Design and Implementation of a 32-bit Lite Version ARM ISA CPU

By

Tan Beng Liong

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONS)

COMPUTER ENGINEERING

Faculty of Information and Communication Technology
(Perak Campus)

JAN 2017

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: _____
_____
_____

Academic Session: _____

I _____
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

_____ _____

(Author's signature) (Supervisor's signature)

Address:
_____
_____
_____

_____
Supervisor's name

Date: _____ **Date**: _____

Design and Implementation of a 32-bit Lite Version ARM ISA CPU

By

Tan Beng Liong

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONS)

COMPUTER ENGINEERING

Faculty of Information and Communication Technology
(Perak Campus)

JAN 2017

DECLARATION OF ORIGINALITY

I declare that this report entitled “Design and Implementation of a 32-bit Lite Version ARM ISA CPU” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : _____

Name : _____

Date : _____

ACKNOWLEDGEMENTS

I would like to thank to Mr. Mok Kai Ming for giving me the opportunities to contribute in this project which help me improve my understanding toward the processor design skill.

Beside thank to University Tunku Abdul Rahman for provide the facilities and comfortable environment to all students to complete their project.

ABSTRACT

This project is a processer design with Verilog HDL for academic purpose. The processor is built in pipelined stage and divided to 5 stages which are instruction fetch (IF), instruction decode (ID), instruction execution (EX), memory (MEM) and write back (WB). It contain the methodology, design hierarchy, connection between each blocks and pin description for each blocks. The processor is built based on ARM instruction structure architecture (ISA). To understand the how the instructions work, an ARM assembly stimulator, ARMSim which is free simulator developed by University of Victoria is downloaded, the ARMSim also used to verify the output of the designed Verilog module by comparing the register file and memory content.

The instruction format and addressing mode of each type of instructions in ARM is studied. The data path of the processor is designed according to the addressing modes of the instructions need to implement to the design. However the arithmetic logic unit (ALU) and barrel shifter block which can perform add, subtract, logical shift (LSL and LSR), arithmetic shift right (ASR) and rotate right (ROR) is designed. For memory cache the address of each segment is refer to the memory map stated in Digital Design and Computer Architecture ARM edition by Sarah L. Harris and David Money Harris. Hazard problem in the pipelined register is solved by implement extra blocks instead of using NOP to achieve a better performance. After designed the Verilog module verification is carry out to make sure the processor work.

The verification is done by using 2 converted ARM assembly program with ARMSim, as stated above the content of both register file and memory cache need to be same. First program used is to test the all instruction implemented worked individually however another is converted from c program to verify that the instructions can worked with each other.

TABLE OF CONTENTS

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

RISC	Reduced instruction set computing
CISC	Complex instruction set computing
GUI	Graphic based user interface
ISA	Instruction set architecture
IP	Intellectual property
GPIO	General purpose input/output
IF	Instruction fetch (pipeline stage)
ID	Instruction decode (pipeline stage)
EX	Execute (pipeline stage)
MEM	Memory (pipeline stage)
WB	Write back (pipeline stage)
ALU	Arithmetic logic unit
RTL	Register transfer level
I/O	Input / output
PC	Program counter
UART	Universal asynchronous receiver/ transmitter

Chapter 1 – Introduction

1.1 Project Background

ARM is a computer processors developer company with reduced instruction set computing (RISC) architectures. A RISC-based processor requires lesser transistors than CISC (complex instruction set computing) processor such as x86 processors in most of personal computer. This means reduces in cost, heat produced and power use can be achieving which is importance factor for light, portable and battery-powered devices such as smartphone, laptops, tablet and embedded systems. Most of the cores introduced by ARM support a 32-bits address space except ARMv8-A architectures support 64-bits. ARM licenses their design to companies that incorporate those core designs into their own products.

1.2 ARM's History

ARM is a British company start at 1980 with the name of Acorn Computer at first. Its first product was a coprocessor module for BBC Micro series of computers. Then they start relatively simple MOS Technology 6502 processor in1981. But the 6502 processor is not strong enough for GUI (graphics based user interface), so ARM decides to design their own processor after studies all the lacking of existing processors. Sophie Wilson developed the instruction set and in 1983, the official Acorn RISC Machine with cooperation with VSLI Technology as silicon partner. Then the ARM2 was introduced which enable lower power consumption, but better performance than Intel 80286. And ARM continue introduce ARM3 and ARM6. ARM 3 had better performance than ARM2. But ARM 6, result of cooperation between Apple and ARM manage to remained essentially same size with ARM2 with further better performance; ARM2 had 30,000 transistors, while ARM6 had 35,000.

Architecture	Core bit-width	ARM holding cores
ARMv1	32	ARM1
ARMv2	32	ARM2, ARM250, ARM3
ARMv3	32	ARM6, ARM7
ARMv4	32	ARM8
ARMv4T	32	ARM7TDMI, ARM9TDMI, SecurCore SC100
ARMv5TE	32	ARM7EJ, ARM9E, ARM10E
ARMv6	32	ARM11
ARMv6-M	32	ARM Cortex-M0, ARM Cortex-M0+, ARM Cortex-M1, SecurCore SC000
ARMv7-M	32	ARM Cortex-M3
ARMv7E-M	32	ARM Cortex-M4, ARM Cortex-M7
ARMv8-M	32	ARM Cortex-M23, ARM Cortex-M33
ARMv7-R	32	ARM Cortex-R4, ARM Cortex-R5, ARM Cortex-R7, ARM Cortex-R8
ARMv8-R	32	ARM Cortex-R52
ARMv7-A	32	ARM Cortex-A5, ARM Cortex-A7, ARM Cortex-A8, ARM Cortex-A9, ARM Cortex-A12, ARM Cortex-A15, ARM Cortex-A17
ARMv8-A	32	ARM Cortex-A32
ARMv8-A	32/64	ARM Cortex-A35, ARM Cortex-A53, ARM Cortex-A57, ARM Cortex-A72, ARM Cortex-A73

Table 1.1: List of ARM microarchitectures (Source: https://en.wikipedia.org/wiki/ARM_architecture#Coprocessors)

1.3 Problem Statement & Motivation

The ARM cores project are available on some sources such as www.opencore.org, the ARM information center (infocenter.arm.com), and other website with ARM documentation. But the ARM's core microarchitecture include in the documentation is very limited, hence the functionalities and implementation of ISA to hardware of the cores are not presented well and the Verilog codes included in the project are hard to understand since the microarchitecture are not well presented in documentation and inconvenience naming conversion used. Since there is no proper or complete documentation that described microarchitecture of 32-bit microprocessor with ARM Instruction Set Architecture in open source website. Hence there is only a very limited details can be obtaining from the project which show how the inside parts of processor work together to achieve the specification that had been described in the documentation. This has affected the use of the ARM softcore, in particular for research purpose.

Microchip design companies design microprocessor as IP for commercial purpose. The IP is not available in the market at an affordable price for research purpose. ARM does offers several licensing models for ARM technology-based product but the license will expire within 3 years a payment needed for the license, which is not suitable for a long run project.

Besides, the verification plan for an ARM microprocessor that are made available on the internet is not well defined and yet not compatible to every design. Therefore, there is a necessary to develop a verification plan to verify the functionality of the module designed.

Chapter2 – Literature Review

ARM is a computer processors developer company with reduced instruction set computing (RISC) architectures. A RISC-based processor requires lesser transistors than CISC (complex instruction set computing) processor such as x86 processors in most of personal computer. This means reduces in cost, heat produced and power use can be achieving which is importance factor for light, portable and battery-powered devices such as smartphone, laptops, tablet and embedded systems. Most of the cores introduced by ARM support a 32-bits address space except ARMv8-A architectures support 64-bits. ARM licenses their design to companies that incorporate those core designs into their own products.

2.0 ISA (Instruction Set Architecture) of ARM

ARM instructions support data transfer, arithmetic and programs flow instructions. The table 2.1 below showed the instructions and its function.

Instruction	**Operation**	**Instruction**	**Operation**
add Rd, Rn, Opd2	Rd ← Rn + Opd2	small Rdh, Rn, Rm, Rdl	{Rdh, Rdl} ← Rn* Rm + {Rdh, Rdl}
adc Rd, Rn, Opd2	Rd ← Rn + Opd2 + carry	str Rd, [Rn], +Opd2	Mem[Rn] ← Rd, Rn ← Rn + Opd2
sub Rd, Rn, Opd2	Rd ← Rn - Opd2	str Rd, [Rn], -Opd2	Mem[Rn] ← Rd, Rn ← Rn – Opd2
sbc Rd, Rn, Opd2	Rd ← Rn – Opd2 – (~carry)	str Rd, [Rn, + Opd2]	Mem[Rn + Opd2] ← Rd
rsb Rd, Rn, Opd2	Rd ← Opd2 – Rn	str Rd, [Rn, - Opd2]	Mem[Rn – Opd2] ← Rd
rsc Rd, Rn, Opd2	Rd ← Opd2 – Rn – (~carry)	str Rd, [Rn, + Opd2]!	Rn ← Rn + Opd2, Mem[Rn] ← Rd
tst Rn, Opd2	Set flags based on Rn & Opd2	str Rd, [Rn, - Opd2]!	Rn ← Rn - Opd2, Mem[Rn] ← Rd
teq Rn, Opd2	Set flags based on Rn ^ Opd2	ldr Rd, [Rn], +Opd2	Rd ← Mem[Rn], Rn ← Rn + Opd2
and Rd, Rn, Opd2	Rd ← Rn & Opd2	ldr Rd, [Rn], -Opd2	Rd ← Mem[Rn], Rn ← Rn – Opd2
eor Rd, Rn, Opd2	Rd ← Rn ^ Opd2	ldr Rd, [Rn, + Opd2]	Rd ← Mem[Rn + Opd2]
orr Rd, Rn, Opd2	Rd ← Rn|Opd2	ldr Rd, [Rn, - Opd2]	Rd ← Mem[Rn – Opd2]
bic Rd, Rn, Opd2	Rd ← Rn & (~Opd2)	ldr Rd, [Rn, + Opd2]!	Rn ← Rn + Opd2, Rd ← Mem[Rn]
cmp Rn, Opd2	Set flags based on Rn – Opd2	ldr Rd, [Rn, - Opd2]!	Rn ← Rn - Opd2, Rd ← Mem[Rn]
cmn Rn, Opd2	Set flags based on Rn + Opd2	strb Rd, [Rn], +Opd2	Mem[Rn] ← $Rd_{[7:0]}$, Rn ← Rn + Opd2
asr Rd, Rm, <Rs|sh>	Rd ← Rm >>> (Rs|sh) (Arithmetic)	strb Rd, [Rn], -Opd2	Mem[Rn] ← $Rd_{[7:0]}$, Rn ← Rn – Opd2
lsl Rd, Rm, <Rs|sh>	Rd ← Rm << (Rs|sh) (Logical)	strb Rd, [Rn, + Opd2]	Mem[Rn + Opd2] ← $Rd_{[7:0]}$
lsr Rd, Rm, <Rs|sh>	Rd ← Rm >> (Rs|sh) (Logical)	strb Rd, [Rn, - Opd2]	Mem[Rn – Opd2] ← $Rd_{[7:0]}$

ror Rd, Rm, <Rs|sh>	Rd ← Rn ror (Rs|sh) (Rotate right)	strb Rd, [Rn, + Opd2]!	Rn ← Rn + Opd2, Mem[Rn] ← $Rd_{[7:0]}$
rrx Rd, Rm, <Rs|sh>	{Rd, C} ← {C, Rd} (Rotate right extend)	strb Rd, [Rn, - Opd2]!	Rn ← Rn - Opd2, Mem[Rn] ← $Rd_{[7:0]}$
mov Rd, Opd2	Rd ← Opd2	ldrb Rd, [Rn], +Opd2	Rd ← $Mem[Rn]_{[7:0]}$, Rn ← Rn + Opd2
mvn Rd, Opd2	Rd ← (~Opd2)	ldrb Rd, [Rn], -Opd2	Rd ← $Mem[Rn]_{[7:0]}$, Rn ← Rn – Opd2
mul Rd, Rn, Rm	Rd ← Rn * Rm [31:0]	ldrb Rd, [Rn, + Opd2]	Rd ← $Mem[Rn + Opd2]_{[7:0]}$
mula Rd, Rn, Rm, Ra	Rd ← (Rn*Rm) + Ra [31:0]	ldrb Rd, [Rn, - Opd2]	Rd ← $Mem[Rn – Opd2]_{[7:0]}$
umullRdh, Rn, Rm, Rdl	{Rdh, Rdl} ← Rn *Rm	ldrb Rd, [Rn, + Opd2]!	Rn ← Rn + Opd2, Rd ← $Mem[Rn]_{[7:0]}$
umlalRdh, Rn, Rm, Rdl	{Rdh, Rdl} ← Rn*Rm + {Rdh, Rdl}	ldrb Rd, [Rn, - Opd2]!	Rn ← Rn - Opd2, Rd ← $Mem[Rn]_{[7:0]}$
smullRdh, Rn, Rm, Rdl	{Rdh, Rdl} ← Rn*Rm	b <label>	PC ← label
		bl<label>	LR ← PC+4, PC ← label

Table 2.1: Instruction set of ARM

2.0.1 Instruction Format

The ARM instruction had classified to 4 general formats:

- data-processing instruction format
- memory instruction format
- multiplication instruction format
- Branch instruction format.

The figure 2.1, 2.2, 2.3 and 2.4 show the differences between instruction formats:

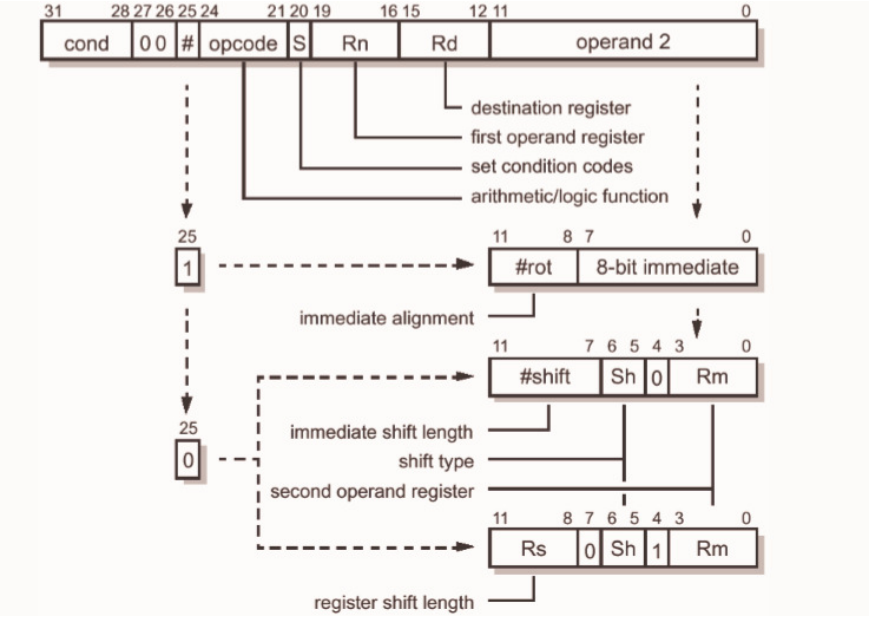


Figure 2.1: Data-processing instruction format

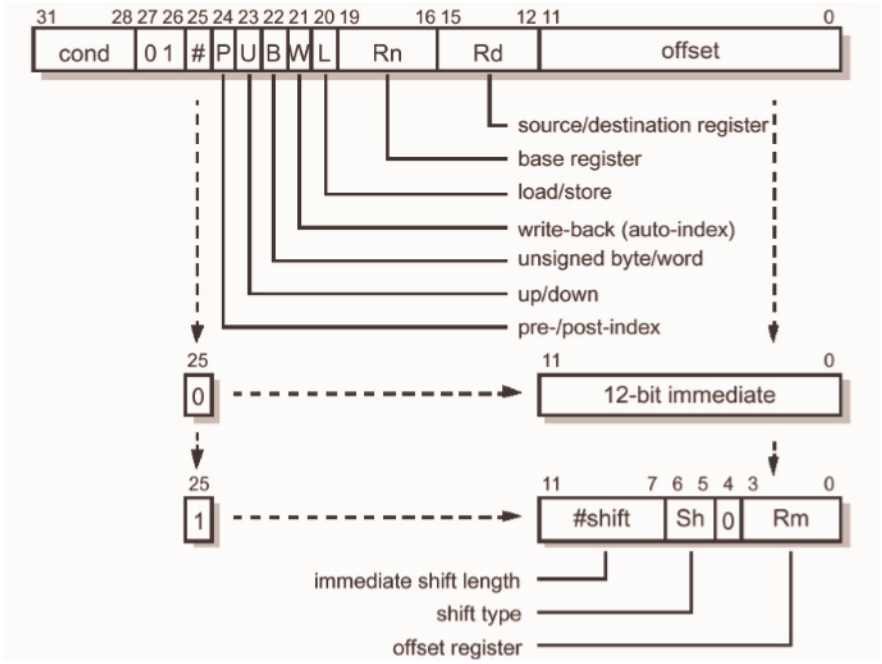


Figure 2.2: Memory instruction format

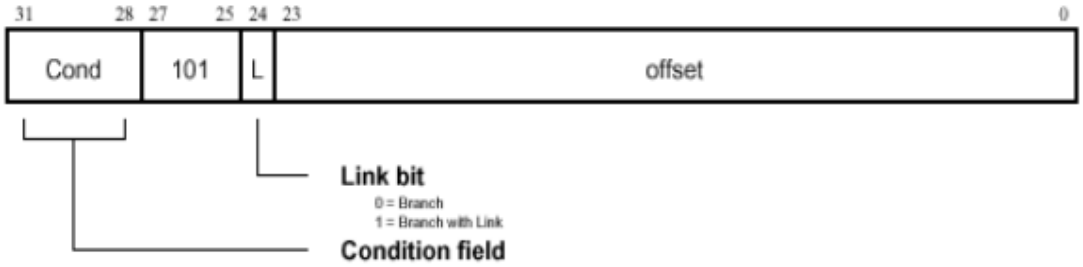


Figure 2.3: Branch instruction format

2.0.2 Condition Encode Instruction

The ARM processor support condition encoding instructions. The condition encoded instruction only will execute when the condition is met with the 4-bits status flag in CPSR updated by previous instructions

Condition	Mnemonic extension	Meaning	Condition flag state
4'h0	Eq	Equal	Z set
4'h1	Ne	Not equal	Z clear
4'h2	cs / hs	Carry set / unsigned higher or same.	C set
4'h3	cc / lo	Carry clear / unsigned lower	C clear
4’h4	Mi	Minus / negative	N set
4’h5	Pl	Plus / positive or zero	N clear
4’h6	Vs	Overflow	V set
4’h7	Vc	No overflow	V clear
4’h8	Hi	Unsigned higher	C set and Z clear
4’h9	Ls	Unsigned lower or same	C clear or Z set
4’h10	Ge	Signed greater than or equal	N == V
4’h11	Lt	Signed lesser than	N != V
4’h12	Gt	Signed greater than	Z == 0, N == V
4’h13	Le	Signed lesser than or equal	Z == 1 or N != V
4’h14	Al	Always (unconditional)	-
4’h15	-	Invalid condition	- or same.

Table 2.2 condition encoding

2.1 Single cycle, multi-cycle and pipelined processor

Single cycle

- The instructions execute and complete in 1 clock cycle.
- No data dependency and hazard problem.
- Longer clock cycle needed to complete 1 instruction.

Multi-cycle

- The instruction subdivided into few steps (depend on instruction)
    - Arithmetic and logical instruction – 4 steps (IF, ID, EX, WB)
    - Save instruction – 4 steps (IF, ID, EX, MEM)
    - Load instruction – 5 steps (IF, ID, EX, MEM, WB)
    - Branch instruction – 2 steps (IF, ID)
    - Branch and link instruction – 3 steps (IF, ID, WB)
- 1 instruction execute at the same time.
- No data dependency and hazard problem.
- In average, shorter clock cycle needed to complete 1 instruction compare to single cycle.

Pipeline

- The instruction subdivided into few steps (maximum step of the instruction)
- Few instructions execute in same time (number of pipeline stage)
- Data dependency and hazard problem (can be solved by implement of addition hardware)
- Execute in clock cycle with number of pipeline stage times shorter than single cycle processor.

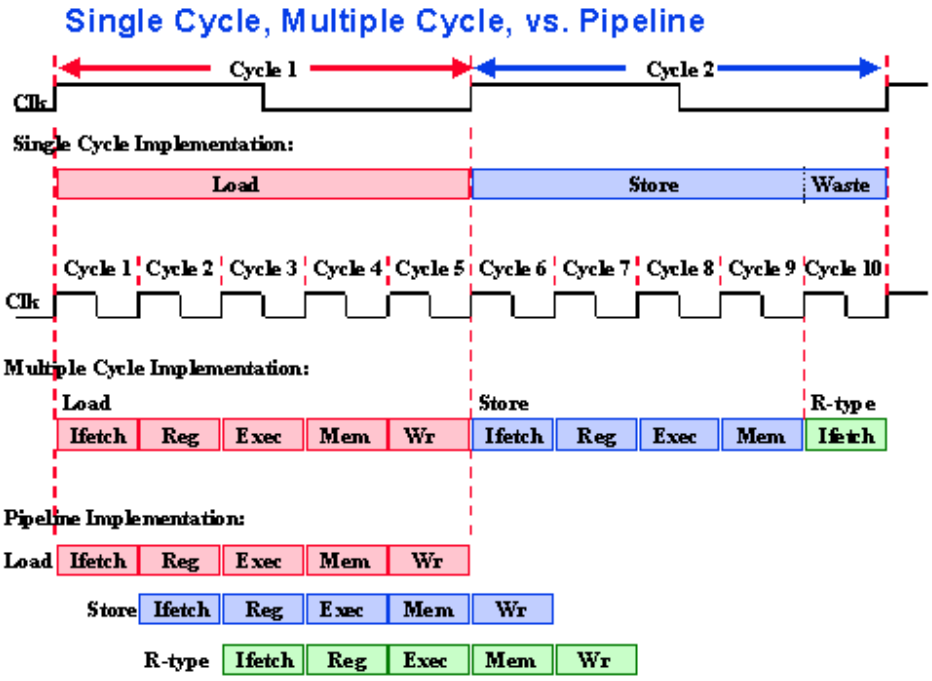


Figure 2.4: Single cycle, multi-cycle vs pipeline processor

2.2 Benchmarking

Two project from www.opencore.org done by ConorSantifort (Amber), and Stephan Nolting (Strom Core) respectively will be used for benchmarking purpose. Beside the ARM7, ARM9, ARM10, and ARM11 introduced by ARM will be used for further benchmarking.

2.2.1 Amber Core

Amber processor is an ARM-compatible 32-bit RISC processor done by ConorSantifort. The Amber core are fully compatible to ARMv2 Instruction set architecture (ISA), the project will develop with Verilog 2001. The Amber project provides a complete embedded system incorporating the Amber core and a number of peripherals, including UARTs, timers, and an Ethernet MAC. There are 2 version of Amber project done which is Amber 23 and Amber 25.

Amber 23 is a 3 –stage pipelined processor which can be represent in fetch, decode and execute. It is capable of 0.8 DMIPS per MHz.

The Amber 25 is a 5-stage pipelined processor which the stages are separate based on fetch, decode, execute, memory, and write-back. Amber 25 have a 15% to 20% better performance compared to the Amber 23 which is 1.0 DMIPS per MHz, but a larger size and more hardware implement needed in Amber 25.

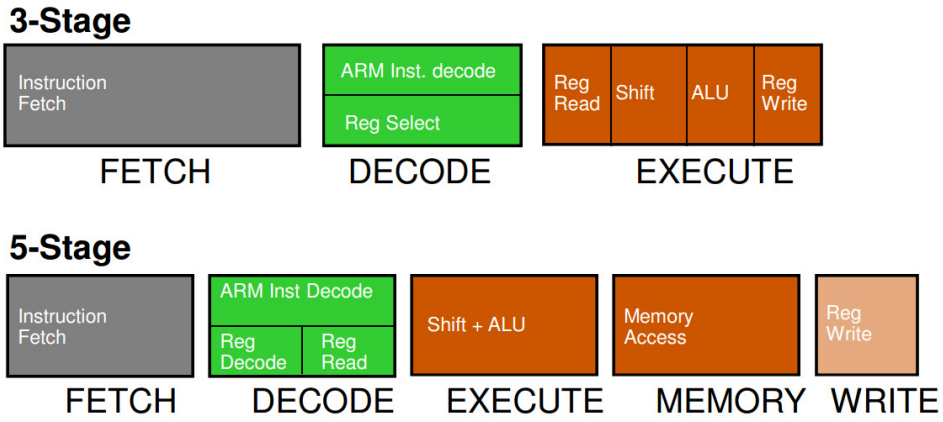


Figure 2.5: 3-Stages and 5-Stages pipeline

ALU in Amber Core

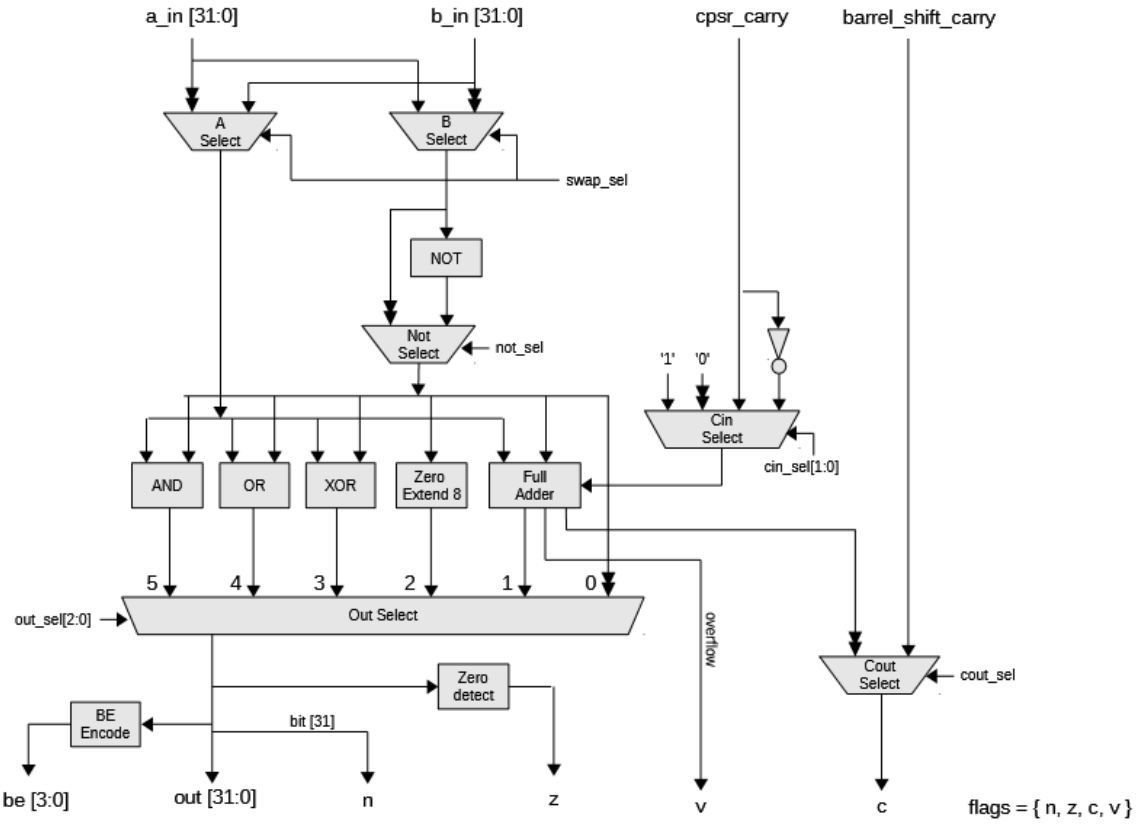


Figure 2.6: Design of ALU in Amber 23

The alu_function[6:0] is the control signals for the ALU. It make up from {swap_sel, not_sel, cin_sel[1:0], cout_sel, out_sel[2:0]}.

Pin	Description
swap_sel	Swap between input a and b.
not_sel	1'b0: use original b, 1'b1: use inverted b
cin_sel[1:0]	Select carry in for the full adder. (1, 0, cpsr_carry, cpsr_carry')
cout_sel	Select carry out for the ALU. 1'b0 : From full adder, 1'b1 from barrel_shift_carry
0ut_sel[2:0]	Select the output for ALU. 3'd0: b 3'd1: adder_out 3'd2: b_zero_extend_8 3'd3: xor_out 3'd4: or_out 3'd5: and_out

Table 2.3: pin description of Amber's ALU

2.2.2 Storm core

The Storm core processor project is done by Stephen Nolting which obtains from www.opencores.org. Same with Amber 23 & 25, Storm also follow ARMv2 instruction architecture with 2 separate caches (Instruction & Data). It is an 8-stage pipelined processor which is instruction access (IA), instruction fetch (IF), instruction decode (ID), operand fetch (OF), multiplication/ shift (MS), execution (EX), memory access (MA), and data write back (WB).

PROESSOR	**AMBER 23**	**AMBER 25**	**STORM**
Opcode and function compatible to	ARMv2	ARMv2	ARMv2
Software compatible?	Yes	Yes	Yes
Pipelined	Yes	Yes	Yes
Number of pipelined stage	3	5	8
Number of cache needed	1	2 (Instruction and memory)	2 (Instruction and memory)
Little /big endian	Little	Little	Both
Wishbone bus system	32-bits	32-bits	32-bits
FPGA implement	Xilinx SP605 Spartan-6 FPGA board	Xilinx SP605 Spartan-6 FPGA board	80 MHz on Xilinx Spartan-3 XC3S400A

Table 2.4: Comparison among Amber 23, Amber 25 &Storm core

2.2.3 ARM7 core

ARM7 core is a 3-stages pipelined processor (Fetch - IF, Decode - ID, Execute - EX) introduced by ARM from 1994 and update periodic. The ARM 7 operate on 32-bits address space. It is compatible to ARMv3 ISA.

Features:

- Register bank:
  1. 2 read ports, 1 write port, access any register.
  2. 1 additional read and write port for r15 (pc).
- Barrel shifter
  1. Shift or rotate the operand by any number of bits.
- ALU.
- Address register and increment.
- Data registers
  1. Hold data passing to and from memory
  2. Instruction decoder and control

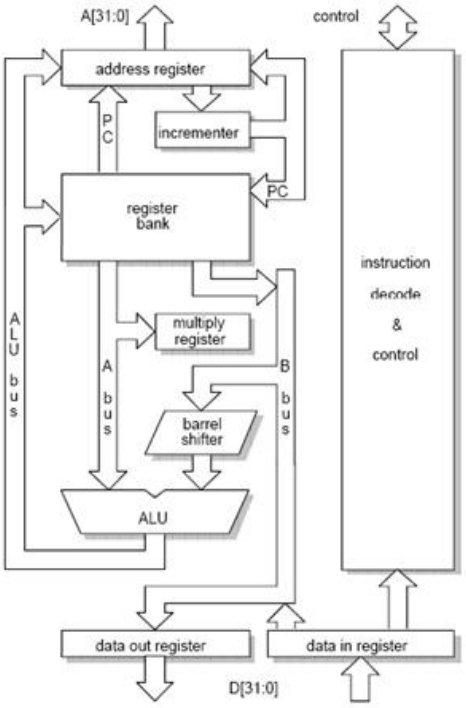


Figure 2.7: data path of ARM 7

2.2.4 ARM9 core

ARM 9 core is 5 stages pipelined processor (instruction fetch-IF, instruction decode-ID, execute-EX, memory access – MEM, data write back - WB). Same with ARM 7, it operate on 32 bits addresses. It is compatible to ARMv5 ISA.

Fetures :

1. Register bank:
    - 3 source operand read ports and 2 write port.
2. Inclusion of address incrementing hardware (for multiple load and store instructions)
3. Memory (Havard architecture)
    - Sepearte instruction and data memory (cache)
4. Higher clock frequency (more pipelined stage)

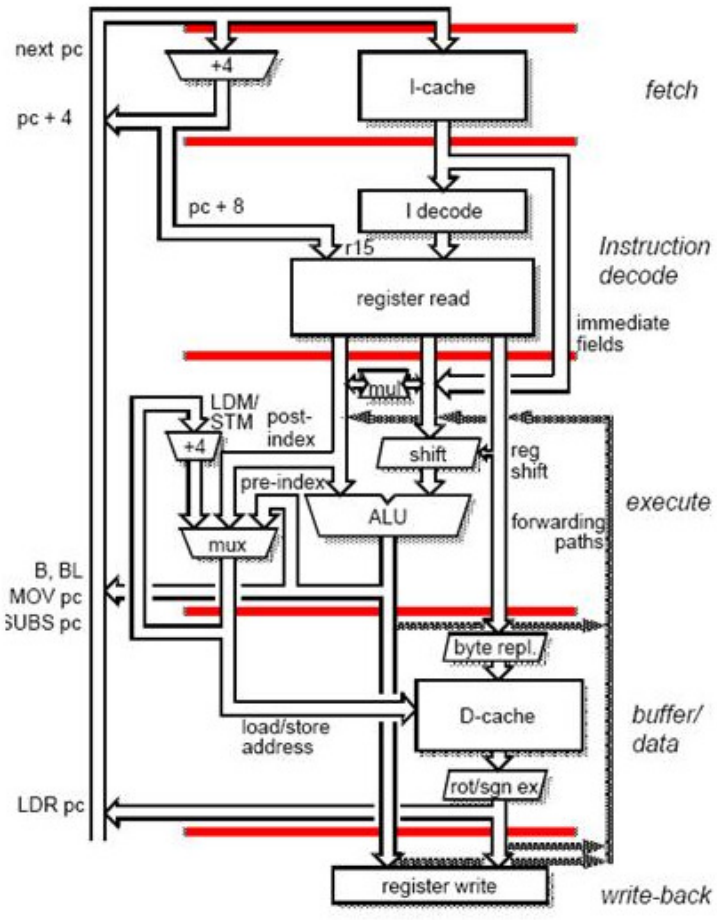


Figure 2.8: data path of ARM 9

In ARM 9 model, data forwarding is allowed to improve the performance. The result are passed between stages as soon as they are available. E.g

ADD r2, r3, r4 //r2 = r3 + r4

ADD r1, r2, r5 //r1 = r2 +r5

The r2 value is immediately forwarded to next operation as soon as it compete the ADD operation by ALU to prevent data Hazard. But for load Hazard problem the data is only ready at the last stage so either insert a NOP or stall the instruction until the data is ready.

2.2.5 ARM10TDMI

ARM10TDMI is a 6 stages pipelined processor. The additional state compared to ARM9 is the issue state (ISS). In issue state, the processor is interpret the instruction fetched from i-cache and determines whether it is an ARM or Thumb instructions. Besides that, ARM10TDMI had hardware to predict branch, which will operate at fetch state to determine the PC value after fetch a branch instruction.

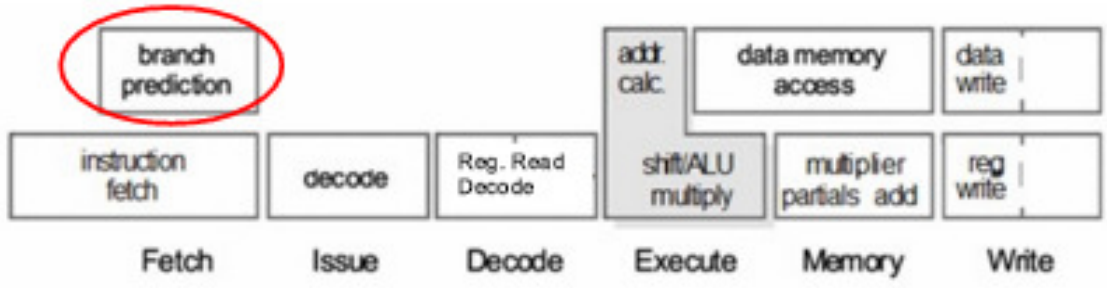


Figure 2.9: pipelined stages of ARM 10 DTMI

2.2.6 ARM11 Core

ARM 11 is an 8-stages pipelined processor. The stages are shown in the figure 2.11.

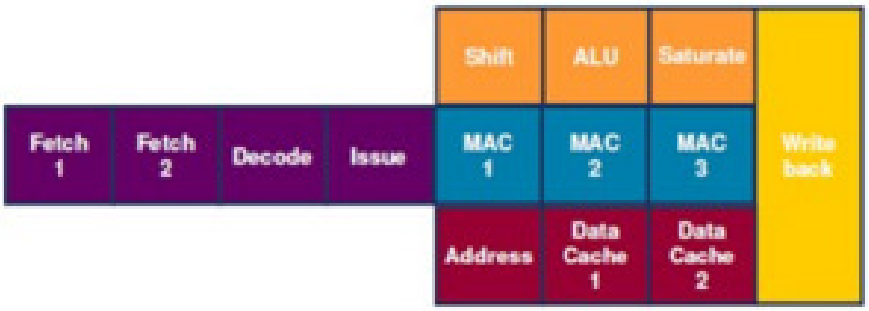


Figure 2.10: pipelined stages of ARM 11

After the issue stage, there is 3 group of different hardware to handle different instructions. The block in orange color is the stage where shift or integer arithmetic instructions go through. While the blocks in blue handle multiplication instructions and block in red will be load/store operation. The ARM11 can maximum handle 4 instructions simultaneously, which is branch prediction, multiplication, ALU operation related instructions and data transfer instructions, which had a much higher performance compared to other ARM core. It also supports data forwarding.

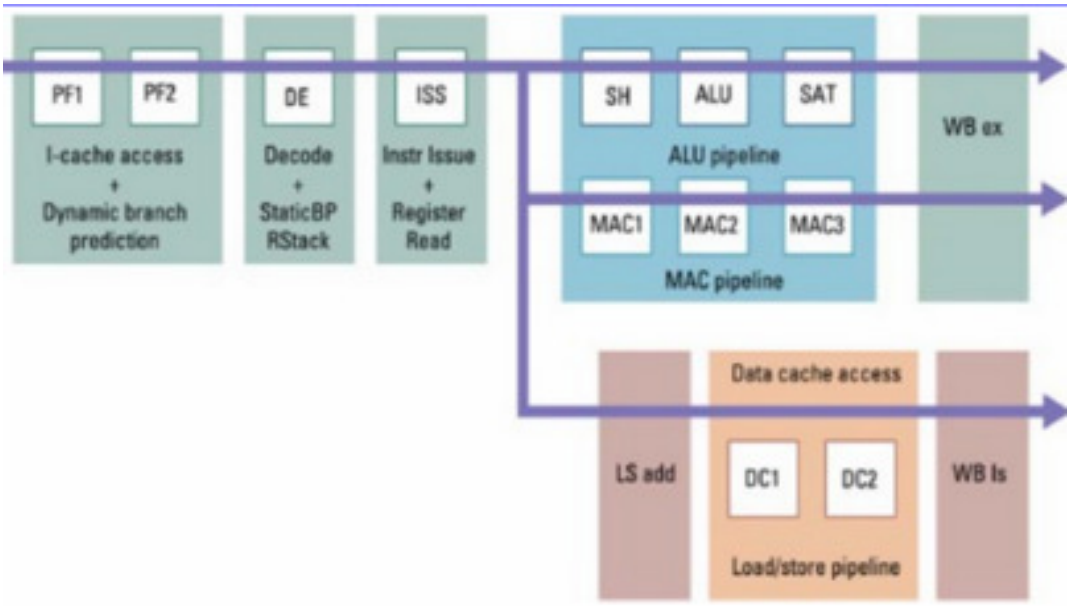


Figure 2.11: grouped pipelined stsges of ARM 11

Features	ARM 7	ARM 9	ARM 10	ARM 11
Pipeline length	3	5	6	8
Java Decode		(ARM9 26 EJ)	(ARM10 26 EJ)	Yes
Branch Prediction	No	No	Static	Dynamic
Independent Load/Store unit	No	No	Yes	Yes
Concurrency	None	None	ALU, MAC, LSU	ALU, MAC, LSU
Architecture	ARMv3	ARMv5TE / ARMv4T	ARMv5TE	ARMv6
Clock Speed	< 130 MHz	130 MHz ~ 200 MHz	300 MHz	1 GHz

Table 2.5: Comparison among ARM7, ARM9, ARM10 & ARM11

Chapter 3 – Project Objective

3.1 Project Scope

The project scope is to modeling and complete verification of the pipelined 32-bit ARM processor, which will be, used as a platform for hardware IP-based research by using Verilog HDL (Hardware Description Language).

The microprocessor model operates on 32-bits data and address. The Instruction Set Architecture used in this project is ARMv2. It consists of three main blocks: control unit, data path unit and memory unit, which will model in Verilog.

After the modeling process, the model will be undergoing verification process to ensure the functionalities and features of the processor. A complete testbench is created to test the functionalities of the whole processor and instructions implemented.

3.2 Objective

The main objective is to design a 32-bits ARM pipelined processors. The sub-objective showed below need to be complete in order to achieve the main objective

- Chip specification: To design an ARM microprocessor which compatible to ARMv2 instruction set architecture (ISA).
- Microarchitecture requirement: To develop an ARM microprocessor, which supports integer arithmetic, multiplication with Booth's algorithm, data-transfer operation, and program flow control instruction.
- RTL: To develop a complete set of Verilog modules that fulfilled and described the microarchitecture requirements above.
- Verification: To create a complete test bench that can verify the all functionalities and instructions implemented to the microprocessor might need remodel of RTL if expected output didn't achieve at the end of verification.

To ensure the processor can be further expand with other research related to ARM architecture, the verification and redesign might need to repeat several time to debug and achieve 100% functionalities.

3.3 Significance and Impacts

The ARM microprocessor will allow researcher to change the micro-architecture based in ARM architecture for experimentation of new design. The microprocessor IP is cheap and affordable with complete documentation. The development environment will allow rapid modeling and verification of experimental hierarchy such as memory, specialized data path, peripherals and etc.

Chapter 4 - Methodology and Technologies Involved

4.1 Design Methodology

Design Methodologies help us to carry out the design work successfully with a set of guidelines. Design methodologies ensure the following (Wolf.W, 2004)

4.2 Universal Design Methodology

Universal Design Methodology (UDM) is a structured method for planning and designing hardware. UDM can be used to design ASICs, FPGAs, CPLDs, and PCBs, in large or small organizations. While some differences occur in designing different hardware types, the basic technique remains the same. The UDM can help:

- Design a that's free from manufacturing defects, that work reliably over device's lifetime and that functions correctly in your system.
- Using least amount of time and resource during design.
- Creating a better schedule on the project.

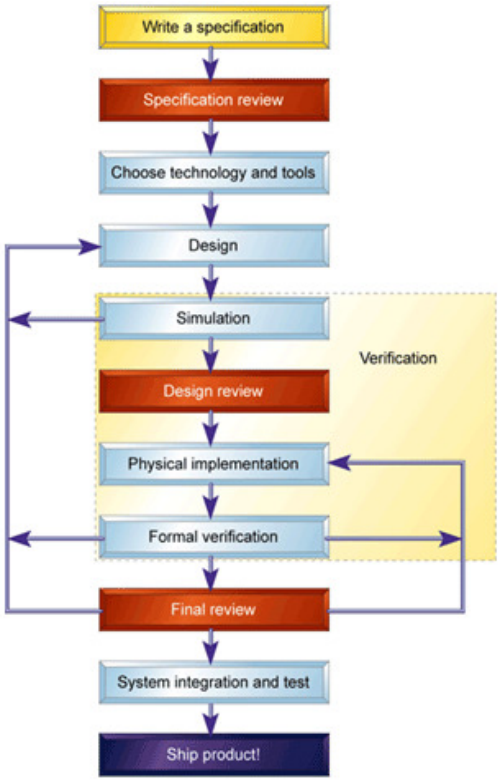


Figure 4.1: UDM flow

Specification and design

The specification need to include:

- External block diagram showing how the device fit into the system.
- Internal block diagram showing each major functional section.
- Description of I/O pins, including output drive capability and input threshold levels
- Timing estimates, including setup and hold times for inputs pins, propagation times for output pins, and clock cycle time.
- Test procedure.

As shown in the diagram after write a specification, a review need to be done in order to know anything being left out or wrong. All functionality decisions must be refer to the specification returned and all subsequent change need to be entered to the specification.

Verification

Verification involved the following stages: simulation, design review, physical implementation, and formal verification. During the simulation, we might need to redesign and repeat the simulation to obtain correct functionality described in specification in earlier state.

After finished the design and simulation, another design review need to be done to make sure whole functionalities include and the accuracy.

Physical implementation stage involves synthesis and place and route but result in a pattern of bits used to program the device.

In formal verification, the physical implementation is checked to ensure the design fully simulated is functionally equivalent to physical implementation of the design.

Completion

The design should be formality with all the steps followed, the final should be a simple sign off. However, the system testing is necessary to ensure that all part of the system work correctly.

4.3 Development Tools

Since in this project will be design by using Verilog HDL therefore the stimulation tool that able to compile and simulate Verilog syntax is necessary. The available and price of the simulation tools are main factor to decide which to be choose. There is a few examples of Verilog development tools and comparison among them:

Simulator	**ISE Simulator**	**ModelSim**	Icarus Verilog
Company / Author	Xilinx	Mentor Graphics	Stephen Williams
Language Support	VHDL-93, Verilog 2001	VHDL, Verilog 2001, System Verilog 2005	Verilog 2001, limited Verilog 2005
Availability for free	No	Yes (For student version)	Yes

Table 4.1: Comparison among Development Tools

ModelSim is chosen from the 3 development since it is available for free and support more language compare to other.

Beside to verify the ARM assembly program, ARMSim (ARM assembly simulator) is used. It will execute the ARM assembly program based on ARM7TDMI processor. ARMSim was developed by Department of Computer Science at the University of Victoria, in Victoria, British Columbia, Canada. It was choosing to use since it was free.

4.4 Design Hierarchy

The module is break into smaller module (chip → unit → block) and each partitioned block and functional verification.

4.5 Implementation Issues and Challenges

1. Data Hazzard:

Happen when there is a data dependency within 5 clock cycle (number of pipelined stages). The result of single assembly instruction (not include multiplication instruction) need 5 clock cycles to write back into register file in a pipelined data path (IF, ID, EX, MEM, WB).

There are 3 situations:

- Read after write (RAW), e.g.

  ADD R1, R2, R3 @EX
  MOV R0, R1 @ID

  The R1 is read during ID stage after ADD R1, R2, R3 instruction where still at EX stage of data path.

- Write after read (WAR), e.g.

  ADD R1, R2, R3 @EX
  SUB R2, R3, R4 @ID

  The R2's value is going to change at SUB R2, R3, R4 but the data is read 1clock cycle earlier before it occurs.

- Write after write (WAW), e.g.

  ADD R1, R2, R3 @EX
  ADD R1, R4, R5 @ID

  The R1 is going to write by 2 instructions, only the result of latest instruction should store in R1.

For, WAR and WAW only will cause a Data Hazards problem when the assembly program executes in concurrent environment. However, this project is no doing a concurrent environment processor therefore only RAW will be the problem to solve.

To solve the problem an extra block (Data forwarding control block) need to implement in the data path to control the data flow.

E.g.

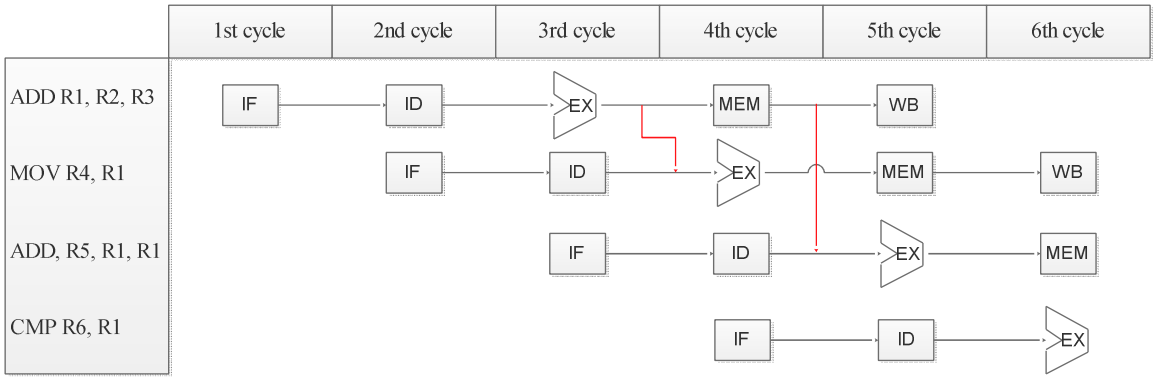


Figure 4.2: Pipeline stage of instruction in different cycle (1)

Note: When execute CMP R6, R1 the new value already wrote to the R1 (half cycle), therefore there is no need data forwarding.

2. Bypassing backwards in time:

There is a bypassing backwards problem when the data from memory is use as operand in next instruction, e.g.

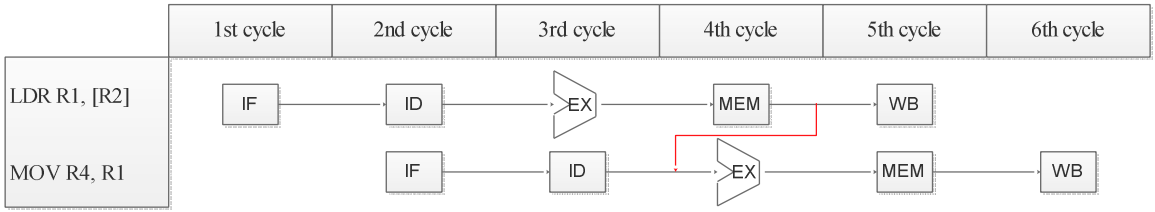


Figure 4.3: Pipeline stage of instruction in different cycle (2)

The data from the memory is not ready yet. Data only read from memory at the end of 4th clock cycle but the data is needed at early of 4th clock cycle. Therefor a stall added with the implementation of interlock block.

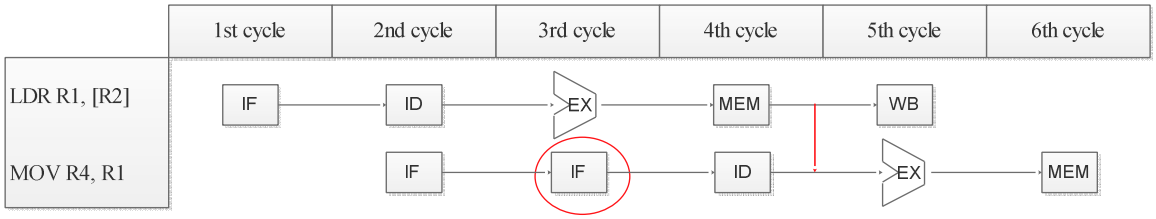


Figure 4.4: Pipeline stage of instruction in different cycle (2)

The instruction delayed 1 clock cycle to complete.

3. PC as destination register for Data-processing instruction:

PC is one of the 16 registers in the register file therefor it also can use as destination of Data-processing instruction such as MOV PC, LR. This make four NOP needed to insert after the instruction until the PC being updated e.g.

MOV PC, LR

NOP

NOP

NOP

NOP

To solve the problem, we can update the PC after ID stage as long as no need ALB (EX), and data from memory (MEM). The instruction which can be improve are only MOV and MVN (without LSL, LSR, ASR and ROR)

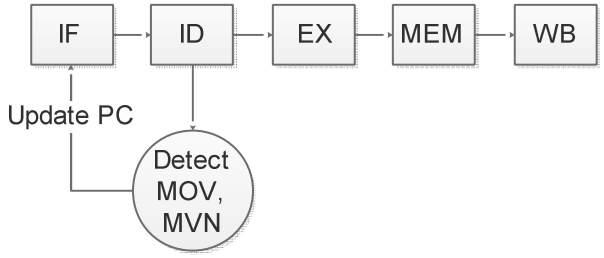


Figure 4.5: MOV and MVN detector

Note: MOVS PC is not supposed to use in User mode. It will affect CPSR and SPSR.

4.6 Schedule and timeline

	FYP1 (May 2016)														FYP2 (Jan 2017)												
Week	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Background Study																											
Specification of design																											
Design Verilog module																											
Develop verification code																											
Verify the design																											
Re-design of Verilog module (if needed)																											
UART implementation																											
Project proposal/ result report																											

Table 4.3: Gantt chart for project 1 & 2.

Chapter 5 – System Specification

Chip level design: RISC32 processor

5.1 Feature

	RISC32
Dummy Instruction Cache (KB)	16
Dummy Data Cache (KB)	16
Data width (bits)	32
Instruction width (bits)	32
General Purpose Register	16
Special Purpose Register	Status flag registers
Pipelined Stage	5
Hazard Handling	Yes
Interlock Handling	Yes
Data Dependency Forwarding	Yes
Branch Prediction	No
Multiplication (size of multiplier and multiplicand)	No
Branch Delay Slot	Not supported
Instruction supported	27

Table 5.1 RISC32 features

5.2 Naming Convention

Instantiation
- [lvl][abbr. mod. name]
- E.g. udp → [unit][data path]

Pin
- [lvl][Type][abbr. mod. name]_[pin name]
- E.g. uidp_imm → [unit][input][data path]_[immediate]

Wire
- [lvl][abbr. mod. name]_[stage]_[pin name]
- E.g. udp_ex_out → [unit][data path]_[EX stage]_[ALU output]

Pipeline register
- [lvl][abbr. mod. name]_[pre-stage][post-stage]_[pin name]
- E.g. udp_ifid_instr → [unit][data path]_[IF stage][ID stage] _[instruction's contain]

Abbreviation:

	Description	Case	Available	Remark
lvl	Level	lower	c : Chip u : Unit b : Block	
abbr. mod. name	Abbreviated module name	lower all	any	e.g. dp – data path
type	Pin type	lower	o : output i : input r : register w : wire f- :function	
stage	Stage name	lower all	if, id, ex, mem, wb	Only for data path module
pin name	Pin name	lower all	any	Several word separate by "_"
pre-stage	Stage name before pipeline		if, id, ex, mem, wb	
Post-stage	Stage name after pipeline		if, id, ex, mem, wb	

Table 5.2 Naming Convention

5.3 RISC32 processor

5.3.1 Processor Interface

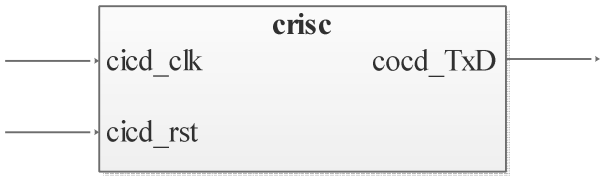


Figure 5.1 Block diagram for RISC32 processor

5.3.2 I/O Pin Description

Pin name : cicd_clk Registered : No Pin class : clock signal Source → Destination : external → crisc Bit size : 1-bit Active : Rising edge Pin Function: Provide a periodic signal for synchronize purpose.
Pin name : cicd_rst Registered : No Pin class : control signal Source → Destination : external → crisc Bit size : 1-bit Active : Active high Pin Function: 1'b 0: normal operation. 1'b 1: reset the chip.

Table 5.3: RISC32 Input Pins Description

Pin name : cocd_TxD Registered : Yes Pin class : data signal Source → Destination : crisc → external device Bit size : 1-bit Active : - Pin Function: Data transmission from UART to external device

Table 5.4: RISC32 Output Pins Description

5.4 System Register

5.4.1 General Purpose Register

Width : 32-bit

Size : 16 units

Retrieving method : 4-bit address as index

Name	Address	Use	Preserved Across A Call?
R0	0	Argument/ return value/ temporary variable	No
R1-R3	1 - 3	Argument/ temporary variable	No
R4-R11	4 – 11	Saved variable	Yes
R12	12	Temporary variable	No
R13 (SP)	13	Stack pointer	Yes
R14 (LR)	14	Link register	Yes
R15 (PC)	15	Program counter	No

Table 5.5 Register file

5.4.2 Special Purpose Register

Width : 1-bit

Size : 4-units

Name	Use
Carry Flag (C)	Carry out of the ALB
Overflow Flag (V)	Set when there is an overflow
Zero Flag (Z)	Set when the result of ALB is zero
Negative Flag (N)	Set when the result of ALB is negative

Table 5.6Status Flag Register

5.5 Instruction Format

The ARM instruction had classified to 3 general formats:

- data-processing instruction format
- memory instruction format
- Branch instruction format.

The figure 5.2, 5.3, and 5.4 show the differences between instruction formats:

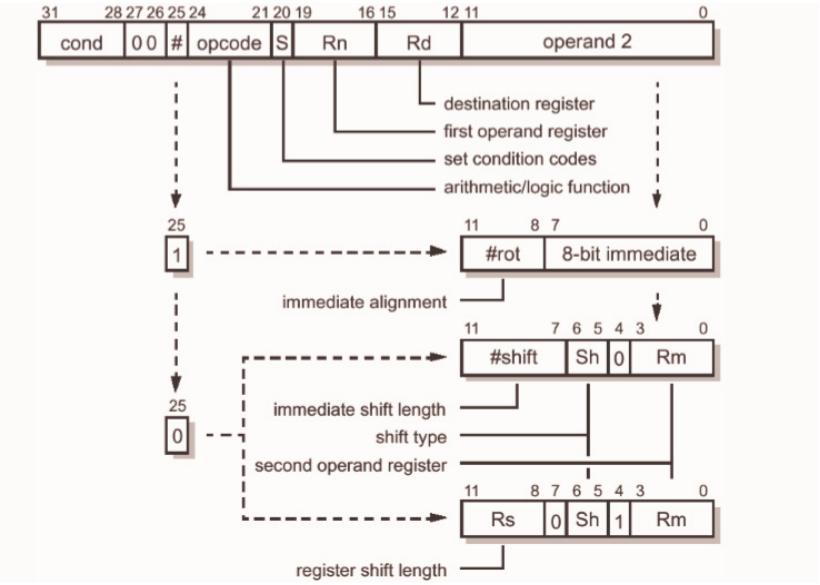


Figure 5.2: Data-processing instruction format

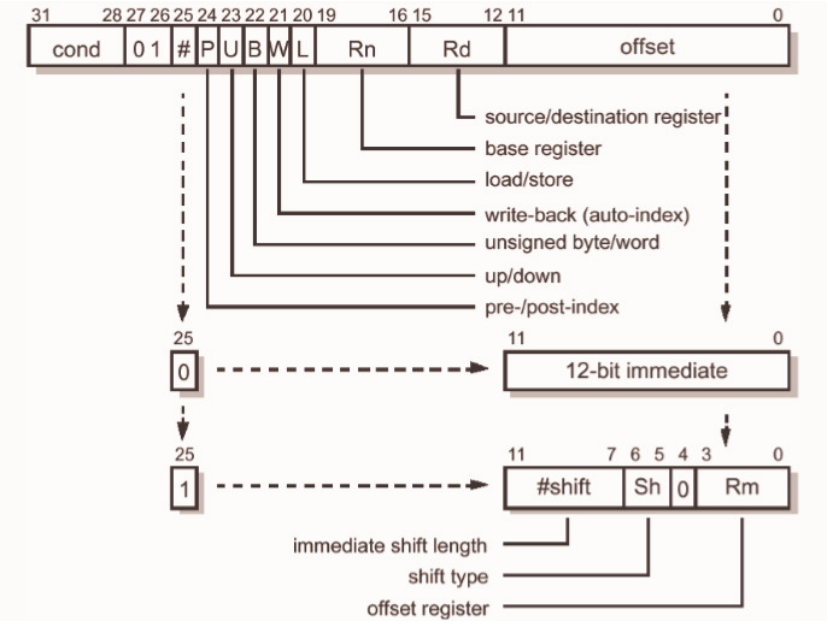


Figure 5.3: Memory instruction format

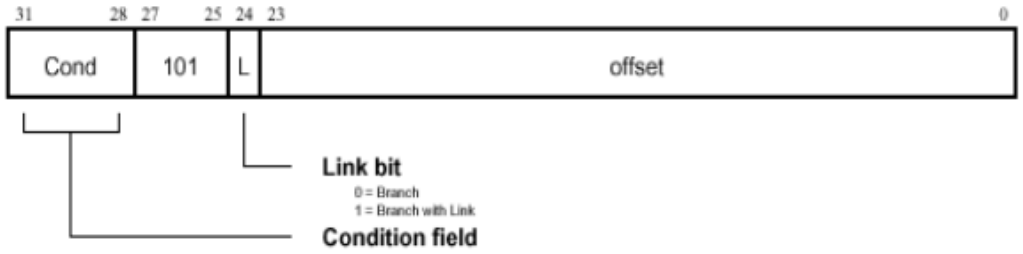


Figure 5.4: Branch instruction format

5.6 Addressing Mode

- Instruction field repetitive
  - ❖ I (instruction[25]) :
    - If 1'b1: indicate immediate addressing mode
    - If 1'b0: indicate register addressing mode
  - ❖ cond (instruction[31:28]): determine whether to execute the instruction or not depend on the status flag.
  - ❖ op (instruction[27:26]): 2'b 00 → Data-processing instruction
    2'b 01 → Memory instruction
    2'b 10 → Branch instruction
  - ❖ funct/cmd (instruction[24:21]): Indicate which logical or arithmetic instruction to be perform.
  - ❖ S (instruction [20]): Update the status flag if 1'b1 else hold the status flag.
  - ❖ Rn (instruction [19:16]): 1st operand register address.
  - ❖ Rd (instruction [15:12]): Destination register address.
  - ❖ rot (instruction [11:8]): amount of rotate.
  - ❖ imm_8 [7:0] (instruction [7:0]): 8-bit immediate. (data-processing instruction)

Rotation value rot	**32-bit immediate value**
4'h0	{ 24'h0, imm_8[7:0]}
4'h1	{ imm_8[1:0], 24'h 0, imm_8[7:2]}
4'h2	{ imm_8[3:0], 24'h 0, imm_8[7:5]}
4'h3	{ imm_8[5:0], 24'h 0, imm_8[7:6]}
4'h4	{ imm_8[7:0], 24'h 0}
4'h5	{ 2'h0, imm_8[7:0], 22'h0}
4'h6	{ 4'h0, imm_8[7:0], 20'h0}
4'h7	{ 6'h0, imm_8[7:0], 18'h0}
4'h8	{ 8'h0, imm_8[7:0], 16'h0}
4'h9	{ 10'h0, imm_8[7:0], 14'h0}
4'h10	{ 12'h0, imm_8[7:0], 12'h0}
4'h11	{ 14'h0, imm_8[7:0], 10'h0}
4'h12	{ 16'h0, imm_8[7:0], 8'h0}
4'h13	{ 18'h0, imm_8[7:0], 6'h0}
4'h14	{ 20'h0, imm_8[7:0], 4'h0}
4'h15	{ 22'h0, imm_8[7:0], 2'h0}

Table 5.7: Encoded immediate value

  - ❖ imm_12 (instruction [11:0]): 12-bit immediate value.

- Rs (instruction [11:8]): 3rd operand register address
- sh (instruction [6:5]): 2'b 00 → LSL
  2'b 01 → LSR
  2'b 10 → ASR
  2'b 11 → ROR
- Rm (instruction [4:0]): 2nd operand register address
- shamt (instruction [11:7]): shift amount (1-31)
- P (instruction [24]): Post index or pre-index
- U (instruction [23]): minus or plus offset
- B (instruction [22]): Byte (if 1'b1)
- W (instruction [21]): Word (if 1'b1)
- L (instruction [20]): Load if 1'b1 else Store (memory instruction)
- Offset (instruction [23:0]): 24-bit value of offset
- L (instruction [24]) (program flow instruction):
  - If 1'b1: store PC+4 to Link Register
  - If 1'b0: hold Link Register's value.

- *Immediate Addressing*, where operand is constant within the instruction itself (show in figure 5.5). E.g. (Note: when sh = 2'b 11, shifter will perform rotation)
  - ADD Rd, Rn, #16
  - MOV Rd, #16

- *Register Addressing*, where operand is a register (show in figure 5.6), the 2nd can be shift according value stored in Rs register or a 5-bit immediate. E.g.
  - MOV Rd, Rm, LSR Rs @ sh → LSR → 2'b01
  - MOV Rd, Rm, LSR #4 @ sh → LSR → 2'b01
  - MOV Rd, Rm @ sh default to LSL → 2'b00, shamt → 0

- *Based Displacement Addressing*, where operand is at the memory location whose address is value stored in a register (show in figure 5.7). E.g.
  - LDR Rd, [Rn]

- *Register indexed displacement addressing with register scaling*, where the operand is at the memory location whose address is the sum of a register with base address (Rn) and register with offset address (Rm). The offset can be shift depend on the instruction, e.g.
  - LDR Rd, [Rn, Rm, LSL #2] @ sh → LSL → 2'b00
  - STR Rd, [Rn, Rm] @ sh default to LSL → 2'b00, shamt → 0

- *Register indexed displacement addressing with immediate scaling*, where the operand is at the memory location whose address is the sum of a register with base address (Rn) and signed extend immediate value of offset address which carry by instruction itself. E.g.
  - LDR Rd, [Rn, #4]

- *Pseudodirect Addressing*, where the jump address is the 24-bit of the instruction concatenated with the upper bits of the PC (show in figure 5.10). E.g
  - BL label @ label's target address → 0xff fff0

5.7 Instruction Set and Description

Operation		Assembler	Machine Language								S update (condition flag)	Register Transfer notation
			31:28 Cond	27:26 op	25 I	24:21 cmd	20 S	19:16 Rn	15:12 Rd	11:0 Src2		
Add	Add	ADD{S} Rd, Rn, <Operand2>	1110	00	A	0100	A	Rn	Rd		N Z C V	Rd ← Rn + Operand2
	With carry	ADC{S} Rd, Rn, <Operand2>	1110	00	A	0101	A	Rn	Rd		N Z C V	Rd ← Rn + Operand2 + C
Subtract	Subtract	SUB{S} Rd, Rn, <Operand2>	1110	00	A	0010	A	Rn	Rd		N Z C V	Rd ← Rn – Operand2
	With carry	SBC{S} Rd, Rn, <Operand2>	1110	00	A	0110	A	Rn	Rd		N Z C V	Rd ← Rn – Operand2 – C’
	Reverse subtract	RSB{S} Rd, Rn, <Operand2>	1110	00	A	0011	A	Rn	Rd		N Z C V	Rd ← Operand2 – Rn
	Reverse subtract with carry	RSC{S} Rd, Rn, <Operand2>	1110	00	A	0111	A	Rn	Rd		N Z C V	Rd ← Operand2 – Rn – C’
Logical	Test	TST Rn, <Operand2>	1110	00	A	1000	1	Rn	xxxx		N Z C	Set flags based on Rn & Src2
	Test equivalence	TEQ Rn, <Operand2>	1110	00	A	1001	1	Rn	xxxx		N Z C	Set flags based on Rn ^ Src2
	Bitwise AND	AND{S} Rd, Rn, <Operand2>	1110	00	A	0000	A	Rn	Rd		N Z C	Rd ← Rn & Operand2
	Bitwise XOR	EOR{S} Rd, Rn, <Operand2>	1110	00	A	0001	A	Rn	Rd		N Z C	Rd ← Rn ^ Operand2
	Bitwise OR	ORR{S} Rd, Rn, <Operand2>	1110	00	A	1100	A	Rn	Rd		N Z C	Rd ← Rn | Operand2
	Bitwise Clear	BIC{S} Rd, Rn, <Operand2>	1110	00	A	1110	A	Rn	Rd		N Z C	Rd ← Rn & (~Operand2)
Compare	Compare	CMP Rn, <Operand2>	1110	00	A	1010	1	Rn	xxxx		N Z C V	Set flags based on Rn - Src2
	Negative	CMN Rn, <Operand2>	1110	00	A	1011	1	Rn	xxxx		N Z C V	Set flags based on Rn + Src2
Move data	Move	MOV{S} Rd, <Operand2>	1110	00	1	1101	A	Rn	Rd		N Z C	Rd ← Operand2
	Not	MVN{S} Rd, <Operand2>	1110	00	X	1111	A	Rn	Rd		N Z C	Rd ← ~(Operand2)

Table 5.8 Data-processing Instruction Set and Description

Note: A – available for both 1 and 0. Refer to table 5.8.

Operand 2	I (instruction[25])	Instruction bits												RTL	Addressing mode
		11	10	9	8	7	6	5	4	3	2	1	0		
#4	1	0 (rot)				4 (8-bit immediate)								Operand 2 = 4	Immediate addressing
Rm	0	0 (shamt)					00 (sh)		0	Rm				Operand 2 = Rm	Register addressing(2)
Rm, LSL #shamt	0	shamt					00 (sh)		0	Rm				Operand 2 = Rm << shamt	Register addressing(2)
Rm, LSR #shamt		shamt					01 (sh)							Operand 2 = Rm >> shamt	
Rm, ASR #shamt		shamt					10 (sh)							Operand 2 = Rm >> shamt	
Rm, ROR #shamt		shamt					11 (sh)							Operand 2 = Rm ror shamt	
Rm, LSL Rs	0	Rs				0	00 (sh)		1	Rm				Operand 2 = Rm << Rs	Register addressing(1)
Rm, LSR Rs							01 (sh)							Operand 2 = Rm >> Rs	
Rm, ASR Rs							10 (sh)							Operand 2 = Rm >>> Rs	
Rm, ROR Rs							11 (sh)							Operand 2 = Rm ror Rs	

Table 5.9 Operand 2 for data processing instruction

Operation		Assembler	Machine Language											S update (condition flag)	Register Transfer notation
			31:28 cond	27:26 op	25 Ī	24 P	23 U	22 B	21 W	20 L	19:16 Rn	15:12 Rd	11:0 Src2		
Store register	Post-index	STR Rd, [Rn], + Src2	1110	01	A	0	1	0	0	0	Rn	Rd		-	Mem [Rn] ← Rd, Rn←Rn+Src2
		STR Rd, [Rn], – Src2	1110	01	A	0	0	0	0	0	Rn	Rd			Mem [Rn] ← Rd, Rn←Rn-Src2
	Offset	STR Rd, [Rn, + Src2]	1110	01	A	1	1	0	0	0	Rn	Rd			Mem [Rn + Src2] ← Rd
		STR Rd, [Rn, – Src2]	1110	01	A	1	0	0	0	0	Rn	Rd			Mem [Rn - Src2] ← Rd
	Pre-index	STR Rd, [Rn, + Src2]!	1110	01	A	1	1	0	1	0	Rn	Rd			Rn←Rn+Src2, Mem [Rn] ← Rd
		STR Rd, [Rn, – Src2]!	1110	01	A	1	0	0	1	0	Rn	Rd			Rn←Rn-Src2, Mem [Rn] ← Rd
Load register	Post-index	LDR Rd, [Rn], +Src2	1110	01	A	0	1	0	0	1	Rn	Rd		-	Rd ← Mem [Rn], Rn←Rn+Src2
		LDR Rd, [Rn], – Src2	1110	01	A	0	0	0	0	1	Rn	Rd			Rd ← Mem [Rn], Rn←Rn-Src2
	Offset	LDR Rd, [Rn, + Src2]	1110	01	A	1	1	0	0	1	Rn	Rd			Rd ← Mem [Rn + Src2]
		LDR Rd, [Rn, – Src2]	1110	01	A	1	0	0	0	1	Rn	Rd			Rd ← Mem [Rn - Src2]
	Pre-index	LDR Rd, [Rn, + Src2]!	1110	01	A	1	1	0	1	1	Rn	Rd			Rn←Rn+Src2, Rd ← Mem[Rn]
		LDR Rd, [Rn, – Src2]!	1110	01	A	1	0	0	1	1	Rn	Rd			Rn←Rn-Src2, Rd ← Mem[Rn]
Store register byte	Post-index	STRB Rd, [Rn], + Src2	1110	01	A	0	1	1	0	0	Rn	Rd		-	Mem [Rn] ← $Rd_{7:0}$, Rn←Rn+Src2
		STRB Rd, [Rn], – Src2	1110	01	A	0	0	1	0	0	Rn	Rd			Mem [Rn] ← $Rd_{7:0}$, Rn←Rn-Src2
	Offset	STRB Rd, [Rn, + Src2]	1110	01	A	1	1	1	0	0	Rn	Rd			Mem [Rn + Src2] ← $Rd_{7:0}$
		STRB Rd, [Rn, – Src2]	1110	01	A	1	0	1	0	0	Rn	Rd			Mem [Rn - Src2] ← $Rd_{7:0}$
	Pre-index	STRB Rd, [Rn, + Src2]!	1110	01	A	1	1	1	1	0	Rn	Rd			Rn←Rn+Src2, Mem [Rn] ← $Rd_{7:0}$
		STRB Rd, [Rn, – Src2]!	1110	01	A	1	0	1	1	0	Rn	Rd			Rn←Rn-Src2, Mem [Rn] ← $Rd_{7:0}$
Load register byte	Post-index	LDRB Rd, [Rn], +Src2	1110	01	A	0	1	1	0	1	Rn	Rd		-	Rd ← Mem [Rn] $_{7:0}$, Rn←Rn+Src2
		LDRB Rd, [Rn], – Src2	1110	01	A	0	0	1	0	1	Rn	Rd			Rd ← Mem [Rn] $_{7:0}$, Rn←Rn-Src2
	Offset	LDRB Rd, [Rn, + Src2]	1110	01	A	1	1	1	0	1	Rn	Rd			Rd ← Mem [Rn + Src2] $_{7:0}$
		LDRB Rd, [Rn, – Src2]	1110	01	A	1	0	1	0	1	Rn	Rd			Rd ← Mem [Rn - Src2] $_{7:0}$
	Pre-index	LDRB Rd, [Rn, + Src2]!	1110	01	A	1	1	1	1	1	Rn	Rd			Rn←Rn+Src2, Rd ← Mem[Rn] $_{7:0}$
		LDRB Rd, [Rn, – Src2]!	1110	01	A	1	0	1	1	1	Rn	Rd			Rn←Rn-Src2, Rd ← Mem[Rn] $_{7:0}$

Table 5.10 Memory instruction set and description

Note: A – available for both 1 and 0. Refer to table 5.10

Source 2 (Src2)	Ī(instruction[25])	Instruction bits												RTL	Addressing mode
		11	10	9	8	7	6	5	4	3	2	1	0		
none	0	0 (12-bit immediate)												Src2 = 0(none)	Based Displacement Addressing
#4	0	4 (12-bit immediate)												Src2 = 4	Register indexed displacement addressing with immediate scaling
Rm	1	0 (shamt)					00 (sh)		0	Rm				Src2 = Rm	Register indexed displacement addressing with register scaling
Rm, LSR #shamt	1	shamt					01 (sh)		0	Rm				Src2 = Rm >> shamt	

Table 5.11 Source 2 for Memory instruction

Operation		Assembler	Machine Language				S update (condition flag)	Register Transfer notation
			31:28 cond	27:26 op	25:24 1L (funct)	23:0 Imm24		
Branch	Without link	B <Address> 0xff0000	1110	10	10	0xff0000 (24- bits immediate word address)	-	PC←(PC+8)+0xff0000<<2
	With link	BL <Address> 0xff0000	1110	10	11	0xff0000 (24-bit immediate word address)		LR←(PC+8)-4; PC←(PC+8)+0xff0000<<2

Table 5.12 Branch Instructions Set and Description

Note: the addressing mode of branch is pseudodirect addressing.

Condition (Instruction [31:28])	Instruction extension	Meaning	Condition flag state to execute instruction
4'h0	Eq	Equal	Z set
4'h1	Ne	Not equal	Z clear
4'h2	cs / hs	Carry set / unsigned higher or same.	C set
4'h3	cc / lo	Carry clear / unsigned lower	C clear
4'h4	Mi	Minus / negative	N set
4'h5	Pl	Plus / positive or zero	N clear
4'h6	Vs	Overflow	V set
4'h7	Vc	No overflow	V clear
4'h8	Hi	Unsigned higher	C set and Z clear
4'h9	Ls	Unsigned lower or same	C clear or Z set
4'h10	Ge	Signed greater than or equal	N == V
4'h11	Lt	Signed lesser than	N != V
4'h12	Gt	Signed greater than	Z == 0, N == V
4'h13	Le	Signed lesser than or equal	Z == 1 or N != V
4'h14	Al	Always (unconditional)	-
4'h15	-	Invalid condition	- or same.

Table 5.13 condition encoding

5.8 Memory Map

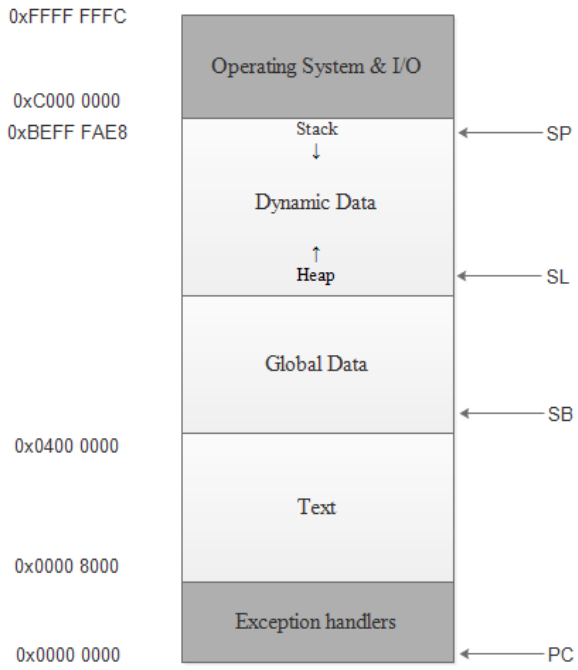


Figure 5.11: Memory Map

- Text Segment
  - Store machine language program.
  - Also known as read only (RO) segment.
- Global Data Segment
  - Store global data which can access by all functions in a program.
  - Also known as read/write (RW) segment.
  - Access using static base (SB) register that point to the start of global segment.
  - SB is conventionally store in R9.
- Dynamic Data Segment
  - Holds stack and heap.
  - Stack pointer (SP) point to top of stack, normally grow downward.
  - SP store in R13
  - Heap store data allocate by program during runtimes, grow upward.
- Exception Handler, OS, and I/O Segments
  - Reserved for exception vector table.

5.9 Operating Procedure

- Start the system
- Porting sequence of instruction into cache (instruction or data)
- Reset the system for at least 2 clocks
- While release the reset, the system will automatically run the program inside instruction cache
- Observe the waveform from the development tools.

Chapter 6 – Microarchitecture Specification

6.1 Design hierarchy

Chip partitioning at System level	Unit partitioning at Architecture level	Block partitioning at RTL level (Microarchitecture level)
crisc (full chip)	udp (data path)	brf (register file)
		balb_shift (ALU & shifter)
		bitl_ctrl (interlock)
		bfw_ctrl (forwarding)
	ucp (control path)	bmain_ctrl
		binstr_ctrl
	ucache (memory cache)	-
	uuart (UART)	bclkctr
		btx (transmitter)
		brx (receiver)
Structural description	Structural description/ Behavioral description	Behavioral description

Table 6.1 Formation of a design hierarchy for crisc microprocessor through top down design methodology

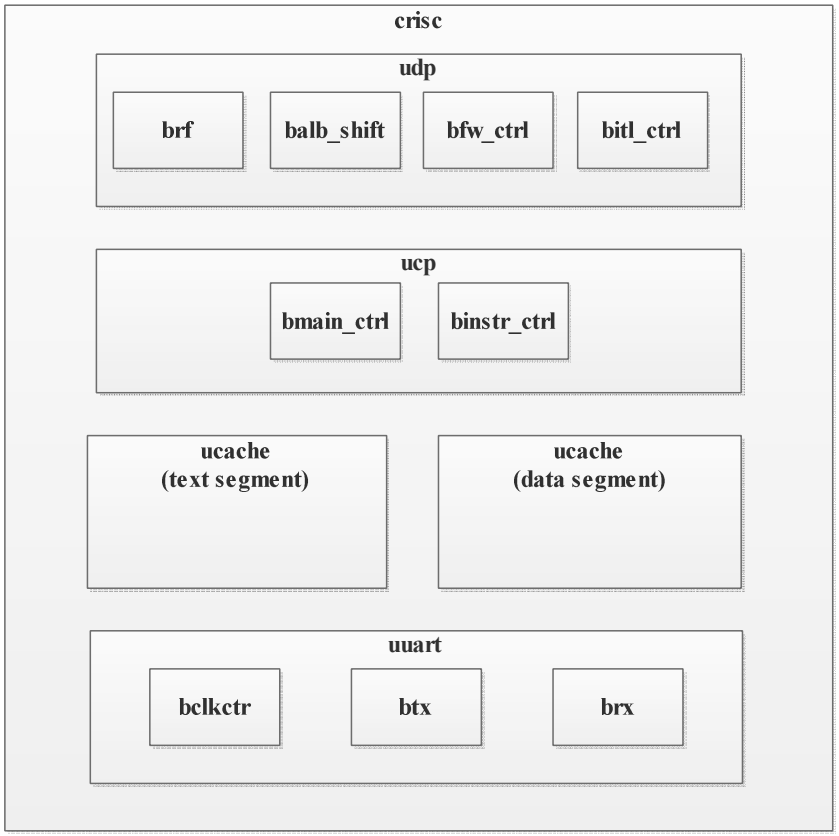


Figure 6.1 crisc architecture and micro-architecture partitioning

6.2 Unit level functional partitioning

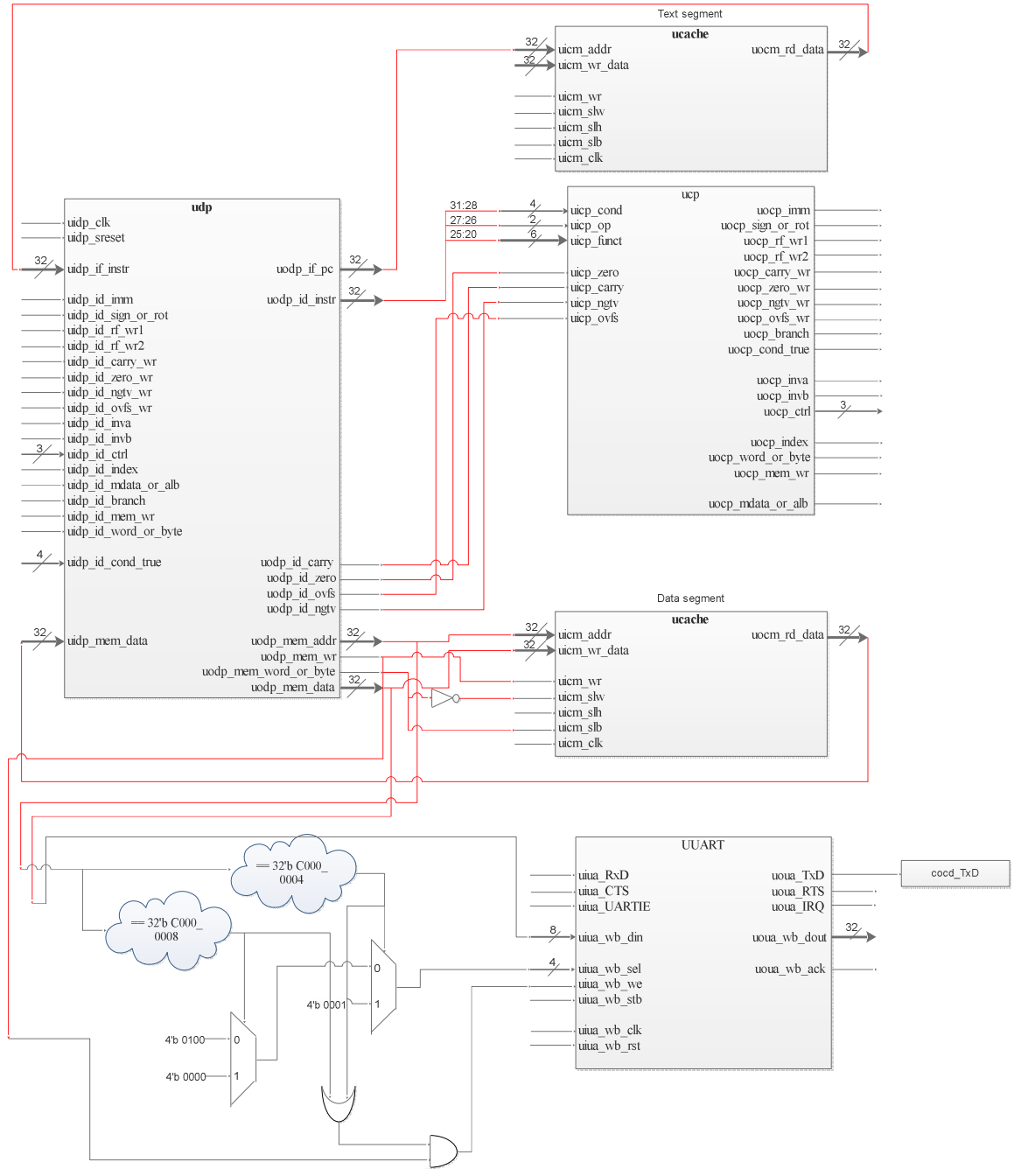


Figure 6.2: unit level functional partitioning of crisc

Chapter 7 – Data path of CRISC (Unit & Block level)

7.1 Feature

Include the addressing mode:

- Register addressing
- Based displacement addressing
- Register indexed displacement addressing with register scaling
- Register indexed displacement addressing with immediate scaling
- Pseudo-direct addressing

Combination of ALB, register file, data forwarding control, and interlock control.

- ALB: perform algorithm and logical operation, generate 4 status flags.
- Register file: 16 Register with width of 32-bit.
- Data forwarding control: overcome data hazard and data dependency problem
- Interlock control: overcome data dependency.

Data dependency in status flags:

2 NOP needed for a branch instruction since the status flag registers generate by ALB in EX stage and store the flag generated at next rising edge of clock. To reduce the NOP the branch instruction should done in 2 stages. Both combination output and register value of status register is used based on the timing of branch instruction (conditional) in the ID stage and status flag register will be place on ID stage and here come the data hazard in status flag register.

1st cycle:

Instruction		Status flag (value in the register)	Status flag (end of EX stage) (combination output)
MOVS R0, #-10	ID	N: 0 C: 0 Z: 0 V: 0	N: 0 C: 0 Z: 0 V: 0
BMI Label	IF		

2nd cycle:

Instruction		Status flag (value in the register)	Status flag (end of EX stage) (combination output)
MOVS R0, #-10	EX	N: 0 C: 0 Z: 0 V: 0	N: 1 C: 0 Z: 0 V: 0
BMI Label	ID		
MOV R0, #0	IF		

3rd cycle:

Instruction	Status flag (value in the register)	Status flag (end of EX stage) (combination output)
MOVS R0, #-10 MEM	N: 1 C: 0 Z: 0 V: 0	N: 1 C: 0 Z: 0 V: 0
BMI Label EX		
MOV R0, #0 ID		

Table 7.1: Status flag problem

The status flag only update after the MOVS R0, #-10 pass EX stage. To solve this the status flag in EX stage is forward to ID so branch can be determined in ID stage.

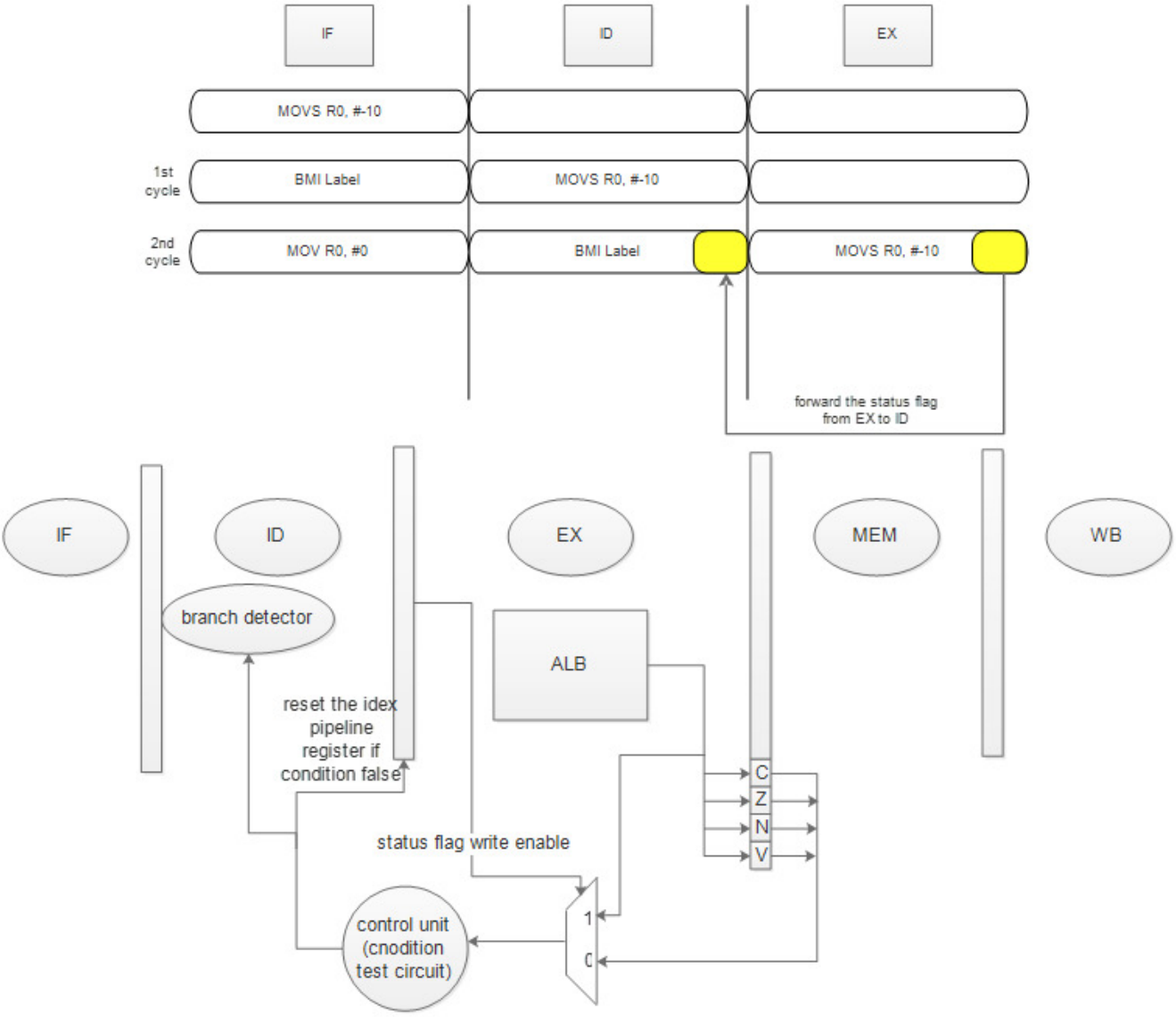


Figure 7.1: Solution for status flag problem.

7.2.1 Block diagram of udp (Data path)

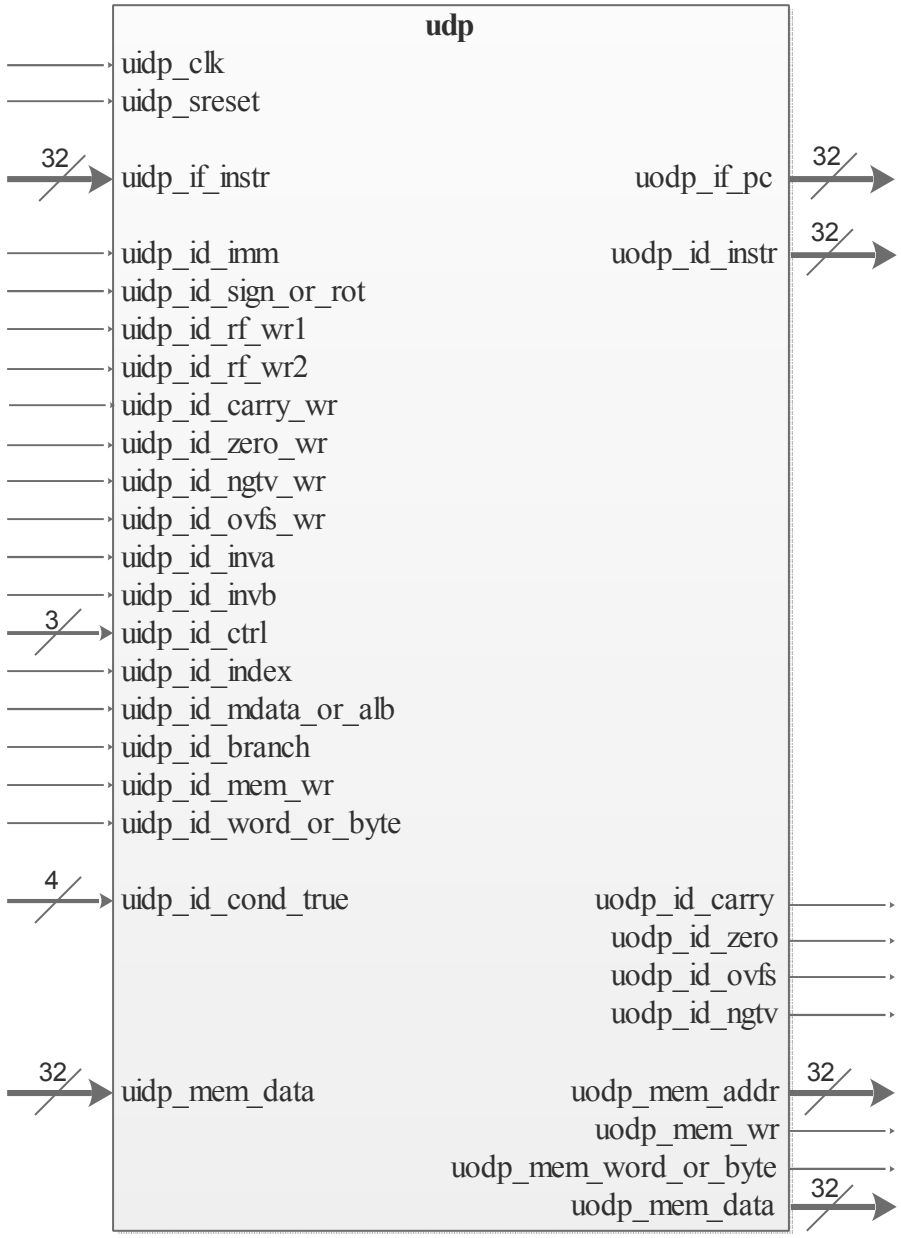


Figure 7.2: block diagram of data path

Input

Pin name : uidp_clk	Registered : No
Pin class : clock signal	
Source → Destination : external → udp	
Bit size : 1-bit	
Active : Rising edge	
Pin Function: Periodic signal for synchronize purpose.	

Pin name : uidp_sreset Registered : No Pin class : control signal Source → Destination : external → udp Bit size : 1-bit Active : Active high Pin Function: Reset the data path when active high else perform normal operation.
Pin name : uidp_if_instr Registered : No Pin class : data signal Source → Destination : u_cache → udp Bit size : 32-bit Active : - Pin Function: Instruction in text segment with uodp_if_pc as the address.
Pin name : uidp_id_imm Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high when the operand 2 is an immediate else active low 1'b 0 : non-immediate operand 1'b 1: immediate operand
Pin name : uidp_id_sign_or_rot Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high when operand 2 will undergoes sign extension else active low for rotation extension 1'b 0 : rotation extension operand 2 1'b 1: sign extension operand 2
Pin name : uidp_id_rf_wr1 Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable 1st write port else active low 1'b 0 : hold the data 1'b 1: write the data to 1st write address
Pin name : uidp_id_rf_wr2 Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable 2nd write port else active low 1'b 0 : hold the data 1'b 1: write the data to 2nd write address
Pin name : uidp_id_carry_wr Registered : Yes

Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update carry flag else active low 1'b 0 : hold previous carry flag 1'b 1: update carry flag
Pin name : uidp_id_zero_wr Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update zero flag else active low 1'b 0 : hold previous zero flag 1'b 1: update zero flag
Pin name : uidp_id_ngtv_wr Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update negative flag else active low 1'b 0 : hold previous negative flag 1'b 1: update negative flag
Pin name : uidp_id_ovfs_wr Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update overflow flag else active low 1'b 0 : hold previous overflow flag 1'b 1: update overflow flag
Pin name : uidp_id_branch Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to change PC to branch target address else active low for normal increment of PC (+4) 1'b 0 : branch to target address 1'b 1: normal +4 increment of PC
Pin name : uidp_id_cond_true Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high if the condition meet else active low to skip the instruction

1'b 0 : skip the instruction 1'b 1: execute the instruction
Pin name : uidp_id_inva Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to invert 1st operand (from Rn) else active low to use original operand 1'b 0 : use original data from Rn for ALB 1'b 1: invert the data from Rn before going through ALB
Pin name : uidp_id_invb Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to invert 2nd operand (from Rm or immediate) else active low to use original operand 1'b 0 : use original data from Rm or immediate for ALB 1'b 1: invert the data from Rm or immediate before going through ALB
Pin name : uidp_id_ctrl Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 3-bit Active : - Pin Function: opcode for the ALB. 3'b 000 : addition 3'b 001: addition with carry 3'b 010: subtraction 3'b 011: subtraction with carry 3'b 100: and AND 3'b 101: or OR 3'b 110: exclusive or XOR 3'b 111: by pass operand b (from Rm)
Pin name : uidp_id_index Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: decide the address mode for memory read and load 1'b 0 : Post-index 1'b 1 : Pre-index
Pin name : uidp_id_word_or_byte Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit

Active : Active high Pin Function: Active high for load or save a byte of data else active low for a word of data 1'b 0 : word 1'b 1: byte
Pin name : uidp_id_mem_wr Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to update memory else active low to hold the previous memory data. 1'b 0 : hold previous memory data 1'b 1: update memory data
Pin name : uidp_id_mdata_or_alb Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high for write data from memory to register file else active low for write data from ALB to register file. 1'b 0 : use data from ALB 1'b 1: use data from memory cache
Pin name : uidp_if_instr Registered : No Pin class : data signal Source → Destination : u_cache → udp Bit size : 32-bit Active : - Pin Function: data in data segment with uodp_mem_addr as the address.

Table 7.2: Input pins description for data path unit

Output

Pin name : uodp_if_pc Registered : Yes Pin class : address signal Source → Destination : udp → ucache Bit size : 32-bit Active : - Pin Function: Address for instruction.
Pin name : uodp_id_instr Registered : Yes Pin class : data signal Source → Destination : udp → ucp Bit size : 32-bit Active : - Pin Function: Instruction content.
Pin name : uodp_id_carry Registered : Yes Pin class : address signal

Source → Destination : udp → ucp Bit size : 1-bit Active : Active high Pin Function: Carry flag. 1'b0 : not carry produce 1'b1 : carry produce
Pin name : uodp_id_zero Registered : Yes Pin class : address signal Source → Destination : udp → ucp Bit size : 1-bit Active : Active high Pin Function: Zero flag. 1'b0 : result is non-zero 1'b1 : result is zero
Pin name : uodp_id_ovfs Registered : Yes Pin class : address signal Source → Destination : udp → ucp Bit size : 1-bit Active : Active high Pin Function: Overflow flag. 1'b0 : no overflow 1'b1 : overflow
Pin name : uodp_id_ngtv Registered : Yes Pin class : address signal Source → Destination : udp → ucp Bit size : 1-bit Active : Active high Pin Function: Negative flag. 1'b0 : positive 1'b1 : negative
Pin name : uodp_mem_addr Registered : Yes Pin class : address signal Source → Destination : udp → ucache Bit size : 32-bit Active : - Pin Function: memory address of data in main memory.
Pin name : uodp_mem_wr Registered : Yes Pin class : control signal Source → Destination : udp → ucache Bit size : 1-bit Active : Active high Pin Function: write the uodp_mem_data to main memory with address uodp_mem_addr if the signal is active high else hold the data in the main memory.
Pin name : uodp_mem_word_or_byte Registered : Yes Pin class : control signal

Source → Destination : udp → ucache Bit size : 1-bit Active : Active high Pin Function: 1'b0 : write or read a word of data 1;b1 : write or read a byte of data	
Pin name : uodp_mem_data Pin class : data signal Source → Destination : udp → ucache Bit size : 32-bit Active : - Pin Function: data to write to main memory.	Registered : Yes

Table 7.3: output pins description for data path unit

7.2.2 Data path block level hierarchy

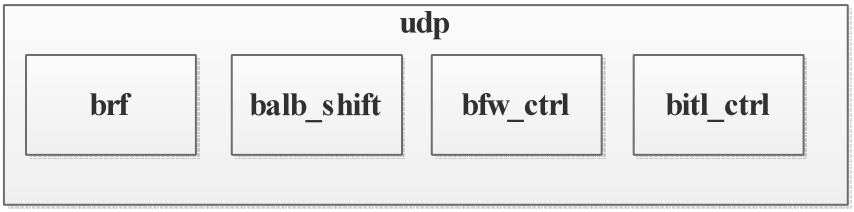


Figure 7.3: partition of data path unit

The data path unit builds up with

- Register file (brf)
- Arithmetic logic block with shift (balb_shift)
- Data forwarding control (bf_ctrl)
- Interlock control (bitl_ctrl)

7.3 Register file (brf)

7.3.1 Functionality

A set of 32-bits register bank with number of 16 registers.
Function of the registers:

Name	Use
R0	Argument/ return value/ temporary variable
R1-R3	Argument/ temporary variable
R4-R11	Saved variables
R12	Temporary variable
R13 (SP)	Stack pointer
R14 (LR)	Link register (return address)
R15 (PC)	Program counter

Table 7.4: General register

7.3.2 Block Diagram

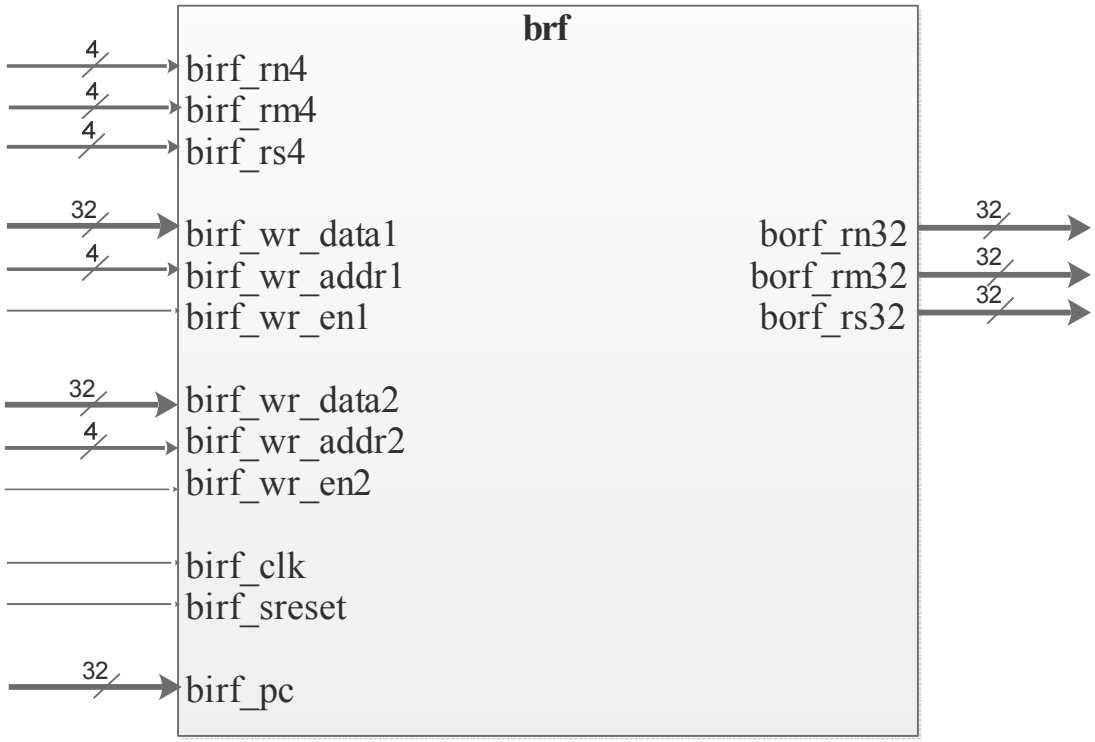


Figure 7.5: block diagram of brf (register file)

Input

Pin name : birf_rn4 Pin class : address signal Source → Destination : udp → brf Bit size : 4-bit Active : - Pin Function: Address for Rn register.	Registered : No
Pin name : birf_rm4 Pin class : address signal Source → Destination : udp → brf Bit size : 4-bit Active : - Pin Function: Address for Rm register.	Registered : No
Pin name : birf_rs4 Pin class : address signal Source → Destination : udp → brf Bit size : 4-bit Active : - Pin Function: Address for Rs register.	Registered : No
Pin name : birf_wr_data1 Pin class : data signal Source → Destination : udp → brf Bit size : 32-bit Active : - Pin Function: Data to write in specific register with 1st write port.	Registered : No
Pin name : birf_wr_addr1 Pin class : address signal Source → Destination : udp → brf Bit size : 4-bit Active : - Pin Function: Address for register where the data should write to (for 1st write port).	Registered : No
Pin name : birf_wr_en1 Pin class : control signal Source → Destination : udp → brf Bit size : 1-bit Active : High Pin Function: Update the data of the register when active high (for 1st write port).	Registered : No
Pin name : birf_wr_data2 Pin class : data signal Source → Destination : udp → brf Bit size : 32-bit Active : - Pin Function: Data to write in specific register with 2nd write port.	Registered : No
Pin name : birf_wr_addr2 Pin class : address signal Source → Destination : udp → brf	Registered : No

Bit size : 4-bit Active : - Pin Function: Address for register where the data should write to (for 2nd write port).
Pin name : birf_wr_en2 Registered : No Pin class : control signal Source → Destination : udp → brf Bit size : 1-bit Active : High Pin Function: Update the data of the register when active high (for 2nd write port).
Pin name : birf_pc Registered : No Pin class : data signal Source → Destination : udp → brf Bit size : 32-bit Active : - Pin Function: Current PC value.

Table 7.5: input pins description of brf

Output

Pin name : borf_rn32 Registered : No Pin class : data signal Source → Destination : brf → udp Bit size : 32-bit Active : - Pin Function: Data from Rn register.
Pin name : borf_rm32 Registered : No Pin class : data signal Source → Destination : brf → udp Bit size : 32-bit Active : - Pin Function: Data from Rn register.
Pin name : borf_rs32 Registered : No Pin class : data signal Source → Destination : brf → udp Bit size : 32-bit Active : - Pin Function: Data from Rs register.

Table 7.6: output pins description of brf

7.3.3 Functional table

birf_wr_en1	birf_wr_en2	birf_clk	Operation
1'b0	1'b0	At negative edge	Hold the previous values
1'b0	1'b1	At negative edge	Write new data to register file (2^{nd} write port data)
1'b1	1'b0	At negative edge	Write new data to register file (1^{st} write port data)
1'b0	1'b1	At negative edge	Write new data to register file (both write port data will write. If the location if same, 2^{nd} write port has higher priority.)
1'bx	1'bx	At positive edge	Read data from register file

Table 7.7: functional table for write enable signal.

Address pin	Operation
birf_rn4	Read data from register file and output with borf_rn32
birf_rm4	Read data from register file and output with borf_rm32
birf_rs4	Read data from register file and output with borf_rs32
birf_wr_addr1	Address of register which the birf_wr_data1 will write to it
birf_wr_addr2	Address of register which the birf_wr_data2 will write to it

Table 7.8: functional table for address pin.

7.3.4 Internal block diagram of Register File

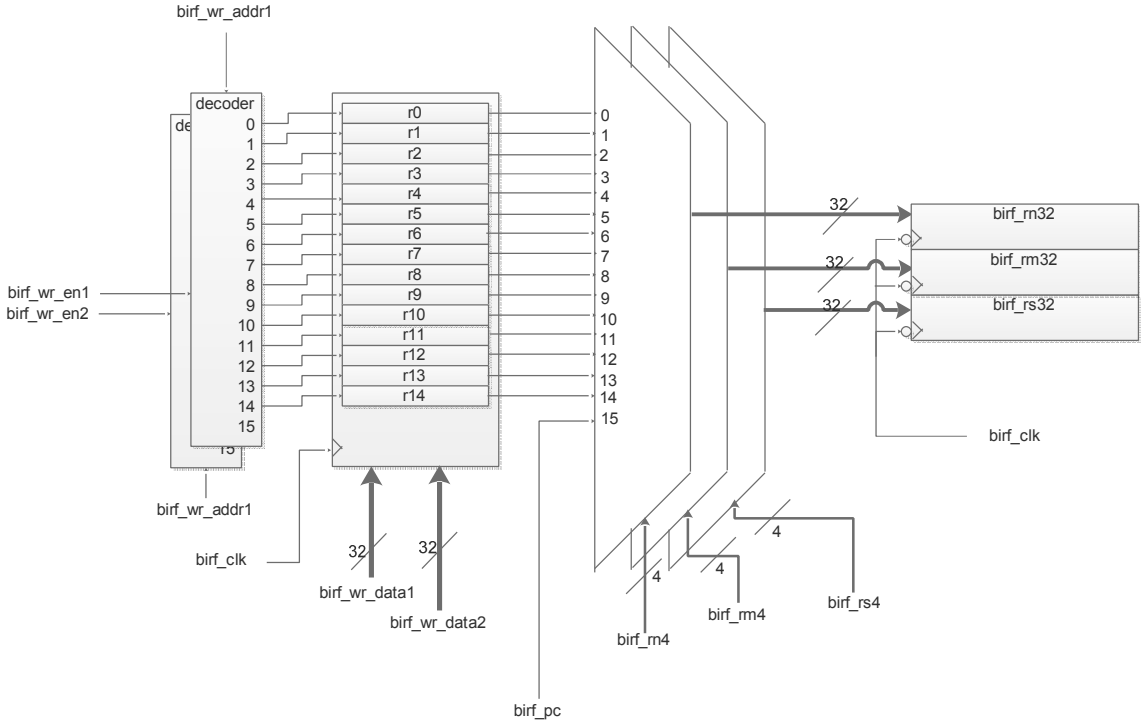


Figure 7.6: Design of register file.

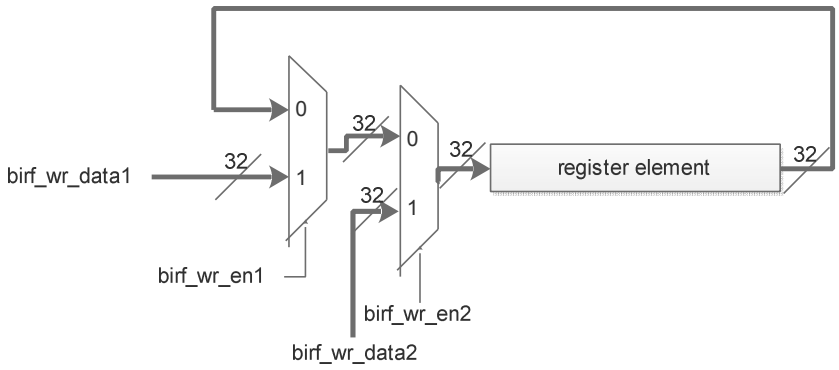


Figure 7.7: Single element of register file

7.4 Arithmetic Logic Block with shift (balb_shift)

7.4.1 Functionality

Combinations of arithmetic logic block and barrel shifter which operates on 32-bits integer operand.
Perform:

- Addition
- Subtraction
- AND (logic)
- OR (logic)
- XOR (logic)
- Logical shift left
- Logical shift right
- Arithmetic shift right
- Rotate right
- Rotate right with extend
- MOV/MVN instruction (copy value to register)

7.4.2 Block Diagram

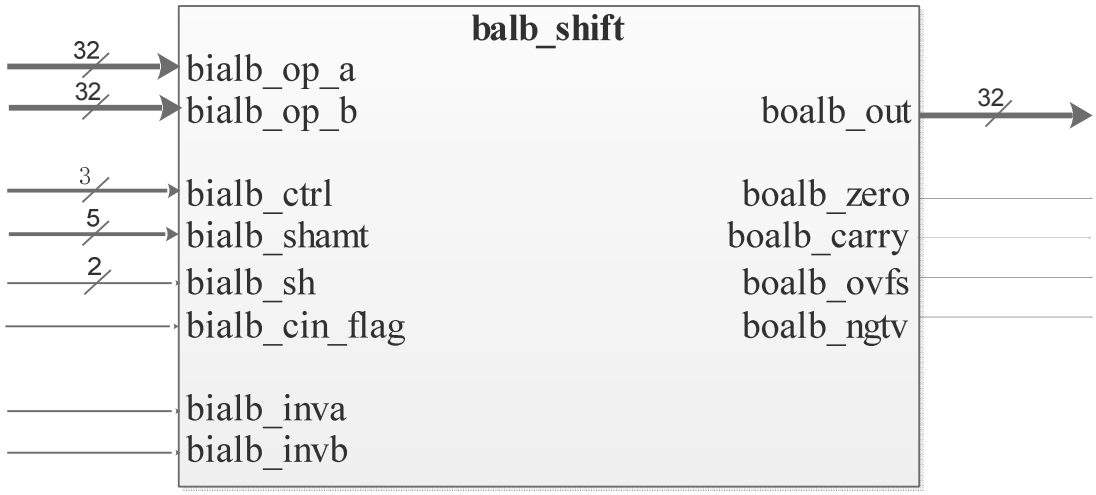


Figure 7.8: Block diagram of balb_shift (ALU and shifter)

Input

Pin name : bialb_op_a	Registered : No
Pin class : data signal	
Source → Destination : udp → balb_shift	
Bit size : 32-bit	
Active : -	
Pin Function: 1st 32-bits operand.	

Pin name : bialb_op_b Registered : No Pin class : data signal Source → Destination : udp → balb_shif Bit size : 32-bit Active : - Pin Function: 2nd 32-bits operand.
Pin name : bialb_ctrl Registered : No Pin class : control signal Source → Destination : udp → balb_shif Bit size : 3-bit Active : - Pin Function: Opcode to select operation to perform.
Pin name : bialb_shamt Registered : No Pin class : control signal Source → Destination : udp → balb_shif Bit size : 5-bit Active : - Pin Function: Number of bit need to shift or rotate.
Pin name : bialb_sh Registered : No Pin class : control signal Source → Destination : udp → balb_shif Bit size : 2-bit Active : - Pin Function: Represent shift type need to perform. 00 = LSL 01 = LSR 10 = ASR 11 = ROR, RRX
Pin name : bialb_cin_flag Registered : No Pin class : data signal Source → Destination : udp → balb_shif Bit size : 1-bit Active : - Pin Function: Current C bit in CPSR.
Pin name : bialb_inva Registered : No Pin class : control signal Source → Destination : udp → balb_shif Bit size : 1-bit Active : High Pin Function: invert value pass in to bi_alb_op_a while active high.
Pin name : bialb_invb Registered : No Pin class : control signal Source → Destination : udp → balb_shif Bit size : 1-bit Active : High Pin Function: invert value pass in to bi_alb_op_b while active high.

Table 7.9: Input pins description of balb_shift

Output

Pin name : boalb_out Pin class : data signal Source → Destination : balb_shif → udp Bit size : 32-bit Active : - Pin Function: Result from balb_shif.	Registered : Yes
Pin name : boalb_zero Pin class : data signal Source → Destination : balb_shif → udp Bit size : 1-bit Active : High Pin Function: Active high while bo_alb_out equal to 32'h 0000_0000.	Registered : Yes
Pin name : boalb_carry Pin class : data signal Source → Destination : balb_shif → udp Bit size : 1-bit Active : High Pin Function: Active high while carry out is 1'b1 when perform addition.	Registered : Yes
Pin name : boalb_ovfs Pin class : data signal Source → Destination : balb_shif → udp Bit size : 1-bit Active : High Pin Function: Active high while there is an overflow.	Registered : Yes
Pin name : boalb_ngtv Pin class : data signal Source → Destination : balb_shif → udp Bit size : 1-bit Active : High Pin Function: Active high while the bo_alb_out is a negative value.	Registered : Yes

Table 7.10: Output pins description of balb_shift

7.4.5 Test plan

Test case	Input	Expected output
1. Addition:	bi_alb_op_a = 32'h 1010_ffff bi_alb_op_b = 32'h 1111_3fed bi_alb_ctrl[2:1] = 2'b 00 bi_alb_inva = 1'b0 bi_alb_invb = 1'b0 bi_alb_cin_flag = 1'b1	
A+B	bi_alb_ctrl[0] = 1'b0	bo_alb_out = 32'h 2122_3fec
A+B+Cin	bi_alb_ctrl[0] = 1'b1	bo_alb_out = 32'h 2122_3fed
2. Subtraction:	bi_alb_op_a = 32'h f010_ffff bi_alb_op_b = 32'h 1111_3fed bi_alb_ctrl[2:1] = 2'b 01 bi_alb_cin_flag = 1'b1	
A-B	bi_alb_ctrl[0] = 1'b0 bi_alb_inva = 1'b0 bi_alb_invb = 1'b1	bo_alb_out = 32'h deff_c012 bo_alb_ngtv = 1'b1
A-B-Cin	bi_alb_ctrl[0] = 1'b1 bi_alb_inva = 1'b0 bi_alb_invb = 1'b1	bo_alb_out = 32'h deff_c012 bo_alb_ngtv = 1'b1
B-A	bi_alb_ctrl[0] = 1'b0 bi_alb_inva = 1'b1 bi_alb_invb = 1'b0	bo_alb_out = 32'h 2100 3fee bo_alb_ngtv = 1'b0
B-A-Cin	bi_alb_ctrl[0] = 1'b1 bi_alb_inva = 1'b1 bi_alb_invb = 1'b0	bo_alb_out = 32'h 2100 3fee bo_alb_ngtv = 1'b0
3. Logical operation:	bi_alb_op_a = 32'h 1010_ffff bi_alb_op_b = 32'h 1111_3fed bi_alb_ctrl[2] = 1'b 1 bi_alb_inva = 1'b0 bi_alb_invb = 1'b0	
AND	bi_alb_ctrl[1:0] = 2'b 00	bo_alb_out = 32'h 1010_3fed
OR	bi_alb_ctrl[1:0] = 2'b 01	bo_alb_out = 32'h 1111_ffff
XOR	bi_alb_ctrl[1:0] = 2'b 10	bo_alb_out = 32'h 0101_c012
4. Move operation:	bi_alb_op_b = 32'h 1010_ffff bi_alb_ctrl[2:0] = 3'b 111	

MOV	bi_alb_invb = 1'b 0	bo_alb_out = 32'h 1010_ffff
MVN	bi_alb_invb = 1'b 1	bo_alb_out = 32'h efef_0000
5. Shift/ rotate:	bi_alb_op_a = 32'h f010_ffff bi_alb_op_b = 32'h f010_ffff bi_alb_inva = 1'b0 bi_alb_cin_flag = 1'b1 bi_alb_shamt = 5'b 00100 bi_alb_ctrl[2:0] = 3'b 111 bi_alb_invb = 1'b 0	
LSL	bi_alb_sh[1:0] = 2'b 00	bo_alb_out = 32'h 010f_fff0
LSR	bi_alb_sh[1:0] = 2'b 01	bo_alb_out = 32'h 0f01_0fff
ASR	bi_alb_sh[1:0] = 2'b 10	bo_alb_out = 32'h ff01_0fff
ROR	bi_alb_sh[1:0] = 2'b 11	bo_alb_out = 32'h ff01_0fff

Table 7.13: test plan for balb_shift

7.4.6 Simulation result

Addition:

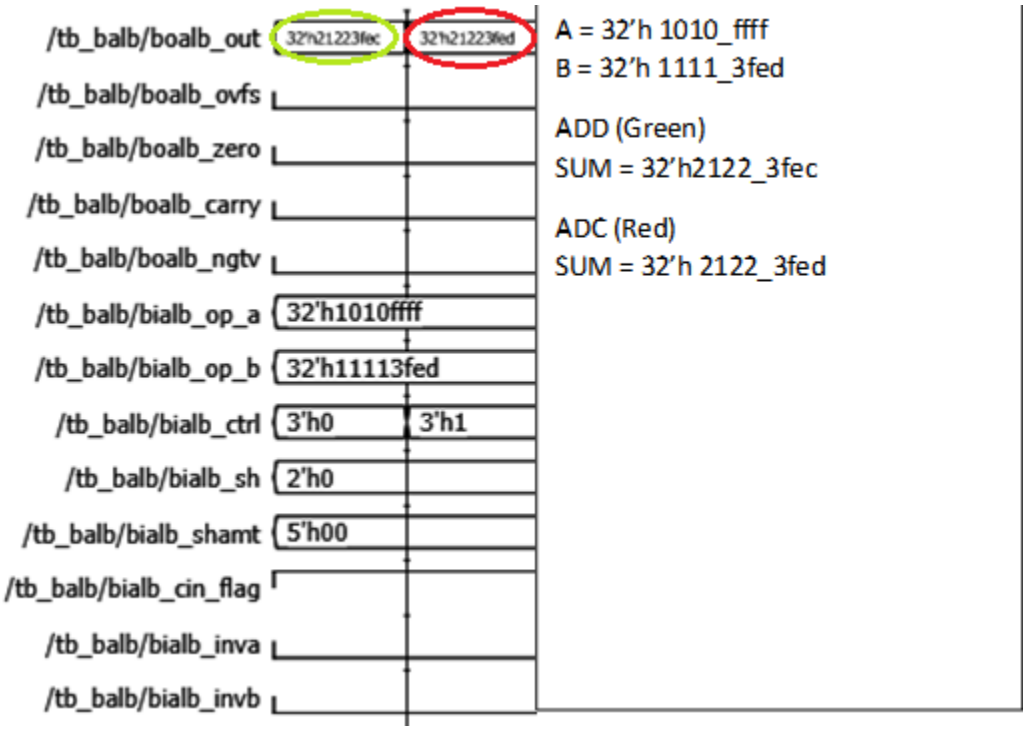


Figure 7.11: simulation result (1) - addition

Subtraction:

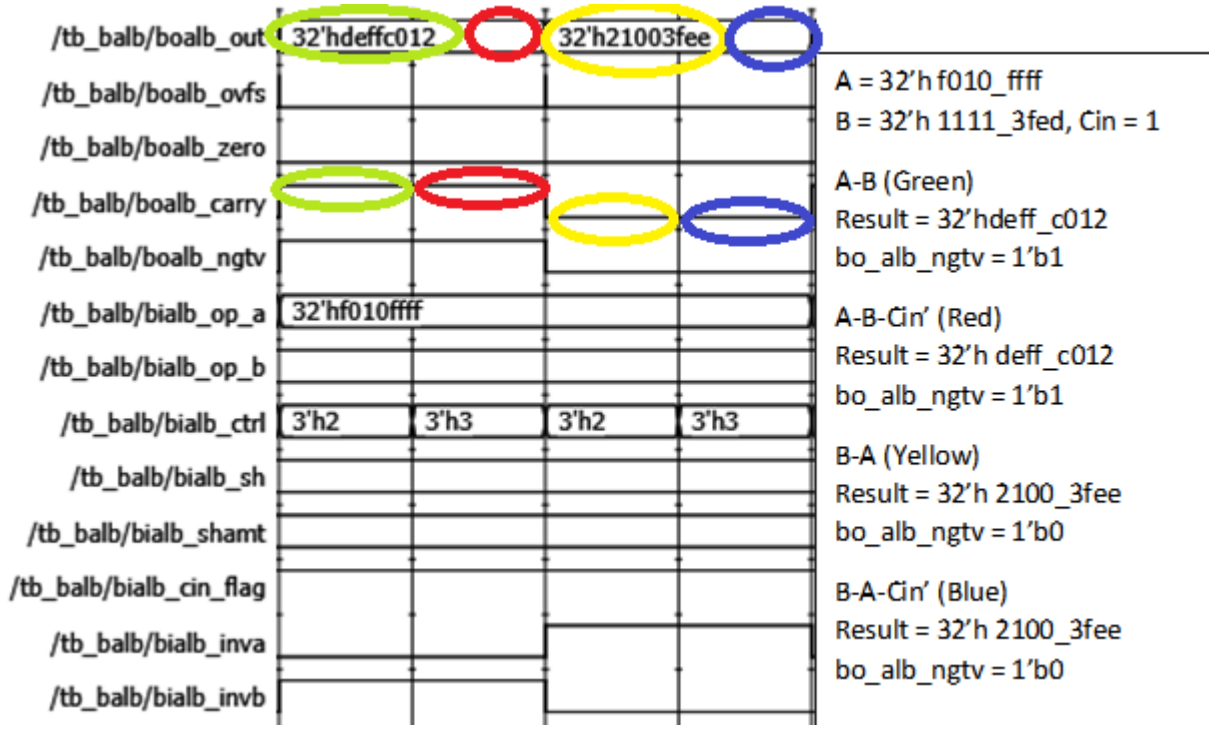


Figure 7.12: simulation result (2) - subtraction

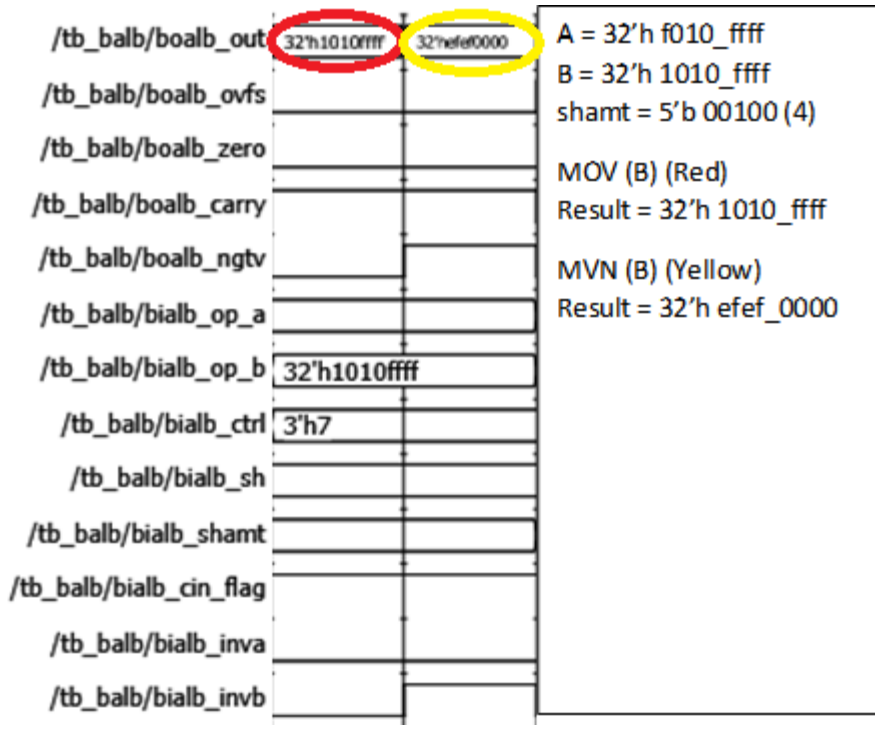


Figure 7.13: simulation result (3) - subtraction

Logical operation:

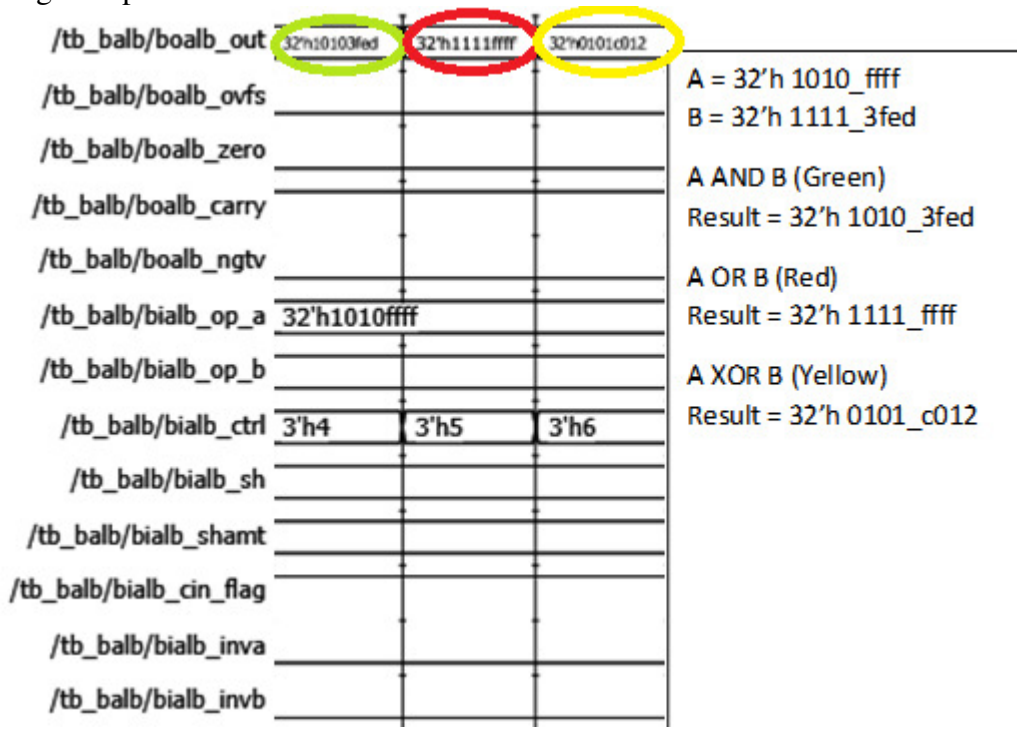


Figure 7.14: simulation result (4) - logical

Shift/ rotate:

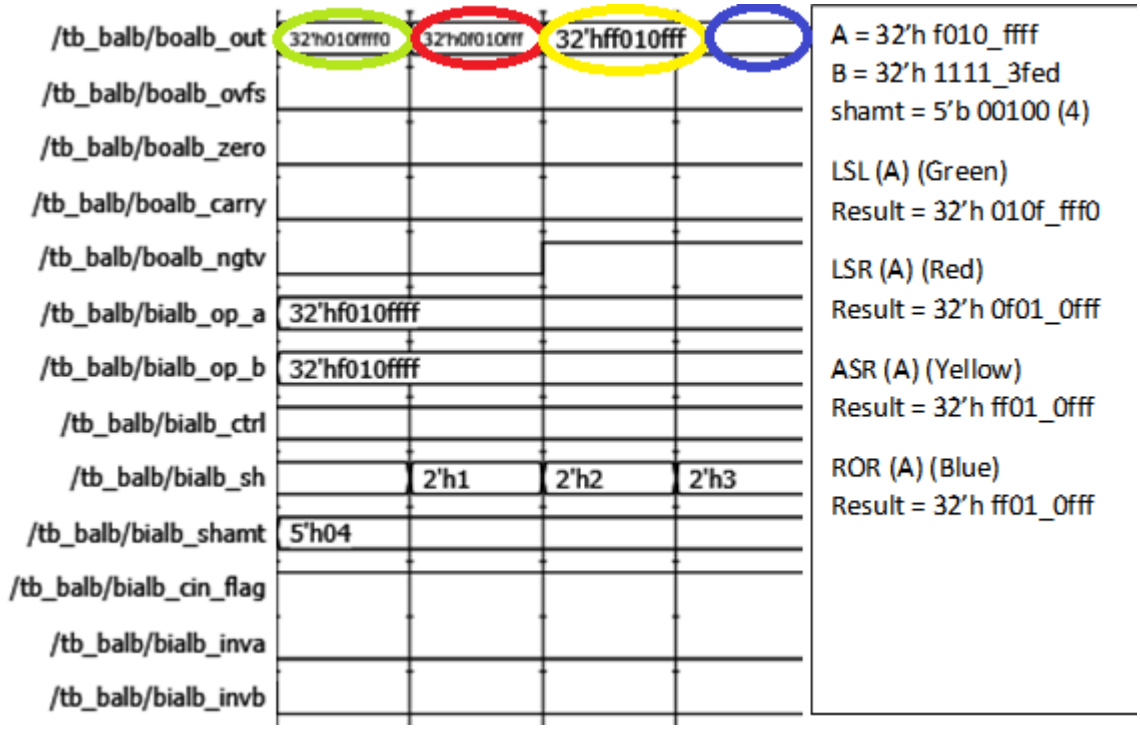


Figure 7.15: simulation result (5) – shift/ rotate

7.5 Data forwarding control (bfw_ctrl)

7.5.1 Functionality

The Forwarding Block is responsible for detecting data dependency problem. When an instruction write to the register destination and the following instruction read from the previous instruction's register destination, data dependency occur, when it occur and then forward the proper data from the corresponding stage to the EX Stage so that the data which goes into ALU is the correct value.

Rn forwarding

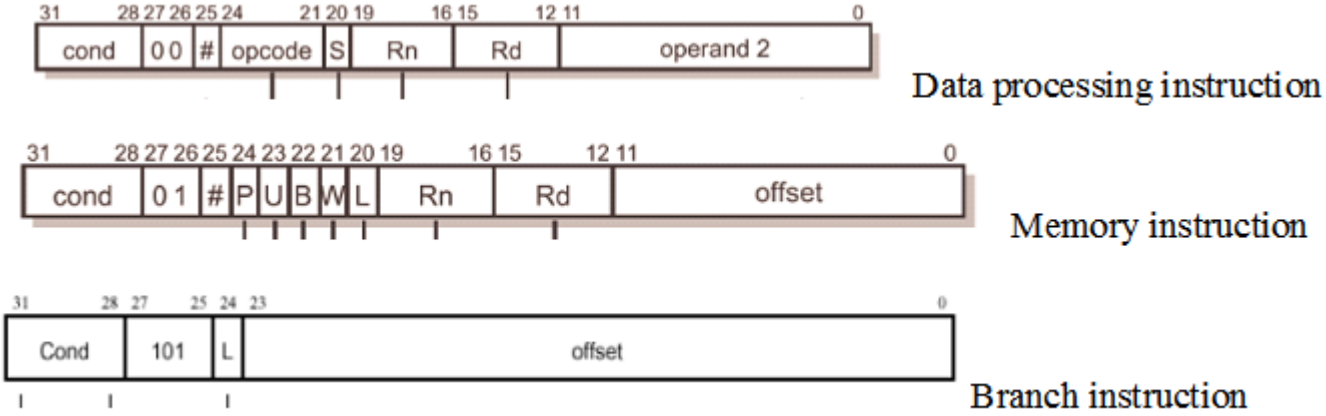


Figure 7.16: Instruction format

From the instruction format,

- The field of Rn's address (19:16 bits) is fixed and always used in both data-processing and memory instruction.
- The branch instruction used 24 bits (23:0) to store the offset mean that 19:16 bits of instruction is not represent as Rn's address, in this case, multiplexer in data path will pass PC and offset to ALU, so Rn value will no affect the result.

Rm and Rs forwarding

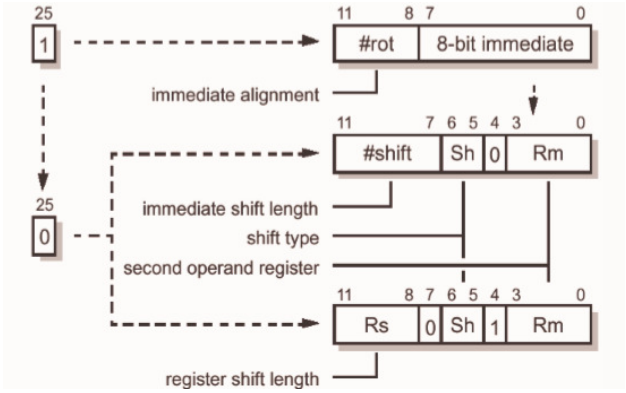


Figure 7.17: Data processing instruction

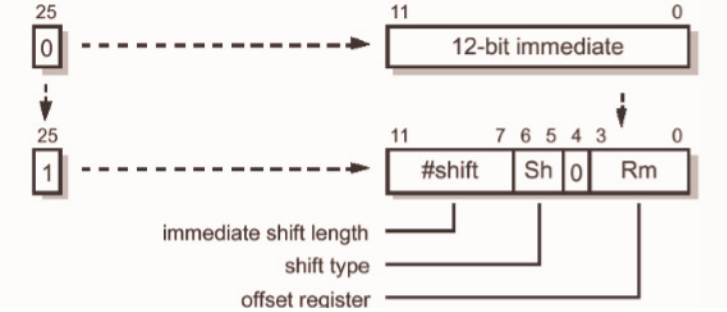


Figure 7.18: Memory instruction

- Rm only used when there is non-immediate instruction.
- Rm's address field located at 3:0 bits of both instructions.

However,

- Rs only appear in Data processing instruction with condition that non-immediate and bit 4 is set (1'b1).
- Rs's address field located at bit11:8 of the data-processing instruction

Type	Type description	Instruction	Assembly format
Data processing	Arithmetic (signed or unsigned) or logical operations between two registers, `$rn` and `Opd2` which can be `immediate` value or register value `$rm`, then store the results into register `$rd`.	`Add`	`add $rd, $rn, Opd2`
		`Adc`	`adc $rd, $rn, Opd2`
		`Sub`	`sub $rd, $rn, Opd2`
		`Sbc`	`Sbc $rd, $rn, Opd2`
		`Srb`	`srb $rd, $rn, Opd2`
		`Src`	`src $rd, $rn, Opd2`
		`Mov`	`mov $rd, Opd2`
		`Mvn`	`mvn $rd, Opd2`
		`Nop`	`nop` (it is equivalent to `mov $0, $0`)
		`Orr`	`orr $rd, $rn, Opd2`
		`And`	`and $rd, $rn, Opd2`
		`Eor`	`eor $rd, $rn, Opd2`
		`Bic`	`bic $rd, $rn, Opd2`
		`Tst`	`tst $rn, Opd2`
		`Teq`	`teq $rn, Opd2`
		`Cmp`	`cmp $rn, Opd2`
		`Cmn`	`cmn $rn, Opd2`
		`Lsl`	`mov $rd, Opd2,LSL shamt`
		`Lsr`	`mov $rd, Opd2,LSR shamt`
		`Asr`	`mov $rd, Opd2,ASR shamt`
		`Ror`	`mov $rd, Opd2,ROR shamt`
		`Rrx`	`mov $rd, Opd2,RRX shamt`

Load	Instructions that are loading a data from the Data Cache into register `$rd`	`ldr`
		`post idx`
		`offset`
		`pre idx`
			
			`ldr $rd, [$rn], $rm`
			`ldr $rd, [$rn, $rm]`
			`ldr $rd, [$rn, $rm]!`

Store	Instructions that are storing a data storing in register `$rd` into the Data Cache	`srt`
		`post idx`
		`offset`
		`pre idx`
			
			`str $rd, [$rn], $rm`
			`str $rd, [$rn, $rm]`
			`str $rd, [$rn, $rm]!`

Branch	Instructions that will jump to the specified location of program if the condition is fulfilled	`B`	`b label`
		`Bl`	`bl label`

Table 7.14: ARM assembly instruction

Among the instructions, instructions that will update the Register File (RF) are the

1. Data processing (except tst, teq, cmp, cmn),
2. Load

These instructions might cause data hazards to the later instructions. When data dependencies happen, forwarding or stalling is needed to solve them. These instructions can be further categorised based on the stages they get their results, since the principle of forwarding is to provide data to the data depending instructions once the data is ready, to ease the design of forwarding circuitry.

1. Results is ready in EX stage
   - Data processing
2. Results is ready in MEM stage
   - Load

7.5.2 Forwarding Block Function Tables

Forward Rn, Rm, Rs

No.	Input									Output	Source
	ID	EX				MEM					
		Reg Write		DestReg		Reg Write		DestReg			
	Reg Rn	Wr_1	Wr_2	Rd_1	Rd_2	Wr_1	Wr_2	Rd_1	Rd_2		
1.	A	0	0	X	X	0	0	X	X	00	ID
2.	A	1	0	B	X	1	0	C	X	00	ID
3.	A	1	0	A	X	0	0	X	X	01	EX
4.	A	X	1	X	A	0	0	X	X	01	EX
5.	A	0	0	X	X	1	0	A	X	10	MEM.Rd1
6.	A	0	0	X	X	0	1	X	A	11	MEM.Rd2
7.	A	0	0	X	X	1	1	A	A	11	MEM.Rd2
8.	A	0	1	X	A	0	1	X	A	01	EX

Table 7.15: functional table for forwarding block

Explanations:

1. The value from register file itself is used as Rn, Rm, and Rs (value from ID stage) since there is not overwrite value in EX and MEM stages.
2. The value from register file itself is used as Rn, Rm, and Rs (value from ID stage) since there the Register destination to be update in EX and MEM stages are B and C respectively which is not related to register A will read in ID stage.
3. The value from ALU output (EX satge) is used as Rn, Rm, and Rs, since there is a write enable and address of register destination in EX stage same with address of register to be read in ID stage.
4. This case is similar to case no. 3.
5. The value from data memory (MEM stage) is used as Rn, Rm, and Rs, since there is a write enable (wr1) and address of register destination (rd1) in MEM stage same with address of register to be read in ID stage. (MEM.Rd1)
6. The value which ALUpassed to MEM stage is used as Rn, Rm, and Rs, since there is a write enable (wr2) and address of register destination (rd2) in MEM stage same with address of register to be read in ID stage. (MEM.Rd2)
7. When both write enable (wr2 and wr1) are asserted and register destinations (rd1 and rd2) are both same address with read register, the 2^{nd} write port had higher priority (wr2 and rd2), therefore the MEM.Rd2 is used.
8. When same address of register destination in EX and MEM are both same with read address in ID stage, value from EX will forward to ID instead of MEM because EX had latest data of the register.

7.5.3 Block diagram

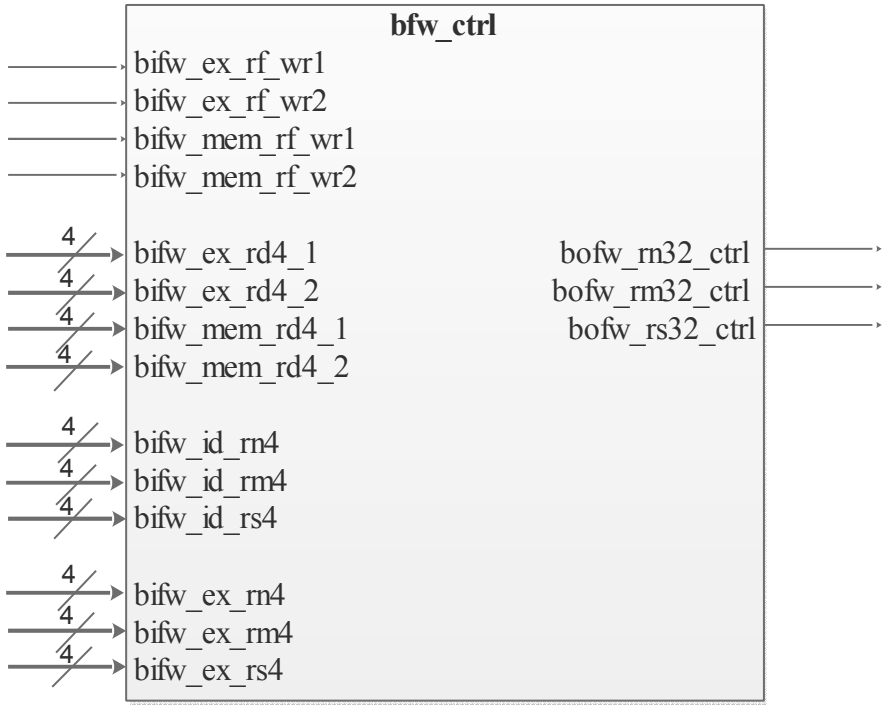


Figure 7.19: block diagram for bfw_ctrl (forwarding control)

Input

Pin name : bifw_ex_rf_wr1 Pin class : control signal Source → Destination : udp → bfw_ctrl Bit size : 1-bit Active : Active high Pin Function: 1st write port enable signal at EX stage.	Registered : No
Pin name : bifw_ex_rf_wr2 Pin class : control signal Source → Destination : udp → bfw_ctrl Bit size : 1-bit Active : Active high Pin Function: 2nd write port enable signal at EX stage.	Registered : No
Pin name : bifw_mem_rf_wr1 Pin class : control signal Source → Destination : udp → bfw_ctrl Bit size : 1-bit Active : Active high Pin Function: 1st write port enable signal at MEM stage.	Registered : No
Pin name : bifw_mem_rf_wr2 Pin class : control signal Source → Destination : udp → bfw_ctrl Bit size : 1-bit	Registered : No

Active : Active high Pin Function: 2nd write port enable signal at MEM stage.
Pin name : bifw_ex_rd4_1 Registered : No Pin class : data signal Source → Destination : udp → bfw_ctrl Bit size : 4-bit Active : - Pin Function: Address for 1st write port Rd (destination register) at EX stage.
Pin name : bifw_ex_rd4_2 Registered : No Pin class : data signal Source → Destination : udp → bfw_ctrl Bit size : 4-bit Active : - Pin Function: Address for 2nd write port Rd (destination register) at EX stage.
Pin name : bifw_mem_rd4_1 Registered : No Pin class : data signal Source → Destination : udp → bfw_ctrl Bit size : 4-bit Active : - Pin Function: Address for 1st write port Rd (destination register) at MEM stage.
Pin name : bifw_mem_rd4_2 Registered : No Pin class : data signal Source → Destination : udp → bfw_ctrl Bit size : 4-bit Active : - Pin Function: Address for 2nd write port Rd (destination register) at MEM stage.
Pin name : bifw_id_rn4 Registered : No Pin class : data signal Source → Destination : udp → bfw_ctrl Bit size : 4-bit Active : - Pin Function: Address for Rn register at ID stage.
Pin name : bifw_id_rm4 Registered : No Pin class : data signal Source → Destination : udp → bfw_ctrl Bit size : 4-bit Active : - Pin Function: Address for Rm register at ID stage.
Pin name : bifw_id_rs4 Registered : No Pin class : data signal Source → Destination : udp → bfw_ctrl Bit size : 4-bit Active : - Pin Function: Address for Rs register at ID stage.
Pin name : bifw_ex_rn4 Registered : No Pin class : data signal

Source → Destination : udp → bfw_ctrl Bit size : 4-bit Active : - Pin Function: Address for Rn register at EX stage.
Pin name : bifw_ex_rm4 Registered : No Pin class : data signal Source → Destination : udp → bfw_ctrl Bit size : 4-bit Active : - Pin Function: Address for Rm register at EX stage.
Pin name : bifw_ex_rs4 Registered : No Pin class : data signal Source → Destination : udp → bfw_ctrl Bit size : 4-bit Active : - Pin Function: Address for Rs register at EX stage.

Table 7.16: input pins description of bfw_ctrl

Output

Pin name : bofw_rn32_ctrl Registered : Yes Pin class : control signal Source → Destination : bfw_ctrl → udp Bit size : 2-bit Active : - Pin Function: Control signal that decide whether there is a forwarding for Rn register or not.
Pin name : bofw_rm32_ctrl Registered : Yes Pin class : control signal Source → Destination : bfw_ctrl → udp Bit size : 2-bit Active : - Pin Function: Control signal that decide whether there is forwarding for Rm register or not.
Pin name : bofw_rs32_ctrl Registered : Yes Pin class : control signal Source → Destination : bfw_ctrl → udp Bit size : 2-bit Active : - Pin Function: Control signal that decide whether there is forwarding for Rs register or not.

Table 7.17: output pins description of bfw_ctrl

7.6 Interlock control (bitl_ctrl)

7.6.1 Functionality

To overcome the problem that data from memory is not ready yet for next instruction.

E.g. LDR R0, [R1] @load value to R0, EX
MOV R2, R0 @copy R0 to R2, ID
R0 is not ready for R2 since it only reaches EX stage.

7.6.2 Block diagram

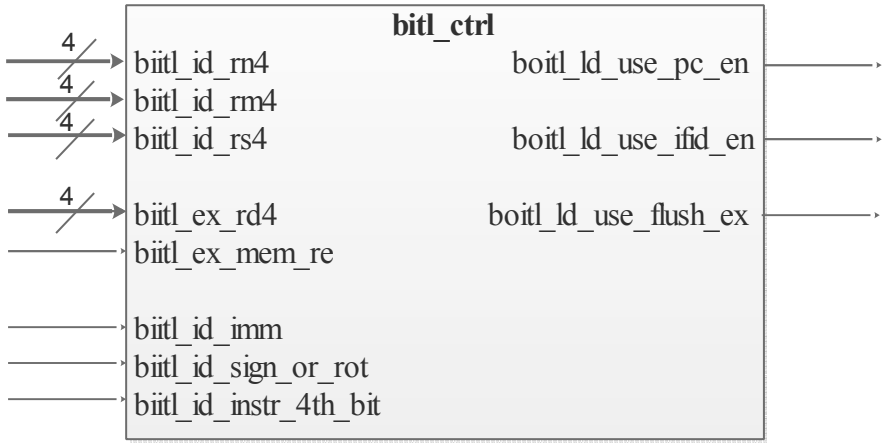


Figure 7.20: block diagram of bitl_ctrl (interlock control)

Input

Pin name : biitl_id_rn4 Pin class : data signal Source → Destination : udp → bitl_ctrl Bit size : 4-bit Active : - Pin Function: Address for Rn register in ID stage.	Registered : No
Pin name : biitl_id_rm4 Pin class : data signal Source → Destination : udp → bitl_ctrl Bit size : 4-bit Active : - Pin Function: Address for Rm register in ID stage.	Registered : No
Pin name : biitl_id_rs4 Pin class : data signal Source → Destination : udp → bitl_ctrl Bit size : 4-bit Active : - Pin Function: Address for Rs register in ID stage.	Registered : No

Pin name : biitl_ex_rd4 Registered : No Pin class : data signal Source → Destination : udp → bitl_ctrl Bit size : 4-bit Active : - Pin Function: Address for Rd (destination register) register in EX stage.
Pin name : biitl_ex_mem_re Registered : No Pin class : control signal Source → Destination : udp → bitl_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high when there will be a data read from data memory else active low.
Pin name : biitl_id_imm Registered : No Pin class : control signal Source → Destination : udp → bitl_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high if the 2nd operand is an immediate value else active low.
Pin name : biitl_id_sign_or_rot Registered : No Pin class : control signal Source → Destination : udp → bitl_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high if the immediate will be sign extend to form a 32-bits data else active low where the immediate undergo rotation extension.
Pin name : biitl_id_instr_4th_bit Registered : No Pin class : control signal Source → Destination : udp → bitl_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high when shift amount stored in Rs register else active low for immediate shift amount

Table 7.18: input pins description of bitl_ctrl

Output

Pin name : boitl_ld_use_pc_en Registered : Yes Pin class : control signal Source → Destination : bitl_ctrl → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable pc else active low to hold the pc value.
Pin name : boitl_ld_use_ifid_en Registered : Yes Pin class : control signal Source → Destination : bitl_ctrl → udp Bit size : 1-bit

Active : Active high Pin Function: Active high to enable ifid pipeline register else active low to hold the ifid pipeline register value.
Pin name : boitl_ld_use_flush_ex Registered : Yes Pin class : control signal Source → Destination : bitl_ctrl → udp Bit size : 1-bit Active : Active high Pin Function: Active high to flush idex pipeline register else active low for normal operation.

Table 7.19: output pins description of bitl_ctrl

7.6.3 Functional table

ex_rd	ex_mem_re	id_imm	id_sign_or_rot	id_instr_4th_bit	Lock
=id_rn	1'b1	X	X	X	Enable
=id_rm	1'b1	1'b 0	X	X	Enable
=id_rs	1'b1	1'b 0	1'b 0	1'b1	Enable
!=id_rn !=id_rm !=id_rs	X	X	X	X	Disable

Lock	boitl_ld_use_pc_en	boitl_ld_use_ifid_en	boitl_ld_use_flush_ex
Enable	1'b 0	1'b 0	1'b 1
Disable	1'b 1	1'b 1	1'b 0

Table 7.20: functional table of bitl_ctrl

Chapter 8 – Control Path of CRISC (Unit & Block level)

8.1 Control Path unit (ucp)

8.1.1 Functionality

Generate several control signal based on the instruction passed in. The output is stated in internal operation section (8.1.4).

8.1.2 Control Path's Unit interface – (Block diagram)

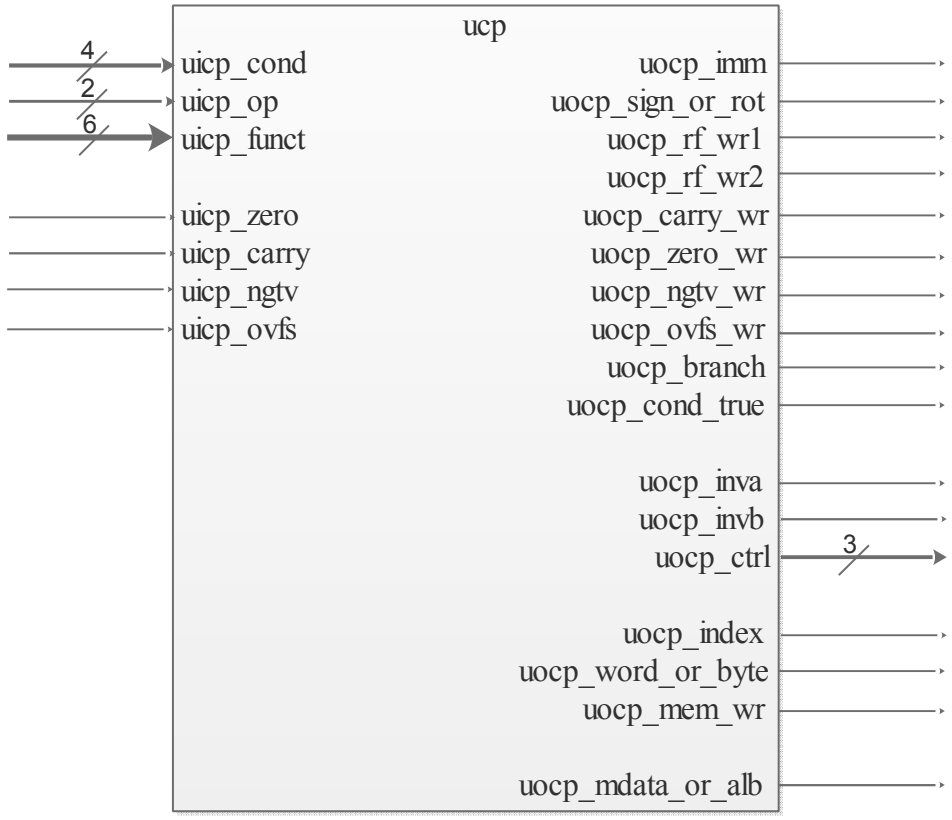


Figure 8.1: block diagram of control path

Input

Pin name : uicp_cond Pin class : data signal Source → Destination : udp → ucp Bit size : 4-bit Active : - Pin Function: mask that represent different condition where the instruction should execute.	Registered : No
Pin name : uicp_op Pin class : data signal	Registered : No

Source → Destination : udp → ucp Bit size : 2-bit Active : - Pin Function: represent instruction type. 2'b 00: Data-processing 2'b 01: Memory 2'b 10: Program flow
Pin name : uicp_funct Registered : No Pin class : data signal Source → Destination : udp → ucp Bit size : 6-bit Active : - Pin Function: Carry the information of instruction for each instruction type. Such as operand 2 is an immediate, operation to be carry out and etc.
Pin name : uicp_zero Registered : No Pin class : data signal Source → Destination : udp → ucp Bit size : 1-bit Active : Active high Pin Function: Latest zero flag that base on instruction that executing or executed in EX stage. 1'b 0: the result is non-zero. 1'b 1: the result is zero.
Pin name : uicp_carry Registered : No Pin class : data signal Source → Destination : udp → ucp Bit size : 1-bit Active : Active high Pin Function: Latest carry out flag that base on instruction that executing or executed in EX stage. 1'b 0: no carry is produced. 1'b 1: carry is produced.
Pin name : uicp_ngtv Registered : No Pin class : data signal Source → Destination : udp → ucp Bit size : 1-bit Active : Active high Pin Function: Latest negative flag that base on instruction that executing or executed in EX stage. 1'b 0: the result's MSB is 1'b0. 1'b 1: the result's MSB is 1'b1.
Pin name : uicp_ovfs Registered : No Pin class : data signal Source → Destination : udp → ucp Bit size : 1-bit Active : Active high

Pin Function: Latest overflow flag that base on instruction that executing or executed in EX stage. 1'b 0: no overflow occurs. 1'b 1: overflow occurs.

Table 8.1: Input pins description of ucp

Output

Pin name : uocp_imm Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high when the operand 2 is an immediate else active low 1'b 0 : non-immediate operand 1'b 1: immediate operand
Pin name : uocp_sign_or_rot Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high when operand 2 will undergoes sign extension else active low for rotation extension 1'b 0 : rotation extension operand 2 1'b 1: sign extension operand 2
Pin name : uocp_rf_wr1 Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable 1st write port else active low 1'b 0 : hold the data 1'b 1: write the data to 1st write address
Pin name : uocp_rf_wr2 Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable 2nd write port else active low 1'b 0 : hold the data 1'b 1: write the data to 2nd write address
Pin name : uocp_carry_wr Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update carry flag else active low 1'b 0 : hold previous carry flag

1'b 1: update carry flag
Pin name : uocp_zero_wr Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update zero flag else active low 1'b 0 : hold previous zero flag 1'b 1: update zero flag
Pin name : uocp_ngtv_wr Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update negative flag else active low 1'b 0 : hold previous negative flag 1'b 1: update negative flag
Pin name : uocp_ovfs_wr Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update overflow flag else active low 1'b 0 : hold previous overflow flag 1'b 1: update overflow flag
Pin name : uocp_branch Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to change PC to branch target address else active low for normal increment of PC (+4) 1'b 0 : branch to target address 1'b 1: normal +4 increment of PC
Pin name : uocp_cond_true Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high if the condition meet else active low to skip the instruction 1'b 0 : skip the instruction 1'b 1: execute the instruction
Pin name : uocp_inva Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit

Active : Active high Pin Function: Active high to invert 1st operand (from Rn) else active low to use original operand 1'b 0 : use original data from Rn for ALB 1'b 1: invert the data from Rn before going through ALB
Pin name : uocp_invb Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to invert 2nd operand (from Rm or immediate) else active low to use original operand 1'b 0 : use original data from Rm or immediate for ALB 1'b 1: invert the data from Rm or immediate before going through ALB
Pin name : uocp_ctrl Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 3-bit Active : - Pin Function: opcode for the ALB. 3'b 000 : addition 3'b 001: addition with carry 3'b 010: subtraction 3'b 011: subtraction with carry 3'b 100: and AND 3'b 101: or OR 3'b 110: exclusive or XOR 3'b 111: by pass operand b (from Rm)
Pin name : uocp_index Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: decide the address mode for memory read and load 1'b 0 : Post-index 1'b 1 : Pre-index
Pin name : uocp_word_or_byte Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high for load or save a byte of data else active low for a word of data 1'b 0 : word 1'b 1: byte
Pin name : uocp_mem_wr Registered : Yes

Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high to update memory else active low to hold the previous memory data. 1'b 0 : hold previous memory data 1'b 1: update memory data
Pin name : uocp_mdata_or_alb Registered : Yes Pin class : control signal Source → Destination : ucp → udp Bit size : 1-bit Active : Active high Pin Function: Active high for write data from memory to register file else active low for write data from ALB to register file. 1'b 0 : use data from ALB 1'b 1: use data from memory cache

Table 8.2: Output pins description of ucp

8.1.3 Block partitioning in ucp

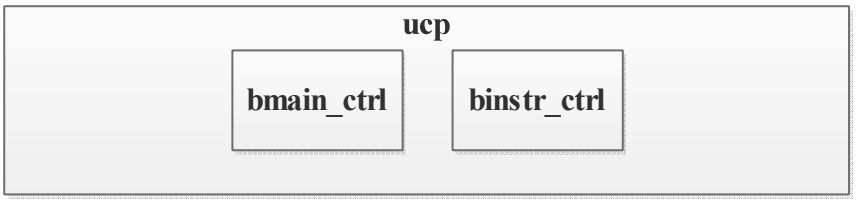


Figure 8.2: partitioning in ucp

8.1.4 Block level partition diagram

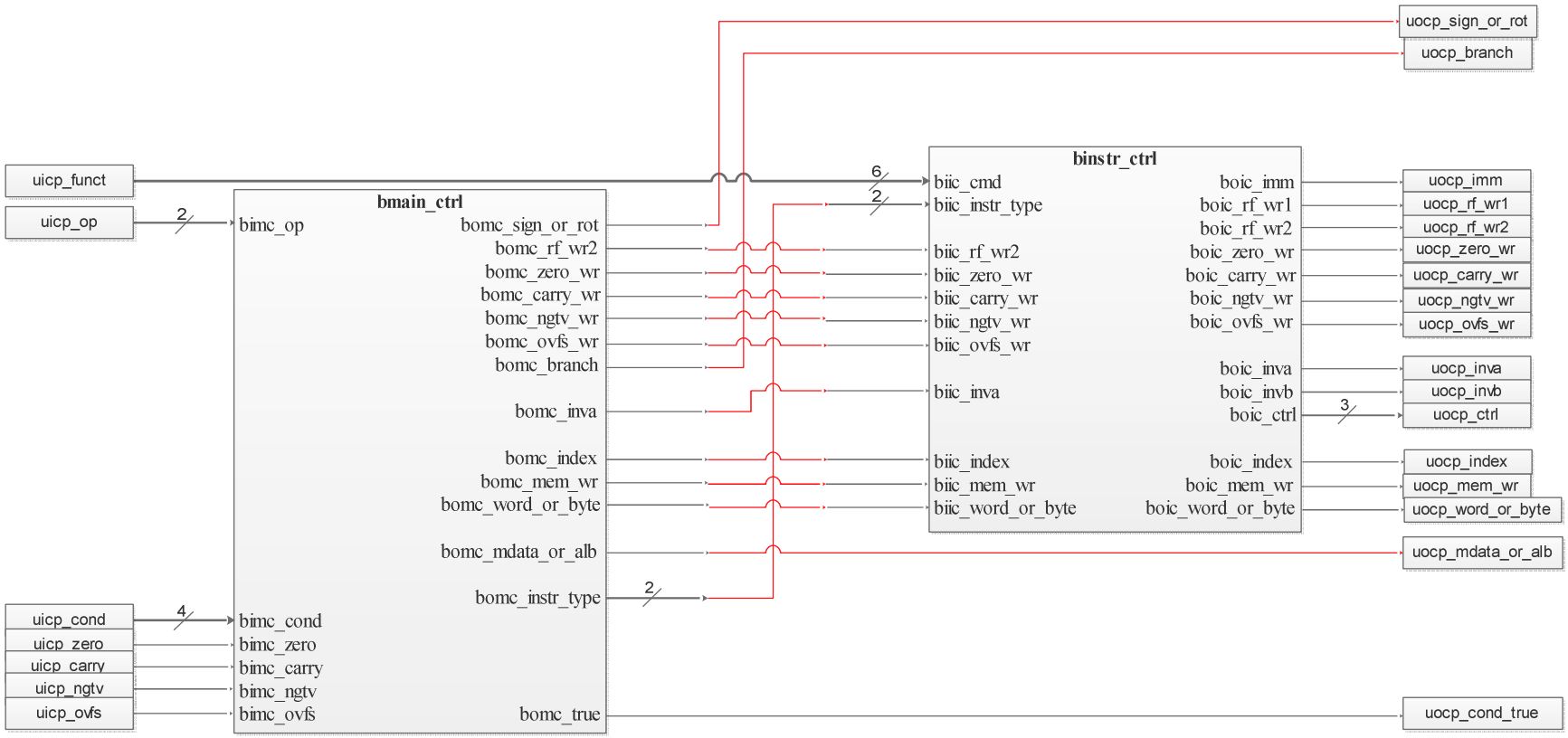


Figure 8.3: internal connection between block in ucp

8.2 Main Control Block (bmain_ctrl)

8.2.1 Functionality

Generate some control signals that are same within a single instruction type (e.g. Data-processing instruction, Memory instruction).

8.2.2 Block diagram

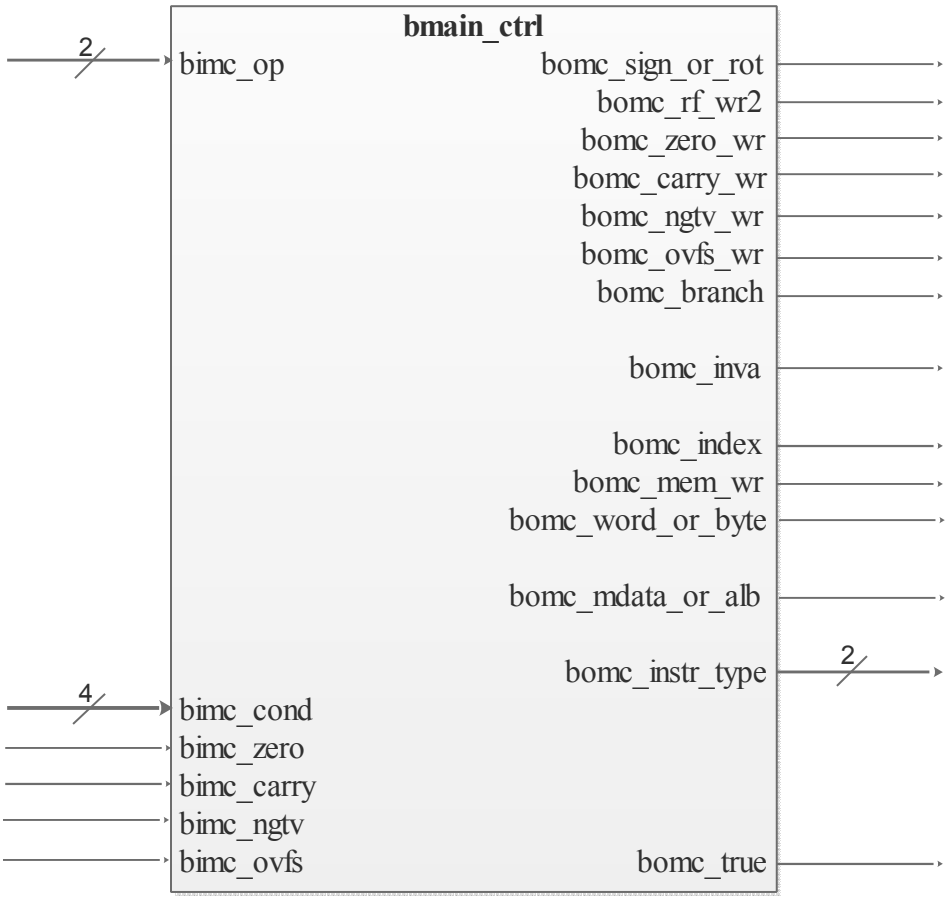


Figure 8.4: Block diagram of main control block

Input

Pin name : bimc_op Registered : No Pin class : data signal Source → Destination : ucp → bmain_ctrl Bit size : 2-bit Active : - Pin Function: 27th and 26th bit of the instruction, differentiate among the instruction type
Pin name : bimc_cond Registered : No Pin class : data signal Source → Destination : ucp → bmain_ctrl

Bit size : 4-bit Active : - Pin Function: mask that represent different condition where the instruction should execute.
Pin name : bimc_zero Registered : No Pin class : data signal Source → Destination : ucp → bmain_ctrl Bit size : 1-bit Active : - Pin Function: Latest zero flag that base on instruction that executing or executed in EX stage. 1'b 0: the result is non-zero. 1'b 1: the result is zero.
Pin name : uicp_carry Registered : No Pin class : data signal Source → Destination : ucp → bmain_ctrl Bit size : 1-bit Active : Active high Pin Function: Latest carry out flag that base on instruction that executing or executed in EX stage. 1'b 0: no carry is produced. 1'b 1: carry is produced.
Pin name : uicp_ngtv Registered : No Pin class : data signal Source → Destination : ucp → bmain_ctrl Bit size : 1-bit Active : Active high Pin Function: Latest negative flag that base on instruction that executing or executed in EX stage. 1'b 0: the result's MSB is 1'b0. 1'b 1: the result's MSB is 1'b1.
Pin name : uicp_ovfs Registered : No Pin class : data signal Source → Destination : ucp → bmain_ctrl Bit size : 1-bit Active : Active high Pin Function: Latest overflow flag that base on instruction that executing or executed in EX stage. 1'b 0: no overflow occur. 1'b 1: overflow occur.

Table 8.7: Input pin description of main control block

Output

Pin name : bomc_sign_or_rot Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → u_cp Bit size : 1-bit Active : Active high Pin Function: Active high when operand 2 will undergoes sign extension else active low for rotation extension 1'b 0 : rotation extension operand 2 1'b 1: sign extension operand 2
Pin name : bomc_rf_wr2 Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high to enable 2nd write port else active low 1'b 0 : hold the data 1'b 1: write the data to 2nd write address
Pin name : bomc_carry_wr Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high to enable update carry flag else active low 1'b 0 : hold previous carry flag 1'b 1: update carry flag
Pin name : bomc_zero_wr Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high to enable update zero flag else active low 1'b 0 : hold previous zero flag 1'b 1: update zero flag
Pin name : bomc_ngtv_wr Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high to enable update negative flag else active low 1'b 0 : hold previous negative flag 1'b 1: update negative flag
Pin name : bomc_ovfs_wr Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit

Active : Active high Pin Function: Active high to enable update overflow flag else active low 1'b 0 : hold previous overflow flag 1'b 1: update overflow flag
Pin name : bomc_branch Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high to change PC to branch target address else active low for normal increment of PC (+4) 1'b 0 : branch to target address 1'b 1: normal +4 increment of PC
Pin name : uocp_inva Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high to invert 1st operand (from Rn) else active low to use original operand 1'b 0 : use original data from Rn for ALB 1'b 1: invert the data from Rn before going through ALB
Pin name : uocp_index Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : - Pin Function: decide the address mode for memory read and load 1'b 0 : Post-index 1'b 1 : Pre-index
Pin name : uocp_mem_wr Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high to update memory else active low to hold the previous memory data. 1'b 0 : hold previous memory data 1'b 1: update memory data
Pin name : uocp_word_or_byte Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Active high for load or save a byte of data else active low for a word of data

1'b 0 : word 1'b 1: byte
Pin name : uocp_mdata_or_alb Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high for write data from memory to register file else active low for write data from ALB to register file. 1'b 0 : use data from ALB 1'b 1: use data from memory cache
Pin name : bomc_instr_type Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 2-bit Active : - Pin Function: Represent the instruction type based on the bimc_op 2'b 00 : data-processing instruction 2'b 01 : memory instruction 2'b 10 : program flow instruction
Pin name : uocp_cond_true Registered : Yes Pin class : control signal Source → Destination : bmain_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high if the condition meet else active low to skip the instruction 1'b 0 : skip the instruction 1'b 1: execute the instruction

Table 8.8: Output pin description of main control blo

8.3 Instruction Control Block (binstr_ctrl)

8.3.1 Functionality

Generate the control signals that might be different within a sing type of instruction type.

8.3.2 Block diagram

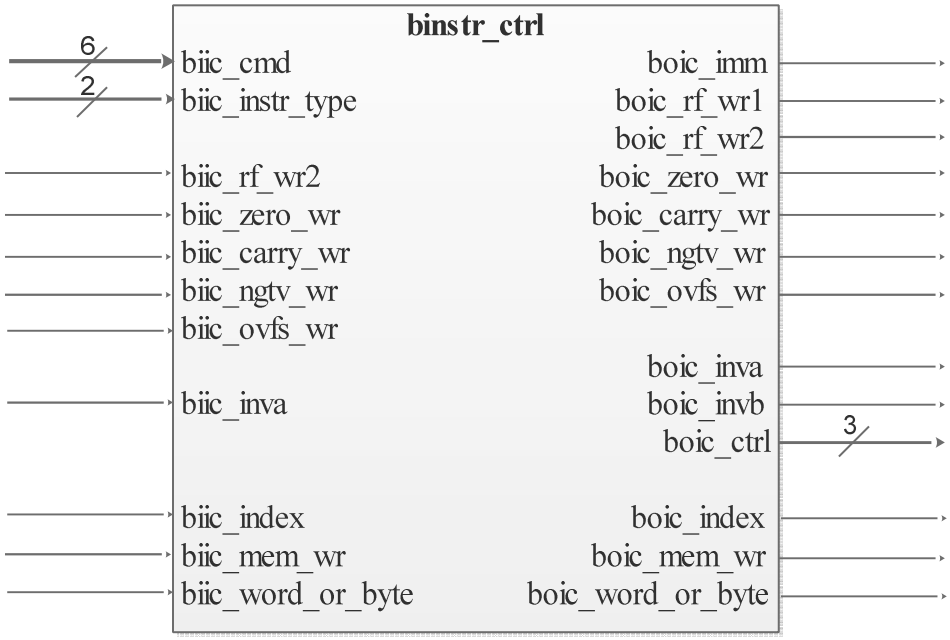


Figure 8.5: block diagram of binstr_ctrl

Input

Pin name : biic_cmd Registered : No Pin class : data signal Source → Destination : ucp → binstr_ctrl Bit size : 6-bit Active : - Pin Function: Carry the information of instruction for each instruction type. Such as operand 2 is an immediate, operation to be carry out and etc.
Pin name : biic_instr_type Registered : No Pin class : data signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 2-bit Active : - Pin Function: Intruction type of the current instruction generated by bmain_ctrl 2'b 00 : data processing instruction 2'b 01 : memory instruction 2'b 10 : program flow instruction
Pin name : biic_rf_wr2 Registered : No Pin class : data signal

Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Generated by bmain_ctrl, the value might change based on instruction type
Pin name : biic_zero_wr Registered : No Pin class : data signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Generated by bmain_ctrl, the value might change based on instruction type
Pin name : biic_carry_wr Registered : No Pin class : data signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Generated by bmain_ctrl, the value might change based on instruction type
Pin name : biic_ngtv_wr Registered : No Pin class : data signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Generated by bmain_ctrl, the value might change based on instruction type
Pin name : biic_ovfs_wr Registered : No Pin class : data signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Generated by bmain_ctrl, the value might change based on instruction type
Pin name : biic_inva Registered : No Pin class : data signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Generated by bmain_ctrl, the value might change based on instruction type
Pin name : biic_index Registered : No Pin class : data signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Generated by bmain_ctrl, the value might change based on

instruction type
Pin name : biic_mem_wr Registered : No Pin class : data signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Generated by bmain_ctrl, the value might change based on instruction type
Pin name : biic_word_or_byte Registered : No Pin class : data signal Source → Destination : bmain_ctrl → binstr_ctrl Bit size : 1-bit Active : Active high Pin Function: Generated by bmain_ctrl, the value might change based on instruction type

Table 8.11: input pins description of binstr_ctrl

Output

Pin name : boic_imm Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high when the operand 2 is an immediate else active low 1'b 0 : non-immediate operand 1'b 1: immediate operand
Pin name : boic_rf_wr1 Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high to enable 1st write port else active low 1'b 0 : hold the data 1'b 1: write the data to 1st write address
Pin name : boic_rf_wr2 Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high to enable 2nd write port else active low 1'b 0 : hold the data 1'b 1: write the data to 2nd write address
Pin name : boic_carry_wr Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit

Active : Active high Pin Function: Active high to enable update carry flag else active low 1'b 0 : hold previous carry flag 1'b 1: update carry flag
Pin name : boic_zero_wr Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update zero flag else active low 1'b 0 : hold previous zero flag 1'b 1: update zero flag
Pin name : boic_ngtv_wr Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update negative flag else active low 1'b 0 : hold previous negative flag 1'b 1: update negative flag
Pin name : boic_ovfs_wr Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high to enable update overflow flag else active low 1'b 0 : hold previous overflow flag 1'b 1: update overflow flag
Pin name : boic_inva Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high to invert 1st operand (from Rn) else active low to use original operand 1'b 0 : use original data from Rn for ALB 1'b 1: invert the data from Rn before going through ALB
Pin name : boic_invb Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high to invert 2nd operand (from Rm or immediate) else active low to use original operand 1'b 0 : use original data from Rm or immediate for ALB 1'b 1: invert the data from Rm or immediate before going through ALB

Pin name : boic_ctrl Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 3-bit Active : - Pin Function: opcode for the ALB. 3'b 000 : addition 3'b 001: addition with carry 3'b 010: subtraction 3'b 011: subtraction with carry 3'b 100: and AND 3'b 101: or OR 3'b 110: exclusive or XOR 3'b 111: by pass operand b (from Rm)
Pin name : boic_index Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: decide the address mode for memory read and load 1'b 0 : Post-index 1'b 1 : Pre-index
Pin name : boic_mem_wr Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high to update memory else active low to hold the previous memory data. 1'b 0 : hold previous memory data 1'b 1: update memory data
Pin name : boic_word_or_byte Registered : Yes Pin class : data signal Source → Destination : binstr_ctrl → ucp Bit size : 1-bit Active : Active high Pin Function: Active high for load or save a byte of data else active low for a word of data 1'b 0 : word 1'b 1: byte

Table 8.12: Output pins description of binstr_ctrl

Chapter 9 – Memory Cache unit (ucache)

9.1 Functionality

Data segment and Text segment in memory map.

9.2 Block diagram

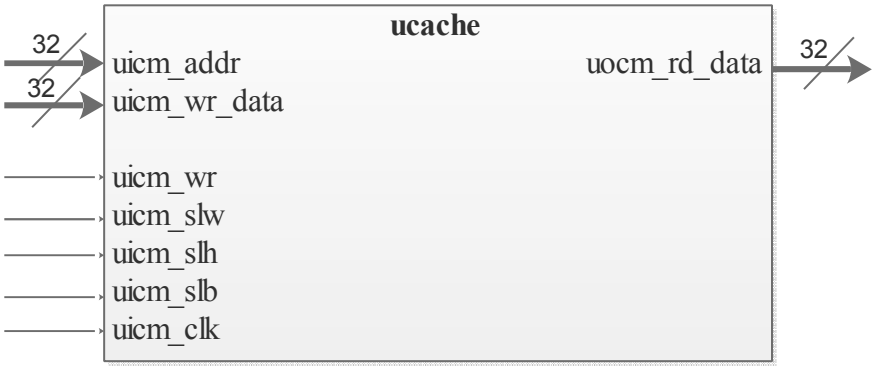


Figure 9.1: block diagram of ucache

Input

Pin name : uicm_addr Registered : No Pin class : data signal Source → Destination : crisc_pipeline → ucache Bit size : 32-bit Active : - Pin Function: Address for data write/read to ucache.
Pin name : uicm_wr_data Registered : No Pin class : data signal Source → Destination : crisc_pipeline → ucache Bit size : 32-bit Active : - Pin Function: Data to write to ucache.
Pin name : uicm_wr Registered : No Pin class : control signal Source → Destination : udp → ucache Bit size : 1-bit Active : Active high Pin Function: 1'b0: hold the content of ucache 1'b1: write the uicm_wr_data to the ucache with uicm_addr as address.
Pin name : uicm_slw Registered : No Pin class : control signal Source → Destination : udp → ucache Bit size : 1-bit Active : Active high Pin Function: 1'b0: data operate in other unit (non-word). 1'b1: data operate in word (unit).
Pin name : uicm_slh Registered : No Pin class : control signal

Source → Destination : udp → ucache Bit size : 1-bit Active : Active high Pin Function: 1'b0: data operate in other unit (non-half-word). 1'b1: data operate in half-word (unit).
Pin name : uicm_slb Registered : No Pin class : control signal Source → Destination : udp → ucache Bit size : 1-bit Active : Active high Pin Function: 1'b0: data operate in other unit (non-byte). 1'b1: data operate in byte (unit).
Pin name : uicm_clk Registered : No Pin class : clock signal Source → Destination : external → ucache Bit size : 1-bit Active : Rising edge Pin Function: Provide a periodic signal for synchronize purpose.

Table 9.1: input pin description of ucache

Output

Pin name : uocm_rd_data Registered : Yes Pin class : data signal Source → Destination : ucache → udp Bit size : 32-bit Active : - Pin Function: data read from ucache with uicm_addr as the address.

Table 9.2: Output pin description of ucache

Chapter 10 – UART unit

A developed UART unit (uuart) is connected with core (data-path and control path). However, due to the reason that exception handler hasn't develop yet, the functionalities of UART used in this project is very limited, the purpose of connect the UART is to show the simple interconnection of I/O with the core.

10.1 UART address

In this project, the address of UART is set as 32'h C000_0004 ~ 32'h C000_0010 which in I/O segment of memory map.

Address	UART's register	uiua_wb_sel [3:0]
32'h C000_0004	Configuration register (UARTCF)	4'b 0001
32'h C000_0008	Transmitter fifo register	4'b 0100
32'h C000_000C	Receiver fifo register	4'b 1000
32'h C000_0010	Status register (UARTSF)	4'b 0010

Table 10.1: Address for UART registers and FIFO

10.2 Operating procedure

In this project, only focus on the transmission of data by UART. Refer to previous developed UART, before start a transmission UART will send a Request-To-Send (RTS) signal to external modern and waiting for a Clear-to-Send (CTS) Signal from the external modern. After UART detect the CTS signal, the data will transmit to the external modern bit-by-bit in a configurable baud rate.

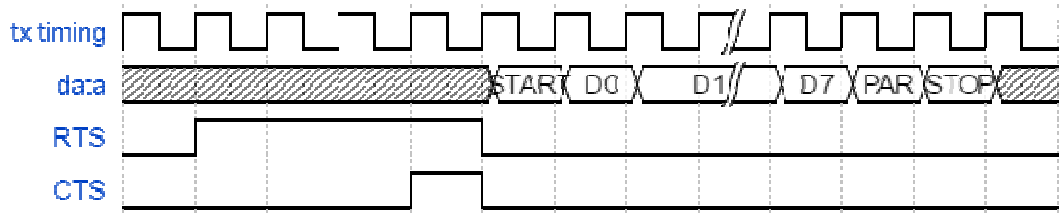


Figure 10.1: Transmission of data by UART

The 8-bit data (d [7:0]) will transmit in a format of {1'b0, d[0], d[1], …, d[7], parity bit, 1'b1} as shown in the diagram below.

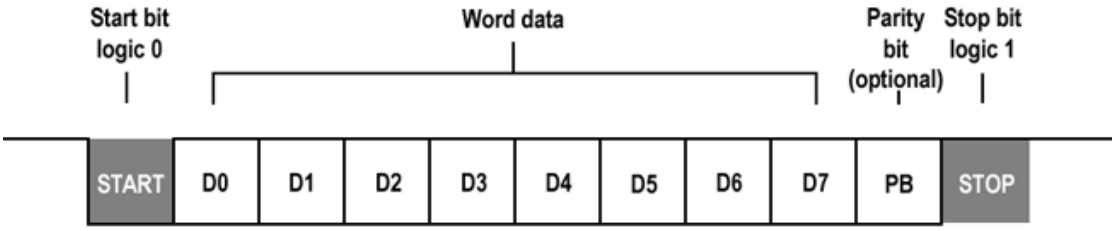


Figure 10.2: UART data transfer protocol

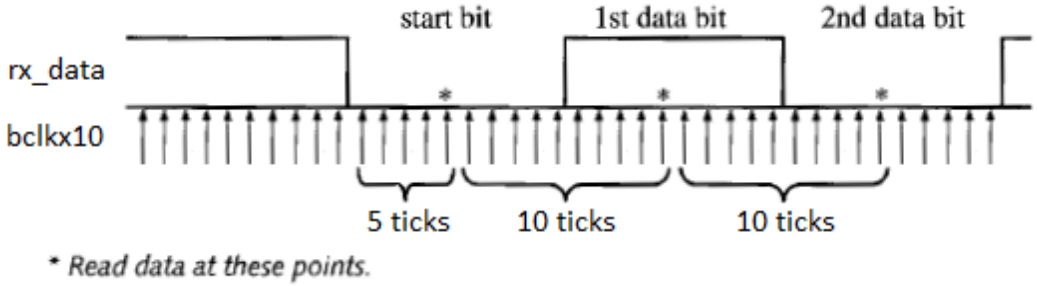


Figure 10.3: UART data receiving protocol

The data receiving will be on the same baud rate. At the rising edge, the data might be not ready (transition might occur) to read. Hence, to avoid read wrong data (at the rising edge) each receiving data will be read approximate at the middle of the clock frame. The 1st clock frame, the receiving data will read after 5 periods of bclkx10, and 10 periods for following receiving data until stop bit. The alternative way will be read the data at falling edge of the baud rate, since the falling edge is at the middle of a clock period (if duty cycle equal to 50%) where the data should be ready.

10.3 uuart functionalities and pin description

- Serialize 8-bit data.
- Transmit the serialized data
- Receive serial data and parallelize to 8-bit.
- Check correctness of data.

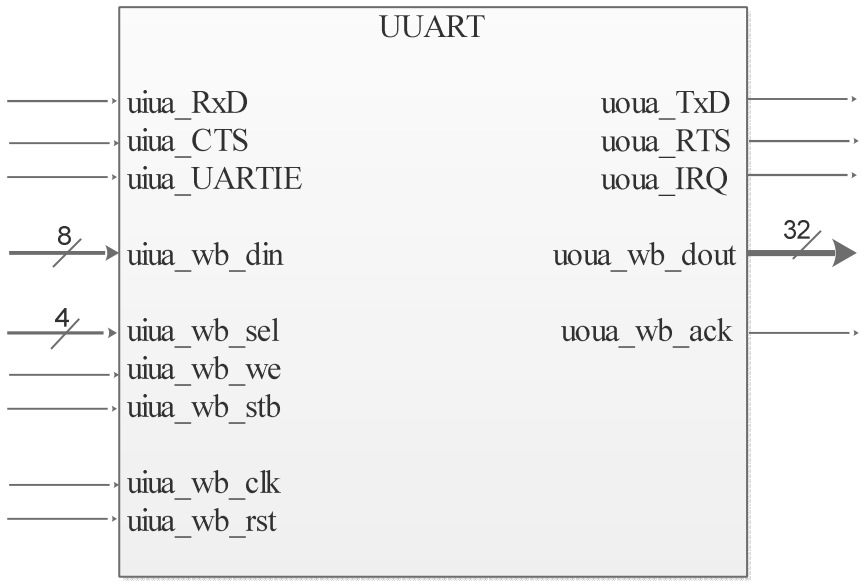


Figure 10.4: Block diagram for uuart

Input

Pin name: uiua_RxD Registered: No Pin class: data Source → Destination: external device → uuart Bit size: 1-bit Active: - Pin Function: Received data from external device through UART port.
Pin name: uiua_CTS Registered: No Pin class: status signal Source → Destination: CPU → uuart Bit size: 1-bit Active: High Pin Function: Allow UART to transmit data when 1'b1.
Pin name: uiua_UARTIE Registered: No Pin class: control signal Source → Destination: CPU → uuart Bit size: 1-bit Active: High Pin Function: Interrupt enable
Pin name: uiua_wb_din Registered: No

Pin class: data Source → Destination: CPU → uuart Bit size: 8-bits Active: - Pin Function: Data to write in UART's registers from CPU.
Pin name: uiua_wb_sel Registered: No Pin class: control signal Source → Destination: Address decoder → uuart Bit size: 4-bits Active: - Pin Function: Select the register in UART to write. 4'b 0001: control register UARTCR 4'b 0010: status register UARTSR 4'b 0100: transmitter FIFO register push enable tx_fifo_push_en 4'b 1000: receiver FIFO register pop enable rx_fifo_pop_en
Pin name: uiua_wb_we Registered: No Pin class: control signal Source → Destination: CPU → uuart Bit size: 1-bit Active: High Pin Function: Allow to data write the register depend on uiua_wb_sel.
Pin name: uiua_wb_stb Registered: No Pin class: status signal Source → Destination: CPU → uuart Bit size: 1-bit Active: High Pin Function:
Pin name: uiua_wb_clk Registered: No Pin class: clock signal Source → Destination: System → uuart Bit size: 1-bit Active: Rising edge Pin Function: Periodic signal for synchronize purpose
Pin name: uiua_wb_rst Registered: No Pin class: control signal Source → Destination: System → uuart Bit size: 1-bit Active: High Pin Function: Reset the UART

Table 10.2: Input pins description for uuart

Output

Pin name: uoua_TxD Pin class: data Source → Destination: uuart → external device Bit size: 1-bit Active: - Pin function: Transmit content in FIFO of transmitter from UART to external device	Registered: Yes
Pin name: uoua_RTS Pin class: status signal Source → Destination: uuart → external device Bit size: 1-bit Active: high Pin function: active high indicate UART request to send data.	Registered: Yes
Pin name: uoua_IRQ Pin class: status signal Source → Destination: uuart → CPU Bit size: 1-bit Active: high Pin Function: error interrupt signal	Registered: Yes
Pin name: uoua_wb_dout Pin class: data Source → Destination: uuart → CPU Bit size: 8-bits Active: - Pin Function: data from UART to CPU register	Registered: Yes
Pin name: uoua_wb_ack Pin class: status signal Source → Destination: uuart → CPU Bit size: 1-bit Active: high Pin Function: acknowledgement to CPU	Registered: Yes

Table 10.3: Outputs pin description for uuart.

Configure Register

The configure register is used to decide interrupt, parity and baud rate.
Bit 7 (UARTEN): UART enable
Bit 6 (RXCIE): Receive Error interrupt enable
Bit 5 (TXEIE): Transmit Error interrupt enable
Bit 4 (PRTEN): Parity Bit Enable
Bit 3 (PRT): Parity Bit
Bit [2:0] (BAUD): Baud rate select

Status Register

Status register will represent the status of received data.
Bit 7 (RXC): Receive status
Bit 6 (TXE): Transmission status
Bit 5 (FE): Framing Error
Bit 4 (PE): Parity Error
Bit [3:1]: Not used
Bit 0 (RxFIM):

10.4 bclkctr functionalities and pins description

- Generate different baud rate. (8)
- Enable transmitter (at rising edge) and receive block (at falling edge).

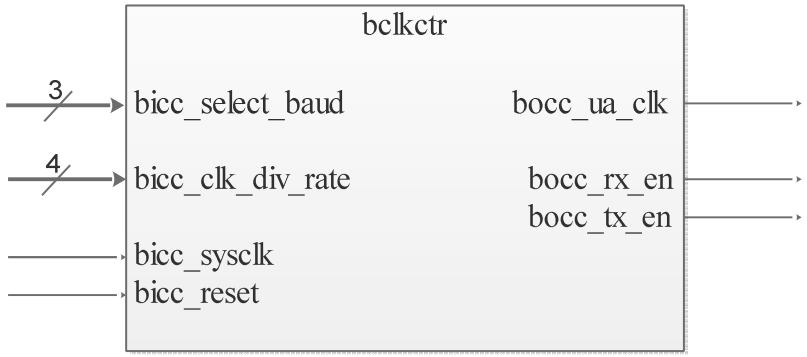


Figure 10.6: block diagram for bclkctr

Input pins

Pin name: bicc_select_baud Register: No Pin class: control signal Source → Destination: UARTCR → bclkctr Bit size: 3-bits Active: - Pin Function: Select baud rate
Pin name: bicc_clk_div_rate Register: No Pin class: control signal Source → Destination: fixed to 4'b 0001 Bit size: 4-bits Active: - Pin Function: the value used to divide the system clock (in this case fixed to 4'b 0001 to divide by 2)
Pin name: bicc_sysclk Register: No Pin class: clock signal Source → Destination: System → bclkctr Bit size: 1-bit Active: - Pin Function: Provide periodic signal for synchronize purpose.
Pin name: bicc_reset Register: No Pin class: control signal Source → Destination: System → bclkctr Bit size: 1-bit Active: High Pin Function: Reset the system to initial condition.

Table 10.4: Inputs pin description for bclkctr

Output pins

Pin name: bocc_ua_clk Pin class: clock signal Source → Destination: bclkctr → btx/brx Bit size: 1-bit Active: - Pin Function: divided clock	Register: Yes
Pin name: bocc_rx_en Pin class: control signal Source → Destination: bclkctr → brx Bit size: 1-bit Active: High Pin Function: Allow receiver block to receive data	Register: Yes
Pin name: bocc_tx_en Pin class: control signal Source → Destination: bclkctr → btx Bit size: 1-bit Active: High Pin Function: Allow transmitter block to transmit data	Register: Yes

Table 10.5: Output pins description for bclkctr

10.5 brx functionalities and pins description

- Receive a data stream from external device.
- Parallelize the data to 8-bit data.
- Check framing error and parity error.

Figure 10.8: block diagram for brx

Input pins

Pin name: birx_rx_data Pin class: data signal Source → Destination: external device → brx Bit size: 1-bit Active: - Pin Function: Data from external device.	Register: No
Pin name: birx_rx_en Pin class: control signal Source → Destination: bclkctr → brx Bit size: 1-bit Active: High Pin Function: Allow receiver block receive data.	Register: No
Pin name: birx_parity_en Pin class: control signal Source → Destination: UARTCR → brx Bit size: 1-bit Active: High Pin Function: Inform receiver block that parity bit is enable (need to check parity bit).	Register: No
Pin name: birx_parity_bit Pin class: data signal Source → Destination: UARTCR → brx	Register: No

Bit size: 1-bit Active: - Pin Function: Expected parity bit.
Pin name: birx_fifo_pop_en Register: No Pin class: control signal Source → Destination: uuart → brx Bit size: 1-bit Active: High Pin Function: request the data stored in FIFO to borx_data_out.
Pin name: birx_sysclk Register: No Pin class: clock signal Source → Destination: System → brx Bit size: 1-bit Active: - Pin Function: Provide a periodic signal for synchronize purpose.
Pin name: birx_ua_clk Register: No Pin class: clock signal Source → Destination: bclkctr → brx Bit size: 1-bit Active: - Pin Function: divided clock
Pin name: birx_reset Register: No Pin class: control signal Source → Destination: System → brx Bit size: 1-bit Active: High Pin Function: Reset the receiver block to initial condition.

Table 10.6: Input pins for brx

Output pins

Pin name: borx_data_out Register: Yes Pin class: data signal Source → Destination: brx → CPU Bit size: 8-bit Active: - Pin Function: Data received.
Pin name: borx_rxc Register: Yes Pin class: status signal Source → Destination: brx → UARTSR Bit size: 1 bit Active: High Pin Function: receive status
Pin name: borx_parity_err Register: Yes Pin class: status signal Source → Destination: brx → UARTSR Bit size: 1 bit

Active: High Pin Function: Indicate the received data have parity error while 1'b1.
Pin name: borx_framing_err Register: Yes Pin class: status signal Source → Destination: brx → UARTSR Bit size: 1-bit Active: High Pin Function: framing error at received data.
Pin name: borx_fifo_empty Register: Yes Pin class: status signal Source → Destination: brx → btx Bit size: 1-bit Active: High Pin Function: Indicate the FIFO in receiver block is empty while 1'b1.
Pin name: borx_fifo_full Register: Yes Pin class: status signal Source → Destination: brx → btx Bit size: 1-bit Active: High Pin Function: Indicate the FIFO in receiver block is full while 1'b1.

Table 10.7: Output pins for brx

10.6 btx functionalities and pins description

- Serialize the 8-bit data to a stream of data.
- Append start bit (1'b 0), serialized data, parity bit and stop bit (1'b 1) together.
- Transmit ready data to receiver of the external device.

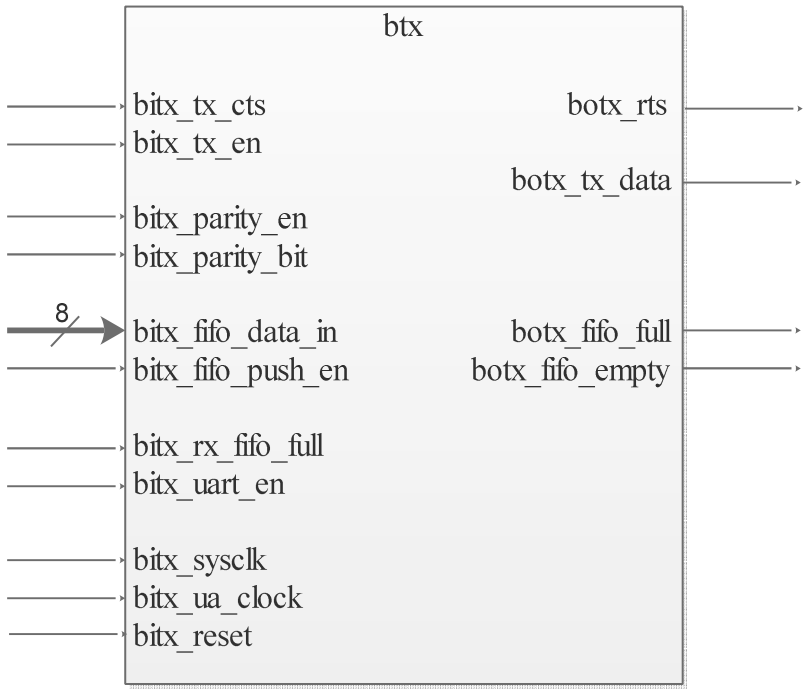


Figure 10.10: block diagram for btx

Input pins

Pin name: bitx_tx_cts Pin class: status signal Source → Destination: → btx Bit size: 1-bit Active: High Pin Function: indicate the external device ready to receive data (allow to transmit) while 1'b1.	Register: No
Pin name: bitx_tx_en Pin class: control signal Source → Destination: bclkctr → btx Bit size: 1-bit Active: High Pin Function: Allow the transmitter block to transmit data.	Register: No
Pin name: bitx_parity_en Pin class: control signal Source → Destination: UARTCR → btx	Register: No

Bit size: 1-bit Active: High Pin Function: Parity bit need to be generated while 1'b1.
Pin name: bitx_parity_bit Register: No Pin class: data signal Source → Destination: UARTCR → btx Bit size: 1-bit Active: High Pin Function: Parity bit value to be transmit.
Pin name: bitx_fifo_data_in Register: No Pin class: data signal Source → Destination: uuart → btx Bit size: 8-bit Active: - Pin Function: Data to store in FIFO before transmission.
Pin name: bitx_fifo_push_en Register: No Pin class: control signal Source → Destination: uuart → btx Bit size: 1-bit Active: High Pin Function: Store the data to FIFO while 1'b1.
Pin name: bitx_rx_fifo_full Register: No Pin class: status signal Source → Destination: brx → btx Bit size: 1-bit Active: High Pin Function: Indicate the FIFO in receiver block is full.
Pin name: bitx_sysclk Register: No Pin class: clock signal Source → Destination: System → btx Bit size: 1-bit Active: - Pin Function: Provide a periodic signal for synchronize purpose.
Pin name: bitx_ua_clk Register: No Pin class: clock signal Source → Destination: bclkctr → btx Bit size: 1-bit Active: - Pin Function:
Pin name: bitx_reset Register: No Pin class: control signal Source → Destination: System → btx Bit size: 1-bit Active: High Pin Function: Reset the transmitter block to initial condition.

Table 10.8: Inputs pin description for btx

Output pins

Pin name: botx_rts Pin class: status signal Source → Destination: btx → external device Bit size: 1-bit Active: High Pin Function: Request to send data from transmitter block.	Register: Yes
Pin name: botx_tx_data Pin class: data signal Source → Destination: btx → external device But size: 1-bit Active: - Pin Function: Data stream transmit.	Register: Yes
Pin name: botx_fifo_full Pin class: status signal Source → Destination: btx → none Bit size: 1-bit Active: High Pin Function: Indicate the FIFO in transmitter block is full while 1'b1.	Register: Yes
Pin name: botx_fifo_empty Pin class: status signal Source → Destination: btx → none Bit size: 1-bit Active: High Pin Function: Indicate the FIFO in transmitter block is empty while 1'b1.	Register: Yes

Table 10.9: Output pins description for btx

10.7 UART address decoder

The UART address decoder will only work on 2 address which is 32'h C000_0008 for transmitter FIFO and C000_0004 UART for configuration register since only UART is used for transmission of data only. Figure below show the combinational logic of the decoder.

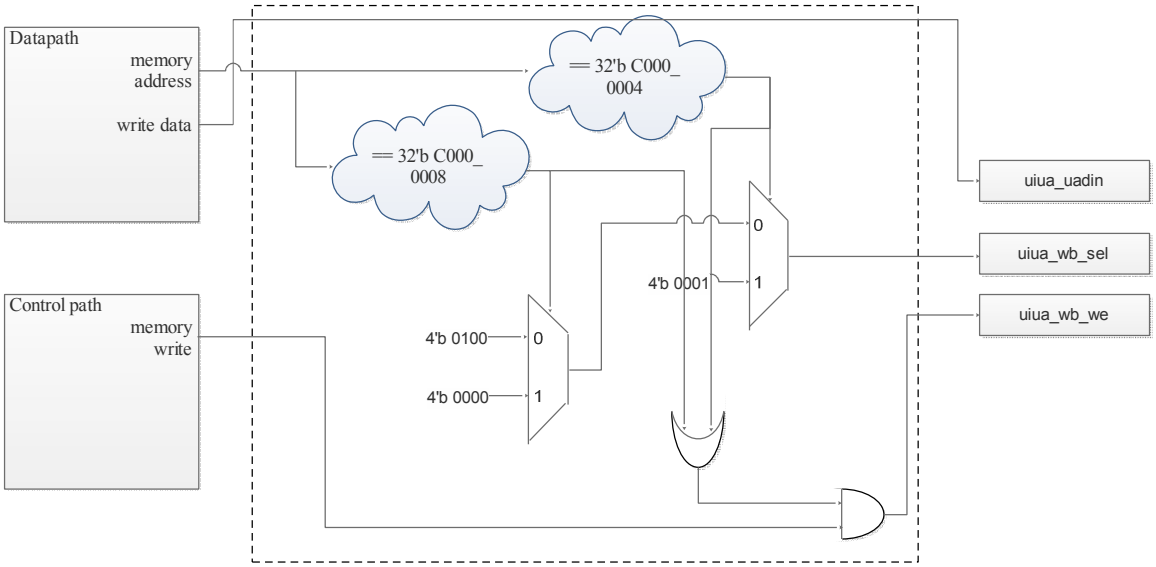


Figure 10.12: Circuit for CPU-UART address decoder

Chapter 11 – Verification Specification

11.1 Verification for crisc

Verification is carrying out after the crisc_pipelineverilog module had designed. The verification is done with load a text file (with .arm extension) to the Text segment memory cache, since this project only include the user instruction (Arithmetic, logical, memory, and program flow).

Verilog code:

$readmemh ("test_instr.arm",tb_crisc_pipeline.DUT.utext_seg.ucm_r_memory);

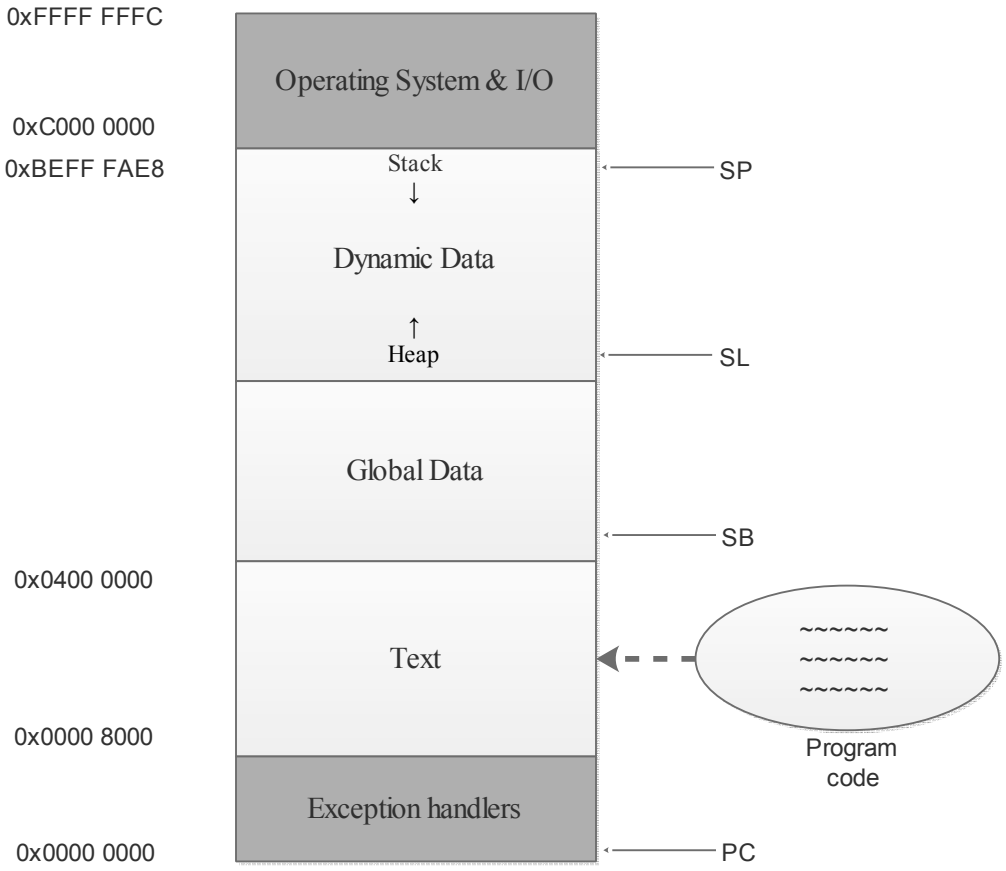


Figure 11.1: Memory map & program code segment

The correctness of the Verilog module is confirmed with the comparison of Register file and data memory of Verilog module with ARMSim (ARM assembly instruction simulator).

11.2 Test Program for RISC 32

The following test program in **Table 11.1** is a program which consists most of the instruction to test crisc_pipeline. This program is a hazard and data dependency free program. When verifying using this program, the outcome of crisc_pipelinemust be the same with the expected output stated. The main purpose of this program is to ensure the correctness of each instruction which involve data-path unit (udp) and control unit (ucp).

The test program in Table 11.2 is a recursive program which is full of data dependency. The NOP is not inserted in the program to test the Data forwarding (bfw_ctrl) and Interlock control (bitl_ctrl). The main purpose is to make sure the program is free from data hazard.

11.2.1 Test program 1

Each instruction of the test program 1 is not related to each other, therefore the register file and data memory should be observe after each instruction. The correctness is verified with ARMSim.

data_processing	ADD	R0, R0, #16	R0 = 16
	ADD	R1, R1, R0	R1 = 16
	ADD	R2, R1, R0, LSL #2	R2 = 16 +16 * 4 = 80
	ADD	R3, R1, R0, LSL R0	R3 = 16 + 16*2^16 = 1048592
	CMP	R0, R4	Carry = 1
	ADC	R4, R4, #16	R4 = 16 + 1 =17
	ADC	R5, R5, R4	R5 = 17 + 1 = 18
	ADC	R6, R5, R4, LSL #1	R6 = 18 + 17*2 + 1 =53
	ADC	R7, R5, R4, LSL R4	R7 = 18 + 17*2^17 + 1 = 2228243
	SUB	R1, R1, #16	R1 = 16-16 =0
	SUB	R0, R4, R0	R0 = 17 -16 = 1
	SUB	R2, R2, R0, LSL #6	R2 = 80 - 1*2^6 = 16
	SUB	R3, R3, R7, LSR R0	R3 = 1048592 - 1114121 = -65529
	RSB	R4, R4, R5	R4 = 18 - 17 = 1
	RSB	R5, R5, R4	R5 = 1 -18 = -17
	RSB	R6, R6, R2, LSL #2	R6 = 16 * 2^2 -53 = 11
	RSB	R7, R7, R6, LSL R6	R7 = 22528 - 2228243 = -2205715
	SBCS	R0, R0, #0	R0 = 1 - 0 - !1 = 1; C = 1
	SBC	R1, R4, R1	R1 = 1 - 0 - !1 = 1
	SBCS	R2, R2, R4, LSL #4	R2 = 16 - 1*2^4 - !0 = -1; C = 1

			R3 = 0xFFFF FFFF - 0xFFFF	
	SBC	R3, R2, R3, ASR R6	FFFF =0	
	TST	R3, #0Xff	N = 0, Z = 0, C = 1	
	TST	R3, R1	N = 0, Z= 1, C = 1	
	TEQ	R1, #0xFF	N = 0, Z= 0, C= 1	
	TEQ	R1, R3, LSL #26	N=0, Z = 0, C = 0	
	AND	R4, R7, #0xFF	R4 = 0xED	
	EOR	R5, R4, #0xF0	R5 = 0x1D	
	ORR	R6, R4, #0xF0	R6 = 0xFD	
	BIC	R7, R4, #0xF0	R7 = 0x0D	
	CMP	R7, R7	N = 0, Z = 1, C = 1, V =0	
	CMN	R7, R6	N = 0, Z = 0, C =0, V = 0	
	MOV	R8, R7	R8 = 0x0D	
	MOV	R8, #0xF0	R8 = 0xF0	
	MVN	R9, R7	R9 = 0xFFFF FFF2	
	MVN	R9, #0xF0	R9 = 0XFFFF FF0F	
	MOV	R8, R9, LSL R7	R8 = 0XFFE1 E000	#(-8+16=8) -> mem[2]
	MOV	R8, R9, LSR #8	R8 = 0x00FF FFFF	
	MOV	R8, R9, ASR #3	R8 = 0XFFFF FFE1	#(-8+20=8) -> mem[3]
	MOV	R8, R9, ROR #9	R8 = 0x87FF FFFF	
	MOV	R2, R0, LSL #28	R2 = 0x1000 0000	
	ORR	R2, R2, R0, LSL #15	R2 = 0X1000 8000	
memory:	STR	R8, [R2]	Dmem[0x1000 8000] = 0x87FF FFFF	
	STR	R9, [R2, #4]	Dmem[0x1000 8004] = 0xFFFF FF0F	
	STR	R0, [R2], #8	Dmem[0x1000 8000] = 0X0000 0001; R2 = 0x1000 8008	
	STR	R8, [R2, #4]!	R2 = 0x1000 800C; Dmem[0x1000 800C] = 0x87FF FFFF	

	LDR	R0, [R2], #-4	R0 = 0x87FF FFFF; R2 = 0x1000 8008
	LDR	R0, [R2, #-4]	R0 = Dmem[0x1000 8000] = 0xFFFF FF0F
	LDR	R0, [R2, #-8]!	R2 = 0x1000 8000; R0 = 0x0000 0001
			below is for test forwarding
	ADD	R0, R0, #-1	R0 = 0x0000 0000
	LDR	R0, [R0, R2]!	R0 = 0X1000 8000; 2nd write port higher priority
	BL	branch	LR = here
here:	B	end	
branch:	MOV	PC, LR	PC = LR, similar to jump in MIPS
	Nop		
	Nop		
	Nop		
	Nop		
end:	MOV	R0, #0	R0 = 0; indicate end of program

Table 10.1 Test program 1 (without data dependency, interlock and hazard.)

11.2.2 Verification for test program 1 for RISC32

For the verification, we need to track the value of register file and memory segment from time to time in order to make sure the correctness.

For Data Processing Instruction (Note-the result is arranged in time increasing order):

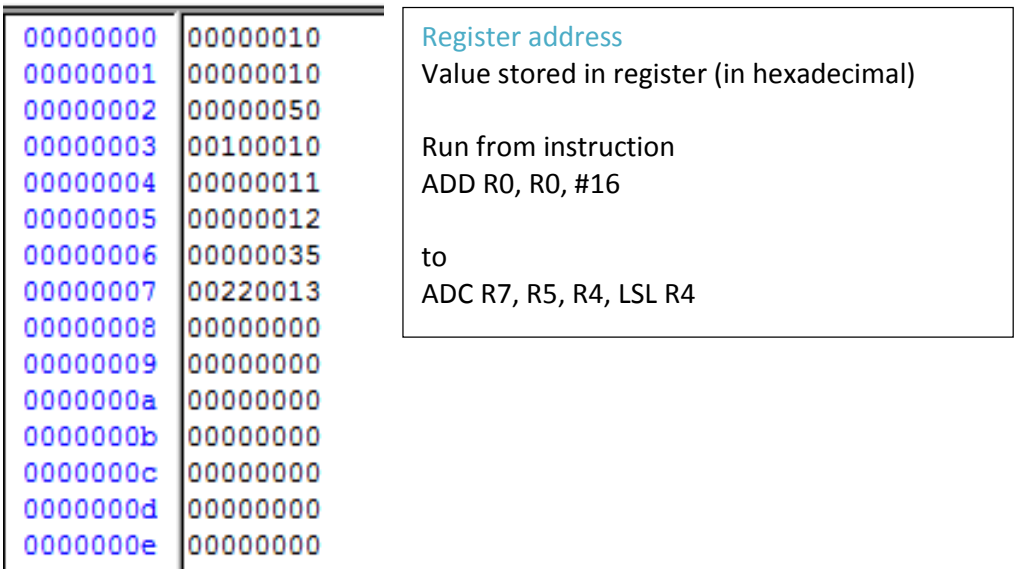


Figure 11.2: Test program 1 result (1)

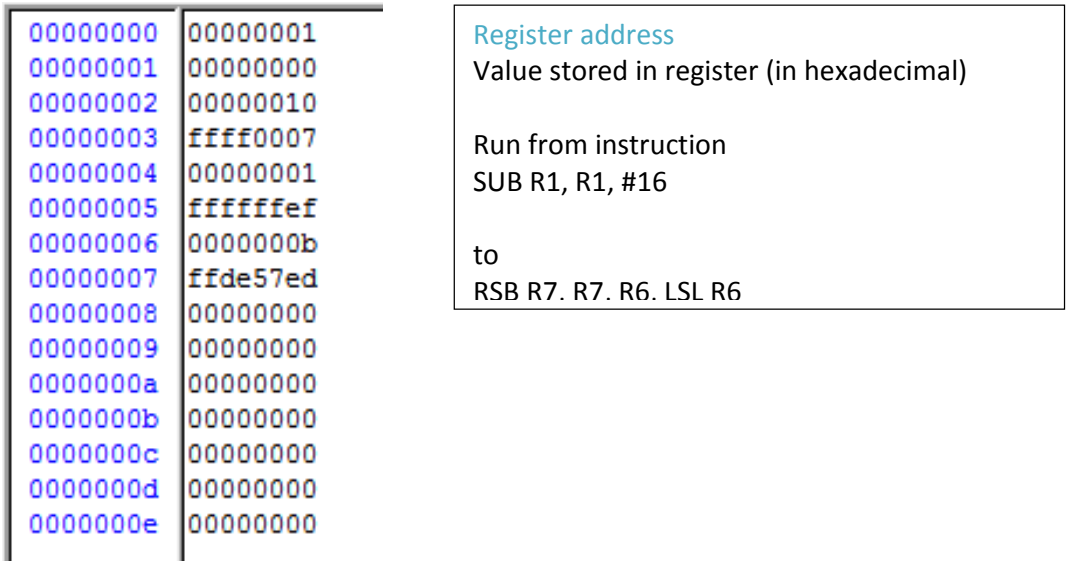
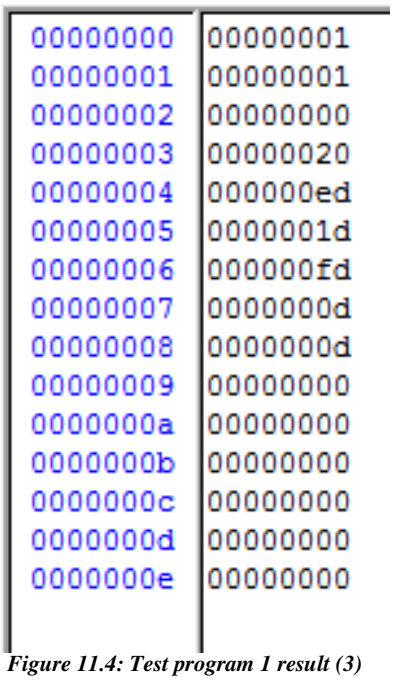


Figure 11.3: Test program 1 result (2)



00000000	00000001
00000001	00000001
00000002	00000000
00000003	00000020
00000004	000000ed
00000005	0000001d
00000006	000000fd
00000007	0000000d
00000008	0000000d
00000009	00000000
0000000a	00000000
0000000b	00000000
0000000c	00000000
0000000d	00000000
0000000e	00000000

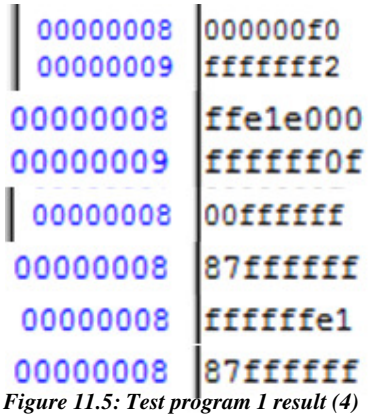
Register address
Value stored in register (in hexadecimal)

Run from instruction
SBCS R0, R0, #0

to
MOV R8, R7

Note: The TEQ, TST, CMP, CMN only affect the value of status flag which didn't show here.

Figure 11.4: Test program 1 result (3)



00000008	000000f0
00000009	fffffff2
00000008	ffe1e000
00000009	ffffff0f
00000008	00ffffff
00000008	87ffffff
00000008	fffffffe1
00000008	87ffffff

Register address
Value stored in register (in hexadecimal)

Run from instruction
MOV R8, #0xF0

to
MOV R8, R9, ROR #9

Figure 11.5: Test program 1 result (4)

For Store Memory Instruction:

From STR R9, [R2] to R8, [R2, #4]!

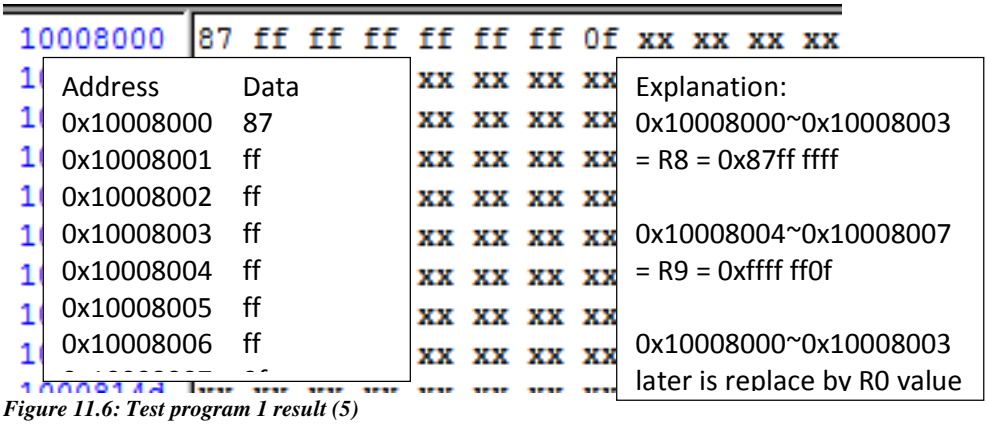


Figure 11.6: Test program 1 result (5)

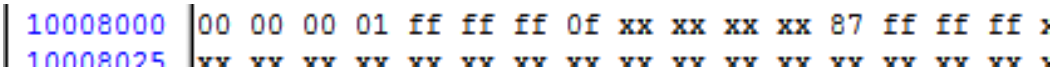


Figure 11.7: Test program 1 result (6)

The value at 0x1000 8000 ~ 0x1000 8003 is replaced by value stored in R0 (0x0000 0001) and 0x1000 800c~0x1000 8010 stored the value of 0x87ff ffff.

For load from memory:

The figures below show the result for the load instruction.

- LDR R0, R2, #-4
- LDR R0, [R2, #-4]
- LDR R0, [R2, #-8]

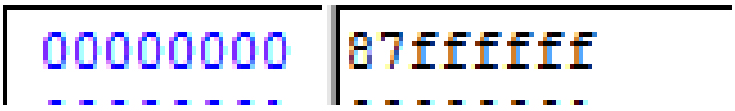


Figure 11.8: Test program 1 result (7) - Value from 0x1000 800c~0x1000 8010

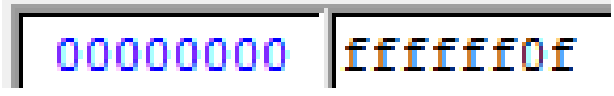


Figure 11.9: Test program 1 result (8) - Value from 0x1000 8004~0x1000 8007

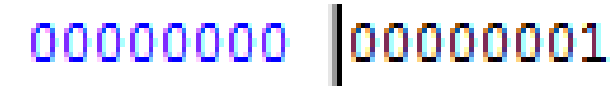


Figure 11.10: Test program 1 result (9) - Value form 0x1000 8000~0x1000 8003

For Program Flow Instruction:

The correctness of B and BL instruction is determined by the PC of the program from time to time. Figure below show the value of PC from time to time.

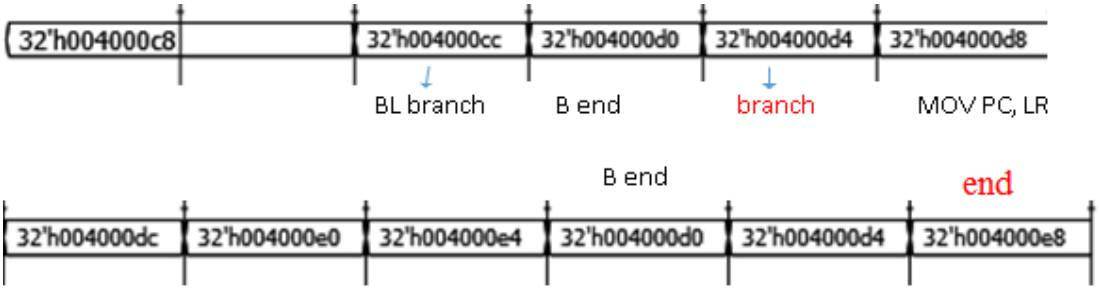


Figure 11.11: Test program 1 result (10) – program flow instruction (B & BL)

Explanation:

When BL branch execute, B end instruction is actually being fetch by CPU to IF stage, but the content of B end instruction is being flush after the BL branch done execute in ID stage.

At branch label, MOV PC, LR instruction is execute, which force the program jump back to 32x0040 00d0 which is content of LR register.

At 32x0040 00d0 B end instruction is fetch and executed and PC jump to 0x0040 00e8 which is the end of program.

11.2.3 Test program 2

The program 2 is a recursive program which converts from C program. The program use is factorial program, R0 as the output of the program R1 as the input (R0 = R1!). The multiplication is not supported by the current crisc_pipeline therefore another multiplication function is implement to the function in assembly code. Different from program 1, program 2 is full with data dependency and hazard, which mainly test the functionality and correctness of bitl_ctrl and bfw_ctrl in udp.

C program:

```
int main(){
        factorial (5);
        return 0;
}
int factorial(n){
        if(n==1) return 1;
        else n*factocrial(n-1);
}
```

Above is the content of C program.

```
_start:
        MOV     R1, #5          n!, input of factorial
        MOV     FP, SP
        ADD     FP, FP, #1024   set FP
        BL      FACT
        B       EXIT            @R3: A
FACT:
        CMP     R1, #1
        BNE     RECUR           if n != 1 branch to recur
        MOV     R0, #1          else return 1
        B       DONE            exit the program
RECUR:
        STR     R1, [SP], #4    store r1
        SUB     R1, R1, #1      n-1
        STR     LR, [FP], #4    save return address
        BL      FACT            call fact(n-1)
        LDR     LR, [FP, #-4]!  load return address
        LDR     R1, [SP, #-4]!  recall n
        MOV     R2, R0          R2 = R0
        STR     LR, [FP], #4    save return address
```

```
        BL      mult
        LDR     LR, [FP, #-4]!      load return address
DONE:
        MOV     PC, LR              continue another loop
        NOP
        NOP
        NOP
        NOP
mult:                               Multiplication (Booth algorithm)
        MOV     R6, #0              reset counter
        MOV     R3, R1, LSL #16     R3: A
        MVN     R4, R1
        ADD     R4, R4, #1          R4: -R1; -M
        MOV     R4, R4, LSL #16     R4: S
        MOV     R5, R2, LSL #1      R5: P
loop:
        CMP     R6, $15
        BEQ     done
        AND     R7, R5, #3          check last 2 bit
        CMP     R7, #2              if == 2'b10
        ADDEQ   R5, R5, R4          P = P+S
        CMP     R7, #1              if == 2'b01
        ADDEQ   R5, R5, R3          P = P+A
        MOV     R5, R5, ASR #1      P >>> 1
        ADD     R6, R6, #1          R6 ++
        B       loop
done:
        MOV     R0, R5, ASR #1      Result
        MOV     PC, LR
        NOP
        NOP
        NOP
        NOP
EXIT:
```

Table 11.2 Test program 2, with data dependency, interlock and hazard.

11.2.4 Verification on test program 2

For the factorial (5) we can know the result will be 5*4*3*2*1 = 120 = 0x78, therefore we just need to compare the value of R0 which is the final result. The operand can be change to other value for double check purpose. (Factorial (4) is run for this case, output should be 24 = 0x18)

Register value for factorial (5) (Note: R1: 0x5, R0: 0x78)

00000000	00000078
00000001	00000005
00000002	00000018
00000003	00050000
00000004	fffb0000
00000005	000000f0
00000006	0000000f
00000007	00000000
00000008	00000000
00000009	00000000
0000000a	00000000
0000000b	10008400
0000000c	00000000
0000000d	10008000
0000000e	0040001c

Figure 11.12: Test program 2 result (1) – factorial (5)

Register value for factorial (4) (Note: R1: 0x4, R0: 0x18)

00000000	00000018
00000001	00000004
00000002	00000006
00000003	00040000
00000004	fffc0000
00000005	00000030
00000006	0000000f
00000007	00000000
00000008	00000000
00000009	00000000
0000000a	00000000
0000000b	10008400
0000000c	00000000
0000000d	10008000
0000000e	0040001c

Figure 11.13: Test program 2 result (2) – factorial (4)

11.3 Verification on UART and core interaction.

A simple assembly code had developed to test the functionalities of UART after connect it to the core (data path and control path). Below shows the code which test the transmission of R1's content via UART.

Test code:

```
MOV R2, #0x0C
MOV R2, R2, LSL #28      @set R2 to value of 0xC000 0000
MOV R0, #0x98            @set the content of UARTCR
STR R0, [R2, #4]         @configure UARTCR (0xC000 0004)
MOV R1, #0xA9            @value to transmit by UART
STR R1, [R2, #8]         @store the value to transmitter FIFO (0xC000 0008)
```

In this test code, R2 is used as the pointer to UARTCR and transmitter FIFO, R0 for UARTCR's content and R1 for value to transmit.
The result is shown below.

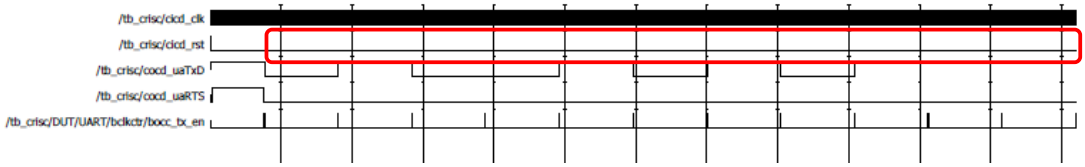


Figure 11.14: Waveform result (1)

The transmit value is 0xA9, which will send in bit stream of {start bit, d[0], d[1], …, d[7], (odd) parity bit, stop bit} (01001010111). The system clock is set to 20 MHz, while baud rate used is 38400Hz.

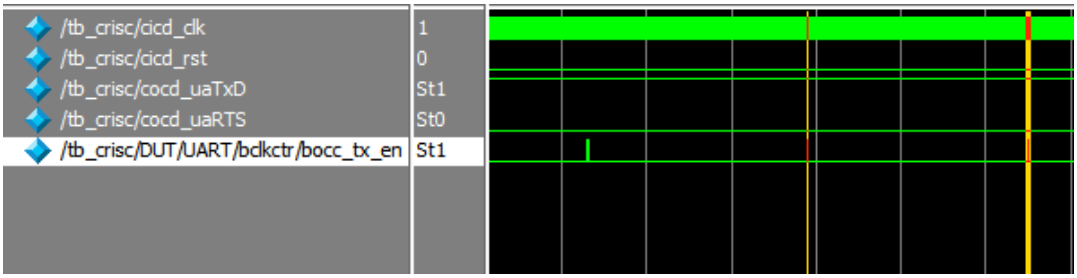


Figure 11.15: Waveform result (2)

The bocc_tx_en is enable after every 520 of system clock period, which is every 2.6×10^{-5} second of 38461Hz approximate to selected baud rate.
The content in transmitter FIFO:

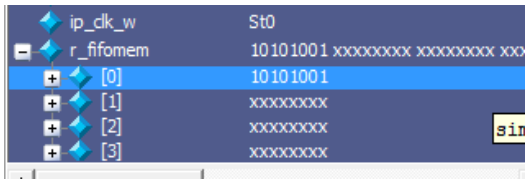


Figure 11.16: transmitter FIFO content

Chapter 12 – Conclusion

A limitation of documentation on ARM core processor especially micro-architecture of the cores on the open source project website (e.g. www.opencore.org). To present the work better the inter-connection between the blocks and functional table for each blocks are included in this report. Documentation is importance for the long term project for modification and adding feature in future. In order to achieve that, the processor is designed with the ARM ISA and a proper documentation is done.

The processor is divided to main two part which is data path and control path. Data path is designed according to the addressing mode to be implemented and Control path main designed to generate control signals for data path developed.

During the design process, several redesigns is done to improve the performance of the processor. For example, branch instructions (B or BL) done execution after 2 clock cycles (IF and ID) instead of 5 clock cycles (IF, ID, EX, MEM and WB). To solve data hazard and data dependency problem in pipelined processor, external blocks, forwarding control and interlock control are implemented to the processor.

References

Electrical Engineering (2014), Why ARM cores consumes relatively lower power than x86. Available at http://electronics.stackexchange.com/questions/74010/why-arm-cores-consumes-relatively-lower-power-than-x86. Access on 30 March 2016.

Advanced RISC Machine Ltd(ARM) (1996), ARM Architecture Reference Manual. Available at http://www.home.marutan.net/arcemdocs/ARM-ARM-RevB.pdf. Access on 30 March 2016.

ConorSantifort (2015), Amber Open Source Project-Amber 2 Core Specification March 2015[Online]. Available at www.opencore.org. Access on 16 March 2016

Stephan Nolting (2012), Storm Core Processor System [Online]. Available at www.opencore.org. Access on 16 March 2016.

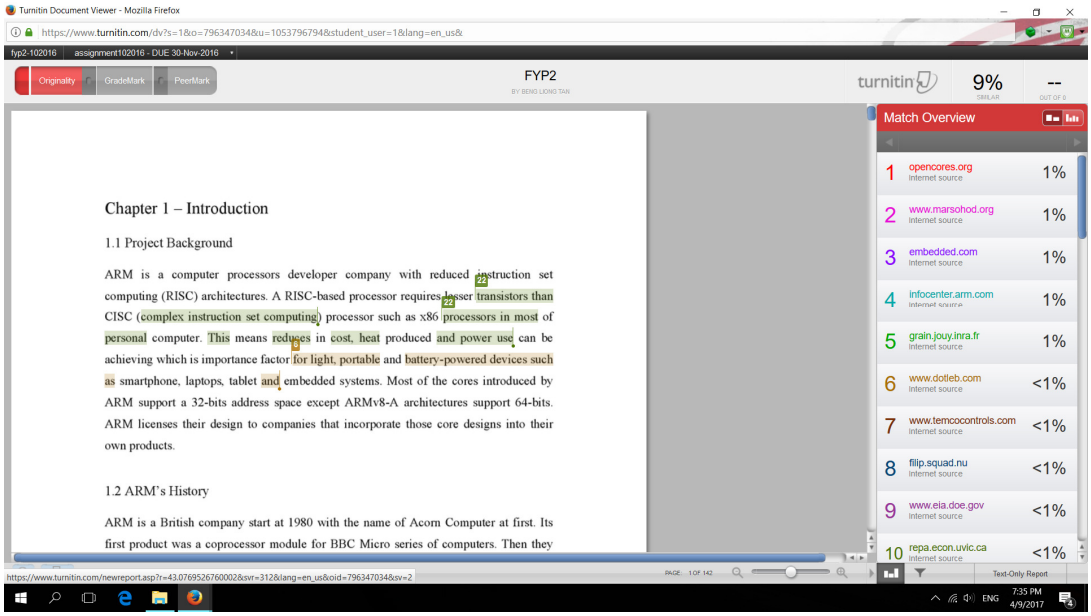
Quentin Jones (2016), ARM architecture Computer architecture M 1. History 2 Design software can be bought (Verilog) – soft core Acorn computer: An English Company Cambridge [Online]. Available at http://slideplayer.com/slide/9462345/#. Access on 20 March 2016.

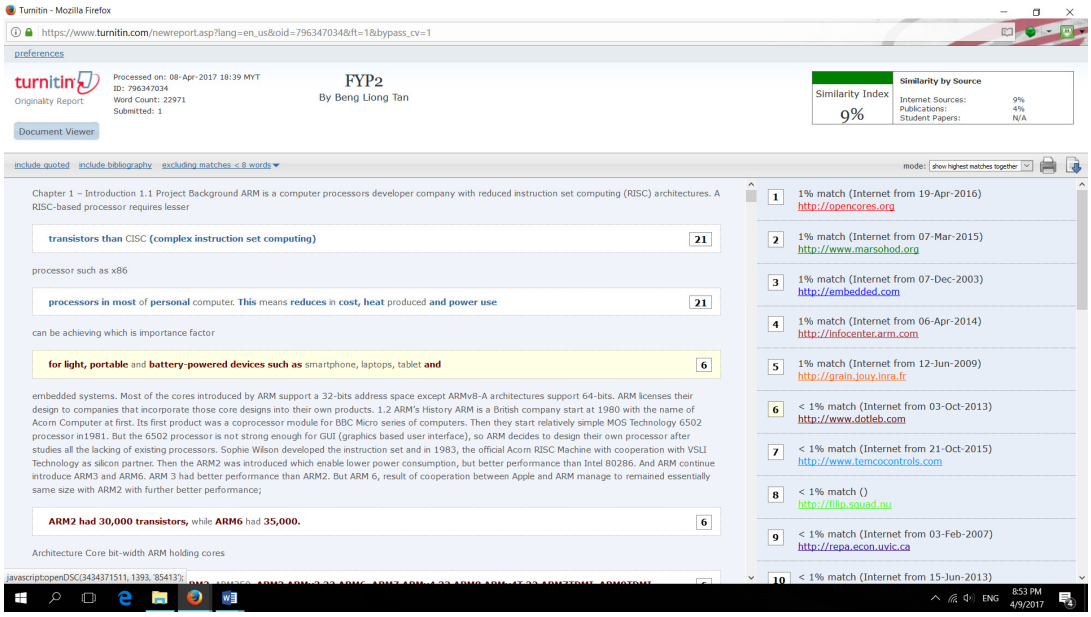
Sarah L. Harris & David Money Harris (2016) Digital Design and Computer Architecture ARM Edition, Morgan Kaumann.

Mok, K. M. Digital System Design Lecture Notes. University Tunku Abdul Rahman. Kampar : s.n., 2009. Lecture Notes.

Alvin R. Lebeck (1997). A Pipelined Processor. Duke University Durham. Available at https://www.cs.duke.edu/courses/fall98/cps104/lectures/week14-l2/sld001.htm. Access on 11 August 2016.

Appendix


Turnitin Document Viewer - Mozilla Firefox
FYP2
9%
Match Overview
Chapter 1 – Introduction
1.1 Project Background
ARM is a computer processors developer company with reduced instruction set computing (RISC) architectures. A RISC-based processor requires lesser transistors than CISC (complex instruction set computing) processor such as x86 processors in most of personal computer. This means reduces in cost, heat produced and power use can be achieving which is importance factor for light, portable and battery-powered devices such as smartphone, laptops, tablet and embedded systems. Most of the cores introduced by ARM support a 32-bits address space except ARMv8-A architectures support 64-bits. ARM licenses their design to companies that incorporate those core designs into their own products.
1.2 ARM's History
ARM is a British company start at 1980 with the name of Acorn Computer at first. Its first product was a coprocessor module for BBC Micro series of computers. Then they
1 opencores.org 1%
2 www.marsohod.org 1%
3 embedded.com 1%
4 infocenter.arm.com 1%
5 grain.jouy.inra.fr 1%
6 www.dotleb.com <1%
7 www.temcocontrols.com <1%
8 filip.squad.nu <1%
9 www.eia.doe.gov <1%
10 repa.econ.uvic.ca <1%


Turnitin - Mozilla Firefox
Originality Report
Processed on: 08-Apr-2017 18:39 MYT
ID: 796347034
Word Count: 22971
Submitted: 1
FYP2
By Beng Liong Tan
Similarity Index
9%
Similarity by Source
Internet Sources: 9%
Publications: 4%
Student Papers: N/A
Document Viewer
Chapter 1 – Introduction 1.1 Project Background ARM is a computer processors developer company with reduced instruction set computing (RISC) architectures. A RISC-based processor requires lesser
transistors than CISC (complex instruction set computing) 21
processor such as x86
processors in most of personal computer. This means reduces in cost, heat produced and power use 21
can be achieving which is importance factor
for light, portable and battery-powered devices such as smartphone, laptops, tablet and 6
embedded systems. Most of the cores introduced by ARM support a 32-bits address space except ARMv8-A architectures support 64-bits. ARM licenses their design to companies that incorporate those core designs into their own products. 1.2 ARM's History ARM is a British company start at 1980 with the name of Acorn Computer at first. Its first product was a coprocessor module for BBC Micro series of computers. Then they start relatively simple MOS Technology 6502 processor in1981. But the 6502 processor is not strong enough for GUI (graphics based user interface), so ARM decides to design their own processor after studies all the lacking of existing processors. Sophie Wilson developed the instruction set and in 1983, the official Acorn RISC Machine with cooperation with VSLI Technology as silicon partner. Then the ARM2 was introduced which enable lower power consumption, but better performance than Intel 80286. And ARM continue introduce ARM3 and ARM6. ARM 3 had better performance than ARM2. But ARM 6, result of cooperation between Apple and ARM manage to remained essentially same size with ARM2 with further better performance;
ARM2 had 30,000 transistors, while ARM6 had 35,000. 6
Architecture Core bit-width ARM holding cores
1 1% match (Internet from 19-Apr-2016) http://opencores.org
2 1% match (Internet from 07-Mar-2015) http://www.marsohod.org
3 1% match (Internet from 07-Dec-2003) http://embedded.com
4 1% match (Internet from 06-Apr-2014) http://infocenter.arm.com
5 1% match (Internet from 12-Jun-2009) http://grain.jouy.inra.fr
6 < 1% match (Internet from 03-Oct-2013) http://www.dotleb.com
7 < 1% match (Internet from 21-Oct-2015) http://www.temcocontrols.com
8 < 1% match () http://filip.squad.nu
9 < 1% match (Internet from 03-Feb-2007) http://repa.econ.uvic.ca
10 < 1% match (Internet from 15-Jun-2013)

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	
ID Number(s)	
Programme / Course	
Title of Final Year Project	

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index:_____ % **Similarity by source** Internet Sources: _____% Publications: _____ % Student Papers:_____ %	
Number of individual sources listed of more than 3% similarity: _____	
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

_____ Signature of Supervisor

Name: _____

Date: _____

_____ Signature of Co-Supervisor

Name: _____

Date: _____