A High speed 16-bit RISC processor chip

Wan-Fu Chen

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
A HIGH SPEED 16-bit RISC PROCESSOR CHIP

by

Wan-Fu Chen

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by: ________________________________
Graduate Advisor - Kenneth W. Hsu, Associate Professor

______________________________
Tony H. Chang, Professor

______________________________
Roy S. Czernikowski, Professor and Department Head

Department of Computer Engineering
College of Engineering
Rochester Institute of Technology
Rochester, New York

May, 1994
This document was produced using Windows version 3.1, Microsoft Word version 2.0 for Windows, and Windows Paintbrush. All circuits are built by using Design Architecture software. The simulation of the circuits was done by using Quicksim II and Accusim software, and the chip design was done by using IC program.

The following names used here and in the remainder of the document are registered trademarks of the respective companies:

Windows Microsoft Corporation,
Microsoft Word Microsoft Corporation,
Paintbrush Microsoft Corporation,
Design Architecture Mentor Graphics Corporation,
Quicksim II Mentor Graphics Corporation,
Accusim Mentor Graphics Corporation,
IC Mentor Graphics Corporation.

Copyright (C) 1994 by Wan-Fu Chen
All rights reserved
Title: A High Speed 16-bit RISC Processor

I, Wan-fu Chen, hereby grant permission to the Wallace Memorial Library of RIT to reproduce my thesis in whole or in part.

Signature: ____________________________

Date: 17 June 94
ABSTRACT

The goal of this thesis is to design and simulate a high speed 16-bit processor chip by using RISC architecture. The high computing speed is achieved by employing a more effective four-stage pipeline. This processor executes every instruction in one clock cycle, and it won't have any delay of executing instructions when it executes Jump, Condition Jump, Call, and Return instructions. Its computing speed is 4 times faster than the speed of the Berkeley RISC II's for the 8-MHz clock. The design includes the main architectural features of the RISC: the 4-stage pipeline, the thirty-two 8-bit register bank, the 16-bit address and data paths, the internal timer, the input port, and the two output ports.

The chip is designed using 2µ CMOS N well two metal layer technology. The processor runs at a clock rate of 16 MHz. The size of the chip is 10535µm by 14677µm. It consists of 24,982 transistors and consumes 200mw.
# Table Of Contents

Abstract...................................................................................................................... i
Table of Contents...................................................................................................... ii
List of Figure.............................................................................................................. v
List of Tables............................................................................................................. vii
Glossary..................................................................................................................... ix

## Chapter 1 Introduction

1.1 16-bit Architecture................................................................................................. 2
1.2 Superpipeline Architecture...................................................................................... 2
1.3 System Interface..................................................................................................... 2
1.4 CPU Register Overview......................................................................................... 4
1.5 CPU Instruction Set Overview............................................................................... 4

## Chapter 2 CPU Instruction Set Details

- ADDC ..................................................................................................................... 10
- ADDI ....................................................................................................................... 11
- SUBC ...................................................................................................................... 12
- SUBI ....................................................................................................................... 13
- AND ....................................................................................................................... 14
- ANDI ..................................................................................................................... 15
- OR .......................................................................................................................... 16
- ORI ....................................................................................................................... 17
- XOR ....................................................................................................................... 18
- XORI ...................................................................................................................... 19
- RRA ...................................................................................................................... 20
- RLA ....................................................................................................................... 21
- JUMP ..................................................................................................................... 22
- JC .......................................................................................................................... 23
- JE .......................................................................................................................... 24
- CMPI ..................................................................................................................... 25
- LDAI ...................................................................................................................... 26
- LDA ....................................................................................................................... 27
- STA ......................................................................................................................... 28
- LDMI ..................................................................................................................... 29
- OUT1 ..................................................................................................................... 30
- OUT2 ..................................................................................................................... 31
- IN .......................................................................................................................... 32
- CALL ...................................................................................................................... 33
- RET ....................................................................................................................... 34
- NOP ....................................................................................................................... 35
Chapter 8  Logic Circuits

8.1 D flip-flop .................................................. 113
8.2 16-bit Register I .......................................... 113
8.3 16-bit Register II ......................................... 113
8.4 8-bit Register I ........................................... 113
8.5 8-bit Register II ........................................... 114
8.6 6-bit Register .................................................. 114
8.7 4-bit Register .................................................. 114
8.8 Instruction Register ........................................ 114
8.9 MEM32 .......................................................... 114
8.10 Accumulator .................................................. 115
8.11 PSW Circuit .................................................. 115
8.12 IT Circuit .................................................... 115
8.13 MEM16 .......................................................... 116
8.14 JK flip-flop .................................................... 116
8.15 16-bit Counter ............................................. 116
8.16 8-bit Counter ................................................ 116
8.17 Program Counter ........................................... 116
8.18 Timer Circuit ................................................ 117
8.19 Full Adder, XOR, AND, and OR Circuit ............... 117
8.20 16-bit Adder and Subtractor .............................. 117
8.21 8-bit Adder and Subtractor .............................. 117
8.22 Stack Pointer ............................................... 117
8.23 ALU ............................................................... 118
8.24 SUB-1 ........................................................... 118
8.25 Control Unit ............................................... 118
8.26 16-bit RISC Processor ..................................... 119

Chapter 9  VLSI Design

9.1 VLSI Design .................................................. 158
9.2 Delay Time of Each Circuit ............................... 158
9.3 Speed Up the Processor ................................... 158

Conclusion ......................................................... 169
References ......................................................... 170
List Of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Processor Internal Block Diagram</td>
<td>3</td>
</tr>
<tr>
<td>1-2</td>
<td>CPU Instruction Formats</td>
<td>4</td>
</tr>
<tr>
<td>2-1</td>
<td>CPU Instruction Formats</td>
<td>9</td>
</tr>
<tr>
<td>2-2</td>
<td>Clearing the extending bit program</td>
<td>50</td>
</tr>
<tr>
<td>2-3</td>
<td>Setting the extending bit program</td>
<td>50</td>
</tr>
<tr>
<td>3-1</td>
<td>Instruction Pipeline Stage</td>
<td>51</td>
</tr>
<tr>
<td>3-2</td>
<td>A example of pipeline</td>
<td>50</td>
</tr>
<tr>
<td>4-1</td>
<td>Program Status Word Register</td>
<td>61</td>
</tr>
<tr>
<td>5-1</td>
<td>RAM read/write Circuit Internal Block Diagram</td>
<td>67</td>
</tr>
<tr>
<td>6-1</td>
<td>Writing Timing Diagram</td>
<td>67</td>
</tr>
<tr>
<td>6-2</td>
<td>Reading Timing Diagram</td>
<td>81</td>
</tr>
<tr>
<td>7-1</td>
<td>Test Program 01-02</td>
<td>82</td>
</tr>
<tr>
<td>7-2</td>
<td>Test Program 11-12</td>
<td>83</td>
</tr>
<tr>
<td>7-3</td>
<td>Test Program 13-14</td>
<td>84</td>
</tr>
<tr>
<td>7-4</td>
<td>Test Program 15-16</td>
<td>85</td>
</tr>
<tr>
<td>7-5</td>
<td>Test Program 17-18</td>
<td>86</td>
</tr>
<tr>
<td>7-6</td>
<td>Test Program 19-20</td>
<td>87</td>
</tr>
<tr>
<td>7-7</td>
<td>Test Program 21-22</td>
<td>89</td>
</tr>
<tr>
<td>7-8</td>
<td>Simulation Result for Test Program 01</td>
<td>90</td>
</tr>
<tr>
<td>7-9</td>
<td>Simulation Result for Test Program 02</td>
<td>91</td>
</tr>
<tr>
<td>7-10</td>
<td>Simulation Result for Test Program 03</td>
<td>92</td>
</tr>
<tr>
<td>7-11</td>
<td>Simulation Result for Test Program 04</td>
<td>93</td>
</tr>
<tr>
<td>7-12</td>
<td>Simulation Result for Test Program 05</td>
<td>94</td>
</tr>
<tr>
<td>7-13</td>
<td>Simulation Result for Test Program 06</td>
<td>95</td>
</tr>
<tr>
<td>7-14</td>
<td>Simulation Result for Test Program 07</td>
<td>96</td>
</tr>
<tr>
<td>7-15</td>
<td>Simulation Result for Test Program 08</td>
<td>97</td>
</tr>
<tr>
<td>7-16</td>
<td>Simulation Result for Test Program 09</td>
<td>98</td>
</tr>
<tr>
<td>7-17</td>
<td>Simulation Result for Test Program 10</td>
<td>99</td>
</tr>
<tr>
<td>7-18</td>
<td>Simulation Result for Test Program 11</td>
<td>100</td>
</tr>
<tr>
<td>7-19</td>
<td>Simulation Result for Test Program 12</td>
<td>101</td>
</tr>
<tr>
<td>7-20</td>
<td>Simulation Result for Test Program 13</td>
<td>102</td>
</tr>
<tr>
<td>7-21</td>
<td>Simulation Result for Test Program 14</td>
<td>103</td>
</tr>
<tr>
<td>7-22</td>
<td>Simulation Result for Test Program 15</td>
<td>104</td>
</tr>
<tr>
<td>7-23</td>
<td>Simulation Result for Test Program 16</td>
<td>105</td>
</tr>
<tr>
<td>7-24</td>
<td>Simulation Result for Test Program 17</td>
<td>106</td>
</tr>
</tbody>
</table>
Figure 7-18-2 Simulation Result for Test Program 18 ..................................................107
Figure 7-19-2 Simulation Result for Test Program 19 ..................................................108
Figure 7-20-2 Simulation Result for Test Program 20 ..................................................109
Figure 7-21-2 Simulation Result for Test Program 21 ..................................................110
Figure 7-22-2 Simulation Result for Test Program 22 ..................................................111
Figure 8-1 D flip-flop ........................................................................................................121
Figure 8-2 16-bit Register I .............................................................................................122
Figure 8-3 16-bit Register II ............................................................................................123
Figure 8-4 8-bit Register I ...............................................................................................124
Figure 8-5 8-bit Register II ..............................................................................................125
Figure 8-6 6-bit Register .................................................................................................126
Figure 8-7 4-bit Register .................................................................................................127
Figure 8-8 Instruction Register .......................................................................................128
Figure 8-9 MEM32 ..........................................................................................................129
Figure 8-10 Accumulator .................................................................................................130
Figure 8-11 PSW Register ...............................................................................................131
Figure 8-12 INT and Timer Circuit ..................................................................................132
Figure 8-13 MEM16 (Stack) ............................................................................................133
Figure 8-14 Jk flip-flop .....................................................................................................134
Figure 8-15 16-bit Counter ...............................................................................................135
Figure 8-16 8-bit Counter .................................................................................................136
Figure 8-17 Program Counter ..........................................................................................137
Figure 8-18 Timer Circuit .................................................................................................138
Figure 8-19 Full Adder ......................................................................................................139
Figure 8-20 XOR ...............................................................................................................140
Figure 8-21 AND ..............................................................................................................141
Figure 8-22 OR ................................................................................................................142
Figure 8-23 16-bit Adder and Subtracter .........................................................................143
Figure 8-24 8-bit Adder and Subtracter ..........................................................................144
Figure 8-25 Stack Pointer ...............................................................................................145
Figure 8-26 ALU ...............................................................................................................146
Figure 8-27 SUB-1 ..........................................................................................................147
Figure 8-28 Control Unit ..................................................................................................148
Figure 8-29 EX1 Stage of the Control Unit .....................................................................149
Figure 8-30 EX2 Stage of the Control Unit .....................................................................150
Figure 8-31 EX3 Stage of the Control Unit .....................................................................151
Figure 8-32 16-bit RISC Processor ................................................................................152
Figure 8-33 Part I of the Processor ................................................................................153
Figure 8-34 Part II of the Processor ................................................................................154
Figure 8-35 Part III of the Processor ...............................................................................155
Figure 8-36 Part IV of the Processor ...............................................................................156
Figure 9-1 Inverter Circuit ..............................................................................................161
Figure 9-2 Tri-State Inverter ................................................................. 162
Figure 9-3 Three-input NOR ............................................................... 163
Figure 9-4 Two-input NOR ................................................................. 164
Figure 9-5 Six-input NAND ................................................................. 165
Figure 9-6 Six-input NOR ................................................................. 166
Figure 9-7 Three-input NAND ......................................................... 167
Figure 9-8 Two-input NAND ......................................................... 168
List of Tables

Table 1-1 CPU Instruction Set: Computational Instructions ........................................... 6
Table 1-2 CPU Instruction Set: Jump and Brand Instructions ....................................... 7
Table 1-3 CPU Instruction Set: Transfer Instructions ...................................................... 7
Table 1-4 CPU Instruction Set: Load Instructions ........................................................... 8
Table 1-5 CPU Instruction Set: Special Instructions ......................................................... 8
Table 5-1 Status Register Fields ...................................................................................... 61
Table 9-1 Delay Time of Each Circuit ............................................................................. 159
<table>
<thead>
<tr>
<th><strong>Glossary:</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Arithmetic Logic Unit</td>
</tr>
<tr>
<td>CISC</td>
<td>Complex Instruction Set Computer</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal-Oxide-Semiconductor</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>HP</td>
<td>Higher-order 8-bit Register for Program Counter</td>
</tr>
<tr>
<td>IR</td>
<td>Instruction Register</td>
</tr>
<tr>
<td>PSW</td>
<td>Program Status Word</td>
</tr>
<tr>
<td>RAM</td>
<td>Random-Access Memory</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced Instruction Set Computer</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale Integration</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The RISC (Reduced Instruction Set Computer) Architecture is a recent trend in computer design. It just has many fewer instructions than the CISC (Complex Instruction Set Computer) does, so the RISC architecture reduces much space on the control area but tends to utilize many internal registers [1]. Because the architecture uses pipeline technology and has a simpler and smaller control system, its computing speed is generally faster than CISC. The purpose of this thesis is to implement a high speed 16-bit processor chip by using RISC architecture and also increase the computing speed by using more stages of a pipeline. This chip can execute an instruction in each clock cycle. For example, if the frequency of the clock is 1 MHz, the chip can execute one million instructions per second. This chip's computing speed is 4 times faster than the RISC II's for an 8-MHz clock.

This chapter describes the following:

- the 16-bit architecture of the processor
- the superpipeline design of the processor (described in detail in chapter 3)
- an overview of the system interface (described in detail in chapter 4)
- an overview of the CPU registers (described in detail in chapter 5)
- an overview of the CPU instruction set (described in detail in chapter 2)
1.1 16-bit Architecture

The natural mode of operation for the processor is as a 16-bit microprocessor; however, all registers are 8-bit registers. The processor provides the following:

- 8-bit integer arithmetic logic unit (ALU)
- 8-bit integer registers
- 8-bit I/O registers
- 16-bit stack registers

Figure 1-1 is a block diagram of the processor internal.

1.2 Superpipeline Architecture [2]

The processor exploits instruction parallelism by using a four-stage superpipeline. Under normal circumstances, one instruction is executed each clock cycle.

1.3 System Interface

The processor supports a 16-bit and an 8-bit system interface. The System Interface includes:

- a 16-bit address bus
- a 16-bit data bus (input direction)
- two 8-bit output ports
- an 8-bit input port
- an interrupt control pin
- a reset pin

1.4 CPU Register Overview

The central processing unit (CPU) provides the following registers:

- 32 general purpose registers
Figure 1-1 Processor Internal Block Diagram
• a Program Counter (PC)
• a 8-bit Timer
• 16 stack registers
• a Program Status Word (PSW) register
• an Accumulator register
• a Higher-order 8-bit Register for the program counter (HP)

1.5 CPU Instruction Set Overview

Each CPU instruction is 16 bits long. As shown in Figure 1-2, there are three instruction formats:

• immediate (I-type)
• only opcode (O-type)
• register (R-type)

<table>
<thead>
<tr>
<th>I-Type (Immediate)</th>
<th>OP</th>
<th>XX</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>O-Type(Only opcode)</td>
<td>OP</td>
<td>XXXXXXXXXX</td>
<td></td>
</tr>
<tr>
<td>R-Type (Register)</td>
<td>OP</td>
<td>XX</td>
<td>register</td>
</tr>
</tbody>
</table>

XX: Don't care

Figure 1-2 CPU Instruction Formats

Each format contains a number of different instructions, which are described further in this chapter. Fields of the instruction formats are described in Chapter 2.
The instruction set can be further divided into the following groupings:

- **Transfer** instructions move data between the accumulator register and registers. They are all register (R-type) instructions.

- **Computational** instructions perform arithmetic, logical, and shift operations on value in registers. They include register (R-type, in which one operand is stored in general registers, the other is stored in the accumulator register, and the result is stored in registers) and immediate (I-type, in which one operand is an 8-bit immediate value) formats.

- **Load** instructions move data between memory and some special registers (the accumulator register, and the Timer). They are all immediate (I-type) instructions.

- **Jump and Branch** instructions change the control flow of a program. Those instructions are always made to paged, absolute address formed by combining an 8-bit immediate address with the high-order 8-bit register (HP) (I-type).

- **Special** instructions perform program calls, returns, and changing the PSW (Program Status Word) register. These instructions are always O-type.

Table 1-1 through 1-5 list CPU instructions.
### Table 1-1 CPU Instruction Set: Computational Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDC</td>
<td>Add registers</td>
</tr>
<tr>
<td>ADDI</td>
<td>Add Immediate</td>
</tr>
<tr>
<td>SUBC</td>
<td>Subtract registers</td>
</tr>
<tr>
<td>SUBI</td>
<td>Subtract Immediate</td>
</tr>
<tr>
<td>AND</td>
<td>AND registers</td>
</tr>
<tr>
<td>ANDI</td>
<td>AND Immediate</td>
</tr>
<tr>
<td>OR</td>
<td>OR registers</td>
</tr>
<tr>
<td>ORI</td>
<td>OR Immediate</td>
</tr>
<tr>
<td>XOR</td>
<td>XOR registers</td>
</tr>
<tr>
<td>XORI</td>
<td>XOR Immediate</td>
</tr>
<tr>
<td>RRA</td>
<td>Shift Right</td>
</tr>
<tr>
<td>RLA</td>
<td>Shift Left</td>
</tr>
<tr>
<td>CMPI</td>
<td>Subtract Immediate without affecting the accumulator register</td>
</tr>
</tbody>
</table>
### Table 1-2 CPU Instruction Set: Jump and Branch Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>Jump</td>
</tr>
<tr>
<td>JC</td>
<td>Branch on C flag True</td>
</tr>
<tr>
<td>JNC</td>
<td>Branch on C flag False</td>
</tr>
<tr>
<td>JE</td>
<td>Branch on Z flag True</td>
</tr>
<tr>
<td>JNE</td>
<td>Branch on Z flag False</td>
</tr>
</tbody>
</table>

### Table 1-3 CPU Instruction Set: Transfer Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA</td>
<td>Load registers</td>
</tr>
<tr>
<td>STA</td>
<td>Store the accumulator register</td>
</tr>
<tr>
<td>LDP</td>
<td>Load PSW register into the accumulator</td>
</tr>
<tr>
<td>STP</td>
<td>Load the accumulator into PSW register</td>
</tr>
</tbody>
</table>
### Table 1-4 CPU Instruction Set: Load Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAI</td>
<td>Load Immediate into the accumulator</td>
</tr>
<tr>
<td>LDMI</td>
<td>Load Immediate into the Timer</td>
</tr>
<tr>
<td>LDHI</td>
<td>Load Immediate into the HP register</td>
</tr>
</tbody>
</table>

### Table 1-5 CPU Instruction Set: Special Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>Program Call</td>
</tr>
<tr>
<td>RET</td>
<td>Return program call</td>
</tr>
<tr>
<td>RETI</td>
<td>Return Interrupt</td>
</tr>
<tr>
<td>RETM</td>
<td>Return Timer interrupt</td>
</tr>
<tr>
<td>NOP</td>
<td>Nothing happens</td>
</tr>
<tr>
<td>OUT1</td>
<td>Load the accumulator into the output portI</td>
</tr>
<tr>
<td>OUT2</td>
<td>Load the accumulator into the output portII</td>
</tr>
<tr>
<td>IN</td>
<td>Load the input port into the accumulator</td>
</tr>
<tr>
<td>HALT</td>
<td>Stop CPU running</td>
</tr>
<tr>
<td>SET</td>
<td>Set C, I, or M flags</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear C, I, or M flags</td>
</tr>
</tbody>
</table>
Chapter 2

CPU Instruction Set Details

This chapter provides a detailed description of the operation of each instruction. Each CPU instruction is 16 bits long. As shown in Figure 2-1, there are three instruction formats:

- immediate (I-type)
- only opcode (O-type)
- register (R-type)

<table>
<thead>
<tr>
<th>I-Type (Immediate)</th>
<th>15</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
<td>XX</td>
<td>immediate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>O-Type (Only opcode)</th>
<th>15</th>
<th>10</th>
<th>9</th>
<th>5</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
<td>XXXXXXXXXXX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R-Type (Register)</th>
<th>15</th>
<th>10</th>
<th>9</th>
<th>5</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
<td>XX</td>
<td>register</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XX: Don't care

Figure 2-1 CPU Instruction Formats
ADDCAddcADDC

Format:

\[
\text{ADDCA,Register}
\]

Description:
The contents of a general register, the contents of the accumulator, and the content of the C flag are added to form the result. The result is placed into the accumulator. An overflow occurs if the carries out of bits 6 and 7 differ (2's complement overflow).

Operation:

\[
A \leftarrow A + \text{Register} + \text{C flag}
\]

Flag affected:

C,V,Z,and S
ADDI  Add Immediate  ADDI

<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>000010</td>
<td>XX</td>
<td>immediate</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format:

ADDI A,Immediate

Description:

The 8-bit immediate, the contents of the accumulator, and the content of the C flag are added to form the result. The result is placed into the accumulator. An overflow occurs if the carries out of bits 6 and 7 differ (2's complement overflow).

Operation:

A ← A + Immediate + C flag

Flag affected:

C, V, Z, and S
Format:

SUBC A,Register

Description:

The contents of a general register and the content of the C flag are subtracted from the contents of the accumulator to form the result. The result is placed into the accumulator. An overflow occurs if the carries out of bits 6 and 7 differ (2's complement overflow).

Operation:

A ← A - Register - C flag

Flag affected:

C, V, Z, and S
SUBI Subtract immediate SUBI

Format:

SUBI A,Immediate

Description:

The 8-bit immediate and the content of the C flag are subtracted from the contents of the accumulator to form the result. The result is placed into the accumulator. An overflow exception occurs if the carries out of bits 6 and 7 differ (2's complement overflow).

Operation:

A ← A - Immediate - C flag

Flag affected:

C,V,Z, and S
Format:

\[ \text{AND } A, \text{Register} \]

Description:

The contents of general register are combined with the contents of the accumulator in a bit-wise logical AND operation. The result is placed into the accumulator register.

Operation:

\[ A \leftarrow A \text{ and Register} \]

Flag affected:

\[ Z, \text{and } S \]

\[ C \leftarrow 0, \text{and } V \leftarrow 0 \]
Format:

ANDI A,Immediate

Description:

The 8-bit immediate is zero-extended and combined with the contents of the accumulator in a bit-wise logical AND operation. The result is placed into the accumulator.

Operation:

A ← A and Immediate

Flag affected:

Z, and S
C ← 0, and V ← 0
### Format:

```
OR A, Register
```

### Description:

The contents of a general register are combined with the contents of the accumulator in a bit-wise logical OR operation. The result is placed into the accumulator register.

### Operation:

```
A ← A or Register
```

### Flag affected:

- Z, and S
- C ← 0, and V ← 0
OR I

Or Immediate

ORI

Format:

OR I A, Immediate

Description:
The 8-bit immediate is zero-extended and combined with the contents of the accumulator in a bit-wise logical OR operation. The result is placed into the accumulator.

Operation:

A ← A or Immediate

Flag affected:

Z, and S

C ← 0, and V ← 0
Format:

\[
\text{XOR } A, \text{Register}
\]

Description:

The contents of general register are combined with the contents of the accumulator in a bit-wise logical XOR operation. The result is placed into the accumulator register.

Operation:

\[
A \leftarrow A \text{ xor Register}
\]

Flag affected:

\[
Z, \text{and } S
\]

\[
C \leftarrow 0, \text{ and } V \leftarrow 0
\]
**XOR I**  Xor Immediate  **XOR I**

![Binary representation of the XORI instruction](image)

**Format:**

XOR I  A,Immediate

**Description:**

The 8-bit immediate is zero-extended and combined with the contents of the accumulator in a bit-wise logical XOR operation. The result is placed into the accumulator.

**Operation:**

A ← A xor Immediate

**Flag affected:**

Z, and S
C ← 0, and V ← 0
**Format:**

RRA

**Description:**

The contents of the accumulator are shifted right by one bit, inserting the value of an extending bit, which is in the accumulator, into the highest-order bit. The value of the least-order is shifted into the extending bit. The result is placed in the accumulator. Figure 2-2 shows the program how to clear the extending bit, and figure 2-3 shows the program how to set the extending bit.

**Operation:**

A ← A shift right one bit

> b7 → b6 → b5 → b4 → b3 → b2 → b1 → b0 → EX →

EX: Extending bit in the accumulator

**Flag affected:**

Z, V, and S
Format:

RLA

Description:

The contents of the accumulator are shifted left by one bit, inserting the value of an extending bit, which is in the accumulator, into the least-order bit. The value of the highest-order is shifted into the extending bit. The result is placed in the accumulator.

Operation:

A ← A shift left one bit

\[
\begin{align*}
  &b_7 < b_6 < b_5 < b_4 < b_3 < b_2 < b_1 < b_0 < \text{EX} \\
\end{align*}
\]

EX: Extending bit in the accumulator

Flag affected:

S, V, and Z
### Format:

JUMP Immediate

### Description:

The 8-bit immediate address is combined with the HP register (high-order bits of the address). The program unconditionally jumps to this calculated address without any delay of executing instructions.

### Operation:

\[
P C \leftarrow H P, \text{ Immediate} \\
PC(15 - 8) \leftarrow HP \\
PC(7 \sim 0) \leftarrow \text{ Immediate}
\]

### Flag affected:

None
Format:

\[
\text{JC} \quad \text{Immediate}
\]

Description:

The 8-bit immediate address is combined with the HP register (high-order bits of the address). If the C flag is true, then the program branches to this calculated address without any delay of executing instructions. Otherwise, the program will continue executing the next address.

Operation:

If \ C \ flag = 1

then

\[ \text{PC} \leftarrow \text{HP}, \text{Immediate} \]

else

\[ \text{PC} \leftarrow \text{PC} +1 \]

Flag affected:

None
Format:

JE Immediate

Description:

The 8-bit immediate address is combined with the HP register (high-order bits of the address). If the Z flag is true, then the program branches to this calculated address without any delay of executing instructions. Otherwise, the program will continue executing the next address.

Operation:

If C flag = 1
then
  PC ← HP, Immediate
else
  PC ← PC +1

Flag affected:

None
CMPI  Compare Immediate  CMPI

<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>010000</td>
<td>XX</td>
<td>immediate</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format:

CMPI  A,Immediate

Description:

The 8-bit immediate is zero-extended and combined with the contents of the accumulator in a subtraction operation. The result will not be placed into the accumulator, but just affects the Z flag. This instruction is useful when a user just wants to check if the contents of the accumulator are equal to some immediate value.

Operation:

A subtract Immediate

Flag affected:

Z, and S
C ← 0, and V←0
**Format:**

LDAI Immediate

**Description:**

The 8-bit immediate is zero-extended and loaded into the accumulator.

**Operation:**

A ← Immediate

**Flag affected:**

None
**LDA**  
Load Register  

<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>5</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>XX</td>
<td>register</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:**

LDA Register

**Description:**

The contents of general register are loaded into the accumulator.

**Operation:**

A ← Register

**Flag affected:**

None
### STA Load Accumulator

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>10</th>
<th>9</th>
<th>5</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>010011</td>
<td>XX</td>
<td>register</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Format:

STA Register

#### Description:

The contents of the accumulator are loaded into general registers.

#### Operation:

Register ← A

#### Flag affected:

None
Load Immediate

Format:

LDMI Immediate

Description:

The 8-bit immediate is zero-extended and loaded into the Timer.

Operation:

Timer ← Immediate

Flag affected:

None
OUT1

Output to Port I

15 10 9

010101 XXXXXXXXXXXX

6 10

Format:

OUT1 A

Description:

The contents of the accumulator are loaded into the output port I.

Operation:

Output Port I ← A

Flag affected:

None
OUT2

Output to PortII

Format:

OUT2 A

Description:

The contents of the accumulator are loaded into the output portII.

Operation:

Output PortII ← A

Flag affected:

None
Format:

IN  A

Description:
The contents of the input port are loaded into the accumulator.

Operation:

A ← Input Port

Flag affected:

None
CALL    Call    CALL

011000  XX  immediate

Format:

CALL    Immediate

Description:

The 8-bit immediate address is combined with the HP register (high-order bits of the address). The program unconditionally jumps to this calculated address without any delay of executing instructions after pushing the contents of the program counter onto the stack.

Operation:

Stack ← PC
Stack Pointer ← Stack Pointer + 1
PC ← HP, Immediate

Flag affected:

None
**Format:**

```
RET
```

**Description:**

This instruction pops the return address off the stack into the program counter.

**Operation:**

- PC ← The contents of the stack popped
- Stack Pointer ← Stack Pointer - 1

**Flag affected:**

None
**Description:**

Nothing happens.

**Operation:**

Nothing

**Flag affected:**

None
Format:

    HALT

Description:

This processor stopped executing any instructions. The processor can run again by enabling the RESET signal.

Flag affected:

None
Format:

\[ \text{JNC} \quad \text{Immediate} \]

Description:

The 8-bit immediate address is combined with the HP register (high-order bits of the address). If the C flag is not true, then the program branches to this calculated address without any delay of executing instructions. If the C flag is set, then the next sequential instruction is executed.

Operation:

If \( \text{C flag} = 0 \)

then

\[ \text{PC} \leftarrow \text{HP}, \text{Immediate} \]

else

\[ \text{PC} \leftarrow \text{PC} + 1 \]

Flag affected:

None
Format:

JNE Immediate

Description:
The 8-bit immediate address is combined with the HP register (high-order bits of the address). If the Z flag is not true, then the program branches to this calculated address without any delay of executing instructions. If the Z flag is set, then the next sequential instruction is executed.

Operation:
If C flag = 0
then
PC ← HP, Immediate
else
PC ← PC +1

Flag affected:
None
**LDHI**  
Load Immediate  

<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>011110</td>
<td>XX</td>
<td>immediate</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format:

LDHI Immediate

Description:

The 8-bit immediate is zero-extended and loaded into the HP register. (Higher-order 8-bit Register for the program counter)

Operation:

HP ← Immediate

Flag affected:

None
**Format:**

CLRI

**Description:**

The I flag is cleared. It will disable the Interrupt signal.

**Operation:**

I flag ← 0

**Flag affected:**

I
Format:

SETI

Description:

The I flag is set. It will enable the Interrupt signal.

Operation:

I flag ← 1

Flag affected:

1
CLRC  Clear C flag  CLRC

Format:

CLRC

Description:
The C flag is cleared.

Operation:

C flag ← 0

Flag affected:

C
Format:

```
SETC
```

Description:

The C flag is set.

Operation:

```
C flag ← 1
```

Flag affected:

```
C
```
Format:

CLRM

Description:

The M flag is cleared. It will disable Timer interrupt.

Operation:

M flag ← 0

Flag affected:

M
Format:

SETM

Description:

The M flag is set. It will enable Timer interrupt.

Operation:

M flag ← 1

Flag affected:

M
RETI

Format:

RETI

Description:
This instruction pops the return address off the stack into the program counter and enable the Interrupt signal and Timer interrupt. This instruction must be used in the Interrupt program.

Operation:

PC ← The contents of the stack popped
Stack Pointer ← Stack Pointer -1

Flag affected:
None
Format:

RETM

Description:

This instruction pops the return address off the stack into the program counter and enable the Interrupt signal and Timer interrupt. This instruction must be used in Timer program.

Operation:

PC ← The contents of the stack popped
Stack Pointer ← Stack Pointer -1

Flag affected:

None
LDP
Load PSW

Format:
LDP

Description:
The contents of PSW register are loaded into the accumulator.

Operation:
\[ A \leftarrow \text{PSW Register} \]

Flag affected:
None
STP Load Accumulator STP

\begin{verbatim}
15 10 9 8 7 6 5 4 3 2 1 0
101000 XXXXXXXXXXXX
\end{verbatim}

Format:

STP

Description:

The contents of the accumulator are loaded into the PSW.

Operation:

PSW Register ← A

Flag affected:

I, M, Z, C, V, and S
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1100</td>
<td>LDAI</td>
<td>#00H; Preset the accumulator with 00H</td>
</tr>
<tr>
<td>0001</td>
<td>0B00</td>
<td>RRA</td>
<td>; Rotate the accumulator right one bit</td>
</tr>
</tbody>
</table>

Figure 2-2 Clearing the extending bit program

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>11FF</td>
<td>LDAI</td>
<td>#FFH; Preset the accumulator with FFH</td>
</tr>
<tr>
<td>0001</td>
<td>0B00</td>
<td>RRA</td>
<td>; Rotate the accumulator right one bit</td>
</tr>
</tbody>
</table>

Figure 2-3 Setting the extending bit program

CLK: 0 → Fetch, 1 → Output (PC, IR)
CLK: 1 → Fetch, 0 → Output (register C, register D, register B, Accumulator)

Figure 3-2 A example of pipeline
Chapter 3

The CPU Pipeline

This chapter describes the basic operation of the CPU pipeline and lists the operation how to execute an instruction on each stage of the pipeline.
3.1 CPU Pipeline Operation

The CPU has a four-stage instruction pipeline; each stage takes one clock cycle. Thus, the execution of each instruction takes at least four clock cycles. Once the pipeline has been filled, four instructions are executed simultaneously. Figure 3-1 shows the four stages of the instruction pipeline. Figure 3-2 shows an example of how the pipeline works.

![Instruction Pipeline Stages](image)

Figure 3-1 Instruction Pipeline Stages

3.2 CPU Pipeline Stages

This section describes each of the four pipeline stages:

- **FE** -- Instruction Fetch
- **EX1** -- First step of execution
- **EX2** -- Second step of execution
- **EX3** -- Third step of execution
**FE -- Instruction Fetch**

During the IF stage, the following occurs:

- The instruction (pointed by Program Counter) in memory will be loaded into the first buffer in the controller.
- Program Counter points next instruction executed.

**EX1 -- First step of execution**

During the EX1 stage, one of the following occurs:

- The Address Register is loaded with the number of general register. This operation is for ADDC, SUBC, AND, OR, XOR, LDA, and STA instructions.
- The Register C is loaded with Immediate. This operation is for ADDI, SUBI, ANDI, ORI, Xori, CMPI, and LDAI instructions.
- Program Counter is loaded with Immediate and the contents of HP register. This operation is for JMP, JC, JE, JNC, JNE, and CALL instructions.
- Timer is loaded with Immediate. This operation is for LDMI instruction.
- The In port catches data from input pins. This operation is for IN instruction.
- Program Counter is loaded with the contents of Stack. This operation is for RET, RETI, and RETM instructions.
- HP register is loaded with Immediate. This operation is for LDHI instruction.
- Nothing happens. This operation is for RLA, RRA, OUT1, OUT2, NOP, LDP, and STP instructions.
- The CPU is stopped running. This operation is for HALT instruction.
- PSW register is loaded with Immediate. This operation is for CLRI, SETI, CLRC, SETC, CLRM, and SETM instructions.

**EX2 -- Second step of execution**

During the EX2 stage, one of the following occurs:

- Register B is loaded with the contents of the general register that Address Register points. This operation is for ADDC, SUBC, AND, OR, and XOR instructions.
- Register B is loaded with the contents of Register C. This operation is for ADDI, SUBI, ANDI, ORI, XORI, and CMPI instructions.
- Nothing happens. This operation is for RRA, RLA, JMP, JC, JE, STA, STMI, OUT1, OUT2, IN, NOP, HALT, JNE, JNC, LDHI, CLRI, SETI, CLRC, SETC, CLRM, SETM, LDP, and STP instructions.
- Register D is loaded with the contents of Register C. This operation is for LDAI instruction.
- Register D is loaded with the contents of the general register that Address register points. This operation is for LDA instruction.
- Stack Pointer is reduced by one. This operation is for CALL instruction.
- Stack Pointer is added by one. This operation is for RET, RETI, and RETM instruction.

**EX3 -- Third step of execution**

During the EX3 stage, one of the following occurs:

- The accumulator is loaded with the contents of ALU (arithmetic logic unit). This operation is for ADDC, ADDI, SUBC, SUBI, AND, ANDI, OR, ORI, XOR, and XORI instructions.
- The contents of the accumulator are shifted left by one bit. This operation is for RLA instruction.
- The contents of the accumulator are shifted right by one bit. This operation is for RRA instruction.
- Nothing happens. This operation is for JMP, JC, JE, LDMI, CALL, RET, RETI, RETM, NOP, HALT, JNC, JNE, LDHI, CLRI, SETI, CLRC, SETC, SETM, and CLRM instructions.
- The Z flag is changed. This operation is not only for CMPI instruction.
- The accumulator is loaded with the contents of Register D. This operation is for LDAI and LDA instructions.
- The general register that Address Register points is loaded with the contents of the accumulator. This operation is for STA instruction.
- The output PORT1 is loaded with the contents of the accumulator. This operation is for OUT1 instruction.
- The output PORTII is loaded with the contents of the accumulator. This operation is for OUT2 instruction.
- The accumulator is loaded with the contents of the input PORT. This operation is for IN instruction.
- The interrupt register is cleared. This operation is for RETI instruction.
- The Timer interrupt register is cleared. The operation is for RETM instruction.
- The PSW register is loaded with the contents of the accumulator. This operation is for LDP instruction.
- The accumulator is loaded with the contents of the PSW register. This operation is for STP instruction.

### 3.3 Limitation of the CPU pipeline

This CPU pipeline executes an instruction per clock cycle even if it executes JUMP, Condition Jump, CALL, and Return instructions. It will not have any delay of executing instructions; however, the following rules must be obeyed:

1. When a user uses STA instruction, next two instructions cannot be STA, LDA, ADDC, SUBC, AND, OR, or XOR instruction.
2. When a user uses CMPI instruction, next two instructions can not be any condition jump instructions.
3. When a user uses STP instruction, next two instructions can not be any condition jump instructions.
This chapter describes the 16-bit and an 8-bit System Interface. The System Interface includes:

- a 16-bit address bus
- a 16-bit data bus (unidirectional)
- two 8-bit output ports
- an 8-bit input port
- an interrupt interface pin
- a reset pin
- a clock pin
4.1 16-bit Address Bus (A0--A15)

The 16-bit address bus is used to choose which address in memory will be fetched.

4.2 16-bit Data Bus (L0--L15)

The 16-bit data bus carries the 16 data bits that the 16-bit address points in memory to the controller.

4.3 Two 8-bit Output ports (DO0--DO7,DO8--DO15)

The OUT1 and OUT2 instructions transfer the contents of the accumulator to these two 8-bit output ports so a user can send the contents of the accumulator or other data to extent devices by using the output ports.

4.4 8-bit Input Port (DI0--DI7)

The IN instruction transfers the contents of the input port to the accumulator, therefore, a user can receive data from other devices by using this input port.

4.5 Interrupt Control Pin (INT)[3]

Sometime it is necessary to interrupt the execution of the main program to answer a request from an I/O device. For instance, an I/O device may send an interrupt signal to the interrupt control pin to indicate that data is ready for input. The computer temporarily stops what it is doing, inputs the data, then returns to what it was doing. An interrupt signal must be asserted for at least one clock cycle. This interrupt control pin actives high and the PC will be loaded 0007H.
4.6 Reset Pin (RESET)

The reset pin carries the RESET signal. This signal may come from an operator reset button or other source. When RESET is high, the CPU will reset the program counter (0000H), and other registers. The CPU remains in reset until the RESET goes low.

4.7 Clock Pin (CLKIN)

The processor doesn't generate an internal clock. A user must send a clock to the CLKin.
This chapter describes the following:

- 32 general purpose registers
- a Program Counter (PC) register
- an 8-bit Timer
- an 8-bit integer arithmetic logic unit (ALU)
- 16 stack registers
- a Program Status Word (PSW) register
- an Accumulator
- a Higher-order 8-bit Address Register for the program counter (HP)
5.1 32 General Purpose Registers (R0--R31)

These general purpose registers are like a small on-chip RAM with addressable memory locations. Control signals select the register for a read or write operation. This means that CPU can either load a register from the 8-bit internal data bus or output the register to another data bus.

5.2 Program Counter (PC) [3]

The program is stored at the beginning of the memory with the first instruction at binary address 0000, the second instruction at address 0001, the third at address 0002, and so on. The program counter, which is part of the control unit, counts from 0000 to FFFFH. Its job is to keep track of the next instruction to be fetched and executed.

5.3 8-bit Timer

This 8-bit timer is an 8-bit counter. CPU can preset the counter with a certain number and the counter will increase one by one. When the content of the counter is 0000, the timer will send the timer interrupt to the controller. If the M flag is high, the computer temporarily stops what it is doing, does a job for the timer interrupt, then return to what it was doing. This interrupt makes PC be loaded 0007H.

5.4 16 stack registers

The 16 stack registers are 16-bit registers. CPU has a CALL instruction (like Interrupt and Timer interrupt) that sends the program to a subroutine. As we know, before the jump takes place, the program counter is incremented and the address is saved at the stack register that the stack pointer points. The stack pointer will increase one after this happening. At
the completion of a subroutine, the RET, RETM, or RETI instruction loads the program counter with the return address, which allows the computer to get back to the main program.

5.5 Program Status Word (PSW) Register

The Program Status Word register (PSW) is a read/write register that contains the operating mode, and interrupt enabling. The following list describes the more important Status register field; Figure 5-1 shows the format of the Status register. Table 5-3 describes the status register fields.

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>V</td>
<td>S</td>
<td>I</td>
<td>M</td>
<td>Z</td>
<td>C</td>
</tr>
</tbody>
</table>
```

Figure 5-1 Program Status Word Register
Table 5-1 Status register Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>It is an overflow flag. When a sum or difference lies outside the normal range of the accumulator ( -128~127), this flag will be high.</td>
</tr>
</tbody>
</table>
| S     | The S flag is set when the accumulator contents become negative during the execution of some instructions.  
S flag = b7 of ALU |
| I     | Interrupt mask: controls the enabling of external interrupt (INT). The interrupt is taken if interrupt is enabled, and the corresponding bit is set in the Interrupt field of the PSW register.  
0→disabled  
1→enabled |
| M     | Timer interrupt Mask: controls the enabling of internal timer interrupt. The interrupt is taken if interrupt is enabled, and the corresponding bit is set in the timer interrupt Mask field of the PSW register  
0→disabled  
1→enabled |
| Z     | 0→the result of ALU is not zero  
1→the result of ALU is zero |
| C     | 1→carry happens during computation  
0→carry doesn’t happen during computation |
5.6 Accumulator

The accumulator is a buffer register that stores intermediate answers during a computer run. Its two-state output goes directly to the ALU, general registers, two output ports, and PSW register. Its input comes from a data bus.

5.7 Higher-order 8-bit Address Register for the program counter (HP)

This register stores the higher-order address that combined with an 8-bit immediate address. Those JUMP, JNE, JNC, JE, JC, and CALL instructions use this register to jump to an address.
Chapter 6

Memory

The processor doesn't provide any instructions for reading or writing RAM (Random-Access Memory) because this kind of instructions will have delay of executing instructions. It doesn't mean that a user cannot use this processor to read or write RAM. There are two output ports and one input port in this processor so a user can use those I/O ports to do some things about RAM.

This chapter describes the following:

- How to build a RAM read/write circuit.
- How to design writing RAM program
- How to design reading RAM program
6.1 Read/Write RAM Circuit:

Figure 6-1 is a block diagram of the circuit internal. The circuit uses DO15 to decide if the data from outport ports is data or address of RAM. Next, the circuit uses DO14 to determine if the processor wants to read or write. Finally, the processor can read RAM from the input port. The following is the detail of signals:

- DO15: 1→ADDRESS, 0→DATA
- DO14: 1→READ, 0→WRITE
- DO13~DO8: ADDRESS
- DO7~DO0: ADDRESS/DATA
- DI0~DI7: DATA (READ)

6.2 Writing RAM Program

Figure 6-2 is the writing RAM diagram. In order to write data into RAM, first, the program sends the address into the RAM through the two output ports. Second, the program sends the data into the RAM through the output port1. Third, the program sets the R/W signal to be low. Finally, the processor writes data into the external RAM. The following is the writing program:

```
LDA    H-address ;Load the accumulator with Hi-ADD
ORI    C0H      ;Set DO15 and R/W high
OUT2   A         ;Output to the circuit
LDA    L-address ;Load the accumulator with Lo-ADD
OUT1   A         ;Output to the circuit
LDAI   40H       ;Clear the DO15
OUT2   A
```
LDAI DATA
OUT1 A ;Output the data to the RAM
LDAI 00H ;Clear the R/W signal
OUT1 A

6.3 Reading RAM Program

Figure 6-3 is the reading RAM timing diagram. In order to read data from RAM, first, the program sends the address into the RAM through the two output ports. Second, the program sets the R/W signal to be high. Finally, the processor reads data from the external RAM through the input port. The following is the program:

LDA H-address ;Load the accumulator with Hi-ADD
ORI C0H ;Set DO15 and R/W high
OUT2 A ;Output to the circuit
LDA L-address ;Load the accumulator with Lo-ADD
OUT1 A ;Output to the circuit
NOP ;Delay one clock cycle time
IN A ;Read data from the RAM
Figure 6-1 RAM read/write Circuit Internal Block Diagram

Figure 6-2 Write timing diagram
Figure 6-3 Read timing diagram
Chapter 7

Test Program

This chapter describes 22 test programs that check if each instruction is working or not, and also shows the simulation result to prove that the CPU pipeline executes an instruction per clock cycle even if it executes JUMP, Condition Jump, CALL, and Return instructions. It will not have any delay of executing instructions. The following is the explanation of each signal in the simulation results:

1. /CON(48): Clock signal
2. /DO0 ~ /DO7: Outputs of the output Register I.
3. /DO8 ~ /DO15: Outputs of the output Register II
4. /N$507: V flag.
5. /N$506: S flag.
6. /P(1): C flag
7. /P(2): Z flag
8. /P(3): M flag
9. /P(4): I flag
10. /P(0): Output of the Timer.
7.1 Test Program 01
This program tests 'NOP', 'LDA,#DATA','OUT1 A', 'OUT2 A', 'STA R5,#DATA', and 'JMP #DATA' instructions. First, the program loads 00H into the accumulator, and then the program transfers the values of the accumulator to the two output registers. Second, the program loads FFH into the accumulator, and then the program transfers the value of the accumulator to the two output registers. Finally, the program repeats each step for a loop. In conclusion, the two output registers produce a rectangular waveform. Figure 7-1-1 shows the program. Figure 7-1-2 shows the simulation result.

7.2 Test Program 02
This program tests 'RRA' instruction. First, the program loads 01H into the accumulator. Second, the program rotates the accumulator right. Third, the program loads the two output registers with the value of the accumulator. Finally, the program repeats the steps from second to third. In conclusion, the two output registers produce that only one of eight bits is high from b7 to b0. Figure 7-2-1 shows the program. Figure 7-2-2 shows the simulation result.

7.3 Test Program 03
This program tests 'RLA' instruction. First, the program loads 01H into the accumulator. Second, the program rotates the accumulator left. Third, the program loads the two output registers with the value of the accumulator. Finally, the program repeats the steps from second to third. In conclusion, the two output registers produce that only one of eight bits is high from b0 to b7. Figure 7-3-1 shows the program. Figure 7-3-2 shows the simulation result.
7.4 Test Program 04
This program tests 'CLR C' and 'ADDI A, #DATA' instructions. First, the program loads 00H into the accumulator, and then clears the flag C. Second, the program adds the value of the accumulator and one up, and then loads the result of computation into the accumulator. Third, the program loads the accumulator into the two output registers. Finally, the program repeats each step again. In conclusion, the contents of the output registers increase one by one. Figure 7-4-1 shows the program. Figure 7-4-2 shows the simulation result.

7.5 Test Program 05
This program tests 'SET C' and 'SUBI A, #DATA' instructions. First, the program loads FFH into the accumulator, and then sets the flag C. Second, the program subtracts the value of the accumulator with one, and then loads the result of computation into the accumulator. Third, the program loads the accumulator into the two output registers. Finally, the program repeats each step again. In conclusion, the contents of the output registers decrease one by one. Figure 7-5-1 shows the program. Figure 7-5-2 shows the simulation result.

7.6 Test Program 06
This program tests 'XORI A, #DATA' instructions. First, the program loads 00H into the accumulator. Second, the program XOR the value of the accumulator with FFH, and then loads the result of computation into the accumulator. Third, the program loads the accumulator into the two output registers. Finally, the program repeats each step again. In conclusion, the two output registers produce a rectangular wave form. Figure 7-6-1 shows the program. Figure 7-6-2 shows the simulation result.
7.7 Test Program 07

This program tests 'ORI A,#DATA' and 'ANDI A,#DATA' instructions. First, the program presets the accumulator with 00H, and then loads the accumulator into the output registers. Second, the program OR the accumulator with FFH and AND the accumulator with FEH. Third, the program loads the accumulator into the output registers. Finally, the program repeats each step again. In conclusion, the two output registers produce rectangular waveform, but b0 of the output registers are always low. Figure 7-7-1 shows the program. Figure 7-7-2 shows the simulation result.

7.8 Test Program 08

This program tests 'IN A' instruction. First, the program loads the accumulator with the input register. Second, the program loads the accumulator into the output registers. Finally, the program repeats each step again. In conclusion, the output registers produce the output that the CPU fetched from the input port. Figure 7-8-1 shows the program. Figure 7-8-2 shows the simulation result.

7.9 Test Program 09

This program tests 'SET C', 'SET I', 'CLR I', 'CLR C', and 'CLR M' instructions. First, the program sets the flag I, C, and M. Second, the program clears the flag I, C, and M. Finally, the program repeats each step again. In conclusion, the flag I, C, and M produces a rectangular waveform. Figure 7-9-1 shows the program. Figure 7-9-2 shows the simulation result.

7.10 Test Program 10

This program tests 'LDA REGISTER' and 'STA REGISTER' instructions. First, the program presets the accumulator with 00H and then loads the
register 0 with the accumulator. Second, the program presets the accumulator with 01H and then loads the register 16 with the accumulator. Third, the program presets the accumulator with 02H and then loads the register 21 with the accumulator. Forth, the program presets the accumulator with 03H and then loads the register 31 with the accumulator. Fifth, the program loads the accumulator with the register 0 and then loads the output register 1 with the accumulator. Sixth, the program loads the accumulator with the register 16 and then loads the output register 1 with the accumulator. Seventh, the program loads the accumulator with the register 21 and the loads the output register 1 with the accumulator. Eight, the program loads the accumulator with the register 31 and then loads the output register 1 with the accumulator. Finally, the program repeats the steps from fifth to eighth. In conclusion, the output register 1 produces the output that is 00H, 01H, and 03H alternately. Figure 7-10-1 shows the program. Figure 7-10-2 shows the simulation result.

7.11 Test Program 11

This program tests 'ADDC A,REGISTER' instruction. First, the program presets the register 0 with 01H and the accumulator with 00H. Second, the program clears the flag C. Third, the program adds the accumulator with the register 0. Forth, the program loads the output registers with the accumulator. Finally, the program repeats the steps from third to forth. In conclusion, the output registers increase one by one. Figure 7-11-1 show the program. Figure 7-11-2 show the simulation result.
7.12 Test Program 12

This program tests 'SUBC A, REGISTER' instruction. First, the program presets the register 0 with 01H and the accumulator with FFH. Second, the program sets the flag C. Third, the program subtracts the accumulator with the register 0. Fourth, the program loads the output registers with the accumulator. Finally, the program repeats the steps from third to forth. In conclusion, the output registers decrease one by one. Figure 7-12-1 shows the program. Figure 7-12-1 shows the simulation result.

7.13 Test Program 13

This program tests 'XOR A, REGISTER' instruction. First, the program presets the register 0 with FFH and the accumulator with 00H. Second, the program XOR the accumulator with the register 0. Third, the program loads the output registers with the accumulator. Finally, the program repeats the steps from second to third. In conclusion, the output registers produce rectangular output. Figure 7-13-1 shows the program. Figure 7-13-1 shows the simulation result.

7.14 Test Program 14

This program tests 'OR A, REGISTER' and 'AND A, REGISTER' instruction. First, the program preset the register 0 with FFH, the register 1 with FFH, and the accumulator with 00H. Second, the program loads the output registers with the accumulator. Third, the program OR the accumulator with the register 1. Fourth, the program loads the output registers with the accumulator. Finally, the program repeats the instructions from the address 0004H to the address 000CH. In conclusion, the output registers produce rectangular output, but b0 of the output
registers are always low. Figure 7-14-1 shows the program. Figure 7-14-2 shows the simulation result.

7.15 Test Program 15

This program tests 'JE ADDRESS' and 'CMPI A,#DATA' instructions. First, the accumulator is loaded with 66H. Second, the program compares the accumulator with 66H. If the accumulator is equal to 66H, the program will preset the accumulator with 66H, otherwise, the program will preset the accumulator with FFH. Third, the program loads the output register I with the accumulator. Finally, the program repeats the steps from second to third. In conclusion, the output register I produces FFH and 66H output alternately. Figure 7-15-1 shows the program. Figure 7-15-2 shows the simulation result.

7.16 Test Program 16

This program tests 'JNE ADDRESS' and 'CMPI A,#DATA' instructions. First, the accumulator is loaded with 66H. Second, the program compares the accumulator with 66H. If the accumulator is not equal to 66H, the program will preset the accumulator with 66H, otherwise, the program will preset the accumulator with FFH. Third, the program loads the output register I with the accumulator. Finally, the program repeats the steps from second to third. In conclusion, the output register I produces FFH and 66H output alternately. Figure 7-16-1 shows the program. Figure 7-16-2 shows the simulation result.

7.17 Test Program 17

This program tests 'JC ADDRESS' and 'JNC ADDRESS' instructions. First, the program sets the flag C. Second, if the flag C is high, the program presets the accumulator with 66H, otherwise, the program presets the accumulator with 55H. Second, the program loads the output
register I with the accumulator. Third, the program clears the flag C. Forth, if the flag C is low, the program presets the accumulator with 55H, otherwise, the program presets the accumulator with FFH. Sixth, the program loads the output register I with the accumulator. Finally, the program repeats the steps from first to sixth. In conclusion, the output register I produces 66H and 55H output alternately. Figure 7-17-1 shows the program. Figure 7-17-2 shows the simulation result.

7.18 Test Program 18
This program tests 'CALL ADDRESS' and 'RET' instructions. First, the program presets the accumulator with FFH. Second, the program loads the output register I with the accumulator. Third, the program jumps to the address 0005H because of CALL instruction. Forth, the program preset the accumulator with 00H. Fifth, the program loads the output register I with the accumulator. Sixth, the program jump back the address 0004H because of the RET instruction. Finally, the program repeats the steps from first to sixth. In conclusion, the output register I produces 00H and FFH output alternately. Figure 7-18-1 shows the program. Figure 7-18-2 shows the simulation result.

7.19 Test Program 19
This program tests 'INTERRUPT' and 'RETI' instructions. First, the program sets the flag I. Second, the program presets the accumulator with 00H. Third, the program loads the output register I with the accumulator. Finally, the program repeats the steps from second to third. In conclusion, the output register I produces 00H output. It will produce FFH output for short time when the interrupt (INT) signal is high. Figure 7-19-1 shows the program. Figure 7-19-2 show the simulation result.
7.20 Test Program 20

This program tests 'STM #DATA' and 'RETM' instructions. First, the program loads the timer with F0H and then sets the flag M. Second, the program loads the accumulator with 00H. Third, the program loads the output register I with the accumulator. Finally, the program repeats the steps from second to third. In conclusion, the output register I produces 00H output. It will produce FFH output for short time after the timer is zero (after 240 clock cycles). Figure 7-20-1 shows the program. Figure 7-20-2 shows the simulation result.

7.21 Test Program 21

This program tests 'HLT' instruction. First, the program sets the flag I. Second, the program loads the accumulator with 00H. Third, the program loads the output register I with the accumulator. Finally, the program repeats the steps from second to third. In conclusion, the program will stop the clock when the INT signal is high. Figure 7-21-1 shows the program. Figure 7-21-2 shows the simulation result.

7.22 Test Program 22

This program tests 'STP' and 'LDP' instructions. First, the program presets the accumulator with FFH. Second, the program loads the PSW register with the accumulator. Third, the program loads the accumulator with the PSW register. Forth, the program loads the output register I with the accumulator. Fifth, the program presets the accumulator with 00H, and then follows the steps from second to forth. Finally, the program repeats the steps from first to fifth. In conclusion, because b6 and b7 of the PSW register are always zero, the output register I produces 00H and 3FH output alternately. Figure 7-22-1 shows the program. Figure 7-22-2 shows the simulation result.
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>1100</td>
<td>LOOP: LDAI</td>
<td>#00H; Load 00H into the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>1500</td>
<td>OUT1 A</td>
<td>Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0003</td>
<td>1600</td>
<td>OUT2 A</td>
<td>Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0004</td>
<td>11FF</td>
<td>LDAI #FFH</td>
<td>Load FFH into the accumulator</td>
</tr>
<tr>
<td>0005</td>
<td>1500</td>
<td>OUT1 A</td>
<td>Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0006</td>
<td>1600</td>
<td>OUT2 A</td>
<td>Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0007</td>
<td>1E00</td>
<td>LDHI #00H</td>
<td>Load 00H into the HP register</td>
</tr>
</tbody>
</table>

Figure 7-1-1 Test program 01

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>1101</td>
<td>LDAI #01H</td>
<td>Load 00H into the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>0B00</td>
<td>LOOP: RRA</td>
<td>Rotate the accumulator right</td>
</tr>
<tr>
<td>0003</td>
<td>1500</td>
<td>OUT1 A</td>
<td>Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0004</td>
<td>1600</td>
<td>OUT2 A</td>
<td>Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0005</td>
<td>1E00</td>
<td>LDHI #00H</td>
<td>Load 00H into HP register</td>
</tr>
<tr>
<td>0006</td>
<td>0D02</td>
<td>JMP LOOP</td>
<td>Jump back the address 0002H</td>
</tr>
</tbody>
</table>

Figure 7-2-1 Test Program 02

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>1101</td>
<td>LDAI #01H</td>
<td>Load 00H into the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>0C00</td>
<td>LOOP: RLA</td>
<td>Rotate the accumulator left</td>
</tr>
</tbody>
</table>
Figure 7-3-1 Test Program 03

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>0001</td>
<td>1100</td>
<td>LDAI</td>
<td>#00H; Load 00H into the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>2100</td>
<td>CLR</td>
<td>C; Clear the flag C</td>
</tr>
<tr>
<td>0003</td>
<td>0201</td>
<td>LOOP</td>
<td>ADDI</td>
</tr>
<tr>
<td>0004</td>
<td>1500</td>
<td>OUT1</td>
<td>A; Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0005</td>
<td>1600</td>
<td>OUT2</td>
<td>A; Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0006</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H; Load 00H into the HP register</td>
</tr>
<tr>
<td>0007</td>
<td>0D03</td>
<td>JMP</td>
<td>LOOP; Jump back the address 0003H</td>
</tr>
</tbody>
</table>

Figure 7-4-1 Test Program 04

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>0001</td>
<td>1FF</td>
<td>LDAI</td>
<td>#FFH; Load FFH into the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>2201</td>
<td>SET</td>
<td>C; Set the flag C</td>
</tr>
<tr>
<td>0003</td>
<td>0401</td>
<td>LOOP</td>
<td>SUBC A,#01H; Subtract the accumulator with 1</td>
</tr>
<tr>
<td>0004</td>
<td>1500</td>
<td>OUT1</td>
<td>A; Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0005</td>
<td>1600</td>
<td>OUT2</td>
<td>A; Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0006</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H; Load 00H into the HP register</td>
</tr>
</tbody>
</table>
### Figure 7-5-1 Test Program 05

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>0001</td>
<td>1100</td>
<td>LDAI</td>
<td>#00H; Load 00H into the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>0AFF</td>
<td>LOOP:</td>
<td>XORI A.#FFH; XOR the accumulator with FFH</td>
</tr>
<tr>
<td>0003</td>
<td>1500</td>
<td>OUT1</td>
<td>A; Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0004</td>
<td>1600</td>
<td>OUT2</td>
<td>A; Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0005</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H; Load 00H into the HP register</td>
</tr>
<tr>
<td>0006</td>
<td>0D03</td>
<td>JMP</td>
<td>LOOP; Jump back the address 0002H</td>
</tr>
</tbody>
</table>

### Figure 7-6-1 Test Program 06

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>0001</td>
<td>1100</td>
<td>LOOP:</td>
<td>LDAI #00H; Preset the accumulator with 00H</td>
</tr>
<tr>
<td>0002</td>
<td>1500</td>
<td>OUT1</td>
<td>A; Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0003</td>
<td>1600</td>
<td>OUT2</td>
<td>A; Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0004</td>
<td>08FF</td>
<td>ORI</td>
<td>A.#FFH; OR the accumulator with FFH</td>
</tr>
<tr>
<td>0005</td>
<td>06FE</td>
<td>ANDI</td>
<td>A.#FEH; AND the accumulator with FEH</td>
</tr>
<tr>
<td>0006</td>
<td>1500</td>
<td>OUT1</td>
<td>A; Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0007</td>
<td>1600</td>
<td>OUT2</td>
<td>A; Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0008</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H; Load 00H into the HP register</td>
</tr>
<tr>
<td>0009</td>
<td>0D03</td>
<td>JMP</td>
<td>LOOP; Jump back the address 0001H</td>
</tr>
</tbody>
</table>

### Figure 7-7-1 Test Program 07
Figure 7-8-1 Test Program 08

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>1700</td>
<td>LOOP</td>
<td>A: Load the input register into the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>1500</td>
<td>OUT1</td>
<td>A: Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0003</td>
<td>1600</td>
<td>OUT2</td>
<td>A: Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0004</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H; Load 00H into the HP register</td>
</tr>
<tr>
<td>0005</td>
<td>0D03</td>
<td>JMP</td>
<td>LOOP; Jump back the address 0001H</td>
</tr>
</tbody>
</table>

Figure 7-9-1 Test Program 09

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>2001</td>
<td>LOOP</td>
<td>SET I: Set the flag I</td>
</tr>
<tr>
<td>0002</td>
<td>2201</td>
<td>SET</td>
<td>C: Set the flag C</td>
</tr>
<tr>
<td>0003</td>
<td>2401</td>
<td>SET</td>
<td>M: Set the flag M</td>
</tr>
<tr>
<td>0004</td>
<td>1F00</td>
<td>CLR</td>
<td>I: Clear the flag I</td>
</tr>
<tr>
<td>0005</td>
<td>2100</td>
<td>CLR</td>
<td>C: Clear the flag C</td>
</tr>
<tr>
<td>0006</td>
<td>2300</td>
<td>CLR</td>
<td>M: Clear the flag M</td>
</tr>
<tr>
<td>0007</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H; Load 00H into HP register</td>
</tr>
<tr>
<td>0008</td>
<td>0D01</td>
<td>JMP</td>
<td>LOOP; Jump back the address 0001H</td>
</tr>
</tbody>
</table>

ADDRESS | CODE | LABEL | LANGUAGE |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>1100</td>
<td>LDAI</td>
<td>#00H: Preset the accumulator with 00H</td>
</tr>
<tr>
<td>0002</td>
<td>1300</td>
<td>STA</td>
<td>R0; Load the register 0 with the accumulator</td>
</tr>
</tbody>
</table>
0003  1101  LDAI  #01H; Preset the accumulator with 01H
0004  1310  STA   R10; Load the register 16 with the accumulator
0005  1102  LDAI  #02H; Preset the accumulator with 02H
0006  1315  STA   R0; Load the register 21 with the accumulator
0007  1103  LDAI  #03H; Preset the accumulator with 03H
0008  131F  STA   R0; Load the register 31 with the accumulator
0009  0000  LOOP: NOP
000A  0000  NOP
000B  1200  LDA   R0; Load the accumulator with the register 0
000C  1500  OUT1  A; Load the output register1 with the accumulator
000D  1210  LDA   R10; Load the accumulator with the register 16
000E  1500  OUT1  A; Load the output register1 with the accumulator
000F  1215  LDA   R15; Load the accumulator with the register 21
0010  1500  OUT1  A; Load the output register1 with the accumulator
0011  121F  LDA   R1F; Load the accumulator with the register 31
0012  1500  OUT1  A; Load the output register1 with the accumulator
0013  1E00  LDHI  #00H; Load 00H into HP register
0014  0D01  JMP   LOOP: Jump back the address 0009H

Figure 7-10-1 Test Program 10

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1101</td>
<td>LDAI</td>
<td>#01H; Preset the accumulator with 01H</td>
</tr>
<tr>
<td>0001</td>
<td>1300</td>
<td>STA</td>
<td>R0; Load the register 0 with the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>1100</td>
<td>LDAI</td>
<td>#00H; Preset the accumulator with 00H</td>
</tr>
<tr>
<td>0003</td>
<td>2100</td>
<td>CLR</td>
<td>C; Clear the flag C</td>
</tr>
<tr>
<td>0004</td>
<td>1500</td>
<td>LOOP</td>
<td>OUT1 A; Load the accumulator into the output register I</td>
</tr>
</tbody>
</table>
Load the accumulator into the output register

LDHI #00H; Load 00H into the HP register

JMP LOOP; Jump back the address 0004H

Figure 7-11-1 Test Program 11

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1101</td>
<td>LDAI</td>
<td>#01H; Preset the accumulator with 01H</td>
</tr>
<tr>
<td>0001</td>
<td>1300</td>
<td>STA</td>
<td>R0; Load the register 0 with the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>11FF</td>
<td>LDAI</td>
<td>#FFH; Preset the accumulator with 00H</td>
</tr>
<tr>
<td>0003</td>
<td>2201</td>
<td>SET</td>
<td>C; Set the flag C</td>
</tr>
<tr>
<td>0004</td>
<td>0401</td>
<td>LOOP</td>
<td>SUBC A,R0; Subtract the accumulator with register 0</td>
</tr>
<tr>
<td>0005</td>
<td>1500</td>
<td>OUT1</td>
<td>A; Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0006</td>
<td>1600</td>
<td>OUT2</td>
<td>A; Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0007</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H; Load 00H into the HP register</td>
</tr>
<tr>
<td>0008</td>
<td>0D03</td>
<td>JMP</td>
<td>LOOP; Jump back the address 0004H</td>
</tr>
</tbody>
</table>

Figure 7-12-1 Test Program 12

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>11FF</td>
<td>LDAI</td>
<td>#FF; Load the accumulator with FFH</td>
</tr>
<tr>
<td>0001</td>
<td>1300</td>
<td>STA</td>
<td>R0; Load the register 0 with the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>1100</td>
<td>LDAI</td>
<td>#00H; Load the accumulator with 00H</td>
</tr>
<tr>
<td>0003</td>
<td>0000</td>
<td>LOOP</td>
<td>NOP</td>
</tr>
<tr>
<td>0004</td>
<td>0AFF</td>
<td>XOR</td>
<td>A,R0; XOR the accumulator with the register 0</td>
</tr>
<tr>
<td>0005</td>
<td>1500</td>
<td>OUT1</td>
<td>A; Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0006</td>
<td>1600</td>
<td>OUT2</td>
<td>A; Load the accumulator into the output register II</td>
</tr>
</tbody>
</table>
0007  1E00  LDHI  #00H: Load 00H into the HP register
0008  0D03  JMP   LOOP: Jump back the address 0003H

Figure 7-13-1 Test Program 13

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>11FF</td>
<td>LDAI #FF; Load the accumulator with FFH</td>
</tr>
<tr>
<td>0001</td>
<td>1300</td>
<td>STA R0; Load the register 0 with the accumulator</td>
</tr>
<tr>
<td>0002</td>
<td>11FE</td>
<td>LDAI #FF; Load the accumulator with FEH</td>
</tr>
<tr>
<td>0003</td>
<td>1301</td>
<td>STA R1; Load the register 1 with the accumulator</td>
</tr>
<tr>
<td>0004</td>
<td>1100</td>
<td>LOOP: LDAI #00H; Preset the accumulator with 00H</td>
</tr>
<tr>
<td>0005</td>
<td>1500</td>
<td>OUT1 A: Load the accumulator into the output register I</td>
</tr>
<tr>
<td>0006</td>
<td>1600</td>
<td>OUT2 A: Load the accumulator into the output register II</td>
</tr>
<tr>
<td>0007</td>
<td>08FF</td>
<td>OR A,R0: OR the accumulator with the register 0</td>
</tr>
<tr>
<td>0008</td>
<td>06FE</td>
<td>ANDI A,R1: AND the accumulator with the register 1</td>
</tr>
<tr>
<td>0009</td>
<td>1500</td>
<td>OUT1 A: Load the accumulator into the output register I</td>
</tr>
<tr>
<td>000A</td>
<td>1600</td>
<td>OUT2 A: Load the accumulator into the output register II</td>
</tr>
<tr>
<td>000B</td>
<td>1E00</td>
<td>LDHI #00H: Load 00H into the HP register</td>
</tr>
<tr>
<td>000C</td>
<td>0D04</td>
<td>JMP   LOOP: Jump back the address 0004H</td>
</tr>
</tbody>
</table>

Figure 7-14-1 Test Program 14

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1E00</td>
<td>LDHI #00H; Load the HP register with 00H</td>
</tr>
<tr>
<td>0001</td>
<td>1166</td>
<td>LDAI #66H; Preset the accumulator with 66H</td>
</tr>
<tr>
<td>0002</td>
<td>1066</td>
<td>LOOP3 CMPI A,#66H; Compare the accumulator with 66H</td>
</tr>
<tr>
<td>0003</td>
<td>0000</td>
<td>NOP</td>
</tr>
</tbody>
</table>
0004 0000  NOP
0005 0F08  JE   LOOP1: Produce a jump when the flag Z is high
0006 1166  LDAI #66H; Preset the accumulator with 66H
0007 0D09  JMP  LOOP2: Jump to the address 0009H
0008 11FF  LDAI #FFH; Preset the accumulator with FFH
0009 1500  LOOP2: OUT1 A: Load the output register I with the accumulator
000A 0D02  JMP  LOOP3: Jump back the address 0002H

Figure 7-15-1 Test Program 15

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H; Load the HP register with 00H</td>
</tr>
<tr>
<td>0001</td>
<td>1166</td>
<td>LDAI</td>
<td>#66H; Preset the accumulator with 66H</td>
</tr>
<tr>
<td>0002</td>
<td>1066</td>
<td>LOOP3</td>
<td>CMPI A,#66H; Compare the accumulator with 66H</td>
</tr>
<tr>
<td>0003</td>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>0F08</td>
<td>JNE</td>
<td>LOOP1: Produce a jump when the flag Z is low</td>
</tr>
<tr>
<td>0006</td>
<td>11FF</td>
<td>LDAI</td>
<td>#FFH; Preset the accumulator with FFH</td>
</tr>
<tr>
<td>0007</td>
<td>0D09</td>
<td>JMP</td>
<td>LOOP2: Jump to the address 0009H</td>
</tr>
<tr>
<td>0008</td>
<td>1166</td>
<td>LOOP1</td>
<td>LDAI #66H; Preset the accumulator with 66H</td>
</tr>
<tr>
<td>0009</td>
<td>1500</td>
<td>LOOP2</td>
<td>OUT1 A: Load the output register I with the accumulator</td>
</tr>
<tr>
<td>000A</td>
<td>0D02</td>
<td>JMP</td>
<td>LOOP3: Jump back the address 0002H</td>
</tr>
</tbody>
</table>

Figure 7-16-1 Test Program 16

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H; Preset HP register with 00H</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>CODE</td>
<td>LABEL</td>
<td>LANGUAGE</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>0000</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H: Preset the HP register with 00H</td>
</tr>
<tr>
<td>0001</td>
<td>11FF</td>
<td>LOOP2</td>
<td>#FFH: Preset the accumulator with FFH</td>
</tr>
<tr>
<td>0002</td>
<td>1500</td>
<td>OUT1</td>
<td>A: Load the output register1 with the accumulator</td>
</tr>
<tr>
<td>0003</td>
<td>1805</td>
<td>CALL</td>
<td>LOOP1: Jump to the address 0005H</td>
</tr>
<tr>
<td>0004</td>
<td>0D01</td>
<td>JMP</td>
<td>LOOP2: Jump back the address 0001H</td>
</tr>
<tr>
<td>0005</td>
<td>1100</td>
<td>LOOP1</td>
<td>#00H: Preset the accumulator with 00H</td>
</tr>
<tr>
<td>0006</td>
<td>1500</td>
<td>OUT1</td>
<td>A: Load the output register1 with the accumulator</td>
</tr>
<tr>
<td>0007</td>
<td>1900</td>
<td>RET</td>
<td>Return to the address 0004H</td>
</tr>
</tbody>
</table>

**Figure 7-17-1 Test Program 17**

**Figure 7-18-1 Test Program 18**
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H: Preset the HP register with 00H</td>
</tr>
<tr>
<td>0001</td>
<td>2001</td>
<td>SET</td>
<td>I: Set the flag I</td>
</tr>
<tr>
<td>0002</td>
<td>1100</td>
<td>LOOP1: LDAI</td>
<td>#00H: Preset the accumulator with 00H</td>
</tr>
<tr>
<td>0003</td>
<td>1500</td>
<td>OUT1</td>
<td>A: Load the output register1 with the accumulator</td>
</tr>
<tr>
<td>0004</td>
<td>0D02</td>
<td>JMP</td>
<td>LOOP1; Jump back the address 0002H</td>
</tr>
</tbody>
</table>

The following program is for the interrupt

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0007</td>
<td>11FF</td>
<td>LDAI</td>
<td>#FFH: Preset the accumulator with FFH</td>
</tr>
<tr>
<td>0008</td>
<td>1500</td>
<td>OUT1</td>
<td>A: Load the output register1 with the accumulator</td>
</tr>
<tr>
<td>0009</td>
<td>2500</td>
<td>RET</td>
<td>I: Return to the address where is interrupted</td>
</tr>
</tbody>
</table>

Figure 7-19-1 Test Program 19

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1E00</td>
<td>LDHI</td>
<td>#00H: Preset the HP register with 00H</td>
</tr>
<tr>
<td>0001</td>
<td>14F0</td>
<td>LDMI</td>
<td>#F0H: Preset the timer with F0H</td>
</tr>
<tr>
<td>0002</td>
<td>2401</td>
<td>SET</td>
<td>M: Set the flag M</td>
</tr>
<tr>
<td>0003</td>
<td>1100</td>
<td>LOOP1: LDAI</td>
<td>#00H: Load the accumulator with 00H</td>
</tr>
<tr>
<td>0004</td>
<td>1500</td>
<td>OUT1</td>
<td>A: Load the output register1 with the accumulator</td>
</tr>
<tr>
<td>0005</td>
<td>0D03</td>
<td>JMP</td>
<td>LOOP1; Jump back the address 0003H</td>
</tr>
</tbody>
</table>

The following is the timer program

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>11FF</td>
<td>LDAI</td>
<td>#FFH: Load the accumulator with FFH</td>
</tr>
<tr>
<td>0011</td>
<td>1500</td>
<td>OUT1</td>
<td>A: Load the output register 1 with the accumulator</td>
</tr>
<tr>
<td>0012</td>
<td>14F0</td>
<td>LDMI</td>
<td>#F0H: Load the timer with F0H</td>
</tr>
<tr>
<td>0013</td>
<td>2600</td>
<td>RETM</td>
<td>: Return to the address interrupted</td>
</tr>
</tbody>
</table>

Figure 7-20-1 Test Program 20
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1E00</td>
<td></td>
<td>LDHI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#00H; Preset the HP register with 00H</td>
</tr>
<tr>
<td>0001</td>
<td>2001</td>
<td></td>
<td>SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>I; Set the flag I</td>
</tr>
<tr>
<td>0002</td>
<td>1100</td>
<td>LOOP1</td>
<td>LDAI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#00H; Preset the accumulator with 00H</td>
</tr>
<tr>
<td>0003</td>
<td>1500</td>
<td>OUT1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A; Load the output register with the accumulator</td>
</tr>
<tr>
<td>0004</td>
<td>0D02</td>
<td>JMP</td>
<td>LOOP1; Jump back the address 0002H</td>
</tr>
</tbody>
</table>

The following is the interrupt program

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0007</td>
<td>1B00</td>
<td>HLT</td>
<td>Stop the clock</td>
</tr>
</tbody>
</table>

Figure 7-21-1 Test Program 21

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>LABEL</th>
<th>LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1E00</td>
<td></td>
<td>LDHI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#00H; Preset the HP register with 00H</td>
</tr>
<tr>
<td>0001</td>
<td>11FF</td>
<td>LOOP</td>
<td>LDAI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#FFH; Preset the accumulator with FFH</td>
</tr>
<tr>
<td>0002</td>
<td>2A00</td>
<td>STP</td>
<td>:Load the PSW register with the accumulator</td>
</tr>
<tr>
<td>0003</td>
<td>2900</td>
<td>LDP</td>
<td>:Load the accumulator with the PSW register</td>
</tr>
<tr>
<td>0004</td>
<td>1500</td>
<td>OUT1</td>
<td>A: Load the output register with the accumulator</td>
</tr>
<tr>
<td>0005</td>
<td>1100</td>
<td>LDAI</td>
<td>#00H; Preset the accumulator with 00H</td>
</tr>
<tr>
<td>0006</td>
<td>2A00</td>
<td>STP</td>
<td>:Load the PSW register with the accumulator</td>
</tr>
<tr>
<td>0007</td>
<td>2900</td>
<td>LDP</td>
<td>:Load the accumulator with the PSW register</td>
</tr>
<tr>
<td>0008</td>
<td>1500</td>
<td>OUT1</td>
<td>A: Load the output register with the accumulator</td>
</tr>
<tr>
<td>0009</td>
<td>0D01</td>
<td>JMP</td>
<td>LOOP; Jump back the address 0001H</td>
</tr>
</tbody>
</table>

Figure 7-22-1 Test Program
Figure 7-10-2 Simulation Result
Figure 7-11-2 Simulation Result
Figure 7-15-2 Simulation Result
Figure 7-20-2 Simulation Result
Chapter 8

Logic Circuits

This chapter describes every logic circuit in this processor. All logic circuits are level-clocked so they can alleviate clock skew and race problems.
8.1 D flip-flop [3]

Figure 8-1 shows a D master-slave flip-flop. A master-slave flip-flop is a combination of two clocked latches; the first is called the master, and the second is the slave. While the clock is high, the master is active and the slave is inactive. While the clock is low, the master is inactive and the slave is active. Therefore, the D flip-flop fetches an input signal during the negative half cycle of the clock and changes its output after the clock goes high. The D flip-flop can be controlled without considering the clock by using the SET and CLEAR signal. When the SET signal is low, the two latches will become to high immediately. The two latches will become to low when the CLEAR signal is low.

8.2 16-bit Register I

Figure 8-2 shows a 16-bit register. This register is for MEM32 circuit. There are 16 D flip-flops in this register. Each D flip-flop's output is connected a three-state inverter. While the ENC signal is high, the register cannot be used to store and output data. While the ENC is low, and ENI is high, the register will store data from input signal (DI0–DI15) during negative half cycle of the clock. The register will show up its contents when the ENC and ENO are both low.

8.3 16-bit Register II

Figure 8-3 shows the other 16-bit register. This register is for SUB-1 circuit. Its structure is almost same with the other but there is not the ENC signal in this register and the outputs from DO10 to DO15 are not controlled by the ENO signal.
8.4 8-bit Register I

Figure 8-4 shows an 8-bit register. This register is for MEM32 circuit. Its structure is same with the 16-bit register I, but just it has 8 D flip-flop inside the register.

8.5 8-bit Register II

Figure 8-5 shows the other 8-bit register. This register is for the ADDRESS REGISTER, REGISTER D, REGISTER B, REGISTER C, INPUT REGISTER, two OUTPUT Registers, and the accumulator circuits. Its structure is same with the 16-bit register II, but it just has 8 D flip-flops inside the register.

8.6 6-bit Register

Figure 8-6 shows a 6-bit register. This register is for the CONTROL UNIT circuit. Its structure is same with the 16-bit register II, but it just has 6 D flip-flops inside the register.

8.7 4-bit Register

Figure 8-7 shows a 4-bit register. This register is for the STACK POINTER circuit. Its structure is same with the 6-bit register, but it just has 4 D flip-flop inside the register.

8.8 Instruction Register

Figure 8-8 shows an instruction register. This register is for the CONTROL UNIT circuit. Normally, this register fetches an instruction from external memory. While the INT signal is high, the register will fetch 39 instead of external memory. While the TIMER signal is high, the register will fetch 40 instead of external memory.

8.9 MEM32

Figure 8-9 shows the MEM32 circuit. This memory circuit is the general registers. There are 32 8-bit registers in this circuit so the CPU has 32
general registers. The signals A0~A4 are used to select the general register wanted.

8.10 Accumulator Circuit

Figure 8-10 shows the accumulator circuit. The following is the explanation for S1, S2, and S3 signals.
1. While the S0 is high, the accumulator will shift right one bit.
2. While the S1 is high, the accumulator will shift left one bit.
3. While the S2 is high, the accumulator will store the data from the input signal.

Therefore, only one of S0, S1, and S3 signals can be high at once.

8.11 PSW Circuit

Figure 8-11 shows the PSW circuit. This circuit is for the PSW register. It stores each flag status. The following is the explanation for some control signal:
1. While the A signal or the B signal is high, the flag C can be changed.
2. While the B signal is high, the flag C , V, and S can be changed.
3. While the C signal is high, the flag Z can be changed.
4. While the D signal is high, the flag M can be changed.
5. While the E signal is high, the flag I can be changed.
6. While the ENI is high, every flag will be loaded with D0~D5. This signal is for the STP instruction.

8.12 IT Circuit

Figure 8-12 shows an IT circuit. This circuit is for CONTROL UNIT circuit. The function is the following:
1. The INT and TIMER signal can be masked by the flag I and M.
2. The INT signal is prior to the TIMER signal.

The following is the explanation of some signals
1. The INT signal is from external interrupt signal.
2. The I signal is the flag I.
3. The TIMER signal is from the internal timer.
4. The M signal is the flag M.

If the INT signal and the TIMER signal go high at the same time, Only the INT signal will active.

8.13 MEM16 Circuit

Figure 8-13 shows the MEM32 circuit. This memory circuit is for the stack registers. There are 16 16-bit registers in this circuit so the CPU has 16 stack registers. The signals A0~A3 are used to select the stack register wanted.

8.14 JK flip-flop [3]

Figure 8-14 shows a JK master-slave flip-flop. Its structure is almost same with the D master-slave flip-flop, but just it has feedback from its output.

8.15 16-bit Counter [3]

Figure 8-15 shows the 16-bit counter circuit. This counter is for the PROGRAM COUNTER. There are 16 JK flip-flops in this counter. Normally, the contents of the counter increase one by a clock cycle. When the LOAD signal is high, the counter will be loaded with a value (L0~L15).

8.16 8-bit Counter [3]

Figure 8-16 shows an 8-bit counter circuit. This counter is for the TIMER circuit. Its structure is same with the 16-bit counter, but just it has 8 JK flip-flops.

8.17 Program Counter

Figure 8-17 shows the program counter circuit. The following is explanation for some control signal:
1. While the S3 goes high, the counter will be loaded with a value (L0–L15).

2. While the S2 goes high, the counter will be loaded with 0010H (Timer interrupt).

3. While the S0 goes high, the counter will be loaded with 0007H (INT interrupt).

Therefore, only one of S0, S2, and S3 signals can go high at once.

8.18 Timer Circuit

Figure 8-18 shows the Timer circuit. When the 8-bit counter counts to zero, the Z0 signal will go high.

8.19 Full Adder, XOR, AND, and OR Circuits [3]

Figure 8-19, 8-20, 8-21, and 8-22 show a full adder, a XOR, an AND, and an OR circuit. Those circuits are for the ALU circuit.

8.20 16-bit Adder and Subtracter

Figure 8-23 shows a 16-bit adder and subtracter circuit. This circuit is for the SUB-1 circuit. There are 16 adders in this circuit. When the SUB-ADD goes high, the circuit will be a 16-bit subtracter. The circuit will be a 16-bit adder while the SUB-ADD goes low.

8.21 8-bit Adder and Subtracter

Figure 8-24 shows an 8-bit adder and subtracter circuit. This circuit is for the ALU circuit. Its structure is same with the 16-bit adder and subtracter, but just has eight adders.

8.22 Stack Pointer

Figure 8-25 shows the stack pointer circuit. While the SUB-ADD signal is high, the contents of this pointer will decrease one. In contrast, the stack pointer will increase one when the SUB-ADD signal is low. The CIN
signal controls the output of the pointer. While the CIN is high, the output will decrease one. If the CIN is low, the output will not change.

8.23 ALU

Figure 8-26 shows the ALU circuit. There are an adder and subtracter, a XOR, an OR, and an AND circuit in this circuit so the ALU can do addition, subtract, AND, OR, and XOR operations. The following is the explanation for some control signals:

1. While the S0, S1, and SUB-ADD signals are low, the ALU does add operation.
2. While the S0 and S1 signals are low, and the SUB-ADD signal is high, the ALU does subtract operation.
3. While the S0 is low and the S1 is high, the ALU does OR operation.
4. While the S0 is high, and the S1 is low, the ALU does AND operation.
5. While the S0 and S1 are high, the ALU does XOR operation.

8.24 SUB-1

Figure 8-27 shows the SUB-1 circuit. This circuit is to help the stack stores the right address when the CALL, INT, and TIMER instructions are executed. When the CPU executes the CALL instruction, the SUB-1 signal must go high in order to increase the output of the 16-bit register one. If the SUB-1 signal is low, the output will not change.

8.25 Control Unit

Figure 8-28 shows the Control Unit circuit. The three stages of the four-stage pipeline are in this circuit. They are EX1, EX2, and EX3 stages. During the EX1 stage, the Instruction Register loads an instruction from the external memory, decodes the instruction, and then sends the correct control signals out. During the EX2 stage, a 6-bit register loads an instruction from the Instruction register and then does the same job with
the EX1 stage. Finally, during the EX3 stage, a 6-bit register loads an instruction from the 6-bit register in the EX2 stage and then does the same job with the EX1 stage. It takes three clock cycles for an instruction to be loaded into the register in the EX3 stage from the external memory. In order to see the circuit clearly, this circuit is divides three parts. Each part represents a stage. Figure 8-29 shows the EX1 stage circuit. Figure 8-30 shows the EX2 stage. Figure 8-31 shows the EX3 stage.

8.26 16-bit RISC Processor [3]

Figure 8-32 shows the 16-bit RISC processor circuit. The following is the explanation for every symbol numbered:

M1. Stack Pointer
M2. 16 Stack registers (MEM16 circuit)
M3. Program Counter
M4. ROM for test program (not part of the circuit).
M7. Address Register
M8. General Registers (MEM32 circuit)
M9. Control Unit
M10. HP Register (8-bit Register II)
M11. Accumulator
M12. ALU
M13. Register B (8-bit Register II)
M14. Timer
M15. PSW Register
M16. Output Register I (8-bit Register II)
M17. Output Register II (8-bit Register II)
M18. Input Register (8-bit Register II)
M19. Register D (8-bit Register II)
M20. Register C (8-bit Register II)

M21. INT and Timer Circuit (IT circuit)

M22. SUB-1 circuit

On the right side of each symbol numbered is output pins, and on the left side is input pins. There are four kinds of buses inside the circuit. The following shows each bus and symbols connected to the bus.

1. BUS A: Output data of M9, M7 input, M20 input, M14 input, M3 input, M10 input and output, M2 output, M15 input.

2. BUS B: M8 output, M20 output, M19 input, M13 input.

3. BUS C: M19 output, M11 input, M12 output. M18 output, M15 output.

4. BUS D: M11 output, M8 input, M16 input, M17 input, M15 input.

The BUS A is used during the EX1 stage, the BUS B is used during the EX2 stage, and the BUS C and the BUS D are used during the EX3 stage.

In order to see the circuit clearly, this circuit is divided into four parts that are Figure 8-33, 8-34, 8-35, 8-36.
D FLIP-FLOP

Figure 8.1 D flip-flop
16-BIT REGISTER

Figure 8.2 16-bit Register I

CLR: ACTIVE LOW
CLK: 0-FETCH 1-OUTPUT
END: ACTIVE LOW
ENC: ACTIVE HIGH
8-BIT REGISTER

Figure 8.4: 8-bit Register
6-BIT REGISTER

CLR: ACTIVE LOW
CLK: 0-FETCH 1-OUTPUT
ENI: ACTIVE HIGH
ENO: ACTIVE LOW

WAN-FU CHEN
30, JAN, 94

Figure 8-6 6-bit Register
INSTRUCTION REGISTER

Figure 8-8 Instruction Register
ACCUMULATOR

S2: ACTIVE HIGH -> LOAD DATA
S1: ACTIVE HIGH -> SHIFT LEFT
S0: ACTIVE HIGH -> SHIFT RIGHT

ENI: ACTIVE HIGH
ENO: ACTIVE LOW

Figure 8-10 Accumulator

WAN-FU CHEN
30, JAN, 94
Figure 8-11 PSW Register
INT AND TIMER CIRCUIT

CLRI

INT

CLK

TIMER

CLRM

VCC

D FLIP-FLOP

CLK

CLR

Q

SET

RQ

INTO

TO

CLRI: ACTIVE LOW
CLRM: ACTIVE HIGH
INT, I, TIMER, M, INTO, TO: ACTIVE HIGH

Figure 8-12 INT and Timer Circuit
JK FLIP-FLOP

Figure 8-14 JK flip-flop

WAN-FU CHEN
30, JAN, 94
8-BIT COUNTER

LOAD: ACTIVE HIGH

SET: ACTIVE LOW

Figure 8-16 8-bit Counter

WAN-FU CHEN
30, JAN, 94
S2: ACTIVE HIGH -> LOAD ADDRESS FROM THE STACK OR REGISTER 5
S1: ACTIVE HIGH -> LOAD ADDRESS OF TIMER INTERRUPT
SO: ACTIVE HIGH -> LOAD ADDRESS OF INT

Figure 8-17 Program Counter
L,OAD: ACTI,VE HI,GH
SET: ACTI,VE LO, W
ZO/: IF AO-A7 = 0 THEN ZO=1

WAN-FU CHEN
30, JAN, 94

Figure 8-18 Timer Circuit
Figure 8-19 Full Adder

FULL ADDER

A B CIN

COUT SUM

WAN-FU CHEN
30, JAN. 94
XOR

Figure 8-20 XOR Circuit
Figure 8-21 AND Circuit
Figure 8-22 OR Circuit
Figure 8-23 16-bit Adder and Subtractor

WAN-FU CHEN

30, JAN, 94
ADDER AND SUBTRACTER

Figure 8-24 8-bit Adder and Subtractor

WAN-FU CHEN
30, JAN, 94
STACK POINTER

Figure 8-25 Stack Pointer

CLR: ACTIVE LOW
CLK: 0-FETCH 1-OUTPUT
ENI: ACTIVE HIGH
SUB-ADD: 0-INCREASE ONE 1-DECREASE ONE
Figure 8-26 ALU Circuit

SO SI
0 0: ADDER OR SUBTRACTER
0 1: OR
1 0: AND
1 1: XOR

ENI: ACTIVE HIGH

SUB-ADD: 1: SUBTRACTER 0: ADDER

WAN-FU CHEN
30, JAN, 94
Figure 8-27 SUB-1 Circuit

SUB-1

SUB-1: ACTIVE LOW

adder and subtractor 16-bit

16-bit register
Figure 8-30 EX2 Stage of the Control Unit
Figure 8-31 EX3 Stage of the Control Unit
Figure 8-34 Part II of the Processor
Figure 8-35 Part III of the Processor
Figure 8-36 Part IV of the Processor
Chapter 9

VLSI Design

This chapter describes VLSI design by using CMOS technology. The delay time of each circuit is calculated, and then the execution speed of this CPU chip was computed. Finally, how to speed up this CPU chip are presented. The size of this chip is 10535μm x 14677μm. There are 24,982 transistors on this chip.

All circuits in this processor are built by using inverters, 2-input NAND gates, 3-input NAND gates, 6-input NAND gates, 2-input NOR gates, 3-input NOR gates, 6-input NOR gates, and Tri-state inverters. These gates can easily be extended to build more complex logic gates. Figure 9-1 shows the inverter circuit. Figure 9-2 shows the Tri-state inverter. Figure 9-3 shows the three-input NOR circuit. Figure 9-4 shows the two-input NOR gate. Figure 9-5 shows the six-input NAND circuit. Figure 9-6 shows the six-input NOR circuit. Figure 9-7 shows the three-input NAND circuit. Figure 9-8 shows the two-input NAND circuit.

9.2 Delay Time of Each Circuit [5]

Table 9-1 lists the delay time of each circuit. During the IF stage, the longest delay time is 1.26ns of the Program Counter. During the EX1 stage, the maximum delay time is 18.19 ns when the Stack stores the data from the SUB-1 circuit. During the EX2 stage, the longest delay time is 10.07 ns while the data is transferred from the General Registers to the Register D or the Register B. During the EX3 stage, the longest delay time is 15.69 ns while the General Register is loaded the data from the accumulator. Therefore, the period of the half cycle of the clock must be greater than 18.19 ns. The processor can run about 16.67MHz while the period of the clock is 60 ns. The performance is 16.67 million instructions per second (MIPS).

9.3 Speed up the Processor [4] [5]

The following are some ways to speed up the processor:

1. Using dynamic CMOS logic circuits instead of static CMOS logic circuit.
2. Using the look-ahead carry technology to speed up the 16-bit Adder and Subtractor circuit and the 8-bit Adder and Subtractor circuit.

3. Adding some powerful drivers on the inputs of Stack and General Register circuits.

4. Trying to reduce the length of buses as soon as possible.

After done this improvement, the CPU could be run at 40 MHz.

*Table 9-1 Delay Time of each Circuits*

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Delay Time( ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D flip-flop</td>
<td>0.68</td>
</tr>
<tr>
<td>JK flip-flop</td>
<td>0.64</td>
</tr>
<tr>
<td>8-bit Register II</td>
<td>2.56</td>
</tr>
<tr>
<td>8-bit Register I</td>
<td>2.98</td>
</tr>
<tr>
<td>16-bit Register I</td>
<td>4.32</td>
</tr>
<tr>
<td>16-bit Register II</td>
<td>3.9</td>
</tr>
<tr>
<td>6-bit Register</td>
<td>2.19</td>
</tr>
<tr>
<td>4-bit Register</td>
<td>1.8</td>
</tr>
<tr>
<td>MEM16</td>
<td>4.32</td>
</tr>
<tr>
<td>MEM32</td>
<td>2.98</td>
</tr>
<tr>
<td>Full adder</td>
<td>Carry(0.52), SUM(1.4)</td>
</tr>
<tr>
<td>4-bit adder and subtracter</td>
<td>Carry(2.79), SUM(2.1)</td>
</tr>
<tr>
<td>8-bit adder and subtracter</td>
<td>Carry(4.87), SUM(2.1)</td>
</tr>
<tr>
<td>16-bit adder and subtracter</td>
<td>Carry(9.03), SUM(2.1)</td>
</tr>
<tr>
<td>SUB-1</td>
<td>12.73</td>
</tr>
<tr>
<td>Decoder</td>
<td>2.42</td>
</tr>
<tr>
<td>Stack Pointer</td>
<td>4.39</td>
</tr>
<tr>
<td>ALU</td>
<td>Adder and Subtracter(8.35), AND, OR, and XOR (4.335)</td>
</tr>
<tr>
<td>-----</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>Instruction Register</td>
<td>5.28</td>
</tr>
<tr>
<td>16-bit counter</td>
<td>1.38</td>
</tr>
<tr>
<td>8-bit Counter</td>
<td>1.34</td>
</tr>
<tr>
<td>Program Counter</td>
<td>1.26</td>
</tr>
<tr>
<td>PSW Register</td>
<td>1.67</td>
</tr>
<tr>
<td>INT and Timer</td>
<td>1.91</td>
</tr>
<tr>
<td>Control Unit</td>
<td>EX1(18), EX2(10), EX3(15.7)</td>
</tr>
</tbody>
</table>
THREE STATE INVERTER

Figure 9-2 Tre-state Inverter

WAN-FU CHEN
30, JAN, 94
NOR 3

Figure 9-3 Three-input NOR

Cin = 27.86 ff

WAN-FU CHEN
30, JAN, 94
NAND6

Figure 9-5: Six-input NAND

Cin = 34.81

A B C D E F
Figure 9-7 Three-input NAND
Conclusion

The processor has been designed and simulated successfully. The RISC processor executes every instruction in a clock cycle even if it executes JUMP, Condition Jump, CALL, and Return instructions. This is the advantage of this processor in comparison with other RISC processors.

The most difficult part of this thesis is the Control Unit design because the Control Unit includes three stages of the pipeline such as stages EX1, EX2, and EX3. A designer must know how to arrange an instruction into each stage without any delay of executing other instructions. According to the simulation and the calculation, the processor can run at 16.67 MHz. The performance is 16.67 million instructions per second (MIPS). The size of this chip is 10535\(\mu\)m X 14677\(\mu\)m, and there are 24,982 transistors on this chip that consumes 200 mw.

Finally, the following are some suggestions to speed up the processor and reduce the size of this chip:

1. Use dynamic CMOS logic instead of static CMOS logic.
2. Use a smaller feature size, for example 0.8 \(\mu\)m instead of 2 \(\mu\)m. It will shrink the size considerably.
3. Use the look-ahead carry technique to speed up the Adder and Subtracter circuits.
4. Add drivers on the inputs of Stack and General Register circuits.
5. Reduce the length of buses as much as possible to increase the bandwidth.

These suggested improvements may increase the CPU speed to the rate of 40 MHz.
References


