



به نام خدا

دانشکده‌ی مهندسی برق و کامپیوتر دانشکده فنی دانشگاه تهران  
مبانی کامپیوتر و برنامه‌نویسی



نیمسال دوم

۹۸-۹۹

عنوان:

برنامه‌سازی و I/O در LC3

استاد: دکتر مرادی

در آخرین جلسه آزمایشگاه شما چند برنامه ی ساده با ماشین LC3 می نویسد و با ورودی و خروجی (I/O) آن آشنا می شوید.

۱. انجام دهید!

۱. از پوشه LC3 اسمبلر ماشین یا همان LC3Edit را باز کنید.
۲. اکنون با فرض این که مقدار X در رجیستر R0 و مقدار Y در رجیستر R1 قرار دارد (مقدار X و مقدار Y را به دلخواه بدهید. نحوه انجام این کار در قسمت راهنمایی سوال توضیح داده شده است)، برنامه ای بنویسد که به ترتیب:
  - حاصل جمع  $X + Y$  را در رجیستر R2 ذخیره کند.
  - حاصل  $X \text{ AND } Y$  را در رجیستر R3 ذخیره کند.
  - حاصل  $X \text{ OR } Y$  را در رجیستر R4 ذخیره کند. (به دنبال دستور OR نگردید. چنین دستوری مستقیماً برای ماشین LC3 وجود ندارد!)
  - اگر X زوج بود آن را در رجیستر R7 و اگر فرد بود در رجیستر R6 ذخیره کند.

قسمت ۱: کد مربوطه را در قسمت زیر قرار دهید.

```
.ORIG x3000
LD R0,X ; R0 = X
LD R1,Y ; R1 = Y
ADD R2,R0,R1 ; R2 = X + Y
AND R3,R1,R0 ; R3 = X AND Y
; X OR Y = NOT [NOT(X) AND NOT(Y)]
NOT R4,R0 ; R4 = NOT(X)
NOT R5,R1 ; R5 = NOT(Y)
AND R4,R4,R5 ; R4 = [NOT(X) AND NOT(Y)]
NOT R4,R4 ; R4 = NOT [NOT(X) AND NOT(Y)] = X OR Y
AND R5,R5,#0 ; FREE R5
; ODD AND EVEN NUMBER DETECTION
AND R5,R0,#1
BRz EVEN ; GO TO EVEN IF X AND #1 WAS ZERO
; ELSE
AND R6,R6,#0 ; FREE R6
ADD R6,R0,#0 ; R6 = R0
AND R5,R5,#0 ; FREE R5
CONTINUE HALT
X .FILL #3
Y .FILL #7
EVEN AND R7,R7,#0 ; FREE R7
ADD R7,R0,#0 ; R7 = R0
BRnzp CONTINUE
.END
```

**دقت کنید که** تمامی عملیات فوق باید در یک برنامه انجام شود و نه چهار برنامه ی مجزا، یعنی پس از این که حاصل جمع  $X + Y$  در R2 ذخیره شد، دیگر نباید تا انتهای برنامه مقدار R2 تغییر کند

### راهنمایی:

- در بخش سوم سوال بدیهی است که باید عملیات OR را با ترکیبی از عملیات AND و NOT پیاده سازی کنید. همچنین در بخش چهارم نیاز دارید که از دستور پرش شرطی یا BR استفاده کنید. برای بررسی این مسأله که آیا متغیر X زوج است یا نه کافی است سایر بیت های آن را به جز بیت کم ارزش صفر کنید. (به کمک دستور AND) سپس مقدار موجود در X اگر صفر شده باشد، دستور BRz به خانه ای که در مقابل این دستور ذکر شده باشد پرش خواهد کرد.

### دقت کنید!

احتمالاً در ابتدای راه اندازی ماشین LC3 تمام رجیسترها صفر هستند. پس برای این که نتایج برنامه ی فوق را به درستی مشاهده کنید، پیش از اجرای برنامه مقدار رجیسترها را به کمک کلیک راست و انتخاب Set Value تغییر دهید.

### ۲. انجام دهید!

۱. یک فایل جدید ایجاد کنید.
۲. اکنون با فرض این که مقدار X در رجیستر R1 و مقدار Y در رجیستر R2 قرار دارد، برنامه ای بنویسید که ابتدا قدر مطلق X را محاسبه کرده و در R1 قرار دهد و سپس قدر مطلق Y را حساب کرده و در R2 قرار دهد و در انتها عددی که قدر مطلق آن بزرگتر است را در خانه ی R2 قرار دهد.

**راهنمایی:** برای فهمیدن این مطلب که مقدار موجود در یک رجیستر مثبت است یا منفی، کافی است تا یک عملیات ریاضی که حاصل آن رجیستر را تغییر نمی دهد انجام دهید و نتیجه را در همان رجیستر ذخیره کنید. با انجام این عملیات flag های n و Z و p و بر اساس مقادیر آن رجیستر تغییر خواهد کرد.

قسمت ۲: کد مربوطه را در کادر زیر قرار دهید.

```
.ORIG X3000
LD R1,X ; R1 = X
LD R2,Y ; R2 = Y
ADD R1,R1,#0
BRn XCHANGE ; IF X WAS NEGATIVE GO TO XCHANGE
CNTNUE1 ADD R2,R2,#0
BRn YCHANGE ; IF Y WAS NEGATIVE GO TO YCHANGE
; WHICH ONE IS BIGGEST ?
CNTNUE2 NOT R3,R2
ADD R3,R1,R3
BRp XCOPY ; IF |X| WAS BIGGER GO TO XCOPY
BACK HALT
XCHANGE NOT R1,R1
ADD R1,R1,#1
BRnzp CNTNUE1
YCHANGE NOT R2,R2
ADD R2,R2,#1
BRnzp CNTNUE2
XCOPY LD R2,X
BRnzp BACK
X .FILL #-9
Y .FILL #-2
.END
```

## آشنایی با I/O:

برای این که برنامه ی ما بتواند به خوبی با کاربر ارتباط برقرار کند، باید بتوانیم ورودی را به آسانی از کاربر بگیریم و خروجی را در فرمتی مناسب به او نشان دهیم. برای این منظور در LC3 از LC3 Console استفاده می کنیم. در این قسمت می توان به برنامه ورودی داد یا چیزی روی آن برای کاربر چاپ کرد.

به جدول زیر توجه کنید:

دستور	توضیح
IN	با رسیدن برنامه به این خط اجرای برنامه متوقف می شود و برنامه منتظر دادن ورودی از طریق LC3 Console توسط کاربر میشود. دستور IN یک کاراکتر اسکی را از کاربر می گیرد و آن را در ۸ بیت کم ارزش R0 ذخیره می کند. <u>دقت کنید</u> که پس از اجرای دستور IN محتوای سابق R0 از بین می رود. پس اگر در آن دیتای خاصی دارید، آن را در جای دیگری ذخیره کنید.

OUT	با اجرای این دستور محتوای R0 (هشت بیت کم ارزش آن) در خروجی چاپ می شود.
PUTS	این دستور که مخفف PUT String است، از آدرسی که R0 به آن اشاره می کند شروع به چاپ خانه های حافظه (هشت بیت کم ارزش آن ها) می کند تا زمانی که به خانه ای با محتوای x0000 (NULL) برسد. به عبارت دیگر یک رشته از کاراکتر ها که از خانه ای با آدرسی که در R0 ذخیره شده است را چاپ می کند.
.STRINGZ	این دستور یکی دیگر از دستوراتی است که با اسمبلر صحبت می کند. (مثل .END, .ORIG, ...). این دستور همانند چند تا دستور .FILL. پشت سرهم می ماند. با این دستور می توانید یک رشته از کاراکتر ها را در خانه های پشت سر هم در حافظه قرار دهید. (این دستور به صورت پیش فرض در انتهای رشته ی شما NULL را اضافه می کند).

✓ برای درک بهتر دستورات فوق به مثال زیر توجه کنید:

```
.ORIG x3050
LEA R0, HELLO ; put address of this label in R0
PUTS ; print the string starting from R0 until x0000 appears
AND R0,R0,0;
IN ; read a character from console (and store it in R0)
ADD R0,R0,1;
OUT ; print R0[7:0] on console
HALT
HELLO .STRINGZ "HELLO WORLD" ; place this string in memory starting with
;address of x3057 (why?)
.END
```

← ۳. انجام دهید! (خواندن از ورودی)

یک برنامه ی ساده بنویسید که عددی را از ورودی خوانده و در یک رجیستر بریزد.

قسمت ۳: کد مربوطه را در قسمت زیر قرار دهید.

```

.ORG X3000

IN

ADD R2,R0,#0      ; R2 = ENTERED NUMBER

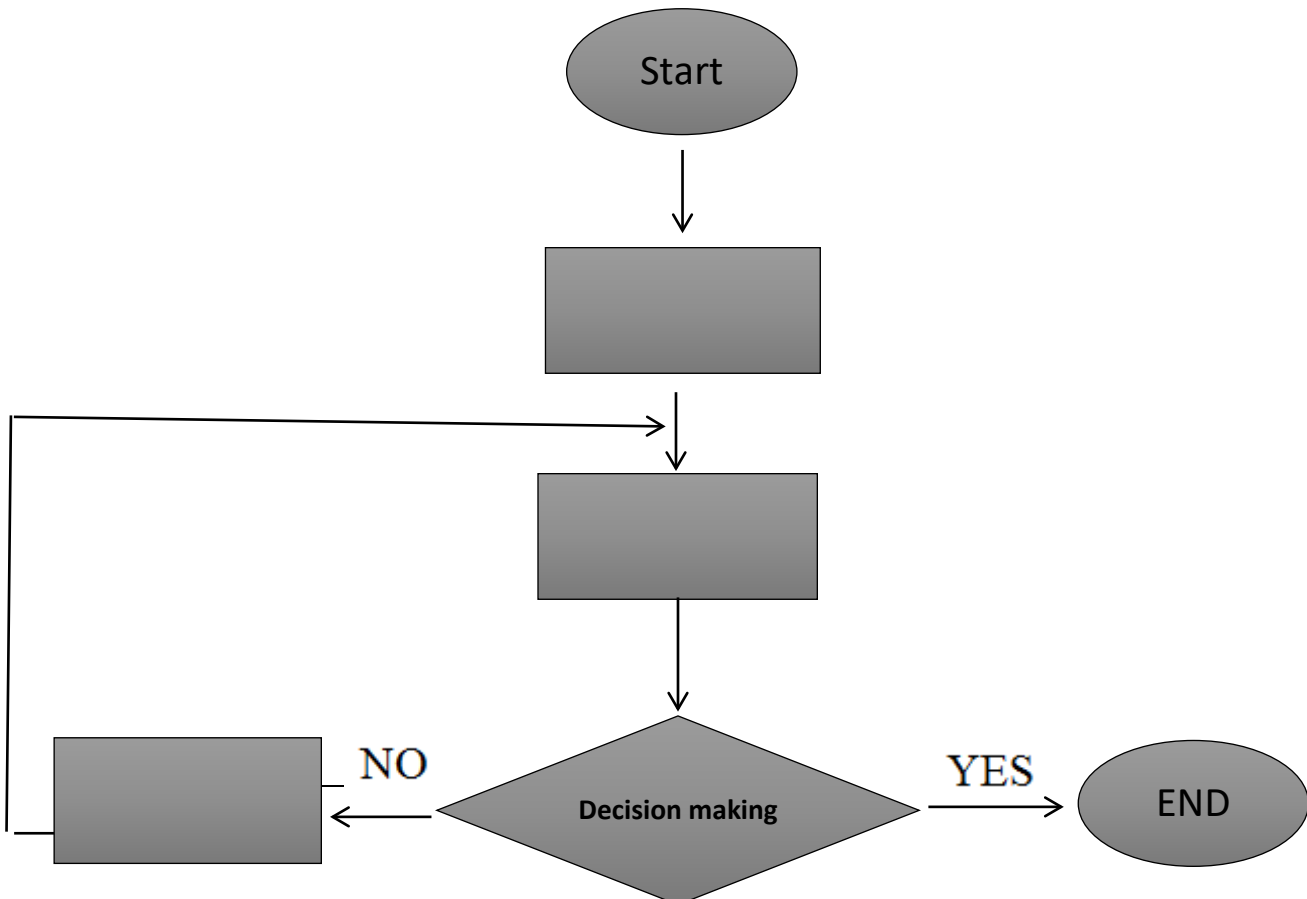
HALT

.END

```

← ۴. انجام دهید! (محاسبه ی حاصل ضرب مقادیر دو رجیستر)

در این قسمت می خواهیم برنامه ای بنویسیم که محتوای دو رجیستر R3 و R4 را در هم ضرب کرده و در رجیستر R5 ذخیره کند. احتمالاً تا حالا متوجه شده اید که شبیه ساز LC3 دستور ضرب ندارد. برای پیاده سازی دستور ضرب از مفهوم جمع استفاده میکنیم. به سادگی برای این کار عدد اول را به اندازه عدد دوم با خودش جمع می کنیم. برای انجام این قسمت می توانید ابتدا فلوچارت<sup>۱</sup> زیر را کامل کنید(تحویل این فلوچارت الزامی نیست):



## فلوچارت حاصل ضرب دو عدد صحیح نامنفی

حال به کمک فلوچارتی که تکمیل کردید، کد اسمبلی این قسمت را بنویسید.

**قسمت ۴:** کد خود را در کادر زیر بنویسید.

```
.ORIG 3000
LD    R3,X      ; R3 = X
LD    R4,Y      ; R4 = Y
AND   R5,R5,#0   ; FREE R5
LOOP  ADD R5,R5,R3 ; R5 IS SUM AND SUM = SUM + X
      ADD R4,R4,#-1 ; DO THIS LOOP Y TIMES
      BRp LOOP
      HALT
X     .FILL #9
Y     .FILL #4
      .END
```

## تبدیل یک قطعه کد به تابع:

برای تبدیل یک قطعه کد به تابع کارهای زیر را انجام دهید.

- یک برچسب (Label) به عنوان نام تابع در نظر بگیرید.
- قطعه کد مورد نظر را زیر این برچسب قرار دهید.
- در پایان قطعه کد، آدرس بازگشت (Return Address) را که در رجیستر R7 ذخیره شده است، در رجیستر PC قرار دهید. (این کار را با اجرای دستور RET به عنوان آخرین دستور تابع خود می توانید انجام دهید).

**توجه:** آدرس بازگشت از تابع که گفته شده در R7 قرار دارد را خودتان باید به هنگام فراخوانی تابع در R7 قرار دهید. در قسمت بعدی (صدا زدن یک تابع) نحوه ی انجام این کار توضیح داده شده است.

راهنمایی: به هنگام می‌توانید از نمونه کد زیر برای نوشتن یک تابع کمک بگیرید.

```
1      LDI R0, N      ; Argument N is now in R0
2      JSR F          ; Jump to subroutine F.
3      STI R1, FN
4      HALT
5 N      .FILL 3120    ; Address where n is located
6 FN      .FILL 3121    ; Address where fn will be stored.
7                      ; Subroutine F begins
8 F      AND R1, R1, x0 ; Clear R1
9      ADD R1, R0, x0 ; R1 ← R0
10     ADD R1, R1, R1 ; R1 ← R1 + R1
11     ADD R1, R1, x3 ; R1 ← R1 + 3. Result is in R1
12     RET              ; Return from subroutine
13     END
```

Listing 5.1: A subroutine for the function  $f(n) = 2n + 3$ .

## صدا زدن (فراخوانی) یک تابع<sup>۲</sup>:

برای اجرای یک تابع لازم است که آدرس سر تابع را در رجیستر PC قرار دهیم. در این صورت ادامه ی اجرای برنامه از سر تابع خواهد بود. بعد از آن که کار تابع تمام شد، مقدار رجیستر PC باید به مقدار قبل از صدا زدن تابع به اضافه ۱ (چرا؟) تغییر کند. مسئله ای که مطرح می شود این است که آدرس بازگشت را چگونه به دست بیاوریم، چون یک تابع ممکن است چندین بار و از آدرس های مختلف صدا زده شود. بنابر این باید قبل از صدا زدن تابع، آدرس فعلی را در جایی که تابع هم از آن با خبر است ذخیره کنیم. برای این کار می توانیم کارهای زیر را انجام دهیم:

آدرس دستور بعد از JSR (یا در واقع PC یک واحد اضافه شده) را در R7 ذخیره کن و به MyFunc پرش کن. JSR MyFunc

توجه کنید که MyFunc یک label است.

حال همانطور که در قسمت قبل توضیح داده شد در پایان اجرای تابع به عنوان آخرین دستور آن باید دستور زیر را بنویسید:

RET

این دستور محتوای رجیستر R7 (که به هنگام فراخوانی با دستور JSR تعیین شده بود) را در PC قرار می دهد.

پس از اجرای این دستور، به دستور بعد از فراخوانی تابع پرش می کنیم و اجرای برنامه ادامه می یابد.

توجه: دستور RET مشابه دستور JMP بوده، با این تفاوت که همواره به آدرسی که در R7 وجود دارد پرش می کند.

<sup>2</sup> Function Call

## ۵. انجام دهید! (تبدیل برنامه های قبلی به دو تابع مجزا)

حال برنامه های قبلی را تبدیل به دو تابع مجزا کنید. یعنی تابع هایی بسازید که یکی عملیات ضرب و دیگری عملیات خواندن از ورودی را انجام دهد.

**دقت کنید:** مطمئن شوید که برنامه ی شما درست کار می کند.

**نکته:** در زمان صدا زده شدن تابع، محتوای رجیستر R7 تغییری می کند؟ (چرا؟)

**قسمت ۵:** کد و پاسخ خود را در کادر زیر بنویسید

در هنگام صدا زدن تابع و استفاده از دستور JSR، آدرس خونه بعدی یعنی PC به اضافه یک در R7 ذخیره میشود. و هنگام RET، به همان خانه ای که در R7 ذخیره شده باز میگردد.

تابعی جهت انجام عمل ضرب:

```
.ORIG 3000
LD      R3,X          ; R3 = X
LD      R4,Y          ; R4 = Y
JSR     ZARB          ; GO TO ZARB AND STORE PC + 1 IN R7
HALT

; ZARB FUNCTION : R5 = R3 * R4
ZARB    AND      R5,R5,#0 ; FREE R5
LOOP    ADD      R5,R5,R3  ; R5 IS SUM AND SUM = SUM + X
        ADD      R4,R4,#-1 ; DO THIS LOOP Y TIMES
        BRp     LOOP
        RET              ; GO TO ADDRESS THAT STORED IN R7
X        .FILL #9
Y        .FILL #4
        .END
```

تابعی جهت خواندن از ورودی:

```
.ORIG X3000
JSR     FREAD          ;GO TO FREAD AND STORE PC + 1 IN R7
HALT
FREAD   IN              ;GET CHARACTER
        ADD      R2,R0,#0 ;R2 = ENTERED NUMBER
        RET              ;GO TO ADDRESS THAT STORED IN R7
        .END
```

موفق باشید