

## Description

The NEC μPD7281 Image Pipelined Processor is a high-speed digital signal processor specifically designed for digital image processing such as restoration, enhancement, compression, and pattern recognition. The μPD7281 employs token-based data-flow and pipelined architecture to achieve a very high throughput rate. A high-speed on-chip multiplier speeds calculations. More than one μPD7281 can easily be cascaded with a minimum amount of interface hardware to increase the throughput rate even further. The μPD7281 is designed to be used as a peripheral processor for minicomputers or microcomputers, thereby relieving the host processor from the burden of time-intensive computations. The μPD7281 has a very powerful instruction set designed specifically for digital image processing algorithms. The Image Pipelined Processor can also be used as either a general purpose digital signal processor or a numeric processor.

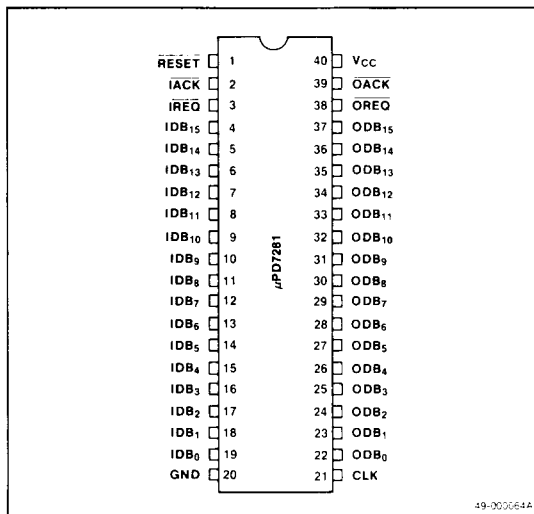
## Features

- Token-based data-flow architecture
- Internal pipelined ring architecture
- Powerful instruction set for image processing
- 17 x 17-bit (including sign bits) fast multiplier: 200 ns
- High-speed data I/O handling
  - Asynchronous two-wire handshaking protocols
  - Separate data input and output pins
- Easy multiple-processor configuration
- Rewritable program stores
- On-chip memories:
  - Link Table (LT): 128 x 16 bits
  - Function Table (FT): 64 x 40 bits
  - Data Memory (DM): 512 x 18 bits
  - Data Queue (DQ): 32 x 60 bits
  - Generator Queue (GQ): 16 x 60 bits
  - Output Queue (OQ): 8 x 32 bits
- NMOS technology
- Single +5 V power supply
- 40-pin DIP

## Applications

- Digital image restoration
- Digital image enhancement
- Pattern recognition
- Digital image data compression
- Radar and sonar processing
- Fast Fourier Transforms (FFT)
- Digital filtering
- Speech processing
- Numeric processing

## Pin Configuration



## Performance Benchmarks

Operation	1 μPD7281	3 μPD7281s	Note
Rotation	1.5 sec	0.6 sec	512 x 512 binary image
1/2 Shrinking	80 ms	30 ms	512 x 512 binary image
Smoothing	1.1 sec	0.4 sec	512 x 512 binary image
3x3 Convolution	3.0 sec	1.1 sec	512 x 512 grey scale image
64-stage FIR Filter	50 μs	18 μs	17-bit fixed point
cos(x)	40 μs	15 μs	33-bit fixed point

## Ordering Information

Part Number	Package Type
μPD7281D	40-pin ceramic DIP

**Pin Identification**

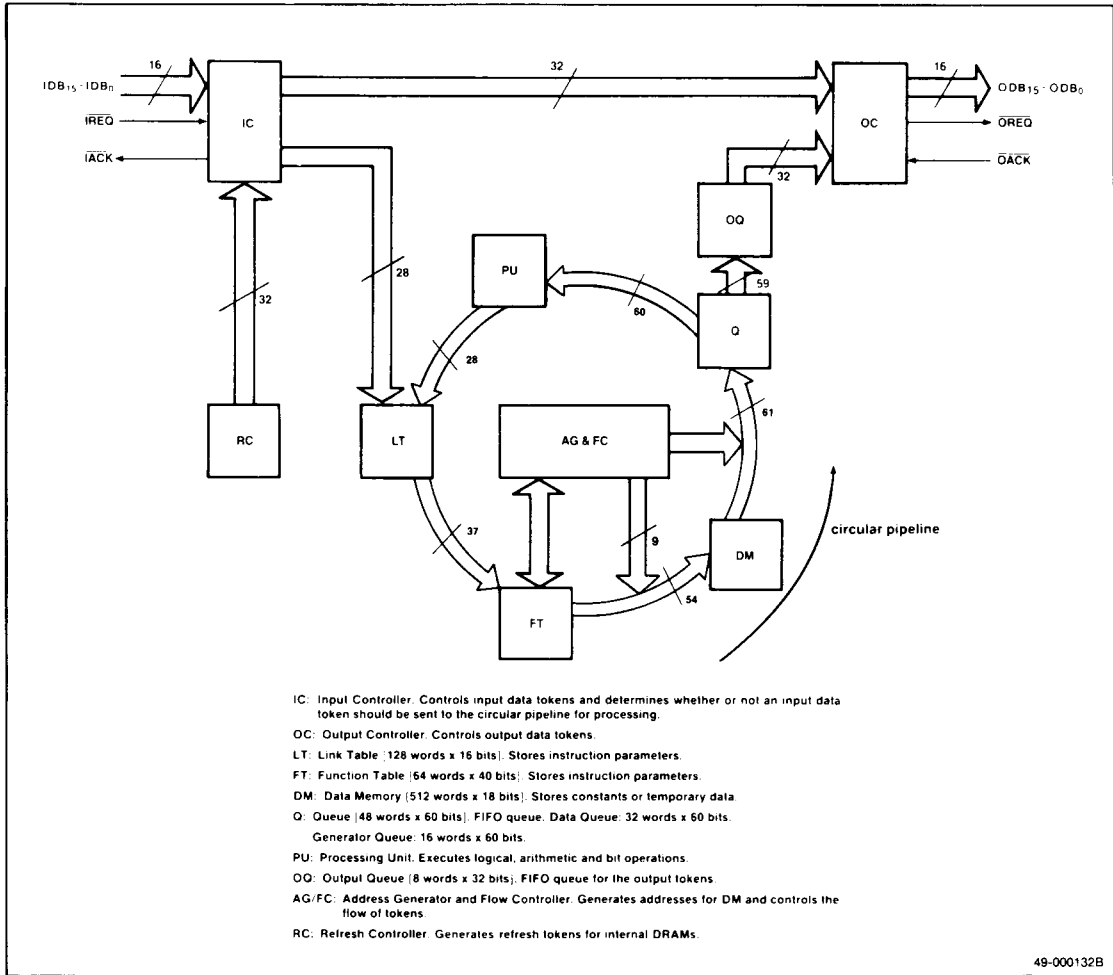
No.	Signal	I/O	At RESET	Description
1	RESET	In		System Reset: A low signal on this pin initializes μPD7281. During the reset, a 4-bit module number should be placed on IDB <sub>15</sub> - IDB <sub>12</sub> .
2	IACK	Out	High	Input Acknowledge: This acknowledge signal is output by the μPD7281 to notify the external data source that a 16-bit data transfer has been completed.
3	IREQ	In		Input Request: This input signal requests a data transfer from an external device to μPD7281.
4-19	IDB <sub>15</sub> - IDB <sub>0</sub>	In		16-bit input data bus: 32-bit input data tokens are input to the Input Controller as two 16-bit words.
20	GND			Power ground
21	CLK	In		System clock input (10 MHz: target spec)
22-37	ODB <sub>15</sub> - ODB <sub>0</sub>	Out	High Impedance	16-bit output data bus: 32-bit output data tokens are output by the Output Controller as two 16-bit words.
38	OREQ	Out	High	Output Request: This signal informs an external device that a 16-bit data word is ready to be transferred out of μPD7281.
39	OACK	In		Output Acknowledge: This acknowledge signal input by the external data destination notifies μPD7281 that a 16-bit data transfer may occur.
40	V <sub>CC</sub>			+5 V power supply

**Architecture**

The μPD7281 utilizes a token-based, data-flow architecture. This novel architecture not only provides multiprocessing capability without complex external hardware, but also offers high computational efficiency within each processor. Taking advantage of the multiprocessing capability of data-flow architecture, almost any processing speed requirements can be satisfied by using as many μPD7281s as needed in the system. Within each μPD7281, the data-flow architecture provides high computational efficiency through concurrent operations. For example, while the Processing Unit (or ALU) spends its time for actual computations only, the internal memory address calculations, internal memory read and write operations and input/output operations are all being done concurrently. Furthermore, in contrast to conventional von Neumann processors, a data-flow processor doesn't fetch instructions, perform subroutine stack operations or do data transfers between registers. Therefore, it does not spend the time required for these operations.

The μPD7281 also utilizes an internally pipelined architecture. As shown in the block diagram, a circular pipeline is formed by five functional blocks: the Link Table (LT), the Function Table (FT), the Data Memory (DM), the Queue (Q), and the Processing Unit (PU). A token entered through the Input Controller (IC) is passed on to the Link Table to be processed around the pipelined ring as many times as needed. When a token is finished being processed, it is queued into Output Queue (OQ) and then output via the Output Controller (OC).

## Block Diagram



3g

## Absolute Maximum Ratings

$T_A = +25^\circ\text{C}$

Supply voltage, $V_{DD}$	-0.5 V to +7.0 V
Input voltage, $V_I$	-0.5 V to +7.0 V
Output voltage, $V_O$	-0.5 V to +7.0 V
Operating temperature, $T_{OPT1}$ (2 m/s air flow)	0°C to +70°C
Operating temperature, $T_{OPT2}$ (No air flow)	0°C to +45°C
Storage temperature, $T_{STG}$	-65°C to +150°C

**Comment:** Exposing the device to stresses above those listed in Absolute Maximum Ratings could cause permanent damage. The device is not meant to be operated under conditions outside the limits described in the operational sections of this specification. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## Capacitance

$T_A = +25^\circ\text{C}$

Parameter	Symbol	Limits		Unit	Test Conditions
		Min	Max		
CLK capacitance	$C_K$		20	pF	$f_c = 1 \text{ MHz}$
Input capacitance	$C_I$	10		pF	(All other pins at 0 V)
Output capacitance	$C_O$		20	pF	

**DC Characteristics**

T<sub>A</sub> = 0°C to +70°C, V<sub>DD</sub> = 5 V ±10%

Parameter	Symbol	Limits			Unit	Test Conditions
		Min	Typ	Max		
Input low voltage 1 (RESET, IDB <sub>15-0</sub> )	V <sub>IL1</sub>	-0.5		0.7	V	
Input high voltage 1 (RESET, IDB <sub>15-0</sub> )	V <sub>IH1</sub>	2.0		V <sub>DD</sub> + 0.5	V	
Input low voltage 2 (IREQ, OACK, CLK)	V <sub>IL2</sub>	-0.5		0.45	V	
Input high voltage 2 (IREQ, OACK, CLK)	V <sub>IH2</sub>	3.5		V <sub>DD</sub> + 0.5	V	
Output low voltage	V <sub>OL</sub>			0.45	V	I <sub>OL</sub> = 2.0 mA
Output high voltage	V <sub>OH</sub>	2.4			V	I <sub>OH</sub> = -400 μA
Input leakage current	I <sub>LJ</sub>			±10	μA	0 V ≤ V <sub>I</sub> ≤ V <sub>DD</sub>
Output leakage current	I <sub>LO</sub>			±10	μA	0 V ≤ V <sub>O</sub> ≤ V <sub>DD</sub>
Supply current	I <sub>DD</sub>	320	500		mA	

**AC Characteristics**

T<sub>A</sub> = 0°C to +70°C, V<sub>DD</sub> = 5 V ±10%

Parameter	Symbol	Limits			Unit	Test Conditions
		Min	Typ	Max		
CLK cycle time	t <sub>CLK</sub>	100		500	ns	Measured at 2 V
CLK pulse width high	t <sub>WKH</sub>	40			ns	
CLK pulse width low	t <sub>WKL</sub>	40			ns	
CLK rise time	t <sub>KR</sub>			10	ns	
CLK fall time	t <sub>KF</sub>			10	ns	
IACK delay time 1 (from IREQ down) (Note 1)	t <sub>DIAL1</sub>	20		50	ns	
IACK delay time 1 (from IREQ up) (Note 2)	t <sub>DIAH1</sub>	20		55	ns	
IACK delay time 2 (from IREQ down)	t <sub>DIAL2</sub>	20		70	ns	
IACK delay time 2 (from IREQ up)	t <sub>DIAH2</sub>	20		70	ns	
Min time between transitions on IREQ and IACK	t <sub>HIQ</sub>	15			ns	
IREQ rise time	t <sub>IQR</sub>			10	ns	

**AC Characteristics (cont)**

T<sub>A</sub> = 0°C to +70°C, V<sub>DD</sub> = 5 V ±10%

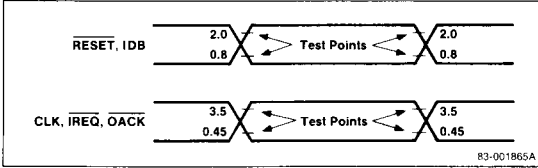
Parameter	Symbol	Limits			Unit	Test Conditions
		Min	Typ	Max		
IREQ fall time	t <sub>IQF</sub>			10	ns	
Data set up time (before IREQ up)	t <sub>SID</sub>	40			ns	
Data hold time (after IREQ up)	t <sub>HID</sub>	0			ns	
OREQ delay time 1 (from OACK down)	t <sub>DOOH</sub>	15		35	ns	
OREQ delay time 1 (from OACK up)	t <sub>DOQL</sub>	15		45	ns	
Min time between transitions on OREQ and OACK	t <sub>DOA</sub>	15			ns	
OACK rise time	t <sub>OAR</sub>			10	ns	
OACK fall time	t <sub>OAF</sub>			10	ns	
Data access time (after OREQ down)	t <sub>DOD</sub>			25	ns	
Data float time (after OREQ up)	t <sub>FOD</sub>	10		35	ns	
Pre RESET high time	t <sub>RVRS</sub> t <sub>CLK</sub>				ns	
RESET low time	t <sub>WRST</sub> 6t <sub>CLK</sub>				ns	
Module number data setup time (after RESET down)	t <sub>DM</sub>			2t <sub>CLK</sub>	ns	
Module number data hold time (after RESET up)	t <sub>HMD</sub>	0			ns	
Reset delay from CLK down	t <sub>DRST</sub>			(1/2)t <sub>CLK</sub>	ns	

**Notes:**

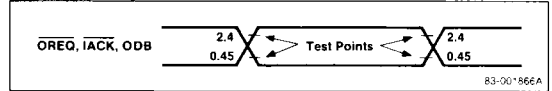
- (1) "Down" = on falling edge
- (2) "Up" = on rising edge
- (3) Output load capacitance: IACK, OREQ = 50 pF; ODB<sub>15-0</sub> = 100 pF

## Timing Waveforms

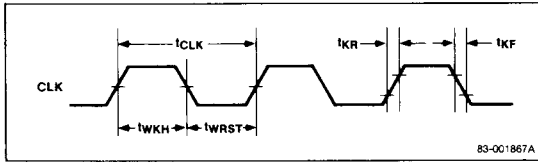
### AC Test Input Voltage



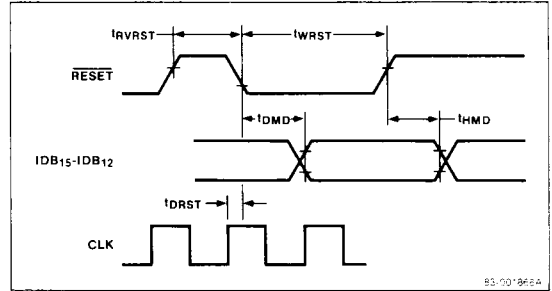
### AC Test Output Voltage



### Clock Timing

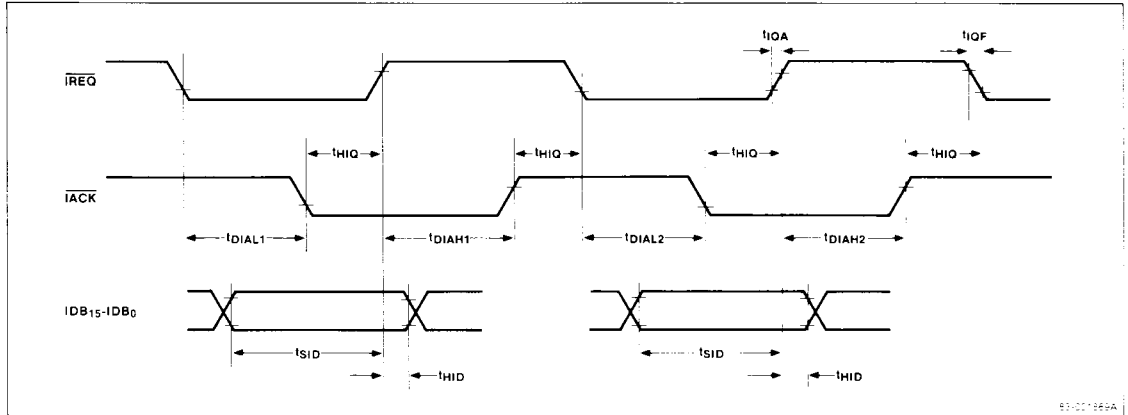


### Module Number and RESET Timing

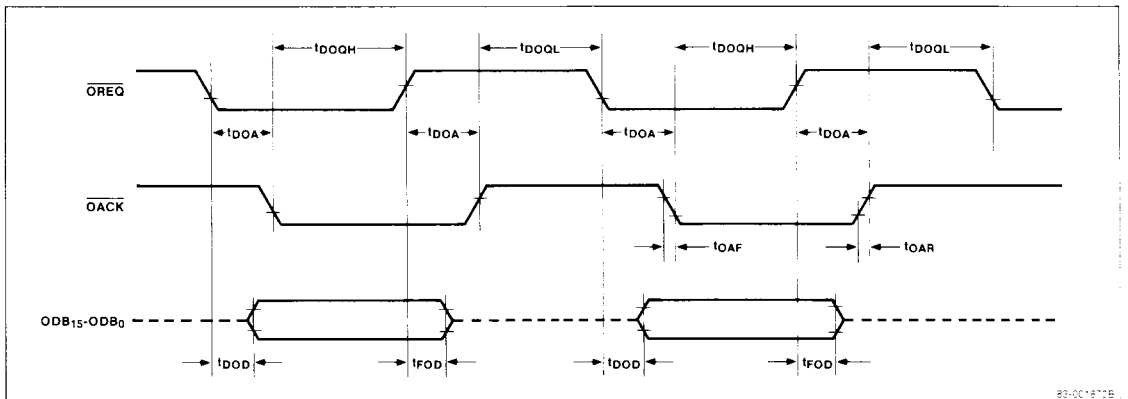


3g

### Input Handshake Timing



### Output Handshake Timing



### Functional Description

As shown in the block diagram, the μPD7281 consists of 10 functional blocks. Before any processing occurs, the host processor down-loads the object code into the Link Table and the Function Table of the μPD7281 by using specially formatted input tokens. At this time, constants may also be sent to the Data Memory to be stored. The contents of the Link Table and the Function Table are closely related to a computational graph. When a computational process is represented graphically, it usually forms a directed data-flow graph. In such a graph, the arcs (or edges, links, etc.) represent the entries in the Link Table and the nodes represent the entries in the Function Table. An arc between any two nodes has a data value, called a "token", and is identified by a corresponding entry in the Link Table. A node in the directed data-flow graph signifies an operation, and the type of operation is logged into the Function Table along with the identification information about the outgoing arc.

A minimal amount of interface hardware is required to configure μPD7281s in a multiprocessor system. As many as 14 μPD7281s can be cascaded together, as

shown in figure 1. Each μPD7281 must be assigned a Module Number (MN) during reset. Figure 2 shows the timing diagram for assigning the module number.

When any token enters a μPD7281, regardless of the total number of μPD7281s used in the system, the Input Controller of that μPD7281 discerns whether or not the entering token is to be processed by checking the Module Number (MN) field of the token. If the Module Number is not the same as the Module Number assigned during reset, the token is passed to the Output Controller so that it can be sent out via the Output Data Bus. However, if the token has the same Module Number, then the Input Controller strips off the MN field and sends the remaining part of the token to the Link Table for processing.

Once a token enters the circular pipeline by accessing the Link Table, it requires seven pipeline clock cycles for the token to fully circulate around the ring. One pipeline clock cycle is needed for the Link Table, the Function Table, or the Data Memory to process an incoming token, and two pipeline clock cycles are needed for the Queue or the Processing Unit to process a token. The Queue requires one pipeline

Figure 1. Connecting Multiple μPD7281s

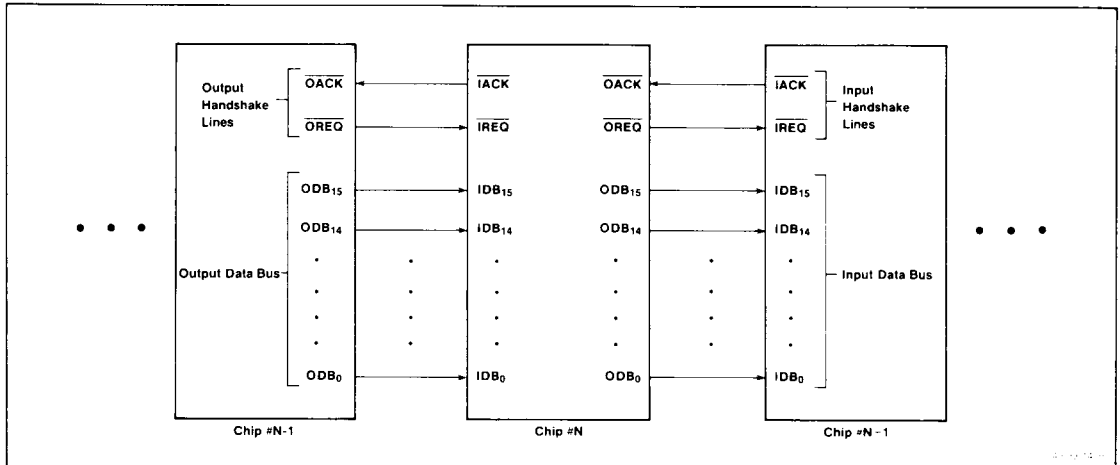
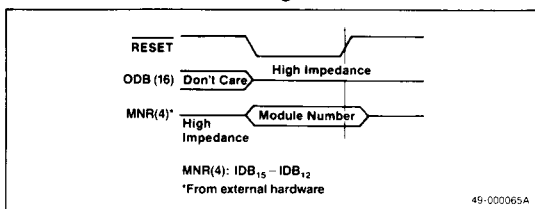


Figure 2. Timing Diagram for Assigning Module Numbers During RESET

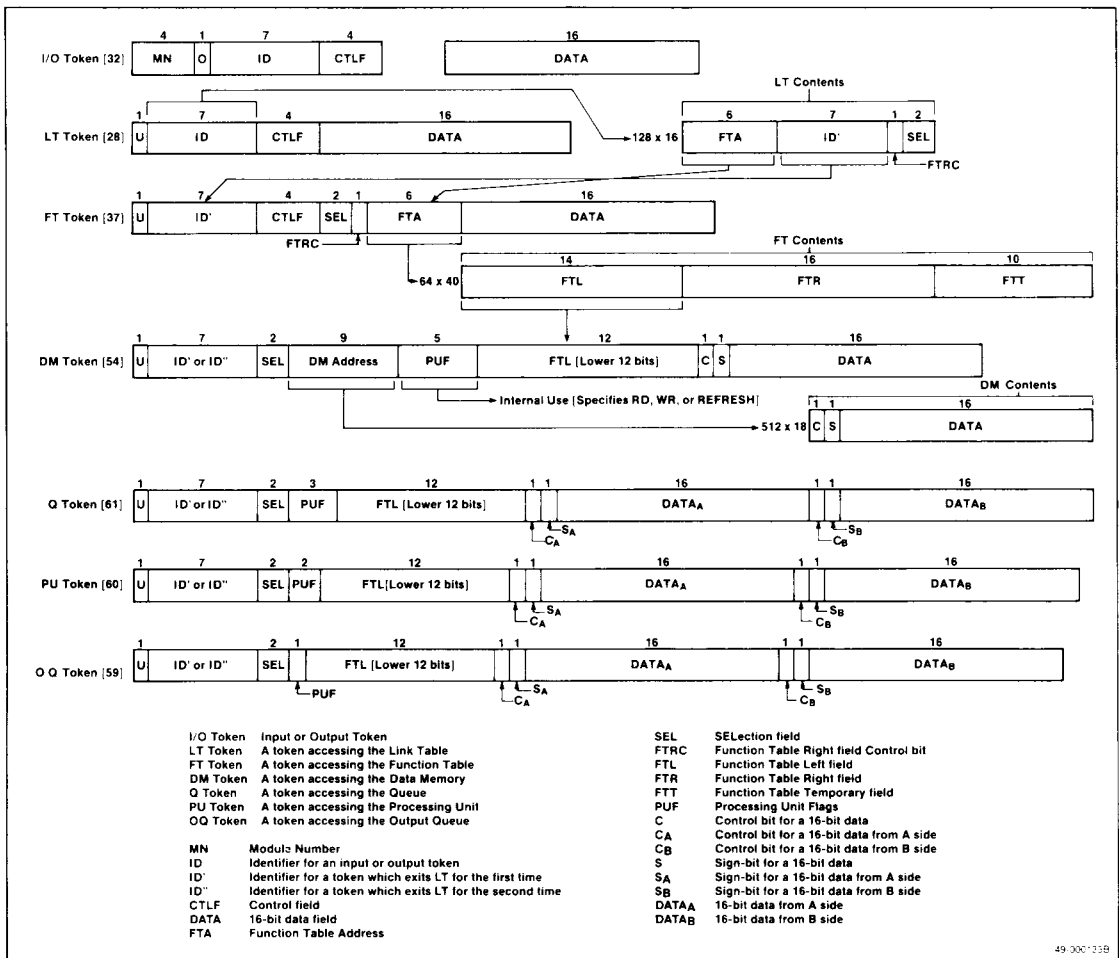


clock cycle to write and one cycle to read. Similarly, the Processing Unit requires one pipeline clock cycle to execute and one clock cycle to output the result. In other words, both the Processing Unit and the Queue are made of two-stage pipelines. Therefore, when seven tokens exist simultaneously in the circular pipeline, the pipeline is full and full parallel processing is achieved.

When a data token flows through each functional block in a given μPD7281, the format of the token changes significantly. The actual transitions of a token format through different functional blocks are shown

in figure 3. A data token flowing within the circular pipeline must have at least a 7-bit Identifier (ID) field and an 18-bit data field. The ID field is used as an address to access the Link Table memory. When a token accesses the LT memory, the ID field of the token is replaced by a new ID (shown as ID' in figure 3) previously stored in the LT memory. As a result, every time a data token accesses LT memory, its ID field is renewed. The data field of a token consists of a control bit, a sign bit and a 16-bit data. A token may have up to two data fields, as well as other fields (OP code, control, etc.) if necessary.

**Figure 3. Token Formats and Transitions**

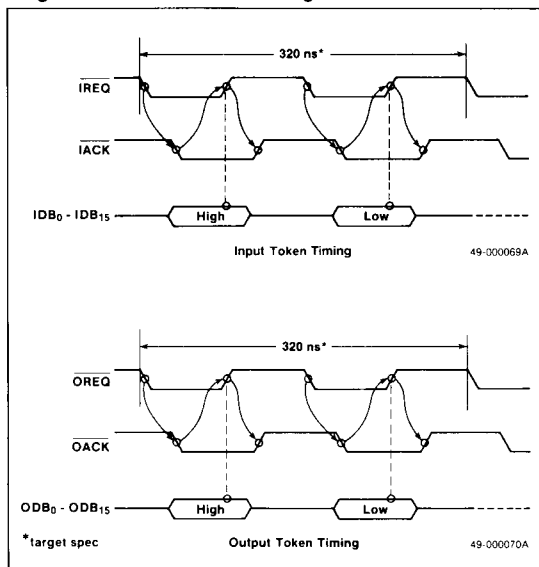


3g

### Input Controller [IC]

A 32-bit token is entered into a μPD7281 in two 16-bit halves using a two-signal request/acknowledge handshake method, as shown in figure 4. The input/output token format is shown in figure 7. After a token is accepted by the IC, the MN field of the token is compared to the Module Number of μPD7281 which was assigned at reset. If the Module Number of the accepted token is not the same, the IC passes the token directly to the Output Controller. If the MN field of the accepted token is the same, then the IC strips off the Module Number and sends the remaining part of the token to the Link Table. The IC also monitors the status of the Processing Unit. If it is busy, the IC delays accepting another token until it is no longer busy. The IC also accepts the refresh tokens from the Refresh Controller (RC) and sends them to the Link Table.

**Figure 4. Handshake Timing Waveforms**



### Output Controller [OC]

The OC outputs 32-bit tokens in two 16-bit halves using a two-signal request/acknowledge handshake method, as shown in figure 4. The types of tokens output by the OC are as follows: output data tokens from the Output Queue, error status data tokens generated internally by OC, DUMP tokens, and passing data tokens from the Input Controller.

### Link Table [LT]

The LT is a 128 x 16-bit dynamic RAM. The ID field of an incoming LT token is used to access the LT memory. The contents of an LT memory location

consist of a 6-bit Function Table Address (FTA), a 7-bit ID, a 1-bit Function Table Right Field Control (FTRC), and a 2-bit Selection (SEL) field. When a token accesses LT memory, its ID field is replaced by the new ID field contained in the memory location being accessed. Therefore, every time a token accesses LT memory, it is given a new ID. The FTA field is used to access FT memory locations. The FTRC bit and the SEL field are used to specify the type of instruction. By using specially formatted tokens, the contents of the LT can either be set during a program download or be read during a diagnosis.

### Function Table [FT]

The FT is a 64 x 40-bit dynamic RAM. As for the case of the Link Table, the contents can either be set during a program download or be read during a diagnosis by using specially formatted tokens.

Each FT memory location consists of a 14-bit Function Table Left field (FTL), a 16-bit Function Table Right field (FTR), and a 10-bit Function Table Temporary field (FTT). These fields contain control information for different types of instructions.

### Address Generator and Flow Controller [AG/FC]

The AG/FC generates the addresses to access the Data Memory (DM) and controls the writing of data to and the reading of data from the Data Memory. AG/FC determines whether the incoming token contains a one-operand instruction or a two-operand instruction. One-operand instruction tokens can be sent directly to the Queue. However, if the token contains a two-operand instruction, then both operands must be available before they can be sent to the Queue. For a two-operand instruction, the token which arrives at the Data Memory first is temporarily stored until the second operand token arrives. When the second operand token exits the Function Table, the AG/FC generates the Data Memory address which contains the first operand. Then, the second operand token and the first operand data read out from the Data Memory are sent to the Queue together.

### Data Memory [DM]

The DM is a 512 x 18-bit dynamic RAM which is used to queue the first operand for a two-operand instruction until the second operand arrives. DM can also be used as a temporary memory or as a buffer memory for I/O data.

### Queue [Q]

The Q is a FIFO memory configured with a 48 x 60-bit dynamic RAM. The Q is used to temporarily store the Processing Unit-bound and the Output Queue-bound tokens. The Q is further divided into two different FIFO memories: a 32 x 60-bit Data Queue (DQ) and a 16 x 60-bit Generator Queue (GQ). The DQ is used for the



PU, OUT and AG/FC instructions. The DQ temporarily stores the PU and AG/FC tokens before they are sent to the Processing Unit for processing. The DQ also temporarily stores the Output Queue tokens before they are sent to the Output Queue. The GQ is used for Generate (GE) instructions only. The DQ will not output tokens to the Output Queue if it is full, and the DQ or GQ will not output tokens to the Processing Unit if the Processing Unit is busy.

In order to control the number of tokens in the circular pipeline to prevent Q overflow, the Q is further restricted by the following two situation rules: when the DQ has eight or more tokens stored, the read from the GQ is inhibited, and when the DQ has fewer than eight tokens stored, the read from the GQ has a higher priority than the read from the DQ. Since instructions stored in the GQ generate tokens, restricting the number of GQ tokens is important in order to keep the Q from overflowing. In case the internal processing speed is slower than the rate of incoming data tokens, the DQ possesses a potential overflow condition. To prevent overflow, the processor is put into restrict/inhibit mode when the DQ reaches a level greater than 23.

### Output Queue [OQ]

The OQ is a first-in first-out (FIFO) memory configured in an 8 x 32-bit static RAM. The OQ is used to temporarily store the output data tokens from the Data Queue so that they can be output by the Output Controller via the output data bus. When OQ is full, it sends a signal to the Data Queue to delay accepting further tokens.

### Processing Unit [PU]

The PU executes two types of instructions: PU and GE. PU instructions include logical, arithmetic (add, subtract and multiply), barrel-shift, compare, data-exchange, bit-manipulation, bit-checking, data-conversion, double-precision adjust, and other operations. The control information for a PU instruction is contained in the Function Table Left field of the PU token. The GE instructions are used to generate a new token, multiple copies of a token, or block copies of a token. They can also be used to set the Control field (CTLF) of a token and to generate external memory addresses. If the current PU operation cannot be completed within a pipeline clock cycle, the PU sends a signal to the

Queue and the Input Controller to prevent them from releasing any more tokens.

### Refresh Controller [RC]

The RC automatically generates refresh tokens for the dynamic RAMs used in the circular pipeline, i.e. the LT, FT, DM, and Q. Each RC token, generated periodically, is sent to the Input Controller and is propagated through the LT, FT, DM and Q, in that order. The RC tokens are deleted after reaching the Q.

### Operation Modes

There are three different modes in which the μPD7281 can operate: Normal, Test, and Break (see figure 5). After an external hardware reset, the μPD7281 is in the Normal mode of operation. The μPD7281 can enter the Test mode for program debugging by inputting a SETBRK token (see figure 6) while the processor is in the Normal mode. If an overflow occurs in the Data Queue or the Generator Queue, the processor enters into the Break mode so that the internal contents of the processor can be examined; see table 1. Table 2 describes the effects of software and hardware resets.

3g

**Table 1. DUMPD Output Token Format**

MN	Z	ID	CTLF	DATA (16-bit field)
0000	0	0000 000	0111	xxxxx(5) GQ Size(5 bits) DQ Size(6 bits)
0000	0	0000 001	0111	xxxx(4) u(1) ID(7) CTLF(4)
0000	0	0000 010	0111	DATA(16)
0000	0	0000 011	0111	xxx (3) u(1) ID(7) x(1) C <sub>B</sub> , S <sub>B</sub> , C <sub>A</sub> , S <sub>A</sub>
0000	0	0000 100	0111	xx(2) FTL (Lower 12 bits) xx(2)
0000	0	0000 101	0111	DATA <sub>A</sub> (16)
0000	0	0000 110	0111	DATA <sub>B</sub> (16)
0000	0	0000 111	0111	xxxxxxxx(9) ID(7)

x: Don't care u: Unused

**Table 2. Effects of Reset Operation**

	Hardware Reset	Software Reset
MN	μPD7281 reads in MN	No Change
High/Low Word Flip-flop	Reset	No Change
Input Inhibit Control	Reset (No constraint)	No Change
LT Break State	Reset	Reset
Internal Operation	Stopped	Stopped
DQ, GQ, and OQ Pointers	Set to 0	Set to 0

Figure 5. μPD7281 Operation Modes

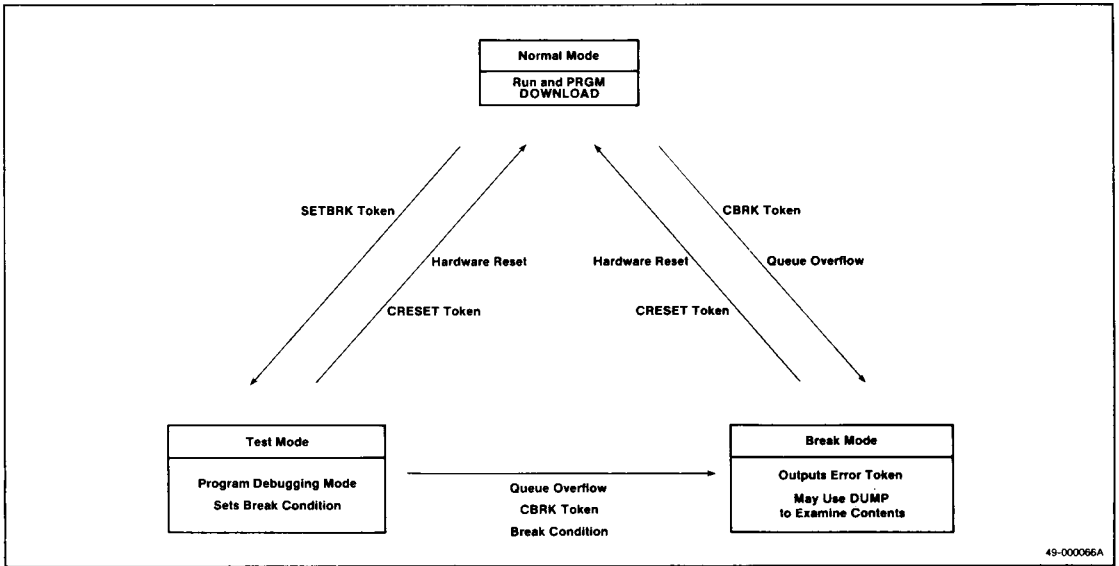
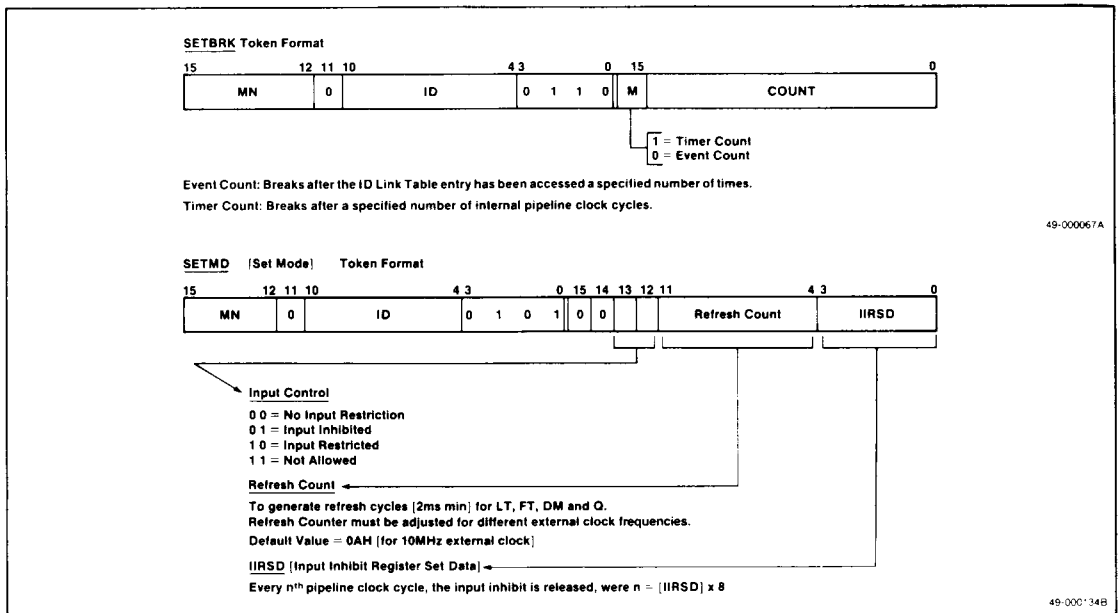


Figure 6. SETBRK (Set Break Condition) and SETMD (Set Mode) Token Formats

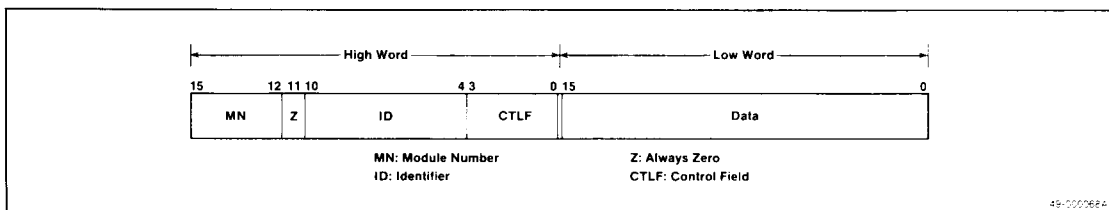


## Input/Output Tokens

The only way any external device can communicate with the μPD7281 is by using the I/O tokens (see figure 7). Both the input and the output tokens have the same format so that a token may flow through a series of multiple processors without a format change. A 32-bit I/O token is divided into upper and lower 16-bit words and input to or output from the μPD7281 a 16-bit word at a time. Object code is down-loaded into the Link

Table and the Function Table using SETLT, SETFTR, SETFTL and SETFTT input tokens. The contents of the Function Table and the Link Table can also be read using RDLT, RDFTR, RDFTL and RDFTT tokens. In order to write or read a value to and from the Data Memory, a program must be down-loaded and executed. Once object code is down-loaded into the μPD7281, data tokens are input to the processor, thereby initiating the processing. For a description of the input and output tokens, see tables 3 and 4.

**Figure 7. Input/Output Token Format**



3g

**Table 3. Input Token Format**

Input Token	High Word (16)				Low Word (16)				Remarks
	MN (4)	Z (1)	ID (7)	CTLF (4)	DATA (16)				
	15 12	11	10 4	3 0	15	0			
SETLT	MN	0	LT address	1 1 0 0	Data to be set in LT				Set LT
SETFTR	MN	0	FT address	1 1 0 1	Data to be set in FTR				Set FT Right Field
SETFTL	MN	0	FT address	1 1 1 0	Data to be set in FTL				Set FT Left Field
SETFTT	MN	0	FT address	1 1 1 1	Data to be set in FTT				Set FT Temporary Field
RDLT	MN	0	LT address	1 0 0 0					Read LT
RDFTR	MN	0	FT address	1 0 0 1					Read FT Right Field
RDFTL	MN	0	FT address	1 0 1 0					Read FT Left Field
RDFTT	MN	0	FT address	1 0 1 1					Read FT Temporary Field
CRESET	MN	0		0 1 0 0					Command Reset
SETMD	MN	0		0 1 0 1	Mode set data				Set Operation Mode
SETBRK	MN	0	ID	0 1 1 0	M (1)	Count (15)		Set Break Condition	
DUMP	MN	0	xxxx(4) DUMP (3)	0 1 1 1					Dump
CBRK	0 0 0 0	0		0 1 0 0					Command Break
VAN	1 1 1 1	0							Vanish Data
PASS	MN*	0							Pass Data
EXEC	MN	0	ID	0 0 C S	Data				Normal Execution Data

\* When MN is not the current module number  
 x: Don't care

**Table 4. Output Token Format**

Output Token	Upper-Order Word (16)							Lower-Order Word (16)			Remarks							
	MN (4)		Z (1)		ID (7)			CTLF (4)		DATA (16)								
	15	12	11	10	4	3	0	15	0									
LTRDD	0	0	0	0	0	LT address			1	0	0	0	Data read from LT	FT Read Data				
FTRRDD	0	0	0	0	0	FT address			1	0	0	1	Data read from FTR	FT Right Field Read Data				
FTLRDD	0	0	0	0	0	FT address			1	0	1	0	Data read from FTL	FT Left Field Read Data				
FTRRDD	0	0	0	0	0	FT address			1	0	1	1	Data read from FTT	FT Temporary Field Read Data				
PASSD	MN		0	ID				CTLFD	Data			Pass Data						
ERR	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	MN(4)MODE(4) 0 0 0 STATUS(5)	Error Data
DUMPD	0	0	0	0	0	0	0	0	DUMP(3)	0	1	1	1	Dump data	Dumped Data			
OUTD	MN		0	ID				0	0	C	S	Data	Output Data					

**Instruction Set Summary**

Tables 5 through 8 summarize the instruction set.

**Table 5. AG/FC Instructions**

Mnemonic	Instruction
QUEUE	Queue
RDCYCS	Read cyclic short
RDCYCL	Read cyclic long
WRCYCS	Write cyclic short
WRCYCL	Write cyclic long
RDWR	Read/Write Data Memory
RDIDX	Read Data Memory with index
PICKUP	Pickup data stream
COUNT	Count data stream
CONVO	Convolve
CNTGE	Count generation
DIVCYC	Divide cyclic
DIV	Divide
DIST	Distribute
SAVE	Save ID
CUT	Cut data stream

**Table 6. PU Instructions**

Mnemonic	Instruction
OR	Logical OR
AND	Logical AND
XOR	Logical EXCLUSIVE-OR
ANDNOT	Logical INVERT an operand then AND: (A•B)
NOT	Invert
ADD	Add
SUB	Subtract

**Table 6. PU Instructions (cont)**

Mnemonic	Instruction
MUL	Multiply
NOP	No operation
ADDSC	Add and shift count
SUBSC	Subtract and shift count
MULSC	Multiply and shift count
NOPSC	NOP and shift count
INC	Increment
DEC	Decrement
SHR	Shift right
SHL	Shift left
SHRBRV	Shift right with bit reverse
SHLBRV	Shift left with bit reverse
CMPNOM	Compare and normalize
CMP	Compare
CMPXCH	Compare and exchange
GET1	Get one bit
SET1	Set one bit
CLR1	Clear one bit
ANDMSK	Mask a word with logical AND
ORMSK	Mask a word with logical OR
CVT2AB	Convert 2's complement to sign-magnitude
CVTAB2	Convert sign-magnitude to 2's complement
ADJL	Adjust long (for double precision numbers)
ACC	Accumulate
COPYC	Copy control bit

**Table 7. GE Instructions**

Mnemonic	Instruction
COPYBK	Copy block
COPYM	Copy multiple
SETCTL	Set control field

**Table 8. OUT Instructions**

Mnemonic	Instruction
OUT1	Output 1 token
OUT2	Output 2 tokens

There are four different types of instructions which can be specified by the SEL field of an FT token. See table 9.

**Table 9. SEL Field of an FT Token**

SEL Type	Description
11 AG/FC	Executes instructions specified by the Function Table Right field while monitoring the Function Table Temporary field.
01 PU	Performs arithmetic, logical, barrel-shift, bit-manipulation, data-conversion, etc.
10 GE	Generates a block or multiple new tokens from a token. Sets the control field of a token. Increments or decrements the data field of a token.
00 OUT	Outputs data tokens from the circular pipeline to the Output Queue after the tokens are finished being processed.

## AG/FC Instructions

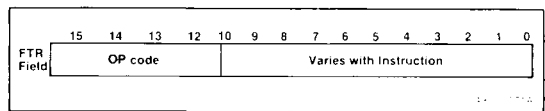
There are 16 AG/FC instructions (see table 10). They can be grouped into three types: Address Generator (AG), Flow Controller (FC), and AG/FC type.

AG type: RDCYCS, RDCYCL, WRCYCS, WRCYCL, RDWR, RDIDX

FC type: PICKUP, COUNT, CUT, DIVCYC, DIV, DIST, CONVO, SAVE, CNTGE

AG/FC type: QUEUE

A 4-bit OP code in the Function Table right field specifies the instruction to be executed.

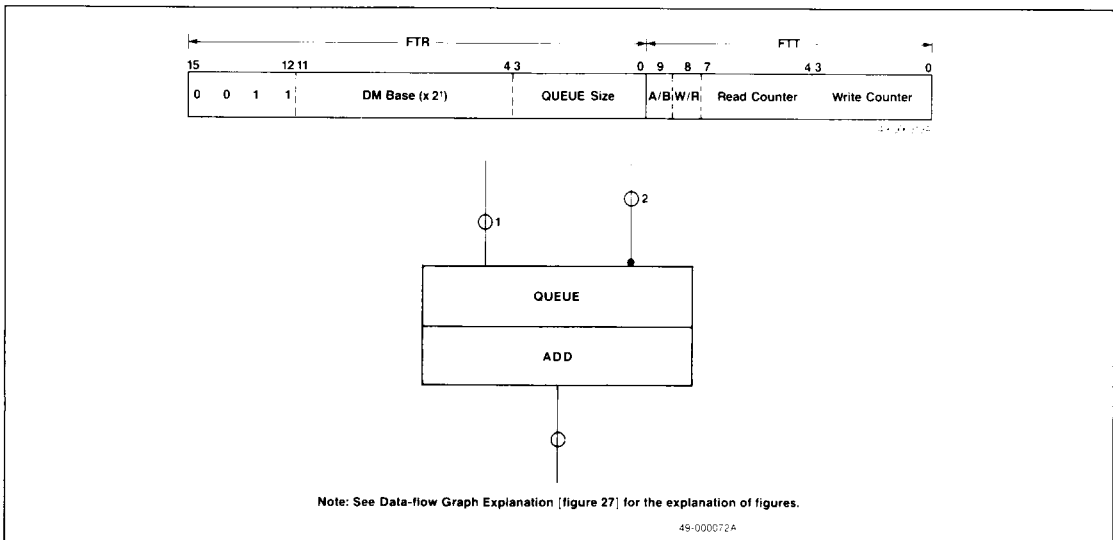


3g

## QUEUE

For a two-operand instruction, a QUEUE instruction is used to temporarily store the first operand token in the Data Memory until the second operand token arrives. The maximum Queue size is 16. See figure 8.

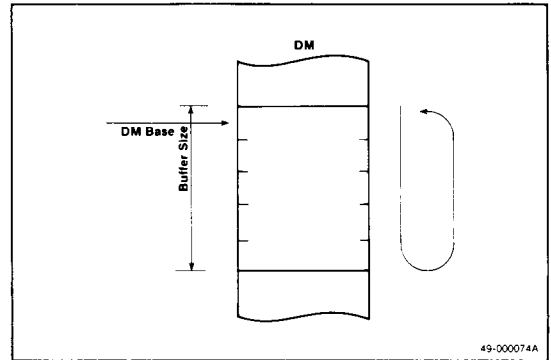
**Figure 8. QUEUE Instruction**



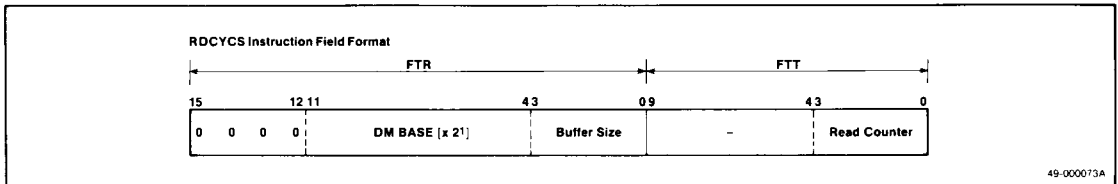
**RDCYCS [Read Cyclic Short]**

RDCYCS reads 18-bit data words from the Data Memory cyclically (see figure 9). The first data to be read is specified by the DM Base address. The last data to be read is specified by the buffer size. The Read Counter (RC) contains the offset address from Data Memory Base (DMB) address. It is incremented each time the Data Memory is accessed. The maximum buffer size is 16.

**Figure 9. RDCYCS Instruction Operation**



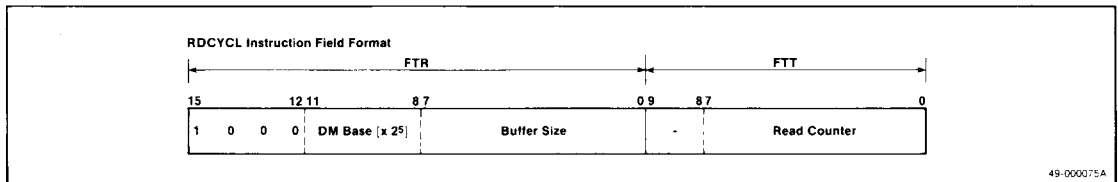
49-000074A



**RDCYCL [Read Cyclic Long]**

RDCYCL reads 18-bit data words from the Data Memory in a cyclic manner like RDCYCS but has a longer cyclic

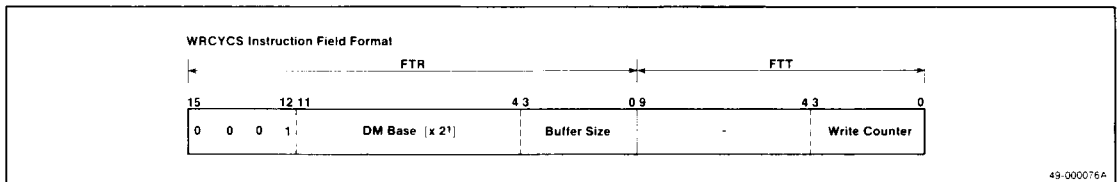
range. The first data to be read is specified by the DM Base address. The last data to be read is specified by the buffer size. The maximum buffer size is 256.



**WRCYCS [Write Cyclic Short]**

WRCYCS writes 18-bit data words into the Data Memory cyclically. The first the Data Memory address

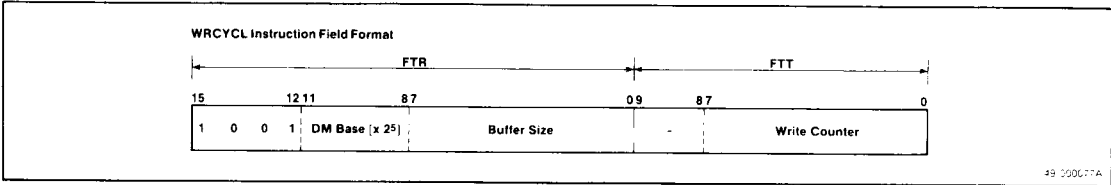
is specified by the DM Base address. The last address is specified by the buffer size. The maximum buffer size is 16.



**WRCYCL [Write Cyclic Long]**

WRCYCL writes 18-bit data words into the data memory in a cyclic manner similar to WRCYCS but has a longer

cyclic range. The first DM address is specified by the DM Base address. The last address is specified by the buffer size. The maximum buffer size is 256.



## RDWR [Read/Write Data Memory]

RDWR is used to write or read data to and from the Data Memory. This instruction reads/modifies/writes the Data Memory with the Address Register as index.

If a token arriving at the instruction has FTRC bit = 0, then the instruction performs a DM read operation. If it has FTRC bit = 1, then the instruction performs a DM write operation.

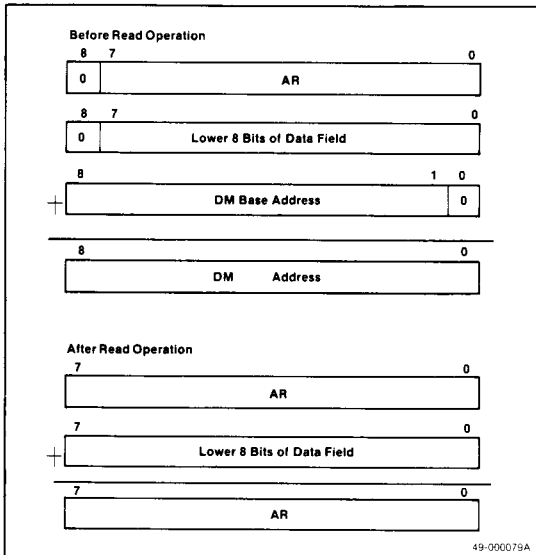
For a token with the FTRC bit = 0, the actual DM address location to be read is determined by the sum of the following three values: 8-bit Address Register (AR),

the lower eight bits of the data field of the token, and the DM Base address. After the read operation, the lower eight bits of the token's data field is added to the value of AR. Additionally, the data field of the token is replaced by the contents read from the Data Memory location.

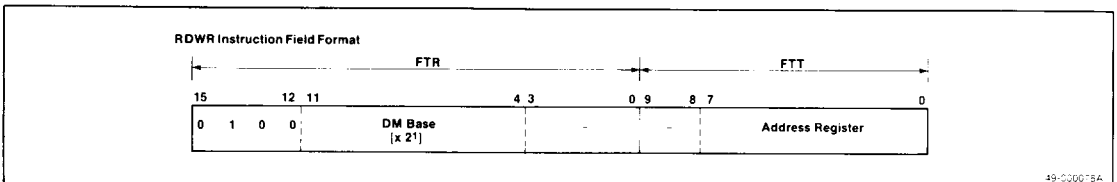
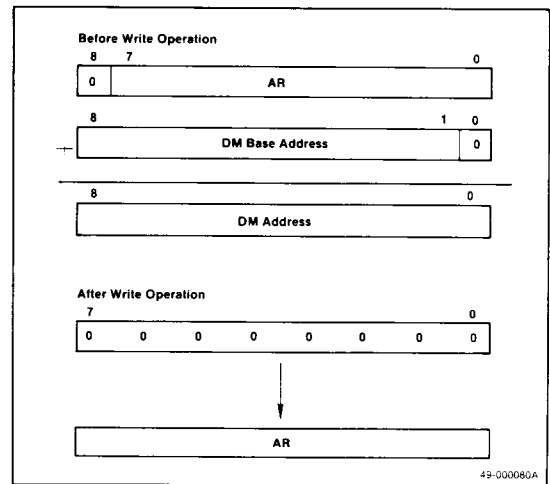
If a token with FTRC bit = 1 is used along with RDWR, a write operation is performed. The Data Memory address location is determined by the sum of 8-bit AR and DM Base address. The 18-bit data from the token is written into the DM address calculated. After the write operation, AR is reset to 00H.

3g

### FTRC = 0



### FTRC = 1



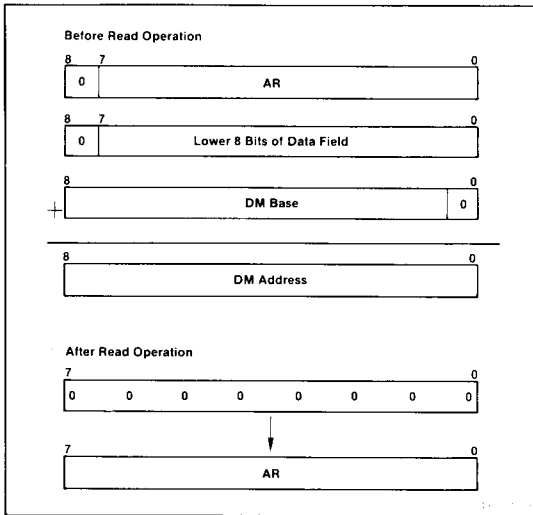
**RDIDX [Read Data Memory with Index]**

RDIDX is used to read the contents of the Data Memory. This instruction is most useful when a part of the Data Memory is used as a look-up table. The RDIDX instruction performs different operations depending upon the FTRC bit of the token using the instruction. If the FTRC bit = 0, then the instruction reads a Data Memory location. The DM address location to be read is determined by the sum of the following three values: the 8-bit AR, the lower eight bits

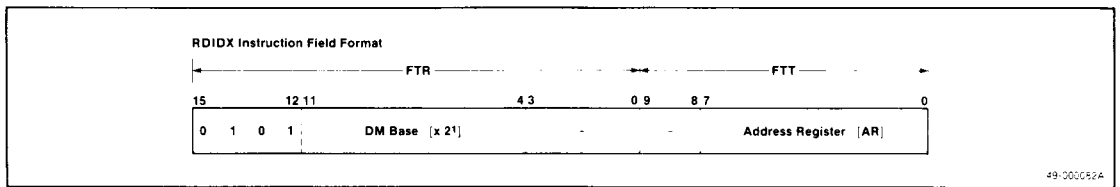
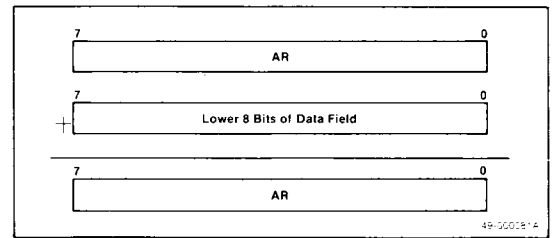
of the token's data field, and the DM Base address. After the read operation, the data field of the token is replaced by the contents of the Data Memory location read. The value of AR is reset to zero after the operation.

If the FTRC bit = 1, no operation is performed on the Data Memory. However, the token's AR contents are replaced by the modulo-256 sum of the lower eight bits of data field and the current contents of AR.

**FTRC = 0**



**FTRC = 1**





## PICKUP [Pickup Data Stream]

This instruction picks up every  $(n+1)^{\text{th}}$  token from a stream of incoming tokens and increments the  $(n+1)^{\text{th}}$  token's ID field by one. The number  $n$  is specified by the Count

Size (CS) of the Function Table Right field.

Figure 10 illustrates the PICKUP instruction with CS = 3.

**Note:** These figures use the data-flow graph convention. See figure 27, Data-flow Graph Explanation for the explanation of figures.

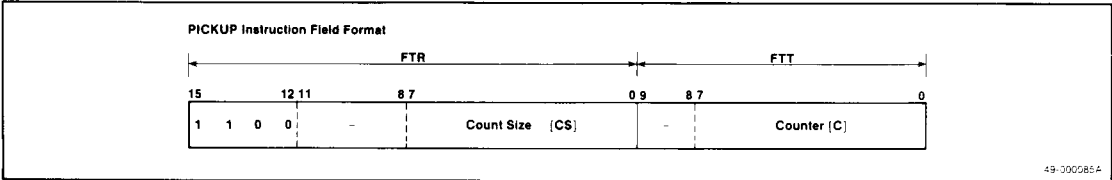
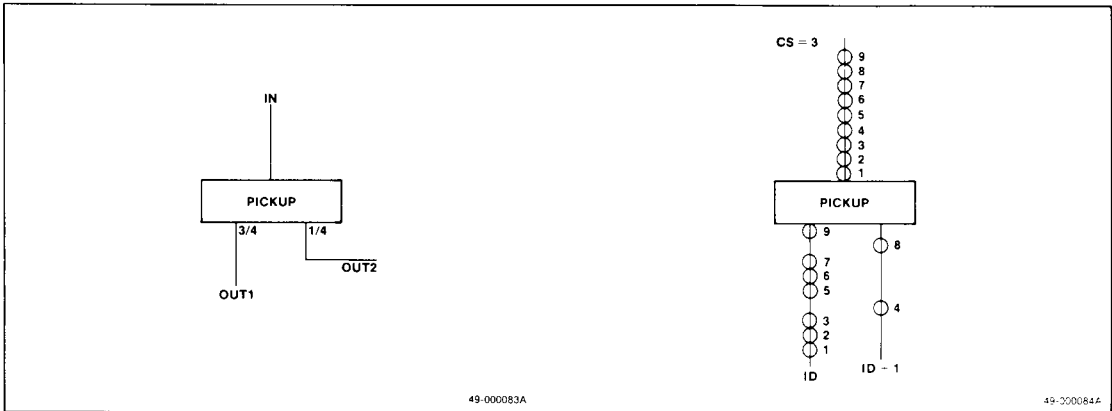


Figure 10. Pickup Instruction



3g

## COUNT [Count Data Stream]

COUNT copies every  $(n+1)^{\text{th}}$  token from a stream of incoming tokens and increments the copied token's ID

field by one. The number  $n$  is specified by CS of the Function Table Right field. Figure 11 illustrates the COUNT instruction with CS = 3.

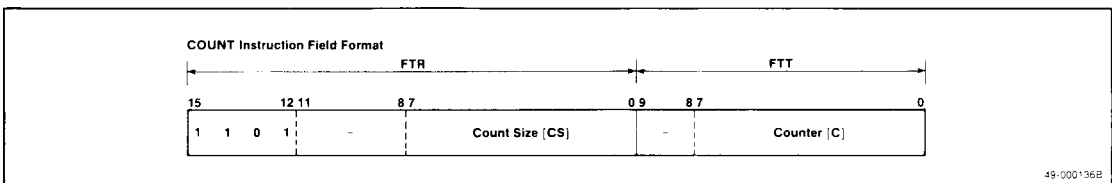
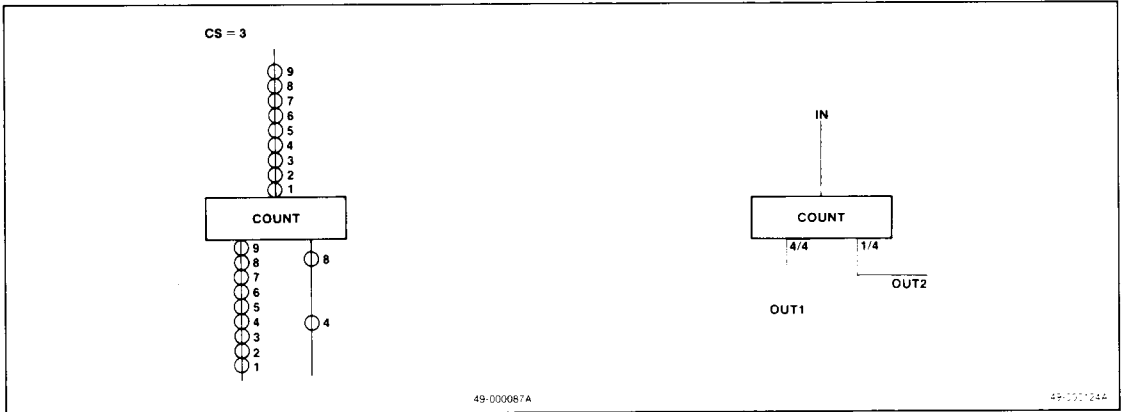


Figure 11. COUNT Instruction



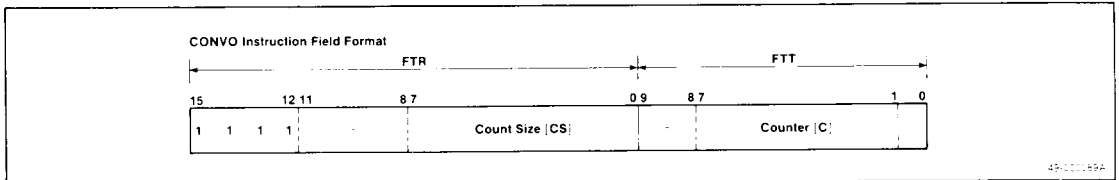
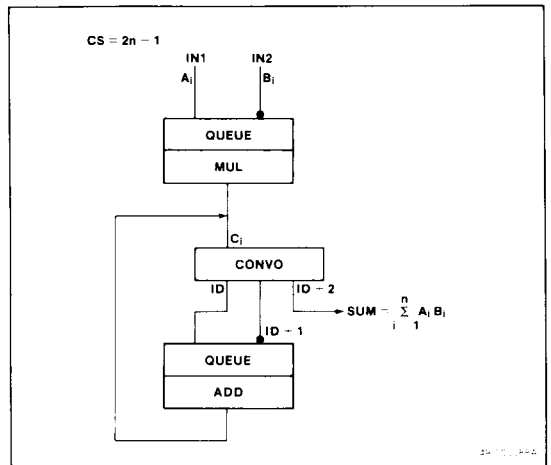
**CONVO [Convolve]**

CONVO instruction is used to perform cumulative operations such as  $\sum A_i$  or  $\prod A_i$ . The CONVO instruction is best suited for convolving two sequences of the same length. Figure 12 illustrates the CONVO instruction by computing

$$SUM = \sum_{i=1}^n A_i B_i$$

The  $A_i$  sequence is input to IN1 while the  $B_i$  sequence is input to IN2. Together they are queued and multiplied to form the  $C_i$  sequence. The  $C_i$ 's arriving at CONVO instruction are queued and added together to form the final answer SUM. The length of the summation,  $n$ , is specified by the CS.

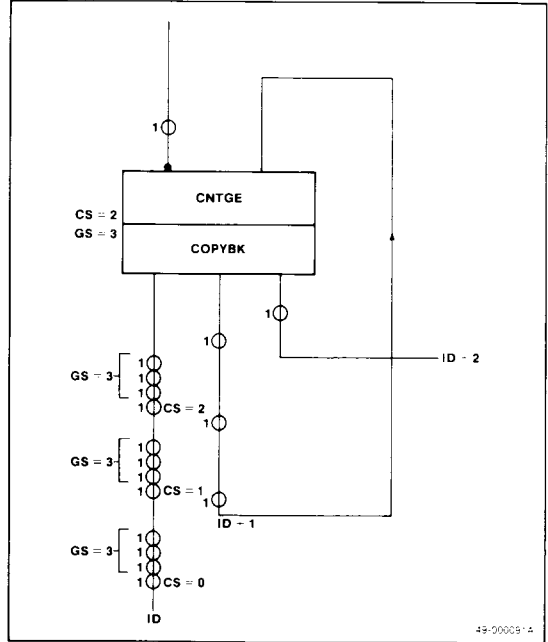
Figure 12. CONVO Instruction



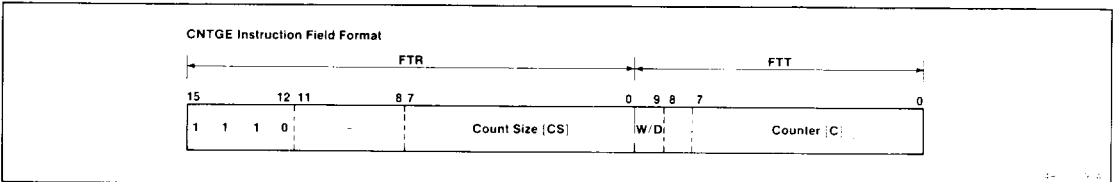
## CNTGE [Count Generation]

CNTGE is normally used with COPYBK (Copy Block) to generate more than 16 copies of a single token (see figure 13). This instruction has both the dead (inactive) state and the wait (active) state. The instruction starts in the dead state. The FTRC bit = 0 tokens that arrive during the dead state of instruction are output to the ID + 2 token stream. It enters the wait state when a token with FTRC bit = 1 arrives and the token is output to ID token stream. Once the instruction is in the wait state, it counts the number of tokens arriving with FTRC bit = 0, outputting them to the ID token stream, until the number exceeds the number specified by CS. If Counter (C) reaches the number specified by Count Size (CS), the instruction automatically enters the dead state. Tokens with the FTRC bit = 1 arriving at CNTGE while the instruction is in the wait state are deleted by the instruction. Once the instruction enters the dead state, it can be reactivated by the arrival of a token with FTRC bit = 1.

Figure 13. CNTGE Instruction



3g

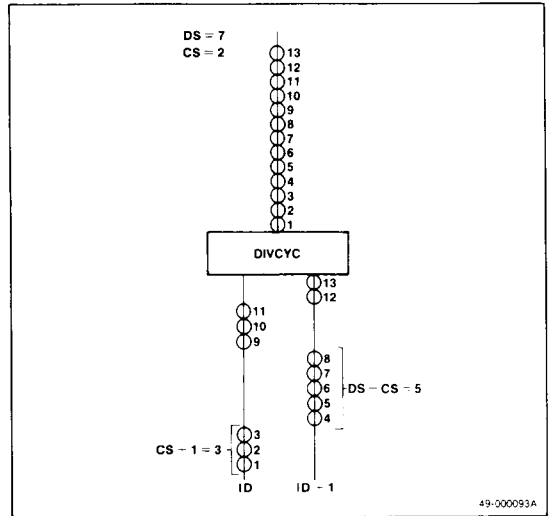


**DIVCYC [Divide Cyclic]**

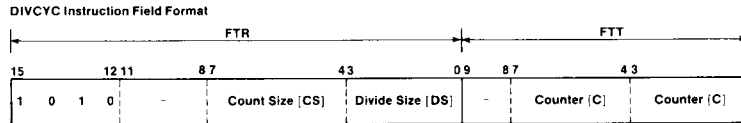
DIVCYC divides an incoming stream of tokens into two streams of tokens: an ID token stream and an ID + 1 token stream. The pattern in which the incoming tokens are divided is specified by the Divide Size (DS) and Count Size (CS). The DS specifies cycle size whereas CS specifies the number of consecutive tokens to be in the ID stream. The first CS + 1 tokens are output to the ID token stream. The following consecutive (DS - CS) tokens are output to the ID + 1 token stream.

Figure 14 illustrates the DIVCYC instruction with DS = 7 and CS = 2. Note that an incoming stream of tokens is divided into a stream of ID tokens and a stream of ID + 1 tokens with a cycle of 8 tokens. Since CS = 2, the number of ID tokens in one cycle is 3, the number of ID + 1 tokens in a cycle is 5.

**Figure 14. DIVCYC Instruction**



49-000093A

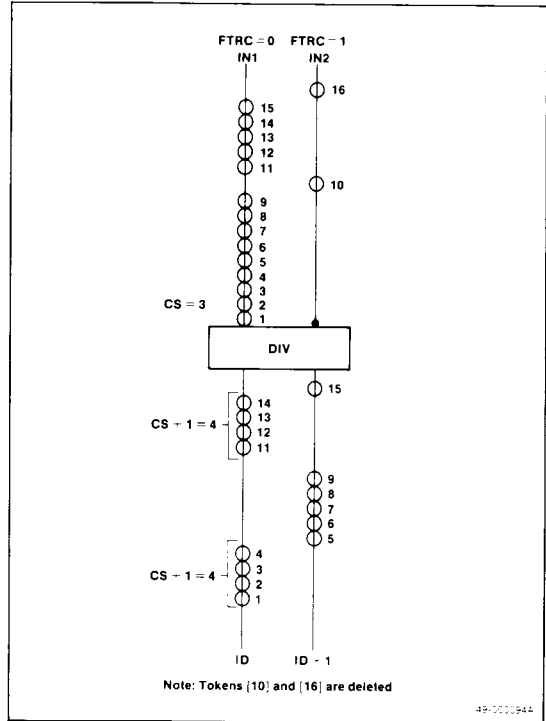


49-000092A

## DIV [Divide]

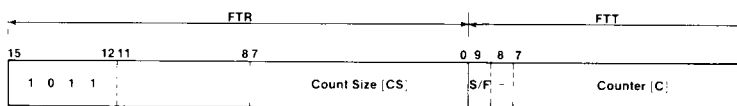
DIV with CS = n divides an incoming stream of tokens with FTRC bit = 0 into two streams of tokens: ID tokens and ID + 1 tokens. The first (n + 1) incoming tokens with FTRC bit = 0 are output as the ID tokens, and the rest of the incoming tokens with FTRC bit = 0 are output as ID + 1 tokens. An incoming token with FTRC bit = 1 is used to reinitialize the DIV instruction. The stream of input tokens with FTRC bit = 0 after the reinitialization is again divided into a stream of (n + 1) ID tokens followed by ID + 1 tokens. A token with FTRC bit = 1 which reinitializes the DIV instruction is deleted from the output token stream. A DIV instruction with CS = 3 is illustrated in figure 15. The 10th and 16th input tokens have FTRC bit = 1, so they reinitialize the DIV instruction.

Figure 15. DIV Instruction



3g

DIV Instruction Field Format

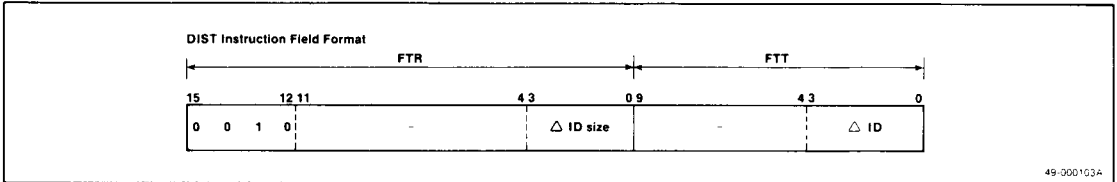
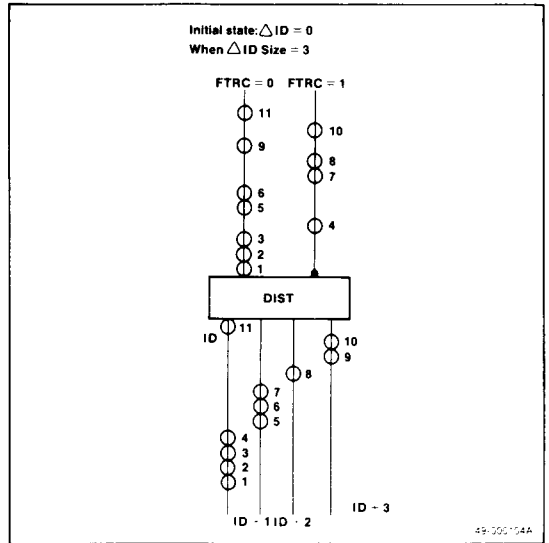


49-000096B

**DIST [Distribute]**

DIST is used to divide a stream of incoming tokens with the same ID into more than one stream of tokens with different IDs (see figure 16). The ΔID size determines the maximum number of output token streams the instruction can have. ΔID is the value added to an incoming token's ID field to form the ID field of the output token. The ΔID field is initially set to zero, and it is incremented by one after a token with FTRC bit = 1 passes through the instruction. However, a token with FTRC bit = 0 has no effect on the value of ΔID field. If the value of ΔID before being incremented by a token with the FTRC bit = 1 is equal to the contents of the ΔID size field, the ΔID field will be reset to zero.

**Figure 16. DIST Instruction**

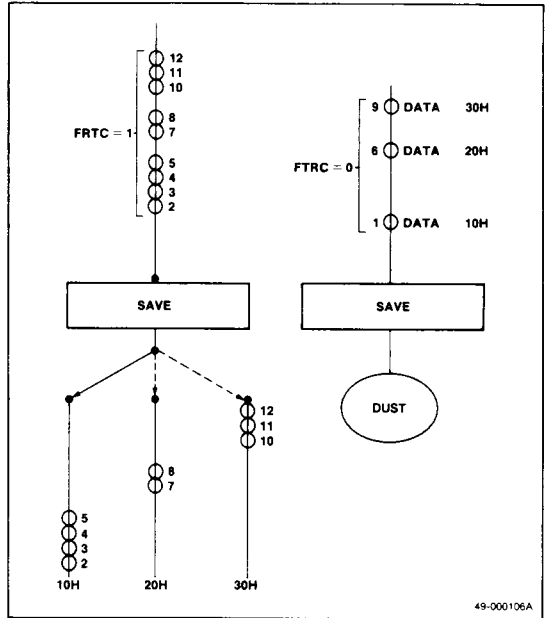


## SAVE [Save ID]

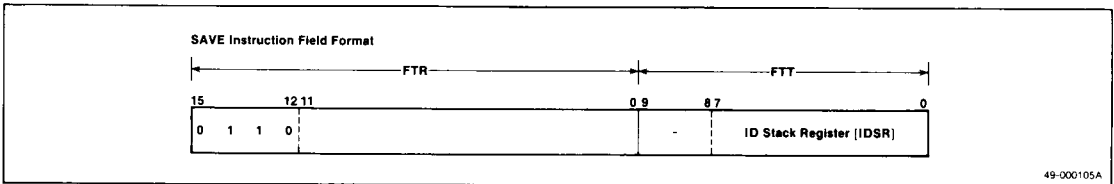
SAVE is used to set the value of the ID field of a token. The instruction performs two different operations depending on whether the token's FTRC bit is 1 or 0. If the token's FTRC bit = 0, the instruction copies the lower eight bits of the data field into the Identifier Stack Register (IDSR) field. However, if the token's FTRC bit is 1, the instruction replaces the token's ID field with the contents of IDSR.

Figure 17 illustrates the use of the SAVE instruction. Token 1 assigns an ID field value of 10H to tokens 2, 3, 4 and 5, token 6 assigns an ID field value of 20H to tokens 7 and 8, and token 9 assigns an ID field value of 30H to tokens 10, 11 and 12. In this example, tokens 1, 6 and 9 are deleted after SAVE instruction.

Figure 17. SAVE Instruction



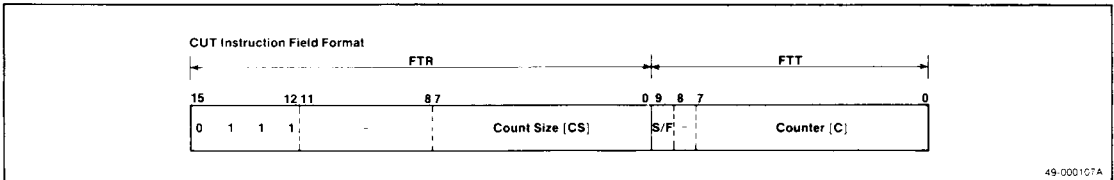
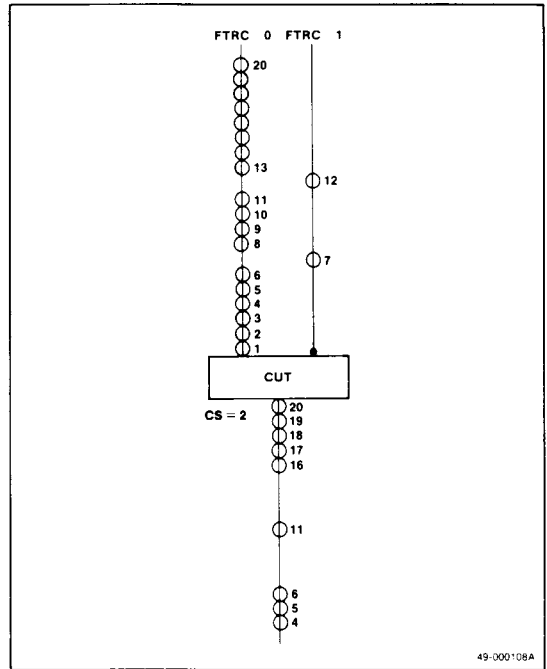
3g



**CUT [Cut Data Stream]**

CUT is used to delete unnecessary tokens from a series of incoming tokens. The first  $n$  tokens arriving at the instruction are deleted, where  $n$  is the value contained in the CS field of the instruction. Initially the S/F bit and the Counter (C) are set to zero. When a token with its FTRC bit = 0 enters the instruction while S/F bit is zero, the token increments the Counter by one and the token itself is deleted. As the first  $(n + 1)$  tokens are deleted by the instruction, the Counter has the same value as  $n$ , the contents of CS field. This condition sets the S/F bit to 1. When the S/F bit is 1, a token with its FTRC bit = 0 can pass through the instruction without being deleted. However, if a token with its FTRC bit = 1 passes through the instruction, it resets the S/F bit to 0, thereby reinitializing the instruction. The token with its FTRC bit = 1 is also deleted after reinitializing the instruction. Figure 18 illustrates the use of CUT to delete tokens 7 and 12 and the three tokens following them.

**Figure 18. CUT Instruction**



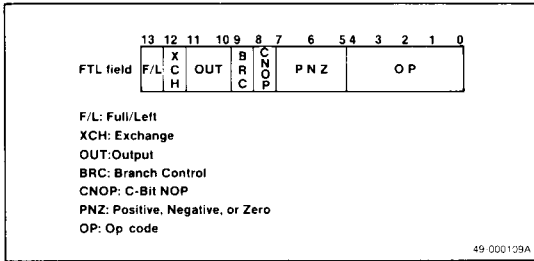


**Table 10. AG and FC Instructions**

Mnemonic	FTR (16)										FTT (10)								FTRC (1)	Operation
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
QUEUE	0	0	1	1	DM Base (x 2 <sup>1</sup> ) (8)		Queue Size (4)	A / B	W / R	Read Counter (4)		Write Counter (4)					Synchronize two tokens			
RDCYCS	0	0	0	0	DM Base (x 2 <sup>1</sup> ) (DMB) (8)		Buffer Size (BS) (4)	(6)		Read Counter (RC) (4)				0			DATA ← (DMB + RC), RC ← RC + 1			
													1			DATA ← (DMB + RC), RC ← RC + 1, when BS = RC, copy with ID + 1				
RDCYCL	1	0	0	0	DM Base (x 2 <sup>5</sup> ) (4)	Buffer Size (8)	(2)	Read Counter (8)						0		DATA ← (DMB + RC), RC ← RC + 1				
													1			DATA ← (DMB + RC), RC ← RC + 1, when BS = RC, copy with ID + 1				
WRCYCS	0	0	0	1	Base (x 2 <sup>1</sup> ) (8)		Buffer Size (4)	(6)		Write Counter (WC) (4)				0		(DMB + WC) ← DATA, WC ← WC + 1, delete token				
													1			(DMB + WC) ← DATA, WC ← WC + 1, when BS = WC, token not deleted				
WRCYCL	1	0	0	1	DM Base (x 2 <sup>5</sup> ) (4)	Buffer Size (8)	(2)	Write Counter (8)						0		(DMB + WC) ← DATA, WC ← WC + 1, delete token				
													1			(DMB + WC) ← DATA, WC ← WC + 1, when BS = WC, token not deleted				
RDWR	0	1	0	0	DM Base (x 2 <sup>1</sup> ) (8)		(4)	(2)	Address Register (AR) (8)					0		DATA ← (DMB + AR + DATA), AR ← AR + DATA				
													1			(DMB + AR) ← DATA, AR ← 0				
RDIDX	0	1	0	1	DM Base (x 2 <sup>1</sup> ) (8)		(4)	(2)	Address Register (8)					0		DATA ← (DMB + AR + DATA), AR ← 0				
													1			AR ← AR + DATA				
PICKUP	1	1	0	0	(4)	Count Size (CS) (8)	(2)	Counter (C) (8)						0		When CS ≠ C, C ← C + 1; when CS = C, distribute, C ← 0				
													1			C ← C + DATA, token deleted				
COUNT	1	1	0	1	(4)	Count Size (8)	(2)	Counter (8)						0		When CS ≠ C, C ← C + 1; when CS = C, copy token, C ← 0				
													1			C ← C + DATA, token deleted				
CUT	0	1	1	1	(4)	Count Size (8)	S / F (1)	Counter (8)						0		When S/F = 0 and C ≤ CS, C ← C + 1, delete token; when S/F = 0 and C > CS, or when S/F = 1, C ← C + 1, token not deleted				
													1			S/F ← 0, C ← 0, token deleted				
DIVCYC	1	0	1	0	(4)	Count Size (4)	Divide Size (4)	(2)	Counter (4)	Counter (4)				0		When C ≤ CS, C ← C + 1; when C > CS, distribute, C ← C + 1; C ← C. When C = DS, C ← 0				
													1			C ← C + DATA, token deleted				
DIV	1	0	1	1	(4)	Count Size (8)	S / F (1)	Counter (8)						0		When S/F = 0 and C ≤ CS, C ← C + 1; when S/F = 0 and C > CS, or when S/F = 1, distribute, C ← C + 1;				
													1			S/F ← 0, C ← 0, token deleted				
DIST	0	0	1	0	(8)	Δ ID Size (4)	(6)	Δ ID (4)						0		ID ← (ID + Δ ID) modulo Δ ID size				
													1			When Δ ID ≠ Δ ID size, ID ← (ID + Δ ID) modulo Δ ID size, Δ ID ← Δ ID + 1. When Δ ID = Δ ID size, Δ ID ← 0				
CONVO	1	1	1	1	(4)	Count Size (8)	(2)	Counter (7)	(1)					0		When CS ≠ C, ID ← ID + C (modulo 2), C ← C + 1; when CS = C, ID ← ID + 2, C ← 0				
SAVE	0	1	1	0	(12)		(2)	ID Stack Register (8) (IDSR)						0		IDSR ← Lower 8-bit of DATA				
													1			ID ← IDSR				
CNTGE	1	1	1	0	(4)	Count Size (8)	W / D (1)	Counter (8)						0		When dead, ID ← ID + 2; when wait, if C = CS, C ← 0, W/D = 0; when wait, if C ≠ CS, C ← C + 1				
													1			When dead, initialization; when wait, delete token				

3g

**PU Instructions**



**Figure 19. The Processing Unit**

**Bit Assignments**

**F/L [Full/Left]:** F/L bit = 0 indicates that the PU instruction is a one-operand instruction, and only the Function Table Left field is meaningful. F/L bit = 1 indicates that the PU instruction is a two-operand instruction, and both the Function Table Left field and the Function Table Right field are meaningful. Therefore, when F/L bit = 1, the PU instruction is used in conjunction with an AG/FC instruction.

**XCH [Exchange]:** This bit controls the exchange operation. Operands will be exchanged just before the two tokens enter the QUEUE when XCH = 1.

**OUT [Output]:** There are four different PU output token formats. The two OUT bits specify the output token format. See table 11.

**Table 11. OUT Bits**

OUT Bits	No. of Outputs	First Output		Second Output	
		ID	DATA, C, S	ID	DATA, C, S
00	1	ID	$X^1$		
01	1	ID	$Y^2$		
10	2	ID	X	ID + 1	X
11	2	ID	X	ID + 1	Y

- Notes:**
1. This is the 18-bit result of the operation output to the X side. It includes the  $C_X$  and  $S_X$  bits.
  2. This is the 18-bit result of the operation output to the Y side. It includes the  $C_Y$  and  $S_Y$  bits.

**BRC [Branch Control]:** The BRC bit controls the flow of the PU output data token. The output data token may be output to either the ID token stream or the ID + 1 token stream. When the BRC bit is set to 1 and the C bit of the PU output data token is also 1, the output data token is sent to the ID + 1 token stream. But when the BRC bit is set to 1 and the C bit of the output data token is 0, the token is sent to the ID token stream. Therefore, using the BRC bit implements a conditional branch on C.

**CNOP Bit:** This bit informs the Processing Unit whether or not the incoming token should be processed. If the CNOP bit is set, and the  $C_A$  bit is not equal to the  $C_B$  bit, then the token passes through the Processing Unit with no operation performed. See table 12.

**Table 12. CNOP Bit**

$C_A$	$C_B$	PU Operation
0	0	Processing specified by the OP code is performed.
0	1	Token passes through the Processing Unit as NOP.
1	0	Token passes through the Processing Unit as NOP.
1	1	Processing specified by the OP code is performed.

**PNZ [Positive, Negative, Zero] Field:** The PNZ field is used to test the resulting condition of the PU operation. If the resulting condition matches the condition set by the PNZ field, then the C bit of the output data token is set to 1. See table 13.

**Table 13. PNZ Field**

P	N	Z	Condition	C <sub>X</sub>	C <sub>Y</sub>	Assembler Description	
0	0	0	No condition set	C <sub>A</sub>	C <sub>B</sub>		
0	0	1	Result of operation = 0	1	1	EQ	True
			Result of operation ≠ 0	0	0		False
0	1	0	Result of operation < 0	1	1	LT	True
			Result of operation ≥ 0	0	0		False
0	1	1	Result of operation ≤ 0	1	1	LE	True
			Result of operation > 0	0	0		False
1	0	0	Result of operation > 0	1	1	GT	True
			Result of operation ≤ 0	0	0		False
1	0	1	Result of operation ≥ 0	1	1	GE	True
			Result of operation < 0	0	0		False
1	1	0	Result of operation ≠ 0	1	1	NE	True
			Result of operation = 0	0	0		False
1	1	1	Overflow generated	1	1	OVF	True
			No overflow generated	0	0		False

**OP Code Field:** This 5-bit OP code field specifies the PU operations to be performed. See table 14

**Table 14. OP Code Field**

Instruction	Mnemonic	Opcode
Logical	OR	0000
	AND	0001
	XOR	0010
	ANDNOT	0011
	NOT	0110
Arithmetic	ADD	1100
	ADSC	1110
	SUB	1101
	SUBSC	1111
	MUL	11010
	MULSC	11110
	NOP	11011
	NOPSC	11111
	INC	01010
	DEC	01011
Shift	SHL	00100
	SHLBRV	00101
	SHR	00110
	SHRBRV	00111
Compare	CMPNOM	01000
	CMP	01001
	CMPXCH	10001
Bit manipulation	GET1	10101
	SET1	10110
	CLR1	10111
Bit check	ANDMSK	01101
	ORMSK	10000
Data conversion	CVT2AB	01110
	CVTAB2	01111
Double precision adjust	ADJL	10100
Accumulative addition	ACC	10010
C bit copy	COPYC	10011

3g

### Logical Instructions

These instructions perform 16-bit logical operations on DATA<sub>A</sub> and DATA<sub>B</sub>. Usually there are no changes in C and S bits between the input token and the output token, however C bits can be affected by PNZ condition when specified.

**OR, AND, XOR:** These instructions perform 16-bit logical OR, AND, and XOR operations using input data tokens from the A and B sides of the Processing Unit. The 16 bit result is output to the X side.

**ANDNOT:** This instruction first complements DATA<sub>A</sub> and then performs logical AND operation with DATA<sub>B</sub>. The 16-bit result is output to the X side.

**NOT:** This is a one-operand instruction which requires 16-bit data input from the A side only. The B side input is ignored. This instruction complements the 16-bit input data from the A side. The 16-bit result is output to the X side.

### Arithmetic Instructions

These instructions perform 17-bit (including the sign bit) arithmetic operations on DATA<sub>A</sub> and DATA<sub>B</sub>. When a PNZ condition is specified, the C bits of output data, C<sub>X</sub> and C<sub>Y</sub>, reflect the setting. However, if no PNZ condition is specified (i.e., PNZ = 000), then C<sub>X</sub> ← C<sub>A</sub> and C<sub>Y</sub> ← C<sub>B</sub>.

**ADD, SUB:** These instructions perform addition or subtraction on DATA<sub>A</sub> and DATA<sub>B</sub> along with the sign bits, S<sub>A</sub> and S<sub>B</sub>. The result is output to the X side. DATA<sub>Y</sub> is normally 0000H. However, if an overflow occurs, then DATA<sub>Y</sub> is equal to +0001H (S<sub>Y</sub> = 0). If an underflow occurs, then the DATA<sub>Y</sub> is equal to -0001H (S<sub>Y</sub> = 1).

**MUL:** This instruction multiplies DATA<sub>A</sub> and DATA<sub>B</sub>. The correct sign bit for the product is determined from S<sub>A</sub> and S<sub>B</sub>. The 33-bit result including a sign bit is output as two 17-bit words, S<sub>X</sub> and DATA<sub>X</sub>, followed by S<sub>Y</sub> and DATA<sub>Y</sub>. DATA<sub>X</sub> is the upper 16-bit word and DATA<sub>Y</sub> is the lower 16-bit word. S<sub>X</sub> holds the resulting sign bit, and S<sub>Y</sub> is a mere duplicate of S<sub>X</sub>.

**NOP:** This instruction performs no operation on the input token. The input data from A and B sides are output to the X and Y sides, respectively, without any change in their contents. If any control other than the OP code (such as PNZ control, BRC control, etc.) has been specified, the output complies with the control.

### Shift Count Instructions

These four Shift Count (SC) instructions first perform the normal operations, then count the number of leading zeros in DATA<sub>X</sub> of the result, and finally output

the number of zeros as DATA<sub>Y</sub> (see table 15). These instructions are provided for easy floating point processing.

**ADDSC, SUBSC, NOPSC:** These instructions perform addition, subtraction, or no operation. The number of preceding zeros in DATA<sub>X</sub> of the result is output as DATA<sub>Y</sub>. If an overflow or an underflow occurs as a result of an operation, DATA<sub>Y</sub> contains +0001H (S<sub>Y</sub> = 0) or -0001H (S<sub>Y</sub> = 1), respectively.

**MULSC:** This instruction performs a normal multiplication operation using the two 17-bit data. The upper order 16-bit data and its sign bit are output as DATA<sub>X</sub> and S<sub>X</sub>, but the lower 16-bit data is not output as DATA<sub>Y</sub>. Instead, the number of preceding zeros in DATA<sub>X</sub> are counted and output as DATA<sub>Y</sub>. The S<sub>Y</sub> bit is always zero.

**Table 15. Shift Count Operation**

DATA <sub>X</sub> After Operation																	SC Output (Y)	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	S <sub>Y</sub>	Y Data	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 0 1 0 H	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0 0 0 F H	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	x	0	0 0 0 E H	
0	0	0	0	0	0	0	0	0	0	0	0	0	1	x	x	0	0 0 0 D H	
0	0	0	0	0	0	0	0	0	0	0	1	x	x	x	x	0	0 0 0 C H	
0	0	0	0	0	0	0	0	0	0	1	x	x	x	x	x	0	0 0 0 B H	
0	0	0	0	0	0	0	0	1	x	x	x	x	x	x	x	0	0 0 0 A H	
0	0	0	0	0	0	0	1	x	x	x	x	x	x	x	x	0	0 0 0 9 H	
0	0	0	0	0	0	1	x	x	x	x	x	x	x	x	x	0	0 0 0 8 H	
0	0	0	0	0	1	x	x	x	x	x	x	x	x	x	x	0	0 0 0 7 H	
0	0	0	0	1	x	x	x	x	x	x	x	x	x	x	x	0	0 0 0 6 H	
0	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	0	0 0 0 5 H	
0	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	0	0 0 0 4 H	
0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0 0 0 3 H	
0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0 0 0 2 H	
0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0 0 0 1 H	
1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0 0 0 0 H*	

**Notes:** \* When an overflow or underflow has occurred x don't care

### Increment and Decrement Instructions

**INC, DEC:** These instructions increment or decrement the 17-bit data from the A side (S<sub>A</sub> and DATA<sub>A</sub>), and outputs the result to X side as S<sub>X</sub> and DATA<sub>X</sub>. The S<sub>Y</sub> and DATA<sub>Y</sub> are normally zero. However, if an overflow or an underflow occurs, then the Y side outputs +0001H (S<sub>Y</sub> = 0) or -0001H (S<sub>Y</sub> = 1), respectively.

### Shift Instructions

**SHR [Shift Right], SHL [Shift Left]:** SHR or SHL instructions perform a barrel-shifting operation on the 16-bit data, DATA<sub>A</sub>. The actual number of shifts and the direction is further specified by the lower five bits of DATA<sub>B</sub> and S<sub>B</sub>, respectively. See figure 20 for detailed operation explanations.

**Figure 20. SHR and SHL**

Right Shift (SHR execution)

S <sub>8</sub>	Lower 5 bits of DATA <sub>8</sub> (No. of shifts)	DATA <sub>x</sub>	DATA <sub>y</sub>
0	0 0 0 0 0	A <sub>15</sub> A <sub>14</sub> ...A <sub>1</sub> A <sub>0</sub>	0...0
0	0 0 0 0 1	0 A <sub>15</sub> A <sub>14</sub> ...A <sub>1</sub>	A <sub>0</sub> 0...0
0	0 0 0 1 0	0 0 A <sub>15</sub> ...A <sub>2</sub>	AA 1 0 0...0
0	0 0 0 1 1	0...0 A <sub>15</sub> ...A <sub>3</sub>	A <sub>2</sub> A <sub>0</sub> 0...0
0	0 0 1 0 0	0...0 A <sub>15</sub> ...A <sub>4</sub>	A <sub>3</sub> ...A <sub>0</sub> 0...0
0	0 0 1 0 1	0...0 A <sub>15</sub> ...A <sub>5</sub>	A <sub>4</sub> ...A <sub>0</sub> 0...0
0	0 0 1 1 0	0...0 A <sub>15</sub> ...A <sub>6</sub>	A <sub>5</sub> ...A <sub>0</sub> 0...0
0	0 0 1 1 1	0...0 A <sub>15</sub> ...A <sub>7</sub>	A <sub>6</sub> ...A <sub>0</sub> 0...0
0	0 1 0 0 0	0...0 A <sub>15</sub> ...A <sub>8</sub>	A <sub>7</sub> ...A <sub>0</sub> 0...0
0	0 1 0 0 1	0...0 A <sub>15</sub> ...A <sub>9</sub>	A <sub>8</sub> ...A <sub>0</sub> 0...0
0	0 1 0 1 0	0...0 A <sub>15</sub> ...A <sub>10</sub>	A <sub>9</sub> ...A <sub>0</sub> 0...0
0	0 1 0 1 1	0...0 A <sub>15</sub> ...A <sub>11</sub>	A <sub>10</sub> ...A <sub>0</sub> 0...0
0	0 1 1 0 0	0...0 A <sub>15</sub> A <sub>12</sub>	A <sub>11</sub> ...A <sub>0</sub> 0...0
0	0 1 1 0 1	0...0 A <sub>15</sub> A <sub>13</sub> 13	A <sub>12</sub> ...A <sub>0</sub> 0...0
0	0 1 1 1 0	0...0 AA 1514	A <sub>13</sub> ...A <sub>0</sub> 0 0
0	0 1 1 1 1	0...0 A 15	A <sub>14</sub> ...A <sub>0</sub> 0
0	1 X X X X	0...0	A <sub>15</sub> ...A <sub>1</sub> A <sub>0</sub>
1	0 0 0 0 0	A <sub>15</sub> A <sub>14</sub> ...A <sub>1</sub> A <sub>0</sub>	0...0
1	0 0 0 0 1	A <sub>14</sub> ...A <sub>0</sub> 0	0...0 A 15
1	0 0 0 1 0	A <sub>13</sub> ...A <sub>0</sub> 0 0	0...0 AA 15 14
1	0 0 0 1 1	A <sub>12</sub> ...A <sub>0</sub> 0...0	0...0 A A 15 13
1	0 0 1 0 0	A <sub>11</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> A <sub>12</sub>
1	0 0 1 0 1	A <sub>10</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>11</sub>
1	0 0 1 1 0	A <sub>9</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>10</sub>
1	0 0 1 1 1	A <sub>8</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>9</sub>
1	0 1 0 0 0	A <sub>7</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>8</sub>
1	0 1 0 0 1	A <sub>6</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>7</sub>
1	0 1 0 1 0	A <sub>5</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>6</sub>
1	0 1 0 1 1	A <sub>4</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>5</sub>
1	0 1 1 0 0	A <sub>3</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>4</sub>
1	0 1 1 0 1	A <sub>2</sub> A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>3</sub>
1	0 1 1 1 0	AA 1 0 0...0	0 0 A <sub>15</sub> ...A <sub>2</sub>
1	0 1 1 1 1	A 0 0...0	0 A <sub>15</sub> ...A <sub>1</sub>
1	1 X X X X	0...0	A <sub>15</sub> ...A <sub>1</sub> A <sub>0</sub>

49-000137C

Left Shift (SHL execution)

S <sub>8</sub>	Lower 5 bits of DATA <sub>8</sub> (No. of shifts)	DATA <sub>x</sub>	DATA <sub>y</sub>
0	0 0 0 0 0	A <sub>15</sub> A <sub>14</sub> ...A <sub>1</sub> A <sub>0</sub>	0...0
0	0 0 0 0 1	A <sub>14</sub> ...A <sub>0</sub> 0	0...0 A 15
0	0 0 0 1 0	A <sub>13</sub> ...A <sub>0</sub> 0 0	0...0 AA 1514
0	0 0 0 1 1	A <sub>12</sub> ...A <sub>0</sub> 0...0	0...0 A A 15 13
0	0 0 1 0 0	A <sub>11</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> A <sub>12</sub>
0	0 0 1 0 1	A <sub>10</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>11</sub>
0	0 0 1 1 0	A <sub>9</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>10</sub>
0	0 0 1 1 1	A <sub>8</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>9</sub>
0	0 1 0 0 0	A <sub>7</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>8</sub>
0	0 1 0 0 1	A <sub>6</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>7</sub>
0	0 1 0 1 0	A <sub>5</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>6</sub>
0	0 1 0 1 1	A <sub>4</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>5</sub>
0	0 1 1 0 0	A <sub>3</sub> ...A <sub>0</sub> 0...0	0...0 A <sub>15</sub> ...A <sub>4</sub>
0	0 1 1 0 1	A <sub>2</sub> A <sub>0</sub> 0...0	0 0 A <sub>15</sub> ...A <sub>3</sub>
0	0 1 1 1 0	AA 1 0 0...0	0 0 A <sub>15</sub> ...A <sub>2</sub>
0	0 1 1 1 1	A 0 0...0	0 A <sub>15</sub> ...A <sub>1</sub>
0	1 X X X X	0...0	A <sub>15</sub> ...A <sub>0</sub>
1	0 0 0 0 0	A <sub>15</sub> A <sub>14</sub> ...A <sub>1</sub> A <sub>0</sub>	0...0
1	0 0 0 0 1	0 A <sub>15</sub> A <sub>14</sub> ...A <sub>1</sub>	A <sub>0</sub> 0...0
1	0 0 0 1 0	0 0 A <sub>15</sub> ...A <sub>2</sub>	AA 1 0 0...0
1	0 0 0 1 1	0...0 A <sub>15</sub> ...A <sub>3</sub>	A <sub>2</sub> A <sub>0</sub> 0...0
1	0 0 1 0 0	0...0 A <sub>15</sub> ...A <sub>4</sub>	A <sub>3</sub> ...A <sub>0</sub> 0...0
1	0 0 1 0 1	0...0 A <sub>15</sub> ...A <sub>5</sub>	A <sub>4</sub> ...A <sub>0</sub> 0...0
1	0 0 1 1 0	0...0 A <sub>15</sub> ...A <sub>6</sub>	A <sub>5</sub> ...A <sub>0</sub> 0...0
1	0 0 1 1 1	0...0 A <sub>15</sub> ...A <sub>7</sub>	A <sub>6</sub> ...A <sub>0</sub> 0...0
1	0 1 0 0 0	0...0 A <sub>15</sub> ...A <sub>8</sub>	A <sub>7</sub> ...A <sub>0</sub> 0...0
1	0 1 0 0 1	0...0 A <sub>15</sub> ...A <sub>9</sub>	A <sub>8</sub> ...A <sub>0</sub> 0...0
1	0 1 0 1 0	0...0 A <sub>15</sub> ...A <sub>10</sub>	A <sub>9</sub> ...A <sub>0</sub> 0...0
1	0 1 0 1 1	0...0 A <sub>15</sub> ...A <sub>11</sub>	A <sub>10</sub> ...A <sub>0</sub> 0...0
1	0 1 1 0 0	0...0 A <sub>15</sub> A <sub>12</sub>	A <sub>11</sub> ...A <sub>0</sub> 0...0
1	0 1 1 0 1	0...0 A <sub>15</sub> A <sub>13</sub> 13	0...0 A <sub>12</sub> ...A <sub>0</sub> 0...0
1	0 1 1 1 0	0...0 AA 1514	0...0 A <sub>13</sub> ...A <sub>0</sub> 0 0
1	0 1 1 1 1	A 0 0...0	0 A <sub>14</sub> ...A <sub>0</sub> 0
1	1 X X X X	0...0	A <sub>15</sub> ...A <sub>1</sub> A <sub>0</sub>

49-000138C

3g

**SHRBRV [Shift Right with Bit Reverse], SHLBRV [Shift Left with Bit Reverse]:** SHRBRV or SHLBRV first reverses the order of the bits in DATA<sub>A</sub> and then performs a normal SHR or SHL operation, respectively. See figure 21.

**Compare Instructions**

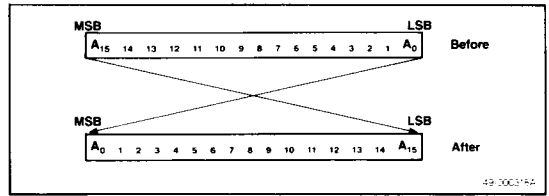
The Compare instructions (see table 16) are different from other PU instructions in that PNZ conditions must be specified along with the instructions. When a compare instruction is used along with a specified PNZ field, the Processing Unit performs a subtract operation. This subtract operation produces a set of PNZ flags, which are compared against the PNZ field specified by the instruction. When these two PNZ fields coincide, the specified PNZ conditions are said to be true. When they do not coincide, the specified PNZ conditions are said to be false (see table 17). The output data from the Processing Unit differs significantly depending on the PNZ conditions. The following three instructions compare the 17-bit data (S<sub>A</sub> and DATA<sub>A</sub>) from the A side against the 17-bit data (S<sub>B</sub> and DATA<sub>B</sub>) from the B side.

**CMPNOM [Compare and normalize]:** If the specified PNZ conditions are false, then the control bits, sign bits and data for both the X and Y sides are set to zero. If the PNZ conditions are true, then C<sub>X</sub> and C<sub>Y</sub> are set to one, S<sub>X</sub> and S<sub>Y</sub> are set to zero, DATA<sub>X</sub> is set to 0001H, and DATA<sub>Y</sub> is set to 0000H.

**CMP [Compare]:** This instruction outputs the 17-bit data words from the A and B sides to the X and Y sides without any change in their contents. It only alters the control bits. If the specified PNZ conditions are true, then C<sub>X</sub> and C<sub>Y</sub> are set to one. If the PNZ conditions are false, then C<sub>X</sub> is set to one and C<sub>Y</sub> is set to zero.

**CMPXCH [Compare and exchange]:** If the specified PNZ conditions are true, then both the input data from the A side and B side are unchanged and output to the X side and Y side, respectively, including their sign bits and the control bits. However, if the PNZ conditions are false, then the input data from the A side is exchanged with the input data from the B side, including the control and sign bits.

**Figure 21. Bit Reversal Operations in SHRBRV and SHLBRV**



**Table 17. PNZ Field Conditions for Compare Instructions**

PNZ	Condition	True/False	Function	Mnemonic
0 0 1	S <sub>A</sub> DATA <sub>A</sub> = S <sub>B</sub> DATA <sub>B</sub>	True	Equal	EQ
	S <sub>A</sub> DATA <sub>A</sub> ≠ S <sub>B</sub> DATA <sub>B</sub>	False	Not equal	
0 1 0	S <sub>A</sub> DATA <sub>A</sub> < S <sub>B</sub> DATA <sub>B</sub>	True	Less than	LT
	S <sub>A</sub> DATA <sub>A</sub> ≥ S <sub>B</sub> DATA <sub>B</sub>	False	Greater or equal	
0 1 1	S <sub>A</sub> DATA <sub>A</sub> ≤ S <sub>B</sub> DATA <sub>B</sub>	True	Less or equal	LE
	S <sub>A</sub> DATA <sub>A</sub> > S <sub>B</sub> DATA <sub>B</sub>	False	Greater than	
1 0 0	S <sub>A</sub> DATA <sub>A</sub> > S <sub>B</sub> DATA <sub>B</sub>	True	Greater than	GT
	S <sub>A</sub> DATA <sub>A</sub> ≤ S <sub>B</sub> DATA <sub>B</sub>	False	Less or equal	
1 0 1	S <sub>A</sub> DATA <sub>A</sub> ≥ S <sub>B</sub> DATA <sub>B</sub>	True	Greater or equal	GE
	S <sub>A</sub> DATA <sub>A</sub> < S <sub>B</sub> DATA <sub>B</sub>	False	Less than	
1 1 0	S <sub>A</sub> DATA <sub>A</sub> ≠ S <sub>B</sub> DATA <sub>B</sub>	True	Not equal	NE
	S <sub>A</sub> DATA <sub>A</sub> = S <sub>B</sub> DATA <sub>B</sub>	False	Equal	

**Note:** The significance of the PNZ bits when Compare instructions are executed differs from that of other instructions. Here, the use of PNZ = 111 or 000 is prohibited.

**Table 16. Compare Instructions**

Mnemonic	Input						Output						Notes
	C <sub>A</sub>	S <sub>A</sub>	DATA <sub>A</sub>	C <sub>B</sub>	S <sub>B</sub>	DATA <sub>B</sub>	C <sub>X</sub>	S <sub>X</sub>	DATA <sub>X</sub>	C <sub>Y</sub>	S <sub>Y</sub>	DATA <sub>Y</sub>	
CMPNOM	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	0	0	0000H	0	0	0000H	When PNZ is False
	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	1	0	0001H	1	0	0000H	When PNZ is true
CMP	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	0	S <sub>A</sub>	A	0	S <sub>B</sub>	B	When PNZ is false
	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	1	S <sub>A</sub>	A	1	S <sub>B</sub>	B	When PNZ is true
CMPXCH	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	When PNZ is true
	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>B</sub>	S <sub>B</sub>	A	C <sub>A</sub>	S <sub>B</sub>	A	When PNZ is false

## Bit Manipulation Instructions

**GET1 [Get one bit]:** This instruction is used to read a particular bit from DATA<sub>A</sub> (see table 18). A bit of DATA<sub>A</sub> specified by the lower 4 bits of DATA<sub>B</sub> is output as the least significant bit of DATA<sub>X</sub>. All other bits of DATA<sub>X</sub> are set to zero. DATA<sub>Y</sub> is also set to zero. The control bits and the sign bits of DATA<sub>X</sub> and DATA<sub>Y</sub> are as follows: C<sub>X</sub> ← C<sub>A</sub>, C<sub>Y</sub> ← C<sub>B</sub>, S<sub>X</sub> ← S<sub>A</sub>, S<sub>Y</sub> ← 0.

**SET1 [Set one bit]:** This instruction is used to set a particular bit of DATA<sub>A</sub>. The bit of DATA<sub>A</sub> to be set is specified by the lower 4 bits of DATA<sub>B</sub>. After the bit is set, the 16-bit result is output as DATA<sub>X</sub>. DATA<sub>Y</sub> is always output as zero. The control bits and the sign bits of DATA<sub>X</sub> and DATA<sub>Y</sub> are as follows: C<sub>X</sub> ← C<sub>A</sub>, C<sub>Y</sub> ← C<sub>B</sub>, S<sub>X</sub> ← S<sub>A</sub>, S<sub>Y</sub> ← 0.

**CLR1 [Clear one bit]:** This instruction is used to reset a particular bit of DATA<sub>A</sub>. The bit of DATA<sub>A</sub> to be reset is specified by the lower 4 bits of DATA<sub>B</sub>. After the bit is reset (cleared), the 16-bit result is output as DATA<sub>X</sub>. DATA<sub>Y</sub> is always output as zero. The control bits and the sign bits of DATA<sub>X</sub> and DATA<sub>Y</sub> are as follows: C<sub>X</sub> ← C<sub>A</sub>, C<sub>Y</sub> ← C<sub>B</sub>, S<sub>X</sub> ← S<sub>A</sub>, S<sub>Y</sub> ← 0.

**Table 18. Bit Addressing**

DATA <sub>B</sub> Bit				DATA <sub>A</sub> Bit Position
3	2	1	0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

## Bit Check Instructions

**ANDMSK [Mask a word with logical AND]:** This instruction tests certain bits in DATA<sub>A</sub>. The bits in DATA<sub>A</sub> to be tested are first masked with a bit pattern in DATA<sub>B</sub>. Only those bits in DATA<sub>A</sub> corresponding to the one bits of DATA<sub>B</sub> are considered. Then only those masked bits

of DATA<sub>A</sub> are ANDed together to set or reset the control bits, C<sub>X</sub> and C<sub>Y</sub>. If the result of the AND operation is 1, then both the C<sub>X</sub> and C<sub>Y</sub> are set to 1. If the result of the operation is 0, then the both C<sub>X</sub> and C<sub>Y</sub> are set to 0. The rest of the output data fields are the following: S<sub>X</sub> ← S<sub>A</sub>, S<sub>Y</sub> ← S<sub>B</sub>, DATA<sub>X</sub> ← DATA<sub>A</sub>, DATA<sub>Y</sub> ← DATA<sub>B</sub>.

**ORMSK [Mask a word with logical OR]:** This instruction tests certain bits in DATA<sub>A</sub>. The bits in DATA<sub>A</sub> to be tested are first masked with a bit pattern in DATA<sub>B</sub>. Only those bits in DATA<sub>A</sub> corresponding to the one bits of DATA<sub>B</sub> are considered. Then only those masked bits of DATA<sub>A</sub> are ORed together to set or reset the control bits, C<sub>X</sub> and C<sub>Y</sub>. If the result of the OR operation is 1, then both C<sub>X</sub> and C<sub>Y</sub> are set to 1. If the result of the operation is 0, then the both C<sub>X</sub> and C<sub>Y</sub> are set to 0. The rest of the output data fields are the following: S<sub>X</sub> ← S<sub>A</sub>, S<sub>Y</sub> ← S<sub>B</sub>, DATA<sub>X</sub> ← DATA<sub>A</sub>, DATA<sub>Y</sub> ← DATA<sub>B</sub>.

## Data Conversion Instructions

**CVT2AB [Convert two's complement to sign-magnitude]:** This instruction converts a 16-bit number in two's complement form to a 17-bit number in sign-magnitude form. The sign of the two's complement number is output as the S<sub>X</sub> bit.

**CVTAB2 [Convert sign-magnitude to two's complement]:** This instruction converts a 17-bit number in sign-magnitude form to a 16-bit number in two's complement form. This operation has the potential danger of an overflow or an underflow. If an overflow or an underflow occurs, the C<sub>X</sub> bit is set to 1.

## Double Precision Adjustment Instruction

**ADJL [Adjust long]:** This instruction is used to adjust a double precision number, in which the sign bits of the upper and lower words are different. This situation may occur after a double precision arithmetic operation. The examples in table 19 illustrate the adjustments of double precision numbers.

**Table 19. Double Precision Adjustment Examples**

	Input/Output	Sign	Data
Input	High (A data)	0	1234H
	Low (B data)	0	5678H
Output	High (X data)	0	1234H
	Low (Y data)	0	5678H
Input	High (A data)	0	1234H
	Low (B data)	1	5678H
Output	High (X data)	0	1233H
	Low (Y data)	0	A988H
Input	High (A data)	1	1234H
	Low (B data)	0	5678H
Output	High (X data)	1	1233H
	Low (Y data)	1	A988H

3g

**Accumulative Addition Instruction**

**ACC [Accumulate]:** This instruction (see figure 22) performs cumulative additions of incoming tokens' data fields. The incoming tokens are classified into type 1 and type 2 tokens. A type 1 token is deleted after the ACC operation, but a type 2 token is not. Moreover, a type 2 token reads the contents of the ACC register, which contains the accumulated sum of tokens. When a type 2 token reads the contents of the ACC register, the ID field of the token is unchanged. However, if an overflow has occurred prior to the arrival of a type 2 token, the ID field is incremented by one. Only the following three tokens qualify as type 2 tokens.

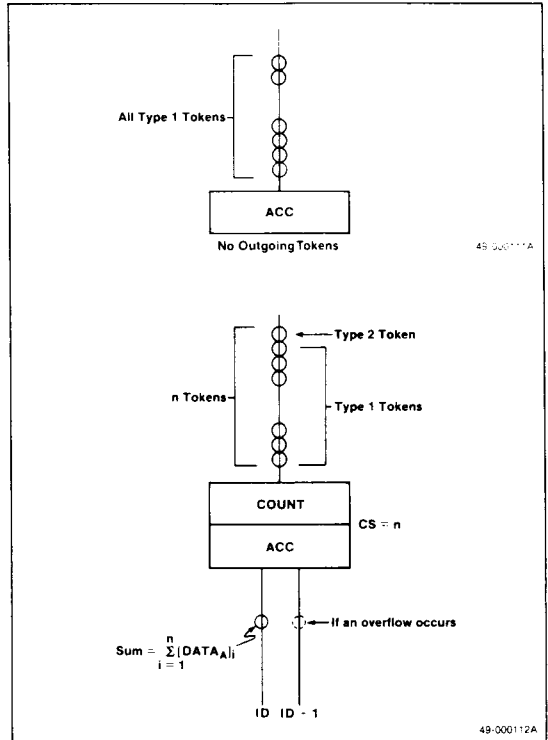
1. If the ACC instruction is used along with RDCYCS instruction, and the token's FTRC bit = 1, and the Buffer Size and Read Counter of RDCYCS instruction are equal.
2. If the ACC instruction is used along with RDCYCL instruction, and the token's FTRC bit = 1, and the Buffer size and Read Counter of RDCYCL instruction are equal.
3. If the ACC instruction is used along with COUNT instruction, and the token's FTRC bit = 0, and the Count Size and Counter of COUNT instruction are equal.

**C Bit Copy Instruction**

**COPYC [Copy control bit]:** This instruction copies the control bit of the A side and outputs it as C<sub>Y</sub>.

$$C_X \leftarrow C_A, S_X \leftarrow S_A, DATA_X \leftarrow DATA_A, C_Y \leftarrow C_A, S_Y \leftarrow S_B, DATA_Y \leftarrow DATA_B.$$

Figure 22. ACC Instruction





**Table 20. PU Instruction (Sheet 1 of 3)**

Mnemonic	OP Code	Input						Output						Notes
		C <sub>A</sub>	S <sub>A</sub>	DATA <sub>A</sub>	C <sub>B</sub>	S <sub>B</sub>	DATA <sub>B</sub>	C <sub>X</sub>	S <sub>X</sub>	DATA <sub>X</sub>	C <sub>Y</sub>	S <sub>Y</sub>	DATA <sub>Y</sub>	
<b>Logical Operations</b>														
OR	00000	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>A</sub>	A OR B	C <sub>Y</sub>	0	0000H	
AND	00001	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>A</sub>	A AND B	C <sub>Y</sub>	0	0000H	
XOR	00010	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>A</sub>	A XOR B	C <sub>Y</sub>	0	0000H	
ANDNOT	00011	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>A</sub>	$\bar{A}$ AND B	C <sub>Y</sub>	0	0000H	
NOT	01100	C <sub>A</sub>	S <sub>A</sub>	A				C <sub>X</sub>	S <sub>A</sub>	$\bar{A}$	C <sub>Y</sub>	0	0000H	
<b>Arithmetic Operations</b>														
ADD	11000	C <sub>A</sub>	0	A	C <sub>B</sub>	0	B	C <sub>X</sub>	0	A + B	C <sub>Y</sub>	0	*	
		C <sub>A</sub>	0	A	C <sub>B</sub>	1	B	C <sub>X</sub>	0	A - B	C <sub>Y</sub>	0	0000H	When A ≥ B, S <sub>X</sub> = 0
		C <sub>A</sub>	1	A	C <sub>B</sub>	0	B	C <sub>X</sub>	1	B - A	C <sub>Y</sub>	1	0000H	When A < B, S <sub>X</sub> = 1
		C <sub>A</sub>	1	A	C <sub>B</sub>	1	B	C <sub>X</sub>	1	A - B	C <sub>Y</sub>	1	0000H	When A ≥ B, S <sub>X</sub> = 1
ADDSC	11100	C <sub>A</sub>	0	A	C <sub>B</sub>	0	B	C <sub>X</sub>	0	A + B	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	
		C <sub>A</sub>	0	A	C <sub>B</sub>	1	B	C <sub>X</sub>	0	A - B	C <sub>Y</sub>	S <sub>S</sub>	*	When A ≥ B, S <sub>X</sub> = 0
		C <sub>A</sub>	0	A	C <sub>B</sub>	1	B	C <sub>X</sub>	1	B - A	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	When A < B, S <sub>X</sub> = 1
		C <sub>A</sub>	1	A	C <sub>B</sub>	0	B	C <sub>X</sub>	0	B - A	C <sub>Y</sub>	S <sub>S</sub>	*	When A < B, S <sub>X</sub> = 0
		C <sub>A</sub>	1	A	C <sub>B</sub>	0	B	C <sub>X</sub>	1	A - B	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	When A ≥ B, S <sub>X</sub> = 1
		C <sub>A</sub>	1	A	C <sub>B</sub>	1	B	C <sub>X</sub>	1	A + B	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	
SUB	11001	C <sub>A</sub>	0	A	C <sub>B</sub>	0	B	C <sub>X</sub>	0	A - B	C <sub>Y</sub>	0	0000H	When A > B, S <sub>X</sub> = 0
		C <sub>A</sub>	0	A	C <sub>B</sub>	1	B	C <sub>X</sub>	1	B - A	C <sub>Y</sub>	1	0000H	When A < B, S <sub>X</sub> = 1
		C <sub>A</sub>	1	A	C <sub>B</sub>	0	B	C <sub>X</sub>	1	A + B	C <sub>Y</sub>	1	*	
		C <sub>A</sub>	1	A	C <sub>B</sub>	1	B	C <sub>X</sub>	0	B - A	C <sub>Y</sub>	0	0000H	When A < B, S <sub>X</sub> = 0
		C <sub>A</sub>	1	A	C <sub>B</sub>	1	B	C <sub>X</sub>	1	A - B	C <sub>Y</sub>	1	0000H	When A ≥ B, S <sub>X</sub> = 1
SUBSC	11101	C <sub>A</sub>	0	A	C <sub>B</sub>	0	B	C <sub>X</sub>	0	A - B	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	When A ≥ B, S <sub>X</sub> = 0
		C <sub>A</sub>	0	A	C <sub>B</sub>	1	B	C <sub>X</sub>	1	B - A	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	When A < B, S <sub>X</sub> = 1
		C <sub>A</sub>	0	A	C <sub>B</sub>	1	B	C <sub>X</sub>	0	A + B	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	
		C <sub>A</sub>	1	A	C <sub>B</sub>	0	B	C <sub>X</sub>	1	A + B	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	
		C <sub>A</sub>	1	A	C <sub>B</sub>	1	B	C <sub>X</sub>	0	B - A	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	When A < B, S <sub>X</sub> = 0
		C <sub>A</sub>	1	A	C <sub>B</sub>	1	B	C <sub>X</sub>	1	A - B	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	When A ≥ B, S <sub>X</sub> = 1
MUL	11010	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>X</sub>	A x B High	C <sub>Y</sub>	S <sub>X</sub>	A x B Low	S <sub>X</sub> = S <sub>A</sub> OR S <sub>B</sub> (logical OR)
MULSC	11110	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>X</sub>	A x B High	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	S <sub>X</sub> = S <sub>A</sub> OR S <sub>B</sub> (logical OR)

3g

Table 20. PU Instruction (Sheet 2 of 3)

Mnemonic	OP code	Input						Output						Notes
		C <sub>A</sub>	S <sub>A</sub>	DATA <sub>A</sub>	C <sub>B</sub>	S <sub>B</sub>	DATA <sub>B</sub>	C <sub>X</sub>	S <sub>X</sub>	DATA <sub>X</sub>	C <sub>Y</sub>	S <sub>Y</sub>	DATA <sub>Y</sub>	
<b>Arithmetic Operations</b>														
NOP	11011	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>A</sub>	A	C <sub>Y</sub>	S <sub>B</sub>	B	
NOPSC	11111	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>A</sub>	A	C <sub>Y</sub>	S <sub>S</sub>	No. of shifts +	
INC	01010	C <sub>A</sub>	0	A				C <sub>X</sub>	0	A + 1	C <sub>Y</sub>	0	*	
		C <sub>A</sub>	1	A				C <sub>X</sub>	0	1	C <sub>Y</sub>	0	0000H	When A = 0, S <sub>X</sub> = 0
		C <sub>A</sub>						C <sub>X</sub>	1	A - 1	C <sub>Y</sub>	1	0000H	When A ≥ 1, S <sub>X</sub> = 1
DEC	01011	C <sub>A</sub>	0	A				C <sub>X</sub>	0	A - 1	C <sub>Y</sub>	0	0000H	When A ≥ 0, S <sub>X</sub> = 0
		C <sub>A</sub>						C <sub>X</sub>	1	1	C <sub>Y</sub>	1	0000H	When A = 0, S <sub>X</sub> = 1
		C <sub>A</sub>	1	A				C <sub>X</sub>	1	A + 1	C <sub>Y</sub>	1	*	
<b>Shift</b>														
SHL	00100	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	0	No. of shifts	C <sub>X</sub>	S <sub>A</sub>	Shift A left	C <sub>Y</sub>	S <sub>A</sub>	Shift A left	
		C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	1	No. of shifts	C <sub>X</sub>	S <sub>A</sub>	Shift A right	C <sub>Y</sub>	S <sub>A</sub>	Shift A right	
SHLBRV	00101	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	0	No. of shifts	C <sub>X</sub>	S <sub>A</sub>	Reverse A and shift left	C <sub>Y</sub>	S <sub>A</sub>	Reverse A and shift left	
		C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	1	No. of shifts	C <sub>X</sub>	S <sub>A</sub>	Reverse A and shift right	C <sub>Y</sub>	S <sub>A</sub>	Reverse A and shift right	
SHR	00110	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	0	No. of shifts	C <sub>X</sub>	S <sub>A</sub>	Shift A right	C <sub>Y</sub>	S <sub>A</sub>	Shift A right	
		C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	1	No. of shifts	C <sub>X</sub>	S <sub>A</sub>	Shift A left	C <sub>Y</sub>	S <sub>A</sub>	Shift A left	
SHRBRV	00111	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	0	No. of shifts	C <sub>X</sub>	S <sub>A</sub>	Reverse A and shift right	C <sub>Y</sub>	S <sub>A</sub>	Reverse A and shift right	
		C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	1	No. of shifts	C <sub>X</sub>	S <sub>A</sub>	Reverse A and shift left	C <sub>Y</sub>	S <sub>A</sub>	Reverse A and shift left	
<b>Comparison</b>														
CMPNOM	01000	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	0	0	0000H	0	0	0000H	When PNZ is false
		C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	1	0	0001H	1	0	0000H	When PNZ is true
CMP	01001	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	0	S <sub>A</sub>	A	0	S <sub>B</sub>	B	When PNZ is false
		C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	1	S <sub>A</sub>	A	1	S <sub>B</sub>	B	When PNZ is true
CMPXCH	10001	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	When PNZ is true
		C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>A</sub>	S <sub>A</sub>	A	When PNZ is false
<b>Accumulative Addition</b>														
ACC	10010	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>X</sub>	ΣA				Used as a pair with AG & FC instruction COUNT
<b>C Bit Copy</b>														
COPYC	10011	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>A</sub>	S <sub>B</sub>	B	

**Table 20. PU Instruction (Sheet 3 of 3)**

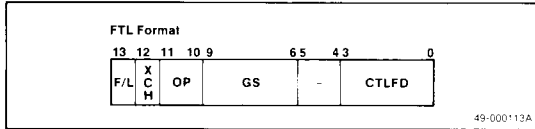
Mnemonic	OP code	Input						Output						Notes
		C <sub>A</sub>	S <sub>A</sub>	DATA <sub>A</sub>	C <sub>B</sub>	S <sub>B</sub>	DATA <sub>B</sub>	C <sub>X</sub>	S <sub>X</sub>	DATA <sub>X</sub>	C <sub>Y</sub>	S <sub>Y</sub>	DATA <sub>Y</sub>	
<b>Bit Operations</b>														
GET1	1 0 1 0 1	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	Bit position	C <sub>X</sub>	S <sub>A</sub>	0000H	C <sub>Y</sub>	0	0000H	When the bit specified by the lower 4 bits of DATA <sub>B</sub> is 0
		C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	Bit position	C <sub>X</sub>	S <sub>A</sub>	0001H	C <sub>Y</sub>	0	0000H	
SET1	1 0 1 1 0	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	Bit position	C <sub>X</sub>	S <sub>A</sub>	A bit in DATA <sub>A</sub> is set	C <sub>Y</sub>	0	0000H	Bit specification by the lower 4 bits of DATA <sub>B</sub>
CLR1	1 0 1 1 1	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	Bit position	C <sub>X</sub>	S <sub>A</sub>	A bit in DATA <sub>A</sub> is cleared	C <sub>Y</sub>	0	0000H	Bit specification by the lower 4 bits of DATA <sub>B</sub>
<b>Bit Check</b>														
ANDMSK	0 1 1 0 1	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	0	S <sub>A</sub>	A	0	S <sub>B</sub>	B	If ANDMSK = 0
		C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	1	S <sub>A</sub>	A	1	S <sub>B</sub>	B	If ANDMSK = 1
ORMSK	1 0 0 0 0	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	0	S <sub>A</sub>	A	0	S <sub>B</sub>	B	If ORMSK = 0
		C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	1	S <sub>A</sub>	A	1	S <sub>B</sub>	B	If ORMSK = 1
<b>Data Conversion</b>														
CVT2AB	0 1 1 1 0	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>X</sub>	Converted A data	C <sub>Y</sub>	0	0000H	Absolute value -- twos complement
CVTAB2	0 1 1 1 1	C <sub>A</sub>	S <sub>A</sub>	A	C <sub>B</sub>	S <sub>B</sub>	B	C <sub>X</sub>	S <sub>X</sub>	Converted A data	C <sub>Y</sub>	0	0000H	Twos complement -- absolute value
<b>Adjustment of Double Precision Numbers</b>														
ADJL	1 0 1 0 0	C <sub>A</sub>	0	A	C <sub>B</sub>	1	B	C <sub>X</sub>	0	A - 1	C <sub>Y</sub>	0	0000H-B	A ≠ 0 AND B ≠ 0
		C <sub>A</sub>	1	A	C <sub>B</sub>	0	B	C <sub>X</sub>	1	A - 1	C <sub>Y</sub>	1	0000H-B	A ≠ 0 AND B ≠ 0
		C <sub>A</sub>	0	A	C <sub>B</sub>	1	0000H	C <sub>X</sub>	0	A	C <sub>Y</sub>	0	0000H	
		C <sub>A</sub>	0	0000H	C <sub>B</sub>	1	B	C <sub>X</sub>	1	0000H	C <sub>Y</sub>	1	B	B ≠ 0
		C <sub>A</sub>	1	A	C <sub>B</sub>	0	0000H	C <sub>X</sub>	1	A	C <sub>Y</sub>	1	0000H	
		C <sub>A</sub>	1	0000H	C <sub>B</sub>	0	B	C <sub>X</sub>	0	0000H	C <sub>Y</sub>	0	B	B ≠ 0
		C <sub>A</sub>	0	A	C <sub>B</sub>	0	B	C <sub>X</sub>	0	A	C <sub>Y</sub>	0	B	
		C <sub>A</sub>	1	A	C <sub>B</sub>	1	B	C <sub>X</sub>	1	A	C <sub>Y</sub>	1	B	

**Notes:** \* If an overflow occurs as the result of A + B, DATA<sub>Y</sub> = 0001H and if no overflow, DATA<sub>Y</sub> = 0000H.

† This indicates the number of consecutive zeros from the MSB of DATA<sub>X</sub>. This number is used to calculate the number of shifts to be performed by subsequent processing.

3g

**GE Instructions**



**Bit Assignments**

**F/L [Full/Left]:** F/L bit = 0 indicates that the GE instruction is used alone, whereas F/L bit = 1 indicates that the GE instruction is used in conjunction with an AG/FC instruction.

**XCH [Exchange]:** XCH bit = 1 indicates that the data from A side and B side are to be exchanged before the two data tokens enter the Queue.

**OP [OP code]:** These two bits select an operation to be performed. See table 21.

**Table 21. OP Bits**

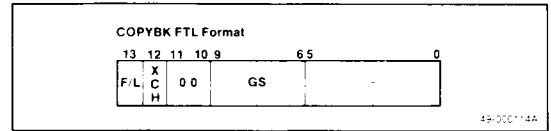
OP	Operation
00	COPYBK (Copy block)
01	COPYM (Copy multiple)
11	SETCTL (Set control field)

**GS [Generation Size]:** These four bits determine the number of copies of a token to be made. A minimum of 2 and a maximum of 17 copies can be made using a GE instruction.

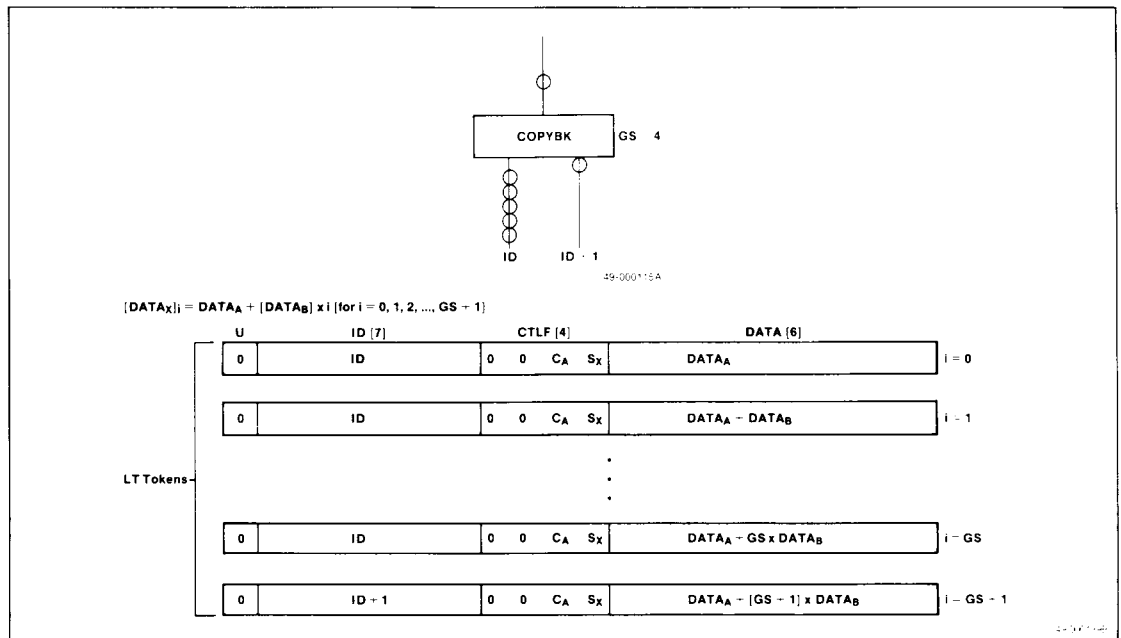
**CTLFD [Control Field]:** This field is used with Set Control Field (SETCTL) instruction. The data in CTLFD field further specifies the types of operations to be performed by the SETCTL instruction.

**COPYBK [Copy Block]**

COPYBK is used to duplicate a block of tokens from a single token. These duplicated tokens have exactly the same ID as the original token except the token copied last which has the original token's ID plus one. The number of tokens to be generated is specified by the GS field, and the COPYBK instruction generates exactly GS + 2 tokens. The data fields of the tokens being duplicated can also be incremented or decremented in a systematic manner. The incremental (or decremental) step value is contained in DATA<sub>B</sub>. The tokens generated are sent to the Link Table. The series of LT tokens output by the instruction is shown in figure 23.



**Figure 23. COPYBK Instruction Output**



## COPYM [Copy Multiple]

COPYM is used to generate multiple tokens from a single token. Each generated token has a different ID value. The number of tokens generated from the original token is  $GS + 2$ . The data field of the tokens being generated can also be incremented or decremented in a systematic manner. The incremental (or decremental) step value is contained in  $DATA_B$ . The

generated tokens are sent to the Link Table as LT tokens. The series of LT tokens output by the COPYM instruction is shown in figure 24.

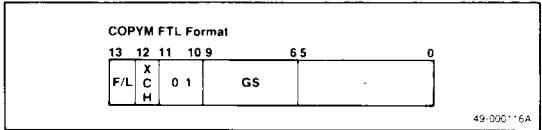
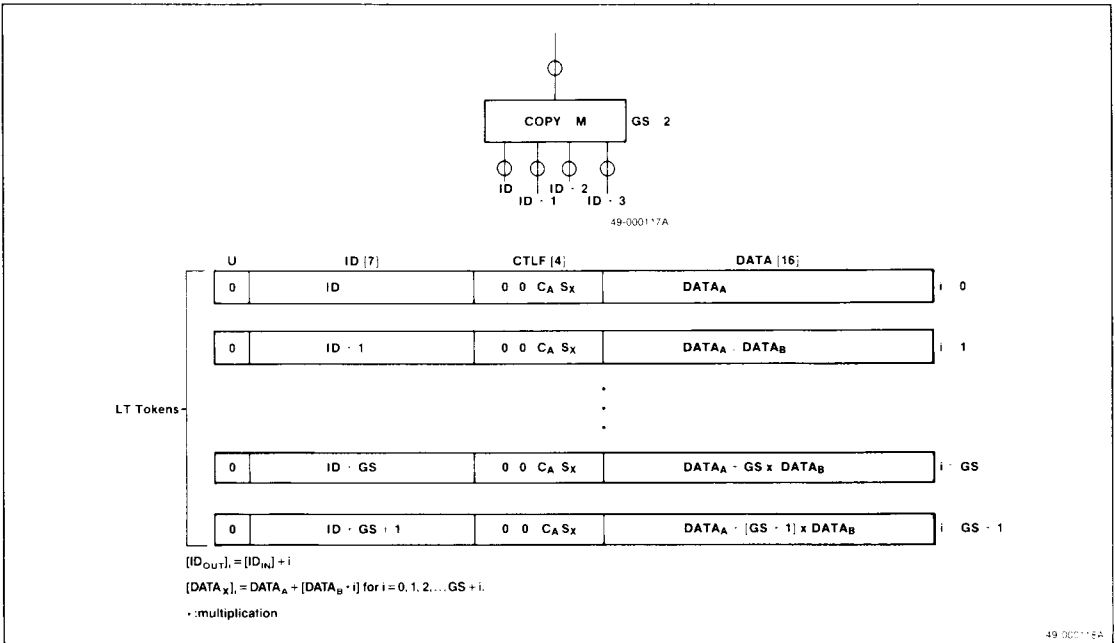
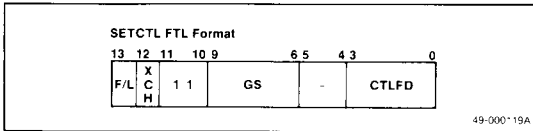


Figure 24. COPYM Instruction Output Tokens



3g

**SETCTL [Set Control Field]**



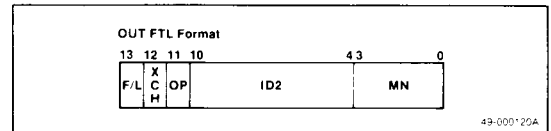
SETCTL is used to read and rewrite the contents of the Link Table and the Function Table. Since it can change the contents of the Link Table and the Function Table, this instruction can be used to write a self-modifying code. The type of operation to be performed is further specified by the contents of CTLFD field, as shown in table 22.

**Table 22. SETCTL Instruction Control Field Operation**

CTLFD	Operation
0 0 C S	Normal data. Operation is exactly the same as COPYM.
1 1 0 0	The data field of this token is used to set a location in the Link Table memory (C and S bits are not included.) After the data is set, the token is deleted.
1 1 0 1	The data field of this token is used to set a location in the Function Table Right field. After the data is set, the token is deleted.
1 1 1 0	The lower 14 bits of the data field of this token are used to set a location in the Function Table Left field (higher bits are ignored.) After the data is set, the token is deleted.
1 1 1 1	The lower 10 bits of the data field of this token are used to set a location in the Function Table Temporary field (higher bits are ignored.) After the data is set, the token is deleted.
1 0 0 0	This token reads the LT address indicated by the ID field and outputs the contents.
1 0 0 1	This token reads the Function Table Right field address indicated by the ID field and outputs the contents.
1 0 1 0	This token reads the Function Table Left field address indicated by the ID field and outputs the contents.
1 0 1 1	This token reads the Function Table Temporary field address indicated by the ID field and outputs the contents.
0 1 0 0	These tokens should not be generated by the Processing
0 1 0 1	Unit. They are operating-mode-related tokens.
0 1 1 0	
0 1 1 1	

**Note:** The set or write operation is performed at the address indicated by the ID field of the token.

**OUT Instructions**



**Bit Assignments**

**F/L [Full/Left]:** F/L bit = 0 indicates that the OUT instruction is to be used alone. F/L bit = 1 indicates that the OUT instruction is to be used in conjunction with an AG/FC instruction.

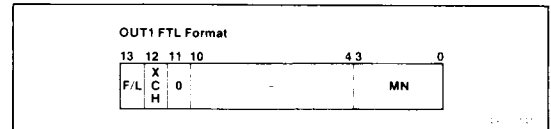
**XCH [Exchange]:** If XCH bit = 1, the output data tokens from the A side are exchanged with those from the B side before they go to the Output Queue. If XCH bit = 0, no exchange operation is performed.

**OP [OP Code]:** This bit is used to further specify the OUT instruction. If OP = 0, then OUT1 instruction is performed, whereas if OP = 1, OUT2 instruction is performed.

**ID2 [Second ID]:** This field is used only by the OUT2 instruction. ID2 is the ID of the second output data token.

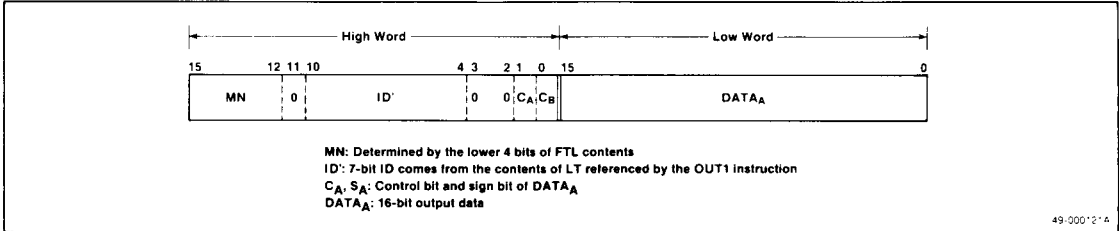
**MN [Module Number]:** This field indicates the destination module of the output data token.

**OUT1**



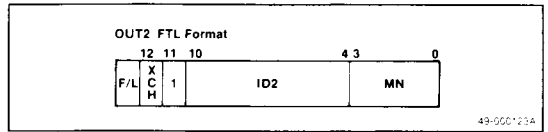
This instruction outputs a 32-bit data token via the Output Data Bus (ODB). Since the size of the ODB is 16 bits, a 32-bit output data token is divided into two 16-bit words and output one 16-bit word at a time. The format of an output data token is shown in figure 24.

**Figure 25. OUT1 Output Token Format**



## OUT2

This instruction outputs two 32-bit data tokens via ODB. Since the ODB is 16 bits wide, each 32-bit token is divided into two 16-bit words and output one 16-bit word at a time. This instruction is useful when a double precision number is to be output. The formats of two output data tokens are shown in figure 25.



3g

**Figure 26. OUT2 Output Tokens Format**

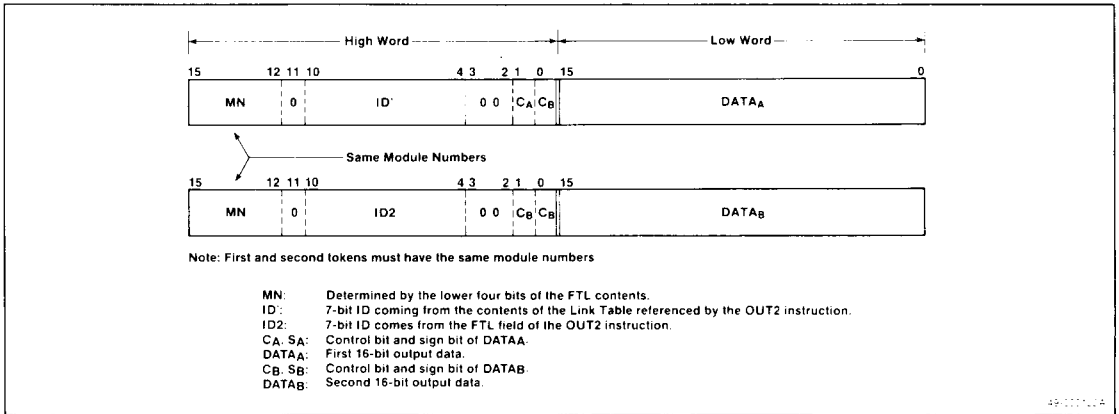


Figure 27. Data-Flow Graph Explanation

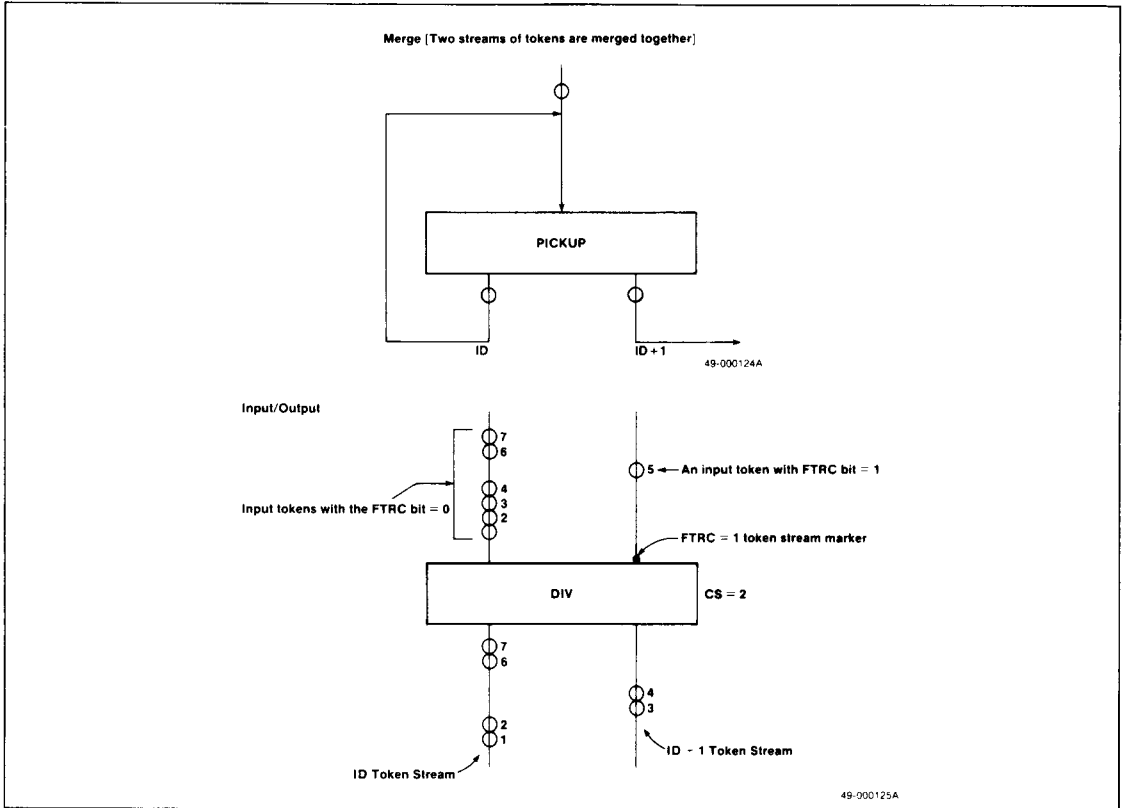




Figure 27. Data-Flow Graph Explanation (cont)

