FASTER THAN A SPEEDING SIMAPE!
MORE POWERFUL THAN A DUPLEX S/360-195!!
ABLE TO COMPILe FORTRAN IN A SINGLE..BOND!!

VECTOR

MUNCH
CRUNCH

SUMACRE-SCION

GROTESQUE...

ATTACK

PAROCHIAL

EXTEND-APP

HORRIFYING...

MACABRE...

QUESTIONABLE...

STAR-100

BRought TO you BY YoUR FRIENDLY, MAo CDC SCIENTIST.
Control Data Corp. STAR-100 (STring ARray) computer system

Advanced Features:
* distributed processing -- stations
* vector / matrix / string processing
* 3-address "macro" instructions
* virtual memory with bit addressing & LRU mapping
* high speed/high density logic (40 ns. clock)

Processing Speeds (for vectors):
* FP +, -, * 10 ns./number
* FP ÷, / 20 ns./number
* vector manip. ±25 ns./number
* boolean 2.5 ns./bit

Data Transfer Speeds:
* main memory: 12 billion bits/sec.
* I/O : 480 million bits/sec. (on 12 channels)
Typical Configuration

- Monitoring & Maintenance Station
- STAR-100 Central Processor
- Paging Station
- Disk Station
- Disk Station
- Tape Station
- Unit Record Station
- Service Station
- Service Station
- Grafix Station
- Terminals

Error Detection:
- parity (all over the place)
- heatsink temperatures
- freon pressure
- room dewpoint
- power fail
- power ripple
- water condensers (head pressure, oil pressure, water temp.)
things which can be monitored by the Monitoring & Maintenance Station:

- number of branches out of instruction buffer
- number of branches within instruction buffer
- number of instruction words referenced
- minor cycles waiting for instruction from MCS
- count of scalar instruction issues blocked due to operand conflict
- minor cycles waiting due to above
- count of recursive uses of FP pipes
- number of space table searches
- count of 1/4 words searched in space table
- count of program interrupts
- count of external interrupts
- count of page fault interrupts
- total count of interrupts
- count of DA channel MCS requests
- count of DA channel MCS requests accepted
- count of normal channel MCS requests
- count of normal channel MCS requests accepted
- count of CPU MCS requests
- count of CPU MCS requests accepted
- total count of MCS requests
- total count of MCS requests accepted
- count of vector instruction results less than 64 words
- count of vector instruction results between 65 and 8K words
- count of vector instruction results greater than 8K words
- count of string instructions with results less than 8 bytes
- count of string instructions with results between 8 and 32 bytes
- count of string instructions with results greater than 32 bytes
- count of FP alignments greater than n
- count of FP normalizations greater than n
- count of FP unnormalizations greater than n
- count of instructions with a certain class of opcodes
- time
- number of vector instructions
- minor cycles needed to buffer operands for alignment
Main Core Storage (MCS)

Access: 350 ms/word

MCS organized as:

- 9-7 x-8 bytes per word
- 8 (or 16) sections
- of 4 banks
- of 2K super words (swords)
- of 512 bits each

MCS address:

<table>
<thead>
<tr>
<th>11</th>
<th>3 or 4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>sword #</td>
<td>section #</td>
<td>bank #</td>
</tr>
</tbody>
</table>

\[ \frac{1}{2} \text{sword transfer order} \]
Figure 3-3. Stream Block Diagram
Figure 3-6. Floating Point Pipe 1

Data from Stream → ADD PIPELINE

MULTIPLY 1
MULTIPLY 2

MERGE 64

MERGE 1
MERGE 2

NORMALIZE COUNT

RECEIVE

EXPONENT COMPARE
COEFFICIENT ALIGNMENT
ADD

NORMALIZE COUNT
NORMALIZE SHIFT

TRANSMIT

DATA TO STREAM

FP32: EXP (C3) 0, 1
FP64: EXP (C3) 16, 48

Decimal +, -, x, all vectors except ÷, √
Figure 3-7. Floating Point Pipe 2
Figure 3-8. String Block Diagram
Figure 4-29. Vector Address Fields for Vector Instruction Example
**INITIAL VECTOR FIELD A**

<table>
<thead>
<tr>
<th>HALF-WORD ADDRESS</th>
<th>0</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>$A_0$</td>
<td></td>
</tr>
<tr>
<td>n+1</td>
<td>$A_1$ (NEAR ZERO)</td>
<td></td>
</tr>
<tr>
<td>n+2</td>
<td>$A_2$</td>
<td></td>
</tr>
<tr>
<td>n+3</td>
<td>$A_3$</td>
<td></td>
</tr>
<tr>
<td>n+4</td>
<td>$A_4$ (NEAR ZERO)</td>
<td></td>
</tr>
<tr>
<td>n+5</td>
<td>$A_5$</td>
<td></td>
</tr>
<tr>
<td>n+6</td>
<td>$A_6$ (NEAR ZERO)</td>
<td></td>
</tr>
<tr>
<td>n+7</td>
<td>$A_7$</td>
<td></td>
</tr>
<tr>
<td>n+8</td>
<td>$A_8$ (NEAR ZERO)</td>
<td></td>
</tr>
</tbody>
</table>

**ELEMENTS DISCARDED IN SPARSE VECTOR FIELD**

**GENERATED ORDER VECTOR Z**

$A_1$ $A_0$ $A_8$ $A_7$ $A_6$ $A_5$ $A_4$ $A_3$ $A_2$ $A_1$

**SPARSE VECTOR FIELD A**

<table>
<thead>
<tr>
<th>HALF-WORD ADDRESS</th>
<th>0</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>$A_0$</td>
<td></td>
</tr>
<tr>
<td>P+1</td>
<td>$A_2$</td>
<td></td>
</tr>
<tr>
<td>P+2</td>
<td>$A_3$</td>
<td></td>
</tr>
<tr>
<td>P+3</td>
<td>$A_5$</td>
<td></td>
</tr>
<tr>
<td>P+4</td>
<td>$A_7$</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** 32-BIT OPERANDS ARE ASSUMED.

Figure 4-38. Example of Compressing Initial Vector Field Into Sparse Vector Field
Figure 4-41. Example of an Add U; A + B → C Sparse Vector Instruction
Figure 4-46. Basic Arithmetic Sequence for Poly. Eval. A(N)
Per B → C(N) Instruction
Figure 4-89. Example of the Character String Merge Instruction
Notes on instruction descriptions on following pages:

R, S, T, U denote registers
A, B, C denote vectors

operand type designations:

FP = Floating-Point (length specified in instruction)
FP32 = 32-bit Floating-Point
FP64 = 64-bit Floating-Point
In = n-bit integer
length field of 64-bit register is bits 0-15

*M* following an instruction means it can only be executed in Monitor mode

vectors can be floating-point
  integer
  string
  bit string
  BCD string (packed decimal)

u after operator means use upper result
l lower
n normalized
s significant
idle *M*

load Breakpoint Register from R

enable CPU for fault testing

initiate I/O on channel n *M*  
MONITOR *M*

load Monitor Interval Timer from R (1 MHz decrementer) *M*  

store Associative Registers *M*  
OC

load Associative Registers *M*  
OD

generate external interrupt designator *M*  
OE

load virtual address key register, translate virtual address *M*  
OF

calculate packed BCD to binary *NON-TYPICAL*

convert binary to packed BCD

load substring of length 1 into T

store T into substring of length 1

eextract first n columns of bit matrix (bit compress)

concatenate two bit matrices (bit merge)

concatenate first n columns of bit matrix with last m columns of another (bit mask)

merge two streams of sorted strings

copy substring elsewhere in same string

scan substring backwards for unequal byte

fill substring with immediate byte

fill substring with byte from R

generate bit matrix with first n columns = B'0'

generate bit matrix with first n columns = B'1'

count leading equal bits in bit substring

count B'1's in bit substring

absolute branch if $R = S$ (FP64)

$\begin{align*}
R \neq S \\
R \geq S \\
R < S \\
R \neq S \quad \text{(FP32)} \\
R \neq S \\
R < S \\
\end{align*}$

scan substring for equal byte

scan substring for unequal byte

load length field of $R$ with immediate integer

add $S$ to length field of $R$ and put in length field of $T$
30 increment R and indexed branch if R ≠ 0
31 test bit, alter bit, and absolute branch if test true
32 test DBBR bit, alter, and funny branch if test true
33 —
34 decrement R and indexed branch if R ≠ 0
35 save instruction address and indexed branch (BAL)
36 —
37 copy length field of R to length field of T
38 load Real-Time Clock into R (1 MHz incrementer)
39 load Job Interval Timer from R (1 MHz decrementer) *M*
3A put DBBR in T, put R in DBBR
3B R * S to T (I32)
3C R * S to T (I48)
3D R * S to T (I48)
3E load R with immediate length (I48)
3F add immediate length to R (I48)

40 R +u S to T (FP32)
41 R +l S to T
42 R +n S to T
43 —
44 R -u S to T
45 R -l S to T
46 R -n S to T
47 —
48 R *u S to T
49 R *l S to T
4A —
4B R *s S to T
4C R /u S to T
4D load R with immediate length (I32)
4E add immediate length to R (I32)
4F R /s S to T (FP32)

50 truncate R to T (FP32)
51 floor R to T
52 ceiling R to T
53 root R to T
54 (un)normalize R per S to T
55 adjust exponent of R per S to T
56 —
57 —
58 R to T
59 absolute R to T
5A exponent of R to T as integer
5B exponent of R with coefficient of S to T
5C lengthen R to T (normalized)
5D lengthen R to T (unnormalized)
5E load A(1) into R
5F store R into A(1)
60 R +u S to T (FP64)
61 R +1 S to T 
62 R +n S to T 
63 R + S to T (I48)
64 R -u S to T (FP64)
65 R -1 S to T 
66 R -n S to T 
67 R - S to T (I48)
68 R *u S to T (FP64)
69 R *1 S to T 
6A — 
6B R *s S to T 
6C R /u S to T 
6D — 
6E R /S S to T 
6F — 

70 truncate R to T (FP64)
71 floor R to T 
72 ceiling R to T 
73 root R to T 
74 (un)normalize R per S to T 
75 adjust exponent of R per S to T 
76 shorten R to T 
77 rounded shorten R to T 
78 R to T 
79 absolute R to T 
7A exponent of R to T as integer 
7B exponent of R with coefficient of S to T 
7C length field of R to T 
7D — 
7E load A(i) into R 
7F store R into A(i) 

80 subvector A +u subvector B to controlled (sub)vector C (FP)
81 — +1 
82 — +n 
83 subvector A + u subvector B to controlled (sub)vector C (I48)
84 subvector A -u subvector B to controlled (sub)vector C (FP)
85 — -1 
86 — -n 
87 subvector A - subvector B to controlled (sub)vector C (I48)
88 subvector A *u subvector B to controlled (sub)vector C (FP)
89 — +1 
8A — *s 
8B — /u 
8C — 
8E — 
8F — /s
truncatu subvector A to controlled (sub)vector C (FP)
floor
ceiling
root
(normalize subvector A per subvector B to controlled (sub)vector C
adjust exp. of
shorten subvector A to controlled (sub)vector C
rounded shorten
subvector A to controlled (sub)vector C
absolute subvector A to controlled (sub)vector C
exponent of
exponent of subvector A w/ coeff. of subvector B to cont. (sub)vector C
lengthen subvector A to controlled (sub)vector C

A0    sparsevector A +u sparsevector B to sparsevector C (FP)
A1    +1
A2    +n
A3    -
A4    -u
A5    -1
A6    -n
A7    -
A8    *u
A9    *1
AA    -
AB    *s
AC    /u
AD    -
AE    -
AF    /s

B0    funny branch if R + S to T = U (I48)
B1
B2
B3
B4
B5
B6    unconditional branch to indexed immediate address
B7    subvector B(i) to vector C(subvector A(i)), i=1,n (FP)
B8    reverse subvector A to controlled (sub)vector C
B9    transpose 8x8 matrix A to registers
BA    vector B(subvector A(i)) to controlled (sub)vector C(i), i=1,n
BB    select vector A(i) or vector B(i) per control to vector C(i), i=1,n
BC    compress vector A per control into sparsevector C
BD    merge vectors A and B per control to vector C
BE    load R with immediate integer (I48)
BF    add immediate integer to R
compare controlled subvectors A and B, subscript of first = to R (FP)

compare subvectors A and B for =, generate control vector

search for vector $A(i) = B(j)$, $j$ to controlled vector C, $i=1,n$

load R with immediate integer (I32)

add immediate integer to R

subvector A(1) elements ≥ subvector B(1) elements form controlled vector C, $i=1,n$ (FP)

subvectors $A + B / 2$ to controlled (sub)vector C (mean)

subvector $A(i) + A(i+1) / 2$ to controlled (sub)vector C(1), $i=1,n$

subvectors $A - B / 2$ to controlled (sub)vector C (mean difference)

subvector $(A(i+1) - A(i))$ to controlled (sub)vector C(1), $i=1,n$ (dcit)

search masked bitstring A for bitstring B, allow partial mismatch

scan substring A using suitable B, bytes/indices to subvector C

maximum of controlled subvector A to R, subscript to S (FP)

minimum

controlled subvector A to R

dot product of controlled subvectors A and B to R

dot product of sparsevectors A and B to R

dot product of sparsevectors A and B to R

polynomial evaluate subvector A(i) with subvector B(##) constants

to controlled (sub)vector C(1), $i=1,n$

R + $i*S$ to controlled (sub)vector C(1), $i=1,n$

bit substring A + bit substring B to bit substring C

BCD substring A + BCD substring B to BCD substring C

compare bit substring A to bit substring B

compare BCD substring A to BCD substring B

merge substrings A and B per mask to substring C

edit & mark substring A per substring B to substring C

substring A bytes + substring B bytes modulo G to substring C bytes

translate substring A using suitable B to substring C

scan substring A using suitable B, byte/index to R (THT)
F0  substring A XOR substring B to substring C
F1  AND
F2  OR
F3  STROKE
F4  PIERCE
F5  IMPLICATION $A \rightarrow B$
F6  INHIBIT
F7  EQUIVALENCE
F8  copy substring A to substring C
F9  copy NOT substring A to substring C
FA  copy scaled BCD substring A to BCD substring C
FB  pack substring A to BCD substring C
FC  unpack BCD substring A to substring C
FD  compare substring A with substring B per mask substring C
FE  search masked substring A for substring B, allow partial mismatch
FF  search masked subvector A for subvector B, allow partial mismatch (Io4)