Journal of Advances and Scholarly Researches in Allied Education
Vol. 14, Issue No. 2, January-2018, ISSN 2230-7540

An Analytical Approach for New Results on Advancement of Applications of Multi-Microprocessors

Mrs. Yamini Balasaheb Pawar[1]* Dr. Avnish Kumar Shukla[2]

[1] Research Scholar, Maharishi University of Information Technology, Lucknow, Uttar Pradesh

[2] Department of Electronics and Communication Engineering, Maharishi University of Information Technology, Lucknow, Uttar Pradesh

Abstract – Some of the most quickly growing fields for computer activity is in embedded devices, whose key role is not computation but which do require knowledge to be implemented. For several embedded device areas developments for hardware technologies rendered several microprocessors a feasible option to uniprocessor systems.

Key Words: Multi-Microprocessors, Applications, Approach

----------X----------

INTRODUCTION

A pipelined computer performs overlapped computations to exploit temporal parallel ism. An array processor uses multiple synchronized arithmetic logic units to achieve spatial parallelism. Multiprocessor systems achieve asynchronous parallelism through a set of interactive processors with shared resources (memories, data bases etc.).The fundamental difference between the array processor and the multiprocessor system is that the dealing out elements in an array processor operate synchronously while processors in a multiprocessor system operate asynchronously.[1]

A pipeline has several dealing out stages, each stage repeatedly executes the same instruction on successive pieces of data received from a preceding processor; thus the result of one processor becomes the input of the next. This according to Flynn's groupification of architectures, is Multiple Instruction stream, Single Data stream (MISD). [FlynnM.J., 1966, 1972). The pipelining technique is for applications in which the major functions are appropriate dependent on each other and the data sets are very large, for eg. Signal dealing out of real time data.[2]

Through the definition of Flynn, the Single Information Stream (SIS0) may be referred to as a traditional central processing device running on one collection of data. This category involves serial computers where commands can be executed sequentially, but overlapped during their running times (pipelines of instruction level).

Multiple processors in arrays execute the same operation, each in a disjoint collection of data for the inclusion of ego vectors, under unified supervision. Multi-Data Stream (SIMD) classified as single-instruction tube. Different screen processors are computers for systems in which normal devices run data structures such as arrays. Those are mainly useful for systems that have a large number of vectors[3].

Every processor can execute a particular series of instructions on a specific collection of data on multiprocessor systems. It is named Multiple Instruction distribution (MIMD). These asynchronous processors are more suitable for consequent transmission, consisting of a series of collective or interaction-communicating processes[8-10].

The computer network, where every 1S processor is inserted into a traditional computer device, and the computers are linked by communication links, is a further category that is considered to be classified under MIMD. The degree of coupling between processors in the machine can differ along a single dimension from networks, multiprocessors and array computers. Patnaik describes this as total access time to a global data system in 1983 for the worst case processors. Any machine is in a multi-processor. Patnaik L.M., processor with direct access in pr1mary memory to global info. [4]Since the communicating between processors is achieved through primary memory sharing, time contact is quicker. Global data in one machine can have to be reached in the computer networks by means of a hop series with other computers. This takes more

time. In array computers, the analog of inter processor communication is the transfer of control information that occurs between the control unit and its associated dealing out elements. The average time between inter process interaction becomes a crucial time constant of an application and provides a good indication of the type of multiple processor organization that will be most suitable.[5-7]

RESEARCH METHODOLOGY

In conventional DBMs, there is the data communication subsystem, which is the terminal handler that receives user requests, invokes application program and delivers the results that it receives from the database transaction manager. An operating system exists to run these software features. The method banks on manipulating the size of the block that is accessed from the disc.

In this approach, disc sectors are grouped to form segments; the smallest made up of two sectors and called the Base Segment (BS). This segment size is programmable to the extent that one out of the several Exponential Segments (ES) can be selected. These are made up of 4,8,16 ... sectors as a trade off, forming respectively ES2, ES3, ES4 ... segments. Depending on the situation, the OS selects either BS or one of the ESs to form a single block of storage. Fig. 1 presents the flowchart for the related software.

The space allocation is handled by software with the aid of a segment availability table (SAT), which specifies the track number and starting sector number of every base segment –BS address-indirectly; in the sense that a free BS will be represented by a 0 and a used up BS by a 1. The physical positions of the Os and 1s in the SAT, read on a dedicated area of memory, will be manipulated by software to determine the corresponding BS address. The address of free BSs will be stacked on a memory block, the size of which can be limited to hold say 256 bytes, so that at any given time 128 SS addresses are available if free. If number of free SSs is less, the stack size is also small, thereby reducing memory requirement. Fig.2 shows the flowchart for creating the BS stack.

In the new format presented, codes form decisive groups based on the number of operand bytes involved in the particular instruction. Group 0 instructions have no operands, its function being that of a controller; while group 1 will have a single operand. A mention of two operands which are exclusively data bytes, like the addend and augend in an ADD operation, constitute group 2. If mention is made of a two byte address, it is group 3. Thus, in a sample case of an 8- bit instruction and 16-b1t address, two MS bits are reserved to demarcate one of the four groups branded.

Next, we propose to specify the location of opcodes in the format. This warrants a scrutiny of the register organization as well. Adverting to the concept of a sma11 register file, which would help to have only a few simple instructions, an organization is arrived at based on analysis of various register profiles. The analysis has been conducted by assigning Register Credit Values (RCV) as given out in Table 2. The registers selected for the new organization are an ACe, an IX, an SP and a position register, apart from the PC, Which is indispensable in the Van Neumann scheme. It may be noted that all memory architecture would have about 47% of data references for temporary storage of intermediate results.

RESULT AND DISCUSSION

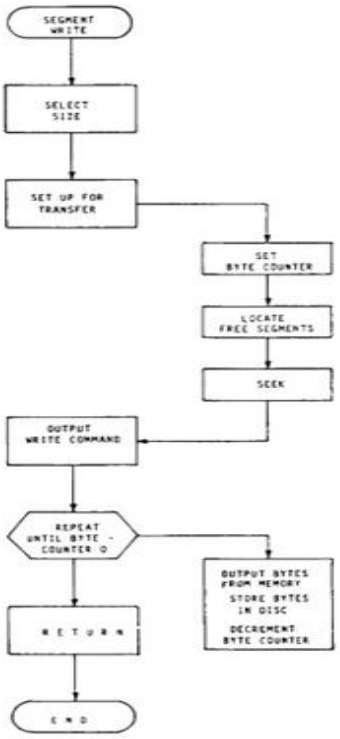

Fig 1 .disc operation flow chart

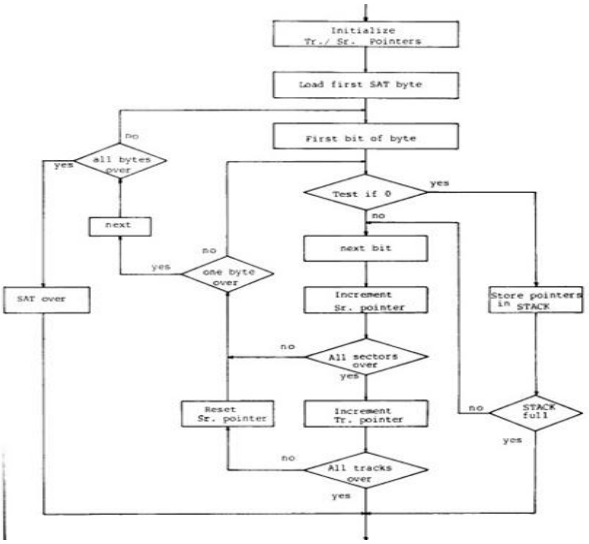

Fig 2. Creating BS stack

The decision regarding BS or ES will select the appropriate segments for storage. A BS will select one address; an ES2 will select four addresses, ES3 eight and ES4 sixteen addresses from the SS stack, forming corresponding single blocks. Before selecting the addresses for the next block, the control software will skip an equal number of addresses in the SS stack, thereby scattering the storage blocks and granting respite to the software for completing housekeeping tasks.

During deletion, the appropriate SAT bits are reset to denote a free position. The free space can then be allocated on demand, either in part or in one block. This will minimize the occurrence of unused free spaces in a "full" disk, thus reducing the cost per bit of storage.

The scheme presented is advantageous owing to the following reasons:

i. The waste of space due to some blocks remaining partly filled is reduced.

ii. The waste of space due to chain pointers is reduced Small blocks would require more pointers.

The system is adaptable to environments calling &or 'large files only', simply by enlarging the size of the base segment. This will reduce the size of the SAT as also limit the file name appendix, providing more space to the user and reducing housekeeping time. At the other end, the base segment can be retained to have the size of one sector, and selection of block size can be from among multiples of base segment.

Block transfer and DHA

Controllers for block devices can conveniently make use of DMA concepts. Here the CPU gives two items of information, in addition to the block address the memory address associated and the number of bytes to transfer. After the controller has read the entire block from the device into its buffer and verified the checksum, it copies the first byte into the main memory, at the specified DMA memory address. Then it increments the DMA address and decrements the DMA count. The process is repeated until the DMA count becomes zero, at which time the controller causes an interrupt.

The Controller described in Chapter 4 will be overtly useful for vectored transfers from the disk. This is particularly significant in the light of the demarcated primary storage, On line data/program can be transferred over to the appropriate blocks, by the OS which will augment the file name/identity of the data block in the disk with the appropriate destination memory address. These addresses can directly go to the DMAC stacks, which will maneuvers the transfer.

Transfer efficiency

In the environment of transfer discussed, we define transfer efficiency as the ratio of the time involved in the actual operative part of the job, to the total time which comprises the supportive part as well. Accordingly, we have

$$T_{io} = T_{rw} + T_{hk}$$

Where T_{io} represents the total time involved in the transfer operation T_{rw} represents the actual time of read/write operation and T_{hk} represents the time taken for associated housekeeping.

Table 1. gives quantitative results which indicate that the new layout cuts down T_{hk} This ;s because, the number of seek operations is reduced as a result of the predefined block size; the drive head need not have to go each and every time to the housekeeping track to locate free sectors. Again, since the segments themselves are scattered, there are instances when one full revolution time is eliminated. Consequently a high efficiency of transfer can be achieved.

Table 1. T_{hk} for the two schemes

Record size in bytes	Thk for fixed block size (ms.)	Thk for pro. block size (ms.)
256	65	65
512	140	90
768	210	140
1024	290	210
1280	380	280

In the 8-bit instruction word example, the next two MS bits specify the register associated with the operation. The remaining bits characterise the logic. Table 8.2 depicts a few bit patterns in the new format. The use of consistent opcode field, a RIse phenomenon, allow rapid decode and swift adaptation. Decoding of the instruction proceeds by identifying the group to which it belongs and then looking for operand specification as well as opcodes. The operand location is also well defined with respect to the groups. For a group 1 instruction, it lies invariably in the ACe. For group 2, one operand lies in the ACe or other register specified by bits b5 and b4, and the second lies in a memory location dedicated for the purpose, layoutated "MEMOP" Memory Operand. It would be desirable to locate MEMOP next to the stack area, so that whenever stack is initialized this address is set, which is pointed to by the SP of a virgin stack.

In group 3, the two-byte address, obviously may be held by MEMOP together with its adjacent higher address location.

Table 2.code format

Addressing is just direct or indexed, again a RISe feature namely few addressing modes. The excessive use of indexed computed mode of addressing, though deviates from the RISe scheme, is advocated as it enables shifting of routines to a different area of memory, by altering the initialization part. This is particularly significant in the light of the partitioned resource memory, it also facilitates an excellent memory expansion technique using indexed mapping.

Adaptation

The well defined format presented, enables swift identification of the different attributes in an instruction. With few addressing modes and the prescribed operand locations, a simple pre-dealing out would enable migration of programs to new environments. Conversely customized instructions can be rebuilt after identifying opcodes and determining operands, by resorting to a table look-up or other standard cross assembler procedures. The sequence proceeds by first fixing the group based on the number of operands involved and assigning the respective bits. The operands are then put in the appropriate locations specified.

The opcode field is checked and the appropriate code/codes generated. As a matter of fact, additional codes may be required to accomplish the operation.

Implementation with fewer instruction types obviously requires additional instruction memory traffic. This is alleviated by providing a small instruction cache as an on chip buffer. This would be kept full by a prefetch logic.

Smart Assistance

Rapid progress in semiconductor process technology enables the increasing integration of systems on chip. Functions previously exclusive to minicomputers or large scale machines are now appearing in microprocessors. This latter motivation led to suggest an on-chip Smart Dealing out Support (SPS) which would promote a high level of system reliability. Two significant aspects of an intelligent assistant have been branded:

- It should provide an insight into the process.
- It should be a participant in the dealing out by providing suitable instructions as and when required.

The insight is provided by the MF structure proposed in the previous chapter. The SPS looks for the MF and off loads its content to a dedicated buffer provided on chip. This will serve as a ready reference as regards the functional role of various entities branded.

Implementing a small set of instruction would result in a substantial saving in control logic. Also, it reduces the range of probable instructions in a stream. The saved area on chip can be used to hold storage facility, which in turn is used to implement the SPS. The SPS participates in the dealing out and strives to provide suitable instructions as and when required.

Observation

Church has given the probability of occurrence of each different group of instructions as shown in Table 3. As a preliminary attempt, only the first two groups which together constitute almost 7S % of all programs, are subjected to scrutiny.

It is often the case that when an instruction is executed,

Table 3.

Functions	Rate of occurrence
Data transfers (LOAD, STORE, MOVE)	45 %
Program Control (BRANCH, CALL, RETURN)	29 %
Arithmetic	10 %
Compare	6 %
Logical	4 %
Shift / Rotate	3 %
Bit Manipulation	2 %
Input/Output and Control	1 %

Not all possible instructions are equally likely to follow. For example, a LOAD ACC instruction is not followed by a STORE ACe with reference to the same address. Taking advantage of these facts, opcodes are grouped into clusters, where the members of a cluster are likely to follow one another. In another instance, an increment instruction would never be followed by a decrement instruction. But when it occurs in a counter routine which makes use of a memory location, then it may be followed by a store instruction. Another code might compare the value of the counter to decide on a branch operation. The SPS foresees a branch code in such a situation and verifies the appropriateness of the instruction in the stream.

A representative case of comprehension by the SPS might occur like this: an instruction refers to a memory location which is specified as a counter flag in the MF structure. The likely sequence would be

LOAD ACC

INCR/DECR

STORE ACC

BRANCH CN

REFERENCES

1. Athas, Wi 11 iam C., Seitz, Charles L. (1988). "Multicomputers : Message Passing Conrecent Computers", Computer.

2. Basu A. (1987). "Parallel Dealing out Systems: a nomenclature based on their characteristics", Proceedings lEE (UK).

3. Belady L.A. (1987). "Evolved Software for the 80's", Computer, Feb., 1987.

4. Ben-Ari M. (1982). "Principles of Conrecent Programming", Prent ice-Ha 11 NJ.

5. Bi tar 1.,Penedo M.H., Stuckle E.O. (1985). "Lessons Learned in Building the TRW Software Productivity System", Proceedings COMPCON San Francisco CA., Spring.

6. Boehm, Barry W. (1981). "Software Engineering Economics", Englewood Cliffs, New Jersey; Prentice-Ha11.

7. Bora 1 H., Dew it t D. J. (1983). "Data Base Machines : An Idea Whose Time Has Passed? A Critique of the Future of Data Base Machines", Data Base Machines, Springer-Verlag.

8. Bourne S.R. (1980). "The UNIX System", Wokingham Addison-Wesley.

9. Bowen B.A., Buhr R.J.A. (1980). "The Logical Layout of Multiple Microprocessor Systems", Prent; ce-Ha 11 ,EnglewoodCl; ffs, NJ, 1980.

10. Burns A. (1988). "Progranvning in Occam 2", Wokingham: Add; son Wesley.

Corresponding Author

Mrs. Yamini Balasaheb Pawar*

Research Scholar, Maharishi University of Information Technology, Lucknow, Uttar Pradesh