CS 271
Computer Architecture & Assembly Language

Lecture 6

Debugging & Internal/External Data Representation

1/20/22, Thursday

Oregon State University

Due Reminder

- Sunday 1/23 11:59 PM via Canvas
  - Program #2 (.asm file)
  - Quiz 1
  - Summary Exercise 3

Recap:

- Convert the following to MASM instructions

```
for (k = 10; k <= n; k++)
        print yes;
```

Recap:

- Convert the following to high level pseudo-code

```
        mov     eax, k
again:
        cmp     eax, n
        jge     done
        mov     edx, offset no
        call    WriteString
        inc     eax
        jmp     again
done:
```

Lecture Topics:

- Using a Debugging System
- Data Representation

Debugging MASM

- Demo

Debugging MASM

- Set breakpoints by clicking in the left margin
- “Debug”, “Start Debugging”
  - Execution will pause at the first breakpoint
- “Debug”, “Windows”, “Registers”
  - To view register contents
  - Register contents are shown in hexadecimal (base 16)
- Use F10 (for now) to execute one instruction
  - Watch register contents
  - Changes in red

Debugging MASM

- Helps to locate logic errors
- Helps to really understand what the MASM statements do
- Hints:
  - Don’t use F11 to step into Irvine library calls (for now)
  - You might have to switch back and forth between the code screen and the I/O screen
  - If you make changes, remember to “Stop Debugging” before you restart the modified program
- Experiment!! With other debugging windows, etc.

Debugging MASM

- Step Over (F10)
  - If no breakpoint, starts at the first statement in **`main`**
  - Execute next instruction
  - If instruction is **`call`**, executes entire called procedure
  - Use this to step over library procedures!
- Step Into (F11)
  - If no breakpoint, starts at the first statement in **`main`**
  - Execute next instruction
  - If instruction is **`call`**, goes to first instruction in called procedure
  - Don’t step into library procedures!

Debugging MASM

- Other useful Debug menu items
  - Before debug session
    - Start Without Debugging
  - During debug session
    - Continue (F5) runs to next breakpoint
    - Restart
    - Stop Debugging

    - Others

Data Representation

Internal Representation

- Just like everything else in a computer, the representation of data is implemented electrically
  - Switches set to off or on
  - With open/closed gates
- There are two states for each gate
- The binary number system used two digits (0 and 1)
- In order to simplify discussion, we use the standard external representation to transcribe the computer's internal representation:
  - off is written as digit 0
  - on is written as digit 1

Internal Representation

- Use the binary number system to represent numeric values electrically
- Switches (gates) are grouped into bytes, words, etc., to represent a numerical value in the binary system
- Note: the number of gates in a group depends on the computer architecture and the type of data represented.
- E.g., for most architectures:
- byte = 8 bits, word = 2 bytes (bits), etc.

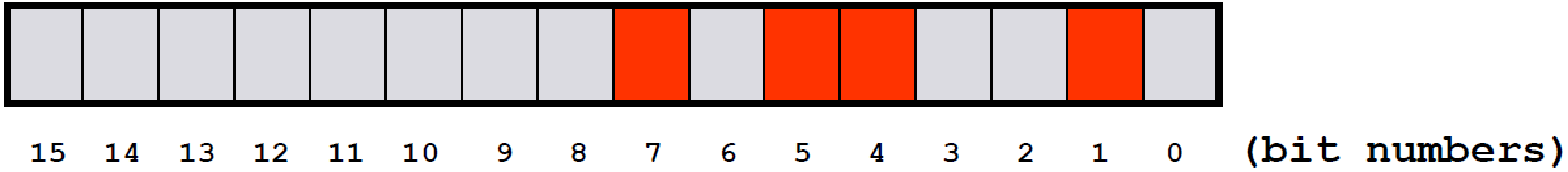
External Representation

- Binary Number System
  - Has 2 digits: 0 and 1 (binary digit)
  - Has places and place values determined by powers of 2.

- (In theory) can uniquely represent any integer value
  - A binary representation is just another way of writing a number that we are accustomed to seeing in decimal form.
- (In practice, inside the computer) representation is finite
  - Representations with too many digits get truncated

Binary Representation

- Place values (right-to-left) are 2^0, 2^1, 2^2, 2^3, 2^4, etc.
- Bits are numbered (right-to-left) starting at 0
- Place value depends on number of “bits” defined for the type.
- Example:
  - A 16-bit integer might be (red is “on”, red = 1)



  - … transcribed by a human as 0000000010110010
  - To convert to its familiar decimal representation, just add up the place values of the places that are “on”.

Converting Binary to Decimal

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0

- In decimal form:
- 128 + 32 + 16 + 2 = 178

Converting Decimal to Binary

- Example: 157
- Method 1: Removing largest powers of 2
  - 157 – 128 = 29 — 1 in 128s place
  - 29 – 16 = 13 — 0 in 64s place, 0 in 32 place, 1 in 16s place
  - 13 – 8 = 5 — 1 in 8s place
  - 5 – 4 = 1 — 1 in 4s place
  - 1 – 1 = 0 — 0 in 2s place, 1 in 1s place
  - 1 0 0 1 1 1 0 1
- Method 2: Successive division by 2
  - 157 ÷ 2 = 78 R 1
  - 78 ÷ 2 = 39 R 0
  - 39 ÷ 2 = 19 R 1
  - 19 ÷ 2 = 9 R 1
  - 9 ÷ 2 = 4 R 1
  - 4 ÷ 2 = 2 R 0
  - 2 ÷ 2 = 1 R 0
  - 1 ÷ 2 = 0 R 1
  - 1 0 0 1 1 1 0 1 (Write remainders, bottom to top)

Numeric Representation

- We will show (later) exactly how an electrical operation can be performed on two electrical numeric representations to give an electrical result that is consistent with the rules of arithmetic.

Other (external) representations

- Every integer number has a unique representation in each “base” > 2
- Hexadecimal is commonly used for easily converting binary to a more manageable form.
  - Because $16 = 2^4$, so 4 binary digits can be represented as one hex digit.
- The hexadecimal number system has 16 digits:
  - 0 1 2 3 4 5 6 7 8 9 A B C D E F
- Place values (right-to-left) are 16^0, 16^1, 16^2, 16^3, 16^4, etc.
  - $16^0 = 2^0$, $16^1 = 2^4$, $16^2 = 2^8$, $16^3 = 2^{12}$, $16^4 = 2^{16}$, etc.

Decimal	Binary	Hexadecimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14
30	11110	1E
40	100100	28

Hexadecimal Conversion

- Example: 6077 (decimal)
- 16-bit binary ←→ hexadecimal:
- Binary 0001 0111 1011 1101
- Hexadecimal 1 7 B D

- Write it as 0x17BD or 17BDh

Converting Decimal ←→ Hexadecimal

- Use same methods as decimal ←→ binary
  - The only difference is the place values
- Example
  - 157 (decimal) = 9D (hex) (0X9D or 9Dh)
- … or convert to binary, then to hex

Representing negative integers

- Must specify size!
  - Specify n: number of bits (8, 16, 32, etc.)
  - There are 2^n possible "codes"

- Separate the "codes" so that half of them represent negative numbers.
  - Note that exactly half of codes have 1 the "leftmost" bit.

Binary form of negative numbers

- Several methods, each with disadvantages.
- We will focus on twos-complement form

- For a negative number x:
  - Specify number of bits
  - Start with binary representation of |x|
  - Change every bit to its opposite, then add 1 to the result.

Binary form of negative numbers

- Example: -13 in 16-bit twos-complement
  - |-13| = 13 = 0000 0000 0000 1101
  - Ones-complement is 1111 1111 1111 0010
  - Add 1 to get 1111 1111 1111 0011 = -13
- Note that –(–13) should give 13. Try it ☺

- Hex representation?
  - Convert binary to hex in the usual way
  - -13 = 1111 1111 1111 0011 = FFF3 h = 0xFFF3

- Convert negative binary to decimal?
  - Find twos complement, convert, and prepend a minus sign

Signed numbers using 4-bit Twos-complement form

-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111

- Notice that all of the negative numbers have 1 in the leftmost bit. All of the non-negative numbers have 0 in the leftmost bit.
  - For this reason, the leftmost bit is called the **sign bit**
- Note: Nobody uses 4-bit representations (“nibble”).
  - Common: 8-bit, 16-bit (extend this diagram yourself ☺)

n-bit twos-complement form

- The 2^n possible codes give
  - Zero (all bits are 0)
  - $(2^{n-1} – 1)$ positive numbers
  - 2^{n-1} negative numbers
- Note: 0 is its own complement
- Note” there is one “weird” number (example: n = 8)
  - 0111 1111 + 1 = 1000 0000
  - 127 + 1 = -128 (inconsistent with rules of arithmetic)
  - 127 is the largest number that can be represented in 8 bits. This means that –(-128) cannot be represented with 8 bits.
    - i.e., the 2’s-complement of 1000 0000 is 1000 0000

Signed or Unsigned?

- A 16-bit representation could be used for signed or signed numbers
  - 16-bit **unsigned** range is 0…65535
  - 16-bit **signed** range is -32768…+32767
- Both forms use the same 65536 codes ($2^{16} = 65536$)
- Example:
  - 1010 1010 1010 1010 **unsigned** is 43690 decimal
  - 1010 1010 1010 1010 **signed** is -21846 decimal
- Example:
  - 1111 1111 1111 1111 **unsigned** is 65535 decimal
  - 1111 1111 1111 1111 **signed** is -1 decimal
- Programs tell the computer which form is being used

Negative Hex (signed integers)

- How can you tell if a hex representation of a signed integer is negative?
  - Recall that a 16-bit signed integer is negative if the leftmost bit is 1
- 16-bit (4 hex digits) examples:
  - 0x7A3E is positive
  - 0x8A3E is negative
  - 0xFFFF is negative

Exercise

- Convert decimal -2345 to:
  16-bit binary:
  4-digit hex:

- Convert signed integer 0xACE9 to:
  binary:
  decimal:

Character and Control Codes

- Letters, digits, special characters … are represented internally as numbers
- ASCII 256 codes (1-byte)
  - E.g., 'A' … 'Z' are codes 65-90
  - E.g., '0' … '9' are codes 48-57
- Unicode 65,536 codes (2-byte)
- Some codes are used for controlling devices
  - E.g., code 10 is "new line" for output device
  - E.g., code 27 is Esc ("escape" key)
- Device controllers translate codes (device-dependent)
- All keyboard input is <u>character</u> (including digits)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Digits

- Digits entered from the keyboard are characters
  - E.g., ‘0’ is character number 48, … ‘9’ is character number 57

- What happens if we add ‘3’ + ‘5’?
  - The answer is 51 + 53 = 104 → ‘h’

- Numeric data types require conversion by the input/output operations

Neutral Representation

- Inside the computer
  - Bytes, words, etc., can represent a finite number of combinations of off/on switches.
  - Each distinct combination is called a code.
  - Each code can be used to represent:
    - Numeric value
    - Memory address
    - Machine instruction
    - Keyboard character
    - Other character
- Representation is neutral
  - The operating system and the programs decide how to interpret the codes.

Interpreting Codes

- It is especially important to learn to interpret hexadecimal (external representation) codes.
  - Frequently used by assembly and debugging systems

- If you need help with binary and/or hexadecimal, google online or ask TA or instructor for help.