Design Document

Zeen Wang, Jermaine Brown,

Helen Wang, Athena Henderson

Team: Yellow-2122a-01

Rose-Hulman Institute of Technology

CSSE 232 Computer Architecture I

Prof. Williamson

April 3, 2022

Link to the Google Doc *DesignDocument*

Description:

Our design uses a single register accumulator to store data and compare it to inputs. At all times we only use one register and use an allocated space in Memory for data. For our addresses we will use sign extensions to target specific places in memory to receive either data for destination. The input must have the correct first bit to target the proper place in memory.

We are going to use 2 registers, the accumulator($acc) and stack pointer($sp). The accumulator is the only register available by the programmer.

I:

Opcode	Immediate	Unused
5	8	3

AI:

Opcode	Address	Immediate
5	8	3

PC relative for bne and beq

A:

Opcode	Address	Unused
5	8	3

We left shift by 1 bit then we sign extent (the most significant bit will be 1 if it is a data and 0 if it is a instruction)

Instructions

Name	Type	Operation	Opcode

load a:
Takes an 8 bit address a and loads the value at memory address a to the accumulator, using the address rule.

save a:
Take an 8 bit address a and save the value in the accumulator into the memory with address a, using the address rule.

loadui:
Takes an 8 bit immediate and load it to the upper 8 bits of the accumulator

beq a imm:
Takes an 8 bit address and a 3 bit immediate. If the value stored at address a is equal to the value of the accumulator, then jump to the address calculated from the immediate using the branch address rule.

bne a imm:
Takes an 8 bit address and a 3 bit immediate. If the value stored at address a is not equal to the value of the accumulator, then jump to the address calculated from the immediate using the branch address rule.

slt a:

Compare the value in the accumulator with the value stored at address a, if the accumulator is less than a then we set the accumulator to 1, else we set the accumulator to 0.

slti imm:
Compare the value in the accumulator with the immediate, if the accumulator is less than the immediate then we set the accumulator to 1, else we set the accumulator to 0.

j a:
Jump to the instruction with address a, calculated using the address rule.

jal a:
Jump to the instruction with address a, calculated using the address rule. Store the current PC + 2 to a fix memory location.

sw imm:
Stored the value in the accumulator in to the stack where it is of offset imm to the stack pointer.

lw imm:
Stored the value from stack where it is of offset imm to the stack pointer to the accumulator.

sub a:
Subtract the value stored at address a from the accumulator and store the result in the accumulator

add a:
Add the value stored at address a to the accumulator and store the result in the accumulator

addi imm:
Add the sign extended immediate to the accumulator and store the result in the accumulator

and a:
And the value stored at address a to the accumulator and store the result in the accumulator

or a:

Or the value stored at address a to the accumulator and store the result in the accumulator

ori imm:
Or the zero extended immediate to the accumulator and store the result in the accumulator

Address rule: We left shift by 1 bit then we sign extent (the most significant bit will be 1 if it is a data and 0 if it is a instruction)
Branch address: Left shift the immediate by 1, sign extend it to 16 bits then add it to the value of the current PC plus 2.

Types

A:

Opcode	Direct Address (rt)	Unused
15 11	10 3	2 0

I:

Opcode	Immediate	Unused
15 11	10 3	2 0

AI:

Opcode	Direct Address (rt)	Immediate
15 11	10 3	2 0

Assembly Sheet

Name	**Type**	**Operation**	**Opcode**
load	A	acc = rt	00001
save	A	Mem[getAddr(rt)] = acc	00010
loadui	I	acc = {imm, 8b’0}	00011

bne	AI	if(acc != Mem[getAddr(rt)])
		PC = PC + 2 + getAddr(imm)
			00100

beq	AI	if(acc == Mem[getAddr(rt)])
		PC = PC + 2 + getAddr(imm)
			00101

slt	A	acc = acc < Mem[getAddr(rt)] ? 1:0	00110
slti	I	acc = acc < SignExtent(imm) ? 1:0	00111
j	A	PC = getAddr(rt)	01000

jal	I	Men[ra] = PC + 2
		PC = getAddr(imm)
			01001

sw	I	sp + SignExtent(imm) = acc	01010
lw	I	acc = sp + SignExtent(imm)	01011
ms	I	sp = sp + SignExtent(imm)	01100
sub	A	acc = acc - Mem[getAddr(rt)]	01101
add	A	acc = acc + Mem[getAddr(rt)]	01110
addi	I	acc = acc + SignExtent(imm)	01111
and	A	acc = acc & Mem[getAddr(rt)]	10000
or	A	acc = acc | Mem[getAddr(rt)]	10001
ori	I	acc = acc | ZeroExtent(imm)	10010
loadi	I	acc = SignExtent(imm)	10011

getAddr = {7{address[7]}, address, 1’b0}
ZeroExtent = {8b’0, imm}
SignExtent = {8{address[7]},imm}
ra = 0xFFFE
sp = 0x1FFF			

Call procedure

We will store the arguments in memory and the callee will read data directly from memory. If we need to use the arguments after the call we will store their value on stack. The return value will be passed back via the accumulator

Example program(s)

High Level Code	Assembly	Machine Code	Addresses
int **relPrime**(int n)	loadi 2	10011 00000010 000	0x 0030
{	save m	00010 10000011 000	0x 0032
int m;	ms -12	01100 11110110 000	0x 0034
	loop: load m	00001 10000011 000	0x 0036
m = 2;	sw 0	01010 00000000 000	0x 0038
while (gcd(n, m) != 1) {	save a	00010 10000000 000	0x 003A
m = m + 1;	load n	00001 10000100 000	0x 003C
}	sw 4	01010 00000100 000	0x 003E
return m;	save b	00010 10000001 000	0x 0040
}	load ra	00001 11111111 000	0x 0042
	sw 8	01010 00001000 000	0x 0044
	jal gcd	01001 11100111 000	0x 0046
	save o	00010 10000101 000	0x 0048
	lw 0	01011 00000000 000	0x 004A
	save m	00010 10000011 000	0x 004C
	lw 4	01011 00000100 000	0x 004E
	save n	00010 10000100 000	0x 0050
	lw 8	01011 00001000 000	0x 0052
	save ra	00010 11111111 000	0x 0054
	loadi 1	10011 00000001 000	0x 0056
	bne o, end	00100 10000101 100	0x 0058
	load m	00001 10000011 000	0x 005A
	add 1	01110 00000001 000	0x 005C
	save m	00010 10000011 000	0x 005E
	j loop	01000 00011011 000	0x 0060
	end: ms 12	01100 00001100 000	0x 0062
	j ra	01000 11111111 000	0x 0064

int **gcd**(int a, int b) {	gcd:				
if (a == 0) {		loadi	0	10011 00000000 000	0x 0002
return b;		bne	a, loop	00100 10000000 010	0x 0004
}		load	b	00001 10000001 000	0x 0006
		j	ra	01000 11111111 000	0x 0008
while (b != 0) {	loop:				0x 000A
if (a > b) {		loadi	0	10011 00000000 000	0x 000C
a = a - b;		bne	b, go	00100 10000001 001	0x 000E
} else {		j	end	01000 00010110 000	0x 0010
b = b - a;	go:	load	b	00001 10000001 000	0x 0012
}		slt	a	00110 10000000 000	0x 0014
}		save	i	00010 10000010 000	0x 0016
		loadi	1	10011 00000001 000	0x 0018
return a;		bne	i, else	00100 10000010 011	0x 001A
}		load	a	00001 10000000 000	0x 001C
		sub	b	01101 10000001 000	0x 001E
		save	a	00010 10000000 000	0x 0020
		j	loop	01000 00000101 000	0x 0022
	else:	load	b	00001 10000001 000	0x 0024
		sub	a	01101 10000000 000	0x 0026
		save	b	00010 10000001 000	0x 0028
		j	loop	00010 00000101 000	0x 002A
	end:	load	a	00001 10000000 000	0x 002C
		j	ra	00010 11111111 000	0x 002E

Data:		Stack:
0xFF00	a(value = m)	0x1FFF
0xFF02	b(value = n)	
0xFF04	i	
0xFF06	m	
0xFF08	n	
0xFF0A	o	

Team repo: set upped, goto link in M1 and join yellow