

## Chapter 5

### Digital Signal Processor

#### Introduction

A system based on a single chip has wide-ranging advantages over other approaches. Not only are power, space, and assembly-time requirements greatly reduced, but development and verification efforts are concentrated in software, where reliable simulation tools and structured design methodologies can be employed to full effect. Further, programmable systems are more flexible; existing solutions can be adapted to changing specifications with simple modifications of the software.

A powerful new class of VLSI microprocessors, commonly known as digital signal processors or DSPs, has emerged within recent years. These devices have been designed primarily for digital signal processing applications. Most of these processors have been classified as DSPs because of the inclusion of an on-chip hardware multiplier as well as other features that make them particularly attractive for computation-intensive applications such as digital signal processing. Nearly all of them, for example, allow computation of a running sum of product (multiply/accumulate), and most allow a shift in memory of one of the multiplicand.

In this chapter, an overview of signal processing is presented to discuss the meaning and potential of digital signal processing. In DSP section, we provide the information about its architecture, instruction set and methodology for application development. The instruction summary of DSP is presented in appendix E.

#### Signal Processing

Digital signal processing has become an important modern tool in fields of science and technology. Digital signal processing is concerned with the representation of signals by sequences of numbers or symbols and the processing of these sequences. The purpose of such processing may be to estimate characteristic parameters of a signal or to transform a signal into a more desirable form. The classical numerical analysis formulas, such as those designed for interpolation, integration, and differentiation, are certainly digital signal processing algorithms. On the other hand, the availability of high speed digital computers has promoted the development of increasingly complex and sophisticated signal processing algorithms, and recent advances in integrated circuit technology promise economical implementations of very complex digital signal processing systems.

Signal processing, its importance is evident in such diverse fields as biomedical engineering, acoustics, sonar, radar, seismology, speech communication, data communication, nuclear science, and many others. In many applications, we may wish to extract some characteristic parameters. Alternatively we may wish to remove interference, such as noise, from the signal or to modify the signal to present it in a form which is more easily interpreted. Signal processing problems are not confined to one-dimensional signals. Many image processing applications require the use of two-dimensional signal processing techniques.

Until recently, signal processing has typically been carried out using analog equipment. In the 1950s, the analysis of some geophysical data which could be recorded on magnetic tape for later processing on a large digital computer was required. This class of problems was one of the first examples of signal processing using digital computers.

Because of the flexibility of digital computers, it was often useful to simulate a signal processing system on a digital computer before implementing it in analog hardware. In this way, a new signal processing algorithm, or system, could be studied in a flexible experimental environment before constructing it. Consequently, a dominant attitude at that time was that the digital computer was being used to approximate, or simulate, an analog signal processing system.

As signals were being processed on digital computers, there was a natural tendency to experiment with increasingly sophisticated signal processing algorithms. Some of these algorithms grew out of the flexibility of the digital computer and had no practical implementation in analog equipment. The techniques and applications of digital signal processing are expanding at a tremendous rate. With the coming of large scale integration and the resulting reduction in cost and size of digital components, together with increasing speed, the class of applications of digital signal processing techniques is growing.

#### Digital Signal Processor

Digital Signal Processing is concerned with the representation of signals (and the information that they contain) by sequences of numbers, and the transformation or processing of such signal representations by numerical computation procedures.

Since the late 1950's, scientists and engineers in research labs have been promoting the advantage of digital signal processing, but practical considerations have prevented application. Now, with the availability of integrated circuits, such as Texas Instruments' TMS320, digital signal processing is leaving the laboratory and entering the world of application. The

reasons for this is that extremely sophisticated signal processing functions can be difficult or impossible to implement using analog methods.

The potential applications will be found in any area where signals arise as representations of information. In many cases, the signals represent information about the state of some physical system. Often, the objective in processing the signal is to prepare the signal for digital transmission to a remote location or for digital storage of the information for later reference. On the other hand, the signal may be processed to remove distortions introduced by transducers, the signal generation environment, or by a transmission system.

### TMS32010

The TMS32010 is the first member of the TMS320 digital signal processor family, designed to support a wide range of high-speed or numeric-intensive applications. The TMS32010 is capable of executing five million instructions per second. This is the result of the comprehensive, efficient, and easily programmed instruction set. Its key features are as follows

- 200 ns instruction cycle
- 288 bytes on-chip data RAM
- 8K bytes external program memory
- 16-bit instruction/data word
- 32-bit ALU/accumulator
- 16 x 16-bit multiply in 200 ns
- 0 to 15-bit barrel shifter
- 8 input and 8 output channels
- 16-bit bidirectional data bus
- Interrupt with full context save
- Signed two's complement fixed-point arithmetic
- Single 5-V supply
- 40-pin DIP

In the following sections, the information about its architecture, instruction set and methodology for application development are presented.

#### 1. Architecture

The TMS320 family utilizes a modified Harvard architecture for speed and flexibility. In a strict Harvard architecture, program and data memory lie in two separate spaces, permitting a full overlap of the instruction fetch and execution. The TMS320 family's modification of the

Harvard architecture allows transfers between program and data spaces to increase the flexibility of the device. This modification permits coefficients stored in program memory to be read into the RAM, eliminating the need for a separate coefficient ROM. It also makes available immediate instructions and subroutines based on computed values.

The TMS32010 utilizes hardware to implement functions that other processors typically perform in software. For example, the TMS32010 contains a hardware multiplier to perform a multiplication in a single 200 ns cycle. There is also a hardware barrel shifter for shifting data on its way into the ALU.

### 1.1 Harvard Architecture

The TMS32010 utilizes a modified Harvard architecture in which program memory and data memory lie in two separate spaces. This permits a full overlap of instruction fetch and execution. A block diagram of the TMS32010 is illustrated in Fig. 5-1.

The maximum amount of program memory that can be directly addressed is 4K X 16-bit words. Instruction in program memory are executed at full speed. Fast memories with access times of under 100 ns are required.

Data memory is the 144 X 16-bit on-chip data RAM. Instruction operands are fetched from this RAM; no instruction operands can be directly fetched from program memory. However, data can be written into the data RAM from a peripheral or read from program memory.

ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

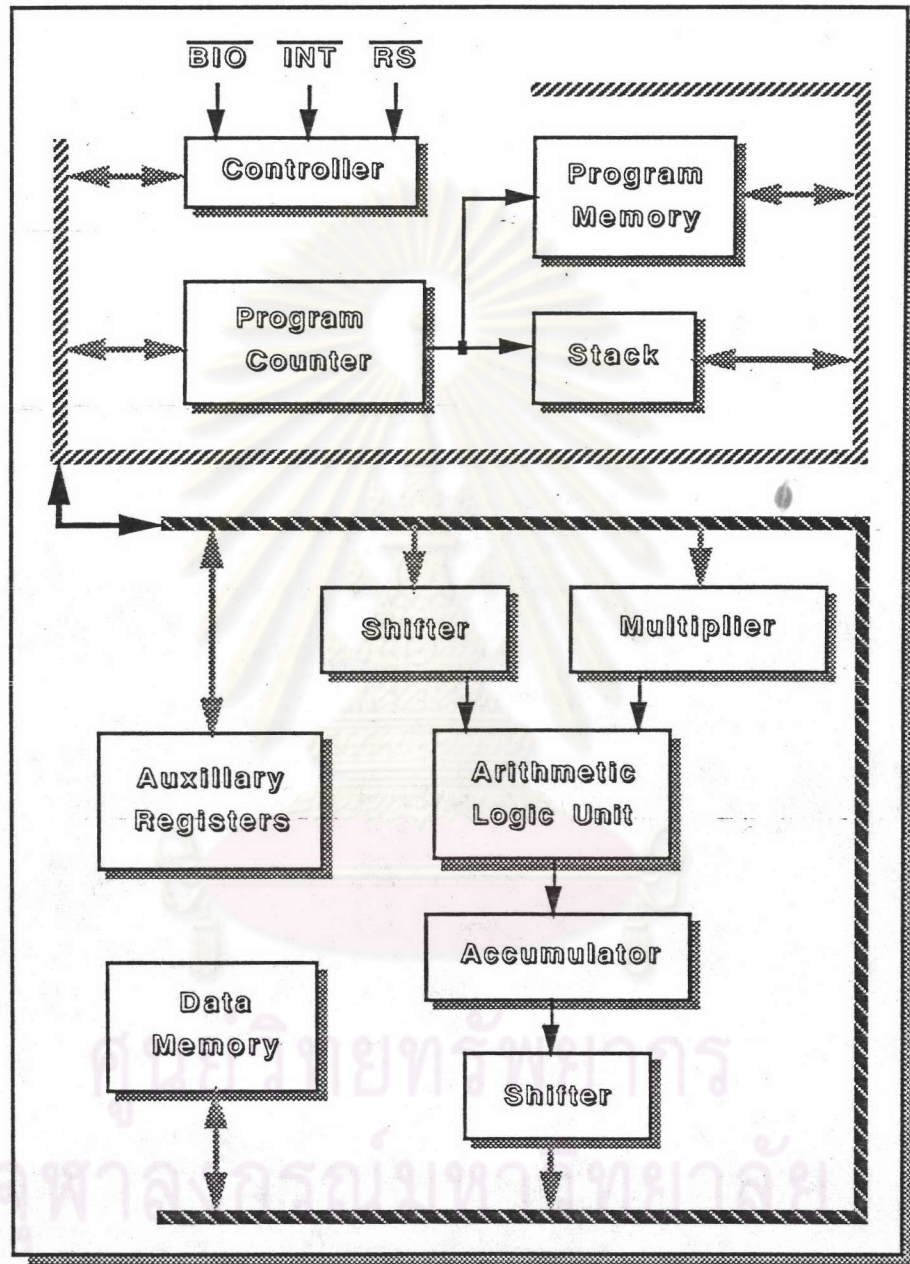


Fig. 5-1 Block diagram of TMS32010.

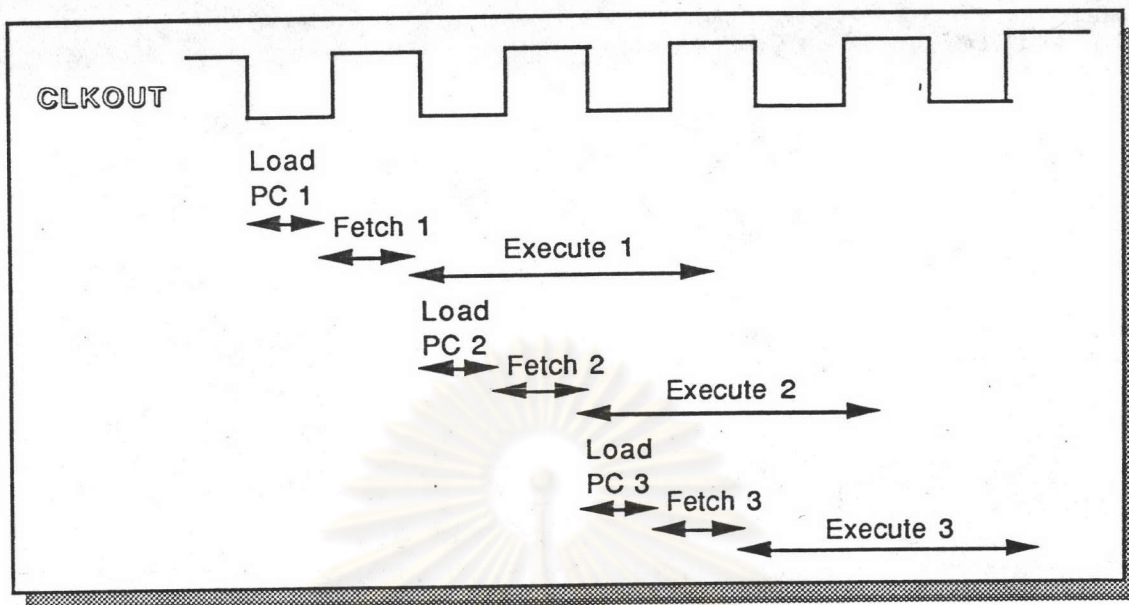


Fig. 5-2 The instruction prefetch and execution cycles of Harvard architecture.

Fig. 5-2 outlines the overlap of the instruction prefetch and execution. On the falling edge of CLKOUT, the program counter (PC) is loaded with the instruction (load PC2) to be prefetched while the current instruction (execute 1) is decoded and is started to be executed. The next instruction is then fetched (fetch 2) while the current instruction continues to execute (execute 1). Even as another prefetch occurs (fetch 3), both the current instruction (execute 2) and the previous instruction are still executing.

## 1.2 Arithmetic Elements

There are four basic arithmetic elements: the ALU, the accumulator, the multiplier, and the shifters. All arithmetic operations are performed using two's complement arithmetic.

Most arithmetic instruction will access a word in the data RAM, and pass it through the barrel shifter. This shifter can left-shift a word 0 to 15-bits, depending on the value specified by the instruction. The data word then enters the ALU where it is loaded into or added/subtracted from the accumulator. After a result is obtained in the accumulator, it can be stored in the data RAM. A parallel left-shifter is present at the accumulator output to aid in scaling results as they are being moved to the data RAM.

### 1.2.1 ALU

The ALU is a general-purpose arithmetic logic unit that operates with a 32-bit data word. The unit will add, subtract, and perform logical operations. The accumulator is always the destination and the primary operand.

### 1.2.2 Accumulator

The accumulator stores the output from the ALU and is also often an input to the ALU. It operates with a 32-bit word length.

### 1.2.3 Multiplier

The 16x16-bit parallel multiplier consists of three units: the T register, the P register, and the multiplier array. The T register is a 16-bit register that stores the multiplicand, while the P register is a 32-bit register that stores the product.

### 1.2.4 Shifters

There are two shifters available for manipulating data: a barrel shifter for shifting data from the data RAM into the ALU and a parallel shifter for shifting the accumulator into the data RAM.

The barrel shifter performs a left-shift of 0 to 15 places on all data memory words that are to be loaded into, subtracted from, or added to the accumulator. The barrel shifter zero-fills the low-order bits and sign-extends the 16-bit data memory word to 32 bits by what is called an arithmetic left-shift. An arithmetic left-shift means that the bits to the left of the MSB of the data word are filled with ones if the MSB is a one or with zeros if the MSB is a zero. This is different from a logical left-shift where the bits to the left of the MSB are always filled with zeros.

The parallel shifter is activated only by the store high-order accumulator word instruction. This shifter left-shifts the entire 32-bit accumulator and places 16 bits into the data RAM, resulting in a loss of the accumulator's high-order bits.

## 1.3 Data Memory

Data memory consists of the 144 words of 16-bit width of RAM present on-chip. All non-immediate data operands reside within this RAM.

## 1.4 Registers

### 1.4.1 Auxiliary Registers

There are two 16-bit hardware registers, the auxiliary registers, that are not part of the data RAM. These auxiliary registers can be used for three functions: temporary storage, indirect addressing of data memory, and loop control.

### 1.4.2 Auxiliary Register Pointer

The auxiliary register pointer is a single bit which is part of the status register. It indicates which auxiliary register is current.

### 1.5 Program Memory

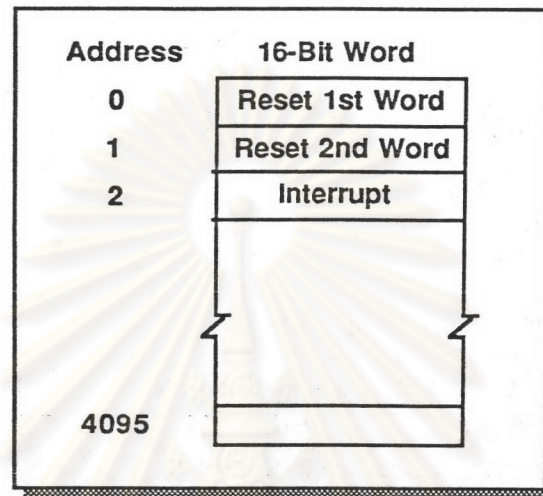


Fig. 5-3 TMS32010 memory map.

Program memory consists of up to 4 K words of 16-bit width. After reset, the TMS32010 will begin execution at location 0. Usually a branch instruction to the reset routine is contained in locations 0 and 1. Upon interrupt, the TMS32010 will begin execution at location 2.

### 1.6 Program Counter and Stack

The program counter (PC) and stack enable the user to perform branches, subroutine calls, and interrupts.

#### 1.6.1 Program counter

The program counter (PC) is a 12-bit register that contains the program memory address of the next instruction to be executed. The device reads the instruction from the program memory location addressed by the PC and increments the PC in preparation for the next instruction prefetch. The PC is initialized to zero by activating the reset line.

Program memory is always addressed by the contents of the PC. The contents of the PC can be changed by a branch instruction if the particular branch condition being tested is true. Otherwise, the branch instruction simply increments the PC.



### 1.6.2 Stack

The stack is 12 bits wide and four layers deep. The PUSH instruction pushes the twelve LSBs of the accumulator onto the top of stack (TOS). The POP instruction pops the TOS into the twelve LSBs of the accumulator.

### 1.7 Status Register

The status register, consists of five status bits. These status bits can be individually altered through dedicated instruction. In addition, the entire status register can be saved in data memory. New values can be reloaded into the status register.

### 1.8 Input/Output Functions

#### 1.8.1 Input and Output

Input and output of data to and from a peripheral is accomplished by the IN and OUT instructions. Data is transferred over the 16-bit data bus to and from the data memory. 128 I/O bits are available for interfacing to peripheral devices: eight 16-bit multiplexed input ports and eight 16-bit multiplexed output ports.

#### 1.8.2 Table Read (TBLR) and Table Write (TRLW)

The TBLR and the TBLW instructions allow words to be transferred between program and data spaces. TBLR is used to read words from program memory into the data RAM. TBLW is used to write words from data RAM to program memory.

### 1.9 BIO Pin

The BIO pin is an external pin which supports bit test and jump operations. When a low is present on this pin, execution of the BIOZ instruction will cause a branch to occur. The BIO pin is useful for monitoring peripheral device status. It is especially useful as an alternative to using an interrupt when it is necessary not to disturb time-critical loops.

### 1.10 Interrupt

The TMS32010's interrupt is generated either by applying a negative-going edge to the interrupt ( $\overline{\text{INT}}$ ) pin or by holding the  $\overline{\text{INT}}$  pin low.

### 1.11 Reset

The reset function is enabled when an active low is placed on the  $\overline{RS}$  pin. The data bus is tristated. The PC and the address bus are then cleared. The  $\overline{RS}$  pin also disables the interrupt.

### 1.12 Clock and Oscillator

The TMS32010 can use either its internal oscillator or an external frequency source for a clock. Use of the internal oscillator is achieved by connecting a crystal. The frequency of CLKOUT and the cycle time of the TMS32010 is one-fourth of the crystal fundamental frequency. An external frequency source can be used by injecting the frequency directly into CLKIN.

### 1.13 Pin Descriptions

Definitions of the TMS32010 pin assignments and descriptions of the function of each pin are presented in Table 5-1.

Table 5-1 TMS32010 Pin Descriptions

SIGNAL	PIN	I/O	DESCRIPTION
Vcc	30		<u>POWER SUPPLIES</u> Supply voltage (+ 5V )
Vss	10		Ground reference
X2/CLKIN	8	IN	<u>CLOCKS</u> Crystal input pin for internal oscillator (X2). Also input pin for external oscillator (CLKIN).
X1	7	IN	Crystal input pin for internal oscillator
CLKOUT	6	OUT	Clock output signal. The frequency of CLKOUT is one-fourth of the oscillator input (external oscillator) or crystal frequency (internal oscillator).
$\overline{WE}$	31	OUT	<u>CONTROL</u> Write Enable. When active (low), $\overline{WE}$ indicates that valid output data from the TMS32010 is available on the bus.
$\overline{DEN}$	32	OUT	Data Enable. When active (low), $\overline{DEN}$ indicates that the TMS32010 is accepting data from the data bus.

Table 5-1 TMS32010 Pin Descriptions (continued)

SIGNAL	PIN	I/O	DESCRIPTION
$\overline{\text{MEN}}$	33	OUT	Memory Enable. $\overline{\text{MEN}}$ will be active low on every machine cycle except when $\overline{\text{WE}}$ and $\overline{\text{DEN}}$ are active.
$\overline{\text{RS}}$	4	IN	<u>INTERRUPTS</u> Reset. When an active low is placed on the $\overline{\text{RS}}$ pin for a minimum of five clock cycles, $\overline{\text{DEN}}$ , $\overline{\text{WE}}$ , and $\overline{\text{MEN}}$ are forced high, and the data bus (D15 through D0) is tristated. The program counter (PC) and the address bus (A11 through A0) are then synchronously cleared.
$\overline{\text{INT}}$	5	IN	Interrupt. The interrupt signal is generated by applying a negative going edge to the $\overline{\text{INT}}$ pin.
$\overline{\text{BIO}}$	9	IN	I/O Branch Control. If $\overline{\text{BIO}}$ is active (low) upon execution of the BIOZ instruction, the device will branch to the address specified by the instruction.
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0	18 17 16 15 14 13 12 11 19 20 21 22 23 24 25 26	I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O	<u>BIDIRECTIONAL DATA BUS</u> D15 (MSB) through D0 (LSB). The data bus is always in the high impedance state except when $\overline{\text{WE}}$ is active (low).

Table 5-1 TMS32010 Pin Descriptions (continued)

SIGNAL	PIN	I/O	DESCRIPTION
			<u>PROGRAM MEMORY ADDRESS BUS AND PORT ADDRESS BUS</u>
A11	27	OUT	Program memory A11 (MSB) through AO (LSB) and port addresses PA2 (MSB) through PA0 (LSB). Addresses A11 through AO are always active and never go to high impedance.
A10	28	OUT	
A9	29	OUT	
A8	34	OUT	
A7	35	OUT	
A6	36	OUT	
A5	37	OUT	
A4	38	OUT	
A3	39	OUT	
A2/PA2	40	OUT	
A1/PA1	1	OUT	
A0/PA0	2	OUT	

## 2. Instructions

The TMS32010's comprehensive instruction set supports both numeric-intensive operations, such as signal processing and general-purpose operations, such as high-speed control. The instruction set consists primarily of single-cycle single-word instructions, permitting execution rates of up to five million instructions per second. Only infrequently used branch and I/O instructions are multicycle. The TMS32010 also contains a number of instructions that shift data as part of an arithmetic operation. These all execute in a single cycle and are very useful for scaling data in parallel with other operations. The instructions summary is listed in appendix E.

The instruction set contains a full set of branch instructions. Combined with the Boolean operations and shifters, these instructions permit the bit manipulation and bit test capability needed for high-speed control operations.

The TMS32010's hardware multiplier allows the MPY instruction to be executed in a single cycle. Two special instructions, TBLR (table read) and TBLW (table write), allow crossover between data memory and program memory. The TBLR instruction transfers words stored in program memory to the data RAM. This eliminates the need for a coefficient ROM separate from the program ROM.

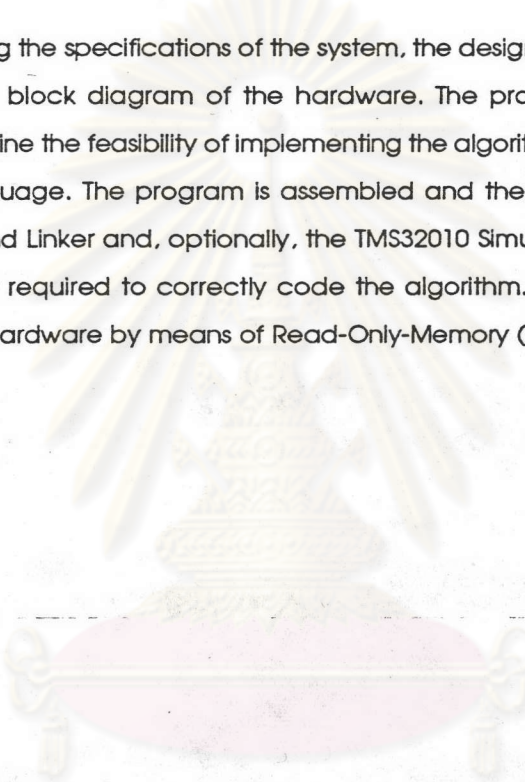
When a very large amount of external data must be addressed necessary to address external data RAM or peripheral by using the IN and OUT instructions; these instructions permit a

data word to be read into the data RAM in only two cycles. This procedure requires a minimal amount of external logic and permits the accessing of almost unlimited amounts of data RAM.

### 3. Methodology for Application Development

A number of development tools are required for designing a system with a microprocessor. This section describes the facilities which are available for the TMS32010 in developing an application. A typical application development flowchart is shown in Fig. 5-4.

After defining the specifications of the system, the designer should draw a flowchart of the software and a block diagram of the hardware. The processor's performance is then evaluated to determine the feasibility of implementing the algorithm. The full algorithm is coded using assembly language. The program is assembled and then verified using the TMS32010 Macro Assembler and Linker and, optionally, the TMS32010 Simulator. Several iterations of the program are usually required to correctly code the algorithm. The verified program is then integrated into the hardware by means of Read-Only-Memory (ROM).



ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

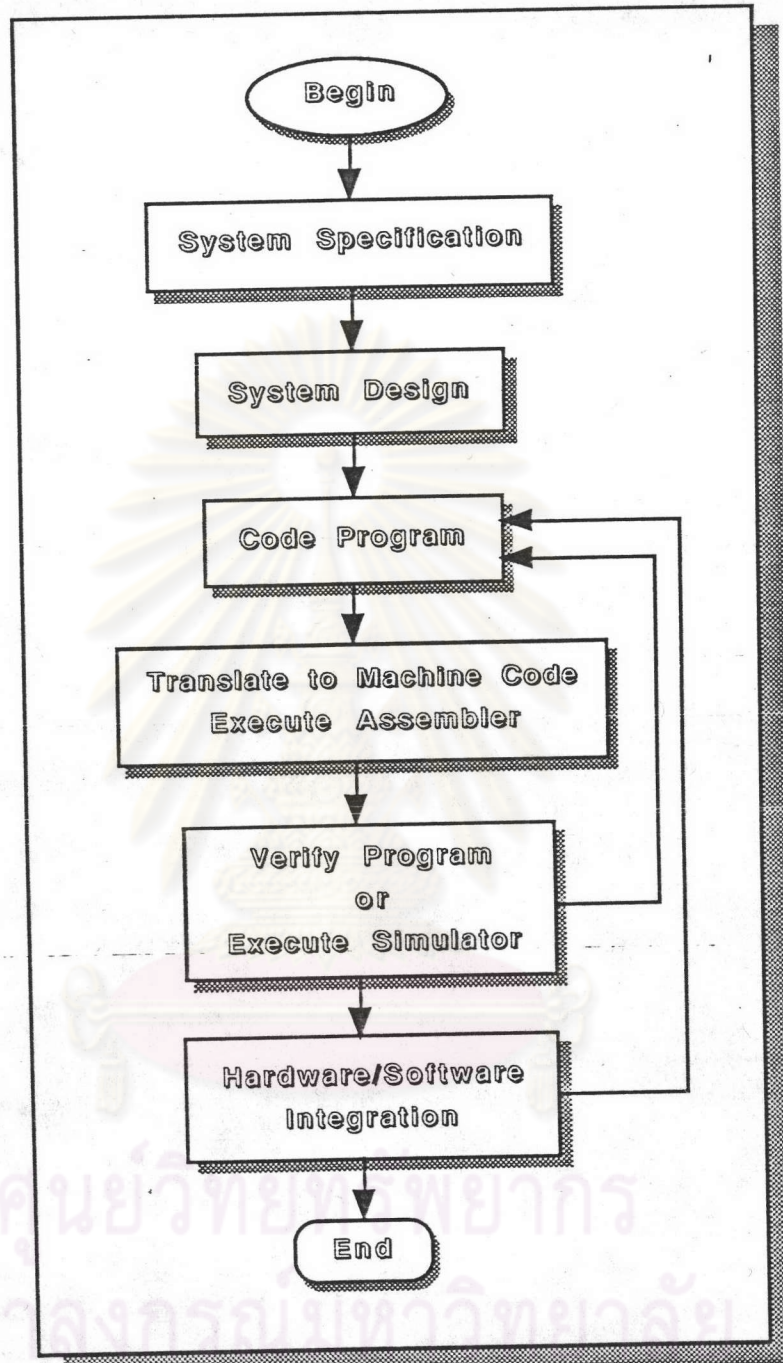


Fig. 5-4 Flowchart of application development.