

# Домаћи задатак из Архитектуре рачунара за школску 2021/2022.

Задатак се састоји из три дела А, Б и В. Сваки део има четири варијанте (А0-А3, Б0-Б3, В0-В3). Студент на основу свог броја индекса добија комбинацију делова који чине његов домаћи задатак (од сваког дела добија по једну варијанту на основу броја индекса). Студент са бројем индекса  $b_4b_3b_2b_1/g_4g_3g_2g_1$  добија делове на следећи начин:

- Део А као  $b_1 \bmod 4$
- Део Б као  $b_2 \bmod 4$
- Део В као  $b_3 \bmod 4$

Пример: студент 0354/2021 ради домаћи задатак А0 Б1 В3.

Решење је потребно написати унутар фајла *VEZBA.ASM* који садржи неопходна подешавања симулације и исти је за све могуће комбинације домаћег задатка.

## Део А

**А0.** Главни програм сместити од адресе 4000h. Иницијализовати IV табелу и IVTP регистар према следећој табели (колона Уређај садржи ознаку уређаја коме припада улаз):

Број улаза	Садржај	Уређај
0	0500h	KP1.1
1	1000h	KP1.2
2	1500h	KP2.1
3	2000h	DMA1.1
4	2500h	DMA1.2
5	3000h	DMA1.4
IVTP = 0000h		

Потребно је учитати низове А и В и сместити их у меморију почевши од адреса 5000h и 6000h, респективно. Оба низа имају по 8h елемената. Низ А учитати са KP1.1 испитивањем бита спремности, а низ В учитати са KP2.1 коришћењем механизма прекида. Учитавање обавити упоредо са обе периферије.

**А1.** Главни програм сместити од адресе 4000h. Иницијализовати IV табелу и IVTP регистар према следећој табели (колона Уређај садржи ознаку уређаја коме припада улаз):

Број улаза	Садржај	Уређај
5	0500h	KP1.1
4	1000h	KP1.2
2	1500h	KP2.1
3	2000h	DMA1.1
1	2500h	DMA1.2
0	3000h	DMA1.4
IVTP = 0100h		

Потребно је учитати низове А и В и сместити их у меморију почевши од адреса 5000h и 6000h, респективно. Оба низа имају по 8h елемената. Низ А учитати са KP1.1 коришћењем механизма прекида, а низ В учитати са KP2.1 испитивањем бита спремности. Учитавање обавити упоредо са обе периферије.

**A2.** Главни програм сместити од адресе 4000h. Иницијализовати IV табелу и IVTP регистар према следећој табели (колона Уређај садржи ознаку уређаја коме припада улаз):

Број улаза	Садржај	Уређај
5	0500h	KP1.1
4	1000h	KP1.2
3	1500h	KP2.1
2	2000h	DMA1.1
0	2500h	DMA1.2
1	3000h	DMA1.4
IVTP = 0200h		

Потребно је учитати низове А и В и сместити их у меморију почевши од адреса 5000h и 6000h, респективно. Оба низа имају по 8h елемената. Низ А учитати са KP1.1 испитивањем бита спремности, а низ В учитати са KP2.1 испитивањем бита спремности. Учитавање обавити упоредо са обе периферије.

**A3.** Главни програм сместити од адресе 4000h. Иницијализовати IV табелу и IVTP регистар према следећој табели (колона Уређај садржи ознаку уређаја коме припада улаз):

Број улаза	Садржај	Уређај
3	0500h	KP1.1
4	1000h	KP1.2
1	1500h	KP2.1
0	2000h	DMA1.1
2	2500h	DMA1.2
5	3000h	DMA1.4
IVTP = 0300h		

Потребно је учитати низове А и В и сместити их у меморију почевши од адреса 5000h и 6000h, респективно. Оба низа имају по 8h елемената. Низ А учитати са KP1.1 коришћењем механизма прекида, а низ В учитати са KP2.1 коришћењем механизма прекида. Учитавање обавити упоредо са обе периферије.

## Део Б

**B0.** Написати потпрограма `int sumAll(int* arr1, int* arr2, int n)` која рачуна збир елемената два низа и резултат враћа унутар регистра R0. Параметри `arr1` и `arr2` представљају показиваче на низове чије елементе треба сабрати, а параметар `n` представља колико елемената из прослеђених низова треба сабрати. Након пријема низова А и В потребно је позвати функцију `sumAll` тако да се саберу сви елементи низова А и В. Након позива функције, резултат из R0 сачувати и на меморијску локацију 9999h. Потребно је још у меморију почев од локације 6100h прекопирати низ В коришћењем DMA1.4 уређаја у пакетском режиму рада.

**B1.** Написати потпрограма `void xorArr(int* arr1, int* arr2, int n)` која врши битско ексклузивно или на следећи начин `arr1[i] = arr1[i] xor arr2[i]`. Параметри `arr1` и `arr2` представљају показиваче на низове, а параметар `n` представља број елемената низова. Након пријема низова А и В потребно је позвати функцију `xorArr` тако да се израчуна  $A[i] = A[i] \text{ xor } B[i]$ . Након позива функције, нулти елемент низа А сместити на локацију 9999h. Потребно је још нулти елемент низа А прекопирати на 8h сукцесивних локација почевши од локације 5100h коришћењем DMA1.4 у пакетском режиму рада.

**Б2.** Написати потпрограм `void process(int* arr1, int n, Request* r)` која у низу дужине `n` на који показује `arr1` проналази најмањи или највећи елемент на основу вредности поља `operation` у структури на коју показује `r`. Вредност 0 поља `operation` значи да се тражи најмањи, а вредност 1 да се тражи највећи елемент. Пронађени елемент треба сместити у поље `element` структуре на коју показује `r`. Након пријема низова `A` и `B` потребно је позвати потпрограм `process` за оба низа, структуре за ова два позива се налазе у меморији на адресама `9996h` и `9998h`, респективно. Структура је дефинисана као `struct Request { int operation; int element; }`.

**Б3.** Написати потпрограм `void processElem(int* elem)` која треба да податак на који показује `elem` комплементира. Након пријема низова `A` и `B` за сваки елемент низа `A[i]` позвати потпрограм `processElem` ако је одговарајући елемент низа `B[i]` паран број. Након овога нулти елемент низа `A` сместити на локацију `9999h`. Потребно је још у меморију почев од локације `5100h` прекопирати новодобијени низ `A` коришћењем `DMA1.4` уређаја у пакетском режиму рада.

## Део В

**В0.** Након завршеног дела Б задатка, низ `A` послати периферији `KP1.2` испитивањем бита спремности. Вредност са меморијске локације `9999h` послати уређају `DMA1.2` у пакетском режиму раду.

**В1.** Након завршеног дела Б задатка, низ `A` послати уређају `DMA1.1` у пакетском режиму рада. Вредност са меморијске локације `9999h` послати уређају `DMA1.2` у режиму рада циклус по циклус.

**В2.** Након завршеног дела Б задатка, низ `A` послати периферији `KP1.2` коришћењем механизма прекида. Вредност са меморијске локације `9999h` послати уређају `DMA1.2` у пакетском режиму раду.

**В3.** Након завршеног дела Б задатка, низ `A` уређају `DMA1.1` у режиму рада циклус по циклус. Вредност са меморијске локације `9999h` послати уређају `DMA1.2` у пакетском режиму раду.

# Документација

У следећим табелама дате су адресе релевантних регистара контролера периферија, све адресе су дате у хексадецималном облику. Меморијски адресни простор и улазно излазни адресни простор су меморијски пресликани (мапирани).

Периферија	Control	Status	Entry	Data
КР1 (КР1.1)	F100	F101	F102	F103
КР1.2	F140	F141	F142	F143
КР1.3	F180	F181	F182	F183
КР1.4	F1C0	F1C1	F1C2	F1C3

Периферија	Control	Status	Entry	Data
КР2 (КР2.1)	F200	F201	F202	F203
КР2.2	F240	F241	F242	F243
КР2.3	F280	F281	F282	F283
КР2.4	F2C0	F2C1	F2C2	F2C3

Формат контролних регистара периферија:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
												nivo	start	enable	ulaz\ilaz

Бит *start* служи за укључивање периферије вредношћу 1.

Бит *ulaz\izlaz* вредношћу 1 означава да се ради о улазу, а вредношћу 0 да се ради о излазу.

Бит *nivo* мора имати вредност 1 уколико периферија ради коришћењем прекида (уколико *enable* бит има вредност 1), у свим други случајевима има вредност 0.

У статусним регистрима контролера периферија најнижи бит је бит *ready*.

Формат статусних регистара периферија:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
															ready

У наредној табели дате су адресе релевантних регистара контролера периферија са директним приступом меморији (DMA).

Контролер	Control	Entry	Count	AR1	AR2
DMA1.1 (DMA)	F000	F002	F004	F005	F006
DMA1.2	F040	F042	F044	F045	F046
DMA1.3	F080	F082	F084	F085	F086
DMA1.4	F0C0	F0C2	F0C4	F0C5	F0C6

Регистар *AR1* је изворишни адресни регистар, а *AR2* је оредишни адресни регистар. Регистар *Count* је бројачки регистар.

Формат контролних регистара контролера периферија са директним приступом меморији:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								burst /cycle_stealing	inc/dec	update	mem	nivo	start	enable	ulaz/izlaz

Бит *start* служи за укључивање периферије вредношћу 1.

Бит *ulaz/izlaz* вредношћу 1 означава да се ради о улазу (подаци се пребацују са периферије у меморију), а вредношћу 0 да се ради о излазу (подаци се пребацују из меморије на периферију). Бит *mem* уколико се ради о улазу или излазу мора имати вредност 0. Бит *mem* уколико има вредност један тада се ради о трансферу података из меморије у меморију.

Бит *nivo* мора имати вредност 1 уколико контролер периферије са директним приступом меморији ради коришћењем прекида (уколико *enable* бит има вредност 1).

Бит *burst /cycle\_stealing* вредношћу 1 означава да се пренос врши у пакетском режиму, а вредношћу 0 да се пренос ради у појединачним циклусима на магистралама.

Бит *update* вредностима 0 и 1 означава да ли је задат режим рада без или са ажурирањем вредности изворишног адресног регистра *AR1*, респективно. Вредност дестинационог адресног регистра *AR2* увек се ажурира.

Бит *inc/dec* вредностима 0 и 1 означава да ли се вредност регистра *AR1* инкрементира или декрементира, респективно, уколико је бит *update* постављен. Вредност регистра *AR2* увек се инкрементира или декрементира на основу бита *inc/dec*.

## Инструкције асемблера

Коментари у асемблеру *PSasm* пишу се након знака !.

Операнд регистра се обележава *rN*, где је *N* индекс регистра у хексадекадном запису. Индекс може имати вредности од 0 до F (15). У табелама које следе је означен са *reg*.

Операнд броја се обележава *x.N*, где је *N* број у хексадекадном запису. Може имати вредности од 0 до FFFF (65535). У табелама које следе је означен са *num*.

Директивом **org x.N** се назначава да све инструкције које следе након ове директиве смештају у меморији почев од адресе *x.N*.

Лабеле у коду се означавају стрингом након кога следи знак :.

У табели 1 су регистри **imr** и **ivtp** означени са **ireg**, а са **preg** су означени регистри **sp** и **psw**.

Табела 1. Инструкције преноса података

Формат инструкције	Опис
<b>ldimm num, reg</b>	У регистар <b>reg</b> уписује вредност <b>num</b> .
<b>ldmem num, reg</b>	У регистар <b>reg</b> уписује вредност из меморије са адресе <b>num</b> .

<b>ldrid [reg1]num, reg2</b>	У регистар <b>reg2</b> уписује вредност из меморије са адресе <b>reg1+num</b> .
<b>stri [reg1], reg2</b>	Уписује вредност регистра <b>reg2</b> у меморијску локацију на адреси која се налази у регистру <b>reg1</b> .
<b>stmem num, reg</b>	Уписује вредност регистра <b>reg</b> у меморијску локацију на адреси <b>num</b> .
<b>mvril reg, ireg</b>	Уписује вредност регистра <b>ireg</b> у регистар <b>reg</b> .
<b>mvrir reg, ireg</b>	Уписује вредност регистра <b>reg</b> у регистар <b>ireg</b> .
<b>mvrri reg1, reg2</b>	Уписује вредност регистра <b>reg2</b> у регистар <b>reg1</b> .
<b>mvrpr reg, preg</b>	Уписује вредност регистра <b>reg</b> у регистар <b>preg</b> .
<b>mvrpl reg, preg</b>	Уписује вредност регистра <b>preg</b> у регистар <b>reg</b> .
<b>push reg</b>	Уписује вредност регистра <b>reg</b> на меморијску локацију на врху стека.
<b>pop reg</b>	Уписује вредност меморијске локације врха стека у регистар <b>reg</b> .
<b>clr reg</b>	Уписује вредност <b>0</b> у регистар <b>reg</b> .

Табела 2. Аритметичке инструкције

Формат инструкције	Опис
<b>add reg1, reg2, reg3</b>	Уписује збир вредности регистара <b>reg2</b> и <b>reg3</b> у регистар <b>reg1</b> . У регистар <b>psw</b> се уписују одговарајући <b>NZCV</b> битови.
<b>sub reg1, reg2, reg3</b>	Уписује разлику вредности регистара <b>reg2</b> и <b>reg3</b> у регистар <b>reg1</b> . У регистар <b>psw</b> се уписују одговарајући <b>NZCV</b> битови.
<b>cmp reg1, reg2</b>	Израчунава разлику вредности регистара <b>reg1</b> и <b>reg2</b> , и у регистар <b>psw</b> уписује одговарајуће <b>NZCV</b> битове.
<b>inc reg</b>	Инкрементира (повећава за 1) вредност регистра <b>reg</b> , и у регистар <b>psw</b> уписује одговарајуће <b>NZCV</b> битове.
<b>dec reg</b>	Декрементира (умањује за 1) вредност регистра <b>reg</b> , и у регистар <b>psw</b> уписује одговарајуће <b>NZCV</b> битове.

Табела 3. Логичке инструкције

Формат инструкције	Опис
<b>and reg1, reg2, reg3</b>	Уписује резултат операције битског „И” над регистрима <b>reg2</b> и <b>reg3</b> у регистар <b>reg1</b> . У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, а <b>V</b> бит се брише.
<b>or reg1, reg2, reg3</b>	Уписује резултат операције битског „ИЛИ” над регистрима <b>reg2</b> и <b>reg3</b> у регистар <b>reg1</b> . У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, а <b>V</b> бит се брише.
<b>xor reg1, reg2, reg3</b>	Уписује резултат операције битског „ексклузивног ИЛИ” над регистрима <b>reg2</b> и <b>reg3</b> у регистар <b>reg1</b> . У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, а <b>V</b> бит се брише.
<b>tst reg1, reg2</b>	Израчунава вредност операције битског „И” над регистрима <b>reg1</b> и <b>reg2</b> , и у регистар <b>psw</b> уписује одговарајуће <b>NZ</b> битове, а <b>V</b> бит

	се брише.
<b>not reg</b>	Уписује комплементирану вредност регистра <b>reg</b> у тај регистар. У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, а <b>V</b> бит се брише.

Табела 4. Инструкције померања и ротирања

Формат инструкције	Опис
<b>asr reg</b>	Врши аритметичко померање удесно битова регистра <b>reg</b> . У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, <b>C</b> бит добија вредност претходног најнижег бита <b>reg</b> регистра, а <b>V</b> бит се брише.
<b>lsr reg</b>	Врши логичко померање удесно битова регистра <b>reg</b> . У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, <b>C</b> бит добија вредност претходног најнижег бита <b>reg</b> регистра, а <b>V</b> бит се брише.
<b>ror reg</b>	Врши ротирање удесно битова регистра <b>reg</b> . У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, <b>C</b> бит добија вредност претходног најнижег бита <b>reg</b> регистра, а <b>V</b> бит се брише.
<b>rorc reg</b>	Врши ротирање удесно битова регистра <b>reg</b> , с тиме да се бит <b>C</b> регистра <b>psw</b> посматра као бит на 16. позицији. У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, а <b>V</b> бит се брише.
<b>sl reg</b>	Врши померање улево битова регистра <b>reg</b> . У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, <b>C</b> бит добија вредност претходног највишег бита <b>reg</b> регистра, а <b>V</b> бит се брише.
<b>rol reg</b>	Врши ротирање улево битова регистра <b>reg</b> . У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, <b>C</b> бит добија вредност претходног највишег бита <b>reg</b> регистра, а <b>V</b> бит се брише.
<b>rolc reg</b>	Врши ротирање улево битова регистра <b>reg</b> , с тиме да се бит <b>C</b> регистра <b>psw</b> посматра као бит на 16. позицији. У регистар <b>psw</b> се уписују одговарајући <b>NZ</b> битови, а <b>V</b> бит се брише.

Табела 5. Инструкције скокова

Формат инструкције	Опис
<b>beql labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на релацију „једнако” ( <b>Z = 1</b> ).
<b>bneq labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на релацију „неједнако” ( <b>Z = 0</b> ).
<b>bgrt labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на релацију „веће” ( <b>(N xor V) or Z = 0</b> ).
<b>bgre labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на релацију „веће или једнако” ( <b>N xor V = 0</b> ).
<b>blss labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на релацију „мање” ( <b>(N xor V) = 1</b> ).
<b>bleq labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на релацију „мање или једнако” ( <b>(N xor V) or Z = 1</b> ).

<b>bgrtu labela</b>	Врши скок на задату лабела у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на релацију „веће” у случају посматрања регистара као неозначених вредности ( <b>C or Z = 0</b> ).
<b>bgreu labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на релацију „веће или једнако” у случају посматрања регистара као неозначених вредности ( <b>C = 0</b> ).
<b>blssu labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на релацију „мање” у случају посматрања регистара као неозначених вредности ( <b>C = 1</b> ).
<b>blequ labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на релацију „мање или једнако” у случају посматрања регистара као неозначених вредности ( <b>C or Z = 1</b> ).
<b>bneq labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на негативну вредност ( <b>N = 1</b> ).
<b>bnng labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на ненегативну вредност ( <b>N = 0</b> ).
<b>bovf labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на то да је дошло до прекорачења ( <b>V = 1</b> ).
<b>bnvf labela</b>	Врши скок на задату лабелу у случају да битови <b>NZCV</b> регистра <b>psw</b> указују на то да није дошло до прекорачења ( <b>V = 0</b> ).
<b>jmp labela</b>	Врши безусловни скок на задату лабелу.
<b>jmpind reg</b>	Врши безусловни скок на задату адресу записану у регистру <b>reg</b> .
<b>jsr labela</b>	Врши скок у потпрограма на задатој лабели.
<b>rts</b>	Врши повратак из потпрограма.
<b>int num</b>	<b>Num</b> је у интервалу [0x00, 0xFF] хексадекадних вредности, врши скок у прекидну рутину чији је број улаза једнак <b>num</b> .
<b>rti</b>	Врши повратак из прекидне рутине.

Табела 6. Инструкције постављања индикатора у PSW регистар

Формат инструкције	Опис
<b>intd</b>	Поставља вредност 0 на биту <b>I</b> регистра <b>psw</b> .
<b>inte</b>	Поставља вредност 1 на биту <b>I</b> регистра <b>psw</b> .
<b>trpd</b>	Поставља вредност 0 на биту <b>T</b> регистра <b>psw</b> .
<b>trpe</b>	Поставља вредност 1 на биту <b>T</b> регистра <b>psw</b> .

Табела 7. Инструкција заустављања

Формат инструкције	Опис
<b>halt</b>	Зауставља процесор.



## Напомене:

- За израду домаћег задатка користити асемблер *SPasm* и симулатор *SPECSeo* на начин описан у материјалима за лабораторијске вежбе;
- Фајл *VEZBA.ASM* на почетку има део који је ограничен коментарима ! *inicijalizacija simulacije* и ! *kraj inicijalizacije*, наредбе које се налазе између ова два коментара не треба мењати;
- Периферију (без DMA) која ради коришћењем механизма прекида није могуће искључити у тренутку када пошаље/прими последњи податак, него тек следећи пут када пошаље захтев за прекид;
- Потпрограми се пишу одмах након краја главног програма (испод наредбе *halt*) тако што се наведе лабела која представља име потпрограма;
- Аргументи потпрограму прослеђују се преко стека у обрнутом редоследу од редоследа у декларацији потпрограма, а са стека се уклањају након позива потпрограма (чишћење стека од аргумената врши позивалац, а не позвани);
- Тип *int* је ширине 16 бита, а сви показивачи су ширине 16 бита и адресибилна јединица је ширине 16 бита.
- Стек расте од виших ка нижим локацијама, а регистар *SP* указује на последњу заузету локацију.