

Name, SURNAME and ID \Rightarrow

KEY

Middle East Technical University
Department of Computer Engineering

CENG 331

Section 2
Fall '2021-2022

Final

- **Duration:** 40+40+20 minutes.

- **Exam:**

- Write your name, surname, and id on ALL THE PAGES!
- The exam will be held in three sessions: 40 + 40 + 20 minutes. This title page will only appear once in Part I and hence read it carefully since it also applies other parts.
- An Appendix will be handed to you as reference material. The appendix will not be graded, but you cannot take it out of the exam, and should return it at the end.
- If you have any questions, write your question on the back of the Appendix, fold it, and pass it to us. We will reply in written form.
- Make sure you cover both your nose and mouth with your mask. If you fail to do so, we have to remove you from the exam for the safety of other students.
- This is a **closed book, closed notes** exam.
- Write your answers only in the indicated spaces. Notes/writing on other parts of the page may be neglected during grading.
- No attempts of cheating will be tolerated. In case such attempts are observed, the students who took part in the act will be prosecuted. The legal code states that students who are found guilty of cheating shall be expelled from the university for **a minimum of one semester!**

1 (8 pts) Bits, ints.

- (a) (4 pts) Consider a computer system where int and unsigned numbers are represented with 30 bits and short and short unsigned numbers are represented with 15 bits.

Assuming that the maximum and minimum of the int/unsigned/short/short unsigned values are represented with capitalized constants. For instance, INTEGER_MAX represents the maximum value that an int variable can get.

What will the following C code print?

2000 0000 | 3FFF

```
int main(){
    unsigned result=INTEGER_MAX-UNSIGNED_MAX;
    short unsigned shortresult=INTEGER_MAX+SHORTINT_MIN;
    printf("Hexadecimal: result=%x shortresult=%x",result,shortresult);
}
```

INTEGER_MAX = 01 1111 1111 1111 1111 1111 1111 1111
 UNSIGNED_MAX = 11 1111 1111 1111 1111 1111 1111 1111
 ~UNSIGNED_MAX = 00 0000 0000 0000 0000 0000 0000 0001

result = 10 0000 0000 0000 0000 0000 0000 0000
 = 0x2 0000 0000

SHORTINT_MIN = 100 0000 0000 0000

INTEGER_MAX = 01 1111 1111 1111 1111 1111 1111 1111
 = 011 1111 1111 1111 = 0x3FFF

- (b) (4 pts) Write the C expression to generate the bit pattern $0^{w-k}1^k$ where a^k represents k repetitions of symbol a . Assume a w -bit data type. Your code may contain references to parameter k , representing the values of k , but not a parameter representing w .

$(1 \ll k) - 1$

$1 \ll k = 0 \dots 01 \underbrace{00 \dots 0}_k$
 $-1 = 1 \dots 11 \underbrace{11 \dots 1}_k$
 $(1 \ll k) - 1 = 0 \dots 00 \underbrace{11 \dots 1}_k$
 $= 0^{w-k} 1^k$

Name, SURNAME and ID \Rightarrow

- 2 (12 pts) Floats.** Consider a computer system where float numbers are represented with 14 bits. The float representation consisted of a single sign bit, followed by 5 bits representing the *exp*, with the remaining bits reserved for *frac*.

- (a) What is the largest positive real number that can be represented (do not consider infinity)? Write the binary representation in the first box and the result in full precision in decimal in the second box.

0 11110 1111 1111

65408

$$(-1)^0 \cdot (2 - 2^{-8}) \cdot 2^{15} = 2^{16} - 2^7$$

- (b) What is the smallest positive real number that can be represented above zero? Write the binary representation in the first box and the result as a fraction such as 1/512 in the second box.

0 00000 0000 0001

1/4194304

$$(-1)^0 \cdot 2^{-8} \cdot 2^{-14} = 2^{-22}$$

- (c) What is the smallest interval between any two consecutive numbers represented in float representation? Write the result as a fraction such as 1/512.

1/4194304

2^{-22}

- (d) What is the largest interval between any two consecutive numbers represented in float representation? Write the result as a decimal number.

128

2^7

Next largest: 0 11110 1111 1110

$$(-1)^0 (2 - 2^{-7}) \cdot 2^{15} = 2^{16} - 2^8$$

$$\text{Difference: } (2^{16} - 2^7) - (2^{16} - 2^8)$$

$$= 2^{16} - 2^7 - 2^{16} + 2^8$$

$$= 2^7$$

Name, SURNAME and ID ⇒

- 3 (7 pts) Assembly.** Circle the C-code that the following assembly code is generated from. *Wrong answers: -2 pts.*

```
func:pushq %rbp
      movq %rsp, %rbp
      movq %rdi, -24(%rbp)
      movl %esi, -28(%rbp)
      movl -28(%rbp), %eax
      subl $1, %eax
      movl %eax, -8(%rbp)
      movl $1, -4(%rbp)
      jmp .L2
.L3:  movl -8(%rbp), %eax
      cltq      ;sign extends eax into rax
      leaq 0(,%rax,4), %rdx
      movq -24(%rbp), %rax
      addq %rdx, %rax
      movl (%rax), %eax
      cmpl %eax, -4(%rbp)
      cmovge -4(%rbp), %eax
      movl %eax, -4(%rbp)
      subl $1, -8(%rbp)
.L2:  cmpl $0, -8(%rbp)
      jns .L3
      movl -4(%rbp), %eax
      popq %rbp
      ret
```

- (a) `int func(int* arr, int len){
 int i, res;
 for(i = len-1, res = 1; i >= 0; i--) res += arr[i];
 return res;}`
- (b) `int func(int* arr, int len){
 int i, res;
 for(i = len-1, res = 0; i >= 0; i--) res = arr[i] > res ? arr[i] : res;
 return res;}`
- (c) `int func(int* arr, int len){
 int i, res;
 for(i = len-1, res = 1; i >= 0; i--) res *= arr[i];
 return res;}`
- ☒ (d) `int func(int* arr, int len){
 int i, res;
 for(i = len-1, res = 1; i >= 0; i--) res = arr[i] > res ? arr[i] : res;
 return res;}`
- (e) `int func(int* arr, int len){
 int i, res;
 for(i = 0, res = 1; i < len; i++) res = arr[i] > res ? arr[i] : res;
 return res;}`

Name, SURNAME and ID ⇒

- 4 (8 pts) Assembly. Consider the following source code, where M and N are constants declared with `#define`:

```
long P[M][N];
long Q[N][M];
long sum_element(long i, long j) {
    return P[i][j] + Q[j][i];
}
```

In compiling this program, gcc generates the following assembly code:

```
sum_element:
    leaq 0(,%rdi,8), %rdx
    subq %rdi, %rdx
    addq %rsi, %rdx
    leaq (%rsi,%rsi,4), %rax
    addq %rax, %rdi
    movq Q(,%rdi,8), %rax
    addq P(,%rdx,8), %rax
    ret
```

Use your reverse engineering skills to determine the values of M and N based on this assembly code.

$M=5, N=7$

Reference to P is at byte offset $8(7i+1)$
Reference to Q is at byte offset $8(5j+i)$
⇒ P has 7 columns
Q has 5 columns

**** End of Part I ****

5

- Page size: 2048 bytes = 2^{11}

 2^{21}

VA $\leftarrow 27 \rightarrow \leftarrow 11 \rightarrow$

VNP	VPO
-----	-----

 $\leftarrow 21 \rightarrow \leftarrow 6 \rightarrow$
 TLB Tag TLB Index

Access $A[0], A[1], \dots, A[i]$

\downarrow \downarrow \downarrow

0×0 $0 \times 0 + 1 \times 4$ $0 \times 0 + i \times 4$

\swarrow sizeof(int) = 4

One page holds $2^{11}/2^2 = 2^9$ elements of A.
TLB miss happens whenever we switch from one page to another.

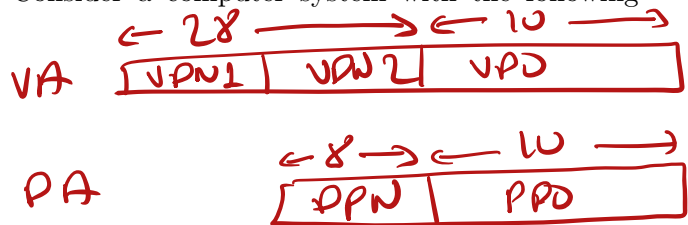
\Rightarrow # of TLB misses $= 2^{30} / 2^9 = 2^{21}$

The size of Physical Memory is irrelevant
The size of TLB is also irrelevant, since we
iterate only once.

Name, SURNAME and ID \Rightarrow

6 (15 pts) Virtual Memory 1. Consider a computer system with the following specifications:

- Virtual memory: 2^{38} bytes
- Physical memory: 2^{18} bytes
- Page size: 2^{10} bytes



We want to have a virtual memory system with a 2-level page system, such that the page tables at the second (i.e. last) level fits in a single page. Assume that the Page Table Entry (PTE) is only big enough to store the Valid bit and the Physical Page Number (PPN), and the size of the PTE can only be a power of 2 bytes (i.e. size of PTE can be 2, 4, 8 but not 3, or 6 bytes).

(a) What is the size of the PTE in bytes?

2

(b) How many PTEs does a second-level page table contain?

$$|\text{VPN2}| = 9$$

$$2^{10}/2 = 2^9 \quad 512$$

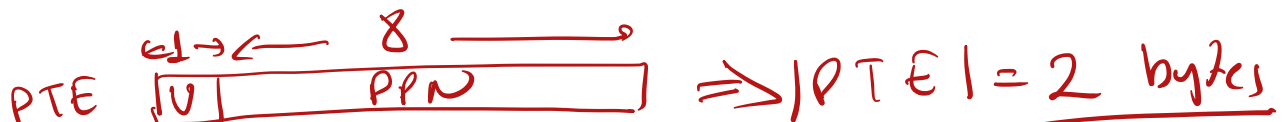
(c) How many PTEs does the first-level page table contain?

$$|\text{VPN1}| = 28 - |\text{VPN2}| = 19$$

$$2^{19}$$

(d) How many bytes is the first level page table?

$$2^{19} \times 2 = 2^{20}$$



Name, SURNAME and ID ⇒

7 (15 pts) Virtual Memory 2. The following problem concerns the way virtual addresses are translated into physical addresses.

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not 4-byte words).
- Virtual addresses are 17 bits wide.
- Physical addresses are 15 bits wide.
- The page size is 512 bytes. $= 2^9$ $1VP01 = 1PP01 = 9$
- The TLB is 4-way set associative with 16 total entries. $1TLBIndex = 2$
- The cache is physically addressed.
- The cache is 2-way set associative, with a 4 byte line size and 16 total lines. $1CacheIndex = 3$
 $1CacheOffset = 2$

In the following tables, **all numbers are given in hexadecimal**. The contents of the TLB, the page table for the first 32 pages, and the cache are as follows:

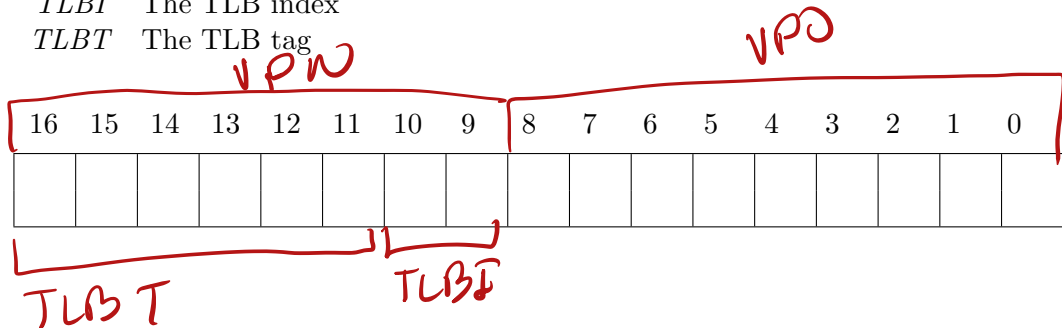
TLB			
Index	Tag	PPN	Valid
0	09	4	1
	12	2	1
	10	0	1
	08	5	1
1	05	7	1
	13	1	0
	10	3	0
	18	3	0
2	04	1	0
	0C	1	0
	12	0	0
	03	1	0
3	06	7	0
	03	1	0
	07	5	0
	02	2	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	0	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	5	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	7	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	7	1	1E	3	0
0F	0	0	1F	1	0

2-way Set Associative Cache												
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	00	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	FD	1	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	1E	1	00	C0	0F	00

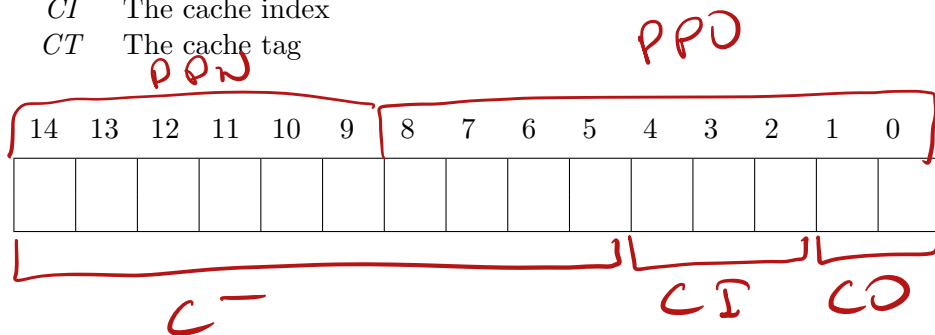
- (1.5 pts) The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following:
(If a field doesn't exist, don't draw it on the diagram.)

VPO The virtual page offset
VPN The virtual page number
TLBI The TLB index
TLBT The TLB tag



- (1.5 pts) The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

PPO The physical page offset
PPN The physical page number
CO The block offset within the cache line
CI The cache index
CT The cache tag



For the given virtual address, indicate the TLB entry accessed, the physical address, and the cache byte value returned **in hex**. Indicate whether the TLB misses, whether a page fault occurs, and whether a cache miss occurs.

If there is a cache miss, enter “-” for “Cache Byte returned”. If there is a page fault, enter “-” for “PPN” and leave parts C and D blank.

Virtual address: 1DDE

- (2 pts) Virtual address format (one bit per box) VPN VPO
- | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
- TLB Tag TLB I

- (4 pts) Address translation

Parameter	Value
VPN	0x 0E
TLB Index	0x 2
TLB Tag	0x 3
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	N
PPN	0x 7

- (2 pts) Physical address format (one bit per box) PPN PPO

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	1	1	1	0	1	1	1	1	0

- (4 pts) Physical memory reference

Parameter	Value
Byte offset	0x 2
Cache Index	0x 7
Cache Tag	0x 7E
Cache Hit? (Y/N)	N
Cache Byte returned	0x —

CS Cache/Byte Offset

**** End of Part II ****

Name, SURNAME and ID \Rightarrow

- 8 (15 pts) Processor design.** We want to add a “smart call” instruction to the Y86-64 sequential implementation (SEQ). The new instruction

`scall D(rA), rB`

will call the function located at address $D+R[rA]$, and store return address at register rB , instead of pushing it on the stack.

The new instruction, named `scall`, will be a 10-byte instruction as follows:

0	1	2	3	4	5	6	7	8	9
E0	ra:rB		D						

where E0 is the 1-byte coding `icode:ifun`, `ra:rB` encode the registers involved and D is an immediate, 8 byte value. Fill out the following form to describe what needs to happen during each stage of this new instruction.

- (a) (2 pts)** Write down the stages of execution.

Stage	Computation
Fetch	$icode:ifun \leftarrow M_1[PC]$ $ra:rB \leftarrow M_1[PC+1]$ $valC \leftarrow M_8[PC+2]$ $valP \leftarrow PC+10$
Decode	$valA \leftarrow R[ra]$
Execute	$valE \leftarrow valA + valC$
Memory	—
Write-back	$R[rB] \leftarrow valP$
PC-update	$PC \leftarrow valE$

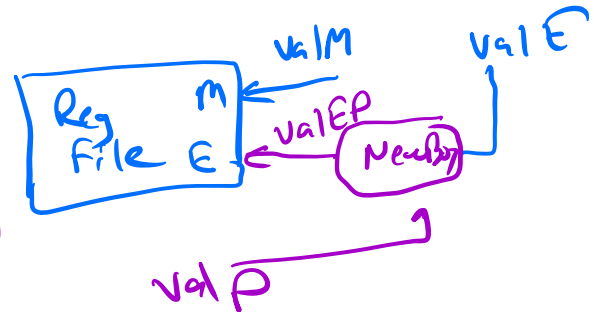
- (b) (13 pts) Write down the HCL code that needs to be changed using the SEQ HCL code in the appendix as a reference.

```
wordsig ISCALL 'I_ISCALL'
bool instr_valid = icode in { ....., ISCALL };
```

```
bool need_regs = icode in { ....., ISCALL }
bool need_valC = icode in { ....., ISCALL }
word srcA = icode in { ....., ISCALL } : rA
word dstE = icode in { ....., ISCALL } : rB
```

Need an extra control box

```
word valEP = { icode == ISCALL : valP
               : valE }
```



```
word aluA = [ icode in { ....., ISCALL } : valA;
              ]
word aluB = [ icode in { ....., ISCALL } : valC;
              ]
word newPC = [ icode == ISCALL : valE;
               ]
```

Name, SURNAME and ID \Rightarrow

9 (10pts) Processor design. Consider the final version of the 5-stage PIPE processor developed during the lectures.

(a) (2 pts) How many bubbles are injected in the pipeline during the execution of the following instructions? (2 pts)

- instruction causing load-use hazard: LU=...1.....
- call instruction: C=... ϕ
- correctly predicted branch instruction: CP=... ϕ
- mispredicted branch instruction: MB=...2.....
- ret instruction: R=...3.....

Now consider the execution of a program with the following characteristics:

- The number of instructions: 1400
- The number of all data hazards: 560
- The number of load-use hazards: 140
- The number of call instructions: 70
- The number of branches: 140
- The probability of mispredicting branches: 0.4
- The number of ret instructions: 14

of cycles to empty the pipeline!
The lack of this term is not penalized in grading.

(b) (6 pts) What is the CPI (Cycles per Instruction) for the execution of this program in terms of LU, C, CP, MB, R?

$$1 + \frac{(140 * LU + 140 * 0.4 * MB + 14 * 3 + 4)}{1400}$$

(c) (2 pts) What is the CPI (Cycles per Instruction) for the execution of this program as a numerical value?

$$1 + \frac{140 + 112 + 42 + 4}{1400} = 1 + \frac{298}{1400} = 1.21$$

**** End of Exam ****