CS 110
Computer Architecture
Review Midterm II

http://shtech.org/courses/ca/

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

Midterm II

- Date: Thursday, May 17
- Time: 10:15 - 12:15 (similar to last time)
- Venue: Teaching Center 301 + 302
- Closed book:
  - You can bring **two** A4 pages with notes ((both sides; English preferred; Chinese is OK): Write your Chinese and **Pinyin** name on the top!
  - This time **HANDWRITTEN** only!
  - You will be provided with the MIPS ”green sheet”
  - No other material allowed!

Midterm II

- Switch cell phones **off**! (not silent mode – off!)
  - Put them in your bags.
- Bags to the front. Nothing except paper, pen, 1 drink, 1 snack on the table!
- No other electronic devices are allowed!
  - No ear plugs, music, …
- Anybody touching any electronic device will FAIL the course!
- Anybody found cheating (copy your neighbors answers, additional material, ...) will FAIL the course!


CASIO
JF-120TM
12 DIGITS
TWO WAY POWER
F CUT 5/4
4 3 2 1 0 A
+/-
MRC
C
ON
AC
7
4
1
0
00
8
5
2
9
6
3
=
+
TAX-
RATE SET
%
TAX RATE
TAX+
GT
X
-


08:08
Thu, Sep 1
HUAWEI


Tue
9


COMPUTER
ORGANIZATION
AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
FIFTH EDITION
DAVID A. PATTERSON
JOHN L. HENNESSY
MK
MORGAN KAUFMANN

FUNCTIONS OF SEVERAL VARIABLES

LEVEL CURVES (one down)

FUNCTION OF n VARIABLES

PARTIAL DERIVATIVES

SECOND PARTIAL DERIVATIVES

CLAIRAUT'S THEOREM

THE CHAIN RULE

CASE 1

CASE 2

CHAIN RULE: GENERAL VERSION

IMPLICIT DIFFERENTIATION

TANGENT PLANE TO A LEVEL SURFACE

NORMAL LINE TO A LEVEL SURFACE

MAXIMUM AND MINIMUM VALUES

FINDING ABSOLUTE MAX. AND MINS.

MAXIMIZING AND MINIMIZING

DOMAINS: Allowed (x,y), (x,y,z) RANGES: ℝ's w/ 's

ε-δ DEFINITION OF CONTINUITY

EQUATIONS OF TANGENT PLANES TO SURFACES

TOTAL DIFFERENTIAL

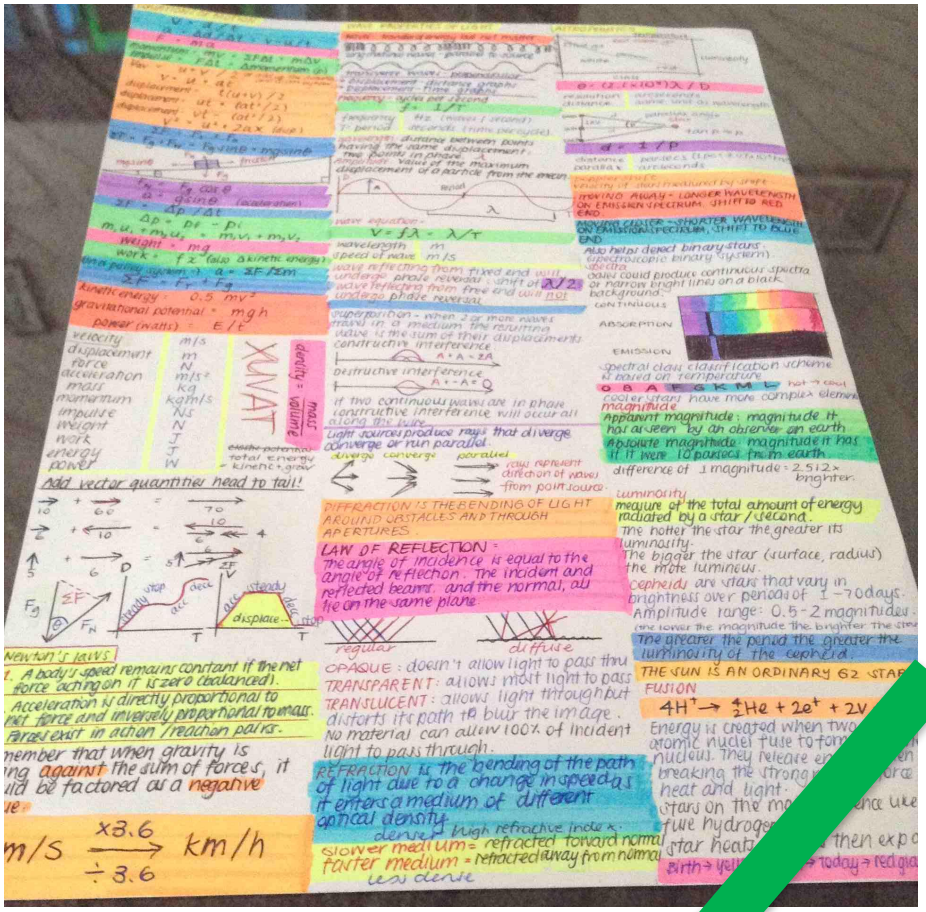
INCREMENTS Δx, Δy, Δz DIFFERENTIALS, dx, dy, dz

THEOREM

DEPENDENCY DIAGRAMS

THE GRADIENT VECTOR

DIRECTIONAL DERIVATIVES



SIFT REFERENCE GUIDE (V.1.1) – CREATING TIMELINES WITH THE SIFT WORKSTATION

SANS COMPUTER FORENSICS and INCIDENT RESPONSE

THE PURPOSE OF THIS REFERENCE GUIDE IS TO WALK THROUGH THE PROCESS OF BOOTING THE SIFT WORKSTATION, CREATING A TIMELINE ("SUPER" OR "MICRO") AND REVIEWING IT.

1. VISIT: http://computer-forensics11.sans.org/community/downloads

oad: SIFT Workstation VM Appliance

Do FT Workstation Installation

2. BOOT SIFT VM

Login: *sansforensics*
Password: *forensics*

$ sudo su

3. ELEVATE PRIVS

4. CONNECT IMAGE TO SIFT

Plug hard drive to physical host and attach to SIFT VM

5. HARD DRIVE MOUNTING *(if you are using log2timeline-sift and Single DD you can skip to 7-A)*

SINGLE OR SPLIT IMAGE (2 options):

EWF/

nt_ewf.py image.E01 /mnt/ewf

 # mage.E01 /mnt/ewf/

Not Needed For 7-A

 # mount -t ntfs -o ro,loop,show_sys_files,streams_interface=wind offset=#### /mnt/ewf/<image> /mnt/windows_mount/

MOUNT TO MOUNT POINT

DD

SINGLE IM

 # mount -t ntfs -o ... w_sys_files,streams_interface=windows,offset=#### image.dd /m... ows_mount/

SPLIT IMAGE (2 step proc...

 # affuse image.001 /mnt/aff

 # mount –t ntfs-3g –o loop,ro,show_... /mnt/aff/<image> /mnt/windows_...

HOW TO CALCULATE THE OFFSET FOR MOUNTING

1. Run mmls to query partition layout
 # mmls image.E01
2. Identify partition and byte offset
3. (Partition byte offset) x (bytes per sector) = **offset #### to use!**
Example: 63 X 512 = 32256

Note: If needed, repeat for each partition. Make new mount point:
***# mkdir** /mnt/windows_mount2/*

6. log2timeline default timezone is set to examiner local host. To change use -z [TIMEZONE] option. To list all available timezones:
 # log2timeline -z list

7-A & 7-B

7-A: AUTOMATED SUPER TIMELINE CREATION

log2timeline-sift -o –z [TIMEZONE] -p [PARTITION #] -i [IMAGE FILE]

DISK IMAGE (prompt for partition, mount, and run):

XP **# log2timeline-sift –z EST5EDT -i image**

WIN7 **# log2timeline-sift -win7 -z EST5EDT -i image**

FOR PARTITION (mount and run using all applicable pl...

XP **# log2timeline-sift –z EST5EDT -p 0 -i parti...**

WIN7 **# log2timeline-sift -win7 -z EST5EDT ...tion**

OTHER USAGE EXAMPLES:

Display list of available plu...
 # log2timeline -f list
Run log2timeline use... use only specific plugins:
 # log2timeline-sift ...refetch –z EST5EDT -i image.dd
Help (man page...
 # log2timeli...

NUAL "MICRO" TIMELINE CREATION

...TIONS] [-f FORMAT] [-z TIMEZONE] [-o OUTPUT MODULE] [-w ...G_FILE/LOG_DIR [--] [FORMAT FILE OPTIONS]

...EM METADATA (using log2timeline or fls)

...stem data w/log2timeline from mounted file system:
 # log... ...f mft -o mactime –r -z EST5EDT -w
mft.bo... ...olume/
OR Extract M... ...age using Sleuthkit:
 # fls -m "" -o offs... ...dd > fls.body
Convert body file for... ...format w/ mactime:
 # mactime –b fls.body –d...

ARTIFACTS (run l2l on mounted file s... ...h plugins recursively)

Extract artifacts w/ log2timeline and ru... ...d file system:
 # log2timeline -f firefox3,chrome -o mactim... ...EDT -w
web.body /mnt/volume/
Convert body file format to CSV format w/ mactime:
 # mactime –b log2timeline.body –d > log2timeline.csv

HELP? OPTIONS? USAGE?
log2timeline -help
Log2timeline-sift -help
L2t_process -help

OTHER log2timeline OUTPUT FORMATS
Note: CSV is Default Output
-BeeDocs - Mac OS X visualization tool
-CEF - Common Event Format - ArcSight
-CFTL - XML file- CyberForensics TimeLab visualization tool
-CSV - comma separated value file
-Mactime - Both older and newer version of the format supported for use by TSK's mactime
-SIMILE - XML file - SIMILE timeline visualization widget
-SQLite - SQLite database
-TLN - Tab Delimited File
-TLN - Format used by some of H Carvey tools, expressed as a ASCII output
-TLNX - Format used by some of H Carvey tools, expressed as a XML document

8. CSV F... ...PUT *(/cases/timeline-output-folder)*

-dat... ...e event, in the format of MM/DD/YYYY
-t... ...of day, expressed in a 24h format, HH:MM:SS
-z: the timezone that was used to call the tool with.
-...: MACB meaning of the fields, comp w/ mactime format.
-source: Source short name (i.e. registry entries are REG)
-sourcetype: Desc of the source ("Internet Explorer" instead of WEBHIST)
-type: Timestamp type (i.e. "Last Accessed", "Last Written")
-user: Username associated with the entry, if one is available.
-host: Hostname associated with the entry, if one is available.
-short: Contains less text than the full description field.
-desc: where majority info is stored, the actual parsed desc of the entry.
-version: Version number of the timestamp object.
-filename: Filename with the full path that contained the entry
-inode: inode number of the file being parsed.
-notes: Some input modules insert additional information in the form of a note, which comes here. Or it can be used during the review.
-format: Input module name used to parse the file.
-extra: Additional information parsed is joined together and put here.

9. FILTER TIMELINE

Filter timeline with date range to include only:
l2t_process -b timeline.csv MM-DD-YYYY..MM-DD-YYYY > filtered.csv
Filter timeline with keyword list (one term per line in keywords.txt):
l2t_process -b timeline.csv -k keywords.txt > filtered.csv
What sources are in your timeline?
awk–F , '{print $6;}' timeline.csv| grep–v sourcetype|sort | uniq
Find all LNK files that reference E Drive
grep"Shortcut LNK" timeline.csv| grep"E:"
FiindMountPoints2 entries that reference E Drive
grep"MountPoints2 key" timeline.csv} | grep"E drive"
grepUSB timeline.csv| grep"SetupAPILog"

File System	M	A	C	B
Ext2/3	Modified	Accessed	Changed	N/A
FAT	Written	Accessed	N/A	Created
NTFS	File Modified	Accessed	MFT Modified	Created
UFS	Modified	Accessed	Changed	N/A

10. CONNECT TO SIFT

✓ ...> SETTINGS -> OPTIONS -> Shared Fo...ers -> Always Enabled (Check)

✓ 2. SIFT Desktop > VMware-Shared-Drive

✓ Access from a Win Machine \\SIFTWORKSTATION

11. REVIEW TIMELINE

Review timelines using:
- Open, Soft, Filter with Excel
- Import into SPLUNK
- SIMILE
- Tapestry

log2timeline PARSING PLUGINS

apache2_error - Apache2 error log file
chrome - Chrome history file
encase_dirlisting - CSV file that is exported from encase
evt - Windows 2k/XP/2k3 Event Log
evtx - Windows Event Log File (EVTX)
exif - Metadata information from files using ExifTool
ff_bookmark - Firefox bookmark file
firefox2 - Firefox 2 browser history
firefox3 - Firefox 3 history file
ftk_dirlisting - CSV file that is exported from FTK Imager (dirlisting)
generic_linux - Generic Linux logs that start with MMM DD HH:MM:SS
iehistory - index.dat file containg IE history
iis - IIS W3C log file
isatxt - ISA text export log file
jp_ntfs_change - CSV output file from JP (NTFS Change log)
mactime - Body file in the mactime format
mcafee - Log file
mft - NTFS MFT file
mssql_errlog - ERRORLOG file produced by MS SQL server
ntuser - NTUSER.DAT registry file
opera - Opera's global history file
oxml - OpenXML document pcap
pcap - PCAP file
pdf - Available PDF document metadata
prefetch - Prefetch directory
recycler - Recycle bin directory
restore 0.9 - Restore point directory
safari - Safari History.plist file
sam - SAM registry file
security - SECURITY registry file
setupapi - SetupAPI log file in Windows XP
skype_sql - Skype database
software - SOFTWARE registry file
sol - .sol (LSO) or a Flash cookie file
squid - Squid access log (http_emulate off)
syslog - Linux Syslog log file
system - SYSTEM registry file
tln - Body file in the TLN format
volatility - Volatility output files (psscan2, sockscan2, ...)
win_link - Windows shortcut file (or a link file)
wmiprov - wmiprov log file
xpfirewall - XP Firewall log

List plugins # log2timeline -f list
...HELP EXPAND THIS LIST. BUILD PLUGINS!!!

BY DAVID NIDES (12/16/2011)
TWITTER: @DAVNADS
BLOG: DAVNADS.BLOGSPOT.COM
EMAIL: DNIDES@KPMG.COM

CREDITS TO: ED GOINGS, ROB LE... KRISTINN GUDJONSSON, KPMG &...!
QUESTIONS/FEEDBACK–CONTACT US!

KEY
Red text – image/source
Blue text – mount point
Purple text – output file
Green text – log2timeline plugins
Brown text - TimeZone

Do not look inside the midterm until we tell you to (until the time starts) !!

Midterm II

- Ask questions today!
- This review session does not/ can not cover all possible topics!
- Topics: SDS till Data-Level-Parallelism

Synchronous Digital Systems

Hardware of a processor, such as the MIPS, is an example of a Synchronous Digital System

Synchronous:

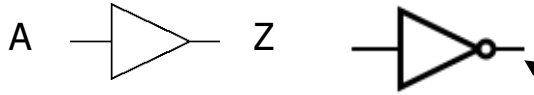
- All operations coordinated by a central clock
  - "Heartbeat" of the system!

Digital:

- Represent all values by discrete values
- Two binary digits: 1 and 0
- Electrical signals are treated as 1's and 0's
  - 1 and 0 are complements of each other
- High /low voltage for true / false, 1 / 0

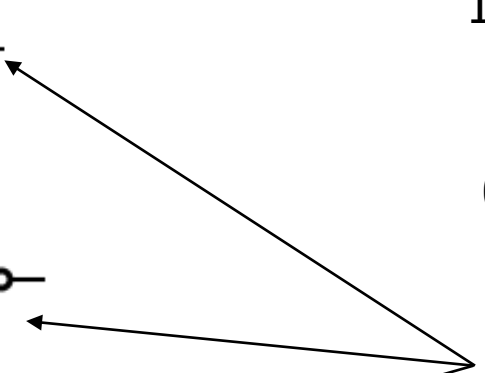
Combinational Logic Symbols

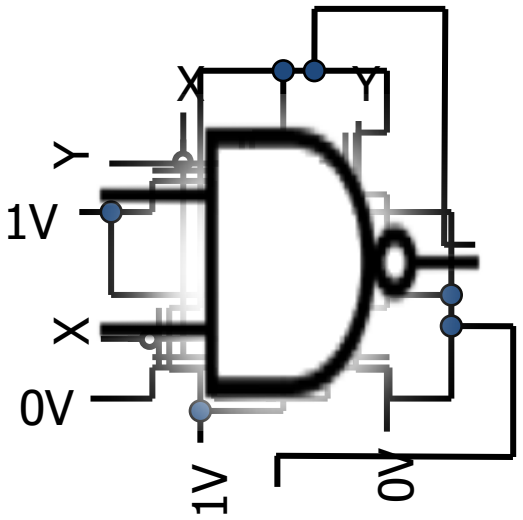
- Common combinational logic systems have standard symbols called logic gates
  - Buffer, NOT



  - AND, NAND

  - OR, NOR





Inverting versions (NOT, NAND, NOR) easiest to implement with CMOS transistors (the switches we have available and use most)

Boolean Algebra

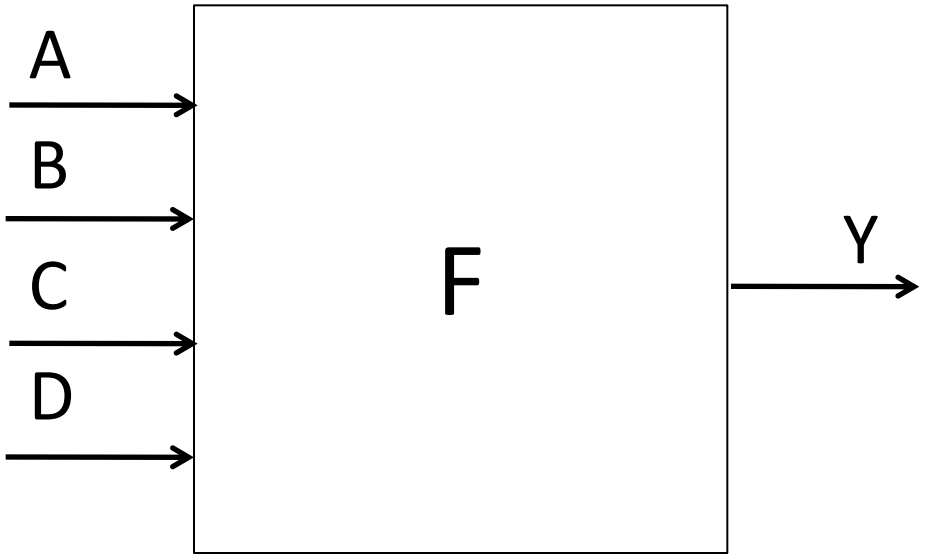
- Use plus "+" for OR
  - "logical sum" 1+0 = 0+1 = 1 (True); 1+1=2 (True); 0+0 = 0 (False)
- Use product for AND (a•b or implied via ab)
  - "logical product" 0*0 = 0*1 = 1*0 = 0 (False); 1*1 = 1 (True)
- "Hat" to mean complement (NOT)
- Thus

ab + a + $\overline{c}$

= a•b + a + $\overline{c}$

= (a AND b) OR a OR (NOT c)

Truth Tables for Combinational Logic

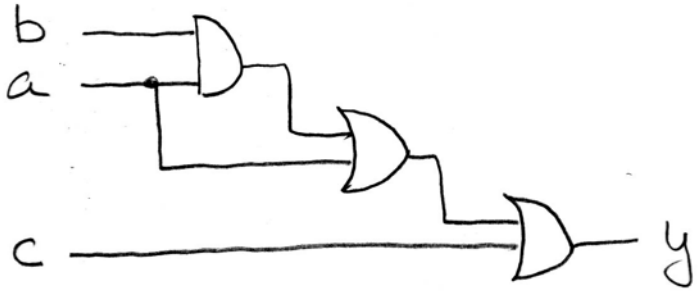


Exhaustive list of the output value generated for each combination of inputs

How many logic functions can be defined with N inputs?

a	b	c	d	y
0	0	0	0	F(0,0,0,0)
0	0	0	1	F(0,0,0,1)
0	0	1	0	F(0,0,1,0)
0	0	1	1	F(0,0,1,1)
0	1	0	0	F(0,1,0,0)
0	1	0	1	F(0,1,0,1)
0	1	1	0	F(0,1,1,0)
0	1	1	1	F(0,1,1,1)
1	0	0	0	F(1,0,0,0)
1	0	0	1	F(1,0,0,1)
1	0	1	0	F(1,0,1,0)
1	0	1	1	F(1,0,1,1)
1	1	0	0	F(1,1,0,0)
1	1	0	1	F(1,1,0,1)
1	1	1	0	F(1,1,1,0)
1	1	1	1	F(1,1,1,1)

Boolean Algebra: Circuit & Algebraic Simplification



original circuit

$$\downarrow$$

$$y = ((ab) + a) + c$$

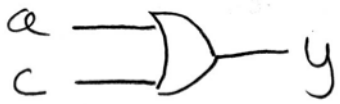
equation derived from original circuit

$$\downarrow$$

$$\begin{aligned} &= ab + a + c \\ &= a(b + 1) + c \\ &= a(1) + c \\ &= a + c \end{aligned}$$

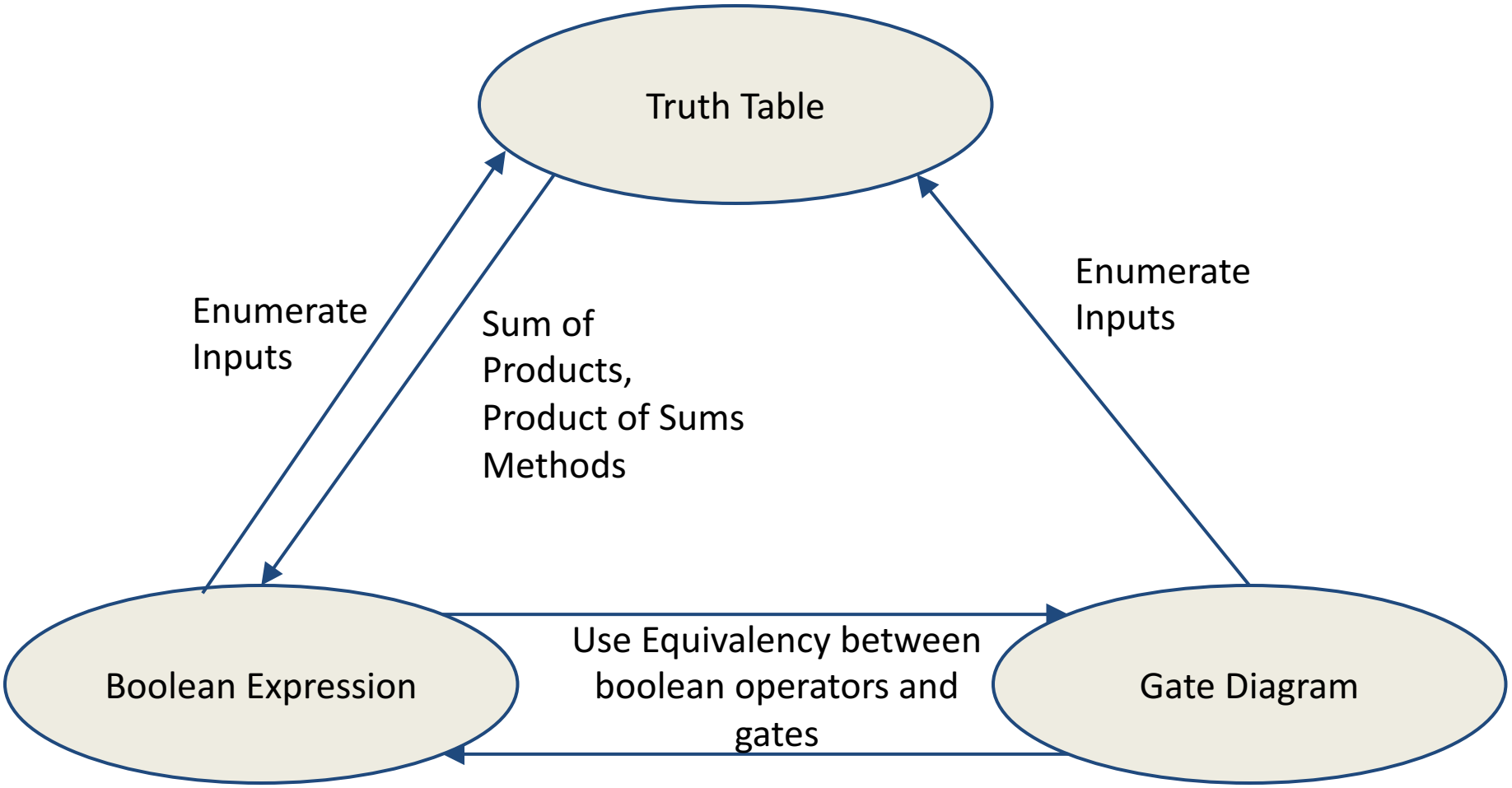
algebraic simplification

$$\downarrow$$



simplified circuit

Representations of Combinational Logic (groups of logic gates)

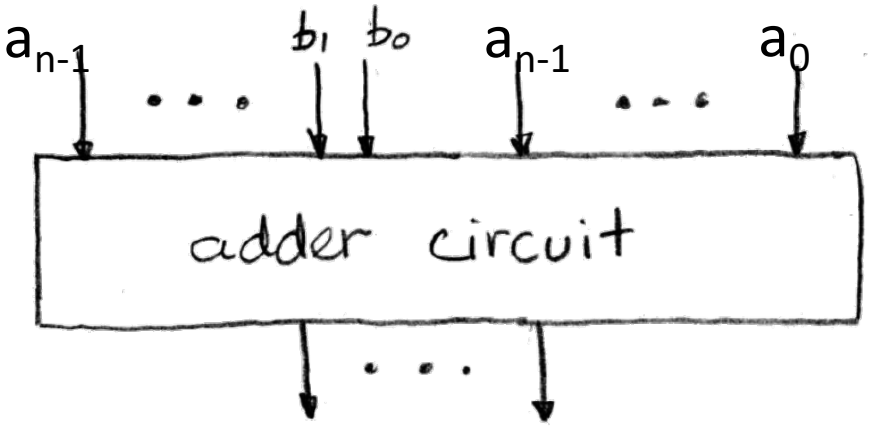


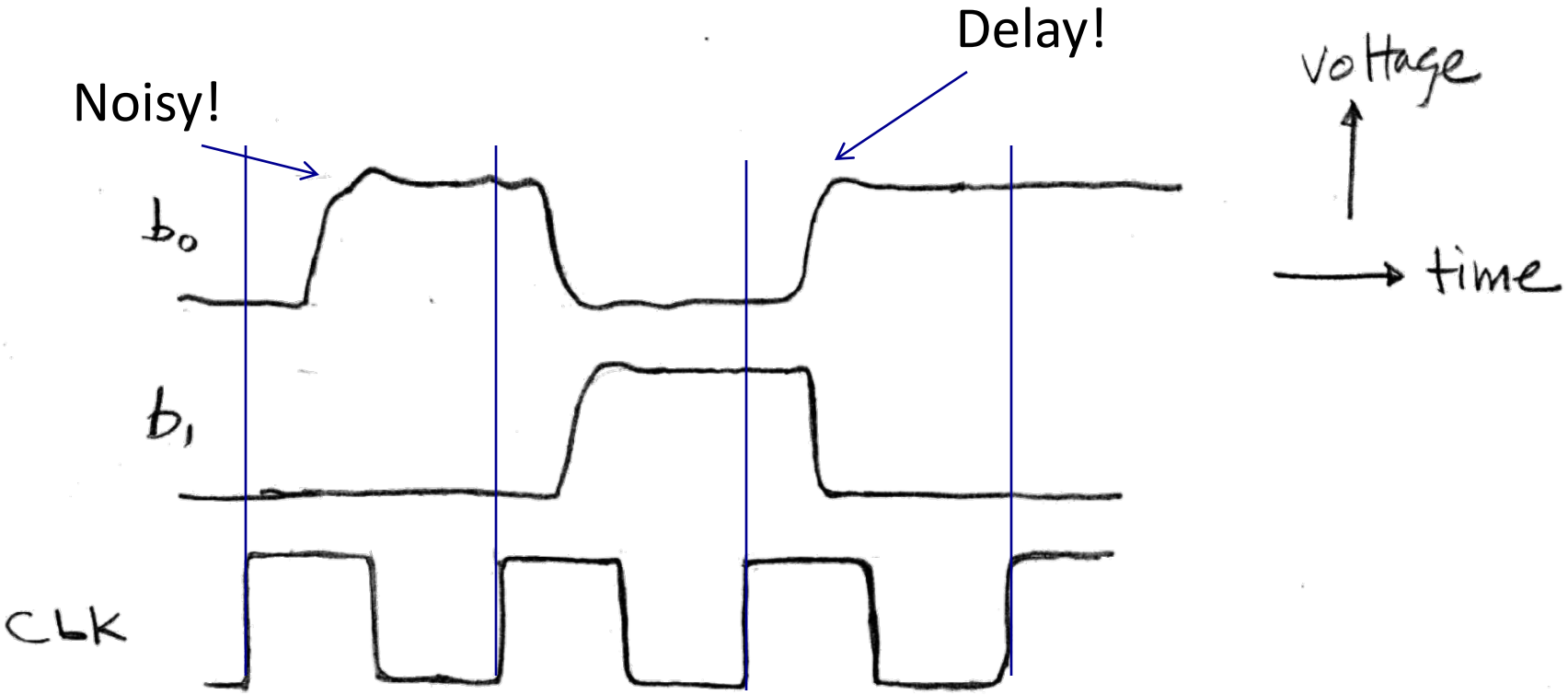
Question

- Simplify $Z = A+BC + \overline{A}(\overline{BC})$

- A: $Z = 0$
- B: $Z = \overline{A(1+ BC)}$
- C: $Z = (A + BC)$
- D: $Z = BC$
- E: $Z = 1$

Signals and Waveforms





Type of Circuits

- *Synchronous Digital Systems* consist of two basic types of circuits:
  - Combinational Logic (CL) circuits
    - Output is a function of the inputs only, not the history of its execution
    - E.g., circuits to add A, B (ALUs)
  - Sequential Logic (SL)
    - Circuits that "remember" or store information
    - aka "State Elements"
    - E.g., memories and registers (Registers)

Uses for State Elements

- Place to store values for later re-use:
  - Register files (like $1-$31 in MIPS)
  - Memory (caches and main memory)
- *Help control flow of information between combinational logic blocks*
  - State elements hold up the movement of information at input to combinational logic blocks to allow for orderly passage

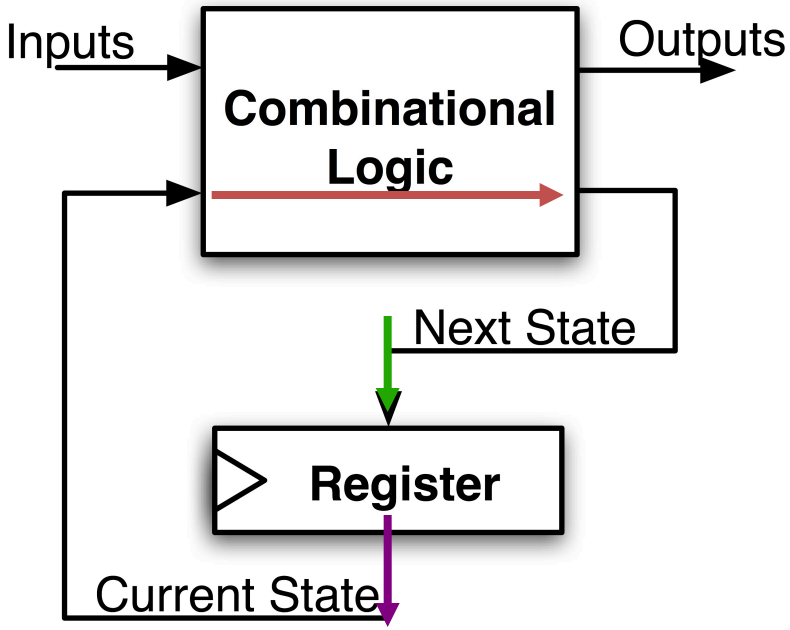
Recap of Timing Terms

- Clock (CLK) - steady square wave that synchronizes system
- Setup Time - when the input must be stable before the rising edge of the CLK
- Hold Time - when the input must be stable after the rising edge of the CLK
- "CLK-to-Q" Delay - how long it takes the output to change, measured from the rising edge of the CLK

- Flip-flop - one bit of state that samples every rising edge of the CLK (positive edge-triggered)
- Register - several bits of state that samples on rising edge of CLK or on LOAD (positive edge-triggered)

Maximum Clock Frequency

- What is the maximum frequency of this circuit?

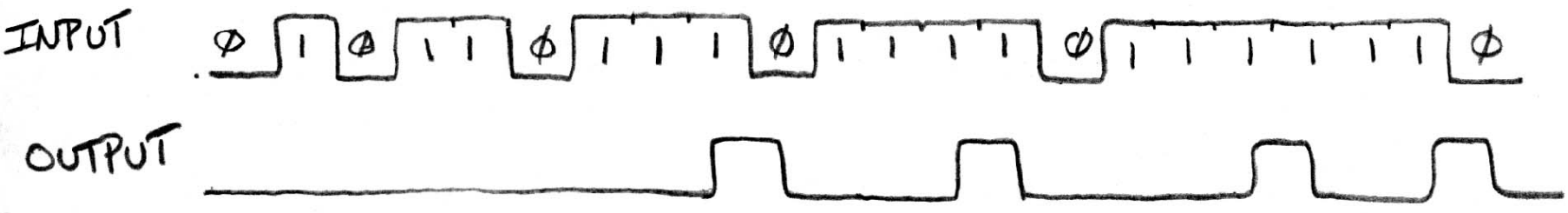


Hint:
Frequency = 1/Period

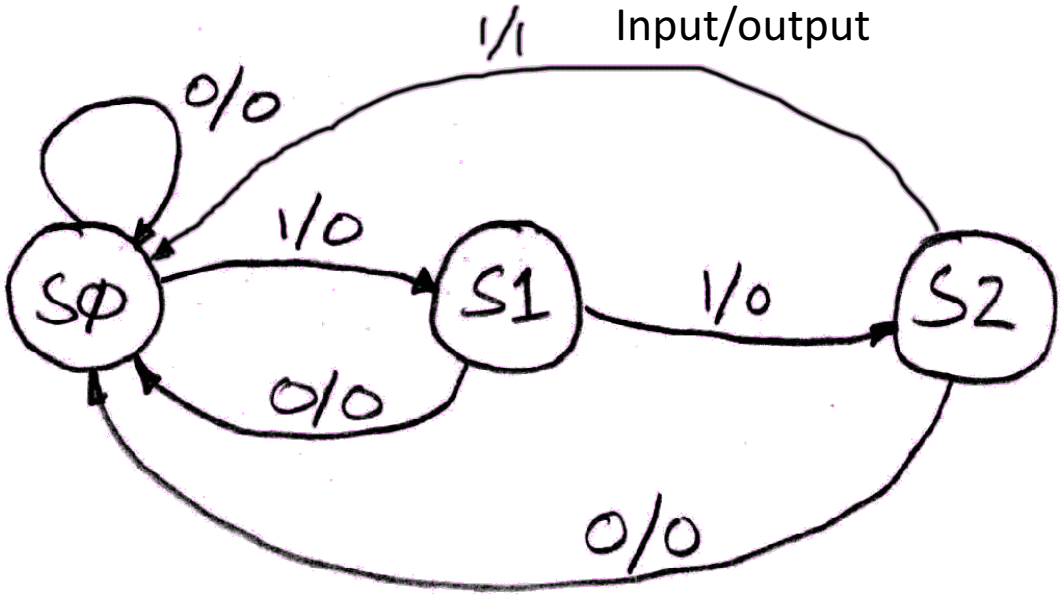
Max Delay = Setup Time + CLK-to-Q Delay + CL Delay

FSM Example: 3 ones…

FSM to detect the occurrence of 3 consecutive 1's in the input.



Draw the FSM…

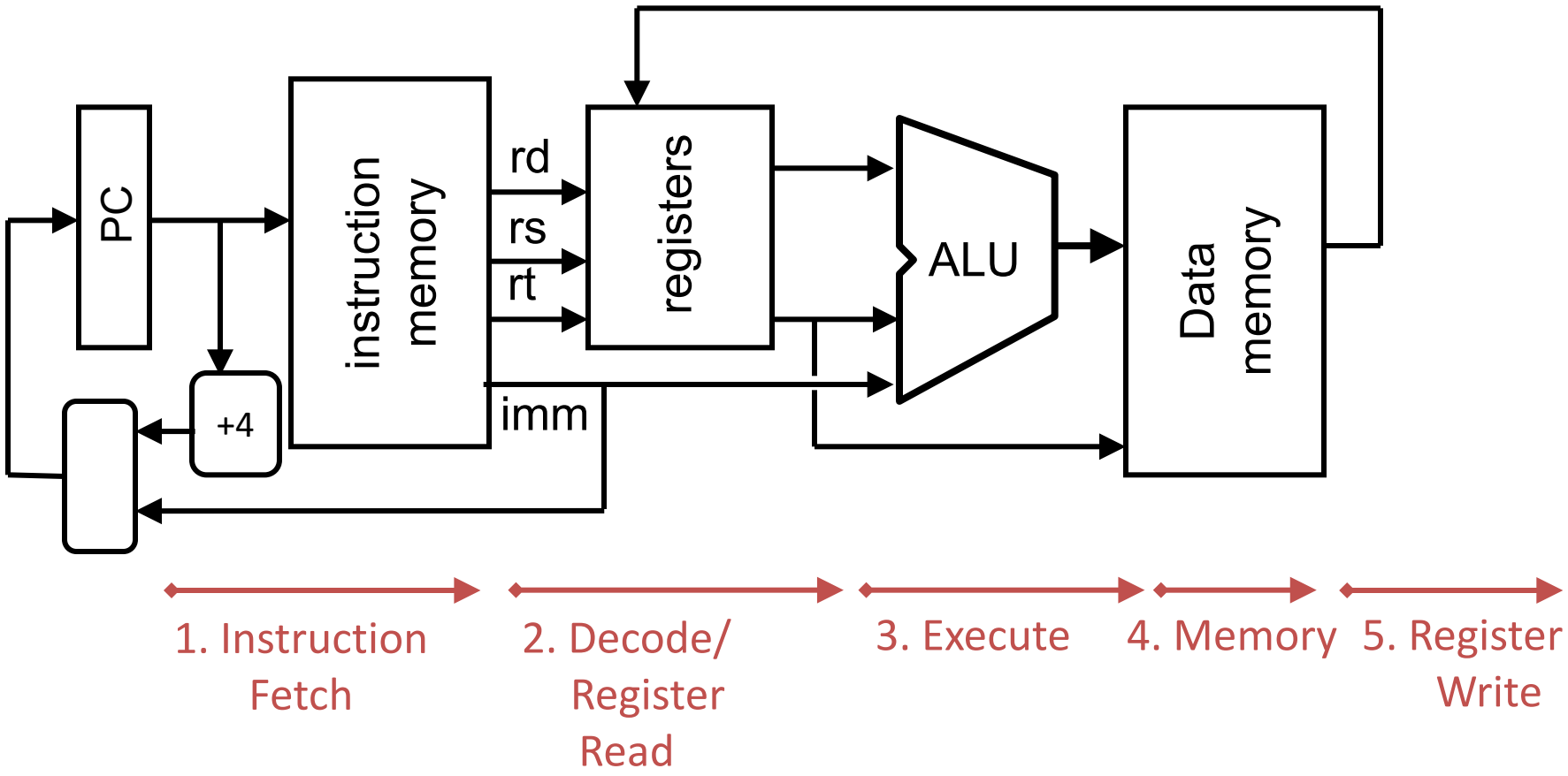


Assume state transitions are controlled by the clock:
on each clock cycle the machine checks the inputs and moves to a new state and produces a new output…

Datapath: Five Stages of Instruction Execution

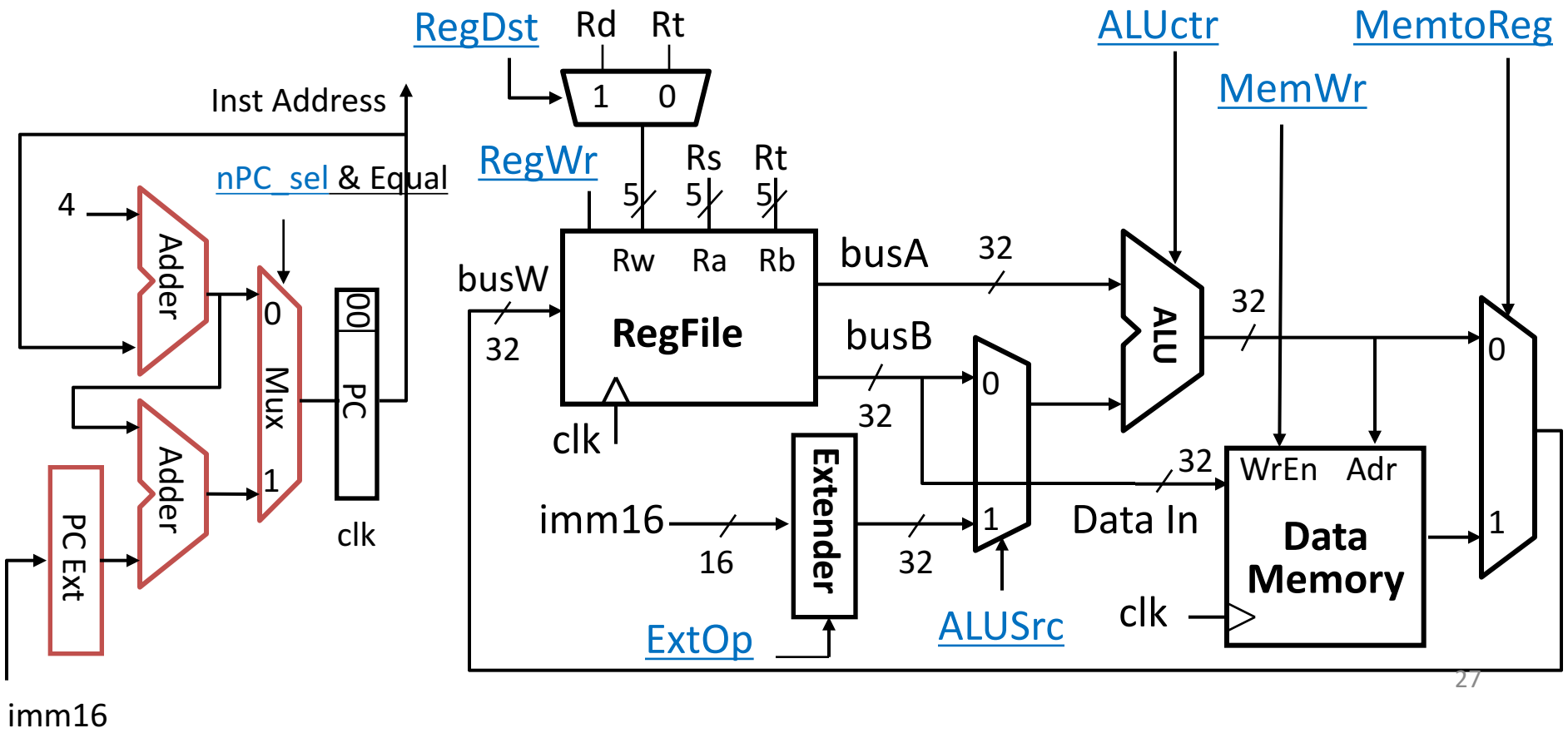
- Stage 1: Instruction Fetch
- Stage 2: Instruction Decode
- Stage 3: ALU (Arithmetic-Logic Unit)
- Stage 4: Memory Access
- Stage 5: Register Write

Stages of Execution on Datapath

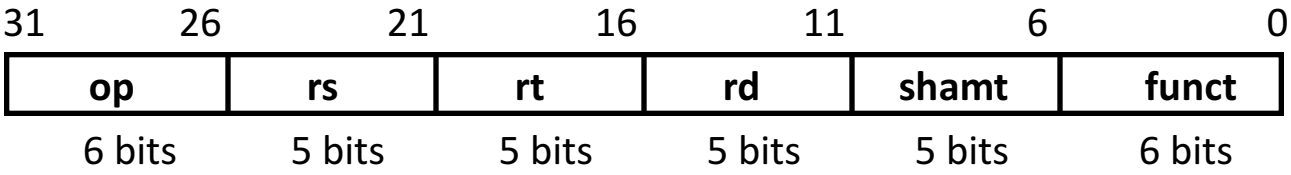


Datapath Control Signals

- ExtOp: "zero", "sign"
- ALUsrc: 0 => regB; 1 => immed
- ALUctr: "ADD", "SUB", "OR"
- nPC_sel: 1 => branch
- MemWr: 1 => write memory
- MemtoReg: 0 => ALU; 1 => Mem
- RegDst: 0 => "rt"; 1 => "rd"
- RegWr: 1 => write register



RTL: The Add Instruction

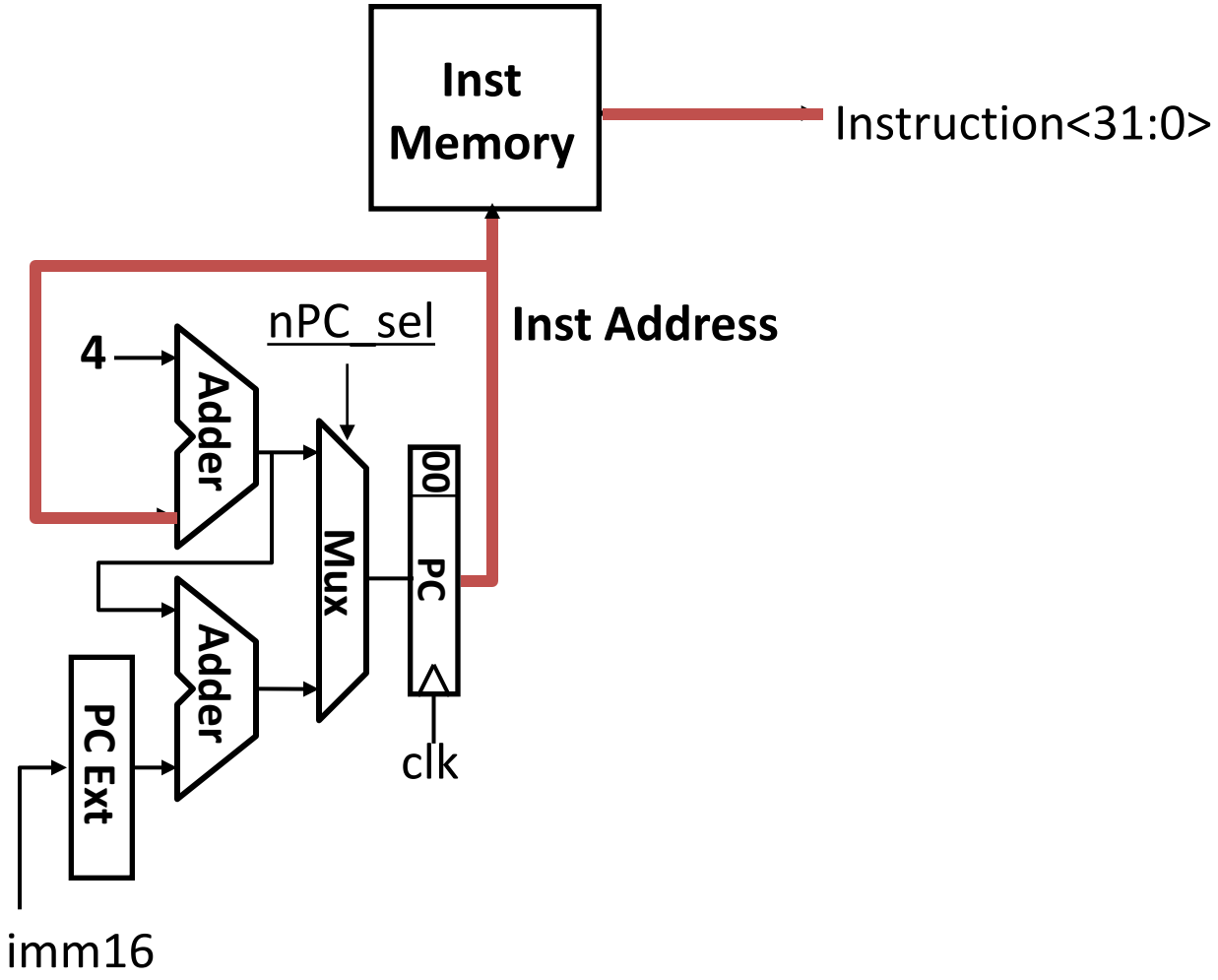


`add rd, rs, rt`

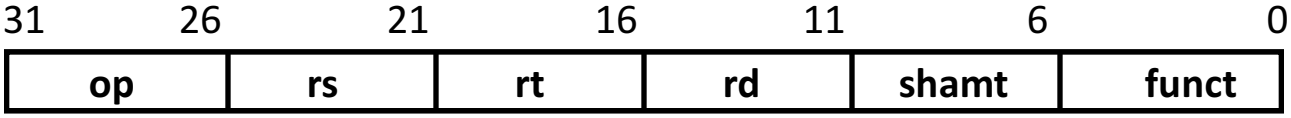
- MEM[PC] Fetch the instruction from memory
- R[rd] = R[rs] + R[rt] The actual operation
- PC = PC + 4 Calculate the next instruction's address

Instruction Fetch Unit at the Beginning of Add

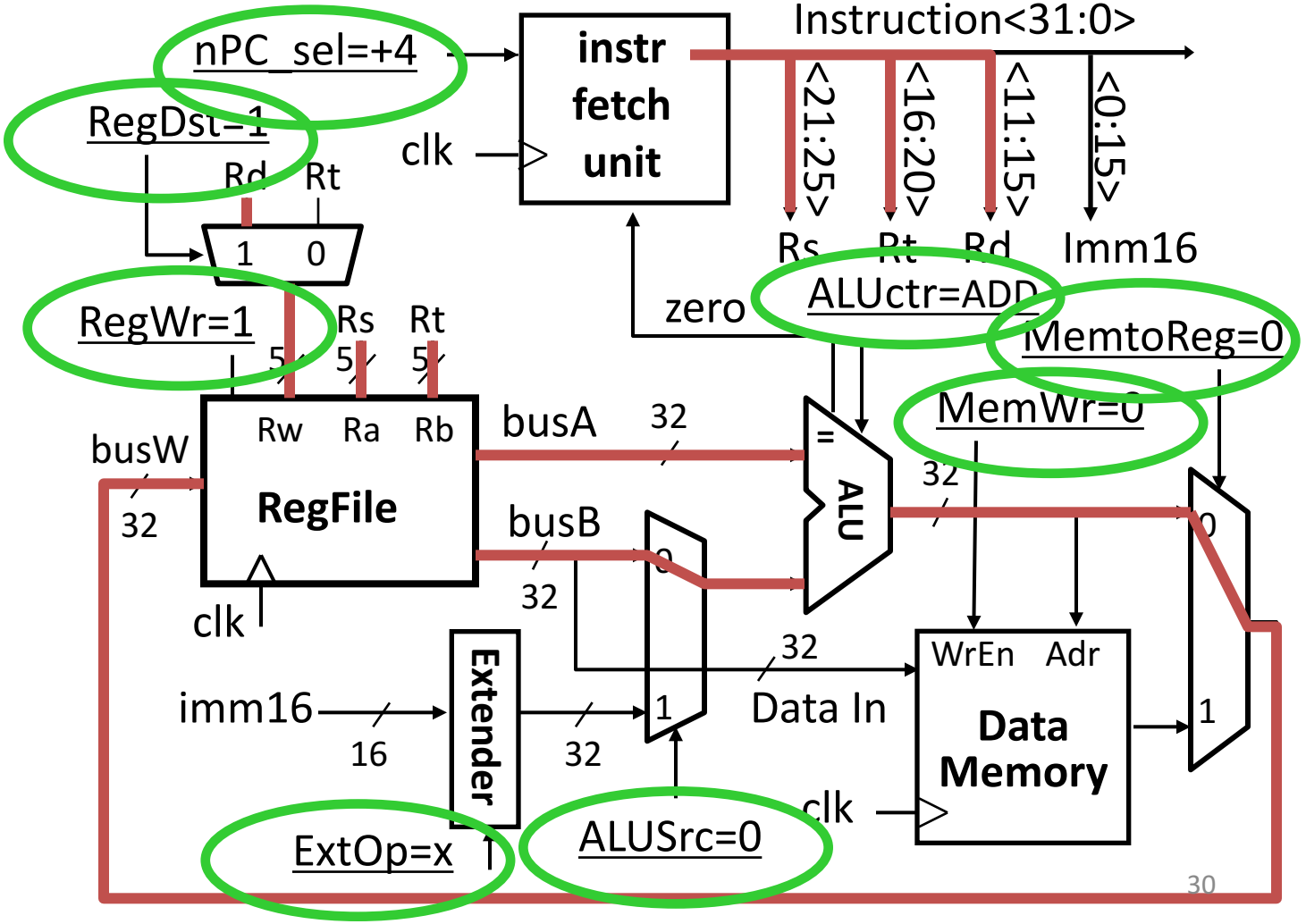
- Fetch the instruction from Instruction memory:
  Instruction = MEM[PC]
  - same for all instructions



Single Cycle Datapath during Add



R[rd] = R[rs] + R[rt]



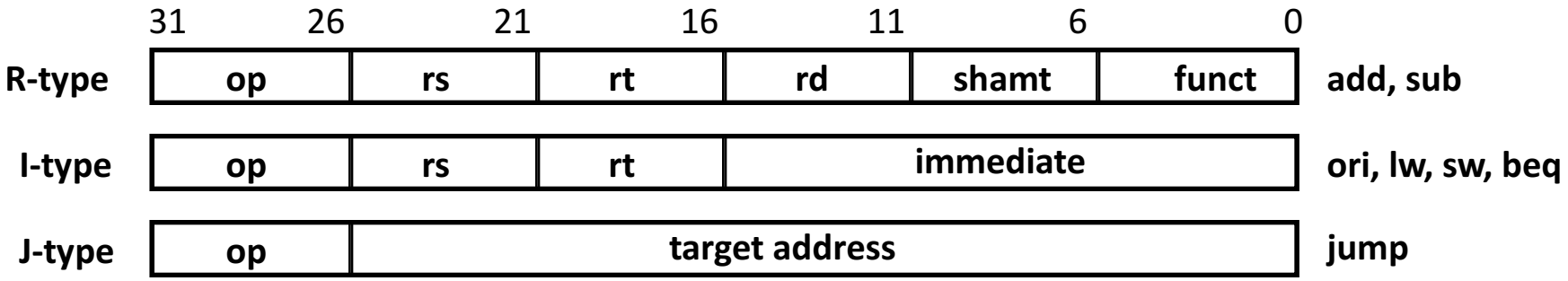
Summary of the Control Signals (1/2)

inst	Register Transfer
add	R[rd] ← R[rs] + R[rt]; PC ← PC + 4
	ALUsrc=RegB, ALUctr="ADD", RegDst=rd, RegWr, nPC_sel="+4"
sub	R[rd] ← R[rs] – R[rt]; PC ← PC + 4
	ALUsrc=RegB, ALUctr="SUB", RegDst=rd, RegWr, nPC_sel="+4"
ori	R[rt] ← R[rs] + zero_ext(Imm16); PC ← PC + 4
	ALUsrc=Im, Extop="Z", ALUctr="OR", RegDst=rt,RegWr, nPC_sel="+4"
lw	R[rt] ← MEM[R[rs] + sign_ext(Imm16)]; PC ← PC + 4
	ALUsrc=Im, Extop="sn", ALUctr="ADD", MemtoReg, RegDst=rt, RegWr, nPC_sel = "+4"
sw	MEM[R[rs] + sign_ext(Imm16)] ← R[rs]; PC ← PC + 4
	ALUsrc=Im, Extop="sn", ALUctr = "ADD", MemWr, nPC_sel = "+4"
beq	if (R[rs] == R[rt]) then PC ← PC + sign_ext(Imm16)] || 00 else PC ← PC + 4
	nPC_sel = "br", ALUctr = "SUB"

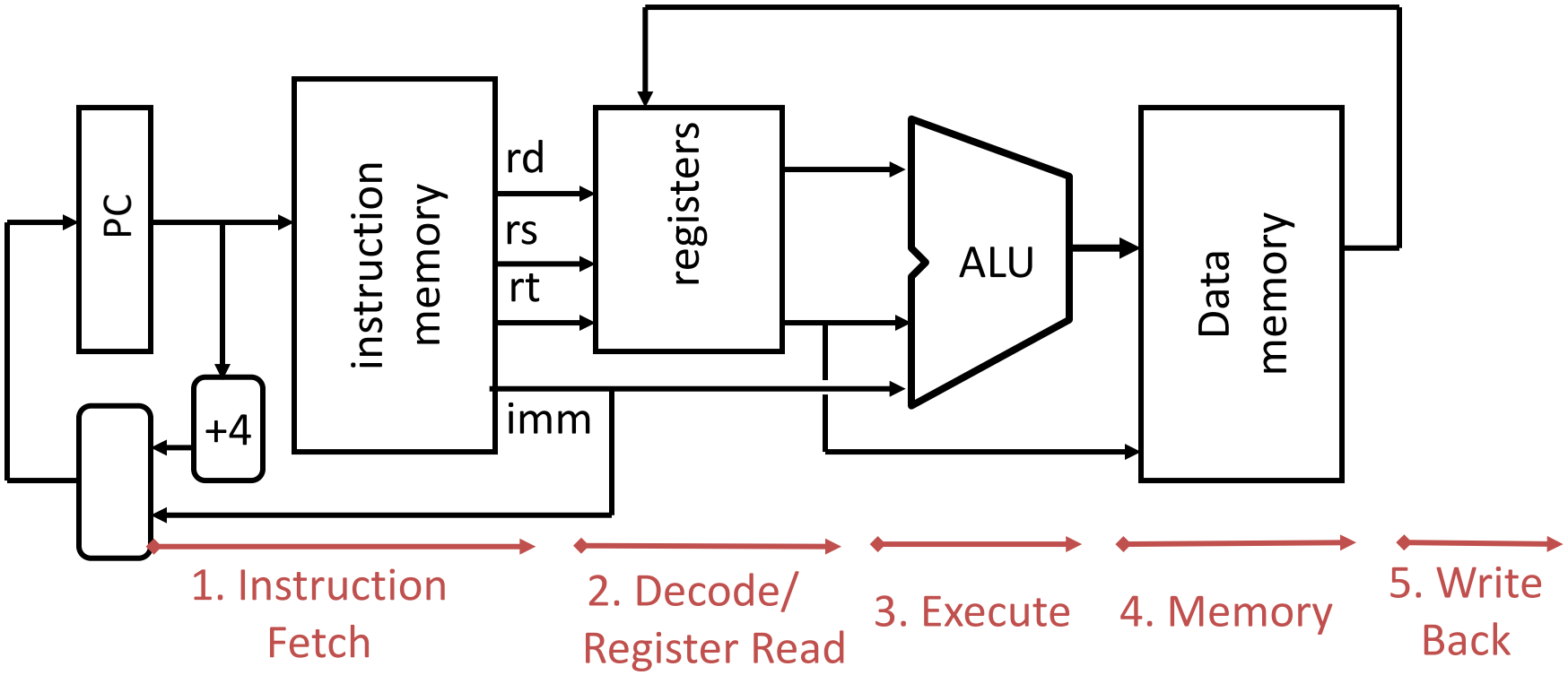
Summary of the Control Signals (2/2)

See Appendix A

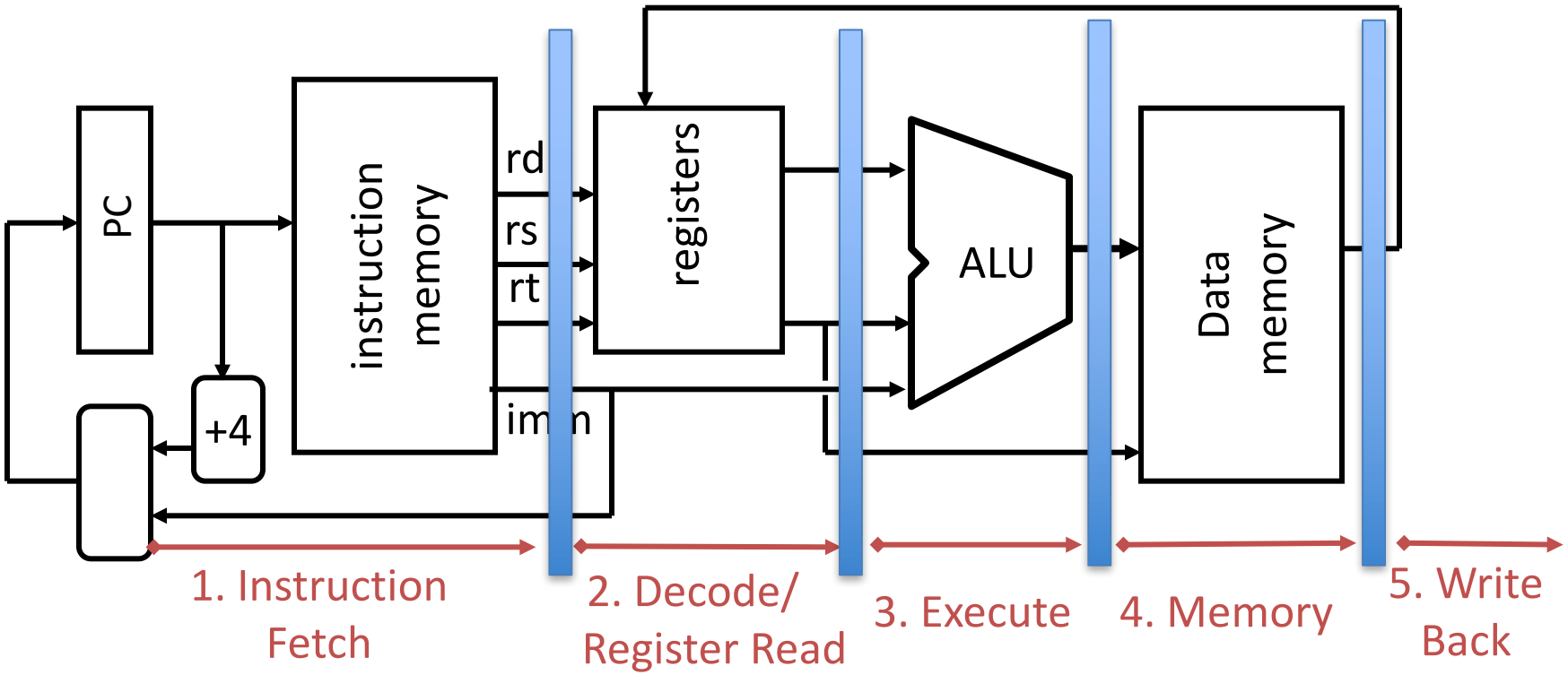
	add	sub	ori	lw	sw	beq	jump
func	10 0000	10 0010	**We Don't Care :-)**				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
nPCsel	0	0	0	0	0	1	x
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	x



Pipelining: Single Cycle Datapath



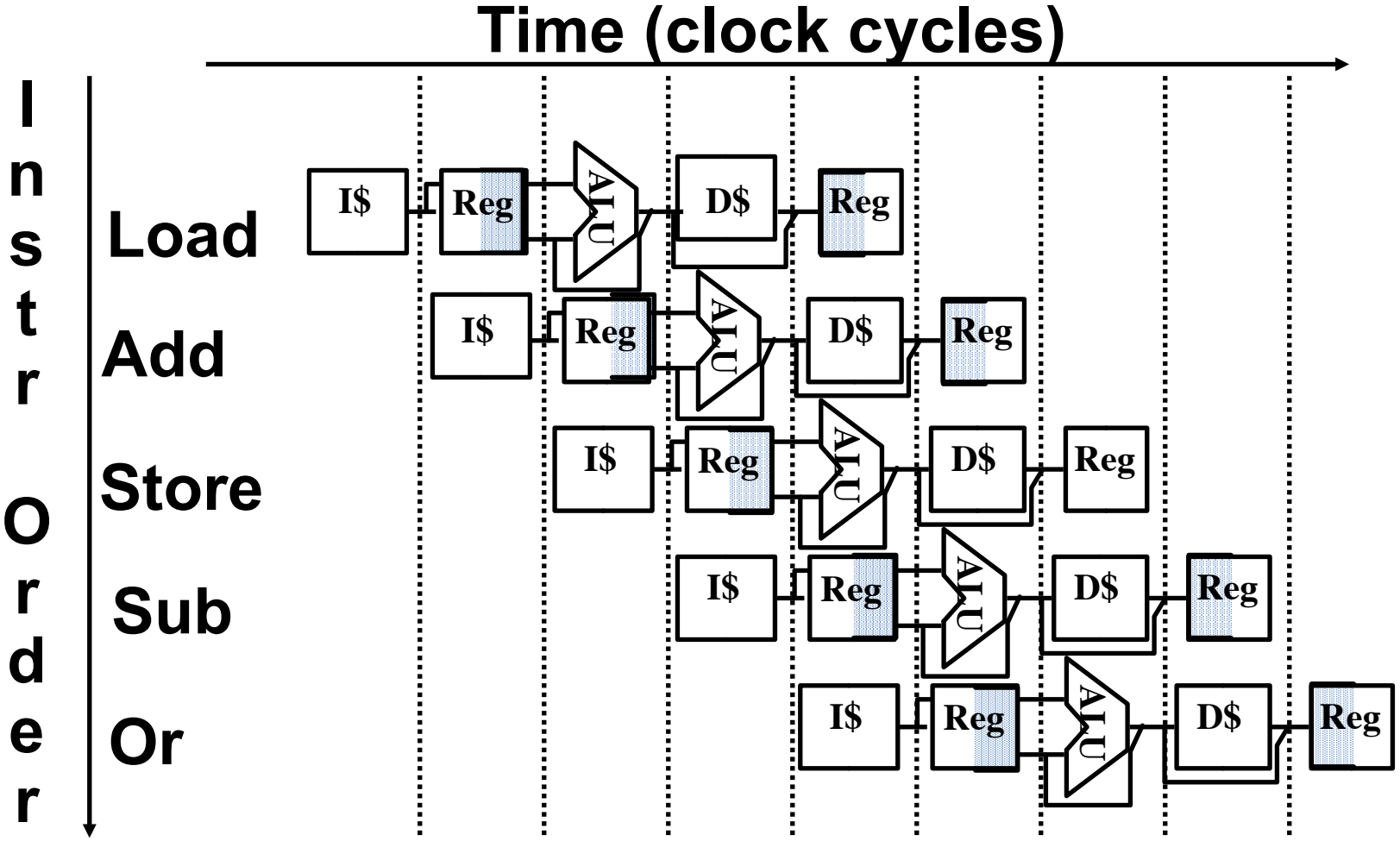
Pipeline registers



- Need registers between stages
  - To hold information produced in previous cycle

Graphical Pipeline Representation

- RegFile: left half is write, right half is read

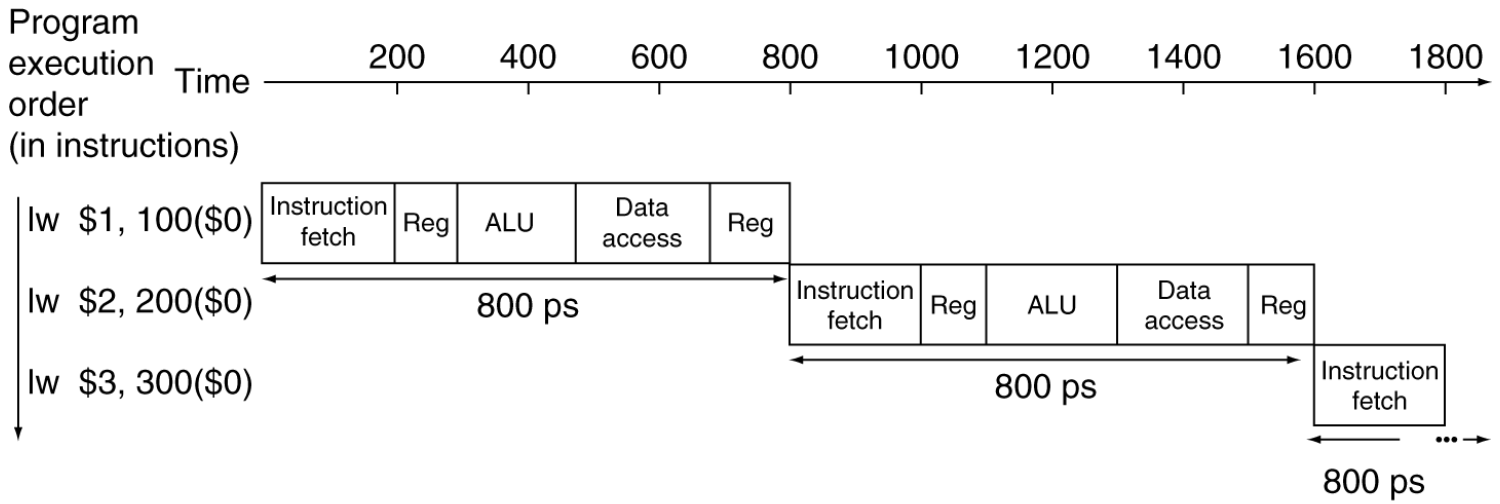


Pipelining Performance (3/3)

Single-cycle

T_c = 800 ps

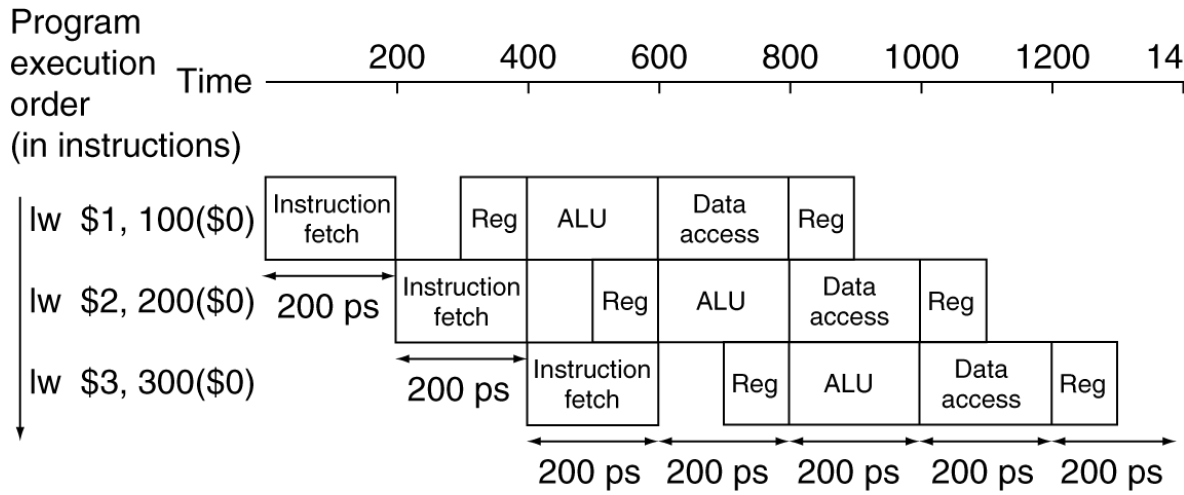
f = 1.25GHz



Pipelined

T_c = 200 ps

f = 5GHz

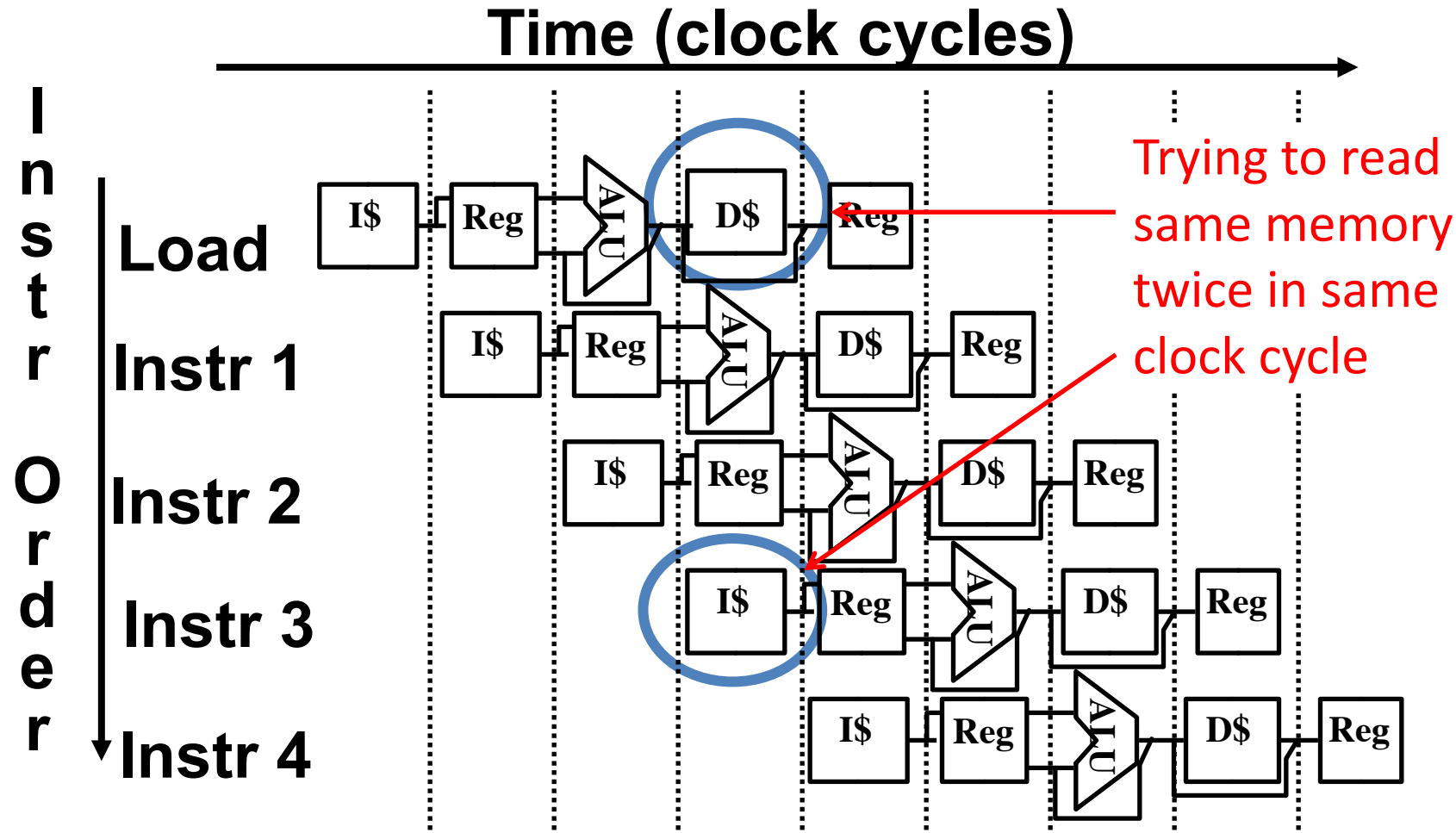


Pipelining Hazards

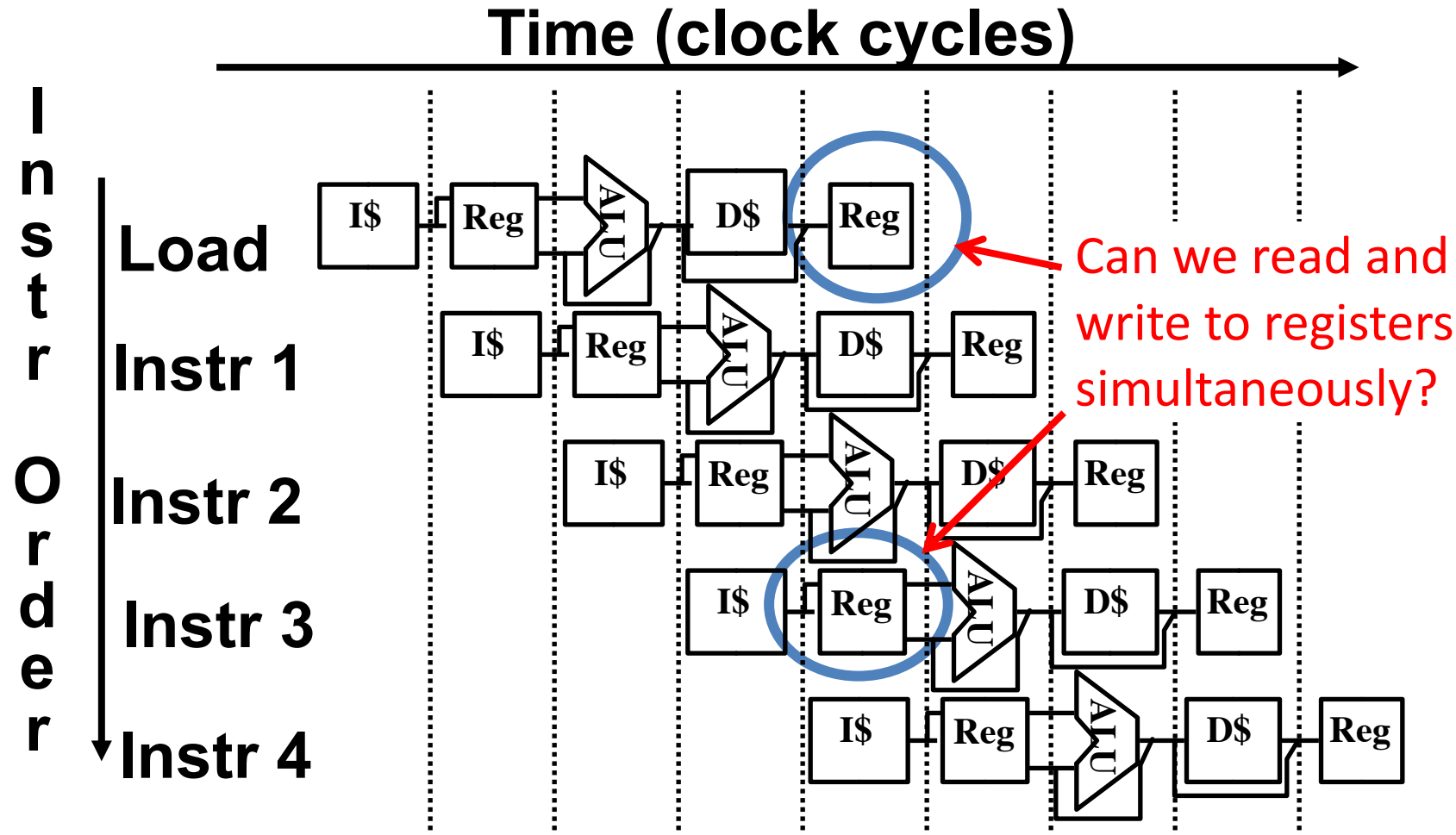
A *hazard* is a situation that prevents starting the next instruction in the next clock cycle

1) *Structural hazard*
   - A required resource is busy (e.g. needed in multiple stages)
2) *Data hazard*
   - Data dependency between instructions
   - Need to wait for previous instruction to complete its data read/write
3) *Control hazard*
   - Flow of execution depends on previous instruction

Structural Hazard #1: Single Memory

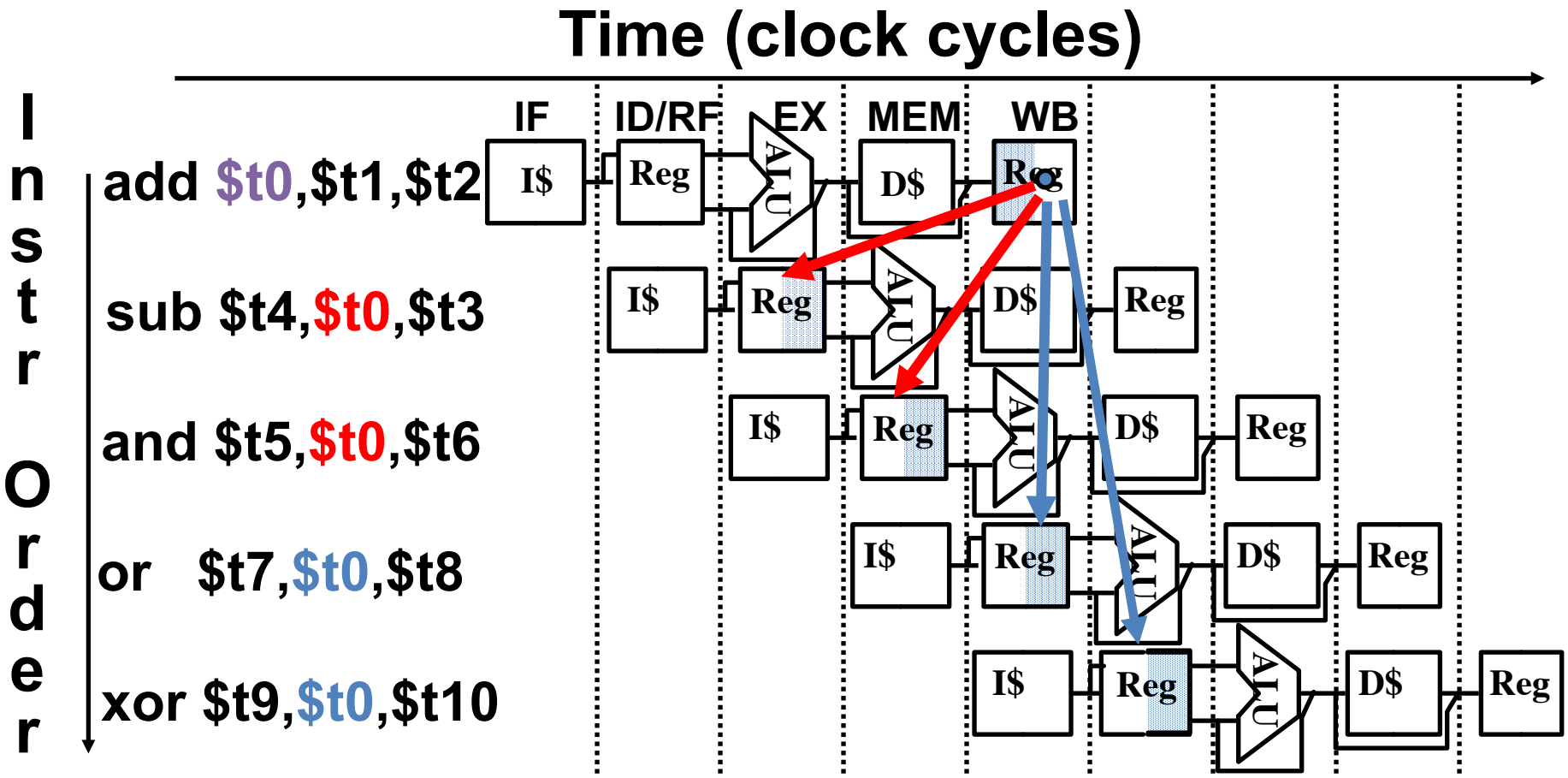


Structural Hazard #2: Registers (1/2)



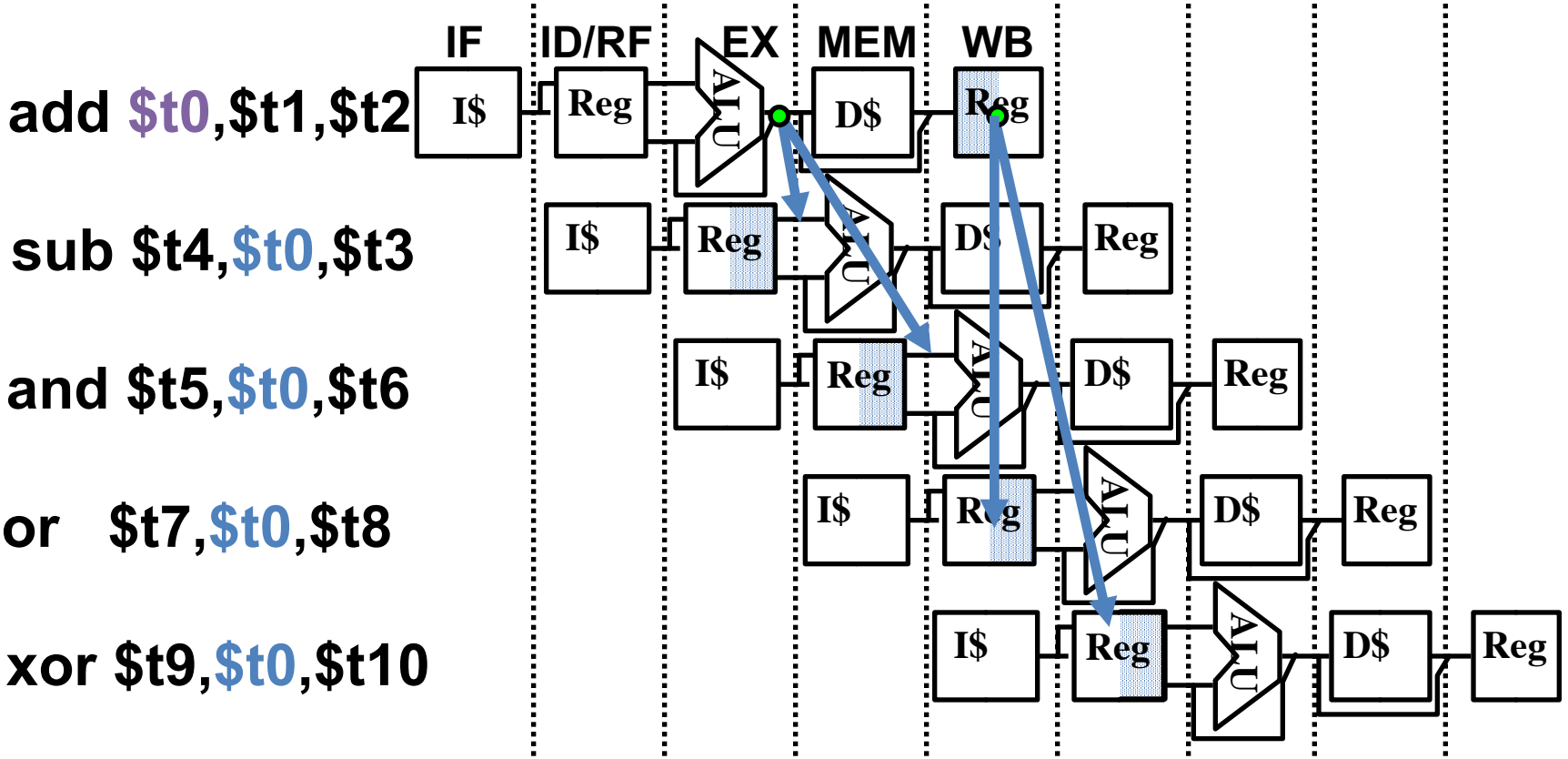
2. Data Hazards (2/2)

- Data-flow *backwards* in time are hazards



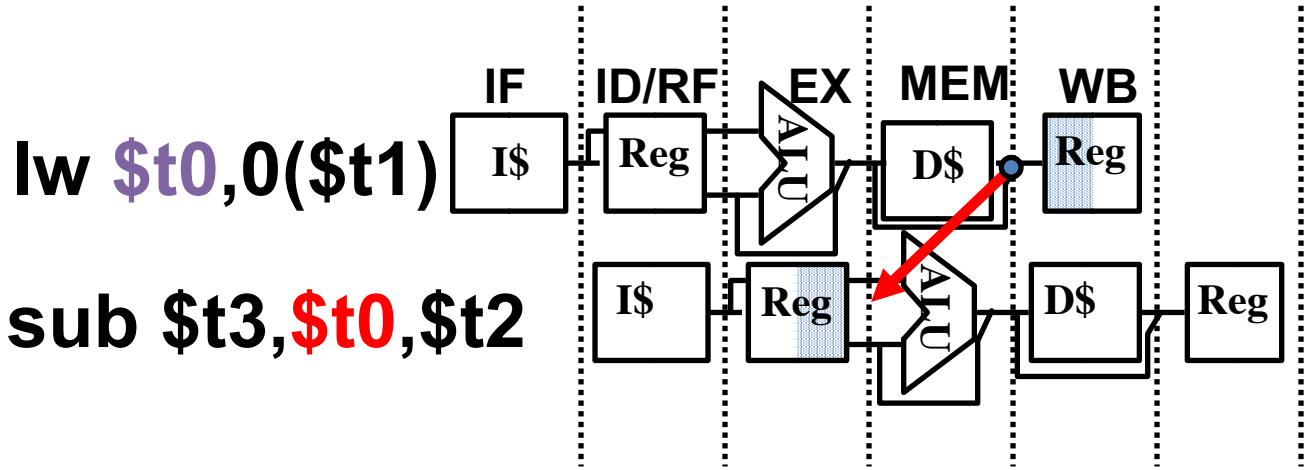
Data Hazard Solution: Forwarding

- Forward result as soon as it is available
  - OK that it's not stored in RegFile yet



Data Hazard: Loads (1/4)

- **Recall:** Dataflow backwards in time are hazards



- Can't solve all cases with forwarding
  - Must *stall* instruction dependent on load, then forward (more hardware)

Data Hazard: Loads (2/4)

- *Hardware* stalls pipeline
  - Called "hardware interlock"

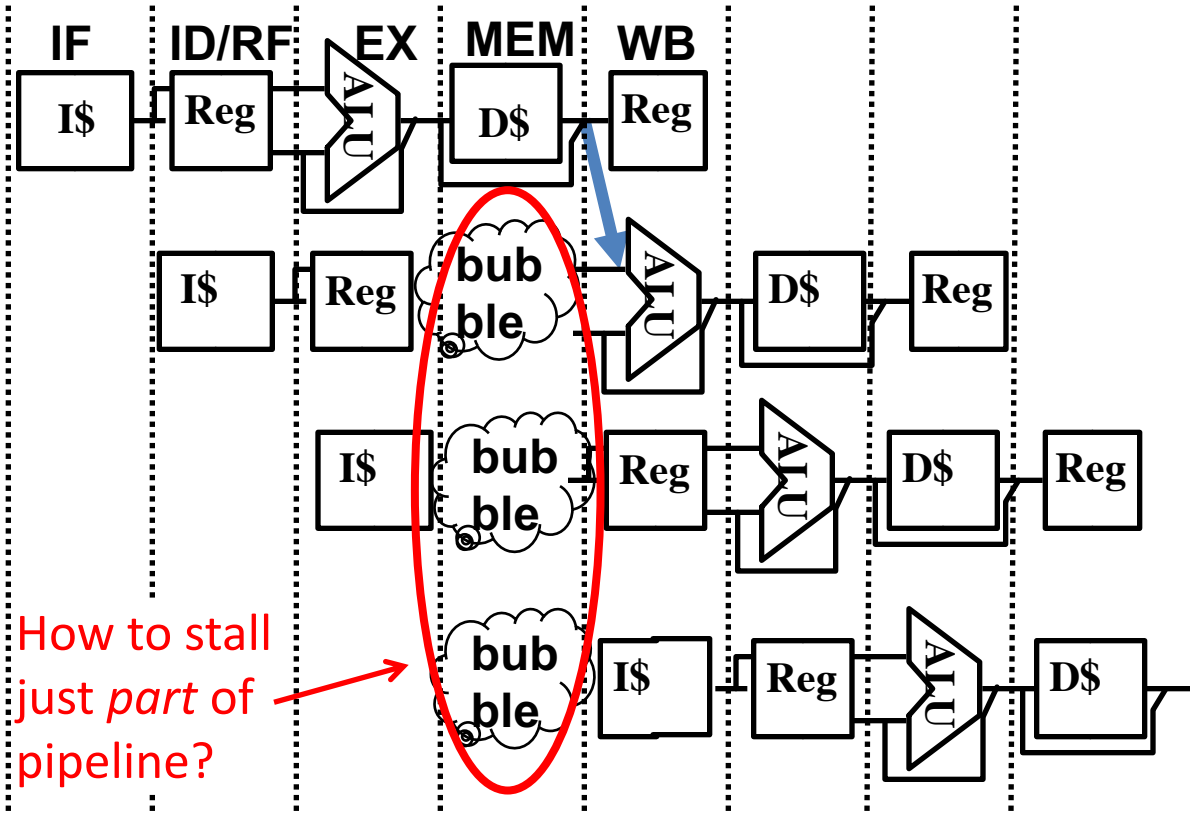
Schematically, this is what we want, but in reality stalls done "horizontally"

lw $t0, 0($t1)

sub $t3,$t0,$t2

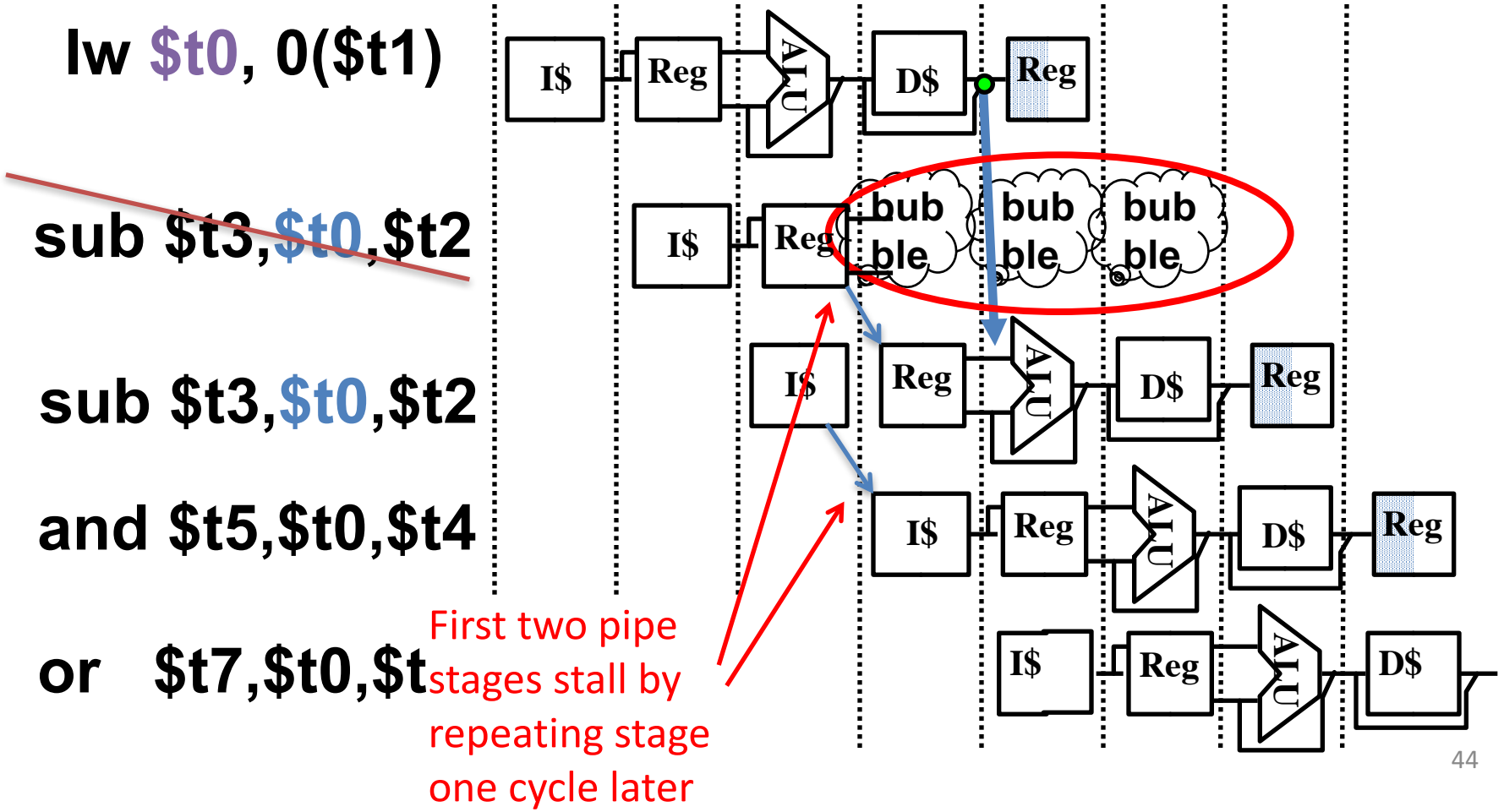
and $t5,$t0,$t4

or $t7,$t0,$t6



Data Hazard: Loads (3/4)

- Stalled instruction converted to “bubble”, acts like nop

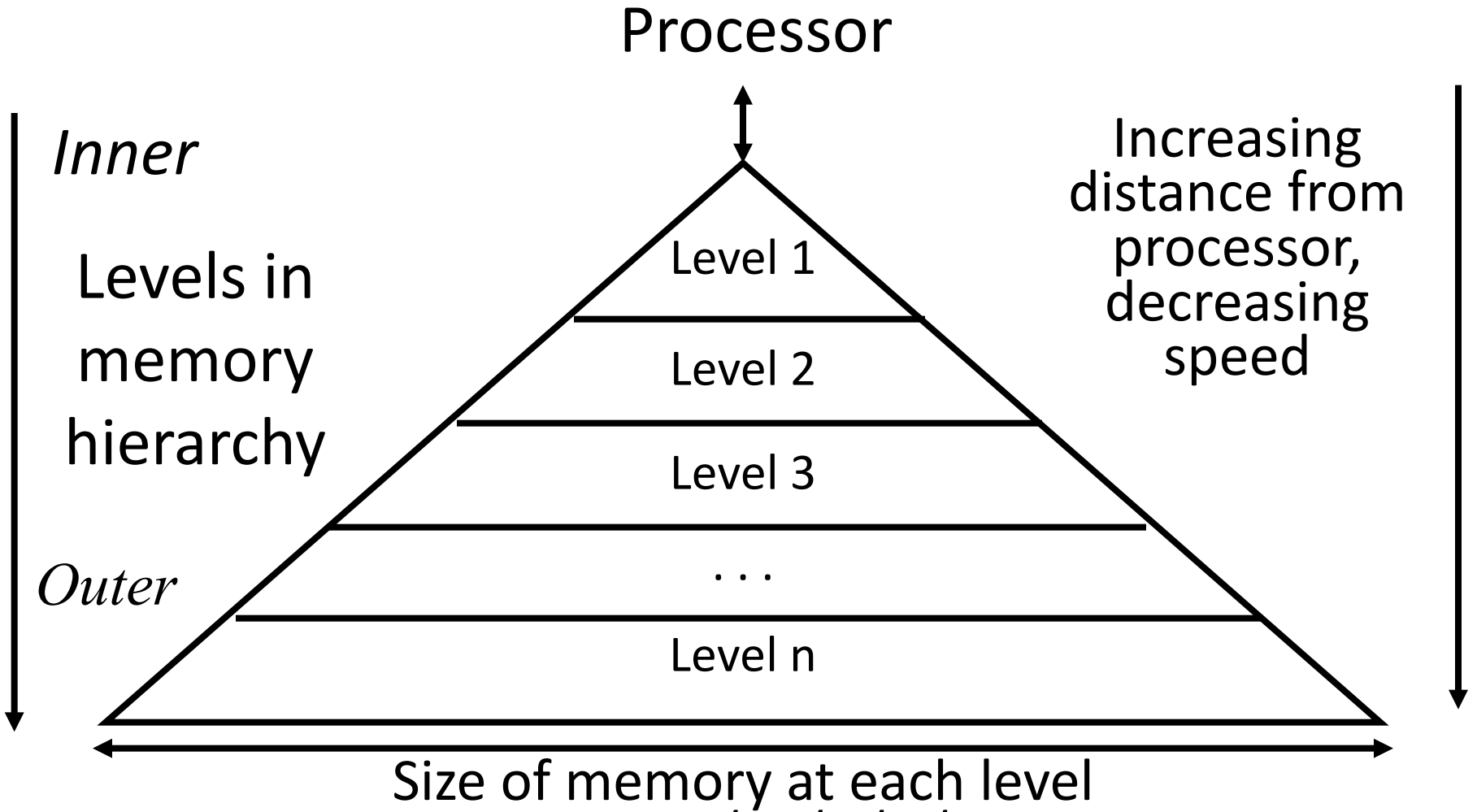


3. Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Still working on ID stage of branch
- BEQ, BNE in MIPS pipeline
- Simple solution Option 1: *Stall* on every branch until branch condition resolved
  - Would add 2 bubbles/clock cycles for every Branch! (~ 20% of instructions executed)

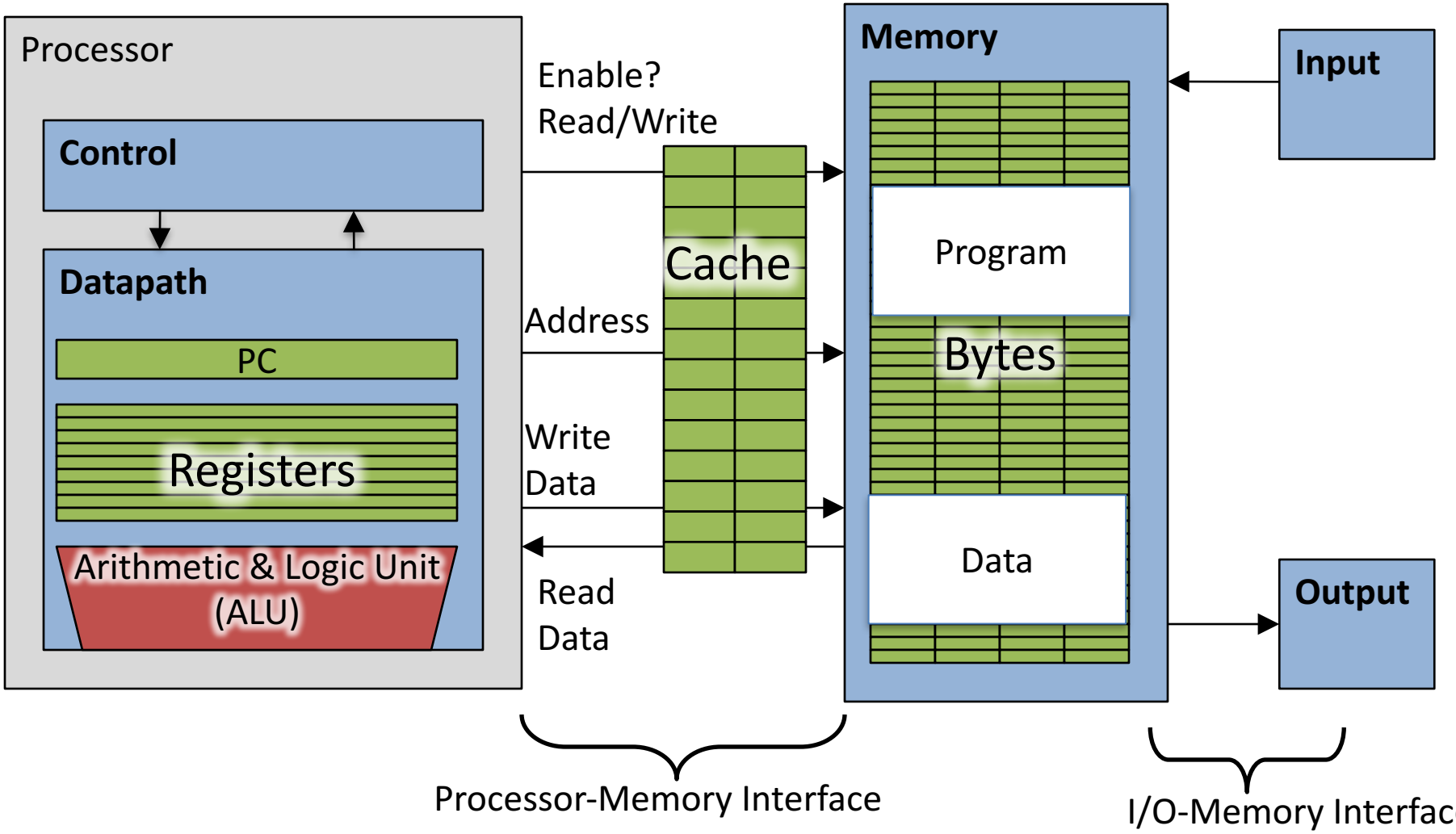
Caches

Big Idea: Memory Hierarchy



As we move to outer levels the latency goes up and price per bit goes down. Why?

Adding Cache to Computer



Total Cash Capacity =

Associativity * # of sets * block_size

*Bytes = blocks/set * sets * Bytes/block*

*C = N * S * B*

address_size = tag_size + index_size + offset_size
= tag_size + log2(S) + log2(B)

Clicker Question: C remains constant, S and/or B can change such that
C = 2N * (SB)' => (SB)' = SB/2
Tag_size = address_size – (log2(S) + log2(B)) = address_size – log2(SB)
= address_size – (log2(SB) – 1)

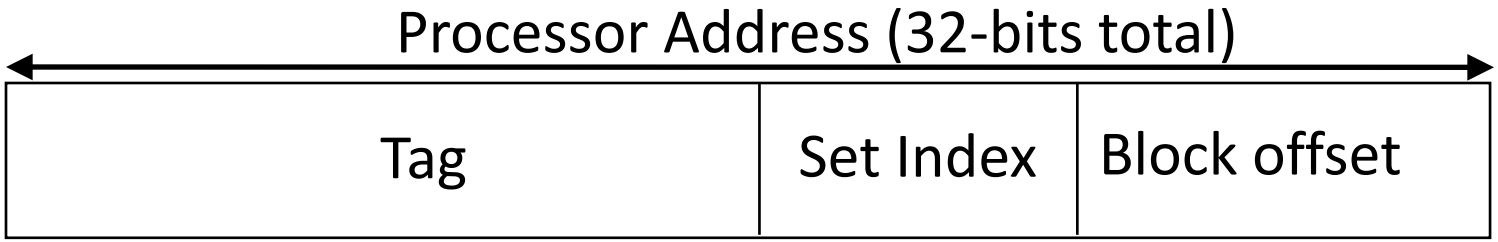
- Principle of Locality for Libraries /Computer Memory
- Hierarchy of Memories (speed/size/cost per bit) to Exploit Locality
- Cache – copy of data lower level in memory hierarchy
- Direct Mapped to find block in cache using Tag field and Valid bit for Hit
- Cache design choice:
  - Write-Through vs. Write-Back

Cache Organizations

- "Fully Associative": Block can go anywhere
  - First design in lecture
  - Note: No Index field, but 1 comparator/block
- "Direct Mapped": Block goes one place
  - Note: Only 1 comparator
  - Number of sets = number blocks
- "N-way Set Associative": N places for a block
  - Number of sets = number of blocks / N
  - N comparators
  - ***Fully Associative: N = number of blocks***
  - ***Direct Mapped: N = 1***

Processor Address Fields used by Cache Controller

- Block Offset: Byte address within block
- Set Index: Selects which set
- Tag: Remaining portion of processor address



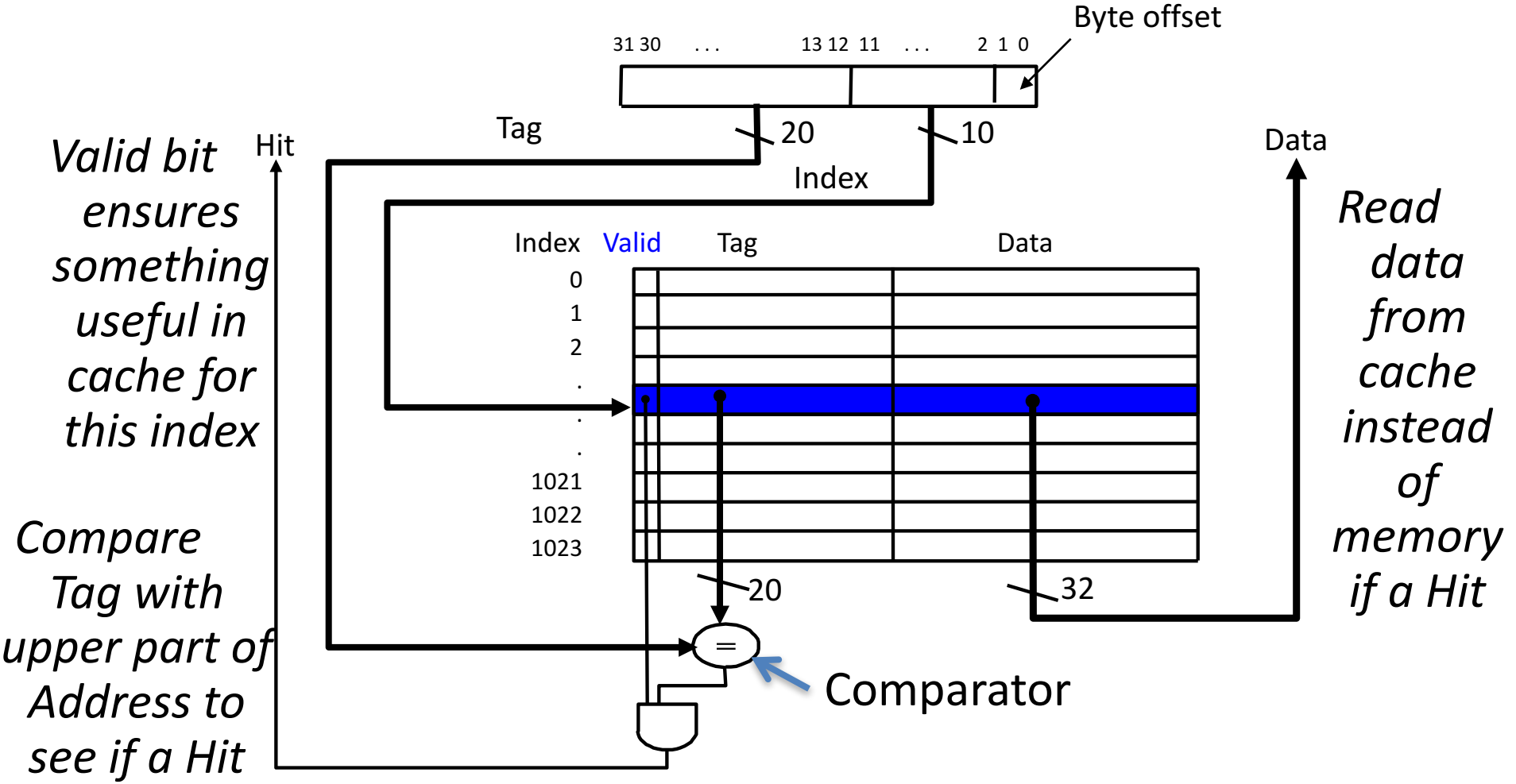
- Size of Index = log2 (number of sets)
- Size of Tag = Address size – Size of Index – log2 (number of bytes/block)

Write Policy Choices

- Cache hit:
  - **write through**: writes both cache & memory on every access
    - Generally higher memory traffic but simpler pipeline & cache design
  - **write back**: writes cache only, memory `written only when dirty entry evicted
    - A dirty bit per line reduces write-back traffic
    - Must handle 0, 1, or 2 accesses to memory for each load/store
- Cache miss:
  - **no write allocate**: only write to main memory
  - **write allocate** (aka fetch on write): fetch into cache
- Common combinations:
  - write through and no write allocate
  - write back with write allocate

Direct-Mapped Cache Review

- One word blocks, cache size = 1K words (or 4KB)



Sources of Cache Misses (3 C's)

- *Compulsory* (cold start, first reference):
  - 1st access to a block, "cold" fact of life, not a lot you can do about it.
    - If running billions of instructions, compulsory misses are insignificant
- *Capacity*:
  - Cache cannot contain all blocks accessed by the program
    - Misses that would not occur with infinite cache
- *Conflict* (collision):
  - Multiple memory locations mapped to same cache set
    - Misses that would not occur with ideal fully associative cache

Impact of Cache Parameters on Performance

- AMAT = Hit Time + Miss Rate * Miss Penalty
  - Note, we assume always first search cache, so must charge hit time for both hits and misses!
- For misses, characterize by 3Cs

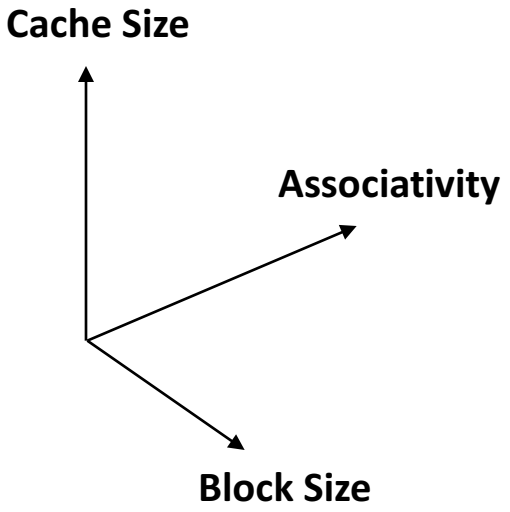
Local vs. Global Miss Rates

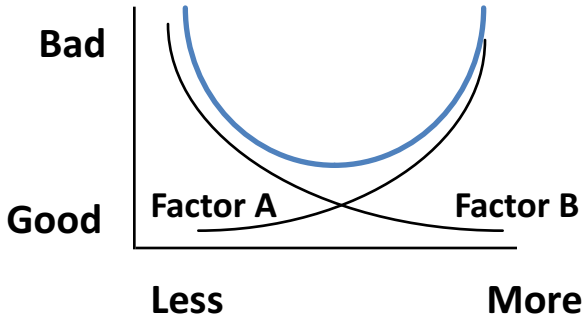
- *Local miss rate* – the fraction of references to one level of a cache that miss
- Local Miss rate L2$ = $L2 Misses / L1$ Misses
- *Global miss rate* – the fraction of references that miss in all levels of a multilevel cache
  - L2$ local miss rate >> than the global miss rate
- Global Miss rate = L2$ Misses / Total Accesses
  = (L2$ Misses / L1$ Misses) × (L1$ Misses / Total Accesses)
  = Local Miss rate L2$ × Local Miss rate L1$

- AMAT = Time for a hit + Miss rate × Miss penalty
- AMAT = Time for a L1$ hit + (local) Miss rate L1$ × (Time for a L2$ hit + (local) Miss rate L2$ × L2$ Miss penalty)

In Conclusion, Cache Design Space

- Several interacting dimensions
  - Cache size
  - Block size
  - Associativity
  - Replacement policy
  - Write-through vs. write-back
  - Write-allocation
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload
    - Use (I-cache, D-cache)
  - Depends on technology / cost
- Simplicity often wins



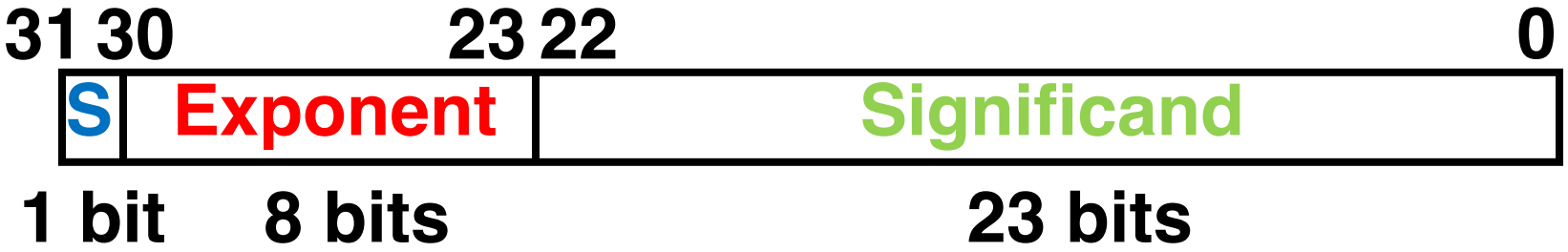


Iron Law of Performance

- A program executes instructions
- `CPU Time/Program`
  `= Clock Cycles/Program x Clock Cycle Time`
  `= Instructions/Program`
  `x Average Clock Cycles/Instruction`
  `x Clock Cycle Time`
- 1st term called *Instruction Count*
- 2nd term abbreviated *CPI* for average ***C****lock Cycles* ***P****er* ***I****nstruction*
- 3rd term is 1 / Clock rate

IEEE 754 Floating-Point Standard (1/3)

Single Precision (Double Precision similar):



- Sign bit: 1 means negative 0 means positive
- Significand in *sign-magnitude* format (not 2's complement)
  - To pack more bits, leading 1 implicit for normalized numbers
  - 1 + 23 bits single, 1 + 52 bits double
  - always true: 0 < Significand < 1 (for normalized numbers)
- Note: 0 has no leading 1, so reserve exponent value 0 just for number 0

IEEE 754 Floating Point Standard (2/3)

- IEEE 754 uses "biased exponent" representation
  - Designers wanted FP numbers to be used even if no FP hardware; e.g., sort records with FP numbers using integer compares
  - Wanted bigger (integer) exponent field to represent bigger numbers
  - 2's complement poses a problem (because negative numbers look bigger)
    - Use just magnitude and offset by half the range

IEEE 754 Floating Point Standard (3/3)

- Called Biased Notation, where bias is number subtracted to get final number
  - IEEE 754 uses bias of 127 for single prec.
  - Subtract 127 from Exponent field to get actual value for exponent

- **Summary (single precision):**

- **$(-1)^S$ x (1 + Significand) x $2^{(Exponent-127)}$**
  - Double precision identical, except with exponent bias of 1023 (half, quad similar)

Flynn* Taxonomy, 1966

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Xeon e5345 (Clovertown)

- Since about 2013, SIMD and MIMD most common parallelism in architectures – usually both in same system!
- Most common parallel processing programming style: Single Program Multiple Data ("SPMD")
  - Single program that runs on all processors of a MIMD
  - Cross-processor execution coordination using synchronization primitives
- SIMD (aka hw-level *data parallelism*): specialized function units, for handling lock-step calculations involving arrays
  - Scientific computing, signal processing, multimedia (audio/video processing)

*Prof. Michael Flynn, Stanford

Big Idea: Amdahl's Law

$$\text{Speedup} = \frac{1}{(1 - F) + \frac{F}{S}}$$

Non-speed-up part

Speed-up part

Example: the execution time of half of the program can be accelerated by a factor of 2.
What is the program speed-up overall?

$$\frac{1}{0.5 + \frac{0.5}{2}} = \frac{1}{0.5 + 0.25} = 1.33$$

Strong and Weak Scaling

- To get good speedup on a parallel processor while keeping the problem size fixed is harder than getting good speedup by increasing the size of the problem.
  - *Strong scaling*: when speedup can be achieved on a parallel processor without increasing the size of the problem
  - *Weak scaling*: when speedup is achieved on a parallel processor by increasing the size of the problem proportionally to the increase in the number of processors
- Load balancing is another important factor: every processor doing same amount of work
  - Just one unit with twice the load of others cuts speedup almost in half

Data Level Parallelism

- Loop Unrolling
- Intel SSE SIMD Instructions
  - Exploit data-level parallelism in loops
  - One instruction fetch that operates on multiple operands simultaneously
  - 128-bit XMM registers
- SSE Instructions in C using Intrinsics
  - Embed the SSE machine instructions directly into C programs through use of intrinsics
  - Achieve efficiency beyond that of optimizing compiler

CS 110
Computer Architecture

Thread-Level Parallelism (TLP) and OpenMP Intro

Instructor:
Sören Schwertfeger

http://shtech.org/courses/ca/

School of Information Science and Technology SIST

ShanghaiTech University

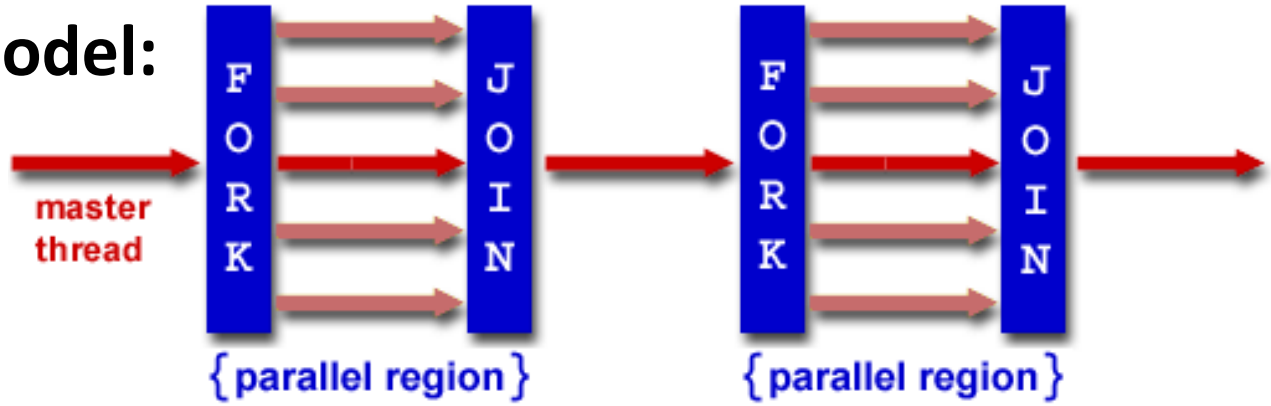
Slides based on UC Berkley's CS61C

Review

- Sequential software is slow software
  - SIMD and MIMD are paths to higher performance
- MIMD thru: multithreading processor cores (increases utilization), Multicore processors (more cores per chip)
- Synchronization – coordination among threads
  - MIPS: atomic read-modify-write using load-linked/store-conditional
- OpenMP as simple parallel extension to C
  - Pragmas for forking multiple Threads
  - ≈ C: small so easy to learn, but not very high level and it's easy to get into trouble

OpenMP Programming Model - Review

- **Fork - Join Model:**



- OpenMP programs begin as single process (*master thread*) and executes sequentially until the first parallel region construct is encountered
  - *FORK:* Master thread then creates a team of parallel threads
  - Statements in program that are enclosed by the parallel region construct are executed in parallel among the various threads
  - *JOIN:* When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master thread

`parallel` Pragma and Scope - Review

- Basic OpenMP construct for parallelization:

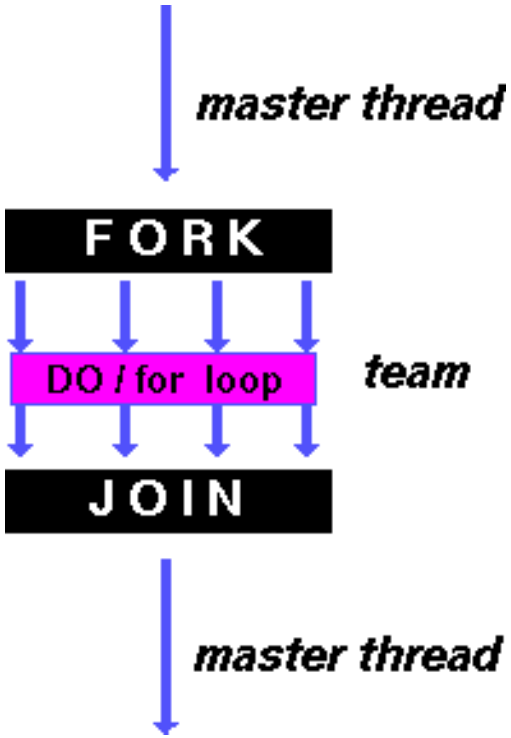
```
#pragma omp parallel
{
   /* code goes here */
}
```

  - *Each* thread runs a copy of code within the block
  - Thread scheduling is *non-deterministic*
- OpenMP default is *shared* variables
  - To make private, need to declare with pragma:

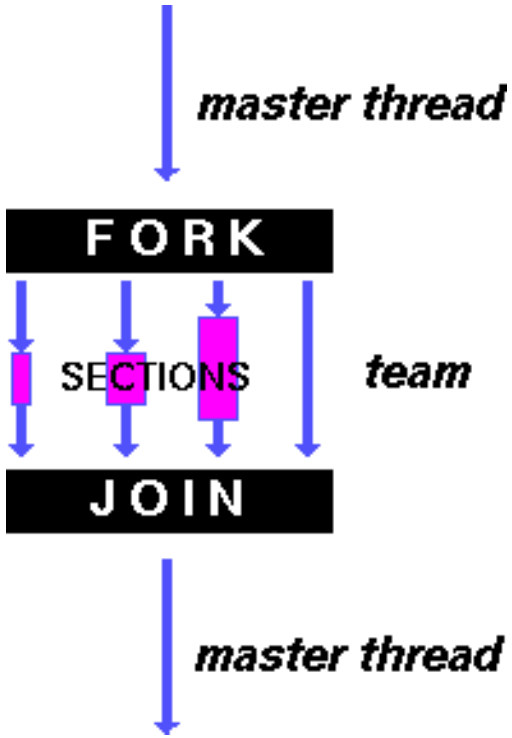
```
#pragma omp parallel private (x)
```

OpenMP Directives (Work-Sharing)

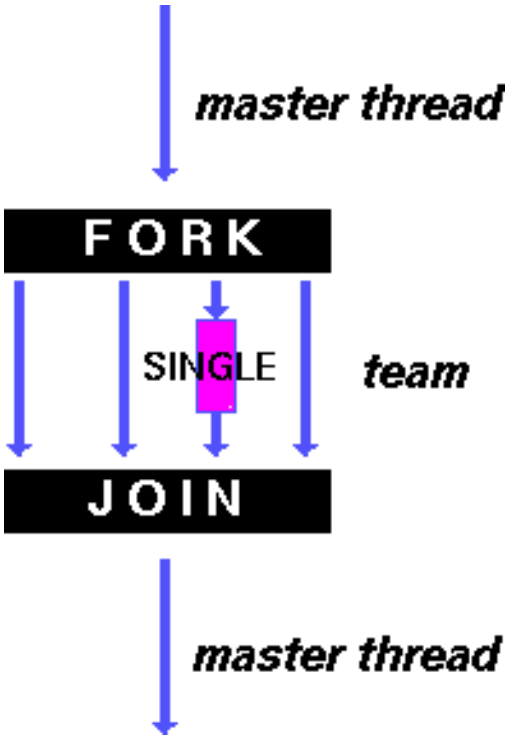
- These are defined *within* a `parallel` section



Shares iterations of a loop across the threads



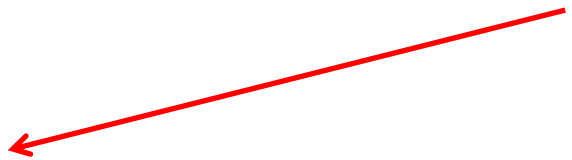
Each section is executed by a separate thread



Serializes the execution of a thread

`Parallel` Statement Shorthand

```
#pragma omp parallel
{
  #pragma omp for
  for(i=0; i<len; i++) { … }
}
```



can be shortened to:

```
#pragma omp parallel for
  for(i=0; i<len; i++) { … }
```

- Also works for `sections`

Building Block: `for` loop

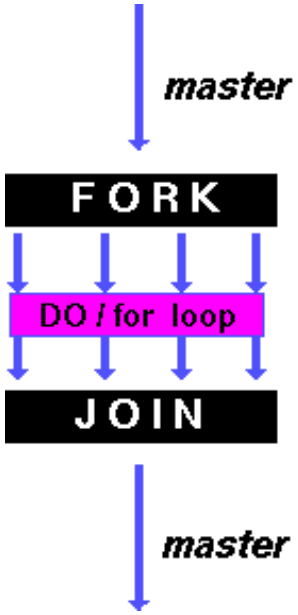
```
for (i=0; i<max; i++) zero[i] = 0;
```

- Breaks *for loop* into chunks, and allocate each to a separate thread
  - e.g. if `max` = 100 with 2 threads:
    assign 0-49 to thread 0, and 50-99 to thread 1
- Must have relatively simple "shape" for an OpenMP-aware compiler to be able to parallelize it
  - Necessary for the run-time system to be able to determine how many of the loop iterations to assign to each thread
- No premature exits from the loop allowed ← In general, don't jump outside of any pragma block
  - i.e. No `break`, `return`, `exit`, `goto` statements

Parallel `for` *pragma*

```
#pragma omp parallel for
   for (i=0; i<max; i++) zero[i] = 0;
```

- Master thread creates additional threads, each with a separate execution context
- All variables declared outside for loop are shared by default, except for loop index which is *private* per thread (Why?)
- Implicit "barrier" synchronization at end of for loop
- Divide index regions sequentially per thread
  - Thread 0 gets 0, 1, …, (max/n)-1;
  - Thread 1 gets max/n, max/n+1, …, 2*(max/n)-1
  - Why?



OpenMP Timing

- Elapsed wall clock time:

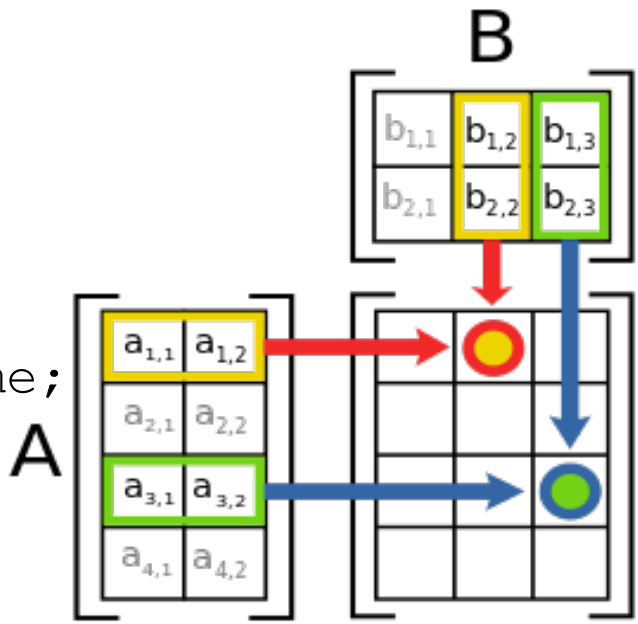
```
double omp_get_wtime(void);
```

  - Returns elapsed wall clock time in seconds
  - Time is measured per thread, no guarantee can be made that two distinct threads measure the same time
  - Time is measured from "some time in the past," so subtract results of two calls to `omp_get_wtime` to get elapsed time

Matrix Multiply in OpenMP

```
// C[M][N] = A[M][P] × B[P][N]
start_time = omp_get_wtime();
#pragma omp parallel for private(tmp, j, k)
  for (i=0; i<M; i++){
    for (j=0; j<N; j++){
      tmp = 0.0;
      for( k=0; k<P; k++){
        /* C(i,j) = sum(over k) A(i,k) * B(k,j)*/
        tmp += A[i][k] * B[k][j];
      }
      C[i][j] = tmp;
    }
  }
run_time = omp_get_wtime() - start_time;
```

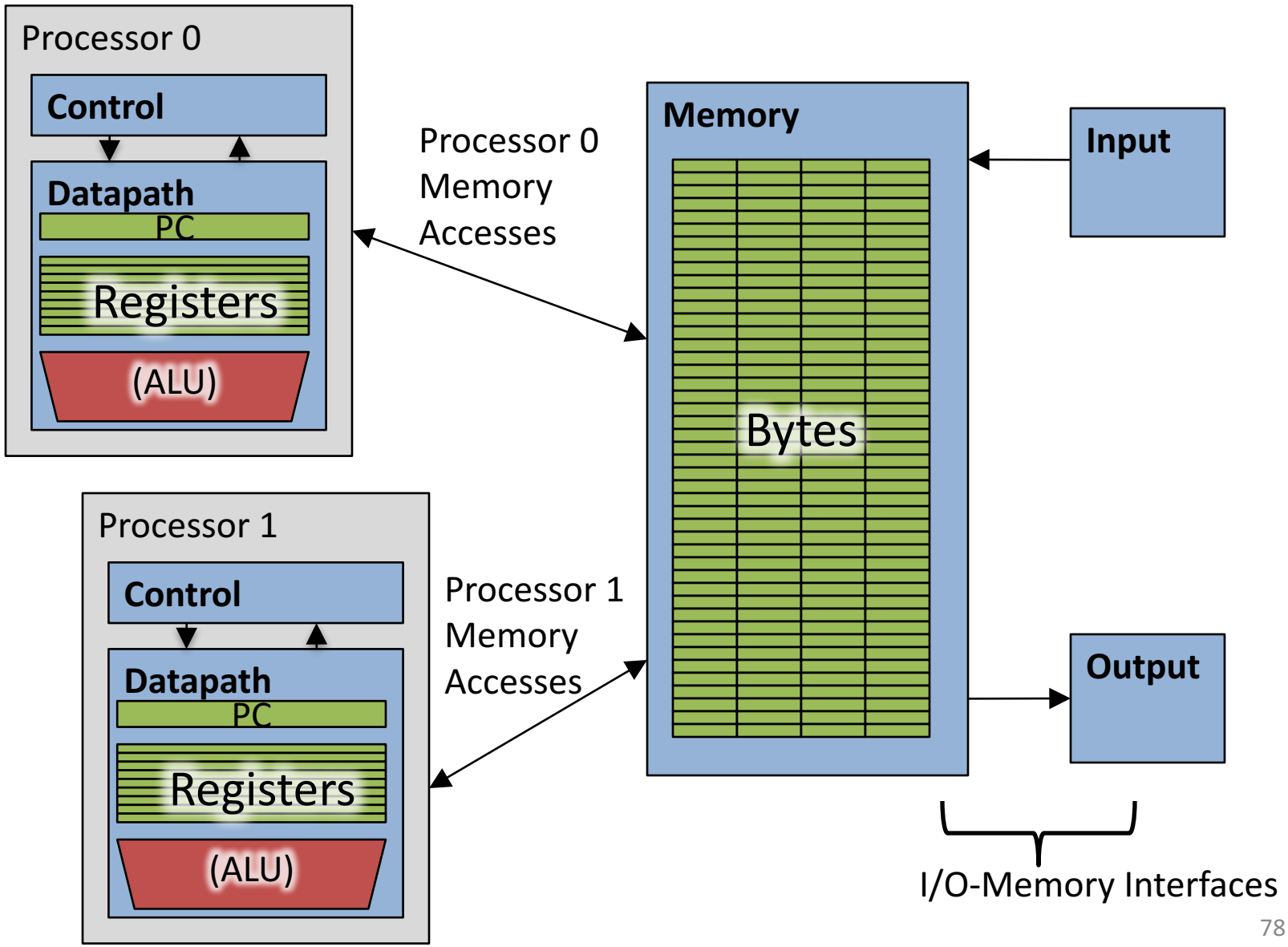
Outer loop spread across N threads;
inner loops inside a single thread



Notes on Matrix Multiply Example

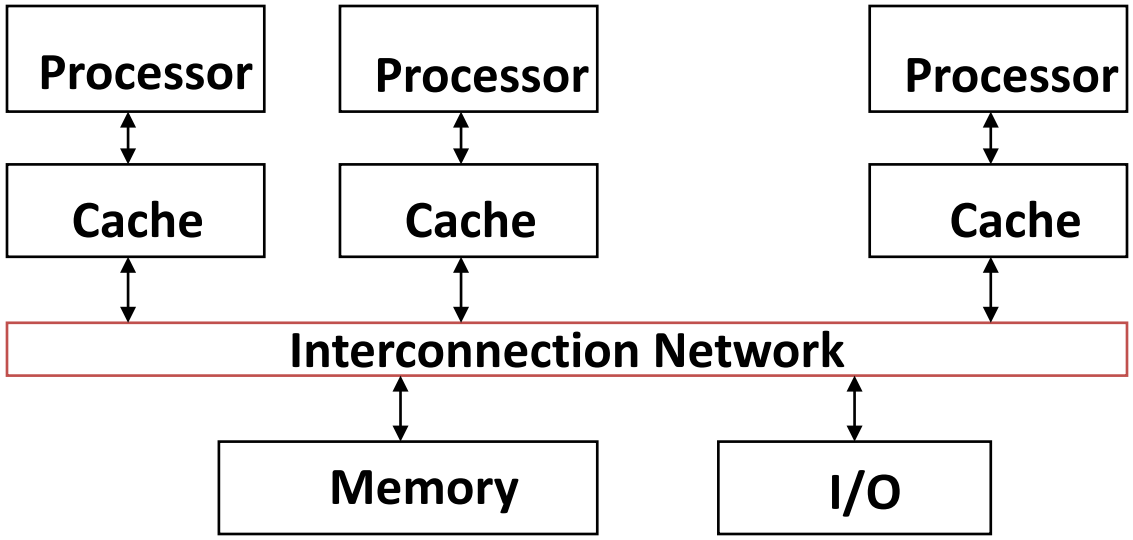
- More performance optimizations available:
  - Higher *compiler optimization* (-O2, -O3) to reduce number of instructions executed
  - *Cache blocking* to improve memory performance
  - Using SIMD SSE instructions to raise floating point computation rate (*DLP*)

Simple Multi-core Processor



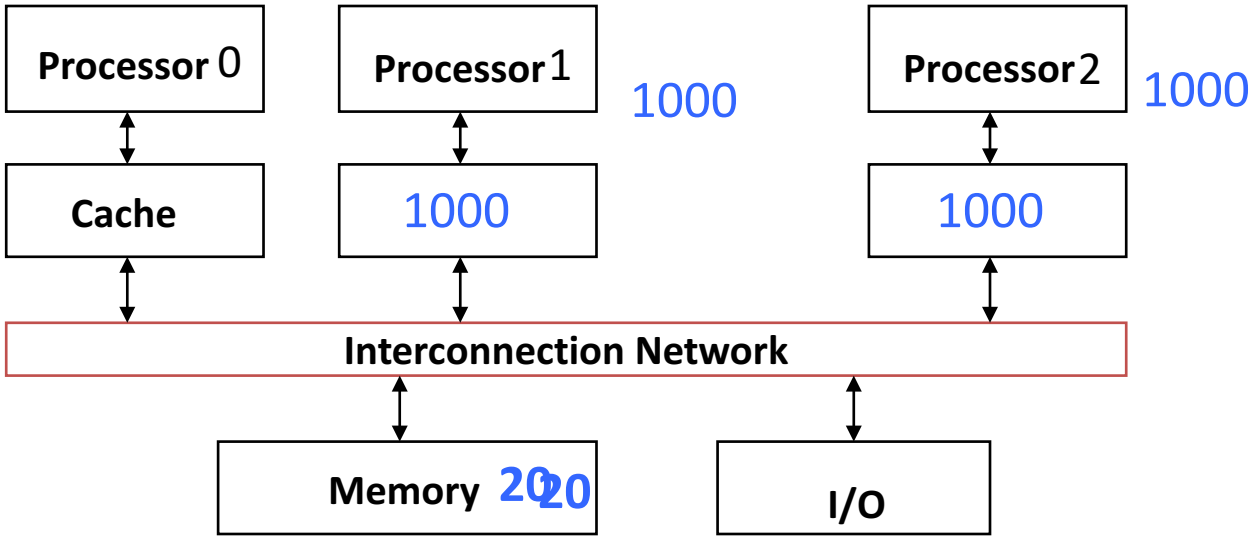
Multiprocessor Caches

- Memory is a performance bottleneck even with one processor
- Use caches to reduce bandwidth demands on main memory
- Each core has a local private cache holding data it has accessed recently
- Only cache misses have to access the shared common memory



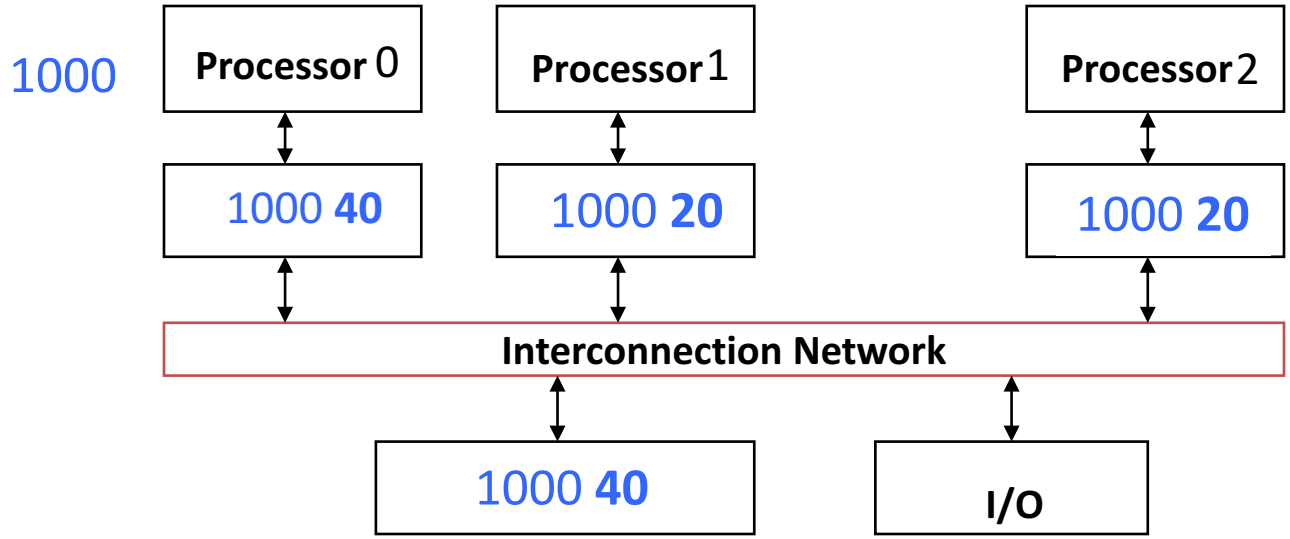
Shared Memory and Caches

- What if?
  - Processors 1 and 2 read Memory[1000] (value 20)



Shared Memory and Caches

- Now:
  - Processor 0 writes Memory[1000] with 40



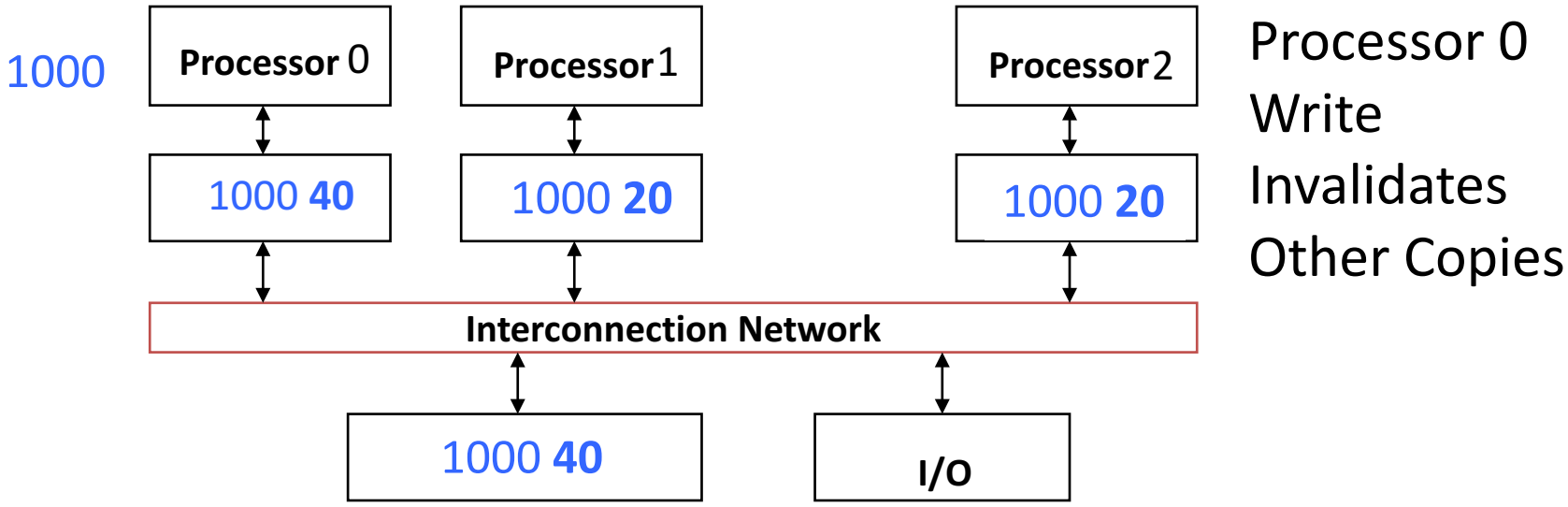
Problem?

Keeping Multiple Caches Coherent

- Architect's job: shared memory
  => keep cache values coherent
- Idea: When any processor has cache miss or writes, notify other processors via interconnection network
  - If only reading, many processors can have copies
  - If a processor writes, invalidate any other copies
- Write transactions from one processor, other caches "snoop" the common interconnect checking for tags they hold
  - Invalidate any copies of same address modified in other cache

Shared Memory and Caches

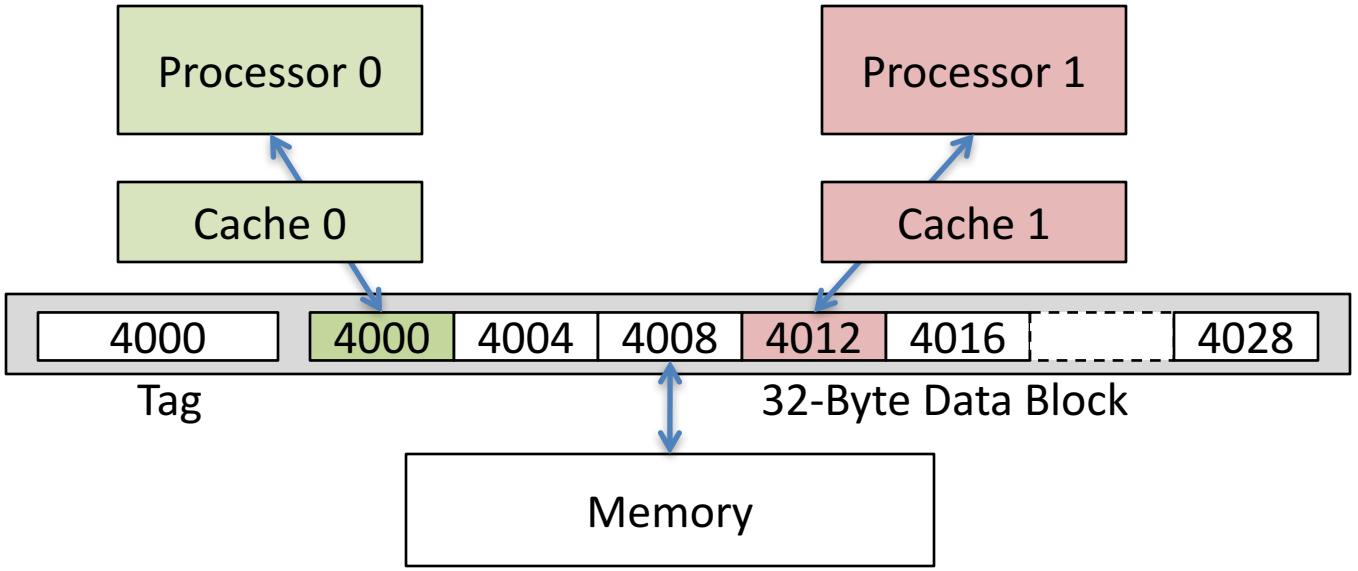
- Example, now with cache coherence
  - Processors 1 and 2 read Memory[1000]
  - Processor 0 writes Memory[1000] with 40



Question: Which statement(s) are true?

- **A: Using write-through caches removes the need for cache coherence**
- **B: Every processor store instruction must check contents of other caches**
- **C: Most processor load and store accesses only need to check in local private cache**
- **D: Only one processor can cache any memory location at one time**

Cache Coherency Tracked by Block



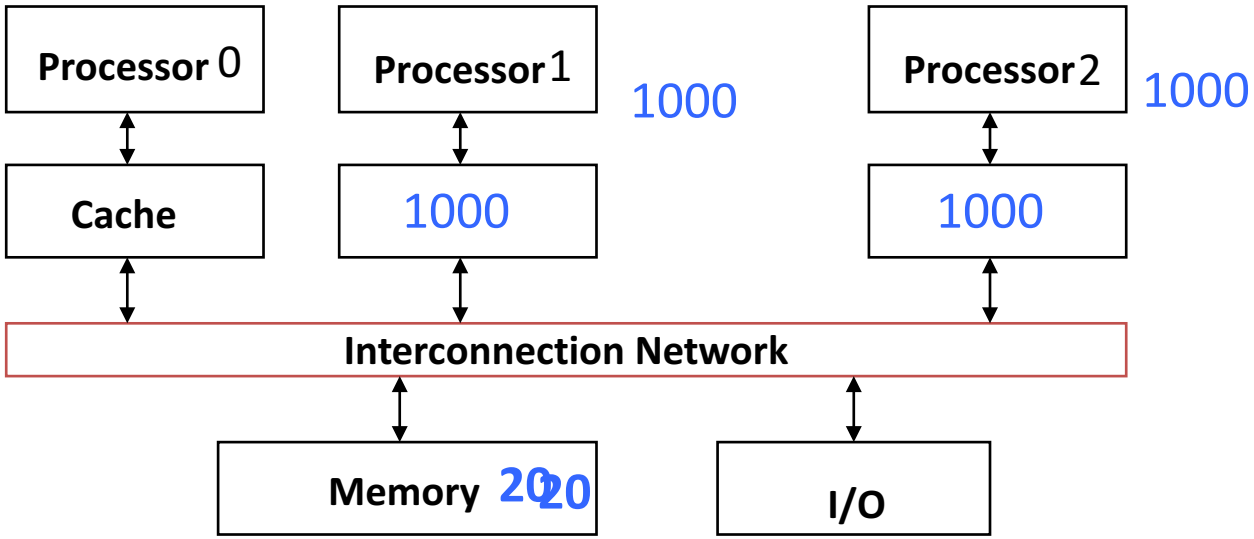
- Suppose block size is 32 bytes
- Suppose Processor 0 reading and writing variable X, Processor 1 reading and writing variable Y
- Suppose in X location 4000, Y in 4012
- What will happen?

Coherency Tracked by Cache Block

- Block ping-pongs between two caches even though processors are accessing disjoint variables
- Effect called *false sharing*
- How can you prevent it?

Shared Memory and Caches

- Use valid bit to ”unload” cache lines (in Processors 1 and 2)
- Dirty bit tells me: ”I am the only one using this cache line”! => no need to announce on Network!



Review: Understanding Cache Misses: The 3Cs

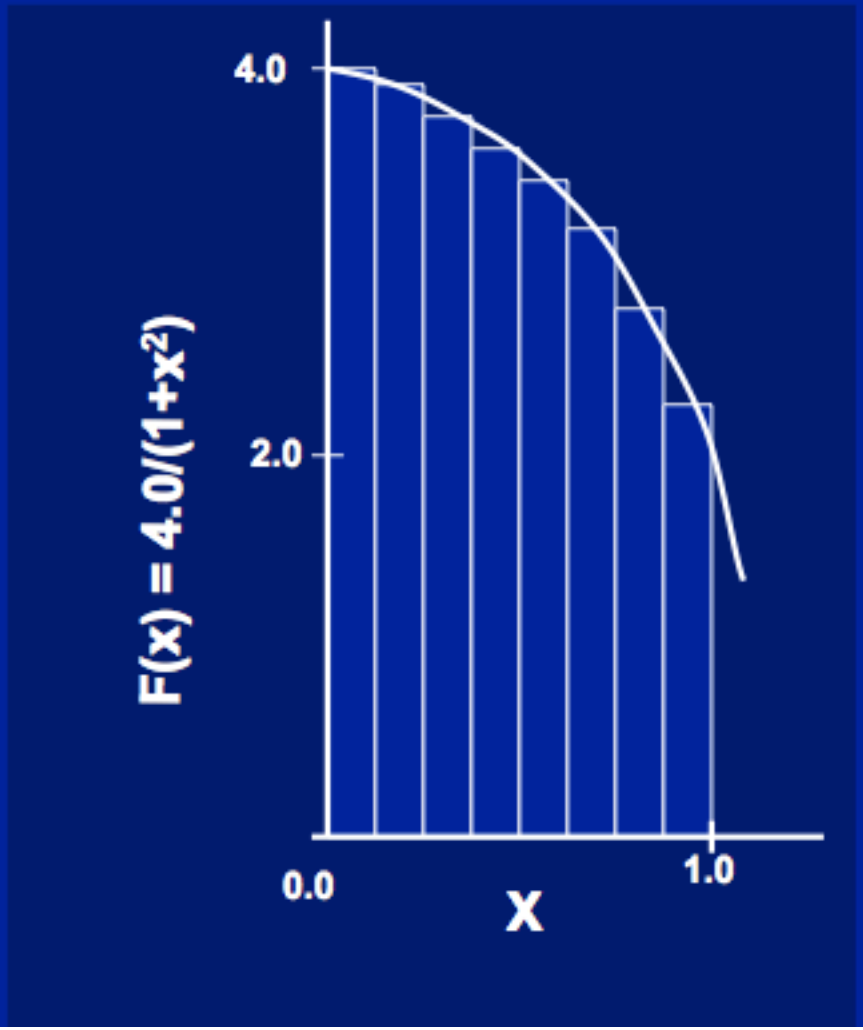
- Compulsory (cold start or process migration, 1st reference):
  - First access to block, impossible to avoid; small effect for long-running programs
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- Capacity (not compulsory and…)
  - Cache cannot contain all blocks accessed by the program ***even with perfect replacement policy in fully associative cache***
  - Solution: increase cache size (may increase access time)
- Conflict (not compulsory or capacity and…):
  - Multiple memory locations map to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity (may increase access time)
  - Solution 3: improve replacement policy, e.g.. LRU

Fourth "C" of Cache Misses: *Coherence* Misses

- Misses caused by coherence traffic with other processor
- Also known as *communication* misses because represents data moving between processors working together on a parallel program
- For some parallel programs, coherence misses can dominate total misses

Example: Calculating π

Numerical Integration



Mathematically, we know that:

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

We can approximate the integral as a sum of rectangles:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i.

Sequential Calculation of π in C

```
#include <stdio.h>          /* Serial Code */
static long num_steps = 100000;
double step;
void main () {
     int i;
     double x, pi, sum = 0.0;
     step = 1.0/(double)num_steps;
     for (i = 1; i <= num_steps; i++) {
       x = (i - 0.5) * step;
       sum = sum + 4.0 / (1.0 + x*x);
     }
     pi = sum * step;
     printf ("pi = %6.12f\n", pi);
}
```

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Parallel OpenMP Version (1)

```
#include <omp.h>
#define NUM_THREADS 4
static long num_steps = 100000; double step;

void main () {
  int i;       double  x, pi, sum[NUM_THREADS];
  step = 1.0/(double) num_steps;
  #pragma omp parallel private ( i, x )
  {
    int id = omp_get_thread_num();
    for (i=id, sum[id]=0.0; i< num_steps; i=i+NUM_THREADS)
    {
      x = (i+0.5)*step;
      sum[id] += 4.0/(1.0+x*x);
    }
  }
  for(i=1; i<NUM_THREADS; i++) sum[0] += sum[i];
  pi = sum[0] * step;
  printf ("pi = %6.12f\n", pi);
}
```

OpenMP Reduction

```
double avg, sum=0.0, A[MAX]; int i;
#pragma omp parallel for private ( sum )
for (i = 0; i <= MAX ; i++)
    sum += A[i];
avg = sum/MAX;  // bug
```

- *Problem is that we really want sum over all threads!*
- *Reduction*: specifies that 1 or more variables that are private to each thread are subject of reduction operation at end of parallel region:
  reduction(operation:var) where
  - *Operation*: operator to perform on the variables (var) at the end of the parallel region : +, *, -, &, ^, |, &&, or ||.
  - *Var*: One or more variables on which to perform scalar reduction.

```
double avg, sum=0.0, A[MAX]; int i;
#pragma omp for reduction(+ : sum)
for (i = 0; i <= MAX ; i++)
    sum += A[i];
avg = sum/MAX;
```

Version 2: parallel for, reduction

```
#include <omp.h>
#include <stdio.h>
/static long num_steps = 100000;
double step;
void main (){
     int i;    double x, pi, sum = 0.0;
     step = 1.0 / (double)num_steps;
#pragma omp parallel for private(x) reduction(+:sum)
     for (i=1; i<= num_steps; i++){
        x = (i - 0.5) * step;
        sum = sum + 4.0 / (1.0+x*x);
     }
     pi = sum * step;
     printf ("pi = %6.8f\n", pi);
}
```

And in Conclusion, …

- Multiprocessor/Multicore uses Shared Memory
  - Cache coherency implements shared memory even with multiple copies in multiple caches
  - False sharing a concern; watch block size!
- OpenMP as simple parallel extension to C
  - Threads, Parallel for, private, reductions …
  - ≈ C: small so easy to learn, but not very high level and it's easy to get into trouble
  - Much we didn't cover – including other synchronization mechanisms (locks, etc.)