PRACTICA EC 2ª CONVOCATORIA

Autor: Carlos Martínez

Professor: Carles Aliagas

CONTINGUT

FASE 1: SUMADOR DE 32 BITS 4
TASCA 1.1 4
Disseny del circuit: 4
Mòdul CLA 16 bits 4
Mòdul CLA 4 bits 4
LCU 5
Mòdul CSA 16 bits 5
Temps de retard 5
Àrea estimada 6
Joc de proves 7
TASCA 1.2 8
Temps de retard i área 9
Joc de proves 9
TASCA 1.3 10
Temps de retard 11
Joc de proves 11
FASE 2: UNIDAD DE PROCESO 12
FASE 3: UNIDAD DE CONTROL 13
FASE 4: INSTRUCCIONES ADICIONALES 14
TAREA 4.1: PUSH 14
TAREA 4.2: POP 15
Joc de proves 16
TAREA 4.2: CMPSETI 17
Joc de proves 18
FASE 5: PRUEBA DE FUNCIONAMIENTO 19
TAREA 5.1 19
TAREA 5.2 20
Joc de proves 21
TAREA 5.3 22
FASE 6: ANÁLISIS DE MEMORIA 22
Mobile processor-Core i7-7600U – Intel 23
L1I Cache (Cache L1 d'instruccions): 23
L1D Cache (Cache L1 de datos): 23

L2 Cache: 23
Anàlisi TASCA 6-2 23
Anàlisi TASCA 6-3 25
Anàlisi TASCA 6-3 25
Anàlisi TASCA 6-4 26
Desktop processor-Ryzen 7 3700U – AMD 27
L1I Cache (Cache L1 d'instruccions): 28
L1D Cache (Cache L1 de datos): 28
L2 Cache: 28
Anàlisi TASCA 6-2 28
Anàlisi TASCA 6-3 30
Anàlisi TASCA 6-4 31
Console processor-Ryzen 7 Zen 2 – AMD 32
L1I Cache (Cache L1 d'instruccions): 32
L1D Cache (Cache L1 de datos): 32
L2 Cache: 32
Anàlisi TASCA 6-2 33
Anàlisi TASCA 6-3 35
Anàlisi TASCA 6-4 35
TASCA 6-5 36
CIRCUITS FASES 2-3-4 36

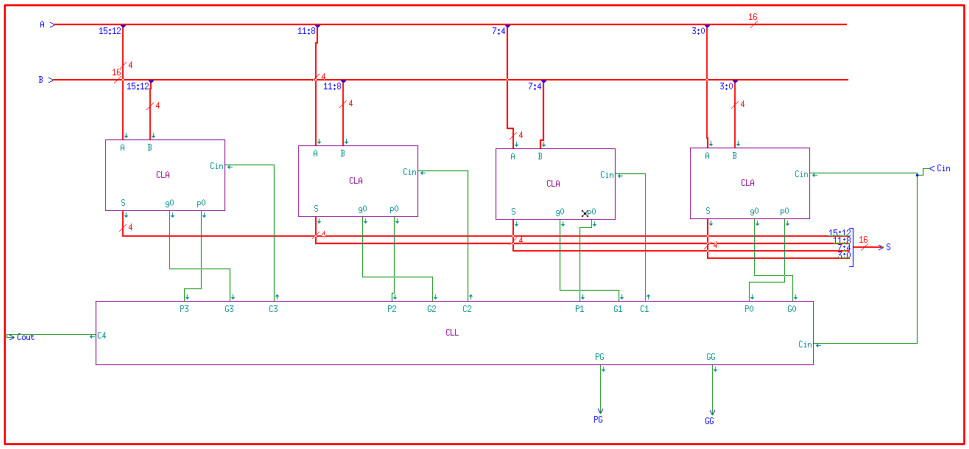
FASE 1: SUMADOR DE 32 BITS

TASCA 1.1

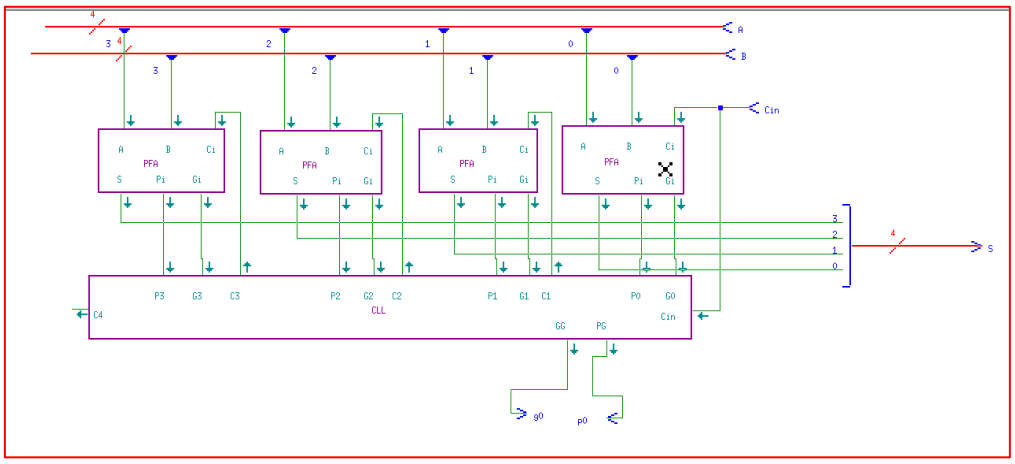
Disseny del circuit:

Per a dissenyar el circuit m'he basat en la implementació feta a teoria i he realitzat un mòdul de CLA de 4 bits que conté 4 PFA a cada mòdul. A més, aquest circuit inclou una LCU, que calcularà la generació i propagació del circuit en cas d'haver.

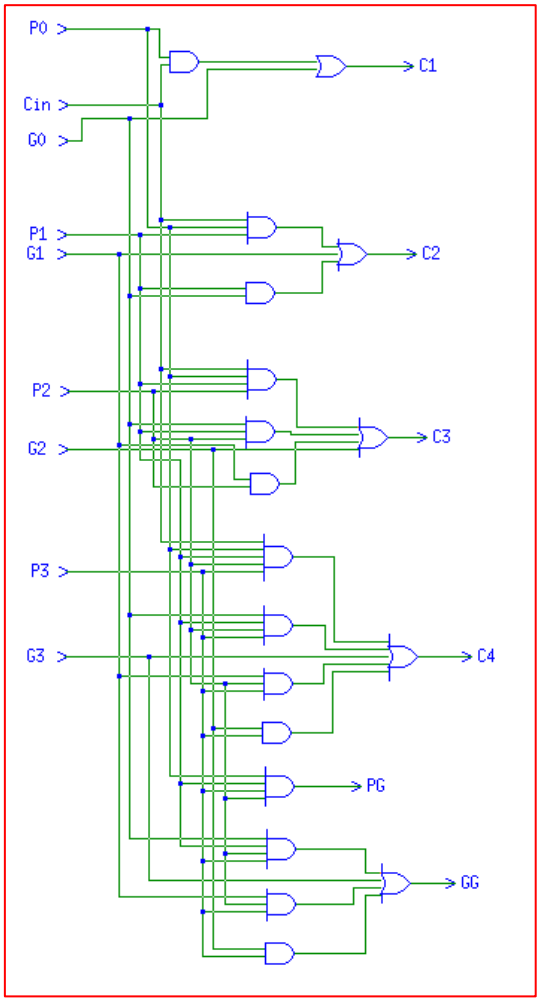
Mòdul CLA 16 bits



Mòdul CLA 4 bits



LCU



Mòdul CSA 16 bits

Temps de retard

Primer CLA:

p_0, p_1, p_2, p_3 = **3T** i g_0, g_1, g_2, g_3= **3T**

PG (p0 de CLA16bits) = 6T i GG (g0 de CLA16bits) = **9T**

C_1, C_2, C_3 = **9T**

S0= **8T** pel PFA y S1,S2, S3 = **13T**

CLA_1, CLA_2 I CLA_3

p_0, p_1, p_2, p_3 = **3T** i g_0, g_1, g_2, g_3= **3T**

PG CLA_1: **6T** GG CLA_1: 9T

PG CLA_2: **6T** GG CLA_2: **9T**

PG CLA_3: **6T** GG CLA_3: **9T**

LCU

$p_0 \ldots p_{12}$= **6T**

$g_0 \ldots g_{12}$= **9T**

PG= **9T** GG=**15T**

C_4= **12T** $C_8 \ldots C_{16}$=**15T**

Càlcul CLA_1

p_4= **6T** g_4=**9T**

PG CLA_1= **6T** GG CLA_1=**9T**

C_5, C_6, C_7= **18T**

S_4=**16T** S_5, S_6, S_7=**22T**

Càlcul CLA_2, CLA_3

p_8 i p_{12}= **6T** g_8 i g_{12}=**9T**

PG CLA_2= **6T** GG CLA_2=**9T**

PG CLA_3=**6T** GG CLA_3=**9T**

$C_{9,\ 10,11,12,13,14,15}$= **21T**

S_8 i S_{12}= **22T** $S_{9,\ 10,\ 11,\ 12,\ 13,\ 14,\ 15}$= **25T**

Camí crític:

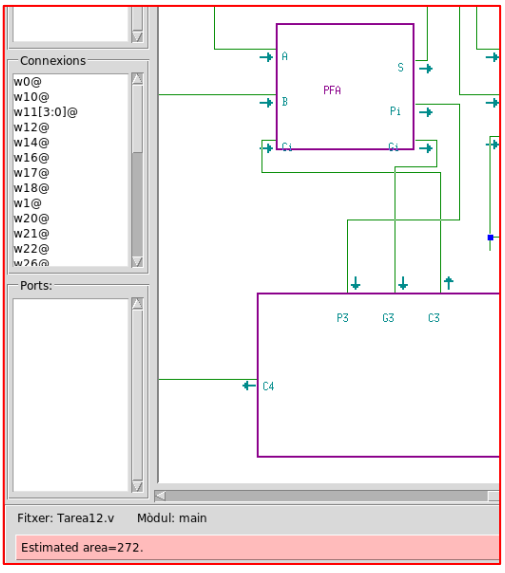
25T

Àrea estimada

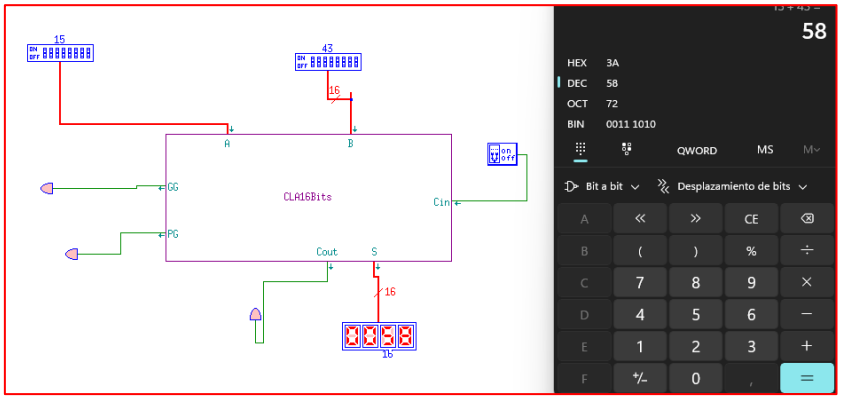
Àrea CLA=272

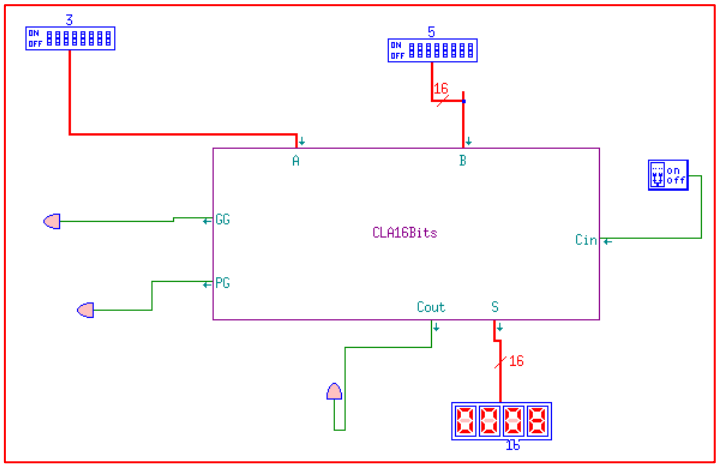
Àrea LCU= 160

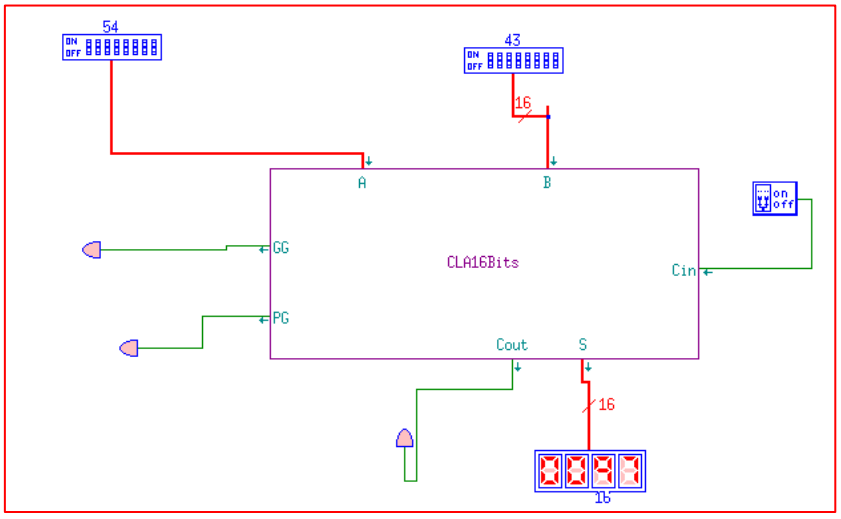
Àrea total CLA 16 bits= (160+272*4)=1248



Joc de proves

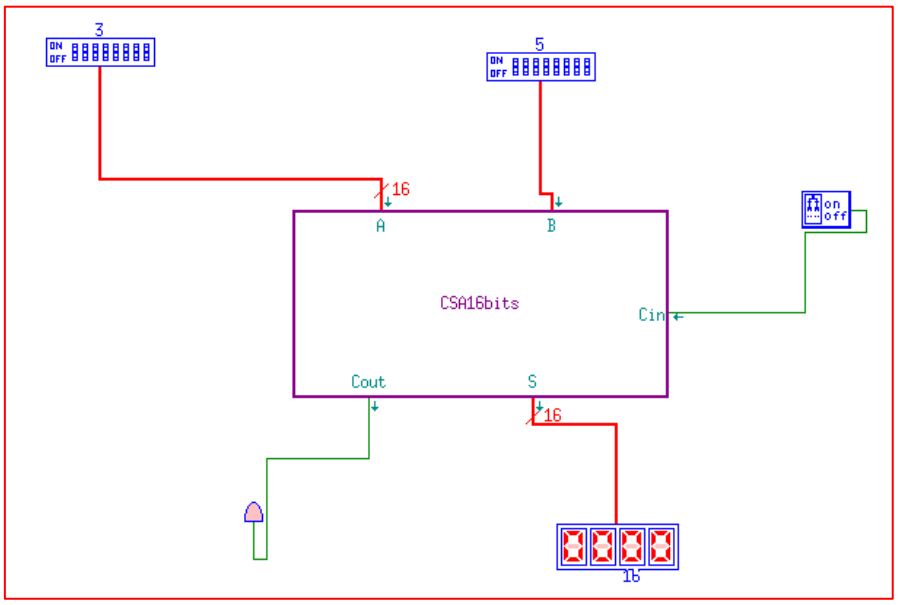




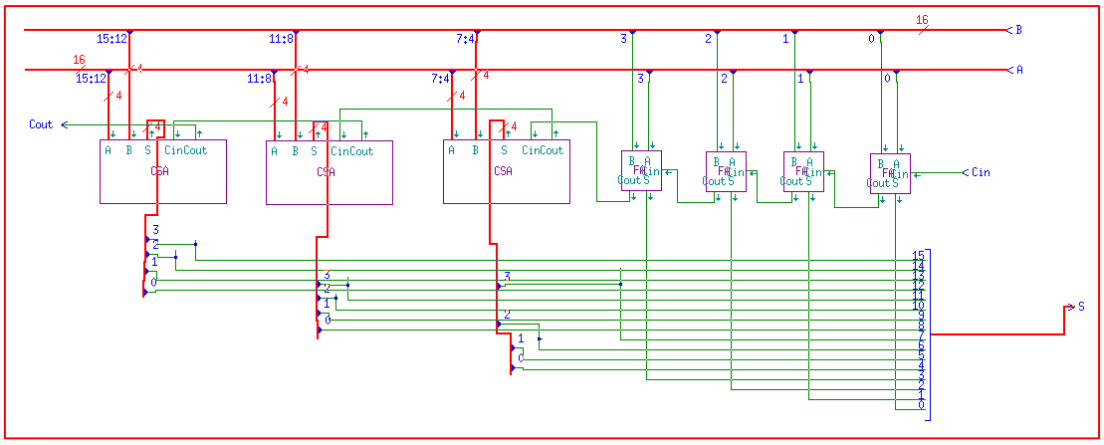


TASCA 1.2

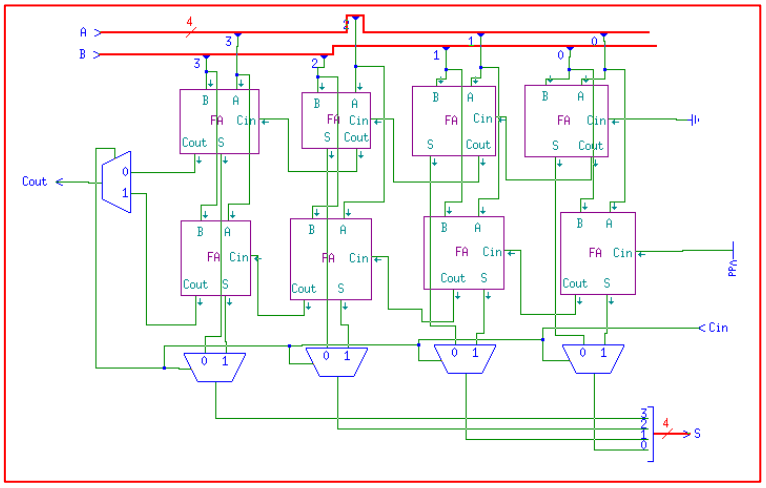
Main



CSA 16 bits



CSA 4 bits

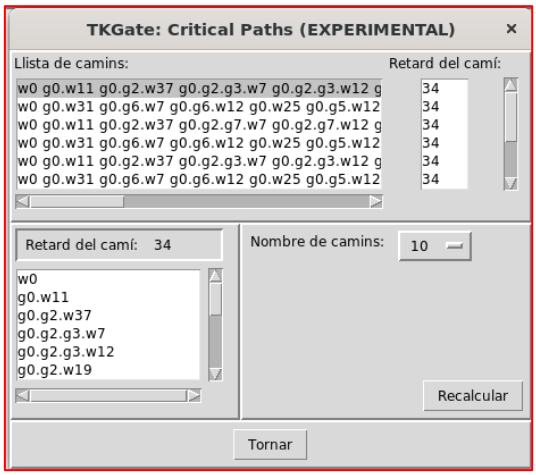


Temps de retard i área

RetardC= 10T+(4-1)*(10T-4T)=28T

RetardS=10T +(4-2)*(10T-4T)+(8T-4T)=26T

Retard CSA=28T+(4-1)*2=28T+6T=**34T**

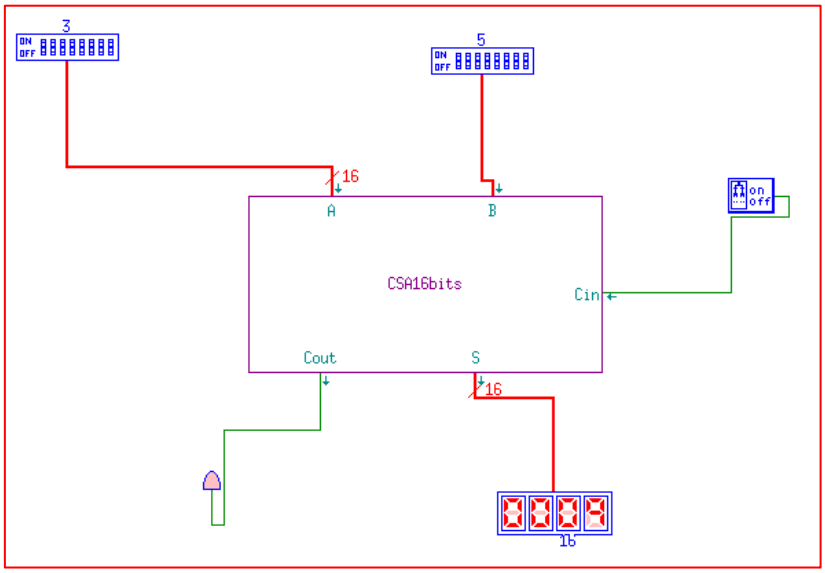


Àrea CPA=4*(2 XOR(8), 2 AND(6), OR(6)= 4*(16+12+6)=136

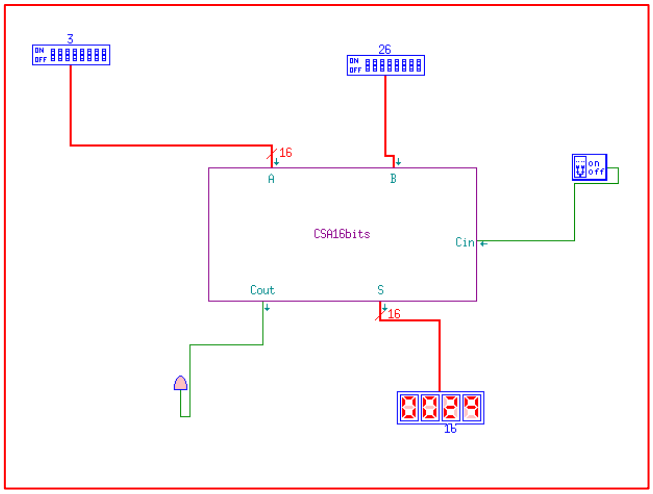
Àrea total= 3*área CSA+àrea CPA=3*312+136 =**1072**

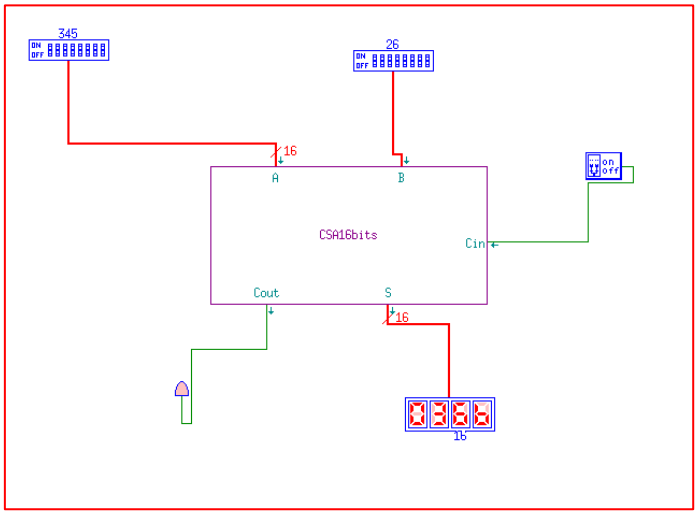
Joc de proves

Carry on



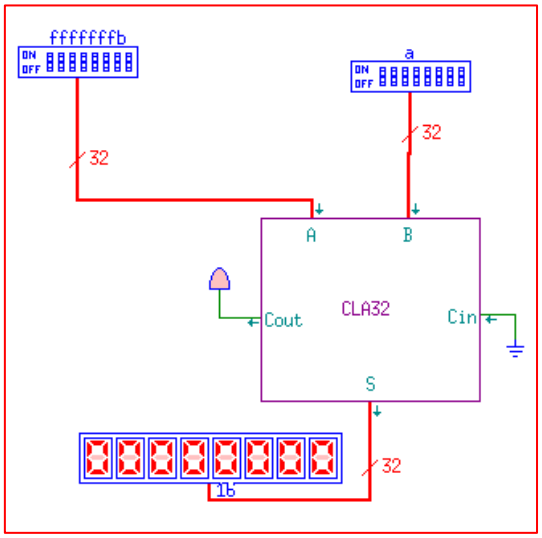
Carry off



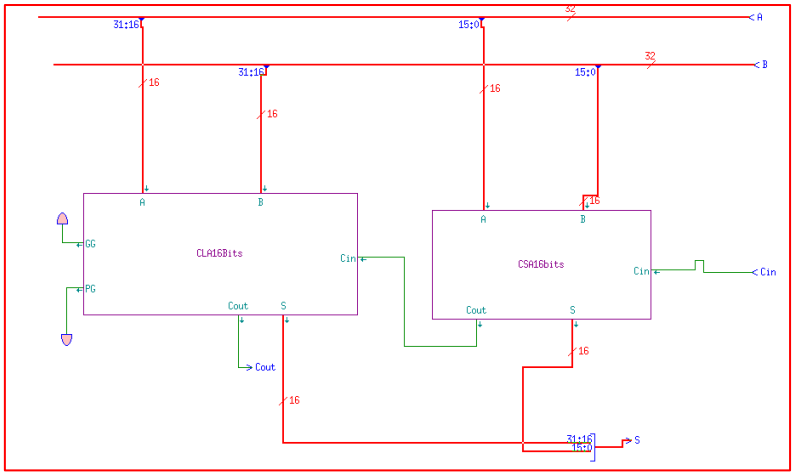


TASCA 1.3

Main



CPA 32 bits

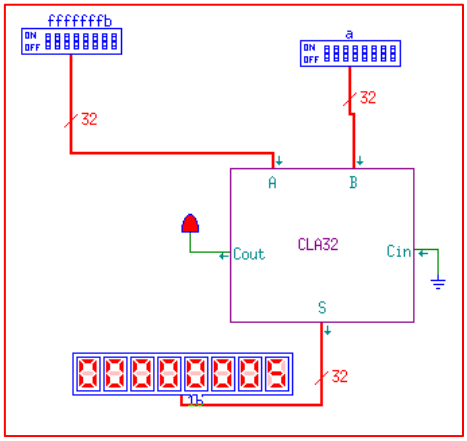


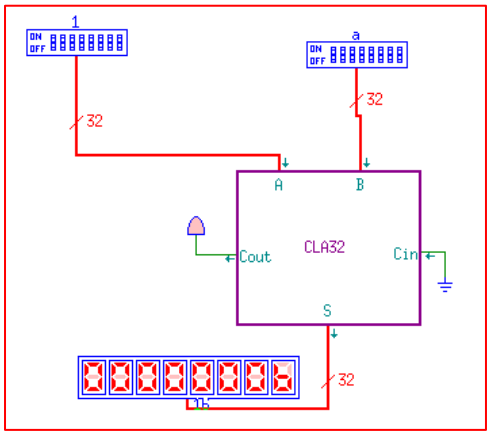
Temps de retard

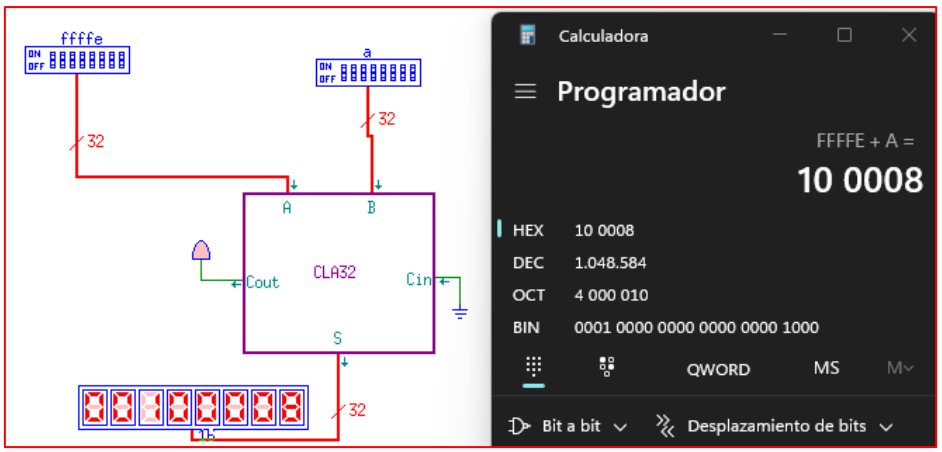
RetardC= 10T+(31-1)*(10T-4T)=196T

RetardS=10T+(32-2)*(10T-4T)+(8T-4T)=194T

Joc de proves







FASE 2: UNIDAD DE PROCESO

Captures del circuit adjntades al final de la documentació´.

Joc de proves, provat amb les proves de les instruccions addicionals afegides.

FASE 3: UNIDAD DE CONTROL

Fitxers utilitzats per a la unitat de control:

```
 1 02/ 1000    #jump
 2 04/ 2190    #beq
 3 20/ 40a1    #add
 4 22/ 41a1    #sub
 5 23/ c83     #lw
 6 24/ 4021    #and
 7 25/ 4061    #or
 8 2A/ 41e1    #slt
 9 2B/ 8a      #sw
10 2C/ 41ad    #push
11 2D/ 8191    #cmpseti
12 2F/ 4C87    #pop
```

Script per a no haver de carregar els fitxers a memoria manualment:

```
 1 #Load
 2 load Fetch.g13 "mult.mem"
 3 load Mem.g0 "mult.mem"
 4 load ctrl.g0 "uc.mem"
 5 #load ALUCtrl.g0 "ALUControl.mem"
 6
 7 #Reset
 8 set reset 1'b1
 9 clock +1
10 set reset 1'b0
```

Captures adjuntades al final de la documentació.

Joc de proves, provat amb les proves de les instruccions addicionals afegides.

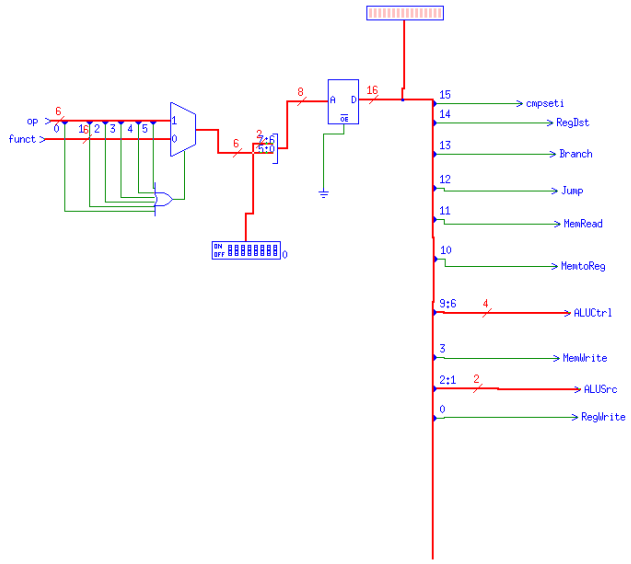
FASE 4: INSTRUCCIONES ADICIONALES

TAREA 4.1: PUSH

Per a poder incloure aquesta instrucció, he hagut de modificar la unitat de control i fer que ALUSrc tingui dos bits en lloc d'un.

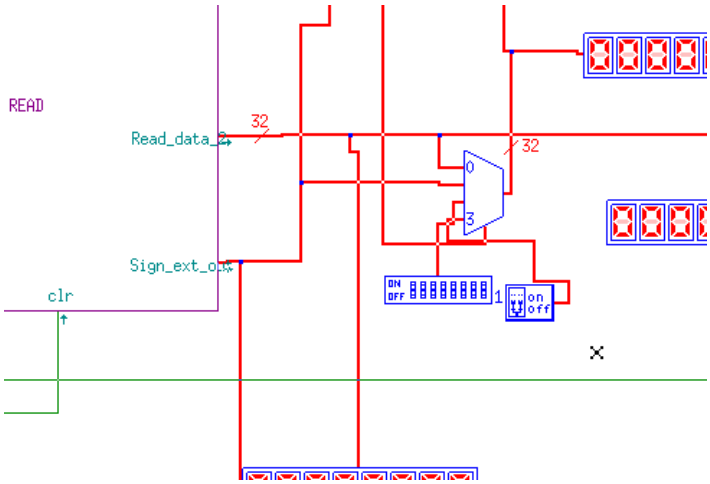
A més, he modificat el multiplexor d'abans de l'ALU per a poder incloure el valor necessari per a fer la resta del push ($29-4).

UC modificada:




```
 1 02/ 1000    #jump
 2 04/ 2190    #beq
 3 20/ 40a1    #add
 4 22/ 41a1    #sub
 5 23/ c83     #lw
 6 24/ 4021    #and
 7 25/ 4061    #or
 8 2A/ 41e1    #slt
 9 2B/ 8a      #sw
10 2C/ 41ad    #push
```

Multiplexor modificat:



Codificació de la instrucció:

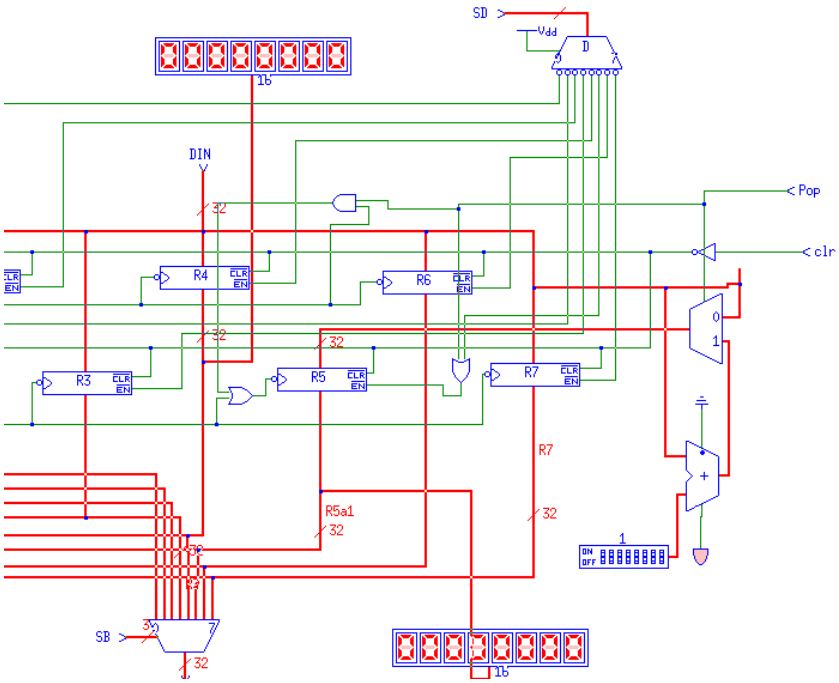
Push $a3

PUSH	101100	11101	00111	11101	00000000000	B3A7E800
	2C	$sp=29	$a3	$sp=29	0	

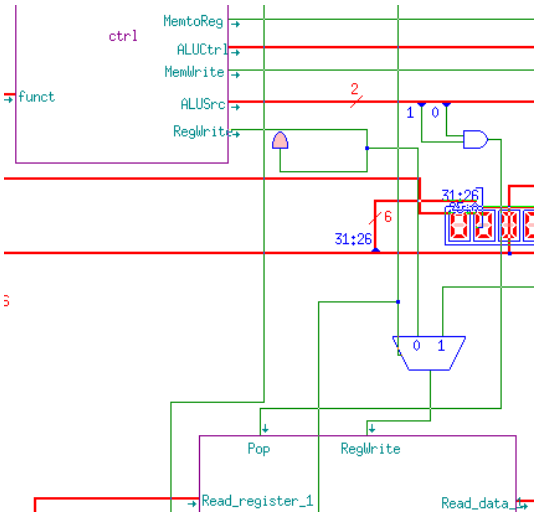
Provarem el seu funcionament posteriorment amb la instrucció pop

TAREA 4.2: POP

Per a la implementació d'aquesta instrucció he hagut de modificar el banc de registres, en concret el registre 29, que és el de l'Stack pointer. A més, he afegit una entrada al mòdul per a indicar-li al banc de registres que la instrucció pop està activada i s'ha de comportar diferent.



A més, miro si la sortida d'ALUSrc és 3 per a identificar que estic utilitzant la instrucció pop.



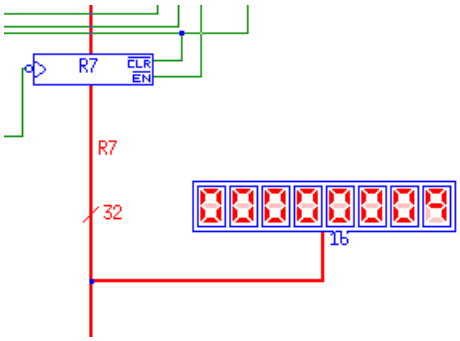
Codificació de la instrucció:

Pop $a3

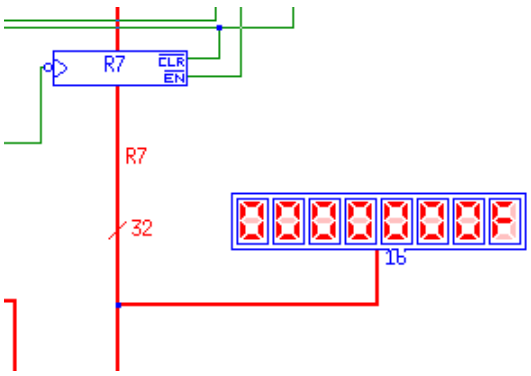
POP $a3	101111	11101	11101	00111	00000000000	BFBD3800
	2F	$sp=29	$sp=29	$a3	0	

Joc de proves

$a3 abans



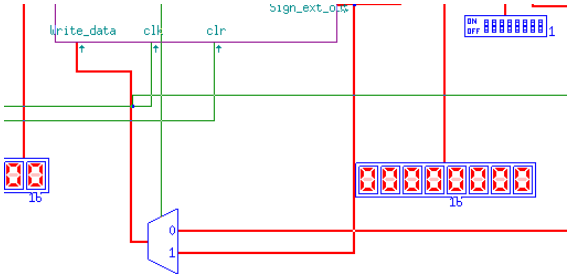
$a3 després

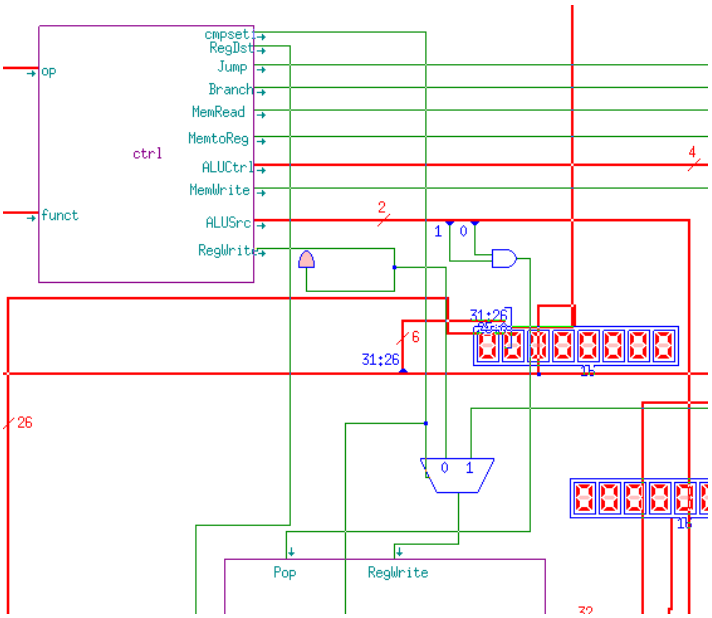


TAREA 4.2: CMPSETI

Per a poder incloure aquesta nova instrucció he hagut d'incloure dos multiplexors per a controlar la senyal de regWrite i l'altre multiplexor controla la dada que arriba per escriure's al banc de registres.

A més, he modificat la UC per a afegir una nova senyal de CMPSETI.





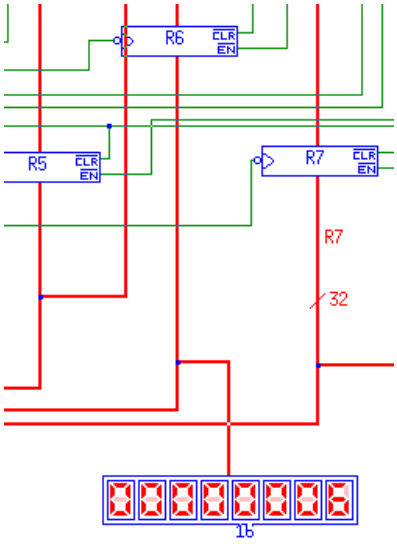
Codificació de la instrucció:

Cmpseti $s0, $a2

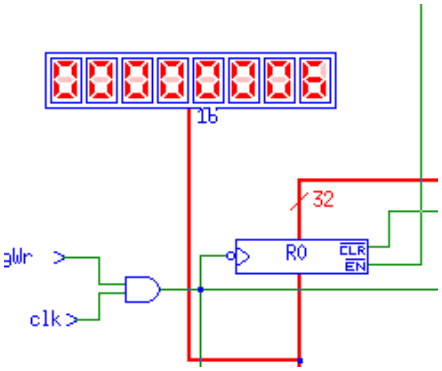
CMPSETI	101101	10000	00110		0000000001100100	B6060064
	2D	$s0	$a2		100	

Joc de proves

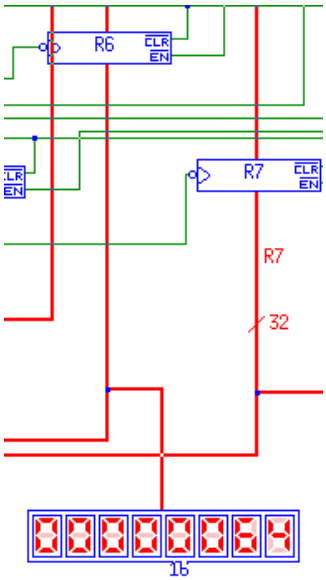
$a2



$s0



Després d'executar cmpseti:



FASE 5: PRUEBA DE FUNCIONAMIENTO

TAREA 5.1

Per a provar el funcionament de les instruccions bàsiques del ISA, he realitzat un programa que consisteix en calcular el factorial d'un número, en concret del número 5.

He realitzat un doble bucle, fent ús de les instruccions AND, OR, LW, ADD, SW, BEQ, SLT.

El fitxer .mem:

```
#main:
00/ 8c090064 00002024 8c050066 00008024 00008824 00009024
 00099825
# perFact:
07/ 0245582A 1160000a 00932020
#per:
0a/ 0232402A 11000003 02649820 02298820 1000fffb
# fiper:
f/ 02499020 00008824 00002024 1000FFF4
#fiperFact:
13/ AC130067
# fi:
14/ 1000ffff

# DATA IN MEMORY
00000064/ 00000001 00000003 00000005 00000000

```

El codi ensamblador:

```
main:
   lw   $t1, 100($zero)   # carga la constante "1"
   # lw   $a0, 101($zero)   # carga el valor de a
   and $a0, $zero, $zero
   lw   $a1, 102($zero)   # carga el valor de b

   and  $s0, $zero, $zero  # s := 0
   and  $s1, $zero, $zero  # i := 0 -> reg 17
   and $s2, $zero, $zero   # j := 0 -> reg 18= 0 AND 0=0
   or $s3, $zero, $t1      # fact:=1-> reg 19= 0 OR 1=1
perFact:
   slt $t3, $s2, $a1
   beq $t3, $zero, fiperFact
   -add $a0, $a0, $s3
per:

   slt  $t0, $s1, $s2       # i < a/j ?
   beq  $t0, $zero, fiper   # si $t0 = 0 pc = fiper

   add  $s3, $s3, $a0   # s := s + b
   add  $s1, $s1, $t1   # i := i + 1

   beq  $zero, $zero, per   # jump per

fiper:
   add $s2, $s2, $t1       # j=j+1-> reg 18= reg 18 + 1
   and $s1, $zero, $zero   # i=0
   -and $a0, $zero, $zero
   beq $zero, $zero, perFact
   # sw $s0, 103($zero)    # salva s
fiperFact:
   sw $s3, 103($zero)      #

fi:
   beq $zero, $zero, fi # bucle infinito
```

TAREA 5.2

Per a comprovar el funcionament d'aquesta fase he realitzat un programa que calcula la resta entre dos números utilitzant un bucle i restant 1 fins que obtenim el resultat de la resta entre els dos.

Al principi del programa he inclòs un push, gairebé al final del programa he ficat un cmpseti per a modificar el valor en cas de que els dos valors siguin iguals.

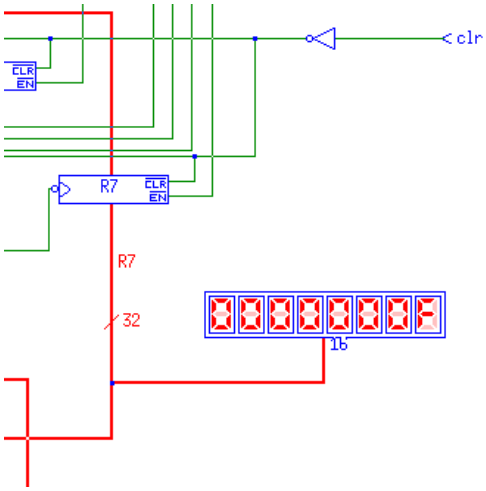
Per finalitzar, he recuperat el valor prèviament salvat utilitzant la instrucció pop.

```
.text
.globl main
#main: inicio del programa

main:

    lw $t1, 100($zero) # n=1
    lw $a1, 101($zero) # b= 15
    lw $a2, 102($zero) # a=6
    lw $a3, 101($zero) # res=15

    and $s0, $zero, $zero # i=0

    push $a3   #B0FDE800    # BFBD3800
    per:
        slt $t0, $s0, $a2
        beq $t0,$zero,fiper
        sub $a3, $a3, $t1
        add $s0,$s0,$t1
        beq $zero,$zero, per
    fiper:
        sw $a3, 103($zero)
        cmpseti $s0, $a2    # B6060064
        pop $a3             # B3A7E800

    fi:
    beq $zero, $zero, fi



    .data

uno: .asciiz "1"
a: .asciiz "15"
b: .asciiz "6"
s: .asciiz "x"
```

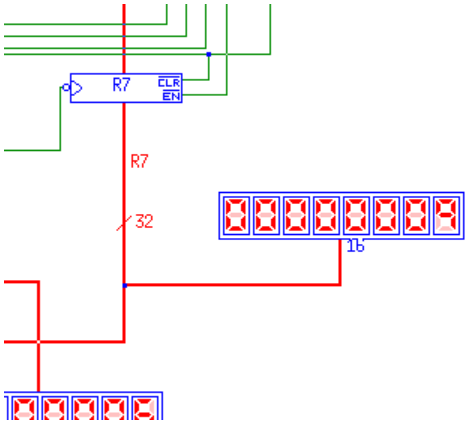
```
#main:
00/ 8c090064 8c050065 8c060066  8c070065 00008024 B3A7E800
# per:
06/ 0206402a 11000003 00e93822 02098020 1000fffb
# fiper:
0b/ ac070067 B6060064 BFBD3800
# fi:
0e/ 1000ffff

# DATA IN MEMORY
00000064/ 00000001 0000000f 00000006 00000000

```

Joc de proves

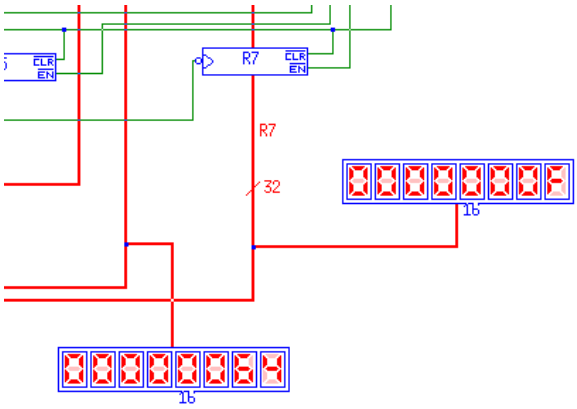
PUSH: salvem valor del registre 7 per a després recuperar-lo.



Abans de recuperar el valor



Valor recuperat



TAREA 5.3

La principal diferència del processador monocicle respecte els altres processadors és el número de cicles que triga cada instrucció i el temps de cicle.

Al processador monocicle cada instrucció triga un cicle en executar-se i el temps de cicle és alt, al multicicle, en canvi, el que faríem seria dividir la instrucció en diversos passos, i a cada cicle executaríem un d'aquests passos, donant-nos un CPI superior a 1 i per això un temps de cicle inferior.

Al processador segmentat passaria una cosa semblant, però a més, a cada cicle iniciaríem una nova execució d'instrucció, així donant-nos un millor rendiment que els processadors mencionats anteriorment.

Per últim, el processador superescalar inclou totes les millores anteriors, però enlloc d'iniciar una instrucció per cicle només, inicia varies, suposant així una millora molt notable.

Hem de tenir en compte que el processador superescalar també inclou predicció de salts, que ens suposa una gran millora, tenint en compte l'alta efectivitat dels predictors avui dia.

En conclusió, el processador monocicle és correcte per a entendre el funcionament a baixa escala, però crec que resultarà més interessant l'ús d'un processador superescalar, ja que ens permetrà optimitzar el temps i realitzar tasques en molt menys temps.

FASE 6: ANÁLISIS DE MEMORIA

Per a la realització d'aquesta pràctica, he fet un script per a cada test on executo de forma automàtica els tests corresponents per a cadascuna de les cachés per als tres processadors.

A mesura que vaig fent els tests, mitjançant la comanda grep, redirecciono a un altre fitxer les dades que m'interessen de cada test, que és la tassa de misses i la informació de la caché personalitzada.

A més, faig un script que s'encarrega d'executar tots els altres scripts, en cas de voler executar tots els scripts d'una vegada.

Aquí adjunto alguna captura per a que es pugui veure l'estructura:

```
#! /bin/bash

cd /lib/specs2000/art/data/ref

art="/home/milax/Documents/GitHub/ECPrac2/Prac2aConvo/FASE6/art/"

intelDir="/home/milax/Documents/GitHub/ECPrac2/Prac2aConvo/FASE6/art/Intel/"
ryzen7Dir="/home/milax/Documents/GitHub/ECPrac2/Prac2aConvo/FASE6/art/Ryzen7"
ryzen7_2Dir="/home/milax/Documents/GitHub/ECPrac2/Prac2aConvo/FASE6/art/Ryzen7Zen"

#TASCA 6.2
#Core i7-7600U – Intel
processor="Intel"

#IL1
tasca="Tasca6-2_IL1i7.txt"
echo $tasca
sim-outorder -fastfwd 100000000 -max:inst 100000000 -cache:il1 CACHE_IL1:64:64:8:l -redir:sim $intelDir/$tasca /lib/specs2000/art/exe/art.exe -scanfile c756hel.in -trainfile1 a10.img -stride 2 -startx 134 -
  starty 220 -endx 139 -endy 225 -objects 1 > $art/test.out 2> $art/test.err

grep "CACHE" $intelDir/$tasca > $intelDir/CACHE$processor

#DL1
tasca="Tasca6-2_DL1i7.txt"
echo $tasca
sim-outorder -fastfwd 100000000 -max:inst 100000000 -cache:dl1 CACHE_DL1:64:64:8:l -redir:sim $intelDir/$tasca /lib/specs2000/art/exe/art.exe -scanfile c756hel.in -trainfile1 a10.img -stride 2 -startx 134 -
  starty 220 -endx 139 -endy 225 -objects 1 > $art/test.out 2> $art/test.err

grep "CACHE" $intelDir/$tasca >> $intelDir/CACHE$processor

#DL2
tasca="Tasca6-2_DL2i7.txt"
echo $tasca
sim-outorder -fastfwd 100000000 -max:inst 100000000 -cache:dl2 CACHE_L2:1024:64:4:l -redir:sim $intelDir/$tasca /lib/specs2000/art/exe/art.exe -scanfile c756hel.in -trainfile1 a10.img -stride 2 -startx 134 -
  starty 220 -endx 139 -endy 225 -objects 1 > $art/test.out 2> $art/test.err

grep "CACHE" $intelDir/$tasca >> $intelDir/CACHE$processor
```

Mobile processor-Core i7-7600U – Intel

Processor: https://en.wikichip.org/wiki/intel/core_i7

Cache: https://en.wikichip.org/wiki/intel/microarchitectures/kaby_lake#Memory_Hierarchy

INFO:

L1I Cache (Cache L1 d'instruccions):

Tamany: 32KB=32768

Associativitat: 8 way set-associative

Sets: 64 sets

Block size: 64B

L1D Cache (Cache L1 de datos):

Tamany: 32KB

Associativitat: 8 way set-associative

Sets: 64 sets

Block size: 64B

L2 Cache:

Tamany: 256KB=262144

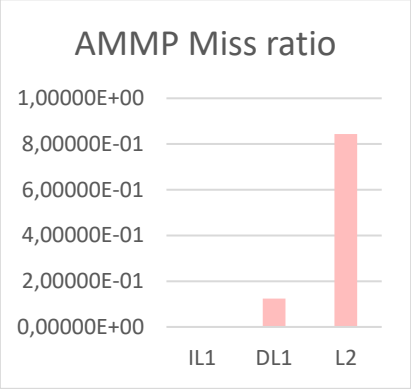
Associativitat: 4 way set-associative

Sets: 1024 sets

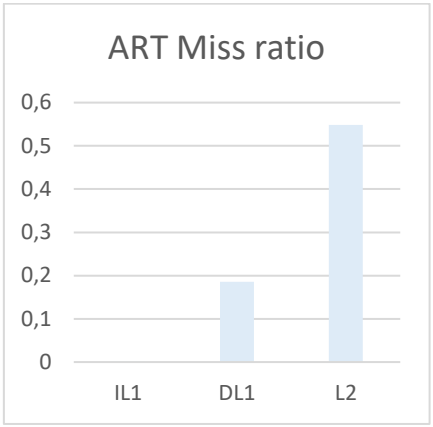
Block size: 64B

Anàlisi TASCA 6-2

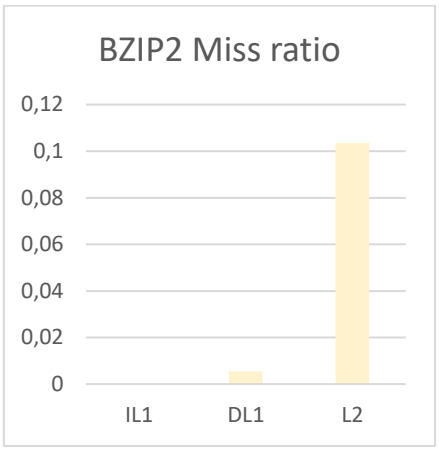
AMP



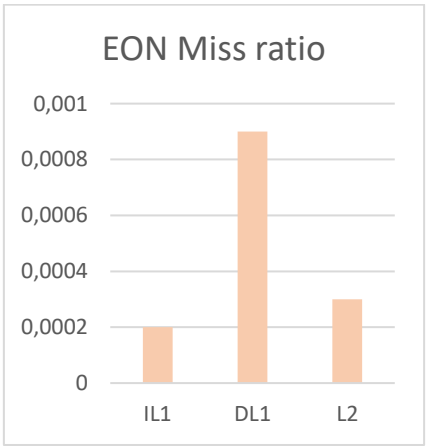
ART



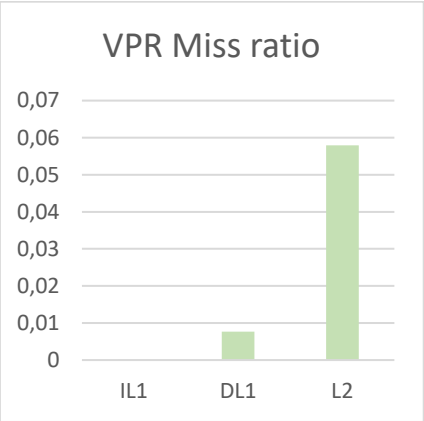
BZIP2



EON

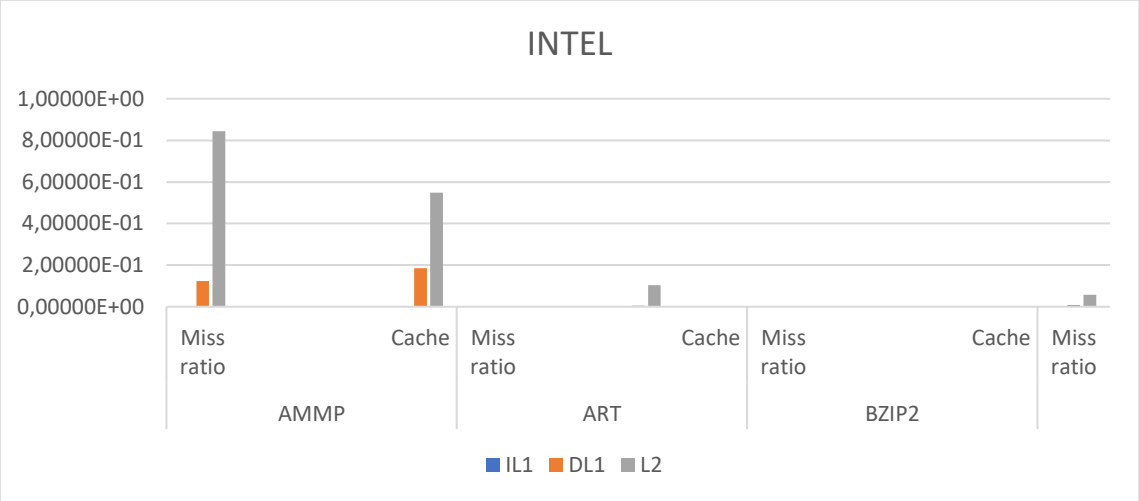


VPR



Anàlisi TASCA 6-3

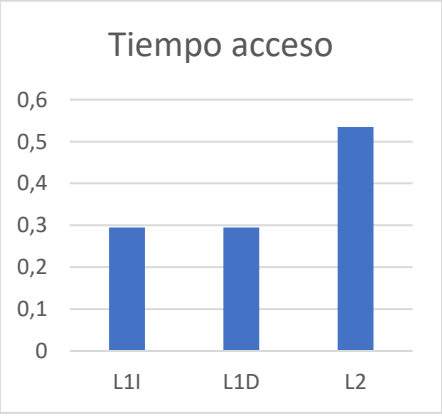
COMPARACIÓ MISS RATIO TESTS



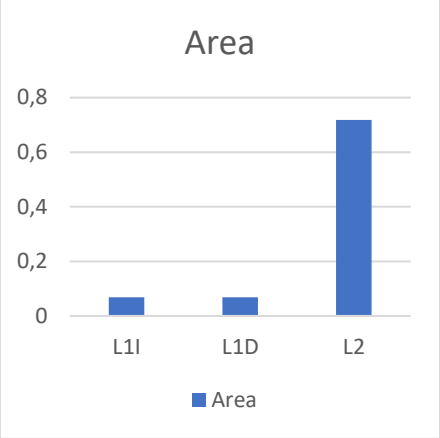
Podem veure que la caché L2 es produeixen més misses que a qualsevol altre cache, sent la caché de dades DL1 la segona amb més tassa d'error d'entre les 3 caches avaluades.

Anàlisi TASCA 6-3

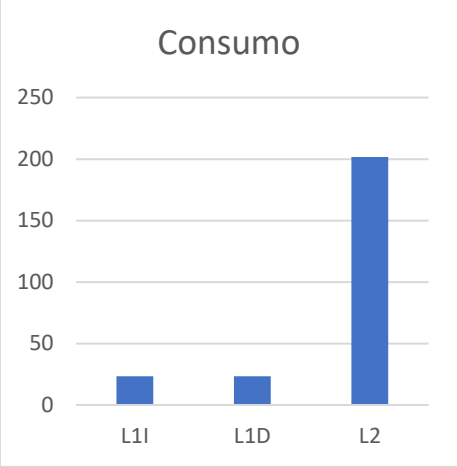
Temps d'accés



Àrea



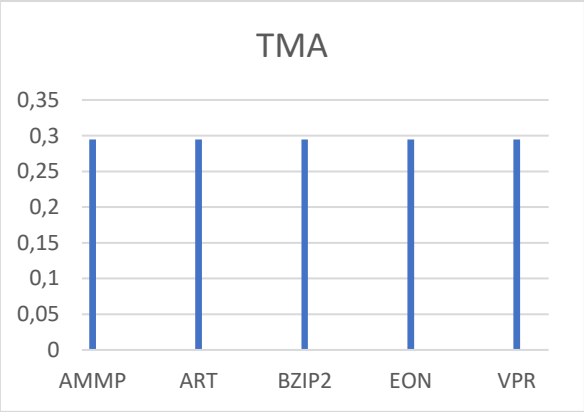
Consum



Anàlisi TASCA 6-4

Tacceso_medio = αTc + (1-α)(Tc+Tm) = Tc + (1-α)Tm

Tc: tiempo acceso caché **Tm: tiempo acceso memoria principal**



	AMMP			ART	
Cache	Miss ratio	Hit	Cache	Miss ratio	Hit
IL1	2,20806E-06	9,99998E-01	IL1	4,27882E-07	0,99999957
DL1	0,1235	8,76500E-01	DL1	0,1855	0,8145
L2	0,8437	1,56300E-01	L2	0,5483	0,4517
TMA	=D7*$B42+C7*D8*($B42+$B43)+C7*C8*D9*($B42+$B43+$B44)+C7*C8*C9*($B42+$B43+$B44+100)				

Desktop processor-Ryzen 7 3700U – AMD

Processor: https://en.wikichip.org/wiki/amd/ryzen_7/3700u

Cache: https://en.wikichip.org/wiki/amd/microarchitectures/zen%2B#Memory_Hierarchy

INFO:

L1I Cache (Cache L1 d'instruccions):

Tamany: 64KB=65536

Associativitat: 4 way set-associative

Sets: 256 sets

Block size: 64B

L1D Cache (Cache L1 de datos):

Tamany: 32KB

Associativitat: 8 way set-associative

Sets: 64 sets

Block size: 64B

L2 Cache:

Tamany: 512KB=524288

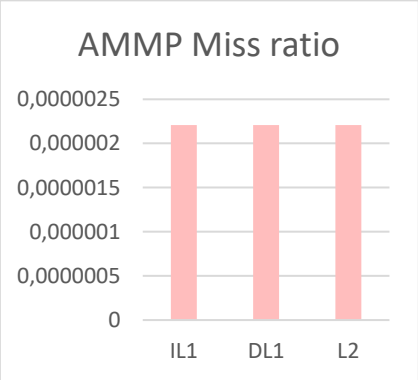
Associativitat: 8 way set-associative

Sets: 1024 sets

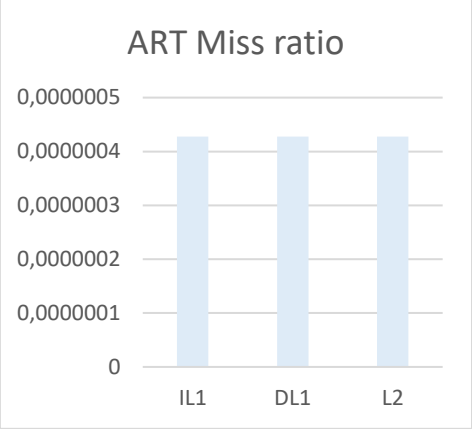
Block size: 64B

Anàlisi TASCA 6-2

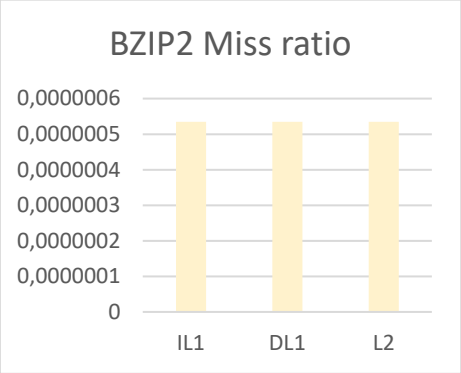
AMP



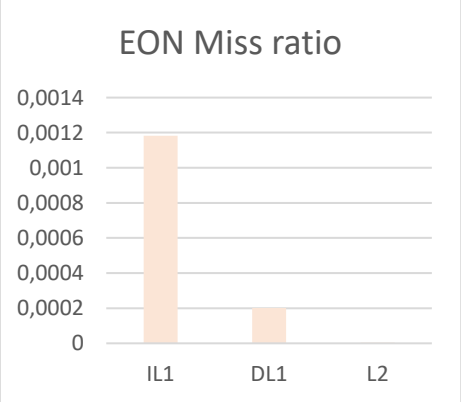
ART



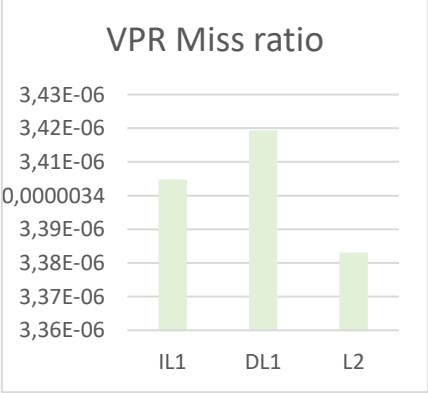
BZIP2

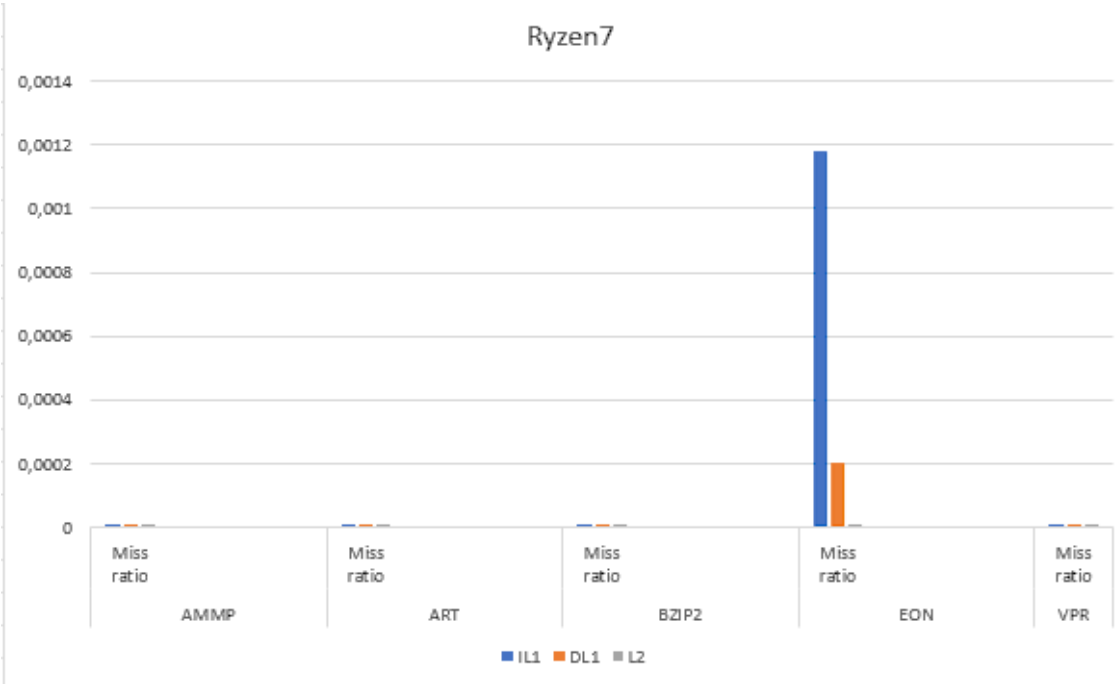


EON



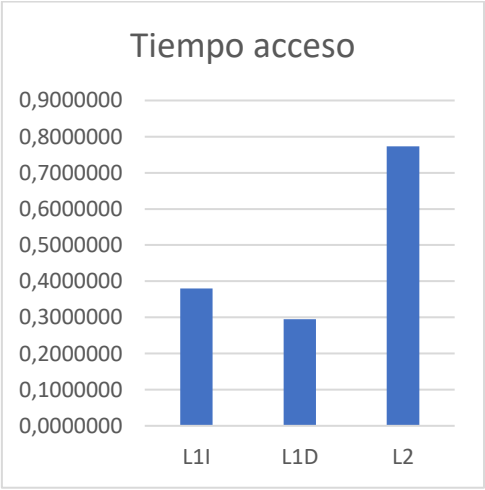
VPR

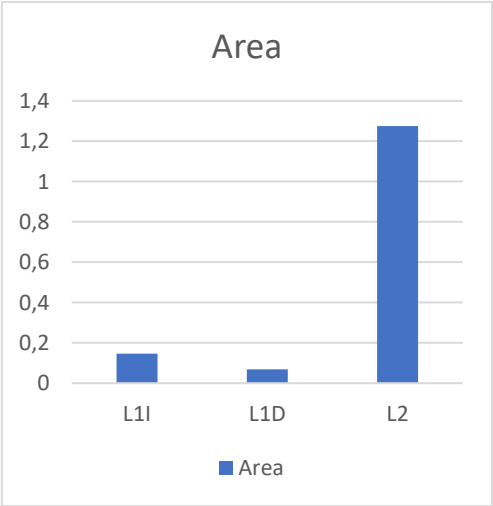


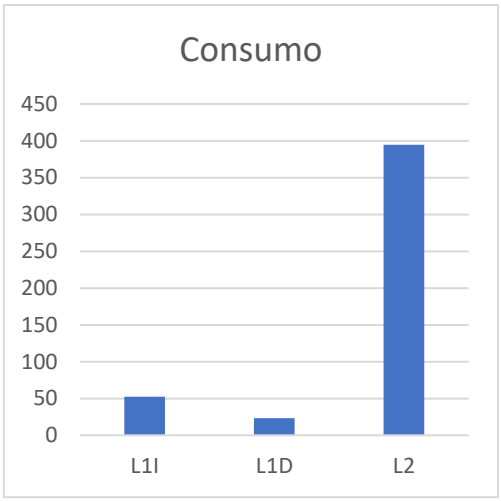


Podem veure que els miss ratio de les cachés al benchmark EON fan que els valors dels altes benchmarks siguin gairebé despreciables com podem veure a la gràfica.

Anàlisi TASCA 6-3

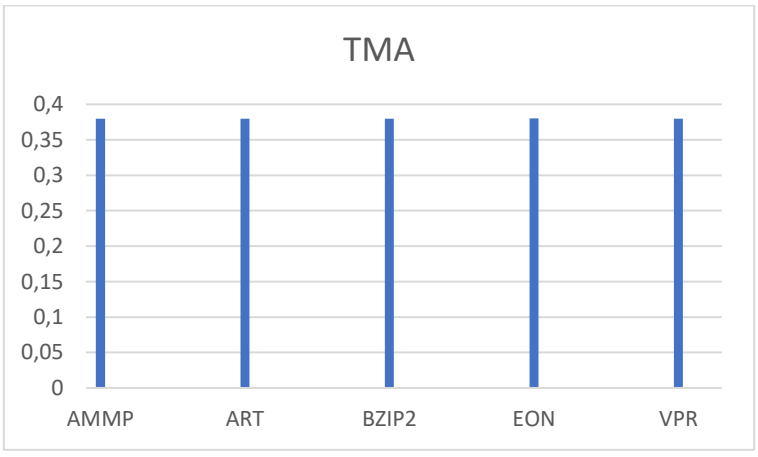






Anàlisi TASCA 6-4

	AMMP			ART	
Cache	Miss ratio	Hit	Cache	Miss ratio	Hit
IL1	2,20806E-06	0,99999779	IL1	4,27882E-07	0,99999957
DL1	2,20806E-06	0,99999779	DL1	4,27882E-07	0,99999957
L2	2,20806E-06	0,99999779	L2	4,27882E-07	0,99999957
TMA	=D7*$B42+C7*D8*($B42+$B43)+C7*C8*D9*($B42+$B43+$B44)+C7*C8*C9*($B42+$B43+$B44+100)				



Console processor-Ryzen 7 Zen 2 – AMD

Processor Zen+:
https://en.wikichip.org/wiki/amd/microarchitectures/zen%2B#:~:text=%5Bedit%5D-,Cache,-L0%20%C2%B5OP%20cache

Processor **Zen2:** https://en.wikichip.org/wiki/amd/ryzen_7#Zen_2:~:text=Count%3A%204-,Zen%202,-%5Bedit%5D

Cache info Zen2:
https://en.wikichip.org/wiki/amd/microarchitectures/zen_2#:~:text=%5Bedit%5D-,Cache,-L0%20Op%20Cache

Niveles:

L1I Cache (Cache L1 d'instruccions):

Tamany: 32KB 32768

Associativitat: 8 way set-associative

Sets: 64 sets

Block size: 64B

L1D Cache (Cache L1 de datos):

Tamany: 32KB= 32768

Associativitat: 8 way set-associative

Sets: 64 sets

Block size: 64B

L2 Cache:

Tamany: 512KB=524288

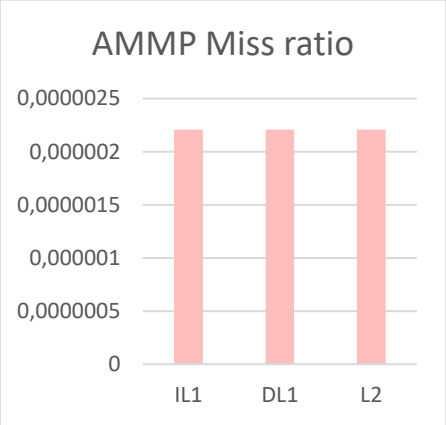
Associativitat: 8 way set-associative

Sets: 1024 sets

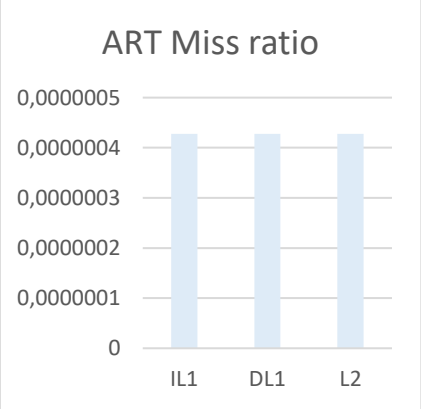
Block size: 64B

Anàlisi TASCA 6-2

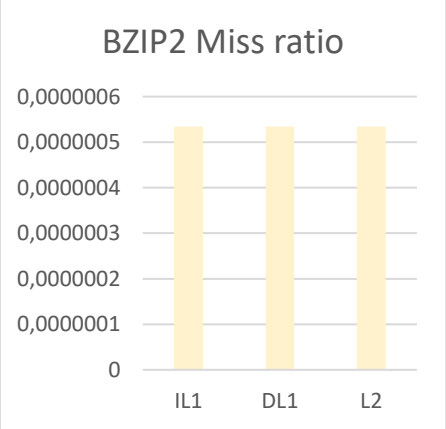
AMP



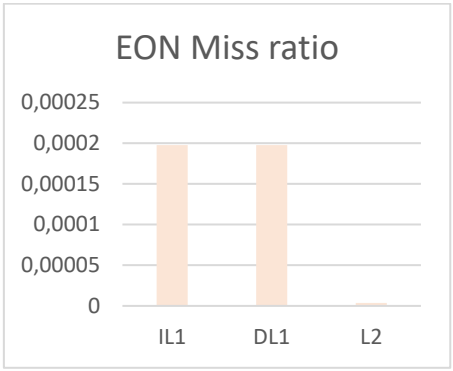
ART



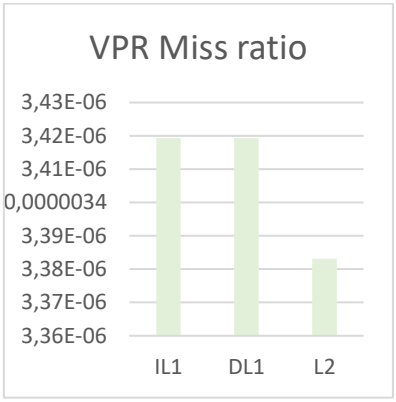
BZIP2

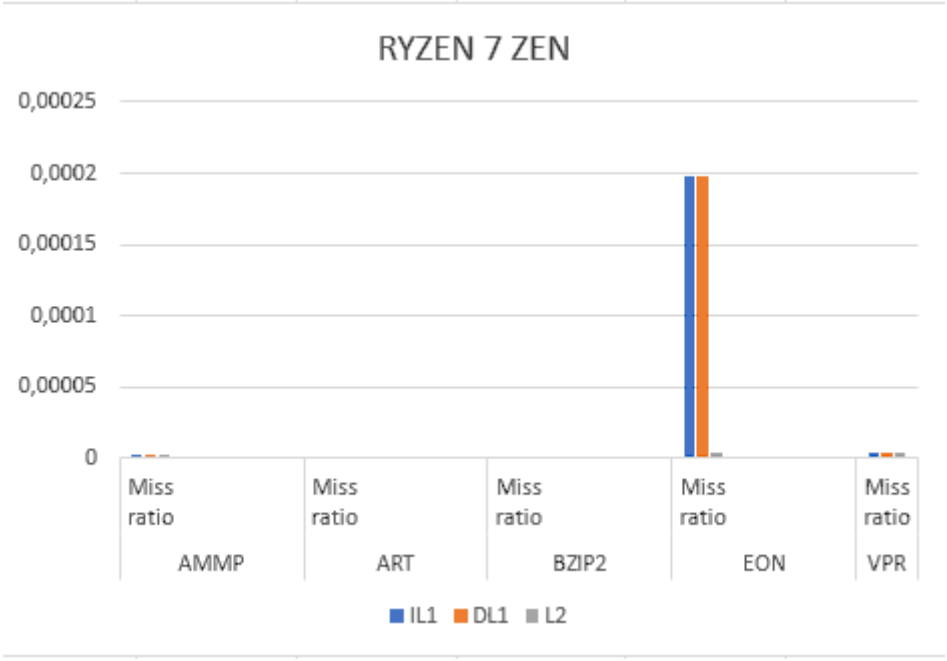


EON



VPR

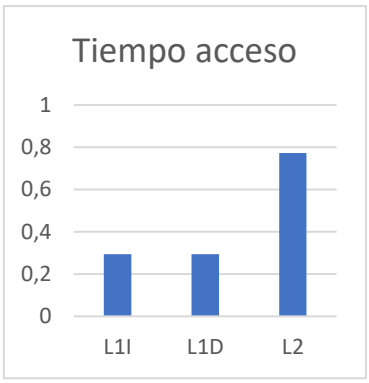


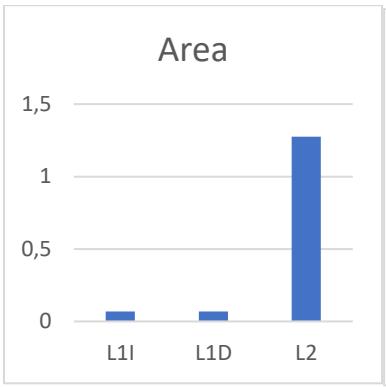


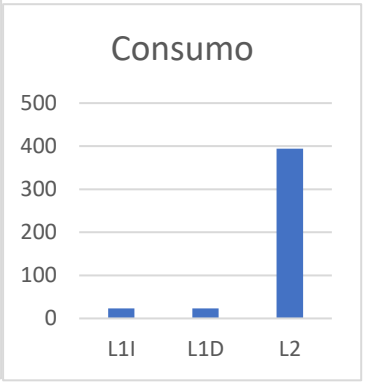
Podem veure , que, un altre cop obtenim els valors més alts al test EON.

Veiem que els dos processadors son Ryzen7, tot i que que tenen cachés diferents i diferents nivells d'associativitat.

Anàlisi TASCA 6-3

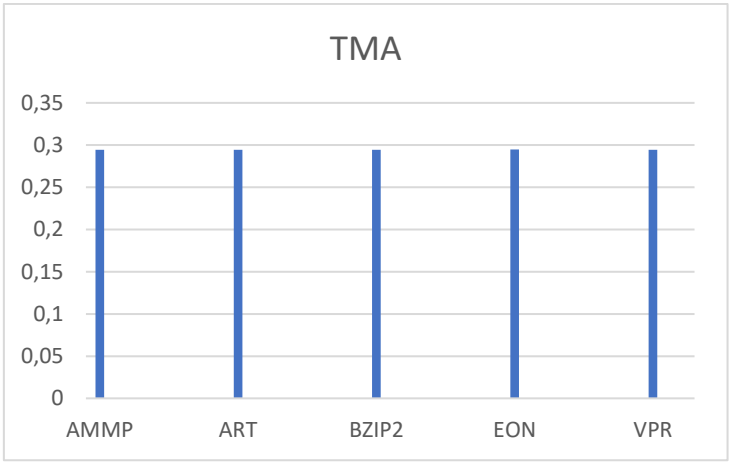




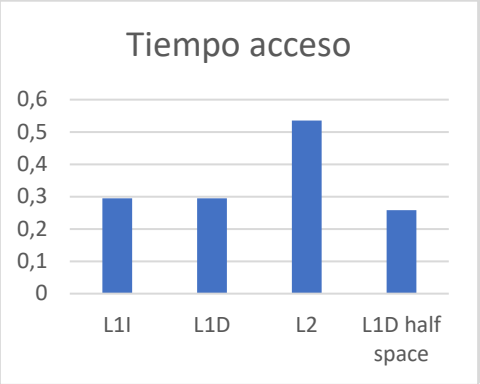


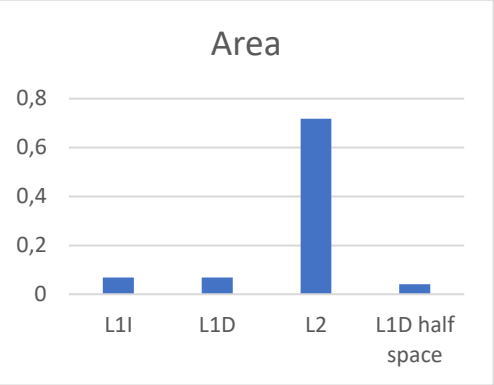
Anàlisi TASCA 6-4

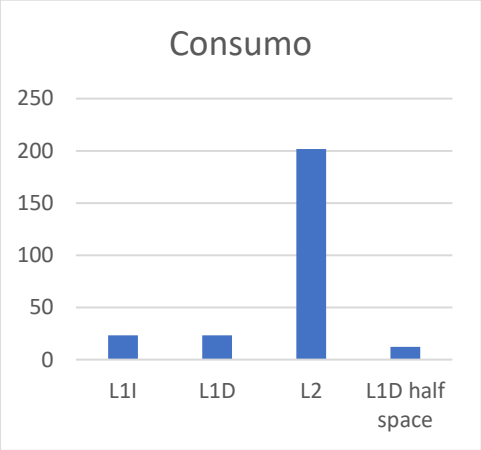
	AMMP			ART	
Cache	Miss ratio	Hit	Cache	Miss ratio	Hit
IL1	2,20806E-06	0,99999779	IL1	4,27882E-07	0,99999957
DL1	2,20806E-06	0,99999779	DL1	4,27882E-07	0,99999957
L2	2,20806E-06	0,99999779	L2	4,27882E-07	0,99999957
TMA	=D7*$B42+C7*D8*($B42+$B43)+C7*C8*D9*($B42+$B43+$B44)+C7*C8*C9*($B42+$B43+$B44+100)				



TASCA 6-5



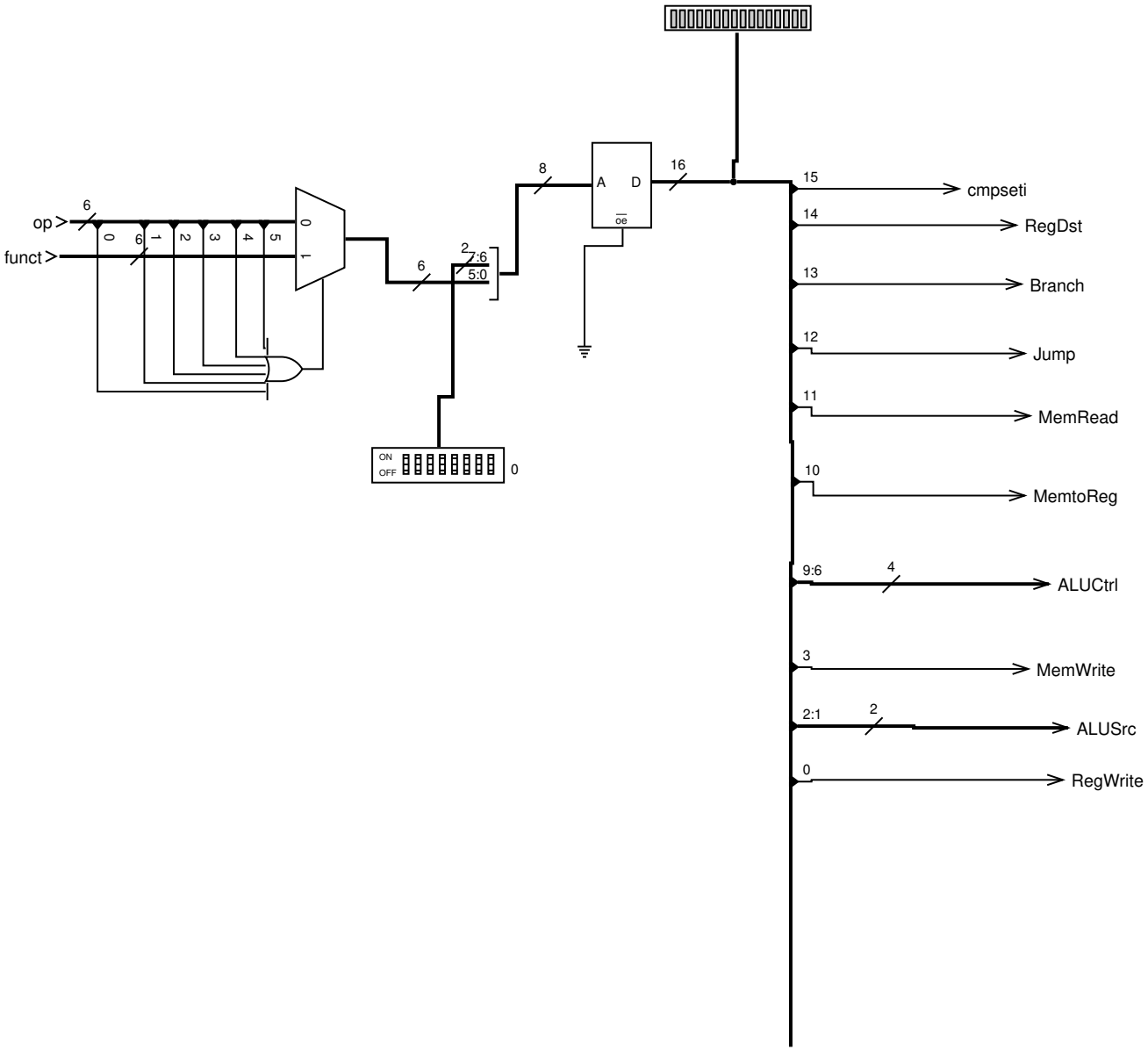




Com podem veure, reduir la mida de la caché a la meitat, ens provoca un decrement en el temps d'accés, d'àrea i de consum.

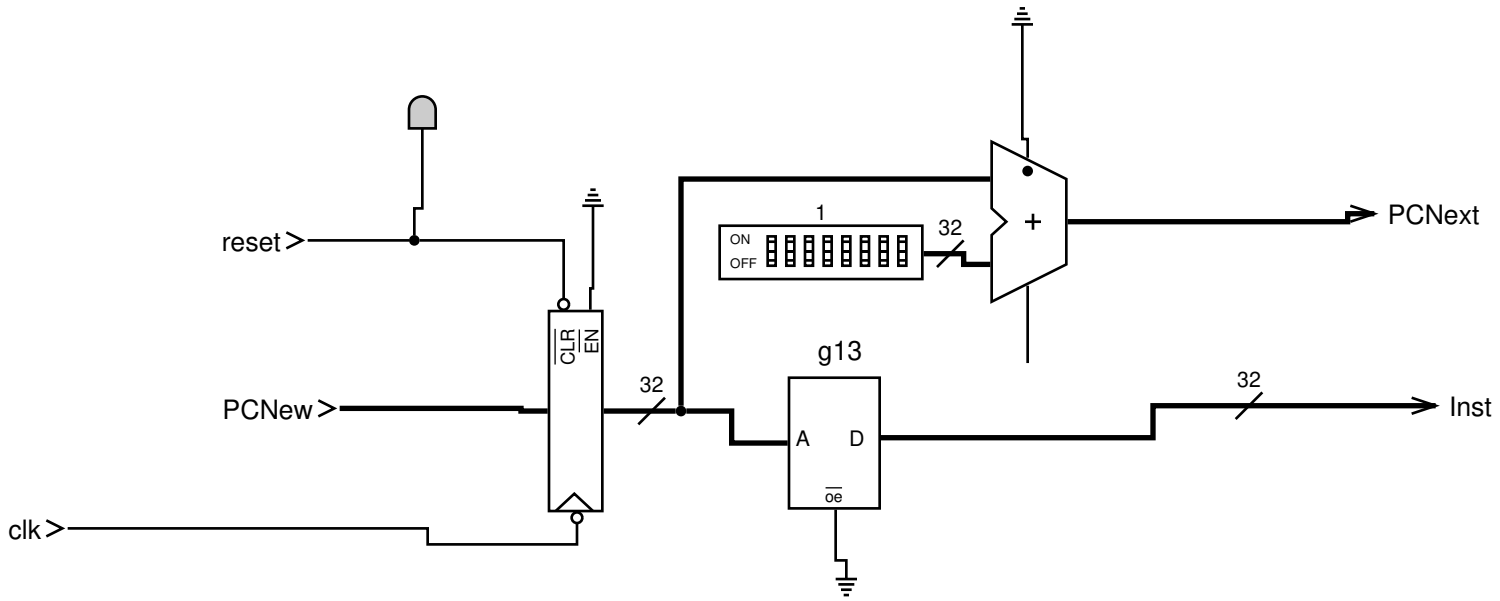
Per una banda, podríem concloure que es una bona solució, ja que ens redueix els temps d'accés i consum, cosa que a gran escala és important, però estaríem perdent capacitat per a emmagatzemar dades i potser això ens suposa un inconvenient en cas de voler guardar moltes dades.

CIRCUITS FASES 2-3-4



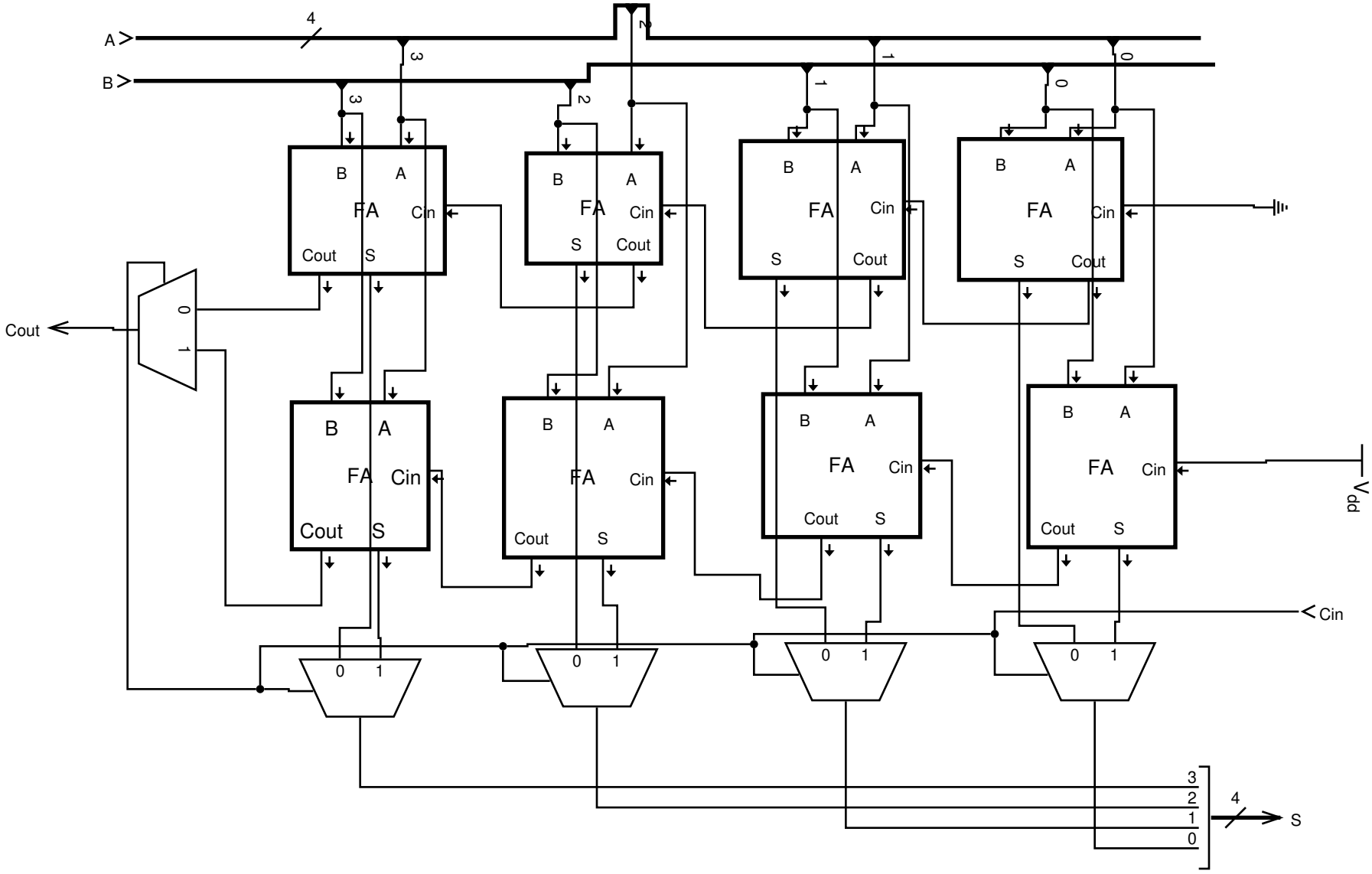
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	1 of 19
Date:	June 21, 2022	**Module:**	ctrl

TKGate



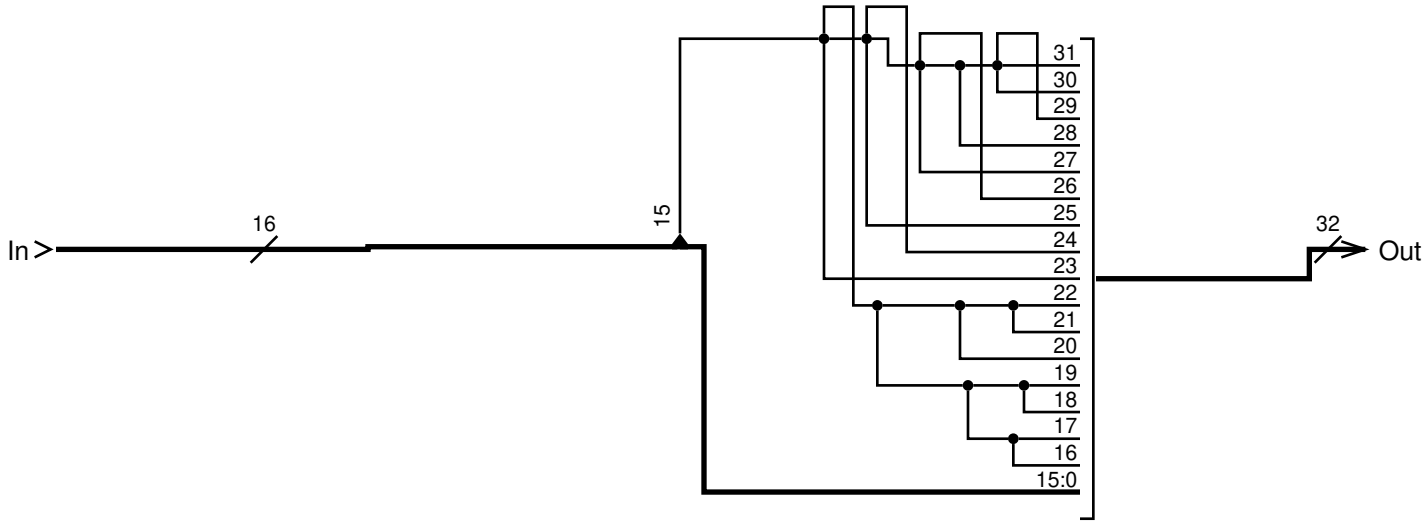
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	2 of 19
Date:	June 21, 2022	**Module:**	fetch

TKGate



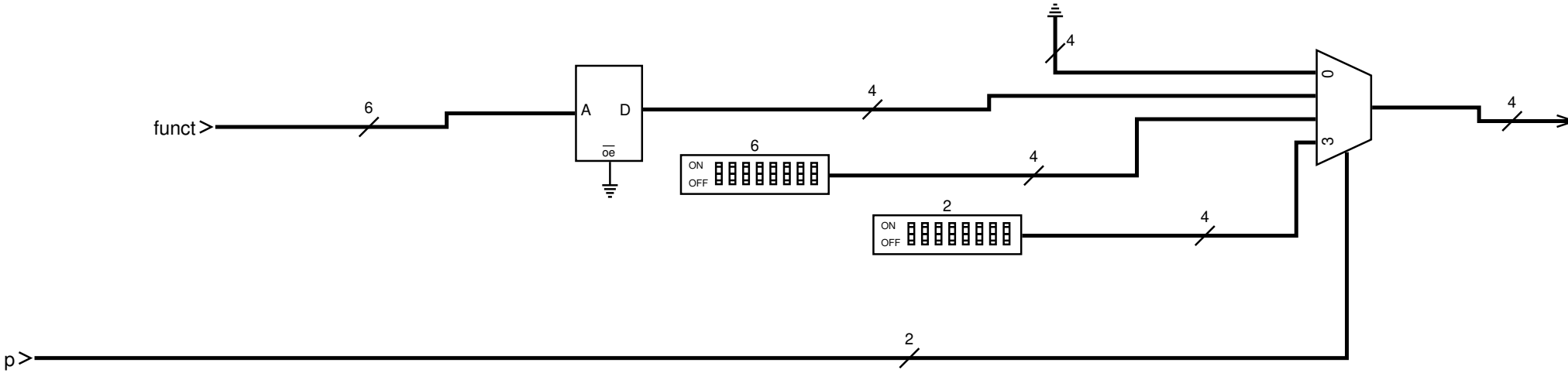
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	3 of 19
Date:	June 21, 2022	**Module:**	CSA

TKGate



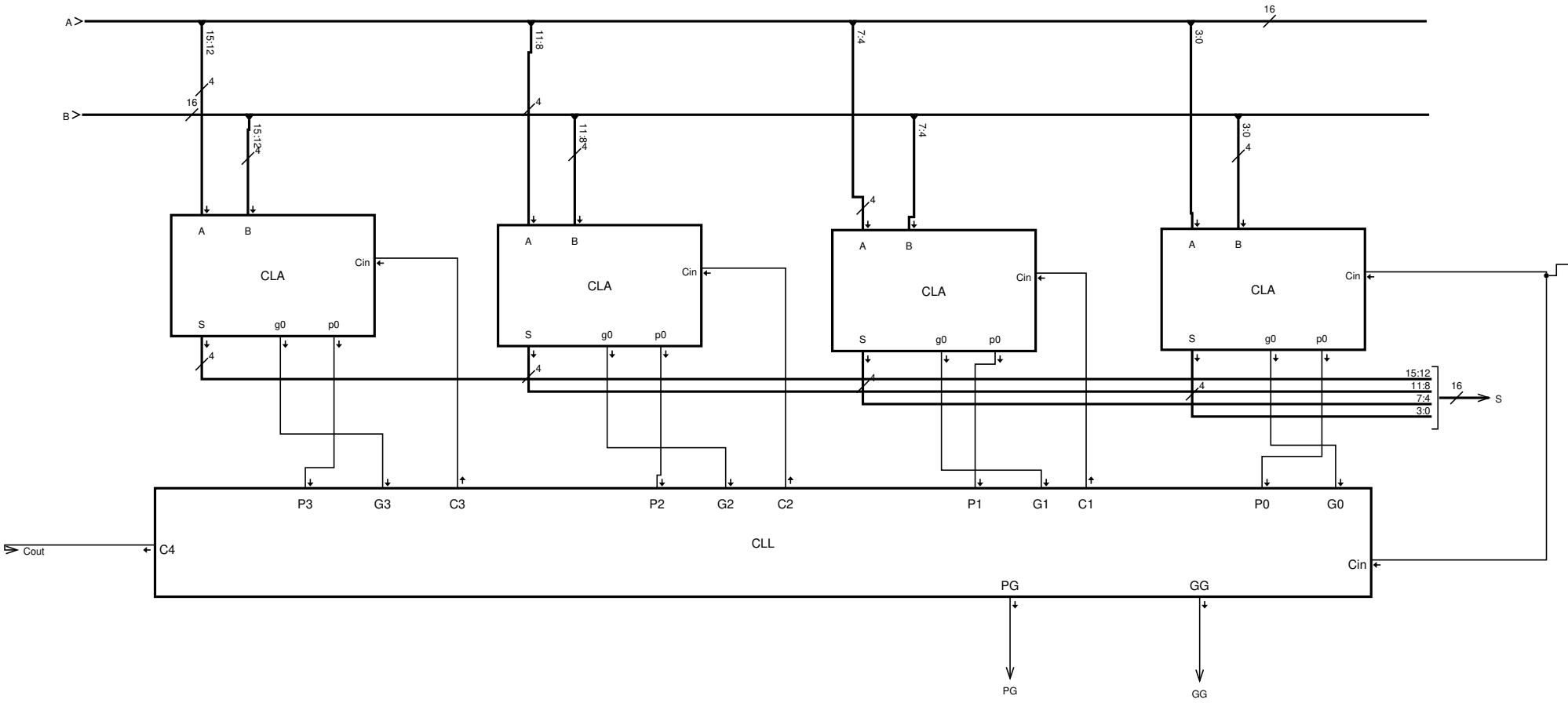
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	4 of 19
Date:	June 21, 2022	**Module:**	signextend

TKGate



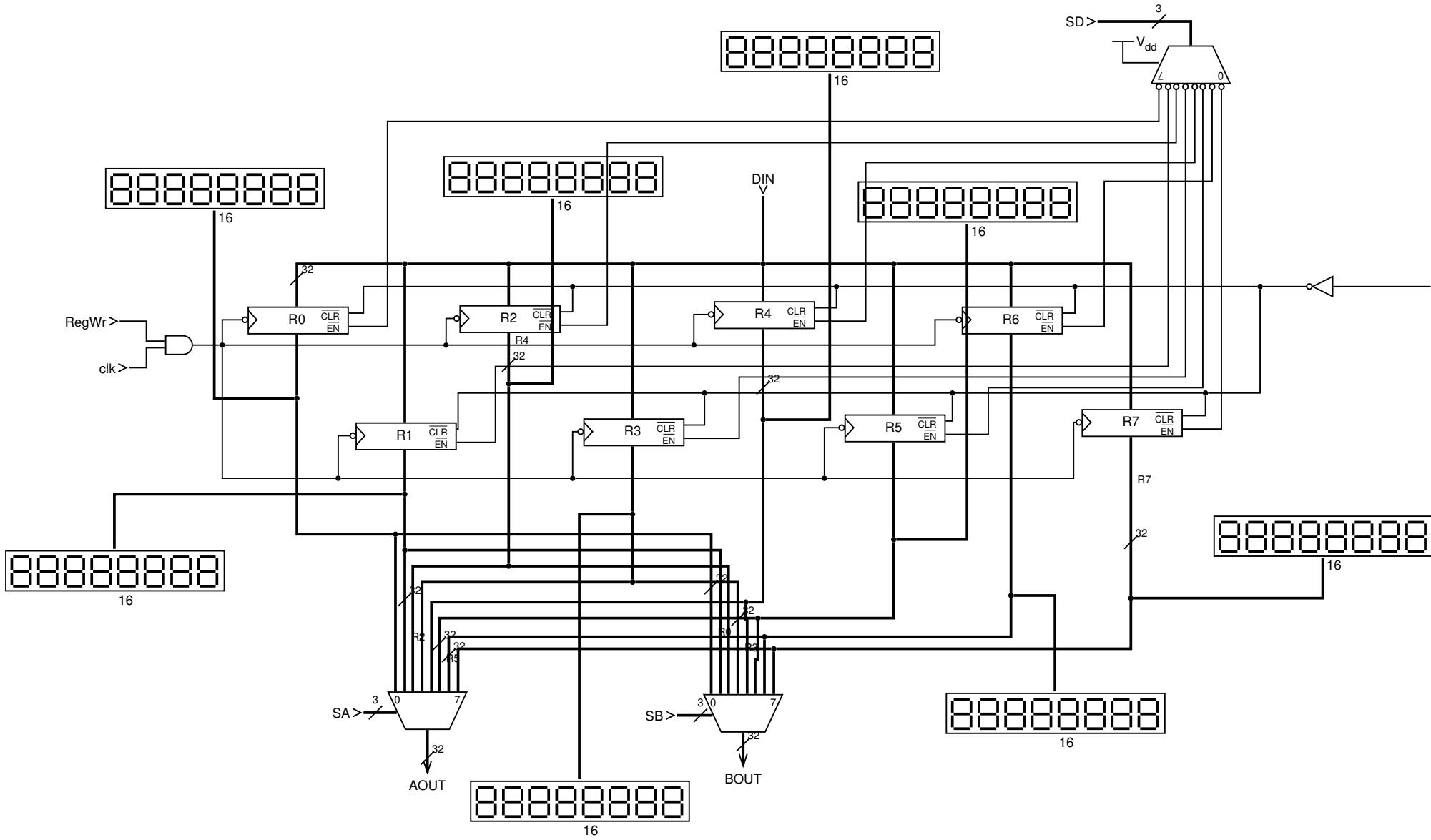
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	5 of 19
Date:	June 21, 2022	**Module:**	ALUCtrl

TKGate



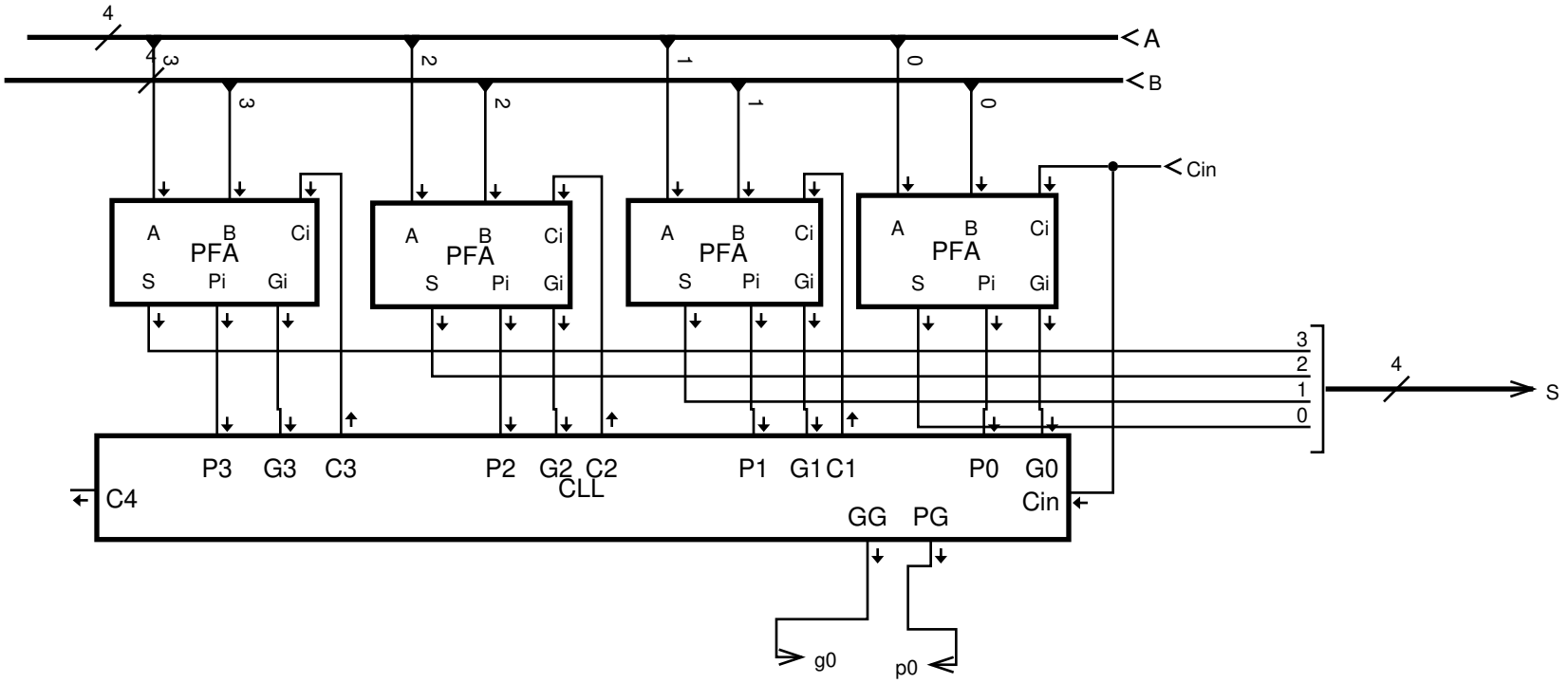
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	6 of 19
Date:	June 21, 2022	**Module:**	CLA16Bits

TKGate



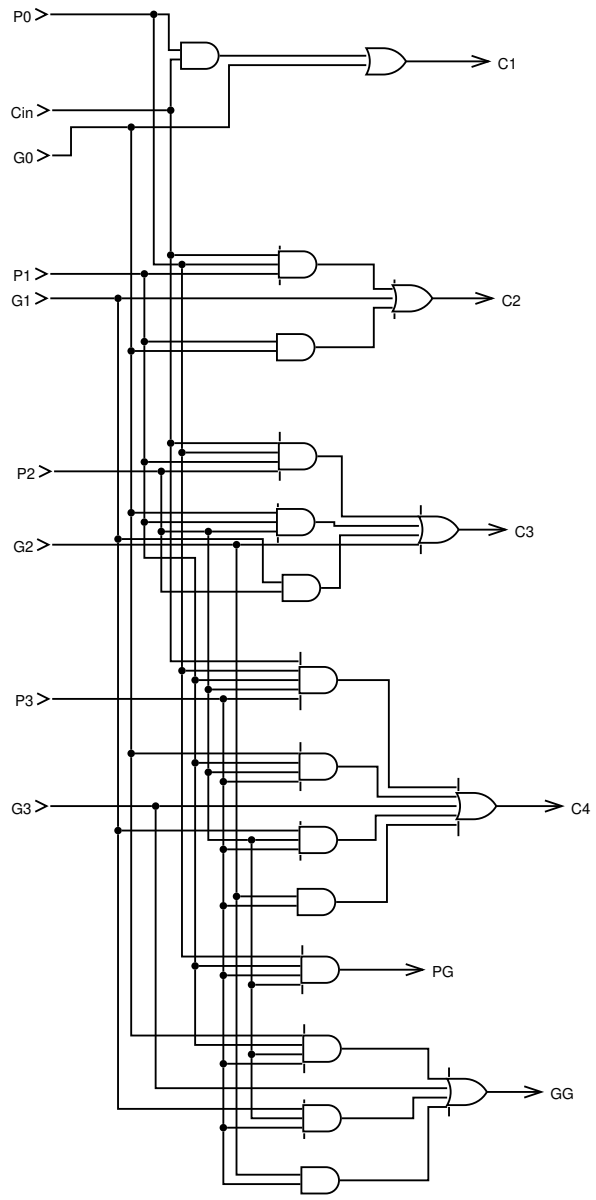
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	7 of 19
Date:	June 21, 2022	**Module:**	Regs8x32

TKGate



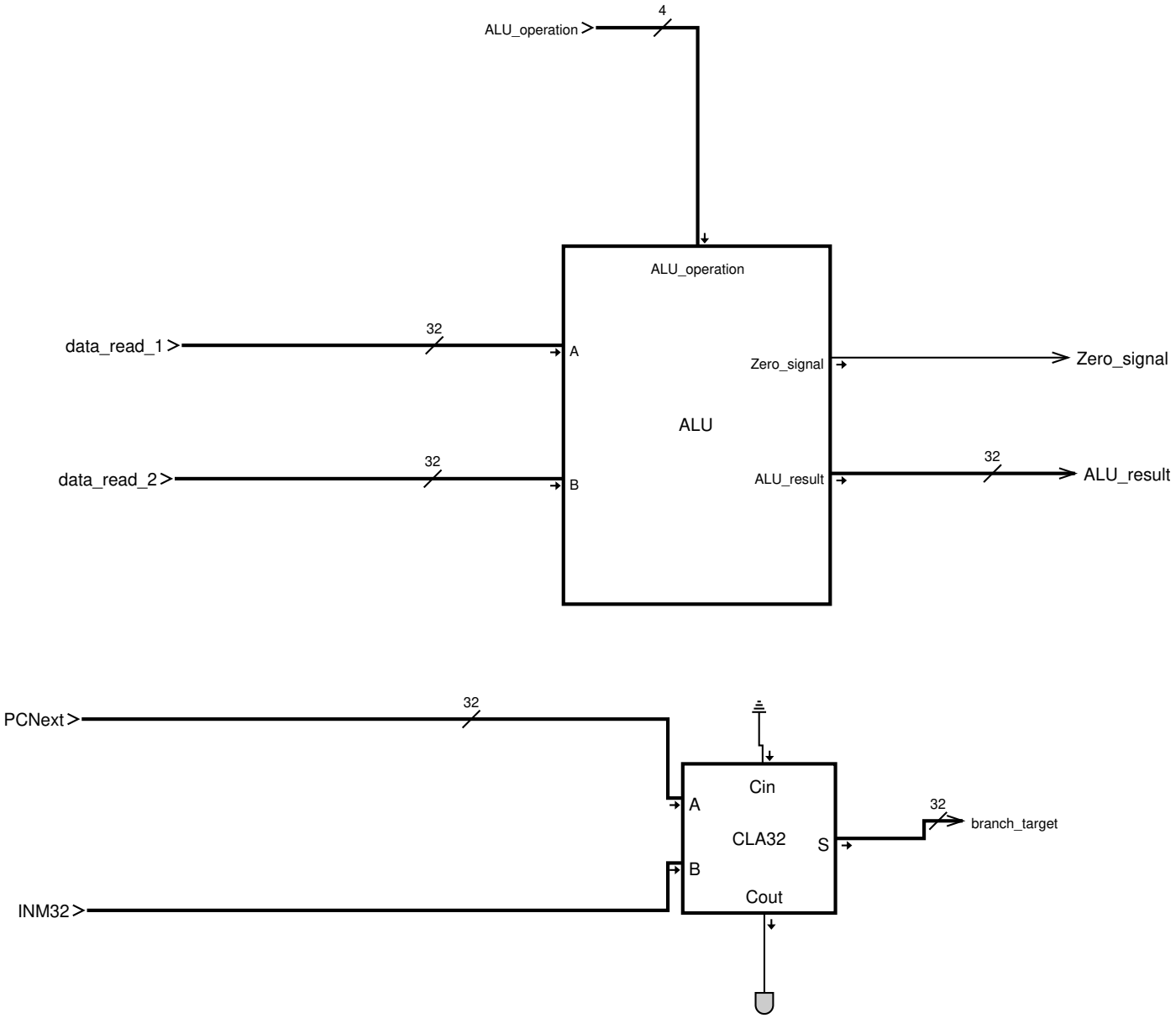
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	8 of 19
Date:	June 21, 2022	**Module:**	CLA

TKGate



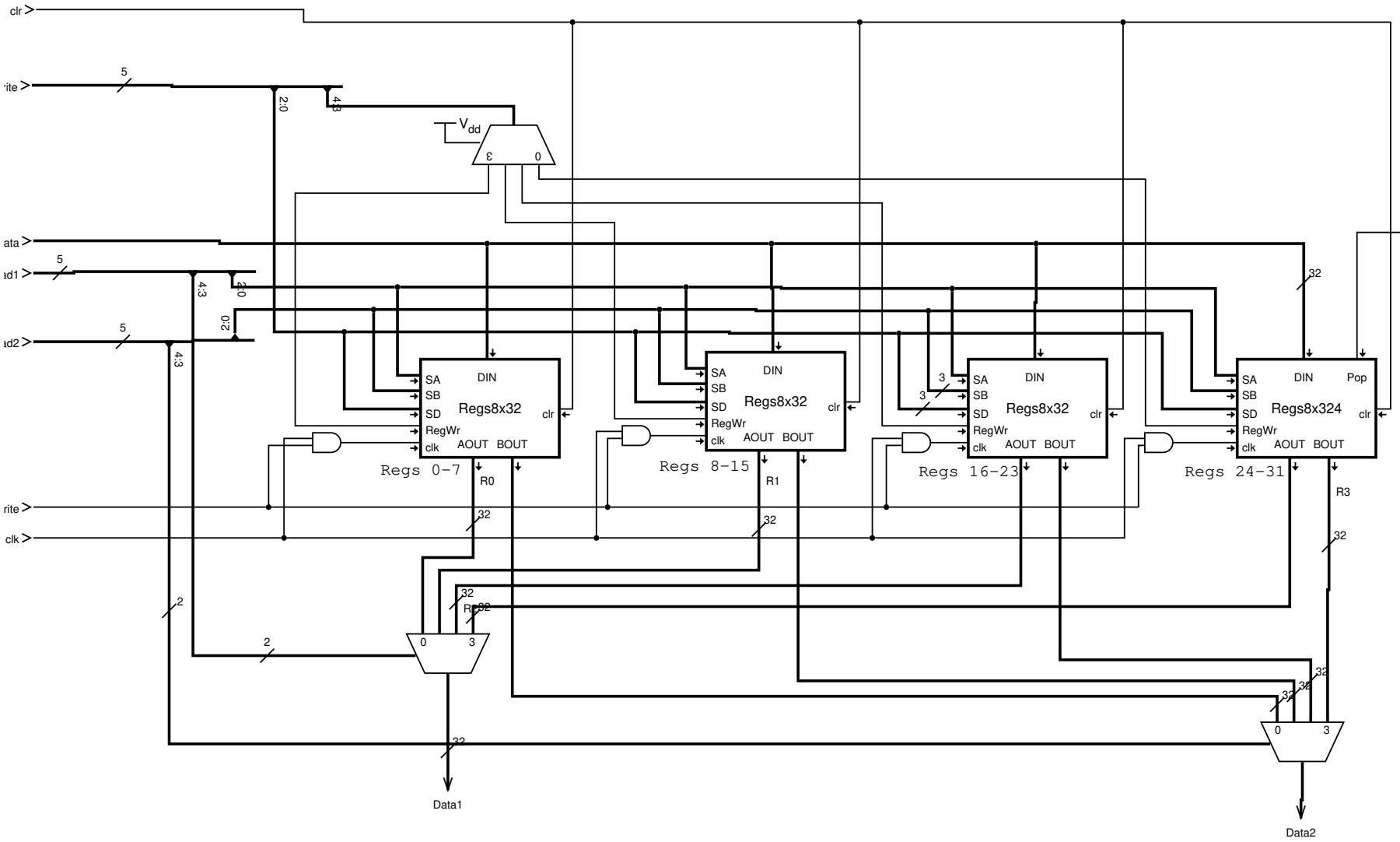
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	9 of 19
Date:	June 21, 2022	**Module:**	CLL

TKGate



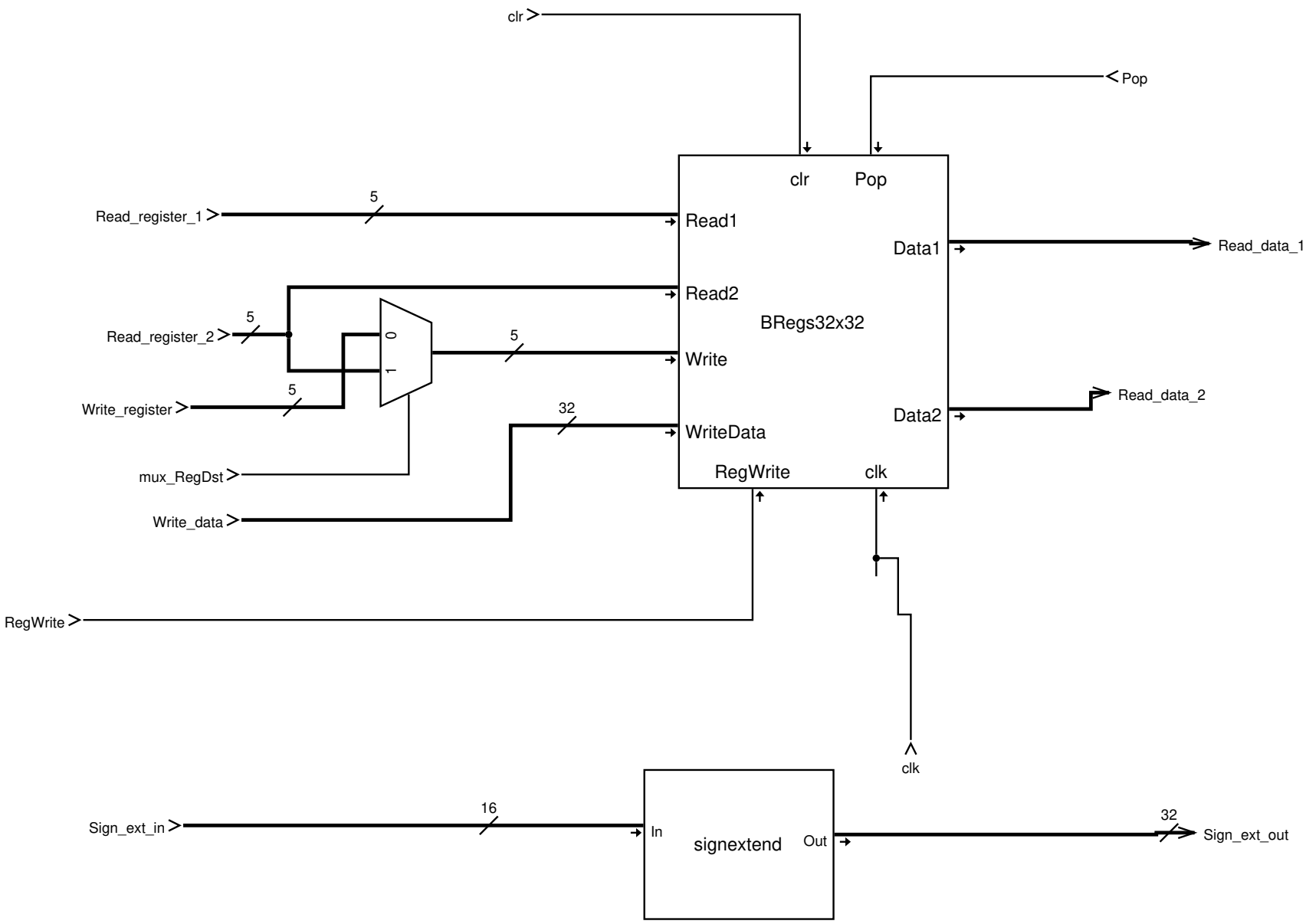
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	10 of 19
Date:	June 21, 2022	**Module:**	EXE

TKGate



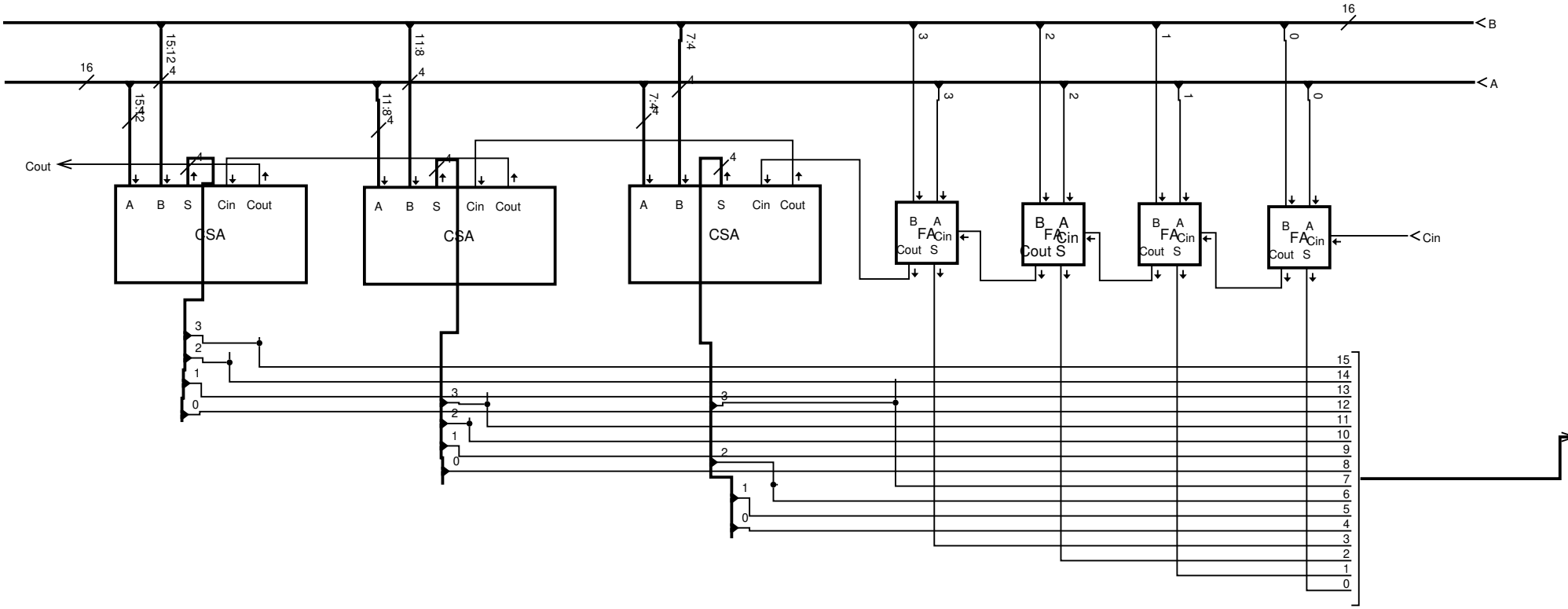
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	11 of 19
Date:	June 21, 2022	**Module:**	BRegs32x32

TKGate



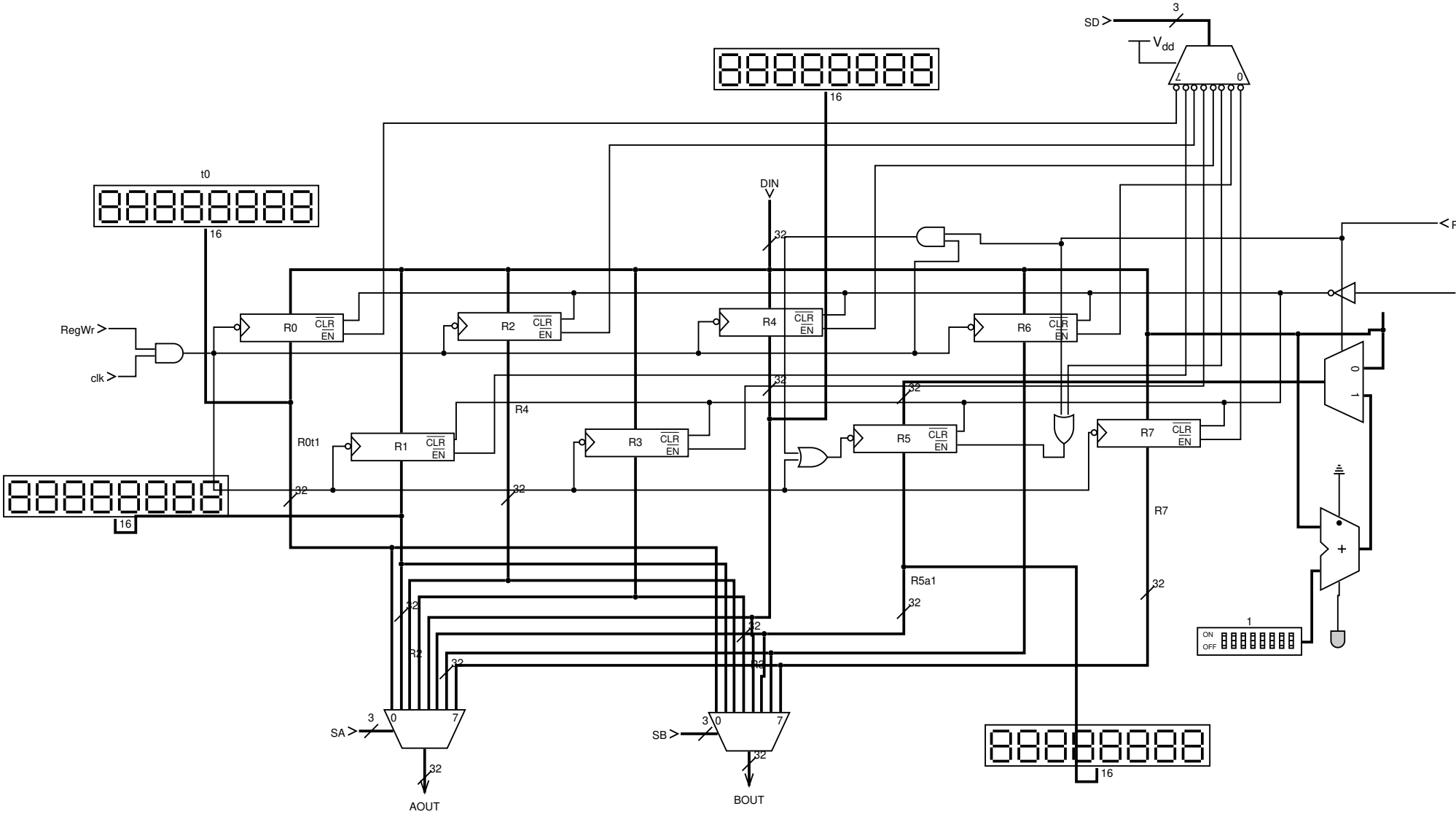
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	12 of 19
Date:	June 21, 2022	**Module:**	READ

TKGate



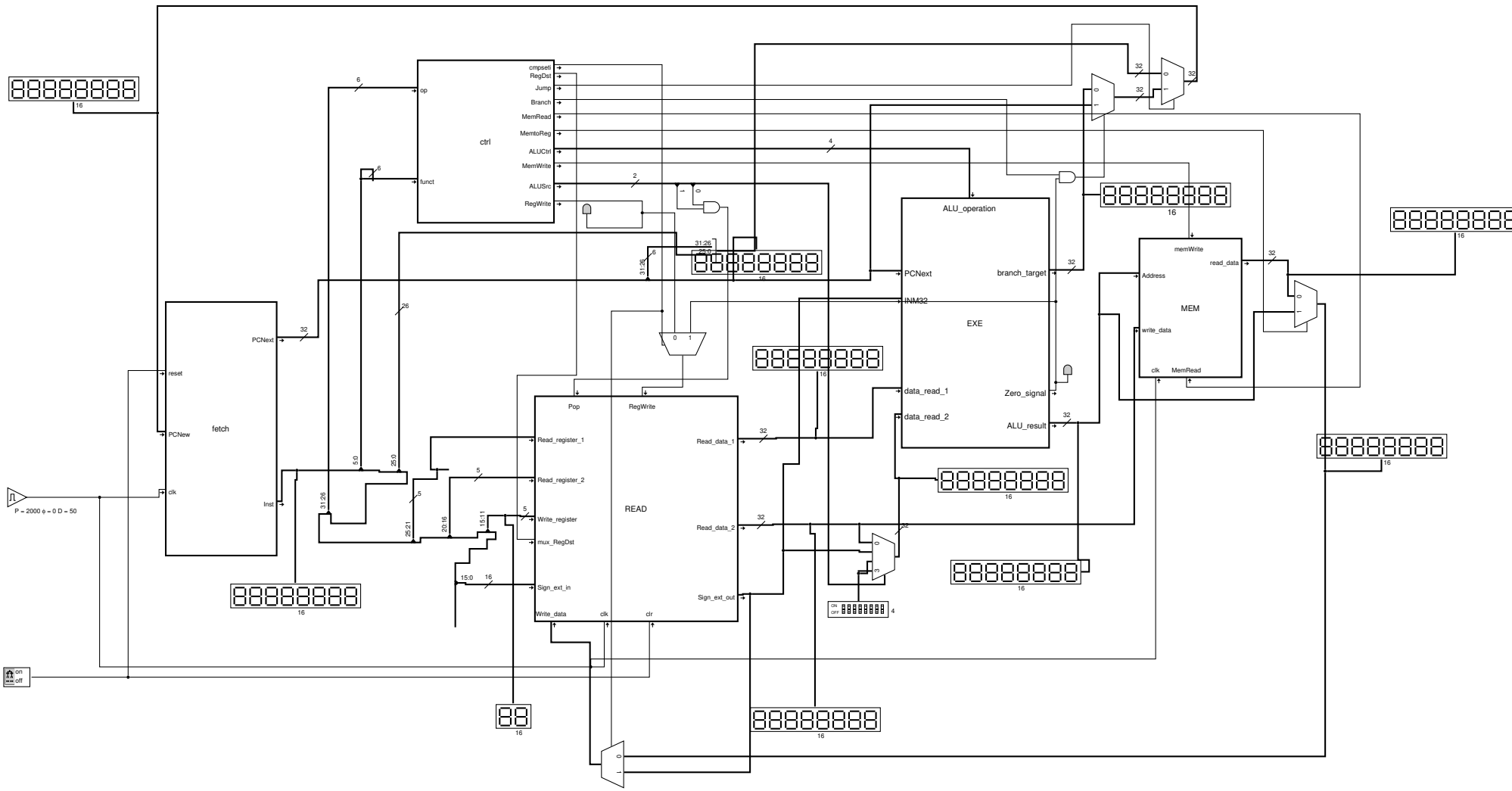
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	13 of 19
Date:	June 21, 2022	**Module:**	CSA16bits

TKGate



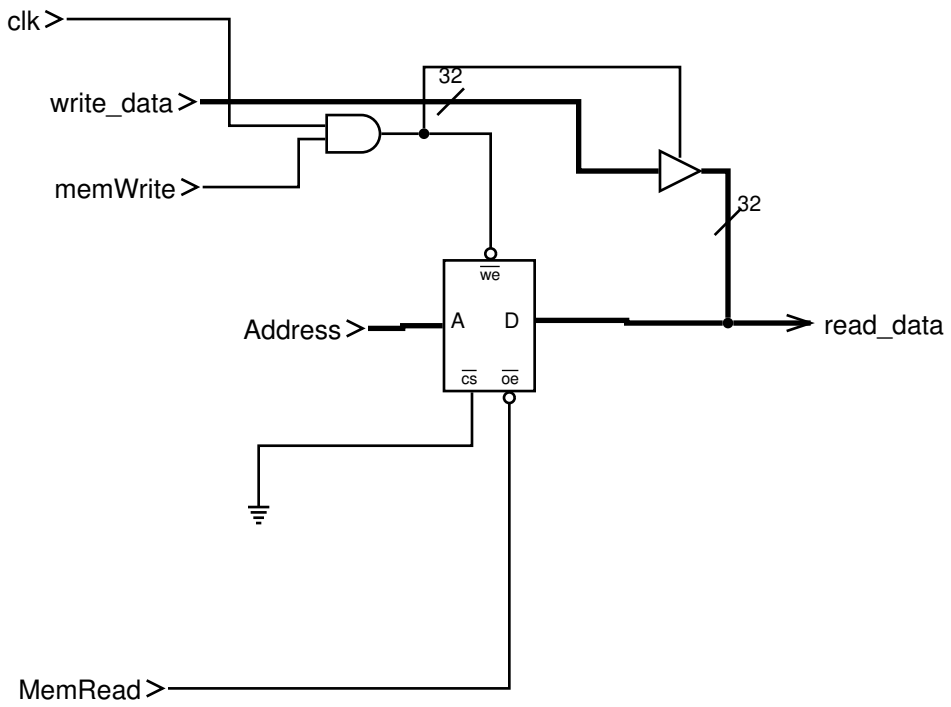
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	14 of 19
Date:	June 21, 2022	**Module:**	Regs8x324

TKGate



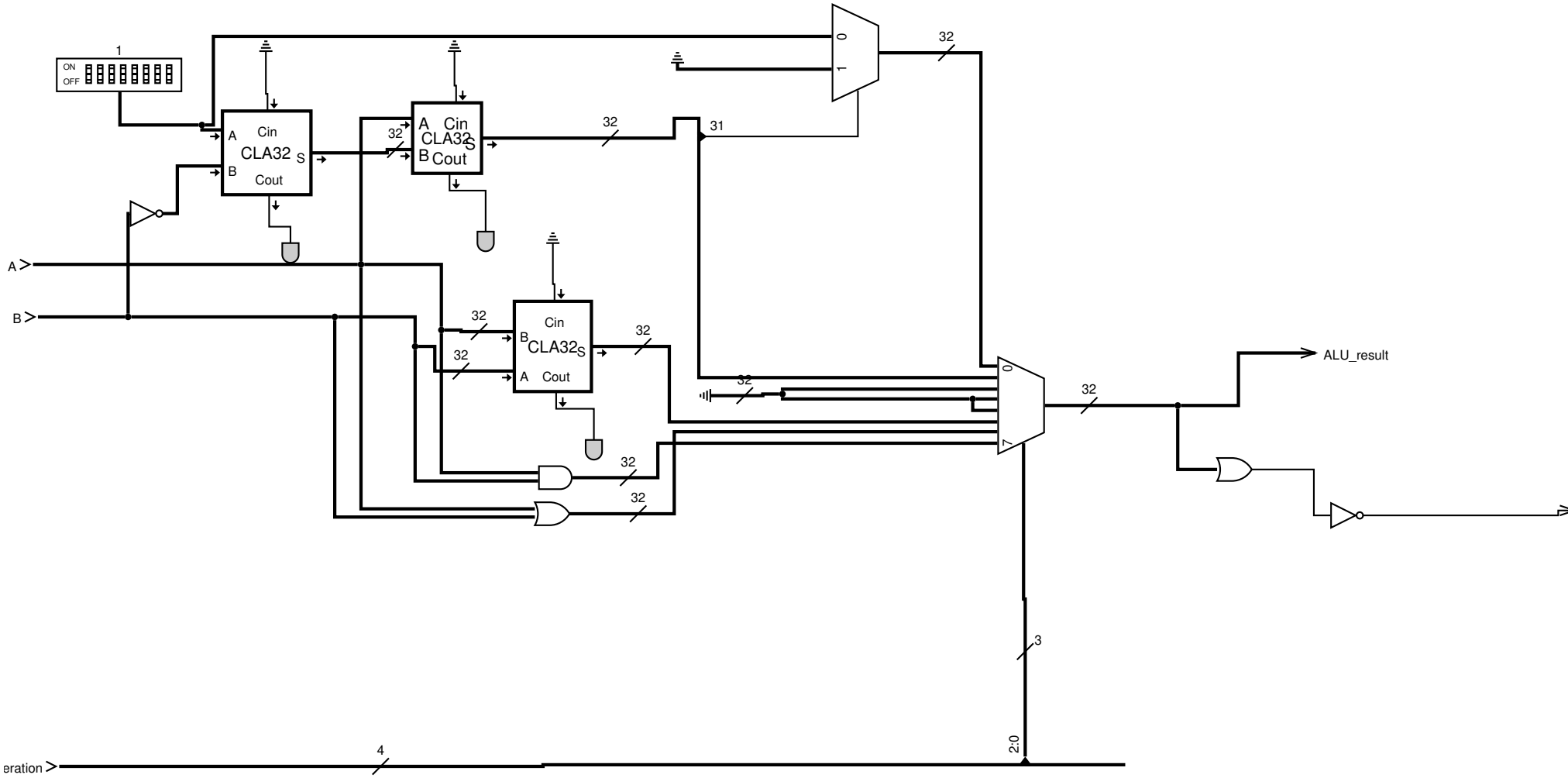
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	15 of 19
Date:	June 21, 2022	**Module:**	main

TKGate



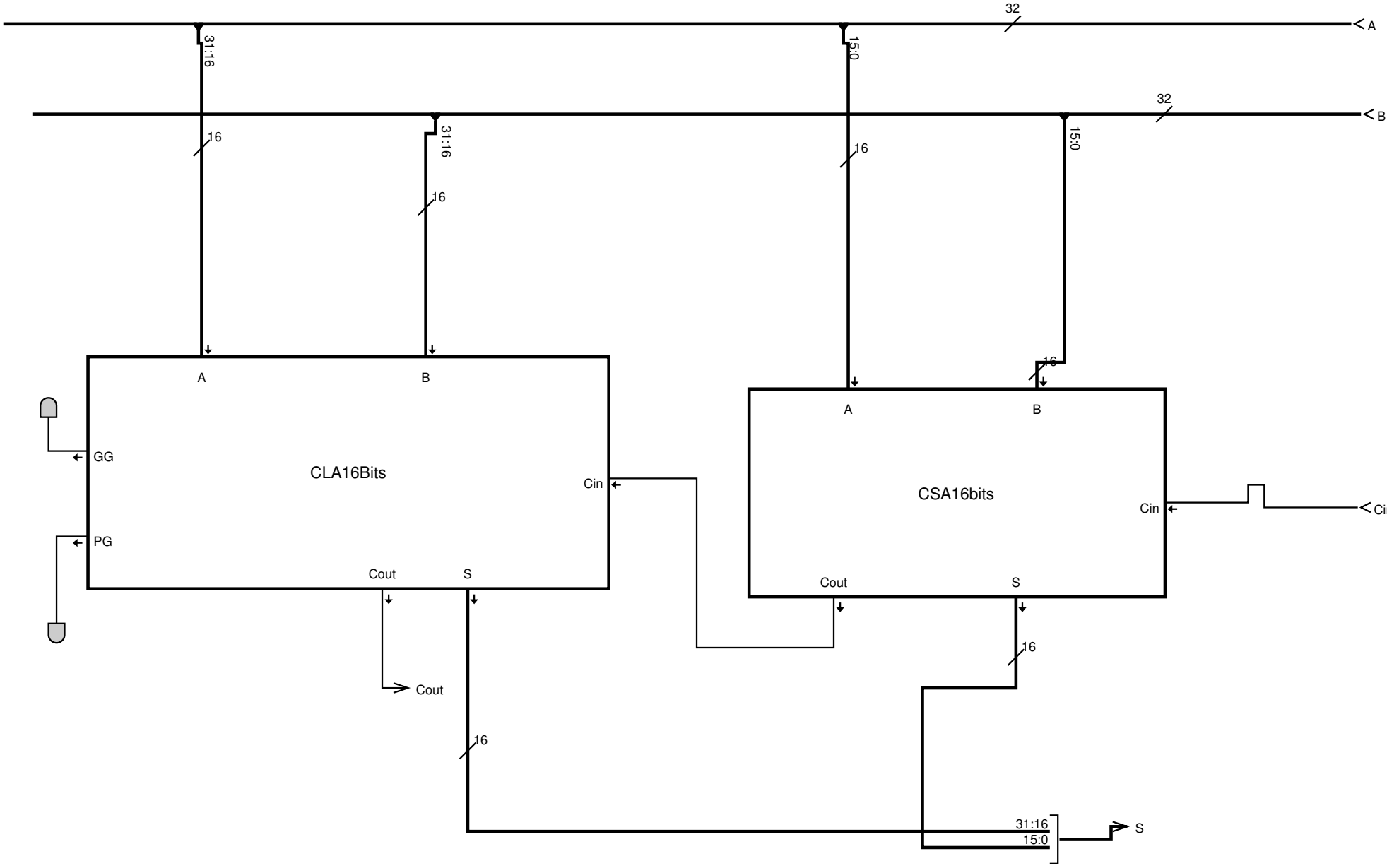
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	16 of 19
Date:	June 21, 2022	**Module:**	MEM

TKGate



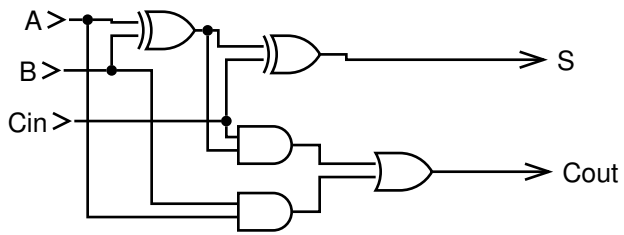
Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	17 of 19
Date:	June 21, 2022	**Module:**	ALU

TKGate

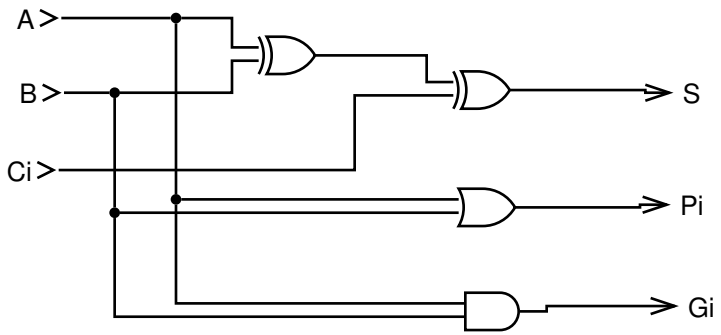


Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	18 of 19
Date:	June 21, 2022	**Module:**	CLA32

TKGate



FA



PFA

Designer:	milax,,,	**Document:**	TAREA6–2.v
Site:	Earth	**Page:**	19 of 19
Date:	June 21, 2022	**Module:**	FA, PFA

TKGate