**FAET630004: AI-Core and RISC Architecture** **(Due: 5/7/21)**

# Homework Assignment #2

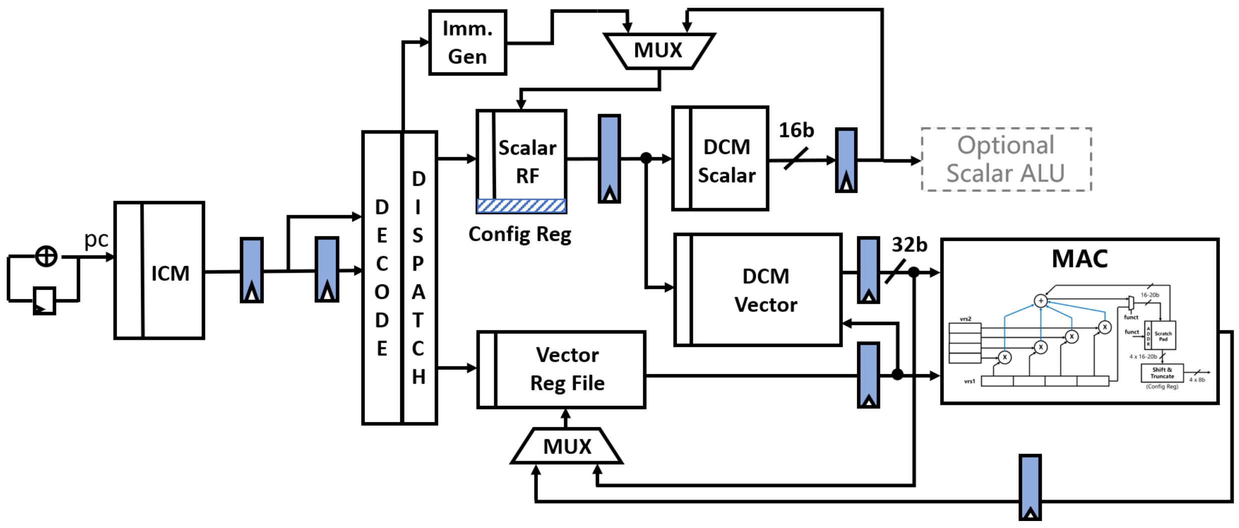
*Instructor:* Chixiao Chen *Name:* XinRu Jia（贾心茹）, *FudanID:* 20212020041

- This HW counts 15% of your final score, please treat it carefully.
- Please submit the electronic copy via mail: faet_english@126.com before the due date.
- It is encouraged to use LaTeX to edit it, the source code of the assignment is available via: https://www.overleaf.com/read/qnqfpcmqvchp
- You can also open it by Office Word, and save it as a .doc file for easy editing. Also, you can print it out, complete it and scan it by your cellphone.
- The assignment needs verilog/SV simulation. It is suggested to use Vivado from Xilinx to complete the simulation. If you do not want to install a local verilog simulator, please use an online tool: https://www.edaplayground.com/, you need register for save.
- You can answer the assignment either in Chinese or English

## Problem 1: Implement a matrix multiplier on a RISC Core (8+7=15 points)

Using the following ISA and hardware architecture to compute $\mathbf{A} \cdot \mathbf{B} + \mathbf{C}$, where $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are $8 \times 8$ matrices. Each element in them are signed integers with 8b length.

| Base: Scalar Instruction | | | | Extension: Vector Instruction | | | | |
|---|---|---|---|---|---|---|---|---|
| LOAD | rd | rs | $imm_{5b}$ | Vload | $vrd_{3b}$ | rs | $imm_{5b}$ | |
| Store | / | rs1 | rs2 | Vstore | $imm_{5b}$ | rs | $vrs_{3b}$ | |
| MOV | rd | $imm_{8b}$ | $funct_{1b}$ | VMAC | $vrd_{3b}$ | $vrs1_{3b}$ | $vrs2_{3b}$ | $funct_{4b}$ |



(a) Write the entire assembly code for computation. (hints: 8 indexed vector register file is not sufficient for 8x8 matrix.)

(b) Propose a superscalar strategy (maximum 2 instruction per fetch), and calculate how many cycles needed. Compare the utilization ratio with and without the superscalar strategy.

***(Solution)*** 1: Implement a matrix multiplier on a RISC Core8+7=15
**(a)**

This problem is essentially a matrix block problem, which is very common in the data mapping process of neural network accelerators——The memory on chip is not sufficient, so we have to divide the activation into many tile and calculate each tile orderly.Depending on the split ways, I provide two methods to deal with the problem a.
**(solution 1:)**

At first, we split three $8 \times 8$ matrices A B C into 48 vectors which has 4 implements. Because it can match the DCM vector and Vector reg file bitwidth (32b).Matrix A is divided horizontally, and matrix B is divided vertically, just as the order of calculation.And we distinguish them by H and L,which is also easily for us to annotate assembly code.Each input vectors are stored in DCM vector. So the $8 \times 8$ matrices calculation can be display as follow(The order of matrices C and S depends on the split methods of matrix B):

$$\begin{bmatrix} A1H & A1L \\ A2H & A2L \\ A3H & A3L \\ A4H & A4L \\ A5H & A5L \\ A6H & A6L \\ A7H & A7L \\ A8H & A8L \end{bmatrix} * \begin{bmatrix} B1H & B2H & B3H & B4H & B5H & B6H & B7H & B8H \\ B1L & B2L & B3L & B4L & B5L & B6L & B7L & B8L \end{bmatrix} + \begin{bmatrix} C1H & C1L \\ C2H & C2L \\ C3H & C3L \\ C4H & C4L \\ C5H & C5L \\ C6H & C6L \\ C7H & C7L \\ C8H & C8L \end{bmatrix} = \begin{bmatrix} S1H & S1L \\ S2H & S2L \\ S3H & S3L \\ S4H & S4L \\ S5H & S5L \\ S6H & S6L \\ S7H & S7L \\ S8H & S8L \end{bmatrix}$$

In instructions, vrd and vrs only have 3bit which is not sufficient for 48 vectors indexing in vector reg file. But we know that, the vectors of same type are stored in contiguous address memory. So we store the base address of input vectors -activation A, weight B and bias C in Scalar reg file, and mark all of them as RF1, RF2, RF3 easily. The same, we need restore the sum vector in DCM vector, and the base address is stored in Scalar reg file called RF4.

In this way, we can use the Vload and Vstore instructions and index vectors by stored base address RF1,RF2 and Immediate.However, we use weight stationary to get over memory Wall, 8 indexed vector reg file is not sufficient. We cannot store all the weight in the DCM(8),so I split the weight matrix B into 4 part furthermore like this:

$$\begin{bmatrix} B1H & B2H & B3H & B4H \end{bmatrix}$$
$$\begin{bmatrix} B5H & B6H & B7H & B8H \end{bmatrix}$$
$$\begin{bmatrix} B1L & B2L & B3L & B4L \end{bmatrix}$$
$$\begin{bmatrix} B5L & B6L & B7L & B8L \end{bmatrix}$$

Besides, we calculate matrix S twice depended on the partitioned matrix A. In a word,my design assembly code looks like this :

```
// Step1:load the weight
// Matric B split1
Vload  VRF1,  RF2,  $0                   //load B1H
Vload  VRF2,  RF2,  $1                   //load B2H
Vload  VRF3,  RF2,  $2                   //load B3H
Vload  VRF4,  RF2,  $3                   //load B4H
// Step2: Initial Bias
Vload  VRF5,  RF3,  $0                   //load C1H
VMAC   VRF8,  VRF5,  /,  $1000            //Add C1H
// Step3: Computing
Vload  VRF5,  RF1,  $0                   //load A1H
VMAC   \   ,  VRF1, VRF5, $0000          //
VMAC   \   ,  VRF2, VRF5, $0001               //
VMAC   \   ,  VRF3, VRF5, $0010               //
VMAC   VRF8,  VRF4, VRF5, $0111               //Partial S1H=A1H * BiH + C1H
VStore $0  ,   RF4, VRF8                      //store partial S1H
//Then we cycle above 2 steps 8 times till A8H
•
•
•
// Step2: Initial Bias
Vload  VRF5,  RF3,  $7                        //C8H
VMAC   VRF8,  VRF5,  /,  $1000                //
// Step3: Computing
Vload  VRF5,  RF1,  $7                        //A8H
VMAC   \   ,  VRF1, VRF5, $0000               //
VMAC   \   ,  VRF2, VRF5, $0001               //
VMAC   \   ,  VRF3, VRF5, $0010               //
VMAC   VRF8,  VRF4, VRF5, $0111               //Partial S8H
VStore $7  ,   RF4, VRF8

// Step1:exchange the weight B, Bias C and Psum S
Vload  VRF1,  RF2,  $4                        //B4H
Vload  VRF2,  RF2,  $5                        //B5H
Vload  VRF3,  RF2,  $6                        //B6H
Vload  VRF4,  RF2,  $7                        //B7H

 // Step2: Initial Bias
Vload  VRF5,  RF3,  $8                        //C1L
VMAC   VRF8,  VRF5,  /,  $1000                 //
// Step3: Computing
Vload  VRF5,  RF1,  $0                        //A1H
VMAC   \   ,  VRF1, VRF5, $0000               //
VMAC   \   ,  VRF2, VRF5, $0001               //
VMAC   \   ,  VRF3, VRF5, $0010               //
VMAC   VRF8,  VRF4, VRF5, $0111               //Partial S1L
VStore $8  ,   RF4, VRF8
//Then we cycle above 2 steps 8 times till A8H
•
•
•
// Step2: Initial Bias
Vload  VRF5,  RF3,  $15                       //C8L
VMAC   VRF8,  VRF5,  /,  $1000                 //
// Step3: Computing
Vload  VRF5,  RF1,  $7                        //A8H
VMAC   \   ,  VRF1, VRF5, $0000               //
VMAC   \   ,  VRF2, VRF5, $0001               //
VMAC   \   ,  VRF3, VRF5, $0010               //
VMAC   VRF8,  VRF4, VRF5, $0111               //Partial S8L
VStore $15  ,   RF4, VRF8

// Step1:exchange the weight B and Psum S
Vload  VRF1,  RF2,  $8                        //B1L
Vload  VRF2,  RF2,  $9                        //B2L
Vload  VRF3,  RF2,  $10                       //B3L
Vload  VRF4,  RF2,  $11                       //B4L

// Step4: Reload Psum
Vload  VRF5,  RF4,  $0                       //load Partial S1H = A1H * BiH + C1H
```

```
VMAC   VRF8,  VRF5,  /,  $1000                //

// Step3: Computing
Vload  VRF5,  RF1,  $0                        //A1L
VMAC   \   ,  VRF1, VRF5, $0000               //
VMAC   \   ,  VRF2, VRF5, $0001               //
VMAC   \   ,  VRF3, VRF5, $0010               //
VMAC   VRF8,  VRF4, VRF5, $0111               //All S1H
VStore $0  ,   RF4, VRF8
//Then we cycle above 2 steps 8 times till A8H
•
•
•
// Step4: Reload Psum
Vload  VRF5,  RF3,  $7                       //Parial S8H
VMAC   VRF8,  VRF5,  /,  $1000                 //
// Step3: Computing
Vload  VRF5,  RF1,  $7                        //A8L
VMAC   \   ,  VRF1, VRF5, $0000               //
VMAC   \   ,  VRF2, VRF5, $0001               //
VMAC   \   ,  VRF3, VRF5, $0010               //
VMAC   VRF8,  VRF4, VRF5, $0111               //All S8H
VStore $7  ,   RF4, VRF8

// Step1:exchange the weight B and Psum S
Vload  VRF1,  RF2,  $13                       //B5L
Vload  VRF2,  RF2,  $14                       //B6L
Vload  VRF3,  RF2,  $15                       //B7L
Vload  VRF4,  RF2,  $16                       //B8L

// Step4: Reload Psum
Vload  VRF5,  RF4,  $8                       //Parial S1L
VMAC   VRF8,  VRF5,  /,  $1000                //

// Step3: Computing
Vload  VRF5,  RF1,  $8                        //A1L
VMAC   \   ,  VRF1, VRF5, $0000               //
VMAC   \   ,  VRF2, VRF5, $0001               //
VMAC   \   ,  VRF3, VRF5, $0010               //
VMAC   VRF8,  VRF4, VRF5, $0111               //All S1L
VStore $8  ,   RF4, VRF8
//Then we cycle above 2 steps 8 times till A8H
•
•
•
// Step4: Reload Psum
Vload  VRF5,  RF3,  $15                       //Parial S8L
VMAC   VRF8,  VRF5,  /,  $1000                 //
// Step3: Computing
Vload  VRF5,  RF1,  $15                       //A8L
VMAC   \   ,  VRF1, VRF5, $0000               //
VMAC   \   ,  VRF2, VRF5, $0001               //
VMAC   \   ,  VRF3, VRF5, $0010               //
VMAC   VRF8,  VRF4, VRF5, $0111               //All S8L
VStore $15  ,   RF4, VRF8
```

It’s worthy adding that the assembly code include 4 big cycles and 8 small cycles , in actual situation, we can replace them by callq , jz and loop to simply the code.

Besides, we can calculate that,number of DCM fetches is 112:
Matrix A: 8 * 4 load
Matrix B: 4 * 4 load
Matrix C: 8 * 2 load
Matrix S: 8 * 2 * 2 store 8 * 2 load

And this block matrix can be expand with the increasing of matrices' size.

**(solution 2:)**

As described in the above method, we have to reload each elements of matrix A from DCM vector twice. Besides, the matrix sum also need to reload once and store twice to complete twice MAC. So I change the split matrices methods, and in this method, the matrix sum only need store once.

The 48 input vectors are divided as follow:

$$\begin{bmatrix} A1H & A1L \\ A2H & A2L \\ A3H & A3L \\ A4H & A4L \\ A5H & A5L \\ A6H & A6L \\ A7H & A7L \\ A8H & A8L \end{bmatrix} * \begin{bmatrix} B1H & B2H & B3H & B4H & B5H & B6H & B7H & B8H \\ B1L & B2L & B3L & B4L & B5L & B6L & B7L & B8L \end{bmatrix} +$$

$$\begin{bmatrix} C1H & C2H & C3H & C4H \\ C5H & C6H & C7H & C8H \\ C1L & C2L & C3L & C4L \\ C5L & C6L & C7L & C8L \end{bmatrix} = \begin{bmatrix} S1H & S2H & S3H & S4H \\ S5H & S6H & S7H & S8H \\ S1L & S2L & S3L & S4L \\ S5L & S6L & S7L & S8L \end{bmatrix}$$

I still hold on the weight stationary,so the matrix B is divided in four parts. But this way I split the weight matrix B into 4 parts spanning multiple lines like this:

$$\begin{bmatrix} B1H & B2H \\ B1L & B2L \end{bmatrix} \begin{bmatrix} B3H & B4H \\ B3L & B4L \end{bmatrix} \begin{bmatrix} B5H & B6H \\ B5L & B6L \end{bmatrix} \begin{bmatrix} B7H & B8H \\ B7L & B8L \end{bmatrix}$$

So the cycle calculation is a little different from the method a. In a word,my design assembly code looks like this :

```
Vload  VRF6,  RF1,  $1                     //load A1L
VMAC   \   ,  VRF3, VRF6, $0000            //
VMAC   \   ,  VRF4, VRF6, $0001            //Partial S1H=A1H * BiH + A1L * BiL + C1H

Vload  VRF5,  RF1,  $0                     //load A2H
VMAC   \   ,  VRF1, VRF5, $0010            //
VMAC   \   ,  VRF2, VRF5, $0011            //
Vload  VRF6,  RF1,  $1                     //load A2L
VMAC   \   ,  VRF3, VRF6, $0010            //
VMAC   VRF8,  VRF4, VRF6, $0111            //All S1H=AiH * BiH + AiL * BiL + C1H
VStore $0  ,   RF4, VRF8                   //store S1H

//Then we cycle above 2 steps 4 times till A8H
•
•
•

//Change the matrix weight B other 3 splits
```

```
//Repeat above 4 small cycles
•
•
•

// Step1:load the weight
// Matric B split1
Vload  VRF1,  RF2,  $0                        //load B7H
Vload  VRF2,  RF2,  $1                        //load B8H
Vload  VRF3,  RF2,  $2                        //load B7L
Vload  VRF4,  RF2,  $3                        //load B8L
// Step2: Initial Bias
Vload  VRF5,  RF3,  $0                        //load C4H
VMAC   VRF8,  VRF5,  /,  $1000                 //Add C4H
// Step3: Computing
Vload  VRF5,  RF1,  $0                        //load A1H
VMAC   \   ,  VRF1, VRF5, $0000               //
VMAC   \   ,  VRF2, VRF5, $0001               //
Vload  VRF6,  RF1,  $1                        //load A1L
VMAC   \   ,  VRF3, VRF6, $0000               //
VMAC   \   ,  VRF4, VRF6, $0001               //Partial S1H=A1H * BiH + A1L * BiL + C1H

Vload  VRF5,  RF1,  $0                        //load A2H
VMAC   \   ,  VRF1, VRF5, $0010               //
VMAC   \   ,  VRF2, VRF5, $0011               //
Vload  VRF6,  RF1,  $1                        //load A2L
VMAC   \   ,  VRF3, VRF6, $0010               //
VMAC   VRF8,  VRF4, VRF6, $0111               //All S1H=AiH * BiH + AiL * BiL + C1H
VStore $0  ,   RF4, VRF8                      //store S1H

//Then we cycle above 2 steps 4 times till A8H
•
•
•
```

Although the matrix sum only store once, in this method, we have to load matrix A fourth to complete MAC. So we can calculate that,number of DCM fetches is 112,too:
Matrix A: 16 * 4 load
Matrix B: 4 * 4 load
Matrix C: 4 * 4 load
Matrix S: 4 * 4 store Of coarse, there are many other functions such as based on actvation stationary or output stationary. Whatever,choose the reasonable splited methods to balance memory wall and latency depends your own situation.

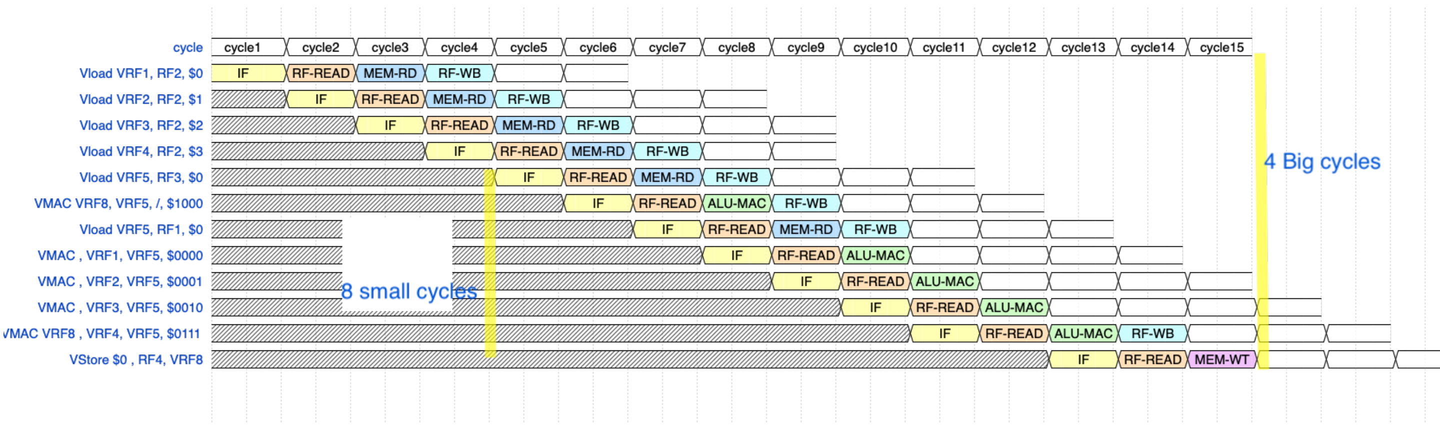


Figure 1: customed pipeline instructions waveform

**(b)**

Based on function 1,we proposed a superscalar strategy. As I said above, this method include 4 big cycles and 8 small cycles, so I just analyze the non-repeated instruction parallelism.

Assumed that ,all steps cost the same time.

As Figure 1,in customed pipeline,we have 4 big cycles and each big cycle has 8 small cycles. And we can figure that,the small cycles' throughout is 11 cycles.Combined above, we can calculate the throughout of whole assembly code:

$$((8 * (11 - 2) + 2 + 4) - 2) * 4 + 2 = 306clks$$

Before superscalar strategy, we assumed that, the DCM vector is TrueDualPortRam. Or we can design two DCM to store for weight and activation data seperatly. In other words, we can load or store two vectors based on different address at the same time.

And Based on Lecture 4, we use the Load+Mac instruction. In other words, we proposed a LoMac instruction to complete two function in 5 cycles.But it must execute with a load instruction at the same time.

```
Vload   Vrd(3b),  rs1(5b),  imm(5b)
VloMac  Vrd(3b),  rs2(5b),  imm(1b), funct(4b)
```

Therefore, most load and LoMac instruction can be run in parallel. But the step2/4(initial bias/reload psum) and MAC in small cycle might cause data hazard. So we have to execute two instructions separately.

And the Step1(load weight) can be combined with Step3(Computing).But cause the VloMac only have imm(1b), so We add more base scalar RF to store base address. Specifically, RF1 and RF3 still works for matrix A and C.RF2 is for Psum. Matrix B need RF4 to RF11 to index. The depth of scalar RF is 32, so it's enough. The improving assembly code show follow:

```
// Step2: Initial Bias
Vload  VRF5,  RF3,  $0                    //load C1H
VLoMac VRF8,  \  ,  \,  $1000             //Add C1H

// Step3: Combine load weight and Computing
Vload  VRF5,  RF1,  $0                    //load A1H
VloMac \   ,  RF4,  $0,  $0000            //load B1H and Mac A1H and B1H
Vload  VRF5,  RF1,  $0                    //load A1H
VloMac \   ,  RF4,  $1,  $0001            //load B2H and Mac A1H and B2H
Vload  VRF5,  RF1,  $0                    //load A1H
VloMac \   ,  RF5,  $0,  $0010            //load B3H and Mac A1H and B3H
Vload  VRF5,  RF1,  $0                    //load A1H
VloMac VRF8,  RF5,  $1,  $0111            //load B4H and Mac A1H and B4H
VStore $0   ,   RF2, VRF8                 //store partial S1H
//Then we cycle above step 8 times till A8H
```

```
•
•
•
// Step2: Initial Bias
Vload  VRF5,  RF3,  $7                          //load C8H
VLoMac VRF8,  \  ,  \,  $1000                   //Add C8H

// Step3: Combine load weight and Computing
Vload  VRF5,  RF1,  $1                          //load A8H
VloMac \   ,  RF4,  $0,  $0000                  //load B1H and Mac A8H and B1H
Vload  VRF5,  RF1,  $0                          //load A1H
VloMac \   ,  RF4,  $1,  $0001                  //load B2H and Mac A8H and B2H
Vload  VRF5,  RF1,  $0                          //load A1H
VloMac \   ,  RF5,  $0,  $0010                  //load B3H and Mac A8H and B3H
Vload  VRF5,  RF1,  $0                          //load A1H
VloMac VRF8,  RF5,  $1,  $0111                  //load B4H and Mac A8H and B4H
VStore $7  ,   RF2, VRF8                        //store partial S8H


//Step2: exchange the weight B, Bias C and Psum S

Vload  VRF5,  RF3,  $8                          //load C1L
VLoMac VRF8,  \  ,  \,  $1000                   //Add C1L

// Step3: Combine load weight and Computing
Vload  VRF5,  RF1,  $0                          //load A1H
VloMac \   ,  RF6,  $0,  $0000                  //load B1H and Mac A1H and B1H
Vload  VRF5,  RF1,  $0                          //load A1H
VloMac \   ,  RF6,  $1,  $0001                  //load B2H and Mac A1H and B2H
Vload  VRF5,  RF1,  $0                          //load A1H
VloMac \   ,  RF7,  $0,  $0010                  //load B3H and Mac A1H and B3H
Vload  VRF5,  RF1,  $0                          //load A1H
VloMac VRF8,  RF7,  $1,  $0111                  //load B4H and Mac A1H and B4H
VStore $0  ,   RF2, VRF8                        //store partial S1L
//Then we cycle above step 8 times till A8H
•
•
•
// Step2: Initial Bias
Vload  VRF5,  RF3,  $15                         //load C8L
VLoMac VRF8,  \  ,  \,  $1000                   //Add C8L

// Step3: Combine load weight and Computing
Vload  VRF5,  RF1,  $1                          //load A8H
VloMac \   ,  RF6,  $0,  $0000                  //load B1H and Mac A8H and B1H
Vload  VRF5,  RF1,  $0                          //load A1H
VloMac \   ,  RF6,  $1,  $0001                  //load B2H and Mac A8H and B2H
Vload  VRF5,  RF1,  $0                          //load A1H
VloMac \   ,  RF7,  $0,  $0010                  //load B3H and Mac A8H and B3H
Vload  VRF5,  RF1,  $0                          //load A1H
VloMac VRF8,  RF7,  $1,  $0111                  //load B4H and Mac A8H and B4H
VStore $7  ,   RF2, VRF8                        //store partial S8L
•
•
•
//The same for 4 cycles, except that, we don't need matrix C to the Psum,
//But we reload the calculated Psum as the initial Bias.
//Refer to the assembly code in Page 4 for details.
```

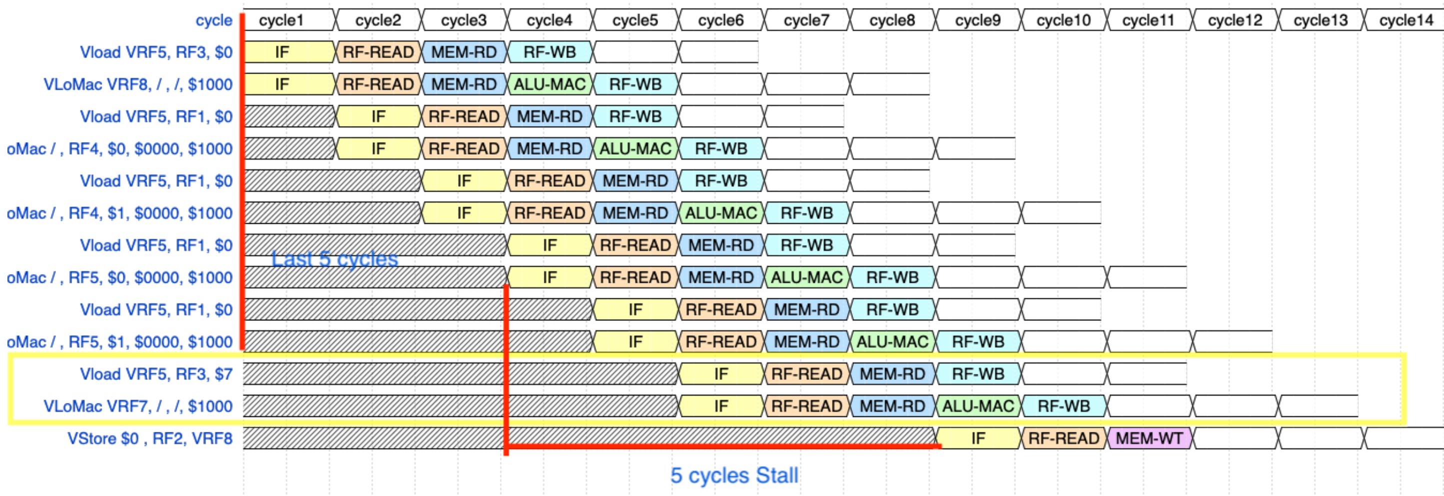


Figure 2: Superscalar strategy instructions waveform

As Figure 2,we have 4 big cycles and each big cycle only has 8 small cycles with superscalar strategy. And we can figure that,the small cycles' throughout is 9 cycles without VStore. And we have to stall the pipeline 3 cycles for VStore. And this instruction, we cannot realize superscalar.However, just like the yellow rectangle, we can run next cycles' instruction. And I confirm that if we change the VRF for Psum, we can avoid the data hazard. Combined above, we can calculate the throughout of whole assembly code:

$$(8 * (9 - 4) + 8) * 4 - 1 + 6 \;=\; 197clks$$

$$Before\ Superscalar : 5 * 8 * 4/306 \;=\; 52.29\%$$

$$After\ Superscalar : 5 * 8 * 4/197 \;=\; 81.22\%$$

In this work, we assumed that all step cost the same time. But in reality, ALU-MAC costs much more longer than others, so superscalar strategy could improve the pipeline latency significantly.

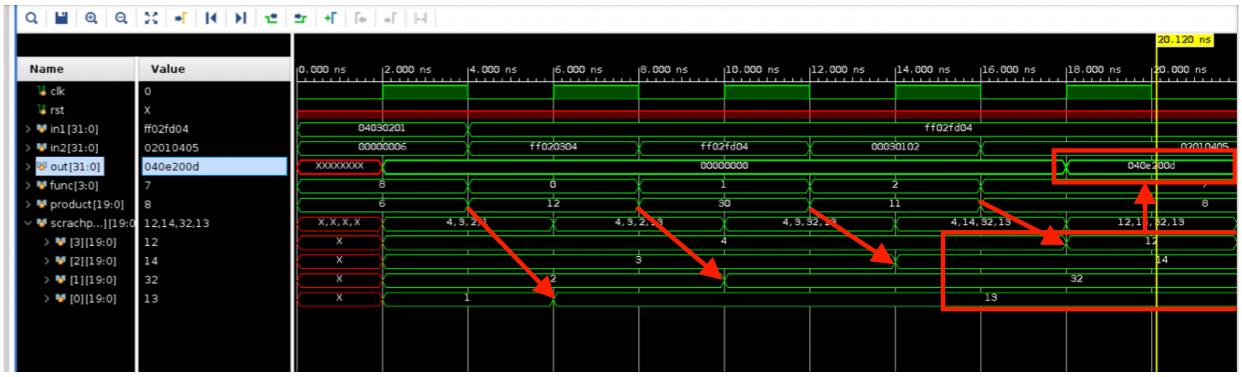


Figure 3: VMAC module waveform

**(c)**

In this system,we realize a MAC module based on the circuit showed in Problem1.The source code was written by SystemVerilog called mac.sv. And the interface of MAC module was showed follow:

```
module mac(
    input logic clk,
    input logic [31:0]in1,
    input logic [31:0]in2,
    input logic [3:0]func,
    output logic [31:0] out

);
```

We also submit the complete source code in the attachment.And the testbench called tb_mac.sv was attached, too. In the testbench, we realize the maxtrix MAC.

$$\begin{bmatrix}-1 & 2 & -3 & 4\end{bmatrix} * \begin{bmatrix}-1 & -1 & 0 & 2\\ 2 & 2 & 3 & 1\\ 3 & -3 & 1 & 4\\ 4 & 4 & 2 & 5\end{bmatrix} + \begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix} = \begin{bmatrix}13 & 32 & 14 & 12\end{bmatrix} 1741619148355.pic_hdcopy$$

In Figure3, we realize 5 VMAC instructions to complete the matrix calculation. The final out is a splicing of 4 quantized results in scratchpad of MAC.0$c$ 0$e$ 20 0$d$ is the hexadecimal of result of 12 14 32 13.We have reached the expected functional goal.The specific source code just like the mac.sv and tb_mac.sv in appendix.I also attached a screenshot of the simulation waveform.