North-Holland
Microprocessing and Microprogramming 24 (1988) 695–700

ISSUES IN CPU-COPROCESSOR COMMUNICATION AND SYNCHRONIZATION

Vojin G. Oklobdzija

IBM T.J.Watson Research Center
P.O.Box 218
Yorktown Heights, NY 10598
U.S.A.
(914) 945-2607

ABSTRACT

This paper deals with the issues in microprocessor fixed and floating-point processors communication and synchronization. In a micro-system consisting of a Fixed Point processor, and Floating-Point coprocessor, it is essential that the two units work at their maximum utilization rate. Three different environments are examined in terms of number of chips. They can consist of each of the units being a separate chip, or two of the units, or all of them being integrated on a single chip. In terms of coupling of the two (or more) processor they can execute their instructions in a lockstep, or run as a complex of independent units. The synchronization strategies are examined in this paper and two approaches for achieving higher performance are proposed.

INTRODUCTION

Improvement in performance and power of the micro-systems has brought increasing demand for the numerically intensive floating-point computation since a number of problem previously residing only on the large system has moved into the computational domain of todays micro and mini computers. With that proliferation of use, floating-point processor (FLP) have been found added or integrated into the micro-system in a variety of different ways. They can range from the simple addition as an afterthought to a well conceived and engineered floating-point processor as an integral part of the micro-system. The addition of a floating-point processor can bring mixed results in terms of the cumulative system performance because in a code with sufficient mix of the fixed-point and the floating-point instructions the overall system performance will be determined by the performance of the slower of the two processors. However, it is not only the mismatch of the speed of the processors that affects the overall performance of the system. This will be further diminished by the time spent communicating and exchanging data and instructions between the fixed-point processor (FXP) and the floating-point processor (FXP). Therefore, the issues in dispatching instructions and data are of very much importance in achieving a well balanced system.

In this paper three different environments are discussed:

1. Fixed and Floating Point processors being two separate custom chips where the communication protocol is controlled by the fixed point.
2. Fixed and Floating-Point processors being two separate chips receiving instructions from a separate instruction unit chip which is added with the purpose of coordinating the two units. Additionally the instruction unit can perform mapping of the architected floating point instructions into the instruction set of a selected floating-point processor [5].
3. Fixed and Floating point processors contained on a single chip. The communication and synchronization of the two is handled by the instruction unit and instruction dispatching mechanism which is a part of the same chip taking advantage of the internal registers and busses, and eliminating off-chip crossing penalties.

Synchronization and Out of Sequence Execution

Ideally both processors will be running concurrently executing their instructions in parallel. In addition they will execute their instructions as they arrive from the instruction queues, therefore allowing out of sequence execution. Under this condition the maximal throughput would be achieved and the performance of the system is determined by the individual performance of the processors and the instruction mix.

However, branches in the program and the data dependencies between FXP and FLP units do not make this conditions possible. In addition the amount of concurrency and parallelism is determined by the implementation of the processors and the available hardware resources. In order to assure consistency of the operations FXP and FLP processors need to be synchronized (at the same point in the program flow) under the following conditions:

- branches in the program (conditional and unconditional)

- exchange of data (either through the memory or exchange of register contents)
- interrupts and procedure calls

In case of branches the need for synchronization is obvious and not to be discussed. If the branch instruction is conditional, involving the condition code generated by the FPU, than synchronization must occur before that condition (condition code register) is transferred from the FLP unit to the unit which makes the decision based on this condition (FXP unit, or Instruction Unit - IU).

Any memory access is a reason for synchronization. If data is to be written to memory it must be assured that the data at the location to be overwritten is no longer needed by the other unit (FXP or FPU). Similarly, if there is an access to memory for data, this memory location must contain the data that represents the status at this point in the program. In other words, if data in this location is a result of an instruction currently executed by the other processor, than this instruction must be completed. Also if the data is already available, the other processor should not be allowed to proceed beyond this point if there is possibility that the execution of the next instruction might alter the data.

As far as interrupts and procedure calls are concerned the processors must achieve synchronization before the status of the process is saved. This might increase the interrupt response time, which is a drawback in this case.

When an exception is detected the status of the processors must be saved in the way which makes it possible to restart the process and indicate the check point precisely in the program. Precise indication of the check-point in the program might be difficult to achieve if out of sequence execution of instruction takes place.

Achieving high degree of concurrent instruction execution could be facilitated by the compiler or passing the code through the scheduler which tries to order instructions in such a way that as much parallelism between the units is made possible.

For the purpose of synchronization special instruction can be created.

Two Separate Chips

Most floating point processors have been developed as an enhancement of the existing microprocessors. Such as: Motorola 68030-68881/6882, Intel 80386-80387/80287, and older National 32032-32081, Intel 8088-8087 etc. Since the central processor has been developed first, there was not much flexibility left since the floating-point processor needed to fit into the set of exiting control signals and established bus protocol.

In this environment the communication protocol is distinguished by three factors:

- the way instruction is passed to the FLP
- the way FLP instructions are added to the existing instruction set
- the amount of freedom and parallelism between the two units

Case 1.

Let us examine the protocol used in INTEL 8088/8087 [9], Fig.1., with respect to those three factors.

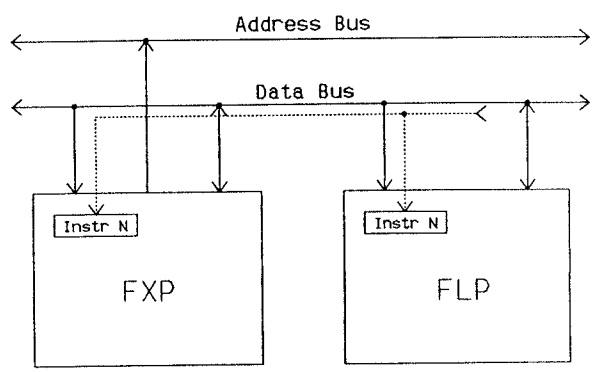

Figure 1. FXP and FLP running in lockstep (Intel 8088/8087)

Floating point instructions are architected as an extension to the existing instruction set through the use of ESC opcode which indicates that the rest of the instruction, as well as a following bytes should be treated as a floating point instruction. Both units monitor the bus and latch the instruction. If the instruction does not contain ESC code it is disregarded by the FPU and FXP executes the instruction. If ESC is detected, the instruction is executed by the FPU while FXP is waiting for its completion. FXP can not issue another instruction fetch while this instruction is executed. If the instruction requires memory access, the FXP takes part in address calculation.

This case illustrates the environment where two units are running in a lockstep. They are tightly coupled and if a FLP instruction requires a long time to execute, FXP is left idling. However, the existing bus is used and by simultaneously fetching the instruction there are no additional cycles spent for delivering the instruction to the FLP unit.

Case 2.

The second case is illustrated in the protocol used by 68030-68881/2 processor [7], Fig.2. The instruction is fetched by the FXP (68030) and after it is recognized as a FLP instruction it is send over the data bus to the coprocessor identified by the coprocessor identification

field (68030 can address up to 8 coprocessors). The only difference between a coprocessor bus transfer and any other is that the function code is issued to indicate the FPU address space and the FPU ID. FXP is instructed if further action is required (fetching of data), and if not it can proceed with the next instruction fetch and execution. The choice of concurrent execution of instructions is determined here on instruction-by-instruction bases. In this case it is not required that the two processors run at the same clock speed. The same protocol is used in 80386/80387 communication where 80387 is addressed as an I/O device at the address 800000F8 and 800000FC.

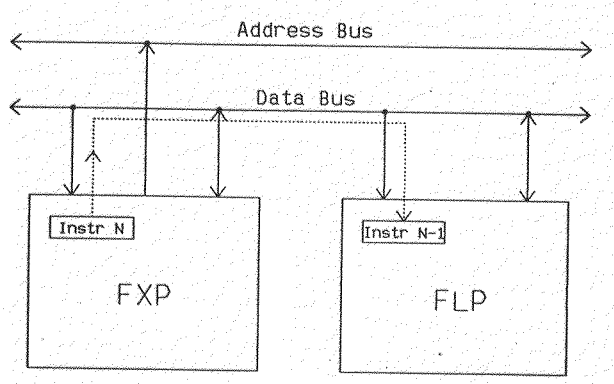

Figure 2. FLP instruction sent over the bus (Motorola 68030/68882)

This protocol has advantage of allowing two asynchronous units (with two different clock speeds), and parallelism on the level of single instruction is achieved. However, the overhead in terms of cycles needed to transfer the FLP instruction from FXP to the FLP unit can diminish the advantage when a fast FLP unit is used and the FLP instructions do not require many cycles to execute.

Case 3.

In the previous two cases the instruction that is fetched is the one to be executed next. There is no provision for buffering of the FLP instructions in the Instruction Prefetch Buffer (IPB). In the performance oriented processors it is common to find IPB to be a part of the processor. This allows instruction to be fetched simultaneously while another instruction is being executed. The existence of the IPBs in the FXP and FLP processors complicates their communication and synchronization protocol because one has to keep track which instructions are still in the IPB and which ones had already been executed. We can imagine the scenario in which FXP has executed all the instructions from its IPB, while FLP unit is still executing the first instruction from its buffer.

Proposed scheme (Fig.3.), for the environment where the instruction unit is a part of FXP processor, is to have FXP take total control over the instructions executed in both units by passing go-ahead signals to the FLP unit for each FLP instruction. This signals are received as tokens and can be buffered in the FLP processor.

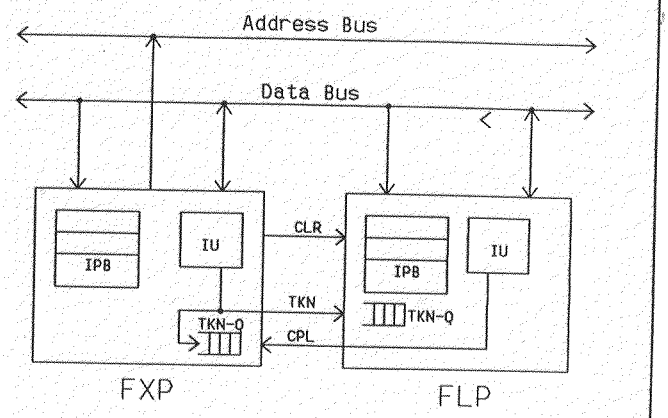

Figure 3. Synchronization sending tokens

Here is how the scheme works:

On the instruction fetch, both processors (FXP and FLP) are latching incoming instruction. This eliminates overhead in the cycles used to communicate the instruction from the FXP to the FLP (either as I/O or memory operation). It also eliminates a dedicated bus in case where pre-decoding is done by the FXP and operation code passed to the FLP. Instructions are encoded in a way which makes easy to distinguish FXP from the FLP instruction (use of one or more bits in the opcode field). Therefore incoming instruction is pre-decoded and it is written into the Instruction Prefetch Buffer (IPB) of the FLP processor only if it is FLP instruction. Therefore FLP processor IPB contains floating-point instructions only. FXP processor writes both types of instructions into its IPB. When FXP is decoding instructions, each time FLP instruction is encountered a token TKN is sent to the FLP processor indicating "approval" to the FLP to execute that instruction. When received, and if busy, FLP will queue the token on the "token queue" TKN-Q. The purpose of the tokens is to assure that the instructions scheduled for execution by the FLP processor are previously "screened" by the FXP. Therefore FLP can never execute an instruction which is at the point in the program beyond the instruction currently decoded by the FXP. For each token sent to the FLP one token is placed onto the completion queue CPL-Q in the FXP. When FLP completes an instruction a completion signal CPL is sent to the FXP which in effect removes one token from the CPL-Q. This is a mechanism of keeping track of the FLP instructions that are fetched and residing in the FLP IPB, but not yet at the decode stage.

For example, when BRANCH instruction is decoded FXP processor waits until its CPL-Q is empty (being cleared by the CPL signals). At this point both processors are synchronized and branch can take place. In addition clear signal CLR is issued to FLP with the effect of clearing FLP IPB of the instructions.
When FXP access or writes FLP data from/to the memory the same synchronization takes place except clearing of the FLP IPB. It is assumed that FXP does address calculation.

Empty CPL-Q indicates that all of the FLP instructions encountered at the FXP decode stage have been executed, thus the two processors are in synchronization with respect to the program. The amount of tokens in the CPL-Q also indicates if FLP is still executing instruction, or has more than one on the queue before reaching the point of synchronization. In case of program exceptions this information could be saved to help pointing to the exact point in the program where the exception occurred.

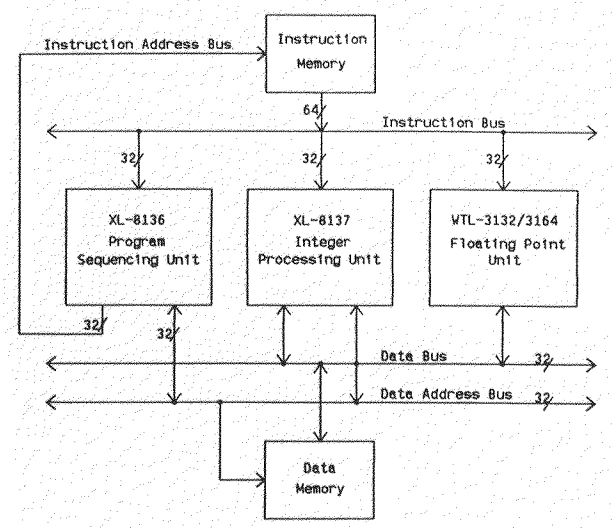

Figure 4. FXP and FLP controlled by the IU (Weitek XL-8064)

FXP and FLP Coordinated by a Separate Instruction Unit

In more recent processors instructions are dispatched from the instruction unit, which is a separate unit and not a part of the FXP unit. Commonly the processor consist of three distinct units: FXP, FLP and IU and it is conceived with the FLP as an integral part of the processor. They can be physically separate chips (Weitek XL-8064), or all three, plus additional units, are integrated on a single chip (Motorola 88100). Their working protocol is influenced by their level of integration.

Weitek XL-8064

Weitek processor consists of three physically separate chips: Sequencing Unit XL-8136 (IU), Integer Processing Unit XL-8137 (FXP), and Floating Point Data Path (FLP) which can be either WTL-3132 or WTL-3164 (32 or 64 bit wide) [6], Fig.4. The instruction bus is 64-bit wide and consist always of two instructions: the first 32-bit belonging to either IU or FXP, and the second 32 bits being FLP instruction.
Both processors run in a lockstep: both instructions are delivered in a 64-bit word at the same cycle and both processors must wait for the completion before the next instruction is executed. However, XL-8136 is allowed to concurrently fetch the next instruction while the current one is being executed, the result of which can be cancelled by the next instruction and thus instruction can be effectively neutralized.
If FLP instruction is not contained at the current cycle, a No-Op is placed in the upper 32-bits of the 64-bit instruction word. This is particularly wasteful of the memory space, especially in the case when the instruction mix does not contain a substantial portion of the floating-point code.

Motorola 88100

Motorola 88100 consists of five units: Integer Unit, Floating-Point Unit, Register File and Sequencer, Instruction Unit and Data Unit [8]. In addition two separate memory management chips 88200 are attached to the instruction and data memory space.
The processor contains four execution units which operate concurrently: in addition to integer and floating point operation, data memory access is performed by the data unit and instruction unit performs an instruction fetch. Units are pipelined and can complete an operation every clock cycle.

In Motorola 88100 data sharing and synchronization is performed by the register sequencer. The register file is common for both units and data is shared through the register file. Synchronization is performed in the register through the "scoreboard" bits associated with each register. The purpose of this bit is to control the register access and not to allow the instruction to proceed if the instruction intends to use the register which is not "cleared" (in use by another instruction). Instruction decode and the source operand fetches are performed in parallel. Branch instruction and decode, target address calculation and next instruction fetch are also performed in parallel.

This synchronization method is made possible by a common register file. This might be performance limitation if the data bandwidth provided by the register file is not sufficient to satisfy demand for data by both processors. Also instruction buffering is only one level deep and instruction cache is required (part of 88200).

Proposed Organization of the Instruction Unit (IU)

This section describes an organization of the instruction unit, which enables concurrent operation and decoding of two instructions, FXP and FLP at the time. Out of sequence execution of FXP and FLP instructions is allowed under conditions described.

The Instruction Unit, Fig.5., embodies the principles of decoupled architecture [1] . It receives instructions from the memory controller and dispatches two at the time for further decoding to the fixed and floating point units. These two instructions can be any combination of fixed and floating point instructions, including fixed (floating) loads and stores. The instructions are received by the pre-decode registers PDS1 and PDS2 and partial decoding is performed in the Pre Decode and Control Unit (PDC). The PDC unit determines if the instruction is Load, Store or Branch and if it requires synchronization between fixed and floating point units.

Normally, Load and Store (fixed and floating) require the units to be synchronized to assure that one of the processors is not using "old" data, i.e. data that would be otherwise overwritten had the processors been in sync. Given the frequency of Load and Store instructions, about 30% of the instruction mix [1], it would practically be required that the processors run in a lockstep most of the time. Other solution would be to use rather complex hardware, which would keep track of the instruction dependencies and correct sequencing of dependent instructions [3].

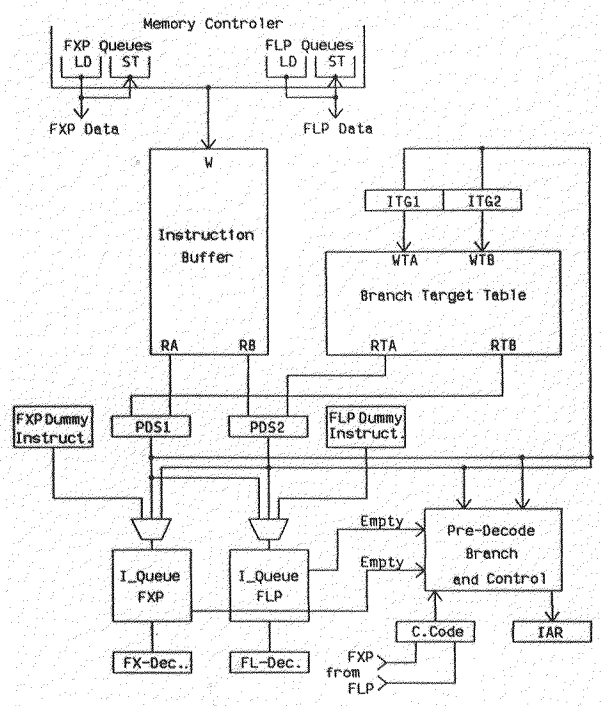

Figure 5. Organization of the Instruction Unit

In many high-end commercial processors (IBM 360/91, 3033, etc.) this problem is solved by using address comparison logic to compare the address of a load with addresses of stores from the floating point unit which have not been performed. This address comparison logic is used infrequently and consumes valuable area on a chip [3],[4].

There are two possible approaches with respect to synchronization:

One is to not to synchronize on Load and Store under the condition that sharing data through the memory is not permitted. The only exchange of data between fixed and floating point processors that is allowed is through the transfer instruction (TFFL), passing the data between the general purpose register files of the corresponding units. The transfer instruction is by itself a instruction requiring synchronization between the FXP and FLP processors. In addition the SYNC instruction is introduced with the sole purpose of providing synchronization between the units.

Reasons in support of this approach are:

- the gain in parallelism having the units running concurrently
- use each others data by the FXP and FLP processors is not that frequent

Having the synchronization on conditional branch and transfer instructions only, the units are made less dependent and they are allowed to operate with higher degree of parallelism.

The other approach is to synchronize on every FLP Load and Store instruction. Often FLP Load and Store is under control of the FXP unit which is used for their address calculation.

Floating Point Load

When a floating point load is detected at the pre-decode stage the following occurs:

- The load instruction is issued to the fixed point unit for address generation.
- A dummy Load (RECEIVE) instruction is issued to the floating point instruction queue (FLP-IQ), containing the destination register.
- Fixed point unit issues the load request to the memory controller by placing the request into the aforementioned memory request registers. The data from the requested memory location is then placed on the data queue inside the memory control unit.
- The RECEIVE instruction, when executed by the floating point unit, takes the data from the data queue and writes the data into the destination register.

Given that there are no out of sequence instructions, data need not to be tagged.

Floating point stores are performed in a similar way issuing

dummy SEND instruction which place data on the FLP Store Data Queue. The destination address is generated by the FXP unit and tagged to the data in the store queue. Data can not be written to the memory by the MCU until both data and address are valid.

CONCLUSION

Concurrent operation of FXP and FLP units of a single processor micro-system were discussed in this paper. The features of several commercially available systems were outlined and their important features discussed with respect to concurrency and instruction overlap. Two new approaches were outlined: one for a two-chip system consisting of a CPU (FXP and IU) and a FLP chip, and another for a system contained on a single chip.

REFERENCES

[1] J.E.Smith et al, *A Simulation Study of Decoupled Architecture Computers*, IEEE-TC, Vol. C-35, No. 8., August 1986.

[2] V.G.Oklobdzija, G.F.Grohosky, *Architectural Study for an Integrated Fixed and Floating Point VLSI-ASIC Processor*, Proceedings of COMPEURO 88, Vrije Universiteit Campus, Brussel, Belgium, April 11-14, 1988.

[3] R.M.Tomasulo, *An Efficient Algorithm for Exploiting Multiple Arithmetic Units*, IBM Journal of Research and Development, January 1967.

[4] D.W.Anderson at al, *The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling*, IBM Journal of Research and Development, January 1967.

[5] *A RISC Tutorial*, Sun Microsystems Inc., 1987. Weitek Corporation.

[6] *WEITEK: XL-Series Overview*, Preliminary Data, January 1988, Weitek Corporation.

[7] *MOTOROLA: MC68882 Technical Data*, Motorola Inc. 1986.

[8] *MOTOROLA: MC88100 Technical Data*, Motorola Inc. 1988.

[9] *INTEL: 80387 / 80386 Reference Manuals*, Intel Corp. 1987.