CS 271 Computer Architecture and Assembly Language

Self-Check for Lecture#15

Solutions

Given the following partial data segment, which starts at address 0x0200 :

```
.data
list      DWORD       1, 2, 6, 24, 120, 720, 5040, 40320
x         DWORD       LENGTHOF list
y         DWORD       SIZEOF list
```

Show addresses in 4-digit hexadecimal.
Show contents in decimal.

1. *x* contains **8**

2. *y* contains **32**

3. The address of *x* is **0x220** (hex)
   Address of list + SIZEOF list = 0x200 + 0x20 = 0x220 (Note: 32 decimal = 0x20)

4. Given this code fragment:

```
mov   esi, OFFSET list          ; esi ← 0x200
mov   eax, [esi+5*TYPE list]    ; TYPE list = 4, so eax gets the contents of
                                ;       esi+20, which is the 6th element of list
```

   eax contains **720**

5. Given this code fragment:

```
mov   esi, OFFSET list          ; esi ← 0x200
mov   ebx, y                    ; ebx ← 32
sub   ebx, TYPE y               ; ebx ← 28   ( = 0x1C )
add   esi, ebx                  ; esi ← 0x21C, so esi contains the address of
                                ;       the 8th element of list
```

   [*esi*] contains **40320**

6. Given this code fragment:

```
mov   esi, OFFSET list
mov   ebx, y
sub   ebx, TYPE y
add   esi, ebx                  ; same as #5 code
mov   al, BYTE PTR [esi+1]      ; [esi] contains 40320.  esi+1 is the address
                                ;       of the 2nd byte.
```

   The AL register contains **157**

40320 decimal = 0x9D80, or 00 00 9D 80 in big-endian.
In the little-endian IA-32, it's 80 9D 00 00.
So the 2nd byte is 0x9D = 157

Given the following partial data segment, which starts at address 0x0200 :

```
.data
matrix    DWORD       20 DUP(5 DUP(?))
x         DWORD       LENGTHOF matrix
y         DWORD       SIZEOF matrix
pal       BYTE        "Hello world.",0
len       DWORD       LENGTHOF pal
```

Show addresses in 4-digit hexadecimal.
Show contents in decimal.

1. *x* contains **100**

2. *y* contains **400**

3. In high-level language notation, the 3rd element of the 9th row is referenced as *matrix*[8][2].

   The address of *matrix*[8][2] is **0x2A8** (hex)
   BaseAddress + elementSize * [(row# * elementsPerRow) + column#]
   0x0200 + 4 * [(8 * 5) + 2]
   0x0200 + (decimal 4 * 42 = 168)
   0x0200 + 0xA8 = 0x2A8

4. Given this code fragment:

```
        mov   esi, OFFSET pal
        mov   ecx, len
        sub   ecx, 2
        cld
one:
        lodsb
        call  WriteChar
        loop  one
        mov   ecx, len
        sub   ecx, 2
        std
two:
        lodsb
        call  WriteChar
        loop  two
```

   WriteChar displays the character in the AL register.
   What is displayed?

   Hello world.dlrow olle

All keyboard input is character. The keyboard digits, '0', '1', '2', …, '9' are ASCII codes 48, 49, 50, …, 57. When a user enters a numeric value, it comes into memory as a string of digits.

This exercise is intended to show why it's necessary to convert to numeric representation before using the string of digits. An ASCII table is provided below.

To add two digit strings, you have to be sure that the strings both have the same length by zero-filling the shorter string on the left. Carry digits would be an additional nightmare. This example ignores both of those problems by adding two strings of equal length with no carry digits.

What is the (string) result of adding the following digit string, digit by digit?

```
a       BYTE        "2458",0
b       BYTE        "6301",0
```

"hgei"
"8" + "1" = 56 + 49 = 105 = "i"
"5" + "0" = 53 + 48 = 101 = "e"
"4" + "3" = 52 + 51 = 103 = "g"
"2" + "6" = 50 + 54 = 104 = "h"

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

Source: www.LookupTables.com