LABORATORY TOPICS

Basic Input/Output

Basic Input/Output

Interrupts

- can be a hardware or software interrupt
- the printer is out of paper
  - hardware interrupt
- assembly program issuing a system call
  - software interrupt

An interrupt simply gets the attention of the processor so that it would context switch and execute the system's interrupt handler being invoked.

Basic Input/Output

Interrupts

- break the program execution cycle (Fetch-Decode-Execute)
- wait for interrupt to be serviced

Linux services

- int 0x80, int 80H – function calls

INT 80H

- system call numbers for this service are placed in EAX

- 1 – sys_exit (system exit)
- 3 – sys_read (read from standard input)
- 4 – sys_write (write to standard output)

INT 80H

```
mov eax, 1
mov ebx, 0
int 80H
```

- terminate program
- return 0 at the end of main program (no error)

INT 80H

```
mov eax, 3
mov ebx, 0
mov ecx, <address>
int 80H
```

- reads user input
- stores input to a variable referenced by address

INT 80H

```
mov eax, 4
mov ebx, 1
mov ecx, <string>
mov edx, <length>
int 80H
```

- outputs a character/string
- character/string must be in ECX and string length must be in EDX before issuing interrupt

Basic Input/Output

sample.asm

```
1   section .data
2       hello       db 'Hello world!',10
3       helloLen    equ $-hello
4
5   section .text
6       global _start
7   _start:
8
9       mov eax,4
10      mov ebx,1
11      mov ecx,hello
12      mov edx,helloLen
13      int 80h
14
15      mov eax,1
16      mov ebx,0
17      int 80h
18
```

Instructions

Basic Input/Output

sample.asm

```
1  section .data
2     hello       db 'Hello world!',10
3     helloLen    equ $-hello
4
5  section .text
6     global _start
7  _start:
8
```

Basic Input/Output

```
 9      mov eax,4
10      mov ebx,1
11      mov ecx,hello
12      mov edx,helloLen
13      int 80h
14
15      mov eax,1
16      mov ebx,0
17      int 80h
18
```

Basic Input/Output

sample.asm

```
1   section .data
2      prompt         db 'Enter your name: '
3      promptLen      equ $-prompt
4
5   section .bss
6      name        resb 10
7
8   section .text
9      global _start
10  _start:
11
12     mov eax,4
13     mov ebx,1
14     mov ecx,prompt
15     mov edx,promptLen
16     int 80h
17
18     mov eax,3
19     mov ebx,0
20     mov ecx,name
21     int 80h
22
23     mov eax,1
24     mov ebx,0
25     int 80h
```

Basic Input/Output

sample.asm

```
1   section .data
2       prompt          db 'Enter your name: '
3       promptLen       equ $-prompt
4
5   section .bss
6       name        resb 10
7
8   section .text
9       global _start
10  _start:
11
```

Basic Input/Output

```
12      mov eax,4
13      mov ebx,1
14      mov ecx,prompt
15      mov edx,promptLen
16      int 80h
17
18      mov eax,3
19      mov ebx,0
20      mov ecx,name
21      int 80h
22
23      mov eax,1
24      mov ebx,0
25      int 80h
```

Converting Characters to Numbers

- Input read from the keyboard are ASCII-coded characters.
- Output displayed on screen are ASCII-coded characters.
- To perform arithmetic, a numeric character must be converted from ASCII to its equivalent value.
- To print an integer, a number must be converted to its equivalent ASCII character(s).

Converting Characters to Numbers

Character to Number:

```
section .bss
  num resb 1
section .text
  mov eax, 3
  mov ebx, 0
  mov ecx, num
  int 80h
```

Converting Characters to Numbers

Character to Number:

```
section .bss
  num resb 1
section .text
  mov eax, 3
  mov ebx, 0
  mov ecx, num
  int 80h

  sub [num], 30h
```

ASCII Table

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
32	20	040		Space	64	40	100	@	@	96	60	140	`	`
33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

ASCII Table

Dec	Hx	Oct	Html	Chr
48	30	060	&#48;	0
49	31	061	&#49;	1
50	32	062	&#50;	2
51	33	063	&#51;	3
52	34	064	&#52;	4
53	35	065	&#53;	5
54	36	066	&#54;	6
55	37	067	&#55;	7
56	38	070	&#56;	8
57	39	071	&#57;	9

Converting Characters to Numbers

Character to Number:

```
mov ecx, num
int 80h
sub [num], 30h
```

num	=	0	0	1	1	0	1	0	0
30h	=	0	0	1	1	0	0	0	0
num	=	0	0	0	0	0	1	0	0

Converting Numbers to Characters

Number to Character:

```
section .text
  mov eax, 4
  mov ebx, 1
  mov ecx, num
  mov edx, 1
  int 80h
```

Converting Numbers to Characters

Number to Character:

```
section .text
  add [num], 30h

  mov eax, 4
  mov ebx, 1
  mov ecx, num
  mov edx, 1
  int 80h
```

Converting Numbers to Characters

Number to Character:

```
add [num], 30h
```

num =	0	0	0	0	0	1	1	0
30h =	0	0	1	1	0	0	0	0
num =	0	0	1	1	0	1	1	0

LABORATORY TOPICS

Implementing Sequential Statements

Assignment Instruction

- mov instruction

  mov destination, source

- mov is a storage command
- copies a value into a location
- destination may be register or memory
- source may be register, memory or immediate
- operands should not be both memory
- operands must be of the same size

Simple Instructions

- **inc** destination
  - increase destination by 1
- **dec** destination
  - decrease destination by 1
- **neg** destination
  - negate destination (2's complement)
- destination may be a register or a memory

Add and Sub Instructions

- **add** destination, source

  destination = destination + source

- **sub** destination, source

  destination = destination – source

- same restrictions as the mov instruction

Add and Sub Instructions

```
add eax, 5
```

eax = eax + 5

```
sub [num], bx
```

[num] = [num] – bx

Multiplication

- **mul** source
- Source is the multiplier

- Source may be a register or memory
- If source is byte-size then

  AX = AL * source
- If source is word-size then

  DX:AX = AX * source
- If source is double word-size then

  EDX:EAX = EAX * source

Multiplication

```
mov al, 2
mov [num], 80H
mul byte [num]
```

Multiplication

```
mov al, 2
mov [num], 80H
mul byte [num]
```

- This results in AX = 2 * 128 = 256 = 0100 H

Multiplication

```
mov al, 2
mov [num], 80H
mul byte [num]
```

- This results in AX = 2 * 128 = 256 = 0100 H
- In Binary, AX = 0000000100000000 B

Multiplication

```
mov al, 2
mov [num], 80H
mul byte [num]
```

- This results in AX = 2 * 128 = 256 = 0100 H
- In Binary, AX = 0000000100000000 B
- So, AH = 1 and AL = 0.

Multiplication

```
mov ax, 2
mov [num], 65535
mul word [num]
```

- This results in 2 * 65535 = 131070
  = 0001FFFE H.
- DX = 1 and AX = FFFE H
- Only when taken together will one get the correct product.

Division

- **div** source
- Source is the divisor
- Source may be a register or memory
- If source is byte-size then

  AH = AX mod source

  AL = AX div source

Division

- If source is word-size then
  DX = DX:AX mod source
  AX = DX:AX div source
- If source is double word-size then
  EDX = EDX:EAX mod source
  EAX = EDX:EAX div source
- div == integer division
- mod == modulus operation (remainder)

Division (Divide word by byte)

```
mov ax, 257
mov [num], 2
div byte [num]
```

Division (Divide word by byte)

```
mov ax, 257
mov [num], 2
div byte [num]
```

- This results in AH = 1 and AL = 128

Division (Divide byte by byte)

```
mov [num], 129
mov al, [num]
mov ah, 0
mov [num], 2
div byte [num]
```

Division (Divide byte by byte)

```
mov [num], 129
mov al, [num]
mov ah, 0
mov [num], 2
div byte [num]
```

- AX = 129 since AH = 0 and AL = 129

Division (Divide byte by byte)

```
mov [num], 129
mov al, [num]
mov ah, 0
mov [num], 2
div byte [num]
```

- AX = 129 since AH = 0 and AL = 129
- This results in AH = 1 and AL = 64

Division (Divide word by word)

```
mov dx, 0
mov ax, 65535
mov [num], 32767
div word [num]
```

- DX:AX = 65535
- num = 32767
- This results in DX = 1 and AX = 2.

Division (Divide word by word)

```
mov dx, 1
mov ax, 65535
mov [num], 65535
div word [num]
```

- Taken together the values in DX and AX is 131071.
- This is divided by 65535.
- The results are in DX = 1 and in AX = 2.

Divide Overflow Error

```
mov ax, 65535
mov [num], 2
div byte [num]
```

- Dividing 65535 by 2 results in a quotient of 32767 with a remainder of 1.
- The value 32767 will not fit in AL, the destination of the quotient, thus resulting in an error.

Let's Try!

Sample: scanf("%c",&x);

```
mov eax, 3
mov ebx, 0
mov ecx, x
int 80h
```

Let's Try!

1. printf("%c",x);

2. sum = x + y;

3. prod = x * y;

   (assume x and y are byte-size variables)

Let's Try!

1. printf("%c",x);

```
mov eax, 4
mov ebx, 1
mov ecx, x
mov edx, 1
int 80h
```

Let's Try!

2. sum = x + y;

```
mov bl, [x]
add bl, [y]
mov [sum], bl
```

Let's Try!

3. prod = x * y;

```
mov al, [x]
mul byte[y]
mov [prod], ax
```