Internal Architecture of Microprocessors

Chapter 5

INTRODUCTION

Most single-chip microprocessors use MOS technology to obtain the very high packing density which is necessary to accommodate a complete processor on one integrated circuit. Unfortunately, MOS logic is relatively slow and faster operating speeds can be obtained with other types of logic systems such as Schottky TTL. However, these faster logic systems are not usually suitable for very high circuit densities and as a result it is necessary to build the microprocessor as a group of integrated circuits rather than as a single package. In breaking down the processor into smaller elements some other advantages accrue to the system designer. These include greater control over the instruction set and facilities for incorporating high speed logic techniques which are too complex for inclusion in a single chip device. This chapter attempts to explain the basic internal architecture of a microprocessor and to cover the various techniques which can be used when the MPU is split into several sections.

For the most part this text concentrates on what are known as "bit-slice" machines. That is, microprocessors which are available as a group of high speed circuit elements rather than as a complete MPU in one integrated circuit package. This makes it possible to adopt a more generalised approach to the subject as well as covering a class of microprocessors which, so far, have been precluded from consideration.

THE INTERNAL ELEMENTS OF A MICROPROCESSOR CIRCUIT

A microprocessor is built using three basic circuit blocks:

(i) Registers
(ii) Arithmetic and logical unit (ALU)
(iii) Control unit

Registers can exist in two forms, either as an array of static memory elements such as flip-flops or as a portion of a random access memory (RAM) which may be of the dynamic or static type. Some microprocessors use both techniques on one chip so that, for example, the accumulator might be composed of static memory devices and all the other registers could be combined into a common dynamic random access memory. In addition to the registers which can be addressed and manipulated by the instruction set, buffer registers are often used for temporary storage of binary information which is being moved from point to point within the MPU. Clearly, a microprocessor which uses dynamic memory techniques must be continuously driven by clock pulses in order to preserve data integrity. Where all registers are implemented with static cells the train of clock pulses may be stopped without losing the stored information (see Chapter 6).

The arithmetic and logic unit usually provides, at the minimum, facilities for addition, subtraction, OR, AND and complementation. Shift operations can be accommodated either by the data selector approach, or by a shift register.

The function of the control unit is somewhat analogous to the puppeteer who manipulates strings so as to cause a puppet to dance the appropriate steps. With an MPU the registers and ALU form the puppet and the control unit is the puppeteer. For most MPU's the control unit accounts for more than half of the total circuit and is highly complex. The control unit uses a principle known as microprogramming to interpret each instruction and cause the processor to perform the required operation. Microprogramming recognises that, in order for the MPU to execute a single instruction, it must go through several steps. The full sequence of steps associated with a single instruction can be

regarded as a small program – hence "microprogramming." The microprogram for each instruction is held in a ROM and as a particular instruction is received, the control unit calls up the appropriate microprogram.

A BIT-SLICE CENTRAL PROCESSING ELEMENT (CPE)

It is common in bit-slice machines to incorporate all the registers, ALU and other related circuits, into a single central processing element (CPE). But it is often impossible to include enough gates to form a full 8- or 16-bit word machine on a single chip, and consequently the central processing element is usually provided as a 2- or 4-bit "slice" of a full processor. Several CPE's are connected together to build up the desired word size in much the same way as 4-bit adders are cascaded to give 16-bit capability.

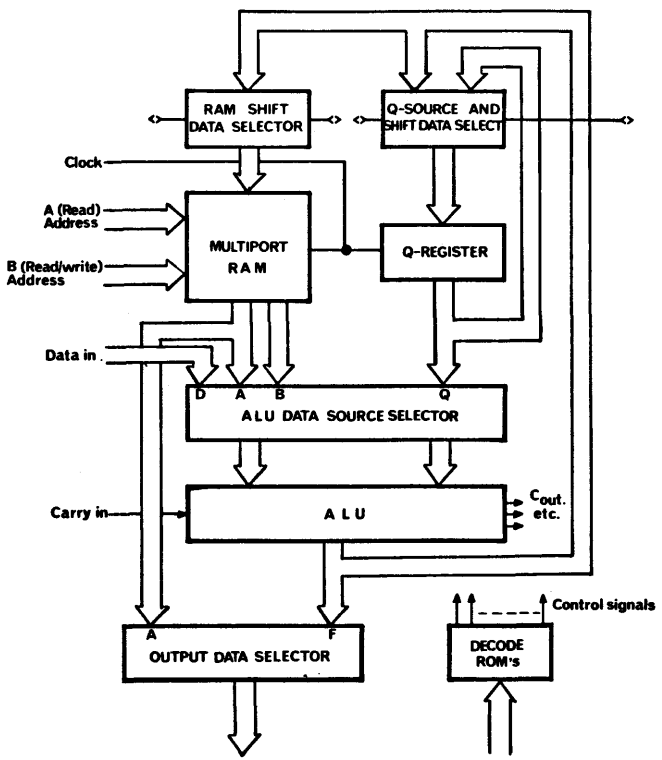

Figure 5-1. Typical 4-bit slice central processing element

Figure 5-1 shows a typical 4-bit slice central processing element. It consists of a multiport RAM which provides sixteen 4-bit registers, an extension register Q which is particularly useful for multiplication, division and other double word length operations, an ALU, two buffer registers A and B, and a read-only-memory (ROM) for decoding the nanoinstructions. A nanoinstruction is a single binary word presented to the CPE which causes the device to carry out one particular operation. The ROM decodes the nanoinstruction into binary control lines which are used internally to control the function of the ALU and the addresses of the multiplexers. It is necessary to use the ROM as a nanoinstruction decoder so as to reduce the nanoinstruction. Typically, the nanoinstruction might be 8 bits wide, whereas the CPE may require control of 20 or more binary lines for each nanoinstruction.

The precise allocation and use of the 16 registers is defined by the system designer. For example, one of the registers would be used as a program counter, another as a stack pointer, and so on according to the chosen architecture of the machine.

A BASIC MICROPROGRAM CONTROL UNIT

Figure 5-2 shows the block diagram of a simple microprogram control unit. It comprises an instruction register which receives the instruction, a starting address ROM, a data selector which selects the address source for the microprogram ROM, the microprogram ROM, and a data selector which is used for conditional jump operations.

The instruction is loaded into the instruction register at the start of an instruction cycle and the op-code is mapped by the starting address ROM so as to specify the beginning of a series of microinstructions which cause the full instruction to be executed. By using a starting address ROM it is possible to make more efficient use of the microprogram ROM and thereby reduce its size. Each word in the microprogram memory is known as a microinstruction and has three fundamental segments, namely, the nanoinstruction which controls the CPE, a next address for the location of the next microinstruction in the sequence and a condition code select address which drives the condition code multiplexer. An additional

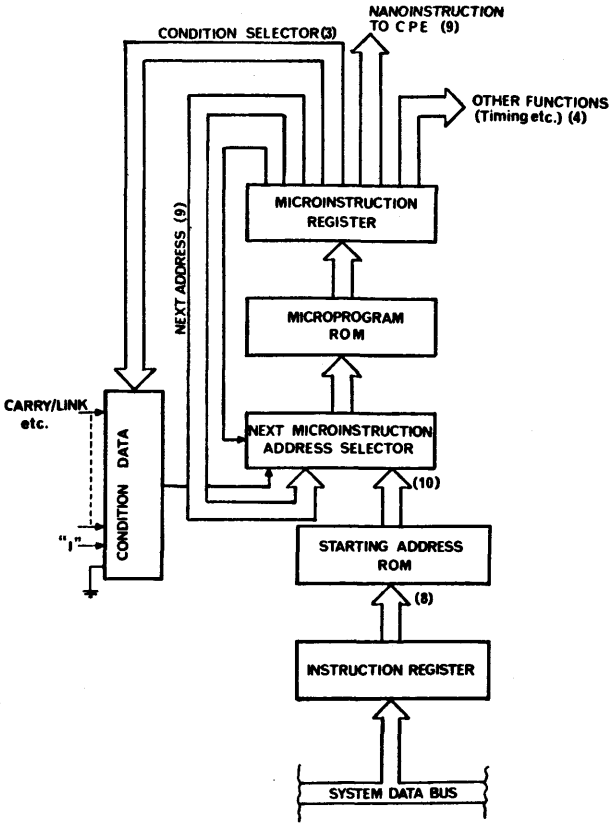

Figure 5-2. A simple microprogram control unit

single bit drives the microprogram address source data selector and usually a few more bits are required to interface with the timing and interrupt logic. For the first microinstruction in a full macroinstruction* the starting address ROM is selected as the ROM address source and thereafter each succeeding microinstruction uses the single bit control to cause the data selector to select the microprogram ROM for the next address. The last microinstruction in the sequence switches the data selector back to obtain the microprogram starting address of the next instruction.

---

*In order to distinguish a normal software instruction from a microinstruction, it is conventional to refer to the former as a macroinstruction.

The single bit output of the condition code selector supplies one bit of the next address so that the next microinstruction is selected from one of two possible instructions by the selected condition. For example, in Figure 5-2, when the selected condition is the carry-link bit, the next microinstruction can be one of two depending on whether the carry-link bit is true or false.

The size of the various addresses and ROM's in a microprogram control unit vary from one processor to another. However, in order to give a better appreciation of the approximate number of bits, some rough indication for an 8-bit machine is given in Figure 5-2. It will be observed that the microprogram ROM is quite large. Many instruction sets and microprogram designs are structured to reduce the size of the microprogram ROM.

ENHANCEMENTS TO THE MICROPROGRAM CONTROL UNIT

It is desirable to reduce the size of the microprogram ROM, not only for cost reasons but also to limit the overall number of pins and interconnections in a given design. Several methods exist for doing this. One common approach recognises that most instructions are carried out by moving through a sequence of microinstructions which are usually stored in adjacent addresses in the microprogram ROM. For this situation, it is adequate to generate the next microinstruction address simply by adding "1" to the current address and, under these circumstances, the amount of microprogram ROM storage needed to specify the next address is reduced. However, at the end of an instruction, it is usual to have an unconditional jump to the set of microinstructions which test for an interrupt before fetching the next instruction. In this, and in many other cases, it is necessary to specify the next microinstruction address in more detail. This is done by setting the nanoinstruction to the CPE in the "no-op" state and using other bits, such as the condition select bits, to form part of the next address. The precise manner in which this next address is encoded into other address fields varies from one processor to another.

The power of microprogramming lies in the tremendous scope for specifying quite a long and complicated process, such as multiply

or divide, in a single instruction. In order to realise this power it is advantageous to have the capability for one microprogram to use another microprogram on a subroutine basis. This can be done by providing the microprogram control unit with a stack for storing subroutine return addresses, in much the same way as a microprocessor uses a stack to handle normal software subroutines. For example, the microprogram for finding the square root of a number c, using the algorithm

$$x = \frac{1}{2}\left(x + \frac{c}{x}\right)$$

would call the division microprogram subroutine in order to evaluate c/x.

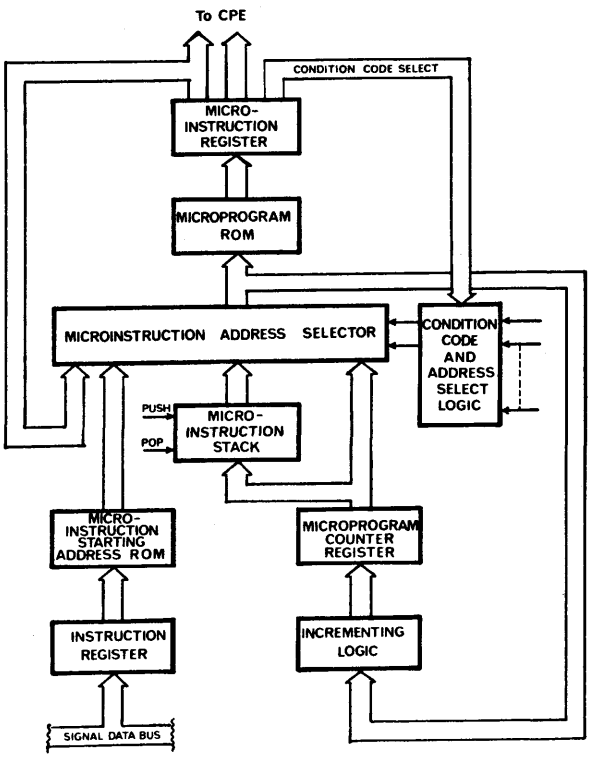

Figure 5-3. Typical bit-slice microprogram control unit

Some instructions, such as multiply and multiple-shift, require that the same sequence of microinstructions be executed several times. Usually a counter is used for this purpose, and at the beginning of the instruction the counter is loaded with the number of times the instruction is to be repeated. Each time the microinstruction sequence is carried out the counter is decremented by 1 until, at the end of the sequence, the zero condition of the counter is used to cause a microprogram branch out of the loop.

Figure 5-3 shows a typical microprogram control unit for a bit-slice machine. Note that the address of the next microinstruction can be selected from one of 4 sources:

(i) The microinstruction itself, as with the simple unit of Figure 5-3.
(ii) An incrementer which adds 1 to the current microprogram address.
(iii) The microinstruction starting address ROM.
(iv) The Push-Pop stack used for storing microinstruction subroutine return addresses.

PIPELINING

Pipelining has been defined as "starting the next job before the current one is finished" but the precise application of pipelining to microprocessor design varies a great deal. Broadly speaking, pipelining consists of adding auxiliary timing registers at certain key locations so that memories can be addressed in advance of their outputs being required. This technique helps to reduce any system slow-down which occurs as a result of memory access time. For example, in the simple control unit of Figure 5-2 it would be advantageous to load the instruction register in advance of the beginning of the instruction cycle, so that when the cycle does begin, the microprogram starting address is already available at the data selector. In this way the access time of the starting address ROM does not add to the overall system timing.

A SIMPLE EXAMPLE OF A MICROPROGRAMMED PROCESSOR

Figure 5-4 shows the overall structure of a simple processor

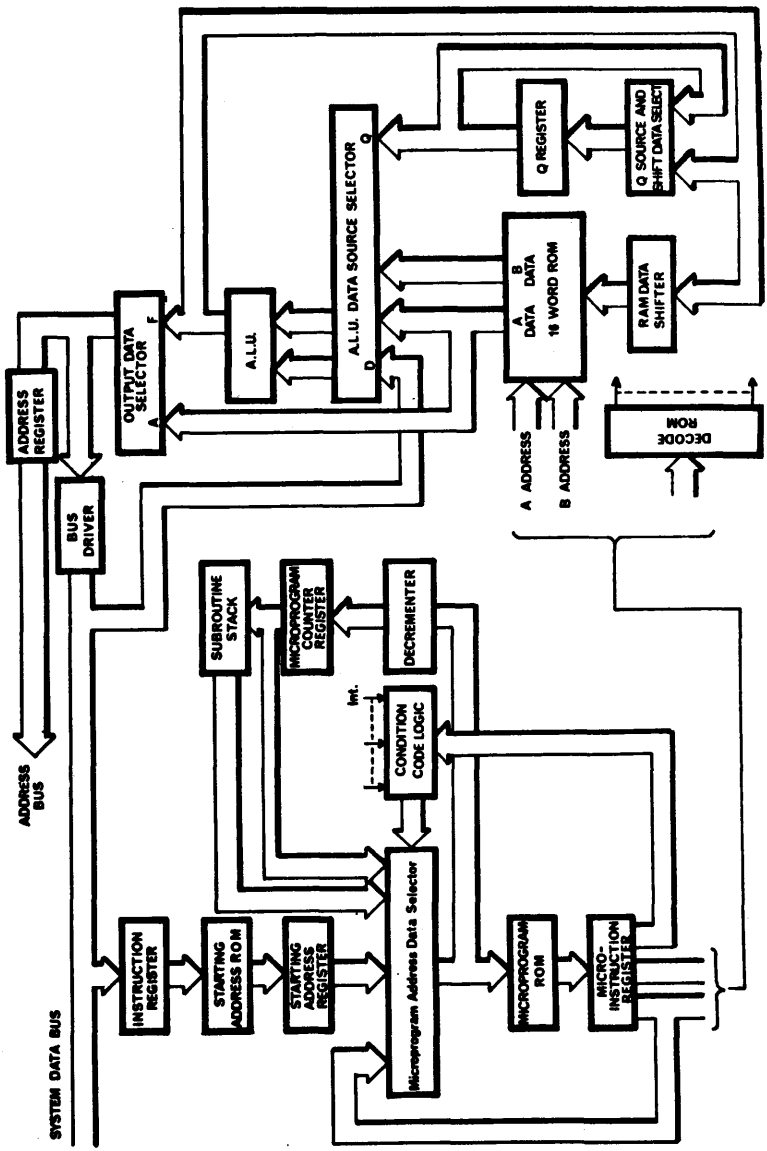

Figure 5-4. Simplified microprogrammed processor using bit-slice elements

operating under microprogram control. Each instruction follow the state graph of Figure 5-5. The loop of instructions, beginnin with State D through A and B, are the normal states followed b each instruction and it is convenient for explanatory purpose to begin with State D. The sequence of events beginning wit State D is as follows:

State D. Microinstruction switches condition selector t examine interrupt line. If line shows no interrupt, then nex address selector selects next consecutive microinstruction addres to give State A. If line shows interrupt request, next micrc program address is taken from current microinstruction to giv State E.

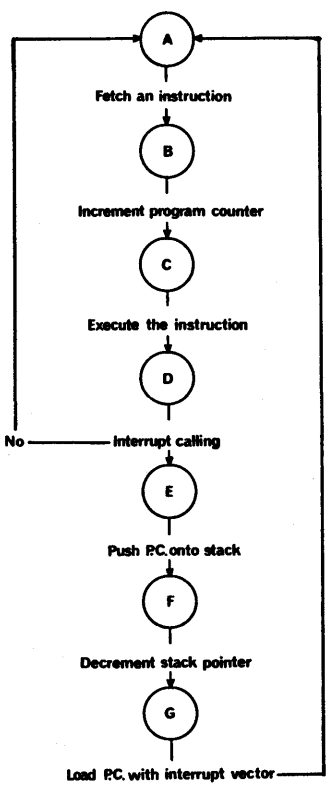

Figure 5-5. State transition diagram for simple processor

State A. Program counter contents are sent out on address bus and incoming data is loaded into instruction register. Microinstruction selects next consecutive address for next microinstruction.

State B. Program counter is incremented by 1 using the ALU and contents are returned via internal bus to the program counter. Microinstruction selects starting address ROM as the address for the next microinstruction.

State C. Execute the instruction.

After executing the instruction the microprogram control unit makes an unconditional jump to State D to begin the same sequence over again.

Specific locations in the 16-word processor RAM are allocated particular duties. A typical allocation might be:

Address 0	General purpose
Address 10	General purpose
Address 11	Index register 2
Address 12	Index register 1
Address 13	Interrupt vector
Address 14	Stack pointer
Address 15	Program counter

Then, for example, incrementing the program counter is done by placing address 15 on the B address lines, setting the ALU to perform the function B + 1 and taking the F outputs of the ALU straight through the RAM shift circuit and back into RAM location 15. On the positive going edge of the clock pulse, location 15 is loaded with the value (p.c. + 1). Incrementing and decrementing any register can be done in much the same way.

As an example of how a full instruction might be executed, consider the case of an interrupt. Normally an interrupt is initiated by an external event but some processors have a single instruction which causes the microprogram control unit to respond as if a hardware interrupt had occurred. The interrupt instruction, usually called software interrupt, is mapped by the microprogram starting

address ROM to give exactly the same starting address as State D on the state transition diagram gives if a hardware interrupt exists. The first microinstruction places address 14 (stack pointer) on the A address lines of the processor RAM and the output data selector selects data bus A and passes this information to the address bus latch.

Hence on the positive going edge of the clock pulse the address register will be loaded with the stack location to be used. At the same time the program counter contents are incremented by 1 as detailed above. The bus structure allows the two jobs to be done concurrently. The next microinstruction in sequence is now called and this places address 15 (program counter) on the A address lines. The output data selector again selects data bus A and the main data bus transceiver routes this information onto the main data bus. On the next positive going edge of the clock pulse, this data (i.e., program counter contents) is written into the main system memory. The stack pointer is concurrently decremented by 1 by putting address 14 on the processor B address lines, setting the ALU to provide outputs (B-1) so that as the program counter contents are pushed onto the stack the stack pointer is decremented. The next sequential microinstruction also provides two simultaneous operations. Firstly, it takes the interrupt vector from location 13 and loads it onto the program counter (location 15). It places 13 on the processor address lines A and 15 on the processor address lines B. Secondly, it routes output A straight through the ALU and the shifter back into the processor RAM. Since the operation makes the next instruction address (i.e., interrupt vector) available on the A data bus, it is taken through the output data selector and loaded to the main address register so that the next memory address is set up well in advance of reading the next instruction. The microprogram control unit then jumps to State A to continue processing. In order to avoid an interrupt interrupting itself, one of the microinstructions in the interrupt handling microprogram usually sets a flag to mask off the interrupts.

Clearly, the microprogram for a particular processor is greatly influenced by the internal architecture of the processor itself. The simple example given above assumes quite a sophisticated

internal bus structure so as to illustrate how one microinstruction can perform several tasks. However, many microprocessors have much simpler internal bus systems and their execution time is somewhat greater because data has to be moved from register to register in a more restricted manner.

The bit-slice approach gives the engineer much the same flexibility in computer hardware as TTL logic gave to the digital systems engineer. However, this flexibility is not always needed or desired and a more limited approach is often preferred.

MULTI-CHIP MICROPROCESSORS

In this section it is necessary to draw the distinction between bit-slice machines which allow variable hardware designs, and multi-chip processors which have a fixed hardware design but permit the user to specify the operation of the microprogram control unit by changing the microprogram ROM's. In its simplest form a multi-chip fixed architecture microprocessor consists of three circuits:

(i) The central processor which may have 8-, 12- or 16-bit capability
(ii) The microprogram sequence control logic
(iii) The microprogram ROM.

The microprogram ROM pattern can be changed quite easily so that the user has a great deal of influence over the way in which each instruction is executed. This ROM is often referred to as the Control ROM (or CROM).

Since the microprogram ROM has more or less complete control over how the processor executes each instruction, it is possible to cause the microprocessor to interpret particular instruction bit patterns in many different ways depending on the ROM pattern. This ability of a processor to behave quite differently according to its microprogram gives rise to the concept of emulation. This means that a processor is microprogrammed to respond as if it were some other processor. The emulator accepts instructions encoded according to the format of the processor which it is emulating, and interprets those instructions in exactly the same

way as the original machine does. Thus it is possible for the emulator to run software developed for the machine which it is emulating. Many minicomputer manufacturers have used emulation to build low-cost microprocessor versions of their own machines, and thereby maintain software compatibility throughout their range of computers.

Another application for multiple chip microprocessors is in the design of special-purpose machines which require unusual instructions. For example, in nonlinear process control it is useful to have single instructions which evaluate polynomials or fit a curve to a set of points. These special instructions can be provided at microprogram level by encoding the program into the microprogram ROM and this method has the advantage that operations can be done much more quickly at microprogram level than under normal macroprogram software. A good example of this is the multiply and divide instructions available on some microprocessors; these instructions are an order of magnitude faster than the same program run in macroinstruction form.

SUMMARY

The internal architecture of microprocessors does not conform to some standard layout; it is influenced by many factors but it always contains an ALU, registers and a microprogram control unit. Careful attention to architecture can yield considerable improvement in instruction execution time and it is likely that advanced hardware techniques, such as pipelining, will be used more and more in microprocessors so as to obtain higher software throughput with the same basic semiconductor technologies.