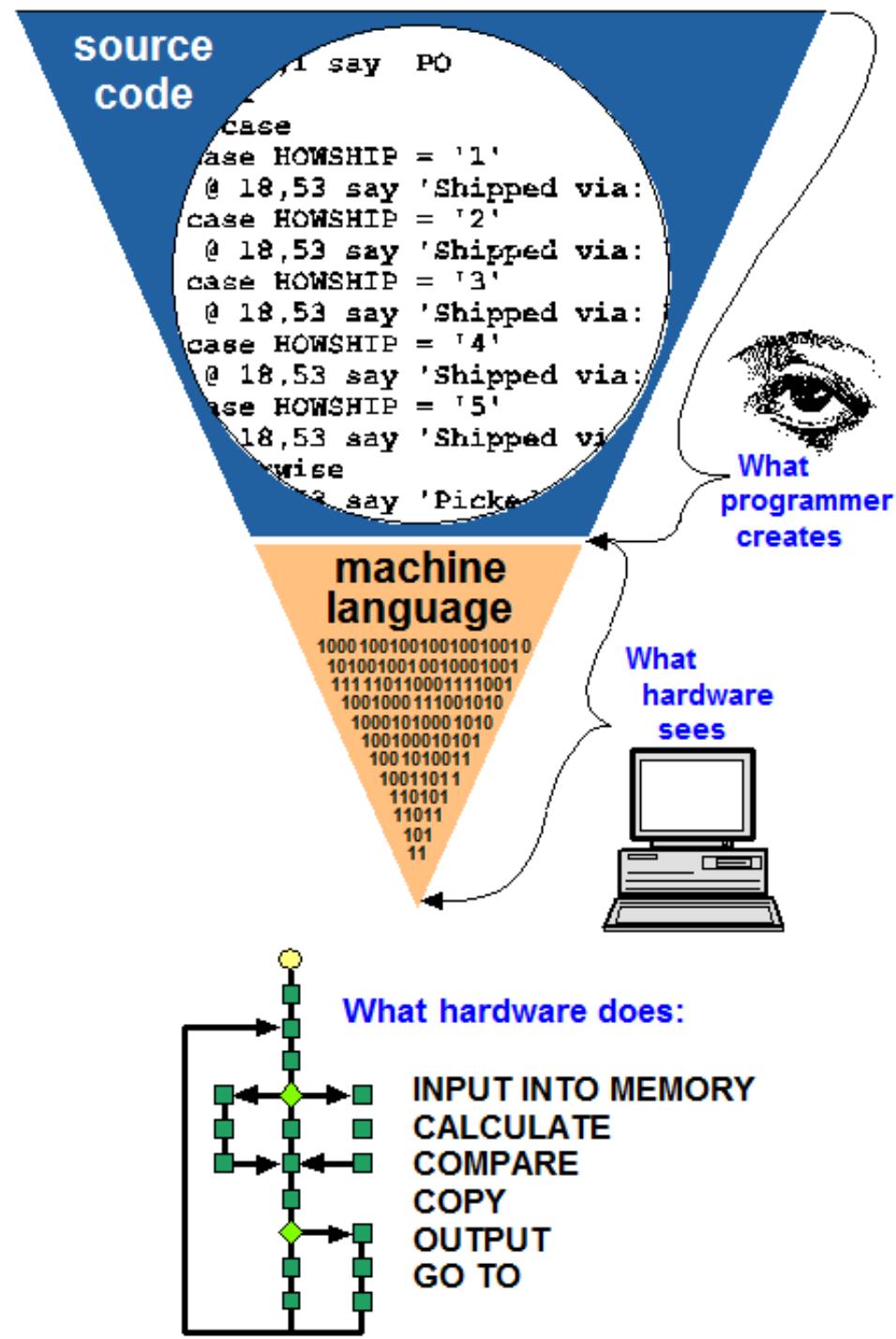


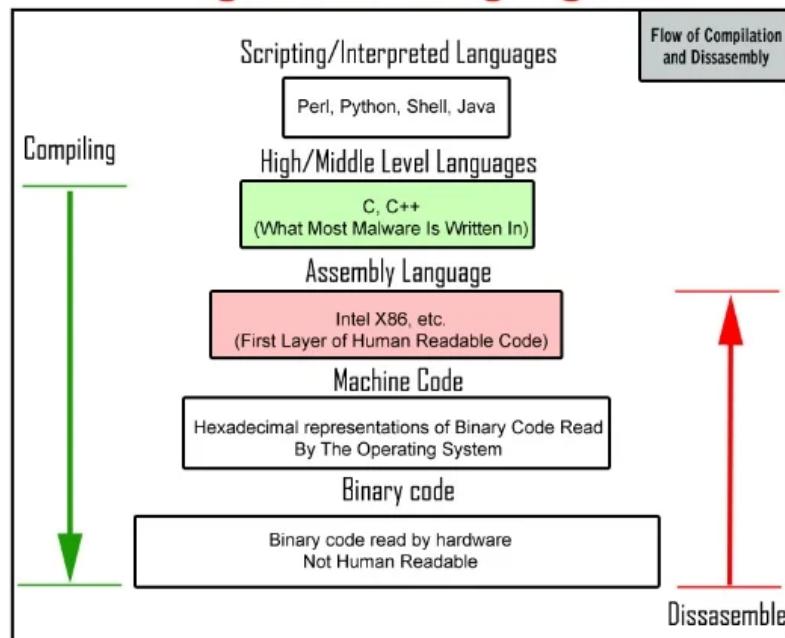
Computer Organization and Architecture

Programming Language

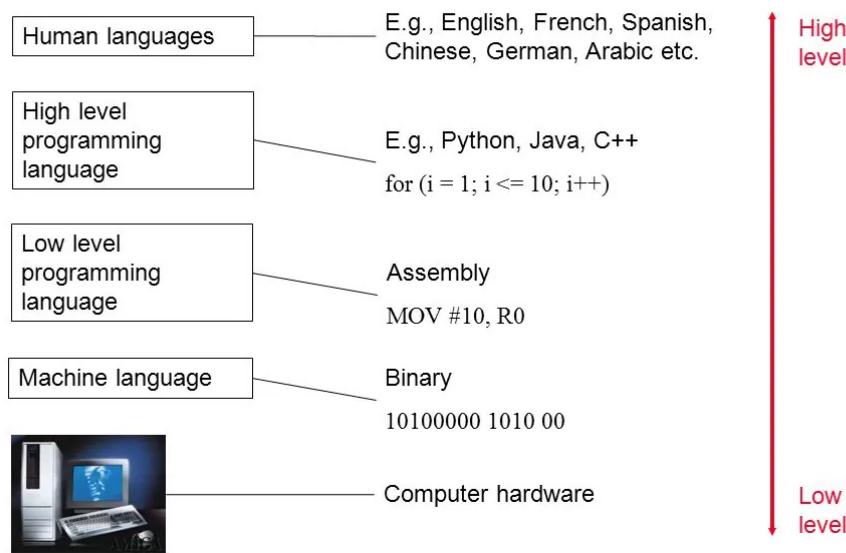
SOURCE CODE TO MACHINE LANGUAGE



High Level Languages

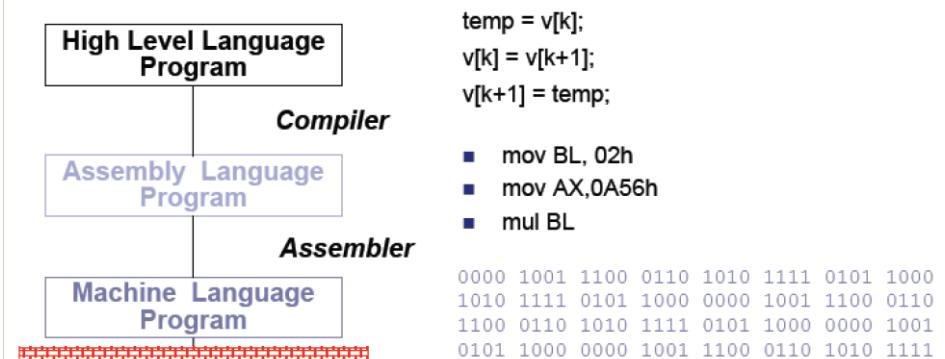


High Vs. Low Level Languages



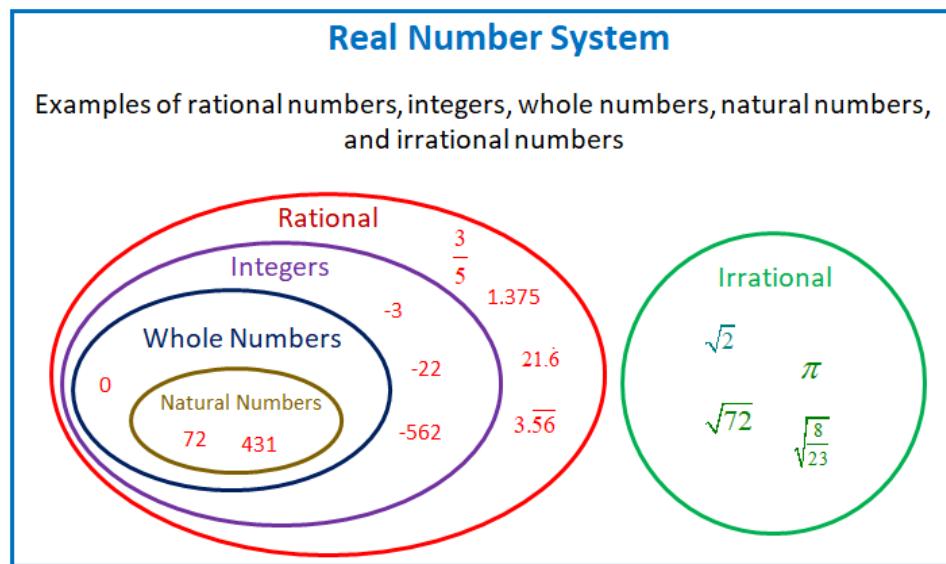
James Tam

Levels of Representation



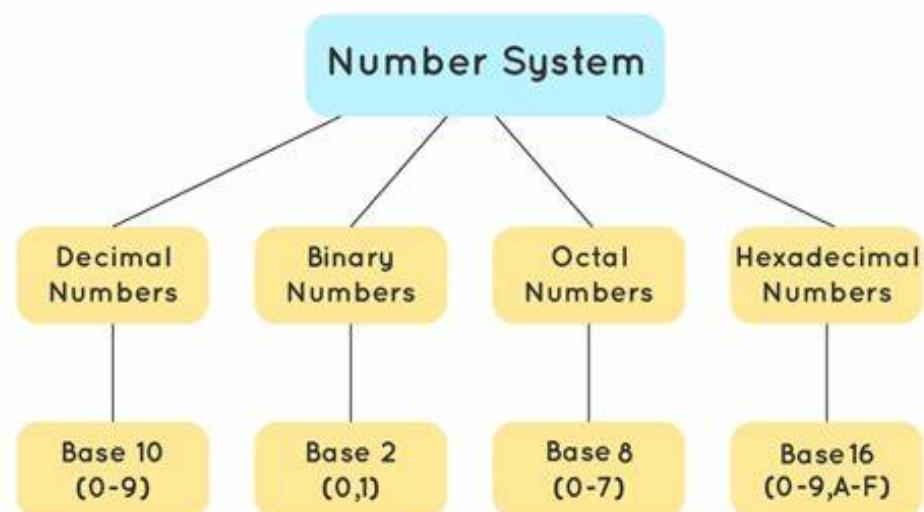
Number System

Real Number System



Number System Conversion

Types of Number System



Number Systems Conversion Chart

Binary																
Place	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Weight	2048	1024	512	256	128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625

Binary, Hex, and Octal Conversions

Binary	Octal	Hexadecimal	Decimal
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

Methodology

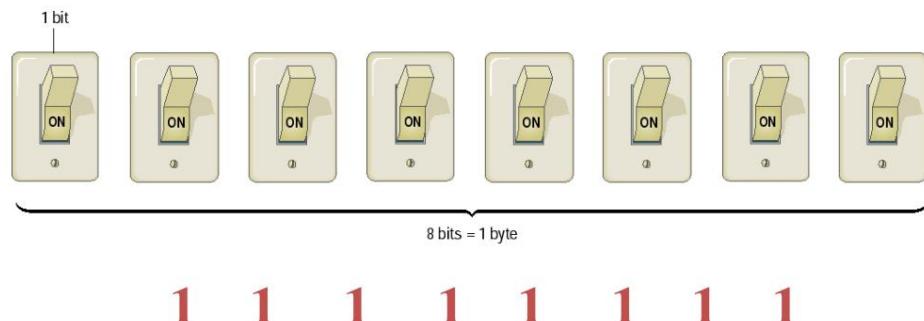
- Convert from decimal to binary
Divide the decimal by the largest binary weight it is divisible by and place a "1" in that column. Then select the next largest weight, if it is divisible put a "1" in that column otherwise place a "0" in the column. Continue until all the columns have either a "1" or "0" resulting in a binary expression.
- Convert from decimal to hex.
Convert to binary first, then group the binary in groups of 4 beginning on the right working to the left. For each group determine the hex value based on the table to the left.
- Convert to octal
Convert to binary first, then group the binary in groups of 3 beginning on the right working to the left. For each group determine the octal value based on the table to the left.

Binary Number System

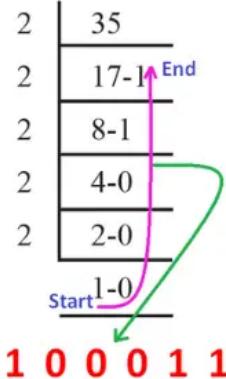
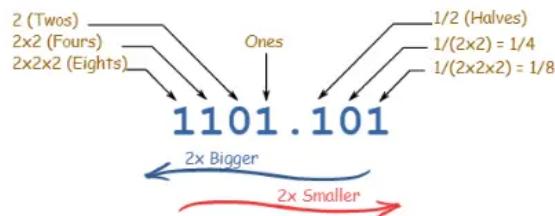
Number System

Bits and Bytes

- A single unit of data is called a bit, having a value of 1 or 0.
- Computers work with collections of bits, grouping them to represent larger pieces of data, such as letters of the alphabet.
- Eight bits make up one byte. A byte is the amount of memory needed to store one alphanumeric character.
- With one byte, the computer can represent one of 256 different symbols or characters.



What is the Binary Number System?



Electrical 4 U

Complement

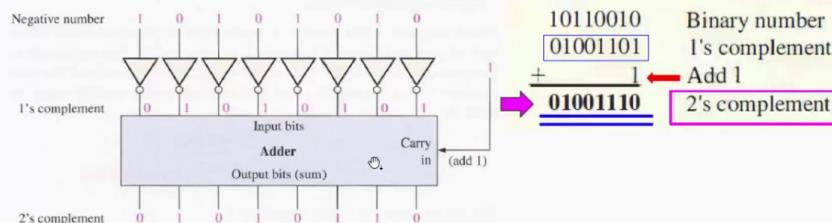
2.5) 1'S and 2'S Complements of Binary Numbers

Finding the 2's Complement

The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

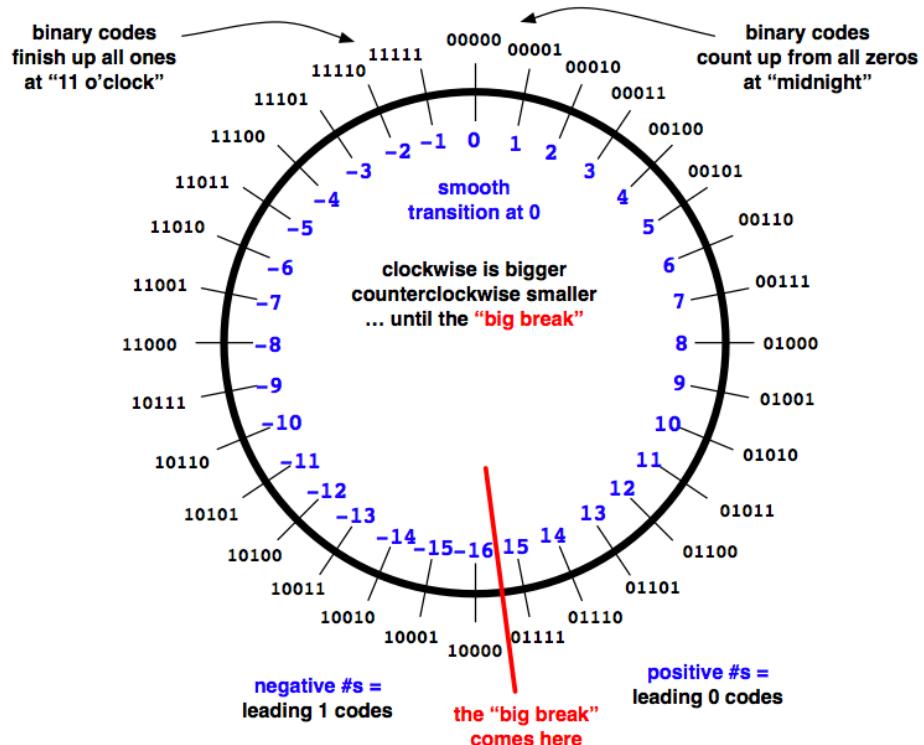
$$\text{2's complement} = (\text{1's complement}) + 1$$

Example 17: Find the 2's complement of 10110010



Source: Digital Fundamentals by Thomas L. Floyd.

27



Complements of numbers

(r-1)'s Complement

- Given a number N in base r having n digits,
- the $(r- 1)$'s complement of N is defined as

$$(r^n - 1) - N$$

- For decimal numbers the base or $r = 10$ and $r- 1 = 9$,

- so the 9's complement of N is $(10^n - 1) - N$

- $99999\dots\dots - N$

9	9	9	9	9
Digit n	Digit n-1	Next digit	Next digit	First digit

Complements

- There are two types of complements for each base- r system: the radix complement and diminished radix complement.

→ the r 's complement and the second as the $(r - 1)$'s complement.

Diminished Radix Complement

Given a number N in case r having n digits, the $(r - 1)$'s complement of N is defined as $(r^n - 1) - N$. For decimal numbers, $r = 10$ and $r - 1 = 9$, so the 9's complement of N is $(10^n - 1) - N$.

Example:

The 9's complement of 546700 is $999999 - 546700 = 453299$.

The 9's complement of 012398 is $999999 - 012398 = 987601$.

- For binary numbers, $r = 2$ and $r - 1 = 1$, so the 1's complement of N is $(2^n - 1) - N$.

Example:

The 1's complement of 1011000 is 0100111

The 1's complement of 0101101 is 1010010

Precision floating-point

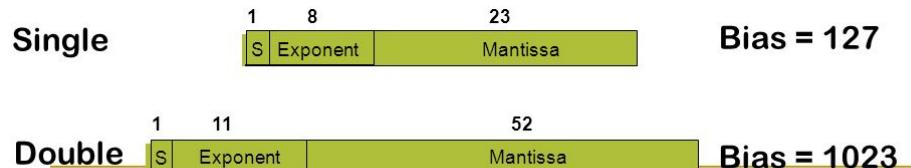
Floating point numbers – the IEEE standard

■ IEEE Standard 754

- ❑ Most (but not all) computer manufacturers use IEEE-754 format
 - ❑ Number represented:

$$(-1)^S * (1.M)^* 2^{(E - \text{Bias})}$$

2 main formats: single and double



Floating-Point Numbers

- ❖ Examples of floating-point numbers in base 10 ...
 - ✧ 5.341×10^3 , 0.05341×10^5 , -2.013×10^{-1} , -201.3×10^{-3}
 - ❖ Examples of floating-point numbers in base 2 ...
 - ✧ 1.00101×2^{23} , 0.0100101×2^{25} , -1.101101×2^{-3} , -1101.101×2^{-6}
 - ✧ Exponents are kept in decimal for clarity
 - ✧ The binary number $(1101.101)_2 = 2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-3} = 13.625$
 - ❖ Floating-point numbers should be normalized
 - ✧ Exactly one non-zero digit should appear before the point
 - In a decimal number, this digit can be from 1 to 9
 - In a binary number, this digit should be 1
 - ✧ Normalized FP Numbers: 5.341×10^3 and -1.101101×2^{-3}
 - ✧ NOT Normalized: 0.05341×10^5 and -1101.101×2^{-6}

Normalized Floating Point Numbers

- ❖ For a normalized floating point number (S, E, F)

$$\begin{array}{|c|c|c|} \hline S & E & F = f_1 f_2 f_3 f_4 \dots \\ \hline \end{array}$$

- ❖ Significand is equal to $(1.F)_2 = (1.f_1 f_2 f_3 f_4 \dots)_2$

❖ IEEE 754 assumes hidden 1. (not stored) for normalized numbers

❖ Significand is 1 bit longer than fraction

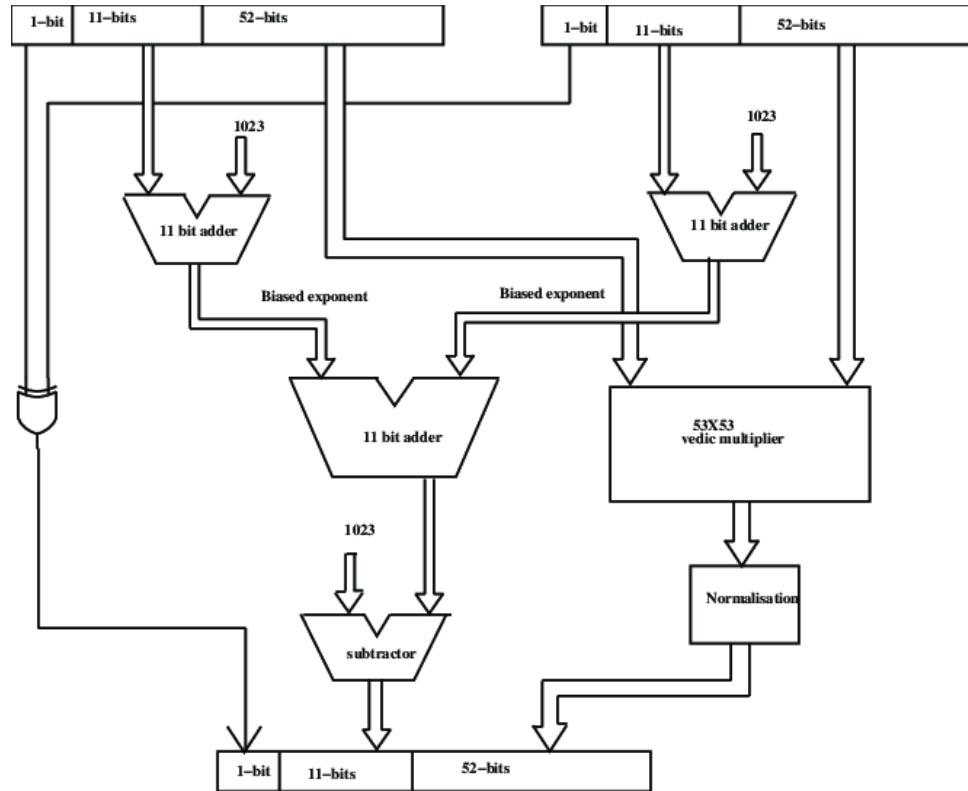
- ❖ Value of a Normalized Floating Point Number is

$$(-1)^S \times (1.F)_2 \times 2^{\text{val}(E)}$$

$$(-1)^S \times (1.f_1 f_2 f_3 f_4 \dots)_2 \times 2^{\text{val}(E)}$$

$$(-1)^S \times (1 + f_1 \times 2^{-1} + f_2 \times 2^{-2} + f_3 \times 2^{-3} + f_4 \times 2^{-4} \dots)_2 \times 2^{\text{val}(E)}$$

$(-1)^S$ is 1 when S is 0 (positive), and -1 when S is 1 (negative)



ASCII

ASCII control characters		ASCII printable characters												Extended ASCII characters											
00	NULL (Null character)	32	space	64	@	96	`	128	Ç	160	á	192	l	224	ó										
01	SOH (Start of Header)	33	!	65	A	97	a	129	ü	161	í	193	ł	225	þ										
02	STX (Start of Text)	34	"	66	B	98	b	130	é	162	ó	194	ł	226	ö										
03	ETX (End of Text)	35	#	67	C	99	c	131	à	163	ú	195	ñ	227	ô										
04	EOT (End of Trans.)	36	\$	68	D	100	d	132	â	164	ñ	196	—	228	ô										
05	ENQ (Enquiry)	37	%	69	E	101	e	133	à	165	N	197	+	229	ö										
06	ACK (Acknowledgement)	38	&	70	F	102	f	134	å	166	º	198	ä	230	µ										
07	BEL (Bell)	39	'	71	G	103	g	135	ç	167	º	199	À	231	þ										
08	BS (Backspace)	40	(72	H	104	h	136	é	168	¿	200	£	232	þ										
09	HT (Horizontal Tab)	41)	73	I	105	i	137	ë	169	®	201	™	233	ú										
10	LF (Line feed)	42	*	74	J	106	j	138	è	170	¬	202	§	234	ó										
11	VT (Vertical Tab)	43	+	75	K	107	k	139	í	171	½	203	¶	235	ú										
12	FF (Form feed)	44	,	76	L	108	l	140	í	172	¼	204	·	236	ý										
13	CR (Carriage return)	45	-	77	M	109	m	141	í	173	í	205	=	237	Ý										
14	SO (Shift Out)	46	.	78	N	110	n	142	À	174	«	206	†	238	-										
15	SI (Shift In)	47	/	79	O	111	o	143	Á	175	»	207	¤	239	.										
16	DLE (Data link escape)	48	0	80	P	112	p	144	É	176	™	208	◊	240	≡										
17	DC1 (Device control 1)	49	1	81	Q	113	q	145	æ	177	¤	209	¤	241	±										
18	DC2 (Device control 2)	50	2	82	R	114	r	146	Æ	178	¤	210	È	242	≡										
19	DC3 (Device control 3)	51	3	83	S	115	s	147	ò	179	¤	211	È	243	≡										
20	DC4 (Device control 4)	52	4	84	T	116	t	148	ó	180	—	212	È	244	¶										
21	NAK (Negative acknowl.)	53	5	85	U	117	u	149	ò	181	À	213	í	245	§										
22	SYN (Synchronous idle)	54	6	86	V	118	v	150	ó	182	À	214	í	246	÷										
23	ETB (End of trans. block)	55	7	87	W	119	w	151	ù	183	À	215	í	247	.										
24	CAN (Cancel)	56	8	88	X	120	x	152	ÿ	184	®	216	†	248	°										
25	EM (End of medium)	57	9	89	Y	121	y	153	Ö	185	—	217	„	249	..										
26	SUB (Substitute)	58	:	90	Z	122	z	154	Ù	186	—	218	„	250	.										
27	ESC (Escape)	59	;	91	[123	{	155	ø	187	†	219	■	251	†										
28	FS (File separator)	60	<	92	\	124		156	£	188	§	220	■	252	„										
29	GS (Group separator)	61	=	93]	125	}	157	Ø	189	¢	221	■	253	‡										
30	RS (Record separator)	62	>	94	^	126	~	158	×	190	¥	222	—	254	■										
31	US (Unit separator)	63	?	95	—			159	f	191	†	223	■	255	nbsp										
127	DEL (Delete)																								

Representing Text

ASCII Example

ASCII Code Chart																
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	~	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	P	q	r	s	t	u	v	w	x	y	z	{		}	—	DEL



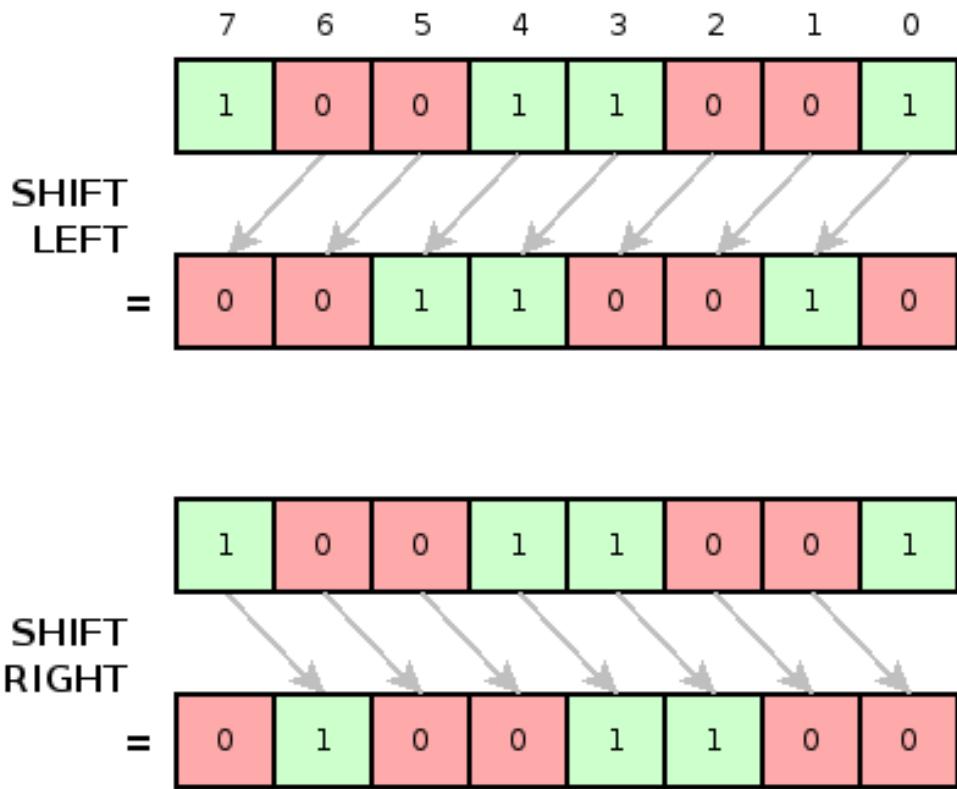
01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

Bitwise Operation

Bitwise Operators

int a = 10, b = 2 for all examples below

Operator	Meaning	Example	Result
~	Bitwise unary NOT	~a	-11
&	Bitwise AND	a&b	2
	Bitwise OR	a b	10
^	Bitwise Ex-OR	a^b	8
>>	Shift right	a>>1	5
>>>	Shift right zero fill	a>>>1	5
<<	Shift left	a<<1	20
&=	Bitwise AND assignment	a &= b	2
=	Bitwise OR assignment	a = b	10
^=	Bitwise Ex-OR assignment	a ^= b	8
>>=	Shift right assignment	a >>= 1	5
>>>=	Shift right zero fill assignment	a >>>= 1	5
<<=	Shift left assignment	a <<= 1	20



Machine Language

Machine Language

- Instructions, like registers and words of data, are also 32 bits long
 - Example: add \$t0, \$s1, \$s2
 - registers have numbers, \$t0=8, \$s1=17, \$s2=18
- Instruction Format:

000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	funct

- Can you guess what the field names stand for?**

op = Basic operation of the instruction: opcode
 rs = The first register source operand
 rt = The second register source operand
 rd = The register destination operand
 shamt = shift amount
 funct = function code

25

Assembly Language



Assembly Language

- Tied to the specifics of the underlying machine
- Commands and names to make the code readable and writeable by humans
- Hand-coded assembly code may be more efficient
- E.g., IA32 from Intel

```

        movl $0, %ecx
.loop:  cmpl $1, %edx
        jle .endloop
        addl $1, %ecx
        movl %edx, %eax
        andl $1, %eax
        je .else
        movl %edx, %eax
        addl %eax, %edx
        addl %eax, %edx
        addl $1, %edx
        jmp .endif
.else:  sarl $1, %edx
.endif: jmp .loop
.endloop:
    
```

Reading IA32 Assembly Language



- Assembler directives: starting with a period (“.”)
 - E.g., “.section .text” to start the text section of memory
 - E.g., “.loop” for the address of an instruction
- Referring to a register: percent size (“%”)
 - E.g., “%ecx” or “%eip”
- Referring to a constant: dollar sign (“\$”)
 - E.g., “\$1” for the number 1
- Storing result: typically in the second argument
 - E.g. “addl \$1, %ecx” increments register ECX
 - E.g., “movl %edx, %eax” moves EDX to EAX
- Comment: pound sign (“#”)
 - E.g., “# Purpose: Convert lower to upper case”

28



High level language



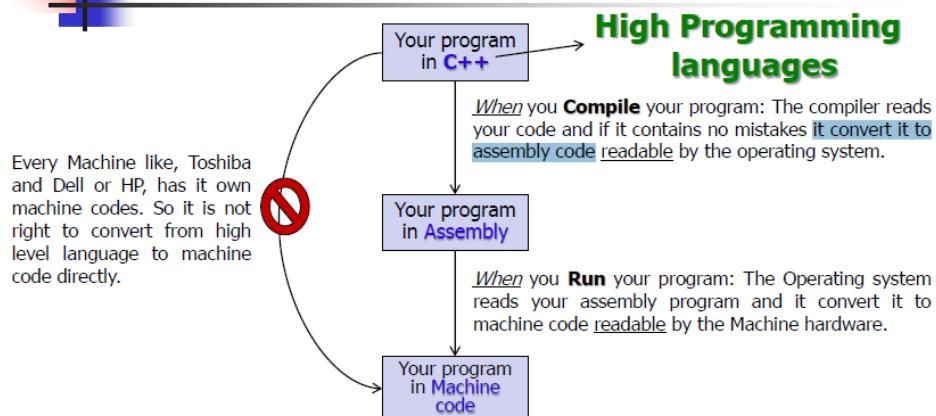
- As stated earlier, a program written in any programming language is a set of logically related instructions. These instructions have two parts, as shown in the following figure:
- The two parts of a programming language instruction are:
 - Operation code (opcode): This part instructs a computer about the operation to be performed.
 - Operand: This part instructs the computer about the location of the data on which the operation specified by the opcode is to be performed.

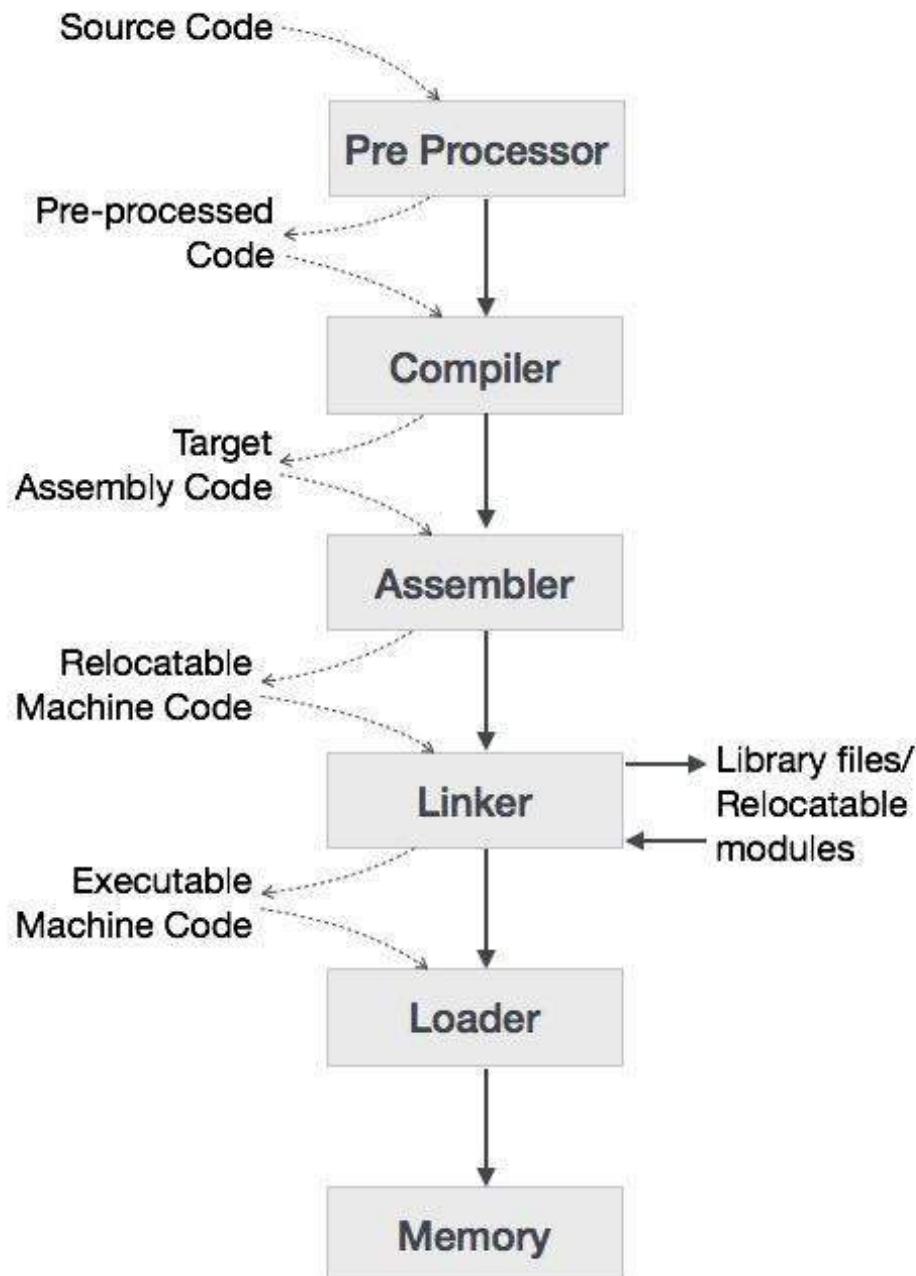


- For example, in the instruction Add A and B, Add is the opcode and A and B are operands

Compiler and Assembler

Compiling and running your program



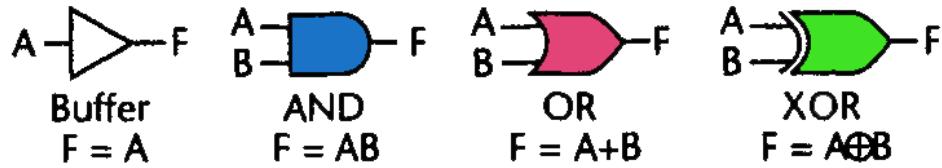


COMPILER VS LINKER VS LOADER

COMPILER	LINKER	LOADER
A software that transforms computer code written in one programming language into another programming language	A computer utility program that takes one or more object files generated by a compiler and combines them into a single executable file	A part of an operating system that is responsible for loading programs to memory
Transforms the source code into object code	Combines multiple object code and links them with libraries	Prepares the executable file for running

Logic Design

Logic Gates

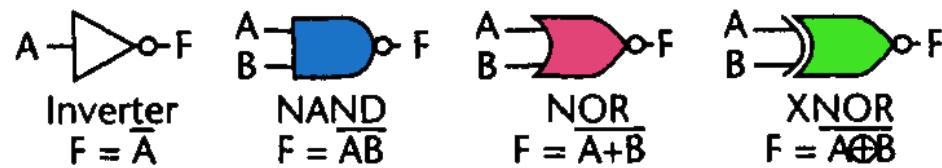


A	F
0	0
1	1

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

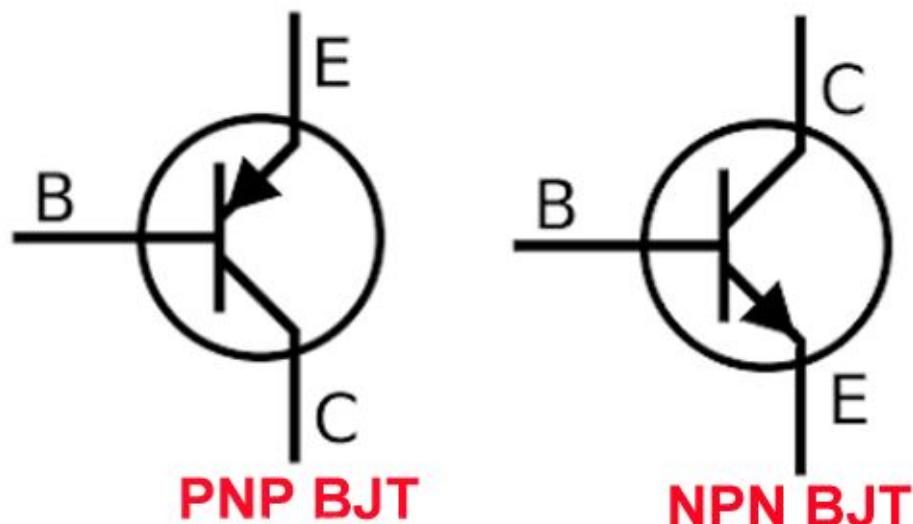


A	F
0	1
1	0

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

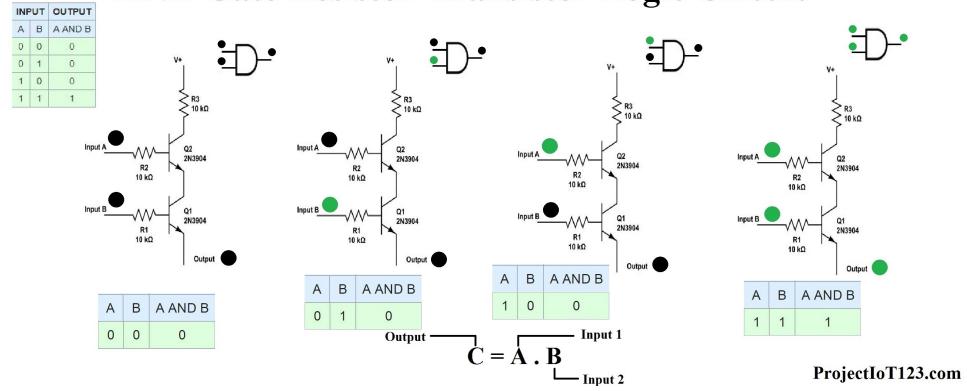
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

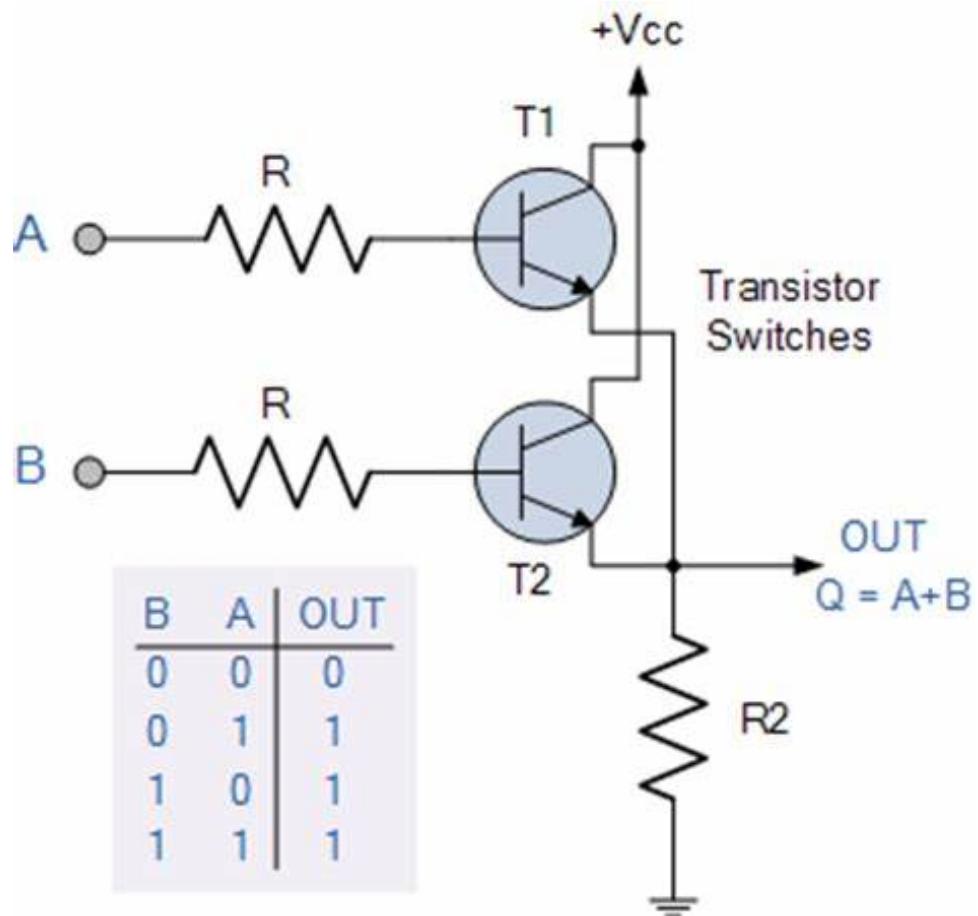


AND Gate

AND Gate Resistor-Transistor Logic Circuit

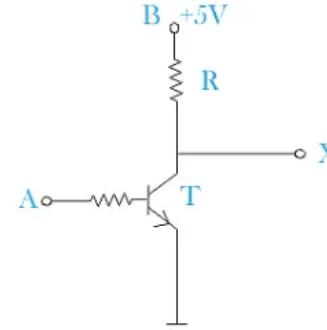
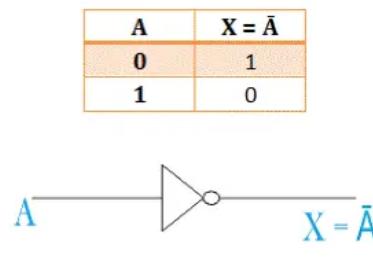


OR Gate



NOT Gate

What is a NOT Gate?



Electrical 4 U

XOR Gate

XOR GATE Truth Table



BOOLEAN EXPRESSION

$$A \cdot \bar{B} + \bar{A} \cdot B$$

$$(A + B) \cdot (\bar{A} + \bar{B})$$

$C = A \oplus B$

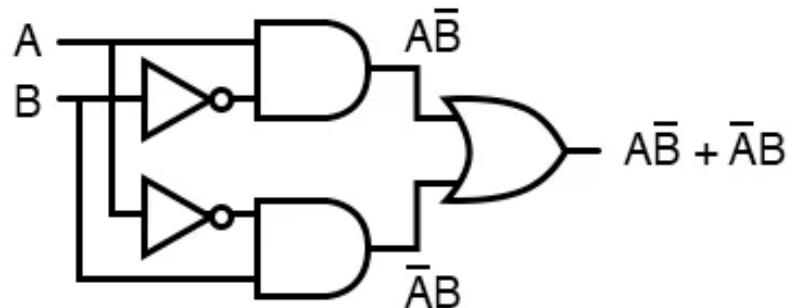
Input1
Input2
Output

INPUT		OUTPUT
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

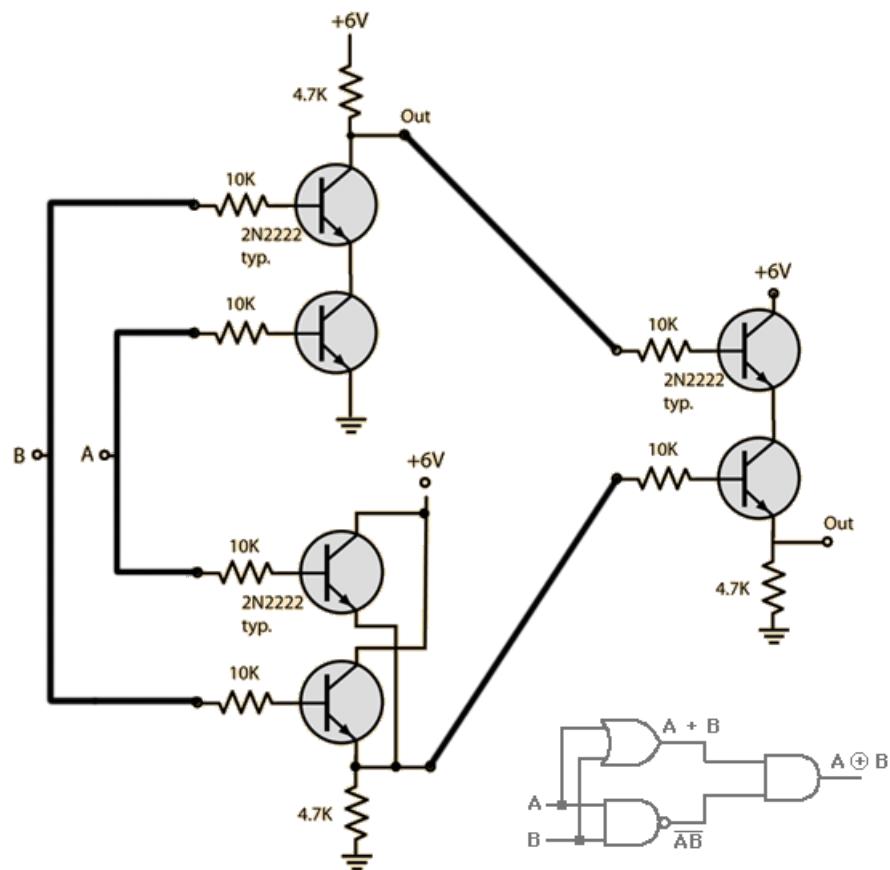
ProjectIoT123.com



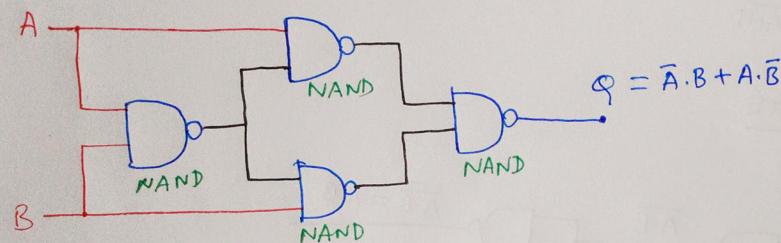
... is equivalent to ...



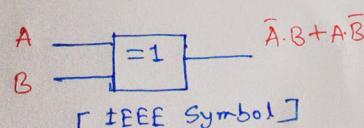
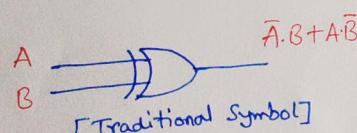
$$A \oplus B = A\bar{B} + \bar{A}B$$



XOR GATE



Which Is Equivalent To

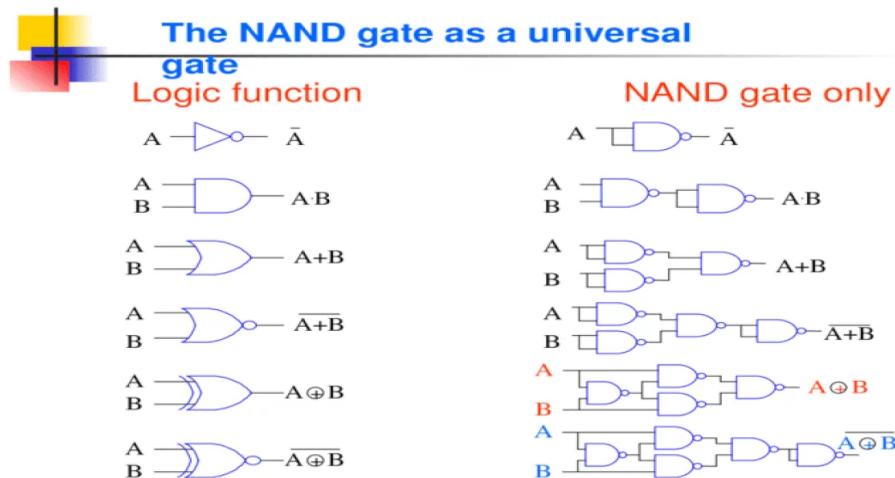
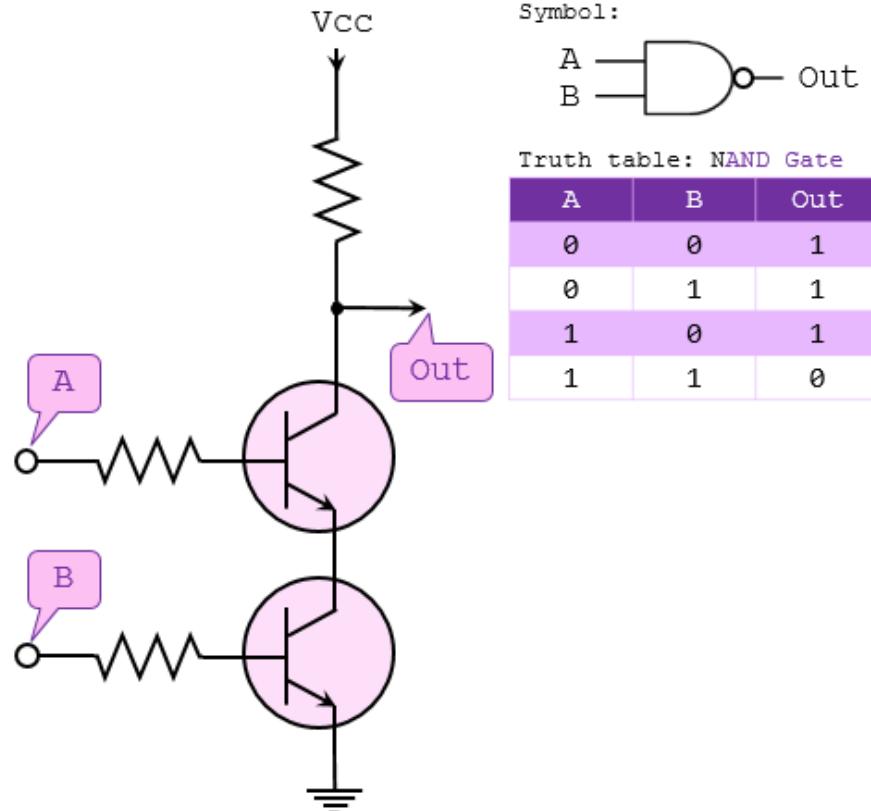


Truth Table For XOR

IN		OUT
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

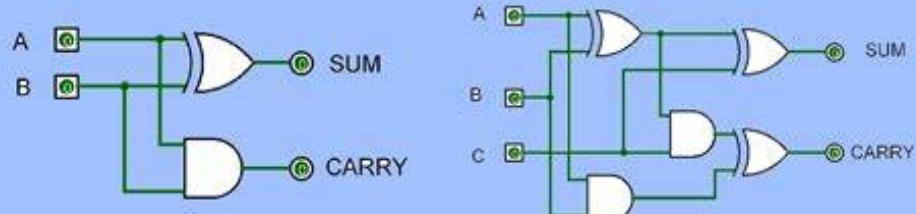
NAND Gate

NAND Gate

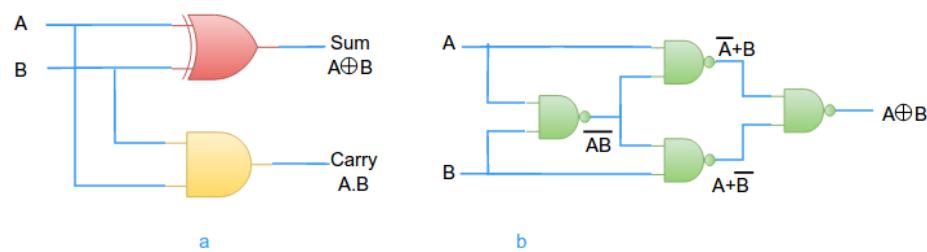
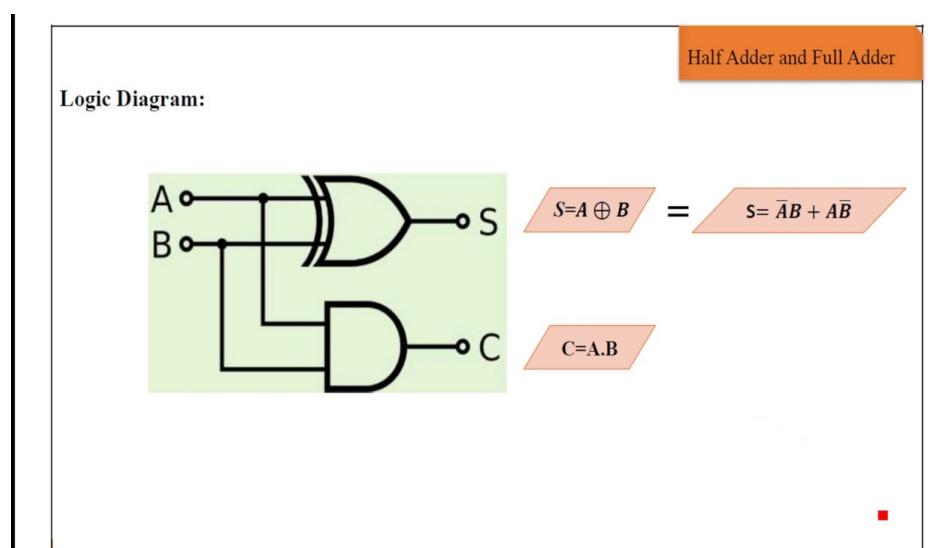


Half-Adder vs Full-Adder

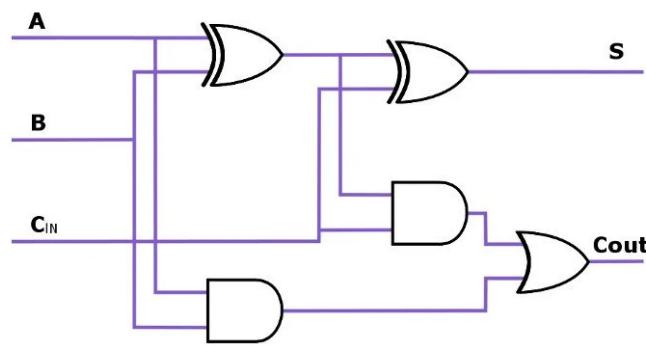
Difference Between Half Adder And Full Adder



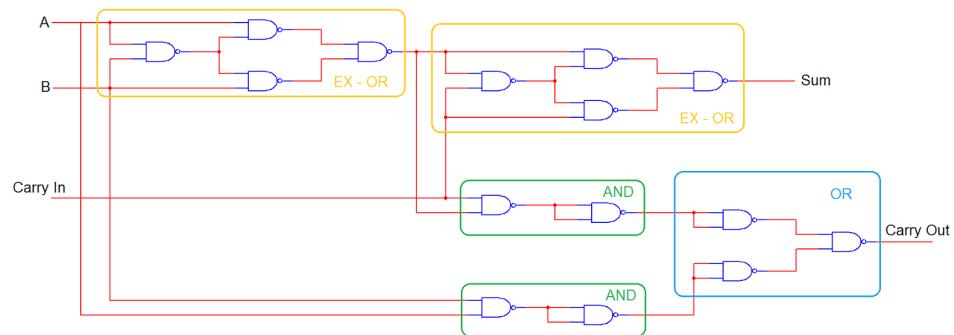
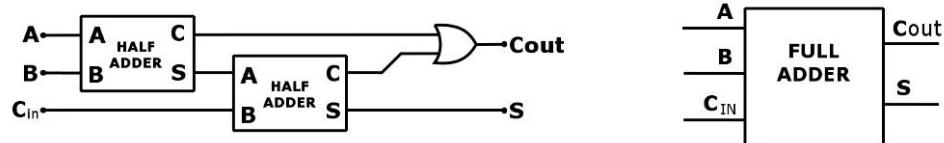
Half-Adder



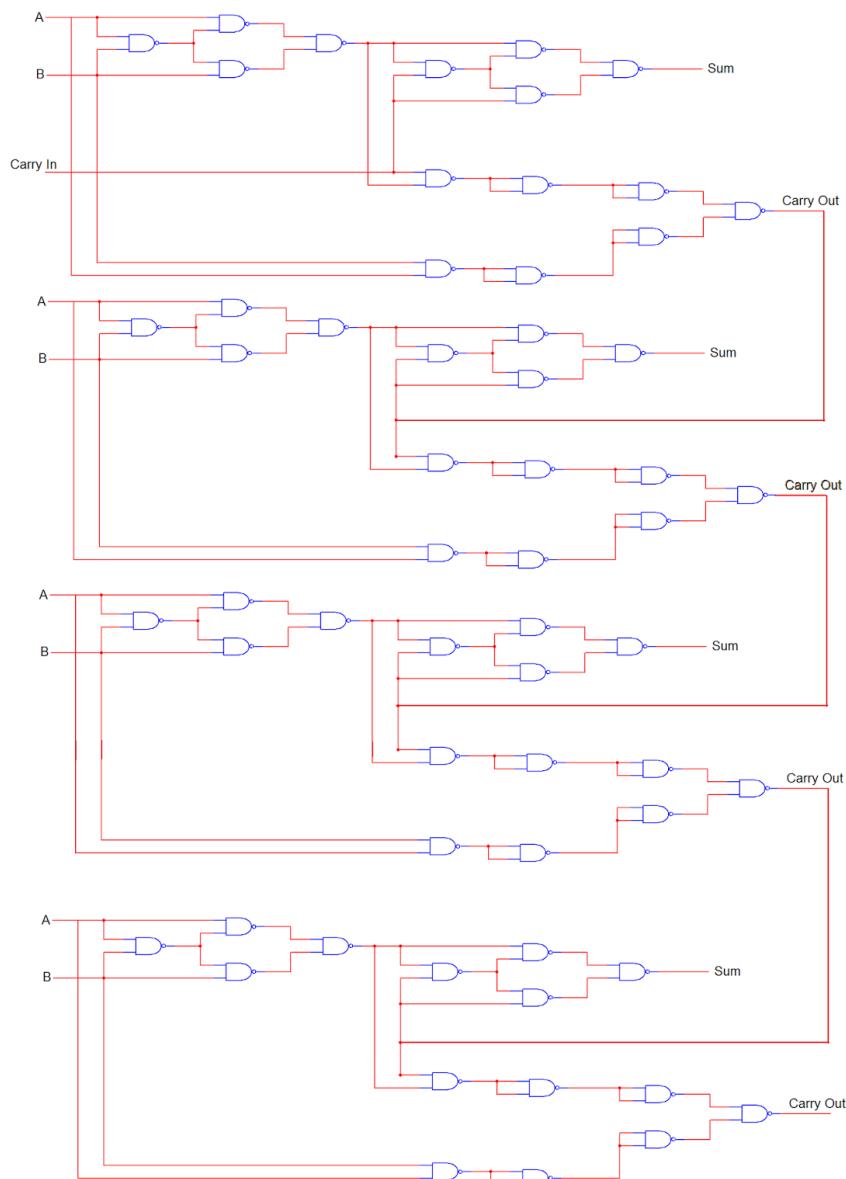
Full Adder



Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



4 Bit Full Adder



Subtract

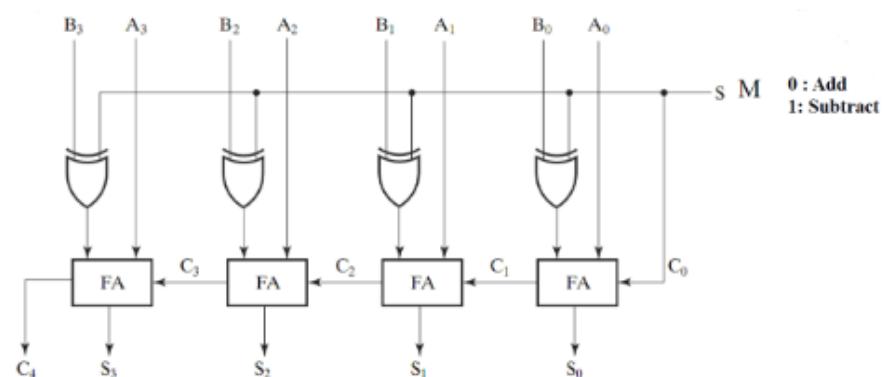
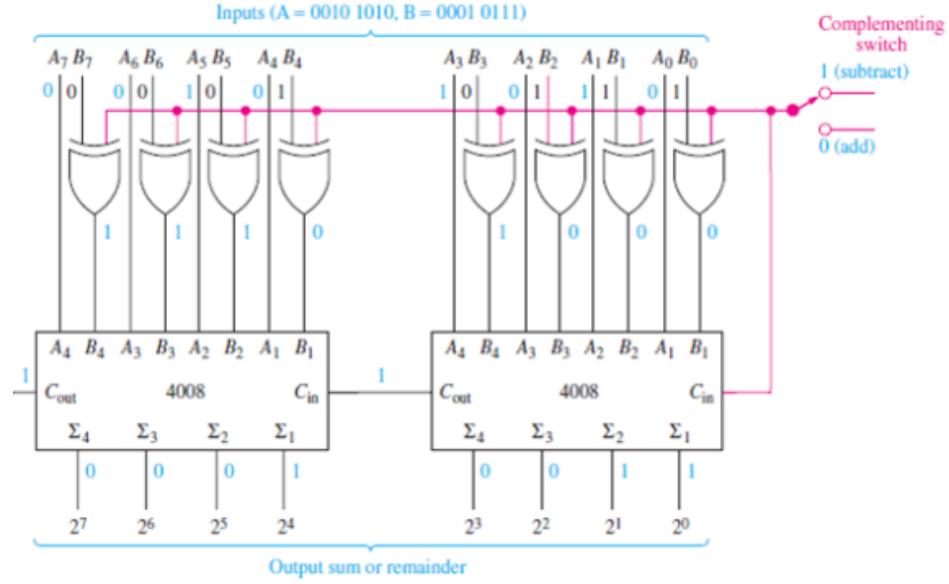


Fig. 1 Modular design of 4-bit adder/subtractor

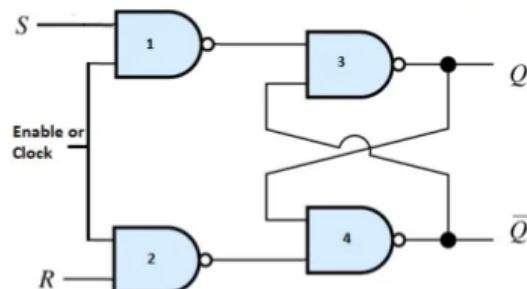


Circuits as Memory

SR Flip Flop

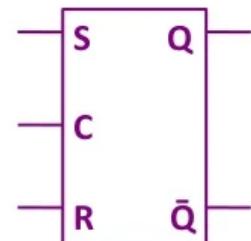


Gated Latch-Clocked RS Flip-flop



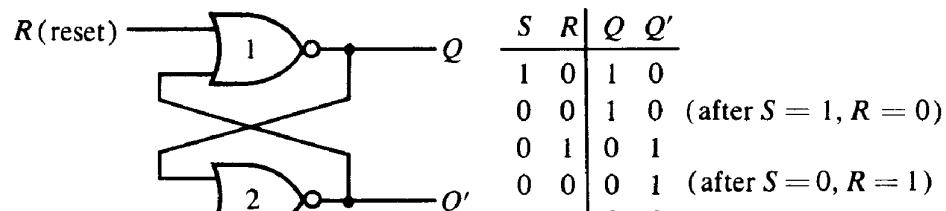
A clocked SR flip-flop.

Logical Symbol



R	S	Enable	Q_n
0	0	x	Q_{n-1}
0	1	1	1
1	0	1	0
1	1	1	Not allowed
x	x	0	Q_{n-1}

Truth table



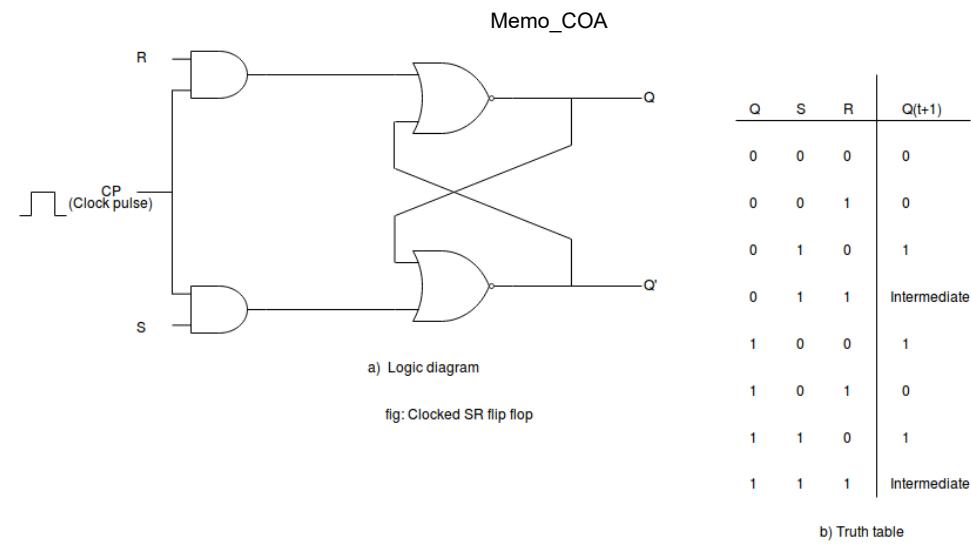
(a) Logic diagram

S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

(after $S = 1, R = 0$)

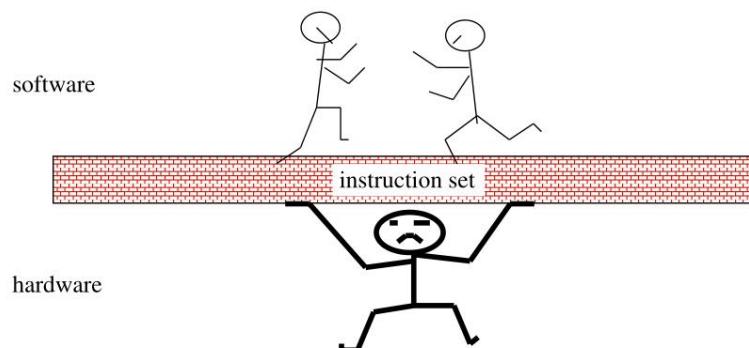
(after $S = 0, R = 1$)

(b) Truth table



Instruction Set Architecture - Interface of S/W and H/w

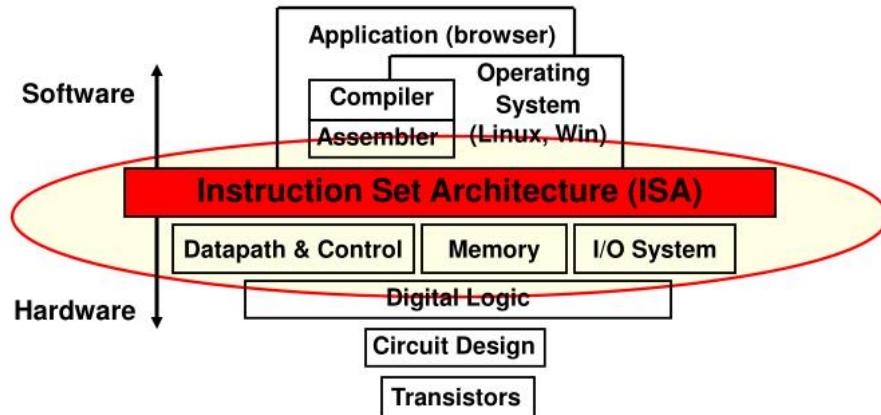
Instruction Set Architecture: Critical Interface



- Properties of a good abstraction
 - ◆ Lasts through many generations (portability)
 - ◆ Used in many different ways (generality)
 - ◆ Provides **convenient** functionality to higher levels
 - ◆ Permits an **efficient** implementation at lower levels

Computer Architecture- 8

The Instruction Set Architecture



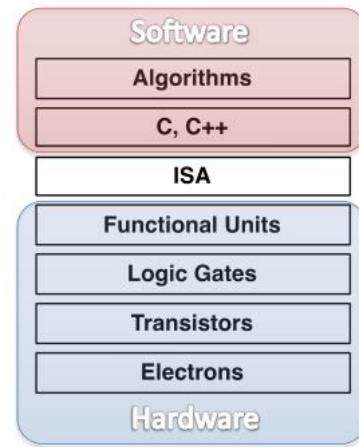
ISA

Instruction Set Architecture

- The computer ISA defines all the *programmer-visible* components and operations of the computer
 - Memory organization
 - address space -- how many locations can be addressed?
 - addressability -- how many bits per location?
 - Register set
 - how many? what size? how are they used?
 - Instruction set
 - opcodes
 - data types
 - addressing modes
- ISA provides all information needed for someone that wants to write a program in **machine language** (or translate from a high-level language to machine language).

Instruction Set Architecture (ISA)

- ISA is the interface provided by the hardware to the software
 - Defines the available:
 - instructions
 - registers
 - addressing modes
 - memory architecture
 - interrupt and exception handling
 - external I/O
 - Syntax defined by assembly language
 - Symbolic representation of the machine instructions
 - Examples: x86, ARM, HCS12



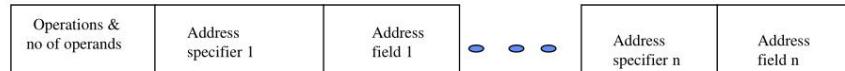
Mike Holenderski, m.holenderski@tue.nl

12



TU/e Technische Universität
Eindhoven University of Technology

Three Examples of Instruction Set Encoding



Variable: VAX (1-53 bytes)

Operation	Address field 1	Address field 2	Address field3
-----------	-----------------	-----------------	----------------

Fixed: DLX, MIPS, PowerPC, SPARC

Operation	Address Specifier	Address field
-----------	-------------------	---------------

Operation	Address Specifier 1	Address Specifier 2	Address field
-----------	---------------------	---------------------	---------------

Operation	Address Specifier	Address field 1	Address field 2
-----------	-------------------	-----------------	-----------------

Hybrid : IBM 360/370, Intel 80x86

EECC551 - Shaaban

#20 Lec # 2 Fall 2000 9-12-2000

CICS vs RICS

CISC vs. RISC

Complex Instruction Set Computer

- Many instructions
 - e.g., 75-100
- Many instructions are macro-like
 - Simplifies programming
- Most microcontrollers are based on CISC concept
 - e.g., PDP-11, VAX, Motorola 68k
 - PIC is an exception

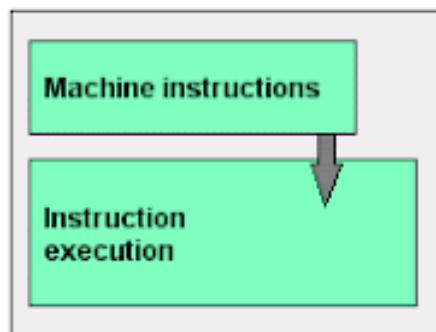
Reduced Instruction Set Computers

- Few instructions
 - e.g., 30-40
- Smaller chip, smaller pin count, & very low-power consumption
 - Simple but fast instructions
- Harvard architecture, instruction pipelining
- Industry trend for microprocessor design
 - e.g., Intel Pentium, PIC

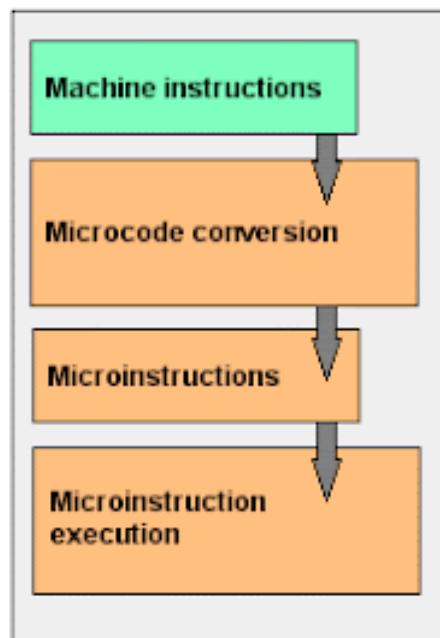
24

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co., Inc.

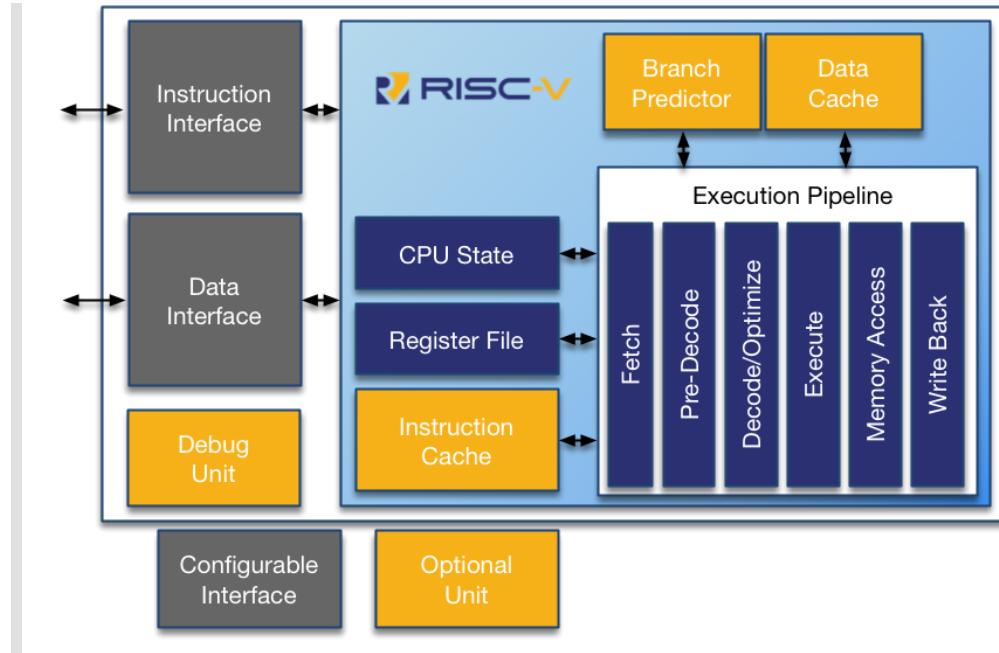
RISC



CISC

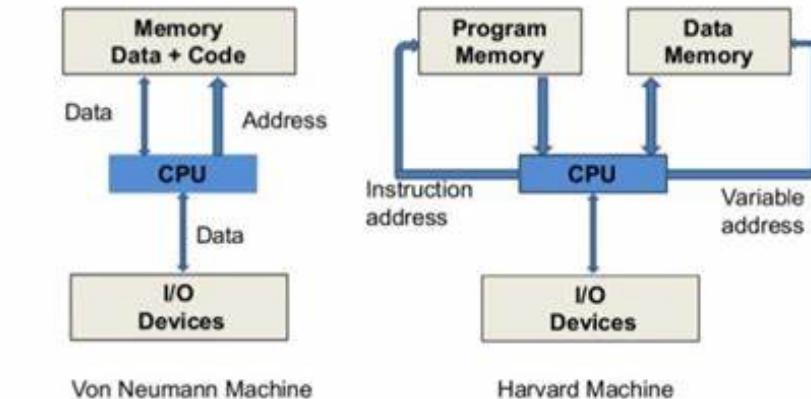


Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0										
Register/register	funct7					rs2					rs1					funct3			rd			opcode																				
Immediate	imm[11:0]					rs1					funct3			rd			opcode																									
Upper Immediate	imm[31:12]										rd			opcode																												
Store	imm[11:5]					rs2			rs1			funct3			imm[4:0]					opcode																						
Branch	[12]	imm[10:5]					rs2			rs1			funct3			imm[4:1]			[11]			opcode																				
Jump	[20]	imm[10:1]					[11]	imm[19:12]					rd			opcode																										
• opcode (7 bit): partially specifies which of the 6 types of <i>instruction formats</i> • funct7 + funct3 (10 bit): combined with <i>opcode</i> , these two fields describe what operation to perform • rs1 (5 bit): specifies register containing first operand • rs2 (5 bit): specifies second register operand • rd (5 bit): Destination register specifies register which will receive result of computation																																										



Memory Architecture

Von Neumann vs. Harvard Architecture

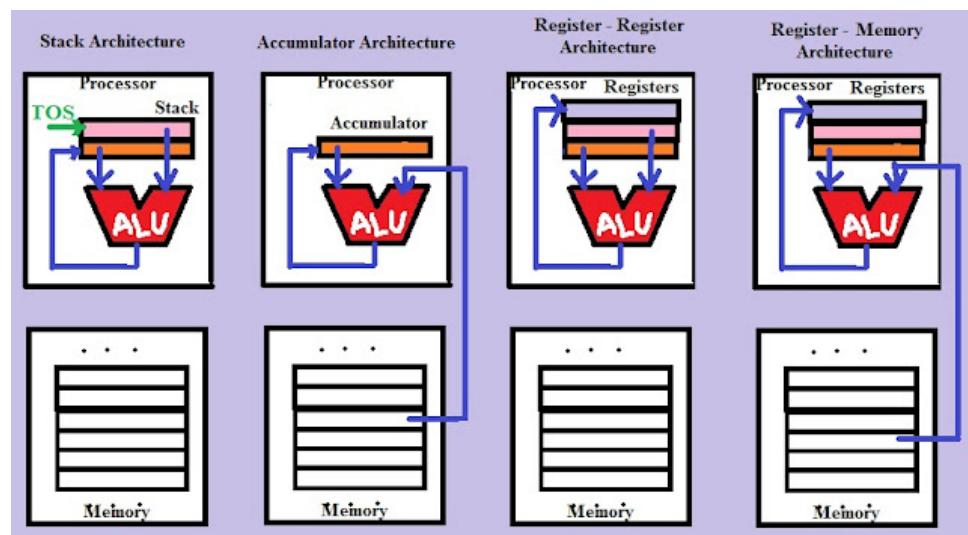
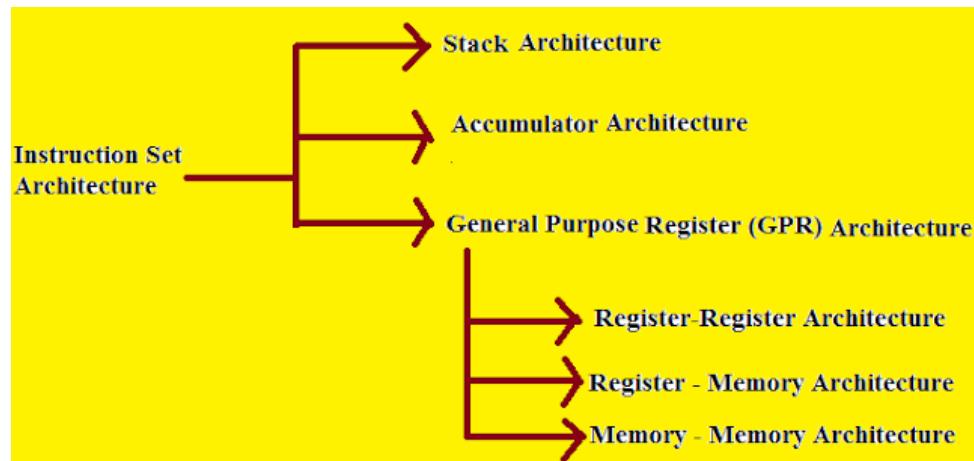


Memory & Timing model

ISA: STORAGE RESOURCES

- "Harvard architecture": Separate instruction and data memories
- Permit use of **single clock cycle per instruction** implementation
- Due to use of "cache" in modern computer architectures, it is a fairly **realistic model**

Classification of ISA (or) Types of ISA



ISA - MIPS Instruction Format and Addressing Model

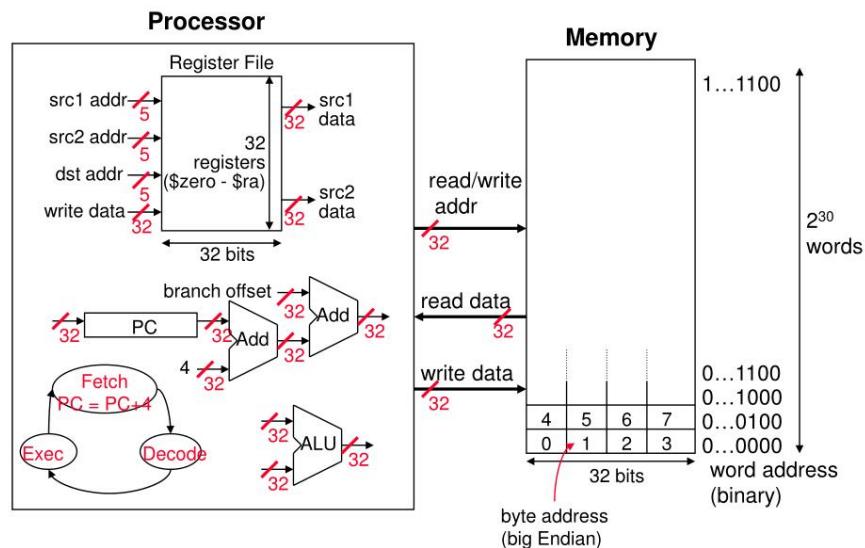
MIPS Design Principle

MIPS Design Paradigms

- Simplicity favors regularity
 - all instructions single size
 - three register operands in arithmetic instr.
 - keep register fields in the same place
- Smaller is faster
 - 32 registers
- Make good compromises
 - large addresses and constants versus unique instruction length
- Make the common case fast
 - PC-relative addressing for conditional branches

MIPS Organization

MIPS Organization So Far



34

MIPS Register File

The MIPS ISA – Register File

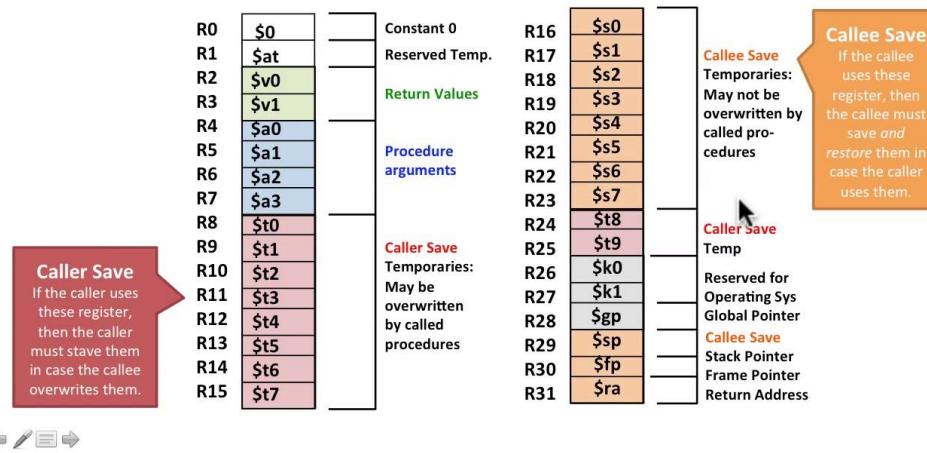
- When writing assembly, these registers can be referenced by their address (number) or name
- General purpose and special purpose registers

#	Name	Purpose	#	Name	Purpose
\$0	\$zero	Constant zero	\$16	\$s0	Temporary – Callee-saved
\$1	\$at	Reserved for assembler	\$17	\$s1	
\$2	\$v0	Function return value	\$18	\$s2	
\$3	\$v1		\$19	\$s3	
\$4	\$a0	Function parameter	\$20	\$s4	
\$5	\$a1		\$21	\$s5	
\$6	\$a2		\$22	\$s6	
\$7	\$a3		\$23	\$s7	
\$8	\$t0	Temporary – Caller-saved	\$24	\$t8	Temporary – Caller-saved
\$9	\$t1		\$25	\$t9	
\$10	\$t2		\$26	\$k0	Reserved for OS
\$11	\$t3		\$27	\$k1	
\$12	\$t4		\$28	\$gp	Global pointer
\$13	\$t5		\$29	\$sp	Stack pointer
\$14	\$t6		\$30	\$fp	Frame pointer
\$15	\$t7		\$31	\$ra	Function return address

17

53

Who saves what?



Instruction Format

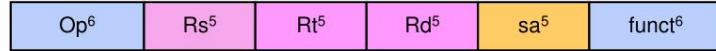
Instruction Formats

- ❖ All instructions are 32-bit wide. Three instruction formats:

- ❖ Register (R-Type)

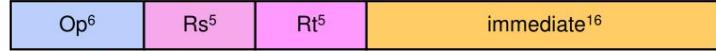
- ❖ Register-to-register instructions

- ❖ Op: operation code specifies the format of the instruction



- ❖ Immediate (I-Type)

- ❖ 16-bit immediate constant is part in the instruction



- ❖ Jump (J-Type)

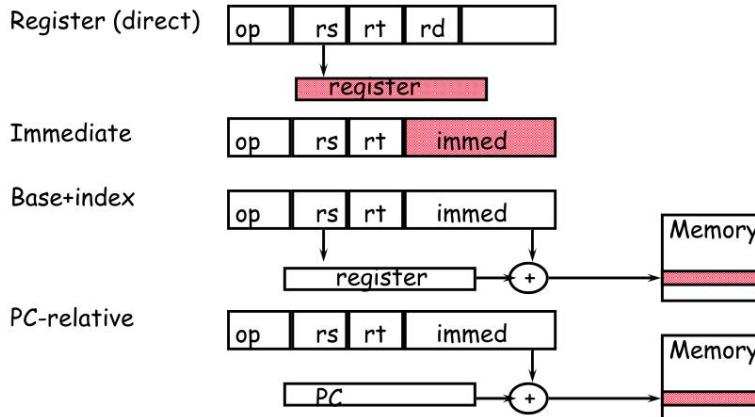
- ❖ Used by jump instructions



Addressing Model

Example: MIPS Instruction Formats and Addressing Modes

- All instructions 32 bits wide



5.6 Addressing Modes

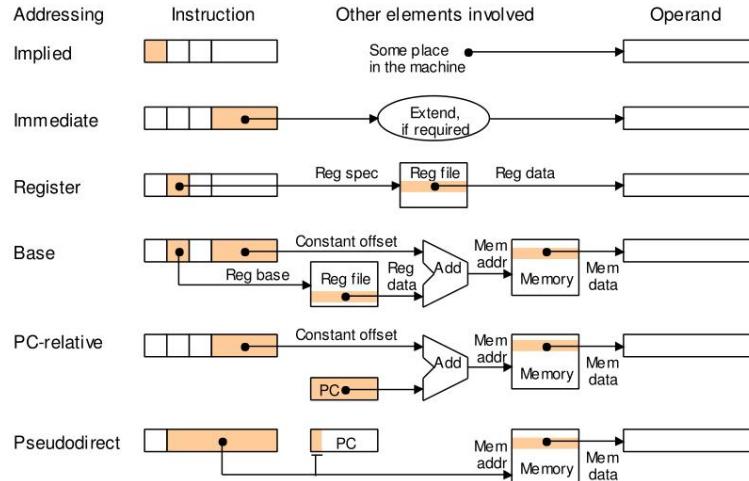


Figure 5.11 Schematic representation of addressing modes in MiniMIPS.

Computer Architecture, Instruction-Set Architecture

Slide 22

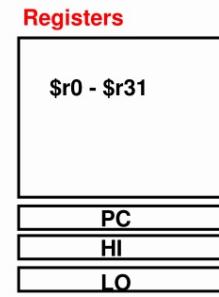
Instruction Categories

MIPS: ISA

Category	Instruction	Op Code	Example	Meaning
Arithmetic	Add	0 and 32	add \$s1, \$s2, \$s3	$\$s1 = \$s2 + \$s3$
(R & I format)	Subtract	0 and 34	sub \$s1, \$s2, \$s3	$\$s1 = \$s2 - \$s3$
	add immediate	8	addi \$s1, \$s2, 6	$\$s1 = \$s2 + 6$
	or immediate	13	ori \$s1, \$s2, 6	$\$s1 = \$s2 \vee 6$
Logical (R & I format)	And	0 and 36	and \$s1, \$s2, \$s3	$\$s1 = \$s2 \wedge \$s3$
	Or	0 and 37	or \$s1, \$s2, \$s3	$\$s1 = \$s2 \mid \$s3$
	Nor	0 and 39	nor \$s1, \$s2, \$s3	$\$s1 = \neg(\$s2 \mid \$s3)$
	And immediate	12	andi \$s1, \$s2, 100	$\$s1 = \$s2 \wedge 100$
	Or immediate	13	ori \$s1, \$s2, 100	$\$s1 = \$s2 \mid 100$
	Shift left logical	0 and 0	sll \$s1, \$s2, 10	$\$s1 = \$s2 \ll 10$
	Shift right logical	0 and 2	srl \$s1, \$s2, 10	$\$s1 = \$s2 \gg 10$
Data Transfer (I format)	load word	35	lw \$s1, 24(\$s2)	$\$s1 = \text{Memory}[\$s2+24]$
	store word	43	sw \$s1, 24(\$s2)	$\text{Memory}[\$s2+24] = \$s1$
	load byte	32	lb \$s1, 25(\$s2)	$\$s1 = \text{Memory}[\$s2+25]$
	store byte	40	sb \$s1, 25(\$s2)	$\text{Memory}[\$s2+25] = \$s1$
	load upper imm	15	lui \$s1, 6	$\$s1 = 6 \cdot 2^{16}$
Cond. Branch (I & R format)	br on equal	4	beq \$s1, \$s2, L	if ($\$s1 == \$s2$) go to L
	br on not equal	5	bne \$s1, \$s2, L	if ($\$s1 != \$s2$) go to L
	set on less than	0 and 42	slt \$s1, \$s2, \$s3	if ($\$s2 < \$s3$) \$s1=1 else \$s1=0
	set on less than immediate	10	slti \$s1, \$s2, 6	if ($\$s2 < 6$) \$s1=1 else \$s1=0
Uncond. Jump (J & R format)	jump	2	j 2500	go to 10000
	jump register	0 and 8	jr \$t1	go to \$t1
	jump and link	3	jal 2500	go to 10000; \$ra=PC+4

MIPS ISA as an Example

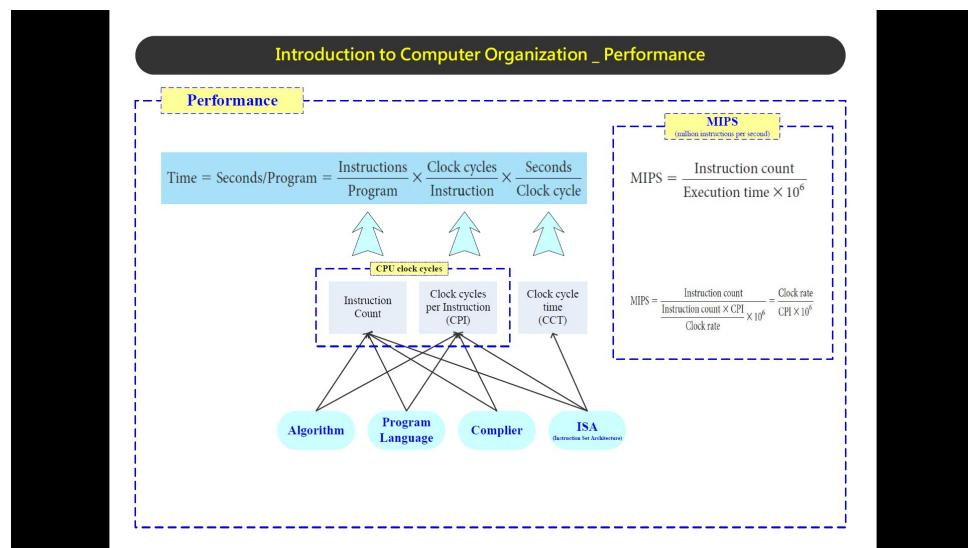
- ▲ Instruction categories:
 - Load/Store
 - Computational
 - Jump and Branch
 - Floating Point
 - Memory Management
 - Special



3 Instruction Formats: all 32 bits wide

OP	\$rs	\$rt	\$rd	sa	funct		
OP	\$rs	\$rt		immediate			
OP			jump target				

ISA - Performance



Performance Summary

- Measurements at different level
- Combined to yield execution time in seconds per program

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
 - Algorithm
 - Programming language
 - Compiler
 - Instruction set architecture

Chapter 1 — Computer Abstractions and Technology — 1

CPU Execution Time: The CPU Equation

- A program is comprised of a number of instructions, I
 - Measured in: instructions/program
- The average instruction takes a number of cycles per instruction (CPI) to be completed.
 - Measured in: cycles/instruction, CPI
- CPU has a fixed clock cycle time C = 1/clock rate
 - Measured in: seconds/cycle
- CPU execution time is the product of the above three parameters as follows:

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$T = I \times CPI \times C$$

EECC550 - Shaaban

#4 Lec # 3 Spring 2003 3-17-2003

Speed Up Equation for Pipelining

$$CPI_{\text{pipelined}} = \text{Ideal CPI} + \text{Average Stall cycles per Inst}$$

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

For simple RISC pipeline, CPI = 1:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

CS211 41

Example: Speed-up Challenge

課程: 計算機組織 / 老師: 朱宗賢

- Suppose you want to achieve a speed-up 90 times faster with 100 processors. What percentage of the original computation can be sequential?

Amdahl's law concept (1967): The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.

- $T_{\text{new}} = T_{\text{parallelizable}}/100 + T_{\text{sequential}}$
- Speedup = $\frac{1}{(1-F_{\text{parallelizable}})+F_{\text{parallelizable}}/100} = 90$
- Solving: $F_{\text{parallelizable}} = 0.999$
- Need sequential part to be 0.1% of original time

1. (2 marks) A particular program has an execution time of 12 seconds on a particular system that has a clock rate of 2.5 GHz. 8 billion (i.e., 8×10^9) machine language instructions are executed. What is the CPI?

2. (10 marks) Suppose that the following clock cycles per instruction, and frequencies of usage by a particular program, have been determined for the four instruction types of a particular system with a clock cycle time of 0.5 nanoseconds.

Instruction type	Clock cycles per instruction	Frequency
A	3	10%
B	1	50%
C	2	20%
D	4	20%

- (a) What MIPS rating is achieved for this program on this system?

- (b) Suppose that the CPU execution time is 0.5 seconds. What must be the number of instructions executed?

- (c) Suppose that through use of a compiler optimization, it is possible to **halve** the number of executed type B instructions, with the numbers of all other types of instructions remaining the same.
 - (i) Would the MIPS rating increase, or would it decrease, when using this optimization?
 - (ii) By what percentage would the MIPS rating change?
 - (iii) By what percentage would the CPU execution time decrease?

3. Calculate the CPU time to execute a program containing one million machine instructions, when the average CPI of the CPU is 2.5 and CPU clock frequency is 2GHz. [15pts]

4. Let us overclock the machine given in problem 3 by increasing the CPU clock frequency from 2GHz to 2.5GHz. Calculate the CPU time of the overclocked machine running the same program containing one million instructions. How much faster (i.e., performance ratio) is the overclocked machine than the original machine? [15pts]

(8 marks) Suppose that the following clock cycles per instruction, and frequencies of usage by a particular program, have been determined for the four instruction types of a system with a clock rate of 2 GHz.

Instruction type	Clock cycles per instruction	Frequency
A	3	20%
B	1	40%
C	2	30%
D	4	10%

- (a) What MIPS rating is achieved for this program on this system?

- (b) Suppose that 500×10^6 instructions are executed in total. What is the CPU execution time?

- (c) Suppose it was possible to replace particular pairs of type C instructions with single type A instructions. Would the MIPS rating increase, or decrease, with this change?

- (d) If it were possible to replace 2/3 of the type C instructions in the manner described in part (c), by what percentage would CPU execution time decrease?

Complete the following questions on CPU performance and show all of your work. For longer decimal results, round to three decimal places.

1. Suppose you wish to run a program P with 5×10^9 instructions on a 3 GHz machine with a CPI of 2.3. What is the expected CPU time?

2. Suppose you have the following instruction mix. What is the CPI?

Instruction Type	Frequency	Cycles
Memory	22	2
ALU	40	1
Branch	3	2
Jump	8	1

3. Consider two different processors P1 and P2 executing the same instruction set with the clock rates and CPIs given in the following table.

1. If the processors each execute a program in 15 seconds, find the number of cycles and the number of instructions for each processor.

2. We are trying to reduce the program execution time by 20%, but this leads to an increase of 25% in the CPI. What clock rate should we have to get this time reduction for each processor?

Processor	Clock Rate	CPI
P1	4 GHz	2
P2	4.5 GHz	1.5

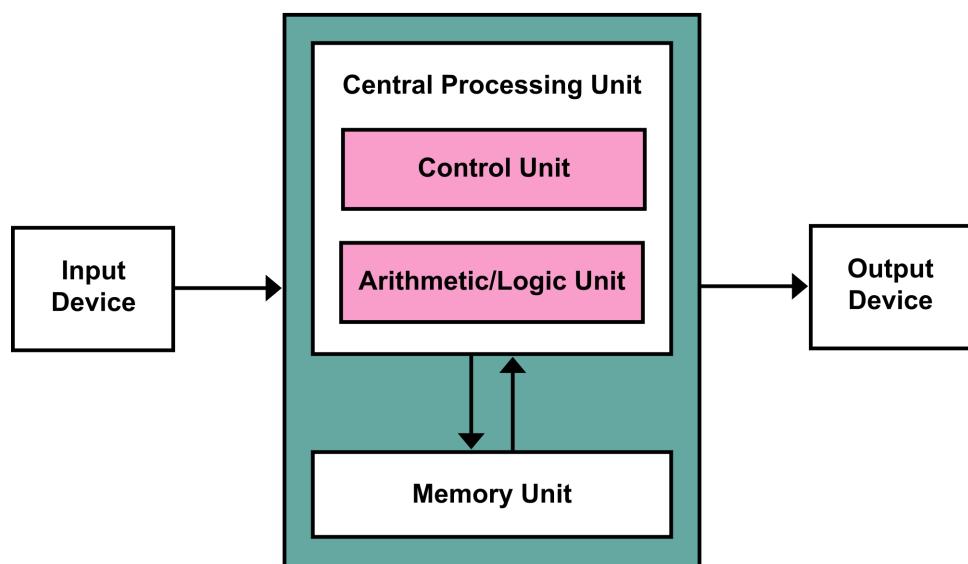
9. (10%) CPU execution time and CPI are computed as follows when considering memory stalls.
 $\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$

$$\text{CPU Clock Cycles} = \sum_{i=1}^n (\text{CPI}_i \times \text{Instruction Count}_i)$$

$$\text{Memory stall cycles} = IC \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty}$$

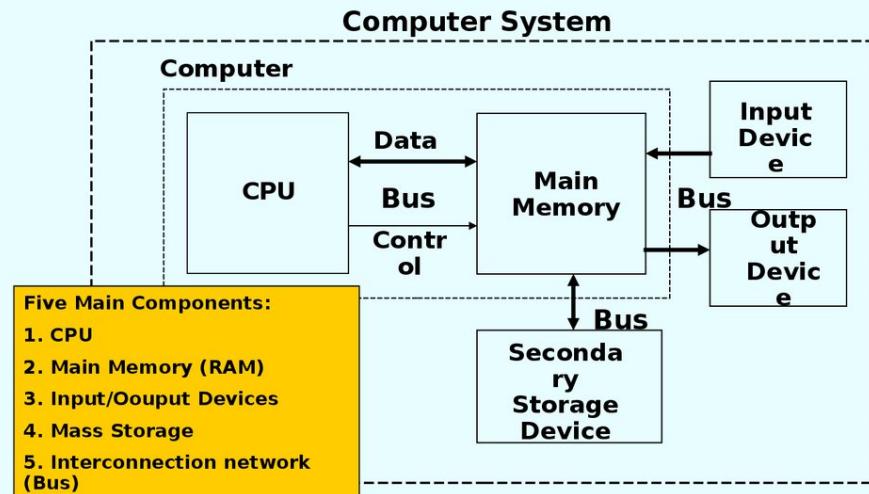
A program contains 30% arithmetic/logic instructions, 50% load/store, 20% control transfer instructions. For a processor, the ideal CPIs for each of the three classes of instructions are 1, 2, 3, respectively. If the miss rate for data cache is 0.2 and for instruction cache 0.1, both with 100 cycles miss penalty. What is the average CPI of the program?

Processor - Von Neumann Architecture

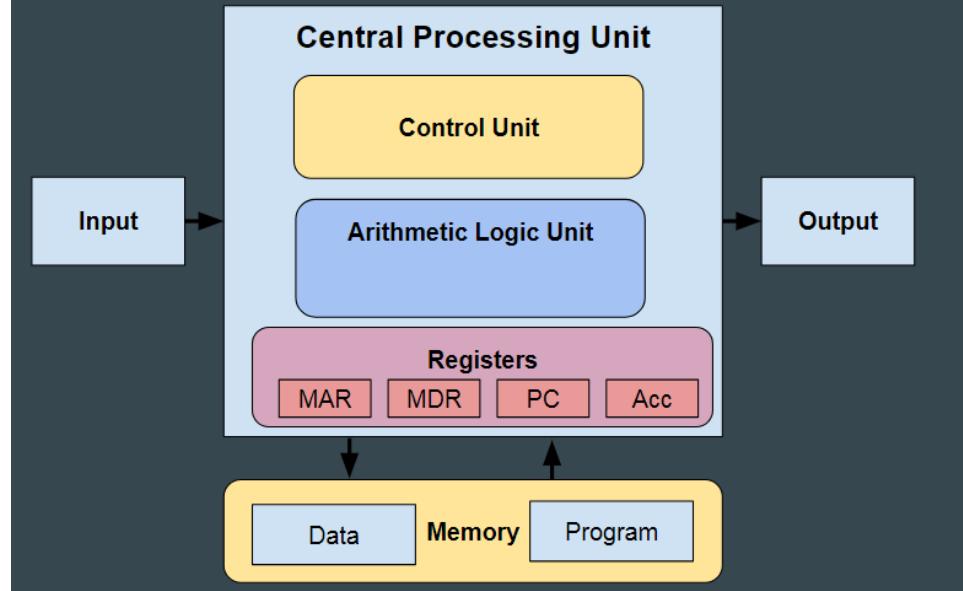


von Neumann Architecture

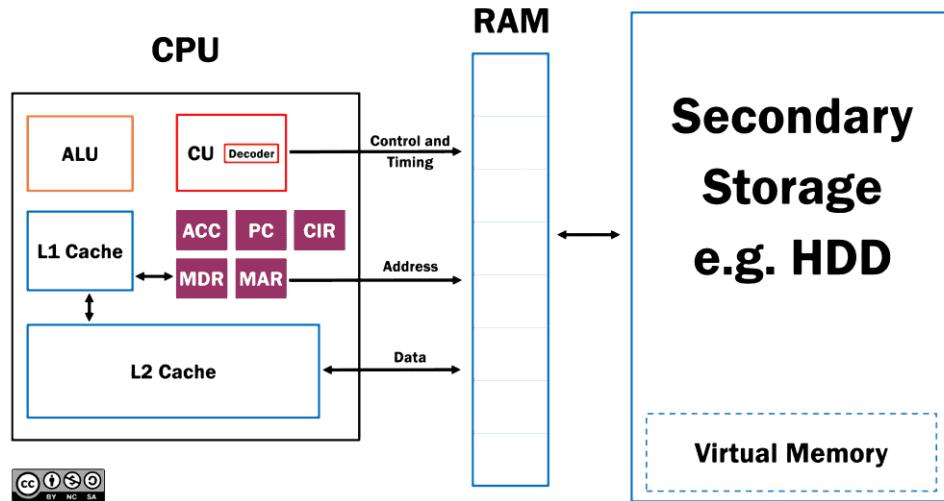
- A more complete view of the computer *system* architecture that integrates interaction (human or otherwise) consists of:



Von Neumann Architecture Diagram



Computer Systems - Von Neumann Architecture



Processor - Datapath

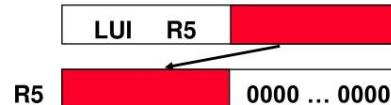
Review: MIPS data transfer instructions

- For all cases, calculate effective address first
 - MIPS doesn't use segmented memory model like x86
 - Flat memory model → EA = address being accessed
- **lb, lh, lw**
 - Get data from addressed memory location
 - Sign extend if **lb** or **lh**, load into **rt**
- **lbu, lhu, lwu**
 - Get data from addressed memory location
 - Zero extend if **lb** or **lh**, load into **rt**
- **sb, sh, sw**
 - Store data from **rt** (partial if **sb** or **sh**) into addressed location

MIPS Data Transfer Instructions

<u>Instruction</u>	<u>Comment</u>
SW R3, 500(R4)	Store word
SH R3, 502(R2)	Store half
SB R2, 41(R3)	Store byte
LW R1, 30(R2)	Load word
LH R1, 40(R3)	Load half word
LHU R1, 40(R3)	Load half word unsigned
LB R1, 40(R3)	Load byte
LBU R1, 40(R3)	Load byte unsigned
LUI R1, 40	Load Upper Immediate (16 bits shifted left by 16)

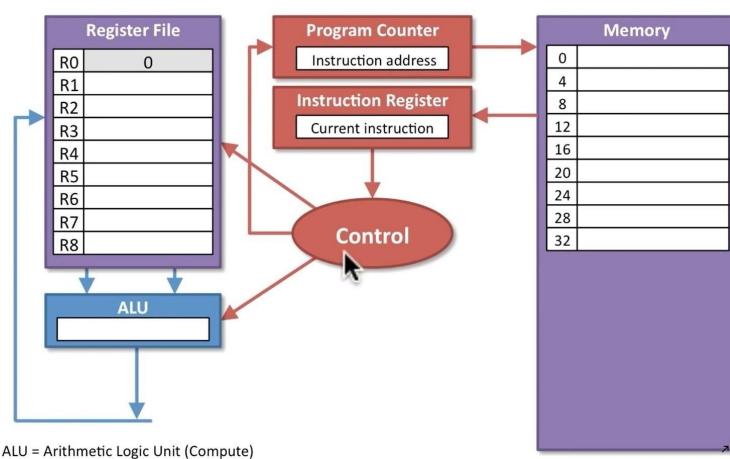
Why do we need LUI?



Data operations in detail

1. Data Operations 26

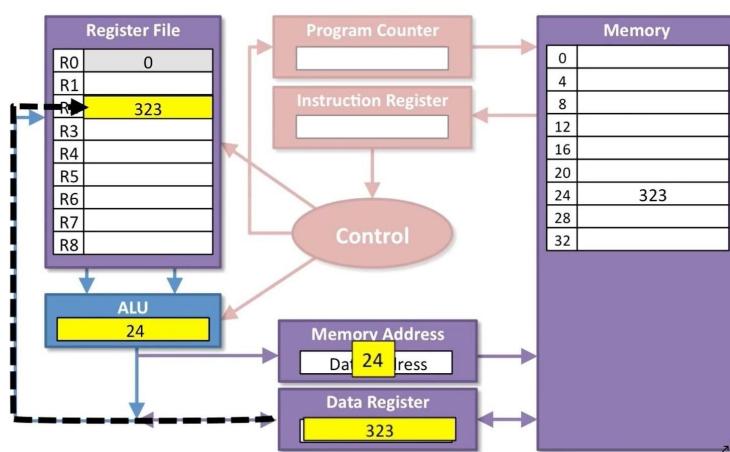
1. Program Counter holds the instruction address.
2. Instructions are *fetched* from memory into the Instruction Register.
3. Control logic *decodes* the instruction and tells the ALU and Register File what to do.
4. ALU *executes* the instruction and results flow back to the Register File.
5. The Control logic *updates* the Program Counter for the next instruction.



Data transfers in detail

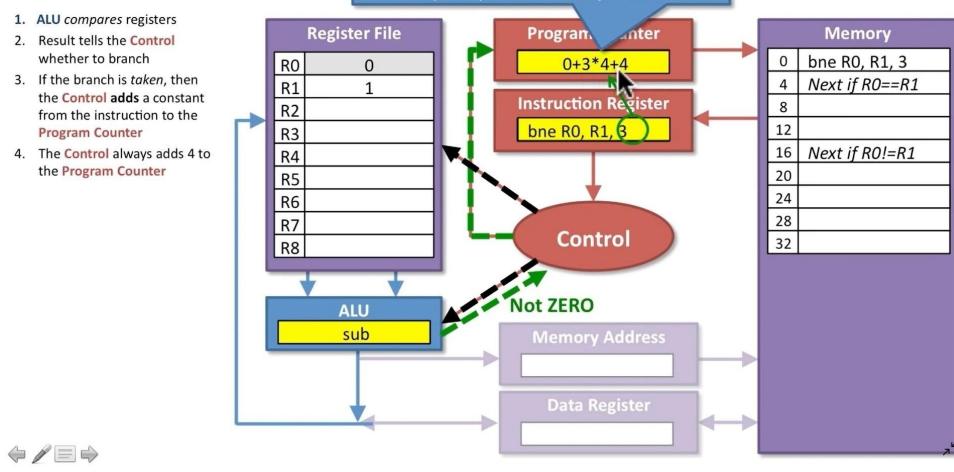
2. Data Transfers 32

1. ALU generates address
2. Address goes to the Memory Address Register
3. Results to/from memory are stored in the Memory Data Register
4. Data from memory can now be stored back into the Register File or to memory can be written.



Sequencing in detail

1. ALU compares registers
2. Result tells the Control whether to branch
3. If the branch is taken, then the Control adds a constant from the instruction to the Program Counter
4. The Control always adds 4 to the Program Counter

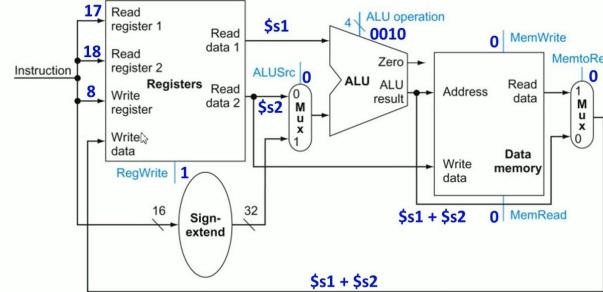


Example: Building a datapath for R – type and memory instructions

`add $t0, $s1, $s2`

R-type	0	\$s1 (17)	\$s2 (18)	\$t0(8)	Shamt (0)	Funct (32)
	31:26	25:21	20:16	15:11	10:6	5:0

Name	Number
\$t0 - \$t7	8-15
\$s0 - \$s7	16-23



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

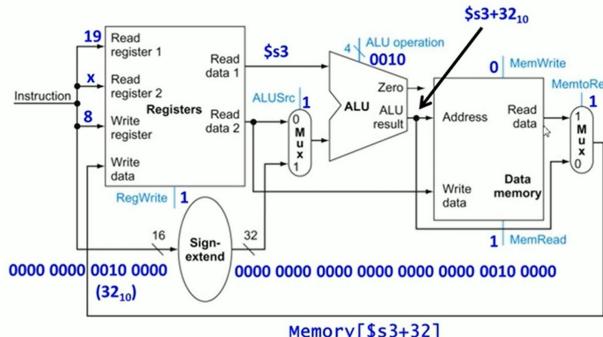
課程:計算機組織 / 老師:朱宗賢

Example: Building a datapath for R – type and memory instructions

`lw $t0, 32($s3)`

Load	35	\$s3 (19)	\$t0 (8)	Address (32)
	31:26	25:21	20:16	15:0

Name	Number
\$t0 - \$t7	8-15
\$s0 - \$s7	16-23



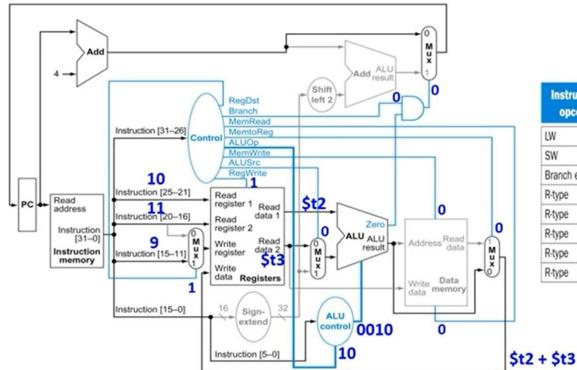
ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

課程:計算機組織 / 老師:朱宗賢

The datapath in operation for an R-type instruction

add \$t1, \$t2, \$t3 # \$t1 = \$t2 + \$t3

R-type	0	\$t2 (10)	\$t3 (11)	\$t1 (9)	Shamt (0)	Funct (32)
	31:26	25:21	20:16	15:11	10:6	5:0



課程: 計算機組織 / 老師: 朱宗賢

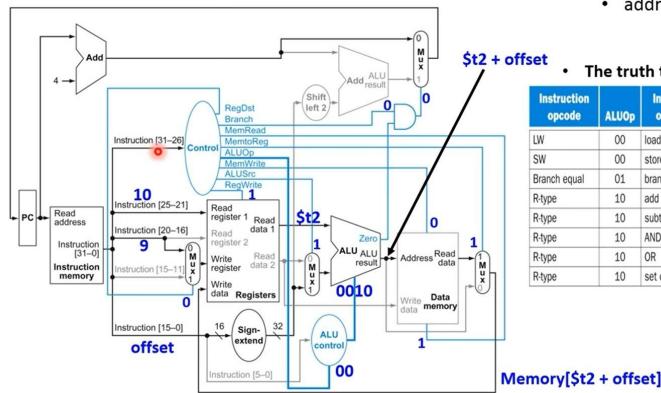
- R-type instructions are used when all the data values used by the instruction are located in registers. Ex: add, sub, or and sll
- Instruction fetch -> register read -> ALU operation -> register write

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
RType	10	add	100000	add	0010
RType	10	subtract	100010	subtract	0110
RType	10	AND	100100	AND	0000
RType	10	OR	100101	OR	0001
RType	10	set on less than	101010	set on less than	0111

The datapath in operation for a load instruction

lw \$t1, offset(\$t2) # \$t1 = Memory[\$t2 + offset]

0x23	\$t2 (10)	\$t1 (9)	offset
31:26	25:21	20:16	15:0



課程: 計算機組織 / 老師: 朱宗賢

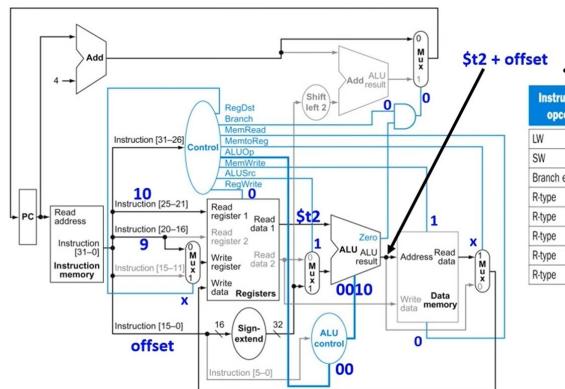
- instruction fetch -> register read -> ALU operation -> data access -> register write
- address line, control line, data line

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
RType	10	add	100000	add	0010
RType	10	subtract	100010	subtract	0110
RType	10	AND	100100	AND	0000
RType	10	OR	100101	OR	0001
RType	10	set on less than	101010	set on less than	0111

The datapath in operation for a store instruction

sw \$t1, offset(\$t2) # Memory[\$t2 + offset] = \$t1

0x2b	\$t2 (10)	\$t1 (9)	offset
31:26	25:21	20:16	15:0



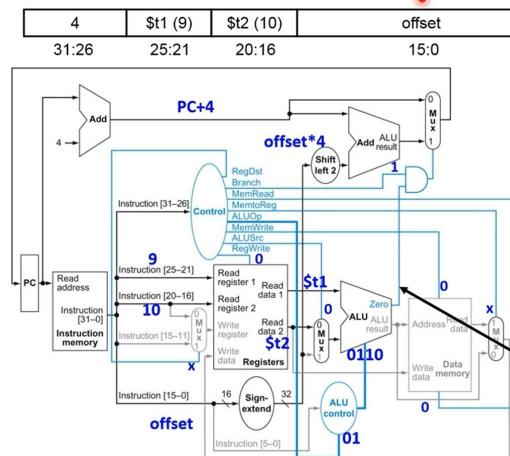
課程: 計算機組織 / 老師: 朱宗賢

- instruction fetch -> register read -> ALU operation -> data access

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
RType	10	add	100000	add	0010
RType	10	subtract	100010	subtract	0110
RType	10	AND	100100	AND	0000
RType	10	OR	100101	OR	0001
RType	10	set on less than	101010	set on less than	0111

The datapath in operation for a branch – on – equal instruction

beq \$t1, \$t2, offset # if (\$t1==\$t2) goto PC+4+offset*4



課程:計算機組織 / 老師:朱宗賢

- instruction fetch -> register read -> ALU operation -> decide which adder result to store in to the PC

The truth table for the 4 ALU control bits

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
Rtype	10	add	100000	add	0010
Rtype	10	subtract	100010	subtract	0110
Rtype	10	AND	100100	AND	0000
Rtype	10	OR	100101	OR	0001
Rtype	10	set on less than	101010	set on less than	0111

The ALU's Zero output is set if (\$t1-\$t2) is 0.

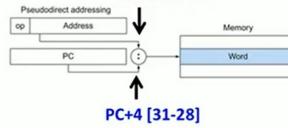
Example: Implementing Jumps

j 10000 # go to location 10000_h (word address)

J-type

2	10000
31:26	25:0

Address <<2

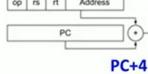


- The MIPS jump instruction only replaces the lower 28bit of PC+4

- The sources of new PC:

1. Incremented PC (PC+4)
2. The branch target PC
3. The jump target PC

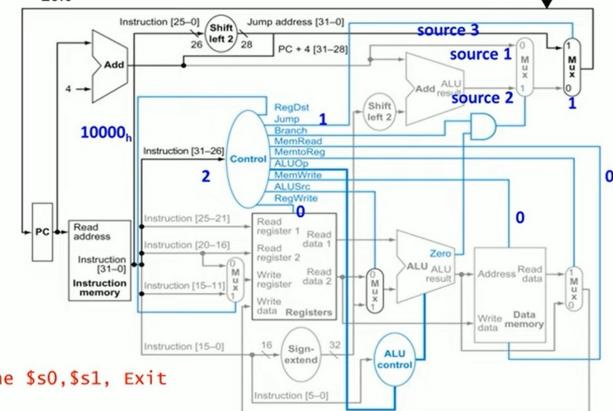
PC-relative addressing



bne \$s0,\$s1, Exit

課程:計算機組織 / 老師:朱宗賢

PC+4 [31:28] 40000
31:28 27:0



Showing Branch Offset in Machine Language

bne

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

PC-relative addressing

j

op	address
6 bits	26 bits

Pseudo-direct Addressing

Assume Loop at location 80000

Loop: sll \$t1, \$s3, 2
add \$t1, \$t1, \$s6
lw \$t0, 0(\$t1)
bne \$t0, \$s5, Exit
addi \$s3, \$s3, 1
j Loop
Exit: ...

80000	0	0	19	9	4	0
80004	0	9	22	9	0	32
80008	35	9	8			0
80012	5	8	21			2
80016	8	19	19			1
80020	2					20000
80024						

國立雲林科技大學資訊工程系 朱宗賢老師

- Target address = PC + 4 + address × 4
- Range = 2^{16} words = 2^{18} bytes = 256 Kb
- PC + 4 + $(2^{15} - 1) \times 4 \sim PC + 4 - (2^{15}) \times 4$
- Target address = PC_{31...28} : (address × 4)
- Range = 2^{26} words = 2^{28} bytes = 256Mb
- (PC & 0xF0000000) | 0x00000000 ~ (PC & 0xF0000000) | 0xFFFFFFF
- Note: jr \$ra #jump to the address stored in \$ra, range = 4G bytes
- 80024 = PC + 4 + address × 4 = 80012 + 4 + 2 × 4
- 80000 = PC_{31...28} : (address × 4) = 80020_{31...28} : (address × 4) = 0 : 2000 × 4

Processor - Pipeline

Pipelining Lessons

- Pipelining doesn't help latency (execution time) of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- ~~No net speedup = Number of pipe stages~~
- Time to "fill" pipeline and time to "drain" it reduces speedup
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages also reduces speedup

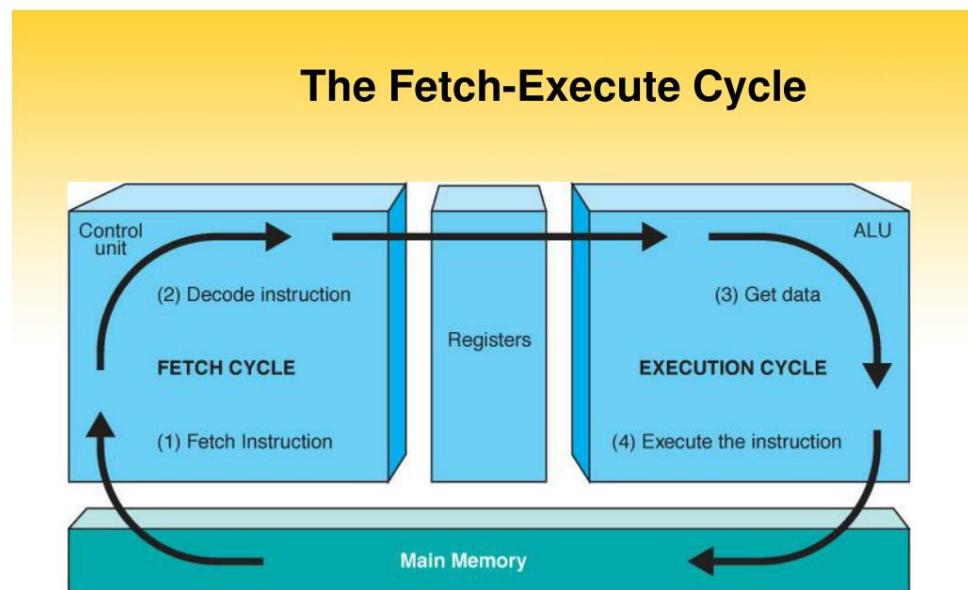
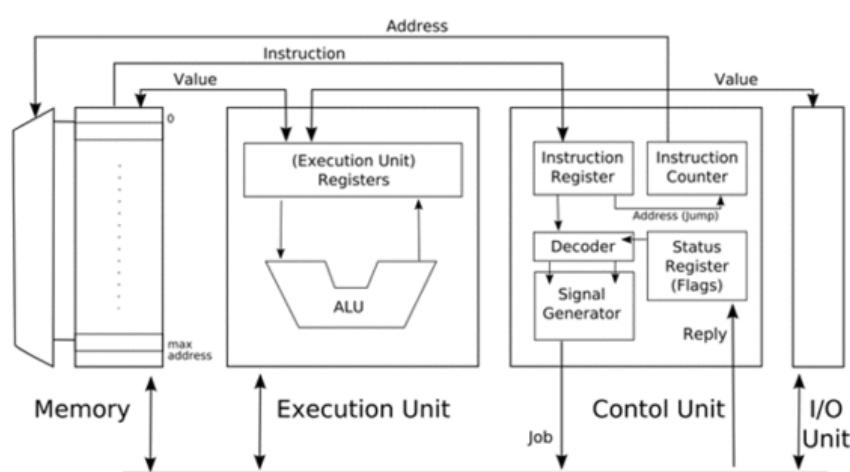
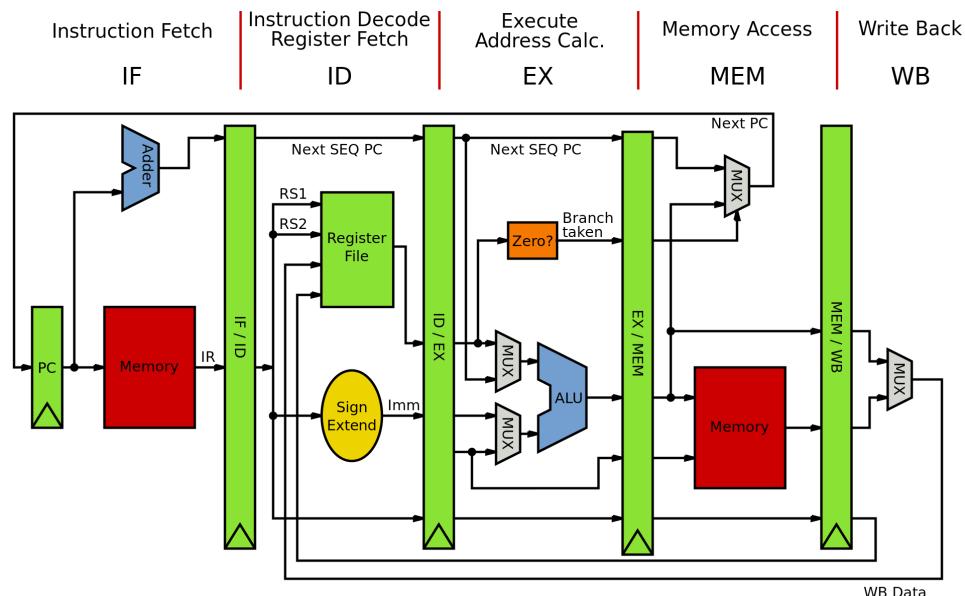
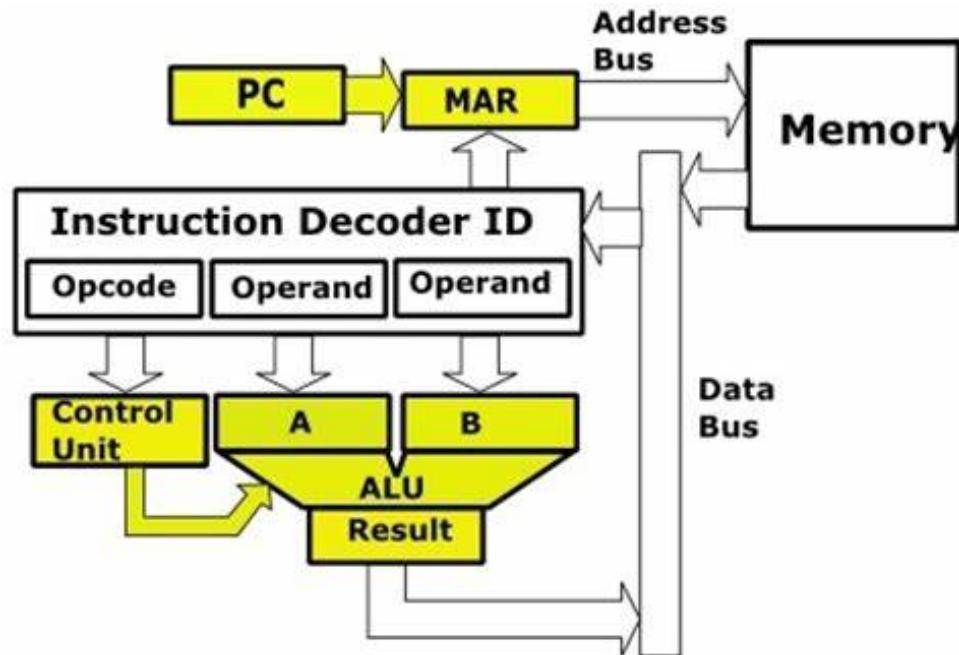


Figure 5.3 The Fetch-Execute Cycle

17

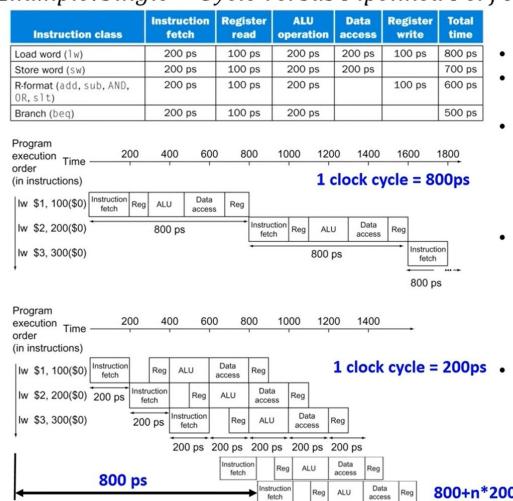
© 2011 Jones and Bartlett Publishers, LLC (www.jbpub.com)



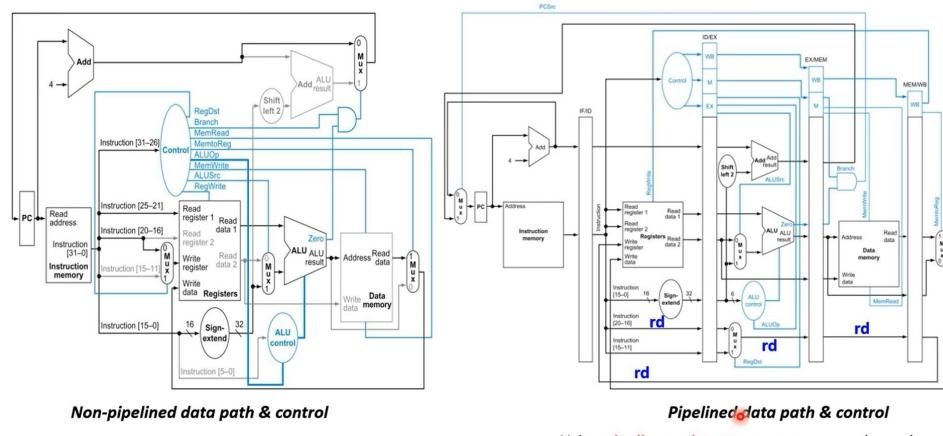


Example: Single – Cycle versus Pipelined Performance

課程: 計算機組織 / 老師: 朱宗賢



- Single-cycle: each instruction takes **one** clock cycle
- Pipeline: each stage takes **one** clock cycle
- 3 load instructions
 - The total execution time in nonpipelined design is **2400 ps**
 - The total execution time in pipelined design is **1400 ps**
- 1,000,003 load instructions
 - The total execution time in nonpipelined design is **800 ps x 1,000,003 = 800,002,400 ps**
 - The total execution time in pipelined design is **800 ps + 200 ps x 1,000,003 = 200,001,400 ps**
 - $800,002,400 / 200,001,400 = \sim 4$
- IF Register read = 200ps, Register write = 200ps
 - The total execution time in nonpipelined design is **1000 ps x 1,000,003 = 1,000,003,000 ps**
 - The total execution time in pipelined design is **800 ps + 200 ps x 1,000,003 = 200,001,400 ps**
 - $1,000,003,000 / 200,001,400 = \sim 5$

Control signals of the pipelined datapath

Pipeline Hazard

Pipeline Hazards (1)

- **Pipeline Hazards** are situations that prevent the next instruction in the instruction stream from executing in its designated clock cycle
- Hazards reduce the performance from the ideal speedup gained by pipelining
- Three types of hazards
 - **Structural hazards**
 - Arise from resource conflicts when the hardware can't support all possible combinations of overlapping instructions
 - **Data hazards**
 - Arise when an instruction depends on the results of a previous instruction in a way that is exposed by overlapping of instruction in pipeline
 - **Control hazards**
 - Arise from the pipelining of branches and other instructions that change the PC (Program Counter)

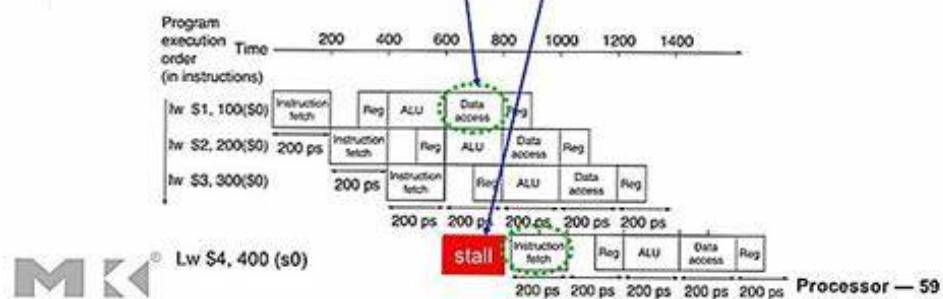
Pipeline Hazards (2)

- Hazards in pipeline can make the pipeline to *stall*
- Eliminating a hazard often requires that some instructions in the pipeline to be allowed to proceed while others are delayed
 - When an instruction is stalled, instructions issued *latter* than the stalled instruction are stopped, while the ones issued *earlier* must continue
- No new instructions are fetched during the stall

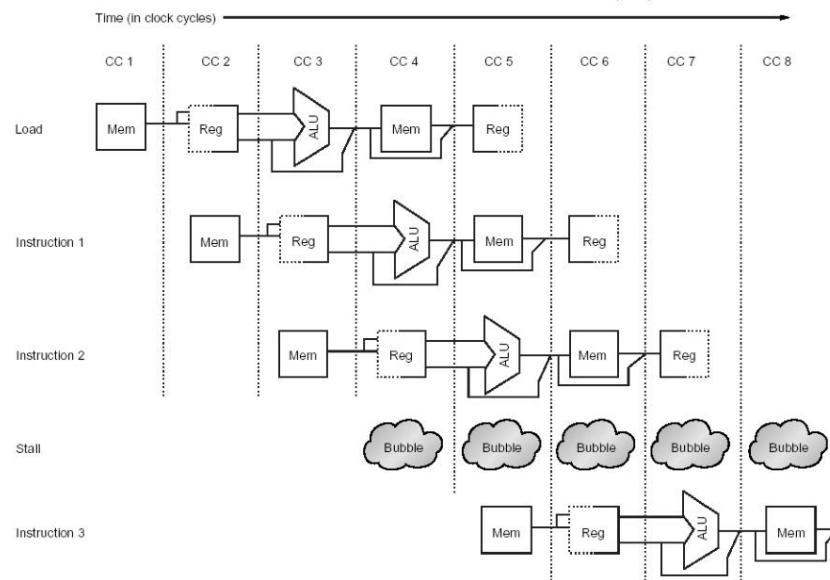
Solution for Pipeline Hazards

Structure Hazards

- Conflict for use of a resource
- Suppose that we has only a single memory instead of two memories (instruction and data) In the MIPS design
 - Load/store requires data access
 - Instruction fetch would have to **stall** for that cycle
 - Would cause a pipeline "bubble"
- Hence, pipelined datapaths require separate instruction/data memories



Structural Hazards (3)



- Stall cycle added (commonly called pipeline *bubble*)

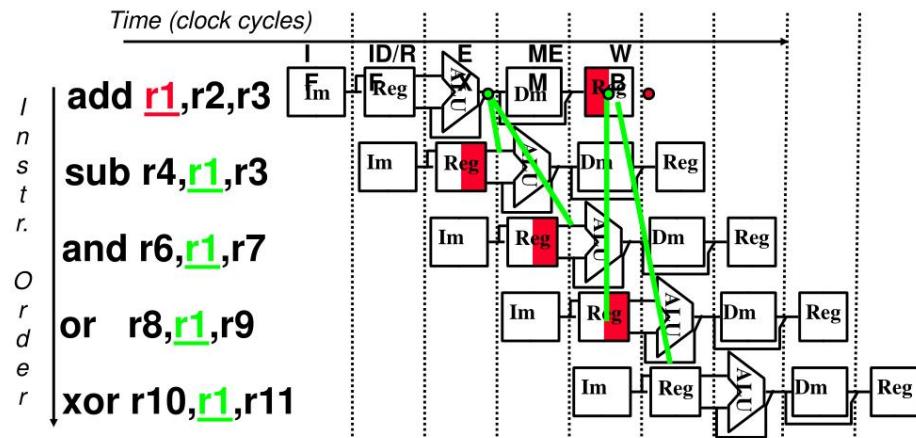
Structural Hazard

- ◆ Different instructions using the same resource at the same time
- ◆ Register File:
 - ❖ Accessed in 2 stages:
 - Read during stage 2 (ID)
 - Write during stage 5 (WB)
 - ❖ Solution: 2 read ports, 1 write port
- ◆ Memory
 - ❖ Accessed in 2 stages:
 - Instruction Fetch during stage 1 (IF)
 - Data read/write during stage 4 (MEM)
 - ❖ Solution: separate instruction cache and data cache
- ◆ Each functional unit can only be used once per instruction
- ◆ Each functional unit must be used at the same stage for all instructions

Pipeline - Data Hazard

Data Hazard Solution:

- “Forward” result from one stage to another



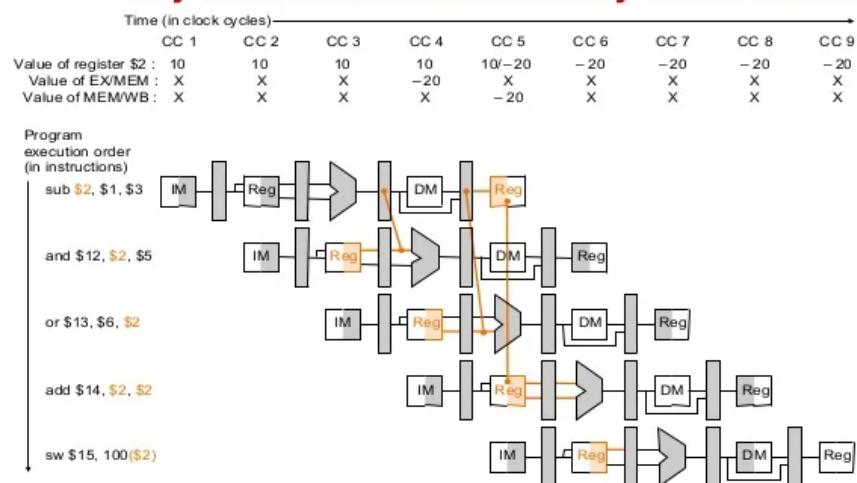
- “or” OK if define read/write properly

cs 152 L1 3.13

DAP Fa97, © U.CB

Data Hazard Solution: Forwarding

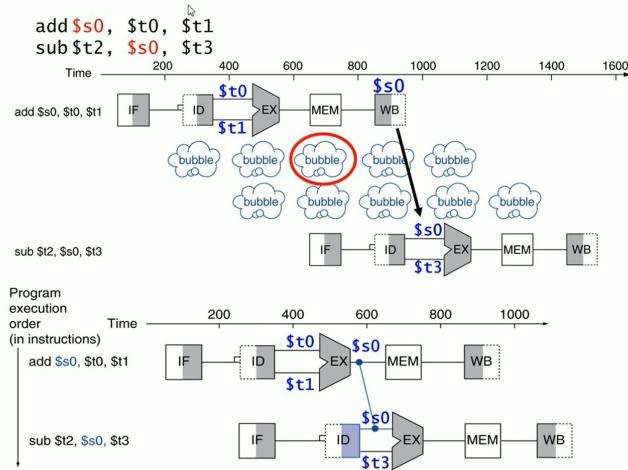
- Key idea: connect data internally before it's stored



CSCE430/830

Pipeline Hazards

Example: Forwarding with Two instructions



Forwarding (or bypassing):
retrieving the missing data element from internal buffer rather than waiting for it to arrive from registers or memory.

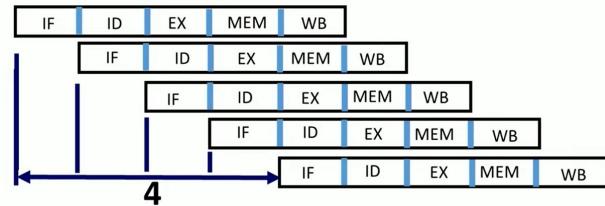
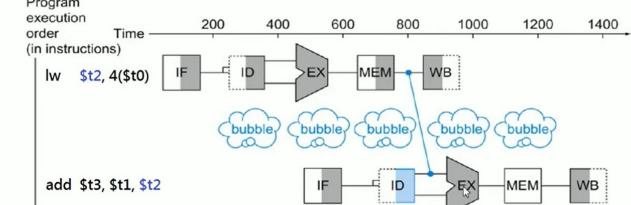
Example: Reordering Code to Avoid Pipeline Stalls

C code for $A = B + E; C = B + F;$

```
lw $t1, 0($t0)
lw $t2, 4($t0)
★ add $t3, $t1, $t2
sw $t3, 12($t0)
lw $t4, 8($t0)
★ add $t5, $t1, $t4
sw $t5, 16($t0)
```

13 cycles = 4 + 1x7 + 2

We need a stall even with forwarding
when an R-format instruction
following a load tries to use the data.

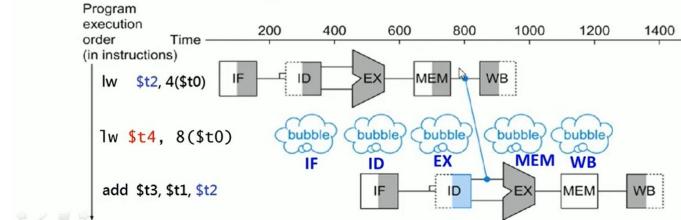


Example: Reordering Code to Avoid Pipeline Stalls

<pre>lw \$t1, 0(\$t0) lw \$t2, 4(\$t0) stall → lw \$t4, 8(\$t0) stall → add \$t3, \$t1, \$t2 sw \$t3, 12(\$t0) lw \$t4, 8(\$t0) add \$t5, \$t1, \$t4 sw \$t5, 16(\$t0)</pre>	<pre>lw \$t1, 0(\$t0) lw \$t2, 4(\$t0) lw \$t4, 8(\$t0) add \$t3, \$t1, \$t2 sw \$t3, 12(\$t0) add \$t5, \$t1, \$t4 sw \$t5, 16(\$t0)</pre>
--	---

13 cycles

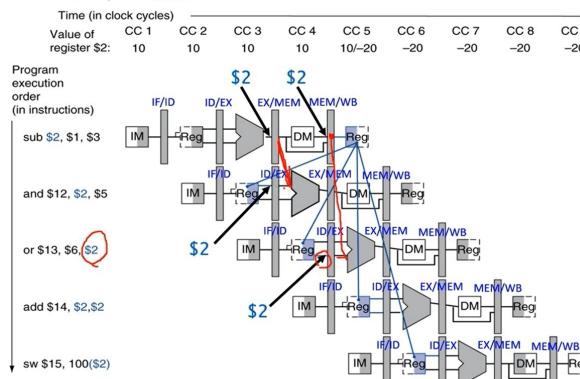
11 cycles



1. Instruction "lw \$t4, 8(\$t0)" does not change the input of "add \$t3, \$t1, \$t2" and "sw \$t3, 12(\$t0)".
2. Instructions "add \$t3, \$t1, \$t2" and "sw \$t3, 12(\$t0)" do not change the input of "lw \$t4, 8(\$t0)".

Dependence Detection

```
sub $2, $1,$3
and $12,$2,$5
or $13,$6,$2
add $14,$2,$2
sw $15,100($2)
```



課程: 計算機組織 / 老師: 朱宗賢

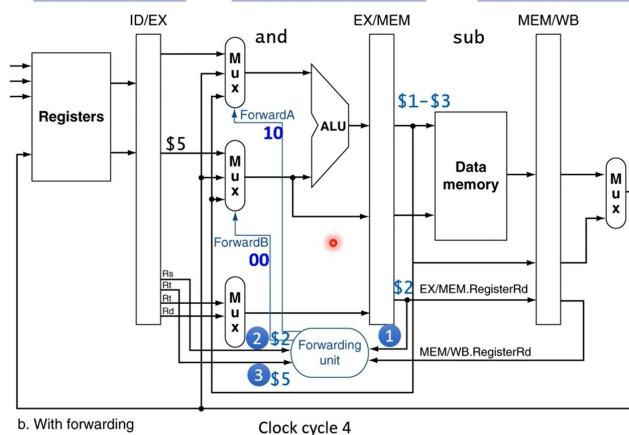
- Non-independent program
 - Detect hazards
 - Forward proper data
- Data hazards when
 - $\text{EX/MEM.RegisterRd} = \text{ID/EX.RegisterRs}$
 - $\text{EX/MEM.RegisterRd} = \text{ID/EX.RegisterRt}$
 - $\text{MEM/WB.RegisterRd} = \text{ID/EX.RegisterRs}$
 - $\text{MEM/WB.RegisterRd} = \text{ID/EX.RegisterRt}$
- But only if forwarding instruction will write to a register!
 - $\text{EX/MEM.RegWrite}=1, \text{MEM/WB.RegWrite}=1$
- And only if Rd for that instruction is not zero
 - $\text{EX/MEM.RegisterRd} \neq 0, \text{MEM/WB.RegisterRd} \neq 0$
- R0: Reserved, read as zero, ignore writes by software.

```
sll $0,$1,3
and $12,$0,$5
or $13,$6,$0
add $14,$0,$0
sw $15,100($0)
```

Independent program

EX Data Hazard – conditions & control signals

- Conditions & control signals
- if (EX/MEM.RegWrite and ($\text{EX/MEM.RegisterRd} \neq 0$) and ($\text{EX/MEM.RegisterRd} = \text{ID/EX.RegisterRs}$)) $\text{ForwardA} = 10$
 if (EX/MEM.RegWrite and ($\text{EX/MEM.RegisterRd} \neq 0$) and ($\text{EX/MEM.RegisterRd} = \text{ID/EX.RegisterRt}$)) $\text{ForwardB} = 10$



課程: 計算機組織 / 老師: 朱宗賢

- ①
 $\text{sub } \$2, \$1, \$3$
 $\text{and } \$12, \$2, \$5$ EX data hazard
 $\text{or } \$13, \$6, \$2$ $\text{ForwardA} = 10$
 $\text{add } \$14, \$2, \$2$
 $\text{sw } \$15, 100(\$2)$
- ②
 $\text{sub } \$2, \$1, \$3$
 $\text{and } \$12, \$0, \$5$ EX data hazard
 $\text{or } \$13, \$6, \$0$ $\text{ForwardA} = 10$
 $\text{add } \$14, \$0, \$0$
 $\text{sw } \$15, 100(\$0)$
- ③
 $\text{sub } \$2, \$1, \$3$
 $\text{and } \$12, \$2, \$5$ EX data hazard
 $\text{or } \$13, \$6, \$2$ $\text{ForwardA} = 10$
 $\text{add } \$14, \$2, \$2$
 $\text{sw } \$15, 100(\$2)$

Mux control	Source
ForwardA = 00	ID/EX
ForwardA = 10	EX/MEM
ForwardA = 01	MEM/WB
ForwardB = 00	ID/EX
ForwardB = 10	EX/MEM
ForwardB = 01	MEM/WB

b. With forwarding

Clock cycle 4

MEM Data Hazard – conditions & signals

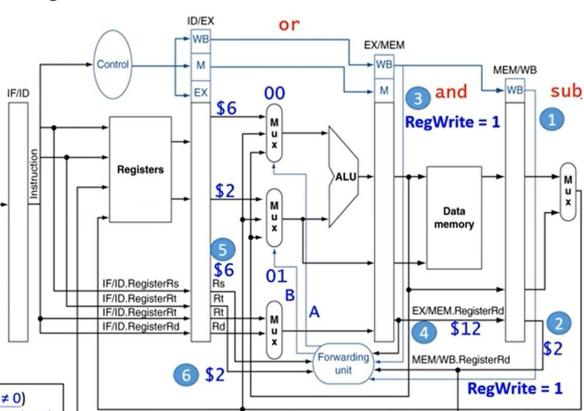
課程: 計算機組織 / 老師: 朱宗賢

```
sub $2, $1,$3
and $12,$2,$5
or $13,$6,$2
add $14,$2,$2
sw $15,100($2)
```

MEM data hazard

Mux control	Source
ForwardA = 00	ID/EX
ForwardA = 10	EX/MEM
ForwardA = 01	MEM/WB
ForwardB = 00	ID/EX
ForwardB = 10	EX/MEM
ForwardB = 01	MEM/WB

2. if (MEM/WB.RegWrite and ($\text{MEM/WB.RegisterRd} \neq 0$)
 and not (EX/MEM.RegWrite and ($\text{EX/MEM.RegisterRd} \neq 0$)
 and ($\text{EX/MEM.RegisterRd} = \text{ID/EX.RegisterRt}$))
 and ($\text{MEM/WB.RegisterRd} = \text{ID/EX.RegisterRt}$))



MEM data hazard happens in clock cycle 5!

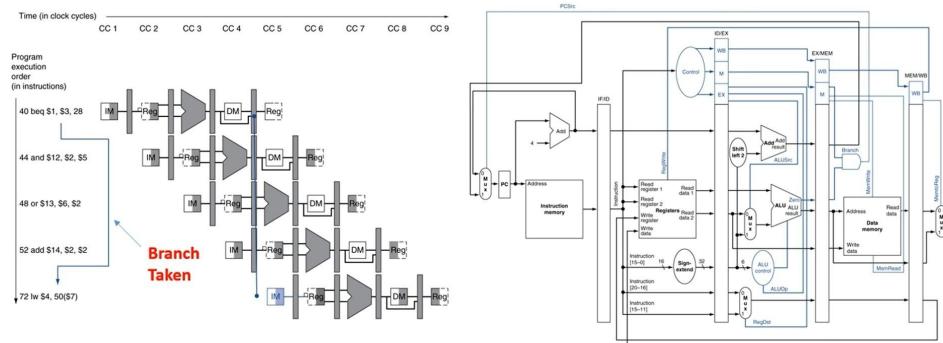
The image you are requesting does not exist or is no longer available.

imgur.com

Pipeline - Control Hazard

Control Hazards – Performance Impact and Solutions Overview 課程:計算機組織 / 老師:朱宗賢

- Control hazards: attempt to make branching decisions before branch condition is evaluated
(1) Taken: $PC = PC + 4 + \text{Offset}$, (2) Not Taken: $PC = PC + 4$
- The decision about whether to branch doesn't occur until the MEM stage
- Branches have a penalty of 3 clock cycles!



Summary - Control Hazard Solutions

- **Stall** - stop fetching instr. until result is available
 - Significant performance penalty
 - Hardware required to stall
- **Predict** - assume an outcome and continue fetching (undo if prediction is wrong)
 - Performance penalty only when guess wrong
 - Hardware required to "squash" instructions
- **Delayed branch** - specify in architecture that following instruction is always executed
 - Compiler re-orders instructions into delay slot
 - Insert "NOP" (no-op) operations when can't use (~50%)
 - This is how original MIPS worked

Control Hazard Review

The nub of the problem:

- In what pipeline stage does the processor fetch the next instruction?
- If that instruction is a conditional branch, when does the processor know whether the conditional branch is taken (execute code at the target address) or not taken (execute the sequential code)?
- What is the difference in cycles between them?

The cost of stalling until you know whether to branch

- number of cycles in between * branch frequency = the contribution to CPI due to branches

Predict the branch outcome to avoid stalling

Spring 2003

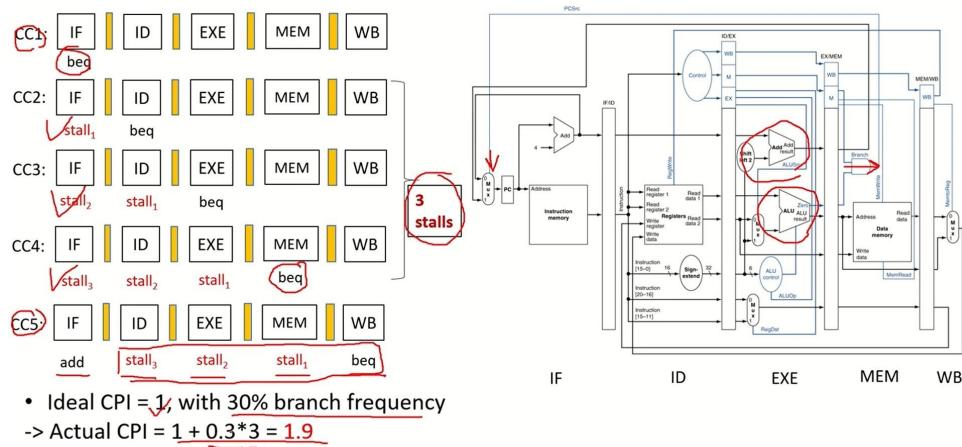
CSE P548

1

Control Hazards – Reducing the Delay of Branches

國立雲林科技大學資工系 朱宗賢老師
「計算機組織」課程

- Control hazard: the flow of instruction addresses is not what the pipeline expect

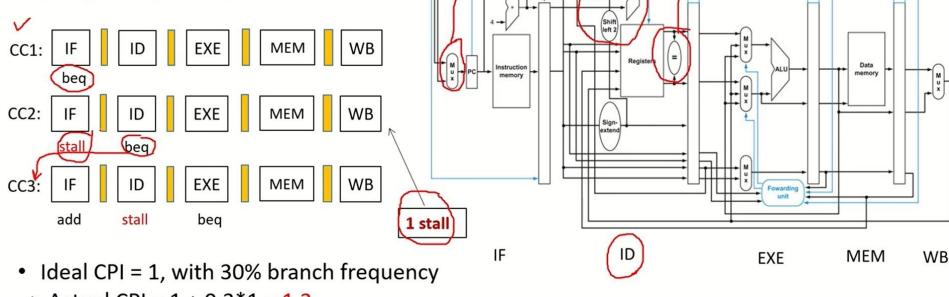


- Ideal CPI = 1, with 30% branch frequency
 \rightarrow Actual CPI = $1 + 0.3 \times 3 = 1.9$

Control Hazards – Reducing the Delay of Branches

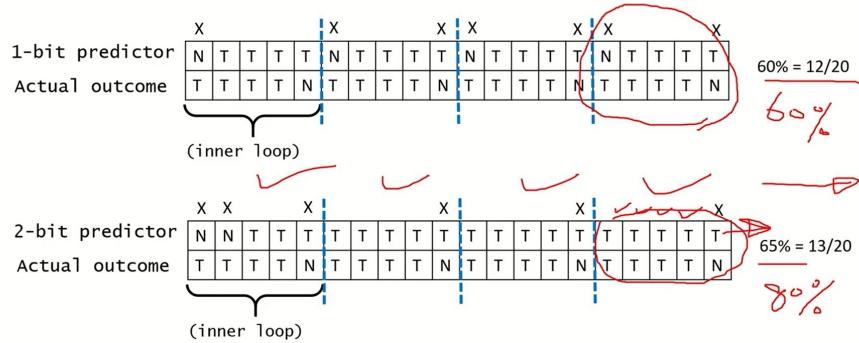
國立雲林科技大學資工系 朱宗賢老師
「計算機組織」課程

- Move hardware to determine outcome from EXE to ID stage:
 - ✓ (1) Target address adder
 - ✓ (2) Register comparator
- One clock-cycle stall on branches.



- Ideal CPI = 1, with 30% branch frequency
 \rightarrow Actual CPI = $1 + 0.3 \times 1 = 1.3$

Control Hazard Solution – Dynamic Branch Prediction

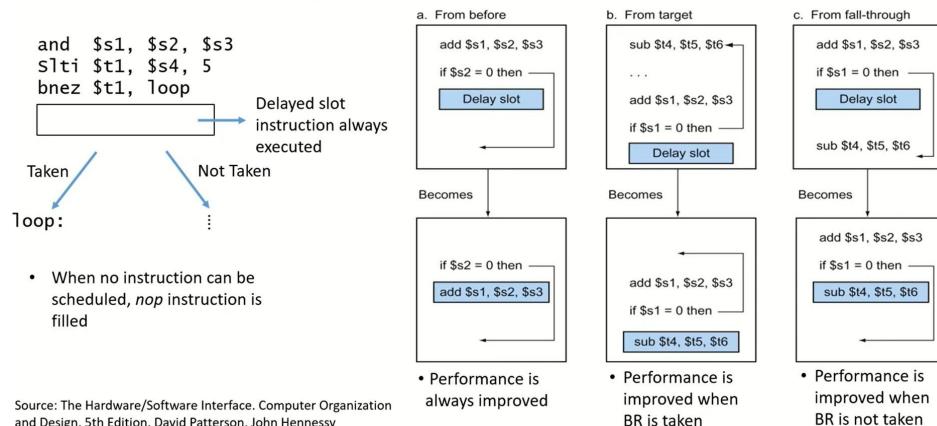
國立雲林科技大學資工系 朱宗賢老師
「計算機組織」課程

- 1-bit predictor: we change our mind if we mispredict once about the branch
- 2-bit predictor: we change our mind if we mispredict twice about the branch
 - Works well for loop-intensive programs

Control Hazard Solution – Scheduling branch-delay slot

國立雲林科技大學資工系 朱宗賢老師
「計算機組織」課程

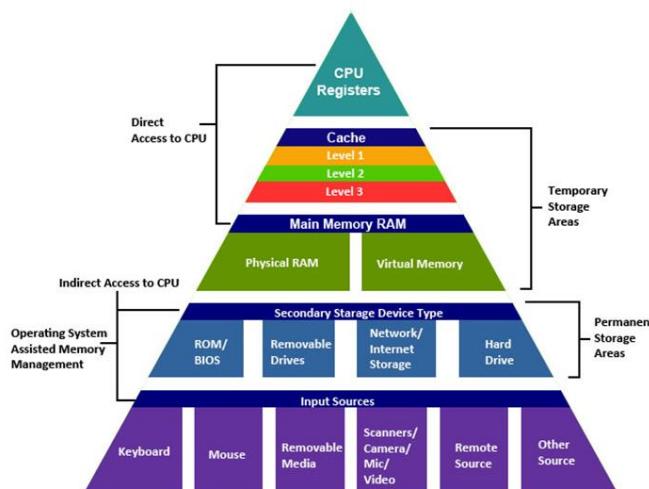
- Branch delay slot: the slot directly after a delayed branch instruction, which in the MIPS architecture is filled by an instruction that does not affect the branch



Source: The Hardware/Software Interface. Computer Organization and Design, 5th Edition, David Patterson, John Hennessy

Memory Hierarchy

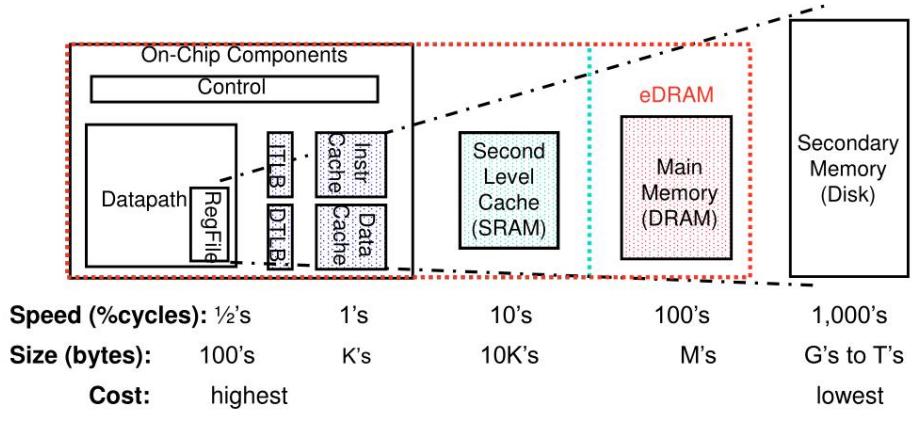
The Memory Hierarchy

<http://cse1.net/recaps/4-memory.html>

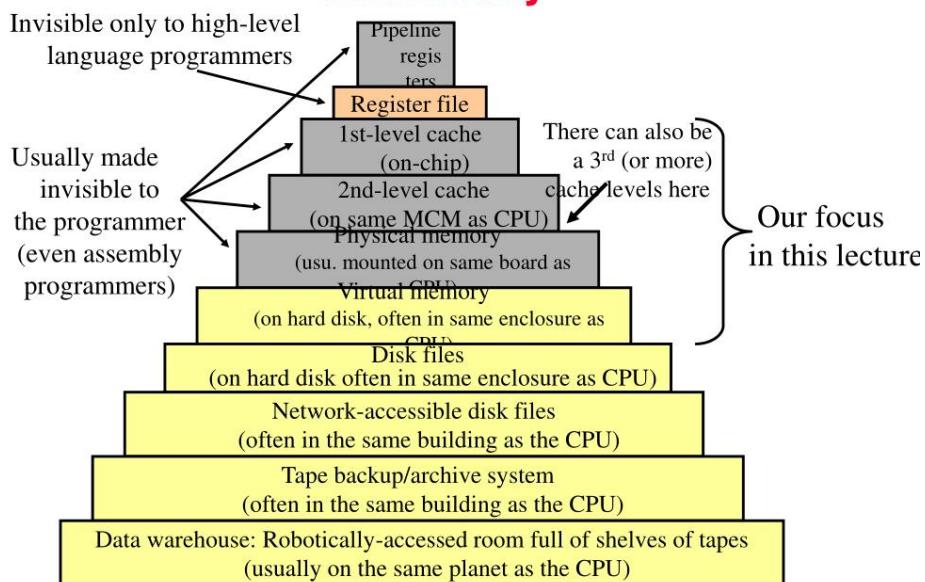
2

A Typical Memory Hierarchy

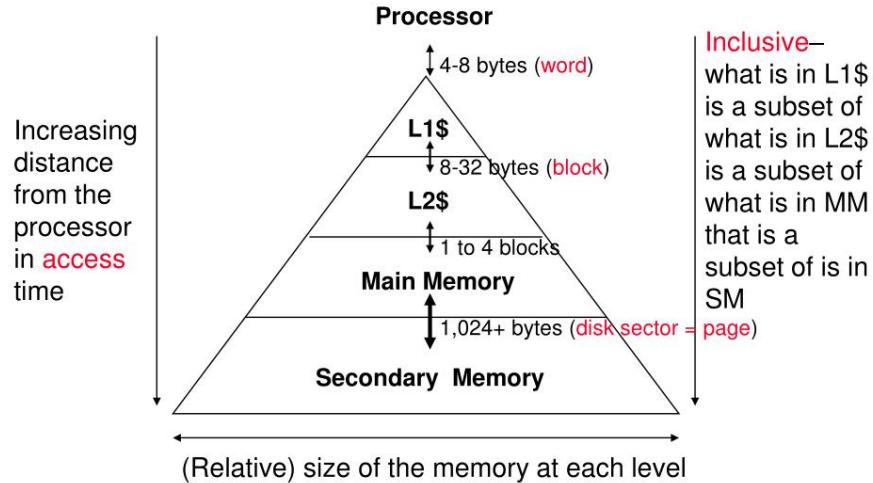
- ❑ By taking advantage of the principle of locality
 - Can present the user with as much memory as is available in the cheapest technology
 - at the speed offered by the fastest technology



Many Levels in Memory Hierarchy



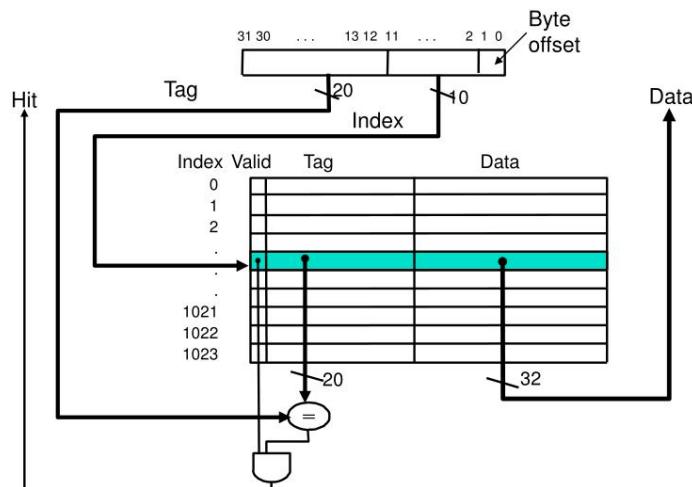
Characteristics of the Memory Hierarchy



Cache

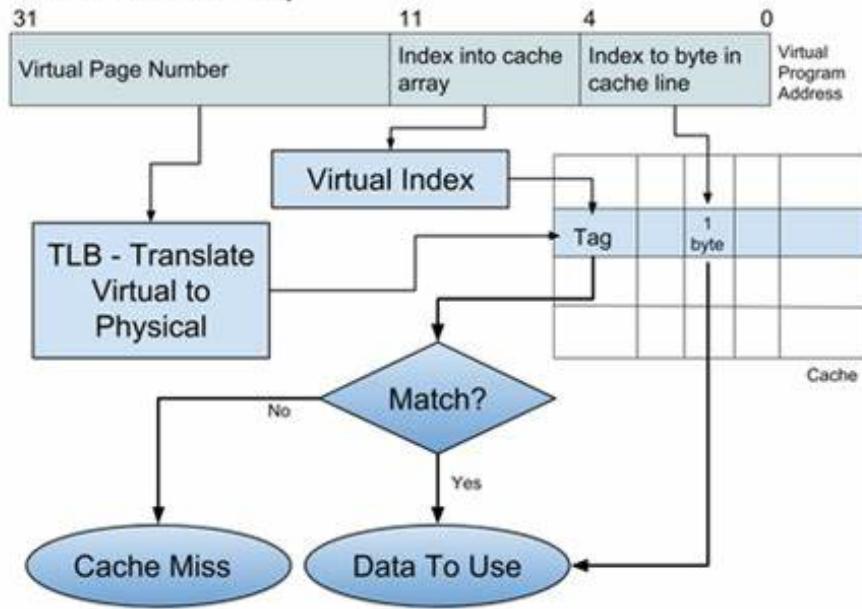
MIPS Direct Mapped Cache Example

- ❑ One word/block, cache size = 1K words



What kind of locality are we taking advantage of?

MIPS Cache Look Up

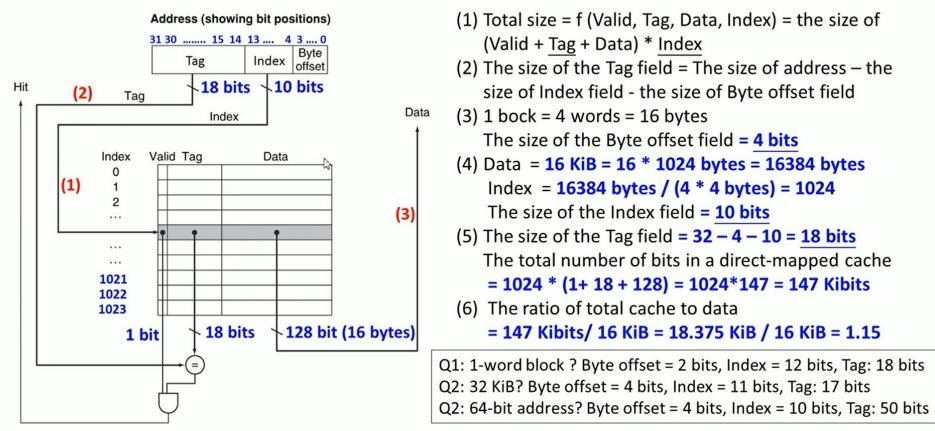


Adapted from Imagination Technologies MIPS basic training course materials

Example: Bits in a Cache

課程:計算機組織 / 老師:朱宗賢

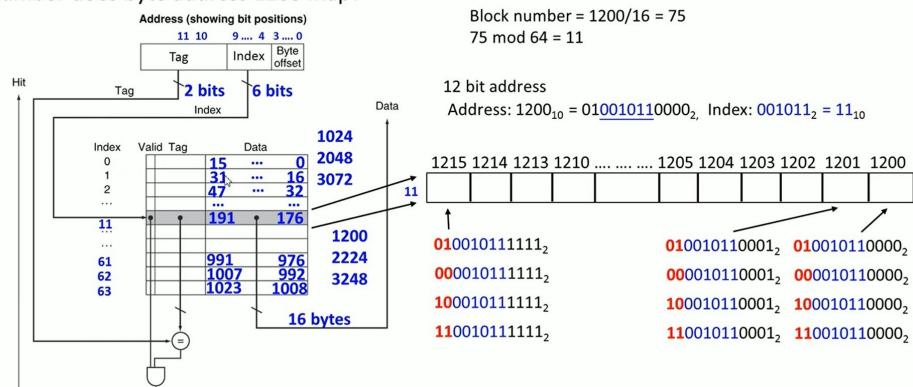
A direct-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address.
(unit : 1 KiB (kibibyte) = 1024 byte, 1 Kibit (Kibibit) = 1024 bit)



Example: Mapping a Address to a Multiword Cache Block

課程:計算機組織 / 老師:朱宗賢

Consider a direct-mapped cache with 64 blocks and a block size of 16 bytes. To what block number does byte address 1200 map?



Cache Performance

Example: Calculating Cache Performance

課程:計算機組織 / 老師:朱宗賢

- Given
 - I-cache miss rate = 2%, D-cache miss rate = 4%, Miss penalty = 100 cycles
 - Load & stores are 36% of instructions (ex: lw, sw)
 - Base CPI (ideal cache) = 2
- Miss cycles per instruction
 - I-cache: $0.02 \times 100 = 2$ cycles, D-cache: $0.36 \times 0.04 \times 100 = 1.44$ cycles
- Actual CPI
 - Base CPI + I-cache miss cycles + D-cache miss cycles = $2 + 2 + 1.44 = 5.44$ cycles
- The amount of execution time spent on memory stalls
 - $3.44 / 5.44 = 63\%$
- Ideal CPU is $5.44/2 = 2.72$ times faster

Q: Base CPI = 1, Actual CPI = 4.44, memory stalls = $3.44 / 4.44 = 77\%$, Ideal CPU is $4.44/1 = 4.44$ times faster

Cache Performance Equations

- **Memory stalls per program (blocking cache):**

$$\text{MemoryStallCycle} = IC \times \left(\frac{\text{MemoryAccesses}}{\text{Instruction}} \right) \times \text{MissRate} \times \text{MissPenalty}$$

$$\text{MemoryStallCycle} = IC \times \left(\frac{\text{Misses}}{\text{Instruction}} \right) \times \text{MissPenalty}$$

- **CPU time formula:**

$$\text{CPU Time} = IC \times (CPI_{\text{Execu}} + \frac{\text{MemoryStallCycle}}{\text{Instruction}}) \times \text{CycleTime}$$

- **More cache performance will be given later!**

46

Cache Performance

- $\text{CPI}_{\text{contributed by cache}} = \text{CPI}_c$
 $= \text{miss rate} * \text{number of cycles to handle the miss}$
- Another important metric
 $\text{Average memory access time} = \text{cache hit time} * \text{hit rate}$
 $+ \text{Miss penalty} * (1 - \text{hit rate})$

Example: Calculating Average Memory Access Time

課程:計算機組織 / 老師:朱宗賢

- Average memory access time (AMAT)
AMAT = Hit time + Miss rate × Miss penalty
- Given:
CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, miss rate = 5%
 $AMAT = 1 + 0.05 \times 20 = 2\text{ns}$ (2 cycles per instruction)
- Design strategy
 - Reducing cache misses by more flexible placement of blocks
 - Reducing the miss penalty using multilevel caches

Improving Cache Performance

• Miss Rate Reduction Techniques:

- * Increased cache capacity
- * Higher associativity
- * Hardware prefetching of instructions and data
- * Compiler-controlled prefetching
- * Larger block size
- * Victim caches
- * Pseudo-associative Caches
- * Compiler optimizations

• Cache Miss Penalty Reduction Techniques:

- * Giving priority to read misses over writes
- * Early restart and critical word first
- * Second-level cache (L_2)
- * Sub-block placement
- * Non-blocking caches

• Cache Hit Time Reduction Techniques:

- * Small and simple caches
- * Avoiding address translation during cache indexing
- * Pipelining writes for fast write hits

EECC551 - Shaaban

#7 Lec # 10 Winter2000 1-23-2000

Qualitative Cache Performance Model

Miss Types

- **Compulsory ("Cold Start") Misses**
 - First access to line not in cache
- **Capacity Misses**
 - Active portion of memory exceeds cache size
- **Conflict Misses**
 - Active portion of address space fits in cache, but too many lines map to same cache entry
 - Direct mapped and set associative placement only
- **Coherence Misses**
 - Block invalidated by multiprocessor cache coherence mechanism

Hit Types

- **Temporal locality hit**
 - Accessing same word that previously accessed
- **Spatial locality hit**
 - Accessing word spatially near previously accessed word

- 42 -

CS 740 F'07

Cache Hit and Cache Miss

Types of Cache Misses: *The Three C's*

- 1 Compulsory:** On the first access to a block; the block must be brought into the cache; also called cold start misses, or first reference misses.
- 2 Capacity:** Occur because blocks are being discarded from cache because cache cannot contain all blocks needed for program execution (program working set is much larger than cache capacity).
- 3 Conflict:** In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame; also called collision misses or interference misses.

EECC551 - Shaaban

#1 Lec # 8 Winter 2001 1-30-2002

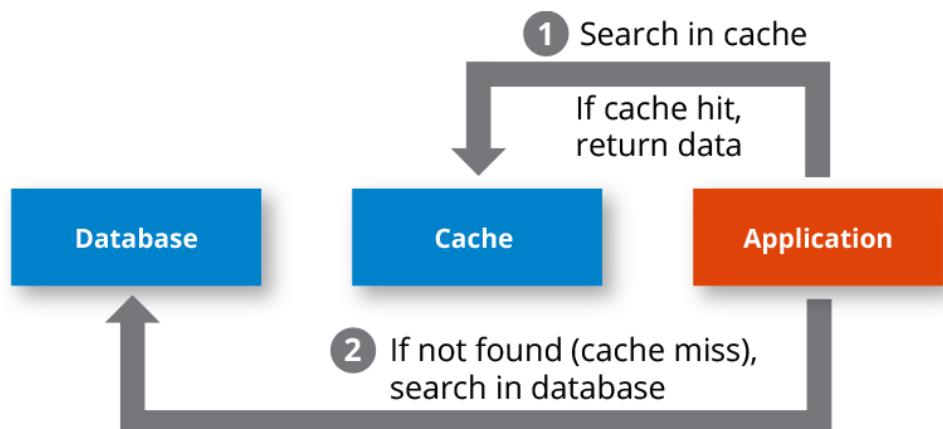
$$\frac{\# \text{ of cache hits}}{(\# \text{ of cache hits} + \# \text{ of cache misses})} = \text{Hit ratio}$$

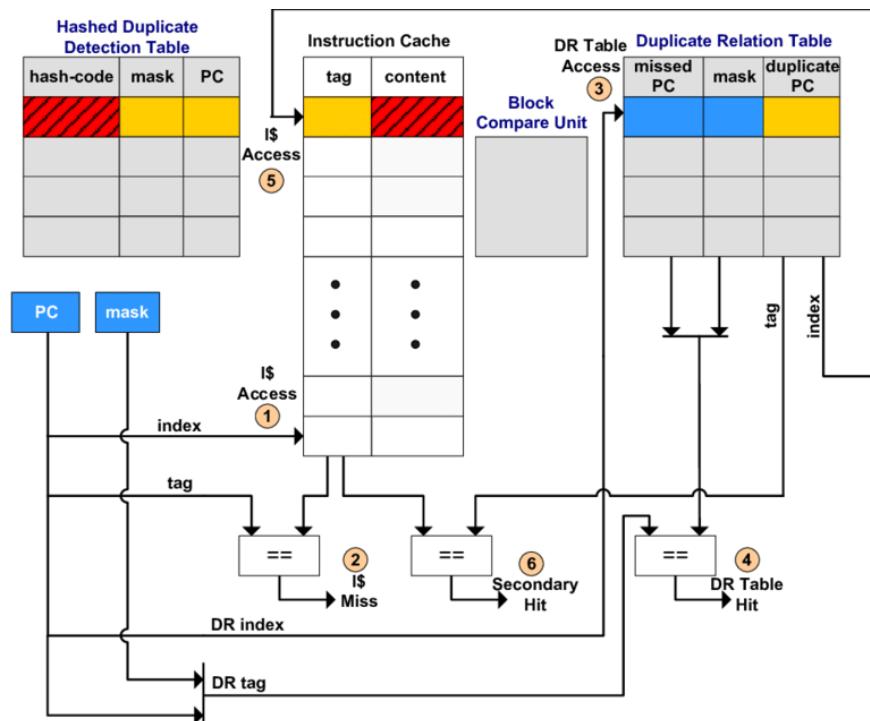
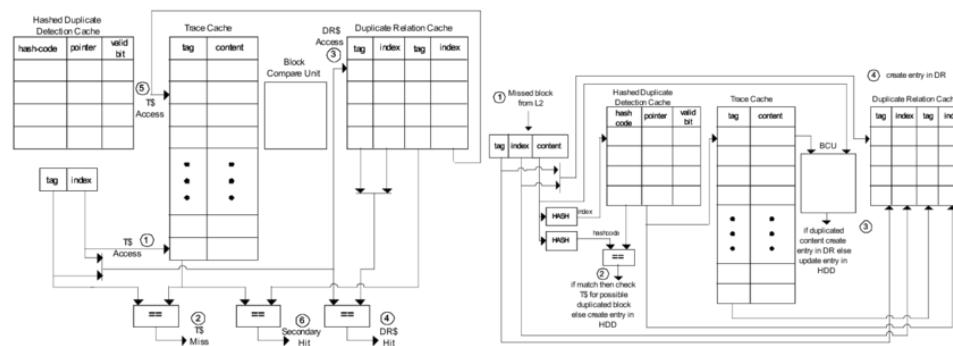
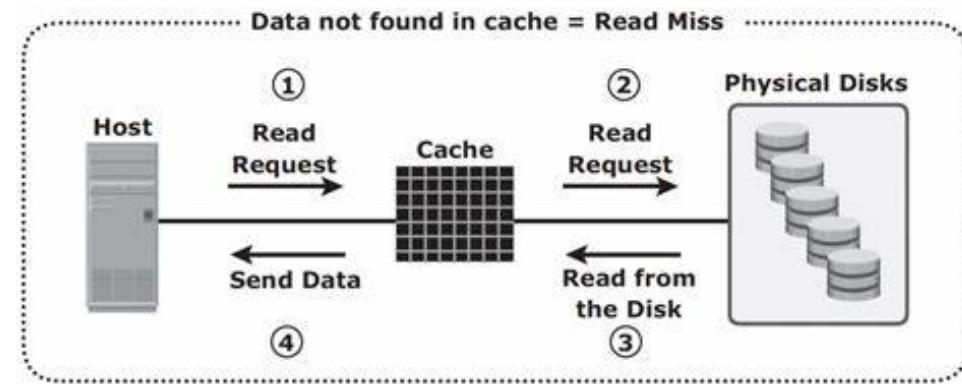
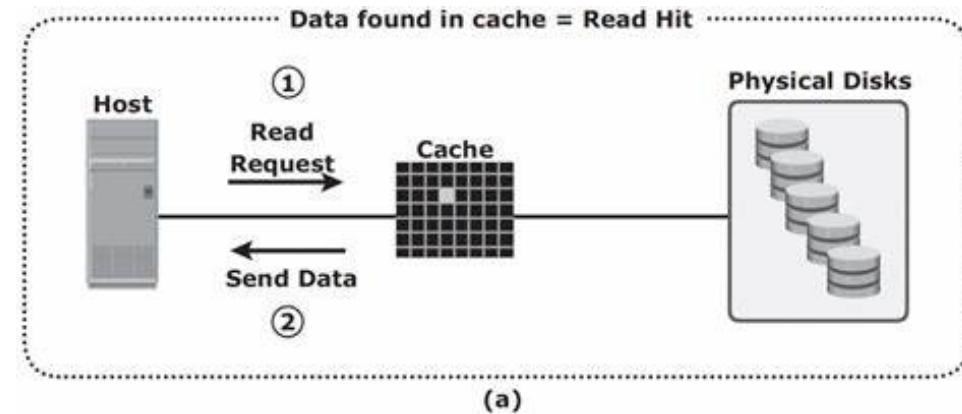
OR

$$\text{Hit ratio} = 1 - \text{Miss ratio}$$

$$\text{Avg mem access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times \text{Miss penalty}_{L1}$$
$$\text{Miss penalty}_{L1} = \text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2}$$

Handle Cache Miss





Associative Cache

Example: Misses and Associativity in Caches

課程: 計算機組織 / 老師: 朱宗賢

Given a 4-block cache and block access sequence (0, 8, 0, 6, 8), find the number of misses for please compare direct mapped, 2-way set associative, fully associative cache organization

	Ans: 5 misses	direct mapped	Block address	Cache index	Hit/miss	Cache content after access			
			0	0	miss	Mem[0]			
			8	0	miss	Mem[8]			
0	0	miss	Mem[0]						
6	2	miss	Mem[0]		Mem[6]				
8	0	miss	Mem[8]		Mem[6]				

	Ans: 4 misses	2-way set associative	Block address	Cache index	Hit/miss	Cache content after access			
			0	0	miss	Mem[0]			
			8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]					
6	0	miss	Mem[0]	Mem[6]					
8	0	miss	Mem[8]	Mem[6]					

	Ans: 3 misses	fully associative	Block address		Hit/miss	Cache content after access			
			0		miss	Mem[0]			
			8		miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	Mem[8]					
6		miss	Mem[0]	Mem[8]	Mem[6]				
8		hit	Mem[0]	Mem[8]	Mem[6]				

(Block number) mod (Number of blocks in the cache)

Step1: (Block number) mod (Number of sets in the cache)

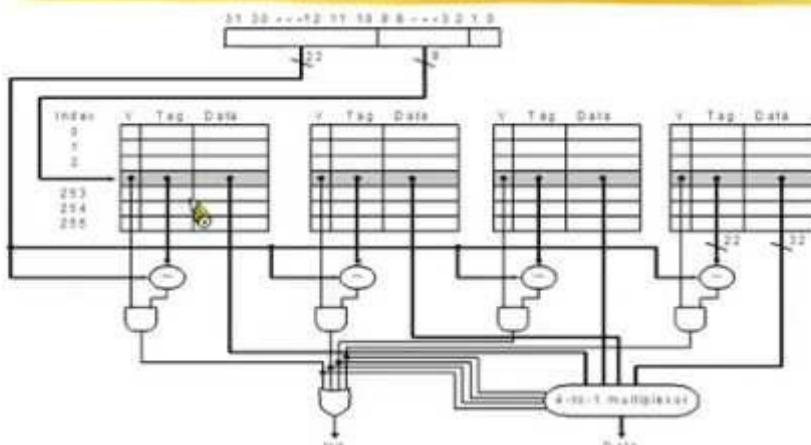
Step2: Search all tags of the set

LRU (least recently used) Policy

Q1: 4 blocks > 8 blocks

Q2: 4 blocks > 16 blocks

A 4-Way Set-Associative Cache



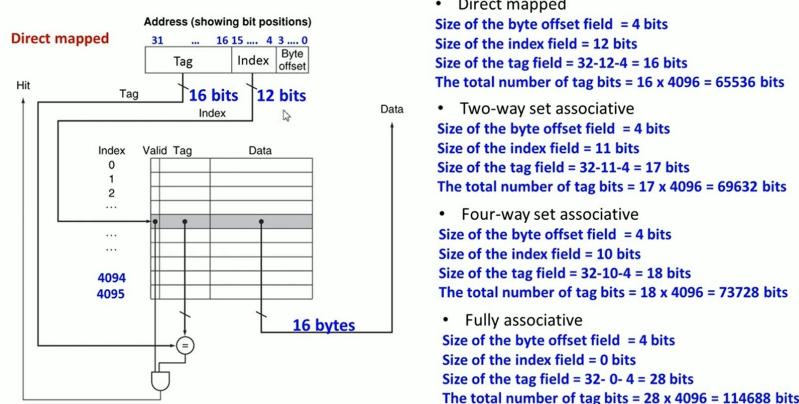
♦ Increasing associativity shrinks index, expands tag

Memory-19

Computer Architecture

課程: 計算機組織 / 老師: 朱宗賢

Q: Assuming a cache of 4096 blocks, a 4-word block size, and a 32-bit address, find the total number of set and the total number of tag bits for caches that are direct mapped, two-way set associative and fully associative.



Block Size Tradeoff

Miss Rate vs. Block Size

Block size	Cache size				
	1K	4K	16K	64K	256K
16	15.05%	8.57%	3.94%	2.04%	1.09%
32	13.34%	7.24%	2.87%	1.35%	0.70%
64	13.76%	7.00%	2.64%	1.06%	0.51%
128	16.64%	7.78%	2.77%	1.02%	0.49%
256	22.01%	9.51%	3.29%	1.15%	0.49%

FIGURE 5.12 Actual miss rate versus block size for five different-sized caches in Figure 5.11. Note that for a 1-KB cache, 64-byte, 128-byte, and 256-byte blocks have a higher miss rate than 32-byte blocks. In this example, the cache would have to be 256 KB in order for a 256-byte block to decrease misses.

2/15/99

CS520S99 Memory

C. Edward Chow Page 34

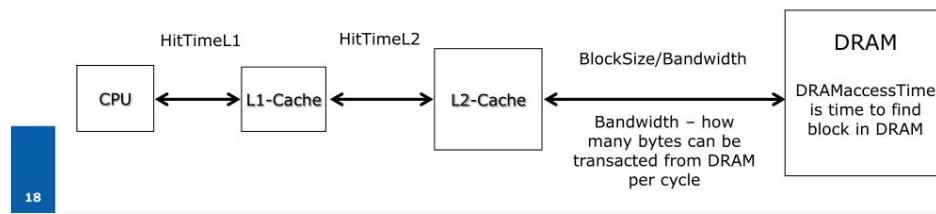
Block Size Tradeoffs

- Larger block sizes...
 - Take advantage of spatial locality
 - Incur larger miss penalty since it takes longer to transfer the block into the cache
 - Can increase the average hit time and miss rate
- Average Access Time (AMAT) = HitTime + MissPenalty*MR



2-level Cache Performance Equations

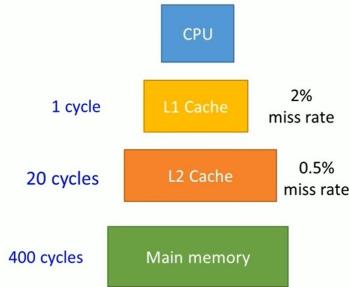
- L1 AMAT = HitTimeL1 + MissRateL1 * MissPenaltyL1
 - MissLatencyL1 is low, so optimize HitTimeL1
- MissPenaltyL1 = HitTimeL2 + MissRateL2 * MissPenaltyL2
 - MissLatencyL2 is high, so optimize MissRateL2
 - MissPenaltyL2 = DRAMaccessTime + (BlockSize/Bandwidth)
 - If DRAM time high or bandwidth high, use larger block size
- L2 miss rate:
 - Global: L2 misses / total CPU references
 - Local: L2 misses / CPU references that miss in L1
 - The equation above assumes local miss rate



Example: Performance of Multilevel Caches

課程: 計算機組織 / 老師: 朱宗賢

- Given
 - CPU base CPI = 1, clock rate = 4GHz, Miss rate/instruction = 2%, Main memory access time = 100ns
- With just primary cache
 - Miss penalty = $100\text{ns}/0.25\text{ns} = 400 \text{ cycles}$, Effective CPI = $1 + 0.02 \times 400 = 9$
- Now add L-2 cache
 - Access time = 5ns, Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
 - Penalty = $5\text{ns}/0.25\text{ns} = 20 \text{ cycles}$
- Primary miss with L-2 miss
 - CPI = $1 + 0.02 \times 20 + 0.005 \times 400 = 1 + 0.4 + 2 = 3.4$
 - Performance ratio = $9/3.4 = 2.6$
- Multilevel Cache Considerations
 - Primary cache - focus on minimal hit time
 - L-2 cache - focus on low miss rate
- Miss rate requirement of L-2 cache
 - $1 + 0.02 \times 20 + R \times 400 < 9 \rightarrow R < 1.9\%$

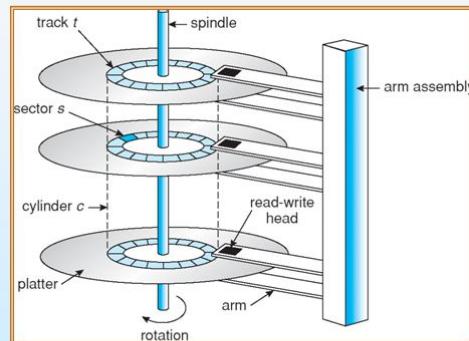


Storage Management

Disk Structure



Disk Structure



Moving-head disk mechanism

- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**

- The logical block is the smallest unit of transfer, usually 512 bytes

- The array of logical blocks is mapped into the **sectors** of the disk sequentially

- Sector 0 is the first sector of the first **track** on the outermost **cylinder**

- Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost

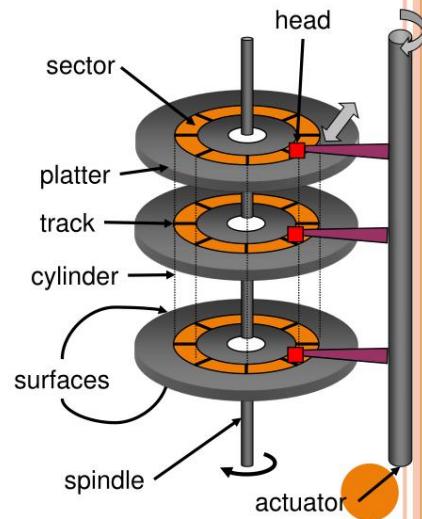
Operating System Concepts – 7th Edition, Jan 1, 2005

12.5

Silberschatz, Galvin and Gagne ©2005

DISK DRIVE STRUCTURE

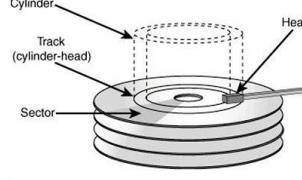
- Data stored on surfaces
 - Up to two surfaces per platter
 - One or more platters per disk
- Data in concentric tracks
 - Tracks broken into sectors
 - 256B-1KB per sector
 - Cylinder: corresponding tracks on all surfaces
- Data read and written by heads
 - Actuator moves heads
 - Heads move in unison



Disk Access Time

Operating System

Disk Structure



- Track
- Sector
- Cylinder
- Spindle
- Read/ Write Head (Arm assembly)
- Seek Time
- Rotational Latency
- RPM
- Transfer Time

Presented by :Parul Vaghanshi

Disk Access Time

Question : Given 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk, please calculate average read time.

Access to a sector involves

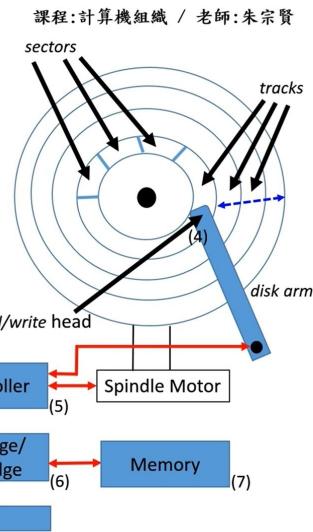
- Controller overhead: receiving orders from CPU and interpreting them
- Seek time: the time it takes for a disk drive to move its head(s) from one track to another
- Rotational latency: the amount of time it takes for the desired sector of a disk
- Data transfer rate: moving data from between disk surface and the host system

Average read time

$$\begin{aligned} & 4\text{ms} \text{ seek time} \\ & + \frac{1}{2} * (60/15,000) = 2\text{ms} \text{ rotational latency} \\ & + 512 / 100\text{MB/s} = 0.005\text{ms} \text{ transfer time} \\ & + 0.2\text{ms} \text{ controller delay} = 6.2\text{ms} \end{aligned}$$

Operating System Design Policy

- File Systems/ Contiguous allocation



Redundant Array of Inexpensive Disk

國立清華大學

RAID Disks

- RAID = Redundant Arrays of Inexpensive Disks
 - provide reliability via redundancy
 - improve performance via parallelism
- RAID is arranged into different levels
 - Striping
 - Mirror (Replication)
 - Error-correcting code (ECC) & Parity bit

Chapter 12: Mass Storage System Operating System Concepts - NTHU USA Lab

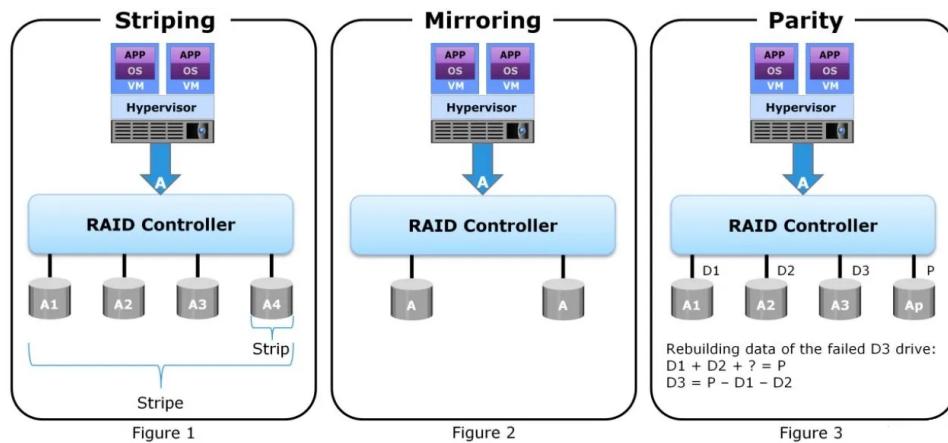


Figure 1

Figure 2

Figure 3

■ RAID 0: non-redundant striping

- Improve **performance via parallelism**

- I/O bandwidth is proportional to the striping count

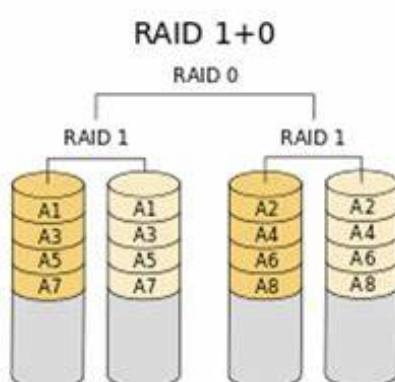
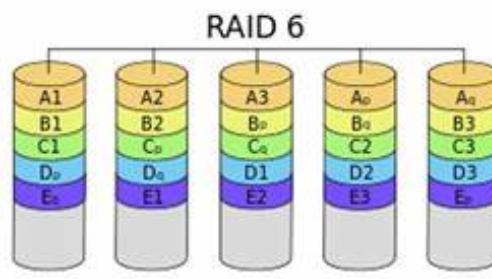
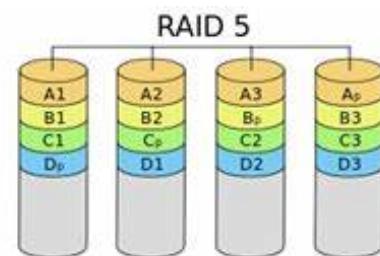
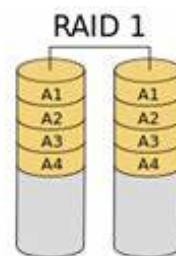
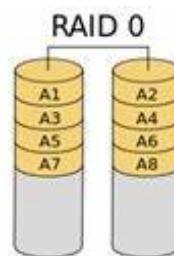
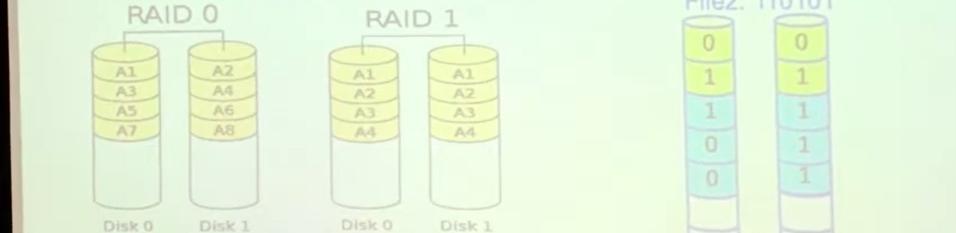
- + Both read and write BW increase by N times (N is the number of disks)

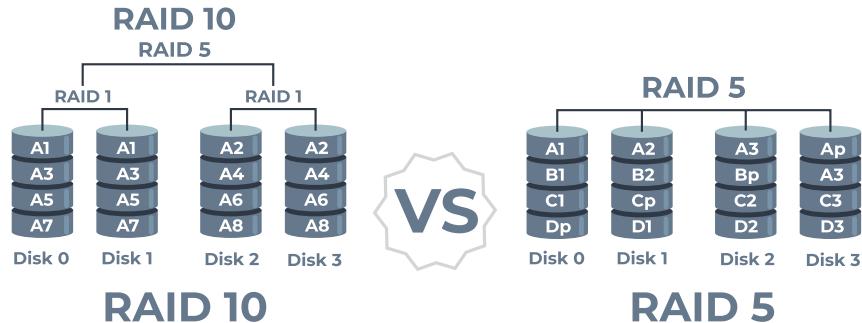
■ RAID 1: Mirrored disks

- Provide **reliability via redundancy**

- + Read BW increases by N times

- + Write BW remains the same

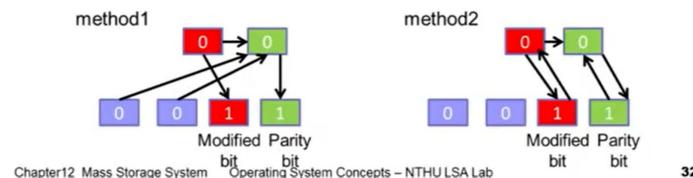




國立清華大學

RAID 5: Distributed Parity

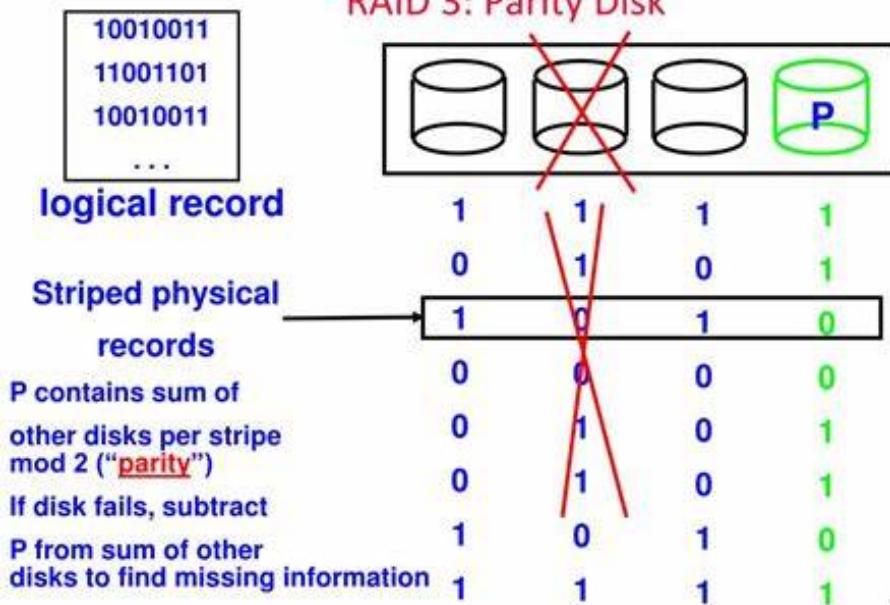
- Read BW increases by N times, because all four disks can serve a read request
- Write BW
 - Method1: (1)read out all unmodified ($N-2$) data bits. (2) re-compute parity bit. (3) write both modified bit and parity bit to disks.
→ write BW = $N / ((N-2)+2) = 1$ → remains the same
 - Method2: (1)only read the parity bit and modified bit. (2) re-compute parity bit by the difference. (3) write both modified bit and parity bit.
→ write BW = $N / (2+2) = N/4$ times faster



32

Redundant Array of Inexpensive Disks

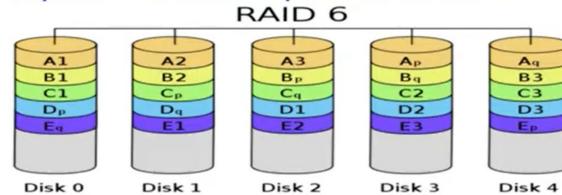
RAID 3: Parity Disk



12.

RAID 6: P+Q Dual Parity Redundancy

- Like RAID 5, but stores extra redundant information to guard against **multiple disk failure**
- Use **ECE code** (i.e. Error Correction Code) instead of single parity bit
- Parity bits are also striped across disks

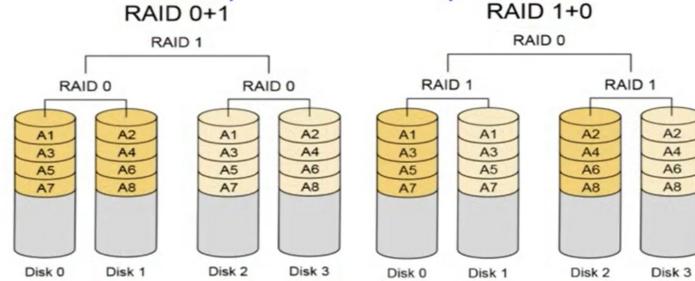


Chapter12 Mass Storage System Operating System Concepts – NTHU LSA Lab

33

Hybrid RAID

- RAID 0+1: Stripe then replicate
- RAID 1+0: Replicate then stripe



*First level often controlled by a controller. Therefore, RAID 10 has better fault tolerance than RAID 01 when multiple disk fails

<http://www.thegeekstuff.com/2011/10/raid10-vs-raid01/>

Chapter12 Mass Storage System Operating System Concepts – NTHU LSA Lab

34

-- Memo End --