Final Exam Topics

Numbers, bits, bytes, and ASCII characters

- Binary numbers, two's complement numbers, hexadecimal numbers.
- Conversion between different number systems, including decimal numbers, binary, hexadecimal.
- Sign extension.
- Addition/subtraction of binary/two's complement numbers. Negate a two's complement number.
- Bitwise logical operations.
- ASCII characters.

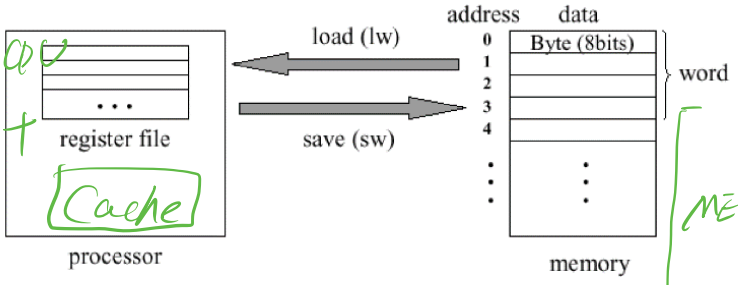
RISC-V ISA

- RISC-V instructions sets. Arithmetic, logical, memory, control flow instructions.

Instruction class	RISC-V examples
Arithmetic	add, sub, addi…
Data transfer	lw, sw, lb, lbu, sb lh, lhu, lui…
Logical	and, or, xor, andi, ori, sll, sra…
Cond. Branch	beq, bne…
Jump	Jal, jalr…

Data

- Accessing data in registers and memory.
- Array in memory. Array of words, half-words, and bytes (characters).

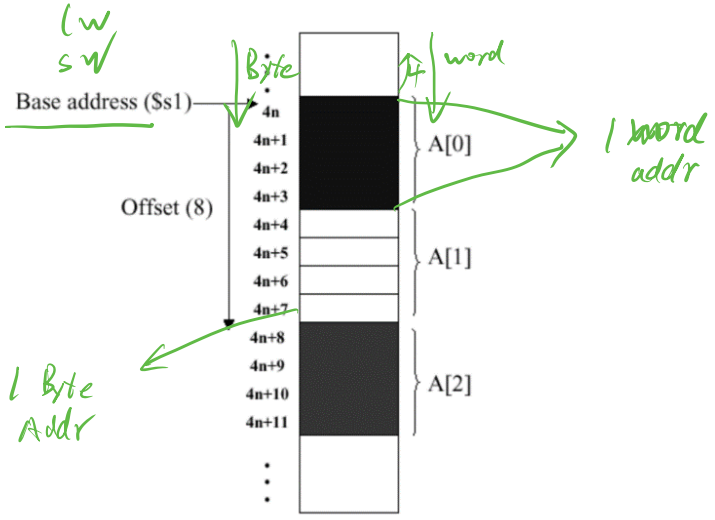


CPU → Cache → MEM

bit ↓ Byte ↓ half-word ↓ word

1 — 8 — 16 — 32

[Cache] block – 1 word, 2 words, N words



Control flow

- Program counter.
- RISC-V support for if-then-else, loops, functions.
- Compare values with beq, bne, etc.
- RISC-V calling convention. Caller-saved /callee-saved registers. Passing parameters to / returning value from functions.
- Stack. Push/pop. Save/restore registers on stack. Allocate storage space on stack.

Instruction Encoding

- Six instruction formats (R, I, S, SB, U, UJ types).
- Fields in different instruction formats.
- Describe the placement of immediate bits in machine code, for different types of formats.
- Describe how assembler place bits in immediate in machine code and how the processor construct.
- immediate when executing instructions.
- Given enough information, encode RISC-V instruction with 32 bits and read encoded instructions.

Assembly code

- Translate pseudocode or simple C code to RISC-V assembly code.
- Understand/debug RISC-V code.

Arithmetic for Computers

- operations on integers
  - addition and subtraction;
  - multiplication and division; multiplication/division algorithm and hardware;
  - RISC-V Multiplication Instructions
- floating-point and real numbers
  - Binary to decimal; Decimal to binary
  - Normalized numbers.
  - Encode Floating Point Numbers with Bits
  - IEEE Std 754-1985: Single precision; double precision
  - Denormal Numbers; meaning.
  - FP Support in RISC-V

Introduction to Digital Design

- use Boolean algebra to describe circuits
  - Boolean operations: AND, OR, NAND, NOR, XOR, NOT
  - use of truth tables to describe functionality

    - Boolean algebra
    - deriving Boolean expressions from truth tables
- decoder; multiplexor
- 1-bit ALU
    - use Nx1 mux to select desired operation
    - operations:
        - add, sub, AND, OR, NOR, etc. with individual gates
        - addition with full adder
        - subtraction with full adder and two's complement trick
- 32-bit ALU
- Describe the general structure of a state machine.
- Given a design of digital circuit, describe its behavior/function.
- Analyze the timing of digital circuit. Sequential Circuits
- Clock cycle, clock rate
- Bistable element, Memory Elements, Flip-Flops
- Triggering: falling edge and rising edge → structure hazard
- Registers: clock, input, output, control
    - Shift Register
- Register Files

Performance

- CPU Execution Time and Throughput; Relative Performance
- CPU Time= CPU Clock Cycles* Clock Cycle Time= CPU Clock Cycles/Clock rate
- CPU Time= Instruction Count *CPI Clock* Cycle Time
    - Instruction Count and CPI
    - average CPI → Exam 2
- How to improve the performance of a computer/CPU? Amdahl's Law Exam 2

Datapath Design Fig 4.21

- general concepts
    - tasks of ``executing an instruction" into stages
    - create the entire datapath for R/I type instruction.
- typical stages
    - instruction fetch (IF) - all
    - instruction decode (ID) / register fetch - all
    - execution (EX/EXE) - varies based on different instruction
    - memory access (MEM) - load and store only
    - register write (WB) - ALU and load only

R I SB U UJ

- standard components
    - register file
    - PC
    - ``add 4" unit, i.e., PC+4
    - sign-extension
    - instruction memory
    - data memory
    - ALU and its functions; ALU Control
    - multiplexors
- implementation choices
    - single-cycle: perform all tasks in a single clock cycle no pipeline Regs
    - multi-cycle/pipeline: use a clock cycle for each stage pipeline Regs

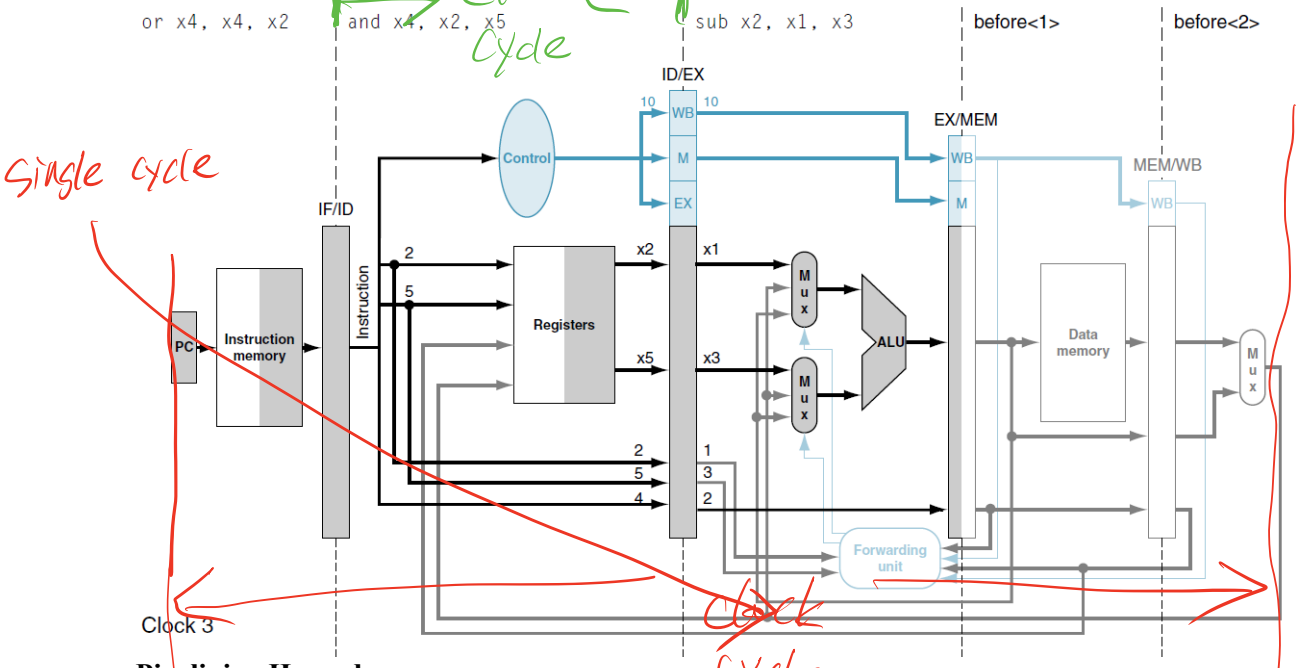
Control Implementation

- single cycle control unit
    - used for directing operations of datapath blocks

- generated based on opcode, function code (e.g., func 3 and func 7)
- figure out what control signals need to be activated for different R/I/S- type instructions.

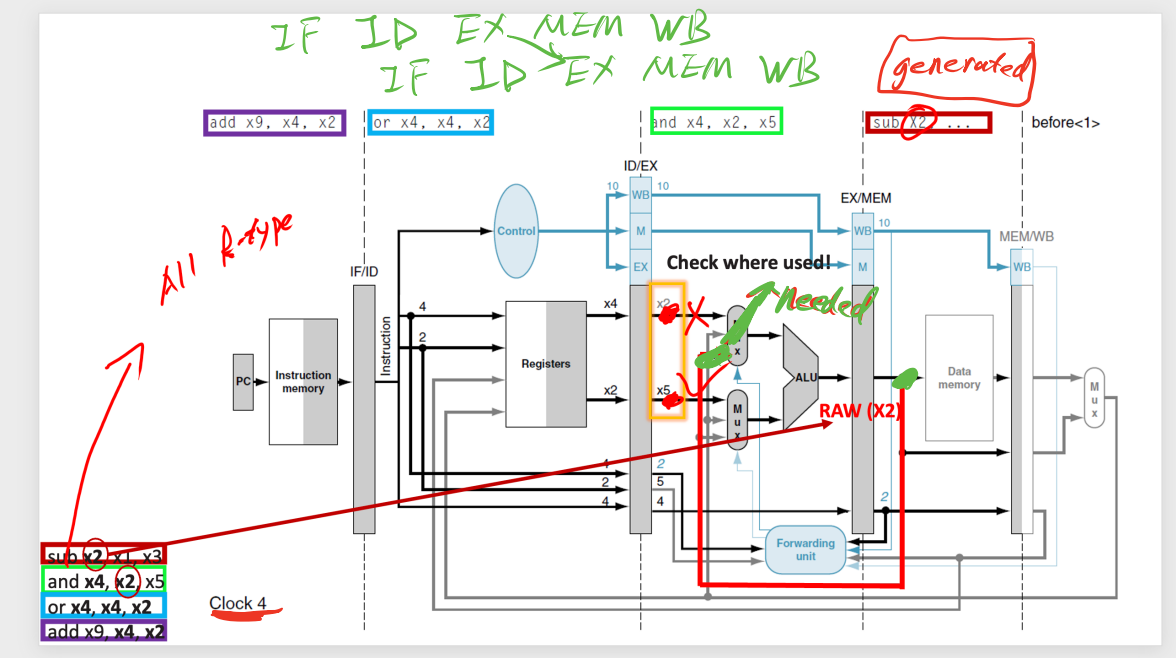
Pipelining

- basic concepts
- simple 5-stage pipeline
- pipeline registers
- pipeline control units

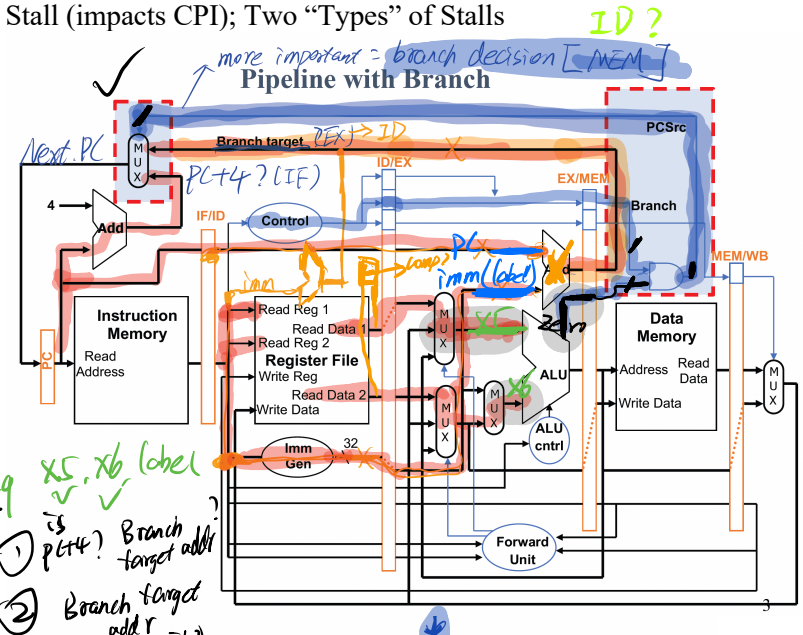


Pipelining Hazard

- Structural hazard
  - Add more resources (IM, DM)
  - Share Resource (Reg)
- Data Hazard
  - RAW, Load-use. Mem-to-mem copy
  - Forwarding
  - Stalling



- Control Hazard
  - Stall (impacts CPI); Two "Types" of Stalls



	1	2	3	4	5	6	7	8
I1:beq x1,x0,I9	IF	ID	EX	MEM	WB			
I2:and x12,x2,x5		IF	ID	EX	✗			
I3:or x13,x6,x2			IF	ID	✗			
I4:add x14,x2,x2				IF	✗			
I9 or I2					IF			

  - Move decision point as early in the pipeline as possible, thereby reducing the number of stall cycles
    - Cost of resolving branch in ID
  - Branch prediction

- Static branch prediction
  - Predict not taken

- Predict not taken works well for branching at "top of the loop"
  - But such loops have "jumps" at the bottom of the loop to return to the top of the loop and incur stalls

- Predict not taken does NOT work well for branching at "bottom of the loop"
  - The branch is taken most times
  - Fall through instruction is always flushed

  - Predict taken

Observation: We can calculate the branch target address early.

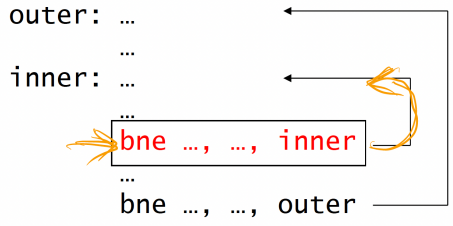
- It is easier to calculate branch target = PC + offset, because PC and offset are available early
- It is harder to read registers (data hazards!) and compare

Predict taken: predict branches will be taken

Once the branch target address is known, go to the branch target before the branch decision is made

- Need to know the branch target address. So predict taken *always* incurs (at least) one stall cycle
- Works for deeper pipelines, where the outcome of comparison is available in much later stages

- Dynamic branch prediction



If the inner loop is performed 10 times, a 1-bit predictor mispredicts twice!

80% prediction accuracy for a branch that is taken 90% of the time

Dynamic predictor

Predict	Actual	Iteraton
NT (X)	T	1
T	T	2
T	T	3
…		
T	T	9
T (X)	NT	10

- Re-order instructions to avoid hazards

Memory/Cache

- CPU/memory performance gap
- Memory hierarchy

- Why do we have a memory hierarchy
- What is memory hierarchy
- How does memory hierarchy work

- Locality principle
  - temporal
  - spatial

- Direct Mapped Cache

Direct-Mapped Cache Example

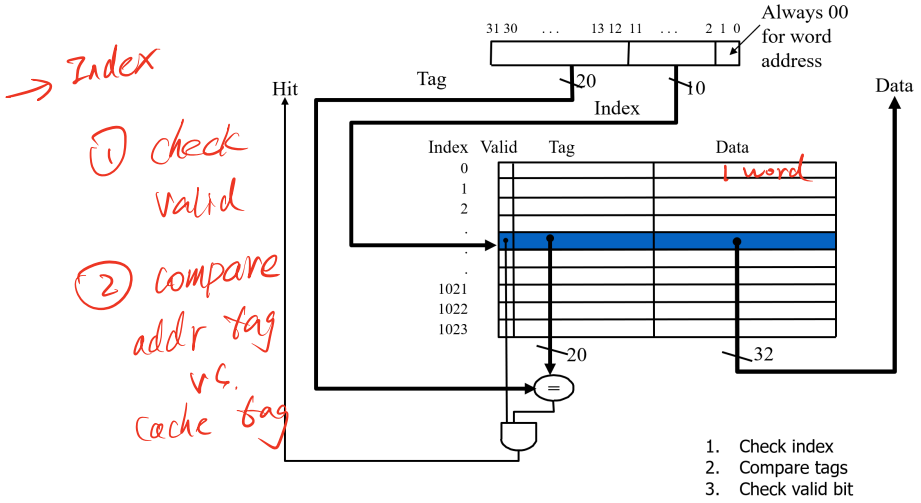
Start with an empty cache (all blocks are not ~~valid/blank~~).
Access the words at the following addresses ~~in sequence~~.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100
0 miss	**4** miss	**8** miss	**12** miss	**16** miss	**12** hit	**16** hit	**60** miss

00	Mem[0]	00	Mem[0]	00	Mem[0]	00	Mem[0]	01	~~00~~	~~Mem[0]~~	01	Mem[16]	01	Mem[16]	01	Mem[16]
		00	Mem[4]	00	Mem[4]	00	Mem[4]	00	Mem[4]		00	Mem[4]	00	Mem[4]	00	Mem[4]
				00	Mem[8]	00	Mem[8]	00	Mem[8]		00	Mem[8]	00	Mem[8]	00	Mem[8]
								00	Mem[12]		00	Mem[12]	00	Mem[12]	11	~~00~~ ~~Mem[12]~~ 60

Direct-Mapped Cache Diagram – Single-word Block

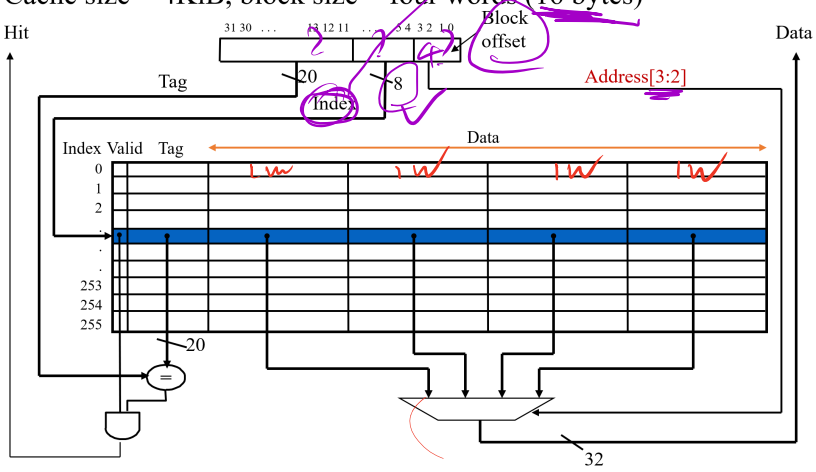
Cache size = 4KiB, block size = one word (4 bytes)



- Multiword Block Direct Mapped Cache

Direct-Mapped Cache Diagram – Multiword Block

Cache size = 4KiB, block size = four words (16 bytes)



Another cache of the same size

Let cache block hold two words (8 bytes). Again, start with an empty cache.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

0 miss

00	Mem[4]	Mem[0]

4 hit

00	Mem[4]	Mem[0]

8 miss

00	Mem[4]	Mem[0]
00	Mem[12]	Mem[8]

12 hit

00	Mem[4]	Mem[0]
00	Mem[12]	Mem[8]

16 miss

~~00~~ 01	~~Mem[4]~~ 20	~~Mem[0]~~ 16
00	Mem[12]	Mem[8]

12 hit

01	Mem[20]	Mem[16]
00	Mem[12]	Mem[8]

16 hit

01	Mem[20]	Mem[16]
00	Mem[12]	Mem[8]

60 miss

01	Mem[20]	Mem[16]
~~00~~ 11	~~Mem[12]~~ 60	~~Mem[8]~~ 56

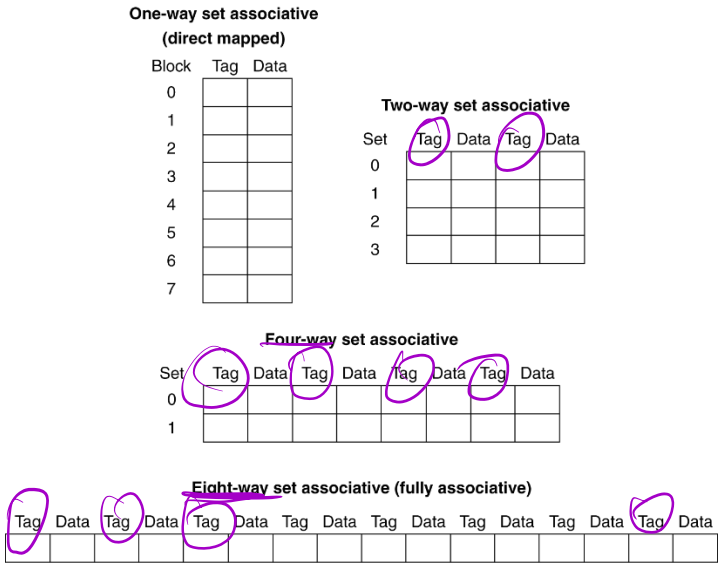
- Handling Write Hits
  - Write-through
  - Write Back
- Handling Miss
  - Write allocate or No write allocate
- Cache Performance
  - Memory-stall cycle; CPI_{stall}

    MemoryStallCycles = accesses/program × miss rate × miss penalty

    MemoryStallCycles = IC × misses/instruction × miss penalty

    CPU time = IC × ($CPI_{perfect}$ + MemoryStallCycles/Instruction) × CC
- Average Memory Access Time (AMAT)

  AMAT = Hit time + Miss rate × Miss penalty
- n-way set associative cache



- Multi-level Cache