

# Lec 7: Full Processor and Pipelining

**Kavita Bala**  
**CS 3410, Fall 2008**  
Computer Science  
Cornell University

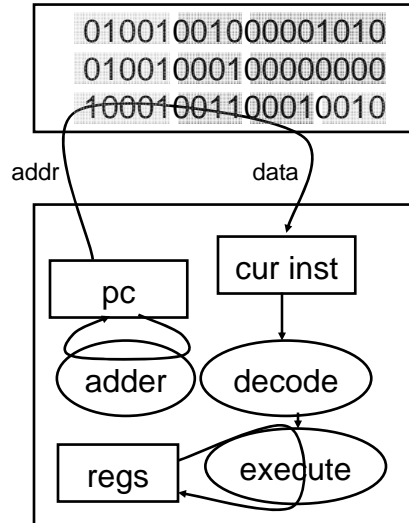
## Announcements

---

- PA 1 out
  - Recitations on it this Tue/Thu/Fri
  
- Don't wait till the last minute
  - We are happy to help
  - Hazards do at end

## Instruction Usage

- Instructions are stored in memory, encoded in binary
- A basic processor
  - fetches
  - decodes
  - executesone instruction at a time



© Kavita Bala, Computer Science, Cornell University

## Instructions

- Load/store architecture
  - Data must be in registers to be operated on
  - Keeps hardware simple
- Emphasis on efficient implementation
- Integer data types:
  - byte: 8 bits
  - half-words: 16 bits
  - words: 32 bits
- MIPS supports signed and unsigned data types

© Kavita Bala, Computer Science, Cornell University

## MIPS Design Principles

---

- Simplicity favors regularity
  - 32 bit instructions
- Smaller is faster
  - Small register file
- Make the common case fast
  - Include support for constants
- Good design demands good compromises
  - Support for different type of interpretations/classes

© Kavita Bala, Computer Science, Cornell University

## Arithmetic Instructions

---

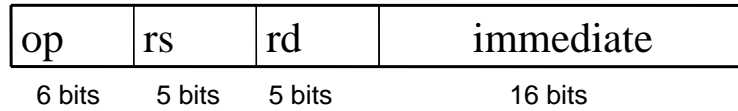
op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- if  $op == 0$  &&  $func == 0x21$  ADD rd, rs, rt
  - $R[rd] = R[rs] + R[rt]$  (unsigned)
- if  $op == 0$  &&  $func == 0x23$  AND rd, rs, rt
  - $R[rd] = R[rs] \& R[rt]$  (unsigned)
- if  $op == 0$  &&  $func == 0x25$  OR rd, rs, rt
  - $R[rd] = R[rs] | R[rt]$

© Kavita Bala, Computer Science, Cornell University

## Arithmetic Instructions (2)

---



- if op == 8
    - $R[rd] = R[rs] + \text{sign\_extend}(\text{immediate})$
  - if op == 12
    - $R[rd] = R[rs] \& \text{immediate}$
- ADDI rd, rs, val

ADDIU rd, rs, val

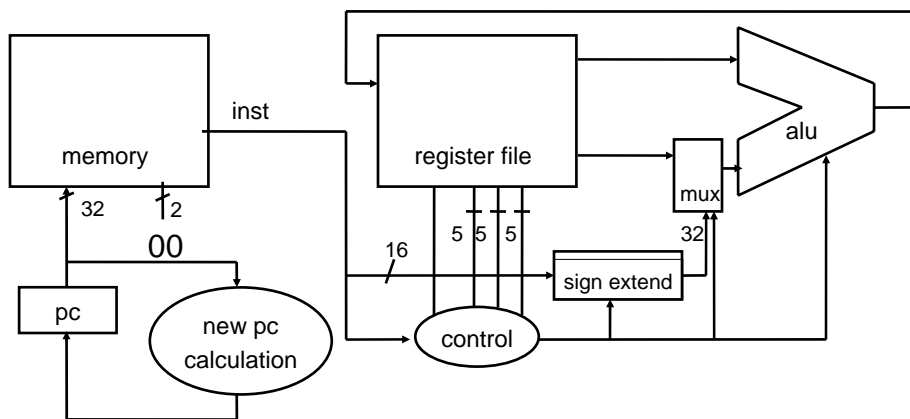
ANDI rd, rs, val

ORI rd, rs, val

© Kavita Bala, Computer Science, Cornell University

## Arithmetic Ops with Immediates

---



© Kavita Bala, Computer Science, Cornell University

## MIPS Instruction Types

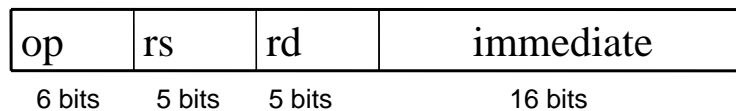
---

- Arithmetic/Logical
  - three operands: result + two sources
  - operands: registers, 16-bit immediates
  - signed and unsigned versions
- Memory Access
  - load/store between registers and memory
  - half-word and byte operations
- Control flow
  - conditional branches: pc-relative addresses
  - jumps: fixed offsets

© Kavita Bala, Computer Science, Cornell University

## Memory Operations

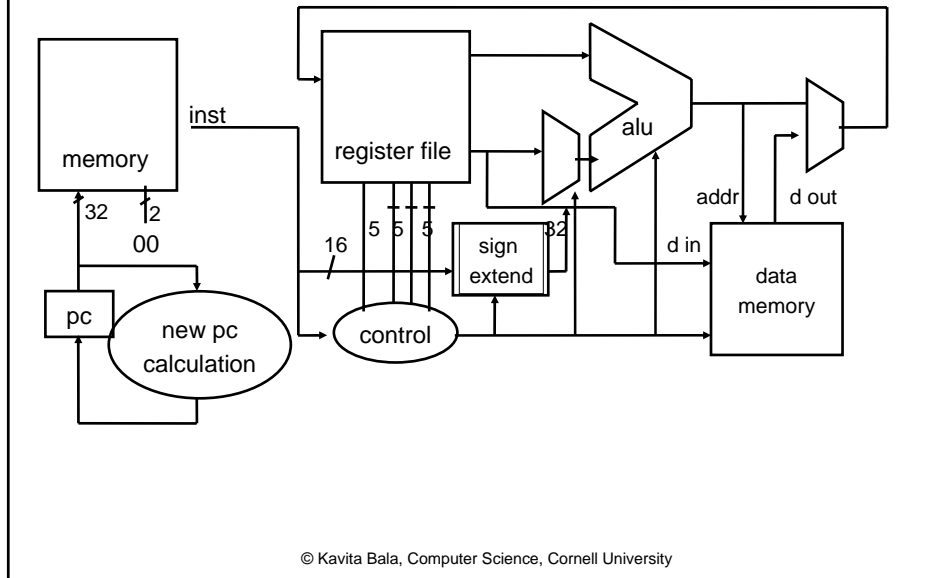
---



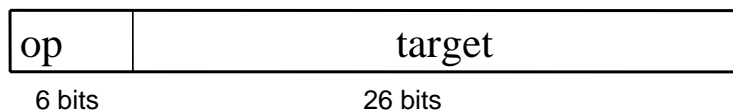
- lb, lbu, lh, lhu, lw
- sb, sh, sw
- Examples  $rs = r4 = \text{addr of array}$ ;  $r3 = rd$ 
  - lw r3, 0(r4)    int array[32]; x = array[0]
  - lw r3, 16(r4)    int array[32]; x = array[4]
  - sw r3, 0(r4)    array[0] = x;

© Kavita Bala, Computer Science, Cornell University

## Store



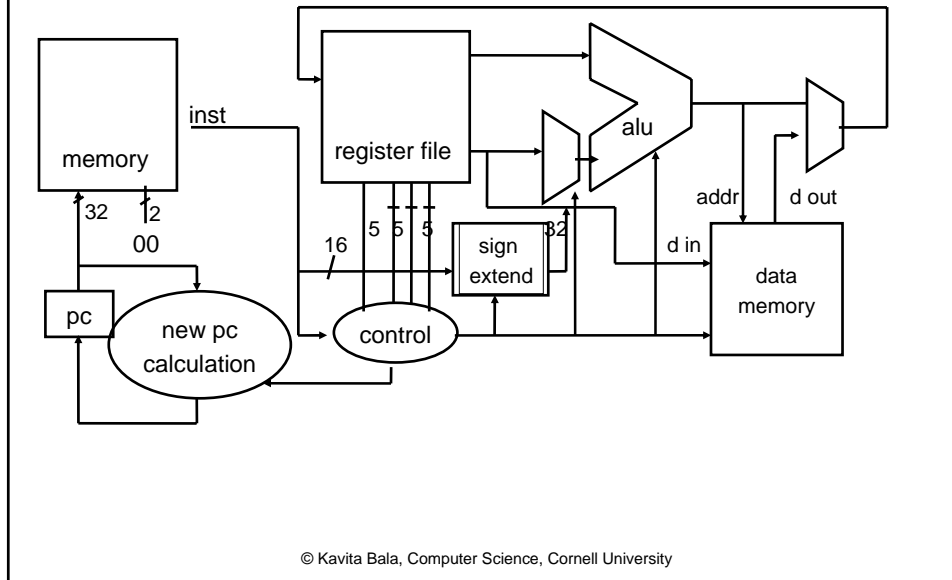
## Control Flow (Absolute Jump)



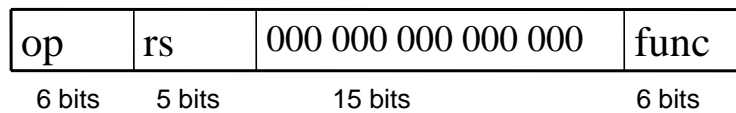
- j, jal
- Absolute addressing
- new PC = high 4 bits of current PC || target || 00
  - Cannot jump from 0xffff000000000000 to 0x0000100000000000
  - Better to make all instructions fit in 32 bits than to support really large absolute jumps
- Examples
  - j L01                      goto L01

© Kavita Bala, Computer Science, Cornell University

## Absolute Jump



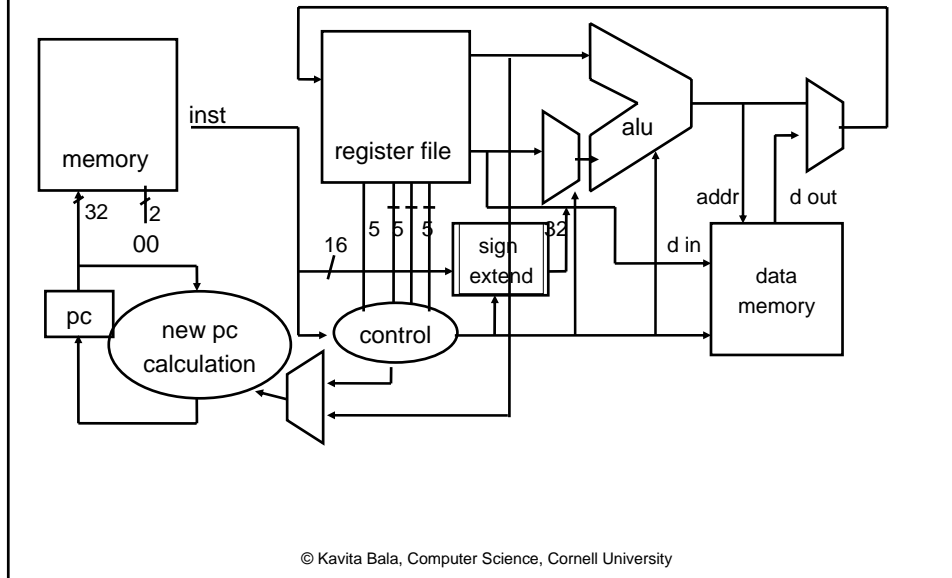
## Control Flow (Jump Register)



- jr rs
- new PC = R[rs]
- Can jump to any address stored in a register

© Kavita Bala, Computer Science, Cornell University

## Jump Register



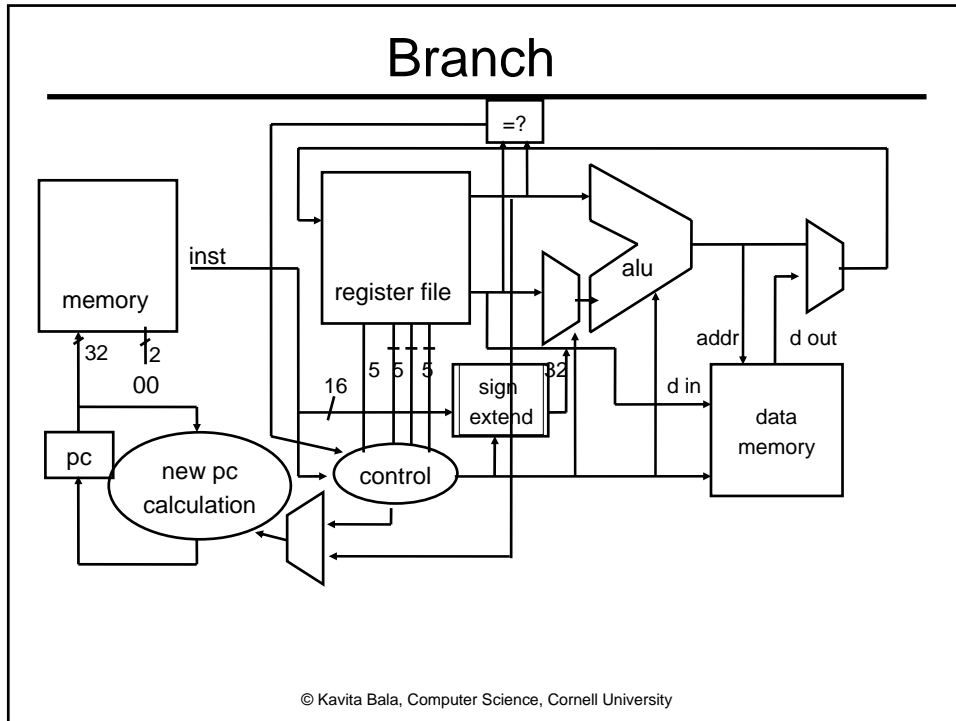
## Control Flow (Branches)



- Some branches depend on the relative values of two registers
- if  $op == 4$  # BEQ
  - if  $R[rs] == R[rt]$ 
    - $new\ PC = old\ PC + sign\_extend(immediate \ll 2)$
- BEQ, BNE

© Kavita Bala, Computer Science, Cornell University



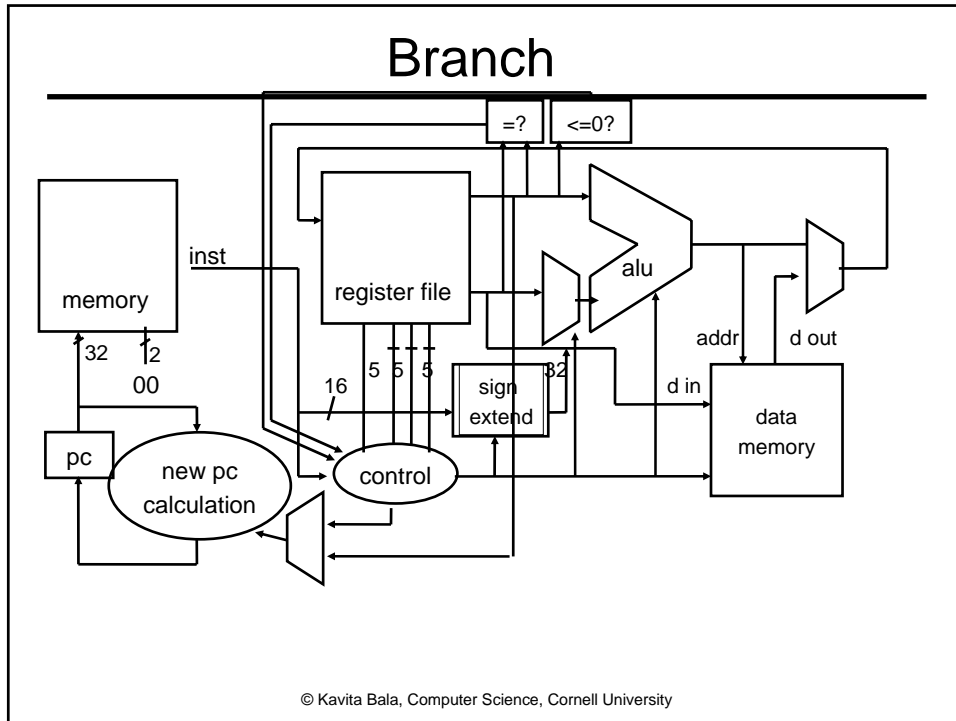


## Control Flow (Branches)

op	rs	subop	immediate
6 bits	5 bits	5 bits	16 bits

- Some branches depend on the value of a single register
- if  $op == 1$  &&  $subop == BLTZ$ 
  - if  $R[rs] < 0$ 
    - $new\ PC = old\ PC + sign\_extend(immediate \ll 2)$
- BGEZ, BGTZ, BLTZ, BLEZ

© Kavita Bala, Computer Science, Cornell University



## Examples

- $A[12] = h + A[8]$
- `lw, $t0, 32($s3)`
- `add $t0, $s2, $t0`
- `sw $t0, 48($s3)`

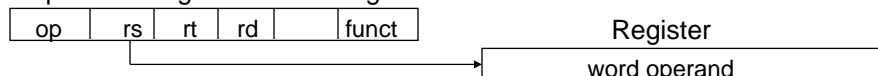
## Example

- if (i == j) f = g + h else f = g - h
- bne \$s3, \$s4, Else
- add \$s0, \$s1, \$s2
- j Exit
- nop
- Else: sub \$s0, \$s1, \$s2
- nop
- Exit:

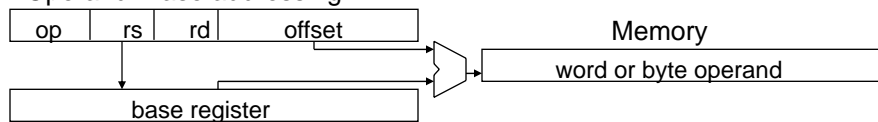
© Kavita Bala, Computer Science, Cornell University

## MIPS Addressing Modes

### 1. Operand: Register addressing



### 2. Operand: Base addressing

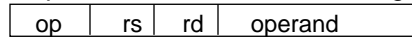


© Kavita Bala, Computer Science, Cornell University

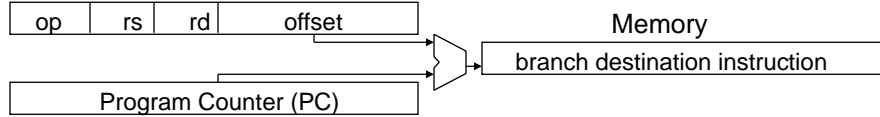
# MIPS Addressing Modes

---

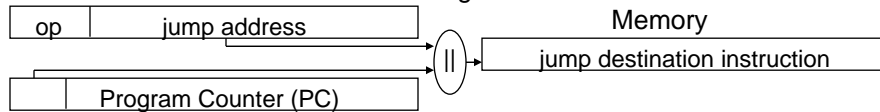
3. Operand: Immediate addressing



4. Instruction: PC-relative addressing



5. Instruction: Pseudo-direct addressing



© Kavita Bala, Computer Science, Cornell University

# Summary

---

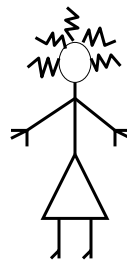
- Full-blown processor

© Kavita Bala, Computer Science, Cornell University

# Pipelining

**Kavita Bala**  
**CS 3410, Fall 2008**  
Computer Science  
Cornell University

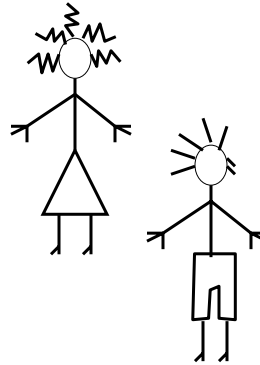
- 
- Alice



---

- Alice

- Bob



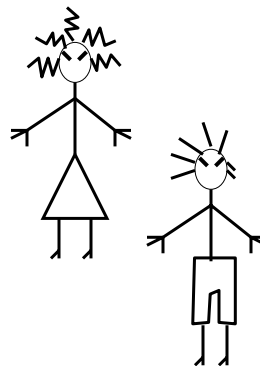
© Kavita Bala, Computer Science, Cornell University

---

- Alice

- Bob

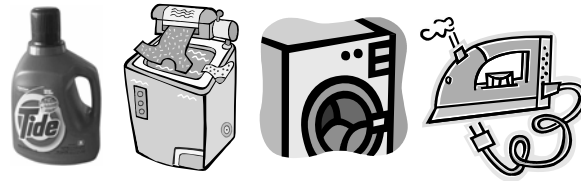
- They don't like each other!



© Kavita Bala, Computer Science, Cornell University

## The Laundry

---

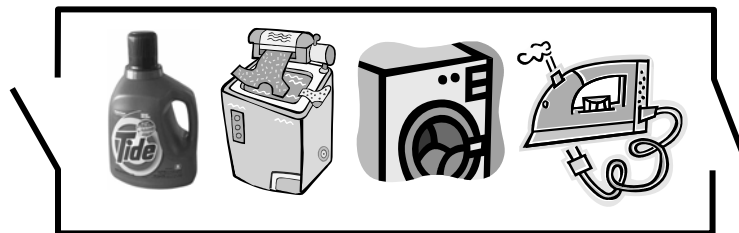


- Four sequential tasks

© Kavita Bala, Computer Science, Cornell University

## Laundry Room Design #1

---

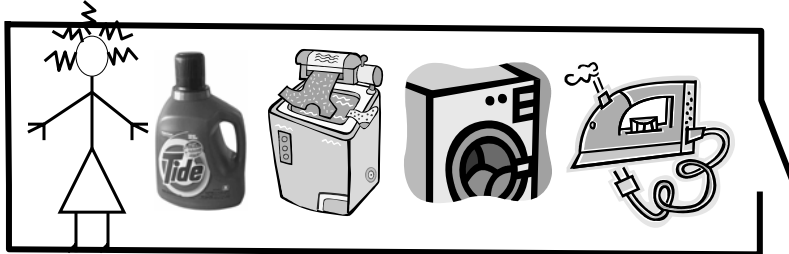


- A large room with a one entry-door and one exit-door

© Kavita Bala, Computer Science, Cornell University

## Laundry Room Design #1

---

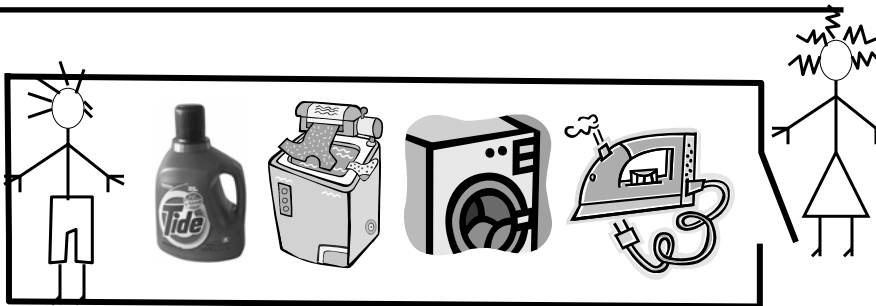


- First Alice owns the room

© Kavita Bala, Computer Science, Cornell University

## Laundry Room Design #1

---

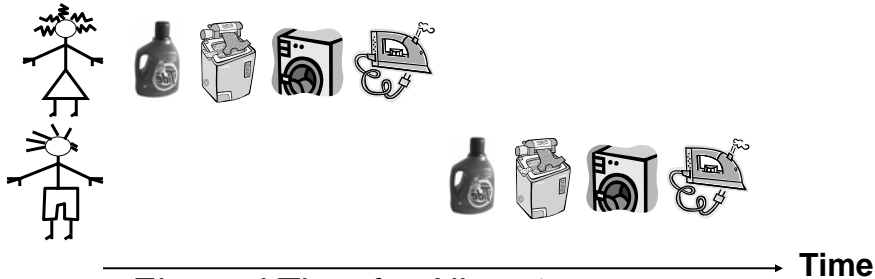


- First Alice owns the room
- Bob can enter as soon as she is done
- No possibility for Alice and Bob to fight

© Kavita Bala, Computer Science, Cornell University



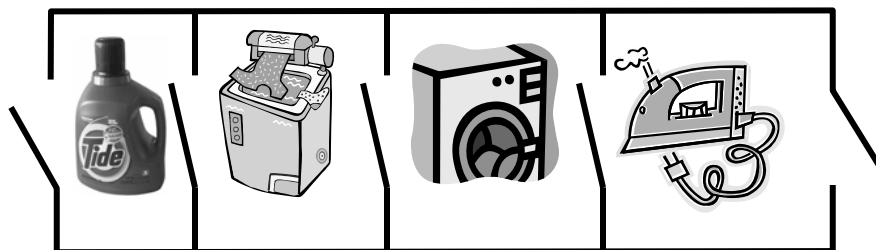
## Laundry Room Design #1



- Elapsed Time for Alice: 4
- Elapsed Time for Bob: 4
- Elapsed Time for both: 8
- A better laundry room can improve utilization and speed up task completion

© Kavita Bala, Computer Science, Cornell University

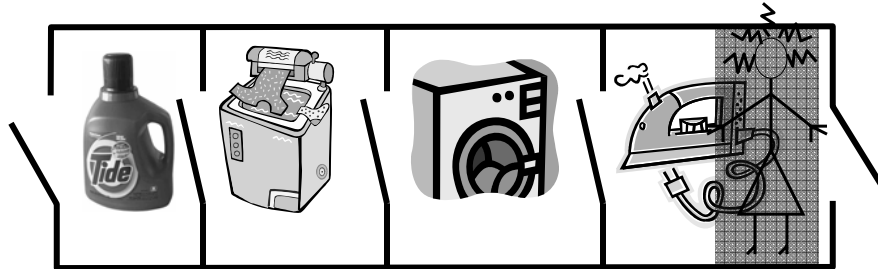
## Laundry Room Design #2



- The room is partitioned into stages
- One person owns a stage at a time, the room can hold up to four people simultaneously

© Kavita Bala, Computer Science, Cornell University

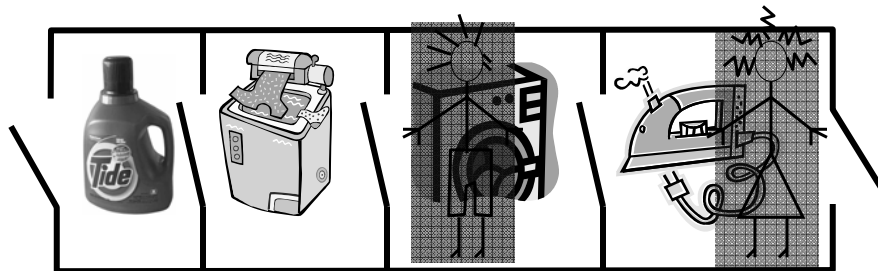
## Laundry Room Design #2



Alice

© Kavita Bala, Computer Science, Cornell University

## Laundry Room Design #2

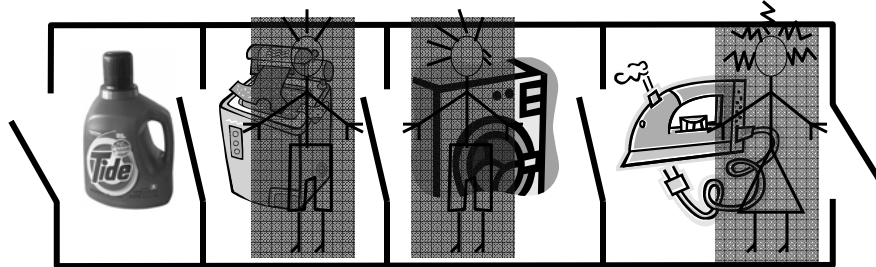


Bob

Alice

© Kavita Bala, Computer Science, Cornell University

## Laundry Room Design #2



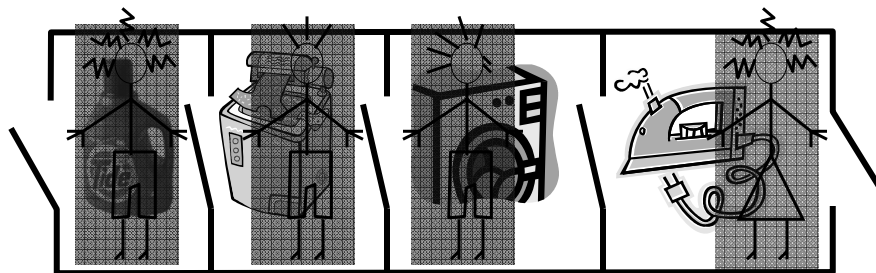
Charlie

Bob

Alice

© Kavita Bala, Computer Science, Cornell University

## Laundry Room Design #2



Dave

Charlie

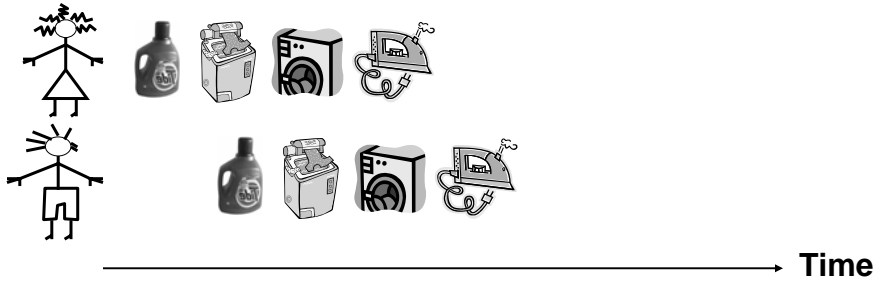
Bob

Alice

© Kavita Bala, Computer Science, Cornell University

## Laundry Room Design #2

---

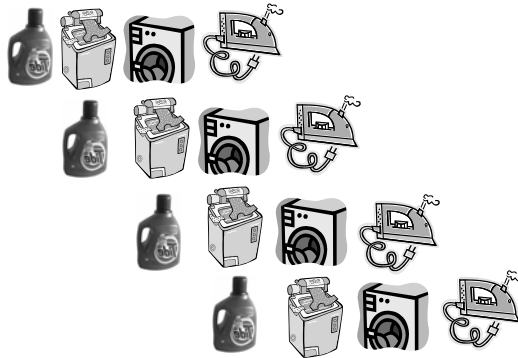


- Elapsed Time for Alice: 4
- Elapsed Time for Bob: 4
- Elapsed Time for both: 5!!!

© Kavita Bala, Computer Science, Cornell University

## Throughput is good

---







- What about latency?

© Kavita Bala, Computer Science, Cornell University

## Look at Real Possible Numbers

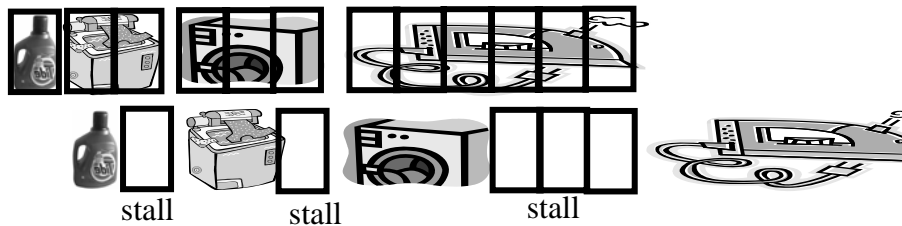
---

-  15 min
-  30 min
-  45 min
-  90 min

© Kavita Bala, Computer Science, Cornell University

## Impact

---

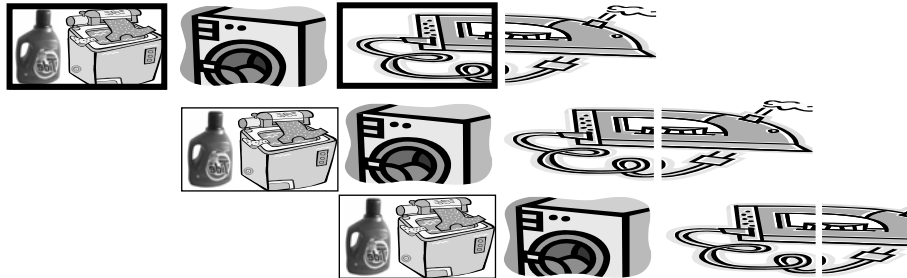


- Latency: 180 min
- Throughput: Batch every 90 min  
– Bottleneck!

© Kavita Bala, Computer Science, Cornell University

## Scenario with varying stage times

---



- Latency: ?
- Throughput: Batch every 45 minutes

© Kavita Bala, Computer Science, Cornell University

## Pipelining

---

- Principle: Latencies can be masked by running operations in parallel
- Need to identify “stages”
- Need mechanisms for isolating the operations
- Need mechanisms for handling dependencies between stages
- Let’s apply this principle to processor design...

© Kavita Bala, Computer Science, Cornell University