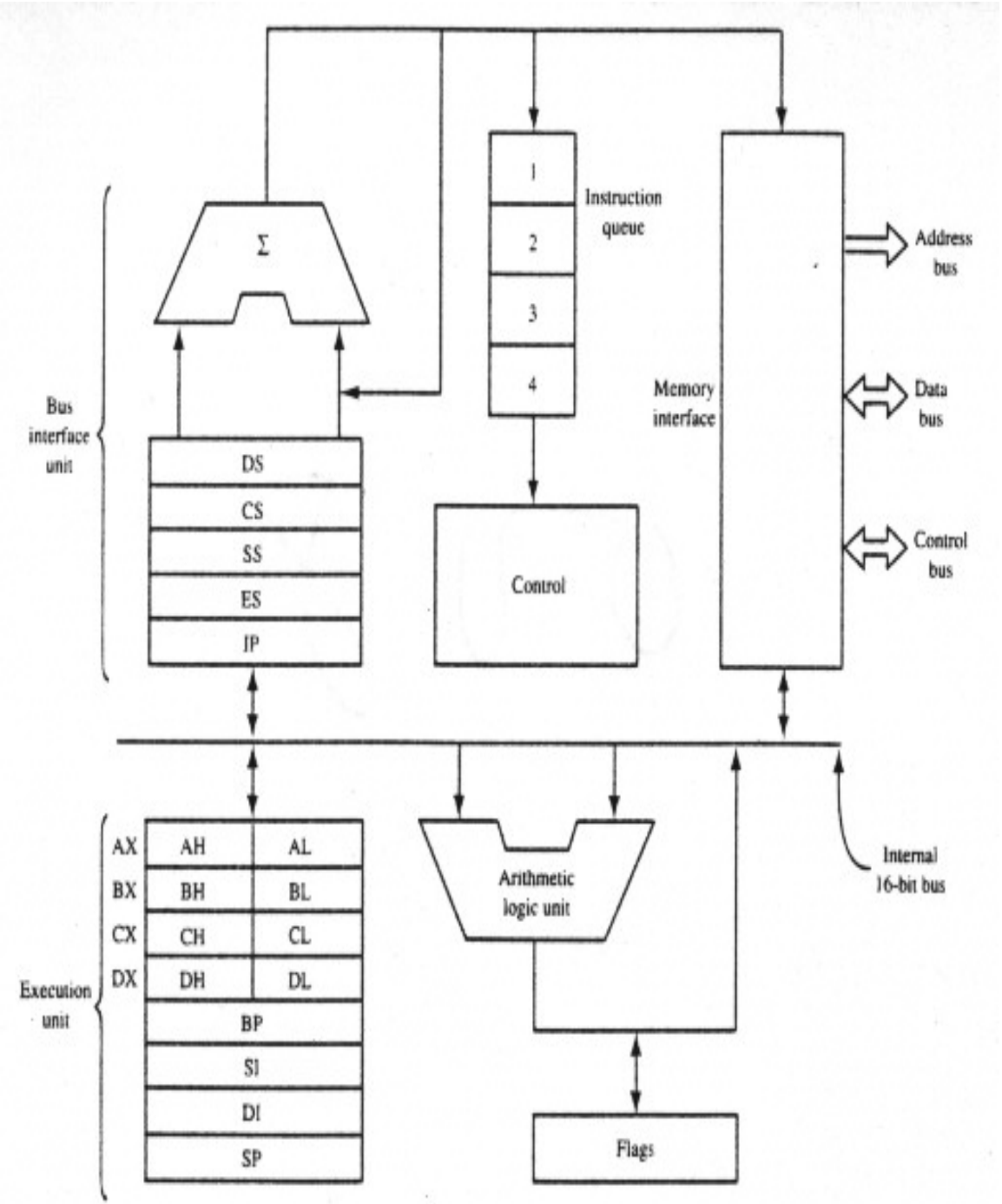
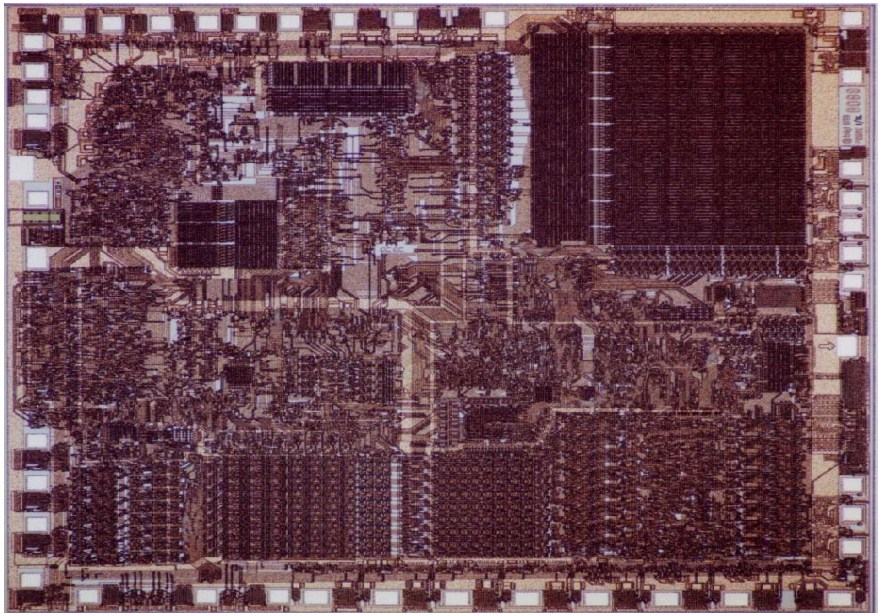
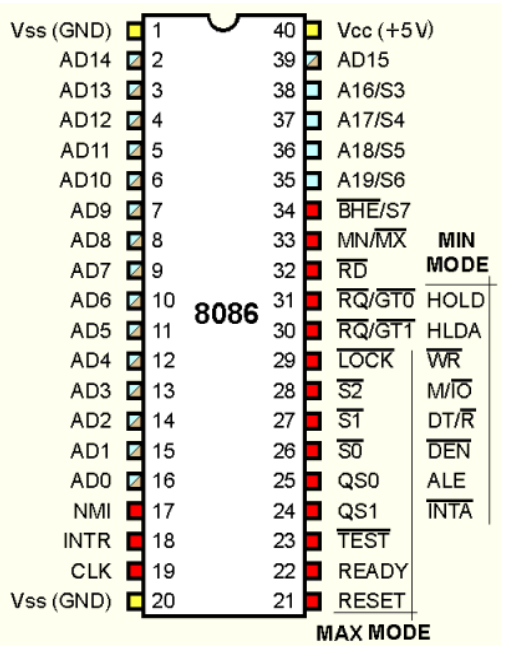
Introduction to Intel IA-32 Architecture

Intel 8086 “Multi-cycle” Architecture (June 1979, 4.77 MHz)







IA-32 Intel Assembler Instructions

	MIPS		Intel IA-32	C Language
Register	`addi`	`$t0,$t1,0`	`mov eax,ebx`	`;eax=ebx`
Increment	`addi`	`$t0,$t0,1`	`inc eax`	`;++eax`
Decrement	`addi`	`$t0,$t0,-1`	`dec eax`	`;--eax`
Immediate	`addi`	`$t0,$0,10`	`mov eax,10`	`;eax=10`
Direct addressing	`la`	`$a0,n`	`mov eax,n`	`;eax=n`
	`lw`	`$t0,0($a0)`		
	`n`	`.word 3`	`n dw 3`	`;int n`
Indirect addr.	`lw`	`$t0,0($t1)`	`mov eax,[ebx]`	`;eax=*ebx`
..with offset	`lw`	`$t0,8($t1)`	`mov eax,[ebx+8]`	`;eax=*(ebx+8)`
Load byte	`la`	`$a0,c`	`xor eax,eax`	`;eax=0`
	`lb`	`$t0,0($a0)`	`mov al,c`	`;unsigned`
	`c`	`.byte 4`	`c db 4`	`;char c`

Intel machine language example

Note: variable size instructions

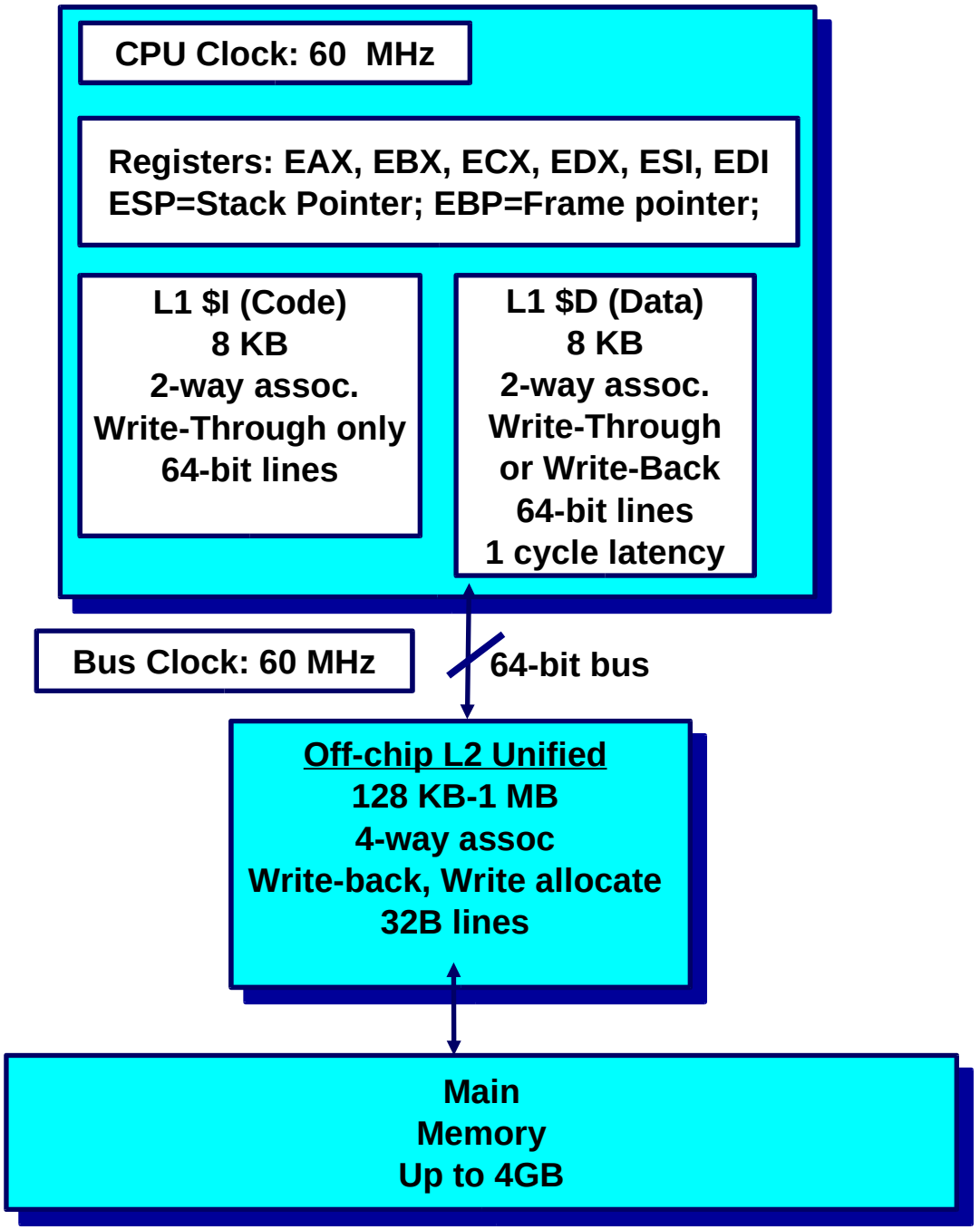
```
Address   machine code              8086 assembler
0000      000A                      a dw 10
0002      00                        b db ?
                                    .code
0000      8B DF                     mov bx,di
0002      8A F9                     mov bh,cl
0004      8B 1E 0000 R              mov bx,a
0008      8A 26 0002 R              mov ah,b
000C      8B 12                     mov dx,[si][bp]
000E      A0 0002 R                 mov al,b
0011      8A 26 0002 R              mov ah,b
0015      BB 0003                   mov bx,3
0018      B1 03                     mov cl,3
001A      C7 06 0000 R 0064         mov a,100
0020      C6 06 0002 R FF           mov b,255
```

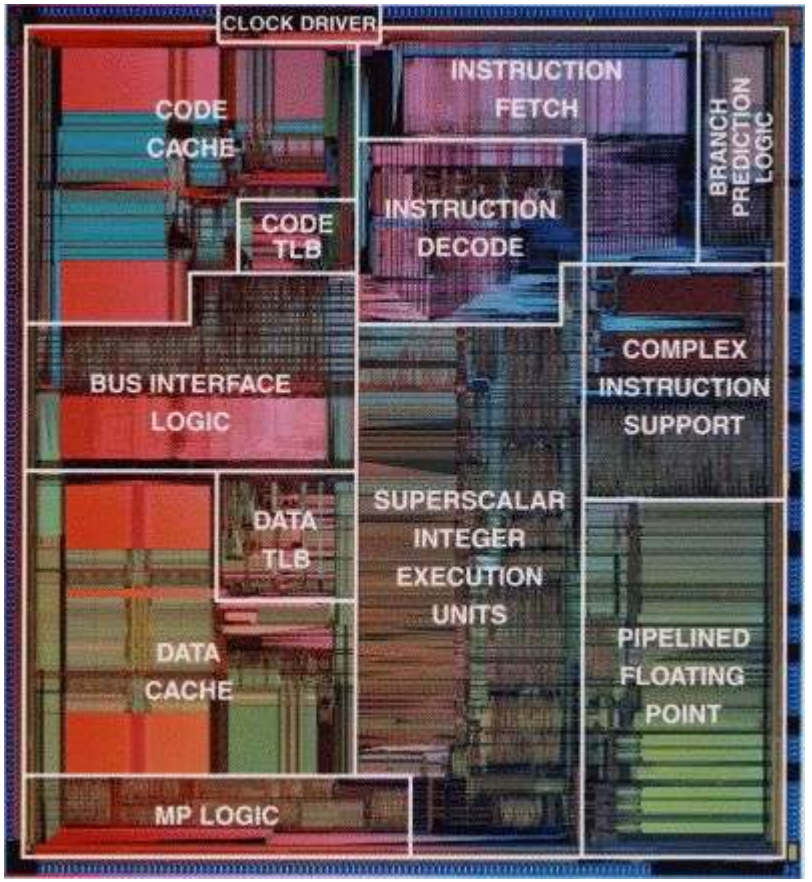
IA-32 For Loop Example

```
; for (i=0; i!=10; i++) { a[i]++; }

          MOV EBX,A  ; EBX = &A = address of A
          MOV EAX,0  ; EAX = i = 0
 FOR:     CMP EAX,10 ; if (i==10)
          JNE EXIT   ; then { goto FEND; }
          INC [EBX]  ; Mem[EBX](=a[i])++
          ADD EBX,4  ; EBX = &a[i+1]
          INC EAX    ; EAX++
          JMP FOR    ; goto FOR
 FEND:    ...
```

Intel Pentium Cache Hierarchy





Introduced March 1993, P60 & P66
Pentium (aka. 80586, P5)
100 MIPS peaks, 50% better than 486
273 pin package, 3.1 Million Transistors
BiCMOS 0.8 micron Process
Production ended in January 1998
http://www.pcguide.com/ref/cpu/fam/g5P54-c.html

Intel Pentium P5 “pipeline” Architecture

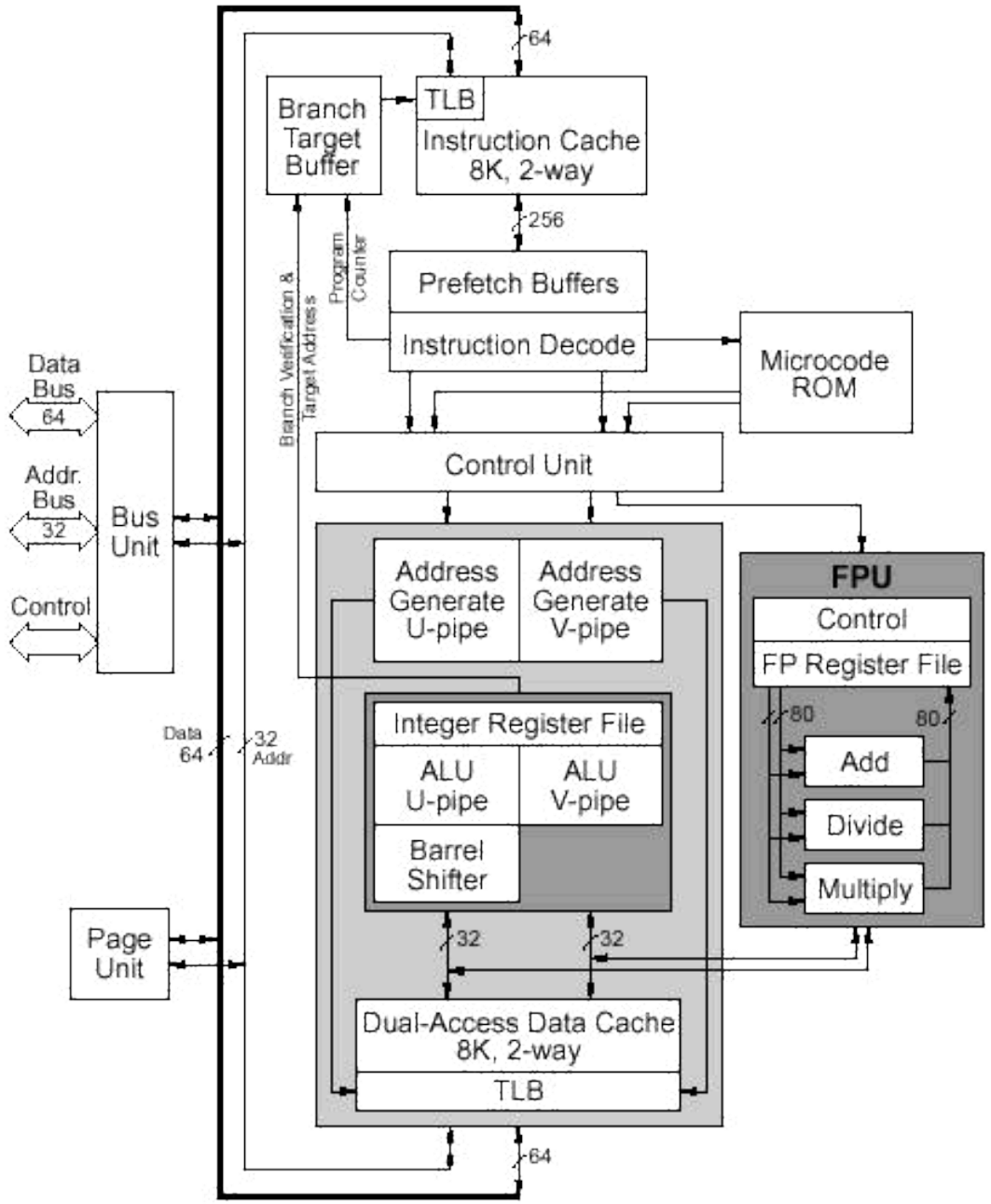
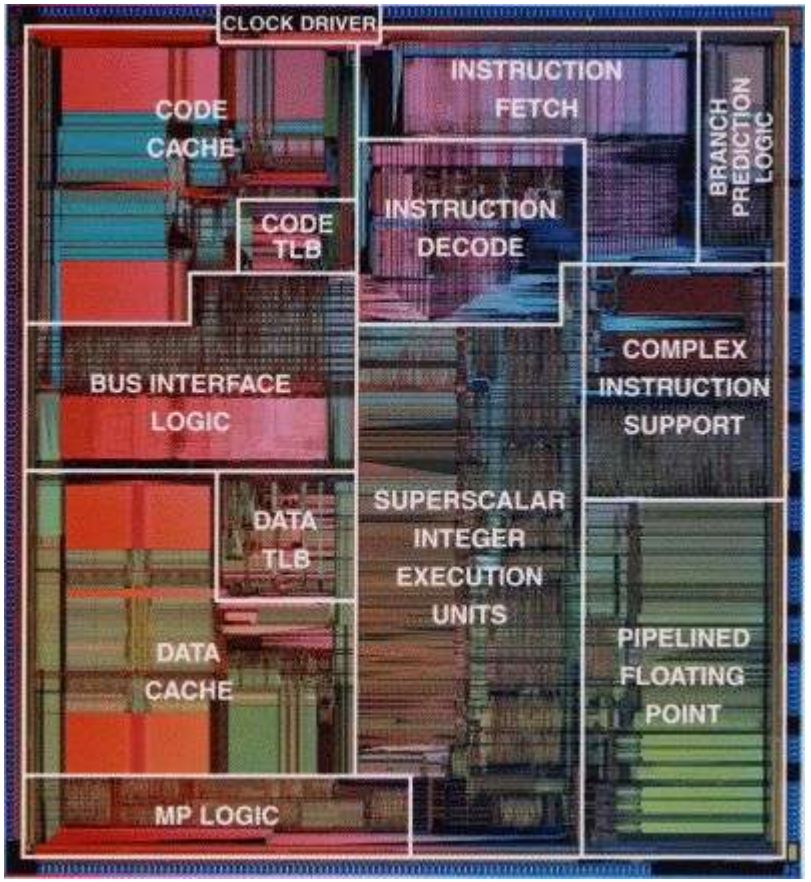
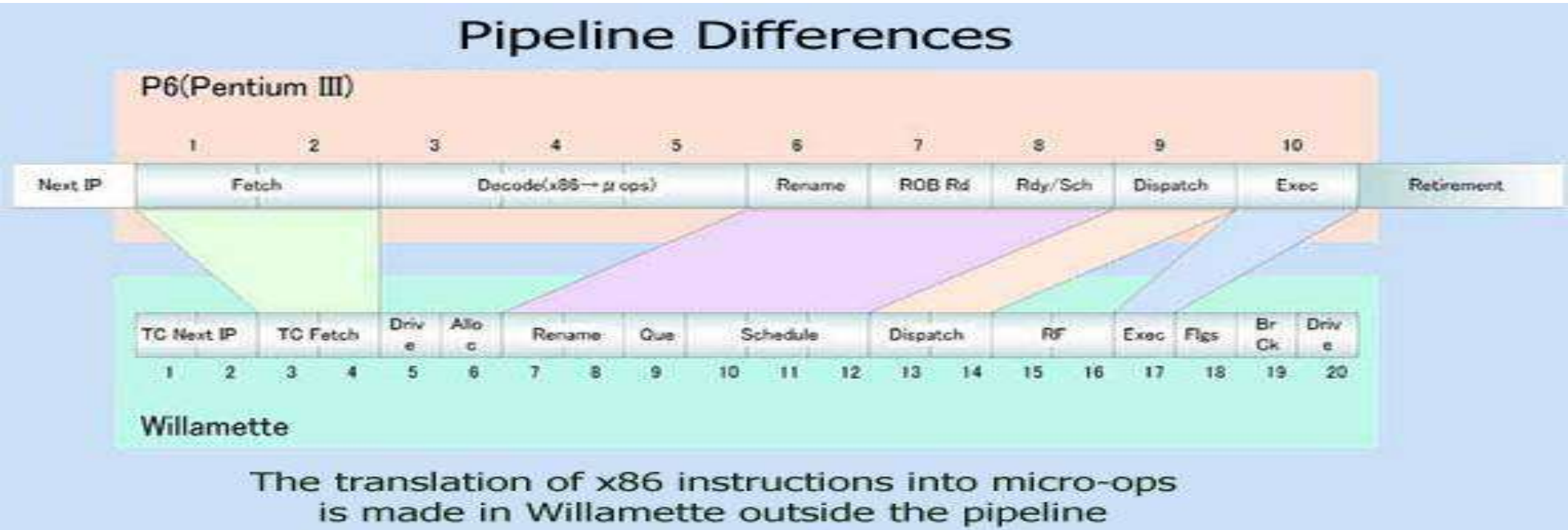


Figure 1. Pentium block diagram.



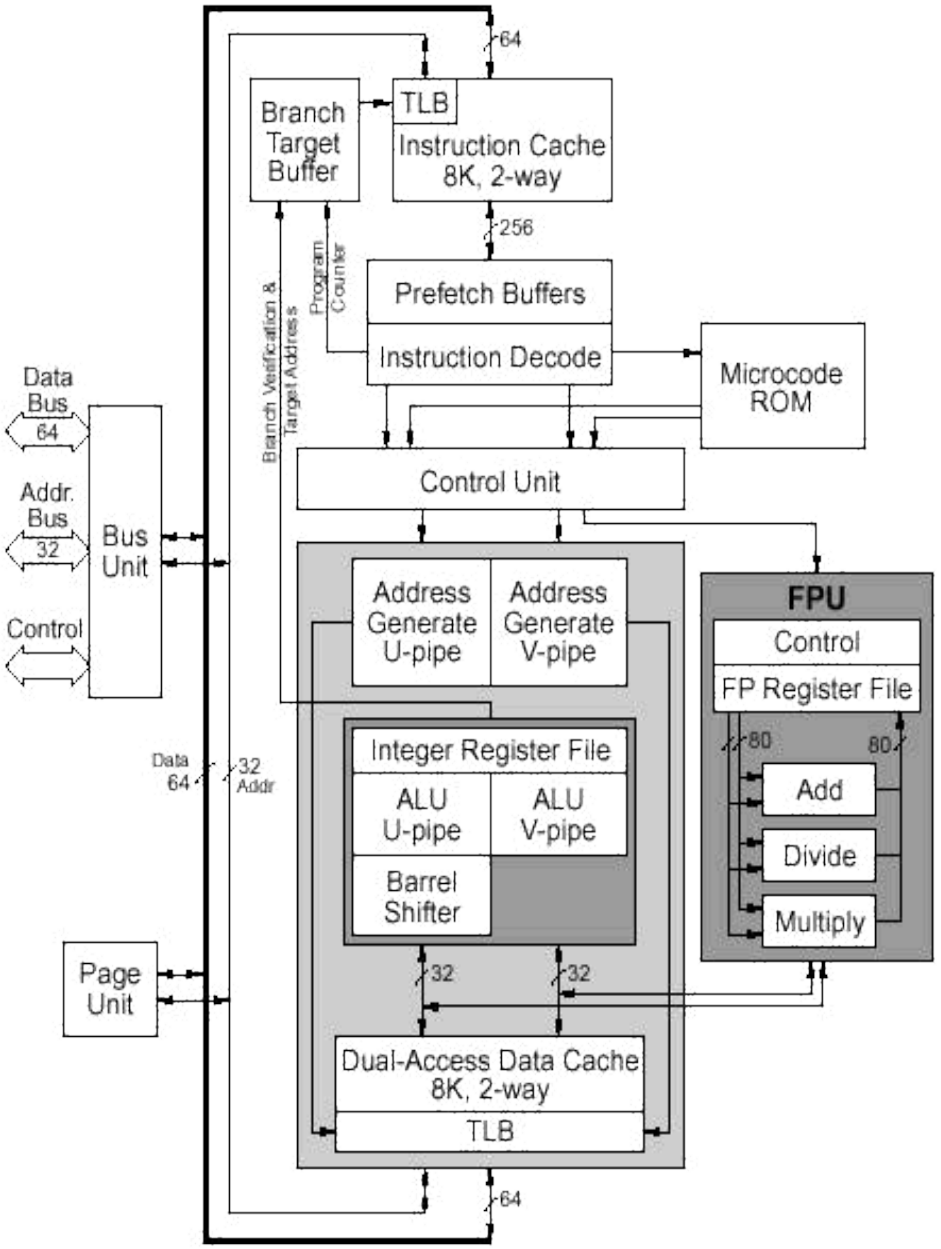
CPU Pipeline comparisons

CPU	# pipeline stages	Max. clock frequency
Pentium	5	300 MHz
Motorola G4	4	500 MHz
Motorola G4e	7	1 GHz
Pentium II and III	10	1.4 GHz
Athlon XP	10/15	2.5 Ghz
Athlon 64	12/17	>3.0 GHz
Pentium 4	20	>3.0 GHz
Pentium 4 “Prescott”	31	>5.0 GHz



http://www.intel.com/technology/itj/q12001/articles/art_2.htm

Intel Pentium Architecture Comparison



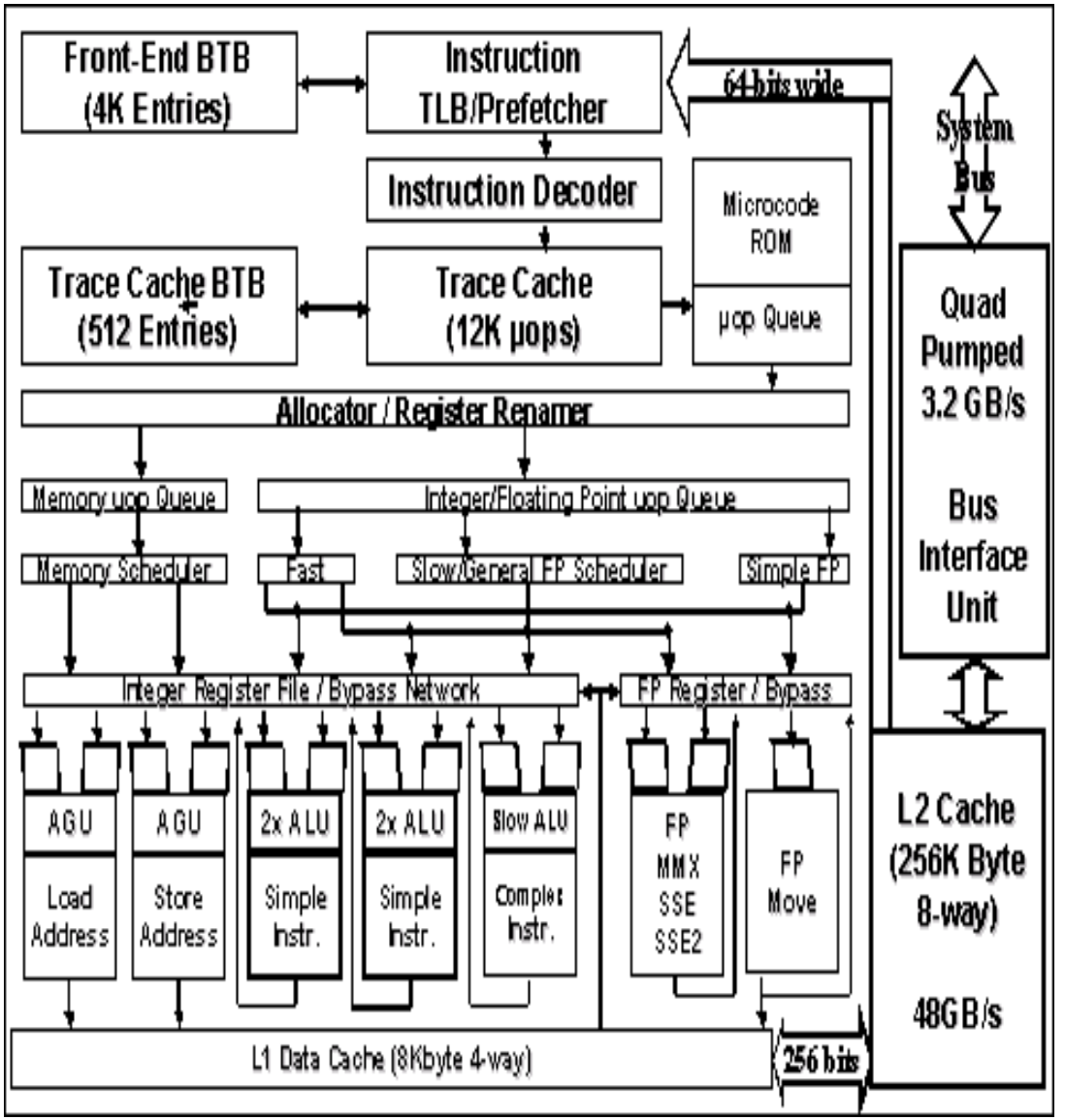
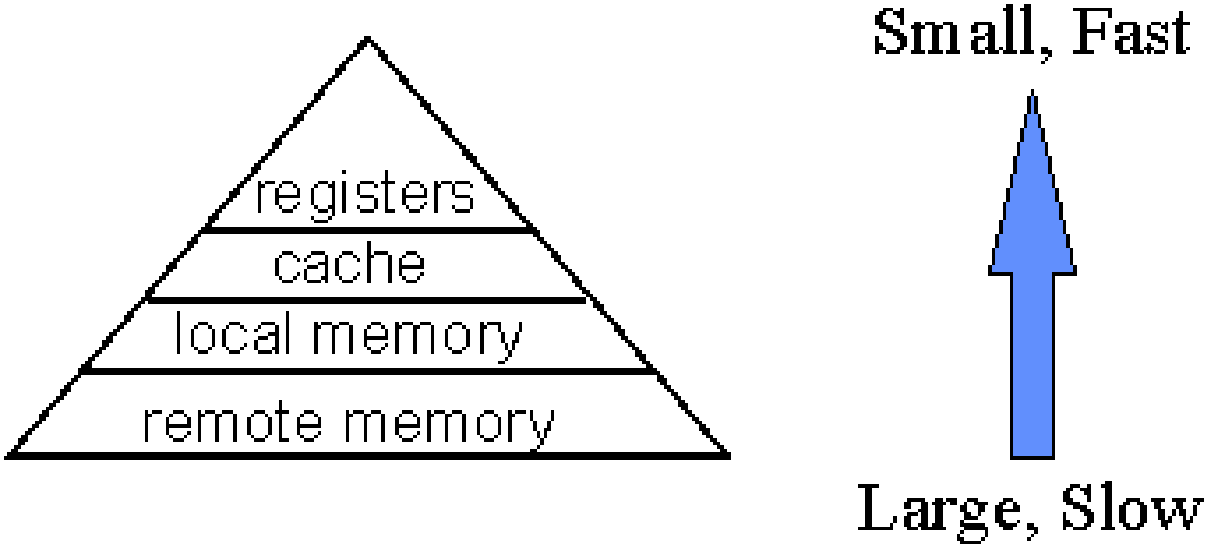


Figure 1. Pentium block diagram.

Memory Hierarchy



- **Can only do useful work at the top**
- 90-10 rule: 90% of time is spent of 10% of program
- Take advantage of locality
  - –temporal locality
    - keep recently accessed memory locations in cache
  - –spatial locality
    - keep memory locations nearby accessed memory locations in cache

Matrix Storage

4 by *4* matrix stored in
local memory
by row and by column

A_{11}	A_{12}	A_{13}	A_{14}
A_{21}	A_{22}	A_{23}	A_{24}
A_{31}	A_{32}	A_{33}	A_{34}
A_{41}	A_{42}	A_{43}	A_{44}

Memory locations

by row
(C)

A_{11} A_{12} A_{13} A_{14} A_{21} A_{22} A_{23} A_{24} A_{31} A_{32} A_{33} A_{34} A_{41} A_{42} A_{43} A_{44}

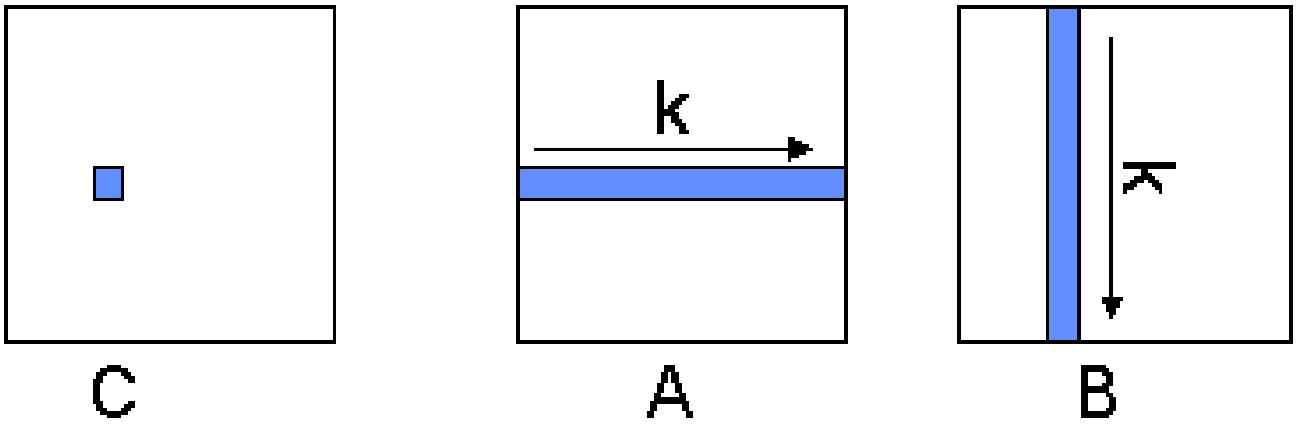
by column
(FORTRAN)

A_{11} A_{21} A_{31} A_{41} A_{12} A_{22} A_{32} A_{42} A_{13} A_{23} A_{33} A_{43} A_{14} A_{24} A_{34} A_{44}

Matrix multiplication: IJK

```
for i = 1, n
    for j = 1, n
        for k = 1, n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
        end
    end
end
```

Inner loop computes C(i, j) = dotproduct(A(i,.:), B(:, j))

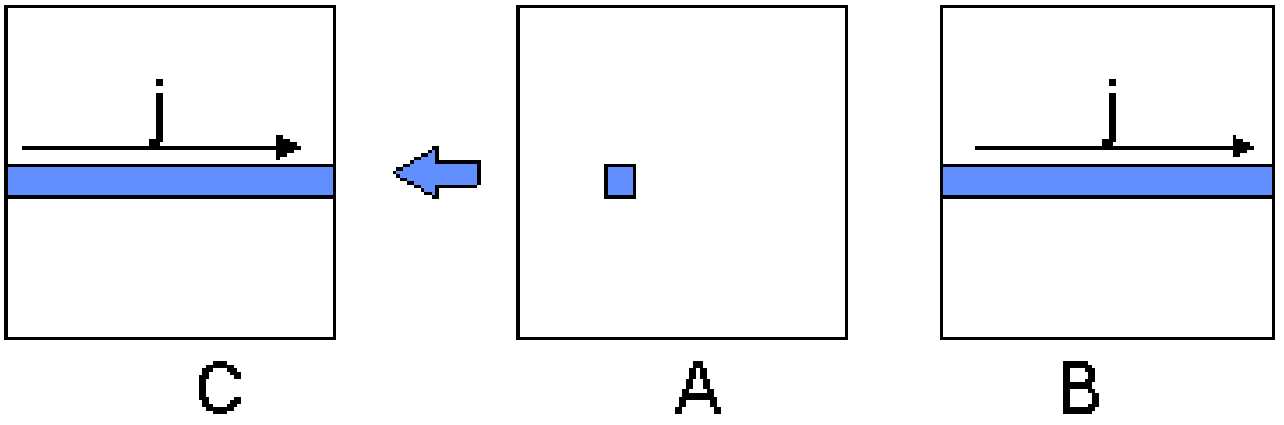


Matrix multiplication: IKJ
Improving Spatial Locality

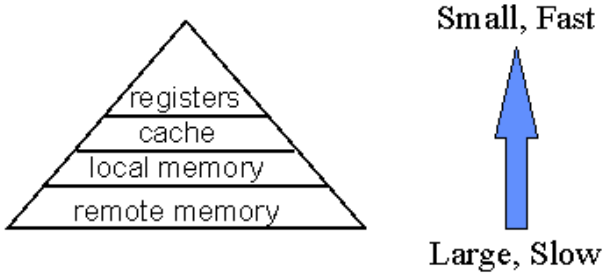
```
for i = 1, n
    for k = 1, n
        for j = 1, n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
        end
    end
end
```

Stride 1

All inner loop memory references are Stride 1; (refer to consecutive memory locations)



Memory Hierarchy

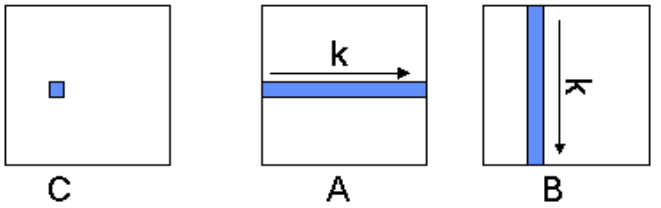


- Can only do useful work at the top
- 90-10 rule: 90% of time is spent of 10% of program
- Take advantage of locality
  - temporal locality
    - keep recently accessed memory locations in cache
  - spatial locality
    - keep memory locations nearby accessed memory locations in cache

Matrix multiplication: IJK

```
for i = 1, n
   for j = 1, n
       for k = 1, n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
       end
   end
end
```

Inner loop computes C(i, j) = dotproduct(A(i,:), B(:, j))



Matrix Storage

4 by *4* matrix stored in local memory by row and by column

A_{11}	A_{12}	A_{13}	A_{14}
A_{21}	A_{22}	A_{23}	A_{24}
A_{31}	A_{32}	A_{33}	A_{34}
A_{41}	A_{42}	A_{43}	A_{44}

Memory locations

by row (C)

$A_{11}\ A_{12}\ A_{13}\ A_{14}\ A_{21}\ A_{22}\ A_{23}\ A_{24}\ A_{31}\ A_{32}\ A_{33}\ A_{34}\ A_{41}\ A_{42}\ A_{43}\ A_{44}$

by column (FORTRAN)

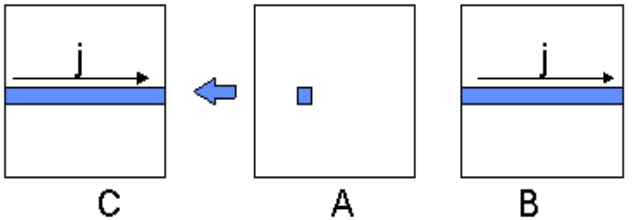
$A_{11}\ A_{21}\ A_{31}\ A_{41}\ A_{12}\ A_{22}\ A_{32}\ A_{42}\ A_{13}\ A_{23}\ A_{33}\ A_{43}\ A_{14}\ A_{24}\ A_{34}\ A_{44}$

Matrix multiplication: IKJ
Improving Spatial Locality

```
for i = 1, n
    for k = 1, n
       for j = 1, n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
       end
   end
end
```

Stride 1

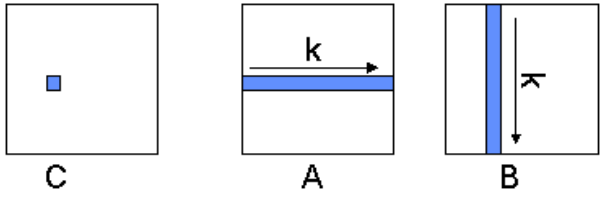
All inner loop memory references are Stride 1; (refer to consecutive memory locations)



Matrix multiplication: IJK

```
for i = 1, n
   for j = 1, n
       for k = 1, n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
       end
   end
end
```

Inner loop computes C(i, j) = dotproduct(A(i,:), B(:, j))



Improving Temoral Locality
Blocking for better cache behavior

- Transform multiplication into multiplications of submatrices

$$\begin{matrix} \mathbf{C11}\ C12\ C13 \\ C21\ C22\ C23 \\ C31\ C32\ C33 \end{matrix} = \begin{matrix} \mathbf{A11}\ \mathbf{A12}\ \mathbf{A13} \\ A21\ A22\ A23 \\ A31\ A32\ A33 \end{matrix} * \begin{matrix} \mathbf{B11}\ B12\ B13 \\ \mathbf{B21}\ B22\ B23 \\ \mathbf{B31}\ B32\ B33 \end{matrix}$$

$$C11 = A11*B11 + A12*B21 + A13*B31$$

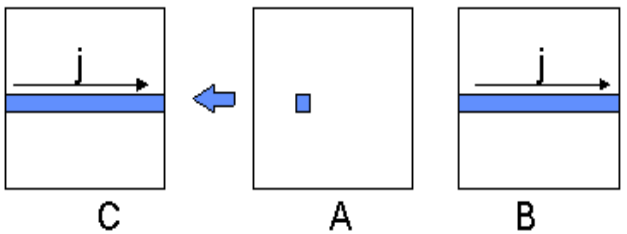
- Size of submatrix multiplication is chosen to fit into cache
  - m x m submatrices
  - $2m^3$ flops for $3m^2$ data in cache
- Each word fetched into cache is used m times

Matrix multiplication: IKJ
Improving Spatial Locality

```
for i = 1, n
    for k = 1, n
        for j = 1, n
             C(i,j) = C(i,j) + A(i,k) * B(k,j)
        end
    end
end
```

Stride 1

All inner loop memory references are Stride 1; (refer to consecutive memory locations)



Matrix multiplication: blocked

```
do ib = 1, n, bs
 do jb = 1, n, bs
  do kb = 1, n , bs
   do i = ib, min(ib+bs, n)
    do k = kb, min(kb+bs, n)
     do j = jb, min(jb+bs, n)
       C(i,j) = C(i,j) + A(i,k) * B(k,j)
     end
    end
   end
  end
 end
end
```

Want C to stay in cache as long as possible