A STUDY ON BISC (BUS INSTRUCTION SET COMPUTER) ARCHITECTURE AND BISC-1

Student Number: 00M34567 Name: Makiko Ichirou SUZUKI Supervisor: Makiko TANAKA

BISC (BUS INSTRUCTION SET COMPUTER) アーキテクチャと
BISC–1 プロセッサに関する研究

鈴木 一郎

本論文では，BISC(Bus Instruction Set Computer) と呼ぶマイクロプロセッサの新しいアーキテクチャを提案する．これは，命令を本質的にプロセッサの内部バスによるレジスタ間のデータ移動だけに限るものである．このため，各部が独立性をもって動作することが可能であり，機能の追加変更に対して，命令セットの変更が不要なため，容易に様々な機能を追加することができる．

1 Introduction

The architectures of microprocessors in ordinary use today are the CISC, RISC, and VLIW versions [1]. The working speeds of these architectures are being increased by multi-stage pipelines, superscalar, branching estimation, and instruction reordering. However, introducing these functions makes the processors increasingly complicated, so it is actually difficult to make additional changes in processor functions according to needs. In case of VLIW processors, although they are simpler than CISC and RISC processors, it is still difficult since instructions of VLIW processors depend on their functions. Since in the field of signal processing or image processing, functions of a processor have to be changed for the purpose, such as JPEG or MPEG–1,2,4,7 or filtering, an architecture by which functions of a processor can be easily changed is needed.

In order to cope with this situation, Yamashita proposed a new architecture called *bus instruction set computer* (BISC) [4]. In this architecture, instructions are essentially limited to data transfers between registers by the internal bus in the processor. Then, the structure of BISC processors is very simple, and units in the processor work highly independently. For example, instructions are fetched to buffers independently with decoding and execution. Instruction decoding is only the decoding of binary numbers. Execution is only data transfer between registers.

Therefore, multi-stage pipelines are not needed. The ALU starts its calculation when all data are transferred. Instructions are not changed even when we make additional changes in processor functions, so it can be done easily. Furthermore, since the time needed for execution of an instruction is very short, the processor can be use its functions effectively. In order to demonstrate BISC architecture we are designing a processor called BISC-1, which has a doubled internal bus for an integer calculation unit.

Lipovski [2] and Corporaal [3] proposed architectures called SECP and TTAs, respectively, which essentially limit instructions only to data transfers. However, their architectures have the following disadvantages. The number of functions, including the number of ALU operations (‘add’, ‘sub’, etc.), is limited to the number of registers. The conditions for conditional branch are also restricted to several types. Since units in the processors don’t work independently, multi-stage pipelines are needed for effective execution and the time for the execution of an instruction is longer. Since the branch instruction is realized by the direct data transfer to the program counter, it is difficult to judge conditions in advance and do branching estimation.

In this paper, we explain the BISC architecture by describing the detail of BISC-1. We show experiments by using Dhrystone 2.1 and optimize the number of buffers in BISC-1.

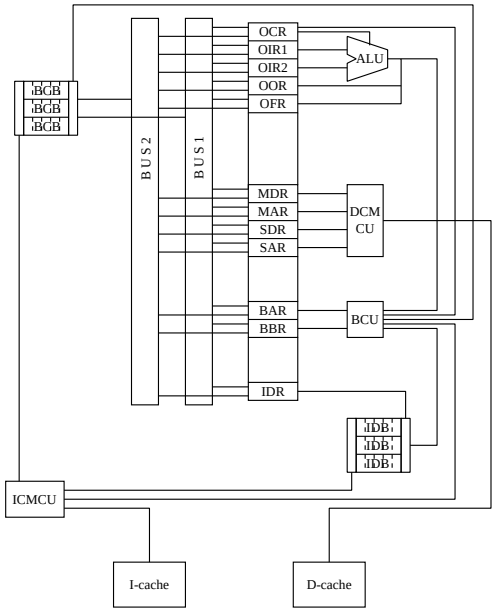
2 Overview of BISC-1

BISC-1 is a 32-bit integer processor, designed as a prototype BISC processor with a doubled bus for two data transfers per cycle.

2.1 Structure of BISC-1

The basic constitution of BISC-1 is shown in Figure 1. BISC-1 mainly consists of the following components, such as a doubled internal bus, a register file, mechanisms for instruction fetch and decoding, and three function units.

- **Bus** ··· An instruction is performed by transferring data through the bus. The bus is doubled so that it is capable of two data transfers per cycle.
- **Register File** ··· Register file consists of 64 registers connected to the bus. Registers are divided into functional and general purpose registers; the former, connected to function unit, plays a specific role and the latter just holds data.



ALU	Arithmetic Logic Unit
BAR	Branch Address Register
BBR	Branch Base Register
BCB	Bus Control Buffer
BCU	Branch Control Unit
BUS1	Bus 1
BUS2	Bus 2
D-cache	Data cache
DCMCU	Data Cache Memory Control Unit
I-cache	Instruction cache
ICMCU	Instruction Cache Memory Control Unit
IDB	Immediate Data Buffer
IDR	Immediate Data Register
MAR	Memory Address Register
MDR	Memory Data Register
OCR	Operation Control Register
OFR	Operation Flag Register
OIR	Operation Input Register
OOR	Operation Output Register
SAR	Stack Address Register
SDR	Stack Data Register

Figure 1: Basic constitution of BISC-1

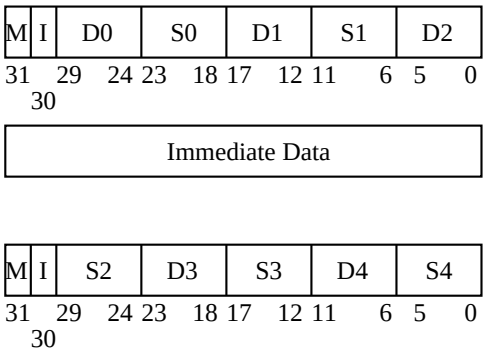


Figure 2: BISC-1 instruction format

- **Bus Control Buffer** (BCB) ··· A BCB consists a set of buffers to store data transfer instructions and there are several BCBs in BISC-1. The active BCB for bus control is chosen by BCU.
- **Immediate Data Buffer** (IDB) ··· An IDB consists a set of buffers to store immediate data and there are several IDBs. The active IDB for immediate data is chosen by BCU as well as BCB.

- **Instruction Cache Memory Control Unit** (ICMCU) ··· ICMCU transports instructions from the instruction cache to BCB or immediate data to IDB. It contains the program counter.
- **Branch Control Unit** (BCU) ··· BCU is a function unit to control the flow of BISC-1. BCU also controls BCB and IDB.
- **Data Cache Memory Control Unit** (DCMCU) ··· DCMCU is a function unit to control data cache memory access.
- **Arithmetic Logic Unit** (ALU) ··· ALU is a function unit to perform 28 operations including 13 operations to judge conditions for control flow.

2.2 Instruction format of BISC-1

Instructions of BISC-1 are restricted to data transfer between registers through the internal bus essentially. So, while traditional CISC or RISC architectures specify operations in the operation field of an instruction word, BISC needs no operation field because it has only one kind of instruction. It needs to specify only two registers of source and destination. Instruction format of BISC-1 is shown in Fig. 2.

Bit 31 is called Mark Bit (M-bit). M-bit indicates a block separation regarding control flow. When M-bit is ‘0’, BISC-1 continues instruction fetch. When M-bit is ‘1’, it is the last instruction word and the following instruction fetch depends on the state of internal registers. Bit 30, called Immediate-Data Bit (I-bit), indicates whether the next word in I-cache is an immediate data. When I-bit is ‘0’, the next word is an instruction word, otherwise an immediate data. The other bits specify

the source/destination registers for data transfers. Two words specify five sets of source/destination register. Bit 29–24 indicates the register code number of destination (D0) and Bit 23–18 indicates one of source (S0). Bit 17–6 (D1, S1) is also written the source/receive register number of the next instruction. Further instruction is written on Bit 5–0 (D2) and Bit 29–24 (S2) of the next word (if I-bit is '1', the word after next).

2.3 Basic mechanism of BISC-1

The program is stored in I-cache. ICMCU fetches words from I-cache and transfers instructions to BCB or an immediate data to IDB. The address for I-cache access is specified by the program counter in ICMCU. Both BCB and IDB to receive instructions and data, respectively, are chosen by BCU.

One of BCB, which is chosen by BCU, receives 12-bit data to specify registers for data transfer from ICMCU. At the same time, one of BCB, which is chosen by BCU, sends 12-bit data as long as BCB has instructions.

In order to enhance the performance, BISC-1 has a doubled bus. Two data transfers can be performed with the doubled bus.

Each bus receives a data transfer instruction, which consists of a pair of 6-bit register code numbers, from BCB. Suppose they are *n* and *m*. Then, the data in the register *n* is transferred to the register *m* through a bus. The preceding instruction uses BUS1 and the following instruction uses BUS2. Only if the data in the register *m* can be read and the register *n* can be written on, then the data in the register *m* is transferred to the register *n*. After that, each bus receives the next instruction. Otherwise, the instructions are not executed in this cycle. A wired logic around a set of registers for a function unit decides locally whether read/write data is possible or not.

IDB is connected to IDR. When the data of IDR are transferred, the next data of IDB are sent to IDR at the same time, and one of IDB, which is chosen by BCU, can receive an immediate data from ICMCU.

3 Function and specification of BISC-1

3.1 Arithmetic logic operation

We show how arithmetic logic operations are performed. First, the data indispensable for the operation are transferred to OCR, OIR1 and OIR2 (or transferred to OCR and OIR1, or transferred to OCR, according to the operation). The data for specifying operations is stored on OCR and operands are stored on OIR1 and OIR2. When all the needed data are transferred to OCR or OIRs, ALU starts the operation and stores the result to

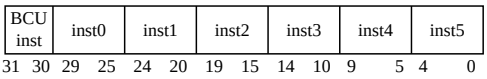


Figure 3: OCR formats

OOR and the flags to OFR. In some operations, the result and flags are not stored. If OOR is 'no receive' from ALU, they are not stored and ALU doesn't execute the next operation until OOR is 'receivable'. After the result produced by ALU is stored to OOR, the next result can't be stored to OOR until the former result is transferred through BUSes. After the data in OOR is transferred, the instruction which transfers the data in OOR can't be performed until the next result produced by ALU is stored to OOR. OFR holds 9 flags including 'carry', 'zero', 'overflow', etc.

ALU in BISC-1 supports 28 operations including 15 arithmetic logic operations ('add', 'sub', etc.) and 13 condition judgment operations ('if zero', 'if not zero', 'if plus or zero', etc.). Since a 5-bit field is wide enough to specify an operation, the word in OCR consists of six 5-bit operation specifiers, as shown in Fig. 3. ALU executes the operation specified by 5 bits in order of height from bit29 on OCR. The higher 2-bit field are used for control flow, as described in Section 3.3.

When the operation is a condition judgment operation, the result of judgment by using the content of OFR is transfered to BCU. OFR have to be set by the previous arithmetic operation.

3.2 Memory access

MDR and MAR are used to load/store memory. To load from memory, the address for the data is transferred to MAR. Then, the data at the address is sent to MDR. To store on memory, data is transferred to MDR at first. Next, the address is transferred to MAR. In case of load, MDR becomes 'no read' when data is written on MAR, and 'readable' when data from memory is written on it. In case of store, MDR becomes 'no write' when data is written on MDR, and MAR becomes 'no write' when data is written on MAR. When storing data on memory is finished, MAR and MDR become 'writable'. SDR and SAR are used to push/pop on the stack.

3.3 Control flow

The control flow function 'no operation' or 'unconditional jump' or 'conditional jump' or 'halt' is specified by the higher 2-bit of OCR. The control flow specifier is transferred to OCR with ALU function specifier and a target address for jump is transferred to BAR. On one hand, in case of unconditional jump, the address in BAR is certainly written on the Program Counter (PC) and for call and return the value in PC is written on BAR at the same time. On the other hand, in case of conditional

Table 1: Cycles and CPI for various number of buffers in BCB (with 2 buffers in IDB).

# of buffers	16	8	4	3
cycles	3345	3346	3347	3370
CPI	0.8119	0.8121	0.8124	0.8180

Table 2: Cycles, usage, and CPI for BISC-1 with 1 and 2 bus (with 4 buffers in BCB, 2 buffers in IDB).

multiplex degree of the bus		1	2
cycles		4638	3347
usage (%)	BUS1	88.83	79.53
	BUS2	–	43.56
CPI		1.1257	0.8124

jump, according to the result of a conditions judgment operation from ALU, the address in BAR is written on PC or not.

As described in Section 2.2. M-bit indicates a point where control flow function should be driven. Immediately after ICMCU fetches the word whose M-bit is evaluated '1', control flow is done. However, 'no operation' causes nothing and 'halt' halts the processor after the last instruction is finished. It is independent from data transfers between registers.

4 Experimental Results

We examined the performance of BISC-1 with 16, 8, 4, and 3 buffers in BCB, and 8, 4, 3, and 2 buffers in IDB. In the first place, we investigated the relation between performance and the number of IDB. As the results of the examinations, it turned out that the number of cycles for Dhrystone 2.1 is constant with the number the buffers in IDB. Therefore, it can be concluded that two buffers in IDB is enough. In the second place, we varied the number of buffers in BCB with two buffers in IDB. The results are shown in Table 1. Cycles in Table 1 mean the number of clocks cycles needed for a main loop in Dhrystone 2.1. CPI is almost constant between 4 and 16 buffers, and CPI with 3 buffers is worse. Therefore, the optimal number of buffers in BCB is four. As a result, the optimal number of buffers in BCB is four and IDB is two, when using this memory system.

The multiplex degree of the bus means the number of executable instructions in a cycle. Therefore, the performance of BISC-1 can be improved by increasing the multiplex degree of the internal bus. So, we compare the doubled bus version with the single bus version. The result is given in Table2. CPI for the doubled bus is smaller than the single bus, so the double bus has an advantage.

Although the usage of BUS1 can be raised by optimization of the instruction stream, the usage of BUS2 is not high. So, it is necessary to use BUS2 more effectively.

It is not necessary to implement so many buffers in BCB and IDB. The reason would be that BCB and IDB have only to hold instructions and an immediate data for the next cycle when using this memory system.

From observation of the process, it was proved that unconditional jump can be performed with little delay. The reason is that the specifier of unconditional jump and the target address is set when data are transferred to OCR and BAR, and instructions after the branch can be fetched previously.

ALU is almost busy. In order to solve this problem, the operation unit has to be added.

5 Conclusions

In this paper we discussed BISC architecture and the structure, specifications, and features of BISC-1 designed as a prototype BISC processor. We also showed the experimental results of BISC-1 and optimized the multiplex degree of the internal bus and the number of buffers in BCB and IDB.

Performance could be improved, by increasing the number of buses and implementing a floating-point unit, a multiplication unit, and a division unit, according to the needs of an application domain. And in order to exploit the performance of BISC-1, we are developing the high performance compiler for BISC-1. We are also planning to implement efficient 'interrupt' of BISC processors.

References

[1] John L. Hennessy and David A. Patterson, "Computer Organization & Design:The Hardware/Software Interface," Morgan Kaufmann Publishers, San Francisco, 1994.

[2] G. Jack Lipovski, "The architecture of a simple, effective control processor," In Second Euromicro Symposium on Microprocessing and Microprogramming, pp.7-18, Oct. 1976.

[3] Henk Corporaal and Paul van der Arend, "Move32int, a sea of gates realization of a high performance transport triggered architecture," Microprocessing and Microprogramming, vol. 38, pp.53-60, Sept. 1993.

[4] Yukihiko Yamashita, "A New Architecture of Multi-Purpose Microprocessor –Bus Instruction Set Computer (BISC)–," Technical report of IEICE, no. CPSY 96-70, pp.9-14, Oct. 1996. (in Japanese)