Stack Based Architecture Design Document

Team 2v: Luke McNeil, Jinhao Sheng, Austin Swatek

Processor Overview: Stack-based

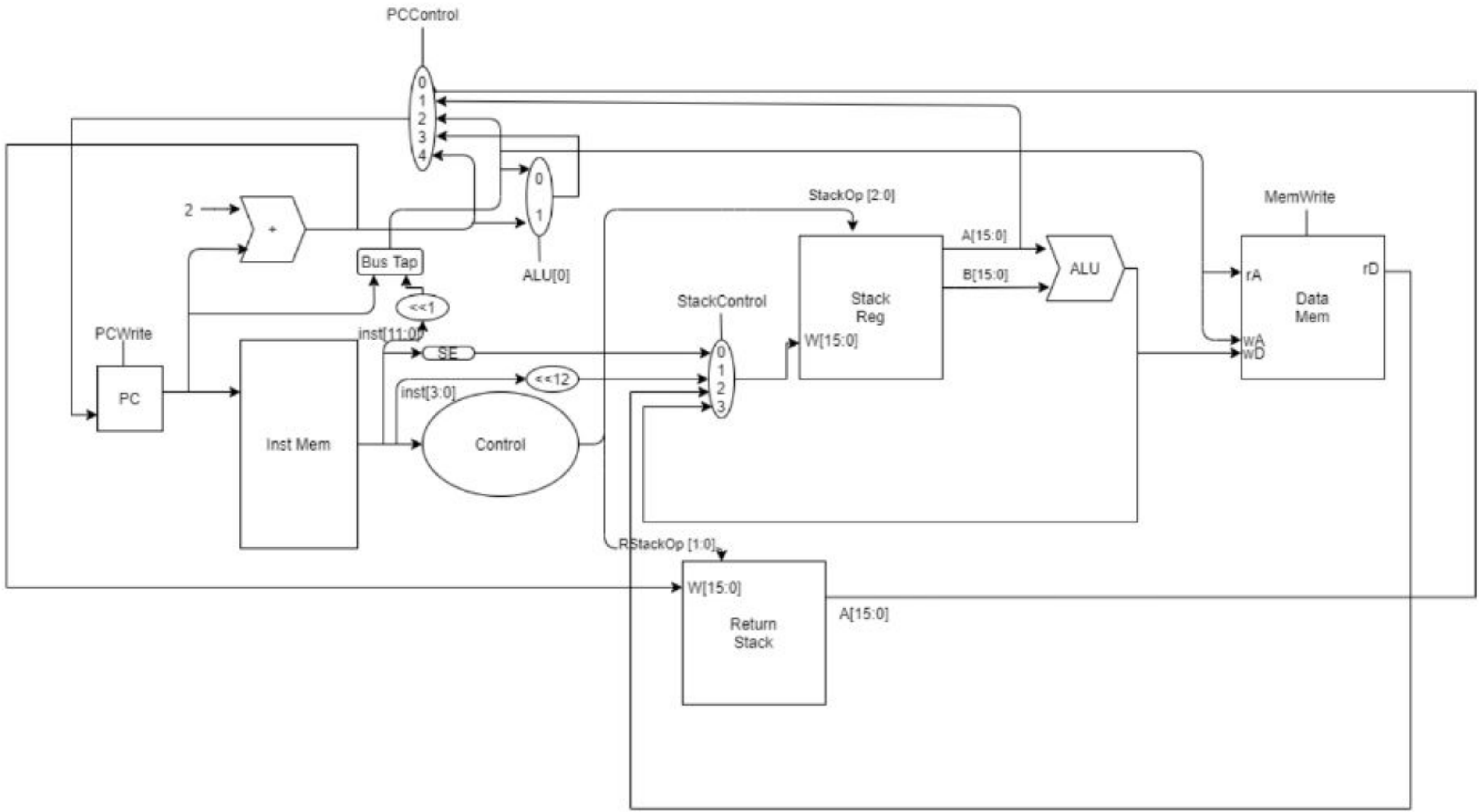
Two stacks of 64 registers:

1. Generic computations
2. Keeping track of function return values

LIFO function calling

1. Caller pushes the arguments that the callee expects to Stack 1
2. Callee pops arguments off the Stack 1
3. Callee operates, then pushes return values back to Stack 1
4. Callee pops return address off Stack 2
5. Callee jumps to return address

Design of the Datapath



Instruction Types

- O-Type
  - Add
  - Sub
  - Dup
  - Swap
- A-Type
  - Push
  - Pop
  - JAL
  - BEZ

Format Type	Size	Structure		Description
O	2 bytes	OP 4	FUNCT 12	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	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.

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

Direct:

- Given a 16-bit value from a register, the address to be jumped to is simply that 16-bit address.

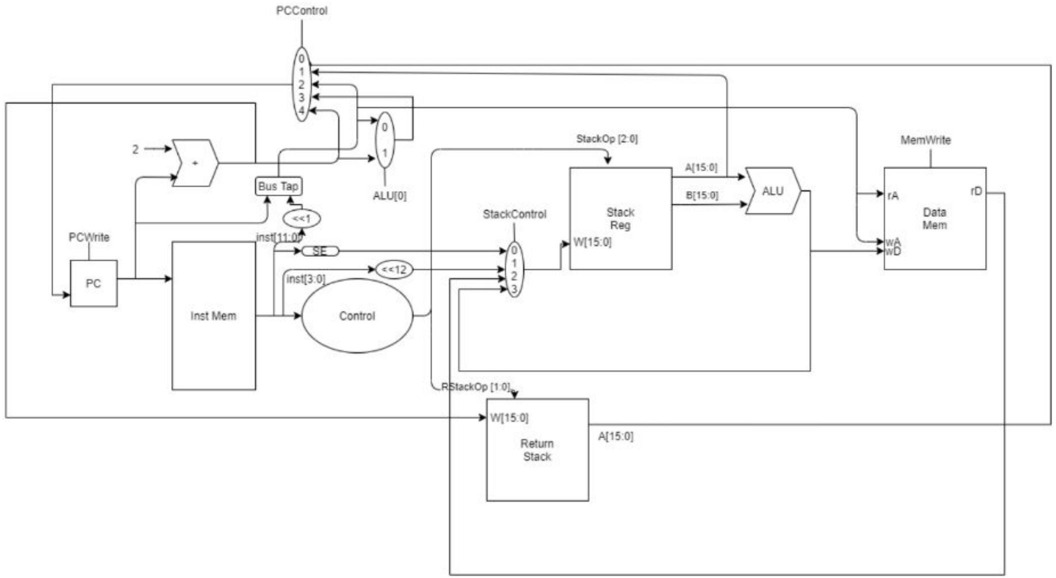
Pseudo-Direct, from 12-bit instruction to 16-bit address:

- Left-shift the 12 bits, [11:0] by one, for 13 bits total: [12:0]
- Take the 3 MSB from $PC and assign them to bits [15:13]
- This 16-bit number, [15:0], is now the address that will be jumped/branched to.

Memory

- Using Xilinx's block memory: blockmemory16kx1
- Instruction Memory
  - Block for the instructions used in program, reads in a 16-bit address and outputs the 16-bit data at that address.
  - Stores the instructions that are going to be executed.
- Data memory
  - Takes in a 16-bit read address, 16-bit write address, and 16-bit data to write. Outputs 16-bit data from read address, writes only if MemWrite control bit is set.
  - Stores the data that is pushed by the programmer.

Control bits



PCControl, 3-bit: how PC will change

StackControl, 3-bit: determines input to stack

StackOP, 3-bit: determines stack operation

RStackOP, 2-bit: nothing changes, value is pushed, or value is popped

ALUOP, 4-bit: determines ALU operation

PCWrite, 1-bit: whether or not to write to PC

MemWrite, 1-bit: whether or not to write to data memory

Assembler and Simulator

- Written in Chez Scheme
- Rel-prime example:
- Instructions are loaded in memory_small.coe

```
MAIN:
        getin
        jal RELPRIME
        halt
RELPRIME:
        pushi 2
RPLOOP:
        over
        over
        jal GCD
        pushi 1
        beq RETURNM
        pushi 1
        add
        j RPLOOP
RETURNM:
        swap
        drop
        return
GCD:
        over
        bez RETURNB
LOOP:
        dup
        bez RETURNA
        over
        over
        swap
        slt
        bez ELSE
        swap
        over
        sub
        swap
        j LOOP
ELSE:
        over
        sub
        j LOOP
RETURNB:
        swap
        drop
        return
RETURNA:
        drop
        return
```

```
0000000000000100
0100000000000011
0000000000000011
0111000000000010
0000000000000110
0000000000000110
0100000000001111
0111000000000001
0001000000001100
0111000000000001
0000000000000000
0011000000000100
0000000000001011
0000000000000010
0000000000001000
0000000000000110
0010000000100000
0000000000000001
0010000000100011
0000000000000110
0000000000000110
0000000000001011
0000000000001001
0010000000011101
0000000000001011
0000000000000110
0000000000001010
0000000000001011
0011000000010001
0000000000000110
0000000000001010
0011000000010001
0000000000001011
0000000000000010
0000000000001000
0000000000000010
0000000000001000
```

Performance of Relative-Prime

- Number of instructions in relprime - 37
- Total number of bytes to store relprime - 54

Input: 0x13B0

- Instructions: 122,357
- Cycles: 122,357
- Cycle Time: 17.44ns or 57.3Mhz
- Execution Time: 2.134 ms

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	2028	4656	43%
Number of Slice Flip Flops	2120	9312	22%
Number of 4 input LUTs	3569	9312	38%
Number of bonded IOBs	98	232	42%
Number of BRAMs	8	20	40%
Number of GCLKs	2	24	8%

Changes We Would Like To Make

- Make it multi-cycle
- Fix memory overflowing
- Add more instructions such as:
    - Non-destructive versions of add, sub, slt, and or
    - Immediate instructions like addi
    - More stack manipulation instructions

Fun Parts In Our Design

- Creating the register stack in verilog
- Being able to create such a simple datapath
- Writing programs in our instruction set