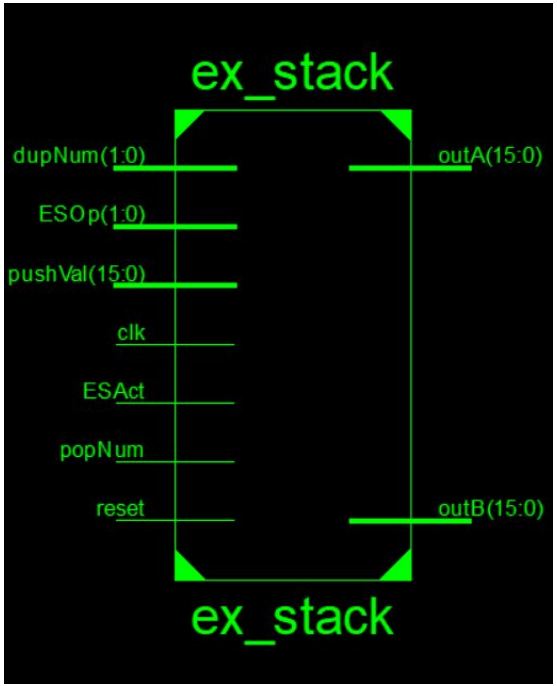
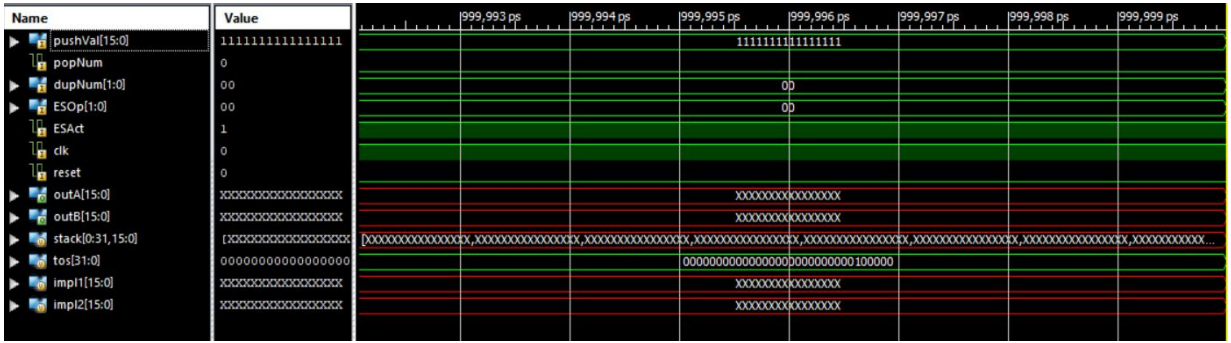
The Eggo Stack

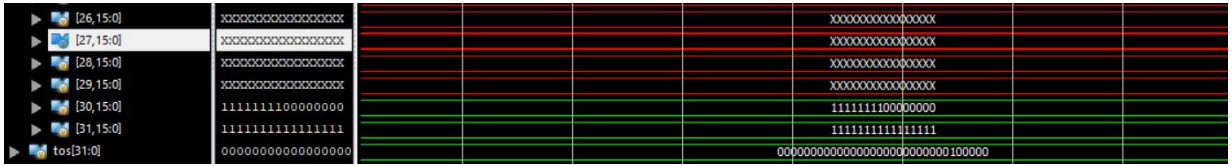
The Toaster Troop
William Dalby, Christian Meinzen, Victoria (Tori) Szalay

The Eggo Stack

Stack Architecture







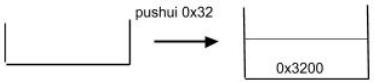
Instructions

Instruction Set

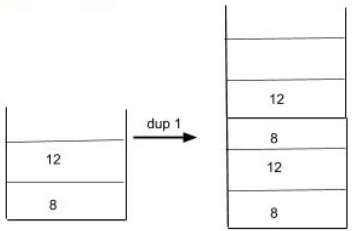
List of Commands:

Command	Opcode	Funct	Type
pushM	0x0 / 0b0000	0b00	C
popM	0x1 / 0b0001	0b00	C
pushR	0x0 / 0b0000	0b01	C
popR	0x1 / 0b0001	0b01	C
pushli	0x2 / 0b0010	n/a	C
pushui	0x3 / 0b0011	n/a	C
dup	0x4 / 0b0100	n/a	A
flip	0x5 / 0b0101	n/a	A
or	0x6 / 0b0110	n/a	A
add	0x7 / 0b0111	n/a	A
sub	0x8 / 0b1000	n/a	A
ls	0x9 / 0b1001	n/a	A
as	0xA / 0b1010	n/a	A
slt	0xB / 0b1011	n/a	A
beq	0xC / 0b1100	n/a	B
bne	0xD / 0b1101	n/a	B
j	0xE / 0b1110	n/a	B
js	0xF / 0b1111	n/a	B

6. **pushui** takes the upper 8 bits of a 16 bit immediate, zero extends it, and stores it on the top of the stack.
   ISA: C-type
   Example: **pushui 0x32**
   Visualization of the stack



7. **dup** looks at the specified amount of data from the top of the stack, copies the data, and pushes it on to the top of the stack
   ISA: A-type
   Example: **dup 1**



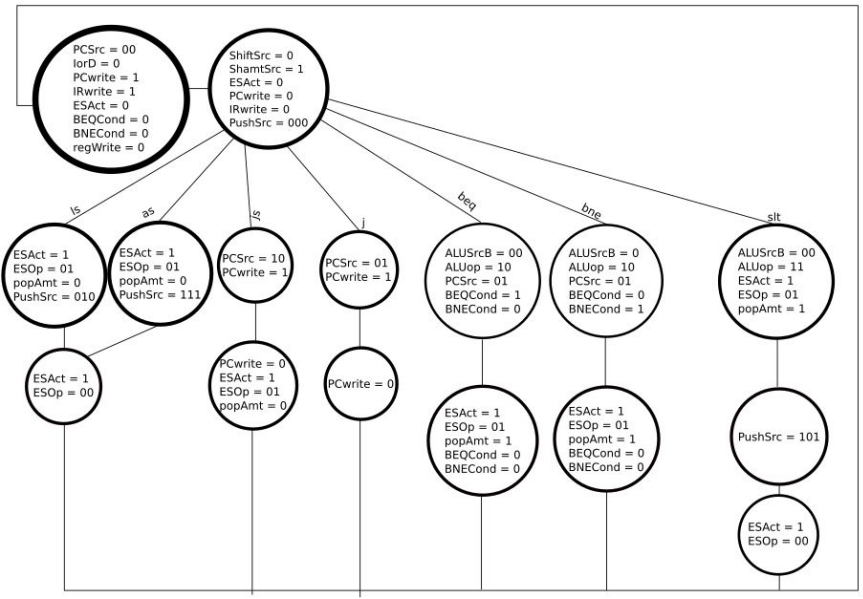
RTL Diagrams

Step	slt	lsr/ lsl	flip	dup	pushUI/LI
	newPC = PC + 2 PC = newPC Inst = Mem[PC]				
	A = ES[Top] B = ES[Top + 2]				
	ALUOut = A < B ES(pop 1)	ShiftOut = A shifted by Shamt in appropriate direction	Push A Push B	dupAMT = inst[1-0] If dupAMT == 0: ES[Top+2] = ES[Top] Top = Top+2 Done If dupAMT == 1: ES[Top+2] = ES[Top-2] ES[Top+4] = ES[Top] Top = Top+4 If dupAMT==2: ES[Top+2] = ES[Top-4] ES[Top+4] = ES[Top -2] ES[Top+6] = ES[Top] Top = Top + 6 Else: ES[Top+2] = ES[Top - 6] ES[Top+4] = ES[Top-4] ES[Top+6] = ES[Top-2] ES[Top+8] = ES[Top] Top = Top + 8	UI: ShiftOut = SE(inst[11-4]) << 8 LI: push ZE(inst[11-4])
	skip	ES(push ShiftOut)			ES(push ShiftOut)
	ES(push ALUout)				

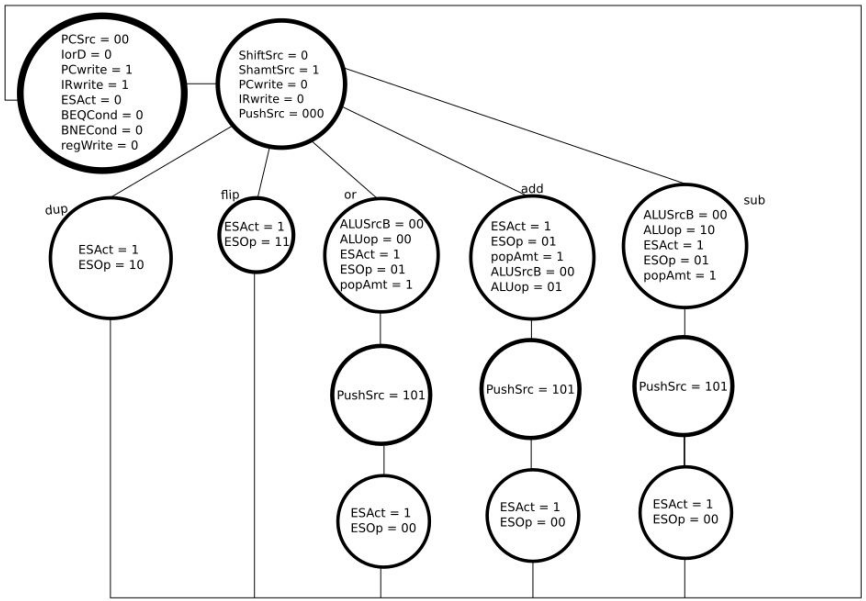
Step	Push/Pop M	Push/Pop R	Arithmetic/ Logic	beq/bne	j	js
	newPC = PC + 2 PC = newPC Inst = Mem[PC]					
	A = ES[Top] B = ES[Top + 2]					
	ALUout = A + SE(inst[11-4])	popR: Reg[inst[1-0]] = A ES(pop 0) pushR: skip	ALUout = A op B ES(pop 1)	ALUOut = A == B	PC = PC[15-11] || inst[11-0]	Address = pop PC = address
	popM: Mem[ALUout] = ES(Top + 2) pushM: ES(pop 0)	pushR: ES(push Reg[inst[1-0]])	Skip	ES(pop 1) if ALUOut == 1 then PC = PC[15-11] || inst[11-0]	skip	ES(pop 0)
	popM: DONE pushM: MemOut = Mem[ALUout]		ES(push ALUout)			
	pushM: skip					
	pushM: ES(push MemOut) DONE					

Multicycle Diagrams

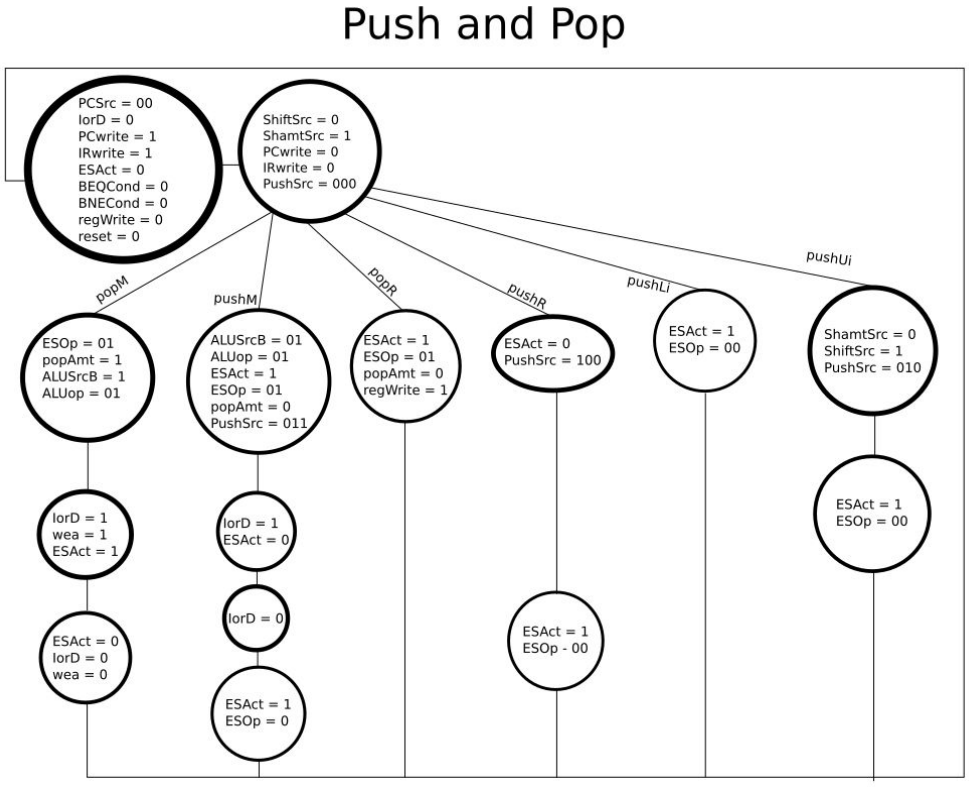
SLT and Shifting



Arithmetic Instructions

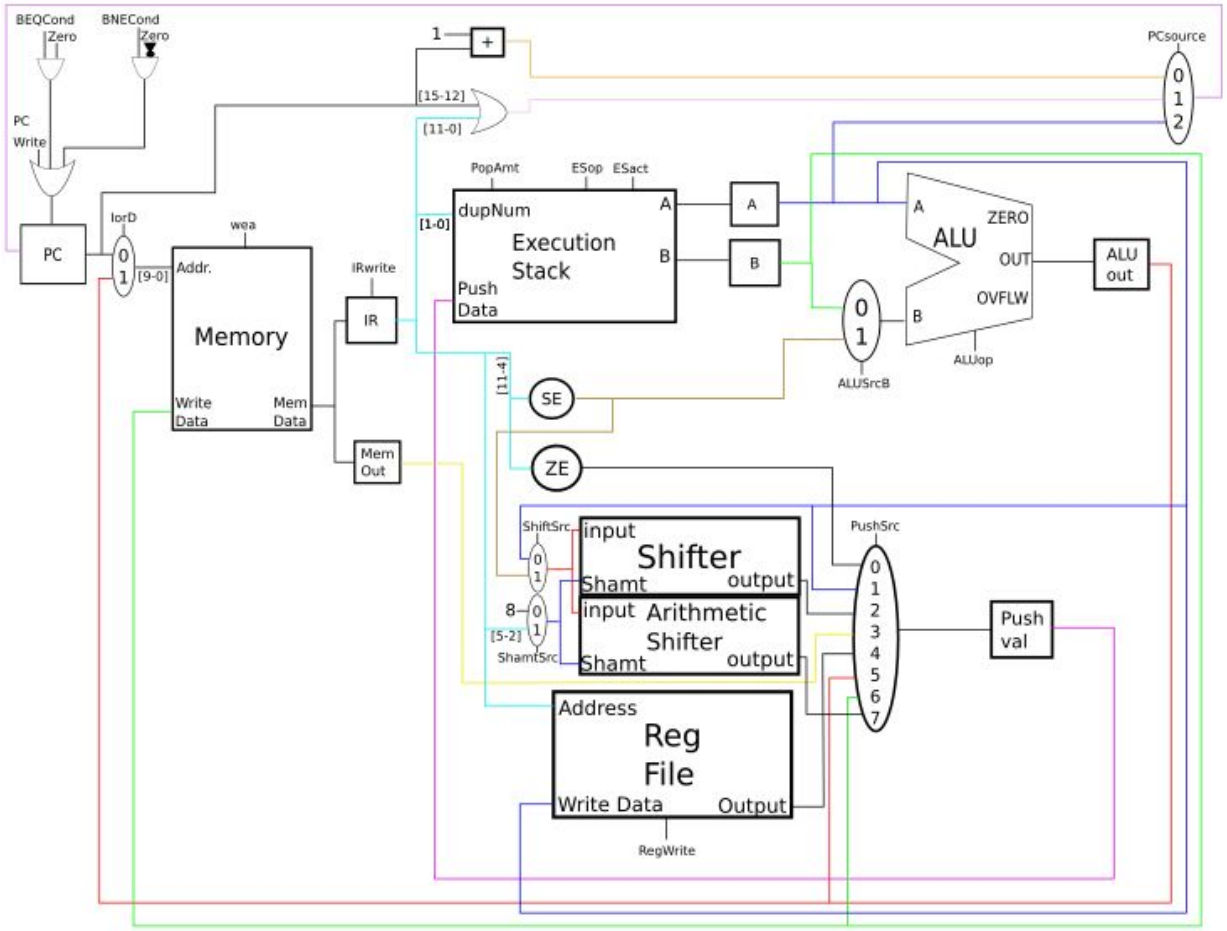


Multicycle Diagrams



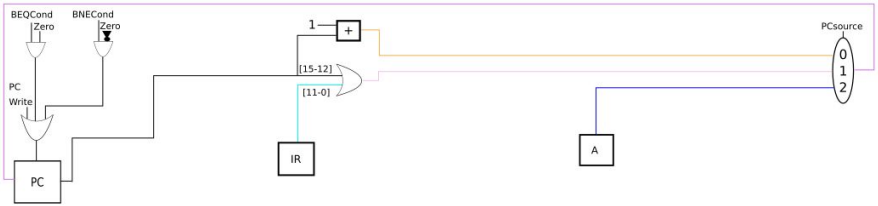
Datapath

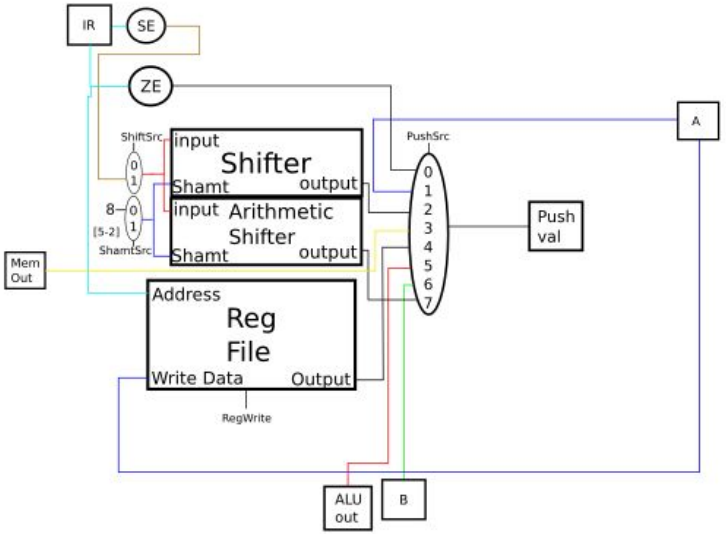
Datapath

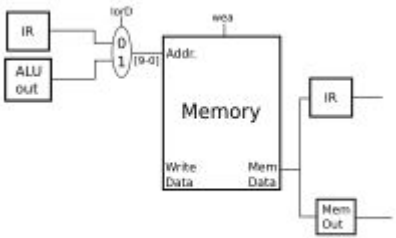

BEQCond
Zero
BNECond
Zero
PC
Write
1
+
[15-12]
[11-0]
PCsource
0
1
2
PopAmt
ESop
ESact
dupNum
Execution
Stack
Push
Data
A
B
[1-0]
A
B
ALU
A
B
ZERO
OUT
OVFLW
ALUop
ALU
out
0
1
ALUSrcB
PC
IorD
0
1
[9-0]
wea
Addr.
Memory
Write
Data
Mem
Data
IRwrite
IR
Mem
Out
[11-4]
SE
ZE
ShiftSrc
0
1
input
Shifter
Shamt
output
8
0
1
[5-2]
ShamtSrc
input
Arithmetic
Shifter
Shamt
output
PushSrc
0
1
2
3
4
5
6
7
Push
val
Address
Reg
File
Write Data
Output
RegWrite

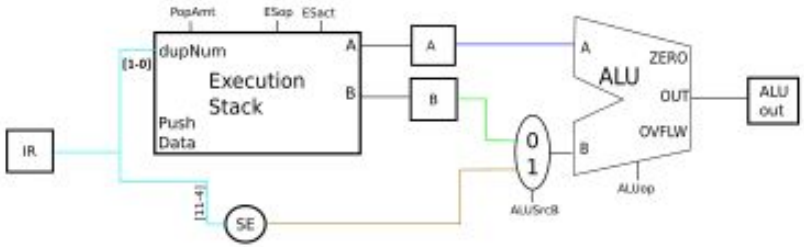
Testing

How we tested

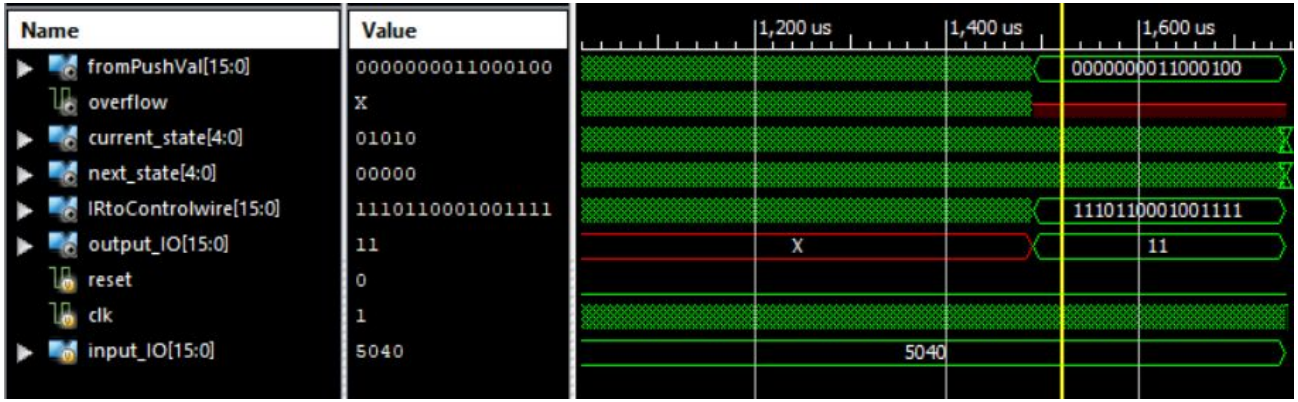








Performance



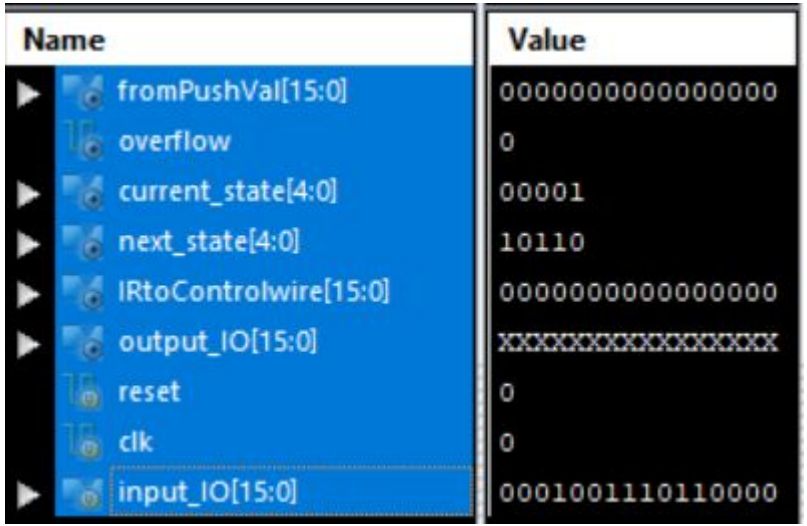
Instructions: 183774
Cycle frequency: 65.6 MHz
Cycle time: 15.23 ns
Cycles: 745206
CPI: 4.055
Execution Time: 11.35ms
Size of Program: 160 bytes

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	4407	4656	94%
Number of Slice Flip Flops	774	9312	8%
Number of 4 input LUTs	8351	9312	89%
Number of bonded IOBs	77	232	33%
Number of BRAMs	1	20	5%
Number of GCLKs	1	24	4%

Extras

Extras

- Arithmetic Shifter
- Memory Mapped I/O



The Assembler

- 3 pass
- 1st
  - Handles PseudoInstructions
    - Push
      - Pushes a 16 bit value to the stack
      - pushLi, pushUi, or
    - bge
    - ble
    - blt
    - Bgt
- 2nd
  - Handles addresses and labels
- 3rd
  - Converts to Machine Code

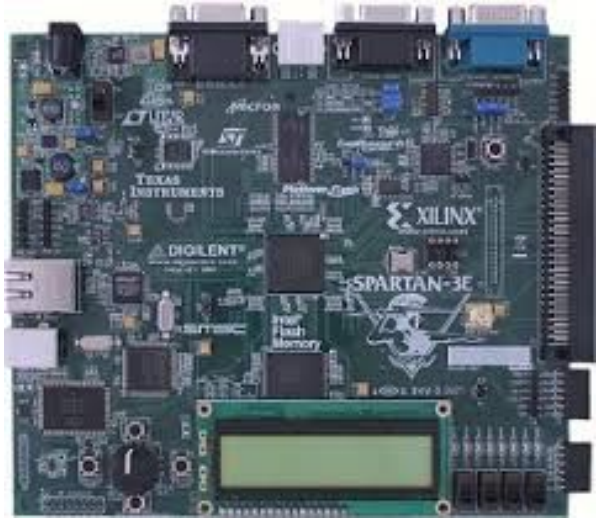
Conclusion

Challenges

- How a stack architecture will actually work
- Memory -- testing takes forever
- Input/Output
- Blocking vs. NonBlocking vs. Sequential vs. Combinational
- Timing
- Putting it all together -- the subsystems

Future Work

- FPGA board implementation
- Advanced Assembler
- Harder, Better, Faster, Stronger!
- Handle Stack Overflow and Interrupts



Questions