SIGNETICS 8X300 PROGRAMMING MANUAL

SIGNETICS 8X300 PROGRAMMING MANUAL

Material used in this document was prepared
by A.H.J. Schatorjé of N.V. Philips, Eindhoven,
The Netherlands.

PREFACE

Signetics 8X300 Programming Manual is designed to provide all the information necessary to prepare code for the 8X300 Microcontroller. Details for every variation within each class of instruction are shown diagrammatically, in binary and in assembly language. Sample programs and a description of the Microcontroller Cross Assembly Program (MCCAP) are included.

Additional information relevant to the application of the 8X300 can be found in the following documents:

8X300 Programming Course
8X300 Reference Manual
Signetics Microcontroller Cross Assembly Program

These and other Signetics product documents are available through the offices listed in the back of this manual.

TABLE OF CONTENTS

THE 8X300 SYSTEM

THE 8X300 SYSTEM

The independent instruction and data input/output (I/O) system of the 8X300 is shown in Fig. 1. The 13-bit address bus, capable of addressing 8192 instructions, and the 16-bit instruction bus allow the 8X300 to access the next instruction while simultaneously performing data I/O with the Interface Vector (IV) bus. As can be seen from the diagram, all data to or from external devices or registers passes via the IV bus.

Figure 2 shows the functional diagram of the 8X300, with the data paths between the elements of the microprocessor and the connections to the address, instruction and IV busses. Although the program instruction addressing is essentially independent of the data flow, links exists to allow address modification or the transmission of data from the program to an output device.

Interface Vector Bus

All data input to or output from the 8X300 goes via the IV bus. This IV bus serves both as an address and data bus and is accompanied by control signals to determine its function. Being an 8-bit bus, it has the capability to address up to 256 I/O registers (IV bytes). The input/output facilities are further expanded by selection of Left Bank (LB) or Right Bank (RB) address, giving a total of 512 addressable IV bytes.

When the 8X300 is required to accept data from or send data to a particular IV byte, it must first enable the IV byte. An IV byte is enabled when its address is presented on the IV bus and the bus control signals indicate that the data is an address on the required bank. The IV byte will remain enabled until another IV byte on the same bank is enabled, at which time it becomes disabled.

Because the Left and Right Banks are independent, one IV byte on each Bank can be active (enabled) simultaneously. Data input from, or output to the IV bus implied data I/O to the active byte on the Bank specified by the Instruction causing the I/O action.

The most significant bit of all data is bit 0.

Internal Data Registers

The 8X300 contains an auxiliary (AUX) register and seven work registers to facilitate data manipulation. A separate overflow register is used to provide overflow indication after an ADD instruction. Figure 2 shows these in the schematic diagram of the 8X300.

The AUX register is used as the implied operand in ADD, AND and XOR instructions: however it can also be used as a normal work register for other instructions.

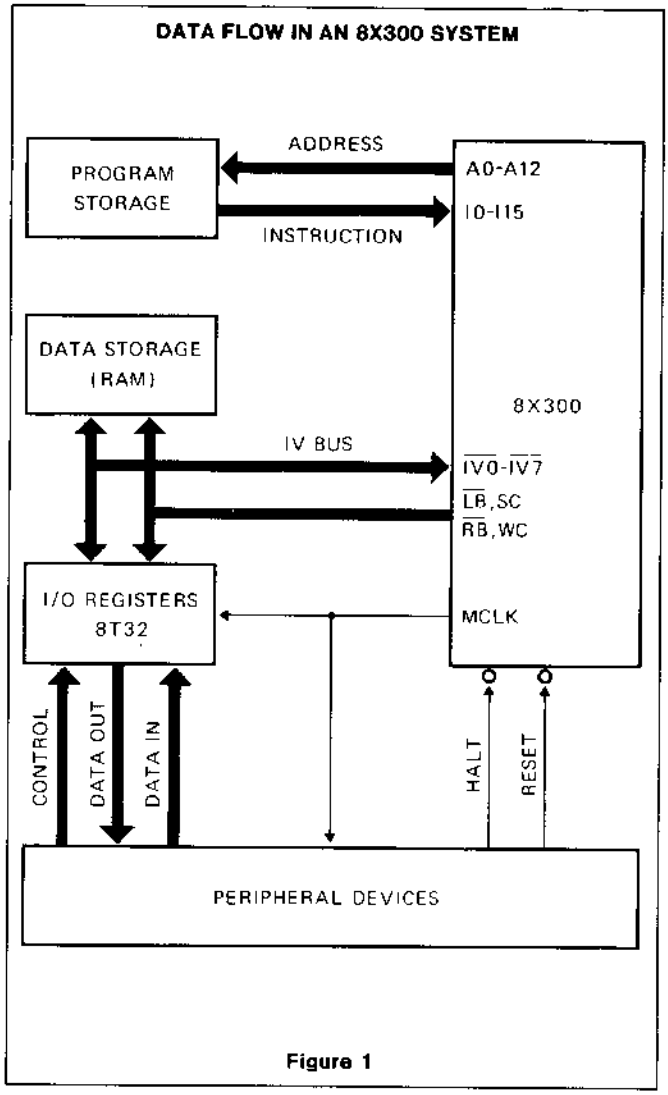


Figure 1

The overflow register can only be used as a source of data. Its seven most significant bits are always zero, while a one in the least significant bit position indicates that overflow occurred during the last ADD instruction. The overflow register contents can only be changed by the result of an ADD instruction.

Table 1 gives details of the data registers of the 8X300 and the corresponding instruction operand values.

Internal Program Registers

There are three registers concerned with instruction execution in the 8X300:

Address Register (AR) — output register holding the address of the current instruction for the program memory;

Program Counter (PC) — holding the address of the current or next instruction to allow modification by the control circuity;

Instruction Register (IR) — Holding the 16-bit instruction word currently being executed.

These registers cannot be addressed as the operand of an instruction although the content of the program counter and address register can be changed as the result of special instructions. The program counter and address register are incremented by one during each instruction cycle to provide the address of the next instruction to be executed. However, a jump instruction can cause this action to be overruled and a new address substituted.

FUNCTIONAL DIAGRAM OF THE 8X300.

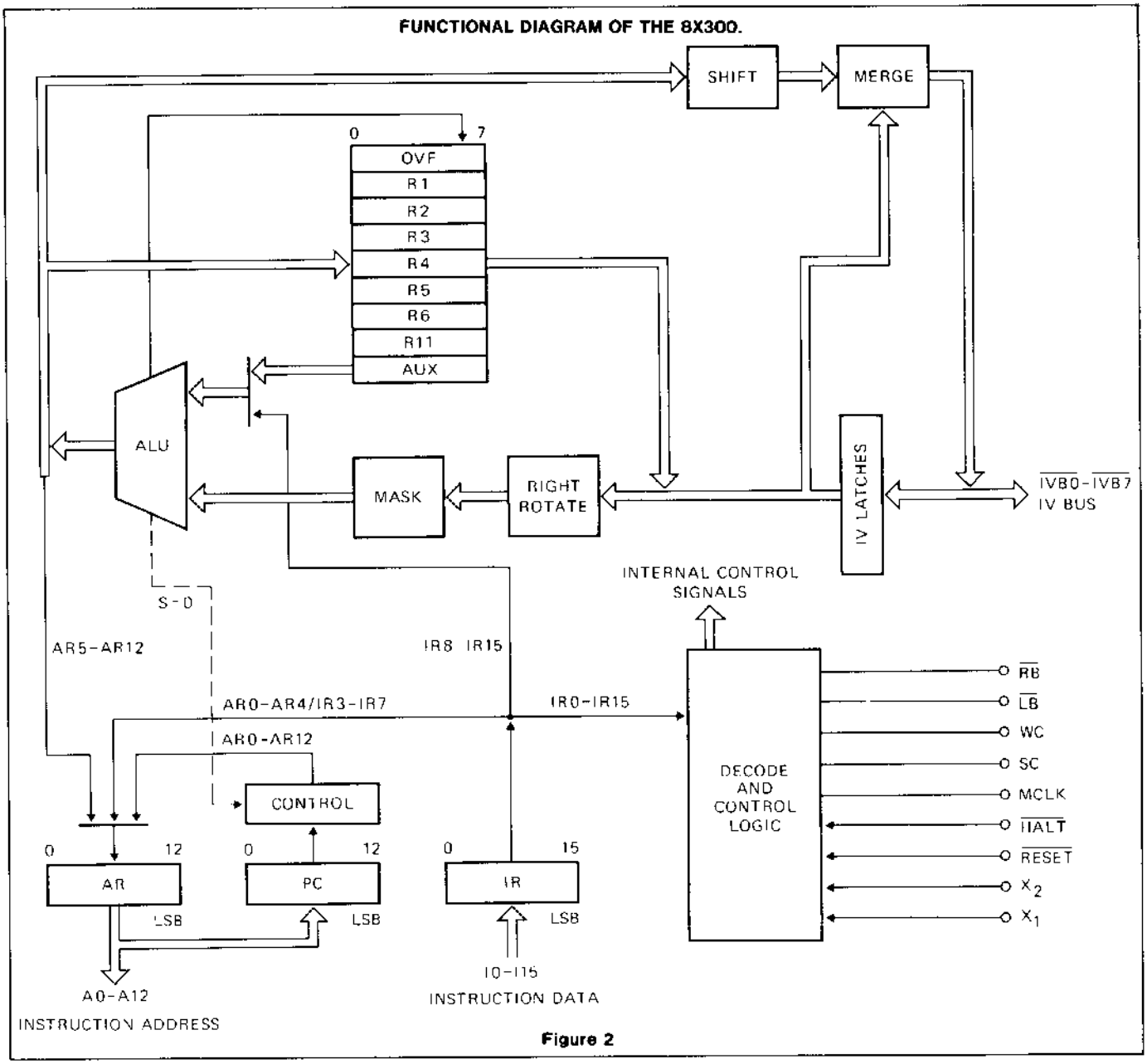


Figure 2

Table 1. INTERNAL DATA REGISTERS OF THE 8X300.

NAME	OCTAL ADDRESS	DESCRIPTION
AUX	00	Work register, containing the implied operand for ADD, AND and XOR instructions.
R1	01	General purpose registers.
R2	02	
R3	03	
R4	04	
R5	05	
R6	06	
R11	11	
OVF	10	Read-only register whose least significant bit indicates overflow status of the last ADD instruction: LSB = 1, overflow occurred. The remaining seven bits are always zero.

The Address Bus

The 8X300 has a separate 13-bit instruction address bus with the capability to address up to 8192 program words.

The Instruction Bus

This is a 16-bit bus which delivers the contents of the selected instruction memory address to the instruction register of the 8X300.

Instruction Formats and Operand Fields

An 8X300 instruction consists of a three bit operation code (OP) followed by a thirteen bit operand field. The operation code determines the class of the instruction to be performed, while the operand field provides details of the data to be processed. Figure 3 shows the general instruction format for the 8X300.

Table 2 shows the various instruction formats and the instructions that use those.

Table 2 INSTRUCTION FORMATS.

Format				Instructions	
0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15		
OP	S	R	D	MOVE ADD AND XOR	register to register register to IV bus address
OP	S	L	D	MOVE ADD AND XOR	register to IV bus IV bus to register IV bus to IV bus IV bus to IV bus address
OP	S	I		XEC NZT	register register
OP	S	L	I	XEC NZT	IV bus IV bus
OP	D	I		XMIT	register IV bus address
OP	D	L	I	XMIT	IV bus
OP	A			JMP	

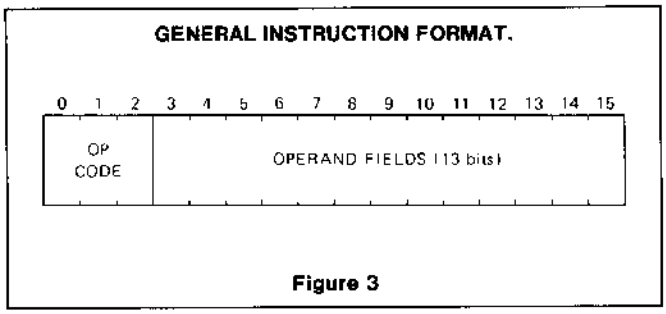


Figure 3

S — Source

This field defines the location of the data byte to be processed. It is a 5-bit field divided into two sub-fields: S_0 (3 bits) and S_1 (2 bits). This allows the address of the source data byte to be specified as two octal digits (maximum is 37). The source can be either a register, in which case both sub-fields are used for the address (see Table 1), or the IV bus. When the source is the IV bus, S_1 specifies the bank (2 = Left bank, 3 = right bank) and S_0 specifies the LSB of the data to be processed: S_0 = n means that the source data byte will be right rotated until bit n is the least significant bit. Thus S_0 = 7 requires no right rotation.

D — Destination

This 5-bit field specifies the destination of the processed data: it can be a register, the IV bus or an IV bus address. Sub-fields D_0 and D_1 allow the destination to be addressed as two octal digits in the same manner as the source field.

When the destination is a register, both sub-fields are used for the address (see Table 1).

When the data is to be used as an IV bus address, the octal values 07 (left bank address) or 17 (right bank address) must be programmed.

When the destination is the IV bus (to the currently enabled IV byte), D_1 specifies the bank (2 = left bank, 3 = right bank) and D_0 specifies the position in the IV byte with which the least significant bit of the processed data field should be aligned.

L — Length

This 3-bit field defines the number of bits in the source and/or destination field.

When the destination is the IV bus, the L field specifies the length of the destination field whose least significant bit is specified by D_0.

When the source is the IV bus, the L field specifies the length of the source field whose least significant bit is specified by S_0. If the destination is also the IV bus then the L field applies to both source and destination.

Note: a value of L = 0 specifies an 8-bit data field.

R — Rotate

For instructions where the source is a register and the destination is either a register or an IV bus address, the 3-bit rotate field is used in place of the length field. A value n means that the source data field is right rotated n-places before being processed.

I — Integer

The integer field is either 5 bits or 8 bits long, depending on the instruction. It is used to provide a constant in the range 0 to 37_8 (5 bits) or 0 to 377_8 (8 bits).

A — Address

The 13-bit address field is used with the jump instruction to define the absolute address to be set into the address register and program counter, i.e. the address of the next instruction to be executed.

Right Rotate and Mask Functions

The combination of right rotate and mask functions allows selection of one or more bits from a source data field. For instructions where both the source and destination are registers, only the rotate function is available, the data being a fixed length of 8 bits.

The right rotate function provides an end-around-shift of one to seven places of the 8-bit source field, see Fig. 4. In this manner, the least significant bit of the bit string required can be positioned in the least significant position of the data byte, ready for further processing, see Fig. 5.

The number of places that the data is to be rotated is specified by the R field, when present, and by the S_0 field when the source is the IV bus.

The R field specifies the number of places the data is to be rotated; the S_0 field specifies the bit of the source data field which will be rotated to bit position seven before masking.

The mask function allows selection of the least significant L bits of the rotated IV bus source data for subsequent processing. The value L is specified by the L-field of the instruction. After masking, the L least significant bits are output to the Arithmetic and Logic Unit (ALU), with the remaining bits of the byte set to zero.

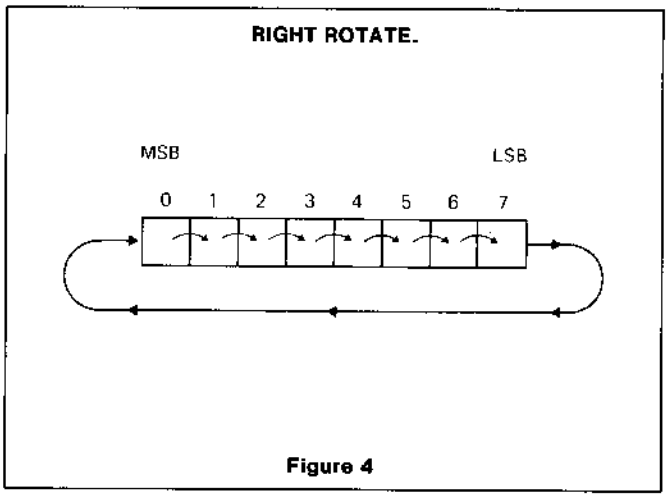


Figure 4

Arithmetic and Logic Unit (ALU)

As its name implies, the ALU performs all the arithmetic and logic functions. For this purpose it has a direct input from the AUX register for the implied operand in ADD, AND and XOR instructions. The output of the ALU may go directly to the address or data registers, or, via the shift and merge circuits, to the IV bus.

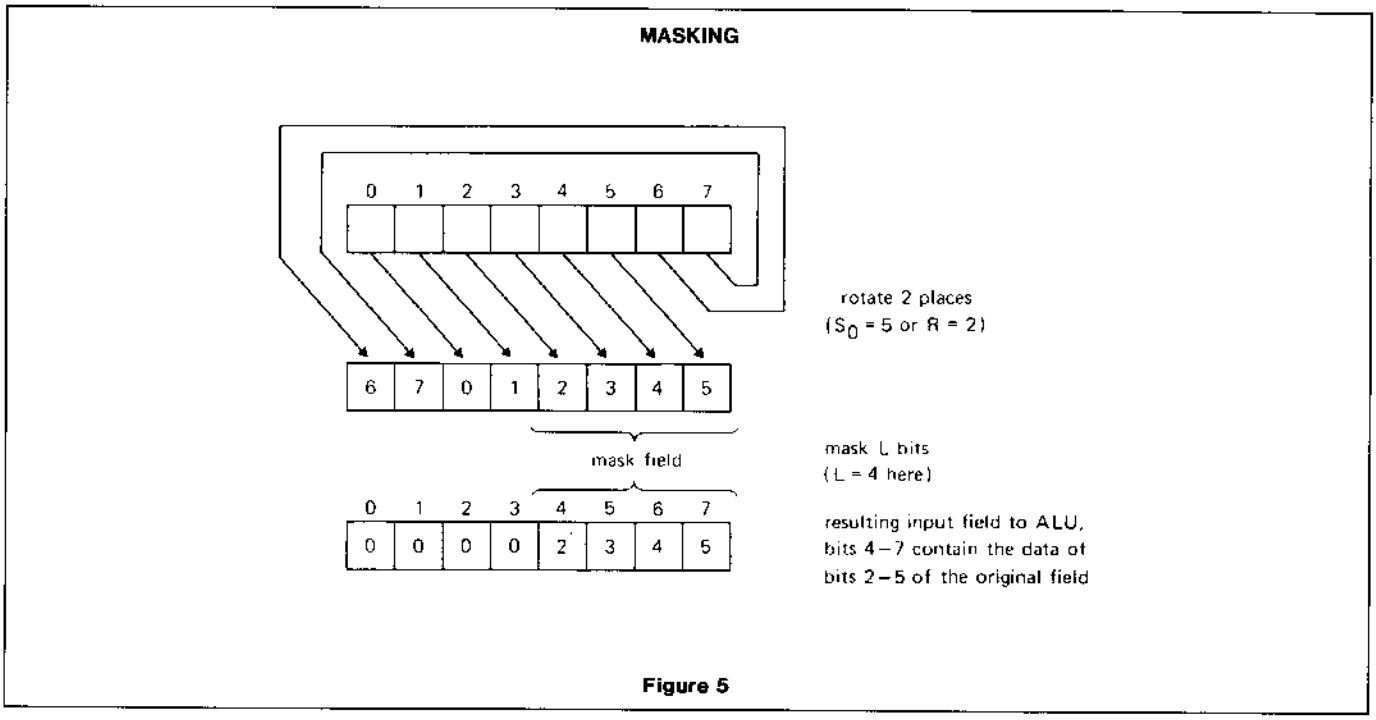


Figure 5

Shift and Merge Functions

The shift and merge functions allow alteration of the state of a bit string within the IV bus data byte. The action of the rotate and mask functions ensures that the required processed data is in the least significant bits of the ALU output; the left shift function then aligns the data in the required bit positions prior to merging, see Fig. 6.

Because the process is not an end-around-shift, data shifted from position 0, the MSB, is lost. The number of positions to be shifted is determined by the value D_0: the data is left shifted until the LSB has reached the bit position specified by D_0.

The merge function allows the user to update part of the existing IV bus data without affecting the remaining parts of the data byte. The length of the bit string to be merged with the existing data is specified by the L-field, the LSB of the bit string being specified by D_0 (after shifting).

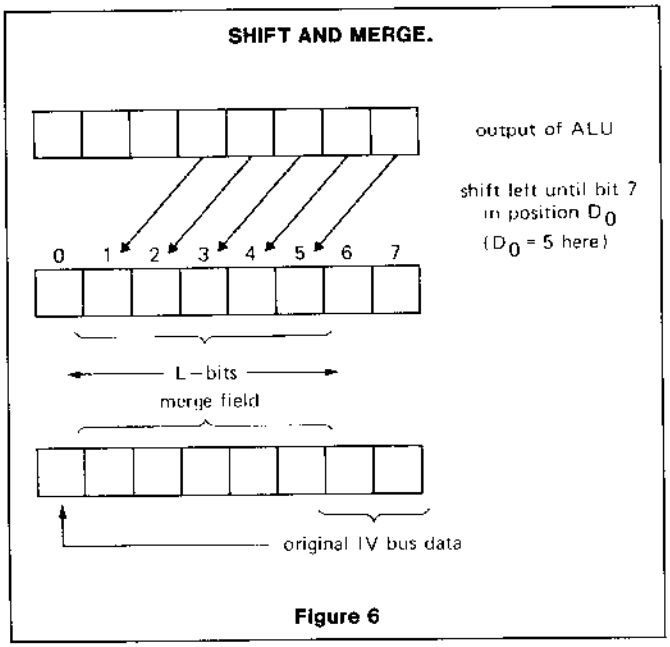


Figure 6

Timing and Instruction Cycle

Each processor operation is executed in one instruction cycle which is internally divided into quarter cycles. During the first quarter cycle the instruction word is accepted by the instruction register and the data input latches are enabled to accept the data on the IV bus. As processing takes place during the second and third quarter cycles, the input data must be stable by the end of the first quarter cycle. The address for the next instruction becomes available during the third quarter cycle enabling access of the program memory during the third and fourth quarter cycles for the ensuing instruction. If data is to be output to the IV bus, output drivers are activiated during the third quarter cycle to present stable output data during the fourth data cycle. Thus, the IV bus works in the input mode during the first two quarter cycles and in the output mode during the last two quarter cycles. Figure 7 shows the breakdown of the instruction cycle time.

During the instruction cycle, the control and decoding logic of the 8X300 selects and activates the required timing and bus control signals in order to execute the current instruction. These signals are shown in Table 3. Figure 8 shows the timing of the control signals during a sequence of three instructions to add the data from an I/O device on the left bank to a running total in storage in a register at the right bank.

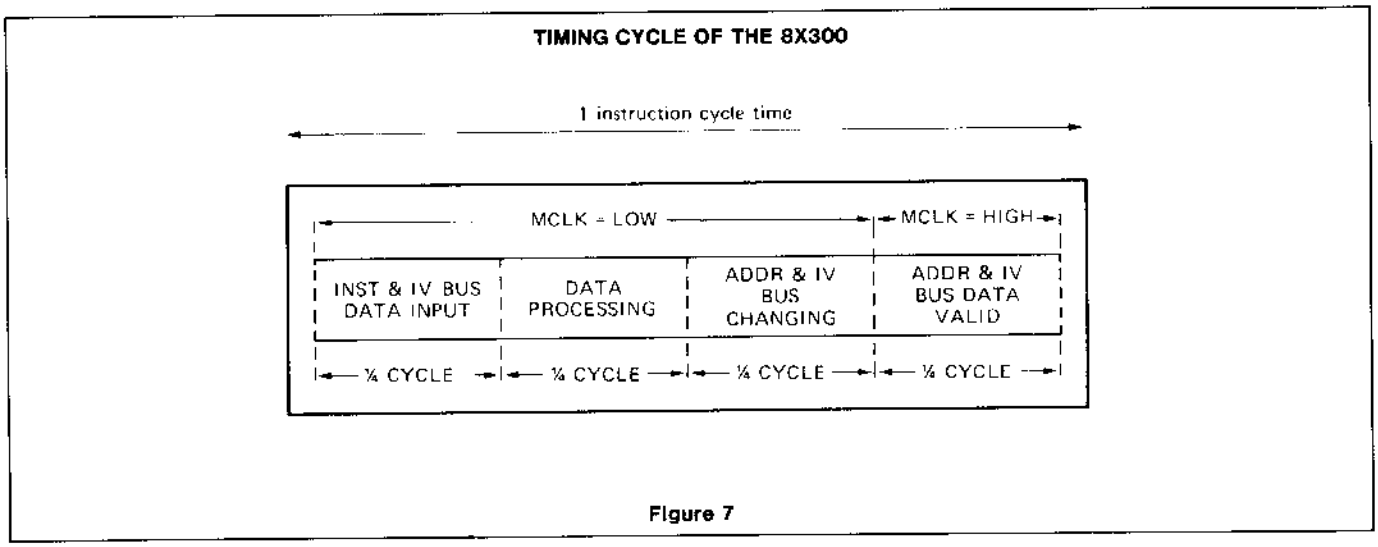
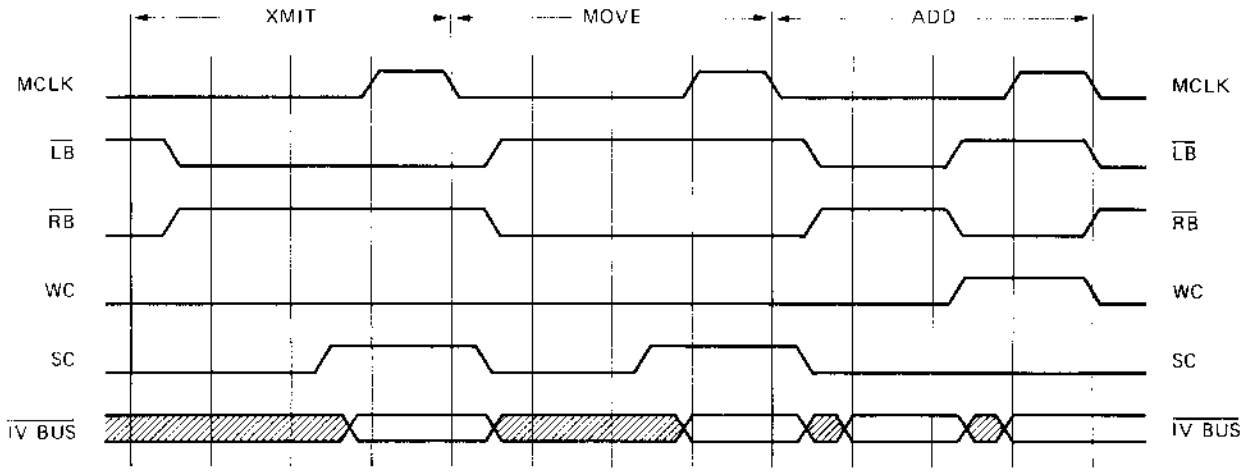


Figure 7

Table 3. I/O TIMING AND CONTROL SIGNALS

SIGNAL	FUNCTION
MCLK	Master clock: used to clock I/O devices or provide synchronization for external logic.
WC	Write Command: HIGH when data is being output to the IV bus.
SC	Select Command: HIGH level indicates that the data output on the IV bus is an address.
$\overline{LB}$	$\overline{Left\ Bank}$: LOW level enables I/O registers on the left bank.
$\overline{RB}$	$\overline{Right\ Bank}$: LOW level enables I/O registers on the right bank.

TIMING OF THE CONTROL SIGNALS DURING A TYPICAL THREE-INSTRUCTION SEQUENCE



750 ns

```
XMIT   5,IVL   SELECT INPUT DEVICE
MOVE   R3,IVR  ADDRESS DATA STORAGE REGISTER
ADD    LB,RB   READ INPUT DATA, ADD TO TOTAL AND STORE
```

Figure 8

THE 8X300 INSTRUCTION SET

THE 8X300 INSTRUCTION SET

The 8X300 instruction set is comprised of eight classes of instruction, each identified by a different OP code value. Variations in the operand specification provide a subset of instructions within the instruction class to give a total of thirty-two instructions. The eight classes of instruction are:

MOVE: 0 Data from the source register or IV bus is moved to the destination register or IV bus. The data may be rotated any number of places and/or masked to any length during the MOVE operation. The source data field remains unchanged after the operation.

ADD: 1 Data from the source register or IV bus is added to the contents of the AUX register in ALU and the result is placed in the destination register or IV bus. The data may be rotated and/or masked during the operation. The source data field and the AUX register remain unchanged unless one is also the destination.

AND: 2 Data from the source register or IV bus is ANDed with the contents of the AUX register and the result is placed in the destination register or IV bus. The data may be rotated and/or masked during the operation. The source data field and AUX register remain unchanged unless one of those is also the destination.

XOR: 3 Data from the source register or IV bus undergoes an EXCLUSIVE OR comparison with the contents of the AUX register. The result is placed in the destination register or IV bus. Data may be rotated and/or masked during the operation. The source data field and AUX register remain unchanged unless one of those is also the destination.

XEC: 4 Causes execution of the instruction at the address formed by replacing the least significant bits of the last address with the sum of the I field and the data in the source register or IV byte. After execution of the instruction at the specified address, instruction execution continues at the address following the XEC instruction, unless the executed instruction caused a jump.

NZT: 5 The least significant bits of the instruction address are replaced by the I field data if the register or IV bus specified by the source field has non-zero contents. The tested data field remains unchanged.

XMIT: 6 The data in the I field is placed in the register or IV bus specified as the destination.

JMP: 7 The address of the next instruction to be executed is changed to that specified by the 13-bit A field of the instruction.

MCCAP — MicroComputer Cross Assembler Program for the 8X300

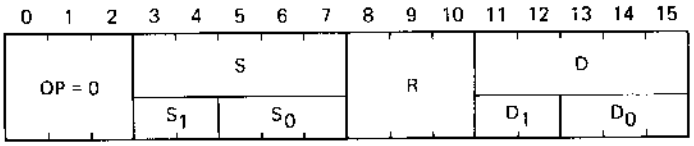
A cross assembler program is available to translate programs written in mnemonic source code. This is far more convenient and includes the advantages of explanatory text within the source program and error detection during the assembly process. The relevant assembler statement is shown in each example of the 8X300 instructions to provide correlation between the cross assembler statements and the 8X300 instruction code. A complete description of MCCAP and listings of typical programs are given at the end of this manual.

The cross assembler is written in FORTRAN and can be used on most computer systems capable of accepting this source language.

MOVE Instructions — Op Code 0

MOVE, Register, Register

Format



Operation

(S) → D

Description

The contents of the register specified by S are right rotated as specified by R and placed in the destination register specified by D. The contents of the source register remain unchanged. The original contents of the destination register are lost.

S specifies the source register.

R specifies the number of places that the source data is to be rotated.

D specifies the destination register.

The order of operation is:

copy the contents of the source register;

right rotate the copied data R places;

move the rotated data to the destination register.

Permitted operand values

S: 00/01/02/03/04/05/06/10/11

R: 0/1/2/3/4/5/6/7

D: 00/01/02/03/04/05/06/11

MOVE Instructions — Op Code 0

Example

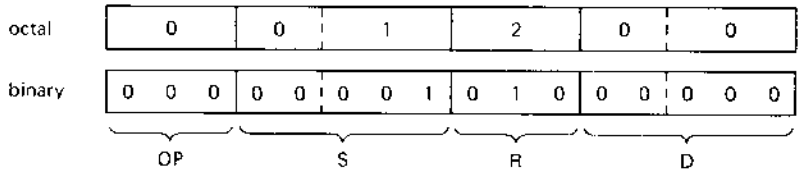
MOVE, Register, Register

Move the contents of R1, right rotated 2 places, to the AUX register.

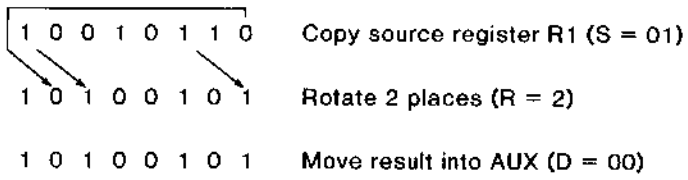
Instruction word

Assembler notation

MOVE R1 (2), AUX



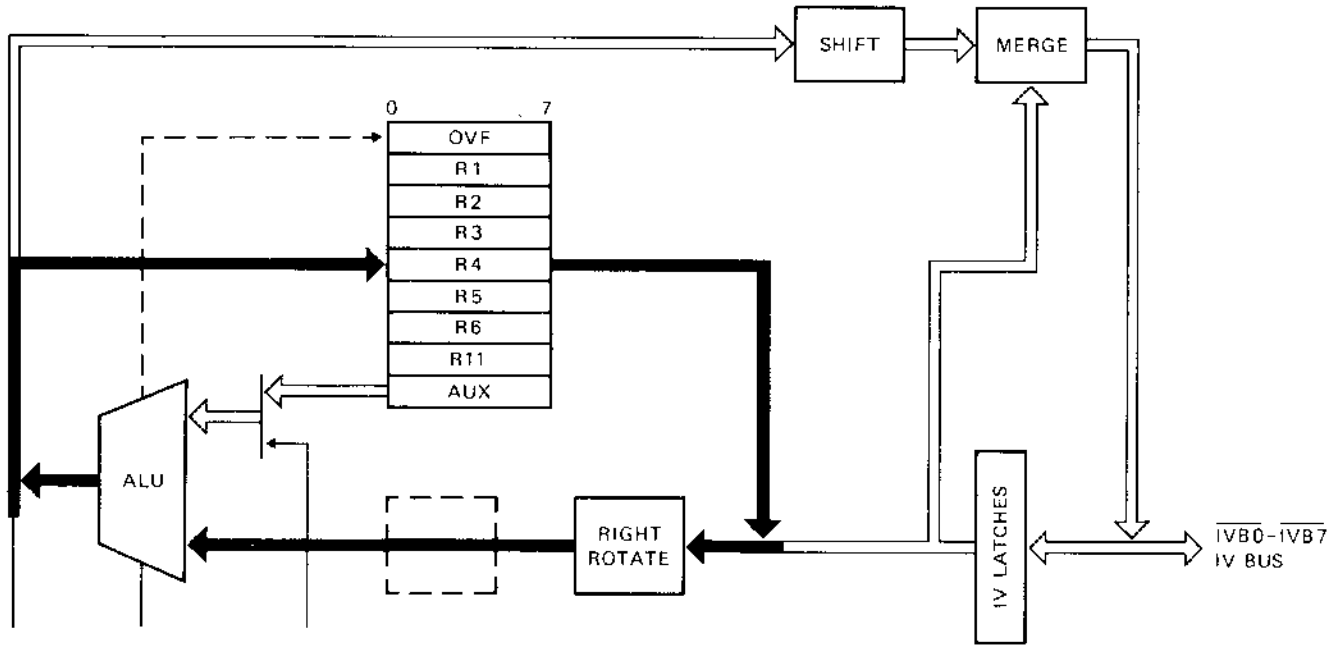
Instruction operation



Result

The original contents of the AUX register are replaced by the rotated data of R_1. The contents of R_1 are not changed.

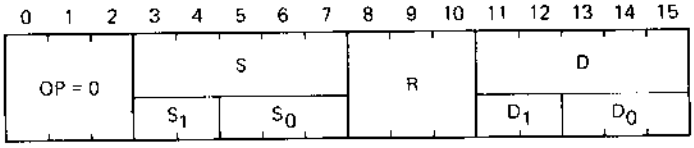
Data flow



MOVE Instructions — Op Code 0

MOVE, Register, IV bus address

Format



Operation

Enable IV byte with address (S)

Description

Enable the IV byte, at the bank specified by D, whose address is given by the right rotated contents of the register specified by S.

S specifies the source register.

R specifies the number of places that the source data is to be rotated.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies the left bank;
D = 17 specifies the right bank.

The order of operation is:

rotate the copied contents of register S by R places;

output the result to the IV bus as an address.

The contents of the source register remain unchanged after the instruction.

Operand values

S: 00/01/02/03/04/05/06/10/11

R: 0/1/2/3/4/5/6/7

D: 07/17

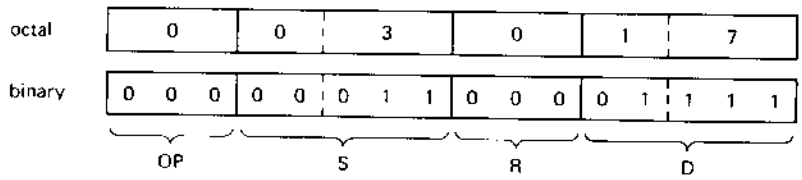
MOVE Instructions — Op Code 0

MOVE, Register, IV bus address

Example

Select the working storage register at the right bank whose address is given by the contents of R3.

Instruction word



Assembler notation

MOVE R3 (0), IVR

Instruction operation

0 0 0 0 0 1 0 0 copy source register R3 (S = 03)

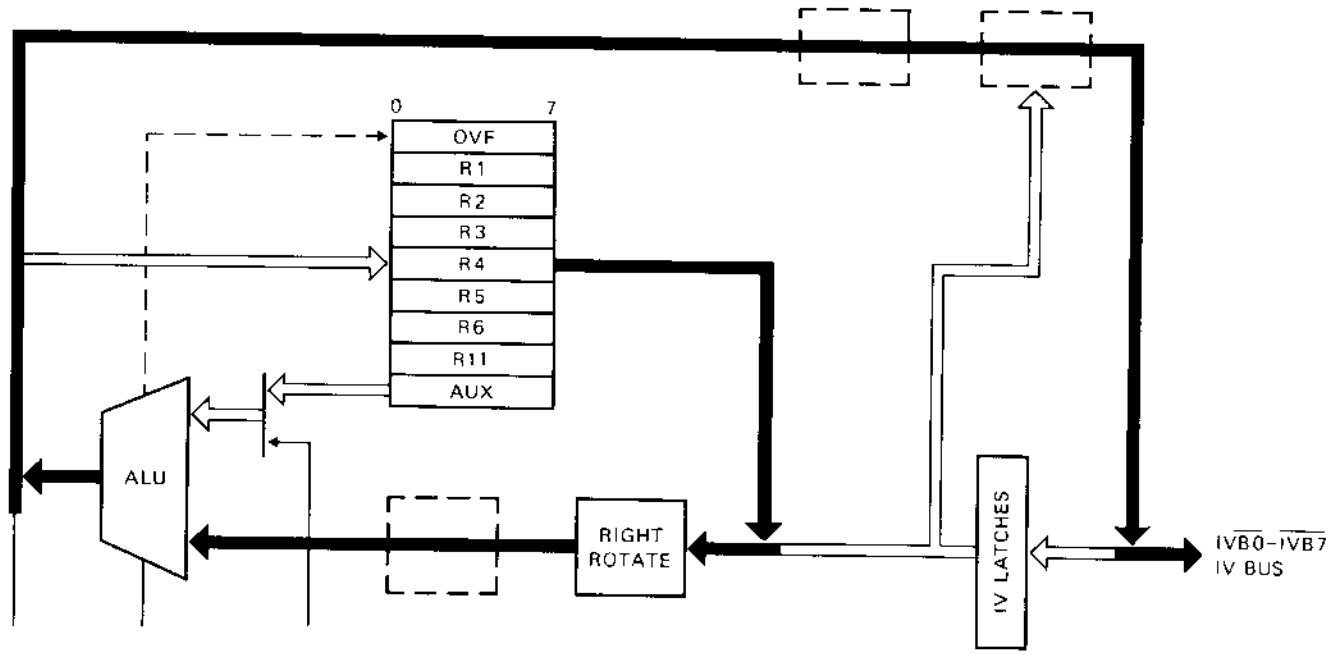
0 0 0 0 0 1 0 0 rotate 0 places

0 0 0 0 0 1 0 0 output to IV bus as address on the right bank (D = 17)

Result

The previously enabled IV byte on the right bank is disabled and the byte with address 004 on the right bank is enabled.

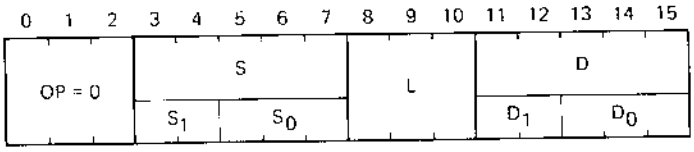
Data Flow



MOVE Instructions — Op Code 0

MOVE, Register, IV bus

Format



Operation

(S) → D

Description

Move the least significant L bits of the register specified by S to the variable length field of the IV bus.

S specifies the source register.

L specifies the length (number of bits) of the masked data field that is to be merged with the existing IV byte data. (L = 0 selects an 8-bit field.)

D_1 specifies the bank of the IV bus which is the destination:
D_1 = 2 selects the left bank;
D_1 = 3 selects the right bank.

D_0 specifies the bit position in the IV byte with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that bit 7 is aligned with bit D_0 of the IV bus.

The order of operation is:

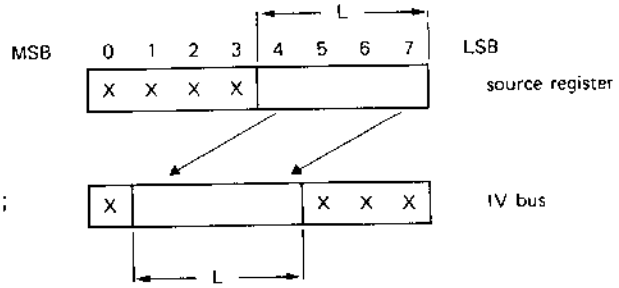
read the contents of the selected IV byte into the IV latches;

copy the contents of the source register;

shift the copied data field as specified by D_0;

merge the least significant L bits with the data in the IV latches;

output the modified data field to the IV byte.



Note that the original data in the IV byte outside the merged L-bit field remains unaltered. The contents of the source register remain unchanged.

Operand values

S: 00/01/02/03/04/05/06/10/11

L: 1/2/3/4/5/6/7/0

D_0: 0/1/2/3/4/5/6/7

D_1: 2/3

Note that L = 0 selects an 8-bit field.

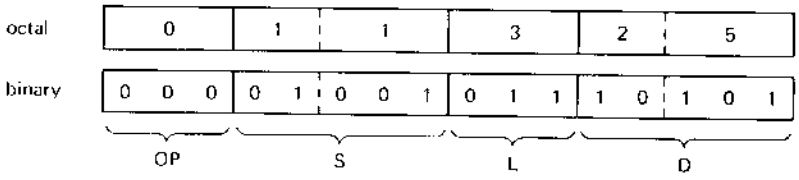
MOVE Instructions — Op Code 0

Example

MOVE, Register, IV bus

Move the contents of the least significant 3 bits of register 11 to the selected IV byte at the left bank, with bit 5 as the least significant position of the IV byte.

Instruction word



Assembler notation

MOVE R11, 3, LIV5

Instruction operation

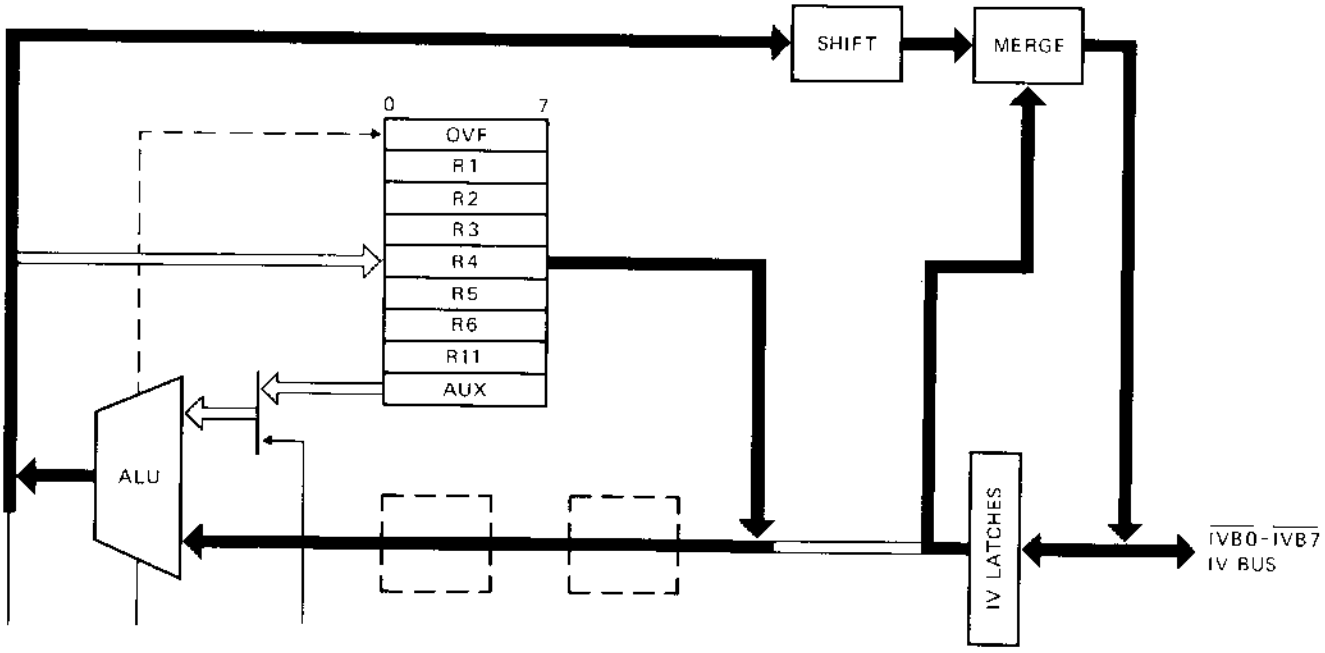
1 1 1 0 1 1 1 1	original IV byte data to input latches
1 1 0 1 1 0 0 0	copy contents of R11 (S = 11)
0 1 1 0 0 0	shift ALU output (D_0 = 5)
1 1 1 0 0 0 1 1	merge the 3-bit field with existing IV data (L = 3)

previous values of IV bus preserved in new IV data

Result

Content of bits 5, 6 and 7 of R11 inserted in bits 3, 4 and 5 of the IV byte. Bits 0, 1, 2, 6 and 7 of the IV byte unchanged.

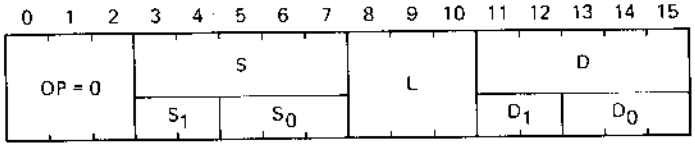
Data flow



MOVE Instructions — Op Code 0

MOVE, IV bus, Register

Format



Operation

(IV byte) → D

Description

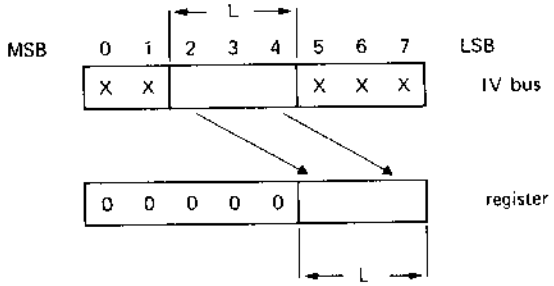
Move the L-bit field of the IV bus data to the least significant L bits of the register specified by D.

S1 specifies the bank of the IV bus which is the data source.
S_1 = 2 selects the left bank;
S_1 = 3 selects the right bank.

S_0 specifies the bit which will be the least significant bit of the input data field after rotation.

L specifies the length (number of bits) of the masked field.
Note that L = 0 selects an 8-bit field.

D specifies the address of the destination register.



The order of operation is:

read data on IV bus specified by S_1 to input latches;

right rotate the input data field as given by S_0;

mask off the least significant L bits of the rotated field;

move the masked field to the least significant L bits of the destination register, with zeros in the unmasked positions.

Operand values

S_0: 0/1/2/3/4/5/6/7

S_1: 2/3

L: 1/2/3/4/5/6/7/0

D: 00/01/02/03/04/05/06/11

MOVE Instructions — Op Code 0

Example

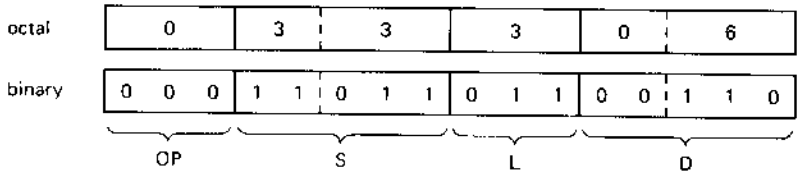
MOVE, IV bus, Register

Move bits 1, 2 and 3 of the enabled IV byte at the right bank to register 6.

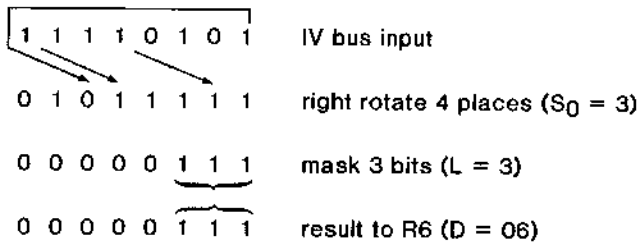
Instruction word

Assembler notation

MOVE RIV3, 3, R6



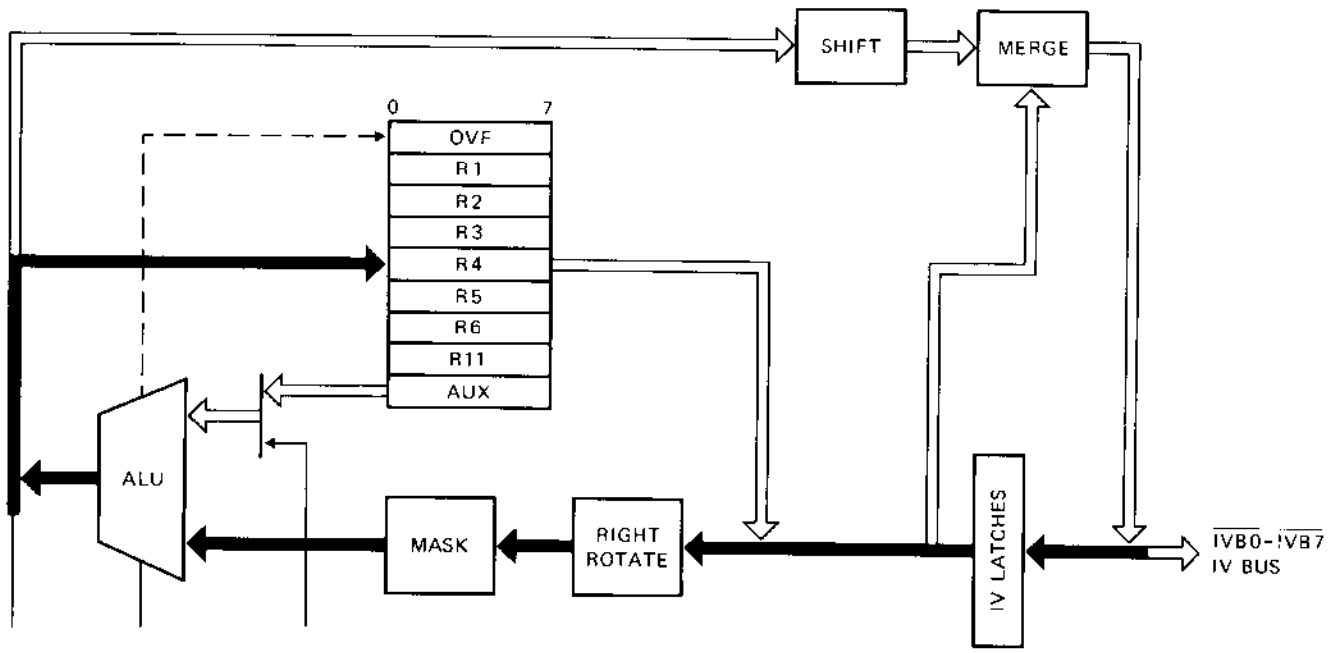
Instruction operation



Result

Bits 1, 2 and 3 of the IV byte at the right bank are inserted into the least significant 3 bits of R6. The other bits of R6 are set to zero. The source IV byte is not altered.

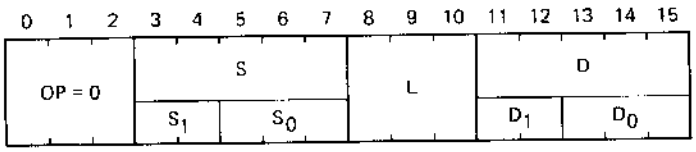
Data flow



MOVE Instructions — Op Code 0

MOVE, IV bus, IV bus

Format



Operation

(S) → D

Description

Move the variable length field, specified by S_0 and L, from the bank specified by S_1 to the field and bank specified by D.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 selects the left bank;
S_1 = 3 selects the right bank.

S_0 specifies the bit which will be the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field that is to be processed and merged with the existing IV bus data.
Note that L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
D_1 = 2 selects the left bank;
D_1 = 3 selects the right bank.

D_0 specifies the bit position in the data from the input latches with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that bit 7 is aligned with bit D_0 of the input latches.

The order of operation is:

read the data from the source IV byte into the input latches;

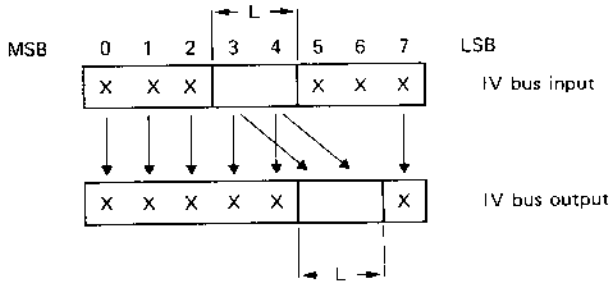
copy the input data and right rotate as specified by S_0;

mask off the least significant L bits;

shift left as specified by D_0;

merge the L-bit field with the data from the input latches;

output 8 bits of data to IV bus.



Note that during the merge phase, the original values of the source field bits outside the masked field are preserved.

Operand values

S_0: 0/1/2/3/4/5/6/7

S_1: 2/3

L: 1/2/3/4/5/6/7/0

D_0: 0/1/2/3/4/5/6/7

D_1: 2/3

MOVE Instructions — Op Code 0

Example

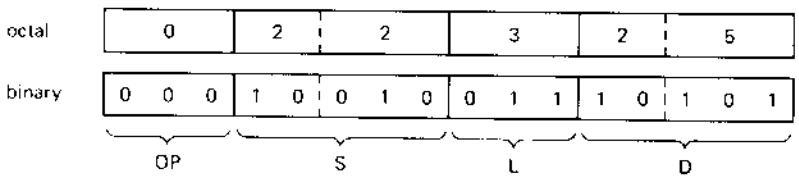
MOVE, IV bus, IV bus

Move bits 0, 1 and 2 of the IV byte at the Left Bank to bits 3, 4 and 5 of the same IV byte.

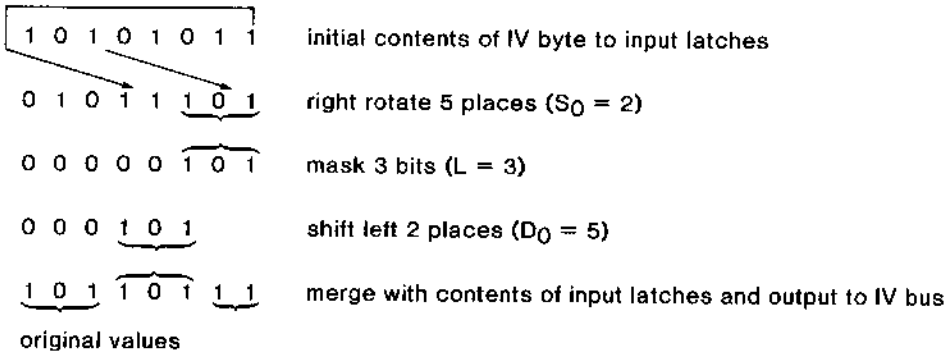
Instruction word

Assembler notation

MOVE LIV2, 3, LIV5



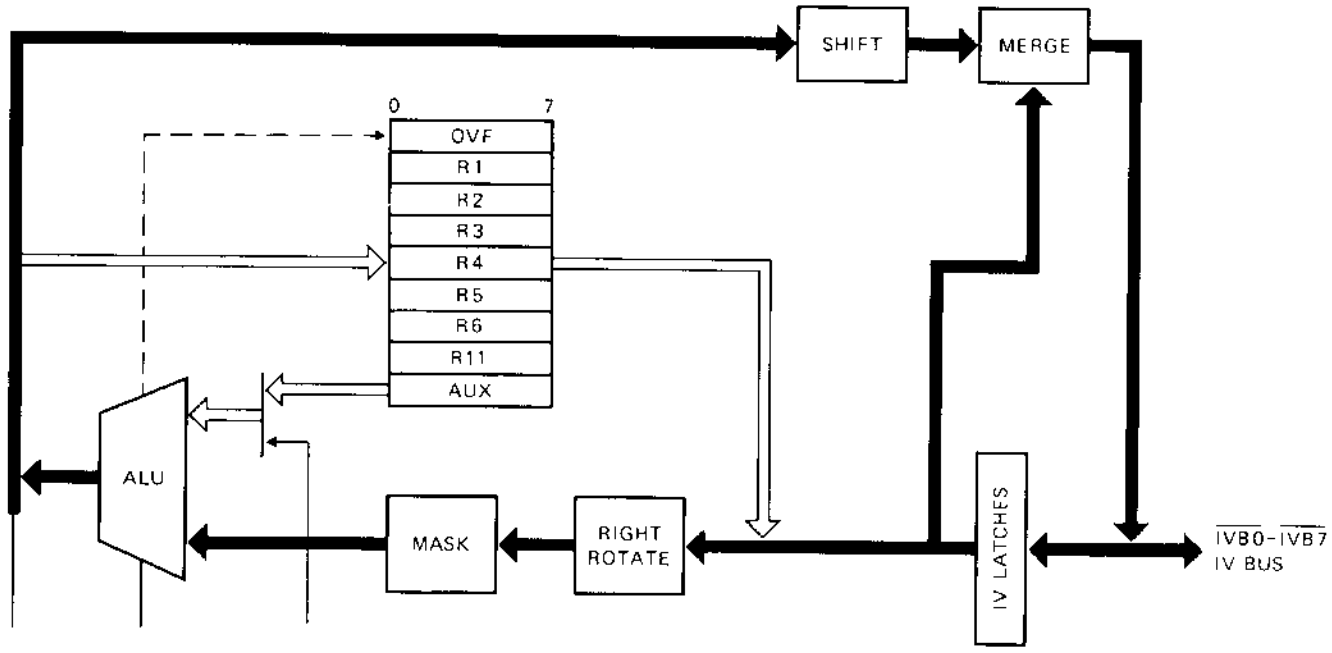
Instruction operation



Result

Bits 3, 4 and 5 contain the same values as bits 0, 1 and 2. All other bits unchanged.

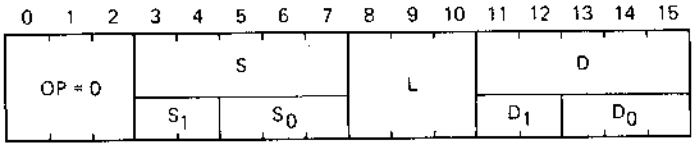
Data flow



MOVE Instructions — Op Code 0

MOVE, IV bus, IV bus address

Format



Operation

Enable the IV byte at the bank specified by D, whose address is given by the bus data specified by S.

Description

Copy the data from the IV bus as specified by S_1, right rotate the data field until bit S_0 is in the least significant position, mask the least significant L bits and output the result to the bank of the IV bus specified by D, as an IV byte address. Bits of the output field outside the mask are set to zero.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 selects the left bank;
S_1 = 3 selects the right bank.

S_0 specifies the bit which will be the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies left bank address (IVL);
D = 17 specifies right bank address (IVR).

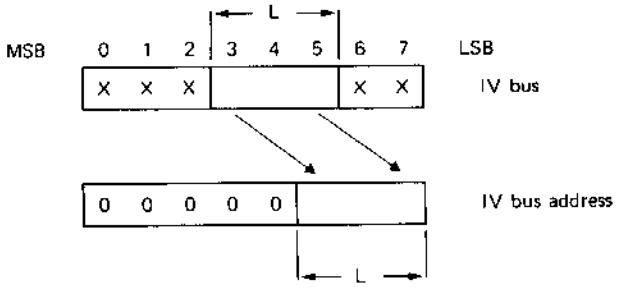
The order of operation is:

copy the input data on the IV bus;

right rotate the input data as given by S_0;

mask the least significant L bits;

output result with zeros in positions outside mask.



Operand values

S_0: 0/1/2/3/4/5/6/7

S_1: 2/3

L: 1/2/3/4/5/6/7/0

D: 07/17.

Note that L = 0 specifies 8-bit field.

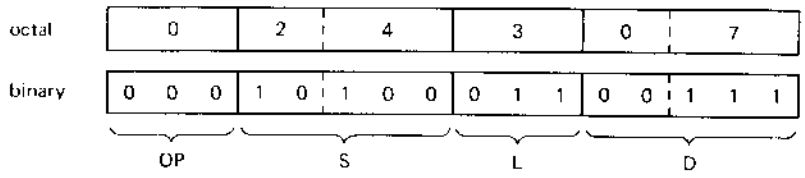
MOVE Instructions — Op Code 0

Example

MOVE, IV bus, IV bus address

Enable the IV byte at the left bank whose address is the value of bits 2, 3 and 4 of the presently enabled IV byte at the left bank.

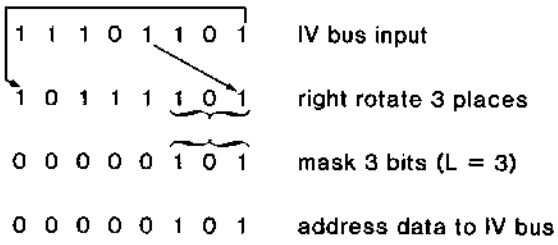
Instruction word



Assembler notation

MOVE LIV4, 3, IVL

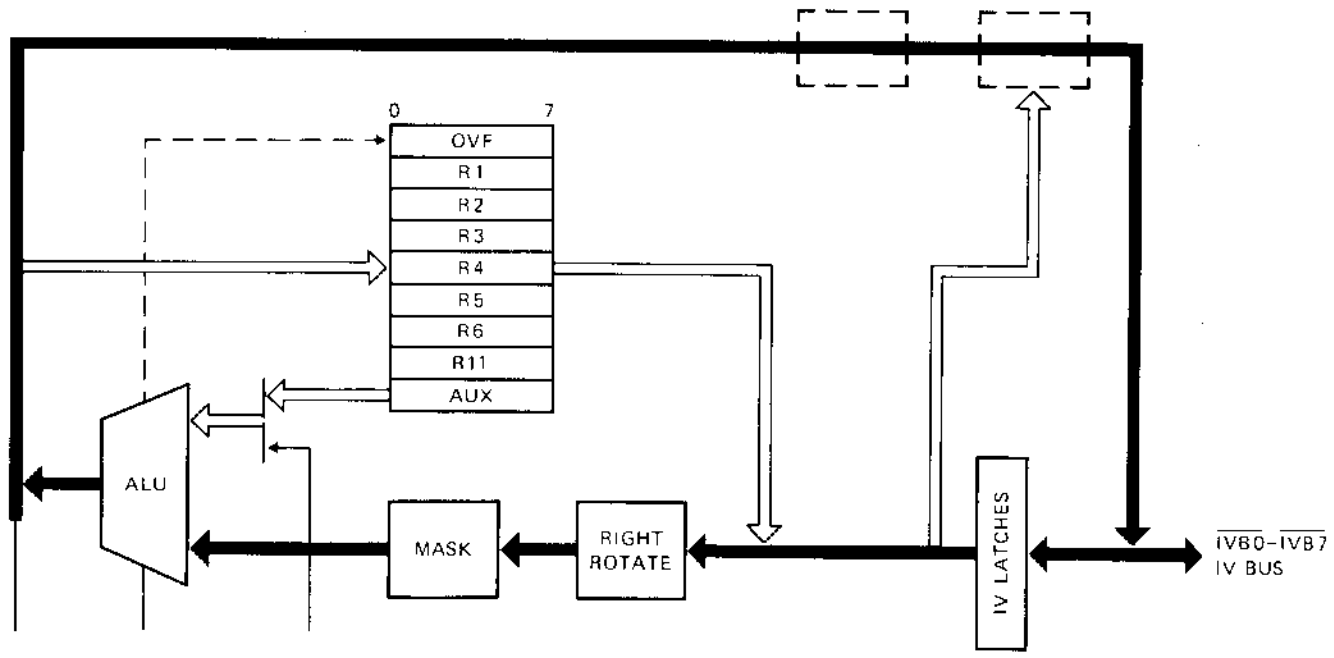
Instruction operation



Result

The IV byte, at the address given by bits 2, 3 and 4 of the previously enabled byte, is enabled. As both bytes are on the same bank, the source byte is disabled when the new address is output on the bus.

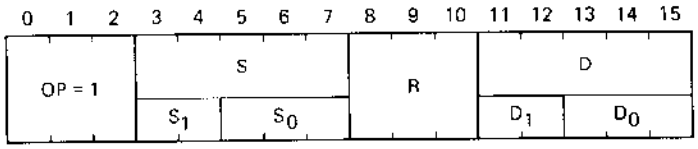
Data flow



ADD Instructions — Op Code 1

ADD, Register, Register

Format



Operation

(S) plus (AUX) → D

Description

Add the right rotated contents of register S to the contents of the AUX register and place the result in register D. If overflow occurs during the addition, bit 7 of the OVF register is set to 1, otherwise it is set to 0.

S specifies the source register.

R specifies the number of places that the source data is to be rotated.

D specifies the destination register.

The order of operation is:

copy the contents of the source register;

right rotate the copied data;

add the right rotated data to the contents of the AUX register;

move the result to the destination register;

set the overflow indication as appropriate.

The contents of the source and AUX registers remain unchanged after the instruction unless one of these is also specified as the destination.

Operand values

S: 00/01/02/03/04/05/06/10/11

R: 0/1/2/3/4/5/6/7

D: 00/01/02/03/04/05/06/11

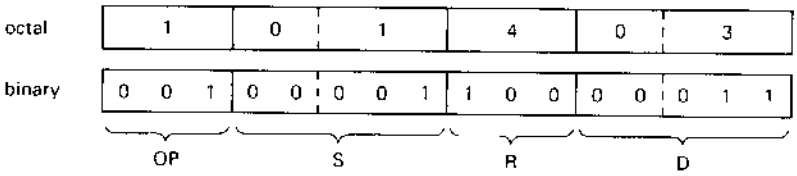
ADD Instructions — Op Code 1

Example

ADD, Register, Register

Add the contents of R1, right rotated 4 places, to the contents of the AUX register and store the result in R3.

Instruction word



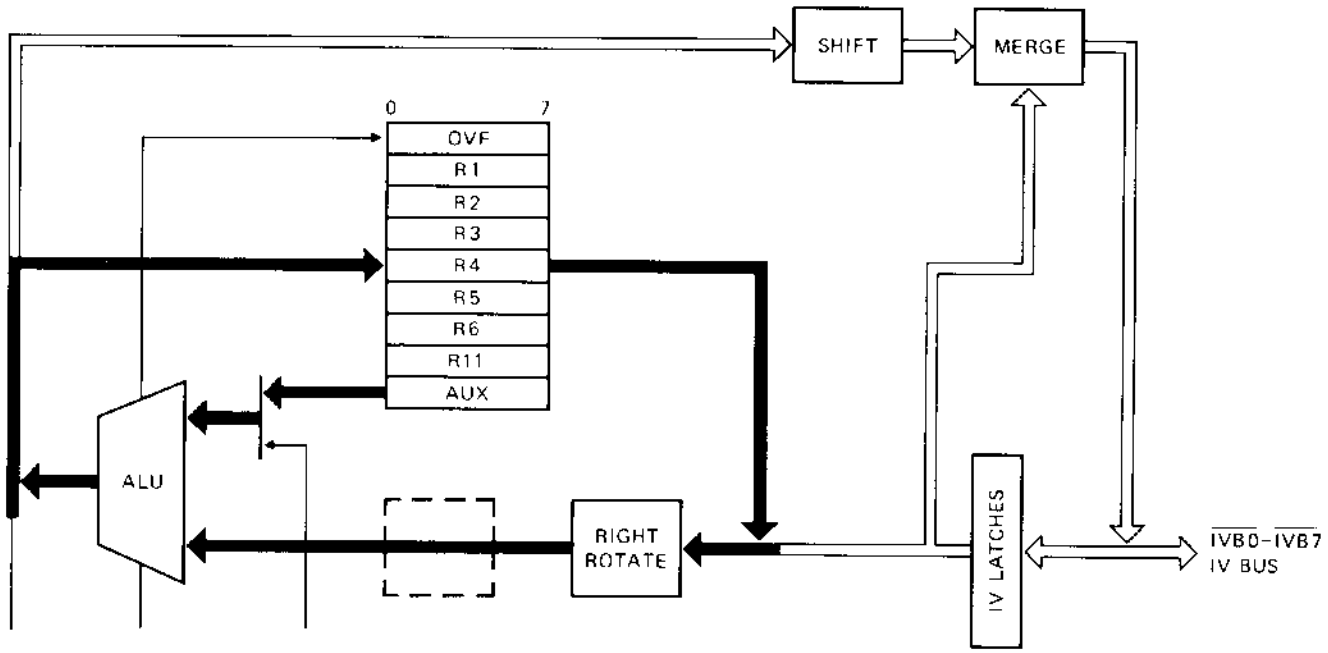
Assembler notation

ADD R1 (4), R3

Instruction operation

0 1 1 1 0 0 0 1	copy source register
0 0 0 1 0 1 1 1	rotate 4 places
1 1 1 0 1 0 0 1	contents of AUX
0 0 0 0 0 0 0 0	sum
0 0 0 0 0 0 0 1	overflow indication in OVF register
0 0 0 0 0 0 0 0	result in R_3

Data flow



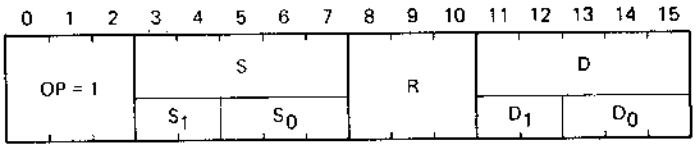
ADD Instructions — Op Code 1

ADD, Register, IV bus address

Operation

Enable the IV byte with address (S) plus (AUX).

Format



Description

Enable the IV byte whose address is given by the sum of the right rotated contents of the source register and the contents of the AUX register at the bank specified by D.

S specifies the source register.

R specifies the number of places that the source data is to be rotated.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies left bank address (IVL);
D = 17 specifies right bank address (IVR).

The order of operation is:

rotate the copied contents of register (S) by R places;

add the rotated data field to the contents of AUX;

set the overflow indication as appropriate;

output the sum to the IV bus as an address.

The contents of the source register remain unchanged after the instruction.

Operand values

S: 00/01/02/03/04/05/06/10/11

R: 0/1/2/3/4/5/6/7

D: 07/17

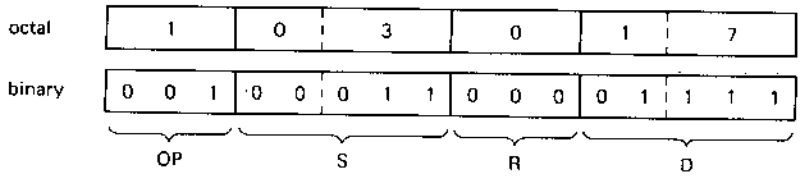
ADD Instructions — Op Code 1

Example

ADD, Register, IV bus address

Enable the IV byte at the right bank whose address is the sum of the contents of R3 and AUX.

Instruction word



Assembler notation

ADD R3 (0), IVR

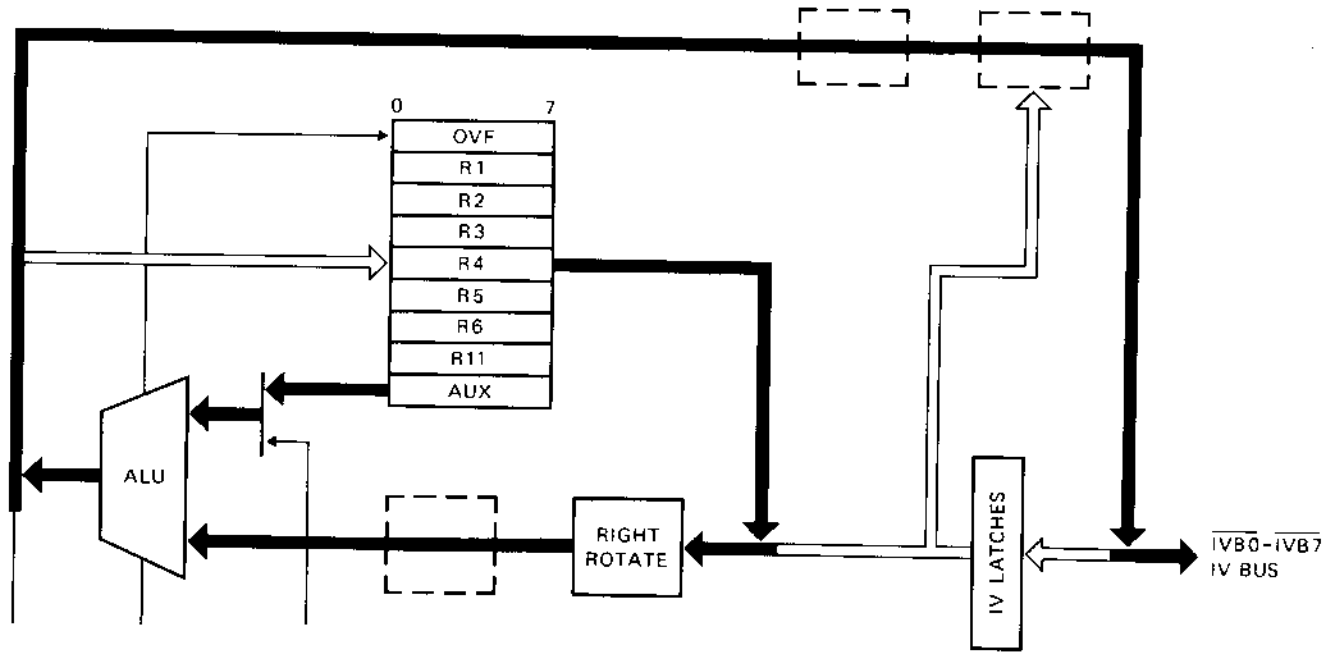
Instruction operation

0 0 0 1 0 1 1 0	copy source data
0 0 0 1 0 1 1 0	no rotation (R = 0)
0 0 0 0 0 1 0 1	contents of AUX
0 0 0 1 1 0 1 1	sum
0 0 0 0 0 0 0 0	OVF register after addition
0 0 0 1 1 0 1 1	result to IV bus as an address at the right bank (D = 17)

Result

The IV byte on the right bank, whose address is the sum of the contents of R3 and the AUX register, is enabled.

Data flow



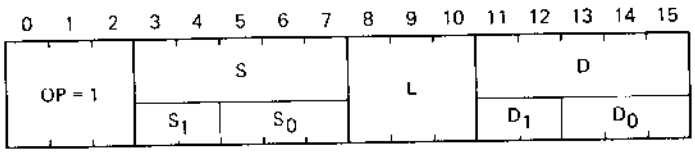
ADD Instructions — Op Code 1

ADD, Register, IV bus

Operation

(S) plus (AUX) → D

Format



Description

Add the contents of the source register to the contents of the AUX register and move the least significant L bits of the result to the IV bus as given by D.

S specifies the source register.

L specifies the length (number of bits) of the masked field that is to be merged with the existing IV byte data. Note that L = 0 selects an 8-bit field.

D_1
specifies the bank of the IV bus which is the destination:
D_1 = 2 selects the left bank;
D_1 = 3 selects the right bank.

D_0
specifies the bit position in the IV byte with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that bit 7 is aligned with bit D_0 of the IV bus.

The order of operation is:

the contents of the destination IV byte are read into the input latches;

the contents of the source register are copied and added to the contents of the AUX register;

the result is left shifted as specified by D;

the overflow indication is set as appropriate;

the shifted data field is merged with the original contents of the IV byte and output to the IV bus.

Note that the bits of the output data field outside the L-bit masked field retain their original values. The contents of the source register remain unchanged after the instruction.

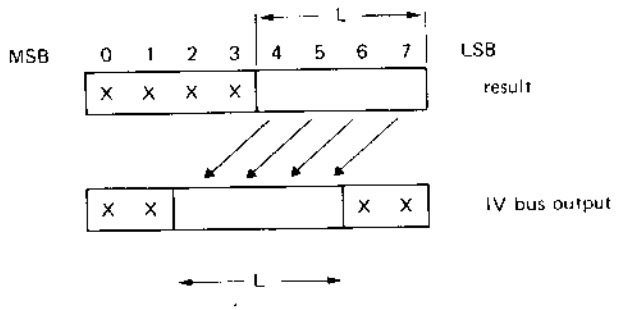
Operand value

S: 00/01/02/03/04/05/06/10/11

L: 1/2/3/4/5/6/7/0

D_0: 0/1/2/3/4/5/6/7

D_1: 2/3



ADD Instructions — Op Code 1

Example

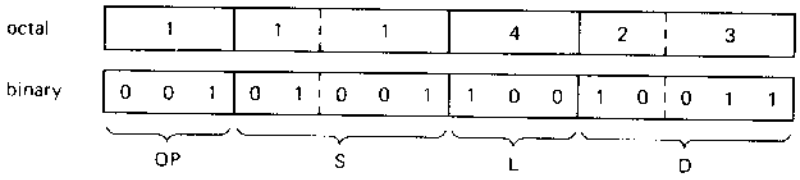
ADD, IV bus, Register

Add the contents of R11 to the contents of the AUX register and output the least significant 4 bits of the sum to bits 0, 1, 2 and 3 of the IV byte at the Left Bank.

Instruction word

Assembler notation

ADD R11, 4, LIV3



Instruction operation

0 0 1 0 0 0 1 1 — copy source data

1 1 0 1 0 1 1 1 — contents of AUX

1 1 1 1 1 0 1 0 — sum

1 0 1 0 — shift left 4 places (D_0 = 3)

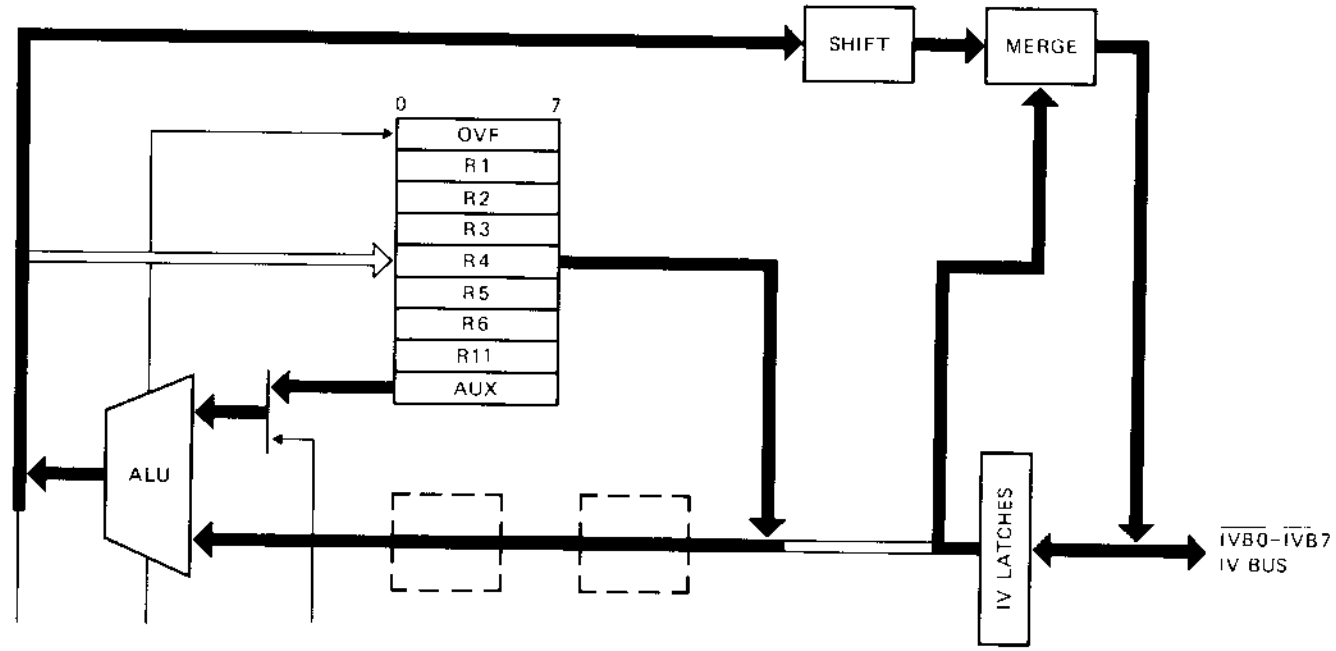
1 0 1 0 1 1 0 1 — merge with original IV bus data and output to IV bus left bank (D_1 = 2)

original content preserved

Result

Bits 0, 1, 2 and 3 of the IV byte at the left bank are set to the values of the least significant 4 bits of the sum of (R11) and (AUX). The overflow indicator is set to zero.

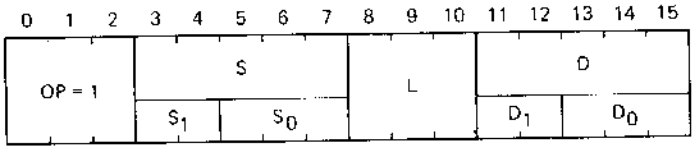
Data flow



ADD Instructions — Op Code 1

ADD, IV bus, IV bus

Format



Operation

(S) plus (AUX) → D

Description

Add the L-bit field of the IV bus source data to the contents of the AUX register and move the least significant L bits of the result to the IV bus field specified by D_0.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 selects the left bank;
S_1 = 3 selects the right bank.

S_0 specifies the bit which will be the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field that is to be processed and merged with the existing IV bus data.
Note that L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
D_1 = 2 selects the left bank;
D_1 = 3 selects the right bank.

D_0 specifies the bit position in the IV byte with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that bit 7 is aligned with bit D_0 of the IV bus.

The order of operation is:

read the data from the IV bus into the input latches;

right rotate the copied input data as given by S_0;

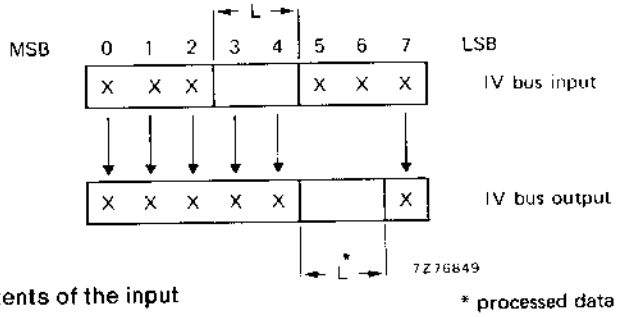
mask off L bits;

add the L-bit field to the contents of the AUX register;

left-shirt the sum as given by D_0;

merge the least significant L bits of the shifted field with the contents of the input latches;

output the merged 8-bit field to the bank of the IV bus given by D_1.



Note that during the merge phase, the original values of the source field outside the masked field are preserved. The original contents of the destinction field are lost.

Operand values

S_0: 0/1/2/3/4/5/6/7

S_1: 2/3

L: 1/2/3/4/5/6/7/0

D_0: 0/1/2/3/4/5/6/7

D_1: 2/3

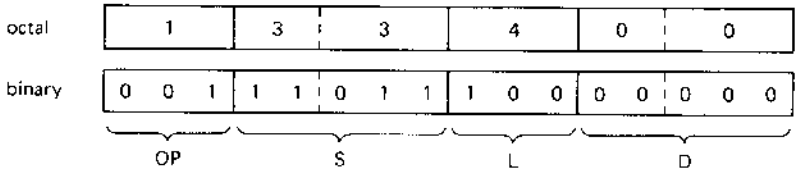
ADD Instructions — Op Code 1

Example

ADD, IV bus, Register

Add bits 0, 1, 2 and 3 of the IV byte at the right bank to the contents of the AUX register and store the result in the AUX register.

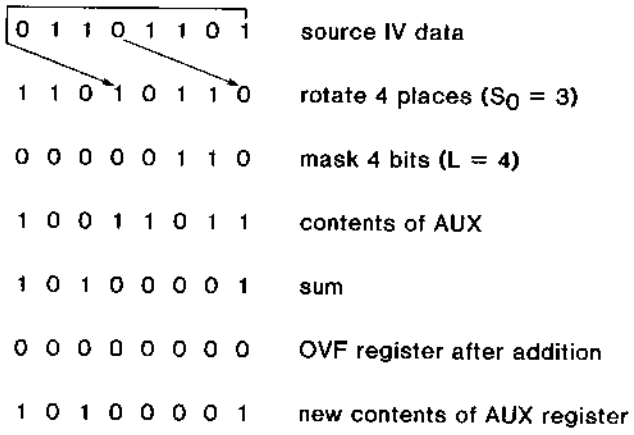
Instruction word



Assembler notation

ADD RIV3, 4, AUX

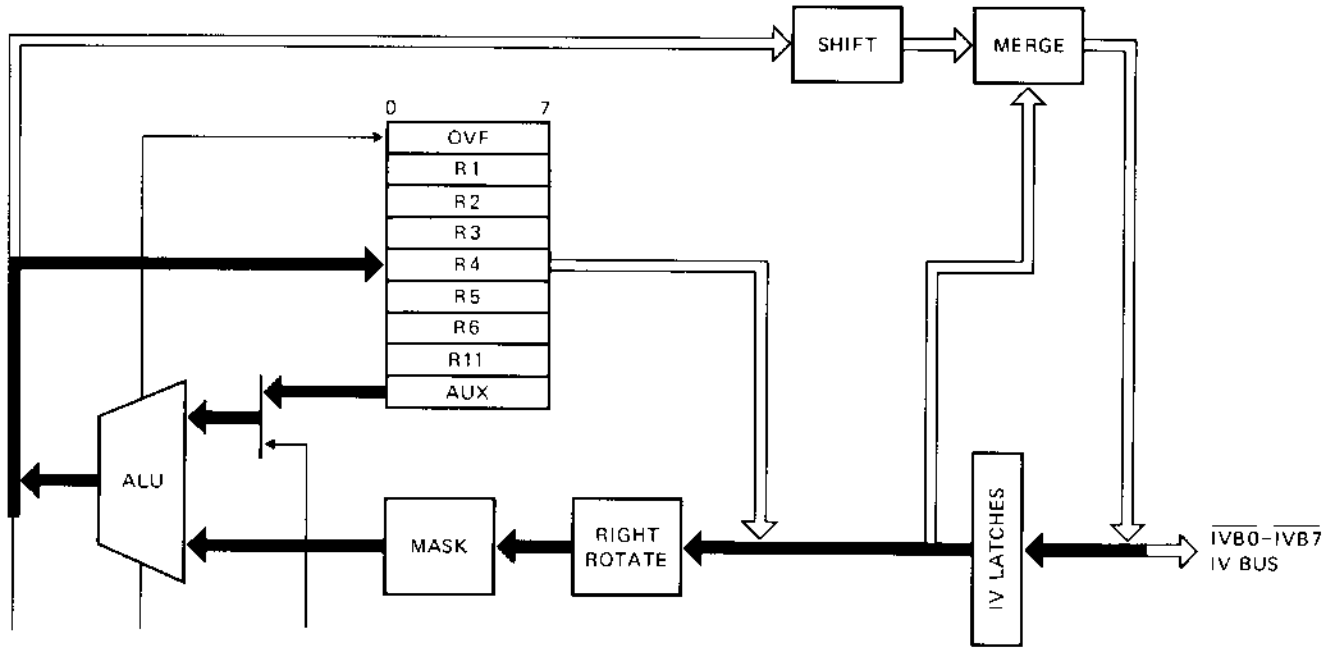
Instruction operation



Result

The 4 most significant bits of the IV byte are added to the AUX register contents. The overflow indicator is set to zero.

Data flow



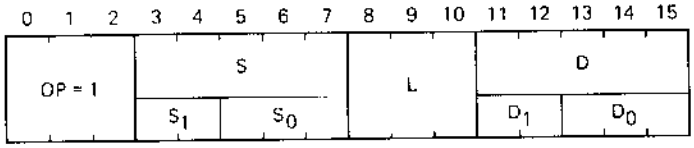
ADD Instructions — Op Code 1

ADD, Register, IV bus

Operation

(S) plus (AUX) → D

Format



Description

Add the L-bit field of the IV bus source data to the contents of the AUX register and place the result in the destination register. Set the overflow indicator as appropriate.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 selects the left bank;
S_1 = 3 selects the right bank.

S_0 specifies the bit which will be the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field.
Note that L = 0 selects an 8-bit field.

D specifies the address of the destination register.

The order of operation is:

read the source IV byte data into the input latches;

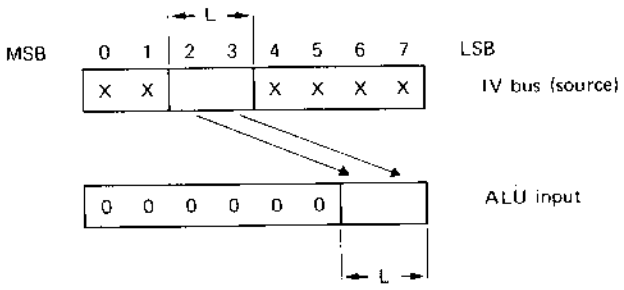
right rotate the input data as given by S_0;

mask the rotated data field as specified by L;

add the masked data to the contents of the AUX register;

set the overflow indicator as appropriate;

move the result of the addition to the destination register.



Operand values

S_0: 2/3

S_1: 0/1/2/3/4/5/6/7

L: 1/2/3/4/5/6/7/0

D: 00/01/02/03/04/05/06/11

Note that L = 0 selects an 8-bit field.

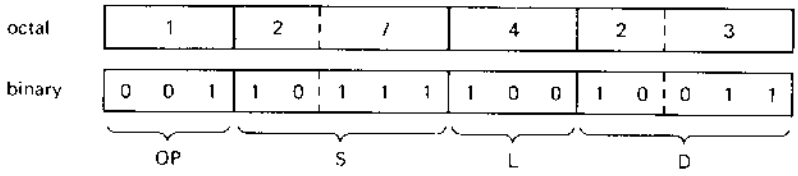
ADD Instructions — Op Code 1

ADD, IV bus, IV bus

Example

Add the contents of bits 4 to 7 of the IV byte at the left bank to the contents of the AUX register and move the least significant 4 bits of the sum to the most significant 4 bits of the IV byte at the left bank.

Instruction word



Assembler notation

ADD LIV7, 4, LIV3

Instruction word

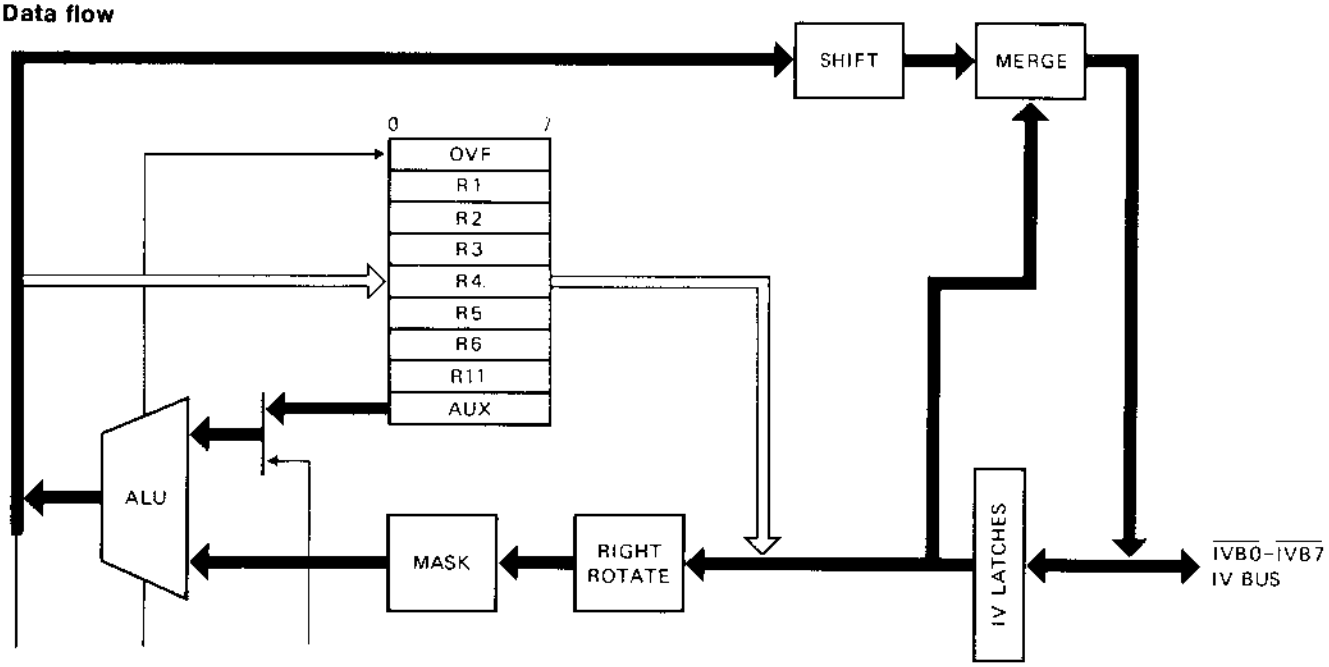
0 1 1 0 0 1 1 0	IV bus data to input latches
0 1 1 0 0 1 1 0	no right rotate (S_0 = 7)
0 0 0 0 0 1 1 0	mask 4 bits (L = 4)
0 0 1 1 0 0 1 0	contents of AUX
0 0 1 1 1 0 0 0	sum
1 0 0 0	shift left 4 places (D_0 = 3)
1 0 0 0 0 1 1 0	merge with input data and output to IV bus

original values

Result

The 4 most significant bits of the IV byte are changed to the values given by the sum of the 4 least significant bits and the contents of the AUX register. The overflow indicator is set to 0.

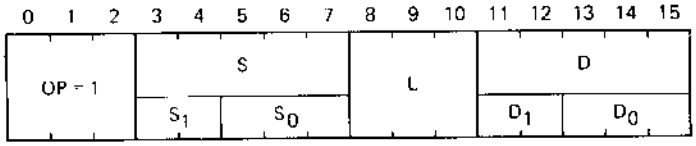
Data flow



ADD Instructions — Op Code 1

ADD, IV bus, IV bus address

Format



Operation

(S) plus (AUX)→D

Description

Enable the IV byte, at the bank specified by D_1, whose address is given by the sum of the L-bit field of the source data and the contents of the AUX register.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 selects the left bank;
S_1 = 3 selects the right bank.

S_0 specifies the bit which will be the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field
Note that L = 0 selects an 8-bit field.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies left bank address (IVL);
D = 17 specifies right bank address (IVR).

The order of operation is:

read the data from the current IV byte into the input latches;

right rotate the copied input data as given by S_0;

mask off the least significant L bits;

add the masked field to the contents of the AUX register;

set the overflow indicator as appropriate;

output the data as in IV bus address at the bank specified by D.

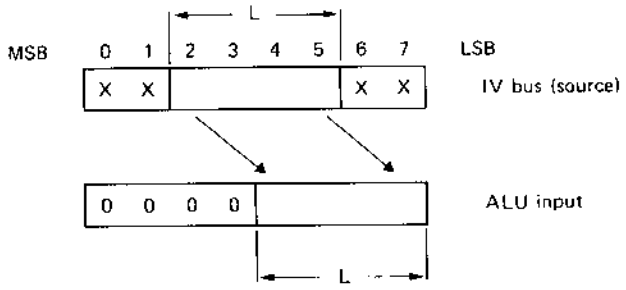
Operand values

S_0: 0/1/2/3/4/5/6/7

S_1: 2/3

L: 1/2/3/4/5/6/7/0

D: 07/17



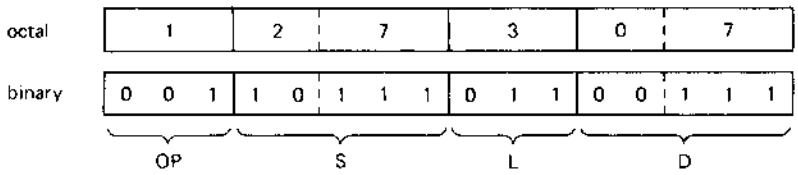
ADD Instructions — Op Code 1

Example

ADD, IV bus, IV bus address

Enable the IV byte at the left bank whose address is the sum of the contents of the AUX register and bits 5, 6 and 7 of the presently enabled IV byte at the left bank.

Instruction word



Assembler notation

ADD LIV7, 3, IVL

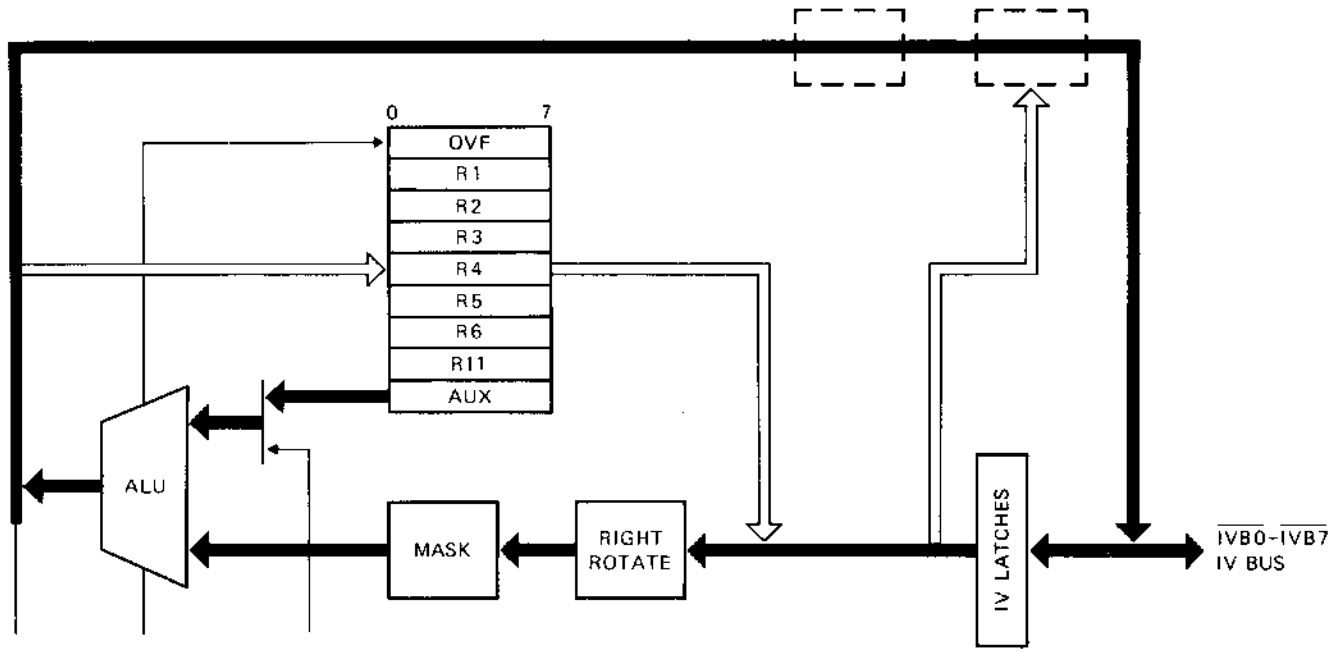
Instruction operation

1 1 0 1 1 1 0 1	IV bus input
1 1 0 1 1 1 0 1	no right rotate (S_0 = 7)
0 0 0 0 0 1 0 1	mask 3 bits (L = 3)
0 0 0 0 1 1 0 1	contents of AUX
0 0 0 1 0 0 1 0	sum
0 0 0 0 0 0 0 0	OVF register
0 0 0 1 0 0 1 0	new IV bus address, left bank.

Result

Original address at left bank disabled and new address, given by sum above, enabled.

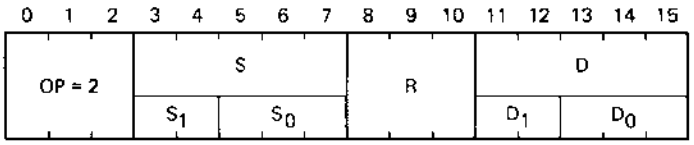
Data flow



AND Instructions — Op Code 2

AND, Register, Register

Format



Operation

(S) Λ (AUX) → D

Description

ADD the right rotated contents of register S with the contents of the AUX register and place the result in register D.

S specifies the source register.

R specifies the number of places that the source data is to be rotated.

D specifies the destination register.

The order of operation is:

copy the contents of the source register;

right rotate the copied data;

AND the right rotated data with the contents of the AUX register;

move the result to the destination register.

The contents of the source and AUX registers remains unchanged after the instruction unless one of these is also the destination register.

Operand values

S: 00/01/02/03/04/05/06/10/11
R: 0/1/2/3/4/5/6/7
D: 00/01/02/03/04/05/06/11

AND Instructions — Op Code 2

Example

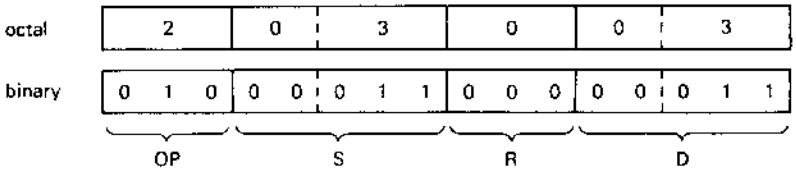
AND, Register, Register

AND the contents of R3 to the contents of the AUX register and store the result in R3.

Instruction word

Assembler notation

AND R3 (0), R3



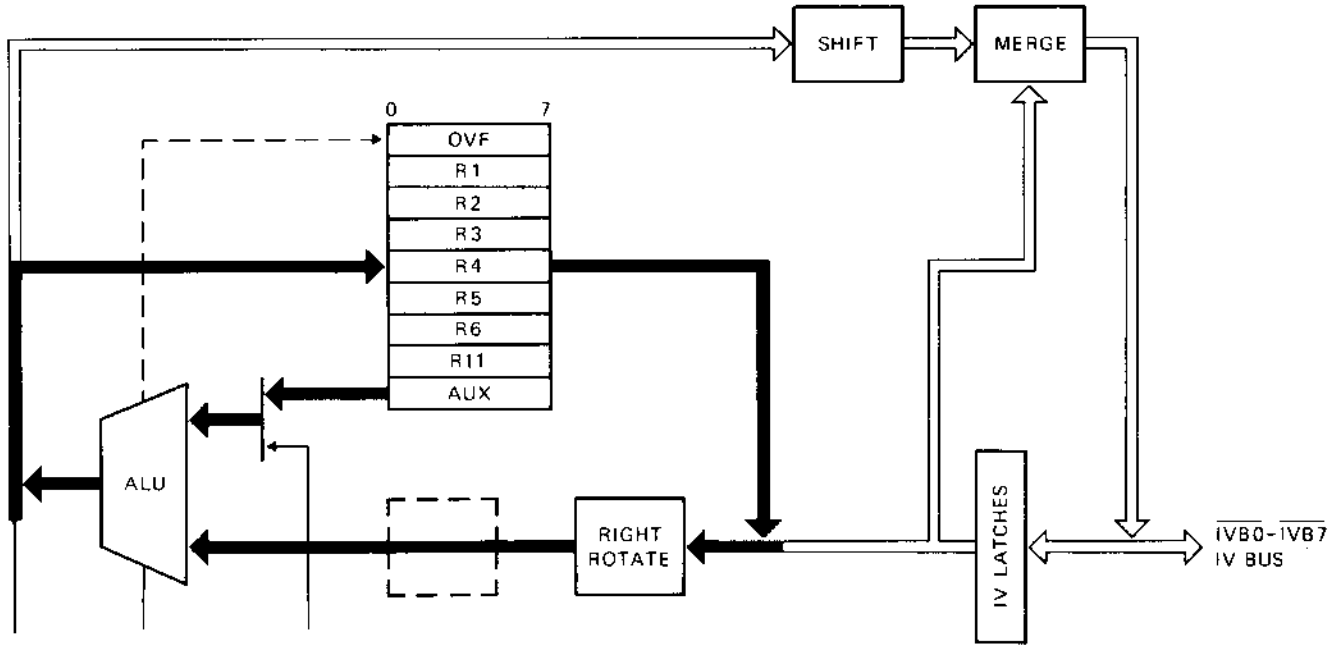
Instruction operation

0 0 0 0 1 1 1 1	initial contents of R3 (no rotation, R = 0)
0 0 0 0 0 0 1 1	contents of AUX
0 0 0 0 0 0 1 1	result of AND function
0 0 0 0 0 0 1 1	new contents of R3

Result

R3 new contains the result of ANDing its original contents with those of the AUX register.

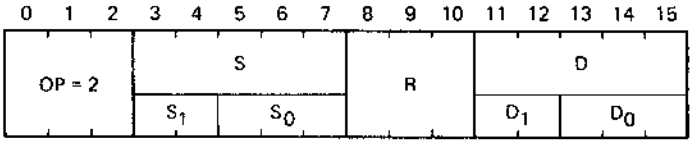
Data flow



AND Instructions — Op Code 2

AND, Register, IV bus address

Format



Operation

Enable IV byte with address (S) Λ (AUX).

Description

Enable the IV byte, at the bank specified by D, whose address is given by the AND operation on the right rotated contents of the source register and the contents of the AUX register.

S specifies the source register.

R specifies the number of places that the source data is to be rotated.

D specified the destination bank of the IV bus for the address data:
D = 07 specifies the left bank;
D = 17 specifies the right bank.

The order of operation is:

rotate the copied contents of register S by R places;

AND the rotated data field with the contents of AUX;

output the result to the IV bus as an address.

The contents of the source register remain unchanged after the instruction.

Operand values

S: 00/01/02/03/04/05/06/10/11

R: 0/1/2/3/4/5/6/7

D: 07/17

AND Instructions — Op Code 2

AND, Register, IV bus address

Example

Select the IV byte at the left bank, whose address is contained in bits 4, 5 and 6 of register R3. It is assumed that the AUX register already contains the required mask.

Instruction word

octal	2			0			3			1			0			7
binary	0	1	0	0	0	0	1	1	0	0	1	0	0	1	1	1
	OP			S					R			D				

Assembler notation

```
AND R3 (1), IVL
```

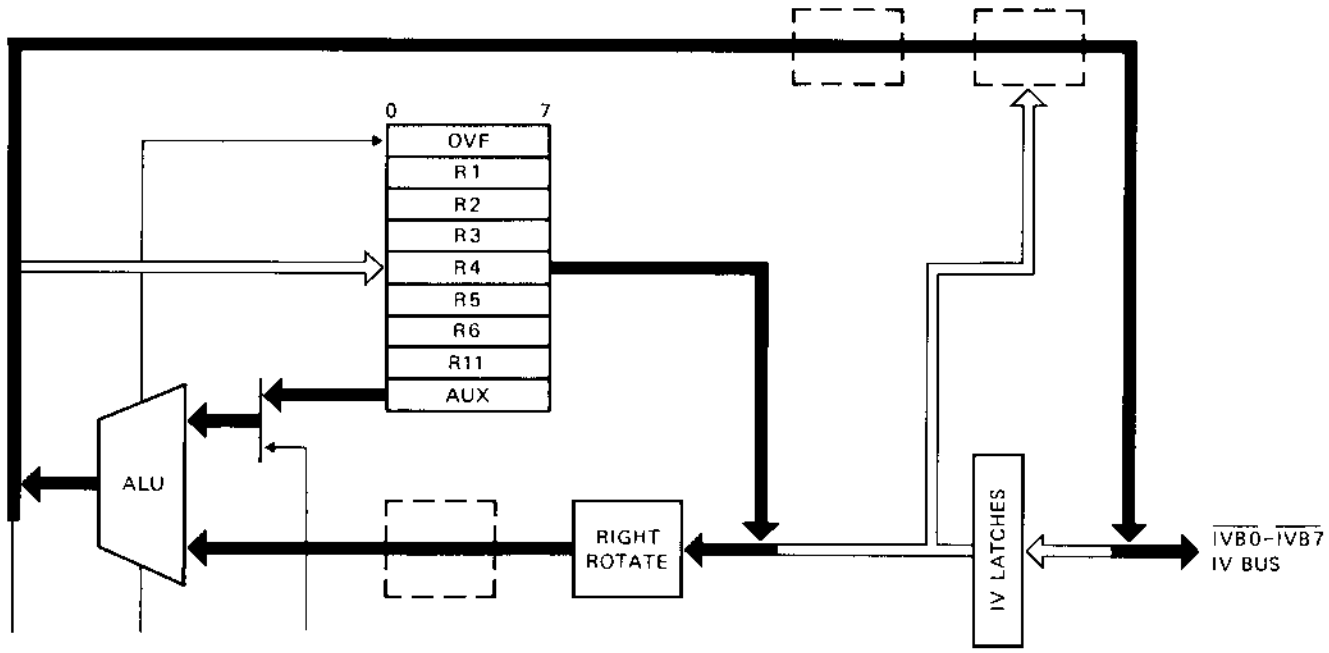
Instruction operation

```
1 1 0 1 1 1 0 1   copy source register
  \           /
1 1 1 0 1 1 1 0   rotate 1 place (R = 1)

0 0 0 0 0 1 1 1   contents of AUX

0 0 0 0 0 1 1 0   result of AND comparison

0 0 0 0 0 1 1 0   new left bank address (D = 07)
```

Result

The IV byte on the left bank with address 006 is enabled.

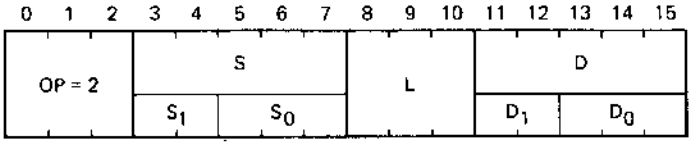
Data flow



AND Instruction — Op Code 2

AND, Register, IV bus

Format



Operation

(S) Λ (AUX)→D

Description

Perform an AND operation on the source register contents and those of the AUX register and move the least significant L bits of the result to the destination field of the IV bus.

S specifies the source register.

L specifies the length (number of bits) of the masked field that is to be merged with the existing IV byte data.
Note that L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
D_1 = 2 selects the left bank;
D_1 = 3 selects the right bank.

D_0 specifies the bit position in the IV byte with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that bit 7 is aligned with bit D_0 of the IV bus.

The order of operation is:

the contents of the destination IV byte are read into the input latches;

the contents of the source register are copied and ANDed with the contents of the AUX register.

the result is left-shifted as specified by D;

the shifted data field is merged with the original contents of the IV byte and output to the IV byte.

Note that the bits of the output data field outside the L-bit processed field retain their original values. The contents of the source register remain unchanged after the instruction.

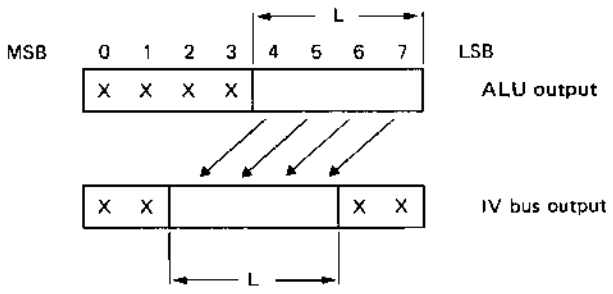
Operand values

S: 00/01/02/03/04/05/06/10/11

L: 1/2/3/4/5/6/7/0

D_0: 0/1/2/3/4/5/6/7

D_1: 2/3



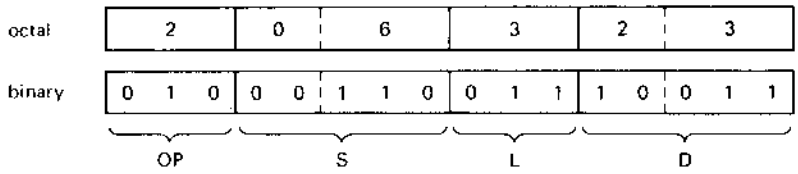
AND Instructions — Op Code 2

Example

AND, Register, IV bus

Perform an AND operation on the contents of R6 and the AUX register and move the least significant 3 bits of the result to bits 1, 2 and 3 of the IV byte at the left bank.

Instruction word



Assembler notation

AND R6, 3, LIV3

Instruction operation

1 0 1 1 0 1 0 1 copy source register

1 1 0 1 1 1 1 0 contents of AUX

1 0 0 1 0 1 0 0 result of AND comparison

0 1 0 0 shift left 4 places (D_0 = 3)

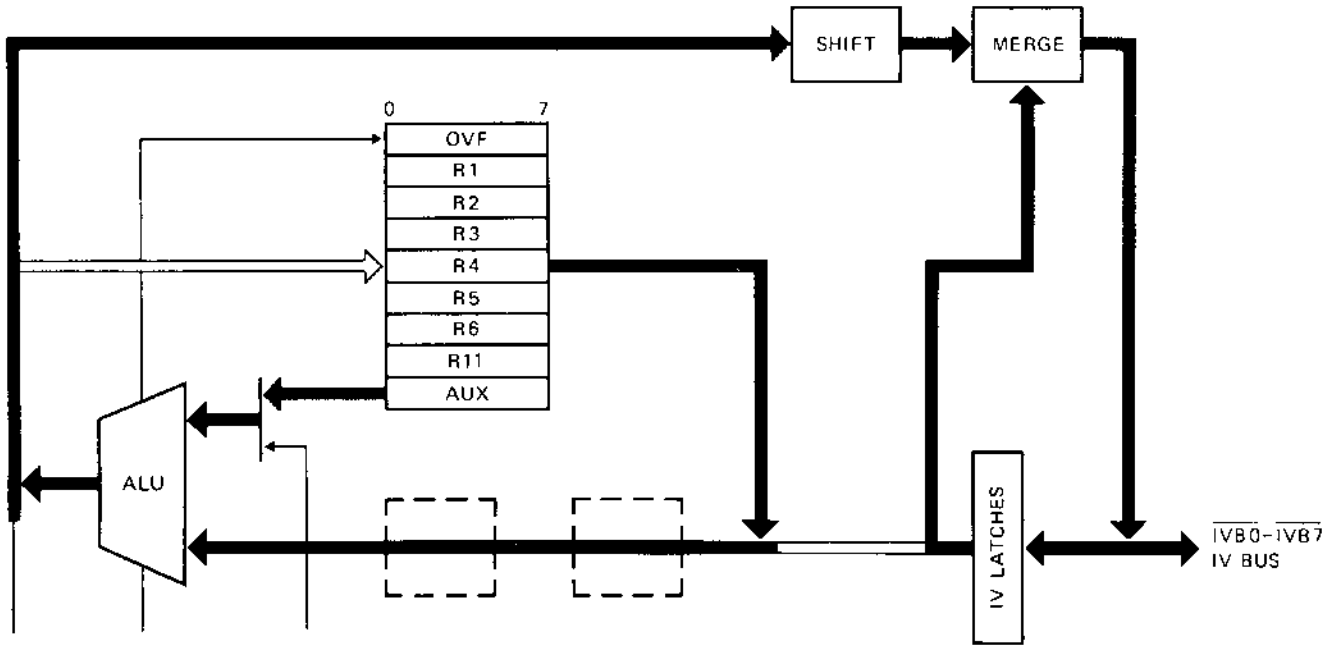
1 1 0 0 1 0 1 0 merge with original IV bus contents (L = 3)

original values

Result

The 3-bit result of the AND operation is moved to bits 1, 2 and 3 of the IV byte at the left bank.

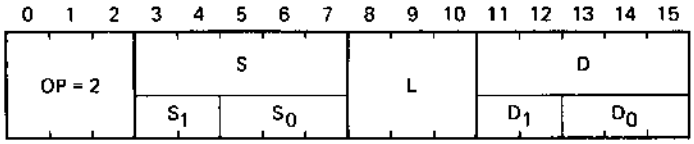
Data flow



AND Instructions — Op Code 2

AND, IV bus, Register

Format



Operation

(S) Λ (AUX)→D

Description

Perform an AND operation on the L-bit field of the IV bus source data and the contents of the AUX register.

S_1 specifies the bank of the IV bus which is the data source;
S_1 = 2 selects the left bank;
S_1 = 3 selects the right bank.

S_0 specifies the bit which will be the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field.
Note that L = 0 selects an 8-bit field.

D specifies the address of the destination register.

The order of operation is:

read the contents of the IV byte into the input latches;

right rotate the input data as given by S_0;

mask the rotated data field as specified by L;

AND the masked data to the contents of the AUX register;

move the result to the destination register.

Operand values

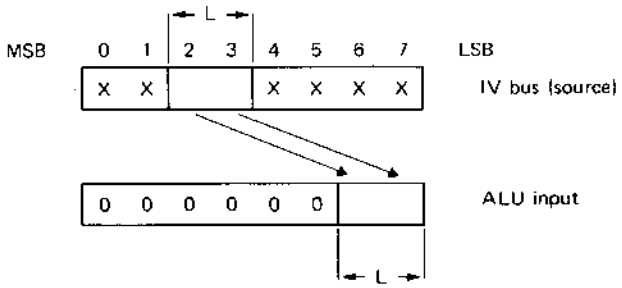
S_0 2/3

S_1 0/1/2/3/4/5/6/7

L: 1/2/3/4/5/6/7/0

D: 00/01/02/03/04/05/06/11.

Note that L = 0 selects an 8-bit field.



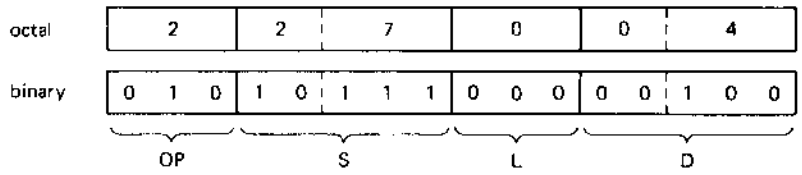
AND Instructions — Op Code 2

Example

AND, IV bus, Register

Perform an AND operation on the contents of the IV byte at the left bank and the contents of the AUX register and store the result in R4.

Instruction word



Assembler notation

AND LIV7, 8, R4

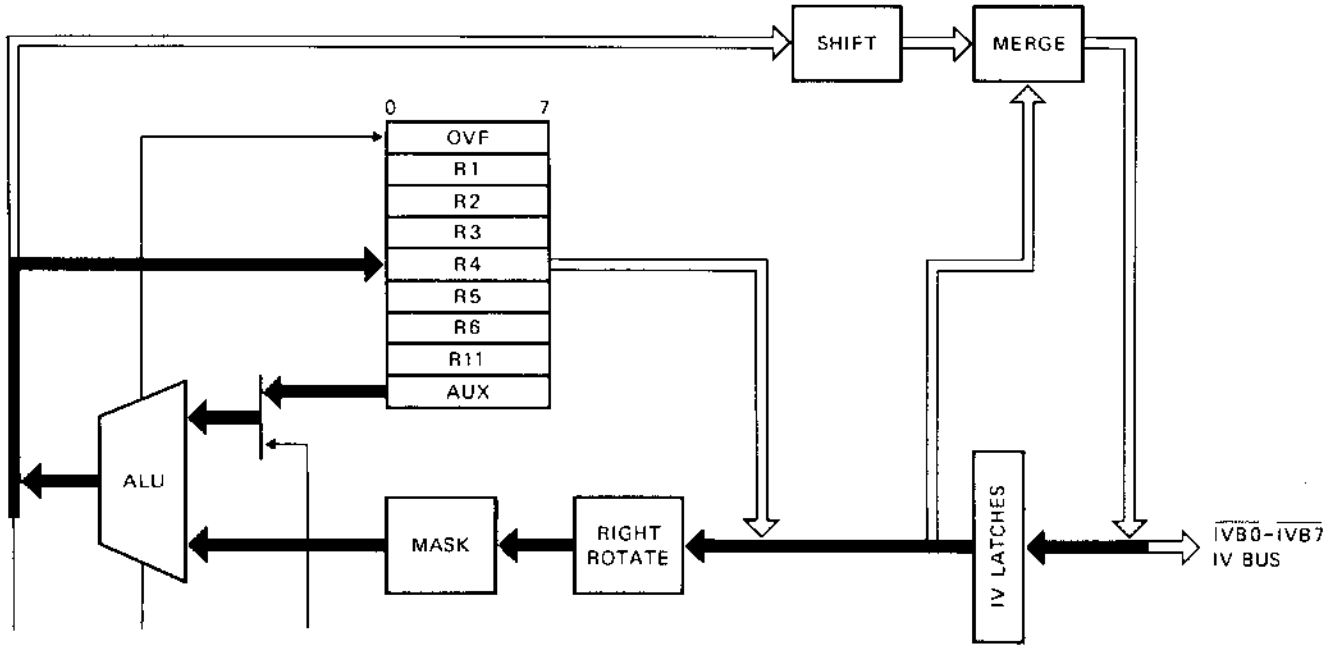
Instruction operation

0 1 0 1 1 0 1 1	IV bus input
0 1 0 1 1 0 1 1	no rotation (S_0 = 7)
0 1 0 1 1 0 1 1	no mask (L = 8 bits)
1 1 0 0 0 1 1 1	contents of AUX
0 1 0 0 0 0 1 1	result of AND
0 1 0 0 0 0 1 1	new contents of R4

Result

R4 contains the result of the AND operation on the contents of the left bank of the IV bus and the AUX register.

Data flow



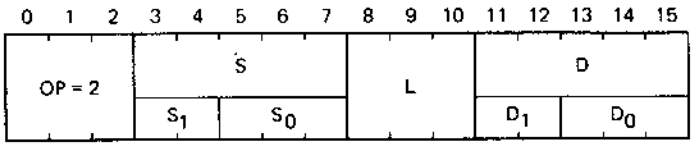
AND Instructions — Op Code 2

AND, IV bus, IV bus

Operation

(S) Λ (AUX) → D

Format



Description

Perform an AND operation on the L-bit field of the IV bus source data and the contents of the AUX register, and move the least significant L bits of the result to the destination field of the IV bus.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 selects the left bank;
S_1 = 3 selects the right bank.

S_0 specifies the bit which will be the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field that is to be processed and merged with the existing IV bus source data. Note that L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
D_1 = 2 selects the left bank;
D_1 = 3 selects the right bank.

D_0 specifies the bit position in the data from the input latches with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that bit 7 is aligned with bit D_0 of the input latches.

The order of operation is:

read the data from the IV bus into the input latches;

right rotate the copied input data as given by S_0;

mask off the least significant L bits;

perform the AND operation on the contents of the AUX register;

left-shift the result as given by D_0;

merge the least significant L bits of the shifted field with the contents of the input latches;

output the merged 8-bit field to the bank of the IV bus given by D_1.

Note that during the merge phase the original values of the bits outside the masked field are preserved. The original data in the destination IV byte is lost.

Operand values

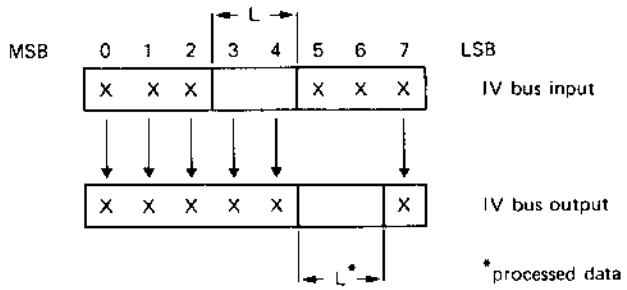
S_0: 0/1/2/3/4/5/6/7

S_1: 2/3

L: 1/2/3/4/5/6/7/0

D_0: 0/1/2/3/4/5/6/7

D_1: 2/3.



AND Instructions — Op Code 2

Example

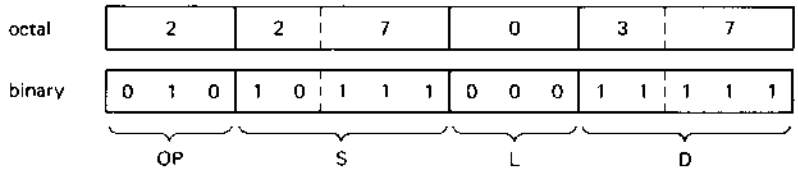
AND, IV bus, IV bus

Mask the most significant 4 bits of the IV bus data at the left bank and move the result to the IV byte at the right bank. (It is assumed that the AUX register has already been loaded with the required contents for this.)

Assembler notation

AND LIV7, 0, RIV7

Instruction word



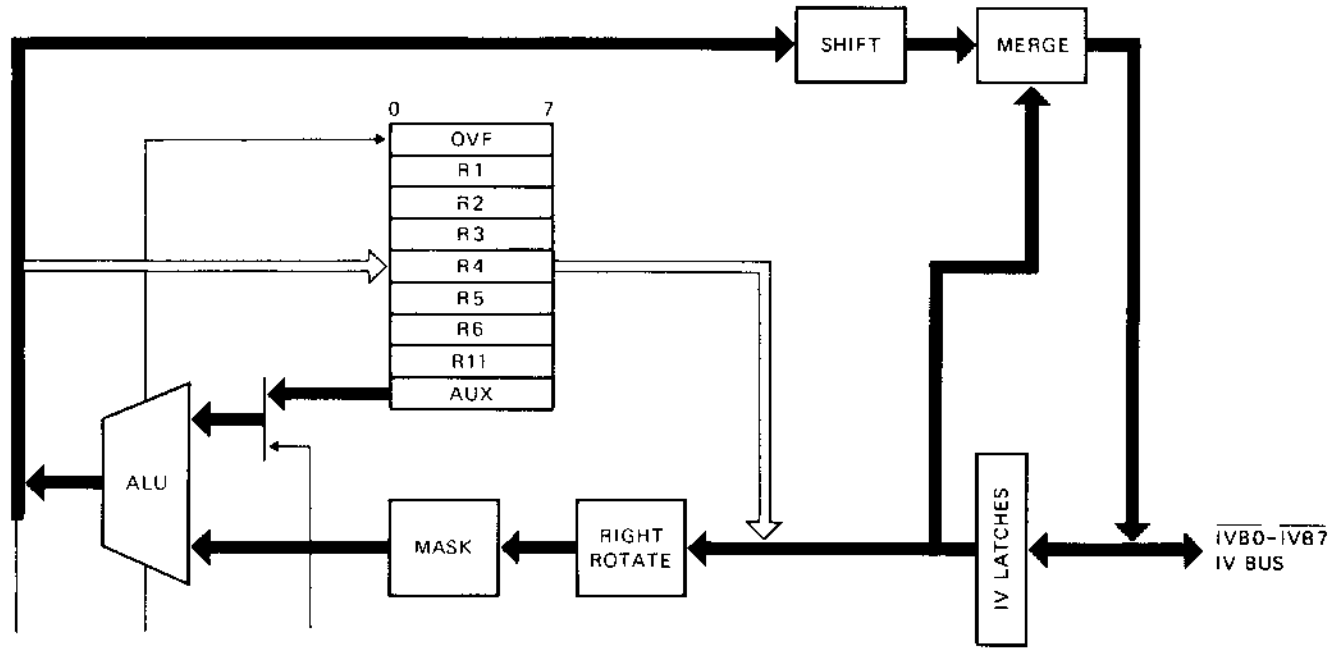
Instruction operation

1 0 1 1 0 1 1 0	IV bus data to input latches
1 0 1 1 0 1 1 0	no rotate or mask (S_0 = 7, L = 0)
1 1 1 1 0 0 0 0	contents of AUX
1 0 1 1 0 0 0 0	result of AND
1 0 1 1 0 0 0 0	no shift (D_0 = 7)
1 0 1 1 0 0 0 0	new IV bus data

Result

The most significant 4 bits of the input data are moved to the IV byte at the right bank.

Data flow



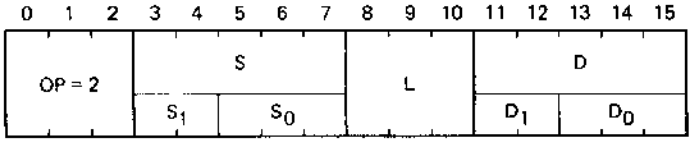
AND Instructions — Op Code 2

AND, IV bus, IV bus address

Operation

(S) Λ (AUX) → D

Format



Description

Enable the IV byte at the bank specified by D, whose address is the result of the AND operation on the L-bit field of the IV bus and the contents of the AUX register.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 selects the left bank;
S_1 = 3 selects the right bank.

S_0 specifies the bit which will be the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field.
Note that L = 0 selects an 8-bit field.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies left bank address (IVL);
D = 17 specifies tight bank address (IVR).

The order of operation is:

read the data from the current IV byte into the input latches;

right rotate the copied input data as given by S_0;

mask off the least significant L bits;

perform the AND operation on the contents of the AND registers;

output the data as an address at the bank specified by D.

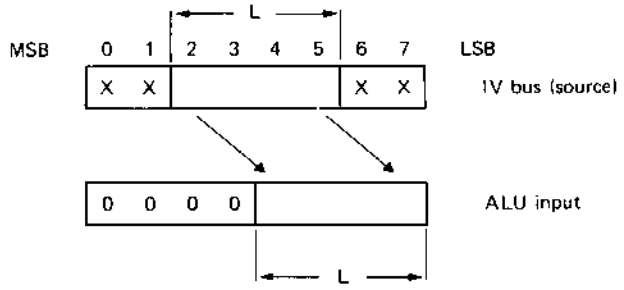
Operand values

S_0: 0/1/2/3/4/5/6/7

S_1: 2/3

L: 1/2/3/4/5/6/7/0

D: 07/17



AND Instructions — Op Code 2

Example

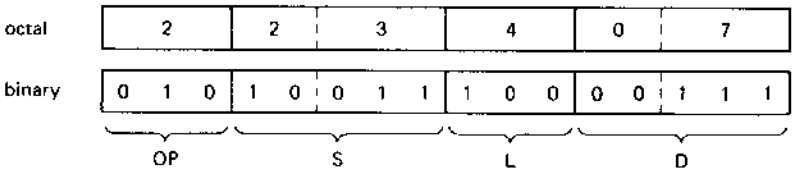
AND, IV bus, IV bus address

Enable the IV byte at the left bank whose address is the result of the AND operation on the contents of the AUX register and bits 0 to 3 of the currently enabled IV byte at the left bank.

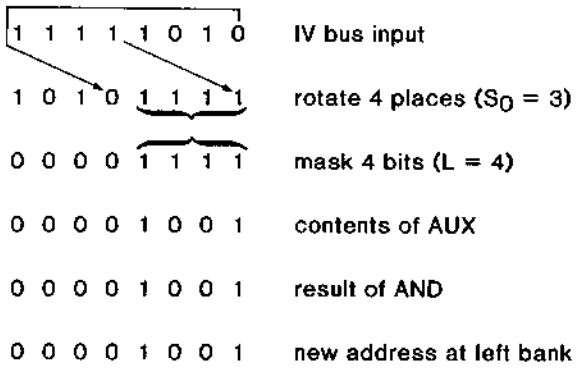
Assembler notation

AND LIV3, 4, IVL

Instruction word



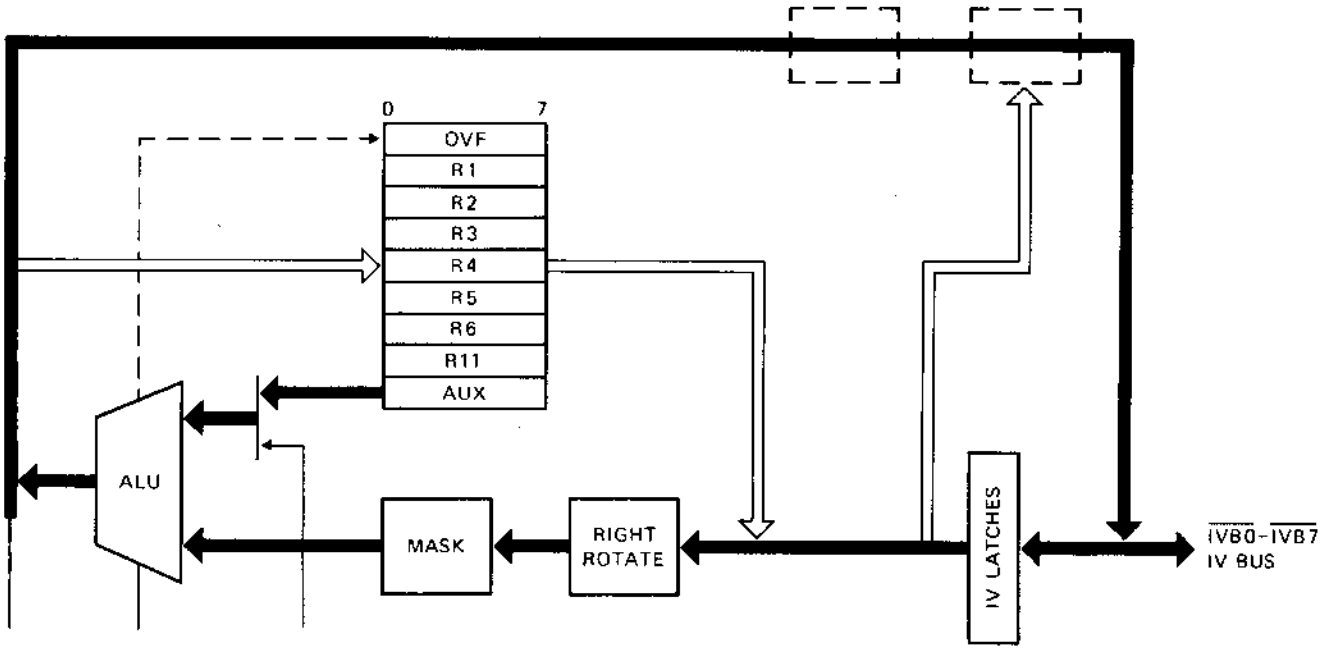
Instruction operation



Result

The previously enabled byte at the left bank is disabled and the byte at address 11 (octal) is enabled.

Data flow



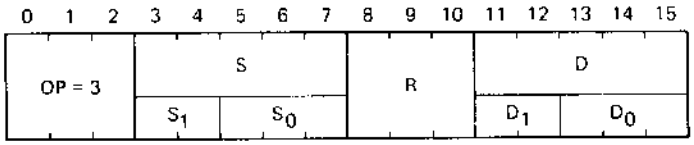
XOR Instructions — Op Code 3

XOR, Register, Register

Operation

(S) ⊕ (AUX) → D

Format



Description

Perform an exclusive OR operation on the right rotated contents of the source register and the contents of the AUX register.

S specifies the source register

R specifies the number of places that the source data is to be rotated.

D specifies the destination register.

The order of operation is:

copy the contents of the source register;

right rotate the copied data;

XOR the right rotated data with the contents of the AUX register;

move the result to the destination register.

The contents of the source and AUX registers remains unchanged after the instruction unless one of these is also specified as the destination register.

Operand values

S: 00/01/02/03/04/05/06/10/11

R: 0/1/2/3/4/5/6/7

D: 00/01/02/03/04/05/06/11

XOR Instructions — Op Code 3

Example

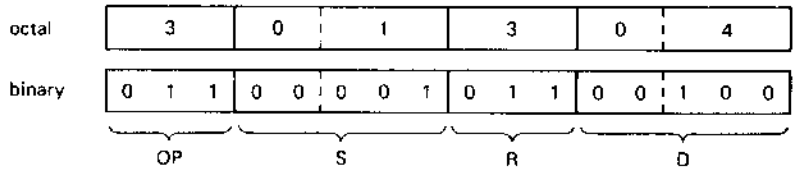
XOR, Register, Register

Perform an exclusive OR operation on the contents of R1, rotated 3 places, and the contents of the AUX register. Store the result in R4.

Assembler notation

XOR R1 (3), R4

Instruction word



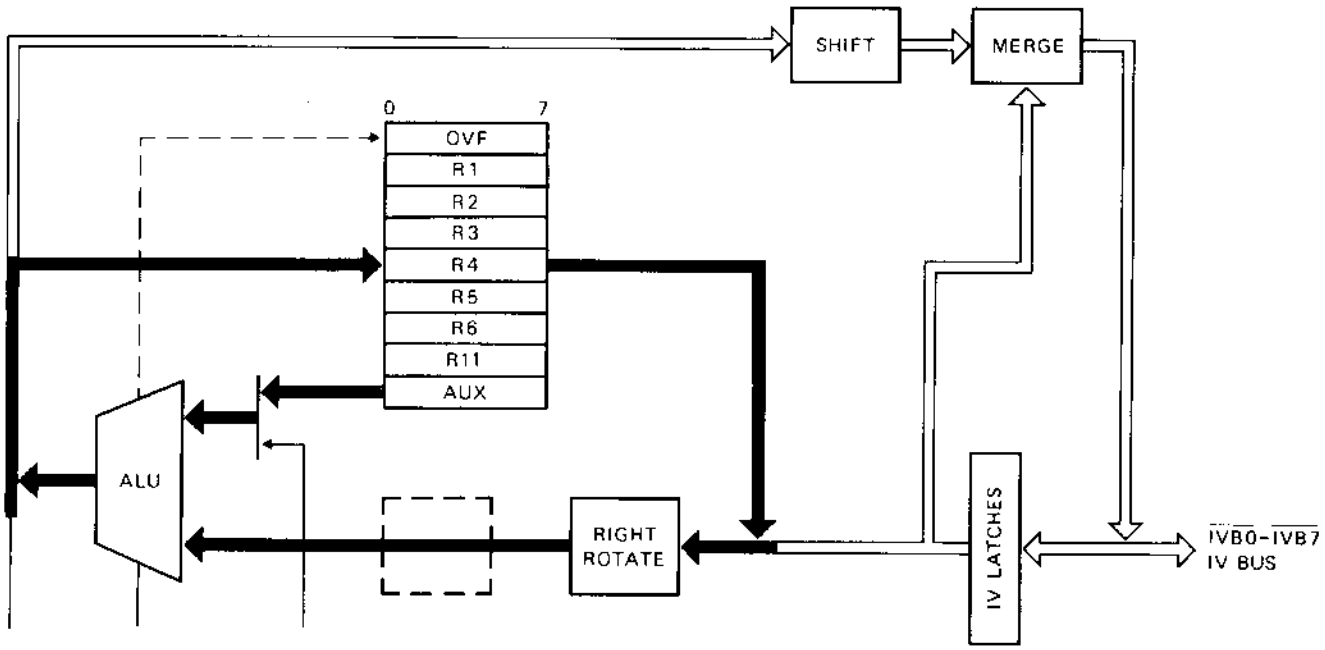
Instruction operation

1 0 0 1 0 0 1 0	copy source register R1 (S = 01)
0 1 0 1 0 0 1 0	rotate 3 places (R = 3)
1 1 1 1 1 1 1 1	contents of AUX
1 0 1 0 1 1 0 1	result of XOR
1 0 1 0 1 1 0 1	new contents of R4

Result

Register R4 holds the result of the XOR operation on the rotated contents of R1 and the contents of AUX.

Data flow



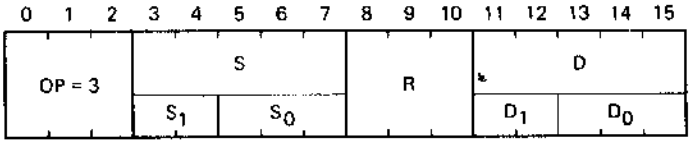
XOR Instructions — Op Code 3

XOR, Register, IV bus address

Operation

Enable the IV byte with address (S) ⊕ (AUX).

Format



Description

Enable the IV byte, at the bank specified by D, whose address is the result of the XOR operation on the right rotated contents of the source register and the contents of the AUX register.

S specifies the source register.

R specifies the number of places that the source data is to be rotated.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies left bank address (IVL);
D = 17 specifies right bank address (IVR).

The order of operation is:

rotate the copied contents of register S by R places;

XOR the rotated data field to the contents of AUX;

output the result to the IV bus as an address.

The contents of the source register remain unchanged after the instruction.

Operand values

S: 00/01/02/03/04/05/06/10/11

R: 0/1/2/3/4/5/6/7

D: 07/17

XOR Instructions — Op Code 3

Example

XOR, Register, IV bus address

Enable the IV byte at the right bank whose address is the result of the XOR operation on the contents of R3 and the AUX register.

Instruction word

octal	3			0			3			0			1		7
binary	0	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1
	OP			S					R				D		

Assembler notation

XOR R3 (0), IVR

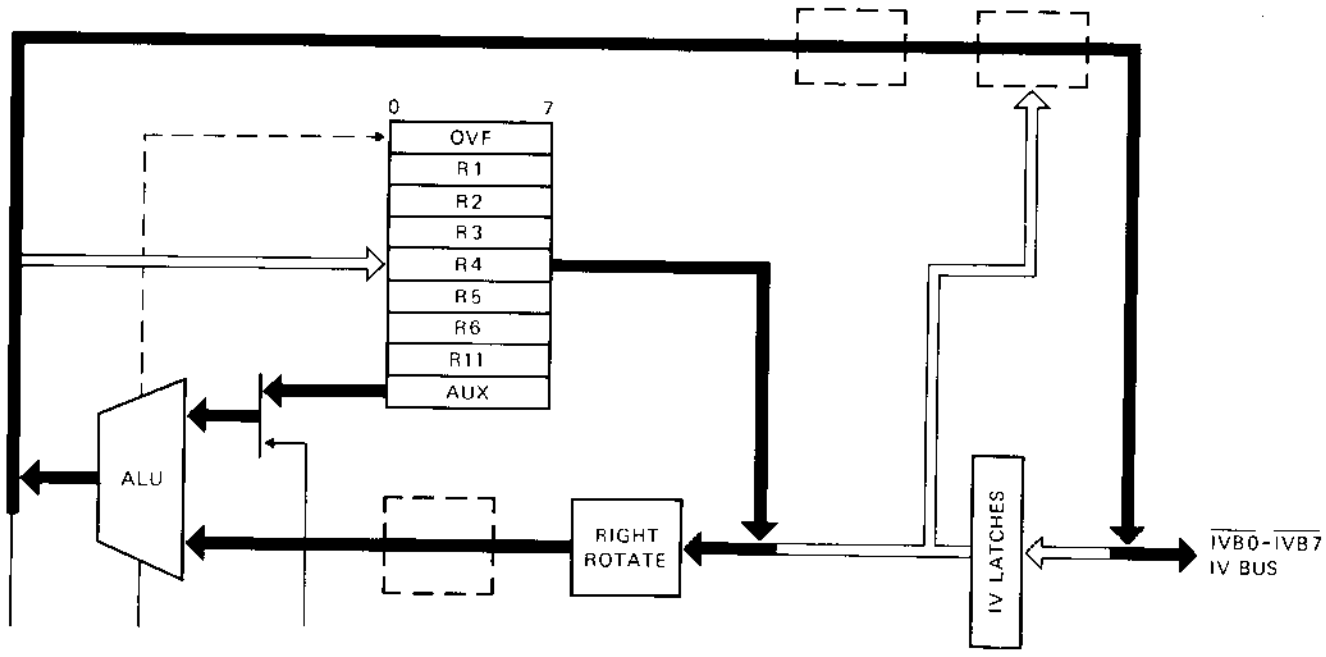
Instruction operation

1 0 1 1 0 1 1 1	copy source register
1 0 1 1 0 1 1 1	no rotate (R = 0)
0 1 1 0 1 1 0 1	contents of AUX
1 1 0 1 1 0 1 0	result of XOR
1 1 0 1 1 0 1 0	new right bank address

Result

The previously enabled IV byte at the right bank is disabled and the IV byte at address 332 (octal) is enabled. The source and AUX registers remain unchanged.

Data flow



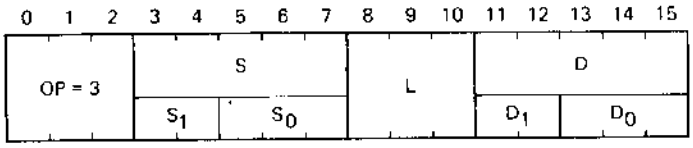
XOR Instructions — Op Code 3

XOR, Register, IV bus

Operation

(S) ⊕ (AUX) →D

Format



Description

Perform an exclusive OR operation on the contents of the source register and the contents of the AUX register. Move the least significant L bits of the result to the L-bit field of the IV bus.

S specifies the source register.

L specifies the length (number of bits) of the masked field that is to be merged with the existing IV byte data.
Note that L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
D_1 = 2 selects the left bank;
D_1 = 3 selects the right bank.

D_0 specifies the bit position in the IV byte with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that bit 7 is aligned with bit D_0 of the IV bus.

The order of operation is:

read the data of the destination IV byte into the input latches;

copy the contents of the source register and perform an XOR operation on the contents of the AUX register;

left-shift the result as specified by D_0;

merge the least significant L-bits of the shifted field with the data in the input latches;

output the merged data to the IV bus.

Note that the bits of the output data field outside the L-bit masked field retain their original values. The contents of the source register remain unchanged after the instruction.

Operand values

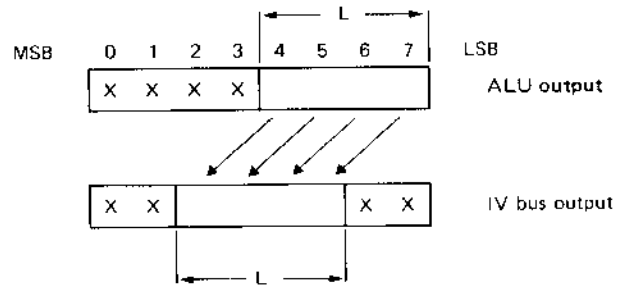
S: 00/01/02/03/04/05/06/10/11

L: 1/2/3/4/5/6/7/0

D_0: 0/1/2/3/4/5/6/7

D_1: 2/3

Note: L = 0 selects an 8-bit field.



XOR Instructions — Op Code 3

Example

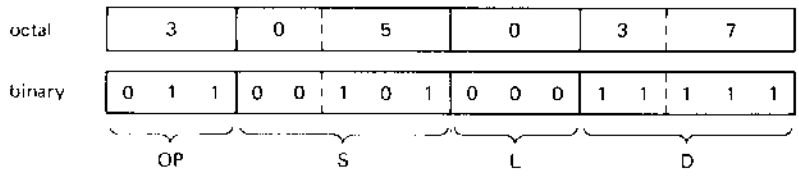
XOR, Register, IV bus

Store the one's complement of the contents of R5 in the IV byte at the right bank. (It is assumed that the AUX register already contains all ones.)

Instruction word

Assembler notation

XOR R5, 0, RIV7



Instruction operation

0 1 1 0 0 1 1 1 copy source register

1 1 1 1 1 1 1 1 contents of AUX

1 0 0 1 1 0 0 0 result of XOR

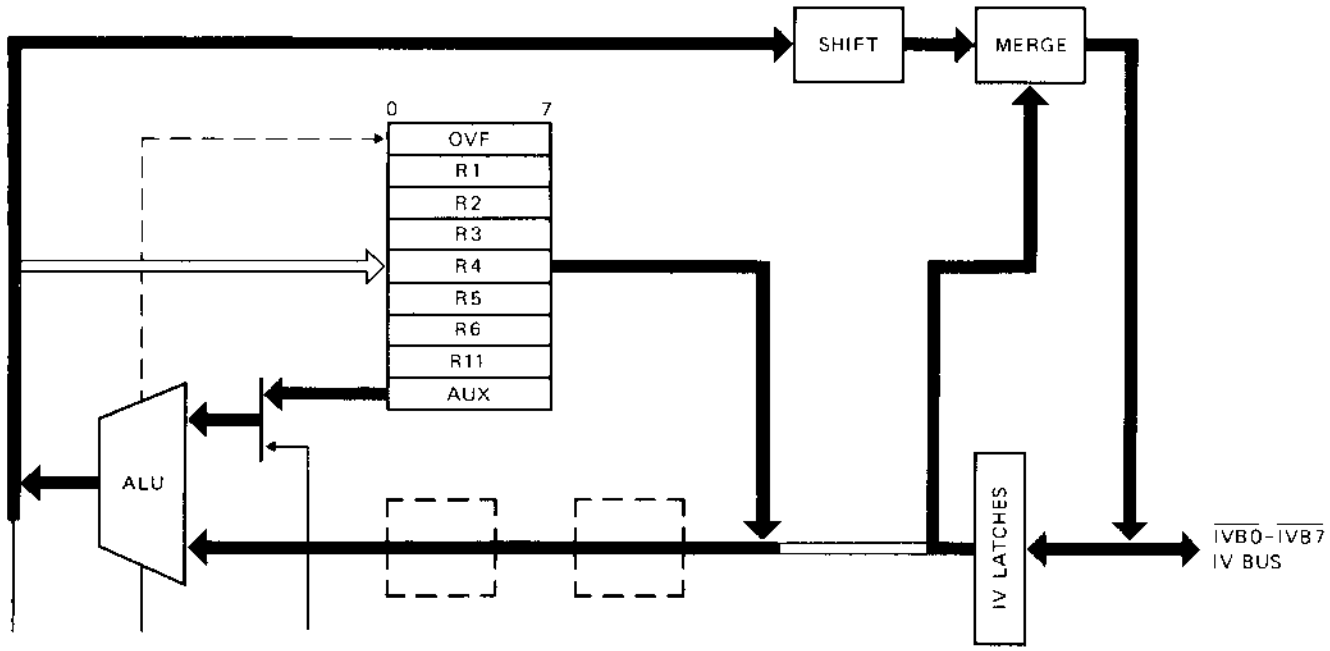
1 0 0 1 1 0 0 0 no shift (D_0 = 7)

1 0 0 1 1 0 0 0 data to IV bus

Result

The one's complement of the source register is output to the right bank of the IV bus.

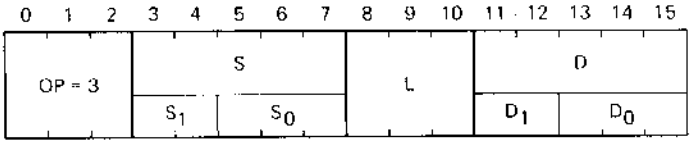
Data flow



XOR Instructions — Op Code 3

XOR, IV bus, Register

Format



Operation

(S) ⊕ (AUX) → D

Description

Perform an exclusive OR operation on the L-bit field of the IV bus source data and the contents of the AUX register. Move the 8-bit result to the register specified by D.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 selects the left bank;
S_1 = 3 selects the right bank.

S_0 specifies the bit which will be the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field.
Note that L = 0 selects an 8-bit field.

D specifies the address of the destination register.

The order of operation is:

read the IV bus data into the input latches;

right rotate the input field as specified by S_0;

mask the rotated data field as specified by L;

XOR the masked data with the contents of the AUX register;

move the 8-bit result to the destination register.

Operand values

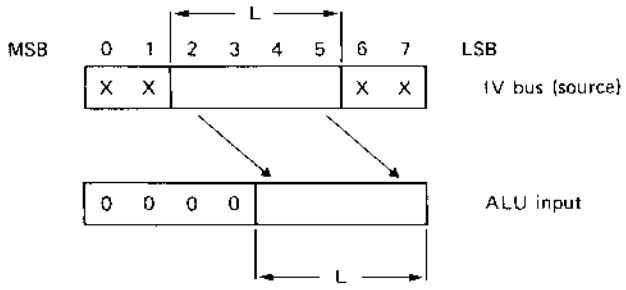
S_0: 2/3

S_1: 0/1/2/3/4/5/6/7

L: 1/2/3/4/5/6/7/0

D: 00/01/02/03/04/05/06/11

Note that L = 0 selects an 8-bit field.



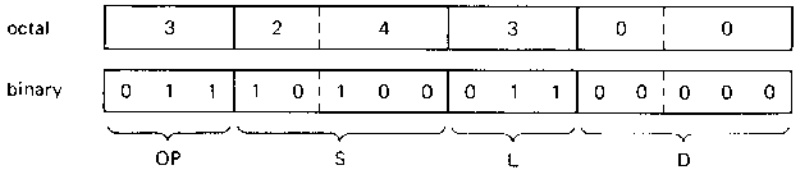
XOR Instructions — Op Code 3

Example

XOR, IV bus, Register

Perform an exclusive OR operation on the contents of bits 2, 3 and 4 of the IV byte at the left bank and the contents of the AUX register. Store the result in the AUX register.

Instruction word



Assembler notation

XOR LIV4, 3, AUX

Instruction operation

1 0 1 1 1 0 1 1 IV bus input

0 1 1 1 0 1 1 1 rotate 3 places (S_0 = 4)

0 0 0 0 0 1 1 1 mask 3 bits (L = 3)

0 0 0 0 0 1 1 1 contents of AUX

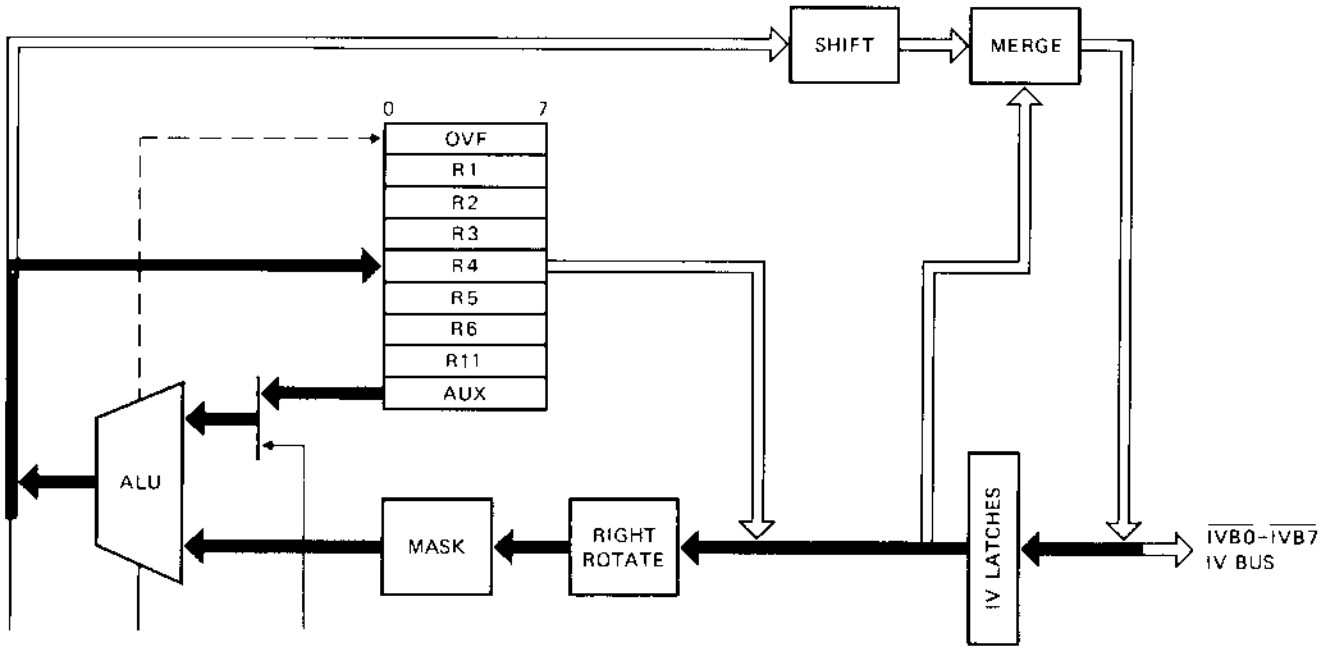
0 0 0 0 0 0 0 0 result of XOR

0 0 0 0 0 0 0 0 new contents of AUX

Result

The contents of the AUX register are changed to the result of the XOR operation. The source IV byte remains unchanged.

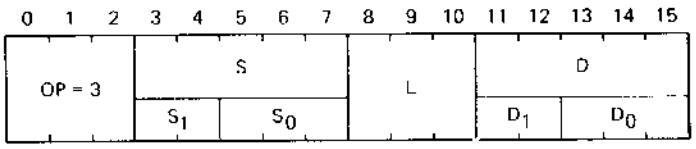
Data flow



XOR Instructions — Op Code 3

XOR, IV bus, IV bus

Format



Operation

(S) ⊕ (AUX) → D

Description

Perform an exclusive OR operation on the L-bit field of the IV bus source data and the 8-bit contents of the AUX register and move the least significant L bits of the result to the destination field of the IV bus, given by D.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 specifies the left bank;
S_1 = 3 specifies the right bank.

S_0 specifies the bit which will be the least significant bit of the input data field after rotation.

L specifies the length (number of bits) of the masked field that is to be processed and merged with the existing IV bus source data.
Note that L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
D_1 = 2 specifies the left bank;
D_1 = 3 specifies the right bank.

D_0 specifies the bit position in the data from the input latches with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that bit 7 is aligned with bit D_0 of the input latches.

The order of operation is:

read the IV bus data into the input latches;

right rotate the input data field until bit S_0 becomes the LSB;

mask the least significant L bits;

XOR the masked field with the contents of the AUX register;

left-shift the result until bit 7 is aligned with bit D_0;

merge the least L bits with the original IV bus data from the input latches;

output the merged 8-bit field to the IV bus.

Note that during the merge phase, the original values of the bits outside the masked field are preserved. The original data in the destination IV byte is lost.

Permitted operand values

S_0: 0/1/2/3/4/5/6/7

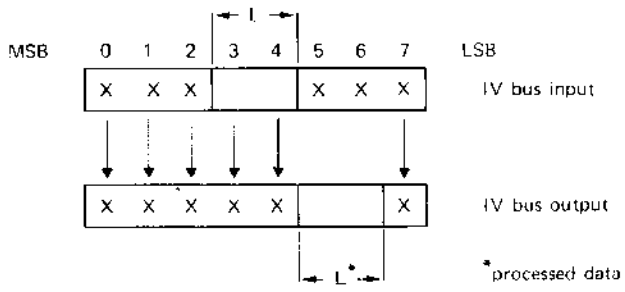
S_1: 2/3

L: 1/2/3/4/5/6/7/0

D_0: 0/1/2/3/4/5/6/7

D_1: 2/3

Note that L = 0 selects an 8-bit field.



XOR Instruction — Op Code 3

XOR, IV bus, IV bus

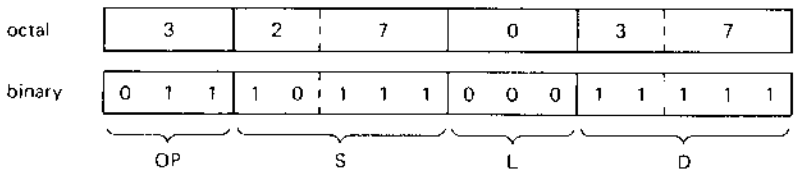
Example

Perform exclusive OR operation on the contents of the AUX register and the contents of the IV byte at the left bank and output the result to the IV byte at the right bank.

Assembler notation

XOR, LIV7, 0, RIV7

Instruction word



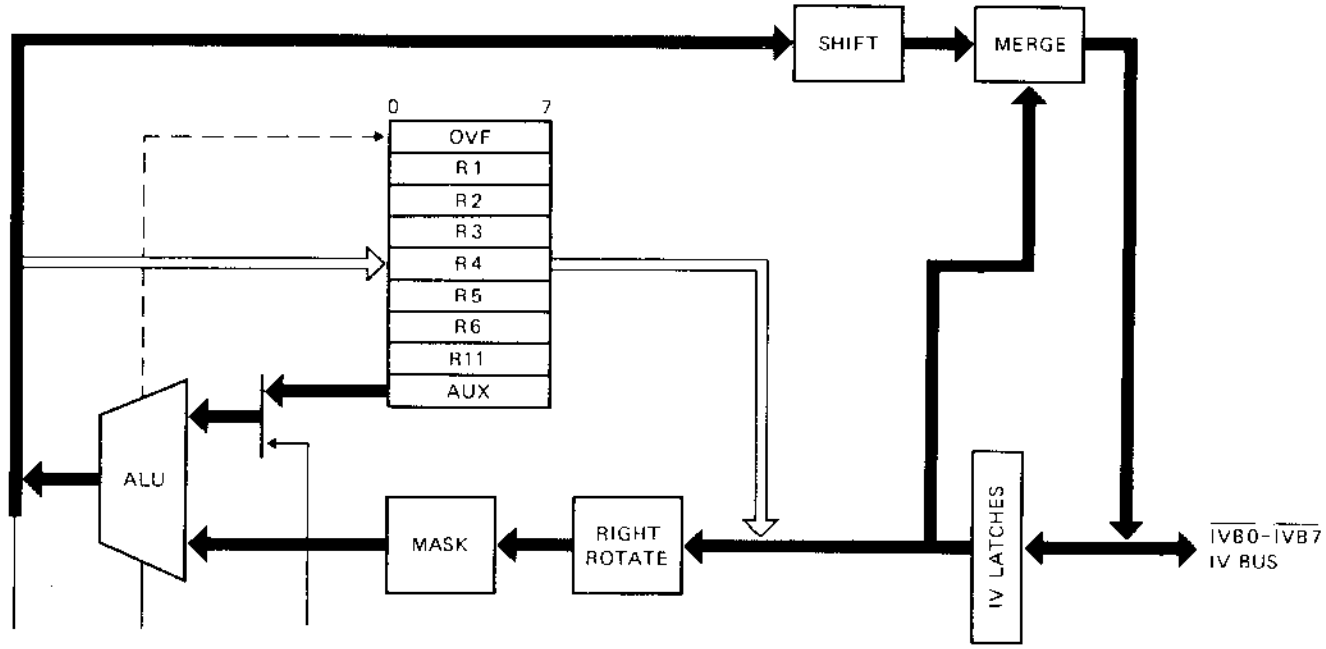
Instruction operation

1 0 1 1 1 0 1 1	IV bus input
1 0 1 1 1 0 1 1	no rotate (S_0 = 7)
1 0 1 1 1 0 1 1	no mask (L = 0)
0 0 1 0 0 1 0 1	contents of AUX
1 0 0 1 1 1 1 0	result of XOR
1 0 0 1 1 1 1 0	no shift (D_0 = 7)
1 0 0 1 1 1 1 0	new IV bus data

Result

The IV byte at the right bank contains the results of the exclusive OR operation on the contents of the AUX register and the IV byte at the left bank.

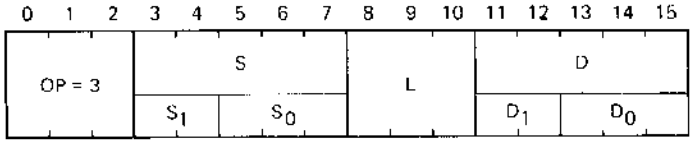
Data flow



XOR Instructions — Op Code 3

XOR, IV bus, IV bus address

Format



Operation

(S) ⊕ (AUX) → D

Description

Enable the IV byte, at the bank specified by D, whose address is the result of the XOR operation on the L-bit field of the IV bus and the contents of the AUX register.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 specifies the left bank;
S_1 = 3 specifies the right bank.

S_0 specifies the bit which will be the least significant bit of the input data field after rotation.

L specifies the length (number of bits) of the mask field.
Note that L = 0 selects an 8-bit field.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies left bank address (IVL);
D = 17 specifies right bank address (IVR).

The order of operation is:

read the IV bus data into the input latches;

right rotate the input data field until bit S_0 becomes the LSB;

mask the least significant L bits;

XOR the masked field with the contents of the AUX register;

move the resulting 8-bit field to the IV bus as an address at the bank specified by D.

Permitted operand values

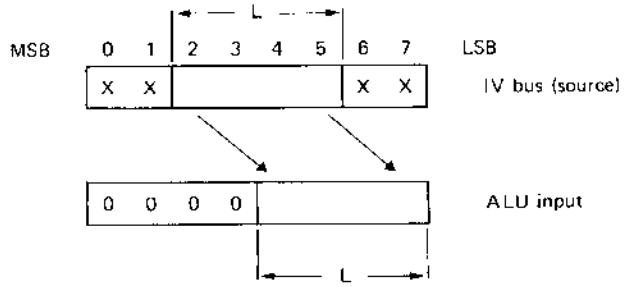
S_0: 0/1/2/3/4/5/6/7

S_1: 2/3

L: 1/2/3/4/5/6/7/0

D: 07/17

Note that L = 0 selects an 8-bit field.



XOR Instructions — Op Code 3

XOR, IV bus, IV bus address

Example

Enable the IV byte at the right bank whose address is the result of the XOR operation on the contents of the AUX register and the least significant 4 bits of the IV byte at the left bank.

Instruction word

octal	3			2			7			4			1		7
binary	0	1	1	1	0	1	1	1	1	0	0	0	1	1	1
	OP			S						L			D		

Assembler notation

XOR LIV7, 4, IVR

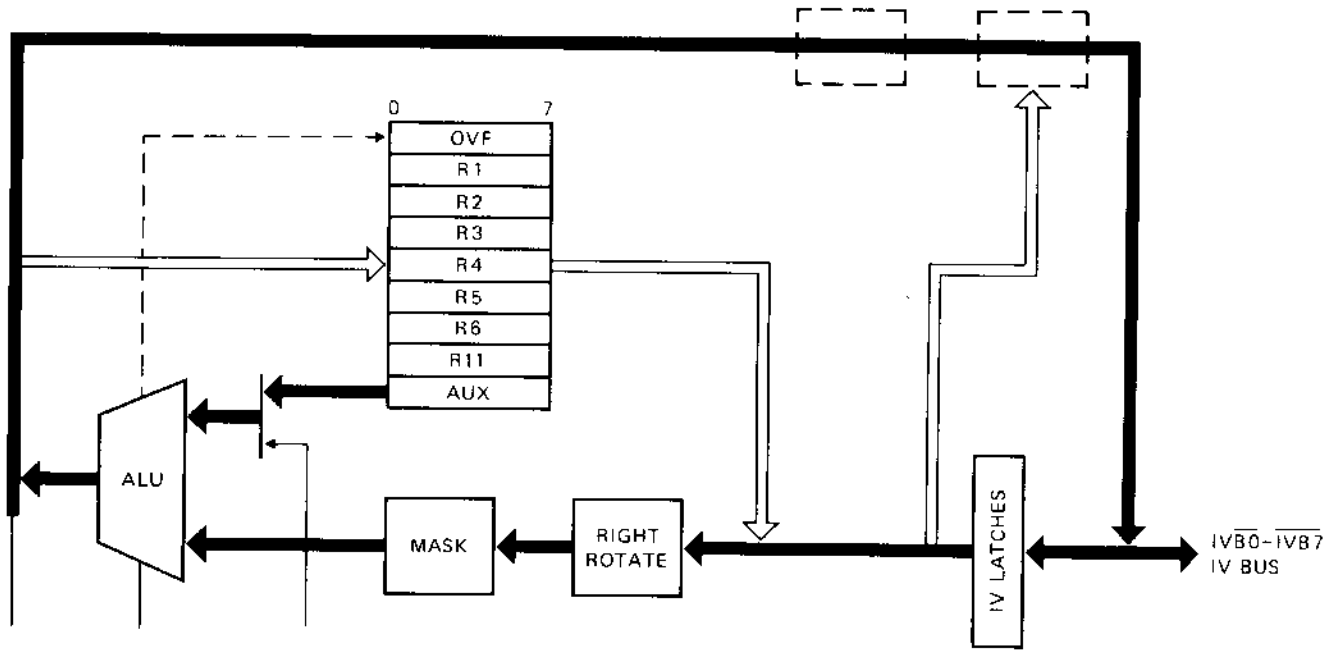
Instruction operation

1 0 1 1 0 1 1 0	IV bus input
1 0 1 1 0 1 1 0	no rotate (S_0 = 7)
0 0 0 0 0 1 1 0	mask 4 bits (L = 4)
0 0 0 0 1 1 1 0	contents of AUX
0 0 0 0 1 0 0 0	result of XOR
0 0 0 0 1 0 0 0	new IV bus right bank address

Result

The previously enabled byte at the right bank is disabled and the byte at address 10 (octal) at the right bank is enabled.

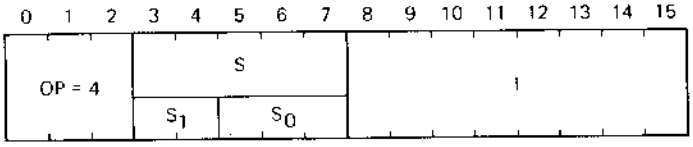
Data flow



XEC Instructions — Op Code 4

XEC, Register

Format



Operation

Execute the instruction at the current page address offset by I + (S). Return to the instruction following the XEC instruction unless an unconditional jump or a satisfied conditional jump is encountered.

Description

Execute the instruction at the address formed by replacing the 8 least significant bits of the contents of the address register with the 8-bit sum of I and the contents of the register specified by S.

S specifies the source register.

I is the 8-bit integer value for address modification.

The order of operation is:

copy the data from the source register;

form the 8-bit sum of the I field value and the source register contents;

modify the address register with the 8-bit sum.

Only the least significant 8-bits of the address register can be changed by this instruction, so that a range of 256 addresses is available. This range of 256 addresses is termed the address page, determined by the five most significant bits of the address register. When the sum of (S) + I is greater than 255 (377 octal) only the least significant 8 bits are used; the overflow register is not changed.

The program counter is not altered by the XEC instruction, so that the original address within the page is retained. During the instruction to be executed, the program counter is incremented by one in the normal way to point to the instruction following the XEC instruction. However, if the executed instruction is a JMP or NZT, the program counter can be changed to the jump address and instruction execution does not return to the address following the XEC instruction.

Permitted operand values

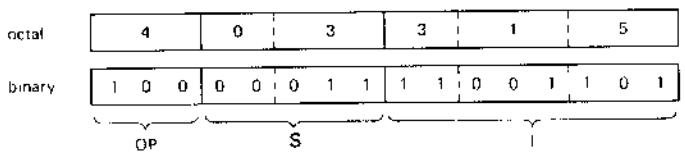
S: 00/01/02/03/04/05/06/10/11

I: $0 \leq I \leq 377_8$

Example

Execute the instruction whose address is given by replacing the least significant 8 bits of the contents of R3 and the octal integer 315.

Instruction word



Instruction operation

Initial value of address register: 710_8 (00001 11001000)
Initial value of program counter: 710_8

I-field	1 1 0 0 1 1 0 1	+
copy contents of R3	1 0 0 0 0 0 1 0	
8-bit sum of R3 and I becomes 8 LSBs of address register	0 1 0 0 1 1 1 1 1 1 7	

New value of address register: 517_8 (00001 01001111)
The program counter and R3 are unchanged.

Assembler notation

XEC 315 (R3)

XEC Instruction — Op Code 4

Result

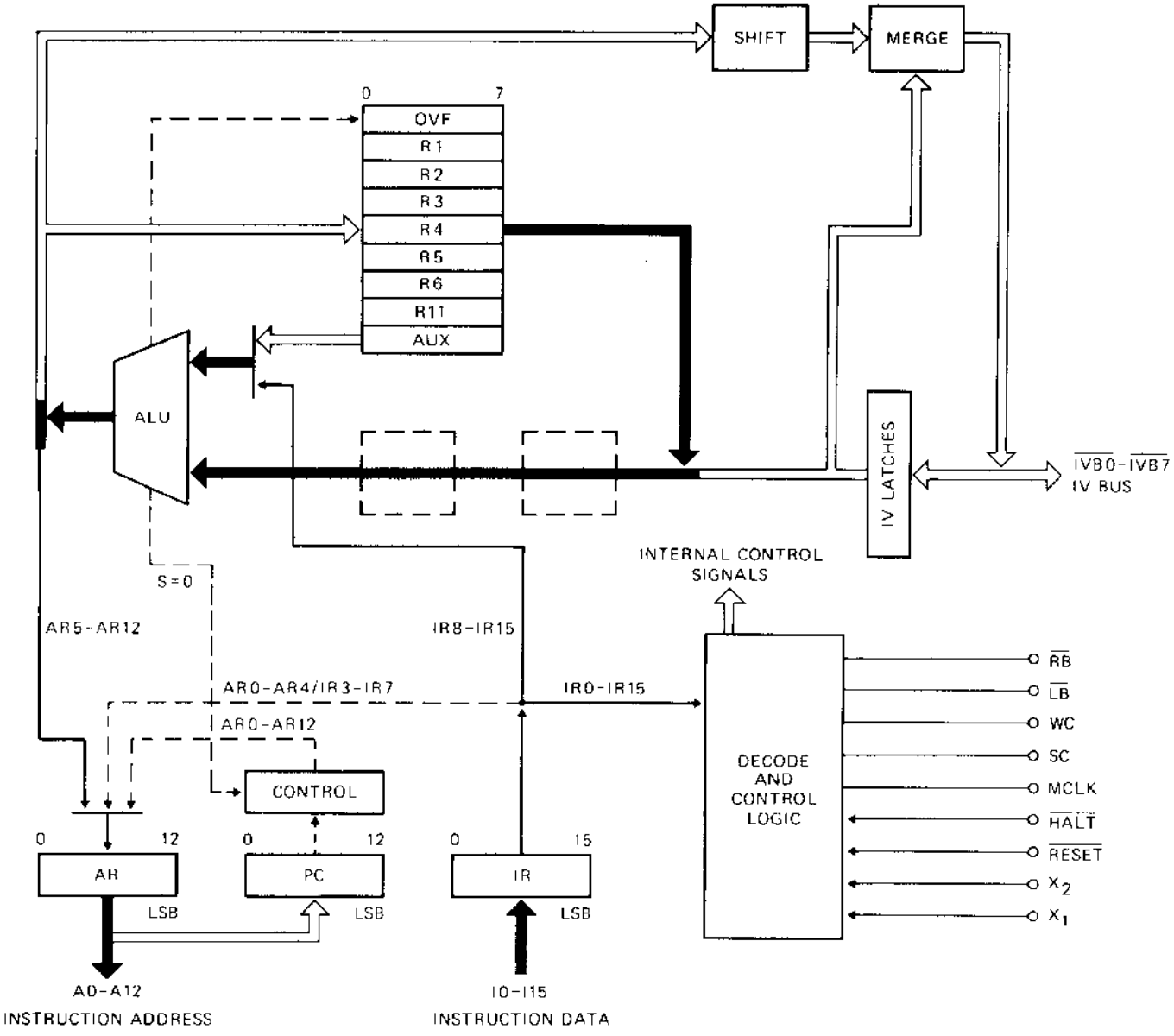
XEC, Register

The value in the address register is changed to 517_8, so that the next instruction to be executed is the one at address 517.

The sequence of instructions executed depends upon the presence of a JMP or NZT instruction:

(a) with jump		(b) without jump	
Address		Address	
710	XEC instruction	710	XEC instruction
517	Instruction to be executed (jump to address 355)	517	instruction to be executed
		711	next instruction
355	next instruction		

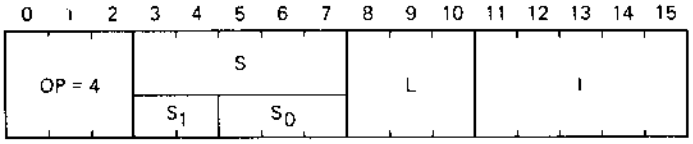
Data flow



XEC Instructions — Op Code 4

XEC, IV bus

Format



Operation

Execute the instruction at the current page address offset by I + (S). Return to the instruction following the XEC instruction unless an unconditional jump or a satisfied conditional is encountered.

Description

Execute the instruction at the address formed by replacing the 5 least significant bits of the contents of the address register with the 5-bit sum of I and the contents of the IV bus field specified by S.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 specifies the left bank;
S_1 = 3 specifies the right bank.

S_0 specifies the bit which will be the least significant bit of the input data field after rotation.

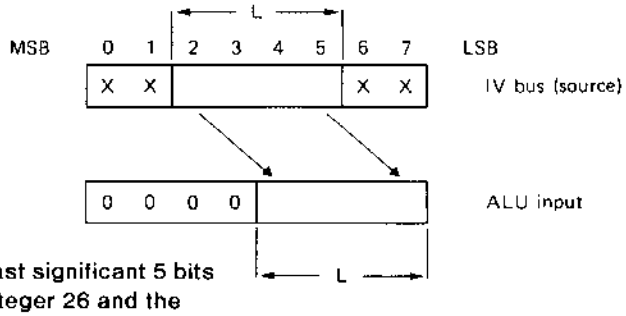
I is the 5-bit integer value for address modification.

L specifies the length (number of bits) of the masked field. The maximum value of L that may be specified is L = 5.

The order of operation is:
read the IV bus data into the input latches;
rotate the input data field as given by S_0;
mask off the least significant L bits;
add the masked field to 5-bit integer;
replace the least significant 5-bits of the contents of the address register with the 5-bit result of the add operation.

Only the least significant 5 bits of the address register can be changed by this instruction, so that a range of 32 addresses is available. This range of 32 addresses is termed the address page, determined by the eight most significant bits of the address register. When the sum (S) + I is greater than 31 (37 octal) only the least significant 5 bits are used; the overflow register is not changed.

The program counter is not altered by the XEC instruction, so that the original address within the page is retained. During the instruction to be executed, the program counter is incremented by one in the normal way to point to the instruction following the XEC instruction. However, if the executed instruction is a JMP or NZT, the program counter can be changed to the jump address and instruction execution does not return to the address following the XEC instruction.



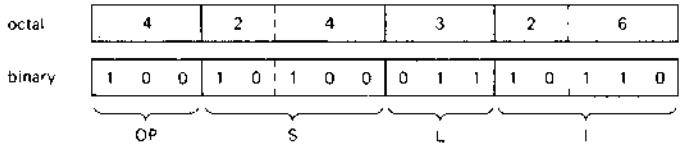
Permitted operand values

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5
I: $0 \leq I \leq 37_8$

Example

Execute the instruction whose address is given by replacing the least significant 5 bits of the contents of the address register with the sum of the octal integer 26 and the contents of bits 2,3 and 4 of the IV byte at the left bank.

Instruction word



Assembler notation

XEC 26H (LIV4,3)

XEC Instructions — Op Code 4

XEC, IV bus

Instruction operation

Initial value of address register: 55_8 (00000001 01101)

IV bus input	1 1 0 1 1 1 0 1	
rotate 3 places	1 0 1 1 1 0 1 1	
mask 3 bits (L = 3)	0 0 0 0 0 0 1 1	
		+
value of I-field	0 0 0 1 0 1 1 0	
5-bit sum to 5 LSBs of address register	1 1 0 0 1	

New value of address register: 71_8 (00000001 11001)

(a) with jump

Address	
55	XEC instruction
71	instruction to be executed (jump to address 150)
150	next instruction

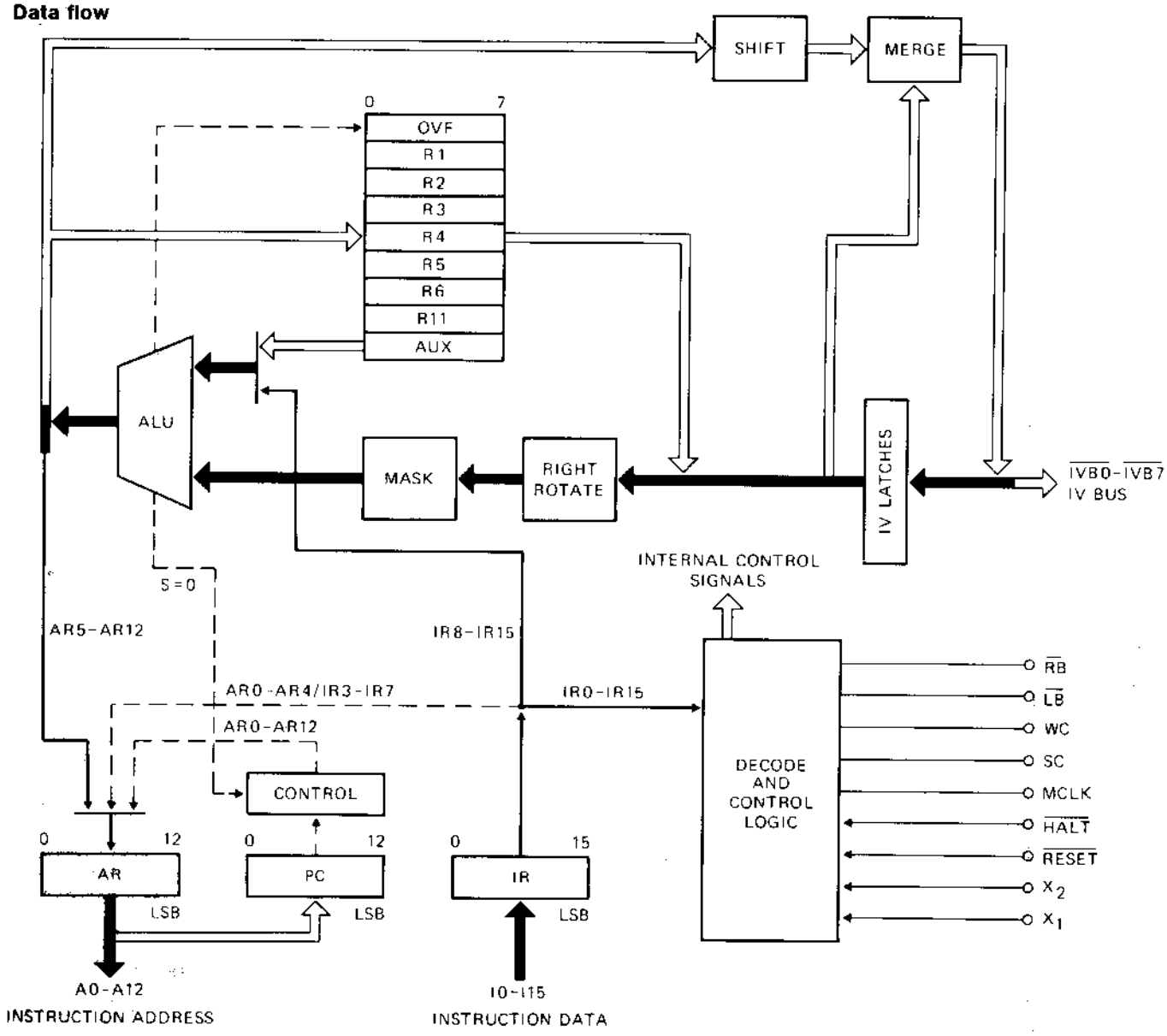
(b) without jump

Address	
55	XEC instruction
71	instruction to be executed
56	next instruction

Result

The value in the address register is changed to 71_8 so that the next instruction to be executed is at address 71. The sequence of instructions executed depends upon the presence of a JMP or NZT instruction:

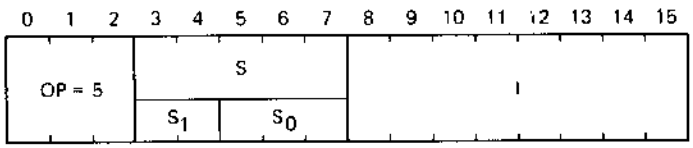
Data flow



NZT Instructions — Op Code 5

NZT, Register

Format



Operation

Jump if (S) ≠ 0

Description

If (S) ≠ 0, jump to the address formed by replacing the 8 least significant bits of the contents of the address register and program counter with the value in the I field.

If (S) = 0, increment the program counter by one.

S specifies the register which is the subject of the test.

I is the 8-bit integer for address modification.

The order of operation is:

copy the contents of the source register;

test the register contents for all zeros;

if the contents are not zeros, replace the least significant 8 bits of the contents of the address register and program counter with the value of the I-field;

if the contents are zeros, increment the program counter by one.

Permitted operand values

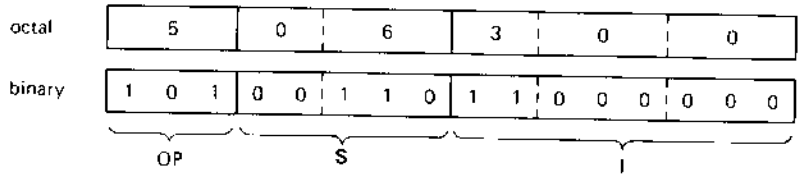
S: 00/01/02/03/04/05/06/10/11

I: $0 \leq I \leq 377_8$

Example

Jump to address 5300_8 if the content of R6 is not zero.

Instruction word



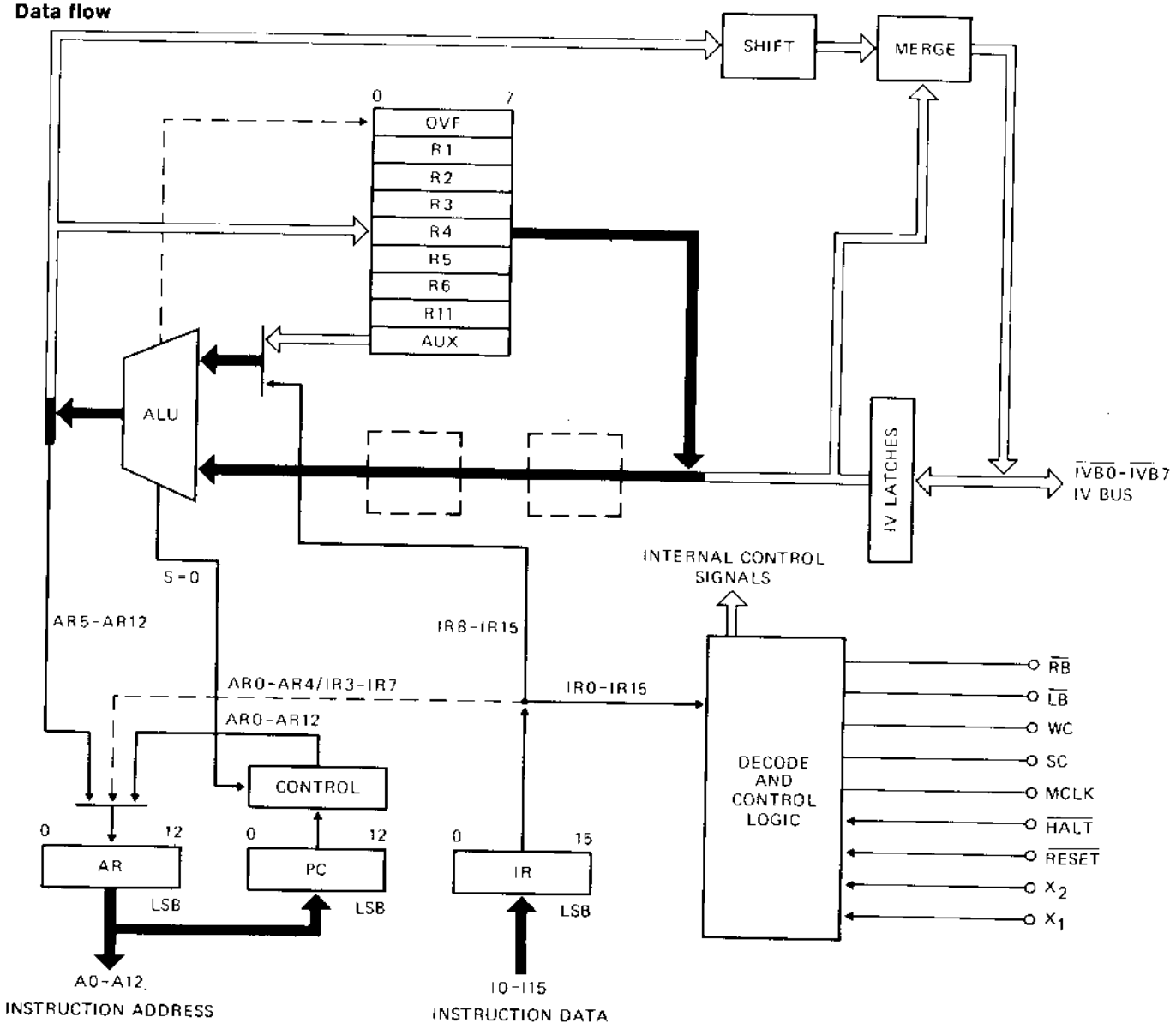
Instruction operation

Initial contents of address register and program counter — 01010 11000011 (5303_8)
contents of I-field — 11000000 (300_8)
new contents of address register and program counter if (R6) ≠ 0 — 01010 11000000 (5300_8)
new contents of address counter and program register if (R6) = 0 — 01010 11000100 (5304_8)

Result

If the contents of R6 are non-zero, the program branches to address 5300, otherwise it continues at address 5304.

Data flow



NZT Instructions — Op Code 5

NZT, Register

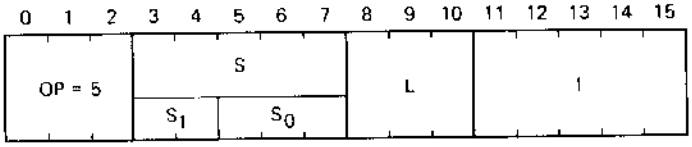
Assembler notation

NZT R6, 300H

NZT Instructions — Op Code 5

NZT, IV bus

Format



Operation

If (S) ≠ 0, jump to the address formed by replacing the 5 least significant bits of the contents of the address register and program counter with the value in the I-field.

Description

If the contents of the L-bit field of the IV bus source data is non-zero, insert the value of the 5-bit I-field into the 5 least significant bits of the address register and program counter. If the contents are all zeros, the program counter is incremented by one.

S_1 specifies the bank of the IV bus which is the data source:
S_1 = 2 specifies the left bank;
S_1 = 3 specifies the right bank.

S_0 specifies the bit which will be the least significant bit of the input data field after rotation.

I is the 5-bit integer value for address modification.

L specifies the length (number of bits) of the masked field.
Note that L = 0 specifies an 8-bit field.

The order of operation is:

read the IV bus data into the input latches;

rotate the copied input data until bit S_0 becomes the LSB;

mask off the least significant L bits;

test the contents of the masked field;

if the contents of the masked field are non-zero, replace the 5 least significant bits of the program counter and address register with the value of the I-field;

if the contents of the masked field are zero, increment the program counter by one.

Permitted operand values

S_0: 0/1/2/3/4/5/6/7

S_1: 2/3

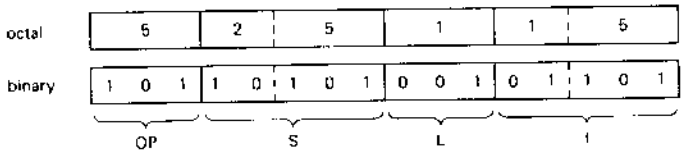
L: 1/2/3/4/5/6/7/0

I: $0 \leq I \leq 37_8$

EXAMPLE

Jump to address 115_8 if the content of bit 5 of the IV byte at the left bank is not zero.

Instruction word



Assembler notation

NZT LIV5, 1, 15H

NZT Instructions — Op Code 5

NZT, IV bus

Instruction operation

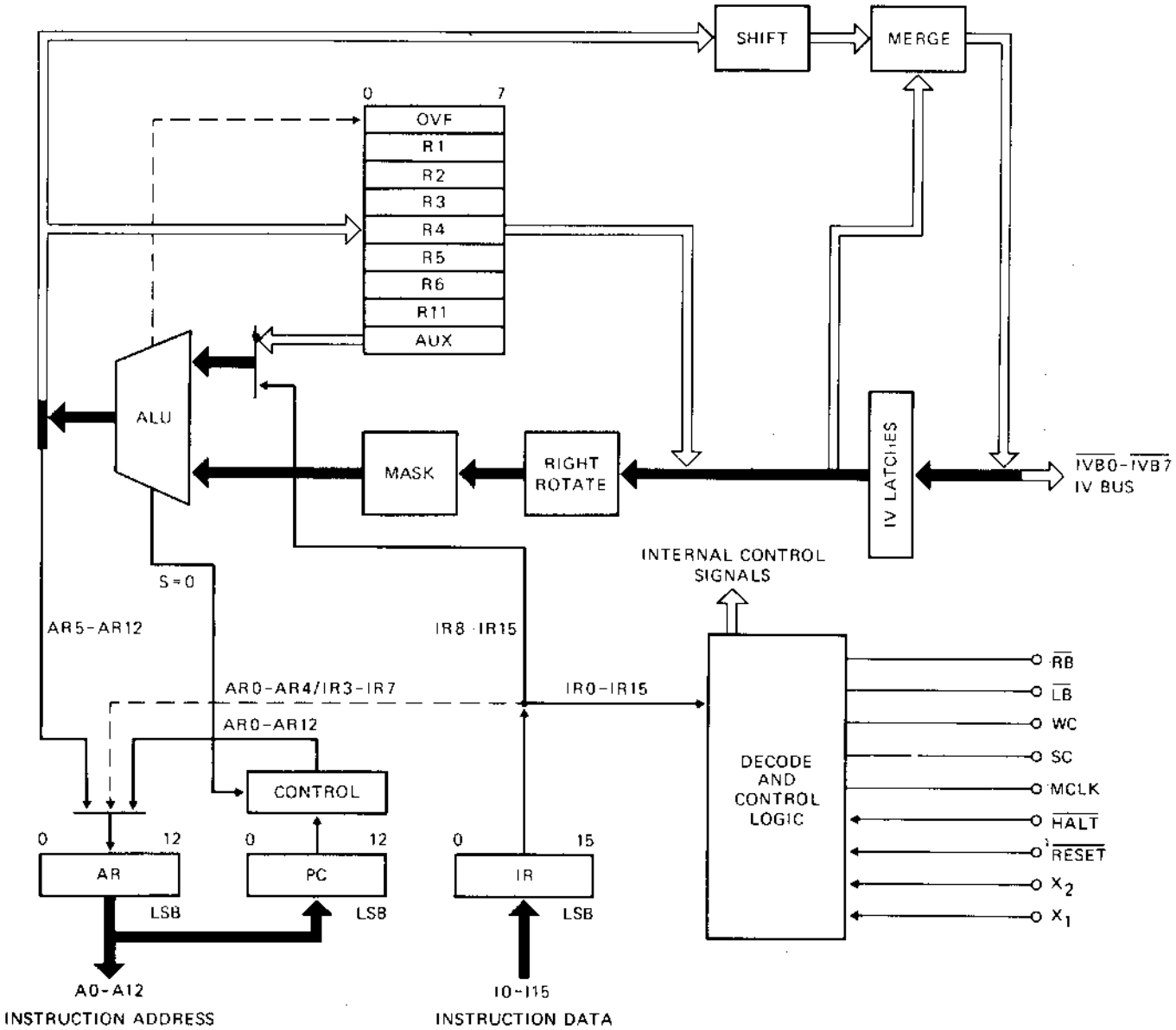
Initial value of address register
and program counter:
137_8 (00000010 11111)

IV bus input	1 0 1 1 0 1 1 0
rotate 2 places	1 0 1 0 1 1 0 1
mask 1 bit (L = 1)	0 0 0 0 0 0 0 1

Mask field ≠ 0 so (I)
moved to AR and PC.
contents of I 0 1 1 0 1
New value of address register
and program counter: 115_8 (00000010 01101)

New value of address register and program
counter if contents of IV byte = 0 : 140_8 (0000001100000)

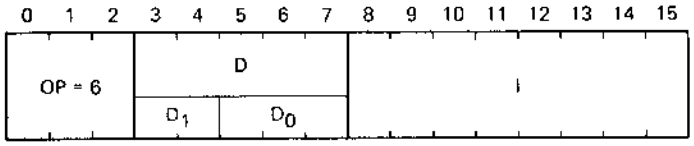
Data flow



XMIT Instructions — Op Code 6

XMIT, Register

Format



Operation

I ⟶ D

Description

Store the value of the 8-bit integer in the register specified by D.

D specifies the register to be loaded.

I is the 8-bit field containing the value to be loaded into the register.

Permitted operand values

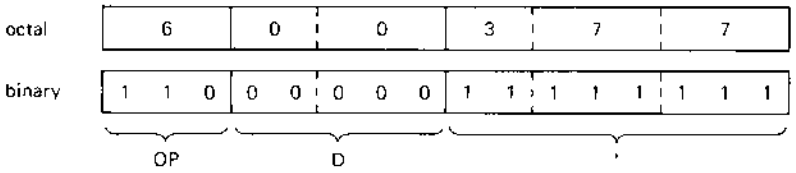
D: 00/01/02/03/04/05/06/11

I: $0 \leq I \leq 377_8$

EXAMPLE

Set the value 377_8 in the AUX register.

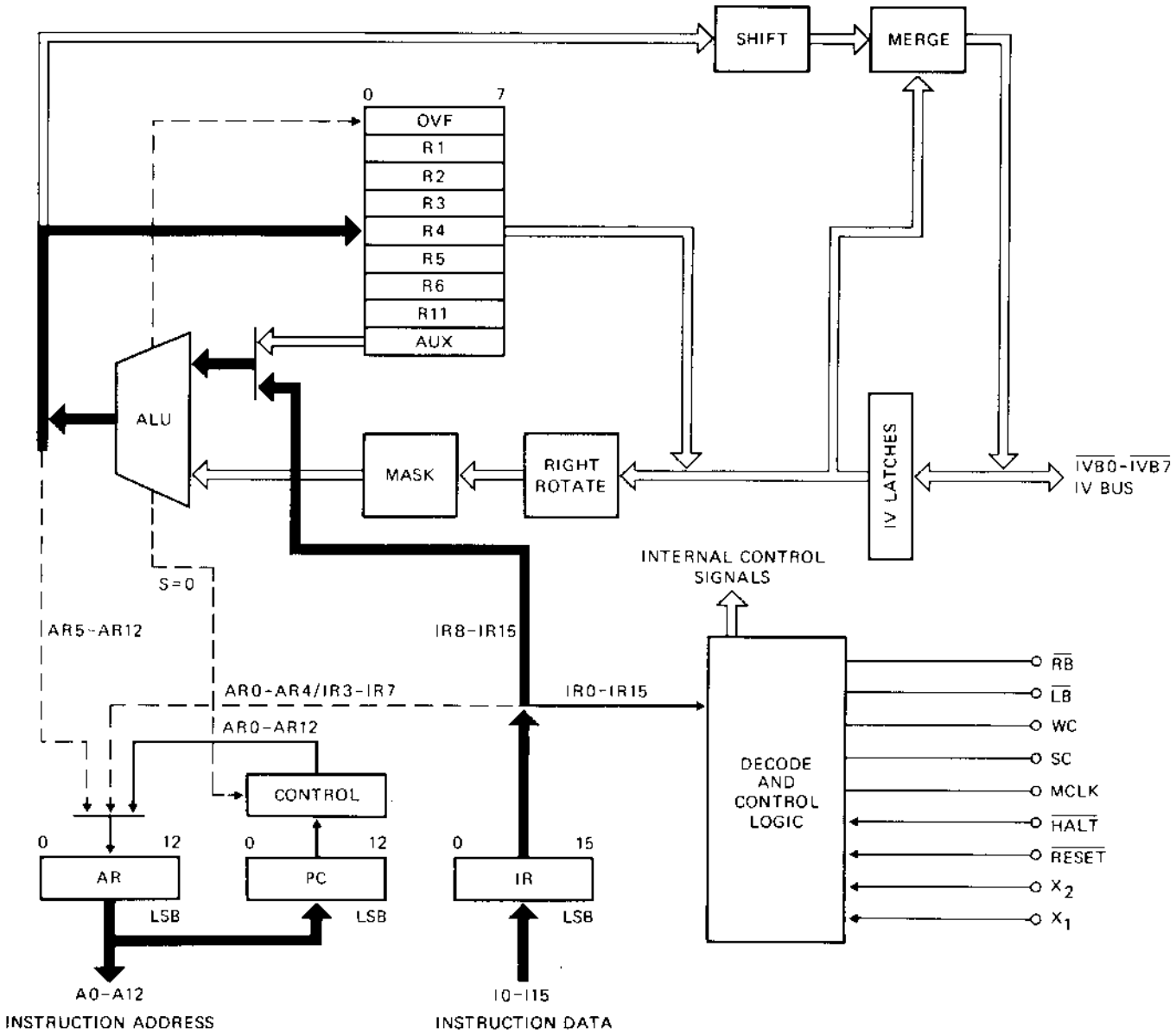
Instruction word



Instruction operation

initial contents of AUX	0 1 1 0 0 1 0 1
I field	1 1 1 1 1 1 1 1
new contents of AUX	1 1 1 1 1 1 1 1

Data flow



XMIT INSTRUCTIONS — OP CODE 6

XMIT, Register

Assembler notation

XMIT, 377H, AUX

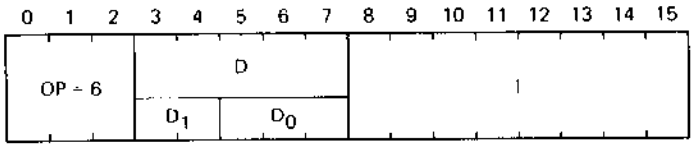
Result

The value of the I field is set in the destination register.

XMIT Instructions — Op Code 6

XMIT, IV bus address

Format



Operation

I ⟶ D

Description

Enable the IV byte, at the bank specified by D, whose address is the 8-bit integer I.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies the left bank address:
D = 17 specifies right bank address.

I is the 8-bit field specifying the address of the byte to be enabled.

The order of operation is:

copy the 8 least significant bits of the instruction word;

output the 8-bit field to the IV bus as an address on the bank specified by D.

Permitted operand values

D: 07 / 17

I: $0 \leq I \leq 377_8$

EXAMPLE

XMIT Instructions — Op Code 6

Enable the IV byte at the left bank whose address is 53_8.

XMIT, IV bus address

Instruction word

octal	6			0			7		0			5			3
binary	1	1	0	0	0	1	1	1	0	0	1	0	1	0	1	1
			OP			D					I				

Assembler notation

XMIT 027H, IVL.

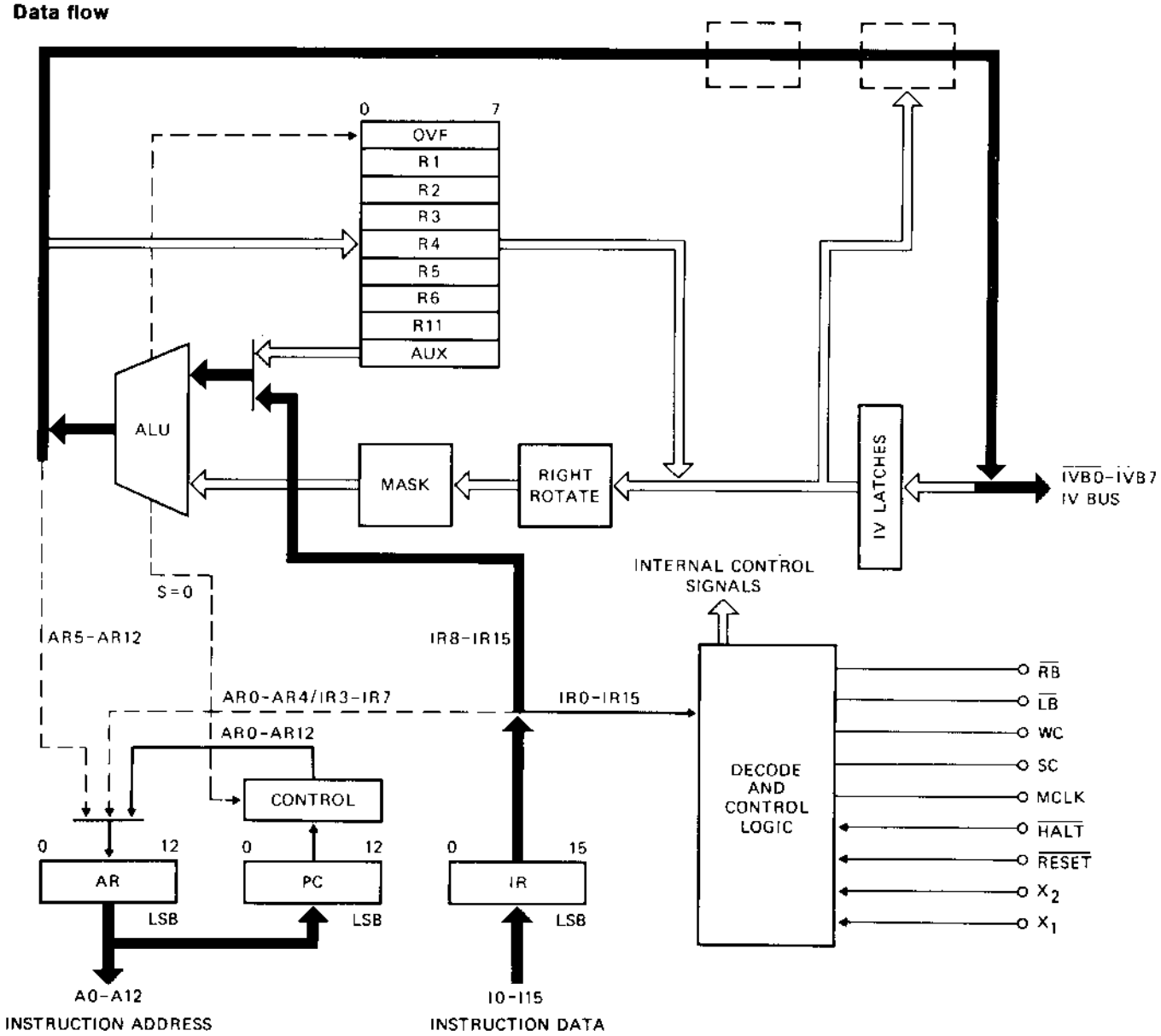
Instruction operation

value of I field (53_8)	0 0 1 0 1 0 1 1
new I/O address at left bank	0 0 1 0 1 0 1 1

Result

The previously enabled IV byte at the left bank is disabled and the byte at address 27_8 at the left bank is enabled. The right bank is not affected.

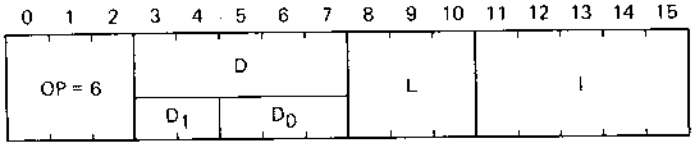
Data flow



XMIT Instructions — Op Code 6

XMIT, IV bus

Format



Operation

I ⟶ D

Assembler notation

XMIT 03, LIV5, 3

Description

Transmit the least significant L bits of the I field to the L-bit field of the IV bus specified by D. If L is greater than 5 bits, the most significant bits of the destination field are filled with zeros.

D_1 specifies the bank of the IV bus which is the destination:
D_1 = 2 selects the left bank;
D_1 = 3 selects the right bank.

L specifies the length of the destination field (number of bits).
Note that L = 0 selects an 8-bit field.

D_0 specifies the bit position in the IV bus with which the least significant bit of the I field data should be aligned. This means that the I field data is left-shifted so that bit 7 is aligned with bit D_0 of the IV bus.

The order of operation is:

read the contents of the destination IV byte into the input latches;

copy the least significant 5 bits of the instruction word;

left shift the copied 5-bit field as specified by D_0;

merge the shifted field, as specified by L, with the contents of the IV latches and output the result to the IV bus.

Note that the data in the IV latches outside the field specified by D_0 and L is not altered.

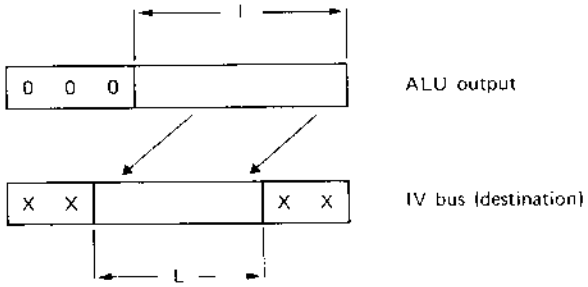
Permitted operand values

D_0: 0/1/2/3/4/5/6/7

D_1: 2/3

L: 1/2/3/4/5/6/7/0

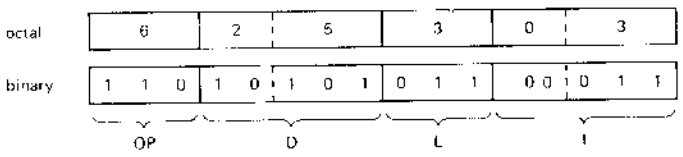
I: $0 \leq I \leq 37_8$



EXAMPLE

Transmit the value 3 to bits 3, 4 and 5 of the IV byte at the left bank.

Instruction word



XMIT Instructions — Op Code 6

XMIT, IV bus

Instruction operation

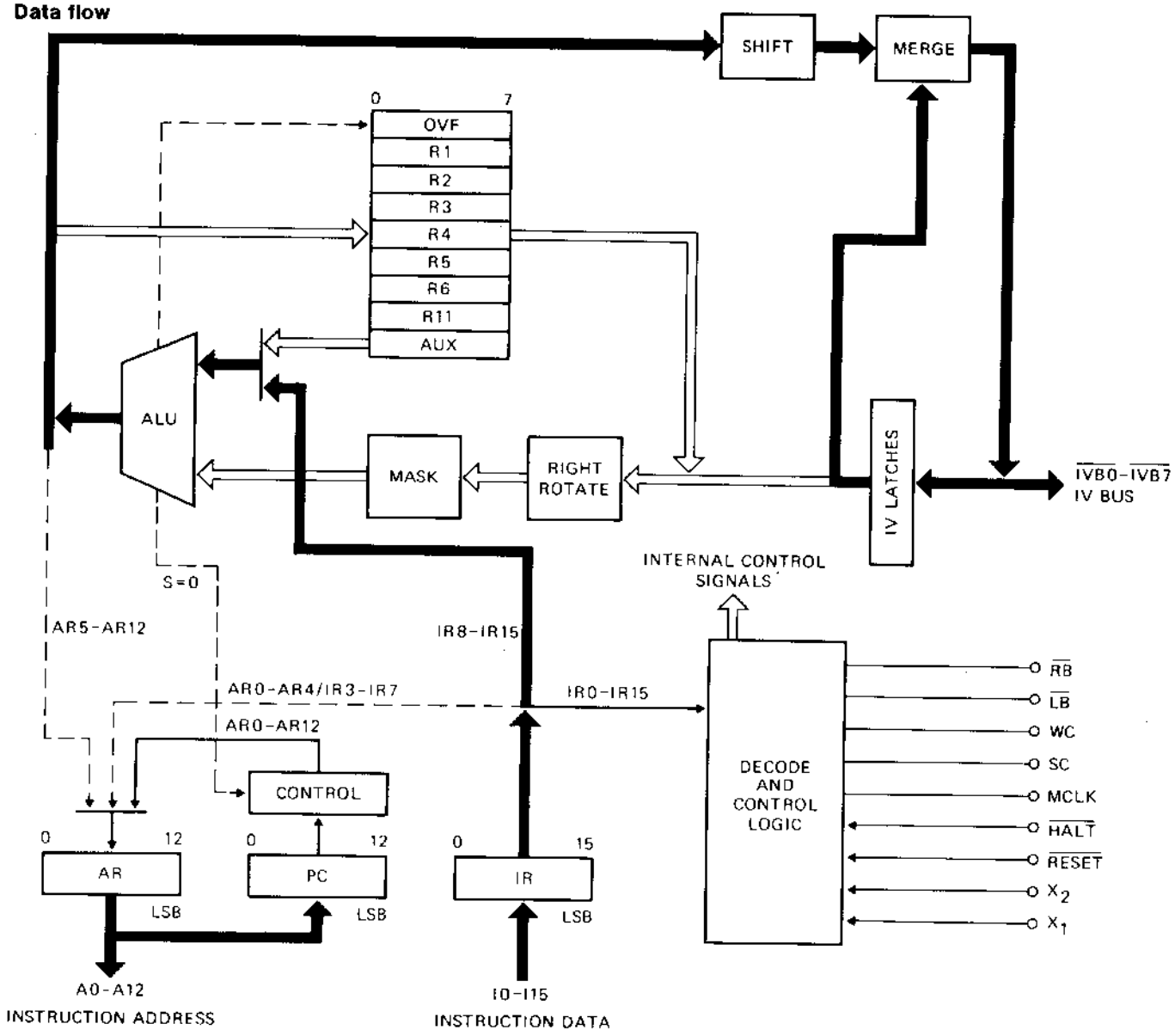
initial contents of IV byte	0 1 0 0 0 1 0 0
value of I field	0 0 0 1 1
output of ALU	0 0 0 0 0 0 1 1
shift 2 places	0 0 0 0 1 1
merge L bits and output to IV bus to IV byte	0 1 0 0 1 1 0 0

orginal data of IV byte

Result

The three least significant bits of the I field are transmitted to bits 3, 4 and 5 of the IV byte at the left bank.

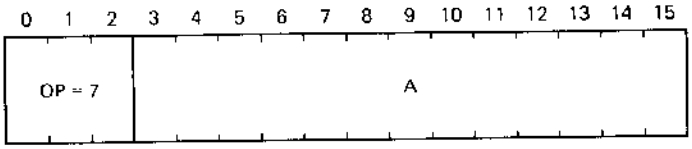
Data flow



JMP Instruction — Op Code 7

JMP, address

Format



Operation

Set the value in the A field into the program counter and address register.

Description

Jump to the instruction address specified by the A field, and continue normal program execution from that address. The contents of the 13-bit A field are loaded into the program counter and address register. The next instruction to be executed is then the instruction at the new address.

A is the 13-bit field specifying the address to which the jump is made.

The order of operation is:

load the address register and program counter with the contents of the A field;

new address value is used for next instruction.

Permitted operand values

A: $0 \leq A \leq 17777_8$ (8191_{10})

JMP Instruction — Op Code 7

JMP, address

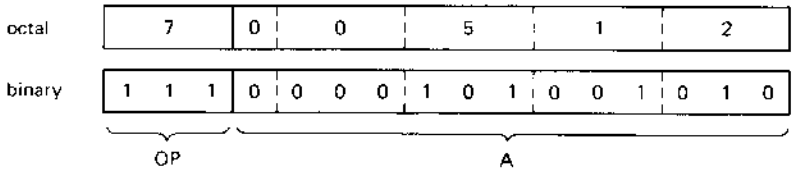
Assembler notation

JMP 512H

Example

Jump to address 512_8.

Instruction word



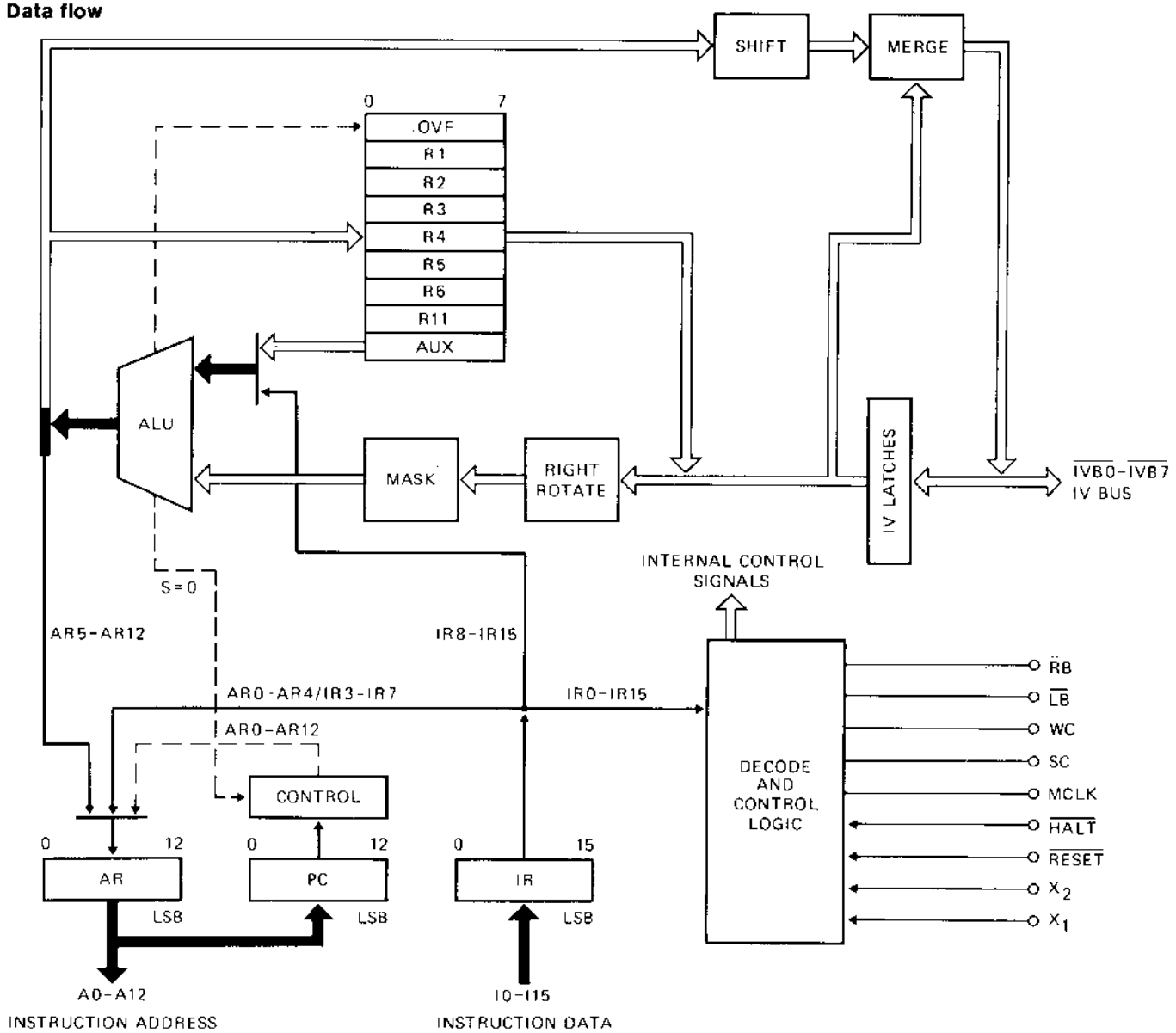
Instruction operation

current address (of JMP instr.) Value of PC and AR	0 000 000 111 010	72_8
new address (contents of A field) new value of PC and AR.	0 000 101 001 010	512_8

Result

The next instruction to be executed will be that at the address specified by the A field (512_8).

Data flow



MICROCONTROLLER CROSS ASSEMBLY PROGRAM

The 8X300 Cross Assembly Program, MCCAP, provides a programming language which allows the user to write programs for the 8X300 in symbolic terms. MCCAP translates the user's symbolic instructions into machine-oriented binary instruction. For example, the jump instruction, JMP, to a user defined position, say ALPHA, in program storage is coded as:

```
JMP ALPHA
```

and is translated by MCCAP into the following 16-bit word (see Figure 1).

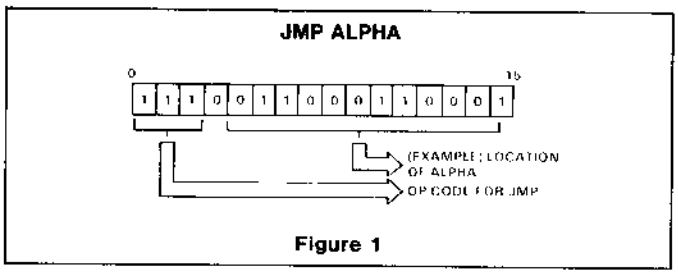


Figure 1

MCCAP allocates the 8X300 program storage and assigns Interface Vector and Working Storage address to symbols as declared in the user's program.

The ability to define data of the Interface Vector as symbolic variables is a powerful feature of MCCAP. Interface Vector variables may be operated on directly using the same instructions as those for variables in Working Storage and for the working registers.

The Assembler Declaration statements of MCCAP allow the programmer to define symbolic variable names for data elements tailored to his application. Individual bits and sequences of bits in Working Storage and on the Interface Vector may be named and operated upon directly by 8X300 instructions.

In addition to simplifying the language and bookkeeping of the program, MCCAP provides program segmentation and communication between segments; i.e., the main program and any subprograms. If a sequence of code appears more than once in a program, it can be written as a separate program segment, a subprogram, and called into execution whenever that subprogram's function is required. Program segmentation also permits the construction of a program in logically discrete units. These segments need not be written sequentially or even by the same person. The various program segments provide a function description, or block diagram, of the application. Communication between segments means that control and data can be transferred in both directions. MCCAP automatically generates the code for subprogram entry and exit mechanisms when the appropriate CALL and RTN statements are invoked.

MCCAP OUTPUT

The output from a MCCAP compilation includes an assembler listing and an object module. During pass two of the assembly process, a program listing is produced. The listing displays all information pertaining to the assembled program. This includes the assembled octal instructions, the user's original source code and error messages. The listing may be used as a documentation tool through the inclusion of comments and remarks which describe the function of a particular program segment. The main purpose of the listing, however, is to convey all pertinent information about the assembled program, i.e., the memory addresses and their contents.

The object module is also produced during pass two. This is a machine-readable computer output produced on paper tape. The output module contains the specifications necessary for loading the memory of the Microcontroller Simulator (MCSIM), for loading the memory of the SMS ROM Simulator, or for producing ROMs or PROMs. The object module can be produced in MCSIM, ROM Simulator or BNPF format.

An example of a MCCAP source program is shown in Figure 2.

PROGRAM STRUCTURE

Program Segments

A MCCAP program consists of one or more program segments. Program segments are the logically discrete units, such as the main program and subprograms, which comprise a user's complete program. Program segments consist of sequences of program statements. The first program segment must be the main program. The main program names the overall program and is where execution begins. All other segments are subprograms; each subprogram must be named. Control and data can be passed in both directions between segments. No segment may call itself, or one of its callers, or the main program. Program segments take the form as shown in Figure 3.

The Assembler Declaration statements define variables and constants. They must precede the use of the declared variables and constants in the Executable Statements in a program. The Executable Statements are those which result in the generation of one or more executable machine instructions.

Subprograms

Subprograms are program segments which perform a specific function. A major reason for using subprograms is that they reduce programming and debugging labor when a specific function is required to be executed at more than one point in a program. By creating the required function as a subprogram, the statements associated with that function may be coded once and executed at many different points in a program. Figure 3 illustrates an example.

The program structure in Figure 3 causes the code associated with PROC WAIT to be executed three times within PROG MANYWAIT. This is accomplished even though the statements associated with PROC WAIT are coded only once, rather than three times.

Subprogram Calls and Returns

For user-provided procedures, a jump to the associated procedure and a return link are created for each procedure reference. The instructions to accomplish this result in subprogram entry time. The instructions to accomplish subpro-

MCCAP SOURCE PROGRAM

MICROCONTROLLER SYMBOLIC ASSEMBLER VER 1.0

```
1680                        *
1681                        *
1682  01544                 PROC    RDCMMD
1683                        *
1684  01544  6 07003        SEL     IVRESP      FDC RESPONSE BYTE
1685  01545  6 20101        XMIT    UR, BCTRL   ESTABLISH USER READ ONLY
1686  01546  6 07002        SEL     IVDATA      HOLDS COMMAND BYTE
1687  01547  0 27305        MOVE    FUNC, R5    FUNCTION CODE
1688  01550  0 24306        MOVE    DADDR, R6   DISK ADDRESS
1689  01551  0 21202        MOVE    BUFF, R2    BUFFER FUNCTION CODE
1690  01552  6 07003        SEL     IVRESP
1691  01553  6 25100        XMIT    0, DONE     SHOW COMMAND IN PROGRESS
1692  01554  6 20100        XMIT    UW, BCTRL   RESTORE USER WRITE
1693  01555  6 27101        XMIT    1, XFR      SIGNAL USER FDC ACCEPTED BYTE
1694  01556  6 07001        SEL     IVCTRL      USER CONTROL BYTE
1695  01557  5 26117        NZT     CMMD, *     WAIT FOR CMMD TO GO LOW
1696  01560  6 07003        SEL     IVRESP      FDC RESPONSE BYTE
1697  01561  6 27100        XMIT    0, XFR      LOWER XFR SIGNAL
1698  01562  6 07001        SEL     IVCTRL      USER CTRL BYTE
1699  01563  4 26123        XEC     *(CMMD), 2  WAIT FOR NEXT COMMAND SIGNAL
1700  01564  6 07003        SEL     IVRESP      SECOND COMMAND BYTE AVAILABLE
1701  01565  6 20101        XMIT    UR, BCTRL   SET IVDATA TO USER READ ONLY
1702  01566  6 07002        SEL     IVDATA      2ND COMMAND BYTE
1703  01567  0 27704        MOVE    TRACK, R4   TRACK ADDRESS
1704  01570  0 27503        MOVE    SECT, R3    SECTOR ADDRESS
1705  01571  6 07003        SEL     IVRESP      FDC RESPONSE BYTE
1706  01572  6 27101        XMIT    1, XFR      SIGNAL USER
1707  01573  6 20100        XMIT    UW, BCTRL   RESTORE USER WRITE
1708  01574  6 07001        SEL     IVCTRL
1709  01575  5 26135        NZT     CMMD,*      WAIT FOR CMMD TO GO LOW
1710  01576  6 07003        SEL     IVRESP      FDC RESPONSE BYTE
1711  01577  6 27100        XMIT    0, XFR      LOWER XFR SIGNAL
1712                        *
1713  01600  7 01652        RTN                 RETURN
1714                        END     RDCMMD
1715
```

Figure 2

PROGRAM SEGMENTS

PROGRAM STATEMENT
DECLARATION STATEMENT(S)
•
•
•
EXECUTABLE STATEMENT(S)
•
•
SUBPROGRAMS
•
END STATEMENT

a. Main Program Form

PROCEDURE STATEMENT
DECLARATION STATEMENT(S)
•
•
•
EXECUTABLE STATEMENT(S)
•
•
•
END STATEMENT

b. Subprogram Form

Figure 3

gram exit result in exit time. The user may utilize the MCCAP procedure mechanism for linking calling programs with called programs or he may create his own instructions to do so. The following describes the linkage mechanism and timing for MCCAP user procedures.

Linkage between called and calling programs is achieved through the generation of an indexed "return jump" table, the length of which corresponds to the number of different times in the program that the subprograms are called. This table is generated automatically by MCCAP when procedure CALL and RTN statements are invoked. For each procedure reference, MCCAP creates two statements in the calling program. Thus, the time required for the subprogram entry is 0.5 microseconds. The subprogram return mechanism requires the execution of two instructions or 0.5 microseconds. These times do not include saving and restoring of the working registers. The total time to save all working registers is 3.5 microseconds, the same time to restore all

registers. Saving of all working registers is normally not necessary, but worst case calculations for entry and exit time below do include this time. Thus, subprogram exit and entry times are:

$$0.5\mu s \leq \text{Entry Time} \leq 4.0\mu s$$
$$0.5\mu s \leq \text{Exit Time} \leq 4.0\mu s$$

Details of the code required for procedure CALL and RTN are provided in the Programming Examples section. See Figures 21 and 22.

Macros

A macro is a sequence of instructions that can be inserted in the assembly source text by encoding a single instruction. The macro is defined only once and may then be invoked any number of times in the program. This facility simplifies the coding of programs, reduces the chance of errors, and makes programs easier to change.

A macro definition consists of a heading, a body and a terminator. This definition must precede any call on the macro. In MCCAP, the heading consists of the MACRO statement which marks the beginning of the macro and names it. The body of the macro is made up of those MCCAP statements which will be inserted into the source code in place of the macro call. The terminator consists of an ENDM statement which marks the physical end of the macro definition.

MCCAP Statements

The MCCAP language consists of thirty statements categorized as follows:

Assembler Directive Statements
Assembler Declaration Statements
Communication Statements
Macro Statements
Machine Statements

The following lists the statements in each category, describes their use, and provides examples. Detailed use of the instructions including rules of syntax and parameter restrictions are described in the MCCAP Reference Manual.

Assembler Directive Statements

Assembler Directive statements define program structure and control the assembler outputs. They do not result in the generation of 8X300 executable code. There are twelve Assembler Directive statements:

PROG Statement
PROC Statement
ENTRY Statement
END Statement
ORG Statement
OBJ Statement
IF Statement
ENDIF Statement
LIST Statement
NLIST Statement
EJCT Statement
SPAC Statement

PROG Statement

Use

Defines the names and marks the beginning of a main program.

Example: PROG PROCESS

PROC Statement

Use

Defines the names and marks the beginning of a subprogram.

Example: PROC WAIT

ENTRY Statement

Use

Defines the name and marks the location of a secondary entry point to a subprogram.

Example: ENTRY POINT 2

END Statement

Use

Terminates a program segment or a complete program.

Examples: *END SUB1*
END MAIN

ORG Statement

Use

Sets the program counter to the value specified in the operand field.

Example: ORG 200

OBJ Statement

Use

To specify the format of the object module.

Examples: OBJ R
OBJ M
OBJ N

NOTE

"R" indicates the ROM Simulator format. "M" indicates the Microcontroller Simulator format. "N" indicates BNPF format.

IF Statement

Use

To mark the beginning of a sequence of code, which may or may not be assembled depending on the value of an expression.

Examples: IF VAL
IF X + Y

MOVE Statement

Use

To copy the contents of a specified register, WS variable or IV variable into a specified register, WS or IV. Defined in Instruction Descriptions.

Examples: MOVE R1(6);R6
MOVE X,Y

NOTE

The first example illustrates a six place right rotate of R1's data before it is moved to R6. The contents of R1 are not affected. The second example may be a Working Storage or Interface Vector variable move, depending on the way X and Y are defined in Declaration Statements.

ADD Statement

Use

To add the contents of a specified register, WS variable, or IV variable to the contents of the AUX register and place the result in a specified register, WS variable or IV variable.

Examples:
```
ADD R1(3),R2
ADD DATA,OUTPUT
```

NOTE

The first example illustrates a three place right rotate of R1's data before the addition is carried out. Under certain conditions a rotate may be used to multiply the specified operand by a power of 2 before the addition is done. The contents of R1 are not affected. The second example suggests that the contents of WS variable have been added to the contents of the AUX register and the result placed in an IV variable, making the result immediately available to the user's system.

AND Statement

Use

To compute the logical AND of the contents of a specified register, WS variable or IV variable and the contents of the AUX register. The logical result is placed in a specified register, WS variable or IV variable. In actual practice, the AND statement is often used to mask out undesired bits of a register.

Examples:
```
AND R2,R2
AND R3(1),R5
AND X,Y
```

NOTE

The first example illustrates the use of an AND statement in what might be a masking operation. If the AUX register contains 00001111 then this statement sets the 4 high order bits of R2 to 0 no matter what they were originally. The 4 low order bits of R2 would be unaffected.

The second example illustrates a one place rotate to the right of R3's data before the AND is carried out. The contents of R3 are not affected. In the third example, X and Y may be parts of the same WS or IV byte, or one may be a WS byte and the other an IV byte.

XOR Statement

Use

To compute the logical exclusive OR of the contents of a specified register, WS variable or IV variable and the contents of the AUX register, and place the result in a specified register, WS variable or IV variable. In practice, the XOR statement is often used to complement a value and to perform comparisions.

Examples:
```
XOR R6,R11
XOR R1(7),R4
XOR X,Y
```

NOTE

The first example illustrates the use of an XOR statement in what might be a complementing operation. If the AUX register contains all 1's then the execution of this statement results in the complement of the contents of R6 replacing the contents of R11. The second and third examples are of the same form as the second and third examples of the AND statement.

XMIT Statement

Use

To transmit or load literal values into registers, WS variables or IV variables.

Examples:
```
XMIT DATA,IVR
XMIT OUTPUT,IVL
XMIT -11,AUX
XMIT -00001011B,AUX
XMIT -13H,AUX
```

NOTE

The first example selects a previously declared WS variable by transmitting its address to the IVR register. The second example selects a previously declared IV variable by transmitting its address to the IVL register. The last three examples all result in the generation of the same machine code. They all load the AUX register with -11_{10}. In the first case, the programmer has written the number in base 10. In the second case, the programmer has written the number in binary and has indicated this by placing a B after the number. In the third case, the number has been written in octal as indicated by an H after the number.

XEC Statement

Use

To select and execute one instruction out of a list of instructions in program memory as determined by the value of a data variable, and then continue the sequential execution of the program beginning with the statement immediately following the XEC unless the selected instruction is a JMP or NZT statement.

Examples:
```
                    XEC JTABLE(R1),3
JTABLE      JMP     GR8ERTHAN
                    JMP LESSTHAN
                    JMP EQUALTO

                    XEC SEND (INPUT),4
                    "NEXT INSTRUCTION"
                    "NEXT INSTRUCTION"

SEND                XMIT 11011011B,AUX
                    XMIT 11111111B,AUX
                    XMIT 10101010B,AUX
                    XMIT 00000000B,AUX
```

NOTE

In the first example, the execution of the program will be transferred to one of three labeled instructions on the basis of whether register R1 contains 0, 1 or 2. In the second example, the XEC statement causes the execution of a statement which transmits a special bit pattern to the AUX register in response to an input signal which is either 0, 1, 2 or 3. After the pattern is transmitted, the execution of the program continues with the next instruction after the XEC.

NZT Statement

Use

To carry out a conditional branch on the basis of whether or not a register, WS variable, or IV variable is zero or non-zero.

Examples:
```
NZT R1,*+2
NZT SIGN,NEG
```

NOTE

In the first example, if the contents of R1 are non-zero, then program execution will continue with the instruction, whose address is the sum of the address of the NZT statement and 2. If the contents of R1 are 0, the program execution continues with the next instruction after the NZT statement. In the second example, if the contents of a WS or IV variable called SIGN is non-zero, then program execution will continue beginning with the instruction whose address is NEG. Otherwise execution continues with the next instruction after the NZT statement.

JMP Statement

Use

To transfer execution of the program to the statement whose address is the operand of the JMP statement.

Examples:
```
JMP START
JMP *-2
```

NOTE

In the first example, execution of the program continues sequentially beginning with the instruction labeled START. In the second example, program ex-

ecution continues beginning with the instruction whose address is the JMP instruction's address minus 2.

SEL Statement

Use

Select a variable in Working Storage or on the Interface Vector, so that subsequent machine instructions may reference that variable.

Examples: SEL DATA
SEL OUTPUT

NOTE

It is the programmer's responsibility to assure that the proper page has been addressed before calling the SEL statement if the variable may be in Working Storage. The SEL statement causes a single instruction, XMIT, to be assembled into the user program. The operand of the XMIT instrucion is the byte address of the named variable (argument of the reference) as it has been allocated in Working Storage or on the Interface Vector.

PROGRAMMING EXAMPLES

This section contains programming examples which demonstrate how the 8X300's instructions can be assembled to perform some simple, commonly required functions. These examples are written as program fragments. They are not complete programs as the Data Declaration and Directive statements have been omitted. Otherwise, they follow standard MCCAP conventions.

Looping

Looping is terminated by incrementing a counter and testing for zero. Register R1 is used as counter register and is loaded with a negative number so that the program counts up to zero. Figure 4 illustrates the process.

LOOPING

```
        XMIT    NEG,R1
                Load negative loop count.
ALPHA   •••
        •
        •
        •       Loop start.
        XMIT    1,AUX
                Store increment value in AUX register
                which is an implicit operand of ADD in-
                struction.
        ADD     R1,R1
                Increment counter register. Add con-
                tents of AUX to contents of R1 and
                store the sum in R1.
        NZT     R1,ALPHA
                Test contents of R1 for zero. If zero, ex-
                ecute next sequential instruction, oth-
                erwise, jump to ALPHA and continue
                execution from there.
        •
        •
        •
```

TIME: 750 nanoseconds

Figure 4

Inclusive–OR (8 Bits)

Generate inclusive-OR of the contents of R1 and R2. Store the logical result in R3. Although the 8X300 does not have an OR instruction, it can be quickly implemented by making use of the fact that (A + B) + (AB) is logically equivalent to A ⊕ B.

INCLUSIVE-OR

```
MOVE    R2,AUX  Load one of the operands into AUX reg-
                ister so that it can be used as the im-
                plicit operand of XOR and AND
                instructions.
XOR     R1,R3   Take exclusive OR of AUX and R1.
                Store result in R3.
AND     R1,AUX  Take AND of AUX and R1. Place results
                in AUX.
XOR     R3,R3   Take exclusive OR of AUX (A + B) and
                R3 (A + B). Store result in R3. R3 now
                contains inclusive OR of R1 and R2.
```

TIME: 1.0 microseconds

Figure 5

Two's Complement (8-Bits)

Generate the two's complement of the contents of R2. Store the result in R3. Assume that R2 does not contain 200_8.

TWO'S COMPLEMENT

```
XMIT    −1,AUX  Load AUX in preparation for XOR.
XOR     R2,R3   1's complement of R2 is now in R3.
XMIT    1,AUX   Load AUX in preparation for ADD.
ADD     R3,R3   2's complement of R2 is now in R3.
```

TIME: 1.0 microseconds

Figure 6

8-Bit Subtract

Subtract the contents of R2 from the contents of R1 by taking the two's complement of R2 and adding R1. Store the difference in R3.

8-BIT SUBTRACT

```
XMIT    −1,AUX  Perform 2's complement, R2.
XOR     R2,R3
XMIT    1,AUX
ADD     R3,AUX  2's complement of R2 is now in AUX.
ADD     R1,R3   R1-R2 is now in R3.
```

TIME: 1.25 microseconds

Figure 7

16-Bit ADD, Register to Register

Add a 16-bit value stored in R1 and R2 to a 16-bit value in R3 and R4. Store the result in R1 and R2.

16-BIT ADD, REGISTER TO REGISTER

MOVE	R2,AUX	Move low order byte of first operand to AUX in preparation for ADD.
ADD	R4,R2	Add the low order bytes of the two operands and store the result in R2. R2 contains the low order byte of the result.
MOVE	R1,AUX	Move high order byte of first operand to AUX.
ADD	OVF,AUX	Add in possible carry from addition of low order bytes.
ADD	R3,R1	Add the high order bytes plus carry and place result in R1. R1 contains the high order byte of the result.

TIME: 1.25 microseconds

Figure 8

16-Bit ADD, Memory to Memory

Add a 16-bit value in Working Storage, OPERAND1, to a 16-bit value in Working Storage, OPERAND2, and store result in Working Storage OPERAND1. H1 and L1 represent the high and low order of bytes OPERAND1. H2 and L2 represent the high and low order bytes of OPERAND2.

16-BIT ADD, MEMORY TO MEMORY

XMIT	L2,IVR	Transmit address of low order byte of second operand to IVR.
MOVE	L2, AUX	Move low order byte to AUX.
XMIT	L1,IVR	Transmit address of low order byte of first operand to IVR.
ADD	L1,L1	Add low order bytes and store result in L1.
MOVE	OVF,AUX	Move possible carry from additon of low order bytes to AUX register.
XMIT	H2,IVR	Add high order byte of second operand to possible carry. Store result in AUX.
ADD	H2,AUX	
XMIT	H1,IVR	
ADD	H1,H1	High order byte of sum is in H1. Low order byte of sum is in L1.

TIME: 2.25 microseconds

Figure 9

Byte Assembly From Bit Serial Input

This is typical of problems associated with interfacing to serial communications lines. An 8-bit byte is assembled from bit inputs that arrive sequentially at the Interface Vector. A single bit on the Interface Vector named STROBE is used to define bit timing, and a second bit, named INPBIT, is used as the bit data interface. Figure 10 illustrates the byte assembly.

BYTE ASSEMBLY

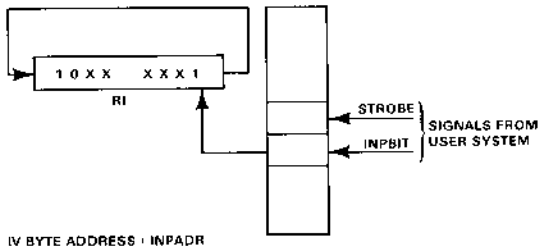


Figure 10

BYTE ASSEMBLY PROGRAM

XMIT	0,R1	R1 will be used as a character buffer. It has been cleared.
XMIT	8,R2	R2 will be used as a bit counter.
XMIT	INPADR,IVL	Select IV Byte that contains INPBIT and STROBE.
NEXT		
BIT NZT	STROBE,*+2	Test STROBE for data ready. The MOVE instruction is executed only when STROBE = 1.
JMP	*−1	Loop until STROBE = 1.
MOVE	INPBIT,AUX	
XOR	R1(1),R1	Rotate R1 one place right. This puts a zero in the least significant bit position. Then take the exclusive OR of this rotated version of R1 and of AUX. Place the result in R1. The least significant bit of R1 will now equal the latest value of INPBIT.
XMIT	−1,AUX	
ADD	R2,R2	Decrement R2. IF R2 is not yet zero, then more bits must be collected to complete the byte being assembled.
MOVE	R1(1),R1	This instruction will only be executed when 8 bits have been collected. After this is done it is still necessary to rotate one more time to get the last INPBIT into the high order bit position of R1.

TIME: 1.8 microseconds per bit (minimum)

Figure 11

Rotate Left

The 8X300 has no instructions which explicitly rotate data to the left. Such an instruction would be redundant because of the circular nature of the rotate operation. For example, a rotate of two places to the left is identical to a rotate of six places to the right. The rotate n places to the left in an 8-bit register, rotate 8-n places to the right. This example illustrates a rotate of the contents of R4 three places to the left.

MOVE R4(5), R4
TIME: 250 nanoseconds

Three Way Compare

The contents of R1 are compared to the contents of R2. A branch is taken to one of three points in the program depending upon whether R1 = R2, R1 < R2, or R1 > R2.

THREE WAY COMPARE

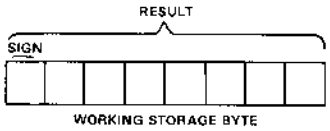


Figure 12

THREE WAY COMPARE PROGRAM

```
XMIT      RESULT,IVR      Choose a working Storage byte by transmitting its address to IVR register.
XMIT      -1,AUX          Load AUX with all 1's, in preparation for complementing contents of R2.
XOR       R2,RESULT       Store complement of R2 in RESULT.
XMIT      1,AUX
ADD       RESULT,AUX      AUX now contains 2's complement of R2.
ADD       R1,RESULT       RESULT now contains R1-R2.
NZT       RESULT,NEQUAL   If RESULT ≠ 0, then R1 ≠ R2.
JMP       EQUAL
NEQUAL    NZT SIGN,LESS   Sign Bit = 1 only when R1 < R2.
GREATER   Continue
                            •
                            •
                            •
EQUAL     Continue
                            •
                            •
                            •
LESS      Continue
TIME  2.0 microseconds
```

Figure 13

Interrupt Polling

Three external interrupt signals are connected to three IV bits. The three bits are scanned by the program to determine the presence of an interrupt request. A branch is taken to one of eight program locations depending upon whether any or all of the interrupt request signals are present. The IV bits associated with the interrupt requests are wired to the low order three bits of the IV byte named Control. Figures 14 and 15 illustrate the interrupt polling.

INTERRUPT POLLING

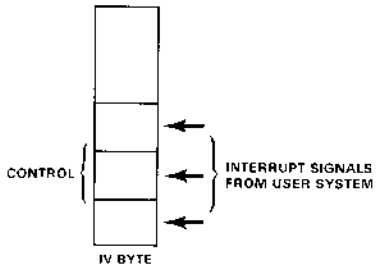


Figure 14

INTERRUPT POLLING PROGRAM

```
XMIT      CONTROL,IVL     Choose proper IV Byte by transmitting its address to IVL register.
XEC       JTABLE (CONTROL),8   Execute the one instruction whose address is the sum of JTABLE and the contents of CONTROL. The 8 indicates the length of the table.
                            •
                            •
                            •
JTABLE    JMP  ALPHA1     Table of 8 instructions, one of which is executed as a result of the XEC instruction above.
          JMP  ALPHA2
                            •
                            •
                            •
          JMP  ALPHA8
TIME: 750 nanoseconds
```

Figure 15

Bit Pattern Detection In An I/O Field

Test input field called Input for specific bit pattern, for example: 1 0 1 1. If pattern is not found, branch to NFOUND, otherwise continue sequential execution. Figures 16 and 17 illustrate the procedure.

BIT PATTERN DETECTION

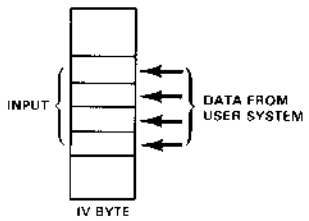


Figure 16

BIT PATTERN DETECTION PROGRAM

```
XMIT      INPUT,IVL       Choose proper IV Byte by transmitting its address to IVL register.
XMIT      1011B,AUX       Store desired bit pattern in AUX register for use as implicit operand of XOR instruction.
XOR       INPUT,AUX       Take exclusive OR of the contents of INPUT and AUX. Store the result in AUX. Now the contents of AUX will be zero if the contents of INPUT are 1011.
NZT       AUX,NFOUND      Test AUX for zero. Branch to NFOUND if non zero.
  •
  •
  •
NFOUND Continue
TIME  1.0 microseconds
```

Figure 17

Control Sequence #1

Set an output bit when an input bit goes high (is set) (see Figure 18).

CONTROL SEQUENCE #1

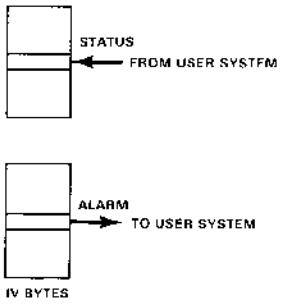


Figure 18

CONTROL SEQUENCE #1 PROGRAM

```
XMIT      STATUS,IVL      Choose input IV byte by transmitting its address to IVL.
NZT       STATUS,*+2      Test input bit to determine whether it is still zero. Skip next instruction if it is not zero.
JMP       *-1             Jump to previous instruction.
XMIT      ALARM,IVL       Choose output IV byte.
XMIT      1,ALARM         Set output bit by loading ALARM with 1.
TIME  1.0 microseconds (minimum)
```

Figure 19

Control Sequence #2

Output a specific 5-bit pattern in response to a specified 3-bit input field.

Subprogram Calls and Returns

The mechanism for managing subprogram calls and returns is based on assigning a return link value to each subprogram caller; this return link value is then used, on exit from the subprogram, to index into the return jump table which returns control to the callers of the subprogram. Figure 21 is an example of a subprogram called from four different locations in the main program.

As seen from Figure 21, each subprogram (or procedure) caller is assigned a "tag" or index values ranging from 0 to 3, or a total of four index values for the four callers. Before jumping to the subprogram, the index value is placed in a previously agreed upon location, register R11 in this case. Upon exit from the subroutine, the index value stored in R11 is used as an offset to the Program Counter in order to execute the proper JMP instruction. The key to returning to the proper caller is the index jump table. Figure 22 gives a detailed description of the return operation.

CONTROL SEQUENCE #2 PROGRAM

```
        XMIT   STATUS,IVI      Choose the IV byte which receives the 3-bit input from
                               user's system.
        MOVE   STATUS,R1       Move the 3 bits of interest from the IV byte to register R1
                               The 3 bits are automatically right justified
        XMIT   ALARM,IVL       Choose the IV byte through which the response is sent to
                               the user's system.
        XEC    PATTERN(R1),8   Select specific pattern from PATTERN table.
        JMP    *+9
PATTERN XMIT   A,ALARM         Transmit proper pattern to output IV byte subfield by execut-
        XMIT   B,ALARM         ing just one of these eight instructions. A through H repre-
        XMIT   C,ALARM         sent the names associated with eight different control bit
        •  •   •               patterns.
        •  •   •
        •  •   •
        XMIT   H,ALARM
        •  •
        •  •
        •  •
TIME: 1.25 microseconds.
```

Figure 20

RETURN OPERATION

```
Address n      XEC* + 1        This instruction results in the execution of the instruction
                               located at the current value of the Program counter p
                               plus 1 plus the contents of R11, which is the caller index
                               value.
Address n+1    JMP A
Address n+2    JMP B           The JMP table follows in consecutive Program Storage
                               locations following SLC
Address n+3    JMP C
Address n+4    JMP D
```

Figure 21

SUBPROGRAM CALL

```
Program
Storage
Address                                          Instruction
000137₈   XMIT 0, R11          Load AUX with 0 Caller # 1
000140₈   JMP SUBR             Jump to start of subprogram
000141₈   Next Instruction
          •
          •
          •
          •
001133₈   XMIT 1, R11          Load AUX with 1 Caller # 2
001134₈   JMP SUBR             Jump to start of subprogram
001135₈   Next Instruction
          •
          • •
          •
003260₈   XMIT 2, R11          Load AUX with 2 Caller # 3
003261₈   JMP SUBR             Jump to start of subprogram
003262₈   Next Instruction
          •
          •
          •
          •
003654₈   XMIT 3, R11          Load AUX with 3 Caller #4
003655₈   JMP SUBR             Jump to start of subprogram
003656₈   Next Instruction
          •
          •
          •
          •
                                a. Main Program
SUBR      Machine Instructions
          JMP  TABL
          •
          •
          •
          •
Subroutine
Return
Code
TABL      XEC    *+1(R11)      Execute JMP located at
                               current PC + 1 + (R11).
          JMP    000141₈       Return to Caller #1
          JMP    001135₈       Return to Caller #2
          JMP    003262₈       Return to Caller #3
          JMP    003656₈       Return to Caller #4
                                b. Subroutine
```

Figure 22

PROGRAMMING EXAMPLES

PROCEDURE NAME: TAD 16

General Description

TAD16 is a double precision (16-bit) 2's complement addition program which checks for arithmetic overflow by comparing the signs of the operands and the result. Overflow has occurred when and only when the operands have like signs and the result has the opposite sign. When overflow occurs the program returns the value 100000 base 8. This is the largest negative 16-bit 2's complement number. TAD16 requires that its two double precision operands always be found in the same four memory locations. These four locations can be anywhere on page 0 of working storage and do not have to be contiguous. All results are stored in the two working storage locations which originally held the second operand. See Figure 23 for the flow chart and Figure 24 for the program listing.

Memory Requirements:

Program Memory: 24 words
Working Storage: 4 bytes

Registers Used and Their Logical Function:

R1 This register is used to hold information on the signs of the operands. R1 contains 0 if both operands are positive, 2 if both operands are negative, or, 1 if the operands have opposite signs.

Symbols:

AL Low order byte of A
AH High order byte of A
A1 High order (sign) bit of A
BL Low order byte of B
BH High order byte of B
B1 High order (sign) bit of B

Timing

Worst Case: 5.25 microseconds when overflow occurs
Best Case: 3.75 microseconds when operands have opposite signs

Calls On Other Library Procedures: None

TAD16 FLOW CHART

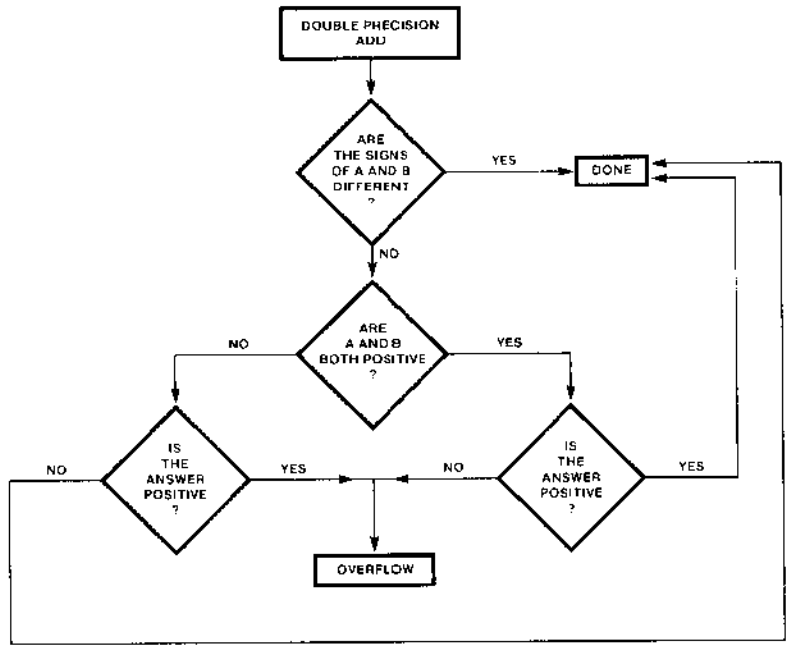


Figure 23

PROGRAM SAMPLE

```
  PROG SAMPLE              *8X300 ASSEMBLER VER 1.0    PAGE    1

   1                              PROG SAMPLE
   2 00000                       PROC TAD16
   3                      *
   4                      *     MCLIB PROCEDURE TO ADD TWO 16 BIT NUMBERS
   5                      *     IN 2'S COMPLEMENT NOTATION AND CHECK
   6                      *     FOR ARITHMETIC OVERFLOW.
   7                      *
   8                      *
   9                      *          DATA DECLARATIONS.
  10        310 7 0    AL  RIV 200,7,8
  11        311 7 0    AH  RIV 201,7,8
  12        311 0 1    A1  RIV 201,0,1
  13        312 7 0    BL  RIV 202,7,8
  14        313 7 0    BH  RIV 203,7,8
  15        313 0 1    B1  RIV 203,0,1
  16                     *
  17 00000 6 17311    TAD       SEL     A1
  18 00001 0 30100              MOVE    A1,AUX
  19 00002 6 17313              SEL     B1
  20 00003 1 30101              ADD     B1,R1
  21                     *                              *R1=0 IF BOTH=0.
  22                     *                              *R1=2 IF BOTH=1.
  23                     *                            *R1=1 IF BOTH DIFF.
  24 00004 6 17310              SEL     AL
  25 00005 0 37000              MOVE    AL,AUX
  26 00006 6 17312              SEL     BL
  27 00007 1 37037              ADD     BL,BL         *AL+BL NOW IN BL.
  28 00010 0 10000              MOVE       OVF,AUX
  29 00011 6 17311              SEL     AH
  30 00012 1 37000              ADD     AH,AUX
  31 00013 6 17313              SEL     BH
  32 00014 1 37037              ADD     BH,BH         *ANSWER IN BH..BL
  33                            ORG     4,256
  34 00015 4 01016              XEC     *+1(R1)
  35 00016 7 00021              JMP     ZEROS
  36 00017 7 00030              JMP     INBOUNDS
  37 00020 7 00023              JMP     ONES
  38                            ORG     4,32
  39 00021 5 30124    ZEROS     NZT     B1,OVERFLOW
  40 00022 7 00030              JMP     INBOUNDS
  41                            ORG     6,32
  42 00023 5 30130    ONES      NZT     B1,INBOUNDS
  43 00024 6 00200    OVERFLOW  XMIT    200H,AUX
  44 00025 0 00037              MOVE    AUX,BH
  45 00026 6 17312              SEL     BL
  46 00027 6 37000              XMIT    0,BL
  47 00030 7 00031    INBOUNDS  RTN
  48                            END TAD16
  49                           END SAMPLE


*TOTAL ASSEMBLY ERRORS =     0
```

Figure 24

PROCEDURE NAME: MUL8X8

General Description:

MUL8X8 is a procedure which multiplies two 8-bit 2's complement numbers. For reasons of speed, negative numbers are converted to positive numbers before the multiplication takes place. Afterward, the product is given the proper sign. The algorithm is a straight forward add and shift routine. The operands are taken from R1 and R2. The low order byte of the sixteen bit result is stored in R1. The high order byte is stored in R3. See Figure 25 for the flow chart and Figure 26 for the program listing.

Memory Requirements:

Program Storage: 47 words
Working Storage: None

Registers Used And Their Logical Function:

R1 Initially contains the multiplier. Eventually contains the low order byte of the product.
R2 Initially contains the multiplicand.
R3 Contains the high order byte of the result.
R4 Bit counter.
R5 Contains information on the sign of the result. R5 = 0 if the result is negative or R5 = 1 if the result is positive.

Timing:

Worst Case: 35.75 microseconds

Calls On Other Library Procedures: None

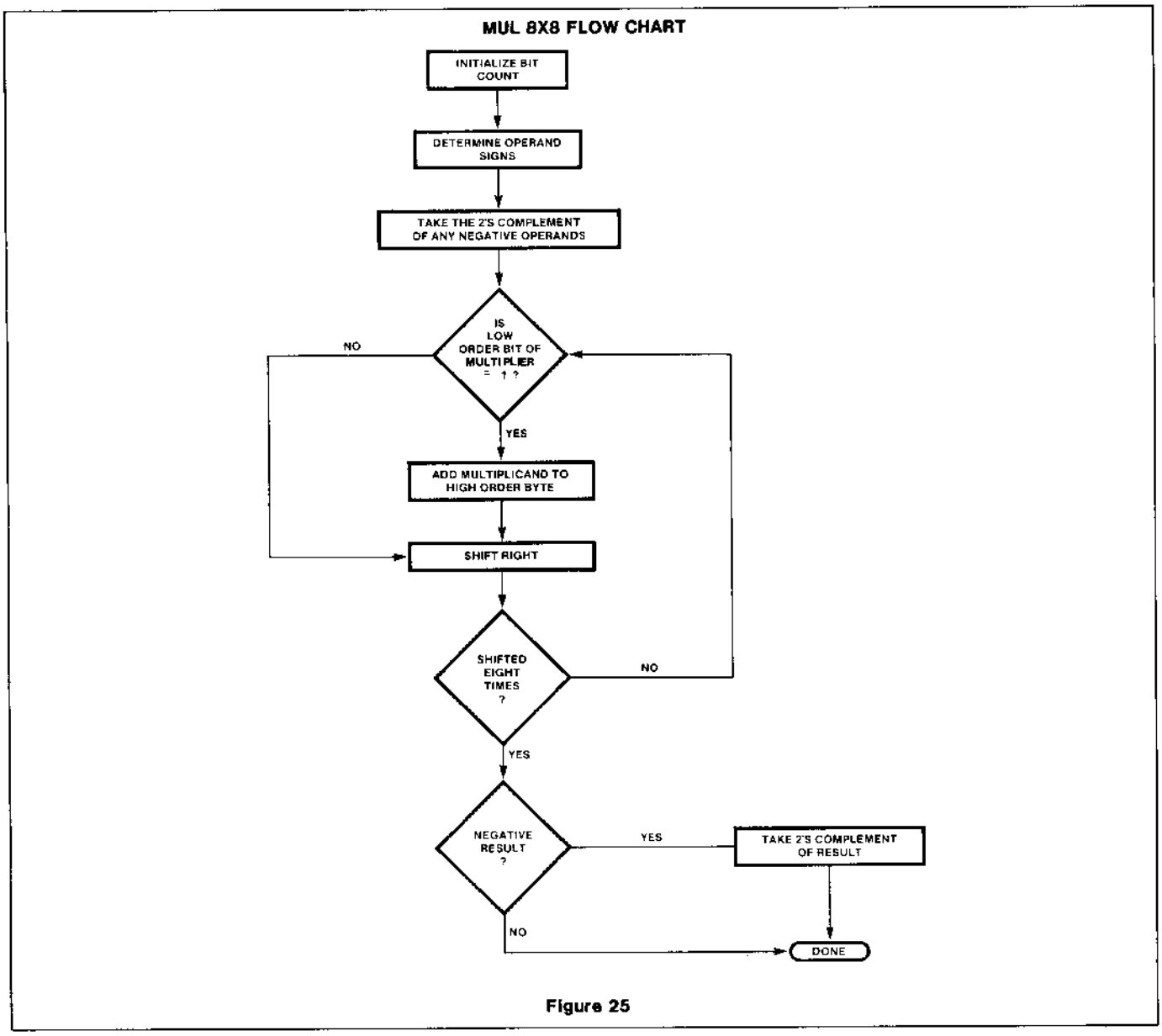


Figure 25

PROGRAM LISTING

```
PROG SAMPLE                *8X300 ASSEMBLER VER 1.0     PAGE    1

 1                              PROG SAMPLE
 2 00000                        PROC MUL8X8
 3                         *
 4                         *    MCLIB PROCEDURE TO MULTIPLY TWO 8 BIT 2'S
 5                         *    COMPLEMENT NUMBERS TO GENERATE A
 6                         *    16-BIT RESULT.
 7                         *
 8                         *
 9 00000 6 03000                XMIT    0,R3
10 00001 6 04370                XMIT    -8,R4
11 00002 6 00001                XMIT    1,AUX   ACCESS/TEST OPERAND1 SIGN.
12 00003 2 01700                AND     R1(7),AUX
13                              ORG      5,256
14 00004 4 00007                XEC     TAB(AUX)
15 00005 5 00011                NZT     AUX,COMP1
16 00006 7 00015                JMP     OP2
17 00007 6 05001   TAB          XMIT    1,R5        POSITIVE.
18 00010 6 05000                XMIT    0,R5        NEGATIVE.
19 00011 6 00377   COMP1 XMIT    377H,AUX  COMP OPERAND1.
20 00012 3 01001                XOR     R1,R1
21 00013 6 00001                 XMIT    1,AUX
22 00014 1 01001                ADD     R1,R1
23 00015 6 00001   OP2          XMIT   1,AUX   ACCESS/TEST OPERAND2 SIGN.
24 00016 2 02700                AND     R2(7),AUX
25 00017 3 05005                XOR     R5,R5
26 00020 5 00022                NZT     AUX,COMP2
27 00021 7 00026                JMP     LOOP
28 00022 6 00377   COMP2 XMIT   377H,AUX    COMP OPERAND2.
29 00023 3 02002                XOR     R2,R2
30 00024 6 00001                XMIT    1,AUX
31 00025 1 02002                ADD     R2,R2
32                              ORG     16,256
33 00026 6 00001   LOOP XMIT    1,AUX       LOW ORDER BIT 1?
34 00027 2 01000                AND     R1,AUX
35 00030 5 00032                NZT     AUX,*+2
36 00031 7 00034                JMP     SHIFT
37 00032 0 02000                MOVE    R2,AUX      YES. ADD MULTIPLICAND.
38 00033 1 03003                ADD     R3,R3
39 00034 6 00177   SHIFT XMIT   177H,AUX    SHIFT PARTIAL.
40 00035 2 01101                AND     R1(1),R1    PRODUCT RIGHT.
41 00036 6 00200                XMIT    200H,AUX
42 00037 2 03100                AND     R3(1),AUX
43 00040 3 01001                XOR     R1,R1
44 00041 6 00177                XMIT    177H,AUX
45 00042 2 03103                AND     R3(1),R3
46 00043 6 00001                XMIT    1,AUX
47 00044 1 04004                 ADD     R4,R4
48 00045 5 04026                NZT     R4,LOOP     DONE?
49                              ORG     9,256
50 00046 5 05056                NZT     R5,ENDD
51 00047 6 00377                XMIT    377H,AUX    NO.2'S COMP.
52 00050 3 01001                XOR     R1,R1
53 00051 3 03003                XOR     R3,R3
54 00052 6 00001                XMIT    1,AUX
PROG SAMPLE                        *8X300 ASSEMBLER VER 1.0     PAGE    2

55 00053 1 01001                ADD     R1,R1
56 00054 0 10000                MOVE    OVF,AUX
57 00055 1 03003                ADD     R3,R3
58 00056 7 00057   ENDD  RTN
59                              END     MUL8X8
60                              END    SAMPLE


*TOTAL ASSEMBLY ERRORS =     0
```

Figure 26

PROCEDURE NAME: SORT

General Description:

SORT is a procedure which sorts the contents of a block of Working Storage locations into descending order. That is, the data is sorted so that as the Working Storage addresses increase, the value of the contents decrease. The boundaries of the block are set by the main program. The lower address boundary must be placed in R1. The high address boundary must be placed in R2. The high address boundary must be placed in R2. The block must be contained within a single memory page and that page must be selected by the main program. The contents of the block are treated as 2's complement numbers. See Figure 27 for the flow chart and Figure 28 for the program listing.

Memory Requirements:

Program Storage: 47 words
Working Storage: None

Registers And Their Logical Function:

R1 This register is used to pass the lower boundary address to the procedure. In the course of execution, this value is changed to its 2's complement.

R2 This register is used to pass the upper boundary address to the procedure. In the course of execution, this value is changed to its 2's complement.

R3 This register is used to hold the current address, N.

R4 This register is used to hold the current contents of N which is denoted as (N).

R6 This register is used as a scratchpad to hold a variety of temporary results.

Timing:

It is difficult to compute the exact timing for this procedure. Six microseconds per byte sorted is a realistic average time.

Calls On Other Library Procedures: None

SORT FLOW CHART

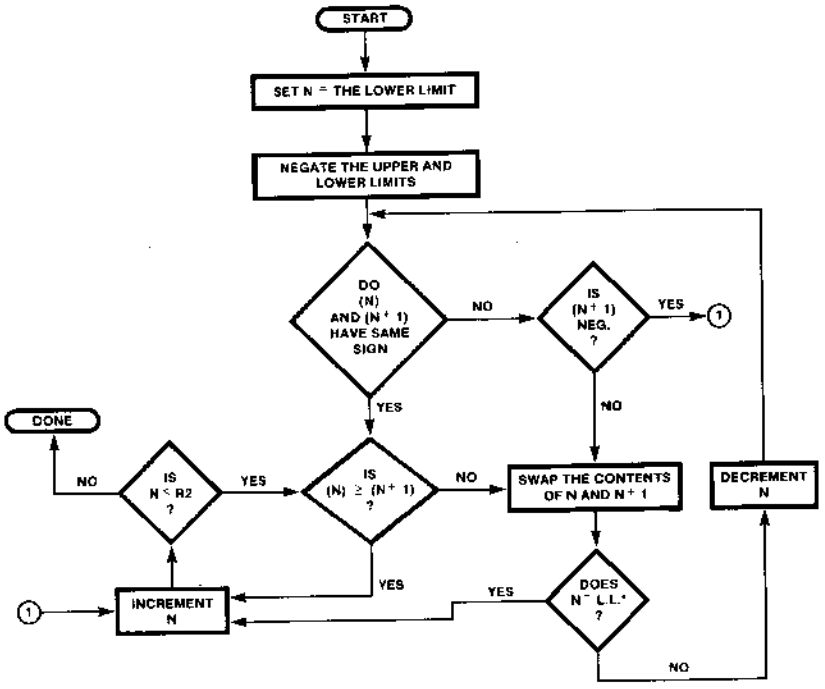


Figure 27

PROGRAM LISTING

```
 PROG SAMPLE                    *8X300 ASSEMBLER VER 1.0     PAGE     1

  1                                    PROG SAMPLE
  2 00000                               PROC SORT
  3                             *
  4                             *
  5                             *      MCLIB PROCEDURE TO SORT THE
  6                             *      CONTENTS OF A BLOCK OF WORKING
  7                             *      STORAGE LOCATIONS INTO DESCENDING
  8                             *      ORDER.
  9                             *
 10                             *
 11          310 7 0   DMY       RIV   200,7,8
 12          310 0 1   SIGN      RIV   200,0,1
 13 00000 0 01003      START     MOVE  R1,R3       R3=N.
 14 00001 6 00377                XMIT  -1,AUX      NEGATE R2.
 15 00002 1 02002                ADD   R2,R2
 16 00003 3 02002                XOR   R2,R2
 17 00004 1 01001                ADD   R1,R1       NEGATE R1.
 18 00005 3 01001                XOR   R1,R1
 19 00006 0 03017                MOVE  R3,IVR      ENABLE N.
 20 00007 0 37004                MOVE  DMY,R4      R4=(N).
 21                              ORG   21,256
 22 00010 6 00001      TEST      XMIT  1,AUX
 23 00011 1 03017                ADD   R3,IVR      ENABLE N+1.
 24 00012 2 04706                AND   R4(7),R6    R6=SIGN OF (N).
 25 00013 0 30100                MOVE  SIGN,AUX    AUX=SIGN OF (N+1).
 26 00014 3 06006                XOR   R6,R6       R6=1 IF DIFF.
 27                              ORG   18,256
 28 00015 5 06036                 NZT   R6,CHECK
 29 00016 6 00377                XMIT  -1,AUX
 30 00017 1 37005                 ADD   DMY,R6
 31 00020 3 06000                XOR   R6,AUX      -(N+1) NOW IN AUX.
 32 00021 1 04006                 ADD   R4,R6      (N)-(N+1) NOW IN R6.
 33 00022 6 00001                XMIT  1,AUX
 34 00023 2 06706                AND   R6(7),R6    R6=0, IF (N)=(N+1).
 35                              ORG   13,256
 36 00024 5 06040                NZT   R6,SWAP
 37                              ORG   10,256
 38 00025 1 03003      NEXTN     ADD   R3,R3       INCREMENT N.
 39 00026 0 03017                MOVE  R3,IVR
 40 00027 0 37004                MOVE  DMY,R4
 41 00030 0 02000                MOVE  R2,AUX      AUX=-UPPER LIMIT.
 42 00031 1 03005                ADD   R3,R6       R6=N-R2.
 43 00032 6 00001                XMIT  1,AUX
 44 00033 2 06706                AND   R6(7),R6     N-R2 MUST BE < = 0.
 45 00034 5 06010                NZT   R6,TEST     R6=0 IF N=R2.
 46 00035 7 00056                JMP   DONE
 47 00036 5 00025      CHECK     NZT   AUX,NEXTN   AUX=SIGN OF (N+1).
 48 00037 7 00040                JMP   SWAP
 49 00040 0 37006      SWAP      MOVE  DMY,R6      R6=(N+1).
 50 00041 0 04037                 MOVE  R4,DMY
 51 00042 0 03017                MOVE  R3,IVR      ENABLE N.
 52 00043 0 06037                MOVE  R6,DMY
 53 00044 0 01000                MOVE  R1,AUX
 54 00045 1 03006                ADD   R3,R6
   PROG SAMPLE                   *8X300 ASSEMBLER VER 1.0     PAGE     2

 55 00046 5 06051                NZT   R6,DECN
 56 00047 6 00001                XMIT  1,AUX
 57 00050 7 00025                JMP   NEXTN
 58 00051 6 00377      DECN      XMIT  -1,AUX
 59 00052 1 03003                ADD   R3,R3
 60 00053 0 03017                MOVE  R3,IVR
 61 00054 0 37004                MOVE  DMY,R4
 62 00055 7 00010                JMP   TEST
 63 00056 7 00057      DONE      RTN
 64                              END SORT
 65                              END SAMPLE


*TOTAL ASSEMBLY ERRORS =     0
```

Figure 28

SIGNETICS HEADQUARTERS

811 East Arques Avenue
Sunnyvale, California 94086
Phone: (408) 739-7700

ALABAMA
Huntsville
Phone: (205) 533-5800
ARIZONA
Phoenix
Phone: (602) 265-4444
CALIFORNIA
Inglewood
Phone: (213) 670-1101
Irvine
Phone: (714) 833-8980
(213) 924-1668
San Diego
Phone: (714) 560-0242
Sunnyvale
Phone: (408) 736-7565
Woodland Hills
Phone: (213) 340-1431
COLORADO
Aurora
Phone: (303) 751-5011
FLORIDA
Pompano Beach
Phone: (305) 782-8225
ILLINOIS
Rolling Meadows
Phone: (312) 259-8300
KANSAS
Wichita
Phone: (316) 683-5552
MASSACHUSETTS
Bedford
Phone: (617) 275-8900
MARYLAND
Columbia
Phone: (301) 730-8100
MINNESOTA
Edina
Phone: (612) 835-7455
NEW JERSEY
Cherry Hill
Phone: (609) 665-5071
Piscataway
Phone: (201) 981-0123
NEW YORK
Wappingers Falls
Phone: (914) 297-4074
Woodbury, L.I.
Phone: (516) 364-9100
OHIO
Worthington
Phone: (614) 888-7143
TEXAS
Dallas
Phone: (214) 661-1296
UTAH
Centerville
Phone: (801) 290-1292

REPRESENTATIVES

ALABAMA
Huntsville
Alpha Marketing
Phone: (205) 533-0766
CALIFORNIA
San Diego
Mesa Engineering
Phone: (714) 278-8021
Sherman Oaks
Astralonics
Phone: (213) 990-5903
CANADA
Montreal, Quebec
Philips Electronics Industries Ltd.
Phone: (515) 342-9180
Ottawa, Ontario
Philips Electronics Industries Ltd.
Phone: (613) 237-3131
Scarborough, Ontario
Philips Electronics Industries Ltd.
Phone: (416) 292-5161
Vancouver, B.C.
Philips Electronics Industries Ltd.
Phone: (604) 435-4411
Winnipeg, Manitoba
Phil-ps Electronics Industries, Ltd.
Phone: (204) 774-1931
CONNECTICUT
Danbury
Kanan Associates
Phone: (203) 743-1812
FLORIDA
Altamonte Springs
Semtronic Associates
Phone: (305) 831-8233
Clearwater
Semtronic Associates
Phone: (813) 461-4675
GEORGIA
Atlanta
Alpha Marketing
Phone: (404) 231-0534
ILLINOIS
Chicago
L-Tec, Inc.
Phone: (312) 286-1500
INDIANA
Fort Wayne
Insul-Reps, Inc.
Phone: (219) 482-1596
Indianapolis
Insul-Reps, Inc.
Phone: (317) 259-4432
KANSAS
Overland Park
Advanced Technical Sales
Phone: (913) 492-4333
MARYLAND
Baltimore
Micro-Comp, Inc.
Phone: (301) 247-8400
MASSACHUSETTS
Reading
Kanan Associates
Phone: (617) 944-8484
MICHIGAN
Bloomfield Hills
Enco Marketing
Phone: (313) 642-0203
MINNESOTA
Edina
Mel Foster Technical Sales
Phone: (612) 835-2252
MISSOURI
St. Louis
Advanced Technical Sales
Phone: (314) 567-6272
NEW JERSEY
Haddonfield
Thomas Assoc., Inc.
Phone: (609) 854-3011
NEW MEXICO
Albuquerque
The Staley Company, Inc.
Phone:(505) 292-0060
NEW YORK
Ithaca
Bob Dean, Inc.
Phone: (607) 272-2187
NORTH CAROLINA
Cary
Montgomery Marketing
Phone: (919) 467-6319
Charlotte
Montgomery Marketing
Phone: (704) 535-2400
OHIO
Centerville
Norm Case Associates
Phone: (513) 433-0966
Rocky River
Norm Case Associates
Phone: (216) 333-4120
OREGON
Portland
Western Technical Sales
Phone: (503) 297-1711
TEXAS
Austin
Cunningham Company
Phone: (512) 459-8947
Dallas
Cunningham Company
Phone: (214) 233-4303
Houston
Cunningham Company
Phone: (713) 683-7231
UTAH
West Bountiful
Barnhill Five, Inc.
Phone: (801) 292-8991
WASHINGTON
Bellevue
Western Technical Sales
Phone: (206) 641-3900
WISCONSIN
Wauwatosa
L-Tec, Inc.
Phone: (414) 774-1000

DISTRIBUTORS

ALABAMA
Huntsville
Hamilton/Avnet Electronics
Phone: (205) 533-1170
Pioneer Electronics
Phone: (205) 837 9300
ARIZONA
Phoenix
Hamilton/Avnet Electronics
Phone: (602) 275-7851
Liberty Electronics
Phone: (602) 249-2232
CALIFORNIA
Costa Mesa
Avnet Electronics
Phone: (714) 754-6051
Culver City
Hamilton Electro Sales
Phone: (714) 558-2183
El Segundo
Liberty Electronics
Phone: (213) 322-8100
Irvine
Schweber Electronics
Phone: (714) 556-3880
Mountain View
Elmar Electronics
Phone: (415) 961-3611
Hamilton/Avnet Electronics
Phone: (415) 961-7000
San Diego
Hamilton/Avnet Electronics
Phone: (714) 279-2421
Liberty Electronics
Phone: (714) 565-9171
Sunnyvale
Intermark Electronics
Phone: (408) 738-1111
CANADA
Downsview, Ontario
Cesco Electronics
Phone: (416) 661-0220
Zentronics
Phone: (416) 635-2822
Mississauga, Ontario
Hamilton/Avnet Electronics
Phone: (416) 677-7432
Montreal, Quebec
Cesco Electronics
Phone: (514) 735-5511
Zentronics Ltd.
Phone: (514) 735-5361
Ottawa, Ontario
Cesco Electronics
Phone: (613) 729-5118
Hamilton/Avnet Electronics
Phone: (613) 226-1700
Zentronics Ltd.
Phone: (613) 238-6411
Quebec City
Cesco Electronics
Phone: (418) 524-4641
Vancouver, B.C.
Bowtek Electronics Co., Ltd.
Phone: (604) 736-1141
Ville St. Laurent, Quebec
Hamilton/Avnet Electronics
Phone: (514) 331-6443
COLORADO
Commerce City
Elmar Electronics
Phone: (303) 287-9611
Denver
Hamilton/Avnet Electronics
Phone: (303) 534-1212
CONNECTICUT
Danbury
Schweber Electronics
Phone: (203) 792-3500
Georgetown
Hamilton/Avnet Electronics
Phone: (203) 762-0361
Hamden
Arrow Electronics
Phone: (203) 248-3801
FLORIDA
Ft. Lauderdale
Arrow Electronics
Phone: (305) 776-7790
Hamilton/Avnet Electronics
Phone: (305) 971-2800
Hollywood
Schweber Electronics
Phone: (305) 927-0511
Melbourne
Arrow Electronics
Phone: (305) 725-1480
Orlando
Hammond Electronics
Phone: (305) 241-6601
GEORGIA
Atlanta
Arrow Electronics
Phone: (404) 455-4054
Schweber Electronics
Phone: (404) 449-9170
Doraville
Arrow Electronics
Phone: (404) 455-4054
Norcross
Hamilton/Avnet Electronics
Phone: (404) 448-0800
ILLINOIS
Chicago
Arrow Electronics
Phone: (312) 893-9420
Bell Industries
Phone: (312) 982-9210
Elk Grove
Schweber Electronics
Phone: (312) 593-2740
Schiller Park
Hamilton/Avnet Electronics
Phone: (312) 671-6094
INDIANA
Indianapolis
Pioneer Electronics
Phone: (317) 849-7300
KANSAS
Lenexa
Hamilton/Avnet Electronics
Phone: (913) 888-8900
MARYLAND
Baltimore
Arrow Electronics
Phone: (301) 247-5200
Gaithersburg
Pioneer Washington Electronics
Phone: (301) 948-0710
Schweber Electronics
Phone: (301) 840 5900
Hanover
Hamilton/Avnet Electronics
Phone: (301) 796-5000
MASSACHUSETTS
Waltham
Schweber Electronics
Phone: (617) 890-8484
Woburn
Arrow Electronics
Phone: (617) 933-8130
Hamilton/Avnet Electronics
Phone: (617) 933-8000
MICHIGAN
Livonia
Hamilton/Avnet Electronics
Phone: (313) 522-4700
Pioneer Electronics
Phone: (313) 525-1800
Troy
Schweber Electronics
Phone: (313) 583-9242
MINNESOTA
Bloomington
Arrow Electronics
Phone: (612) 887-6400
Eden Prairie
Schweber Electronics
Phone: (612) 941-5280
Edina
Hamilton/Avnet Electronics
Phone: (612) 941-3801
MISSOURI
Hazelwood
Hamilton/Avnet Electronics
Phone: (314) 731-1144
NEW MEXICO
Albuquerque
Hamilton/Avnet Electronics
Phone: (505) 765-1500
NEW YORK
Buffalo
Summit Distributors
Phone: (716) 884-3450
East Syracuse
Hamilton/Avnet Electronics
Phone: (315) 437-2642
Farmingdale, L.I.
Arrow Electronics
Phone: (516) 694-6800
Rochester
Hamilton/Avnet Electronics
Phone: (716) 442-7820
Schweber Electronics
Phone: (716) 424-2222
Westbury, L.I.
Hamilton/Avnet Electronics
Phone: (516) 333-5800
Schweber Electronics
Phone: (516) 334-7474
NEW JERSEY
Cedar Grove
Hamilton/Avnet Electronics
Phone: (210) 239-0800
Cherry Hill
Milgray-Delaware Valley
Phone: (609) 424-1300
Moorestown
Arrow/Angus Electronics
Phone: (609) 235-1900
Mt. Laurel
Hamilton/Avnet Electronics
Phone: (609) 234-2133
Saddlebrook
Arrow Electronics
Phone: (201) 797-5800
Somerset
Schweber Electronics
Phone: (201) 469-6008
PENNSYLVANIA
Horsham
Schweber Electronics
Phone: (215) 441-0600
Pittsburgh
Pioneer/Pittsburgh
Phone: (412) 782-2300
NORTH CAROLINA
Greensboro
Hammond Electronics
Phone: (919) 275-6393
Pioneer Electronics
Phone: (919) 273-4441
Raleigh
Hamilton/Avnet Electronics
Phone: (919) 829-8030
OHIO
Beachwood
Schweber Electronics
Phone: (216) 464-2970
Cleveland
Arrow Electronics
Phone: (216) 464-2000
Hamilton/Avnet Electronics
Phone: (216) 461-1400
Pioneer Standard Electronics
Phone: (216) 587-3600
Dayton
Hamilton/Avnet Electronics
Phone: (513) 433-0610
Pioneer Standard Electronics
Phone: (513) 236-9900
Kettering
Arrow Electronics
Phone: (513) 253-9176
OKLAHOMA
Tulsa
Component Specialties
Phone: (918) 664-2820
TEXAS
Dallas
Component Specialists
Phone: (214) 357-6511
Hamilton/Avnet Electronics
Phone: (214) 661-8204
Quality Components
Phone: (214) 387-4949
Schweber Electronics
Phone: (214) 661-5010
Houston
Component Specialists
Phone: (713) 771-7237
Hamilton/Avnet Electronics
Phone: (713) 780-1771
Quality Components
Phone: (713) 772-7100
Schweber Electronics
Phone: (713) 784-3600
UTAH
Salt Lake City
Alta Electronics
Phone: (801) 486-4134
Hamilton/Avnet Electronics
Phone: (801) 972-4300
WASHINGTON
Bellevue
Hamilton/Avnet Electronics
Phone: (206) 746-8750
Liberty Electronics
Phone: (206) 453-8300
WISCONSIN
Milwaukee
Arrow Electronics
Phone: (414) 764-6600
New Berlin
Hamilton/Avnet Electronics
Phone: (414) 784-4510

FOR SIGNETICS PRODUCTS WORLDWIDE

ARGENTINA
Fapesa I.y.C.
Buenos-Aires
Phone: 652-7436/7478
AUSTRIA
Osterreichische Philips
Wien
Phone: 93 26 11
AUSTRALIA
Philips Industries-ELCOMA
Lane-Cove N.S.W.
Phone: (02) 427-0888
Queensland
Brisbane
(07) 277-3332
South Australia
Adelaide
(08) 223-4022
Victoria
Melbourne
(03) 699-0300
Western Australia
Perth
(09) 277-4199
BELGIUM
M.B.L.E.
Brussels
Phone: 523 00 00
BRAZIL
Ibrape, S.A.
Sao Paulo
Phone: 284-4511
CANADA
Philips Electron Devices
Toronto
Phone: 425-5161
CHILE
Philips Chilena S.A.
Santiago
Phone: 39-4001
COLOMBIA
Sadapa S.A.
Bogota
Phone: 600600
DENMARK
Miniwatt A/S
Kobenhavn
Phone: (01) 69 16 22
FINLAND
Oy Philips Ab
Helsinki
Phone: 1 72 71
FRANCE
R.T.C.
Paris
Phone: 355-44-99
GERMANY
Valvo
Hamburg
Phone: (040) 3296-1
HONG KONG
Philips Hong Kong, Ltd.
Hong Kong
Phone: 12-245121
INDIA
Semiconductors, Ltd.
(REPRESENTATIVE ONLY)
Bombay
Phone: 293-667
INDONESIA
P. T. Philips-Ralin Electronics
Jakarta
Phone: 581058
IRAN
Berkeh Company, Ltd.
Tehran
Phone: 831564
ISRAEL
Rapac Electronics, Ltd.
Tel Aviv
Phone: 477115-6-7
ITALY
Philips S.p.A.
Milano
Phone: 2-6994
JAPAN
Signetics Japan, Ltd.
Tokyo
Phone: (03) 230-1521
KOREA
Philips Elect Korea Ltd.
Seoul
Phone: 44-4202
MEXICO
Electronica S.A. de C.V.
Mexico D.F.
Phone: 533-1180
NETHERLANDS
Philips Nederland B.V.
Eindhoven
Phone: (040) 79 33 33
NEW ZEALAND
Philips Electrical Ind. ELCOMA
Auckland
Phone: 867119
NORWAY
Electronica A.S.
Oslo
Phone: (02) 15 05 90
PAKISTAN
Elmac Ltd
Karachi
Phone: 515-122
PERU
Cadesa
Lima
Phone: 628599
PHILIPPINES
Philips Industrial Dev., Inc.
Makala-Rizal
Phone: 868951-9
SINGAPURE/MALAYSIA
Philips Singapore Pte., Ltd.
Singapore
Phone: 538811
SOUTH AFRICA
E.D.A.C. (PTY), Ltd.
Johannesburg
Phone: 24-6701-3
SPAIN
Copresa S.A.
Barcelona
Phone: 329 63 12
SWEDEN
Elcoma A.B.
Stockholm
Phone: 08/67 97 80
SWITZERLAND
Philips A.G.
Zurich
Phone: 01/44 22 11
TAIWAN
Philips Taiwan, Ltd.
Taipei
Phone: (02) 551-3101-5
THAILAND
Saeng Thong Radio, Ltd.
Bangkok
Phone: 252-7195, 252-7395
UNITED KINGDOM
Mullard, Ltd.
London
Phone: 01-580 6633
UNITED STATES
Signetics International Corp.
Sunnyvale, California
Phone: (408) 739-7700
URUGUAY
Luzilectron S.A.
Montevideo
Phone: 9143 21
VENEZUELA
Industrias Venezolanas
Philips S.A.
Caracas
Phone: 360-511

signetics

a subsidiary of **U.S. Philips Corporation**

Signetics Corporation
P.O. Box 9052
811 East Arques Avenue
Sunnyvale, California 94086
Telephone 408/739-7700

Printed in USA June 1978

signetics

November 14, 1978

Dear Field Sales People;

The 8X300 Programming Manual has a few errors in its present form. Please note the changes listed below. Thank you.

Larry Leppert

-- 8X300 PROGRAMMING MANUAL ERRATA --

PAGE NO. 33:

Under the boldface type in the upper right corner which reads "Add Instructions -- Op Code 1", the next line should read "Add, Register, IV Bus" instead of "Add, IV Bus, Register."

PAGE NO. 34 and 36:

Have been interchanged; that is, the text on page 34 should appear on page 36 and the text on page 36 should appear on page 34. In addition, on the existing page 36, the line in the upper left corner which reads "Add, Register, IV Bus" should read "Add, IV Bus, Register."

PAGE NO. 75:

The assembler notation in the upper right corner should read "XMIT 053H, IVL", instead of "XMIT 027H, IVL." The result should refer to "the byte at address 53_8" rather than "the Byte at Address 27_8."