
دانشکده مهندسی کامپیوتر

**پروژه درس طراحی کامپیوتری سیستمهای دیجیتال**

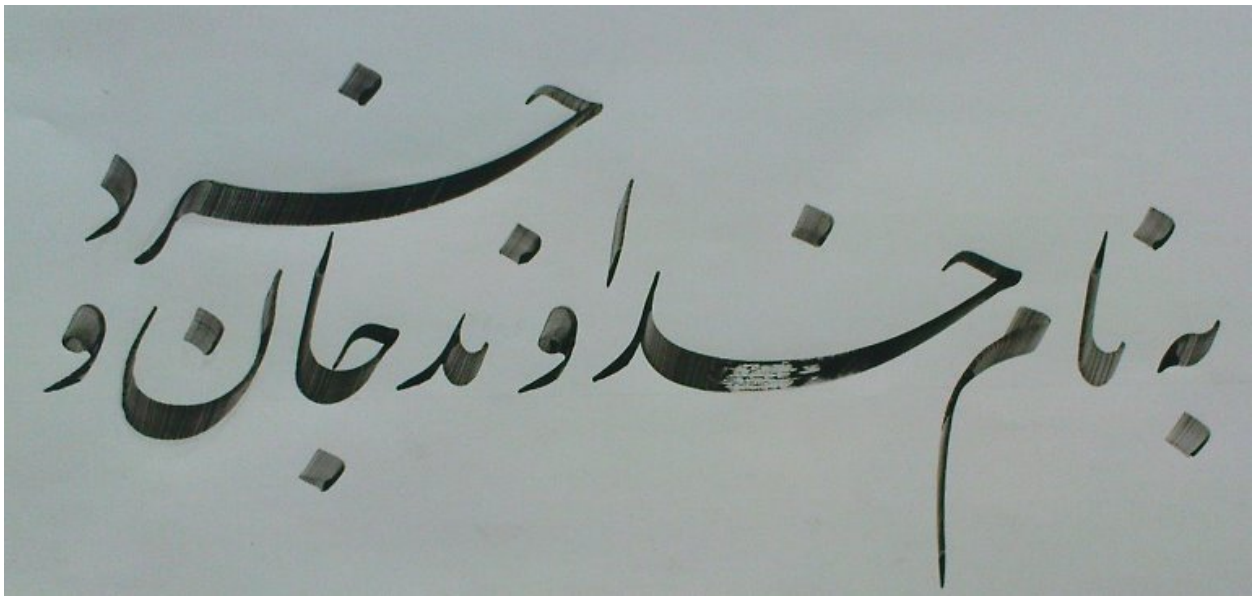
**نیمسال تحصیلی دوم 99**

# طراحی یک واحد پردازشی ضرب دو ماتریس

سید یاسین موسوی، فراز قهرمانی، میلاد سعادت، پیمان حاجی محمد، سینا مظاهری، پرهام چاوشیان

استاد : دکتر فرشاد بهاروند

تیر 1400


به نام خداوند جان و خرد

| تقسیم وظایف تیم | | |
|---|---|---|
| اعضای گروه | وظیفه | درصد مشارکت |
| سید یاسین موسوی(98110351) | داک و کد ماژول ها | 16.66% |
| فراز قهرمانی (98109594) | سنتز و تست | 16.66% |
| میلاد سعادت (98100442) | داک | 16.66% |
| پرهام چاوشیان (98100118) | داک و تست | 16.66% |
| سینا مظاهری (98171159) | کد ماژول ها | 16.66% |
| پیمان حاجی محمد (98170776) | کد ماژول ها | 16.66% |

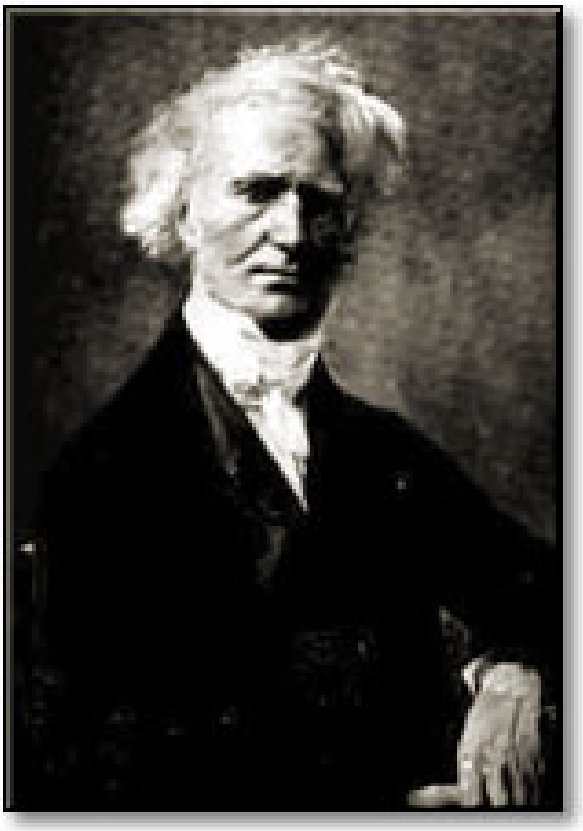
# فهرست مطالب

# فصل اول : مقدمه

## 1-1 چکیده

در این گزارش پروژه قصد داریم نحوه پیاده سازی و نتایج حاصل از سنتز برای نحوه محاسبه ضرب ماتریسی را با استفاده از عملیات Divide and Conquer شرح دهیم. استاندارد مورد استفاده برای عملیات ضرب پروژه IEEE 754 بوده و از دو روش parallel و sequential برای محاسبه نتایج استفاده شده تا هم مساحت کار کمترین باشد و همچنین در کمترین زمان ممکن عملیات انجام شود. برای پیاده سازی این عملیات از استاندارد های معروف استفاده شده و از control unit برای تعیین ورودی و خروجی مآژول های مختلف استفاده شده است. زبان مورد استفاده برای پیاده سازی الگوریتم ها verilog میباشد و از زبان++c برای مدل طلایی استفاده کرده ایم.

## 2-1 تاریخچه



ضرب ماتریسی نوعی از ضرب دودویی است که از دو ماتریس ، یک ماتریس خروجی میدهد، اولین بار ضرب ماتریسی توسط ژاک فیلیپ ماری بنت ( Jacques Philippe Marie Binet)،ریاضیدان فرانسوی در سال 1812 ابداع شد، ضرب ماتریسیی یک ابزار قوی در جبر خطی(linear algebra) است.

امروزه از ضرب ماتریسی به همراه عملیات ممیز شناور (floating-point operation) استفاده می شود تا عملیات های با اهداف بالاتر را اجرا نمود. ما در این گزارش قصد ساخت و پردازش یک ضرب ماتریسی بر اساس ممیز شناور را داریم. البته امروزه از عملیات های دیگری غیر از ممیز شناور برای پردازش ماتریس ها استفاده میکنند به این دلیل که عملیات ممیز شناور برای ضرب و جمع های متوالی بسیار نابهینه است.

## 3-1 نحوه کلی عملکرد

هدف ما ضرب دو ماتریس در همدیگر با استفاده از عملیات ممیز شناور و همچنین کمتر کردن مساحت و افزایش سرعت بود. ما 3 مدل اصلی مختلف ساختیم

- Control unit
- Data path
- memory

ابتدا ماتریس ها را به صورت سطری در حافظه ذخیره می کنیم سپس بر اساس الگوریتمی که در control unit وجود دارد ماتریس ها را به ماتریس های 2 * 1 و 1 *1 و 1*2 و 2*2 کوچکتر تقسیم میکنیم، حال ممکن است گاهی این ماتریس ها طول و یا عرض آن کمتر شود، در اینصورت آنها را با اضافه کردن خانه هایی با عدد 0 تبدیل به ماتریس 2*2 میکنیم.
حال ما این ضرب های کوچکتر را به پروسسور های مختلف میدهیم تا به صورت همزمان ضرب ماتریسی انجام شود. در واقع خواندن و نوشتن از حافظه به شکل Sequential و پردازش به شکل Parallel هست.

## 4-1 پایه ریاضی

### 4-1-1 ضرب ماتریسی

ابتدایی ترین مبانی ریاضیات استفاده شده در این پروژه ضرب ماتریسی است، به این صورت که برای دو ماتریسA که m*n وBکه n*p است داریم:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

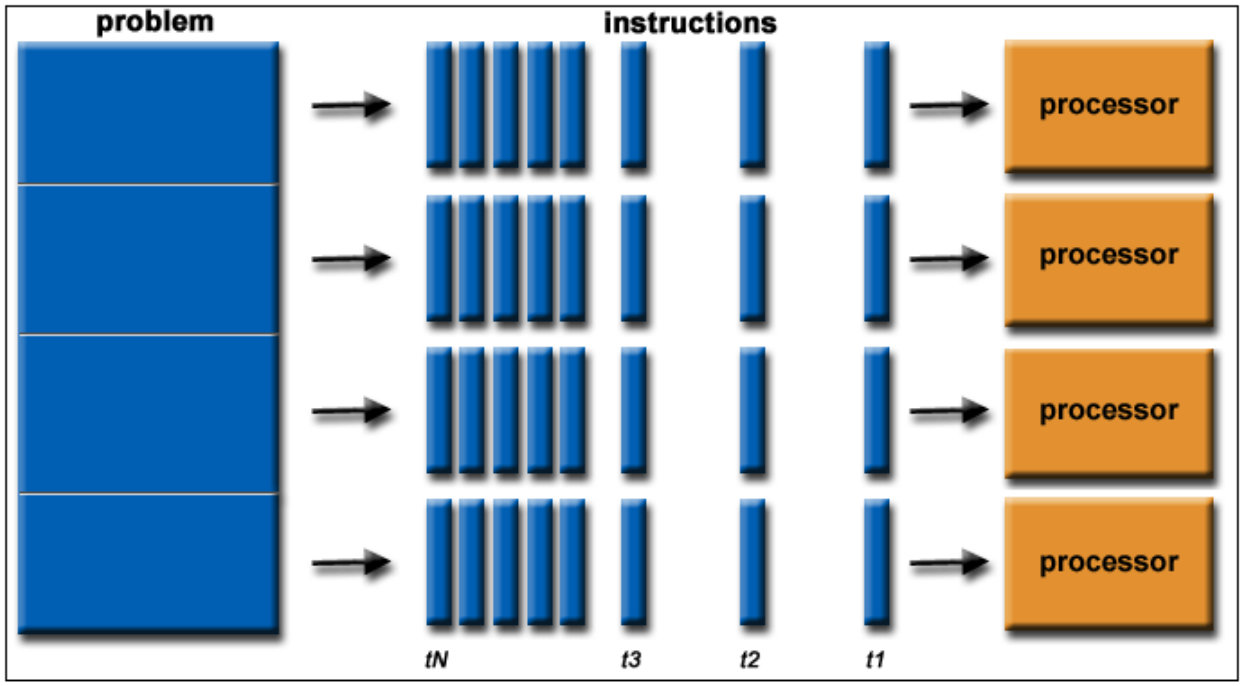
$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

که ماتریس C=AB یک ماتریس m*p میباشد. و درایه خانه iوj آن به صورت زیر محاسبه میشود:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj},$$

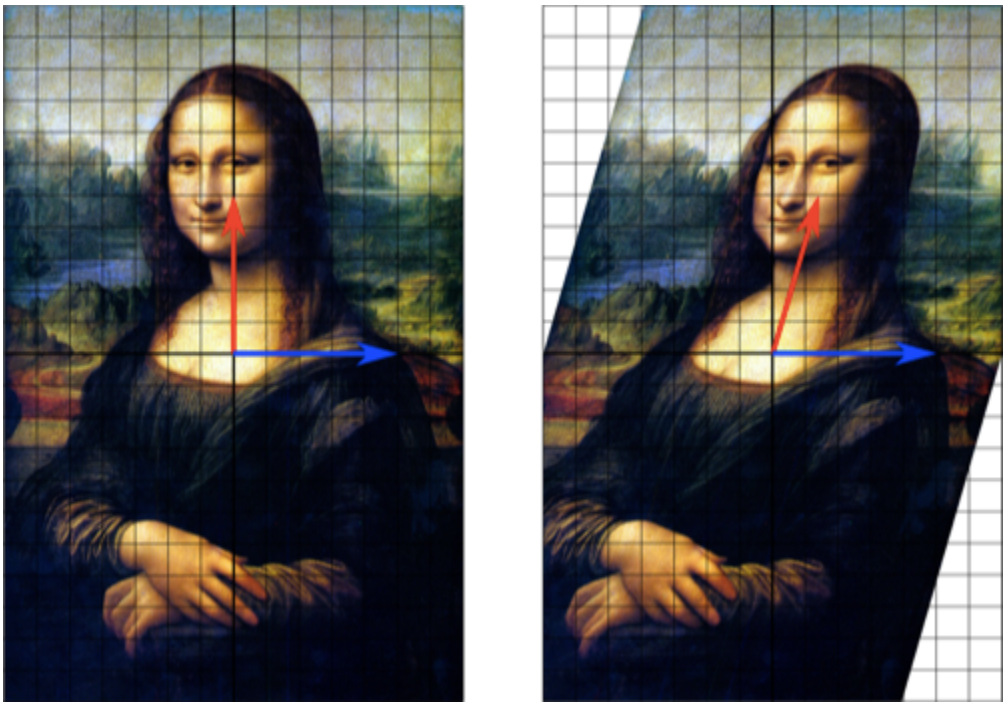
## 4-1-2 تقسیم ماتریس

عملیات دیگری که در این پروژه استفاده شده عملیات تقسیم کردن یک ماتریس بزرگ به بلوک های کوچکتر 2*2 میباشد که توضیحات بیشتر برای آن در فصل دوم ارائه میشود.
یکی از دلایل اصلی تقسیم ماتریس به بخش های کوچکتر امکان محاسبه موازی در زبان های سخت افزاری است که این امکان را به ما میدهد که همزمان بلوک های کوچکتر را در هم ضرب کنیم تا در زمان صرفه جویی شود.



## 5-1 کاربرد ها

ضرب ماتریسی عملیات بسیار پرکاربردی است. برای مثال  ضرب ماتریسی برای تبدیلات مختصاتی در گرافیک 3D بسیار حیاتی است. از ضرب ماتریسی در جبر خطی برای انجام انواع تبدیلات استفاده میشود. همچنین امروزه در اکثر نرم افزار های مدرن از ضرب ماتریسی استفاده میشود ، برای مثال برای مشخص کردن اهمیت صفحات وب از ضرب ماتریسی استفاده میشود. همچنین برای محاسبات مبتنی بر تانسور ها هم استفاده میشوند.«تانسور» (Tensor)، نقطه‌ای از فضا است که توسط یک یا چند شاخص که بیانگر مرتبه آن است، توصیف می‌شود. به‌طور کلی، تانسوری با مرتبه n در فضای m بعدی، n شاخص و$m^n$  مولفه دارد و از قواعد تبدیل معینی تبعیت می‌کند. مثلاً، تانسوری با مرتبه یک در فضای سه‌بعدی، یک شاخص و 3 مولفه دارد. در واقع، تانسورها تعمیمی از اسکالرها (که بدون شاخص هستند)، بردارها (که یک شاخص دارند) و ماتریس‌ها (که دو شاخص دارند) با تعداد دلخواهی از شاخص‌ها هستند.
همچنین در فیزیک کوانتوم هم از ماتریس ها برای توضیح برخی روابط استفاده میشوند، از سوی دیگر در نظریه گراف ها هم با استفاده از ماتریس مجاورت بسیاری از مسائل را حل میکنند.
یکی از کاربرد های ابتدایی ماتریس ها را در بینایی ماشین میتوانید در تصویر زیر ببینید:



## 6-1 استاندارد ها

### 1-6-1استاندارد IEEE754

به لحاظ تاریخی، کامپیوترهای گوناگون انتخاب‌های متفاوتی در تعیین مبنا، کران‌های نما و ارقام مانتیسِ نمایش ممیز شناور داشته‌اند. در سال 1985 با تلاش‌های گروهی متشکل از ریاضیدانان، دانشمندان علوم کامپیوتر و شرکت‌های تولید سخت‌افزار به سرپرستی ویلیام کاهان از دانشگاه کالیفرنیا، استانداردی برای نمایش اعداد ممیز شناور تحت عنوان IEEE 754 به سازندگان سخت‌افزارها عرضه شد. هم‌اکنون در بیشتر کامپیوترها از این استاندارد استفاده می‌شود. استاندارد IEEE، چند قالب کلی با دقت‌های مختلف از جمله دقت معمولی، دقت مضاعف و دقت‌های معمولی و مضاعف توسعه یافته برای نمایش اعداد ارائه می‌کند. در این‌جا به منظور آشنایی بیشتر با شیوه‌ی نمایش اعداد در این استاندارد، نحوه‌ی نمایش در دقت معمولی و مضاعف را شرح می‌دهیم.
مبنای در نظر گرفته شده در این استاندارد $\beta=2$ است. مطابق این استاندارد، در دقت معمولی از 32 بیت و در دقت مضاعف از 64 بیت برای نمایش یک عدد استفاده می‌شود. هر نمایش از سه بخش تشکیل می‌شود که عبارتند از علامت (s)، نمای تعدیل یافته (c) و قسمت کسری مانتیس نرمال شده (f).
این سه بخش با استفاده از روابط زیر مشخص می‌شوند به‌صورت (s|c|f) در کنار هم قرار می‌گیرند:
دقت معمولی:

$$x=\pm(1.f)2\times 2e=(-1)s(1.f)2\times 2c-127$$

دقت مضاعف:

$$x=\pm(1.f)2\times 2e=(-1)s(1.f)2\times 2c-1023$$

در دقت معمولی، از 32 بیت اختصاص داده شده برای نگهداری عدد، یک بیت برای علامت (s) استفاده می‌شود به‌طوری که $s=0$ برای علامت مثبت و $s=1$ برای علامت منفی به‌کار می‌رود. از 31 بیت باقیمانده، 8 بیت آن برای نگهداری نمای تعدیل یافته (c) و 23 بیت آن برای قسمت کسری مانتیس نرمال شده (f) استفاده می‌شود.
در دقت مضاعف، از 64 بیت اختصاص داده شده برای نگهداری عدد، یک بیت برای علامت (s) و از 63 بیت باقیمانده، 11 بیت آن برای نگهداری نمای تعدیل یافته (c) و 52 بیت آن برای قسمت کسری مانتیس نرمال شده (f) استفاده می‌شود.
همان‌طور که ملاحظه می‌کنید شکل کلی قالب‌های ذکر شده در دقت‌های معمولی و مضاعف، کمی شبیه به نمایش ممیز شناور نرمال است. فقط باید توجه داشت که در استاندارد IEEE مانتیس x به صورت (2.1)f نرمال و تنها قسمتی از مانتیس که با f نشان داده شده است، نمایش داده می‌شود. در واقع، چون اولین بیت مانتیس نرمال شده همواره برابر با 1 است نیازی به ذخیره‌سازی آن نیست. در عوض، این بیت برای نمایش نما مورد استفاده قرار می‌گیرد.

**مثال: عدد x=−45.75 را در نظر بگیرید. می‌خواهیم این عدد را در استاندارد IEEE با دقت معمولی نمایش دهیم. برای این منظور، ابتدا نمایش دودویی آن را می‌یابیم. داریم x=−(101101.11). حال باید این عدد را به فرم f)2×2e.1( درآوریم:**

x=−(1.0110111)×25

اکنون از این تساوی باید مقادیر f، s و c را بیابیم. با توجه به قالب کلی دقت ساده داریم:

s=1, f=0110111, e=5=c−127

در نتیجه c=132=(10000100)2 و بنابراین:

x=1|10000100|01101110000000000000000

**مثال: عدد زیر در استاندارد IEEE با دقت معمولی نمایش داده شده است.**

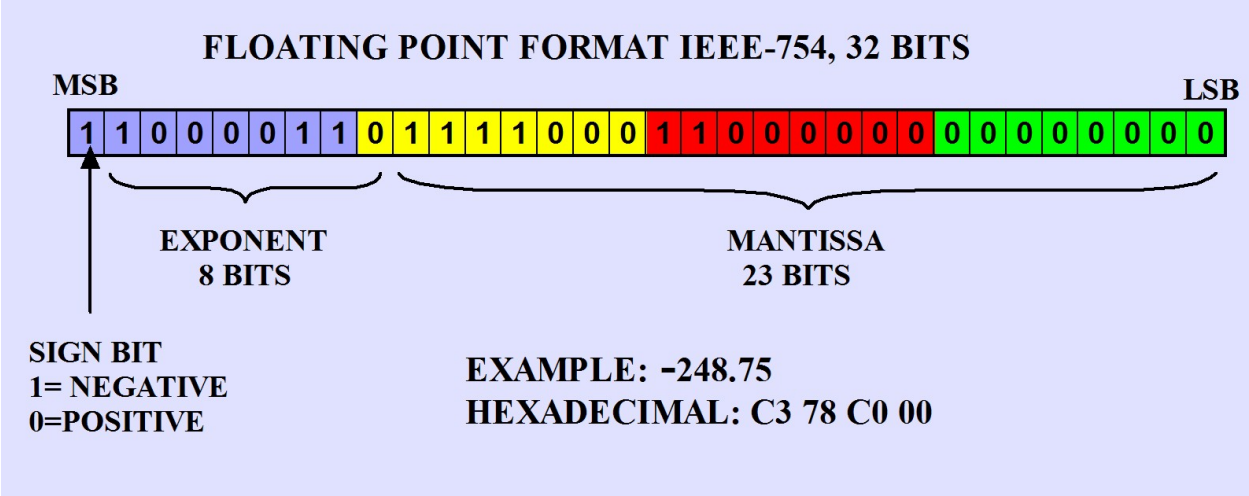
y=0|10000001|00110000000000000000000

می‌خواهیم نمایش اعشاری آن را بیابیم. با توجه به نمایش فوق داریم:

s=0, c=(10000001)2=129, f=10011

بنابراین y عددی مثبت است و e=c−127=129−127=2. در نتیجه:

y=+(1.f)2×2e=(1.10011)2×22=(110.011)2=6.375



مراجع:

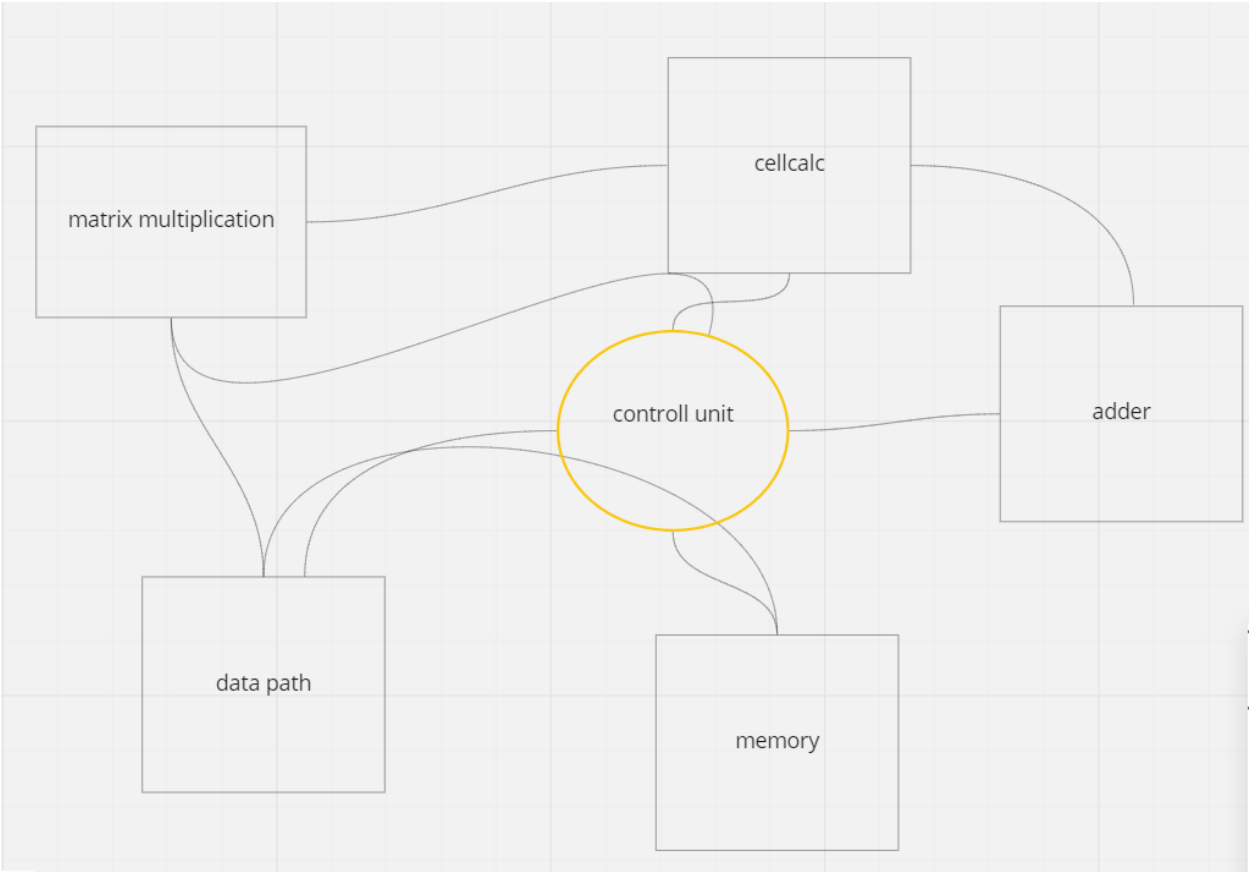

1. Jacques Philippe Marie Binet - Wikipedia
2. Hardware Matrix Multiplication – MaiTRIX
3. Matrix multiplication - Wikipedia
4. What is Parallel Computing? Definition and FAQs | OmniSci
5. IEEE-754 FLOATING POINT REPRESENTATION OF VARIABLES MANTISSA EXPONENT PUNTO FLOTANTE S.A.
6. Single-Precision Floating Point Matrix Multiplier Using Low-Power Arithmetic Circuits: Soumya Gargave, Yash Agrawal and Rutu Parekh

# فصل2 :

# توصیف معماری سیستم

پروژه از ماژول های متفاوت و یک control unit تشکیل شده است، control unit وظیفه دارد تا سیگنال ها و حالت های مختلف را برای ماژول ها تبیین کند به عبارتی مرکز فرماندهی است و بقیه ماژول ها از آن پیروی میکنند. همچنین ماژول memory هم وجود دارد که داده ها از آن خوانده و به آن نوشته خواهند شد. دیگر ماژول های مورد استفاده عبارتند از:

- ماژول matrix multiplication
- ماژول data path
- ماژول cellcalc
- ماژول main memory




به طور خلاصه ماژول cellcalc وظیفه محاسبات بلوک های ماتریسی را دارد، سیگنال ها توسط control unit آماده میوند در هر استیت و data path ورودی های cellcal را مشخص میکند، matrix multiplication هم از هر کدام از ماژول datapath و memory و control_unit یک عدد در خود میسازد و ارتباط بین آنها را همچون یک واسط برقرار میسازد

## 2-1: اینترفیس های سیستم

### 2-1-1: ورودی سیستم:

برای ورودی دادن به سیستم ابتدا باید اندازه ماتریس ها و سپس مقادیر ماتریس ها را در قسمت مموری ذخیره کنیم. برای ذخیره سازی اندازه ماتریس ها در خانه اول حافظه تعداد سطر های ماتریس اول در خانه دوم آن تعداد ستون های ماتریس اول که این تعداد با تعداد سطر های ماتریس دوم برابر است را قرار می دهیم و در خانه سوم تعداد ستون های ماتریس دوم را قرار می دهیم. سپس ماتریس اول را به صورت سطری در و ماتریس دوم را به صورت ستونی در مموری قرار می دهیم.(اگر ماتریس A به صورت [[a,b],[c,d]] باشد ترتیب سطری آن به شکل a,b,c,d است و ترتیب ستونی آن به شکل a,c,b,d است.) سپس با سیگنال getting_input که از ماژول بالاتر گرفته می شود شروع به دریافت اطلاعات از ماژول مموری و قرار دادن آن در رجیستر های مربوطه در datapath میکند. پس از تمام شدن کار انتقال اطلاعات با توجه به ابعاد ماتریس سیگنال start فعال شده و کار محاسبات انجام می شود.

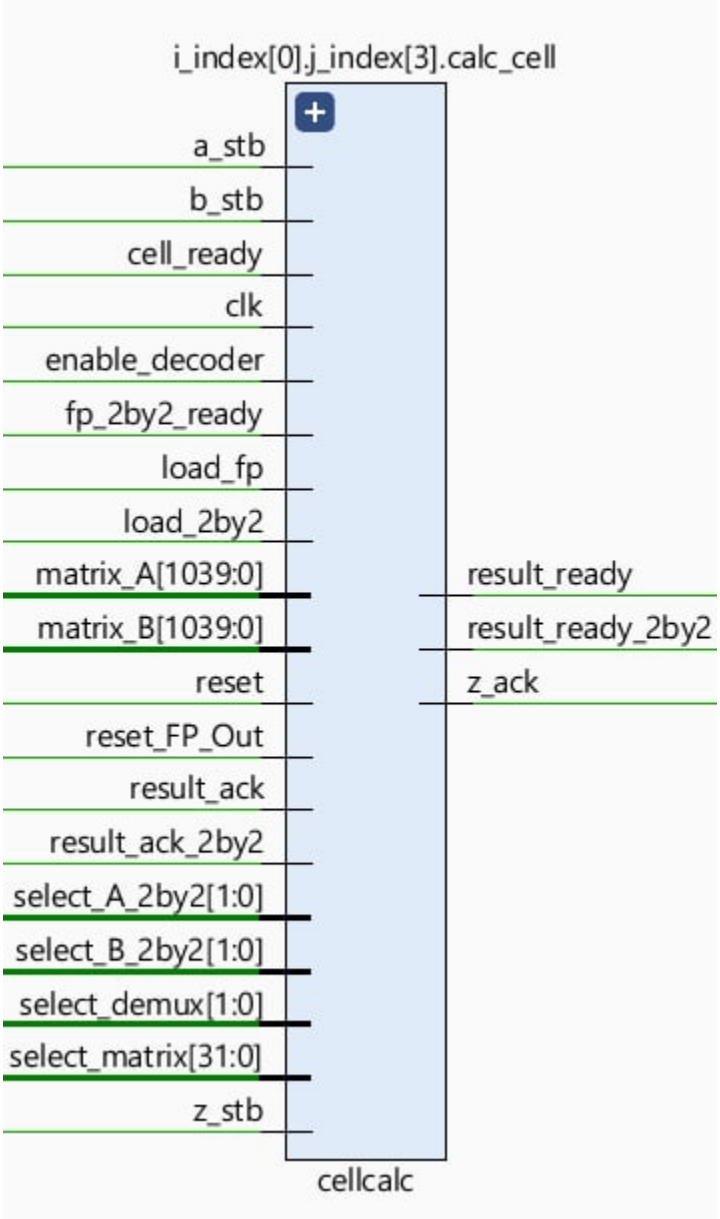
### 2-1-2: خروجی های سیستم

پس از اتمام کار سیگنال done در ماژول matrixMultiplier فعال می شود که به معنای تمام شدن کار ماست. می توانیم خروجی ها که در ماژول datapath و در cell_out ها ذخیره شده اند را از آنجا بخوانیم و مورد استفاده قرار دهیم.

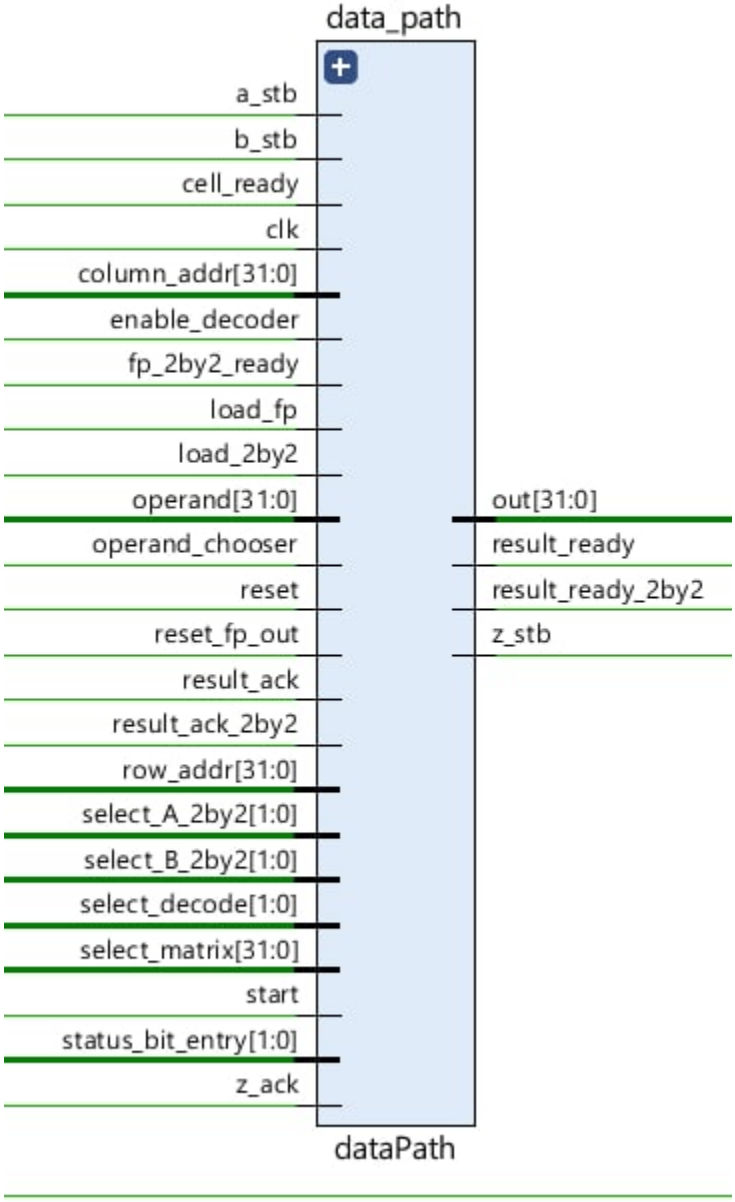
### 2-1-3: کلاک سیستم

کلاک برای کل سیستم یکسان است و برای همه ی ماژول ها از یک سیگنال استفاده می شود و چون در مدار ما clock domain crossing وجود ندارد مشکلی پیش نمی آید.

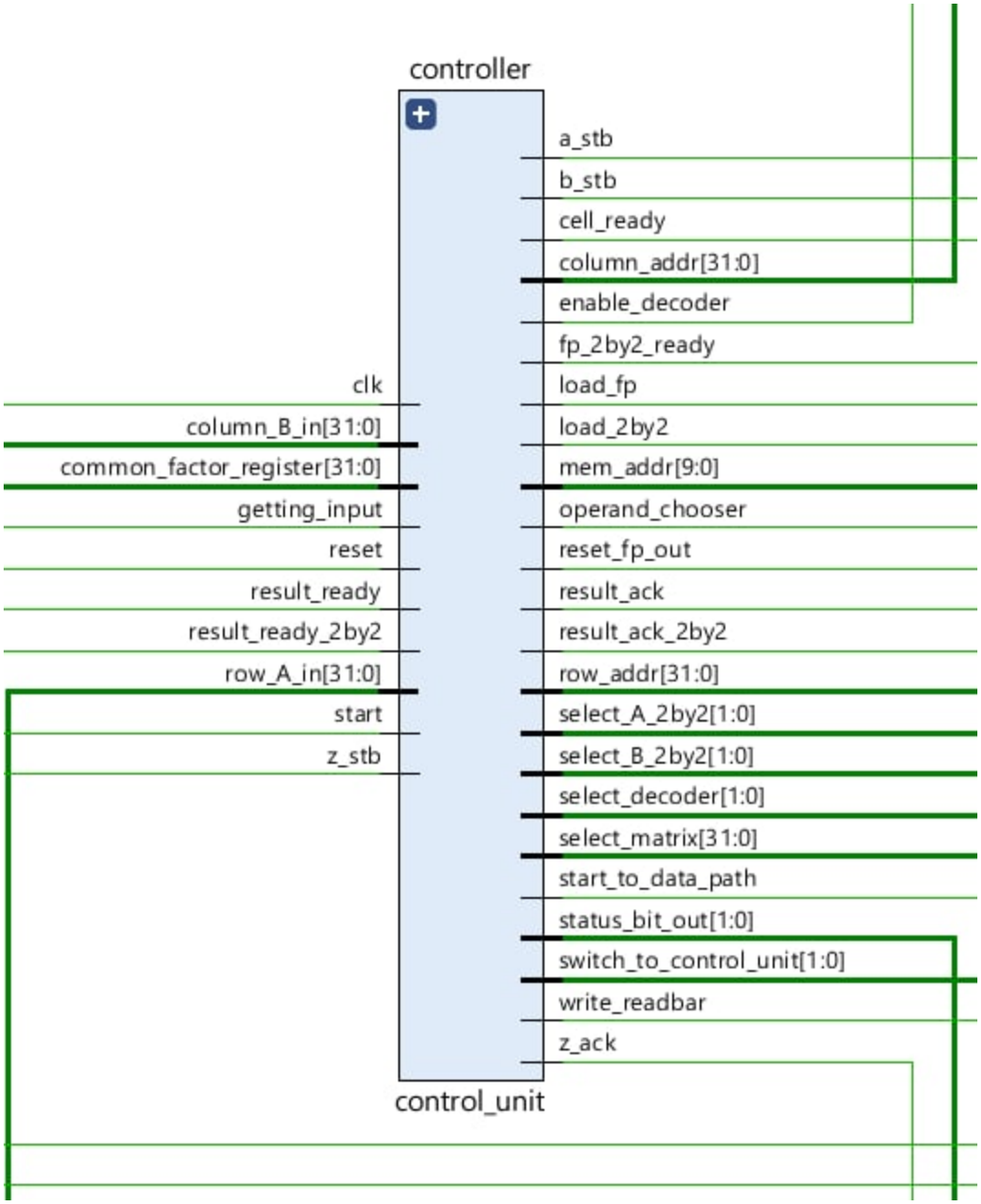
2-2: دیاگرام بلوکی سخت افزارها



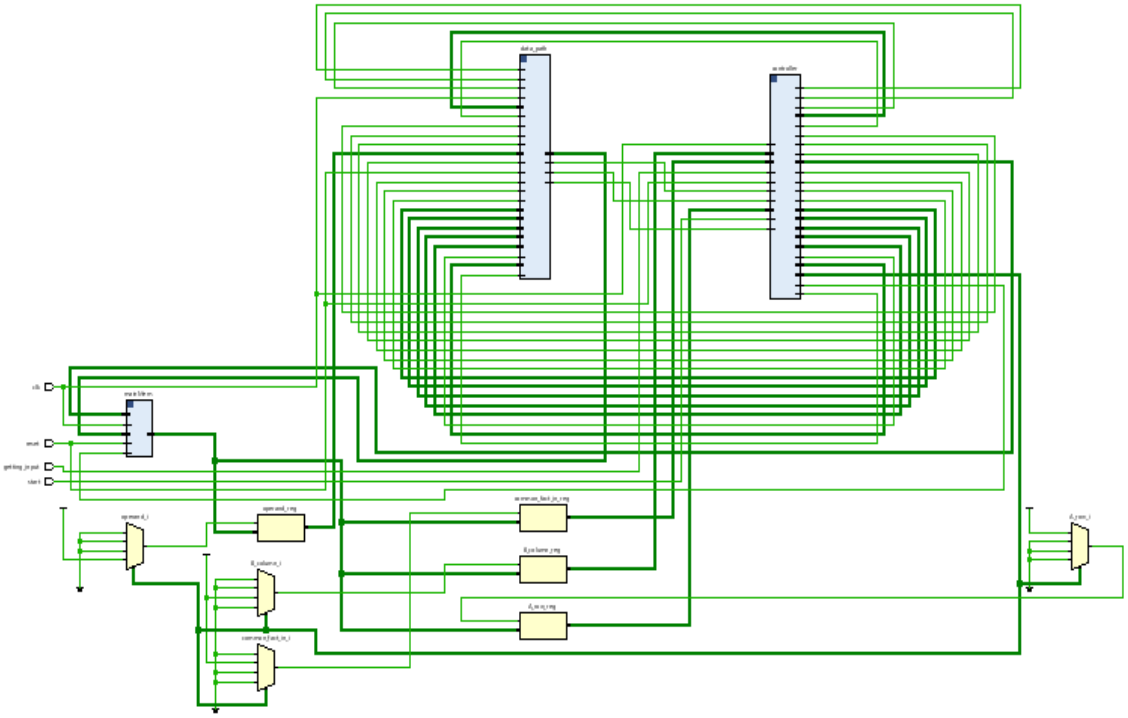

Cellcalc Diagram



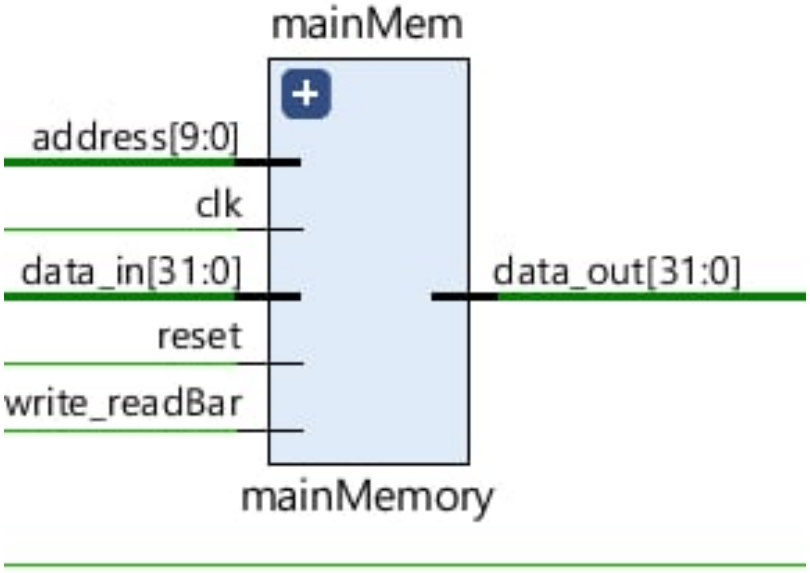

Data path diagram




Controller Diagram



Matrix multiplication Diagram




Main Memory Diagram

## 2-3: توصیف هر ماژول به صورت جداگانه

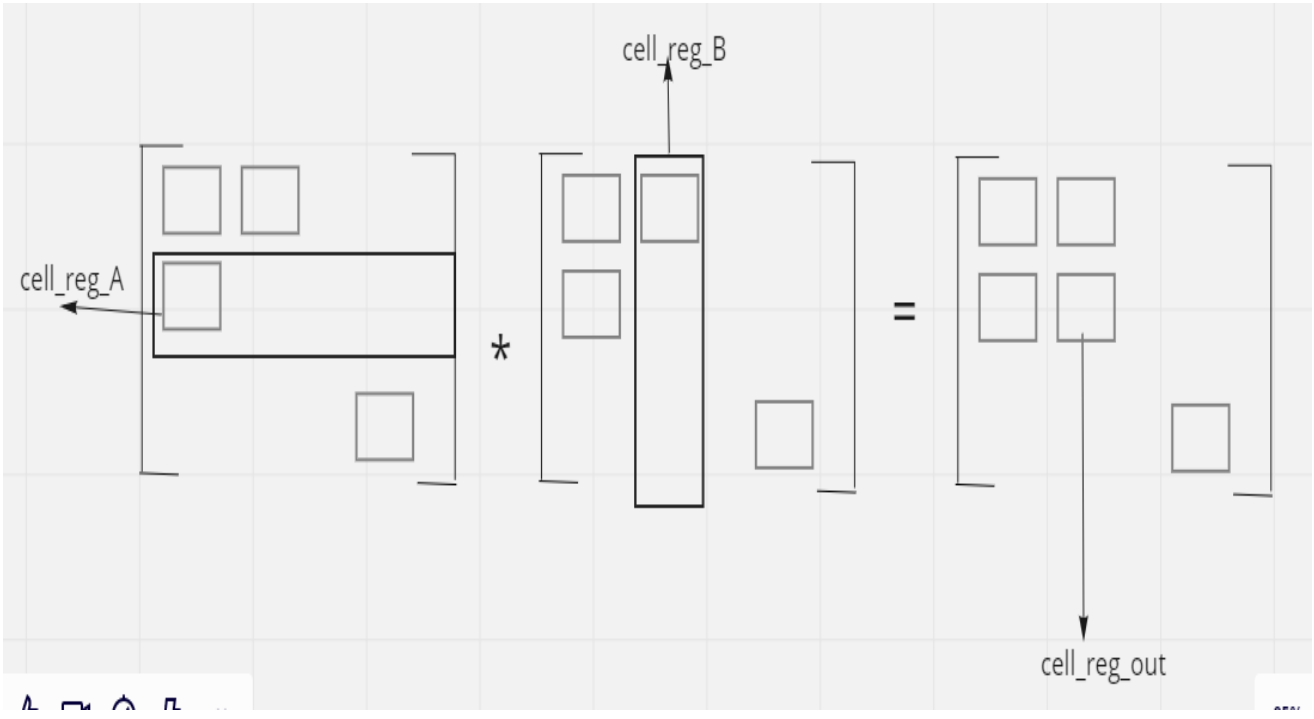
### 2-3-1 : ماژول memory

این ماژول حاوی 1024 تا رجیستر 32 بیتی است که برای نگهداری اطلاعات از آن استفاده می شود.
این ماژول نسبت به لبه بالارونده کلاک حساس است و در این لبه اگر سیگنال write_readBar فعال باشد ورودی داده شده را در آدرس داده شده ذخیره می کند و اگر فعال نباشد آدرس را در یک متغیر ذخیره کرده و مقدار آن را به عنوان خروجی به ما می دهد. این واحد در هر پالس کلاک تنها یکی از رجیسترهای خود را در اختیار ما قرار می دهد.
همچنین این ماژول یک سیگنال reset دارد که در لبه پایین رونده آن تمام رجیسترهای حافظه برابر 0 قرار داده می شوند.

### 2-3-2 : ماژول datapath

این ماژول حاوی تعداد مورد نیاز از cellcalc هاست. این تعداد به ماتریس های ورودی بستگی دارد اگر بخواهیم ماتریس A با n سطر و p ستون را در ماتریس B با p سطر و m ستون ضرب کنیم حاصل ماتریسی با n سطر و m ستون خواهد بود و ما نیاز به $\lceil \frac{n}{2} \rceil * \lceil \frac{m}{2} \rceil$ ماژول cellcalc برای محاسبه جواب نهایی نیاز داریم که این تعداد ماژول در این ماژول قرار می گیرند.
این ماژول در ابتدا دو آرایه ی cell_reg_A و cell_reg_B را با استفاده از دستوراتی که از کنترل یونیت می گیرد پر می کند.
هر کدام از خانه های این آرایه ها دو ورودی یکی از ماژول های cellcalc را نشان می دهند. همانطور که در توضیحات ماژول cellcalc توضیح داده خواهد شد این ماژول یک سطر از ماتریس اول که به ماتریس های دو در دو تقسیم شده و همچنین یک ستون از ماتریس دوم که تقسیم شده را گرفته و در هم ضرب می کند. در نتیجه در هر کدام از cell_reg_A یا cell_reg_B یک سطر یا یک ستون از ماتریس ها هستند و چون برای هر ماتریس 2*2 به 130 بیت نیاز داریم (128 بیت اطلاعات و 2 بیت برای تعیین اندازه) و تعداد این ماتریس های 2*2 در یک سطر یا ستون پس از تقسیم بندی برابر $\lceil \frac{p}{2} \rceil$ (این عدد را common_fact می نامیم) است در نتیجه هر کدام از سلول های cell_reg_A و cell_reg_B به $\frac{p}{2} * 130$بیت برای ذخیره سازی اطلاعات نیاز دارند.
برای قرارگیری این ماتریس ها از چند سیگنال استفاده می کنیم:
یک سیگنال به نام operand_chooser برای این است که تصمیم بگیریم operand را در cell_reg_A قرار دهیم یا آن را در cell_reg_B بریزیم.

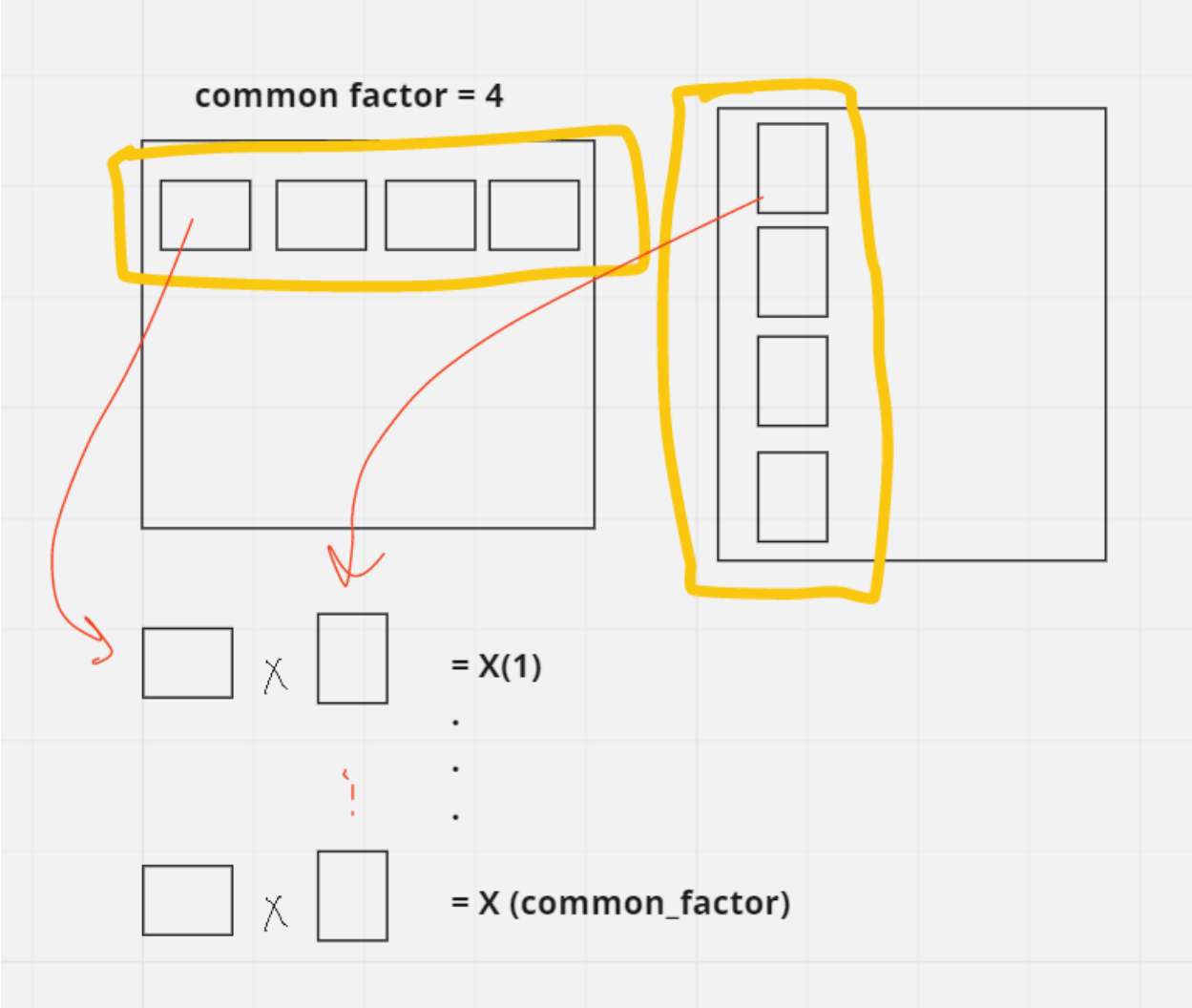
سیگنال counter مشخص می کند که کدام یک از خانه های ماتریس دو در دو در حال حاضر در حال ریخته شدن در هر کدام cell_reg_A یا cell_reg_B است. برای مثال مقدار 00 آن برای خانه بالا چپ ماتریس دو در دو است و در 32 بیت اول مقصد ذخیره می شود.
سیگنال common_factor_counter از 0 تا common_fact پیش می رود تا مشخص کند کدام یک از ماتریس های دو در دو را داریم ذخیره می کنیم. از آنجا که این p حداکثر 16 است در نتیجه common_fact حداکثر 8 است و برای آن از یک case statement استفاده می کنیم.
در هر مرحله ذخیره سازی ماتریس های دو در دو یک status_bit_entry هم داریم که دو بیت آخر را تشکیل می دهند و نشان دهنده سایز ماتریس هستند به عنوان مثال 00 نشان دهند ماتریس با ابعاد یک در یک است.
مقدار سیگنال operand_chooser از control unit به ما داده می شود.
برای counter با توجه به مقدار آن و status_bit_entry که تعداد کل اعداد را نشان می دهد می توان مقدار آن را به دست آورد به این صورت که تا زمانی که به مقدار مورد نظر نرسیده است آن را اضافه می کنیم و پس از رسیدن به مقدار مورد نظر آن را صفر می کنیم.
مقدار common_factor_counter هم پس از اتمام یک ماتریس دو در دو باید یک واحد افزایش پیدا کند یعنی زمانی که counter برابر 0 قرار می دهیم. این افزایش مقدار را تا جایی انجام می دهیم که به مقدار مورد نیاز که همان common_fact باشد برسد و در آنجا متوقف می شویم.
در نهایت با توجه به سیگنال های ورودی خروجی این cellcalc ها به control unit فرستاده می شود تا در ممووری ذخیره شود.



## 3-2-1 ماژول CellCalc

### 1-3-2-1 الگوریتم

ماژول Cellcal در واقع ماژول مخصوص ضرب کردن ماتریس هاست. به طور کلی فرآیند ضرب کردن را میتوان در عکس زیر دید:

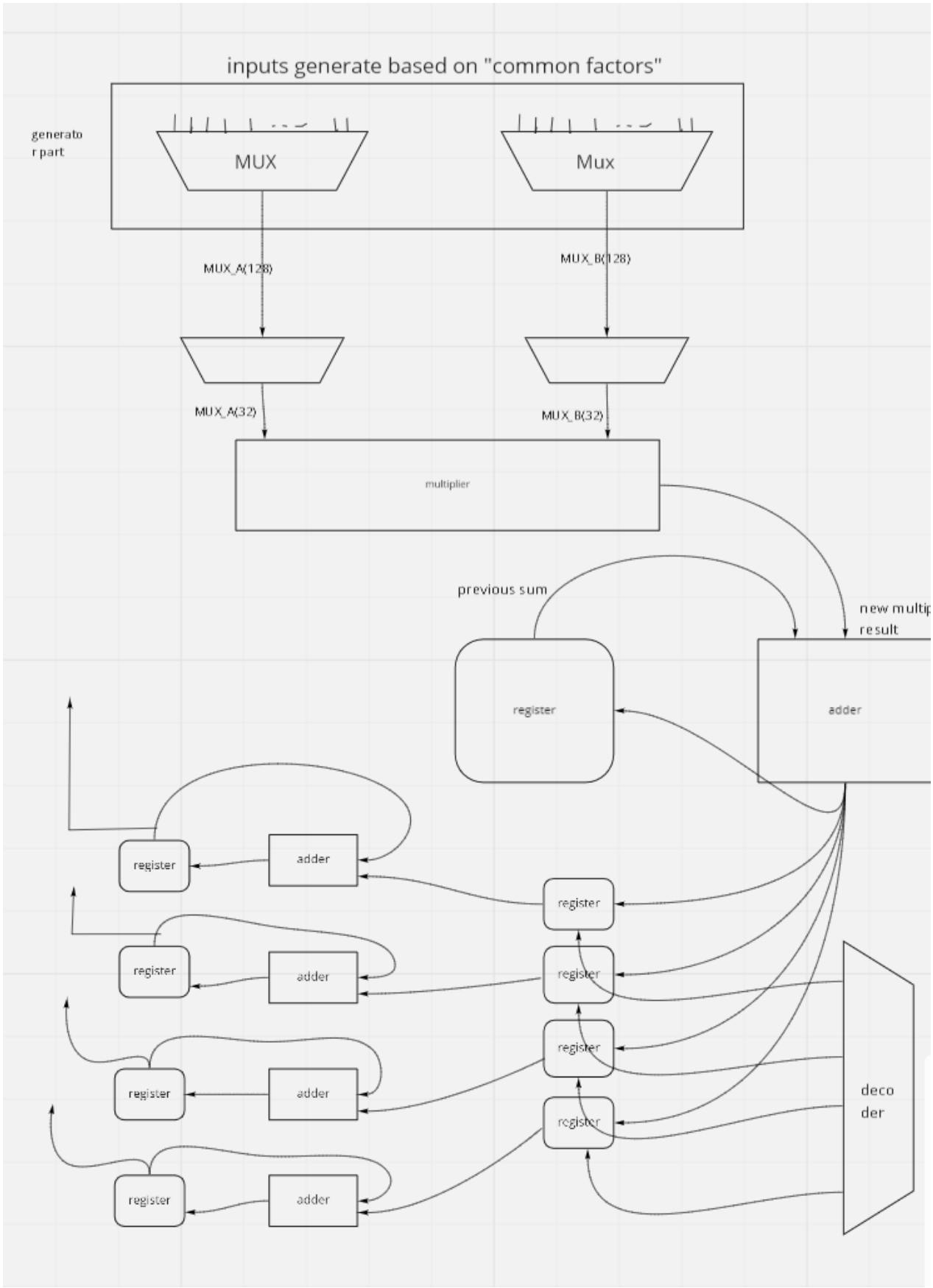



ابتدا ما ماتریس ها را به بلوک های 2*2 تقسیم میکنیم(تا جای ممکن، ممکن است بعضی ابعادشان کوچکتر شود که آن ها را هم تبدیل به 2*2 میکنیم)، همچنین تعداد اینن تقسیم بندی را Common_factor می نامیم. حال فرض کنید میخواهیم اولین بلوک 2*2 ماتریس حاصلضرب A و B یعنی C را بدست بیاوریم، برای اینکار سطر اول بلوکی ماتریس A را در ستون بلکوی اول ماتریس B ضرب کنیم. فرض کنید common_factor برابر با 4 باشد ، در اینصورت سطر اول A دارای 7 یا 8 عضو میباشد. حال ما ستون اول را به 4 بلوک 2*2 تقسیم میکنیم و اولین بلوک آن را در اولین بلوک 2*2 ستون بلوکی اول B صرب میکنیم ، در این صورت یک ماتریس موقت X1 بدست می آید، اگر به ترتیب همین روند را برای i از 1 تا common_factor انجام دهیم

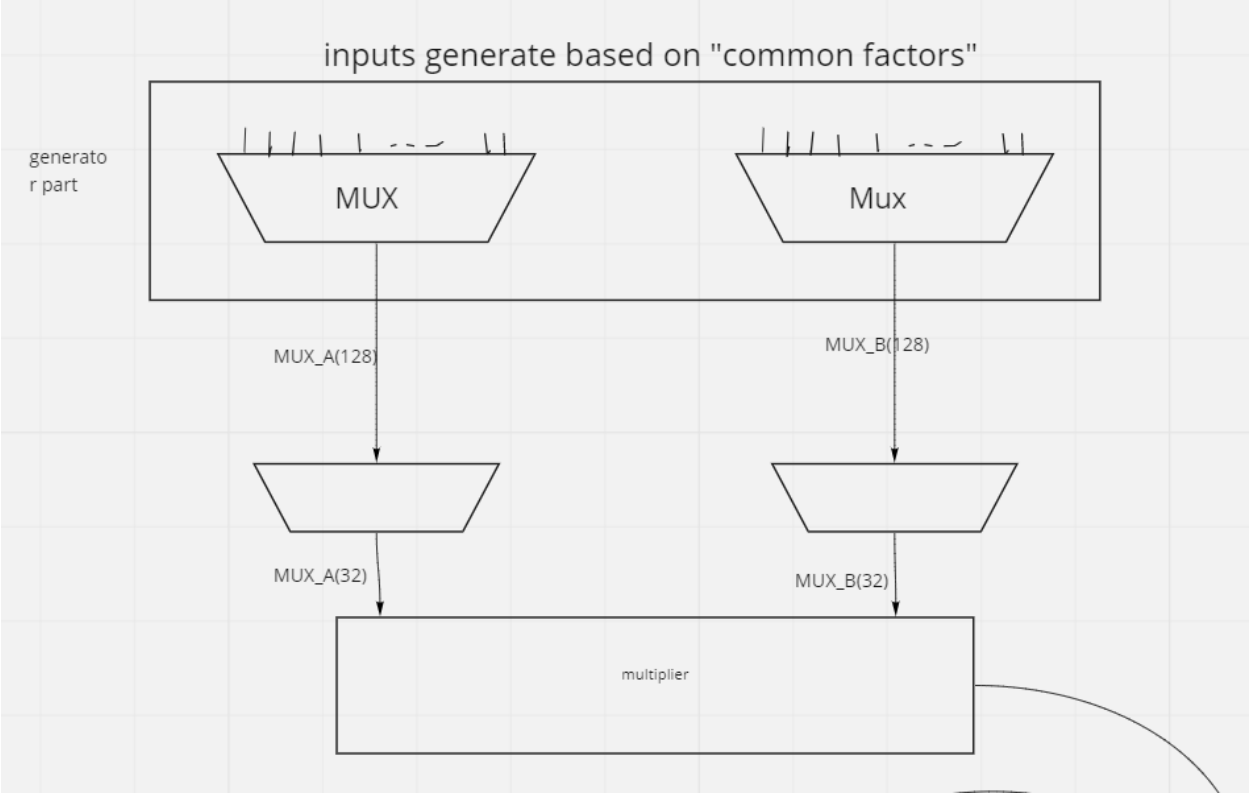
آنگاه ماتریس های $X_i$ را بدست خواهیم آورد. حال با کمی دقت می فهمید که حاصل ضرب بلوکی ما در واقع برابر با حاصل جمع $X_i$ ها میباشد. بنابراین الگوریتم ضرب در اینجا توضیح داده شد، اما پیاده سازی آن جزئیات بیشتری دارد که در ادامه ارائه خواهد شد.

### 2-3-2-1 نحوه پیاده سازی

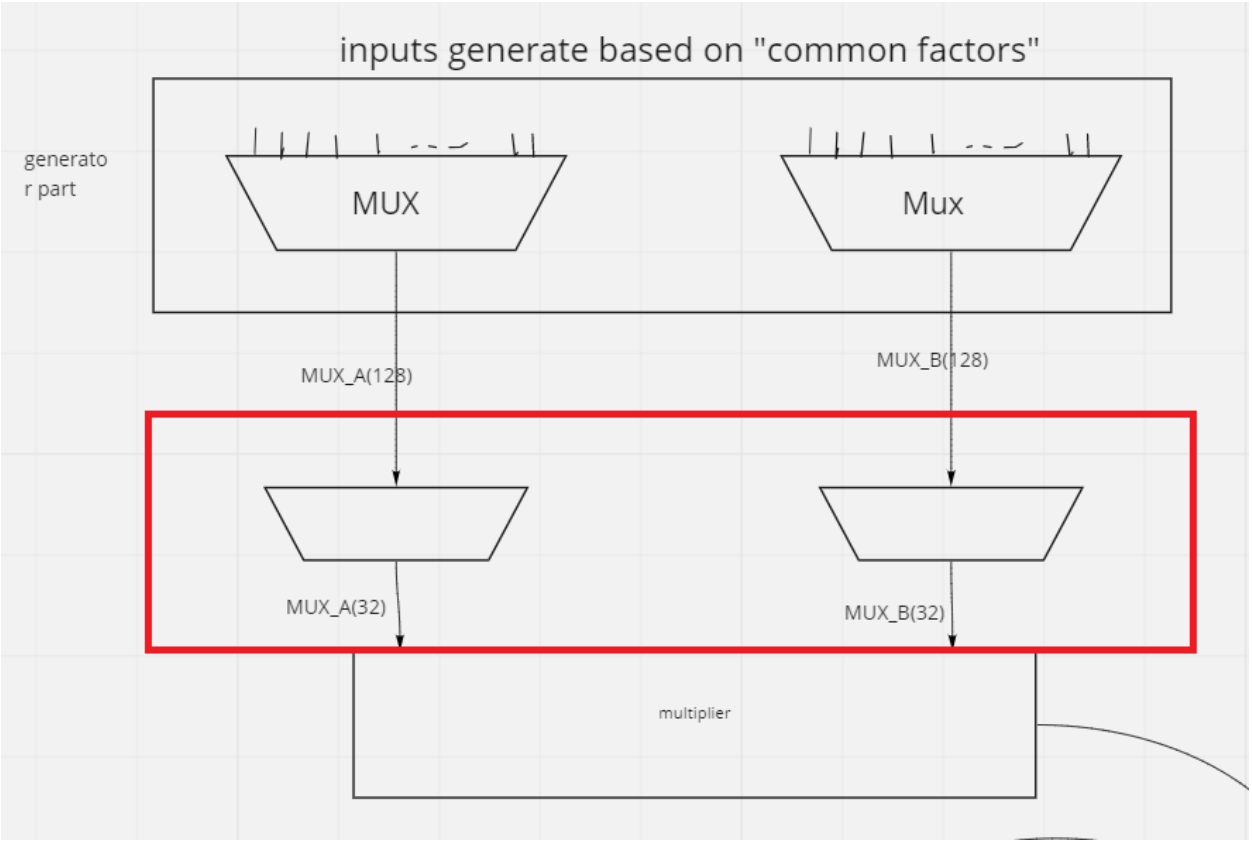
ابتدا شمای کلی فرآیند را میتوانید در تصویر زیر ببینید
توضیح کلی به این صورت است که در ابتدا ما به روشی در خواهیم یافت که چه تعداد بلوک 2*2 در سطر A و یا ستون B وجود دارند، در واقع عدد بدست آمده همان Common_factor است. حال بایستی هر بار یک جفت از بلوک ها را انتخاب کنیم و در هم ضرب کنیم ، این عملیات ضرب کردن در یک پروسسور جدا انجام میشود، به عبارتی به صورت موازی انجام میشود. پس از ضرب گردن جفت ماتریس های 2*2 بایستی این ماتریس ها را با هم جمع کنیم و در نهایت یک ماتریس 2*2 بدست می آید که حاصل ضرب یکی از سطر های A در یکی از ستون های B میباشد. حال بخش بخش کد را مرور میکنیم:


inputs generate based on "common factors"
generato
r part
MUX
Mux
MUX_A(128)
MUX_B(128)
MUX_A(32)
MUX_B(32)
multiplier
previous sum
new multip
result
register
adder
register
adder
register
register
adder
register
register
adder
register
register
adder
register
deco
der

**گام اول: انتخاب بلوک ها برای ضرب و انتخاب درایه های خاص**

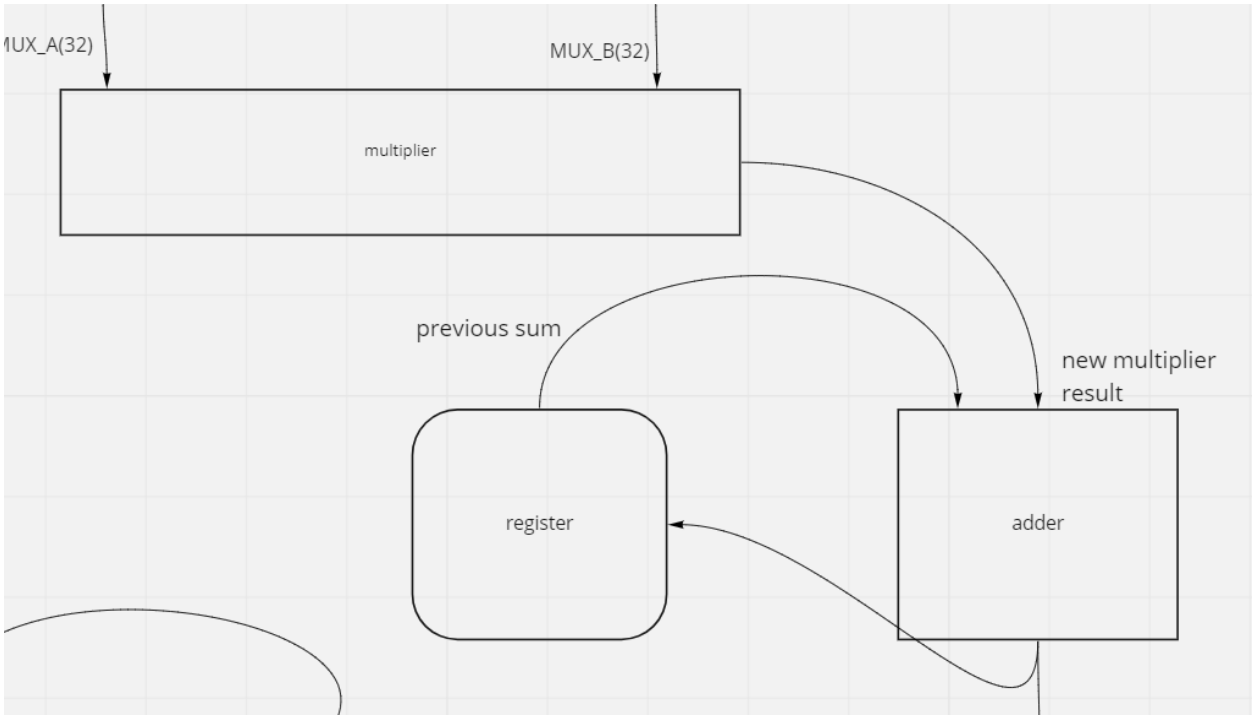



در ابتدا ما بایستی با کمک یکه مالتی پلکسر یک جفت از بلوک ها را انتخاب کنیم، منتهی به این دلیل که ما سایز ورودی ماکس ها را نمیدانیم بایستی از generate استفاده کنیم، به این صورت که بر اساس common_factor تعداد ورودی های ماکس مشخص میشوند. حال که ورودی ها مشخص شدند بر اساس سیگنال های کنترلی (select_matrix_A و select_matrix_B و خود common_factor1) خروجی هم محض خواهد شد، خروجی این ماکس ها آرایه های 130 بیتی هستند که 2 بیت آنها تگ (برای مشخص کردن اینکه داخل کدام درایه از 2*2 هستیم) و 128 بیت آن (4*32) در بردارند 4 درایه بلوک 2*2 به صورت پشت سر هم هستند.




در مرحله بعد بایستی درایه های مخصوص برای ضرب کردن را بیابیم، در اینجا هم یک ماکس داریم که از بین 4 درایه (128 بیت) 1 درایه (32 بیت) را خروجی میدهد، در نهایت این خروجی ها وارد ضرب کننده می شوند و در هم ضرب میشوند حاصل این ضرب به گام دوم میرود

## گام دوم: ضرب اولیه




همانطور که گفتیم ابتدا یک جفت بلوک انتخاب شدند و درایه های خاصی از آن ها را در هم ضرب کردیم، حال دقت کنید که در ضرب دو ماتریس بلوکی 2*2 برای بدست آوردن هر درایه حاصل ، ما باید 2 جفت عدد را در هم ضرب کرده و سپس این اعداد را با هم جمع کرده و خروجی این حاصل جمع است. مثلا فرض کنید ماتریس های ما به صورت زیر باشند:

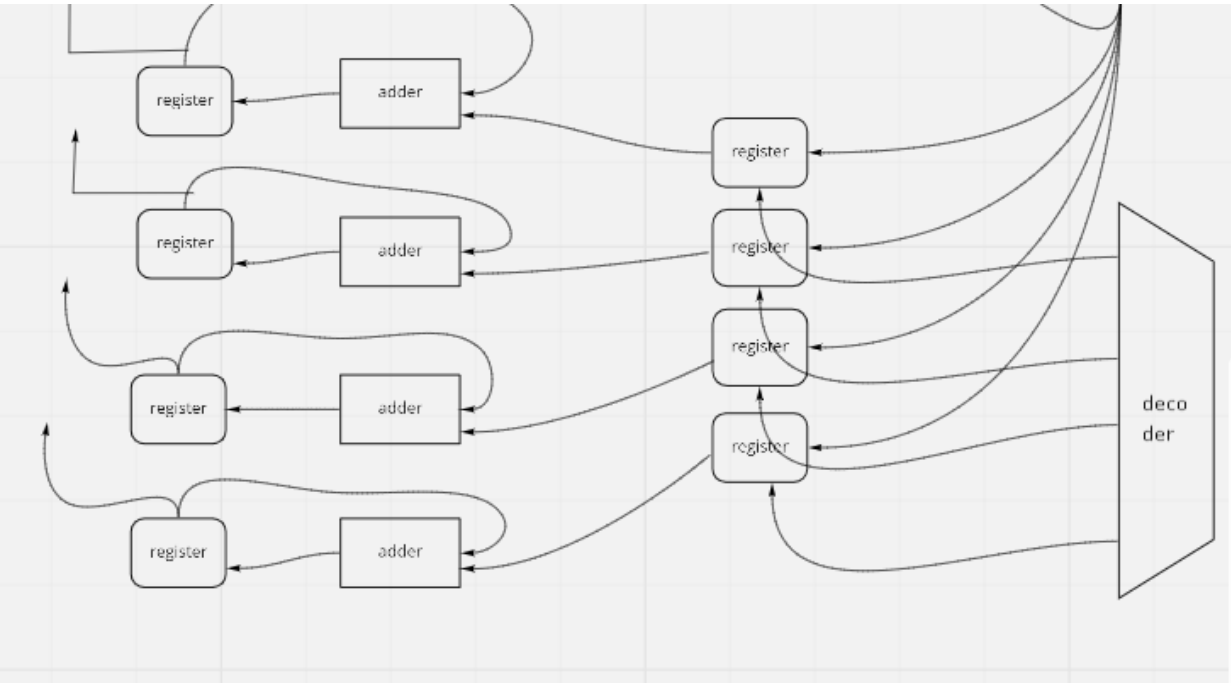
$$A = [[a, b], [c, d]],\ B = [[e, f], [g, h]]$$

درایه بالا چپ میشود:

$$a.e + b.g = result$$

کل عملیات بالا در این قسمت انجام میشود، یکبار a.e حساب میشود و در رجیستر نگهداری میشود و بار دوم b.g به حاصل قبلی اضافه میشود(نام این جمع کننده در کد fp_adder میباشد). حال دقت کنید که ما بایستی این حاصل را با حاصل ضرب های بلوک های بعدی جمع بکنیم، که به گام سوم میرسیم:

**گام سوم: جمع همه ی حاصل ها برای هر درایه**

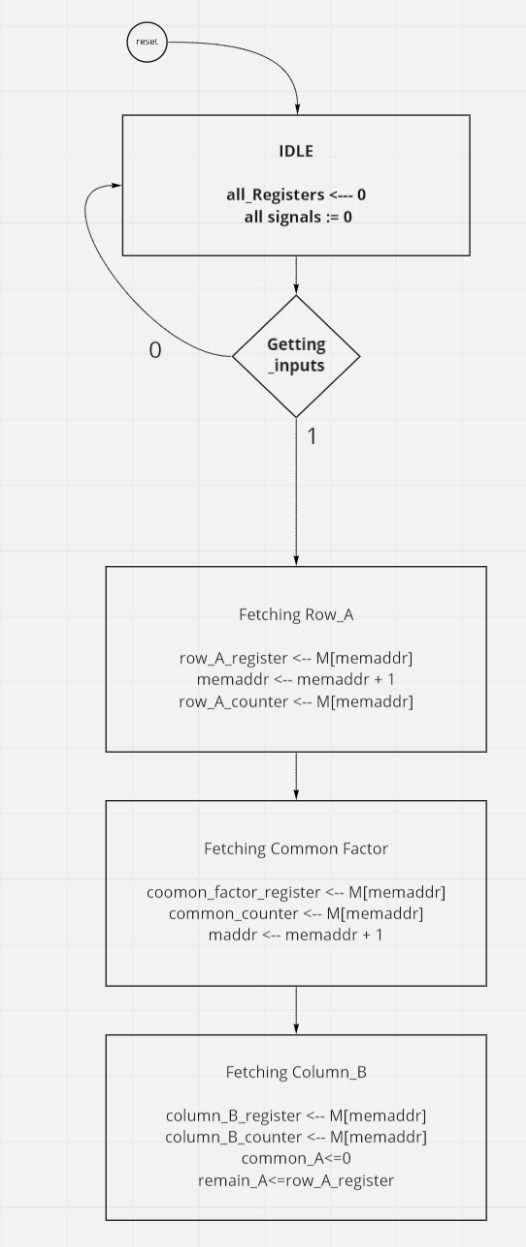



تا به حال ما دو بلوک 2*2 A و B را گرفتیم و AB=C را سعی کردیم حساب کنیم، تا به حال درایه بالا چپ آن را حساب کردیم(همان عبارت result در گام دوم)، اما بایستی result را با اعداد بدست آمده از ضرب های بلوکی دیگر هم جمع کنیم، برای اینکار result را به جمع های قبلی جمع میکنیم، به عبارتی ابتدا با یک دیکدر و به استفاده از سیگنال های کنترلی (select_demux) مشخص میکنیم که result باید با کدام رجیستر جمع شود (در اینجا چون خانه بالا چپ است با رجیستر بالایی)

با ادامه دادن این روند برای دیگر درایه ها ضرب انجام میشود و حاصل ضرب بلوک ها مشخص میشود

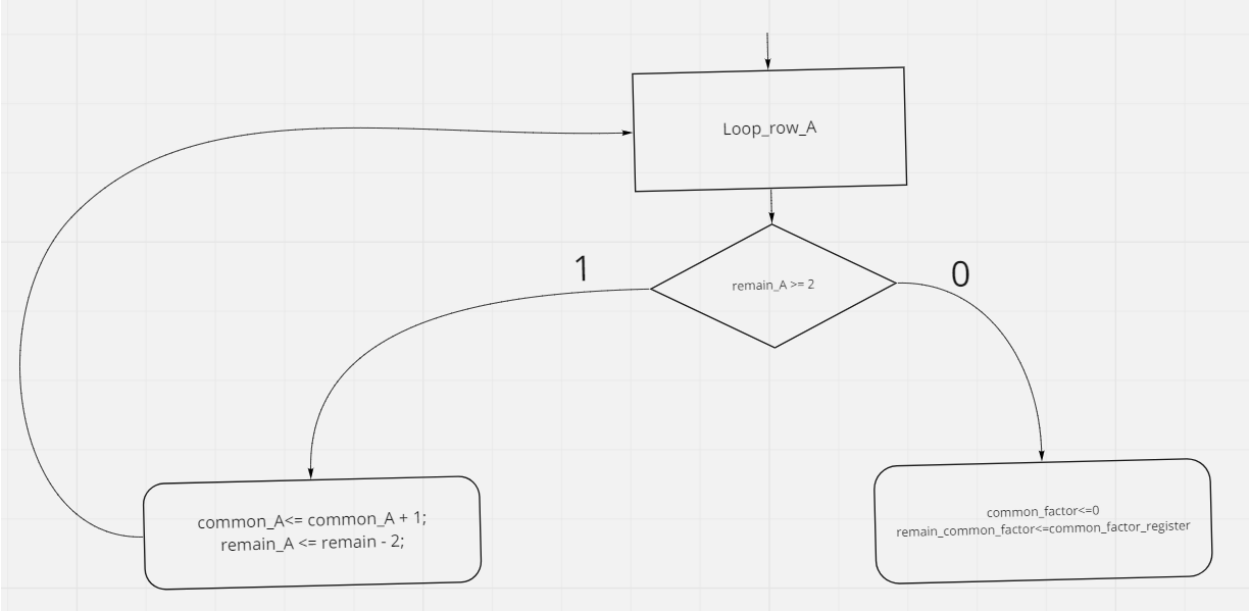
**گام چهارم (آخر) خروجی دادن در این جا صرفا سیگنال "cell_ready" که 1 شد خروجی ها آماده شده اند.**

## 5-3-2 ماژول control unit

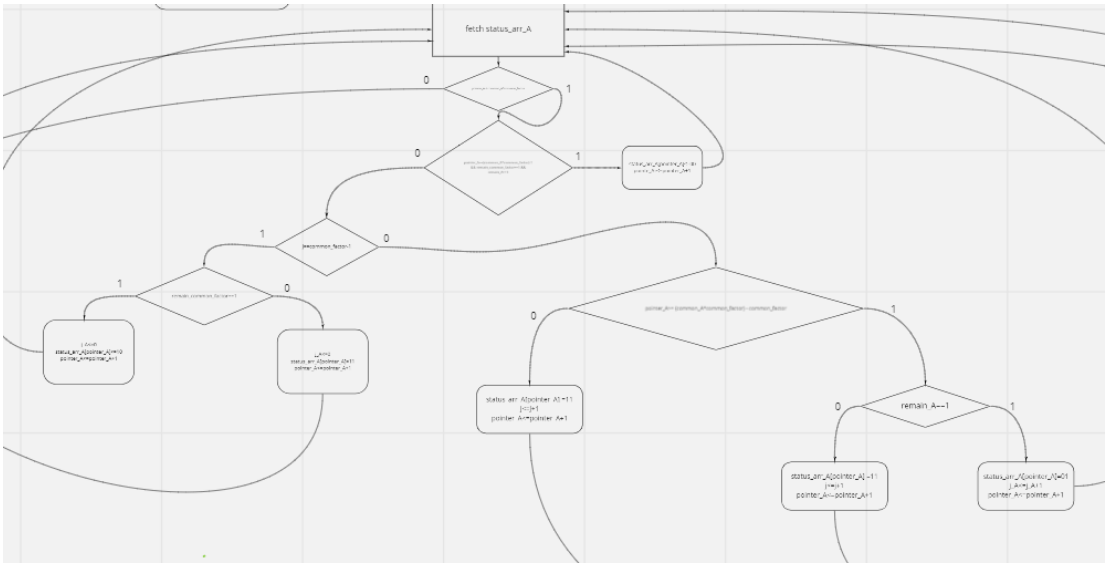



این ماژول اصلی ترین و پیچیده ترین ماژول پروژه است که سیگنال ها و استیت های مختلف را برنامه ریزی می کند و برای ماژول های دیگر نوع عملیات را مشخص میکند، به طور کلی ASM chart برای این ماژول بسیار پهناور بوده و امکان نمایش آن در اینجا وجود ندارد اما ما سعی میکنیم در قسمت های کوچکتر روند کار را توضیح دهیم.
(برای دسترسی به ASM chart به صورت کامل به این لینک مراجعه بفرمایید)

در اینجا هم asm را به چند بخش تقسیم میکنیم:
در این بخش سیگنال های مربوط به دریافت اندازه ها و خود مقادیر ماتریس ها از memory خوانده میشوند مثلا مقادیر ماتریس A وB و همچنین تعداد درایه های سطر A یا ستون B(دقت کنید که این دو عدد با هم یکسان هستند چون ضرب ماتریسی است)
همچنین تعداد درایه های ستون A و سطر B را هم ذخیره میکنیم




سپس 3 بار پشت سرهم(sequential) ماژول هایی مشابه ماژول بالا حساب می شوند ، این ماژول ها برای حساب کردن common factor و همچنین factor_A و Factor_B هستند( factor_A در واقع تعداد بلوک ها در یک ستون با عرض 2 است و factor_B هم تعداد بلوک های یک سطر B میباشد)
در اینجا ما مقدار factor ها را محاسبه میکنیم

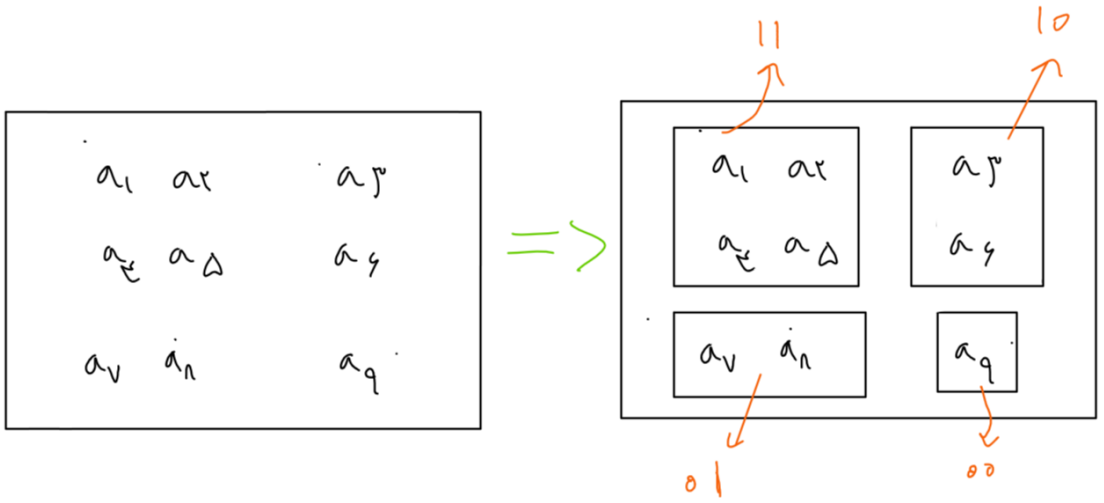



(تصویر ناواضح است برای دیدن تصویر بهتر  به این لینک مراجعه بفرمایید )

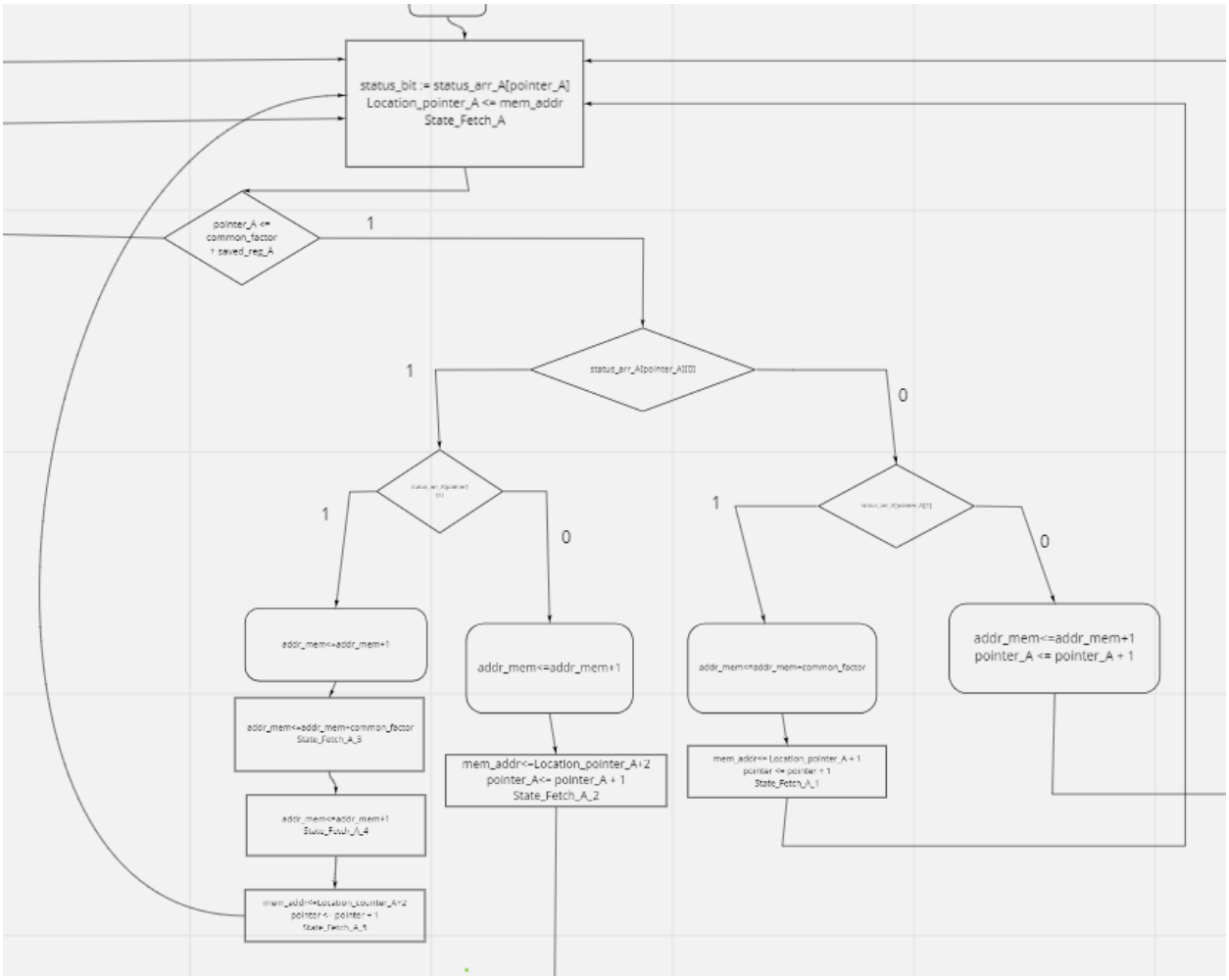
در اینجا ما باید یک مفهومی را توضیح دهیم، بلوک ها در ماتریس ما 4 حالت دارند:

- 2*2 : کد 11
- 1*2 : کد 10
- 2*1: کد 01
- 1*1: کد 00

ما به هر حالت یک کد اختصاص میدهیم، حال به هر ماتریس یک آرایه از بیت ها اختصاص می دهیم(مثلا status_arr_A) که در این ارایه حالت بلوک بندی ماتریس A مشخص شده است ، مثالی از نحوه نامگذاری را در شکلی زیر میبینید:




در پایین این ماژول دوباره تکرار شده منتها بریا ماتریس B و status_arr_B را ساخته ایم




(تصویر ناواضح است برای دیدن تصویر بهتر به این لینک مراجعه بفرمایید )

در این بخش ما ورودی های مخصوص هر cellcalc را آماده خواهیم کرد به اینصورت که هر cellcalc نیاز به یک سطر بلوکی و یک ستون بلوکی دارد، حال ما این ورودی ها را با استفاده از cell_reg_A و مشخص خواهیم کرد. در بخش اول asm ما cell_reg_A را آماده کردیم و سپس باز همین ساختار تکرار میشود و cell_reg_B ورودی های cellcalc ها را مشخص میکند.

تا اینجای کار بخش sequential پروژه را شرح کردیم، حالا وارد بخش parralel میشویم

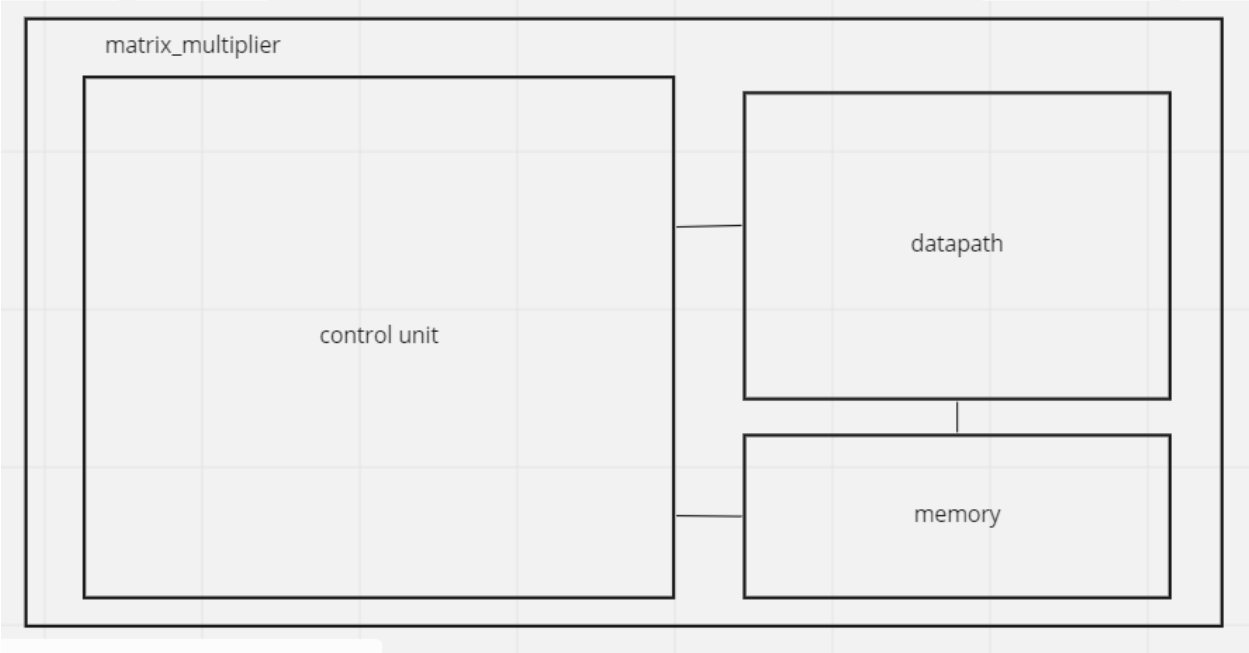
در واقع تا اینجای کار همه ی موردی های cellcalc ها آماده و به تعداد مورد نیاز ماژول cellcalchi ساخته شده است. حال بایستی صرفا همه را به صورت موازی اجرا کرد و بر حسب استیتی که در آن قرار دارند به استیت بعدی راهنمایی بشوند.

در این قسمت هم cellcalc ها محاسبه می شوند، بر اساس سایز ماتریس خروجی و حالت های مختلفی که ممکن است پیش بیاید استیت های مختلف برنامه ریزی شده اند.

تصویر مناسب این بخش بسیار بزرگ بوده و لطفا برای دیدن Asm آن به این لینک مراجعه بفرمایید

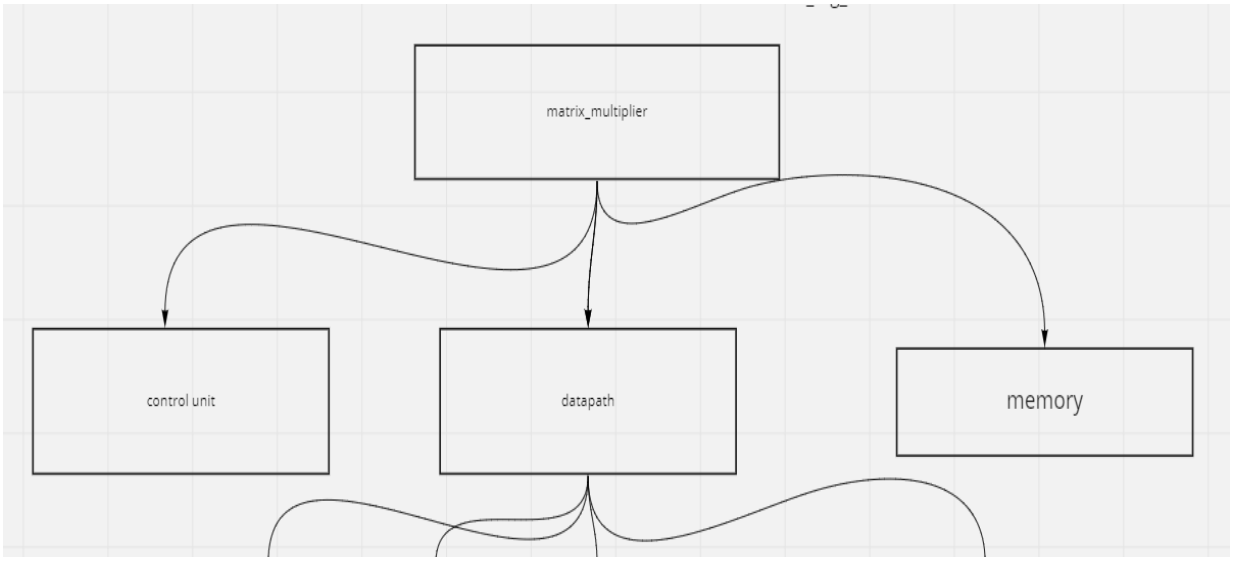
## 2-3-4 : ماژول matrix_multiplier

در این ماژول از هر کدام از ماژول datapath و memory و control_unit یک عدد قرار داده شده است که میخواهیم این ها را در کنار هم در یک ماژول قرار دهیم و با سیم هایی که تعریف شده ورودی ها و خروجی های مربوط به هر بخش که در بخش دیگر قرار دارد را به هم مرتبط سازیم. این سیگنال ها اکثرا از طرف کنترل یونیت مقدار دهی می شوند و مقدار آن ها برای ماژول های دیگر فرستاده می شود.

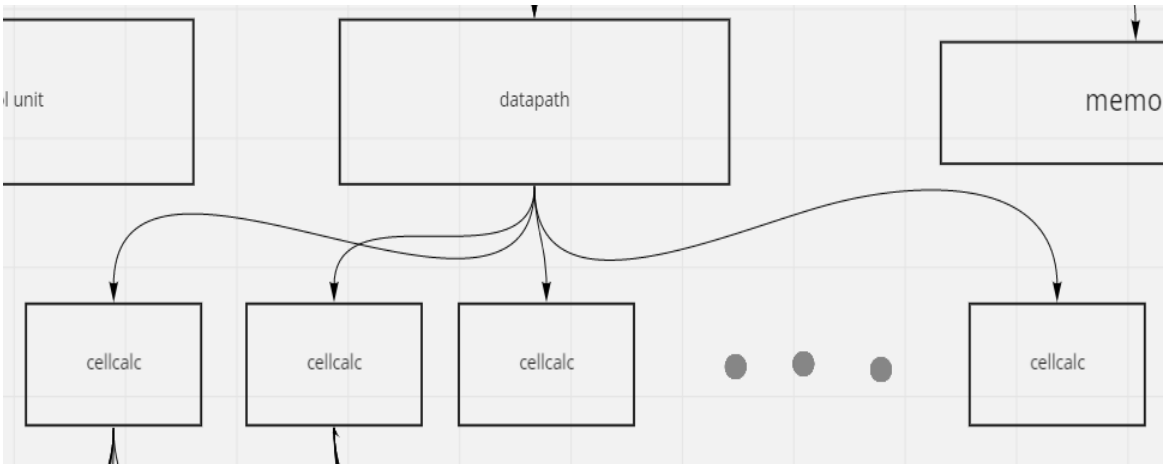


## 2-4: ساختار درختی سیستم

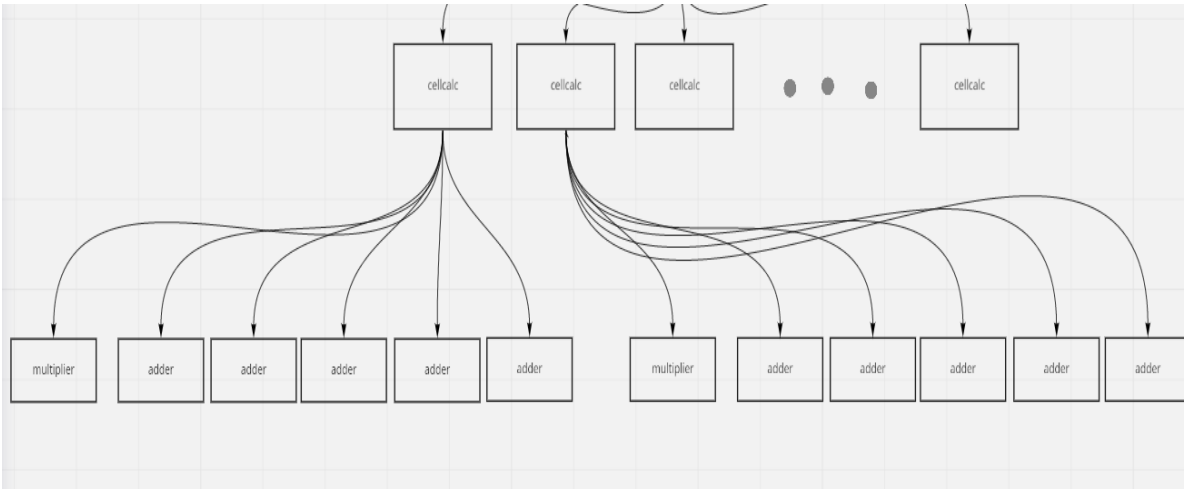
ماژول matrix_multiplier که ماژول اصلی برنامه است شامل سه زیر ماژول , control_unit datapath و memory است که وظیفه هر کدام در بخش قبل توضیح داده شد.




همانطور که در قسمت قبل توضیح داده شد ماژول datapath خود شامل تعداد مورد نیاز از ماژول های cellcalc است که این تعداد برابر $\lceil \frac{n}{2} \rceil * \lceil \frac{m}{2} \rceil$ تا ماژول است. که در زیر با سه نقطه نشان داده شده است.
ماژول memory هیچ زیر ماژولی ندارد و تنها شامل چندین رجیستر برای ذخیره سازی اطلاعات است.
ماژول control_unit هم تنها سیگنال های مورد نیاز بقیه ماژول ها را تعیین می کند و نیازی به زیر ماژول ندارد.



هر ماژول cellcalc شامل 5 adder و یک multiplier است. 4 adder آن برای هر یک از خانه ها استفاده می شود برای محاسبه همه خانه ها مشترک است.



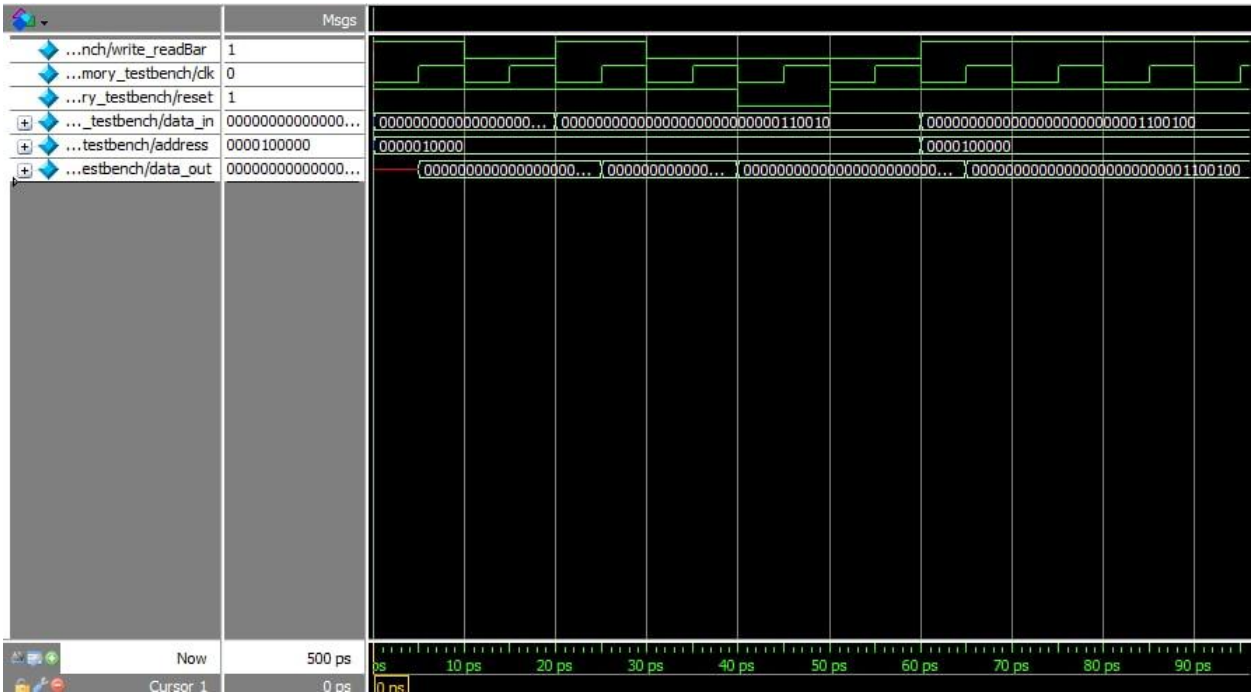
# فصل 3 : روند شبیه سازی و نتایج حاصله

## 3-1: مدل طلایی(golden model)

این مدل با زبان پایتون نوشته شده است و با استفاده از کتابخانه numpy به تولید سه عدد رندوم سپس با تابع random.normal از این کتابخانه دو آرایه با تعداد مورد نیاز از اعداد اعشاری ساخته و سپس آن را reshape میکنیم تا به ماتریس های دو بعدی تبدیل شود. و در آخر با استفاده از dot از این کتابخانه این دو را در هم ضرب می کنیم.

## 3-2: نتایج تست بنچ برای ماژول های مختلف

### 3-2-1: تست بنچ ممووری




آدرس 16 را مورد بررسی قرار می دهیم و در آن دو مقدار مختلف ریخته ایم. سپس سیگنال ریست را فعال می کنیم و مشاهده می کنیم که این مقدار صفر شده است. سپس سراغ یک آدرس دیگر رفته و این کار ها را تکرار می کنیم.

***متاسفانه فرصت کافی برای اتمام مراحل تست و سنتز پروژه در دسترس نبود ولی ما پروژه رو کامل پیاده سازی کردیم و در مرحله دیباگ و تست فرصت کم آوردیم***