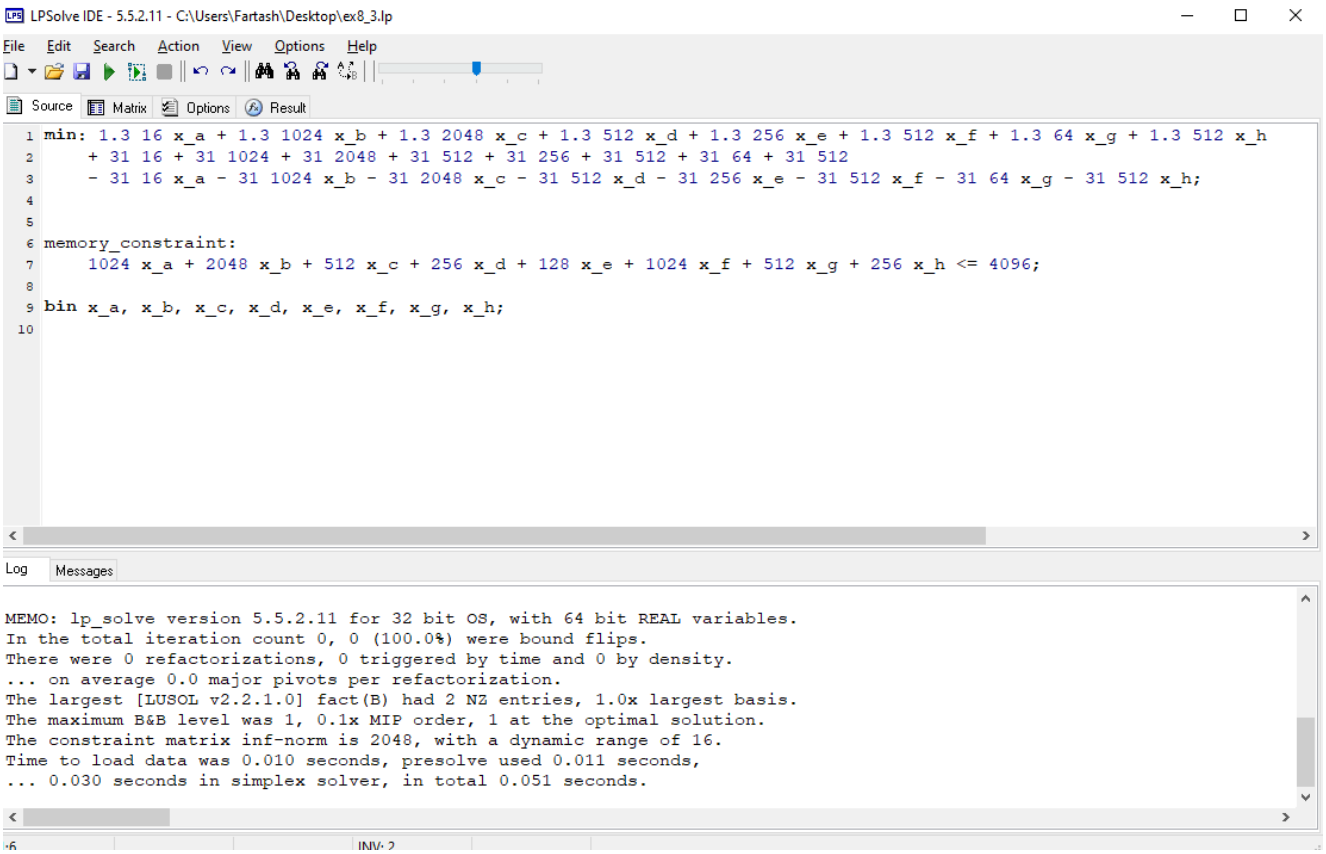
Table 7.2 SPM mapping: **left**, accesses to variables; **right**, memory characteristics

Variable	Size [bytes]	Number of accesses
a	1024	16
b	2048	1024
c	512	2048
d	256	512
e	128	256
f	1024	512
g	512	64
h	256	512

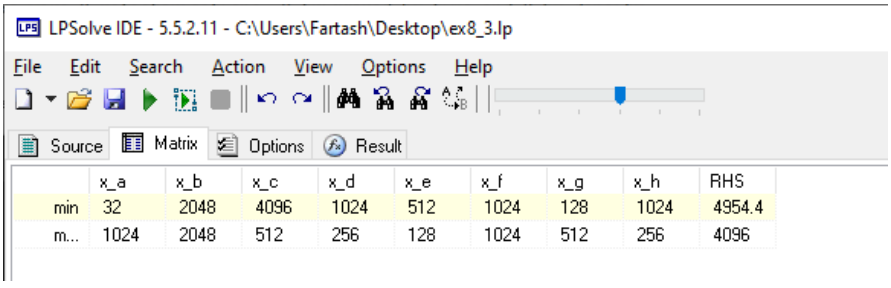
Memory	Size [bytes]	Energy per access
Scratchpad	4096 (4 k)	1.3 nJ
Main memory	262,144 (256 k)	31 nJ

7.6 Suppose that your computer is equipped with a main memory and a scratchpad memory. Sizes and the required energy per access are shown in Table 7.2 (right). Characteristics of accesses to variables are as indicated in Table 7.2 (left).

Which of those variables should be allocated to the scratchpad memory, provided that we use a static, non-overlaying allocation of variables? Use the integer linear problem (ILP) model to select the variables. Your result should include the ILP model as well as the results. You may use the *lp_solve* program [17] to solve your ILP problem.




```
Model name:  'LPSolver' - run #1
Objective:   Minimize(R0)

SUBMITTED
Model size:        1 constraints,       8 variables,           8 non-zeros.
Sets:                                   0 GUB,                 0 SOS.

Using DUAL simplex for phase 1 and PRIMAL simplex for phase 2.
The primal and dual simplex pricing strategy set to 'Devex'.


Relaxed solution             4954.4 after          0 iter is B&B base.

Feasible solution            4954.4 after          0 iter,         0 nodes (gap 0.0%)

Optimal solution             4954.4 after          0 iter,         0 nodes (gap 0.0%).

Relative numeric accuracy ||*|| = 0

 MEMO: lp_solve version 5.5.2.11 for 32 bit OS, with 64 bit REAL variables.
      In the total iteration count 0, 0 (100.0%) were bound flips.
      There were 0 refactorizations, 0 triggered by time and 0 by density.
       ... on average 0.0 major pivots per refactorization.
      The largest [LUSOL v2.2.1.0] fact(B) had 2 NZ entries, 1.0x largest basis.
      The maximum B&B level was 1, 0.1x MIP order, 1 at the optimal solution.
      The constraint matrix inf-norm is 2048, with a dynamic range of 16.
      Time to load data was 0.010 seconds, presolve used 0.011 seconds,
       ... 0.030 seconds in simplex solver, in total 0.051 seconds.
```



	x_a	x_b	x_c	x_d	x_e	x_f	x_g	x_h	RHS
min	32	2048	4096	1024	512	1024	128	1024	4954.4
m...	1024	2048	512	256	128	1024	512	256	4096