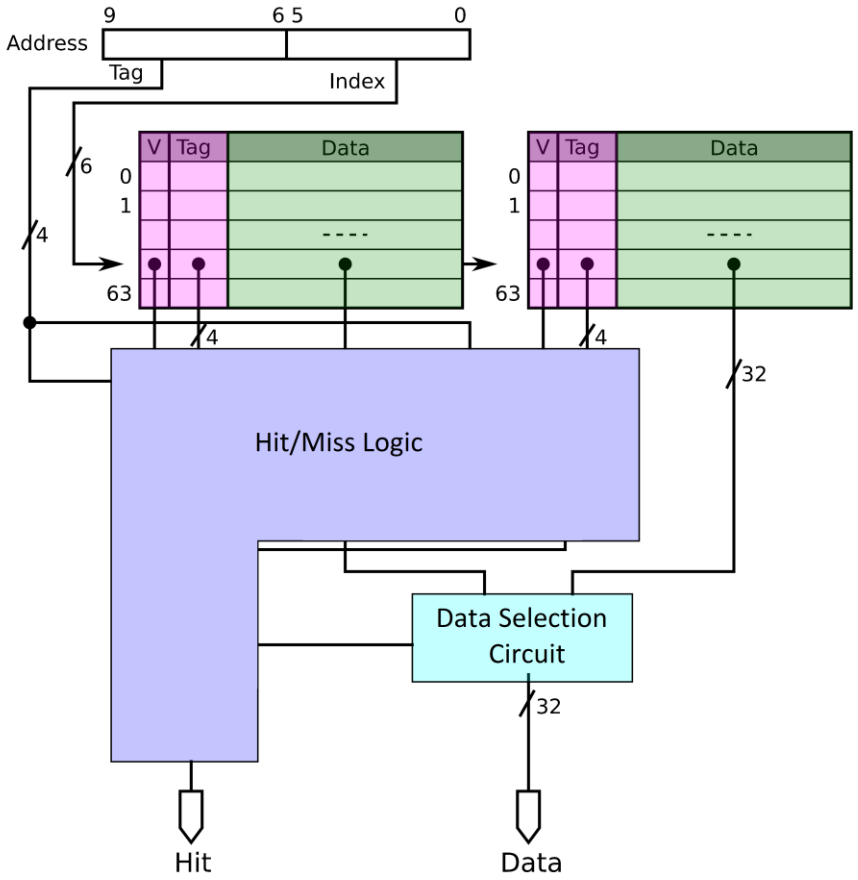
پروژه‌ی میان‌ترم معماری کامپیوتر

نیم‌سال دوم ۹۴-۹۵

مهلت تحویل ۹۴/۱/۱۸

حافظه‌ی نهان

شکل زیر یک حافظه‌ی نهان انجمنی ۲ مجموعه‌ای[۱] را نشان می‌دهد. در این حافظه هر مجموعه[۲] دارای ۶۴ خط است و هر خط تنها یک کلمه‌ی ۴ بایتی دارد. برای سهولت کار هریک از قسمت‌هایی که به طور جداگانه نیاز به پیاده‌سازی دارند در شکل زیر با رنگ‌های متفاوت نشان‌داده‌شده‌اند. نحوه‌ی پیاده‌سازی این قسمت‌ها در زیر توضیح داده خواهد شد.



شکل ۱ - ساختار کلی حافظه‌ی نهان

سوال ۱) توضیح دهید چرا در ساختار بالا نیازی به offset وجود ندارد.

---

[۱] 2-way set associative cache

[۲] way

همان‌طور که در درس خوانده‌اید، قسمت index برای انتخاب سطرهای حافظه‌ی نهان استفاده می‌شود. پس از اینکه سطر موردنظر انتخاب شد، قسمت tag داده‌ی ورودی با قسمت tag داده‌ی موجود در خطوط انتخاب شده از حافظه‌ی نهان مقایسه می‌شود و در صورتی که با یکدیگر مطابقت داشتند، حافظه‌ی نهان اعلام hit می‌کند و در غیراینصورت miss اعلام می شود. در صورتی که hit اعلام شود، داده‌ی در ست تو سط یک مدار انتخاب شده و خروجی داده می‌شود.

به منظور آسان‌سازی ساخت این حافظه‌ی نهان، آن را به تعدادی بخش اصلی تقسیم کرده‌ایم که عبارتند از:

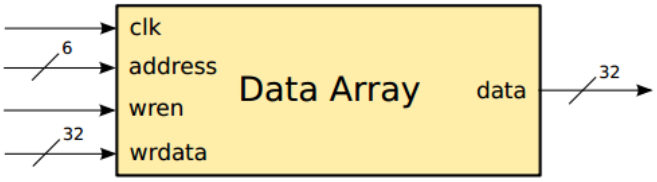
۱- **آرایه‌ی داده**،[۳] که مسئولیت نگهداری داده‌های هر مجموعه[۴] را برعهده دارد.

۲- **آرایه‌ی Tag-Valid**، که قسمت Valid و قسمت Tag هر داده‌ی موجود در حافظه‌ی نهان را نگهداری می‌کند.

۳- **منطق تطابق**،[۵] مداری است که بررسی می‌کند که آیا تطابق[۶] رخ داده یا خیر؛

۴- **آرایه‌ی LRU**، مداری‌است که تصمیم می‌گیرد کدام خط از حافظه‌ی نهان باید برای جایگزینی انتخاب شود.

۱. آرایه‌ی داده



شکل ۲ - Data Array

هدف از این آرایه جا دادن ۶۴ خط داده‌ی هریک از مجموعه‌های حافظه‌ی نهان است (توجه داشته باشد که هر way یا مجموعه تنها ۶۴ خط دارد).

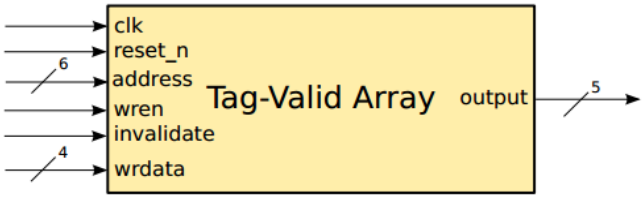
---

۳ Data Array
۴ way
۵ Hit Logic
۶ Hit

در این آرایه سینگال **address**، آدرس خطی از حافظه‌ی نهان که باید نوشته یا خوانده شود را مشخص می‌کند. داده‌ی خوانده شده از طریق سیگنال **data** خروجی داده می‌شود. فرایند نوشتن، یک فرایند همگام[۷]است؛ هرگاه سینگال **wren** فعال باشد داده‌ی ورودی از سیگنال **wrdata** در آدرس تعیین شده نوشته می‌شود.

این ماژول را در فایل data_array.vhd پیاده‌سازی کنید.

۲. آرایه‌ی Tag-Valid



شکل ۳ - آرایه‌ی Tag-Valid

هدف از این آرایه ذخیره‌سازی بیت‌های Tag و Valid مربوط به داده‌های موجود در Cache است. این آرایه دارای ۶۴ خط است و هر خط آن از دارای دو قسمت است؛ ۴ بیت برای Tag و یک بیت برای Valid.

فرایند خواندن و نو شتن در آن م شابه آرایه‌ی داده ا ست، با این تفاوت که در اینجا **wrdata** تنها دربردارنده‌ی ۴ بیت Tag ورودی است و سیگنال **invalidate** نیز برای تغییر بیت Valid در این آرایه به کار می‌رود. به این صورت که اگر مقدار این سیگنال برابر یک با شد، بیت Valid در خط تعیین شده تو سط سینگال **Address** برابر صفر قرار داده می‌شود.

این ماژول را در فایل Tag-Valid_araay.vhd پیاده‌سازی کنید.

۳. منطق مطابقت

کاری که این ماژول انجام می‌دهد این است که تعیین می‌کند آیا مطابقت رخ داده یا خیر. این ماژول همچنین این وظیفه را هم دارد که در صورت رخ دادن یک مطابقت، تعیین کند این مطابقت در کدامیک از مجموعه‌ها[۸] رخ داده‌است.

---

[۷] Synchronous
[۸] way

شکل ۴ - منطق مطابقت

سیگنال **w0** و **w1** به ترتیب همان بیت‌های tag و valid مجموعه‌ی اول و مجموعه‌ی دوم هستند. سیگنال **tag** همان تگی است که به عنوان ورودی به حافظه‌ی نهان هم می‌دهیم (همان تگی که در حافظه‌ی نهان به‌دنبال آن می‌گردیم). برای مثال فرض کنیم که میخواهیم ببینیم داده‌ی که تگ آن ۴ بوده و index آن ۲۰ است در حافظه‌ی نهان وجود دارد یا نه، یعنی Tag = 0100. در این صورت به خط بیستم حافظه‌ی نهان رجوع می‌کنیم، فرض کنیم در خط بیستم حافظه‌ی نهان، در way-0 داده‌ای با تگ ۷ و valid = 1 و در way_1 داده‌ای با تگ ۴ و valid = 1 موجود است. در این‌صورت سیگنال w0 برابر است با $\underbrace{1}_{valid=1}\ \underbrace{0111}_{Tag=7}$ و سیگنال w1 نیز برابر ا ست با $\underbrace{1}_{Valid=1}\ \underbrace{0100}_{Tag=4}$ در این صورت چون داده‌ی موجود در way-1 همان داده‌ای ا ست که به دنبالش می‌گردیم، خواهیم داشت w0_valid = 0 و w1_valid = 1 و hit = 1.

سیگنال **w0_valid** در صورتی فعال می شود که داده‌ی ما در مجموعه‌ی اول[۹] با شد. سیگنال **w1_valid** نیز به طور مشابه تعیین می‌شود.

این ماژول را باید در فایل miss_hit_logic.vhd پیاده‌سازی کنید.

۴. سیاست جایگزینی حافظه‌ی نهان[۱۰]

همانطور که در درس خوانده‌اید، یک سیاست جایگزینی تعیین می‌کند که در صورتی که نیاز باشد داده‌ای از حافظه‌ی اصلی به حافظه‌ی نهان منتقل شود و حافظه‌ی نهان پر باشد، این داده باید جایگزین کدامیک از داده‌ها شود. در این پروژه برای سادگی کار از سیاست نامتاخر ترین بروزرسانی[۱۱] استفاده می شود. این سیاست، سطری را که برای مدت زمان زیادی بروزرسانی نشده برای جایگزینی انتخاب می‌کند.

این قطعه باید همواره بداند که کدام خط از حافظه‌ی نهان بروزرسانی می‌شود و در نتیجه هرگاه سطری از حافظه‌ی نهان دچار تغییر می شود، این قطعه نیز باید دوباره برر سی کند و ببیند کدام سطر برای طولانی ترین

---

[۹] Way 0
[۱۰] Cache Replacement Policy
[۱۱] Least Recently Updated

مدت بروزرسانی نشده است. خروجی این قطعه تنها یک بیت است، که تعیین می‌کند آیا **w1** باید برای جایگزینی انتخاب شود یا خیر.

این قطعه را در فایل lru_array.vhd پیاده‌سازی کنید.

۵. کنترل‌کننده‌ی حافظه‌ی نهان

می‌دانیم که تمام قطعاتی که تا کنون تعریف کرده‌ایم دارای سیگنال‌های کنترلی‌ای هستند، این سیگنال‌ها باید توسط یک واحد تعیین شوند. این واحد که کنترل‌کننده‌ی حافظه‌ی نهان[۱۲]نام دارد، وظیفه دارد در صورت رخ‌دادن حوادثی خاص؛ سیگنال‌های کنترلی مناسب را به قطعات مختلف ارسال کند. در ابتدای کار این واحد کنترل منتظر یک درخواست خواندن یا نوشتن از طرف واحد پردازنده‌ی مرکزی[۱۳]می‌ماند. در صورتی که درخواست نوشتن را دریافت کند تنها کاری که باید انجام شود این است که آن خط از حافظه‌ی نهان را نامعتبر[۱۴] کند و همچنین سیگنال write حافظه‌ی اصلی را نیز فعال کند. توجه داشته باشید که در طی یک فرایند نوشتن، داده‌ی موجود در حافظه‌ی نهان تغییری نمی‌کند، فقط بیت valid آن تغییر کرده و نامعتبر می‌شود. برای سادگی در پیاده‌سازی، در یک فرایند نوشتن؛ یک set از هر دو مجموعه را به طور کل نامعتبر می‌کنیم. پس وقتی می‌خواهیم خطی را در حافظه‌ی اصلی بنویسیم باید دو خط را در حافظه‌ی اصلی و حافظه‌ی نهان نامعتبر کنیم. برای مثال فرض کنید به عنوان ورودی بخواهیم داده‌ای را در آدرس 25 یا همان $(0000\,011001)_2$ حافظه‌ی اصلی بنویسیم. کاری که باید انجام دهیم این است که در خانه‌ی ۲۵ ام آرایه‌ی tag-valid، بیت valid هر دو مجموعه‌های آن را برابر ۰ قرار دهیم.

در صورتی که درخواست خواندن داده شود، در صورتی که خط موردنظر در حافظه‌ی نهان موجود باشد (یعنی هم index و هم tag داده‌ای که دنبالش می‌گردیم با یکی از داده‌های موجود در RAM مطابقت کند)؛ باید بیت Valid این خانه را بررسی کنیم، در صورتی که بیت Valid آن یک بود، کنترل‌کننده به وضعیت[۱۵]اولیه‌ی خود بازمی‌گردد و منتظر یک درخواست خواندن یا نوشتن دیگر می‌ماند. ولی در صورتی که بیت Valid آن یک نبود با آن مانند یک Miss برخورد می‌شود. اگر Miss رخ دهد، کنترل‌کننده سیگنال خواندن از RAM را فعال می‌کند و منتظر می‌ماند تا داده‌ی دریافت شده از RAM به کنترل‌کننده برسد (نحوه‌ی تشخیص داده‌ی دریافتی به این صورت است که این ماژول دارای یک سیگنال ورودی به نام ram_ready است که هرگاه داده از RAM

---

[۱۲] Cache Controller
[۱۳] Central Processing Unit - CPU
[۱۴] invalidate
[۱۵] state

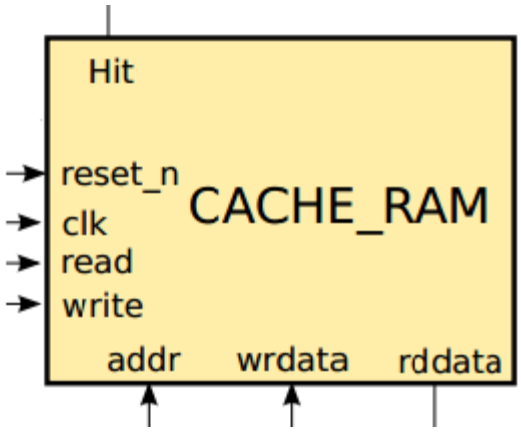
رسیده باشد این سیگنال فعال می شود). بعد از اینکه این داده دریافت شد، در حافظه‌ی نهان نوشته می شود و کنترل‌کننده نیز به وضعیت اول خود بازمی‌گردد و منتظر یک درخواست خواندن یا نوشتن می‌ماند.

۶. حافظه‌ی اصلی[۱۶]

یک ماژول به نام RAM در نظر بگیرید که مانند آرایه‌ی داده با شد. اندازه‌ی این حافظه‌ی اصلی را از روی مقدار Tag و Index بیابید. حافظه‌ی اصلی دارای یک سیگنال ورودی به نام rw است که اگر فعال باشد عمل نوشتن انجام می‌شود و در غیراینصورت عمل خواندن انجام می‌شود. برای پیاده‌سازی این حافظه‌ی اصلی از نحوه‌ی طراحی ماژول‌های آرایه‌ی داده کمک بگیرید.

این ماژول یک سیگنال خروجی به نام Data_ready نیز دارد که هرگاه داده‌ی خوانده شده از حافظه‌ی اصلی آماده با شد فعال می شود. علت وجود این سیگنال همان است که در بخش کنترل‌کننده‌ی حافظه‌ی نهان ذکر شد. ورودی‌های این حافظه‌ی اصلی عبارتند از: Data_in (32 bit), Address(?), rw, clk و خروجی‌ها آن هم همانطور که ذکر شد عبارتند از: Data_out(32 bit), Data_ready.

شما باید تمامی ماژول‌های گفته شده را طراحی کرده و آن‌ها را در قالب یک ماژول کلی به نام Cache_RAM قرار دهید که به شکل زیر است:



که در آن سیگنال‌های reset_n, clk, read, write, addr(10 bits), wrdata(32 bits) ورودی‌های مدار هستند و سیگنال Hit و rddata(32 bits) سیگنال‌های خروجی مدار را تشکیل می‌دهند.

[۱۶] Main Memory - RAM

طرز کار کلی مدار پروژه اینگونه است که یک داده‌ی ۳۲ بیتی با آدرسی ۱۰ بیتی به ورودی داده می‌شود. در صورتی که سیگنال read فعال باشد، می‌خواهیم داده‌ی موجود در آدرس تعیین شده در addr را بخوانیم. ابتدا به cache مراجعه می‌کنیم و اگر داده وجود داشت آن‌را به عنوان خروجی به rddata می‌دهیم و سیگنال hit را نیز توسط ماژول منطق مطابقت فعال می‌کنیم و منتظر داده‌ی بعدی می‌مانیم. در صورتی که داده‌ی خواسته شده وجود نداشت، کنترلر حافظه‌ی نهان به رم درخواست خواندن می‌دهد و داده‌ی خوانده شده را دریافت کرده و در حافظه‌ی نهان می‌نویسد. سپس آن را به خروجی rddata می‌دهد. اگر حافظه‌ی نهان پر بود، ماژول lru_array تصمیم می‌گیرد که کدامیک از way ها باید جایگزین شوند.

در صورتی که سیگنال write فعال بود، ابتدا داده‌ی موجود در آدرس خواسته شده از RAM را نامعتبر می‌کنیم و سپس همین‌کار را با داده‌ی موجود با این آدرس در حافظه‌ی نهان نیز انجام می‌دهیم. سپس داده‌ی ورودی wrdata را در محل تعیین شده از RAM می‌نویسیم. توجه داشته باشید که در این حالت باید به حافظه‌ی نهان مراجعه کنیم و در صورتی که آدرسی که میخواهیم در RAM بنویسیم در حافظه‌ی نهان موجود باشد، آن را در حافظه‌ی نهان نامعتبر می‌کنیم (یعنی بیت valid آن را صفر می‌کنیم). اما در صورتی که در حافظه‌ی نهان وجود نداشت کاری با آن نداریم.

کارهایی که در این پروژه انجام خواهید داد عبارتند از:

۱. ماژول‌های خواسته شده را به زبان VHDL بنویسید.

۲. برای قسمت کنترل‌کننده‌ی حافظه‌ی نهان یک FSM طراحی کرده و سپس مدار را با استفاده از این FSM طراحی کنید.

۳. ماژولی به نام Cache بسازید و تمامی قسمت‌های ۱ تا ۴ را در آن قرار دهید.

۴. ماژولی به نام RAM بسازید و آن را مطابق توضیحات پیاده‌سازی کنید.

۵. ماژول RAM_Cache را مطابق توضیحات بسازید و در آن RAM و Cache و Controller را قرار دهید.

۶. نحوه‌ی اتصال ماژول‌های خود به یکدیگر را به طور گرافیکی نمایش دهید (برای مثال در ماژول Cache، شکلی بکشید و در آن نحوه‌ی متصل شدن ماژول data_array به lru_array را مشخص کنید). توجه داشته باشد که در رسم این شکل در صورتی که لازم بود گیت‌های منطقی اضافه کنید آن‌ها را هم در شکل نشان دهید. برای مثال ممکن است لازم باشد برای ساخت ورودی rw، یک سیگنال به سیگنال دیگری AND شود.

۷. ماژول کلی RAM_Cache را با ا ستفاده از یک سری ورودی آزمایش کنید و شکل موج آن را در گزارش پروژه‌ی خود قرار دهید. توجه داشته باشید که برای آزمایش کردن ورودی‌های مختلف نیازی به testbench نو شتن نی ست و می‌توانید مقادیر را به طوری دستی force کنید.

برای این پروژه باید یک گزارش تهیه کنید. گزارش تهیه شده باید به صورت تایپ شده تحویل داده شود. در این گزارش باید نحوه‌ی ساخت هریک از ماژول‌های ذکر شده تو ضیح داده شود؛ برای مثال اگر قرار ا ست در ماژول *Hit_logic* از یک مدار ترکیبی ا ستفاده کنید، باید جدول کارنوی آن را بک شید و در گزارش قرار دهید. پس از طراحی هریک از ماژول‌های خود، باید ماژول مورد نظر را به تنهایی تست کرده (مقداردهی دستی) و نتایج آن را در گزارش خود ذکر کنید (برای مثال از شکل موج خروجی عکس گرفته و در گزارش خود وارد کنید). همچنین FSM مربوط به مدار کنترلی و منطق پشت مدار *LRU* را نیز در گزارش خود توضیح دهید.

بخش امتیازی

برای بخش امتیازی دو ق سمت درنظر گرفته شده‌ا ست. در صورت تمایل به انجام ق سمت امتیازی **فقط یکی از دو ق سمت زیر را انتخاب کنید.** در صورتی که هردو بخش را انجام دهید نمره‌ی شما با حالتی که فقط یکی از آن‌ها را انجام دهید فرقی نخواهد داشت.

بخش امتیازی ۱: پیاده‌سازی پروژه روی بورد FPGA

ماژول‌های ساخته شده را روی بوردهای *Altera DE2* پیاده سازی کنید. برای پیاده سازی این ماژول‌ها روی بورد، به یک مدار کلاک‌ساز نیز نیاز دارید. گزارش کاملی از نحوه‌ی پیاده‌سازی و انجام این قسمت را به طور جداگانه‌ای تحت عنوان bonus_report.pdf بنویسید.

بخش امتیازی ۲- طراحی واحد جایگزین‌کننده‌ی 2Q

در این پروژه ماژولی به نام *lru_array* وجود داشت، وظیفه‌ی این ماژول همانطور که گفته شد انتخاب یک خانه از حافظه‌ی نهان برای جایگزینی است. در این قسمت علاوه بر ماژولی که در صورت سوال گفته شده، ماژولی به نام 2Q پیاده سازی کنید. نحوه‌ی کار سیاست *2Q* این‌گونه است که دارای دو صف *FIFO* و *LRU* است. وقتی که بلوکی برای اولین بار درخواست می شود(زمانی که هنوز در حافظه‌ی نهان نیست)، آن را در بافر ویژه که از نوع *FIFO* است قرار می‌دهیم. نام این بافر *A1* است. در صورتی که یک آدرس موجود در *A1* درخواست داده شود، این صفحه را به بافری به نام *Am* انتقال می‌دهیم. این بافر از نوع *LRU* است. در صورتی که آدرسی بیش از یک زمان معین (این زمان را اینگونه تعیین کنید که برای مثال اگر از ۲۰ درخواست داده شده، هیچ‌کدام این آدرس را درخواست ندهند)آن آدرس از بافر *A1* حذف می‌شود.

ابتدا ورودی‌ها و خروجی‌های لازم این ماژول را تعیین کنید و سپس آن را در *VHDL* پیاده‌سازی کنید.

*** برای ایجاد سهولت در طراحی و ساخت ماژول‌های گفته‌شده، تعدادی ماژول به زبان *VHDL* در پوشه‌ی پروژه قرار دارند. این فایل‌ها در پوشه‌ای به نام *examples* قرار دارند.

نکات مهم

- پروژه باید به صورت انفرادی انجام شود.
- تمام طراحی‌ها باید به زبان VHDL باشد.
- برای شبیه‌سازی می‌توانید از نرم‌افزارهای Modelsim، Xilinx ISE و active HDL استفاده کنید، اما توصیه می‌شود به منظور سهولت کار و یکپارچگی همه‌ی پروژه‌ها؛ از نرم‌افزار Modelsim استفاده کنید.
- کدهای خود را در فایل‌های قرار داده شده در فایل zip ای که پروژه در آن قرار دارد بنویسید. در صورت نیاز به اضافه کردن ماژول‌ها دیگر، تمام این ماژول‌ها را در فایل‌های مربوطه به خود قرار دهید. برای مثال اگر نیاز دارید تا ماژولی جدید طراحی کنید که در کنار data_array کار کند، این ماژول را در فایلی که ماژول data_array در آن ا ستفاده می شود (ماژول پدر) قرار دهید. در نتیجه اگر قرار با شد ماژولی به نام my_module در کنار data_array کار کند، این ماژول باید در فایل cache.vhd قرار گیرد.
- توصیه می‌شود به منظور سهولت کار و همچنین قابلیت بازیابی کدهای خود، از نرم‌افزارهای کنترل نسخه مانند github یا امثال آن استفاده کنید.
- در نهایت تمام فایل‌های پروژه‌ی خود را به همراه گزارش پروژه که به صورت report.pdf ذخیره شده‌است، به صورت یک فایل zip درآورده و آن را به صورت زیر نام‌گذاری کنید:

Mohammad_Rajabi_9231039

- از قراردادن هرگونه فایل دست‌نوشته و اسکن شده در گزارش خود خودداری کنید.

موفق باشید