50001

[redacted]

HOW TO USE THE CPS/1 MICRO-COMPUTER SYSTEM

• Canadian and Foreign Patents Pending.

CONTENTS

1	**INTRODUCTION**	**1**
1.1	**General Description of the CPS/1 System**	**1**
1.2	**Addressing**	**2**
1.3	**Instruction Formats**	**3**
1.4	**Organization**	**3**
2	**CENTRAL PROCESSOR INSTRUCTIONS**	**5**
2.1	**Memory Reference Instructions**	**5**
2.2	**Housekeeping Instructions**	**7**
2.3	**Accumulator and Working Register Instructions**	**10**
3	**PROGRAMMING**	**16**
3.1	**Introduction**	**16**
3.2	**Input/Output**	**16**
3.3	**Subroutines**	**19**
3.4	**Looping**	**20**
3.5	**Testing**	**21**
3.6	**Arithmetic**	**22**
4	**INSTRUCTION TIMES**	**23**
	GLOSSARY	**24**

1 INTRODUCTION

1.1 General Description of the CPS/1 System

The CPS/1 is the first in a series of general purpose microcomputer systems by Microsystems. The system central processor is contained entirely on one MOS LSI integrated circuit chip. System memory, depending on size, is contained on two or more additional LSI chips. The CPU contains two memory pointers: the usual program counter (PC), and a data pointer (DP), which allows logical, as well as physical separation, of program and data. Both the PC and the DP are 12. bits long and can directly address 4096. memory locations. A memory expander chip is available to extend addressing capability to 256K. locations (K = 1024.). Each memory location contains 4 bits of data (one nibble, which is half a byte).

microsystems international

MICROSYSTEMS INTERNATIONAL LIMITED, BOX 3529 STA.C, OTTAWA, CANADA K1Y 4J1

Printed in Canada

1.1 Continued . . .

The CPU uses a 12. bit address bus and a 4 bit, bi-directional data bus to connect to memory and input/output devices. These two buses, together with the 5 control lines, form a 21. line communications bus (COMBUS). A portion of the memory address space is allocated to input/output devices. Thus the CPU can use all its instructions which normally refer to memory to refer to I/O devices over the COMBUS. The COMBUS can be expanded to handle 8 bits of bi-directional data. This allows straight-forward interfacing of byte-oriented devices. Since each I/O port is a memory location, the interfacing of external devices is greatly simplified.

The processor performs a program by executing instructions fetched from consecutive memory locations as counted by the PC. Following the completion of an instruction, the PC is incremented by 2 or 4, depending on the length of the instruction. Sequential program flow is altered by modifying the PC during an instruction. The CPS/1 allows for conditional, unconditional, and subroutine call types of PC modification.

In addition to the PC and DP, two other registers are of interest to the programmer. These are the 4 bit accumulator (AC) and the 1 bit overflow register (OF). Data can be moved into or out of the AC from memory. The contents of a memory location can be ADD'ed or NAND'ed to the AC. The AC and OF can be rotated together left or right. The contents of AC and OF can be tested for various conditions.

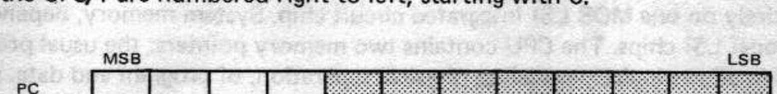
A unique feature of the CPS/1 is the set of working registers. There are eight 4 bit data registers and eight 12. bit address registers. These are implemented as part of the memory address space and are external to the CPU chip. Although access speed is not improved over regular memory, addressing overhead is reduced, resulting in shorter instructions. This allows the working registers to be very effective as scratch pad storage to hold such items as intermediate results and loop counts. As an additional convenience, the working registers can be addressed as normal memory locations.

Throughout this document decimal numbers are distinguished from octal numbers by use of periods after decimal numbers. This is the same convention as for CPS/1 Assembler language.

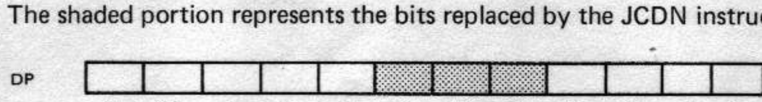
1.2 Addressing

The CPS/I can directly address 4096. nibbles of main memory. The first 32. nibbles must be RAM for implementation of the working data and address registers. The remaining memory space can be used for RAM, ROM or I/O addresses. Memory can be expanded to vertually any size by the use of field switching. The memory is logically divided into 256. nibble pages by the JCDN instructions, which replace the 8 low-order bits of the PC when a jump occurs. Program flow between pages takes place by normal PC incrementing between instructions, or by the exchange jump XPD.

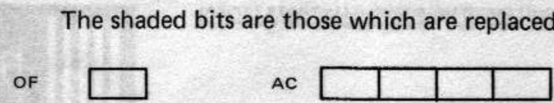
The bits of the registers in the CPS/1 are numbered right to left, starting with 0.



The shaded portion represents the bits replaced by the JCDN instructions.



The shaded bits are those which are replaced by the DP modifier.



1.2 Continued . . .

Program execution starts at location 1024. following a system reset. If less than 1024. nibbles of memory are installed, the appropriate location for program start should be assigned to address 1024..

1.3 Instruction Formats

There are two lengths of CPU instructions: 8 bit and 16. bit. Within each length class there are several different formats. The first four bits of each instruction determine the major functions.

FUNCTION
7 – 0

OPERATE AND MEMORY REFERENCE VIA UNMODIFIED DATA POINTER

FUNCTION	0	WORKING REGISTER ADDRESS
7 – 4	3	2 – 0

WORKING REGISTER REFERENCE

FUNCTION	1	DATA POINTER MODIFIER
7 – 4	3	2 – 0

MODIFIED DATA POINTER REFERENCE TO MEMORY

FUNCTION	IMMEDIATE DATA
7 – 4	3 – 0

SHORT IMMEDIATE (SINGLE NIBBLE)

FUNCTION	IMMEDIATE DATA
15. – 12.	11. – 0

LONG IMMEDIATE (THREE NIBBLES)

FUNCTION	CONDITION	ADDRESS
15. – 12.	11. – 8	7 – 0

LONG CONDITION JUMP

1.4 Organization

The CPS/1 is organized around two buses, the 4 bit data bus and the 12. bit address bus. Figure 1 shows a block diagram of the CPU. Data and instructions enter the CPU over the data bus. Instructions are stored in OPR and OPA. Data go to temporary storage (temp store) or to the AC. The PC/DP pair and their associated incrementer operate from the 12.bit address bus. The data bus and address bus are buffered off the chip by the data buffer and memory address buffer, respectively.

From the programmer's point-of-view, the CPS/1 is better represented by the diagram of Figure 2. The hierarchy of storage is indicated by the AC, working registers, and main memory. The DP and PC can address data and instructions in all of main memory. The working registers and the AC are addressed directly from the instructions. The adder/shifter/incrementer in the control section operate on the AC and on memory or register data.

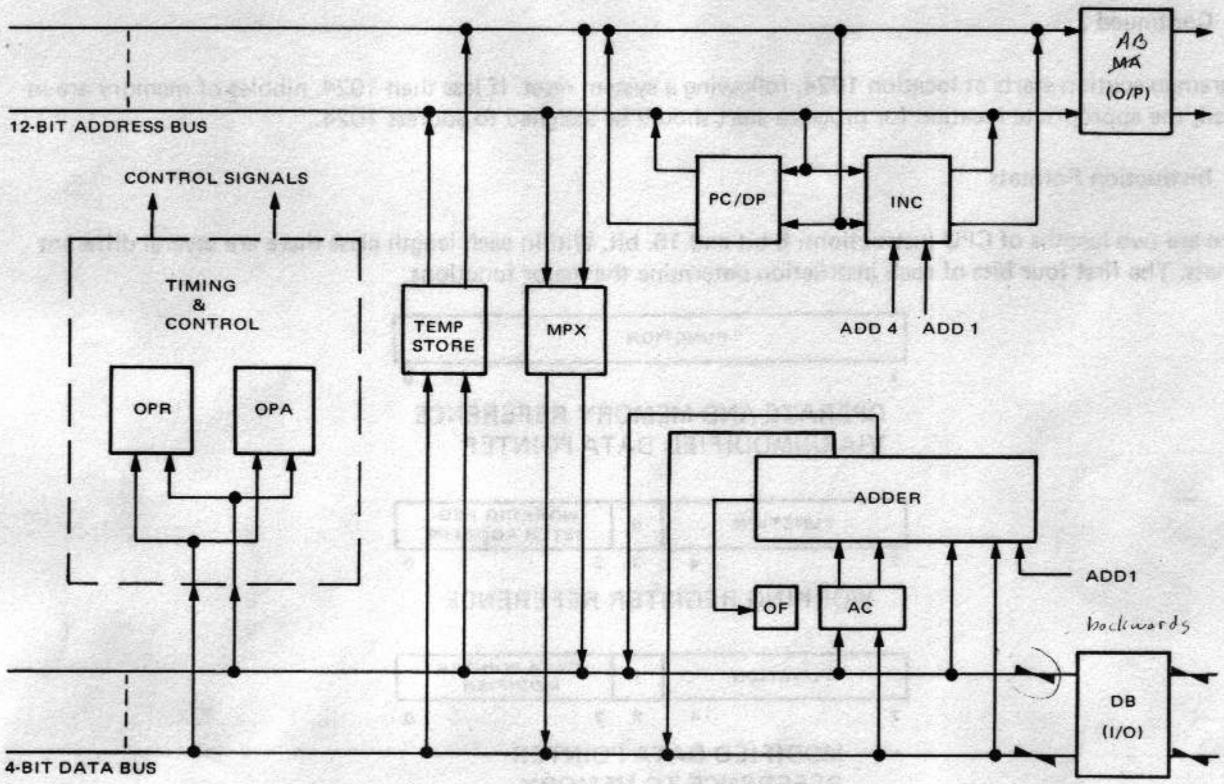


FIG. 1. CPS/1 BLOCK DIAGRAM

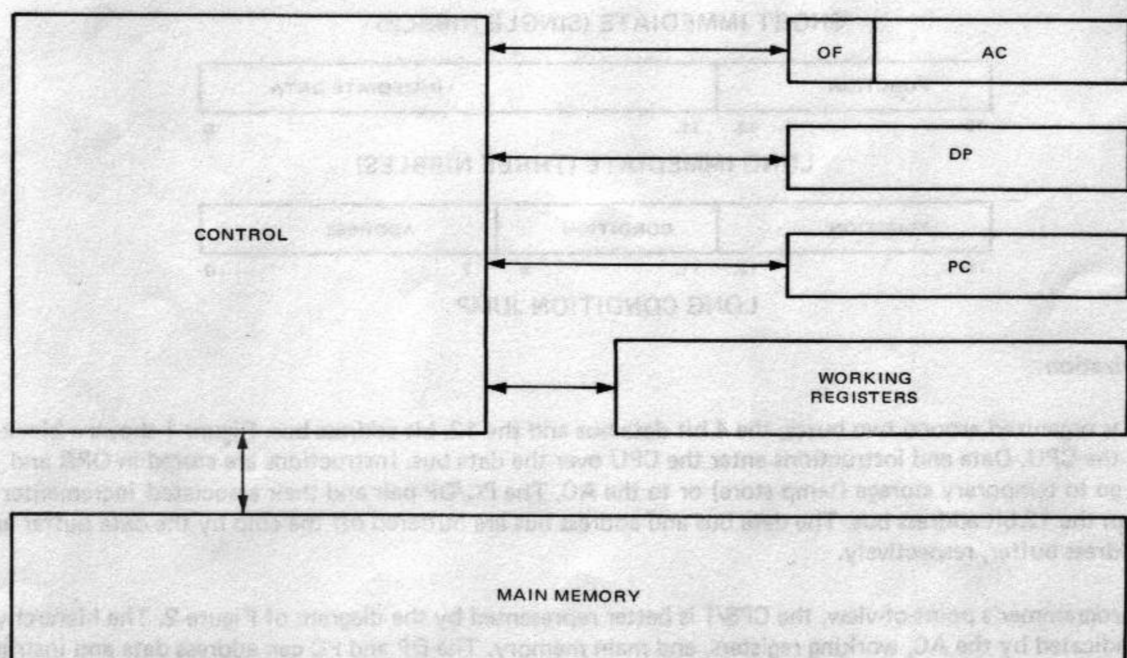


FIG. 2. CPS/1 FUNCTIONAL DIAGRAM

2 CENTRAL PROCESSOR INSTRUCTIONS

This chapter describes the instruction set of the CPS/1 and the effect of each instruction on the CPU registers and memory. For ease of presentation, the instructions are grouped by function. Timing is given for each instruction in clock cycles (memory cycles), as explained in Section 4. The mnemonics given in the boxes with each instruction are those recognized by the assembler (CPAL/1) with two exceptions: JCDN and OPR which are not, themselves, instructions but generic terms for classes of instructions. Where appropriate, examples of instruction usage and special techniques of a very general nature are included. More detailed programming examples are given in Section 3.

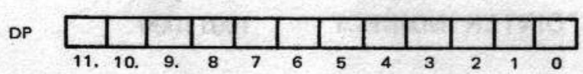
In the instruction descriptions, these conventions are used:

XXX...	represents immediate data or DP modifier bits,
DDD	represents a working data register (D0-D7),
AAA	represents a working address register (A0-A7),
CCCC	are the condition codes for jumps, and
PPPP	is the operate micro-code.

The following general rules can be stated about the instructions:

a) The overflow register (OF) is affected only by instructions which add to, increment, or rotate the accumulator, and those instructions which name the OF explicitly.

b) The modified-data-pointer instruction replaces bits 6 – 4 only for the duration of the instruction. At the end of the instruction the DP resumes the value it had at the beginning of the instruction.



c) Incrementing the data pointer and program counter cause wrap-around at address 4095.. The carry is discarded and the register is set to 0.

.
.
.
4094.
4095.
0000
0001
0002
.
.
.

2.1 Memory Reference Instructions

Memory references in the CPS/1 are made using the data pointer (DP). The value of the 12. bits in the DP specifies a memory location or the first of several consecutive locations. The DP is not automatically incremented following a reference; therefore, the program is free to modify the DP as desired.

Many of the memory reference instructions allow temporary modification to the DP. In these instructions the modification is in effect only during the fetching of the operand from memory. The modification is done by replacing bits 6 – 4 in the DP by the modifier from the instruction.

2.1.1 Programming Conventions. Instructions which do not modify the DP require only the operation code to be specified. Instructions which modify the DP require the modifying bits to be present in the operand field. Since the modifier is 3 bits, the number must be in the range 1 – 7 (0 is no modification). These bits are indicated by XXX in the instruction description.

2.1.2 LOAD and STORE Instructions.

LAD	LOAD ACCUMULATOR	00011000	3 cycles 2 nibbles

The data at the memory location specified by the data pointer are loaded into the accumulator. The previous contents of the accumulator are lost.

LAM	LOAD ACCUMULATOR (MODIFIED)	00011XXX	3 cycles 2 nibbles

The data pointer is temporarily modified by replacing bits 6 – 4 with XXX from the instruction. The data at the memory location specified by the modified data pointer are loaded into the accumulator. The previous contents of the accumulator are lost.

SAD	STORE ACCUMULATOR	00101000	3 cycles 2 nibbles

The accumulator is stored in the memory location specified by the data pointer. The contents of the accumulator are not affected.

LDID	LOAD DATA POINTER INDIRECT	10011000	5 cycles 2 nibbles

The data pointer is loaded with the three memory locations specified by the data pointer. The initial value of the data pointer is lost. The first location from which the data pointer is loaded MUST have an address whose two low order bits are 01.

Consider a stack of addresses which begin at location 425. To retrieve the first address from the top of the stack the following instructions are executed:

```
LDI     425     ;LOAD DP IMMEDIATE WITH STACK ADDRESS
LDID            ;FETCH FIRST ADDRESS FROM STACK
•
•
•
LAD             ;LOAD AC WITH DATA SPECIFIED BY
                ;FIRST ADDRESS OF STACK
```

Note that the instructions which modify the data pointer do not add the modifier to the DP, but rather replace bits 6 – 4. This provides for accessing arrays of data without changing the DP. An example is two 16 digit decimal registers located at addresses 400 and 420, respectively. The following instructions load the first digit from each register into the accumulator:

2.1.2 Continued . . .

```
        LDI     400     ;SET DP TO FIRST REGISTER
        .
        .
        LAD             ;LOAD FIRST DIGIT OF FIRST REGISTER,
        .               ;EFFECTIVE ADDRESS IS 400 (100000000)
        .
        LAM     1       ;LOAD FIRST DIGIT OF SECOND REGISTER,
        .               ;EFFECTIVE ADDRESS IS 420 (100010000)
        .
        SAD             ;TRANSFER DIGIT TO FIRST REGISTER
```

Thus, arrays of data can be stepped through without the need to constantly save and restore the data pointer.

2.1.3 Count Instructions.

ISZD	INCREMENT AND SKIP IF ZERO	01101000	4 cycles 2 nibbles

The contents of the memory location specified by the data pointer are incremented by 1. If the increment results in zero, the PC is incremented by 4, skipping 4 locations (one 4 nibble instruction or two 2 nibble instructions). The OF is not affected.

ISZM	INCREMENT AND SKIP IF ZERO (MODIFIED)	01101XXX	4 cycles 2 nibbles

The data pointer is temporarily modified by replacing bits 6 – 4 with XXX from the instruction. The contents of the memory location specified by the data pointer are then incremented by 1. If the increment results in zero, the PC is incremented by 4, skipping 4 locations. (One 4 nibble instruction or two 2 nibble instructions.) The OF is not affected.

The two previous instructions are used to count loop iterations or to successively modify a nibble for a series of operations. Consider a block of 12. locations which contains data from which it is desired to calculate a check sum (the sum of all items without regard to carry overflow). The following instructions can achieve this operation:

```
        LAI     -12     ;LOAD COUNT OF NEGATIVE 12
        SAD             ;STORE IT @DP
LOOP:   .
        .               ;PROCESSING...
        .
        ISZD            ;INCREMENT COUNT, SKIP IF ZERO
        JMP     LOOP    ;NOT DONE, RETURN FOR MORE PROCESSING.
```

2.2 Housekeeping Instructions

These are the instructions which affect the CPU registers other than the accumulator (AC). They are used to control program flow by affecting the program counter (PC) and to control data flow by affecting the data pointer (DP). There are instructions for jumping, testing, subroutine calling, incrementing and decrementing DP, and causing program delays. The only instructions in this group which require operands are LDI and jump.

2.2 Continued . . .

NOP2	TWO NIBBLE NO OPERATION	00000000	3 cycles 2 nibbles

The program counter is advanced 2 nibbles. None of the registers are affected.

NOP4	FOUR NIBBLE NO OPERATION	1110000000000000	5 cycles 4 nibbles

The program counter is advanced 4 nibbles. None of the registers are affected.

The NOPs are used to cause program delays and to pad a program for address alignment or provide skip protection. The program sequence below uses the ISZ instructions, but does not want the skip to affect the logic of the program (modulo 16. counter, for instance). The NOP4 is used in place of the normal JMP following the ISZ.

```
•
•
ISZD          ;INCREMENT COUNT
NOP4          ;DO NOTHING
SAR           ;NEXT INSTRUCTION AFTER ISZD
•
•
```

The sequence has the same effect if the ISZ operand is either zero or non-zero.

LDI	LOAD DATA POINTER IMMEDIATE	1101XXXXXXXXXXXX	5 cycles 4 nibbles

The X-bits from the instruction replace the current data pointer. This instruction is used to initially set the DP to a known value. Note that the DP is undefined following a system reset.

IDP	INCREMENT DATA POINTER	000010000 ?	3 cycles 2 nibbles

The data pointer (DP) is incremented by 1. If the DP is 7777 before IDP is executed, the value after incrementing is 0000 (the incrementing is done modulo 4096.). The OF is not affected.

DDP	DECREMENT DATA POINTER	10100000	5 cycles 2 nibbles

The data pointer (DP) is decremented by 1. If the DP is 0000 before DDP is executed, the value after decrementing is 7777. The OF is not affected.

XPD	EXCHANGE PROGRAM COUNTER AND DATA POINTER	00001111	3 cycles 2 nibbles

The contents of the data pointer (DP) and the program counter (PC) are swapped. The next instruction executed is taken from the new PC.

2.2 Continued . . .

The XPD instruction is the subroutine-call instruction. The normal sequence is as follows:

```
•
•
LDI     SUB1     ;LOAD ADDRESS OF SUBROUTINE
XPD              ;EXCHANGE PC & DP
•
•
```

Arguments are normally passed to the subroutine via the working registers. Return is made from the subroutine by simply giving another XPD, which returns control to the instruction following the calling point. If the DP is required in the subroutine, it can be saved in a working register. Note that the call sequence is pure, (i.e., re-entrant) since no memory is modified.

JCDN	JUMP ON CONDITION GROUP	1110CCCCXXXXXXXX	5 cycles 4 nibbles

If the conditions specified by bits CCCC are satisfied, then bits 7 – 0 of the program counter (PC) are replaced by the X-bits of the instruction. If the conditions are not satisfied, the effect is an NOP.

Special mnemonics are assigned to the conditions for jumps, as follows:

CONDITION	LETTER
OF = 1	T (true)
OF = 0	F (false)
AC > 9	G (greater than)
AC ≤ 9	L (less than or equal)
AC = 0	Z (zero)
AC ≠ 0	N (non-zero)

This can be combined to yield the 16. values of the 4 bit condition field, as follows:

CCCC	MNEMONIC	CONDITIONS
0	NOP4	never jump
1	JG	AC > 9.
2	JZ	AC = 0
3	JGZ	AC > 9. OR, AC = 0
4	JT	OF = 1
5	JTG	OF = 1 OR, AC > 9.
6	JTZ	OF = 1 OR, AC = 0
7	JTGZ	OF = 1 OR, AC > 9. OR, AC = 0
8	JMP	always jump
9.	JL	AC ≤ 9.
10.	JN	AC ≠ 0
11.	JLN	AC ≤ 9. AND, AC ≠ 0
12.	JF	OF = 0
13.	JFL	OF = 0 AND, AC ≤ 9.
14.	JFN	OF = 0 AND, AC ≠ 0
15.	JFLN	OF = 0 AND, AC ≤ 9. AND, AC ≠ 0

2.2 Continued . . .

The jump is an "on page" jump, not a relative jump. For this purpose the memory is divided into pages of 256. locations, i.e., page boundaries have 8 low-order zeros in their addresses. Since the jump consists of replacing the low-order bits of the PC, the JMP instructions cannot cross these page boundaries, not even if the JMP is executed on the last location of a page.

The conditional jumps are used to alter program flow in response to data-dependent conditions. As an example, consider a loop of instructions to be executed until the overflow register (OF) is set or the accumulator is zero as follows:

```
LOOP:   .                     ;INSTRUCTIONS FOR PROCESSING
        .                     ;DATA...
        .
        JTZ     LOOP          ;DO AGAIN IF NOT SATISFIED
        .
        .
```

2.3 Accumulator and Working Register Instructions

This group of instructions references the accumulator (AC) and/or the 16. working registers. There is one arithmetic set of instructions and one logical set which reference the AC and memory.

2.3.1 Programming Conventions. Where a working register is required it is supplied by the programmer as an operand. The 8 data registers are named D0 through D7, and the 8 address registers, A0 through A7. The working registers can also be addressed via the data pointer (DP).

2.3.2 Arithmetic Instructions.

ADD	ADD MEMORY TO ACCUMULATOR	01011000	4 cycles 2 nibbles

The data at the memory location specified by the data pointer are added to the contents of the accumulator (AC). The original contents of AC are lost. If there is a carry from the high order bit of the AC, the overflow register (OF) is unconditionally set to 1.

ADM	ADD MEMORY TO ACCUMULATOR (MODIFIED)	01011XXX	4 cycles 2 nibbles

The data pointer (DP) is temporarily modified by replacing bits 6 – 4 with XXX from the instruction. The data at the memory location specified by the modified DP are added to the contents of the accumulator (AC). The original contents of the AC are lost. If there is a carry from the high order bit of the AC, the OF is unconditionally set to 1.

ADR	ADD REGISTER TO ACCUMULATOR	01010DDD	4 cycles 2 nibbles

The data from a working data register DDD is added to the contents of the accumulator (AC). The original contents of AC are lost. If there is a carry from the high order bit of AC, the OF is unconditionally set to 1.

The ADD instructions can be used, in conjunction with the conditional jump instructions, to do binary and decimal arithmetic. Consider two binary, multiple-precision numbers, A and B, each 12. nibbles (48. bits) long, and stored at 1000 and 1020, respectively. The following routine adds B to A and stores the result C (located at 1040 in memory).

2.3.2 Continued . . .

```
         LDI      1000     ;LOAD DP WITH ADDRESS OF 'A'
         LAI      -12      ;SET UP LOOP COUNTER
         SAR      D2       ;IN WORKING REGISTER D2
         CLF               ;CLEAR OVERFLOW REGISTER
LOOP:    CLARL             ;CLEAR AC, LOAD CARRY INTO AC
         ADD               ;ADD IN A NIBBLE OF 'A'
         ADM      1        ;ADD IN A NIBBLE OF 'B'
         SAM      2        ;STORE RESULT NIBBLE, CARRY IN OF
         IDP               ;INCREMENT DP TO NEXT NIBBLE
         ISZR     D2       ;DONE YET?
         JMP      LOOP     ;NO - RETURN FOR NEXT NIBBLE
         JT       OVFL     ;YES, GO TO OVFL IF OVERFLOW...
         .
         .
```

In this example, if a carry occurs in the first ADD, none can occur in the ADM, because the AC is zero.

The same routine as just described, with slight modification can perform decimal addition. Assuming D3 contains the constant 6, the following instructions can perform this task:

```
         .
         .
LOOP:    CLARL             ;CLEAR AC, LOAD CARRY
         ADD               ;ADD DIGIT OF 'A'
         ADM      1        ;ADD DIGIT OF 'B'
         JFL      STORE    ;SKIP NEXT INSTRUCTION IF AC <OR= 9 & OF = 0
         ADR      D3       ;ADD CORRECTION
STORE:   SAM      2        ;STORE RESULT DIGIT
         .
         .
```

The number 6 is added because the AC overflows at 15. rather than at 10.. If the result is greater than 9. a carry should result. The carry is forced by adding 6.

Consider 9. + 9. = 18., which is 8 with a 1 carry:

```
   1001      9.
  +1001     +9.
 1 0010     12.
  +0110     +6
 1 1000     18.
```

Subtraction is done by complementing the subtrahend and adding. The 1's complement is used for binary, and the 9's complement for decimal.

2.3.2 Continued . . .

The use of 1's complement arithmetic allows for simple complementing and carry propagation as in addition. Using the previous example of A and B, together with C, C is calculated as : C = A −B, to 48. bits of precision, by executing the following instructions:

```
        .
        .
LOOP1:  LAM     1       ;GET NIBBLE OF 'B'
        COM             ;COMPLEMENT IT
        SAR     D4      ;SAVE IT IN A WORK REGISTER
        CLARL           ;LOAD CARRY FROM PREVIOUS NIBBLE
        ADD             ;ADD NIBBLE OF 'A'
        ADR     D4      ;ADD COMPLEMENT OF 'B'
        SAM     2       ;STORE RESULT IN 'C'
        IDP             ;INCREMENT DP TO NEXT NIBBLE
        ISZR    D2      ;DONE YET?
        JMP     LOOP1   ;NO, DO NEXT NIBBLE
        JF      DONE    ;DONE IF NO CARRY
        LDI     1040    ;ADD ONE TO 'C'
        LAI     -12     ;SET UP COUNT
        SAR     D2      ;IN D2
        CLF             ;CLEAR OF
LOOP2:  CLARL           ;GET CARRY
        ADD             ;ADD NIBBLE OF 'C'
        SAD             ;AND STORE RESULT
        IDP             ;INCREMENT DP FOR NEXT
        ISZR    D2      ;DONE ?
        JMP     LOOP2   ;NO, DO MORE...
DONE:   .
        .
```

The previous instructions first calculate the 1's complement of B, saving it in a working register. The process then continues as in addition. After the addition is finished, the result C is incremented at LOOP 2 if there is a carry at the end of the addition.

Decimal subtraction is done the same way except for the calculation of the 9's complement. The 9's complement is formed for each digit by subtracting the digit from 9. (or by adding 7, which is the 2's complement of 9, and then forming the 2's complement of the addition). Thus decimal subtraction can be done by replacing the first 6 instructions in the above example with the following instructions:

```
        .
        .
        CLARL           ;SAVE CARRY
        LAR     D6      ;LOAD -9
        ADD     1       ;ADD DIGIT OF 'B'
        COM             ;COMPLEMENT AND
        IAC             ;INCREMENT TO SET 9'S COMPLEMENT
        CLF             ;CLEAR OF
        ADR             ;ADD PREVIOUS CARRY
```

2.3.2 Continued . . .

```
         ADD               ;ADD DIGIT OF 'A'
         JFL      DONE     ;SKIP IF NO CARRY
         ADR      D3       ;ADD CORRECTION
DONE:    .
         .
```

and inserting between the ADD and SAD in LOOP 2 the following:

```
         .
         .
         JFL      NOC      ;CHECK IF CARRY
         ADR      D3       ;ADD CORRECTION
NOC:     .
         .
```

2.3.3 Logical Instructions.

NAD	NAND MEMORY WITH ACCUMULATOR	01001000	4 cycles 2 nibbles

The data at the memory location specified by the data pointer are NAND'ed with the contents of the accumulator (AC). The original contents of the AC are lost and are replaced by the NAND'ed result.

NAM	NAND MEMORY WITH ACCUMULATOR (MODIFIED)	01001XXX	4 cycles 2 nibbles

The data pointer (DP) is temporarily modified by replacing bits 6 – 4 with XXX from the instruction. Then a NAD is executed with the modified DP.

NAR	NAND REGISTER WITH ACCUMULATOR	01000DDD	4 cycles 2 nibbles

The data from working register DDD are NAND'ed with the contents of the accumulator (AC). The original contents of the AC are lost and are replaced by the NAND'ed result.

The NAND instruction provides the programmer with a single logical instruction which can easily provide the AND and OR functions.

INPUT	AND	OR
00	0	0
01	0	1
10	0	1
11	1	1

These functions can be computed directly by the following algorithms:

FUNCTION	DEFINITION	ALGORITHM
AND	$\overline{\overline{A \cdot B}}$	COMPLEMENT OUTPUT OF NAND
OR	$\overline{\overline{A} \cdot \overline{B}}$	COMPLEMENT INPUTS TO NAND

2.3.3 Continued . . .

The following examples illustrate the use of these algorithms.

EXAMPLE 1: AND memory with AC

```
NAD                 ;NAND MEMORY TO AC
COM                 ;COMPLEMENT TO GET AND
```

EXAMPLE 2: OR memory with AC

```
COM                 ;COMPLEMENT AC
SAR     1           ;SAVE IT
LAD                 ;LOAD ARGUMENT
COM                 ;COMPLEMENT IT
NADR    1           ;NAND TO GET 'OR'
```

2.3.4 Working Register Instructions.

LAR	LOAD ACCUMULATOR FROM REGISTER	00010DDD	3 cycles 2 nibbles

The data from working data register DDD is loaded into the accumulator (AC).

SAR	STORE ACCUMULATOR IN REGISTER	00100DDD	3 cycles 2 nibbles

The contents of the accumulator (AC) are stored in working data register DDD. The AC is not affected.

LAI	LOAD ACCUMULATOR IMMEDIATE	0111XXXX	4 cycles 2 nibbles

The four bits XXXX from the instruction are loaded into the accumulator.

The LAI instruction is useful for loading loop counts. The XXXX bits can be a 2's complement number which is incremented to zero. The following instructions illustrate this method:

```
        LAI     -5      ;SET UP COUNTER (1Ø11)
        SAR     DØ      ;IN DATA REGISTER Ø
        •
        •
TOP:    •               ;PROCESSING
        •
        ISZR    DØ      ;INCREMENT & SKIP IF ZERO
        JMP     TOP
        •
```

2.3.4 Continued . . .

LDR	LOAD DATA POINTER FROM REGISTER	10010AAA	5 cycles 2 nibbles

The address from working address register AAA is loaded into the data pointer (DP).

SDR	STORE DATA POINTER IN REGISTER	10000AAA	5 cycles 2 nibbles

The contents of the data pointer (DP) are stored in working address register AAA. The DP is not affected.

SIDR	STORE INCREMENTED DATA POINTER IN REGISTER	10001AAA	5 cycles 2 nibbles

The data pointer (DP) is incremented by 1, then stored in working address register AAA. The incremented DP is still available to the program. Incrementing a DP containing all 1's yields all 0's.

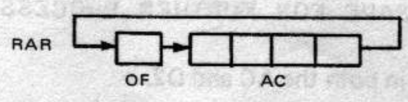
ISZR	INCREMENT AND SKIP IF ZERO REGISTER	01100DDD	4 cycles 2 nibbles

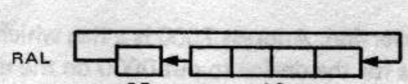
The working data register DDD is incremented by 1. If the result of the increment is zero (was all 1's), the next 4 nibbles of program are skipped.

OPR	ARITHMETIC OPERATE GROUP	0011PPPP	3 cycles 2 nibbles

This set of instructions operates on the accumulator (AC) and the overflow registers (OF). Each instruction performs one or more functions. The following table gives the value of PPPP, the mnemonic for that value (for use with the assembler) and the actions executed.

PPPP	MNEMONIC	DESCRIPTION
0	COM	complement AC
1	RAR	rotate OF and AC right one bit
2	RAL	rotate OF and AC left one bit
3	IAC	increment AC by 1, OF set if AC is 1111
4	CLA	clear AC
5	CLARR	CLA then RAR
6	CLARL	CLA then RAL
7	STA	set AC = 0001
8	CLF	clear OF
9.	CLFRR	CLF then RAR
10.	CLFRL	CLF then RAL
11.	IACCF	IAC then CLF
12.	STF	set OF = 1
13.	STFRR	STF then RAR
14.	STFRL	STF then RAL
15.	IACSF	IAC then STF





2.3.4 Continued . . .

The sequence to form the 2's complement of two numbers is as follows:

```
COM                ;COMPLEMENT AND
IAC                ;INCREMENT. (OF DESTROYED)
```

To save the contents of the OF:

```
CLARL              ;CLEAR AC AND ROTATE IN OF
SAR      DØ        ;STORE AC
```

To retrieve the contents of the OF:

```
LAR      DØ        ;LOAD OF INTO AC
CLFRR              ;ROTATE BACK INTO OF (CLF CLEARS AC)
```

3 PROGRAMMING

3.1 Introduction

This section covers in detail the techniques of programming the CPS/1 to perform particular tasks. Emphasis is placed on those unique features ot the CPS/1 which are useful in micro-systems. Throughout the following discussions the mnemonics given in Section 2 are used to identify the instructions. The examples given are as processed by CPAL/1, the CPS/1 assembler.

When the CPS/1 is initialized (reset), the only register defined is the program counter (PC). The PC is set to 2000 and program execution is initiated. From this point on the instructions are executed sequentially unless modified by a jump or subroutine call.

3.2 Input/Output

Bringing data into the CPS/1 is greatly simplified over most other mini/micro-systems. Each external device connected to the CPS/1 is assigned (responds to) one or more addresses in the memory space. For example, consider an analog to digital converter (A/D) which responds as follows: each time address 6000 is sent down the address bus, the A/D places the numeric (4 bit) representation of its analog input on the data bus. This is simply the action of a normal memory location being read. Sending out the address results in the data in the location addressed being placed on the data bus. Each address must have only one device associated with it; a memory location and an A/D on the same address would cause errors.

The instructions required to read the A/D (input a value from the A/D) into the AC and also save it are as follows:

```
LDI      7ØØ1      ;LOAD DP WITH A/D ADDRESS
LAD                ;LOAD AC WITH A/D VALUE
SAR      D2        ;AND SAVE FOR FURTHER PROCESSING
```

After this sequence is executed, the A/D value is in both the AC and D2.

The A/D takes an indeterminate time to do a conversion. Address 7000 is a flag which indicates whether the conversion is done or not done. This flag provides for the device to put 0000 on the data bus if not done and 1111 on the bus if done. The sequence to check if done, then read the data is as follows:

3.2 Continued . . .

```
         LDI     7000     ;LOAD DP WITH ADDRESS OF FLAG
TEST:    LAD              ;FETCH FLAG
         JZ      TEST     ;JUMP BACK IF NOT DONE.
         IDP              ;INCREMENT DP TO ADDRESS OF VALUE
         LAD              ;LOAD VALUE
         SAR     D2       ;SAVE IT
```

This program stays in the test loop (LAD, JZ) until the A/D signals ready by sending a non-zero flag.

Most devices like the A/D make available more than 4 bits. In this case, successive nibbles of the value are assigned successive addresses. In addition, other control functions can be assigned. Figure 3 shows a complex A/D system with a 16-line multiplexer. The address assignment can be as follows:

LOAD/STORE	ADDRESS	FUNCTION
STORE	7000	SELECT LINE : data from AC selects line 0 – 15.
STORE	7001	START CONVERSION : data from AC is ignored
LOAD	7002	DONE FLAG : 0 return indicates not done
LOAD	7003	VALUE : low order 4 bits of A/D value
LOAD	7004	VALUE : high order bits of A/D value
STORE	7005	RESET : reset A/D converter, clear flag

A routine to select a line (the line number in the AC to start) and read the value of the analog signal into D5, D6 is as follows:

```
 1   READ:   LDI    7000    ;LOAD ADDRESS OF A/D INTO DP
 2           SAD            ;SEND LINE NUMBER
 3           IDP            ;INCREMENT TO START FUNCTION
 4           SAD            ;SEND START (DATA IS IGNORED)
 5           IDP            ;INCREMENT TO  ADDRESS OF DONE FLAG
 6   WAIT:   LAD            ;LOAD FLAG
 7           JZ     WAIT    ;JUMP BACK IF ZERO
 8           IDP            ;IT IS READY, INCREMENT TO VALUE
 9           LAD            ;LOAD LOW ORDER BITS
10           SAR    D5      ;SAVE THEM
11           IDP            ;INCREMENT TO NEXT NIBBLE
12           LAD            ;LOAD NEXT NIBBLE
13           SAR    D6      ;AND STORE IT.....
```

The following is a step-by-step analysis of this program:

1. The data pointer is loaded with the address 7000, the first address assigned to the A/D subsystem.

2. The SAD instruction normally stores the AC in the memory location specified by the data pointer, but connected to address 7000 is not memory but the multiplexer of the A/D. The 4 bits sent over the data bus by the SAD instruction select the proper line.

3. The IDP adds 1 to the DP. The DP now specifies location 7001.

4. Another SAD selects the start function in the A/D. The data from the AC which the CPU places on the data bus are ignored by the A/D. The start pulse initiates conversion of the analogue signal.

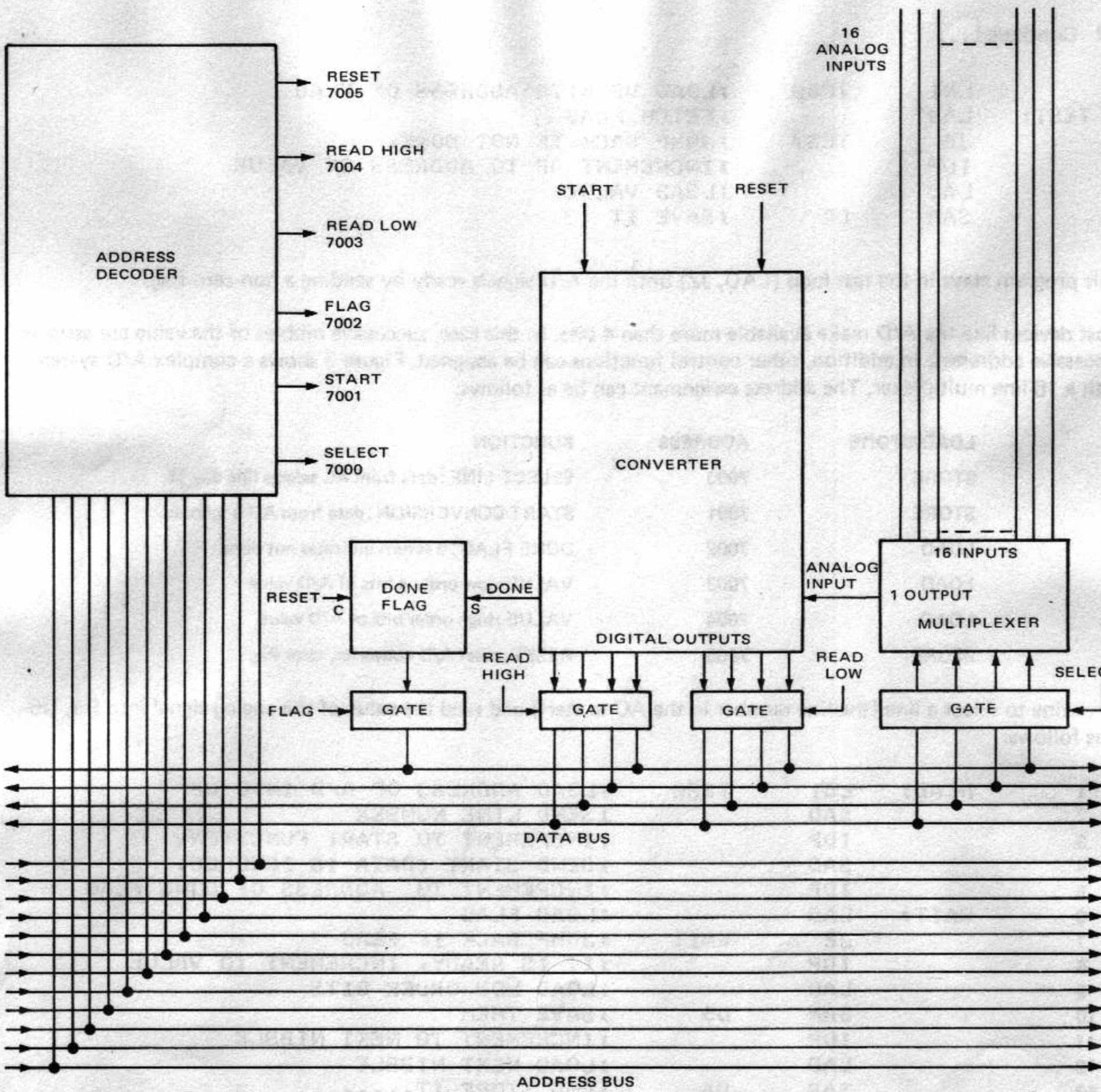


FIG. 3. A/D CONVERTER SUBSYSTEM

3.2 Continued . . .

5. Following this IDP, the DP contains the address of the A/D DONE flag.

6. The LAD instruction reads the contents of the A/D DONE flag into the AC (i.e., when the A/D detects address 7002, it places the flag on the data bus).

7. If the value read into the AC from the data bus is 0 the JZ test (jump if zero AC) is successful, control returns to WAIT (instruction 6), and the test sequence is repeated. Only when the LAD instruction in 6 returns a 1 does control fall through to 8.

3.2 Continued . . .

8. The data pointer is incremented to 2003 to fetch in the first 4 bits of the numeric value of the analogue signal selected in 2.

9. The LAD instruction causes the A/D unit to place the 4 low-order bits of the value on the data bus and load them into the AC.

10. The SAR stores the AC in a working register in preparation for bringing in the high-order bits.

11, 12, 13 repeat 8, 9, 10 for the high order bits.

3.3 Subroutines

A subroutine is a unit of program whose function is required in more than one place in a larger program but appears only once. Each time the function is required, the subroutine is called. This arrangement results in less memory being required for a particular program and less time required for program preparation.

The subroutine (located at 2000) to perform the task of adding D2 to D3 and storing the result in D4 is as follows:

```
2000    CALC:   LAR     D2      ;LOAD D2 INTO AC
2002            ADR     D3      ;ADD IN D3
2004            SAR     D4      ;STORE RESULT
2006            CLA             ;CLEAR AC
2010            XPD             ;RETURN TO CALLER
```

The last instruction, XPD, is used to both call a subroutine and return from a subroutine. To call CALC from another part of the program the following instructions are required:

```
        LDI     2000    ;LOAD ADDRESS OF SUBROUTINE IN DP
        XPD             ;PLACE ADDRESS OF CALC IN PC,
                        ;AND ADDRESS OF NEXT INSTRUCTION
                        ;IN DP.
```

The CPAL/1 assembler allows the use of symbolic addresses in place of numeric addresses. The instruction below has the same result as the previous one; i.e., the assembler recognizes that CALC begins at 2000.

```
        LDI     CALC
        XPD
```

After entering a subroutine, the DP contains the address of the instruction immediately following the XPD which called the subroutine. This is called the return address. If the DP is to be modified by the subroutine, the DP (return address) must be saved in a working register.

Sometimes the arguments required by the subroutine are not contained in the working registers but are known only at the point of calling. This situation can be handled by placing the arguments immediately following the XPD which calls the subroutine. After the XPD, the DP contains not the return address but the address of the argument. After the argument is fetched by the subroutine, the subroutine increments the DP to the proper return address. The instruction sequence is as follows:

```
2000    LDI     SUBA    ;LOAD ADDRESS OF SUBA
2004    XPD             ;CALL
2006    ..DATA..        ;ARGUMENT
2007    ..NEXT INSTRUCTION...
```

3.3 Continued . . .

The subroutine SUBA may be as follows:

```
SUBA:    LAD               ;FETCH ARGUMENT (DP=2006)
         IDP               ;INCREMENT DP TO 2007
         •
         •                 ;PROCESSING
         •
         XPD               ;RETURN
```

In place of the actual data, the argument(s) following a subroutine call can be addresses of data, as follows:

```
         LDI     SUBB      ;LOAD ADDRESS OF SUBROUTINE
         XPD               ;CALL SUBROUTINE
         ••ADDRESS OF ARGUMENT
         ••NEXT INSTRUCTION
         •
         •
SUBB:    SDR     A2        ;SAVE RETURN ADDRESS
         LDID              ;LOAD DP WITH ADDRESS OF ARGUMENT
         LAD               ;LOAD ARGUMENT
         •
         •                 ;PROCESSING
         •
         LDR     A2        ;RESTORE RETURN ADDRESS
         IDP               ;BUMP IT PAST ARGUMENT
         IDP
         IDP
         XPD               ;AND RETURN
```

In the example the data pointer must be incremented by 3 after it is restored in order to move past the address of the argument. The hardware requires that the last octal digit of the DP be 1 or 5 prior to LDID (i.e., $XXXXXXXXXX01_2$).

3.4 Looping

Looping is the process of repeating a group of instructions a number of times. In one type the number of times can be known when the program is written or can be calculated during program execution. Another type of loop is one in which the loop is done until some quantity is zero or equal to another quantity. The inclusion of "immediate" instructions makes simple loops easy to set up, and the ISZ instruction makes loops easy to control. The following instructions illustrate a typical loop:

```
         LAI     -5        ;SET UP COUNT
         SAR     D3        ;IN WORK REGISTER
         •
TOP:     •
         •                 ;PROCESSING
         •
         ISZR    D3        ;INCREMENT COUNT, SKIP IF ZERO
         JMP     TOP       ;NOT DONE, RETURN FOR MORE PROCESSING
```

In this loop a working register is loaded with negative 5 and incremented until it is zero. The ISZR instruction has the dual function of counting and testing.

3.4 Continued . . .

The CPAL/1 assembler treats a number preceeded by a minus sign as a 2's complement. The following binary digits are bit-patterns of some 2's complement negative numbers (2's complement is generated by complementing the magnitude of the number, then adding 1, disregarding any carry):

```
 -0   =  0000
 -1   =  1111
 -2   =  1110
-14.  =  0010
-15.  =  0001
-16.  =  0000
```

The technique of the above loop is limited to counts of 16. or less. By extending the loop count over two locations, the count can be increased to 256.. The following instructions illustrate a count of 150. (15. X 10.).

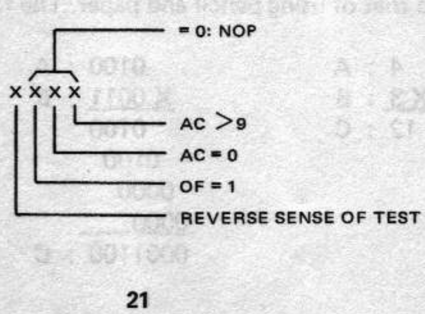
```
         LAI     -15        ;DO OUTER LOOP 15 TIMES
         SAR     DØ
LOOP1:   LAI     -1Ø        ;DO INNER LOOP 1Ø TIMES
         SAR     D1
LOOP2:   .
         .                  ;PROCESSING
         .
         ISZR    D1         ;COUNT INNER LOOP
         JMP     LOOP2
         ISZR    DØ         ;COUNT OUTER LOOP
         JMP     LOOP1
```

This technique can be extended to use as many locations as required. Although the DP cannot be tested, it can be used to count up to 4096 (12. bits) with the SIDR instruction, as follows:

```
         LDI     Ø          ;LOAD DP WITH ZERO
         SDR     A2         ;STORE IN WORK REGISTER
LOOP:    .
         .                  ;PROCESSING
         .
         LDR     A2         ;INCREMENT DP
         SIDR    A2         ;WHICH IS REALLY COUNTER...
```

3.5 Testing

Only the very simplest of programs require no testing of the conditions of program or data. Most programs use the ability to test data to determine the future action or flow of the program. The CPS/1 has a very powerful set of instructions for testing. The AC can be tested alone or in combination with the OF. The JCDN group of instructions provides the ability to micro-program the test conditions. The "condition" nibble is encoded as follows:



3.5 Continued . . .

The reverse sense is applied after the other tests; therefore, all the stated conditions must be true if the sense is reversed. This can be summarized by the following two statements:

Branch if AC $>$ 9. OR AC = 0 OR OF = 1.
Branch if AC $\leqslant$ 9. AND AC = 1 AND OF = 0.

In addition to the above conditions, numbers can be compared in magnitude by subtraction. If B is subtracted from A, the OF gives the relative size of A : B, as follows:

OF	A : B
0	A $\leqslant$ B
1	A $>$ B

The subtraction in this case is 1's complement addition. The instructions are as follows:

```
LAR     DØ      ;LOAD 'B'
COM             ;NEGATE IT   < CLF/AC
ADR     D1      ;ADD IN 'A'
JT      BIG     ;JUMP IF A>B
JZ      EQU     ;JUMP IF A=B
•               ;HERE IF A<B
```

This routine has three exits: A $>$ B, A = B, A $<$ B. It is often necessary to mask certain bits in a group. The NAND instruction is used for this purpose.

The following instructions are used to set the two middle bits of the AC to 0.

```
SAR     D4      ;SAVE AC
LAI     9       ;LOAD BIT PATTERN
NAR     D4      ;NAND WITH AC
COM             ;COMPEMENT FOR 'AND'
```

Another technique for testing is to rotate the OF and AC together in order to separate bits of a nibble. Bit 1 of the AC can be tested by masking and executing JZ or JN. It can also be tested as follows:

```
RAL             ;MOVE BIT 1 INTO BIT Ø
RAL             ;MOVE BIT Ø INTO OF
JT      SET     ;JUMP IF BIT WAS =1
```

3.6 Arithmetic

Addition and subtraction of binary and decimal numbers is explained in Section 2. Binary multiplication is done in the CPS/1 in a manner similar to that of using pencil and paper. The following calculation illustrates C = A X B:

```
   4 : A            0100 : A
  X3 : B          X 0011 : B
  12 : C            0100
                   0100
                  0000
                 0000
                 0001100 : C
```

3.6 Continued . . .

The flow chart in Fig. 4 gives the algorithm for this type of multiplication (starting with the right hand bit of B).

4 INSTRUCTION TIMES

Basic timing for the CPS/1 is supplied by a dual-phase clock. Since this clock is derived external to the CPU chip it can be adjusted by the system designer to meet demands of memories and other components. In the following discussion, times are given in both clock cycles and seconds, based on a 900ns clock. The two phases, are labled ϕ_1 and ϕ_2 as shown in Fig. 5.

Fetching an instruction requires 2 cycles: 1.8μs (2c). This is the time required to bring the first two (or only) nibbles of the instruction into the CPU. The instructions then require 1, 2 or 3 cycles to execute. The instruction fetch indicator (IF) control line is true during the first two cycles of each instruction. The lists on the next page divide the instructions into three classes by execution time.

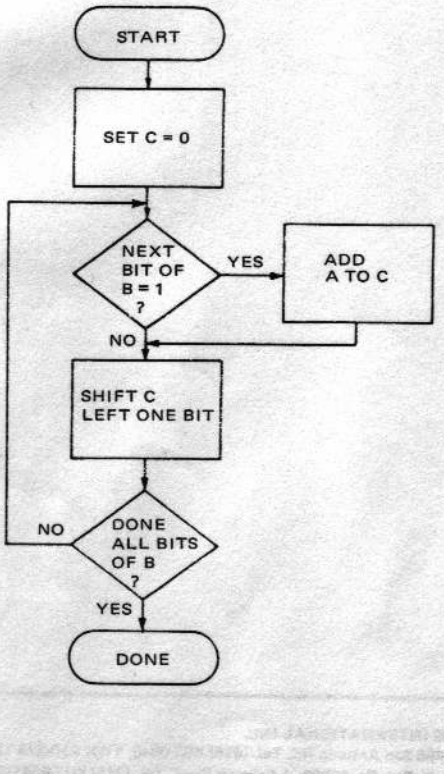


FIG. 4. FLOW CHART FOR MULTIPLICATION

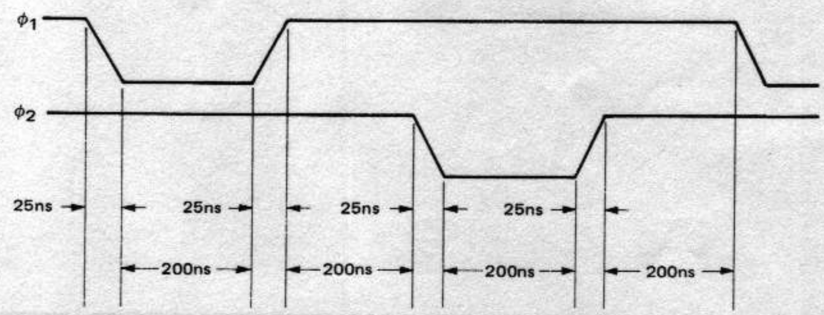


FIG. 5. EXAMPLE OF 900ns CLOCK CYCLE

4 . Continued . . .

SINGLE-CYCLE EXECUTE (2.7μs OR 3c TOTAL TIME)

CLA	IACSF	STA
CLARL	IDP	STF
CLARR	LAD	STFRL
CLF	LAM	STFRR
CLFRL	LAR	RAL
CLFRR	NOP2	RAR
COM	SAD	XPD
IAC	SAM	
IACCF	SAR	

TWO-CYCLE EXECUTE (3.6μs OR 4c TOTAL TIME)

ADD	ISZR
ADM	LAI
ADR	NAD
ISZD	NAM
ISZM	NAR

THREE-CYCLE EXECUTE (4.5μs OR 5c TOTAL TIME)

DDP	JLN	LDI
JF	JMP	LDID
JFL	JN	LDR
JFLN	JT	NPO4
JFN	JTG	SDR
JG	JTGZ	SIDR
JGZ	JTZ	
JL	JZ	

GLOSSARY

AC	accumulator (4 bits)
BYTE	8 bits
c	clock cycle
CPU	central processing unit
DP	data pointer (12. bits)
IF	instruction fetch indicator
I/O	input/output
I/P	input
K	1024.
LSI	large scale integration
NIBBLE	4 bits
OF	overflow register (1 bit)
O/P	output
PC	program counter (12. bits)
pROM	programmable read only memory
RAM	random (read/write) access memory
ROM	read only memory
R/W	read/write (data in, data out) indicator

head office
800 Dorchester Boulevard West, Montreal 101, Canada. Tel. (514) 875-2814

research, development and manufacturing centre
75 Moodie Drive, Ottawa Canada. Tel. (613) 828-9191
Box 3529, Station C, Ottawa, Canada K1Y 4J1, (Mailing Address)

marketing offices
CANADA:
Montreal, Quebec: 800 Dorchester Blvd., West. Tel. (514) 875-2814, TWX 610-421-4647
Ottawa 3, Ontario K1Y 4J1: Box 3529 Sta. C. Tel. (613) 828-9191, TWX 610-562-1910
Toronto, Ontario: P.O. Box 247, Port Credit. Tel. (416) 279-1358

U.S.A.: MICROSYSTEMS INTERNATIONAL INC.
Palo Alto, Calif. 94306: 450 San Antonio Rd. Tel. (415) 493-0848, TWX 910-373-1281
Philadelphia, Huntingdon Valley, Pa. 19006: 1 Fairway Plaza. Tel. (215) WI7-5641/2

EUROPE:
B–1050 Brussels, Belgium: 16 Avenue de la Toison d'Or. Tel. (02) 13.74.65, Telex 24836

UNITED KINGDOM AND SCANDINAVIA:
London W.1. England: 1 Great Cumberland Place. Tel. (01) 402-5521, Telex 261211
(a company incorporated in Canada with limited liability)

GERMANY: MICROSYSTEMS INTERNATIONAL GmbH
D–7000 Stuttgart 1 (W): Gustav Sieglestrasse 96. Tel. (0711) 65 31 16/26, Telex 722290

MICROSYSTEMS INTERNATIONAL LIMITED, BOX 3529 STATION C, OTTAWA, CANADA K1Y 4J1