I.T.B.A

Arquitectura de Computadoras
Primer Parcial

Fecha: 4/Oct/2017

Alumno:
Nro. de Legajo:

Condición mínima de aprobación: 2 puntos del Ejercicio 1, 1 punto del Ejercicio 2 y 1 punto del Ejercicio 3

Ejercicio 1 (4 puntos)

Se dispone de un microprocesador de tipo Intel (tiene salida IO/M) de 8 bits de bus de datos y 32 bits de bus de direcciones. El dueño del proyecto nos brinda 2 (dos) módulos de memoria RAM, cada uno de ellos de 1G x 8 y nos pide que los ubiquemos a partir de la dirección 0h en forma contigua para lograr 2GBytes de RAM.

Además tenemos que decodificar un módulo de ROM de 256M x 8 para el código que corre al encender el sistema, pero el único dato que tenemos es que el registro EIP al iniciarse el procesador toma el valor FFFFFF00h. El dueño del proyecto nos avisa que su código ocupa 200 MBytes.

Por una cuestión de espacio y economía Ud. solo puede usar hasta **3 decodificadores como máximo** para resolver la decodificación completa de RAM y ROM. Pueden ser decodificadores de 2 a 4, 3 a 8 y 4 a 16. No dispone de compuertas para la decodificación.

Se pide **en este orden** que:

1) Dibuje el mapa de memoria.
2) Dibuje un diagrama esquemático de los integrados utilizados para la RAM y ROM (líneas de datos, direcciones, etc) Debe colocar nombres a **todas** las líneas de circuito utilizadas.
3) Resuelva la decodificación con estas condiciones y dibuje el circuito obtenido.
4) ¿Qué pedido le tienen que hacer Ud. al programador del código que irá en la ROM para que todo esto funcione correctamente al iniciarse el sistema?

Ejercicio 2 (3 puntos)

En base al ejercicio anterior, le pide a Ud. un grupo de desarrolladores que escriba una función en Assembler de Intel. Ud. debe realizar una rutina que será llamada desde C y tiene el siguiente formato:

Int imprime_pantalla(char encabezado, int tam_enc, char* pie, int tam_pie)*

Esta rutina debe imprimir dos cadenas de caracteres en una pantalla de 480 caracteres monocromáticos (80 columnas y 6 filas) que se encuentra mapeada a partir de la dirección de memoria A0000000h. Cada posición de memoria representa un carácter de la pantalla.

La rutina imprime una cadena llamada *encabezado* en la primera fila de la pantalla y otra cadena llamada *pie* en la última fila de la pantalla. Además recibe el tamaño de cada cadena a imprimir el cual es un número que puede ir de 1 a 80 como máximo. Si la rutina recibe un valor fuera de ese rango retorna el valor 1. Si la rutina se ejecuta sin problemas retorna el valor cero.

Como caso de uso y para contestar lo pedido en el ejercicio Ud. probará con la cadena encabezado *"Este es el título"* con 17 como tamaño de cadena, y la cadena pie "Fin de mensaje" con 14 de tamaño.

Se pide **en este orden** que:

1) Dibuje el estado de la pila al iniciar la rutina con cada uno de las direcciones de memoria y contenidos de la pila, utilizando el caso de uso propuesto.
2) Realice la rutina en Assembler, pero **solo implemente** la parte que imprime la cadena *pie*. NO escriba la parte de código que imprime el encabezado.

Arquitectura de Computadoras
Primer Parcial

Ejercicio 3 (3 puntos)

Ud. encuentra este código en un sitio de Internet para repasar el concepto y uso de system calls, el programador intentó realizar una rutina que lea una tecla de standar input y que pasarla de minúscula a mayúscula, pero no funciona. Ud. debe encontrar y explicar todos los errores del código y mostrar la solución para que funcione correctamente.

```
section .bss
section .data
        Buff resb 1

section .text
        global _start
_start:

Read:   mov eax,3               ; Specify sys_read call
        mov ebx,0               ; Specify File Descriptor 0: Standard Input
        mov ecx,[Buff]          ; Pass address of the buffer to read to
        mov edx,1               ; Tell sys_read to read one char from stdin
        int 80h                 ; Call sys_read
        cmp eax,0               ; Look at sys_read's return value in EAX
        je Exit                 ; Jump If Equal to 0 (0 means EOF) to Exit
                                ; or fall through to test for lowercase
        cmp byte [Buff],61h     ; Test input char against lowercase 'a'
        jb Write                ; If below 'a' in ASCII chart, not lowercase
        cmp byte [Buff],7Ah     ; Test input char against lowercase 'z'
        ja Write                ; If above 'z' in ASCII chart, not lowercase
                                ; At this point, we have a lowercase character
        sub byte [Buff],20h     ; Subtract 20h from lowercase to give uppercase...
                                ; ...and then write out the char to stdout
Write:  mov eax,4               ; Specify sys_write call
        mov ebx,1               ; Specify File Descriptor 1: Standard output
        mov ecx,Buff            ; Pass address of the character to write
        mov edx,1               ; Pass number of chars to write
        int 80h                 ; Call sys_write...
        jmp Read                ; ...then go to the beginning to get another character

Exit:   mov eax,1               ; Code for Exit Syscall
        mov ebx,0               ; Return a code of zero to Linux
        int 80H                 ; Make kernel call to exit program
```

Por otro lado encontró en la documentación de Linux, la siguiente información y pudo verificar que es correcta con respecto a los system call y la tabla ascii

3. sys_read

Syntax: `ssize_t sys_read(unsigned int fd, char * buf, size_t count)`

Source: fs/read_write.c

Action: read from a file descriptor

1. sys_exit

Syntax: `int sys_exit(int status)`

Source: kernel/exit.c

Action: terminate the current process

4. sys_write

Syntax: `ssize_t sys_write(unsigned int fd, const char * buf, size_t count)`

Source: fs/read_write.c

Action: write to a file descriptor

Arquitectura de Computadoras
Primer Parcial

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

ASM en Linux - Syscalls

Por ejemplo en **pampero** el listado de los system call se encuentra en "/usr/include/asm".

Nombre	ID
#define __NR_exit	1
#define __NR_fork	2
#define __NR_read	3
#define __NR_write	4
#define __NR_open	5
#define __NR_close	6
#define __NR_waitpid	7
#define __NR_creat	8
#define __NR_link	9
#define __NR_unlink	10
#define __NR_execve	11

#define __NR_chdir	12
#define __NR_time	13
#define __NR_mknod	14
#define __NR_chmod	15
#define __NR_lchown	16
#define __NR_break	17
#define __NR_oldstat	18
#define __NR_lseek	19
#define __NR_getpid	20
#define __NR_mount	21
#define __NR_umount	22

4294967296

Facial

0 0000
3 0011
4 0100
7 0111
F 1111

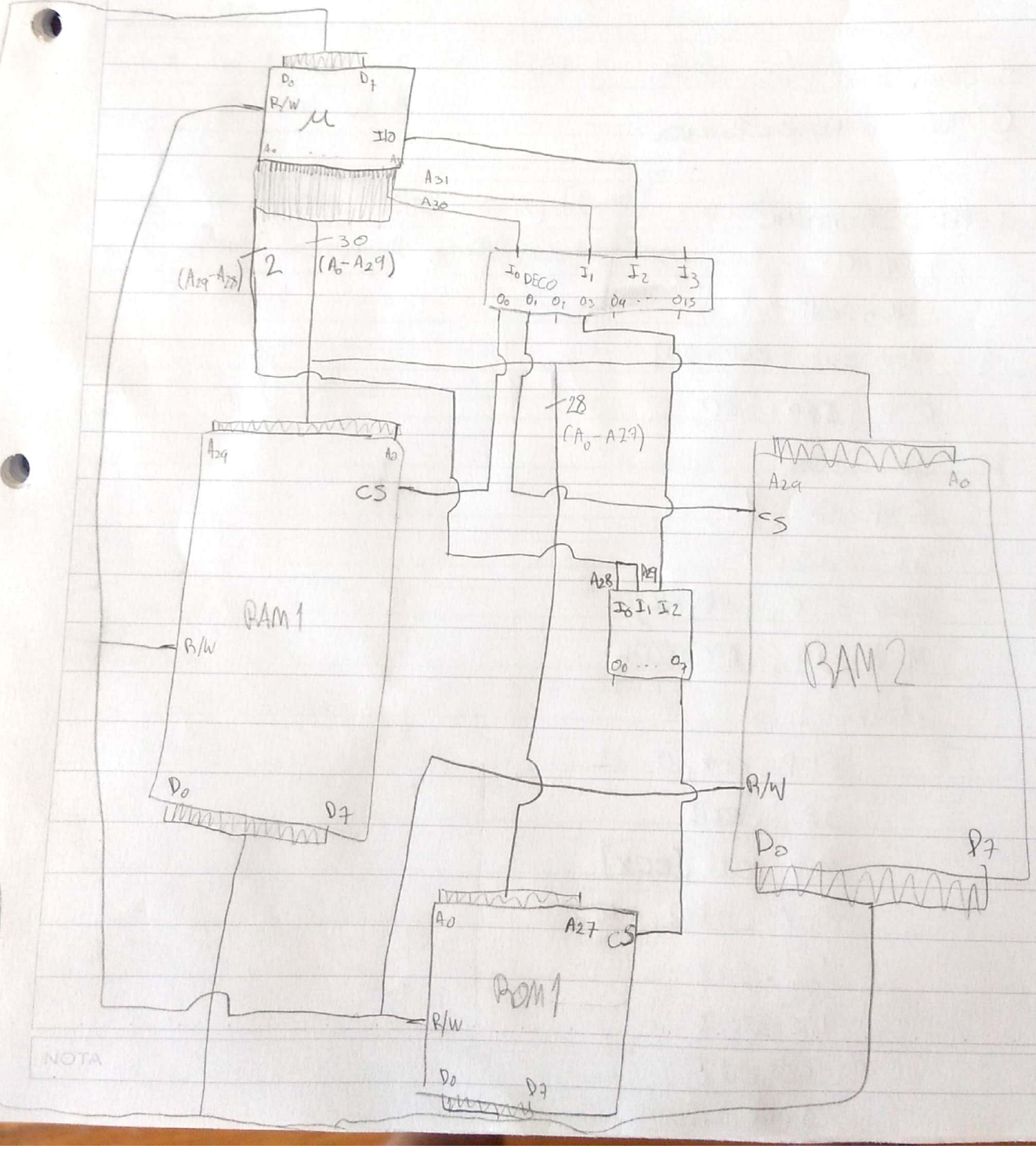
HOJA Nº
FECHA

1)

00000000 } RAM1
3FFFFFFF
40 } RAM2
7F ...

FOO... } ROM
FF ...

II) Un SMP al código del programa en la dirección FFFFFF00h



NOTA

2)

1) Sea m la dirección del stack asignada a la función

m	
m+4	EBP del contexto anterior
m+8	EIP de retorno
m+12	Puntero a string encabezado
m+16	tam_enc
m+20	Puntero a pie
m+24	tam_pie

2)

```
section .text
GLOBAL imprime_Pantalla

imprime_Pantalla:
    enter
    Push edi
    MOV eax, [ebp + 20]
    cmp eax, 80
    jg .error
    Cmp eax, 1
    jl .error
    Mov ecx, [ebp+16]
    Mov edx, A0000000h
    .loop:
        Cmp eax, 0
        je .exit
        Mov edi, [ecx]
        MoV [edx], edi
        dec eax
        inc ecx
        inc edx
        Jmp .loop
```

HOJA Nº

FECHA

- error:

```
Mov eax, 1
Pop edi
Leave
ret
```

- exit:

```
Mov eax, 0
Pop edi
leave
ret
```

3] ○ la línea Buff resb 1 debe estar en la sección .bss

○ la línea Mov ecx, [Buff] no lleva corchetes, ya que la syscall recibe un puntero