

1 Exceptions

Exceptions are unexpected events that require jumping to an OS handler. Sources include page faults, illegal instructions, divide by zero, etc.

1.1 Define precise exceptions.

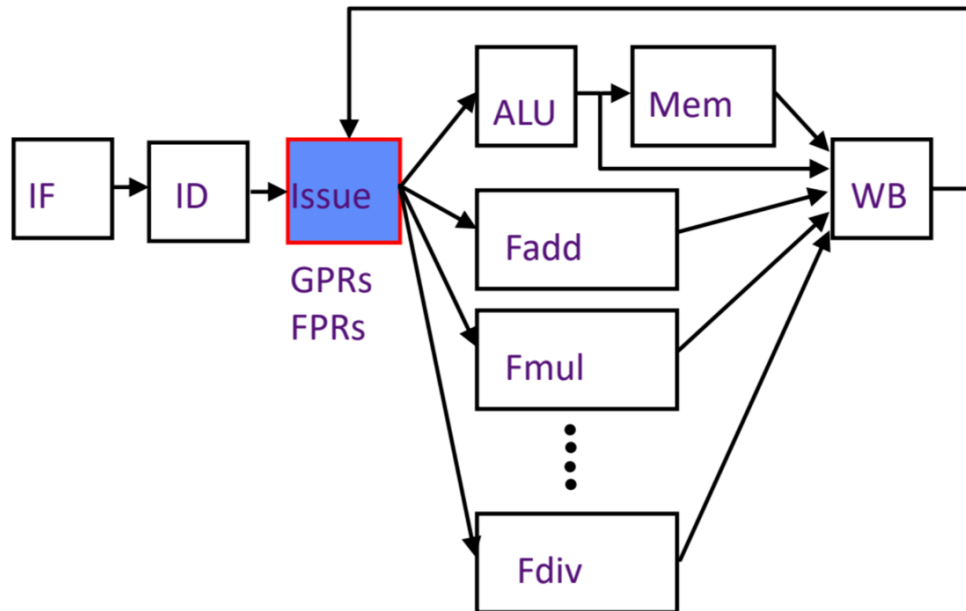
1.2 Why are precise exceptions useful?

1.3 Why might one not want to always implement precise exceptions?

1.4 Describe how program execution time may be affected by adding support for precise exceptions. Recall, $\text{execution time} = \text{instructions/program} * \text{cycles/instruction} * \text{time/cycle}$.

1.5 Compare and contrast precise exception handling with branch misprediction.

2 More Complex Pipelines



The Issue stage reads operands from the register files, checks hazards, and routes each instruction to the right functional unit (FU). Issue and completion could be out-of-order! Note that *completion* means the result exists and other instructions can use it; *commit* is different and will be made clearer later in the course.

2.1 What data hazards could exist in this pipeline? What mechanism is used to prevent them?

2.2 What are the structural hazards if we have unpipelined multi-cycle FUs vs. pipelined FUs with varying latencies? Assume we can only issue one instruction per cycle.

3 Cache Organization

3.1 Why caches?

3.2 Define and give an example of a program that exhibits the following:

(a) Temporal locality:

(b) Spatial locality:

3.3 Consider a 1 KiB 4-way set-associative cache with 32-byte cache lines. The address is 12 bits wide. How are the address bits partitioned?

Tag:

Index:

Offset:

4 Replacement Policies

4.1 What are some replacement policies? Why is random replacement not as bad as it sounds?

Suppose we see the following stream of accesses where A, B, C, D, and E represent unique addresses from different lines that all map to the same set:

A, B, C, D, B, A, E

Assume the cache has four ways and all lines in the set are initially invalid. When address E is accessed, one of the cache lines must be evicted. For each replacement policy, which line gets evicted?

4.2 First In, First Out (FIFO)

State: the next line to replace. On a replacement, increment the state. Assume the state starts at 0.

Access	Way0	Way1	Way2	Way3	Policy State After Request
A					
B					
C					
D					
B					
A					
E					

4.3 Not Most Recently Used (NMRU)

Almost the same as FIFO. State: the next line to replace. On a replacement *or* a hit to the line-to-be-replaced, increment the state. Assume the state starts at 0.

Access	Way0	Way1	Way2	Way3	Policy State After Request
A					
B					
C					

Access	Way0	Way1	Way2	Way3	Policy State After Request
D					
B					
A					
E					

4.4 Least Recently Used (LRU)

State: ways ordered from most recently used to least recently used.

Access	Way0	Way1	Way2	Way3	Policy State After Request
A					
B					
C					
D					
B					
A					
E					

4.5 Pseudo-LRU (PLRU)

State: a tree of bits (root bit, then the two child bits) pointing toward the pseudo-least-recently-used way.

Access	Way0	Way1	Way2	Way3	Policy State After Request
A					
B					
C					
D					
B					
A					
E					

5 Causes of Cache Misses

Recall: Average memory access time (AMAT) = Hit time + Miss rate * Miss penalty

5.1 Define the following terms:

(a) Compulsory Misses

(a) Capacity Misses

(a) Conflict Misses

6 Cache Write Policies

6.1 Describe the cache write policies on:

(a) Cache Hit:

(b) Cache Miss:

6.2 What combinations of these policies make sense?

6.3 What issues arise when multiple CPUs access the same shared memory?

7 Cache Optimizations

Recall: Average memory access time (AMAT) = Hit time + Miss rate * Miss penalty

For each technique, indicate whether implementing it will increase, decrease, or have no change on each aspect.

Technique	Hit Time	Miss Penalty	Miss Rate	Hardware Complexity
Smaller, simpler caches				
Multi-level caches				
Smart replacement policy				
Pipelined writes				
Write buffer				
Sub-blocks (sector cache)				