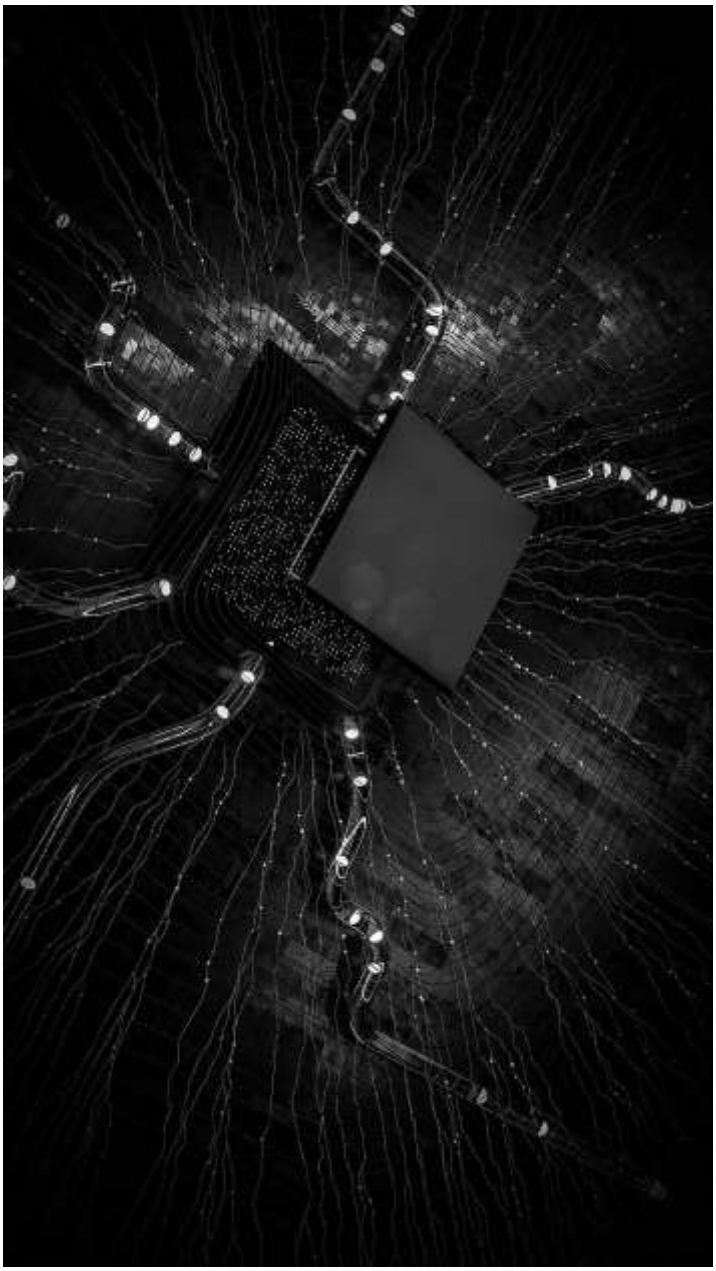

آزمایشگاه ریزپردازنده

تمرین پنج

سینا عربی – سالار جهانشیری – امیرعلی وکیلی

سوالات تحلیلی:

- **در برنامه نویسی اسمبلی ARM ، از چه دستوری برای قرار دادن یک مقدار 32 بیتی ثابت در یک رجیستر استفاده می‌شود؟ این دستور به چه شکل عمل می‌کند؟**

برای انجام این کار میتوان از دستور های mov,ldr استفاده کرد.

دستور mov

سینتکس استفاده از آن به شکل : $mov \quad < reg >, < immediate >$

است که در قسمت reg میتوان یکی از رجیستر ها برای مثال ro را قرار داد و در قسمت ایمدیت نیز میتوان یک عدد همانند 43 را گذاشت.

نحوه عملکرد دستور mov به این صورت است که ابتدا مقدار <immediate>را به صورت باینری به 32 بیت تبدیل می کند. سپس این مقدار باینری را در رجیستر <reg>ذخیره می کند.

دستور LDR:

سینتکس استفاده از آن به شکل : $mov \quad < reg >, < address >$

است که در قسمت reg میتوان یکی از رجیستر ها برای مثال ro را قرار داد و در قسمت آدرس نیز میتوان آدرس مقدار ثابت مد نظرمان را گذاشت.

دستور ldr به طور کلی از دستور mov کندتر است، زیرا برای بارگیری مقدار از حافظه به رجیستر به زمان بیشتری نیاز دارد.

- **محدودیت مقدار immediate در دستورات ریاضی (Thumb-2) چند بیت است؟**

دستورالعمل (ISA)های ۳۲ بیتی را Thumb-2 معرفی میکند، محدودیت بر روی immediate ها ۱۲ بیت می‌باشد که دستورالعمل های دیگر تعاریف مشابه قبل دارند به غیر از دستورالعمل های شیفتی و این به این علت است که برخلاف x86 دستورالعمل هایarm ، معمولا ۳۲ و یا ۱۶ بیتی می‌باشند وقتی از Thumb-2 استفاده می‌کنند.

نحوه کار:

برای پیاده‌سازی توابع در ابتدا رجیسترهای لازم برای تغییر را در استک PUSH کرده و در نهایت هنگام خروج از تابع POP می‌کنیم، تا تاثیری روی روند کار بقیه توابع و تابع اصلی برنامه(MAIN) نداشته باشند.

برای بررسی اول بودن عدد N همانند قبل باقی‌مانده آن به اعداد 2 تا N-1 را محاسبه نموده و در صورت صفر بودن، عدد را غیر اول اعلام می‌کنیم.

```
73  IS_PRIME :
74      PUSH {LR}
75      PUSH {R0}
76      PUSH {R1}
77      PUSH {R2}
78      PUSH {R3}
79      PUSH {R4}
80      PUSH {R5}
81      PUSH {R6}
82
83      MOV R2, #2
84      LDR R0, =NUM
85      LDR R1, [R0]
86      LOOP_IS_PRIME:
87
88          CMP R2, R1
89          BEQ IS_PRIME_TRUE
90
91          @CALC MOD
92          SDIV R3, R1, R2 @R3 = R1 / R2
93          MUL R4, R3, R2  @R4 = R2 * R3
94          SUB R3, R1, R4  @R3 = R1 mod R2
95          ADD R2, R2, #1
96          CMP R3, #0
97          BNE LOOP_IS_PRIME
98
99          LDR R9, =FLAG
100         MOV R7, 0
101         STR R7, [R9]
102
103         POP {R6}
104         POP {R5}
105         POP {R4}
106         POP {R3}
107         POP {R2}
108         POP {R1}
109         POP {R0}
110         POP{LR}
111         BX LR
112
```

نکته مهم در این بخش عدم دسترسی مستقیم به باقی‌مانده تقسیم و محاسبه آن از طریق تفاضل عدد و ضرب خارج قسمت در مقسوم‌علیه(قضیه تقسیم) می‌باشد.

```
IS_PRIME_TRUE:
    MOV R7, 1
    LDR R9, =FLAG
    STR R7, [R9]

    POP {R6}
    POP {R5}
    POP {R4}
    POP {R3}
    POP {R2}
    POP {R1}
    POP {R0}
    POP{LR}
    BX LR
```

در نهایت اگر بعد از بررسی تمام N-2 عدد، بر هیچ‌کدام بخش‌پذیر نبود، متغیر FLAG را لود کرده و مقدار آن را 1 می‌کنیم.

برای بررسی پالیندرومی بودن عدد N(مثلا abc) همانند قبل برعکس آن را(cba) را محاسبه کرده و در صورتی که این دو عدد یکسان باشند، عدد را پالیندرومی اعلام می‌کنیم.

```
REVERSE:
    PUSH {LR}
    PUSH {R0}
    PUSH {R1}
    PUSH {R2}
    PUSH {R3}
    PUSH {R4}
    PUSH {R5}
    PUSH {R6}

    MOV R7, #0
    MOV R2, #10

    LDR R0, =NUM
    LDR R1, [R0]

    LOOP_REVERSE:
        @CALC MOD
        SDIV R3, R1, R2 @R3 = R1 / R2
        MUL R4, R3, R2  @R4 = R2 * R3
        SUB R5, R1, R4  @R5 = R1 mod R2

        MOV R1, R3      @R1 = R1 / 10

        MUL R7, R7, R2 @R7 = R7 * 10
        ADD R7, R7, R5  @R7 = R7 + R5


        CMP R1, #0
        BNE LOOP_REVERSE

        POP {R6}
        POP {R5}
        POP {R4}
        POP {R3}
        POP {R2}
        POP {R1}
        POP {R0}
        POP{LR}
        BX LR
```

برای محاسبه برعکس عدد به ترتیب هر بار عدد را تقسیم بر 10 کرده و باقی‌مانده آن را با 10 برابر عدد ساخته شده در مرحله قبل جمع می‌کنیم، در نهایت برعکس این عدد در رجیستر R7ساخته شده است.

```
IS_PALINDROME:
    PUSH {LR}
    PUSH {R0}
    PUSH {R1}


    BL REVERSE  @PUT REVERSE OF NUM IN R7

    LDR R0, =NUM
    LDR R1, [R0]
    CMP R7, R1
    BEQ IS_PALINDROME_TRUE

    LDR R9, =FLAG
    MOV R7, 0
    STR R7, [R9]

    POP {R1}
    POP {R0}
    POP{LR}
    BX LR


    IS_PALINDROME_TRUE:
        MOV R7, 1
        LDR R9, =FLAG
        LDR R1, [R9]
        ADD R7, R7, R1
        STR R7, [R9]

        POP {R1}
        POP {R0}
        POP{LR}
        BX LR
```

در ادامه در صورتی که عدد اصلی که در رجیستر R1 قرار دارد با R7 معادل باشد، عدد پالیندرومی است و این بار FLAG لود شده با 1 جمع و دوباره ذخیره می‌شود.

```
main :
    PUSH {LR}
    PUSH {R0}
    PUSH {R1}
    PUSH {R2}
    PUSH {R3}
    PUSH {R4}

    MOV R2, #0


    ITERATE_NUMBERS:
        LDR R0, =NUM
        LDR R1, [R0]

        BL IS_PRIME
        BL IS_PALINDROME

        LDR R9, =FLAG
        LDR R3, [R9]

        MOV R4, #0
        STR R4, [R9]


        MOV R4, R1
        ADD R4, R4, #1
        LDR R0, =NUM
        STR R4, [R0]

        CMP R3, #2
        BNE ITERATE_NUMBERS


        LDR R0, =PP_NUMBERS
        ADD R0, R0, R2
        STR R1, [R0]
        ADD R2, R2, #1

        CMP R2, #10
        BNE ITERATE_NUMBERS
```

در بخش main برنامه از عدد 2 شروع به بررسی اول و پالیندرومی بودن اعداد می‌کنیم، در صورتی که بعد از بررسی مقدار FLAG 2 باشد، یعنی عدد هر دو ویژگی را داشته و در آرایه PP_NUMBERS ذخیره می‌شود. این کار را تا زمانی که به ده عدد برسیم ادامه داده و برنامه را تمام می‌کنیم.

در ادامه خروجی برنامه سیو شده در ردیف اول مموری نشان داده شده است: