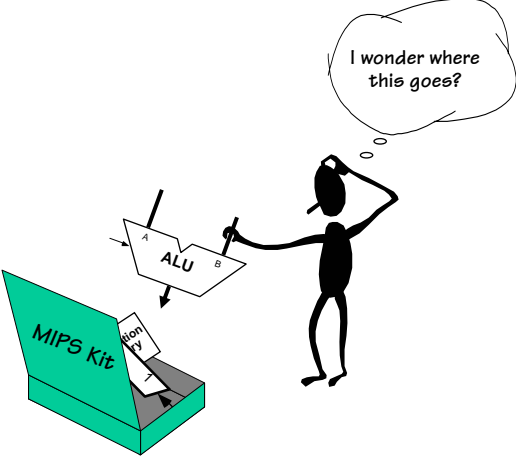


Building a Computer



I wonder where this goes?


Comp 120 – Fall 2005 3/1/05 L15- Building a Computer 1

THIS IS IT!

“Motivating Force”
or
“Inciting Incident”

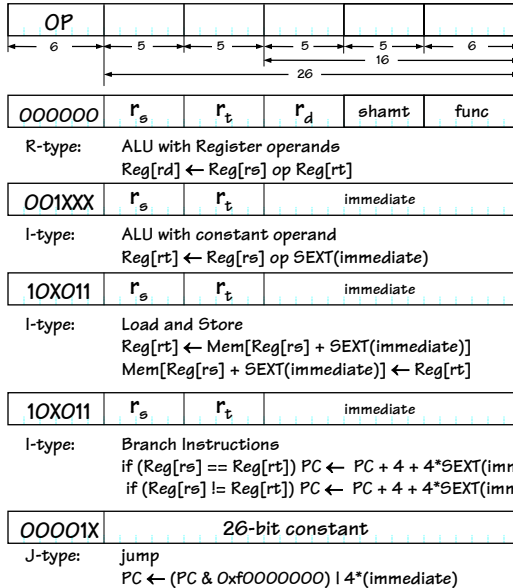
This is the point in the course where the PLOT actually begins. We are now ready to build a computer.

The ingredients are all in place, now it is time to build a legitimate computer. One that executes instructions, much the way any other desktop, PDA, or other computer does.



Comp 120 – Fall 2005 3/1/05 L15- Building a Computer 2

The MIPS ISA



•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

Design Approach

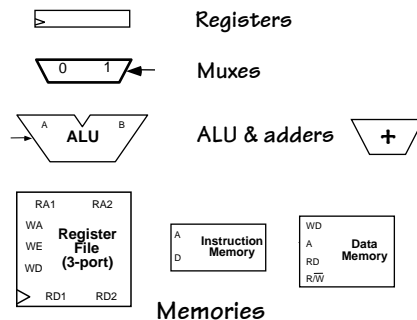
Incremental Featurism

Each instruction class can be implemented using a simple component repertoire. We'll try implementing data paths for each class individually, and merge them (using MUXes, etc).

Steps:

1. 3-Operand ALU instructions
2. ALU w/immediate instructions
2. Load & Store Instructions
3. Jump & Branch instructions
4. Exceptions
5. Merge data paths

Our Bag of Components:



A Few ALU Tweaks

Let's review the ALU that we built a few lectures ago.
(With a few minor additions)

Sub	Bool	Shft	Math	OP
0	XX	0	1	A+B
1	XX	0	1	A-B
X	X0	1	1	0
X	X1	1	1	1
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

Flags V,C N Flag R Z Flag

Comp 120 - Fall 2005 3/1/05 L15- Building a Computer 5

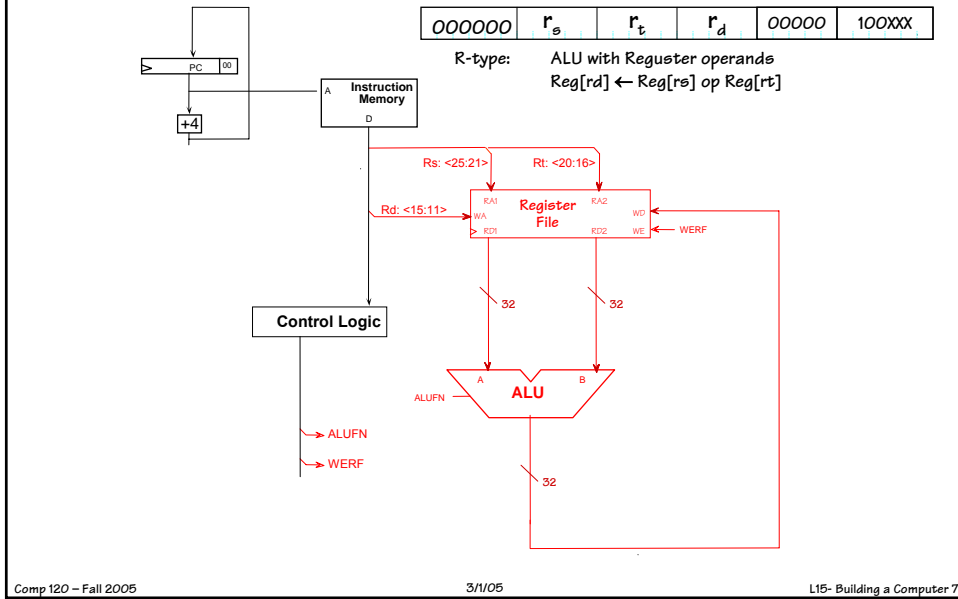
Instruction Fetch/Decode

- Use a counter to FETCH the next instruction:
PROGRAM COUNTER (PC)

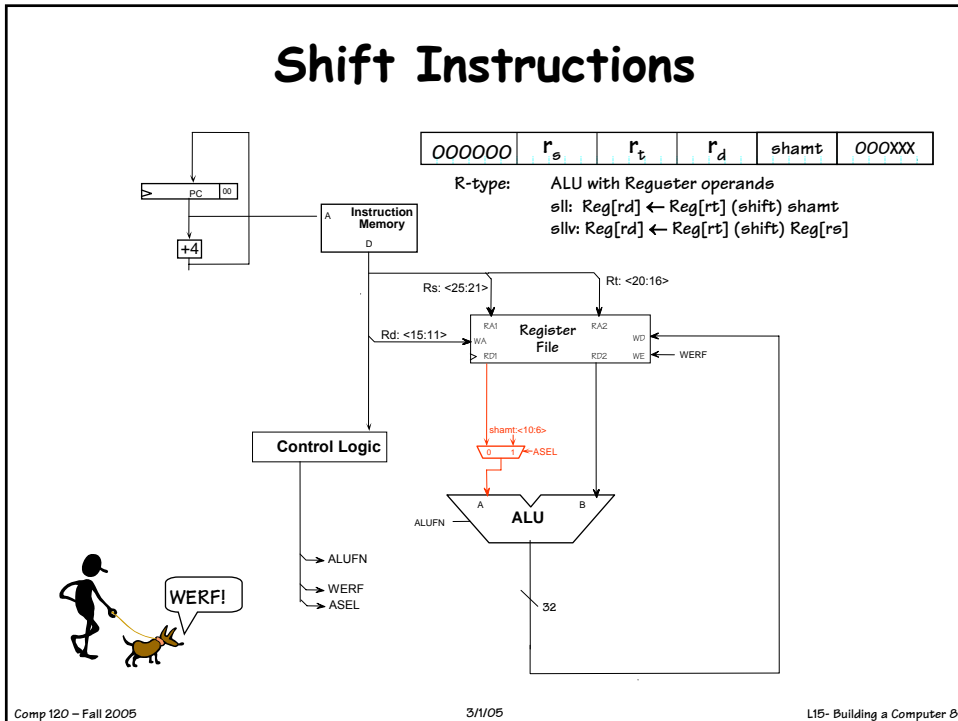
- use PC as memory address
- add 4 to PC, load new value at end of cycle
- fetch instruction from memory
 - use some instruction fields directly (register numbers, 16-bit constant)
 - use bits <31:26> and <5:0> to generate controls

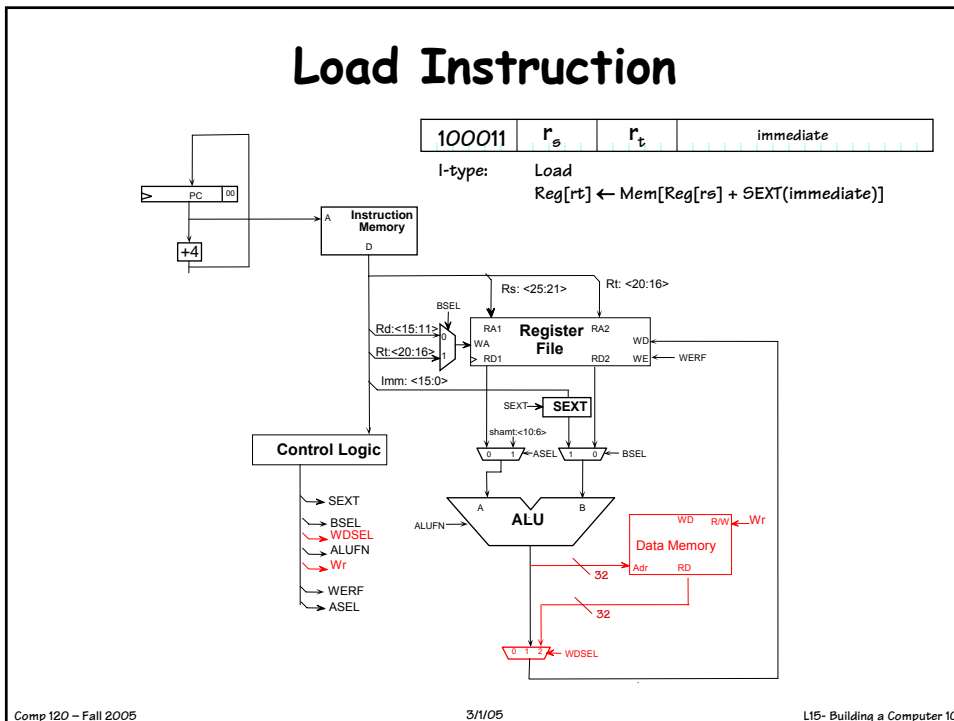
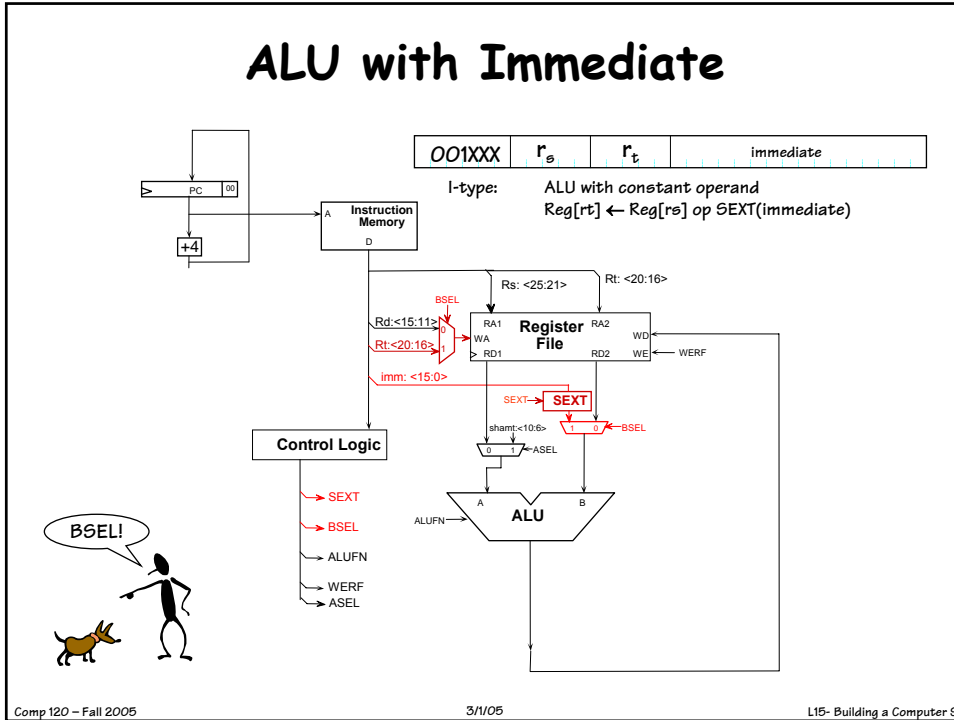
Comp 120 - Fall 2005 3/1/05 L15- Building a Computer 6

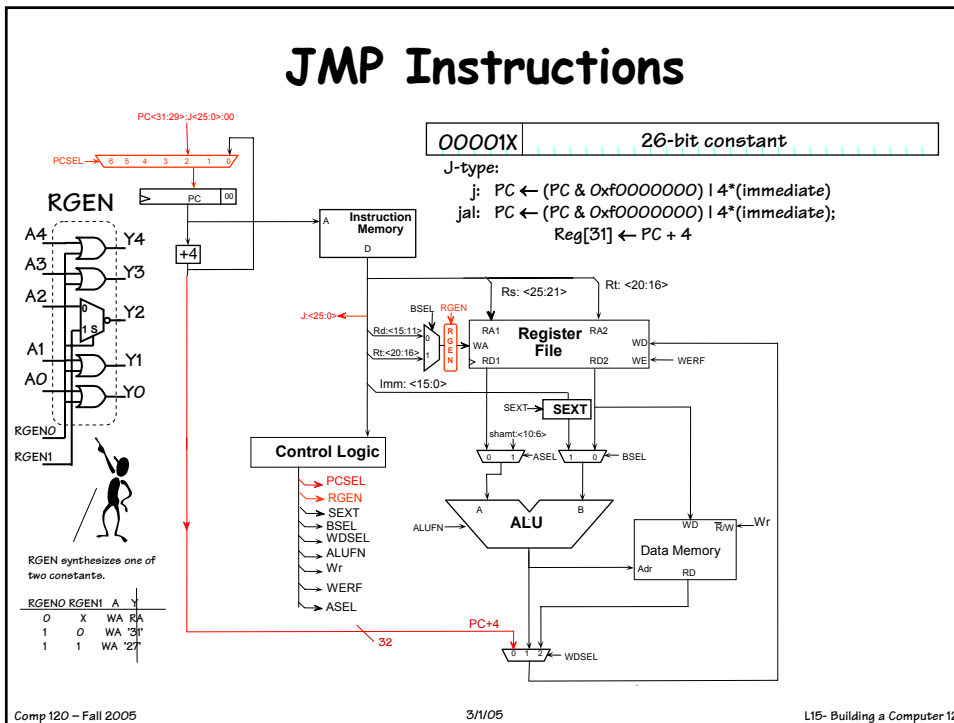
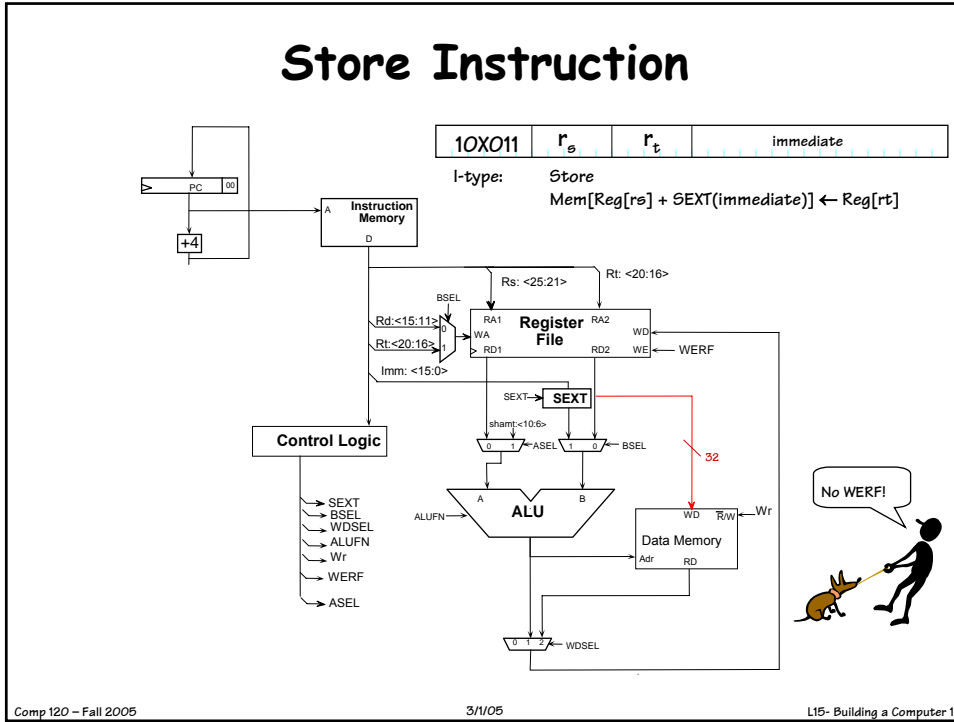
3-Operand ALU Data Path

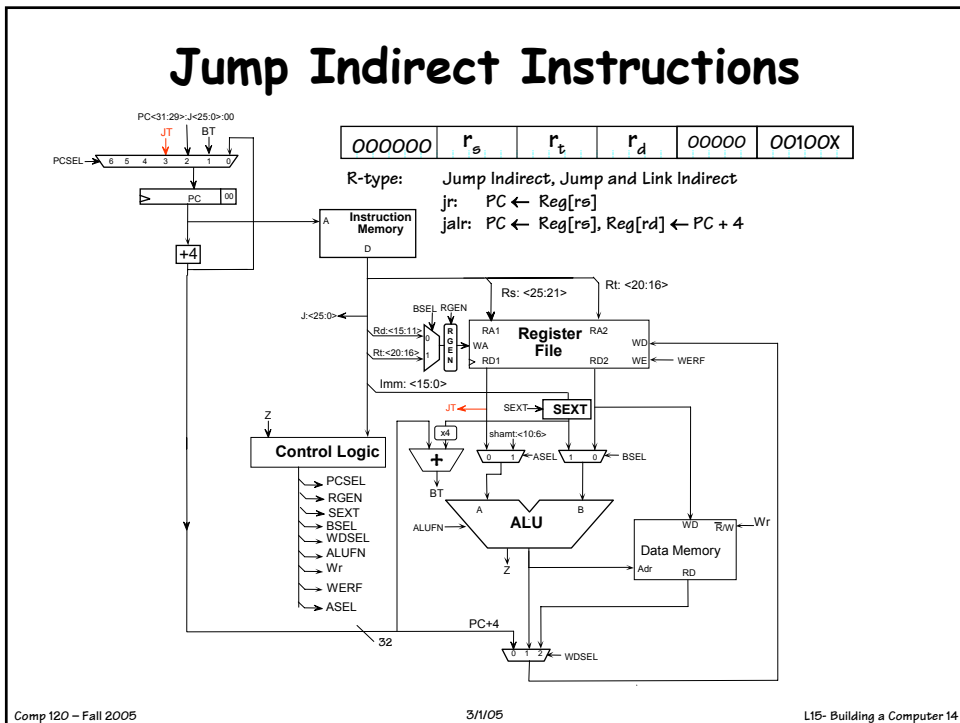
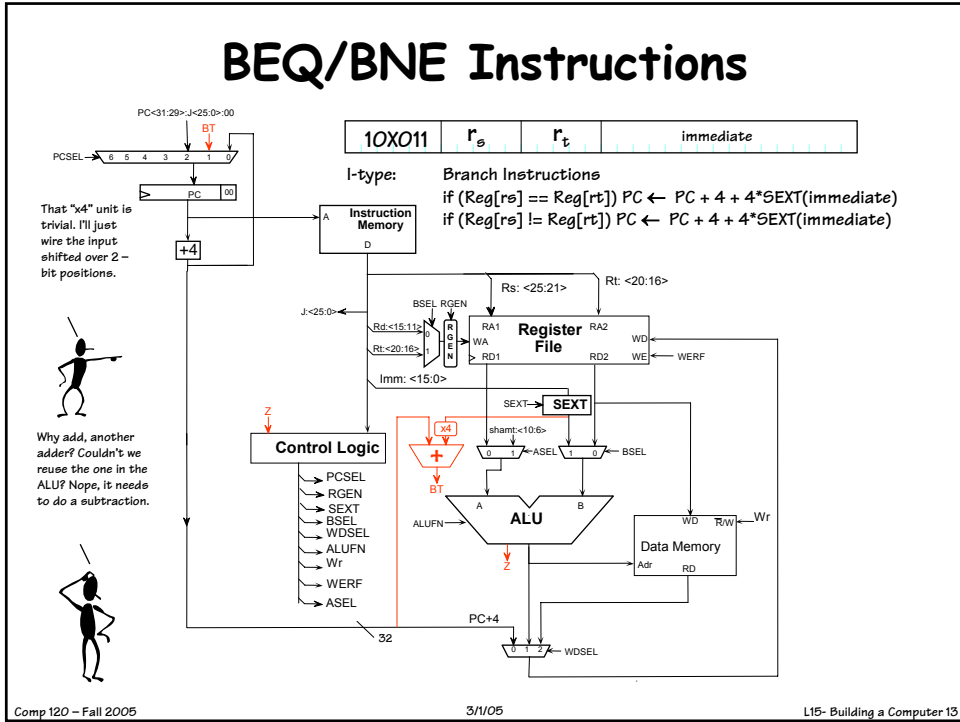


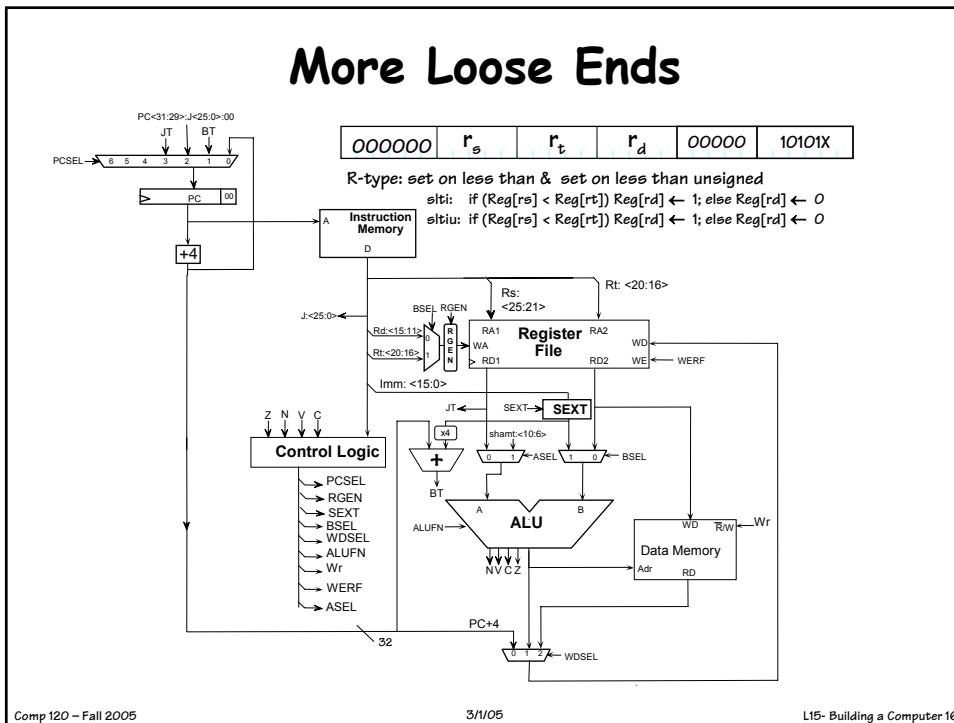
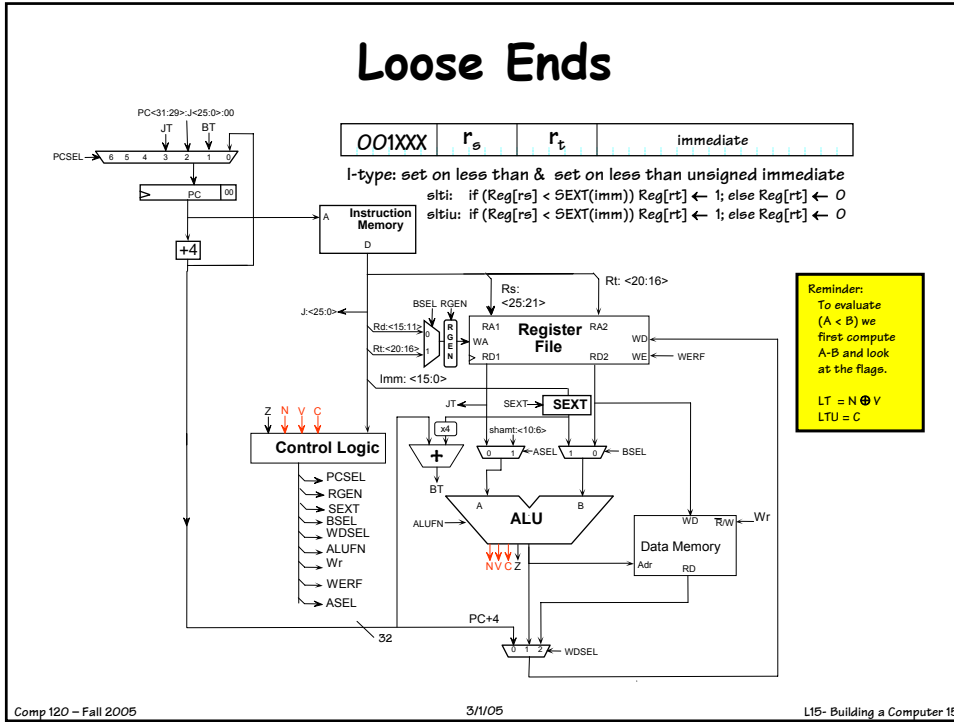
Shift Instructions

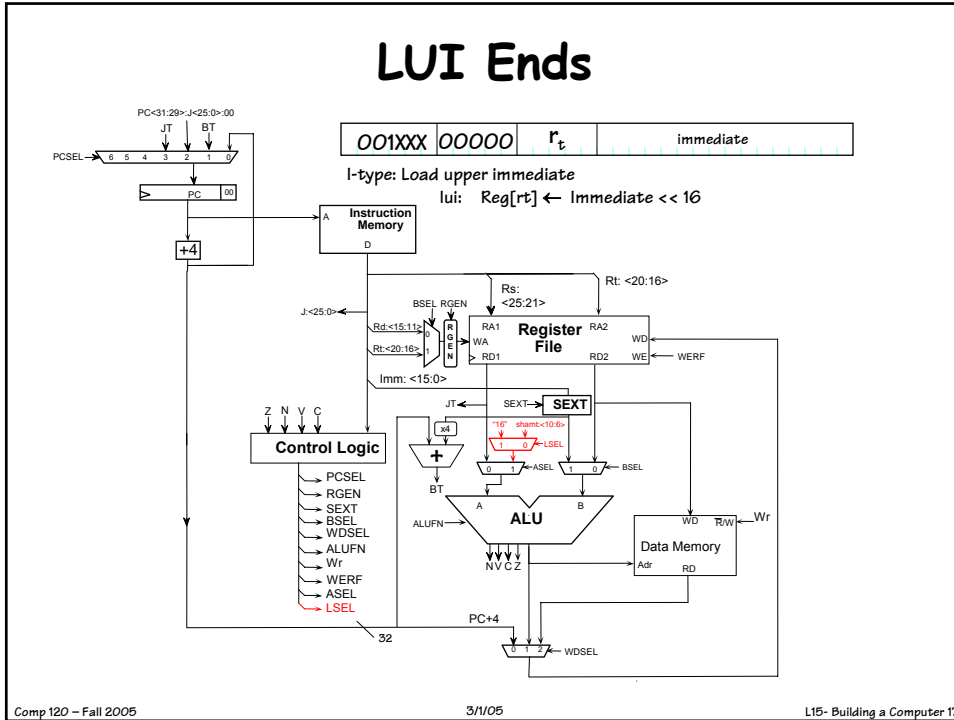












Reset, Interrupts, and Exceptions

FIRST, we need some way to get our machine into a known initial state. This doesn't mean that all registers will be initialized, just that we'll know where to fetch the first instruction. We'll call this control input, RESET

We'd also like **RECOVERABLE INTERRUPTS** for

- **FAULTS** (eg, Illegal Instruction)
 - CPU or SYSTEM generated [synchronous]
- **TRAPS & system calls** (eg, read-a-character)
 - CPU generated [synchronous]
- **I/O events** (eg, key struck)
 - externally generated [asynchronous]

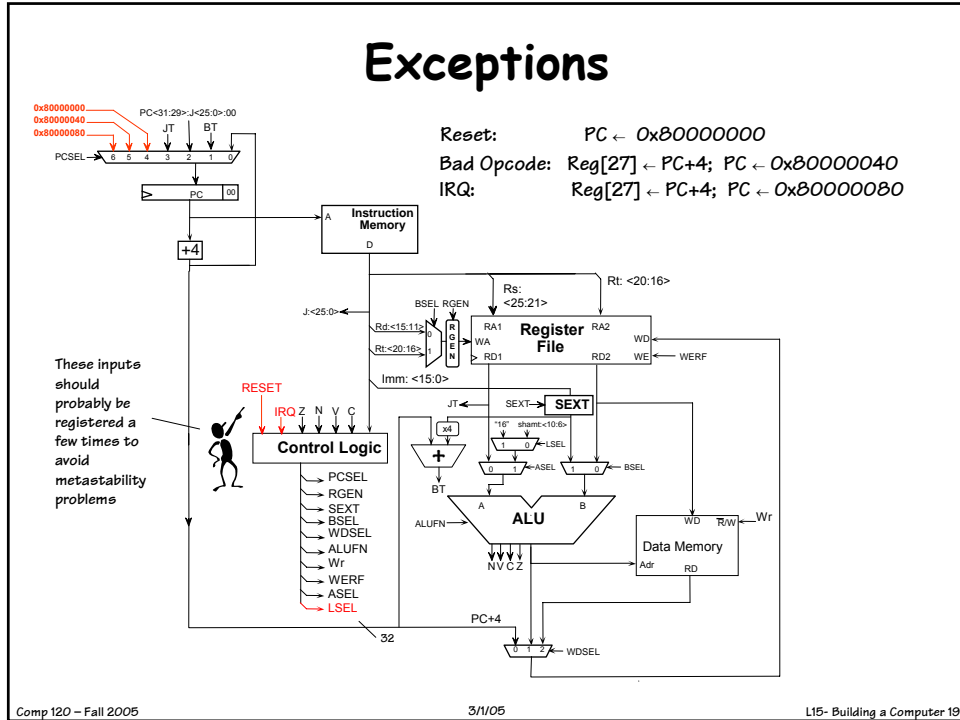
EXCEPTION GOAL: Interrupt running program, invoke exception handler, return to continue execution.

(Implemented as an "agreed" upon Illegal instruction)

These are "Software" notions of synchrony and asynchrony.



Exceptions



MIPS: Our Final Version

