Personal computer (PC): A computer designed for use by an individual, usually incorporating a graphics display, a keyboard, and a mouse.

Server: A computer used for running larger programs for multiple users, often simultaneously, and typically accessed only via a network.

Supercomputer: A class of computers with the highest performance and cost; they are configured as servers and typically cost tens to hundreds of millions of dollars.

Embedded computer: A computer inside another device used for running one predetermined application or collection of software.

Personal mobile devices (PMDs): Small wireless devices used to connect to the Internet; they rely on batteries for power, and software is installed by downloading apps. Conventional examples are smart phones and tablets.

Cloud Computing: Refers to large collections of servers that provide services over the Internet; some providers rent dynamically varying numbers of servers as a utility.

Software as a Service (SaaS): Delivers software and data as a service over the Internet, usually via a thin program such as a browser that runs on local client devices, instead of binary code that must be installed, and runs wholly on that device. Examples include web search and social networking.

Multicore Microprocessor: A microprocessor containing multiple processors (“cores”) in a single integrated circuit.

Acronym: A word constructed by taking the initial letters of a string of words. For example: RAM is an acronym for Random Access Memory, and CPU is an acronym for Central Processing Unit.

Eight Great Ideas in Computer Architecture

1. **Design for Moore’s Law**
   It states that integrated circuit resources double every 18–24 months. Therefore, computer architects must anticipate where the technology will be when the design finishes rather than design for where it starts.

2. **Use Abstraction to Simplify Design**
   Lower-level details are hidden to offer a simpler model at higher levels.

3. **Make the Common Case Fast**
   Making the common case fast will tend to enhance performance better than optimizing the rare case.

4. **Performance via Parallelism**
   Computer architects have offered designs that get more performance by performing operations in parallel.

5. **Performance via Pipelining**
   A particular pattern of parallelism is so prevalent in computer architecture that it merits its own name: pipelining. The pipeline icon is a sequence of pipes, with each section representing one stage of the pipeline.

6. **Performance via Prediction**
   In some cases, it can be faster on average to guess and start working rather than wait until you know for sure, assuming that the mechanism to recover from a misprediction is not too expensive and the prediction is relatively accurate.

7. **Hierarchy of Memories**
   The fastest, smallest, and most expensive memory per bit is placed at the top of the hierarchy and the slowest, largest, and cheapest per bit at the bottom.

8. **Dependability via Redundancy**
   Since any physical device can fail, systems are made dependable by including redundant components that can take over when a failure occurs and to help detect failures.

Systems software: Software that provides services that are commonly useful, including operating systems, compilers, loaders, and assemblers.

Operating system: Supervising program that manages the resources of a computer for the benefit of the programs that run on that computer.

Compiler: A program that translates high-level language statements into assembly language statements.

Binary Digit: Also called a bit. One of the two numbers in base 2 (0 or 1) that are the components of information.

Instruction: A command that computer hardware understands and obeys.

Assembler: A program that translates a symbolic version of instructions into the binary version.

Assembly Language: A symbolic representation of machine instructions.

Machine Language: A binary representation of machine instructions.

High-Level Programming Language: A portable language such as C, C++, Java, or Visual Basic that is composed of words and algebraic notation that can be translated by a compiler into assembly language.

Input Device: A mechanism through which the computer is fed information, such as a keyboard.

Output Device: A mechanism that conveys the result of a computation to a user, such as a display, or to another computer.

Liquid Crystal Display: A display technology using a thin layer of liquid polymers that can be used to transmit or block light according to whether a charge is applied.

Active-Matrix Display: A liquid crystal display using a transistor to control the transmission of light at each individual pixel.

Pixel: The smallest individual picture element. Screens are composed of hundreds of thousands to millions of pixels, organized in a matrix.

Integrated Circuit: Also called a chip. A device combining dozens to millions of transistors.

Central Processor Unit (CPU): Also called processor. The active part of the computer, which contains the data-path and control and which adds numbers, tests numbers, signals I/O devices to activate, and so on.

Datapath: The component of the processor that performs arithmetic operations.

Control: The component of the processor that commands the data-path, memory, and I/O devices according to the instructions of the program.

Memory: The storage area in which programs are kept when they are running and that contains the data needed by the running programs.

Dynamic-Random-Access-Memory (DRAM): Memory built as an integrated circuit; it provides random access to any location.

Cache Memory: A small, fast memory that acts as a buffer for a slower, larger memory.

Static Random Access Memory (SRAM): Memory built as an integrated circuit, but faster and less dense than DRAM.

Instruction Set Architecture: Also called architecture. An abstract interface between the hardware and the lowest-level software that encompasses all the information necessary to write a machine language program that will run correctly, including instructions, registers, memory access, I/O, and so on.

Application Binary Interface (ABI): The user portion of the instruction set plus the operating system interfaces used by application programmers. It defines a standard for binary portability across computers.

Implementation: Hardware that obeys the architecture abstraction.

Volatile Memory: Storage, such as DRAM, that retains data only if it is receiving power.

Non-Volatile Memory: A form of memory that retains data even in the absence of a power source and that is used to store programs between runs. A DVD disk is non-volatile.

Main Memory: Also called primary memory. Memory used to hold programs while they are running; typically consists of DRAM in today’s computers.

Secondary Memory: Non-volatile memory used to store programs and data between runs; typically consists of flash memory in PMDs and magnetic disks in servers.

Magnetic Disk: Also called hard disk. A form of non-volatile secondary memory composed of rotating platters coated with a magnetic recording material.

Flash Memory: A non-volatile semi-conductor memory. It is cheaper and slower than DRAM but more expensive per bit and faster than magnetic disks.

Advantages of Networked Computers:

1. *Communication*: Information is exchanged between computers at high speeds.
2. *Resource Sharing*: Rather than each computer having its own I/O devices, computers on the network can share I/O devices.
3. *Non-Local Access*: By connecting computers over long distances, users need not be near the computer they are using.

Local Area Network (LAN): A network designed to carry data within a geographically confined area, typically within a single building.

Wide Area Network (WAN): A network extended over hundreds of kilo-meters that can span a continent.

Transistor: An on/off switch controlled by an electric signal.

Very large-scale integrated (VLSI) circuit: A device containing hundreds of thousands to millions of transistors.

Silicon: A natural element (found in sand) that is a semiconductor.

Semiconductor: A substance that does not conduct electricity well.

Silicon Crystal Ingot: A rod composed of a silicon crystal that is between 8 and 12 inches in diameter and about 12 to 24 inches long.

Wafer: A slice from a silicon ingot no more than 0.1 inches thick, used to create chips.

Defect: A microscopic flaw in a wafer or in patterning steps that can result in the failure of the die containing that defect.

Die: The individual rectangular sections that are cut from a wafer, more informally known as chips.

Yield: The percentage of good dies from the total number of dies on the wafer.

Elaboration: The cost of an integrated circuit can be expressed in three simple equations:

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{yield}}$$

$$\text{Dies per wafer} \approx \frac{\text{Wafer area}}{\text{Die area}}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

The first equation is straightforward to derive. The second is an approximation, since it does not subtract the area near the border of the round wafer that cannot accommodate the rectangular dies (see Figure 1.13). The final equation is based on empirical observations of yields at integrated circuit factories, with the exponent related to the number of critical processing steps.

Hence, depending on the defect rate and the size of the die and wafer, costs are generally not linear in the die area.

Response Time: Also called execution time. The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on.

Throughput: Also called bandwidth. Another measure of performance, it is the number of tasks completed per unit time.

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

If X is n times as fast as Y, then the execution time on Y is n times as long as it is on X:

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

CPU Execution Time: Also called CPU time. The actual time the CPU spends computing for a specific task.

User CPU Time: The CPU time spent in a program itself.

System CPU Time: The CPU time spent in the operating system performing tasks on behalf of the program.

Clock Cycle: Also called tick, clock tick, clock period, clock, or cycle. The time for one clock period, usually of the processor clock, which runs at a constant rate.

Clock Period: The length of each clock cycle.

$$\begin{matrix}\text{CPU execution time} \\ \text{for a program}\end{matrix} = \begin{matrix}\text{CPU clock cycles} \\ \text{for a program}\end{matrix} \times \text{Clock cycle time}$$

$$\begin{matrix}\text{CPU execution time} \\ \text{for a program}\end{matrix} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \begin{matrix}\text{Average clock cycles} \\ \text{per instruction}\end{matrix}$$

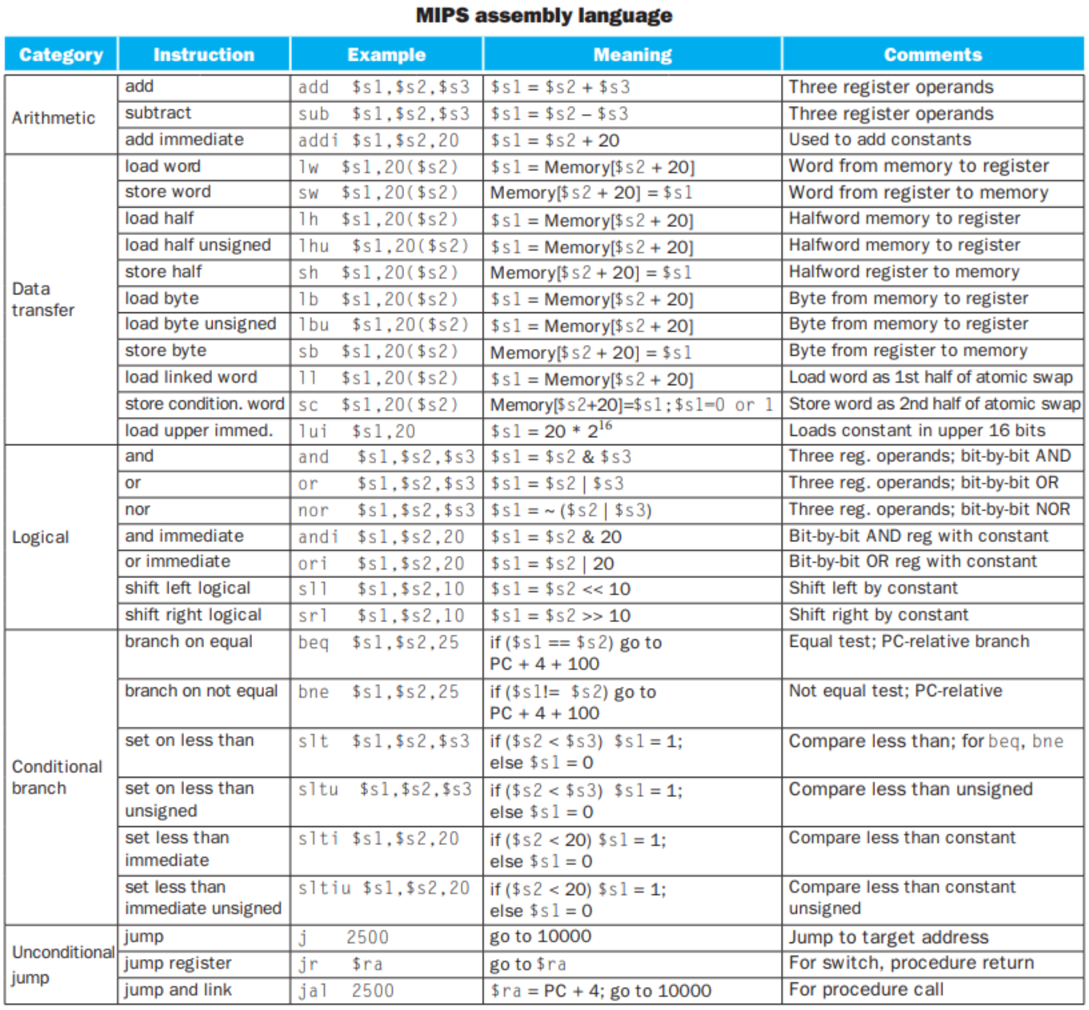
Clock Cycles Per Instruction (CPI): Average number of clock cycles per instruction for a program or program fragment.

Instruction Count: The number of instructions executed by the program.

Instruction Mix: A measure of the dynamic frequency of instructions across one or many programs.

Instruction Set: The vocabulary of commands understood by a given architecture.

Stored-Program Concept: The idea that instructions and data of many types can be stored in memory as numbers, leading to the stored-program computer.



MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	`$s1` = `$s2` + `$s3`	Three register operands
	subtract	`sub $s1,$s2,$s3`	`$s1` = `$s2` – `$s3`	Three register operands
	add immediate	`addi $s1,$s2,20`	`$s1` = `$s2` + 20	Used to add constants
Data transfer	load word	`lw $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Word from memory to register
	store word	`sw $s1,20($s2)`	Memory[`$s2` + 20] = `$s1`	Word from register to memory
	load half	`lh $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Halfword memory to register
	load half unsigned	`lhu $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Halfword memory to register
	store half	`sh $s1,20($s2)`	Memory[`$s2` + 20] = `$s1`	Halfword register to memory
	load byte	`lb $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Byte from memory to register
	load byte unsigned	`lbu $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Byte from memory to register
	store byte	`sb $s1,20($s2)`	Memory[`$s2` + 20] = `$s1`	Byte from register to memory
	load linked word	`ll $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Load word as 1st half of atomic swap
	store condition. word	`sc $s1,20($s2)`	Memory[`$s2`+20]=`$s1`;`$s1`=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	`lui $s1,20`	`$s1` = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	`and $s1,$s2,$s3`	`$s1` = `$s2` & `$s3`	Three reg. operands; bit-by-bit AND
	or	`or $s1,$s2,$s3`	`$s1` = `$s2` | `$s3`	Three reg. operands; bit-by-bit OR
	nor	`nor $s1,$s2,$s3`	`$s1` = ~ (`$s2` | `$s3`)	Three reg. operands; bit-by-bit NOR
	and immediate	`andi $s1,$s2,20`	`$s1` = `$s2` & 20	Bit-by-bit AND reg with constant
	or immediate	`ori $s1,$s2,20`	`$s1` = `$s2` | 20	Bit-by-bit OR reg with constant
	shift left logical	`sll $s1,$s2,10`	`$s1` = `$s2` << 10	Shift left by constant
	shift right logical	`srl $s1,$s2,10`	`$s1` = `$s2` >> 10	Shift right by constant
Conditional branch	branch on equal	`beq $s1,$s2,25`	if (`$s1` == `$s2`) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1,$s2,25`	if (`$s1`!= `$s2`) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1,$s2,$s3`	if (`$s2` < `$s3`) `$s1` = 1; else `$s1` = 0	Compare less than; for `beq`, `bne`
	set on less than unsigned	`sltu $s1,$s2,$s3`	if (`$s2` < `$s3`) `$s1` = 1; else `$s1` = 0	Compare less than unsigned
	set less than immediate	`slti $s1,$s2,20`	if (`$s2` < 20) `$s1` = 1; else `$s1` = 0	Compare less than constant
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if (`$s2` < 20) `$s1` = 1; else `$s1` = 0	Compare less than constant unsigned
Unconditional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to `$ra`	For switch, procedure return
	jump and link	`jal 2500`	`$ra` = PC + 4; go to 10000	For procedure call

Registers: The operands of arithmetic instructions are restricted; they must be from a limited number of special locations built directly in hardware called registers. Registers are primitives used in hardware design that are also visible to the programmer when the computer is completed.

Word: The natural unit of access in a computer, usually a group of 32 bits; corresponds to the size of a register in the MIPS architecture.

Data Transfer Instruction: A command that moves data between memory and registers.

Address: A value used to delineate the location of a specific data element within a memory array.

Load: The data transfer instruction that copies data from memory to a register is traditionally called load. The actual MIPS name for this instruction is lw, standing for load word.

Alignment Restriction: A requirement that data be aligned in memory on natural boundaries.

Computers divide into those that use the address of the left most or “big end” byte as the word address versus those that use the rightmost or “little end” byte. MIPS is in the big-endian camp.

Store: The instruction complementary to load is traditionally called store; it copies data from a register to memory. The actual MIPS name is sw, standing for store word.

Spilling Registers: The process of putting less commonly used variables (or those needed later) into memory.

Registers take less time to access and have higher throughput than memory, making data in registers both faster to access and simpler to use. Accessing registers also uses less energy than accessing memory.

Binary Digit: Also called binary bit. One of the two numbers in base 2, 0 or 1, that are the components of information.

Least Significant Bit: The rightmost bit in a MIPS word.

Most Significant Bit: The left most bit in a MIPS word.

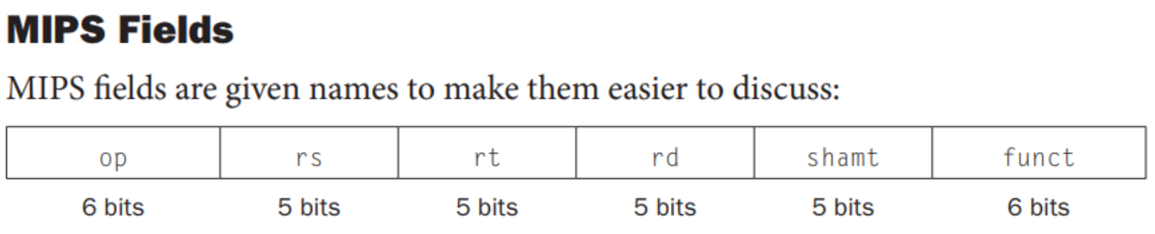
Two’s Complement: Two’s complement gets its name from the rule that the unsigned sum of an n-bit number and its n-bit negative is 2^n; hence, the negation or complement of a number x is 2^n - x, or its “two’s complement.”

One’s Complement: A notation that represents the most negative value by 10 . . . 000two and the most positive value by 01 . . . 11two, leaving an equal number of negatives and positives but ending up with two zeros, one positive (00 . . . 00two) and one negative (11 . . . 11two). Th e term is also used to mean the inversion of every bit in a pattern: 0 to 1 and 1 to 0.

Biased Notation: A notation that represents the most negative value by 00 . . . 000two and the most positive value by 11 . . . 11two, with 0 typically having the value 10 . . . 00two, thereby biasing the number such that the number plus the bias has a non-negative representation.

Instruction Format: A form of representation of an instruction composed of fields of binary numbers.

Machine language: Binary representation used for communication within a computer system.



MIPS Fields

MIPS fields are given names to make them easier to discuss:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

(These are for R-type, that is, Register-Type Instructions)

1. **Op**: Basic operation of the instruction, traditionally called the opcode. It is the field that denotes the operation and format of an instruction.
2. **Rs**: The first register source operand.
3. **Rt**: The second register source operand.
4. **Rd**: The register destination operand. It gets the result of the operation.

5. **Shamt**: Shift amount.
6. **Funct**: This field, often called the function code, selects the specific variant of the operation in the op field.

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

(These are for I-type, that is immediate-type instructions)

In a load word instruction, the **rt** field specifies the destination register, which receives the result of the load.

AND: A logical bit-by-bit operation with two operands that calculates a 1 only if there is a 1 in both operands.

OR: A logical bit-by-bit operation with two operands that calculates a 1 if there is a 1 in either operand.

NOT: A logical bit-by-bit operation with one operand that inverts the bits; that is, it replaces every 1 with a 0, and every 0 with a 1.

NOR: A logical bit-by-bit operation with two operands that calculates the NOT of the OR of the two operands. That is, it calculates a 1 only if there is a 0 in both operands.

Conditional Branch: An instruction that requires the comparison of two values and that allows for a subsequent transfer of control to a new address in the program based on the outcome of the comparison.

Basic Block: A sequence of instructions without branches (except possibly at the end) and without branch targets or branch labels (except possibly at the beginning).

Jump address table: Also called jump table. A table of addresses of alternative instruction sequences.

PC-relative addressing: An addressing regime in which the address is the sum of the program counter (PC) and a constant in the instruction.

Name	Register number	Usage	Preserved on call?
`$zero`	0	The constant value 0	n.a.
`$v0–$v1`	2–3	Values for results and expression evaluation	no
`$a0–$a3`	4–7	Arguments	no
`$t0–$t7`	8–15	Temporaries	no
`$s0–$s7`	16–23	Saved	yes
`$t8–$t9`	24–25	More temporaries	no
`$gp`	28	Global pointer	yes
`$sp`	29	Stack pointer	yes
`$fp`	30	Frame pointer	yes
`$ra`	31	Return address	yes

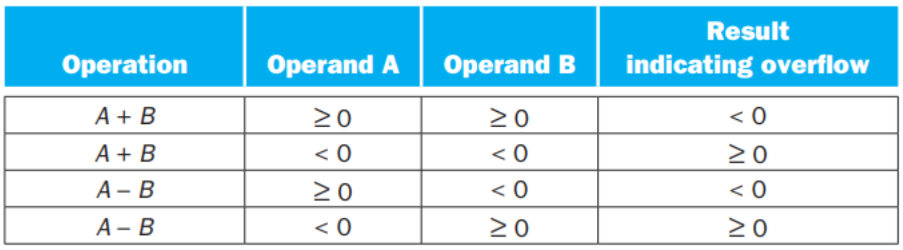
FIGURE 2.14 MIPS register conventions. Register 1, called `$at`, is reserved for the assembler (see Section 2.12), and registers 26–27, called `$k0–$k1`, are reserved for the operating system. This information is also found in Column 2 of the MIPS Reference Data Card at the front of this book.

op(31:26)								
28–26 31–29	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	set less than imm. unsigned	andi	ori	xori	load upper immediate
2(010)	TLB	FlPt						
3(011)								
4(100)	load byte	load half	lwl	load word	load byte unsigned	load half unsigned	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	load linked word	lwc1						
7(111)	store cond. word	swc1						

op(31:26)=010000 (TLB), rs(25:21)								
23–21 25–24	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(00)	mfc0		cfc0		mtc0		ctc0	
1(01)								
2(10)								
3(11)								

op(31:26)=000000 (R-format), funct(5:0)								
2–0 5–3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	shift left logical		shift right logical	sra	sllv		srlv	srav
1(001)	jump register	jalr			syscall	break		
2(010)	mfhi	mthi	mflo	mtlo				
3(011)	mult	multu	div	divu				
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)
5(101)			set l.t.	set l.t. unsigned				
6(110)								
7(111)								

Arithmetic Logic Unit (ALU): Hardware that performs addition, subtraction, and usually logical operations such as AND and OR.



Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

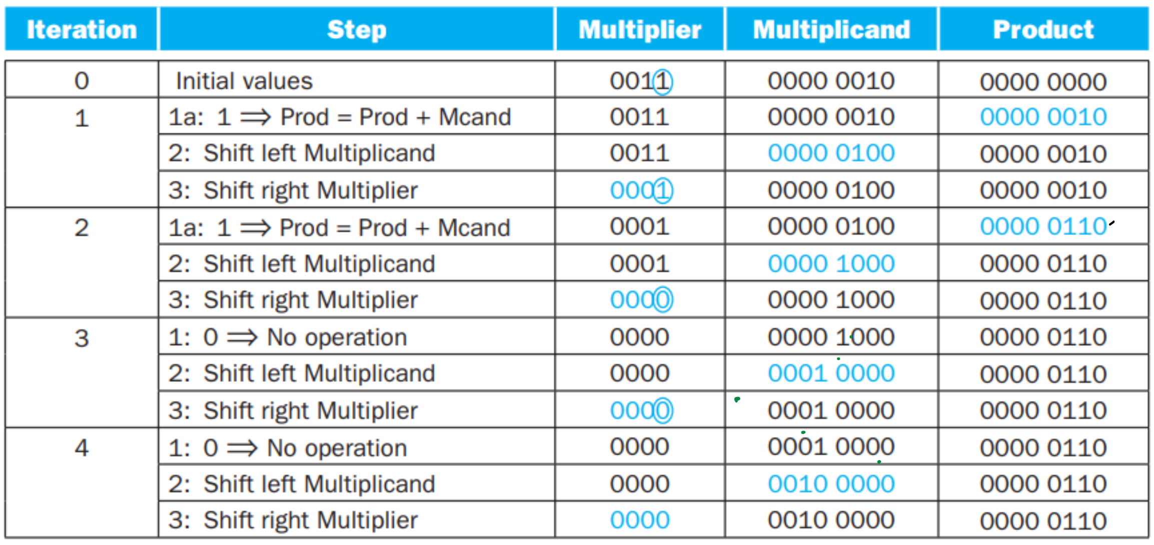
Exception: Also called interrupt on many computers. An unscheduled event that disrupts program execution; used to detect overflow.

Interrupt: An exception that comes from outside of the processor (Some architectures use the term interrupt for all exceptions).

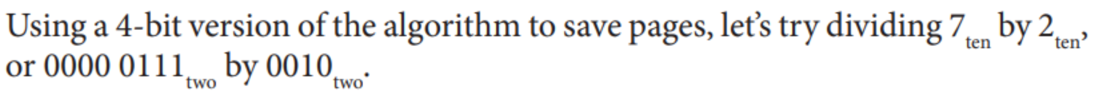
MIPS includes a register called the **exception program counter (EPC)** to contain the address of the instruction that caused the exception.

Saturation: When a calculation overflows, the result is set to the largest positive number or most negative number, rather than a modulo calculation as in two's complement arithmetic.

Using 4-bit numbers to save space, multiply $2_{ten} \times 3_{ten}$, or $0010_{two} \times 0011_{two}$.



Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	001①	0000 0010	0000 0000
1	1a: 1 $\Rightarrow$ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	000①	0000 0100	0000 0010
2	1a: 1 $\Rightarrow$ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	000⓪	0000 1000	0000 0110
3	1: 0 $\Rightarrow$ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	000⓪	0001 0000	0000 0110
4	1: 0 $\Rightarrow$ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110


Using a 4-bit version of the algorithm to save pages, let's try dividing 7_{ten} by 2_{ten}, or $0000\ 0111_{two}$ by 0010_{two}.

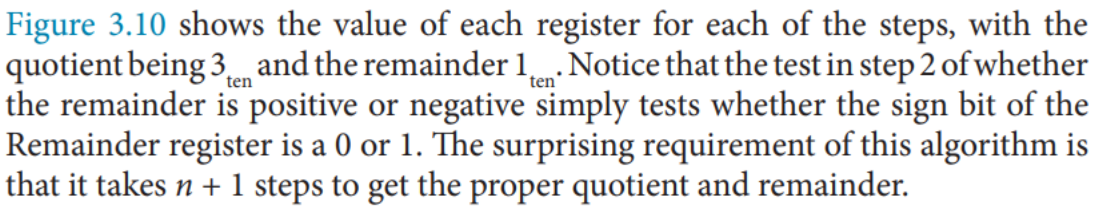
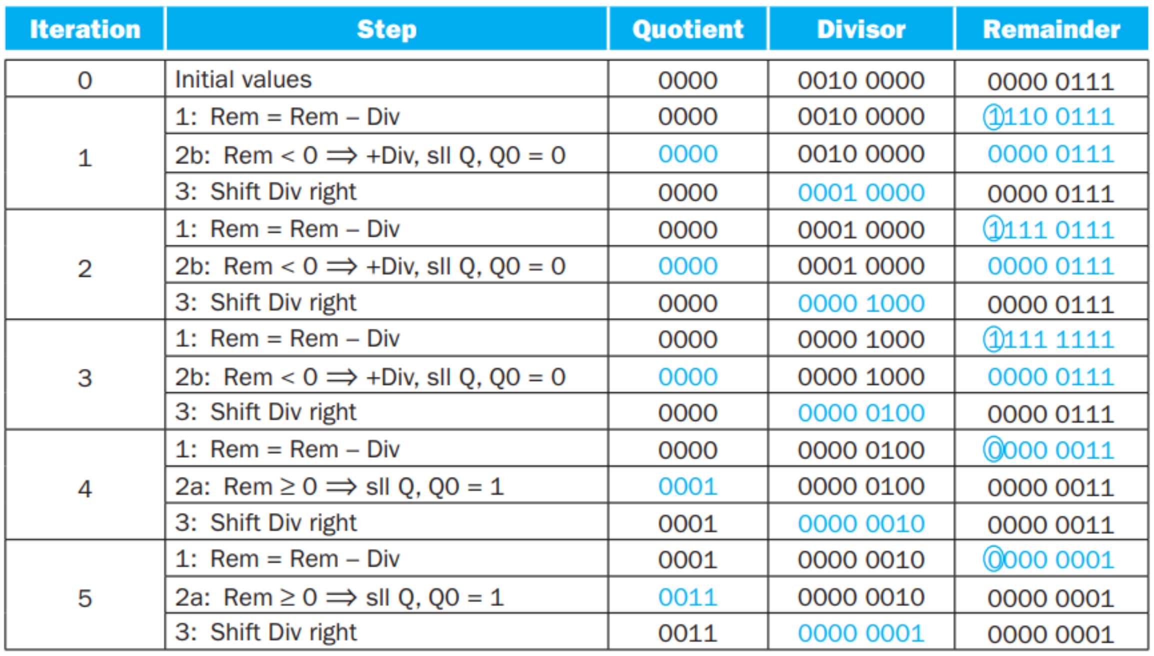


Figure 3.10 shows the value of each register for each of the steps, with the quotient being 3_{ten} and the remainder 1_{ten}. Notice that the test in step 2 of whether the remainder is positive or negative simply tests whether the sign bit of the Remainder register is a 0 or 1. The surprising requirement of this algorithm is that it takes $n + 1$ steps to get the proper quotient and remainder.



Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	①110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	①111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	①111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	⓪000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	⓪000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Scientific Notation: A notation that renders numbers with a single digit to the left of the decimal point.

Normalized: A number in floating-point notation that has no leading 0s.

Floating Point: Computer arithmetic that represents numbers in which the binary point is not fixed.

Fraction: The value, generally between 0 and 1, placed in the fraction field. The fraction is also called the mantissa.

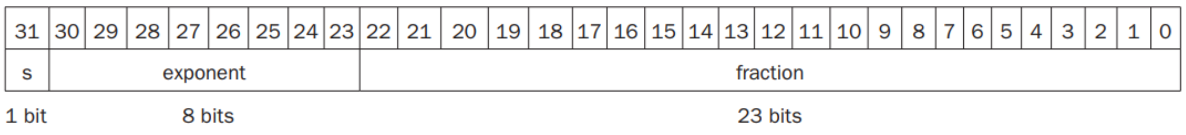
Exponent: In the numerical representation system of floating-point arithmetic, the value that is placed in the exponent field.

A designer of a floating-point representation must find a compromise between the size of the fraction and the size of the exponent, because a fixed word size means you must take a bit from one to add a bit to the other. This **trade-off** is between precision and range: increasing the size of the fraction enhances the precision of the fraction, while increasing the size of the exponent increases the range of numbers that can be represented.

Overflow (Floating-point): A situation in which a positive exponent becomes too large to fit in the exponent field.

Underflow (Floating-point): A situation in which a negative exponent becomes too large to fit in the exponent field.

Single precision: A floating-point value represented in a single 32- bit word.

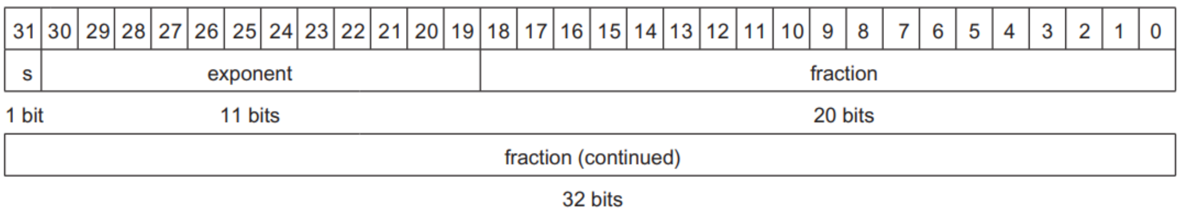


In general, floating-point numbers are of the form

$$(-1)^S \times F \times 2^E$$

F involves the value in the fraction field and E involves the value in the exponent field; the exact relationship to these fields will be spelled out soon. (We will shortly see that MIPS does something slightly more sophisticated.)

Double precision: A floating-point value represented in two 32-bit words.



Thus 00 … 00_{two} represents 0; the representation of the rest of the numbers uses the form from before with the hidden 1 added:

$$(-1)^S \times (1 + \text{Fraction}) \times 2^E$$

where the bits of the fraction represent a number between 0 and 1 and E specifies the value in the exponent field, to be given in detail shortly. If we number the bits of the fraction from *left to right* s1, s2, s3, …, then the value is

$$(-1)^S \times (1 + (s1 \times 2^{-1}) + (s2 \times 2^{-2}) + (s3 \times 2^{-3}) + (s4 \times 2^{-4}) + ...) \times 2^E$$

The desirable notation must therefore represent the most negative exponent as 00 … 00two and the most positive as 11 … 11two. Th is convention is called biased notation, with the bias being the number subtracted from the normal, unsigned representation to determine the real value.

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

Single Precision Bias: 127

Double Precision Bias: 1023

3.27 [20] <§3.5> IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern to represent -1.5625×10^{-1} assuming a version of this format, which uses an excess-16 format to store the exponent. Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision IEEE 754 standard.

A Basic MIPS Implementation

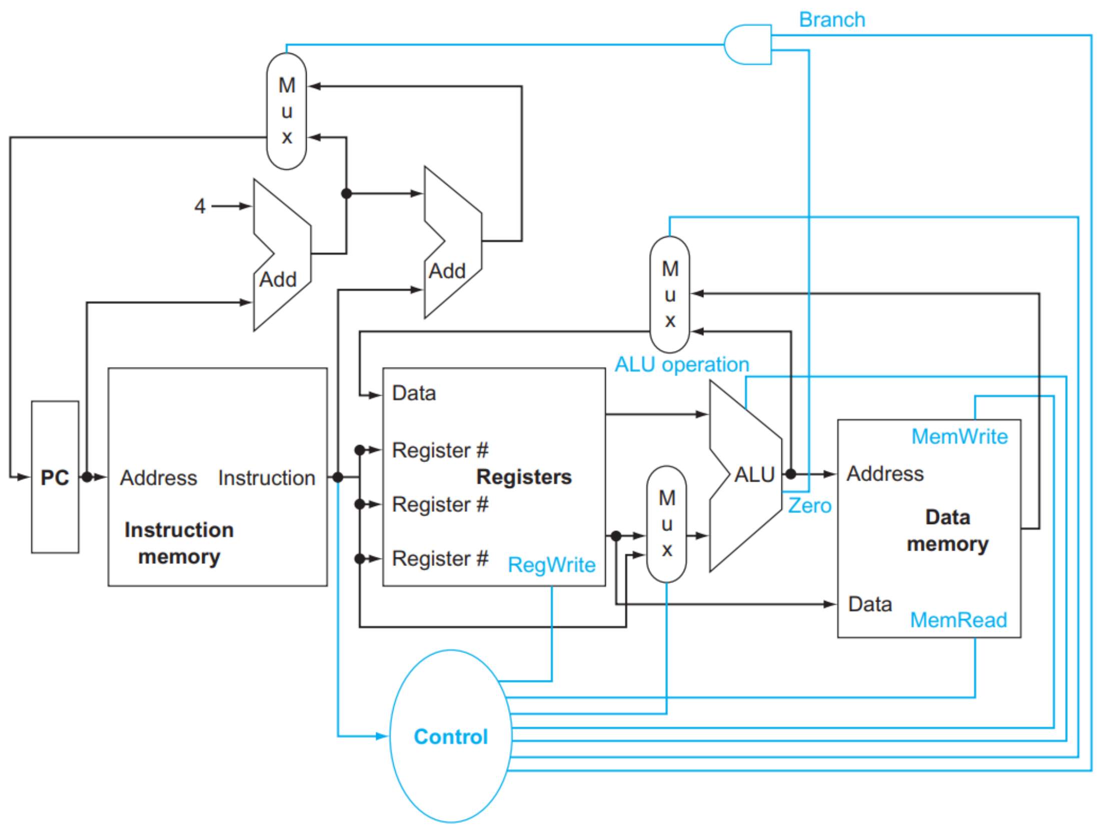
We will be examining an implementation that includes a subset of the core MIPS instruction set

1. The memory-reference instructions load word (lw) and store word (sw)
2. The arithmetic-logical instructions add, sub, AND, OR, and slt
3. The instructions branch equal (beq) and jump (j), which we add last.

For every instruction, the first two steps are identical:

1. **IF -** Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory.
2. **ID -** Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, we need to read only one register, but most other instructions require reading two registers.

After these two steps, the actions required to complete the instruction depend on the instruction class.



Combinational Element: An operational element, such as an AND gate or an ALU.

State Element: A memory element, such as a register or a memory.

Clocking Methodology: The approach used to determine when data is valid and stable relative to the clock.

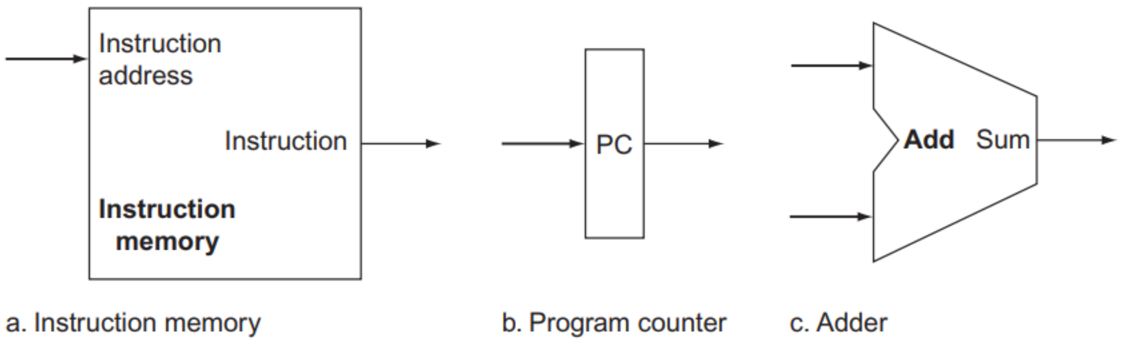
Edge-Triggered Clocking: A clocking scheme in which all state changes occur on a clock edge.

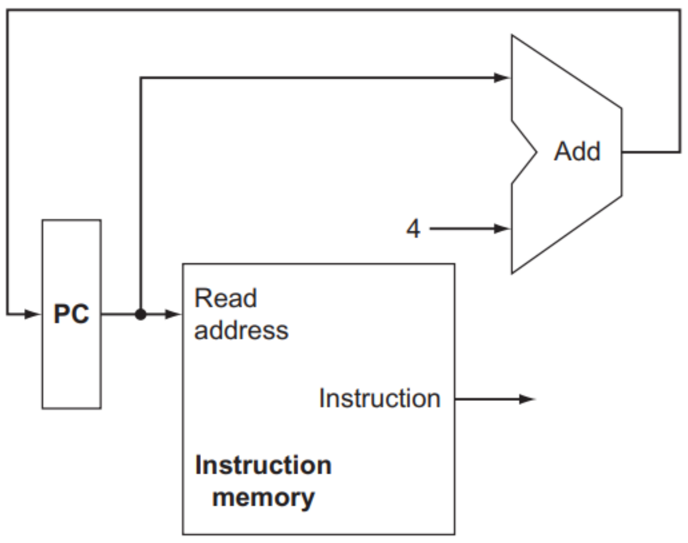
Control signal: A signal used for multiplexor selection or for directing the operation of a functional unit.
Data signal: Contains information that is operated on by a functional unit.

Asserted: The signal is logically high or true.
De-asserted: The signal is logically low or false.

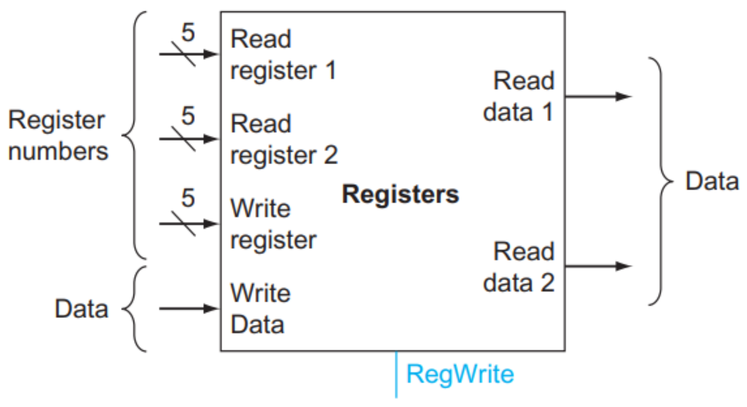
Datapath element: A unit used to operate on or hold data within a processor. In the MIPS implementation, the data-path elements include the instruction and data memories, the register file, the ALU, and adders.

Memory unit: To store the instructions of a program and supply instructions given an address.
Program counter (PC): The register containing the address of the instruction in the program being executed.
Adder: To increment the PC to the address of the next instruction.

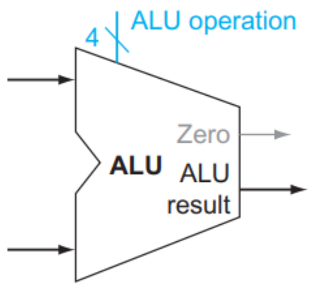




Register file: A state element that consists of a set of registers that can be read and written by supplying a register number to be accessed.

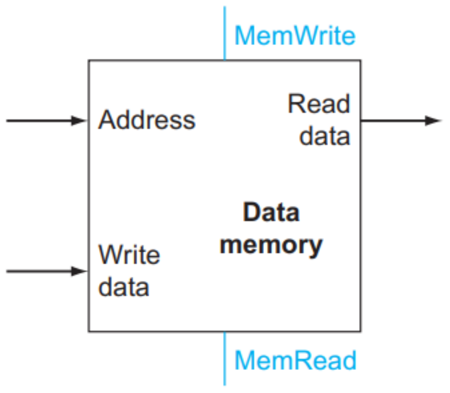


a. Registers

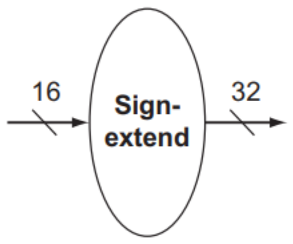


b. ALU

Sign-extend: A data-path item used to increase the size of a data item by replicating the high-order sign bit of the original data item in the highorder bits of the larger, destination data item.



a. Data memory unit



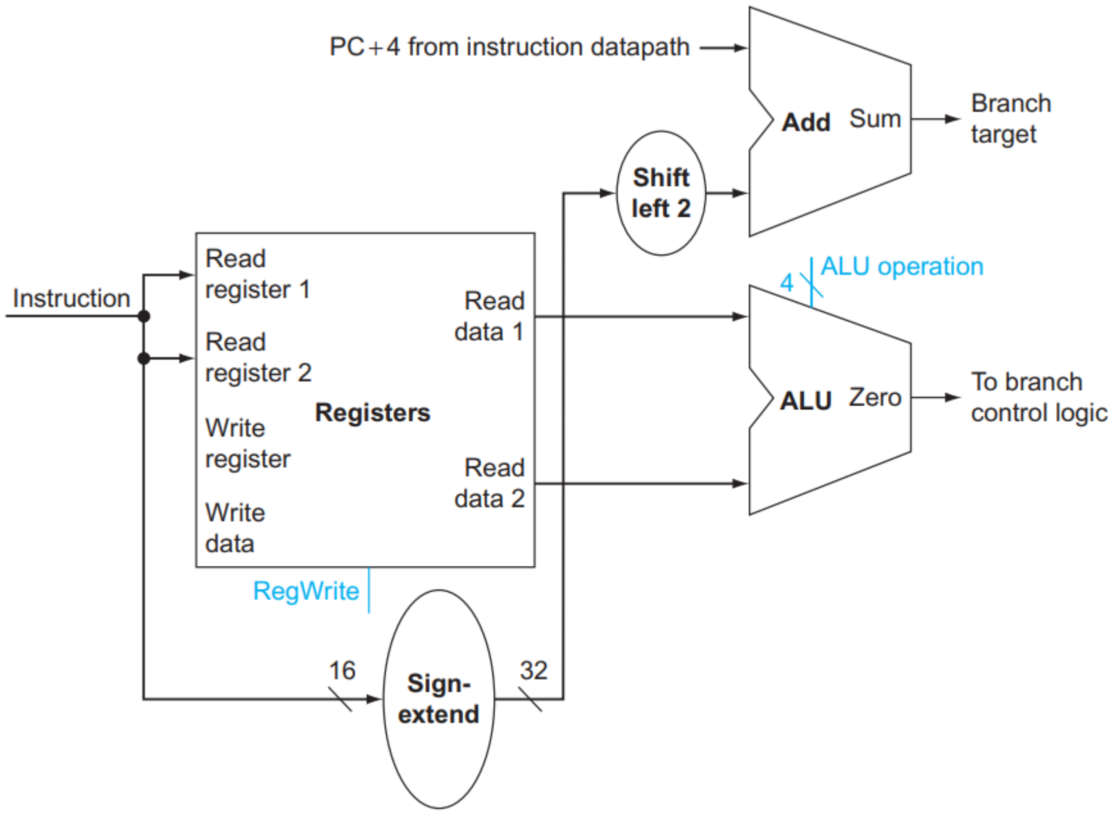
b. Sign extension unit

Branch Target Address: The address specified in a branch, which becomes the new program counter (PC) if the branch is taken. In the MIPS architecture the branch target is given by the sum of the offset field of the instruction and the address of the instruction following the branch.

Branch Taken: A branch where the branch condition is satisfied and the program counter (PC) becomes the branch target. All unconditional jumps are taken branches.

Branch Not Taken or (untaken branch): A branch where the branch condition is false and the program counter (PC) becomes the address of the instruction that sequentially follows the branch.

Branch: A type of branch where the instruction immediately following the branch is always executed, independent of whether the branch condition is true or false.

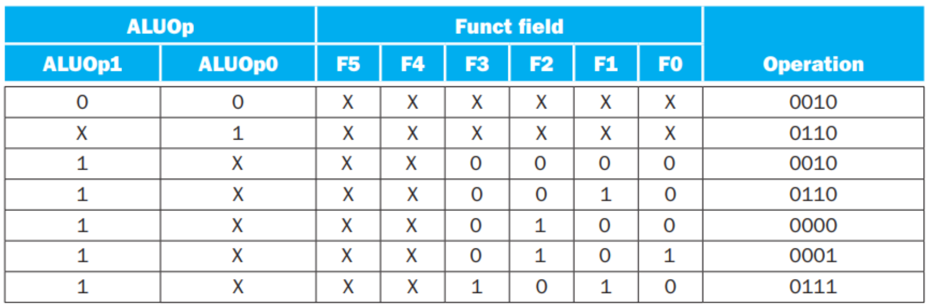


ALU Control

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

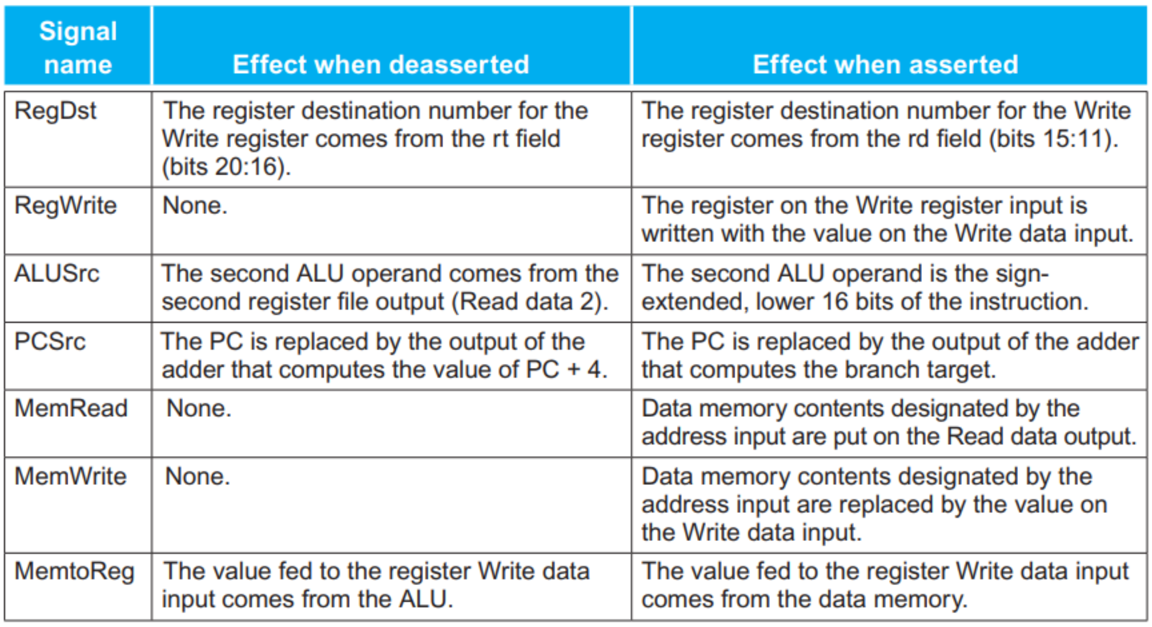
For branch equal, the ALU must perform a subtraction.

Main Control Unit

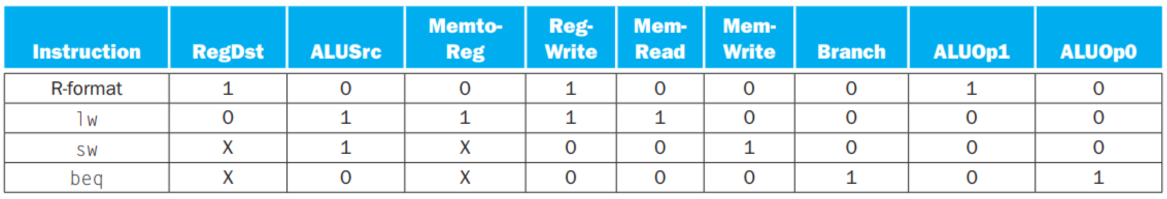


ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

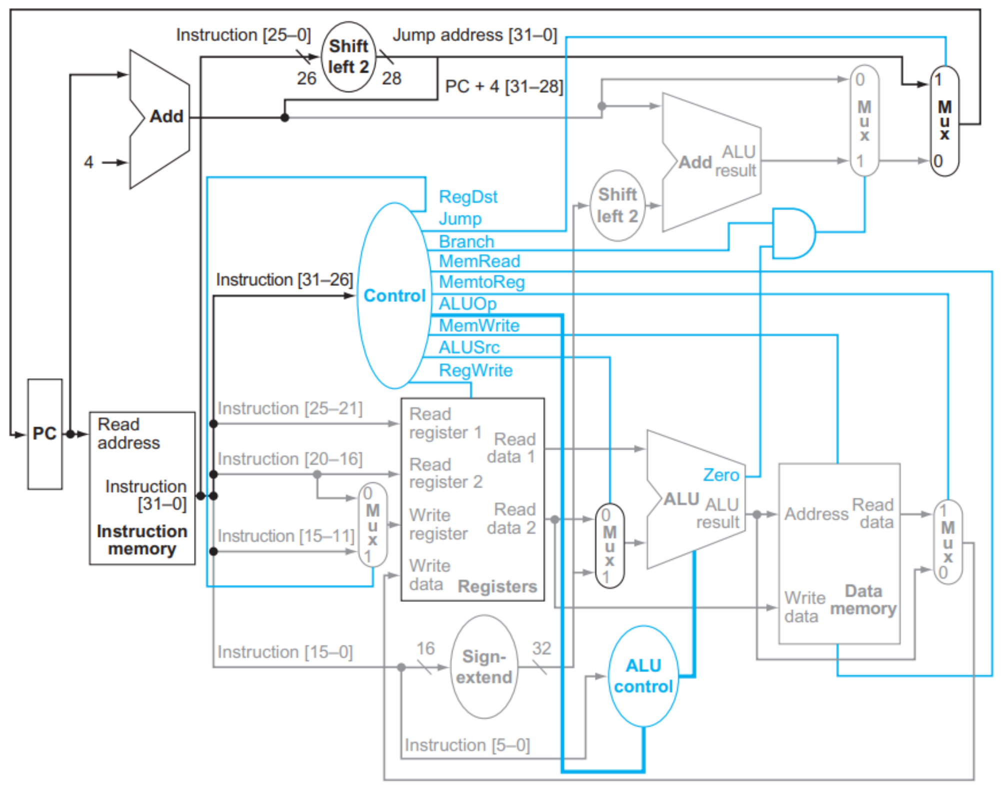
Effect of Control Lines (7 here + 2 from ALUop)



Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.



Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1



Pipelining: An implementation technique in which multiple instructions are overlapped in execution, much like an assembly line. All steps—called stages in pipelining—are operating concurrently. As long as we have separate resources for each stage, we can pipeline the tasks. The reason pipelining is faster for many loads is that everything is working in parallel, so more loads are finished per hour.

MIPS instructions classically take five steps:

1. **IF** Fetch instruction from memory.
2. **ID** Read registers while decoding the instruction. The regular format of MIPS instructions allows reading and decoding to occur simultaneously.
3. **EX** Execute the operation or calculate an address.
4. **MEM** Access an operand in data memory.
5. **WB** Write the result into a register.

Pipelining improves performance by increasing instruction throughput, as opposed to decreasing the execution time of an individual instruction, but instruction throughput is the important metric because real programs execute billions of instructions.

Why MIPS is designed for pipeline execution:

1. All MIPS instructions are the same length. This restriction makes it much easier to fetch instructions in the first pipeline stage and to decode them in the second stage.
2. MIPS has only a few instruction formats, with the source register fields being located in the same place in each instruction.
3. Memory operands only appear in loads or stores in MIPS.
4. Operands must be aligned in memory.

There are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called **hazards**, and there are three different types.

Structural Hazard: When a planned instruction cannot execute in the proper clock cycle because the hardware does not support the combination of instructions that are set to execute.

Data Hazard: Also called a pipeline data hazard. When a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available.

Forwarding: Also called bypassing. A method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer-visible registers or memory.

Load-use Data Hazard: A specific form of data hazard in which the data being loaded by a load instruction has not yet become available when it is needed by another instruction.

Pipeline stall: Also called **bubble**. A stall initiated in order to resolve a hazard.

Control Hazard: Also called branch hazard. When the proper instruction cannot execute in the proper pipeline clock cycle because the instruction that was fetched is not the one that is needed; that is, the flow of instruction addresses is not what the pipeline expected.

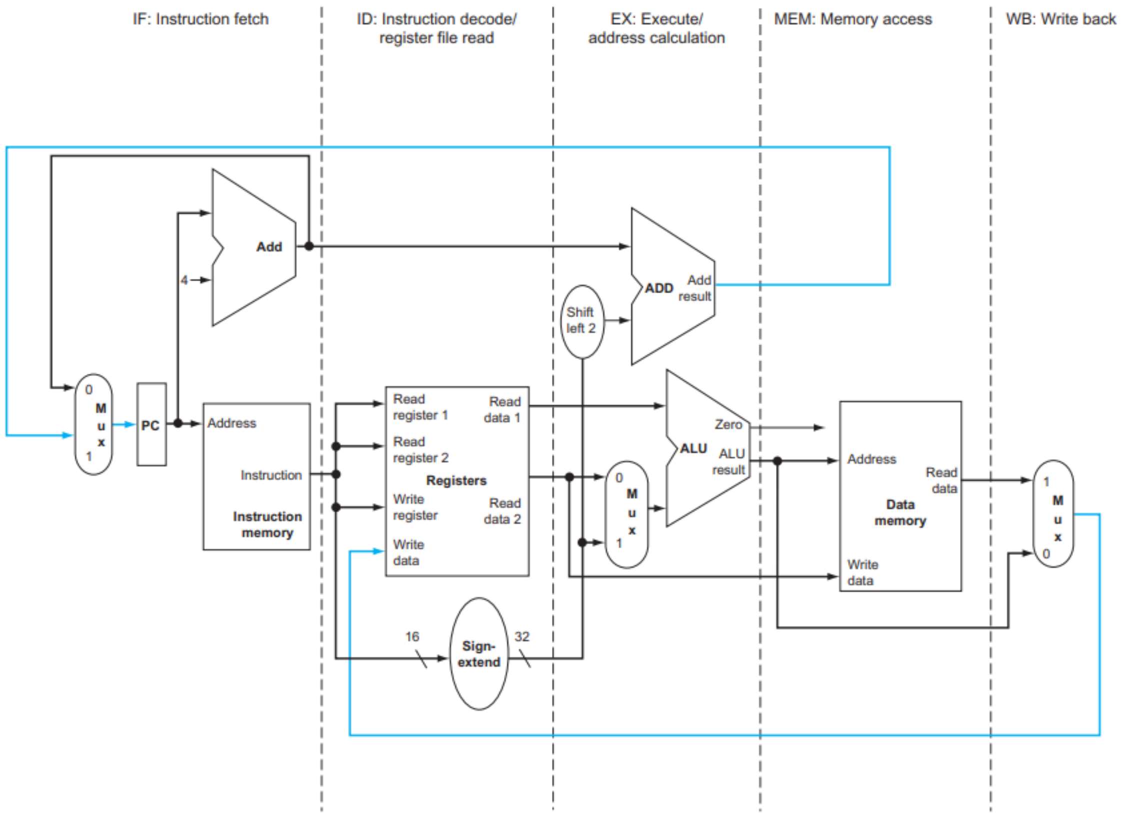
Computers use prediction to handle branches. One simple approach is to predict always that branches will be untaken. When you’re right, the pipeline proceeds at full speed. Only when branches are taken does the pipeline stall.

Branch Prediction: A method of resolving a branch hazard that assumes a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome.
One popular approach to dynamic prediction of branches is keeping a history for each branch as taken or untaken, and then using the recent past behaviour to predict the future.

There is a third approach to the control hazard, called **delayed decision**. Called the delayed branch in computers, and mentioned above, this is the solution actually used by the MIPS architecture. The delayed branch always executes the next sequential instruction, with the branch taking place after that one instruction delay.

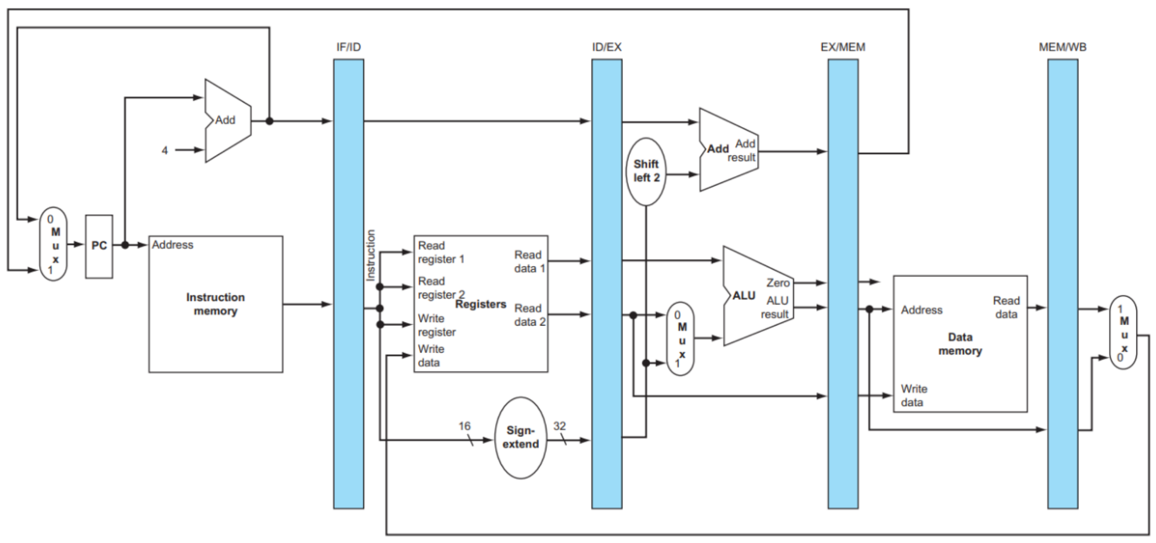
Pipelining is a technique that exploits **parallelism** among the instructions in a sequential instruction stream.

Latency (pipeline): The number of stages in a pipeline or the number of stages between two instructions during execution.



Instructions and data move generally from left to right through the five stages as they complete execution. There are, however, two exceptions to this left -to-right flow of instructions:

1. The write-back stage, which places the result back into the register file in the middle of the data-path.
2. The selection of the next value of the PC, choosing between the incremented PC and the branch address from the MEM stag.



Two pairs of hazard conditions:

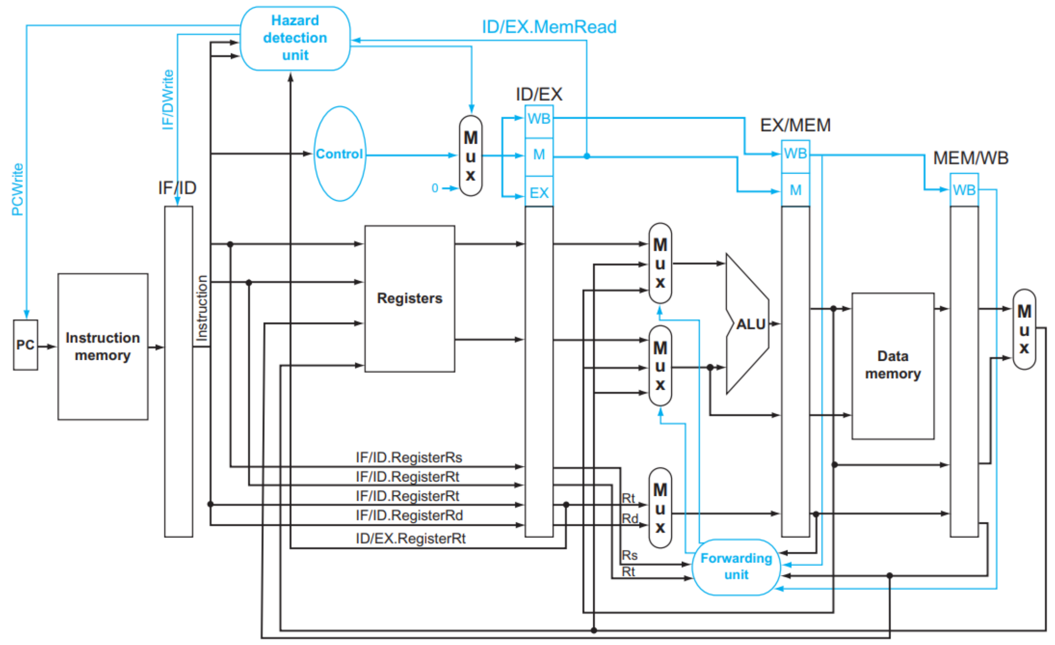
1a. EX/MEM.RegisterRd = ID/EX.RegisterRs

1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs

2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Nop: An instruction that does no operation to change state.



Instruction-Level Parallelism: The parallelism among instructions.

Multiple issue: A scheme whereby multiple instructions are launched in one clock cycle.

Static Multiple Issue: An approach to implementing a multiple-issue processor where many decisions are made by the compiler before execution.

Dynamic Multiple Issue: An approach to implementing a multiple-issue processor where many decisions are made during execution by the processor.

Issue slots: The positions from which instructions could issue in a given clock cycle; by analogy, these correspond to positions at the starting blocks for a sprint.

Speculation: An approach whereby the compiler or processor guesses the outcome of an instruction to remove it as a dependence in executing other instructions.

Issue Packet: The set of instructions that issues together in one clock cycle; the packet may be determined statically by the compiler or dynamically by the processor.

Very Long Instruction Word (VLIW): A style of instruction set architecture that launches many operations that are defined to be independent in a single wide instruction, typically with many separate opcode fields.

Use Latency: Number of clock cycles between a load instruction and an instruction that can use the result of the load without stalling the pipeline.

Loop Unrolling: A technique to get more performance from loops that access arrays, in which multiple copies of the loop body are made and instructions from different iterations are scheduled together.

Register Renaming: The renaming of registers by the compiler or hardware to remove anti-dependences.

Anti-dependence: Also called name dependence. An ordering forced by the reuse of a name, typically a register, rather than by a true dependence that carries a value between two instructions.

Superscalar: An advanced pipelining technique that enables the processor to execute more than one instruction per clock cycle by selecting them during execution.

Dynamic Pipeline Scheduling: Hardware support for reordering the order of instruction execution so as to avoid stalls.

Commit Unit: The unit in a dynamic or out-of-order execution pipeline that decides when it is safe to release the result of an operation to programmer visible registers and memory.

Reservation Station: A buffer within a functional unit that holds the operands and the operation.

Reorder Buffer: The buffer that holds results in a dynamically scheduled processor until it is safe to store the results to memory or a register.

Out-Of-Order Execution: A situation in pipelined execution when an instruction blocked from executing does not cause the following instructions to wait.

In-Order Commit: A commit in which the results of pipelined execution are written to the programmer visible state in the same order that instructions are fetched.

Processor	ARM A8	Intel Core i7 920
Market	Personal Mobile Device	Server, Cloud
Thermal design power	2 Watts	130 Watts
Clock rate	1 GHz	2.66 GHz
Cores/Chip	1	4
Floating point?	No	Yes
Multiple Issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline Stages	14	14
Pipeline schedule	Static In-order	Dynamic Out-of-order with Speculation
Branch prediction	2-level	2-level
1st level caches / core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D
2nd level cache / core	128 - 1024 KiB	256 KiB
3rd level cache (shared)	–	2 - 8 MiB

FIGURE 4.74 Specification of the ARM Cortex-A8 and the Intel Core i7 920.

Principle of Locality: The principle of locality states that programs access a relatively small portion of their address space at any instant of time.

Temporal Locality: The principle stating that if a data location is referenced then it will tend to be referenced again soon.

Spatial locality: The locality principle stating that if a data location is referenced, data locations with nearby addresses will tend to be referenced soon.

Memory Hierarchy: A structure that uses multiple levels of memories; as the distance from the processor increases, the size of the memories and the access time both increases. The faster memories are more expensive per bit than the slower memories and thus are smaller.

Block (or Line): The minimum unit of information that can be either present or not present in a cache.

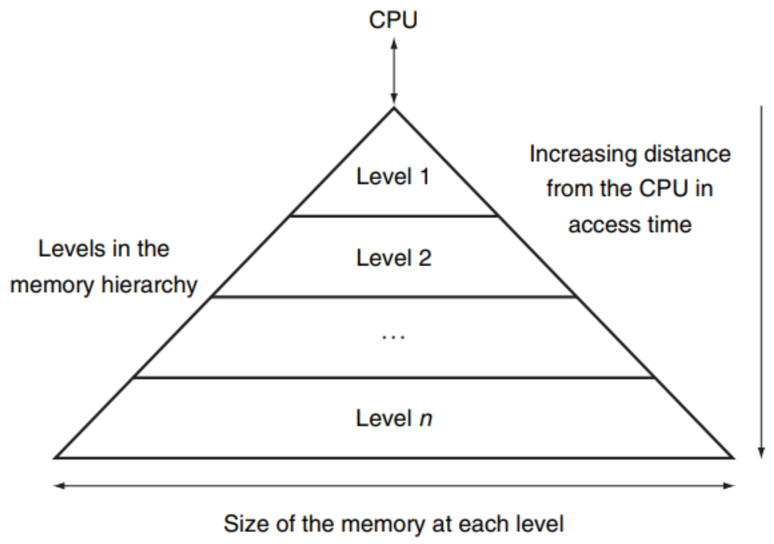
Hit Rate: The fraction of memory accesses found in a level of the memory hierarchy.

Miss Rate: The fraction of memory accesses not found in a level of the memory hierarchy.

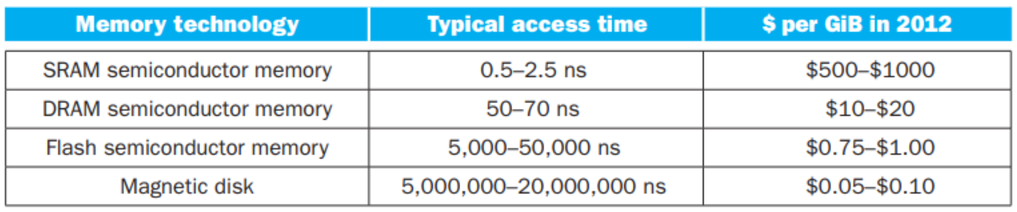
Hit Rate + Miss Rate = 1

Hit Time: The time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or a miss.

Miss Penalty: The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor.


CPU
Level 1
Level 2
...
Level n
Increasing distance from the CPU in access time
Levels in the memory hierarchy
Size of the memory at each level

Memory Technologies



Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

Track: One of thousands of concentric circles that makes up the surface of a magnetic disk.

Sector: One of the segments that make up a track on a magnetic disk. A sector is the smallest amount of information that is read or written on a disk.

Cylinder: The term cylinder is used to refer to all the tracks under the heads at a given point on all surfaces.

Seek: The process of positioning a read / write head over the proper track on a disk.

Seek Time: The time to move the head to the desired track.

Rotational latency: Also called rotational delay. The time required for the desired sector of a disk to rotate under the read / write head. Usually assumed to be half the rotation time.

Transfer time: The time to transfer a block of bits. The transfer time is a function of the sector size, the rotation speed, and the recording density of a track.

Cache: A safe place for hiding or storing things.

Direct-mapped cache: A cache structure in which each memory location is mapped to exactly one location in the cache. Almost all direct-mapped caches use this mapping to find a block: (Block address) modulo (Number of blocks in the cache).

Tag: A field in a table used for a memory hierarchy that contains the address information required to identify whether the associated block in the hierarchy corresponds to a requested word.

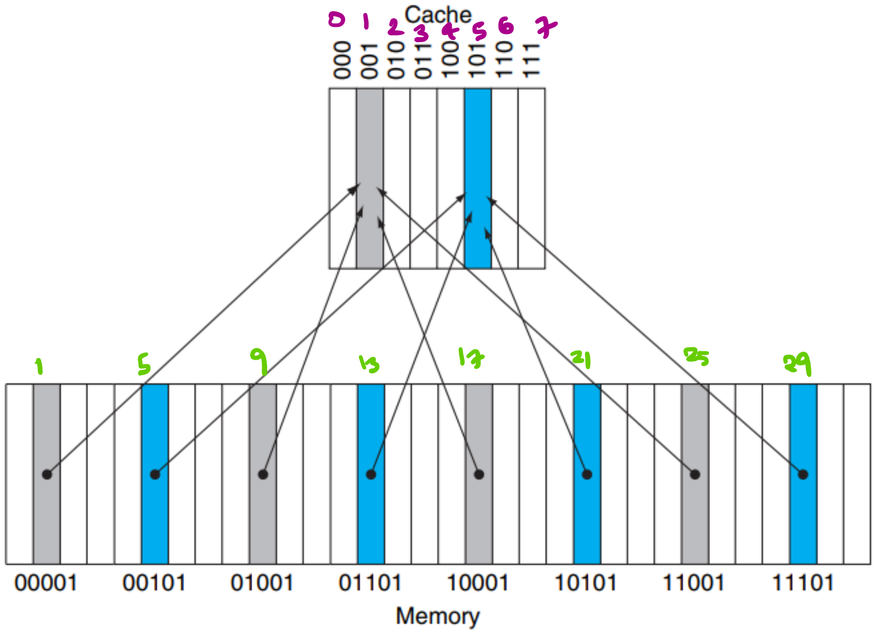


FIGURE 5.8 A direct-mapped cache with eight entries showing the addresses of memory words between 0 and 31 that map to the same cache locations. Because there are eight words in the cache, an address X maps to the direct-mapped cache word X modulo 8. That is, the low-order $\log_2(8) = 3$ bits are used as the cache index. Thus, addresses 00001_{two}, 01001_{two}, 10001_{two}, and 11001_{two} all map to entry 001_{two} of the cache, while addresses 00101_{two}, 01101_{two}, 10101_{two}, and 11101_{two} all map to entry 101_{two} of the cache.

Valid Bit: A field in the tables of a memory hierarchy that indicates that the associated block in the hierarchy contains valid data.

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

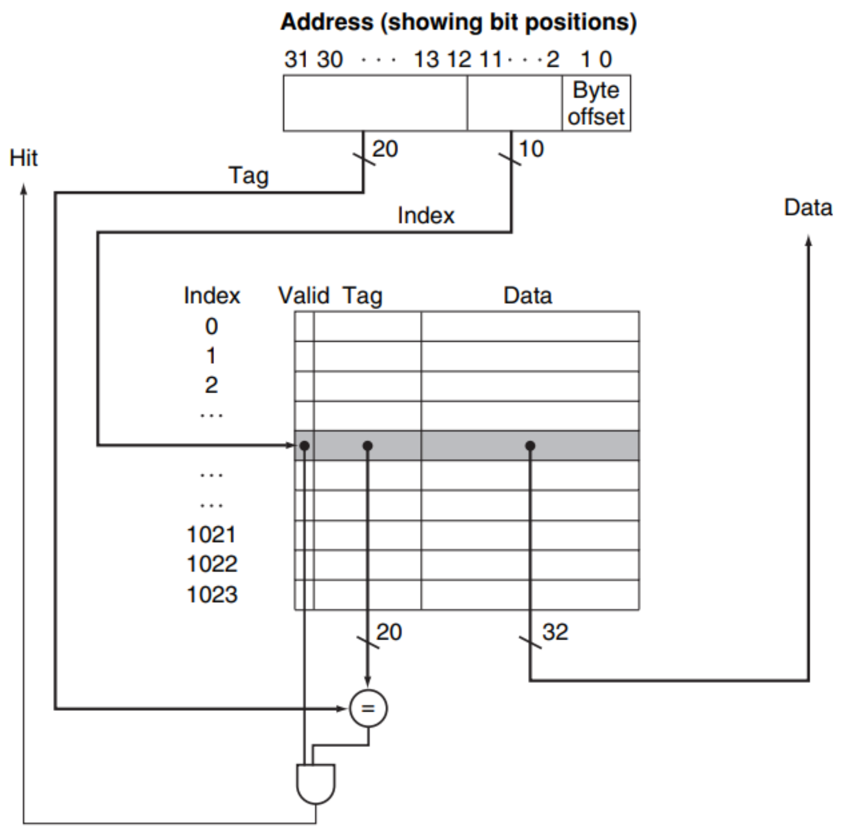
d. After handling a miss of address (10000_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})



Cache Miss: A request for data from the cache that cannot be filled because the data is not present in the cache.

Write-Through: A scheme in which writes always update both the cache and the next lower level of the memory hierarchy, ensuring that data is always consistent between the two.

Write Buffer: A queue that holds data while the data is waiting to be written to memory.

Write-Back: A scheme that handles writes by updating values only to the block in the cache, then writing the modified block to the lower level of the hierarchy when the block is replaced.

Split Cache: A scheme in which a level of the memory hierarchy is composed of two independent caches that operate in parallel with each other, with one handling instructions and one handling data.

Error Detection Code: A code that enables the detection of an error in data, but not the precise location and, hence, correction of the error.