Make title page, Table of contents, executive summary.

Stack Based Architecture Design Document

Team 2v(Jinhao Sheng, Luke McNeil, Austin Swatek, Yiju Hao)

~~https://docs.google.com/document/d/1rOVMKmwiSBdTVJDZL7f2QUDD_nJh6b0vAcjDHI0XbUY/edit?usp=sharing~~

not necessary.

Your exec summary will make this clear.

1. Description of the registers available to the assembly language programmer and those reserved for any specific purpose.

We don't need any additional individual registers. We have 2 stacks of registers. One of these stacks is the main one where all of the operations are performed. The other is the return address stack which simply remembers all of the return values allowing for nested function calls.

2. An unambiguous English description of each machine language instruction format type and its semantics (see Section 2.5, pages 80 through 87 of your book).

Format Type	Size	Structure	Addressing Mode	Description
O	2 bytes	OP 4 | FUNCT 12	Direct ?	OP - basic operation of the instruction FUNCT - sets the variant of the operation This format type is used for instructions that take no arguments.
A	2 bytes	OP 4 | IMM/ADDR 12	Pseudo-Direct	OP - basic operation of the instruction IMM/ADDR - a 12 bit constant or address This format type is for any instruction that takes one argument which is either an immediate or address.

3. An unambiguous English description of the syntax and semantics of each instruction (see pages A-51 through A-80 of your book).

Name	Type	Argument	Description
add	O		Pop the top two values off the stack, add them, and put the result on the stack.
beq	A	label	Pop the top two values off the stack. If they are equal, then branch to label.
bez	A	label	Pop the top value off the stack. If it is zero then branch to label.
dup	O		Push onto the stack a duplicate of the value

			currently on top of the stack.
drop	O		Pop the top value off the stack, throwing it away.
exit	O		Exit the program.
getin	O		Read a 16 bit number from input and push it to the stack.
j	A	target	Jump to target.
jal	A	target	Jump to target, and push onto the return address stack the address of the next instruction.
js	O		Pop the top value off the stack and jump to that address.
lui	A	immediate	Shift the immediate left by 12 bits and then push it into the stack
over	O		Push onto the stack the value of the second element on the stack.
or	O		Pop the top two values off the stack, or them, and put the result on the stack.
pop	A	address	Pop the top value off the stack and put it in memory at address.
push	A	address	Push the value at the specified address onto the stack.
pushi	A	immediate	Push onto the stack sign extended immediate.
return	O		Pop the top element off the return address stack, and jump there.
slt	O		Pop the top two elements off the stack, and push a 1 to the stack if the second from the top element is less than the top element. Otherwise, push a 0.
sub	O		Pop the top two values off the stack, subtract them, and put the result on the stack.
swap	O		Swap the top two elements on the stack.

combine tables

4. **The rule for translating each assembly language instruction into machine language. This probably requires the addressing modes to be defined (i.e. is branch PC relative?).**

O Types		
Instruction	**OP**	**FUNCT**
add	0x0	0x000
dup	0x0	0x001
drop	0x0	0x002
exit	0x0	0x003
getin	0x0	0x004
js	0x0	0x005

over	0x0	0x006
or	0x0	0x007
return	0x0	0x008
slt	0x0	0x009
sub	0x0	0x00A
swap	0x0	0x00B

A Types	
Instruction	**OP**
beq	0x1
bez	0x2
j	0x3
jal	0x4
pop	0x5
push	0x6
pushi	0x7
lui	0x8

Instructions	**Format Type**	**Addressing Modes**
js, return	O	Direct
j, jal, beq, bez	A	Pseudo-Direct

Pseudo-Direct Example

- Going from 16 bit address to the 12 bits in the instruction
    a. Shift the 16 bit number right 1
    b. Chop off the 4 most significant bits
    c. Use this 12 bit number in the instruction ADDR field.
- Going from 12 bits in the instruction to a 16 bit address
    a. Shift the 12 bits to the left 1
    b. Put on the front of these 13 bits the 3 most significant bits from $PC.
    c. Use this 16 bit number as the address to go to.

Show diagram for clarity

Direct Example

- Here the jump is looking at the value in a 16 bit register. The address to jump to is simply those 16 bits.

5. **An explanation of any procedure call conventions, especially relating to register and stack use (see pages A-22 through A-26 of your book).**

To prepare to call a function the caller must put all arguments that the callee expects to receive on top of the stack. Then the caller must call jal to go to the callee. This command will push the return address to the return address stack. The callee's responsibilities are to pop the arguments off the stack and leave on the stack any return values. The callee will then do the

Show an example.

return instruction which will pop the top element off the return address stack going back to the correct spot.
Included below in the code fragments (section #7) is an example of nested function calling.

6. **Example assembly language program demonstrating that your instruction set supports a program to find relative primes using the algorithm on the project page.**

This is state-dependent not static...

RelPrime and Sample Procedure Call					
ADDR	**MC**	**LABEL**	**ASM**	**STACK**	**RETURN STACK**
0x0	0x0004	MAIN:	getin	() -> (n)	()
0x2	0x4003		jal RELPRIME	(n) -> (relprime(n))	() -> (0x4)
0x4	0x0003		exit	(relprime(n))	()
0x6	0x7002	RELPRIME:	pushi 2	(n) -> (n, m)	(0x4)
0x8	0x0006	RPLOOP:	over	(n, m) -> (n, m, n)	(0x4)
0xA	0x0006		over	(n, m, n) -> (n, m, n, m)	(0x4)
0xC	0x400F		jal GCD	(n, m, n, m) -> (n, m, gcd)	(0x4) -> (0x4, 0x8)
0xE	0x7001		pushi 1	(n, m, gcd) -> (n, m, gcd, 1)	(0x4)
0x10	0x100C		beq RETURNM	(n, m, gcd, 1) -> (n, m)	(0x4)
0x12	0x7001		pushi 1	(n, m) -> (n, m, 1)	(0x4)
0x14	0x0000		add	(n m 1) -> (n, m+1)	(0x4)
0x16	0x3004		j RPLOOP	(n, m+1)	(0x4)
0x18	0x000A	RETURNM:	swap	(n, m) -> (m, n)	(0x4)
0x1A	0x0002		drop	(m, n) -> (m)	(0x4)
0x1C	0x0007		return	(m)	(0x4) -> ()
0x1E	0x0006	GCD:	over	(n, m, a, b) -> (n, m, a, b, a)	(0x4, 0x8)
0x20	0x2020		bez RETURNB	(n, m, a, b, a) -> (n, m, a, b)	(0x4 0x8)
0x22	0x0001	LOOP:	dup	(n, m, a, b) -> (n, m, a, b, b)	(0x4 0x8)
0x24	0x2023		bez RETURNA	(n, m, a, b, b) -> (n, m, a, b)	(0x4 0x8)
0x26	0x0006		over	(n, m, a, b) -> (n, m, a, b, a)	(0x4 0x8)
0x28	0x0006		over	(n, m, a, b, a) -> (n, m, a, b, a, b)	(0x4 0x8)
0x2A	0x000A		swap	(n, m, a, b, a, b) -> (n, m, a, b, b, a)	(0x4 0x8)
0x2C	0x0008		slt	(n, m, a, b, b, a) -> (n, m, a, b, b<a)	(0x4 0x8)
0x2E	0x201B		bez ELSE	(n, m, a, b, b<a) -> (n, m, a, b)	(0x4 0x8)
0x30	0x000A		swap	(n, m, a, b) -> (n, m, b, a)	(0x4 0x8)
0x32	0x0006		over	(n, m, b, a) -> (n, m, b, a, b)	(0x4 0x8)
0x34	0x0009		sub	(n, m, b, a, b) -> (n, m, b, a-b)	(0x4 0x8)
0x36	0x000A		swap	(n, m, b, a-b) -> (n, m, a-b, b)	(0x4 0x8)
0x38	0x3011		j LOOP	(n, m, a-b, b)	(0x4 0x8)
0x3A	0x0006	ELSE:	over	(n, m, a, b) -> (n, m, a, b a)	(0x4 0x8)
0x3C	0x0009		sub	(n, m, a, b, a) -> (n, m, a, b-a)	(0x4 0x8)
0x3E	0x3011		j LOOP	(n, m, a, b-a)	(0x4 0x8)
0x40	0x000A	RETURNB:	swap	(n, m, a, b) -> (n, m, b, a)	(0x4 0x8)
0x42	0x0002		drop	(n, m, b, a) -> (n, m, b)	(0x4 0x8)
0x44	0x0007		return	(n, m, b)	(0x4, 0x8) -> (0x4)

0x46	0x0002	RETURNA:	drop	(n, m, a, b) -> (n, m, a)	(0x4, 0x8)
0x48	0x0007		return	(n, m, a)	(0x4, 0x8) -> (0x4)

7. **Assembly language fragments for common operations. For example, this might be loading an address into a register, iteration, conditional statements, reading data from the input port, reading from and writing to the display register, or writing to the output port.**

Any conventions to Follow?

Sample Return					
Int main() {return (2 + 3) - 1;}					
ADDR	**MC**	**LABEL**	**ASM**	**STACK**	**RETURN STACK**
0x0	0x7002	MAIN:	pushi 2	() -> (2)	()
0x2	0x7003		pushi 3	(2) -> (2, 3)	()
0x4	0x0000		add	(2, 3) -> (5)	()
0x6	0x7001		pushi 1	(5) -> (5, 1)	()
0x8	0x000A		sub	(5, 1) -> (4)	()
0xA	0x0003		exit	(4)	()

Loading an 16 bit address onto the stack. (load 1000 0000 0000 0001)					
ADDR	**MC**	**LABEL**	**ASM**	**STACK**	**RETURN STACK**
0x0	0x8008	MAIN:	lui 0x8	() -> (0x8000)	()
0x2	0x7001		pushi 1	(0x8000) -> (0x8000, 0x1)	()
0x4	0x0007		or	(0x8000, 0x1) -> (0x8001)	()
0x6	0x0003		exit		()

Sample For Loop					
int main(){ **int x = 1** **for(int i = 0; i < 5; i++){ x++;}**					
ADDR	**MC**	**LABEL**	**ASM**	**STACK**	**RETURN STACK**
0x0	0x7001	MAIN:	pushi 1	() -> (x)	()
0x2	0x7000		pushi 0	(x)-> (x i)	()
0x4	0x0001	LOOP:	dup	(x i) -> (x i i)	()
0x6	0x7005		pushi 5	(x i i) -> (x i i 5)	()
0x8	0x0009		slt	(x i i 5) -> (x i i<5)	()
0xA	0x7001		pushi 1	(x i i<5) -> (x i i<5 1)	()
0xC	0x1009		beq OP	(x i i<5 1) -> (x i)	()
0xE	0x0002		drop	(x)	()
0x10	0x0003		exit	(x)	()
0x12	0x7001	OP:	pushi 1	(x i) -> (x i 1)	()
0x14	0x0000		add	(x i 1) -> (x i++)	()
0x16	0x000A		swap	(x i++) -> (i++ x)	()
0x18	0x7001		pushi 1	(i++ x) -> (i++ x 1)	()

0x1A	0x0000		add	(i++ x 1) -> (i++ x++)	()
0x1C	0x000A		swap	(i++ x++) -> (x++ i++)	()
0x1E	0x3002		j LOOP	(x++ i++)	()

Return Chain					
int main() {return f1(2);} int f1(int a) {return f2(a);} int f2(int b) {return f3(b);} int f3(int c) {return c+1;}					
ADDR	**MC**	**LABEL**	**ASM**	**STACK**	**RETURN STACK**
0x0	0x7002	MAIN:	pushi 2	() -> (2)	()
0x2	0x4003		jal F1	(2)	() -> (0x4)
0x4	0x0003		exit	(3)	()
0x6	0x4005	F1:	jal F2	(2)	(0x4) -> (0x4, 0x8)
0x8	0x0007		return	(3)	(0x4) -> ()
0xA	0x4007	F2:	jal F3	(2)	(0x4, 0x8) -> (0x4, 0x8, 0xC)
0xC	0x0007		return	(3)	(0x4, 0x8) -> (0x4)
0xE	0x7001	F3:	pushi 1	(2) -> (2, 1)	(0x4, 0x8, 0xC)
0x10	0x0000		add	(2, 1) -> (3)	(0x4, 0x8, 0xC)
0x12	0x0007		return	(3)	(0x4, 0x8, 0xC) -> (0x4, 0x8)

Reading Data From the Input Port					
ADDR	**MC**	**LABEL**	**ASM**	**STACK**	**RETURN STACK**
0x0	0x0004	MAIN:	getin	() -> (input)	()
0x2	0x0003		exit	(input)	()

8. **Machine language translations of your assembly programs (relprime and your fragments).**

See in Parts 6 & 7.

M2 task assignment:

- **1st meeting: break instructions into small steps and move data from one register to another, determine single-cycle or multi-cycle**
- **Luke & Austin: RTL Description of each instruction**
  Jinhao & Yiju : A list of generic components specifications needed for RTL
- **2nd meeting: debug and test the processor through Xilinx ISE and fix existed problems**

Put in journals