Computer Organization and Design

Putting it All Together

Henry Fuchs

Slides adapted from Aselmo Lastra and from Montek Singh, who adapted them
from Leonard McMillan and from Gary Bishop
Back to McMillan & Chris Terman, MIT 6.004 1999

Thursday April 9, 2015

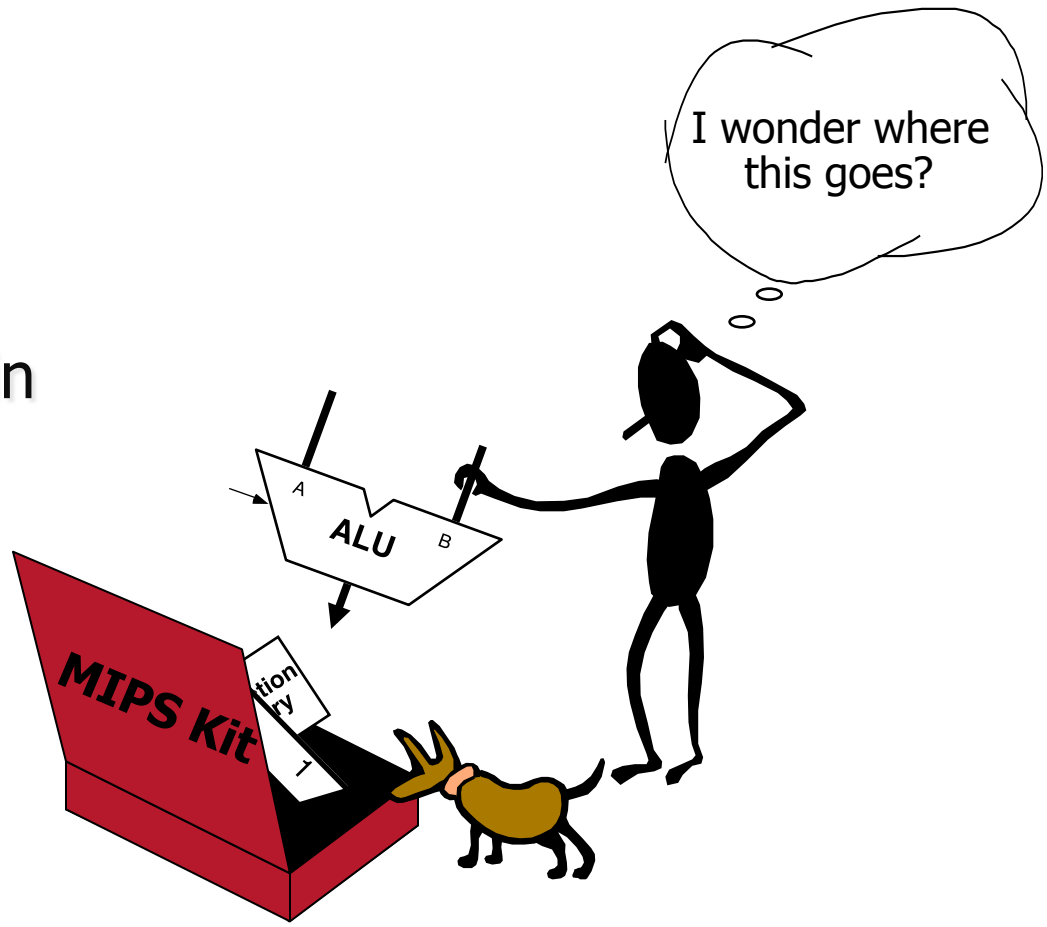
Lecture 15

Read 4.1 – 4.4

Putting it All Together

THIS IS IT!

- We are now ready to assemble a computer.
- The ingredients are all in place, so let's put them together…

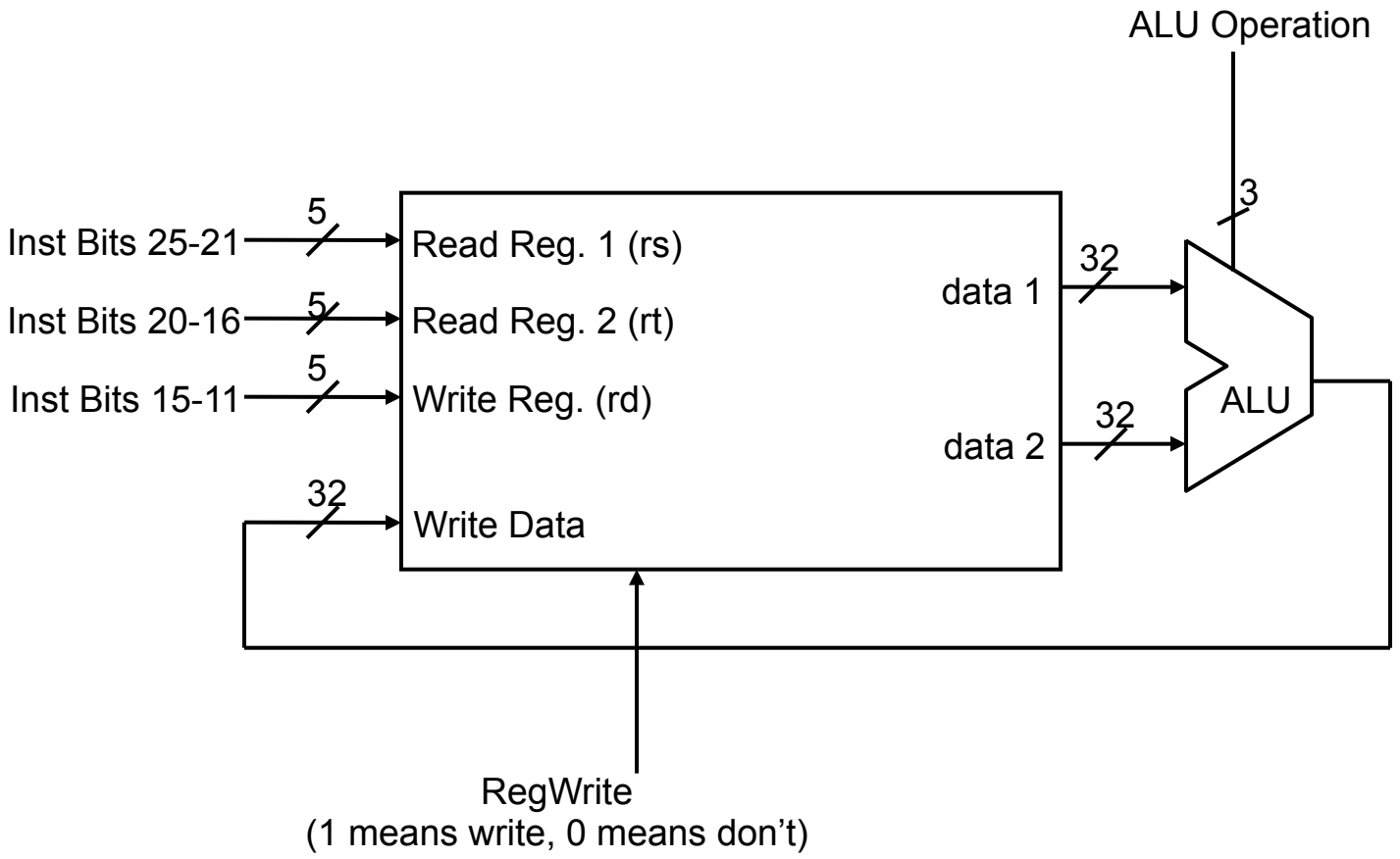


Datapath and Control

- Datapath
  - Consists of all of those components that store or process data
  - Registers, ALU, memories
- Control
  - Consists of those components that tell datapath components what to do and when
  - Clock, control logic (finite state machines or combinational look-up tables)

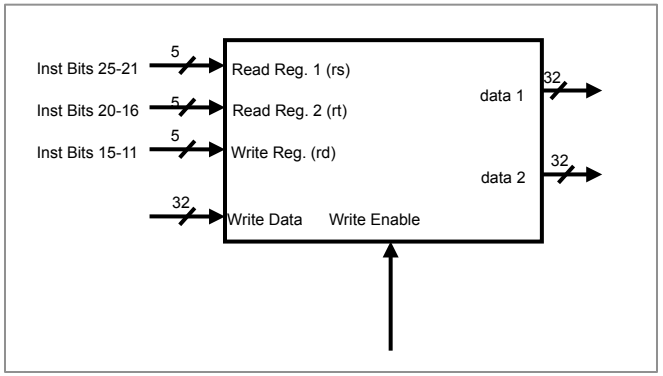
Datapath for R-type Instructions

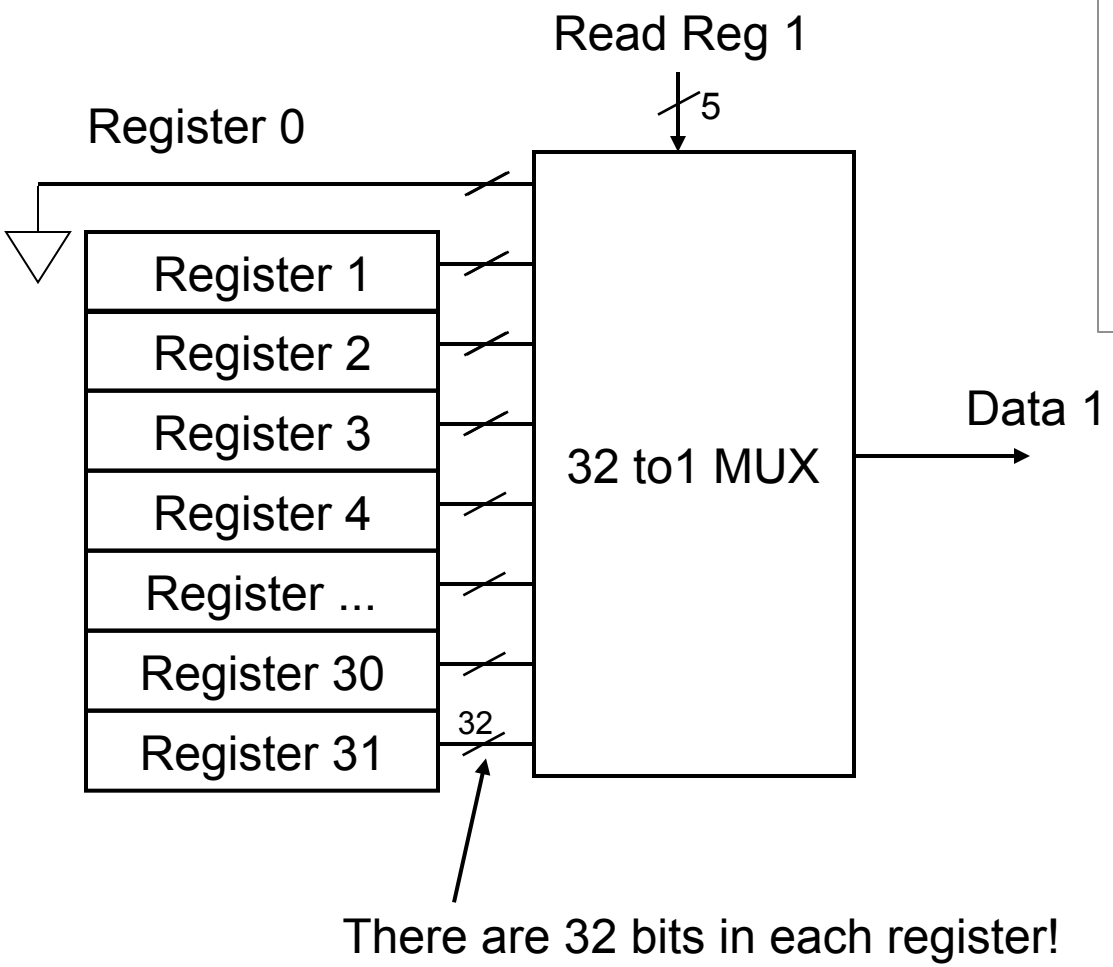
- Registers and ALU
  - All of the registers together are called register bank, or "register file"



Register File

- 32 registers ($0-$31), each 32 bits wide
- 2 ports for reading, 1 port for writing



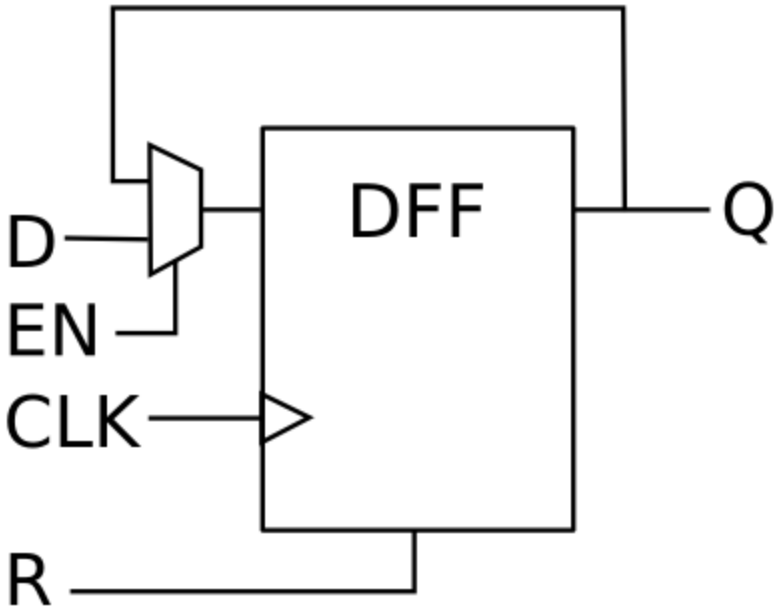


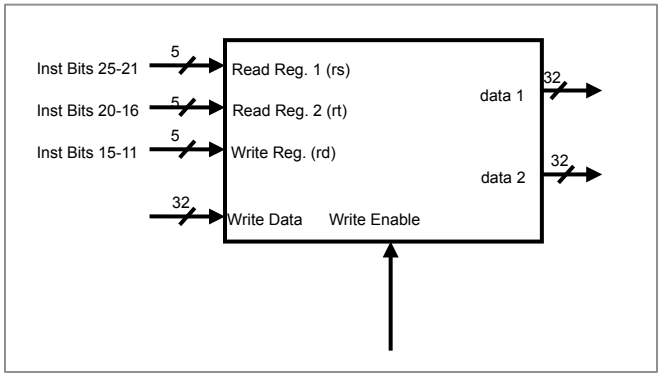
LOT'S OF CONNECTIONS!

And this is just one port! Remember, there's data1 and data2 coming out of the register file!

How Is Register Constructed

- Out of Flip Flops, 32 of them
  - But with an enable, tied to Write Enable

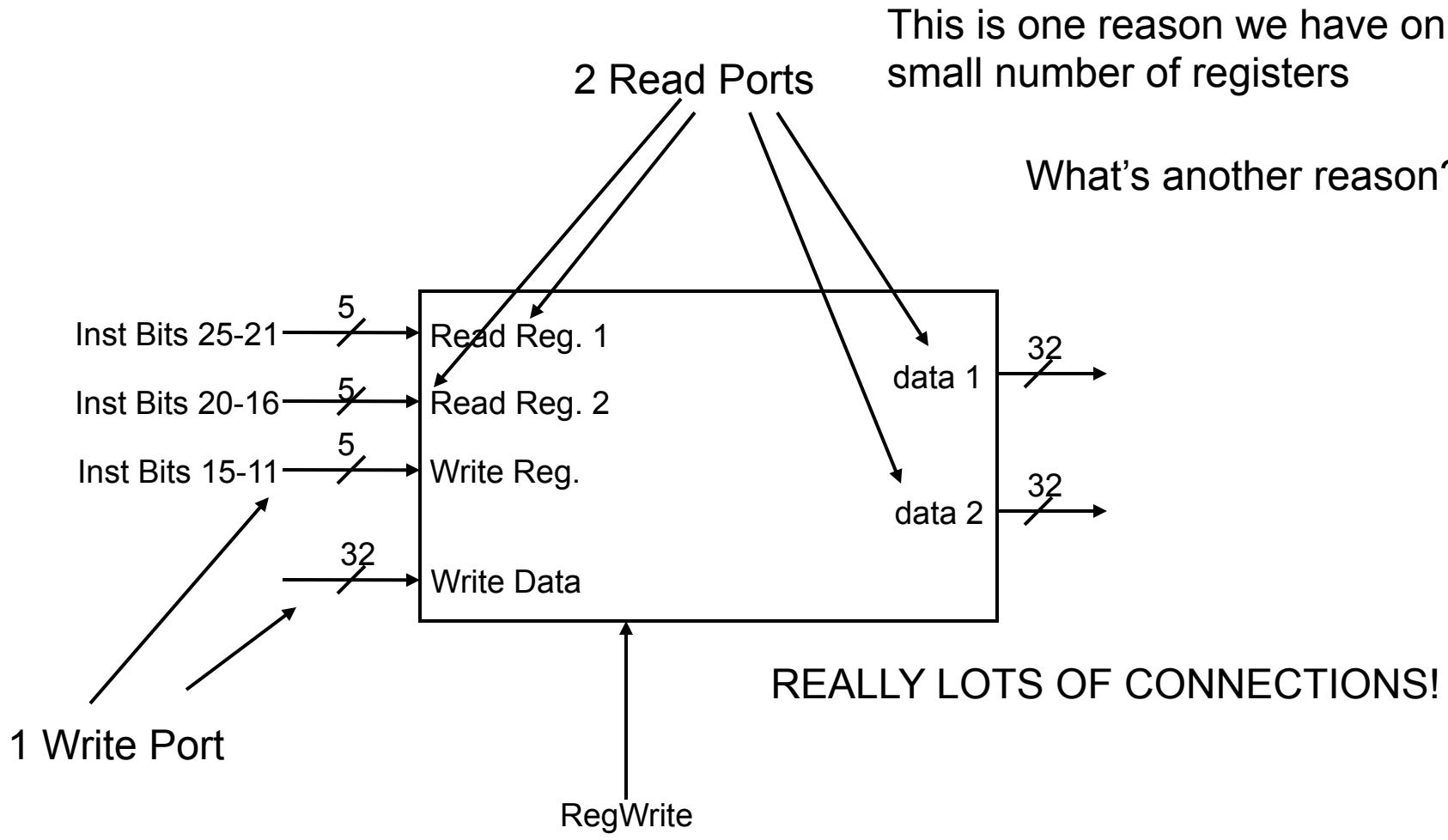




Register File has 3 ports

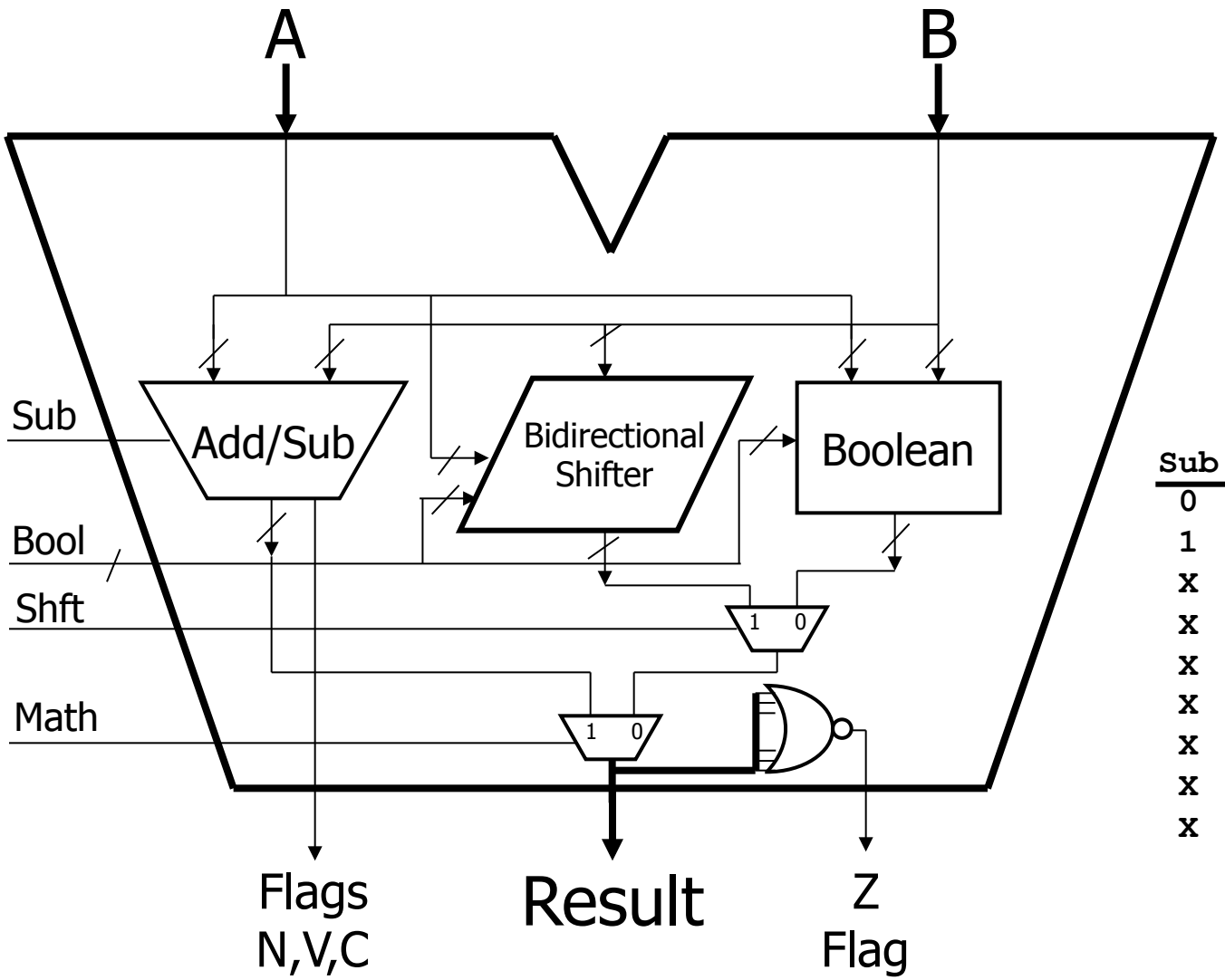
This is one reason we have only a small number of registers

What's another reason?



Let's review our ALU

- Let's review the ALU that we built a few lectures ago.

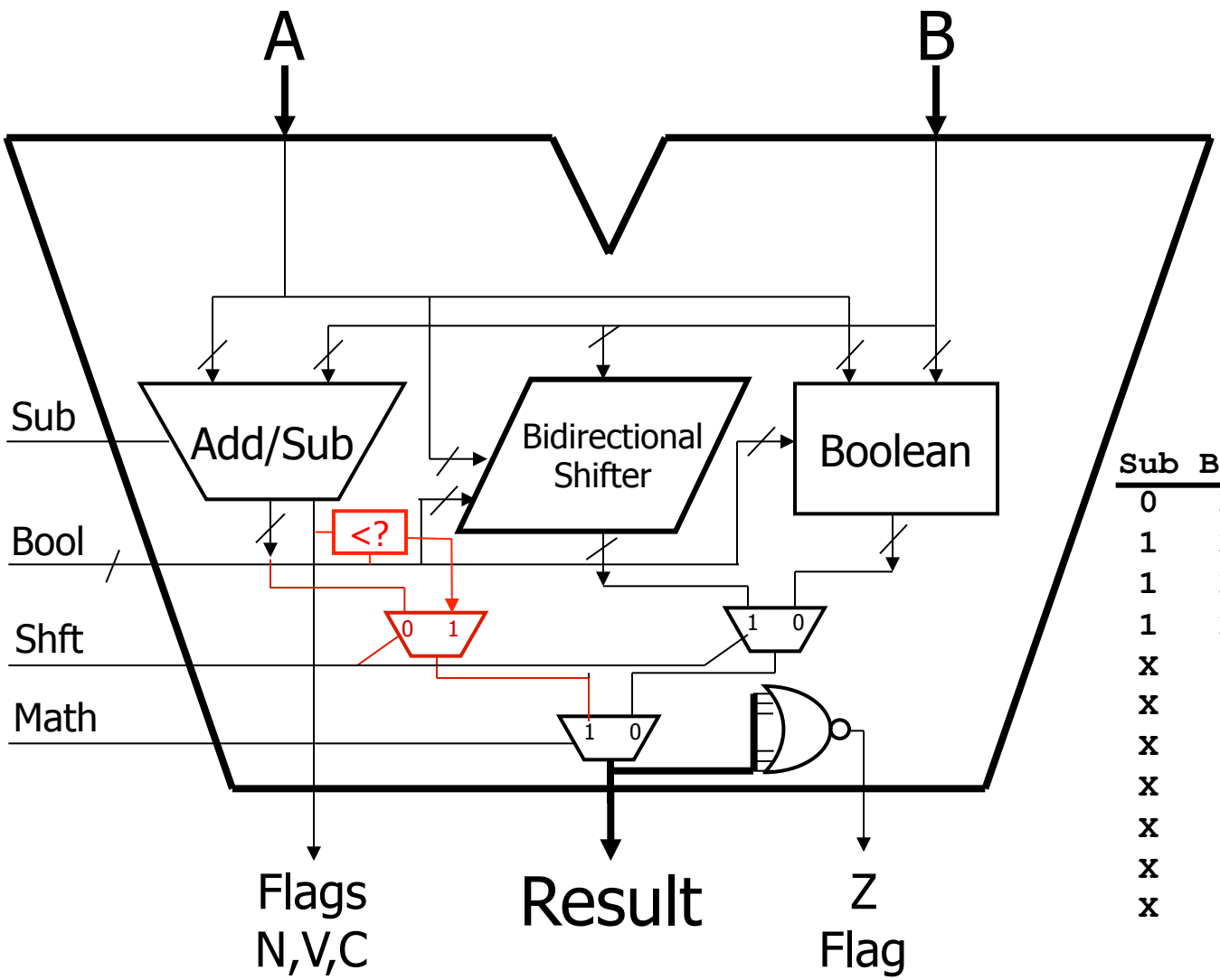


5-bit ALUFN

Sub	Bool	Shft	Math	OP
0	XX	0	1	A+B
1	XX	0	1	A-B
X	00	1	0	B<<A
X	10	1	0	B>>A
X	11	1	0	B>>>A
X	00	0	0	A & B
X	01	0	0	A | B
X	10	0	0	A ^ B
X	11	0	0	A | B

A minor modification to our ALU

- Here's that ALU with a minor modification to support comparisons



5-bit ALUFN

Sub	Bool	Shft	Math	OP
0	XX	0	1	A+B
1	XX	0	1	A-B
1	X0	1	1	A LT B
1	X1	1	1	A LTU B
X	00	1	0	B<<A
X	10	1	0	B>>A
X	11	1	0	B>>>A
X	00	0	0	A & B
X	01	0	0	A | B
X	10	0	0	A ^ B
X	11	0	0	$\overline{A \mid B}$

Design Approach

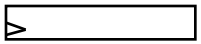
"Incremental Featurism"

- We will implement circuits for each type of instruction individually, and merge them (using MUXes, etc).

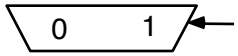
Steps:

1. 3-Operand ALU instrs
2. ALU w/immediate instrs
2. Loads & Stores
3. Jumps & Branches
4. Exceptions (briefly)

Our Bag of Components:

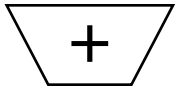


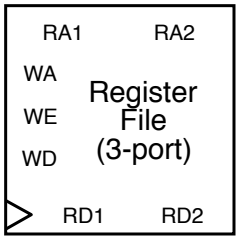
Registers

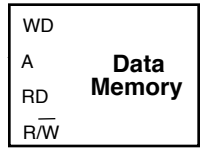


Muxes

ALU & adders

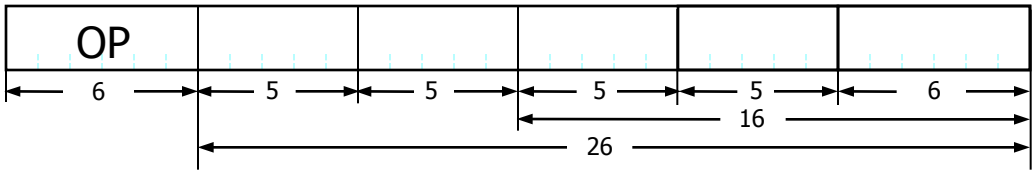






Memories

Review: The MIPS ISA



000000	r_s	r_t	r_d	shamt	func

R-type: ALU with Register operands
Reg[rd] ← Reg[rs] op Reg[rt]

001XXX	r_s	r_t	immediate

I-type: ALU with constant operand
Reg[rt] ← Reg[rs] op SEXT(immediate)

10X011	r_s	r_t	immediate

I-type: Load and Store
Reg[rt] ← Mem[Reg[rs] + SEXT(immediate)]
Mem[Reg[rs] + SEXT(immediate)] ← Reg[rt]

10X011	r_s	r_t	immediate

I-type: Branch Instructions
if (Reg[rs] == Reg[rt]) PC ← PC + 4 + 4*SEXT(immediate)
if (Reg[rs] != Reg[rt]) PC ← PC + 4 + 4*SEXT(immediate)

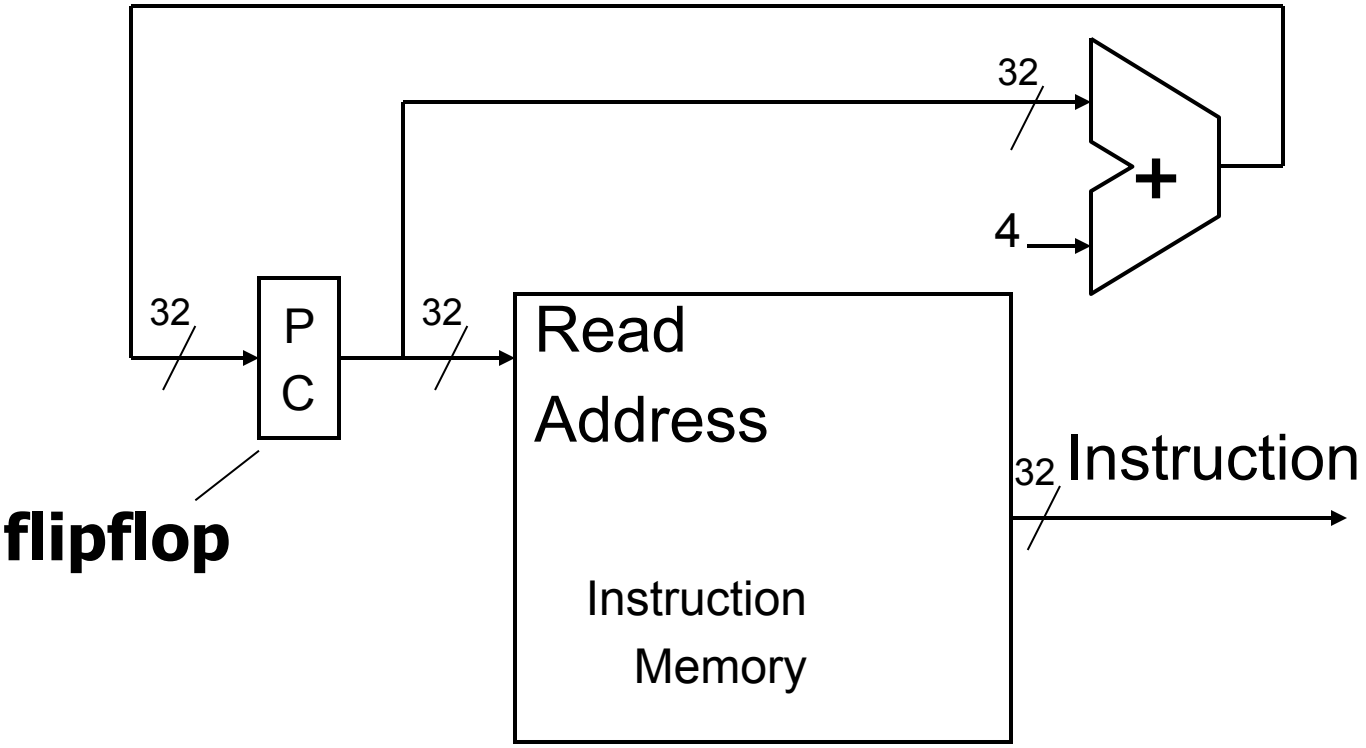
00001X	26-bit constant

J-type: jump
PC ← (PC & 0xf0000000) | 4*(immediate)

The MIPS instruction set as seen from a Hardware Perspective

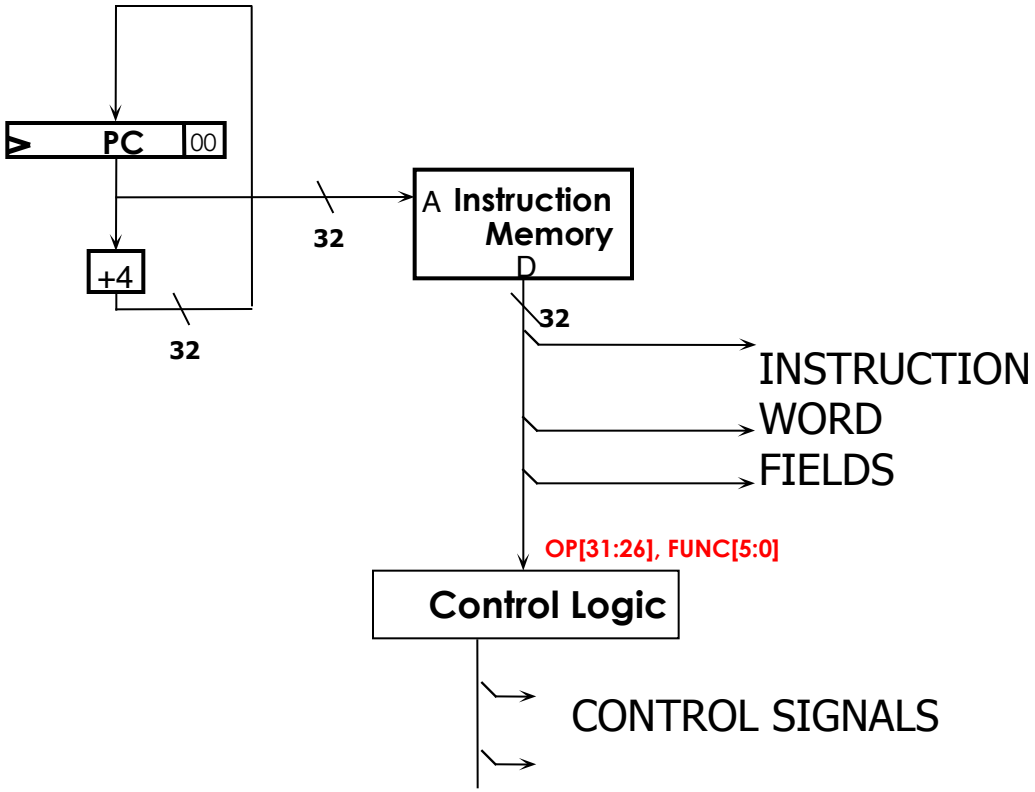
Instruction classes distinguished by types:
1) 3-operand ALU
2) ALU w/immediate
3) Loads/Stores
4) Branches
5) Jumps

Fetching Sequential Instructions



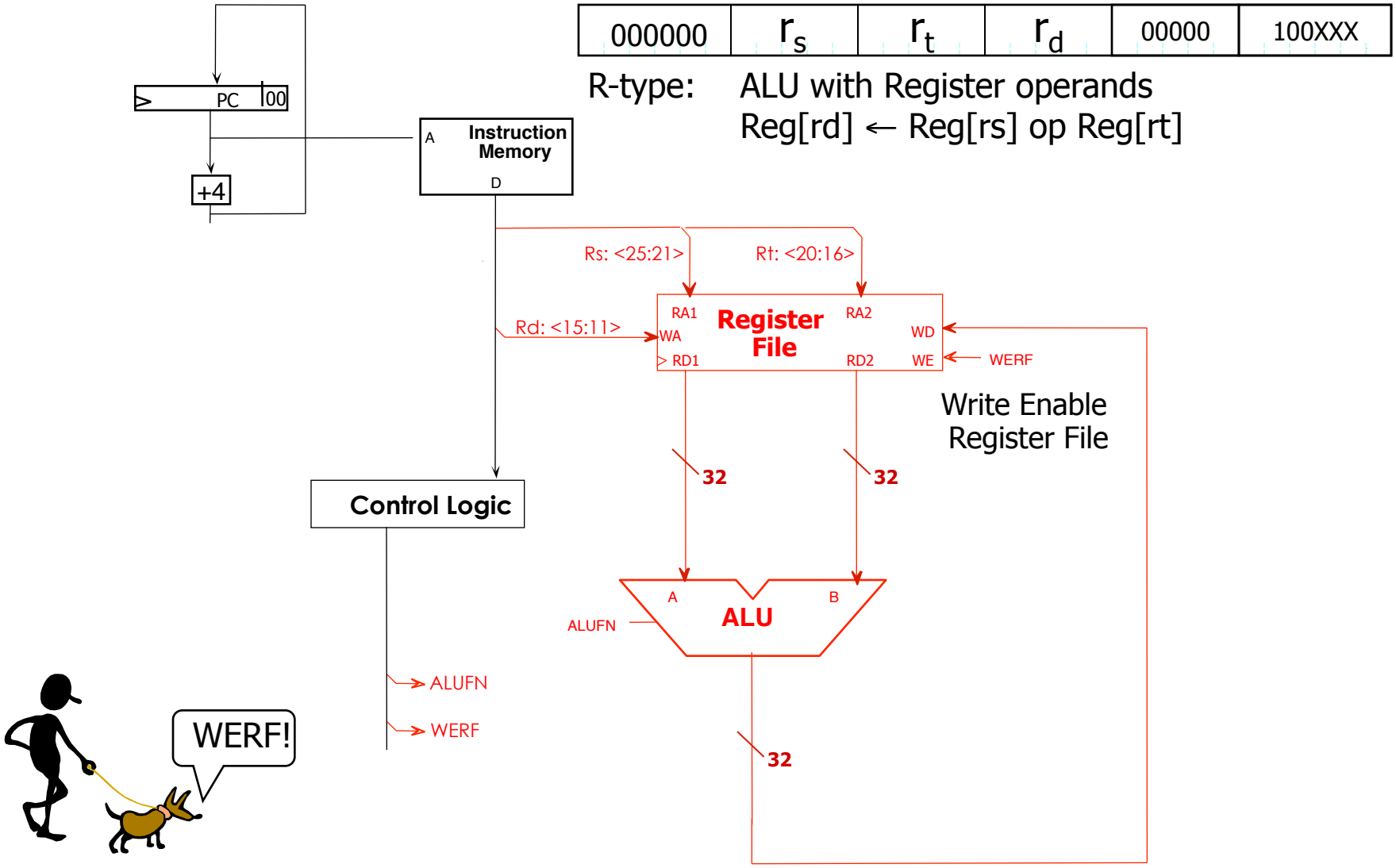
We will talk about branches and jumps later.

Instruction Fetch/Decode

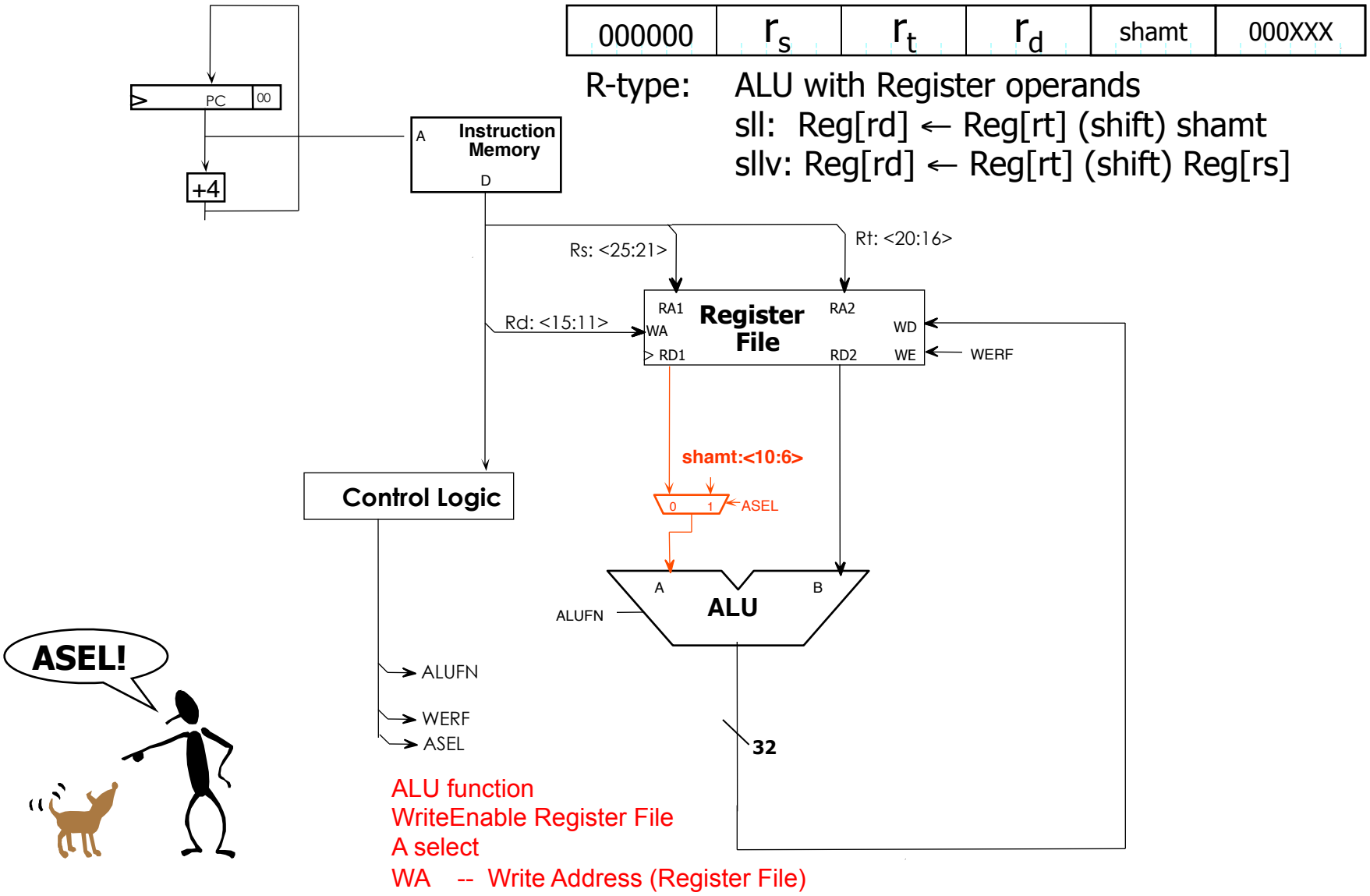


- Use a counter to FETCH the next instruction:
  - PROGRAM COUNTER (PC)
    - use PC as memory address
    - add 4 to PC, load new value at end of cycle
  - fetch instruction from memory
    - use some instruction fields directly (register numbers, 16-bit constant)
  - decode rest of the instruction
    - use bits <31:26> and <5:0> to generate controls

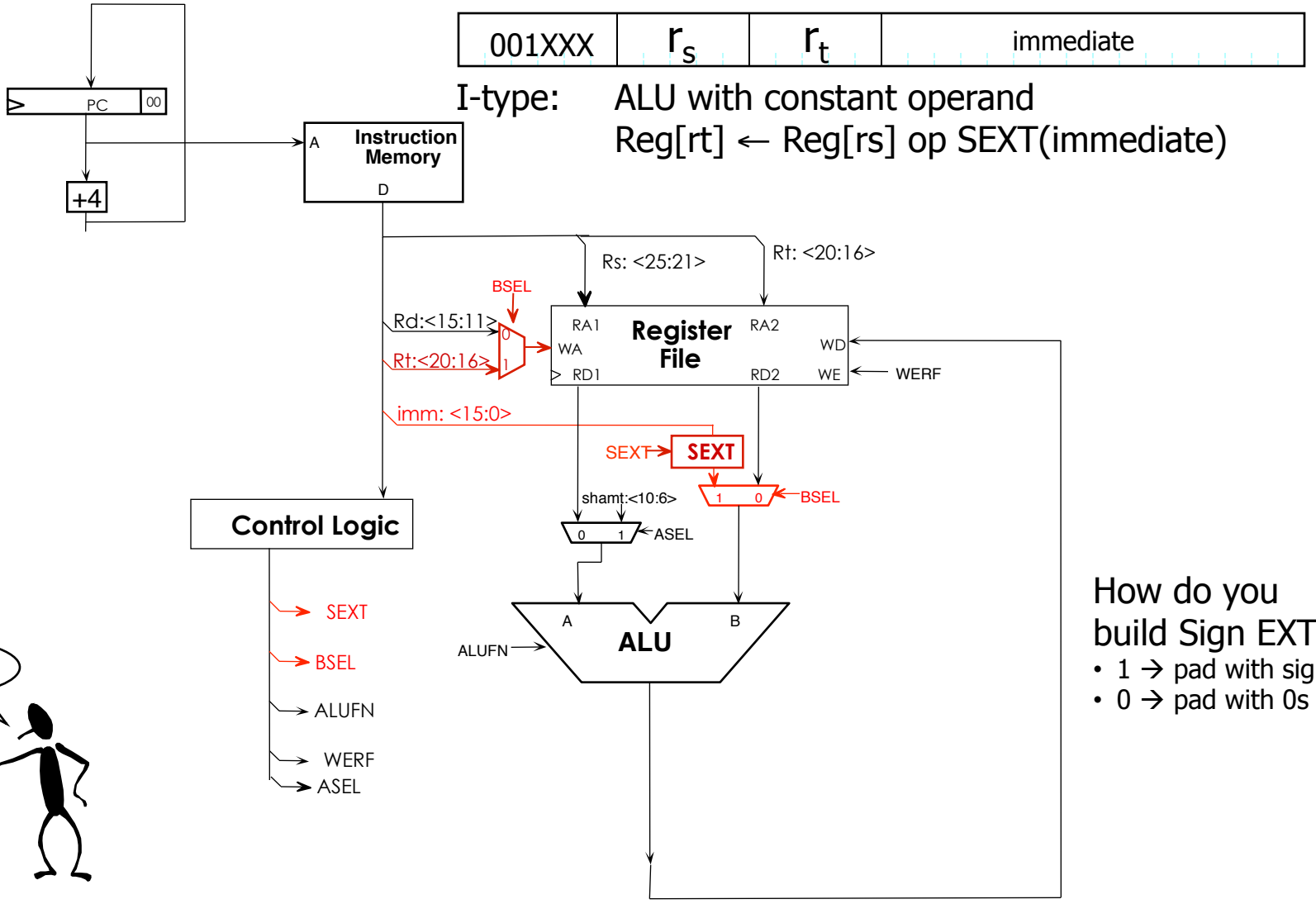
3-Operand ALU Data Path



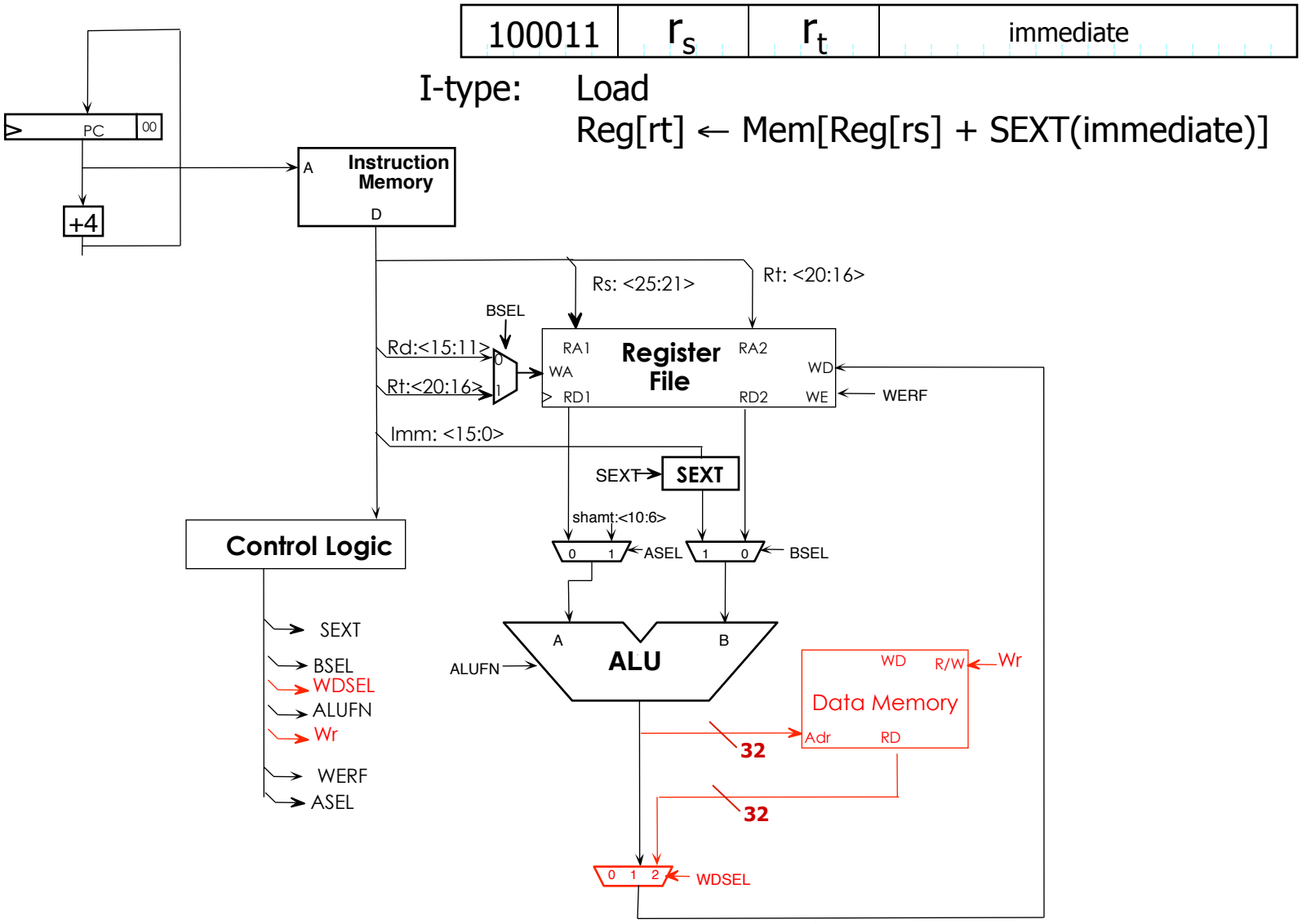
Shift Instructions



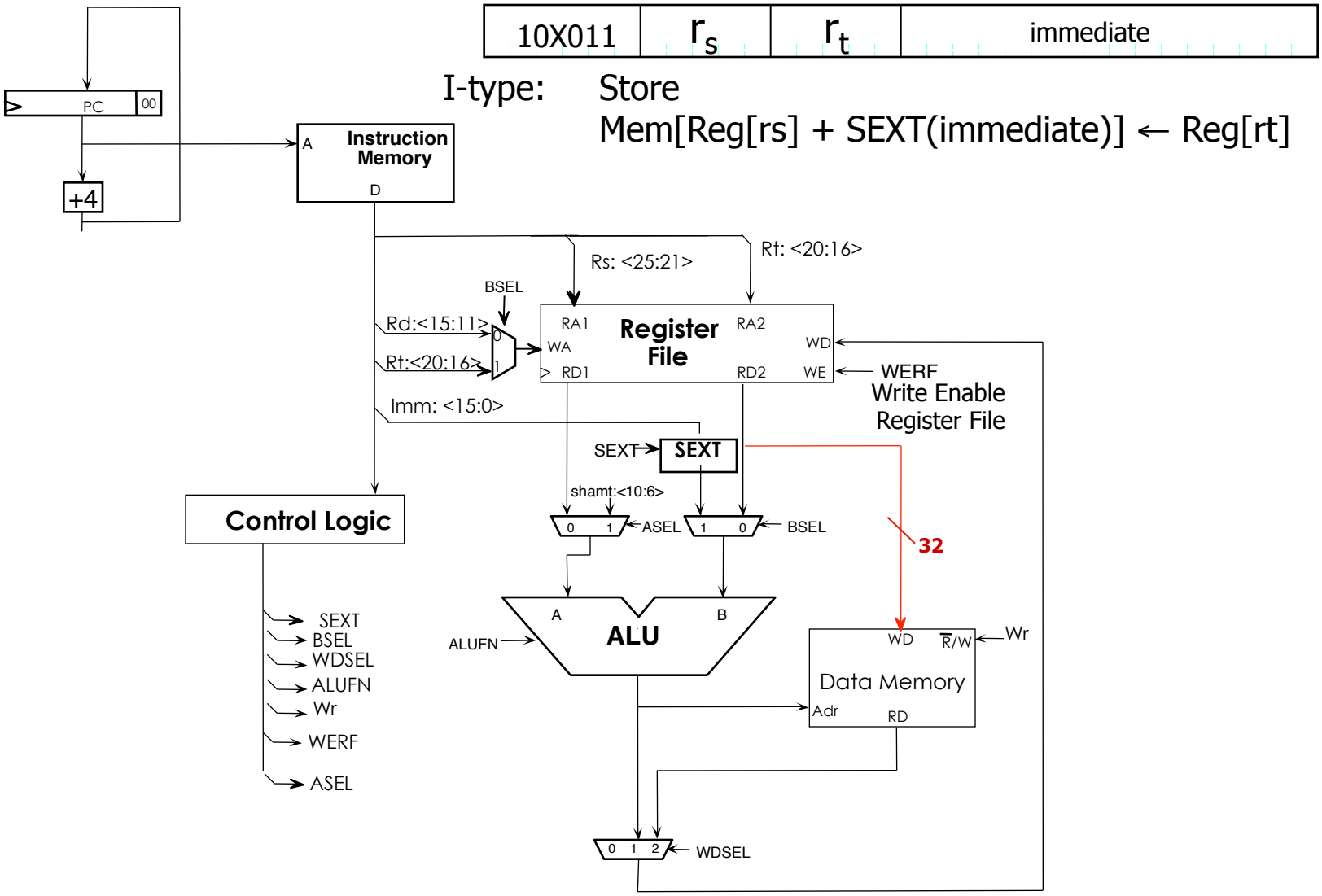
ALU with Immediate



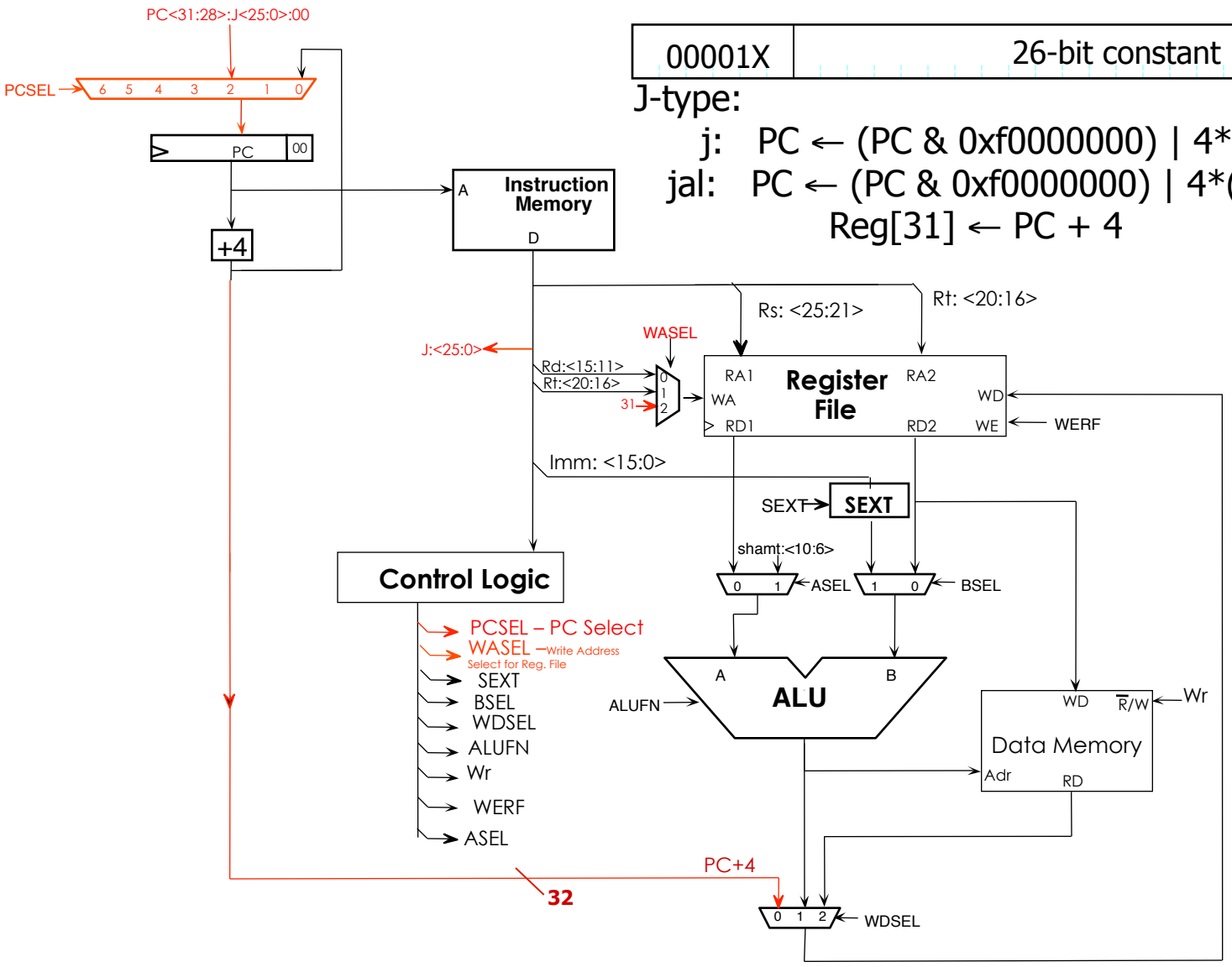
Load Instruction



Store Instruction



JMP Instructions



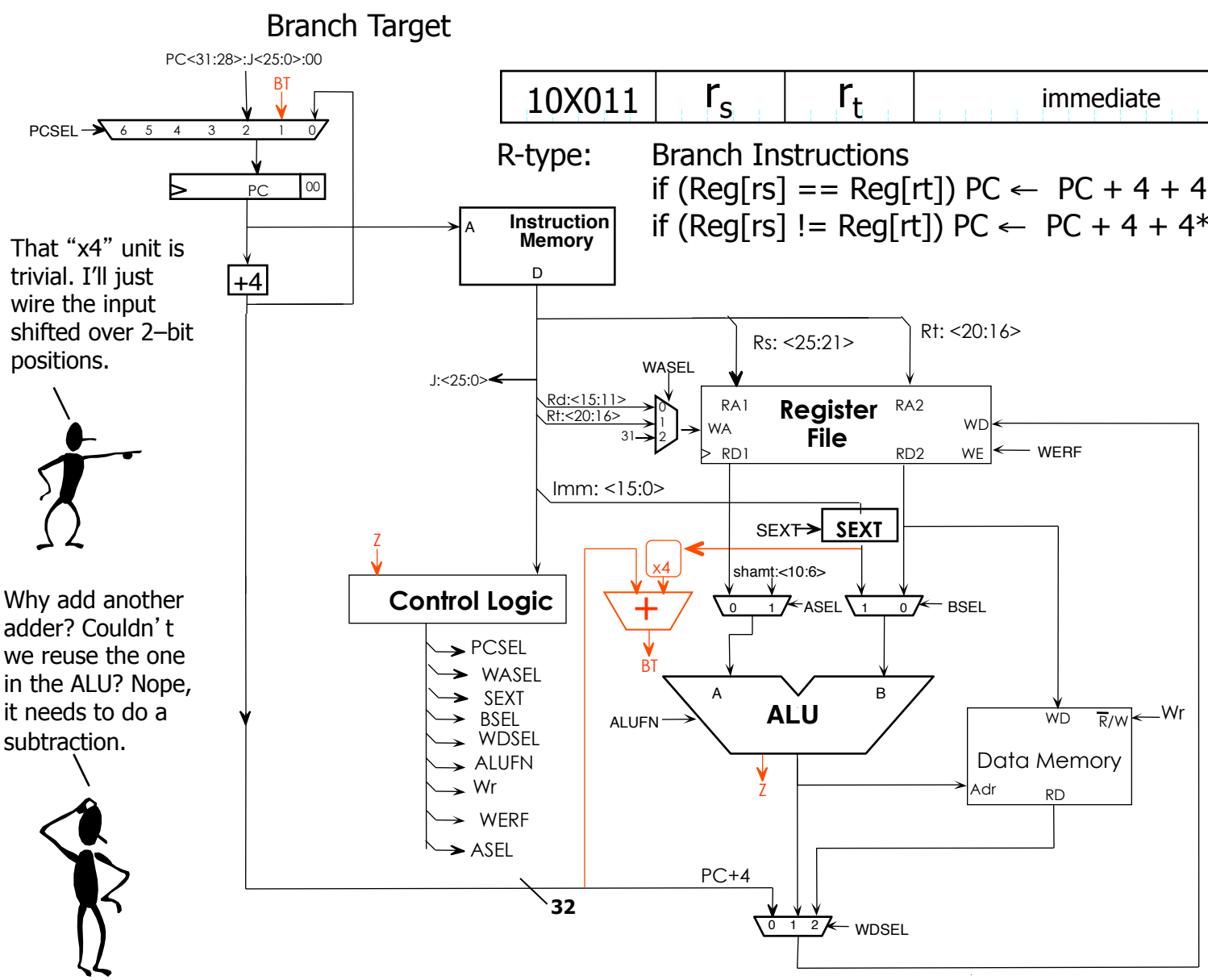
00001X	26-bit constant

J-type:

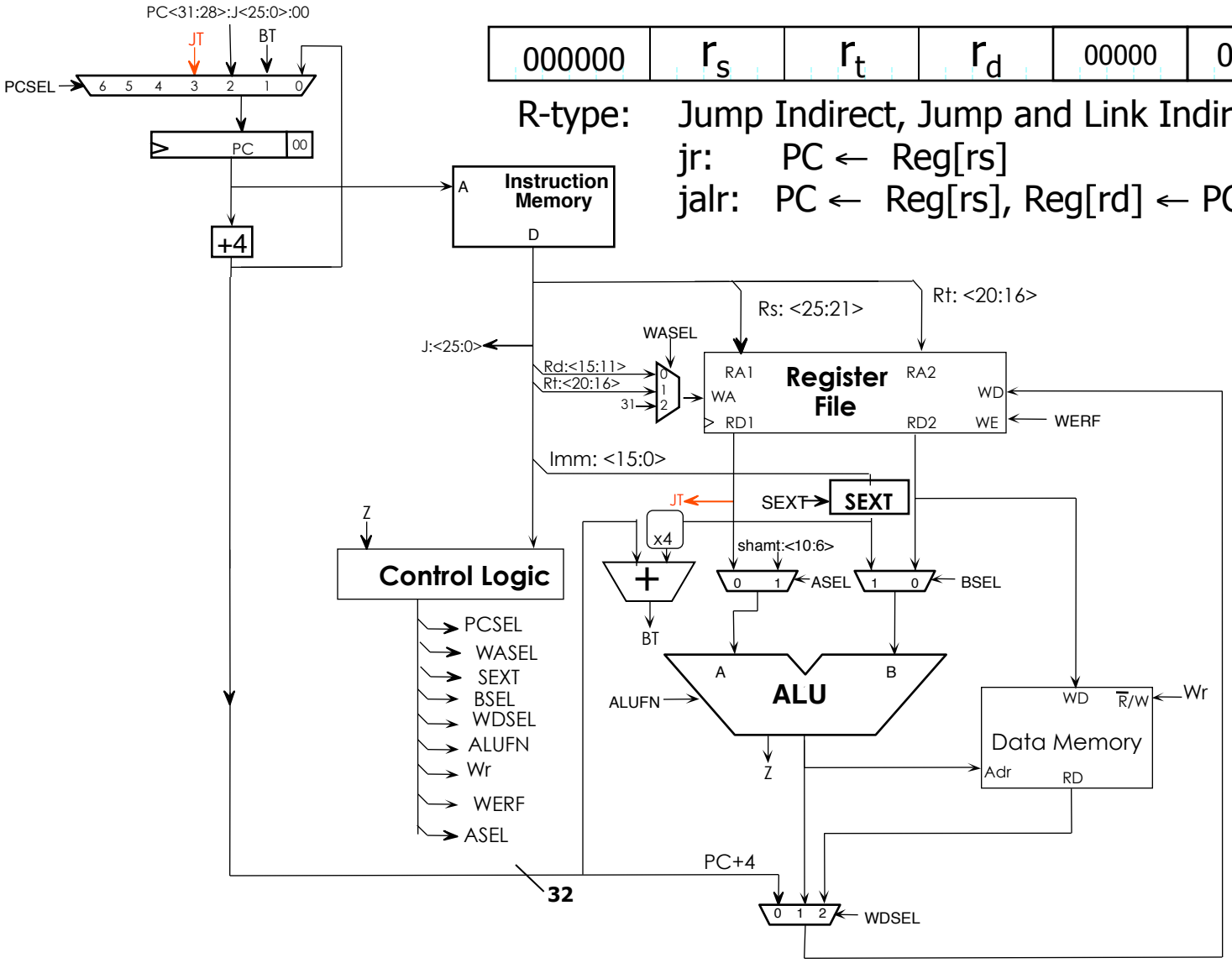
j: PC ← (PC & 0xf0000000) | 4*(immediate)

jal: PC ← (PC & 0xf0000000) | 4*(immediate);
Reg[31] ← PC + 4

BEQ/BNE Instructions



Jump Indirect Instructions



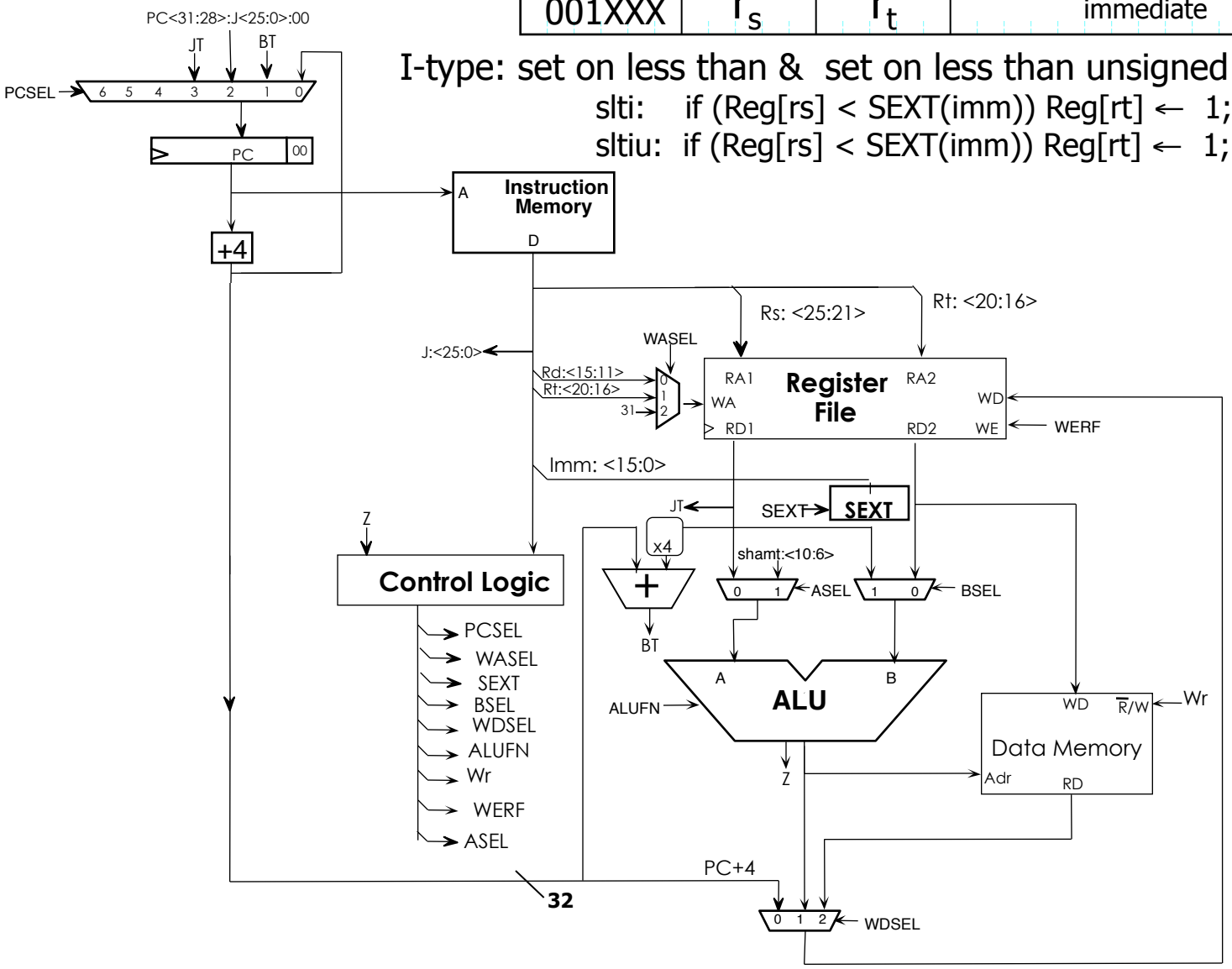
Comparisons

001XXX	r_s	r_t	immediate

I-type: set on less than & set on less than unsigned immediate

slti: if (Reg[rs] < SEXT(imm)) Reg[rt] ← 1; else Reg[rt] ← 0

sltiu: if (Reg[rs] < SEXT(imm)) Reg[rt] ← 1; else Reg[rt] ← 0



Reminder:
To evaluate (A < B) we first compute A-B and look at the flags.

LT = N ⊕ V
LTU = C

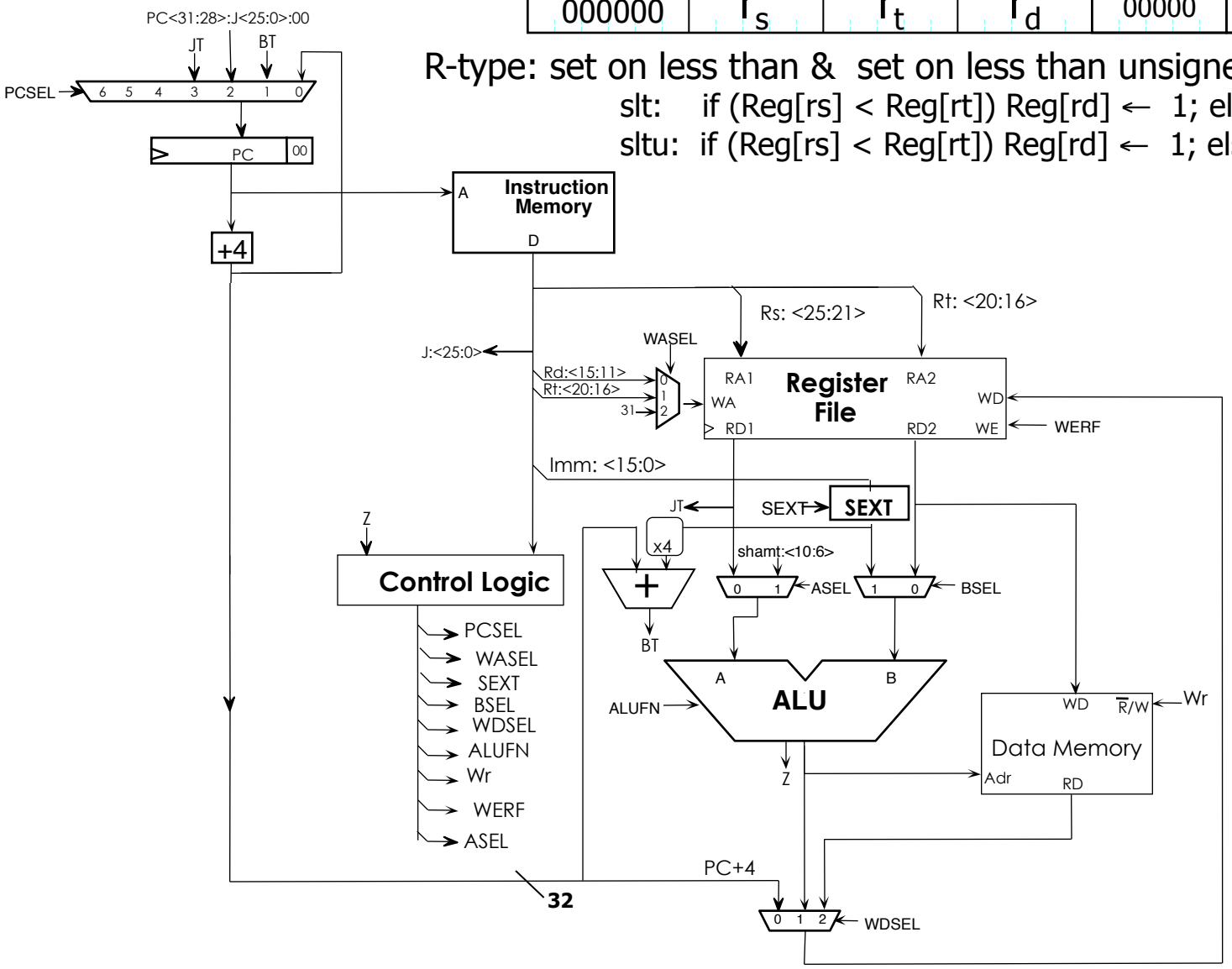
More comparisons

000000	r_s	r_t	r_d	00000	10101X

R-type: set on less than & set on less than unsigned

slt: if (Reg[rs] < Reg[rt]) Reg[rd] ← 1; else Reg[rd] ← 0

sltu: if (Reg[rs] < Reg[rt]) Reg[rd] ← 1; else Reg[rd] ← 0



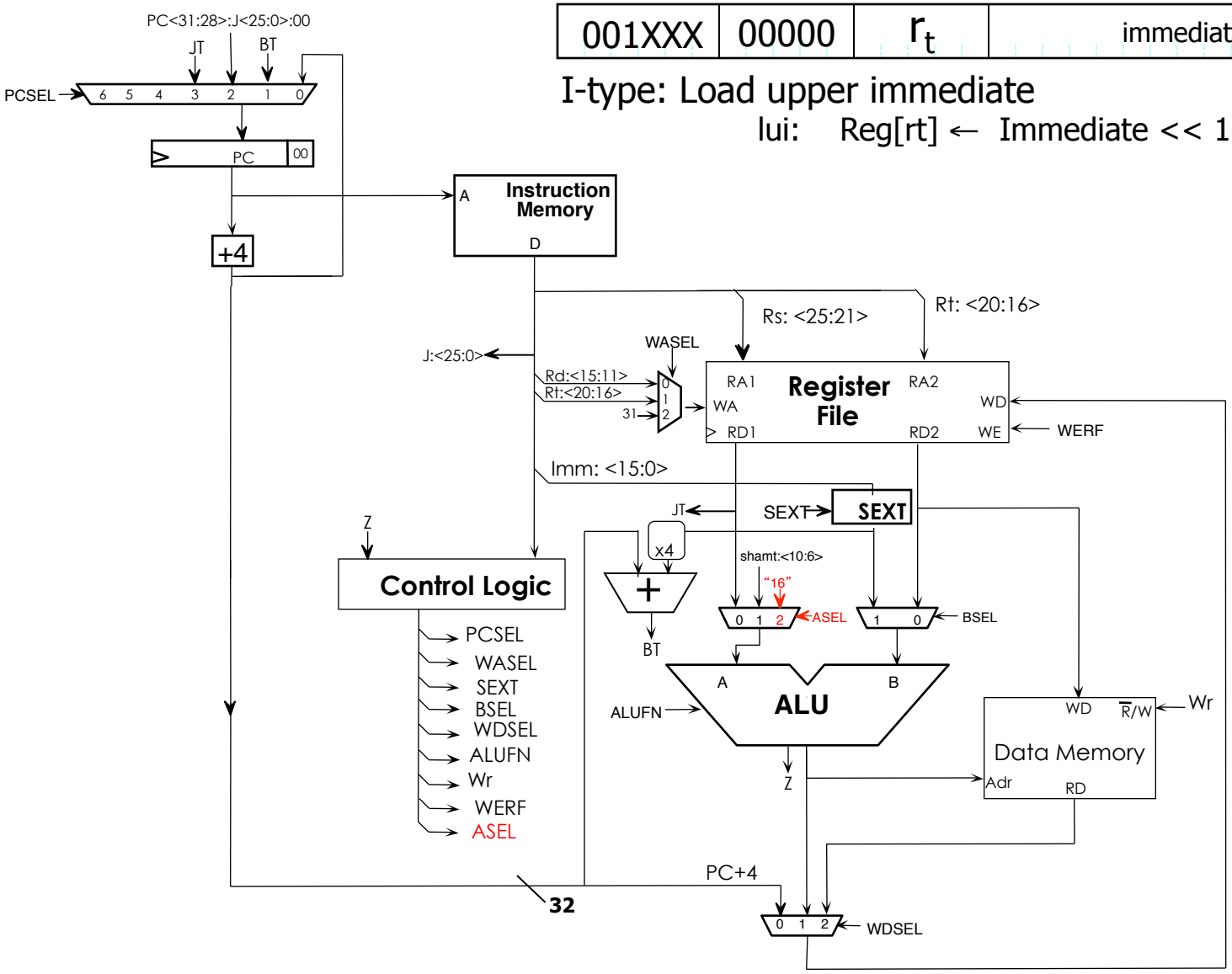
LUI

Shift "immediate" Left 16 bits

001XXX	00000	r_t	immediate

I-type: Load upper immediate

lui: Reg[rt] ← Immediate << 16



Reset, Interrupts, and Exceptions

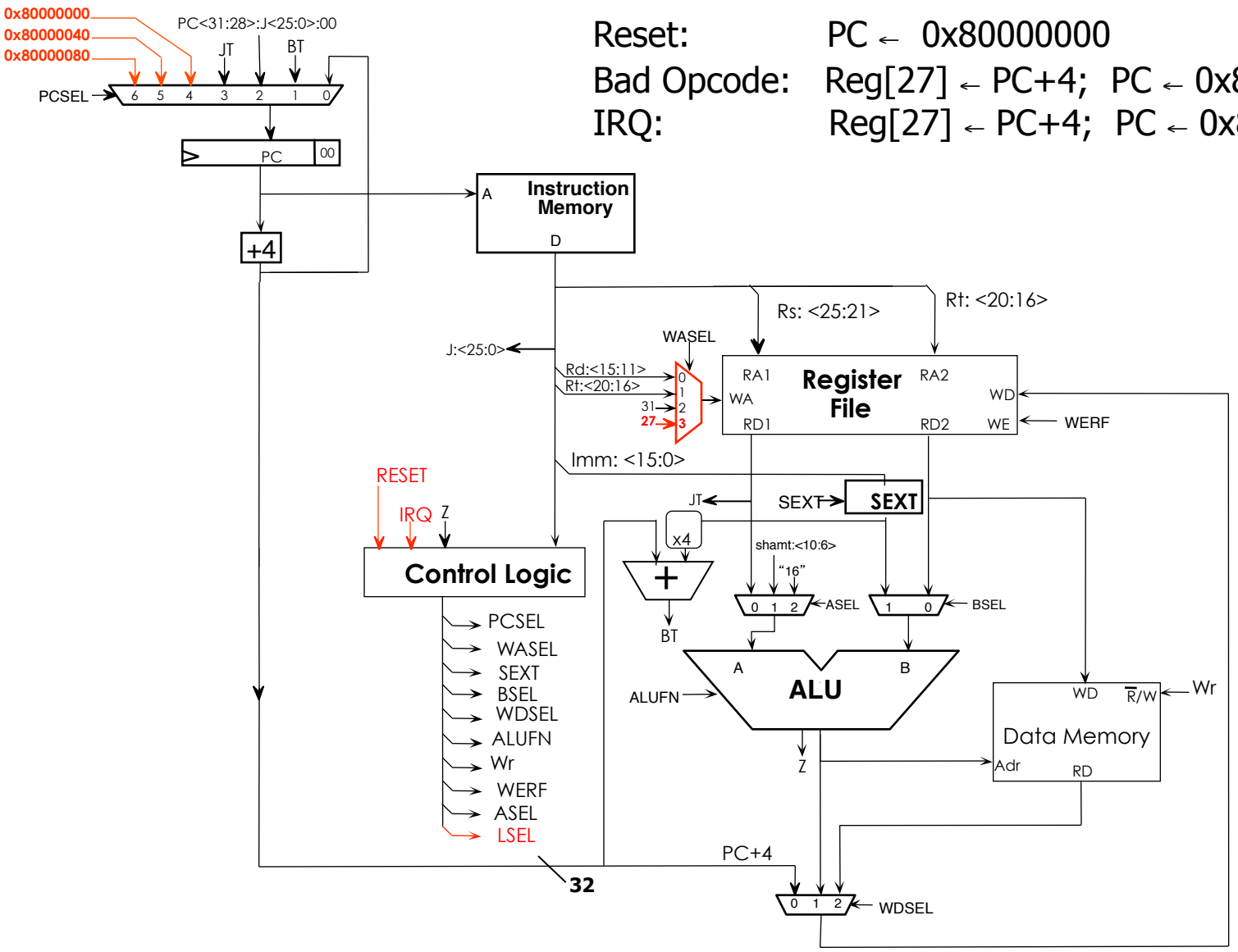
- Upon reset/reboot:
  - Need to set PC to where boot code resides in memory
- Interrupts/Exceptions:
  - any event that causes interruption in program flow
    - FAULTS: e.g., nonexistent opcode, divide-by-zero
    - TRAPS & system calls: e.g., read-a-character
    - I/O events: e.g., key pressed
- How to handle?
  - interrupt current running program
  - invoke exception handler
  - return to program to continue execution
- Registers $k0, $k1 ($26, $27)
  - reserved for operating system (kernel), interrupt handlers
  - any others used must be saved/restored

Exceptions

Reset: PC ← 0x80000000

Bad Opcode: Reg[27] ← PC+4; PC ← 0x80000040

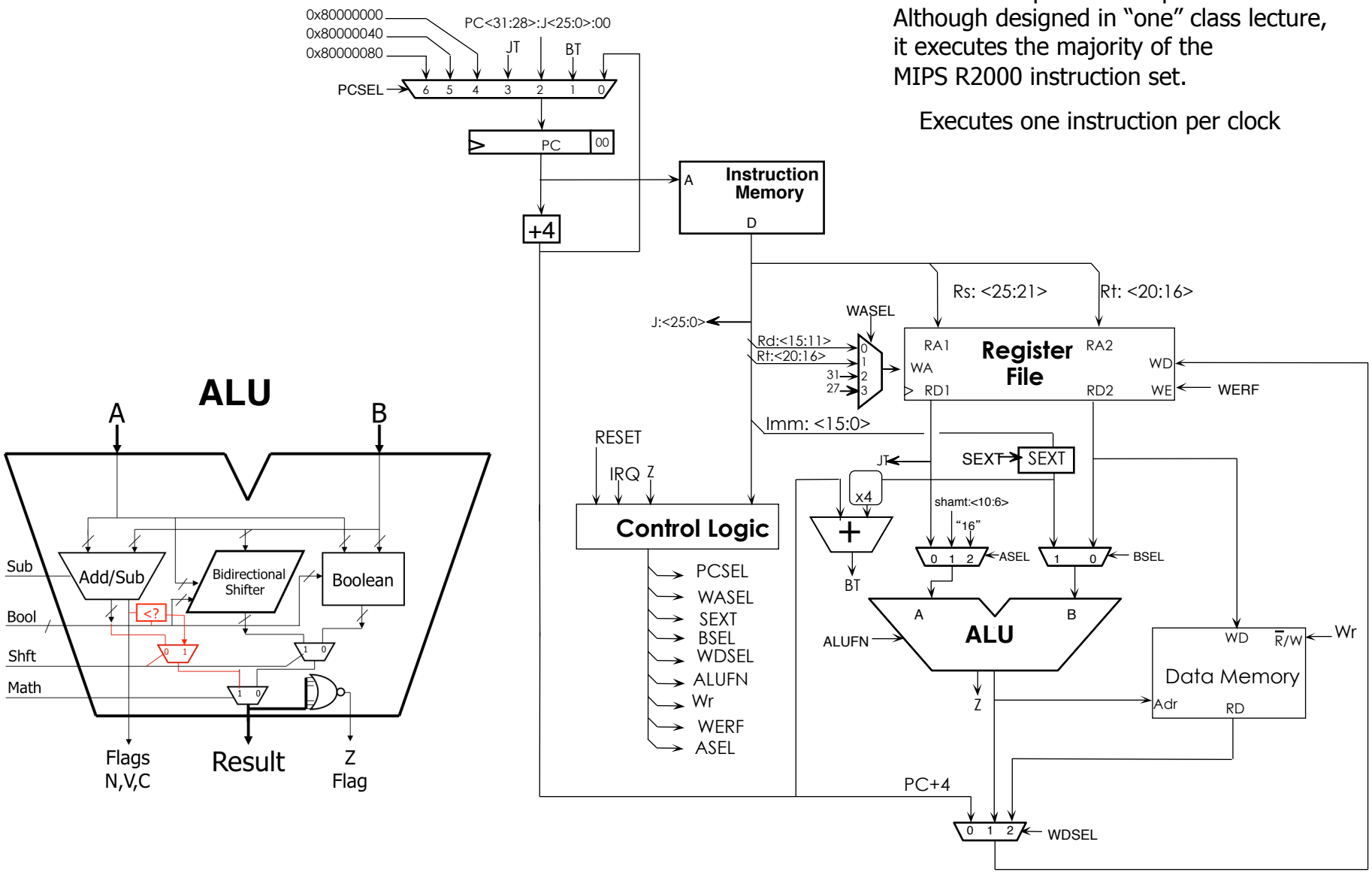
IRQ: Reg[27] ← PC+4; PC ← 0x80000080



MIPS: Our Final Version

This is a complete 32-bit processor. Although designed in "one" class lecture, it executes the majority of the MIPS R2000 instruction set.

Executes one instruction per clock



MIPS Control

- The control unit can be built as a large ROM

Instruction	RESET	IRQ	PCSEL	SEXT	WASEL	WDSEL	ALUFN Sub	ALUFN Bool	ALUFN Shift	ALUFN Math	WR	WERF	ASEL	BSEL
X	1	X	4	0	0	0	0	00	0	0	0	0	0	0
X	0	1	6	0	3	0	0	00	0	0	0	0	0	0
add	0	0	0	0	0	1	0	00	0	1	0	1	0	0
sll														
andi														
lw														
sw														
beq														

Summary

- We have designed a full "miniMIPS" processor!
  - has datapath, which includes registers, ALU
  - instruction and data memories
  - control unit governs everything!

- Next set of classes: some advanced topics
  - memory hierarchy: caches etc.
  - pipelining the processor: benefits and challenges