Lebron Mk. 2

Team: V-2223c-02

Aidan Kirk, Dalton Bumb, Avichal Jadeja, Vishrut Patwari

Link to Google Document
https://docs.google.com/document/d/16eSBUKxdq_vlAOVXJIrhp27hVrMAWL0CEv2cxJjvVYw/edit?usp=sharing

Design

Our design will use a register-stack based architecture. There will be two stacks. A data stack and a procedure stack. The data stack is for operations inside of procedures, and the procedure stack for saving arguments/return values when executing procedure calls. Our motivation for using the register-stack architecture is to create a stack that strikes an even balance between efficient subroutine calls and local variable storage. The stack will also enable operands and intermediate results to be more dynamic in nature. This architecture will enable us to retain simplicity while not compromising on overall efficiency.

Measure Of Performance

- Execution time
- Short instruction syntax

Registers

Since all the registers make up the stack, there are no registers directly available to the programmer. The only way for the programmer to access registers is to perform operations on the stack.

Procedure Call Conventions

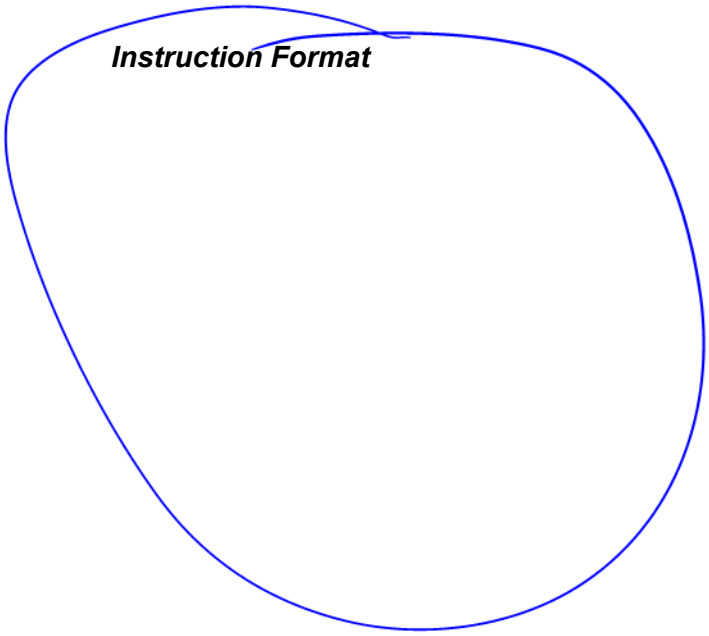
When executing a procedure call, it is the caller's responsibility to place input arguments on the procedure stack (a0 at top and a1 at next… etc) and it is the callee's responsibility to place its return value on the top of the procedure stack and not alter any of the arguments passed in on the procedure stack.

Addressing Modes

Instructions that implement jump use PC relative addressing, an example would be jnz. With PC relative addressing, the immediate is the instruction offset shifted left one then sign extended. For instructions that involve immediate values we use immediate addressing. An example would

push or pop. This simply sign extends the immediate and stores it on the stack. Majority of Instructions that manipulate the stack utilize direct addressing. An example of this would be the swap instruction. The immediate field in this instruction is not used.

Instruction Format



Procedure Stack Visualization

arg1
arg2
…more args…
ra (return address)

Data Stack *Visualization*

Top
Next
Other data

ISA Table

Instruction Name	***Type***	***Opcode***	**Behavior**	***Description***
Push	P	000010	Top = addr	Places a value from memory onto the top of the data stack
Pushi	P	100011	Top = imm[9:0]	Places an immediate onto the top of the data stack
Pop	P	000011	Addr = Top Top = next	Removes Top from the data stack and places it in the specified memory address
Peek	P	000101	Addr = Top	Saves top of data stack to specified

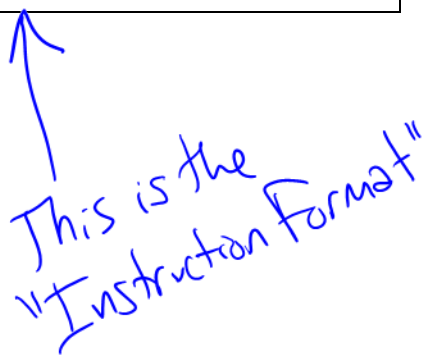
				memory address. (does not remove top)
jz	P	001101	Pop a If a = 0, jump to label	Pops top of data stack. If top is equal to 0, then jump to label otherwise, fall through.
jnz	P	010011	Pop a If a != 0, jump to label	Pops top of data stack. If top is not equal to 0, then jump to label otherwise, fall through.
jmp	P	001110	pc = a	Unconditional jump
lup	P	010001	Push Big[15-6] Shift left 6	Loads upper 10 of large immediate
lli	P	010010	Push Big[5-0]	Loads lower 6 of large immediate
drop	A	111101	Top = next	Delete item at top of stack
Sub	A	000110	Top = next - top	Pops two items from data stack, subtract them, then put on top of data stack
Add	A	000111	Top = next + top	Pops two items from data stack, add them, then put on top of data stack
Swap	A	001000	a = Top b = next next = a Top = b	Pops two items from the data stack, and pushes them back to the data stack in the order they were popped.

Dup	A	001001	a = Top push a	Duplicates the item at the top of the data stack and pushes it to the data stack
eq	A	001010	Top = 0/1	Pushes a 1 to the top of the data stack if top and next are equal, 0 if not. (Doesn't pop anything)
lt	A	001011	Top = 0/1	Pushes a 1 to the top of the data stack if next < top. 0 Otherwise
le	A	010111	Top = 0/1	Pushes a 1 to the top of the data stack if next <= top. 0 otherwise
geq	A	001100	Top = 0/1	Pushes a 1 to the top of the data stack if next >= top. 0 otherwise
store	A	001111	Top[procs] = a	Pops the top of the data stack and pushes the value onto the procedure stack. This does remove the top from the data stack.
ret	A	010000	a = Top[procs]	Pops the top of the procedure stack and pushes the value into the top of the data stack. The does

				remove the item at the top of the procedure stack.
swapproc	A	111111	In procedure stack: a = Top b = next next = a Top = b	Swaps top and next in the procedure stack
dupproc	A	111110	a = Top push a	Duplicates the top value in the procedure stack and pushes it back on

Assembly to Machine Code

Instruction Type	***15 12***	***11 6***	***5 4***	***3 0***
P-type	Imm[9:0]		opcode	
A-type	xxxxxxxxx		opcode	



Memory Map

Stack(0XFFFF FFFF) Grows downwards
Code(0X0000 4000) Grows upwards
Globals(0X0000 0000) Grows upwards

needs to be bounded (min/max addresses)

Machine Language Translation

temp = 0x0000 0002
start = 0x0000 0004

Rel Prime Code to Assembly

Address	***RelPrime Assembly Code***	***Machine Code***	***Comments***
0x4000	PROGRAM:		
0x4002	pushi 2	***0000000010*** **100011**	*// push 2 onto data stack*
0x4004	dup	***0000000000 001001***	*// duplicate the top of the stack (2 in this case)*
0x4006	store	***0000000000 001111***	*// store 2 to the procedure stack*
0x4008	swapproc	***0000000000 111111***	*// swap the top 2 elements in the procedure stack*
0x400A	GCD:		
0x400C	dupproc	***0000000000 111110***	*// duplicate top item in procedure stack*
0x400E	ret	***0000000000*** **010000**	*// retrieve item from top of procedure stack and push it into data stack*
0x4010	pushi 0	***0000000000*** **100011**	*// push 0 onto the data stack*
0x4012	eq	**0000000000 001010**	*// pushes one onto data stack if top two items are equal*
0x4014	jz WHILE	**0000001000 001101**	*// jump 4 instructions (8 bytes) if a != 0*
0x4016	swap	***0000000000*** **001000**	*// swap items at the top of data stack*
0x4018	store	***0000000000*** **001111**	*// store top of data stack to top of procedure stack*
0x401A	jmp RELPRIME	***0000110110*** **001110**	*// jump to RELPRIME, forward 27 instructions, or 54 bytes*

0x401C	WHILE:		
0x401E	drop	***0000000000*** **111101**	*//* delete value from top of data stack
0x4020	swapproc	***0000000000*** **111111**	*// stop top two items at top of procedure stack*
0x4022	dupproc	***0000000000*** **111110**	*// duplicate item at the top of procedure stack*
0x4024	ret	***0000000000*** **010000**	*// retrieve value from top of procedure stack and push into data stack*
0x4026	pushi 0	***0000000000*** **100011**	*// push 0 onto data stack*
0x4028	eq	***0000000000*** **001010**	*// push 1 onto stack if top two values are equal*
0x402A	jnz RETURN	***0000011110*** **010011**	*// jump to return if top of stack is not zero*
0x402C	drop	***0000000000*** **111101**	*// delete top value from data stack*
0x402E	le	***0000000000*** **010111**	*// push 1 onto data stack if second value is <= first value*
0x4030	jnz ANOTGREATER	***0000001010*** **010011**	*// got ANOTGREATER if top of stack is not 0*
0x4032	dup	***0000000000*** **001001**	*// duplicate top value of the data stack*
0x4034	pop temp	***0000000010*** **000011**	*// remove and store top value of stack to address temp*
0x4036	sub	***0000000000*** **000110**	*// subtract the top element of the data stack from the second element of the stack. Original*

			values have been removed from stack.
0x4038	push temp	***0000000010 000010***	*// push data at address temp to top of data stack*
0x403A	ANOTGREATER:		
0x403C	swap	***0000000000*** **001000**	*// swap items at the top of data stack*
0x403E	dup	***0000000000*** **001001**	*// duplicate top value of the data stack*
0x4040	pop temp	***0000000010*** **000011**	*// remove and store top value of stack to address temp*
0x4042	sub	***0000000000*** **000110**	*// subtract the top element of the data stack from the second element of the stack. Original values have been removed from stack.*
0x4044	push temp	***0000000010 000010***	*// push data at address temp to top of data stack*
0x4046	swap	***0000000000*** **001000**	*// swap items at the top of data stack*
0x4048	jmp WHILE	***1111010100*** **001110**	*// jump to WHILE, back 22 instructions, or 44 bytes*
0x404A	RETURN:		
0x404C	store	***0000000000*** **001111**	*// store top of data stack to top of procedure stack*
0x404E	jmp RELPRIME	***0000000020*** **001110**	*// jump to RELPRIME, 2 bytes forward.*
0x4050	RELPRIME:		
0x4052	LOOP:		

0x4054	ret	***0000000000*** **010000**	*// retrieve value from top of procedure stack and push into data stack*
0x4056	pushi 1	***0000000001*** **100011**	*// push 1 onto the data stack*
0x4058	eq	**0000000000 001010**	*// push 1 onto stack if top two values are equal*
0x405A	jnz FINISH	***0000001110*** **010011**	*// got FINISH if top of stack is not 0*
0x405C	ret	***0000000000*** **010000**	*// retrieve value from top of procedure stack and push into data stack*
0x405E	pushi 1	***0000000001*** **100011**	*// push 1 onto the data stack*
0x4060	add	***0000000000*** **000111**	*// add the first two items on the stack together. These two items are removed from the stack.*
0x4062	store	***0000000000*** **001111**	*// store top of data stack to top of procedure stack*
0x4064	swap	***0000000000*** **001000**	*// swap items at the top of data stack*
0x4066	jmp GCD	***1110100100*** **001110**	*// jump to GCD (back 46 instructions, or 92 bytes)*
0x4068	FINISH:		
0x406A	store	***0000000000*** **001111**	*// store top of data stack to top of procedure stack*
0x406C	ret	***0000000000*** **010000**	*//* transfer n to stack
0x406E	ret	***0000000000*** **010000**	*//* transfer m to stack
0x4070	ret	***0000000000*** **010000**	*// transfer*

0x4072	pop start	***0000000100* 000011**	*// pop ra to save for jump*
0x4074	drop	***0000000000* 111101**	*// get rid of m*
0x4076	push start	***0000000100* 000010**	*// push ra back onto stack to be stored back into procedure stack*
0x4078	store	***0000000000* 001111**	*// store ra back into procedure stack*
0x407A	store	***0000000000* 001111**	*// store n back into procedure stack*

how to return?

Basic Operations

Assume:
b = 0x0000
c = 0x0002
a = 0x0004
i = 0x0006

Add two numbers:

High Level Code:
a = b + c;

Address	***Recursive Assembly Code***	***Machine Code***	***Comments***
0x4000	PROGRAM:		
0x4002	push b	***0000000000 000010***	*// push value at address b onto data stack*
0x4004	push c	***0000000010 000010***	*// push value at address c onto the top of the data stack*
0x4006	add	***0000000000* 000111**	*// add the first two items on the data stack together. These*

			two items are removed from the stack.
	pop a	***0000000100* 000011**	*// pop top of the data stack to address a*

High Level Code:
a = b + 5;

Address	***RelPrime Assembly Code***	***Machine Code***	***Comments***
0x4000	PROGRAM:		
0x4002	push b	***0000000000 000010***	*// push value at address b onto data stack*
0x4004	pushi 5	***0000000101* 100011**	*// push the immediate 5 to the top of the data stack*
0x4006	add	***0000000000* 000111**	*// add the first two items on the data stack together. These two items are removed from the stack.*
	pop a	***0000000100* 000011**	*// pop top of the data stack to address a*

Subtract two numbers:

High Level Code:
a = b - c;

Address	***RelPrime Assembly Code***	***Machine Code***	***Comments***
0x4000	PROGRAM:		
0x4002	push b	***0000000000 000010***	*// push value at address b onto data stack*

0x4004	push c	***0000000010 000010***	*// push value at address c onto the top of the data stack*
0x4006	sub	***0000000000*** **000110**	*// subtract the top element of the data stack from the second element of the stack. Original values have been removed from stack.*
	pop a	***0000000100*** **000011**	*// pop top of the data stack to address a*

<u>High Level Code:</u>
a = b - 5;

Address	***RelPrime Assembly Code***	***Machine Code***	***Comments***
0x4000	PROGRAM:		
0x4002	push b	***0000000000 000010***	*// push value at address b onto data stack*
0x4004	pushi 5	***0000000101*** **100011**	*// push the immediate 5 to the top of the data stack*
0x4006	sub	***0000000000*** **000110**	*// subtract the top element of the data stack from the second element of the stack. Original values have been removed from stack.*
0x4008	pop a	***0000000100*** **000011**	*// pop top of the data stack to address a*

Simple if statements:

<u>High Level Code:</u>
if (b == 0) {
b = 3 + b
}

a = b + c

Address	***RelPrime Assembly Code***	***Machine Code***	***Comments***
0x4000	PROGRAM:		
0x4002	pushi 0	***0000000000*** **100011**	*// push the immediate 0 to the top of the data stack*
0x4004	push b	***0000000000 000010***	*// push value at address b onto data stack*
0x4006	eq	***0000000000*** **001010**	*// push 1 onto the data stack if top two values are equal*
0x4008	jz SkipBody	***0000000100*** **001101**	*// pop top of the stack to address a*
0x400A	pushi 3	***0000000011*** **100011**	*// push the immediate 3 to the top of the data stack*
0x400C	add	***0000000000*** **000111**	*// add the first two items on the data stack together. These two items are removed from the stack.*
0x400E	pop b	***0000000000*** **000011**	*// pop the top of the data stack to address b*
0x4010	SkipBody:		
0x4012	push c	***0000000010 000010***	*// push value at address c onto the top of the data stack*
0x4014	add	***0000000000*** **000111**	*// add the first two items on the data stack together. These two items are removed from the stack.*
0x4016	pop a	***0000000100*** **000011**	*// pop top of the data*

			stack to address a

<u>High Level Code:</u>
a = 3;
b = a + c;
if (a < b) {
// some code (not present in assembly)
}
a = b + c

Address	***RelPrime Assembly Code***	***Machine Code***	***Comments***
0x4000	PROGRAM:		
0x4002	pushi 3	***0000000011*** **100011**	*// push the immediate 3 onto the top of the data stack*
0x4004	push c	***0000000010 000010***	*// push value at address c onto the top of the data stack*
0x4006	add	***0000000000*** **000111**	*// add the first two items on the data stack together. These two items are removed from the stack.*
0x4008	dup	***0000000000*** **001001**	*// duplicate top value of the data stack*
0x400A	pop b	***0000000000*** **000011**	*// pop the top of the data stack to address b*
0x400C	lt	***0000000000*** **001011**	Pushes a 1 to the top of the data stack if next < top. 0 Otherwise
0x400E	jnz End	***0000001000*** **010011**	*// got End if top of the data stack is not 0*
0x4010	push c	***0000000110 000010***	*// push value at address c onto the*

			top of the data stack
0x4012	add	***0000000000*** **000111**	*// add the first two items on the data stack together. These two items are removed from the stack.*
0x4014	pop a	***0000000100*** **000011**	*// pop top of the data stack to address a*
0x4016	End:		

Loops:

<u>High Level Code:</u>

```
while (i > 0) {
        a = a + 1;
}
```

Address	***RelPrime Assembly Code***	***Machine Code***	***Comments***
0x4000	PROGRAM:		
0x4002	pushi 0	***0000000000*** **100011**	*// push immediate 0 to the top of the data stack*
0x4004	push i	***0000000110 000010***	*// push value at address i onto the top of the data stack*
0x4006	While:		
0x4008	le	***0000000000*** **010111**	Pushes a 1 to the top of the data stack if next <= top. 0 Otherwise
0x400A	jnz End	***0000000100*** **010011**	*// got End if top of the data stack is not 0*
0x400C	push a	***0000000100*** **000010**	*// push value at address c onto the top of the data stack*
0x400E	pushi 1	***0000000001*** **100011**	*// push immediate 1 to the top of the data*

			stack
0x4010	add	***0000000000*** **000111**	*// add the first two items on the data stack together. These two items are removed from the stack.*
0x4012	jmp While	***1111110100*** **001110**	*// jump to while unconditionally*
0x4014	End:		