

國立交通大學

電機學院與資訊學院 資訊學程

碩士論文

32-BIT RISC CPU 實作-使用 ASIP 方式

32-Bit RISC CPU implementation – Using ASIP approach



研究生：謝宜軒

指導教授：陳昌居 教授

中華民國九十五年六月

32-BIT RISC CPU 實作-使用 ASIP 方式

32-Bit RISC CPU implementation - Using ASIP approach

研究生：謝宜軒

Student：I-Hsuan Hsieh

指導教授：陳昌居

Advisor：Chang-Jiu Chen

國立交通大學
電機學院與資訊學院專班 資訊學程
碩士論文



Submitted to Degree Program of Electrical Engineering and Computer Science

College of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

32-BIT RISC CPU 實作 - 使用 ASIP 方式

學生：謝宜軒

指導教授：陳昌居

國立交通大學 電機學院與資訊學院 資訊學程 (研究所) 碩士班

摘 要

本論文研製之 ASIP (Application Specific Instruction Set Processor) 於目前的 SOC 中使用的相當廣泛，一個 SOC 中有時使用一個以上的 ASIP。而 ASIP 與一般的 general purpose Processor 最大的不同，在於 ASIP 執行的程式一般都是固定於 chip 上，而非於 DRAM, SRAM 上的不同程式。所以為了達到 CHIP area 與 Performance 的需求，ASIP 的 generation tool 都可以方便使用者自行定義 instruction set, 內部的 pipe line stage, function unit、等。在定義好之後，可以自動產生 HDL source 及 compiler, simulator 等 tool set, 使的 user 可以針對某個特殊應用來設計所需的 ASIP 使用。

我們經過評估後決定使用日本 Osaka University 的 ASIP Meister tool (<http://www.eda-meister.org/>), Tool 這套可以產生 VHDL RTL code 及 C compiler generator files for COSY compiler generator (<http://www.ace.nl>)。ASIP 在定義指令集及架構採用 graphic GUI, 並且 tool 可以在 RTL code 合成前就能預估出 gate count, power, speed 供參考。我們預備以 32-bit RISC 架構來設計一個 ASIP, 並實做出一個晶片。對 ASIP 的設計流程作一完整實作及瞭解，並驗證 ASIP meister Tool 功能。對於在 SOC 設計例如網路、圖形、數位電視、等應用需要 ASIP 時，可供參考。

設計的 CPU 為 32 bit RISC, 5 stage pipe line, instruction set 以 OPEN source 的 OPEN RISC (<http://www.opencores.org/projects.cgi/web/or1k>) 為架構。因為我們無法取得 COSY compiler generator, 因此必須使用 OPEN RISC 的 GNU tool chain 的 C compiler. 因為無 COSY compiler generator, 使我們在 instruction set 的調整受限，但是可以評估以 ASIP 方式的 CPU 實作結果。

在得到 ASIP 產生的 RTL code 之後，我們再以 Synopsys 等 CIC (www.cic.org.tw) 提供的工具以 UMC .18 的 cell lib, 依據 CIC 的要求，進行 Scan chain insert, IC layout APR (Auto Place and Route), DRC/LVS 等完整 IC 下線的工作。

- 32-Bit RISC CPU implementation - Using ASIP approach

Student : I-Hsuan Hsieh

Advisors : Dr.Chang-Jiu Chang

Degree Program of Electrical Engineering and Computer Science National Chiao Tung University

ABSTRACT

ASIP(Application Instruction Set Processor) are commonly used in today's SOC design. Some SOC chips use more than one ASIPs. The most difference between ASIP and a general purpose CPU is that ASIP usually execute instructions fixed in chip instead of different programs in RAM. To optimize the chip area and performance, ASIP tool can let user generate own instruction set, pipe line state, function unit. And generate RTL code, compiler, simulator of the dedicated ASIP to be used.

We select the ASIP meister tool (<http://www.eda-meister.org/>) from Osaka University as our ASIP design tool. Which can generate VHDL source code and C compiler generator files for COSY compiler generator (<http://www.ace.nl>). The ASIP meister use graphic interface in defining instructions and pipe line stages. And can estimate the gate count, power, performance before chip implementation. We will define a 32-bit RISC CPU, and generate RTL code and implement real chip. For verify ASIP design flow and the ASIP meister tool. For SOC design such as network, graphic, digital, ... applications that need ASIPs, the ASIP methodology can be used.

Our ASIP is a 5 stage 32bit RISC, and the instruction set is from open source OR1K (<http://www.opencores.org/projects.cgi/web/or1k>). Since we could not get the COSY compiler generator, we have to use the OR1K GNU C compiler. So we are not able to refine instructions, but we still can evaluate the ASIP approach for a CPU design.

After the RTL code, we will prepare the chip implementation. We will use UMC .18 um cell lib to synthesis by Synopsis, and other tools from CIC to do scan chain insert, APR(Auto Place and Route), DRC/LVS for chip tape out requirement by CIC.

誌 謝

感謝老師的耐心指導，及國家晶片中心

/opencore.org/eda-meister.org 等提供的諮詢與協助!感謝我的父母

讓我有今日的一切，最後謝謝支持我的妻子!



目 錄

| | | |
|------|-----------------------------|-----|
| 中文提要 | | i |
| 英文提要 | | ii |
| 誌謝 | | iii |
| 目錄 | | iv |
| 表目錄 | | v |
| 圖目錄 | | vi |
| 一、 | 緒論..... | 1 |
| 1.1 | 動機..... | 1 |
| 1.2 | ASIP 的優點..... | 2 |
| 1.3 | ASIP 的工具..... | 2 |
| 二、 | 使用 ASIP-Meister..... | 4 |
| 2.1 | ASIP-Meister 簡介..... | 4 |
| 2.2 | 定義指令集..... | 5 |
| 2.3 | 微運算..... | 7 |
| 三、 | OR1K Processor..... | 8 |
| 3.1 | OR1K 架構簡介..... | 8 |
| 3.2 | OR1K ORBAS32 I 指令集..... | 9 |
| 四、 | OR1K 以 ASIP-Meister 實作..... | 13 |
| 4.1 | 設計目標及架構..... | 13 |
| 4.2 | 系統資源..... | 13 |
| 4.3 | 儲存資源..... | 16 |
| 4.4 | 介面定義..... | 18 |
| 4.5 | 指令的類別與定義..... | 19 |
| 4.6 | C 語言類別與指令行為描述..... | 20 |
| 4.7 | 指令微運算定義..... | 21 |
| 4.8 | 手動的修改..... | 24 |
| 4.9 | 產生的 VHDL 原始碼..... | 26 |
| 4.10 | 驗證方式..... | 27 |
| 五、 | 晶片實作..... | 29 |
| 5.1 | 設計流程..... | 29 |
| 5.2 | 模擬結果..... | 29 |
| 5.3 | 晶片規格..... | 31 |
| 5.4 | 佈局結果錯誤說明..... | 32 |
| 5.5 | 測試規劃..... | 33 |
| 六、 | 結論..... | 34 |
| 參考文獻 | | 36 |

| | | |
|-----|------------------------------|----|
| 附件一 | Test program..... | 37 |
| 附件二 | Software simulation log..... | 44 |



表目錄

| | | |
|-----|--------------------------------|----|
| 表—1 | PC 單元的功能..... | 16 |
| 表—2 | ALU 單元的介面..... | 16 |
| 表—3 | Data Hazard 處理..... | 25 |
| 表—4 | CPU Top VHDL 模組..... | 26 |
| 表—5 | Test Program Simulator 結果..... | 28 |
| 表—6 | 佈局結果錯誤..... | 32 |
| 表—7 | R3000 設計時間..... | 34 |
| 表—8 | R3000 合成結果..... | 35 |



圖目錄

| | | |
|------|-------------------------------|----|
| 圖—1 | 設計目標 | 13 |
| 圖—2 | 系統資源 | 14 |
| 圖—3 | 儲存資源 | 16 |
| 圖—4 | 暫存器檔 | 17 |
| 圖—5 | 暫存器 | 17 |
| 圖—6 | 記憶體 | 17 |
| 圖—7 | 介面定義 | 18 |
| 圖—8 | 指令類別 | 19 |
| 圖—9 | 指令格式 | 20 |
| 圖—10 | C 類別寬度 | 21 |
| 圖—11 | 指令行為描述 | 21 |
| 圖—12 | 指令微運算定義 | 22 |
| 圖—13 | 巨指令展開後的微運算 | 23 |
| 圖—14 | 例外處理 | 23 |
| 圖—15 | 巨指令 Fetch | 24 |
| 圖—16 | CPU Top 方塊圖 | 27 |
| 圖—17 | Test Program Modelsim 結果 | 28 |
| 圖—18 | RTL with scan cell simulation | 29 |
| 圖—18 | Gate level simulation | 30 |
| 圖—20 | Post layout simulation | 30 |
| 圖—21 | Chip Layout | 31 |
| 圖—22 | 驗證系統方塊 | 33 |

1.1 動機

在今天的 SOC 晶片中，經常需要一個以上的 processor，例如 TI OMAP 有 ARM processor 加上 TI 的 DS。有些網路晶片上有的使用數個 ARM 7 來作不同的 network layer protocol 的運算，根據不同的需求設計者必須挑不同的 Processor 來整合到 SOC 的晶片內。但是不同的應用需要的 bit 寬度及指令集，function unit 都不一樣，常造成大材小用的狀況。一來會產生 chip area 的浪費，二來 IP 授權的費用也不低，因此有所謂 re-configurable processor 及 ASIP (Application Specific Instruction Set Processor) 的產生，一般而言 re-configurable processor 的架構限制較大，可以調整的範圍較小。而 ASIP 的調整則視 ASIP tool 而定，常見及較著名的 ASIP tools 如 [Covare Processor Designer](#)[1], Target Compiler Technologies NV 的 [chess/checker tool suit](#)[2], ASIP solution 的 [ASIP-meister](#)[3] tool 都可以產生 C compiler 及 HDL source code 可以合成為 FPGA/ASIC。另外也有些 Tool 如 [Expression](#)[4] 可以產生 Compiler/Simulator, [Sim-nml](#)[5] 可以產生 Compiler/Simulator/High Level Synthesis。我們針對可以合成品片的 ASIP 工具中，向 ASIP-meister 商品化前的日本 Osaka University 要了 ASIP-meister 的軟體，來進行一個 32-bit RISC 的 CPU 設計實作。用來評估實際 ASIP tool 於 ASIP 設計上的方便性及效能的結果。

而選用的 CPU 架構，我們挑了沒有商業授權問題的 [OR1K](#)[6] 來作為 CPU 架構，在指令集我們挑了 OR1K BAS32 的子集，除了兩個關於 SPR (Special Purpose Register) 的指令沒 implement 外，其他所有指令都以 ASIP-meister 完成。產生的 VHDL code，我們接著以 CIC 提供的 IC tool，進一步的進行以 UMC .18um cell 來合成，然後加 scan chain，再進行 APR (Auto Place and Route) 的工作，最後在作 DRC/LVS 等驗證工作，完成送 CIC 下線的所有預備工作。並瞭解 ASIP 完成的 32-bit RISC CPU 的成果如何。

在 ASIP 的工具及應用上，國外都有前述許多的研究及實際的應用例子。在國內則有相關的研究有：

(1) 應用於網路路由器之低功率特定指令集處理器設計，(2) 針對語音解碼處理之精簡指令集處理器架構，(3) 特殊應用系統整合處理器軟硬體輔成設計環境於多媒體應用之最佳化設計與實作，(4) 實作可客製化嵌入式處理器與其發展工具，(5) ASIP 的 GCC 移植工具。其中(3)，(5)是軟體方面的研究，(1)，(2)，(5)是Processor硬體方面的研究，(2)，(5)]分別以ARM，Altera Nios為主Processor加上相關加速的硬件及指令完成。(1)以Expression作架構探索但是並無法以Expression直接產生可合成的Processor RTL code。

1.2 ASIP 的優點

使用 ASIP 相較於使用一般的 Processor, 可以有以下優點。第一, 可以針對特殊的應用來決定 ASIP 的架構。第二, 可以把不必要的功能及指令去除, 達到最低的成本。例如, DSP 常用的 Multiply and accumulate 指令若有需要及可以加入, 而 pipe line stage 可以視運算的速度極限而定, 若要快則必須適當的增加 pipe line stage, 相對的 Die 面積就會增加。相反的要減少 stage, 一般速度變慢, 但是 Die 面積不一定會降低, 必須視實際狀況而定。在執行的軟體確定後, 可以把沒用的指令拿掉, 例如乘法的指令及除法的指令。ASIP 的特性就是這個 Processor 就是針對這個 Application 只要讓這個特殊的 Application 能夠正常運作即可, 也不必用超過此 Application 的硬體來完成。例如 MP3 的 processor 一般是 16 bit, 而不需用到 32 bit, 因為 16 bit 的 DAC 的精度已足夠, 用不著也沒有 32 bit 的 DAC。

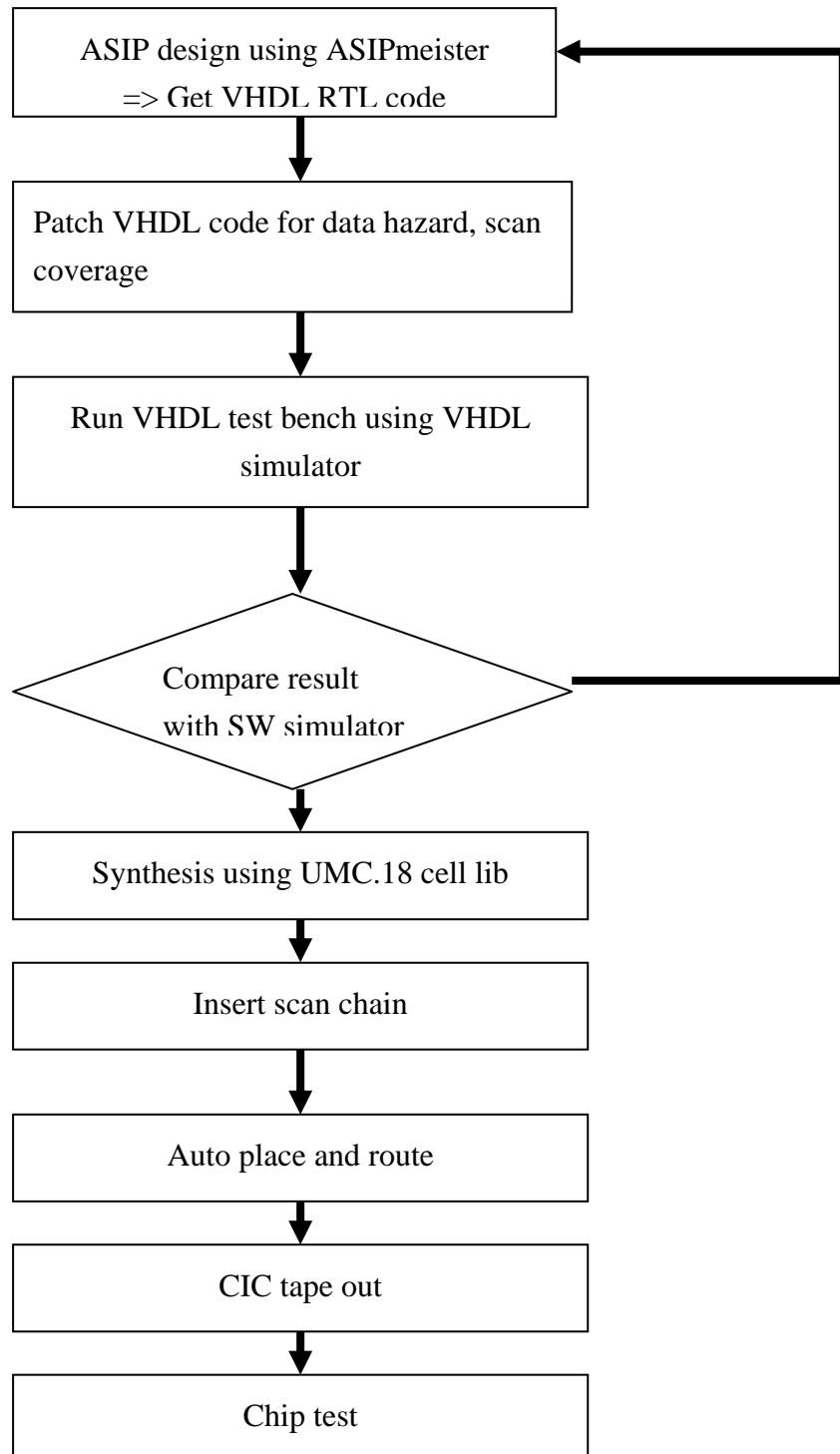
1.3 ASIP 的工具

要能快速的產生使用者定義的 ASIP, 及產生配套的軟體工具如 C Compiler, simulator, HDL code (Verilog or VHDL) 就必須靠 ASIP 的工具。Cware Processor Designer 來自學術界的 LISA, 而 ASIPmeister 也即將由學術研究而商業化, Chess/Checker 則為 IMEC (<http://www.imec.be/>) 的研究產物。因為要是設計 ASIP 的 Tool 必須能: 第一, 有一套定義 CPU 架構的方法, Cware 採用的是 LISA, 可以定義 RISC, DSP, VLIW 等較複雜的架構, 而 Chess/Checker 採用的是 nML, 同樣的也可以定義非 RISC 的 CPU。LISA 及 nML 都是 language base, 而 ASIPmeister 則是 graphic based, 因此只能定義 RISC like 的 processor。先有一套定義 processor 架構, 指令集的工具後, 使用者才能依此定義出使用者需要的 CPU。NML 的演化版 Sim-NML 則是 INDIAN INSTITUTE OF TECHNOLOGY KANPUR 所定義, 並且有許多的 processor example 如 powerPC603 的 Sim-NML model。如果 Processor 內部 pipe line 架構, 運算單位, 定址法, 指令集都無法精準的描述出來, 自然就無法以此產生 C compiler, Simulator, HDL code。

也因為此工具的領域橫跨 Compiler back-end code generation, processor 架構, VLSI design。因此 Tool 開發都經歷相當長的時間, 也因此大都是由學術界來開始研究並加以商品化的。Cware 及 Target 的 Tool 都可以產生 Compiler/Simulator/HDL code, 而 ASIPmeister 只能產生 Compiler/HDL code, 並無法產生 Simulator。而且 ASIPmeister 產生的為 COSY compiler generator 所使用的描述檔, 必須有 COSY 的 Tool 才能得

到 C compiler。但是 ASIPmeister 有開放試用版給學術界使用，因此我們選擇了 ASIPmeister 為設計的工具。

實作流程如下：



二·使用 ASIP-Meister

2.1 ASIP-Meister 簡介

關於仔細的 ASIP Meister 的使用方法，可以參考 [ASIP Meister User's Manual](#)[7]及 [ASIP Meister Tutorial](#)[8]。在此我們只作簡單及最重要的地方加以介紹。ASIP Meister 設計上分成 11 步驟：

- Design Goal & Arch Design：設定 Pipe line stage 及各個 stage 的功能
- Resource Declaration：調用 ASIP Meister 內建的各種 function block 如 register, register file, PC unit, ALU unit, DMAU(Data Memory Adress Unit.)並設定調用的參數，如 ADDER 的 bit width 及 type(Carry look ahead, ...)等。
- Storage Spec：內部具循序的單元宣告，register, register file, PC, DMAU .. 等，並說明其屬性，例如 GPR0 為 zero register, GPR28 為 stack pointer.. 等。
- Interface Definition：CPU 最外層的 I/O port 定義。
- Instruction Definition：包括了 instruction type 及 instruction 的定義。你必須先定義好有幾種 instruction type 然後再來一個一個的定義指令。每個 type 你必須定義好，type name，指令中有幾個 field, 各個 field 的寬度及屬性都要先定好。再加入指令時，必須先叫出是什麼 type, 再去定義每個 field 直於本指令中的值或代表意義，最後定義組合語言的格式。
- Arch. Level Estimations：可以用來粗估本 ASIP 的面積，延遲，功耗。
- C Definition：告訴 C compiler 的各種變量的寬度。ilers 所用到，若不產生 C compiler 此部分可以不給。
- Behavior Description：用來產生 C compiler，類似 Micro OP Description, 但是一個是用來產生 VHDL 的硬體設計，而 Behavior description 則是在產生給 COSY 的 description 使用。
- Micro OP Description：這是最重要的部分，定義 ASIP 微指令(micro code)的寫法，用來說明個個 stage 中的各個硬體動作。

- HDL Generation : 最後的步驟用來產生 VHDL source code , 可以產生給 Simulation 用的 source 及可以合成硬體的 source 。
- Swdev Generation : 產生給 COSY compiler generator 的檔案 , 用來產生 C compiler.

2.2 定義指令集

這是產生 CPU 硬體 HDL code 及 軟體 Compiler 前的最基本動作。首先你必須把所有指令的位元組類似的指令先 group 起來 , 定義成指令的類別。例如 ASIP Meisters 所附的 Demo project DLX CPU , DLX 是所有修計算機架構的人都很熟悉的 CPU , 是經典的 RISC processor 。

以 DLX 而言有 7 大類的指令 :

- R—R : Register, Register.
 - 6 bit OP code
 - 5 bit rs0 : Reg direct, source 0
 - 5 bit rs1 : Reg direct, source 1
 - 5 bit rd : Reg direct, destination
 - 11 bit func : Operation function type.
- R_I : Register, Immediate.
 - 6 bit OP code
 - 5 bit rs0 : Reg direct, source 0
 - 5 bit rd : Reg direct, destination
 - 16 bit const : Immediate data
- L_S : Load, Store.
 - 6 bit OP code
 - 5 bit rs0 : Reg indirect with displacement, reg of address
 - 5 bit rd : Reg direct, source/destination

- 16 bit const : Reg indirect with displacement, displacement of address
- B : Relative branch
 - 6 bit OP code
 - 5 bit rs0 : Reg direct , source
 - 5 bit “00000” : Fixed to zero
 - 16 bit const : PC relative, displacement to PC
- J : Absolutely jump
 - 6 bit OP code
 - 26 bit const : Absolute address
- J-R : Jump register.
 - 6 bit OP code
 - 5 bit rs0 : Reg direct , source
 - 10 bit zero : Fixed to zero.
 - 11 bit func : Operation function type.
- LHI : Load high
 - 6 bit OP code
 - 5 bit “00000” : Fixed to zero
 - 5 bit rd : Reg direct, destination
 - 16 bit const: Immediate data

第一步就是依照 ASIP Meister 的介面把這幾種指令類別建好，然後再將所有的指令一個個的輸入例如，ADD 的指令屬於 R—R 的類別，你必須輸入 6 bit OP code 的值 “000000”，及 11 bit function 的值”00000100000”，及組合語言的格式”ADD rs0 rs1 rd”即可，其他的位元關於兩個 source, 一個 destination 的 register select 於先前的指令類定義時即有資訊不用再輸入。因此先建好指令的類別，可以加快後面的個別指令的輸入。

另外 A S I P Meister 不允許同一個位元組拆成不連續的位元位置，例如 J type 的指令你不可以把 26 bit const 拆成兩組 bit 0~5 及 bit 11~31，而將 op 擺在 bit 6~10。26 bit const 必須是連續性的 bit 位置。

2.3 微運算

在所有的指令都定義好之後，幾可以預估 ASIP 的 area, speed, power 的表現。之後，必須針對 C compiler 的產生，我們必須告知 ASIP Meister 兩個資訊，C 語言的變量 Bit 數（例如 char type, int type, ...），然後完成 Behavior 的描述，如此 ASIP Meistera 才能據以產生給 COSY Compiler Generator 用的相關檔案。這部分我們就不詳細說明，把重點擺在微指令上。

微運算(Micro Operation)在此係指各個指令於所有的 pile line stage 所必須作的硬體動作，是以硬體邏輯(hard-wired)方式完成，並非以內部的 ROM 存放的方式的微指令(Micro code)方式，用來加快速度。基本上所有的儲存單元都必須在 Storage Spec 或 Resource Declaration 中先宣告，在微運算你可以依照 ASIP Meister 定義的微運算語法來描述在各個 stage 時，你所定義使用的 Resource/Storage 將如何運作來產生你所需求的動作。

一些常見的一連串的微運算你可以定義成巨集(Macro)，只要在巨集開始的那個 stage 呼叫此巨集，A S I P Meister 會自動把巨集定義的微運算 include 進來，如此可以省下使用者的時間。詳細的微運算定義方式可以參考 ASIP Meister Tutorial 及 Micro-Operation Description Coding Guide[8]。以下我們舉 D L X 為例列出其完整的微運算定義（件附錄一）。

關於巨集，必須注意的是巨集中的 stage 1 並非指絕對 stage 編號，而是相對於你呼叫 Macro 的 stage，例如 D L X 微運算巨集 FETCH() 只有一個 stage, stage 1。如果你把 FETCH() Macro 擺在某指令的 stage 2，那關於 FETCH() 中定義 stage 1 的動作實際會在某指令的 stage 2 中執行。在微指令的定義除了，各個 stage 外還有一欄 variable 是可以用來宣告連接各個 Resource/Storage 的 wire 用的，各個 stage 中也可以宣告 wire，但是跨 stage 的連接線就必須擺在 variable。和 Verilog 的語法類似，wire 只能是連接線不能有儲存信號的功能，即便是一個 1 bit 的 D flip-flop 都必須事先在 Resource Declaration 中宣告。

三 · OR1K Processor

3.1 OR1K 架構簡介

OR1K 是在 Open Source 的領域內較成熟的 R I S C 架構及有較多數的資源，並且有經過實際 F P G A / A S I C 驗。設計與 DLX, 及 M I P S R3000 十分相近。並且有完成 simulator, GNU C compiler, RTOS porting.. 等相關的工具於其網站上可以獲得。因此我們選用 **OR1K 的指令集**[9] 來設計我們的 A S I P。

- 線性，32/64 位元邏輯地址。實際地址大小視設計而定。
- 固定長度指令集，並有下列不同指令集：
 - 32/64 位元基本指令集 (ORBIS32/64)：指令長度 32 位元，對齊 32 位元邊界，運算資料 32/64 位元。
 - 向量/數位信號處理延伸指令 (ORVFX64)：指令長度 32 位元，對齊 32 位元邊界，運算資料 8/16/32/64 位元。
 - 浮點運算延伸指令 (ORFPX32/64)：指令長度 32 位元，對齊 32 位元邊界，運算資料 32/64 位元。
- 兩種簡單定址方式：
 - 暫存器加 16 位元有號數值。
 - 暫存器加 16 位元有號數值，並更新暫存器值為計算後值。
- 大多數指令有兩個暫存器運算元（或一個暫存器和一個常數），結果置於另一個暫存器。
- 32 個通用暫存器。
- 跳躍指令有一個指令的延遲使流水線儘量充滿。
- 0V flag 存放算數運算 overflow. CY flag 存放算數運算 carry. F flag 存放比較結果。

3.2 OR1K ORBAS32 I 指令集

- 1.add rD, rA, rB : Add rA, rB to rD. Update OV, CY.
- 1.addi rD, rA, I : Add rA, sign extended 15 bit constant to rD. Update OV, CY.
- 1.and rD, rA, rB : logic and rA , rB to rD.
- 1.andi rD, rA, K : logic and rA, zero extended 15 bit constant to rD.
- 1.bf N : branch to current PC plus signed extended 26 bit constant << 2 if F flag is set.
- 1.bnf N : branch to current PC plus signed extended 26 bit constant << 2 if F flag is cleared.
- 1.j N : jump to current PC plus signed extended 26 bit constant << 2.
- 1.jal N : jump to current PC plus signed extended 26 bit constant << 2 and set LR(link register) to next instruction address after delay slot.
- 1.jalr rB : jump to address stored in rB and set LR(link register) to next instruction address after delay slot.
- 1.jr rB : jump to address stored in rB.
- 1.lbs rD, I[rA] : load byte which address calculated by rA plus signed extended 16 bit constant, sign extend to 32 bit and store to rD.
- 1.lbz rD, I[rA] : load byte which address calculated by rA plus signed extended 16 bit constant, zero extend to 32 bit and store to rD.
- 1.lhs rD, I[rA] : load half word which address calculated by rA plus signed extended 16 bit constant, sign extend to 32 bit and store to rD.
- 1.lhz rD, I[rA] : load half word which address calculated by rA plus

signed extended 16 bit constant, zero extend to 32 bit and store to rD.

- 1.lws rD, I[rA] : load word which address calculated by rA plus signed extended 16 bit constant, and store to rD.
- 1.lwz rD, I[rA] : load word which address calculated by rA plus signed extended 16 bit constant, and store to rD.
- *1.mfspr : move from Special-Prupose Register(not implemented).
- 1.movhi rD, K : move zero extended 16 bit constant << 16 to rD.
- *1.mtspr : move to Special-Prupose Register(not implemented).
- 1.mul rD, rA, rB : multiply signed rA, rB and store to rD(contracated). Update OV, CY.
- 1.muli rD, rA, I : multiply signed rA, signed 16 bit constant and store to rD(trancated). Update OV, CY.
- 1.mulu rD, rA, rB : multiply unsigned rA, rB and store to rD(contracated). Update OV, CY.
- 1.nop : no operation.
- 1.or rD, rA, rB : logic or rA , rB to rD.
- 1.ori rD, rA, K : logic or rA, zero extended 15 bit constant to rD.
- *1.rfe : return from exception(not implemented).
- 1.rori rD, rA, L : rotate right rA with 6-bit immediately value and store to rD.
- 1.sb I[rA], rB : store byte of rB to address calculated by rA plus signed extended 16 bit constant.
- 1.sfeq rA, rB : set F flag if rA==rB.
- 1.sfges rA, rB : set F flag if rA>=rB singed.
- 1.sfgeu rA, rB : set F flag if rA>=rB uninged.
- 1.sfgts rA, rB : set F flag if rA>rB singed.

- 1. sfgtu rA, rB : set F flag if rA>rB unsinged.
- 1. sfles rA, rB : set F flag if rA<=rB singed.
- 1. sfleu rA, rB : set F flag if rA<=rB unsinged.
- 1. sflts rA, rB : set F flag if rA<rB singed.
- 1. sfltu rA, rB : set F flag if rA<rB unsinged.
- 1. sfnes rA, rB : set F flag if rA!=rB.
- 1. sh I[rA], rB : store half word of rB to address calculated by rA plus signed extended 16 bit constant.
- 1. sll rD, rA, rB : shift left logical rA with 6-bit value in rB[4:0] and store to rD. zero is shifted to rA[0] during shifting.
- 1. slli rD, rA, L : shift left logical rA with 6-bit immediate value L and store to rD. zero is shifted in to rA[0] during shifting.
- 1. sra rD, rA, rB : shift right arithmetic rA with 6-bit value in rB[4:0] and store to rD. rA[31] is shifted in to rA[31] during shifting.
- 1. srai rD, rA, L : shift right arithmetic rA with 6-bit immediate value L and store to rD. rA[31] is shifted in to rA[31] during shifting.
- 1. srl rD, rA, rB : shift right logical rA with 6-bit value in rB[4:0] and store to rD. zero is shifted to rA[31] during shifting.
- 1. srli rD, rA, L : shift right logical rA with 6-bit immediate value L and store to rD. zero is shifted to rA[31] during shifting.
- 1. sub rD, rA, rB : subtract rB from rA and store result to rD. Update CY, OV.
- 1. sw I[rA], rB : store word of rB to address calculated by rA plus signed extended 16 bit constant.
- 1. xor rD, rA, rB : logic xor rA , rB to rD.
- 1. xori rD, rA, K : logic xor rA, signed extended 15 bit constant to

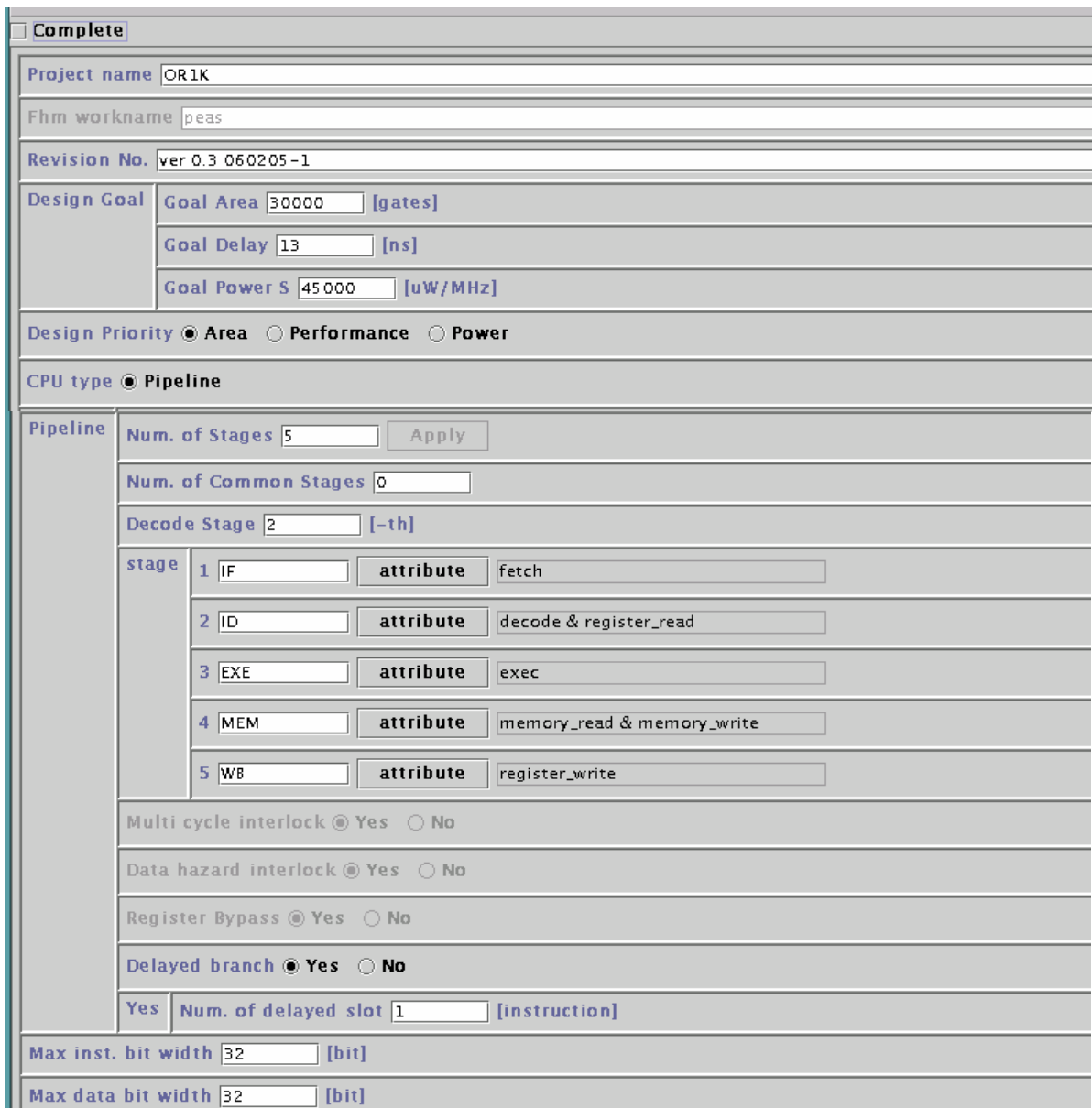
rD.



四 · OR1K 以 ASIP-Meister 實作

4.1 設計目標及架構

如圖-1 中，我們的是設計目標希望 gate count 在 30K 以內，delay 13ns. 流水線是常見的 5 stage, 安排也是很標準的 fetch, decode, execution, memory, write back. 設計使用 delayed branch, 且 delay slot 是 1 個 stage. 而最大指令及資料的寬度都設成 32。



| | |
|---|---|
| <input type="checkbox"/> Complete | |
| Project name | OR1K |
| Fhm workname | peas |
| Revision No. | ver 0.3 060205-1 |
| Design Goal | Goal Area <input type="text" value="30000"/> [gates] |
| | Goal Delay <input type="text" value="13"/> [ns] |
| | Goal Power S <input type="text" value="45000"/> [uW/MHz] |
| Design Priority <input checked="" type="radio"/> Area <input type="radio"/> Performance <input type="radio"/> Power | |
| CPU type <input checked="" type="radio"/> Pipeline | |
| Pipeline | Num. of Stages <input type="text" value="5"/> <input type="button" value="Apply"/> |
| | Num. of Common Stages <input type="text" value="0"/> |
| | Decode Stage <input type="text" value="2"/> [-th] |
| | stage 1 <input type="text" value="IF"/> <input type="button" value="attribute"/> <input type="text" value="fetch"/> |
| | 2 <input type="text" value="ID"/> <input type="button" value="attribute"/> <input type="text" value="decode & register_read"/> |
| | 3 <input type="text" value="EXE"/> <input type="button" value="attribute"/> <input type="text" value="exec"/> |
| | 4 <input type="text" value="MEM"/> <input type="button" value="attribute"/> <input type="text" value="memory_read & memory_write"/> |
| | 5 <input type="text" value="WB"/> <input type="button" value="attribute"/> <input type="text" value="register_write"/> |
| | Multi cycle interlock <input checked="" type="radio"/> Yes <input type="radio"/> No |
| | Data hazard interlock <input checked="" type="radio"/> Yes <input type="radio"/> No |
| | Register Bypass <input checked="" type="radio"/> Yes <input type="radio"/> No |
| | Delayed branch <input checked="" type="radio"/> Yes <input type="radio"/> No |
| | Yes <input type="button"/> Num. of delayed slot <input type="text" value="1"/> [instruction] |
| | Max inst. bit width <input type="text" value="32"/> [bit] |
| | Max data bit width <input type="text" value="32"/> [bit] |

圖—1 設計目標

4.2 系統資源

在這個階段中我們必須告訴 ASIPmeister, 我們必須用到哪些 ASIPmeister 提供的資源例如圖-2 中, PC, IR(指令暫存器), IMAU (指令地址產生單元), .. 等。ASIPmeister 已經內建好 CPU 常見的系統單元, 在使用時可以依照不同的需求去引用他, 如位元數, 加法的架構等。以 OR1K 而言使用了 PC, IR, IMAU, DMAU, GPR, ALU0, EXT0, MULO, SFT0, EXT1, CY, OV, F, SFT1, ADDER0 這些單元。至於這些單元可以有哪些功能及介面方式, 可以在 Function Set 及 Port Set 的欄位中瞭解。Function Set 中你可以理解這個單元有那些功能可以於定義指令的微指令時使用, 以 PC 單元為例在表 4-1 中我們可以明白 PU 單元可以有 nop, reset, inc, write, read 等 5 項功能可以使用。於 ALU 單元 Port Set 表 4-2 中我們可以明白在 VHDL 的模組中會有哪些 I/O 信號。

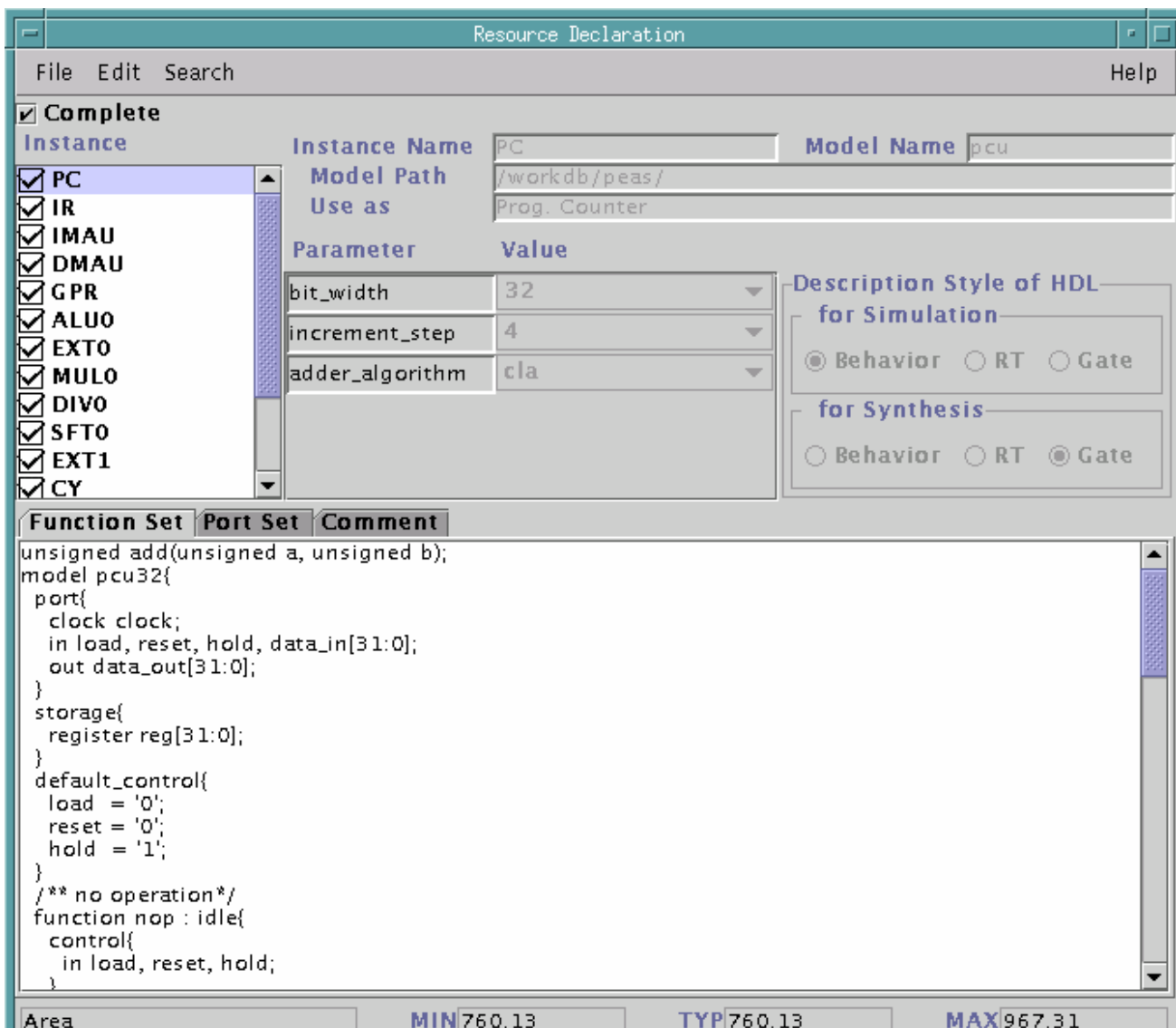


圖-2 系統資源

Function Set OF PC:

```
unsigned add(unsigned a, unsigned b);
model pcu32{
  port{
    clock clock;
    in load, reset, hold, data_in[31:0];
    out data_out[31:0];
  }
  storage{
    register reg[31:0];
  }
  default_control{
    load = '0';
    reset = '0';
    hold = '1';
  }
  /** no operation*/
  function nop : idle{
    control{
      in load, reset, hold;
    }
    protocol{
      [load == 0 && reset == 0 && hold == 1]{
      }
    }
  }
  /** reset */
  function reset : reset{
    assignment{
      reg = 0;
    }
    control{
      in reset;
    }
    protocol{
      [reset == 1]{
        store reg;
      }
    }
  }
  /** increment */
  function inc{
    assignment{
      reg = add(reg, 4);
    }
  }
}
```



```

/** write : set program counter value */
function write{
  input{
    bit_vector data_in;
  }
  assignment{
    reg = data_in;
  }
  control{
    in bit load;
  }
  protocol{
    [load = '1' && hold data_in]{
      store reg;
    }
  }
}
/** read : read program counter value */
function read{
  output{
    bit_vector data_out;
  }
}
priority{ ( reset > ( inc | write ) ), read}
}

```

表-1 PC 單元的功能

Port Set of ALU:

```

a   in   bit_vector  31  0data
b   in   bit_vector  31  0data
cin in   bit         data
result out bit_vector 31  0data
cout out bit         data

```

表-2 ALU 單元的介面

4.3 儲存資源

在這個設計階段，我們必須定義好 Processor 內所有的儲存資源包括暫存器檔，暫存器，記憶體如圖-3 所示。

| Storage Name | Resource | Bit Width | Usage | Location | Binary |
|--------------|----------|-----------|----------|----------|-----------|
| GPR[asc] | GPR | 32 | register | original | [bin-asc] |

圖-3 儲存資源

至於暫存器檔 (register file) 如圖-4 所示宣告了 32 個暫存器。而在圖-5 中我們可以定義所用到的暫存器，其中可以有在暫存器檔中的，例如 GPR9 是用來當 link register 的，必須在這裡定義。而圖-6 中，我們定義了記憶體使用的指令記憶體 (IM) 及資料記憶體 (DM)。

| Storage Name | Register Class | Resource | Bit Width | Register Number | Usage | Location | Binary |
|--------------|----------------|----------|-----------|-----------------|----------|----------|--------|
| GPR0 | GPR | GPR | 32 | 0 | register | original | 00000 |
| GPR1 | GPR | GPR | 32 | 1 | register | original | 00001 |
| GPR2 | GPR | GPR | 32 | 2 | register | original | 00010 |
| GPR3 | GPR | GPR | 32 | 3 | register | original | 00011 |
| GPR4 | GPR | GPR | 32 | 4 | register | original | 00100 |
| GPR5 | GPR | GPR | 32 | 5 | register | original | 00101 |
| GPR6 | GPR | GPR | 32 | 6 | register | original | 00110 |
| GPR7 | GPR | GPR | 32 | 7 | register | original | 00111 |
| GPR8 | GPR | GPR | 32 | 8 | register | original | 01000 |
| GPR9 | GPR | GPR | 32 | 9 | register | original | 01001 |
| GPR10 | GPR | GPR | 32 | 10 | register | original | 01010 |
| GPR11 | GPR | GPR | 32 | 11 | register | original | 01011 |
| GPR12 | GPR | GPR | 32 | 12 | register | original | 01100 |
| GPR13 | GPR | GPR | 32 | 13 | register | original | 01101 |
| GPR14 | GPR | GPR | 32 | 14 | register | original | 01110 |
| GPR15 | GPR | GPR | 32 | 15 | register | original | 01111 |
| GPR16 | GPR | GPR | 32 | 16 | register | original | 10000 |
| GPR17 | GPR | GPR | 32 | 17 | register | original | 10001 |
| GPR18 | GPR | GPR | 32 | 18 | register | original | 10010 |
| GPR19 | GPR | GPR | 32 | 19 | register | original | 10011 |
| GPR20 | GPR | GPR | 32 | 20 | register | original | 10100 |
| GPR21 | GPR | GPR | 32 | 21 | register | original | 10101 |
| GPR22 | GPR | GPR | 32 | 22 | register | original | 10110 |
| GPR23 | GPR | GPR | 32 | 23 | register | original | 10111 |
| GPR24 | GPR | GPR | 32 | 24 | register | original | 11000 |
| GPR25 | GPR | GPR | 32 | 25 | register | original | 11001 |
| GPR26 | GPR | GPR | 32 | 26 | register | original | 11010 |
| GPR27 | GPR | GPR | 32 | 27 | register | original | 11011 |
| GPR28 | GPR | GPR | 32 | 28 | register | original | 11100 |
| GPR29 | GPR | GPR | 32 | 29 | register | original | 11101 |
| GPR30 | GPR | GPR | 32 | 30 | register | original | 11110 |
| GPR31 | GPR | GPR | 32 | 31 | register | original | 11111 |

圖-4 暫存器檔

| Storage Name | Resource | Bit Width | Usage | Location |
|--------------|----------|-----------|----------------------|----------|
| PC | PC | 32 | program counter | original |
| IR | IR | 32 | instruction register | original |
| CY | CY | 1 | carry-flag | original |
| OV | OV | 1 | overflow-flag | original |
| F | F | 1 | register | original |
| SP | GPR | 32 | stack pointer | GPR1 |
| FP | GPR | 32 | frame pointer | GPR2 |
| LR | GPR | 32 | link register | GPR9 |
| RV | GPR | 32 | return register | GPR11 |
| RVH | GPR | 32 | return register | GPR12 |

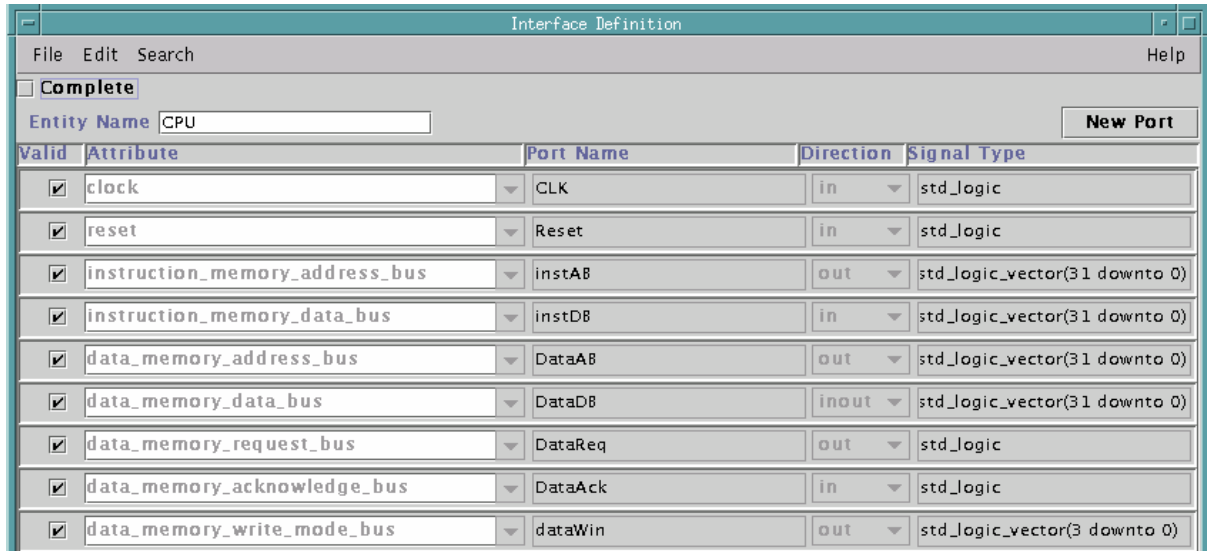
圖-5 暫存器

| Storage Name | Resource | Bit Width | Usage | Access Bit |
|--------------|----------|-----------|--------------------|------------|
| IMEM | IMAU | 32 | instruction memory | 8 |
| DMEM | DMAU | 32 | data memory | 8 |

圖-6 記憶體

4.4 介面定義

在完成前述的步驟後，系統會產生圖-7 的介面定義。由 dataWin 中我們可以知道記憶體雖然整體是 32 位元，但 write 是以 8 位元為一單位，所以 write 的信號是 4 bit 的 vector。而 DataReq, 及 DataAck 則是方便存取不同速度的記憶體延長讀寫週期用的。



| Valid | Attribute | Port Name | Direction | Signal Type |
|-------------------------------------|--------------------------------|-----------|-----------|-------------------------------|
| <input checked="" type="checkbox"/> | clock | CLK | in | std_logic |
| <input checked="" type="checkbox"/> | reset | Reset | in | std_logic |
| <input checked="" type="checkbox"/> | instruction_memory_address_bus | instAB | out | std_logic_vector(31 downto 0) |
| <input checked="" type="checkbox"/> | instruction_memory_data_bus | instDB | in | std_logic_vector(31 downto 0) |
| <input checked="" type="checkbox"/> | data_memory_address_bus | DataA8 | out | std_logic_vector(31 downto 0) |
| <input checked="" type="checkbox"/> | data_memory_data_bus | DataD8 | inout | std_logic_vector(31 downto 0) |
| <input checked="" type="checkbox"/> | data_memory_request_bus | DataReq | out | std_logic |
| <input checked="" type="checkbox"/> | data_memory_acknowledge_bus | DataAck | in | std_logic |
| <input checked="" type="checkbox"/> | data_memory_write_mode_bus | dataWin | out | std_logic_vector(3 downto 0) |

圖-7 介面定義



4.5 指令的類別與定義

在圖-8 與圖-9 中我們宣告了了指令的類別及定義，把指令的格式及機器碼都完整的寫出。其中必須注意的是 ASIPmeister 並不允許把一個 field 拆開來，例如一個 16 bit constant，是不允許定義在 bit 0-5 及 bit 15-24。

| Instruction Definition | | | | | | | | | | | | |
|--|----|-----|-----|------------|------------|------------|------------------------|--------------|--------------|----------|----------|-----|
| Edit Search Sort | | | | | | | | | | | | |
| Instruction type/Instruction Exception | | | | | | | | | | | | |
| Instruction Type Definition | | | | | | | | | | | | |
| Name | # | MSB | LSB | Field Type | Field Attr | Name/Value | Addr mode | Operand Name | element | reg clas | | |
| R_R | 1 | 31 | 26 | OP-code | name | opcode | | | | | | |
| | | 25 | 21 | Operand | name | rd | Reg Direct | rd | Resource | GPR | | |
| | | 20 | 16 | Operand | name | rs0 | Reg Direct | rs0 | Resource | GPR | | |
| | | 15 | 11 | Operand | name | rs1 | Reg Direct | rs1 | Resource | GPR | | |
| | | 10 | 10 | Reserved | binary | 0 | | | | | | |
| | | 9 | 6 | OP-code | name | func1 | | | | | | |
| | | 5 | 4 | Reserved | binary | 00 | | | | | | |
| | | 3 | 0 | OP-code | name | func2 | | | | | | |
| | | R_I | 1 | 31 | 26 | OP-code | name | opcode | | | | |
| | | | | 25 | 21 | Operand | name | rd | Reg Direct | rd | Resource | GPR |
| 20 | 16 | | | Operand | name | rs0 | Reg Direct | rs0 | Resource | GPR | | |
| 15 | 0 | | | Operand | name | const | Immediate data | const | Immediate | | | |
| B | 1 | 31 | 26 | OP-code | name | opcode | | | | | | |
| | | 25 | 0 | Operand | name | const | PC relative address | const | Symbol | | | |
| JR | 1 | 31 | 26 | OP-code | name | opcode | | | | | | |
| | | 25 | 16 | Reserved | binary | 000000000 | | | | | | |
| | | 15 | 11 | Operand | name | rs1 | Reg Direct | rs1 | Resource | GPR | | |
| | | 10 | 0 | Reserved | binary | 0000000000 | | | | | | |
| I | 1 | 31 | 26 | OP-code | name | opcode | | | | | | |
| | | 25 | 21 | Operand | name | rd | Reg Direct | rd | Resource | GPR | | |
| | | 20 | 17 | Reserved | binary | 0000 | | | | | | |
| | | 16 | 16 | OP-code | binary | 0 | | | | | | |
| R_L | 1 | 15 | 0 | Operand | name | const | Immediate data | const | Immediate | | | |
| | | 31 | 26 | OP-code | name | opcode | | | | | | |
| | | 25 | 21 | Operand | name | rd | Reg Direct | rd | Resource | GPR | | |
| | | 20 | 16 | Operand | name | rs0 | Reg Direct | rs0 | Resource | GPR | | |
| | | 15 | 8 | Reserved | binary | 00000000 | | | | | | |
| NOP | 1 | 7 | 6 | OP-code | name | func | | | | | | |
| | | 5 | 0 | Operand | name | const | Immediate data | const | Immediate | | | |
| | | 31 | 24 | OP-code | name | opcode | | | | | | |
| SYS | 1 | 23 | 16 | Reserved | binary | 00000000 | | | | | | |
| | | 15 | 0 | Operand | name | const | Immediate data | const | Immediate | | | |
| SETF | 1 | 31 | 16 | OP-code | name | opcode | | | | | | |
| | | 15 | 0 | Operand | name | const | Immediate data | const | Immediate | | | |
| | | 31 | 21 | OP-code | name | opcode | | | | | | |
| | | 20 | 16 | Operand | name | rs0 | Reg Direct | rs0 | Resource | GPR | | |
| ST | 1 | 15 | 11 | Operand | name | rs1 | Reg Direct | rs1 | Resource | GPR | | |
| | | 10 | 0 | Reserved | binary | 0000000000 | | | | | | |
| | | 31 | 26 | OP-code | name | opcode | | | | | | |
| LD | 1 | 25 | 21 | Operand | name | rd | Reg Indirect with Disp | rd | Resource | GPR | | |
| | | 20 | 16 | Operand | name | rs0 | Reg Direct | rs0 | Resource | GPR | | |
| | | 15 | 0 | Operand | name | const | Reg Indirect with Disp | addr | Displacement | | | |
| | | 31 | 26 | OP-code | name | opcode | | | | | | |
| LD | 1 | 25 | 21 | Operand | name | rd | Reg Direct | rd | Resource | GPR | | |
| | | 20 | 16 | Operand | name | rs0 | Reg Indirect with Disp | rs0 | Resource | GPR | | |
| | | 15 | 0 | Operand | name | const | Reg Indirect with Disp | addr | Displacement | | | |

圖-8 指令類別

| | | | | | | | | |
|-------|------|---|-------------------------|-------------------------|-----------|-------------------------|-------------------------|-------------------|
| add | R_R | 1 | 1,1,1,0,0,0 | r,d | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | add rd rs0 rs1 |
| addi | R_I | 1 | 1,0,0,1,1,1 | r,d | r,s,0 | c,ons,t | | addi rd rs0 const |
| and | R_R | 1 | 1,1,1,0,0,0 | r,d | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,1,1 | and rd rs0 rs1 |
| andi | R_I | 1 | 1,0,1,0,0,1 | r,d | r,s,0 | c,ons,t | | andi rd rs0 const |
| bf | B | 1 | 0,0,0,1,0,0 | c,ons,t | | | | bf const |
| bnf | B | 1 | 0,0,0,0,1,1 | c,ons,t | | | | bnf const |
| j | B | 1 | 0,0,0,0,0,0 | c,ons,t | | | | j const |
| jal | B | 1 | 0,0,0,0,0,1 | c,ons,t | | | | jal const |
| jalr | JR | 1 | 0,1,0,0,1,0 | 0,0,0,0,0,0,0,0,0,0,0,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | jalr rs1 | |
| jr | JR | 1 | 0,1,0,0,0,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | jr rs1 | |
| lbs | LD | 1 | 1,0,0,1,0,0 | r,d | r,s,0 | c,ons,t | | lbs rd addr |
| lbz | LD | 1 | 1,0,0,0,1,1 | r,d | r,s,0 | c,ons,t | | lbz rd addr |
| lhs | LD | 1 | 1,0,0,1,1,0 | r,d | r,s,0 | c,ons,t | | lhs rd addr |
| lhz | LD | 1 | 1,0,0,1,0,1 | r,d | r,s,0 | c,ons,t | | lhz rd addr |
| lws | LD | 1 | 1,0,0,0,1,0 | r,d | r,s,0 | c,ons,t | | lws rd addr |
| lwz | LD | 1 | 1,0,0,0,0,1 | r,d | r,s,0 | c,ons,t | | lwz rd addr |
| movhi | I | 1 | 0,0,0,1,1,0 | r,d | 0,0,0,0,0 | c,ons,t | | movhi rd const |
| mul | R_R | 1 | 1,1,1,0,0,0 | r,d | r,s,0 | r,s,1 | 0,1,1,0,0,0,0,0,1,1,0 | mul rd rs0 rs1 |
| muli | R_I | 1 | 1,0,1,1,0,0 | r,d | r,s,0 | c,ons,t | | muli rd rs0 const |
| mulu | R_R | 1 | 1,1,1,0,0,0 | r,d | r,s,0 | r,s,1 | 0,1,1,0,0,0,0,1,0,1,1 | mulu rd rs0 rs1 |
| nop | NOP | 1 | 0,0,0,1,0,1,0,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | c,ons,t | | | nop const |
| or | R_R | 1 | 1,1,1,0,0,0 | r,d | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,1,0,0 | or rd rs0 rs1 |
| ori | R_I | 1 | 1,0,1,0,1,0 | r,d | r,s,0 | c,ons,t | | ori rd rs0 const |
| rori | R_L | 1 | 1,0,1,1,1,0 | r,d | r,s,0 | 0,0,0,0,0,0,0,0,1,1 | c,ons,t | rori rd rs0 const |
| sb | ST | 1 | 1,1,0,1,1,0 | r,d | r,s,0 | c,ons,t | | sb addr rs0 |
| sfeq | SETF | 1 | 1,1,1,0,0,1,0,0,0,0,0,0 | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | sfeq rs0 rs1 | |
| sfges | SETF | 1 | 1,1,1,0,0,1,0,1,0,1,1 | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | sfges rs0 rs1 | |
| sfgeu | SETF | 1 | 1,1,1,0,0,1,0,0,0,1,1 | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | sfgeu rs0 rs1 | |
| sfgts | SETF | 1 | 1,1,1,0,0,1,0,1,0,1,0 | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | sfgts rs0 rs1 | |
| sfgtu | SETF | 1 | 1,1,1,0,0,1,0,0,0,1,0 | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | sfgtu rs0 rs1 | |
| sfles | SETF | 1 | 1,1,1,0,0,1,0,1,1,0,1 | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | sfles rs0 rs1 | |
| sfleu | SETF | 1 | 1,1,1,0,0,1,0,0,1,0,1 | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | sfleu rs0 rs1 | |
| sflts | SETF | 1 | 1,1,1,0,0,1,0,1,1,0,0 | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | sflts rs0 rs1 | |
| sfltu | SETF | 1 | 1,1,1,0,0,1,0,0,1,0,0 | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | sfltu rs0 rs1 | |
| sfne | SETF | 1 | 1,1,1,0,0,1,0,0,0,0,1 | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,0,0,0 | sfne rs0 rs1 | |
| sh | ST | 1 | 1,1,0,1,1,1 | r,d | r,s,0 | c,ons,t | | sh addr rs0 |
| sll | R_R | 1 | 1,1,1,0,0,0 | r,d | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,1,0,0,0 | sll rd rs0 rs1 |
| slli | R_L | 1 | 1,0,1,1,1,0 | r,d | r,s,0 | 0,0,0,0,0,0,0,0,0,1 | c,ons,t | slli rd rs0 const |
| sra | R_R | 1 | 1,1,1,0,0,0 | r,d | r,s,0 | r,s,1 | 0,0,0,1,0,0,0,1,0,0,0 | sra rd rs0 rs1 |
| srai | R_L | 1 | 1,0,1,1,1,0 | r,d | r,s,0 | 0,0,0,0,0,0,0,0,1,0 | c,ons,t | srai rd rs0 const |
| srl | R_R | 1 | 1,1,1,0,0,0 | r,d | r,s,0 | r,s,1 | 0,0,0,0,1,0,0,1,0,0,0 | srl rd rs0 rs1 |
| slli | R_L | 1 | 1,0,1,1,1,0 | r,d | r,s,0 | 0,0,0,0,0,0,0,0,0,1 | c,ons,t | slli rd rs0 const |
| sub | R_R | 1 | 1,1,1,0,0,0 | r,d | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,0,1,0 | sub rd rs0 rs1 |
| sw | ST | 1 | 1,1,0,1,0,1 | r,d | r,s,0 | c,ons,t | | sw addr rs0 |
| xor | R_R | 1 | 1,1,1,0,0,0 | r,d | r,s,0 | r,s,1 | 0,0,0,0,0,0,0,0,1,0,1 | xor rd rs0 rs1 |
| xori | R_I | 1 | 1,0,1,0,1,1 | r,d | r,s,0 | c,ons,t | | xori rd rs0 const |

圖-9 指令格式

4.6 C 語言類別與指令行為描述

在圖-10 與-11 中，我們定義了 C 語言各類別的位元寬度，及各指令的行為描述。這些是用來產生 C compiler 用的，並不影響實際 RTL code 的產生。但是即便不用產生 C compiler, ASIPmeister 依舊要求要完成此階段設計才能往下一階段進行。

| C Definition | | | |
|--|-----------|---------------|------|
| File | | | Help |
| <input checked="" type="checkbox"/> Complete | | | |
| DataType | | Ckf Prototype | |
| Data Type | Alignment | Size | |
| char | 8 | 8 | |
| short | 16 | 16 | |
| int | 32 | 32 | |
| long | 32 | 32 | |
| float | 32 | 32 | |
| double | 64 | 64 | |
| pointer | 32 | 32 | |
| struct | 8 | none | |
| stack | 32 | none | |
| data | 8 | none | |

圖-10 C 類別寬度

| Behavior Description | | | |
|--|----------------|-------------------|------------------|
| File | | | Edit Search Help |
| <input checked="" type="checkbox"/> Complete | | | |
| Instruction | | Format | |
| add #1 | add rd rs0 rs1 | | |
| Operand | | | |
| Name | Usage | Addr Mode/Storage | Data Type |
| rd | register | GPR | SInt31to0 |
| rs0 | register | GPR | SInt31to0 |
| rs1 | register | GPR | SInt31to0 |
| OV | overflow-flag | OV | |
| CY | carry-flag | CY | |
| Behavior Description | | | |
| rd=rs0+rs1{CY,OV}; | | | |

圖-11 指令行為描述

4.7 指令微運算定義

在圖-12 中可以看到 add 指令的微指令，包括了在各個流水線的階段各個運算單元所必須進行的動作，你也可以定義一些 Variable。若 Variable 跨流水線的階段，則 ASIPmeister 會自動產生暫存器來傳遞到下一階段。為了簡化微運算的撰寫，ASIPmeister 提供了巨指令功能。如 add 指令中的微運算的 FETCH(), GPR2READ(rs0,rs1), ALUEXEC(add,source0,soucel), WRITEBACK(rd,result), WRITECYOV()都是巨指令，要展開巨指令可以按圖-12 右上方 Macro Expansion，可以觀看實際展開的微運算如圖-13。

| Name | Stage | Behavior Description |
|----------|----------|-------------------------------------|
| add #1 | VARIABLE | |
| addi #1 | | |
| and #1 | | |
| andi #1 | | |
| bf #1 | | FETCH() |
| bnf #1 | IF | |
| j #1 | | |
| jal #1 | | |
| jalr #1 | | GPR2READ(rs0,rs1) |
| jr #1 | ID | |
| lbs #1 | | |
| lbz #1 | | |
| lhs #1 | | ALUEXEC(add,source0,soucel) |
| lhz #1 | EXE | |
| lws #1 | | |
| lwz #1 | | |
| movhi #1 | MEM | |
| mul #1 | | |
| muli #1 | | |
| mulu #1 | | |
| nop #1 | | WRITEBACK(rd,result) WRITECYOV() |
| or #1 | WB | |
| ori #1 | | |
| rori #1 | | |
| sb #1 | | |
| sfeq #1 | | |
| sfges #1 | | |
| sfgeu #1 | | |

圖-12 指令微運算定義

| Micro Op. Description with Macro Expansion | | |
|--|----------|--|
| File Search | | |
| Common Instruction Exception Macro | | |
| Name | Stage | Behavior Description |
| add #1 | VARIABLE | wire [31:0] current_pc; |
| addi #1 | | wire [31:0] source0; |
| and #1 | | wire [31:0] source1; |
| andi #1 | | wire [31:0] result; |
| bf #1 | | wire [3:0] flag; |
| bnf #1 | | wire flag_cy; |
| j #1 | | wire flag_ov; |
| jal #1 | | wire [31:0] inst; |
| jalr #1 | | current_pc = PC.read(); |
| jr #1 | | inst = IMAU.read(current_pc); |
| lbs #1 | IF | null = IR.write(inst); |
| lbz #1 | | null = PC.inc(); |
| lhs #1 | ID | source0 = GPR.read0(rs0); |
| lhz #1 | | source1 = GPR.read1(rs1); |
| lws #1 | EXE | <result, flag> = ALU0.add(source0, source1); |
| lwz #1 | | |
| movhi #1 | MEM | |
| mul #1 | | |
| muli #1 | | |
| mulu #1 | | |
| nop #1 | | |
| or #1 | | |
| ori #1 | | |
| rori #1 | WB | null = GPR.write0(rd, result); |
| sb #1 | | |
| sfeq #1 | | flag_cy=flag[3]; |
| sfges #1 | | flag_ov=flag[0]; |
| sfgeu #1 | | null=CY.write(flag_cy); |
| sfgts #1 | | null=OV.write(flag_ov); |

圖-13 巨指令展開後的微運算

在微運算中有例外 (Exception) 的處理，目前我們使用的 ASIPmeister 版本只支援重置 (reset) 並未支援中斷等其他例外。在圖-14 中我們可以看到 reset 時，PC，GPR，IR，F，CY，OV 等暫存器執行 reset 的功能，而這些 reset 的功能可以在 ASIPmeister 系統資源宣告時察看 Function Set，

| Micro Op. Description | | |
|------------------------------------|-----------------|--|
| File Edit Search Help | | |
| <input type="checkbox"/> Complete | Macro Expansion | |
| Common Instruction Exception Macro | | |
| Name | Stage | Behavior Description |
| reset | VARIABLE | |
| | STAGE 1 | null = PC.reset(); null = GPR.reset(); null = IR.reset(); null = F.reset(); null = CY.reset(); null = OV.reset(); |

圖-14 例外處理

巨指令用來簡化微指令的撰寫，圖-15 是讀取指令的巨指令 Fetch。請注意於巨指令中的 stage 1 並不是指絕對的 stage 1 而是相對於呼叫 Fetch 的那個流水線階段，如果你在 EXE 的階段呼叫 Fetch，那麼 stage 1 就是 EXE 的階段而非 IF 的階段。

| | | |
|---|------------------------|--|
| <pre>FETCH() GPR2READ(arg1,arg2) SHIFT(ope,src1,src2) WRITEBACK(arg1,ar) JUMP() WRITELINKREG() MUL(ope,arg1,arg2) DIVIDE(ope,arg1,ar) ALUEXEC(ope,arg1, -----</pre> | <p>VARIABLE</p> | <pre>wire [31:0] current_pc;</pre> |
| | <p>STAGE 1</p> | <pre>wire [31:0] inst; current_pc = PC.read(); inst = IMAU.read(current_pc); null = IR.write(inst); null = PC.inc();</pre> |

圖-15 巨指令 Fetch

4.8 手動的修改

在完成了 ASIPmeister 的設計後產生的 RTL code，在合成邏輯前。我們有再加以修改，主要是因為 ASIPmeister 目前的版本並不支援 Data hazard 的解決。但是我們評估要去改 ORIK GCC source 困難度更高，便決定修改 ASIPmeister 產生的 VHDL 源碼(表-3)，因為 ASIPmeister 有支援 data memory 及 multi-cycle 指令時的等待週期(wait state)，所以在加入檢查 Data hazard 時若有 Hazard 就產生等待週期。

第二個修正是少數電路如 ASIPmeister 產生的 DMAU 資料地址單元，有些信號並不是與主時脈同步，對於 on-chip 的記憶體存取及 scan chain insert 較不方便，因此一律將之改為與主時脈同步的方式。

```

-- *** Start included statmes for data hazard control
r_rd_ID <= '0';
r_rs0_ID <= type_RR or type_RI or type_RL or type_SETF or type_ST or type_LD;
r_rs1_ID <= type_RR or type_JR or type_SETF or type_ST;
w_rd_ID <= type_RR or type_RI or type_I or type_RL or type_LD or
      (decoded_jalr or decoded_jal );

-- Between IF/ID
rd_ID <= DATAIN_IR_DATA_OUT(25 downto 21) when (decoded_jalr or decoded_jal ) = '0'
      else "01001"; -- write to R9 if jal,jalr
rs0_ID <= DATAIN_IR_DATA_OUT(20 downto 16);
rs1_ID <= DATAIN_IR_DATA_OUT(15 downto 11);

-- stall logic
valid_r_ID <= valid_stage_ID;
valid_w_EXE <= valid_stage_EXE;
valid_w_MEM <= valid_stage_MEM;
valid_w_WB <= valid_stage_WB;
match_ID_EXE_rd <= (valid_r_ID and r_rd_ID and valid_w_EXE and w_rd_EXE) when (rd_ID = rd_EXE)
      else '0';
match_ID_EXE_rs0 <= (valid_r_ID and r_rs0_ID and valid_w_EXE and w_rd_EXE) when (rs0_ID = rd_EXE)
      else '0';
match_ID_EXE_rs1 <= (valid_r_ID and r_rs1_ID and valid_w_EXE and w_rd_EXE) when (rs1_ID = rd_EXE)
      else '0';
match_ID_MEM_rd <= (valid_r_ID and r_rd_ID and valid_w_MEM and w_rd_MEM) when (rd_ID = rd_MEM)
      else '0';
match_ID_MEM_rs0 <= (valid_r_ID and r_rs0_ID and valid_w_MEM and w_rd_MEM) when (rs0_ID = rd_MEM)
      else '0';
match_ID_MEM_rs1 <= (valid_r_ID and r_rs1_ID and valid_w_MEM and w_rd_MEM) when (rs1_ID = rd_MEM)
      else '0';
match_ID_WB_rd <= (valid_r_ID and r_rd_ID and valid_w_WB and w_rd_WB) when (rd_ID = rd_WB)
      else '0';
match_ID_WB_rs0 <= (valid_r_ID and r_rs0_ID and valid_w_WB and w_rd_WB) when (rs0_ID = rd_WB)
      else '0';
match_ID_WB_rs1 <= (valid_r_ID and r_rs1_ID and valid_w_WB and w_rd_WB) when (rs1_ID = rd_WB)
      else '0';
match_ID_EXE <= match_ID_EXE_rd or match_ID_EXE_rs0 or match_ID_EXE_rs1;
match_ID_MEM <= match_ID_MEM_rd or match_ID_MEM_rs0 or match_ID_MEM_rs1;
match_ID_WB <= match_ID_WB_rd or match_ID_WB_rs0 or match_ID_WB_rs1;

my_stall_IF <= '0'; -- list here, but not used now
my_stall_ID <= match_ID_EXE or match_ID_MEM or match_ID_WB;
my_stall_EXE <= match_ID_MEM or match_ID_WB;
my_stall_MEM <= match_ID_WB;
-- Modified source of original source
stall_req_ID <= '0' or
      my_stall_ID;
stall_req_EXE <= multcyc_MUL0_start_stall_EXE
      or my_stall_EXE;
stall_req_MEM <= multcyc_DMAU_req_stall_MEM
      or my_stall_MEM;
stall_req_WB <= '0';

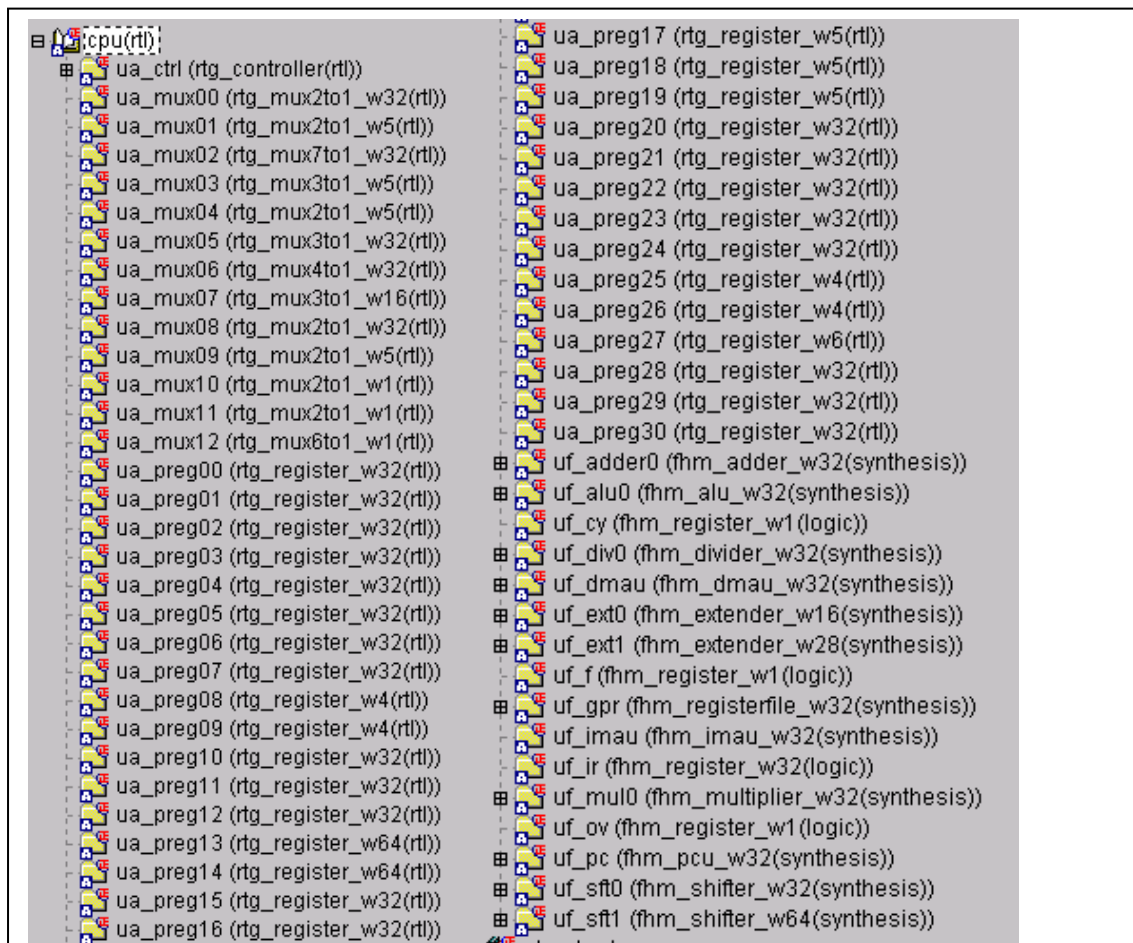
-- end of modified source

```

表-3 Data Hazard 處理

4.9 產生的 VHDL 原始碼

表-4 我們可以看到產生的 CPU 原始碼，其中最重要的是 ua_ctrl 是產生所有控制信號的模組，而 uf_開頭的就是 ASIPmeister 提供的系統單元。其他的 ua_mux, ua_preg 等就是依據為指令中的描述來產生各個系統單元間資料傳送的流向而由 ASIPmeister 產生。在圖-16 我們可以看到整個的方塊圖，也可以追蹤各系統單元資料傳送的過程。如果同一個系統單元的輸入有可能來自一個以上的系統單元，那麼 ASIPmeister 會自動的加上 MUX，而這些資料的傳送是跨流水線的階段，那麼 ASIPmeister 會自動的加上需要的流水線暫存器。



The image shows a screenshot of a VHDL project hierarchy in a software development environment. The hierarchy is organized into two columns. The left column shows the 'cpu rtl' package containing various control and data path modules. The right column shows the 'uf' package containing functional units. Each module is represented by a small icon and its full name in parentheses.

| Module Name | Module Name |
|----------------------------------|---|
| cpu rtl | ua_preg17 (rtg_register_w5 rtl) |
| ua_ctrl (rtg_controller rtl) | ua_preg18 (rtg_register_w5 rtl) |
| ua_mux00 (rtg_mux2to1_w32 rtl) | ua_preg19 (rtg_register_w5 rtl) |
| ua_mux01 (rtg_mux2to1_w5 rtl) | ua_preg20 (rtg_register_w32 rtl) |
| ua_mux02 (rtg_mux7to1_w32 rtl) | ua_preg21 (rtg_register_w32 rtl) |
| ua_mux03 (rtg_mux3to1_w5 rtl) | ua_preg22 (rtg_register_w32 rtl) |
| ua_mux04 (rtg_mux2to1_w5 rtl) | ua_preg23 (rtg_register_w32 rtl) |
| ua_mux05 (rtg_mux3to1_w32 rtl) | ua_preg24 (rtg_register_w32 rtl) |
| ua_mux06 (rtg_mux4to1_w32 rtl) | ua_preg25 (rtg_register_w4 rtl) |
| ua_mux07 (rtg_mux3to1_w16 rtl) | ua_preg26 (rtg_register_w4 rtl) |
| ua_mux08 (rtg_mux2to1_w32 rtl) | ua_preg27 (rtg_register_w6 rtl) |
| ua_mux09 (rtg_mux2to1_w5 rtl) | ua_preg28 (rtg_register_w32 rtl) |
| ua_mux10 (rtg_mux2to1_w1 rtl) | ua_preg29 (rtg_register_w32 rtl) |
| ua_mux11 (rtg_mux2to1_w1 rtl) | ua_preg30 (rtg_register_w32 rtl) |
| ua_mux12 (rtg_mux6to1_w1 rtl) | uf_adder0 (fhm_adder_w32 synthesis) |
| ua_preg00 (rtg_register_w32 rtl) | uf_alu0 (fhm_alu_w32 synthesis) |
| ua_preg01 (rtg_register_w32 rtl) | uf_cy (fhm_register_w1 logic) |
| ua_preg02 (rtg_register_w32 rtl) | uf_div0 (fhm_divider_w32 synthesis) |
| ua_preg03 (rtg_register_w32 rtl) | uf_dmau (fhm_dmau_w32 synthesis) |
| ua_preg04 (rtg_register_w32 rtl) | uf_ext0 (fhm_extender_w16 synthesis) |
| ua_preg05 (rtg_register_w32 rtl) | uf_ext1 (fhm_extender_w28 synthesis) |
| ua_preg06 (rtg_register_w32 rtl) | uf_f (fhm_register_w1 logic) |
| ua_preg07 (rtg_register_w32 rtl) | uf_gpr (fhm_registerfile_w32 synthesis) |
| ua_preg08 (rtg_register_w4 rtl) | uf_imau (fhm_imau_w32 synthesis) |
| ua_preg09 (rtg_register_w4 rtl) | uf_ir (fhm_register_w32 logic) |
| ua_preg10 (rtg_register_w32 rtl) | uf_mul0 (fhm_multiplier_w32 synthesis) |
| ua_preg11 (rtg_register_w32 rtl) | uf_ov (fhm_register_w1 logic) |
| ua_preg12 (rtg_register_w32 rtl) | uf_pc (fhm_pcu_w32 synthesis) |
| ua_preg13 (rtg_register_w64 rtl) | uf_sft0 (fhm_shifter_w32 synthesis) |
| ua_preg14 (rtg_register_w64 rtl) | uf_sft1 (fhm_shifter_w64 synthesis) |
| ua_preg15 (rtg_register_w32 rtl) | |
| ua_preg16 (rtg_register_w32 rtl) | |

表-4 CPU Top VHDL 模組

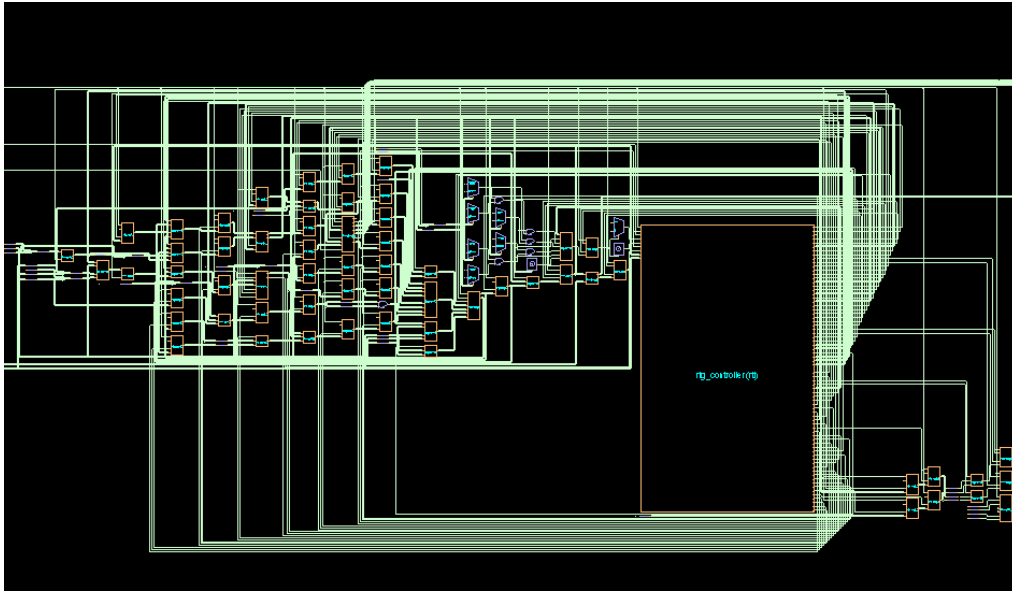


圖-16 CPU Top 方塊圖

4.10 驗證方式

我以附件一的測試軟體於軟體 OR1K simulator 中執行得到附件二的執行過程檔。再將測試程式的機器碼放入 VHDL test bench 中跑模擬 (圖-17)，並已觀察 PC 及 Register 變化 (表-5)，來分析 ASIPmeister 的 VHDL 是否符合 OR1K 軟體 simulator 的結果，若有不符則再回去 ASIPmeister 修改設計。一般而言都是修改微運算的部分，直到 VHDL 模擬結果與軟體 simulator 的結果一致以後，才依此 VHDL 原始碼進行晶片設計。

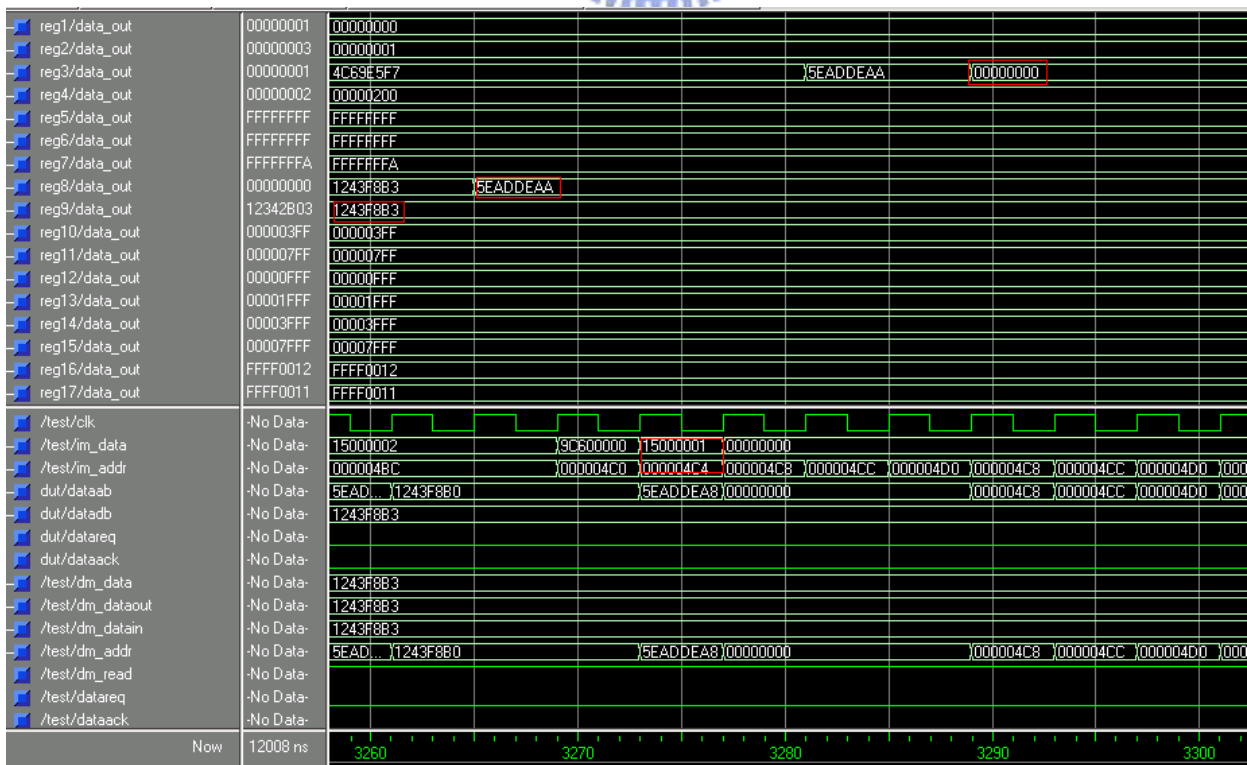


圖-17 Test Program Modelsim 結果

| | | |
|--|--------------|-----------------------------|
| | | 0000046c 1.bnf 0x3 |
| r8 =00000007 r8 =00000007 | | 00000470 1.addi r8,r8,0x1 |
| r8 =00000008 r8 =00000008 | | 00000474 1.addi r8,r8,0x1 |
| | r2 =00000001 | 00000478 1.sfeq r0,r2 |
| | | 0000047c 1.bnf 0x3 |
| r8 =00000009 r8 =00000009 | | 00000480 1.addi r8,r8,0x1 |
| r9 =00000000 | | 00000488 1.movhi r9,0 |
| r3 =00000494 r3 =00000494 | | 0000048c 1.ori r3,r3,0x494 |
| r8 =0000000a r8 =0000000a | | 00000490 1.addi r8,r8,0x1 |
| r3 =0000000a | r8 =0000000a | 00000494 1.or r3,r0,r8 |
| | | 00000498 1.nop 0x2 |
| r9 =1243f8a9 EA =00000000 | | 0000049c 1.lwz r9,0x0(r31) |
| r8 =1243f8b3 r9 =1243f8a9 r8 =1243f8b3 | | 000004a0 1.add r8,r9,r8 |
| EA =00000000 r8 =1243f8b3 | | 000004a4 1.sw 0x0(r31),r8 |
| r9 =1243f8b3 EA =00000000 | | 000004a8 1.lwz r9,0x0(r31) |
| r3 =4c690000 | | 000004ac 1.movhi r3,0x4c69 |
| r3 =4c69e5f7 r3 =4c69e5f7 | | 000004b0 1.ori r3,r3,0xe5f7 |
| r8 =5eaddeaa r8 =5eaddeaa r3 =4c69e5f7 | | 000004b4 1.add r8,r8,r3 |
| r3 =5eaddeaa | r8 =5eaddeaa | 000004b8 1.or r3,r0,r8 |
| | | 000004bc 1.nop 0x2 |
| r3 =00000000 | | 000004c0 1.addi r3,r0,0x0 |
| | | 000004c4 1.nop 0x1 |

表-5 Test Program Simulator 結果



五·晶片實作

5.1 設計流程

ASIP design =>
VHDL code generated =>
Write test program and run OR1K simulator to get log file =>
RTL simulation using ModelSim =>
Synthesis using Synopsys Design Compiler =>
Add scan chain using DFT compiler =>
RTL with scan cell simulation using ModelSim =>
Run TetraMax to get coverage log =>
Placement using SOC Encounter =>
Pre-layout Gate level Simulation using ModelSim =>
APR using SOC Encounter =>
Post-layout Gate level simulation using ModelSim=>
Replace GDS with Vertuso =>
DRC/LVS with Calibre

5.2 模擬結果

除了在邏輯合成前跑過模擬外，我們分別在合成後加了 Scan chain 及 pre-layout, post-layout 都跑了模擬（圖-18, 19, 20），除了存取外部 data memory 有部分 unknown 外，都沒有 setup/hold time 等違反 Timing 的錯誤產生。

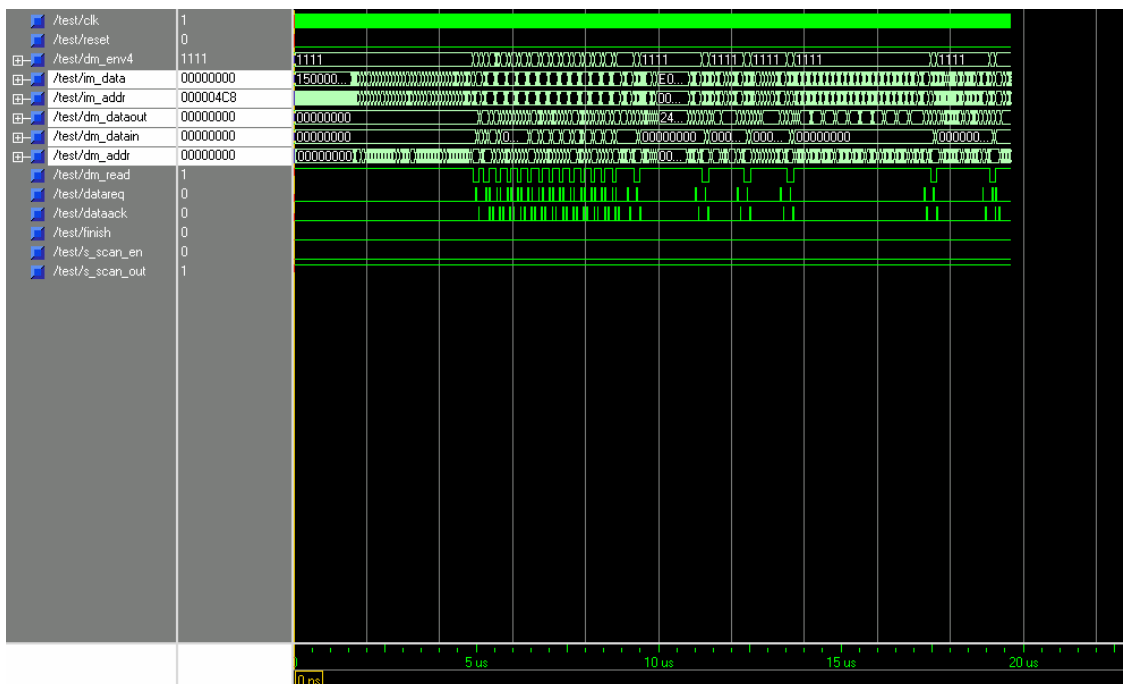


圖-18 RTL with scan cell simulation

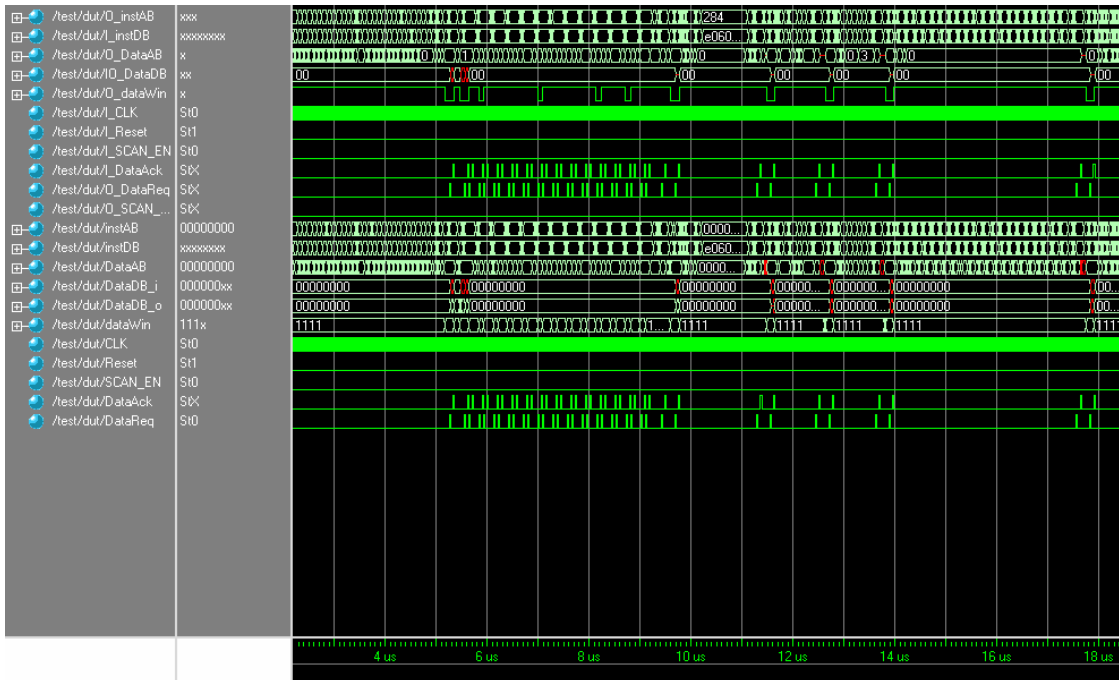


圖-19 Gate level simulation

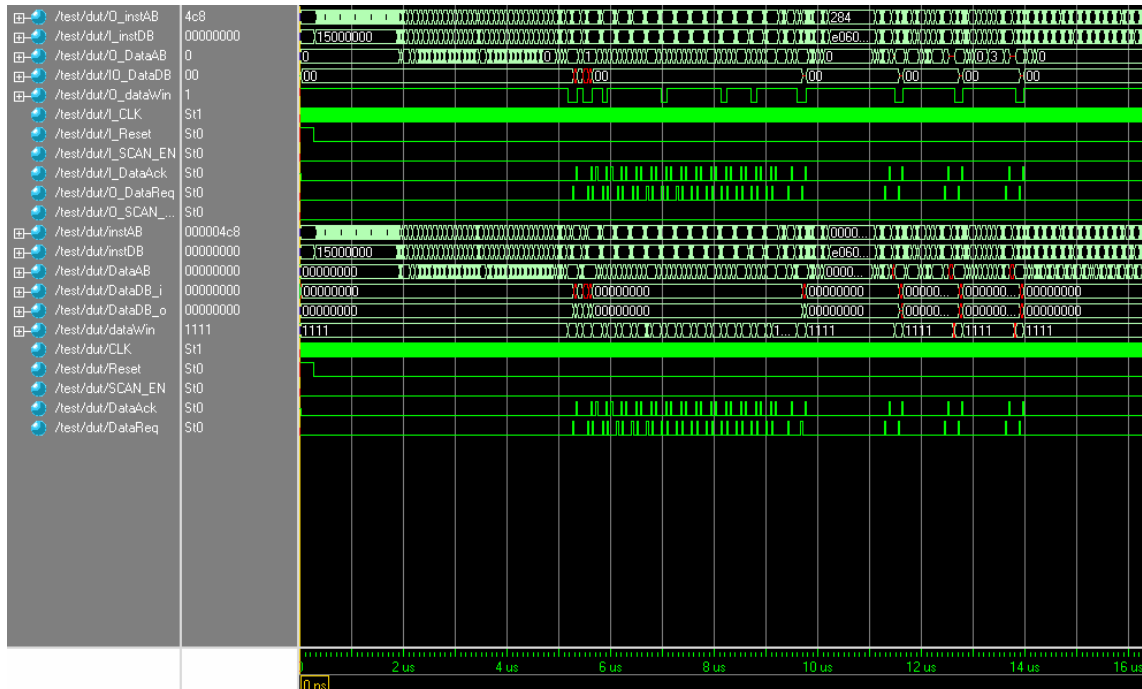


圖-20 Post layout simulation

5.3 晶片規格

Core Voltage : 1.8V

I/O Voltage : 3.3V

System Clock : 35Mhz (Asipmeister estimate: 32.7ns, Synthesis before scanchain added: 14.39ns, final: 28.45ns. Using 10ns SRAM as external instruction and data memory).

Chip Area : 1.4x1.4 mm Core Area : 1x1 mm

Power Consumption Simulation(Toggle rate probability 0.5) : 31mw

Utilization of core : 50.7%

Core Gate count : 32653 Gate(Asipmeister estimate:59.5K(typ), Synthesis before scanchain added:29.4K, final:32.5K) Area: 0.325 mm²

為了節省經費及本設計只有 32K gate, 所以我們並沒有將所有 Bus 的 pad 拉出, instruction address 只拉了 11 條, 而 data bus 指用 low byte, 地址只拉了兩條。但也足夠驗證所有除了 load/store word 及 half word 指令。因此將 die size 固定在 1.4x1.4mm, 64 I/O pad. 雖然 power for I/O 及 core 只有各一對, 但於 SOC Encounter Power 分析時將 Toggle probability 設成 0.5 時都只有 30mw 左右, 且 power strip 有三組, 不會有電流密度問題

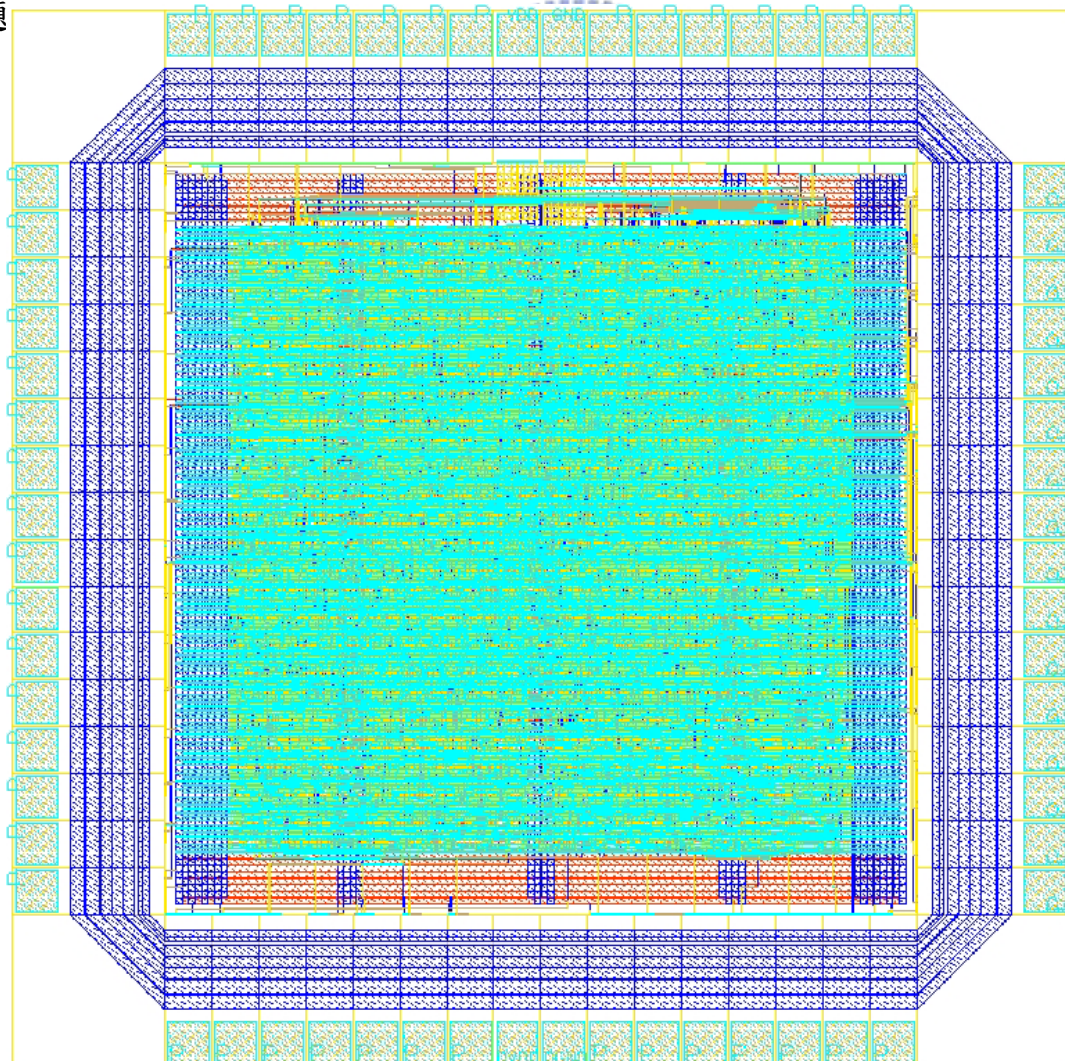


圖-21 Chip Layout

5.4 佈局結果錯誤說明

```

1. DRC : No error.

2. LVS :

          Layout      Source      Component Type
          -----      -
Ports:          62      64      *
*****
                                INCORRECT NETS
DISC#  LAYOUT NAME                                SOURCE NAME
*****
  1    Net 4                                ** no similar net **
      --- Devices on layout net 4 ---
      X22/X0(675.860,5.178)  PVSSR          ** no similar instance **
      DGND: 4                    ** no similar net **
-----
  2    Net 5                                ** no similar net **
      --- Devices on layout net 5 ---
      X8/X7(615.740,5.178)  PVDDR          ** no similar instance **
      DVDD: 5                    ** no similar net **
  3    ** no similar net **                    DVDD
      --- Devices on source net DVDD ---
  4    ** no similar net **                    DGND
*****
                                INCORRECT PORTS
DISC#  LAYOUT NAME                                SOURCE NAME
*****
  5    ** missing port **                    DVDD on net: DVDD
  6    ** missing port **                    DGND on net: DGND

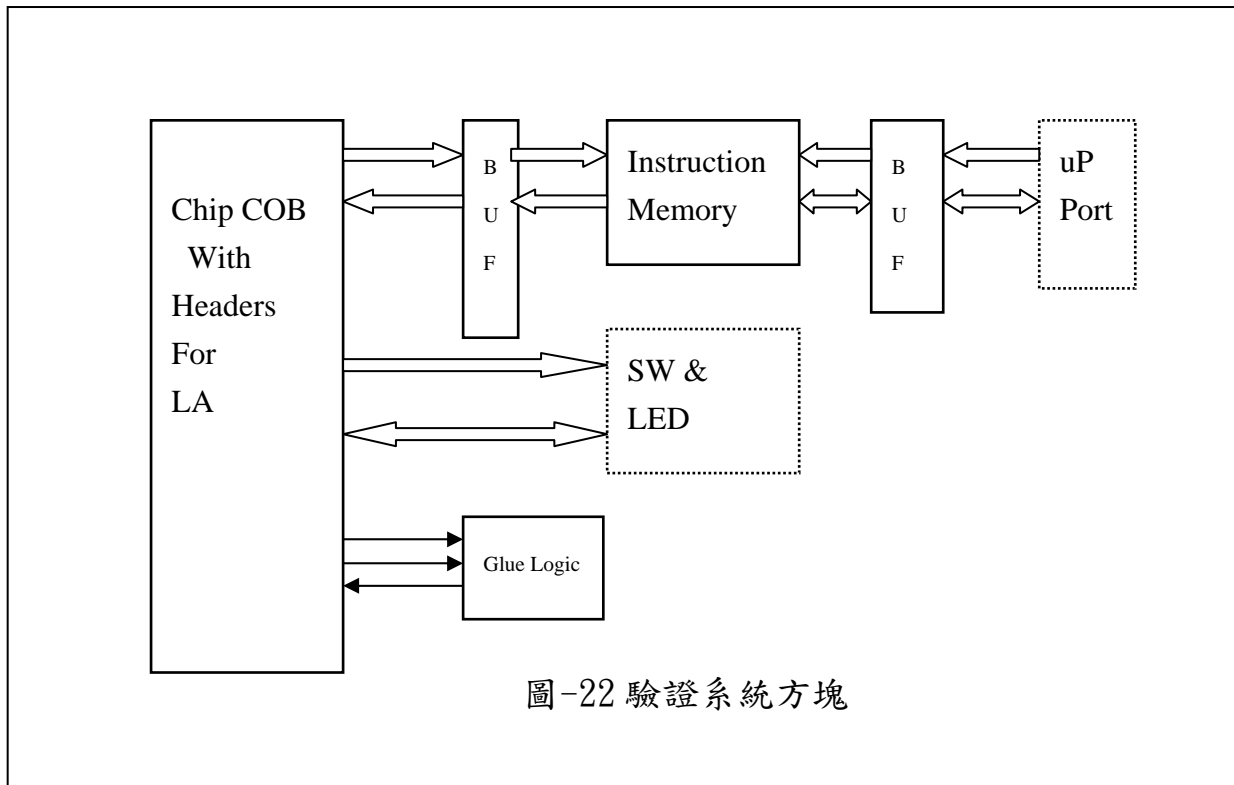
```

表-6 佈局結果錯誤

說明：

PAD power PAD PVDDR, PVSSR 的 DVDD 及 DGND 於 APR 時並未 route,但實際 I/O PAD 的 DVDD, DGND 會自動接在一起。

5.5 測試規劃



驗證時需設計一小 CHIP COB 版並將晶片 Bond 在上面，並設計上方的小測版用來測試驗證程式。Data Memory 經由 Buf 線路形成 dual port, 可以用來下載程式。SW 及 LED 用來當簡單的輸入及輸出。Glue logic 用來將同步的 clk, request, acknowledge 轉為外部 memory read, memory write 信號。

因本驗證系統須待 CIC 晶片完成後才能驗證，因此目前只是規劃，尚未完成。

六·結論

關於 ASIP 設計工具的最大功能，在於加快設計及修改的週期，以便在各種設計的可能中挑到一個最合乎要求的設計。以[10]中的論文提到，在一個學習了 PEAS-III(後來更名為 ASIPmeister)工具的學生，設計一個 32 bit R3000 的時間(表 6-1)需約 58 小時，修改成 24-bit 的 RISC processor 只需 13 小時。在本實作的設計及學習時間上，學習的時間並未記錄，在設計的時間上實際花的時間第一版的工作時間約 12 天，但後續與 OR1K software simulator 實際比較及修改微運算的過程約花了 12 天。因此實際設計花了約 $24 \times 8 = 196$ 小時，是[10]中 58 小時的三倍多。分析一下，[10]中的 modifying errors 只花了 2 小時，而我花了與寫第一版一樣長的時間，原因可能是我本身對 OR1K 不熟的關係及必須處理 Data hazard。

TABLE III
WORK LOAD FOR DESIGNING A RISC CONTROLLER

| works | first design (hour) | modification (hour) |
|---|------------------------|------------------------|
| selecting resources | 3 | 1 |
| determining instruction set architecture | 12 | 8 |
| writing micro operation description | 40 | 2 |
| modifying errors | 2 | 2 |
| total | 58 | 13 |

表-7 R3000 設計時間

至於合成出來的 gate count，表-8 中可以知道[10]的 R3000 設計合成 30.8K。而 OR1K 也與 R3000 類似是 32-bit RISC，我的合成結果是 32.6K。而 OR1K 的功能與 R3000 相仿，因此兩者的 gate count 結果也差異不大。實作的結果證實[10]所提的設計時間及 gate count 與本設計比較，除設計時間[10]中較短外，本設計與[10]的數據相符。也因此可以證實 ASIP 工具可以加速設計週期，以本設計而言共花了 24 天工作天即可以產生完整的 RTL-code，相較於完全手動撰寫，在時間上的確有很大的幫助。

至於 C compiler 的產生，因為我們沒有 COSY compiler generator 可以測試，雖然 ASIPmeister 有產生相關的檔案，但無法驗證 C compiler 的正確與否，也無法進行針對特定 application 來進行架構調整的工作。因此對於要進一步使用此 ASIP 工具設計 ASIP 時，必須向 Target Compiler Technology 申請學術版的 COSY 或購買。沒有 COSY，設計出來的 ASIP 就沒有 C compiler 可用，就不能去調整 Processor 架構。我曾評估自行修改 GNU C compiler 中的 Machine Description file，但是必須花不少時間。

最後決定只作 general purpose RISC, 以免需自行設計 C compiler。ASIPmeister 是用來設計 ASIP, 而我的實作受限於 C compiler 無法產生, 因此只作 general purpose RISC, 是日後可以改進的地方。

TABLE I
RESULTS OF LOGIC SYNTHESIS FOR R3000

| component | # | Cell Area (gates) | Total Area (gates) | Frequency (MHz) |
|-----------------------------|----|----------------------|-----------------------|--------------------|
| user specified resources | 12 | 21,833.5 | 33,232.2 | 70.97 |
| registers | 23 | 6,305.8 | 7,772.6 | 934.58 |
| selectors | 10 | 1,163.5 | 1,822.6 | 437.80 |
| controller | 1 | 1,662.3 | 2,474.8 | 110.01 |
| sum of the above | 45 | 30,965.0 | 45,302.1 | 70.97 |
| processor | 1 | 30,883.0 | 49,643.2 | 50.28 |

using Design Compiler (0.5 μ m CMOS library)

表-8 R3000 合成結果

在完整的 OR1K BAS32 I 指令能以 ASIPmeister 完成後, 要進一步的刪掉不必要的指令, 或者要增加特殊的指令都可以很快速的修改。以[10]中所提的例子在完成 32-bit 的 R3000 設計後, 表-7 中 modification 顯示他們改成 24-bit 的指令只花了 13 小時。同樣的在本設計完成後要修改也是十分容易, 可以因應不同的需求來規劃修改。本實作證實了 RISC processor 以 ASIP 的方式, 在縮短設計時間上是十分具有優勢的, 而產生的 RTL-code, 在 gate count 及速度上也有可以接受的結果。

参 考 文 献

- [1]. <http://www.coware.com/products/processor designer.php> Coware Company.
- [2]. <http://www.retarget.com/brfchschk.html> Target Compiler Tech.
- [3]. <http://www.asip-solutions.com/> Asip-solutions Company
- [4]. <http://www.ics.uci.edu/~express/> Expression home page.
- [5]. Souvik Basu, Rajat Moona, "High Level Synthesis from Sim-nML Processor Models," *vlsid*, p. 255, 16th International Conference on VLSI Design, 2003.
- [6]. <http://www.opencores.org/projects.cgi/web/orlk/overview> Orlk home page.
- [7]. ASIP Meister User's Manual
- [8]. ASIP Meister Tutorial.
- [9]. OpenRISC 1000 Architecture Manual, 26/June/2004.
- [10]. Kitajima, A. ; Itoh, M. ; Sato, J. ; Shiomi, A. ; Takeuchi, Y. ; Imai, M. " Effectiveness of the ASIP design system PEAS-III in design of pipelined processors " , Page(s):649 - 654 , Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific 30 Jan.-2 Feb. 2001

附件一：test program

```
/* Basic instruction set test */
#include "../support/spr_defs.h"
#define MEM_RAM 0x00000000
.global _main
.global _buserr_except
.global _dpf_except
.global _ipf_except
.global _lpint_except
.global _align_except
.global _illegal_except
.global _hpint_except
.global _dtlbmiss_except
.global _itlbmiss_except
.global _range_except
.global _syscall_except
.global _res1_except
.global _trap_except
.global _res2_except

.section .text
.org 0x0
l.nop
l.j _main
l.nop
_buserr_except:
_dpf_except:
_ipf_except:
_lpint_except:
_align_except:
_illegal_except:
_hpint_except:
_dtlbmiss_except:
_itlbmiss_except:
_range_except:
_syscall_except:
_res1_except:
_trap_except:
_res2_except:
/* target=uclinux-or32 start at 0x100 */
_main:
.org 0x100
l.nop
l.j _regs
l.nop

_reggs:
l.addi r1,r0,0x1
l.addi r2,r1,0x2
l.addi r3,r2,0x4
l.addi r4,r3,0x8
l.addi r5,r4,0x10
l.addi r6,r5,0x20
l.addi r7,r6,0x40
l.addi r8,r7,0x80
```



```

l.addi    r9,r8,0x100
l.addi    r10,r9,0x200
l.addi    r11,r10,0x400
l.addi    r12,r11,0x800
l.addi    r13,r12,0x1000
l.addi    r14,r13,0x2000
l.addi    r15,r14,0x4000
l.addi    r16,r15,0x8000

```

```

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

```

```

l.or  r3,r0,r16
l.nop NOP_REPORT /* Should be */

```

```

l.movhi r31, hi(MEM_RAM)
l.ori  r31,r31, lo(MEM_RAM)
l.sw  0(r31),r16

```

```

_mem:  l.movhi  r3,0x1234
       l.ori   r3,r3,0x5678

```

```

l.sw  4(r31),r3

```

```

l.lbz r4,4(r31)
l.add r8,r8,r4
l.sb  11(r31),r4
l.lbz r4,5(r31)
l.add r8,r8,r4
l.sb  10(r31),r4
l.lbz r4,6(r31)
l.add r8,r8,r4
l.sb  9(r31),r4
l.lbz r4,7(r31)
l.add r8,r8,r4
l.sb  8(r31),r4

```

```

l.lbs  r4,8(r31)
l.add  r8,r8,r4
l.sb   7(r31),r4
l.lbs  r4,9(r31)
l.add  r8,r8,r4
l.sb   6(r31),r4
l.lbs  r4,10(r31)
l.add  r8,r8,r4

```



```

l.sb    5(r31), r4
l.lbs   r4, 11(r31)
l.add   r8, r8, r4
l.sb    4(r31), r4

```

```

l.lhz   r4, 4(r31)
l.add   r8, r8, r4
l.sh    10(r31), r4
l.lhz   r4, 6(r31)
l.add   r8, r8, r4
l.sh    8(r31), r4

```

```

l.lhs   r4, 8(r31)
l.add   r8, r8, r4
l.sh    6(r31), r4
l.lhs   r4, 10(r31)
l.add   r8, r8, r4
l.sh    4(r31), r4

```

```

l.lwz   r4, 4(r31)
l.add   r8, r8, r4

```

```

l.or    r3, r0, r8
l.nop   NOP_REPORT /* Should be */

```

```

l.lwz   r9, 0(r31)
l.add   r8, r9, r8
l.sw    0(r31), r8

```

_arith:

```

l.addi  r3, r0, 1
l.addi  r4, r0, 2
l.addi  r5, r0, -1
l.addi  r6, r0, -1
l.addi  r8, r0, 0

```

```

l.sub   r7, r5, r3
l.sub   r8, r3, r5
l.add   r8, r8, r7

```

```

# l.div  r7, r7, r4
l.add   r9, r3, r4
l.mul  r7, r9, r7

```

```

# l.divu r7, r7, r4
l.add   r8, r8, r7

```

```

l.or    r3, r0, r8
l.nop   NOP_REPORT /* Should be */

```

```

l.lwz   r9, 0(r31)
l.add   r8, r9, r8
l.sw    0(r31), r8

```

_log:

```

l.addi  r3, r0, 1
l.addi  r4, r0, 2
l.addi  r5, r0, -1
l.addi  r6, r0, -1
l.addi  r8, r0, 0

```




```

l.andi    r8,r8,1
l.and r8,r8,r3

l.xori    r8,r5,0xa5a5
l.xor r8,r8,r5

l.ori r8,r8,2
l.or  r8,r8,r4

l.or  r3,r0,r8
l.nop NOP_REPORT /* Should be */

l.lwz r9,0(r31)
l.add r8,r9,r8
l.sw 0(r31),r8

```

_shift:

```

l.addi r3,r0,1
l.addi r4,r0,2
l.addi r5,r0,-1
l.addi r6,r0,-1
l.addi r8,r0,0

```

```

l.slli r8,r5,6
l.sll r8,r8,r4

```

```

l.srli r8,r8,6
l.srl r8,r8,r4

```

```

l.srai r8,r8,2
l.sra r8,r8,r4

```

```

l.or  r3,r0,r8
l.nop NOP_REPORT /* Should be */

```

```

l.lwz r9,0(r31)
l.add r8,r9,r8
l.sw 0(r31),r8

```

_flag:

```

l.addi r3,r0,1
l.addi r4,r0,-2
l.addi r8,r0,0

```

```

l.sfeq r3,r3
l.andi r4,r5,0x200
l.add r8,r8,r4

```

```

l.sfeq r3,r4
l.andi r4,r5,0x200
l.add  r8,r8,r4

```

```

l.sfne r3,r3
l.andi r4,r5,0x200
l.add  r8,r8,r4

```

```

l.sfne r3,r4
l.andi r4,r5,0x200

```



l.add r8,r8,r4

l.sfgtu r3,r3
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfgtu r3,r4
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfgeu r3,r3
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfgeu r3,r4
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfltu r3,r3
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfltu r3,r4
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfleu r3,r3
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfleu r3,r4
l.andi r4,r5,0x200
l.add r8,r8,r4
l.sfgts r3,r3
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfgts r3,r4
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfges r3,r3
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfges r3,r4
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sflts r3,r3
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sflts r3,r4
l.andi r4,r5,0x200
l.add r8,r8,r4

l.sfls r3,r3
l.andi r4,r5,0x200



```

        l.add    r8,r8,r4

        l.sfls  r3,r4
        l.andi  r4,r5,0x200
        l.add    r8,r8,r4

        l.or   r3,r0,r8
l.nop NOP_REPORT /* Should be */

        l.lwz r9,0(r31)
        l.add r8,r9,r8
        l.sw  0(r31),r8

        l.addi r1,r0,0 /* R1<= 0 */
        l.addi r2,r0,1 /* R2<=1 */

_jump:
        l.addi  r8,r0,0

        l.j   _T1
        l.addi  r8,r8,1

_T2:    l.jr   r9
        l.addi  r8,r8,1

_T1:    l.jal  _T2
        l.addi  r8,r8,1

        l.sfeq  r0,r1
        l.bf   _T3
        l.addi  r8,r8,1

_T3:    l.sfeq  r0,r2
        l.bf   _T4
        l.addi  r8,r8,1

        l.addi  r8,r8,1

_T4:    l.sfeq  r0,r1
        l.bnf  _T5
        l.addi  r8,r8,1

        l.addi  r8,r8,1

_T5:    l.sfeq  r0,r2
        l.bnf  _T6
        l.addi  r8,r8,1

        l.addi  r8,r8,1

_T6:    l.movhi r3,hi(_T7)
        l.ori  r3,r3,lo(_T7)
        /* l.rfe should not have a delay slot */

        l.addi  r8,r8,1

_T7:    l.or   r3,r0,r8
l.nop NOP_REPORT /* Should be */

```



```
l.lwz r9,0(r31)
l.add r8,r9,r8
l.sw 0(r31),r8

l.lwz r9,0(r31)
l.movhi r3,0x4c69
l.ori r3,r3,0xe5f7
l.add r8,r8,r3

l.or r3,r0,r8
l.nop NOP_REPORT /* Should be */

l.addi r3,r0,0
l.nop NOP_EXIT

/* stack */
.org 0x800

.section .stack
.space 0x100
_tmp_stack:

/* error */
.org 0x900

_error:
```



附 件 二 : software simulation log

```

00000100 l.nop 0
00000104 l.j 0x2
00000108 l.nop 0
r1 =00000001 0000010c l.addi r1,r0,0x1
r2 =00000003 r1 =00000001 00000110 l.addi r2,r1,0x2
r3 =00000007 r2 =00000003 00000114 l.addi r3,r2,0x4
r4 =0000000f r3 =00000007 00000118 l.addi r4,r3,0x8
r5 =0000001f r4 =0000000f 0000011c l.addi r5,r4,0x10
r6 =0000003f r5 =0000001f 00000120 l.addi r6,r5,0x20
r7 =0000007f r6 =0000003f 00000124 l.addi r7,r6,0x40
r8 =000000ff r7 =0000007f 00000128 l.addi r8,r7,0x80
r9 =000001ff r8 =000000ff 0000012c l.addi r9,r8,0x100
r10=000003ff r9 =000001ff 00000130 l.addi r10,r9,0x200
r11=000007ff r10=000003ff 00000134 l.addi r11,r10,0x400
r12=00000fff r11=000007ff 00000138 l.addi r12,r11,0x800
r13=00001fff r12=00000fff 0000013c l.addi r13,r12,0x1000
r14=00003fff r13=00001fff 00000140 l.addi r14,r13,0x2000
r15=00007fff r14=00003fff 00000144 l.addi r15,r14,0x4000
r16=ffffff r15=00007fff 00000148 l.addi r16,r15,-32768
r31=ffffff r1 =00000001 0000014c l.sub r31,r0,r1
r30=ffffffc r31=ffffff r2 =00000003 00000150 l.sub r30,r31,r2
r29=ffffff5 r30=ffffffc r3 =00000007 00000154 l.sub r29,r30,r3
r28=ffffffe6 r29=ffffff5 r4 =0000000f 00000158 l.sub r28,r29,r4
r27=ffffffc7 r28=ffffffe6 r5 =0000001f 0000015c l.sub r27,r28,r5
r26=ffffff88 r27=ffffffc7 r6 =0000003f 00000160 l.sub r26,r27,r6
r25=ffffff09 r26=ffffff88 r7 =0000007f 00000164 l.sub r25,r26,r7
r24=fffffe0a r25=ffffff09 r8 =000000ff 00000168 l.sub r24,r25,r8
r23=fffffc0b r24=fffffe0a r9 =000001ff 0000016c l.sub r23,r24,r9
r22=fffff80c r23=fffffc0b r10=000003ff 00000170 l.sub r22,r23,r10
r21=fffff00d r22=fffff80c r11=000007ff 00000174 l.sub r21,r22,r11
r20=ffffe00e r21=fffff00d r12=00000fff 00000178 l.sub r20,r21,r12
r19=ffffc00f r20=ffffe00e r13=00001fff 0000017c l.sub r19,r20,r13
r18=ffff8010 r19=ffffc00f r14=00003fff 00000180 l.sub r18,r19,r14
r17=ffff0011 r18=ffff8010 r15=00007fff 00000184 l.sub r17,r18,r15
r16=ffff0012 r17=ffff0011 r16=ffff0012 00000188 l.sub r16,r17,r16
r3 =ffff0012 r16=ffff0012 0000018c l.or r3,r0,r16
00000190 l.nop 0x2
r31=00000000 00000194 l.movhi r31,0
r31=00000000 r31=00000000 00000198 l.ori r31,r31,0
EA =00000000 r16=ffff0012 0000019c l.sw 0x0(r31),r16
r3 =12340000 000001a0 l.movhi r3,0x1234
r3 =12345678 r3 =12345678 000001a4 l.ori r3,r3,0x5678
EA =00000004 r3 =12345678 000001a8 l.sw 0x4(r31),r3
r4 =00000012 EA =00000004 000001ac l.lbz r4,0x4(r31)
r8 =00000111 r8 =00000111 r4 =00000012 000001b0 l.add r8,r8,r4
EA =0000000b r4 =00000012 000001b4 l.sb 0xb(r31),r4
r4 =00000034 EA =00000005 000001b8 l.lbz r4,0x5(r31)
r8 =00000145 r8 =00000145 r4 =00000034 000001bc l.add r8,r8,r4
EA =0000000a r4 =00000034 000001c0 l.sb 0xa(r31),r4
r4 =00000056 EA =00000006 000001c4 l.lbz r4,0x6(r31)
r8 =0000019b r8 =0000019b r4 =00000056 000001c8 l.add r8,r8,r4
EA =00000009 r4 =00000056 000001cc l.sb 0x9(r31),r4

```

```

r4 =00000078 EA =00000007          000001d0 l.lbz r4,0x7(r31)
r8 =00000213 r8 =00000213 r4 =00000078 000001d4 l.add r8,r8,r4
EA =00000008 r4 =00000078          000001d8 l.sb 0x8(r31),r4
r4 =00000078 EA =00000008          000001dc l.lbs r4,0x8(r31)
r8 =0000028b r8 =0000028b r4 =00000078 000001e0 l.add r8,r8,r4
EA =00000007 r4 =00000078          000001e4 l.sb 0x7(r31),r4
r4 =00000056 EA =00000009          000001e8 l.lbs r4,0x9(r31)
r8 =000002e1 r8 =000002e1 r4 =00000056 000001ec l.add r8,r8,r4
EA =00000006 r4 =00000056          000001f0 l.sb 0x6(r31),r4
r4 =00000034 EA =0000000a          000001f4 l.lbs r4,0xa(r31)
r8 =00000315 r8 =00000315 r4 =00000034 000001f8 l.add r8,r8,r4
EA =00000005 r4 =00000034          000001fc l.sb 0x5(r31),r4
r4 =00000012 EA =0000000b          00000200 l.lbs r4,0xb(r31)
r8 =00000327 r8 =00000327 r4 =00000012 00000204 l.add r8,r8,r4
EA =00000004 r4 =00000012          00000208 l.sb 0x4(r31),r4
r4 =00001234 EA =00000004          0000020c l.lhz r4,0x4(r31)
r8 =0000155b r8 =0000155b r4 =00001234 00000210 l.add r8,r8,r4
EA =0000000a r4 =00001234          00000214 l.sh 0xa(r31),r4
r4 =00005678 EA =00000006          00000218 l.lhz r4,0x6(r31)
r8 =00006bd3 r8 =00006bd3 r4 =00005678 0000021c l.add r8,r8,r4
EA =00000008 r4 =00005678          00000220 l.sh 0x8(r31),r4
r4 =00005678 EA =00000008          00000224 l.lhs r4,0x8(r31)
r8 =0000c24b r8 =0000c24b r4 =00005678 00000228 l.add r8,r8,r4
EA =00000006 r4 =00005678          0000022c l.sh 0x6(r31),r4
r4 =00001234 EA =0000000a          00000230 l.lhs r4,0xa(r31)
r8 =0000d47f r8 =0000d47f r4 =00001234 00000234 l.add r8,r8,r4
EA =00000004 r4 =00001234          00000238 l.sh 0x4(r31),r4
r4 =12345678 EA =00000004          0000023c l.lwz r4,0x4(r31)
r8 =12352af7 r8 =12352af7 r4 =12345678 00000240 l.add r8,r8,r4
r3 =12352af7 r8 =12352af7          00000244 l.or r3,r0,r8
00000248 l.nop 0x2
r9 =ffff0012 EA =00000000          0000024c l.lwz r9,0x0(r31)
r8 =12342b09 r9 =ffff0012 r8 =12342b09 00000250 l.add r8,r9,r8
EA =00000000 r8 =12342b09          00000254 l.sw 0x0(r31),r8
r3 =00000001          00000258 l.addi r3,r0,0x1
r4 =00000002          0000025c l.addi r4,r0,0x2
r5 =ffffffff          00000260 l.addi r5,r0,-1
r6 =ffffffff          00000264 l.addi r6,r0,-1
r8 =00000000          00000268 l.addi r8,r0,0x0
r7 =fffffffe r5 =ffffffff r3 =00000001 0000026c l.sub r7,r5,r3
r8 =00000002 r3 =00000001 r5 =ffffffff 00000270 l.sub r8,r3,r5
r8 =00000000 r8 =00000000 r7 =fffffffe 00000274 l.add r8,r8,r7
r9 =00000003 r3 =00000001 r4 =00000002 00000278 l.add r9,r3,r4
r7 =fffffffa r9 =00000003 r7 =fffffffa 0000027c l.mul r7,r9,r7
r8 =fffffffa r8 =fffffffa r7 =fffffffa 00000280 l.add r8,r8,r7
r3 =fffffffa r8 =fffffffa          00000284 l.or r3,r0,r8
00000288 l.nop 0x2
----- 100 instruction -----
r9 =12342b09 EA =00000000          0000028c l.lwz r9,0x0(r31)
r8 =12342b03 r9 =12342b09 r8 =12342b03 00000290 l.add r8,r9,r8
EA =00000000 r8 =12342b03          00000294 l.sw 0x0(r31),r8
r3 =00000001          00000298 l.addi r3,r0,0x1
r4 =00000002          0000029c l.addi r4,r0,0x2
r5 =ffffffff          000002a0 l.addi r5,r0,-1

```

```

r6 =ffffffff          000002a4 l.addi r6,r0,-1
r8 =00000000          000002a8 l.addi r8,r0,0x0
r8 =00000000 r8 =00000000 000002ac l.andi r8,r8,0x1
r8 =00000000 r8 =00000000 r3 =00000001 000002b0 l.and r8,r8,r3
r8 =00005a5a r5 =ffffffff 000002b4 l.xori r8,r5,-23131
r8 =ffffa5a5 r8 =ffffa5a5 r5 =ffffffff 000002b8 l.xor r8,r8,r5
r8 =ffffa5a7 r8 =ffffa5a7          000002bc l.ori r8,r8,0x2
r8 =ffffa5a7 r8 =ffffa5a7 r4 =00000002 000002c0 l.or r8,r8,r4
r3 =ffffa5a7          r8 =ffffa5a7 000002c4 l.or r3,r0,r8
                                000002c8 l.nop 0x2
r9 =12342b03 EA =00000000 000002cc l.lwz r9,0x0(r31)
r8 =1233d0aa r9 =12342b03 r8 =1233d0aa 000002d0 l.add r8,r9,r8
EA =00000000 r8 =1233d0aa          000002d4 l.sw 0x0(r31),r8
r3 =00000001          000002d8 l.addi r3,r0,0x1
r4 =00000002          000002dc l.addi r4,r0,0x2
r5 =ffffffff          000002e0 l.addi r5,r0,-1
r6 =ffffffff          000002e4 l.addi r6,r0,-1
r8 =00000000          000002e8 l.addi r8,r0,0x0
r8 =ffffffc0 r5 =ffffffff 000002ec l.slli r8,r5,0x6
r8 =ffffff00 r8 =ffffff00 r4 =00000002 000002f0 l.sll r8,r8,r4
r8 =03ffffffc r8 =03ffffffc          000002f4 l.srli r8,r8,0x6
r8 =00ffffff r8 =00ffffff r4 =00000002 000002f8 l.srl r8,r8,r4
r8 =003fffff r8 =003fffff          000002fc l.srai r8,r8,0x2
r8 =000fffff r8 =000fffff r4 =00000002 00000300 l.sra r8,r8,r4
r3 =000fffff          r8 =000fffff 00000304 l.or r3,r0,r8
                                00000308 l.nop 0x2
r9 =1233d0aa EA =00000000 0000030c l.lwz r9,0x0(r31)
r8 =1243d0a9 r9 =1233d0aa r8 =1243d0a9 00000310 l.add r8,r9,r8
EA =00000000 r8 =1243d0a9          00000314 l.sw 0x0(r31),r8
r3 =00000001          00000318 l.addi r3,r0,0x1
r4 =ffffffff          0000031c l.addi r4,r0,-2
r8 =00000000          00000320 l.addi r8,r0,0x0
r3 =00000001 r3 =00000001          00000324 l.sfeq r3,r3
r4 =00000200 r5 =ffffffff 00000328 l.andi r4,r5,0x200
r8 =00000200 r8 =00000200 r4 =00000200 0000032c l.add r8,r8,r4
r3 =00000001 r4 =00000200          00000330 l.sfeq r3,r4
r4 =00000200 r5 =ffffffff 00000334 l.andi r4,r5,0x200
r8 =00000400 r8 =00000400 r4 =00000200 00000338 l.add r8,r8,r4
r3 =00000001 r3 =00000001          0000033c l.sfne r3,r3
r4 =00000200 r5 =ffffffff 00000340 l.andi r4,r5,0x200
r8 =00000600 r8 =00000600 r4 =00000200 00000344 l.add r8,r8,r4
r3 =00000001 r4 =00000200          00000348 l.sfne r3,r4
r4 =00000200 r5 =ffffffff 0000034c l.andi r4,r5,0x200
r8 =00000800 r8 =00000800 r4 =00000200 00000350 l.add r8,r8,r4
r3 =00000001 r3 =00000001          00000354 l.sfgtu r3,r3
r4 =00000200 r5 =ffffffff 00000358 l.andi r4,r5,0x200
r8 =0000a00 r8 =0000a00 r4 =00000200 0000035c l.add r8,r8,r4
r3 =00000001 r4 =00000200          00000360 l.sfgtu r3,r4
r4 =00000200 r5 =ffffffff 00000364 l.andi r4,r5,0x200
r8 =00000c00 r8 =00000c00 r4 =00000200 00000368 l.add r8,r8,r4
r3 =00000001 r3 =00000001          0000036c l.sfgeu r3,r3
r4 =00000200 r5 =ffffffff 00000370 l.andi r4,r5,0x200
r8 =00000e00 r8 =00000e00 r4 =00000200 00000374 l.add r8,r8,r4
r3 =00000001 r4 =00000200          00000378 l.sfgeu r3,r4

```

```

r4 =00000200 r5 =ffffffff          0000037c l.andi r4,r5,0x200
r8 =00001000 r8 =00001000 r4 =00000200 00000380 l.add r8,r8,r4
r3 =00000001 r3 =00000001          00000384 l.sfltu r3,r3
r4 =00000200 r5 =ffffffff          00000388 l.andi r4,r5,0x200
r8 =00001200 r8 =00001200 r4 =00000200 0000038c l.add r8,r8,r4
r3 =00000001 r4 =00000200          00000390 l.sfltu r3,r4
r4 =00000200 r5 =ffffffff          00000394 l.andi r4,r5,0x200
r8 =00001400 r8 =00001400 r4 =00000200 00000398 l.add r8,r8,r4
r3 =00000001 r3 =00000001          0000039c l.sfleu r3,r3
r4 =00000200 r5 =ffffffff          000003a0 l.andi r4,r5,0x200
r8 =00001600 r8 =00001600 r4 =00000200 000003a4 l.add r8,r8,r4
r3 =00000001 r4 =00000200          000003a8 l.sfleu r3,r4
r4 =00000200 r5 =ffffffff          000003ac l.andi r4,r5,0x200
r8 =00001800 r8 =00001800 r4 =00000200 000003b0 l.add r8,r8,r4
r3 =00000001 r3 =00000001          000003b4 l.sfgts r3,r3
r4 =00000200 r5 =ffffffff          000003b8 l.andi r4,r5,0x200
r8 =00001a00 r8 =00001a00 r4 =00000200 000003bc l.add r8,r8,r4
r3 =00000001 r4 =00000200          000003c0 l.sfgts r3,r4
r4 =00000200 r5 =ffffffff          000003c4 l.andi r4,r5,0x200
r8 =00001c00 r8 =00001c00 r4 =00000200 000003c8 l.add r8,r8,r4
r3 =00000001 r3 =00000001          000003cc l.sfges r3,r3
r4 =00000200 r5 =ffffffff          000003d0 l.andi r4,r5,0x200
r8 =00001e00 r8 =00001e00 r4 =00000200 000003d4 l.add r8,r8,r4
r3 =00000001 r4 =00000200          000003d8 l.sfges r3,r4
r4 =00000200 r5 =ffffffff          000003dc l.andi r4,r5,0x200
r8 =00002000 r8 =00002000 r4 =00000200 000003e0 l.add r8,r8,r4
r3 =00000001 r3 =00000001          000003e4 l.sflts r3,r3
r4 =00000200 r5 =ffffffff          000003e8 l.andi r4,r5,0x200
r8 =00002200 r8 =00002200 r4 =00000200 000003ec l.add r8,r8,r4
r3 =00000001 r4 =00000200          000003f0 l.sflts r3,r4
r4 =00000200 r5 =ffffffff          000003f4 l.andi r4,r5,0x200
r8 =00002400 r8 =00002400 r4 =00000200 000003f8 l.add r8,r8,r4
r3 =00000001 r3 =00000001          000003fc l.sfiles r3,r3
r4 =00000200 r5 =ffffffff          00000400 l.andi r4,r5,0x200
r8 =00002600 r8 =00002600 r4 =00000200 00000404 l.add r8,r8,r4
r3 =00000001 r4 =00000200          00000408 l.sfiles r3,r4
r4 =00000200 r5 =ffffffff          0000040c l.andi r4,r5,0x200
r8 =00002800 r8 =00002800 r4 =00000200 00000410 l.add r8,r8,r4
r3 =00002800          r8 =00002800 00000414 l.or r3,r0,r8
          00000418 l.nop 0x2

-----          200 instruction -----
r9 =1243d0a9 EA =00000000          0000041c l.lwz r9,0x0(r31)
r8 =1243f8a9 r9 =1243d0a9 r8 =1243f8a9 00000420 l.add r8,r9,r8
EA =00000000 r8 =1243f8a9          00000424 l.sw 0x0(r31),r8
r1 =00000000          00000428 l.addi r1,r0,0x0
r2 =00000001          0000042c l.addi r2,r0,0x1
r8 =00000000          00000430 l.addi r8,r0,0x0
          00000434 l.j 0x4
r8 =00000001 r8 =00000001          00000438 l.addi r8,r8,0x1
          00000444 l.jal -2
r8 =00000002 r8 =00000002          00000448 l.addi r8,r8,0x1
r9 =0000044c          0000043c l.jr r9
r8 =00000003 r8 =00000003          00000440 l.addi r8,r8,0x1
          r1 =00000000          0000044c l.sfeq r0,r1

```


| | | | |
|--------------|--------------|--------------|-----------------------------|
| | | | 00000450 l.bf 0x2 |
| r8 =00000004 | r8 =00000004 | | 00000454 l.addi r8,r8,0x1 |
| | r2 =00000001 | | 00000458 l.sfeq r0,r2 |
| | | | 0000045c l.bf 0x3 |
| r8 =00000005 | r8 =00000005 | | 00000460 l.addi r8,r8,0x1 |
| r8 =00000006 | r8 =00000006 | | 00000464 l.addi r8,r8,0x1 |
| | r1 =00000000 | | 00000468 l.sfeq r0,r1 |
| | | | 0000046c l.bnf 0x3 |
| r8 =00000007 | r8 =00000007 | | 00000470 l.addi r8,r8,0x1 |
| r8 =00000008 | r8 =00000008 | | 00000474 l.addi r8,r8,0x1 |
| | r2 =00000001 | | 00000478 l.sfeq r0,r2 |
| | | | 0000047c l.bnf 0x3 |
| r8 =00000009 | r8 =00000009 | | 00000480 l.addi r8,r8,0x1 |
| r3 =00000000 | | | 00000488 l.movhi r3,0 |
| r3 =00000494 | r3 =00000494 | | 0000048c l.ori r3,r3,0x494 |
| r8 =0000000a | r8 =0000000a | | 00000490 l.addi r8,r8,0x1 |
| r3 =0000000a | | r8 =0000000a | 00000494 l.or r3,r0,r8 |
| | | | 00000498 l.nop 0x2 |
| r9 =1243f8a9 | EA =00000000 | | 0000049c l.lwz r9,0x0(r31) |
| r8 =1243f8b3 | r9 =1243f8a9 | r8 =1243f8b3 | 000004a0 l.add r8,r9,r8 |
| EA =00000000 | r8 =1243f8b3 | | 000004a4 l.sw 0x0(r31),r8 |
| r9 =1243f8b3 | EA =00000000 | | 000004a8 l.lwz r9,0x0(r31) |
| r3 =4c690000 | | | 000004ac l.movhi r3,0x4c69 |
| r3 =4c69e5f7 | r3 =4c69e5f7 | | 000004b0 l.ori r3,r3,0xe5f7 |
| r8 =5eaddeaa | r8 =5eaddeaa | r3 =4c69e5f7 | 000004b4 l.add r8,r8,r3 |
| r3 =5eaddeaa | | r8 =5eaddeaa | 000004b8 l.or r3,r0,r8 |
| | | | 000004bc l.nop 0x2 |
| r3 =00000000 | | | 000004c0 l.addi r3,r0,0x0 |
| | | | 000004c4 l.nop 0x1 |

