SLAC-PUB-2032
October 1977
(I)

SEMI-AUTONOMOUS CONTROLLER FOR DATA ACQUISITION
THE BRILLIANT ADC*

M. Breidenbach, E. Frank, J. Hall,† and D. Nelson

Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

ABSTRACT

A set of high speed 16-bit computers and ADC's has been designed and built for the data collection, compression, and correction system of the SLAC/LBL Mark II Magnetic Detector. The "Brilliant ADC" controls the analog multiplexing of a CAMAC crate of data acquisition modules, digitizes the analog data, and executes microprogrammed algorithms for data handling and correction.

Introduction

The SLAC/LBL Mark II Magnetic Detector is a large solenoidal detector for the storage rings SPEAR and PEP. The detector includes large arrays of drift chambers and liquid argon ionization detectors. The ~3200 drift chamber signals are processed by 32 channel time-to-amplitude converters (TAC's), and the ~4000 liquid argon signals are processed by 32-channel Sample and Hold Analog Modules (SHAM's).[1] The TAC's and SHAM's hold their analog information on FET isolated capacitors and, under control of the "Brilliant ADC" (BADC), multiplex their data onto the analog bus. A system block diagram is shown in Fig. 1. The BADC

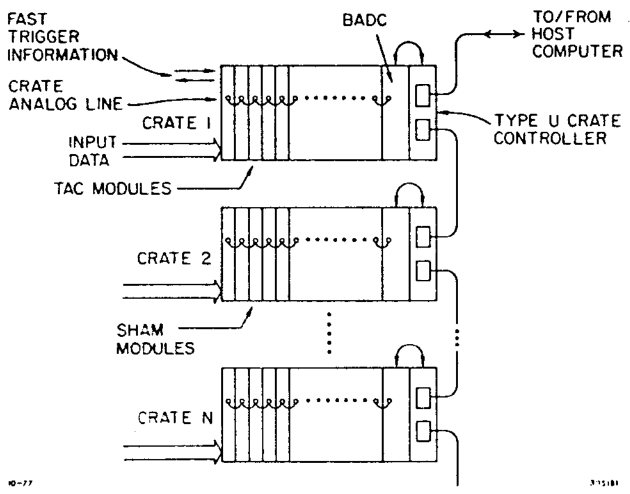

Fig. 1 System Block Diagram

digitizes the signals, and does data compression, correction, and formatting under the control of a 16 bit, high speed, microprogrammable processor. Each BADC controls one CAMAC crate of modules (608 channels), and will perform algorithms such as elimination of data below threshold and quadratic corrections to data above threshold in 3-10 ms. The Mark II will use about 16 BADC's.

System Considerations

The motivations for the BADC are primarily economy, reduction of event data acquisition time and host computer processing time, and simplification of the structure of the analysis program. The first point is achieved by allowing the BADC costs to be spread over 600 channels and by removing digitization hardware from the TAC's and SHAM's. The removal of this hardware allows a packaging density of 32 channels per single width CAMAC module without using hybridization, thus amortizing crate, controller, and BADC costs over 600 channels. The event data acquisition time is reduced because only channels having good data are transmitted, and the data is buffered so the host computer channel[2] can run at full speed. The hardware is configured so that an entire branch can be read by a single channel program not involving the host computer. Host computer processing time is reduced because all data correction that is independent of data from other channels (e.g., pre-amplifier gain for the SHAM's or cable lengths for the TAC's) is done by the BADC. Since the BADC is faster for half word arithmetic than the host machine, a time factor of more than the number of BADC's is saved. Program structure is improved because approximately 28,000 constants are removed from the analysis program, saving storage space and overlay swapping. Also, each data word enters with its functional label, e.g., a drift chamber layer and wire number, rather than an arbitrary channel number to be translated.

Architecture

A simplified block diagram of the BADC is shown in Fig. 2. The CPU is implemented using 4 AMD 2901 4-bit slices as an ALU and AMD 2909's for the microprogram sequencer.[3] The program is written in microcode (rather than in higher level instructions which are then interpreted by microcode) in order to maximize the execution speed. The microword is 48-bits wide, and the fields are shown in Table I. The microprogram is usually stored in a PROM 256 words long, but PROM's 512 words long can be used via a page control bit. The 16-bit immediate constant field is also used as an effective 9-bit branch address field; the immediate constant or address function being selected by another bit. A 3-bit branch condition field selects among no branch, unconditional branch, and branch on zero, non-zero, signed, not signed, overflow and carry flags. Another bit causes a pause of the CPU clock at the beginning of the cycle, so that an external device, such as the data memory or ADC, may acknowledge completion. This allows fetched data to the ALU to be used during this cycle. Three fields of three bits each control the ALU source, destination, and function. Two fields of four bits each control the A and B port selection of the ALU registers. A one bit field controls the least significant carry input to the ALU. A two bit field is used to control the SHIFT-ROTATE multiplexor. These two bits in combination with the ALU function control and one combination of the encoded microword field allow selection among left or right shifts with 0 or 1 fill, left or right rotation, arithmetic shifts, and shift with conditional arithmetic for a multiplication subroutine.

Two breakpoint "switches" are incorporated into the design for debugging and status checks. Break 1 is internal to the CPU and may be set and reset by the microcode. Break 2 is external, controlled by the ADC or by a front panel switch. Either breakpoint switch may be tested by a conditional branch instruction explicity controlled by 2 microcode bits.

Finally, 5 bits are encoded to control mutually exclusive operations. These include multiply control, push and pop operations on the sequencer stack, control of CAMAC Q and L lines, and control of "peripheral devices," e.g., the RAM and the ADC multiplexor section.

The RAM for the BADC is used for constant storage, a data buffer, and control table storage, but not as program storage.

*Work supported by Department of Energy

†Present address: Carlton, Oregon

(Presented at the IEEE 1977 Nuclear Science Symposium, San Francisco, Ca., October 19-21, 1977)

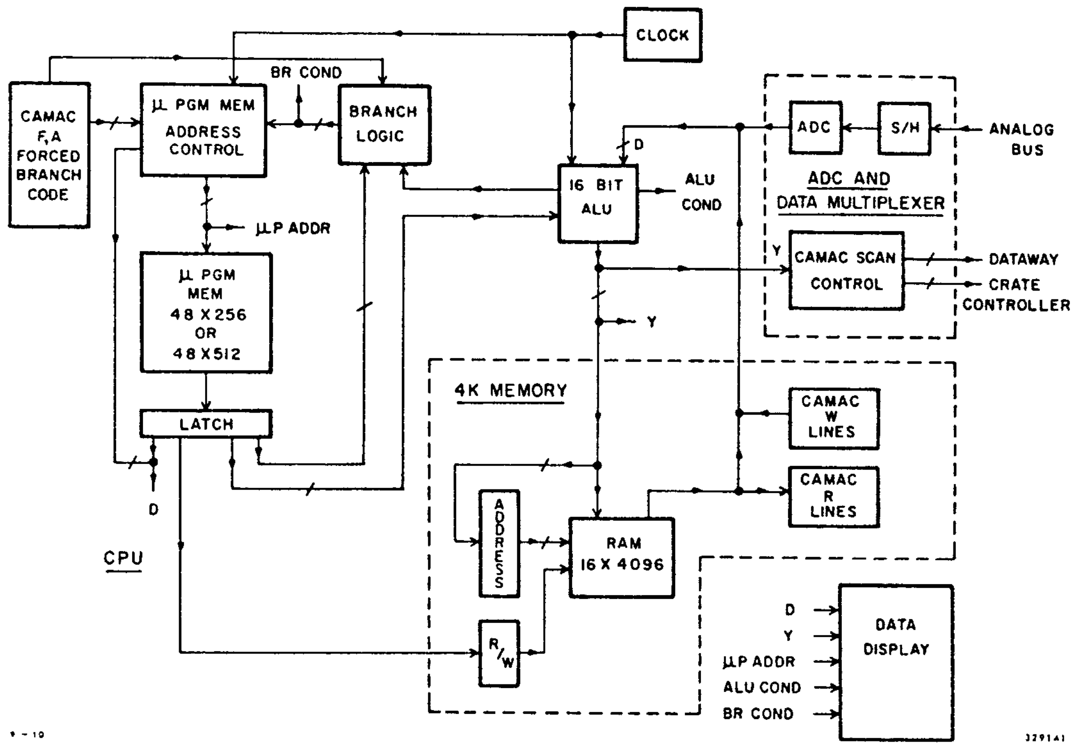

Fig. 2 Simplified block diagram of the BADC. The dotted lines enclose the ADC-MUX board and the memory board.

The memory is 4096 words by 16-bits per memory board. Normally only 1 memory board is used, but expansion to 32,768 words is possible. Memory read access time is 220ns.

CAMAC commands are executed by forcing branches to predefined locations in the microprogram PROM. Some CAMAC commands also directly affect the hardware, such as the F24-F26 LAM enable or the F9 reset instruction. A front panel NIM input also causes a forced branch so that the BADC operation may be initiated by the trigger logic without involving CAMAC. The BADC does not support interrupts (i.e., it is not possible to return from a forced branch). The BADC data interface to the R and W lines is by a pair of 16-bit buffers. The CAMAC R line buffer is loaded by every RAM read access, and is gated onto the R lines by valid CAMAC read functions. The W line buffer is loaded from CAMAC by any valid CAMAC write function, and this buffer is enabled onto the ALU input bus by one of the encoded microcode instructions. The W data may be enabled onto the ALU input and the ALU output written to RAM in a single instruction.

The CPU clock has 3 modes: short, long, and pause. The short cycle is used for all instructions not involving conditional branches or external devices and is 200 ns. The long cycle is used for conditional branches and is 360 ns. The pause cycle is arbitrarily long and waits for external device acknowledgment.

The BADC takes over control of the CAMAC crate via the SLAC Type U^4 crate controller. Autonomous control of the crate at the moment is a nonstandard aspect of CAMAC. The protocol chosen here is essentially that the host computer shall not address the crate while the BADC is in control. If the crate is addressed at this time, this condition is latched by the BADC and it can take appropriate error exits under software control. The rear cable connection between the BADC and crate controller includes encoded N lines, a controller enable signal from the BADC, and a controller active signal from the branch to the BADC. The cable assembly also includes the rather trivial LAM grading for the BADC. BADC control of the CAMAC function, subaddress, and strobe lines is by pulling these open collector lines down at the dataway. The TAC's and SHAM's have nearly identical control and analog multiplexing sections, and use F1 as a high order address line, along with the subaddress lines, to select among the 32-channels. The TAC's and SHAM's latch the address data with S1, which is generated by the BADC. The BADC CPU can set the starting address of this multiplexor control, and it can increment the address as part of an ADC read instruction.

The analog section is a pipeline, going from the modules to a sample and hold and then to an ADC. Thus while the CPU is analyzing channel i, the ADC can be digitizing channel i + 1, and the modules can be setting up channel i + 2. The sample and hold is a Burr-Brown SHM60 with a 1μs acquisition time for a 10 volt step. The ADC is the Datel EH12B3, a 12-bit, 2μs maximum conversion time device. The TAC's and SHAM's both operate with 0 to +5V outputs.

Construction

The BADC is constructed as a triple width CAMAC module, and is shown in Figs. 3 and 4. Both the CPU and memory boards are 4 layer printed circuits, while the ADC-MUX board has 2 layers. The front panel includes LED's indicating the PROM address, branch code, CPU condition flags, L, X, Q and clock indicators, and a set of 16 LED's that can be switched to show either the ALU input or output. Other controls are reset, single step, breakpoint, and external clock switches.

The design of the BADC is such that the ADC-MUX board is treated as an external device, so that its modification or replacement--for example, the addition of a buffer for the R lines to accept digital data from other modules--is rather easy.

Software

One of the problems associated with designing a computer with a new instruction set is that there are typically no programming aids available. Even such a simple tool as an assembler has to be written for each new instruction set. In the case of the BADC, this problem was overcome by writing an assembler-assembler (i.e., a program which generates an assembler for a given instruction set as its output) called

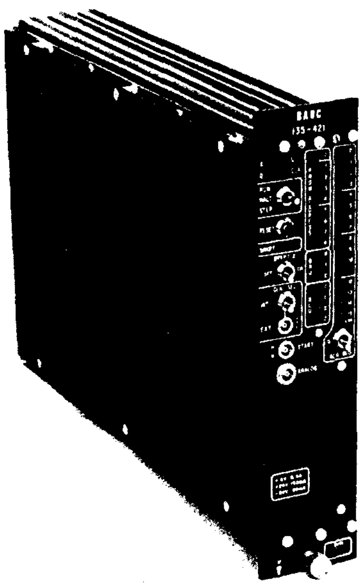

Fig. 3 Photograph of the BADC

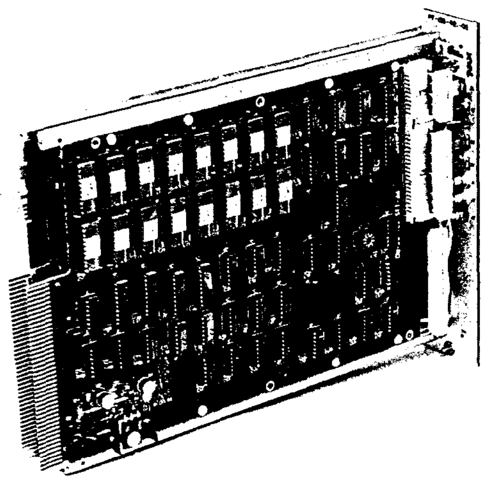

Fig. 4 Photograph of the disassembled BADC to show the memory board and the inter-board cabling.

MIMIC (Machine Independent Micro Code assembler). A MIMIC assembly consists of two phases. The first phase is used to define the machine in terms of its microcode fields and to define instructions to be used in the actual programming as operations on those fields. For example, the 3-bit field specifying the ALU opcode could be defined by the Define Machine Field instruction as:

ALUOP DMF 3

After the machine fields are defined, DMI (Define Machine Instruction) and DKW (Define Key Word) instructions are used to structure the actual assembly instructions. Fig. 5 is an actual listing of the DMF, DMI, and DKW instructions used for the BADC.

The second phase of a MIMIC assembly is similar to a normal assembly. The programmer codes using the instructions defined via DMI's in the first phase. The major contrast between MIMIC and normal assemblies is that several operations may be performed during a single instruction cycle, which is, of course, an implication of the use of microcode. The different (compatible) operations are coded on the same line, separated by commas. An example is shown in Fig. 6.

Software preparation for the BADC was done at the SLAC Triplex facility (two IBM 370/168's and a 360/91). The source code was written using the text editor WYLBUR. MIMIC actually consists of an extensive set of macros for the IBM H assembler, and a post-processor used to clean up the assembler listings and generate a simplified object module. The object modules can then be transmitted over a link to a CAMAC interface, and either loaded into a special debugging RAM or burned into PROM's. The debugging RAM is a separate CAMAC module that can be connected to the BADC to replace its PROM's, thus simplifying the debugging process. While MIMIC was initially written for the BADC, it has been used for other microcoded machines.

The first algorithm that has been developed for the BADC is a simple data correction and compression routine. Let Q_i be the ADC result for the ith channel. Then the data is discarded if $Q_i < \epsilon_i$; else $Q_i' = \alpha_i (Q_i - \delta_i) + \beta_i (Q_i - \delta_i)^2$ is stored in the RAM with the channel identification label in the adjacent word. The four constants are independently stored in the RAM for each of 608 channels. The two multiplications are done by a subroutine which takes approximately 3μs per 16-bit by 16-bit (32-bit product) multiplication. Data transfers between CAMAC and RAM are mediated by the program; CAMAC is slow enough that there is time for a short loop. The program also includes diagnostics for the BADC and for the other modules: for example, there is a mode in which any block of channels can be continually scanned so oscilloscope observations of the analog bus can be easily made. Finally, the BADC hardware allows a register directly loaded from CAMAC to be substituted for the ADC, thus allowing detailed comparison of the algorithm executed by the BADC with its simulation on the host machine.

Summary

Sixteen production models of the BADC have been built and tested. It is already quite clear that the programmability is a major advantage, since modifications and additions to the program are being actively pursued.

Acknowledgment

We would like to thank D. Porat for help in the design of the BADC, and also thank D. Scharre for help in the development of the software.

References

1. E. L. Cisneros, et al., IEEE Transactions on Nuclear Science, NS24 413 (1977).

2. We are presently using a CAMAC interface connected to the Multiplexor Input Output Processor (MIOP) of an XDS Sigma-5 computer.

3. The AMD2900 Family Data Book, (AMD Inc., California, 1976).

4. D. Horelick, IEEE Transactions on Nuclear Science, NS22 517 (1975).

```
+---> MIMIC - PASS 1 (V1.1) <--- CONTACT EHFSEA IF PROBLEMS
+
+
                MACRO
&LABEL                  EQU    &PARM
                        PUNCH '&LABEL     EQU     &PARM'
                MEND
BRADR           DMF     16,DEFAULT=0       REALLY ONLY 8 BITS OF ADDRESS
IMMCON          DMF     1,DEFAULT=0      ASSERT TO USE BRADR AS IMMEDIATE CON
BRCON           DMF     3,DEFAULT=0      BRANCH CONDITION CODES
DEST            DMF     3,DEFAULT=1
ALUOP           DMF     3,DEFAULT=0
SOURCE          DMF     3,DEFAULT=0
BREG            DMF     4,DEFAULT=0
AREG            DMF     4,DEFAULT=0
CARRY           DMF     1,DEFAULT=0
MSHFT           DMF     2,DEFAULT=0      MULTIPLY ROTATE SHIFT MUX CONTROL
READREQ         DMF     1,DEFAULT=0      ASSERT WHEN READ DATA TO BE GOOD THIS INS
BRK1REQ         DMF     1,DEFAULT=0
BRK2REQ         DMF     1,DEFAULT=0       FOR EXTERNAL FLAG
ENCODS          DMF     5,DEFAULT=0     ENCODED MICROCODE
*
*       BREAKPNT CODES
*
SETBRK          DKW     ALSOSET=((ENCODS,X'8'))
CLRBRK          DKW     ALSOSET=((ENCODS,X'9'))
REQBRK          DKW     ALSOSET=((BRK1REQ,1))
*
*
SETL            DKW   ALSOSET=((ENCODS,X'B'))
RSTL            DKW   ALSOSET=((ENCODS,X'C'))
RSTQ            DKW   ALSOSET=((ENCODS,X'A'))
*
WYM             DKW   ALSOSET=((ENCODS,X'12'))    WRITE Y TO MEM
WYA             DKW   ALSOSET=((ENCODS,X'13'))    WRITE Y TO (MEM) ADDRESS
WYMCWD          DKW   ALSOSET=((ENCODS,X'15'))    Y TO MEM AND CAM W TO D
CWD             DKW   ALSOSET=((ENCODS,X'14'))    CAMAC W TO D
WTD             DKW   ALSOSET=((ENCODS,X'13'))    WRITE TEST DATA
*                           WRITE MUX STARTING ADDRESS - NO INCREMENT
WMSA            DKW   ALSOSET=((ENCODS,X'1A'),(READREQ,1))
*                           WRITE MUX STARTING ADDRESS
WMSAI           DKW   ALSOSET=((ENCODS,X'19'),(READREQ,1))
*
RMD             DKW   ALSOSET=((ENCODS,X'11'),(READREQ,1))    READ MEM TO D
*
*                 ADC INSTRUCTIONS
*                       USE RAD TO PUT ADC DATA ON D LINES. READREQ IS IN FREE.
*                       USE RADERR=LOC    TO TAKE BRANCH TO LOC IN EVENT OF ERROR
*
RAD             DKW   ALSOSET=((ENCODS,X'18'),(READREQ,1))  READ ADC ETC
RADERR          DKW   POS=BRADR,ALSOSET=((BRK2REQ,1),(BRCON,UN))
*
*
RRQ             DKW   ALSOSET=((READREQ,1))       HOLD CLK SO DATA GOOD
FRQ             DKW   ALSOSET=((BRK2REQ,1))
*
ROT             DKW   ALSOSET=((MSHFT,2))
SHFT1           DKW   ALSOSET=((MSHFT,1))
ARI             DKW   ALSOSET=((MSHFT,3))
*
*               NULL OPERATION  DEST OF F
NOP             DKW   ALSOSET=((DEST,1))
*
*
JMP              DKW POS=((BRCON,UN),BRADR,ENCODS),SINGLE=BRADR
IMMED            DKW POS=BRADR,ALSOSET=((IMMCON,1))
ALU              DKW  POS=(AREG,BREG,ALUOP,SOURCE,DEST,CARRY)
*
*             MULTIPLICATION ALGORITHM
*
*             PUT MULTIPLICAND IN 'A' REGISTER
*             PUT MULTIPLIER IN 'Q' REGISTER AND DOWNSHIFT
*             CLEAR 'B' REGISTER, WHICH WILL RECEIVE MOST SIG PART
*                OF RESULT. LEAST SIG PART WILL WIND UP IN Q REG.
*             DO N-1 (15)    XSCA'S. MULTIPLY.SHIFT.COND ADD.
*             THEN DO A XSCS MULTIPLY,SHIFT,COND SUBTRACT.
*
*             XSCA TURNS ON :
*                                 1   S1,S2 FOR ARITHMETIC SHIFT
*                                 2   DEST=4
*                                 3   ALUOP=0
*                                 4   SOURCE=3 (MULT WILL TURN '1' ON AND OFF
*                                 5   CARRY=0
*                                 6   ENCODS=F    (MULTIPLY)
*
*             XSCS TURNS ON : 1   S1,S2 FOR ARITHMETIC SHIFT
*                                 2   DEST=4
*                                 3   ALUOP=1
*                                 4   SOURCE=3 (MULT WILL TURN '1' ON AND OFF
*                                 5   CARRY=1
*                                 6   ENCODS=F    (MULTIPLY)
*
XSCA               DKW POS=(AREG,BREG),ALSOSET=((MSHFT,3),                      *
                       (DEST,4),(ALUOP,0),(SOURCE,3),(ENCODS,X'F'))

*
XSCS               DKW POS=(AREG,BREG),ALSOSET=((MSHFT,3),                      *
                       (DEST,4),(ALUOP,1),(SOURCE,3),(CARRY,1),(ENCODS,X'F'))

*
UP              DMI   KEY=(IMMED,ALU,JMP,XSCA,XSCS,RADERR),PKW=(RSTL,           *
                        RSTQ,WYM,WYA,WYMCWD,CWD,WTD,WMSAI,WMSA,RMD,RAD,RRQ,FRQ,  *
                        NOP,SETBRK,CLRBRK,REQBRK,SETL,ROT,SHFT1,ARI)
*
            END
```

Fig. 5 DMF, DMI and DKW Instructions for the BADC

```
                                  *      ADC READ TEST ROUTINE
                                  *
                                  *      CAMAC INSTRUCTIONS
                                  *        F0A0  READS DIRECT ADC REGISTER
  LOC    OBJECT CODE              *        F16A0 WRITE EXTERNAL ADC REGISTER
                                  *
                                  CAMWAIT     OP   ALU=(R0,R0,RANDS,ZA,F),JMP=CAMWAIT
000000 000073200000
                                  F0A0        OP   ALU=(R0,R0,RORS,DZ,F3F),RAD,RADERR=DISPQC
000001 000A76F80068
000002 000172000000                           OP   JMP=F0A0
                                  F16A0       OP   ALU=(R1,R1,RANDS,ZQ,FBF),WMSA
000003 00000710889A
000004 022086F91000                           OP   ALU=(R2,R2,RORS,DZ,FBF),IMMED=X'220'
                                  RADC        OP   ALU=(R3,R3,RORS,DZ,FBF),RAD,RADERR=DISPQC
000005 000A76F99888
000006 000976188C20                           OP   ALU=(R1,R1,RPLUSS,ZB,FBF,CARRY),FRQ,JMP=DISPQ
000007 000513890800                           OP   ALU=(R1,R2,REXORS,AB,F),JMP=(NZ,RADC)
000008 000372099800                           OP   ALU=(R3,R3,RORS,ZB,F),JMP=F16A0
                                  *
                                  DISPQ       OP   ALU=(R0,R0,RANDS,ZQ,F),JMP=DISPQ
000009 000973100000
                                  DISPQC      OP   ALU=(R0,R0,RANDS,ZQ,F),JMP=DISPQC
00000A 000A73100000
                                  *
                                              END
```

Fig. 6 An example of MIMIC Phase 2 Code

Table I

BIT	NAME		DESCRIPTION
0		D0	These 16-bits connected to ALU input
1		D1	(D Lines) by μ code bit 10 (immediate const).
2		D2	
3		D3	
4		D4	
5		D5	
6		D6	
7	Page Bit	D7	This bit also used as prom page address.
8	BR0	D8	
9	BR1	D9	
A	BR2	DA	
B	BR3	DB	These 8-bits are the prom branch address during
C	BR4	DC	a branch instruction
D	BR5	DD	
E	BR6	DE	
F	BR7	DF	
10	CONSTANT		Selects bits 0-F as immediate constant.
11	BRCOND 2		
12	BRCOND 1		Conditional Branch Codes.
13	BRCOND 0		
14	I8		
15	I7		ALU Destination Codes.
16	I6		
17	I5		
18	I4		ALU Operation Codes.
19	I3		
1A	I2		
1B	I1		ALU Source Codes
1C	I0		
1D	B3		
1E	B2		B-Port ALU Register Address.
1F	B1		
20	B0		
21	A3		
22	A2		A-Port ALU Register Address.
23	A1		
24	A0		
25	ALUCARRY		Least Significant Carry Input to ALU.
26	SRS1		Shift-Rotate multiplexor Control.
27	SRS0		
28	Status Req		Causes execute pause at phase 0 of CPU clock.
29	BRKPT 1 REQ		Test status of BREAKPNT 1.
2A	BRKPT 2 REQ		Test status of BREAKPNT 2.
2B	EXT		Select external or internal encoded μ codes.
2C	OPCODE 0		
2D	OPCODE 1		Encoded bits
2E	OPCODE 2		
2F	OPCODE 3		