PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 :
G06F 9/00

A2

(11) International Publication Number: WO 00/70446

(43) International Publication Date: 23 November 2000 (23.11.00)

(21) International Application Number: PCT/US00/13198

(22) International Filing Date: 12 May 2000 (12.05.00)

(30) Priority Data:

60/134,253	13 May 1999 (13.05.99)	US
09/418,663	14 October 1999 (14.10.99)	US
09/524,178	13 March 2000 (13.03.00)	US

(71) Applicant: ARC INTERNATIONAL U.S. HOLDINGS INC. [US/US]; 6862 Santa Teresa Boulevard, San Jose, CA 95119 (US).

(72) Inventors: WARNES, Peter; 12a Walpole Road, East Ham, Greater London E6 1AR (GB). GRAHAM, Carl; 15 Woodland Road, North Chingford, Greater London E4 7ET (GB).

(74) Agent: NATAUPSKY, Steven, J.; Knobbe, Martens, Olson & Bear, LLP, 16th floor, 620 Newport Center Drive, Newport Beach, CA 92660-8016 (US).

(81) Designated States: AE, AG, AL, AM, AT, AT (Utility model), AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, CZ (Utility model), DE, DE (Utility model), DK, DK (Utility model), DM, DZ, EE, EE (Utility model), ES, FI, FI (Utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KR (Utility model), KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (Utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD AND APPARATUS FOR LOOSE REGISTER ENCODING WITHIN A PIPELINED PROCESSOR

(57) Abstract

An improved method and apparatus for implementing instructions in a pipelined central processing unit (CPU) or user-customizable microprocessor. In a first aspect of the invention, an improved method of "loosely" encoding register numbers to indicate register immediate data operand usage is disclosed. One embodiment comprises instruction words having multi-bit data fields defined therein which encode various types of immediate operands. Such multi-bit field definitions provide the programmer with additional flexibility in performing a variety of operations, including non-commutative operations. A method of synthesizing a processor design incorporating the aforementioned "loose" register encoding is also disclosed. Exemplary gate logic synthesized using the aforementioned method, and a computer program and system capable of implementing these methods are further disclosed.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

Code	Country	Code	Country	Code	Country	Code	Country
AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

METHOD AND APPARATUS FOR LOOSE REGISTER ENCODING WITHIN A PIPELINED PROCESSOR

This application claims priority to U.S. Provisional Patent Application Serial No. 60/134,253 filed May 13, 1999, entitled "Method And Apparatus For Synthesizing And Implementing Integrated Circuit Designs," and to co-pending U.S. Patent Application No. 09/418,663 filed October 14, 1999, entitled "Method And Apparatus For Managing The Configuration And Functionality Of A Semiconductor Design," which claims priority to U.S. Provisional Patent Application Serial No. 60/104,271 filed October 14, 1998, of the same title.

Background of the Invention

1. Field of the Invention

The present invention relates to the field of integrated circuit design, specifically to the use of a hardware description language (HDL) for implementing instructions in a pipelined central processing unit (CPU) or user-customizable microprocessor.

2. Description of Related Technology

RISC (or reduced instruction set computer) processors are well known in the computing arts. RISC processors generally have the fundamental characteristic of utilizing a substantially reduced instruction set as compared to non-RISC (commonly known as "CISC") processors. Typically, RISC processor machine instructions are not all micro-coded, but rather may be executed immediately without decoding, thereby affording significant economies in terms of processing speed. This "streamlined" instruction handling capability furthermore allows greater simplicity in the design of the processor (as compared to non-RISC devices), thereby allowing smaller silicon and reduced cost of fabrication.

RISC processors are also typically characterized by some or all of the following attributes: (i) load/store memory architecture (i.e., only the load and store instructions have access to memory; other instructions operate via internal registers within the processor); (ii) single cycle execution of most instructions; (iii) fixed length easily decoded instruction format; (iv) unity of processor and compiler, as well as a compiler which is simpler and easier to write; (v) hardwired control; (vi) fewer addressing modes; (vii) relatively static instruction format; and (viii) pipelining.

RISC Load/Store Architecture

As previously indicated, the load/store architecture of the RISC processor greatly simplifies the operation of the device by restricting memory accesses to only the load and

store instructions; other operations are "register-to-register". Hence, the typical RISC processor also employs a large number of internal registers to handle such operations. The following illustrates a simple load/store operation in support of an addition:

5 Operation: a = b + c

Instructions:

load r3, a	Load register 3 with value from source location a
load r4, b	Load register 4 with value from source location b
add r5, r3, r4	Add registers 3 and 4, and store result in register 5
store e, r5	store contents of register 5 at destination location e

10

As illustrated in the foregoing example, prior art RISC processors generally use an intermediate register (e.g., r5) to hold data during load/store operations to memory. Since most RISC processors rely on such a load/store mechanism for accessing and modifying
15 memory values, the instruction efficiency suffers when a simple memory access is desired.

Addressing Modes

An addressing mode is a way of accessing an operand, wherever it may be found. In general, operands can be located in memory or in a CPU register or they can be literal values defined in the code itself. Possible addressing modes used in the microprocessor
20 include, among others, "implied" addressing, wherein the opcode specifies the operand(s); "immediate" addressing, wherein the instruction itself contains the operand; "direct" addressing, wherein the operand is a memory address or register designation; "indirect" addressing, wherein the operand specifies the address of the desired operand; and "indexed" addressing, wherein two or more values are added or otherwise manipulated to get the
25 address of the operand.

Of the above-listed addressing modes, "immediate" addressing is often useful within a RISC processor, since the operand is contained directly within the instruction. As noted above, an immediate instruction (generally denoted by the syntax "imm" or derivations thereof) typically contains the operand within the instruction itself. An
30 immediate instruction generally has the operand as a literal value following a particular character such as the "#" sign. The format of the operand may vary. For example, the instruction may have an operand as follows:

```
$1234          ;operand is the literal value $1234
```

```
Buffer      ;operand is the literal value attached to "Buffer"
'Y'         ;operand is ASCII code for an upper case Y
```

Short Immediate v. Long Immediate

5 When using immediate addressing/data, smaller data words (commonly less than half the size of the instruction word) can frequently be encoded within the parent instruction word which specifies the operation. This approach is often called "short immediate" addressing/data; however, this approach imposes obvious restrictions on the allowable addresses/operands which can be used within the single instruction word. 10 Conversely, long immediate addressing/data requires more than one instruction word, yet removes many of the restrictions associated with short immediate modes.

Register Encoding

Registers within the RISC processor are closely tied to the instruction set, since such registers are frequently specified as operands by the instructions, or alternatively used 15 to generate addresses relating to operands.

Prior art instruction encoding schemes are generally structured such that either one or two bits out of the total number of bits in the instruction are utilized for indicating immediate operand usage, or immediate operand usage is implied by an alternative instruction type, as shown below. Generally the immediate data will be encoded in the 20 instruction word using a fixed set of bits, or by using the bits that would otherwise be used to describe a source data register.

```
add     r0,r1,r2        ; r0 = r1 + r2  ;       register-register add
addi    r0,r1,10        ; r0 = r1 + 10  ;       register-immediate add
```

25

This approach is consistent with the tendency to attempt to minimize instruction word and register size, thereby reducing the required silicon necessary to implement the processor design. However, using the instruction opcode to infer that of immediate data held within the instruction reduces the flexibility with which instructions may be encoded. For 30 example, using the foregoing prior art approach of encoding, it is effectively impossible to generate multiple long immediate constants using a single instruction word. In addition, if an operation (such as a subtract) is non-commutative, and it was desired to allow immediate data to be present as either the subtractor or the subtrahend, it would be necessary to have two immediate versions of the instruction as well as the register-register

version of the instruction. Figs.1a-1c illustrates typical prior art register encoding formats for register-register, register-immediate, and immediate-register instructions.

Generally, prior art processors do not allow all combinations of immediate data to be used with all instructions. Often it is not possible to choose between using a portion of the instruction word to encode immediate data, or to allow a subsequent instruction word to be used entirely for immediate data thus allowing a larger range of values to be encoded. In a processor with a user-extendable instruction set, this would place a limitation on how a programmer might use a new instruction, especially if the function of the new instruction was non-commutative.

Pipelining

Pipelining is a technique for increasing the performance of processor by dividing the sequence of operations within the processor into segments which are effectively executed in parallel when possible. In the typical pipelined processor, the arithmetic units associated with processor arithmetic operations (such as ADD, MULTIPLY, DIVIDE, etc.) are usually "segmented", so that a specific portion of the operation is performed in a given segment of the unit during any clock cycle. Hence, these units can operate on the results of a different calculation at any given clock cycle. As an example, in the first clock cycle two numbers A and B are fed to the multiplier unit 10 and partially processed by the first segment 12 of the unit. In the second clock cycle, the partial results from multiplying A and B are passed to the second segment 14 while the first segment 12 receives two new numbers (say C and D) to start processing. The net result is that after an initial startup period, one multiplication operation is performed by the arithmetic unit 10 every clock cycle.

The depth of the pipeline may vary from one architecture to another. In the present context, the term "depth" refers to the number of discrete stages present in the pipeline. In general, a pipeline with more stages executes programs faster but may be more difficult to program if the pipeline effects are visible to the programmer. Most pipelined processors are either three stage (instruction fetch, decode, and execute) or four stages (such as instruction fetch, decode, operand fetch, and execute, or alternatively instruction fetch, decode/operand fetch, execute, and writeback), although more or less stages may be used.

When developing the instruction set of a pipelined processor, several different types of "hazards" must be considered. For example, so called "structural" or "resource contention" hazards arise from overlapping instructions competing for the same resources (such as busses, registers, or other functional units) which are typically resolved using one

or more pipeline stalls. So-called "data" pipeline hazards occur in the case of read/write conflicts which may change the order of memory or register accesses. "Control" hazards are generally produced by branches or similar changes in program flow.

Interlocks are generally necessary with pipelined architectures to address many of these hazards. For example, consider the case where a following instruction (n +1) in an earlier pipeline stage needs the result of the instruction n from a later stage. A simple solution to the aforementioned problem is to delay the operand calculation in the instruction decoding phase by one or more clock cycles. A result of such delay, however is that the execution time of a given instruction on the processor is in part determined by the instructions surrounding it within the pipeline. This complicates optimization of the code for the processor, since it is often difficult for the programmer to spot interlock situations within the code.

"Scoreboarding" may be used in the processor to implement interlocks; in this approach, a bit is attached to each processor register to act as an indicator of the register content; specifically, whether (i) the contents of the register have been updated and are therefore ready for use, or (ii) the contents are undergoing modification such as being written to by another process. This scoreboard is also used in some architectures to generate interlocks which prevent instructions which are dependent upon the contents of the scoreboarded register from executing until the scoreboard indicates that the register is ready. This type of approach is referred to as "hardware" interlocking, since the interlock is invoked purely through examination of the scoreboard via hardware within the processor. Such interlocks generate "stalls" which preclude the data dependent instruction from executing (thereby stalling the pipeline) until the register is ready.

Alternatively, NOPs (no-operation opcodes) may be inserted in the code so as to delay the appropriate pipeline stage when desired. This later approach has been referred to as "software" interlocking, and has the disadvantage of increasing the code size and complexity of programs that employ instructions that require interlocking. Heavily software interlocked designs also tend not to be fully optimized in terms of their code structures.

Branch and Jump Instructions

Another important consideration in processor design is program branching or "jumps". All processors support some type of branching instructions. Simply stated, branching refers to the condition where program flow is interrupted or altered. Other operations such as loop setup and subroutine call instructions also interrupt or alter

program flow in a similar fashion. The term "jump delay slot" is often used to refer to the slot within a pipeline subsequent to a branching or jump instruction being decoded. The instruction after the branch (or load) is executed while awaiting completion of the branch/load instruction. Branching may be conditional (i.e., based on the truth or value of one or more parameters) or unconditional. It may also be absolute (e.g., based on an absolute memory address), or relative (e.g., based on relative addresses and independent of any particular memory address).

Branching can have a profound effect on pipelined systems. By the time a branch instruction is inserted and decoded by the processor's instruction decode stage (indicating that the processor must begin executing a different address), the next instruction word in the instruction sequence has been fetched and inserted into the pipeline. One solution to this problem is to purge the fetched instruction word and halt or stall further fetch operations until the branch instruction has been executed. This approach, however, by necessity results in the execution of the branch instruction in several instruction cycles, typically equal to the depth of the pipeline employed in the processor design. This result is deleterious to processor speed and efficiency, since other operations can not be conducted by the processor during this period.

Alternatively, a delayed branch approach may be employed. In this approach, the pipeline is not purged when a branch instruction reaches the decode stage, but rather subsequent instructions present in the earlier stages of the pipeline are executed normally before the branch is executed. Hence, the branch appears to be delayed by the number of instruction cycles necessary to execute all subsequent instructions in the pipeline at the time the branch instruction is decoded. This approach increases the efficiency of the pipeline as compared to multi-cycle branching described above, yet also complexity (and ease of understanding by the programmer) of the underlying code.

Based on the foregoing, an improved approach to register encoding within a pipelined and interlocked RISC processor is needed. Such an improved approach would allow the programmer/designer increased flexibility in encoding registers within the processor, yet overcome some of the disabilities associated with the load/store architecture (e.g., requirement to use an intermediate register to store immediate values), thereby optimizing instruction set and processor performance. Furthermore, a programmer could infer the use of short immediate data (held in the instruction word) or long immediate data (in a subsequent instruction word) in any source field of the processor's instruction words.

Ideally, this improved approach would also be compatible with other processor design considerations including, inter alia, interlocking and branch control schemes. Additionally, the ability to readily synthesize improved processor designs incorporating the aforementioned improvements in an application-specific manner, and using available synthesis tools, is of significant utility to the designer and programmer.

Summary of the Invention

The present invention satisfies the aforementioned needs by providing an improved method and apparatus for encoding registers and executing instructions within a pipelined processor architecture.

In a first aspect of the invention, a method of "loosely" encoding register numbers to indicate register immediate operand usage is disclosed. In one embodiment, a plurality of expanded (e.g., six-bit) register fields are used within the long instruction word of the processor, thereby providing enhanced flexibility in the instruction and operand formats available. This approach also affords the ability to store immediate values directly to memory without using an intermediate register. The use of short immediate data (held in the instruction word) or long immediate data (in a subsequent instruction word) in any source field of processor instructions may also be inferred. Non –commutative operations are also more efficiently handled using this approach.

In a second aspect of the invention, an improved method of synthesizing the design of an integrated circuit incorporating the aforementioned jump delay slot method is disclosed. In one exemplary embodiment, the method comprises obtaining user input regarding the design configuration; creating customized HDL functional blocks based on the user's input and existing library of functions; determining the design hierarchy based on the user's input and the library and generating a hierarchy file, new library file, and makefile; running the makefile to create the structural HDL and scripts; running the generated scripts to create a makefile for the simulator and a synthesis script; and synthesizing the design based on the generated design and synthesis script.

In a third aspect of the invention, an improved computer program useful for synthesizing processor designs and embodying the aforementioned synthesis method is disclosed. In one exemplary embodiment, the computer program comprises an object code representation stored on the magnetic storage device of a microcomputer, and adapted to run on the central processing unit thereof. The computer program further comprises an interactive, menu-driven graphical user interface (GUI), thereby facilitating ease of use.

In a fourth aspect of the invention, gate logic implementing the aforementioned "loose" register encoding and functionality, and synthesized using the foregoing method of synthesizing a processor design, is disclosed. In one exemplary embodiment, the gate logic for selecting a first source field within the register comprises a series of eight 4-bit
5 multiplexers.

In a fifth aspect of the invention, an improved processor architecture utilizing the foregoing "loose" register encoding methodology is disclosed. In one exemplary embodiment, the processor comprises a reduced instruction set computer (RISC) having a multi-stage pipeline which utilizes "loose" register architecture to, inter alia, effect storage
10 of immediate values directly to memory without the use of intermediate registers. In another embodiment, the processor includes a processor core, DSP core, a memory with a plurality of memory banks, and a memory interface for parallel interfacing of DSP functions with banks within the memory.

In a sixth aspect of the invention, an improved apparatus for running the
15 aforementioned computer program used for synthesizing logic associated with pipelined processors is disclosed. In one exemplary embodiment, the system comprises a stand-alone microcomputer system having a display, central processing unit, data storage device(s), and input device.

Brief Description of the Drawings

20 Figs. 1a-1c illustrate a typical prior art register coding scheme for a RISC processor.

Fig. 2 is a logical flow diagram illustrating the generalized methodology of locating data within "loosely" encoded registers within a pipelined processor according to the present invention.

Figs. 3a-3c graphically illustrate a first embodiment of the register encoding
25 architecture of the present invention.

Fig. 4 is a logical flow diagram illustrating the generalized methodology of synthesizing processor logic which incorporates "loose" register encoding according to the present invention.

Fig. 5 is a schematic diagram illustrating one embodiment of synthesized logic used to
30 select the data source for a first field of the instruction word of Fig. 3.

Fig. 6 is a schematic diagram illustrating a first embodiment of synthesized logic (unconstrained) used to implement the 4-bit multiplexers of the data source selection logic of Fig. 5.

Fig. 7 is a schematic diagram illustrating a second embodiment of synthesized logic (constrained) used to implement the 4-bit multiplexers of the data source selection logic of Fig. 5.

Fig. 8 is a schematic diagram illustrating a first embodiment of synthesized logic (unconstrained) used to implement the flag setting functionality of the present invention.

Fig. 9 is a schematic diagram illustrating a second embodiment of synthesized logic (constrained) used to implement the flag setting functionality of the present invention.

Fig. 10 is a block diagram of a processor design incorporating "loose" register encoding according to the present invention.

Fig. 11 is a functional block diagram of a computing device incorporating the hardware description language of the present invention, used to synthesize the logic apparatus of Figs. 5-9.

Detailed Description of the Invention

Reference is now made to the drawings wherein like numerals refer to like parts throughout.

As used herein, the term "processor" is meant to include any integrated circuit or other electronic device capable of performing an operation on at least one instruction word including, without limitation, reduced instruction set core (RISC) processors such as the ARC user-configurable core manufactured by the Assignee hereof, central processing units (CPUs), and digital signal processors (DSPs).

Additionally, it will be recognized by those of ordinary skill in the art that the term "stage" as used herein refers to various successive stages within a pipelined processor; i.e., stage 1 refers to the first pipelined stage, stage 2 to the second pipelined stage, etc. While the following discussion is cast in terms of a three stage pipeline (i.e., instruction fetch, decode, and execution stages), it will be appreciated that the methodology and apparatus disclosed herein are broadly applicable to processor architectures with one or more pipelines having more or less than three stages.

It is also noted that while the following description is cast in terms of VHSIC hardware description language (VHDL), other hardware description languages such as Verilog® may be used to describe various embodiments of the invention with equal success. Furthermore, while an exemplary Synopsys® synthesis engine such as the Design Compiler 1999.05 (DC99) is used to synthesize the various embodiments set forth herein, other synthesis engines such as Buildgates® available from Cadence Design Systems, Inc., may be used. IEEE std. 1076.3-1997, IEEE Standard VHDL Synthesis Packages specify an industry-

accepted language for specifying a Hardware Definition Language-based design and the synthesis capabilities that may be expected to be available to one of ordinary skill in the art.

Lastly, it will be recognized that while the following description illustrates specific embodiments of logic synthesized by the Assignee hereof using the aforementioned synthesis engine and VHSIC hardware description language, such specific embodiments being constrained in different ways, these embodiments are only exemplary and illustrative of the design process of the present invention.

The method and apparatus for loose register encoding according to the present invention is now described. In general, the invention utilizes an expanded multi-bit register field to indicate the register immediate operand usage. Stated simply, the invention comprises using register numbers within the processor to indicate short immediate ("shimm") and long immediate ("limm") operands. This approach is termed "loose" in that it effectively expands or unpacks the number of bits ordinarily necessary to indicate this information. For example, one embodiment of the instruction word of the CPU core of the present invention employs 6 bit register fields to indicate register AND immediate operand usage (e.g., shimm/limm). In contrast, a typical prior art instruction word would utilize only 1 or 2 bits to indicate this information, or imply the presence of immediate data using the opcode field of the instruction. Hence, the approach is somewhat counter-intuitive, in that it uses more bit capacity than the minimum required to represent the information.

The "loose" register encoding architecture of the present invention has several benefits for RISC–based processors (such as the Applicant's ARC core previously described), however, including: (i) overall enhanced programming flexibility; (ii) the ability to store immediate values directly to memory without using an intermediate register; (iii) the ability to use short or long immediate data in either a first source register field ("source 1") or a second source register field ("source 2"), which is of benefit for instructions with non-commutative behavior; (iv) the ability to indicate that the result of an instruction should be discarded by using an 'immediate data' register value in the destination field of an instruction. This enables the programmer to make a comparison between two values and set flags as a result, but not cause any of the general purpose registers in the processor to be changed; and (v) the ability to use both short and long immediate data as source data in one instruction. In the case where extension instructions may be added to a processor, this latter ability proves useful in the design and operation of specialized instructions. Since most RISC processors rely on a load-store mechanism for accessing and modifying memory values (i.e., only LOAD and STORE operations may

access memory space), the instruction efficiency suffers when simple memory access is desired. By employing the loose encoding scheme of the present invention, program memory may be optimized while achieving the simplicity inherent in a RISC architecture.

Referring now to Fig. 2, one embodiment of the generalized methodology of locating data within "loosely" encoded registers according to the present invention is described. The first step 202 of the method 200 comprises determining if the register number of interest within the current instruction specifies a general purpose register (e.g., r0-r31 in the embodiment of Table 1 below). If the register number does specify a general purpose register, then the data is selected from the specified core register per step 204, and the process 200 is completed for that register number. If a general purpose register is not specified, the register number is next examined to determine whether it specifies an immediate data value per step 206. If an immediate data value is specified, the type of immediate data value; i.e., short immediate (shimm) or long immediate (limm) is determined in step 208. If the register number does not specify an immediate data value in step 206, the specified data value is obtained from the cited source as appropriate in step 210.

If short immediate data is specified in step 208, the data is extracted from the relevant portion of the current instruction word. If long immediate data is specified in step 208, then the data is extracted from the subsequent instruction word(s) as appropriate.

A first exemplary embodiment of the register and instruction architecture of the present invention useful with the foregoing method is described in Table 1 below:

Table 1

Register	Immediate Operand	Description
r0-r31		Register value.
r32-r59		Extension registers (application specific).
r60	loopcnt	Loop counter register
r61	shimmf	use 9 bit signed short immediate from instruction word and set status flags on result.
r62	limm	use 32 bit long immediate from next instruction word
r63	shimm	Use 9 bit signed short immediate from instruction word

As indicated in Table 1, sixty-four total registers are specified (i.e., r0-r63). The first thirty-two registers (r0-r31) are general purpose registers used to reflect the register value. The next 28 registers (r32-r59) are extension registers which are specific to the particular application. The next register (r60) is the loop counter register, which in the ARC processor is used as part of the zero overhead looping mechanism, to maintain count of the number of iterations remaining in a loop construct. The last three registers (r61-r64) are utilized to indicate an immediate operand (shimmf, limm, or shimm, respectively). Since the bit in the instruction word used to set the flags is required to encode short immediate data, there are two versions of shimm ; one with (i.e. shimmf) and one without (i.e. shimm) flag setting. Figs. 3a-3c graphically illustrate the foregoing embodiment of the register encoding structure according to the present invention.

The aforementioned approach gives the programmer/designer complete flexibility to specify a variety of different instruction formats, including the following eight exemplary instruction formats:

Table 2

Format No.	Syntax
1.	op.<cc>.<f> a,b,c
2.	op.<cc>.<f> a,b,l
3.	op.<cc>.<f> a,l,c
4.	op.<cc>.<f> a,l,l
5.	op.<cc>.<f> a,b,c
6.	op.<cc>.<f> a,b,s
7.	op.<cc>.<f> a,s,c
8.	op.<cc>.<f> a,s,s

where:

op = instruction operation

<cc> = optional condition code for execution

<f> = optional set status flags

a = Destination register

b = Source 1 register

c = Source 2 register

s = shimm (9 bit signed short immediate)

l = limm (32 bit long immediate)

5

It will be recognized that while the foregoing eight instruction formats of Table 2 are specified for purposes of illustration herein, other formats may be used depending on the particular application. For example, an instruction format may be used which has a lesser or greater number of registers than the 64 registers illustrated above. Furthermore, the
10 invention may be embodied in an instruction format having only two source operands, or one source and one destination operand. It will also be appreciated that the instruction format of the present invention could be embodied such that the syntax of the word is altered from that shown above; e.g., the order of the source and destination fields may be changed or permuted.

15 Table 3 provides a second exemplary embodiment of the instruction format according to the present invention, used in conjunction with Applicant's "ARC" RISC core:

Table 3

Format No.	Syntax	Description
9.	op b,c	two source fields, destination is implied
10.	op b,s	one source field, one shimm
11.	op b,l	one source field, one limm
12.	op s,c	shimm, one source field
13.	op l,c	limm, one source field
14.	op s,l	shimm, limm
15.	op l,s	limm, shimm
16.	op s,s	shimm, shimm
17.	op l,l	limm, limm
18.	op a,b	one destination field, one source
19.	op a,s	one destination, shimm
20.	op a,l	one destination, limm

Note that in the second embodiment of Table 3, only two fields (other than the
20 instruction operation "op") are specified. Furthermore, no conditional fields or flag setting fields are specified either, although it will be readily apparent that such conditional and/or

flag-setting fields may be used with these formats.

The following two formats from Table 2 are noted in particular:

4. op.<cc>.<f> a,l,l

8. op.<cc>.<f> a,s,s

These two formats are particularly useful to provide a MOV (move) immediate instruction by using the AND operation. In the ARC processor, using the short immediate register encoding extracts the short immediate value from the instruction word. If the short immediate register encoding is used in two source fields, both source fields will take the value of the short immediate field, however two different short immediate values cannot be encoded. By using the long immediate register encoding the data in the subsequent instruction word can be used in one or both source fields, but two different long immediate values cannot be used. In the present invention, it is possible to have both a short and a long immediate value, however; this advantageously enables the storing of an immediate value to an immediate location in memory.

Hence the instruction AND.a,l,l moves the contents of the subsequent long immediate instruction word to destination register "a". The operation carried out logically ANDs the same value with itself, which results in the original value.

The foregoing two formats (4. and 8.) are also used by the shift instructions to generate multiple long immediate constants using a single word instruction, as illustrated in the following example:

```
ASL.a,s,s      ; a = s << (s & 31)
```

(shift instructions only use the bottom 5 bits of an immediate value)
(short immediate data is 9 bits in length)

In the foregoing example, the bottom 5 bits of the source short immediate value is used to shift the entire 9 bit short immediate value to enable a wider range of immediate values to be placed in a register using a single instruction word than would otherwise be possible by using just the MOV (AND) operation described above with unshifted 9 bit short immediate data.

As previously described, the "loose" architecture of the present invention can also be used to store immediate values directly to memory without using an intermediate register as in prior art RISC devices, as illustrated by the following example:

```
ST s,[b,s] ; [b+s] = s (shimms MUST match)
ST 1,[b,s] ; [b+s] = 1 (wherein "l" designates long immediate data)
ST s,[s,s] ; [s+s] = s (shimms MUST match)
```

Also, by using register r63 (Table 1) as a destination, the register write-back of the result is caused to be discarded, which is useful for the situation where just the setting of the status flags of the result are required (such as for test/compare), without respect to any MOV instruction. The assembler syntax for this function uses an immediate value of "0" as the destination of the instruction, as shown below:

```
op.<cc>.<f>   0,b,c
op.<cc>.<f>   0,b,l
op.<cc>.<f>   0,l,c
op.<cc>.<f>   0,l,l
op.<cc>.<f>   0,b,c
op.<cc>.<f>   0,b,s
op.<cc>.<f>   0,s,c
op.<cc>.<f>   0,s,s
```

In the present embodiment, a file is used to contain the multiplexers which select how data gets selected onto the source 1 and source 2 buses. These buses are used, inter alia, at stage 3 of the pipeline as the inputs to the arithmetic logic unit (ALU), as illustrated by the following example:

<u>Stage 2 Result Multiplexers</u>

Source 1 field;

```
with s1a select:
   s1_direct <= qd_a  when
```

```
r0  | r1  | r2  | r3  | r4  | r5  | r6  | r7  |
r8  | r9  | r10 | r11 | r12 | r13 | r14 | r15 |
r16 | r17 | r18 | r19 | r20 | r21 | r22 | r23 |
r24 | r25 | r26 | r27 | r28 | r29 | r30 | r31 |

loopcnt    when  rlcnt,
shimmex    when  rfshimm | rnshimm,
pliw       when  rlimm,
x1data     when  others;
```

It is noted that in the illustrated example, the stage 2 result is initially selected using the "s1a" field, with the shortcut subsequently added.

Source 2 field;

```
with s2a select:
        s2_direct <= qd_b  when

r0  | r1  | r2  | r3  | r4  | r5  | r6  | r7  |
r8  | r9  | r10 | r11 | r12 | r13 | r14 | r15 |
r16 | r17 | r18 | r19 | r20 | r21 | r22 | r23 |
r24 | r25 | r26 | r27 | r28 | r29 | r30 | r31 |

loopcnt    when  rlcnt,
shimmex    when  rfshimm | rnshimm,
pliw       when  rlimm,
x1data     when  others;
```

Since the short immediate (shimm) field overlaps the flag-setting bit in the conditional instruction format, additional logic is used to control flag setting. In one embodiment of this logic, either the ".f " bit of the instruction or the value implied by the short immediate data register number is used, or is set to "false" if the instruction cannot set the flags (e.g., load/store, branches/jump). Special case flag-setters (Jcc.F and FLAG) are handled separately in a separate file (flags.vhd). If a 3-operand extension instruction is being used

which uses the short immediate region of the instruction for a purpose other than encoding short immediate data as indicated by the xshimm signal, the flags are not set. This feature of the present embodiment is further illustrated by the following exemplary flag setting calculation:

Stage 3 Flag Setting Calculation:

```
ip3setflags <= '0'    WHEN f_no_fset(ip3i) = '1'
                      or (xshimm AND x_idecode3 AND xt_aluop) = '1' ELSE
                      ip3shimmf    WHEN ip3shimm = '1'          ELSE
                      ip3_fbit;
```

It is also noted that various different embodiments of the foregoing multiplexers may be employed consistent with the invention, depending on the specific method used to code the VHDL. The coding of such different embodiments of the multiplexers, based on the foregoing functionality, is well within the capability of one of ordinary skill in the programming arts, and accordingly will not be described further herein.

The methods and apparatus of the present invention may also advantageously be used in conjunction with (either alone or collectively) methods of pipeline control and interlocking employed within a pipelined processor including, inter alia, those disclosed in Applicant's co-pending U.S. Patent Applications entitled "Method And Apparatus For Jump Control In A Pipelined Processor," "Method And Apparatus For Jump Delay Slot Control In A Pipelined Processor," and "Method And Apparatus For Processor Pipeline Segmentation And Re-assembly," each filed contemporaneously herewith, each being incorporated by reference herein in its entirety.

Method of Synthesizing

Referring now to Fig. 4, the method 400 of synthesizing logic incorporating the jump delay slot mode functionality previously discussed is described. The generalized method of synthesizing integrated circuit logic having a user-customized (i.e., "soft") instruction set is disclosed in Applicant's co-pending U.S. Patent Application Serial No. 09/418,663 entitled "Method And Apparatus For Managing The Configuration And Functionality Of A Semiconductor Design" filed October 14, 1999, which is incorporated by reference herein in its entirety.

While the following description is presented in terms of an algorithm or computer program running on a microcomputer or other similar processing device, it can be appreciated that other hardware environments (including minicomputers, workstations, networked computers, "supercomputers", and mainframes) may be used to practice the method. Additionally, one or more portions of the computer program may be embodied in hardware or firmware as opposed to software if desired, such alternate embodiments being well within the skill of the computer artisan.

Initially, user input is obtained regarding the design configuration in the first step 402. Specifically, desired modules or functions for the design are selected by the user, and instructions relating to the design are added, subtracted, or generated as necessary. For example, in signal processing applications, it is often advantageous for CPUs to include a single "multiply and accumulate" (MAC) instruction. In the present invention, the instruction set of the synthesized design is modified so as to incorporate one or more of the foregoing loose register encoding formats (or another comparable register coding format) therein. The technology library location for each VHDL file is also defined by the user in step 402. The technology library files in the present invention store all of the information related to cells necessary for the synthesis process, including for example logical function, input/output timing, and any associated constraints. In the present invention, each user can define his/her own library name and location(s), thereby adding further flexibility.

Next, in step 403, customized HDL functional blocks based on the user's input and the existing library of functions specified in step 402 are created.

In step 404, the design hierarchy is determined based on the user's input and the aforementioned library files. A hierarchy file, new library file, and makefile are subsequently generated based on the design hierarchy. The term "makefile" as used herein refers to the commonly used UNIX makefile function or similar function of a computer system well known to those of skill in the computer programming arts. The makefile function causes other programs or algorithms resident in the computer system to be executed in the specified order. In addition, it further specifies the names or locations of data files and other information necessary to the successful operation of the specified programs. It is noted, however, that the invention disclosed herein may utilize file structures other than the "makefile" type to produce the desired functionality.

In one embodiment of the makefile generation process of the present invention, the user is interactively asked via display prompts to input information relating to the desired design such as the type of "build" (e.g., overall device or system configuration), width of

the external memory system data bus, different types of extensions, cache type/size, etc. Many other configurations and sources of input information may be used, however, consistent with the invention.

In step 406, the makefile generated in step 404 is run to create the structural HDL. This structural HDL ties the discrete functional block in the design together so as to make a complete design.

Next, in step 408, the script generated in step 406 is run to create a makefile for the simulator. The script to generate a synthesis script is also run in step 408.

At this point in the program, a decision is made whether to synthesize or simulate the design (step 410). If simulation is chosen, the user runs the simulation using the generated design and simulation makefile (and user program) in step 412. Alternatively, if synthesis is chosen, the user runs the synthesis using the synthesis script(s) and generated design in step 414. After completion of the synthesis/simulation scripts, the adequacy of the design is evaluated in step 416. For example, a synthesis engine may create a specific physical layout of the design that meets the performance criteria of the overall design process yet does not meet the die size requirements. In this case, the designer will make changes to the control files, libraries, or other elements that can affect the die size. The resulting set of design information is then used to re-run the synthesis script.

If the generated design is acceptable, the design process is completed. If the design is not acceptable, the process steps beginning with step 402 are re-performed until an acceptable design is achieved. In this fashion, the method 400 is iterative.

Synthesized Logic

Referring now to Figs. 5-9, exemplary logic for implementing the "loose" register encoding functionality previously described herein, and synthesized using the method of Fig. 4, is described.

Fig. 5 illustrates one embodiment of the top hierarchy level logic for loose register encoding source 1 selection. In the embodiment of Fig. 5, the top level logic instantiates eight identical 4-bit multiplexers to make up the full 32 bits. [Note that the logic illustrated in Fig. 5 has been split into two levels for clarity of presentation]. Such logic may also be employed for the selection of source 2 data as well.

Fig. 6 illustrates a first embodiment of the aforementioned 4-bit multiplexer useful for loose register encoding of source 1. In the multiplexer of Fig. 6, no operational or design constraints were placed on the synthesis.

Fig. 7 illustrates a second embodiment of the 4-bit multiplexer for loose register encoding of source 1, except that the logic has been constrained to provide the shortest path from the long immediate data input bus ('p1iw') to the output bus ('s1_direct').

Fig. 8 illustrates a first embodiment of flag setting logic according to the invention (unconstrained).

Fig. 9 illustrates a second embodiment of the flag setting logic of the invention, except being constrained for minimum area.

Referring now to Fig. 10, an exemplary pipelined processor fabricated using a 1.0 um process and incorporating the logic of Figs. 5-9 is illustrated. As shown in Fig. 10, the processor 1000 is an ARC microprocessor-like CPU device having, inter alia, a processor core 1002, on-chip memory 1004, and an external interface 1006. The device is fabricated using the customized VHDL design obtained using the method 400 of the present invention, which is subsequently synthesized into a logic level representation, and then reduced to a physical device using compilation, layout and fabrication techniques well known in the semiconductor arts.

It will be appreciated by one skilled in the art that the processor of Fig. 10 may contain any commonly available peripheral such as serial communications devices, parallel ports, timers, counters, high current drivers, analog to digital (A/D) converters, digital to analog converters (D/A), interrupt processors, LCD drivers, memories and other similar devices. Further, the processor may also include custom or application specific circuitry. The present invention is not limited to the type, number or complexity of peripherals and other circuitry that may be combined using the method and apparatus. Rather, any limitations are imposed by the physical capacity of the extant semiconductor processes which improve over time. Therefore it is anticipated that the complexity and degree of integration possible employing the present invention will further increase as semiconductor processes improve.

It is also noted that many IC designs currently use a microprocessor core and a DSP core. The DSP however, might only be required for a limited number of DSP functions, or for the IC's fast DMA architecture. The invention disclosed herein can support many DSP instruction functions, and its fast local RAM system gives immediate access to data. Appreciable cost savings may be realized by using the methods disclosed herein for both the CPU & DSP functions of the IC.

Alternatively, the processor 1000 of Fig. 10 may be synthesized so as to incorporate a memory interface useful for interfacing between one or more IC (e.g., DSP) functions and

the memory array of the processor 1000, as described in Applicant's co-pending U.S. Patent application entitled "Memory interface and Method of Interfacing Between Integrated Circuits," filed March 10, 2000, and incorporated by reference herein in its entirety.

Additionally, it will be noted that the methodology (and associated computer program) as previously described herein can readily be adapted to newer manufacturing technologies, such as 0.35, 0.18 or 0.1 micron processes, with a comparatively simple re-synthesis instead of the lengthy and expensive process typically required to adapt such technologies using "hard" macro prior art systems.

Referring now to Fig. 11, one embodiment of a computing device capable of synthesizing, inter alia, the logic structures of Figs. 5-9 herein is described. The computing device 1100 comprises a motherboard 1101 having a central processing unit (CPU) 1102, random access memory (RAM) 1104, and memory controller 1105. A storage device 1106 (such as a hard disk drive or CD-ROM), input device 1107 (such as a keyboard or mouse), and display device 1108 (such as a CRT, plasma, or TFT display), as well as buses necessary to support the operation of the host and peripheral components, are also provided. The aforementioned VHDL descriptions and synthesis engine are stored in the form of an object code representation of a computer program in the RAM 1104 and/or storage device 1106 for use by the CPU 1102 during design synthesis, the latter being well known in the computing arts. The user (not shown) synthesizes logic designs by inputting design configuration specifications into the synthesis program via the program displays and the input device 1107 during system operation. Synthesized designs generated by the program are stored in the storage device 1106 for later retrieval, displayed on the graphic display device 1108, or output to an external device such as a printer, data storage unit, other peripheral component via a serial or parallel port 1112 if desired.

While the above detailed description has shown, described, and pointed out novel features of the invention as applied to various embodiments, it will be understood that various omissions, substitutions, and changes in the form and details of the device or process illustrated may be made by those skilled in the art without departing from the invention. The foregoing description is of the best mode presently contemplated of carrying out the invention. This description is in no way meant to be limiting, but rather should be taken as illustrative of the general principles of the invention. The scope of the invention should be determined with reference to the claims.

WHAT IS CLAIMED IS:

1. A method of encoding data within an instruction word, comprising:

providing a first instruction word having an op-code and a plurality of fields, each of said fields comprising a plurality of bits;

5 associating a first of said fields with a first data source;

associating a second of said fields with a second data source; and

performing a logical operation using said first and second data sources as operands, said logical operation being specified by said op-code.

2. The method of Claim 1, wherein said first and second fields comprise 6- bits
10 each.

3. The method of Claim 1, wherein the act of associating with a first data source comprises associating said first field with an immediate operand.

4. The method of Claim 3, wherein the act of associating with a second data source comprises associating said second field with an immediate operand.

15 5. The method of Claim 4, wherein said first data source is a short immediate operand, and said second data source is a long immediate operand located within a second instruction word.

6. A method of encoding data within an instruction word, comprising:

providing a first instruction word having an op-code and a plurality of fields, each
20 of said fields comprising a plurality of bits;

associating a first of said fields with a destination register;

associating a second of said fields with a first data source;

associating a third of said fields with a second data source;

performing a logical operation using said first and second data sources as operands,
25 said logical operation being specified by said op-code; and

storing the result of said logical operation in said destination register.

7. The method of Claim 6, wherein the act of associating with a first data source comprises associating said second field with an immediate operand, and the act of associating with a second data source comprises associating said third field with an
30 immediate operand.

8. The method of Claim 7, wherein at least one of said immediate operands comprises a long immediate operand disposed within a second instruction word.

9. A method of generating a long immediate constant using a single word instruction, comprising:

providing an instruction word having an op-code and at least one short immediate value associated therewith, said at least one short immediate value comprising a plurality of bits;

selecting a portion of said plurality of bits of said at least one short immediate value;

shifting all of said bits of said at least one short immediate value using said op-code and only said portion of bits to produce a shifted immediate value; and

storing said shifted immediate value in a register.

10. The method of Claim 9, wherein said op-code comprises a shift operation, and said portion comprises 5-bits of said short immediate word.

11. A method of providing source data associated with a field within an instruction word encoded with at least one register number, comprising:

determining when said at least one register number identifies a first type of register;

determining when said register number identifies an immediate data value;

determining the type of immediate data value identified by said at least one register number; and

extracting immediate data of the type identified by said at least one register number, said immediate data being provided as said source data.

12. The method of Claim 11, wherein the act of identifying a first type of register comprises identifying one type of a plurality of different types of registers, wherein only said first type of register is used to identify immediate data.

13. The method of Claim 12, wherein said immediate data comprises short immediate data contained within a single instruction word.

14. The method of Claim 11, wherein the act of determining the type of immediate data comprises selecting the type of immediate data from a group consisting of short immediate data and long immediate data.

15. The method of Claim 14, wherein said group further consist of short immediate data with flag-setting.

16. A processor design synthesized by the method comprising:

inputting information to a first file specific to the design to include at least one instruction word, said at least one instruction word comprising at least one register number, said at least one register number being related to a short immediate or long immediate operands;

defining the location of at least one library file;

generating a second file using said first file, said library file, and user input information;

running said second file to create a structural description language model; and

synthesizing said design based on said structural description language model.

5 17. The method of Claim 16, wherein the act of synthesizing comprises running synthesis scripts based on said description language model.

18. The method of Claim 17, further comprising the act of generating a third file for use with a simulation, and simulating said design using said third file.

19. The method of Claim 18, further comprising the act of evaluating the
10 acceptability of the design based on said simulation.

20. The method of Claim 19, further comprising the acts of revising the design to produce a revised design, and re-synthesizing said revised design.

21. The method of Claim 16, wherein the act of inputting information including said at least one instruction word comprises inputting an instruction word having a plurality
15 of register numbers, only a portion of said plurality of register numbers being representative of immediate data.

22. The method of Claim 21, wherein the act of inputting further comprises providing a plurality of input parameters associated with said design, said parameters comprising:

20 (i) a cache configuration; and

(ii) a memory interface configuration.

23. A machine readable data storage device comprising:

a data storage medium adapted to store a plurality of data bits; and

a computer program rendered as a plurality of data bits and stored on said data storage
25 medium, said program being adapted to run on the processor of a computer system and synthesize integrated circuit logic for use in a processor having a pipeline and incorporating an instruction set having at least one instruction word, said at least one instruction word comprising a plurality of register numbers, only a portion of said plurality of register numbers being representative of immediate operand data.

30 24. The storage device of Claim 23, wherein said immediate operand data comprises immediate operands selected from the group comprising short immediate operands, long immediate operands, and short immediate operands with flag setting.

25. The data storage device of Claim 23, wherein said data storage medium is a compact disk-read only memory (CD-ROM), and said plurality of data bits comprises an object representation of said program.

26. A digital processor comprising:

5 a processor core having a multistage instruction pipeline, said core being adapted to decode and execute an instruction set comprising a plurality of instruction words;

a memory; and

an instruction set, at least a portion of which is stored in said memory, said instruction set comprising a plurality of instruction words, at least one of said instruction

10 words comprising a plurality of data bits, at least a portion of said data bits forming at least one data field, said at least one data field representing at least one register number, said at least one register number representing immediate operand data.

27. The processor of Claim 26, wherein said at least one data field comprises 6-bits.

15 28. The processor of Claim 26, wherein said at least one instruction word comprises a first source field and a second source field, at least one of said first and second source fields specifying long immediate data.

29. The processor of Claim 26, further comprising a memory interface having:

a plurality of functional ports operatively coupled to respective ones of a plurality

20 of functions of said processor core; and

a plurality of memory ports operatively coupled to respective ones of a plurality of memory banks within said memory;

said memory interface arbitrating accesses to individual ones of said memory banks by individual ones of said functions of said processor core.

25 30. The processor of Claim 29, said memory interface further comprising a synchronized protocol between at least one of said function ports and a respective one of said functions of said processor core.

31. The processor of Claim 26, said instruction set further comprising at least one jump instruction word having a plurality of data bits, at least a portion of said plurality

30 of data bits of said at least one jump instruction word comprising a plurality of jump delay slot modes.

32. A reduced instruction set processor having (i) a plurality of registers; (ii) an instruction set, at least one instruction word within said instruction set comprising a first data source field and a second data source field, at least one of said first and second source

relating to a first register within said processor; and (iii) at least one arithmetic logic unit, wherein said at least one instruction word is executed using the method comprising:

determining when said first register comprises a first type of register;

determining when said first register identifies an immediate data value;

5 determining the type of immediate data value identified by said first register; and

extracting immediate data of the type identified by said first register, said immediate data being provided as one operand for said at least one instruction word.

33. An apparatus for synthesizing the design of logic used within a digital processor, comprising:

10 a central processing unit;

a data storage device operatively coupled to said central processing unit, said data storage device being adapted to store and retrieve a computer program;

an input device adapted to generate signals in response to inputs from a user of said system;

15 a computer program, stored on said data storage device, said program being adapted to receive said signals and permit said user to:

input information to a first file specific to the design to include said instruction set comprising a plurality of instruction words, at least one of said instruction words comprising a plurality of data bits, at least a portion of said data

20 bits forming at least one data field, said at least one data field representing at least one register number, said at least one register number representing immediate operand data.;

define the location of at least one library file;

generate a second file using said first file, said library file, and user input

25 information;

run said second file to create a structural description language model; and

synthesize said design based on said description language model.

34. A reduced instruction set processor having (i) a plurality of registers; (ii) an instruction set, at least one instruction word within said instruction set comprising a first

30 data source field and a second data source field, at least one of said first and second source relating to a first register within said processor; and (iii) at least one arithmetic logic unit, further comprising:

means for determining when said first register comprises a first type of register;

means for determining when said first register identifies an immediate data value;

means for determining the type of immediate data value identified by said first register; and

means for extracting immediate data of the type identified by said first register, said immediate data being provided as one operand for said at least one instruction word.

5 35. A instruction word for use within a digital processor having a plurality of registers, comprising:

an operation code (op-code);

a first data field, said first data field comprising a plurality of data bits; and

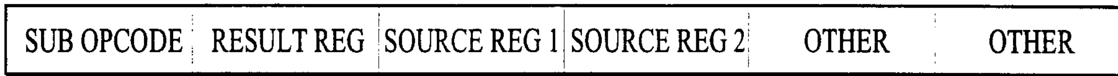
a second data field, said first data field comprising a plurality of data bits;

10 wherein said first and second data fields define immediate operand data.

36. The instruction word of Claim 35, further comprising a short immediate operand.

FIG. 1a
PRIOR ART

REGISTER - REGISTER TYPE INSTRUCTION



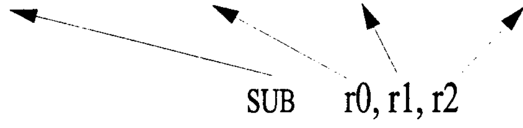
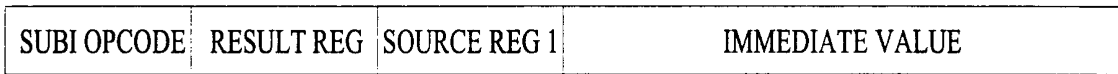


FIG. 1b
PRIOR ART

REGISTER - IMMEDIATE TYPE INSTRUCTION



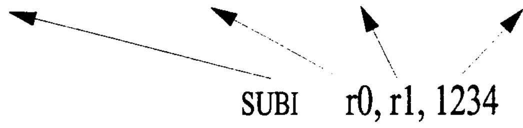
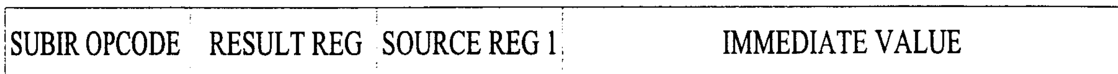


FIG. 1c
PRIOR ART

IMMEDIATE - REGISTER TYPE INSTRUCTION



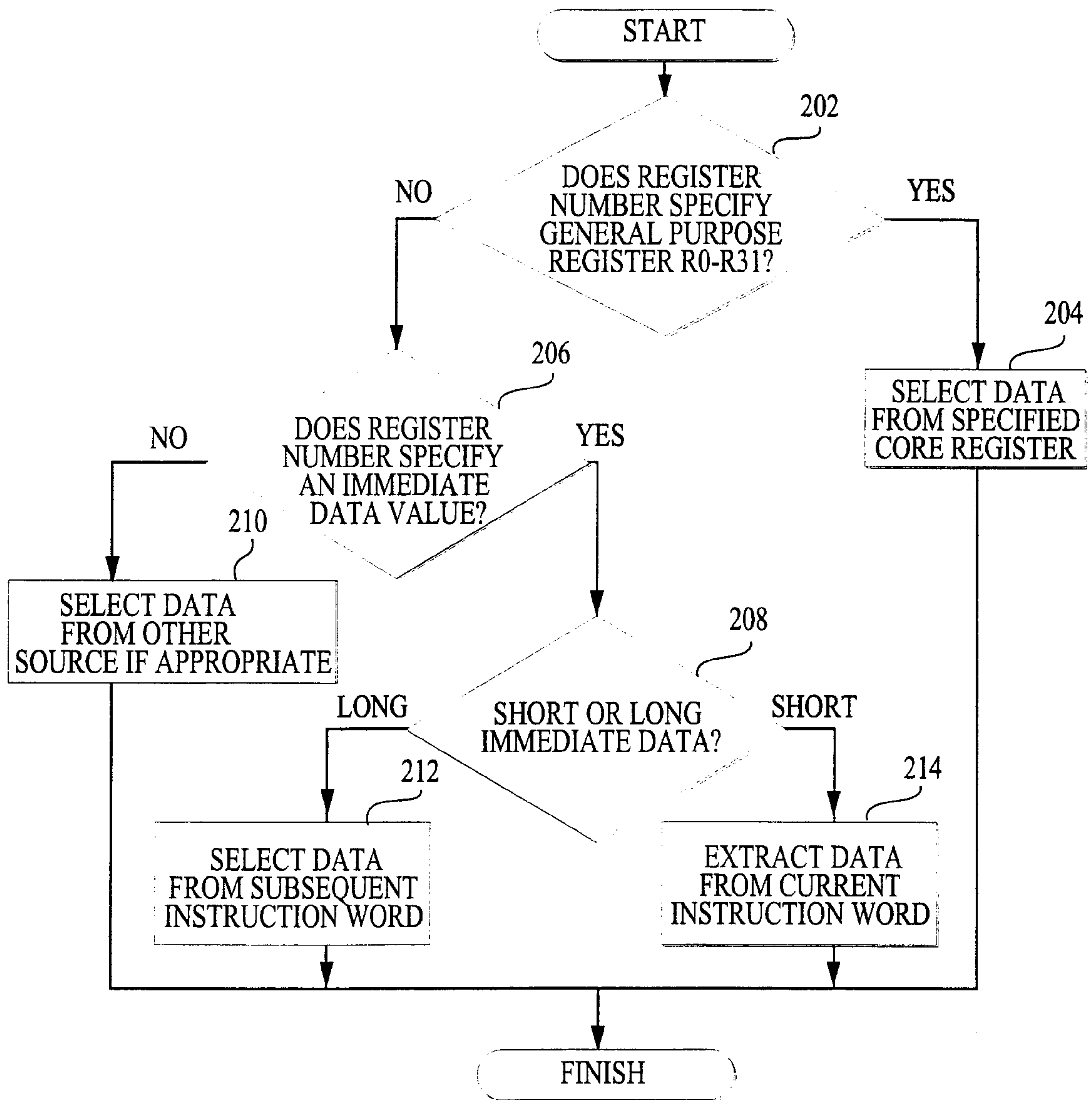
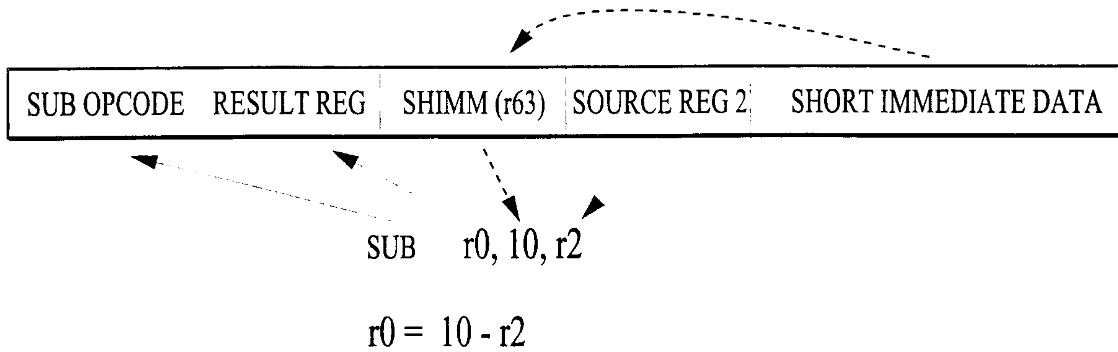


FIG. 2

FIG. 3a

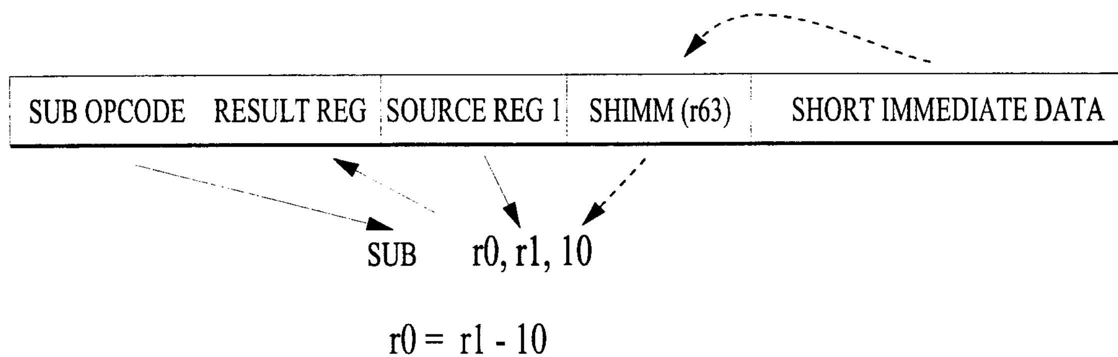
SHORT IMMEDIATE DATA USED IN SOURCE FIELD 1:



r0 = 10 - r2

FIG. 3b

SHORT IMMEDIATE DATA USED IN SOURCE FIELD 2:

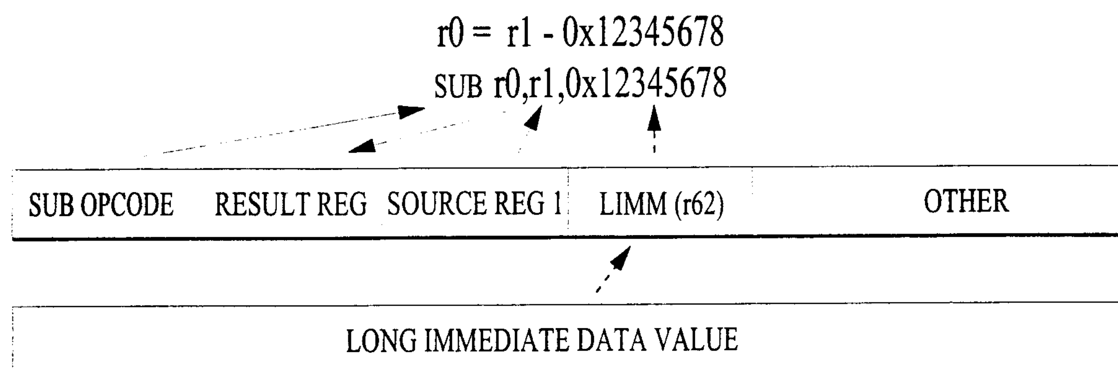


r0 = r1 - 10

FIG. 3c

LONG IMMEDIATE DATA USED IN SOURCE FIELD 2:

r0 = r1 - 0x12345678



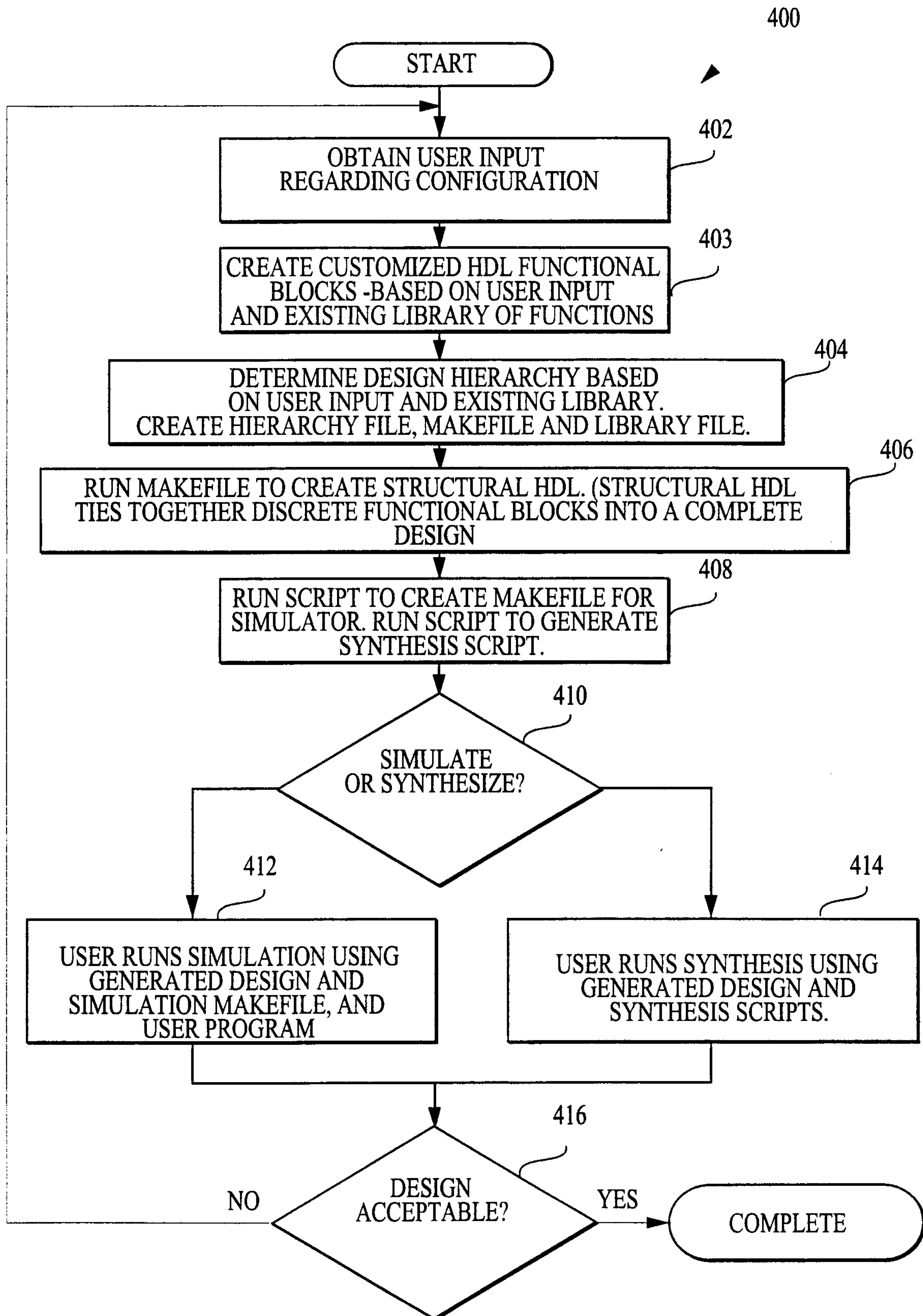


FIG. 4

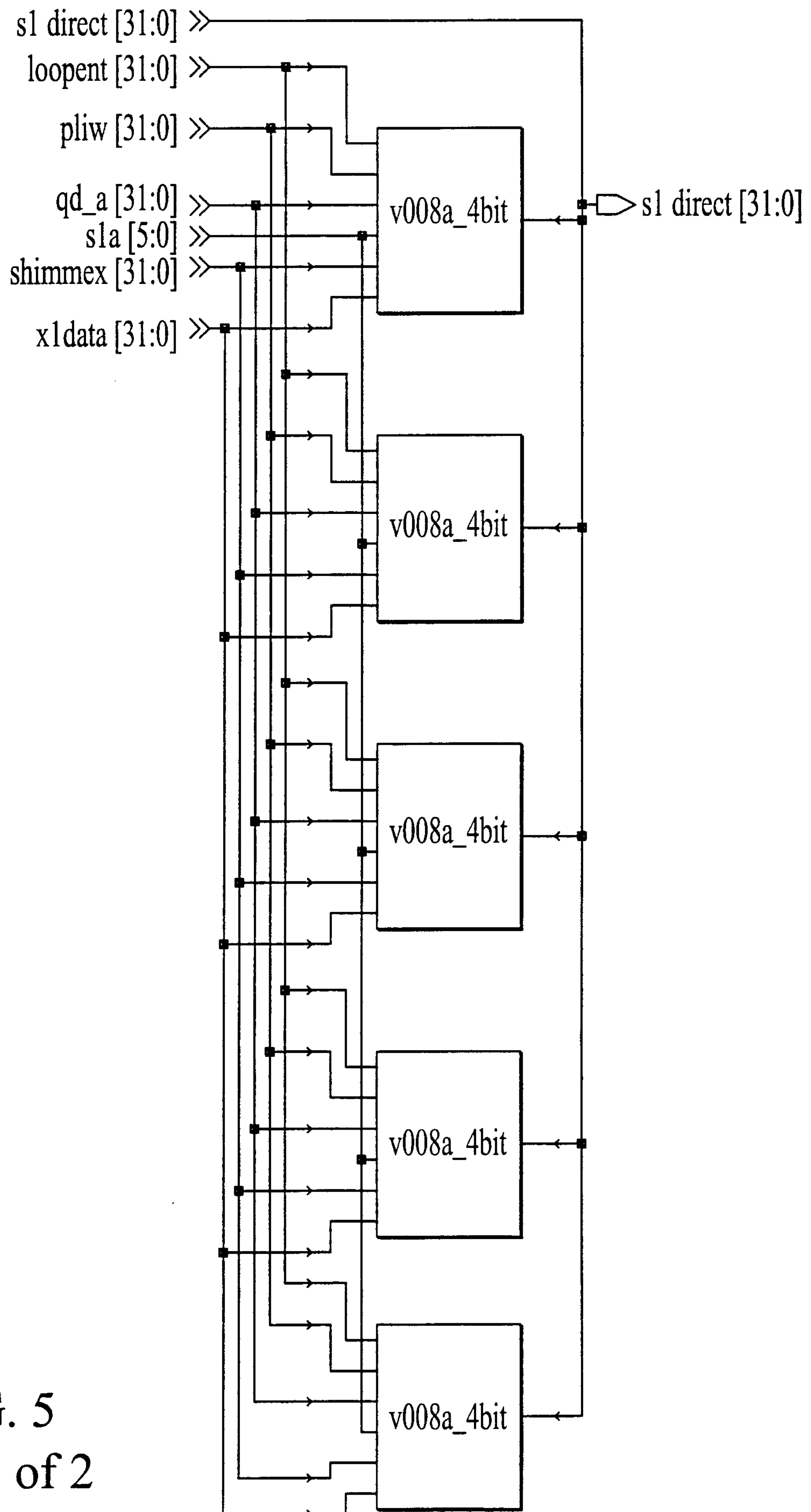


FIG. 5
part 1 of 2

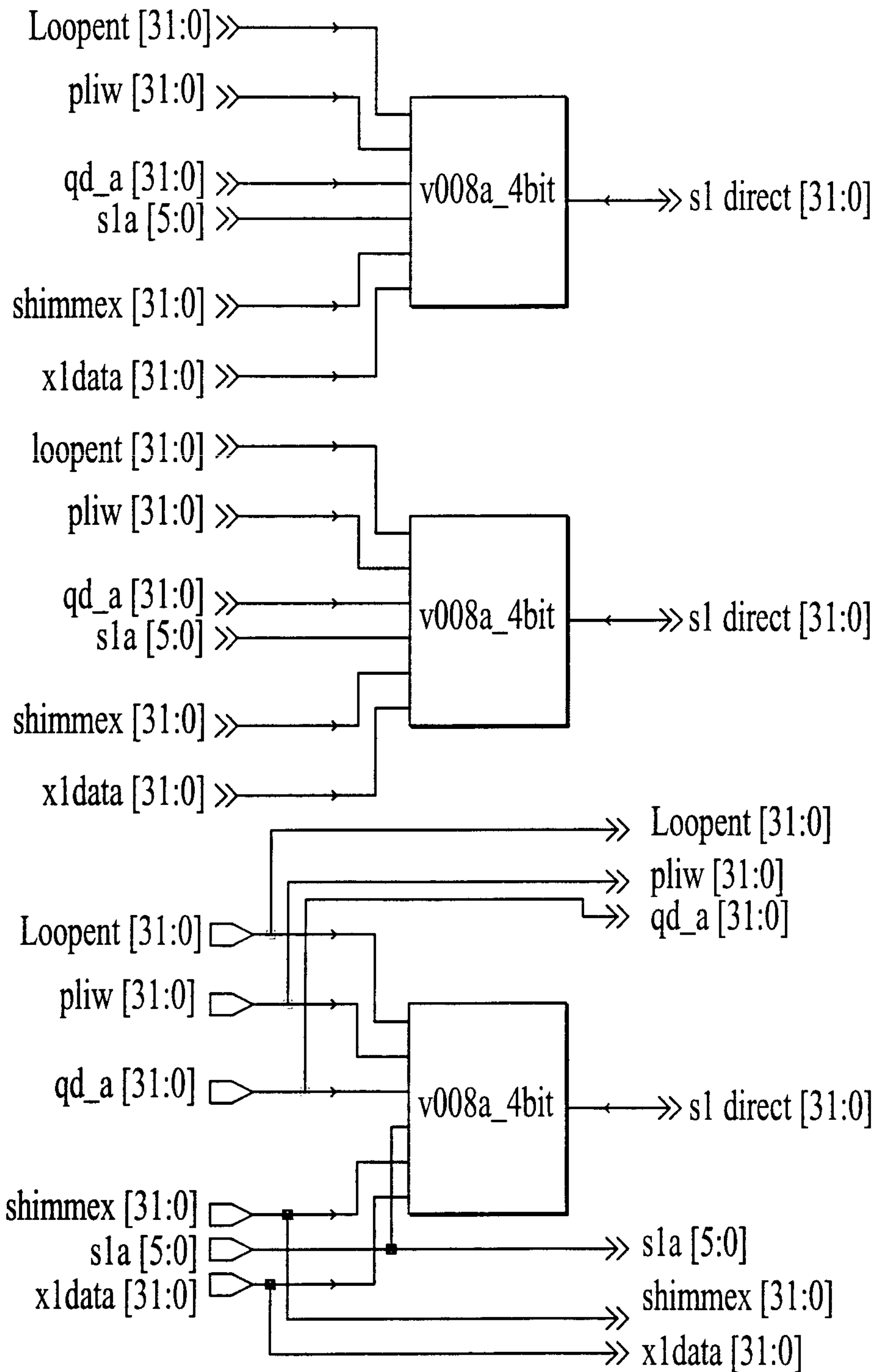


FIG. 5
part 2 of 2

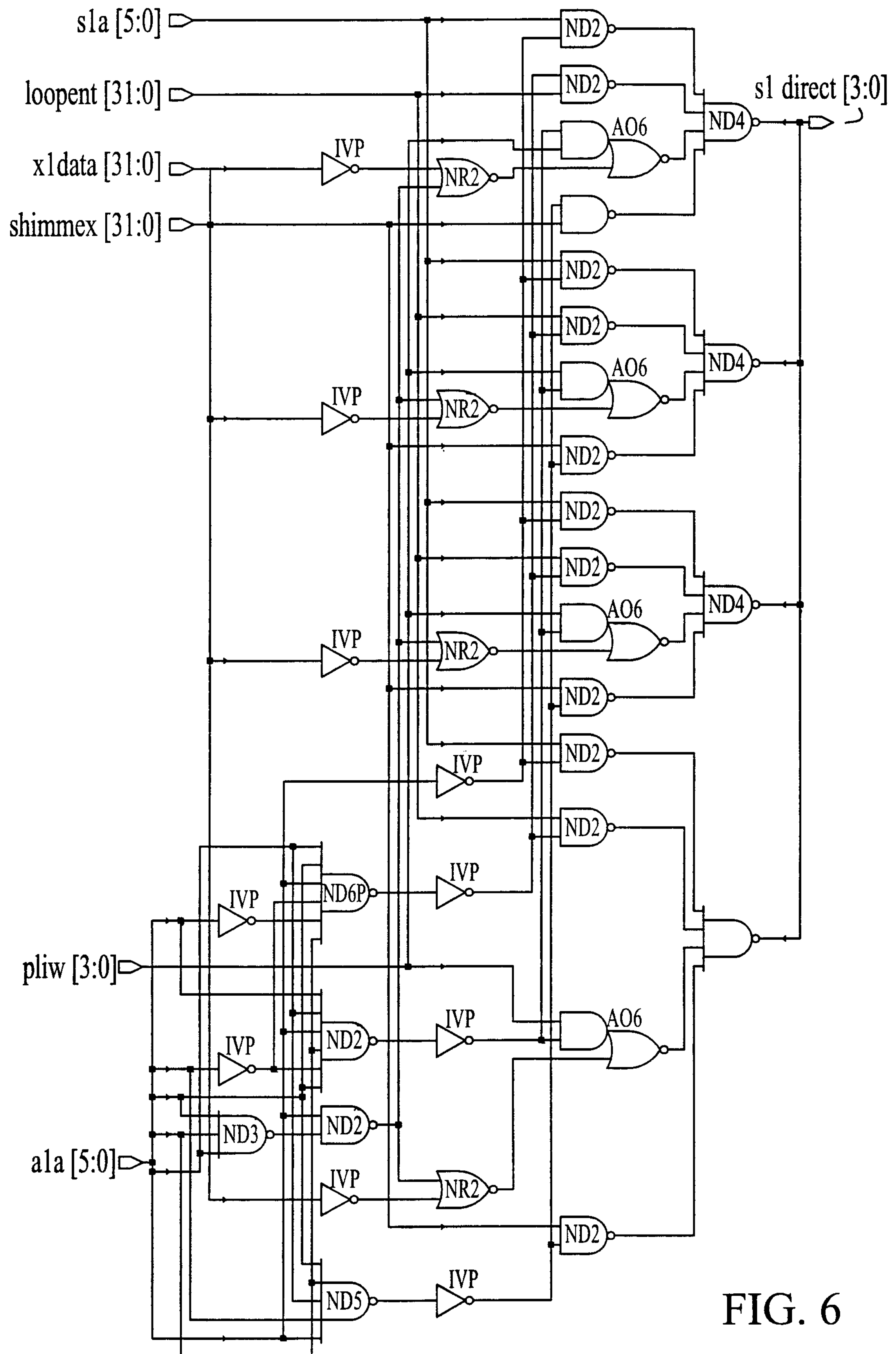


FIG. 6

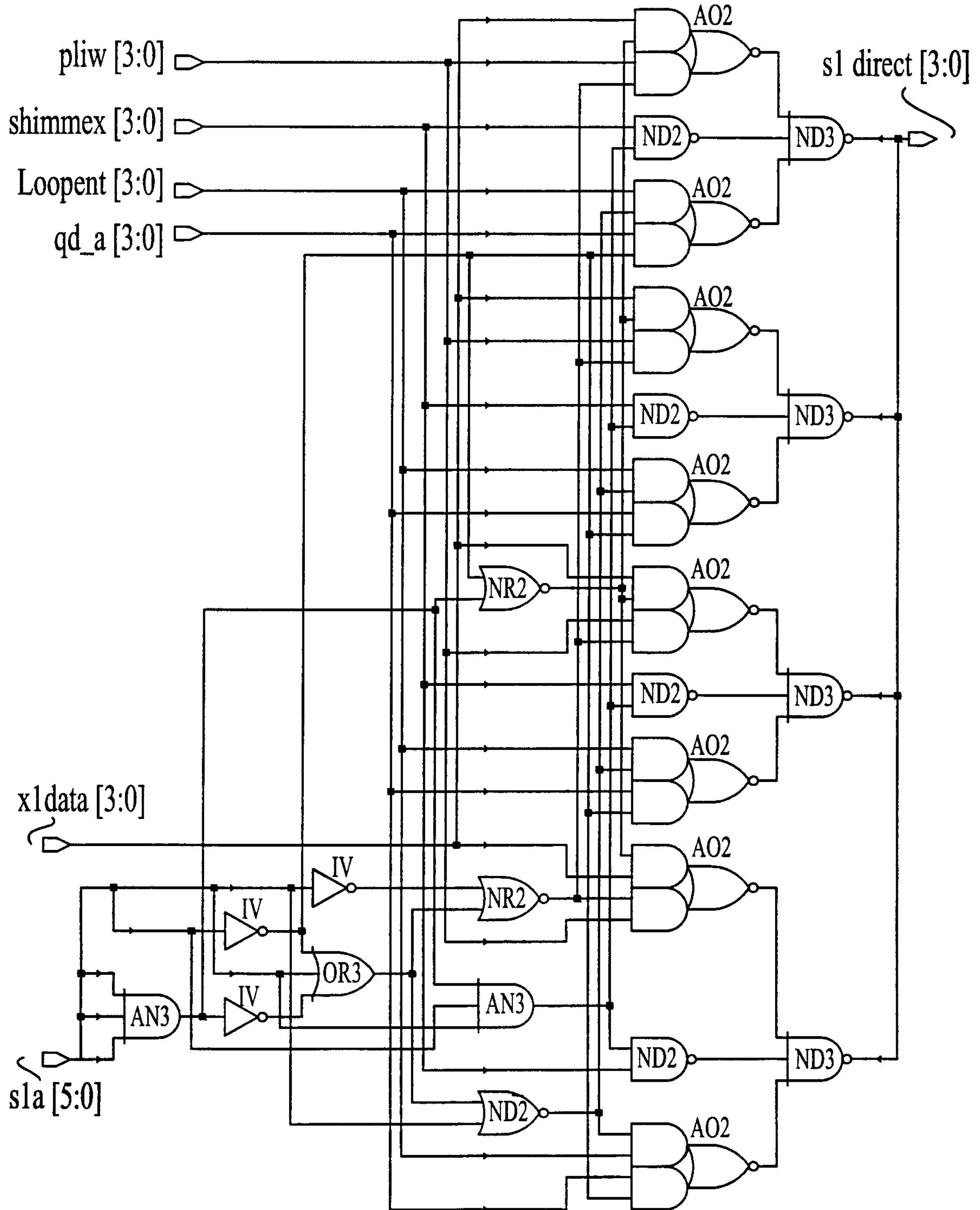


FIG. 7

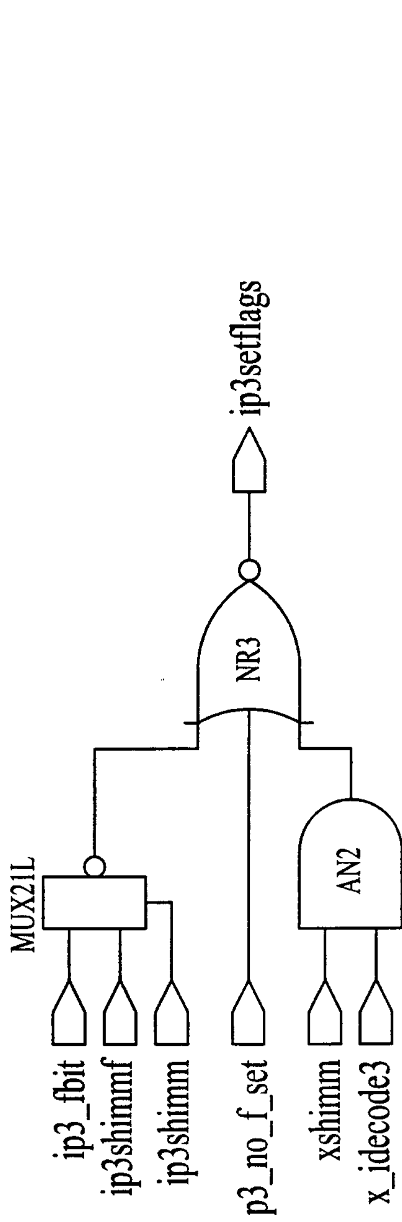


FIG. 8

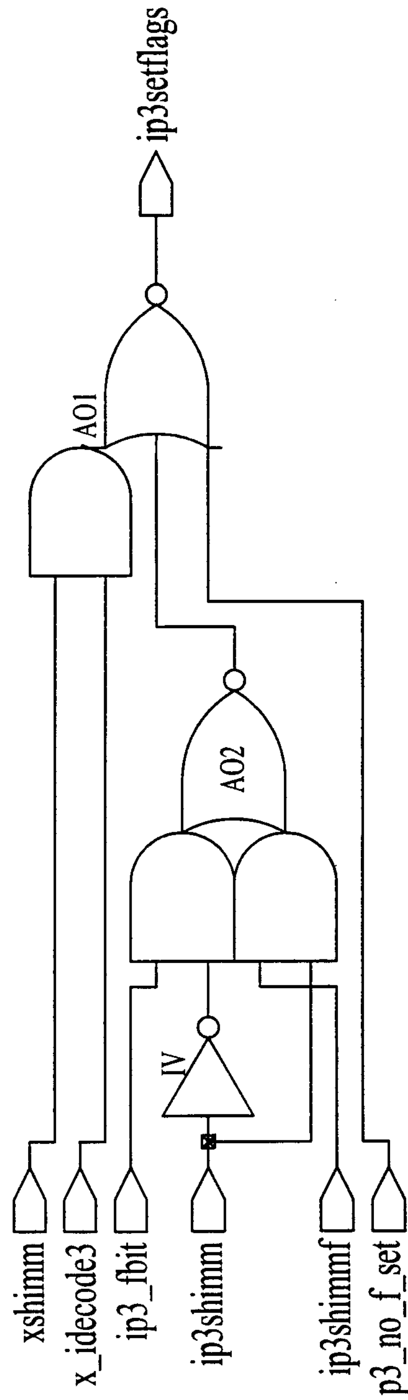


FIG. 9

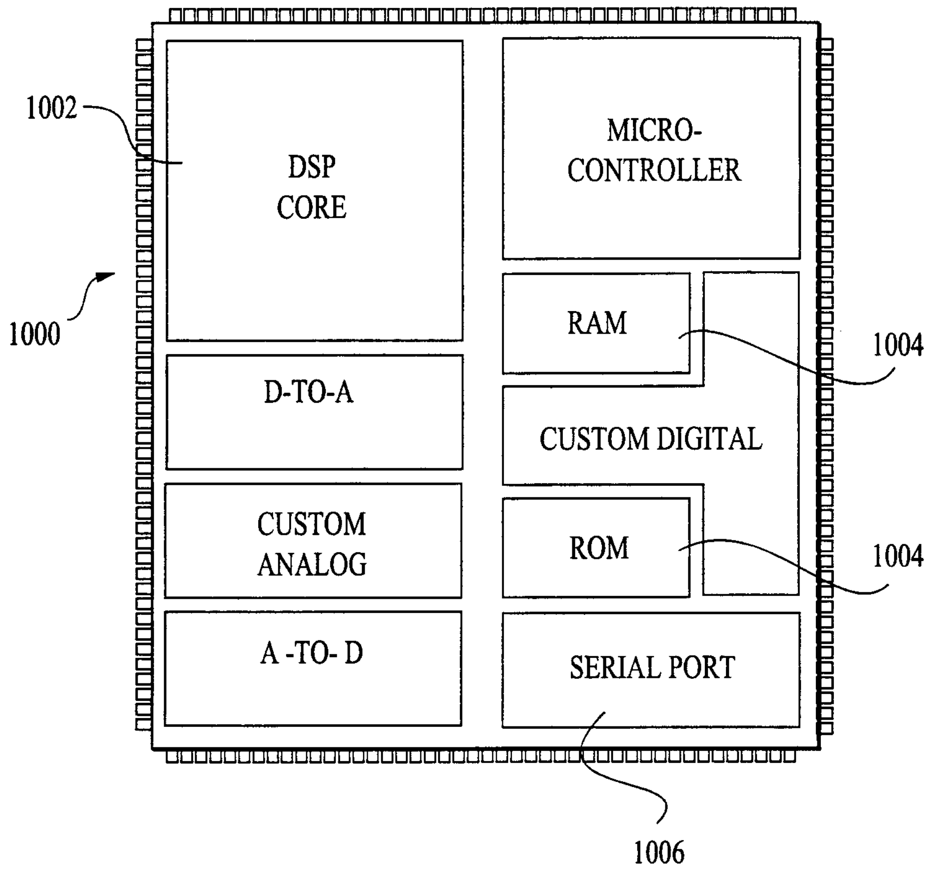


FIG. 10

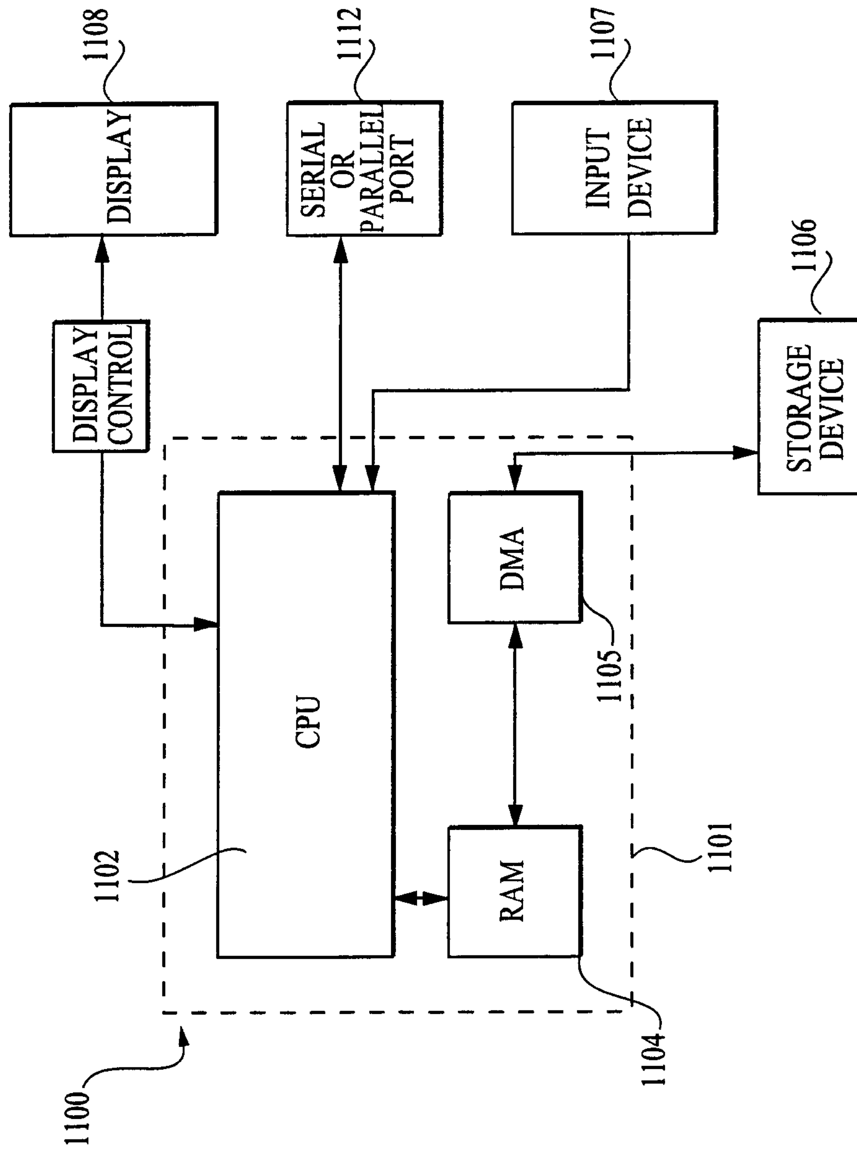


FIG. 11