SLAC-PUB-248
October 1966

A Concise On Line Control System*

by

Sam Howry

Stanford Linear Accelerator Center
Stanford University, Stanford, California

I. C-670418-1

*Work supported by U.S. Atomic Energy Commission.

(Submitted to 1967 Spring Joint Computer Conference, Atlantic City, New Jersey, April 1967)

Introduction

The introduction of hardware multilevel interrupts has made the small computer a suitable device for on line analysis and control of experiments in physics[1], and such systems have been developed in the past.[2] However, until recently little attention has been given to the man-machine interface, that part of the system referred to as the command post.[3] This paper describes an operational system at the Stanford Linear Accelerator Center (SLAC) where this area has been more fully explored.

A small on-line computer system has been developed at SLAC to monitor and control magnets and other equipment leading from the end of the accelerator to the experimental areas. The compact user language, easily grasped by the operators of the control room, provides a high degree of interaction with the environment. Internal computer time has been correlated with time of day so that the user has the capability in the language to synchronize events to a resolution of 1/360 sec over a 6 hour time interval. All input output including signals to and responses from the environment are completely multiprocessed, and two users may gain access on a time sharing basis.

Machine

The computer is an SDS 925, a 24 bit machine with a 1.75 μsec memory cycle time, and 4K memory. There is a typewriter (teletype Model 35), card reader and punch, all on a single channel, and 16 levels of priority interrupts.[6]

System Configuration

The system is shown schematically in Fig. 1. At present there are 25 A/D channels, (rate - 1 per 200 msec), which read magnet currents. Another input is the direct magnetic field (flip coil) measurement of the bending magnets leading to experimental area A. Outputs include 24 D/A channels (rate - 1 per 5 msec.) These provide the magnet power supplies with a reference voltage. For some magnets, the computer produces a number, which is counted down by a stepping motor (SLOSYN),

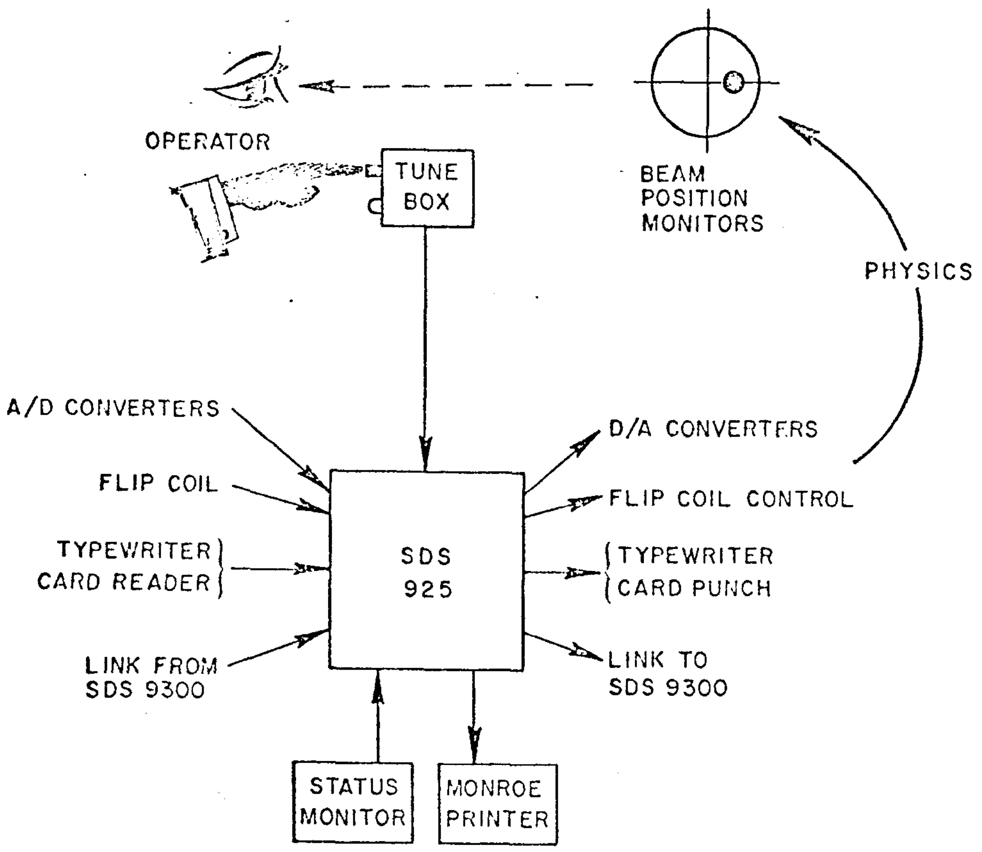
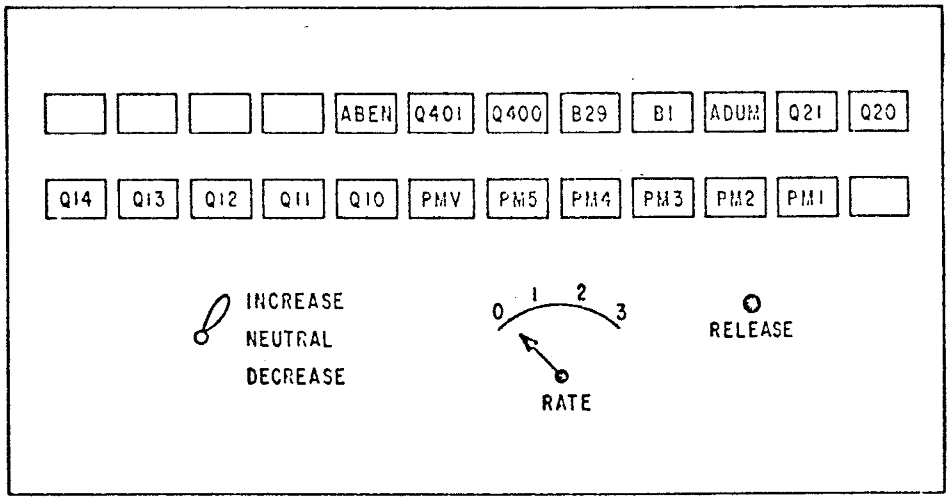


FIG. 1 -- BEAM SWITCHYARD COMPUTER SYSTEM

632-1-A



632-2-A

FIG. 2 -- FRONT PANEL OF TUNE BOX

each step creating a fixed current change.

The tune box allows the operator to make a fine adjustment on magnet settings much as he would with a potentiometer. This box is shown in Fig. 2. Using the typewriter, he can specify the sensitivity of the lever ('TUNE ALL BY .01 ;' gives one percent of value sensitivity, see Table 1 below.). Then by pushing the desired magnet buttons and the lever, he can change the magnets independently or in concert. The RATE dial provides single step mode or 3 nominal rates of change.

The link with the experimental area A computer (SDS 9300) consists of two 24 bit buffers (one for each direction) along with two interrupt signals to each computer indicating change of state of the registers (full to empty or vice versa). The experimental area A computer system is described in reference [4].

The status monitor interface packs status signals (1 for fault condition 0 otherwise) into 63 16 bit words. These are read into the computer 360 times/sec and checked for change of state. Any such change is logged (with the time) on the Model 1040 Monroe Printer (a 20 line/sec, digits only, 12 char/line device). This independent activity is multiprocessed with the other functions, and takes about 50% of the CPU time.

User Language

The instructions available to the user are in Table 1 and their full description is given in reference [5]. The table is in a kind of Backus Normal Form, although the language is not formally expressible this way because of the existence of noise words (see below). Language input may come from the typewriter, cards, or over the link from experimental area A. The typewriter/card channel and the link may be operating simultaneously.

The entities considered in this discussion of the language are: characters, words, clauses, instructions, and blocks. Universal delimiting characters are blanks and carriage returns. These two are functionally equivalent and may be used interchangeably to format the input copy. Words are then strings which contain no delimiters. A clause is a group of consecutive words, one of which identifies the clause.

SUMMARY OF INSTRUCTIONS

TABLE 1

Repeatable Instructions - clauses may be added

```
STEP <elt> BY <no.> ;
STEP FLC ;
CARDS ;
OUTPUT <elt> ;
RECORD <elt> ;
SEND <string> ;
TIME ;
CLOCK ;
DATE <string> ;
```

Clauses

```
EVERY  <time>  )
AT     <time>  }  - time clauses
UNTIL  <time>  )
TO  <destination> - destination clause
<label>  :        - label clause
```

'Once only' instructions - clauses are ignored

```
KILL <label> ;
SET <elt> BY <no.> ;
TUNE <elt> BY <no.> ;
TUNE ALL BY <no.> ;
```

<time> format examples

```
(9HR)           )
(9:00MIN)       }  refers to 9 o'clock, or 9 hours
(9:00:00SEC)    )
(T+5MIN)        }
(T+5:00SEC)     }  refers to current time plus 5 minutes
```

<destination> list

```
                          2 - Typewriter output
                          3 - Card output
                          4 - Link output
              anything else - typewriter output
      no destination clause - typewriter output
```

<string> definition - any sequence of characters, not spaces or carriage returns, the first of which is a letter or the character '*'

<label> definition - any string

<no.> definition - any sequence of digits with one decimal point somewhere in it. The sequence may be preceded by a minus sign '-'

<elt> list

PM1A	Q10	Q21	ABEN	COV	C1OV
PM2A	Q11	ADJM	FLC	C1H	T1OH
PM3A	Q12	B1	SL10	C1HR	T1OV
PM4A	Q13	B29	S1OR	C1V	SL30
PM5A	Q14	Q400	SL11	C1VR	S3OR
PMV	Q20	Q401	COH	C1OH	SL31
MF					

Thus

```
            L1  :  SET  B1  =  7.02
```

consists of the clauses

```
            L1  :
            SET B1
            =   7.02
```

The order of the clauses is irrelevent but within a clause position is important. Instructions consist of certain combinations of clauses. The above three clauses make up a legal instruction where the SET and = clauses are necessary but the other one is optional. However, the final nature of the instruction is not decided until the terminating semicolon word. On receiving this the computer searches backwards (i.e., from right to left) for a meaningful instruction. If there is none, the indication

```
                    ERR
```

is typed and the accumulated words are cleared out. If there are several, the computer takes the last one. For example, a bad typist might create

```
        TIME  L1  5  L1  :  SE  STEP  Q10  BY  .1  BY  .01 ;
```

and the computer picks out

```
        L1  :  STEP  Q10  BY  .01  ;
```

as the instruction to process. In this way special 'word delete' and 'line delete' characters are unnecessary. When educating users, the exact level of permissiveness of the language need not be spelled out as they discover this as they become more experienced.

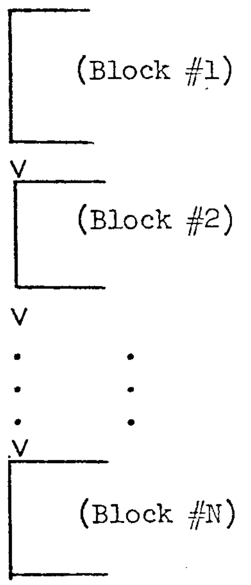
As Table 1 indicates there are several optional clauses which give the instructions flexibility. For example consider the two instructions

```
SETI  :  AT  (9:05MIN)  EVERY  (15SEC)  UNTIL  (9:20MIN) STEP B1 BY .05 PC ;
PUNI  :  AT  (9:05:10 SEC) EVERY (15SEC) UNTIL (9:20:10SEC) OUTPUT B1 to 3 ;
```

SETI causes the current in magnet B1 to be increased (by 5% of its value) every 15 seconds starting at 9:05 until 9:20. PUNI causes the value of the current in B1 to be punched on cards every 15 seconds but 10 seconds behind the execution of SETI, thus allowing the magnet to arrive at a steady state value.

Because of limited memory space there is at present no provision for cliches i.e. - tieing to a label a block of (possibly modifiable) instructions, thereby coining a new instruction. However, the same effect may be achieved by using card decks for memory as the next example shows. The following instructions, on cards, are placed in the reader.

```
LO  :  EVERY  (5MIN)  CARDS  ;
```



```
KILL  LO  ;  V
```

The user types 'CARDS' this shifts control from the typewriter to the card reader and the instructions LO plus those in block #1 are loaded. The 'V' returns control to the typewriter. At the end of 5 minutes LO is again executed and control passes to the reader for block #2. This continues until the last block, along with the 'KILL LO ; ' is read in. This last instruction terminates the activity.

The language may be expanded, (given additional memory space) by merely adding procedures which recognize new clauses (compile procedures) and corresponding ones which treat the new function at execution time (execute procedures).

Channel Scheduling

As mentioned earlier the typewriter/card input-output pass through a single data channel and in addition the typewriter input-output must share the same device. To overcome this difficulty the computer scans the status of all input-output 360 times a second using the following scheduling algorithm.

```
        if   {in middle of card read or punch cycle} then {continue}
else    if   {RED BUTTON HOLD} then begin
             if {channel clear} then {switch to TI}
             else if {channel not on TI} then {clear channel at end of its cycle}
             end
else    if   {in TO} then {continue}
else    if   {exists backlog of typewriter output} then {switch to TO}
else    if   {exists backlog of card punch output} then {switch to CO}
else    if   {input control is at card reader and it is ready}  then
                                                   {switch to CI}
else    if   {in TI} then {continue} else {switch to TI}
             where TI  =  typewriter input mode
                   TO  =  typewriter output mode
                   CI  =  card reader mode
                   CO  =  card punch mode
```

As a result, this is what the user sees: the typewriter light is normally on (TI mode). It occasionally goes off (with no warning, for about 1/2 sec) when a card is being punched (or read). When cards are being punched at the maximum rate it flickers. When there are large amounts of typewriter output it goes off steady. By pushing a red button (along side of the keyboard) the output is inhibited (although it still ques up internally), the light comes on, and the operator is free to type one instruction. The hold is released when the ';' word is received and the output continues. More may be loaded by pushing it down again, or by simply holding it down. Instructions like

```
        LO  :  EVERY ( 1 PUL) TIME  ;
```

very quickly swamp the output buffer and the means of recovery is to use the red button to insert

```
        KILL LO  ;
```

Finally, if the operator is keying in (without first pushing red button) and at that instant the computer decides to type, the input character may or may not be garbled. So to be safe (when the output subsides) he retypes the clause he was working on and continues.

Implementation

The implementation is characterized by extensive use of two relatively simple list structures. The structure shown in Fig. 3 is a circular, one-directional loop, with two markers A, B pointing to the last free node and last active node respectively. The number of nodes in a given loop is its length and is fixed.

Six such structures are used to completely multiprocess all of the input-output. For these buffers the pointers around the loop are implied by sequential locations in the machine. Advancing the markers amounts to adding 1 modulo n where n is the length of the loop. The nodes represent one computer word: a coded address-plus-value for the D/A channel, and BCD characters for the others. When information is to be output to the typewriter, for example the internal program dumps it into the loop using marker B. When the typewriter is ready it triggers an interrupt procedure which takes a character out using marker A. If there are no free nodes left, the data to be output is lost and the computer continues. When this happens the channel is "overloaded", but will recover automatically when the data burst subsides. The lengths of the individual buffers are chosen to handle reasonable data bursts through their respective channels without overloading.

A single such circular loop is used to dispatch the system commands at the precise time desired. In this case the nodes represent blocks of information: a pointer to next node, the entry of a particular procedure, its execution time, its frequency (if it is to be executed more than once), its parameter list, etc. Such blocks are compiled from source language instructions by background procedures and inserted among the active nodes, ordered as to increasing execution time. When a clock pulse interrupts the computer, current time is increased by 1. Then active nodes are lapsed into the free list (using marker A) until one is found whose execution time is greater than current time. As each node passes into the free list its procedure is executed. Since the nodes are ordered in time, no further interrogation is necessary at this time and the computer leaves the CLOCK PULSE interrupt level.

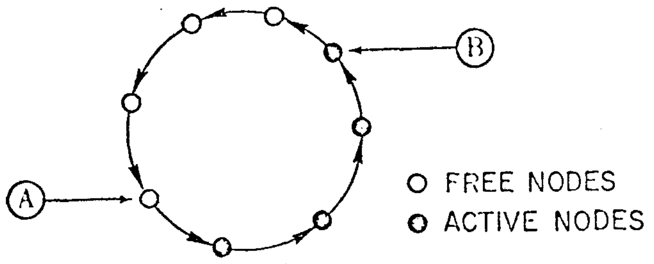


FIGURE 3 -- CIRCULAR LOOP STRUCTURE

632-3-A

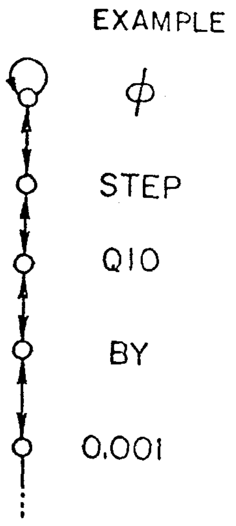


FIGURE 4 --WORD STACK STRUCTURE

632-4-A

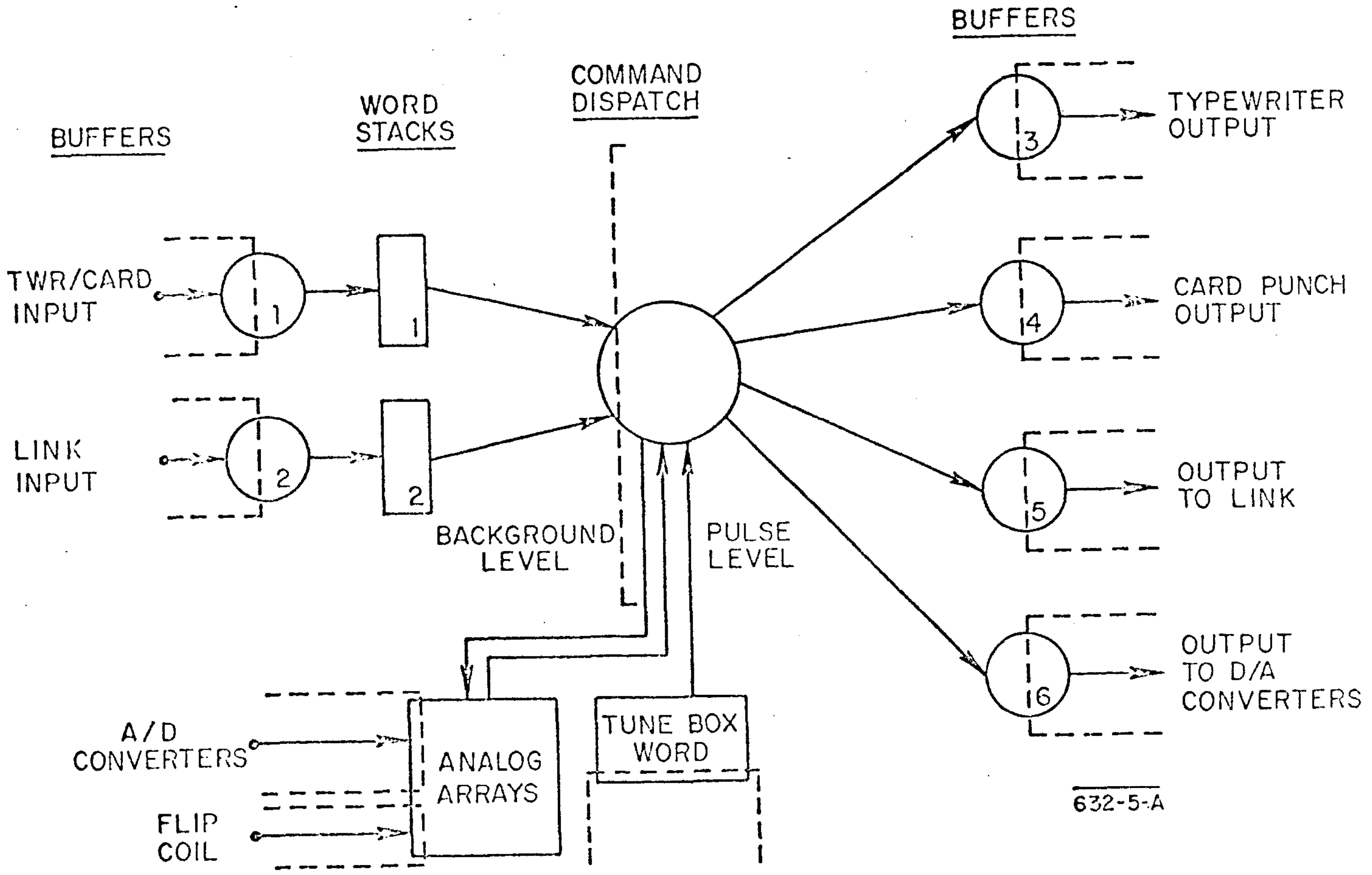


FIG. 5--PROGRAM BLOCK DIAGRAM

In this manner a 360 cps external pulse is the only hardware clock required to effect all the real time commands to the system. In the case of overload, i.e., no free nodes, the new command cannot be inserted and the computer ignores it and prints an indication.

The other data structure is that of the word stack. It is a linear bidirectional list with a variable number of nodes and a marker at the bottom. Each node represents a single input language word, as indicated in Fig. 4, containing such information as type, value, etc. In the case of a clause identifier such as STEP, it also contains the corresponding compile procedure entry location. The structure continues to grow downward as input language comes in until the terminating semicolon. Then the structure is complete and ready for various procedures to convert it into the block of information mentioned in the previous paragraph. There are two such word stacks, one for the card/typewriter input channel and one for the computer link from the SDS 9300. A single background program alternates character by character, between the two, building up the structures. When one is completed, no further building is done until the complete structure has been converted to a block and destroyed.

The program block diagram is shown in Fig. 5. The dashed brackets indicate different levels of interrupt with the higher priority on concave side. Note there are only two main levels of interrupt: PULSE (the 360 cps clock), and the background. The others merely take data into or out of buffers and return, freeing the channel. The tune box word contains the state of the magnet buttons, and is used by an execution procedure to address D/A values when called for. Continuously refreshed values in the analog arrays are available when needed by any of the execution procedures.

Example

This section contains an example showing how the system was used in its own development. The strength of the magnetic field in the large bending magnets leading to experimental area A determines the momentum of those electrons which pass through a narrow slit, and continue to ESA. (See Fig. 6).

Some experiments call for a very accurate estimate of this momentum. Others require that a given momentum achieved on Monday, say, can be accurately reproduced on Friday. The operator has at his disposal a current control, but because of hysteresis effects, current does not define the magnetic field (hence the beam momentum) uniquely. A slowly turning flip coil positioned in a reference magnet measures the field directly to 6 significant digits. When a single flip is completed the flip coil interface interrupts the computer with this number and, using calibration data the beam momentum is determined. The computer then can adjust the current to achieve the desired value.

The response of the bending magnets to several momentum change requests is shown in Fig. 7. Notice that when going from 7.020 to 6.990 GeV/C, the control algorithm sets it too low. By such tests, a satisfactory algorithm is arrived at empirically. The above calculation takes very little of the computer's time and other activities may go on simultaneously.

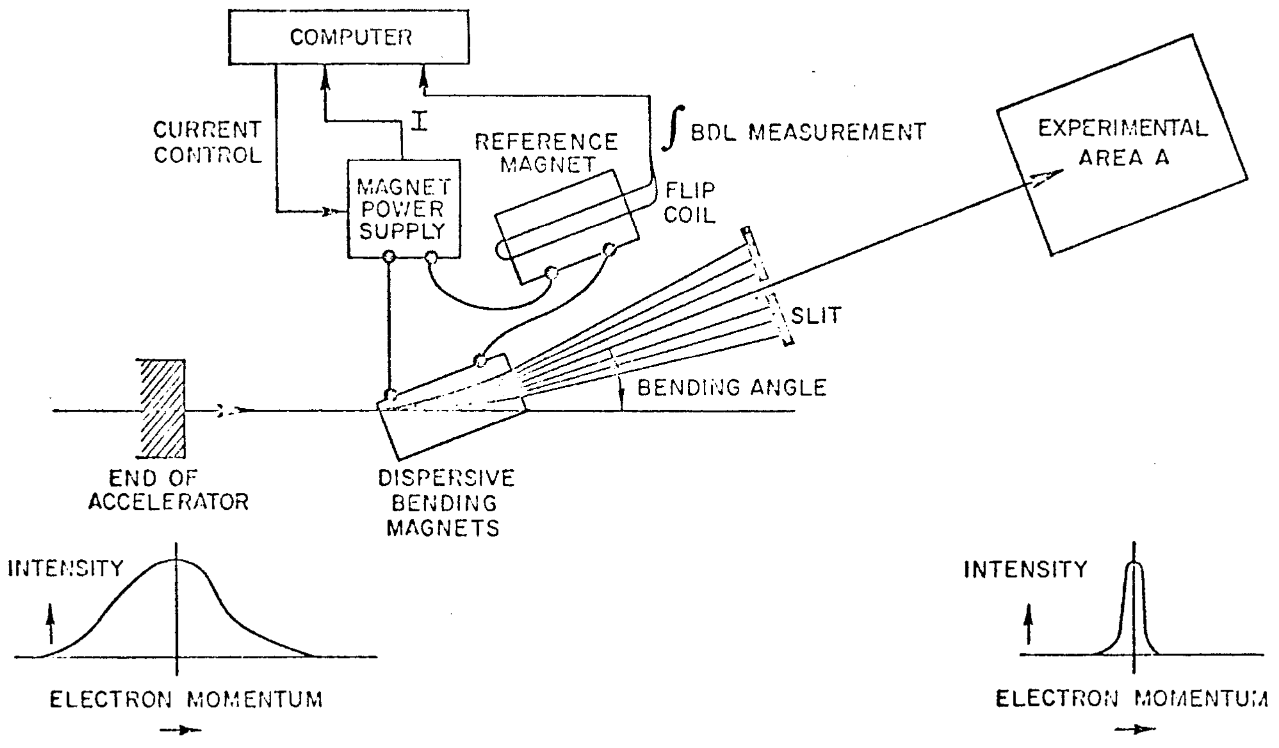
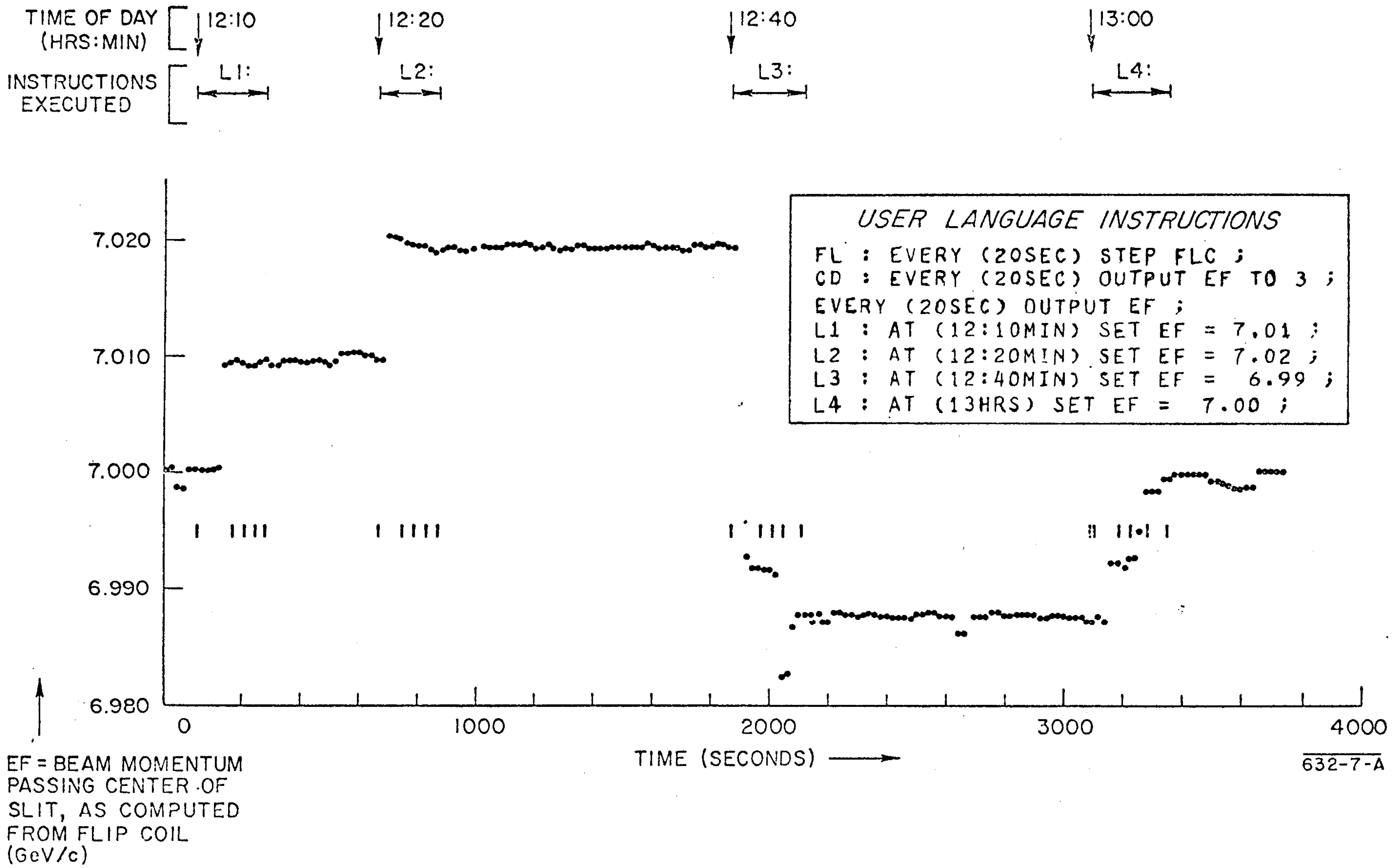


FIGURE 6 -- SCHEMATIC SHOWING METHOD OF MOMENTUM DETERMINATION.

632-6-A



USER LANGUAGE INSTRUCTIONS

```
FL : EVERY (20SEC) STEP FLC ;
CD : EVERY (20SEC) OUTPUT EF TO 3 ;
EVERY (20SEC) OUTPUT EF ;
L1 : AT (12:10MIN) SET EF = 7.01 ;
L2 : AT (12:20MIN) SET EF = 7.02 ;
L3 : AT (12:40MIN) SET EF =  6.99 ;
L4 : AT (13HRS) SET EF =  7.00 ;
```

FIG. 7 -- MAGNET RESPONSE TO SET INSTRUCTIONS

Acknowledgement

The hardware interface of the computer and related equipment is due to Dick Scholl, Ed Seppi, and Mike Hu, and their work, including many crucial user-engineer-programmer discussions is gratefully acknowledged.

References

[1] W. F. Miller, "The Role of Computers in Experimental Physics: A System for On-Line Analyzers", Proceedings Conference on Utilization of Multiparameter Analyzers in Nuclear Physics, November 1962. Columbia University, New York, CU (PN P2) - 227.

[2] Proceedings EANDC Conference on Automatic Acquisition and Reduction of Nuclear Data, July 13-16, 1964, Gesellschaft fur Kernforschung m.b.H. Karlsruhe, Germany, 1964.

[3] W. F. Miller, "Computation and Control in Complex Experiments", Proceedings IBM Scientific Computing Symposium on Man-Machine Communication, May 3-5, 1965, p. 113 ff.

[4] R. M. Brown, Mary Anne Fisherkeller, Antony E. Gromme, John V. Levy, "The SLAC High-Energy Spectrometer Data Processing System", (In Press IEEE Proceedings Computer Issue, December 1966)

[5] SLAC Report, CGTM 10, "BSY Control Computer System Language", August 1966. Author.

[6] SDS 925 Computer Reference Manual. SDS 900099A scientific data systems.