



# Számítógép Architektúrák (MIKNB113A)

5. előadás: Szekvenciális Hálózatok  
Vezérlő egységek I.: klasszikus és reguláris  
módszerek

Előadó: Dr. Vörösházi Zsolt  
[voroshazi.zsolt@virt.uni-pannon.hu](mailto:voroshazi.zsolt@virt.uni-pannon.hu)

# Jegyzetek, segédanyagok:

- Könyvfejezetek:

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

- Oktatás → Tantárgyak → Számítógép  
Architektúrák

- ([chapter05.pdf](#))

- Fóliák, óravázlatok .ppt (.pdf)

- Feltöltésük folyamatosan

# Vezérlő egységek általánosan

- A számítógép vezérlési funkcióit ellátó *szekvenciális* egység. Cél: vezérlő jelek generálása ütemezett módon.
- **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 tervezési 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 általános 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. korábban a RISC architektúrák):
  - Mealy-modell,
  - Moore-modell (Példa: FIR szűrő tervezése – FSM állapotgép segítségével).
    - Multiplexeres / késleltetési / Shift-regiszteres megvalósítások
- **II. Mikroprogramozott („reguláris” vezérlési szerkezettel** – 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 (<- nem tárgyaljuk)

# Ismétlés: Kombinációs hálózatok

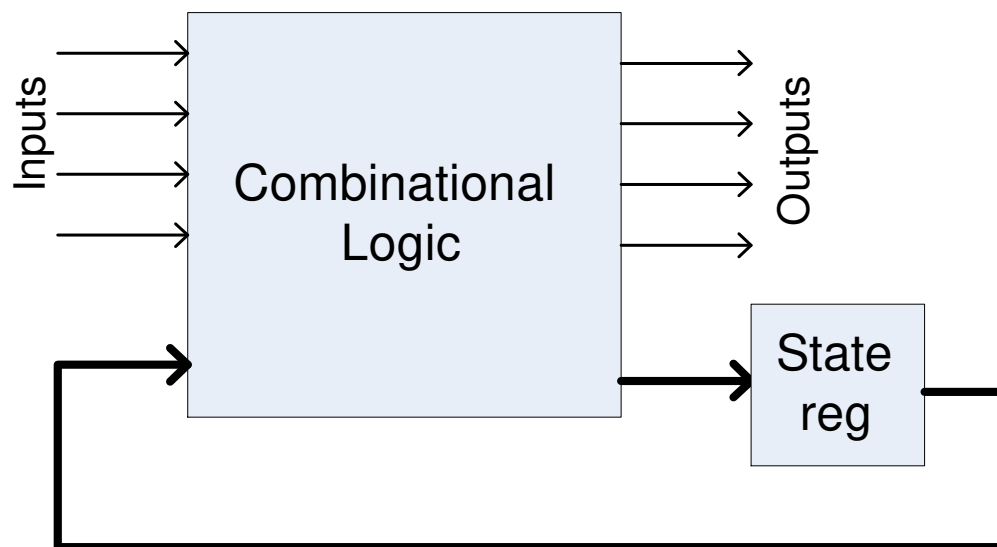
- **(K.H.) Kombinációs logikai hálózatról** beszélünk: ha a mindenkori kimeneti kombinációk értéke csupán a bemeneti kombinációk pillanatnyi értékétől függ (tároló „kapacitás”, vagy memória nélküli hálózatok).



# Sorrendi hálózatok:

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

Vezérlő  
egységek  
alapjául szolgáló  
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 áramkörü elemek (pl. ALU, memória elemek) összehangolt működésének biztosítása.
- Az időzítő-vezérlő áramkörök ***szekvenciális rendszerek*** – mivel az áramkörü egységek tevékenységének egymáshoz viszonyított *időbeli sorrendiségét* biztosítják – melyek az aktuális *kimenet* értékét a *bemenet*, és az *állapotok* függvényében határozzák meg.

# Az időzítő-vezérlő lehet:

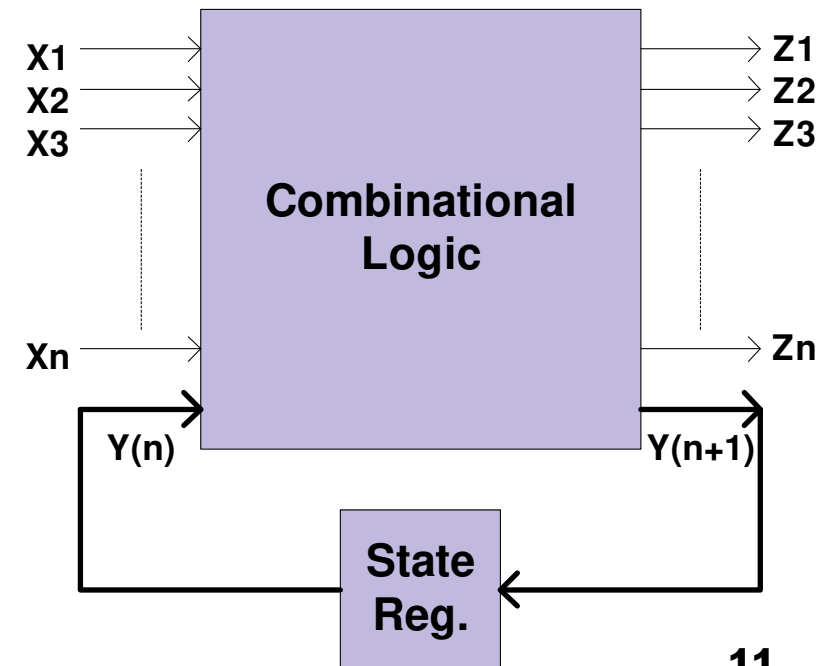
- **huzalozott:** áramkörökkel, *dedikált* összeköttetésekkel fizikailag megvalósított (Mealy, Moore modellek alapján, illetve programozható PLD-k), vagy
  - Pl: korai RISC architektúrák
- **mikroprogramozott:** az adatútvonal (data-path) vezérlési pontjait Memóriából (ROM) kiolvasott *vertikális-* vagy *horizontális-*mikrokódú utasításokkal állítják be.
  - Pl: mai CISC, RISC architektúrák



# I. Klasszikus vezérlési módszerek: Huzalozott vezérlő egységek

# 1.) Mealy-modell

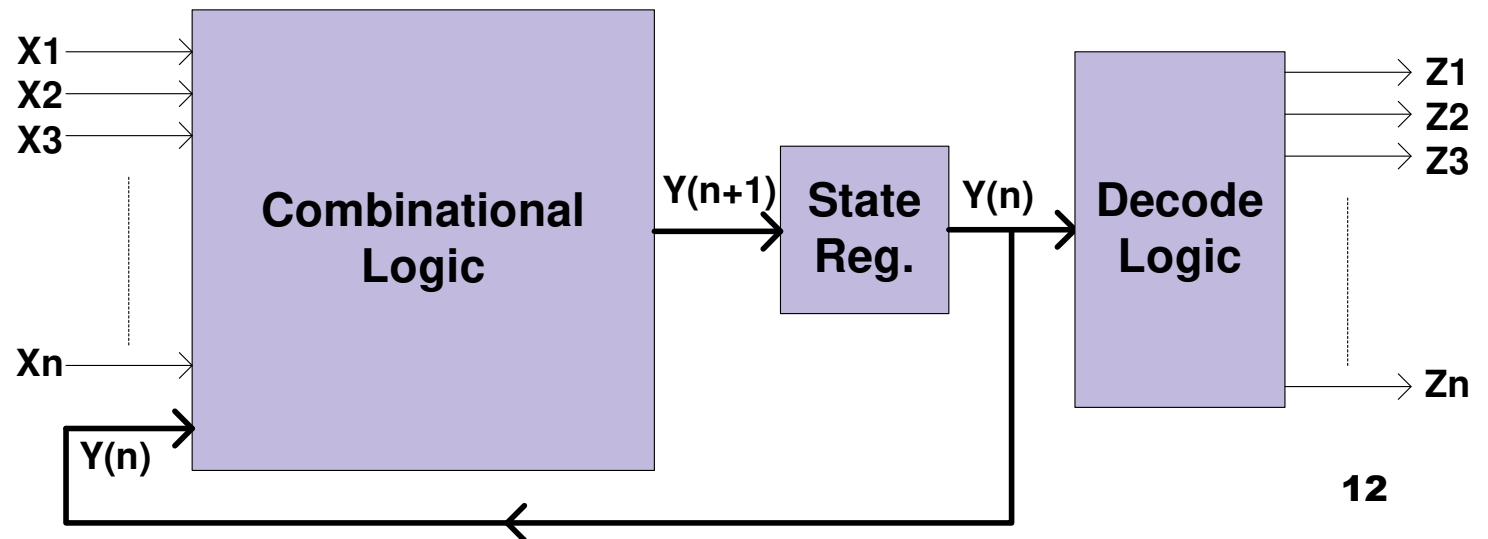
- A *sorrendi hálózatok* egyik alapmodellje. Késleltetés: a kimeneten az eredmény véges időn belül jelenik meg! Korábbi értékek visszacsatolódnak a bemenetre: kimenetek nemcsak a bemenetek pillanatnyi értékétől, hanem a korábbi állapotoktól is függenek. Problémák merülhetnek fel az állapotok és bemenetek közötti szinkronizáció hiánya miatt (változó hosszúságú kimenetet - dekódolás). Ezért alkalmazzuk legtöbbször 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 függenek (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**





# Szekvenciális vezérlő rendszerek – egyedi késleltetési módszer

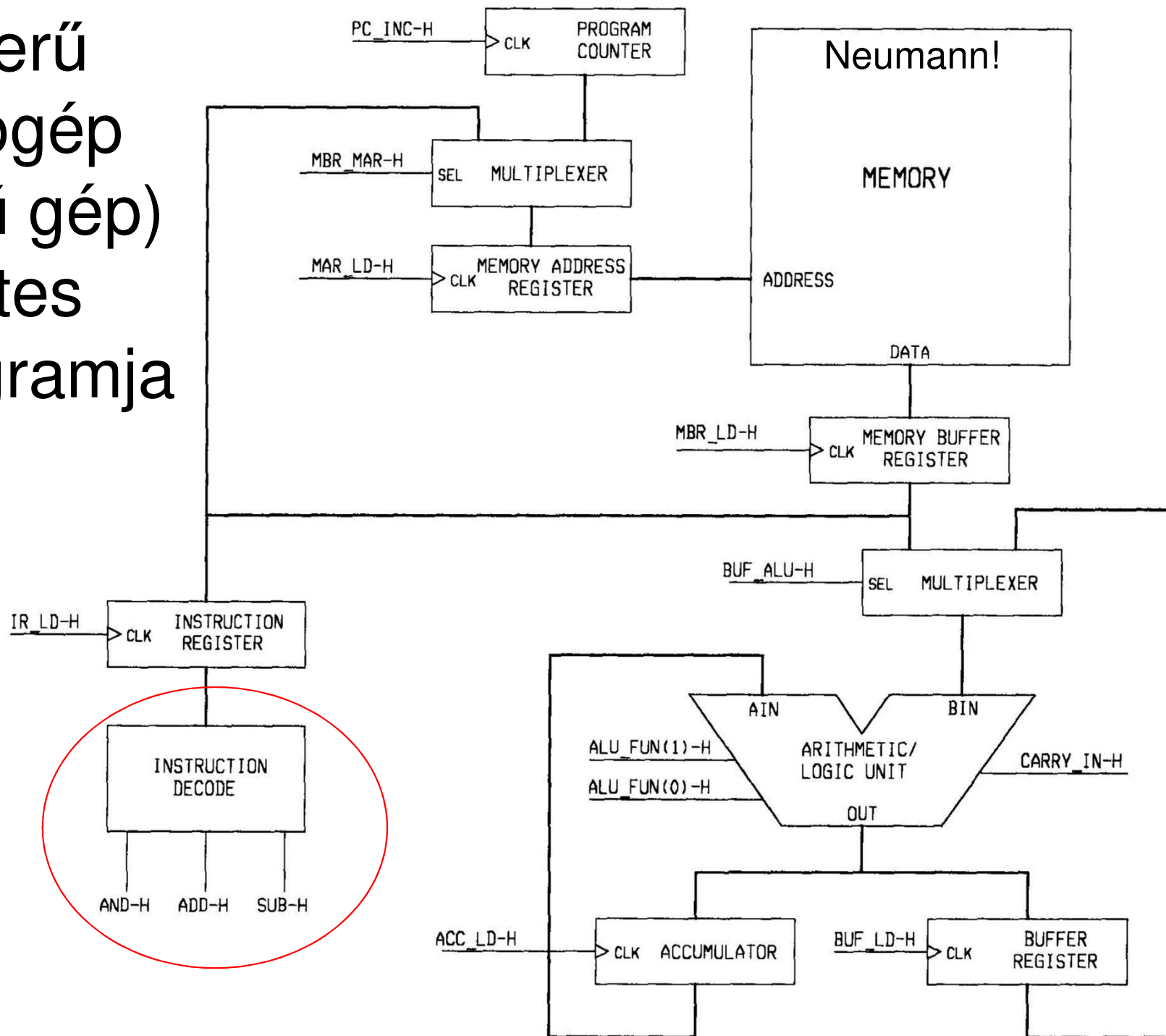
# Szekvenciális vezérlő rendszerek

- Ahelyett, hogy egyetlen regiszter tömbben tárolnánk a rendszer állapotait, ebben az esetben *külön regisztereket* definiálunk egy egyszerű számítógép (egycímű gép) blokkdiagramját felhasználva.
- Regiszter-transzfer műveletek sorozatán mutatjuk be ennek a *késleltetéses* vezérlő rendszernek a működését.

# Egyszerű számítógép (egycímű gép) részletes blokkdiagramja

Példa: 3  
egyszerű utasítás

- ADD
- SUB
- AND



# Példa: Egycímű gép vezérlési funkciója

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

## Késleltetések!

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

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

| ALU_FUN |   | OUT function                |
|---------|---|-----------------------------|
| 0       | 0 | bitenkénti AND (A_In, B_In) |
| 0       | 1 | bitenkénti OR (A_In, B_In)  |
| 1       | 0 | inverz NOT (B_In)           |
| 1       | 1 | bináris ADD (A_In, B_In)    |

[40 ns]

[40 ns]

[40 ns]

[80 ns]

# Három utasítás RTL leírása

**Table 5.2.** Register Transfers for Three Instructions.

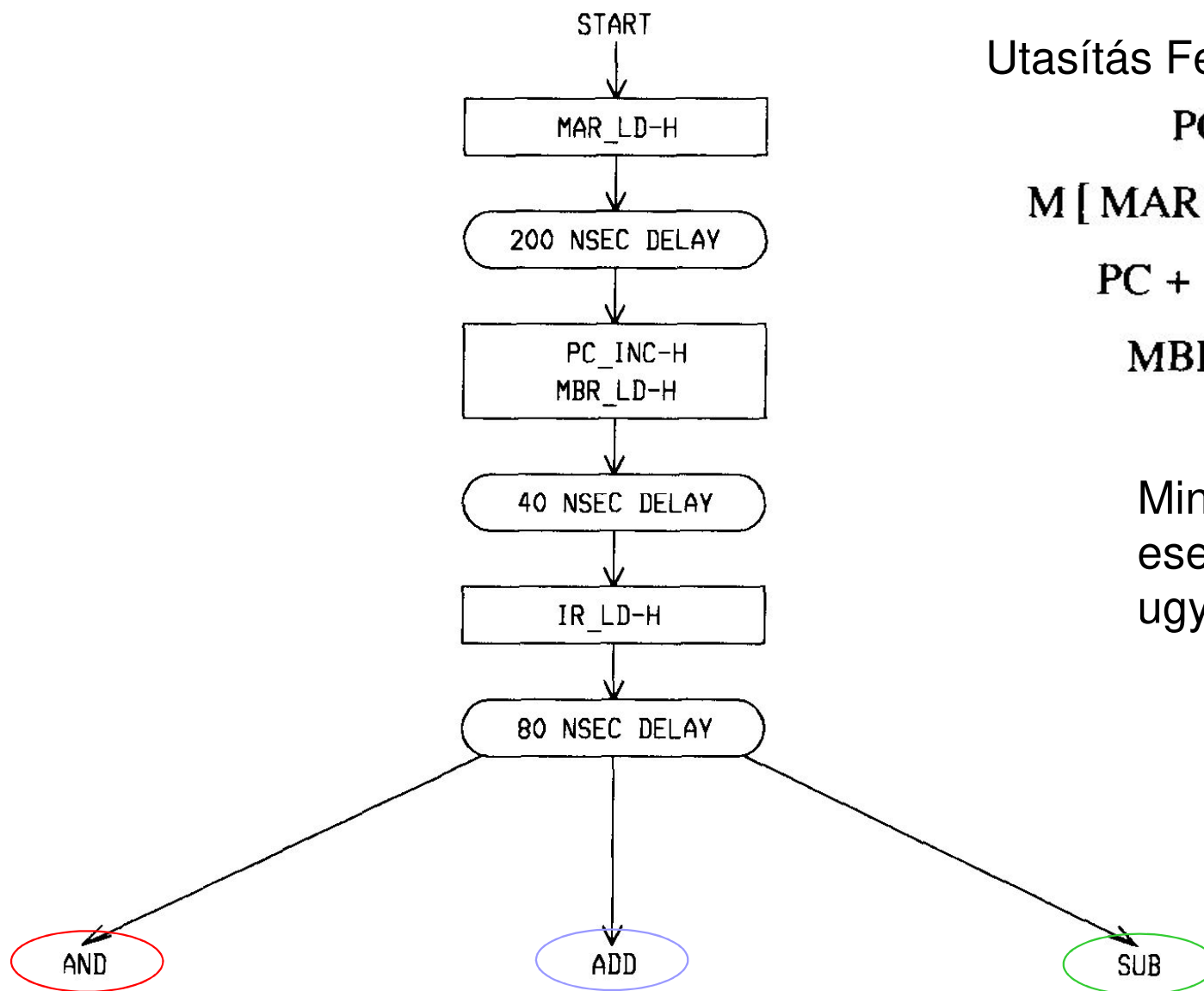
*Register Transfers for Example*

| <i>AND Instruction</i>                          | <i>ADD Instruction</i>                          | <i>SUBTRACT Instruction</i>                                   |
|-------------------------------------------------|-------------------------------------------------|---------------------------------------------------------------|
| Fetch: $PC \rightarrow MAR$                     | $PC \rightarrow MAR$                            | $PC \rightarrow MAR$                                          |
| $M[MAR] \rightarrow MBR$                        | $M[MAR] \rightarrow MBR$                        | $M[MAR] \rightarrow MBR$                                      |
| $PC + 1 \rightarrow PC$<br>$MBR \rightarrow IR$ | $PC + 1 \rightarrow PC$<br>$MBR \rightarrow IR$ | $PC + 1 \rightarrow PC$<br>$MBR \rightarrow IR$               |
| Execute: $MBR \rightarrow MAR$                  | $MBR \rightarrow MAR$                           | $MBR \rightarrow MAR$                                         |
| $M[MAR] \rightarrow MBR$                        | $M[MAR] \rightarrow MBR$                        | $M[MAR] \rightarrow MBR$                                      |
| $MBR \bullet ACC \rightarrow ACC$               | $MBR + ACC \rightarrow ACC$                     | $\neg MBR \rightarrow BUF$<br>$BUF + ACC + 1 \rightarrow ACC$ |
|                                                 |                                                 | <div>2's</div> <div>Comp</div>                                |

# Tervezői feladatok:

- Eddig rendelkezésre áll:
  - RTL leírás,
  - Vezérlési funkciók,
  - Blokk diagram (egycímű gép)
- Most: a vezérlési rész tervezése következhet
  - Vezérlő jelek gondos beállítása (assertion levels):
    - jelnév\_L, vagy jelnév\_H meghatározása

# Három utasítás késleltetési folyamat ábrája (delay flowchart)



Utasítás Fetch:

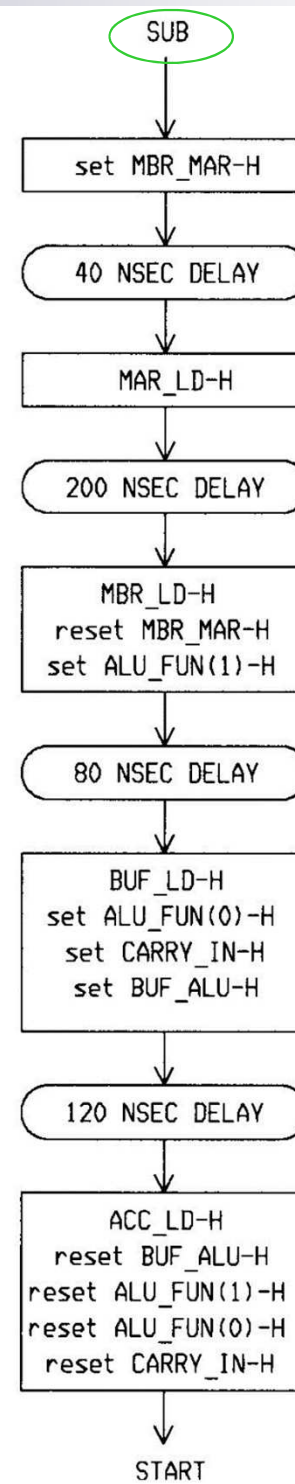
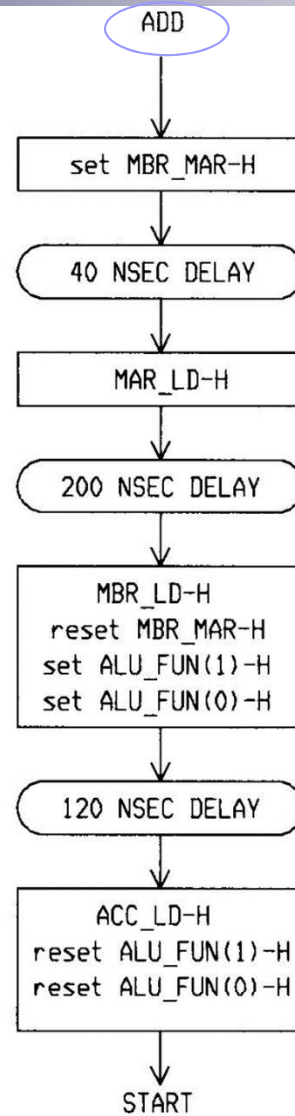
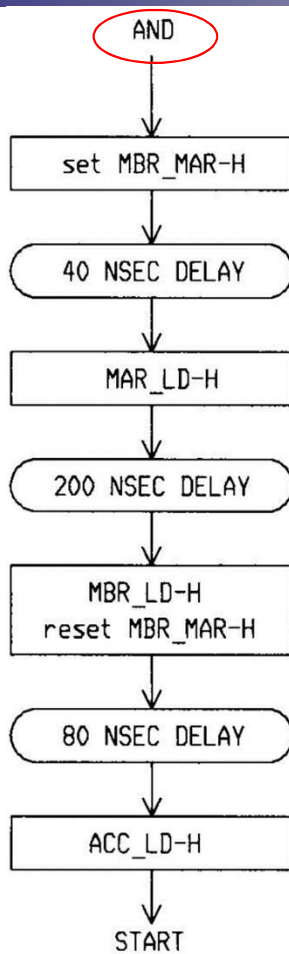
$PC \rightarrow MAR$

$M[MAR] \rightarrow MBR$

$PC + 1 \rightarrow PC$

$MBR \rightarrow IR$

Mindhárom  
esetben  
ugyanaz.



(folyt.)

2's  
Complement



## II. Mikrokódos vezérlők – reguláris vezérlési struktúrák

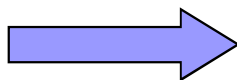
# Ismétlés: Vezérlő egységek

- 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 előállítás).**

# Klasszikus vs. reguláris módszer

- Eddig a „klasszikus”, késleltetési 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!*

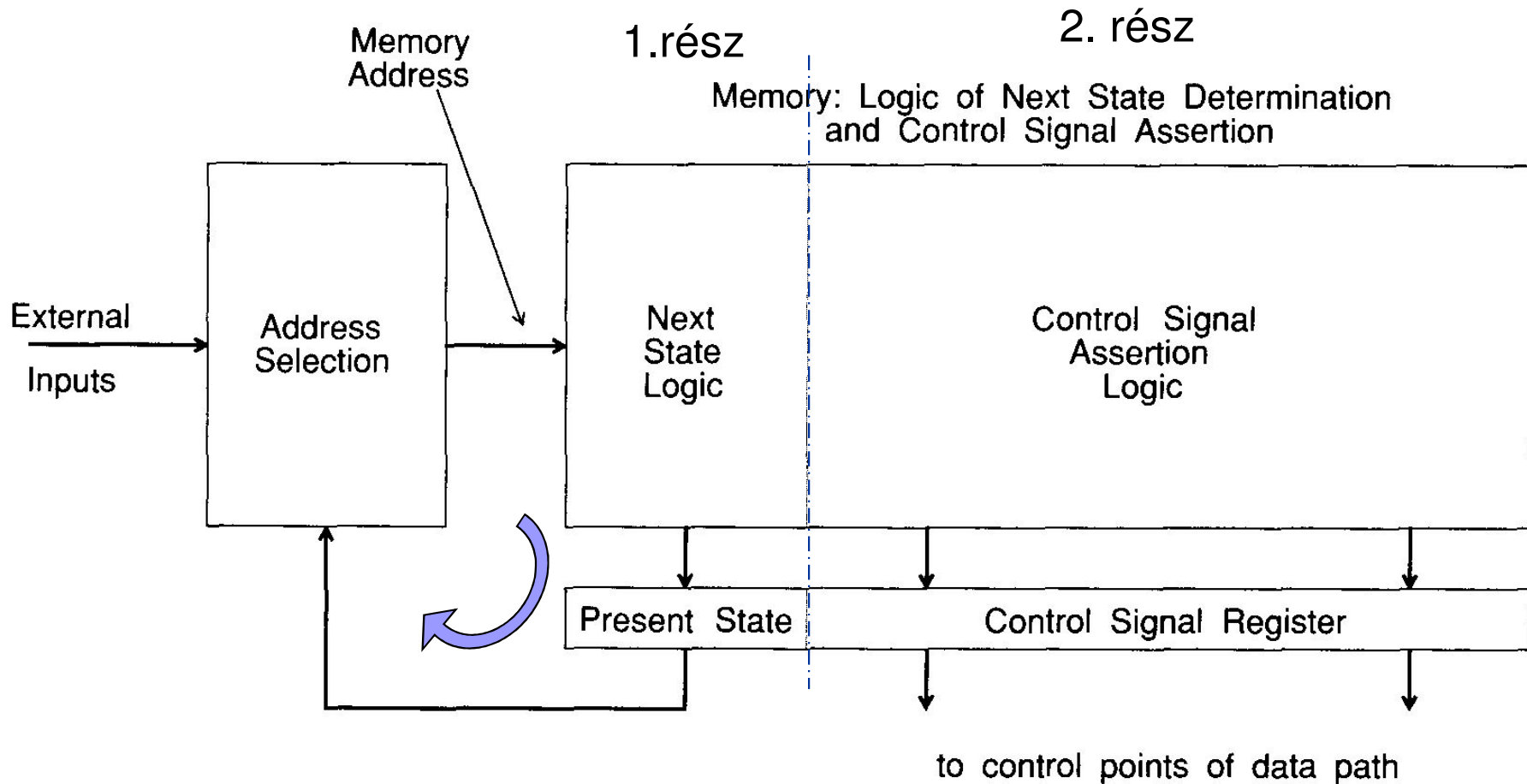
# Reguláris módszer: mikrokódos vezérlés tulajdonságai

- Mikrokód: *gépi kódú utasításokat* (IR)  <sup>leképezés</sup> *legalacsonyabb szintű áramköri (hw) utasítások* sorozatára leképező köztes kód
- 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), stb.

# 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 '1'
  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

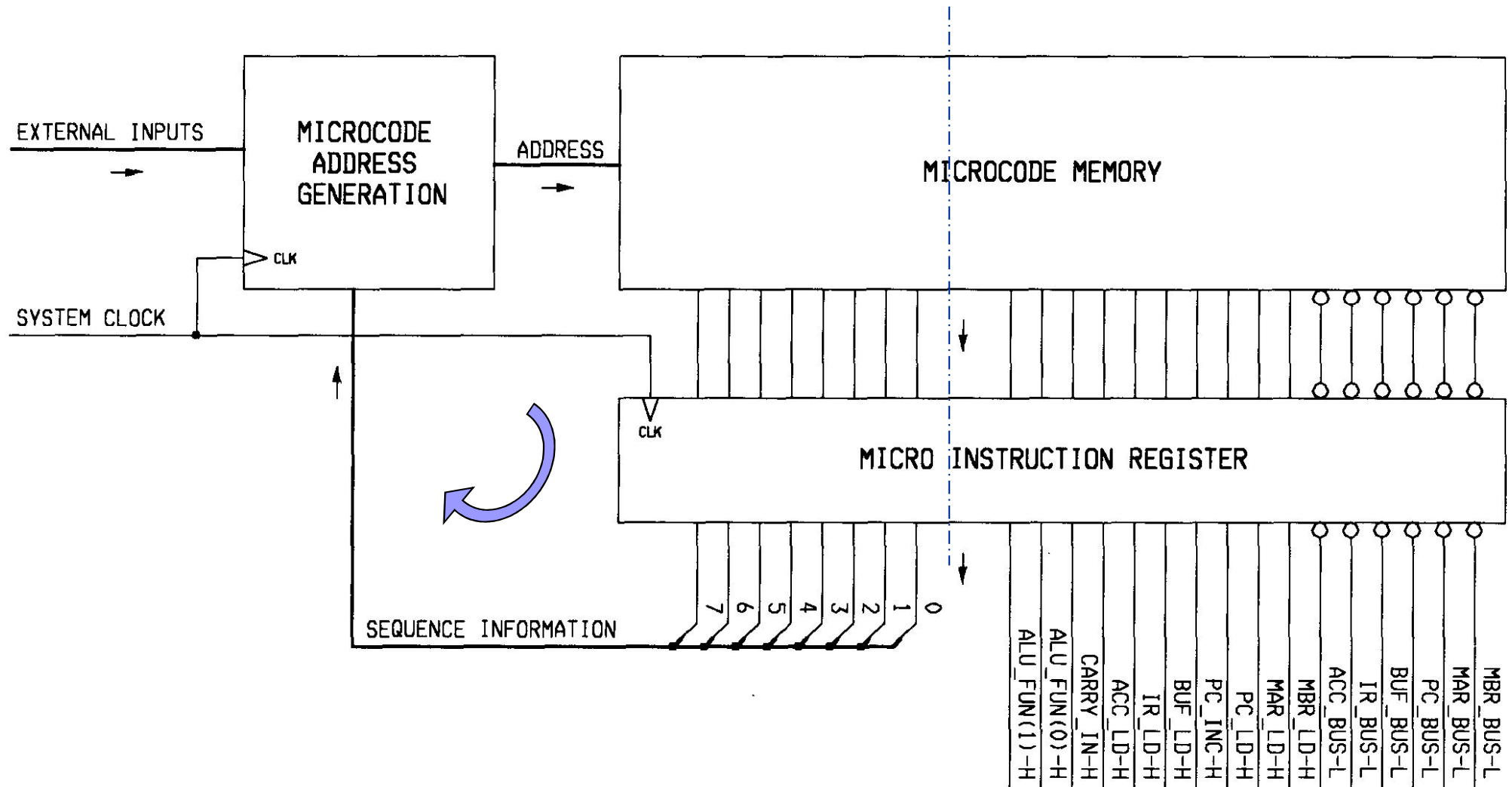


- 1.rész: szabályozza az eszköz működését a megfelelő állapotok *sorrendjében*
- 2.rész: szabályozza az adatfolyamot a megfelelő *vezérlőjelek beállításával* (assertion) az adatúton (vezérlési pontokon)

# 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 (~ ld. 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 lépéseiként 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.

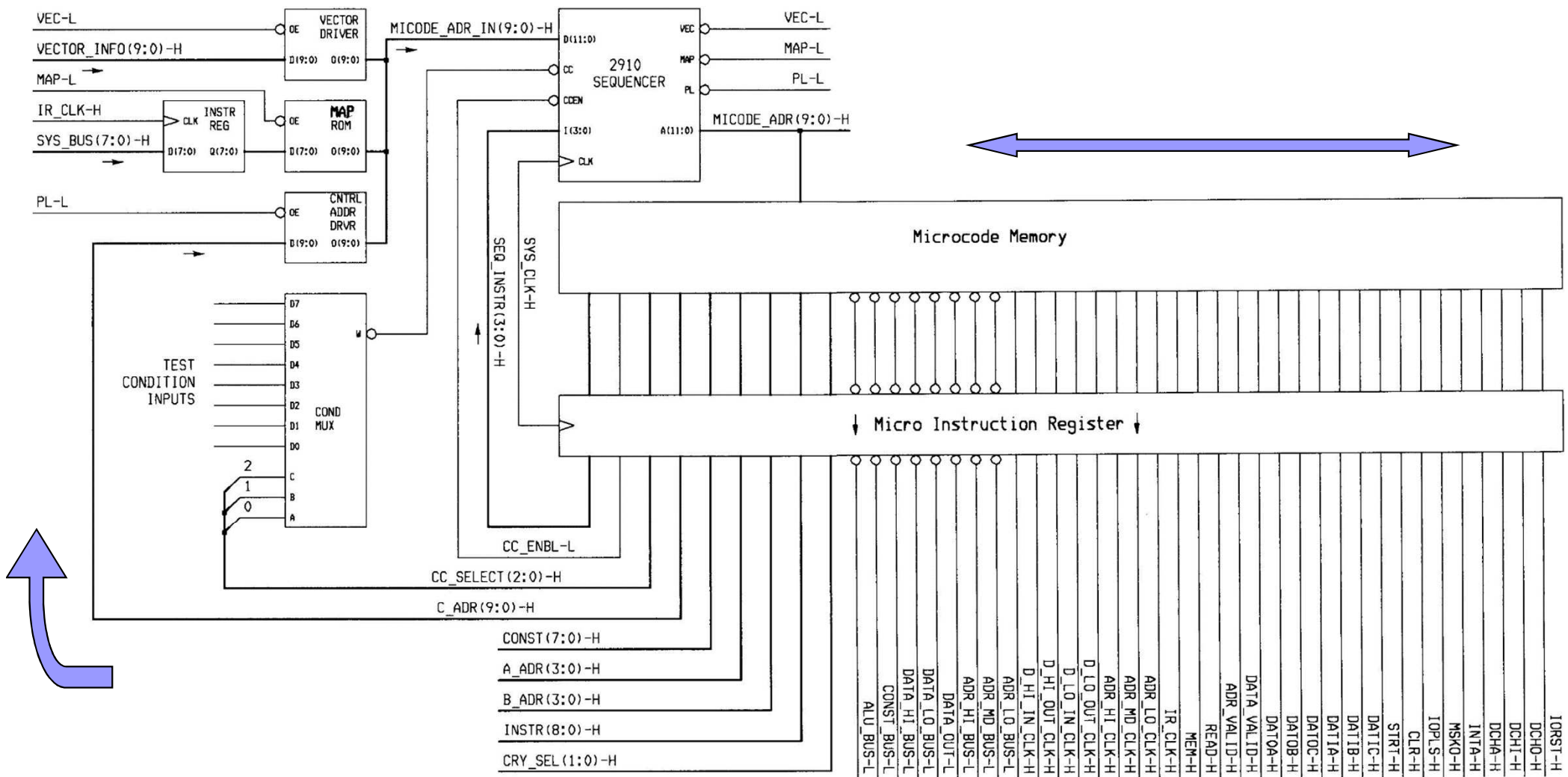
# „Általános” mikrokódos vezérlők tulajdonságai

- 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 ( $\sim ns$ ) is számolni kell (nem csak a visszacsatolt aktuális állapot késleltetésével.)<sup>30</sup>

# 1.) Horizontális mikrokódos vezérlő

- Mindenegyres 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 mindkét irányba átvihető. Növekszik a sebesség, mivel nincs szükség a vezérlőjelek dekódolását végző dekódoló logikára. Így minimálisra csökken a műveletek ciklusideje.
- Azonban nagyobb az **erőforrás** szükséglete, **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 közülük egyszerre csak az egyiket állítjuk be (dekódoljuk). A jeleket ezután **dekódolni** 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  $\rightarrow$  N bites kimeneti busz. Több mikroutasítás szükségeltetik  $\Rightarrow$  így a mikrokódú memóriát „**vertikálisan**” meg kell növelni.

# „Vertikális” mikrokódos vezérlő

