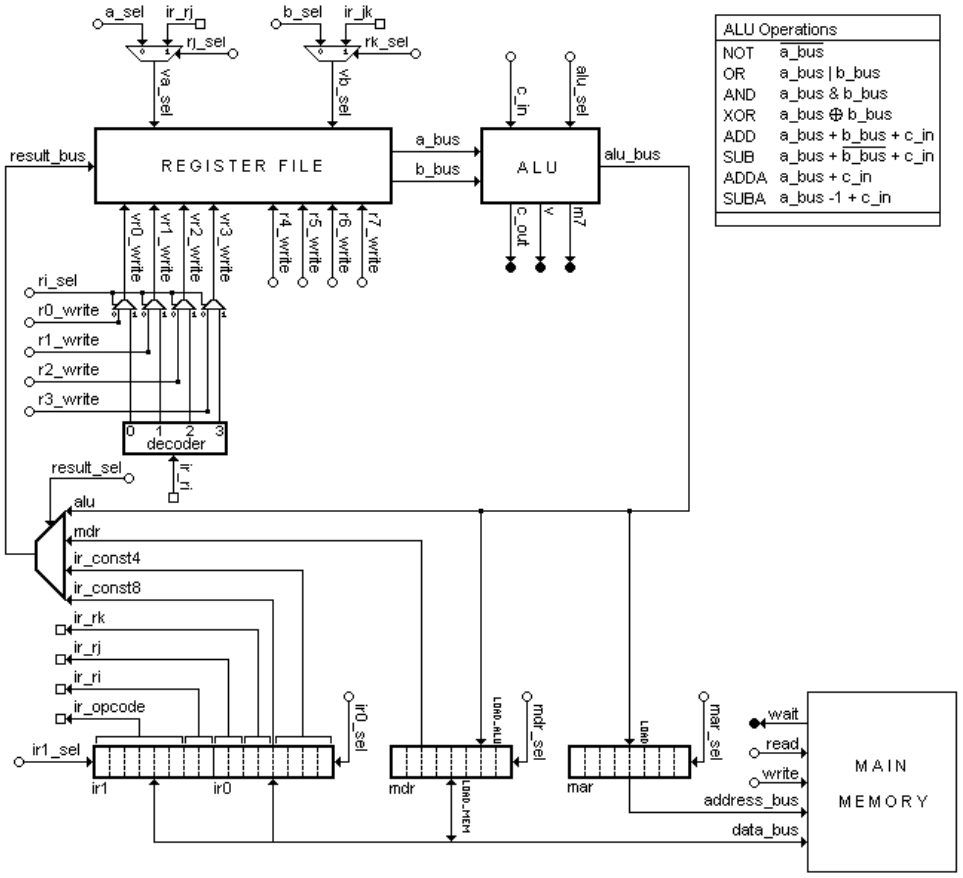
21. studenog 2022.

Zavod za elektroniku, mikroelektroniku,
računalne i inteligentne sustave

Arhitektura računala 2

Međuispit

1. Razmatramo model mikroprogramskog procesora gdje je `r7` programsko brojilo, a `r6` pokazivač stoga. Napišite mikrokod instrukcije `INC (rj), rk` koja uvećava za 2 podatke koji se nalaze u memoriji i počinju od adrese zapisane u `rj`. Broj podataka zapisan je u `rk`. Neka je operacijski kod instrukcije 000010. Bilo koji se registar može koristititi kao pomoćni, no potrebno je očuvati kontekst korištenjem stoga. Napišite program u memoriji koji će uvećati za dva niz [1, 10] koji je pohranjen u memoriji na adresi 100 nadalje. pretpostavite da su vam dostupne funkcije `SETSP (000011)` i `SET ri (000100)` koje postavljaju pokazivač stoga odnosno `ri` na 8-bitnu konstantu. Prikažite sadržaj memorije nakon izvođenja programa.



ALU Operations	
NOT	$\overline{\text{a\_bus}}$
OR	a_bus | b_bus
AND	a_bus & b_bus
XOR	a_bus ⊕ b_bus
ADD	a_bus + b_bus + c_in
SUB	a_bus + $\overline{\text{b\_bus}}$ + c_in
ADDA	a_bus + c_in
SUBA	a_bus -1 + c_in

Memory Interface		
result_sel=	**ALU**	Place the **alu_bus** value on the **result_bus**.
	MDR	Place the mdr value on the **result_bus**.
	IR_CONST4	Place **ir_const4** on the **result_bus**. (**ir_const4** = last 4 bits of ir0)
	IR_CONST8	Place **ir_const8** on the **result_bus**. (**ir_const8** = all 8 bits of ir0)
ir0_sel=	**LOAD**	Load the value on the **memory_bus** into ir0 (Instruction Register 0).
ir1_sel=	**LOAD**	Load the value on the **memory_bus** into ir1 (Instruction Register 1).
mar_sel=	**LOAD**	Load the value on the **alu_bus** into mar (Memory Address Register).
mdr_sel=	**LOAD_ALU**	Load the value on the **alu_bus** into mdr (Memory Data Register).
	LOAD_MEM	Load the value on the **memory_bus** into mdr (Memory Data Register).
read		Place the value at the main memory address mar on the **memory_bus**.
write		Write the value of mdr to the main memory location mar.

2. Razmatramo priručnu memoriju s linijama od 16B, ukupnom veličinom 128B, 2× asocijativnim preslikavanjem te 8-bitnim adresama. Zadan je sljedeći slijed adresiranja početno prazne priručne memorije: 0xd4, 0x59, 0xc6, 0xd6, 0x1d, 0x9f, 0x0e, 0x98, 0xd8, 0x1c.
   (a) Za svako adresiranje navesti radi li se o pogotku (H), promašaju (M) ili zamjeni (R).
   (b) Za svaku zamjenu odrediti je li uzrokovana nedovoljnim kapacitetom ili nedovoljnom asocijativnošću.
   (c) Odrediti prosječno vrijeme pristupa ako je vrijeme pogotka $1\Delta T$ a vrijeme promašaja $100\Delta T$.

3. Dana je funkcija `find_max(int arr[], int n)` koja pronalazi najveći element u vektoru cijelih brojeva `arr` veličine `n`, te njezin strojni prijevod. Povežite svaki redak te funkcije s odgovarajućim odsječkom strojnog koda.

Kod u C-u:

```
1   int find_max(int* arr, int n)
2   {
3       int i, max = arr[0];
4       for (i =1 ; i < n ; i++){
5           if (arr[i] > max){
6               max = arr[i];
7           }
8       }
9       return max;
10  }
```

Strojni kod:

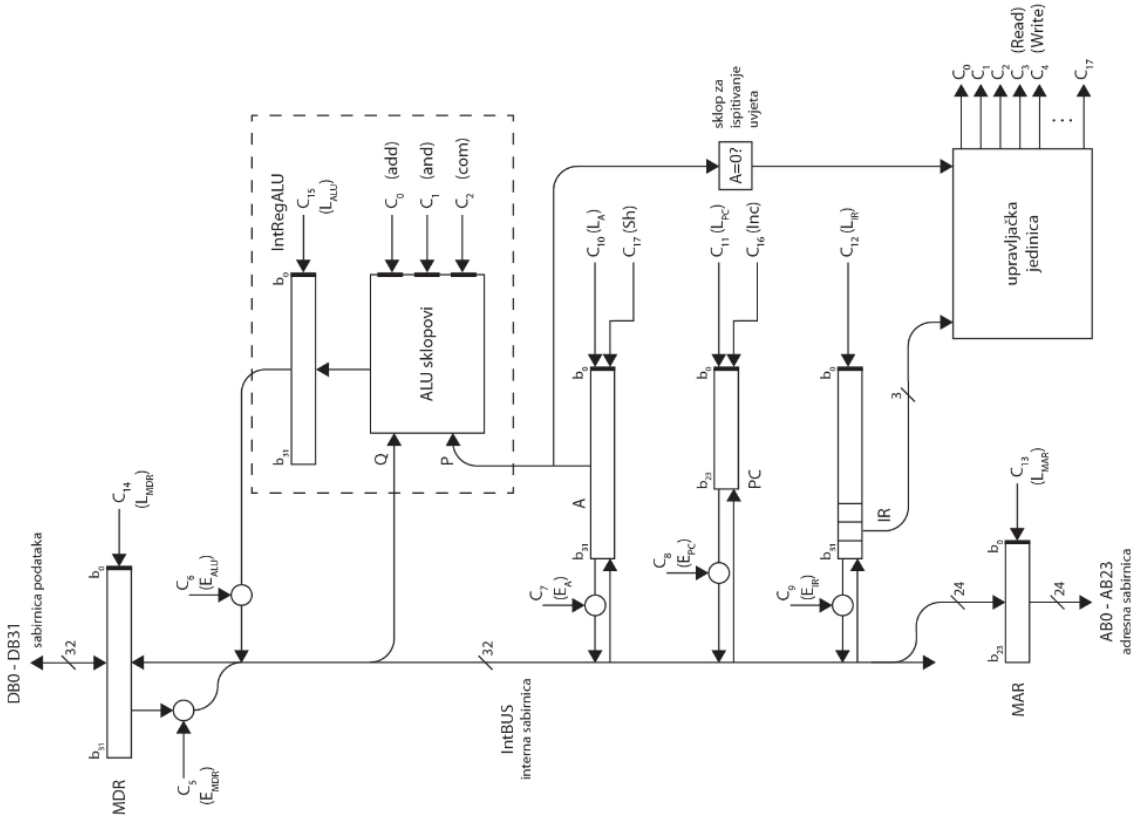
```
1     push ebp
2     mov ebp, esp
3     sub esp, 16
4     mov eax, DWORD PTR [ebp+8]
5     mov eax, DWORD PTR [eax]
6     mov DWORD PTR [ebp-8], eax
7     mov DWORD PTR [ebp-4], 1
8     jmp .L2
9   .L4:
10    mov eax, DWORD PTR [ebp-4]
11    lea edx, [0+eax*4]
12    mov eax, DWORD PTR [ebp+8]
13    add eax, edx
14    mov eax, DWORD PTR [eax]
15    cmp DWORD PTR [ebp-8], eax
16    jge .L3
17    mov eax, DWORD PTR [ebp-4]
18    lea edx, [0+eax*4]
19    mov eax, DWORD PTR [ebp+8]
20    add eax, edx
21    mov eax, DWORD PTR [eax]
22    mov DWORD PTR [ebp-8], eax
23  .L3:
24    add DWORD PTR [ebp-4], 1
25  .L2:
26    mov eax, DWORD PTR [ebp-4]
27    cmp eax, DWORD PTR [ebp+12]
28    jl .L4
29    mov eax, DWORD PTR [ebp-8]
30    leave
31    ret
```

4. Na vanjsku sabirnicu pojednostavnjenog CISC procesora spojen je ulazno-izlazni uređaj koji ima 2 8-bitna registra: (i) registar stanja i (ii) podatkovni registar. Registrima ulazno-izlaznih uređaja pristupamo jednako kao i memorijskim lokacijama ("memorijsko U/I preslikavanje"). Registar stanja javlja se na adresi $F000, a podatkovni registar na adresi $F001. Neka se u registru stanja početno nalazi vrijednost $C5, a u podatkovnom registru $41.

   Napišite program koji čita podatak iz podatkovnog registra ulazno-izlaznog uređaja, te ako je podatak različit od 0, upisuje podatak u memoriju na adresu $A000. Početna adresa programa neka je $1000. Na raspolaganju su instrukcije: LDA adr (op. kod $B6, apsolutno adresiranje), BNE (op. kod $26, relativno adresiranje), BEQ (op. kod $27, relativno adresiranje) te STA adr (op. kod $B7, apsolutno adresiranje).

   - Skicirajte sadržaj dijela memorije u kojem se nalazi program;
   - nacrtajte stanje na vanjskim sabirnicama tijekom izvođenja programa, te
   - odredite konačne sadržaje svih registara procesora koje je moguće odrediti.

5. Za 8-instrukcijski procesor potrebno je izvesti instrukciju LDAR X kod koje se efektivna adresa memorijskog operanda tvori kao PC+X. Rješenje treba opisati i) promjene puta podataka te ii) izmjene upravljačke jedinice.



```
// ====================== PRIBAVI ======================

fetch0: a_sel=7, b_sel=7, alu_sel=OR, mar_sel=LOAD;             // MAR <- PC
fetch1: ir1_sel=LOAD, read, if wait then goto fetch1 endif;   // IR_high <- MEM(MAR)
fetch2: a_sel=7, c_in, alu_sel=ADDA, r7_write;          // PC <- PC+1
fetch3: a_sel=7, b_sel=7, alu_sel=OR, mar_sel=LOAD;             // MAR <- PC
fetch4: ir0_sel=LOAD, read, if wait then goto fetch4 endif; // IR_low <- MEM(MAR)
fetch5: a_sel=7, c_in, alu_sel=ADDA, r7_write, goto opcode[IR_OPCODE]; // PC <- PC+1



// ============= DIO OPERACIJSKIH KODOVA =============
//SETR4
opcode[0]: r4_write, result_sel=IR_CONST8, goto fetch0; //IR_CONST8 -> r4

//SETR5
opcode[1]: r5_write, result_sel=IR_CONST8, goto fetch0; //IR_CONST8 -> r5

//INC (rj), rk
opcode[2]: a_sel=6, alu_sel=SUBA, r6_write goto opcode2.1; // r6--

//SETSP konst
opcode[3]: r6_write, result_sel=IR_CONST8, goto fetch0; //IR_CONST8 -> ri

//SET ri, konst
opcode[4]: ri_sel, result_sel=IR_CONST8, goto fetch0; //IR_CONST8 -> ri


// ================ DIO EKSTENZIJE ================


opcode2.1: a_sel=6, alu_sel=ADDA, mar_sel=LOAD; // r6 -> MAR
opcode2.2: a_sel=4, alu_sel=ADDA, mdr_sel=LOAD_ALU; // r4 -> MDR
opcode2.3: write, if wait then goto opcode2.3 endif; // push r4
opcode2.4: a_sel=6, alu_sel=SUBA, r6_write; // r6--
opcode2.5: a_sel=6, alu_sel=ADDA, mar_sel=LOAD; // r6 -> MAR
opcode2.6: a_sel=5, alu_sel=ADDA, mdr_sel=LOAD_ALU; // r5 -> MDR
opcode2.7: write, if wait then goto opcode2.7 endif; // push r5

opcode2.8: a_sel=4, b_sel=4, alu_sel=XOR, r4_write; // r4 <- 0
opcode2.9: a_sel=4, rk_sel, alu_sel=SUB, c_in if m_7 then goto opcode2.10 else goto opcode2.17 e
opcode2.10: rj_sel, b_sel=4, alu_sel=ADD, mar_sel=LOAD; // rj + r4 -> MAR
opcode2.11: mdr_sel=LOAD_MEM, read, if wait then goto opcode2.11 endif; // MEM[rj+r4] -> MDR
opcode2.12: result_sel=MDR, r5_write; // MDR -> r5
opcode2.13: a_sel=5, alu_sel=ADDA, c_in, r5_write; // r5++
opcode2.14: a_sel=5, alu_sel=ADDA, c_in, mdr_sel=LOAD_ALU; // r5++ -> MDR
opcode2.15: write, if wait then goto opcode2.15 endif; // MDR -> MEM[rj + r4]
opcode2.16: a_sel=4, alu_sel=ADDA, c_in, r4_write, goto opcode2.9; //r4++

opcode2.17: a_sel=6, alu_sel=ADDA, mar_sel=LOAD; // r6 -> MAR
opcode2.18: mdr_sel=LOAD_MEM, read, if wait then goto opcode2.18 endif; // pop r5
opcode2.19: result_sel=MDR, r5_write; // MDR -> r5
opcode2.20: a_sel=6, alu_sel=ADDA, c_in, r6_write; // r6++
opcode2.21: a_sel=6, alu_sel=ADDA, mar_sel=LOAD; // r6 -> MAR
opcode2.22: mdr_sel=LOAD_MEM, read, if wait then goto opcode2.22 endif; // pop r4
opcode2.23: result_sel=MDR, r4_write; // MDR -> r4
opcode2.24: a_sel=6, alu_sel=ADDA, c_in, r6_write, goto fetch0; // r6++


// SETR4 4
0: 000000 00
1: 4

// SETR5 5
2: 000001 00
3: 5

// SETSP 255
4: 000011 00
5: 255
```

```
// SET r0, 100
6: 000100 00
7: 100

// SET r1, 10
8: 000100 01
9: 10

// INC (r0), r1
10: 000010 00
11: 00 01 0000


100: 1
101: 2
102: 3
103: 4
104: 5
105: 6
106: 7
107: 8
108: 9
109: 10
```

ZAD 5/ LDAR X

- dodati još jedan bit za kod instrukcije
  ⇒ 4 žice u upravljačkoj jedinici
- dodati signal C_{18} - postavlja gornjih 8 bitova akumulatora na 0

A ← IR [23:0] : $C_9 : (\phi_8 + \phi_9) \cdot I_9$
$C_{10} : (\phi_9) \cdot I_9$
$C_{18} : (\phi_9) \cdot I_9$

PC → A : $C_8 : (\phi_{10} + \phi_{11}) \cdot I_9$
$C_0 : (\phi_{10} + \phi_{11}) \cdot I_9$
$C_{15} : \phi_{11} \cdot I_9$

MAR ← PC+A : $C_6 = (\phi_{12} + \phi_{13}) \cdot I_9$
$C_{13} = \phi_{13} \cdot I_9$

MDR ← MEM [MAR] : $C_3 : (\phi_{14} + \phi_{15}) \cdot I_9$
$C_{14} : \phi_{15} \cdot I_9$

A ← MDR : $C_5 : (\phi_{16} + \phi_{17}) \cdot I_9$
$C_{10} : \phi_{17} \cdot I_9$

ZAD 2

linija 16 B
veličina 128 B
⇒ broj linija = $\frac{128}{16} = 8$

2x asocijativno preslikavanje ⇒ 4 bucketa

	c0	cf
0	d0 9d 10	df 9f 1f
1	50 10 d0	5f 1f df
2		
3		

	index	bucket	tag	
d4	11	01	0100	MISS
59	01	01	1001	MISS
c6	11	00	0110	MISS
d6	11	01	0110	HIT
1d	00	01	1101	MISS/SWAP (A)
9f	10	01	1111	MISS/SWAP (A)
0c	00	00	1110	MISS
98	10	01	1000	HIT
d8	11	01	1000	MISS/SWAP (A)
1c	00	01	1110	MISS/SWAP (A)

ZAD 4

```
LDA   $F001
BEQ   $03
STA   $A000
```

```
1000 : B6
1001 : F0
1002 : 01
1003 : 27
1004 : 03
1005 : B7
1006 : A0
1007 : 00
1008 : ...
```

A	1000	1001	1002	F001	1003	1004	XX	1005	1006	1007	A000
R							XX				
W							XX				
D	B6	F0	01	41	27	03	XX	B7	A0	00	41
A	XX	XX	XX	41	41	41	41	41	41	41	41
PC	1001	1002	1003	1003	1004	1005	1005	1006	1007	1008	1008
IR	B6	B6	B6	B6	27	27	27	B7	B7	B7	B7
DC	XX	F0XX	F001	F001	F001	F001	F001	F001	A001	A000	A000
PR						03	03	03	03	03	03

PRIBAVI | IZVRŠI | PRIBAVI | IZVRŠI | PRIBAVI | IZVRŠI