

DA RESTITUIRE INSIEME AGLI ELABORATI e A TUTTI I FOGLI
 → NON USARE FOGLI NON TIMBRATI
 → ANDARE IN BAGNO PRIMA DELL'INIZIO DELLA PROVA
 → NO FOGLI PERSONALI, NO TELEFONI, SMARTPHONE/WATCH, ETC

NOTA: per l'esercizio 4 dovranno essere consegnati DUE files: il file del programma VERILOG e il file relativo all'output (screenshot o copy/paste)

1) [12/30] Trovare il codice assembly MIPS corrispondente al seguente programma (usando solo e unicamente istruzioni della tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS riportate qua sotto per riferimento).

```
union header {
    struct {
        union header *ptr;
        unsigned size;
    } s;
};
typedef union header Header;

static Header base;
static Header *r;
r *freep = NULL;

void *malloc(unsigned nbytes)
{
    Header *p, *prevp;
    unsigned nunits;
    nunits=(nbytes+sizeof(Header)-1)/sizeof(Header)+1;

    if((prevp = freep) == NULL) {
        base.s.ptr = freep = prevp = &base;
        base.s.size = 0;
    }
    for(p = prevp->s.ptr; ; prevp = p, p = p->s.ptr) {
        if(p->s.size >= nunits) {
            if(p->s.size == nunits)
                prevp->s.ptr = p->s.ptr;
            else {
                p->s.size -= nunits;
                p += p->s.size;
                p->s.size = nunits;
            }
            freep = prevp;
            return (void *) (p + 1);
        }
        if(p == freep)
            if (p = sbrk(nunits)) == NULL)
                return NULL;
    }
}
```

Instructions

Opcode+Funcnt (hexadecimal)	Instruction	Example	Meaning	Comments
00+20/00+21	add	add/addu \$1,\$2,\$3	\$1 = \$2 + \$3	(signed/unsigned) 3 operands; exception possible
00+22/00+23	subtract	sub/subu \$1,\$2,\$3	\$1 = \$2 - \$3	(signed/unsigned) 3 operands; exception possible
08/09	add immediate	addi/addiu \$1,\$2,100	\$1 = \$2 + 100	(signed/unsigned) 4 operands; exception possible
00+18/00+19	multiplication	mult/multu \$1, \$2	Hi,Lo = \$1 x \$2	(signed/unsigned) 64-bit Product; result in Hi,Lo
00+1A/00+1B	division	div/divu \$1, \$2	Hi = \$1 % \$2, Lo = \$1 / \$2	(signed/unsigned) division
00+10/00+12	move from Hi / move from Lo	mfi/mflo \$1	\$1 = Hi (\$1 = Lo)	Create copy of Hi (Create a copy of Lo)
00+2A/00+2B	set on less than	slt/sltu \$1,\$2,\$3	if (\$2 < \$3) \$1 = 1; else \$1 = 0	(signed/unsigned) compare \$2 and \$3 (less than)
0A/0B	set on less than immediate	slti/sltiu \$1,\$2,100	if (\$2 < 100) \$1 = 1; else \$1 = 0	(signed/unsigned) compare \$2 and constant (less than)
00+24/25/26/27	and / or / xor / nor	and/or/xor/nor \$1,\$2,\$3	\$1 = \$2 & \$3 / \$2 \$3 / \$2 ^ \$3 / ~(\$2 \$3)	3 register operands; Logical AND/OR/XOR/NOR
0C/0D/0E	andi / ori / xori	andi/ori/xori \$1,\$2,100	\$1 = \$2 & 100 / \$2 100 / \$2 ^ 100	Logical AND/OR/XOR register, constant
00+00	shift left logical	sll \$1,\$2,10	\$1 = \$2 << 10	Shift left by constant
00+02/00+03	shift right (l=logical,a=arithmetic)	srl/sra \$1,\$2,10	\$1 = \$2 >> 10	Shift right by constant (for arithmetic: sign is preserved)
00+04	shift left logical	sllv \$1,\$2,10	\$1 = \$2 << \$3	Shift left by variable
00+06/00+07	shift right (l=logical,a=arithmetic)	srlv/srav \$1,\$2,10	\$1 = \$2 >> \$3	Shift right by variable (for arithmetic: sign is preserved)
23/20	load word / load byte	lw/lb \$1,100(\$2)	\$1 = Memory[\$2+100]	Data from memory to register
24	load byte unsigned	lbu \$1,100(\$2)	\$1 = Memory[\$2+100]	Data from mem. To reg.; no sign extension
2B/28	store word / store byte	sw/sb \$1,100(\$2)	Memory[\$2+100] = \$1	Data from register to memory
0F	load upper immediate	lui \$1,0x1234	\$1 = 0x12340000	load most significant 16 bits
PSEUDOINSTRUCTION	load address	la \$1,var	\$1 = &var	Load address of var (lui \$1,H16(&var);ori \$1,L16(&var) H16/L16=high/low 16 bits of &var
02	jump	j 10000	go to 10000	Jump to target address
00+08	jump register	jr \$31	go to \$31	For switch, procedure return
03	jump and link	jral 10000	\$31 = PC + 4; go to 10000	For procedure call
04	branch on equal	beq \$1,\$2,100	if (\$1 == \$2) go to PC+4+100	Equal test; PC relative branch
05	branch on not equal	bne \$1,\$2,100	if (\$1 != \$2) go to PC+4+100	Not equal test; PC relative
00+0C	syscall	syscall	call OS service Sv0	See table of system calls below
10+10, rs=10	rfe	rfe	shift right (k,e) bits in STATUS reg	Exit Kernel Mode, Enable Interrupts
PSEUDOINSTRUCTION	branch unconditional	b 100	go to PC+4+100	PC relative branch (e.g., beq \$0,\$0,100)
PSEUDOINSTRUCTION	no operation	nop	do nothing	Do nothing (e.g. sll \$0,\$0,0)
30	load-linked	ll \$1,100(\$2)	\$1=Memory[\$2+100]	Read and start to monitor the given memory location
38	store-conditional	sc \$1,100(\$2)	Memory[\$2+100]=\$1 or →	return 0 if a coherence action happens since the previous ll (\$1 must be different from 0)
11+00 fmt=10/11	adds / add.d	add.s \$f0,\$f2,\$f4	\$f0=\$f2+\$f4	Single and double precision add
11+01 fmt=10/11	subs / sub.d	sub.s \$f0,\$f2,\$f4	\$f0=\$f2-\$f4	Single and double precision subtraction
11+02 fmt=10/11	mul.s / mul.d	mul.s \$f0,\$f2,\$f4	\$f0=\$f2*\$f4	Single and double precision multiplication
11+03 fmt=10/11	div.s / div.d	div.s \$f0,\$f2,\$f4	\$f0=\$f2/\$f4	Single and double precision division
11+05 fmt=10/11	abs.s / abs.d	abs.s \$f0,\$f2	\$f0=ABS(\$f2)	Single and double precision absolute value
11+06 fmt=10/11	mov.s / mov.d	mov.s \$f0,\$f2	\$f0←\$f2	Single and double precision move
11+07 fmt=10/11	neg.s / neg.d	neg.s \$f0,\$f2	\$f0 = - (\$f2)	Single and double precision opposite value
11+3C (31, 32, 3D, 3E, 3F) fmt=10/11	c.lt.s / c.lt.d (ne,eq,gt,le,ge)	c.lt.s \$f0,\$f2	Temp=(\$f0 < \$f2)	Single and double: compare \$f0 and \$f2 <,>,<=,>=
11+00 fmt=4/0	move to/from coprocessor 1	mtc1/mfc1 \$1,\$f2	\$f2=\$1 / \$1=\$f2	Move \$1 to/from C1 reg. \$f2 (no conversion)
10+00 fmt=4/0	move to/from coprocessor 0	mtc0/mfc0 \$1,\$f2	\$f2=\$1 / \$1=\$f2	Move \$1 to/from C0 reg. \$f2 (no conversion)
11+00 fmt=6/2	move to/from control reg of cop.1	ctc1/cfc1 \$1,\$cfc2	\$cfc2=\$1 / \$1=\$cfc2	Move \$1 to/from C1-CONTROL register
11 fmt=8, ft=1/0	branch on true/false	bclt/bcfl label	if (Temp = true/false) go to label	Temp is 'Condition-Code'
31/39	load/store floating point (32bit)	lwc1/swc1 \$f0,0(\$1)	\$f0←Memory[\$1] / Memory[\$1]←\$f0	Data from FP (C1) register to memory
11+21, fmt=10/11+22, fmt=11	convert from/to single to/from double	cvt.d.s/cvt.s.d \$f0,\$f2	\$f0=(double)\$f2/\$f0=(single)\$f2	Type conversion
11+24, fmt=11/11+20	convert from/to single to/from integer	cvt.w.s/cvt.s.w \$f1,\$f0	\$f1=(int)\$f0 / \$f0=(single)\$f1	Type conversion

Register Usage

Name	Reg. Num.	Usage
\$zero	0	The constant value 0
\$s0-\$s7	16-23	Saved
\$t0-\$t9	8-15,24-25	Temporaires
\$a0-\$a3	4-7	Arguments

Name	Reg.Num.	Usage
\$v0-\$v1	2-3	Results
\$fp, \$sp	30,29	frame pointer, stack pointer
\$ra, \$gp	31,28	return address, global pointer
\$k0-\$k1	26,27	Kernel usage

Reg. Num.	Usage
\$f0, \$f2	Return values
\$f12,\$f14	Function arguments
\$f20,\$f22,\$f24,\$f26,\$f28,\$f30	Saved registers
\$f4,\$f6,\$f8,\$f10,\$f16,\$f18	Temporaires registers

System calls

Service Name	Serv.No.(Sv0)	INPUT Arguments	OUTPUT Args	Service Name	Serv.No.(Sv0)	INPUT Arguments	OUTPUT Arguments
print int	1	\$a0=integer to print	---	read float	6	---	\$f0=float
print float	2	\$f12=float to print	---	read double	7	---	\$f0-f1=double
print double	3	(\$f12,\$f13)=double to print	---	read string	8	\$a0=address of input buffer, \$a1=max chars to read	---
print string	4	\$a0=address of ASCIIZ string to print	---	sbrk	9	\$a0=Number of bytes to be allocated	\$v0=pointer to allocated memory
read int	5	---	\$v0=integer	exit	10	---	---

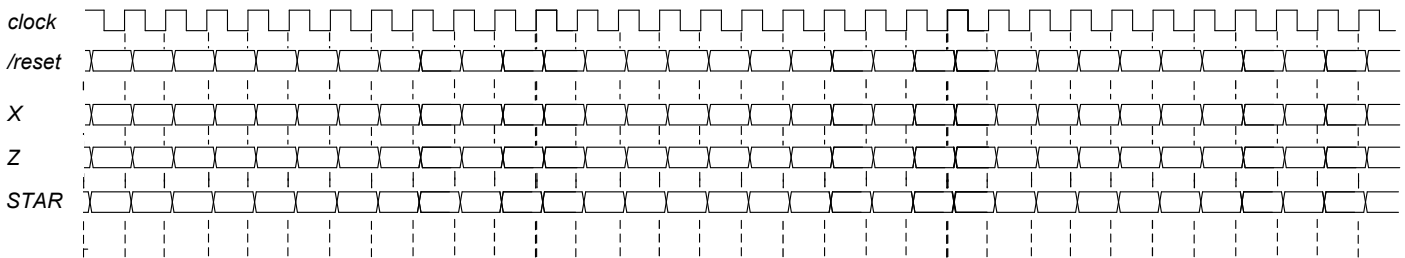
- 2) [5/30] Si consideri una cache di dimensione 768B e a 6 vie di tipo write-back. La dimensione del blocco e' 64 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 177, 1163, 223, 2181, 200, 3221, 175, 1184, 2182, 3201, 4176, 8173, 2176, 9183, 8251, 4176, 5171, 7178, 3191, 181. Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso alla cache, riportare i tag contenuti in cache al termine e la lista dei blocchi (ovvero il loro indirizzo) via via eliminati durante il rimpiazzamento ed inoltre in corrispondenza di quale riferimento il blocco e' eliminato.

- 3) [5/30]] Disegnare un possibile schema logico/architetturale del meccanismo di paginazione inversa (nota: dovranno essere esplicitati tutti i blocchi logico/architetturali usando elementi noti visti durante il corso, eccetto il blocco che genera un hash di 20 bit a partire da 52 bit) che riceve in ingresso un indirizzo virtuale a 64 bit, ha una dimensione di pagina pari a 4KiB, uno spazio di indirizzamento fisico a 32 bit.

- 4) [8/30] Descrivere e sintetizzare in Verilog una rete sequenziale basata sul modello di Moore con flip-flop D, con un ingresso X su un bit e una uscita Z su un bit che riconosca le sequenze NON interallacciate 1,0,1,0. Il modulo TopLevel e' dato con sequenza di ingresso 0,0,1,1,0,1,1,0,1,1,0,0,1,0,0,0,1,0,1,1,0,1,0,0. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità. Nota: si puo' svolgere l'esercizio su carta oppure con ausilio del simulatore salvando una copia dell'output (diagramma temporale) e del programma Verilog su USB-drive del docente.

```

Module Toplevel;
  reg reset_; initial begin reset_=0; #22 reset_=1; #300; $stop; end
  reg clock; initial clock=0; always #5 clock<=(!clock);
  reg X;
  wire z=Xxx.z;
  wire [2:0] STAR=Xxx.STAR;
  initial begin X=0;
    wait(reset_==1);
    @(posedge clock); X<=0;@(posedge clock); X<=0;@(posedge clock); X<=1;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=1;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=1;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=0;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=0;
    @(posedge clock); X<=0;@(posedge clock); X<=1;@(posedge clock); X<=0;
    @(posedge clock); X<=1;@(posedge clock); X<=1;@(posedge clock); X<=0;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=0;
    $finish;
  end
  XXX Xxx(X,Z,clock,reset_);
endmodule
    
```



SOLUZIONE

ESERCIZIO 1

```
.data
base: .space(8)
freep: word 0
.globl main
.text
malloc:
# NO CALL FRAME
S1: # a0=nbytes, sizeof(Header)=8
    addi $t0,$a0,8 # nbytes +8
    addi $t0,$t0,-1 # (nbytes +8 -1)
    srl $t0,$t0,3 # (/)8
    addi $t0,$t0,1 # (/)8+1
    # t0=nunits
#-----
S2: la $t1, freep # &freep
    lw $t2,0($t1) # VALORE_DI(freep)
    # t2 = prevp # (prevp=freep)
    bne $t2,$0,FINEIF1 # if(==NULL)
    la $t3,base # &base
    add $t2,$0,$t3 # prevp=&base
    sw $t3,0($t1) # freep=&base
    sw $t3,0($t3) # base.s.ptr=&base
    sw $0,4($t3) # base.s.size=0
FINEIF1:
#-----
S3:
#-----
S31:# t4=p
    lw $t4,0($t2) # p=prevp->s.ptr
#-----
S3.2:# VUOTO!
#-----
S3.3:
INIFOR:
S331:
    # (t0=nunits), (t4=p)
    lw $t5,4($t4) # p->s.size
    # t5 = p->s.size
    slt $t6,$t5,$t0 # A>B => !(A<B)
    bne $t6,$0,FINEIF2 # if (!(p->s.size<nunits))
```

```
CORPOIF2:
    bne $t5,$t0,ELSE3 # if (p->s.size==nunits)
    lw $t7,0($t4) # (p->s.ptr)
    sw $t7,0($t2) # prevp->s.ptr=(.)
    j FINEIF3
ELSE3:
    sub $t5,$t5,$t0 # (p->s.size-nunits)
    sw $t5,4($t4) # p->size=(.)
    add $t4,$t4,$t5 # p+=p->s.size
    sw $t0,4($t4) # p->s.size=nunits
FINEIF3:
    sw $t2,0($t1) # freep=prevp
    addi $v0,$t4,8 # p+1, il "+1" e' un Header
    # v0 e' il valore di ritorno di "malloc"
    j FINEFUN
FINEIF2: #chiudo S3.3.1
#-----
S3.3.2:
    lw $t3,0($t1) # freep aggiornato
    bne $t4,$t3,FINEIF4 # if (p==freep)
    addi $v0,$0,9 # servizio 9 => SBRK
    add $a0,$t0,$0 # a0=nunits da allocare
    # NOTA: IL SEGUENTE CODICE E' ADATTATO
    # AL FINE DI AVERE UNA VERISONE FUNZIONANTE
    sll $a0,$a0,3 # sbrk(nunits * sizeof(Header))
    syscall #sbrk # v0 = pun. memoria allocata
    add $t4,$0,$v0 # p=sbrk(.)
    add $t8,$0,$0
    addi $t8,$0,-1 # t8 = -1
    beq $t4,$t8,FINENULL # if(== -1) ADATTATO!
    sw $t0,4($v0) # up->s.size=nunits
    sw $t4,0($t4) # up->s.ptr=p
    sw $v0,0($t4) # p->s.ptr = p
    add $t4,$t4,$0 # p = freep
    j FINEIF5
FINENULL:
    add $v0,$0,$0 # v0 <= 0
    j FINEFUN
FINEIF5:
FINEIF4: # chiude S332 e CORPOFOR
    add $t2,$0,$t4 # prevp=p
    lw $t4,0($t4) # p=p->s.ptr
    j INIFOR # chiudo FOR, S3.3.2 e S3.3
#-----
FINEFUN: jr $ra
```

<pre>#TESTBENCH printHEAptop: add \$a0,\$0,\$0 # get HEAptop addi \$v0,\$0,9 # serv.9 syscall #sbrk add \$v1,\$0,\$v0 # salvo in v1 serv.4 la \$a0,HEAptop # stampa msg addi \$v0,\$0,4 # syscall #print_str add \$a0,\$0,\$v1 # nuovo HEAptop addi \$v0,\$0,1 # serv.1 syscall #print_int la \$a0,RTN # stampa RTN addi \$v0,\$0,4 # serv.4 syscall # print_str jr \$ra</pre>	<pre>printMALLOC: #a0 = pun. da stampare add \$t0,\$a0,\$0 # t0=pun. la \$a0,heapmsg addi \$v0,\$0,4 # serv.4 syscall #print_str add \$a0,\$t0,\$0 # a0=pun addi \$v0,\$0,1 # serv.1 syscall #print_int la \$a0,RTN # stampa RTN addi \$v0,\$0,4 # serv.4 syscall # print_str jr \$ra #-----</pre>	<pre>main: jal printHEAptop addi \$a0,\$0,256 # alloca 256B jal malloc add \$a0,\$v0,\$0 jal printMALLOC jal printHEAptop addi \$a0,\$0,256 # alloca 256B jal malloc add \$a0,\$v0,\$0 jal printMALLOC jal printHEAptop addi \$a0,\$0,256 # alloca 256B jal malloc add \$a0,\$v0,\$0 jal printMALLOC jal printHEAptop addi \$v0,\$0,10 #serv.10 syscall #exit</pre>	<p>Console</p> <pre>Heap Top:268697600 Malloc returns:268697608 Heap Top:268697864 Malloc returns:268697872 Heap Top:268698128 Malloc returns:268698136 Heap Top:268698392</pre>
--	--	---	--

ESERCIZIO 2

Si ricava S=C/B/A=# di set della cache=768/64/6, XM=X/B, XS=XM%S, XT=XM/S.
 A=6, B=64, C=768, RP=LRU, Thit=4, Tpen=40, 20 references:

=== T	X	XM	XT	XS	XB	H	[SET]:USAGE	[SET]:MODIF	[SET]:TAG
=== R	177	2	1	0	49	0	[0]:5,0,0,0,0,0	[0]:0,0,0,0,0,0	[0]:1,-,-,-,-,-
=== W	1163	18	9	0	11	0	[0]:4,5,0,0,0,0	[0]:0,0,0,0,0,0	[0]:1,9,-,-,-,-
=== R	223	3	1	1	31	0	[1]:5,0,0,0,0,0	[1]:0,0,0,0,0,0	[1]:1,-,-,-,-,-
=== W	2181	34	17	0	5	0	[0]:3,4,5,0,0,0	[0]:0,0,0,0,0,0	[0]:1,9,17,-,-,-
=== R	200	3	1	1	8	1	[1]:5,0,0,0,0,0	[1]:0,0,0,0,0,0	[1]:1,-,-,-,-,-
=== W	3221	50	25	0	21	0	[0]:2,3,4,5,0,0	[0]:0,0,0,0,0,0	[0]:1,9,17,25,-,-
=== R	175	2	1	0	47	1	[0]:5,2,3,4,0,0	[0]:0,0,0,0,0,0	[0]:1,9,17,25,-,-
=== W	1184	18	9	0	32	1	[0]:4,5,2,3,0,0	[0]:0,1,0,0,0,0	[0]:1,9,17,25,-,-
=== R	2182	34	17	0	6	1	[0]:3,4,5,2,0,0	[0]:0,1,0,0,0,0	[0]:1,9,17,25,-,-
=== W	3201	50	25	0	1	1	[0]:2,3,4,5,0,0	[0]:0,1,0,1,0,0	[0]:1,9,17,25,-,-
=== R	4176	65	32	1	16	0	[1]:4,5,0,0,0,0	[1]:0,0,0,0,0,0	[1]:1,32,-,-,-,-
=== W	8173	127	63	1	45	0	[1]:3,4,5,0,0,0	[1]:0,0,0,0,0,0	[1]:1,32,63,-,-,-
=== R	2176	34	17	0	0	1	[0]:2,0,0,0,0,0	[0]:0,1,0,1,0,0	[0]:1,9,17,25,-,-
=== W	9183	143	71	1	31	0	[1]:2,3,4,5,0,0	[1]:0,0,0,0,0,0	[1]:1,32,63,71,-,-
=== R	8251	128	64	0	59	0	[0]:1,2,4,3,5,0	[0]:0,1,0,1,0,0	[0]:1,9,17,25,64,-
=== W	4176	65	32	1	16	1	[1]:2,5,3,4,0,0	[1]:0,1,0,0,0,0	[1]:1,32,63,71,-,-
=== R	5171	80	40	0	51	0	[0]:0,1,3,2,4,5	[0]:0,1,0,1,0,0	[0]:1,9,17,25,64,40
=== W	7178	112	56	0	10	0	[0]:5,0,2,1,3,4	[0]:0,1,0,1,0,0	[0]:56,9,17,25,64,40
=== R	3191	49	24	1	55	0	[1]:1,4,2,3,5,0	[1]:0,1,0,0,0,0	[1]:1,32,63,71,24,-
=== W	181	2	1	0	53	0	[0]:4,5,1,0,2,3	[0]:0,0,0,1,0,0	[0]:56,1,17,25,64,40

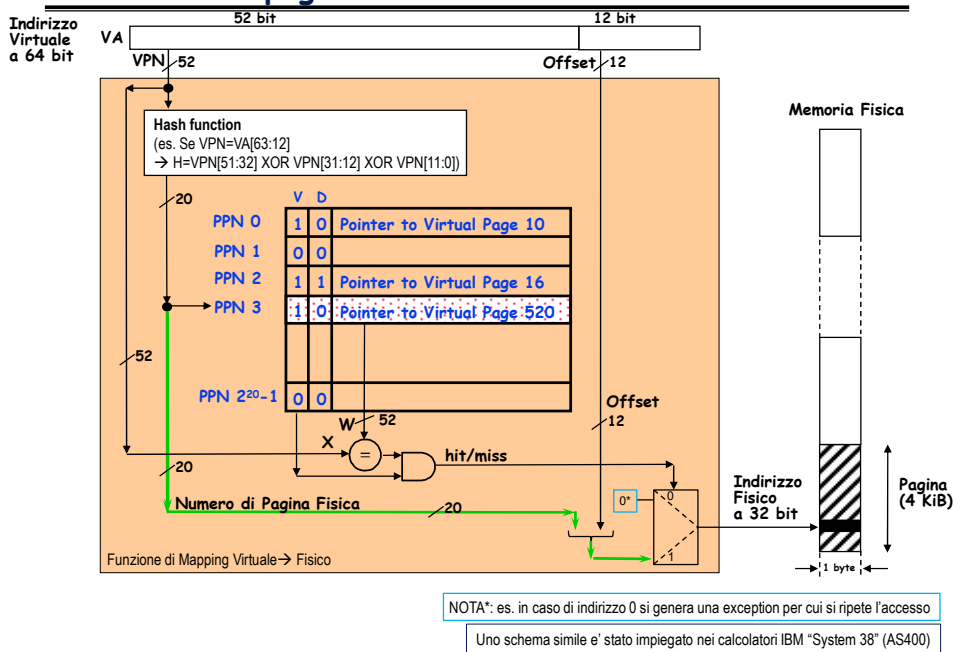
CONTENUTI dei 2 SET al termine:

LISTA BLOCCHI USCENTI:

(out: XM=2 XT=1 XS=0)
 (out: XM=18 XT=9 XS=0)

ESERCIZIO 3

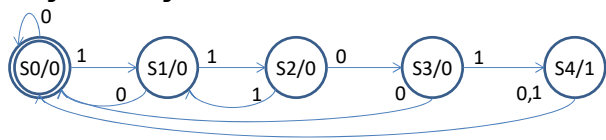
Tabella delle pagine inverse - schema architetturale



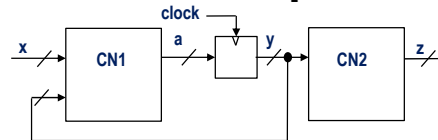
ESERCIZIO 4

E' utile rappresentare il diagramma a stati del riconoscitore di sequenza richiesto:

Diagramma degli Stati



Schema di riferimento per Moore



Codice VERILOG:

```

module XXX(x,z,clock,reset_);
input clock,reset_,x;
output z;
reg [2:0] STAR;
reg OTR;
parameter S0='B000, S1='B001, S2='B010, S3='B011, S4='B100;
always @(reset_==0) begin STAR<=0; end
assign z=(STAR==S4) ? 1:0;
always @(posedge clock) if (reset_==1)
    casex (STAR)
        S0: begin STAR<=(x==0)?S0:S1; end
        S1: begin STAR<=(x==0)?S0:S2; end
        S2: begin STAR<=(x==0)?S3:S1; end
        S3: begin STAR<=(x==0)?S0:S4; end
        S4: begin STAR<=S0; end
    endcase
endmodule
    
```

Diagramma temporale:

