Vector Processors Motivations:

- Cannot increase performance with deeper pipelin
  because:
  -clock cycle time limitation (latch delay)
  -increase dependences with deeper pipeline

- Cannot increase performance by multiple is-
  suing because:
  -limitation of Instruction Fetch and decode
  rate (memory bottelneck)
  -Not enough ILP

Concept of Vector Processing

Provide operations that work on vectors.
Vectors are alinear array of numbers.
One vector operation could add two 64 elements

Advantages of Vector Operations:

- Operations on each two elements do not depend on previous results (No data hazards)
- Need single vector instruction to do the work of multiple instructions. This reduces Instruction memory bandwidth.
- Could use Interleaved memory to reduce latency cost of main memory and fetch data elements.
  Multiple elements could be accessed from multi-bank memory in single access.
- Reduce Control hazards because an entire loop is replaced by single vector operation.

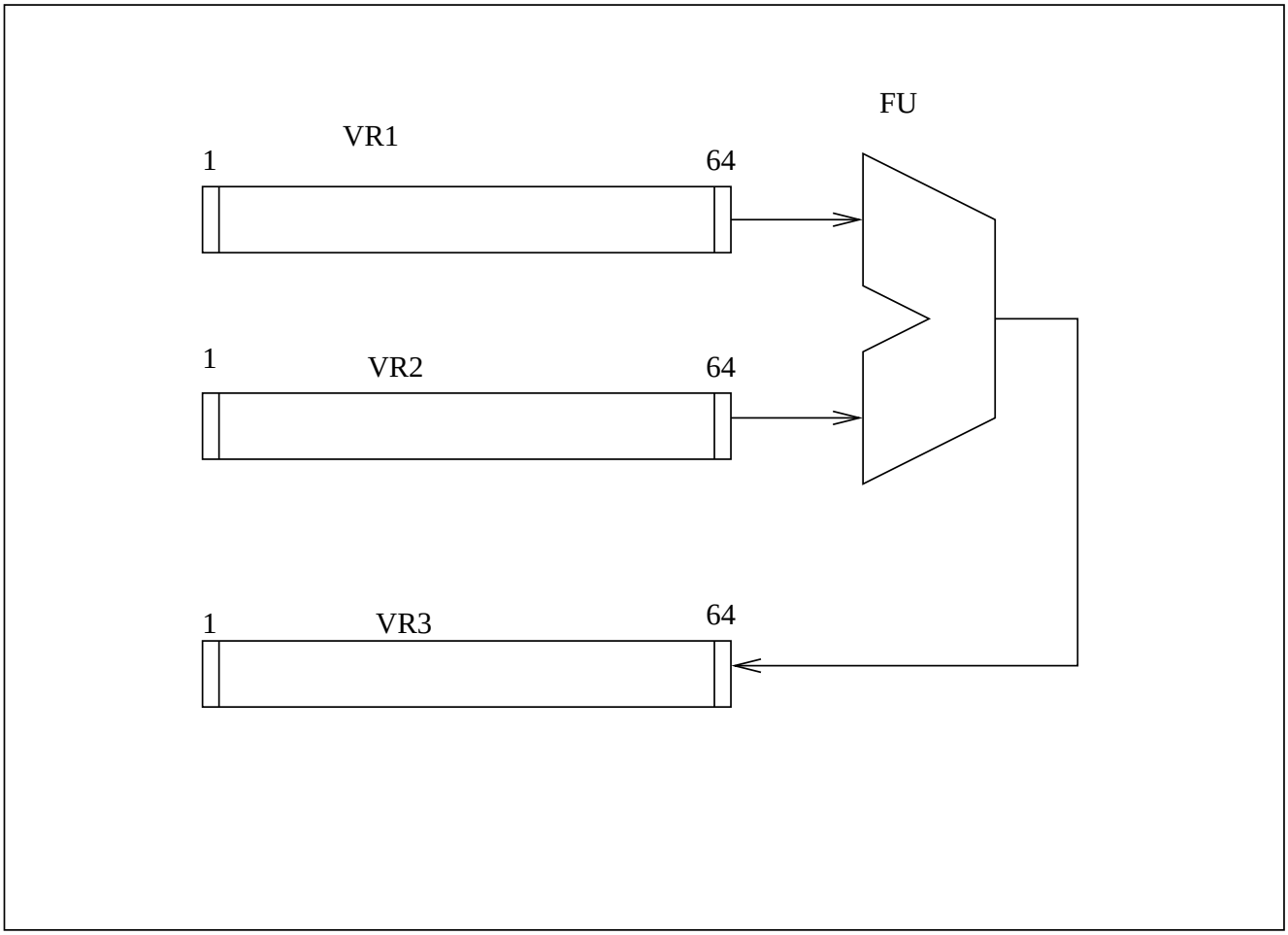
The Basic Vector Architecture

OP VR3, VR2, VR1 ; 64 Operation

VR3(i) = VR2(i) OP VR1(i)

Scalar operation:

R3 = R2 OP R1



The Vector Architecture Components:

- 1-Pipelined function units for vector operations.
  Clock cycle time= latency pipeline depth
  Vector Function Units can use very deep pipeline (no hazards), this increases clock frequency.

- 2-Multiple Vector Function Units:
  Multiply, ADD Subtract, Divide, Integer, Logical.
  This allows multiple vector operations

- 3-It uses Scalar Unit (same as the basic pipeline)

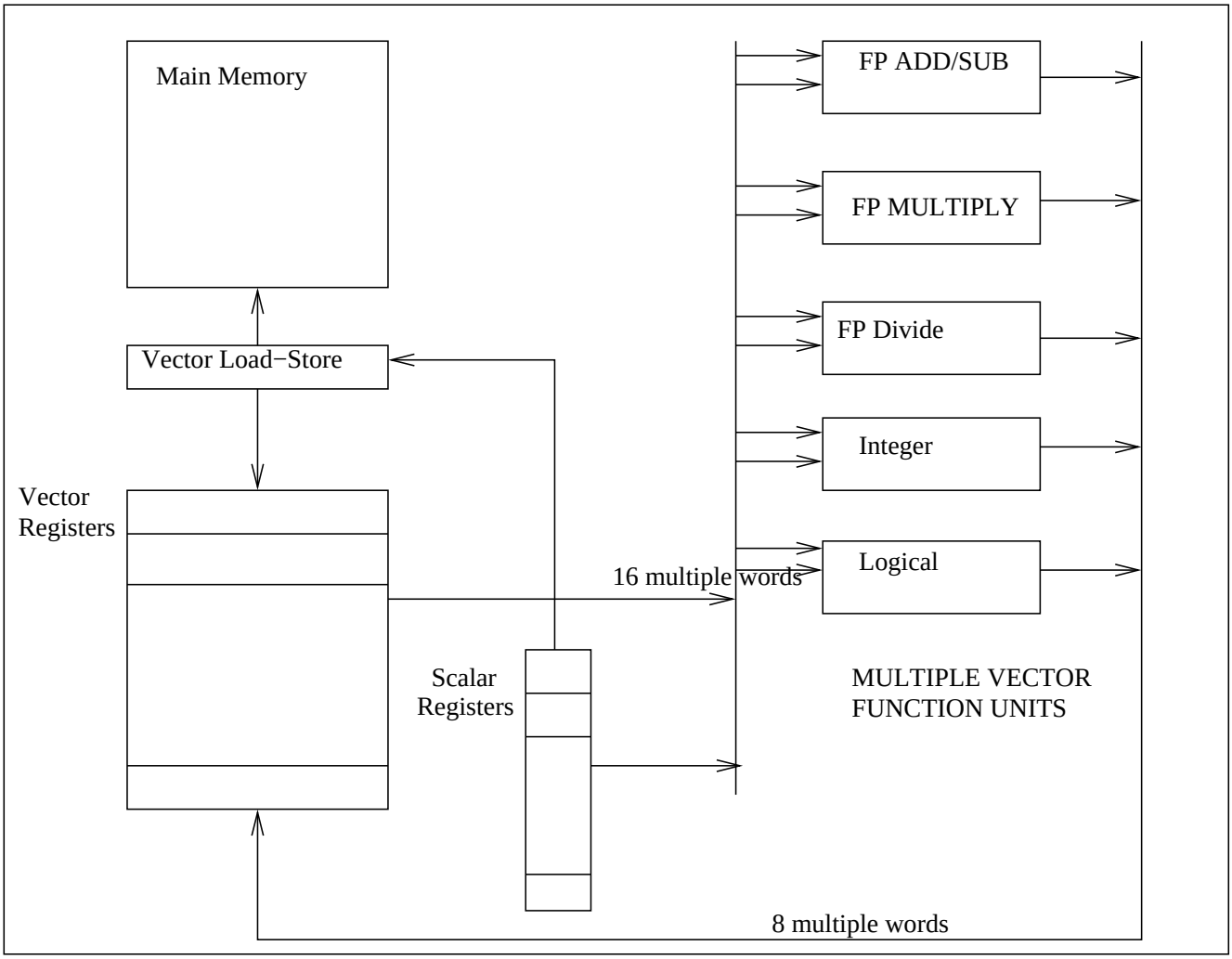
- 4-Vector Register File
  Each Vector register consists of fixed length bank holding a vector. (DLXV= 64 elements)

The Vector Architecture Components:

- 5- Multiple read/write ports for the vector register file (16 read ports and 8 write).

- 6-Vector Load-Store Unit:
  Load or Store Vector from memory.
  Fully pipelined, words can be moved fromto memory at a rate of 1 word/clock cycle (need Interleaved memory).

- 7-Scalar Registers:
  To compute address for loadstore unit and the 32 GPR, 32 FP registers used in DLX.



Cray T-90 , 1996

Clock rate= 500 MHz

Vector registers =8

each register = 128 X 64 bits

8 FP add, 1 FP Multiply, 4 LoadStore, 1 Int AddSub , 2 Logial, 1 Shift, 1Reciprocal

DLXV Instruction Set

```
ADD V1, V2, V3 ;    V1(i) = V2(i) + V3(i)
ADDSV V1, F0, V2 ;  V1(i) = F0 + V2(i)
SUBV V1, V2, V3 ;   V1(i0 = V2(i) - V3(i)
MULTV V1, V2, V3 ;  V1(i) = V2(i) xV3(i)
DIVV V1, V2, V3;    V1(i) = V2(i) /V3(i)
LV V1, R1 ;         V1(i) = M [R1 +i]
SV R1, V1 ;         M[R1 +i] = V1(i)
LVWS V1, (R1,R2) ;  V1(i) = M[R1 + ixR2]
SVWS (R1,R2), V1 ;  M[R1 +ixR2] =V1(i)
SEQV V1, V2; if(V1(i)==V2(i))
                VM(i) =1  where VM is mask Reg
MOVI2S VLR, R1; VLR=R1 strip mining
```

Example: Compare the performance of DLX to DLXV for DAXPY routine assume vector register= 64 elements

$$Y = A \times X + Y$$

DLX Code:

```
     LD F0,B R
     ADDI R4, RX, #512 ; R4= Last address
lOOP LD F2, 0(RX); F2=X[i]
     MultD F2, F0, F2; F2=A.X[i]
     LD F4, 0(Ry) ; F4= Y[i]
     ADDD F4, F4, F2 ; F4=A.X[i]+y[I]
     SD 0(Ry), F4    ; Y[i]=A.X[i] + Y[i]
     ADDI Rx, Rx, #8 ; i+1 for X
     ADDI Ry, Ry, #8 ; i+1 for Y
     SUB R20, R4, Rx ; R20=64 -I
     BNZ R20,lOOP
```

number of instructions = 2 + 64x9=578

Total time = 578 cycles with no hazards

```
DLXV Code:
LD F0, A;   1 cycle
LV V1, Rx ; V1[i]= X[i] 64 cycles
MULTV V2, V1, F0 ; V2[i]=AX[i] 64 cycles
LV V3, Ry ; V3[i]=Y[i]  64 cycles
ADDV V4, V2, V3 ; V4[i]=AX[i] + Y[i] 64 cycles
SV Ry, V4  ; Y[i]=AX[i] + Y[i]   64 cycles

only 6 Instructions
Instruction memory bandwidth =578/6 = 1/100
No control hazards or overhead (BNZ, SUB, ADDI)

total time 321 cycles
```

Applications For Vector Operations

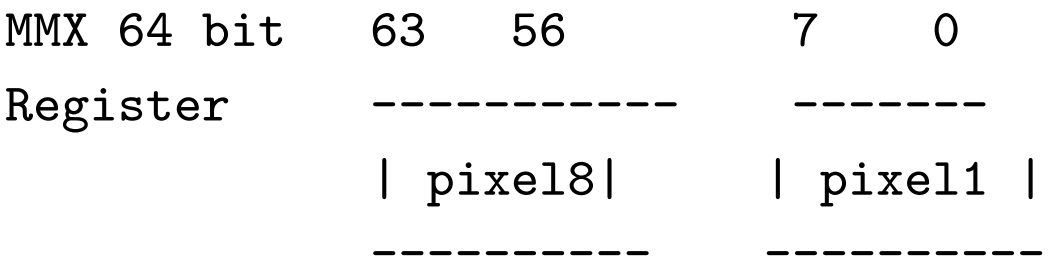
1-Multimedia Processing (Compress, graphics,..)
2- Standard Scientific computing (Matrix Multiplication, FFT, CONVL, SORT)
3-Database (data mining, image/video serving)
4-Operating SystemsNetworking (memcpy,..)
5-Speech, handwriting recognition

EXAMPLE: MMX

MMX Technology is a set of Instructions for multimedia and communication applications.

- Single Instruction Multiple Data (SIMD) technique for vector operations
- 57 New Instructions
- Uses eight 64 bit MMX Registers
- four new data types:
  - 1-Packed byte - 8 bytes packed into 64 bit
  - 2-Packed word - four 16 bit words
  - 3-Packed double words - two 32 bit words
  - 4- Quadword - one 64 bit word

Graphics Pixel Data use 8 bits integers, 8 of these pixels can be packed together and moved to MMX register.
When MMX instruction executes, it operates on all the 8 Pixels at once (SIMD) to perform arithmetic or logic operations.



MMX is integrated to Intel architecture
MMX is fully compatible with exisisting applications and operating system by aliasing its registers and stste on FP registers and state.

MMX Instruction Set It operates on byte (B), word (W), double word (DW) or quad word (QW).

- Basic arithmetic operations (add, sub, multiply, shift, and multiply-add)
  PADD[B,W,D] = add with wrap around
  PMULHW = packed multiply high on words

- Comparison operations:
  PCMPEQ[B,W,D] = packed compare for equality

- Conversion between data types:
  PACKUSWB = pack words into bytes (unsigned)

- Data transfer
  MOV[D,Q] = move [double word, QW] to MMX register or from MMX register

- shift
  PSLL[W,D,Q] = packed shift left logical by Immediate

Examples

```
PADD[W]:
        ------------------------------
MMX1    a3          a2        a1      FFFF
        ------------------------------
        +           +         +        +
MMX2    ------------------------------
        b3          b2        b1      8000
        ------------------------------
=       ------------------------------
        a3+b3     a2+b2     a1+b1 7FFF  warp around
        ------------------------------

PMADDWD: 16bX16b ---> 32 b
        ------------------------------
MMX1    a3        a2        a1        a0
        ------------------------------
        X          X        X         X
MMX2    ------------------------------
        b3        b2        b1        a0
        ------------------------------=
=       ------------------------------
        a3Xb3+ a2Xb2     a1Xb1+ a0Xb0
        ------------------------------
```

Examples

```
PCMPGT[W]:
        ----------------------
MMX1    23     45     16     39
        ---------------------
        gt?    gt?    gt?    gt?
MMX2    ---------------------
        31      7     16     67
        ---------------------
=       ---------------------
        0000   FFFF   0000   0000
        ---------------------
```

Application Example: Chroma Keying

Conditional selection of pixels and overlay on a baclground

TV Weatherman overlaid on the Image of waether map

Assume a person picture to overlay it on a picture of spring blossom

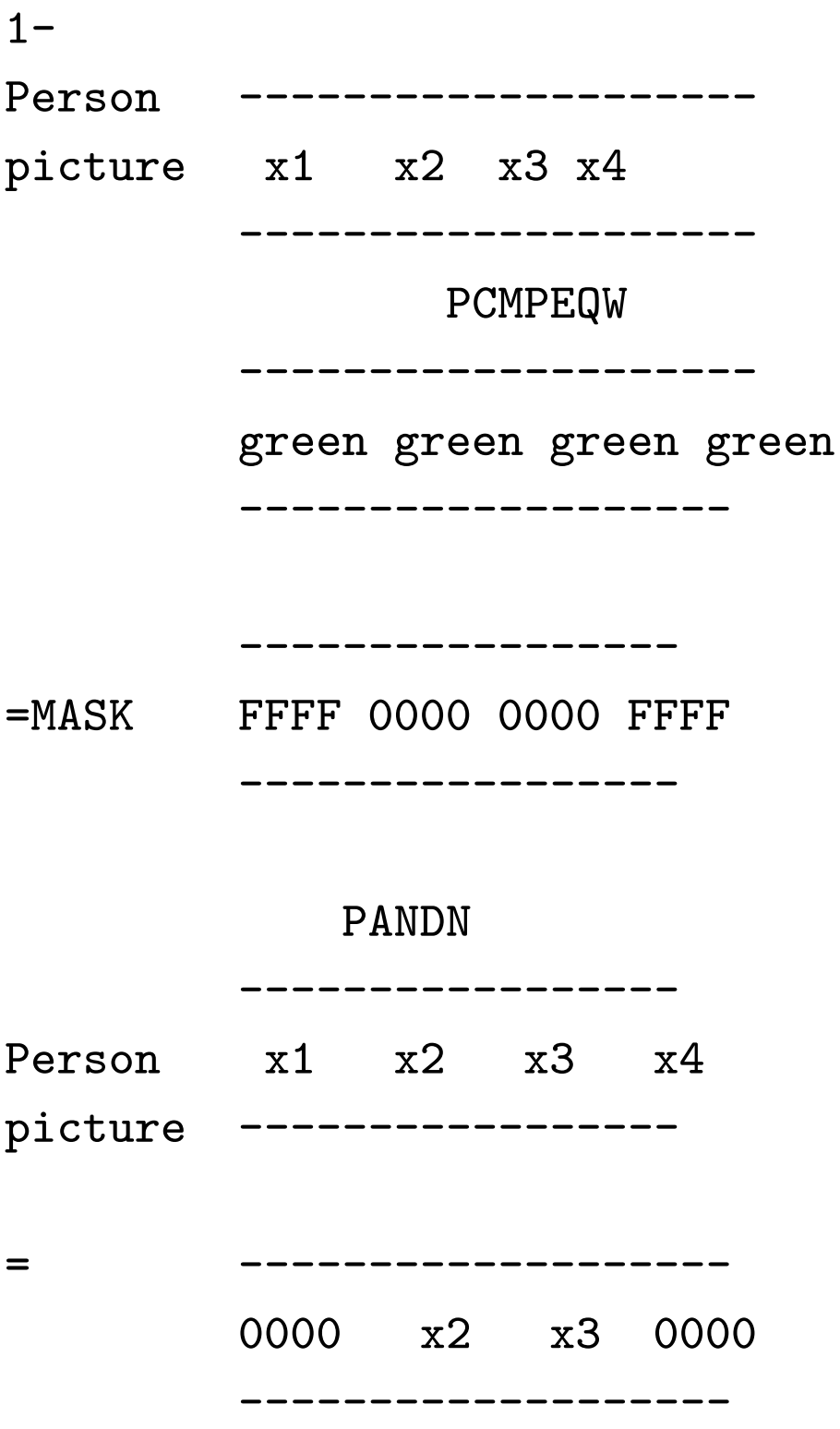
1-Assume that person picture has green background

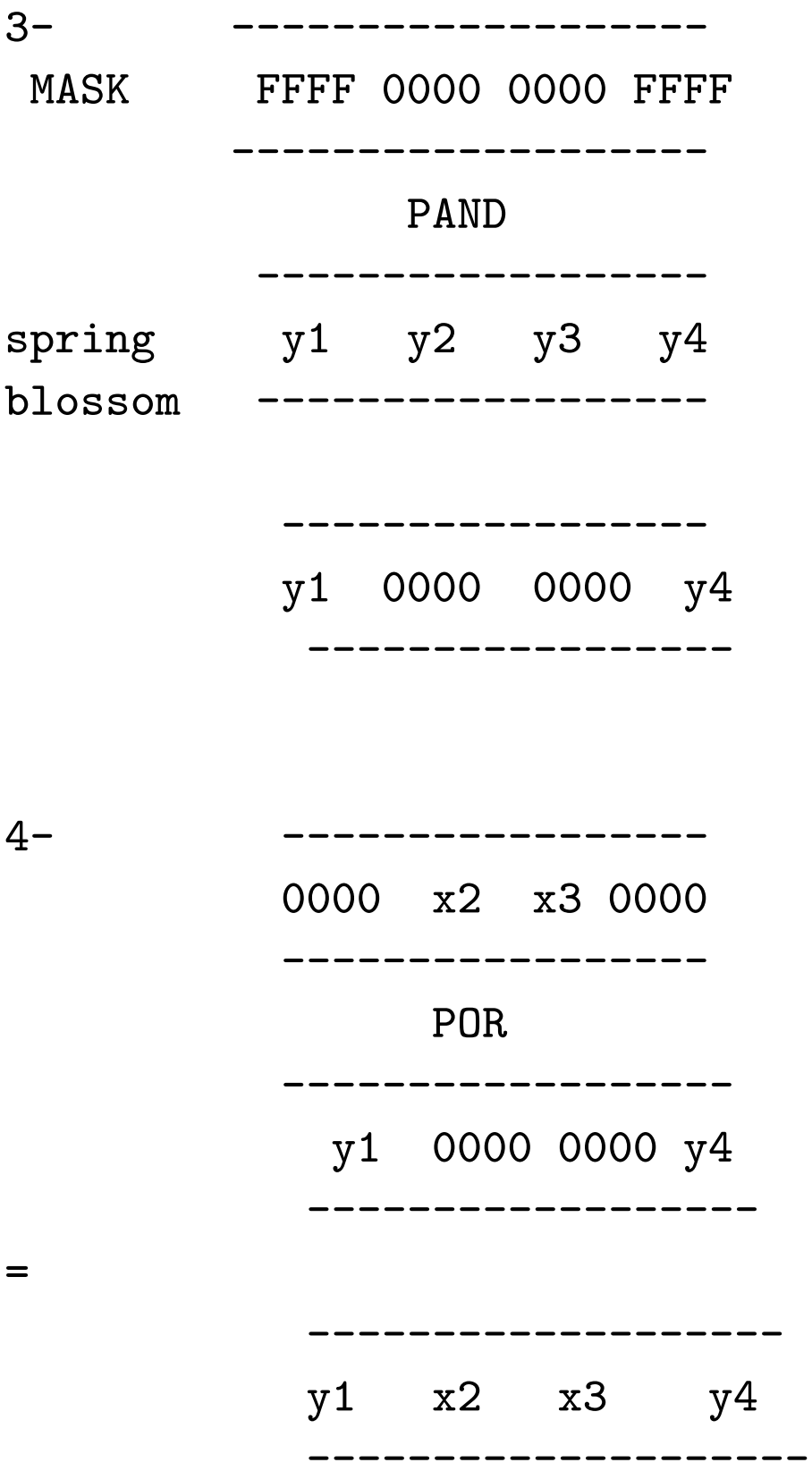
2-Compare each pixel of person picture with pixel of green colour using PCMPEQ (mask for person's face)

3-USE AND NOT instruction between MASK and person's picture, get person's face only

4-Use AND instruction between MASK and spring blossom, get the spring blossom only in place of green background

Use OR instruction for results of 3 and 4, get person's picture with spring blossom in the background


1-
Person
picture
x1
x2
x3
x4
PCMPEQW
green green green green
=MASK
FFFF 0000 0000 FFFF
PANDN
Person
picture
x1
x2
x3
x4
=
0000
x2
x3
0000


3-
MASK
FFFF 0000 0000 FFFF
PAND
spring
blossom
y1 y2 y3 y4
y1 0000 0000 y4
4-
0000 x2 x3 0000
POR
y1 0000 0000 y4
=
y1 x2 x3 y4