به نام خدا

طراحی کامپیوتری سیستم‌های دیجیتال - پاییز ۱۴۰۳

پروژه چهارم : **طراحی و پیاده‌سازی واحد پردازشی پایه (Processing Element) شتاب‌دهنده[1] Eyeriss**

طراحان: محمدحسین نیکخواه - سیدصدرا قوامی

هدف پروژه :

در این تمرین قصد داریم نمونه اولیه‌ای از ساختار واحدهای پردازش‌کننده شتاب‌دهنده Eyeriss را طراحی کنیم. در پیاده‌سازی این تمرین، ماژول بافر پیاده‌سازی شده در تمرین قبلی مورد نیاز است.

مقدمه:

پیش‌تر با شبکه‌های عصبی پیچشی، و شتاب‌دهنده Eyeriss آشنا شده‌اید. در ابتدا، معماری این شتاب‌دهنده و روند محاسبات به تفصیل توضیح داده می‌شود. در ادامه، جزئیات مربوط به هر بخش واحد پردازشی که نیازمند پیاده‌سازی است معرفی شده و نکات آن ذکر می‌گردد.

معماری Eyeriss:

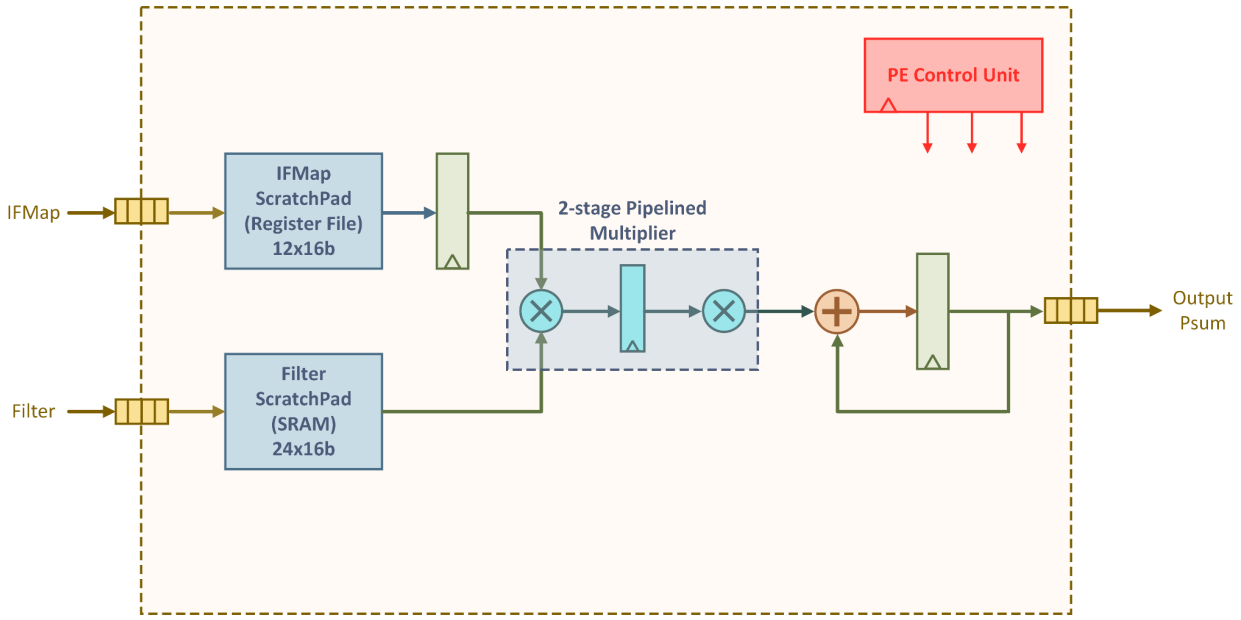
ساختار کلی Eyeriss، به صورت یک آرایه 12*14 از واحدهای پردازشی است، که از طریق ساختاری به نام شبکه روی تراشه[2] به بافر سراسری[3] یا واحدهای همسایه متصل می‌شوند.

هر کدام از واحدهای پردازش‌کننده، توانایی انجام کانولوشن یک بعدی را دارد. به عبارتی دیگر، این واحد با دریافت یک ماتریس تک سطری فیلتر، و یک ماتریس تک سطری ورودی، نتیجه کانولوشن را در چند مرحله ارائه می‌دهد. کانولوشن دو بعدی و سه بعدی را می‌توان مجموعه‌ای از کانولوشن‌های یک بعدی در نظر گرفت. به همین دلیل، با داشتن یک واحد پردازشی، توانایی انجام این کانولوشن ها نیز میسر می‌باشد. اما برای افزایش سرعت محاسبات، و کاهش بار پردازشی، از مجموعه‌ای از این واحدها برای انجام عملیات کلی در شتاب‌دهنده Eyeriss استفاده شده است.

[1] Accelerator
[2] Network-On-Chip
[3] Global Buffer

ساختار واحدهای پردازشی (PE):

تصویر زیر، معماری داخلی PE را نشان می‌دهد. به طور کلی در این ساختار، بافرهای ورودی و خروجی وظیفه handshaking و انتقال داده به و یا دریافت آن از خارج PE را دارند و قسمت داخلی، داده ها را از داخل این بافرها دریافت می‌کند. بافرهای IFMap و Filter(مشخص شده با زنگ زرد)، به ترتیب وظیفه دریافت داده‌های ماتریس ورودی و فیلتر را از خارج PE بر عهده دارند. در ادامه این داده‌ها به حافظه‌های ScratchPad متناظر خود انتقال یافته و در آن ذخیره می‌شوند. توضیحات مربوط به حافظه ScratchPad و نحوه پیاده‌سازی آن در بخش بعدی توضیح داده شده‌است.

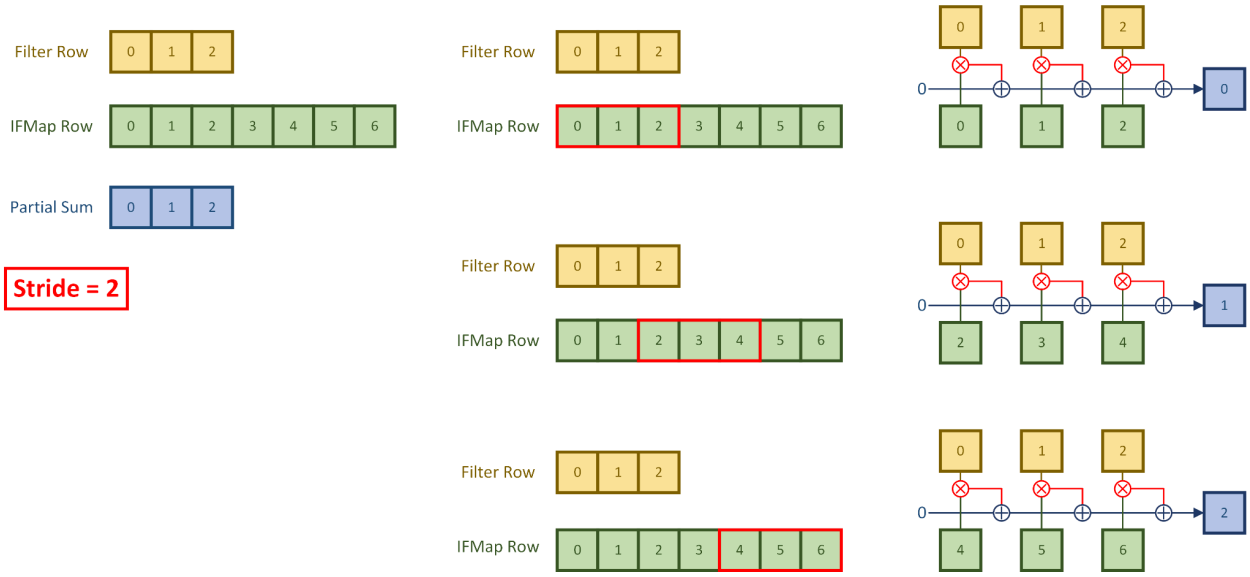


پس از ذخیره شدن داده‌ها، هر کدام از درایه‌های متناظر که می‌بایست در هم ضرب شوند، از روی حافظه‌ها خوانده شده، وارد pipeline می‌شوند تا ضرب و جمع آنها انجام شود. در نهایت، پس از انجام تعداد مشخصی ضرب و جمع و محاسبه یکی از درایه‌های خروجی، این داده به بافر خروجی (output Psum) منتقل می‌شود تا در حافظه سراسری نوشته شده یا برای انجام ادامه محاسبات به واحد دیگری منتقل شود[4]. در ادامه، نحوه محاسبات کانولوشن یک بعدی شرح داده شده‌است.

[4] علت آنکه خروجی واحد، Psum یا Partial-Sum نامیده شده بدین جهت است که در صورتی که محاسبات برای کانولوشن دو یا سه بعدی باشد، خروجی بدست آمده تنها بخشی از محاسبات ضرب و جمع برای درایه متناظر است، و می بایست سطرهای دیگر از ماتریس های ورودی و فیلتر نیز در یکدیگر ضرب شده و نتیجه آن با نتیجه حاصل از PE جمع گردد. لذا به گونه‌ای محاسبات انجام شده برای یک درایه خروجی در PE، جزئی از محاسبات کامل است.

مراحل انجام کانولوشن یک بعدی:

شکل زیر ترتیب و نحوه انجام محاسبات مربوط به یک کانولوشن تک بعدی را نمایش می‌دهد. برای این کار باید پنجره‌ای به اندازه طول فیلتر روی ردیف ورودی قرار دهیم. سپس درایه‌های مشخص شده توسط پنجره را در درایه متناظر آن در فیلتر ضرب کرده و نتایج را با یکدیگر جمع می‌کنیم. در نهایت به این ترتیب درایه اول مربوط به Partial sum محاسبه می‌شود. در اینجا مفهومی تعریف می‌شود تحت عنوان Stride که مشخص می‌کند در هر مرحله پنجره قرار داده شده بر روی ورودی چند خانه به جلو حرکت کند. در مثال زیر میزان Stride برابر 2 است که موجب شده پنجره به اندازه دو خانه به جلو بلغزد. به همین ترتیب درایه‌های دوم و سوم خروجی نیز محاسبه می‌شوند.



پیاده‌سازی:

1. حافظه‌های ScratchPad :

SPM، یا ScratchPad Memory، نوعی حافظه داخلی سریع است که برای ذخیره و دریافت سریع داده‌های موقت استفاده می‌شود. در بسیاری از موارد، این حافظه جایگزین ساده شده‌ای از Cache می‌باشد، با این تفاوت که ساختار ساده تری داشته، و زمان دسترسی آن ثابت است.

در واحد پردازشی Eyeriss، دو نوع ScratchPad با پیاده‌سازی متفاوت وجود دارد:

1. Register Type ScratchPad:

این نوع حافظه مشابه Register File ، از گروهی از رجیسترها تشکیل می‌شود. ورودی‌های این ماژول شامل آدرس خواندن (raddr)، آدرس نوشتن (waddr)، سیگنال کنترلی نوشتن (wen) داده ورودی برای نوشتن (din) بوده و دارای یک خروجی داده خوانده‌شده (dout) می‌باشد. توجه شود که نوشتن درصورت فعال بودن سیگنال wen، روی لبه بالارونده سیگنال Clock صورت گرفته اما خواندن به صورت آسنکرون صورت می‌گیرد.

- این حافظه را پیاده‌سازی کنید. در ماژول نوشته شده، عرض بیت داده، تعداد خانه‌های حافظه، و عرض بیت آدرس‌دهی حافظه به صورت پارامتر در نظر گرفته شود. از عملکرد صحیح آن اطمینان حاصل کنید.

2. SRAM Type ScratchPad :

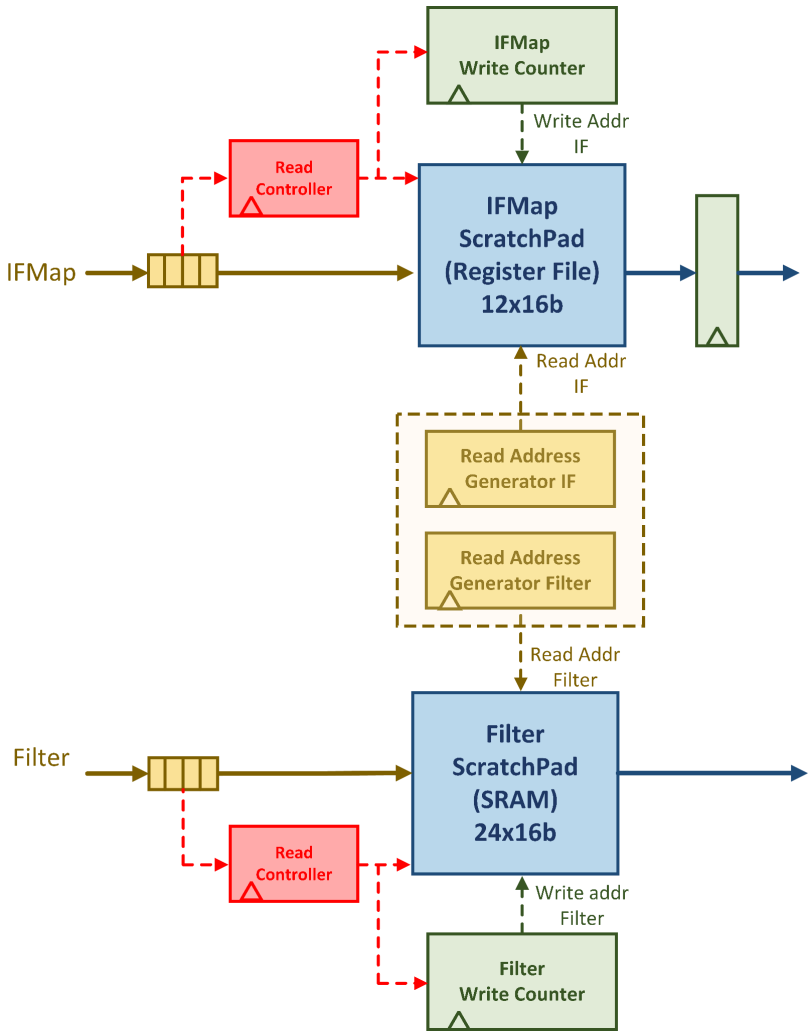
این نوع حافظه به صورت SRAM ساخته می‌شود. ورودی‌های این ماژول شامل فعال ساز ماژول (chip_en)، دستور نوشتن (wen)، دستور خواندن (ren)، آدرس خواندن (raddr)، آدرس نوشتن (waddr)، داده ورودی برای نوشتن (din)، و داده خروجی خوانده‌شده (dout) می‌باشد. در صورت فعال بودن ماژول (chip_en = 1) دستورهای خواندن و نوشتن مستقل از یکدیگر بررسی می‌شوند. در صورتی که دستور نوشتن داشته باشیم (wen = 1)، داده ورودی روی حافظه نوشته می‌شود. همچنین در صورتی که دستور خواندن داشته باشیم (ren = 1)، داده متناظر با آدرس raddr، روی خروجی قرار می‌گیرد. توجه شود در این نوع ScratchPad، خواندن نیز مثل نوشتن به صورت سنکرون رخ می‌دهد و داده روی لبه بالا رونده Clock در دسترس است.

- این حافظه را پیاده‌سازی کنید. مشابه قسمت قبل، پارامترهای عرض بیت داده، تعداد خانه‌های حافظه، و عرض بیت آدرس‌دهی حافظه را در نظر بگیرید. از عملکرد صحیح این ماژول اطمینان حاصل کنید.

2. کنترل‌کننده خواندن از بافرها (Buffer Read Controller)

برای اینکه به صورت خودکار و مرتبا، داده‌های موجود در بافر وارد ScratchPad ها شوند، نیازمند یک واحد کنترلی هستیم تا سیگنال‌های کنترلی مربوط به خواندن از بافر، و نوشتن در ScratchPad متناظر آن را مدیریت کند. به همین منظور، نیازمند پیاده‌سازی Buffer Read Controller هستیم. این ماژول، در صورتی که اجازه نوشتن در ScratchPad وجود داشته باشد، و در عین حال داده‌ای در FIFO موجود باشد، دستورهای کنترلی را به

گونه‌ای فعال می‌کند تا داده از بافر، به حافظه ScratchPad منتقل گردد. تصویر زیر، ارتباط این واحدهای کنترلری را با بافرها و ScratchPad ها نشان می‌دهد:



- ابتدا یک شمارنده[5] برای شمارش و نگهداری آدرس نوشتن در ScratchPad ها بنویسید.
- واحد کنترلی توضیح داده شده را طراحی و اجرا کنید.
- اتصال مناسب هر کدام از بافرها به ScratchPad متناظر و واحد کنترلی خواندن از هر کدام از این بافرها را متناسب با شکل برقرار کنید.

نکته: توجه شود که عرض بیت داده در بافرهای IFMap (عرض : IFMAP_BUFFER_WIDTH)، دو بیت بیشتر از عرض بیت داده در حافظه ScratchPad (عرض : IFMAP_SPAD_WIDTH) تعیین شود. دو بیت انتهایی هر داده ارسالی به بافر، ماهیت کنترلی دارند و جزو داده نیستند. به این نکته در تعریف ماژول‌ها و اتصال آنها دقت شود:

[5] Counter

بیت های داده ارسالی برای بافر IFMap

Start_row[0]	End_row[0]	IFMap Data [IFMAP_DATA_WIDTH-1 : 0]

- با انجام تست مناسب، صحت جابجایی داده بین بافر و ScratchPad را بررسی کنید.

3. کنترل‌کننده‌ی نوشتن در بافر خروجی:

برای اینکه پس از اتمام محاسبات، Psum محاسبه شده به بافر خروجی منتقل شود، نیازمند واحدی هستیم تا با بررسی شرایط، داده خروجی را به بافر منتقل کند. این واحد، ورودی done به معنای اتمام محاسبه یک Psum را از واحد کنترلی اصلی دریافت می‌کند و بررسی می‌کند که آیا امکان نوشتن در بافر وجود دارد یا خیر. چنانچه قابلیت نوشتن در بافر خروجی وجود داشته باشد، محتوای رجیستر Psum با فعال‌سازی دستور نوشتن بافر خروجی، در آن نوشته می‌شود. در صورتی که امکان نوشتن وجود نداشته باشد، این واحد سیگنال خروجی stall را به واحد کنترل اصلی می‌دهد، تا از تغییرات و یا نابودی مقدار Psum جلوگیری کند.

- این واحد کنترلی را نیز طراحی و پیاده‌سازی کنید.
- بافر خروجی را به این واحد متصل کنید و با نوشتن تست، از صحت عملکرد این واحد نیز مطمئن شوید و آن را گزارش کنید.

4. تولیدکننده آدرسِ خواندن از ScratchPad ها :

این بخش، روند اصلی الگوریتم محاسبات کانولوشن را تعیین می‌کند. این واحد مشخص می‌کند تا داده‌های ورودی (Input Feature Map) و فیلتر (Filter) به چه ترتیبی می‌بایست در هم ضرب شوند.

می دانیم که داده‌های ورودی به بافرها و نتیجتا داده‌های داخل ScratchPad به ترتیب هستند. برای مثال، ترتیب داده‌های ورودی به بافر IFMap برای سطر i ام به صورت زیر از چپ به راست است:

$a_i[0]$, $a_i[1]$, $a_i[2]$, …, $a_i[N-1]$

همین ترتیب برای فیلترها هم وجود دارد، با این تفاوت که به ازای هر ورودی IFMap، چند فیلتر در ScratchPad ذخیره می‌شود. فرض شود 10 فیلتر داشته باشیم و ابعاد فیلتر را هم 3 در نظر بگیریم. داده ها به ترتیب زیر به بافر فیلتر ارسال می‌شوند:

$c_0[0]$, $c_0[1]$, $c_0[2]$, $c_1[0]$, $c_1[1]$, $c_1[2]$, …, $c_{10}[2]$

که در آن $c_i[j]$ مشخص کننده درایه j ام فیلتر i ام است.

در انجام محاسبات، به ترتیب کانولوشن هر یک از فیلترها روی یک سطر از ورودی محاسبه می‌شود. برای این منظور تمامی درایه های فیلتر در درایه های متناظر از ورودی ضرب شده و نتیجه آن ها با یکدیگر جمع می‌شود. پس از این مرحله، یکی از درایه‌های Psum حاضر شده است و می‌بایست پنجره‌ی فیلتر روی ورودی‌ها به اندازه Stride به جلو حرکت کند. در نتیجه برای شمارش اندیس پنجره روی داده‌های IFMap و فیلتر نیازمند یک شمارنده هستیم. از طرفی دیگر پس از هر بار پیمایش کامل پنجره، به اندازه Stride پنجره رو به جلو حرکت می‌کند، که نقطه شروع پنجره ورودی نیز می‌بایست توسط شمارنده دیگری ذخیره شود. با هربار پیمایش کامل یک پنجره روی همه‌ی داده‌های ورودی، محاسبات مربوط به یک فیلتر تمام می‌شود و به همین جهت فیلتر بعدی برای محاسبه باید خوانده شود، که نگهداری اشاره‌گر به اولین درایه از فیلتر مدنظر نیز به کمک شمارنده/رجیستر قابل پیاده‌سازی است.

- به کمک توضیحات داده شده، ماژول‌های تولیدکننده آدرس خواندن از هر یک از ScratchPad ها را پیاده‌سازی کنید.
- در این پیاده‌سازی، عرض فیلتر (filter_size) و stride به صورت ورودی در اختیار شما قرار داده می شود. دو پارامتر در نظر بگیرید که عرض بیت این دو ورودی را مشخص می‌کند. توجه شود که این دو ورودی در تعیین آدرس‌های فیلتر و IFMap تاثیرگذار هستند.

5. واحد کنترل اصلی:

این بخش، روند اصلی کنترل محاسبات، راه‌اندازی pipeline و تشخیص شروع و اتمام محاسبات را برعهده دارد. واحد کنترل پس از دریافت یک پالس روی ورودی Start، منتظر می‌شود تا اولین درایه ورودی و حداقل یک درایه از یک فیلتر وارد ScratchPad ها شده تا pipeline را فعال کند و محاسبات انجام شود. پس از این، به ترتیب کانولوشن هر یک از فیلترها روی یک سطر از ورودی محاسبه می‌شود. پس از این، مادامی که محاسبات یک سطر به اتمام نرسیده، pipeline را فعال می‌کند، درصورت نیاز دستورهای clear برای رجیسترها و یا شمارنده‌ها را در زمان مناسب ارسال می‌کند، و پس از اتمام محاسبات هر یک از داده‌های Psum، سیگنال done را برای کنترلر نوشتن بافر خروجی ارسال می‌کند.

6. تکمیل مسیرداده :

حال با قرار دادن رجیسترها، ضرب کننده و جمع‌کننده مسیرداده را تکمیل کنید.

- برای شبیه‌سازی فعالیت ضرب‌کننده دو مرحله‌ای، از یک ضرب کننده ساده و یک رجیستر استفاده کنید.
- در اتصال‌ها، به تطبیق عرض بیت‌ها دقت کنید.

- توجه شود که عرض بیت تمامی ماژول‌های نوشته‌شده در این بخش مانند ضرب‌کننده یا رجیستر، باید پارامتری باشند.

سایر نکات

- - انجام این تمرین به صورت گروه های دونفره در دو فاز خواهد بود:

1. در فاز اول `controller` و `datapath` را طراحی کرده و در موعد تعیین شده برای فاز اول داخل سایت بارگذاری کنید.

2. در فاز دوم `controller` و `datapath` طراحی شده در فاز اول را در برنامه Modelsim و با زبان verilog پیاده سازی کرده و در موعد معین برای فاز دوم در داخل سایت بارگذاری کنید.

- برای فاز دوم تمرین لازم است فایل های `Testbench` و `HDL` خود را مطابق توضیح داده شده در `trunk` در `subdirectory` های `trunk/doc` بارگذاری کنید. همچنین اطمینان حاصل کنید که با اجرای `trunk/sim/sim_top.tcl` تست پنج شما اجرا می شود. برای اجرای این اسکریپت می توانید از دستور زیر در Modelsim استفاده کنید:

>> do <sim_file>

- فایل ها و گزارش خود را تا قبل از موعد تحویل هر فاز، با نام CAD_HW4_P1_<SID>.zip و CAD_HW4_P2_<SID>.zip به ترتیب در محل های مربوطه در صفحه درس آپلود کنید.

- برای آزمودن کد خودتان در این تمرین تست بنچ‌های مربوطه را خود شما طراحی و پیاده سازی می کنید، اما با توجه به اینکه تمارین بعدی درس مبتنی بر ادامه دادن این تمرین هستند حتما در طراحی و پیاده‌سازی خود به قابلیت مقاومت در برابر تغییر و پارامتری بودن ورودی‌ها و خروجی‌ها توجه لازم را داشته باشید.

- نام گذاری صحیح متغیرها، تمیزی کد و توضیحات و پارامتری بودن ورودی‌های ماژول‌ها می تواند تا حدودی کاستی‌های کد را در بخش‌های دیگر جبران کند.

- - هدف این تمرین یادگیری شماست! در صورت کشف تقلب، مطابق با قوانین درس برخورد خواهد شد.

موفق باشید