CS152 Discussion Section 5

Midterm 1 Review

Feb 19
Spring 2024

Administrivia

- Midterm 1
    - Feb 27th during lecture time and place
- See the midterm logistics page on the website
    - 1 two-sided cheat sheet
        - Must be printed (cannot be used with an electronics)

General Advice

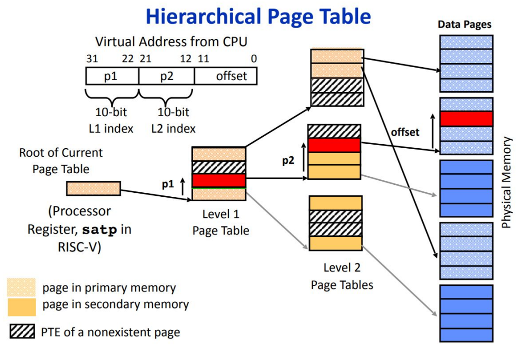
- Some problems may involve new variations on a basic design, but a solid understanding of the fundamentals will help you reason about new designs
    - e.g., what are the implications of making a particular modification?
- Consider how architectural features interact with each other
    - e.g., how does virtual memory impact cache design?
    - “Cross-cutting issues” sections in textbook can give perspective
- State your assumptions

Virtual Memory

- Motivations for address translation
- Paging vs segmentation
- Hierarchical page tables
  - How many PTEs are in each level of the page table?
  - What portion of the virtual address space do non-leaf PTEs map?
- Page table walk
  - How does the page table walker determine the location of the next level page table?
- TLBs
  - Compare TLBs to normal data caches
  - What happens on a context switch?

Virtual Memory

- Motivations for address translation
  - Wanted to overlap program execution because IO is slow. Location independent programs (base register), programs should not affect one another (bound register)
- Paging vs segmentation
  - Paging simplifies allocation, worst case disk traffic is bounded, may increase fragmentation
- Hierarchical page tables
  - How many PTEs are in each level of the page table?
  - What portion of the virtual address space do non-leaf PTEs map?
- Page table walk
  - How does the page table walker determine the location of the next level page table?
- TLBs
  - Compare TLBs to normal data caches
  - What happens on a context switch? TLB flush

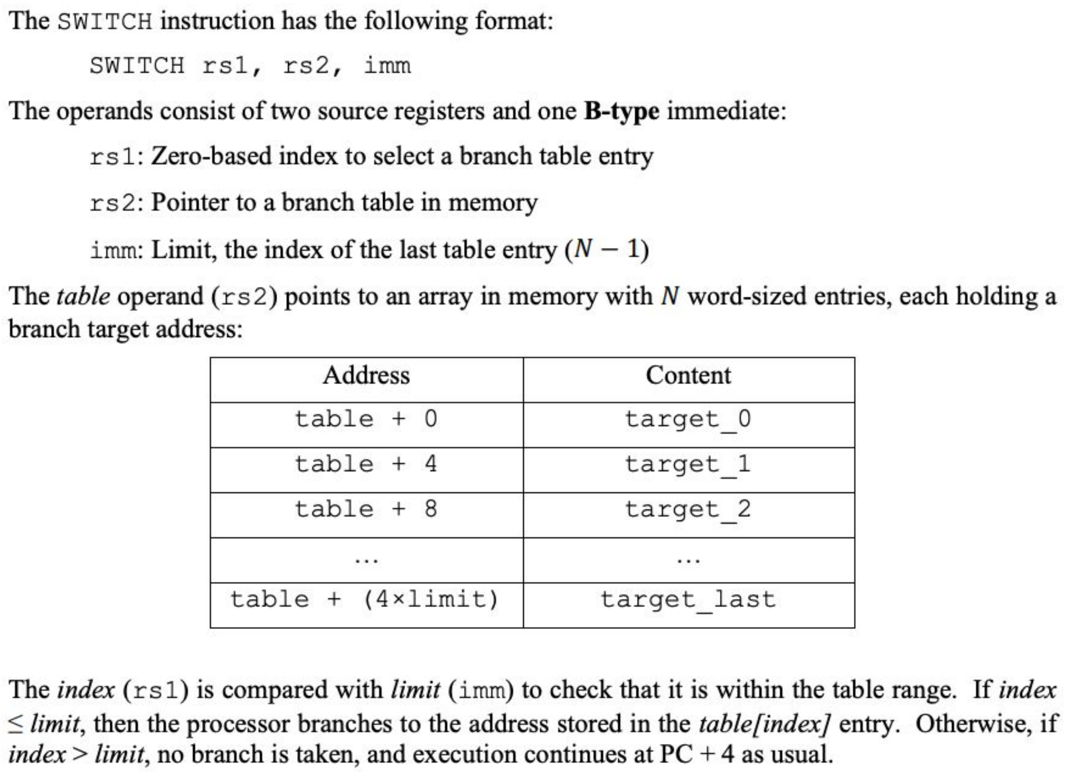


RISC-V Sv32 Virtual Memory Scheme

Microprogramming

- Decompose a complex instruction into multiple states (microinstructions)
  - Write pseudocode first
- Specify outputs for current state and state transition
- Optimize microinstructions to perform multiple actions if possible
  - Constraint is usually the single shared bus
- Simplify control signals by using don’t-cares (*)
  - Prioritize correctness first
- Be familiar with the single-bus machine from Handout 1
  - Excerpts from Handout 1 will be included for reference in exam materials
  - Note: Old exam solutions use a different RegEn/RegWr and MemEn/MemWr convention

Microprogramming (2020 MT1 Q2)



The `SWITCH` instruction has the following format:

```
SWITCH rs1, rs2, imm
```

The operands consist of two source registers and one **B-type** immediate:

`rs1`: Zero-based index to select a branch table entry

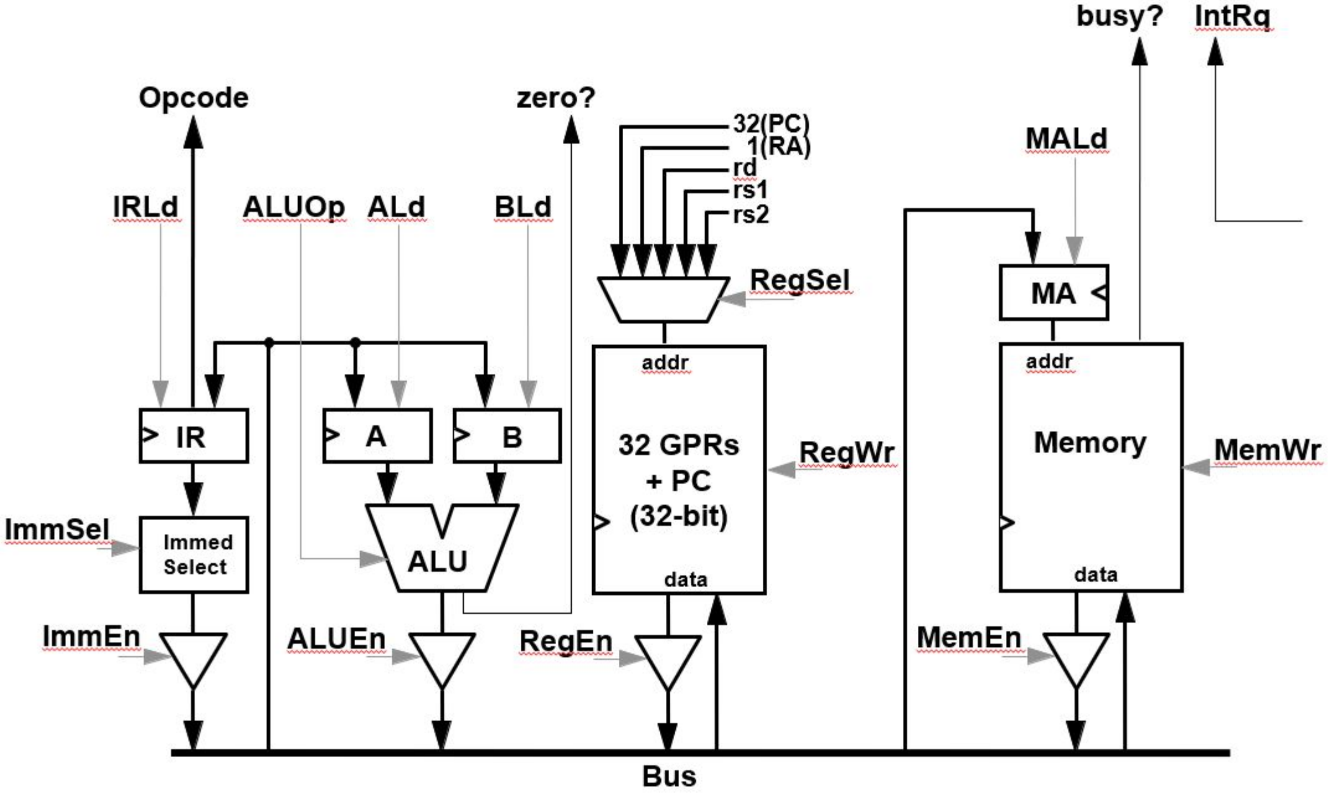
`rs2`: Pointer to a branch table in memory

`imm`: Limit, the index of the last table entry ($N - 1$)

The *table* operand (`rs2`) points to an array in memory with N word-sized entries, each holding a branch target address:

Address	Content
`table + 0`	`target_0`
`table + 4`	`target_1`
`table + 8`	`target_2`
…	…
`table + (4×limit)`	`target_last`

The *index* (`rs1`) is compared with *limit* (`imm`) to check that it is within the table range. If *index* ≤ *limit*, then the processor branches to the address stored in the *table[index]* entry. Otherwise, if *index* > *limit*, no branch is taken, and execution continues at PC + 4 as usual.


busy?
IntRq
Opcode
zero?
32(PC)
1(RA)
rd
rs1
rs2
MALd
IRLd
ALUOp
ALd
BLd
RegSel
MA
addr
addr
IR
A
B
32 GPRs
+ PC
(32-bit)
RegWr
Memory
MemWr
ImmSel
Immed
Select
ALU
data
data
ImmEn
ALUEn
RegEn
MemEn
Bus

State	Pseudocode	ldIR	Reg Sel	RegWr	en Reg	ldA	ldB	ALU Op	en ALU	ld MA	Mem Wr	en Mem	Imm Sel	en Imm	μBr	Next State
FETCH0	MA := PC; A := PC	*	PC	0	1	1	*	*	0	1	0	0	*	0	N	*
	IR := Mem	1	*	0	0	0	*	*	0	0	0	1	*	0	S	*
	PC := A + 4	0	PC	1	0	0	*	INC_A_4	1	*	0	0	*	0	D	*
...																

State	Pseudocode	ldIR	Reg Sel	RegWr	en Reg	ldA	ldB	ALU Op	en ALU	ld MA	Mem Wr	en Mem	Imm Sel	en Imm	μBr	Next State
(cont'd)																

State	Pseudocode	IR Ld	Reg Sel	Reg Wr	Reg En	A Ld	B Ld	ALUOp	ALU En	MA Ld	Mem Wr	Mem En	Imm Sel	Imm En	μBr	Next State
FETCH0:	MA ← PC; A ← PC	*	PC	0	1	1	*	*	0	1	*	0	*	0	N	*
	IR ← Mem	1	*	*	0	0	*	*	0	0	0	1	*	0	S	*
	PC ← A+4	0	PC	1	1	0	*	INC_A_4	1	*	*	0	*	0	D	*
…																
SWITCH0:	A ← Imm	0	*	*	0	1	*	*	0	*	*	0	B	1	N	
	B ← R[rs1]	0	rs1	0	1	0	1	*	0	*	*	0	*	0	N	
	If (A < B) { μBr to FETCH0 } A ← R[rs1]	0	rs1	0	1	1	0	SLTU	0	*	*	0	*	0	NZ	FETCH0
	A, B ← A + B	0	*	*	0	1	1	ADD	1	*	*	0	*	0	N	
	A ← A + B	0	*	*	0	1	*	ADD	1	*	*	0	*	0	N	
	B ← R[rs2]	*	rs2	0	1	0	1	*	0	*	*	0	*	0	N	
	MA ← A + B	*	*	*	0	*	*	ADD	1	1	*	0	*	0	N	
	PC ← Mem	*	PC	1	1	*	*	*	0	0	0	1	*	0	S	
	μBr to FETCH0	*	*	*	0	*	*	*	0	*	*	0	*	0	J	FETCH0

Iron Law of Processor Performance

- Terms: instructions/program, cycles/instruction, time/cycle
- Know which terms are influenced by:
  - Software
  - Architecture
  - Microarchitecture
  - Process technology
- Evaluate trade-offs between different design decisions
- State all assumptions (be reasonable)

Hazards

- Structural hazards
  - Shared bus, register file ports, unpipelined functional units, etc.
- Data hazards: RAW, WAR, WAW
  - How can WAW hazards arise in an in-order pipeline?
  - Consider through memory as well
- Control hazards
  - Where in a pipeline are unconditional direct jumps (JAL), unconditional indirect jumps (JALR), and conditional branches resolved?
  - Know what the jump/branch penalties are for a given pipeline

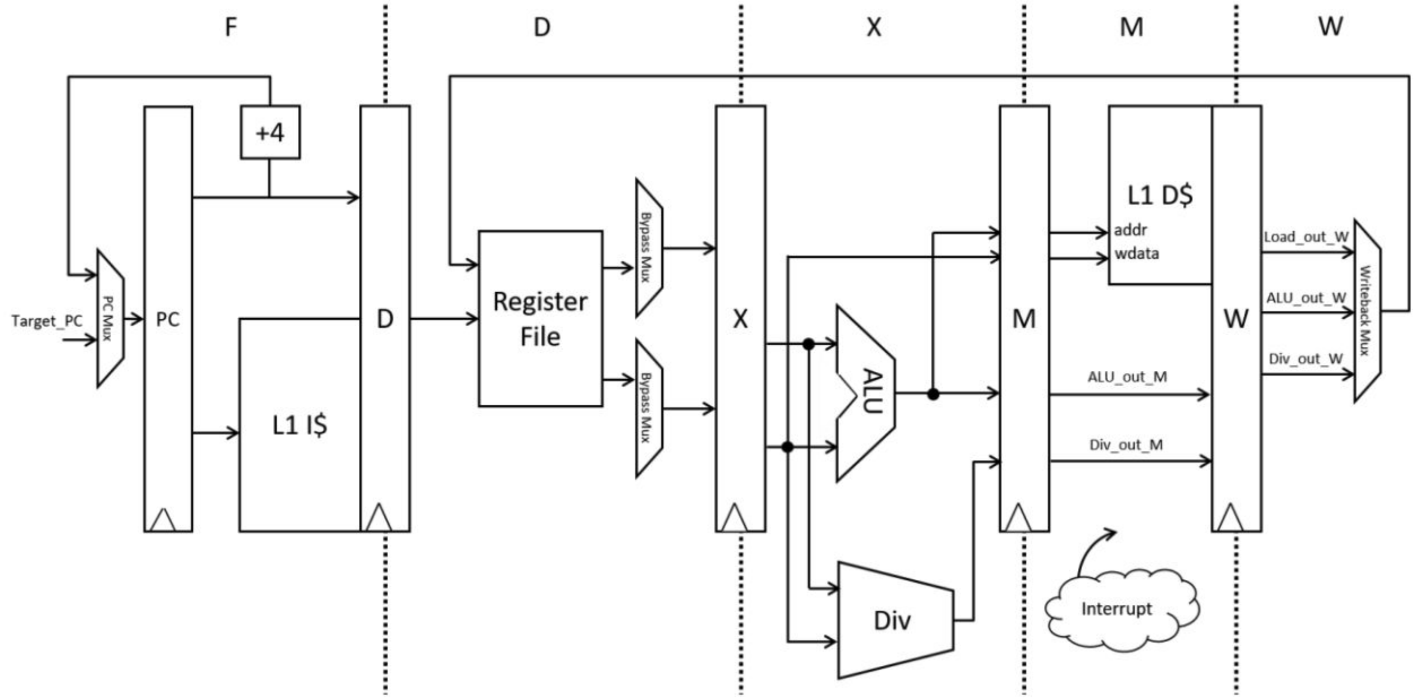
Pipelining

- What are the bypass paths in a fully-bypassed 5-stage datapath?
  - Does it ever make sense to bypass from the end of a stage to the start of a stage?
  - When can you “remove” a bypass path? What can make a bypass path unnecessary?
- Latency vs throughput/bandwidth vs occupancy
- How to calculate CPI
  - Measure from when the first instruction finishes to when last instruction finishes
  - CPI of 5-stage pipeline ≠ 5 when no stalls
- What is a critical path in synchronous digital logic?

Exceptions

- Synchronous traps vs asynchronous interrupts
- Precise vs imprecise exceptions
- How to implement precise exceptions in a pipeline
    - Where is the commit point in a pipeline?

Pipelining (2020 MT1 Q3)



Pipelining (2020 MT1 Q3)

What is the latency of a divide operation when the iterative divider produces 2 bits per cycle until it outputs a full 32-bit result?

Pipelining (2020 MT1 Q3)

What is the latency of a divide operation when the iterative divider produces 2 bits per cycle until it outputs a full 32-bit result?

32 / 2 = 16 cycles

Pipelining (2020 MT1 Q3)

What is the occupancy of a divide operation in cycles?

Pipelining (2020 MT1 Q3)

What is the occupancy of a divide operation in cycles?

16 cycles

Pipelining (2020 MT1 Q3)

Note that the `div` instruction in RISC-V cannot raise a data-dependent exception. To avoid pipeline stalls while a multi-cycle divide operation is in progress, the pipeline control logic allows subsequent instructions that do not depend on the divide result to be issued and completed before the divide has completed.

What additional hazards might be caused by `div` instructions, aside from the structural hazard on the divider itself? If any, describe how they could be resolved using an interlock.

Pipelining (2020 MT1 Q3)

Note that the `div` instruction in RISC-V cannot raise a data-dependent exception. To avoid pipeline stalls while a multi-cycle divide operation is in progress, the pipeline control logic allows subsequent instructions that do not depend on the divide result to be issued and completed before the divide has completed.

What additional hazards might be caused by `div` instructions, aside from the structural hazard on the divider itself? If any, describe how they could be resolved using an interlock.

WAW hazards can arise from the out-of-order completion of `div` instructions. For any subsequent instruction that writes to the same destination register as an ongoing `div` instruction, one approach is to stall at the D stage (or alternatively, in X or M at the potential cost of higher-fanout stall signals). This can be implemented using a comparator that checks the destination register of the `div` against the destination of the instruction in decode, or with a scoreboard that contains a bit-vector to track pending writes.

A potential structural hazard also exists on the write port of the register. When a `div` and another instruction that writes a register are both in the W stage, one of them must stall.

Although not a true control hazard, when a branch following a `div` that has not yet completed is taken, the ongoing divide operation should not be killed in the partial pipeline flush.

Caches

- Temporal and spatial locality
- 3 C’s of cache misses: compulsory, conflict, capacity
- Cache organization: associativity, replacement policy, write policy
- Memory system optimizations
- AMAT = hit time + miss rate * miss penalty
  - Know how to generalize equation for multi-level memory hierarchy
  - What design aspects affect each term?
- For a given cache configuration, where can lines be placed?
  - Which addresses correspond to which sets?

Conflict vs Capacity Misses

- **Conflict** misses: avoided with wider associativity and better replacement policy
  - More clearly identifiable when cache is underutilized due to poor placement
- **Capacity** misses: avoided only by increasing overall capacity
  - Harder to differentiate from conflict misses when access pattern is less regular
- Comparing miss *ratios* (miss rates) is sometimes more useful than attempting to categorize individual misses
  - Capacity component of miss ratio for cache C =
    miss ratio of fully-associative cache similar to C (same capacity, same line size)
    with *ideal replacement* minus compulsory miss ratio of C

Calculating Misses (PS2 Q2)

4 KiB direct-mapped cache with 32-byte cache lines

```
int A[128][32]; // row-major order
```

- Elements are striped across sets in groups of 8
- Large strided accesses (i.e., along a column) leave some cache sets unused
  - One column of 128 elements does *not* fit in 128 sets at once
- Same pattern even when base address of A is not aligned to cache size (4 KiB boundary)
  - Starting set index will be greater than 0
  - When determining whether interleaved accesses to two arrays will cause conflicts, the absolute addresses do not matter - only their relative displacement

Set	Elements

0	`A[ 0][0..7]`
	`A[32][0..7]`
	`A[64][0..7]`
	`A[96][0..7]`

1	`A[ 0][8..15]`
	`A[32][8..15]`
	`A[64][8..15]`
	`A[96][8..15]`

2	`A[ 0][16..23]`
	`A[32][16..23]`
	`A[64][16..23]`
	`A[96][16..23]`

3	`A[ 0][24..31]`
	`A[32][24..31]`
	`A[64][24..31]`
	`A[96][24..31]`

4	`A[ 1][0..7]`
	`A[33][0..7]`
	`A[65][0..7]`
	`A[97][0..7]`

...	...

127	`A[ 31][24..31]`
	`A[ 63][24..31]`
	`A[ 95][24..31]`
	`A[127][24..31]`

Prefetching

- HW prefetching vs SW prefetching
- What types of misses do they prevent?
- Usefulness vs Timeliness
  - What happens when prefetches are not useful?
  - What happens when prefetches are not timely?