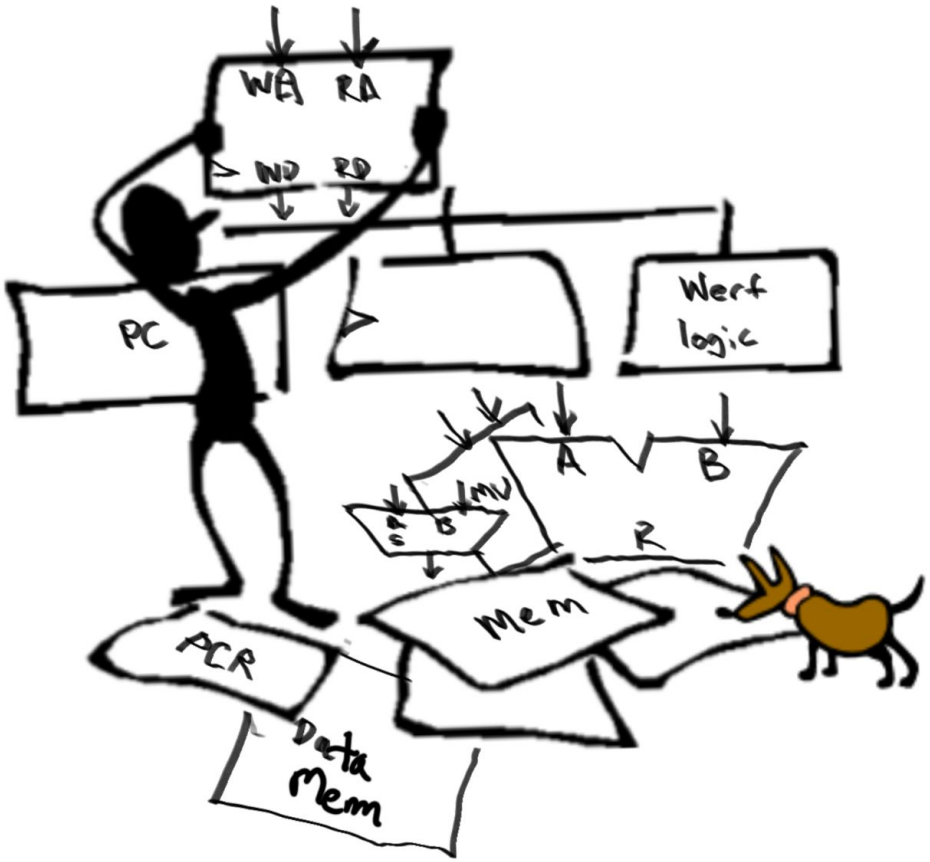
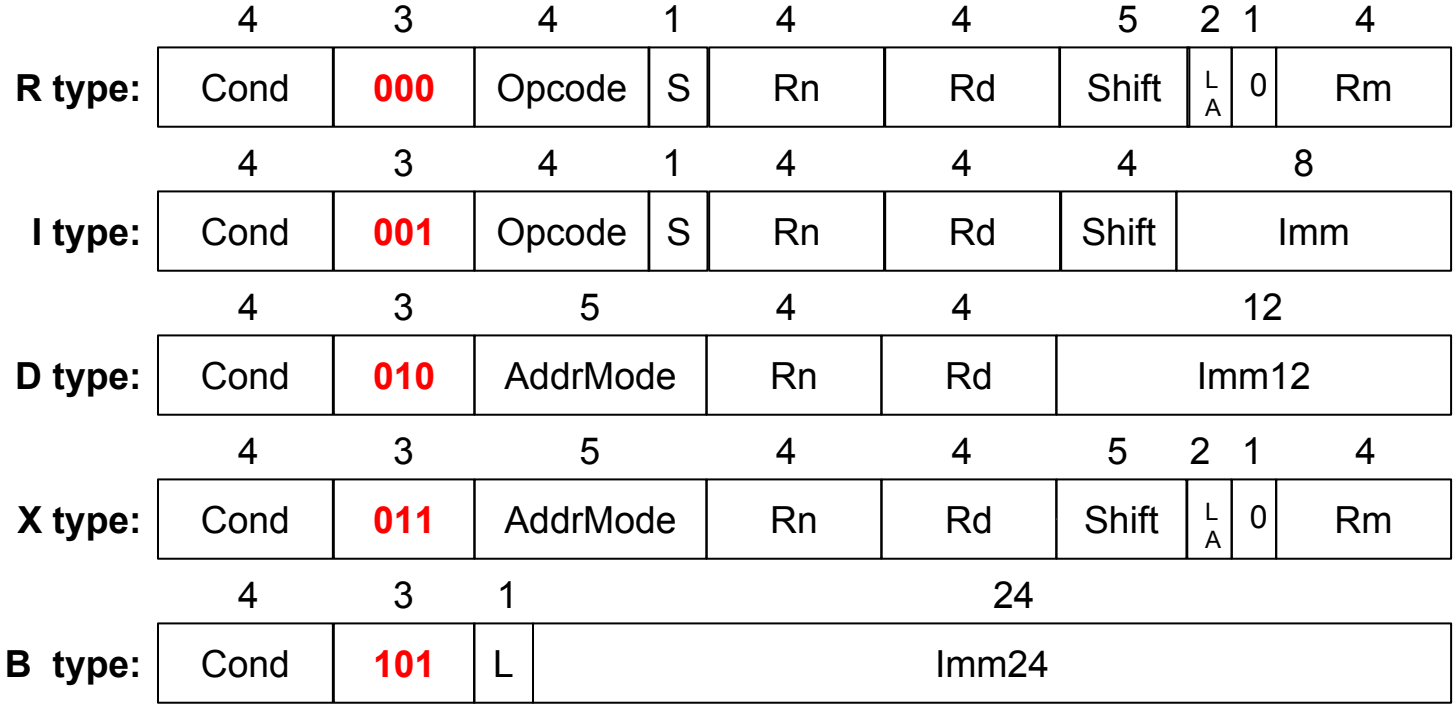
Building a Computer



The ARM7 ISA



Five key instruction formats:

0) ALU with two register operands
1) ALU with a register and an immediate operand
2) Load/Store with an immediate offset
3) Load/Store with a register offset
5) Branch

R-type Data Processing

ALU instructions with register operands

Rd - register file write address

Rn, Rm - register source operands

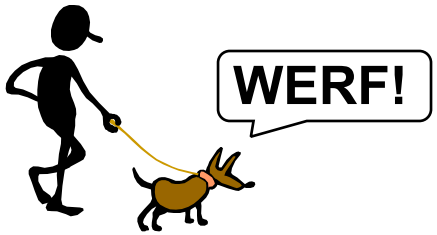
Shift or Rs - Optional shift of Rm

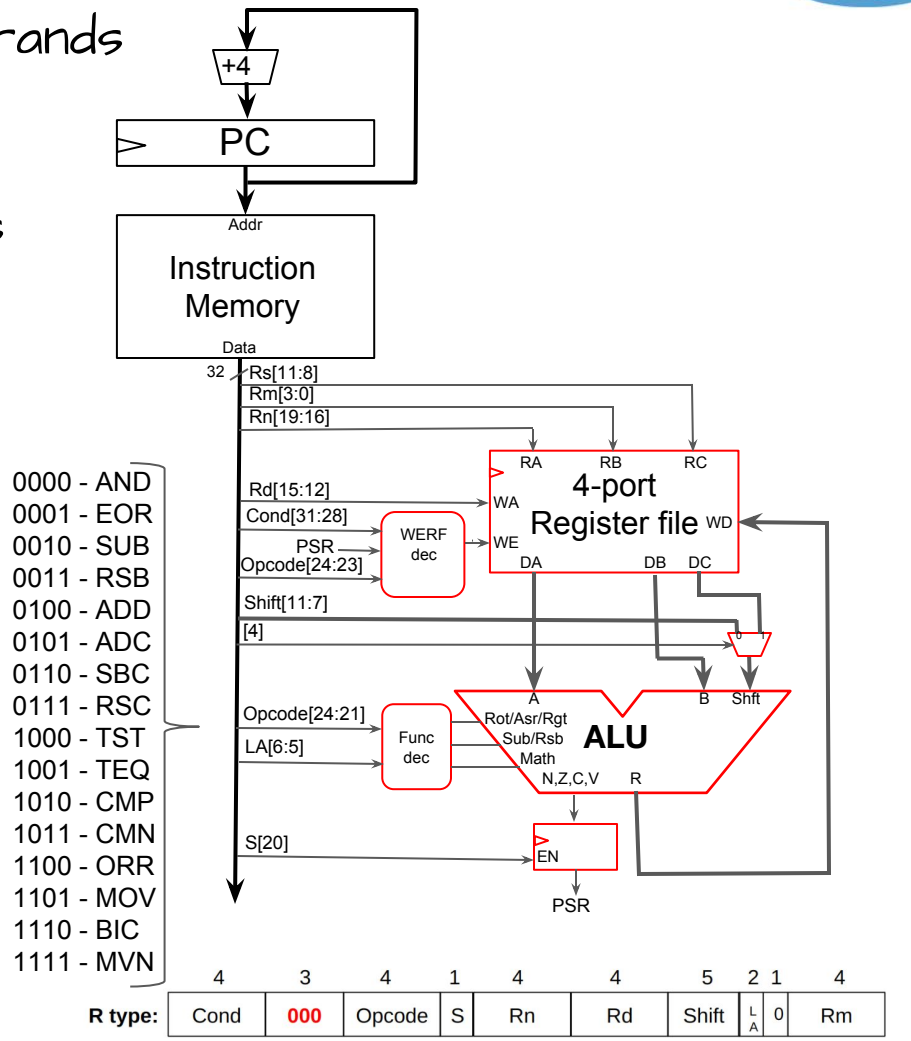
LA - direction and type of shift

S-bit - controls update of PSR

Func decoding from ALU lecture

Register write back controlled by WERF logic



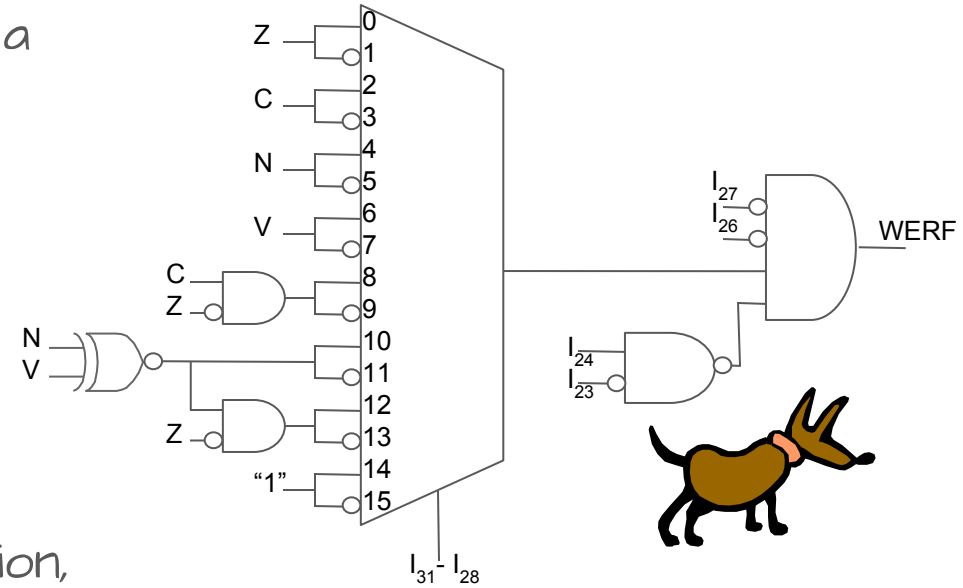


WERF LOGIC

Not every instruction updates a destination register

CMP, CMN, TST, TEQ don't update any register

Conditional execution is controlled by the WERF logic. WERF is set only if the condition, as determined by the PSR flags, is met. Otherwise it is zero and the register is not updated.



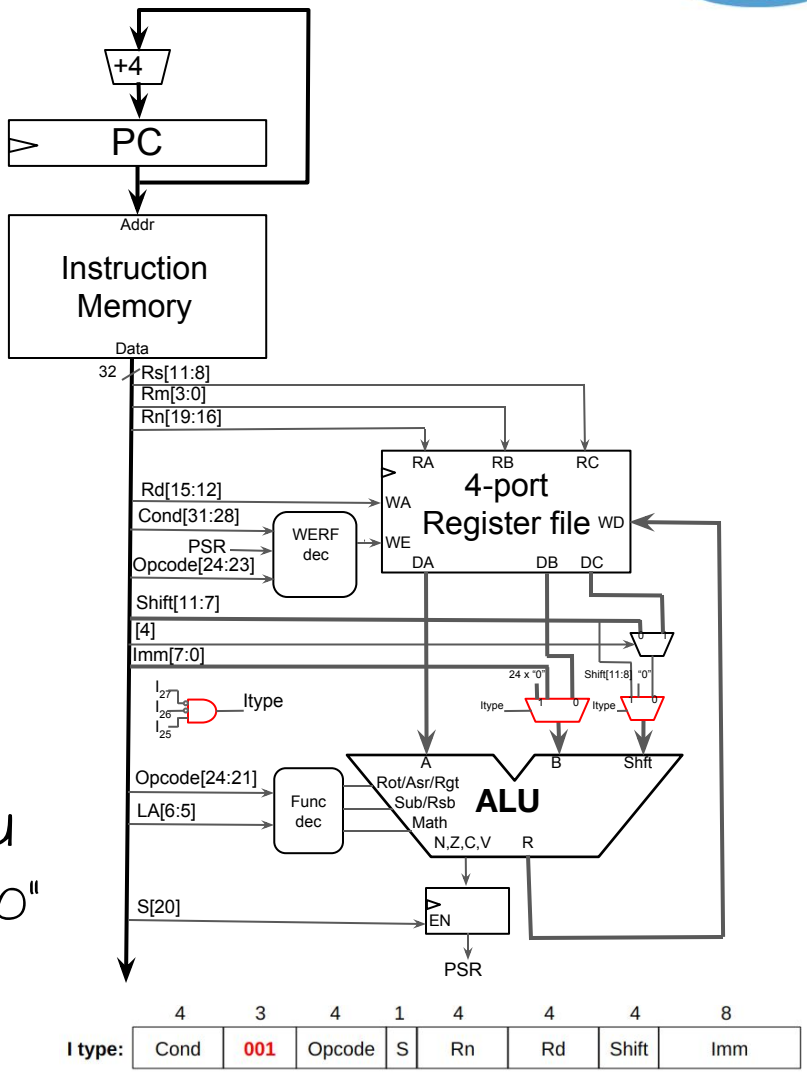
I-type Data Processing

ALU instructions with a register and an immediate operand

Rd - register file write address
Rn - register source operand
Imm8 - 8-bit immediate operand
Shift - Optional rotate

Adds a mux to the B input of the ALU
- 8-bit immediate value is zero-extended 24-bits

And a mux to the shift input of the ALU
- LSB of shift is set to always be "0"



Data Transfer

Load/Store instructions using a base register and an immediate offset

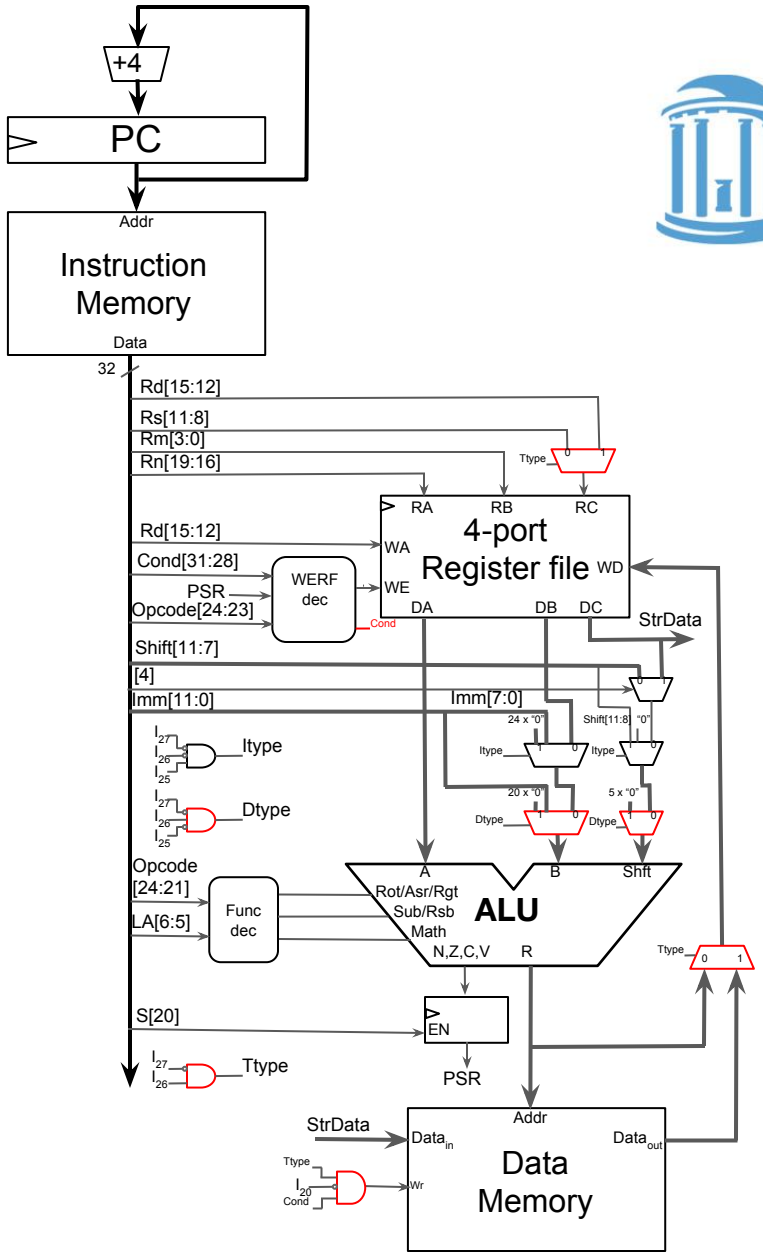
Rd - register file write address
Rn - base register
Imm12 - 8-bit immediate operand

Adds another mux to the ALU's B input
- 12-bit immediate value is zero-extended 20-bits

All bits of shift are set to "0"

	4	3	5	4	4	12
D type:	Cond	010	AddrMode	Rn	Rd	Imm12

	4	3	5	4	4	5	2	1	4
X type:	Cond	011	AddrMode	Rn	Rd	Shift	LA	0	Rm

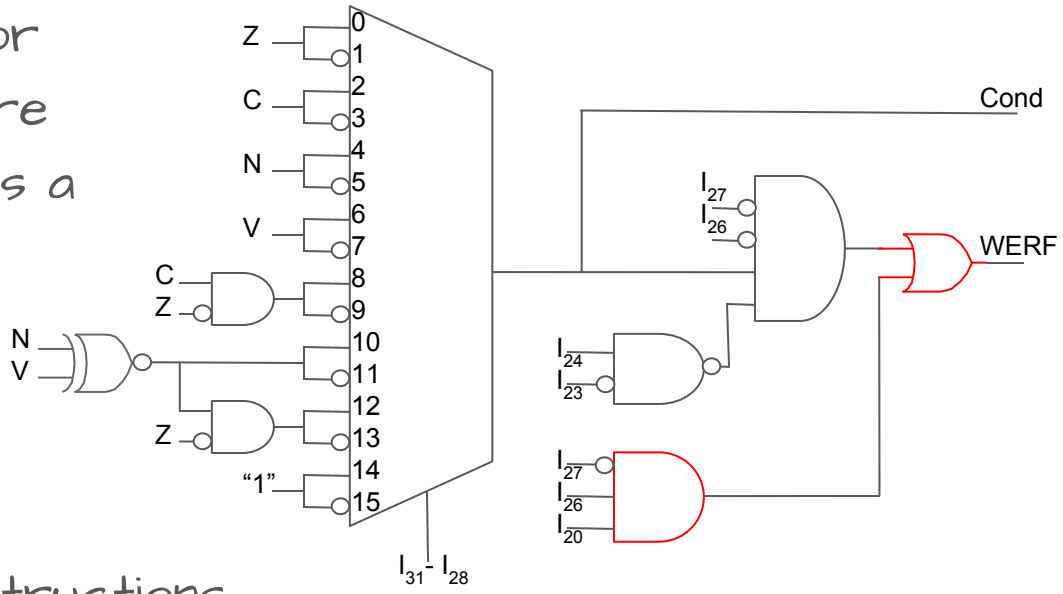


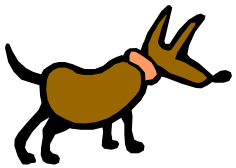
WERF Modifications

We need to also allow for conditional Load and Store instructions. This requires a few small mods to the WERF logic.

It includes bringing out a "Cond" signal that can be used to qualify Store instructions.

We also generate WERF if the instruction is a LOAD and the condition is valid.

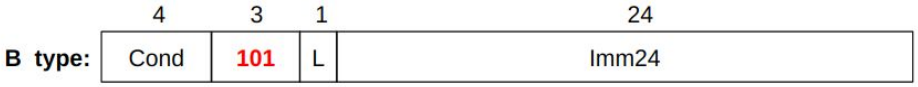


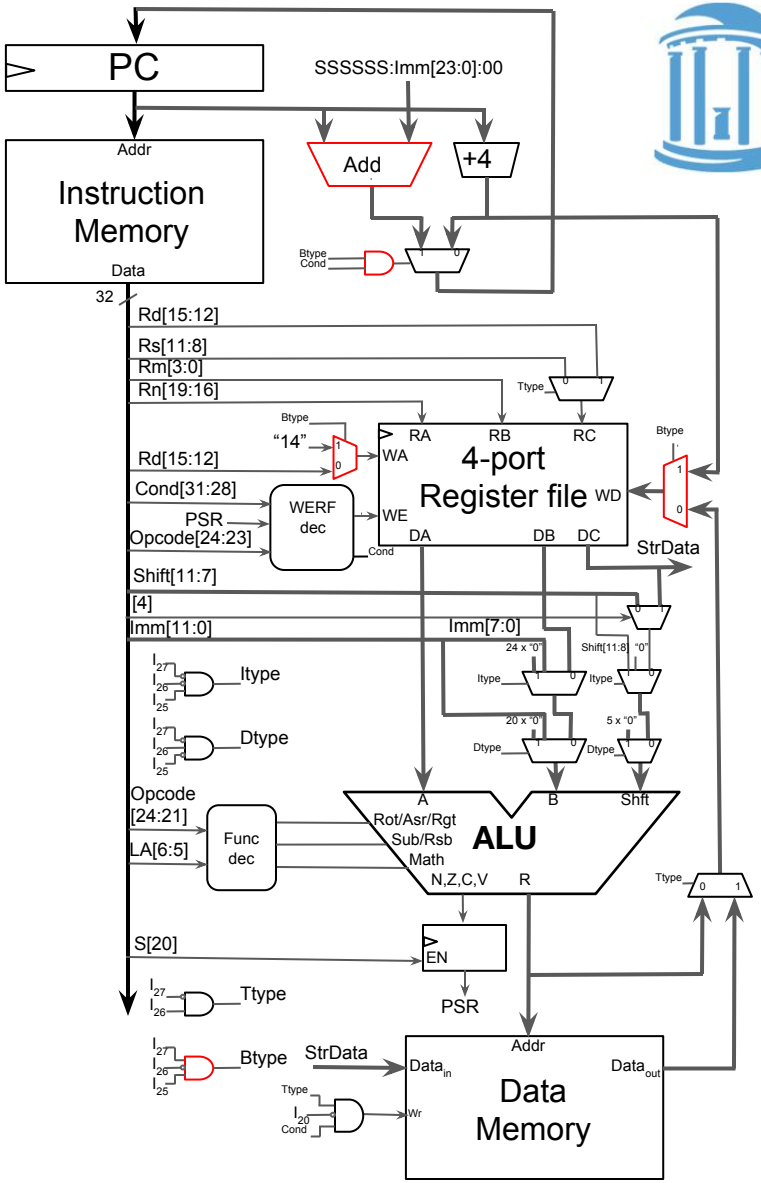


Branch Instructions

If condition is true then add 24-bit "sign-extended" immediate value to the current PC.

If the "link" bit is set, then we need to write PC+4 into R14. This is accomplished with muxes on the register file's WA and WD inputs.



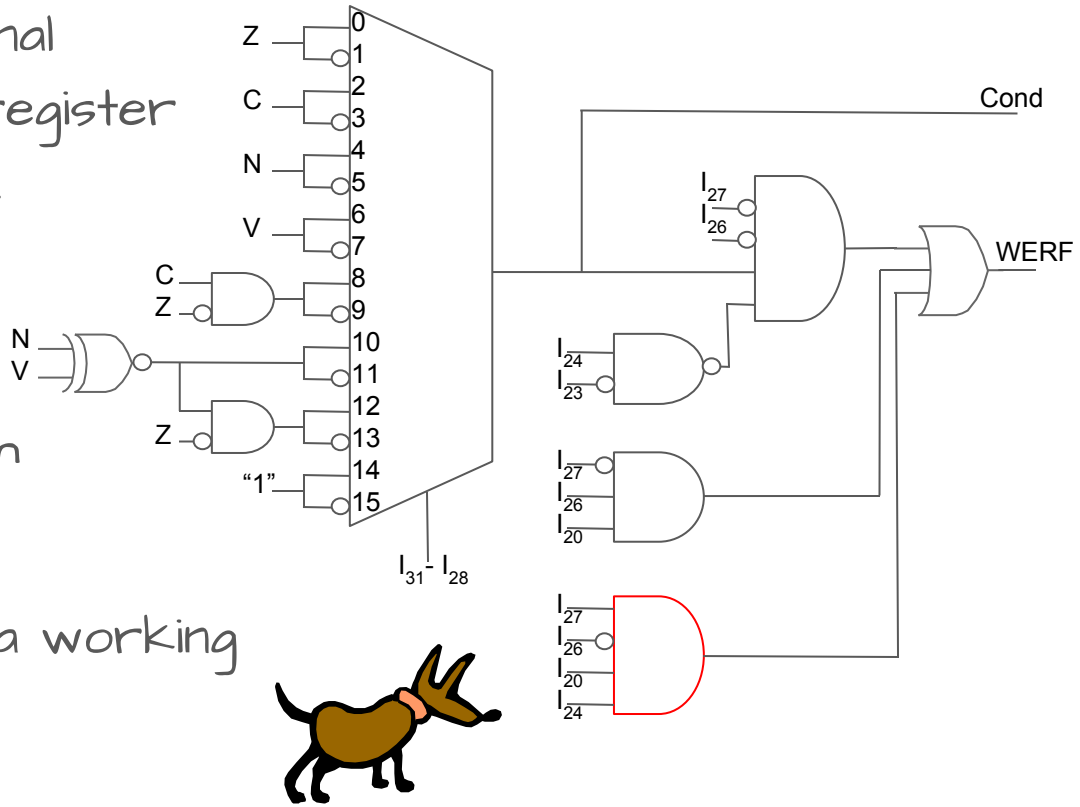


Final WERF Modification

Finally allow for conditional updates of the the link register (R14) via the WERF logic.

This third type of WERF merely decodes A branch instruction with it's "L" bit set.

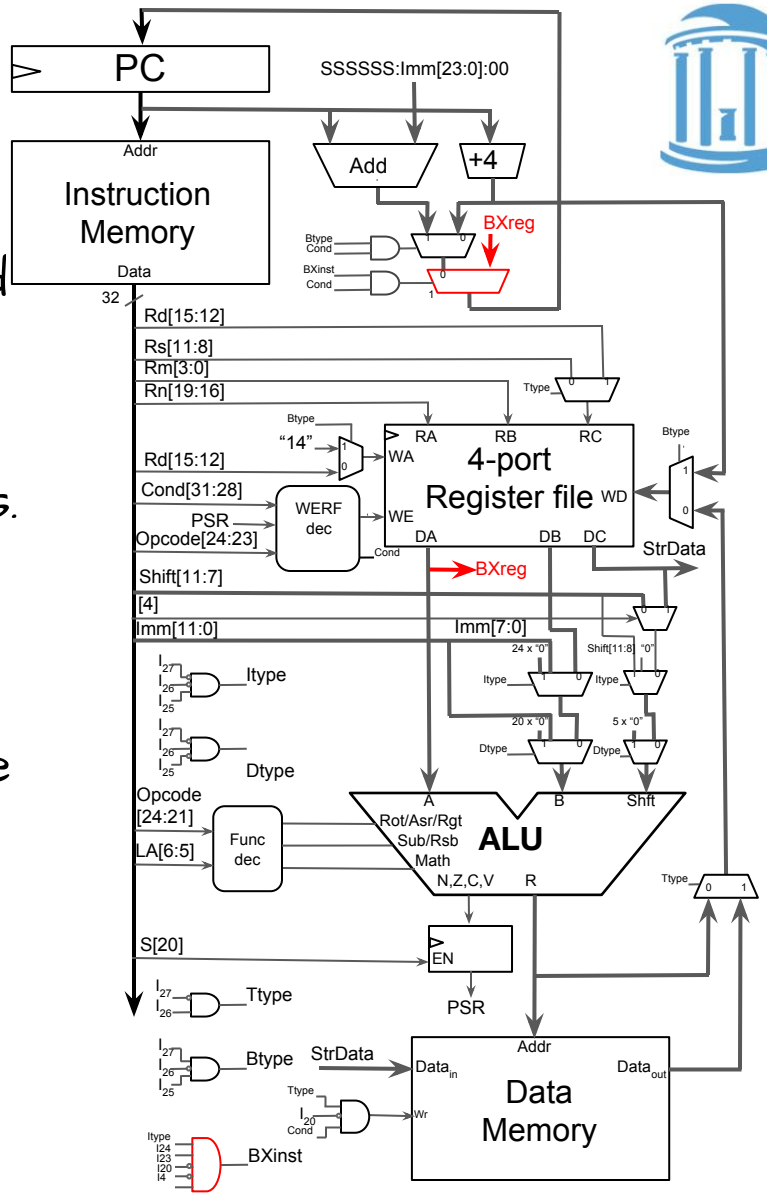
We're getting closer to a working ARM7 processor.



Leftovers

Lastly, we need to implement the "BX" instruction, which, if you recall, was used to branch using a register's contents as a target address. It was also our goto instruction for returning from functions.

The Rn field specifies that target register, and it needs to be routed to a mux so that it is an option for loading the PC.



	4	3	21	4
R type:	Cond	000	1 0010 1111 1111 1111 0001	Rn

What' missing?

Instruction Type	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Notes
Data Processing (reg3)	Cond				0	0	0	OpCode				S	Rn				Rd				Shift					LA		0	Rm				
Data Processing (reg4)	Cond				0	0	0	OpCode				S	Rn				Rd				Rs				0	LA		1	Rm				
The following use the reg4 format with b7 = 1 and b4 = 1																																	
~~Multiply~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~0~~	~~L~~	~~U~~	~~A~~	~~S~~	~~Rd~~				~~Rn~~				~~Rs~~				~~1~~	~~0~~	~~0~~	~~1~~	~~Rm~~				~~Rd and Rn are swapped~~
~~SWP instruction~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~B~~	~~0~~	~~0~~	~~Rn~~				~~Rd~~				~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~0~~	~~1~~	~~Rm~~				
~~LDRH/STRH~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~P~~	~~U~~	~~0~~	~~W~~	~~L~~	~~Rn~~				~~Rd~~				~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~	~~Rm~~				
~~LDRSB~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~P~~	~~U~~	~~0~~	~~W~~	~~L~~	~~Rn~~				~~Rd~~				~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~1~~	~~0~~	~~1~~	~~Rm~~				
~~LDRSH~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~P~~	~~U~~	~~0~~	~~W~~	~~L~~	~~Rn~~				~~Rd~~				~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~1~~	~~1~~	~~1~~	~~Rm~~				
~~LDRH/STRH (imm)~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~P~~	~~U~~	~~1~~	~~W~~	~~L~~	~~Rn~~				~~Rd~~				~~immHi~~				~~1~~	~~0~~	~~1~~	~~1~~	~~immLo~~				
~~LDRSB (imm)~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~P~~	~~U~~	~~1~~	~~W~~	~~L~~	~~Rn~~				~~Rd~~				~~immHi~~				~~1~~	~~1~~	~~0~~	~~1~~	~~immLo~~				
~~LDRSH (imm)~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~P~~	~~U~~	~~1~~	~~W~~	~~L~~	~~Rn~~				~~Rd~~				~~immHi~~				~~1~~	~~1~~	~~1~~	~~1~~	~~immLo~~				
The following reinterpet the TST, TEQ, CMP, and CMN instructions when S = 0																																	
Branch and Exchange (BX)	Cond				0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn				reg4, Rn instead of Rm
~~MRS~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~P_s~~	~~0~~	~~0~~	~~1~~	~~1~~	~~1~~	~~1~~	~~Rd~~				~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~reg3~~
~~MSR~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~P_d~~	~~1~~	~~0~~	~~1~~	~~0~~	~~0~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~Rm~~				
~~MSR (reg flags)~~	~~Cond~~				~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~P_d~~	~~1~~	~~0~~	~~1~~	~~0~~	~~0~~	~~0~~	~~1~~	~~1~~	~~1~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~Rm~~				
Data Processing (imm)	Cond				0	0	1	OpCode				S	Rn				Rd				Rotate				Imm								
~~MSR (imm flags)~~	~~Cond~~				~~0~~	~~0~~	~~1~~	~~1~~	~~0~~	~~P_d~~	~~1~~	~~0~~	~~1~~	~~0~~	~~0~~	~~0~~	~~1~~	~~1~~	~~1~~	~~1~~	~~Rotate~~				~~Imm~~								
Data Transfer (imm offset)	Cond				0	1	0	P	U	B	W	L	Rn				Rd				Immediate offset												
Data Transfer (reg offset)	Cond				0	1	1	P	U	B	W	L	Rn				Rd				Shift					LA		0	Rm				
~~Block Transfer~~	~~Cond~~				~~1~~	~~0~~	~~0~~	~~P~~	~~U~~	~~0~~	~~W~~	~~L~~	~~Rn~~				~~Register Vector~~																
Branch	Cond				1	0	1	L	offset																								

Next time

- Getting kick-started
- External changes in the flow of execution
- Performance measures