CMPE 310
Systems Design and Programming

L3: Chapter 1 – *Introduction to the Microprocessor and Computer*

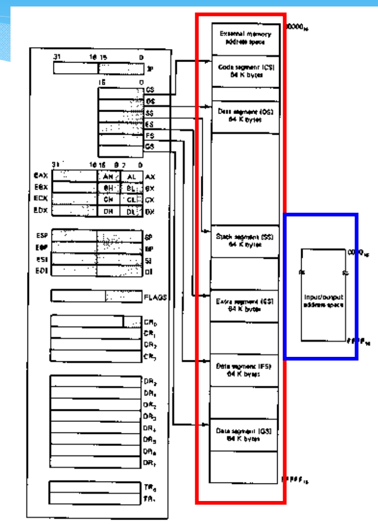
UMBC
AN HONORS UNIVERSITY IN MARYLAND

L3 Objectives

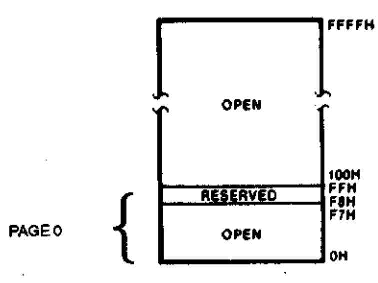
- Describe the memory address space
- Describe the IO space
- Describe how data is organized in the memory/ IO space
- Discuss the importance of data alignment
- Describe the data types

Memory and Input/Output

* Architecture implements independent memory and input/output address spaces
  * Why?
* Memory address space- 1,048,576 bytes long (1MB)
* Input/output address space- 65,536 bytes long (64KB)

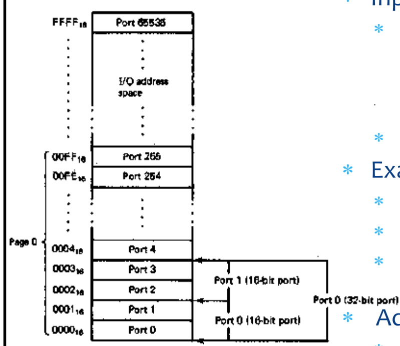


I/O Address Space



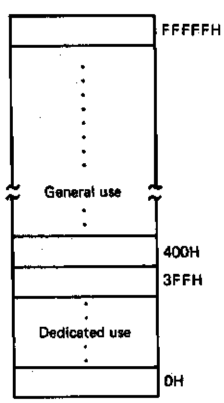
* I/O addresses are NOT memory-mapped addresses on the x86
* Input/output address space
  * Place where I/O devices are normally implemented
  * I/O addresses are only 16-bits in length
  * Independent 64K-byte address space
  * Address range 0000H through FFFFH
* Page 0
  * First 256 byte addresses → 0000H - 00FFH
  * Can be accessed with direct or variable I/O instructions
  * Ports F8H through FF reserved

Organization of the I/O Data



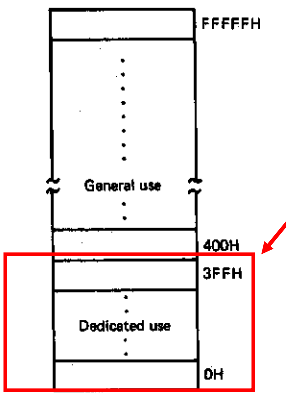
- Input/output data organization
  - Supports byte, word, and double-word I/O ports
    - 64K independent byte-wide I/O ports
    - 32K independent word-wide I/O ports
    - 16K independent double-word-wide I/O ports
  - **What determines the size use for a given port?**
- Examples (aligned I/O ports):
  - Byte ports 0,1,2 → addresses 0000H, 0001H, and 0002H
  - Word ports 0,1,2 → addresses 0000H, 0002H, 0004H
  - Double-word ports 0,1,2 → addresses 0000H, 0004H, 0008H
- Advantages of Isolated I/O
  - Complete memory address space available for use by memory
  - I/O instructions tailored to maximize performance
- Disadvantage of Isolated I/O
  - All inputs/outputs must take place between I/O port and accumulator register

Address Space



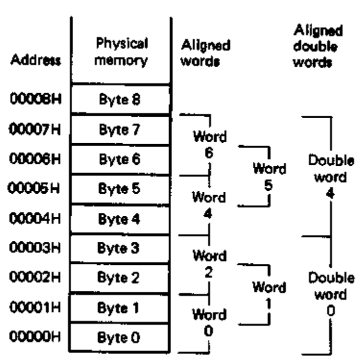
- Memory organized as individual bytes
- Memory address space corresponds to the 1M addresses in the range 00000H to FFFFFH
- 00000H= 00000000000000000000_2
- FFFFFH= 11111111111111111111_2
- 2^{20}= 1,048,576 = 1M unique addresses
- Data organization:
  - Byte: content of any individual byte address
  - Word: contents of two contiguous byte addresses
  - Double-word: contents of 4 contiguous byte addresses

Dedicated and General Use of Memory



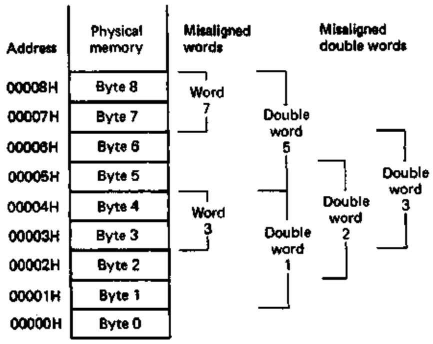
- Memory address space is partitioned into general use and dedicated use areas
- Dedicated (0H – 3FFH):
  - Interrupt vector table
  - 1st 1024 bytes
    - Addresses 0H → 3FFH
  - 256 4-byte pointers
    - 16-bit segment base address
    - 16-bit offset
- General use:
  - 400H → FFFFFH
  - Used for stack, code, and data

Aligned Words, Double words



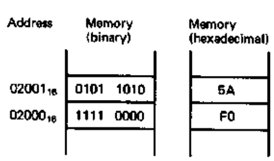
- Words and double words of data can be stored in memory at an even or odd address boundary
  - Examples of even address boundaries:
    00000_{16}, 00002_{16}, 00004_{16}
  - Examples of odd address boundaries:
    00001_{16}, 00003_{16}, 00005_{16}
- Aligned double-words are stored at **even addresses** that are a multiple of **4**
  - Examples: double words 0 and 4

Misaligned Words



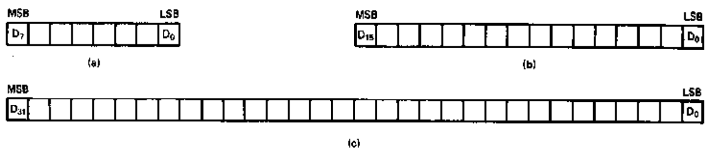
* 80x86 architecture supports access of aligned or misaligned data
* Words stored across a double-word boundary are said to be "misaligned or unaligned words"
  * Examples: words 3 and 7
* Misaligned double-words are stored at addresses that are not a multiple of 4
  * Examples: double words 1, 2, 3 and 5
* There is a performance impact for accessing unaligned data in memory (16-bit data bus)
  * Why?

Examples of Words of Data



* "Little endian" organization
* Most significant byte at high address
* Least significant byte at low address

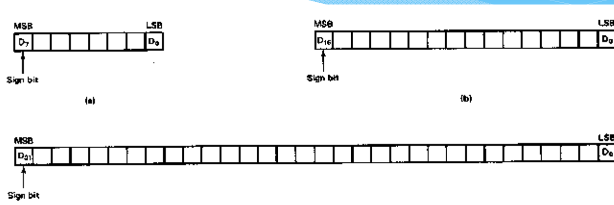
Unsigned Integers



All numbers are binary in memory
All bits represent data
Types:

Sizes	**Range**
8-bit	$0 \rightarrow 255_{10}$
16-bit	$0 \rightarrow 65{,}535_{10}$
32-bit	$0 \rightarrow 4{,}294{,}967{,}295_{10}$

Signed Integers



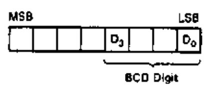
MSB is sign bit (0/1 → +/-)
Remaining bits represent value
Negative numbers expressed in 2's complement notation
Types:

Sizes	**Range**
8-bit	$-128 \rightarrow +127$
16-bit	$-32{,}768 \rightarrow +32{,}767$
32-bit	$-2{,}147{,}483{,}648 \rightarrow +2{,}147{,}483{,}647$

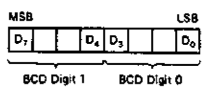
BCD Numbers

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

(a)



(b)



(c)

- Direct coding of numbers as binary coded decimal (BCD) numbers supported
- Unpacked BCD [Fig (b)]
  - Lower four bits contain a digit of a BCD number
  - Upper four bits filled with zeros (zero filled)
- Packed BCD [Fig (c)]
  - Lower significant BCD digit held in lower 4 bits of byte
  - More significant BCD digit held in upper 4 bits of byte
  - Example: At memory address 01000H is 10010001
    - **What number does that represent in BCD?**

Character Code

- ***ASCII***: American Standard Code for Information Interchange
  - Represent each character with a **7-bit string**
  - 128 different characters (see ASCII table)
  - Uppercase & lowercase alphabet, numerals, punctuation, nonprinting control characters
- Eighth bit may be used for parity (forcing number of 1s to be even or odd)
- Check out www.asciitable.com
- ASCII is somewhat eclipsed now by Unicode

Standard ASCII

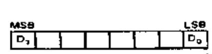
ASCII	MSBs - $b_6b_5b_4$							
$b_3b_2b_1b_0$	000 - 0	001 - 1	010 - 2	011 - 3	100 - 4	101 – 5	110 - 6	111 - 7
0000 – 0	NUL	DLE	SP	0	@	P	`	p
0001 – 1	SOH	DC1	!	1	A	Q	a	q
0010 – 2	STX	DC2	"	2	B	R	b	r
0011 – 3	ETX	DC3	#	3	C	S	c	s
0100 – 4	EOT	DC4	$	4	D	T	d	t
0101 – 5	ENQ	NAK	%	5	E	U	e	u
0110 – 6	ACK	SYN	&	6	F	V	f	v
0111 – 7	BEL	ETB	'	7	G	W	g	w
1000 – 8	BS	CAN	(	8	H	X	h	x
1001 – 9	HT	EM	)	9	I	Y	i	y
1010 – A	LF	SUB	*	:	J	Z	j	z
1011 – B	VT	ESC	+	;	K	[	k	{
1100 – C	FF	FS	,	<	L	\	l	|
1101 – D	CR	GS	-	=	M	]	m	}
1110 – E	SO	RS	.	>	N	^	n	~
1111 - F	SI	US	/	?	O	_	o	DEL

- "Yeccch!"
- 1011001 1100101 1100011 1100011 1100011 1101000 01000001

ASCII Data

		b_7	0	0	0	0	1	1	1	1
		b_6	0	0	1	1	0	0	1	1
		b_5	0	1	0	1	0	1	0	1
b_4 b_3 b_2 b_1	H_0 \ H_1		0	1	2	3	4	5	6	7
0 0 0 0	0		NUL	DLE	SP	0	@	P	'	p
0 0 0 1	1		SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2		STX	DC2	"	2	B	R	b	r
0 0 1 1	3		ETX	DC3	#	3	C	S	c	s
0 1 0 0	4		EOT	DC4	$	4	D	T	d	t
0 1 0 1	5		ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6		ACK	SYN	&	6	F	V	f	v
0 1 1 1	7		BEL	ETB	'	7	G	W	g	w
1 0 0 0	8		BS	CAN	(	8	H	X	h	x
1 0 0 1	9		HT	EM	)	9	I	Y	i	y
1 0 1 0	A		LF	SUB	*	:	J	Z	j	z
1 0 1 1	B		VT	ESC	+	;	K	[	k	}
1 1 0 0	C		FF	FS	,	<	L	\	l	|
1 1 0 1	D		CR	GS	–	=	M	]	m	{
1 1 1 0	E		SO	RS	.	>	N	^	n	~
1 1 1 1	F		SI	US	/	?	O	-	o	DEL

(a)



(b)

- American Code for Information Interchange (ASCII) code
- ASCII information storage in memory
  - Coded one character per byte
  - 7 LS-bits = $b_7b_6b_5b_4b_3b_2b_1$
  - MS-bit filled with 0
  - **How will we represent 'A' in ASCII?**

So we discussed…

* Describe the memory address space
* Describe the IO space
* Describe how data is organized in the address/ IO space
* Discuss the importance of data alignment
* Describe the data types

Next time

* 80x86 Hardware Specification

STOP