HPCSS Jan 77

THE USES OF A MICROPROCESSOR IN AN INTERACTIVE COMPUTING SYSTEM

Charles A. Grant
Mark L. Greenberg
Department of Electrical Engineering
and Computer Science
University of California, Berkeley

Abstract

The intent of this paper is to make the observation that microprocessors have reached the performance level where it is reasonable to use one microprocessor to perform many functions in an interactive computing system. Our work will discuss the techniques for "multi-microprogramming" a single microprocessor to allow it to perform many logically distinct, simultaneous functions. An example will be given of a time-sharing system whose hardware components are modems, primary memory, auxiliary storage and a single microprocessor.

1. INTRODUCTION

Microprogrammable processors are well suited to a number of tasks relevant to the operation of an interactive computing system. In the past, a microprocessor was included in a computing system to perform a single dedicated function. The reason for using microprocessors is to replace complex specially designed hardware with simple and flexible firmware. Some of the tasks for which it is reasonable to utilize microprocessors are terminal input-output multiplexing, input-output device controlling, data transfer, communications, real-time data gathering, scheduling of processes and central processing unit instruction emulation.

Microprocessors currently available on the market have improved in speed, size and interfacing flexibility to the point where it is feasible to consolidate many microprocessor functions into a single microprocessor, thereby significantly reducing the total hardware necessary for a complete computing system. We proceed to discuss a microprocessor program organization to accomplish this. The final section will give an example which applies the technique to a specific problem.

2. MICROPROCESSOR LOGICAL ORGANIZATION

The microcode of a microprocessor can be divided into logically distinct functional units. Each functional unit performs a different task asynchronously with respect to all the other functional units. The functional units share the microprocessor one at a time. There exists a separate body of microcode called the scheduler which is responsible for transferring control among the functional units. A functional unit once entered, runs to completion and then passes control back to the scheduler. Associated with each functional unit is an attention flag. The scheduler transfers control to a functional unit only if its attention flag is set. When the functional unit is entered, the attention flag is cleared. An attention flag may be set by the action of any functional unit or by the occurance of some event external to the microprocessor.

3. SCHEDULER

It is the job of the scheduler to select the next functional unit to run. If more than one attention flag is set then the scheduler must use some algorithm to determine which functional unit to run first. This algorithm must be simple since the scheduler is called often and its overhead must be small. A fixed priority algorithm is both simple and quite powerful. Each functional unit is assigned a fixed priority. Of the functional units with attention flags set, the scheduler selects the functional unit of highest priority.

A fixed priority scheduler can be implemented by scanning the attention flags in fixed order until one is found which is set. In some microprocessors such a scheme can be implemented with one micro-instruction per attention flag. Since high priority func-

tional units will in general be called more often than low priority functional units, the average scan time will be less than half the maximum. If the number of functional units is too large, then scanning will take too long and it might be desirable to have a piece of hardware to determine the next functional unit to run. A microprocessor instruction which locates the leading one bit in a register would make this scheduler a two instruction routine.

4. FUNCTIONAL UNIT RESOURCE REQUIREMENTS

Each functional unit requires microprocessor resources of two types a) time on the microprocessor b) microprocessor working storage, e.g., high speed registers. We will develop criteria for satisfying these resource requirements.

Time. The timing environment of the functional unit with priority i can be characterized by three values:

(1) Te(i) - the maximum time functional unit i executes. This includes scheduling time. Thus Te(i) is the worst case time between entrys to scheduler for functional unit i.

(2) Ta(i) - maximum acceptable time between raising of the attention flag and entry to the functional unit.

(3) Tw(i) - minimum time between attention flag settings for the functional unit.

The priorities of the various functional units must be assigned in such a way that under worst conditions every functional unit i will be called within time Ta(i) after setting its attention flag. Formally stated:

Criterion 1: For every i: $Ta(i+1) > \sum_{j=1}^{i} Te(j) * \frac{Ta(i+1)}{Tw(j)}$

Furthermore we must insure that no attention flag is set which was not previously reset. This is satisfied if:

Criterion 2: For every i: $Tw(i) > Ta(i)+Te(i)$

It may be true that for a given set of functional units there exists no assignment of priorities which satisfy the two criteria. It may be possible to meet the criteria by dividing a functional unit into two or more functional units. The first sub-unit would set the attention flag for the second before it returns control to the scheduler. An example of this will be given later.

Space. Microprocessor working storage is generally allocated in fixed size units called words. Words of storage may be implemented in one of several kinds of hardware. For example, a word may be a high speed register, a medium speed scratch-pad register or a word of core memory. Working storage in microprocessor systems, particularly high speed resisters, is limited. Thus it is desirable to optimize its use.

All words (registers) with identical performance characteristics are said to belong to the same class. Thus, all high speed registers might be a class. If the sum of the registers in a given class required by all functional units exceeds the available number of registers, then it becomes necessary to allocate registers to more than one functional unit. In this case it must be insured that the use of a register by one functional unit does not conflict with the use of the register by some other functional unit.

A register may be used by a functional unit in one of four ways:

(1) Temporary Register - Used to hold temporary results during the execution of a single functional unit.

(2) Dedicated Register - Used to hold values for a future execution of the same functional unit. If a register is used as a dedicated register then the register may not be used as any other type of register.

(3) Passing Register - Used to pass values from the current functional unit to another functional unit. When a value is to be passed to another functional unit(s) then the attention flag of that functional unit must be set by the current functional unit.

(4) Receiving Register - The register is used by the current functional unit to receive a value from some other functional unit.

To avoid conflicting use of a register the following criterion must be satisfied:

Criterion 3: For every register x, a functional unit using x as a receiving register must have higher priority than any other functional unit using x as a temporary or passing register.

This criterion insures that when a value is being passed in a register, no functional unit will use the register until the value has been received.

A microprocessor program organization with interruptable functional units was rejected because:

(1) Registers to hold return links to intertupted functional units are expensive.

(2) In general more working storage would be required.

(3) The nature of the functions under consideration does not require immediate response even for high priority functions.

The following topics are not discussed in this paper due to lack of space.

(1) What can be done about the heavy I/O interfacing register requirements of a microprocessor.

(2) What are important capabilities for microprocessor instructions to have.

(3) Details of microprogramming techniques to implement specific types of functional units.

5. EXAMPLE

A simplified example of the applications of a multi-microprogrammed microprocessor will now be given. All the numbers in this section are contrived for the purposes of the example. The microprocessor is assumed to have characteristics approximating:

a) 150 ns instruction cycle time
b) 2K microcode memory words
c) 24 high-speed registers
d) A reasonable micro-instruction set

This microprocessor plus the following listed items will be the only hardware required by a hypothetical 64-user dedicated time-sharing system giving 2 second response to trivial requests:

a) 32 K words of 1 sec primary memory interfaced directly to microprocessor I/O registers.
b) 64 terminals each of whose serial data lines are latched to a single bit within microprocessor I/O registers.
c) Secondary storage device (disk or drum) with a transfer rate of 1 word every 12 μsec. Control lines from the device are latched directly to a microprocessor I/O register. Data lines are routed through a serial-parallel/parallel-serial converter to an I/O register.

The microcode is initially partitioned into four functional units:

(1) CPU. This functional unit emulates the order codes and addressing structure of the machine presented to the user. This functional unit can always run since there are always instructions to execute. Its function is to execute the instruction in memory designated by the p-counter and then return to the scheduler. The CPU is the functional unit with lowest priority. The quantities Ta and Tw do not apply to the CPU.

(2) Disk Transfer Unit. This unit is responsible for transfering words of data between the disk and primary memory. The disk hardware must raise the attention flag for this functional unit when it has another word read from the disk or is ready for another word to write on the disk. The unit will count the number of words transferred and then notify the disk control functional unit when the transfer is complete by raising its attention flag.

(3) Disk Control Unit. This functional unit is responsible for starting a data transfer and handling disk error conditions. Its attention flag is set by the disk transfer unit when a transfer is completed and by the hardware when a disk arm move is completed or when a sector gap is reached. The unit issues commands to the hardware to start arm movements and data transfers. It also senses the state of the disk and indicates error conditions to the CPU.

(4) Terminal Multiplexor. This unit is responsible for sending and receiving bits of information from the terminals connected to the system. The unit performs bit serial to parallel conversion and manages a character buffer for each terminal. The attention flag is set by an external clock often enough to properly sample the input lines.

Table 1 shows the timing and high speed register requirements of each functional unit. The units of time are microseconds

It is clear that this organization violates the previously given timing criteria. Te(CPU) is 20μsec. because there are some instructions (e.g., multiply and divide) which may take that long. This may be solved by creating new functional units for each instruction that will take more than, say 5 sec. Then whenever a long instruction is decoded by the CPU functional unit, the attention flag for the appropriate new functional unit will be raised and control will be transferred to the scheduler. Each of these new functional units will be further subdivided so that they will never execute too long without returning control to the scheduler. The multiplexor may also be modularized in a very natural way (scanning only a subset of the lines in each sub functional unit.)

The above modifications have solved the timing problems. To allocate registers properly, we must share the temporary registers used by the CPU and multiplexor. This can be done by guaranteeing that the CPU and multiplexor (and their functional units) never intersperse execution. The modified organization is shown in Table 2. A priority ordering which satisfies the three criteria is transfer unit, control unit, multiplexor 2, multiplexor 3, multiplexor 4, multiply 1, multiply 2, multiplexor 1, CPU.

CONCLUSION

This short paper has demonstrated the feasibility of using a single microprocessor to perform several sophisticated tasks. The authors would be pleased to consult on the subject in more detail with those interested in applying these techniques.

TABLE 1

	Te	Ta	Tw	Dedicated Registers	Temp. Reg.	Passing Reg.	Receiving Reg.
CPU	20	-	-	15	6	0	0
Multiplexor	35	250	1000	0	6	0	0
Transfer Unit	2	10	12	2	0	0	0
Control Unit	4	30	12000	1	3	0	0

TABLE 2

	Te	Ta	Tw	Dedicated Registers	Temp. Reg.	Passing Reg.	Receiving Reg.
CPU	5			15	0	6	0
Multiply 1	8				0	0	6
Multiply 2	8					0	6
Multiplexor 1	9	190	1000	2		6	0
Multiplexor 2	9	20	1000			0	6
Multiplexor 3	9	20	1000			0	6
Multiplexor 4	9	20	1000			0	6
Transfer Unit	2	10	12	2	0	0	0
Control Unit	4	30	12000	0	3	0	0