

Design of a 16-bit microprogrammable computer with bit-slice devices

In his concluding article **Dogan Ibrahim** describes the remaining subsystems of the computer

The first article described the basic design principles of the data and control section of a bit-slice computer. We shall now consider the design of the remaining subsystems of the computer.

The I/O subsystem provides the communication between the processor and the outside world¹. The computer accepts data from the input devices, processes them and the result is sent to the external devices via the output ports. There are three types of I/O according to the method of controlling the data transfer:

- program controlled I/O
- interrupt controlled I/O
- direct-memory-access I/O

The I/O ports can either be bidirectional, in which case the external data will be sent and received via the same lines, or the input and output lines can be separate.

The I/O subsystem that will be described is the program controlled I/O with separate input and output lines. The block diagram of the I/O is shown in Figure 1. The Am2917² bus transceivers interface the external I/O signals to the data bus of the computer under the control of the CCU. For a 16-bit I/O port, four Am2917s are required. The I/O port addresses can either be derived from the system address bus or a 16-bit port bus can be derived from the system data bus. The address of an I/O port is selected from a decoder (cascade as many as required). The I/O subsystem is controlled via the two control signals IOR (I/O request) and IOM (I/O mode).

PROGRAM CONTROL UNIT (PCU)

The function of the PCU is to address the next instruction to be fetched from the memory. The PCU consists basically of a program counter register, a stack and the associated logic. A block diagram of a typical PCU is shown in Figure 2. The address information to the PCU is derived from the data bus. The program counter register stores the current memory address and it is incremented after every macroinstruction. During program jump instructions, the program counter is loaded with the jump address and the

D Ibrahim, Department of Electrical & Electronic Engineering, The City University, London, UK

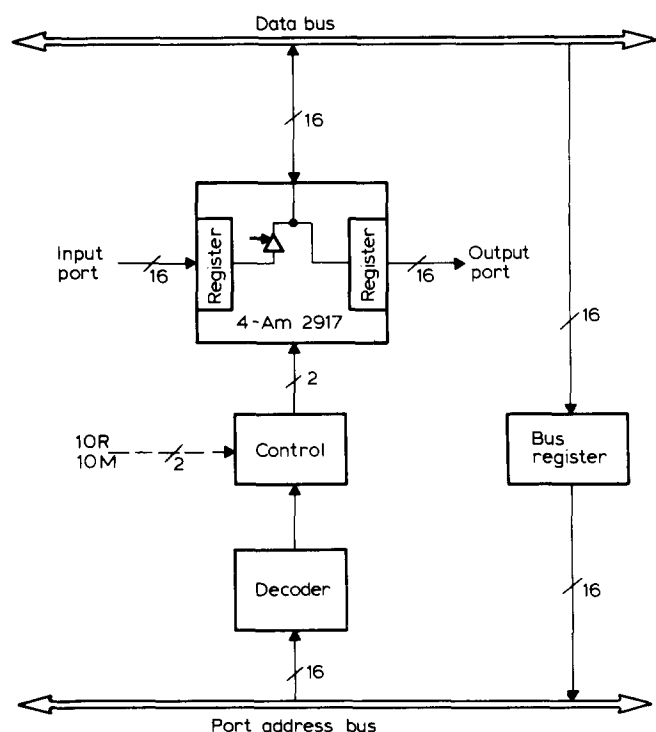


Figure 1. I/O subsystem designed with Am2917 bus transceivers

program execution continues from there. If a jump-to-subroutine instruction is executed, the program counter register and the stack pointer are incremented and the contents of the program counter are pushed onto the stack. On the next clock cycle the program counter register is loaded with the subroutine service address. During a return-from-subroutine, the stack pointer is decremented and the program execution will continue with the instruction after the jump-to-subroutine.

The PCU can be designed by using the LSI Am2911³ microprogram sequencers. The Am2911 is a four-bit sequencer containing an incrementer and a register (program counter). The device contains a multiplexer that selects the source of data at the Y outputs. Data can be selected from the data bus (via the D inputs), from the program counter register or from a 4 x 4

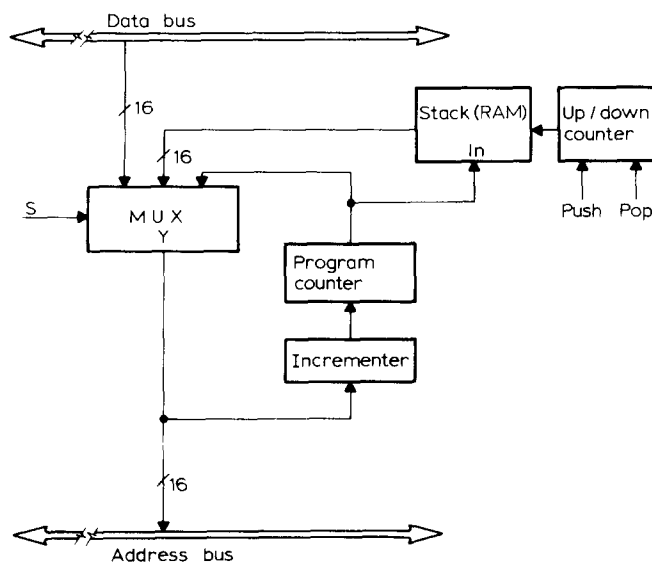


Figure 2. Block diagram of a typical PCU

last-in-first-out stack. For a 16-bit computer, four Am2911s should be cascaded by ripple carry connection mode. The block diagram of the PCU using the Am2911s is shown in Figure 3. The PCU is controlled by the CCU via the four control signals A_1 to A_4 . $\overline{CC}$ is the signal derived from the condition code select logic and it is used as a test for the conditional program executions.

INTERRUPT CONTROL UNIT

Interrupts are external asynchronous events that change the normal program flow in a computer. By interrupting the processor, the external logic asks the processor to stop whatever it is currently doing in order to service the needs of the external logic. Upon servicing the interrupt, the processor carries on with its suspended operations. With no interrupt capabilities, the processor depends upon the software polling techniques to service the peripheral devices. If the number of such devices is large then the polling program becomes very complicated and time consuming. Interrupts can be used to increase the system throughput by eliminating the need for software polling techniques.

During all interrupt handling mechanisms the following steps are executed by the processor:

- L2 Executing
 - receive interrupts
 - if interrupt accepted go to L1
 - go to L2
- L1 recognize the interrupting device
 - save the status bits
 - disable further interrupts
 - save the program counter contents
 - service the interrupt
 - restore the status bits
 - restore the program counter
 - go to L2

Depending upon the system requirements the interrupt control logic can be designed in several ways^{4,5}.

The ICU that will be described here is based upon the

Am2914 interrupt encoder⁶. The Am2914 is a bipolar interrupt control unit that accepts eight external interrupts (but can be expanded). The device is controlled by a 4-bit control field I_{0-3} enabling 16 different interrupt mechanisms.

The simplified block diagram of the Am2914 is shown in Figure 4. An 8-bit mask register is used to mask any interrupt. Each interrupt line has a priority level and during multiple interrupts, the Am2914 will only recognize the interrupt of the highest priority level (if it is not masked). The priority encoder generates a 3-bit vector corresponding to the input interrupt level. The status register is used to point to the lowest priority at which an interrupt will be accepted. The contents of the status register are compared with the 3-bit vector and an interrupt request will be generated if the vector is greater than or equal to status. The clear logic is used to individually clear the interrupt latches.

The block diagram of the interrupt control unit is shown in Figure 5. The ICU is controlled from the CCU. The bidirectional mask and status bits are connected to the system data bus so that they can be loaded or read by controlling the Am2914. The 3-bit vector is decoded in a PROM and the 16-bit address information is presented to the data bus. The PROM is programmed appropriately so

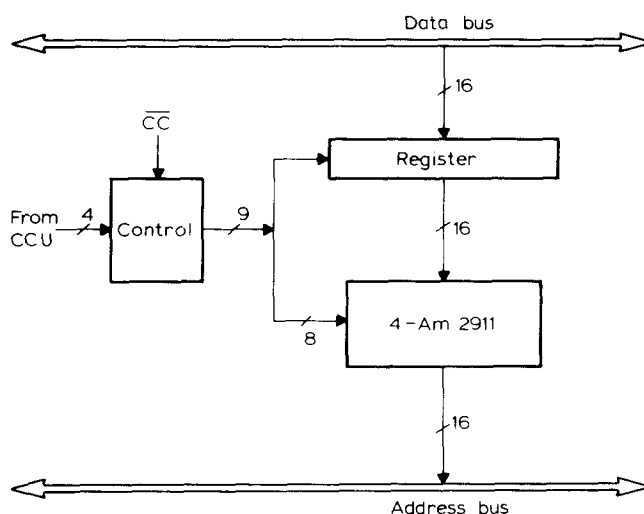


Figure 3. PCU designed with Am2911 microprogram sequencers

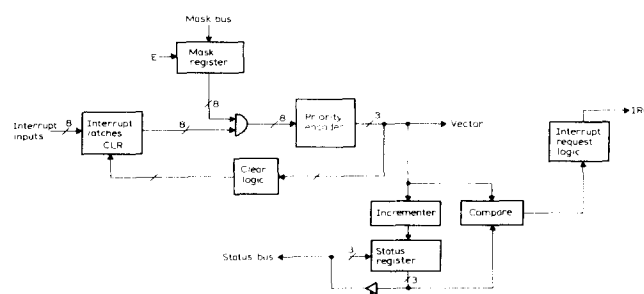


Figure 4. Simplified block diagram of Am2914 priority encoder

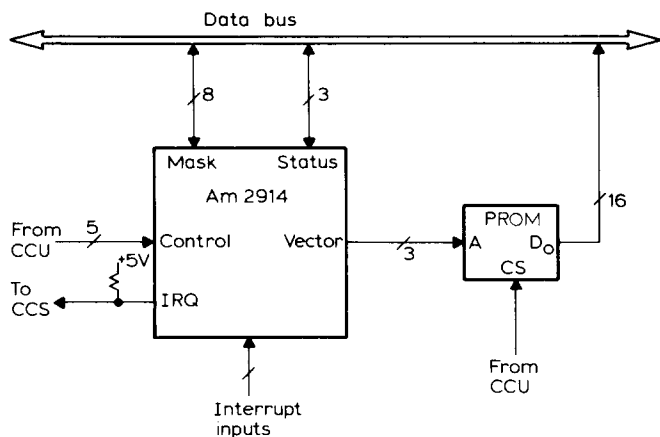


Figure 5. ICU designed with Am2914 priority encoder

that each vector is decoded to a different interrupt service address in the main memory (this is accomplished by loading the program counter of PCU via the data bus).

The interrupt request line of Am2914 is presented to the condition code select logic (CCS) of the APU. Since the interrupts will only be accepted during the instruction fetch cycle, then a test must be made before the fetch cycle to see if any interrupt has occurred. If an interrupt has occurred then the current value of the program counter will be pushed onto stack and the Am2914 will be programmed to present the vector of the interrupt to the PROM.

At the same time the PROM output will be enabled to the data bus and the program counter will be loaded with the address of the interrupt service routine. Upon servicing the interrupt, the program counter will be popped off the stack and execution will resume from the suspended point.

MEMORY SYSTEM

Memory capacity of a system depends upon the application and the amount of data being handled by the processor. Although any type of memory can be used in a micro-computer, semiconductor memories are usually used for the cost, speed and hardware reasons. There are two basic applications of the memories in a computer system:

- to store the system monitor (executive). This type of memory is usually a nonvolatile storage (ROM or PROM) and for a small computer the capacity of this memory varies between 1k and 4k
- memories are also used to store the user program (software) or data. This type of memory is usually a RAM and its capacity may range typically from 1k to 64 kwords. (For dedicated applications which always run the same program, the program is usually stored in a ROM or EPROM)

Special care is needed when designing memory systems for the bit-slice bipolar computers. The access time of the memory should be compatible with the speed and requirements of the computer. Because the minimum clock period of a bit-slice computer can be as low as 150 ns then the memory access time should be around 60 ns so that the

memory can be accessed in a single clock cycle. Since it is difficult to achieve such a high speed with the MOS memories, it may be necessary to use MOS memories with access times of about 100 ns, and then to increase the clock period during memory access operations (synchronous memory control).

Bipolar memories are available with access times down to 30 ns but they have several disadvantages such as high cost and high power consumption. Another method is to use the MOS memories and suspend the operations of the processor until the memory system sends a ready signal to the processor (asynchronous memory control). The ready signal can be generated from a delayed version of the memory request signal. Alternatively for higher system throughput, clocked memories can be used (such as Am9130 and Am9140)².

Another problem that has to be considered when using MOS memories in a bipolar computer is the loading and driving requirements of the memory. For large memory systems buffers will be required between the address bus and memory address input to minimise the AC loading on the bus. Also a typical MOS memory can drive two TTL loads and thus the output of the memory system can not drive the data bus directly. This problem can be avoided by using three-state bipolar bus drivers (such as Am74LS241) at the output of the MOS memories.

A block diagram of a typical memory system is shown in Figure 6. The CCU controls the memory system with two signals: MEMRQ and MODE. MEMRQ indicates that there is a request for the memory and the MODE indicates whether it is a read or write operation. The memory blocks are selected by the active low chip select (CS) inputs which are driven by the chip select decoder logic (such as Am74LS138).

TIMING UNIT

The timing unit generates the system clocks, synchronizes the internal and external events, initializes the processor and responds to the external supervisory signals. The system clock can be derived from a crystal controlled oscillator.

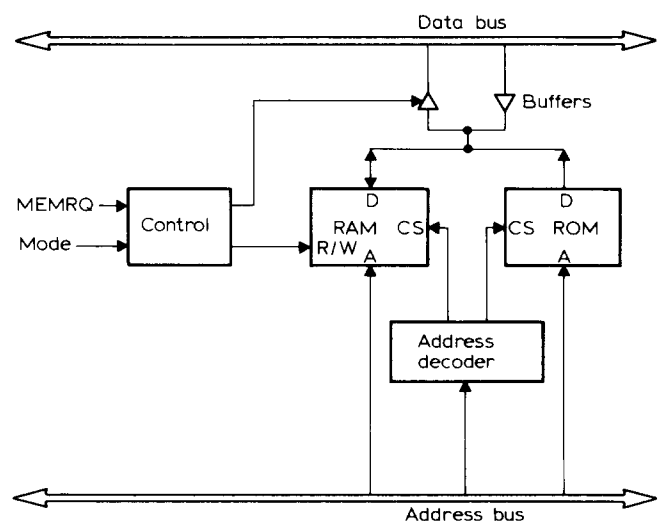


Figure 6. Block diagram of the memory system

The synchronization task of the timing unit allows the events that are slower than the CCU (such as the memory system) to be synchronized with the CCU. During memory access operations the timing unit may disable the clock from the CCU until the memory system has performed the required operation and responded with a ready signal. Upon receiving the ready signal the timing unit will reenale the clock to the CCU. Initialization includes the power-on-reset and external reset. When the power is applied to the system or when the external reset is activated the timing unit presets the system to a known state. By this, all the critical devices (such as the status register, program counter and the microprogram sequencer) are cleared and execution starts from address zero of the microprogram memory. The supervisory signals control the timing unit via external switches and these include the run, halt and single clock. Run enables the timing unit so that clock is distributed to all parts of the computer. Halt signal on the contrary (recognised only during fetch cycle) disables the clock from the computer. Single clock signal (active when the timing unit is in halt mode) sends a single clock pulse to the system each time it is activated. This signal is useful for tracing the execution steps of the computer or it can be used during fault diagnosis.

The Am2925 system clock generator⁶ incorporates all the features of the timing unit described above and all that is required is an external crystal for its voltage controlled oscillator.

MICROPROGRAM FORMATTING AND MICROPROGRAMMING

The first part of this paper and the preceding sections have described the basic design principles of a 16-bit micro-programmable computer. All of the control signals of the computer are derived from the microprogram memory in the CCU.

The approximate number of signals controlling the subsystems of the computer are summarized below:

Subsystem	No. of control signals required
Arithmetic and logic unit	13
Data shifting logic	5
Condition code select logic	3
Operand select logic	10
Data bus buffers	2
Instruction register	1
Microprogram sequencer	4
Microprogram branch address	12
I/O unit	2
Program control unit	4
Interrupt control unit	5
Memory system	2
Timing unit	3

The total number of bits required is approximately 66. A microprogram memory this wide will require many ROM/PROMs, many registers (pipeline) and a large printed circuit board area. This will increase the cost and the power consumption of the system considerably. The microprogram word length can be made shorter by formatting the control

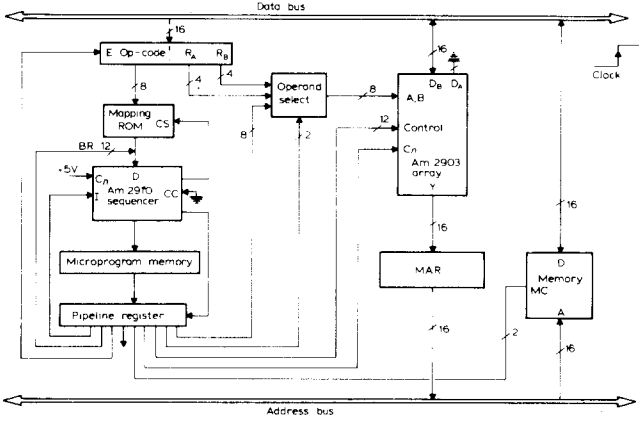


Figure 7. Simplified microprogrammable computer architecture

signals. The control signals that are not expected to be active simultaneously can share the same outputs of the micro-program memory. An example is that the arithmetic and logic unit can share the same bits as the interrupt control unit and the microsequencer branch address. By formatting, the width of the microprogram memory can be reduced to approximately 48 bits.

When the power is first applied to the system or when the reset button is activated, the execution starts from address zero of the microprogram memory where a test will be made to see if an interrupt has occurred.

Microprogram addresses 1 and 2 contain the macro-instruction fetch cycle. The fetch cycle occupies two micro-instructions: firstly to fetch the instruction from the memory and secondly to clock it into the instruction register and present it to the Am2910 sequencer. After the execution of every macroinstruction, a jump to address zero must be performed by the microsequencer in order to fetch the next instruction from the memory.

In order to understand the principles of microprogram-ming, an example will be presented to show the micro-instructions involved in the execution of a typical macro-instruction. Let the block diagram of the computer be as in Figure 7 where it is simplified for the purposes of this example. The program counter unit is omitted and the memory addresses are generated in the Am2903s and sent out to the memory address register (MAR). Assume that register 15 of 2903s is to be used as the program counter register.

The macroinstruction that will be described is the 'Add registers ($R_B + R_A \rightarrow R_B$)' which adds the contents of register B to register A (B and A defined by user) and the result will be stored back in register B. This instruction will take the format:

15	8	3	0
Op-code	1st Operand register No.	2nd Operand and destination register	

Letting the op-code for this instruction be the hexadecimal number 20_H, the instruction will be stored in the memory as 20XY_H where X denotes the first register number and

Y denotes the second register number which is also the destination register number. Thus, the instruction 2012_H will perform the operation $R_1 + R_2 \rightarrow R_2$. If we assume that the microoperations to perform the 'add registers' instruction are stored starting from address 008_H of the microprogram memory then the mapping ROM address 20_H must be loaded with the data 008_H.

Also assume that the instruction 2012_H is stored at location 0145_H of the main memory and that the instruction fetch cycle starts at address 1 of the microprogram memory. Thus, the memory map of the overall system will be as follows:

Main memory		Mapping ROM	
Address	Data	Address	Data
.	---	.	---
.	---	.	---
0144	---	1F	---
0145	2012	20	008
0146	---	21	---
.	---	.	---
.	---	.	---

Microprogram memory

Address	Data
000	---
001	Fetch
002	---
.	---
.	---
.	---
008	Microcode for 'add registers' instruction
009	---
00A	---

The microprogram memory format of the computer is shown in Figure 8. The functions of the microprogram control bits are summarized below:

Field	Function	
I ₀	ALU control bit	
F	ALU function select	
D	ALU destination select	
I _{EN} , O _{EY} , E _A	ALU control bits	
C _n	ALU 'carry in' control	
BR	Microprogram sequencer branch address	
E	When low, instruction register will be loaded with new data on the next clock low-to-high transition	
I	Sequencer control bits	
M, C	Memory control bits, selected as below:	
M	C	Memory operation
0	0	No memory request
0	1	No memory request
1	0	Request: read
1	1	Request: write

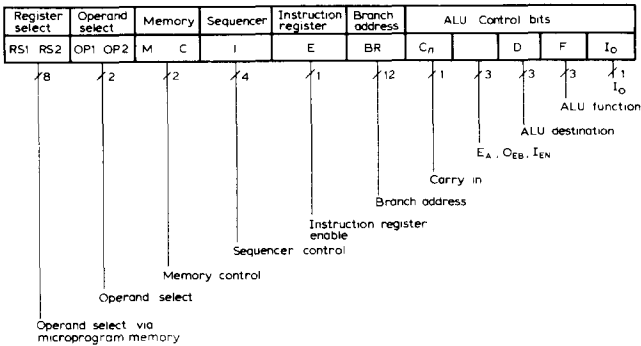


Figure 8. Microprogram memory format of the computer in Figure 7

Instruction	Address	Register select		Operand select		Memory		Am 290	Instruction register	Branch address	ALU control bits						
		RS1	RS2	OP1	OP2	M	C	I			C _n	E _a	O _{EY}	I _{EN}	D	F	I ₀
Fetch	000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fetch +1	001	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fetch +2	002	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0
.	003	x	x	x	x	0	0	2	0	0	0	0	0	0	0	0	0
.	004	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	005	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	006	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	007	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Add registers	008	x	x	0	0	0	0	3	0	0	0	0	0	0	0	0	0
.	009	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	00A	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	00B	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	00C	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figure 9. Microprogram for the 'add registers' macroinstruction (data in hexadecimal; x = don't care)

OP1, OP2		Operand select logic control bits, selected as below:	
OP1	OP2	A address	B address
0	0	R _A	R _B
0	1	R _B	R _A
1	0	R _A	R _A
1	1	RS1	RS2
RS1, RS2		Operand register number (when selected via the microprogram memory)	

The microinstructions for the implementation of the macroinstruction $R_B + R_A \rightarrow R_B$ is shown in Figure 9: Initially assume that 0145_H, the address of the 'add registers' instructions is in R15, the program counter register, and that the microprogram sequencer contains 001_H at its output. On the first clock pulse, the microinstruction at address 001_H of the microprogram memory will be transferred to the output of the pipeline register and the sequencer output will go to 002_H. This puts the system into the beginning of the instruction fetch cycle and the following will happen:

- the function of the fetch cycle is to send the contents of the program counter (R15) to the memory address register (MAR) and to increment the program counter. The ALU performs the operation $R15 + 1 \rightarrow Y$ and the result appears at the input of the MAR
- on the next clock pulse, the MAR will be loaded with 0145_H and a memory read operation is performed.

At the end of this cycle, the instruction 2012_H will be on the data bus at the input of the instruction register. During this cycle the program counter is incremented by performing the operation $R15 + C_n \rightarrow R15$ and the instruction register is enabled ($E = \text{low}$)

- after the third clock pulse, the instruction 2012_H will be loaded into the instruction register and the op-code will be decoded by the mapping ROM into the data 008_H . The sequencer selects this data at its output and feeds it to the microprogram memory so that the pipeline register input will contain the microinstruction 008_H at the end of this cycle
- on the fourth clock pulse, the microinstruction 008_H will be at the output of the pipeline register. The operand select logic will select the R_A and R_B fields of the operand to the RAM A and B addresses of the ALU respectively so that RAM A address will contain 001_H and the RAM B address will contain 002_H . During this cycle, the ALU will add the contents of RAM A address to the contents of RAM B address and the result will be stored in the RAM address selected by the B address-field. During this cycle, the sequencer performs a branch operation where the branch address is the microprogram address 001_H (The fetch microinstruction) and by the end of this cycle the fetch microinstruction will be at the input of the pipeline register
- on the next clock pulse, the fetch microinstruction will be at the output of the pipeline register and the next macroinstruction will be fetched from the memory

CONCLUSIONS

The basic design principles of a 16-Bit microprogrammable computer have been discussed. Up to a few years ago the

LSI support devices for bit-slice computers were very limited in number and function and the design engineer had the task of connecting the available bipolar MSI logic circuits to generate the required function. A large team of engineers were required to carry out the design and testing of the computer. Nowadays the bit-slice devices are supported by a large number of bipolar LSI devices. This gave rise to an increase in the overall system reliability. The power consumption reduced by a large amount and a smaller printed circuit area is required. Also the design and testing of the computer are simplified by a large amount and only a few engineers are required to build a microprogrammable computer.

ACKNOWLEDGEMENT

The author wishes to thank the Science Research Council for financial support.

REFERENCES

- 1 Peatman, J B *Microcomputer based design* McGraw-Hill Inc. (1977)
- 2 *Am2900 Microprocessor family* Advanced Micro Devices Inc., Sunnyvale, California (1976)
- 3 *Microprogramming handbook* Advanced Micro Devices Inc., Sunnyvale, California (1976)
- 4 *An introduction to microcomputers* Adam Osborne and Associates Inc., Berkley, California (1975)
- 5 Kraft, G D and Troy, W N *Mini/microcomputer hardware design* Bell Telephone Laboratories Inc. (1979)
- 6 *The Am2900 family data book* Advanced Micro Devices Inc., Sunnyvale, California (1978)



the leading international journal for users and developers of the computer as a design aid

Coverage includes: * Building design * Civil and structural engineering * Mechanical engineering and manufacture * Electrical engineering * Chemical engineering * Marine engineering * CAD/CAM systems * Numerical control

ANNUAL SUBSCRIPTION (six issues) £50.00 UK, £55.00 Overseas (\$143.00)

Further details from:

Marilyn Fitchett, IPC Science & Technology Press Ltd., PO Box 63, Westbury House, Bury Street, Guildford, Surrey GU2 5BH, England.
Telephone: (0483) 31261. Telex: 859556 Scitec G