# Program Control Instruction Class of x86 Instruction Set Architecture Compatible CPU

**NG TEIK HUAT**

**A project report submitted in partial fulfilment of the**
**requirements for the award of the degree of**
**Master of Engineering (Electrical – Computer & Microelectronic System)**

**Faculty of Electrical Engineering**
**Universiti Teknologi Malaysia**

**JUNE 2014**

Dedicated, in thankful appreciation for support, encouragement and understandings to my beloved mother, father, brothers and sisters.

# ACKNOWLEDGEMENT

In order to achieve everything that I have done for my final year project, I received a lot of helps from many people along the way. They have guided me, advised me and encouraging me.

First and foremost, I would like to express my heartily gratitude to my supervisor, Prof. Dr. Mohamed Khalil bin Hj. Mohd Hani for the guidance and enthusiasm given throughout the progress of this project.

My appreciation also goes to my family and all my beloved friends and MEH course mate who has been so tolerant and supports me along the year. Thanks for their encouragement, love, emotional and financial supports that they had given to me.

Nevertheless, my great appreciation dedicated to Faculty of Electrical Engineering, UTM for the environment and material support and encouragement throughout the project.

# ABSTRACT

Digital system designers nowadays facing the challenge of need to come out new design or improved design fast such as IP core or interface system that will be embedded into System-On-Chip. Rapid prototyping requires platform for development design and testing of any digital system. A platform that is flexible to fine tune where the source code or the internal design is visible and accessible by designer is preferable to increasing the opportunity for design exploration. Technology leading company such as Intel, Altera, and Xilinx have these platform ready but with limited access to internal design and source code as it is their company IP and trade secret.

Currently most digital system was design and implemented on FPGA before goes into production. The design in hardware description language like Verilog HDL was compiled into physical netlists using compiler tools such as Synopsys, Altera Quartus II and consume by the FPGAs, thereby reducing the design cycle while increasing the opportunity for design exploration.

# ABSTRAK

Rekaan sistem digital hari kini menghadapi cabaran kritikal disebabkan dunia reka bentuk dalam era baru ini mengunakan cara IP untuk membentuk sistem baharu dengan secepat mungkin dalam bentuk SOC. Prototaip perlu dibentuk secepat mungkin untuk membolehkan kajian, eksperimen dan analisis dijalankan untuk sistem baru. Sebuah platform yang fleksibel untuk diubahsuai di mana kod wujud dan boleh diakses adalah keperluan untuk memudah and mempercepatkan proses mencipta sistem digital baru. Syarikat teknologi terkemuka seperti Intel, Altera, dan Xilinx mempunyai platform ini bersedia tetapi dengan akses terhad kepada reka bentuk dalaman dan kod kerana ia adalah rahsia perniagaan.

Pada masa kini kebanyakan ciptaan sistem digital baru melalui FPGA sebelum ke pengeluaran sebenar. Reka bentuk dalam kod bahasa seperti Verilog HDL menggunakan program seperti Synopsys, Altera Quartus II dan keluaran alat-alat ini diguna oleh FPGAs , ini akan mengurangkan kitaran reka bentuk di samping meningkatkan peluang untuk penerokaan reka bentuk baru.

# TABLE OF CONTENT

# LIST OF TABLES

# LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | | |
|---|---|---|
| CPU | - | Central Processing Unit |
| ISA | - | Instruction Set Architecture |
| RISC | - | Reduced Instruction Set Computer |
| CISC | - | Complex Instruction Set Computer |
| ILD | - | Instruction Length Decoder |
| BPB | - | Branch Prediction Buffer |
| ASM | - | Algorithmic State Machine |
| NextPC | - | Next Program Count Control Unit |
| PC | - | Program Counter |
| BFM | - | Bus Functional Model |
| RAPPID | - | Revolving Asynchronous Pentium Processor Instruction Decoder |
| HAT | - | Heads and Tails |

# LIST OF APPENDICES

# CHAPTER 1

# INTRODUCTION

## 1.1 Background

Central processing unit (CPU) is a microprocessor that served as the Heart of computer. Microprocessors are regarded as one of the most important devices in modern worlds as more and more electronics gadgets are depending on it. CPU is the hardware unit in a computer that carries out the instruction of a computer program such as logical operation or arithmetic operation which in other words providing computational ability and control. CPU nowadays also referred as core since this important unit of a computer is now embedded with others important Intellectual Property (IP) unit in a single chip in a form called System-On-Chip (SOC).

Typical CPU are served for general purpose computation which support a wide range of general arithmetic and logic operation unlike others dedicated processor which design and optimized for certain function or algorithm. CPU are usually incorporate arithmetic and logic functional units as well as the associated control logic, instruction processing circuitry, and a portion of the memory hierarchy known as Cache that is to enhanced the speed of memory Read and Write operation from and to memory device such as RAM. Portions of the interface logic for the input/output (I/O) and memory subsystems may also be infused.

Instruction set Architecture (ISA) defines how software programmers communicates with the hardware in CPU from the perspective of a software programmers, mean how hardware response to certain code in software. . In this

project, we will focus on the x86 ISA compatible design. x86 ISA is a Complex Instruction Sets Computer (CISC), extra effort required in instruction decode compare to RISC as their instruction is more complex in term of variable length and format, while RISC is always fixed length instruction in general. RISC ISA support only simple addressing modes, memory access only through two instruction, load and store, but x86's support multiple complex addressing modes, many instructions have memory access flow.

In this project, the exploration is focus on instruction fetch and pre-decode unit of the x86 ISA design. Zet processor which is an open implementation of the x86 ISA also known as IA-32 (Intel Architecture-32) available in opencore.org is use as a reference in exploring the design of instruction fetch and pre-decode unit of an IA-32 core. Zet is synthesizable and has been FPGA proven, currently four different FPGA boards are supported which are Xilinx ML-403, Altera DE0, Altera DE1 and Altera DE2-115 boards.

## 1.2 Problem Statement

Digital system designers nowadays facing the challenge of need to come out new design or improved design fast such as IP core or interface system that targeted for embedded into System-On-Chip. Rapid prototyping requires platform for development design and testing of any digital system. The platform in this context refer to a processor core that able to attached with any of our newly designed co-processor or an interface system or others IP to test its functionality and performance in an environment such as Field-Programmable Gate Array (FPGA). A platform that is flexible to fine tune and open to access where the source code or the internal design that is visible and accessible by designer is preferable where it easier for debug and increase the opportunity for design exploration. Intel processor that implement x86 ISA also known as Intel Architecture (IA) which is widely use in the world seems to lack of x86 ISA compatible platform that is open or free for research and development work currently.

Platform provider from leading company available now such as Intel (Quark), Xilinx (Micro-Blaze) and Altera (Nios) are not open source. Their internal designs are IP and Trade Secret that is not for sale or available for public affordable price. Hence, preventing researchers to access into the core design to make detail monitoring or debugging and modification while using this platform to do research work. Open source platform available in internet are usually lack of proper documentation and might be incomplete. While a good documentation is very important to provide good confidence level to the designer when using the platform and also the documentation affect the reusability of the design and also the support of extending the design. Extra effort will be requires from the designer that using the platform for debugging the design when encounter any issue. Zet processor is x86 ISA compatible processor where its design source codes are freely available in OpenCore.org. Although it has proven its functionality through FPGA emulation but the design methodology and specification are not proper documented giving us hard time to understand the design purely through analyzing the design source code.

The Zet processor design does not pipeline, which is probably due to the issue with CISC type ISA of variable length for instruction that requires effort to pre-decode the instruction length during instruction fetch stages before actual decoding the opcodes and functions of the instructions. Zet implement the instruction length decoder with a traditional instruction length decoder, a state machine that analyze byte per byte of the instruction bytes chunk fetch. This method causes the cycles requires for pre-decode is inconsistent with the variable of length of instructions and make it hard to be pipeline.

## 1.3 Objectives

Our main objective is to design an X86 ISA processor. The processor design was separate to two parts, program control instruction class and arithmetic instruction class. In this project, the program control instruction class was handled. The objective in this project is:

1. To analyze the Zet processor design of it instruction fetch control and

program control instruction class unit.

2. To modify the instruction fetch stages such a way that enable pipeline design.
3. To develop verification environment such as test benches and bus functional model (BFM) serve to analyze, verify and bench marking the design with proper documentation.

## 1.4 Scope

This project is one part of a large project where the final goal is to design a x86 ISA cpu. The core design is separated into two parts. First part, program control instruction class which is responsible to handle the program control, PC and all program control related instruction such as conditional jump and subroutine call also including the instruction fetch and pre-decode of x86 instruction set. Second part, the arithmetic instruction class where this portion will be handling all logical and arithmetic operation and data memory access of all related instructions such as mov, add, store and load. In this project my scope is to design the program control instruction class that is x86 ISA compatible. The arithmetic instruction class will be handle by my counterpart.

In details, the program control instruction class includes the instruction fetch unit and program control instructions such as branch and jump handling unit which can be decomposed into three main block of design which is the Instruction Pre-decoder unit that decodes the instruction length, the Branch Prediction Unit to improve conditional branch type instruction handling in pipeline design, and the next instruction address (PC) computation unit.

The scope of design specification will support only fix opcode length one byte and prefix are not supported. The data size of the processor supported is 16bits. The instruction bytes chunk are assume readily in instruction cache when processor issue instruction fetch and no extra waiting time for cache to get data from RAM. The program control instruction class will be responsible to provide pre-decoded instruction and any control signals requires to the arithmetic instruction class which is

handled by another person at the same time. The design will be modeled in Verilog HDL and using synthesis tool Altera Quartus II SE then simulate with Modelsim-Altera.

## 1.5 Summary of works

This project aim to enable the pipelining base on Zet processor designs which then requires modification in instruction fetch and pre-decode unit so that is fulfil the requirement for pipeline design. This project does a one cycle instruction length decoder which realize using lookup table method to resolve the issue of inconsistent variable cycles to pre-decode instruction length that gating pipeline design. To enable pipelining, the program control instruction handling will need to modify accordingly such as stall the stages when pending flag status and check flag only when flag register data is updated for condition jump instruction. To further improve the performance, a branch prediction unit is design with simple 2 bit local branch predictor to predict for conditional branch instruction outcome. Finally a comparison between Zet processor instruction fetch and pre-decode performance with this project design is made and reported. With the enable of pipeline, higher throughput can be achieved versus the original Zet processor design.

# REFERENCE


1. G. Antonio, L. Fernando and M. Grigorios, *Processor Microarchitecture: An Implementation Perspective*, Morgan & Claypool, 2011.
2. C. S. SU, *Implementation of Instruction Decoding Logic Using Hardware and Software*, Faculty of Electrical Engineering, Universiti Teknologi Malaysia.
3. Intel Corporation, *Intel® 64 and IA-32 Architecture Software Developer's Manual Vol.1 Basic Architecture*, Intel Corporation.
4. A. Donald and A. Dror, *Architecture of Pentium Processor*, 1993..
5. G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, P. Roussel, *The Microarchitecture of the Pentium 4 Processor*, Desktop Platforms Group, Intel Corp., 2001.
6. T. Shanley, *Pentium Pro and Pentium II System Architecture*, MindShare, Inc., 1997.
7. T. Shanley, *x86 Instruction Set Architecture*, MindShare, Inc., 2009.
8. A. Fog, *Instruction tables Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs*, Technical University of Denmark, 2013.
9. Mamun Bin Ibne Reaz, Md. Shabiul Islam, Md. S. Sulaiman, *A Single Clock Cycle MIPS RISC Processor Design using VHDL*, Faculty of Engineering, Multimedia University, 2002.
10. K. S. Stevens, et al., *RAPPID: An Asynchronous Instruction Length Decoder*, Strategic CAD Lab, Intel Corporation, 2001.

11. P. Heidi and A. Krste, *Heads and Tails: A VariableLength Instruction Format Supporting Parallel Fetch and Decode*, MIT Laboratory for Computer Science, 2001

12. A. Fog, *The microarchitecture of Intel, AMD and VIA CPUs*, Technical University of Denmark, 2014.

13. W. Andrew and C. Alex, *Executing Compressed Programs on An Embedded RISC Architecture*, Department of Electrical Engineering, Princeton University, 1992.

14. V. R. Madduri, *Instruction Length Decoder*, U.S. Patent 7 640 417 B2, Dec. 29, 2009.

15. S. Hauck, *Asynchronous design methodologies: An overview*, Proc. IEEE, vol. 83, pp. 69–93, Jan. 1995.

16. K. S. Stevens, et al., *CAD directions for high-performance asynchronous circuits*, in Proc. Digital Automation Conf. (DAC'99), June 1999, pp. 116–121.

17. W. Chou, et al., *Average-case optimized technology mapping of one-hot domino circuits*, in Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems, 1998, pp. 80–91.

18. D. Kearney, *Theoretical limits on the data dependent performance of asynchronous circuits*, in Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems, Apr. 1999, pp. 201–207.

19. M. Roncken, et al., *CA-BIST for asynchronous circuits:A case study on the RAPPID asynchronous instruction length decoder*, in Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems, 2000, pp. 62–72.

20. C. J. Myers, *Computer-aided synthesis and verification of gate-level timed circuits*, Ph.D. dissertation, Dept. of Electr. Eng., Stanford Univ., Stanford, CA, Oct. 1995.

21. K. Y. Yun and A. E. Dooply, *Optimal evaluation clocking of self-resetting domino pipelines*, Proc. 1999 Asia and South Pacific Design Automation Conference, Jan. 1999, Hong Kong, pp. 121-124.

22. J. S. Coke, et al., D*etermining Length of Instruction with Address Form Field Exclusive of Evaluating Instruction Specific Opcode in Three Byte Escape Opcode*, U.S. Patent 8 402 252 B2, Mar. 19, 2013

23. C. Lefurgy et al., *Improving code density using compression techniques*, in MICRO-30, pages 194–203, Research Triangle Park, North Carolina, December 1997.

24. G. Araujo et al., *Code compression based on operand factorization*, in MICRO-31, pages 194–201, December 1998.

25. V. Narayanan, B.A. Chappell and B.M. Fleischer, *Static timing analysis for self-resetting circuits*, Proc. Int. Conf. Computer Aided Design (ICCAD), 1996.

26. H. Zheng, *Specification and compilation of timed systems*, Master's thesis, University of Utah, 1998.

27. G. M. Zeus, *Zet - The x86 (IA-32) open implementation,* http://zet.aluzina.org, May 5, 2014.