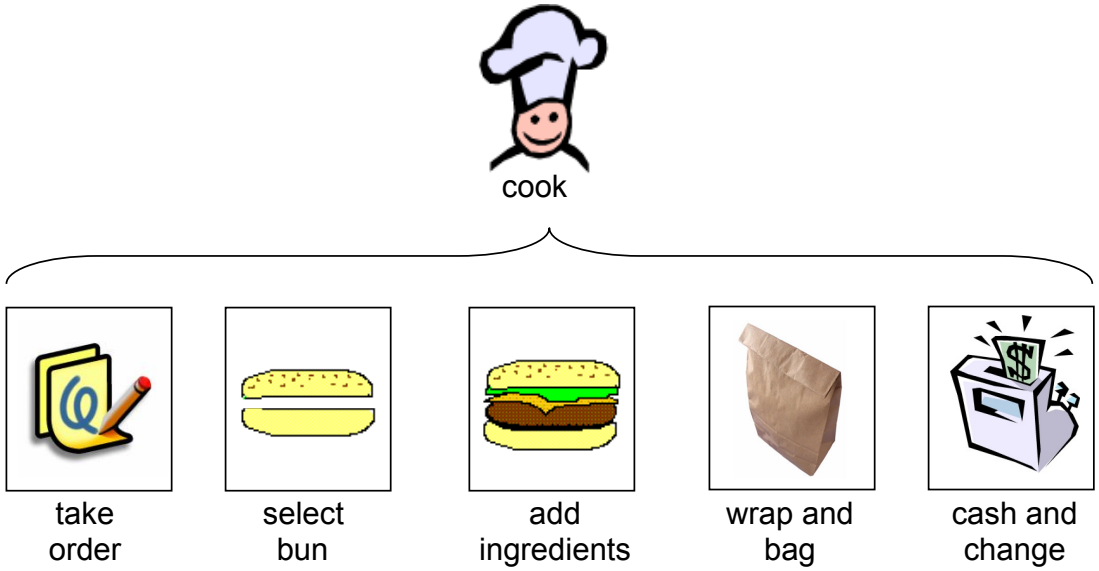
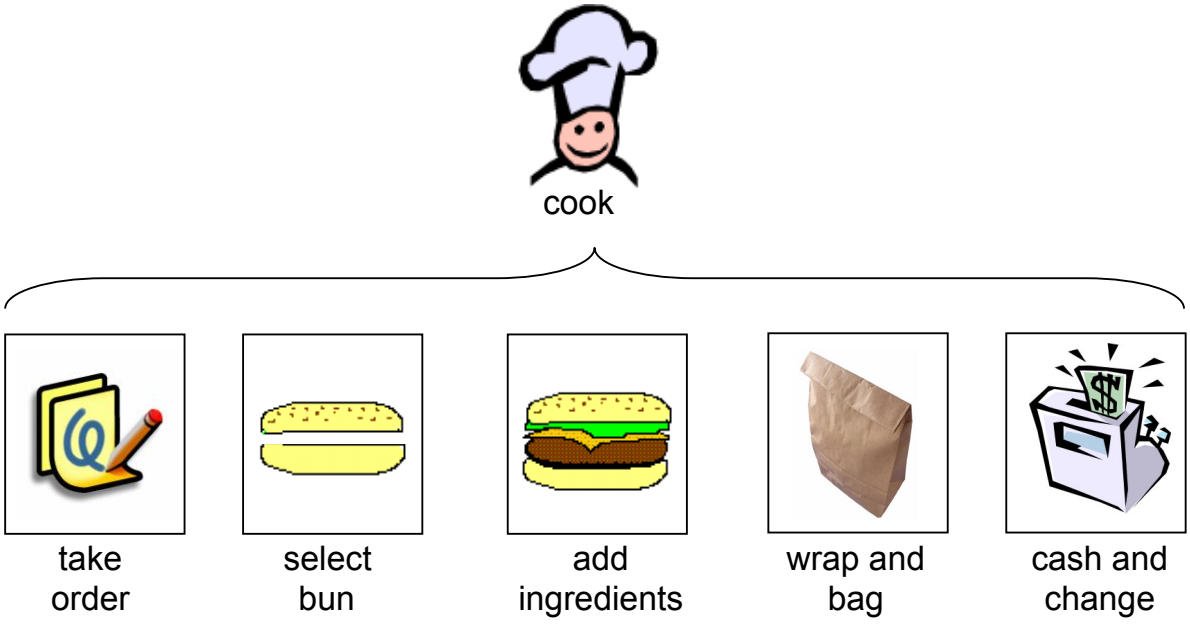
ECE243

CPU: Pipelining

A Fast-Food Sandwich Shop

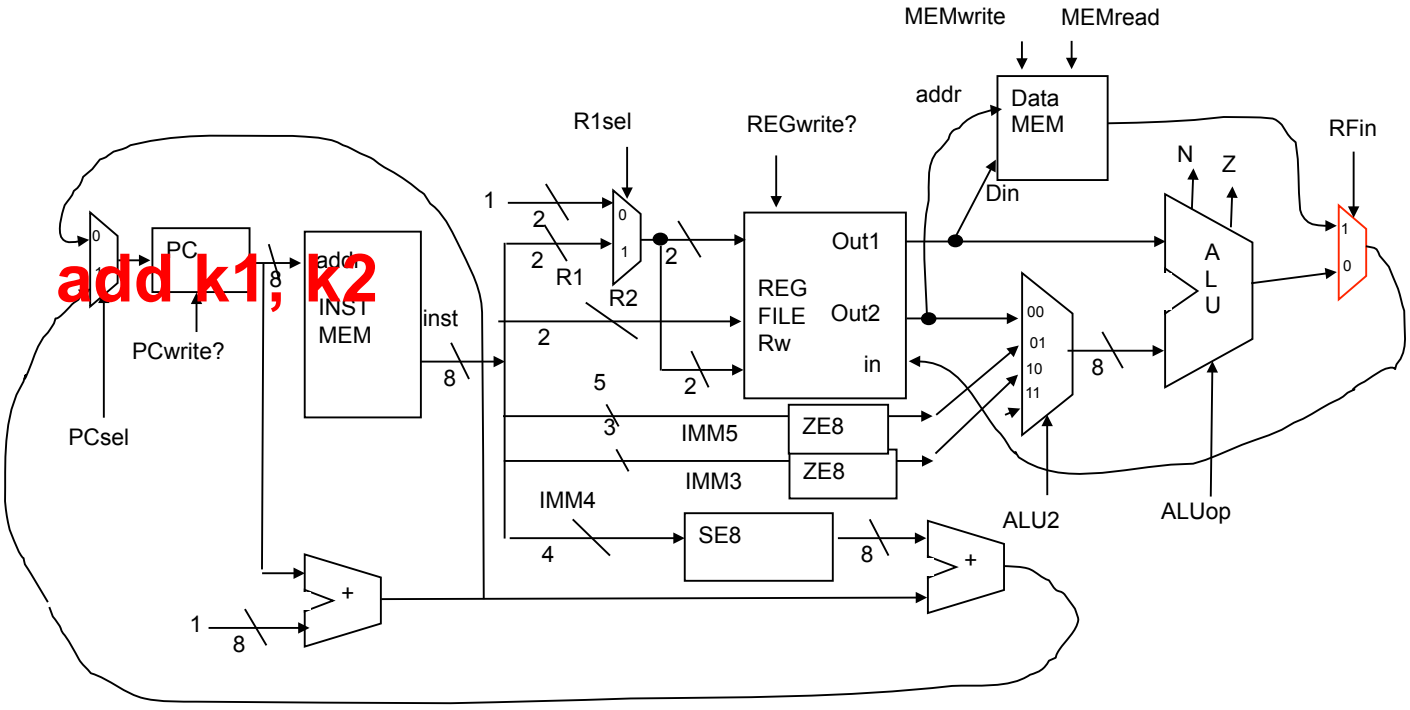


With One Cook



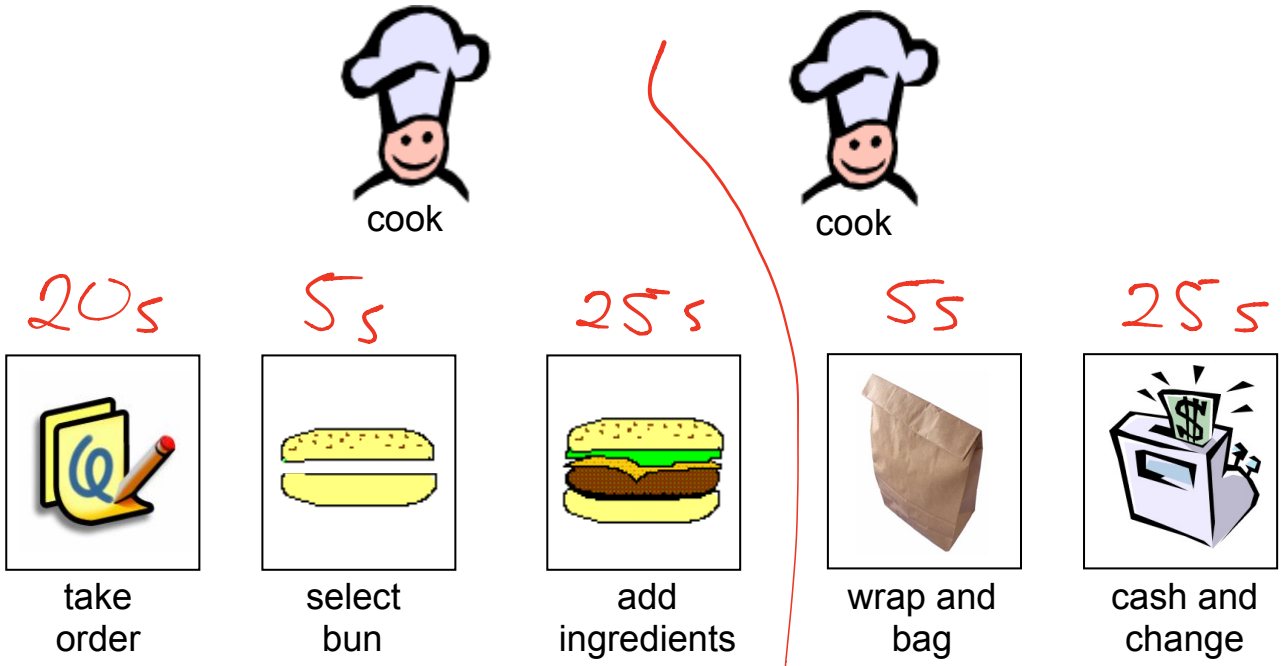
- one customer is serviced at a time

Like the single-cycle CPU



- one instruction flows through at a time

With Two Cooks?



Break up work so tasks are even

cook 1: 50s total

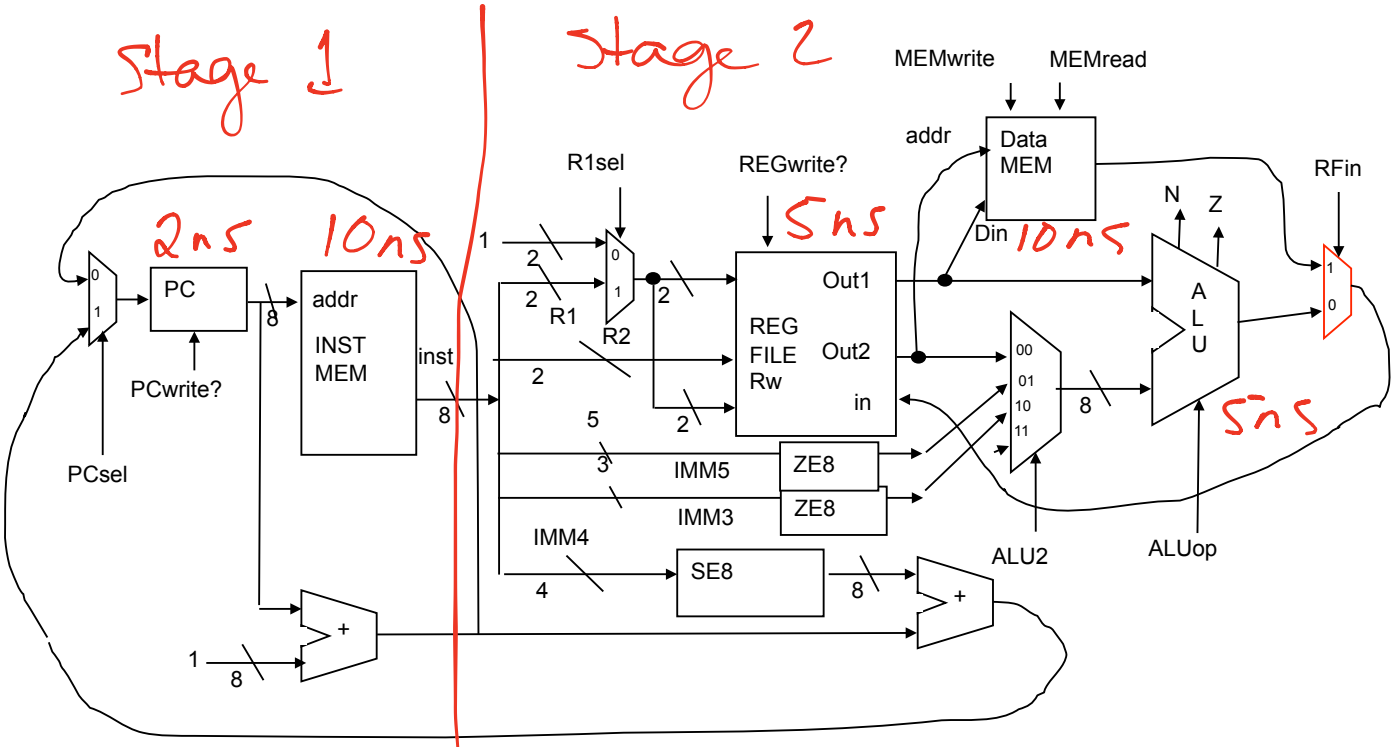
cook 2: 30s total

can service 1 new customer every 50s, 2 customers serviced @ a time

Pipelining

- Like an assembly line
- Doesn't change the interface or result
  - improves performance

Pipelining a CPU (rough idea)



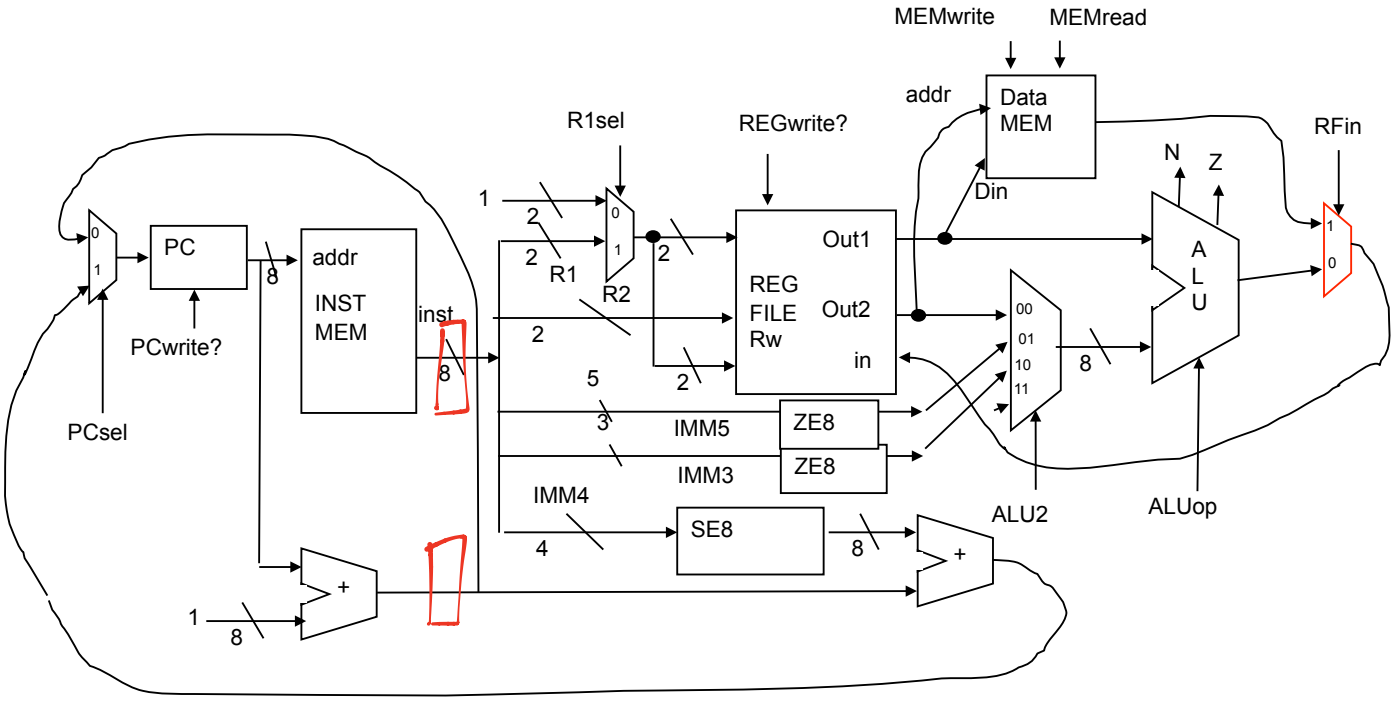
Break into 2 stages

stage 1: 12ns

stage 2: 15ns (instr uses either data mem or ALU)

2 instrs at once

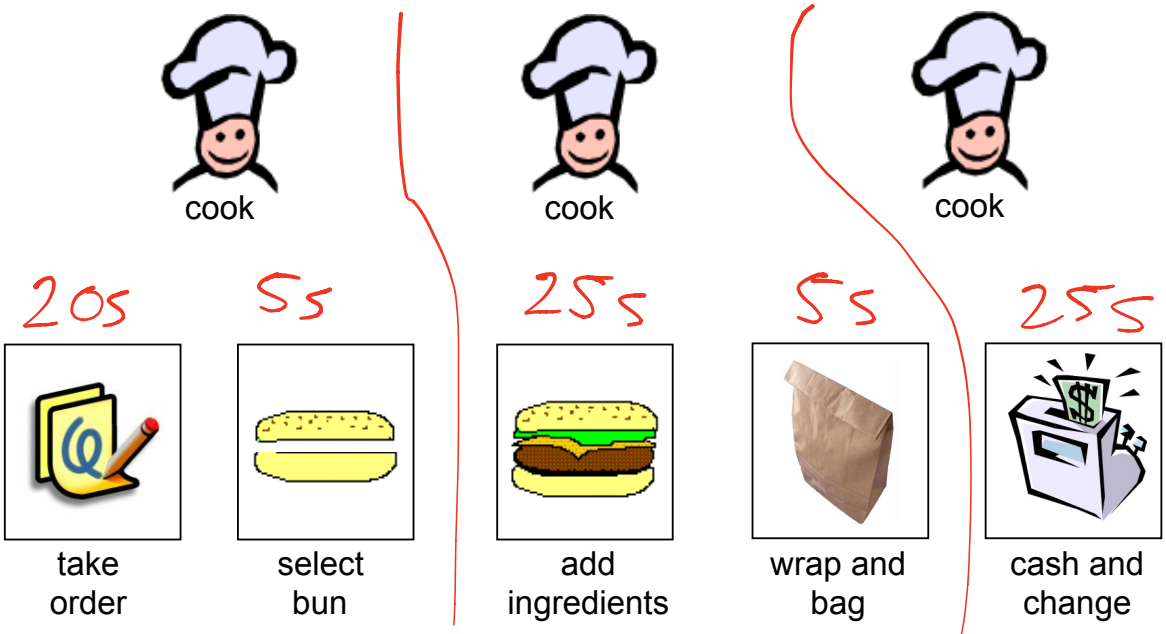
Pipelining Details:



Insert pipe stage registers to store intermediate values

modify control logic to support pipe stages

With Three Cooks?



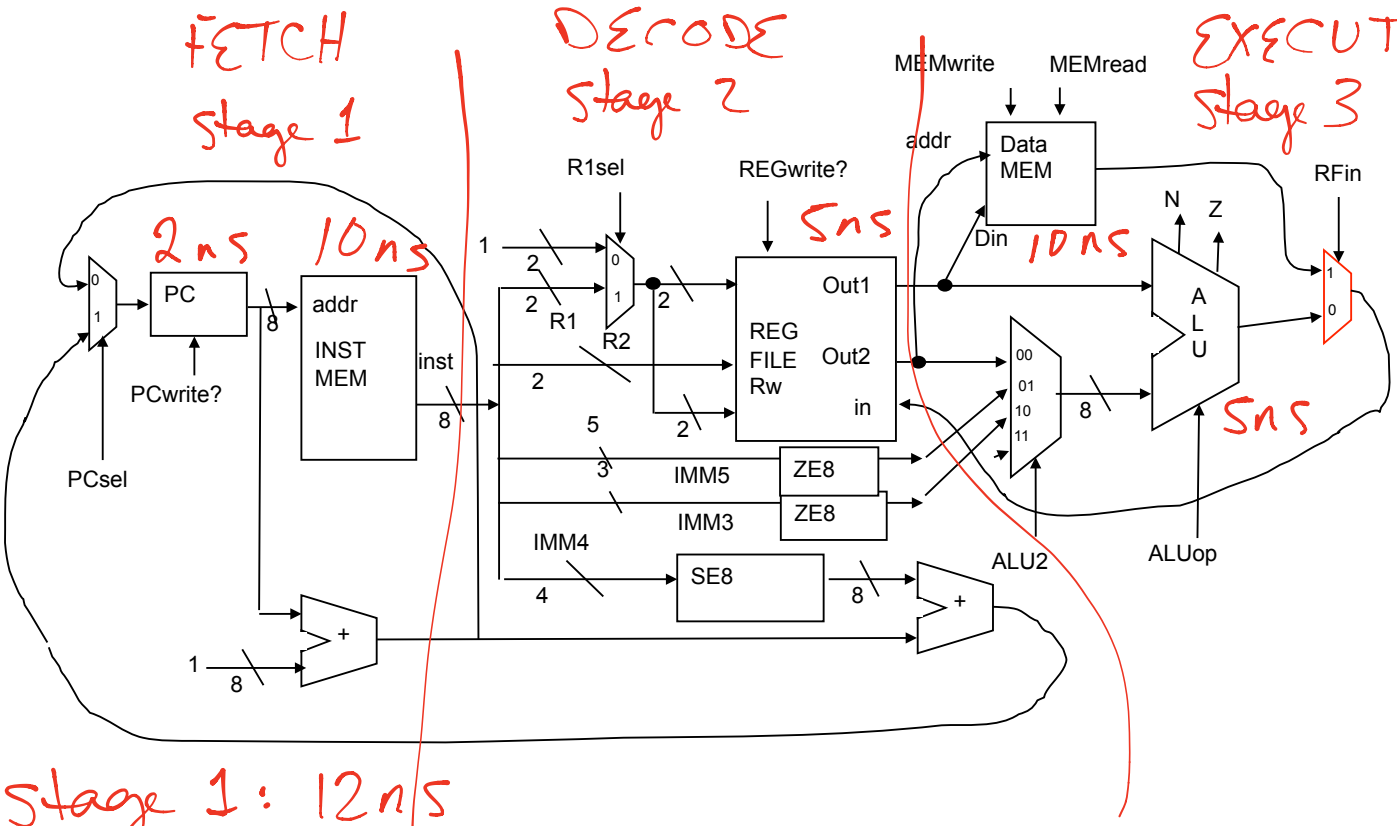
cook 1: 25s

cook 2: 30s

cook 3: 25s

Can service new customer every 30s

Pipelining a CPU (rough idea)



stage 1: 12ns

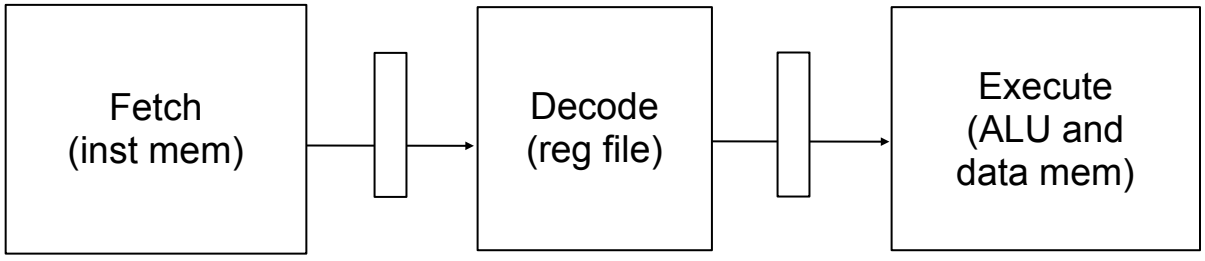
stage 2: 5ns

stage 3: 10ns

can start processing new instr every 12ns

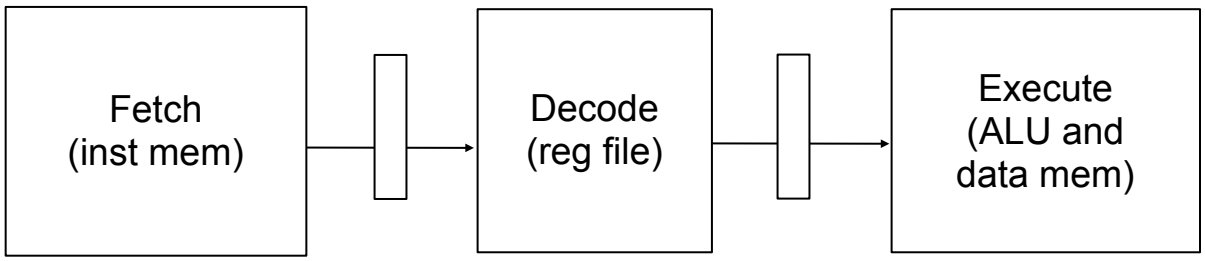
3 instr at once

Visualizing Pipelining



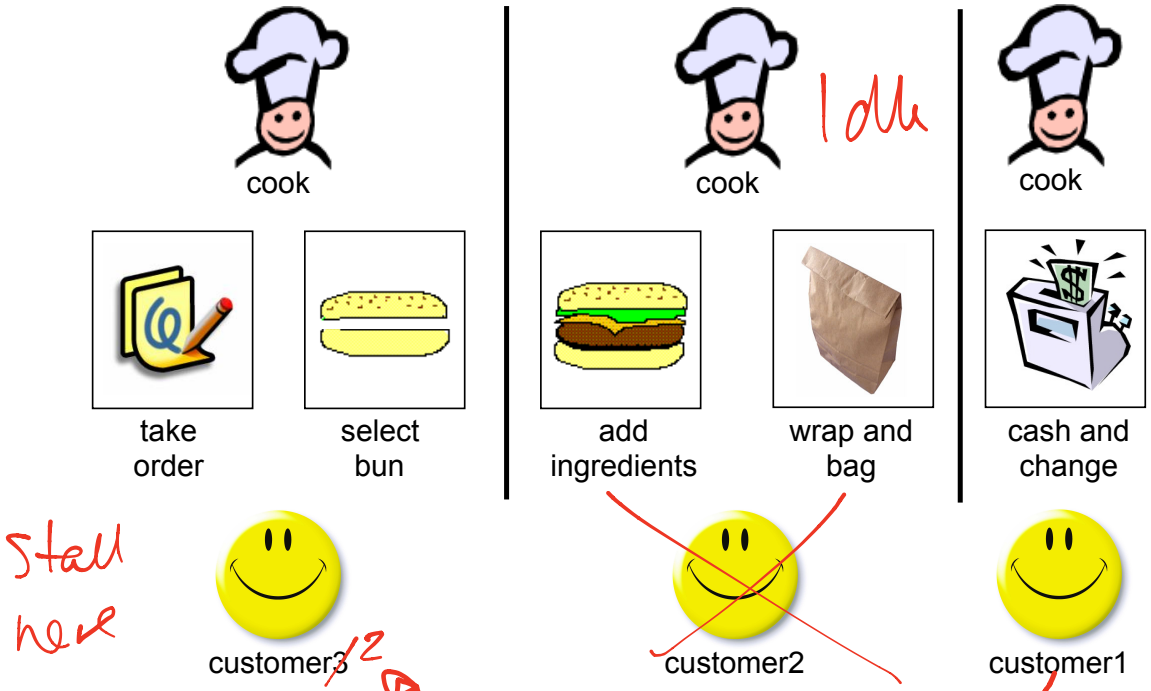
Cycle	Fetch	Decode	Execute
1	Instr 1		
2	Instr 2	Instr 1	
3	Instr 3	Instr 2	Instr 1
4	Instr 4	Instr 3	Instr 2

Visualizing Pipelining (again)



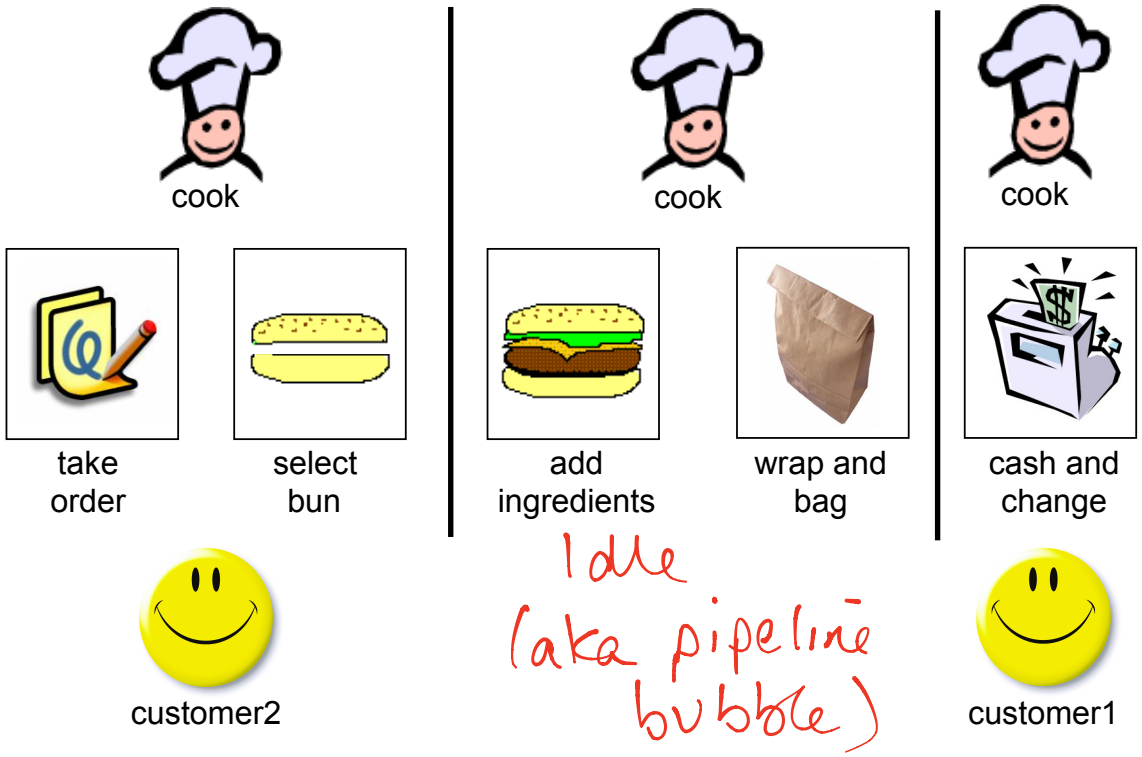
Cycle	1	2	3	4	5
inst1	F	D	E		
inst2		F	D	E	
inst3			F	D	E
inst4				F	D

Fast Food Hazards



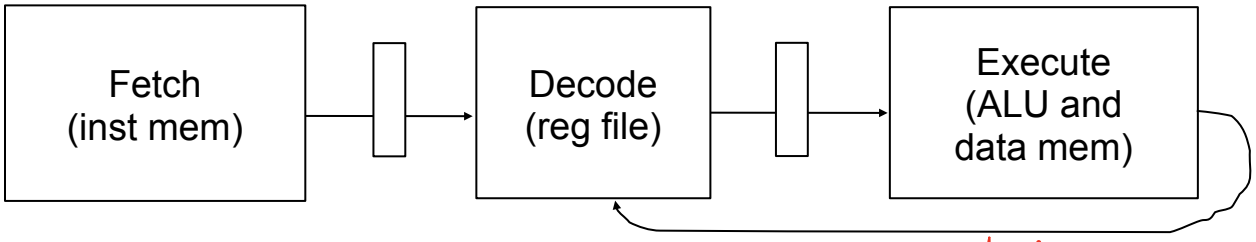
What if: c1 and c2 are friends, c2 has no money, and c2 needs to know how much change c1 will get before ordering (to ensure c2 can afford his order)?

Fast Food Hazards



C2 will have to wait (take stall in 1st stage until C1 can tell him how much change he got

CPU Hazards

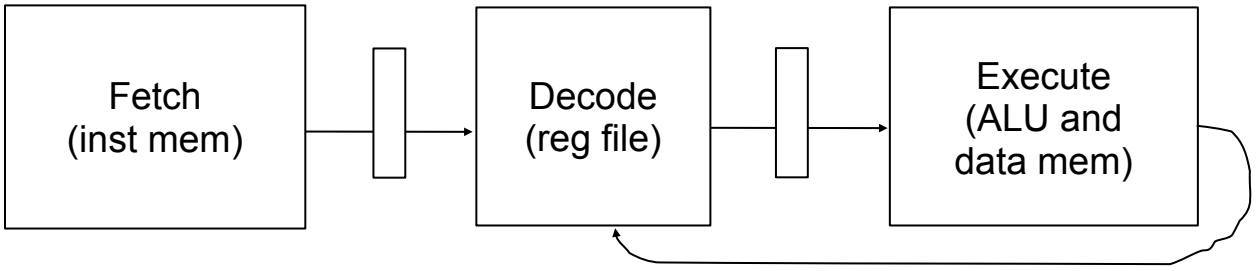


add k3, k0 add k1, k2 ok

add k3, k1 add k1, k2 X

- called a data hazard
- must be observed to ensure correct execution
- there are two solutions to data hazards

Solution1: Stalling



Cycle	1	2	3	4	5
add k1,k2	F	D	E		
add k3,k1		F	F (stall)	D	E
sub k0,k2				F	D
add k2,k2					F

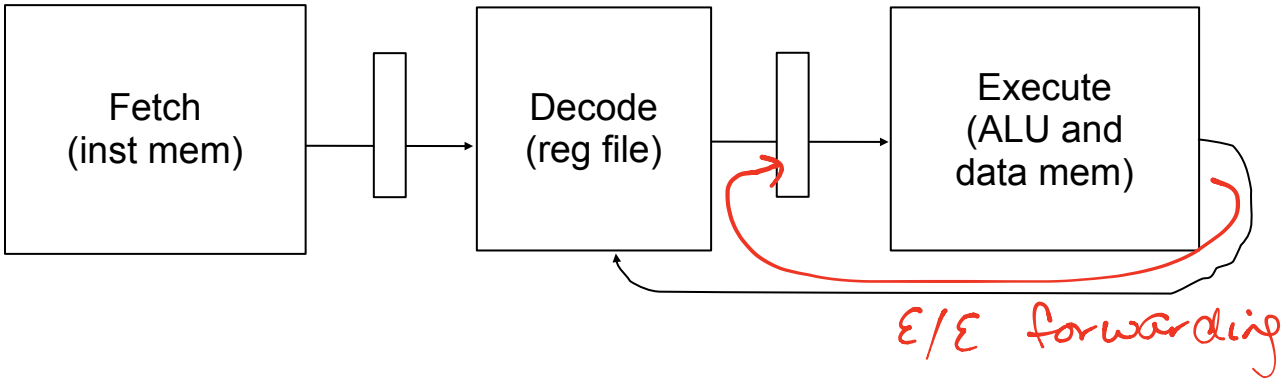
observe hazard by stalling
insert bubble into pipeline

How to insert bubbles

- option1: hardware stalls the pipeline
  - need extra logic to do so
  - happens 'automatically' for any code
- option2: compiler inserts "no-ops"
  - a no-op is an instruction that does nothing
  - ex: add r0,r0,r0 (NIOS)
  - compiler must do it right or wrong results!
- example: inserting a bubble with a no-op:

```
add k1, k2
nop
add k3, k1
```

Solution2: Forwarding Lines



- add “forwarding” logic
  - to pass values directly between stages

Cycle	1	2	3	4	5
add k1,k2	F	D	E		
add k3,k1		F	D	E	
sub k0,k2			F	D	E
add k2,k2				F	D

Control Hazards

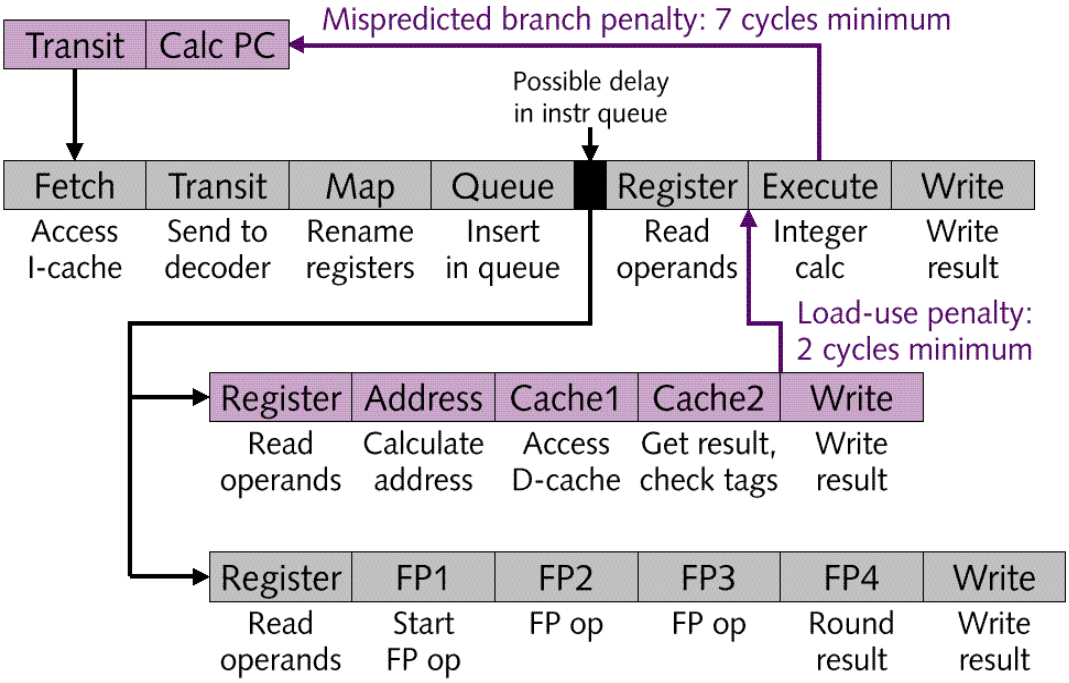
Cycle	1	2	3	4	5	6	7
add k1,k2	F	D	E		F	D	E
bnz -2		F	D	E		F	D
add k3,k1			F	DX			F
add k2,k2				FX			

- cpu predicts each branch is not taken
- Better: predict taken
  - why?---loops are common, usually taken
- More advanced: remember what each branch did last time

- "branch predictor":
  - a table that remembers what each branch did the last time
  - uses this to make a prediction next time

Some Real CPU Pipelines

21264 Pipeline (Alpha)



Microprocessor Report 10/28/96

Pentium IV's Pipeline: