Computer Architecture Unit 2

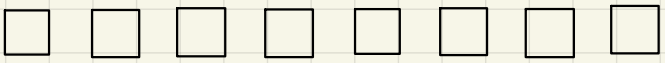
22/02/2022

BYTES (INTRODUCTION)

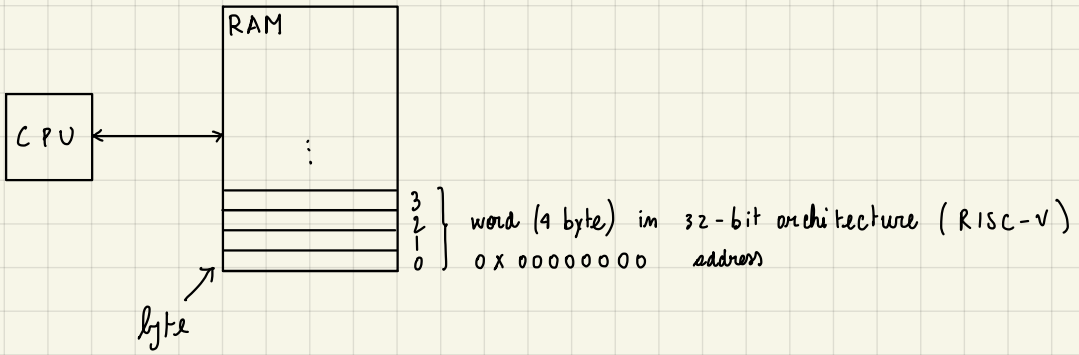
bit

(flip flop)

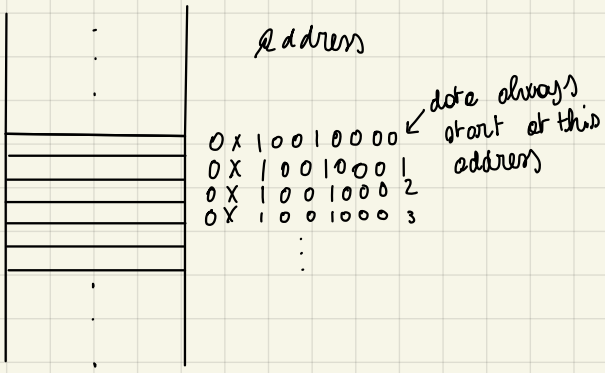
byte (8 bits)



Kilobyte : 10^3	bytes	Kibibyte	1024 bytes
Megabyte : 10^6	bytes	Mebibyte	1024 kibibytes
Gigabyte : 10^9	bytes	Gibibyte	1024 Mebibytes
Terabyte : 10^{12}	bytes	Tebibyte	1024 Gibibyte



STORING INTEGERS IN RAM

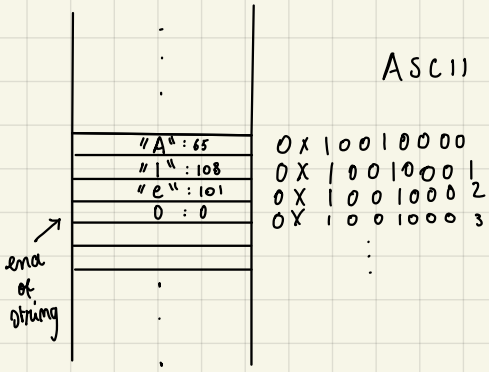


Big Endian: less significant ↓ most significant

Little Endian: most significant ↓ less significant

- 2-complement is used
- 4-bytes → 2^{32} numbers → $[-2^{32}, +2^{32}-1]$

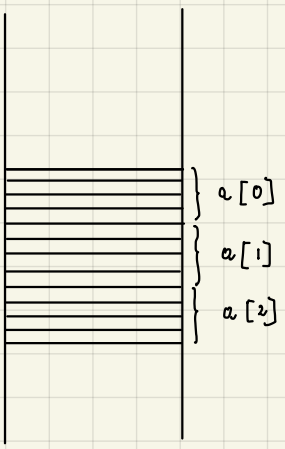
STORING STRINGS IN RAM



ASCII : 1 character → 1 byte

STORING ARRAYS IN RAM

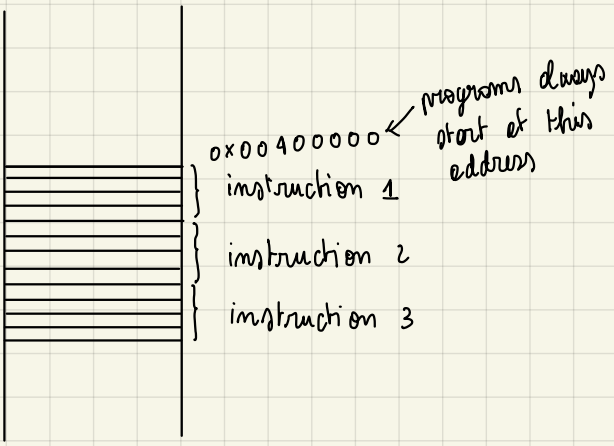
$a = [7, 2, -3]$

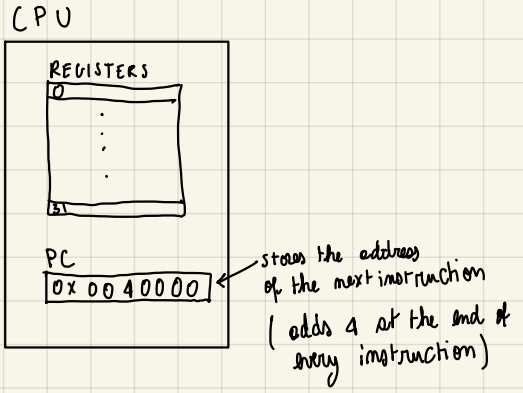


- we have to store also the lenght of the array

STORING PROGRAMS IN RAM

- In RISC-V every instruction is stored in one word

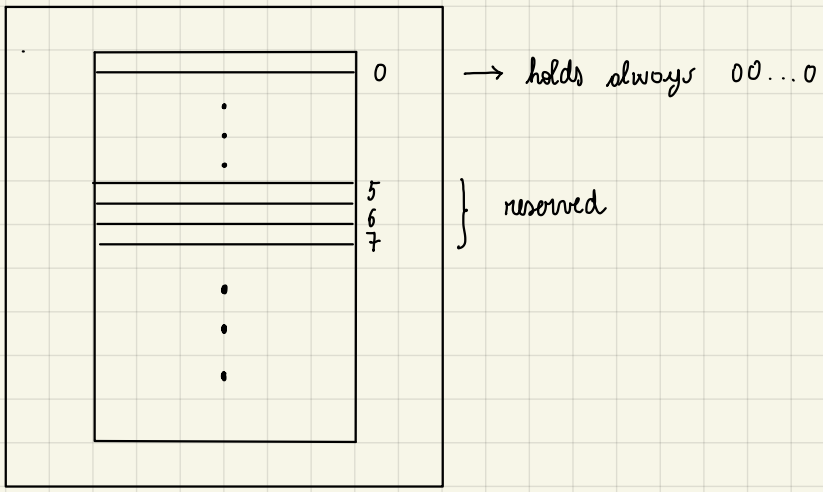




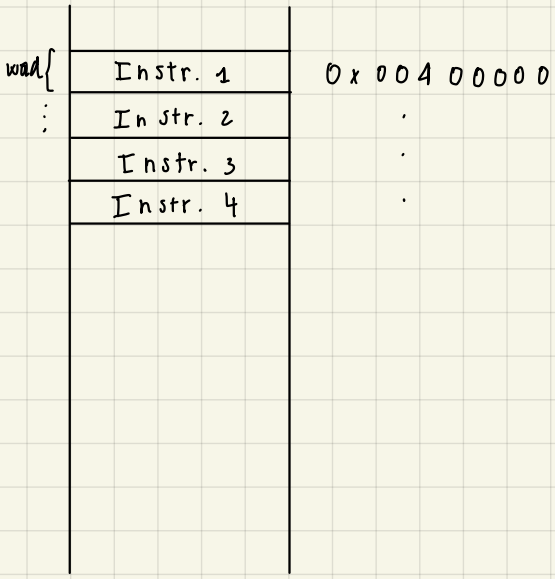
- Fetch and execute

25/02/2022

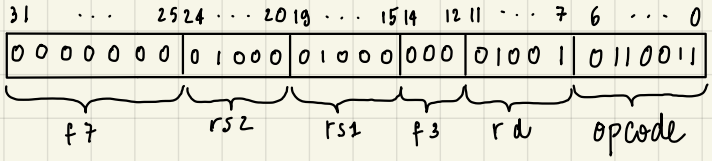
REGISTERS IN CPU



INSTRUCTIONS IN RISC-V ARCHITECTURE

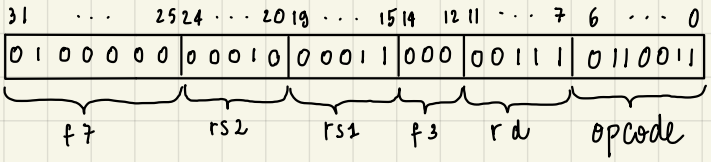


① Add 5, 8, 8 ⇒ R[5] ← R[8] + R[8]



32 bits

② Sub 7, 3, 2 ⇒ R[7] ← R[3] - R[2]



EX. 1

R[5] ← R[28] + R[29] + R[30]

Add 5, 28, 29

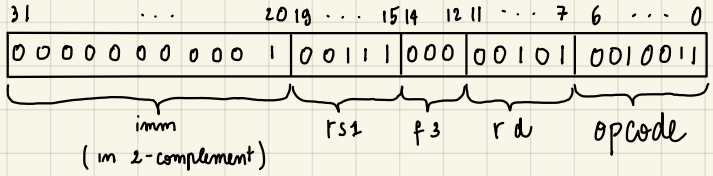
Add 5, 5, 30

EX. 2

R[5] ← R[7]

Add 5, 7, 0

③ Addi 7, 7, 1 => R[7] ← R[7] + 1



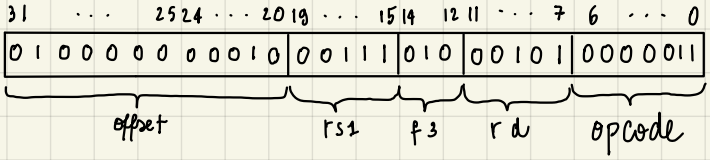
Ex.

R[5] ← 3

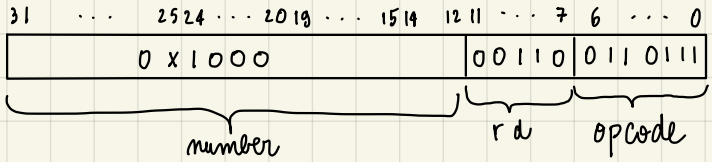
Addi 5, 0, 3

loads a word from the memory

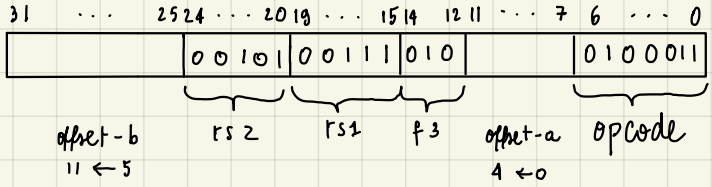
④ lw 5, offset(7) => R[5] ← MEM[R[7] + offset]



⑤ lui 6, 0x1000

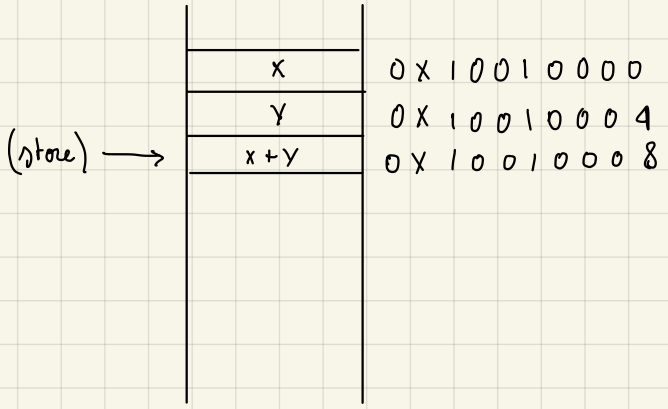


⑥ sw 5, offset(8) => MEM[R[8] + offset] ← R[5]



EXERCISE

R[5] ← x + y




```
lui 1, 0x10010
lw 5, 0 (1)
lw 6, 4 (1)
Add 5, 5, 6
sw 5, 8 (1)
addi 17, 0, 10   } to close the program
ecall            }
```

⑦ beq 5,6, offset => if R[5] = R[6] : PC ← PC + offset

EX.

```
add 5,0,0      0x00400000
add 6,0,0      0x00400004
beq 5,6,0      0x00400008
addi 5,5,1     0x0040000C
addi 6,6,1     0x00400010
```

⑧ bne 5,6, offset => if R[5] ≠ R[6] : PC ← PC + offset

```
    addi 5,0,0        bne 6,0,-8 loop
    addi 6,0,10
loop: add 5,5,6
    addi 6,6,-1
```

08/03/2022

REGISTERS IN THE CPU

0 : x0 zero registers

5 → 7 . x5 → x7 $t_0 \rightarrow t_2$ they usually store temporary values

10-11 : x10 - x11 $a_0 - a_1$

17 : x17, a_7

SYSTEM CALLS

ecall

$a_7 = 10 \Rightarrow$ exit()

$a_7 = 1 \Rightarrow$ print_int (a_0)

$a_7 = 4 \Rightarrow$ print_string (a_0)

↘ contains an address in memory where the string starts

LOGICAL INSTRUCTIONS

and x7, x5, x6 ⇒ x7 ← x5 AND x7

andi x7, x5, 0x01 x7 ← x5 AND 0x01

or . . .

ori . . .

xor . . .

xori

11/03/2022

ble / blt t_0, t_1, label ⇒ PC goes to label if $t_0 < / \leq t_1$

bge / bgt t_0, t_1, label ⇒ / / / / $t_0 > / \geq t_1$

FUNCTIONS

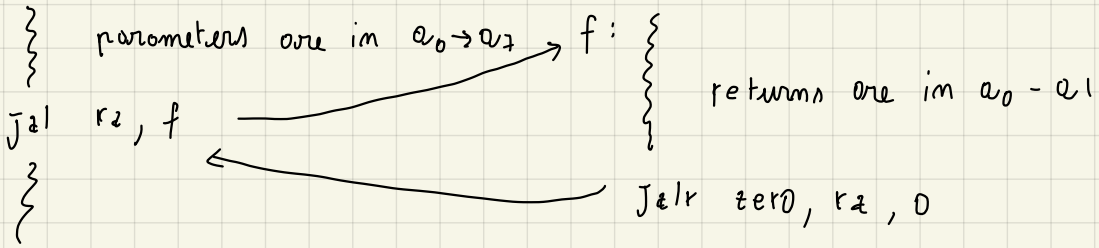
jal r2, offset ⇒ r2 ← PC+4

PC ← PC + offset

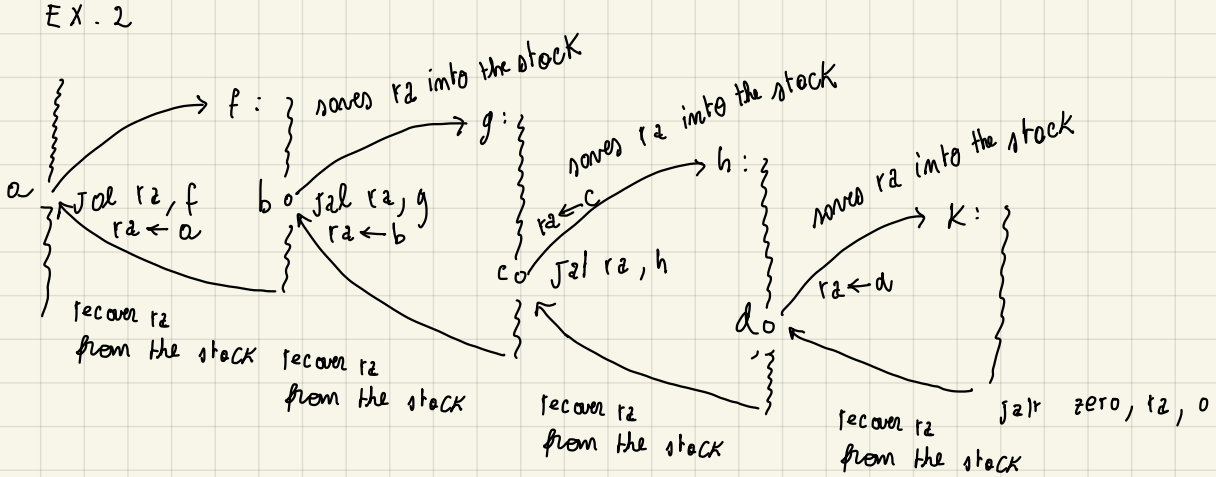
jalr x_1, r_2, offset ⇒ $x_1 \leftarrow pc + 4$

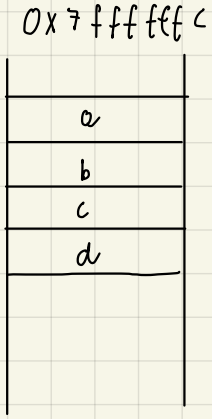
$pc \leftarrow r_2 + offset$

EX. 1



EX. 2





IMPLEMENTATION OF A STACK

EX. 1 ← stack pointer

```
addi  sp, sp, -4
sw    r2, 0(sp)
lw    r2, 0(sp)
addi  sp, sp, +4
```

EX. 2

```
addi  sp, sp, -8
sw    r2, 0(sp)
sw    a0, 4(sp)
lw    a0, 0(sp)
lw    a2, 4(sp)
addi  sp, sp, 8
```

FACTORIAL

```
f:    bne  a0, zero, rec
      li   a0, 1
      jalr zero, ra, 0
rec:  addi sp, sp, -8
      sw   ra, 0(sp)
      sw   a0, 4(sp)
      addi a0, a0, -1
      jal  ra, f
      lw   t0, 4(sp)
      mul  a0, a0, t0
      lw   ra, 0(sp)
      addi sp, sp, 8
      jalr zero, ra, 0
```

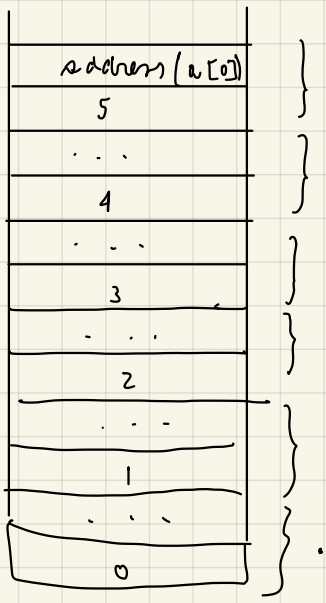
20

24

28

32

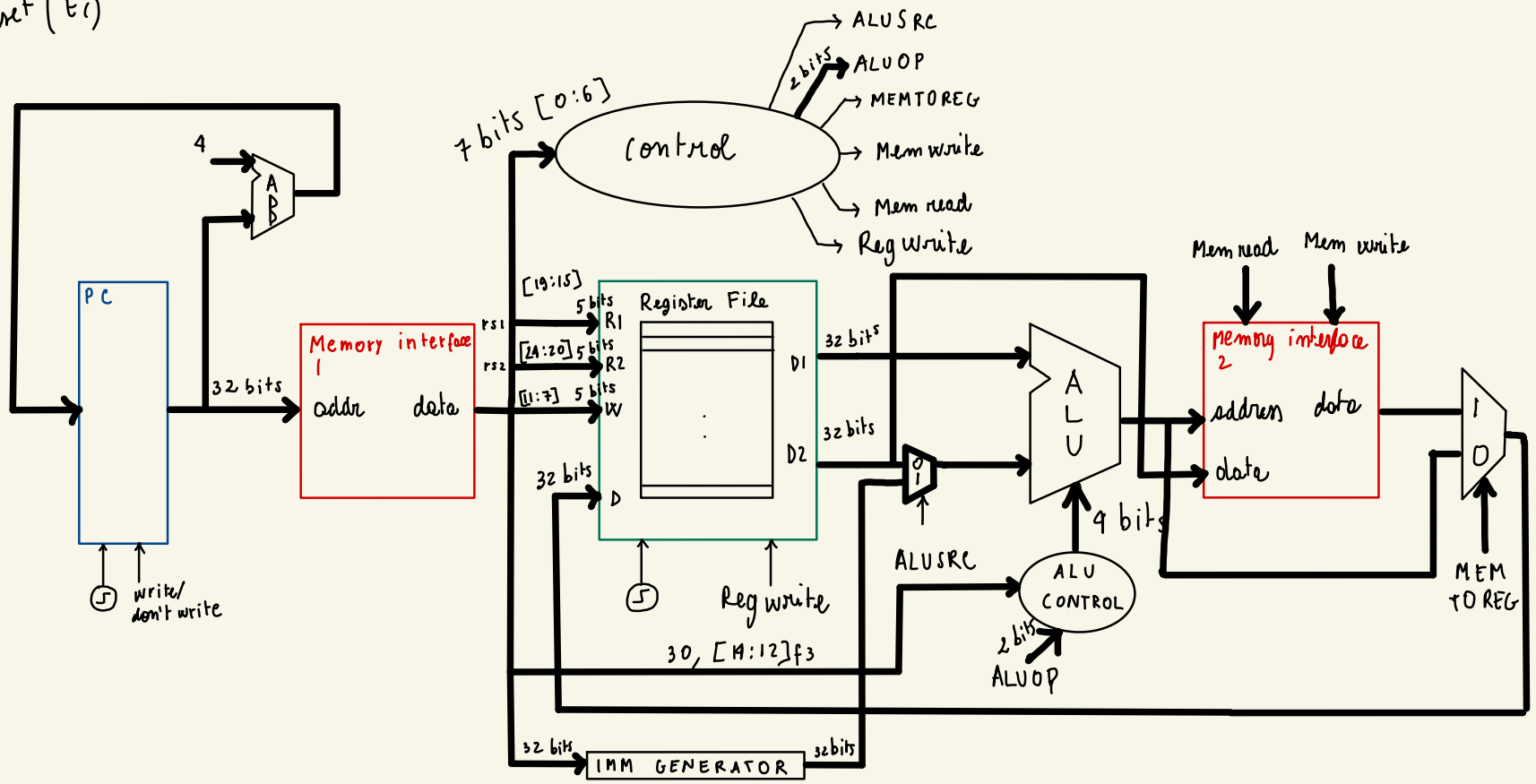
t0



16/03/2022

add t_0, t_1, t_2
sub t_0, t_1, t_2
addi t_0, t_1, imm
lw t_0, offset(t_1)
sw t_0, offset(t_1)

SINGLE CYCLE ARCHITECTURE

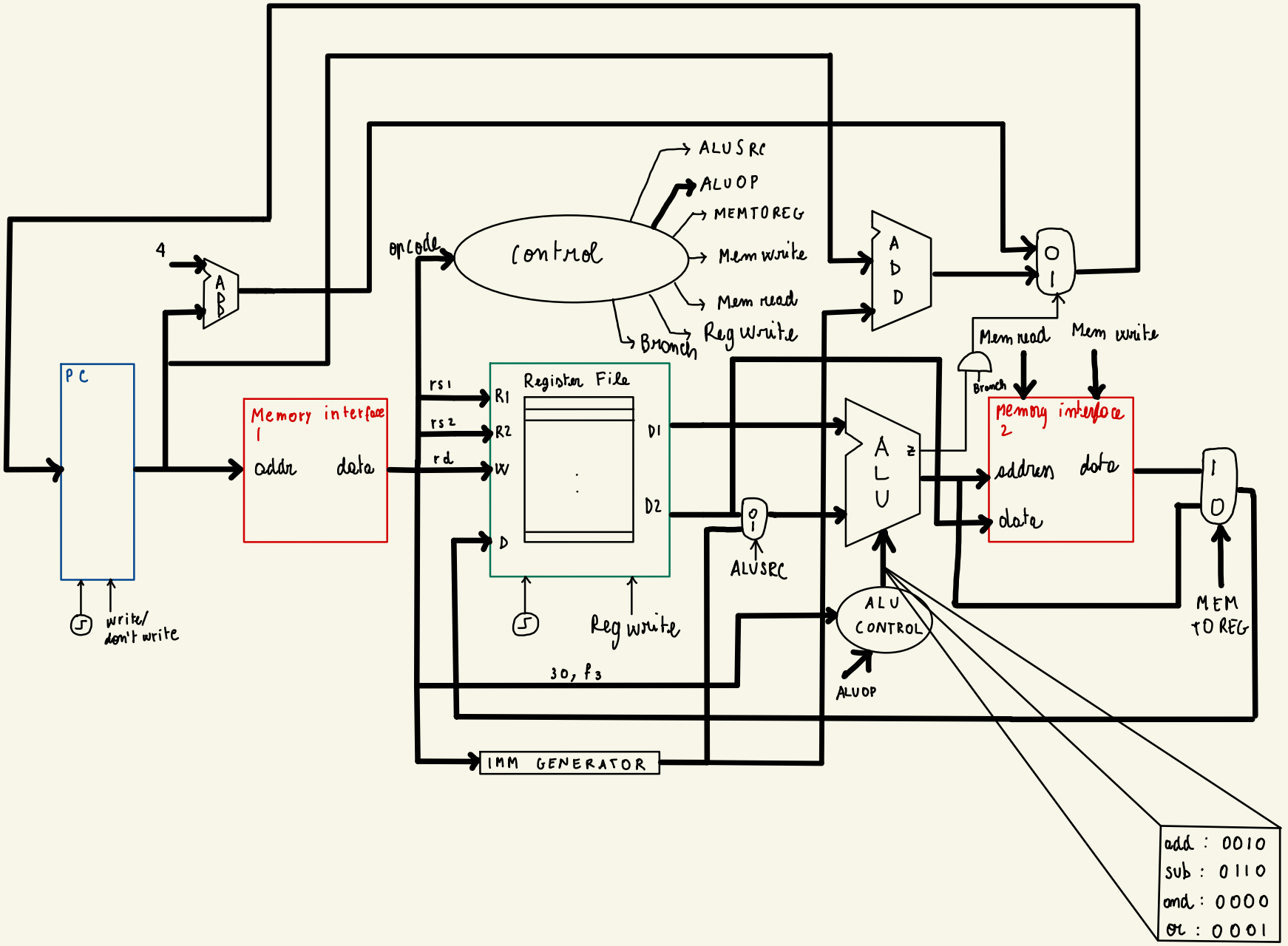


23/03/2021

add t_0, t_1, t_2
sub t_0, t_1, t_2
addi t_0, t_1, imm
lw t_0, offset(t_1)
sw t_0, offset(t_1)
beq t_1, r_2, address

beq x7, x8, offset

SINGLE CYCLE ARCHITECTURE



Ex 1

t_0 : 0x10010000

t_1 : 0xFFFFFFFE

t_2 : 0x00000005

t_3 : 0xFFFFFFFF

Ex. 2

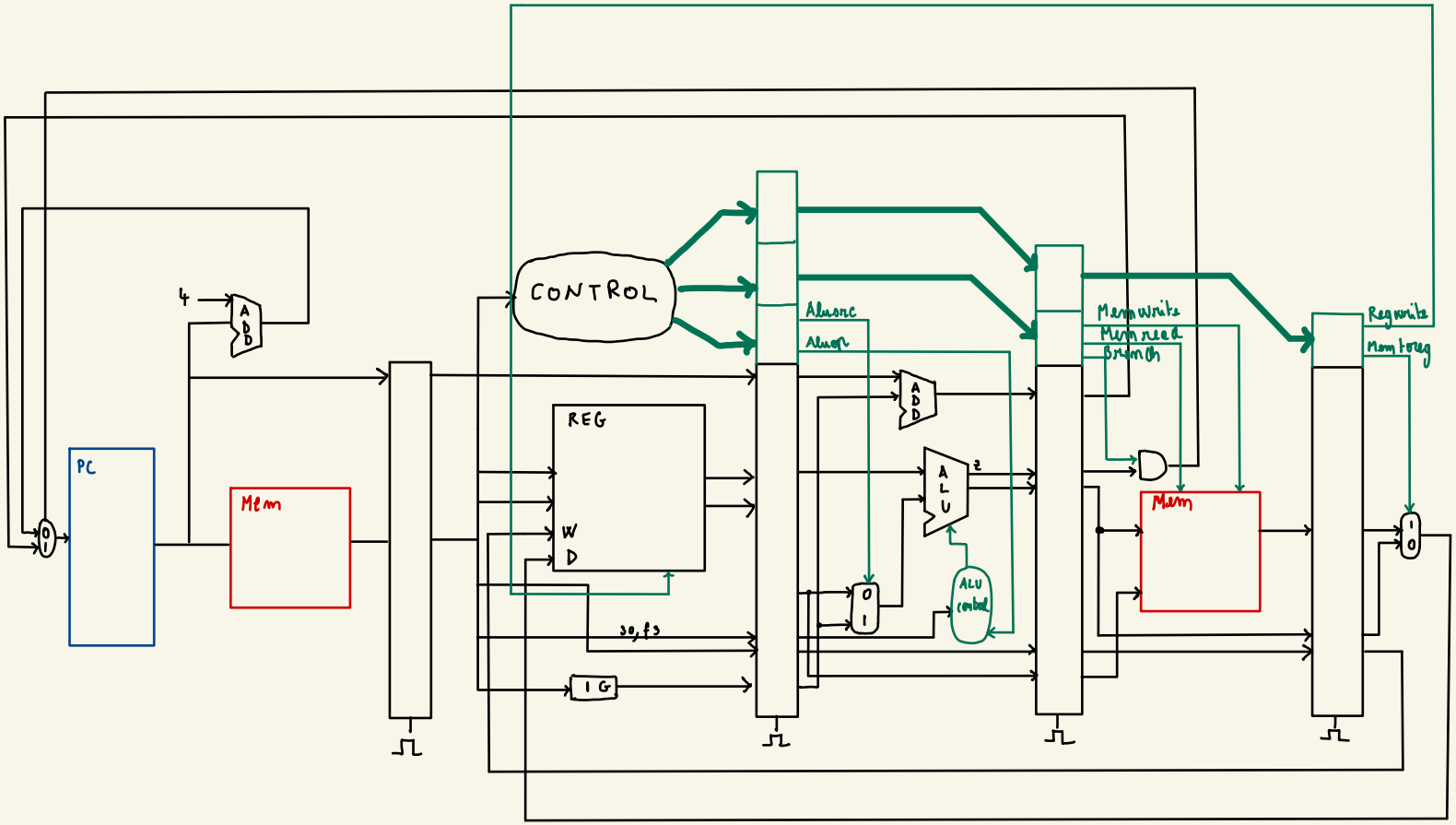
```
la  t0, x   =   lui  t0, 0x10010
                addi t0, t0, 24

lw  t0, x   =   lui  t0, 0x10010
                lw   t0, 24(t0)
```

```
.data
    .asciz "Alessandro"
    .align 2
    .word 5, 7, 8
x:  .word 6
```

13/04/2022

PARALLELISM

PIPELINING



DATA HAZARDS

EX. 1

PROBLEM

add (x2), x_3, x_5

sub x6, x7, (x2)

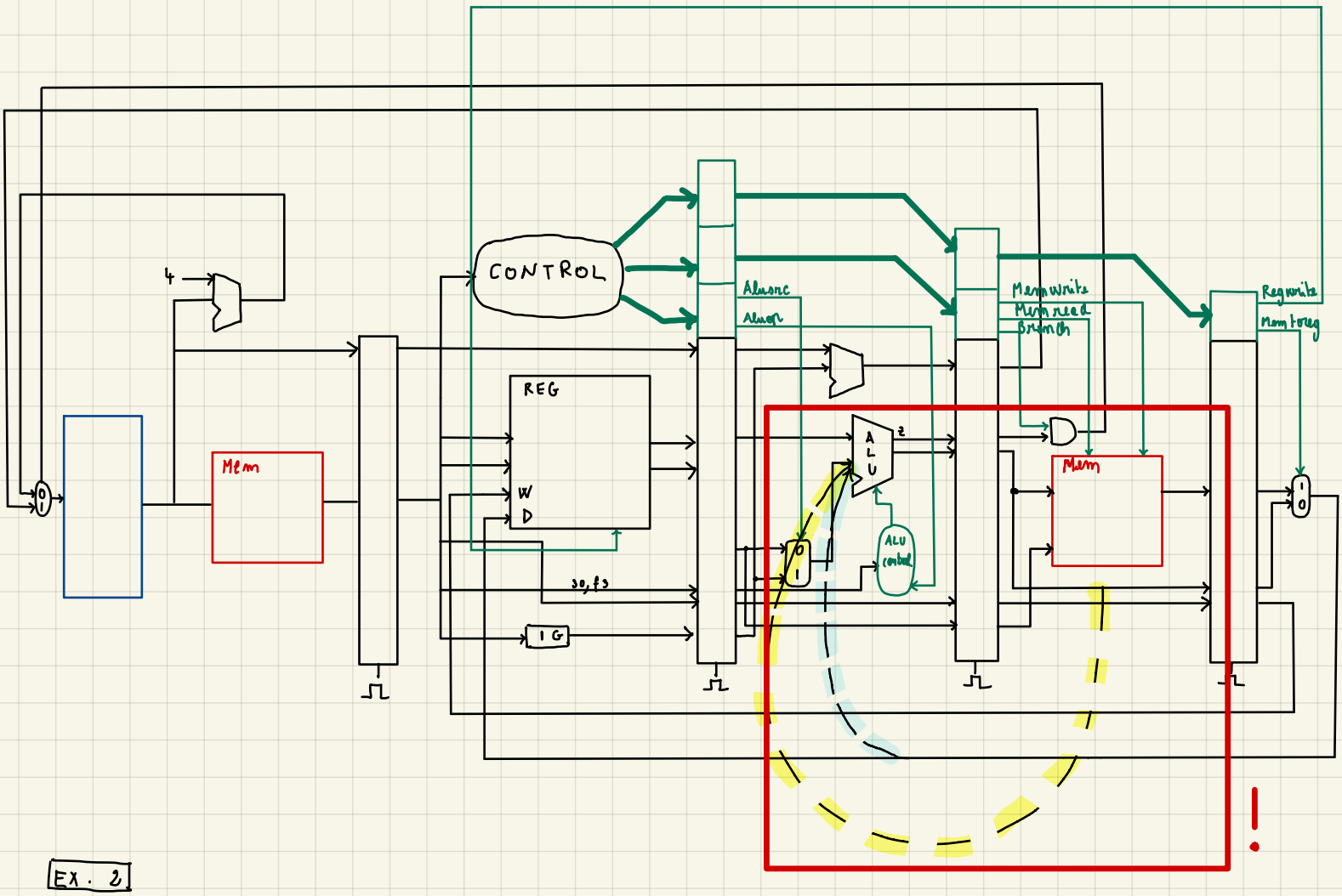
the second instruction needs the first one to complete first!

1st solution (less efficient)

Put some "bubbles" between the two istruction in order to let the first instruction complete.

2nd solution (Forwarding)

Enable the architecture to detect when a situation of that type happens and put the "future" value of x2 in the place of the "old one"



EX. 2

beq x1, x2, label

...

...

PROBLEM

The architecture will continue to execute the instructions after the branch even if it should not

1st solution (loop unrolling)

reduce the number of branches

EX.

```
for i in range(1, 1000)        (compiler) →   for i in range(1, 100):
    do_stuff()                                    do_stuff
                                                  do_stuff
                                                  :
                                                  do_stuff
```

2nd solution

Execute both the next instruction and the instruction at label address and discard the wrong one.

3rd solution

Predict before wether the branch will happen or not.

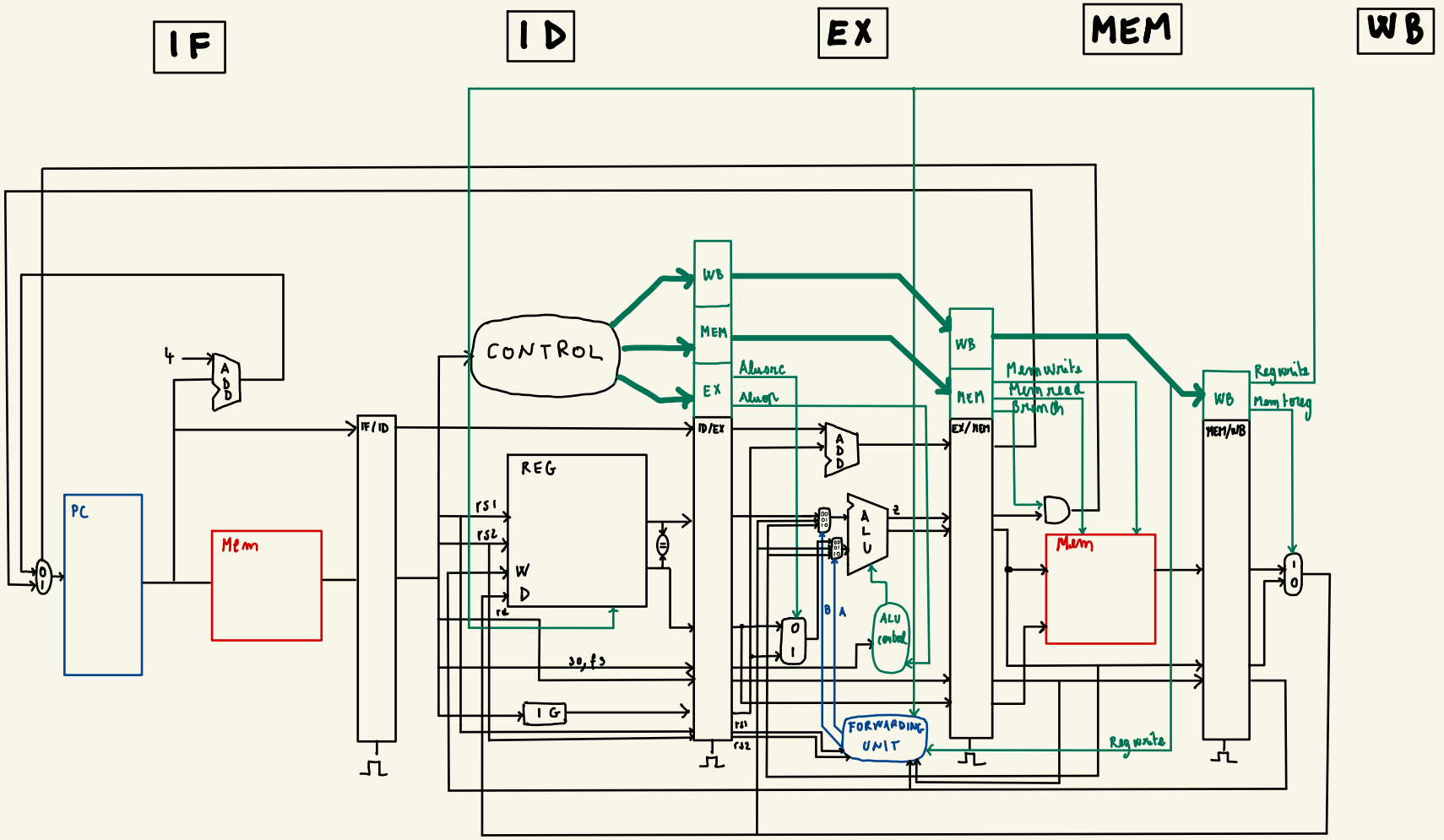
20/04/2022

FORWARDING

EX. 1
```
add  x1, x1, x2
add  x2, x2, x1
```

EX. 2
```
add  x1, x1, x2
addi x7, x7, 1
sub  x2, x1, x4
```

EX. 3
```
add  x1, x1, x2
add  x1, x1, x3
add  x1, x1, x4
```



Forwarding unit logic

if (EX/MEM . RegWrite) and (EX.MEM . rd ≠ 0) and (ID/EX . rs1 = EX.MEM . rd) :

Forward A = 10

if (EX/MEM . RegWrite) and (EX.MEM . rd ≠ 0) and (ID/EX . rs2 = EX.MEM . rd) :

Forward B = 10

if (MEM/WB . RegWrite) and (MEM/WB . rd ≠ 0) and (ID/EX . rs1 = MEM/WB . rd) and not((EX/MEM . RegWrite) and (EX.MEM . rd ≠ 0) and (ID/EX . rs1 = EX.MEM . rd)) :

Forward A = 01

if (MEM/WB . RegWrite) and (MEM/WB . rd ≠ 0) and (ID/EX . rs2 = MEM/WB . rd) and not((EX/MEM . RegWrite) and (EX.MEM . rd ≠ 0) and (ID/EX . rs2 = EX.MEM . rd)) .

Forward B = 01

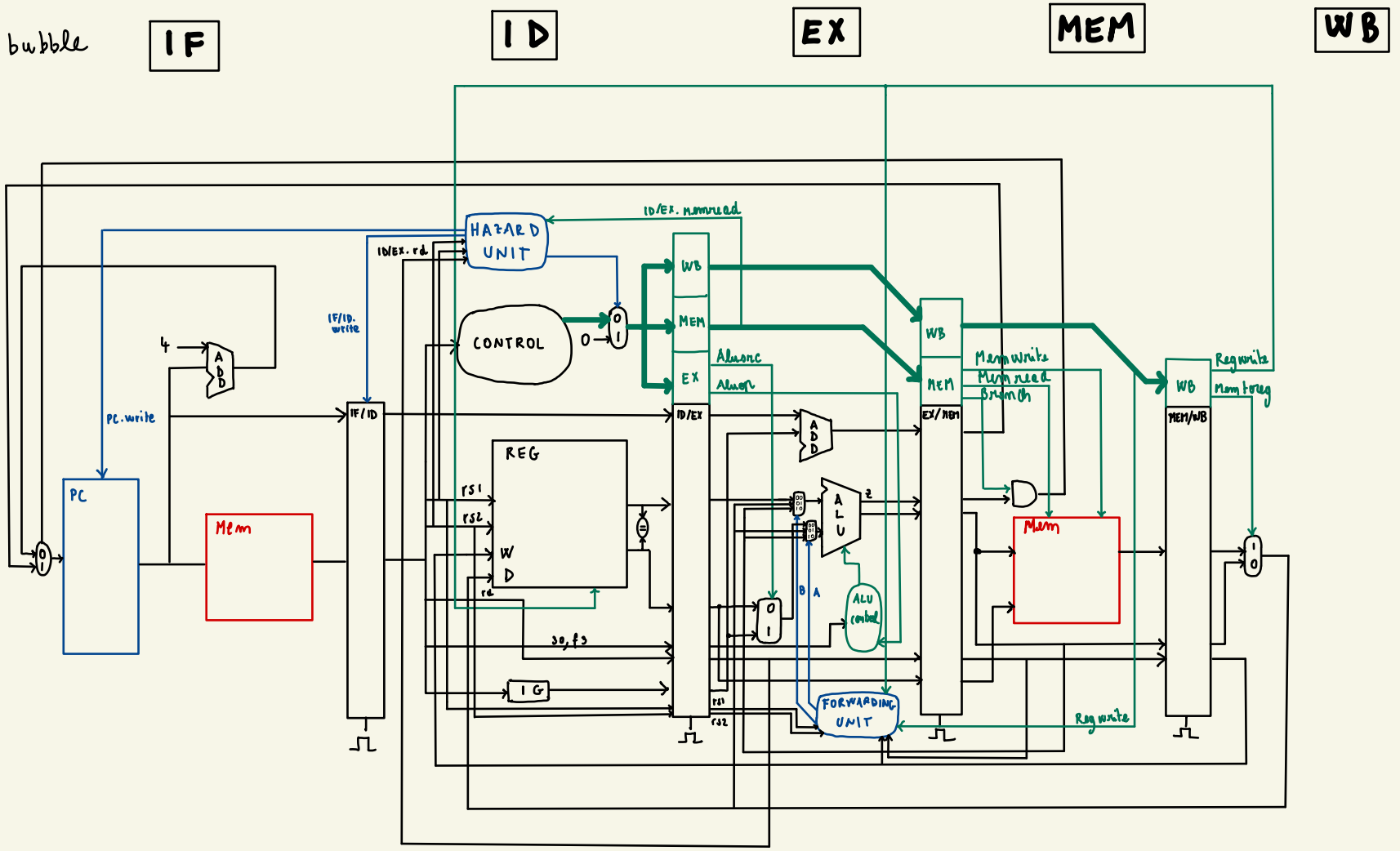
22/04/2022

STALLING

EX.

lw x6, 0(x7) ↙ bubble

add x8, x8, x6



HAZARD UNIT LOGIC

if ID/EX. Memread and (ID/EX.rd = IF/ID.rs1 or ID/EX.rd = IF/ID.rs2) :

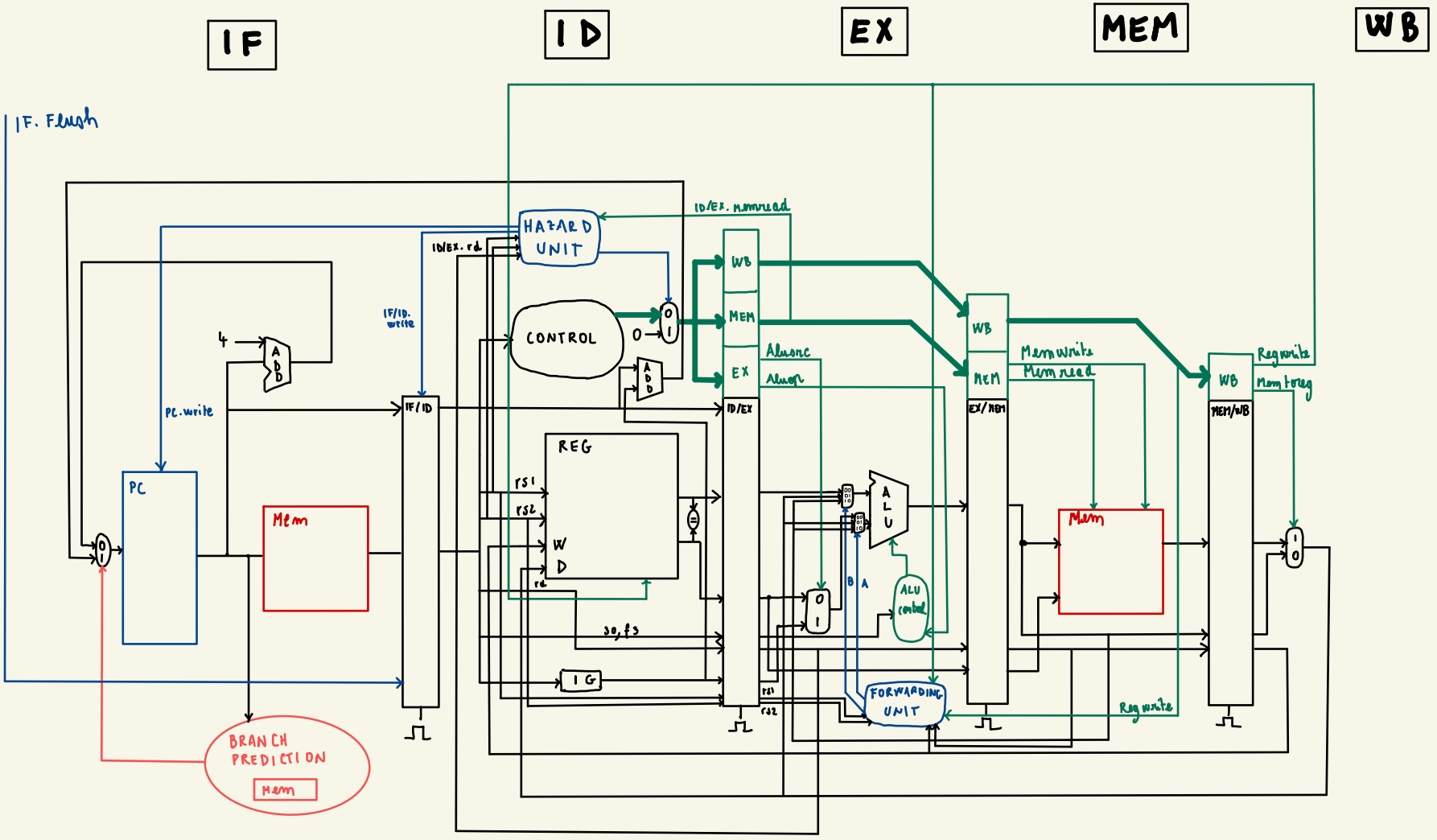
stall the pipeline

BRANCH PREDICTION

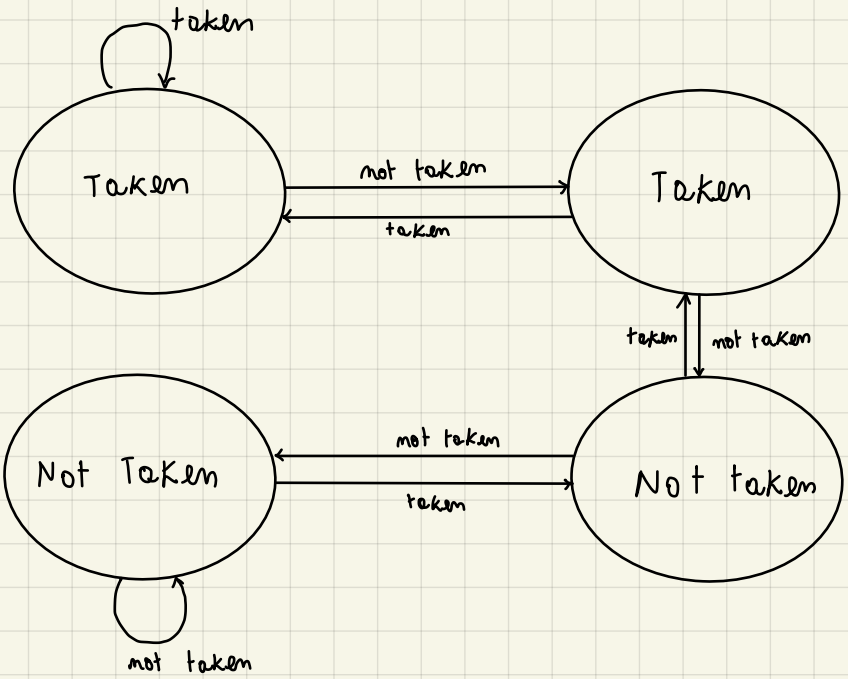
EX.

for i = 1 to 1000 000

:



Branch prediction logic

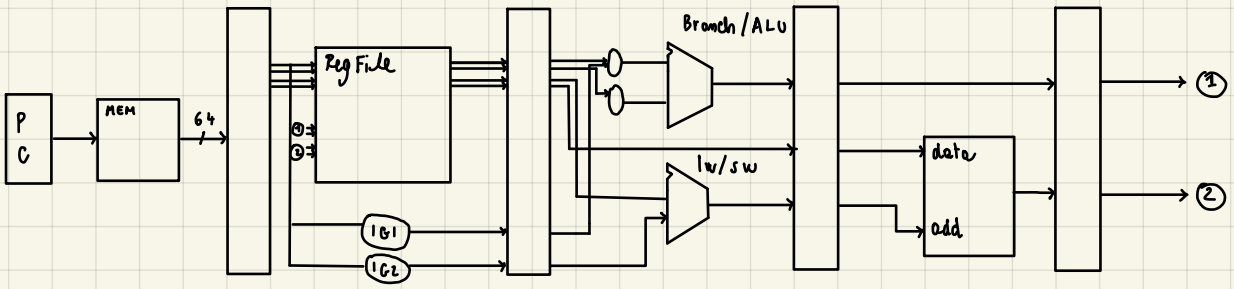


MULTIPLE ISSUE ARCHITECTURE

pocket

branch/ALU	lw/sw

Static architecture



EXERCISE

```
.data
        x:  (array 1)
        y:  (array 2)
        z:  (array 3)
        n:  len
.text
        la  t0, x
        la  t1, y
        la  t2, z
        lw  a1, n
loop:   lw  s0, 0(t0)
        lw  s1, 0(t1)
        add s1, s0, s1
        sw  s1, 0(t2)
        addi t0, t0, 4
        addi t1, t1, 4
        addi t2, t2, 4
        addi a1, a1, -1
        bne  a1, zero, loop
```

Multiple issue version (loop only)

```
addi t0, t0, 4 | lw  s0, 0(t0)
addi t1, t1, 4 | lw  s1, 0(t1)
addi a1, a1, -1 |
add  s1, s0, s1 |
addi t2, t2, 4 | sw  s1, 0(t2)
bne  a1, zero, loop |
```

Multiple issue version loop unrolling (loop only)

```
addi a1, a1, -2 | lw s0, 0(t0)
addi t2, t2, 8  | lw s1, 0(t1)
addi t0, t0, 8  | lw s2, 4(t0)
add  t3, s1, s2 | lw s3, 4(t1)
addi t1, t1, 8  | sw t3, -8(t2)
add  t4, s2, s3
bne  a1, zero cicly | sw t4, -4(t2)
```

Performance for arrays of 10^6 items

① Performance of single clock cycle 200 MHz

clocks per loop

$$\frac{9 \cdot 10^6}{200 \cdot 10^6 \frac{1}{s}} = 0,045\ s = 45\ ms$$

② Pipeline 1 GHZ
no branch prediction

$$\frac{11 \cdot 10^6}{1 \cdot 10^9 \frac{1}{s}} = 11\ 10^{-3}\ s = 11\ ms$$

③ Pipeline 1 GHz
branch prediction
code reordering

$$\frac{9 \cdot 10^6}{1 \cdot 10^9 \frac{1}{s}} = 9\ ms$$

④ Multiple issue arch branch prediction 1 GHz

$6 \cdot 10^6\ ns = 6\ ms$

⑤ Multiple issue arch branch prediction loop unrolling 1 GHz

$6 \cdot \frac{10^6}{2}\ ns = 3,5\ ms$

27/04/2022

MEMORY

LOCALITY

- spatial locality
- temporal locality

EX.

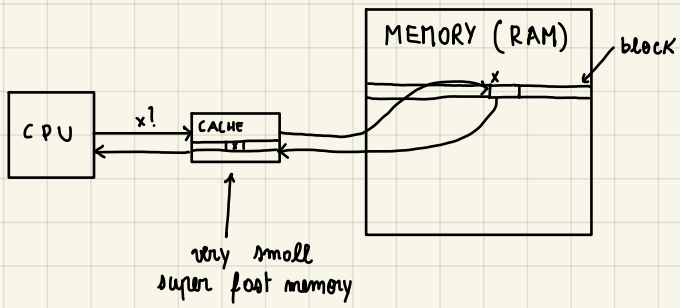
```
m := a[0]
for i := 1 to 1000 000 :
    if a[i] < m :
        m := a[i]
```

temporal locality

spatial locality

CACHE MEMORY

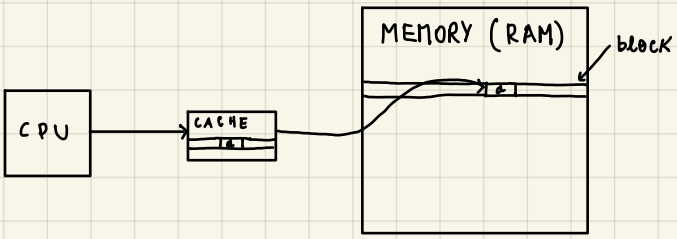
Read



(i) the value asked by the CPU is already in the cache → hit → faster access

(ii) Else → miss → the value is saved in the cache (together with his block) for future access

Write

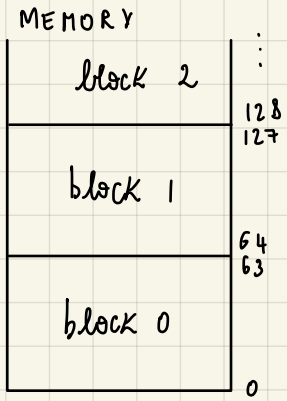


(i) the value is saved in the cache

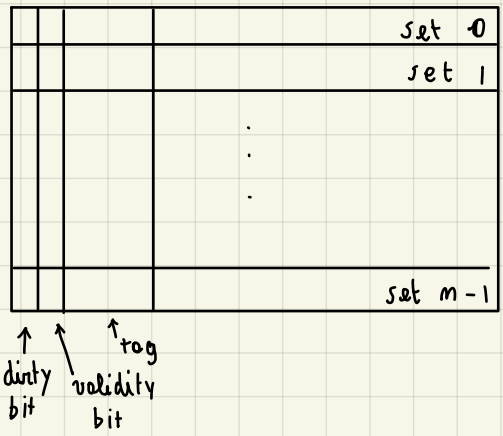
(ii) If the cache needs to be overwritten the value in there saved in the memory

1 block is 64 bytes

There are 2^k sets in the cache



CACHE

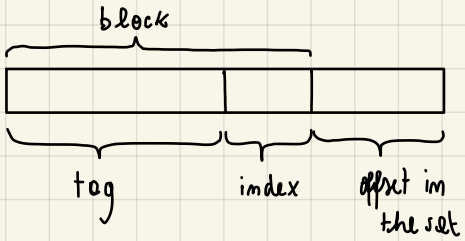


x = address

block = [x / 64] = x[6:]

set = block % msets = block[:4]

tag = [block / msets]



EXERCISE 4

- cache with 4 cache
- block size of 32 bytes

1	2	3	4	5	6	7	8
M	H	M	M	M	H	H	

access chronology: 36, 48, 0, 100, 164, 172, 96, 92

1) Miss → block 1 goes in set 1 of the cache

2) Hit (block 1 already in the cache)

3) Miss → block 0 goes into set 0

4) Miss → block 3 goes into set 3

5) Miss → block 5 goes into set 1

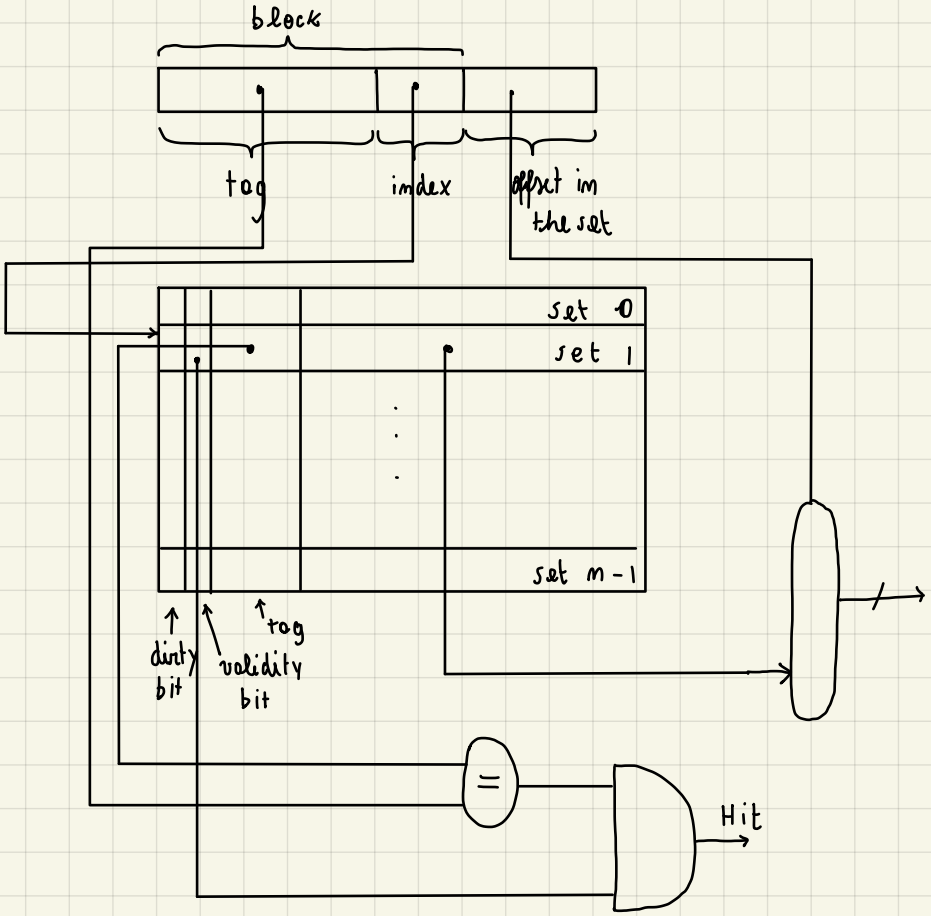
6) Hit (block 5 already in the cache)

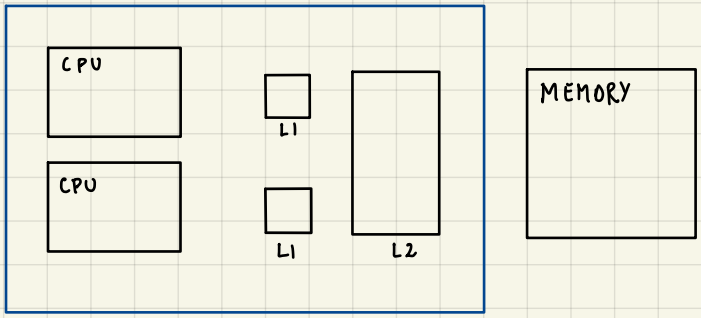
7) Hit (block 3 already in the cache)

8) Miss → block 2 goes into set 2

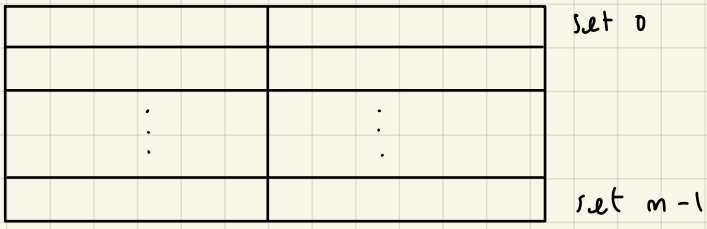
29/04/2022

CACHE





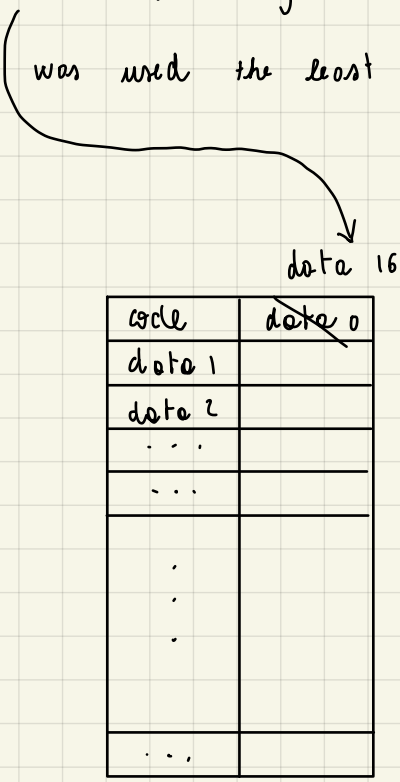
2-WAY ASSOCIATIVE CACHE



Ex.

```
.text
  {
ciclo: lw   t2, 0(t0)
       add  t3, t3, t2
       addi t0, t0, 4
       addi t1, t1, -1
       bne  t1, zero, ciclo
```

- block size = 64 bytes
- two-way associative cache with 16 sets
- LRU (least recently used): the block which was used the least recently is discarded



Analysis (two-way cache)

		ACCESSES	MISS
iteration	0	6	1
iteration	1	6	0
⋮			
iteration	16	6	1
iteration	17	6	0
⋮			
iteration	31	6	0
iteration	32	6	1

Miss rate in the first 16 iterations = $\frac{1}{16 \cdot 6} \simeq 1\%$

(it is the same for the others)

Analysis (one-way cache)

		ACCESSES	MISS
iteration	0	6	2
iteration	1	6	2
⋮		⋮	⋮
iteration	15	6	2
iteration	16	6	1

iteration	17	6	0
⋮		⋮	⋮
iteration	31	6	0
iteration	32		
⋮			

$$\text{miss rate} = \frac{32 + 31(1)}{16 \cdot 6 + 31\,(16 \cdot 6)} = 2{,}4\,\%$$

04/05/2022

.data

x: .word	____	0x10010000	set 0
y: .word	____		set 0
z: .word	____		set 0
n: .word	4096		

.text

M	lo,	t_0, x	0x00400000
H	lo	t_1, y	
H	lo,	t_2, z	
H	lw	t_3, n	

block: 0x00010000 → set 0

block : 64
set : 32
one-way associative
write-back

ciclo:	M	lw	t_4, 0(t_0)	M
	M	lw	t_5, 0(t_1)	M
	M	add	t_6, t_4, t_5	
	H	sw	t_6, 0(t_2)	M
	M	addi	t_0, t_0, 4	
	H	/	t_1 t_1 /	
	H	/	t_2 t_2 /	
	H	/	t_3 t_3 -1	
	M	bne	t_3, zero, ciclo	→ set 1

i 1

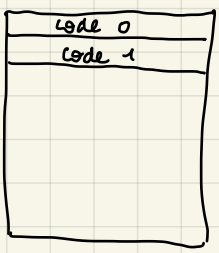
i 0	8	12
i 1	6	12
⋮		
i 15	6	12
i 16	4	12
⋮		
i 31	4	12
i 32	3	12
i 33	3	12
⋮		

i2) H M M M M H M M M H H H

i16) H M H M H H M H H H M

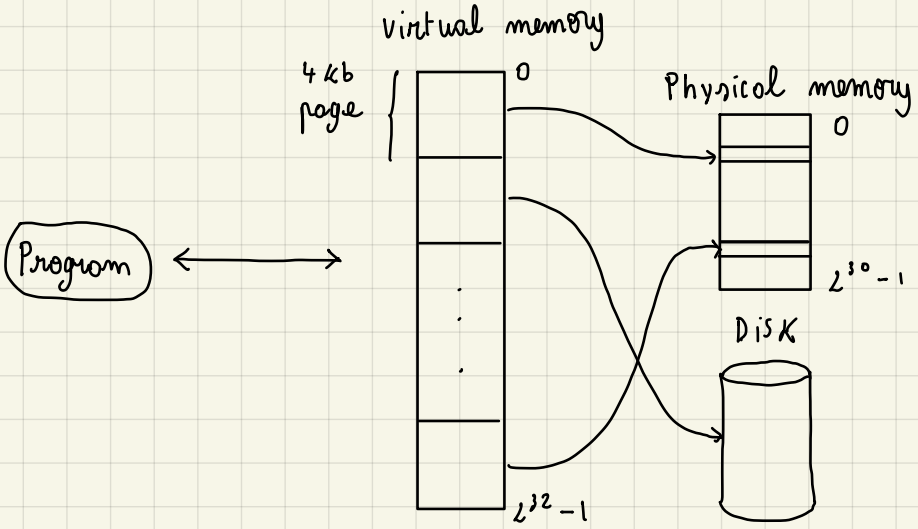
i32) H M H M H H M H H H H

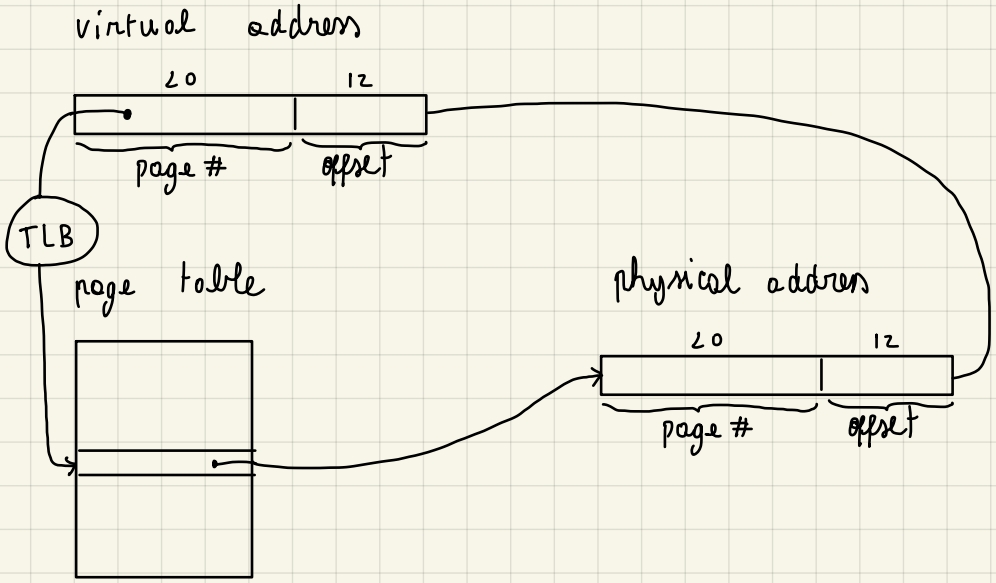
$$\text{miss rate} = \frac{8 + 15 \cdot 6 + 4 \cdot 16 + 3 \cdot 16 \cdot 30}{12 \cdot 16 + 12 \cdot 16 + 12 \cdot 16 \cdot 30} = 26\,\%$$



11/05/2022

VIRTUAL MEMORY



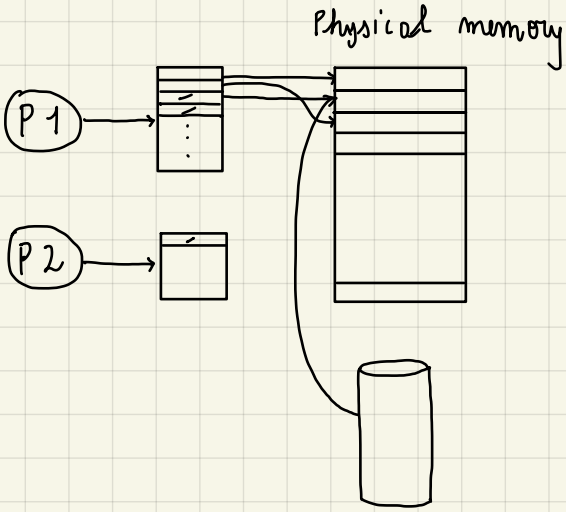


TLB: translation look-aside buffer

↓

small cache for the page table

1 page = 4096 bytes



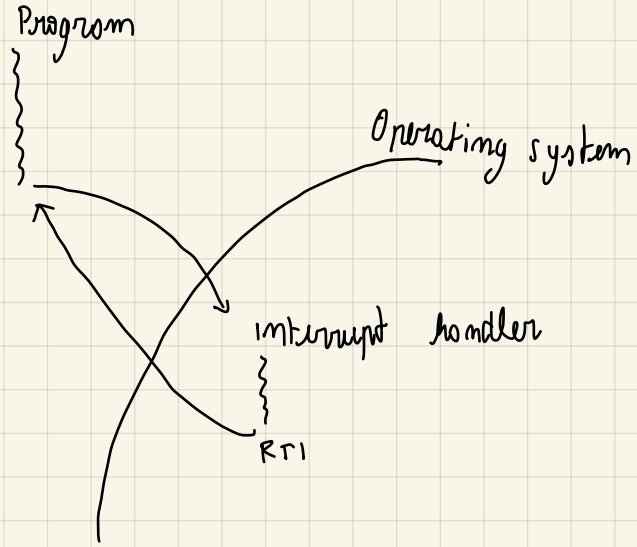
thrashing

13/05/2022

EXCEPTIONS / INTERRUPTS

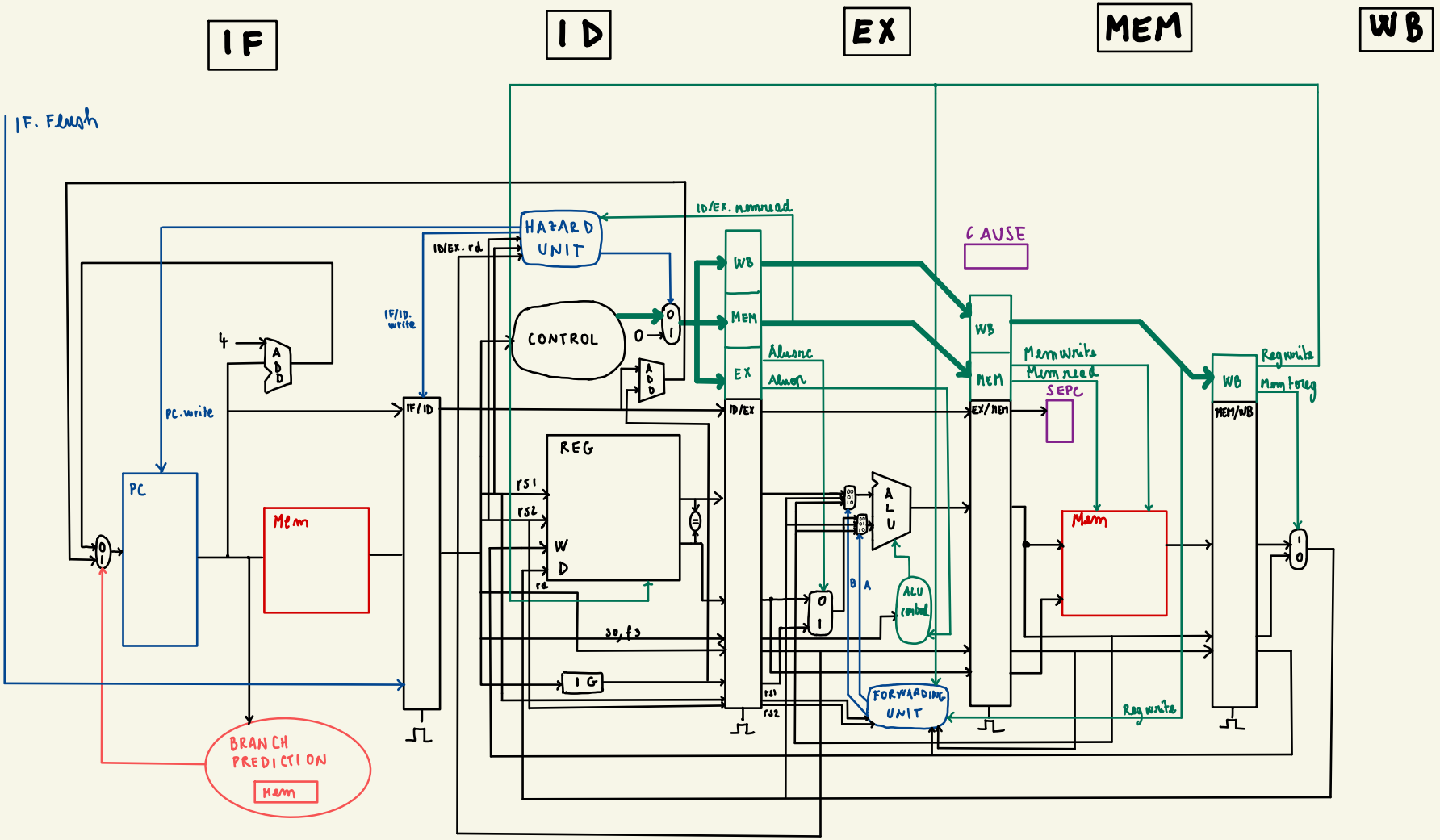
EXAMPLES

- signal to the process
- division by zero
- overflow
- (syscall)
- undefinite instruction
- memory violation
- hardware fault
- end of time slice
- I/O operations

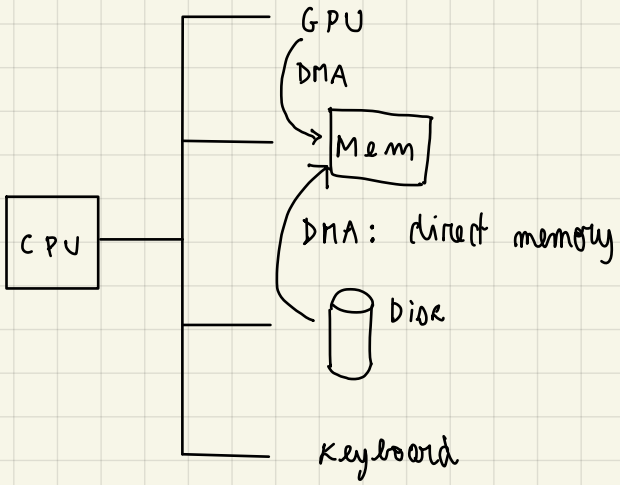


TIME SLICE

vector interrupt



I/O



DATA FROM I/O DEVICES TO CPU

Two ways:

1) I/O instructions (not very used)

2) Memory mapped I/O

EX.

0x 0000 0000
0x 0001 0000 } addresses reservated to the devices

INTERACTION BETWEEN I/O DEVICES AND

Polling: CPU asks periodically to the device if it has completed

interrupt driven I/O: I/O device sends a signal to the CPU after it has completed

PARALLELISM

SISD (single instruction single data)

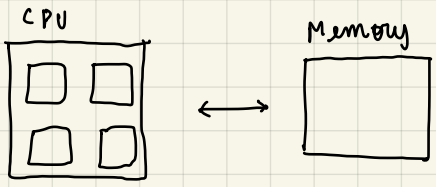
SIMD (/ / multiple /)

MISD (multiple / single /)

MIMD (multiple / multiple /) → multicores

MIMD

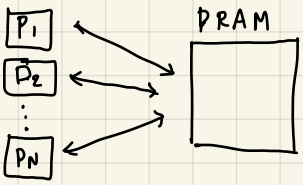
Multiple CPUs
Multiple cores



Multithreading

SIMD

Theoretical PRAM



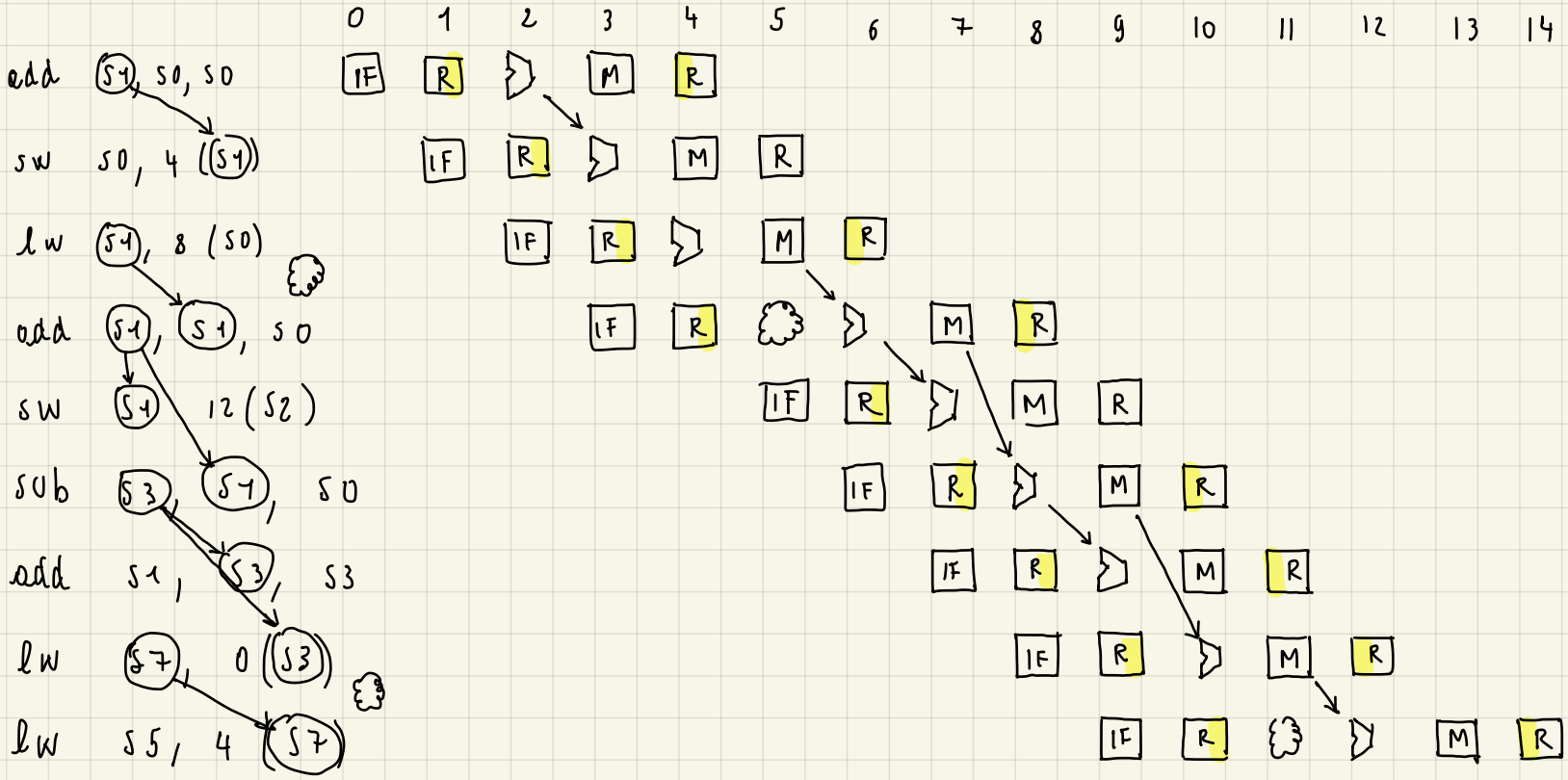
20/05/2022

Exercise 1. How many clock cycles are needed to complete the following code on the Risc-V pipelined architecure? (Figure in the next page.) Remember that:

- Execution completes when the last instruction exits the pipeline.
- If you read and write the same register of the register file in the same clock cycle, the value that is read is the value that is being witten.

Show a figure of the pipeline with bubbles and forwardings.

```
add s1, s0, s0
sw s0, 4(s1)
lw s1, 8(s0)
add s1, s1, s0
sw s1, 12(s2)
sub s3, s1, s0
add s1, s3, s3
lw s7, 0(s3)
lw s5,4(s7)
```


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
add s4, s0, s0
sw s0, 4(s4)
lw s4, 8(s0)
add s1, s1, s0
sw s4 12(s2)
sub s3, s4, s0
add s1, s3, s3
lw s7, 0(s3)
lw s5, 4 s7
IF R M R

Exercise 2. Consider a two-way associative cache with four sets and block of 8 words. The cache is initially empty and replacement of blocks is done by using LRU. Show which of the following memory accesses are hit and which are misses: 128, 16, 4, 184, 68, 204, 104, 28, 252, 240, 244, 40, 136, 164, 56, 256, 152, 180, 156, 64, 48.

block = 32 bytes

address	block	set	H/M
128	4	0	M
16	0	0	M
4	0	0	H
184	5	1	M
68	2	2	M
204	6	2	M
104	3	3	M
28	0	0	H
252	7	3	M
240	7	3	H
244	7	3	H
40	1	1	M
136	4	0	H
164	5	1	H
56	1	1	H
256	8	0	M
152	4	0	H
180	5	1	H
156	4	0	H
64	2	2	H
48	1	1	H

		8
0	4	~~0~~
1	5	1
2	2	6
3	3	7

0	0
1	32
2	64
3	96
4	128
5	160
6	192

Exercise 3. Consider the following code:

```
.data
v:      .word 0,1,7,4,2,...     # array of 1024 integers
w:      .word 0,0,0,0,0,...     # array of 1024 integers
n:      .word 1024

.text
        lui s0, 0x10012
        lw s2, 0(s0)
        lui s0, 0x10010
        lui s1, 0x10011
loop:   lw t0, 0(s0)
        sw t0, 0(s1)
        addi s2, s2, -1
        bne s2, zero, loop
        ecall
        li a7, 10
        ecall
```

data 0
data 64
code 0

addi s0, s0, 4
addi s1, s1, 4

This program copy one array into another.

1. **Question A:** what is the approximate total miss rate if you have an instruction cache and a data cache, both one-way associative with 8 blocks of 64 bytes?
2. **Question B:** What is the speed-up (how faster is it) if we use the Risc-V multiple issue architecture with loop unrolling of 4 loops instead of the standard pipelined architecure? Show the code.

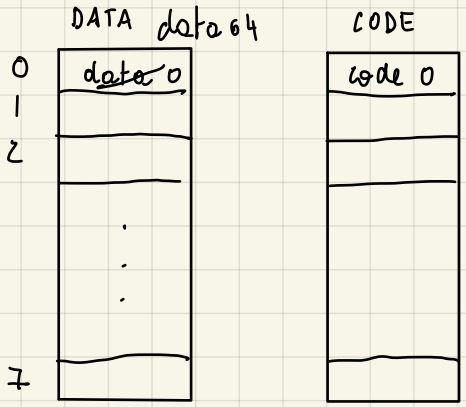
(1)

i 0 : H M H M H H H H

⋮ always the same

i 1023

miss rate : $\frac{1}{8} = 25\%$



(2)

standard pipeline (branch prediction)

clock cycles (for the loop) : 6 apart first and last iteration

total clock cylles $\simeq 6 \cdot 1024$

multiple issue architecture :

```
loop:   lw   t0, 0(s0)    | addi s0, s0, 16
        lw   t1, -12(s0)  |      /
        lw   t2, -8(s0)   |      /
        lw   t3, -4(s0)   |      /
        sw   t0, 0(s1)    | addi s1, s1, 16
        sw   t1, -12(s1)  |      /
        sw   t2, -8(s1)   | addi s2, s2, -4
        sw   t3, -4(s1)   | bne  s2, zero, loop
```

total clock cycles $= 8 \cdot \frac{1024}{4} = 2 \cdot 1024$

$\frac{6 \cdot \cancel{1024}}{2 \cdot \cancel{1024}} = 3 \Rightarrow$ M1 is 3X faster

.data

x: .word ____ 0x10010000 data 0
y: .word ____ 0x10014000 data 256
z: .word ____ 0x10018000 data 512
n: .word 4096 · 4 = 2^{14}

block = [x/64]

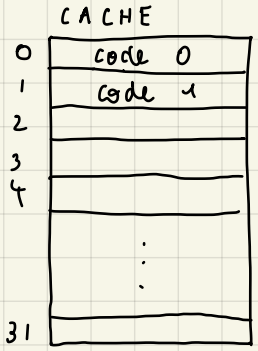
set = block % nsets

.text

```
        lo  t0, x    0x00400000
        la  t1, y
        lo  t2, z
        lw  t3, n
ciclo:  lw  t4, 0(t0)
        lw  t5, 0(t1)
        add t6, t4, t5
        sw  t6, 0(t2)
        addi t0, t0, 4
        /    t1  t1  /
        /    t2  t2  /
        /    t3  t3  -1
        bne t3, zero, ciclo
```

(16 words)

block size : 64 bytes
number of sets : 32
one-way associative
write-back



memory accesses per cycle: 12

		MISSES
i 0:	M M M M M H M M H H H M	8
i 1:	H M M M M H M M H H H H	6
⋮		⋮
i 15		
i 16	H M H M H H M H H H H M	4
⋮		⋮
i 32	H M H M H H M H H H H H	3
⋮		⋮
i 511		
i 512		

x 8

$$\frac{8 + 6 \cdot 15 + 4 \cdot 16 + 3 \cdot 480}{512 \cdot 12} = 26\%$$

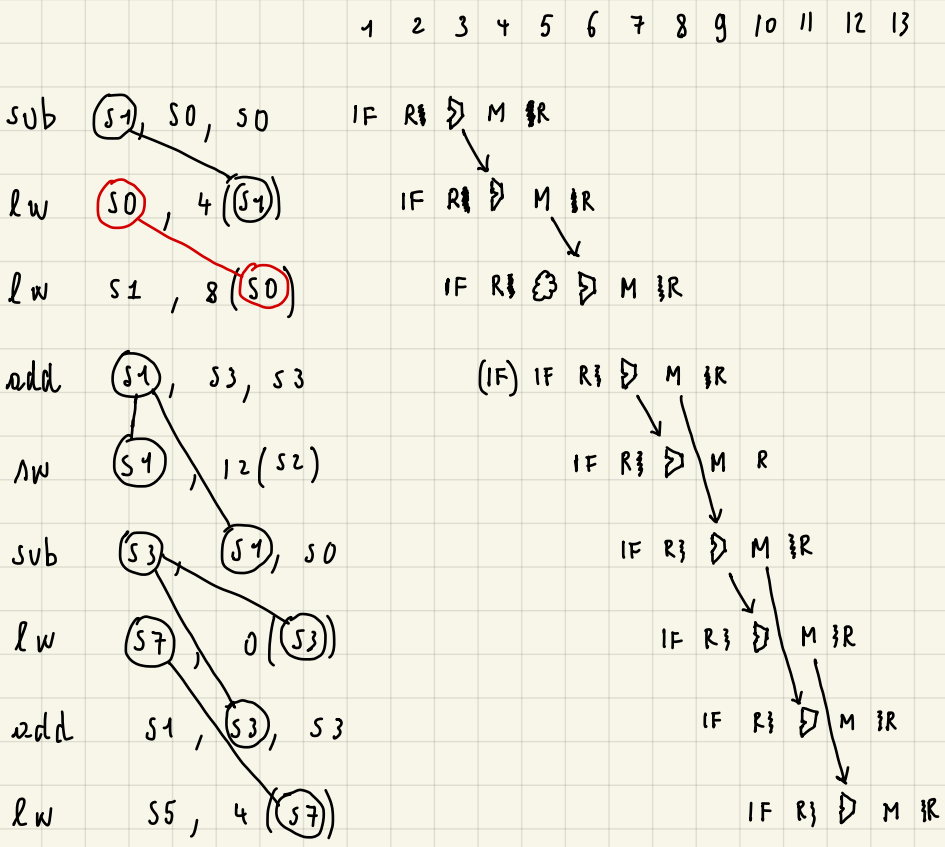
SAMPLE EXAM 2

Exercise 1. How many clock cycles are needed to complete the following code on the Risc-V pipelined architecure? (Figure in the next page.) Remember that:

- Execution completes when the last instruction exits the pipeline.
- If you read and write the same register of the register file in the same clock cycle, the value that is read is the value that is being witten.

Show a figure of the pipeline with bubbles and forwardings.

```
sub s1, s0, s0
lw s0, 4(s1)
lw s1, 8(s0)
add s1, s3, s3
sw s1, 12(s2)
sub s3, s1, s0
lw s7, 0(s3)
add s1, s3, s3
lw s5,4(s7)
```



Exercise 2. Consider a two-way associative cache with two sets and block of 8 words. The cache is initially empty and replacement of blocks is done by using LRU. Show which of the following memory accesses are hit and which are misses: 16, 136, 184, 252, 160, 80, 252, 80, 48, 136, 60, 204, 100, 88, 196, 60, 32, 176, 80, 224.

block size: 32 bytes

	BLOCK	SET	H/M
16	0	0	M
136	4	0	M
184	5	1	M
252	7	1	M
160	5	1	H
80	2	0	M
252	7	1	H
80	2	0	H
48	1	1	M
136	4	0	H
60	1	1	H
204	6	0	M
100	3	1	M
88	2	0	M
196	6	0	H
60	1	1	H
32	1	1	H
176	5	1	M
80	2	0	H
224	7	1	M

0	6	2
1	1	5

0	0
32	1
64	2
96	3
128	4
160	5
192	6
224	7
256	8

Exercise 3. Consider the following code:

```
.data
n000:   .word 8, n001    # list of 1024 integers
n001:   .word -3, n002
n002:   .word 5, n003
[...]

lista: .word n000

.text
        lui s0, 0x10012
        lw s1, 0(s0)
        li s2,0
loop:   beq s1, zero, fine
        lw s1, 4(s1)
        addi s2, s2, 1
        bne s1, zero, loop
fine:   ecall
        li a7, 10
        ecall
```

data 0

This program counts the number of nodes in the list.

1. **Question A:** what is the approximate total miss rate if you have an instruction cache and a data cache, both one-way associative with 8 blocks of 64 bytes?
2. **Question B:** What is the speed-up (how faster is it) if we use the Risc-V multiple issue architecture with loop unrolling of 4 loops instead of the standard pipelined architecure? Show the code.

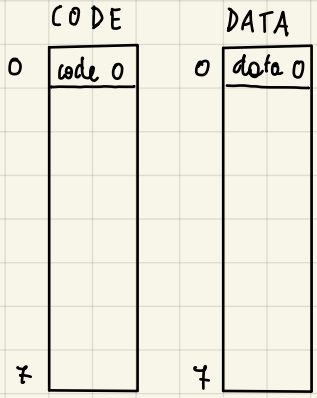
①

i 0 : H H M H H

i 1 : H H H H H

⋮

i 8 : H H H H H



$$\text{miss rate} = \frac{1}{40} = 2{,}5\,\%$$

②

```
beq s1, zero, fine | lw s1, 4(s1)
addi s2, s2, 1     |   /
beq  s1, zero, fine | lw s1, 4(s1)
addi s2, s2, 1     |   /
beq  s1, zero, fine | lw s1, 4(s1)
```

addi s2, s2, 1 | ✓

beq s4, zero, fine | lw s4, 4(s4)

addi s2, s2, 1 | ✓

bne s4, zero, loop | ✓

$$\frac{4 \cdot 1024}{9 \cdot 256} = \frac{16 \cdot \cancel{256}}{9 \cdot \cancel{256}} = 1,\overline{7}$$

M1 is 1,7x faster