

پژوهش نهایی درس معماری کامپیوتر

استاد: دکتر چشمی خانی

## پیاده سازی یک سیستم مدیریت مالی به زبان اسambilی RISC-V و تست عملکرد آن در شبیه ساز CPU Lator

نویسنده

مهديار صلواتي

کد دانشجویی: ۴۰۲۲۴۳۰۸۰

[ma.salavati@mail.sbu.ac.ir](mailto:ma.salavati@mail.sbu.ac.ir)

### چکیده

در این پژوهه قصد داریم یک سیستم مدیریت قرض دادن پول طراحی کنیم که به زبان اسambilی RISC-V نوشته شده است. در ادامه آن را در شبیه ساز CPU Lator تست کرده و نتایج را ارائه خواهیم کرد.

### ۱۰. روند کلی

برای سادگی و ایجاد خط فکری منظم، ابتدا سعی می کنیم پژوهه را در زبان C++ انجام داده و با مپ کردن هر عبارت سطح بالا به معادل آن در اسambilی کد نهایی دست یابیم. در پیاده سازی کد سطح بالا باید با رویکردی پیش رویم تا پیاده سازی آن در سطح پایین تر به دشواری نیانجامد.

به طور کلی دو رویکرد اساسی در پیاده سازی کد C++ می توان پیش گرفت. اول آن که با دانستن  $n$  دستور حداکثر  $n$  دستور از نوع ۱ می توان داشت. حال با توجه به اینکه ممکن است یک طرف از تمامی پرداخت ها مربوط به یک شخص خاص باشد پس برای هر فرد باید  $n$  فضا داشته باشیم تا مشکلی پیش نیاید. در نتیجه چنین رویکردی نیاز به فضای حافظه ای از پیچیدگی  $\mathcal{O}(n^2)$  داریم.

اما رویکرد بهینه این است که از تخصیص پویای حافظه Dynamic Memory Allocation استفاده کرد. به این صورت که دیگر نیاز نداریم تا همه حافظه های مورد نیاز را pre-allocate کنیم. پس در بدترین حالت این رویکرد به فضای حافظه ای از پیچیدگی  $\mathcal{O}(n)$  (که در آن  $n$  تعداد دستورات است) نیازمندیم. این موضوع به صورت ریاضی اثبات می شود:

Let:

- $i$  = number of people
- $j$  = number of total transactions
- Worst-case:  
Set  $i = 2n$  and  $j = 2n$ .

$$\bullet \ Cost = \underbrace{\sum_{k=1}^i 1}_{\text{people}} + \underbrace{\sum_{k=1}^j 1}_{\text{transactions}} = 2n + 2n = 4n$$

تصویر زیر در یک مثال تفاوت این دو رویکرد را نشان می‌دهد:

`for n = 3:`

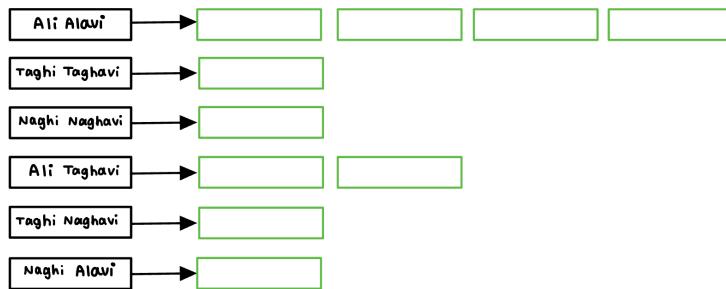
---

### 1) NAIVE APPROACH




---

### 2) DYNAMIC MEMORY ALLOCATION



# فهرست مطالب

|    |                                   |       |
|----|-----------------------------------|-------|
| ۱  | روند کلی                          | ۱.۰   |
| ۶  | C++ کد                            | ۱     |
| ۶  | ساختار node                       | ۱.۱   |
| ۶  | ساختار nameNode                   | ۲.۱   |
| ۷  | تابع main                         | ۳.۱   |
| ۱۱ | تابع یکم                          | ۴.۱   |
| ۱۲ | تابع دوم                          | ۵.۱   |
| ۱۳ | تابع سوم                          | ۶.۱   |
| ۱۴ | تابع چهارم                        | ۷.۱   |
| ۱۴ | تابع پنجم                         | ۸.۱   |
| ۱۵ | تابع ششم                          | ۹.۱   |
| ۱۶ | RISC-V کد اsemblی                 | ۲     |
| ۱۶ | parse float                       | ۱.۲   |
| ۱۸ | format float                      | ۲.۲   |
| ۲۰ | strcmp                            | ۳.۲   |
| ۲۰ | cmd1                              | ۴.۲   |
| ۲۰ | تبدیل مقدار عددی و مقداردهی اولیه | ۱.۴.۲ |
| ۲۰ | جستجوی فرد اول در لیست            | ۲.۴.۲ |
| ۲۱ | ایجاد گره جدید برای فرد اول       | ۳.۴.۲ |
| ۲۲ | بررسی و ایجاد حساب برای فرد اول   | ۴.۴.۲ |

|    |  |          |
|----|--|----------|
| ۲۲ | جستجوی فرد دوم در لیست                     | ۵.۴.۲    |
| ۲۲ | ایجاد گره جدید برای فرد دوم                | ۶.۴.۲    |
| ۲۳ | بررسی و ایجاد حساب برای فرد دوم            | ۷.۴.۲    |
| ۲۴ | بهروزرسانی موجودی حسابها                   | ۸.۴.۲    |
| ۲۴ | بررسی و بهروزرسانی لیست تراکنش‌های فرد اول | ۹.۴.۲    |
| ۲۵ | بررسی و بهروزرسانی لیست تراکنش‌های فرد دوم | ۱۰.۴.۲   |
| ۲۶ |  | cmd۲ ۵.۲ |
| ۲۶ | مقداردهی اولیه و بررسی لیست خالی           | ۱.۵.۲    |
| ۲۶ | یافتن بیشترین موجودی                       | ۲.۵.۲    |
| ۲۷ | بررسی وجود موجودی مثبت                     | ۳.۵.۲    |
| ۲۷ | یافتن فرد با بیشترین موجودی                | ۴.۵.۲    |
| ۲۸ | مقایسه نام‌ها برای ترتیب الفبایی           | ۵.۵.۲    |
| ۲۸ | ذخیره نام فرد انتخاب شده                   | ۶.۵.۲    |
| ۲۸ | چاپ نتیجه                                  | ۷.۵.۲    |
| ۲۹ |  | cmd۳ ۶.۲ |
| ۳۰ |  | cmd۴ ۷.۲ |
| ۳۰ | مقداردهی اولیه و جستجوی فرد مورد نظر       | ۱.۷.۲    |
| ۳۱ | بررسی وجود حساب برای فرد مورد نظر          | ۲.۷.۲    |
| ۳۱ | شمارش بدھکاران                             | ۳.۷.۲    |
| ۳۲ | چاپ نتیجه                                  | ۴.۷.۲    |
| ۳۲ | حالت‌های خاص و خطأ                         | ۵.۷.۲    |
| ۳۲ |  | cmd۵ ۸.۲ |
| ۳۴ |  | cmd۶ ۹.۲ |
| ۳۴ | جستجوی فرد اول در لیست                     | ۱.۹.۲    |
| ۳۴ | بررسی وجود حساب برای فرد اول               | ۲.۹.۲    |
| ۳۴ | جستجوی تراکنش با فرد دوم                   | ۳.۹.۲    |
| ۳۵ | پردازش و نمایش نتیجه                       | ۴.۹.۲    |
| ۳۵ | حالت‌های خاص و خطأ                         | ۵.۹.۲    |

۳ تست عملکرد

۳۷

# فصل ۱

## C++ کد

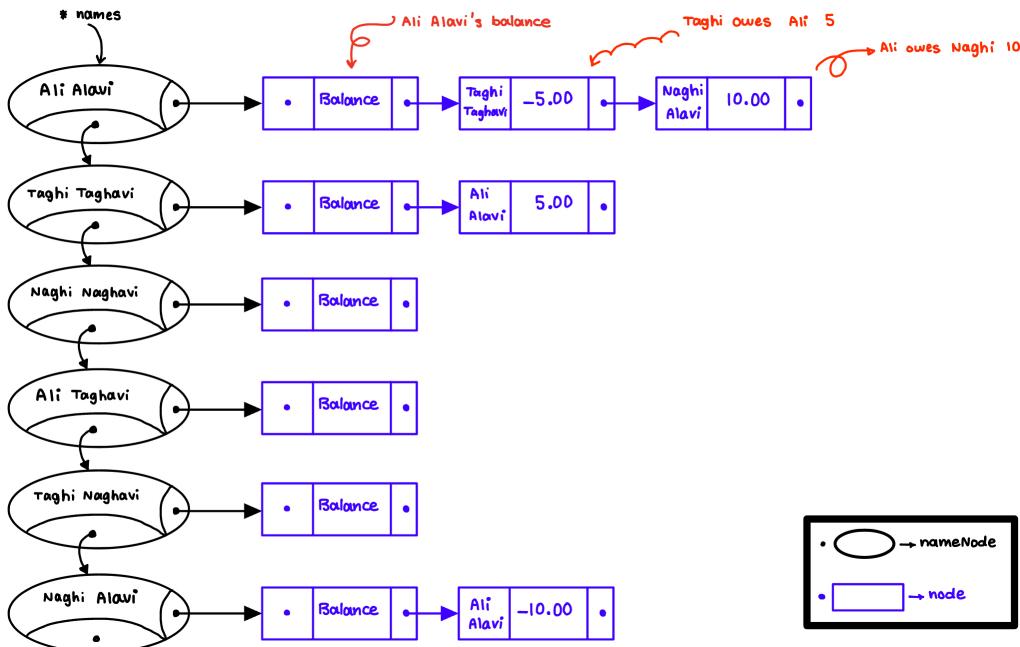
### ۱.۱ ساختار node

```
۱ struct node {  
۲     string name;  
۳     double amount;  
۴     node *next;  
۵ };
```

### ۲.۱ ساختار nameNode

```
۱ struct nameNode {  
۲     string name;  
۳     node *node;  
۴     nameNode *next;  
۵ };
```

از دو ساختار معرفی شده برای نگه داشتن دو لیست پیوندی استفاده می‌کنیم. شکل زیر ساختار کلی این دو لیست پیوندی را در کنار هم نشان می‌دهد:



تابع main ۳۰۱

با توجه به اینکه در هر خط از ورودی کد دستور به همراه تمامی آرگومان هایش قرار گرفته اند، باید ابتدا ورودی را parse کنیم:

```
1 string line = "";
2
3     for (int i = 0; i < n; i++) {
4
5         getline(cin, line);
6
7         int saveIndex1 = 0;
8
9         int saveIndex2 = 0;
10
11        int x;
12
13        string name1 = "";
14
15        string name2 = "";
16
17        string amountS = "";
18
19        double amount;
20
21        int number0fSpaces = 0;
22
23
24        for (int j = 0; j < line.size(); j++) {
25
26            if (line[j] == ' ')
27
28                number0fSpaces++;
29
30            if (number0fSpaces == 1) {
31
32                string xS = "";
33
34                for (int k = 0; k < j; k++) {
35
36                    xS += line[k];
37
38                }
39
40                cout << xS;
41
42            }
43
44        }
45
46    }
47
48}
```

```

    }
    x = stoi(xS);
    saveIndex1 = j;
}
if (numberOfSpaces == 2) {
    for (int k = saveIndex1 + 1; k < j; k++) {
        name1 += line[k];
    }
    saveIndex2 = j;
}
if (numberOfSpaces == 3) {
    for (int k = saveIndex2 + 1; k < j; k++) {
        name2 += line[k];
    }
    saveIndex3 = j;
}
}
}

if (numberOfSpaces == 0) {
    string xS = "";
    for (int k = 0; k < line.size(); k++) {
        xS += line[k];
    }
    x = stoi(xS);
}

if (numberOfSpaces == 1) {
    for (int k = saveIndex1 + 1; k < line.size(); k++) {
        name1 += line[k];
    }
}
if (numberOfSpaces == 2) {
    for (int k = saveIndex2 + 1; k < line.size(); k++) {
        name2 += line[k];
    }
}
if (numberOfSpaces == 3) {
    for (int k = saveIndex3 + 1; k < line.size(); k++) {
        amountsS += line[k];
    }
}

```

```

61     }
62     amount = stod(amountS);
63 }
```

در ادامه در یک سویچ کیس، برای دستورات مختلف، اسمی داده شده را در میان گره ها جستجو می کنیم و در صورت عدم موفقیت، یک گره جدید برای آن نام می سازیم:

```

1 switch (x) {
2
3     case 1:
4     {
5         bool found1 = false;
6         for (nameNode *ptr = names; ptr != nullptr; ptr = ptr->
7             next) {
8             if (ptr->name == name1) {
9                 n1_node = ptr;
10                found1 = true;
11                break;
12            }
13        }
14        if (!found1) {
15            nameNode *newNode = new nameNode();
16            newNode->name = name1;
17            newNode->node = new node{name1, 0.0, nullptr};
18            newNode->next = names;
19            names = newNode;
20            n1_node = newNode;
21        }
22    }
23
24    bool found2 = false;
25    for (nameNode *ptr = names; ptr != nullptr; ptr = ptr
26          ->next) {
27        if (ptr->name == name2) {
28            n2_node = ptr;
29            found2 = true;
30            break;
31        }
32    }
33    if (!found2) {
34        nameNode *newNode = new nameNode();
35        newNode->name = name2;
```

```

۳۳             newNode->node = new node{name2, 0.0, nullptr};
۳۴             newNode->next = names;
۳۵             names = newNode;
۳۶             n2_node = newNode;
۳۷         }
۳۸     }
۳۹     fun1(name1, name2, amount, n1_node, n2_node);
۴۰     break;
۴۱ case 2:
۴۲     fun2(names);
۴۳     break;
۴۴ case 3:
۴۵     fun3(names);
۴۶     break;
۴۷ case 4:
۴۸     for (nameNode *ptr = names; ptr != nullptr; ptr = ptr->
۴۹         next) {
۵۰         if (ptr->name == name1) {
۵۱             n1_node = ptr;
۵۲             break;
۵۳         }
۵۴     }
۵۵     fun4(n1_node);
۵۶     break;
۵۷ case 5:
۵۸     for (nameNode *ptr = names; ptr != nullptr; ptr = ptr->
۵۹         next) {
۶۰         if (ptr->name == name1) {
۶۱             n1_node = ptr;
۶۲             break;
۶۳         }
۶۴     }
۶۵     fun5(n1_node);
۶۶     break;
۶۷ case 6:
۶۸     for (nameNode *ptr = names; ptr != nullptr; ptr = ptr->
۶۹         next) {
۷۰         if (ptr->name == name1) {
۷۱             n1_node = ptr;
۷۲             break;

```

```
    }
}
fun6(name2, n1_node);
break;
}
}
return 0;
```

٤٠١ تابع یکم

در این تابع دو گره از نوع `nameNode` داریم که باید در ابتدا `balance` های هر یک را آپدیت کرده و سپس وارد لیست `node` های آن شده و اسم طرفی که با اون مبادله مالی داشتیم را نیز آپدیت کنیم تا بدانیم هر کس چقدر به دیگری بدهکار یا از او بللکا است:

```
1 void fun1(const string &name1, const string &name2, double amount,
2           nameNode *n1_node, nameNode *n2_node) {
3
4     n1_node->node->amount -= amount;
5
6     n2_node->node->amount += amount;
7
8
9     node *p1 = n1_node->node->next;
10
11    bool found1 = false;
12
13    while (p1 != nullptr) {
14
15        if (p1->name == name2) {
16
17            p1->amount -= amount;
18
19            found1 = true;
20
21            break;
22
23        }
24
25        p1 = p1->next;
26
27    }
28
29    if (!found1) {
30
31        node *nn1 = new node();
32
33        nn1->name = name2;
34
35        nn1->amount = -amount;
36
37        nn1->next = n1_node->node->next;
38
39        n1_node->node->next = nn1;
40
41    }
42
43
44    node *p2 = n2_node->node->next;
45
46    bool found2 = false;
47
48    while (p2 != nullptr) {
49
50        if (p2->name == name1) {
```

```

۲۷     p2->amount += amount;
۲۸     found2 = true;
۲۹     break;
۳۰   }
۳۱   p2 = p2->next;
۳۲ }
۳۳ if (!found2) {
۳۴   node *nn2 = new node();
۳۵   nn2->name = name1;
۳۶   nn2->amount = amount;
۳۷   nn2->next = n2_node->node->next;
۳۸   n2_node->node->next = nn2;
۳۹ }
۴۰ }
```

## ۵.۱ تابع دوم

در این تابع باید در میان کسانی که بیشترین موجودی را دارند، فردی را که از نظر ترتیب لغت نامه‌ای کوچک‌تر است را خروجی دهیم. اگر چنین فردی با موجودی مثبت وجود نداشت ۱- چاپ می‌کنیم.

```

۱ void fun2(nameNode *names) {
۲   if (names == nullptr) {
۳     cout << -1 << endl;
۴     return;
۵   }
۶
۷   double max_worth = names->node->amount;
۸
۹   for (nameNode *ptr = names; ptr != nullptr; ptr = ptr->next) {
۱۰     if (ptr->node->amount > max_worth) {
۱۱       max_worth = ptr->node->amount;
۱۲     }
۱۳   }
۱۴
۱۵   if (max_worth <= 1e-9) {
۱۶     cout << -1 << endl;
۱۷     return;
۱۸   }
۱۹
۲۰   string min_name = "";
۲۱   for (nameNode *ptr = names; ptr != nullptr; ptr = ptr->next) {
```

```

۲۲     if (abs(ptr->node->amount - max_worth) < 1e-9) {
۲۳         if (min_name == "" || ptr->name < min_name) {
۲۴             min_name = ptr->name;
۲۵         }
۲۶     }
۲۷ }
۲۸ cout << min_name << endl;
۲۹ }
```

## ۶.۱ تابع سوم

درست مانند تابع دوم است با این تفاوت که باید در میان افراد با کمترین موجودی جستجو کنیم.

```

۱ void fun3(nameNode *names) {
۲     if (names == nullptr) {
۳         cout << -1 << endl;
۴         return;
۵     }
۶
۷     double min_worth = names->node->amount;
۸
۹     for (nameNode *ptr = names; ptr != nullptr; ptr = ptr->next) {
۱۰         if (ptr->node->amount < min_worth) {
۱۱             min_worth = ptr->node->amount;
۱۲         }
۱۳     }
۱۴
۱۵     if (min_worth >= -1e-9) {
۱۶         cout << -1 << endl;
۱۷         return;
۱۸     }
۱۹
۲۰     string min_name = "";
۲۱     for (nameNode *ptr = names; ptr != nullptr; ptr = ptr->next) {
۲۲         if (abs(ptr->node->amount - min_worth) < 1e-9) {
۲۳             if (min_name == "" || ptr->name < min_name) {
۲۴                 min_name = ptr->name;
۲۵             }
۲۶         }
۲۷     }
۲۸     cout << min_name << endl;
```

```
۲۹ }
```

## ۷.۱ تابع چهارم

در این تابع یک نام داریم و باید تعداد افرادی که به این نام پول داده اند را چاپ کنیم. در نتیجه کافی است لیست node آن نام را یک دور پیمایش کرده و با دیدن هر amount مثبتی شمارش داشته باشیم:

```
۱ void fun4(nameNode *person_node) {
 ۲   if (person_node == nullptr) {
 ۳     cout << 0 << endl;
 ۴     return;
 ۵   }
 ۶   int count = 0;
 ۷   node *ptr = person_node->node->next;
 ۸   while (ptr != nullptr) {
 ۹     if (ptr->amount > 1e-9) {
 ۱۰       count++;
 ۱۱     }
 ۱۲     ptr = ptr->next;
 ۱۳   }
 ۱۴   cout << count << endl;
 ۱۵ }
```

## ۸.۱ تابع پنجم

درست مانند تابع چهارم عمل می کند با این تفاوت که این بار باید تعداد amount های منفی را بشماریم:

```
۱ void fun5(nameNode *person_node) {
 ۲   if (person_node == nullptr) {
 ۳     cout << 0 << endl;
 ۴     return;
 ۵   }
 ۶   int count = 0;
 ۷   node *ptr = person_node->node->next;
 ۸   while (ptr != nullptr) {
 ۹     if (ptr->amount < -1e-9) {
 ۱۰       count++;
 ۱۱     }
 ۱۲     ptr = ptr->next;
 ۱۳   }
 ۱۴   cout << count << endl;
```

۱۵ }

## ۹.۱ تابع ششم

در این تابع وارد لیست node های نام اول شده و به دنبال node ای با نام دوم می‌گردیم. هرگاه یافت شد، amount مربوط به آن گره خروجی ما است:

```

1 void fun6(const string &name2, nameNode *n1_node) {
2     if (n1_node == nullptr) {
3         cout << fixed << setprecision(2) << "0.00" << endl;
4         return;
5     }
6     double amount_owed = 0.0;
7     node *ptr = n1_node->node->next;
8     while (ptr != nullptr) {
9         if (ptr->name == name2) {
10             amount_owed = ptr->amount;
11             break;
12         }
13         ptr = ptr->next;
14     }
15     cout << fixed << setprecision(2) << amount_owed << endl;
16 }
```

## ۲ فصل

# RISC-V کد اسambilی

### parse float ۱.۲

در ابتدا باید با یکی از محدودیت های شبیه ساز Lator CPU روبرو شویم. عملیات های float print و float read به وسیله Terminal ممکن نیست و دارای Bug است. در نتیجه برای حل این مشکل با رسیدن به اعداد، آن را به صورت رشته می خوانیم و به float تبدیل می کنیم و همچنین در زمان پرینت کردن نیز float را به string برمی گردانیم و سپس خروجی را چاپ می کنیم. کد زیر string را به float تبدیل می کند. این گونه که ابتدا علامت آن را تشخیص داده و سپس از چپ شروع کرده و کاراکتر رشته را می خواند و ۴۸ واحد ASCII را از کاراکتر اعداد کم می کنیم تا به رقم برسیم و قبل از رفتن به رقم بعدی رقم قبلی را در  $10$  ضرب می کنیم چرا که در مبنای  $10$  هستیم. همچنین با رسیدن به  $0$ . عمل مشابهی را انجام می دهیم اما با خواندن هر رقم یک متغیر را در  $10$  ضرب می کنیم. چرا که در انتها باید مقدار عددی اعشار را تقسیم بر توانی از  $10$  کرده و با قسمت صحیح جمع کنیم:

```
1 parse_float:
2     addi sp, sp, -12
3     sw    ra, 8(sp)
4     sw    s2, 4(sp)
5     sw    s3, 0(sp)
6     mv    t0, a0
7     li    s2, 0
8     lb    t1, 0(t0)
9     li    t2, 1
10    bne   t1, t2, start_parse
11    li    s2, 1
12    addi t0, t0, 1
13 start_parse:
14    li    t1, 0
15    li    t2, 0
```

```
15      li    t3, 1
16      li    t4, 0
17      li    s3, 10
18
19  parse_loop:
20      lb    t5, 0(t0)
21      beq   t5, x0, parse_done
22      li    t6, 10
23      beq   t5, t6, parse_done
24      li    t6, '\u'
25      beq   t5, t6, parse_done
26      li    t6, '.'
27      beq   t5, t6, is_decimal
28      addi  t5, t5, -'0'
29      bnez  t4, parse_frac
30
31  parse_int_part:
32      mul   t1, t1, s3
33      add   t1, t1, t5
34      j     parse_next
35
36  parse_frac:
37      mul   t2, t2, s3
38      add   t2, t2, t5
39      mul   t3, t3, s3
40      j     parse_next
41
42  is_decimal:
43      li    t4, 1
44
45  parse_next:
46      addi  t0, t0, 1
47      j     parse_loop
48
49  parse_done:
50      fcvt.s.w fa1, t1
51      fcvt.s.w fa2, t2
52      fcvt.s.w fa3, t3
53      fdiv.s fa2, fa2, fa3
54      fadd.s fa0, fa1, fa2
55      beq   s2, x0, parse_ret
56      fneg.s fa0, fa0
57
58  parse_ret:
59      lw    s3, 0(sp)
60      lw    s2, 4(sp)
61      lw    ra, 8(sp)
```

```

56      addi sp, sp, 12
57      ret

```

## format float ۲.۲

برای تبدیل float به string نیز از چپ شروع کرده و با یک تقسیم چپ ترین رقم را گرفته و با باقی مانده گرفتن بر توان های ۱۰ باقی عدد را ذخیره می کنیم. برای بخش اعشاری هم به این گونه عمل می کنیم اما از آنجا که فقط ۲ رقم اعشار نیاز داریم پس کافی است تا توان دوم ۱۰ این عمل را ادامه دهیم:

```

1 format_float:
2     addi sp, sp, -20
3     sw    ra, 16(sp)
4     sw    s2, 12(sp)
5     sw    s3, 8(sp)
6     sw    s4, 4(sp)
7     sw    s5, 0(sp)
8     mv    t0, a1
9     la    t1, zero_float
10    flw   f12, 0(t1)
11    flt.s t1, fa0, f12
12    beq   t1, x0, format_positive
13    li    t2, '-'
14    sb    t2, 0(t0)
15    addi  t0, t0, 1
16    fabs.s fa0, fa0
17 format_positive:
18    fcvt.w.s s4, fa0, rtz
19    mv    s5, s4
20    beq   s4, x0, print_zero_char
21    li    t1, 1
22    li    s3, 10
23 find_divisor_loop:
24    mul   t2, t1, s3
25    ble   t2, s4, continue_divisor_loop
26    j     print_int_digits
27 continue_divisor_loop:
28    mv    t1, t2          # t1 = t2
29    j     find_divisor_loop
30 print_int_digits:
31    beq   t1, x0, int_done
32    div   t2, s4, t1

```

```
    rem  s4, s4, t1
    addi t2, t2, 48
    sb   t2, 0(t0)
    addi t0, t0, 1
    div  t1, t1, s3
    j    print_int_digits
print_zero_char:
    li   t1, '0'
    sb   t1, 0(t0)
    addi t0, t0, 1
int_done:
    li   t1, '.'
    sb   t1, 0(t0)
    addi t0, t0, 1
    fcvt.s.w f1, s5
    fsub.s fa0, fa0, f1
    li   t1, 100
    fcvt.s.w f1, t1
    fmul.s fa0, fa0, f1
    la   t1, fp_half
    flw  f1, 0(t1)
    fadd.s fa0, fa0, f1
    fcvt.w.s t1, fa0, rtz
    li   t2, 10
    div  t3, t1, t2
    rem  t4, t1, t2
    addi t3, t3, 48
    sb   t3, 0(t0)
    addi t0, t0, 1
    addi t4, t4, 48
    sb   t4, 0(t0)
    addi t0, t0, 1
    sb   x0, 0(t0)
    lw   s5, 0(sp)
    lw   s4, 4(sp)
    lw   s3, 8(sp)
    lw   s2, 12(sp)
    lw   ra, 16(sp)
    addi sp, sp, 20
    ret
```

## strcmp ۳.۲

از کد ساده زیر برای مقایسه دو رشته کمک می‌گیریم. خروجی ۰ به معنای برابر بودن دو رشته و خروجی مثبت به معنای بزرگتر بودن رشته اول از رشته دوم از نظر ترتیب لفظ نامه‌ای است:

```

1 strcmp:
2   lb    t0, 0(a0)
3   lb    t1, 0(a1)
4   sub  t2, t0, t1
5   bnez t2, diff
6   beqz t0, equal
7   addi a0, a0, 1
8   addi a1, a1, 1
9   j     strcmp
10 diff:
11   mv    a0, t2
12   ret
13 equal:
14   li    a0, 0
15   ret

```

## cmd1 ۴.۲

این تابع معادل دستور شماره ۱ است که برای ثبت تراکنش مالی بین دو نفر استفاده می‌شود. کد به بخش‌های منطقی زیر تقسیم شده است:

## ۱.۴.۲ تبدیل مقدار عددی و مقداردهی اولیه

```

1 cmd1:
2   mv    a0, s4
3   jal   ra, parse_float
4   fmv.s fs0, fa0

```

- s4: حاوی آدرس رشته عددی ورودی

- fa0/fs0: مقدار float مبلغ تراکنش پس از تبدیل

## ۲.۴.۲ جستجوی فرد اول در لیست

```

1   la    t1, names_head
2   lw    s5, 0(t1)
3 find1:

```

```

4    beqz  s5,  create1
5    lw     t0,  0(s5)
6    mv     a0,  t0
7    la     a1,  name_buf1
8    jal    ra,  strcmp
9    beqz  a0,  name1_ok
10   lw    s5,  8(s5)
11   j     find1

```

• اشارهگر به گره جاری در لیست s5:

• آدرس نام فرد جاری t0:

### ۳.۴.۲ ایجاد گره جدید برای فرد اول

```

1 create1:
2   li    a0, 12
3   li    a7, 9
4   ecall
5   mv    s6, a0
6   li    a0, 9
7   li    a7, 9
8   ecall
9   mv    t0, a0
10  sw    t0, 0(s6)
11  la    t1, name_buf1
12 copy1:
13  lb    t2, 0(t1)
14  sb    t2, 0(t0)
15  addi  t1, t1, 1
16  addi  t0, t0, 1
17  bnez  t2, copy1
18  la    t0, names_head
19  lw    t1, 0(t0)
20  sw    t1, 8(s6)
21  sw    s6, 0(t0)
22  sw    x0, 4(s6)

```

• اشارهگر به گره جدید ایجاد شده s6:

• رجیسترها موقت برای کپی کردن نام t0/t1/t2:

## ۴.۴.۲ بررسی و ایجاد حساب برای فرد اول

```

1 after_find1:
2     lw      t0, 4($s6)
3     beqz  t0, make1
4     j      got1
5 make1:
6     li      a0, 16
7     li      a7, 9
8     ecall
9     mv      t0, a0
10    lw      t1, 0($s6)
11    sw      t1, 0(t0)
12    la      t1, zero_float
13    flw   f12, 0(t1)
14    fsw   f12, 4(t0)
15    sw      x0, 12(t0)
16    sw      t0, 4($s6)
17 got1:

```

• اشارهگر به حساب جدید: t0

• مقدار صفر برای مقداردهی اولیه حساب: f12

## ۵.۴.۲ جستجوی فرد دوم در لیست

```

1     la      t1, names_head
2     lw      s5, 0(t1)
3 find2:
4     beqz  s5, create2
5     lw      t0, 0(s5)
6     mv      a0, t0
7     la      a1, name_buf2
8     jal   ra, strcmp
9     beqz  a0, name2_ok
10    lw      s5, 8(s5)
11    j      find2

```

• اشارهگر به گره جاری در لیست: s5

• آرگومان‌های تابع مقایسه رشته: a0/a1

## ۶.۴.۲ ایجاد گره جدید برای فرد دوم

```

1 name2_ok:
2     mv    s7, s5
3     j     after_find2
4 create2:
5     li    a0, 12
6     li    a7, 9
7     ecall
8     mv    s7, a0
9     li    a0, 9
10    li    a7, 9
11    ecall
12    mv    t0, a0
13    sw    t0, 0(s7)
14    la    t1, name_buf2
15 copy2:
16    lb    t2, 0(t1)
17    sb    t2, 0(t0)
18    addi   t1, t1, 1
19    addi   t0, t0, 1
20    bnez  t2, copy2
21    la    t0, names_head
22    lw    t1, 0(t0)
23    sw    t1, 8(s7)
24    sw    s7, 0(t0)
25    sw    x0, 4(s7)

```

• اشاره‌گر به گره جدید فرد دوم: s7

• رجیسترهاي موقت برای عملیات کپی: t0/t1/t2

۷.۴.۲ برسی و ایجاد حساب برای فرد دوم

```

1 after_find2:
2     lw    t5, 4(s7)
3     beqz t5, make2
4     j     got2
5 make2:
6     li    a0, 16
7     li    a7, 9
8     ecall
9     mv    t5, a0
10    lw    t1, 0(s7)

```

```

11    sw    t1, 0(t5)
12    la    t1, zero_float
13    flw   f12, 0(t1)
14    fsw   f12, 4(t5)
15    sw    x0, 12(t5)
16    sw    t5, 4(s7)
17 got2:

```

• اشارهگر به حساب جدید فرد دوم :t5

• مقدار صفر برای مقداردهی اولیه :f12

#### ۸.۴.۲ بهروزرسانی موجودی حسابها

```

1    lw    t0, 4(s6)
2    flw   fa1, 4(t0)
3    fsub.s fa1, fa1, fs0
4    fsw   fa1, 4(t0)
5    lw    t0, 4(s7)
6    flw   fa1, 4(t0)
7    fadd.s fa1, fa1, fs0
8    fsw   fa1, 4(t0)

```

• مقدار موقت برای محاسبات اعشاری :fa1

• مبلغ تراکنش که قبلاً ذخیره شده بود :fs0

#### ۹.۴.۲ بررسی و بهروزرسانی لیست تراکنش‌های فرد اول

```

1    lw    t0, 4(s6)
2    lw    t4, 12(t0)
3    li    t3, 0
4 p1_loop:
5    beqz  t4, p1_done
6    lw    t6, 0(t4)
7    lw    t1, 0(t6)
8    mv    a0, t1
9    la    a1, name_buf2
10   jal   ra, strcmp
11   beqz  a0, p1_found
12   lw    t4, 12(t4)
13   j     p1_loop
14 p1_found:

```

```

15     flw    f13, 4(t4)
16     fneg.s f14, fs0
17     fadd.s f13, f13, f14
18     fsw    f13, 4(t4)
19     li     t3, 1
20 p1_done:
21     bnez   t3, trans2
22 p1_create:
23     li     a0, 16
24     li     a7, 9
25     ecall
26     mv     t5, a0
27     sw     s7, 0(t5)
28     fneg.s f14, fs0
29     fsw    f14, 4(t5)
30     lw     t0, 4(s6)
31     lw     t6, 12(t0)
32     sw     t6, 12(t5)
33     sw     t5, 12(t0)

```

• t3: فلگ نشان‌دهنده یافتن رکورد

• t4/t5/t6: اشاره‌گرهای موقت برای پیمایش لیست

#### ۱۰.۴.۲ بررسی و بهروزرسانی لیست تراکنش‌های فرد دوم

```

1 trans2:
2     lw     t0, 4(s7)
3     lw     t4, 12(t0)
4     li     t3, 0
5 p2_loop:
6     beqz   t4, p2_done
7     lw     t6, 0(t4)
8     lw     t1, 0(t6)
9     mv     a0, t1
10    la     a1, name_buf1
11    jal    ra, strcmp
12    beqz   a0, p2_found
13    lw     t4, 12(t4)
14    j     p2_loop
15 p2_found:
16     flw    f13, 4(t4)

```

```

17      fadd.s f13, f13, fs0
18      fsw    f13, 4(t4)
19      li     t3, 1
20 p2_done:
21      bnez  t3, next
22 p2_create:
23      li     a0, 16
24      li     a7, 9
25      ecall
26      mv    t5, a0
27      sw    s6, 0(t5)
28      fsw   fs0, 4(t5)
29      lw    t0, 4(s7)
30      lw    t6, 12(t0)
31      sw    t6, 12(t5)
32      sw    t5, 12(t0)
33      j     next

```

• t3: فلگ نشان‌دهنده یافتن رکورد

• f13/f14: مقادیر موقت برای محاسبات اعشاری

## cmd2 ۵.۲

این تابع معادل دستور شماره ۲ است که فرد با بیشترین موجودی را پیدا می‌کند. کد به بخش‌های زیر تقسیم می‌شود:

### ۱.۵.۲ مقداردهی اولیه و بررسی لیست خالی

```

1 cmd2:
2      la    t1, names_head
3      lw    t2, 0(t1)
4      beqz t2, print_minus1
5      lw    s5, 4(t2)
6      flw   f12, 4(s5)
7      mv    t4, t2

```

• t2: اشاره‌گر به سر لیست اسامی

• f12: حاوی بیشترین مقدار موجودی یافت شده

• t4: اشاره‌گر برای پیمایش لیست

### ۲.۵.۲ یافتن بیشترین موجودی

```

1 max_loop:
2     beqz t4, max_done
3     lw t5, 4(t4)
4     flw f13, 4(t5)
5     flt.s t0, f12, f13
6     beqz t0, skip_upd2
7     fmv.s f12, f13
8 skip_upd2:
9     lw t4, 8(t4)
10    j max_loop

```

• t5: اشارگر به حساب فرد جاری

• f13: مقدار موجودی فرد جاری

• t0: نتیجه مقایسه اعشاری

### ۳.۵.۲ بررسی وجود موجودی مثبت

```

1 max_done:
2     la t6, eps_float
3     flw f13, 0(t6)
4     fle.s t0, f12, f13
5     bnez t0, print_minus1

```

• f13: مقدار اپسیلون برای مقایسه اعشاری

• t0: نتیجه مقایسه برای بررسی مثبت بودن موجودی

### ۴.۵.۲ یافتن فرد با بیشترین موجودی

```

1     li t3, 0
2     mv t4, t2
3 find_max:
4     beqz t4, done_max
5     lw t5, 4(t4)
6     flw f14, 4(t5)
7     fsub.s f15, f14, f12
8     fabs.s f15, f15
9     flt.s t0, f15, f13
10    bnez t0, eq_max
11    j cont2

```

• t3: ذخیره‌سازی نام فرد با بیشترین موجودی

• f14/f15: مقادیر موقت برای مقایسه اعشاری

#### ۵.۵.۲ مقایسه نام‌ها برای ترتیب الفبایی

```

1 eq_max:
2   lw    t5, 0(t4)
3   beqz t3, set_max
4   mv    a0, t5
5   mv    a1, t3
6   jal   ra, strcmp
7   bltz a0, set_max
8 cont2:
9   lw    t4, 8(t4)
10  j     find_max

```

• آرگومان‌های تابع مقایسه رشته‌ها: a0/a1

• t5: نام فرد جاری

#### ۶.۵.۲ ذخیره نام فرد انتخاب شده

```

1 set_max:
2   mv    t3, t5
3   j     cont2

```

• t3: نام فرد با بیشترین موجودی و کمترین ترتیب الفبایی

#### ۷.۵.۲ چاپ نتیجه

```

1 done_max:
2   mv    a0, t3
3   li    a7, 4
4   ecall
5   la    a0, newline
6   li    a7, 4
7   ecall
8   j     next

```

• a0: نام فرد برای چاپ

• a7: شماره سامانه‌کال برای چاپ رشته

## cmd۳ ۶.۲

کد این بخش درست مانند بخش قبلی است اما این دفعه به دنبال مینیمم balance هستیم:

```

1 cmd3:
2     la      t1, names_head
3     lw      t2, 0(t1)
4     beqz  t2, print_minus1
5     lw      s5, 4(t2)
6     beqz  s5, min_loop_next_person
7     flw    f12, 4(s5)
8     j      min_loop_start
9 min_loop_next_person:
10    la     t6, zero_float
11    flw   f12, 0(t6)
12 min_loop_start:
13    mv     t4, t2
14 min_loop:
15    beqz  t4, min_done
16    lw     t5, 4(t4)
17    beqz  t5, skip_min
18    flw   f13, 4(t5)
19    flt.s t0, f13, f12
20    beqz  t0, skip_min
21    fmv.s f12, f13
22 skip_min:
23    lw     t4, 8(t4)
24    j      min_loop
25 min_done:
26    la     t6, eps_float
27    flw   f13, 0(t6)
28    fneg.s f14, f13
29    fge.s t0, f12, f14
30    bnez  t0, print_minus1
31    li     t3, 0
32    mv     t4, t2
33 find_min:
34    beqz  t4, done_min
35    lw     t5, 4(t4)
36    beqz  t5, cont_min
37    flw   f15, 4(t5)

```

```

۳۸     fsub.s f16, f15, f12
۳۹     fabs.s f16, f16
۴۰     flt.s t0, f16, f13
۴۱     bnez  t0, eq_min
۴۲     j      cont_min
۴۳ eq_min:
۴۴     lw    t5, 0(t4)
۴۵     beqz t3, set_min2
۴۶     mv    a0, t5
۴۷     mv    a1, t3
۴۸     jal   ra, strcmp
۴۹     bltz a0, set_min2
۵۰ cont_min:
۵۱     lw    t4, 8(t4)
۵۲     j     find_min
۵۳ set_min2:
۵۴     mv    t3, t5
۵۵     j     cont_min
۵۶ done_min:
۵۷     mv    a0, t3
۵۸     li    a7, 4
۵۹     ecall
۶۰     la    a0, newline
۶۱     li    a7, 4
۶۲     ecall
۶۳     j     next

```

## cmd۴ ۷.۲

این تابع معادل دستور شماره ۴ است که تعداد افراد بدھکار به یک فرد خاص را محاسبه می‌کند. کد به بخش‌های زیر تقسیم می‌شود:

## ۱.۷.۲ مقداردهی اولیه و جستجوی فرد مورد نظر

```

۱ cmd4:
۲     la    t1, names_head
۳     lw    s5, 0(t1)
۴ find4:
۵     beqz s5, print_zero4
۶     lw    t4, 0(s5)
۷     mv    a0, t4

```

```

8      la    a1, name_buf1
9      jal   ra, strcmp
10     beqz a0, got4
11     lw    s5, 8(s5)
12     j    find4

```

- s5: اشارهگر به گره جاری در لیست اسامی
- t4: آدرس نام فرد جاری
- a0/a1: آرگومان‌های تابع مقایسه رشته‌ها

#### ۲.۷.۲ بررسی وجود حساب برای فرد مورد نظر

```

1 got4:
2     lw    t0, 4(s5)
3     beqz t0, print_zero4
4     lw    t4, 12(t0)
5     li    t1, 0
6     la    t6, eps_float
7     flw   f13, 0(t6)

```

- t0: اشارهگر به حساب فرد مورد نظر
- t4: اشارهگر به لیست تراکنش‌ها
- t1: شمارنده تعداد بدھکاران
- f13: مقدار اپسیلون برای مقایسه اعشاری

#### ۳.۷.۲ شمارش بدھکاران

```

1 cred4:
2     beqz t4, done4
3     flw   f14, 4(t4)
4     flt.s t0, f13, f14
5     beqz t0, skip4
6     addi  t1, t1, 1
7 skip4:
8     lw    t4, 12(t4)
9     j    cred4

```

- f14: مقدار تراکنش جاری
- t0: نتیجه مقایسه برای تشخیص بدھکار بودن
- t4: اشارهگر به تراکنش بعدی

## چاپ نتیجه ۴.۷.۲

```

1 done4:
2     mv    a0, t1
3     li    a7, 1
4     ecall
5     la    a0, newline
6     li    a7, 4
7     ecall
8     j     next

```

- a0: تعداد بدهکاران برای چاپ
- a7: شماره سامانه‌کال برای چاپ عدد

## ۵.۷.۲ حالتهای خاص و خطأ

```

1 print_zero4:
2     li    a0, 0
3     li    a7, 1
4     ecall
5     la    a0, newline
6     li    a7, 4
7     ecall
8     j     next

```

- a0: مقدار صفر برای چاپ در صورت عدم وجود بدهکار
- a7: شماره سامانه‌کال برای چاپ عدد

## cmd5 ۸.۲

این بخش نیز درست مانند بخش قبل است با این تفاوت که در اینجا به دنبال تعداد طلبکاران هستیم:

```

1 cmd5:
2     la    t1, names_head
3     lw    s5, 0(t1)
4 find5:
5     beqz s5, print_zero5
6     lw    t4, 0(s5)
7     mv    a0, t4
8     la    a1, name_buf1
9     jal   ra, strcmp

```

```
10      beqz  a0,  got5
11      lw     s5,  8(s5)
12      j     find5
13  got5:
14      lw     t0,  4(s5)
15      beqz  t0,  print_zero5
16      lw     t4,  12(t0)
17      li     t1,  0
18      la     t6,  eps_float
19      flw   f13,  0(t6)
20      fneg.s f13,  f13
21  deb5:
22      beqz  t4,  done5
23      flw   f14,  4(t4)
24      flt.s t0,  f14,  f13
25      beqz  t0,  skip5
26      addi   t1,  t1,  1
27  skip5:
28      lw     t4,  12(t4)
29      j     deb5
30  done5:
31      mv   a0,  t1
32      li   a7,  1
33      ecall
34      la   a0,  newline
35      li   a7,  4
36      ecall
37      j    next
38  print_zero5:
39      li   a0,  0
40      li   a7,  1
41      ecall
42      la   a0,  newline
43      li   a7,  4
44      ecall
45      j    next
```

## cmd6 ۹.۲

این تابع معادل دستور شماره ۶ است که میزان بدھی یا طلب یک فرد به فرد دیگر را محاسبه می‌کند. کد به بخش‌های زیر تقسیم می‌شود:

## ۱.۹.۲ جستجوی فرد اول در لیست

```

1 cmd6:
2     la    t1, names_head
3     lw    s5, 0(t1)
4 find6:
5     beqz s5, print_zero6
6     lw    t4, 0(s5)
7     mv    a0, t4
8     la    a1, name_buf1
9     jal   ra, strcmp
10    beqz a0, got6
11    lw    s5, 8(s5)
12    j     find6

```

- s5: اشاره‌گر به گره جاری در لیست اسامی
- t4: آدرس نام فرد جاری
- a0/a1: آرگومان‌های تابع مقایسه رشته‌ها

## ۲.۹.۲ بررسی وجود حساب برای فرد اول

```

1 got6:
2     lw    t0, 4(s5)
3     beqz t0, print_zero6
4     lw    t4, 12(t0)

```

- t0: اشاره‌گر به حساب فرد اول
- t4: اشاره‌گر به لیست تراکنش‌های فرد اول

## ۳.۹.۲ جستجوی تراکنش با فرد دوم

```

1 scan6:
2     beqz t4, print_zero6
3     lw    t5, 0(t4)
4     lw    t1, 0(t5)
5     mv    a0, t1

```

```

6      la    a1, name_buf2
7      jal   ra, strcmp
8      beqz a0, found6
9      lw    t4, 12(t4)
10     j     scan6

```

- t5: اشارهگر به گره نام فرد در تراکنش
- t1: آدرس نام فرد در تراکنش
- a0/a1: آرگومان‌های تابع مقایسه رشته‌ها

#### ۴.۹.۲ پردازش و نمایش نتیجه

```

1 found6:
2     flw  fa0, 4(t4)
3     la   a1, io_buffer
4     jal  ra, format_float
5     la   a0, io_buffer
6     li   a7, 4
7     ecall
8     la   a0, newline
9     li   a7, 4
10    ecall
11    j    next

```

- fa0: مقدار بدهی/طلب به صورت float
- a1: بافر خروجی برای فرمت‌دهی عدد
- a7: شماره سامانه‌کال برای چاپ رشته

#### ۵.۹.۲ حالتهای خاص و خطأ

```

1 print_zero6:
2     flw  fa0, zero_float
3     la   a1, io_buffer
4     jal  ra, format_float
5     la   a0, io_buffer
6     li   a7, 4
7     ecall
8     la   a0, newline
9     li   a7, 4
10    ecall
11    j    next

```

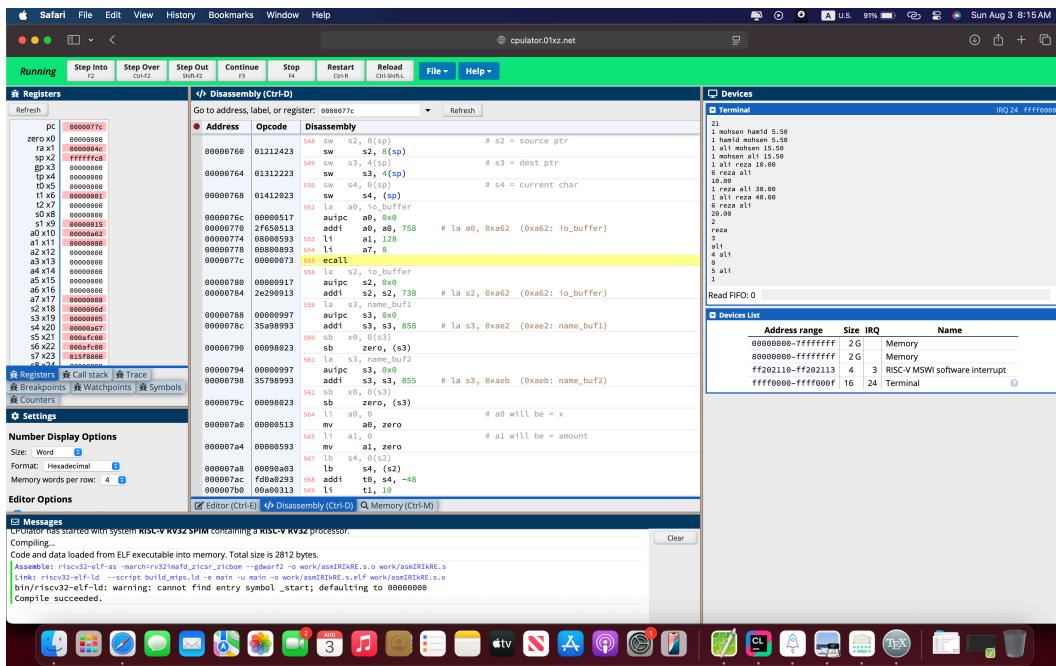
- fa0: مقدار صفر برای زمانی که تراکنشی یافت نشد
- a0: آدرس بافر خروجی
- a7: شماره سامانه‌کال برای چاپ رشته

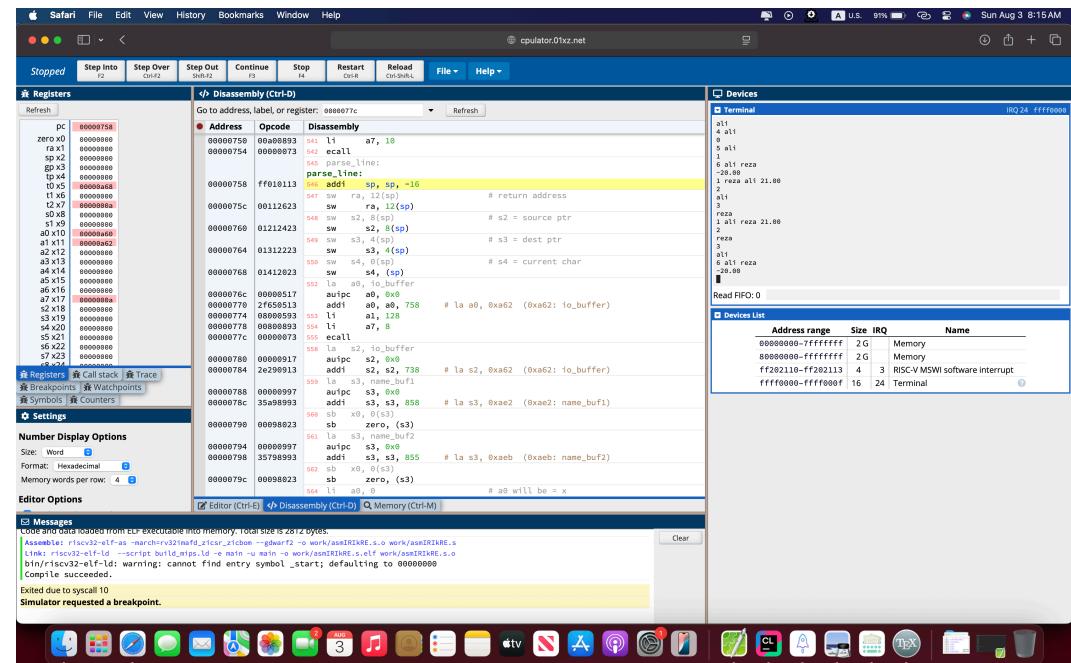
٣ فصل

تست عملکرد

برای این کار تست کیس مطرح شده در صورت پروژه و یک تست کیس دلخواه را با استفاده از ابزار CPU Lator بررسی می کنیم:

تست کیس اول:





تست کیس دوم:

## Input:

|    |             |
|----|-------------|
| 1  | 11          |
| 2  | 1 A B 29.01 |
| 3  | 3           |
| 4  | 1 A C 1.19  |
| 5  | 2           |
| 6  | 1 B C 5.61  |
| 7  | 2           |
| 8  | 5 A         |
| 9  | 6 A B       |
| 10 | 6 A C       |
| 11 | 6 B C       |
| 12 | 6 C B       |

## Expected Output:

|   |        |
|---|--------|
| 1 | A      |
| 2 | B      |
| 3 | B      |
| 4 | 2      |
| 5 | -29.01 |
| 6 | -1.19  |

