Mikroutasítások

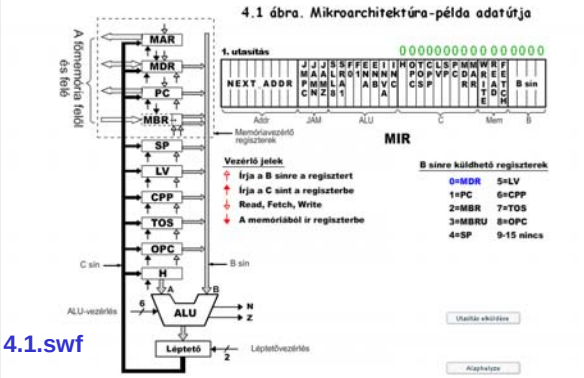
24 bit: az adatút vezérléséhez

9 bit: a következő utasítás címének megadásához,
3 bit: a következő utasítás kiválasztásának módjára.
Ez adja a 36 bites mikroutasítást: **4.5. ábra**.

9	3	8	9	3	4
NEXT ADDRESS	JMPC JAMN JAMZ	SLL8 SRA1 F0 F1 ENA ENB INVA INC	H OPC TOS LV SP PC MDR MAR	WRITE READ FETCH	B sín
Addr	JAM	ALU	C	Mem	B

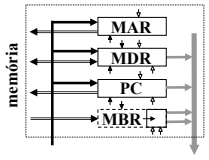
0 = MDR	1 = PC	2 = MBR	3 = MBRU	4 = SP
5 = LV	6 = CPP	7 = TOS	8 = OPC	9-15 semmi

SP ⇒ B sín; ALU ⇒ C; C ⇒ SP



4.1.swf

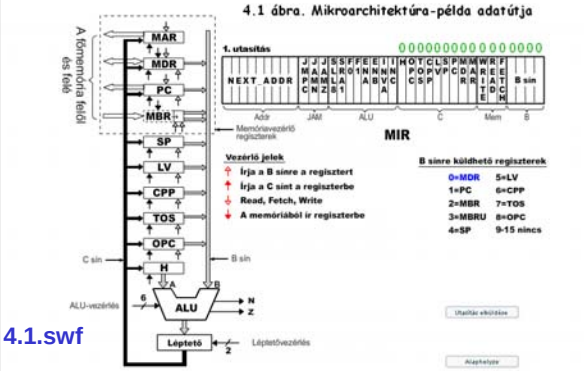
Memória ciklus



A memória ciklus az adatút végén kezdődik (a **C** sínről a regiszterek feltöltése után), ezért ha a memória címet módosította ez a mikroutasítás, akkor a memória cím a módosított **MAR** ill. **PC** regiszter értéke lesz.

Az olvasás eredménye csak két ciklussal később használható az **ALU**-ban, mert **MDR** ill. **MBR** csak a következő adatút ciklus vége felé töltődik fel a memóriából, addig **MDR** ill. **MBR** régi értéke érhető el.

Memória ciklus: MAR = LV; rd; TOS = MDR



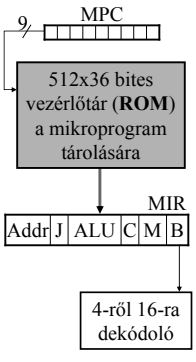
4.1.swf

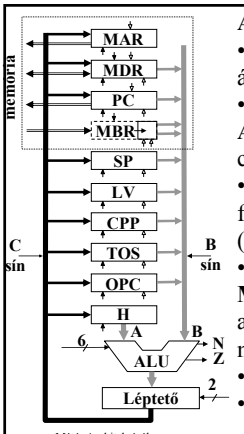
A mikroprogram

Mic-1: 4.6. ábra.

- 512x36 bites vezérlőtár a mikroprogramnak,
- **MPC** (MicroProgram Counter): mikroprogram-utasításszámláló.
- **MIR** (MicroInstruction Register): mikroutasítás-regiszter.

Az adatút ciklus **(4.6. ábra)** elején **MIR** feltöltődik a vezérlőtár **MPC** által mutatott szavával.

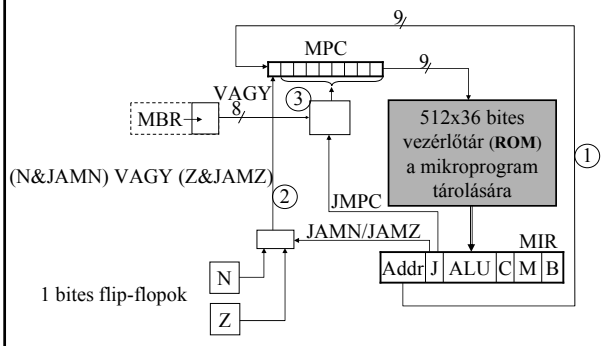




Adatút ciklus (4.6. ábra):

- (**MIR** feltöltődik a vezérlőtár **MPC** által mutatott szavával.)
- Kialakul a **B** sín kívánt tartalma, **ALU** és a **léptető** megtudja, mit kell csinálnia,
- Az **ALU** és a **léptető** elvégzi a feladatát, a **C** sín, **N** (Negative) és **Z** (Zero) megkapja az új értékét,
- A regiszterek feltöltődnek a **C** sínről. **MBR**/**MDR** megkapja az értékét, ha az előző ciklus adatot kért a memóriából.
- Kialakul **MPC** új értéke.
- Memória ciklus kezdete.

MPC új tartalmának kialakítása



Mic-1: 4.6. ábra.

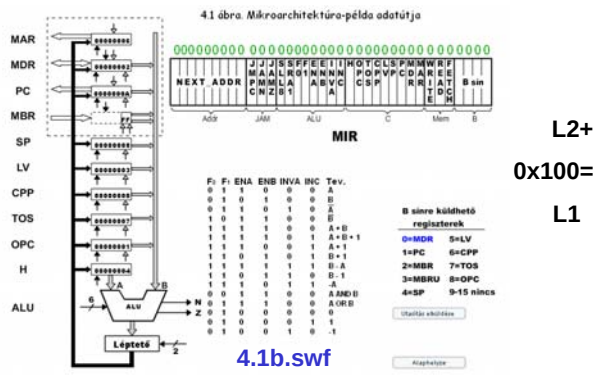
MPC új tartalma, JAMN/JAMZ

- A 9 bites következő cím (**Addr**) az **MPC**-be kerül.
- **(JAMN *ÉS* N) *VAGY* (JAMZ *ÉS* Z)** és **MPC** legmagasabb bitjének logikai vagy kapcsolata képződik **MPC** legmagasabb helyértékén. Pl.:

Cím	Addr	JAM	Adatút vezérlő bitek	
0x75	0x092	001	. . .	JAMZ =1

esetén a mikroprogram a
0x092 címen folytatódik, ha **Z = 0**,
0x192 címen folytatódik, ha **Z = 1**.
Feltételes ugrás – elágazás – a **mikroprogramban**.

JAMN: if(LV-H < 0) goto L1; else goto L2



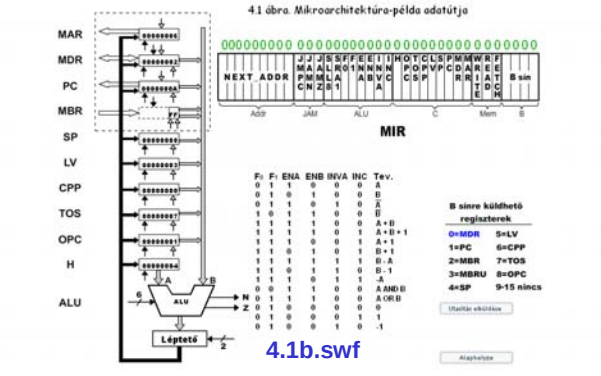
L2+
0x100=
L1

MPC új tartalma, JMPC

- **JMPC** esetén **MPC** 8 alacsonyabb helyértékű bitjének és **MBR** 8 bitjének bitenkénti vagy kapcsolata képződik **MPC**-ben az adatút ciklus vége felé (**MBR** megérkezése után).
  Ilyenkor **Addr** 8 alacsonyabb helyértékű bitje általában **0**
  Feltétlen ugrás az **MBR**-ben tárolt címre – kapcsoló utasítás:
  goto(MBR) vagy **goto(MBR *OR* 0x100)**

Kezdődhet az újabb mikroutasítás végrehajtása.

JMPC: TOS=H+SP+1; goto(MBR)



Mic-1 működése

•	(**MPC**) ⇒ **MIR**
• regiszter ⇒ **B** sín, **ALU**, léptető megtudja, mit kell csináljon,	**Addr** ⇒ **MPC**
• eredmény ⇒ **C**, **N**, **Z**	
• **C** ⇒ regiszterekbe mem. ⇒ **MDR** és/vagy mem. ⇒ **MBR**	**JAMN**, **JAMZ** és **(N)**, **(Z)** alapján módosul **MPC**
• Memória ciklus indítása (**rd**, **wr**, **fetch**)	**JMPC** és **(MBR)** alapján módosul **MPC**.

Mic-1 programozása (4.5, 6. ábra)

36 bites bináris utasításokat kellene megadnunk.

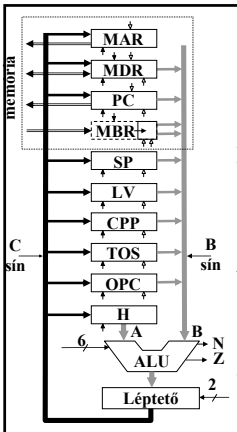
Pl.: Egy ciklusban növeljük **SP**-t 1-gyel és olvasást kezdeményezünk a memóriából, folytatás a 122-es utasításnál. Szimbolikusan ilyesmi:

ReadRegister = SP, ALU = INC, Write SP, Read, NextAddress = 122;

Nehézkes, helyette:

SP = SP + 1; rd

A folytatás címet csak akkor tüntetjük fel, ha az nem a következőként írt mikroutasítás (pl. **goto Main1**).



MAL (Micro Assembly Language)

SOURCE: a **B** sínre kötött regiszterek bármelyike: **MDR**, **PC**, **MBRU** (előjel nélküli - **MBR U**nsigned), **MBR**, **SP**, **LV**, **CPP**, **TOS**, **OPC**.

DEST: a **C** sínre kapcsolt regiszterek bármelyike: **MAR**, **MDR**, **PC**, **SP**, **LV**, **CPP**, **TOS**, **OPC**, **H**. Több regiszter is kaphatja ugyanazt az értéket.

wr: memóriába írás **MDR**-ből a **MAR** címre.

rd: memóriából olvasás **MDR**-be a **MAR** címről.

fetch: 8 bites utasításkód betöltése **MBR**-be a **PC** címről.

Nem megengedett pl. az alábbi utasítás pár:

MAR = SP; rd
MDR = H // A memóriából is most kapna értéket!

Feltétlen ugrás:

goto Main1

Az **Addr** mezőbe **Main1** címét kell írni.

Feltétlen ugrás MBR szerint (kapcsoló utasítás):
Ilyenkor **JMPC = 1**

goto (MBR *OR value*)

value általában **0** vagy **0x100**.

Feltételes elágazás, pl.: **TOS** (Top Of Stack) alapján
Z = TOS ; if (Z) goto L1; else goto L2
// **Z=1**, ha **TOS=0**, különben **Z=0**.

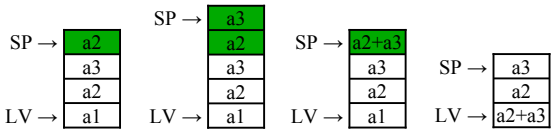
Cím	Addr	JAM	Adatút vezérlő bitek	
0x75	0x092	001	. . .	JAMZ =1

esetén a mikroprogram az
L2 **0x092** címen folytatódik, ha **Z = 0**,
L1 **0x192** címen folytatódik, ha **Z = 1**.

Az **L1** és **L2** címek különbsége **256** (**0x100**) kell legyen (**4.7. ábra**)!

A verem operandusok és az eredmény ideiglenes tárolására is használható (operandus verem), pl. (**4.9. ábra**):

a1 = a2 + a3



IJVM (Integer Java Virtual Machine): a **JVM** egész értékű aritmetikát tartalmazó része.

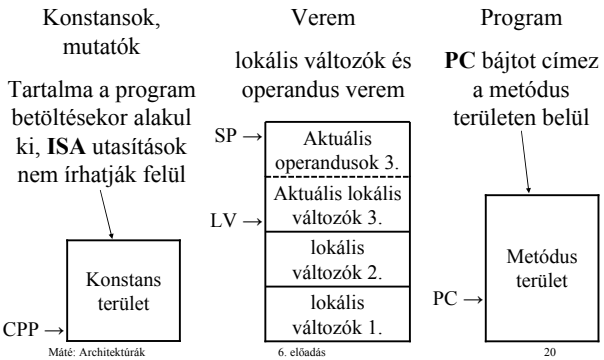
Az IJVM utasítások szerkezete:

- az első mező az **opcode** (Operation Code, műveleti kód),
- az esetleges második mezőben az **operandus** meghatározására szolgáló adat van.

Mikroprogram: betölti, értelmezi és végrehajtja az **IJVM** utasításokat:
betöltés-végrehajtás (fetch-execute) ciklus.

Az IJVM memóriamodellje (4.10. ábra)

A **4 GB** memória, **1 G** szóként is szervezhető.



IJVM néhány utasítása: **4.11. ábra** (részlet).

hex	Mnemonic	jelentés
10	BIPUSH *byte*	Beteszi a *byte*–ot a verembe
A7	GOTO *offset*	Feltétel nélküli ugrás *offset* -re
60	IADD	Kivesz a veremből két szót, az összegüket a verembe teszi
99	IFEQ *offset*	Kivesz a veremből egy szót, ha 0, akkor *offset* -re ugrik
9F	IF_ICMPEQ *offset*	Kivesz a veremből két szót, ha egyenlők, akkor *offset* -re ugrik
15	ILOAD *varnum*	Beteszi *varnum* -ot a verembe
36	ISTORE *varnum*	Kivesz a veremből egy szót, és eltárolja *varnum* -ba
64	ISUB	Kivesz a veremből két szót, a különbségüket a verembe teszi
00	NOP	Nem csinál semmit
5F	SWAP	A verem két felső szavát megcseréli

Java (C) program	IJVM program 4.14. ábra			Bin. kód
	1	ILOAD j	// i = j + k	15 02
	2	ILOAD k		15 03
i = j + k;	3	IADD		60
if(i = = 3)	4	ISTORE i		36 01
k = 0;	5	ILOAD i	// if(i = = 3)	15 01
else	6	BIPUSH 3		10 03
j = j – 1;	7	IF_ICMPEQ L1		9F 00 0D
	8	ILOAD j	// j = j – 1	15 02
	9	BIPUSH 1		10 01
	10	ISUB		64
	11	ISTORE j		36 02
	12	GOTO L2		A7 00 0F
	13 L1:	BIPUSH 0	// k = 0	10 00
	14	ISTORE k		36 03
	15 L2:			

IJVM megvalósítása Mic-1-en (4.11., 17. ábra)

Előkészület a gép indításakor: **PC** a végrehajtandó utasítás címét, **MBR** az utasítás kódját tartalmazza.

A főciklus legelső mikroutasítása a **Main1**, ez:

PC=PC+1; fetch; goto(MBR);

PC most a végrehajtandó utasítás utáni bájtra mutat, ez lehet egy újabb utasítás kódja, vagy operandus.

PC új értékének kialakulása után indul a **fetch**-csel kezdeményezett memória ciklus, ez a program következő bájtját olvassa **MBR**-be (a következő mikroutasítás végén lesz **MBR**-ben a bájt).

goto (MBR) elugrik az utasítás feldolgozásához.

Minden utasítás feldolgozását végző függvény első mikroutasítása az utasítás kódnak megfelelő címen van a mikroprogram tárban.

A mikroutasítások elhelyezkedése a mikroprogram tárban (részlet)

	0 8	1 9	2 A	3 B	4 C	5 D	6 E	7 F
0	NOP1	IAND3	POP3	SWAP2	SWAP3	SWAP4	SWAP5	SWAP6
8	LDC_W3	LDC_W4	IINC3	IINC4	IINC5	IINC6	IFLT2	IFLT3
10	BIPUSH1	BIPUSH2	BIPUSH2	LDC_W1	LDC_W2	ILOAD1	ILOAD2	ILOAD3
18	ILOAD4	ILOAD5	IFLT4	INVOKEV19	INVOKEV20	INVOKEV21	INVOKEV22	INVOKEV23
20	F	F2						
28								
30							ISTORE1	ISTORE2
38	ISTORE3	ISTORE4	ISTORE5	ISTORE6				
40								
48								
50								POP1
58	POP2	DUP1	DUP2					SWAP1
60	IADD1	IADD2	IADD3		ISUB1	ISUB2	ISUB3	
68								
70								
78							IAND1	IAND2
80	IOR1	IOR2	IOR3		IINC1	IINC2	INVOKEV1	INVOKEV2
88	INVOKEV3	INVOKEV4	INVOKEV5	INVOKEV6	INVOKEV7	INVOKEV8	INVOKEV9	INVOKEV10

Látható, hogy nem helyezhetjük egymás után az egyes utasítások feldolgozását végző mikroutasítás sorozatot, ezért inkább azt a megoldást választottuk, hogy minden mikroutasítás tartalmazza a következő címét.

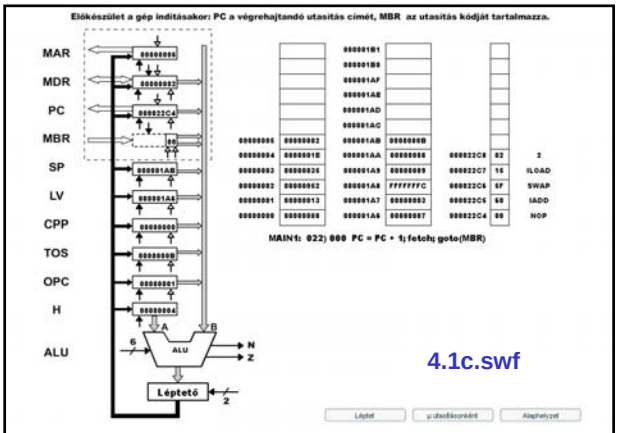
Ha az első utasítás pl. **NOP** (No OPeration, nem csinál semmit), ennek a kódja **0x00**, ezért a **0x00** címen kezdődik a **NOP** feldolgozását végző függvény. Ez egyetlen **goto Main1** mikroutasítás.

IJVM megvalósítása Mic-1-en (4.11., 17. ábra)

A főciklus a **Main1**-nél kezdődik;
PC a végrehajtandó utasítás címét,
MBR az utasítás kódját tartalmazza.

Main1 a következő utasítást vagy adatbájtot olvassa.

Címke	Műveletek // kommentár
Main1	PC = PC + 1; fetch; goto(MBR)
nop1	goto Main1
iadd1	MAR = SP = SP – 1; rd
iadd2	H = TOS
iadd3	MDR = TOS = MDR + H; wr; goto Main1



A verem két felső szavának cseréje (4.17. ábra)

Megállapodás szerint **TOS** tartalmazza a verem tetején lévő szót! Ez többnyire előny.

SP → A
B

swap1	MAR = SP – 1; rd // A 2. szó címe, olvasás	MAR → B
swap2	MAR = SP // MAR a verem tetejére mutat	MAR → A MDR ⇐ B

A verem két felső szavának cseréje (4.17. ábra)

swap1	MAR = SP – 1; rd // A 2. szó címe, olvasás	MAR → B
swap2	MAR = SP // MAR a verem tetejére mutat	MAR → A MDR ⇐ B
swap3	H = MDR; wr // 2. szó H-ba, verem tetejére	H = B MDR ⇒ (MAR)
swap4	MDR = TOS // verem régi teteje	MDR = A

SP → B
B

swap4-ben előny, hogy **TOS** tartalmazza a verem tetején lévő szót.

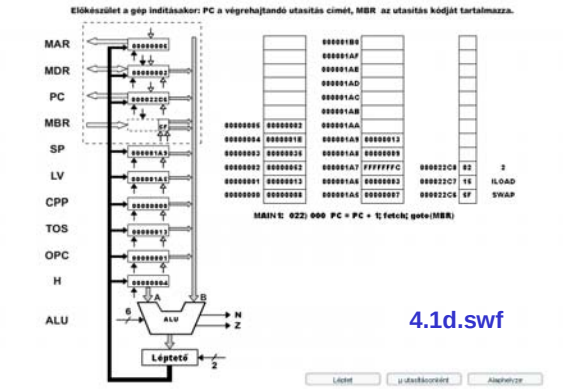
A verem két felső szavának cseréje (4.17. ábra)

swap1	MAR = SP – 1; rd // A 2. szó címe, olvasás	MAR → B
swap2	MAR = SP // MAR a verem tetejére mutat	MAR → A MDR ⇐ B
swap3	H = MDR; wr // 2. szó H-ba, verem tetejére	H = B MDR ⇒ (MAR)
swap4	MDR = TOS // verem régi teteje	MDR = A
swap5	MAR = SP – 1; wr // a 2. szóba	MAR → B MDR ⇒ (MAR)
swap6	TOS = H; goto Main1 // TOS frissítése	

swap6-ban hátrány, mert ez az utasítás csak azért kell, hogy **TOS** tartalmazza a verem tetején lévő szót.

SP → B
A

A verem két felső szavának cseréje (4.17. ábra)



A WIDE utasítás

A **WIDE** utasítás valójában prefixum: önmagában nem csinál semmit, csak jelzi, hogy a következő utasításnak 16 bites indexe van. Pl.:

ILOAD *varnum* lokális változó a verembe
varnum a lokális változó 8 bites indexe.

WIDE

ILOAD *varnum* lokális változó a verembe
varnum a lokális változó 16 bites indexe.

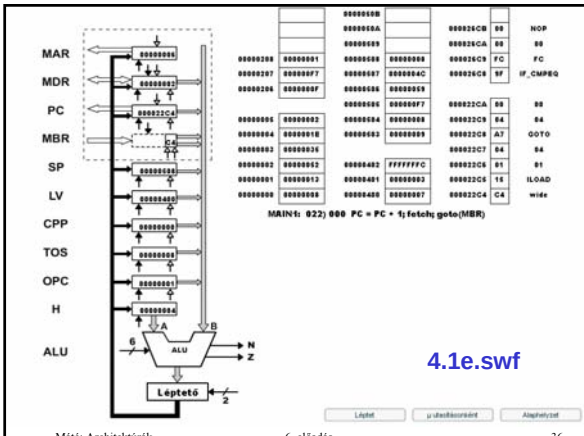
w_iload1 címe = **iload1** címe + **0x100**

ILOAD *varnum* lokális változó a verembe
varnum a lokális változó 8 bites indexe.

Main1	PC = PC + 1; fetch; goto(MBR)	MBR = ILOAD
iload1	H = LV	MBR ⇐ *varnum*
iload2	MAR = H + MBRU; rd // rd(LV+*varnum*)	
iload3	MAR = SP = SP + 1	MDR ⇐ (MAR)
iload4	PC = PC + 1; fetch; wr	(MAR) ⇐ MDR
iload5	TOS = MDR; goto Main1	MBR ⇐ opkód

WIDE
ILOAD *varnum* lokális változó a verembe
varnum a lokális változó 16 bites indexe.

Main1	PC = PC + 1; fetch; goto(MBR)	MBR = WIDE
iload1	H = LV	
iload2	MAR = H + MBRU; rd // rd(LV+*varnum*)	
iload3	MAR = SP = SP + 1	MDR ⇐ (MAR)
iload4	PC = PC + 1; fetch; wr	(MAR) ⇐ MDR
iload5	TOS = MDR; goto Main1	MBR ⇐ opkód
wide1	PC = PC + 1; fetch; goto(MBR *OR* 0x100)	MBR ⇐ ILOAD
w_iload1	PC = PC + 1; fetch // index 2. bájtja	MBR ⇐ 1. bájt
w_iload2	H = MBRU << 8 // 1. bájt léptetése	MBR ⇐ 2. bájt
w_iload3	H = H *OR* MBRU // H = a 16 bites index	
w_iload4	MAR = LV + H; rd; goto iload3	rd(LV+*varnum*)



Az **GOTO *offset*** utasítás. **PC** relatív: **PC** értékéhez hozzá kell adni a két bájtos, előjeles ***offset*** értékét. **Mic-1** program:

Main1	PC = PC + 1; fetch; goto(MBR)	
goto1	OPC=PC–1 // Main1-nél : PC=PC+1	MBR ⇐ 1. bájt
goto2	PC=PC+1; fetch // ***offset*** 2. bájtja	
goto3	H=MBR<<8 // 1. (előjeles) bájt <<8	MBR ⇐ 2. bájt
goto4	H=MBRU OR H // 16 bites ***offset***	
goto5	PC=OPC+H; fetch; goto Main1	// PC új értéke

Main1	PC = PC + 1; fetch; goto(MBR)	MBR ⇐ opkód

goto5 kezdeményezi a **PC** új értékénél lévő bájt olvasását, **Main1**-ben **goto(MBR)** már a folytatás első opkódjának megfelelő címre ugrik!

Feladatok

Milyen részei vannak az egy bites **ALU**-nak?
Milyen vezérlő bemenetei vannak az **ALU**-nak?
Milyen vezérlő bemenetek esetén lesz **1** az eredmény?
Milyen eredményt szolgáltat az $\mathbf{F_0}$**=0**, $\mathbf{F_1}$**=1**, **ENA=0**, **ENB=0**, **INVA=1**, **INC=1** vezérlő bemenet?
A **Mic-1** mely regisztere lehet az **ALU** bal/jobb operandusa?
Hova tárolhatja a **Mic-1** az eredményt?
Érvényes utasítás-e **Mic-1**-en a **H=OPC-H**? Miért?
Érvényes utasítás-e **Mic-1**-en a **H=H-OPC**? Miért?

Feladatok

Milyen utasításai vannak a **Mic-1** gépnek?
Milyen ugró utasításai vannak a **Mic-1** gépnek?
Milyen értékeket vehet föl a **SOURCE** operandus?
Milyen értékeket vehet föl a **DEST** operandus?
Mit jelent a **wr**?
Mely utasítások tudnak olvasni a memóriából, és hogy működnek?
Hogy lehet védekezni az ellen, hogy **MDR** egyszerre kapjon értéket a memóriából és a **C** sínről?
Mi az operandus verem?

Feladatok

Hogy történik a memóriából olvasás?
Hogy történik a memóriába írás?
Mire szolgál a **MAR**/**MDR** regiszter?
Ha egy mikroutasítás módosítja **MAR** tartalmát, és olvas a memóriából, akkor mely címről fog olvasni?
Memóriából olvasás után mikor használható **MDR** új értéke az adatúton illetve **MPC** meghatározásához?
Mire szolgál a **PC** és az **MBR** regiszter?
Mire szolgál az **N** és a **Z** regiszter?
Mire szolgál a **H** regiszter?

Feladatok

Milyen memória műveletei vannak a **Mic-1** -nek?
Milyen jelek szükségesek a **Mic-1** adatútjának vezérléséhez?
Hány jel szolgál az **A** sín vezérlésére?
Hány jel szolgál a **B** sín vezérlésére?
Hány jel szolgál az **ALU** és a léptető vezérlésére?
Hány jel szolgál a **C** sín vezérlésére?
Hány jel szolgál a memória elérésére?
Milyen részei vannak a **Mic-1** mikroutasításainak?
Milyen részei vannak az adatút ciklusnak?

Feladatok

Milyen részei vannak a **Mic-1** mikroutasításainak?
Milyen típusú memória a mikroprogram tároló?
Mire szolgál az **MPC** regiszter?
Mire szolgál az **MIR** regiszter?
Miért van szükség az **Addr** mezőre?
Milyen részei vannak az adatút ciklusnak?
Hány bit kell a **B**/**C** sín vezérléséhez?
Mire szolgál a **JMPN**/**JMPZ** bit?
Mire szolgál a **JMPC** bit?
Hogy alakul ki **MPC** új tartalma?

Feladatok

Miért nem megengedett az

MAR = SP; rd

MDR = H

utasítás pár?

Hogy valósítható meg feltétlen ugrás a mikroprogramban?

Hogy valósítható meg feltételes ugrás a mikroprogramban?

Hogy valósítható meg kapcsoló utasítás a mikroprogramban?

Feladatok

Minek a rövidítése az **IJVM**?

Ismertesse az **IJVM** memóriamodelljét!

Milyen utasításai vannak az **IJVM**-nek?

Mi a **BIPUSH/DUP/IADD/SWAP** utasítás feladata?

Mi a **GOTO** utasítás feladata?

Mi a **IFEQ/IF_ICMPEQ** utasítás feladata?

Hogy működik a **Mic-1** főciklusa?

Mit tartalmaz **PC** és **MBR** a főciklus indulásakor?

Hogy valósítható meg **Mic-1**-en a **NOP** utasítás?

Hogy valósítható meg **Mic-1**-en az **IADD** utasítás?

Feladatok

Mire szolgál a **SWAP** utasítás?

Hogy valósítható meg a **SWAP** utasítás?

Mire szolgál a **WIDE** utasítás?

Hogy valósítható meg a **WIDE** utasítás?

Mire szolgál az **ILOAD** utasítás?

Hogy valósítható meg az **ILOAD** utasítás?

Mire szolgál a **WIDE ILOAD** utasítás?

Hogy valósítható meg a **WIDE ILOAD** utasítás?

Mire szolgál a **GOTO** utasítás?

Hogy valósítható meg a **GOTO** utasítás?

Az előadáshoz kapcsolódó

Fontosabb tételek

A Mic-1 működése, adatút ciklusa, memória ciklusa, mikroprogramja.

MPC új értékének kialakulása Mic-1-en.

Az IJVM, az IJVM memória modellje, az IJVM megvalósítása Mic-1-en

A WIDE utasítás hatása és működése Mic-1-en