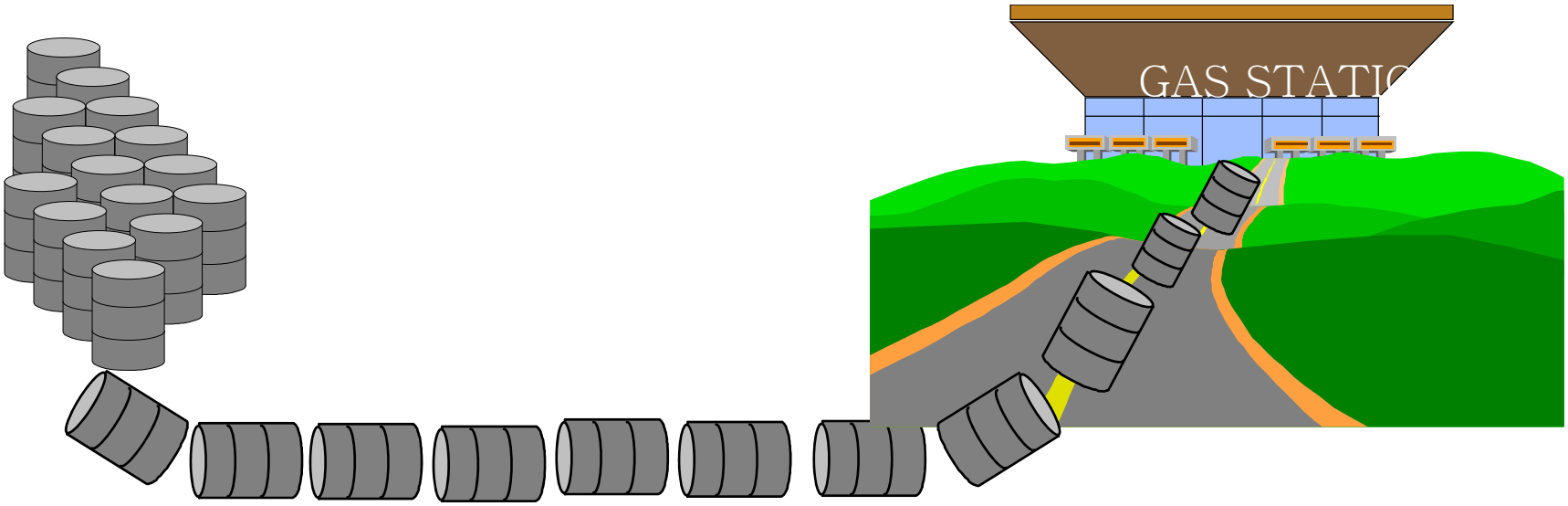
Pipelined Processor Design

- Hazards
  - ¿ Data
  - ¿ Resource/Structural
  - ¿ Control



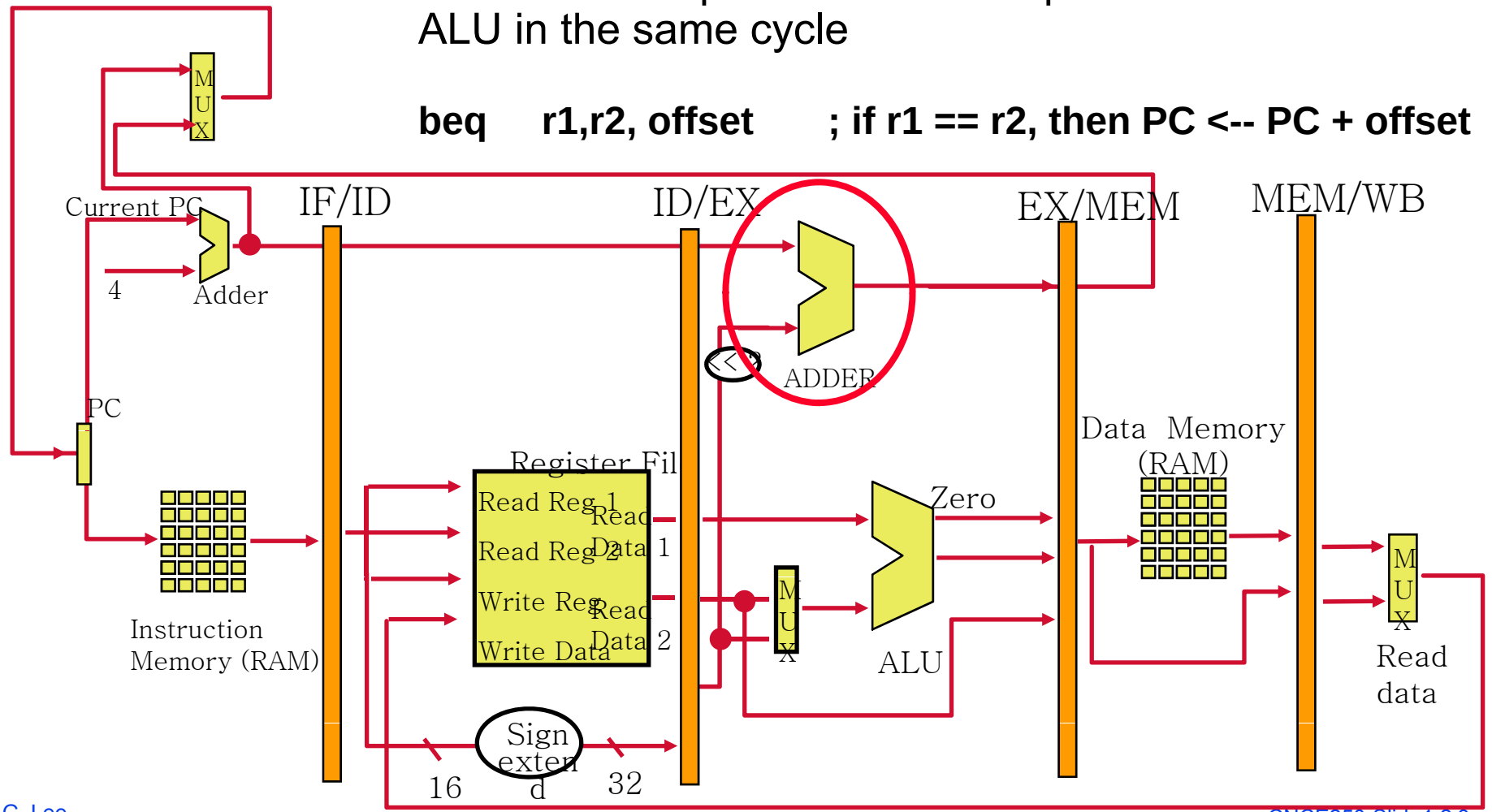
Hazards

- Recall Data Hazards are just one type of hazard that can occur in a machine.
- Hazards (Dependency)
  - ¿ Data hazards
    - Ø Instruction depends on result of prior computation which is not ready yet
  - ¿ Structural hazards
    - ⁄ HW cannot support a combination of instructions
  - ¿ Control hazards
    - Ø pipelining of branches and other instructions which change the PC

Structural Hazard Example 1

- W/out adder, both the address computation and the arithmetic computation would require access to the ALU in the same cycle

beq r1,r2, offset ; if r1 == r2, then PC <-- PC + offset



Control Hazards - Branches

Example code

Address	Instruction	
36	NOP	
40	ADD R30,R30,R30	
44	BEQ R1, R3, 24	<- this branch to address 72
48	AND R12, R2, R5	
52	OR R13, R6, R2	
56	ADD R14, R2, R2	
60	...	
64	...	
68	...	
72	LW R4, 50(R7)	
76	...	

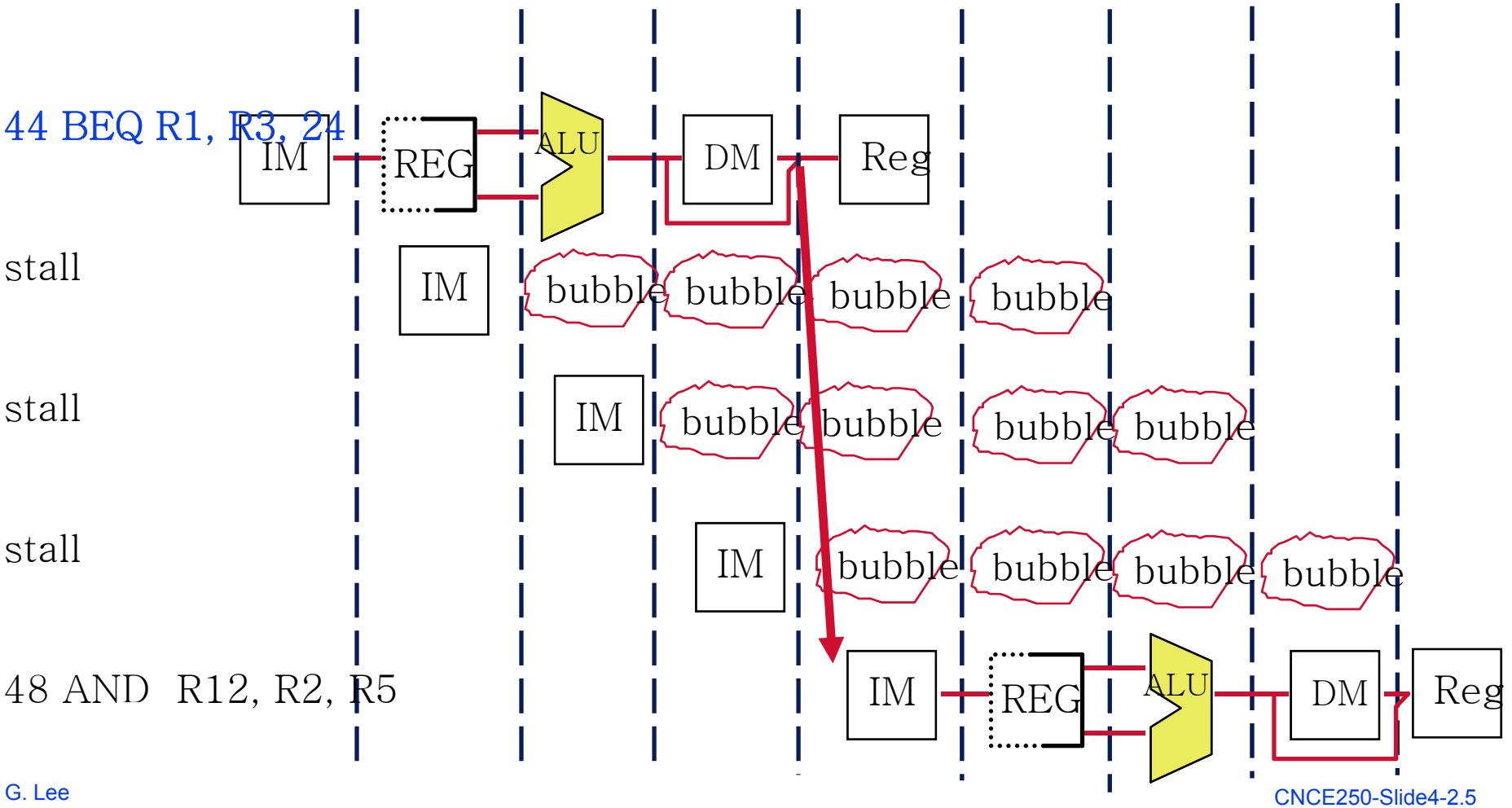
Flow of instructions if branch is taken: 36, 40, 44, 72, ...

Flow of instructions if branch is not taken: 36, 40, 44, 48, ...

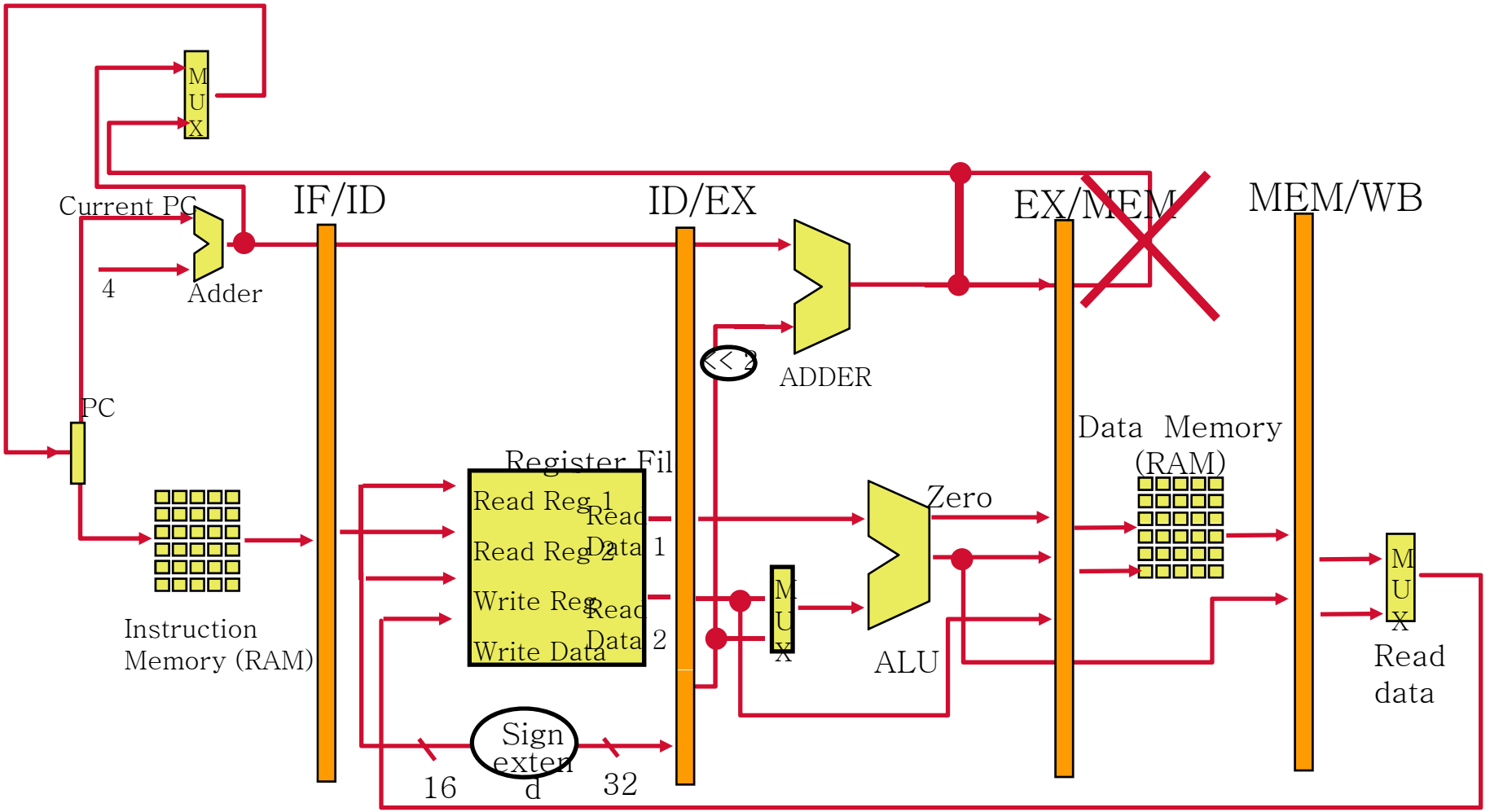
Always Stalling

Two possible improvement:
1. Branch prediction
   assume one way and pull back if incorrect
2. Reduce the no. of stalls
   by calculating branch condition as early as possible

Flow of instructions if

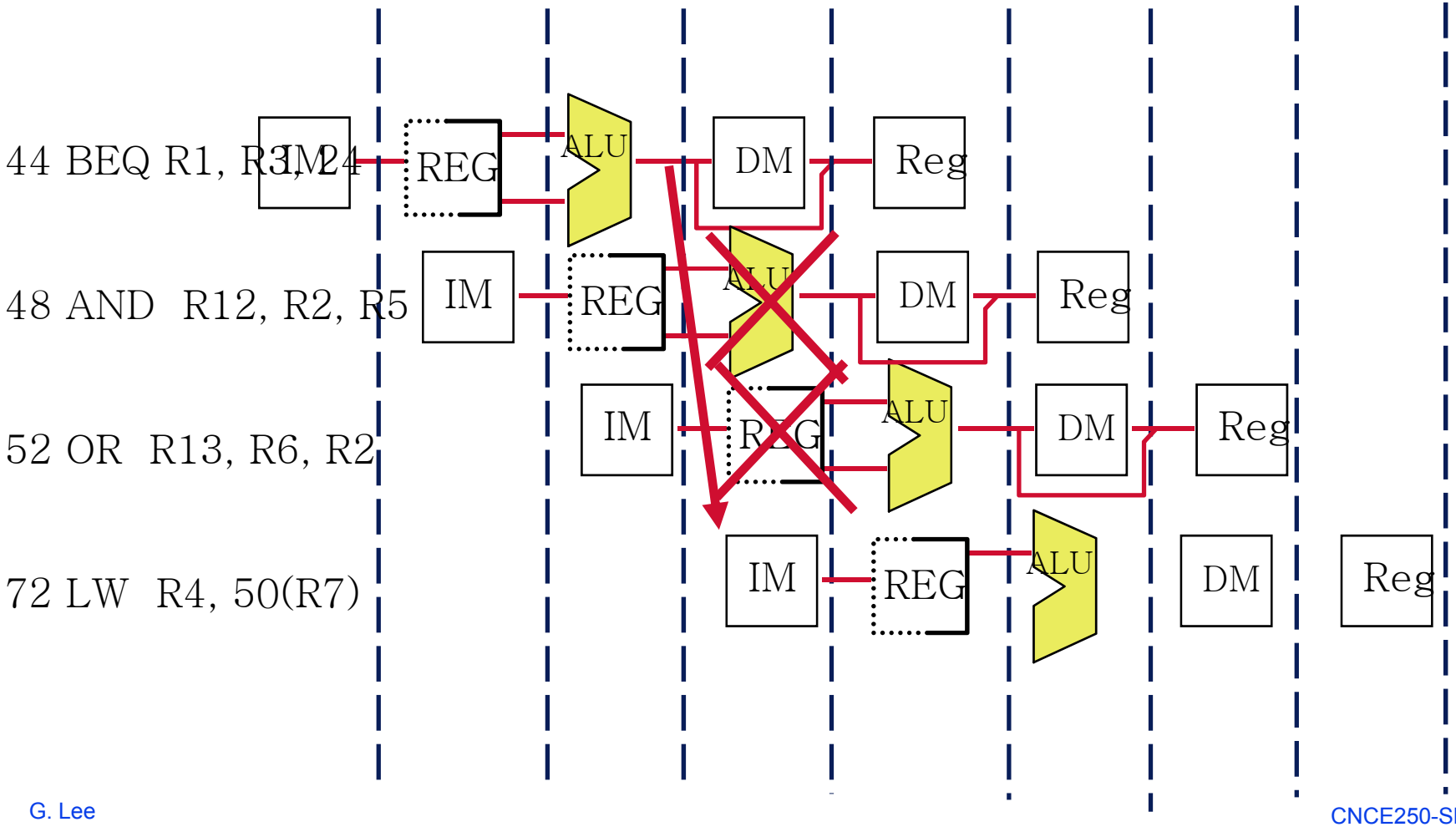


Move the Branch Computation Forward

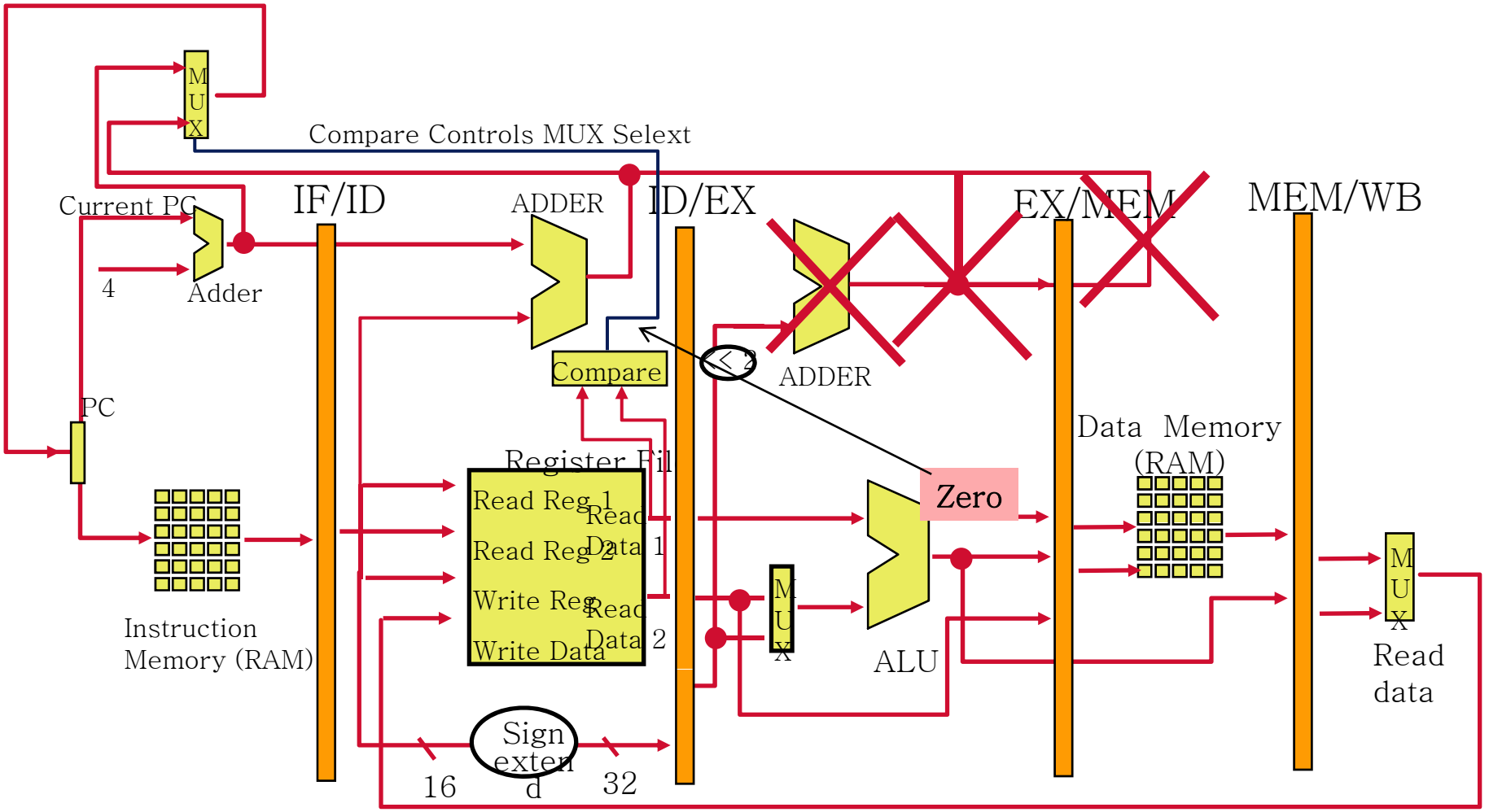


Branch with New Datapath

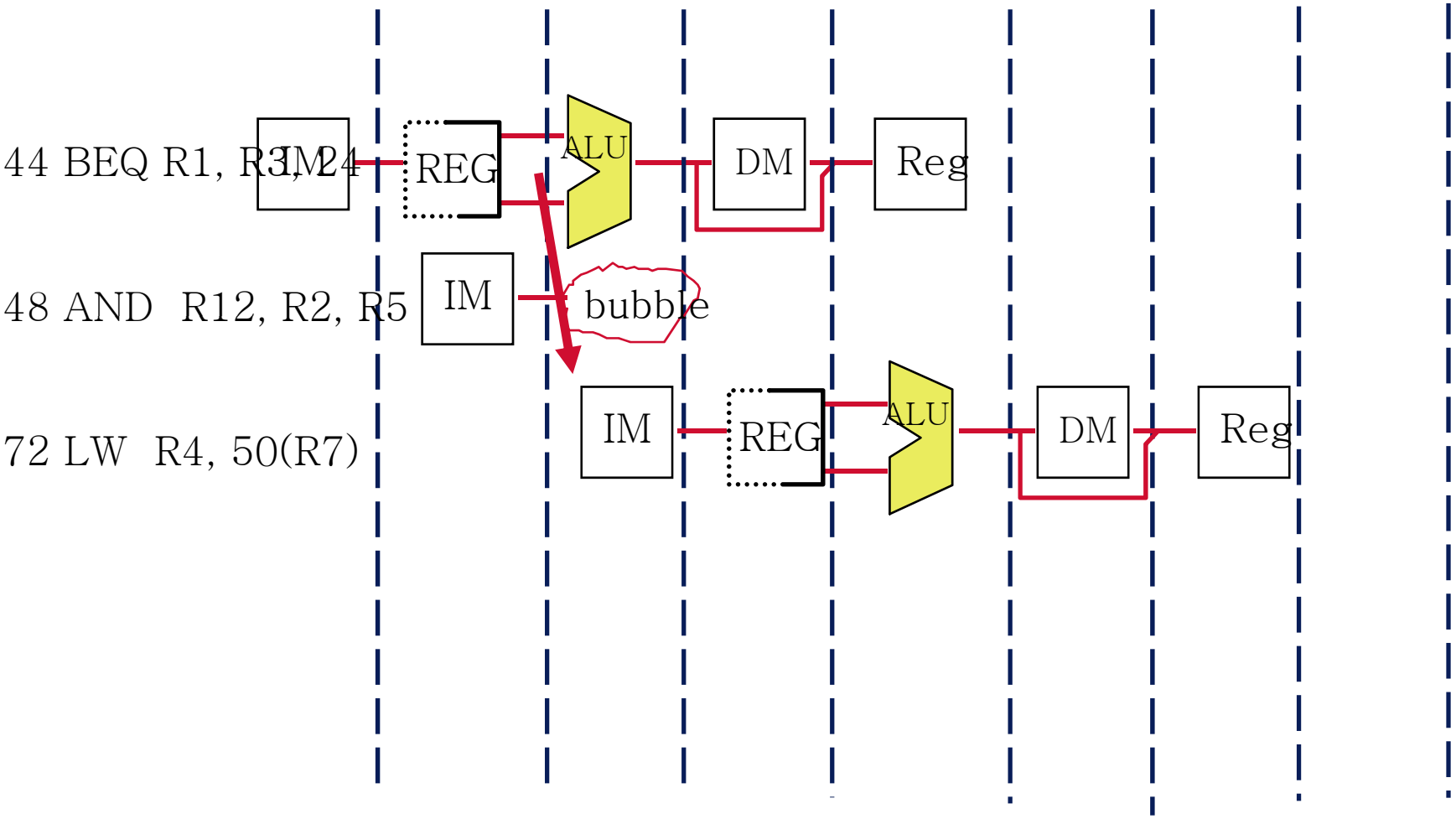
- Reducing penalty 1 cycle



Move the Branch Computation Further Forward

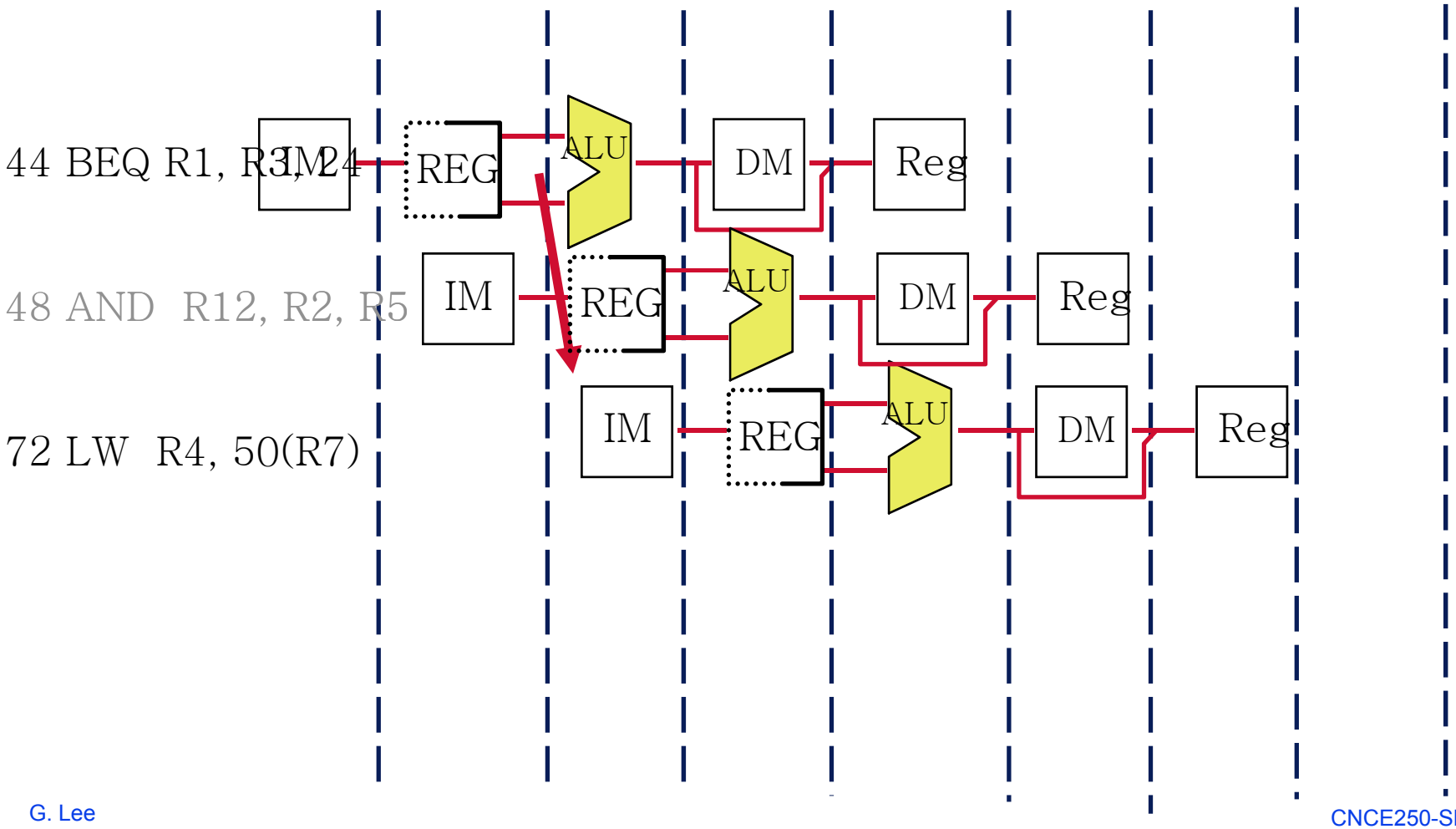


Another New and Improved Datapath



Branch Delay Slot

Assuming always one next to the branch is something unrelated



Rewriting the Code for a Branch Delay Slot

- **Without Branch Delay Slot**

Address	Instruction
36	NOP
40	ADD R30,R30,R30
44	BEQ R1, R3, 24
48	AND R12, R2, R5
52	OR R13, R6, R2
56	ADD R14, R2, R2
60	...
64	...
68	...
72	LW R4, 50(R7)
76	...

With Branch Delay Slot

Address	Instruction
36	NOP
40	BEQ R1, R3, 28
44	ADD R30, R30, R30
48	AND R12, R2, R5
52	OR R13, R6, R2
56	ADD R14, R2, R2
60	...
64	...
68	...
72	LW R4, 50(R7)
76	...

- Flow of instructions if branch is taken: 36, 40, 44, 72, ...
- Flow of instructions if branch is not taken: 36, 40, 44, 48, ...

Performance of Pipelined Systems

- Stalls due to data and branch hazards make performance less than one instruction per cycle
- Compiler is critical in determining overall performance
  - ¿ Compiler generates code that avoids stalls
- Example

```
lw   R15, 0x00(R2)
add  R14, R15, R15
lw   R16, 0x04(R2)
```

  - ¿ Might become:

```
lw   R15, 0x00(R2)
lw   R16, 0x04(R2)
add  R14, R15, R15
```

Pipeline Speedup and Throughput

- Assume instruction execution takes n stages
  - ¿ s_1, s_2, ... s_n with each stage per cycle t_c
- Without pipelining (assuming still n stages)
  - ¿ Throughput = $1/(n*t_c)$ (for to n stage)
  - ¿ Latency = 1/throughput = $n*t_c$
- With pipelining
  - ¿ Throughput $n/ n*t_c = 1/ t_c$
  - ¿ Latency = n/throughput = t_c
  - ¿ Speedup = $n*t_c / t_c <= n$

Exceptions and Interrupts

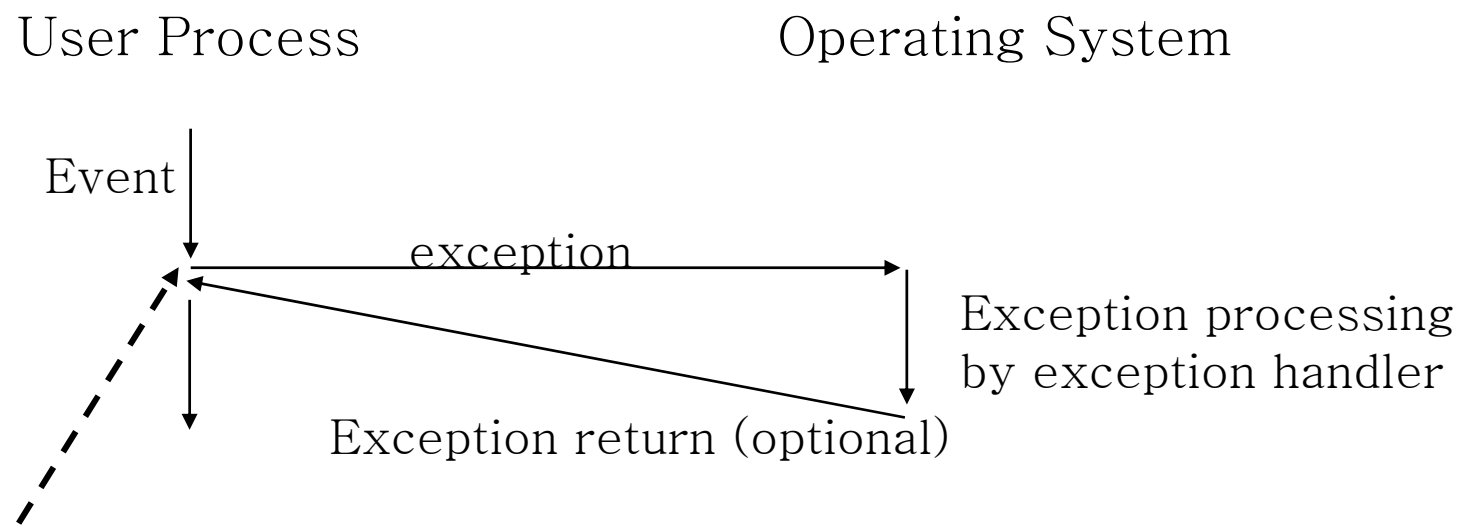
- Exceptions are exceptional events that disrupt the normal flow of a program
- Terminology varies between different machines
- Examples of Interrupts
  - ¿ User hitting the keyboard
  - ¿ Disk drive asking for attention
  - ¿ Arrival of a network packet
- Examples of Exceptions
  - ¿ Memory error
  - ¿ Overflow
  - ¿ Page fault

Handling Exceptions and Interrupts

- When do we jump to an exception
- Upon detection, invoke the OS to service the event
  - ¿ Right when it occurs?
  - ¿ What about in the middle of executing a multi-cycle instruction
    - Ø Difficult to abort the middle of an instruction
  - ¿ Processor checks for event at the end of every instruction
  - ¿ Processor provides EPC and Cause registers to help OS determine cause of event
    - Ø EPC - Exception Program Counter
      - v Holds PC that the OS should jump to when resuming execution
    - Ø Cause Register
      - v Holds bit-encoded cause of the exception

Exception Flow

- When an exception (or interrupt) occurs, control is transferred to the OS

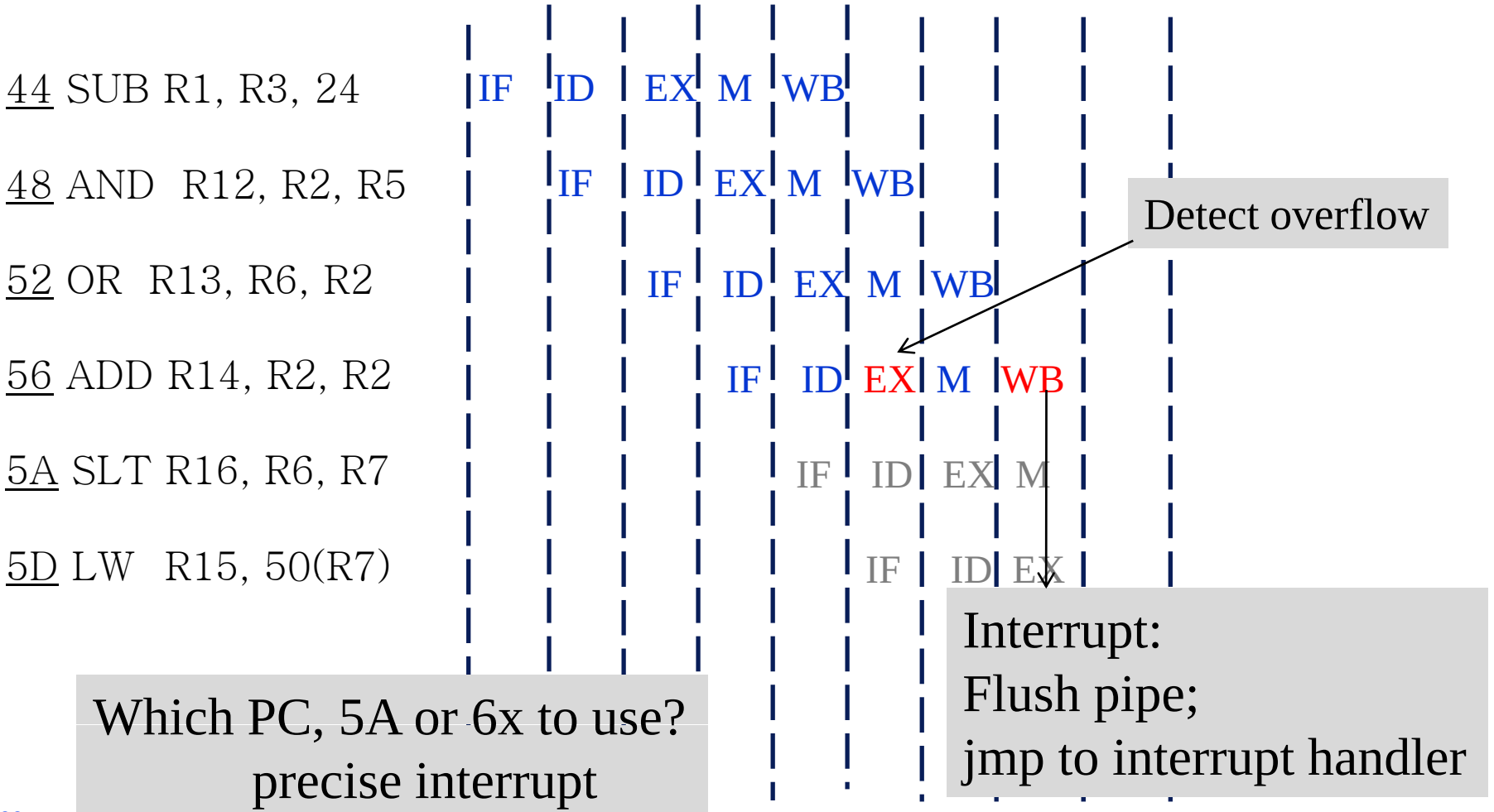


Flush out instructions fetched after exception by making them as nops

g1

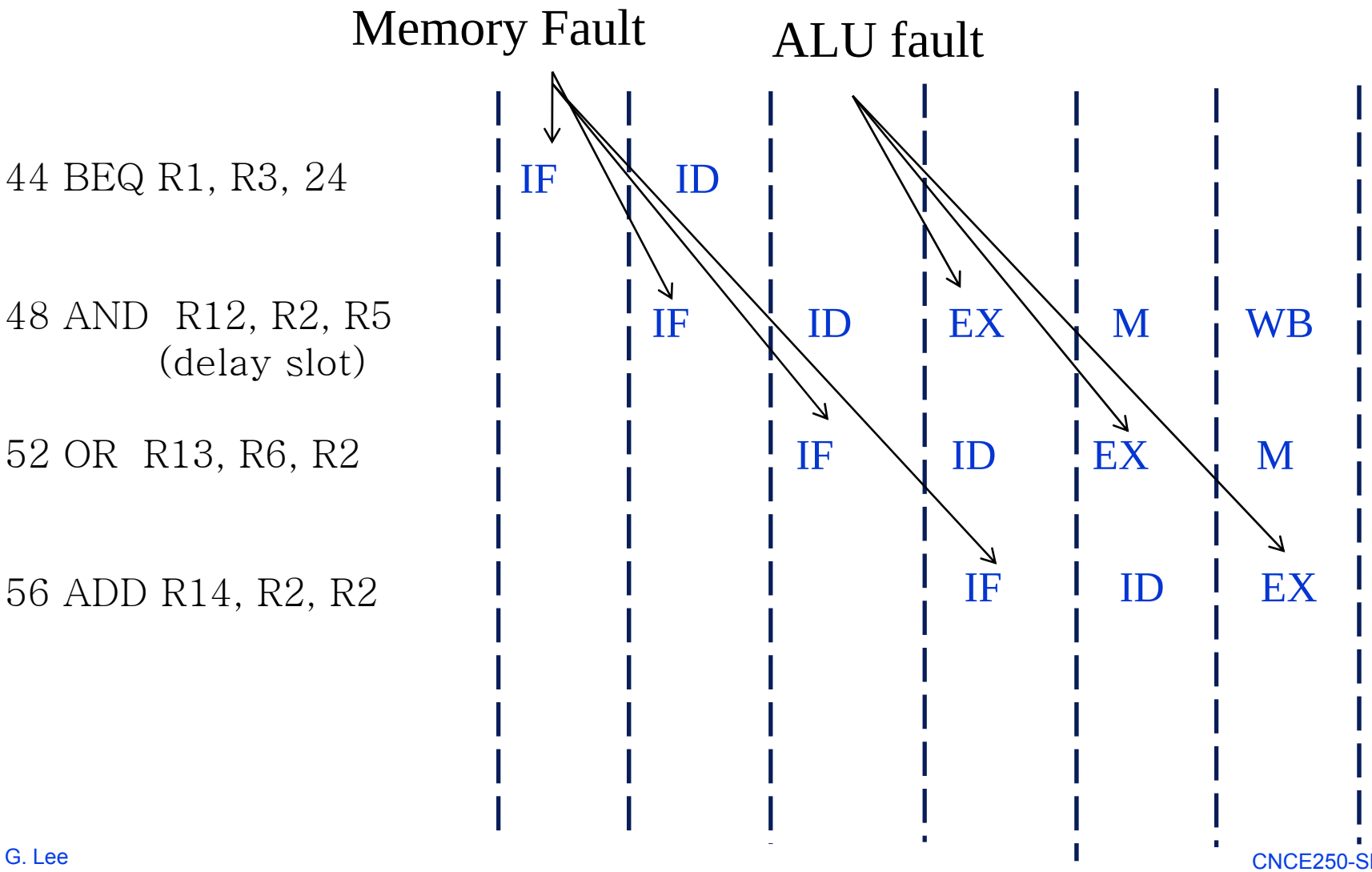
Flushing Pipe for exception

Assume overflow at ADD – EX stage



g1 one important aspect of handling interrupt is not to allow any "write" to memory/registers after detecting interrupt.
Here the instruction order may be different from inturrupt order, which may make the same instruction run multiple times. To avoid this, interrupt handling is done at the end of instruction instead of right on interrupt.
ghlee, 2015-04-15

Out-of-Order


Memory Fault
ALU fault
44 BEQ R1, R3, 24
48 AND R12, R2, R5
(delay slot)
52 OR R13, R6, R2
56 ADD R14, R2, R2
IF
ID
EX
M
WB

Dynamic Branch Prediction

With single-issue pipe, dynamic branch prediction may be a novel scheme,

but an essential feature for multiple-issue pipes

Dynamic Prediction based on branch history

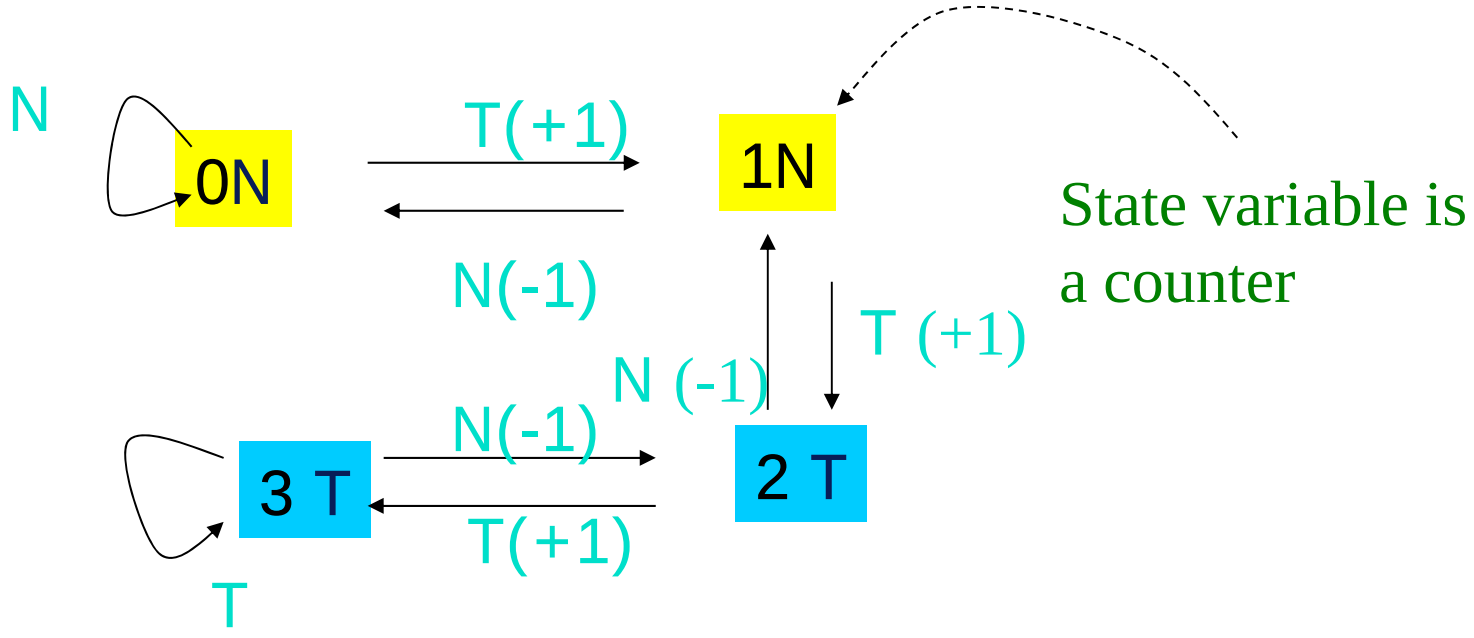
- Just looking at the history of the branch for prediction → prediction in isolation

- Looking at the history of other branches in addition to the branch for prediction

  → correlating prediction

Bi-modal (saturating counter) predictor

- Bi-modal predictor:
  - ¿ Only Two consecutive mis-predictions cause prediction change.
  - ¿ 2-bit “saturating” counter: state variable is a number



Instruction Level Parallelism

So far, the best CPI in a static in-order pipeline is **1**.

This is because every clock cycle, only one instruction would be inserted into the pipeline. And instructions get executed one by one in the program order.

Can we get a smaller CPI, for example, 0.5?

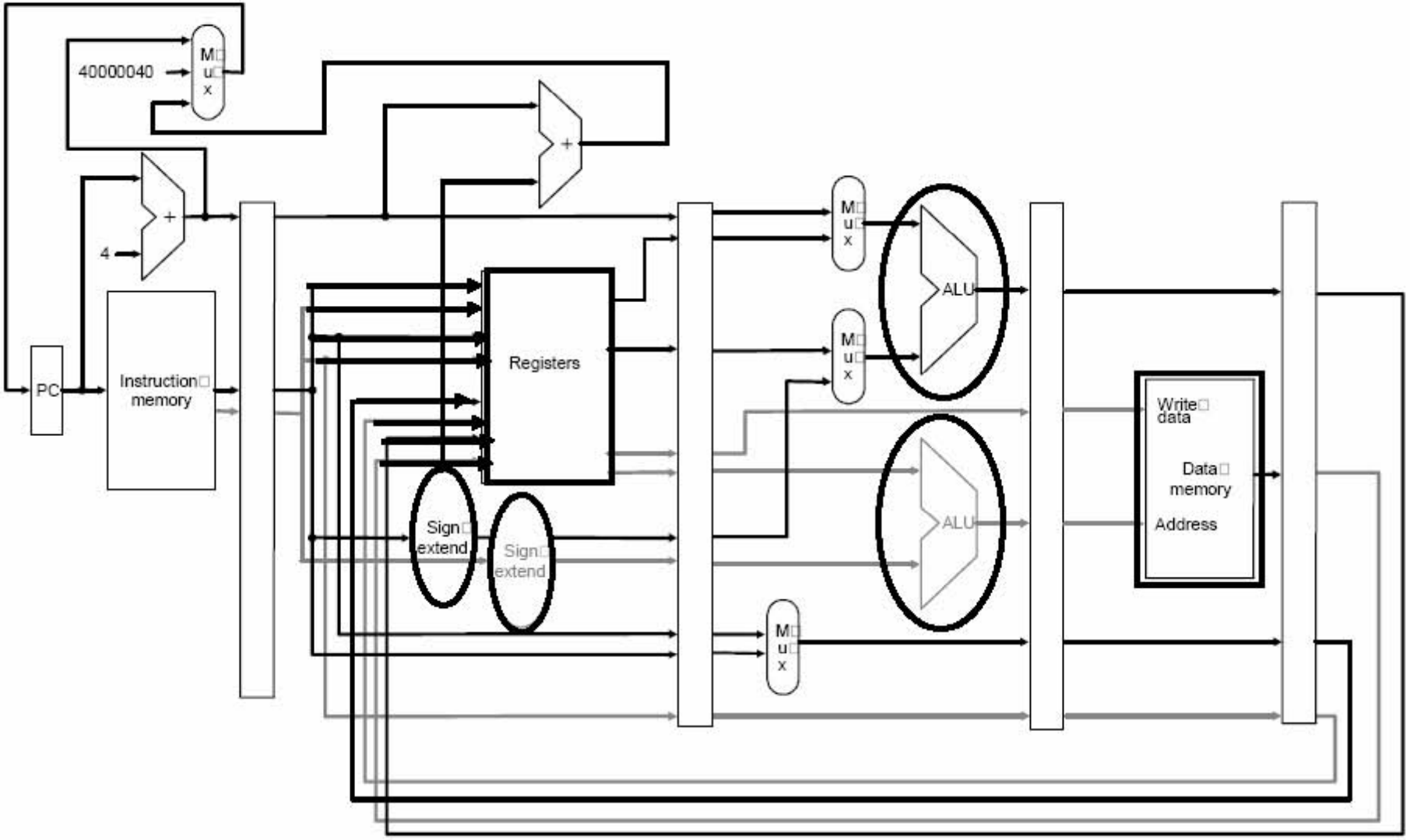
That means we fetch 2 instructions per cycle and complete them simultaneously.

```
lw      r4, 100(r5)
add     r2, r3 r4
sub     r5  r3 r6
sw      r7, 50(r8)
beq     r2, r5, loop
```

Parallelism exists for add and sub. Even the order of completion doesn't change the results.

1. complete them at the same time
2. complete them out of order

Superscalar



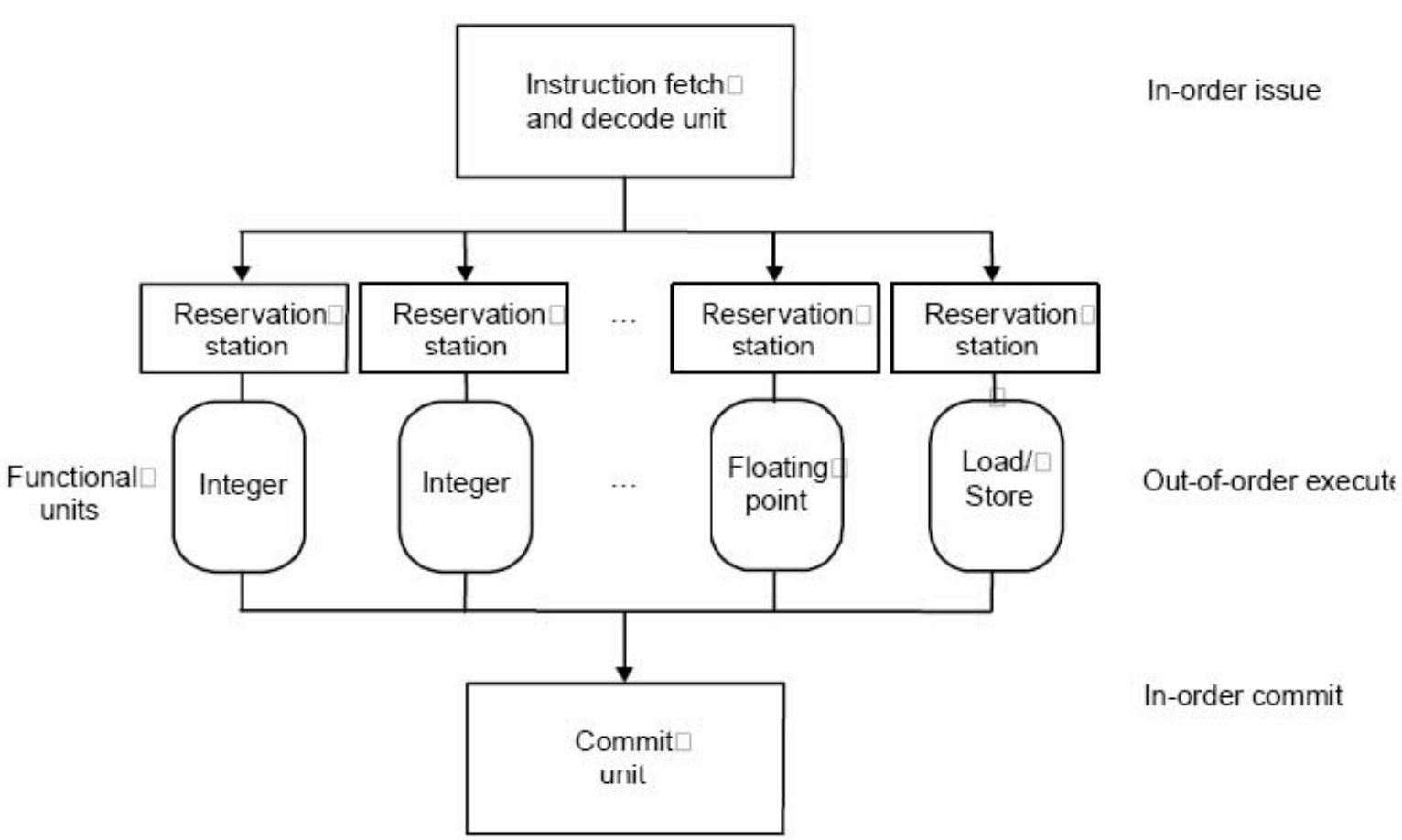
Dynamic Scheduling

- The hardware performs the “scheduling”
  - ¿ hardware tries to find instructions to execute
  - ¿ out of order execution is possible
  - ¿ speculative execution and dynamic branch prediction
- All modern processors are very complicated
  - ¿ DEC Alpha 21264: 9 stage pipeline, 6 instruction issue
  - ¿ PowerPC and Pentium: branch history table
  - ¿ Compiler technology important

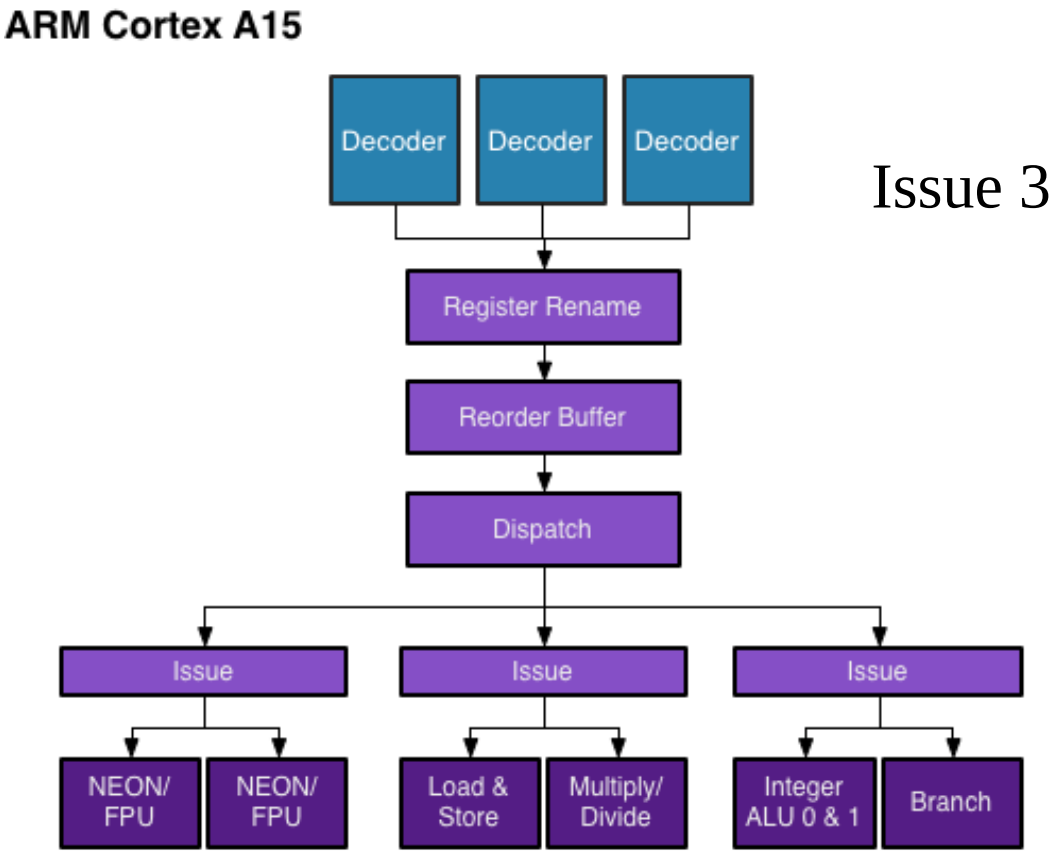
finding more than one instructions (ILP)

- This class has given you the background you need to learn more in graduate classes

A Modern Pipelined processor

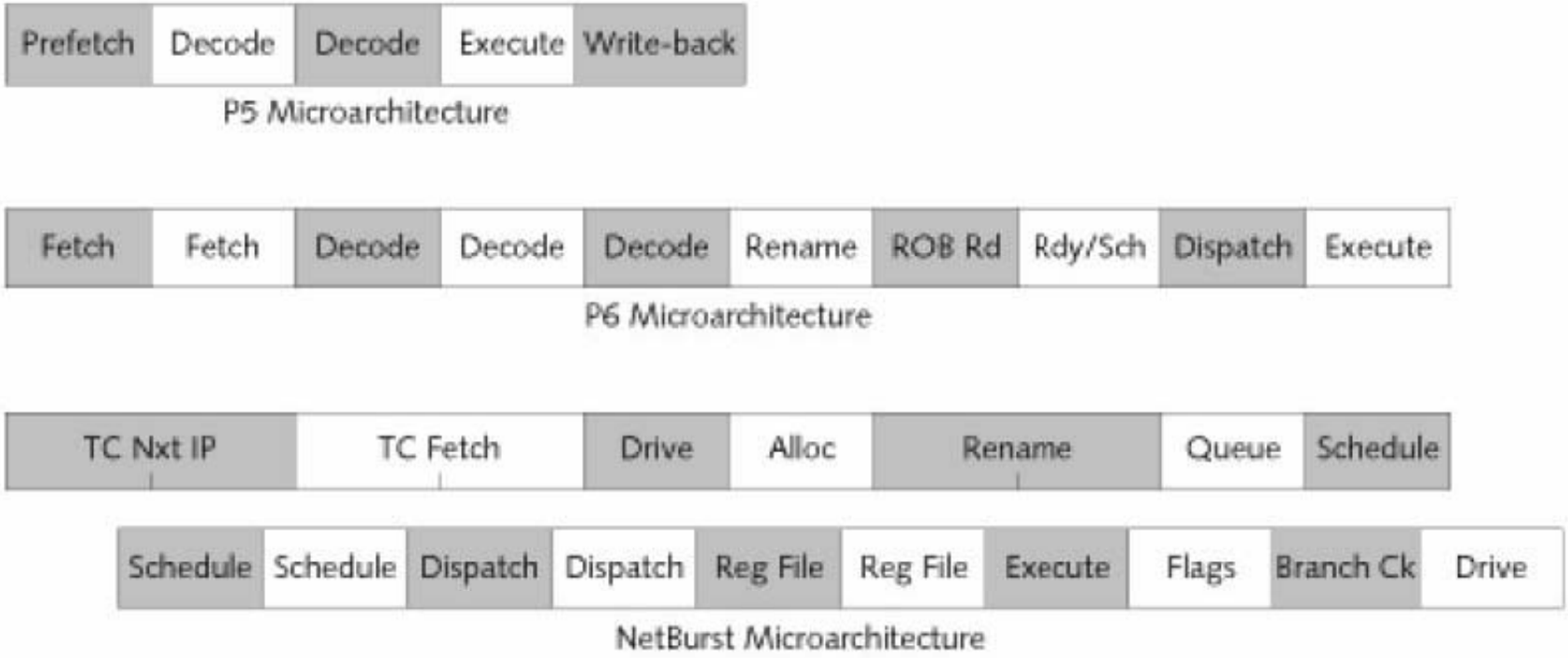


Real Stuff – ARM Cortex A15



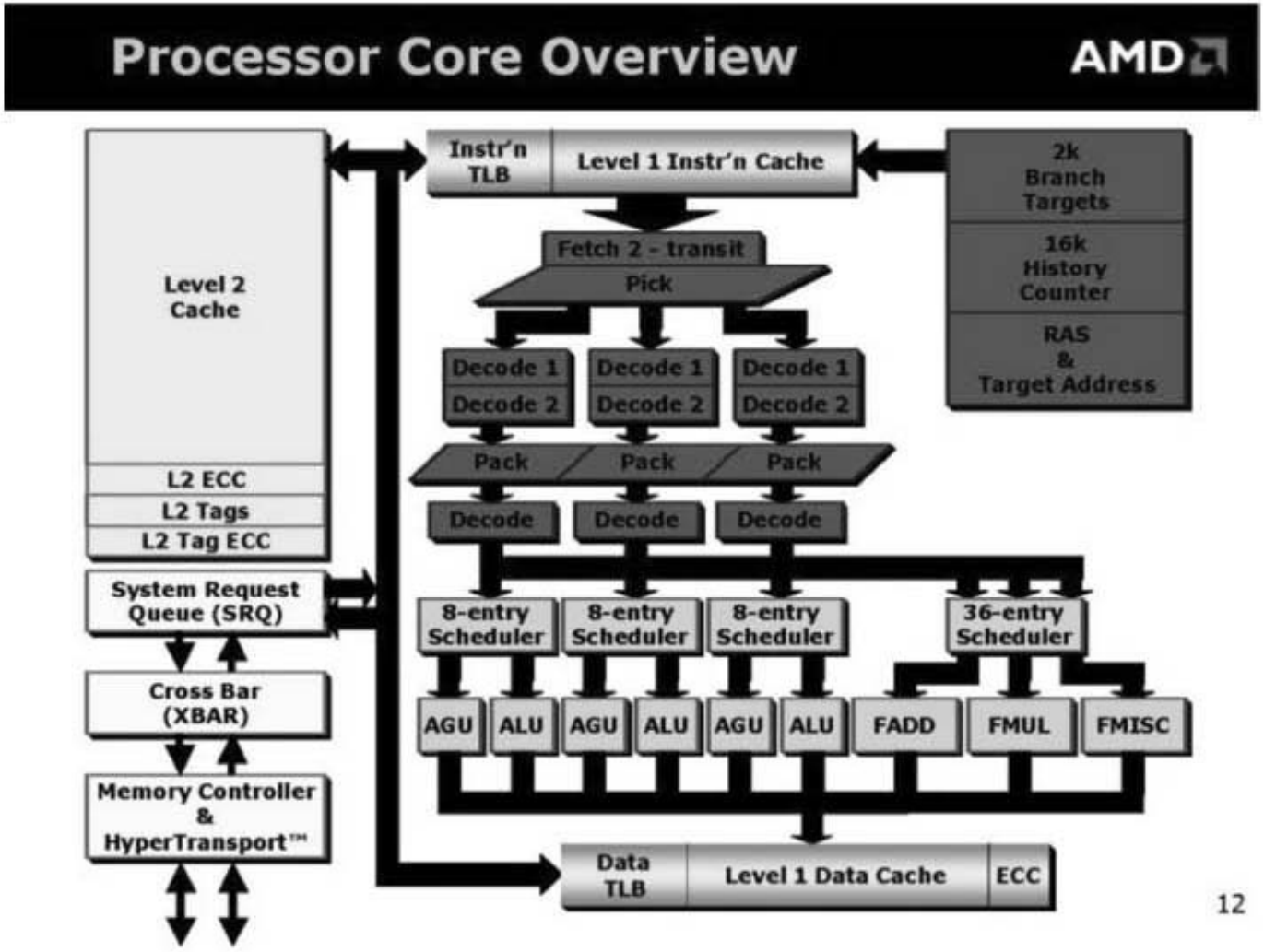
Issue 3 instructions/cycle

Real Stuff - Pentium



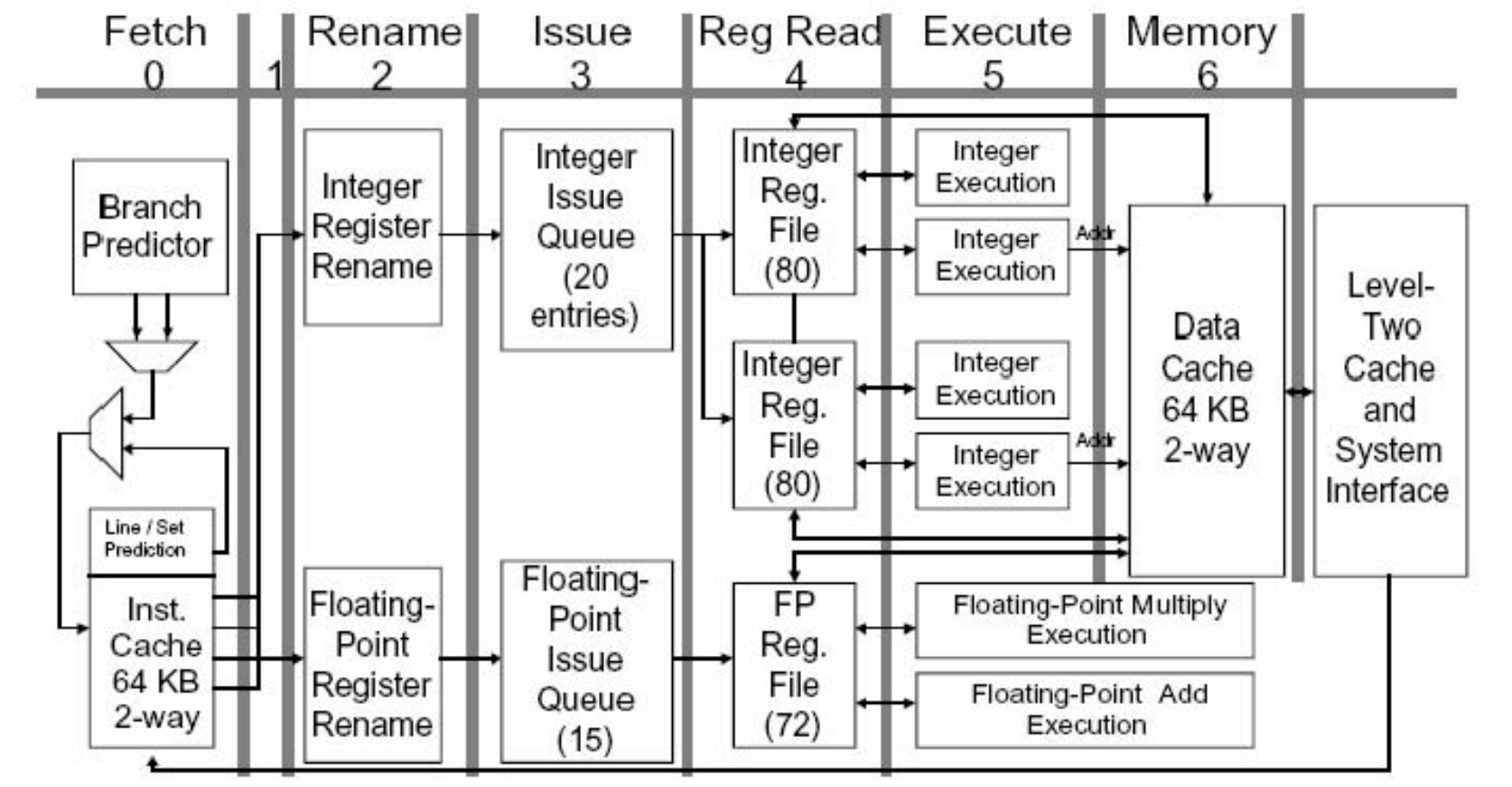
- **Pentium (P5) = 5 stages**
  Pentium Pro, II, III (P6) = 10 stages
- **Pentium 4 (NetBurst) = 20 stages**
- ***What are critical to performance?***

Real Stuff- AMD Opteron's processor



http://chip-architect.com/news/2003_09_21_Detailed_Architecture_of_AMDs_64bit_Core.html#1.12

Real Stuff- Alpha processor – Pure RISC

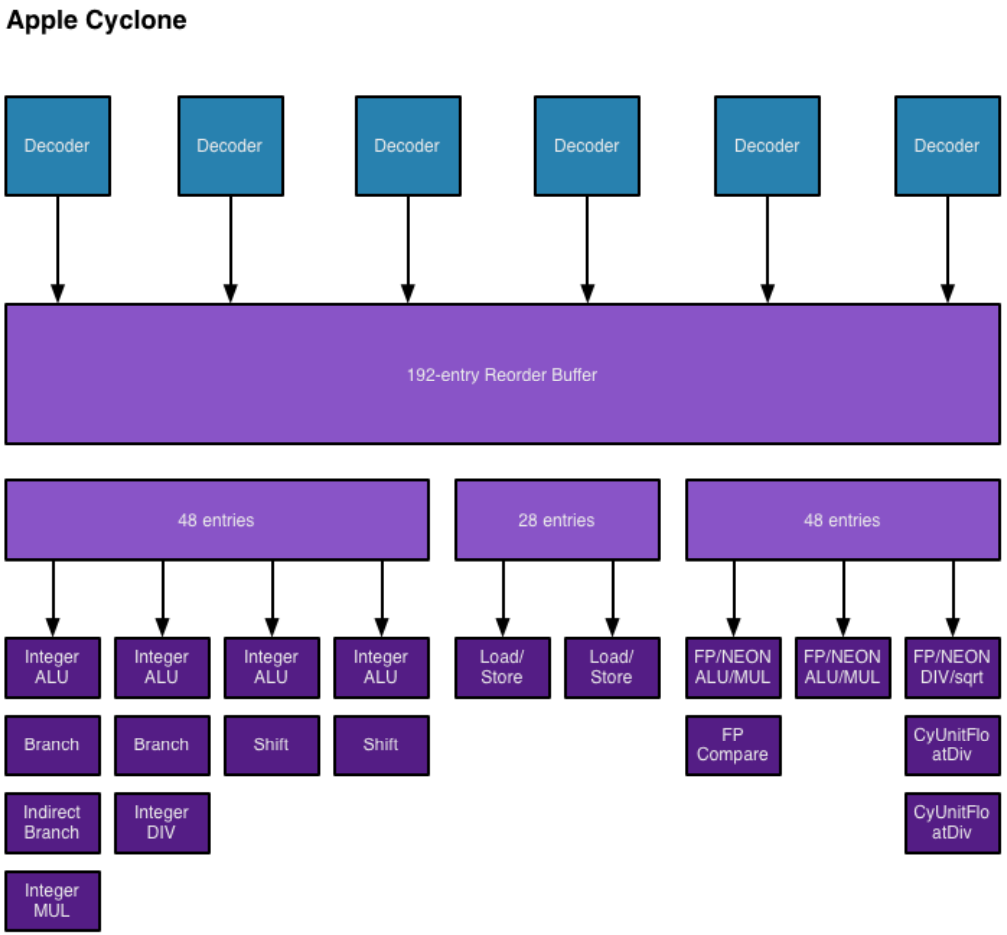


Alpha 21264's processor core

http://h18002.www1.hp.com/alphaserver/download/ev6chip.pdf

Recent AP – Apple A7

ARM ISA v8 64-bit



More on Processor Architecture

CNCE433 – Computer Organization (?)

CNCE4xx – Processor Architecture

CRE652 – Advanced Processor Architecture

Fall

CNCE433 – Processor Organization

CRE505 – Advanced Computer Architecture

CRE642 – Trusted Computing

CRE652 – Advanced Processor Architecture