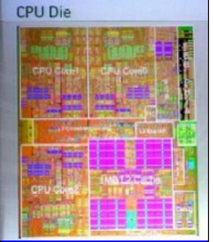
COMP 273
Introduction to Computer Systems

COMP 273

Digital Logic (Part 1) - RAM

Information Representation in Today’s Computers

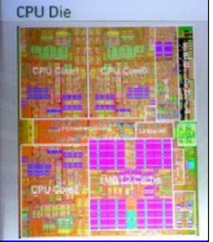
Prof. Joseph Vybihal



COMP 273
Introduction to Computer Systems

Announcements

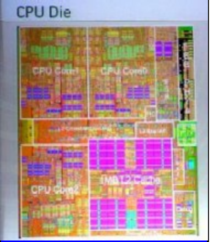
- Midterm Exam
  - In class
  - February 12, 2015



COMP 273
Introduction to Computer Systems

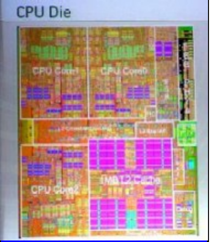
Question

Any questions about the System-board, bus, CPU layout and operation from last class?



Lecture Outline

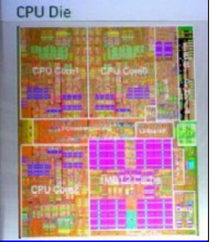
- Information Theory
- RAM
- Important Memory Representations
    - Machine language
    - Number Systems
    - Encoding Systems



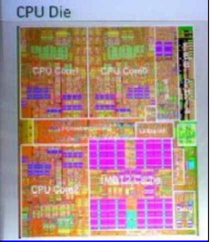
Try This Out At Home

- Write a small program with a lot of text. Compile it and look at the machine code. Identify the data and convert it back into characters and numbers.
  - Restrict your program to only positive integer and ASCII characters.
  - These are the simplest structures to identify and convert (some free-ware does it for you)

- Start reading the Soul Of A New Machine
- Think of physical ways to represent data



Information Theory



What is data?

Data is information

Examples:

- Characters (letters, punctuation, spaces)
- Symbols
- Numbers
- Java program instructions & Apps
- Web pages
- Images
- Databases
- Etc.

Claude E. Shannon, 1948

Entropy

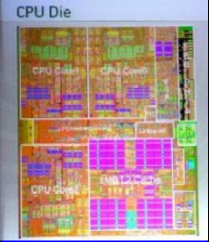
- How much work does it take to communicate one letter to someone?

The medium

- How can we transmit that single letter?

Realization

- The medium IS the message!



Claude E. Shannon, 1948

Entropy (in computers)

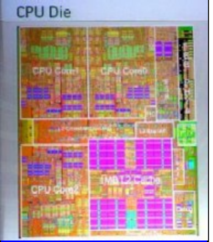
- How many bits do we need to represent a single character?

The medium (in computers)

- Light (optics), sound (WiFi), signals (wire)

Realization

- The message is characterized by the medium



COMP 273
Introduction to Computer Systems

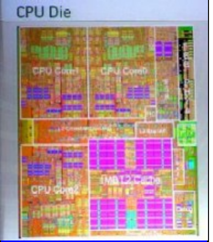
Conclusion

Entropy

- We need to find an efficient way to represent letters within a computer

Medium

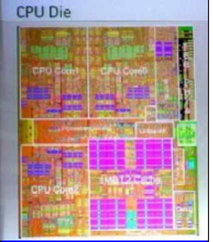
- Which medium should we pick?
  - Because each kind effects how we will do entropy and how we build software and hardware to access those letters.



Question

In how many ways can we represent information physically in a computer?

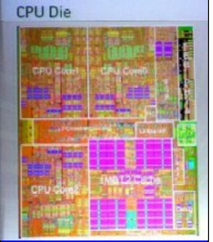
- Light bulbs
- Metal bar
- Punch cards
- Chemicals
- Laser formed pot marks
- Etc…



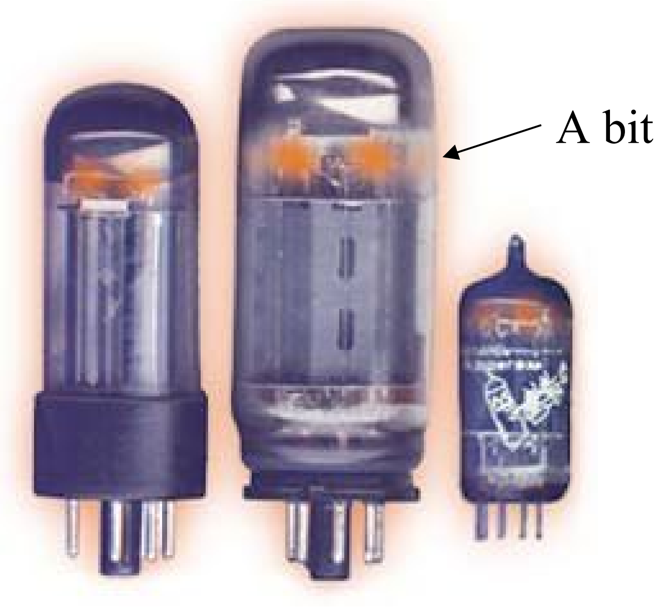
Answer

Light Bulbs

They are the easiest to use. They have only two states. So we will adopt binary.

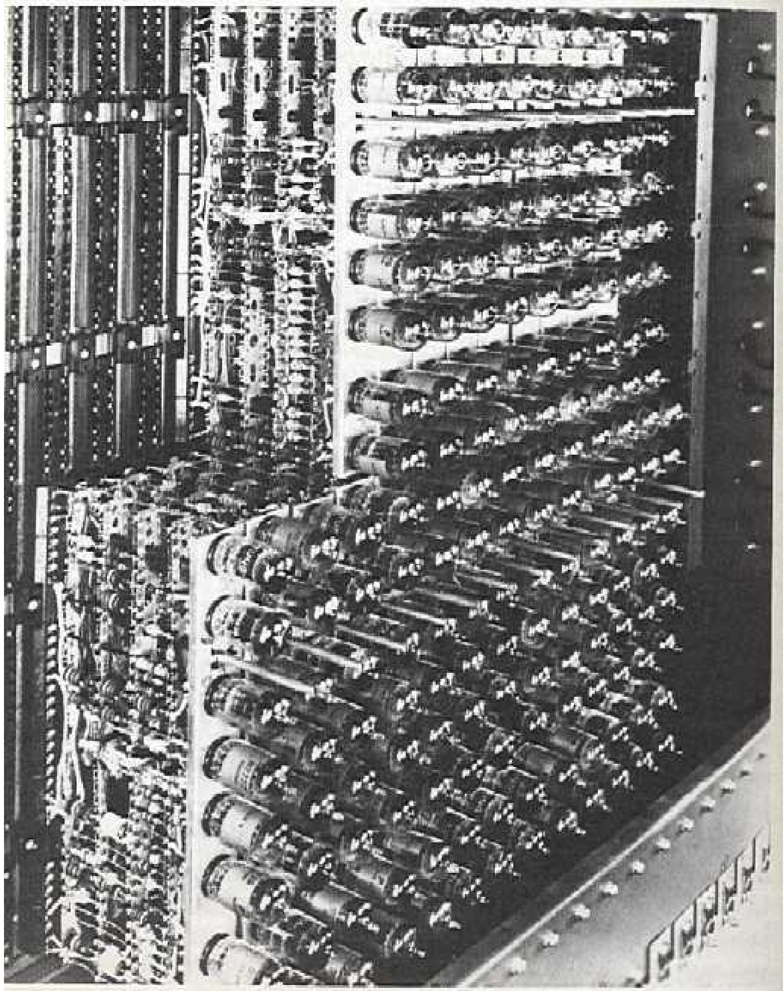


Bits and Bytes

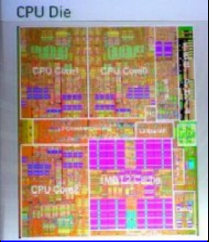


Coding data as light

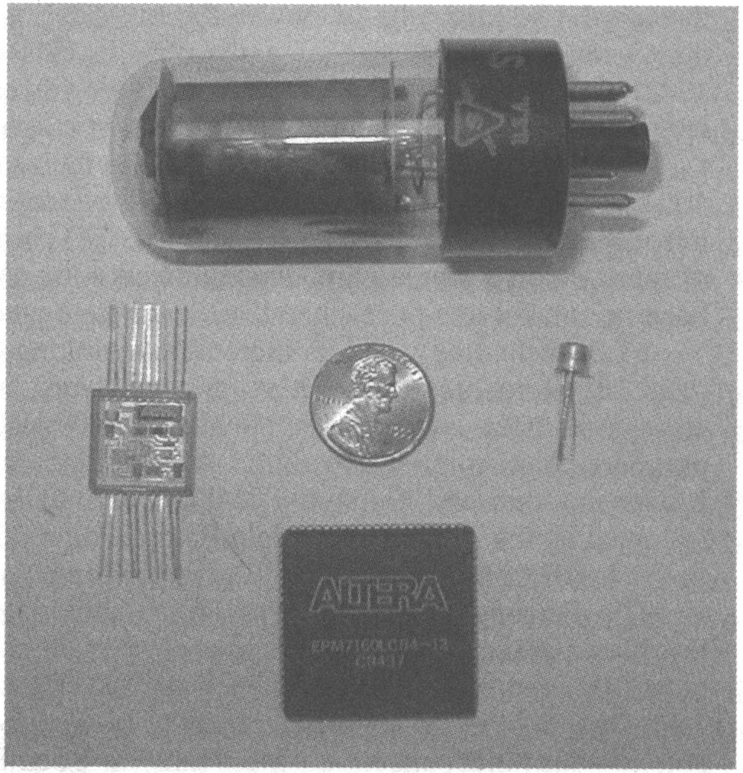
Bit	= 1 bit
Nibble	= 4 bits
Byte	= 8 bits



RAM ~in 1940

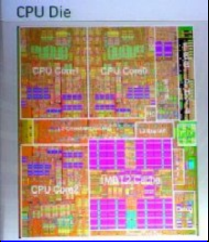


Bit Structures



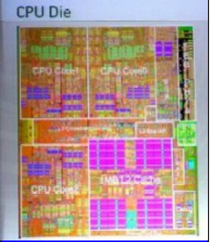
Courtesy of Linda Null

FIGURE 1.2 Comparison of Computer Components
Clockwise, starting from the top:
1) Vacuum Tube
2) Transistor
3) Chip containing 3200 2-input NAND gates
4) Integrated circuit package (the small silver square in the lower left-hand corner is an integrated circuit)



Bits and Information

Light Bulbs can be used to encode

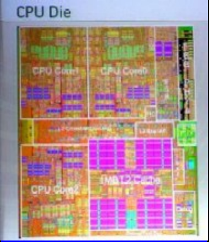


Fundamental Unit of Data

- The Bit
- Binary Value: 0 and 1
- Physical representation:
  - 2-5V is 1
  - 0-1V is 0

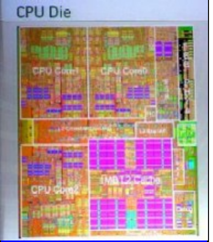
Space provided for fluctuations in current

- Used to represent:
  - 1/0
  - T/F
  - On/Off
  - Yes/No
  - Flags and Switches (status information / turn on or off message)



Three basic things to encode

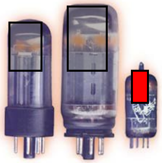
- Characters
  - Use a predetermined table of value pairs
    - Character : Integer code-number
- Numbers
  - Use binary arithmetic
- Program Instructions
  - Use a predetermined structure & table of value pairs
    - Instruction : Gate sequence code-number

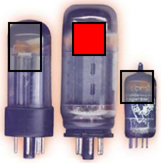


Binary Codes - encoding

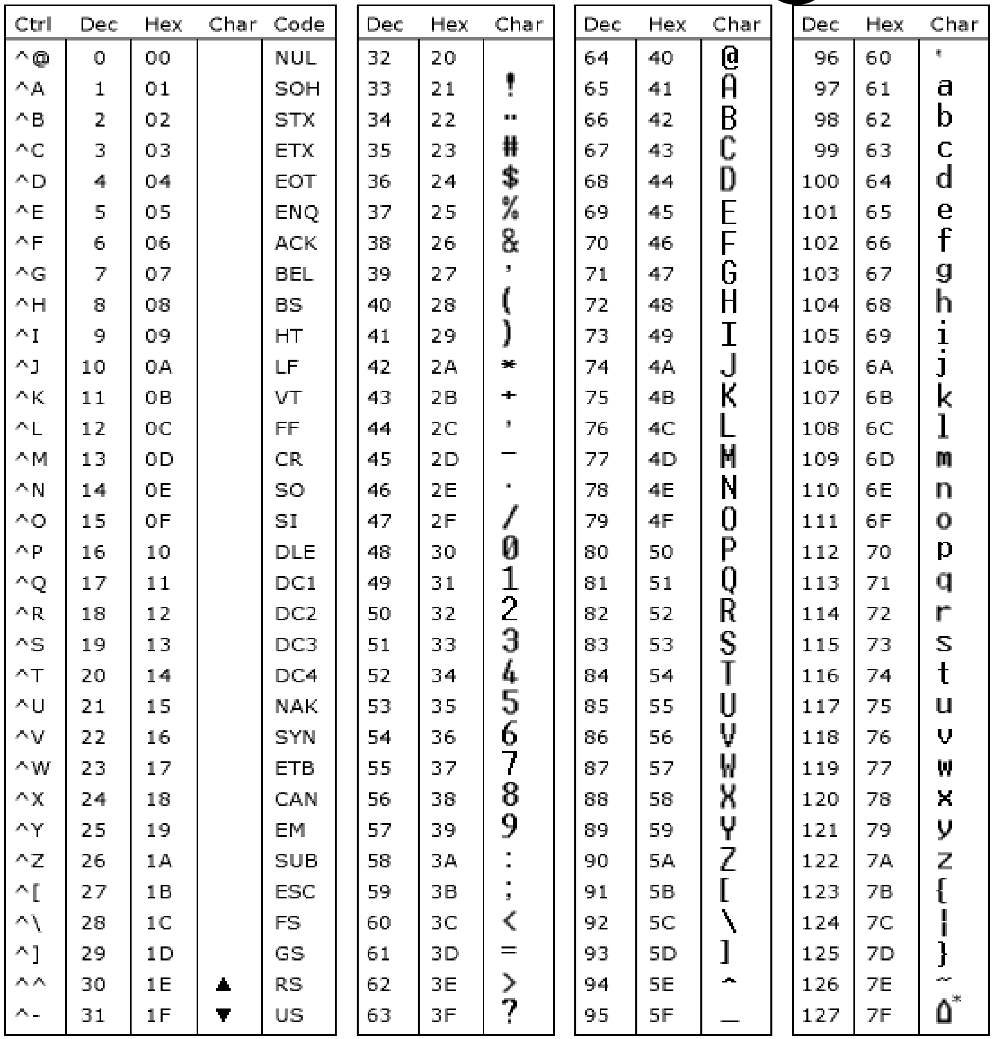
3 bit binary math

000 0


001 1


010 2

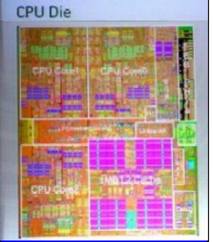
Numbers



Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20		64	40	@	96	60	`
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	"	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	'	71	47	G	103	67	g
^H	8	08		BS	40	28	(	72	48	H	104	68	h
^I	9	09		HT	41	29	)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B		ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	|
^]	29	1D		GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂*

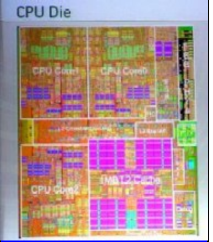
* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

Characters



Example

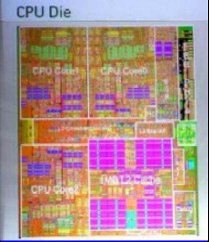
If I have 4 bits (known as a “nibble”), how many unique sequences can I make?



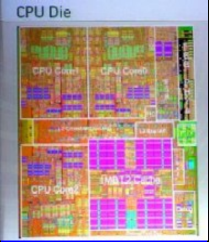
Digital Data

PC data is digital

Each unique sequence represents a single piece of data

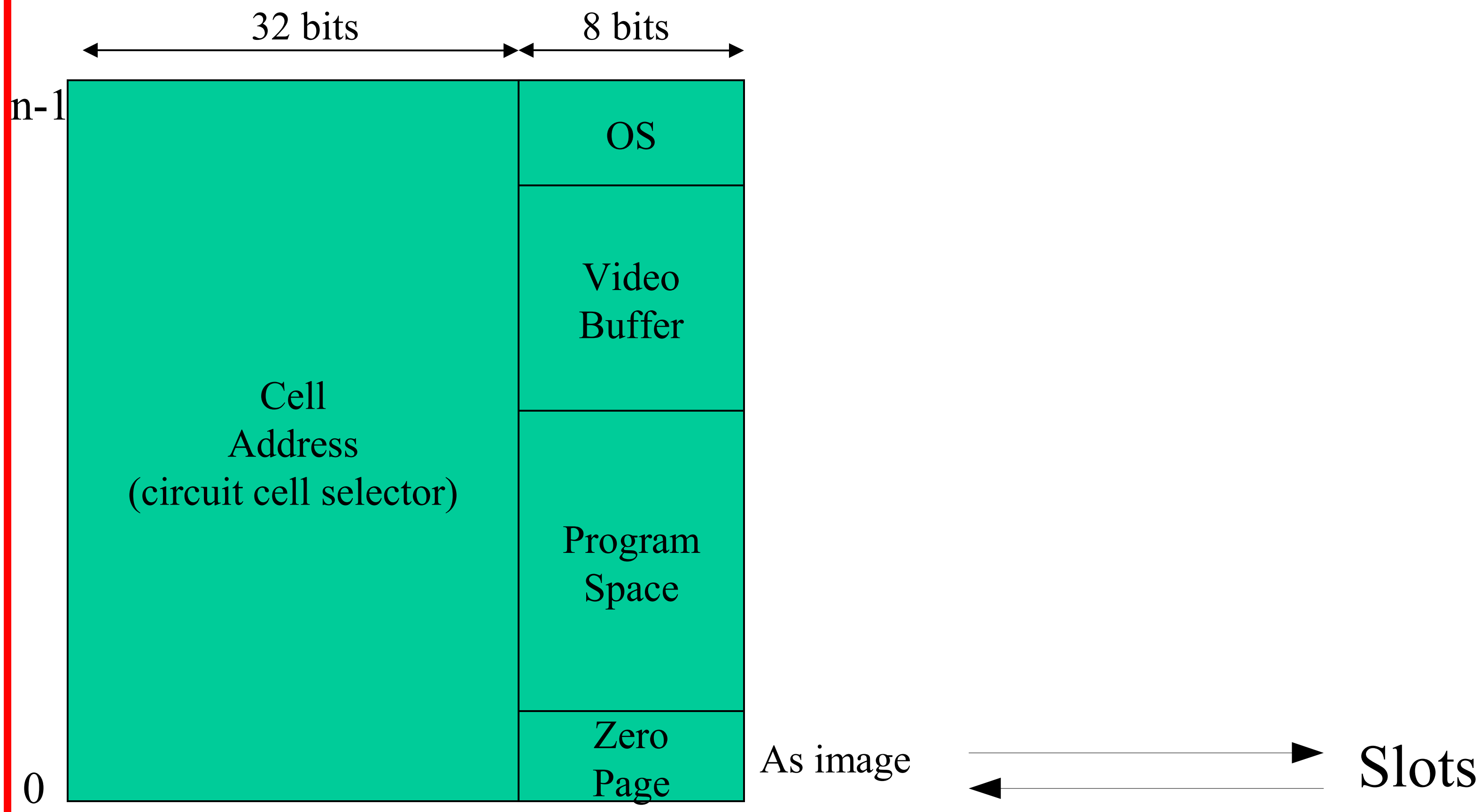


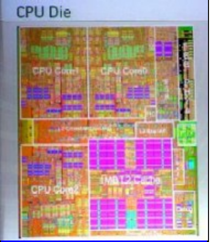
RAM



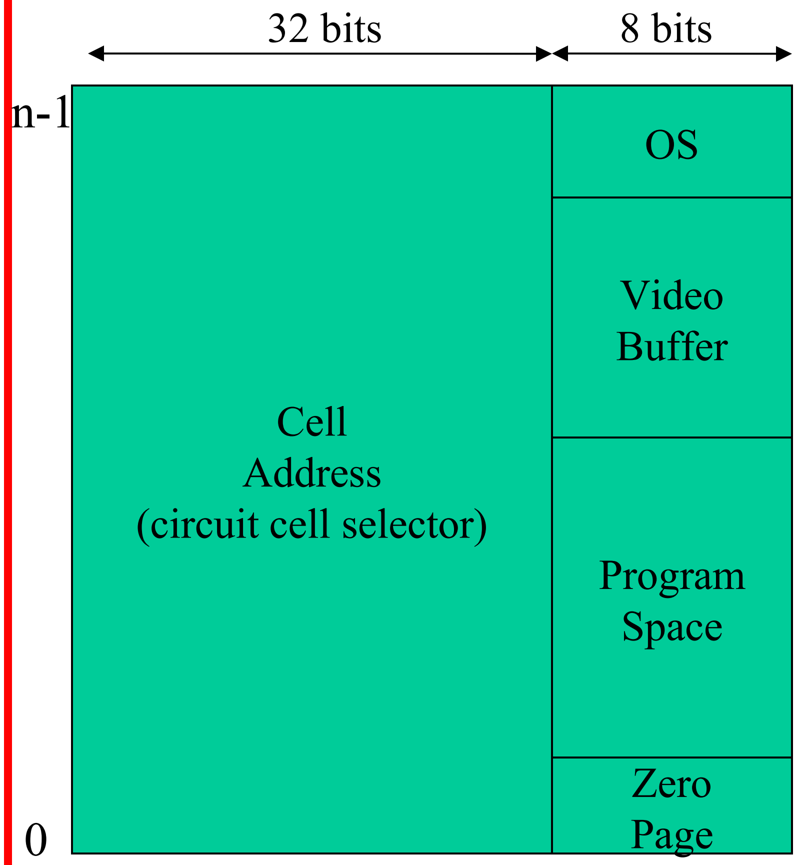
COMP 273
Introduction to Computer Systems

Basic RAM Addressing Schematic



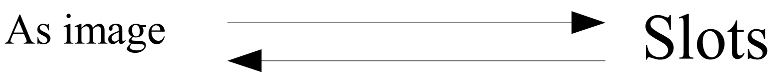


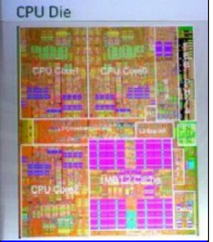
Basic RAM Addressing Schematic



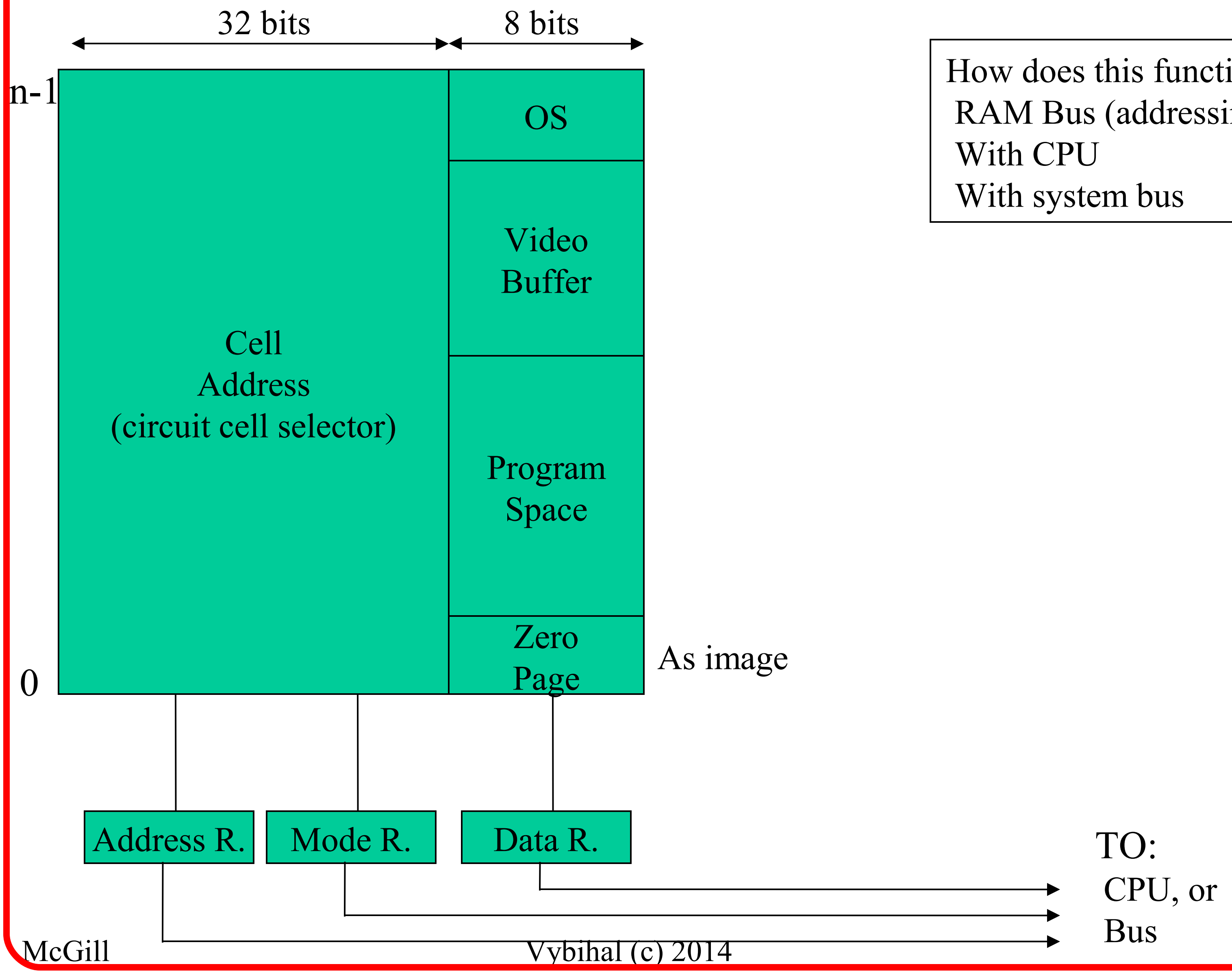
$2^{32} = 4{,}294{,}967{,}296$

$2^{64} = 18{,}446{,}744{,}073{,}709{,}551{,}616$

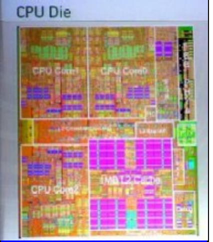




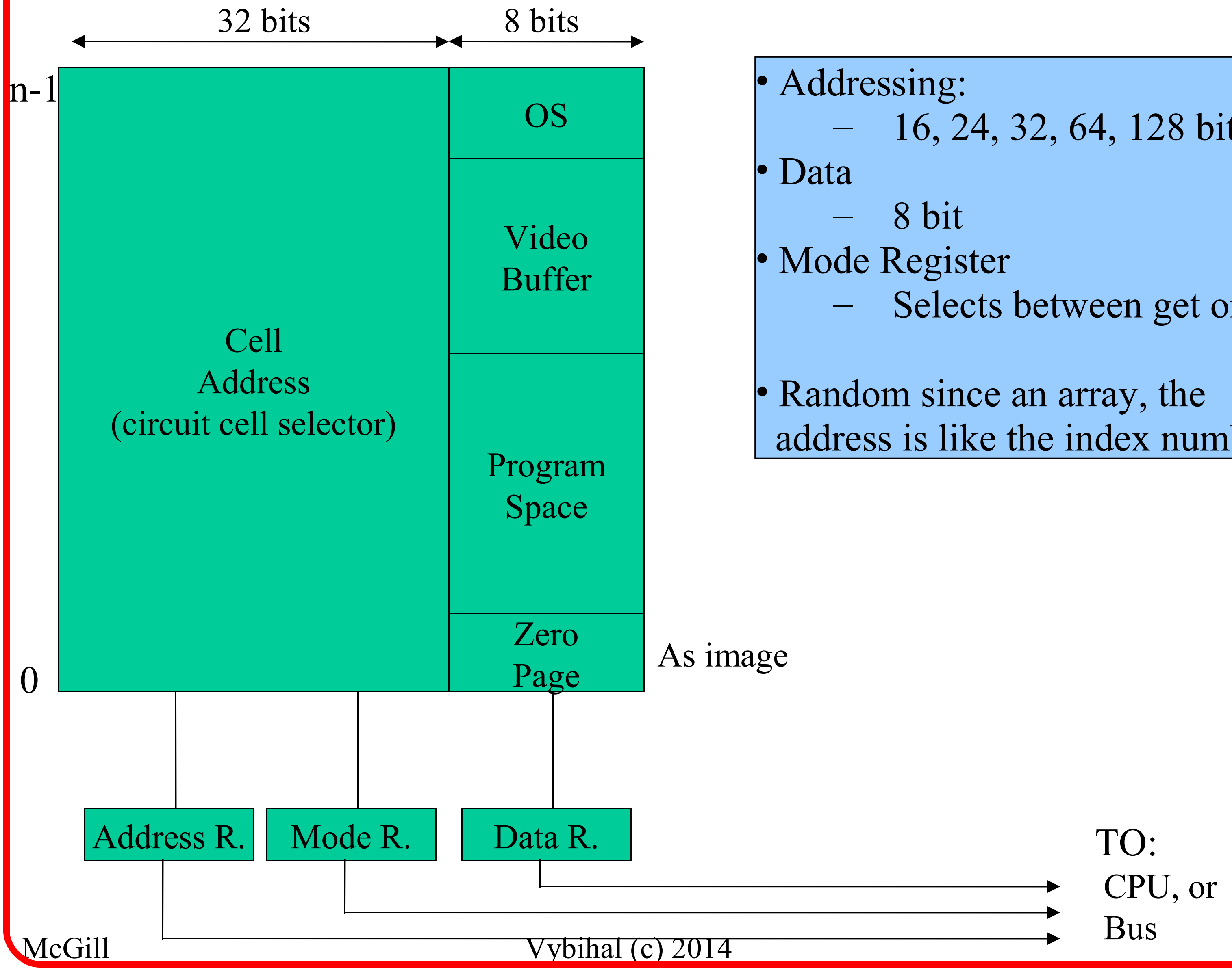
Basic RAM Addressing Schematic



How does this function?
RAM Bus (addressing)
With CPU
With system bus



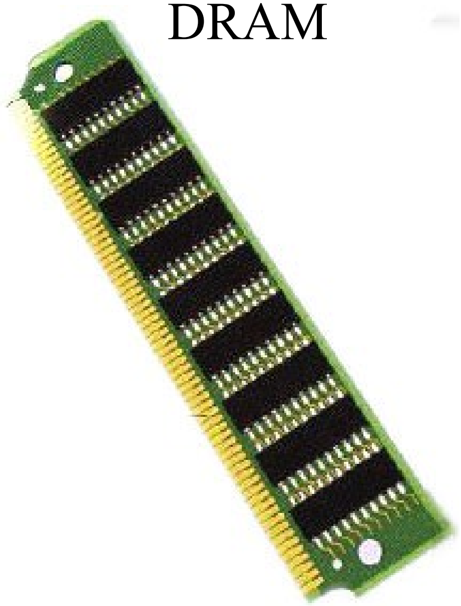
Basic RAM Addressing Schematic



- Addressing:
  - 16, 24, 32, 64, 128 bit
- Data
  - 8 bit
- Mode Register
  - Selects between get or put
- Random since an array, the address is like the index number

RAM

DRAM

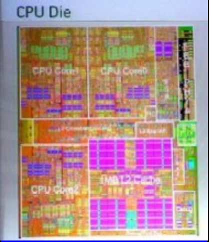


Random Access Memory

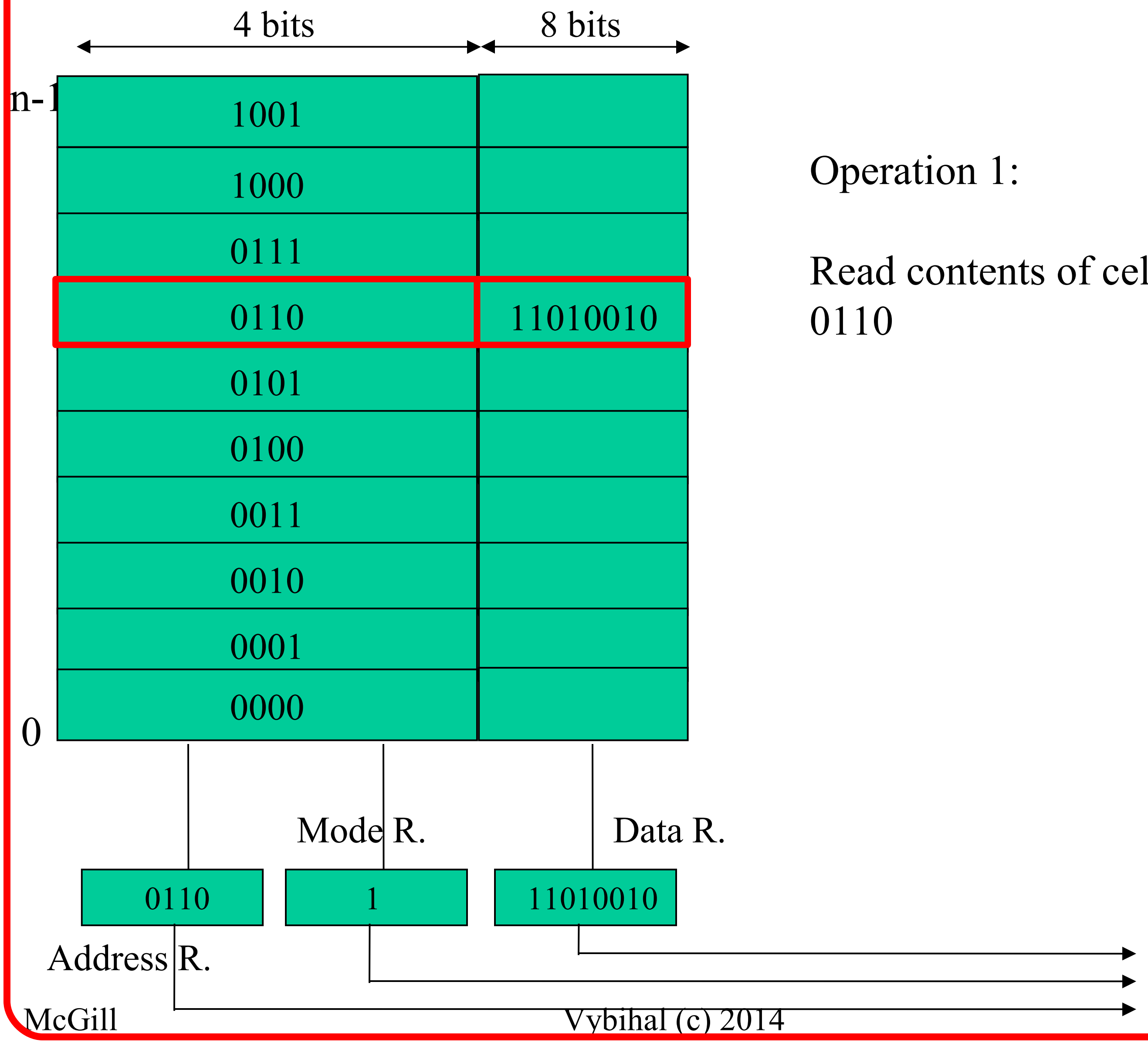
Read:
- **Address Register =** integer
- **Mode Register** = read flag
- **Data Register** loaded from RAM

Write:
- **Address Register** = integer
- **Mode Register** = write flag
- **Data Register** = data
- Data will be saved into RAM at integer address.

<u>Memory Hierarchy consists of</u>:
- registers (for CPU)
- on-die SRAM (static RAM) caches,
- DRAM (dynamic RAM),
- Paging systems, and
- Virtual memory or swap space on a hard-drive



Basic RAM Addressing Schematic



4 bits	8 bits
1001	
1000	
0111	
0110	11010010
0101	
0100	
0011	
0010	
0001	
0000	

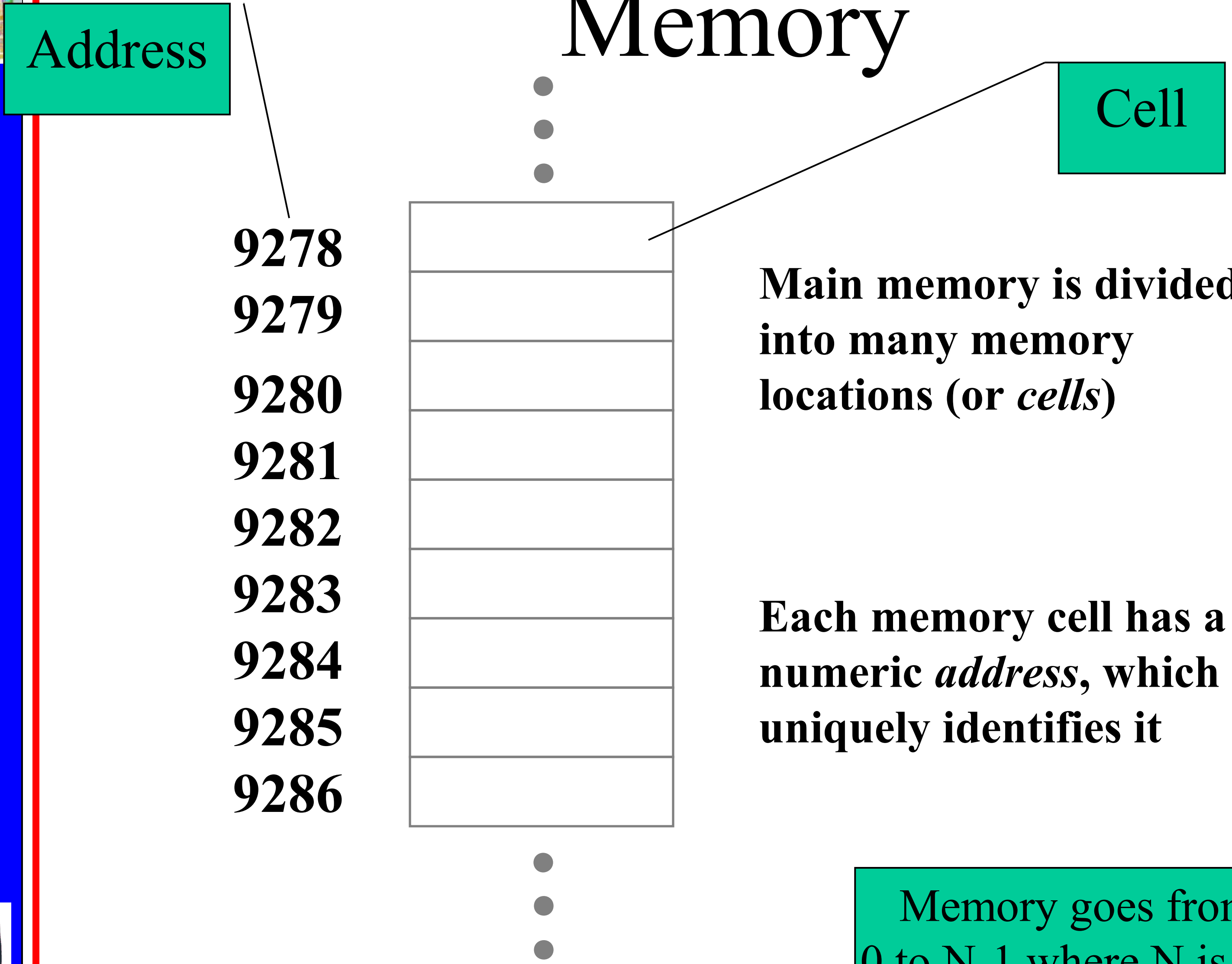
Address R.	Mode R.	Data R.
0110	1	11010010

TO:
CPU, or
Bus

Operation 1:

Read contents of cell
0110

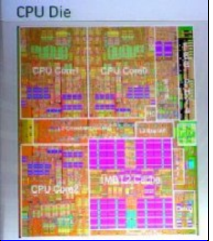
Memory



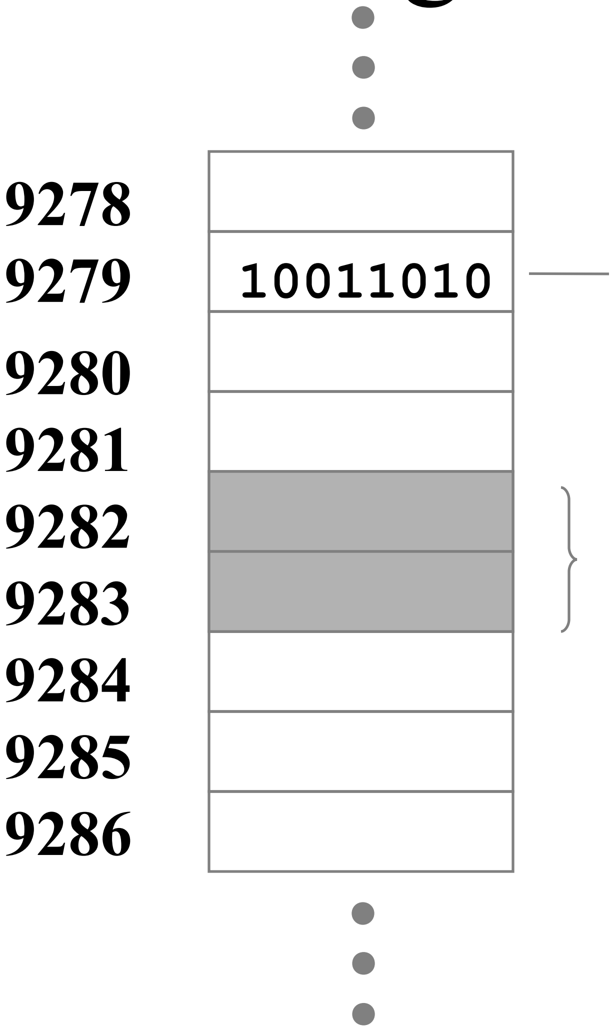
Main memory is divided into many memory locations (or *cells*)

Each memory cell has a numeric *address*, which uniquely identifies it

Memory goes from address 0 to N-1 where N is the amount you purchased.



Storing Information



9278

9279 10011010

9280

9281

9282

9283

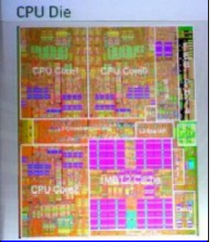
9284

9285

9286

Each memory cell stores a set number of bits (usually 8 bits, or one *byte*)

Large values are stored in consecutive memory locations

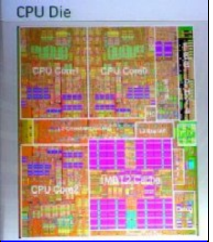


COMP 273
Introduction to Computer Systems

Data

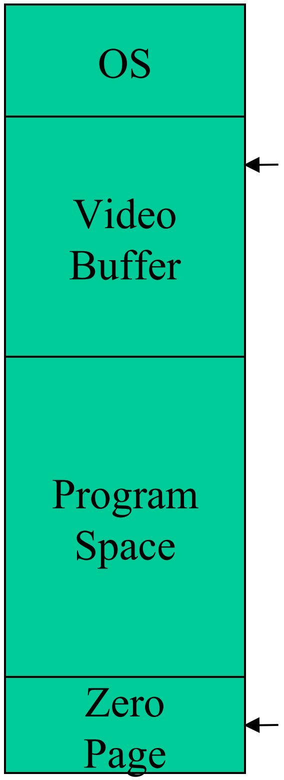
- Data – Information (bits & bytes) stored in RAM or secondary storage. Data can be computer instructions (a program) or information (like words and numbers).
- Built-In Types – Common built-in formats that the CPU can understand (eg. integer, real and character); physical circuits exists to interpret these values.
- Language Types – Extra data formats defined by your programming language (eg. strings); no circuits exist, they need to be simulated.

How can a language simulate a type?



Basic RAM Schematic Formats

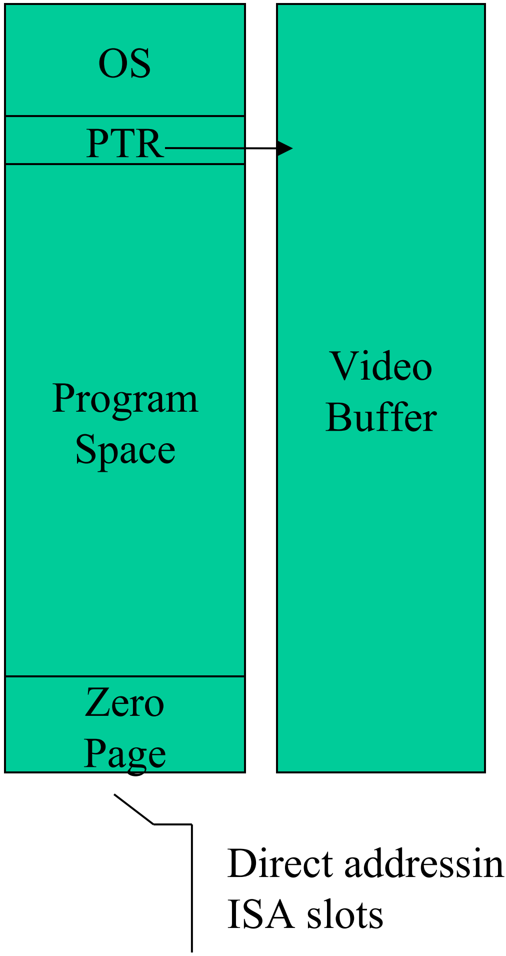
Type 1



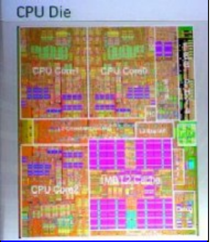
- It is standard RAM, except that each byte is mapped physically to an actual pixel on a screen.
- Depending of the screen resolution modes supported by the monitor, each pixel may map to one or more bytes (integer color codes + ROM)

A buffer containing an **image** of all the registers of devices attached to the PC

Type 2



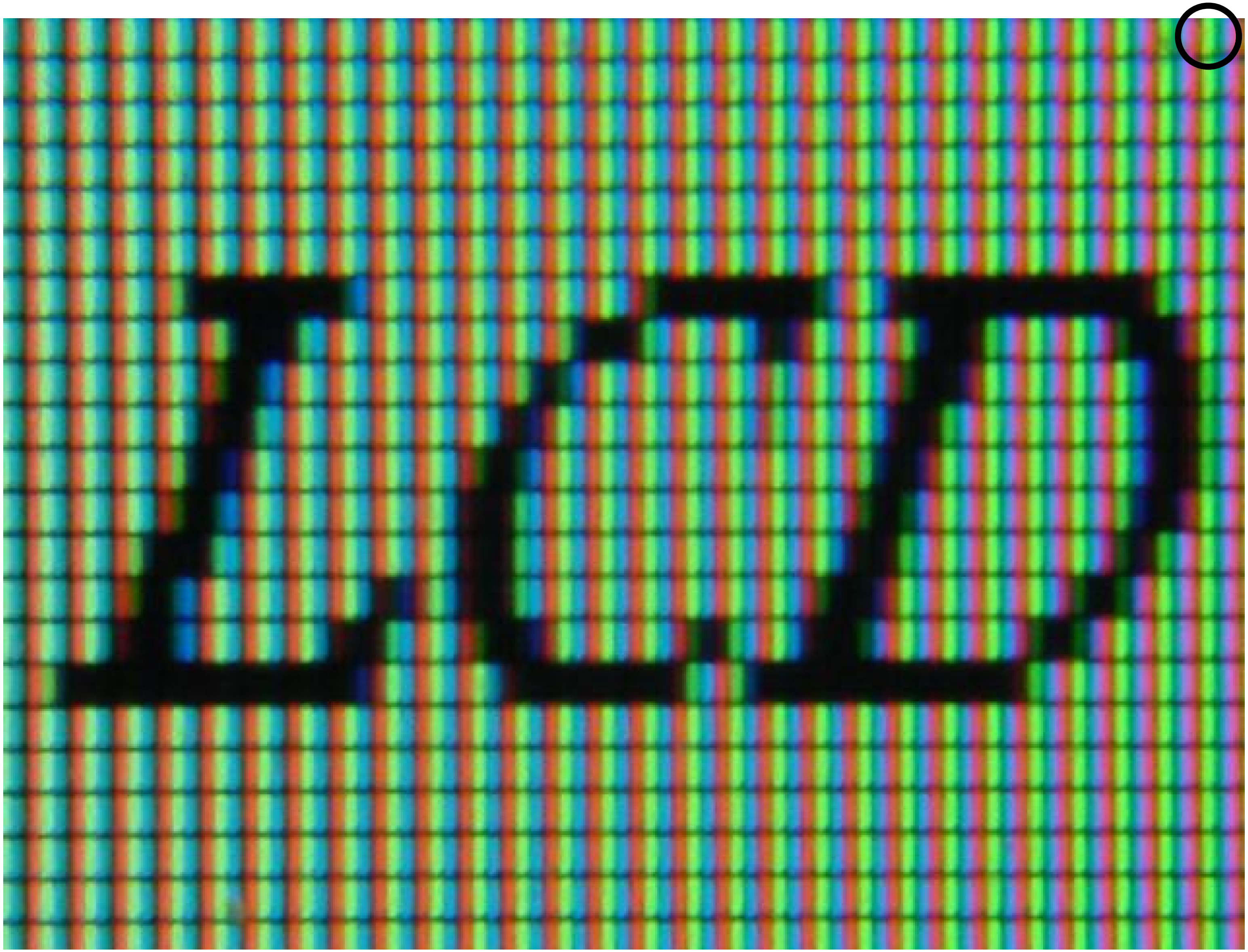
Direct addressing to ISA slots

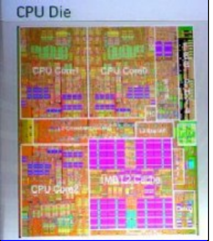


Video and RAM

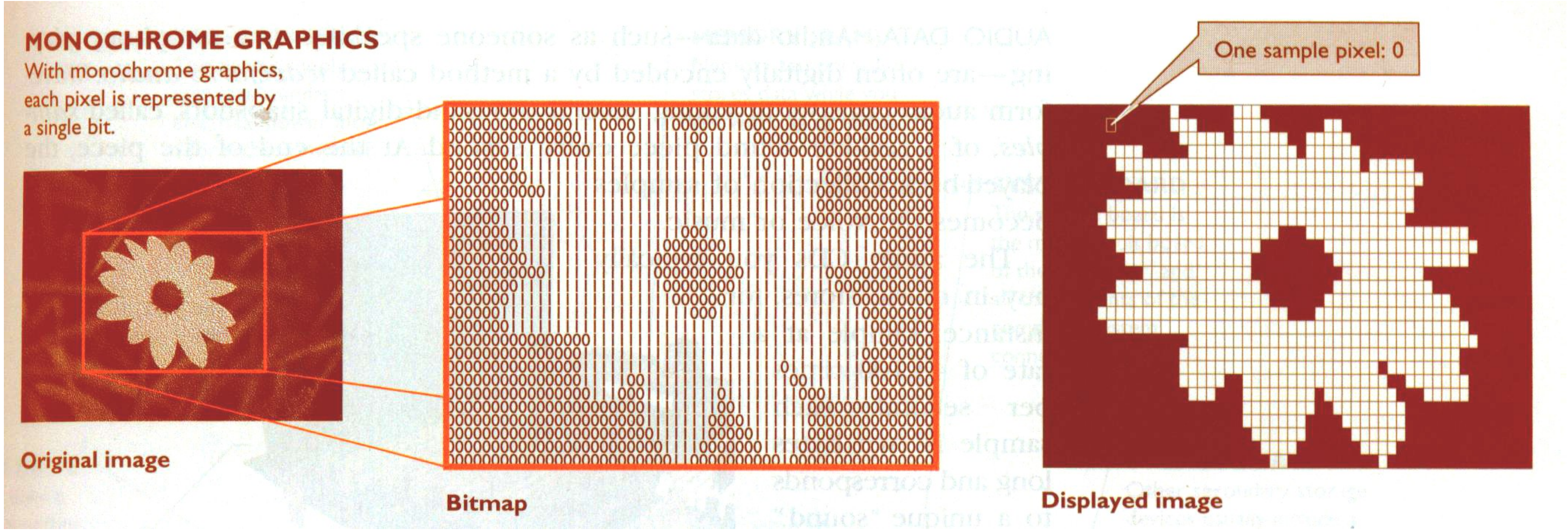
The RAM memory buffer maps 1:1 to pixels

Pixels





Each pixel is represented as an integer number that represents the colour of that pixel.



MONOCHROME GRAPHICS

With monochrome graphics, each pixel is represented by a single bit.

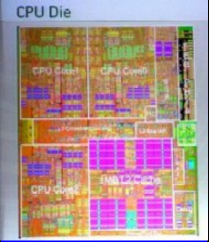
Original image

Bitmap

Displayed image

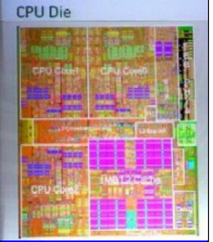
One sample pixel: 0

If your image supports 32 million colours, the number size in bits is 25 bits long or 4 bytes per pixel.



RAM & Machine Language

What is Assembly & Machine Language?



Breaking Down Instructions

Java or C:

```
a = b + (C + d) * 4;
```

Broken down:

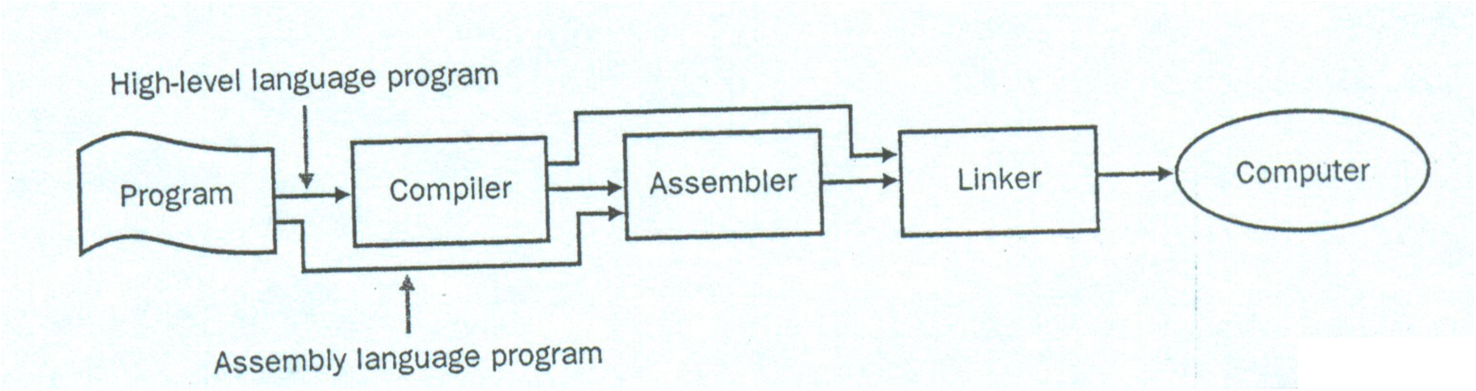
```
Temp1 = c + d;
Temp2 = Temp1 * 4;
a = b + Temp2;
```

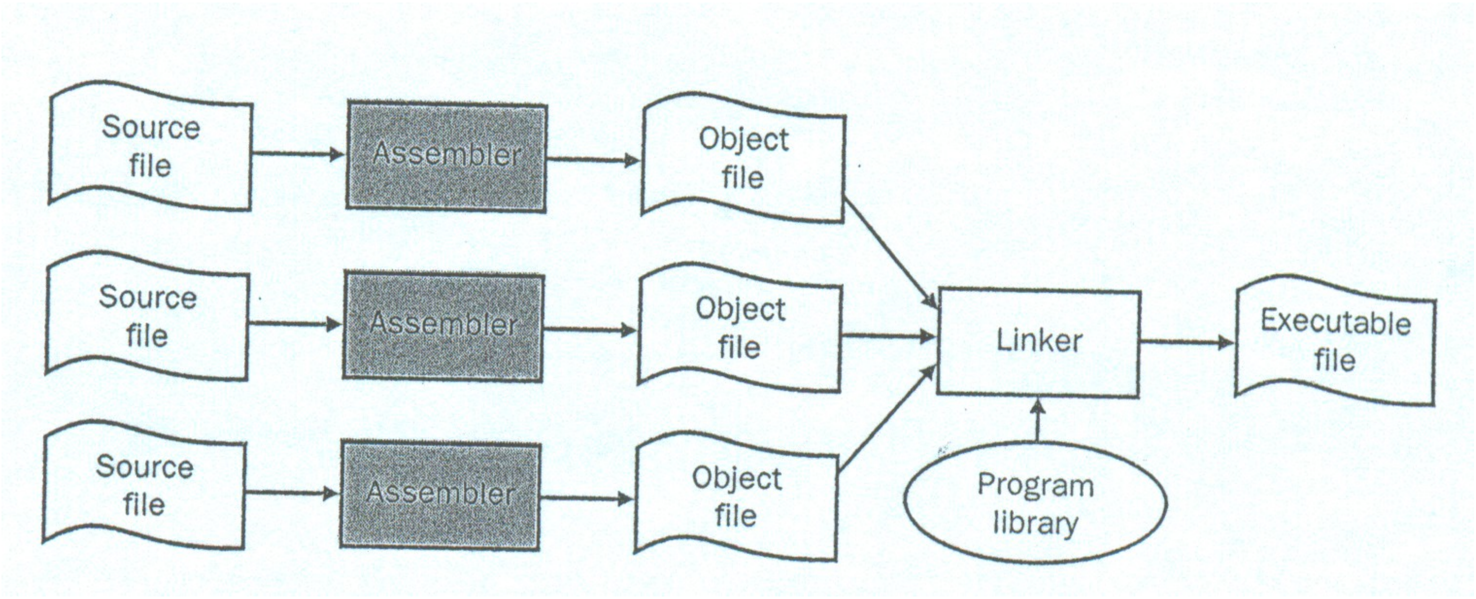
Assembler (like broken down C):

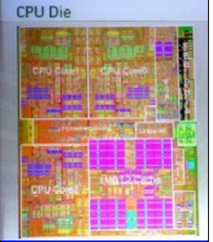
```
add temp1, c, d
mul temp2, 4, temp1
add a, temp2, b

OPCODE  DEST  LNO  RNO
```

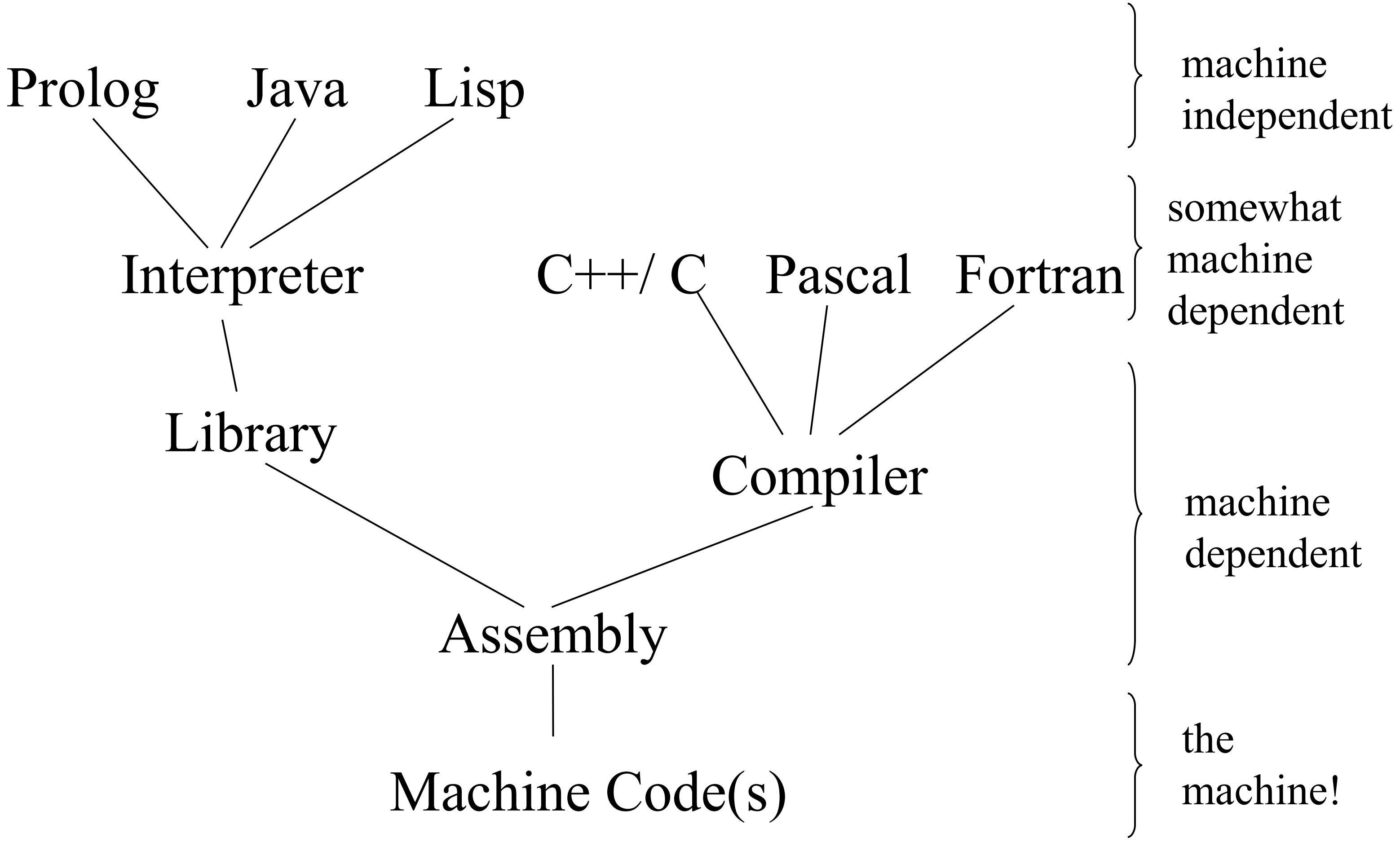
The Compilation Process

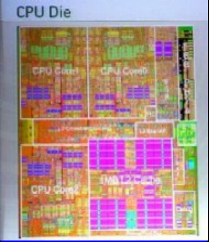






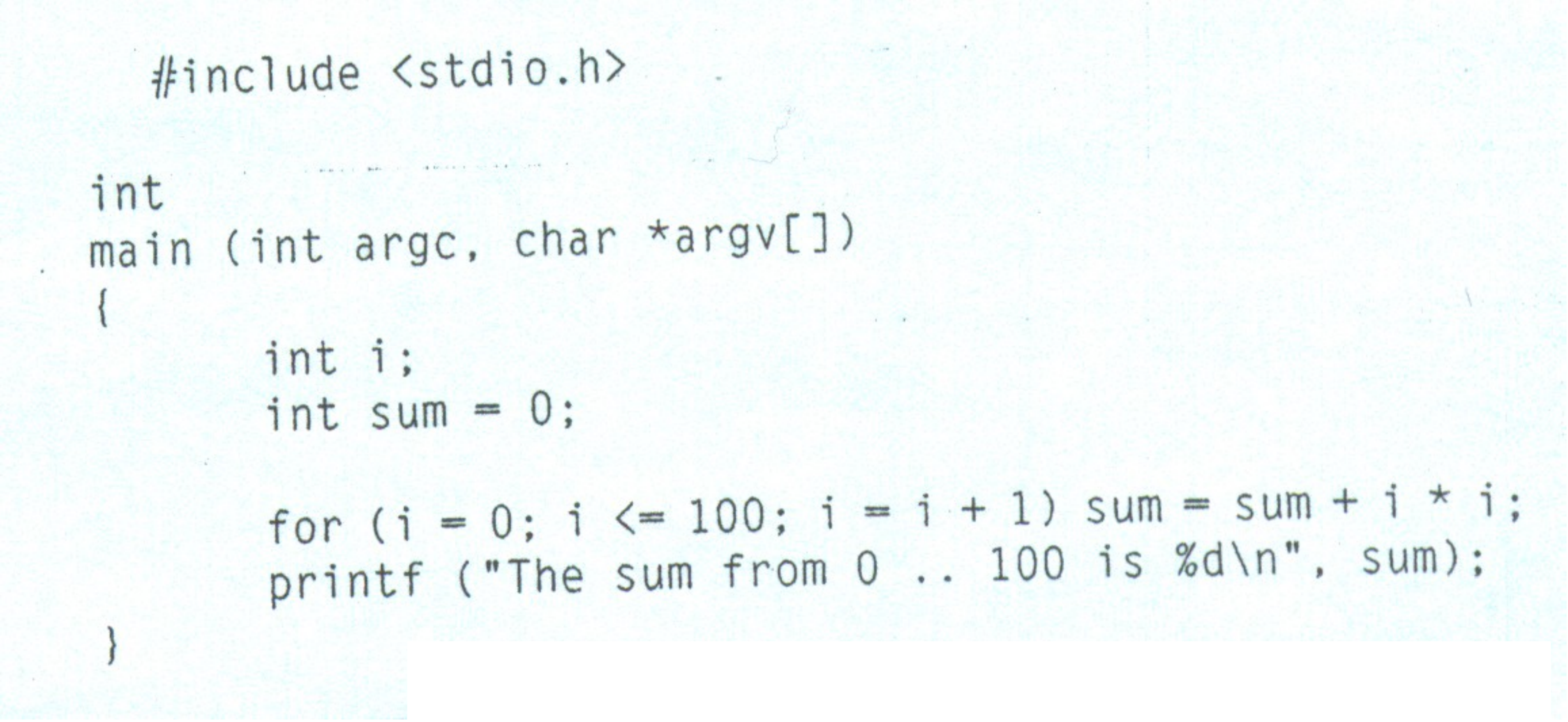
Machine Dependence of Programming Languages





Machine Code (1/4)

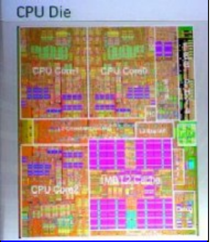
C Code



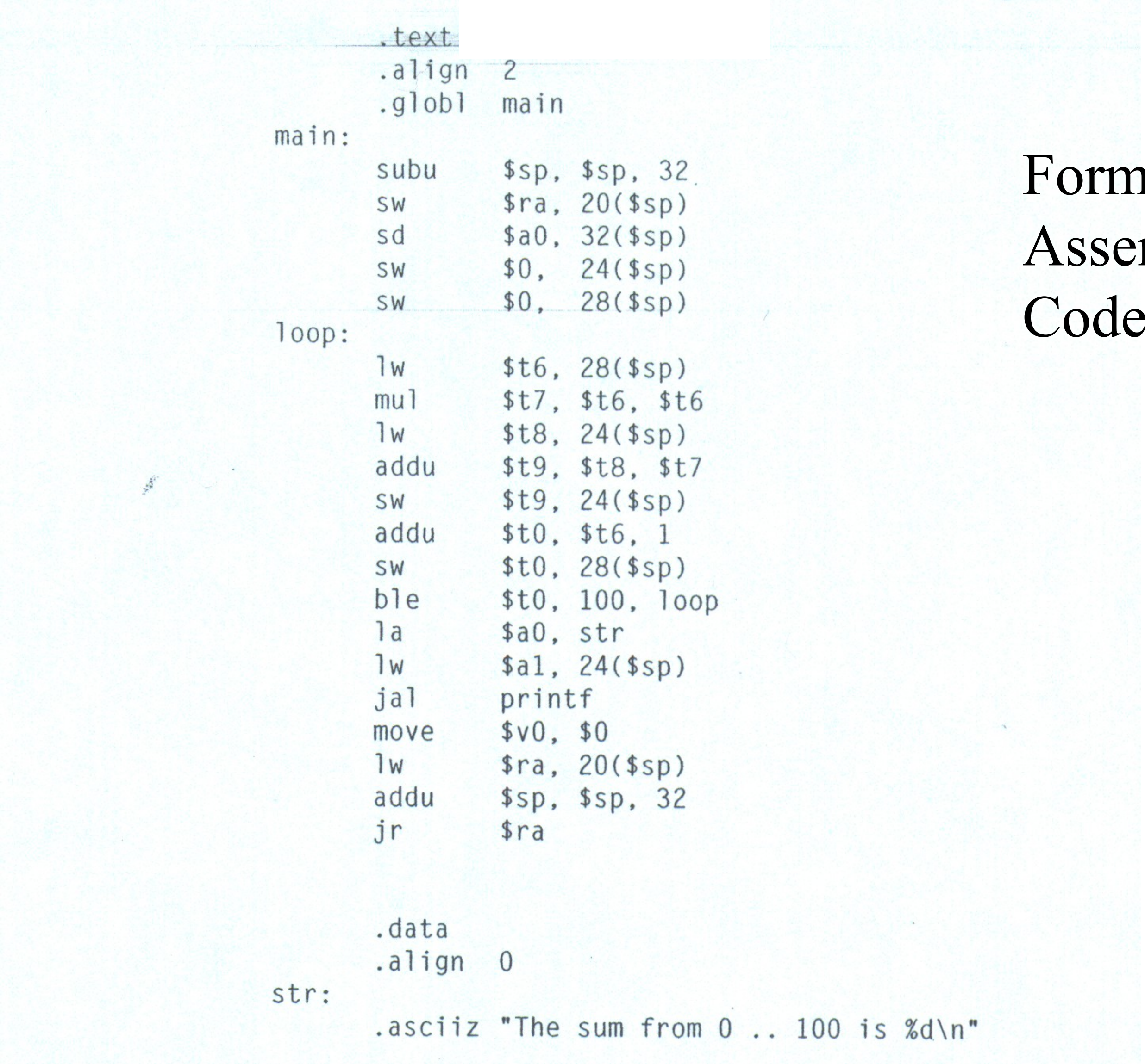
```
  #include <stdio.h>

int
main (int argc, char *argv[])
{
        int i;
        int sum = 0;

        for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
        printf ("The sum from 0 .. 100 is %d\n", sum);
}
```



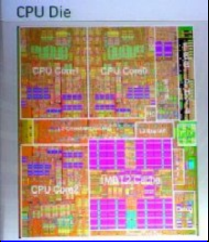
Machine Code (2/4)



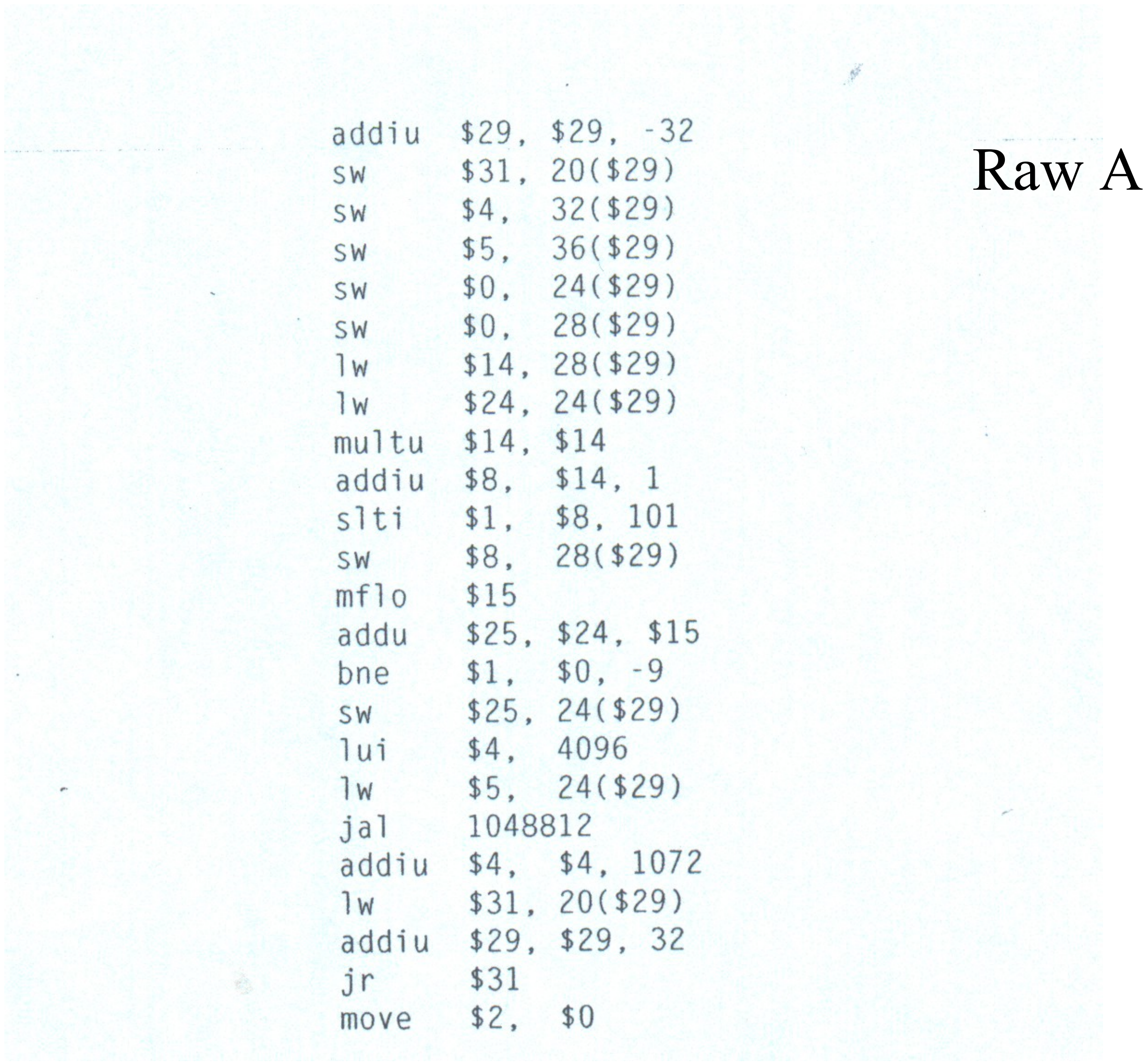
```
          .text
          .align   2
          .globl   main
main:
          subu     $sp, $sp, 32
          sw       $ra, 20($sp)
          sd       $a0, 32($sp)
          sw       $0,  24($sp)
          sw       $0,  28($sp)
loop:
          lw       $t6, 28($sp)
          mul      $t7, $t6, $t6
          lw       $t8, 24($sp)
          addu     $t9, $t8, $t7
          sw       $t9, 24($sp)
          addu     $t0, $t6, 1
          sw       $t0, 28($sp)
          ble      $t0, 100, loop
          la       $a0, str
          lw       $a1, 24($sp)
          jal      printf
          move     $v0, $0
          lw       $ra, 20($sp)
          addu     $sp, $sp, 32
          jr       $ra


          .data
          .align   0
str:
          .asciiz "The sum from 0 .. 100 is %d\n"
```

Formatted Assembler Code

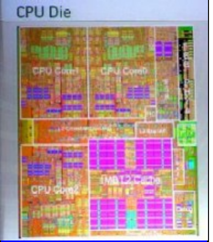


Machine Code (3/4)

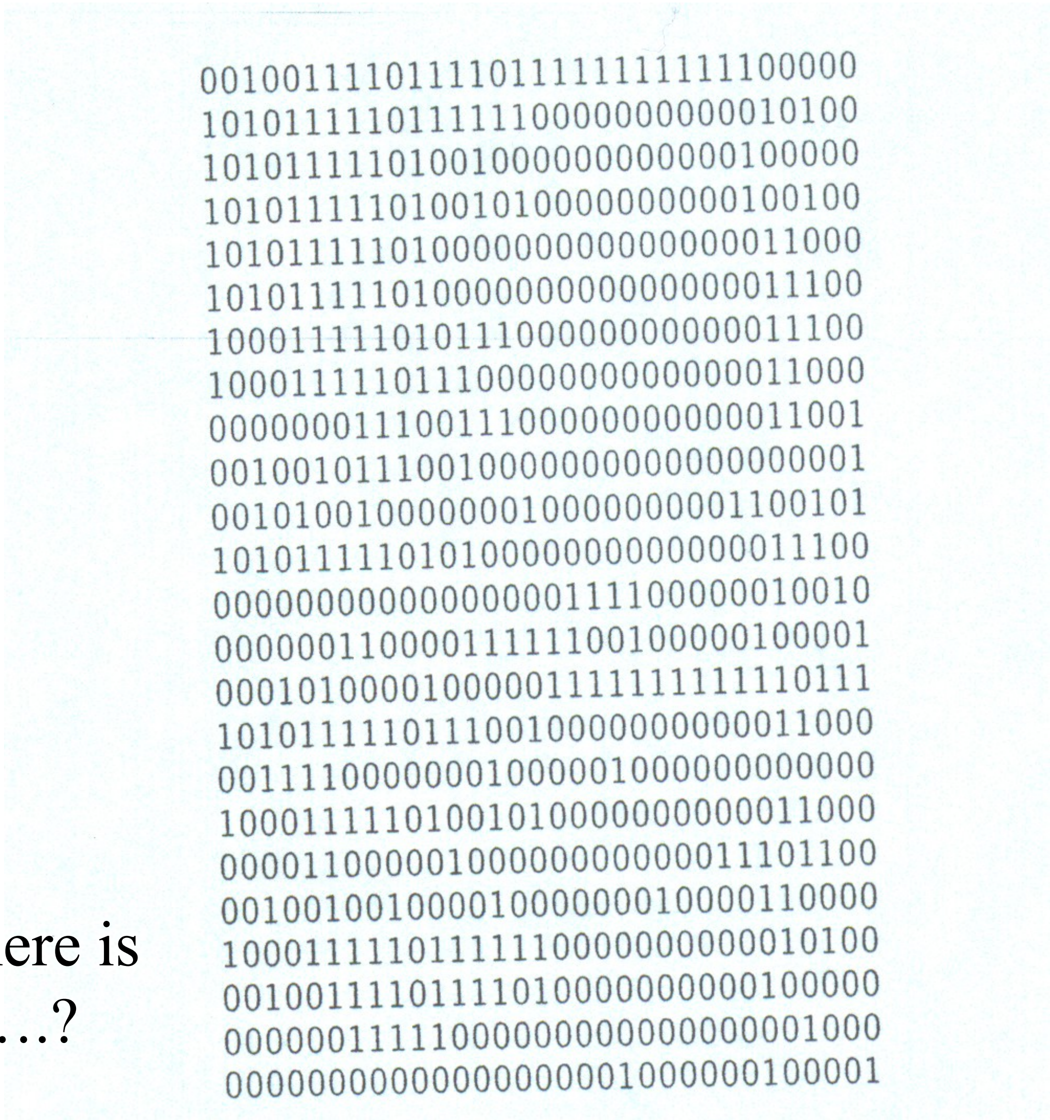


```
addiu   $29, $29, -32
sw      $31, 20($29)
sw      $4,  32($29)
sw      $5,  36($29)
sw      $0,  24($29)
sw      $0,  28($29)
lw      $14, 28($29)
lw      $24, 24($29)
multu   $14, $14
addiu   $8,  $14, 1
slti    $1,  $8, 101
sw      $8,  28($29)
mflo    $15
addu    $25, $24, $15
bne     $1,  $0, -9
sw      $25, 24($29)
lui     $4,  4096
lw      $5,  24($29)
jal     1048812
addiu   $4,  $4, 1072
lw      $31, 20($29)
addiu   $29, $29, 32
jr      $31
move    $2,  $0
```

Raw Assembler Code

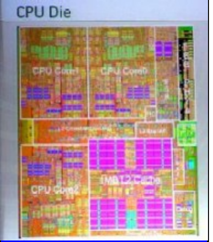


Machine Code (4/4)



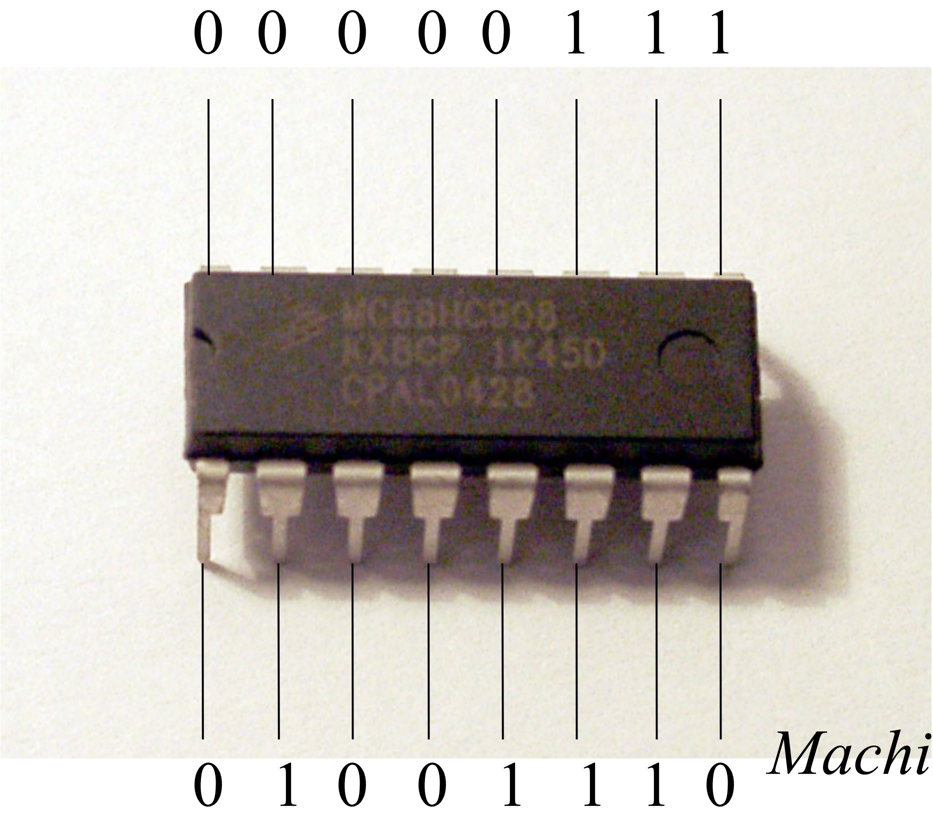
```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Hmm, where is my error….?



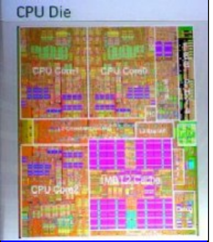
COMP 273
Introduction to Computer Systems

The Machine’s Language


0 0 0 0 0 1 1 1

0 1 0 0 1 1 1 0

Machine Language

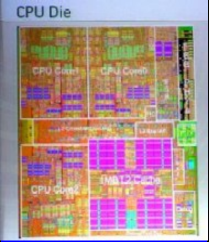


Important Number Representations

Bit Groupings (1/2)

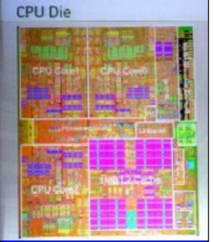
- Used to code information
  - Nibble: 4 bits
  - Byte: 8 bits ← *Primary addressable unit for PC*
  - Word: 16 bits
  - Long Word: 32 bits (or just Word)
  - Quad Word: 64 bits (or just Word)

Note: the address size itself may be different.
EG: 486 28bit address



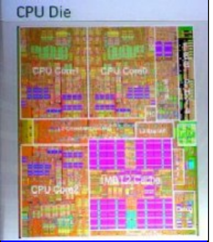
About WORDS

- Modern definition of a Byte:
  - The smallest number of bits assembled into a unit that can be addressed by a single number in RAM.
- Modern definition of a Word:
  - The "common" "size" of a register within the CPU.
    - Where *size* refers to the number of bits, and
    - *Common* refers to the register size the CPU was designed best to function with. (CPUs often have multiple registers sizes)

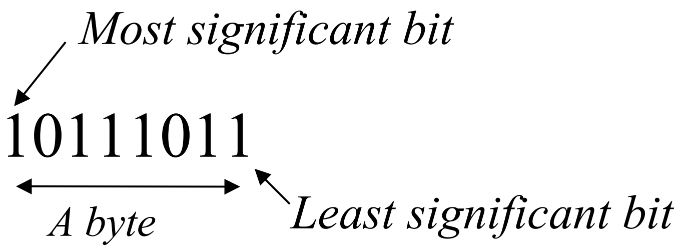


Notes about Words

- Word sizes can differ between CPUs.
  - 8 bit
  - 16 bit
  - 32 bit
  - 64 bit
  - 128 bit
- The RAM has not been plagued by this
  - A byte is always 8 bits
  - A byte is the smallest address in RAM
  - UNICODE may effect this in the future...



Bit Groupings (2/2)

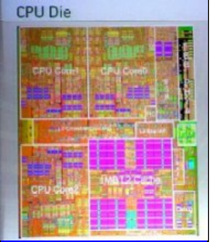

Most significant bit

10111011

A byte

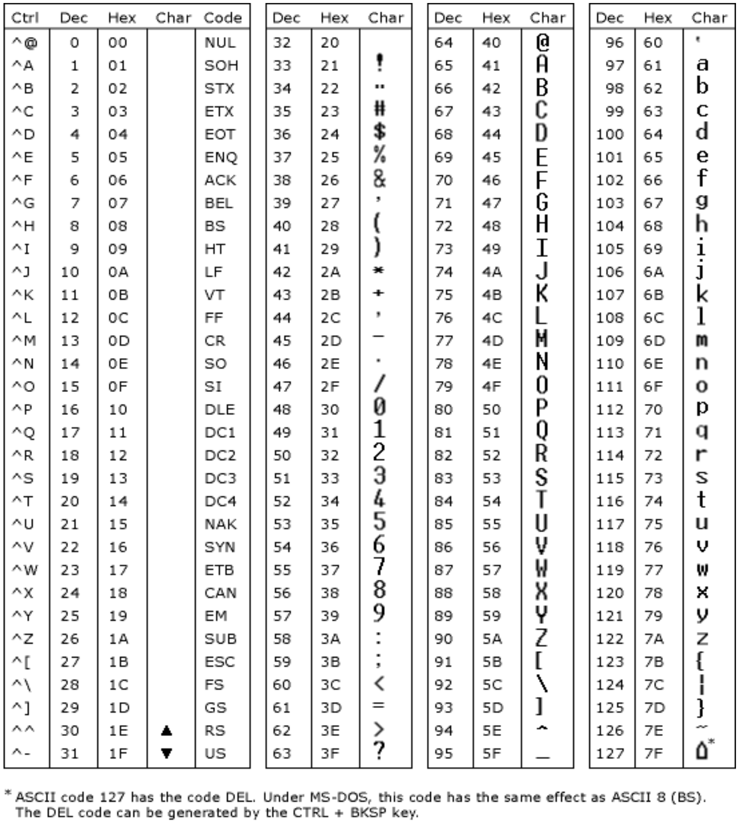
Least significant bit

- Information represented as a code of bits
- Two basic forms of binary exist:
  - Numerical binary representation
  - Tabulated binary encoding standards
    - ASCII
    - UNICODE
    - Machine Instructions
    - Etc.



Tabulated Binary Encoding Standards

- Arbitrary grouping of bits into patterns that have been formally agreed upon by international organizations to represent one item of information



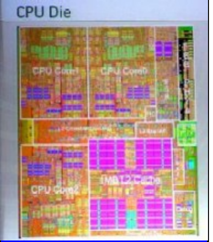
Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20		64	40	@	96	60	`
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	"	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	'	71	47	G	103	67	g
^H	8	08		BS	40	28	(	72	48	H	104	68	h
^I	9	09		HT	41	29	)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B		ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	|
^]	29	1D		GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂*

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

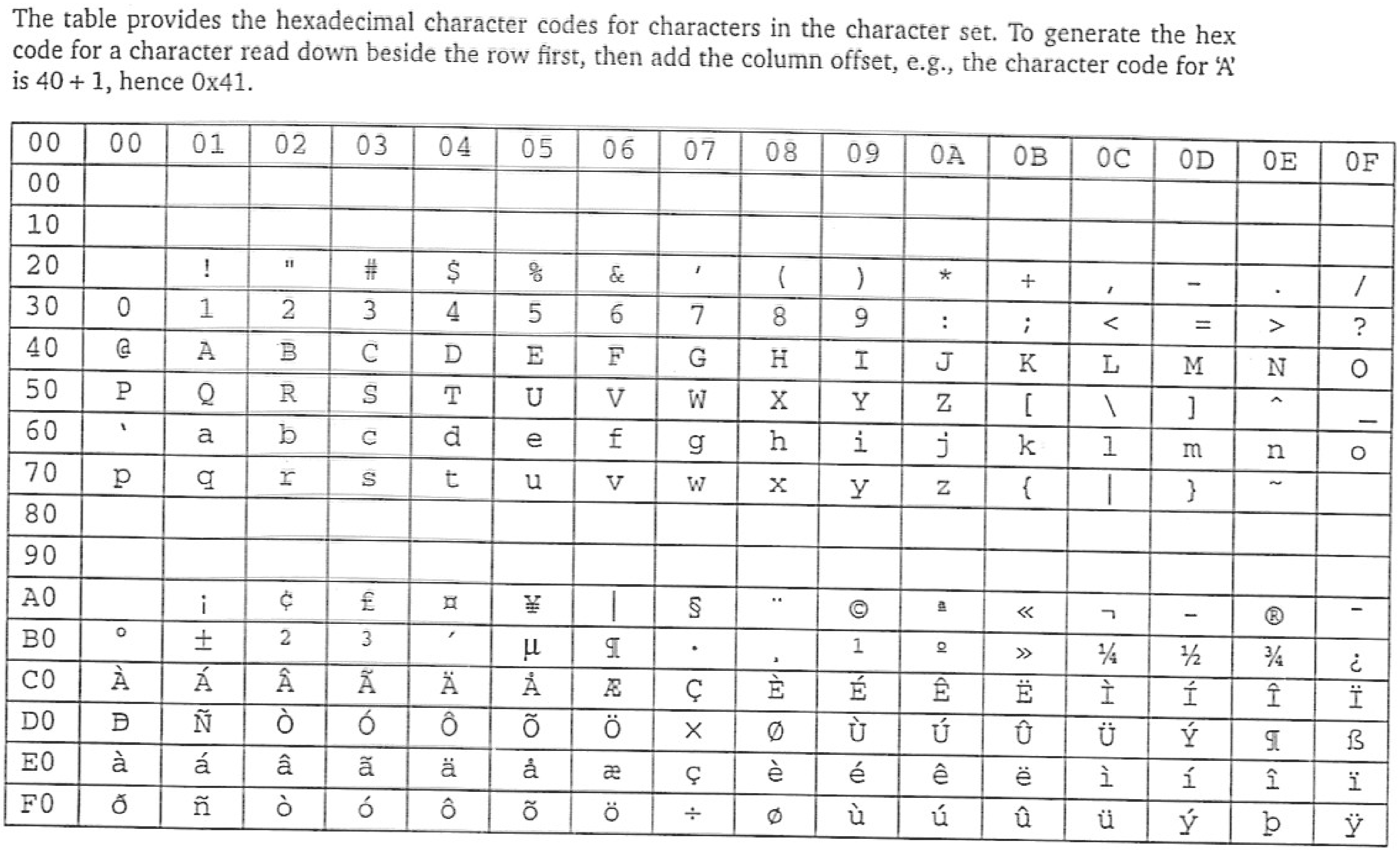
ASCII

SPACE = 32 = 00100000
A = 65 = 01000001
a = 97 = 01100001

8 bits



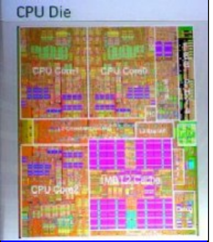
Extended ASCII Table



The table provides the hexadecimal character codes for characters in the character set. To generate the hex code for a character read down beside the row first, then add the column offset, e.g., the character code for 'A' is 40 + 1, hence 0x41.

00	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00																
10																
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
80																
90																
A0		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	Đ	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	¶	ß
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

We will talk about Hex to Binary conversions soon

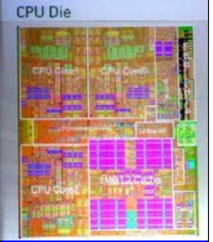


Character Data Codes

Name	Bits per Symbol	Total Symbols	Comments
BCD	6	64	A-Z; 0-9, $ … (only capitals)
ASCII	7	128	a-z; A-Z; 0-9; Bel, Tab, $, …
USASCII	8	256	Even parity bit for transmit
EBCDIC	8	256	Odd parity bit (only IBM)
UNICODE	16	65,536	Many languages

USASCII and Unicode are most popular

BCD = Binary Coded Decimal
ASCII = American Standard Code for International Interchange
EBCDIC = Extended Binary Coded Decimal Interchange Code
Unicode = Universal Code



COMP 273
Introduction to Computer Systems

Binary Integer Numbers

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

- Counting
- Addition
- Subtraction
- Multiplication
- Division

Not based on tabular encoding.
Mathematics based.