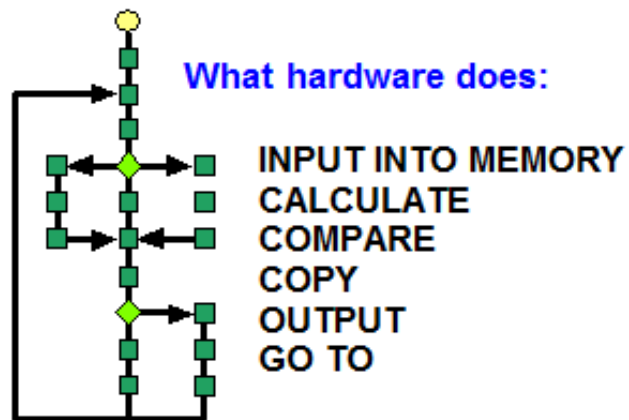
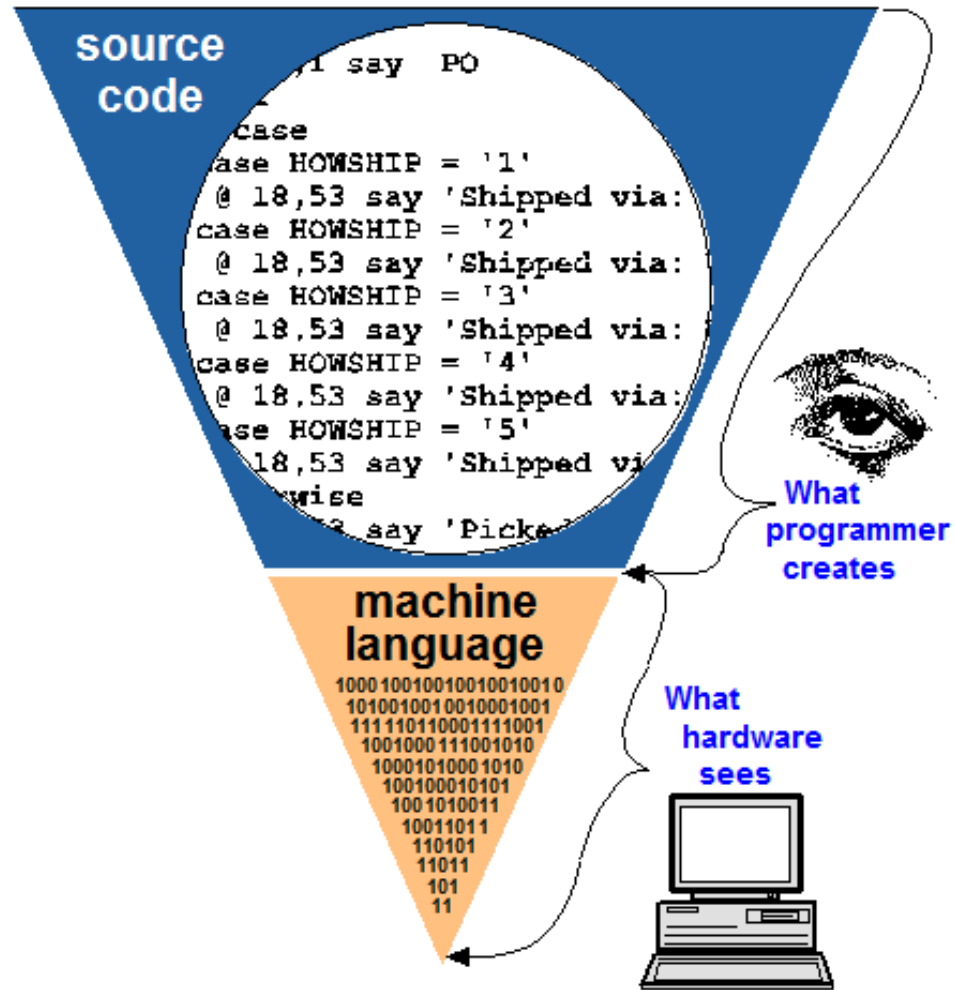
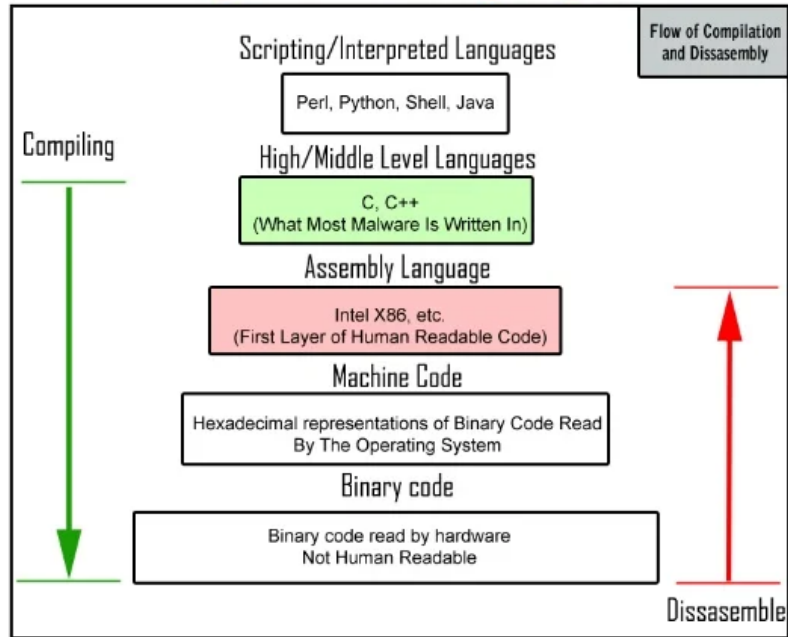


Computer Organization and Architecture

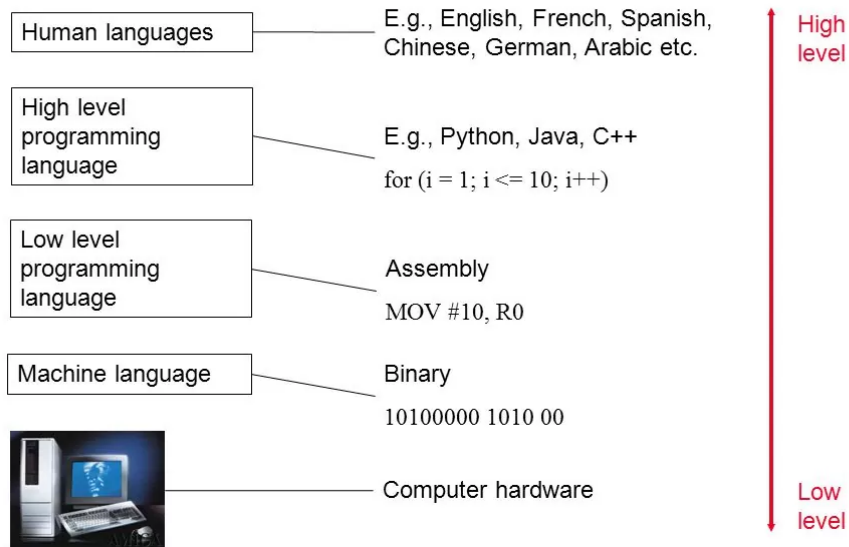
SOURCE CODE TO MACHINE LANGUAGE



High Level Languages

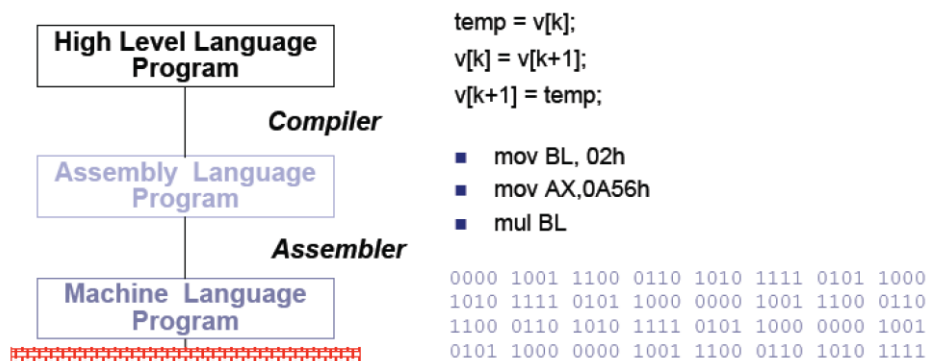


High Vs. Low Level Languages



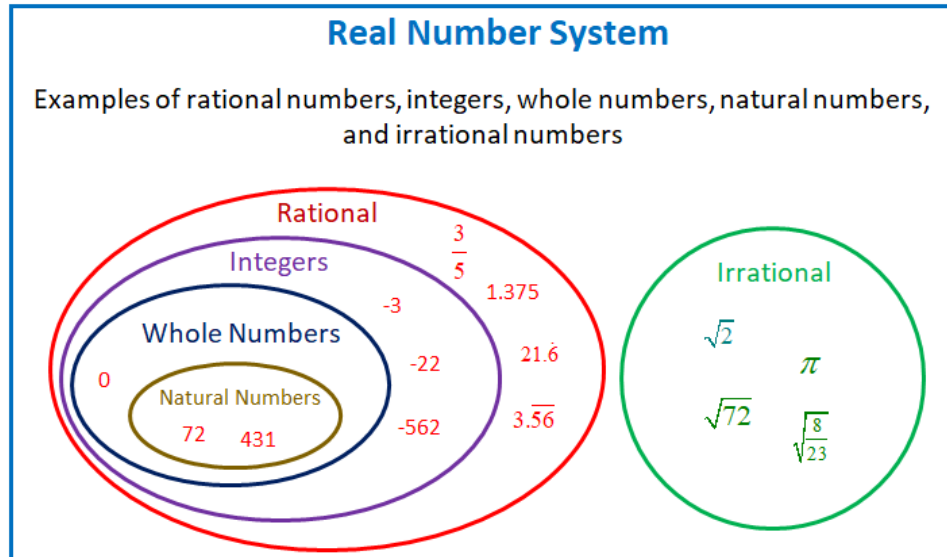
James Tam

Levels of Representation



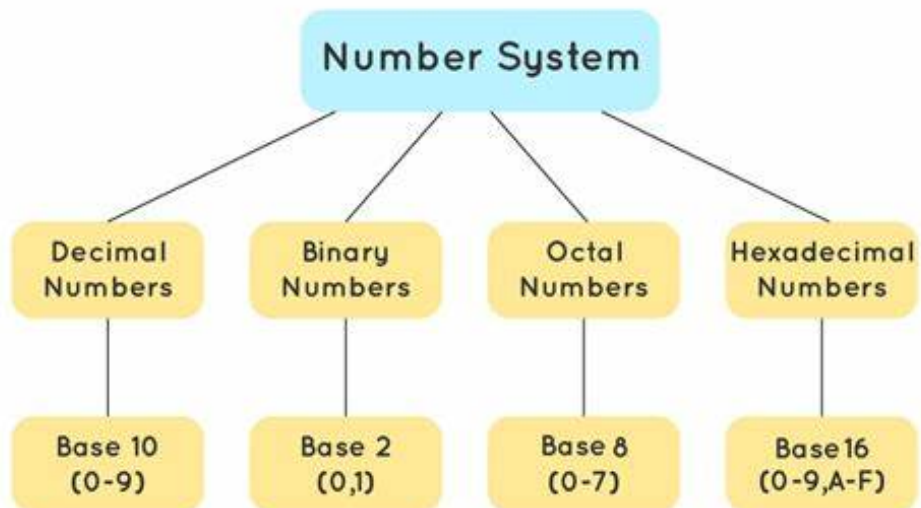
Number System

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

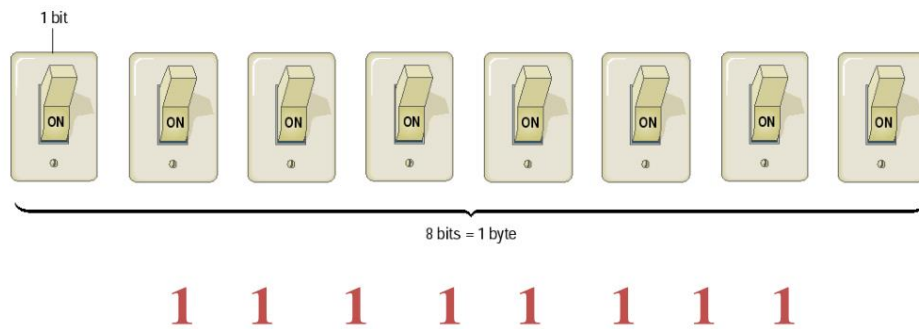
- Convert from decimal to binary
Divide the decimal by the largest binary weight it is divisible by and place a "1" in that column. Then select the next largest weight, if it is divisible put a "1" in that column otherwise place a "0" in the column. Continue until all the columns have either a "1" or "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

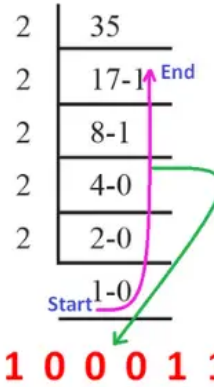
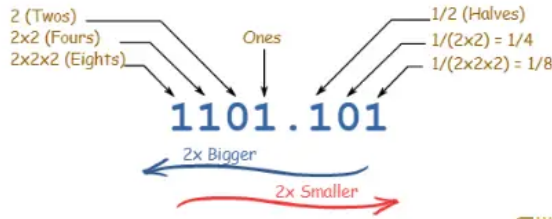
Number System

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?



1 0 0 0 1 1



Electrical 4 U

Complement

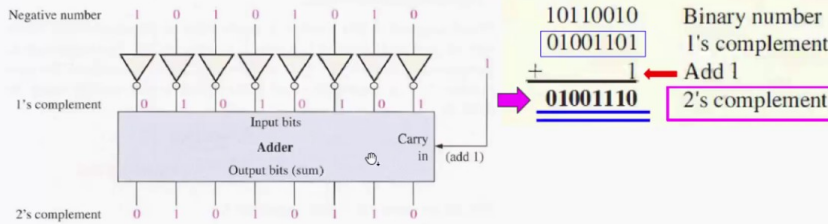
2.5) 1'S and 2'S Complements of Binary Numbers

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.

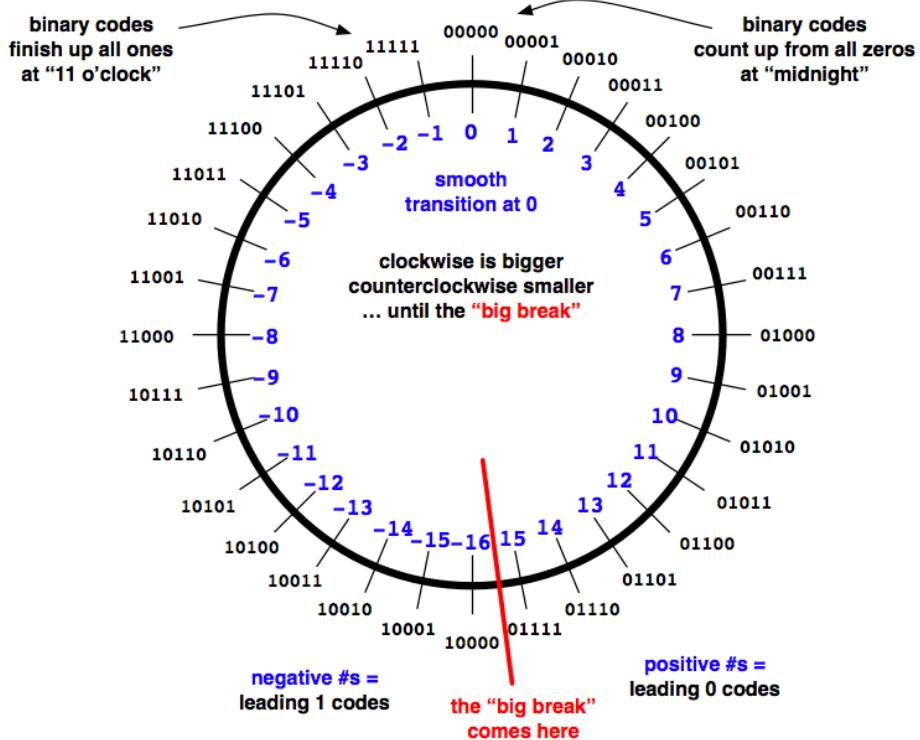
$$\text{2's complement} = (\text{1's complement}) + 1$$

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 $(10^n - 1) - N$

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

- $99999 \dots - N$

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 base 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 manufactures 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}
 - ↑ *decimal point*
- ❖ Examples of floating-point numbers in base 2 ...
 - ✧ 1.00101×2^{23} , 0.0100101×2^{25} , -1.101101×2^{-3} , -1101.101×2^{-6}
 - ↑ *binary point*
 - ✧ 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)



- ❖ **Significand** is equal to $(1.F)_2 = (1.f_1f_2f_3f_4\dots)_2$
 - ❖ 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

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

$$(-1)^S \times (1.f_1f_2f_3f_4\dots)_2 \times 2^{\text{val}(E)}$$

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

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

Floating Point

ICS 233 - KFUPM

© Muhamed Mudawar slide 8

Floating-Point Representation

- ❖ A floating-point number is represented by the triple
 - ❖ S is the **Sign bit** (0 is positive and 1 is negative)
 - Representation is called **sign and magnitude**
 - ❖ E is the **Exponent field** (signed)
 - Very large numbers have large positive exponents
 - Very small close-to-zero numbers have negative exponents
 - More bits in exponent field increases **range of values**
 - ❖ F is the **Fraction field** (fraction after binary point)
 - More bits in fraction field improves the **precision** of FP numbers



$$\text{Value of a floating-point number} = (-1)^S \times \text{val}(F) \times 2^{\text{val}(E)}$$

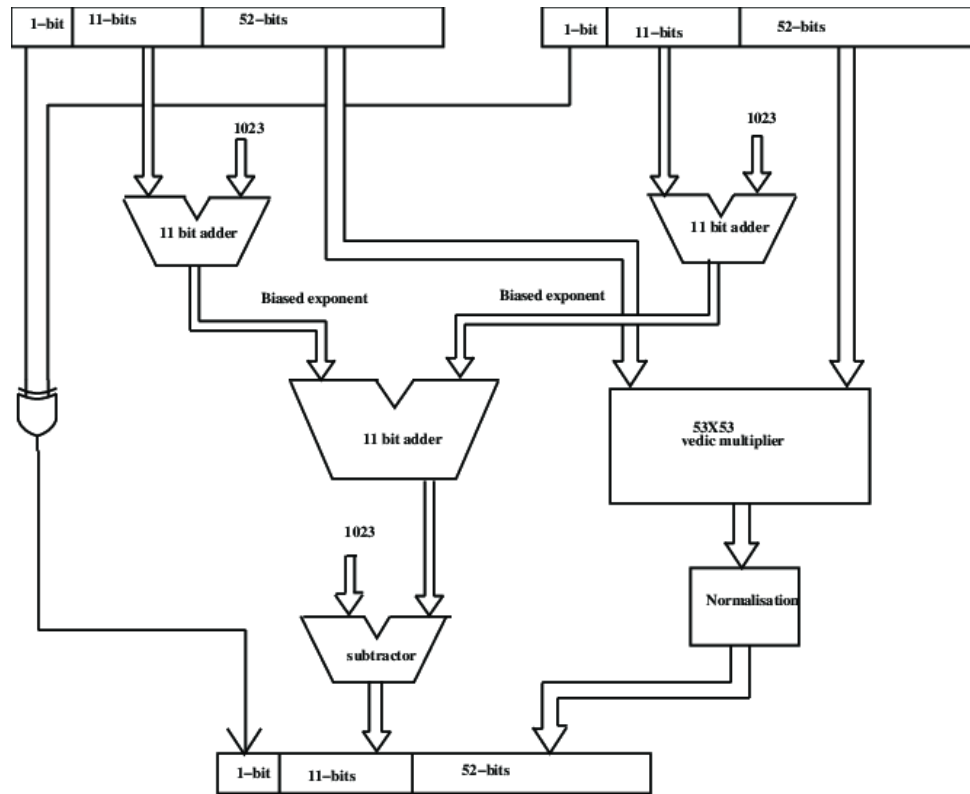
Biased Exponent - Cont'd

- ❖ For **double precision**, exponent field is **11 bits**
 - ❖ E can be in the range 0 to 2047
 - ❖ $E = 0$ and $E = 2047$ are **reserved for special use**
 - ❖ $E = 1$ to 2046 are used for **normalized** floating point numbers
 - ❖ Bias = 1023 (half of 2046), $\text{val}(E) = E - 1023$
 - ❖ $\text{val}(E=1) = -1022$, $\text{val}(E=1023) = 0$, $\text{val}(E=2046) = 1023$
- ❖ Value of a Normalized Floating Point Number is

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

$$(-1)^S \times (1.f_1f_2f_3f_4\dots)_2 \times 2^{E - \text{Bias}}$$

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



ASCII

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

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

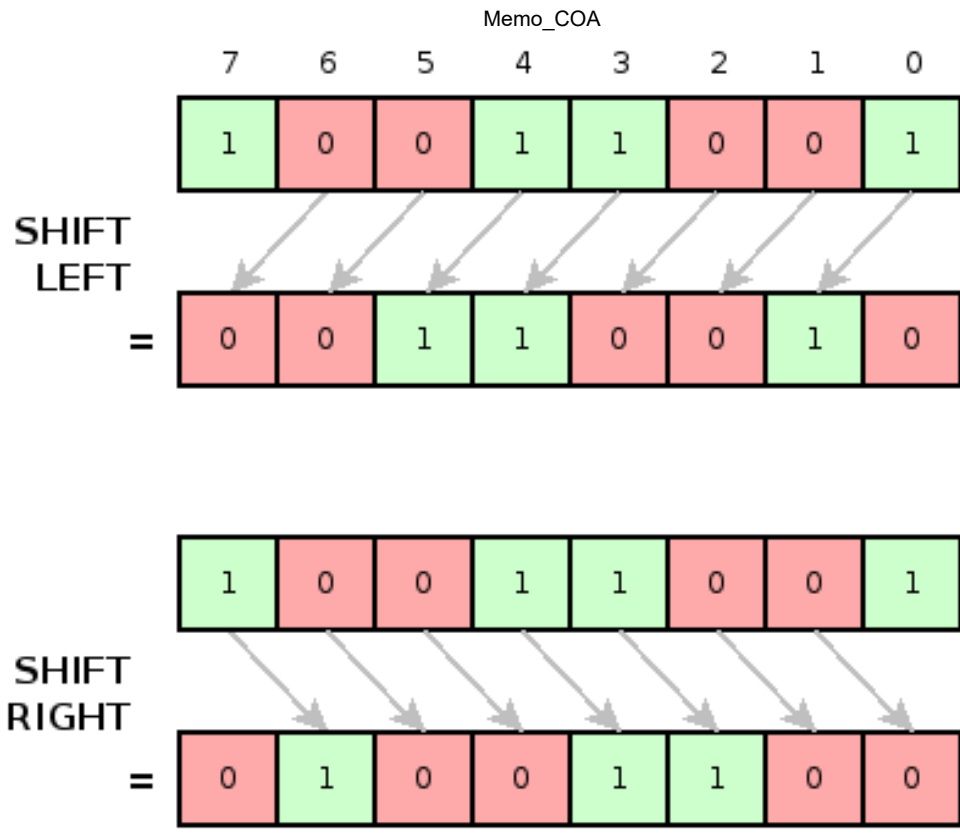
01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

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

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



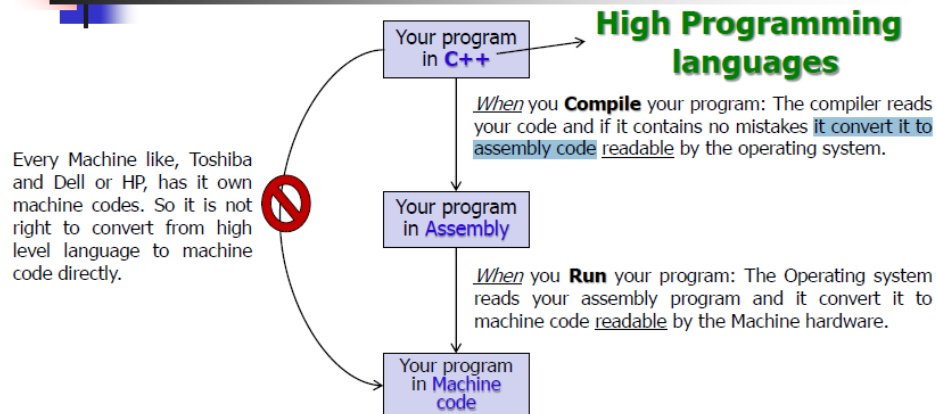
- As stated earlier, a program written in any programming language is a set of logically related instructions. These instructions have two parts, as shown in the following figure:
- The two parts of a programming language instruction are:
 - *Operation code (opcode)*: This part instructs a computer about 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.

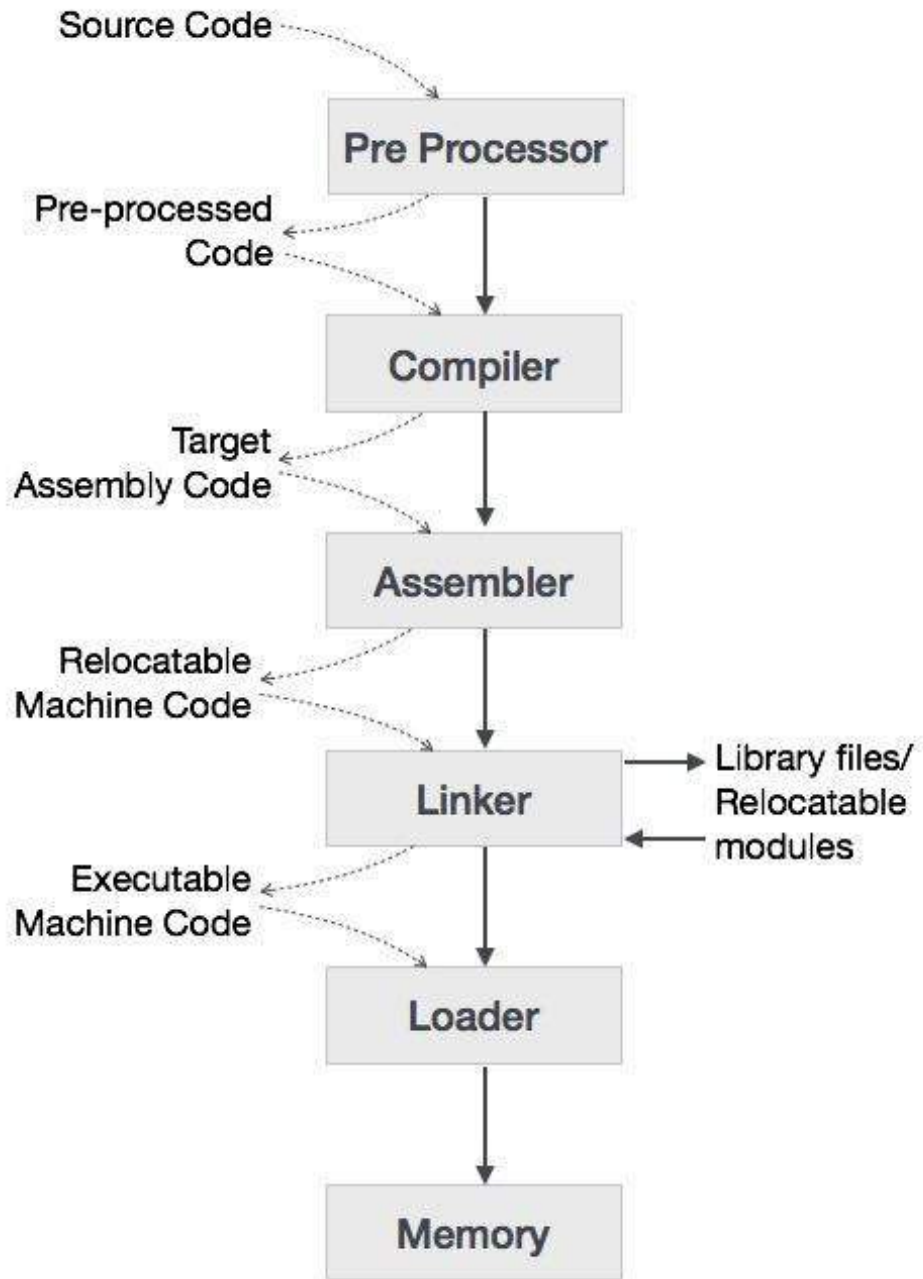


- For example, in the instruction Add A and B, Add is the opcode and A and B are operands

Compiler and Assembler

Compiling and running your program



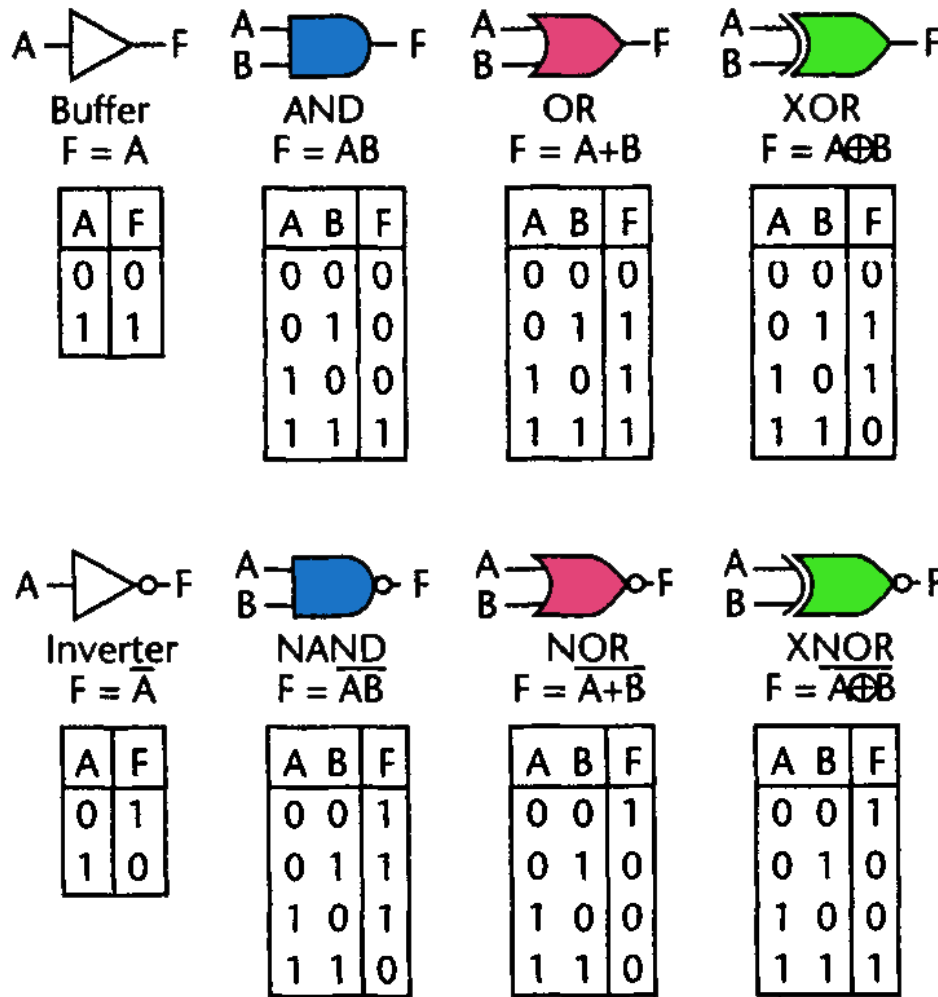


COMPILER VS LINKER VS LOADER

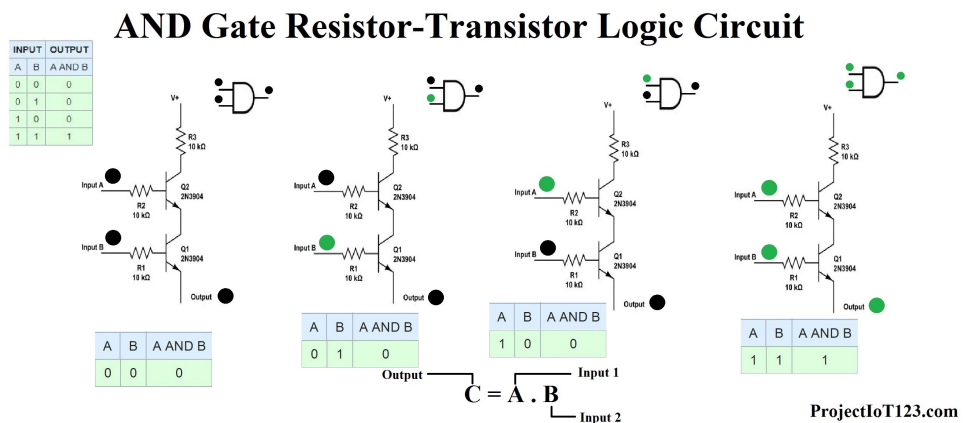
COMPILER	LINKER	LOADER
A software that transforms computer code written in one programming language into another programming language	A computer utility program that takes one or more object files generated by a compiler and combines them into a single executable file	A part of an operating system that 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
Visit www.pediaa.com		

Logic Design

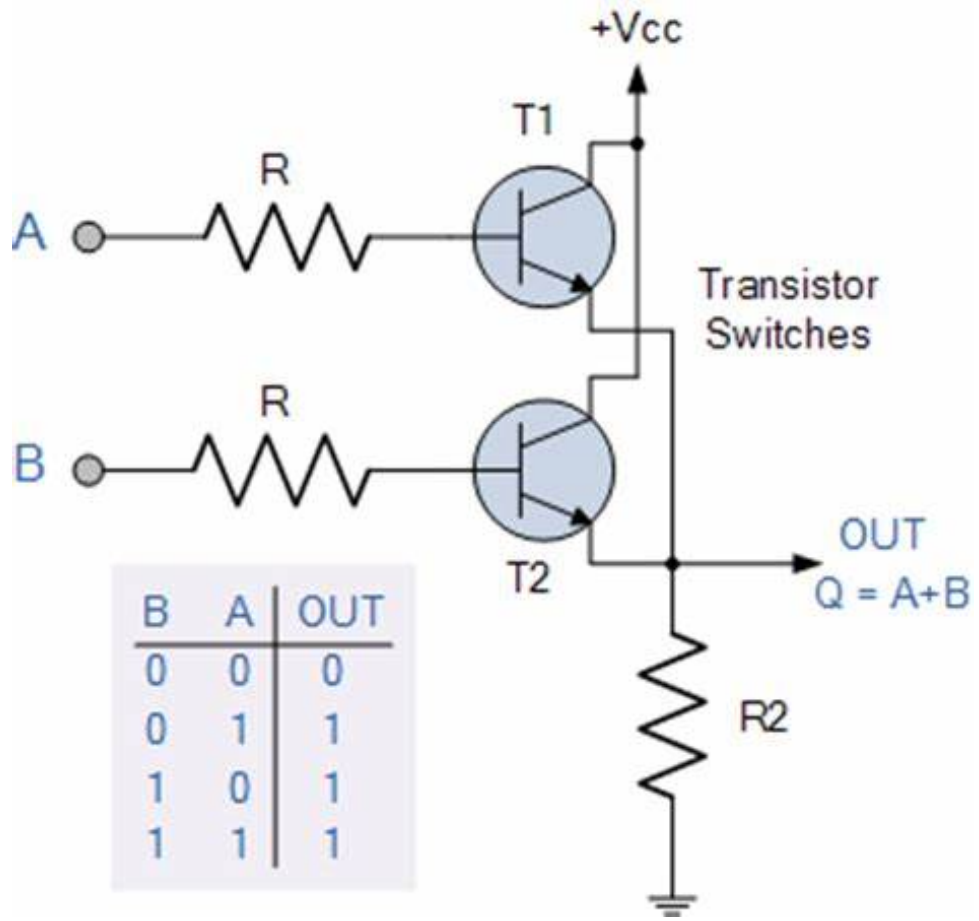
- Logic Gates



- AND Gate



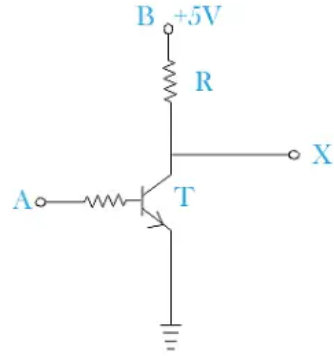
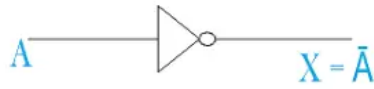
- OR Gate



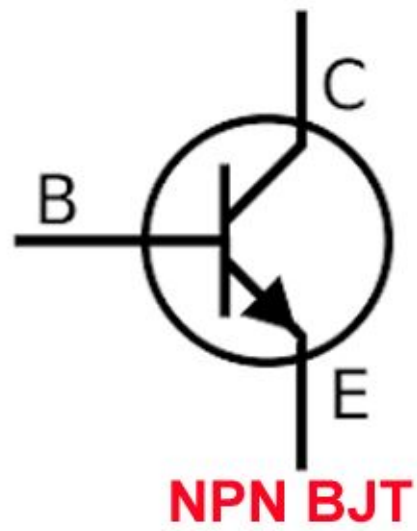
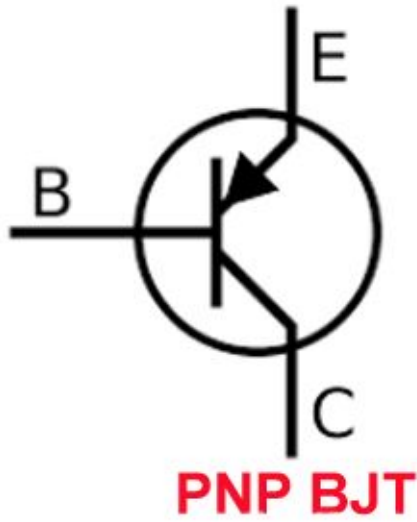
- NOT Gate

What is a NOT Gate?

A	X = \bar{A}
0	1
1	0



 **Electrical 4 U**



- XOR Gate

XOR GATE Truth Table



INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

BOOLEAN EXPRESSION

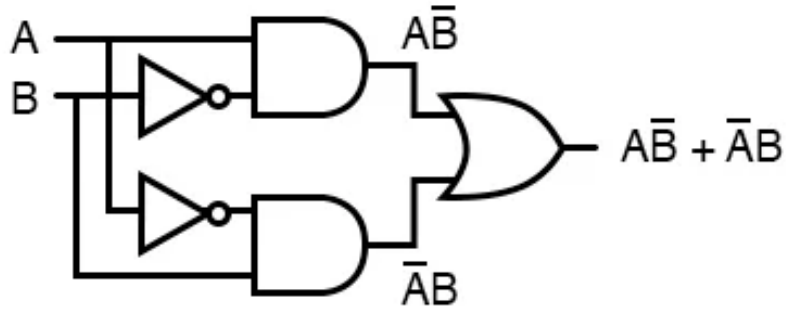
$$\begin{aligned}
 &A \cdot \bar{B} + \bar{A} \cdot B \\
 &(A + B) \cdot (\bar{A} + \bar{B})
 \end{aligned}
 \left. \vphantom{\begin{aligned} &A \cdot \bar{B} + \bar{A} \cdot B \\ &(A + B) \cdot (\bar{A} + \bar{B}) \end{aligned}} \right\} \begin{array}{l} \text{C} = A \oplus B \\ \text{Output} \end{array}$$

— Input1
— Input2

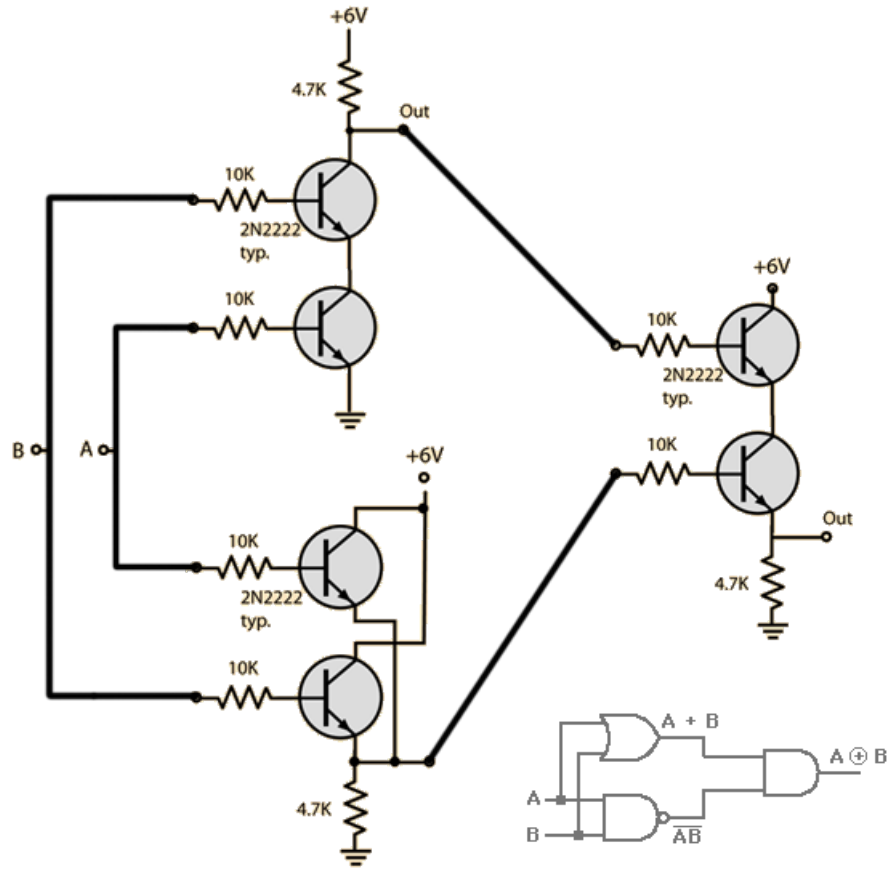
ProjectIoT123.com



... is equivalent to ..



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



XOR GATE

Which Is Equivalent To

$\bar{A} \cdot B + A \cdot \bar{B}$
 [Traditional Symbol]

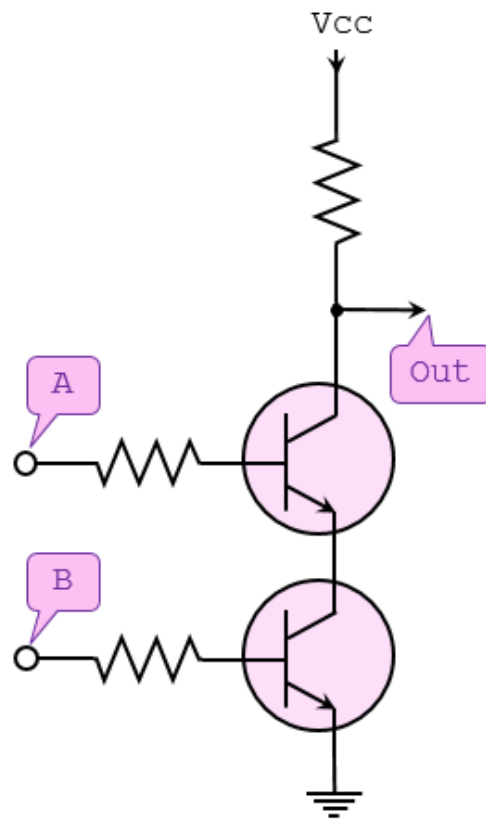
$\bar{A} \cdot B + A \cdot \bar{B}$
 [IEEE Symbol]

Truth Table For XOR

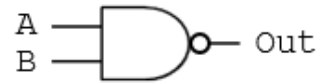
IN		OUT
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

• NAND Gate

NAND Gate



Symbol:



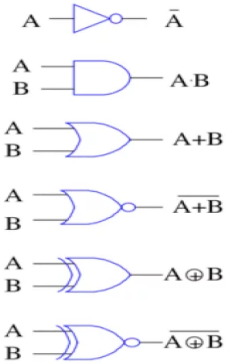
Truth table: NAND Gate

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

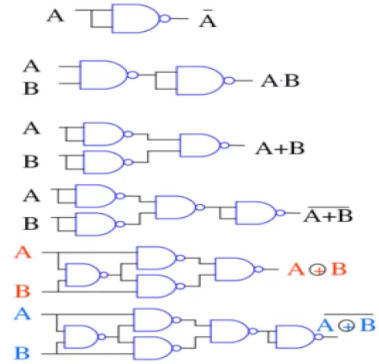


The NAND gate as a universal gate

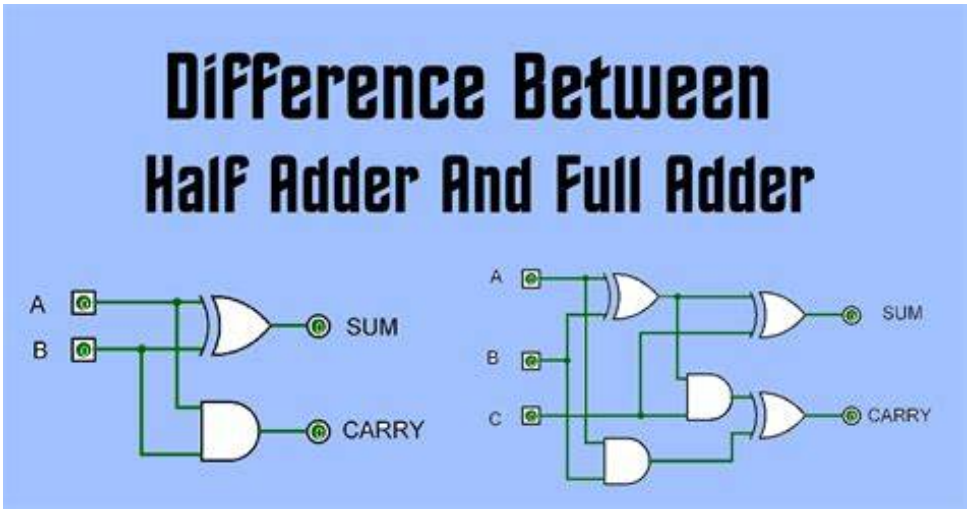
Logic function



NAND gate only



- Half-Adder vs Full-Adder



• **Half-Adder**

Half Adder and Full Adder

Logic Diagram:

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

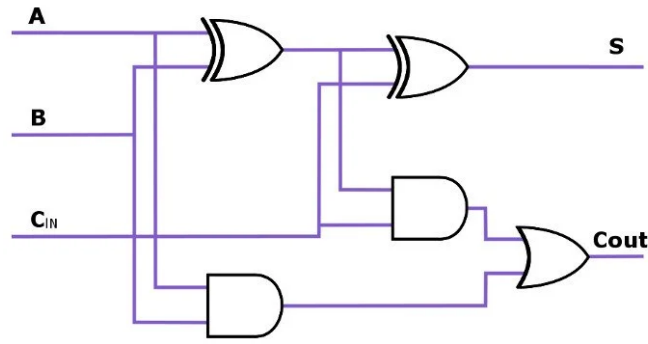
$C = A \cdot B$

a

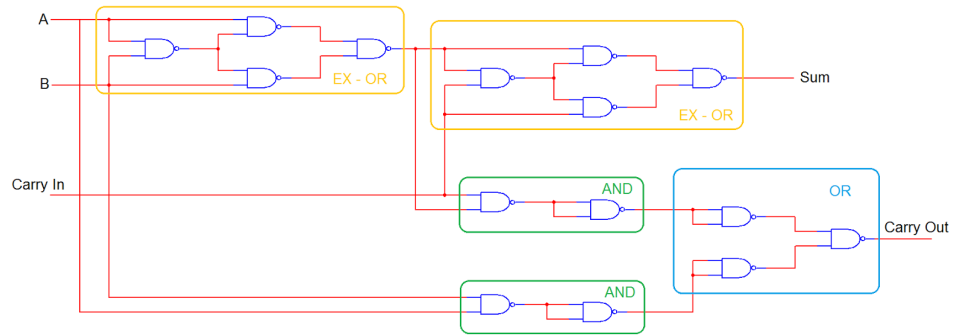
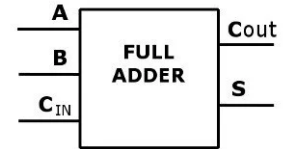
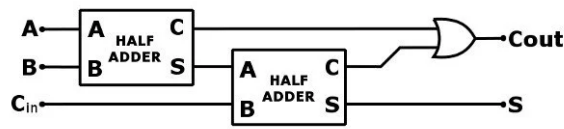
b

Figure-1 : a)Half Adder b)XOR implementation using NAND gates

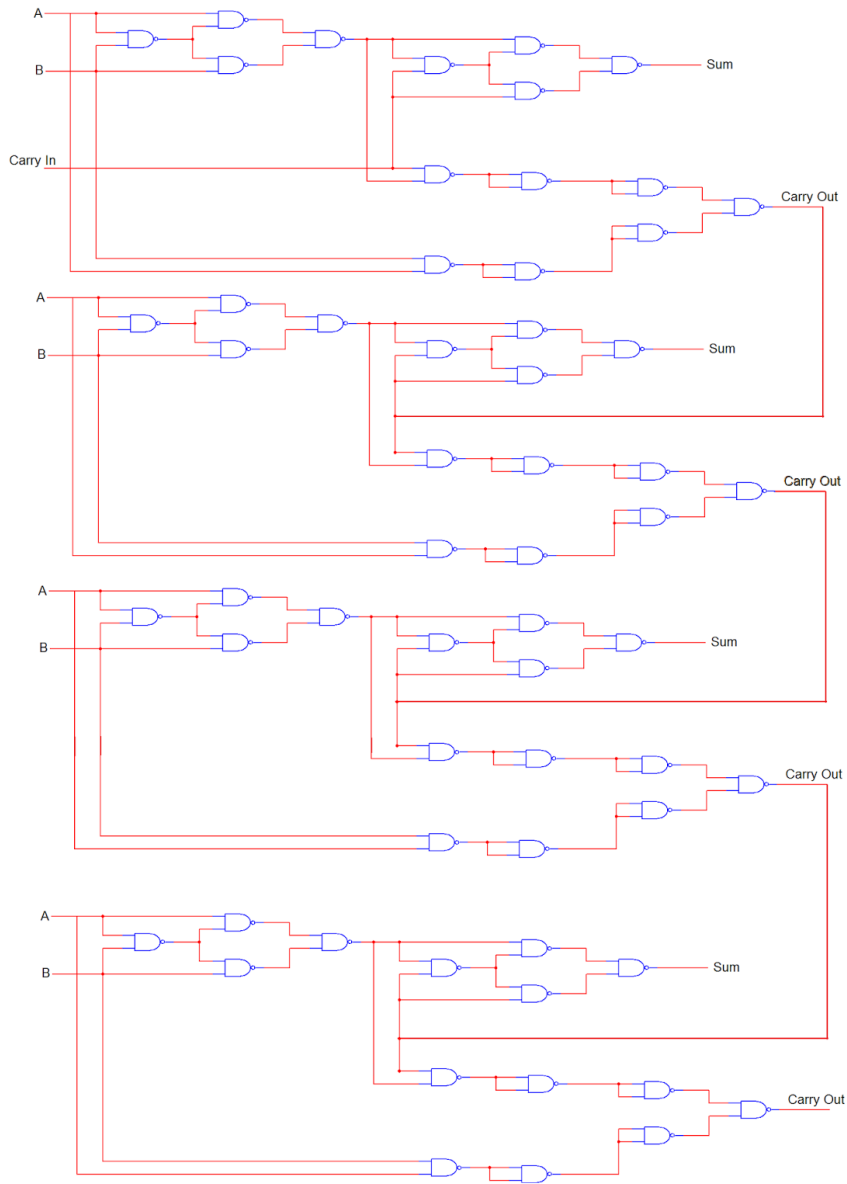
• **Full Adder**



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**

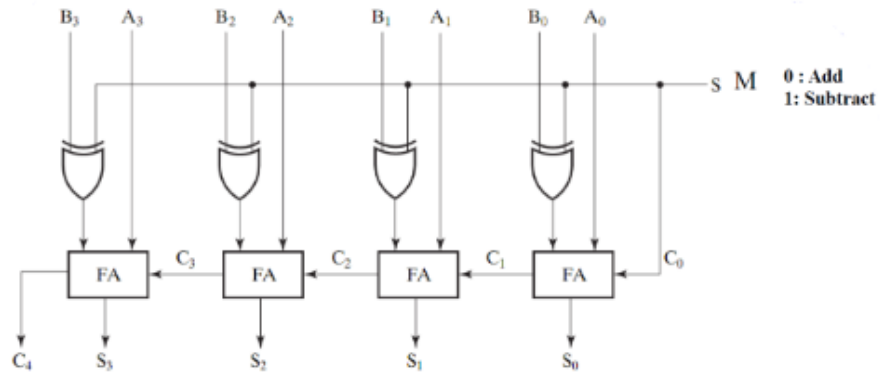
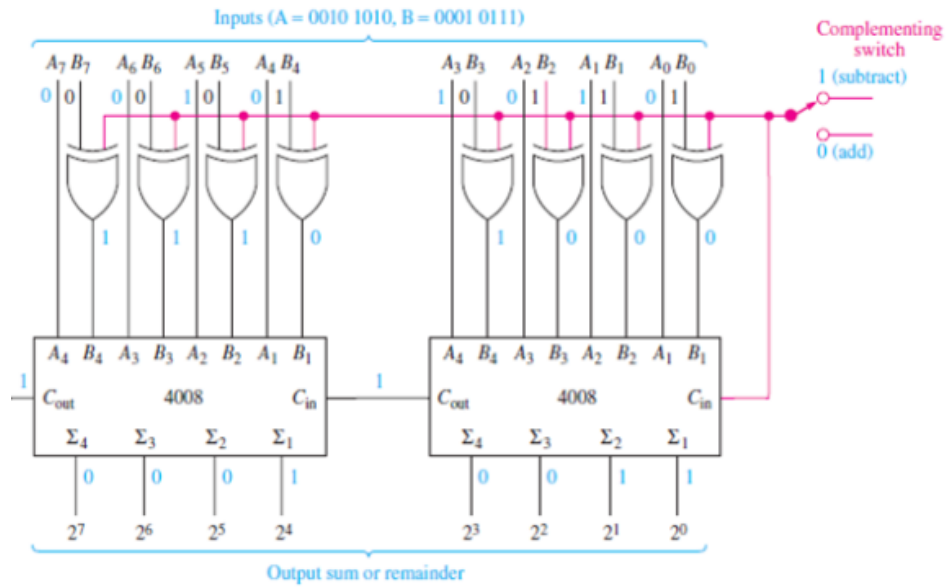


Fig. 1 Modular design of 4-bit adder/subtractor

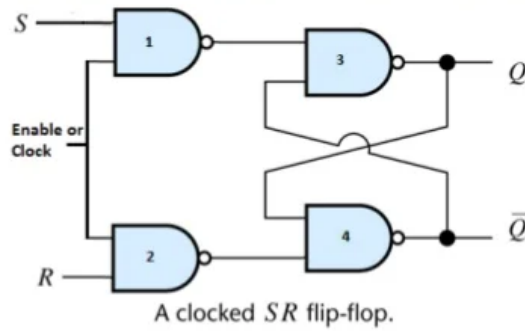


Circuits as Memory

- SR Flip Flop



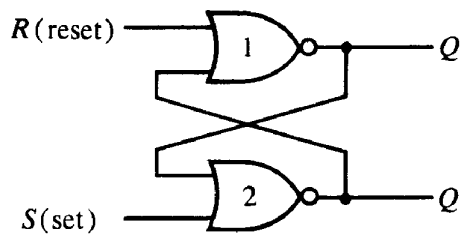
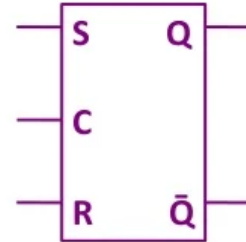
Gated Latch-Clocked RS Flip-flop



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

Truth table

Logical Symbol

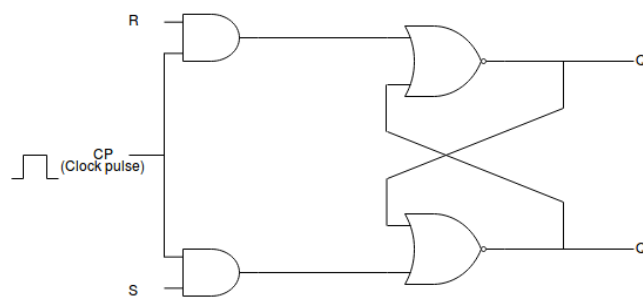


S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

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

(a) Logic diagram

(b) Truth table



a) Logic diagram

fig: Clocked SR flip flop

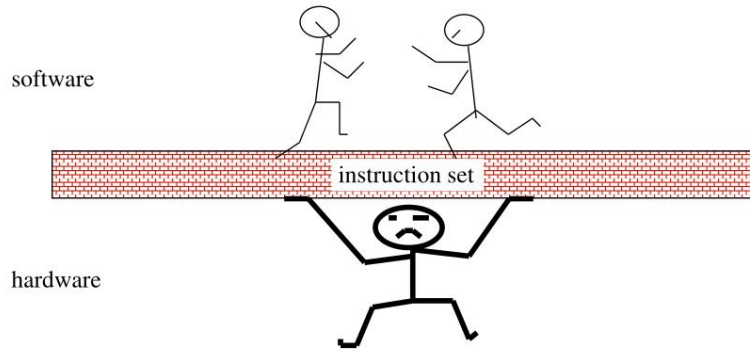
Q	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Intermediate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Intermediate

b) Truth table

Computer Organization and Architecture

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

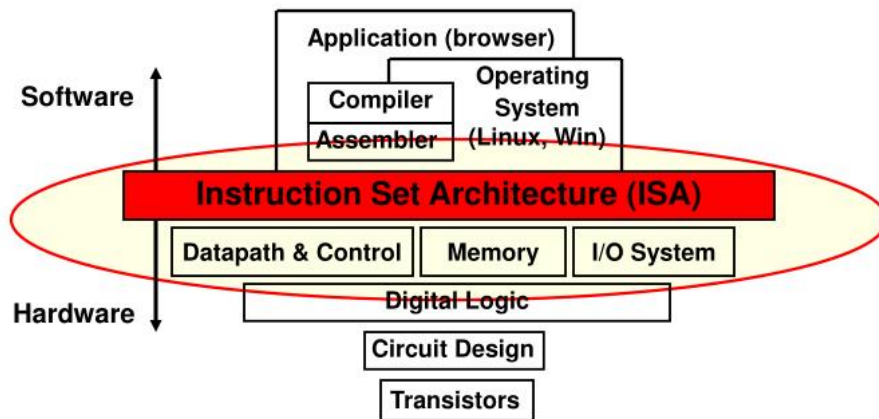
Instruction Set Architecture: Critical Interface



- ▣ 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



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).

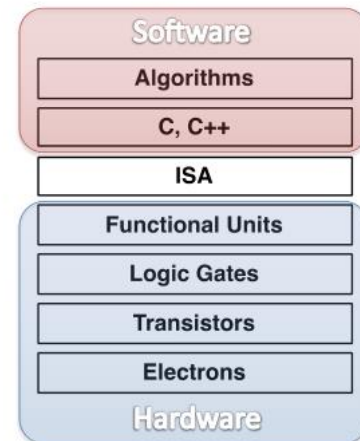
BYU CS 224

ISA

5

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

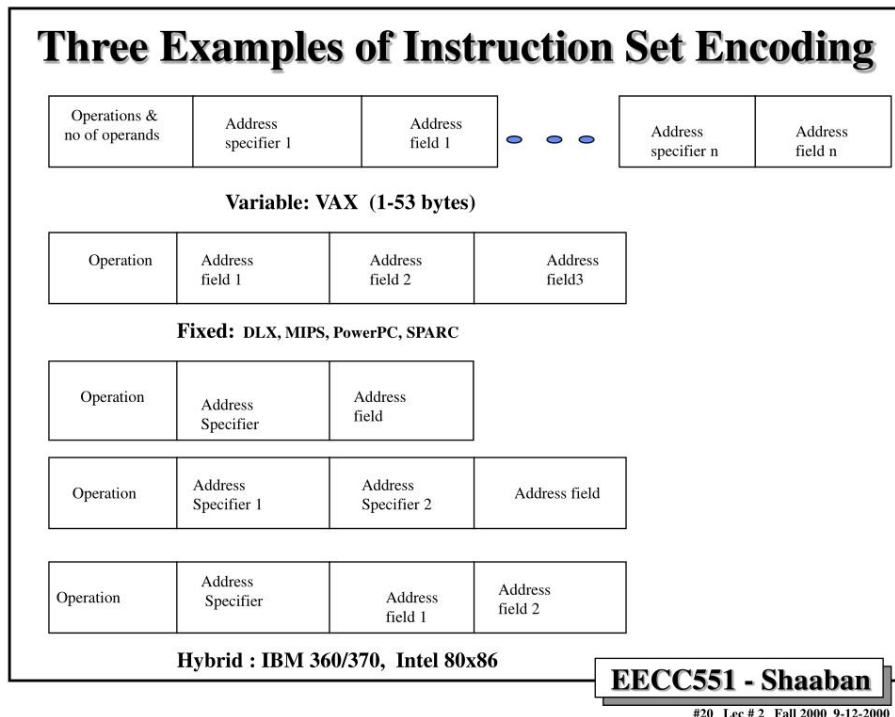


Mike Holenderski, m.holenderski@tue.nl

12



TU/e Technische Universiteit
Eindhoven
University of Technology



CICS vs RICS

CISC vs. RISC

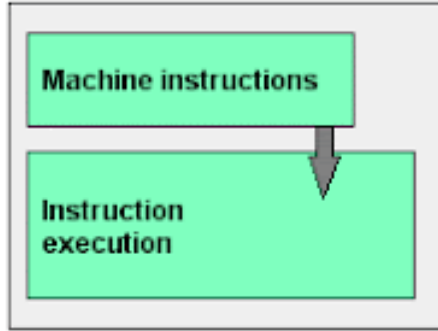
Complex Instruction Set Computer

- 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

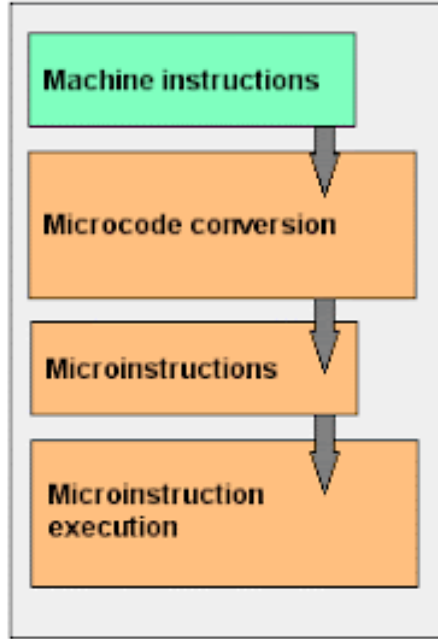
Reduced Instruction Set Computers

- 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

RISC



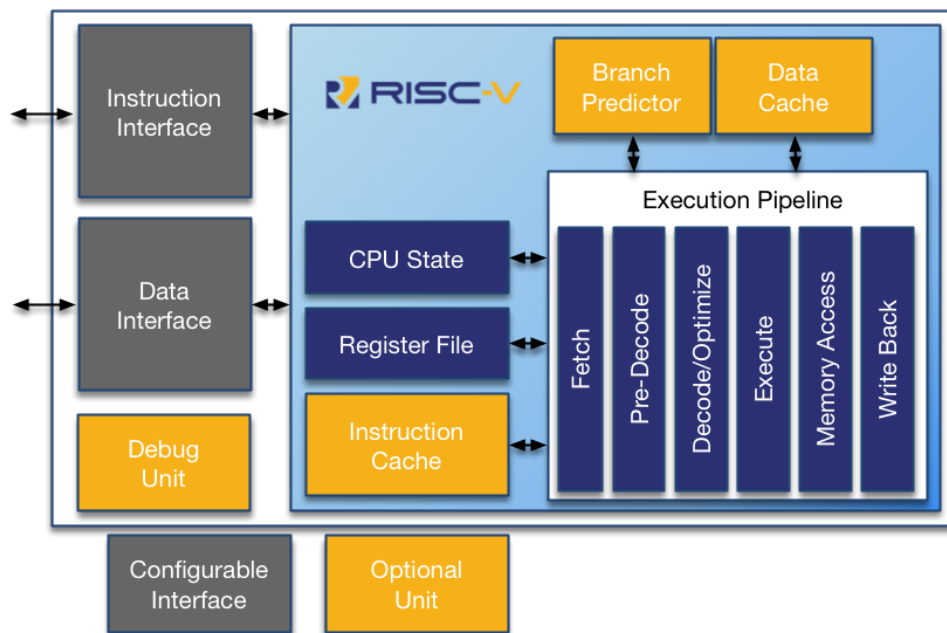
CISC



32-bit RISC-V Instruction Formats

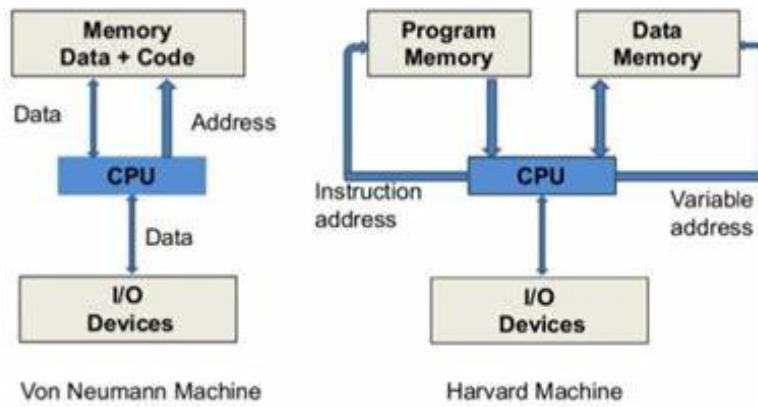
Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd			opcode								
Immediate	imm[11:0]											rs1					funct3			rd			opcode									
Upper Immediate	imm[31:12]											rd			opcode																	
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 *instruction formats*
- **funct7 + funct3 (10 bit):** combined with **opcode**, 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



6

Memory & Timing slides

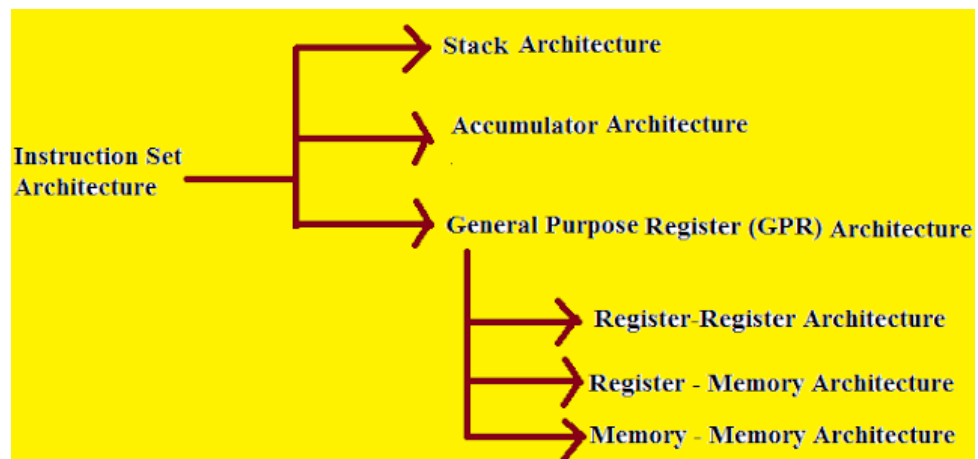
isa@uak.ac.id/courses/171917/lectures/section_50759/Multiprogramming%202.pdf

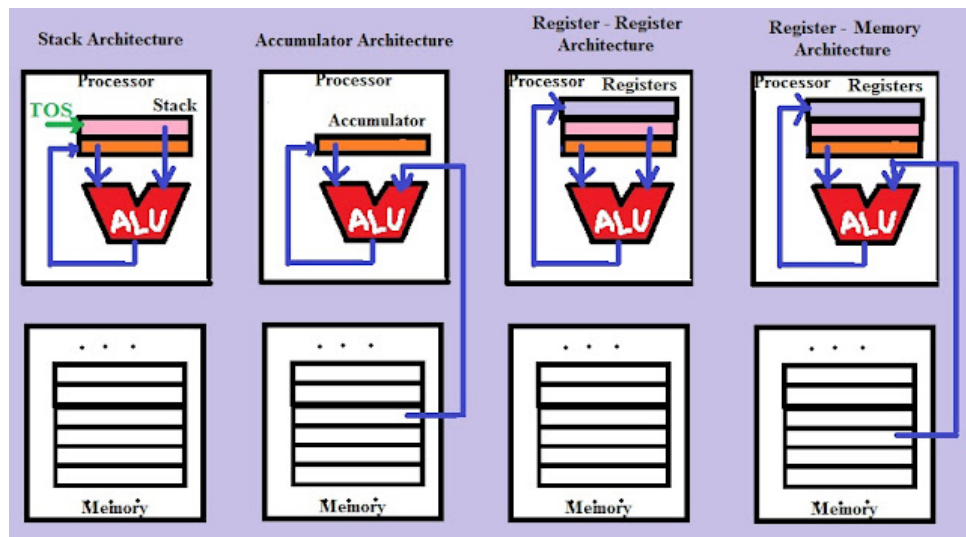
ISA: STORAGE RESOURCES

- **"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**

The diagram shows four storage resources: Instruction memory (2¹⁵ x 16), Data memory (2¹⁵ x 16), Program counter (PC), and Register file (8 x 16).

Classification of ISA (or) Types of ISA



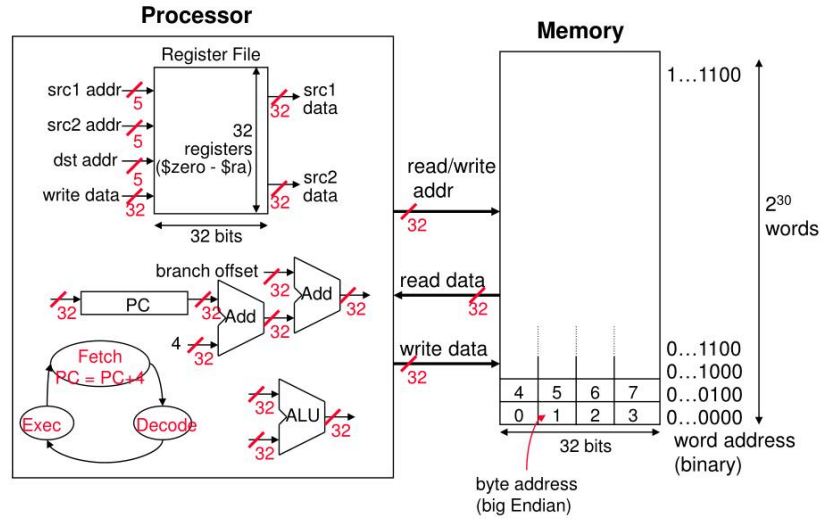


ISA - MIPS Instruction Format and Addressing Model

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 So Far



34

The MIPS ISA – Register File

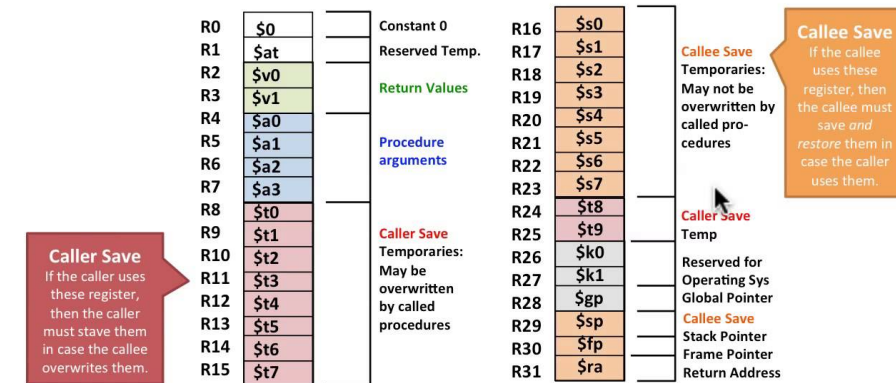
- When writing assembly, these registers can be referenced by their address (number) or name
- General purpose and special purpose registers

#	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	Function parameter	\$19	\$s3	
\$4	\$a0				
\$5	\$a1				
\$6	\$a2				
\$7	\$a3	Temporary – Caller-saved	\$20	\$s4	
\$8	\$t0		\$21	\$s5	
\$9	\$t1		\$22	\$s6	
\$10	\$t2		\$23	\$s7	
\$11	\$t3		\$24	\$t8	Temporary – Caller-saved
\$12	\$t4		Reserved for OS	\$25	\$t9
\$13	\$t5			\$26	\$k0
\$14	\$t6	\$27		\$k1	
\$15	\$t7	\$28	\$gp	Global pointer	
		\$29	\$sp	Stack pointer	
		\$30	\$fp	Frame pointer	
		\$31	\$ra	Function return address	

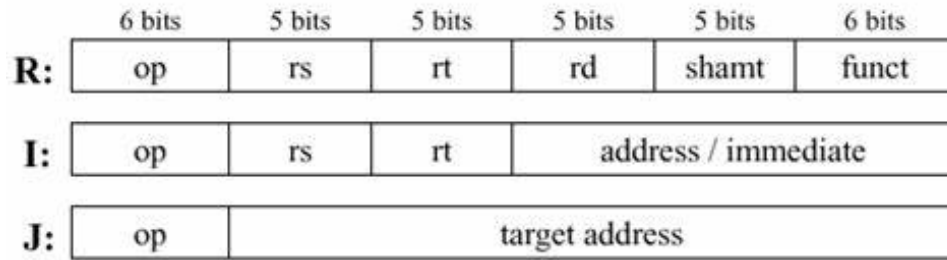
17

Who saves what?

53



Instruction Format



op: basic operation of the instruction (opcode)

rs: first source operand register

rt: second source operand register

rd: destination operand register

shamt: shift amount

funct: selects the specific variant of the opcode (function code)

address: offset for load/store instructions ($\pm 2^{15}$)

immediate: constants for immediate instructions

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)

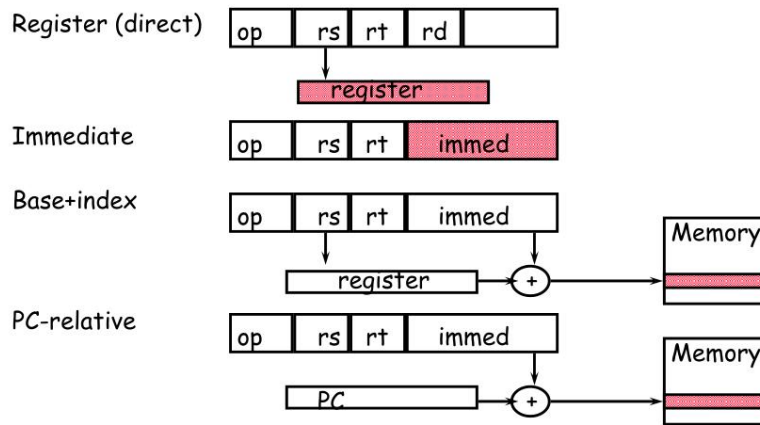
✧ Used by jump instructions



Addressing Model

Example: MIPS Instruction Formats and Addressing Modes

- All instructions 32 bits wide



ECE 361

3-45

5.6 Addressing Modes

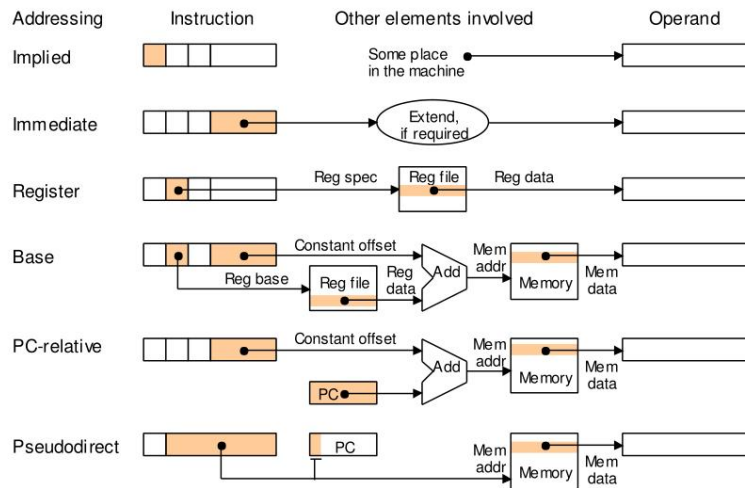


Figure 5.11 Schematic representation of addressing modes in MiniMIPS.

Computer Architecture, Instruction-Set Architecture

Slide 22

Instruction Categories

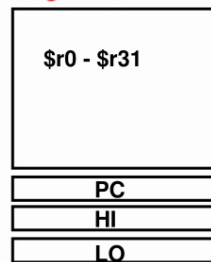
MIPS: ISA					
Category	Instruction	Op Code	Example	Meaning	
Arithmetic	Add	0 and 32	add \$s1, \$s2, \$s3	$\$s1 = \$s2 + \$s3$	
(R & I format)	Subtract	0 and 34	sub \$s1, \$s2, \$s3	$\$s1 = \$s2 - \$s3$	
	add immediate	8	addi \$s1, \$s2, 6	$\$s1 = \$s2 + 6$	
	or immediate	13	ori \$s1, \$s2, 6	$\$s1 = \$s2 \vee 6$	
Logical (R & I format)	And	0 and 36	and \$s1, \$s2, \$s3	$\$s1 = \$s2 \& \$s3$	
	Or	0 and 37	or \$s1, \$s2, \$s3	$\$s1 = \$s2 \$s3$	
	Nor	0 and 39	nor \$s1, \$s2, \$s3	$\$s1 = \sim(\$s2 \$s3)$	
	And immediate	12	andi \$s1, \$s2, 100	$\$s1 = \$s2 \& 100$	
	Or immediate	13	ori \$s1, \$s2, 100	$\$s1 = \$s2 100$	
	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	$\$s1 = 6 * 2^{16}$	
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	40
	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

Registers



3 Instruction Formats: all 32 bits wide

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

ISA - Performance

Compiler Variations, MIPS, Performance: An Example (Continued)

$$\text{MIPS} = \text{Clock rate} / (\text{CPI} \times 10^6) = 100 \text{ MHz} / (\text{CPI} \times 10^6)$$

$$\text{CPI} = \text{CPU execution cycles} / \text{Instructions count}$$

$$\text{CPU clock cycles} = \sum_{i=1}^n (\text{CPI}_i \times C_i)$$

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} / \text{Clock rate}$$

- **For compiler 1:**

- $\text{CPI}_1 = (5 \times 1 + 1 \times 2 + 1 \times 3) / (5 + 1 + 1) = 10 / 7 = 1.43$

- $\text{MIP}_1 = 100 / (1.428 \times 10^6) = 70.0$

- $\text{CPU time}_1 = ((5 + 1 + 1) \times 10^6 \times 1.43) / (100 \times 10^6) = 0.10 \text{ seconds}$

- **For compiler 2:**

- $\text{CPI}_2 = (10 \times 1 + 1 \times 2 + 1 \times 3) / (10 + 1 + 1) = 15 / 12 = 1.25$

- $\text{MIP}_2 = 100 / (1.25 \times 10^6) = 80.0$

- $\text{CPU time}_2 = ((10 + 1 + 1) \times 10^6 \times 1.25) / (100 \times 10^6) = 0.15 \text{ seconds}$

EECC551 - Shaaban

#47 Lec #1 Winter 2003 12-1-2003

Speed Up Equation for Pipelining

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

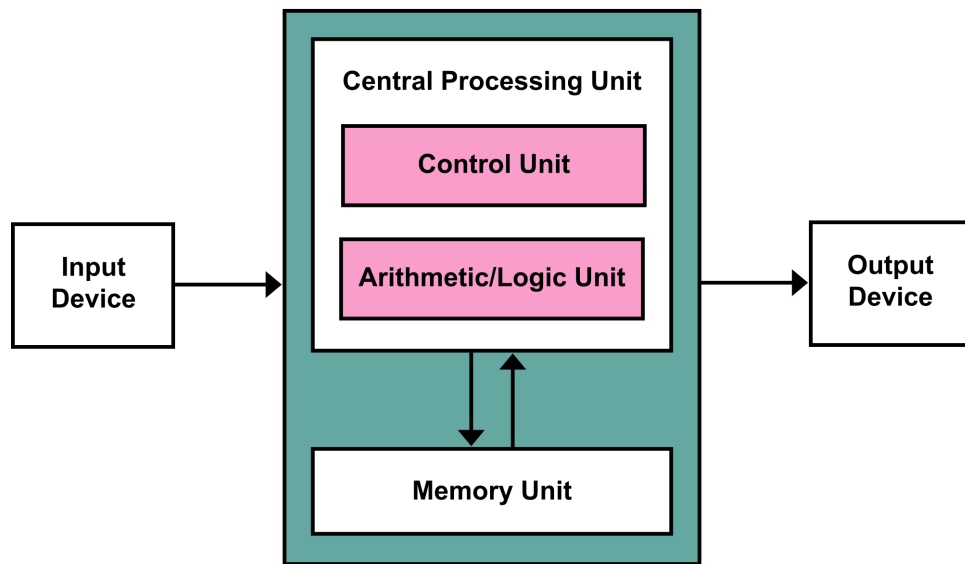
$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

For simple RISC pipeline, $\text{CPI} = 1$:

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

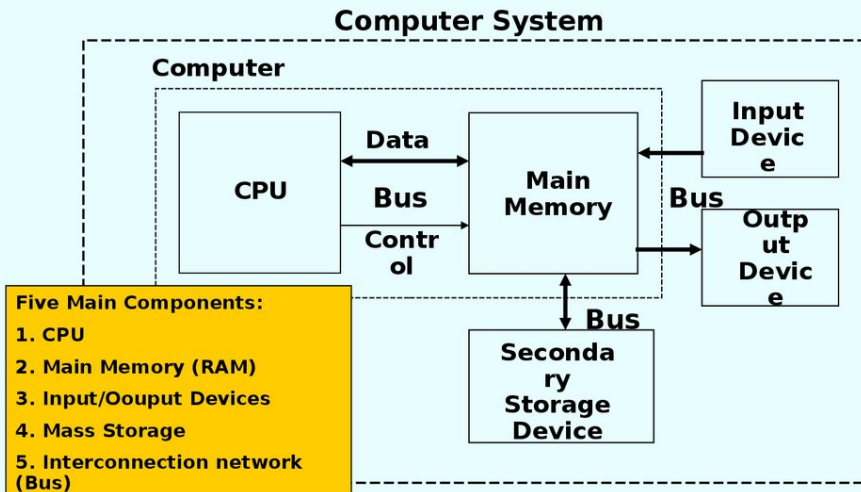
CS211 41

Processor - Von Neumann Architecture

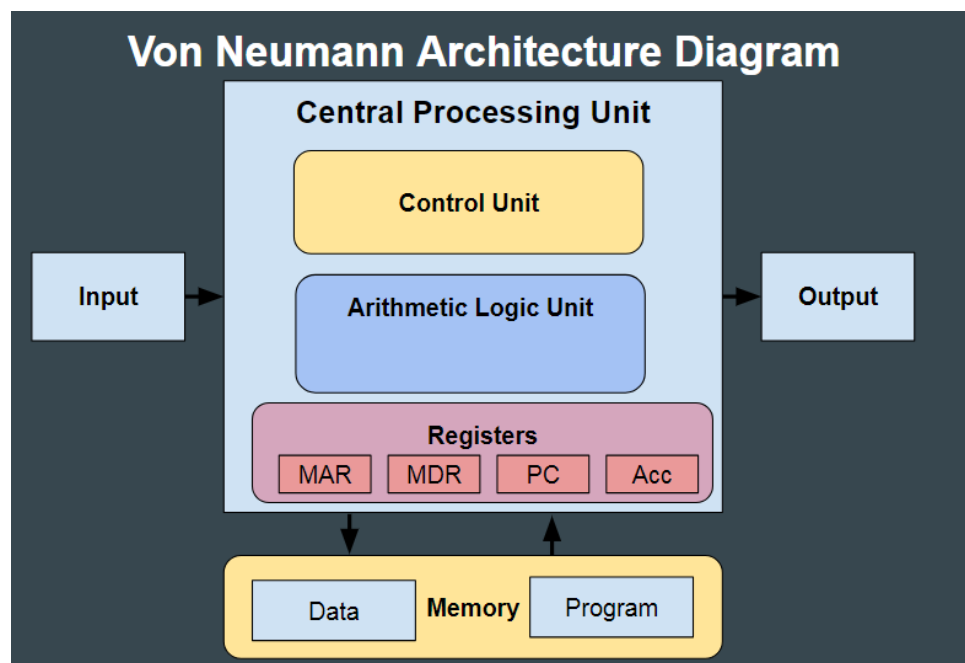


von Neumann Architecture

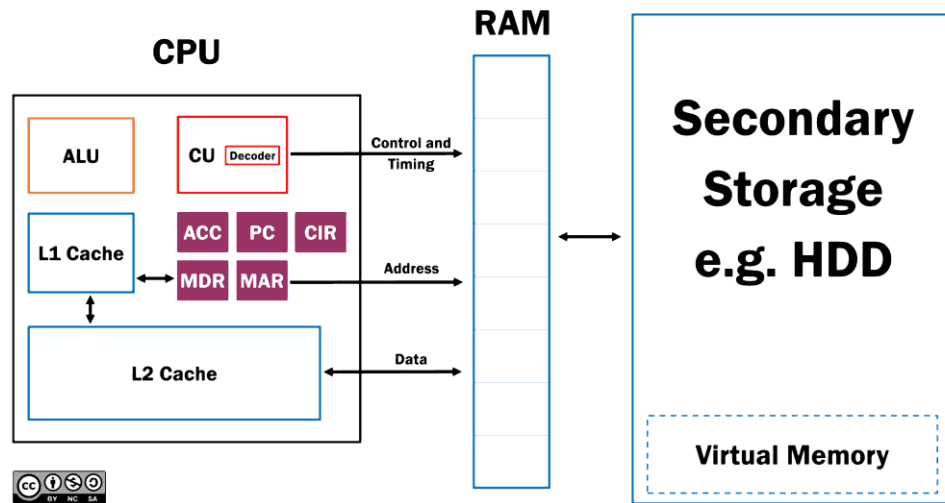
- A more complete view of the computer *system* architecture that integrates interaction (human or otherwise) consists of:



Von Neumann Architecture Diagram



Computer Systems - Von Neumann Architecture



Processor - 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
- *Not to mention* 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

The Fetch-Execute Cycle

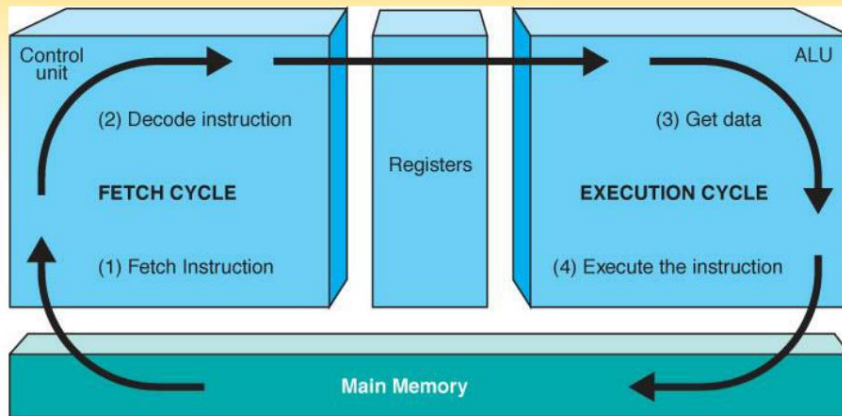
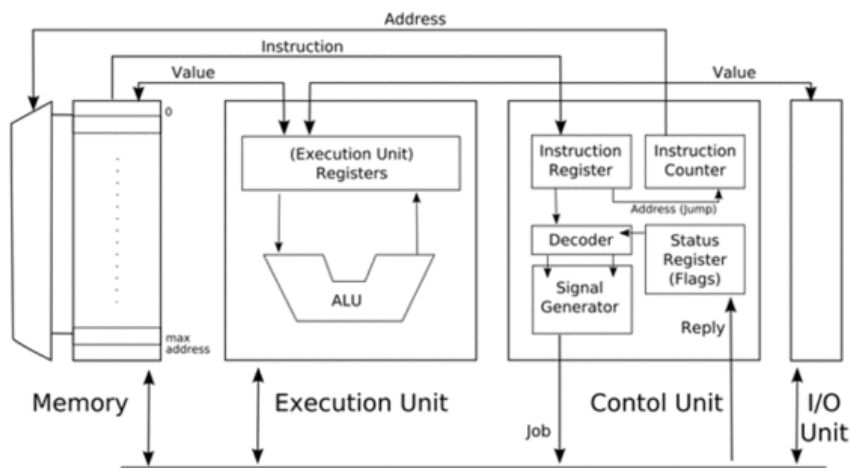


Figure 5.3 The Fetch-Execute Cycle

17

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



AQA Computing

The Fetch-Execute cycle

Program Counter: 0000 0000 0000 0100

Memory Address Register: 0000 0000 0000 0011

Memory Buffer Register: 0101 0000 0000 0110

Current Instruction Register (CIR): 0101 0000 0000 0110

Op-code: 0101 Operand: 0000 0000 0110

Accumulator: 0000 0000 0000 0100

1	
2	
3	0101 0000 0000 0110 "ADD #6"
4	
5	
6	
:	

Fetch

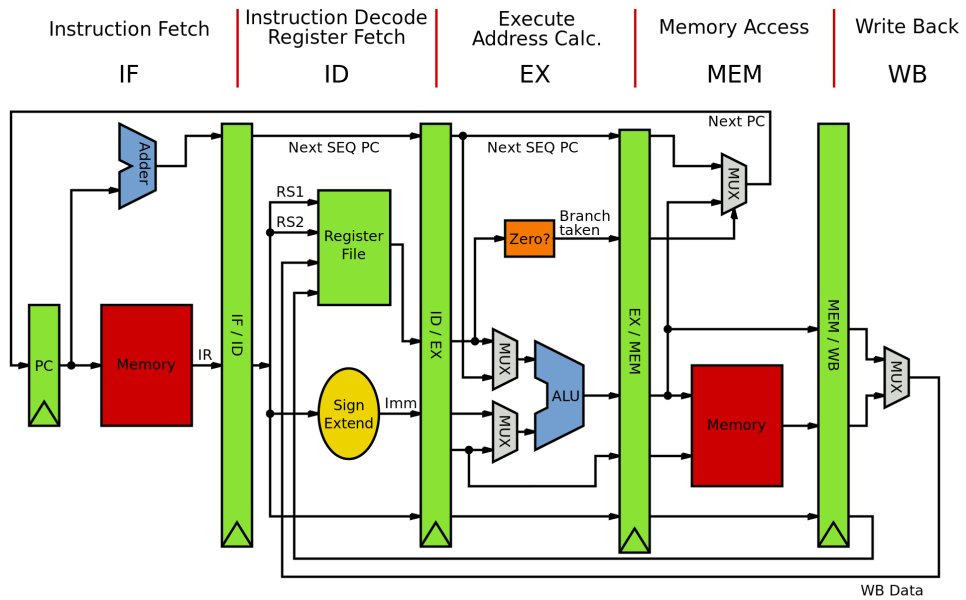
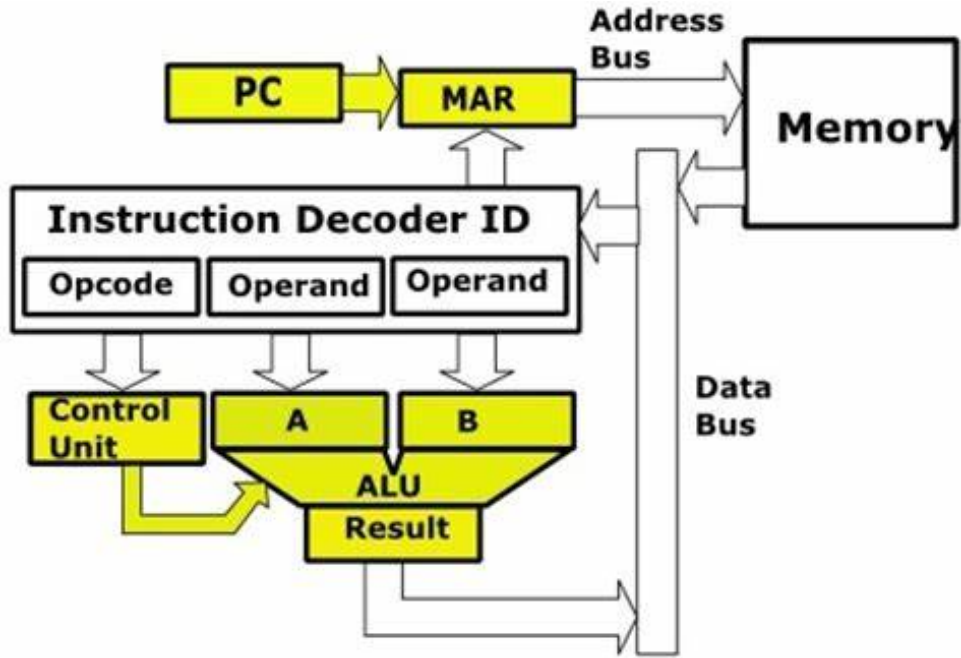
MAR ← [PC]

PC ← [PC] + 1

MBR ← memory contents

CIR ← [MBR]

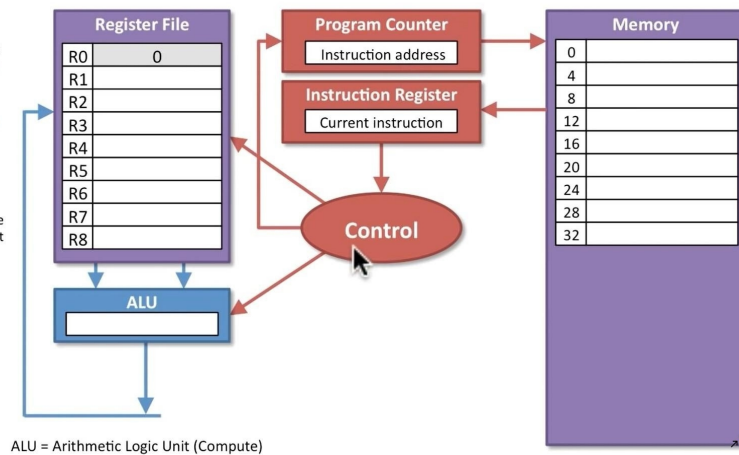
Screencast-O-Matic.com



Data operations in detail

1. Data Operations

1. Program Counter holds the instruction address.
2. Instructions are *fetch*ed from memory into the Instruction Register.
3. Control logic *dec*odes the instruction and tells the ALU and Register File what to do.
4. ALU *ex*ecutes the instruction and results flow back to the Register File.
5. The Control logic *up*dates the Program Counter for the next instruction.



ALU = Arithmetic Logic Unit (Compute)

Data transfers in detail

2. Data Transfers

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

2. Sequencing

The label constant is in instruction words, so it needs to be multiplied by 4 to convert to byte address.

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

Processor - Datapath

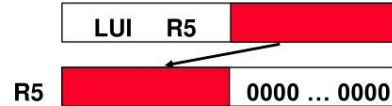
Review: MIPS data transfer instructions

- 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

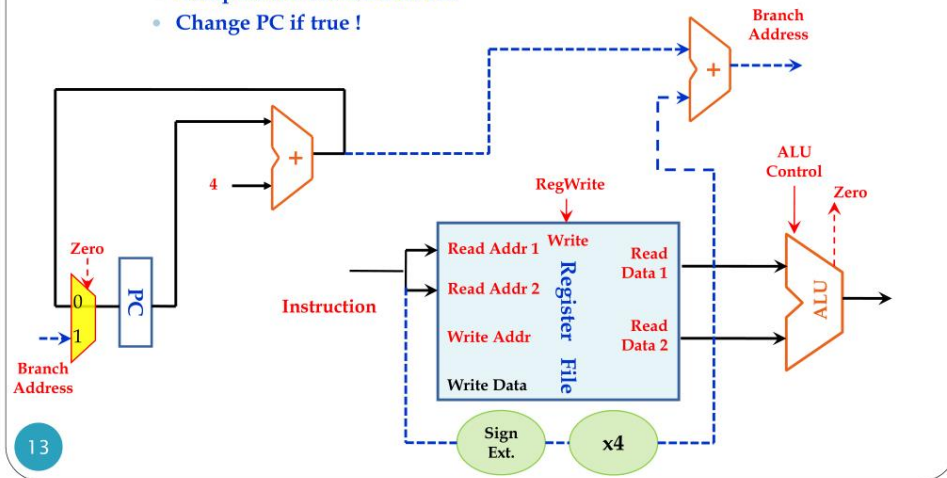
<u>Instruction</u>	<u>Comment</u>
SW R3, 500(R4)	Store word
SH R3, 502(R2)	Store half
SB R2, 41(R3)	Store byte
LW R1, 30(R2)	Load word
LH R1, 40(R3)	Load half word
LHU R1, 40(R3)	Load half word unsigned
LB R1, 40(R3)	Load byte
LBU R1, 40(R3)	Load byte unsigned
LUI R1, 40	Load Upper Immediate (16 bits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

Why do we need LUI?

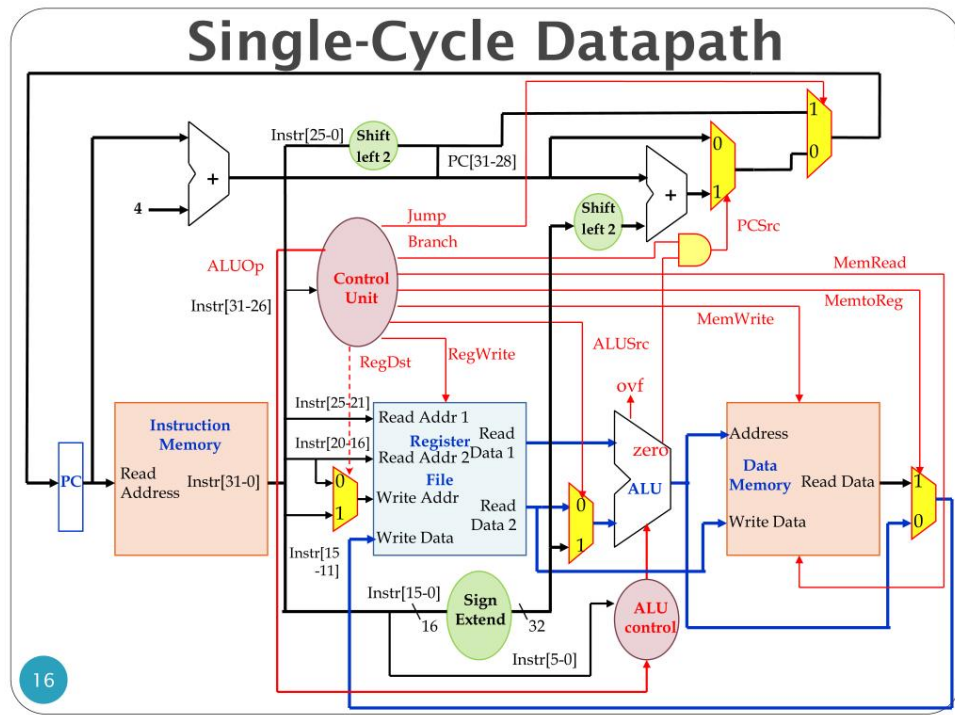


Single-Cycle Datapath

- Execution Datapath
 - Branch Instruction
 - Compare the two registers
 - Compute the branch address
 - Change PC if true !



13



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)

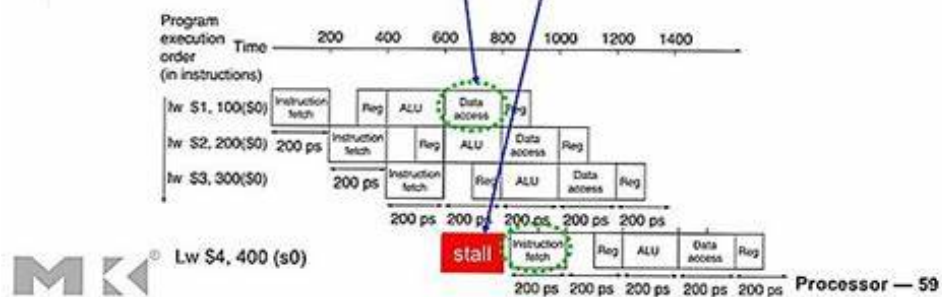
Pipeline Hazards (2)

- 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 *later* than the stalled instruction are stopped, while the ones issued *earlier* must continue
- No new instructions are fetched during the stall

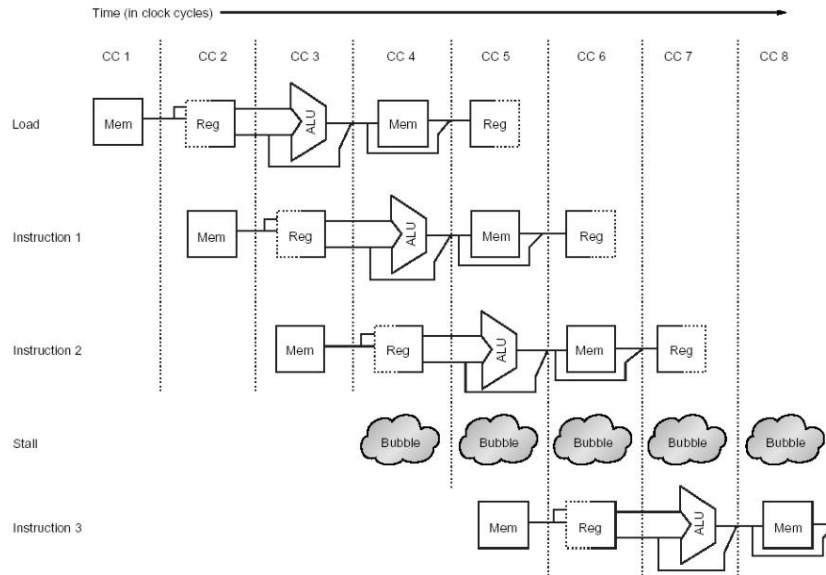
Solution for Pipeline Hazards

Structure Hazards

- **Conflict for use of a resource**
- Suppose that we have 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)



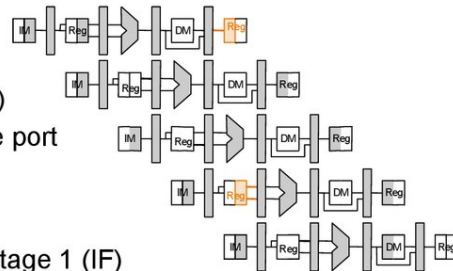
- Stall cycle added (commonly called pipeline *bubble*)

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

Consider the following set of instructions in a 5-stage pipeline.
 Operands are read in ID.
 MEM is memory Write for result; RW is Register Write for result

R3 is accessed in READ mode; expect the result of ADD to be available in R3

- ADD R3, R6, R5 - Results to be written in R3
- SUB R4, R3, R5 - R3 has one of the operand
- OR R6, R3, R7 - R3 has one of the operand
- AND R8, R3, R7 - R3 has one of the operand
- XOR R12, R3, R10 - R3 has one of the operand

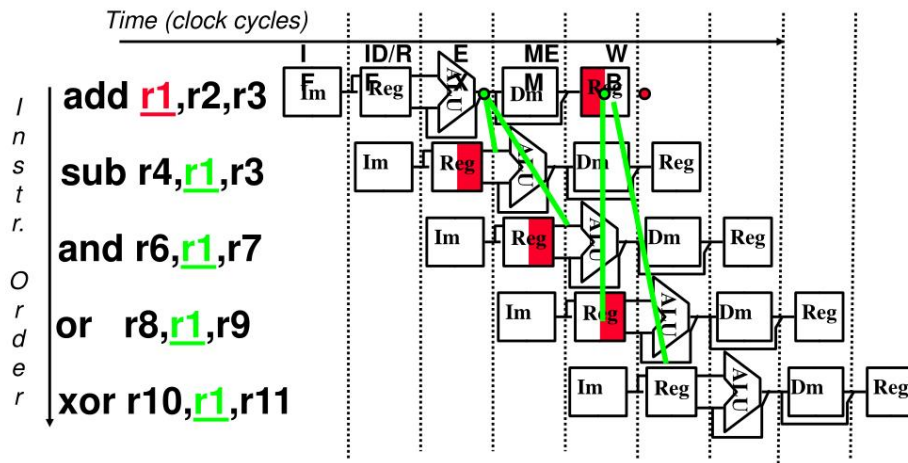
But result of ADD written in R3 at t5

	t1	t2	t3	t4	t5	t6	t7	t8	t9
ADD R3, R6, R5	IF	ID	IE	MEM	RW R3	--	--	--	--
SUB R4, R3, R5	--	IF	ID R3	IE	MEM	RW	--	--	--
OR R6, R3, R7	--	--	IF	ID R3	IE	MEM	RW	--	--
AND R8, R3, R9	--	--	--	IF	ID R3	IE	MEM	RW	--
XOR R10, R3, R11	--	--	--	--	IF	ID R3	IE	MEM	RW

Note when each instruction is accessing R3

Data Hazard Solution:

- **“Forward”** result from one stage to another



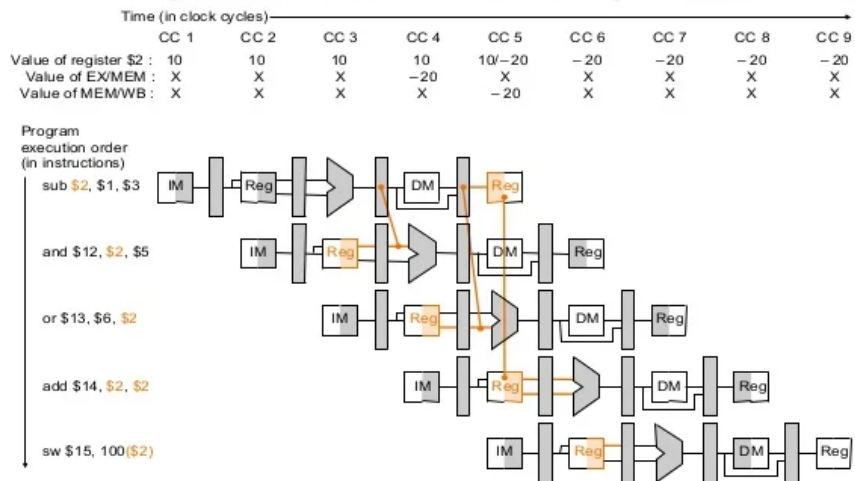
- **“or”** OK if define read/write properly

cs 152 L1 3.13

DAP Fa97, © U.CB

Data Hazard Solution: Forwarding

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



Assumption:

- The register file forwards values that are read and written during the same cycle.

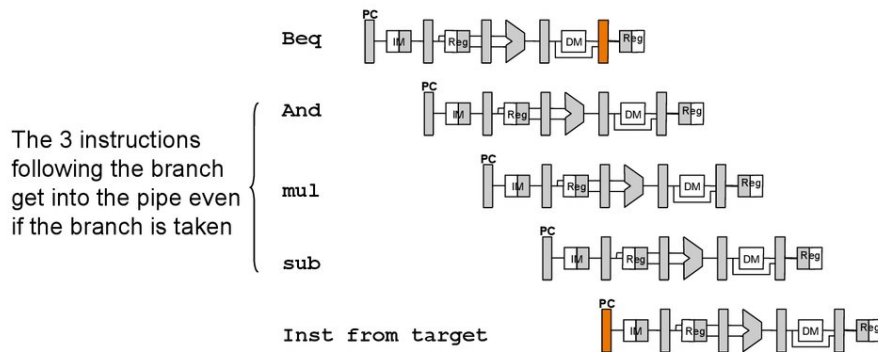
CSC430/830

Pipeline Hazards

Control Hazard

- Also known as *branch hazard*
- Pipeline makes wrong decision on branch prediction
- Brings instructions into pipeline that must subsequently be discarded
- Dealing with Branches
 - Multiple Streams
 - Prefetch Branch Target
 - Loop buffer
 - Branch prediction
 - Delayed branching

Control Hazard on Branches



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

CSCB130/830

Pipeline Hazards

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

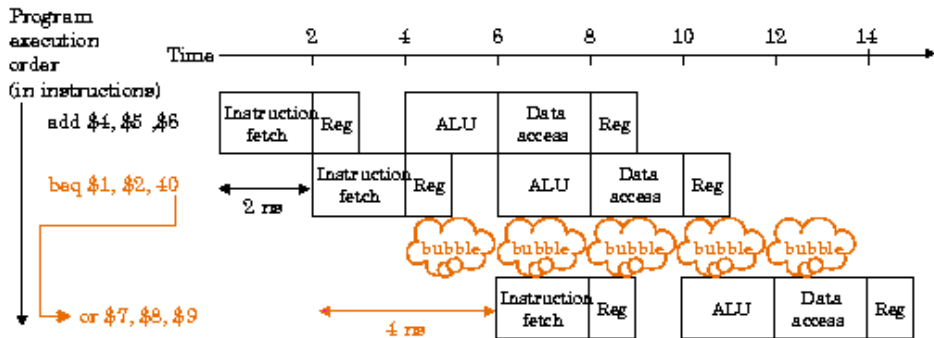
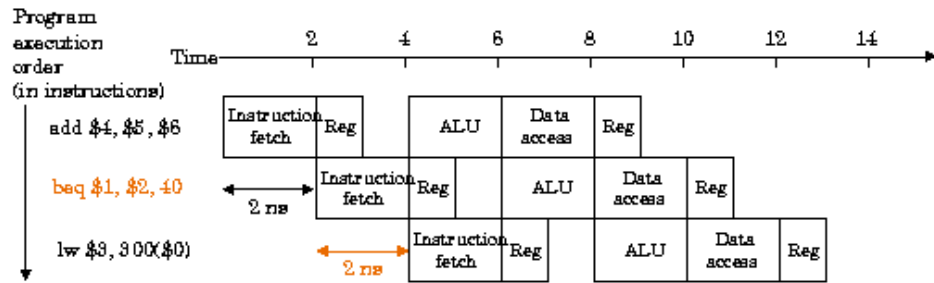
- number of cycles in between * branch frequency = the contribution to CPI due to branches

Predict the branch outcome to avoid stalling

Spring 2003

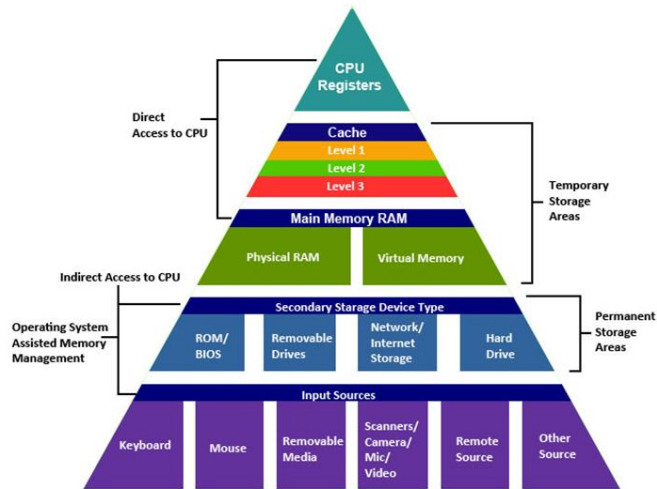
CSE P548

1



Memory Hierarchy

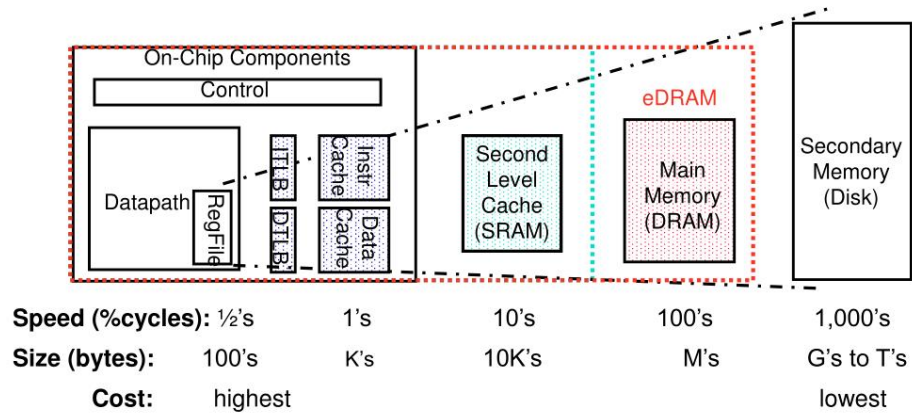
The Memory Hierarchy



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

A Typical Memory Hierarchy

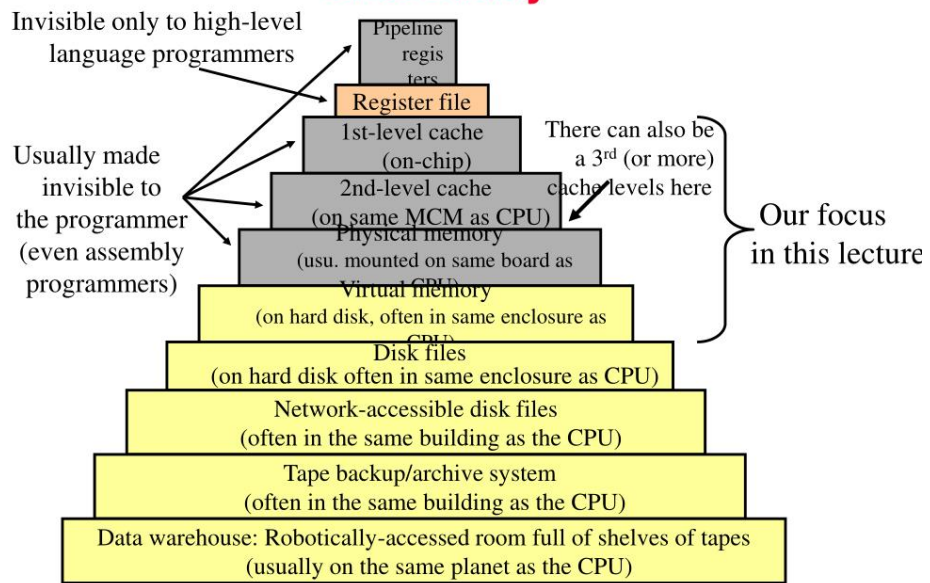
- By taking advantage of the principle of locality
 - Can present the user with as much memory as is available in the cheapest technology
 - at the speed offered by the fastest technology



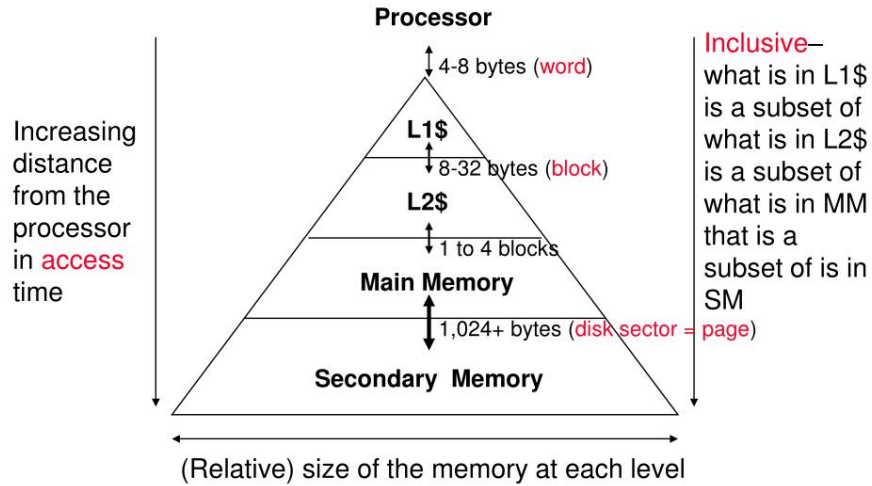
CEG3420 L13.9

Qiang Xu CUHK, Spring 2011

Many Levels in Memory Hierarchy



Characteristics of the Memory Hierarchy



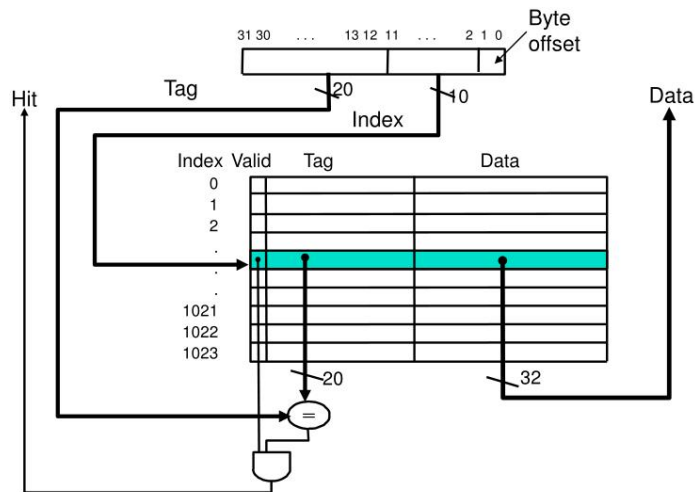
CPE432 Chapter 5A.7

Dr. W. Abu-Sufah, UJ

Block Size / Hit Rate / Miss Rate

MIPS Direct Mapped Cache Example

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

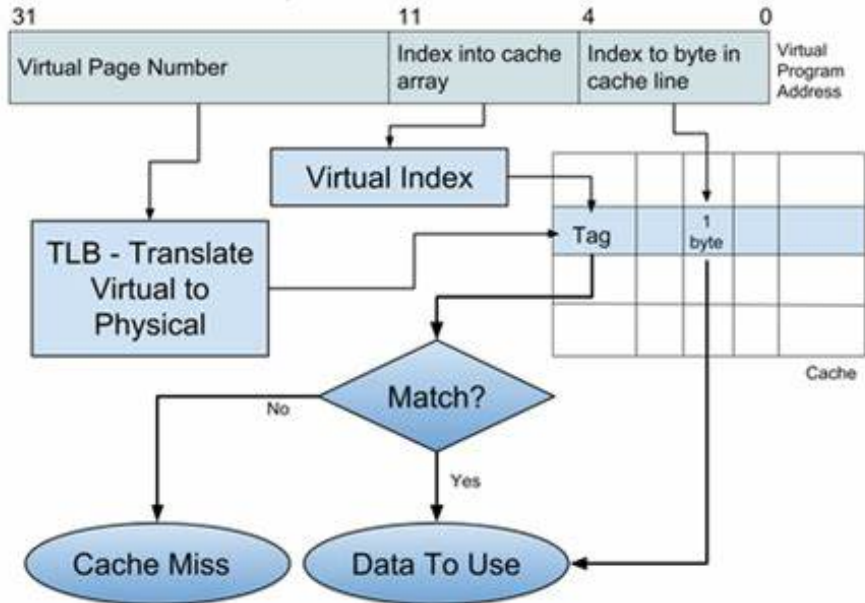


What kind of locality are we taking advantage of?

CEG3420 L13.34

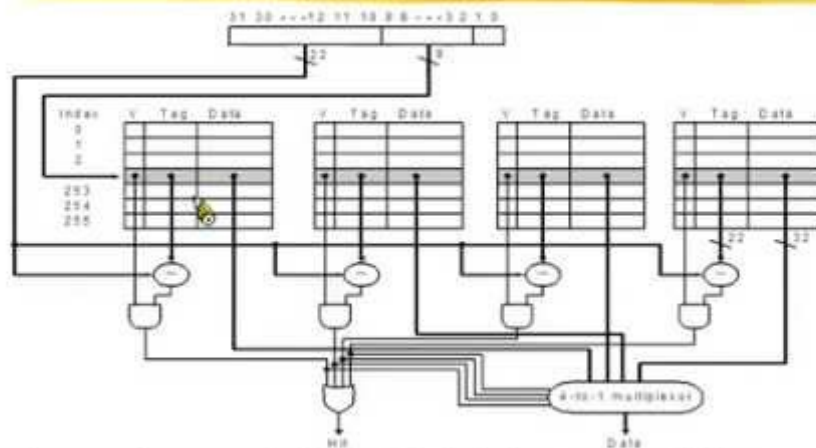
Qiang Xu CUHK, Spring 2011

MIPS Cache Look Up



Adapted from Imagination Technologies MIPS basic training course materials

A 4-Way Set-Associative Cache



◆ Increasing associativity shrinks index, expands tag

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.

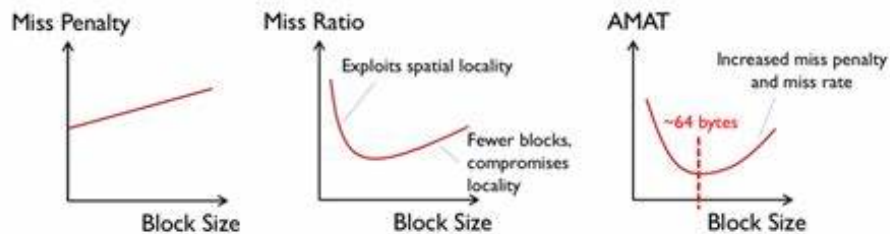
2/15/99

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



of cache hits

= Hit ratio
 (# of cache hits + # of cache misses)

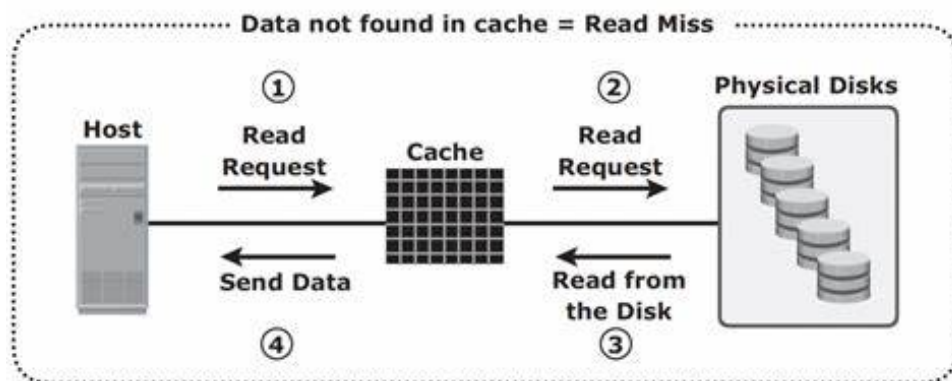
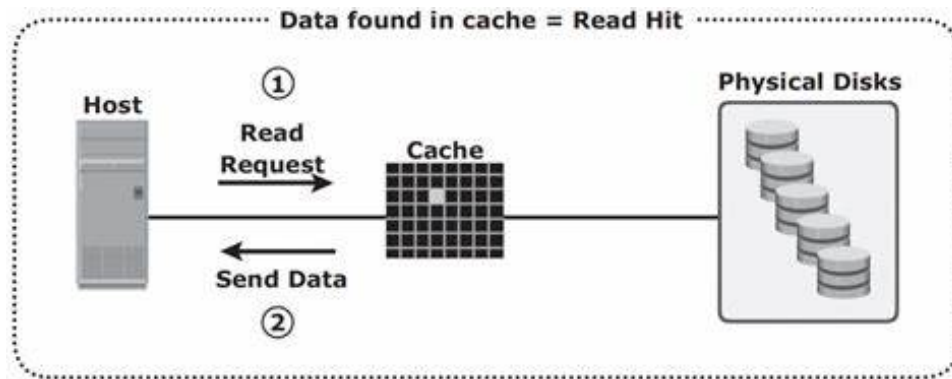
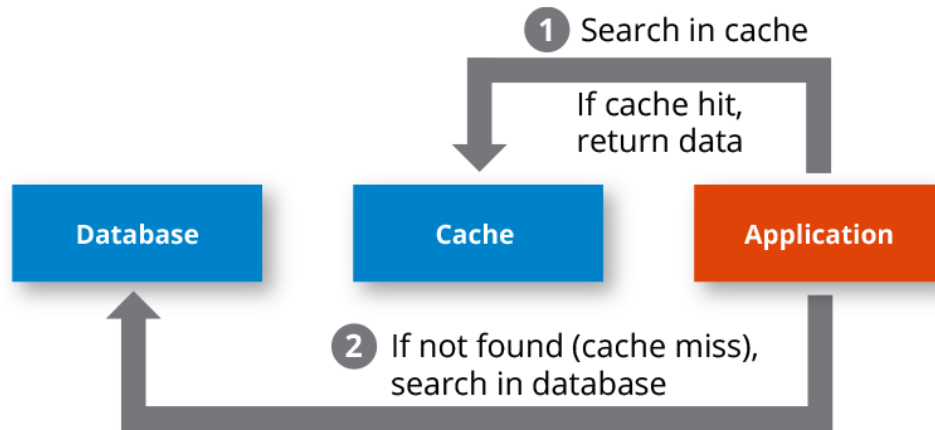
OR

Hit ratio = 1 - 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

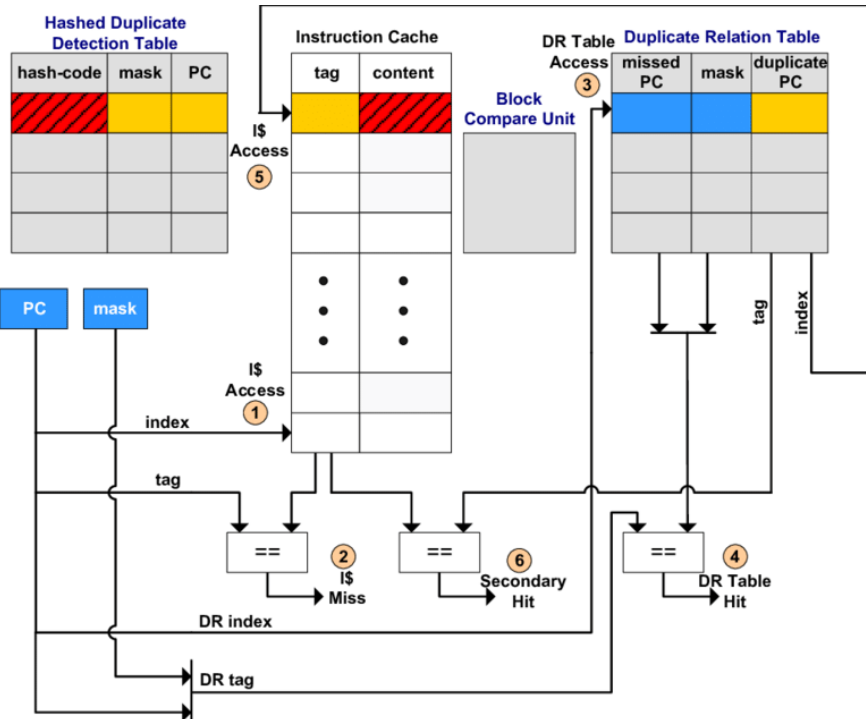
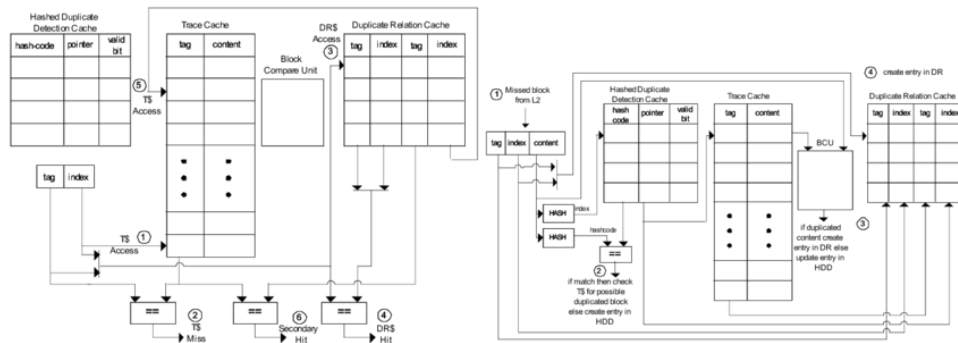


Types of Cache Misses: *The Three C's*

- 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



Cache Performance

Cache Performance Equations

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

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

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

- **CPU time formula:**

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

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

46

Cache Performance

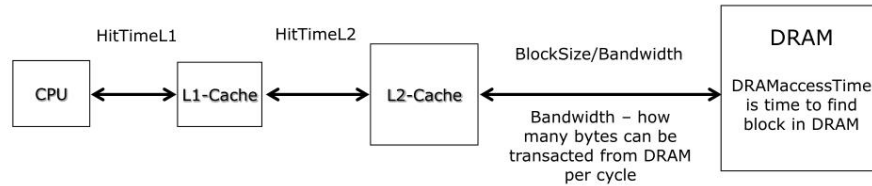
- $CPI_{\text{contributed by cache}} = CPI_c$
= miss rate * number of cycles to handle the miss
- Another important metric
Average memory access time = cache hit time * hit rate
+ Miss penalty * (1 - hit rate)

Caches CSE 471

15

2-level Cache Performance Equations

- L1 AMAT = $\text{HitTimeL1} + \text{MissRateL1} * \text{MissPenaltyL1}$
 - MissLatencyL1 is low, so optimize HitTimeL1
- $\text{MissPenaltyL1} = \text{HitTimeL2} + \text{MissRateL2} * \text{MissPenaltyL2}$
 - MissLatencyL2 is high, so optimize MissRateL2
 - $\text{MissPenaltyL2} = \text{DRAMaccessTime} + (\text{BlockSize}/\text{Bandwidth})$
 - 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



18

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 (L₂)
- * 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

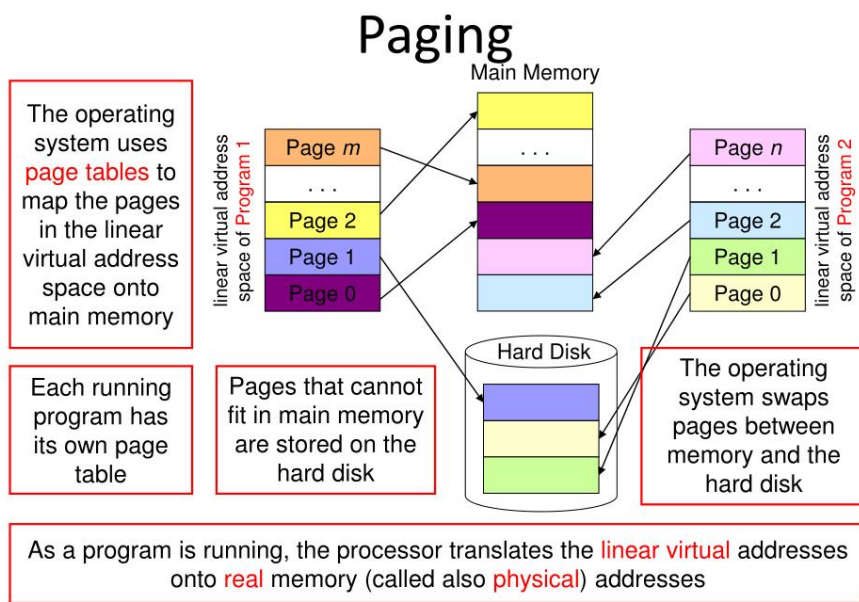
Miss Types

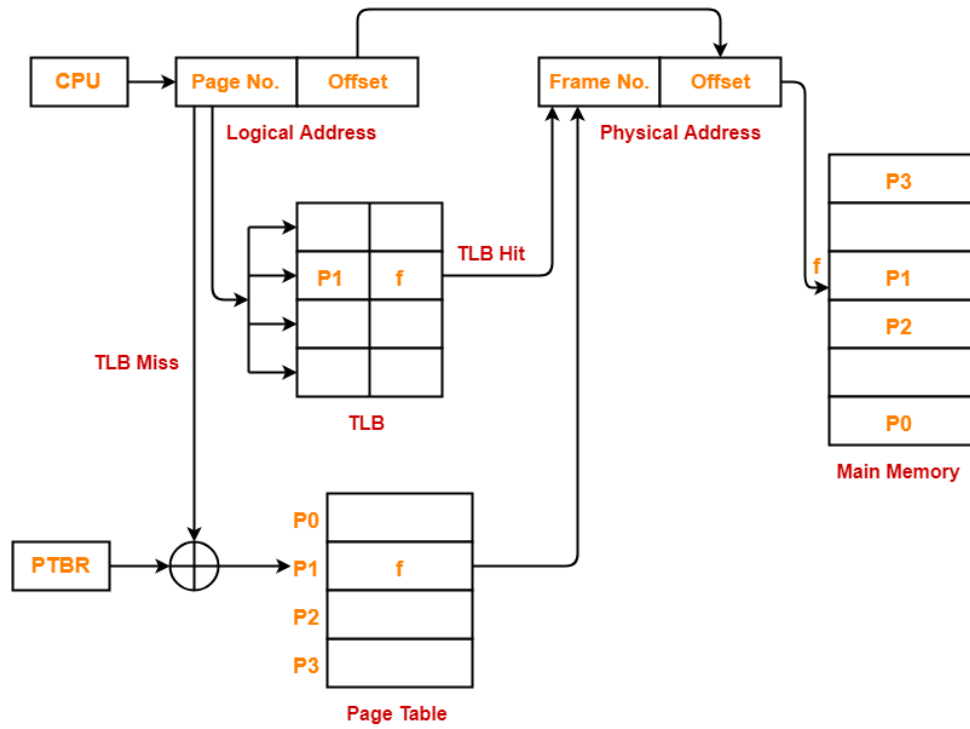
- **Compulsory** ("Cold Start") Misses
 - First access to line not in cache
- **Capacity** Misses
 - Active portion of memory exceeds cache size
- **Conflict** Misses
 - Active portion of address space fits in cache, but too many lines map to same cache entry
 - Direct mapped and set associative placement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

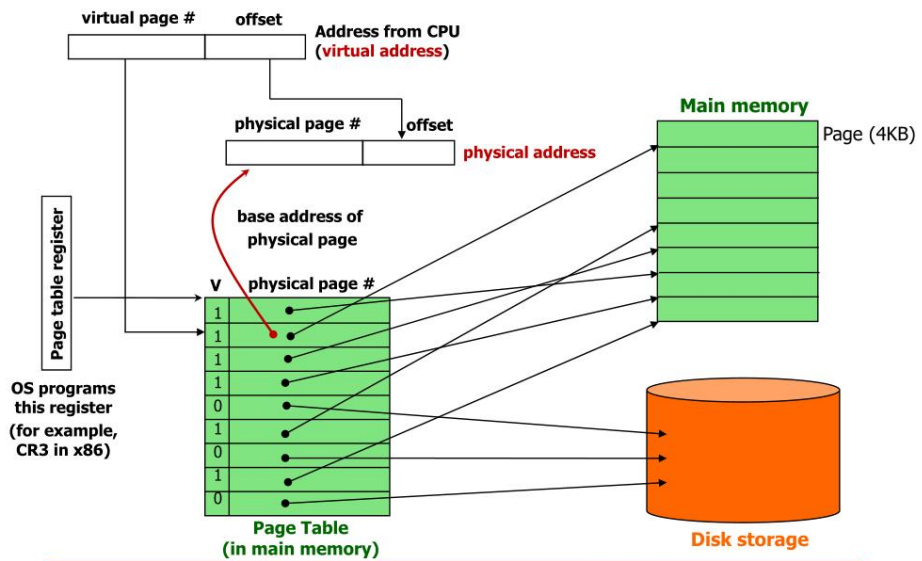
Address Translation Mechanism





Translating Logical Address into Physical Address

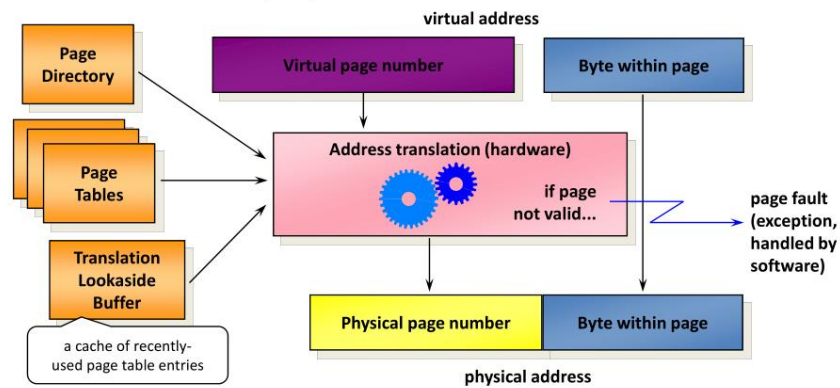
Address Translation Mechanism



Address Translation with Cache

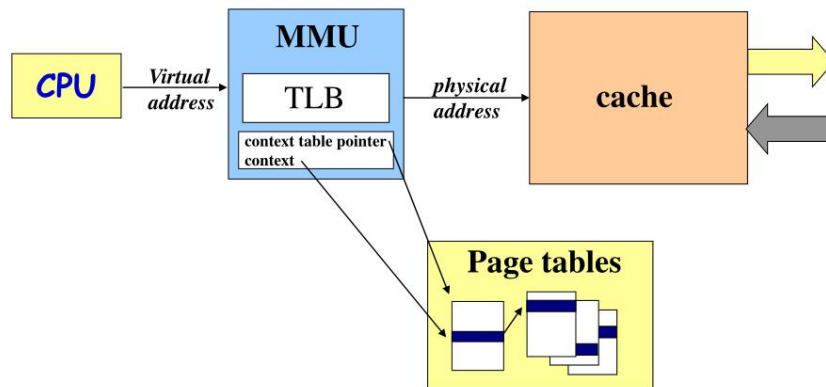
Virtual Address Translation

- The hardware converts each valid virtual address to a physical address



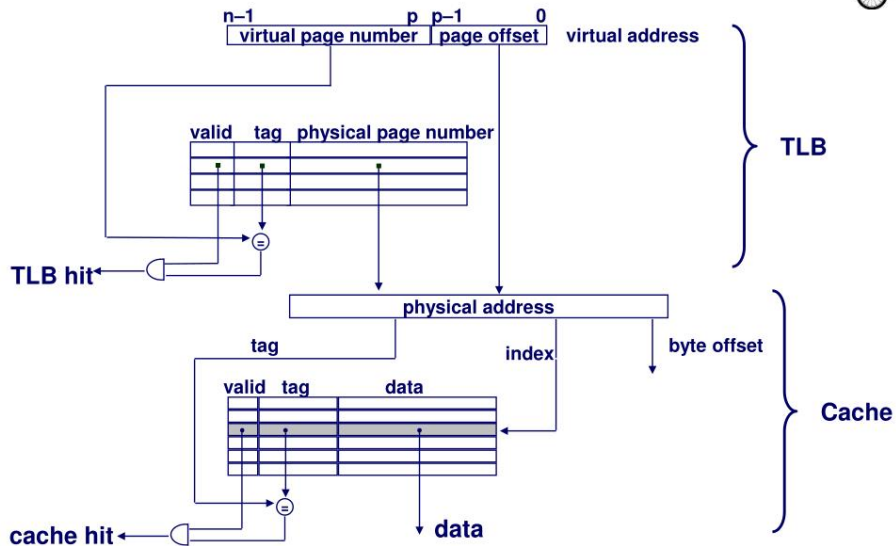
31

Address Translation Overview





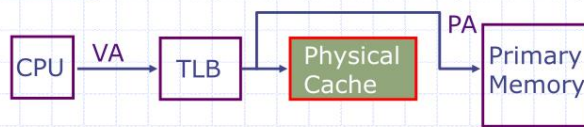
Address Translation With a TLB



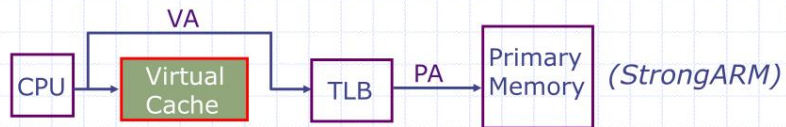
- 32 -

CS 105

Physical or Virtual Address Caches?

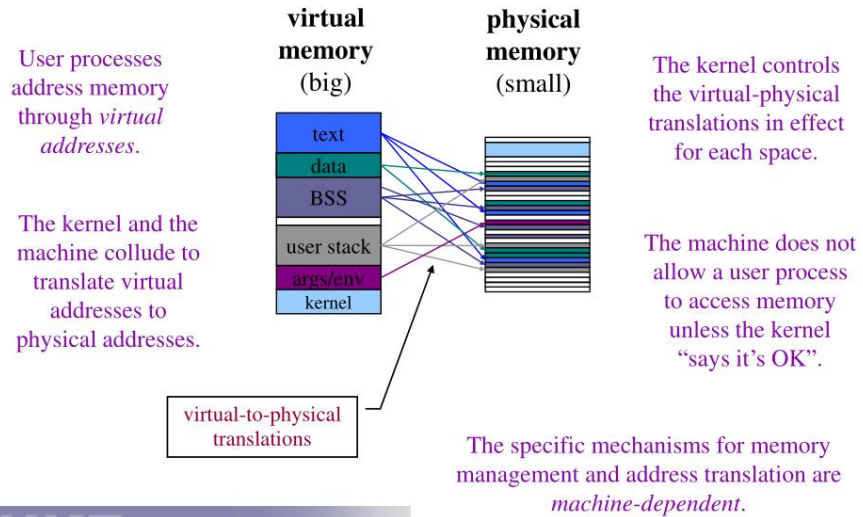


Alternative: place the cache before the TLB

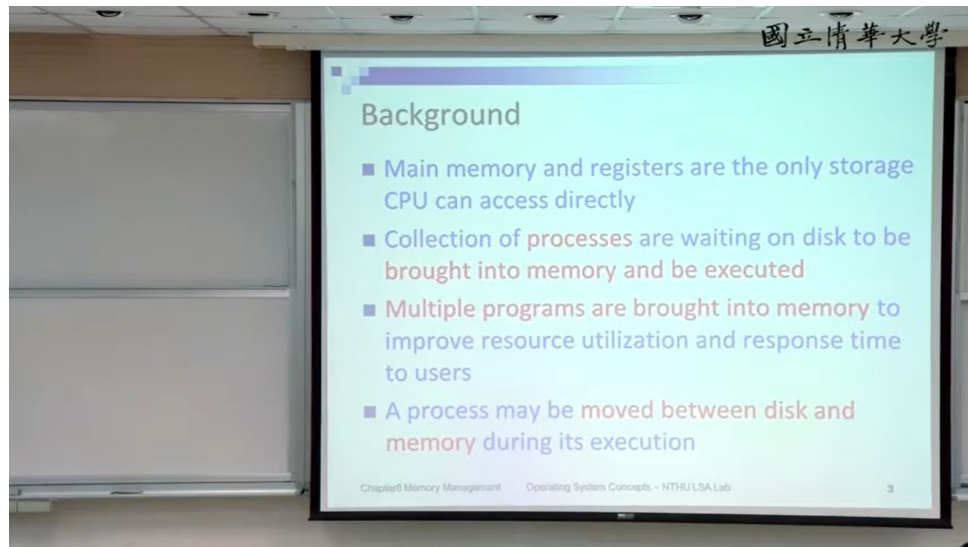


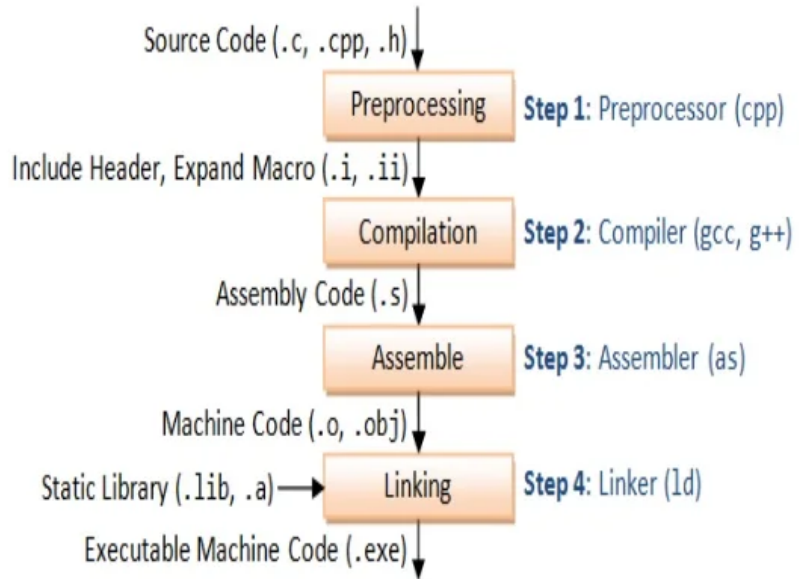
- ◆ one-step process in case of a hit (+)
- ◆ cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- ◆ *aliasing problems* due to the sharing of pages (-)

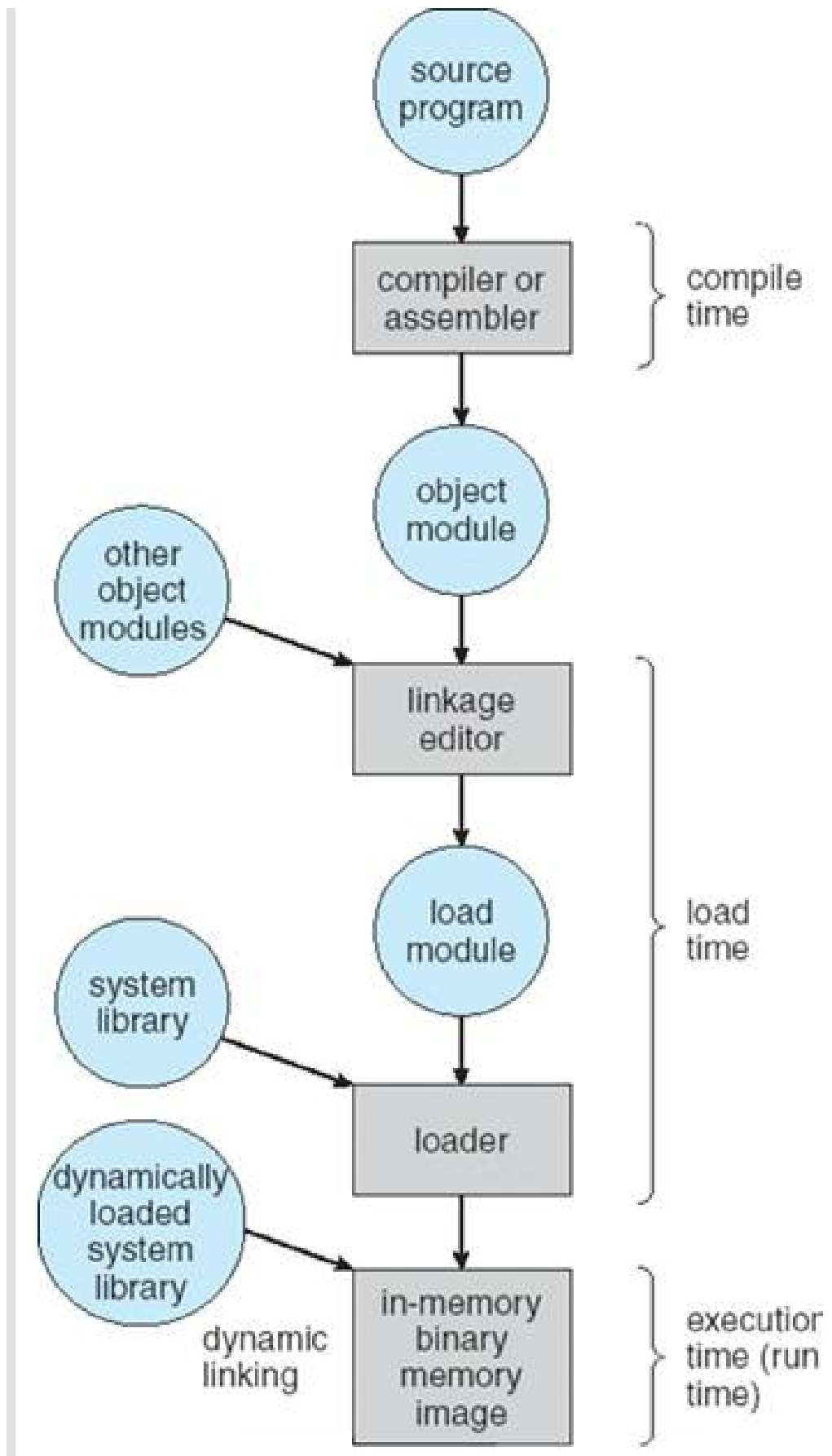
Review: Virtual Addressing



Memory Management

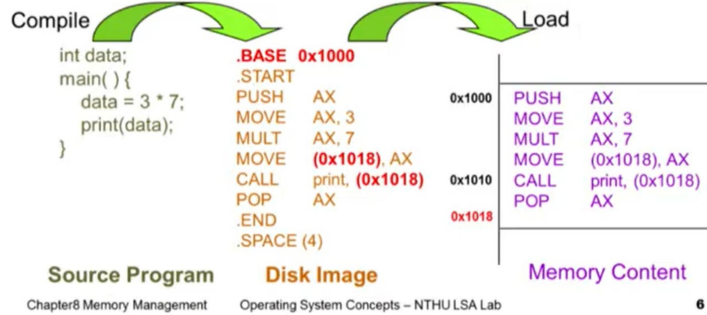






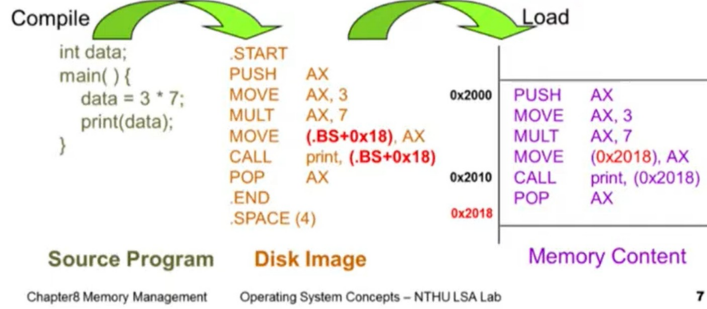
Address Binding – Compile Time

- Program is written as symbolic code
- Compiler translates symbolic code into *absolute code*
- If starting location changes → **recompile**
- Example: MS-DOS .COM format binary



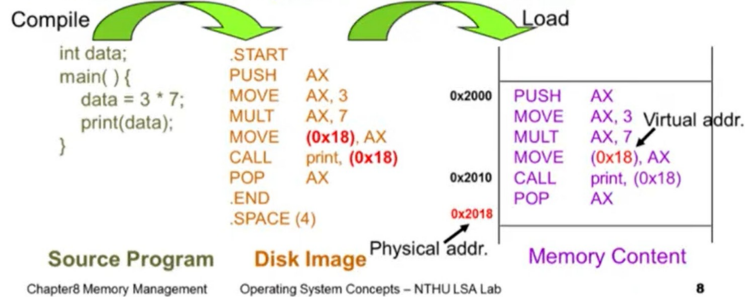
Address Binding – Load Time

- Compiler translates symbolic code into *relocatable code*
- **Relocatable code:**
 - Machine language that can be run from any memory location
- If starting location changes → **reload the code**



Address Binding – Execution Time

- Compiler translates symbolic code into *logical-address (i.e. virtual-address) code*
- Special **hardware (i.e. MMU)** is needed for this scheme
- Most general-purpose OS use this method

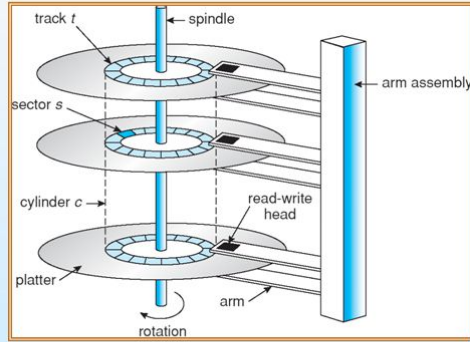


Storage Management

Disk Structure



Disk Structure



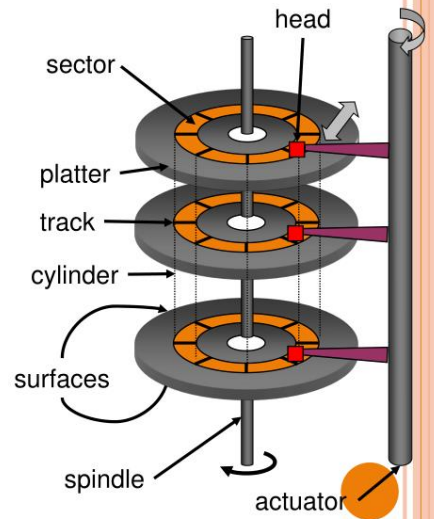
Moving-head disk mechanism

- 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

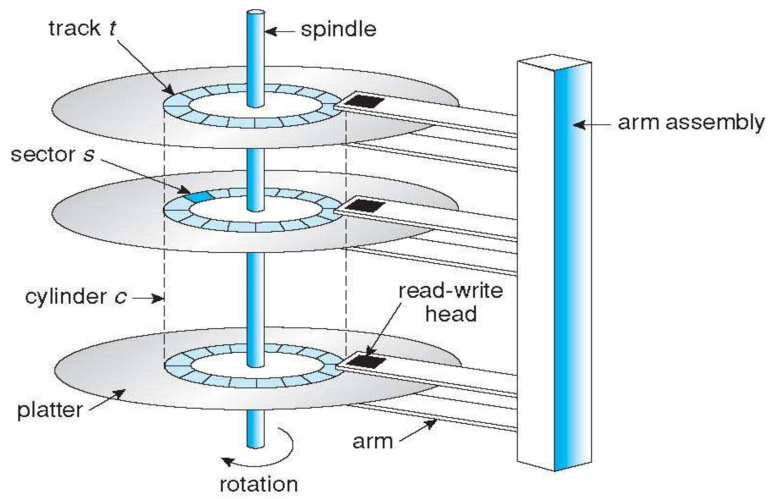


DISK DRIVE STRUCTURE

- 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



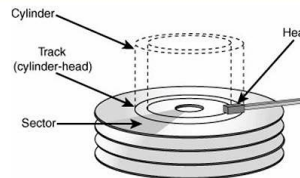
Moving-head Disk Mechanism



12.5

Operating System

Disk Structure



- Track
- Sector
- Cylinder
- Spindle
- Read/ Write Head (Arm assembly)

- Seek Time
- Rotational Latency
- RPM
- Transfer Time

Presented by :Parul Vaghamshi

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, ISA Lab 27

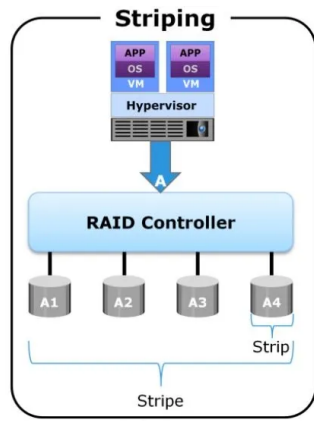


Figure 1

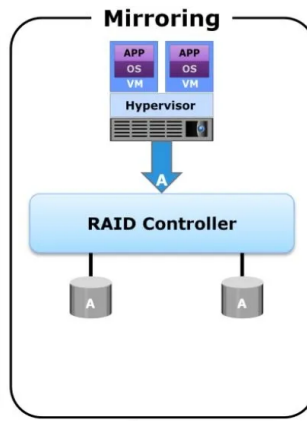


Figure 2

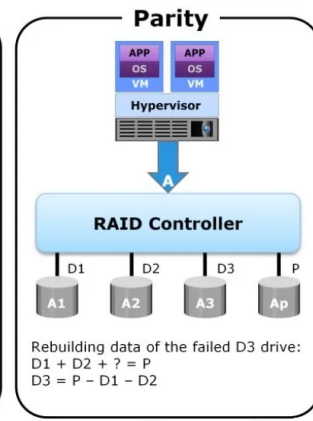
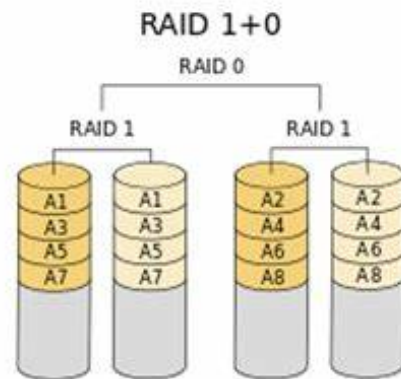
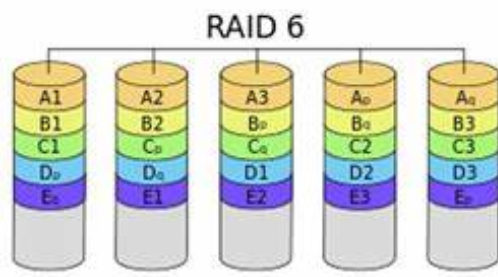
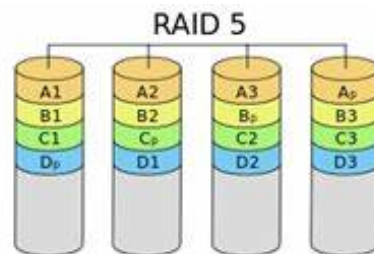
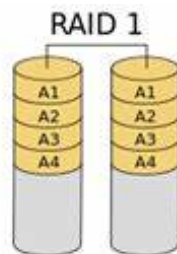
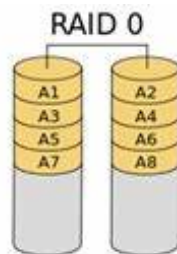


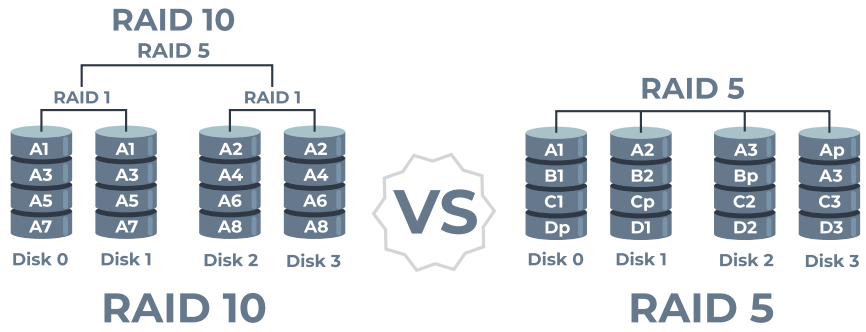
Figure 3

國立清華大學

- 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

RAID 0 Example
File1: 0011
File2: 110101

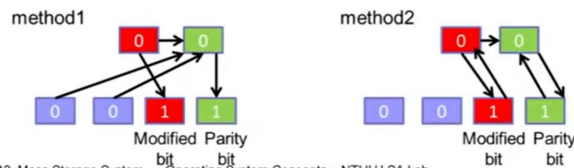


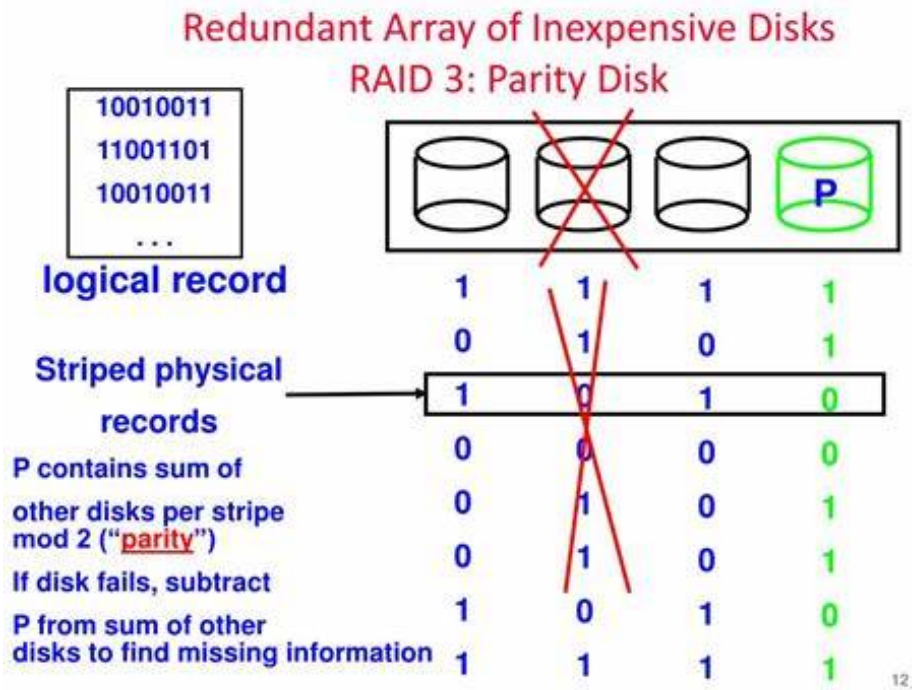


RAID 5: Distributed Parity

國立清華大學

- 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

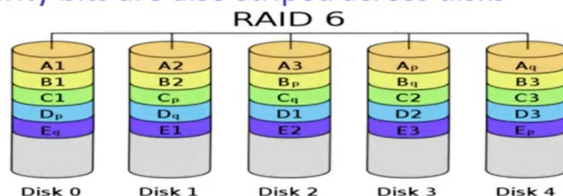




RAID 6: P+Q Dual Parity Redundancy

國立清華大學

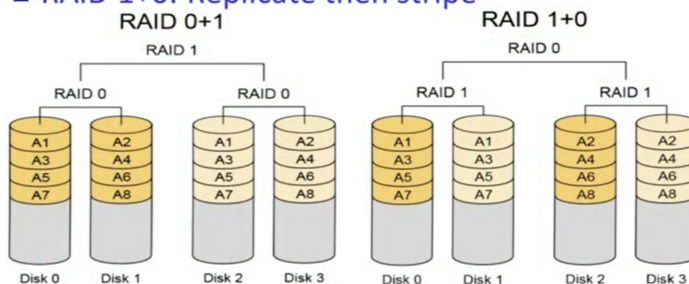
- Like RAID 5, but stores extra redundant information to guard against **multiple disk failure**
- Use **ECE code** (i.e. Error Correction Code) instead of single parity bit
- Parity bits are also striped across disks



Hybrid RAID

國立清華大學

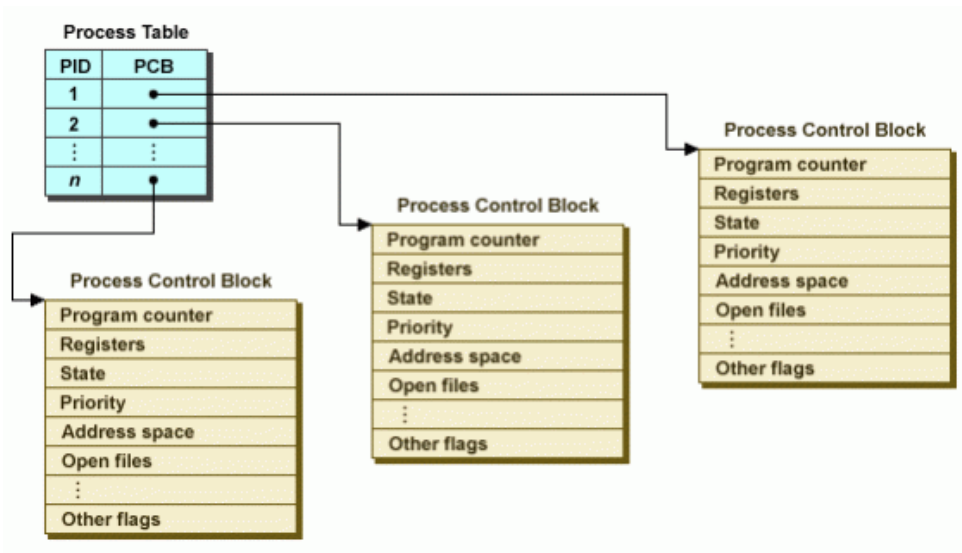
- RAID 0+1: Stripe then replicate
- RAID 1+0: Replicate then stripe



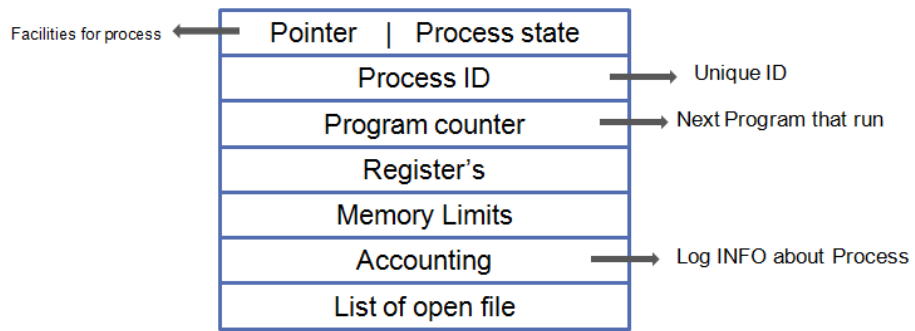
*First level often control by a controller. Therefore, RAID 10 has better fault tolerance than RAID 01 when multiple disk fails

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

- PCB(Process Control Block)

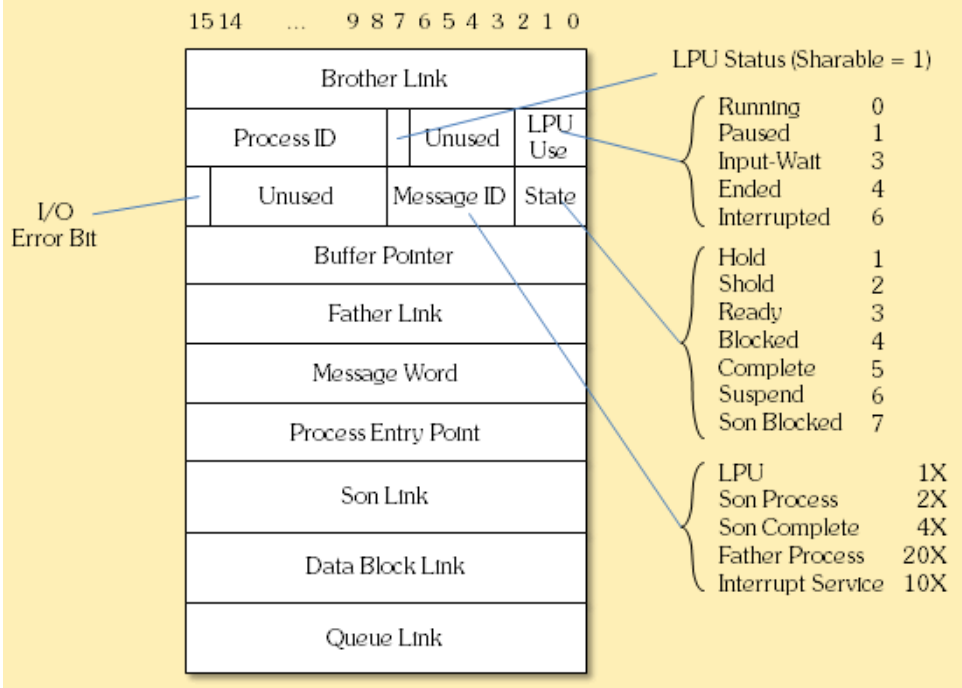


PROCESS CONTROL BLOCK (PCB)

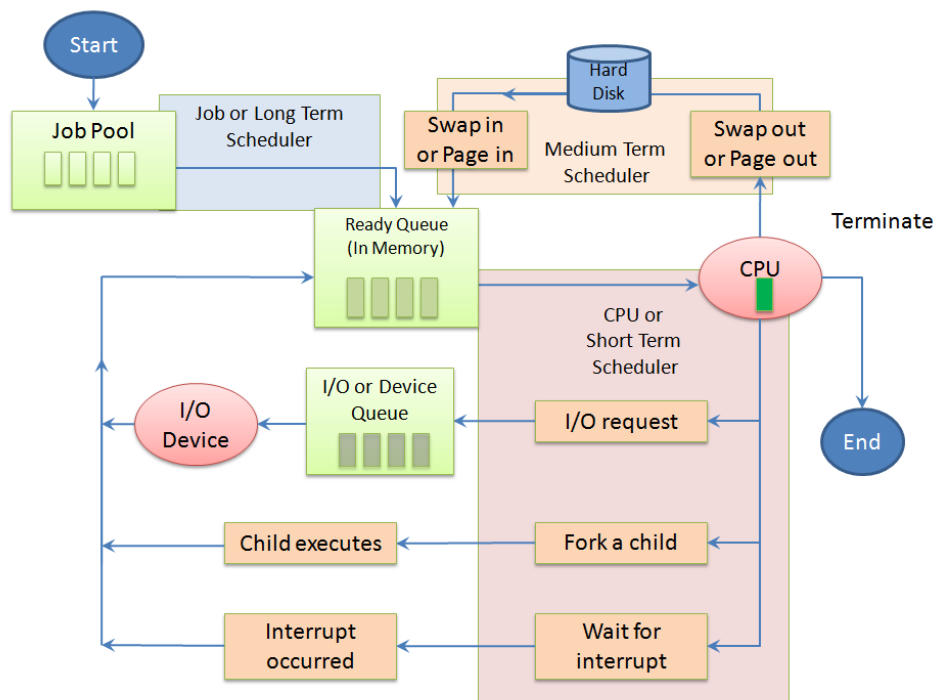
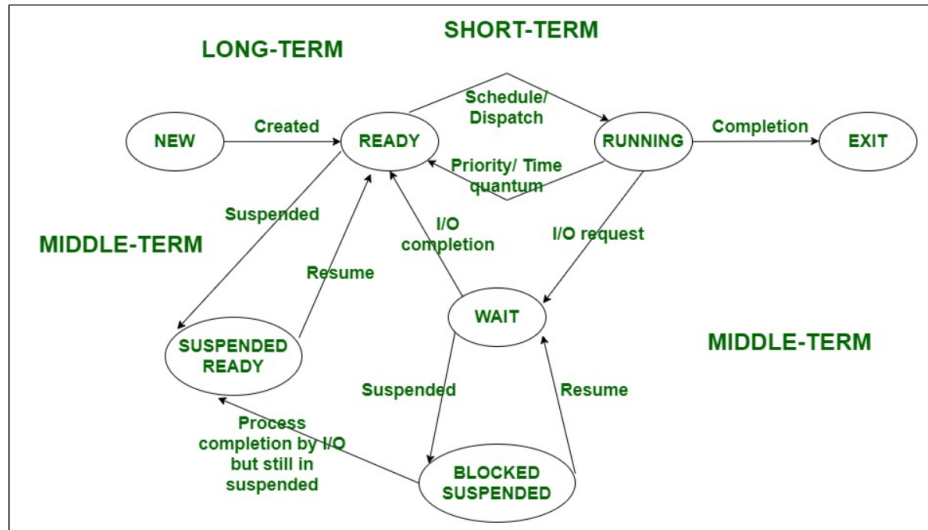


PCB Diagram

Process Control Block (PCB)



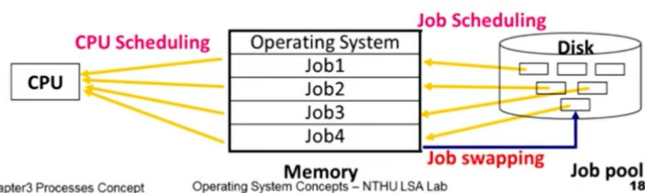
- Scheduling



Schedulers

國立清華大學

- **Short-term scheduler (CPU scheduler)** – selects which process should be executed and allocated CPU (Ready state → Run state)
- **Long-term scheduler (job scheduler)** – selects which processes should be loaded into memory and brought into the ready queue (New state → Ready state)
- **Medium-term scheduler** – selects which processes should be swapped in/out memory (Ready state → Wait state)

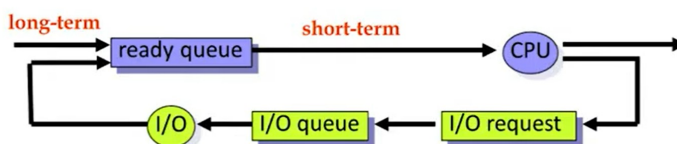


Long-Term Scheduler

- Control **degree of multiprogramming**
- Execute less frequently (e.g. invoked only when a process leaves the system or **once several minutes**)
- Select a **good mix of CPU-bound & I/O-bound** processes to increase system overall performance
- UNIX/NT: no long-term scheduler
 - Created process placed in memory for short-term scheduler
 - Multiprogramming degree is bounded by hardware limitation (e.g., # of terminals) or on the self-adjusting nature of users

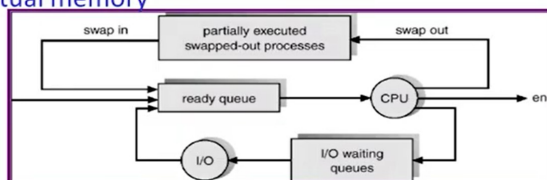
Short-Term Scheduler

- Execute quite frequently (e.g. **once per 100ms**)
- Must be efficient:
 - if 10 ms for picking a job, 100 ms for such a pick,
 - ➔ overhead = $10 / 110 = 9\%$

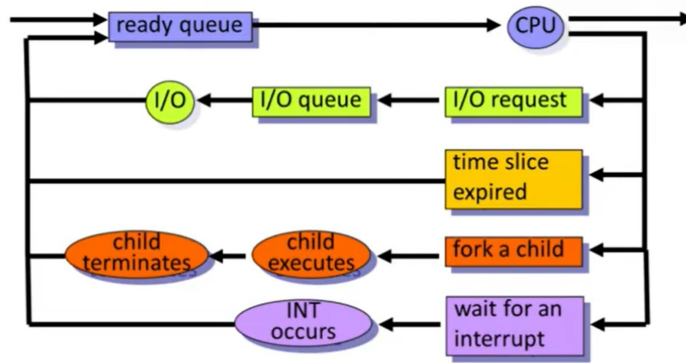


Medium-Term Scheduler

- **swap out**: removing processes from memory to reduce the degree of multiprogramming
- **swap in**: reintroducing swap-out processes into memory
- Purpose: improve process mix, free up memory
- Most modern OS doesn't have medium-term scheduler because having sufficient physical memory or using virtual memory



Process Scheduling Diagram

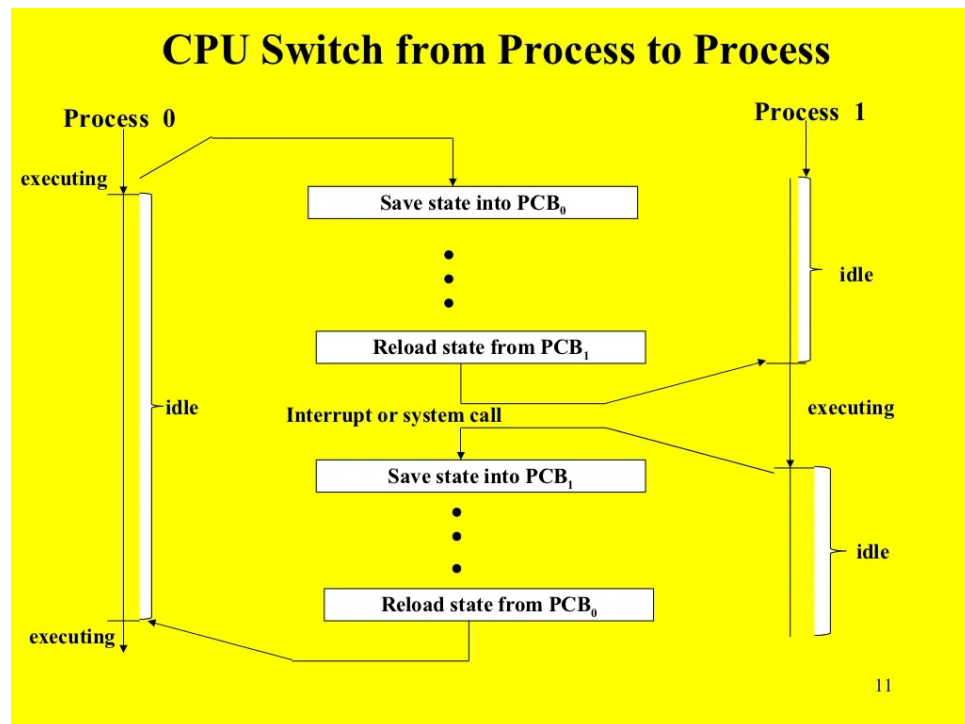


Differentiate between short term and long term scheduler.

{**Note: Any other relevant difference shall be considered**}

Sr. No	Short term scheduler	Long term scheduler
1	It is a CPU scheduler	It is a job scheduler
2	It selects processes from ready queue which are ready to execute and allocates CPU to one of them.	It selects processes from job pool and loads them into memory for execution.
3	Access ready queue and CPU.	Access job pool and ready queue
4	It executes frequently. It executes when CPU is available for allocation.	It executes much less frequently. It executes when memory has space to accommodate new process.
5	Speed is fast	Speed is less than short term scheduler
6	It provides lesser control over degree of multiprogramming	It controls the degree of multiprogramming

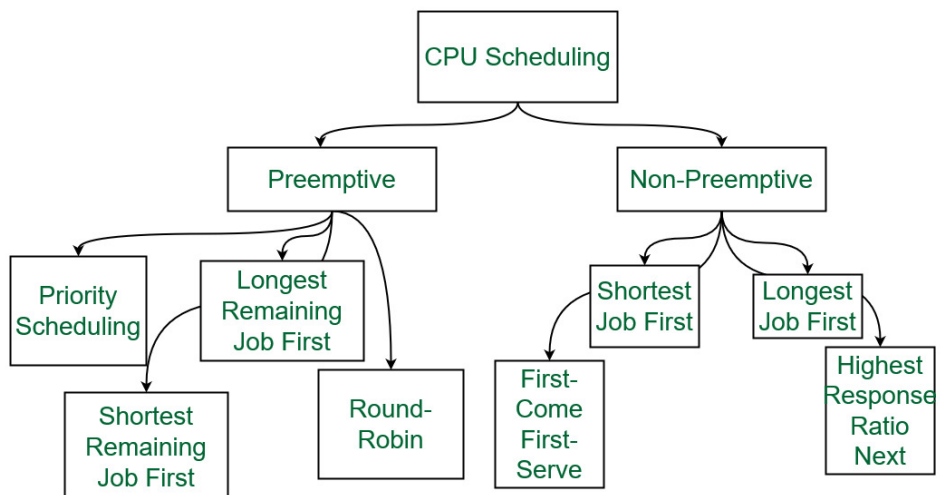
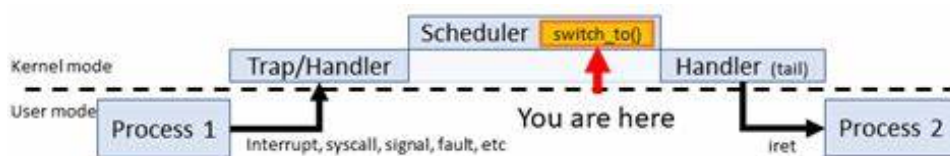
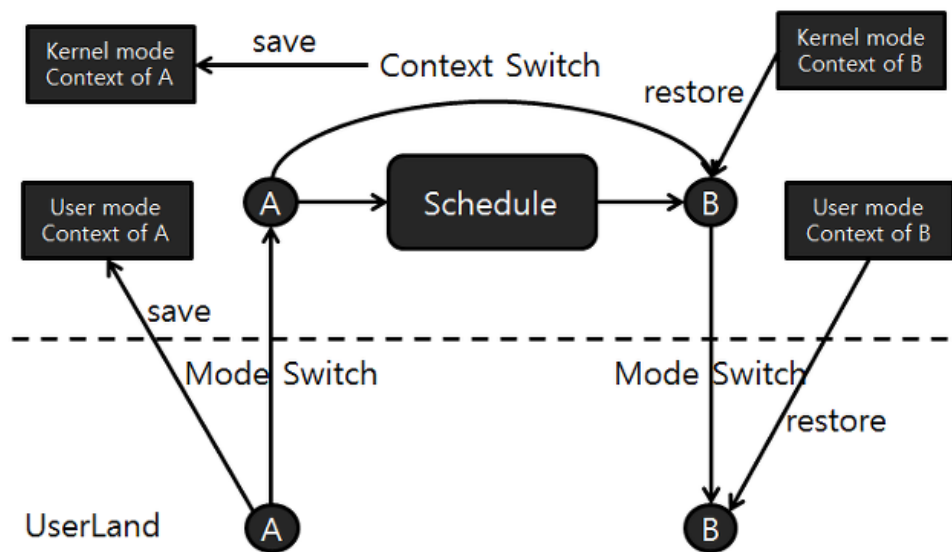
- Context Switch



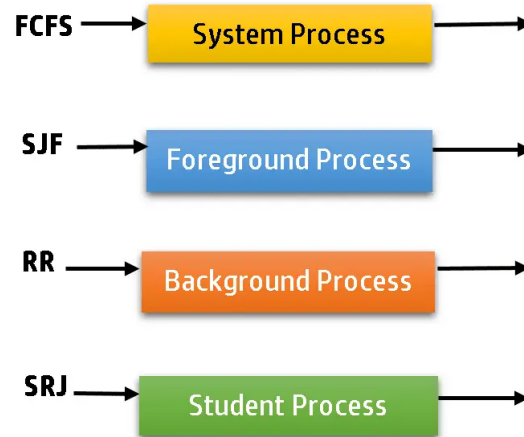
Context Switch

- **Context Switch:** Kernel saves the state of the old process and loads the saved state for the new process
- Context-switch time is purely **overhead**
- Switch time (about 1~1000 ms) depends on
 - > memory speed
 - > number of registers
 - > existence of special instructions
 - ◆ a single instruction to save/load all registers
 - > hardware support
 - ◆ **multiple sets of registers** (Sun UltraSPARC – a context switch means changing register file pointer)

Kernelland



Highest Priority



Lowest Priority

Created by Notes Jam

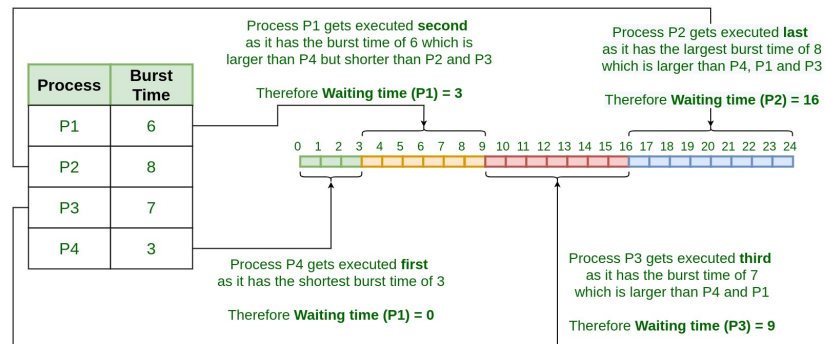
CPU SCHEDULING CRITERIA

- CPU Utilization: Percent of time that the CPU is busy executing a process.
- Throughput: Number of processes executed per unit time.
- Turnaround Time: The interval of time between submission of a process and its completion.
- Waiting Time: The amount of time the process spends in the ready queue waiting for the CPU.
- Response Time: The time between submission of requests and first response to the request.

Round Robin(RR)

- Each process is allotted a time slot(q). After this time has elapsed, the process is pre-empted and added to the end of the ready queue.
- Performance of the round robin algorithm
 - q large \rightarrow FCFS
 - q small $\rightarrow q$ must be greater than the context switch time; otherwise, the overhead is too high

Shortest Job First (SJF) Scheduling Algorithm

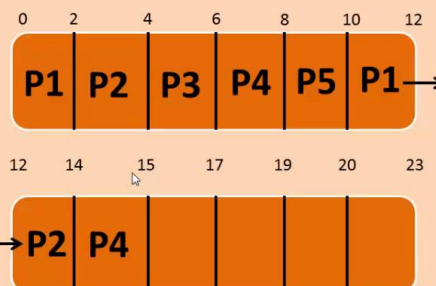


ROUND ROBIN SCHEDULING

PROCESSES	BURST TIME
P1	6 4 2
P2	5 3 1
P3	2
P4	3 1
P5	7 5

Given Time Quantum
2 units

Gantt Chart :



NOTE : Assume that all the Processes arrives at $t = 0$

CPU Scheduling (40 points)

Process	Burst Time	Priority	Arrival Time
P1	12	3	0
P2	6	4	2
P3	4	1	4
P4	18	2	6

Table 1: Process Information.

Consider the processes described in Table 1.

Questions: What is the average waiting time of those processes for each of the following scheduling algorithms? (Draw a Gantt chart for each algorithm.)

- (a) First Come First Serve (FCFS)
- (b) Non-preemptive Shortest Job First (NP-SJF)
- (c) Preemptive Shortest Job First (P-SJF)
- (d) Priority Scheduling
- (e) Round Robin, with the following assumptions:

Assumption (1). The scheduling time quantum is 5 time units.

Assumption (2). If a new process arrives at the same time as the time slice of the executing process expires, the OS puts the executing process in the ready queue, followed by the new process.

Question 3: Consider the following set of processes, their arrival times, length of the CPU burst time given in milliseconds:

Process	Arrival Time	Burst Time	Priority
P1	1	12	3
P2	2	3	1
P3	2	16	4
P4	3	10	2
P5	7	1	1

Let us schedule the execution of these processes using the following scheduling algorithms: First Come First Served (FCFS), Shortest Job First (SJF), Round Robin (RR), and a new type of scheduling algorithm called non-preemptive priority scheduling. The following are the assumptions:

- 1) Larger priority number implies higher priority
- 2) Assume the quantum (aka time slice) of 2 for RR algorithm
- 3) Non-Preemptive means once scheduled, a process cannot be preempted (i.e. it runs to completion).

3a [10 points] What is the response (completion) time of each process for each of the scheduling algorithms. Write the times in the table below.

	FCFS	SJF	Priority	RR
P1				
P2				
P3				
P4				
P5				

3b [10 points] Which of the above methods results in the longest average wait time. Show calculations and explain.

國立清華大學

Approximate Shortest-Job-First (SJF)

- SJF difficulty: no way to know length of the next CPU burst
- **Approximate SJF**: the next burst can be predicted as an **exponential average** of the measured length of previous CPU bursts

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n \leftarrow \text{history}$$

new one

Commonly,

$$= \alpha t_n + (1-\alpha)\alpha t_{n-1} + (1-\alpha)^2 \alpha t_{n-2} + \dots$$

$$\alpha = 1/2 \rightarrow = \left(\frac{1}{2}\right)t_n + \left(\frac{1}{2}\right)^2 t_{n-1} + \left(\frac{1}{2}\right)^3 t_{n-2} + \dots$$

Chapter5 Process Scheduling Operating System Concepts – NTHU LSA Lab 27

國立清華大學

Priority Scheduling

- A priority number is associated with each process
- The CPU is allocated to the highest priority process
 - Preemptive
 - Non-preemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem: starvation (low priority processes never execute)
 - e.g. IBM 7094 shutdown at 1973, a 1967-process never run
- **Solution: aging** (as time progresses increase the priority of processes)
 - e.g. increase priority by 1 every 15 minutes

Chapter5 Process Scheduling Operating System Concepts – NTHU LSA Lab 29

國立清華大學

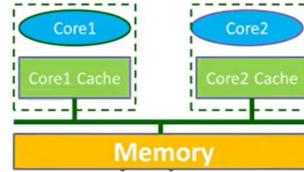
Evaluation Methods

- **Deterministic modeling** – takes a particular predetermined workload and defines the performance of each algorithm for that workload
 - Cannot be generalized
- **Queueing model** – mathematical analysis
- **Simulation** – random-number generator or trace tapes for workload generation
- **Implementation** – the only completely accurate for algorithm evaluation

Chapter5 Process Scheduling Operating System Concepts – NTHU LSA Lab

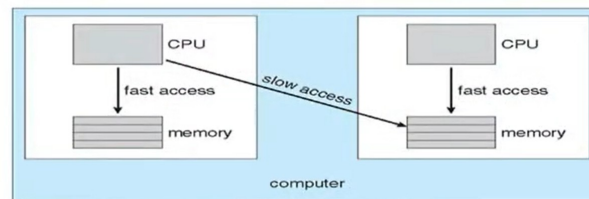
Processor affinity

- **Processor affinity:** a process has an affinity for the processor on which it is currently running
 - A process populates its recent used **data in cache memory of its running processor**
 - **Cache invalidation and repopulation has high cost**
- **Solution**
 - **soft affinity:**
 - ✦ possible to migrate between processors
 - **hard affinity:**
 - ✦ not to migrate to other processor



NUMA and CPU Scheduling

- **NUMA (non-uniform memory access):**
 - Occurs in systems containing combined CPU and memory boards
 - **CPU scheduler and memory-placement works together**
 - A process (assigned affinity to a CPU) can be allocated memory on the board where that CPU resides



Load-balancing

- Keep the workload evenly distributed across all processors
 - Only necessary on systems where each processor has its own private queue of eligible processes to execute
- Two strategies:
 - **Push migration:** move (push) processes from overloaded to idle or less-busy processor
 - **Pull migration:** idle processor pulls a waiting task from a busy processor
 - Often implemented in parallel
- Load balancing often counteracts the benefits of processor affinity

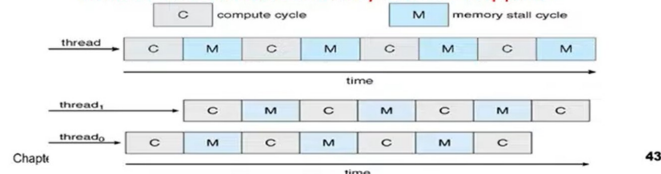
Multi-core Processor Scheduling

Multi-core Processor:

- Faster and consume less power
- **memory stall**: When access memory, it spends a significant amount of time waiting for the data become available. (e.g. cache miss)

Multi-threaded multi-core systems:

- Two (or more) hardware threads are assigned to each core (i.e. **Intel Hyper-threading**)
- Takes advantage of memory stall to make progress on another thread while memory retrieve happens



43

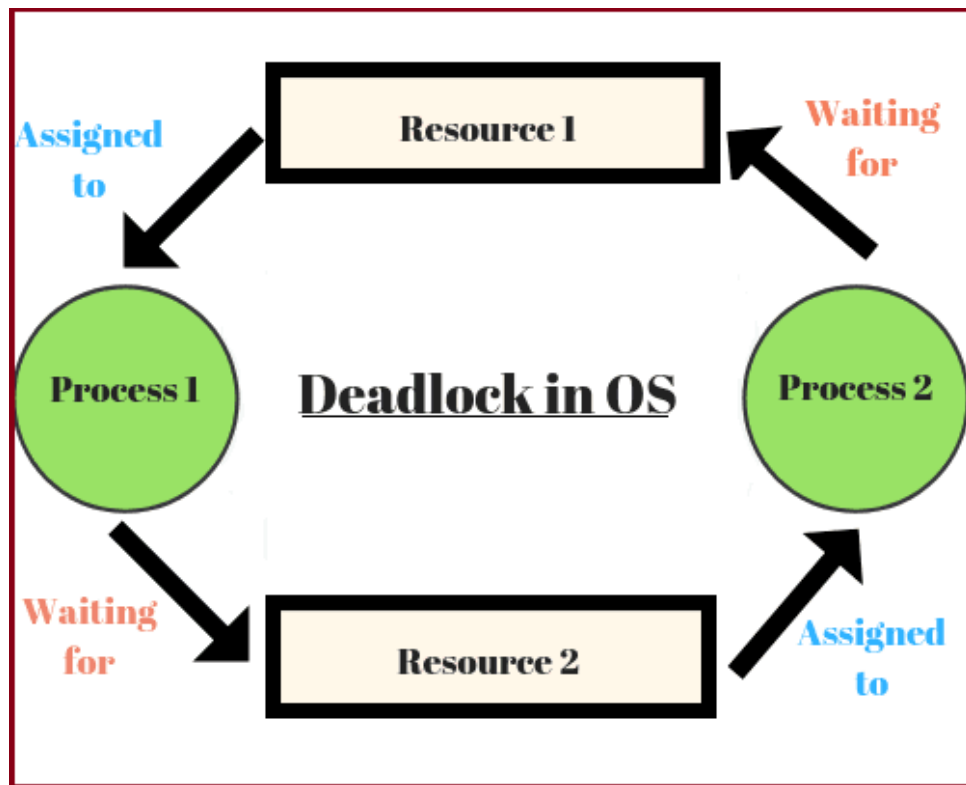
Real-Time Scheduling

- Real-time does not mean speed, but **keeping deadlines**
- Soft real-time requirements:
 - Missing the deadline is unwanted, but is not immediately critical
 - Examples: multimedia streaming
- Hard real-time requirements:
 - Missing the deadline results in **a fundamental failure**
 - Examples: nuclear power plant controller

Real-Time Scheduling Algorithms

- FCFS scheduling algorithm – Non-RTS
 - $T1 = (0, 4, 10) == (\text{Ready}, \text{Execution}, \text{Period})$
 - $T2 = (1, 2, 4)$
- Rate-Monotonic (RM) algorithm
 - Shorter period \rightarrow higher priority
 - Fixed-priority RTS scheduling algorithm
- Earliest-Deadline-First (EDF) algorithm
 - Earlier deadline \rightarrow higher priority
 - Dynamic priority algorithm

-Deadlock



Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_n is waiting for a resource that is held by P_0 .

7.5

- Resource Allocation Graph

Resource-Allocation Graph: Definition

A set of vertices V and a set of edges E .

- V is partitioned into two types:
 - $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system.
 - $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_1 \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

The resource-allocation graph is therefore a bipartite directed graph. What would be the graph representing the bridge-crossing example?



Resource Allocation Graph

Set of Vertices

 $V \rightarrow \begin{cases} \text{Set of Processes } \{P_1, P_2, P_3, \dots, P_n\} \\ \text{Set of Resources } \{R_1, R_2, R_3, \dots, R_m\} \end{cases}$

Set of Edges

 $E \rightarrow \begin{cases} P_i \rightarrow R_j \text{ (Request Edge)} \\ R_j \rightarrow P_i \text{ (Assignment Edge)} \end{cases}$

□ ⇒ Resource type
○ ⇒ Process

□ with dots ⇒ instance of resource type

Subscribe JENNY'S LECTURES CS/IT NET&JRF for full syllabus



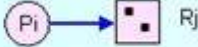
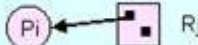
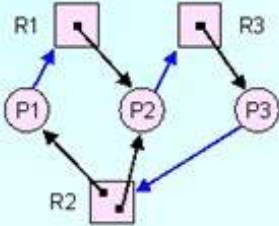
If no cycle in RAG, then no process in the system is deadlocked & if it contains cycle then deadlock may exist

Cycle + one instance of resource type
↓
Deadlock

Cycle + more than one instance
↓
Deadlock may exist

	Allocation			Request			Availability			
	P_1	P_2	P_3	R_1	R_2	R_3	R_1	R_2	R_3	
P_1	0	0	1	1	0	0	0	0	0	✓ P_1
P_2	1	0	1	0	1	0	0	0	0	✓ P_2
P_3	0	1	0	0	0	1	0	0	0	✓ P_3

Resource Allocation Graph

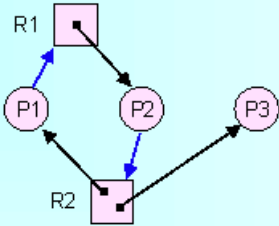
- Process 
- Resource type with 2 instances 
- P_i requests instance of R_j 
- An instance of R_j is allocated to P_i 
- A graph with a deadlock: 

Yair Amir

Fall 00 / Lecture 4

8

Resource Allocation Graph (cont.)

- A graph with a cycle but without a deadlock: 

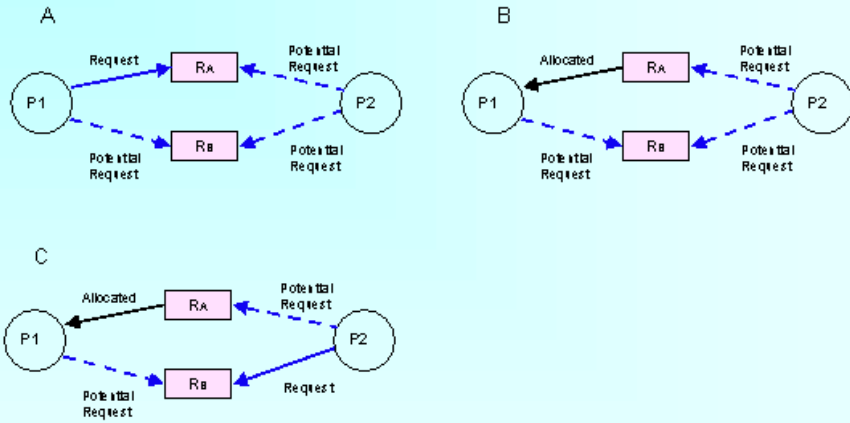
- If there are no cycles then there is no deadlock.
- If there is a cycle:
 - **If there is only one instance per resource type then there is a deadlock.**
 - **If there is more than one instance for some resource type, there may or may not be a deadlock.**

Yair Amir

Fall 00 / Lecture 4

9

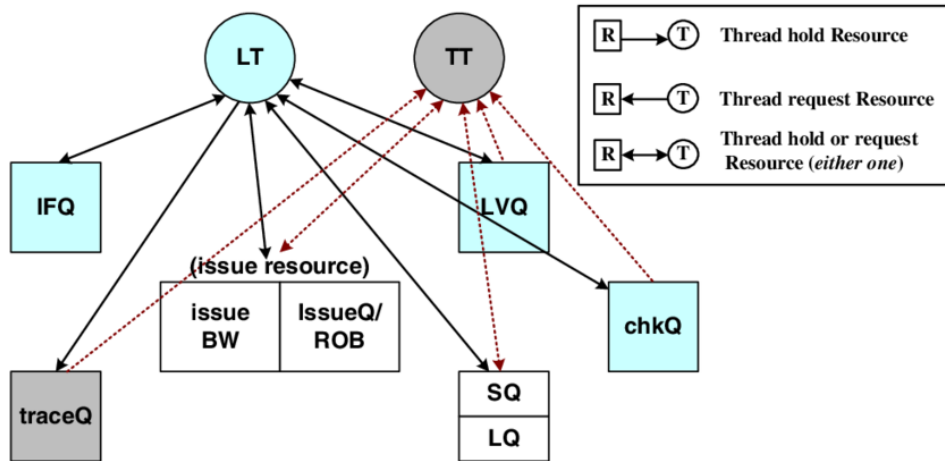
Resource Allocation Graph Algorithm (cont.)



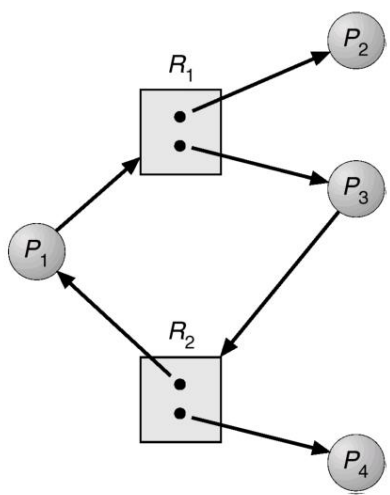
Yair Amir

Fall 00 / Lecture 4

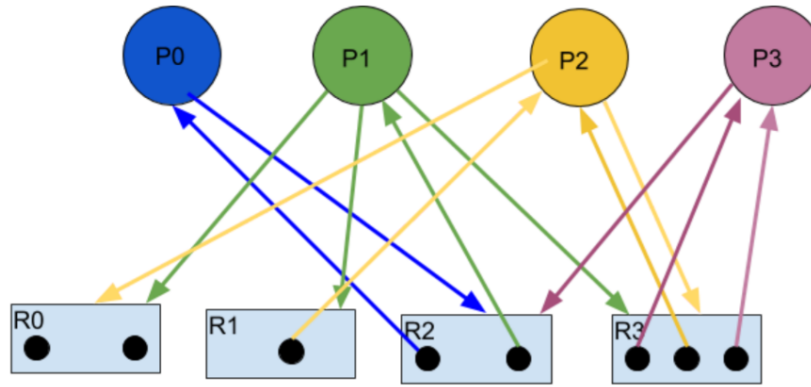
20



Resource Allocation Graph With A Cycle But No Deadlock



- If graph contains no cycles \Rightarrow no deadlock.
- If graph contains a cycle \Rightarrow
 - if only one instance per resource type, then deadlock.
 - if several instances per resource type, possibility of deadlock.



[6 pts] Fill in the resource matrix according to the graph above:

	Allocation				Need				Available			
	R0	R1	R2	R3	R0	R1	R2	R3	R0	R1	R2	R3
P0												
P1												
P2												
P3												

Approaches to Deadlock Prevention

Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Figure 6-14. Summary of approaches to deadlock prevention.

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639

Deadlock Avoidance

- Deadlock avoidance is a technique used to avoid deadlock.
- It requires information about how different processes would request different resources.
- **Safe state:** if deadlock not occur then safe state.
- **Unsafe state:** if deadlock occur then unsafe state.
- The idea of avoiding a deadlock is simply not allow the system to enter an unsafe state the may cause a deadlock.



3.5 Deadlock Detection

- If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur.
- In this environment, the system may provide:
 - An algorithm that examines the state of the system to determine whether a deadlock has occurred.
 - An algorithm to recover from the deadlock.
- Single Instance of Each Resource Type.
- Several Instances of a Resource Type.
- When should we invoke the detection algorithm?
- The answer depends on two factors:
 1. How *often* is a deadlock likely to occur?
 2. How *many* processes will be affected by deadlock when it happens?

20

The Difference Between Deadlock Prevention and Deadlock Avoidance

• Deadlock Prevention:



- Preventing deadlocks by constraining how requests for resources can be made in the system and how they are handled (system design).
- The goal is to ensure that at least one of the necessary conditions for deadlock can never hold.

• Deadlock Avoidance:



- The system dynamically considers every request and decides whether it is safe to grant it at this point.
- The system requires additional a priori information regarding the overall potential use of each resource for each process.
- Allows more concurrency.

Similar to the difference between a traffic light and a police officer directing traffic.

Comparison Deadlock	Starvation
<ul style="list-style-type: none"> • Definition • Other name • Arising conditions • Avoidance/prevention Techniques 	<ul style="list-style-type: none"> • Starvation occurs when a process waits for an indefinite period of time to get the resource it requires. Or Starvation is where low priority processes get blocked, and high priority process proceeds. • Lived lock • Uncontrolled management of resources, Process priorities being strictly enforces Use of random selection, Scarcity of resources
<ul style="list-style-type: none"> • Deadlock occurs when none of the processes in the set is able to move ahead due to occupancy of the required resources by some other process. Or Deadlock is where no process proceeds, and get blocked • Circular waiting • These four conditions arising simultaneously – mutual exclusion, hold and wait, no-preemption and circular wait • Infinite resources, Waiting is not allowed, Sharing is not allowed, Preempt the resources, All Requests made at the starting 	

BetweenMates.com

Handling Deadlocks

國立清華大學

- Ensure the system will **never** enter a **deadlock state**
 - **deadlock prevention**: ensure that at least one of the 4 **necessary conditions** cannot hold
 - **deadlock avoidance**: **dynamically** examines the resource-allocation state before allocation
- Allow to **enter a deadlock state** and **then recover**
 - **deadlock detection**
 - **deadlock recovery**
- **ignore the problem** and pretend that deadlocks never occur in the system
 - used by most operating systems, including UNIX.

Deadlock Recovery

國立清華大學

- Process termination
 - abort all deadlocked processes
 - abort 1 process at a time until the deadlock cycle is eliminated
 - ✦ which process should we abort first?
- Resource preemption
 - select a victim: which one to preempt?
 - rollback: partial rollback or total rollback?
 - starvation: can the same process be preempted always?

Bankers' Algorithm

The banker's algorithm

- A state is safe iff there exists a sequence $\{P_1..P_n\}$ where each P_i is allocated all of its needed resources to be run to completion
 - ◆ i.e.: we can always run all the processes to completion from a safe state
- The **safety algorithm** is the part that determines if a state is safe
- Initialization:
 - ◆ all processes are said to be “unfinished”
 - ◆ set the work vector to the amount resources available: $W(i) = V(i)$ for all i ;

41

Banker's Algorithm

1. Look for a new row in R which is smaller than A . If no such row exists the system will eventually deadlock → not safe.
2. If such a row exists, the process may finish. mark that process (row) as terminate and add all of its resources to A .
3. Repeat Steps 1 and 2 until all rows are marked → safe state
If some are not marked → not safe.

30

Example of Banker's Algorithm

	Allocation A B C
P0	0 1 0
P1	2 0 0
P2	3 0 2
P3	2 1 1
P4	0 0 2

	Need A B C
P0	7 4 3
P1	1 2 2
P2	6 0 0
P3	0 1 1
P4	4 3 1

Available A B C
3 3 2

Try to find a row in $Need_i$ that is \leq Available.

- P1. run completion. Available becomes = $[3\ 3\ 2] + [2\ 0\ 0] = [5\ 3\ 2]$
- P3. run completion. Available becomes = $[5\ 3\ 2] + [2\ 1\ 1] = [7\ 4\ 3]$
- P4. run completion. Available becomes = $[7\ 4\ 3] + [0\ 0\ 2] = [7\ 4\ 5]$
- P2. run completion. Available becomes = $[7\ 4\ 5] + [3\ 0\ 2] = [10\ 4\ 7]$
- P0. run completion. Available becomes = $[10\ 4\ 7] + [0\ 1\ 0] = [10\ 5\ 7]$

We found a sequence of execution: P1, P3, P4, P2, P0. State is safe

40

Banker's Algorithm for a single resource

<table border="1" style="margin: auto;"> <tr><td></td><td>Has</td><td>Max</td></tr> <tr><td>A</td><td>0</td><td>6</td></tr> <tr><td>B</td><td>0</td><td>5</td></tr> <tr><td>C</td><td>0</td><td>4</td></tr> <tr><td>D</td><td>0</td><td>7</td></tr> </table> <p>Free: 10 Any sequence finishes</p>		Has	Max	A	0	6	B	0	5	C	0	4	D	0	7	<table border="1" style="margin: auto;"> <tr><td></td><td>Has</td><td>Max</td></tr> <tr><td>A</td><td>1</td><td>6</td></tr> <tr><td>B</td><td>1</td><td>5</td></tr> <tr><td>C</td><td>2</td><td>4</td></tr> <tr><td>D</td><td>4</td><td>7</td></tr> </table> <p>Free: 2 C,B,A,D finishes</p>		Has	Max	A	1	6	B	1	5	C	2	4	D	4	7	<table border="1" style="margin: auto;"> <tr><td></td><td>Has</td><td>Max</td></tr> <tr><td>A</td><td>1</td><td>6</td></tr> <tr><td>B</td><td>2</td><td>5</td></tr> <tr><td>C</td><td>2</td><td>4</td></tr> <tr><td>D</td><td>4</td><td>7</td></tr> </table> <p>Free: 1 Deadlock (unsafe state)</p>		Has	Max	A	1	6	B	2	5	C	2	4	D	4	7
	Has	Max																																													
A	0	6																																													
B	0	5																																													
C	0	4																																													
D	0	7																																													
	Has	Max																																													
A	1	6																																													
B	1	5																																													
C	2	4																																													
D	4	7																																													
	Has	Max																																													
A	1	6																																													
B	2	5																																													
C	2	4																																													
D	4	7																																													

- Bankers' algorithm: before granting a request, ensure that a sequence exists that will allow all processes to complete
 - Use previous methods to find such a sequence
 - If a sequence exists, allow the requests
 - If there's no such sequence, deny the request
- Can be slow: must be done on each request!

1. Using the banker's algorithm, determine whether the following state is unsafe based on the snapshot of the system below. If the state is safe, provide a safe sequence of execution. Otherwise explain why it is unsafe. Show your calculations for full credit.

[10 points]

	<u>Allocation</u>	<u>Max (Demand)</u>	<u>Available</u> = (1, 0, 0, 2)
	A B C D	A B C D	
P_0	3 0 1 4	5 1 1 7	
P_1	2 2 1 0	3 2 1 1	
P_2	3 1 2 1	3 3 2 1	
P_3	0 5 1 0	4 6 1 2	
P_4	4 2 1 2	6 3 2 5	

Sample question (bankers algorithm)

	Allocation	Max	Need	Available
	A B C D	A B C D	A B C D	3 2 2 1
P0	4 0 0 1	7 0 2 1		
P1	1 1 0 0	1 6 5 0		
P2	1 0 4 5	3 3 4 6		
P3	0 4 2 1	1 5 6 2		
P4	0 3 1 2	2 4 3 2		

Using Bankers algorithm answer the following:

1. How many resources of type A, B, C and D are there?
2. What are the contents of the Need matrix?
3. Is the system in a safe state? Provide reasoning for your answer (show the sequence in which the processes would finish)
4. If a request from process P2 arrives for additional resources of {0, 2, 0, 0}, can the Bankers algorithm grant the request immediately? Provide reasoning for your answer. (14 Marks)

Q1. **Deadlocks.** The Banker's algorithm is used for deadlock avoidance. Consider the state of resource availability and allocation defined by the following matrices.

Claim Matrix

	R1	R2	R3
P1	3	1	4
P2	6	1	3
P3	3	2	2
P4	4	2	2

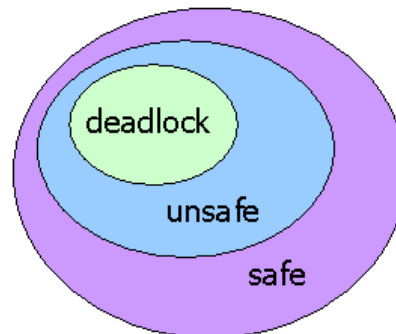
Allocation Matrix

	R1	R2	R3
P1	2	1	1
P2	5	1	1
P3	2	0	1
P4	0	0	2

- (1) Assuming that the total amounts for resources R1, R2, and R3 are 10, 2, and 10, should a new request to the Banker's algorithm by process P3 to acquire one additional resource from R1 and one additional resource from R3 be approved or denied? Explain why or why not.
- (2) Assuming that the total amounts for resources R1, R2, and R3 are 10, 2, and 10, should a new request to the Banker's algorithm by process P4 to acquire one additional resource from R3 be approved or denied? Explain why or why not.
- (3) Assuming that the total amounts for resources R1 and R2 are 10 and 2, what is the minimum amount for resource R3 that would render the above state a safe state under the Banker's algorithm?
- (4) Given your answer for part (3) what are all the possible orderings for the four processes P1, P2, P3, and P4 to complete their execution subject to the Banker's algorithm.
- (5) Assuming that the total amounts for resources R1 and R2 are 10 and 2, what is the minimum amount for resource R3 that would make it possible for the Banker's algorithm to allow process P1 to complete its execution before all other three processes?

The Banker's Algorithm

- Idea: know what each process *might* ask for
- Only make allocations that leave the system in a *safe* state
- Inefficient



Resource allocation state space

Jenny's Lectures
Computer Science / Information Technology

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P ₁	1	0	0	0	1	7	5	0	0	7	5	0	0	7	5	0
P ₂	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P ₃	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0
P ₄	0	0	1	4	0	6	5	6	2	14	12	12	0	6	4	2
	2	9	10	12					3	14	12	12				

@ Need Matrix?
 @ Is system in safe state?
 if yes then find safe sequence.

$$\text{Total} = \begin{matrix} A & B & C & D \\ \hline 3 & 14 & 12 & 12 \end{matrix}$$

$$\text{Need}_i = \text{Allocation}_i - \text{Max}_i$$

Jenny's Lectures
Computer Science / Information Technology

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P ₁	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P ₂	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P ₃	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0
P ₄	0	0	1	4	0	6	5	6	2	14	12	12	0	6	4	2
	2	9	10	12					3	14	12	12				

@ Need Matrix?
 @ Is system in safe state? **Yes**
 if yes then find safe sequence.

$$\text{Total} = \begin{matrix} A & B & C & D \\ \hline 3 & 14 & 12 & 12 \end{matrix}$$

$$\text{Need}_i = \text{Allocation}_i - \text{Max}_i$$

Sequence: - P₀ P₂ P₃ P₄ P₁

Jenny's Lectures
Computer Science / Information Technology

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P ₁	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P ₂	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P ₃	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0
P ₄	0	0	1	4	0	6	5	6	2	14	12	12	0	6	4	2
	2	9	10	12					3	14	12	12				

otherwise goto step 4
 Step 4:- if flag[i]=0 for all i then system is in safe state otherwise unsafe state.

$$\text{Total} = \begin{matrix} A & B & C & D \\ \hline 3 & 14 & 12 & 12 \end{matrix}$$

$$\text{Need}_i = \text{Allocation}_i - \text{Max}_i$$

also: Input:- Processes

- any 2 out of 3 (Max Need, Allocation)
- available or total no. of resources

flag[i]=0 for i=0 to (n-1) & find Need[n][m] = Max[n][m] - allocation[n][m]

find a process P_i such that: - flag[i]=0 & Need_i <= Available

if such i exists then flag[i]=1, available = available + Allocate; goto step 1

n = no. of processes
 m = no. of resources

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	2	0	0	1	4	2	1	2	3	3	2	1	2	2	1	1
P ₁	3	1	2	1	5	2	5	2					2	1	3	1
P ₂	2	1	0	3	2	3	1	6					0	2	1	3
P ₃	1	3	1	2	1	4	2	4					0	1	1	2
P ₄	1	4	3	2	3	6	6	5					2	2	3	3
	9	9	6	9												

Resource-Request algo:-

Step 1:- if Request_i ≤ Need_i; then go to step 2
 otherwise error

Step 2:- if Request_i ≤ Available then go to step 3
 otherwise P_i will wait

Step 3:- System pretend as if request has been granted by modifying the state as follows:-

- Available -= Request_i
- Allocation_i += Request_i
- Need_i -= Request_i

① Need Matrix? Yes

② Is system in safe state? if Yes find safe sequence

③ if request from P₁ arrives for (1,1,0,0) can request be immediately granted? Yes

④ if request from P₄ arrives for (0,0,2,0) can it be immediately granted? No

If modified resource-allocation state is safe then request granted otherwise P_i will wait & old allocation state is restored

Deadlock Detection & Recovery

- Allow the system to enter into deadlocked state
- Deadlock detection algorithms (2 types)

Recovery techniques

- Single instance (Wait-for graph)
 - (Detect cycle)
- Multiple instances (Banker's algo)

Subscribe JENNY'S LECTURES CS/IT NET&JRF for full syllabus

Deadlock Recovery

① Optimistic approach (Preemption of Resources & Processes)

↳ Pre-empt some resources from process & give these resources to other processes until the deadlock cycle is broken

selecting a victim } → Priority of process

Rollback } → How long the process has computed?

Starvation } → how much longer a process will compute before completion

→ How many & what type of resources process has used

→ How many resources the process needs to complete its execution etc.

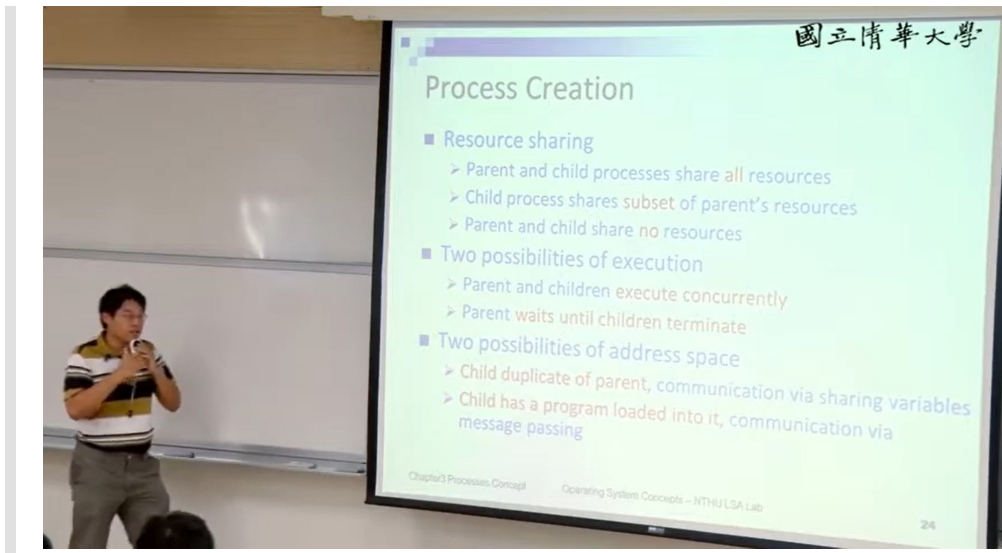
② Pessimistic Approach (Process Termination)

↳ Abort all deadlocked processes *costly*

↳ Abort one process at a time and decide next to abort after deadlock detection

↓ overhead of calling detection algo again & again

Process Create



國立清華大學

UNIX/Linux Process Creation

- **fork** system call
 - Create a new (child) process
 - The new process **duplicates the address space** of its parent
 - Child & Parent **execute concurrently** after fork
 - Child: return value of fork is 0
 - Parent: return value of fork is PID of the child process
- **execlp** system call
 - Load a new binary file into memory – **destroying the old code**
- **wait** system call
 - The parent waits for **one of its child processes** to complete

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 25

國立清華大學

UNIX/Linux Process Creation

- **Memory space of fork():**
 - Old implementation: A's child is an **exact copy** of parent
 - Current implementation: use **copy-on-write** technique to store **differences in A's child address space**

free memory
B
free memory
A
kernel

Originally

free memory
A's child
B
free memory
A
kernel

After A does an **fork**

free memory
C
B
free memory
A
kernel

After the child does an **execlp**

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 26

UNIX/Linux Example

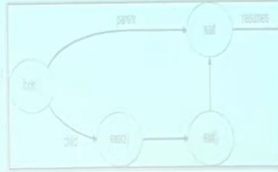
```

#include <stdio.h>
void main ( )
{
  int A;
  /* fork another process */
  A = fork( );

  if (A == 0) { /* child process */
    printf("this is from child process\n");
    execlp("/bin/ls" "ls", NULL);
  } else { /* parent process */
    printf("this is from parent process\n");
    int pid = wait(&status);
    printf("Child %d completes" pid);
  }
  printf("process ends %d\n", A);
}

```

Output:
 this is from child process
 this is from parent process
 a.out hello.c readme.txt
 Child 32185 completes
 process ends 32185



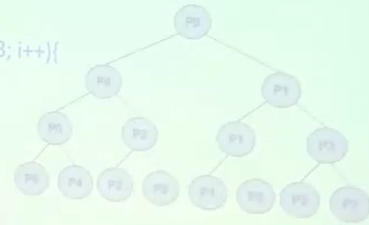
Example Quiz:

■ How many processes are created?

```

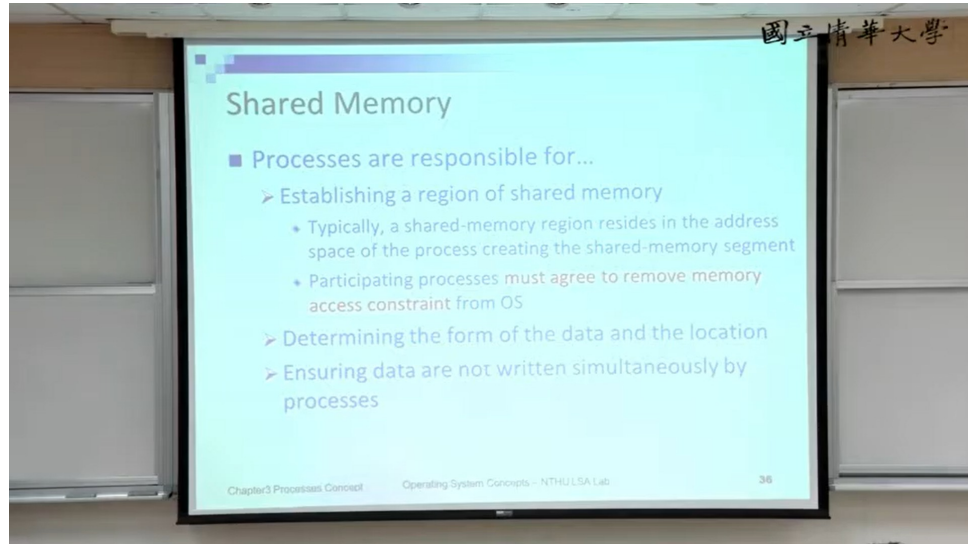
#include <stdio.h>
#include <unistd.h>
int main() {
  for (int i=0; i<3; i++){
    fork();
  }
  return 0;
}

```



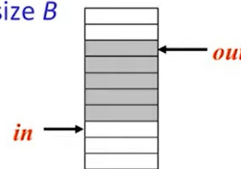
Process Termination

- Terminate when the last statement is executed or **exit()** is called
 - > All resources of the process, including physical & virtual memory, open files, I/O buffers, are deallocated by the OS
- Parent may terminate execution of children processes by specifying its PID (**abort**)
 - > Child has exceeded allocated resources
 - > Task assigned to child is no longer required
- Cascading termination:
 - > killing (exiting) parent → killing (exiting) all its children



Consumer & Producer Problem

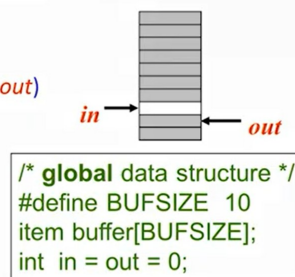
- **Producer** process produces information that is consumed by a **Consumer** process
- Buffer as a circular array with size B
 - next free: in
 - first available: out
 - empty: $in = out$
 - full: $(in+1) \% B = out$

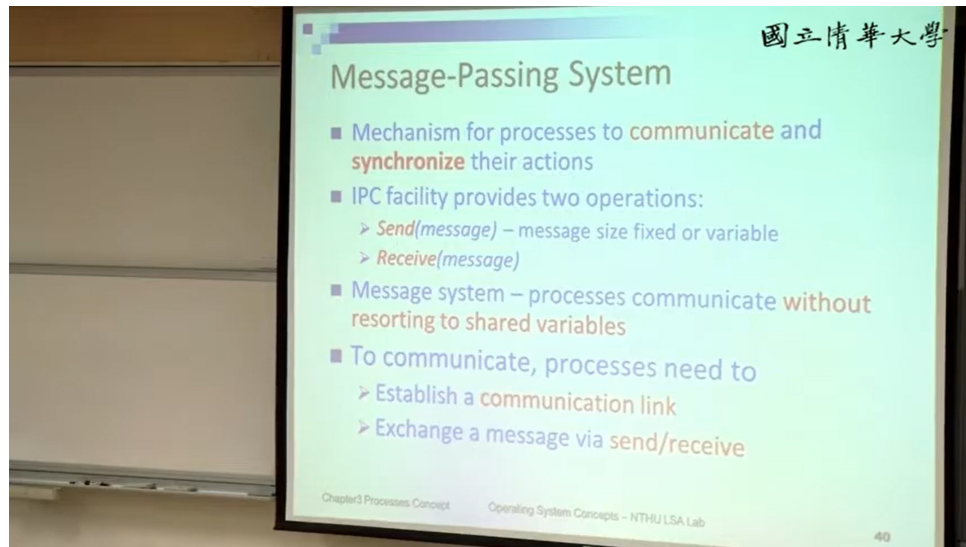


Shared-Memory Solution

```

/*producer*/
while (1) {
    while (((in + 1) % BUFFER_SIZE) == out)
        ; //wait if buffer is full
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
/*consumer*/
while (1) {
    while (in == out); //wait if buffer is empty
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
}
    
```

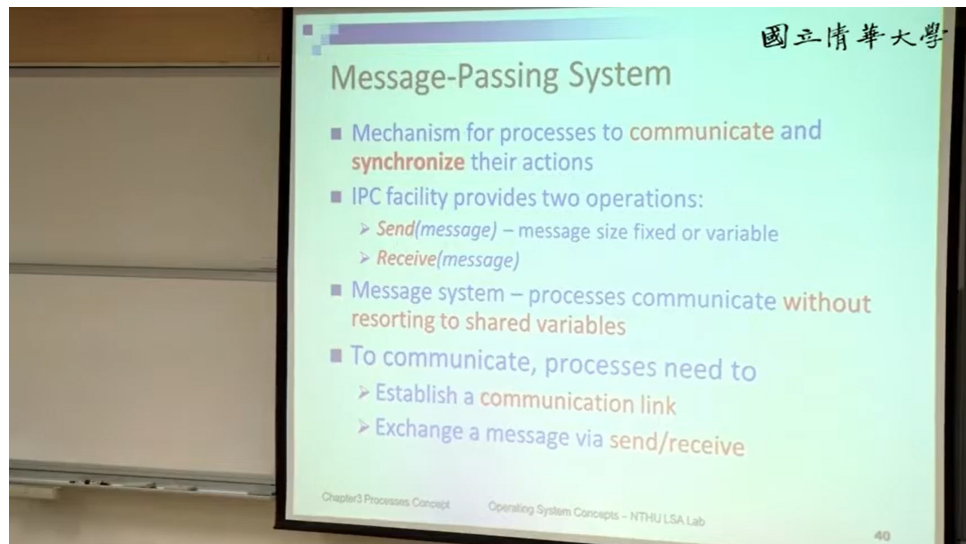




國立清華大學

Message-Passing System

- Implementation of communication link
 - physical (e.g., shared memory, HW bus, or network)
 - logical (e.g., **logical properties**)
 - ✦ **Direct or indirect communication**
 - ✦ Symmetric or asymmetric communication
 - ✦ **Blocking or non-blocking**
 - ✦ Automatic or explicit buffering
 - ✦ Send by copy or send by reference
 - ✦ Fixed-sized or variable-sized messages

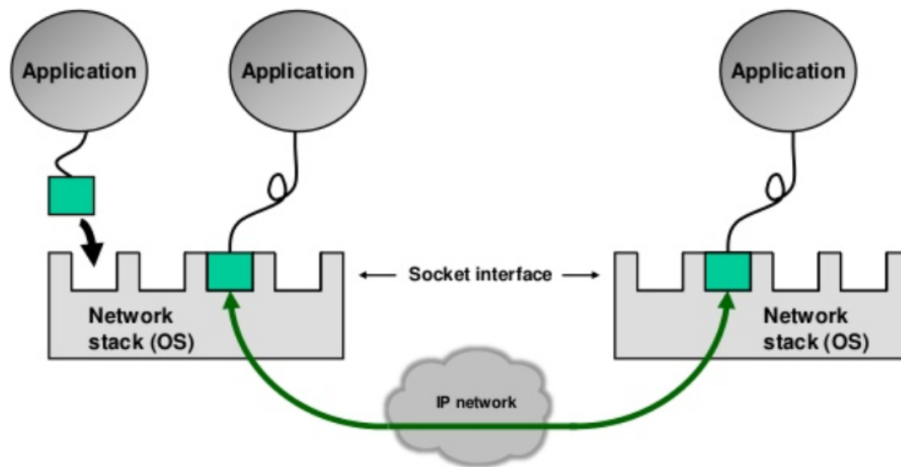


國立清華大學

Communication Methods

- Sockets:
 - A network connection identified by IP & port
 - Exchange unstructured stream of bytes
- Remote Procedure Calls:
 - Cause a procedure to execute in another address space
 - Parameters and return values are passed by message

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 34

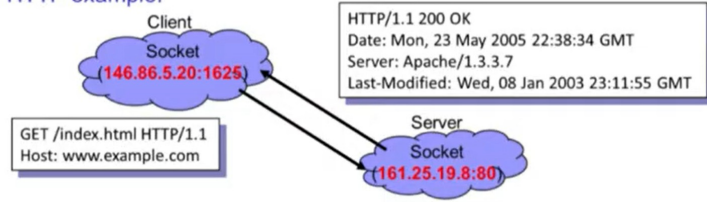


國立清華大學

Sockets

- Considered as a low-level form of communication **unstructured stream of bytes** to be exchanged
- **Data parsing responsibility falls upon the server and the client applications**

HTTP example:



國立清華大學

Sockets

- A socket is identified by a concatenation of IP address and port number
- Communication consists between a pair of sockets
- Use 127.0.0.1 to refer itself

```

Server (161.25.19.8)
  socket()
  bind()
  listen()
  accept()
  read()
  write()
  close()

Client (146.86.5.20)
  socket()
  connect()
  write()
  read()
  close()

Well-known port 161.25.19.8:80
Assign port 146.86.5.20:1625
Block until client requests
Data req.
Data reply
    
```

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 48

國立清華大學

Remote Procedure Calls: RPC

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
 - allows programs to call procedures located on other machines (and other processes)
- Stubs – client-side proxy for the actual procedure on the server

```

client
  val = server.someMethod(A,B)
  stub

remote object
  boolean someMethod (Object x, Object y)
  {
    implementation of someMethod
  }
  skeleton

A, B, someMethod
boolean return value
    
```

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 50

國立清華大學

Client and Server Stubs

Client stub:

- Packs parameters into a message (i.e. parameter marshaling)
- Calls OS to send directly to the server
- Waits for result-return from the server

```

Client
  Call remote procedure
  Wait for result
  Return from call

Server
  Request
  Reply
  Call local procedure and return results

Time
    
```

Server stub:

- Receives a call from a client
- Unpacks the parameters
- Calls the corresponding procedure
- Returns results to the caller

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 51

Threads

Thread Control Block

- **Thread Control Block (TCB)** is a data structure in the operating system kernel which contains thread-specific information needed to manage it. The TCB is "the manifestation of a thread in an operating system".
 - An example of information contained within a TCB is:
 - Stack pointer: Points to thread's stack in the process
 - Program counter
 - State of the thread (running, ready, waiting, start, done)
 - Thread's register values
 - Pointer to the Process control block (PCB) of the process that the thread lives on
- The Thread Control Block acts as a library of information about the threads in a system. Specific information is stored in the thread control block highlighting important information about each thread.

國立清華大學

Threads

- A.k.a **lightweight process**: basic unit of CPU utilization
- All threads belonging to the same process share
 - code section, data section, and OS resources (e.g. open files and signals)
- But each thread has its own
 - thread ID, program counter, register set, and a stack

The diagram illustrates the difference between single-threaded and multithreaded processes. On the left, a 'single-threaded' process is shown with a single thread pointing to a shared set of resources: code, data, files, registers, and stack. On the right, a 'multithreaded' process is shown with three threads, each having its own registers and stack, but sharing the code, data, and files sections. A red box highlights the shared code, data, and files sections in the multithreaded process.

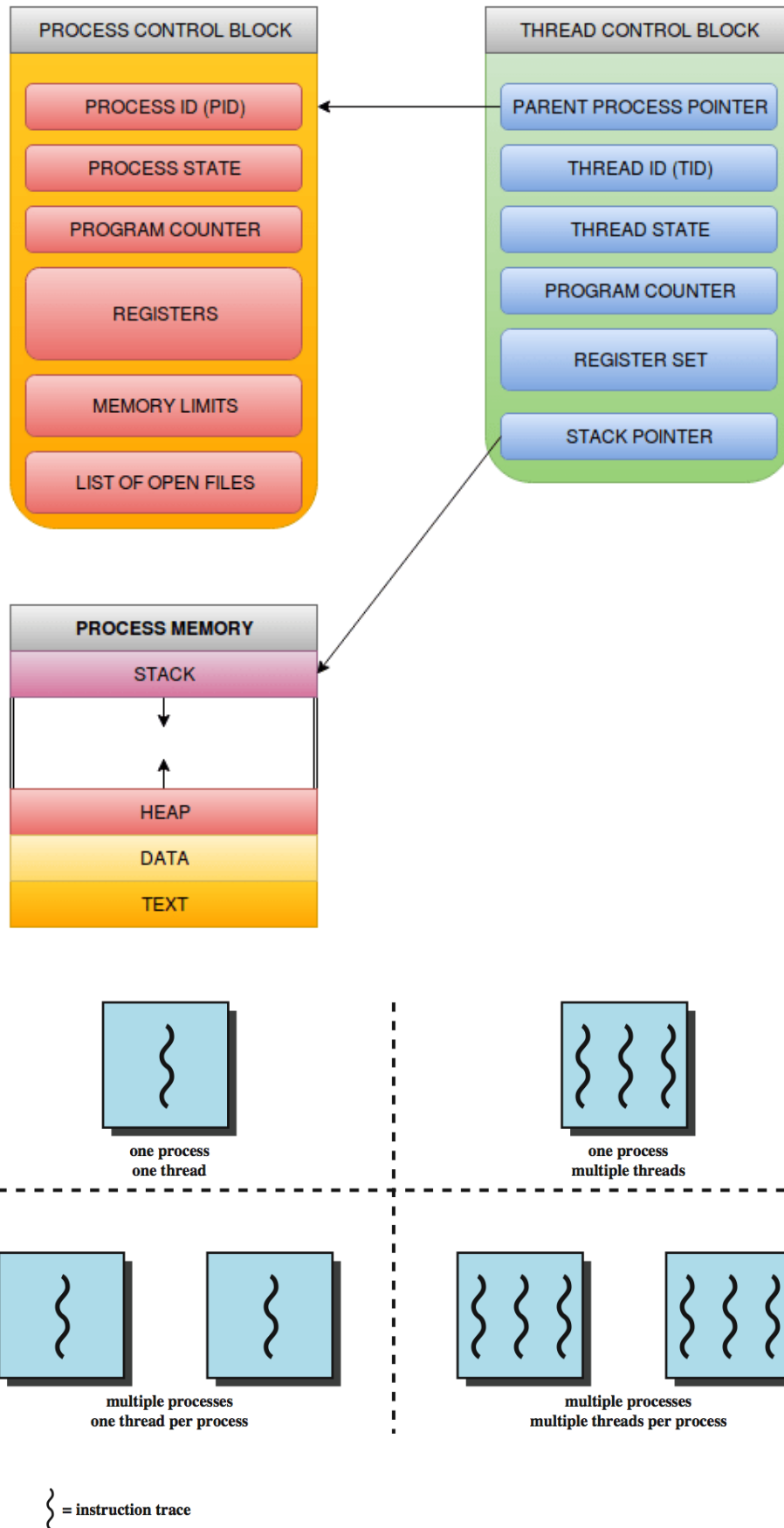
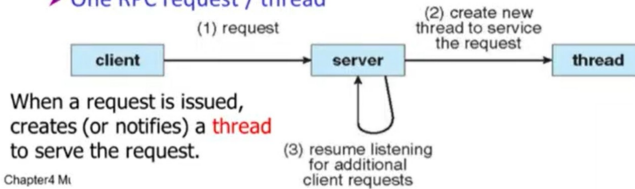


Figure 4.1 Threads and Processes [ANDE97]

Thread	Process
It is lightweight entity.	It is heavy weight entity
If a thread ends working process keep working.	Process may keep working and if a process terminates all its threads will terminate also.
Communication b/w threads happens via memory.	Communication b/w Process happens via OS.
The Creation of thread and context switching is inexpensive	The creation of the process is expensive.

Motivation

- **Example: a web browser**
 - One thread displays contents while the other thread receives data from network
- **Example: a web server**
 - One request / process: poor performance
 - One request / thread: better performance as code and resource sharing
- **Example: RPC server**
 - One RPC request / thread



Benefits of Multithreading

- **Responsiveness:** allow a program to continue running even if part of it is blocked or is performing a lengthy operation
 - **Resource sharing:** several different threads of activity all within **the same address space**
 - **Utilization of MP arch.:** Several thread may be **running in parallel** on different processors
 - **Economy:** Allocating memory and resources for process creation is costly. In Solaris, creating a process is about **30 times slower than is creating a thread**, and **context switching is about five times slower**. A register set switch is still required, but **no memory-management related work is needed**
- Chapter4 Multithreaded Operating System Concepts – NTHU LSA Lab 5

Challenges in Multicore Programming

- **Dividing activities:** divide program into concurrent tasks
 - **Data splitting:** divide data accessed and manipulated by the tasks
 - **Data dependency:** synchronize data access
 - **Balance:** evenly distribute tasks to cores
 - **Testing and debugging**
- Chapter4 Multithreaded Operating System Concepts – NTHU LSA Lab 8

User vs. Kernel Threads

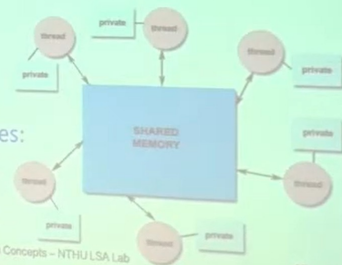
- User threads – thread management done by **user-level threads library**
 - POSIX Pthreads
 - Win32 threads
 - Java threads
- Kernel threads – supported by the **kernel (OS) directly**
 - Windows 2000 (NT)
 - Solaris
 - Linux
 - Tru64 UNIX

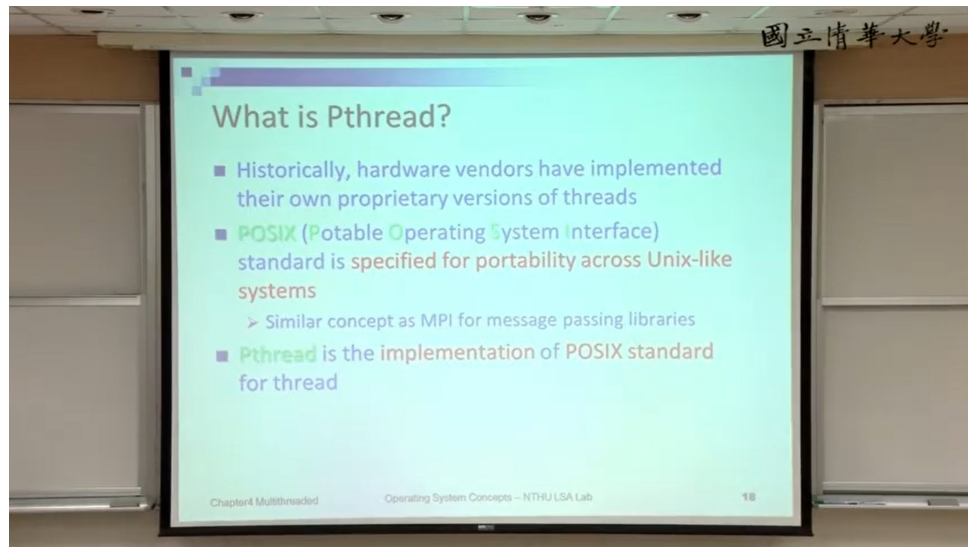
User vs. Kernel Threads

- User threads
 - **Thread library** provides support for thread creation, scheduling, and deletion
 - Generally **fast** to create and manage
 - **If the kernel is single-threaded, a user-thread blocks → entire process blocks** even if other threads are ready to run
- Kernel threads
 - The **kernel** performs thread creation, scheduling, etc.
 - Generally **slower** to create and manage
 - If a thread is blocked, the kernel can schedule another thread for execution

Shared-Memory Programming

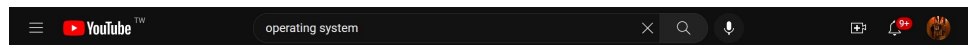
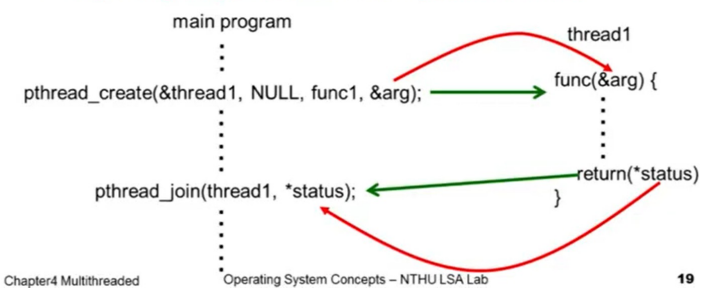
- **Definition:** Processes communicate or work together with each other through a **shared memory space** which can be accessed by all processes
 - Faster & more efficient than message passing
- Many issues as well:
 - Synchronization
 - Deadlock
 - Cache coherence
- Programming techniques:
 - Parallelizing compiler
 - Unix processes
 - Threads (**Pthread**, Java)





Pthread Creation

- `pthread_create(thread,attr,routine,arg)`
 - **thread**: An **unique identifier** (token) for the new thread
 - **attr**: It is used to set **thread attributes**. NULL for the default value:
 - **routine**: The routine that the thread will execute once it is created
 - **arg**: A **single argument** that may be **passed to routine**

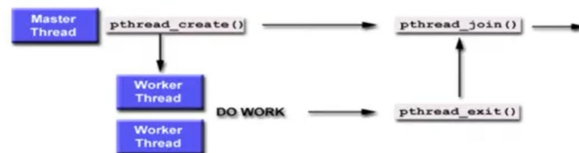


Pthread Joining & Detaching

- `pthread_join(threadId, status)`
 - **Blocks** until the specified **threadId** thread terminates
 - One way to **accomplish synchronization** between threads
 - Example: to create a pthread barrier

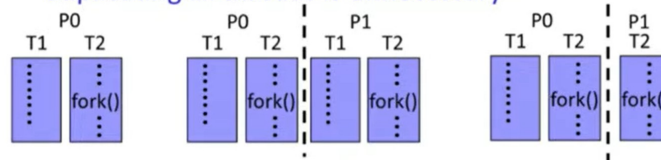

```
for (int i=0; i<n; i++) pthread_join(thread[i], NULL);
```

- `pthread_detach(threadId)`
 - Once a thread is **detached**, it can **never** be joined
 - Detach a thread could **free some system resources**



Semantics of fork() and exec()

- Does **fork()** duplicate only the calling thread or all threads?
 - Some UNIX system support two versions of **fork()**
- **execlp()** works the same; **replace the entire process**
 - If **exec()** is called immediately after forking, then duplicating all threads is unnecessary



Chapter4 Multithreaded

Operating System Concepts – NTHU LSA Lab

26

Signal Handling

- Signals (**synchronous** or **asynchronous**) are used in UN systems to notify a process that an event has occurred
 - Synchronous: illegal memory access
 - Asynchronous: <control-C>
- A **signal handler** is used to process signals
 1. Signal is generated by particular event
 2. Signal is delivered to a process
 3. Signal is handled
- Options
 - Deliver the signal to the thread to which the signal applies
 - Deliver the signal to every thread in the process
 - Deliver the signal to certain threads in the process
 - Assign a specific thread to receive all signals for the process

Chapter4 Multithreaded

Operating System Concepts – NTHU LSA Lab

28

Thread Pools

- Create a number of threads in a pool where they await work
- Advantages
 - Usually slightly **faster to service a request** with an existing thread **than create a new thread**
 - Allows the number of threads in the application(s) to be **bound to the size of the pool**
- **# of threads**: # of CPUs, expected # of requests, amount of physical memory

Chapter4 Multithreaded

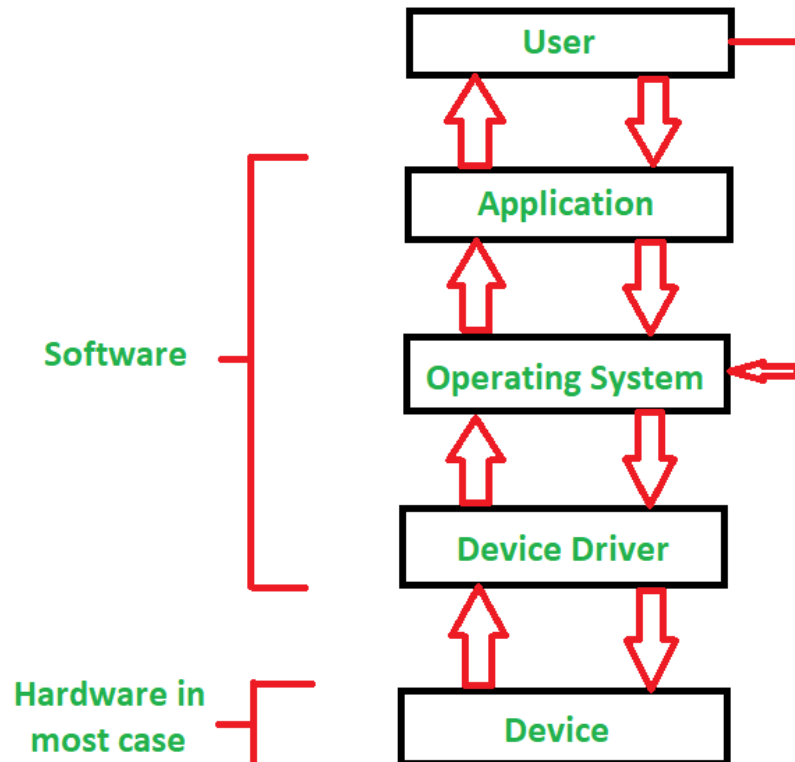
Operating System Concepts – NTHU LSA Lab

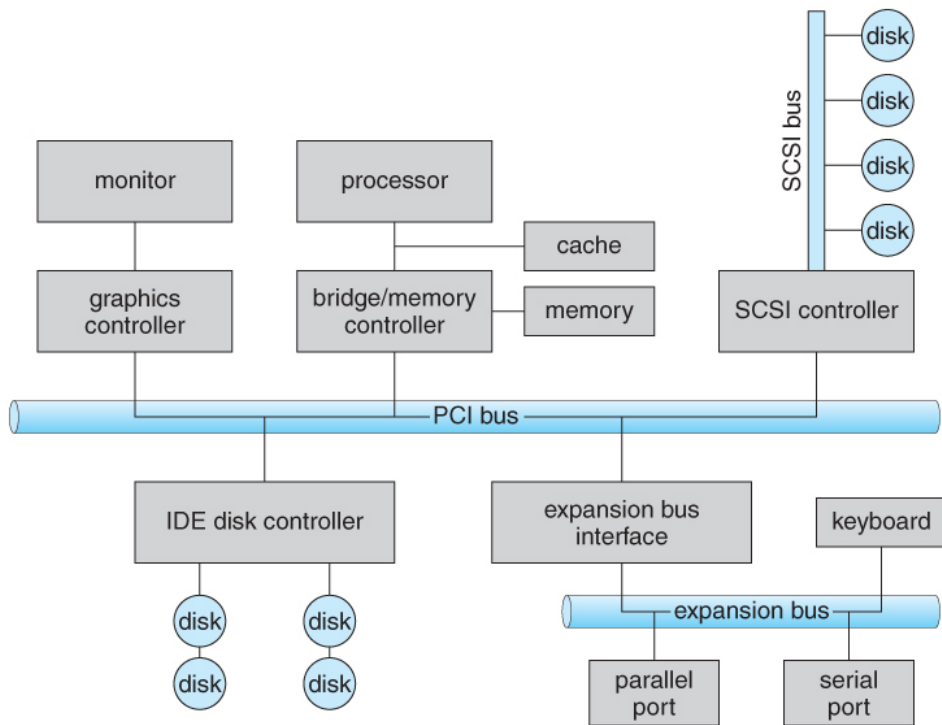
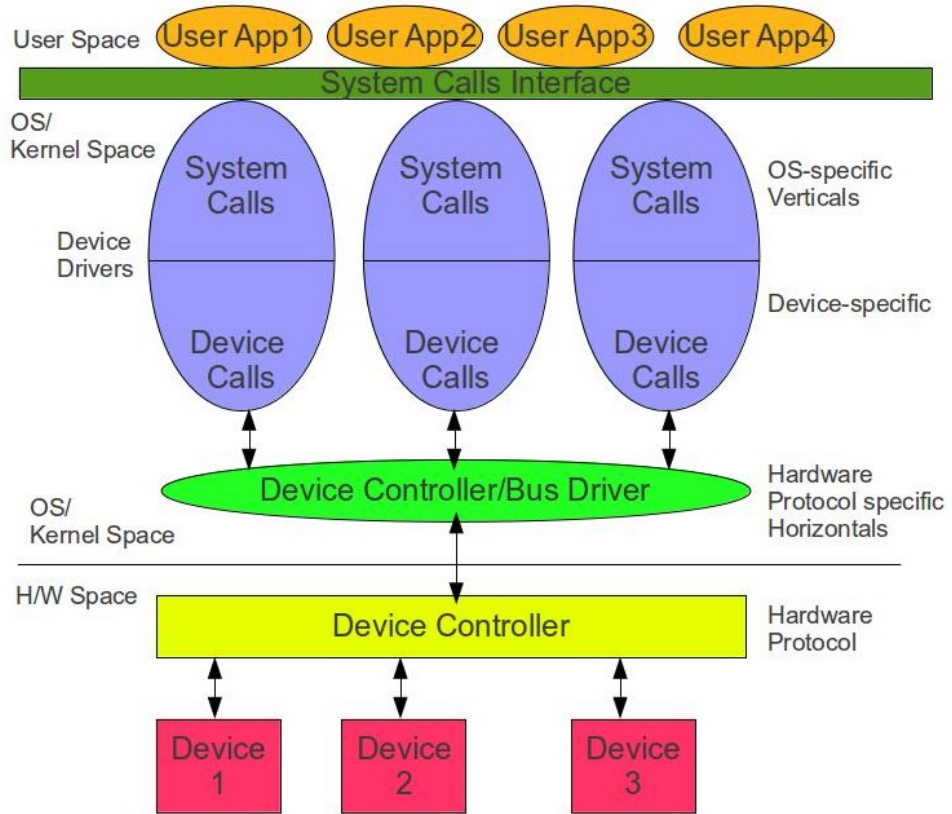
29

I/O System Mangement

I/O Hardware

- Computers support a wide variety of I/O devices, but common concepts apply to all:
 - **Port**: connection point for a device
 - **Bus**: set of wires that connects to many devices, with a protocol specifying how information can be transmitted over the wires
 - **Controller**: a chip (or part of a chip) that operates a port, a bus, or a device
- How can the processor communicate with a device?
 - Special instructions allow the processor to transfer data and commands to registers on the device controller
 - The device controller may support **memory-mapped I/O**:
 - The same address bus is used for both memory and device access.
 - The same CPU instructions can access memory locations or devices.





I/O Methods Categorization 國立清華大學

- Depending on how to address a device:
 - **Port-mapped I/O**
 - ✦ Use different address space from memory
 - ✦ Access by special I/O instruction (e.g. **IN**, **OUT**)
 - **Memory-mapped I/O**
 - ✦ Reserve specific memory space for device
 - ✦ Access by standard data-transfer instruction (e.g. **MOV**)
 - ☺ More efficient for large memory I/O (e.g. graphic card)
 - ☹ Vulnerable to accidental modification, error

I/O Methods Categorization 國立清華大學

- Depending on how to interact with a device:
 - **Poll** (busy-waiting): processor periodically check status register of a device
 - **Interrupt**: device notify processor of its completion
- Depending on who to control the transfer:
 - **Programmed I/O**: transfer controlled by CPU
 - **Direct memory access (DMA) I/O**: controlled by **DMA controller** (a special purpose controller)
 - ✦ Design for large data transfer
 - ✦ Commonly used with memory-mapped I/O and interrupt I/O method

Chapter 13 I/O Systems Operating System Concepts - NTHU, USA Lab 10

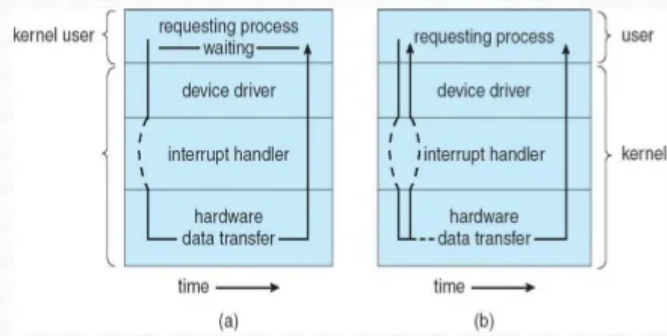
IO System Performance

Common Concepts in I/O Hardware

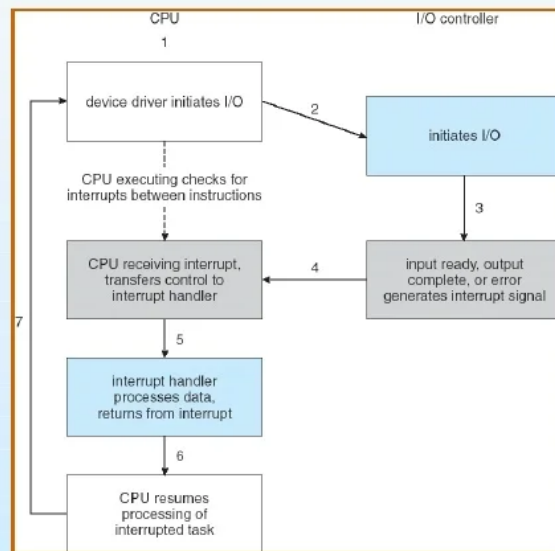
- ▶ **Common concepts**: signals from I/O devices interface with computer.
- ▶ **Port**: connection point for device
- ▶ **Bus**: set of wires and a protocol that specifies a set of messages that can be sent on the wires.
- ▶ **Controller**: a collection of electronics that can operate a port, a bus, or a device.
 - Sometimes integrated and sometimes separate circuit board (host adapter)
 - Contains processor, microcode, private memory, bus controller, etc

Amir H. Payberah (Tehran Polytechnic) I/O Systems 1393/9/15 6 / 57

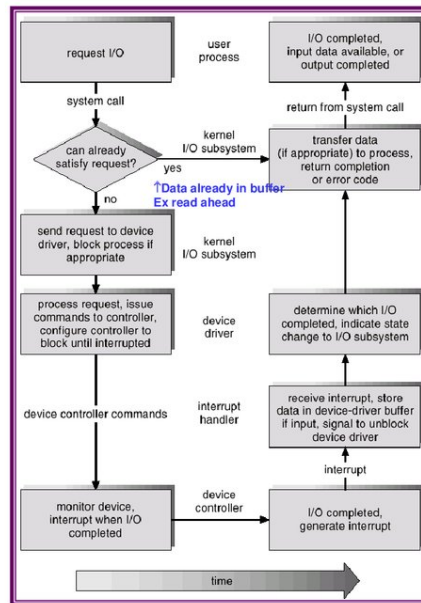
Two I/O Methods



Interrupt-Driven I/O Cycle



Life Cycle of An I/O Request

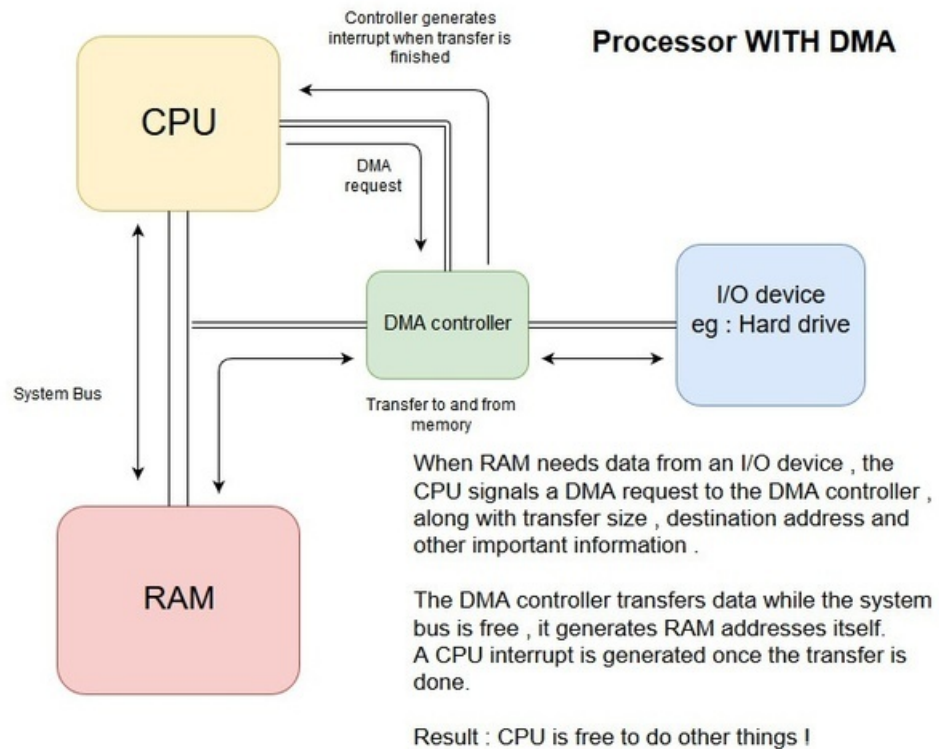


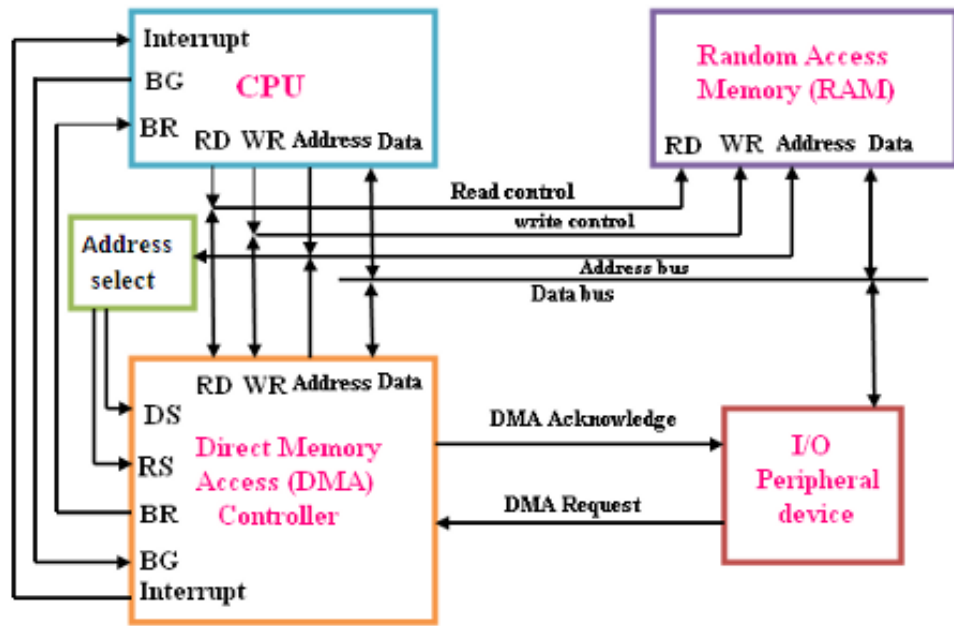
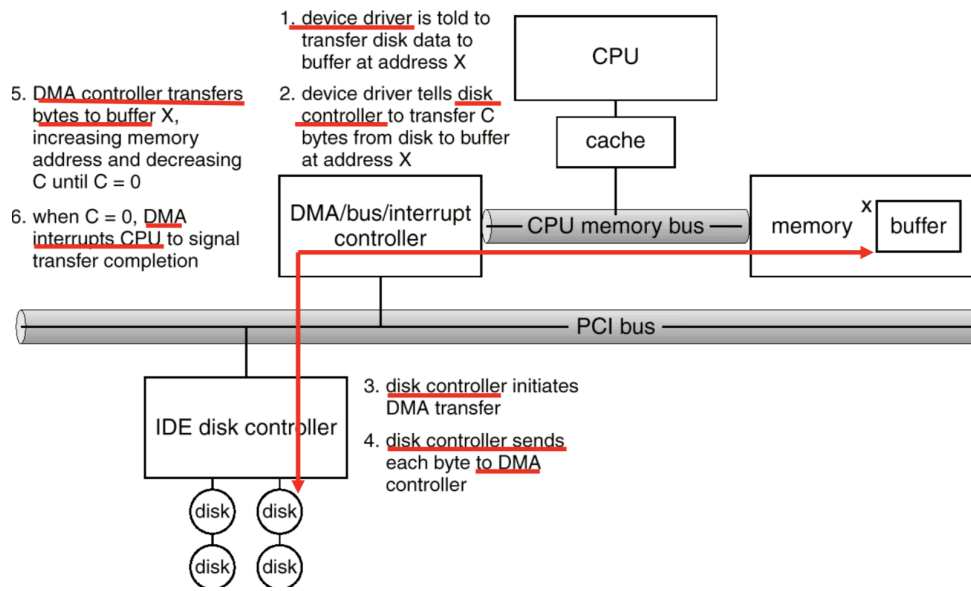
Operating System Concepts

13.25

Silberschatz, Galvin and Gagne ©2002

DMA(Direct Memroy Access)





-- Memo End --