Quiz 4 Review

Suvinay Subramanian

6.823 Spring 2016

Topics Snapshot

» Microcoding

» VLIW
- Loop unrolling, software pipelining, predicated execution, speculative execution, trace scheduling

» Vector Computers
- Vector lanes, vector length register, masking, chaining

» GPUs
- Warps, branch divergence (masking)

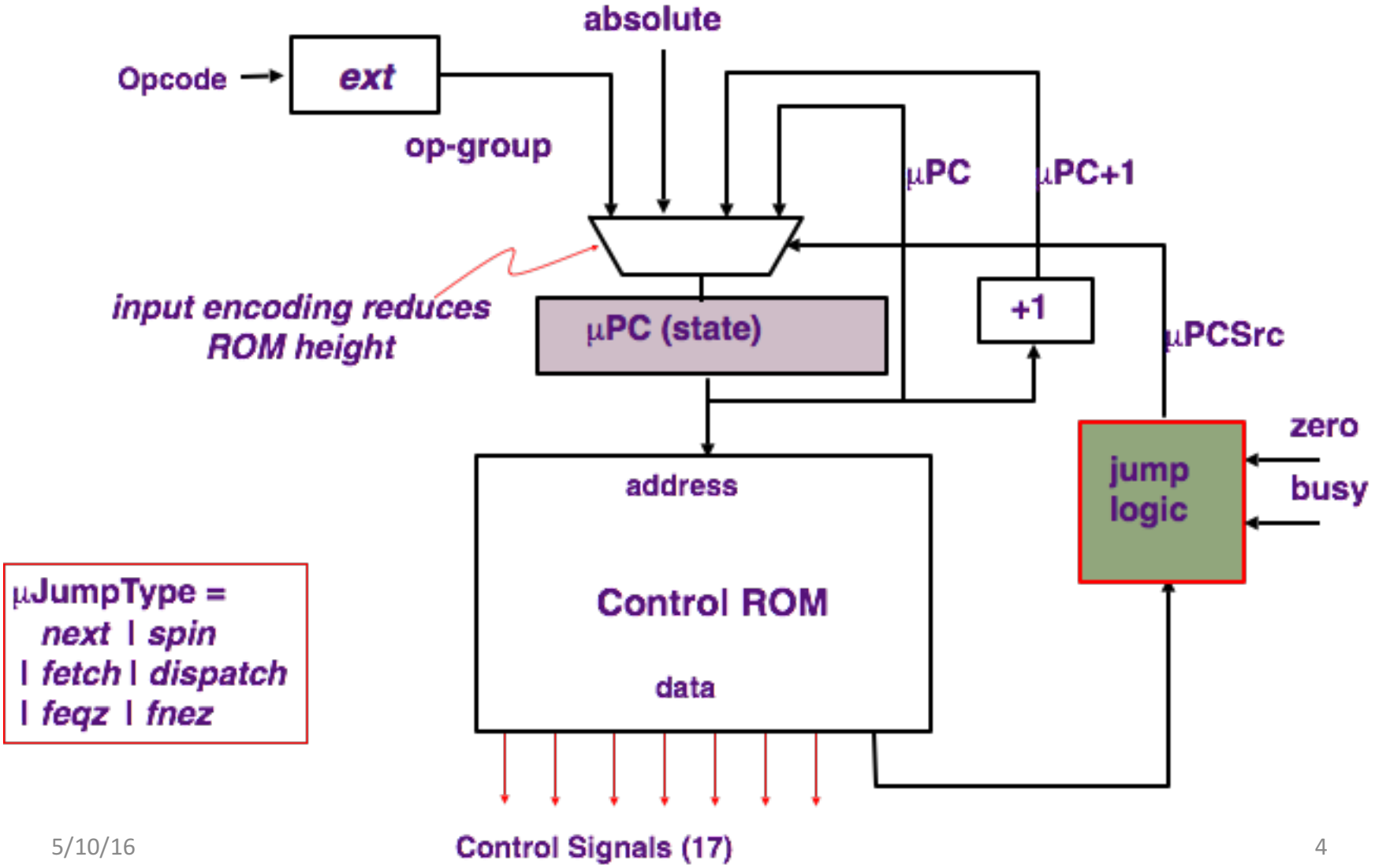
» Transactional Memory
- Eager, lazy versioning
- Optimistic, pessimistic conflict detection

Microcoding

» Abstraction layer between hardware and architecture of computer
  - i.e. separates ISA from actual hardware implementation details.

» Layer of hardware-level instructions that implement higher-level (i.e. ISA) instructions

Microcode Implementation



Microcode Fragments

State	Control points	next-state
$\mathbf{fetch_0}$	MA ← PC	next
$\mathbf{fetch_1}$	IR ← Memory	spin
$\mathbf{fetch_2}$	A ← PC	next
$\mathbf{fetch_3}$	PC ← A + 4	dispatch
…		
$\mathbf{ALU_0}$	A ← Reg[rs]	next
$\mathbf{ALU_1}$	B ← Reg[rt]	next
$\mathbf{ALU_2}$	Reg[rd]←func(A,B)	fetch
$\mathbf{ALUi_0}$	A ← Reg[rs]	next
$\mathbf{ALUi_1}$	B ← $\mathbf{sExt_{16}}$(Imm)	next
$\mathbf{ALUi_2}$	Reg[rd]← Op(A,B)	fetch

Topics Snapshot

» Microcoding

» VLIW
- Loop unrolling, software pipelining, predicated execution, speculative execution, trace scheduling

» Vector Computers
- Vector lanes, vector length register, masking, chaining

» GPUs
- Warps, branch divergence (masking)

» Transactional Memory
- Eager, lazy versioning
- Optimistic, pessimistic conflict detection

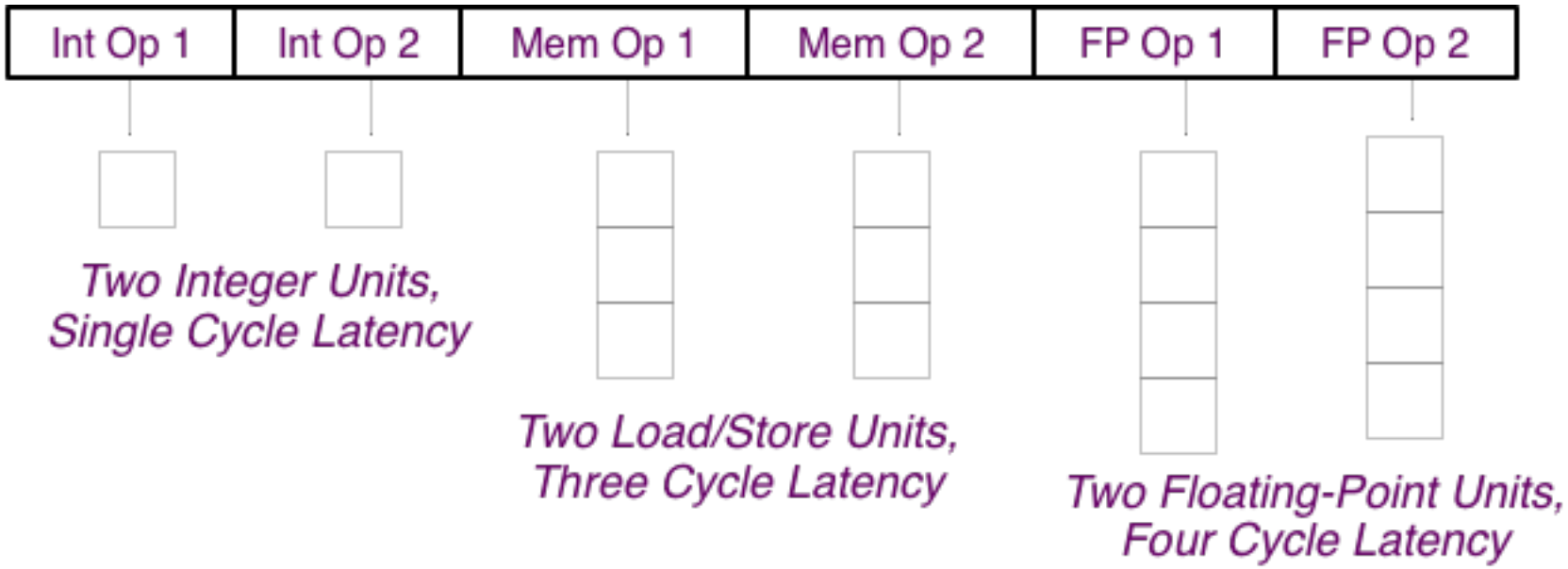
VLIW

» Premise: Static instruction scheduling + super-scalar execution to extract ILP.

» Tradeoff: Complex hardware vs Complex compiler "Conservation of complexity"

- VLIW machines: Compilers figure out independent instructions and schedules them suitably

VLIW Hardware



- Multiple operations packed into one instruction
- Each operation slot is for a fixed function
- Constant operation latencies are specified

VLIW Software

» Key Questions:

- How do we find independent instructions to fetch/execute?
- How to enable more compiler optimizations?

» Key Ideas:

- Get rid of control flow
  - Predicated execution, loop unrolling
- Optimize frequently executed code-paths
  - Trace scheduling
- Others: Software pipelining, speculative execution

Loop Unrolling

» Unroll loop to perform M iterations at once
  - Get more independent instructions
  - Need to be careful about case where M is not a multiple of number of loop iterations

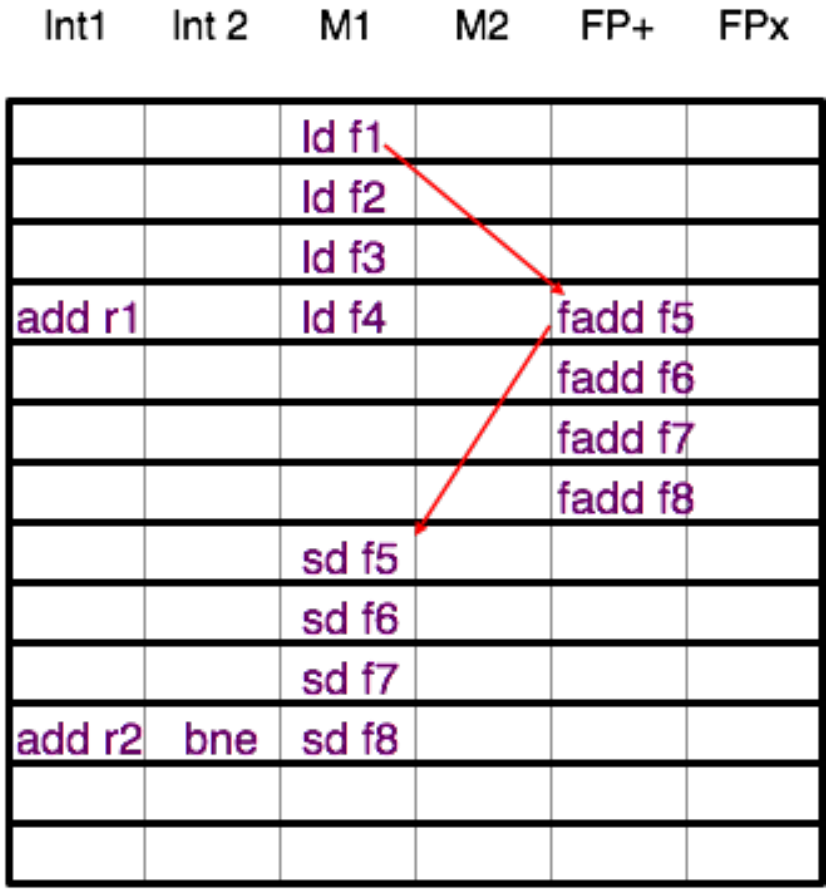
```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

```
for (i=0; i<N; i+=4)
{
    B[i]   = A[i] + C;
    B[i+1] = A[i+1] + C;
    B[i+2] = A[i+2] + C;
    B[i+3] = A[i+3] + C;
}
```

Loop Unrolling

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       sd f8, 24(r2)
        add r2, 32
       bne r1, r3, loop
```

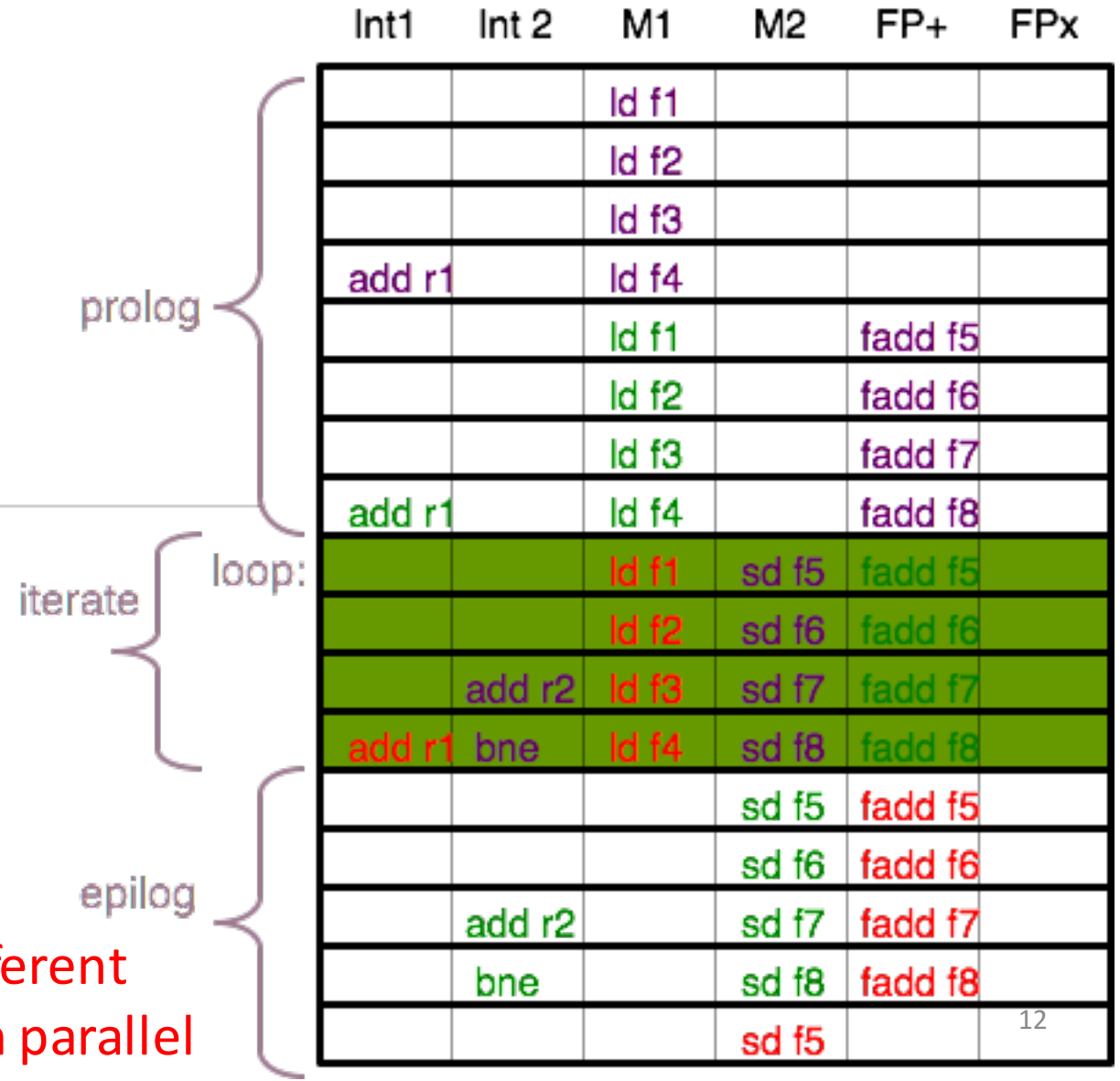
Schedule



You should be familiar with filling in such a table

Software Pipelining

```
loop: ld f1, 0(r1)
      ld f2, 8(r1)
      ld f3, 16(r1)
      ld f4, 24(r1)
      add r1, 32
      fadd f5, f0, f1
      fadd f6, f0, f2
      fadd f7, f0, f3
      fadd f8, f0, f4
      sd f5, 0(r2)
      sd f6, 8(r2)
      sd f7, 16(r2)
      add r2, 32
      sd f8, -8(r2)
      bne r1, r3, loop
```



Execute different iterations in parallel

Predicated, Speculative Execution

» Limited ILP within a basic-block; branches limit available ILP

» Predication: Eliminate hard-to-predict branches by converting control dependence to data dependence

- Each instruction (within the branch basic block) has a predicate bit set
- Only instructions with true predicates are executed and committed.

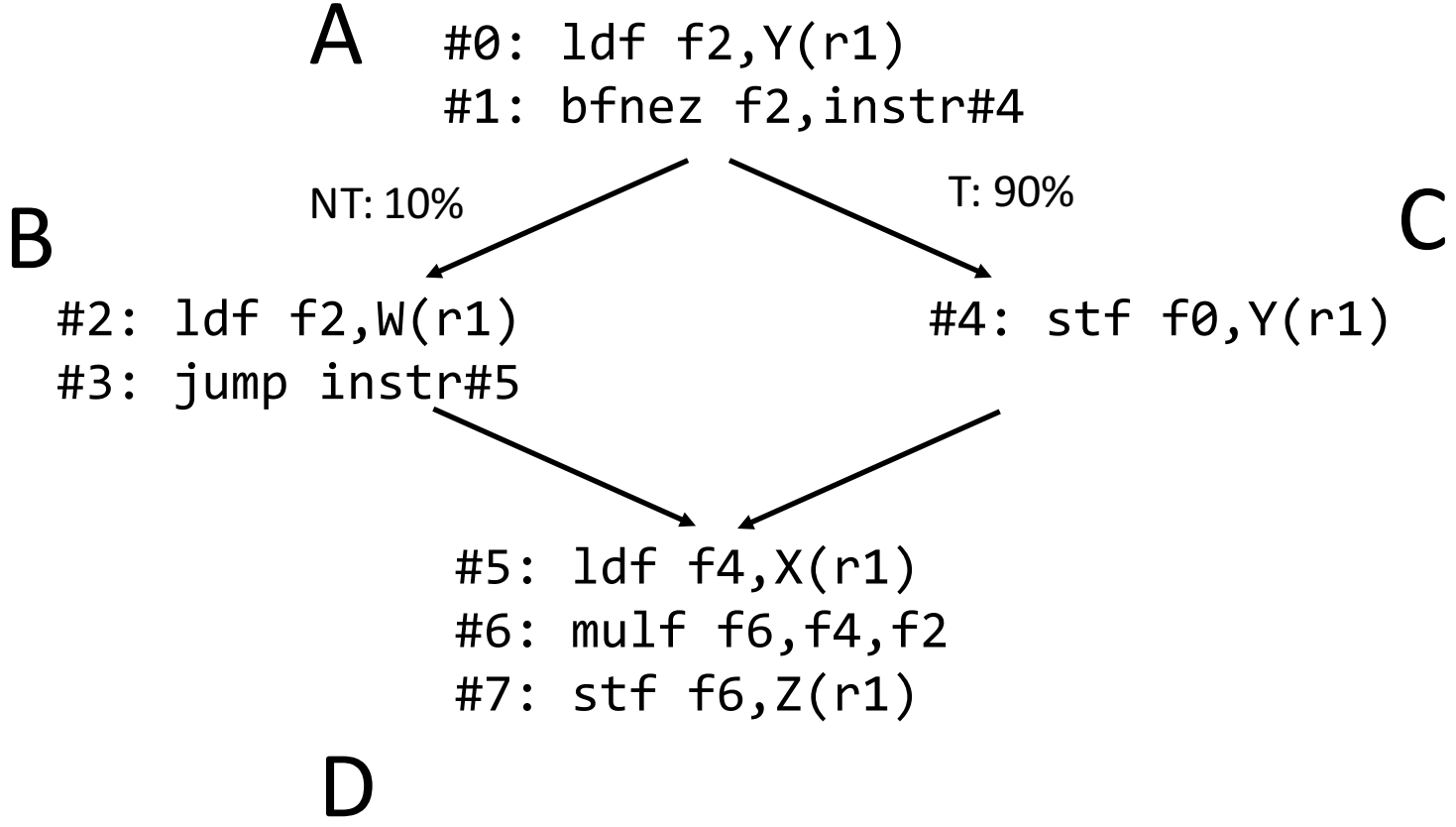
» Speculation: Move instructions above branches to explore more ILP options

Trace Scheduling

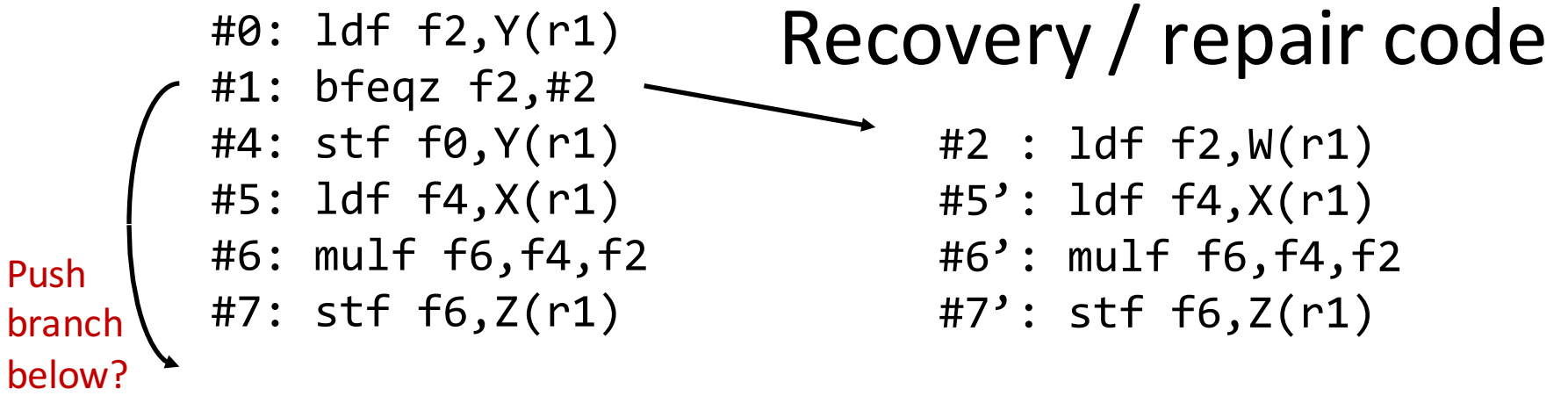
» Idea: For non-loop situations:

- Find common path in program trace
- Re-align basic blocks to form straight-line trace
  - Trace: Fused basic-block sequence
- Schedule trace
- Create fixup code in case trace != actual path

Trace Scheduling



Trace Scheduling



A, C, D superblock (trace)

Topics Snapshot

» Microcoding

» VLIW
- Loop unrolling, software pipelining, predicated execution, speculative execution, trace scheduling

» Vector Computers
- Vector lanes, vector length register, masking, chaining

» GPUs
- Warps, branch divergence (masking)

» Transactional Memory
- Eager, lazy versioning
- Optimistic, pessimistic conflict detection

Vector Computers

» Single-instruction multiple data (SIMD)
  - Single instruction to operate on an entire vector (instead of scalars)

» Vector length register (VLR)
  Vector masking (conditional execution)
  Vector chaining
  Vector lanes

If we ask code, we will provide syntax, meaning for vector instructions

GPUs

» Single-instruction multiple thread (SIMT)
  - Multiple instruction streams of scalar instructions

» Warps: A set of threads grouped together, and executing the same instruction (modulo divergence)

» Branch divergence: Handled through masking

Topics Snapshot

» Microcoding

» VLIW
- Loop unrolling, software pipelining, predicated execution, speculative execution, trace scheduling

» Vector Computers
- Vector lanes, vector length register, masking, chaining

» GPUs
- Warps, branch divergence (masking)

» Transactional Memory
- Eager, lazy versioning
- Optimistic, pessimistic conflict detection

Transactional Memory

» Idea: No locks, only shared data
Idea: Optimistic (speculative) concurrency
  - Execute critical section speculatively
  - Abort on conflicts

» Key properties:
  - Atomicity (all or nothing)
  - Isolation (no other code can observe updates before commit)
  - Serializability

Transactional Memory Taxonomy

» Data Versioning
- Eager
- Lazy

» Conflict Detection
- Pessimistic
- Optimistic

Data Management Policy

How to manage the "tentative work" that a transaction does

1. Eager versioning (undo-log based)
   - Update memory location directly
   - Maintain undo info in a log
   - Fast commits
   - Slow aborts
2. Lazy versioning (write-buffer based)
   - Buffer data until commit in a write buffer
   - Update actual memory locations at commit
   - Fast aborts
   - Slow commits

Conflict Detection Policy

How to ensure isolation between transactions

1. Pessimistic detection
   Check for conflicts during loads or stores

2. Optimistic detection
   Detect conflicts when a transaction attempts to commit

Topics Snapshot

» Microcoding

» VLIW
- Loop unrolling, software pipelining, predicated execution, speculative execution, trace scheduling

» Vector Computers
- Vector lanes, vector length register, masking, chaining

» GPUs
- Warps, branch divergence (masking)

» Transactional Memory
- Eager, lazy versioning
- Optimistic, pessimistic conflict detection

Thank You!
All the best ☺