

Microprogrammed Control Unit (MCU) Programming Reference Manual

JOHN D. ROBERTS, JR.

*Information Systems Group
Office of Associate Director of Research for Electronics*

August 15, 1972



NAVAL RESEARCH LABORATORY
Washington, D.C.

CONTENTS

Abstract	ii
Problem Status	ii
Authorization	ii
Acknowledgments	iii
NRL SIGNAL PROCESSING ELEMENT (SPE)	1
MICROPROGRAMMED CONTROL UNIT (MCU)	2
MCU Architecture	3
MCU Control Word Fields	7
AMIL	9
General Definitions	9
Instructions	10
Memory Operation	11
Adder Op	12
Address Adjust	14
Auxiliary Transfer	15
Condition Phrase	15
Successor	16
I/O Instruction	17
GLOSSARY OF ACRONYMS	18
APPENDIX A – Input/Output Control	19
APPENDIX B – A Microprogramming Language (AMIL) Syntax	26
APPENDIX C – AMIL Key Words	29
APPENDIX D – AMIL Sample Programs and Listings	32
APPENDIX E – AMIL Translator	38

ABSTRACT

The Microprogrammed Control Unit (MCU) is a high-speed, user-microprogrammable, executive, input-output processor and interrupt handler for the NRL Signal Processing Element (SPE), a part of the All Applications Digital Computer (AADC).

This report describes the MCU architecture, a new programming language (AMIL), and its translator. AMIL (A Microprogramming Language) is a FORTRAN-like language which allows MCU users to write microprograms in a convenient format instead of binary bit patterns. The AMIL translator converts AMIL programs into the micro-instruction bit patterns which the MCU will execute. This program marks the start of MCU software development.

PROBLEM STATUS

This is a report on one phase of the problem; work is continuing on other phases.

AUTHORIZATION

NRL Problem B02-06
NAVAIR Tasks 3F21-241-601 and 2F00-241-601
and
NRL Problem B02-10
NAVELEX Task XF21.241.015.K152, formerly XF53.241.003.K032

Manuscript submitted August 7, 1972.

ACKNOWLEDGMENTS

The architectural design of the MCU described in this report was performed by J. Ihnat, W. R. Smith, Jr., and the author. Appendix A was written by Smith.

The original MCU conceptual design and considerable valuable guidance were received from Y. S. Wu, manager of the Signal Processing Element project, and B. Wald, head of the Information Systems Group.

**MICROPROGRAMMED CONTROL UNIT (MCU)
PROGRAMMING REFERENCE MANUAL**

NRL SIGNAL PROCESSING ELEMENT (SPE)

The MCU (Microprogrammed Control Unit) is a high-speed, executive, input-output (I/O) processor and interrupt handler for the NRL Signal Processing Element (SPE), a part of the All Applications Digital Computer (AADC). It is the job of the MCU to supervise all SPE elements and to direct and initiate all data transfers between these elements. Referring to Fig. 1, one can see that the MCU directs the SPAU (Signal Processing Arithmetic Unit) and the CC's (Channel Controllers) via the Z bus, and communicates with up to 8 buffer memories over its two channels to the SCU (Storage Control Unit). To better understand the importance of the MCU, one needs to briefly examine the functions of the other SPE elements.

Signal Processing Arithmetic Unit (SPAU) — The SPAU is an extremely high-speed, special-purpose arithmetic unit which is entirely self-contained and capable of doing a Fast Fourier Transform on n points in only $n/2 \log_2 n$ basic clock cycles of 300 nsec each. This action is initiated over the Z bus by the MCU but is self-sustained and

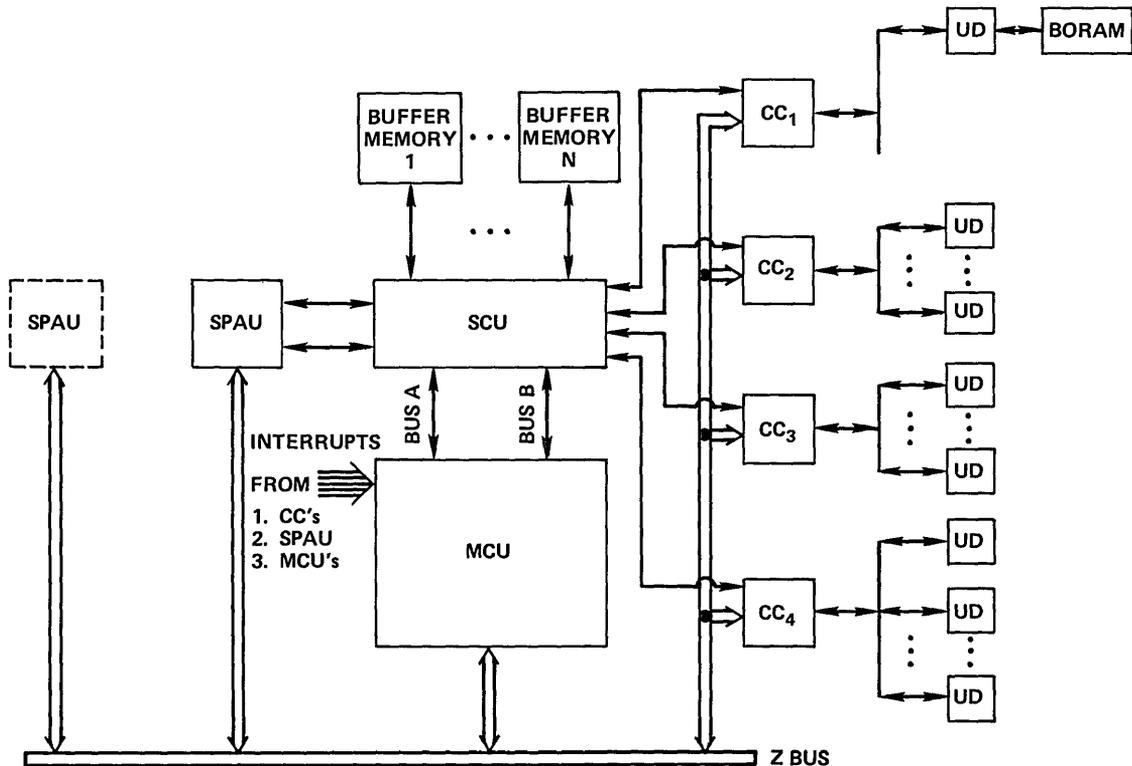


Fig. 1 - SPE system configuration

requires no further MCU intervention until it has finished. Inherent in this capability is the SPAU's ability to generate buffer memory addresses.

Additional functions of the SPAU include filtering and high-speed multiplies. Because of the high-speed capability of the SPAU, it has been supplied with two channels into the SCU and, therefore, the buffer memories.

Storage Control Unit (SCU) — It is the duty of the Storage Control Unit to resolve and assign all buffer memory requests during each 150-nsec clock cycle of the system. Eight channels each with a different priority enter the SCU. Each one is capable of requesting a specific buffer memory every system clock cycle, since the cycle time for any buffer is also only 150 nsec. The SCU must resolve all of these requests and assign the available buffer memories according to the request priorities. If duplicate buffer requests occur, the lower priority request waits. The priorities are obtained by physical position of the channel connection with the SCU.

Channel Controller (CC) — Channel controllers are included to allow devices to send and receive data from buffer memories without having to interrupt the MCU for each transmitted character (byte). When the MCU wants to perform an I/O operation, it sends out a series of signals over the Z bus which causes the CC connected to the requested device to respond with the status of said device. Upon receipt of an acceptable status, the MCU then sends the command and necessary data to the CC. The CC then bears the responsibility for completing the I/O requested. It must set up the necessary flags, counters, and status registers so that it can multiplex data from the current device along with devices it is already servicing. The CC requests only byte transfers between itself and unbuffered devices (UD) and must therefore do the buffering before it generates the appropriate buffer address and requests a memory cycle from the SCU. In summary, the CC is a buffered multiplexor or selector channel which sends and receives data from buffer memories and sends and receives commands and status information from the MCU.

MICROPROGRAMMED CONTROL UNIT (MCU)

The MCU is the microprogrammable executive for the SPE. Users will write microprograms (or have them written) which will direct and control all elements of the SPE. It will be the responsibility of the MCU to initiate and keep records of all I/O operations. Concurrently, it may be doing preprocessing on a block of data before requesting action from the SPAU. Similarly, it may have to do postprocessing of SPAU output before outputting the results or sending them back to the SPAU for yet another operation. In addition to these functions, the MCU must service the interrupts from the SPAU, CC's, and other MCU's, if any. To handle all of these responsibilities, it is necessary for the MCU to do many things at a very fast rate. As a result, the MCU operates at a 150-nsec clock cycle time, with the ability to do all operations, including buffer memory accesses, within one cycle. To achieve this high rate of control, the MCU operates from a single-format, 64-bit-wide, microprogram control word. From this wide control word, we achieve benefits such as increased speed due to the highly decoded fields and high hardware utilization (and, therefore, performance improvement) from the ability to control all of the registers and gates during each cycle.

We have seen that the MCU is the SPE controller, and that the microprogram control words control the MCU. Let us now examine the MCU architecture which is controlled by these microprograms.

MCU Architecture

To obtain fast basic clock rates, the MCU must be lean, but to do the required work it must have sufficient support hardware. These are the requirements which dictate the design as shown in Fig. 2 and the timing in Fig. 3.

All data entering or leaving the MCU travels over one of two channels, Bus A or Bus B, to the buffer memory. Each channel can be used for one memory operation during every MCU cycle due to the matched speed of the MCU and buffer memories. MCU register and data path widths are given in Table 1.

Table 1
MCU Register and Data Path Widths

<u>Register</u>	<u>Width</u>	<u>Comment</u>			
LSA	16				
LSB	16				
CTR	16				
SAR	4	Can shift from 0 to 15 bits			
CSAR	12	Control Store addresses can be from 0 to 4095			
ACSAR	12				
Z	16				
		Left Right			
BARA	16 } 16 }	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="width: 30px; text-align: center;">3</td> <td style="width: 60px; text-align: center;">12</td> <td style="width: 30px; text-align: center;">1</td> </tr> </table>	3	12	1
3	12	1			
BARB		3 - Buffer Address (BM 0 to 7) 12 - Word Address to 32 bit word 1 - Half Word Address - which 16 bits of above 32 bit word			
FSDR	32	Able to capture full 32 bits of BUSA			
<u>Data Path</u>	<u>Width</u>	<u>Comment</u>			
Local Store input	16	Bus A and Bus B <i>inputs</i> are only 16 bits			
Local Store address	4	Control Store fields or least significant 4 bits of BARA or BARB			
Bus A	32				
Bus B	32				
FSU output	16	Field selected is right justified			
Adder inputs	16	Leading zeros are inserted where necessary			
Adder output	16				
Barrel switch input	16				
Barrel switch output	16				
Z output	16	All sources for Z output receive only the right most number of bits desired.			

One support register, the SAR, has already been mentioned. Another is the Counter (CTR) which can be loaded with a literal value and counted up to overflow which can be checked and thus cause appropriate action. Other conditions that can be checked are based on results of the last adder operation and include adder overflow, result equal to zero, Z register most significant bit set (sign), and Z register least significant bit set (odd or even, flag, etc.).

One may conditionally or unconditionally alter the instruction execution sequence which is controlled by the test logic and next address selection logic. Two registers are provided for control store address selection. The Control Store Address Register (CSAR) is the only one which addresses the writable Control Store. It can be set from the other address selection register, the Alternate Control Store Address Register (ACSAR), the literal field of the control word, or from its incrementer. In addition to these, the Interrupt Control Unit (ICU) can set the CSAR to allow for interrupt handling. At the beginning of each cycle, the CSAR contains the address of the currently executing control word. Under direction of the new control word, the CSAR and ACSAR are selectively altered from one of eight choices. For example, in normal sequential program stepping, the CSAR is incremented during each clock cycle and the ACSAR is unchanged. For subroutine calls the ACSAR retains the return address (the old CSAR + 1) and the CSAR holds the address of the subroutine.

The Interrupt Control Unit (ICU) mentioned earlier contains no programmable elements. Upon receipt of an interrupt of higher priority than the current level executing in the MCU, the MCU operations are suspended, all necessary registers are saved, and the appropriate interrupt handling routine address is passed to the CSAR. This routine executes then restores the MCU to its preinterrupt status. The user will be unaware of this action except for deviations in expected execution times.

I/O action is initiated by the MCU by sending out an I/O command over the Z bus. The programmer must select the proper command operation code, count, device address, buffer address, etc., to be sent out on the Z bus. This process will be discussed further in Appendix A.

The last element of the MCU is the Field Select Unit (FSU). This device allows the programmer to address fields within a word. As data are brought in over bus A, the programmer may specify that during any transfer the 32 bits of data also be put into the Field Select Data Register (FSDR). In subsequent cycles after this operation, the user may select one of seven predefined fields from the FSDR as an operand for the adder. The output will be a 16-bit value with the selected field right justified with leading zeroes.

For example, the FSDR might contain two 16-bit data words (4 bytes) and the field definitions could be on byte boundaries (4) or word boundaries as shown below.

FSDR

Byte 1	Byte 2	Byte 3	Byte 4
8 bits	8	8	8

- FSU (1) implies byte 1 right justified with 8 leading zeroes
- FSU (2) implies byte 2
- FSU (3) and (4) implies byte 3 and 4
- FSU (5) implies byte 1 and byte 2 (left half, 16 bits)
- FSU (6) implies byte 3 and byte 4 (right half, 16 bits)

With this byte addressing capability, the MCU in many cases will only need to do one memory operation for every 4 bytes of data, thus increasing the effective memory bandwidth.

One important attribute of the MCU is flexibility. This is achieved as a result of microprogrammed control. The MCU is controlled from a 4096- by 64-bit Control Store. Each 64-bit-wide word is a single-format highly decoded command. With a single-format, as compared to a multiple-format machine, each bit or series of bits must activate the same paths during every MCU cycle, which reduces the hardware required for control. Highly decoded fields increase the MCU's speed by the nontrivial amount of time it takes to decode highly encoded fields. We shall now examine the microprogrammed control.

MCU Control Word Fields

The 64-bit MCU control word is composed of 17 fields as given below:

<u>Field I.D.</u>	<u>Position (bit #)</u>	<u>Function</u>
COND	1-3	Condition test select
NOT	4	True or False of condition
NEXT	5-7	Next control store address selection
SAUX	8-10	Set auxiliary register
SARA	11-12	Source for buffer address register A (BARA)
SARB	13-14	Source for buffer address register B (BARB)
SLSA	15-16	Source for Local Store A input
SLSB	17-18	Source for Local Store B input
WBUF	19-21	Write Buffer Selection
FSU	22-24	Field Select Unit output selection
ADIN	25-28	Adder input selection
ADOP	29-32	Adder/Shifter Operation selection
SAAL	33-36	Source Address Literal for LSA
SBAL	37-40	Source Address Literal for LSB
DAAL	41-44	Destination Address Literal for LSA
DBAL	45-48	Destination Address Literal for LSB
LIT	49-64	Literal (all-purpose data, address)

Each of these fields is further broken down into bit responsibility by the following chart. (Code point actions are summarized using key words of AMIL, see Appendix C.)

MCU Control Field Definitions

<u>COND(1)</u>		<u>NOT(2)</u>		<u>NEXT(3)</u>		<u>SAUX(4)</u>		<u>SARA(5)</u>	
0	None	0	True	0	STEP	0	None	0	None
1	MOST	1	False	1	SKIP	1	ACSAR = LIT	1	-1
2	LEAST			2	SAVE	2	ACSAR = Z	2	+1
3	ADROV			3	CALL	3	SAR = LIT	3	Z
4	CTROV			4	JUMPL	4	SAR = Z		
5	ZERO			5	JUMPA	5	CTR = LIT		
6	extra			6	JUMPZ	6	CTR = Z		
7	I/O			7	INTRET	7	extra		
<u>SARB(6)</u>		<u>SLSA(7)</u>		<u>SLSB(8)</u>		<u>WBUF(9)</u>		<u>FSU(10)</u>	
0	None	0	None	0	None	0	No Write	0	FSDR = BUF(BARA)
1	-1	1	BUF(BARA)	1	BUF(BARA)	1	BUF(BARA)= LSA	1	Field 1
2	+1	2	BUF(BARB)	2	BUF(BARB)	2	BUF(BARB)= LSA	2	Field 2
3	Z	3	Z	3	Z	3	BUF(BARA)= LSB	3	Field 3
						4	BUF(BARB)= LSB	4	Field 4
						5	BUF(BARA)= Z	5	Field 5
						6	BUF(BARB)= Z	6	Field 6
						7	BUF(BARA)= LSA	7	Field 7
							BUF(BARB)= LSB		
<u>ADIN(11)</u>				<u>ADOP(12)</u>				<u>SAAL(13)</u>	
	Left		Right						
0	LSA		LIT	0	No Op			0	Use BARA register
1	LSA		FSU	1	Z = L + R				to select LSA (i)
2	LSA		BARB	2	Z = L - R			1 to 15	Select LSA reg.
3	LSA		LSB	3	Z = R - L				specified
4	LSB		LIT	4	Z = L + 1				
5	LSB		FSU	5	Z = L - 1				
6	LSB		BARB	6	Z = L				
7	LSB		BARA	7	Z = \bar{L}				
8	BARA		LIT	8	Z = R				
9	BARA		LSA	9	Z = L $\bar{\vee}$ R				
10	BARB		LIT	10	Z = L AND R				
11	BARB		BARA	11	Z = L XOR R				
12	FSU		LIT	12	Z = L EQV R				
13	CTR		LIT	13	Z = L Left Shifted				
14	ACSAR		LIT	14	Z = L Right Shifted				
15	SAR		LIT	15	Z = L Circularly Shifted				
<u>SBAL(14)</u>				<u>DAAL(15)</u>		<u>DBAL(16)</u>		<u>LIT(17)</u>	
0	Use BARB register			Same as	SAAL	Same as	SBAL	A 16 bit integer	
	to select LSB(i)							value	
1 to 15	Select LSB								
	register								
	specified								

If a user were required to set each control word bit by bit according to the field definitions just given, it is obvious how tedious microprogramming would be.

For this reason, a new language, AMIL (A Microprogramming Language), has been created to allow users to write microprograms in a FORTRAN-like register transfer

language as opposed to ones and zeroes. As a result, users will now be able to write AMIL programs and allow the translator to convert his program into MCU bit patterns. This translator has been developed and is currently operational on a time-sharing service available to NRL.

The remainder of this document will describe the syntax of AMIL and its semantics as prescribed by the translator.

AMIL

AMIL is syntactically described using Backus-Naur Form (BNF) with the following additions:

1. { } Left and right braces are used to encompass an English language definition of a syntactic category.
2. *{ }* Left and right asterisk-braces are used to encompass a string of syntactic categories which may appear in any order. For example,

$$\langle \text{cat} \rangle ::= * \{ \langle \text{cat1} \rangle \langle \text{cat2} \rangle \} *$$

implies $\langle \text{cat} \rangle ::= \langle \text{cat1} \rangle \langle \text{cat2} \rangle \mid \langle \text{cat2} \rangle \langle \text{cat1} \rangle$

AMIL is designed to give the user many of the benefits one receives from high level languages, such as free field coding and symbolic next address selection, but with the desired constraint imposed that MCU registers be named explicitly to make sure the programmer is intimately involved with the facilities.

An AMIL program is composed of any number of instructions up to 4096 followed by an END psuedostatement. Each instruction consists of combinations of up to 7 different general types. These types are memory operations, adder operations, address adjustment, auxiliary transfers, condition phrases, successor specification, and I/O instructions.

After giving some general definitions, each of the types will be defined. The complete AMIL syntax is given in Appendix B and key words are listed in Appendix C. Appendix D includes two sample programs and their output from the AMIL translator, while Appendix E gives a complete list of error messages generated by the translator.

General Definitions

Syntax:

$$\langle \text{Program} \rangle ::= \langle \text{Body} \rangle \langle \text{End Line} \rangle$$

$$\langle \text{Body} \rangle ::= \langle \text{Instruction} \rangle \mid \langle \text{Body} \rangle \langle \text{Instruction} \rangle$$

$$\langle \text{Instruction} \rangle ::= \langle \text{Label Phrase} \rangle * \{ \langle \text{Memory Operation} \rangle \langle \text{Adder Op} \rangle \langle \text{Address Adjust} \rangle \langle \text{Auxiliary Transfer} \rangle \langle \text{Condition Phrase} \rangle \langle \text{Successor} \rangle \langle \text{I/O Instruction} \rangle \} * \$$$

$$\langle \text{Label Phrase} \rangle ::= \cdot \langle \text{Label} \rangle \mid \langle \text{Empty} \rangle$$

$$\langle \text{Label} \rangle ::= \langle \text{Letter} \rangle \mid \langle \text{Label} \rangle \langle \text{Letter} \rangle \mid \langle \text{Label} \rangle \langle \text{Symbol} \rangle \mid \langle \text{Label} \rangle \langle \text{Digit} \rangle$$

$$\langle \text{Letter} \rangle ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z$$

$$\langle \text{Digit} \rangle ::= \emptyset|1|2|3|4|5|6|7|8|9$$

⟨Symbol⟩ :: = +|-|/|()|=|,| .

⟨Empty⟩ :: = {the absence of anything}

⟨End Line⟩ :: = END \$

Semantics:

A program can consist of up to 4096 instructions followed by an END pseudoinstruction. Each instruction is entered via TTY one line at a time with an "*" ending each line which is not the last of an instruction. A "\$" is used to signal end of instruction and must always be present. Any characters after an "*" or "\$" are treated as comments and thus cause no action except for their listing. Acceptable letters and symbols are limited because the translator accepts only CDC standard FORTRAN characters. The string "END" is a key word, and like all key words, it must not contain any embedded blanks. ⟨Empty⟩ is the null category and is always considered to be present.

To enter a comment statement which occupies an entire line, the line must start with an asterisk.

Instructions

Syntax:

⟨Instruction⟩ :: = ⟨Label Phrase⟩ *{⟨Memory Operation⟩ ⟨Adder Op⟩ ⟨Address Adjust⟩
 ⟨Auxiliary Transfer⟩ ⟨Condition Phrase⟩ ⟨Successor⟩
 ⟨I/O Instruction⟩}* \$

⟨Label Phrase⟩ :: = . ⟨Label⟩ | ⟨Empty⟩

⟨Label⟩ :: = ⟨Letter⟩ | ⟨Label⟩ ⟨Letter⟩ | ⟨Label⟩ ⟨Symbol⟩ | ⟨Label⟩ ⟨Digit⟩

Semantics:

An instruction starts with an optional label phrase which is a "." followed by a label. A label is a string of up to 9 characters which must begin with a letter and contain no embedded blanks. Instructions are translated into consecutive control word locations starting with zero. Thus, the address where each instruction will reside is known at translation time. When a label is preceded by "." this causes the translator to save the current control store address and the label in a table. Any subsequent reference to this label causes the associated address to be substituted. This allows programs to refer to addresses symbolically.

After the optional label phrase, one can include any number of the 7 categories (memory op, adder op, etc.) in any order. It is important to note that their order has no relation to when they are executed within an MCU cycle. Examples:

```
.LOOP   INPUT (BUF(BARA), LSA (7)) $
.LOOP+1 OUTPUT (LSB (s), BUF(BARB)) $
.LOOP+2 LSB (9) = LSA (3) + 425 $
        INC BARA, LSA (4) = LSA (4) +1, JUMP TO LOOP $
        INPUT (BUF(BARB), LSB (1)), INC BARA, *
        LSA (6) = LSB (7) + FSU (3), SKIP $
```

Memory Operation

Syntax:

```

⟨Memory Operation⟩ :: = *{
  ⟨Buffer Read A to LSA⟩ ⟨Buffer Read A to LSB⟩}*|
  *{⟨Buffer Read A to LSA⟩ ⟨Buffer Read B to LSB⟩}*|
  *{⟨Buffer Read A to LSA⟩ ⟨Buffer Write B⟩}*|
  *{⟨Buffer Read A to LSB⟩ ⟨Buffer Read B to LSA⟩}*|
  *{⟨Buffer Read A to LSB⟩ ⟨Buffer Write B⟩}*|
  *{⟨Buffer Read B to LSA⟩ ⟨Buffer Read B to LSB⟩}*|
  *{⟨Buffer Read B to LSA⟩ ⟨FSU Register Set⟩}*|
  *{⟨Buffer Read B to LSA⟩ ⟨Buffer Write A⟩}*|
  *{⟨Buffer Read B to LSB⟩ ⟨FSU Register Set⟩}*|
  *{⟨Buffer Read B to LSB⟩ ⟨Buffer Write A⟩}*|
  *{⟨FSU Register Set⟩ ⟨Buffer Write B⟩}*|
  OUTPUT (LSA ⟨LS Subscript⟩, BUF(BARA)) OUTPUT (LSB ⟨LS
    Subscript⟩, BUF(BARB))|
  OUTPUT (LSB ⟨LS Subscript⟩, BUF(BARB)) OUTPUT (LSA ⟨LS
    Subscript⟩, BUF(BARA))|
  ⟨Empty⟩

```

```

⟨Buffer Read A to LSA⟩ :: = INPUT (BUF(BARA), LSA ⟨LS Subscript⟩) | ⟨Empty⟩

```

```

⟨Buffer Read A to LSB⟩ :: = INPUT (BUF(BARA), LSB ⟨LS Subscript⟩) | ⟨Empty⟩

```

```

⟨Buffer Read B to LSA⟩ :: = INPUT (BUF(BARB), LSA ⟨LS Subscript⟩) | ⟨Empty⟩

```

```

⟨Buffer Read B to LSB⟩ :: = INPUT (BUF(BARB), LSB ⟨LS Subscript⟩) | ⟨Empty⟩

```

```

⟨Buffer Write A⟩ :: = OUTPUT (⟨Buffer Write Source⟩, BUF(BARA)) | ⟨Empty⟩

```

```

⟨Buffer Write B⟩ :: = OUTPUT (⟨Buffer Write Source⟩, BUF(BARB)) | ⟨Empty⟩

```

```

⟨Buffer Write Source⟩ :: = Z | LSA ⟨LS Subscript⟩ | LSB ⟨LS Subscript⟩

```

```

⟨FSU Register Set⟩ :: = INPUT (BUF(BARA), FSDR) | ⟨Empty⟩

```

```

⟨LS Subscript⟩ :: = (⟨Register Number⟩)

```

```

⟨Register Number⟩ :: = ⟨FSU field⟩|8|9|10|11|12|13|14|15|BARA|BARB

```

```

⟨FSU field⟩ :: = ∅|1|2|3|4|5|6|7

```

Semantics:

During each MCU cycle up to two memory transfers can occur with the following limitations. There may not be conflicting uses of sources, destinations or address register selection. Each memory operation starts with the key word dictating the direction of the transfer (i.e., INPUT or OUTPUT) followed by an opening parenthesis, source, destination, and the closing parenthesis. The operation reads like an English language description of the transfer if one substitutes the word "from" for the opening parenthesis and "to" for the comma separating source and destination.

INPUT (BUF(BARA), LSA (7)) becomes INPUT from BUF(BARA) to LSA (7). "BUF(BARA)" implies the buffer memory and its address which is pointed to by BARA. "LSA (7)" means Local Store A register seven. The source for an INPUT operation is BUF(BARA) or BUF(BARB) and the destination LSA, LSB, or FSDR.

FSDR as mentioned earlier is the Field Select unit Data Register. The only exception about duplicate address register selection occurs here. One may INPUT from BUF(BARA) to FSDR and one other destination.

Sources for OUTPUT operations can come from Z, LSA, or LSB contents at the beginning of the current cycle. Destinations can be only buffer memories as specified by BUF(BARA) and BUF(BARB). Note the form of the special case where two OUTPUTS can occur. Memory operations can occur by themselves as instructions or anywhere inside an instruction. Local Stores are addressed in two ways, directly and indirectly. Directly means the register number is given. Indirectly means the register number is specified by the low-order four bits of an address register. For Local Store A, BARA is used and for Local Store B, BARB.

Examples:

(Legal -) INPUT (BUF(BARB), LSA (10)), OUTPUT (LSB (5),
 BUF(BARA)) \$ legal
 INPUT (BUF(BARA), LSB (15)), INPUT (BUF(BARB),
 LSA (5)) \$ legal

(Illegal -) INPUT (BUF(BARA), LSA (5)), OUTPUT (LSB (7),
 BUF(BARA)) \$ illegal (duplicate use of BARA)
 INPUT (BUF(BARA), LSA (6)), INPUT (BUF(BARB),
 LSA (3)) \$ illegal (duplicate use of LSA)

Adder Op

Syntax:

⟨Adder Op⟩ ::= ⟨Destination⟩ = ⟨Left Input⟩ | ⟨Destination⟩ = ⟨Right Input⟩ |
 ⟨Destination⟩ = ⟨Left Input⟩ ⟨Unary Operator⟩ | ⟨Destination⟩ = Z |
 ⟨Destination⟩ = ⟨Right Input⟩ ⟨Binary Operator⟩ ⟨Left Input⟩ |
 ⟨Destination⟩ = ⟨Left Input⟩ ⟨Binary Operator⟩ ⟨Right Input⟩ | ⟨Empty⟩

⟨Left Input⟩ ::= BARA | BARB | SAR | CTR | ACSAR | LSA ⟨LS Subscript⟩ |
 LSB ⟨LS Subscript⟩ | FSU (⟨FSU Field⟩)

⟨Integer⟩ ::= ⟨Digit⟩ | ⟨Integer⟩ ⟨Digit⟩

⟨Right Input⟩ ::= BARA | BARB | ⟨Integer⟩ | LSB ⟨LS Subscript⟩ | COMP1
 ⟨Integer⟩ | FSU (⟨FSU field⟩) | COMP2 ⟨Integer⟩

⟨Unary Operator⟩ ::= +1 | -1 | COMP | LEFT | RIGHT | CIRC

⟨Binary Operator⟩ ::= + | - | OR | AND | XOR | EQV

$\langle \text{Destination} \rangle :: = \langle \text{Regular Destination} \rangle |$
 $\langle \text{Destination} \rangle, \langle \text{Regular Destination} \rangle |$
 $\langle \text{Destination} \rangle, \langle \text{Auxiliary Destination} \rangle$

$\langle \text{Regular Destination} \rangle :: = Z | \text{BARA} | \text{BARB} | \text{LSA} \langle \text{LS Subscript} \rangle | \text{LSB} \langle \text{LS Subscript} \rangle$

Semantics:

Adder operations involve one or two operands, an operation and finally one or more destinations for the result. The symbol "=" means "is replaced by" as in FORTRAN. Unary operations occur only on left inputs, but all registers can be a left input. The unary operation "COMP" means take the one's complement of the left input. "LEFT" and "RIGHT" refer to shifting (end-off zero fill) the left input by the number of bits specified by the Shift Amount Register (SAR) in the direction given by the mnemonic. "CIRC" requests a left-circular shift of SAR bits. To shift right-circular SAR bits, simply complement the SAR value and shift left circular.

Binary operators include two's complement addition and subtraction along with the four basic Boolean operations.

Left and right inputs may appear in any order except for subtraction where order is important. When an $\langle \text{Integer} \rangle$ is specified in one of the three ways (i.e. $\langle \text{Integer} \rangle$, COMP1 $\langle \text{Integer} \rangle$, or COMP2 $\langle \text{Integer} \rangle$) it is imperative to remember that integers will be stored in a single literal field of the control word. It is, therefore, impossible to specify more than one integer within an instruction. COMP1 and COMP2 cause the integer which follows to be complemented, as above, before being inserted into the LIT field of the control word.

Although the syntax appears to allow any combination of Left and Right adder inputs, this is not the case and only the 16 combinations given below are legal. Any other combination will result in a diagnostic.

<u>Left</u>	<u>Right</u>
LSA(i)	Integer FSU(k) BARB LSB(j)
LSB(j)	Integer FSU(k) BARB BARA
BARA	Integer LSA(i)
BARB	Integer BARA
FSU(k)	Integer
ACSAR	Integer
SAR	Integer
CTR	Integer

This configuration has all inputs on the left at least once, thus allowing each input to be

1. Shifted
2. Transferred to any legal output destination
3. Complemented.

The adder output can go to one or more destinations. AMIL requires that the first destination be either Z, BARA, BARB, LSA, or LSB. Once one of these is specified, any or all of the remaining destinations may be specified including a choice of one from either of ACSAR, SAR, or CTR.

If the operation chosen is simply a transfer, then plus zero is the implied operation. This need not be explicitly written since this is the default operation.

Examples:

Legal LSB (3) = LSA (4) + LSB (5) \$
 LSA (1), SAR = BARA + 10 \$ (note - two destinations)
 ACSAR = LSB (12) \$ (+ 0 need not be included)
 LSA (1) = LSA (1) or LSB (7)
 Z = LSA (9) + 65535 \$
 LSB (5) = LSB (5) COMP \$ (LSB (5) is one's complemented)
 BARB = LSB (10) + COMP1 36 \$ (36 is added to LSB (10))
 LSA (3) = LSA (3) LEFT \$ (LSA (3) is Left Shifted SAR bits)
 LSA (1), LSB (2), BARA, BARB, ACSAR = LSA (5) EQV LSB (2) *
 \$ (many Destinations)
 BARA = BARA + 100 \$

Illegal SAR, LSA (1) = BARA + 10 \$ (Destinations in improper order)
 LSA (7) = CTR + 25 \$ (Illegal Left-Right combination)

Address Adjust

Syntax:

Address Adjust ::= <INC> | <DEC> | <INC>, <DEC> | <INC>, <INC> |
 <DEC>, <INC> | <DEC>, <DEC> | <Empty>

<INC> ::= INC BARA | INC BARB

<DEC> ::= DEC BARA | DEC BARB

Semantics:

During the latter part of an MCU cycle the address registers BARA and BARB can be adjusted by either an increment (INC) or decrement (DEC) on either or both registers. The timing is mentioned to show there is no conflict with memory operations whose use of the address registers causes their contents to be captured early in the cycle, much before an adjustment.

It is not possible, however, to adjust BARA or BARB and also store the results of an adder operation in the same register in the same cycle.

Examples:

Legal INPUT (BUF(BARA), LSA (2)), INC BARA, DEC BARB \$
 DEC BARA, DEC BARB, OUTPUT (LSB(3), BUF(BARB)) \$
 INC BARA, LSA (6) = LSA(5) + BARB \$
 INC BARA, INC BARB \$

Illegal INC BARA, BARA = LSA(5) + LSB(6) \$
 INC BARA, BARB \$

Auxiliary Transfer

Syntax:

⟨Auxiliary Transfer⟩ ::= ⟨Auxiliary Destination⟩ = ⟨Auxiliary Source⟩ | ⟨Empty⟩
 ⟨Auxiliary Source⟩ ::= ⟨Literal⟩ | Z | COMP1 ⟨Integer⟩ | COMP2 ⟨Integer⟩
 ⟨Literal⟩ ::= ⟨Integer⟩ | ⟨Label⟩
 ⟨Auxiliary Destination⟩ ::= ACSAR | SAR | CTR

Semantics:

It is possible in the MCU to make an additional data transfer concurrently with an adder operation. This is normally used to put constants (counts, addresses, or shift amounts) into one of three registers but could be used to store the last adder operation's result, Z, in the selected register. If an adder operation occurs in an instruction with Z also being selected as the source for an auxiliary transfer, then Z is the result of this adder operation. The following instructions illustrate this. They would cause identical responses in the MCU.

1. LSA(7) = LSA(5) AND LSB(12), CTR = Z \$
2. LSA(7), CTR = LSA(5) AND LSB(12) \$

COMP1 and COMP2 have the same meaning as before. One should recall that labels are always resolved into integer addresses and stored in the literal field of the control word with only one allowed per instruction. If one had an unused literal field in an instruction and he wanted to set up a jump address in ACSAR, he could use the auxiliary transfer using the label name. When he wanted to jump the address of the label he would simply "Jump To ACSAR" which requires no literal field. (See "Successor" discussion.)

Condition Phrase

Syntax:

⟨Condition Phrase⟩ ::= IF ⟨Not⟩ ⟨Condition⟩ | ⟨Empty⟩
 ⟨Not⟩ ::= NOT | ⟨Empty⟩
 ⟨Condition⟩ ::= LEAST | MOST | ADROV | CTROV | ZERO

Semantics:

The only part of an MCU instruction which is actually conditional is the next address selection. All operations go on as normal except that the "successor" specified is selected only if the "condition and not" combination is satisfied; otherwise a default step occurs. The conditions have the following meanings:

IF LEAST- If the least significant bit of Z from the last adder operation (a previous cycle) is set (i.e. one)

IF NOT LEAST- If least significant bit of Z is zero

MOST- Most significant bit of Z from last adder Op

ADROV- Adder overflow on the last adder Op

CTROV- Counter (CTR) overflow after the last counter increment which occurs automatically and only by checking for CTROV

ZERO- Result of the last adder operation was zero.

From the above discussion, it would seem logical and orderly to include the next address selection <Successor> immediately following <Condition Phrase>, but it is not required.

Examples:

<u>Address</u>	<u>Instruction</u>	
10	LSA(7)=LSA(7) + 1	\$Assume LSA(7)=0 initially
11	IF LEAST JUMP TO 13 LSA(7)=LSA(7)+1	\$Least is set from 10 and \$LSA (7) now is 2
12	LSA(7)=LSA(7)+2	\$Not executed if LSA(7) is even
13	LSB(5)=LSA(7)	\$LSB(5)=2 in our case

Successor*Syntax:*

<Successor> :: = STEP | SKIP | SAVE | < Call > | INTRET | <Jump>

<Jump> :: = JUMP TO <Jump Destination>

<Call> :: = CALL <Literal>

<Jump Destination> :: = <Literal> | ACSAR | Z

Semantics:

A successor field may always appear whether a condition is present or not. If no condition is present the next address selection is unconditional and always occurs. One can choose the next address from eight possibilities.

Two registers are involved in addressing the control store. The CSAR is the work-horse which is incremented under normal conditions to create the default program sequencing. The ACSAR is used to hold return address or jump addresses. The following chart shows the effect of each instruction on the two registers.

<u>Successor</u>	<u>Next Instruction</u>	<u>Next CSAR</u>	<u>Next ACSAR</u>
STEP	CSAR+1	CSAR+1	-----
SKIP	CSAR+2	CSAR+2	-----
SAVE	CSAR+1	CSAR+1	CSAR+1
CALL	CS Literal	CS Literal	CSAR+1
JUMP TO ACSAR	ACSAR	ACSAR	-----
JUMP TO Literal	CS Literal	CS Literal	-----
JUMP TO Z	Z	Z	-----
INTRET	CSAR from interrupt stack	CSAR from stack	-----

Upon encountering an interrupt of high enough priority, the MCU automatically pushes the current CSAR into a stack and generates an address for the appropriate interrupt handling routine. When a return to preinterrupt status is desired, the INTRET successor is specified to pop the stack and restore the CSAR for normal sequencing.

Examples:

IF MOST JUMP TO ERROR \$ Jump to address error if Z most is set

LSB(7) = LSA(6) + LSB(5), CALL MULT \$ call subroutine mult

I/O Instruction

Syntax:

$\langle \text{I/O Instruction} \rangle :: = \text{I/O} (\langle \text{LS Subscript} \rangle, \langle \text{Literal} \rangle) |$
 $\langle \text{Empty} \rangle$

Semantics:

I/O, in the sense of the syntax above, is dealing with devices other than buffers. At the current time (7/1/72), I/O will be handled in the following fashion. A programmer will assemble an I/O control word in LSA which will specify device, address (I.D.) op code, etc. If more information (or data) is required, it should be in Z. The programmer then simply includes the I/O command with the correct Local Store A subscript in his instruction (without a condition phrase) and the contents of LSA(i) will be placed on the Z bus. At this point the MCU waits for an acknowledge which means the command has been accepted, completed if just a status check, or rejected. In the latter case, the acknowledge clocks the status into Z and the MCU will automatically jump to the address in the literal field. In short, the command from Local Store comes from the register specified by $\langle \text{LS Subscript} \rangle$ and the Reject address comes from $\langle \text{Literal} \rangle$. For a further explanation, see Appendix A.

Example:

I/O (12, Bombout) \$ Send out an I/O command with control from
LSA(12) - if rejected jump to address "Bombout"

I/O (3, 1273) \$ Control = LSA (3), Reject = address 1273

GLOSSARY OF ACRONYMS

<u>Acronym</u>	<u>Meaning</u>
AADC	All Applications Digital Computer
AMIL	A Microprogramming Language
BORAM	Block Oriented Random Access Memory
CC	Channel Controller
DC	Device Controllers
DMA	Direct Memory Access
MCU	Microprogrammed Control Unit
SCC	Selector Channel Controllers
SCU	Storage Control Unit
SPAU	Signal Processing Arithmetic Unit
SPE	Signal Processing Element
UD	Unbuffered Device

Appendix A

INPUT/OUTPUT CONTROL

ORGANIZATION

There are two types of input/output (I/O) channels in the SPE: Direct Memory Access (DMA) buffered channels and a single unbuffered channel called the Z bus. Eight/sixteen buffered channels enable high-speed data transfer between buffer memories and system devices or MCU's. The Z bus allows direct communication under MCU control and on a word-by-word basis between the Z register of an MCU and all devices connected to the Z bus. The Z bus also enables direct MCU-to-MCU communication.

Figure A1 shows an SPE configuration with I/O system elements and interconnections.

All buffered channel communications pass through the Storage Control Unit (SCU). It is the responsibility of the SCU to manage buffer memory requests originating from MCU's, SPAU's, and system channel controllers. Devices which access buffer memory over buffered channels are interfaced to the buffered channels by Selector Channel Controllers (SCC). SCC's also interface with the Z bus and are responsible for interpreting device requests coming over the Z bus from MCU's. These requests originate in the form of MCU I/O instructions and can call upon an SCC to initiate various device I/O operations over its buffered channel interface.

The SCC's are intended to be standard I/O elements interfacing between buffered channels and Device Controllers (DC). DC's interface between SCC's and I/O devices and must be tailored to meet the interface requirements of a particular device type. DC's interface to SCC's over Z-bus-compatible connections. This allows a DC to connect directly to the Z bus for direct unbuffered communication with an MCU or to connect to an SCC for buffered channel communication.

SCC's and DC's can request MCU action via interrupt lines provided in the MCU's for such purposes. Separate SCC's or DC's sharing a single interrupt line must have hardware to resolve competition among the units for interrupt service.

An MCU generating an I/O request addressed to another MCU for the purpose of MCU-to-MCU communication causes the addressed MCU to raise an internal interrupt line. An I/O acknowledge instruction by the interrupted MCU completes the data transfer over the Z bus.

UNBUFFERED CHANNEL

The MCU exchanges commands and unbuffered data with devices over a bidirectional, byte-multiplexed channel called the Z bus. The channel can be interrupt driven with information transferred from/to the Z register upon execution of an I/O instruction. There is but one Z bus regardless of the number of MCU's a system may contain. MCU's are in charge of Z bus usage and resolve Z-bus access among themselves on a first-come, first-served basis.

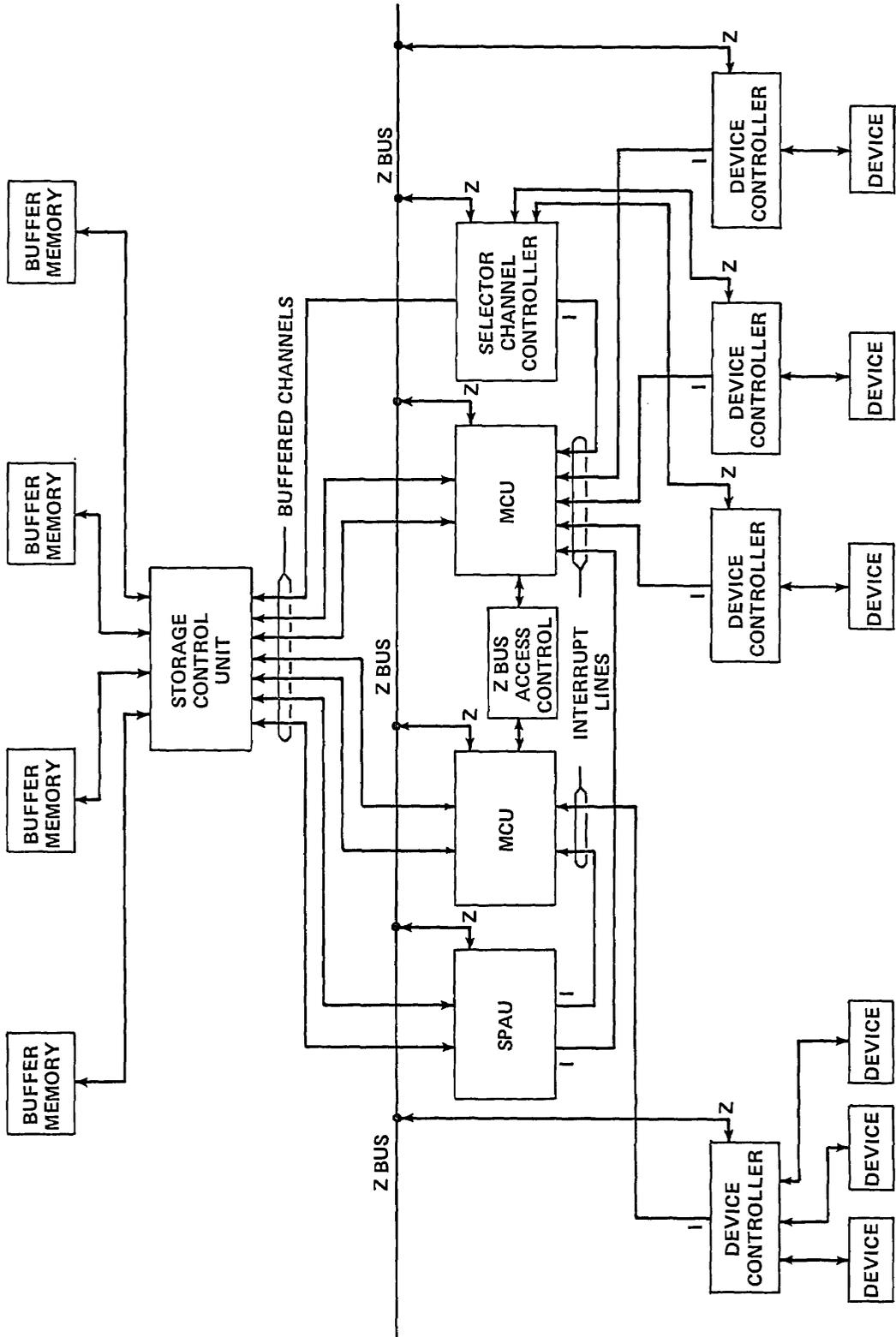


Fig. A1 - SPE configuration illustrating the I/O system

Devices wishing Z-bus access must alert the MCU via an interrupt. A single interrupt line may serve many devices via a Device Controller. The controller must resolve simultaneous requests for service. The interrupt line is deactivated upon transmission of an I/O command by the servicing MCU to the interrupting device's controller. Different interrupt lines servicing different sets of devices are subject to the priority resolution scheme built into the MCU interrupt mechanism, priority being determined by the position of the interrupt line on the MCU. The maximum burst transfer rate over the Z bus, based upon an MCU cycle time of 150 nsec, is 2 MHz.

Z BUS

The Z bus consists of 30 lines, 16 of which are bidirectional data lines. The remaining 14 are control lines, as follows:

- a. Eight Device Address Lines (bidirectional)
- b. Input/Output Line (output)
- c. Data/Control Line (output)
- d. Continue Line (output)
- e. Command Line (bidirectional)
- f. Acknowledge Line (bidirectional)
- g. Reject Line (input).

The Device Address Lines alert a specified device controller or channel controller to connect to the Z-bus data lines for an information transfer.

The Input/Output Line specifies the direction of information transfer.

The Data/Control Line specifies whether the transfer will involve data or control information. If control information is specified, an Input command will bring a status word from the device to the MCU; an Output command will send a command word from the MCU to the device.

The Continue Line is used to specify, where appropriate, that a device be prepared to accept another I/O word as continuation of the present I/O.

The Command Line alerts all devices on the Z bus that an I/O command has been put on the bus.

The Acknowledge Line is raised by a device to acknowledge to the MCU that the device has accepted the I/O command.

The Reject Line is raised by a device to notify the MCU that the device cannot accept the I/O command.

The Device Address Lines, Command Line, and Acknowledge Line take part in MCU-to-MCU communication as well as MCU-to-device communication and are therefore bidirectional. An MCU continually monitors the Command Line and Device Address Lines for an I/O command directed at it by another MCU. An MCU can raise the Acknowledge Line in response to an I/O command in the same manner as a device.

MCU-Z-BUS INTERFACE

The MCU interfaces with the Z bus through its Z register and Local Store A (LSA). The Z bus data lines input to and are driven from the Z register. The Z register must be loaded with data or command information prior to an I/O output command. An I/O input operation loads the Z register from the Z bus for subsequent use.

A control word must be set up in LSA prior to the I/O command. The LSA location containing the control word is specified by the LSA Address Field, and 12 bits of the selected word directly drive 12 control lines of the Z bus as shown in Fig. A2. The Acknowledge Line is driven indirectly through I/O control logic. The Reject Line inputs to the MCU Sequence Unit and causes a program jump to the Control Store location specified by the Lit field contents in the I/O command word.

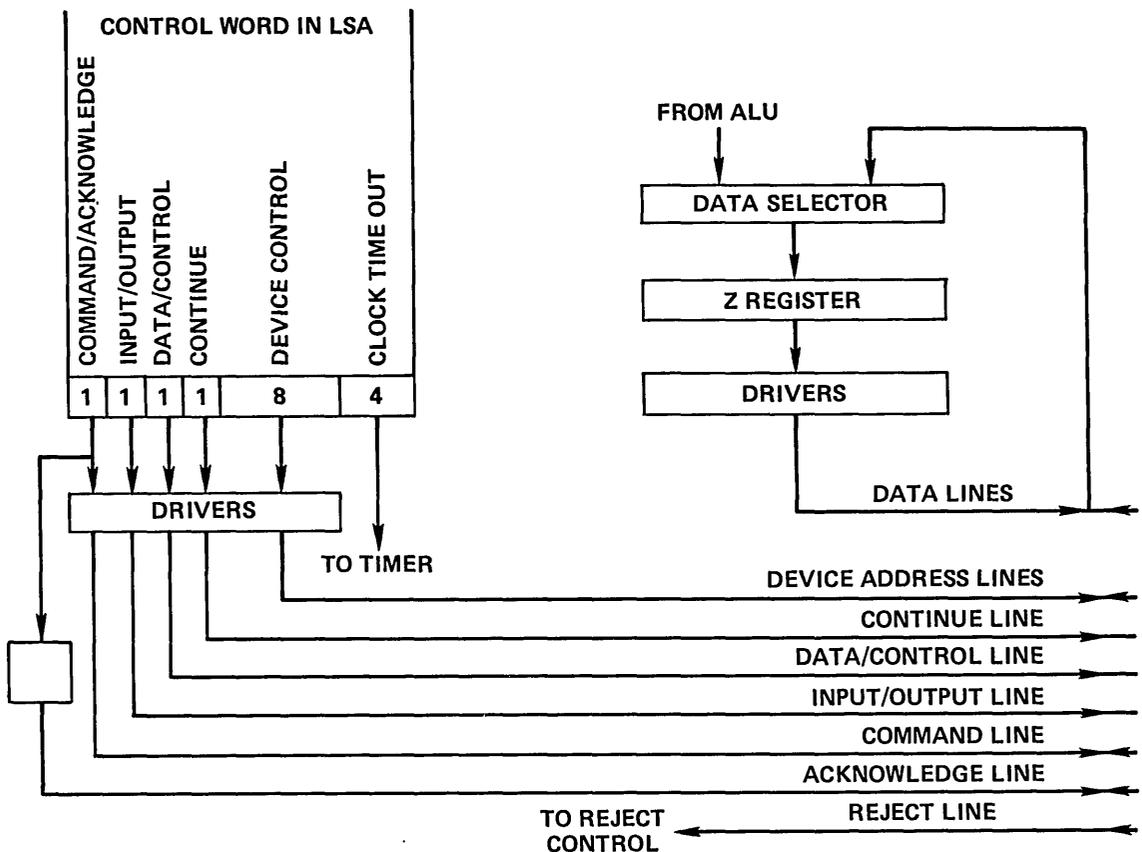


Fig. A2 - MCU-to-Z-bus interface

Four bits of the command word set up in LSA do not go out over the Z bus but go to a clock time-out counter in the MCU. These bits allow the program to specify a time interval in powers of 2 from 2^0 to 2^{15} clock cycles. This interval limits the time that the MCU will wait inhibited in the I/O state for the return of an Acknowledge signal from an addressed device. In the event of a time out, an internal interrupt is generated and the MCU leaves the I/O state to process that interrupt.

UNBUFFERED I/O OPERATION

I/O operations are initiated by an I/O instruction in an MCU. Devices cannot initiate I/O operations directly but do so by appealing to an MCU through the interrupt system. An I/O command is specified by a code point of 7 in the command word successor field. Prior to the I/O instruction, the I/O control word must be set up in LSA and, if the transfer is to be an output, data or command information must be set up in the Z register. Also, the programmer must specify in the LSA address field the location of the I/O control word in LSA, and the address of the Reject jump must be specified in the Lit field.

The MCU Command Register is loaded every cycle at P/4. If an I/O operation is specified, the MCU raises an I/O Request Line to the Z-Bus Control Unit which resolves requests for the Z bus from different MCU's. Access to the bus is granted strictly on a first-come, first-served basis. If the Z bus is available, the MCU will receive a Z-Bus Available Line which, combined with the I/O Request Line, initiates the I/O cycle at P/1. Absence of the Z-Bus Available signal inhibits MCU operation until the bus becomes available at a later cycle.

The MCU I/O cycle proceeds as follows. P/2 sets the I/O Command FF or P/3 sets the I/O Acknowledge FF, depending on the state of the Command/Acknowledge bit in the I/O control word read from LSA. An I/O command causes the control word drivers from LSA to be energized, and, if the command is an output, the data drivers from the Z register are also energized. Further clock pulses for MCU operation are suspended until the receipt of the Acknowledge Line signal in conjunction with P/3 clears the I/O Command FF thereby de-energizing all Z-bus drivers. Simultaneously, if the I/O command is an input, the information on the Z-bus data lines is clocked into the Z register. The MCU then proceeds with normal operation.

If the I/O operation is an Acknowledge, the I/O Acknowledge FF is set at P/3. This energizes a driver on the Acknowledge Line only. Simultaneously, the Z-bus data lines are clocked into the Z register and the MCU continues with normal operation.

If the Reject Line is raised in response to an I/O Command, the MCU remains inhibited until the conjunction of P/1 and the Reject Line signal which clears the I/O Command FF and clocks the contents of the CS Lit field into the CSAR. This causes the MCU to resume normal processing at the I/O Reject jump address.

BUFFERED CHANNEL CONTROLLER

The Buffered Channel Controller can take several forms. In the SPE system a Selector Channel Controller is specified as a generalized Direct Memory Access Controller.

The SCC interfaces with the Storage Control Unit (SCU), the Z bus, and up to 8 Device Controllers (DC). Each DC may be attached to up to 4 SCC's.

The SCC interfaces with the Z bus to receive commands from and to transmit status information to an MCU. The SCC, in addition to the Z-bus lines, has an interrupt line to alert an MCU of a change in device status.

The SCC-DC interface is identical to the Z-bus interface except that the data width is 32 lines. Eight, 16, or 32 data-line DC's may be connected to the SCC. The particular width for an SCC-DC operation is determined by a 2-bit field (WDC) in the MCU I/O Command to the SCC.

The SCC contains a 32-bit assembly register at the SCC-DC interface and a 32-bit data exchange register at the SCU-SCC interface. Two incrementing address registers and a word counter are contained in the SCC.

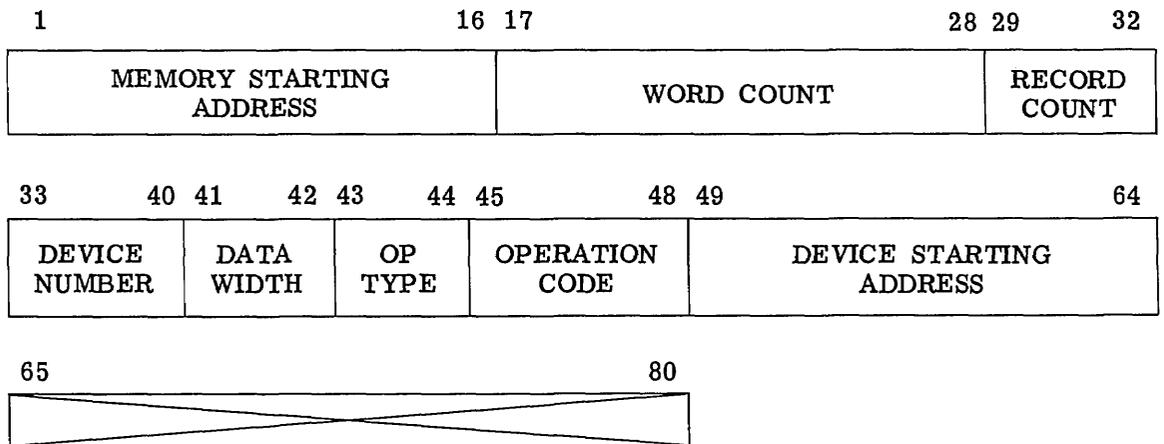
If an SCC is connected to the highest priority SCU position, a transfer rate to buffer memory of 192 mbits/sec is possible. The maximum transfer rate is dependent on the round-trip line delay between the SCC and DC.

SCC Command and Format

The SCC Command Format includes the following fields:

1. Buffer Memory Starting Address (16 bits)
2. Word Count (12 bits)
3. Record Count (4 bits)
4. Device Number (8 bits)
5. SCC-DC Data Width (2 bits)
6. Operation Type (2 bits)
7. Operation Code (4 bits)
8. DEVICE STARTING ADDRESS.

SCC COMMAND FORMAT

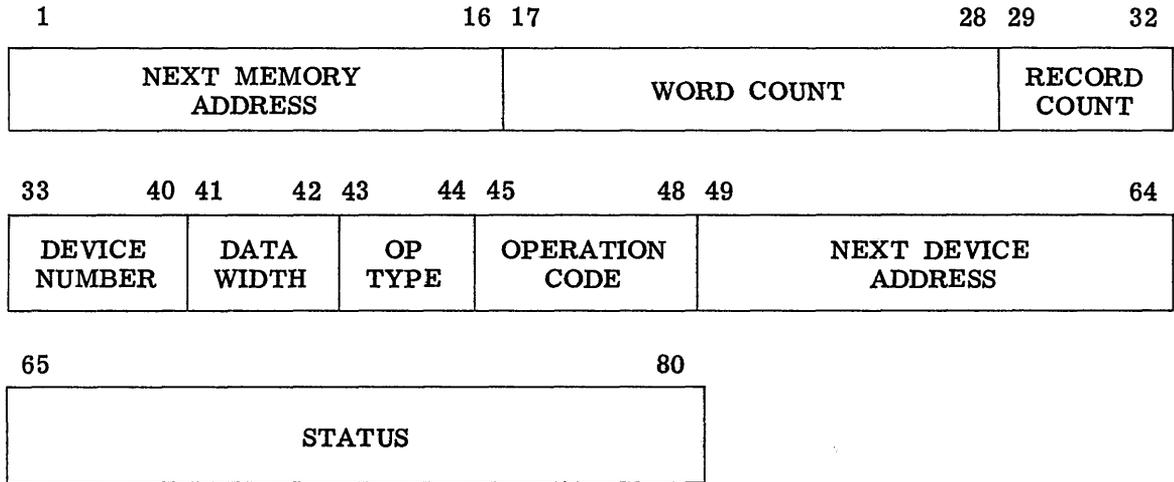


SCC Status Information and Format

1. Next Memory Address (16 bits)
2. Word Count (12 bits)
3. Record Count (4 bits)
4. Device Number (8 bits)
5. SCC-DC Data Width (2 bits)
6. Operation Type (2 bits)

- 7. Operation Code (4 bits)
- 8. Next Device Address (16 bits)
- 9. Status (16 bits)

SCC STATUS FORMAT



Appendix B

A MICROPROGRAMMING LANGUAGE (AMIL) SYNTAX

$\langle \text{Program} \rangle ::= \langle \text{Body} \rangle \langle \text{End Line} \rangle$

$\langle \text{Body} \rangle ::= \langle \text{Instruction} \rangle | \langle \text{Body} \rangle \langle \text{Instruction} \rangle$

$\langle \text{Instruction} \rangle ::= \langle \text{Label Phrase} \rangle * \{ \langle \text{Memory Operation} \rangle \langle \text{Adder Op} \rangle \langle \text{Address Adjust} \rangle$
 $\langle \text{Auxiliary Transfer} \rangle \langle \text{Condition Phrase} \rangle \langle \text{Successor} \rangle$
 $\langle \text{I/O Instruction} \rangle \} * \$$

$\langle \text{Label Phrase} \rangle ::= \cdot \langle \text{Label} \rangle | \langle \text{Empty} \rangle$

$\langle \text{Label} \rangle ::= \langle \text{Letter} \rangle | \langle \text{Label} \rangle \langle \text{Letter} \rangle | \langle \text{Label} \rangle \langle \text{Symbol} \rangle | \langle \text{Label} \rangle \langle \text{Digit} \rangle$

$\langle \text{Letter} \rangle ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z$

$\langle \text{Digit} \rangle ::= \emptyset | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$

$\langle \text{Symbol} \rangle ::= + | - | / | (|) | = | , | .$

$\langle \text{Empty} \rangle ::= \{ \text{the absence of anything} \}$

$* \{ \text{categories} \} * ::= \{ \text{the categories inside of } * \{ \text{ and } \} * \text{ can appear in any order} \}$

$\langle \text{Memory Operation} \rangle ::= * \{ \langle \text{Buffer Read A to LSA} \rangle \langle \text{Buffer Read A to LSB} \rangle \} * |$
 $* \{ \langle \text{Buffer Read A to LSA} \rangle \langle \text{Buffer Read B to LSB} \rangle \} * |$
 $* \{ \langle \text{Buffer Read A to LSA} \rangle \langle \text{Buffer Write B} \rangle \} * |$
 $* \{ \langle \text{Buffer Read A to LSB} \rangle \langle \text{Buffer Read B to LSA} \rangle \} * |$
 $* \{ \langle \text{Buffer Read A to LSB} \rangle \langle \text{Buffer Write B} \rangle \} * |$
 $* \{ \langle \text{Buffer Read B to LSA} \rangle \langle \text{Buffer Read B to LSB} \rangle \} * |$
 $* \{ \langle \text{Buffer Read B to LSA} \rangle \langle \text{FSU Register Set} \rangle \} * |$
 $* \{ \langle \text{Buffer Read B to LSA} \rangle \langle \text{Buffer Write A} \rangle \} * |$
 $* \{ \langle \text{Buffer Read B to LSB} \rangle \langle \text{FSU Register Set} \rangle \} * |$
 $* \{ \langle \text{Buffer Read B to LSB} \rangle \langle \text{Buffer Write A} \rangle \} * |$
 $* \{ \langle \text{FSU Register Set} \rangle \langle \text{Buffer Write B} \rangle \} * |$
 $\text{OUTPUT (LSA} \langle \text{LS Subscript} \rangle, \text{BUF(BARA)) OUTPUT (LSB} \langle \text{LS}$
 $\text{Subscript} \rangle, \text{BUF(BARB)) |}$
 $\text{OUTPUT (LSB} \langle \text{LS Subscript} \rangle, \text{BUF(BARB)) OUTPUT (LSA} \langle \text{LS}$
 $\text{Subscript} \rangle, \text{BUF(BARA)) |}$
 $\langle \text{Empty} \rangle$

<Buffer Read A to LSA> :: = INPUT (BUF(BARA), LSA <LS Subscript>) | <Empty>
 <Buffer Read A to LSB> :: = INPUT (BUF(BARA), LSB <LS Subscript>) | <Empty>
 <Buffer Read B to LSA> :: = INPUT (BUF(BARB), LSA <LS Subscript>) | <Empty>
 <Buffer Read B to LSB> :: = INPUT (BUF(BARB), LSB <LS Subscript>) | <Empty>
 <Buffer Write A> :: = OUTPUT (<Buffer Write Source>, BUF(BARA)) | <Empty>
 <Buffer Write B> :: = OUTPUT (<Buffer Write Source>, BUF(BARB)) | <Empty>
 <Buffer Write Source> :: = Z | LSA <LS Subscript> | LSB <LS Subscript>
 <FSU Register Set> :: = INPUT (BUF(BARA), FSDR) | <Empty>
 <LS Subscript> :: = (<Register Number>)
 <Register Number> :: = <FSU field> | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | BARA | BARB
 <FSU field> :: = \emptyset | 1 | 2 | 3 | 4 | 5 | 6 | 7
 <Adder Op> :: = <Destination> = <Left Input> | <Destination> = <Right Input> |
 <Destination> = <Left Input> <Unary Operator> | <Destination> = Z |
 <Destination> = <Right Input> <Binary Operator> <Left Input> |
 <Destination> = <Left Input> <Binary Operator> <Right Input> | <Empty>
 <Left Input> :: = BARA | BARB | SAR | CTR | ACSAR | LSA <LS Subscript> |
 LSB <LS Subscript> | FSU (<FSU Field>)
 <Integer> :: = <Digit> | <Integer> <Digit>
 <Right Input> :: = BARA | BARB | <Integer> | LSB <LS Subscript> | COMP1
 <Integer> | FSU (<FSU field>) | COMP2 <Integer>
 <Unary Operator> :: = +1 | -1 | COMP | LEFT | RIGHT | CIRC
 <Binary Operator> :: = + | - | OR | AND | XOR | EQV
 <Destination> :: = <Regular Destination> |
 <Destination> , <Regular Destination> |
 <Destination> , <Auxiliary Destination>
 <Regular Destination> :: = Z | BARA | BARB | LSA <LS Subscript> | LSB <LS Subscript>
 <Address Adjust> :: = <INC> | <DEC> | <INC>, <DEC> | <INC>, <INC> |
 <DEC>, <INC> | <DEC>, <DEC> | <Empty>
 <INC> :: = INC BARA | INC BARB
 <DEC> :: = DEC BARA | DEC BARB

$\langle \text{Auxiliary Transfer} \rangle ::= \langle \text{Auxiliary Destination} \rangle = \langle \text{Auxiliary Source} \rangle | \langle \text{Empty} \rangle$

$\langle \text{Auxiliary Source} \rangle ::= \langle \text{Literal} \rangle | Z | \text{COMP1} \langle \text{Integer} \rangle | \text{COMP2} \langle \text{Integer} \rangle$

$\langle \text{Literal} \rangle ::= \langle \text{Integer} \rangle | \langle \text{Label} \rangle$

$\langle \text{Auxiliary Destination} \rangle ::= \text{ACSAR} | \text{SAR} | \text{CTR}$

$\langle \text{Condition Phrase} \rangle ::= \text{IF} \langle \text{Not} \rangle \langle \text{Condition} \rangle | \langle \text{Empty} \rangle$

$\langle \text{Not} \rangle ::= \text{NOT} | \langle \text{Empty} \rangle$

$\langle \text{Condition} \rangle ::= \text{LEAST} | \text{MOST} | \text{ADROV} | \text{CTROV} | \text{ZERO}$

$\langle \text{Successor} \rangle ::= \text{STEP} | \text{SKIP} | \text{SAVE} | \langle \text{Call} \rangle | \text{INTRET} | \langle \text{Jump} \rangle$

$\langle \text{Jump} \rangle ::= \text{JUMP TO} \langle \text{Jump Destination} \rangle$

$\langle \text{Call} \rangle ::= \text{CALL} \langle \text{Literal} \rangle$

$\langle \text{Jump Destination} \rangle ::= \langle \text{Literal} \rangle | \text{ACSAR} | Z$

$\langle \text{End Line} \rangle ::= \text{END} \$$

$\$::= \{ \text{the end of instruction delimiter} \}$

$\langle \text{I/O Instruction} \rangle ::= \text{I/O} \langle \text{LS Subscript} \rangle, \langle \text{Literal} \rangle | \langle \text{Empty} \rangle$

Appendix C

AMIL KEY WORDS

The following is a list of key words (reserved words) which cannot be used except in the context defined by the syntax.

AMIL Key Words and Symbols

Key Word	Meaning
Registers:	
LSA (BARA or Integer)	- Local Store A
LSB (BARB or Integer)	- Local Store B
BARA	- Buffer Address Register A
BARB	- Buffer Address Register B
CTR	- Counter
SAR	- Shift Amount Register
Z	- Adder/Shifter output register
INPUT	- Buffer Memory Read
OUTPUT	- Buffer Memory Write
FSU (integer)	- Field Select Unit Field Specification
FSDR	- Field Select Unit Data Register
Conditions:	
MOST	- Most significant bit of the adder output (Z)
LEAST	- Least significant bit of Z
ADROV	- Adder Overflow
CTROV	- Counter Overflow
ZERO	- Adder output equal to ZERO
IF	- Signals conditional instruction
NOT	- Signals testing of NOT of the condition
Adder Operations:	
+	- Addition (Left + Right)
-	- Subtraction (Left - Right)
OR	- Logical OR

Key Word	Meaning
Adder Operations (Cont.):	
AND	- Logical AND
XOR	- Logical Exclusive Or
EQV	- Logical Equivalence
COMP	- Complement Left Input (one's complement)
-1	- Left Input minus one (Left -1)
+1	- Left Input plus one (Left +1)
LEFT	- LEFT shift the left input by amount in SAR
RIGHT	- RIGHT shift the left input by amount in SAR
CIRC	- CIRCular shift right the left input by SAR
INC	- Increment the register specified (BARA or BARB)
DEC	- Decrement the register specified (BARA or BARB)
I/O	- I/O instruction occurs this cycle
Successors:	
STEP	- Execute the instruction in the next (+1) control word (this is the default successor)
SKIP	- SKIP the next instruction and execute the one following that one
SAVE	- SAVE the current address +1 in ACSAR then STEP
CALL	- Jump to the address in LIT field and save return address (current address +1) in ACSAR
JUMP TO (integer or label)	- Jump to the address in the LIT field
JUMP TO ACSAR	- Jump to the address in ACSAR (returns)
JUMP TO Z	- Jump to the address in Z (calculated addresses)
INTRET	- Interrupt return
Key Symbols:	
=	- Assignment operator (can be read as 'goes to')
\$	- End of instruction delimiter (comments can follow)
,~	- Separator (between key words - not inside)
blank	- Separator (between key words - not inside)
()	- Used to delimit source or destinations in buffer memory operations; also used to delimit subscripts and field i.d.'s
*	- Instruction continuation delimiter

Key Word	Meaning
Key Symbols (Cont.):	
.	- Label delimiter
END	- End of program delimiter
COMP1	- Take the one's complement of the literal which follows
COMP2	- Take the two's complement of the literal which follows

Appendix D

AMIL SAMPLE PROGRAMS AND LISTINGS

This section contains a sample problem, the solution programmed in AMIL, and the listing of the AMIL translator's output for the program.

Also included is another translator run, but not of a program that performs any function. The program is simply an exercise of the translator and serves as a good representation of the many different forms of AMIL instructions.

SAMPLE PROGRAM IN AMIL

Problem: 1600 data points have been collected and put into Buffer Memory. There are 40 samples from each of 40 channels. The points are in time order as illustrated below:

Input

	Sample from
Memory location(i)	Channel(1) Time(1)
(i+1)	Channel(2) Time(1)
⋮	⋮
	Channel(40) Time(1)
	Channel(1) Time(2)
	⋮
(i+1599)	Channel(40) time(40)

The problem is to demultiplex these samples into channel order so signal processing (fft, filters, etc.) can proceed. The desired output is in the following form:

Output

Memory location(j)	Channel(1) Time(1)
(j+1)	Channel(1) Time(2)
	⋮
(j+39)	Channel(1) Time(40)
(j+40)	Channel(2) time(1)
	⋮
(j+1599)	Channel(40) time(40)

The input is in one Buffer Memory and the output will be into another Buffer Memory.

The program to solve this problem follows:

Line No.	Number of Executions	AMIL Instructions
1	1	BARA = 1000 \$ Location 500 of buffer memory 0
2	1	BARB = 10192 \$ Location 1000 of buffer memory 1
3	1	LSB(1),CTR = 40 COMP1, SAVE \$
4	1600	INPUT(BUF(BARA),LSA(1)), OUTPUT(LSA(1),BUF(BARB)), * INC BARB, BARA = BARA + 40, IF NOT CTROV * (Then) JUMP TO ACSAR \$
5	40	(Else) LSB(1) = LSB(1) + 1, CTR = COMP1 40, STEP \$
6	40	BARA = BARA - 1599, IF NOT ADROV JUMP TO ACSAR \$
7	1	(Else) END \$
Total =		1684

Please note that it took the MCU only 1684 clocks (1 clock per instruction execution) to do a problem involving 3200 memory transfers. The key to this almost real-time rate is in line number 4, which demonstrates the large amount of parallelism available in the MCU. In this one instruction a new sample is read from Buffer Memory A into Local Store Register 1 while the previous value in LSA(1) is written into Buffer Memory B. Also occurring is the increment of BARA, the address register associated with Memory A, by 40 at a time to get from channel to channel for this time sample. Similarly BARB is being incremented by 1 to put the demultiplexed samples in channel order. Finally, the counter (CTR) is being checked for overflow which would cause the Else condition to be executed (i.e., line 5). Checking of the CTR causes it to be incremented. If no overflow occurred then the combination of the SAVE in line 3 and the JUMP in line 4 will cause line 4 to be executed repeatedly until the overflow occurs (i.e., 40 times). Line 5 and 6 set up the next channel pass addresses and jump back to line 4 to demultiplex samples from the next channel. Line 6 also checks for completion of the entire task and line 5 resets the CTR.

An actual computer run of the same program follows. This is a copy of the listing.

72/06/28. 10.48.30.
PROGRAM AMIL

PLEASE ENTER TRANSLATION CONTROL PARAMETERS

INPUT FILE NAME IS
? MULTPLX
OUTPUT FILE NAME IS
? TEMP
SOURCE LISTING DESIRED
? YES
OBJECT LISTING DESIRED
? YES

SOURCE LISTING FOLLOWS

```

BARA = 1000 S
BARB = 10192 S
LSB(1),CTR = COMPI 40, SAVE S
INPUT(BUF(BARA),LSA(1)), OUTPUT(LSA(1),BUF(BARB)), INC BARB *
      BARA = BARA + 40, IF NOT UTRCV JUMP TO ACSAR S
LSB(1) = LSB(1) + 1, CTR = COMPI 40, STEP S
BARA = BARA - 1599, IF NOT ADRCV JUMP TO ACSAR S
END S

```

LABEL RESOLUTION TIME = 0. SECONDS

OUTPUT LISTING FOLLOWS

ADDR.	FIELDS(1-17)																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0	0	0	3	0	0	0	0	1	0	8	0	0	0	0	1000
2	0	0	0	0	0	3	0	0	0	1	0	8	0	0	0	0	10192
3	0	0	2	6	0	0	0	3	0	1	0	8	0	0	0	1	65495
4	4	1	5	0	3	2	1	0	2	1	8	1	1	0	1	0	40
5	0	0	0	5	0	0	0	3	0	1	4	4	0	1	0	1	65495
6	3	1	5	0	3	0	0	0	0	1	8	2	0	0	0	0	1599

NO COMPILATION ERRORS ENCOUNTERED

COMPILATION TIME = .36 SECONDS
END.

The following is a sample program which does nothing except illustrate many of the formats which are legal in AMIL.

72/06/23. 09.44.53.
PROGRAM AMIL

PLEASE ENTER TRANSLATION CONTROL PARAMETERS

```

INPUT FILE NAME IS
? AMILIN
OUTPUT FILE NAME IS
? TEMP
SOURCE LISTING DESIRED
? YES
OBJECT LISTING DESIRED
? YES

```

SOURCE LISTING FOLLOWS

```

BARA = LSA(7) + LSB(8) S
OUTPUT(LSA(7),BUF(BARA)), LSA(8) = LSB(7) OR BARB S
OUTPUT(LSB(8),BUF(BARA)), LSA(12) = LSA(12) ORV FSU(5) S
      OUTPUT(Z,BUF(BARA)), LSB(15) = FSU(5) + 7 S
      LSB(6) = 15 + FSU(6) S
      BARA = BARA OR LSB(5) S
      BARA = BARA + LSA(10) S
INC BARA, INC BARB S
      INC BARA, DEC BARB S
INC BARA S
INC BARB S

```

```

DEC BARA S
  DEC BARB S
  DEC BARA, DEC BARB S
DEC BARA, INC BARB S
LSB(2),CTR,LSA(8),BARA,BARB = LSA(9) - 345 S
BARA,BARB = BARA - 64, CTR = 64, SKIP S
BARB,BARA,SAR = LSB(BARB) + FSU(5) S
LSA(7),AC SAR = 563 S
IF ADROV JUMP TO 34, LSA(7),CTR = LSA(4) XOR LSB(5) S
CTR = 99, LSA(1),LSB(1) = LSB(6) S
SAR = NOLABEL S
.LABEL1 SKIP, SAR = LABEL2 S
SAR = COMP1 6 S
  CTR = 256 S
IF ZERO SAR = LABEL1 INTRET, INC BARA, INC BARB S
AC SAR = COMP1 65532 SAVE S
.LABEL3 OUTPUT(LSA(7),BUF(BARB)) JUMP TO LABEL5 S
CTR = LABEL3 S
OUTPUT(LSA(0),BUF(BARA)) OUTPUT(LSB(7),BUF(BARB)) INTRET S
INPUT(BUF(BARA),FSDR) INPUT(BUF(BARB),LSA(9)) STEP S
INPUT(BUF(BARA),LSA(7)) OUTPUT(LSB(4),BUF(BARB)) SKIP S
  LSA(0) = FSU(5) + LSB(8), IF CTRCV JUMP TO ACSAR S
INPUT(BUF(BARA),LSA(8)), LSB(9) = LSA(7) + 55, INC BARB S
  LSA(7) = FSU(3) OR LSB(5), JUMP TO Z S
CTR = COMP1 24 S
  CTR = COMP2 24 S
Z = LSB(3) COMP S
Z = CTR COMP S
SAR = 6 S
.LABEL2 INPUT(BUF(BARA),FSDR) OUTPUT(Z,BUF(BARB)), JUMP TO Z S
INPUT(BUF(BARA),LSA(8)), LSB(9) = LSA(7) + 55, INC BARB S
LSA(1) = LSB(9) IF MOST JUMP TO LABEL5 S
BARA = LSA(12) OR LSB(9), IF ZERO CALL S
LSB(8) = LSB(7) - LSA(3) S
  LSB(8) = LSA(3) - LSB(7) S
BARB = BARA - 26 S
LSB(BARB) = BARA + 99 S
IF LEAST SAVE, LSA(8) = LSA(8) + 1 S
IF NOT LEAST SAVE, LSA(8) = LSA(8) - 1 S
BARB = LSA(4) XOR BARB S
LSB(0) = LSB(0) RIGHT S THIS IS I RIGHT SHIFT OF LSB(BARB)
LSB(15) = 35 S
BARB = BARB + 1267 S
BARA = BARA + 2 S
BARB = BARA S
LSB(7) = CTR S
LSA(6) = SAR COMP S
LSB(BARB) = ACSAR S
LSA(BARA) = BARB S
INPUT(BUF(BARB),LSB(5)) S
  INPUT(BUF(BARA),LSB(3)), OUTPUT(LSA(5),BUF(BARB)) *
    LSA(8) = 12 + LSB(14), INC BARA, INC BARB, ACSAR = 12 S
LSA(7) = LSA(15) CIRC, JUMP TO LABEL3 S
Z = LSA(9) LEFT S
Z = LSA(8) AND LSB(4) S
.LABEL5 SAR = LABEL5 S
.NOLABEL LSA(BARA) = LSB(BARB) + 64 S
IF NOT ADROV LSA(15) = LSA(9) XOR LSB(6) JUMP TO 28 S
END S

```

The following is the translator's object listing for the above program.

LABEL RESOLUTION TIME = .02 SECONDS

OUTPUT LISTING FOLLOWS

ADDR.	FIELDS(1-17)																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0	0	0	3	0	0	0	0	1	3	1	7	8	0	0	0
2	0	0	0	0	0	0	3	0	1	1	6	3	7	7	8	0	0
3	0	0	0	0	0	0	3	0	3	5	1	6	12	8	12	0	0
4	0	0	0	0	0	0	0	3	0	5	12	1	0	0	0	15	7
5	0	0	0	0	0	0	0	3	0	6	12	1	0	0	0	6	15
6	0	0	0	0	3	0	0	0	0	1	7	3	0	5	0	0	0
7	0	0	0	0	3	0	0	0	0	1	9	1	10	0	0	0	0
8	0	0	0	0	2	2	0	0	0	1	0	0	0	0	0	0	0
9	0	0	0	0	2	1	0	0	0	1	0	0	0	0	0	0	0
10	0	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0
11	0	0	0	0	0	2	0	0	0	1	0	0	0	0	0	0	0
12	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
13	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
14	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0
15	0	0	0	0	1	2	0	0	0	1	0	0	0	0	0	0	0
16	0	0	0	6	3	3	3	3	0	1	0	2	9	0	8	2	345
17	0	0	1	5	3	3	0	0	0	1	8	2	0	0	0	0	64
18	0	0	0	4	3	3	0	0	0	5	5	1	0	0	0	0	0
19	0	0	0	2	0	0	3	0	0	1	0	8	0	0	7	0	563
20	3	0	4	6	0	0	3	0	0	1	3	11	4	5	7	0	34
21	0	0	0	5	0	0	3	3	0	1	4	6	0	6	1	1	99
22	0	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0	67
23	0	0	1	3	0	0	0	0	0	1	0	0	0	0	0	0	41
24	0	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0	65529
25	0	0	0	5	0	0	0	0	0	1	0	0	0	0	0	0	255
26	5	0	7	3	2	2	0	0	0	1	0	0	0	0	0	0	23
27	0	0	2	1	0	0	0	0	0	1	0	0	0	0	0	0	3
28	0	0	4	0	0	0	0	0	2	1	0	0	7	0	0	0	66
29	0	0	0	5	0	0	0	0	0	1	0	0	0	0	0	0	28
30	0	0	7	0	0	0	0	0	7	1	0	0	0	7	0	0	0
31	0	0	0	0	0	0	2	0	0	0	0	0	0	0	9	0	0
32	0	0	1	0	0	0	1	0	4	1	0	0	0	4	7	0	0
33	4	0	5	0	0	0	3	0	0	5	5	1	0	8	0	0	0
34	0	0	0	0	0	2	1	3	0	1	0	1	7	0	3	9	55
35	0	0	6	0	0	0	3	0	0	3	5	3	0	5	7	0	0
36	0	0	0	5	0	0	0	0	0	1	0	0	0	0	0	0	65511
37	0	0	0	5	0	0	0	0	0	1	0	0	0	0	0	0	65512
38	0	0	0	0	0	0	0	0	0	1	4	7	0	3	0	0	0
39	0	0	0	0	0	0	0	0	0	1	13	7	0	0	0	0	0
40	0	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0	5
41	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
42	0	0	0	0	0	2	1	3	0	1	0	1	7	0	8	9	55
43	1	0	4	0	0	0	3	0	0	1	4	6	0	9	1	0	66
44	5	0	3	0	3	0	0	0	0	1	3	3	12	9	0	0	0
45	0	0	0	0	0	0	0	3	0	1	3	2	3	7	0	8	0
46	0	0	0	0	0	0	0	3	0	1	3	2	3	7	0	8	0
47	0	0	0	0	0	3	0	0	0	1	8	2	0	0	0	0	26
48	0	0	0	0	0	0	0	3	0	1	8	1	0	0	0	0	99
49	2	0	2	0	0	0	3	0	0	1	0	4	8	0	8	0	0
50	2	1	2	0	0	0	3	0	0	1	0	5	8	0	8	0	0
51	0	0	0	0	0	3	0	0	0	1	2	11	4	0	0	0	0
52	0	0	0	0	0	0	0	3	0	1	4	14	0	0	0	0	0
53	0	0	0	0	0	0	0	3	0	1	0	8	0	0	0	15	35
54	0	0	0	0	0	3	0	0	0	1	10	1	0	0	0	0	1267

Appendix E

AMIL TRANSLATOR

Two sample runs of the AMIL translator have already been given. To help understand these outputs more fully, a brief explanation follows.

The AMIL translator is written in KRONOS 2.0 FORTRAN to run on the Control Data KRONOS time sharing service. To translate an AMIL program it must have been created and saved in a permanent file. The name of this file is the proper response to the first question the translator asks,

"INPUT FILE NAME IS".

The translator proceeds to read in the file and produce the bit patterns for each instruction. These patterns are stored in another file whose name should be given in response to the translator's second question,

"OUTPUT FILE NAME IS".

The next two questions ask whether the "SOURCE" (input) file or "OBJECT" (output) file should be listed:

"SOURCE LISTING DESIRED"

"OBJECT LISTING DESIRED"

The only allowable responses are "YES" or "NO." Any other response causes the question to be repeated (up to three times). In both examples the reply was "yes" and the listings are included.

If during translation syntax errors are encountered an error message is printed out along with the instruction. AMIL can respond with one of 42 error messages. These messages are listed below.

```
ERROR - MISSING ) TO CLOSE INPUT OR OUTPUT STAT.
ERROR - DUPLICATE USE OF LIT FIELD AT ADDR. = _____
ERROR - LABEL STARTS WITH ILLEGAL CHARACTER
ERROR - MISSING ( AFTER INPUT OR OUTPUT
ERROR - BUFFER READA OR READB SPECIFIED TWICE
ERROR - ILLEGAL DESTINATION FOR BUFFER READ
ERROR - ILLEGAL COMBINATION OF BUFFER OUTPUTS
ERROR - ILLEGAL ADDRESS SELECT FOR OUTPUT
ERROR - MISSING , AFTER SOURCE FOR OUTPUT OR INPUT
```

55	0	0	0	0	3	0	0	0	0	1	3	1	0	0	0	0	2
56	0	0	0	0	0	3	0	0	0	1	3	6	0	0	0	0	9
57	0	0	0	0	0	0	0	3	0	1	13	6	0	0	0	7	0
58	0	0	0	0	0	0	3	0	0	1	15	7	0	0	3	0	0
59	0	0	0	0	0	0	0	3	0	1	14	6	0	0	0	0	0
60	0	0	0	0	0	0	3	0	0	1	10	6	0	0	0	0	0
61	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	5	0
62	0	0	0	1	2	2	3	1	2	1	4	1	5	14	8	3	12
63	0	0	4	0	0	0	3	0	0	1	0	15	15	0	7	0	23
64	0	0	0	0	0	0	0	0	0	1	0	13	9	0	0	0	0
65	0	0	0	0	0	0	0	0	0	1	3	10	8	4	0	0	0
66	0	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0	66
67	0	0	0	0	0	0	3	0	0	1	4	1	0	0	0	0	64
68	3	1	4	0	0	0	3	0	0	1	3	11	9	6	15	0	23

NO COMPILATION ERRORS ENCOUNTERED

COMPILATION TIME = 3.23 SECONDS
 END.

ERROR - ILLEGAL MODIFIER FOLLOWING INC
ERROR - BARA USED TWICE AS A DESTINATION
ERROR - BARB USED TWICE AS A DESTINATION
ERROR - DUPLICATE AUXILLARY TRANSFER
ERROR - MISSING = IN AUXILLARY TRANSFER
ERROR - ILLEGAL SOURCE FOR AUXILLARY TRANSFER
ERROR - LABEL LONGER THAN NINE CHARACTERS
ERROR - UNRECCGNIZABLE STATEMENT
ERROR - MISSING = IN ADDER OPERATION
ERROR - ILLEGAL OPERAND IN ADDER OPERATION
ERROR - ILLEGAL INPUT OPERAND COMBINATION FOR ADDER

ERROR - UNKNOWN ADDER OPERATOR
ERROR - ILLEGAL CONDITION
ERROR - MISSING (AFTER FSU
ERROR - FSU SUBSCRIPT GREATER THAN 7
ERROR - ILLEGAL SOURCE FOR FSDR LOAD
ERROR - ILLEGAL SOURCE FOR OUTPUT
ERROR - ILLEGAL SOURCE FOR INPUT I/C INSTR.
ERROR - MISSING) AFTER FSU FIELD ID
ERROR - ILLEGAL FORMAT FOR JUMP SUCCESSOR
ERROR - DUPLICATE USE OF FSU OR FSDR
ERROR - INTEGER MORE THAN 5 DIGITS LONG
ERROR - TWO DIFFERENT LITERALS SPECIFIED
ERROR - INTEGER LARGER THAN 65535
ERROR - MISSING \$ OR * AT END OF THIS INSTRUCTION

ERROR - N GREATER THAN 9 IN SUBROUTINE BUILD
ERROR - UNABLE TO LOCATE
ERROR - ILLEGAL PLACEMENT OF * OR \$
ERROR - SAME L.S. USED TWICE AS SOURCE OR DEST.
ERROR - MISSING (AFTER LS
ERROR - ILLEGAL LOCAL STORE SUBSCRIPT.
ERROR - MISSING) AFTER LOCAL STORE SUBSCRIPT
ERROR - LOCAL STORE SUBSCRIPT_____ IS TOO LARGE

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Microprogramming Programming language Micro-translator Signal processing All Applications Digital Computer						

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Research Laboratory Washington, D.C. 20390		2a. REPORT SECURITY CLASSIFICATION Unclassified	
		2b. GROUP	
3. REPORT TITLE MICROPROGRAMMED CONTROL UNIT (MCU) PROGRAMMING REFERENCE MANUAL			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) An interim report on a continuing problem.			
5. AUTHOR(S) (First name, middle initial, last name) John D. Roberts, Jr.			
6. REPORT DATE August 15, 1972		7a. TOTAL NO. OF PAGES 45	7b. NO. OF REFS
8a. CONTRACT OR GRANT NO. NRL Problem B02-06		9a. ORIGINATOR'S REPORT NUMBER(S) NRL Report 7476	
b. PROJECT NO.		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
c.			
d.			
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY Department of the Navy (Naval Air Systems Command and Naval Electronics Systems Command), Washington, D.C. 20360	
13. ABSTRACT <p>The Microprogrammed Control Unit (MCU) is a high-speed, user-microprogrammable, executive, input-output processor and interrupt handler for the NRL Signal Processing Element (SPE), a part of the All Applications Digital Computer (AADC).</p> <p>This report describes the MCU architecture, a new programming language (AMIL), and its translator. AMIL (A Microprogramming Language) is a FORTRAN-like language which allows MCU users to write microprograms in a convenient format instead of binary bit patterns. The AMIL translator converts AMIL programs into the microinstruction bit patterns which the MCU will execute. This program marks the start of MCU software development.</p>			