Design of 32 bit processor

By Barun Parua(21CS10014) and Ranjim Prabal Das(21CS10054)

Summary

Here is a brief summary of our implementation for the 32 bit processor. We have tried to keep it as modular as possible. We have used the following modules:

We have modules for memory, data and instruction separate for better utilization. They are basically arrays of registers that allow storing large amounts of information in a word addressable way.

Note that for this part we haven't used the instruction memory as we only needed to implement the ALU, Register Bank and the Control Unit. Here our processor basically takes in the instruction as an input and then decodes it to perform the required operation. We have used the testbench to provide the instructions to the processor. We have given extensive comments in the code to explain the working of the processor.

Apart from that, we have the ALU to do all the operations and a Register Bank to implement register operations. They were provided in the earlier assignment itself as well. We also have some extra operations in the alu for better implementation of the processor.

Finally, we have the control unit which is basically a collection of multiple control signals that are used to control the processor. The opcodes are used to select the muxes for the operations. The control unit also has the logic for the branching and the jump operations. The push and pop operations are also implemented in the control unit.

The testbench is also given along with the code. It can be used to get proper outputs to verify operations of our processor. Again, we haven't made the instruction memory, but supported normal instructions rather than push pop call return as they are the part of the next part.

List of Modules

1. ALU
2. Control Unit
3. PC Adder
4. Register Bank with Stack Pointer
5. Sign Extender

6. Instruction Memory
7. Data Memory
8. Condition Checker For Branching
9. Various Multiplexers

Registers

1. PC
2. IR
3. A
4. B
5. Imm
6. ALUOut/Z
7. NPC
8. LMD

Signals

1. PCin - Input to PC
2. NPCin - Input to NPC
3. ReadIM - Read Instruction Memory
4. IRin - Input to IR
5. ReadRegPort1 - Read Register Port 1
6. ReadRegPort2 - Read Register Port 2
7. Ain - Input to A
8. Bin - Input to B
9. Imm - Input to Imm
10. SelA/NPC’ - Select A or NPC’ using Mux
11. SelB/Imm’ - Select B or Imm’ using Mux
12. ALUfunc - ALU Function
13. EnbCond - Enable Condition Checker
14. SelCond - Select Condition Checker 2 bit
15. ZtoMem - Z to Memory Mux Select
16. ZtoNPC - Z to NPC Mux Select
17. NPCtoB - NPC to B Mux Select || for Data, making easy in case of Call
18. LMDto16 - LMD to Multiplexer 16 Select
19. AtoZ - A to Z Mux Select || for Address
20. LMDtoZ - LMD to Z Mux Select
21. WriteMem - Write Memory

22. ALUout - Output of ALU
23. Zin - Input to Z
24. MuxPC - Output of MuxPC
25. MuxWB - Output of MuxWB
26. WriteReg - Write Register
27. ReadMem - Read Memory
28. LMDin - Input to LMD

Basic Components

7 muxes in total
1 adder for PC
1 sign extender for Imm [basically a signed reg]
1 ALU
1 Condition Checker
1 Register Bank
1 Instruction Memory
1 Data Memory

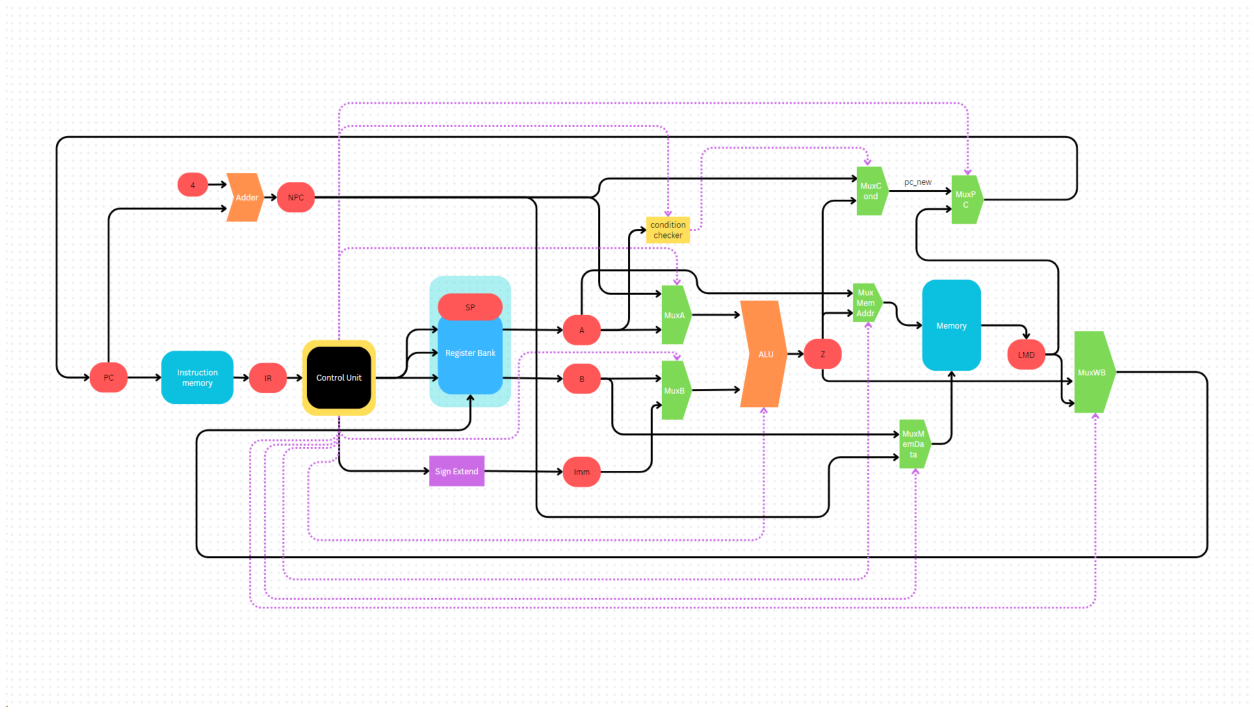
Control Signal triggers for each instruction

Since our data path is designed specifically for this instruction set, most of the instructions are executed in 1 clock cycle.

instruction	en_rp1	en_rp2	en_wp	selA	selR	we	re
ADD	1	1	1	1	1	0	0
SUB	1	1	1	1	1	0	0
AND	1	1	1	1	1	0	0
OR	1	1	1	1	1	0	0
XOR	1	1	1	1	1	0	0
SLA	1	1	1	1	1	0	0
SRA	1	1	1	1	1	0	0
SRL	1	1	1	1	1	0	0
ADDI	1	0	1	0	1	0	0

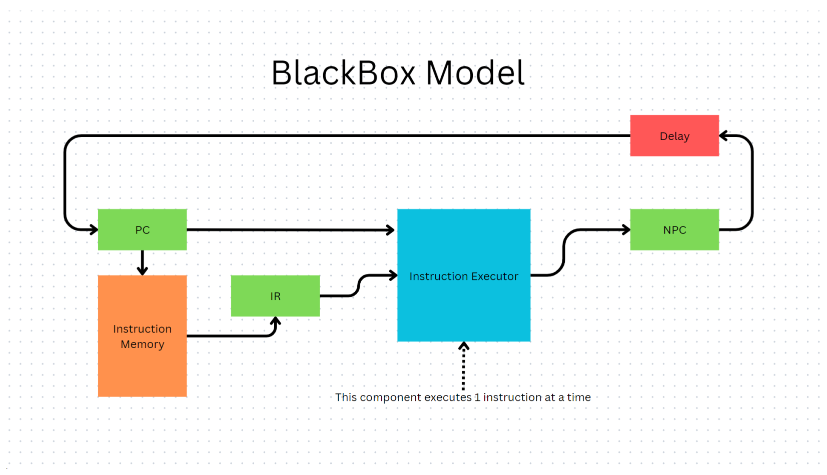
SUBI	1	0	1	0	1	0	0
ANDI	1	0	1	0	1	0	0
ORI	1	0	1	0	1	0	0
XORI	1	0	1	0	1	0	0
SLAI	1	0	1	0	1	0	0
SRAI	1	0	1	0	1	0	0
SRLI	1	0	1	0	1	0	0
NOT	1	0	1	1	1	0	0
NOTI	1	0	1	0	1	0	0
LD	1	0	1	0	1	0	1
ST	1	1	0	0	0	1	0
LDSP	1	0	1	0	1	0	1
STSP	1	1	0	0	0	1	0
BR	0	1	0	0	0	0	0
BMI	0	1	0	0	0	0	0
BPL	0	1	0	0	0	0	0
BZ	0	1	0	0	0	0	0
PUSH	1	0	1	1	1	1	0
POP	1	0	1	1	1	0	1
CALL	1	0	1	1	1	1	0
RET	1	0	1	1	1	0	1
MOVE	1	0	1	1	1	0	1
HALT	0	0	0	0	0	0	0
NOP	0	0	0	0	0	0	0

Complete Data Path Diagram



Black Box Model for Better Implementation of Next Part

Here is a small modular representation of the part we have done this time, so that we can better integrate next time. It can be seen that a merged implementation with Instruction Memory and sync with Program Counter is needed for us to be able to run the final codes.


BlackBox Model
Delay
PC
NPC
Instruction Executor
IR
Instruction Memory
This component executes 1 instruction at a time