**Final Exam CMPE-523 12.01.2018 (45 points, 150 min)**

St. Name, Surname______________________________________ St.Id#____________

Instructor Alexander Chefranov

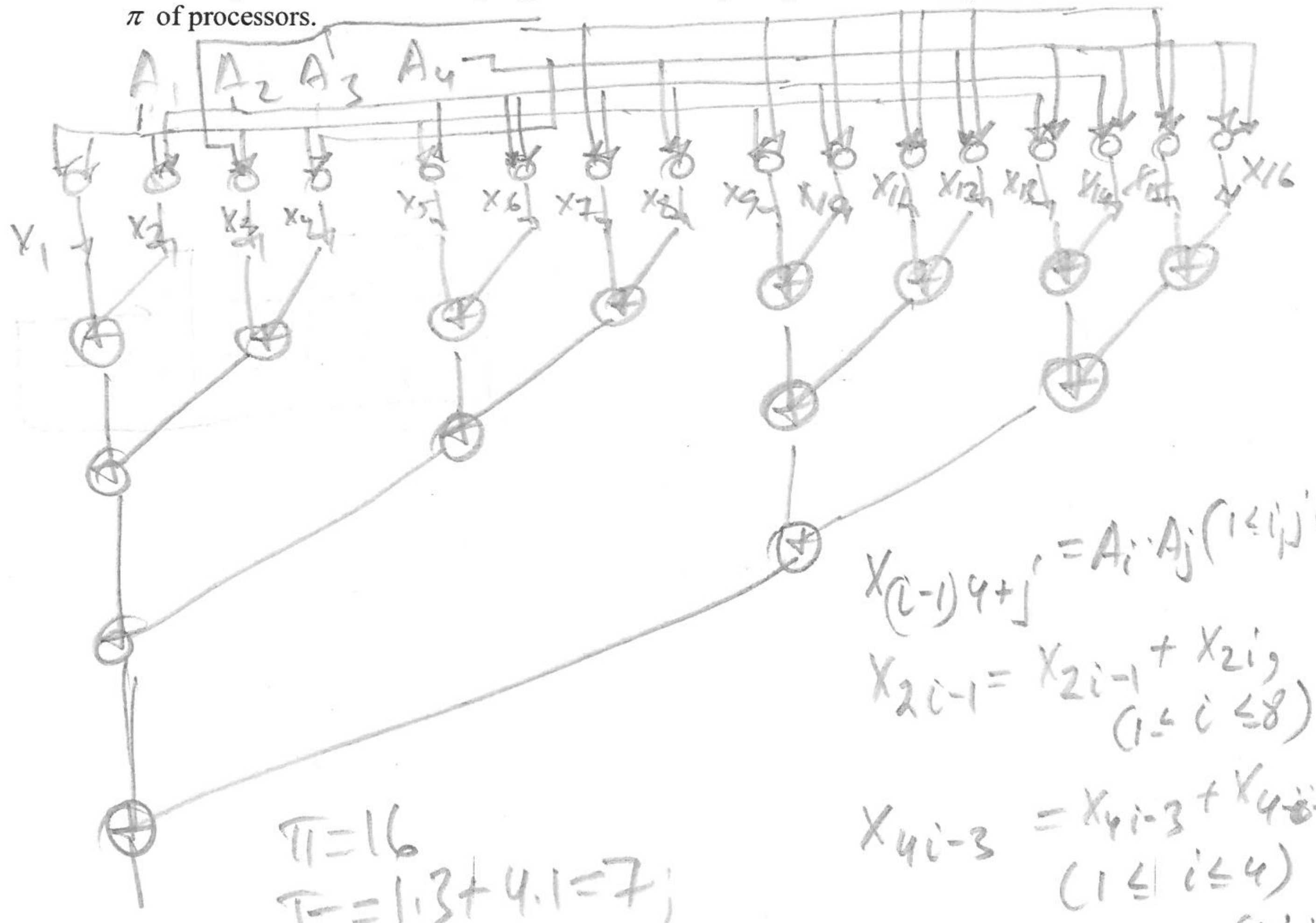
*Totally 7 tasks, 45 points, 9 pages*

| Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 | Task 7 | Total |
|---|---|---|---|---|---|---|---|
| 6 | 6 | 7 | 7 | 7 | 6 | 6 | 45 |
| | | | | | | | |

**Task 1. (6 points).** Using associativity, draw the flattest possible dependence graph for the following calculation

$$\cdot\sum_{i=1}^{4}\sum_{,j=1}^{4} A_i A_j$$

Write SIMD pseudocode for its calculation. Assume that addition takes 1 time unit, and multiplication takes 3 time units. What is the minimal number $\pi$ of processors providing maximal performance for that program? Estimate speedup and efficiency for that number $\pi$ of processors.



$X_{(i-1)4+j} = A_i \cdot A_j \ (1 \le i, j \le 4)$

$X_{2i-1} = X_{2i-1} + X_{2i}, \ (1 \le i \le 8)$

$X_{4i-3} = X_{4i-3} + X_{4i-1}, \ (1 \le i \le 4)$

$X_{8i-7} = X_{8i-7}, \ (1 \le i \le 2)$

$X_1 = X_1 + X_9$;

$\pi = 16$

$T_\pi = 1 \cdot 3 + 4 \cdot 1 = 7$;

$T_1 = 16 \cdot 3 + 15 \cdot 1 = 63$;

$S_\pi = \frac{T_1}{T_\pi} = \frac{63}{7} = 9$;

$E_\pi = \frac{S_\pi}{\pi} = \frac{9}{16}$;

**Task 2. (6 points).** Consider the code below

```
X[i]=c[i], (1<=i<=n);
For j:=1 step 1 until n-1
  X[i]:=x[i]+A[I,j]*x[j], (j+1<=i<=min(j+m,n));
```

What problem is solved by the code?
Assuming a SIMD computer with the distributed memory has N=4=n, m=n-1=3 processing elements, show memory allocation for the code. Trace the code. Assume C=(1,2,3,4), A=$\begin{vmatrix} 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 2 & 1 & 0 \end{vmatrix}$. Rewrite the code using y=broadcast(x) operation for broadcasting a scalar x to the local variable y of all the processing elements.

The problem is $X = C + AX$

| | $PE_1$ | $PE_2$ | $PE_3$ | $PE_4$ |
|---|---|---|---|---|
| C | 1 | 2 | 3 | 4 |
| A | 0 | 3 | 2 | 4 |
| | 0 | 0 | 1 | 2 |
| | 0 | 0 | 0 | 1 |
| | 0 | 0 | 0 | 0 |
| X | | | | |
| t | | | | |

| j | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | | 5 | 5 | 8 |
| 2 | | | 10 | 18 |
| 3 | | | | 28 |

```
$x_i = c_i$, $(1 \le i \le n)$
for j = 1 step 1 until n-1
  $t_i$ = broadcast$(x_j)$, $(1 \le i \le n)$
  $x_i = x_i + A_{ij} \cdot t_i$, $(j+1 \le i \le \min(j+m, n))$
```

**<u>Task 3 (7 points)</u>** Write SIMD assembly code to calculate the scalar product of two vectors, $(X,Y) = \sum_{i=1}^{N} X_i Y_i$. Assume the number of the each vector components is N=4, and the number of processing elements is N=4. Specify memory layout (distribution of vectors over memory blocks). Give necessary explanations. SIMD assembly language instructions are on the last page.

```
X   BSS   1.4
Y   BSS   1.4
    lod   X
    mul   Y
    route -1
    radd
    rout  -2
    radd
```

$(X,Y)$ is held in every $A_i$, $i=1,4$

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| X | | | | |
| Y | | | | |

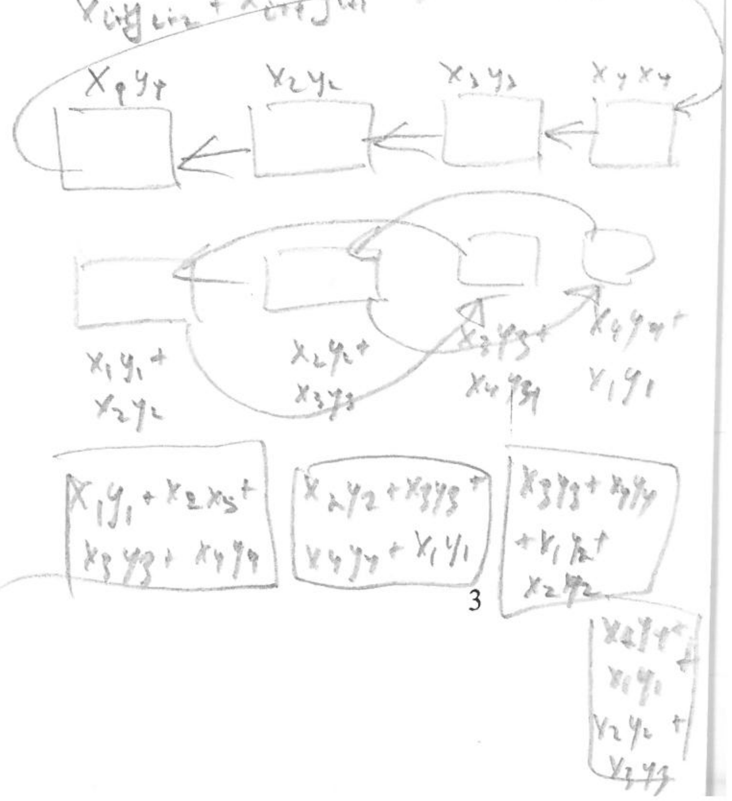
X in $A_i$

$X_i y_i$ in $A_i$

now $X_{i+1} y_{i+1} \rightarrow PE_i$

$X_i y_i + X_{i+1} y_{i+1}$

$X_{i+2} y_{i+2} + X_{i+3} y_{i+3} \rightarrow PE_{i-1}$



$X_1 y_1$
$X_2 y_2$
$X_3 y_3$
$X_4 x_4$
$X_1 y_1 + X_2 y_2$
$X_2 y_2 + X_3 y_3$
$X_3 y_3 + X_4 y_4$
$X_4 y_4 + X_1 y_1$
$X_1 y_1 + X_2 x_3 + X_3 y_3 + X_4 y_4$
$X_2 y_2 + X_3 y_3 + X_4 y_4 + X_1 y_1$
$X_3 y_3 + X_4 y_4 + X_1 y_1 + X_2 y_2$
$X_4 y_4 + X_1 y_1 + X_2 y_2 + X_3 y_3$

**Task 4 (7 points)** Consider the code below

```
V     BSS          1*8                  input array
V'    BSS          1*8                  output array
      lod    V                          Get the input vector
      movA toR                           and ready it for routing
      route  +1                         All PEs send their data right
      ldxi   mask, =(01111111)           but only the last 7
      mask                                perform
      radd                                  the add of the received data
      movAtoR                           Results of
      ldxi   mask, =(11111111)           the add are
      mask                                sent two steps
      route  +2                           to the right by all
      ldxi   mask, =(00111111)          Received values are
      mask                               used only by
      radd                                the last 6 PEs
      movAtoR                           Results of
      ldxi   mask, =(11111111)           the add are
      mask                                sent four steps
      route  +4                           by all
      ldxi   mask, =(00001111)          Only 4 additions
      mask                               are done
```

(ldxi mark, =(11111111)
mark

enable all PE's

radd in the last step

Trace the code assuming V=(5,3,2,1,1,2,3,4). Write instruction(s) providing saving in the memory (to V') of the result calculated. What problem is solved by the code?

The problem is the prefix problem; V' = 5, 8, 10, 11, 12, 14, 17, 21

Save result back to memory

S to V'

| P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 |
|---|---|---|---|---|---|---|---|
| V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 |
| 5 | 3 | 2 | 1 | 1 | 2 | 3 | 4 |

| mask | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 5 | 3 | 2 | 1 | 1 | 2 | 3 | 4 | ← load ✓ | | | | | | | | |
| | | | | | | | mov A to R → | | | 5 | 3 | 2 | 1 | 1 | 2 | 3 | 4 |
| | | | | | | | route+1 → | | | 4 | 5 | 3 | 2 | 1 | 1 | 2 | 3 |
| 01111111 | ldxi mark,=(01111111), mark | | | | | | | | | | | | | | | | |
| | 8 | 8 | 5 | 3 | 2 | 3 | 5 | 7 | radd | | | | | | | | |
| | | | | | | | mov A to R | | | | 8 | 5 | 3 | 2 | 3 | 5 | 7 |
| 11111111 | ldxi mark,=(11111111), mark | | | | | | | | | | | | | | | | |
| | | | | | | | route +2; | | | 5 | 7 | 4 | 8 | 5 | 3 | 2 | 3 |
| 00111111 | ldxi mark,=(00111111); mark | | | | | | | | | | | | | | | | |
| | | | 9 | 11 | 7 | 6 | 7 | 10 | radd | | | | | | | | |
| | | | | | | | mov A to R; | | | | | 9 | 11 | 7 | 6 | 7 | 10 |
| 11111111 | ldxi mark,=(11111111); mark | | | | | | | | | | | | | | | | |
| | | | | | | | rout+4; | | | 7 | 6 | 7 | 10 | 5 | 7 | 9 | 11 |
| 00001111 | ldxi mark,='00001111'; mark | | | | | | | | | | | | | | | | |
| | 5 | 8 | 9 | 11 | 12 | 13 | 16 | 21 | radd | | | | | | | | |
| | V | V | | ✓ | | | | ✓ | | | | | | | | | |

**Task 5 (7 points)** Consider FORTRAN-90 statement A(5:10)=A(1:6). If before the operation, A=(1,2,3,4,5,6,7,8,9,10), what is A after the operation. Write equivalent C-like pseudocode. Give necessary explanations.

$A_{old}$ = 1 2 3 4 5 6 7 8 9 10

$A_{new}$ = 1 2 3 4 1 2 3 4 5 6

```
for (i = 5; i < 11; i++)
    $A_i = A_{i-4}$;
```

| i | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ | $A_9$ | $A_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 5 | 1 | 2 | 3 | 4 | 1 | 6 | 7 | 8 | 9 | 10 |
| 6 | 1 | 2 | 3 | 4 | 1 | 2 | 7 | 8 | 9 | 10 |
| 7 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 8 | 5 | 10 |
| 8 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 9 | 10 |
| 9 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 10 |
| 10 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 |

Thus, C-code does not comply with the Fortran expectations. To solve the problem, a temporary memory shall be used

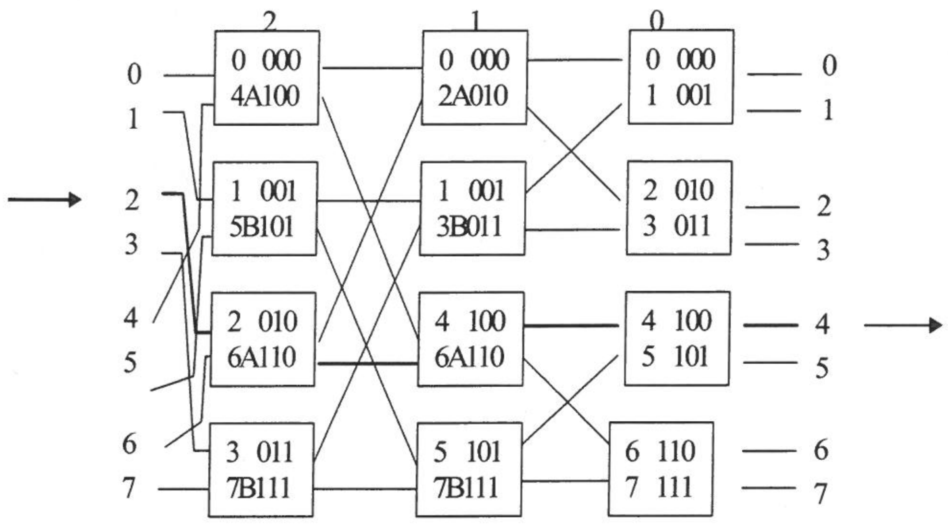
```
for (i = 1, 6) $temp_i = A_i$;
for (i = 5, 10) $A_i = temp_{i-4}$;
```

**<u>Task 6 (6 points)</u>** Assume, there are two processes, Producer and Consumer, share a common buffer, B, for keeping one data item. Initially, B is empty. Producer, generates data items, and writes them into B, if it is empty. After writing into, the buffer becomes full. When B is full, a data item from it can be read by Consumer, thus making B empty. Write a semaphore solution for Producer and Consumer synchronization so that Producer and Consumer are mutually excluded when accessing to B. Give necessary explanations.

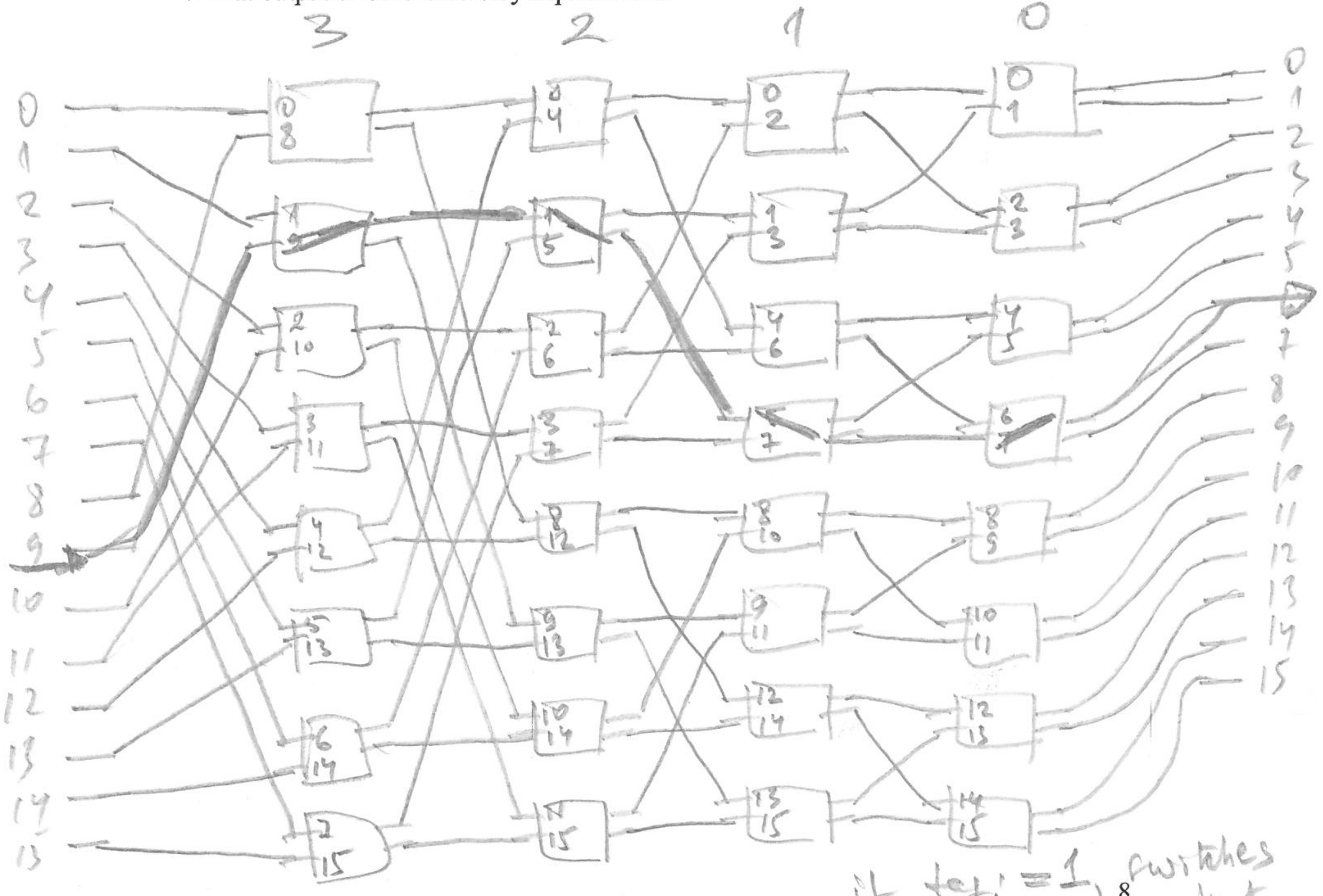
```
Semaphore S=1; buffer B; // empty
Producer:

        data preparing;
        P(S); // acquire semaphore
          if(B is empty) {
            Write to B;
             B is full;

          }
        V(S); // release semaphore


Consumer:

        preparing to consume data;
        P(S); // acquire semaphore
          if (B is full) {
            read from B;
            B is empty;

          }
        V(S); // release semaphore
```

**Task 7 (6 points)**

Consider the following hypercube interconnection network



Draw a 16x16 interconnection switch and define for it a routine tag for connecting input 9 with output 6. Give necessary explanations.



$I = 9 = 1001$

$O = 6 = {}^{+}\,0110$

$1111$ – tag

if $tag_i = 1$, switches output

if $tag_i = 0$, forward to the same output

| Instruction | Assembly code | | Action |
|---|---|---|---|
| Vector load | lod | a, index, i | $S_k \rightarrow A_k \leftarrow M_k[pea_k], (0 \le k \le N-1)$ |
| Vector store | sto | a, index, i | $S_k \rightarrow M_k[pea_k] \leftarrow A_k, (0 \le k \le N-1)$ |
| Vector add | add | a, index, i | $S_k \rightarrow A_k \leftarrow A_k + M_k[pea_k], (0 \le k \le N-1)$ |
| Vector subtract | sub | a, index, i | $S_k \rightarrow A_k \leftarrow A_k - M_k[pea_k], (0 \le k \le N-1)$ |
| Vector multiply | mul | a, index, i | $S_k \rightarrow A_k \leftarrow A_k \times M_k[pea_k], (0 \le k \le N-1)$ |
| Vector divide | div | a, index, i | $S_k \rightarrow A_k \leftarrow A_k / M_k[pea_k], (0 \le k \le N-1)$ |
| Broadcast | bcast | index | $S_k \rightarrow R_k \leftarrow R_{X[\text{index}]}, (0 \le k \le N-1)$ |
| Move PE register | mov $\left\{\begin{matrix}A\\R\\I\end{matrix}\right\}$ | to $\left\{\begin{matrix}A\\R\\I\end{matrix}\right\}$ | $S_k \rightarrow \left\{\begin{matrix}A_k\\R_k\\I_k\end{matrix}\right\} \leftarrow \left\{\begin{matrix}A_k\\R_k\\I_k\end{matrix}\right\}, (0 \le k \le N-1)$ |
| Register add | radd | | $S_k \rightarrow A_k \leftarrow A_k + R_k, (0 \le k \le N-1)$ |
| Register subtract | rsub | | $S_k \rightarrow A_k \leftarrow A_k - R_k, (0 \le k \le N-1)$ |
| Register multiply | rmul | | $S_k \rightarrow A_k \leftarrow A_k \times R_k, (0 \le k \le N-1)$ |
| Register divide | rdiv | | $S_k \rightarrow A_k \leftarrow A_k / R_k, (0 \le k \le N-1)$ |

Figure 3-6
Set of vector instructions for an SIMD machine.

| Instruction | Assembly code | | Action |
|---|---|---|---|
| Load index | ldx | ix2, a.index | $X[ix2] \leftarrow M[ca];$ |
| Store index | stx | ix2, a.index | $M[ca] \leftarrow X[ix2];$ |
| Load index immediate | ldxi | ix2, a.index | $X[ix2] \leftarrow ca;$ |
| Increment index | incx | ix2, a.index | $X[ix2] \leftarrow X[ix2] + ca;$ |
| Decrement index | decx | ix2, a.index | $X[ix2] \leftarrow X[ix2] - ca;$ |
| Multiply index | mulx | ix2, a.index | $X[ix2] \leftarrow X[ix2] \times ca;$ |
| Load data | cload | a, index | $A \leftarrow M[ca];$ |
| Store data | cstore | a, index | $M[ca] \leftarrow AC;$ |
| Compare and branch | cmpx | index.ix2, a | $(X[index] \le X[ix2]) \rightarrow PC \leftarrow ca;$ |

Figure 3-7
SIMD control unit instruction set.

| Instruction | Assembly code | | Action |
|---|---|---|---|
| Compare < | clt | a, index, i | $S_k \rightarrow (M_k[\text{pea}_k] < A_k) \rightarrow mask(k) \leftarrow 1, (0 \le k \le N-1);$ |
| Compare = | ceq | a, index, i | $S_k \rightarrow (M_k[\text{pea}_k] = A_k) \rightarrow mask(k) \leftarrow 1, (0 \le k \le N-1);$ |
| Compare > | cgt | a, index, i | $S_k \rightarrow (M_k[\text{pea}_k] > A_k) \rightarrow mask(k) \leftarrow 1, (0 \le k \le N-1);$ |
| ... | ... | | ... |
| Mask PEs | mask | | $S_k \leftarrow mask(k), (0 \le k \le N-1);$ |
| Save enables | stmask | | $\text{mask}(k) \leftarrow S_k, (0 \le k \le N-1);$ |
| CU move | move $\left\{\begin{matrix}\text{i}\\\text{m}\\\text{AC}\end{matrix}\right\}$ | to $\left\{\begin{matrix}\text{i}\\\text{m}\\\text{AC}\end{matrix}\right\}$ | $\left\{\begin{matrix}X[j]\\mask\\\text{AC}\end{matrix}\right\} \leftarrow \left\{\begin{matrix}X[j]\\mask\\\text{AC}\end{matrix}\right\};$ |

Figure 3-11
Cooperative SIMD instructions involving the mask.

| Instruction | Assembly code | Action |
|---|---|---|
| Broadcast *AC* | cbcast | $S_k \rightarrow R_k \leftarrow AC, (0 \le k \le N-1);$ |

Figure 3-8
A simple cooperative instruction.

| Instruction | Assembly code | Action |
|---|---|---|
| Vector loop | `vecloop ix1, ix2, adr` | $(X[ix1] < X[ix2]) \rightarrow$ $(X[ix1] \leftarrow X[ix1] + N)$; *next* $(X[ix2] < X[ix1]) \rightarrow$ $\text{mask} \leftarrow \text{mask} \wedge \Pr(X[ix2] \bmod N)$; *next* $\text{PC} \leftarrow \text{adr})$; |

Figure 3-18
A vector looping instruction.