THE IERE-4BE CENTRAL PROCESSING UNIT

By B. Copeland* & E. Ackbarali**

ABSTRACT

This paper describes the IERE-4BE Central Processing Unit (CPU). The IERE-4BE CPU prototype was developed at the Department of Electrical and Computer Engineering of The University of the West Indies. It is the first CPU developed at this institution. It is a 4-bit CPU with a 12-bit address bus. The CPU was developed as an aid in teaching computer science as well as structured digital electronic design principles.

1.0 INTRODUCTION

In what follows, we present the design of the IERE-4BE Central Processing Unit. The IERE-4BE external bus lines include a 4-bit data bus, a 12-bit address bus and a single control line. It features three addressing modes and can execute 18 different instructions, seven of which are branch instructions.

The CPU is structured using the well-known Von Neumann architecture. Because it was designed for educational purposes, its structure has been intentionally kept simple. However, the instructions provided are more than what is required to be functionally complete in that a potential user has all that is necessary to solve any programming problem.

There were several motivating factors for embarking upon such a project. Apart from its already mentioned role in the teaching of relevant principles in CPU design, the IERE-4BE is also targeted as a sample device in the development of regional skills in structured logic system design. Moreover, it is felt that a project of this nature is essential to building student confidence in the design and fabrication of sophisticated electronic systems. This last point is all the more important in the Caribbean context since the region has no electronics industry, apart from the screwdriver consumer electronics industry. Moreover, it seems, at least to many financial and technology experts, that it will not develop one in the near future. In addition, largely due to the cost of components, there is relatively small activity at the hobbyist level. Yet in this age of intensifying global competition, all productive sectors of the region rely heavily on electronic equipment of varying complexity. In view of the above, the Department of Electrical and Computer Engineering has embarked upon a programme to enhance its strength in the area of electronics; this project represents its boldest attempt in this area to date.

The CPU design structure is discussed in Section 2. The detailed data unit and system architecture are treated in Sections 3 and 4 respectively. The Control Unit structure and micro-instruction format are then presented in Sections 5 and 6. Further details on the design and implementation of the processor can be found in [5].

2.0 DESIGN STRUCTURE

The IERE-4BE CPU design is based on the standard 2-level structure for large scale digital systems. This structure is described in any good logic design text (see [1] for example). The structure for this particular design is shown in **Figure 1**. The Data Unit (DU) contains all the necessary hardware to perform all CPU data processing functions; the required sequencing of DU operations is orchestrated by applying the appropriate signals to its control inputs. These signals are generated by the Control Unit (CU) based on the logic levels read on the DU status lines. CU control outputs include data strobe and output enable inputs to registers within the DU as well as mode controls to the DU Algorithmic and Logic Unit (ALU - *see next page*).

* *Senior Lecturer, Department of Electrical & Computer Engineering, The University of the West Indies (UWI)*
** *Development Engineer, Real Time Systems Group, Faculty of Engineering, UWI*

Pertinent discussion will be published in July 1997 West Indian Journal of Engineering if received by May, 1997.

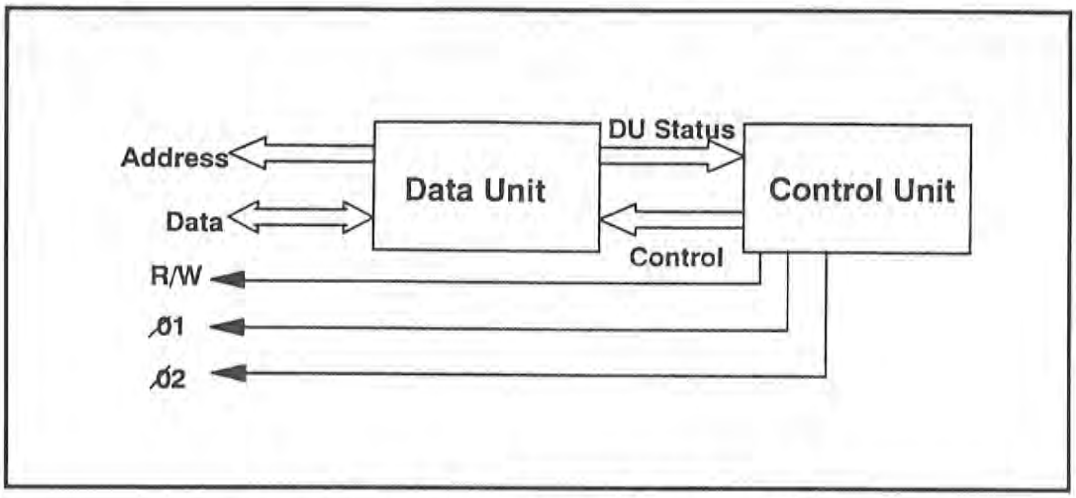


Figure 1: **Overall Design Structure of IERE-4BE CPU**

The CU can generate a sequence of control actions after reading a single status word. For example, instruction opcodes form part of the DU status. After reading an opcode for an arithmetic left shift, the CU outputs sequences which latch and output enable the appropriate DU registers to effect the shift operation. Upon completion of this operation, another sequence is immediately generated to direct the DU to load a new opcode byte from memory.

As is usual in this type of structure, a two-phase clock is used to ensure that DU and CU registers are not clocked simultaneously thus possibly violating register setup and hold time requirements. In addition, the system uses a two-phase non-overlapping clock ($\phi 1$ and $\phi 2$) to maximise the number of register transfers per cycle thus increasing CPU execution speed.

3.0 IERE-4BE DATA UNIT

Figure 2 shows the data processing structure of the IERE-4BE. Of the registers shown, only the A, CC and PC registers are visible to the programmer. All other registers are used to facilitate the operation of the CPU. The A register functions as the CPU 4-bit accumulator. The Condition Code register (CC) indicates whether the last instruction resulted in a zero result (Z=1) or generated a carry (C=1). As usual, the programme counter (PC) is used to sequence access to external memory. Whereas in CPUs such as the Intel 80 x 86 family memory addresses are determined based on calculations using the PC, IERE-4BE addresses are obtained directly from the PC. The small size of the CPU register file does make the programmer's task tedious but minimises hardware complexity. The functions of the remaining processor registers are shown on the next page.

As seen from the block diagram, most of these registers are interconnected by way of an internal 4-bit bus. Data can be moved from a source register to a destination register by simply enabling the source register to output data to the bus and latching the destination register to load this data after it has become stable. Some registers are more than 4 bits wide. For example, the IR consists of 2 4-bit registers IRH (for the high nybble) and IRL (for the low nybble). Likewise the MAR consists of 3 4-bit registers MARH (high nybble), MARM (middle nybble) and MARL (lower nybble).

With the exception of the PC and the ALU, all registers were implemented using the 74LS173 which is a quad flipflop with tristate output. The PC is provided with two control lines; one is used to load the PC from the MAR while the other is used to increment the current PC contents. The ALU was implemented using the 74LS181 which has 6 control lines and is therefore capable of 64 different operations.

In order to perform any task, the Data Unit registers are used to transfer data to and from external memory. Information is routed to the A and TR registers before the ALU is configured to perform the desired operation. Results are then transferred from the R register to the accumulator. The results can then be transferred back to external memory.

4.0 IERE-4BE ARCHITECTURE

The IERE-4BE has an external 4-bit data bus and 12-bit address bus. There are three categories of data which must be obtained from external memory. First, there is instruction data (operands); these are always 1 nybble wide. Secondly, there are CPU opcodes which are always 1 byte wide - the IERE-4BE can therefore facilitate 256 different instructions. Finally, there are addresses which are always 3 nybbles wide - the IERE-4BE therefore has a 4K x 4 physical address space.

FUNCTIONS OF PROCESSOR REGISTERS

A	This is the CPU accumulator. It is also used to hold one of the ALU arguments.
TR	Temporary storage register. It is used to hold the second argument for ALU operations.
R	Results register. This register is used to latch the results of all ALU operations.
CC	Condition Code register. Stores the status of the last ALU operation.
IR	The Instruction Register for storage of instruction opcodes for processing by the Control Unit.
MAR	Memory Address Register. A 12-bit register for buffering address information for indirect accesses to memory and for branch instructions.
ADLA	Address Latch. This register drives the external address bus.
PC	The Programme Counter. Automatically incremented to point to the next instruction in memory.
DRIN	Data Register In. Inputs data from the external 4-bit data bus.
DROUT	Data Register Out for holding data to be outputted to the external data bus.

The IERE-4BE supports three addressing modes: inherent, immediate and indirect. In inherent mode, the data for the relevant instruction is contained in the instruction itself. Thus, ASLA (Arithmetic Shift Left A-register through the Carry bit) operates on data already contained in register A. In immediate mode, the data is contained within the instruction itself and follows the opcode in the programme sequence in external memory. Thus, the assembler code

```
LDA   #$F
```

causes the A register to be loaded with the hexadecimal number F. The machine code is [0 1 F]. (See **Table 1** on page 11). In the indirect addressing mode, the instruction contains the address of the data. Thus, the assembler code

```
LDA   $020
```

causes data at location $020 to be transferred to the A-register. The machine code is [8 1 0 2 0].

The complete instruction set is shown in **Table 1**; note that a "*" in the C/Z columns indicates that the respective flag is affected and that blank entries under the "Addressing Modes" column indicates that the instruction does not operate under the relevant mode. Inherent and immediate mode instructions are all 3 nybbles long. In both cases, the first two nybbles contain the opcode while the last nybble contains the operand. The last nybble is irrelevant in the inherent mode and can therefore assume any value; it was included in the instruction format of the current implementation for simplicity in Control Unit design. In indirect mode, the instruction width is always 5 nybbles; again the first two nybbles are the instruction opcode while the last three nybbles are for the operand address.

The most significant bit of the first nybble indicates the addressing mode: if it is a 1, then the indirect addressing mode is in effect, otherwise the mode is either immediate or inherent (the actual machine code determines which).

5.0 IERE-4BE CONTROL UNIT

The IERE-4BE Control Unit employs a micro-programmed EPROM (the Control Store) as its central element; this makes it possible for a user to modify the instruction set. The contents of the Control Store follows directly from an exhaustive specification of the

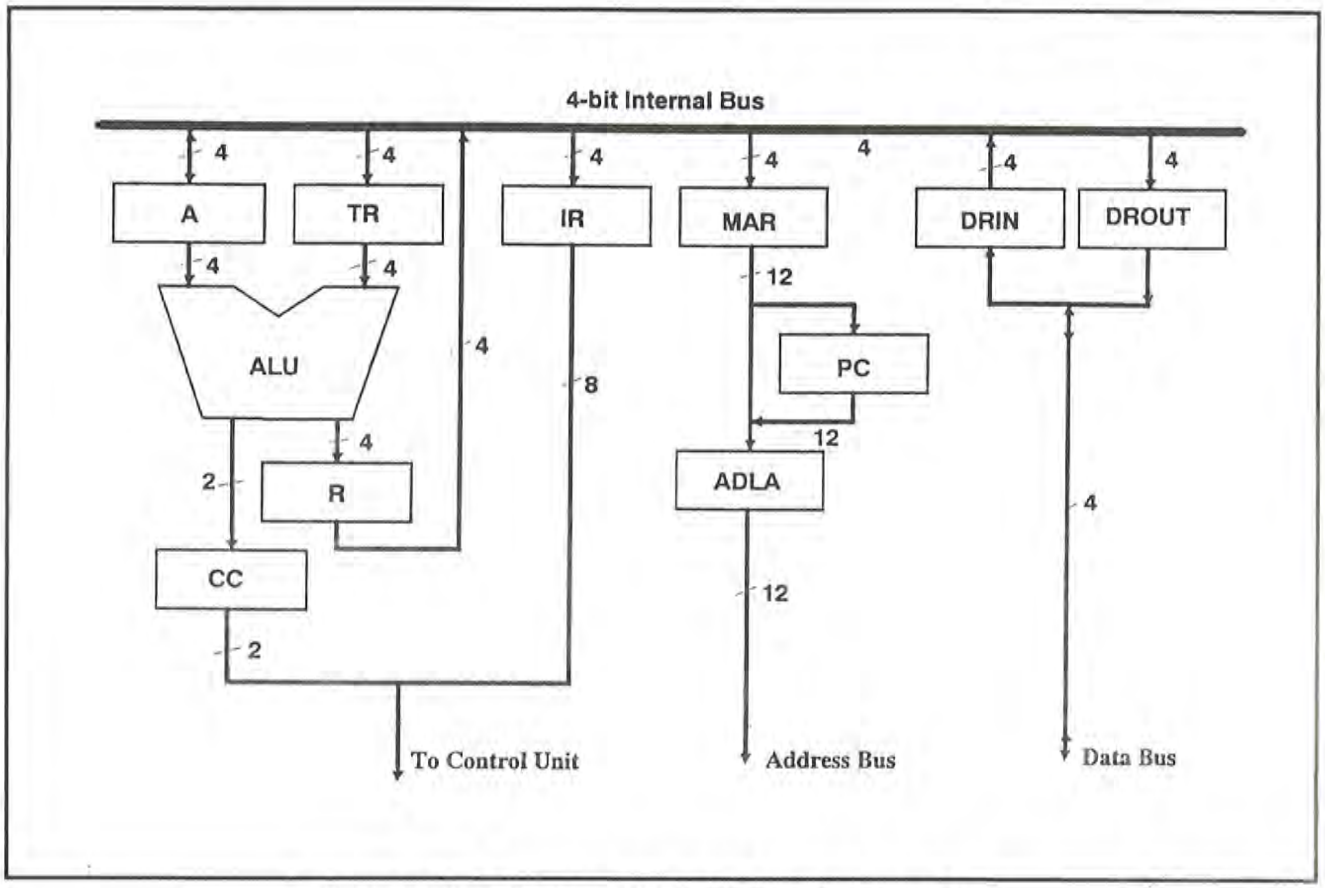


Figure 2: **Block Diagram of IERE-4BE Data Processing Unit**

signals required to control the registers of the Data Processing Unit. The Control Unit design uses four operational cycles:

1. Opcode Fetch Cycle: Two-nybble opcodes are fetched from external memory and loaded into the IR.

2. Direct Operand Fetch Cycle: A single nybble immediately following the opcode is fetched into the DRIN.

3. Indirect Operand Fetch Cycle: Data at the 12-bit address immediately following the opcode is fetched into the DRIN.

4. Execution Cycle: The instruction is executed.

An instruction cycle is always made up of three of these cycles. An Opcode Fetch and an Execution Cycle always respectively occur at the beginning and end of an instruction cycle. However, the third (intermediate) cycle can be either a Direct Operand Fetch; this depends upon the Addressing Mode in use. Register transfer operations to implement these cycles are shown in **Figure 3**. Transfer operations for the execution cycle depends on the actual instruction and are therefore not included.

Each of these transfer operations can easily be translated into a sequence of register control inputs using the usual methods. For example, a register transfer operation

R1 ← R2

translates to the following control sequence

ENABLE(R2), STROBE(R1)

which enables the output of R2 and subsequently asserts the load control of R1. Similarly, PC incrementation is performed by setting its register controls to the count mode and pulsing its clock input once (a single control line is used for this operation in the IERE-4BE).

The Control Unit has the structure shown in Figure 4: its inputs are the 2 status bits of the CC and the 8 data bits of the IR. Its outputs include individual register output enable and latch controls, as well as other specialised controls. The latter includes PC incrementation and ALU function selection.

The Control Unit consists of the following main elements:

* **Control Store:**
  This is EPROM containing the micro-instruction sequences for controlling the Data Unit.

* **Control Buffer Register (CBR):**
  Buffers the current micro-instruction address for the Control Store.

* **Control Address Register (CAR):**
  Provides the address of the next micro-instruction in the Control Store.

* **Control Store Vector Table (CSVT):**
  An EPROM containing vectors (base addresses) for each micro-instruction sequence in the Control Store.

* **Vector Table Pointer Register (VTPR):**
  This is a pointer into the Control Store Vector Table.

The system structure is not unlike that of the jump table assembly language construct with two levels of indirection: the CSVT contains start addresses of all micro-instruction sequences in the control store while the VTPR in turn points to the address of the addresses contained in the CSVT.

At the end of each execution sequence or on reset, the CAR and VTPR are cleared. Clearing the CAR points to the top of the Control Store EPROM. This corresponds to the start of the opcode fetch micro-instruction sequence.

At the end of the Opcode Fetch cycle, the CAR is loaded with new data from the CSVT. The address of this data is determined by IRO-7. IRO-6 are buffered by the VTPR. However, it may be recalled that for an indirect instruction IR7=1 while it is zero for a direct instruction. Since the VTPR was previously cleared, the CSVT address can either be 1000000_2 for indirect instructions or 00000000_2 for direct instructions. Once loaded, the CAR is again allowed to sequence through the relevant micro-instruction addresses.

	ADDRESSING MODES					
	Inherent	Immediate	Indirect			
Instruct.	Opcode	Opcode	Opcode	Description	C	Z
LDA		01	81	A← (M)	*	*
STA			82	(M)←A		
ADDA		03	83	A← (A) + (M)	*	*
SUBA		04	84	A← (A) - (M)	*	*
ASLA	05		-	C← A(3)← A(2)← A(1)← A(0)← 0	*	*
ASRA	20			A(3)→A(2)→A(1)→A(0)→C	*	*
CMPA		06	86	Test :(A) - (M)	*	*
ANDA		07	87	A←(A).(M)		*
ORA		08	88	A←(A)V(M)		*
NOTA	09			A←(A)'		*
BGT			8A	Branch If C =0 and Z=0		
BLT			8B	Branch If C=1 and Z=0		
BEQ			8C	Branch If Z=1		
BNE			8D	Branch If Z=0		
BGE			8E	Branch If C=0		
BLE			8F	Branch If C=1		
HALT	10					

Table 1: **IERE-4BE Instruction Set**

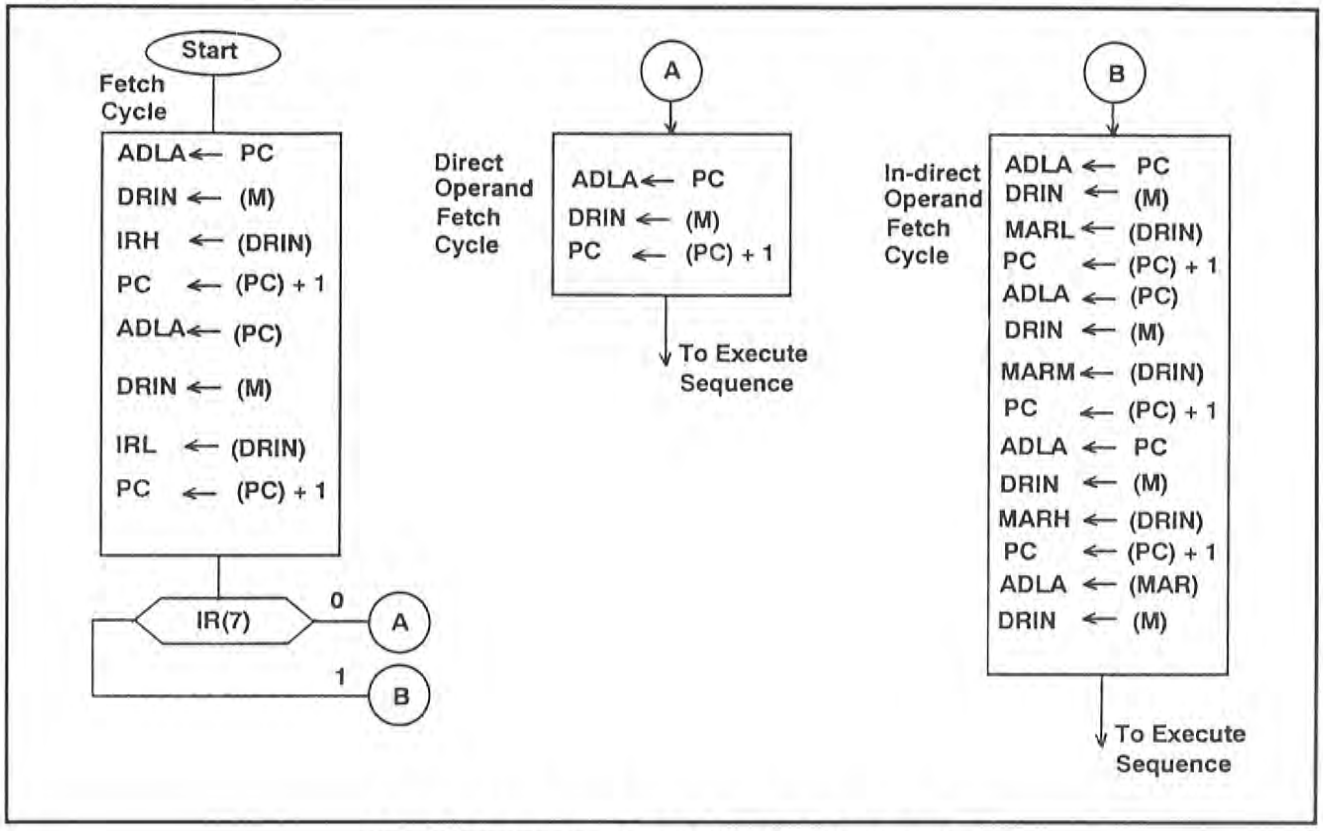


Figure 3: **Fetch, Direct and Indirect Fetch Operand**

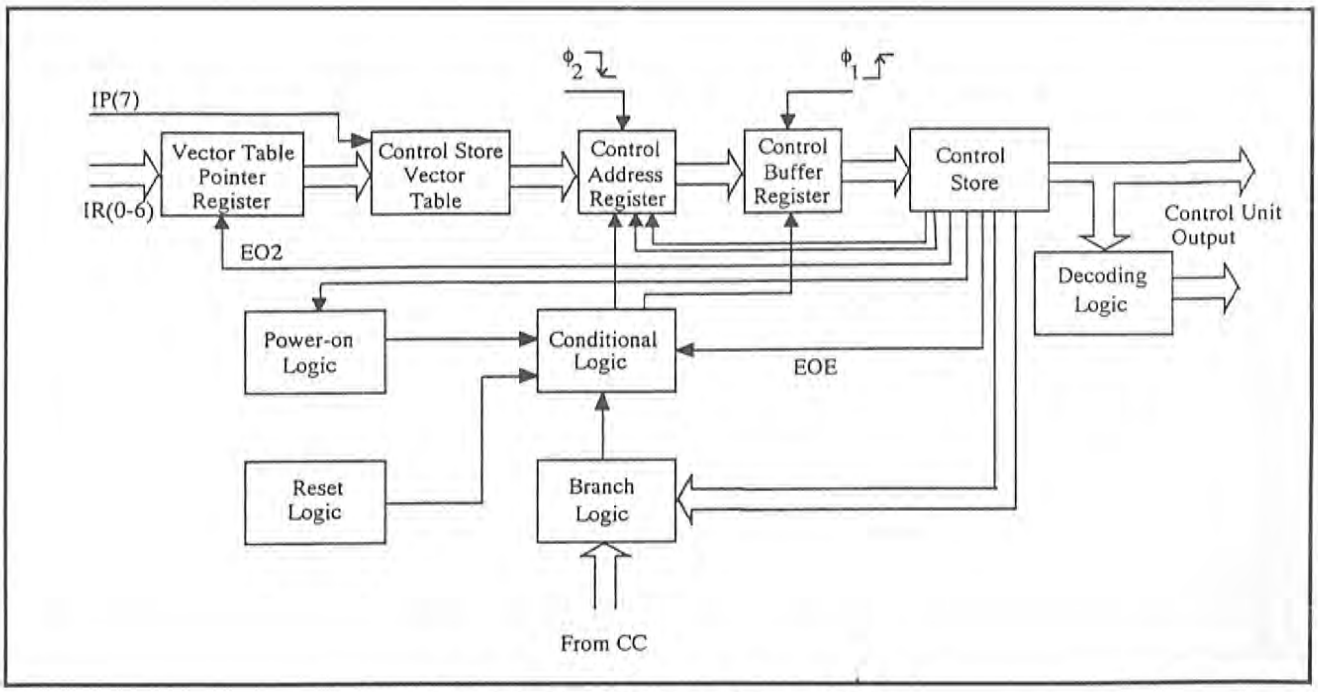


Figure 4: **Block Diagram of Control Unit**

At the end of each Operand Fetch cycle, IR(0-6) are transferred to the Vector Table Pointer Register to obtain the start addresses for the appropriate execution micro-code sequence in the CSVT. Again, the CSVT sequences through consecutive Control Store locations until the end of the execution sequence is encountered. The entire cycle is then repeated.

The unit is sequenced by a two-phase non-overlapping clock. The CAR is loaded only on the falling edge of $\phi 2$; data is latched into CBR on the rising edge of $\phi 1$. This allows the Control Store address to remain stable while the next micro-instruction address is being formed in the CAR.

The other logic elements of the Control Unit are used to force variations in the actions described in the paragraph above. For example, branch conditions are evaluated using the branch logic. If branch conditions are evaluated to be false, the processor is forced to execute the next instruction, otherwise the next micro-instruction sequence is executed. These alternatives are effected as above, i.e., by either clearing the CAR or allowing its contents to be incremented.

6.0 MICRO-INSTRUCTION FORMAT

The use of a micro-programmed Control Unit greatly simplifies the required logic; it also has the advantage of flexibility since the microcode stored in EPROM can be easily modified.

The micro-instruction word of the IERE-4BE is 24 bits wide with format as shown in **Table 2**. The CPU employs a mix of horizontal and vertical micro-instruction formats [2]. The functions of each of the signals are also listed in the table. Most of the bits of the micro-instruction word is directly connected to a control line of an element in the Data Unit or in the Control Unit, thus implementing what is known as an horizontal micro-instruction format. However, bits 15 to 23 are encoded and therefore implement a vertical format. Encoding is employed to minimise bit width; it is applied to signals which will never be simultaneously asserted. This applies, for example, to the load and clear controls of a single register.

The micro-instruction sequences for the Fetch cycle are shown in **Table 3** as an example.

Each row describes the actions taken in a single clock cycle. The actions indicated on the first row are enabled by setting b5=0 for an address latch load and b18-20=110 for a data register load, b15-17=110 to output enable the PC. In the next cycle b21-23=000 to increment the PC, b18-20=001 to load the IRH and b4=0 to output enable the input data register. For reliability, these actions do not occur simultaneously and take place only on certain clock transitions; **Figure 5** shows how they relate to the clocks.

The Control Unit uses 2732 EPROMS for the Control Store and Control Store Vector Table. The CAR utilises two cascaded 74LS163 counters while the CBR and Vector Table Pointer are implemented using the 74LS273 octal latch. The other elements are realised using standard TTL combinational logic devices.

7.0 EDUCATIONAL ASPECTS

The IERE-4BE provides a hands-on platform for the teaching of topics in computer science and electronic logic design in a variety of ways:

1. Through easy access to all CPU elements, students can easily monitor the status of all internal registers and data as well as control paths; this is further facilitated by the fact that the CPU can operate at clock rates down to DC.

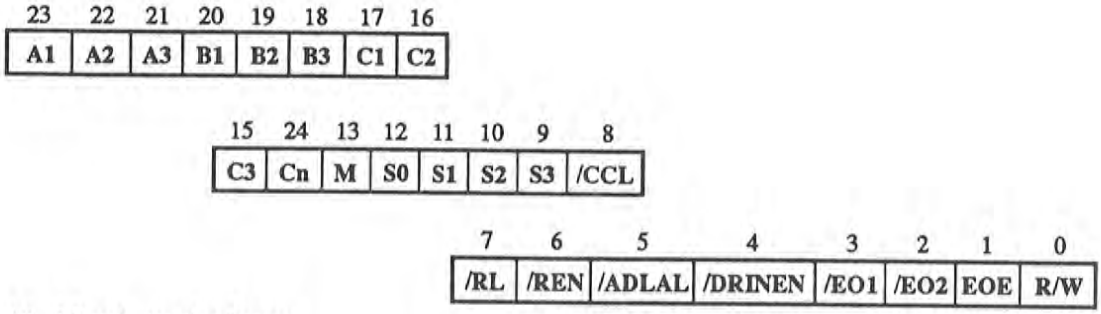
2. The micro-instruction format allows the student the flexibility of adding his own instructions through modification of the Control Store and CSVT EPROMs.

3. In addition, because of the open structure, students can fairly easily explore and evaluate other design options. For example,

   i) Students can modify the design to explore the effect of alternate clocking schemes on CPU performance.

   ii) Students can explore design improvements such as the addition of stack and interrupt facilities; in fact these aspects were successfully attempted in the 1995/96 academic year [7,8].

4. The development of hardware/software support systems can and have been explored. For example, a recent final year project focussed on the design of a cross assembler



(a) Microinstruction Format

Control Signal	Function
A1-3	Encoded Group A: PC increment control plus flags for each of the 6 branches
B1-3	Encoded Group B: Load controls on IR, MAR, DRIN. MAR enable control
C1-3	Encoded Group C: Load controls on A, PC, TR, DROUT. Enable on A, PC, DROUT
Cn, M, SO-3	ALU Control Signals
/CCL,/RL	Load controls on CC and R
/REN	Data output on R
/ADLAL	Address Latch load control
/DRINEN	DRIN enable
/EO1, /EO2	Loads new data into Primary Register and CAR respectively
EOE	End of Execute Cycle. Forces jump to operand fetch cycle
R/W	Memory Read/Write control

(b) Signal Description

Table 2: Microprogramming Signals

	Microcode Bit Number																							
Action	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADLA ← (PC), DRIN ← (M)	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0
IRL ← (DRIN), PC ← (PC)+1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0
ADL ← (PC), DRINV(M)	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0
IRH ← (DRIN), PCV(PC)+1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0
End of Fetch	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0

Table 3: Microinstruction Sequence for Fetch Cycle

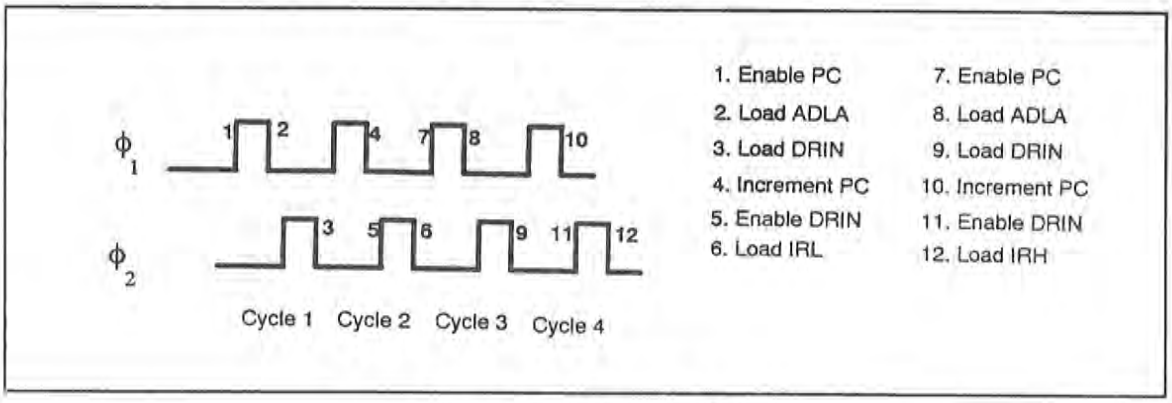


Figure 5: **Fetch Cycle Timing**

for the IERE-4BE [6] while another focussed on the design of a simple development board (including monitor software) for the CPU [7].

The open structure afforded by its implementation using standard logic components gives the CPU one distinct disadvantage. The process of making modifications to a complex design implemented in this way is prone to many disastrous hardware errors. Students can be frustrated by having to spend an excessive amount of time in debugging hardware problems.

Recently another final year project looked at the implementation of the CPU using Field Programmable Gate Arrays (FPGAs)[4,8]. FPGAs incorporate very dense programmable logic with judicious quantities of routing resources; current CAD tools allow designers to do all their design in software, completely test their design through simulation and then finally download to programme the FPGA device. In the case of the IERE-4BE, the entire design can fit on a single XC4005A chip [8]. Because FPGAs are reprogrammable, design changes, including changes in the microprogramme EPROM, can easily be made at the software level; little or no actual physical rewiring is required. This results in a tremendous decrease in the occurrence of fatal errors while minimising the delay between modification and actual implementation.

8.0 CONCLUSION

This paper has described the IERE-4BE CPU designed and fabricated at the Department of Electrical and Computer Engineering at The University of the West Indies. The IERE-4BE represents the University's first attempt at such a project. It was designed to enhance the Department's skills in structured logic design and logic implementation while being an educational tool for students. The current design has been realised using random TTL logic and standard EPROM devices. A breadboard implementation of the CPU was found to be capable of reliable operation at clock rates well exceeding 1MHz. While speed is not a major consideration in this design, it is expected to be considerably improved when the CPU is implemented on a printed circuit board.

REFERENCES

1. Breeding, W. *"Advanced Logic Design"*, Prentice Hall 1990.

2. Tanenbaum, A.S. *"Structured Computer Organisation"*, 3rd Ed, Prentice Hall, Englewood Cliffs, NJ. 1990.

3. Andrews, M. *"Principles of Firmware Engineering in Microprogram Control"*, Computer Science Press Inc., 1988.

4. *"The Programmable Logic Data Book"*, XILINX, 1994.

5. Ackbarali, E. *"Enhancements to the IERE-4B Central Processing Unit"*, Final Year Project Report, Department of Electrical and Computer Engineering, The University of the West Indies (UWI), St. Augustine, May 1994.

6. Turner, M. *"The Printed Circuit Board Implementation of the IERE-4B CPU Prototype and the IERE Assembler"*, Final Year Project Report, Department of Electrical and Computer Engineering, UWI, St. Augustine, May 1995.

7. Amin, N. *"The IBIS Educational Development System"*, Final Year Project Report, Department of Electrical and Computer Engineering, UWI, St. Augustine, May 1996.

8. Seegoolam, S. *"The FPGA Implementation of the IERE-4BE and Its Modification to the 8-bit IBIS"*, Final Year Project Report, Department of Electrical and Computer Engineering, UWI, St. Augustine, May 1996. ■