|  |
| --- |
| rochester institute of technology |
| My Smart Little Computer |
| ISA Document |
|  |
| **Jenny Zhen & Grant Kurtz** |
| **April 9th, 2014** |

|  |
| --- |
| The Instruction Set Architecture document specifies all programmer visible portions of the simulated computer. Programmer visible sections are the instruction types, address modes, instruction encoding, general purpose registers, and real instructions available. |

Table of Contents

[Introduction 2](#_Toc384759653)

[Architecture 2](#_Toc384759654)

[Main Memory Requirements 2](#_Toc384759655)

[Programmer-Usable Registers 2](#_Toc384759656)

[Internal CPU Registers 2](#_Toc384759657)

[Internal Busses 2](#_Toc384759658)

[The Instruction Set 3](#_Toc384759659)

[Instruction Layouts 3](#_Toc384759660)

[Address Modes Supported 3](#_Toc384759661)

[List of Instructions 3](#_Toc384759662)

[Halting 4](#_Toc384759663)

[Areas for Improvement 4](#_Toc384759664)

# Introduction

The purpose of this document is to specify the programmer visible portions of a simulated computer to test and demonstrate the effectiveness of branch prediction on a pipelined RISC architecture. The instruction set is kept simple and small but with a large enough variability to create a complete testing environment. Instructions are limited to only a single byte in width so as to keep the pipeline itself simple, to minimize the complexity necessary to add branch prediction to the pipeline.

# Architecture

## Main Memory Requirements

The word and data size is 12 bits, and the address size is 8 bits. This means that there are a total of 256 bytes of total memory, hex 00 to FF.

## Programmer-Usable Registers

There are four programmer visible registers.

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **Name** | **Size** | **Notes** |
| R0, R1, R2, R3 | Reg 0, Reg 1, Reg 2, Reg 3 | 12 | These are the four registers that a user can directly manipulate with the assembly code |

## Internal CPU Registers

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **Name** | **Size** | **Notes** |
| PC | Program Counter | 8 | Location of instruction to execute. |
| IR | Instruction Register | 12 | Register to hold the instruction while it is executing. |

## Internal Busses

|  |  |  |  |
| --- | --- | --- | --- |
| **Bus** | **Name** | **Size** | **Notes** |
| ABus | Address Bus | 8 | For transferring effective addresses to memory |
| DBus | Data Bus | 12 | For transferring data between memory and the general purpose registers. |

# The Instruction Set

## Instruction Layouts

All instructions have the same general format, and are 12-bits, or one byte long.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| R-Type | 11..8 OPC | 7..6 RD | 5..4 RS | 3..2 RT | 1..0 N/A |
| I-Type | 11..8 OPC | 7..6 RS | 5..4 RT | 3..0 IMM | |
| J-Type | 11..8 OPC | 7..0 IMM | | | |

|  |  |  |
| --- | --- | --- |
| **Field** | **Size** | **Notes** |
| OPC | 4-bits | The opcode for this instruction. |
| RD | 2-bits | The destination register for R-type instructions. |
| RS | 2-bits | The first source register for R-type instructions and the destination register for I-type instructions. |
| RT | 2-bits | The source register for both R-type and I-type instructions. |
| N/A | 2-bits | The last two bits of R-type instructions are ignored. |
| IMM | 3-bits or 8-bits | The immediate contains information for use in the address mode. |

## Address Modes Supported

Only one address mode is supported and that is Displacement Mode. For I-type instructions, the RT register is the base, and the immediate field is the displacement from the base. Since addresses are only 8-bits, if the immediate plus the RT register exceeds the size of 8-bits, the overflow is truncated.

## List of Instructions

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | **Mnemonic** | **Description** | **RTL** |
| 0000 | NOP | No operation |  |
| 0001 | ADD | Add RS and RT, store the result in RD | RD = RS + RT |
| 0010 | ADDI | Adds RT and the immediate, storing the result in RS | RS = RT + IMM |
| 0011 | AND | Bitwise AND of RS and RT into RD | RD = RS & RT |
| 0100 | OR | Bitwise OR of RS and RT into RD | RD = RS | RT |
| 0101 | XOR | Bitwise XOR of RS and RT into RD | RD = RS ⊕ RT |
| 0110 | SLL | Logical shift left of RS by RT into RD | RD = RS << RT |
| 0111 | SRL | Logical shift right of RS by RT into RD | RD = RS >> RT |
| 1000 | SRA | Logical arithmetic shift right of RS by RT into RD | RD = RS >>a RT |
| 1001 | SB | Stores RS into memory at location of AM | Mem[ RT + IMM ] ←RS |
| 1010 | LB | Loads from memory at location of AM into RS | RS ← Mem[ RT + IMM ] |
| 1011 | SLT | Sets RD to 1 if RS < RT, otherwise RD to 0 | If RS < RT then RD = 1, else RD = 0 |
| 1100 | BEZ | Branch to location EA if RS is 0 | If RS == 0 then PC = EA |
| 1101 | BNE | Branch to location EA if RS is not zero | If RS != 0 then PC = EA |
| 1110 | JUMP | Jump to memory location specified by EA | PC = EA |
| 1111 | HALT | Halt the computer |  |

# Halting

Aside from halting after executing a HALT instruction, the CPU will also need to stop when it attempts to execute past the end of the legal memory addresses (check this by checking PC overflow).

When that happens, the simulator should print the message “MACHINE HALTED due to”, plus the reason (with the following message):

|  |  |
| --- | --- |
| **Reason** | **Message** |
| PC overflow | MACHINE HALTED due to PC overflow |
| halt instruction | MACHINE HALTED due to halt instruction |

# Areas for Improvement

There are a few areas in this document that could be improved or altered slightly.

For R-Type instructions there are two unused bits. R-Type instructions could be modified to support four more instructions, but it would not add much for testing the pipeline or branch prediction.

The internal registers are assumed to be duplicated between each stage of the pipeline as necessary. Details for those sections were purposefully left out as it is not necessary until the hardware is fully decided upon.

Only one address mode is supported, which is more than sufficient for testing the effectiveness of branch prediction as whether or not a jump is taken is all that is necessary, rather than how the effective address used to make the jump was generated.