



Digitális Rendszerek és Számítógép Architektúrák (VEMKKN3214A)

5. előadás: Vezérlő egységek: huzalozott,
mikroprogramozott módszerek

Előadó: Dr. Vörösházi Zsolt
voroshazi.zsolt@virt.uni-pannon.hu

Jegyzetek, segédanyagok:

- Könyvfejezetek:

- <http://www.virt.uni-pannon.hu>

- Oktatás → Tantárgyak → Digitális Rendszerek és Számítógép Architektúrák (Nappali)
(chapter05.pdf)

- FPGA-s rész fóliákon

- Fóliák, óravázlatok .ppt (.pdf)

- Feltöltsük folyamatosan

Vezérlő egységek általánosan

- A számítógép vezérlési funkcióit ellátó **szekvenciális** egység.
- **Feladata:** az operatív tárban lévő gépi kódú utasítások értelmezése, részműveletekre bontása, és a szekvenciális (sorrendi) hálózat egyes funkcionális részeinek vezérlése (vezérlőjel- és cím-generálás)
- Vezérlő egység tervezésének lépései:
 - megfelelő technológia, és rendszerkomponensek kiválasztása
 - komponensek összekapcsolása a működési sorrendnek megfelelően
 - RTL leírás alkalmazása az akciók ill. adatátvitel pontos leírására
 - **adatút (data-path)** megtervezése (*legfontosabb!*)
 - kívánt vezérlő jelek azonosítása, meghatározása

Adatút (Data-path) tervezés szempontjai

- Gazdaságosság (költség)
- Interfész szükséglet (protokollok)
- Sebesség (S)
- Felület (A)
- Energia (disszipált teljesítmény) (P, D)
- Dinamikai tartomány (számrendszerek)
- Rugalmasság (többcélúság)
- Kezelhetőség (probléma, hiba során)
- Környezet (pl. ipari v. irodai használat?)

Vezérlő egységek fajtái:

- I. **Huzalozott (klasszikus)** módszerek (pl. korai RISC architektúrák):
 - Mealy-modell,
 - Moore-modell (Példa: FIR szűrő tervezése – FSM állapotgép segítségével).
 - Multiplexeres / késleltetéses / Shift-regiszteres megvalósítások
- II. **Mikroprogramozott („reguláris” vezérlési szerkezzettel** – pl. CISC, ill. mai RISC architektúrák):
 - Horizontális mikrokódos vezérlő,
 - Vertikális mikrokódos vezérlő.
- III. **Programozható logikai eszközök (PLD):**
 - Maszk-programozható: PLA, PAL, PROM, CPLD,
 - Újrakonfigurálható (szoftveresen): FPGA

Ismétlés:Kombinációs hálózatok

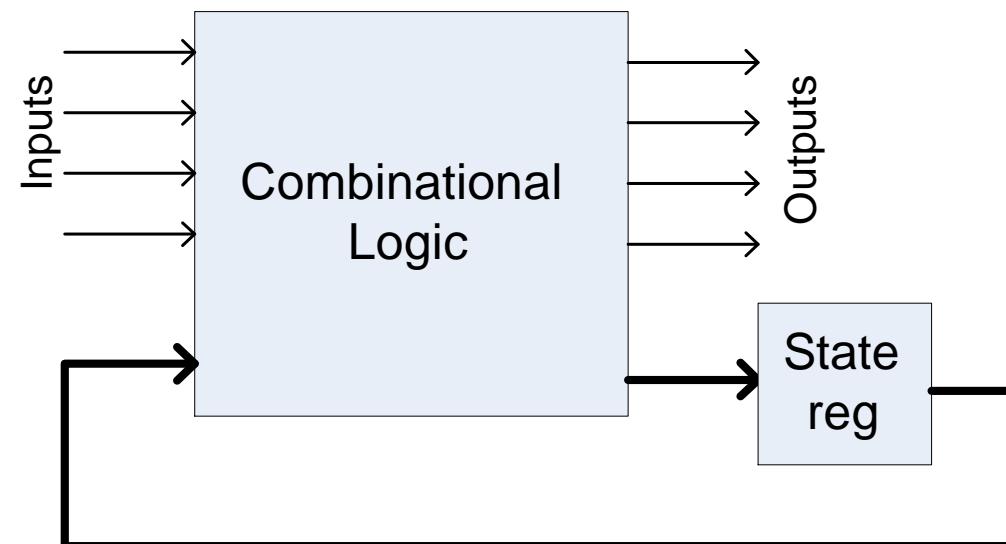
- **(K.H.) Kombinációs logikai hálózatról beszélünk:** ha a mindenkor kimeneti kombinációk értéke csupán a bemeneti kombinációk pillanatnyi értékétől függ (tároló „kapacitás”, vagy memória nélküli hálózatok).



Sorrendi hálózatok:

- **(S.H.) Sorrendi (szekvenciális) logikai hálózatról** beszélünk: ha a mindenkorai kimeneti kombinációt, nemcsak a pillanatnyi bemeneti kombinációk, hanem a korábban fennállt bementi kombinációk és azok sorrendje is befolyásolja. (A szekunder /másodlagos kombinációk segítségével az ilyen hálózatok képessé válnak arra, hogy az ugyanolyan bemeneti kombinációkhöz más-más kimeneti kombinációt szolgáltassanak, attól függően, hogy a bemeneti kombináció fellépésekor, milyen értékű a szekunder kombináció, pl. a State Register tartalma)

Vezérlő egységek
alapjául szolgáló
sorrendi hálózat!

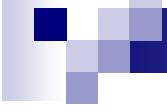


Időzítő – vezérlő egység:

- Az **időzítő (ütemező)** határozza meg a vezérlő jelek előállításának sorrendjét.
- Egy **időzítő-vezérlő** egység általános feladata az egyes funkciók megvalósítását végző áramköri elemek (pl. ALU, memória elemek) összehangolt működésének biztosítása.
- Az időzítő-vezérlő áramkörök ***szekvenciális rendszerek*** – mivel az áramköri egységek tevékenységének egymáshoz viszonyított *időbeli sorrendisége*t biztosítják – melyek az aktuális *kimenet* értékét a *bemenet*, és az *állapotok* függvényében határozzák meg.

Az időzítő-vezérlő lehet:

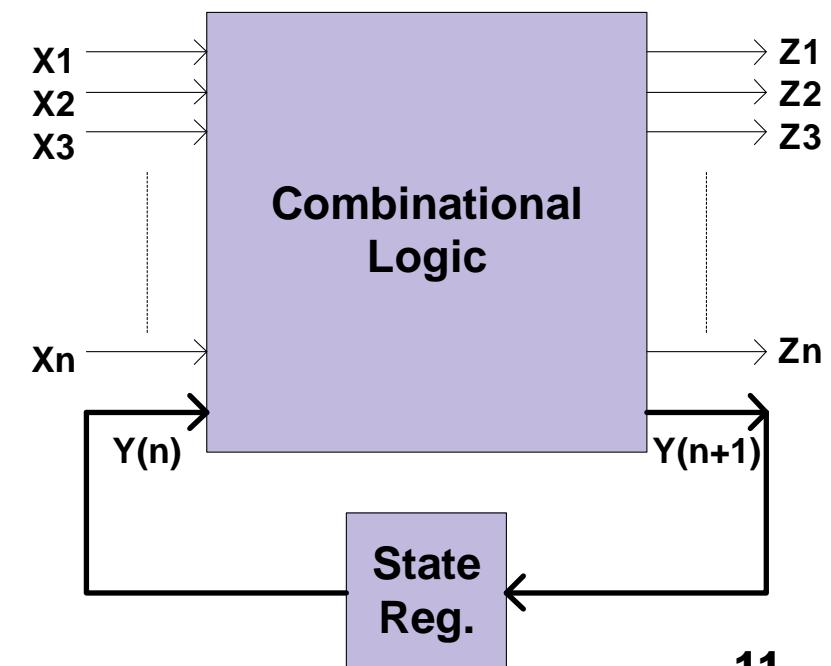
- **Huzalozott/Klasszikus módszer:** áramkörökkel, dedikált összeköttetésekkel fizikailag megvalósított (Mealy, Moore, MUX-os modellek alapján, illetve programozható PLD-k), vagy
 - Pl: korai RISC architektúrák
- **Mikroprogramozott/Reguláris módszer:** az adatútvonal (data-path) vezérlési pontjait Memóriából (ROM) kiolvasott *vertikális*- vagy *horizontális*-mikrokódú utasításokkal állítják be.
 - Pl: mai CISC, RISC architektúrák
- **Programozható logikai áramkörökkel** megvalósított: PLD/FPGA



I. Klasszikus vezérlési módszerek: Huzalozott vezérlő egységek

1.) Mealy-modell

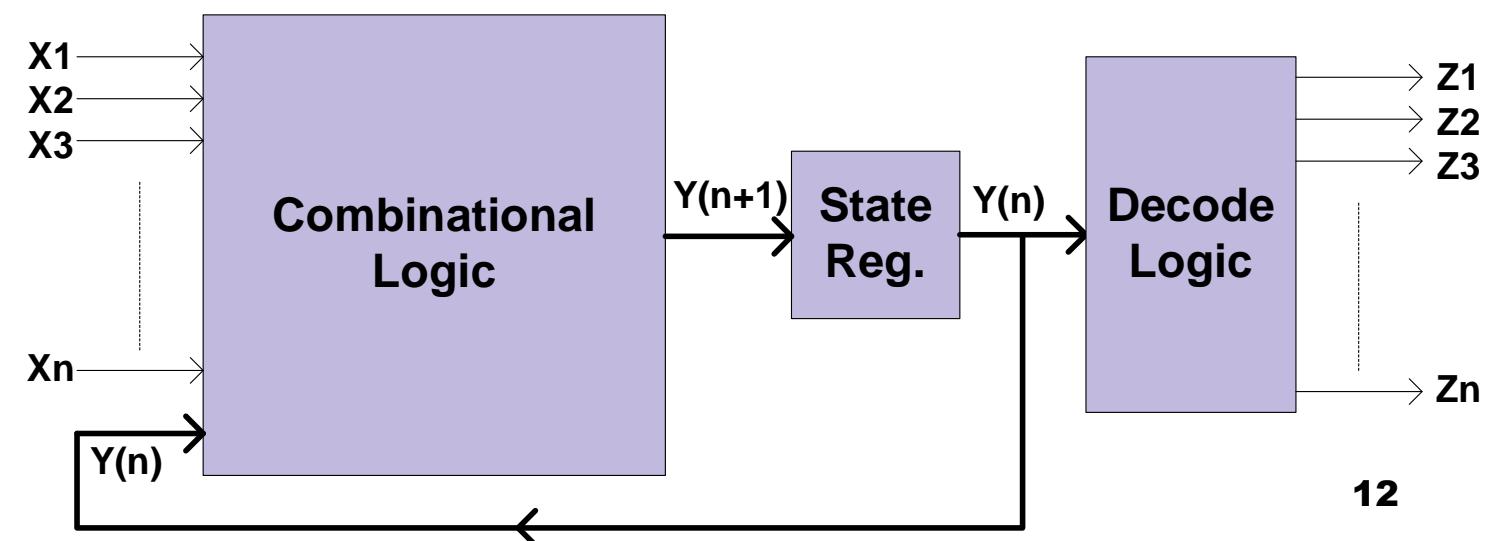
- A sorrendi hálózatok egyik alapmodellje. Késleltetés: a kimeneten az eredmény véges időn belül jelenik meg! Korábbi értékek visszacsatolódnak a bemenetre: kimenetek nemcsak a bemenetek pillanatnyi értékétől, hanem a korábbi állapotuktól is függnek. Problémák merülhetnek fel az állapotok és bemenetek közötti szinkronizáció hiánya miatt (változó hosszúságú kimenet - dekódolás). Ezért alkalmazzuk legtöbb esetben a második, Moore-féle automata modellt.
- Három halmaza van: (Visszacsatolni az állapotregisztert a késleltetés miatt kell)
 - X – a bemenetek,
 - Z – a kimenetek,
 - Y – az állapotok halmaza.
- Két leképezési szabály a halmazok között:
 - $\delta(X_n, Y_n) \rightarrow Y_{n+1}$: következő állapot fgv.
 - $\mu(X_n, Y_n) \rightarrow Z_n$: kimeneti fgv.

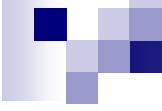


2.) Moore-modell

- A kimenetek közvetlenül csak a pillanatnyi állapottól függnek (bemenettől függetlenek v. közvetve függenek). Tehát a kimenetet nem a bemenetekhez, hanem az állapotoknak megfelelően szinkronizáljuk.
- Három halmaza van:
 - X – a bemenetek,
 - Z – a kimenetek,
 - Y – az állapotok halmaza.
- Két leképezési szabályok
 - $\delta(X_n, Y_n) \rightarrow Y_{n+1}$: köv. állapot fgv.
 - $\mu(Y_n) \rightarrow Z_n$: kimeneti fgv.

Input \Rightarrow Next-State \Rightarrow Present-State \Rightarrow Output





Vezérlő egység tervezési feladat:

Adatút (data-path) tervezése FIR szűrőhöz

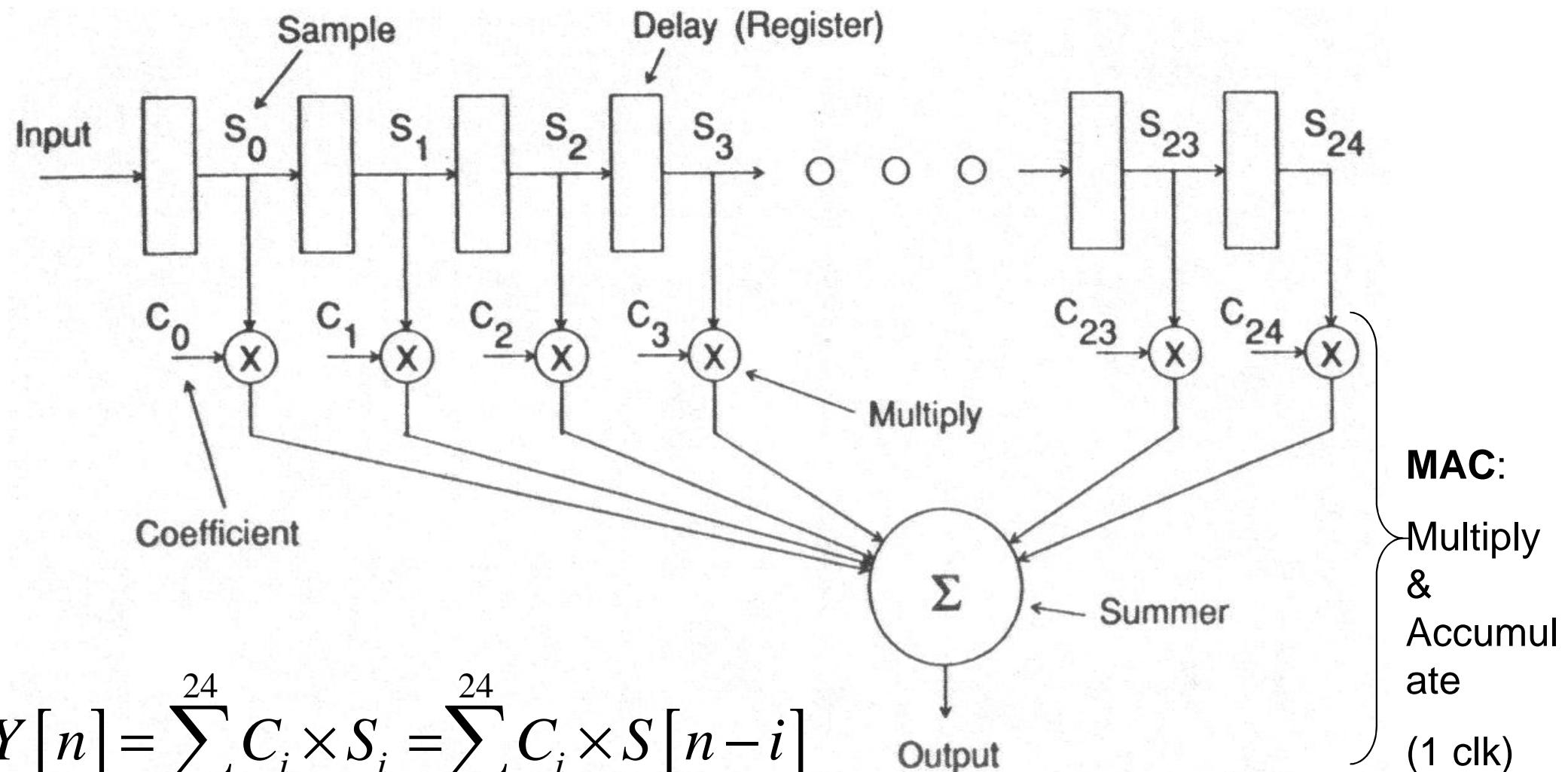
Adatút tervezés - FIR szűrő

- **FIR = Finite Impulse Response** szűrővel egy digitális rendszerben könnyen kiszámíthatók pl. skaláris/ belső szorzat, mátrixműveletek stb. Alkalmazzák a digitális jelfeldolgozásban (DSP) áteresztő szűrőknél. **Feladat:** tervezünk egy FIR szűrőt, amely 25 együtthatót képes kezelni, és a kimenetét képező eredményt a következő egyenettel kapjuk meg:

$$output = \sum_{i=0}^{24} S_i \times C_i$$

- ahol S_i a bemeneti adatfolyam i. mintája, ill. C_i az i. konstans értéke. A/D konverterrel állítja elő a mintákat, ill. D/A konverterrel alakítja vissza analóg jellé, további feldolgozás szempontjából elfogadható formába. minden egyes mintát meg kell szorozni a neki megfelelő együtthatóval (koefficiens), majd a szorzatokat össze kell adni. Egyenként 25 mintát veszünk késleltetve, (elvileg 25 különböző szorzó áramkör kellene). De ezt a feladatot a MAC egység látja el.

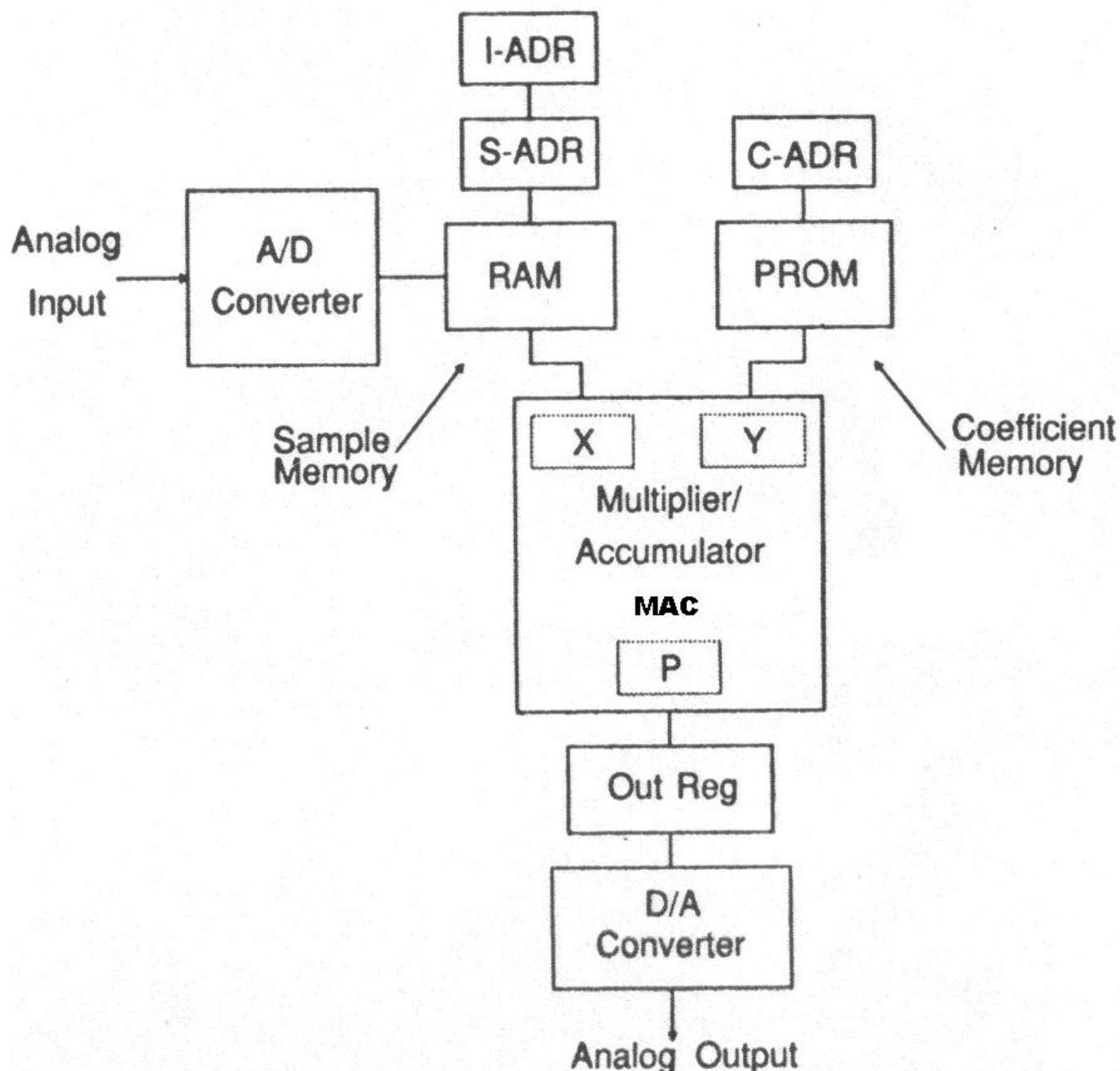
Példa: FIR szűrő - feldolgozás



Példa: FIR szűrő felépítése

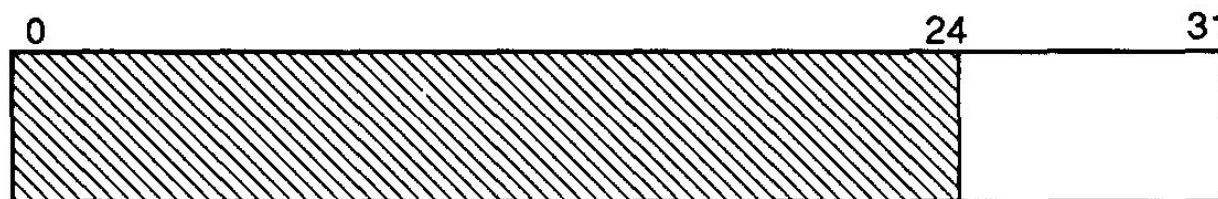
- **IMAC: (Multiplier/Accelerator):** a szorzást, és az összeadást egy órajel ciklusban képes elvégezni. Három regiszterből áll: két bemeneti (X és Y) és egy kimeneti regiszterből (P).
- **Együttható-memória (Coefficient memory):** C_MEM tárolja a Ci konstansok értékeit. Ez egy kisméretű PROM memória.
- **Együttható-memória címregiszter: (C_ADR)** egy számláló, amely a aktuális együttható címét azonosítja. 0-ról indul minden egyes iterációkor, és a felhasznált együtthatók címei egyesével *inkrementálódnak*.
- **Minta-memória (S_MEM):** az éppen aktuális és az azt megelőző 24 mintát tároljuk el. Egy RAM, amely 32 értékéből csak 25-t használ.
- **Minta-memória címregiszter: (S_ADR++)** Ez egy számláló, amely az aktuális *minta címét* inicializálja, minden egyel *inkrementálódik*, ezáltal a (sorrendben) lévő minták címére mutat (azonosítja)
- **Kezdeti minta címregiszter (I_ADR--):** ez a regiszter azonosítja az algoritmus kezdőpontját minden egyes átmenetnél/pass (aktuális iteráció kezdőcíme *mindig 1-el kevesebb lesz*)
- **A/D konverter: (ADIN)** minden iterációban ezen az egységen keresztül kapjuk az új adatot.
- **D/A konverter: (DAOUT)** A kimeneti regiszterből kapja az adatot, és analóg jellé alakítja, a további feldolgozás számára elfogadható formában.
- **Kimeneti regiszter: (OUT)** MAC-ből jövő értéket tárolja és a D/A konverternek továbbítja.

Példa: FIR szűrő blokkszintű felépítése



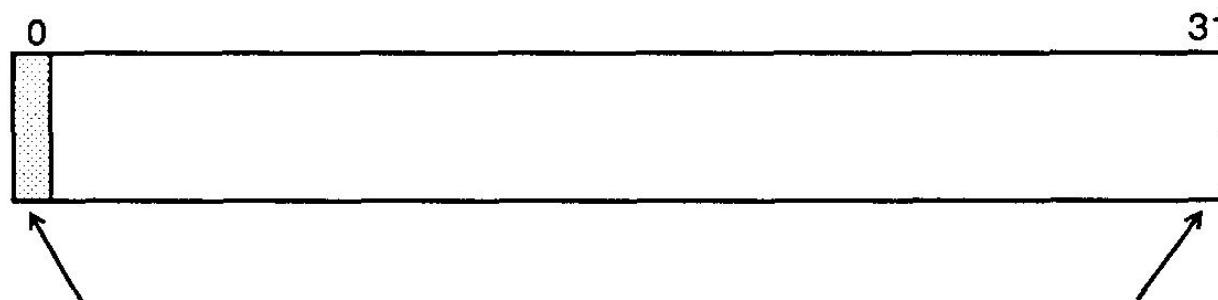
Koefficiens- és minta-adat tárolása

- Mindkettő C_MEM, S_MEM is 32 elemű tárolók



Coefficients
stored in first
25 locations of
PROM C_ADDR++
(0...24)

- Első betöltött minta után:



First sample
stored at
location 0
(0th sample)
S_ADDR++

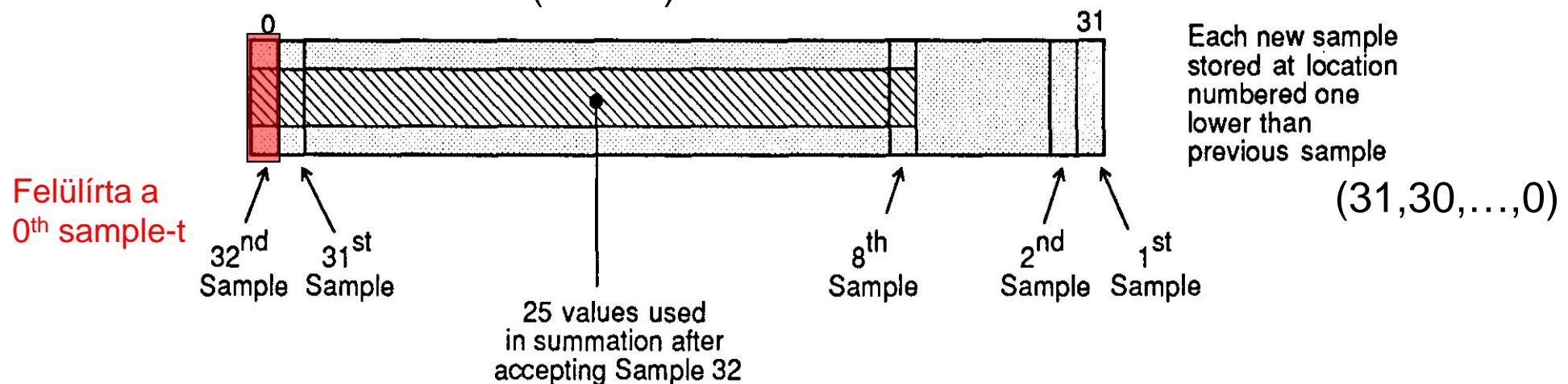
Samples start
at location 0
of RAM S_ADDR++
(0- ...)

Initial Sample
Address Register
decrements to
point to location 31

IADR--
(31-...) dekrementálása!

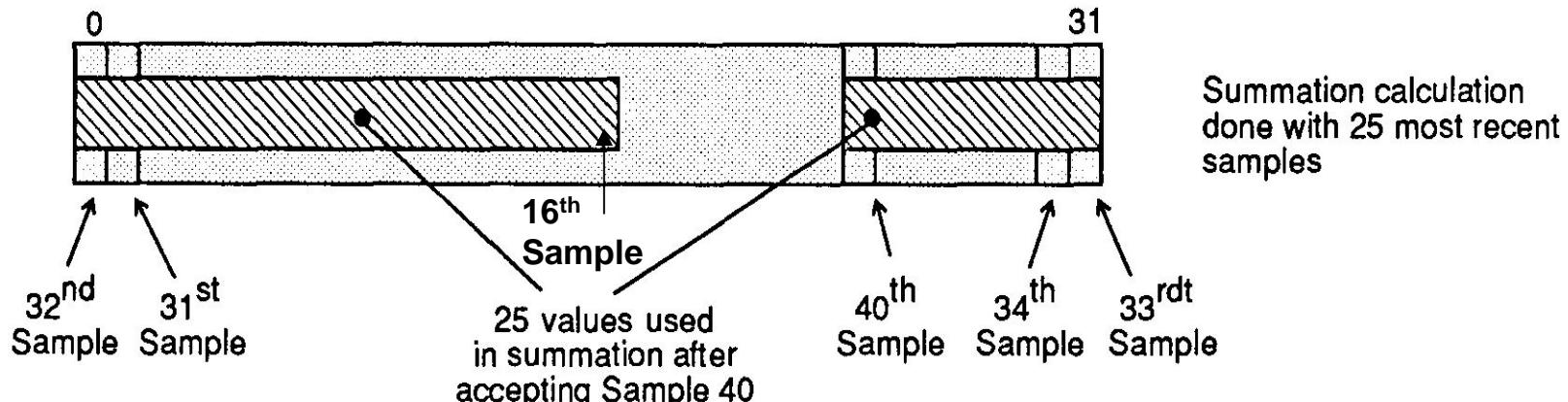
Koefficiens és adat tárolás (folyt)

- 33 betöltött minta után (=32nd):



- 40 betöltött minta után:

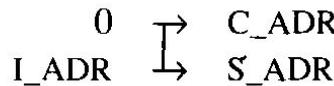
Kimenetre mindig az $N=25$ **legutolsó** érték kerül!



A 25 legutolsó minta mindenegyes iterációban sorrendben, de a RAM különböző területén tárolódik el.

FIR - RTL leírása

start: if (ADIN not ready) goto start



Check input data.

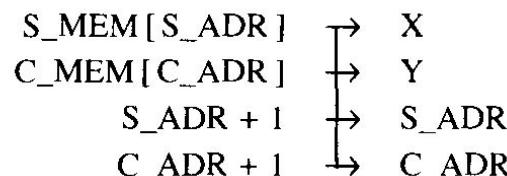
Várakozik..., új adat érkezéséig



Clear coefficient address.
Load sample address.

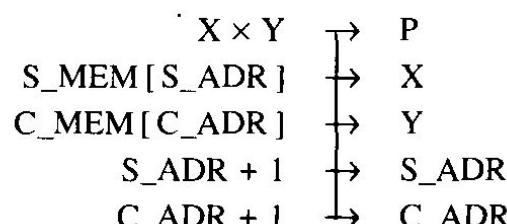
Data to sample memory.
Decrement initial address.

Szimultán betöltés és dekrementálás



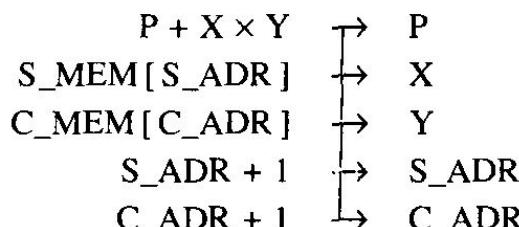
Load sample to X reg.
Load coefficient to Y reg.
Increment sample address.
Increment coefficient address.

Első minta (X) és első koefficiens (Y) betöltése a MAC bemenetére



Load product register.
Load sample to X reg.
Load coefficient to Y reg.
Increment sample address.
Increment coefficient address.

Szorzat (P) számításával egyidejűleg új mintát (X) és új koefficiens (Y) töltünk be



Add product into P register.
Load sample to X reg.
Load coefficient to Y reg.
Increment sample address.
Increment coefficient address.

Iterációs ciklus, amíg „not Done”, tehát nem érjük el az utolsó mintát!

if (not done) goto over

Repeat for all samples.

Work is done when C_ADDR is 25.

P → OUT

Update output value.

goto start

Start over.



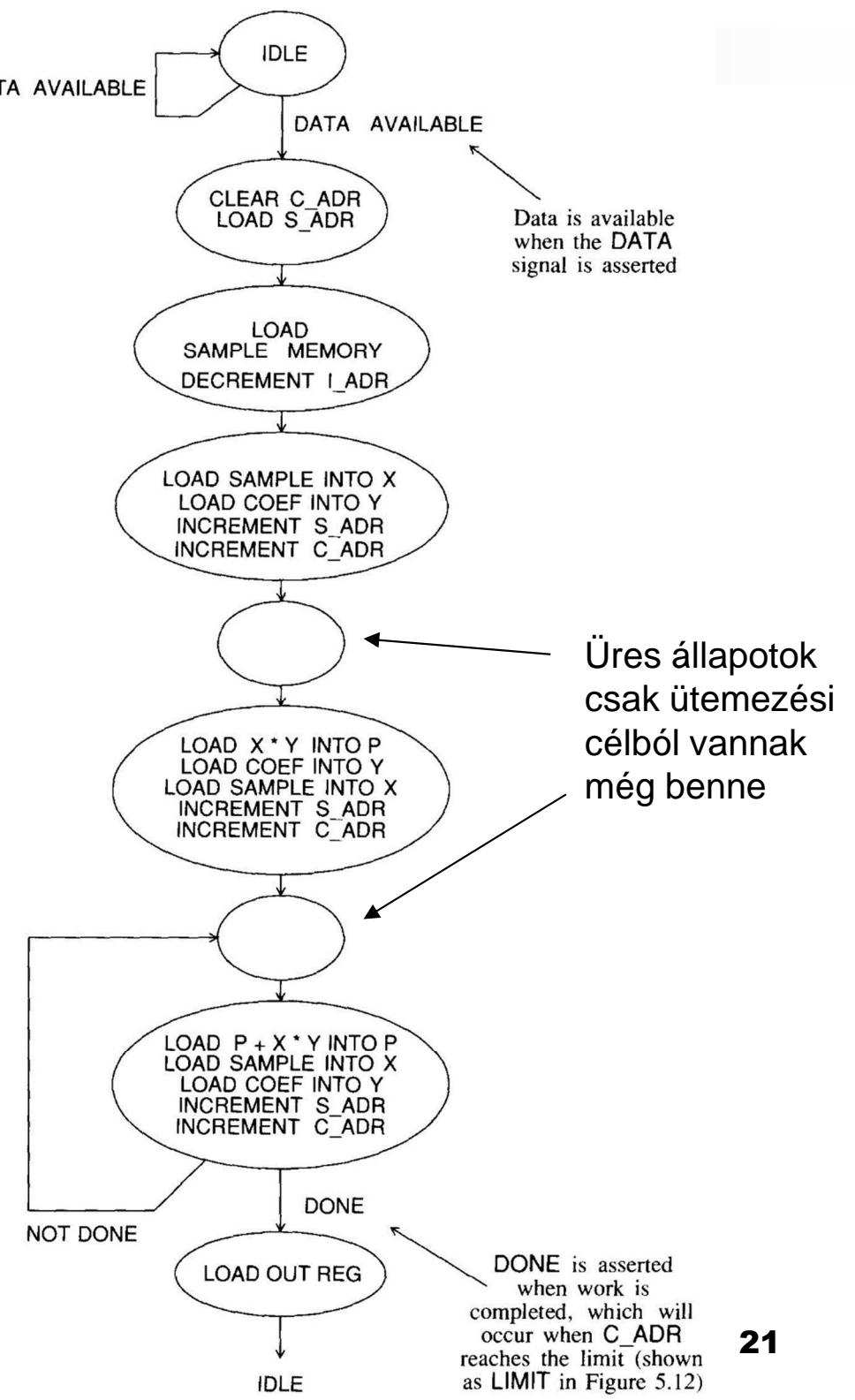
NO DATA AVAILABLE

RTL leírásnak megfelelő „előzetes” állapotdiagram

Vezérlési szekció: adatátvitel nem azonnal, hanem véges idő alatt megy végbe!

Ezt a tervezőnek kezelnie kell tudni (előzetes diagramm).

RTL leírásból képzett diagramnak lehetnek még üres állapotai is, azonban később az optimalizálás, véglegesítés során ezek kiesnek – egyszerűsödnek!



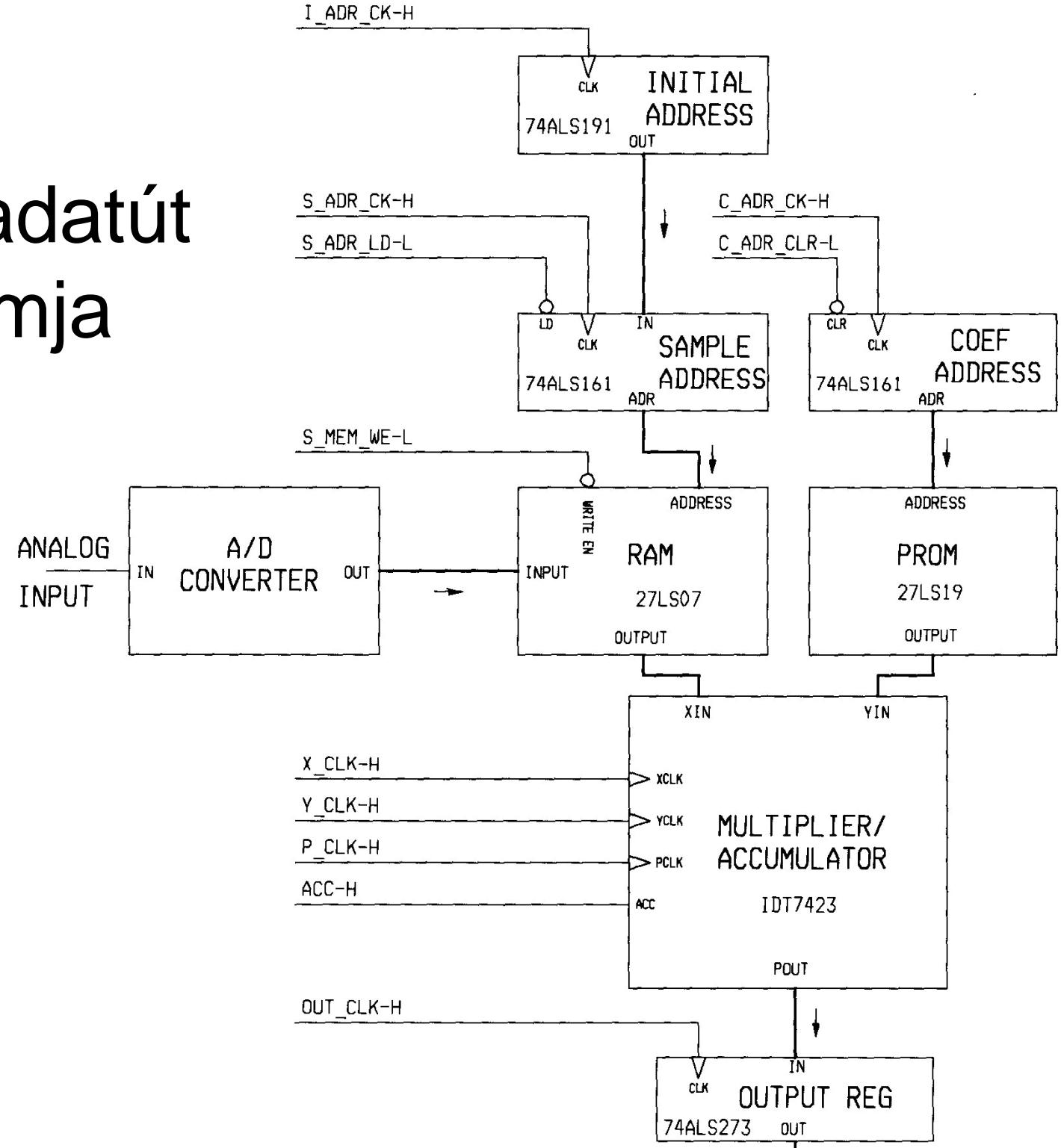
FIR szűrő „részletes” adatút blokkdiagramja

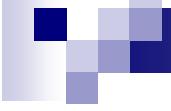
Végeles adatút
diagramm:

RTL leírás + előzetes
diagramm alapján
kapjuk!

Itt már a **vezérlő jelek**
is fel vannak tüntetve!

További tervezői
feladatunk lesz
ezeknek a vezérlő-
jeleknek a megfelelő
ütemben való
generálása!





Vezérlő egység tervezése egyszerű állapotgép (FSM) segítségével – FIR szűrőhöz

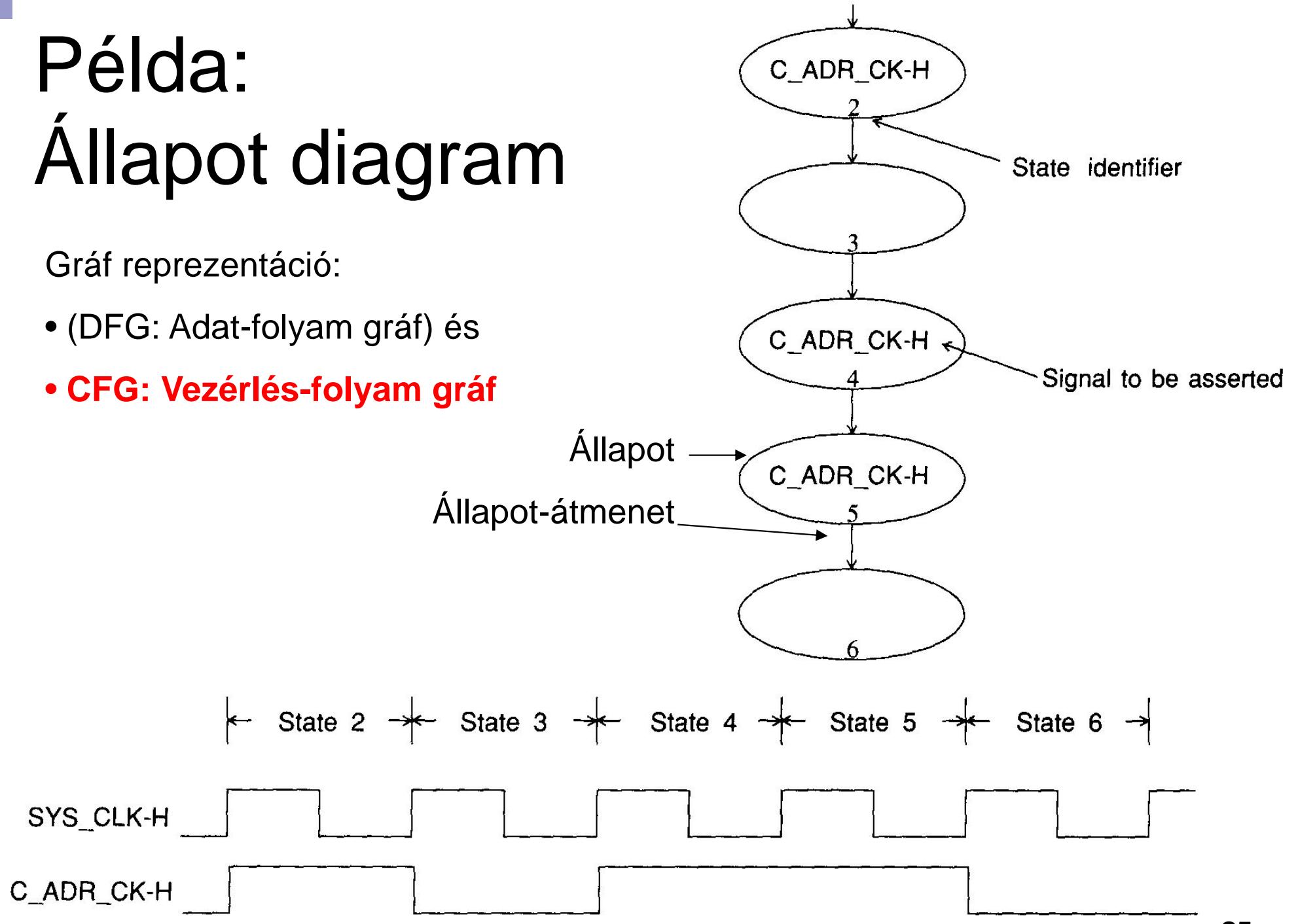
Tervezői feladatok (FIR szűrő):

- Eddig rendelkezésre áll:
 - RTL leírás,
 - Előzetes és
 - részletes blokk diagram
- Most: a vezérlési rész tervezése következhet
 - 1.) Vezérlő jelek beállítása (assertion levels)
 - 2.) „Mapping” (hozzárendelés): a Moore modellt vesszük alapul, amelyhez a Véges Állapotú Automatát (FSM) próbáljuk illeszteni / hozzárendelni.

Példa: Állapot diagram

Gráf reprezentáció:

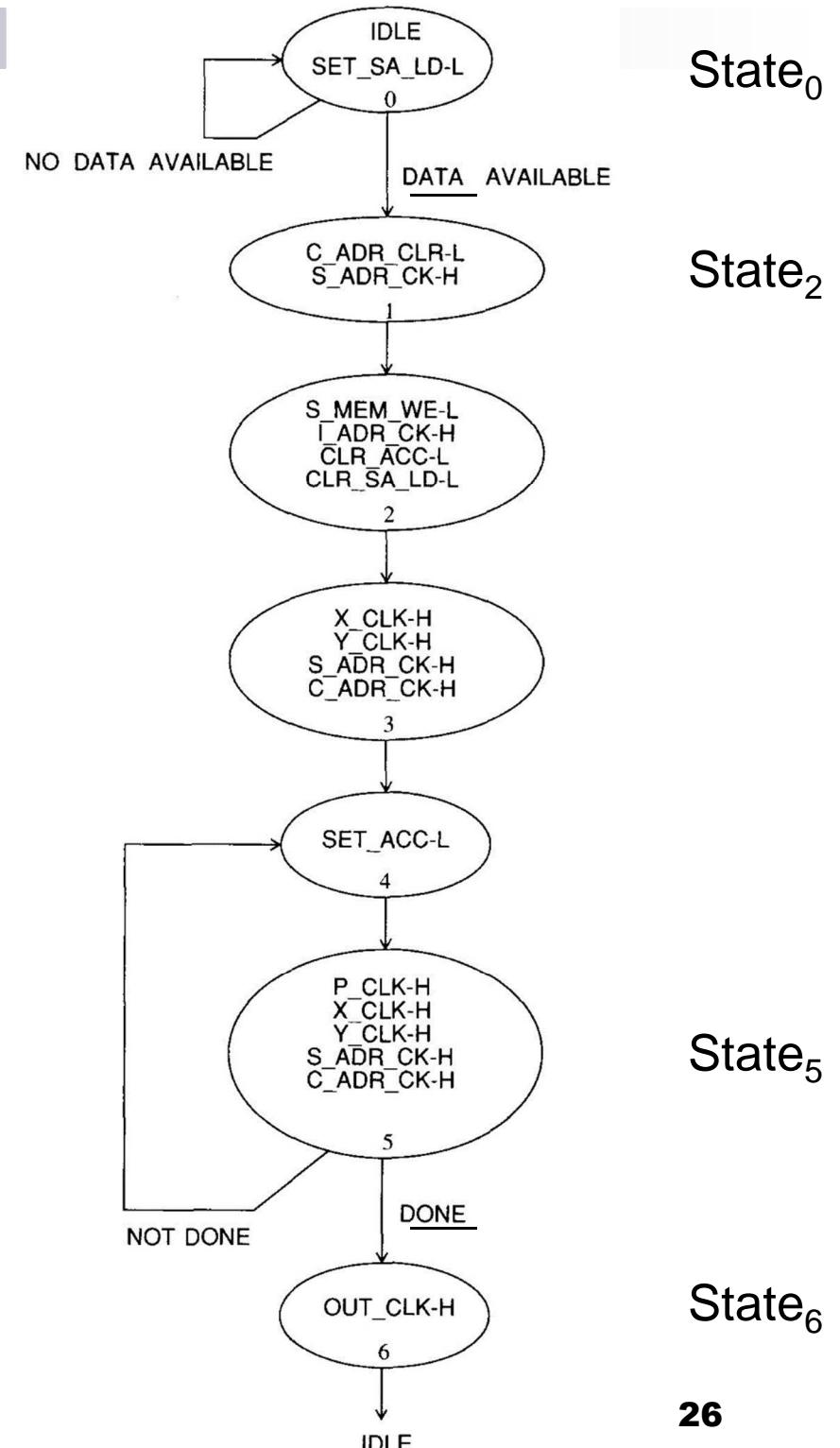
- (DFG: Adat-folyam gráf) és
- **CFG: Vezérlés-folyam gráf**



1.) FIR szűrő részletes FSM állapot diagramma

SET-RESET FF-kal:

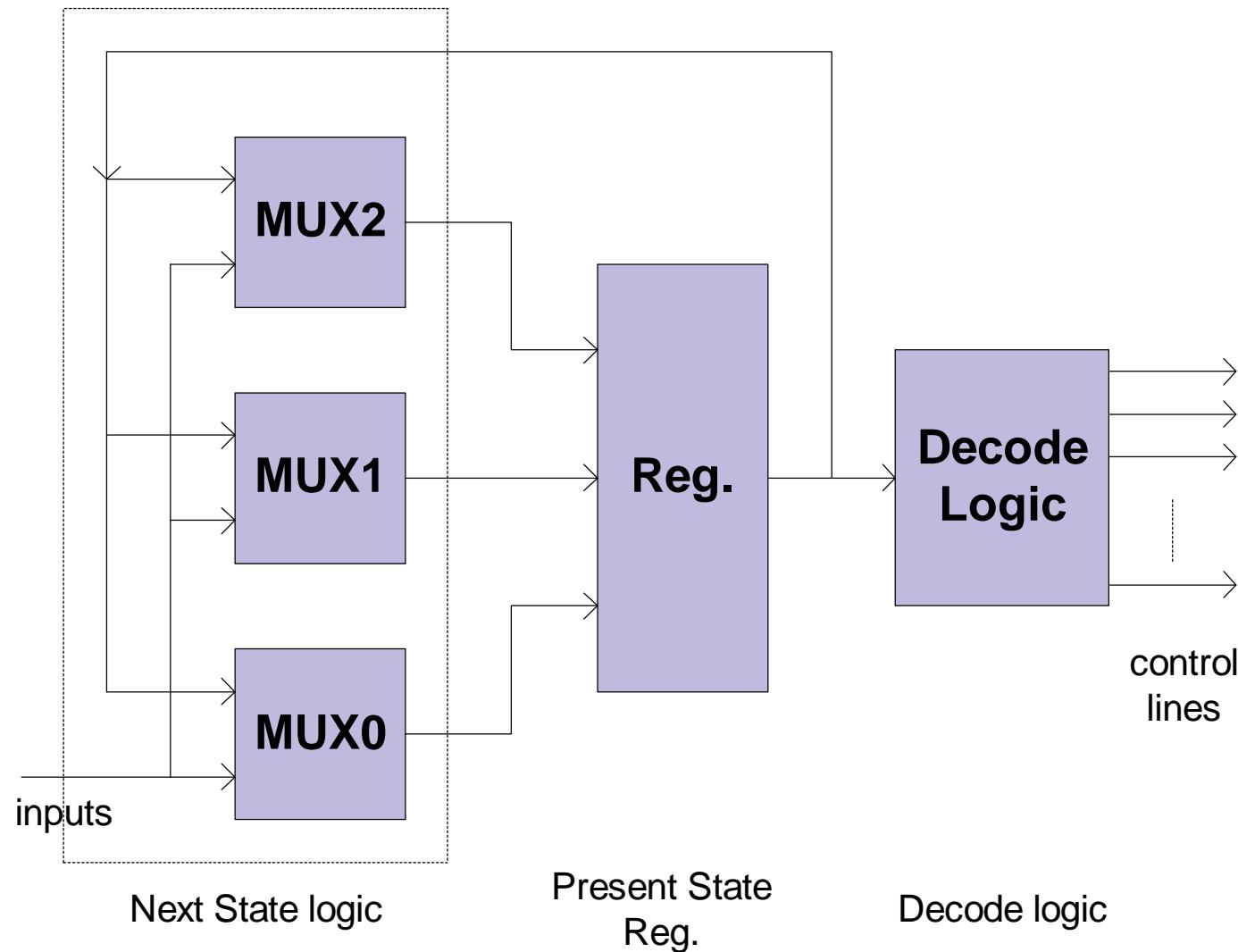
- Set S_ADR_LD (State0)
- Reset / Clear S_ADR_LD (State2)
- Reset ACC_L (State2)
- Set ACC_L (State4)
- Vezérlés folyam: State₀, ..., State₆



2.) „Mapping”: FIR szűrő vezérlőjének multiplexeres megvalósítása

- **Present State Reg:** aktuális állapotot tárolja
- **Next State Logic:** következő állapot kiválasztása a bemenetek, és a visszacsatolt állapotuktól függően (multiplexerek!)
- **Output Logic:** az aktuális állapot dekódolt formája kerül a kimenetekre (vezérlőjel generálás)

FIR szűrő vezérlőjének multiplexeres megvalósítása



FIR szűrő vezérlőjének multiplexeres megvalósítása

■ További két input signal

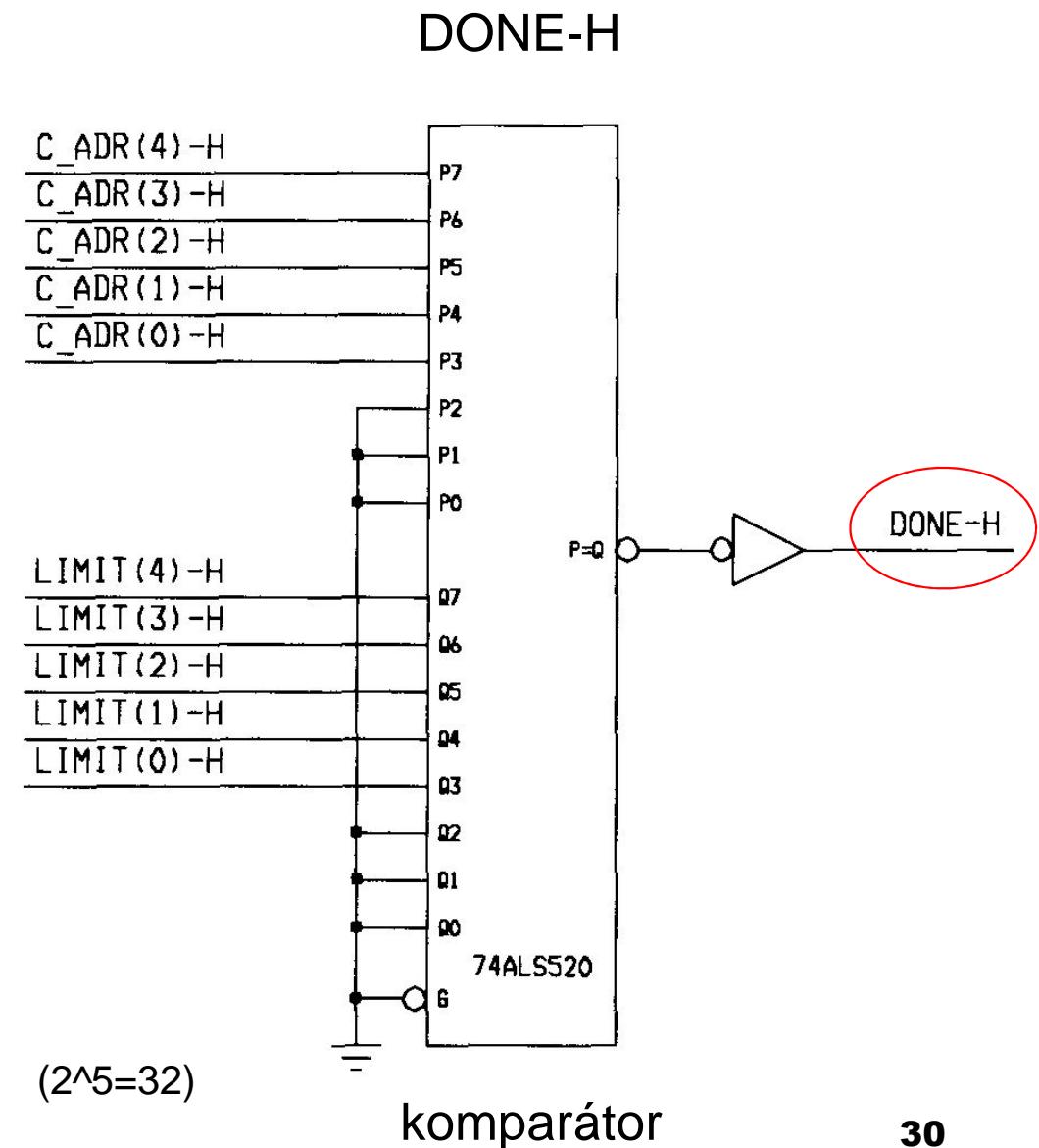
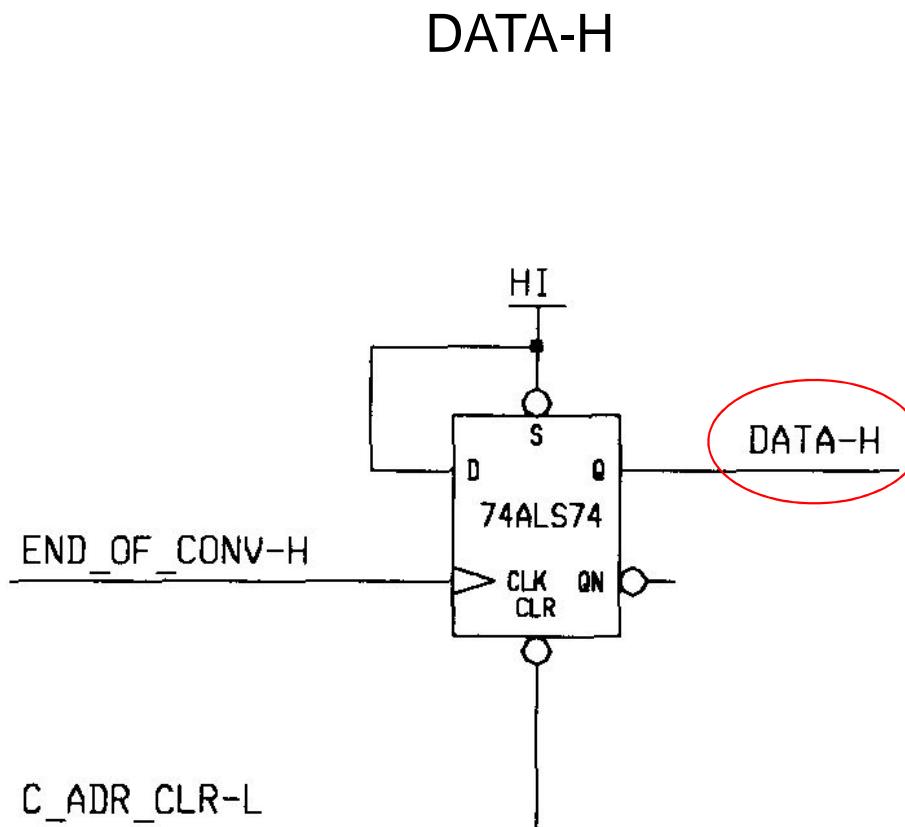
- DATA-H: A/D felől új feldolgozandó adat érkezett (flag)
- DONE-H: szükséges számú iteráció végrehajtódott

Table 5.1. Next State Multiplexer Specifications.

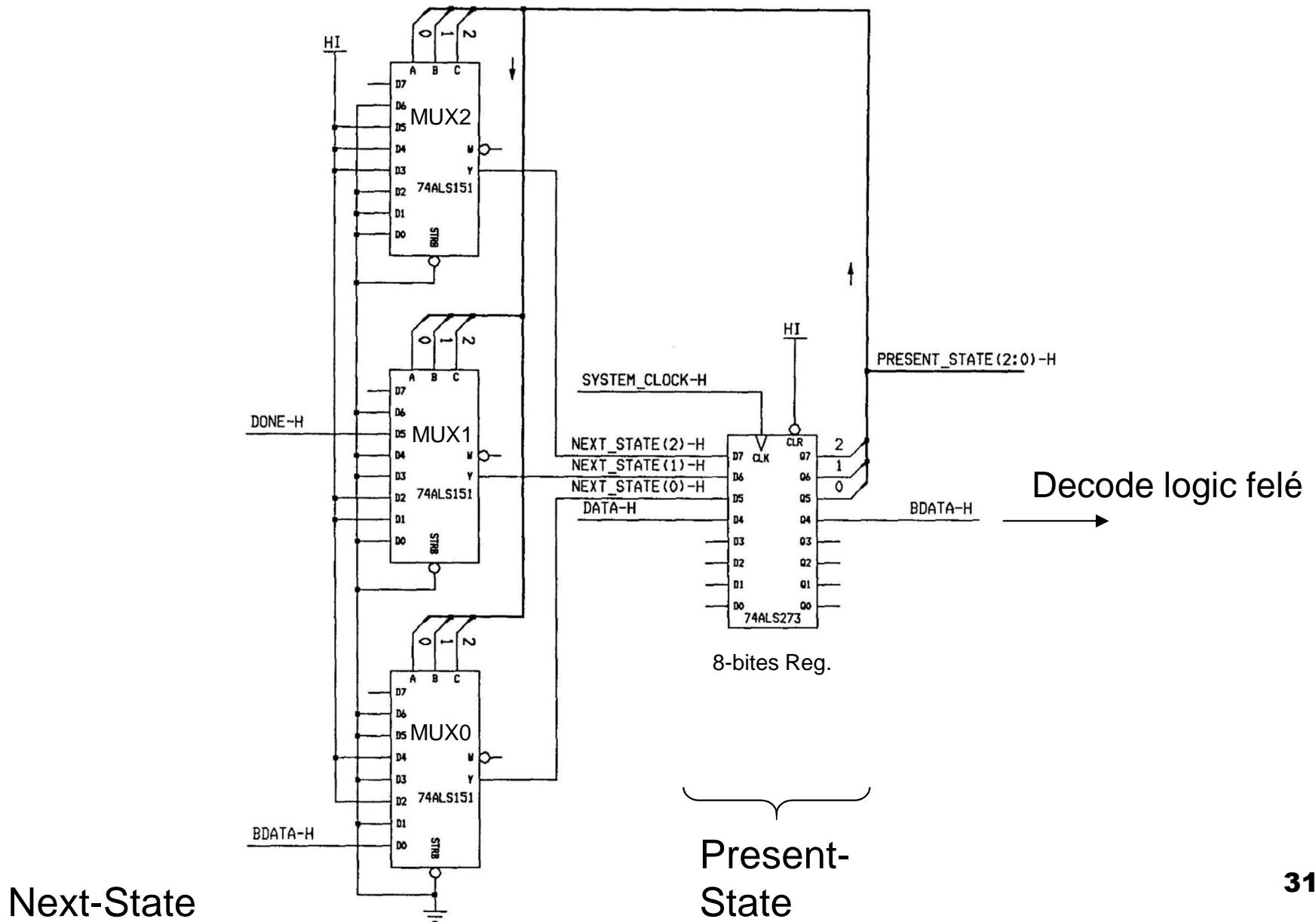
	<i>MUX 2</i>	<i>MUX 1</i>	<i>MUX 0</i>
State 0	0	0	DATA-H
State 1	0	1	0
State 2	0	1	1
State 3	1	0	0
State 4	1	0	1
State 5	1	DONE-H	0
State 6	0	0	0

FIR szűrő részletes FSM állapot-diagram alapján kapjuk a táblázatot!

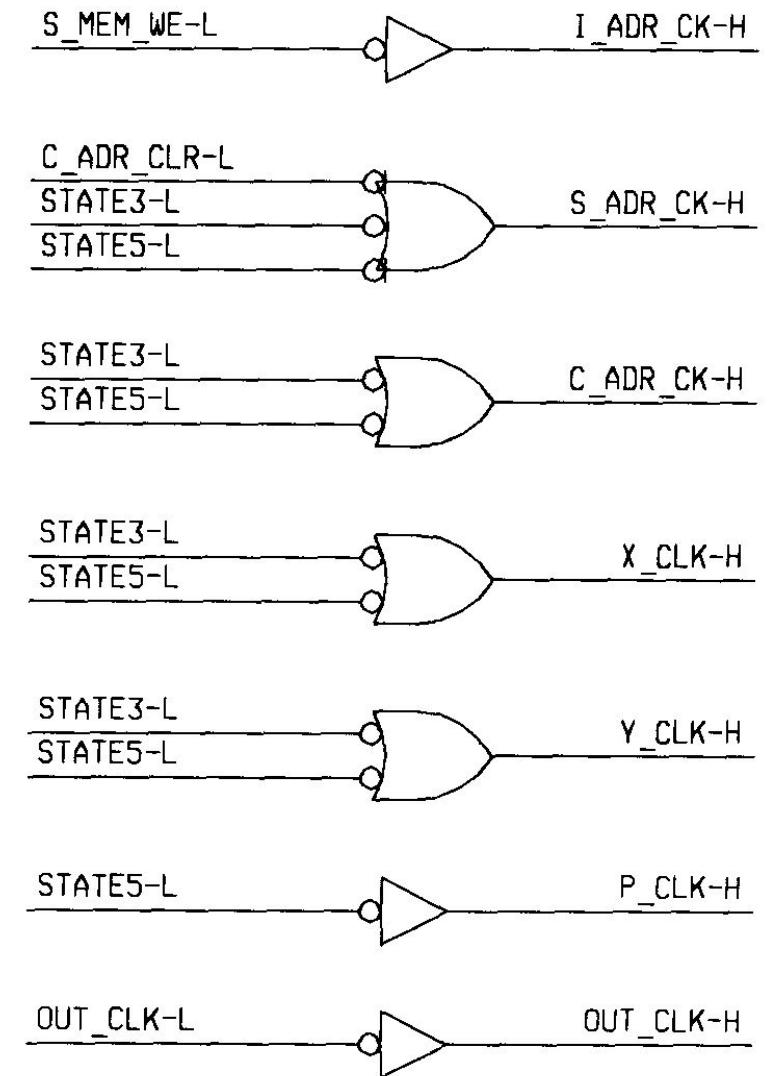
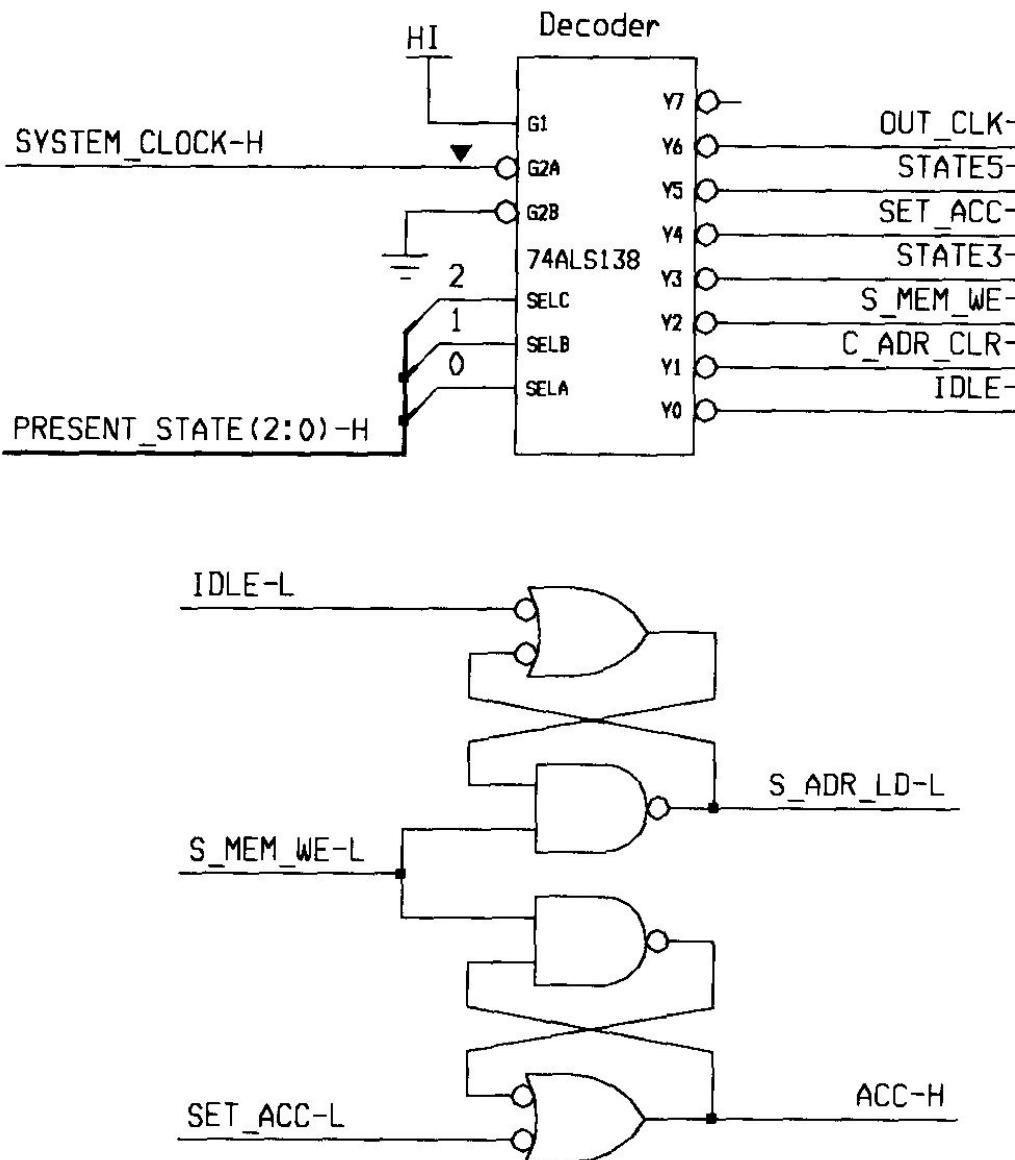
FSM vezérlőjelei

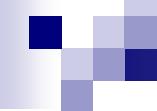


FSM vezérlő egységének Next-State és Present-State logikája (blokk diagramm)



Present-State logikai áramkörei





Szekvenciális vezérlő rendszerek – egyedi késleltetéses módszer

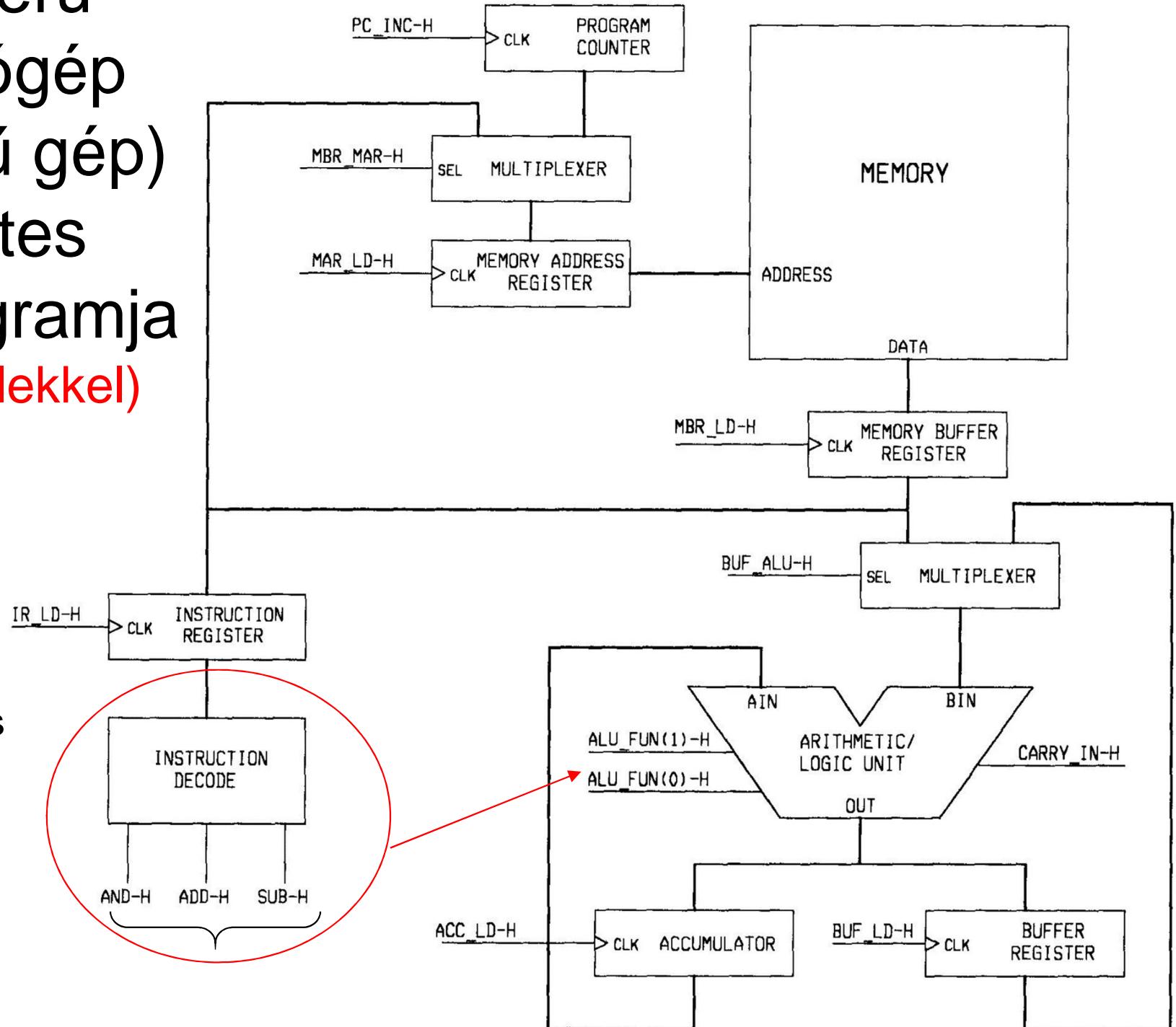
Szekvenciális vezérlő rendszerek

- Ahelyett – mint azt az állapotgépnél (FSM) láthattuk – hogy egyetlen regiszter tömbben tárolnánk a rendszer állapotait, ebben az esetben *külön regiszterekeket* definiálunk egy egyszerű számítógép (egycímű gép) blokkdiagramját felhasználva.
- Regiszter-transzfer műveletek sora mutatja be ennek a késleltetéses vezérlő rendszernek a működését.

Egyszerű számítógép (egycímű gép) részletes blokkdiagramja (vezérlő jelekkel)

Példa: 3
egyszerű utasítás

- ADD
- SUB
- AND



Példa: Egycímű gép vezérlési funkciója

- Mivel példaként három egyszerű két-operandusú utasítást (AND, ill ADD, SUB) akarunk végrehajtani egy egycímű gépen, ezért a második operandus értékét az ACC-ből kell betölteni!
- Ehhez az ALU néhány alapvető funkciója:

Késleltetések!

$T_{REG} = 40\text{ ns}$

$T_{MEM} = 200\text{ ns}$

ALU_FUN		OUT function	
0	0	bitenkénti AND (A_In, B_In)	[40 ns]
0	1	bitenkénti OR (A_In, B_In)	[40 ns]
1	0	inverz NOT (B_In)	[40 ns]
1	1	bináris ADD (A_In, B_In)	[80 ns]

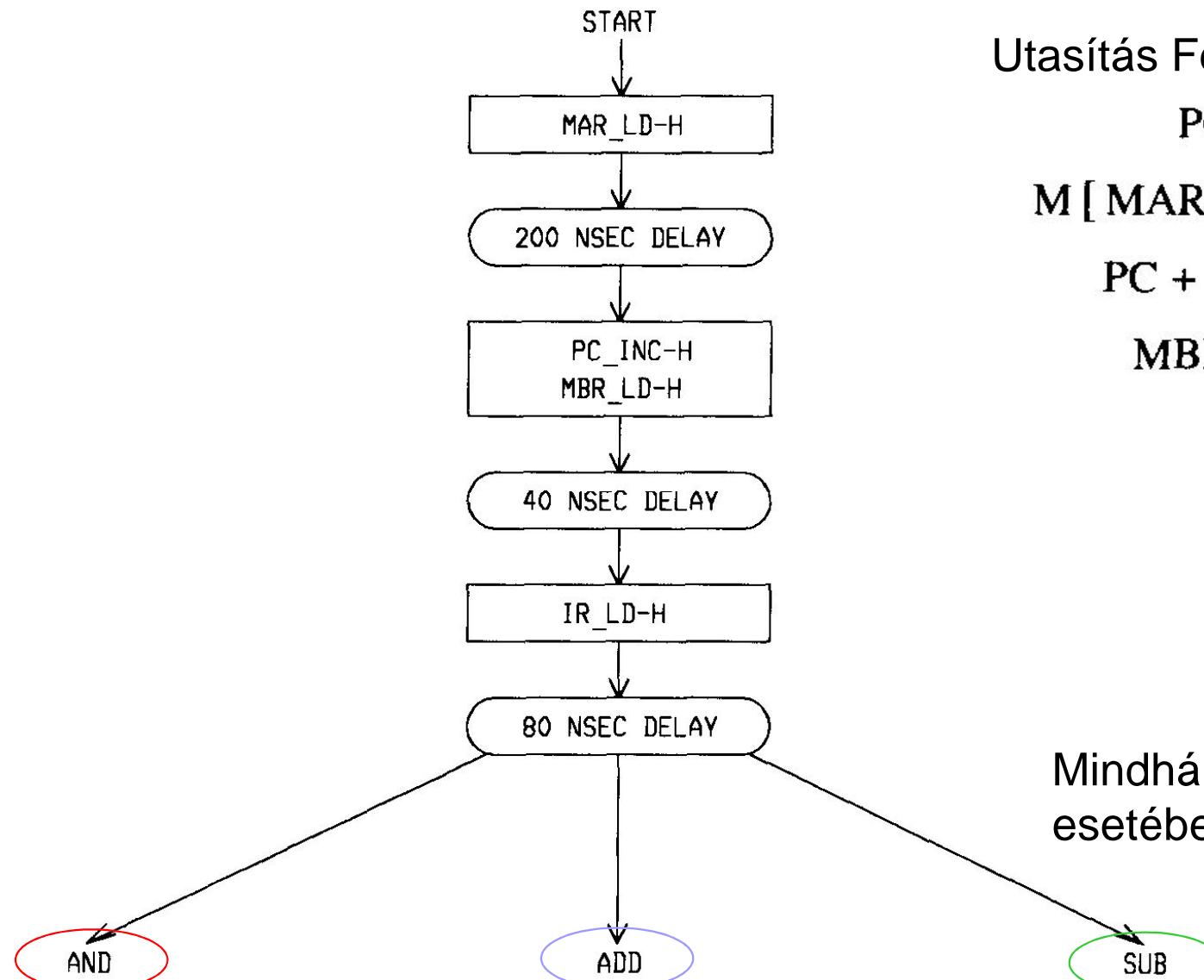
Három utasítás RTL leírása

Table 5.2. Register Transfers for Three Instructions.

Register Transfers for Example

<i>AND Instruction</i>	<i>ADD Instruction</i>	<i>SUBTRACT Instruction</i>
$PC \rightarrow MAR$	$PC \rightarrow MAR$	$PC \rightarrow MAR$
$M[MAR] \rightarrow MBR$	$M[MAR] \rightarrow MBR$	$M[MAR] \rightarrow MBR$
$PC + 1 \xrightarrow{\quad} PC$ $MBR \xrightarrow{\quad} IR$	$PC + 1 \xrightarrow{\quad} PC$ $MBR \xrightarrow{\quad} IR$	$PC + 1 \xrightarrow{\quad} PC$ $MBR \xrightarrow{\quad} IR$
$MBR \rightarrow MAR$	$MBR \rightarrow MAR$	$MBR \rightarrow MAR$
$M[MAR] \rightarrow MBR$	$M[MAR] \rightarrow MBR$	$M[MAR] \rightarrow MBR$
$MBR \bullet ACC \rightarrow ACC$	$MBR + ACC \rightarrow ACC$	$\neg MBR \rightarrow BUF$ $BUF + ACC + 1 \rightarrow ACC$

Három utasítás késleltetéses folyamat ábrája (delay flowchart)



Utasítás Fetch:

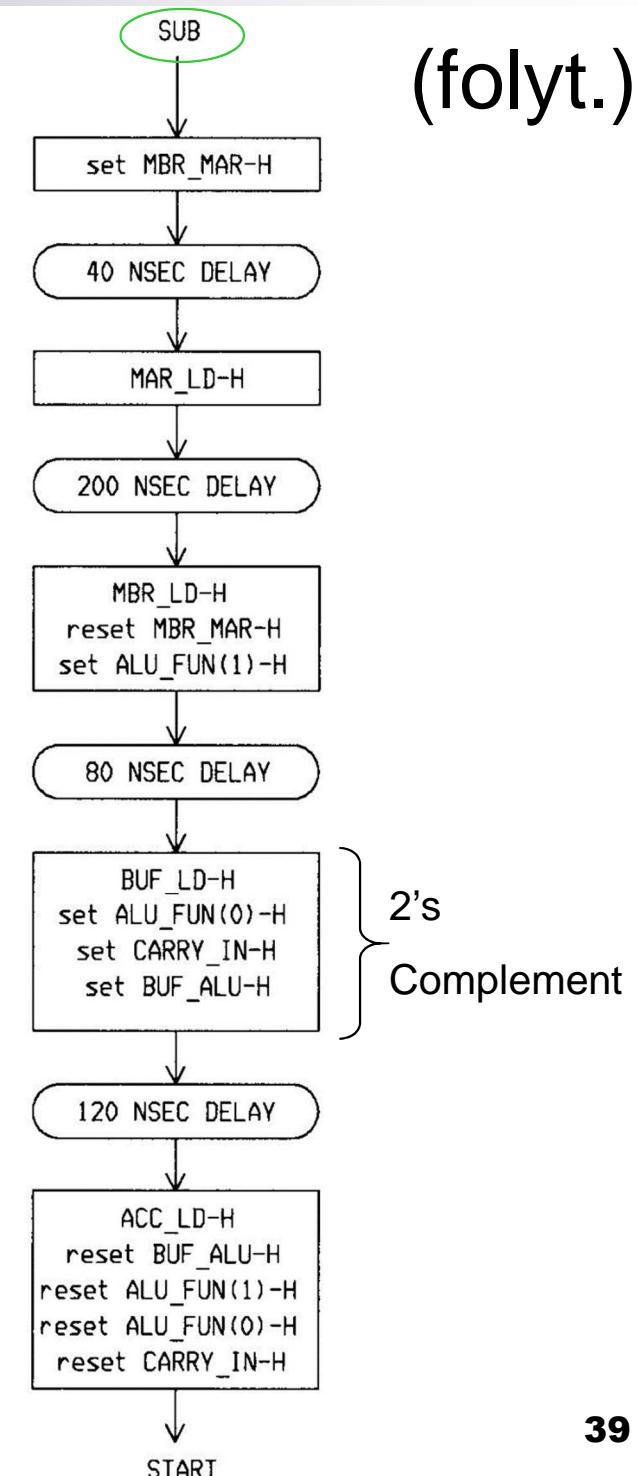
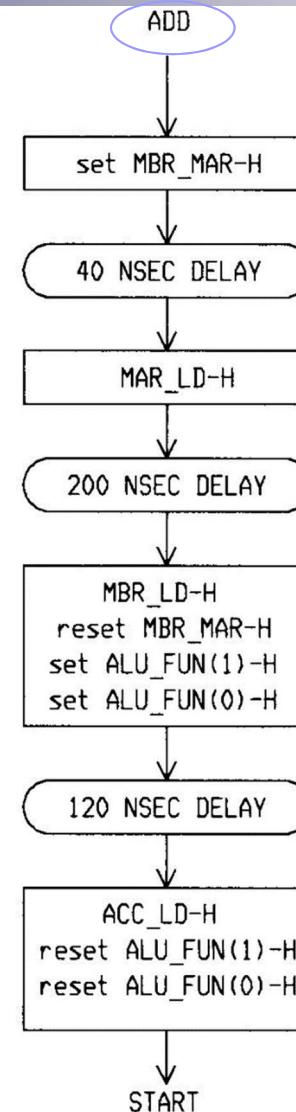
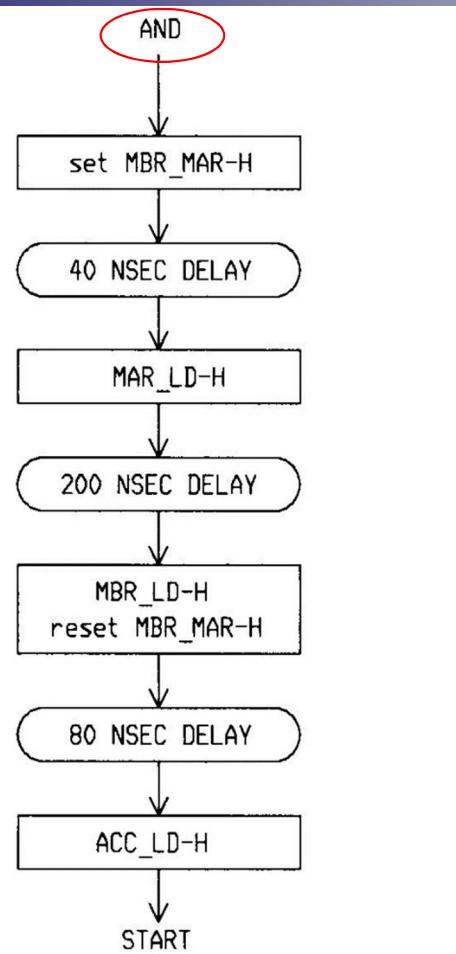
$$PC \rightarrow MAR$$

$$M [MAR] \rightarrow MBR$$

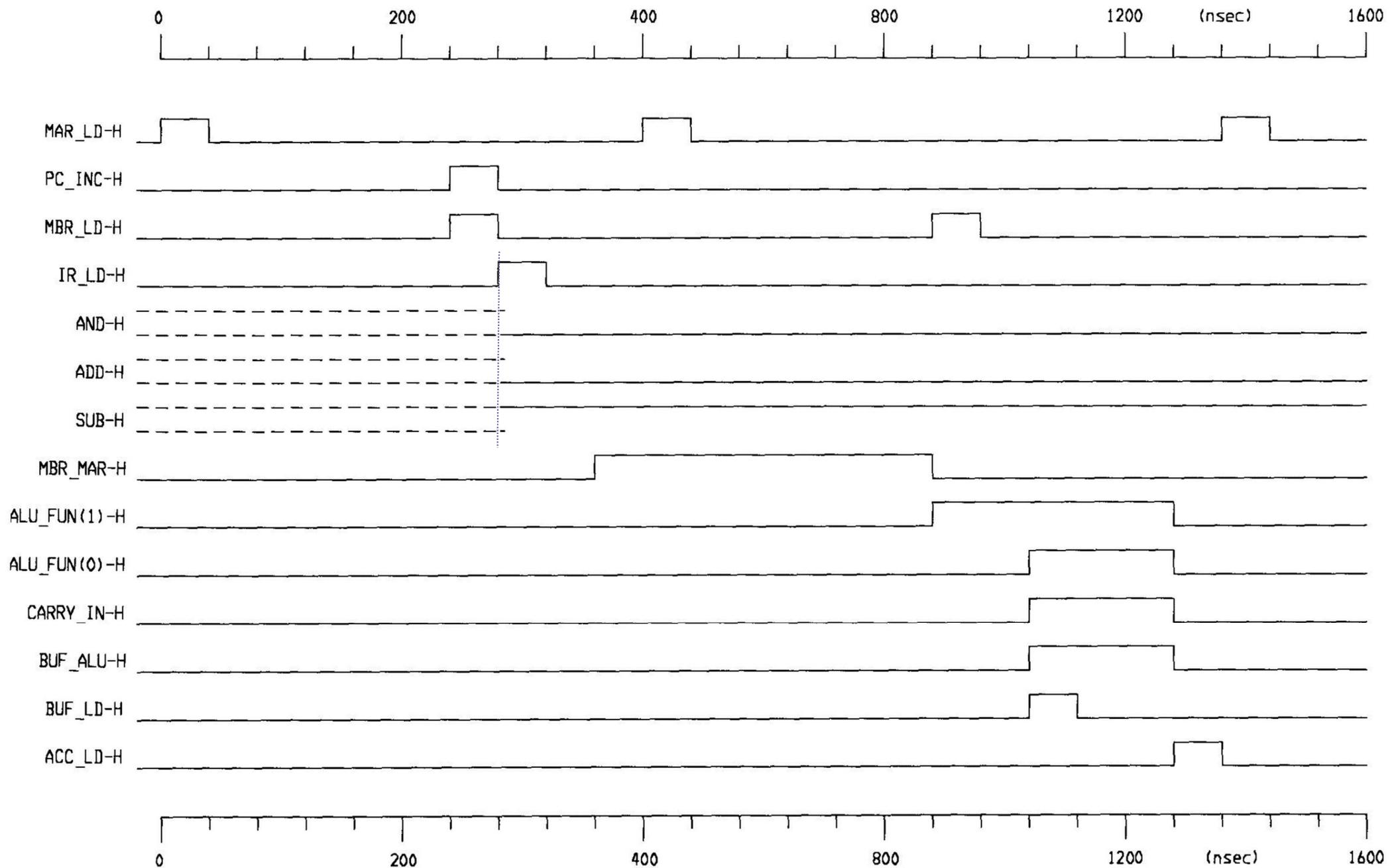
$$\begin{array}{c} PC + 1 \rightarrow PC \\ MBR \rightarrow IR \end{array}$$

Mindhárom utasítás esetében ugyanaz.

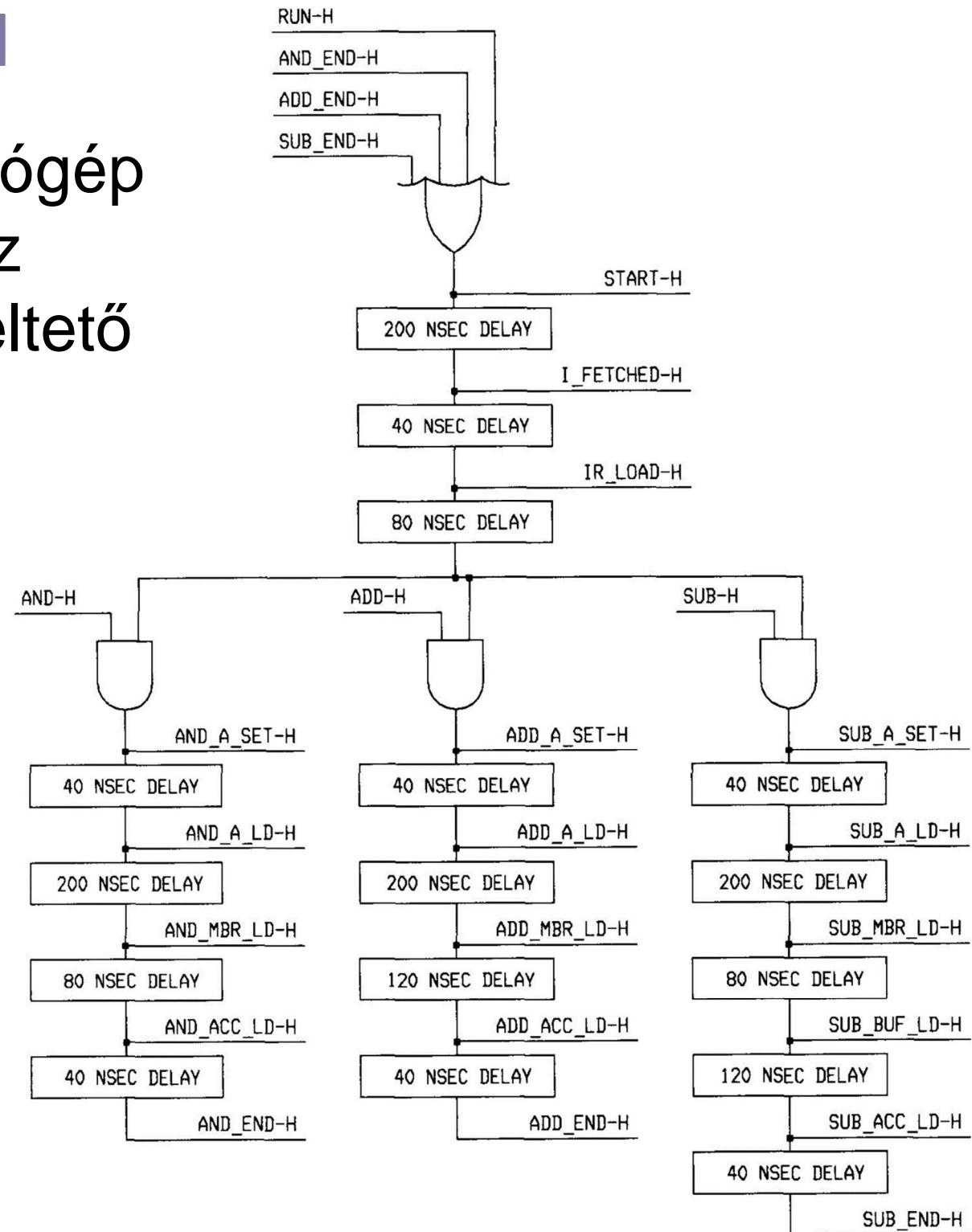
(folyt.)

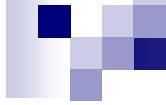


PI: SUB művelethez tartozó időzítési diagram



Egyszerű számítógép vezérléséhez szükséges késleltető elemek



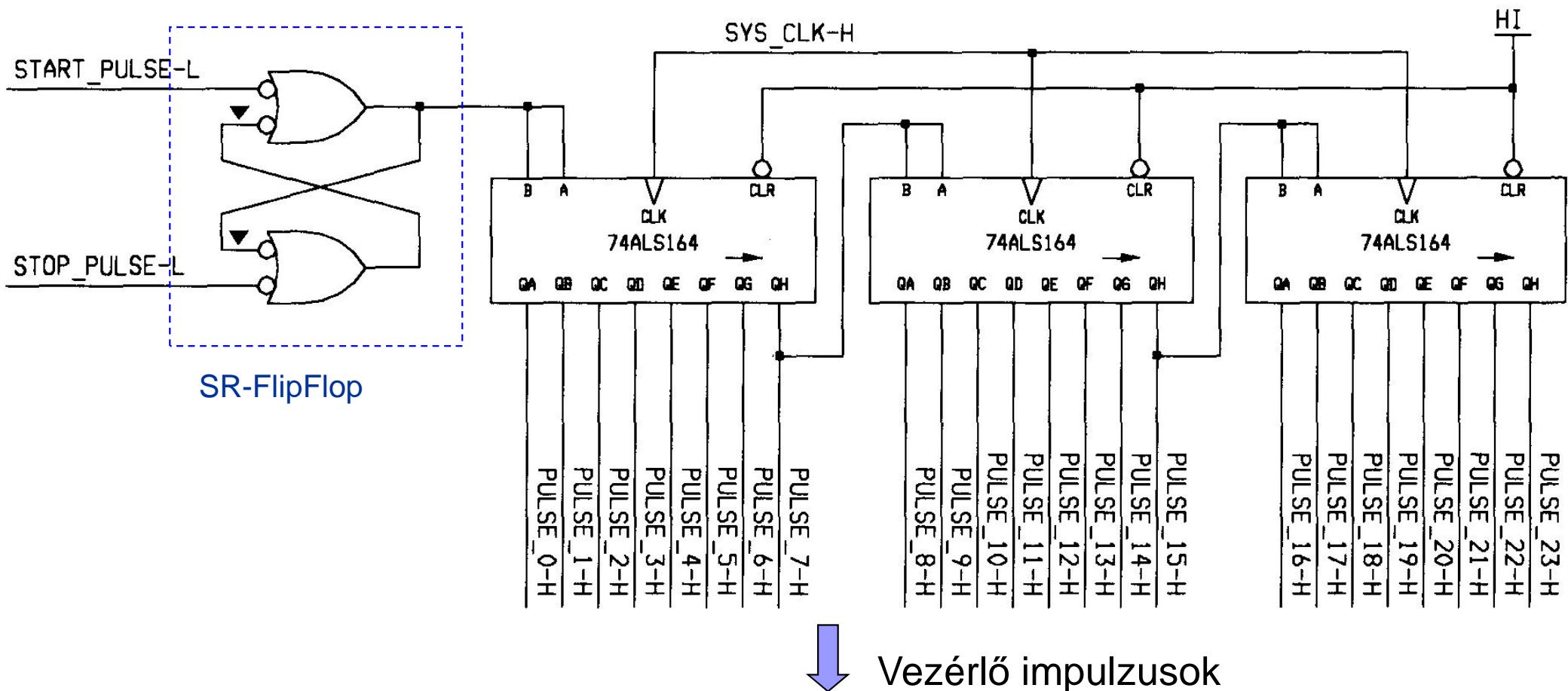


Szekvenciális vezérlő rendszerek tervezése Shift-regiszteres időzítővel

Vezérlő Shift-regiszteres időzítővel

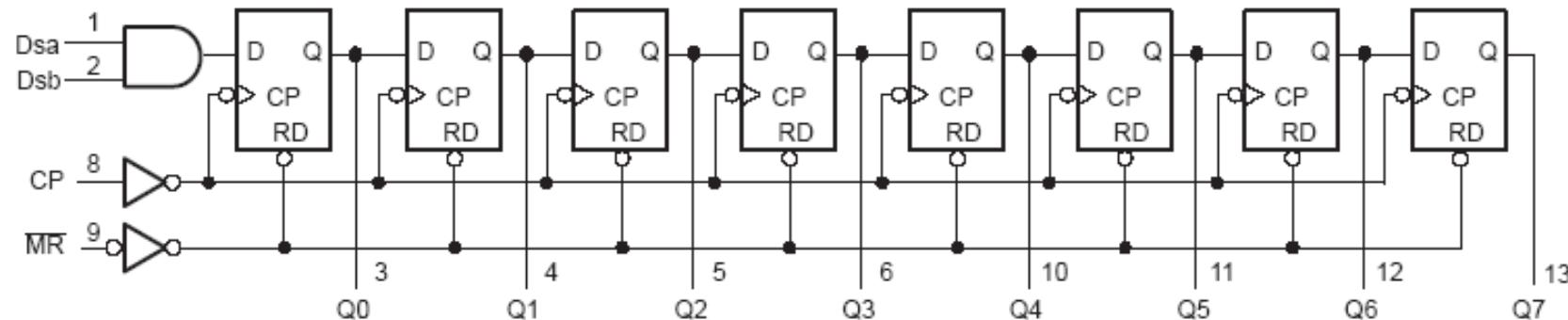
- Ez a megvalósítás az egyedi késleltetéses módszerhez nagyban hasonlít, ugyanis:
 - *Adatút-diagrammot* használunk a vezérlőjelek azonosítására,
 - *Folyamat-diagrammot* a regiszter-transzferek (RTL) ábrázolására, míg
 - *Idő-diagrammot* a vezérlőjelek kölcsönhatásának leírására.
- Hárrom 8-bites Shift-regiszter segítségével generálja az impulzusokat (közvetve a vezérlő jeleket is).

Sorrendi vezérlő egységek megvalósítása Shift-regiszterekkel



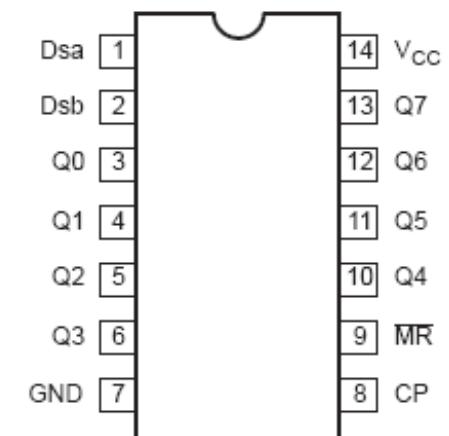
74ALS164

■ 8-bites serial in/paralel out shift regiszter



- Dsa / Dsb: két adatbemenet (egyiket lehet engedélyezőnek is definiálni)
- Qn: adatkimenetek
- CP: clock pulse
- MR: low master-reset

Több 74ALS164-et összekötve szinkron shift reg. kapunk



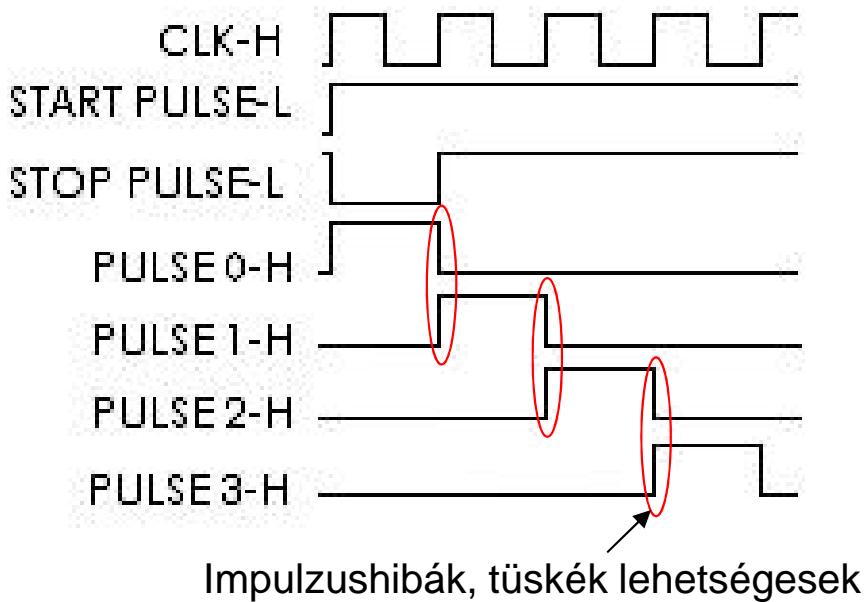
1. módszer: Nem-átlapoló impulzusok

- Inicializáláskor a kívánt impulzus előállítását a START_PULSE_L beállításával érhetjük el, amelyet a teljes folyamat végéig „L” alacsony-aktív szinten tartunk. A következő órajelciklusban a PULSE_0-H jel állítódik be magas jelszintre *rövid* 40 ns-os *impulzus* ideig.
- A STOP_PULSE-L vezérlőjelet a PULSE_0-H jel negálásával kapjuk meg (abban az esetben, ha egy ciklus, azaz 40ns ideig tart). Az órajel minden egyes felfutó élére shiftelődik az impulzus. Ekkor nem lapolódnak át, mivel egymás utáni 40ns –os részekből állnak össze, és a megfelelő PULSE_XX kimeneti vezérlőjelek OR kapcsolatából képződnek.

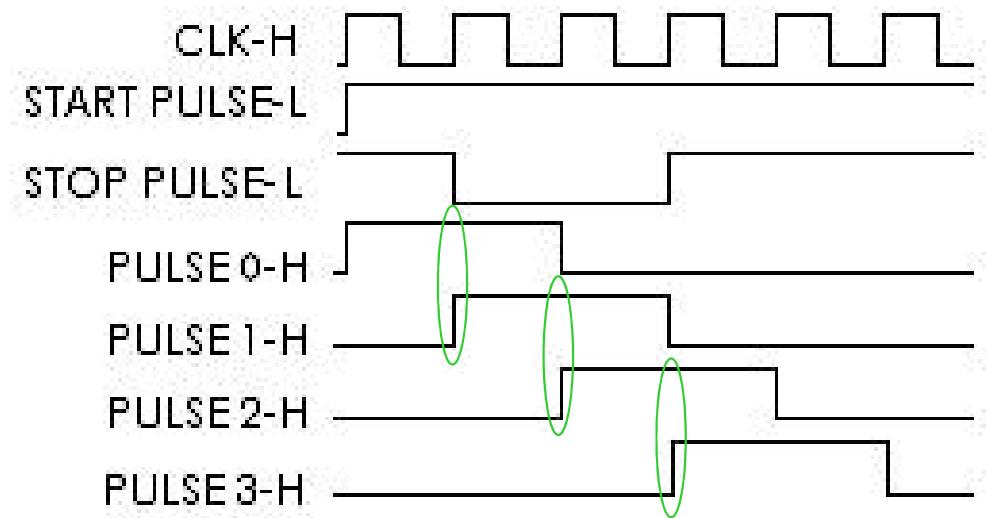
2. módszer: Átlapoló impulzusok

- Bizonyos esetekben azonban nem kívánt impulzushibák ún. tüskék (glitch) keletkezhetnek (pl. ha az egyik jel alacsony szinten marad, a másik viszont magas jelszintre vált). Ezeket a nem megfelelő jelváltásokat vagy SET-RESET flip-flopok használatával küszöbölhetjük ki, vagy pedig alkalmazni kell az **átlapoló impulzusok** technikáját.
- Ezt az idődiagramot a jobboldali ábra mutatja. Két egység hosszú *impulzusokat* (80ns) egyszerűen létrehozhatunk a PULSE_1-H jel és az invertált STOP_PULSE-L jel OR kapcsolatával (a bal oldali ábra jeleiből!). Az így kapott impulzus mentes lesz a hibáktól, és kiküszöbölhetők a hazárdjelenségek.

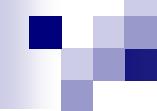
„Nem-átlapoló” és „átlapoló” impulzusok bemutatása idődiagramon



Nemátlapoló rövid impulzusok (40ns)



Átlapoló hosszú impulzusok (80ns)



Mikrokódos vezérlők – reguláris vezérlési struktúrák

Ismétlés: Vezérlő egységek

- Általánosságban: a vezérlő egység feladata a memóriában lévő gépi kódú program utasításainak
 - értelmezése (decode),
 - részműveletekre bontása,
 - és ezek alapján az egyes funkcionális egységek vezérlése (**a vezérlőjelek megfelelő sorrendben történő előállítása**).

Klasszikus vs. reguláris módszer

- Eddig a „klasszikus”, késleltetéses módszereket tárgyaltuk (*huzalozott* és *shift-regiszteres* példákkal). A rendszer tervezésekor, miután a feladat elvégzéséhez szükséges vezérlőjeleket definiáltuk, meg kell határozni a kiválasztásuk sorrendjét, és egyéb specifikus információkat (rendszer ismeret, tervezési technikák, viselkedési leírások – pl.VHDL). Vezérlő egység:
 - Kombinációs hálózat (hard-wired = huzalozott), vagy
 - FSM: véges állapotú automata alapú.
- Wilkes (1951): A komplex, többcímű (operandusú), illetve vezérlési szerkezeteket „**reguláris módszerrel**” lehet gyorsítani, egyszerűsíteni: nevezetesen gyors memória elemeket kell használni az utasítássorozatok tárolásánál. Ugyan a klasszikus módszernél használt állapotgépekkel (FSM) modellezik a reguláris vezérlő egység működését, majd ezt a modellt transzformálják át mikrokódos memóriát (ami nem azonos az operatív memóriával!) használva. Az adatútvonal vezérlési pontjait memóriából (ROM) kiolvasott *vertikális-* vagy *horizontális-mikrokódú* utasításokkal állítják be!

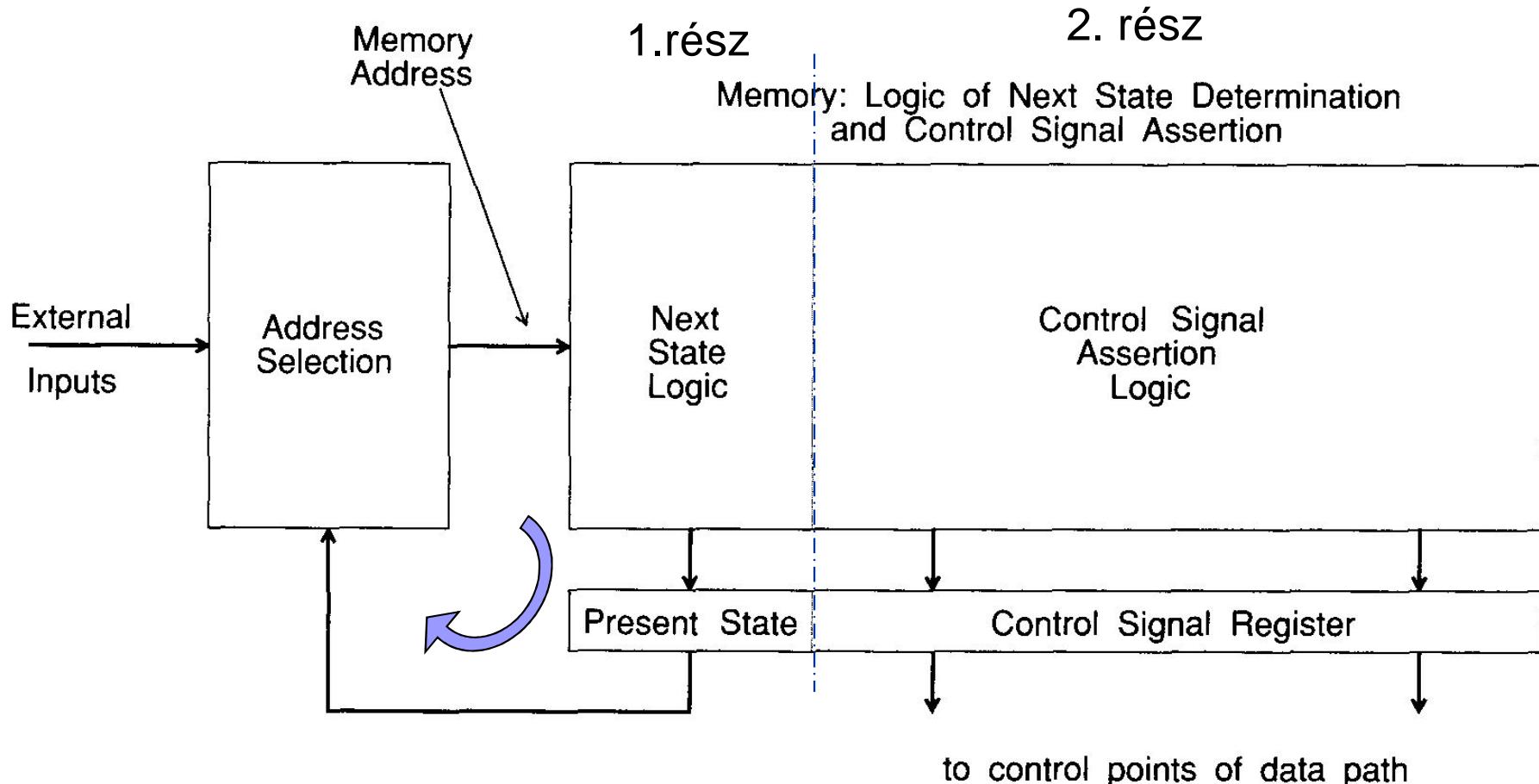
Reguláris módszer: mikrokódos vezérlés tulajdonságai

- **Mikrokód:** gépi kódú utasításokat (IR) legalacsonyabb szintű áramköri (hw) utasítások sorozatára leképező köztes kód leképezés 
- Szerepe: **értelmezés** (interpreter / translator) a fenti két szint között:
 - A gépi kódú utasítások változtatásának lehetősége (RISC, CISC), anélkül hogy a HW változna
 - Mai rendszerek olvasható mikrokódját gyors memóriában (általában ROM), vagy PLD-ben tárolják (írható esetben RAM, vagy Flash is lehet)
- Alkalmazás: CPU, GPU, lemezvezérlők, NPU (network processor unit)

Mikroprogram = mikroutasítások sorozata

- Példa: Tipikus mikroprogram (~RTL-szintű utasítás szekvenciák sorozata):
 1. Load Reg[1] to the "A" side of the ALU
 2. Load Reg[7] to the "B" side of the ALU
 3. Select the ALU to perform 2's-comp addition
 4. Select the ALU's carry input to zero
 5. Store the result value in Reg[8]
 6. Update the "condition codes" with the ALU status flags ("Negative", "Zero", "Overflow", and "Carry")
 7. Microjump (MicroPC) for the next microinstruction

FSM megvalósítása Memóriával

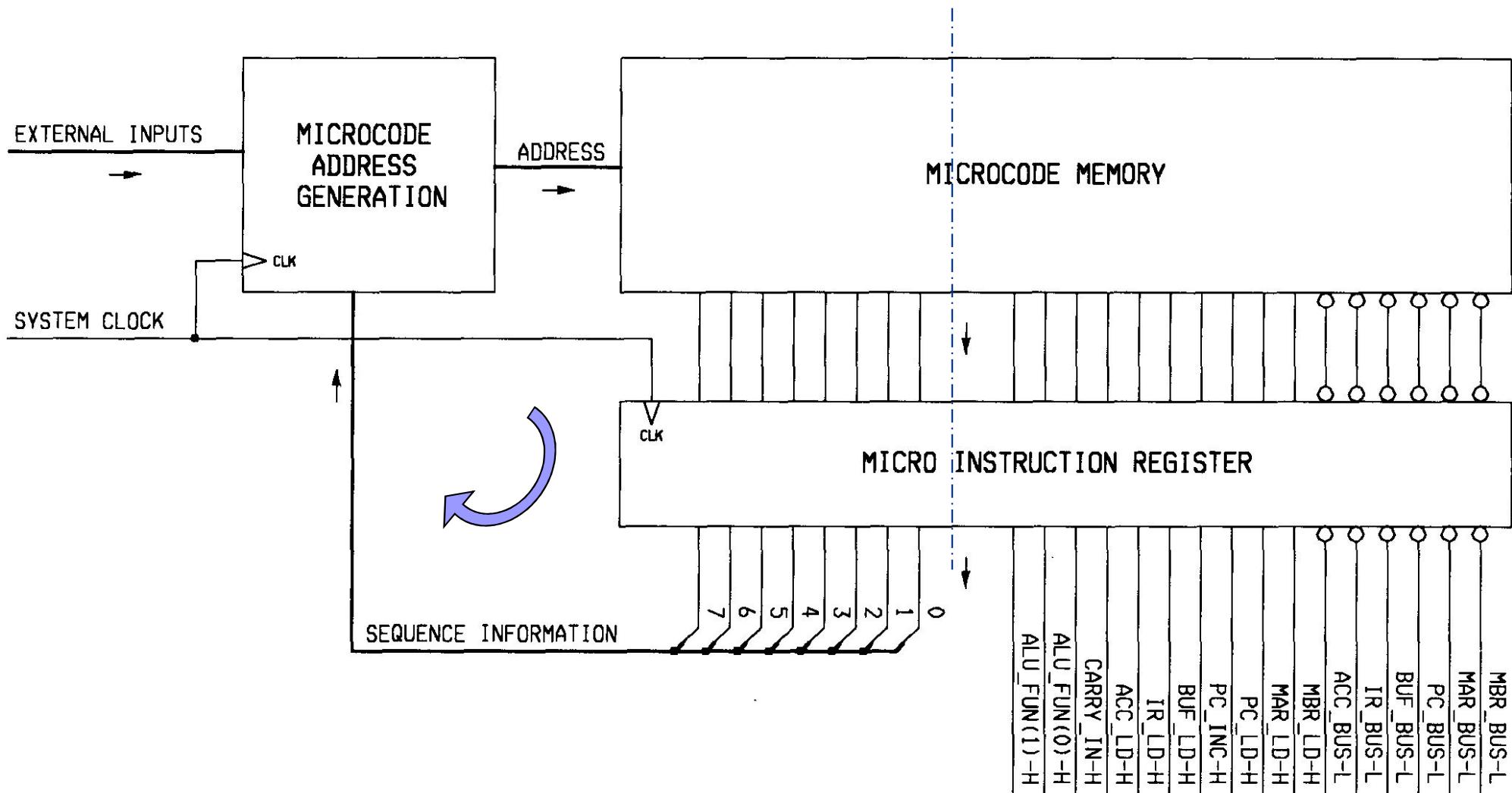


- 1.rész: szabályozza az eszköz működését a megfelelő állapotok sorrendjében
- 2.rész: szabályozza az adatfolyamot a megfelelő vezérlőjelek beállításával (assertion) az adatúton (vezérlési pontokon)

FSM megvalósítása Memóriával (folyt)

- **Address Selection:** (mint új elem) a következő utasítás (*Next State*), és beállítani kívánt vezérlőjel (*control signal assertion*) címére mutat a memóriában (~ Id. MAR).
- A memória címet (**memory address**-t) külső bemenő jelek és a *present state* együttesen határozzák meg. E cím segítségével megkapjuk az adott vezérlő információ pontos helyét a memóriában, ill. ez az információ, mint új állapot betöltődik a generált vezérlőjeleket tároló (*Control Signal Register*).
- **Next-State** kiválasztásához szükséges logikai memória méretét az aktuális állapotok száma, az állapotdiagram komplexitása, és a bemenetek száma határozza meg.
- **Control Signal** generálásához szükséges logikai memória méretét a bemenetek száma, a függvény (vezérlő jel) komplexitása, és a vezérlőjelek száma határozza meg.

Általános Mikrokódos vezérlő



Általános Mikrokódos vezérlő felépítése

- ***Micro Instruction Register:*** a „Present State” (aktuális állapot) regisztert + a Control Signal regisztert egybeolvasztja (az adatút vezérlővonalainak beállítása / kiválasztása). Mikroutasítások sorrendjében generálódik a vezérlőjel!
- ***Microcode Memory:*** a Control Signal Assertion Logic vezérlőjel generálás/beállítás + „Next-State” kiválasztása (mikroprogram eltárolása) összevonása
- ***Microcode Address Generator:*** a vezérlő jelet az aktuális mikroutasítások lépéseként sorban generálja, de címkiválasztási folyamat komplex. **Sebesség a komplexitás rovására változhat!** (komplexebb vezérlési funkciót alacsonyabb sebességgel képes csak generálni). A következő cím kiválasztása még az aktuálisan futó mikroutasítás végrehajtása alatt végbemegy! Számlálóként működik: egyik címről a másik címre inkrementálódik (mivel a mikroutasításokat tekintve szekvenciális rendszerről van szó). Kezdetben resetelni kell.

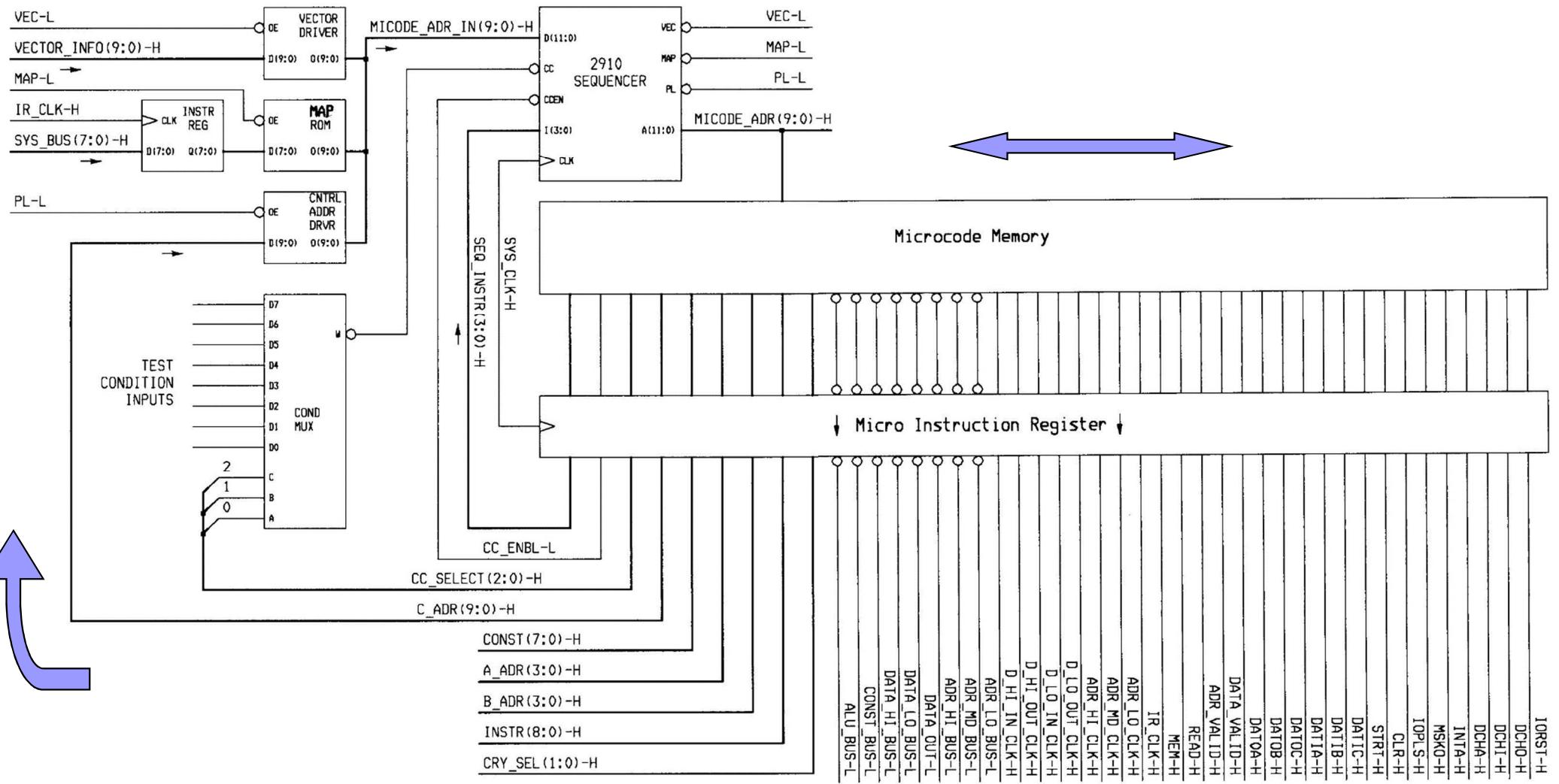
Általános Mikrokódos vezérlők tulajdonságai

- Egy gépi ciklus alatt egy **mikroprogram** fut le (amely mikroutasítások sorozatából áll). A műveleti kód (utasítás opcode része) a végrehajtandó mikroprogramot jelöli ki. A mikrokódú memória általában csak statikus módon olvasható gyárilag konfigurált ROM, ha írható is, akkor dinamikus mikroprogramozásról beszélünk.
- Ha a mikroprogram utasításai szigorúan **szekvenciálisan** futnak le, akkor a címüket egy egyszerű számláló inkrementálásával megkaphatjuk. Memóriából érkező bitek egyik része a következő **cím kiválasztását** (Sequence Information), míg a fennmaradó bitek az **adatáramlást** biztosítják.
- Mai gyors félvezető alapú memóriáknak köszönhetően kis mértékben lassabb, mint a huzalozott vezérlő egységek, mivel ekkor a memória elérési idejével (~ns) is számolni kell (nem csak a visszacsatolt aktuális állapot késleltetésével.)⁵⁸

1.) Horizontális mikrokódos vezérlő

- Mindenegyes vezérlőjelhez saját vonalat rendelünk, ezáltal horizontálisan megnő a mikro- utasításregiszter kimeneteinek száma, (**horizontálisan** megnő a mikrokód). Minél több funkciót valósítunk meg a vezérlőjelekkel, annál **szélesebb** lesz a mikrokód.
- Ennek köszönhetően ez a **leggyorsabb** mikrokódos technika, mivel minden bit független egymástól ill. egy mikrokóddal többszörös (konkurens) utasítás is megadható.
 - Pl: a megfelelő funkcionális egységeket (memória, ALU, regiszterek stb.) egyszerre tudjuk az órajellel aktiválni, ezáltal egy órajelciklus alatt az információ minden irányba átvihető. Növekszik a sebesség, mivel nincs szükség a vezérlőjelek dekódolását végző dekódoló logikára. Így minimálisra csökken a műveletek ciklusideje.
- Azonban nagyobb az **erőforrás** szükségléte, **fogyasztása**.

Horizontális mikrokódos vezérlő



2.) Vertikális mikrokódos vezérlő

- Nem a sebességen van a hangsúly, hanem hogy **takarékoskodjon** az erőforrásokkal (fogyasztás, mikrokódban a bitek számával), ezért is **lassabb**.
- Egyszerre csak a szükséges (korlátozott számú) biteket kezeljük, egymástól nem teljesen függetlenül, mivel közük egyszerre csak az egyiket állítjuk be (deködöljük). A jeleket ezután **deködolni** kell (több időt vesz igénybe). A kiválasztott biteket megpróbáljuk minimális számú vonalon keresztül továbbítani.
- A műveletek párhuzamos (konkurens) végrehajtása korlátozott. Dekódolás: $\log_2(N)$ számú dekódolandó bit -> N bites kimeneti busz. Több mikroutasítás szükségtetik \Rightarrow így a mikrokódú memóriát „**vertikálisan**” meg kell növeli.

Vertikális mikrokódos vezérlő

